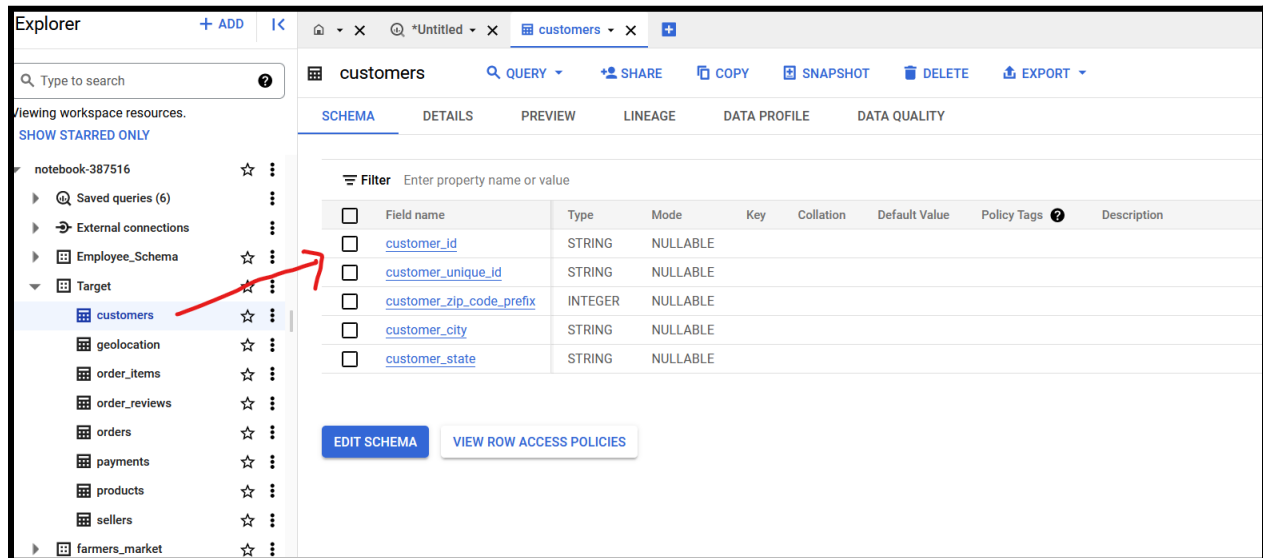


### Q.1.a.answer

In order to check the data type of any table using BigQuery is, Double-click on the name of the table which is mentioned in the left side under the dataset 'target'. And on the right side we can check data types of all columns.



### Q.1.b.answer

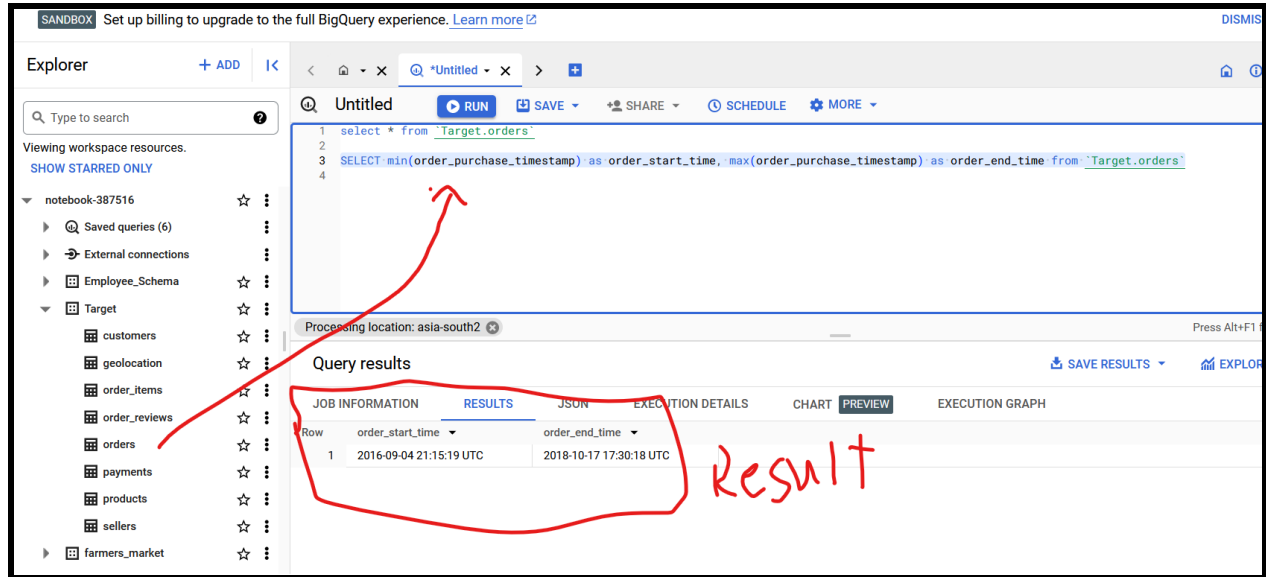
#### Query

```
SELECT min(order_purchase_timestamp) as order_start_time,  
max(order_purchase_timestamp) as order_end_time from `Target.orders`
```

#### Explanation

To find the time range, we will look for the table name 'order' in which we already have 'order\_purchase\_timestamp' col - time range here means the day and time when the first order was placed and the last day when the order was placed. Using the Avg function Max and Min will give us the time range - the ideal option when the data type is in the format of date/time.

**Screenshot of the result -**



### Q.1.c.answer

#### Query -

select count(distinct c.customer\_city) as city, count(distinct c.customer\_state) as state,  
count(distinct o.order\_id) as total\_order,

min(o.order\_purchase\_timestamp) as order\_start\_time, max(o.order\_purchase\_timestamp) as  
order\_end\_time

from Target.customers c left join Target.orders o on c.customer\_id=o.customer\_id

#### Explanation

Here, we have found out all orders placed from total cities and states, in the given time range, which we have solved in question 2 - in this query, we have used aggregate functions like count, min, max, and left join because we have to find out all orders placed by customers.

## Screenshot of the result -

The screenshot shows a data analytics interface with a sidebar on the left containing a search bar and a list of workspace resources. The main area displays a SQL query in a text editor, which is then executed to produce a table of results. The query filters for orders from 'Target' and calculates various metrics. The results table has columns for city, state, total\_order, order\_start\_time, and order\_end\_time.

```
SELECT min(order_purchase_timestamp) as order_start_time, max(order_purchase_timestamp) as order_end_time from `Target.orders`  
#Count the Cities & States of customers who ordered during the given period.  
  
select count(distinct c.customer_city) as city, count(distinct c.customer_state) as state, count(distinct o.order_id) as total_order,  
min(o.order_purchase_timestamp) as order_start_time, max(o.order_purchase_timestamp) as order_end_time  
from Target.customers c left join Target.orders o on c.customer_id=o.customer_id
```

Row	city	state	total_order	order_start_time	order_end_time
1	4119	27	99441	2016-09-04 21:15:19 UTC	2018-10-17 17:30:18 UTC

## Q.2.a.answer

### Query

```
select Extract(year from order_purchase_timestamp) as years_count, count(order_id) as  
order_count from Target.orders  
group by years_count order by years_count;
```

### Explanation

The growing trend here means that with time, there is an increase in total orders. Now that we know that we have to show total orders, now we focus on time, which is in question is in years - means (2016, 2017, 2018,.....n).

## Screenshot of results

The screenshot shows a data analytics interface with a sidebar on the left containing a search bar and a list of workspace resources. The main area displays a SQL query in a text editor, which is then executed to produce a table of results. The query filters for orders from 'Target' and calculates various metrics. The results table has columns for years\_count and order\_count.

```
#Is there a growing trend in the no. of orders placed over the past years?  
  
select Extract(year from order_purchase_timestamp) as years_count, count(order_id) as order_count from Target.orders  
group by years_count order by years_count;
```

Row	years_count	order_count
1	2016	329
2	2017	45101
3	2018	54011

## Q.2.b.answer

### Query

```
select Extract(month from order_purchase_timestamp) as month_count, count(order_id) as  
order_count from Target.orders  
group by month_count order by month_count asc;
```

### Screenshot

The screenshot shows a data analytics interface. On the left, there's a sidebar with a tree view of workspace resources. The main area displays a SQL query in a text editor, which is highlighted in blue. Below the query editor, there's a section for 'Query results' with tabs for 'JOB INFORMATION', 'RESULTS', 'JSON', 'EXECUTION DETAILS', 'CHART', 'PREVIEW', and 'EXECUTION GRAPH'. The 'RESULTS' tab is active, showing a table with 10 rows and 3 columns: 'Row', 'month\_count', and 'order\_count'. The table data is as follows:

Row	month_count	order_count
1	1	8069
2	2	8508
3	3	9893
4	4	9343
5	5	10573
6	6	9412
7	7	10318
8	8	10843
9	9	4305
10	10	4959

## Q.2.c.Answer

```
SELECT order_purchase_timestamp,
```

```
    CASE
```

```
    WHEN EXTRACT(HOUR FROM order_purchase_timestamp) BETWEEN 0 AND 6  
    THEN '0-6 hrs: Dawn'
```

```
    WHEN EXTRACT(HOUR FROM order_purchase_timestamp) BETWEEN 7 AND 12  
    THEN '7-12 hrs: Morning'
```

```
    WHEN EXTRACT(HOUR FROM order_purchase_timestamp) BETWEEN 13 AND 18  
    THEN '13-18 hrs: Afternoon'
```

```
    WHEN EXTRACT(HOUR FROM order_purchase_timestamp) BETWEEN 19 AND 23  
    THEN '19-23 hrs: Night'
```

```
    END AS order_time_of_day,
```

```
    COUNT(*) AS order_count
```

```
from `Target.orders`
```

```
GROUP BY order_purchase_timestamp, order_time_of_day
```

```
ORDER BY order_time_of_day;
```

### Explanation:

In the question it is clearly asked to segregate the data by range for that we can use the 'Case' condition which will segregate the data in 4 section 'Dawn', 'Morning', 'Afternoon' and 'Night' also, here we will count total orders placed by the customers for each section segregation this is

where group by roles come into the play. Also we could use the where clause with the country as 'Brazil' but we are assuming that the complete data is of "brazil" country.

### Screenshot:

The screenshot shows a SQL IDE interface. On the left is the 'Explorer' pane with a search bar and a tree view of workspace resources including 'notebook-387516', 'Saved queries (6)', 'External connections', 'Employee\_Schema', and 'Target' (which contains tables like customers, geolocation, order\_items, order\_reviews, orders, payments, products, sellers, and farmers\_market). The main area is the 'Untitled' query editor, which contains the following SQL query:

```

1 SELECT order_purchase_timestamp,
2
3     CASE
4         WHEN EXTRACT(HOUR FROM order_purchase_timestamp) BETWEEN 0 AND 6 THEN '0-6 hrs: Dawn'
5         WHEN EXTRACT(HOUR FROM order_purchase_timestamp) BETWEEN 7 AND 12 THEN '7-12 hrs: Morning'
6         WHEN EXTRACT(HOUR FROM order_purchase_timestamp) BETWEEN 13 AND 18 THEN '13-18 hrs: Afternoon'
7         WHEN EXTRACT(HOUR FROM order_purchase_timestamp) BETWEEN 19 AND 23 THEN '19-23 hrs: Night'
8     END AS order_time_of_day,
9     COUNT(*) AS order_count
10  FROM Target.orders
11 GROUP BY order_purchase_timestamp, order_time_of_day
12 ORDER BY order_time_of_day;
13

```

Below the query editor is the 'Query results' pane, which has tabs for 'JOB INFORMATION', 'RESULTS', 'JSON', 'EXECUTION DETAILS', 'CHART', 'PREVIEW', and 'EXECUTION GRAPH'. The 'RESULTS' tab is active, showing a table with 7 rows of data:

Row	order_purchase_timestamp	order_time_of_day	order_count
1	2017-12-05 01:07:58 UTC	0-6 hrs: Dawn	1
2	2017-12-05 01:07:52 UTC	0-6 hrs: Dawn	1
3	2018-01-16 01:25:39 UTC	0-6 hrs: Dawn	1
4	2017-12-07 00:02:16 UTC	0-6 hrs: Dawn	1
5	2018-03-05 03:47:11 UTC	0-6 hrs: Dawn	1
6	2017-04-17 00:10:48 UTC	0-6 hrs: Dawn	1
7	2018-05-07 01:24:47 UTC	0-6 hrs: Dawn	1

### Question.3.a

#### Query:

Select extract(month from o.order\_purchase\_timestamp) as order\_month, c.customer\_state, count(\*) as order\_count  
 from Target.orders o join Target.customers c on o.customer\_id=c.customer\_id  
 group by o.order\_purchase\_timestamp,c.customer\_state  
 order by order\_month, c.customer\_state

#### Explanation:

As mentioned, we require month-by-month (1,2,3,4,...n) orders placed for each state. In this, we will use the customer and order table to collect states and total order count using left join. The key point here is to extract data by month, and in the "order\_purchased\_timestamp." The month is not available separately, so we will extract all months.

### Screenshot

Viewing workspace resources.

SHOW STARRED ONLY

- notebook-387516
  - Saved queries (6)
  - External connections
  - Employee\_Schema
  - Target
    - customers
    - geolocation
    - order\_items
    - order\_reviews
    - orders
    - payments
    - products
    - sellers
    - farmers\_market

Untitled

[RUN](#) [SAVE](#) [SHARE](#) [SCHEDULE](#) [MORE](#)

```

1
2 #Get the month on month no. of orders placed in each state.
3
4 Select extract(month from o.order_purchase_timestamp) as order_month, c.customer_state, count(*) as order_count
5
6 from Target.orders o join Target.customers c on o.customer_id=c.customer_id
7
8 group by o.order_purchase_timestamp, c.customer_state
9
10 order by order_month, c.customer_state

```

Query results [SAVE RESULTS](#)

JOB INFORMATION RESULTS JSON EXECUTION DETAILS CHART [PREVIEW](#) EXECUTION GRAPH

Row	order_month	customer_state	order_count
1	1	AC	1
2	1	AC	1
3	1	AC	1
4	1	AC	1
5	1	AC	1
6	1	AC	1
7	1	AC	1
8	1	AC	1
9	1	AL	1

### Question.3.b

#### Query

```

SELECT distinct customer_state, COUNT(customer_unique_id) AS toal_customer
FROM Target.customers
GROUP BY customer_state
ORDER BY customer_state;

```

#### Explanation -

Customer distribution across all the states means here we have to find and count total unique customers for each state.

## Screenshot

The screenshot shows a SQL query editor interface. On the left is a sidebar with a search bar and a tree view of workspace resources. The main area displays a SQL query titled 'Untitled' with a 'RUN' button. Below the query editor, the 'Query results' section is visible, showing a table with two columns: 'customer\_state' and 'total\_customer'. The results are as follows:

Row	customer_state	total_customer
1	AC	81
2	AL	413
3	AM	148
4	AP	68
5	BA	3380
6	CE	1336
7	DF	2140
8	ES	2033
9	GO	2020

t:

### Q.4.answer.a.

#### Query

With MonthlyPayments as (

select

o.order\_id,

Extract(year from o.order\_purchase\_timestamp) as order\_year,

Extract(Month from o.order\_purchase\_timestamp) as order\_month,

Sum(P.payment\_value) as total\_payment

from Target.orders o

Join

Target.payments p on o.order\_id=p.order\_id

where

extract(year from o.order\_purchase\_timestamp) In (2017, 2018)

and extract(month from o.order\_purchase\_timestamp) between 1 and 8

Group by

o.order\_id,

extract (year from o.order\_purchase\_timestamp),

extract(month from o.order\_purchase\_timestamp)

)

SELECT

```

        (SUM(CASE WHEN order_year = 2018 THEN total_payment ELSE 0 END) -
SUM(CASE WHEN order_year = 2017 THEN total_payment ELSE 0 END))
        / SUM(CASE WHEN order_year = 2017 THEN total_payment ELSE 0 END) * 100 AS
percentage_increase
FROM
    MonthlyPayments
WHERE
    order_year IN (2017, 2018)
GROUP BY
    order_year;

```

```

select * from `Target.orders`

```

### **Explanation -**

Somehow i managed to write a query by understanding the problem but still not sure -

### **Q.4.answer.b.**

#### **Query**

```

SELECT c.customer_state AS State, SUM(p.payment_value) AS TotalOrderPrice,
AVG(p.payment_value) AS AverageOrderPrice
FROM Target.customers c
JOIN Target.orders o ON c.customer_id = o.customer_id
JOIN Target.payments p ON o.order_id = p.order_id
GROUP BY c.customer_state
ORDER BY c.customer_state;

```

### **Explanation**

In this problems there will be three tables will be engaged which are 'Customers' from which we will get each state, 'Orders' orders table will give us the order\_id for which we will find out total sum and avg of the price from the table 'Payment'

### **Screenshot**



Q.4. Answer.c

Query

```

select c.customer_state, sum(od.freight_value) as sum_freight_value, avg(od.freight_value) as
total_freight_avg from Target.customers c
join Target.orders o on c.customer_id=o.customer_id
join Target.order_items od on o.order_id=od.order_id
group by customer_state

```

Explanation:

Just like the above question, in this problems also, we will use three tables 'Customers', 'Order\_items' and 'Orders' - Frieght\_value can be calculated using the table order\_items although to connect with order\_items we will use 'order\_id' which is a common col for both tables "order\_items" and 'Orders' and result for each state 'Customer table' we will use group by.

Screenshot

Query results

Row	State	TotalOrderPrice	AverageOrderPrice
1	AC	19680.61999999...	234.2930952380...
2	AL	96962.05999999...	227.0774238875...
3	AM	27966.92999999...	181.6034415584...
4	AP	16262.79999999...	232.3257142857...
5	BA	616645.82000000...	170.8160166204...
6	CE	279464.02999999...	199.90273962804...
7	DF	355141.08000000...	161.1347912885...
8	ES	325967.55000000...	154.7069530137...

#### Q.4. Answer.c

#### Query

```

select c.customer_state, sum(od.freight_value) as sum_freight_value, avg(od.freight_value) as
total_freight_avg from Target.customers c
join Target.orders o on c.customer_id=o.customer_id
join Target.order_items od on o.order_id=od.order_id
group by customer_state

```

#### Explanation:

Just like the above question, in this problems also, we will use three tables 'Customers', 'Order\_items' and 'Orders' - Frieght\_value can be calculated using the table order\_items although to connect with order\_items we will use 'order\_id' which is a common col for both tables "order\_items" and 'Orders' and result for each state 'Customer table' we will use group by.

#### Screenshot

Query results

Row	customer_state	sum_freight_value	total_freight_avg
1	RN	18860.09999999...	35.65236294896...
2	CE	48351.58999999...	32.71420162381...
3	RS	135522.74000000...	21.73580433039...
4	SC	89660.26000000...	21.47036877394...
5	SP	718723.06999999...	15.14727539041...
6	MG	270853.46000000...	20.63016680630...
7	BA	100156.67999999...	26.36395893656...
8	RJ	305589.31000000...	20.96092393168...
9	GO	53114.97999999...	22.76681525932...

### Q.5.answer.1.

#### Query:

```
select order_id, (order_estimated_delivery_date-order_purchase_timestamp) as
delivery_time,
(order_estimated_delivery_date - order_delivered_customer_date) as
diffe_estimated_delivery
from `Target.orders`;
```

#### Screenshot

The screenshot shows a SQL query editor interface. On the left, there's a sidebar with a search bar and a list of workspace resources including 'notebook-387516', 'Saved queries (6)', 'External connections', 'Employee\_Schema', and 'Target'. The 'Target' database is expanded, showing tables like 'customers', 'geolocation', 'order\_items', 'order\_reviews', 'orders', 'payments', 'products', 'sellers', and 'farmers\_market'. The main editor area shows a query titled 'Untitled' with the following SQL code:

```
1 #Find the no. of days taken to deliver each order from the order's purchase date as delivery time.Also, calcula
2 estimated & actual delivery date of an order.Do this in a single query.
3 select order_id, (order_estimated_delivery_date-order_purchase_timestamp) as delivery_time,
4 (order_estimated_delivery_date - order_delivered_customer_date) as diffe_estimated_delivery
5 from `Target.orders`;
6
```

Below the query editor, there's a section for 'Query results' with tabs for 'JOB INFORMATION', 'RESULTS', 'JSON', 'EXECUTION DETAILS', 'CHART', 'PREVIEW', and 'EXECUTION GRAPH'. The 'RESULTS' tab is active, showing a table with 10 rows of data. The columns are 'Row', 'order\_id', 'delivery\_time', and 'diffe\_estimated\_delivery'. The data shows various order IDs and their corresponding delivery times and differences.

Row	order_id	delivery_time	diffe_estimated_delivery
1	7a4df5d8cf4090e541401a20a...	0-0 0 396:49:27	null
2	35de4050331c6c644cddc86f4...	0-0 0 814:52:2	null
3	b5359909123fa03c50bdb0cfe...	0-0 0 886:52:8	null
4	dba5062fbd3af4fb6c33b1e04...	0-0 0 606:38:56	null
5	90ab3e7d52544ec7bc3363c82...	0-0 0 586:47:26	null
6	fa65dad1b0e818e3ccc5cb0e3...	0-0 0 659:14:26	null
7	1df2775799eecd9dd8502425...	0-0 0 756:56:55	null
8	6190a94657e1012983a274b8...	0-0 0 802:23:30	null
9	58ce513a55c740a3a81e8c8b7...	0-0 0 365:54:53	null
10	088683f795a3d30bfd61152c4f...	0-0 0 757:57:13	null

### Q.5.Answer.b.

#### Query

```
SELECT c.customer_state, AVG(oi.freight_value) AS avg_freight_value
FROM customers AS c
JOIN orders AS o
ON c.customer_id = o.customer_id
JOIN order_items AS oi
ON o.order_id = oi.order_id
GROUP BY c.customer_state
ORDER BY avg_freight_value DESC
LIMIT 5

UNION ALL
```

```

SELECT c.customer_state,AVG(oi.freight_value) AS avg_freight_value
FROM customers AS c
JOIN orders AS o
ON c.customer_id = o.customer_id
JOIN order_items AS oi
ON o.order_id = oi.order_id
GROUP BY c.customer_state
ORDER BY avg_freight_value ASC
LIMIT 5;

```

This query gave me error everytime -

### Q.5.c

#### Query

```

SELECT
    c.customer_state,
    AVG(DATEDIFF(order_delivered_customer_date, order_purchase_timestamp)) AS
avg_delivery_time
FROM
    customers AS c
JOIN
    orders AS o
ON
    c.customer_id = o.customer_id
GROUP BY
    c.customer_state
ORDER BY
    avg_delivery_time DESC
LIMIT 5

```

UNION ALL

```

SELECT
    c.customer_state,
    AVG(DATEDIFF(order_delivered_customer_date, order_purchase_timestamp)) AS
avg_delivery_time
FROM
    customers AS c
JOIN
    orders AS o
ON

```

```

        c.customer_id = o.customer_id
GROUP BY
        c.customer_state
ORDER BY
        avg_delivery_time ASC
LIMIT 5;

```

#### Q.4.Answer.c.

##### Query

```

SELECT c.customer_state,
       AVG(DATE_DIFF(o.order_delivered_customer_date, o.Order_Estimated_Delivery_Date,
DAY)) AS AvgDeliveryDifference
FROM `Target.customers` as c
JOIN `Target.orders` as o ON c.Customer_ID = o.Customer_ID
GROUP BY c.customer_state
ORDER BY AvgDeliveryDifference ASC
LIMIT 5;

```

##### Screenshot

The screenshot shows a SQL query editor interface. The query is as follows:

```

1 SELECT c.customer_state,
2       AVG(DATE_DIFF(o.order_delivered_customer_date, o.Order_Estimated_Delivery_Date, DAY)) AS AvgDeliveryDifference
3 FROM `Target.customers` as c
4 JOIN `Target.orders` as o ON c.Customer_ID = o.Customer_ID
5 GROUP BY c.customer_state
6 ORDER BY AvgDeliveryDifference ASC
7 LIMIT 5;
8

```

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

Row	customer_state	AvgDeliveryDifference
1	AC	-19.7625
2	RO	-19.1316872427...
3	AP	-18.7313432835...
4	AM	-18.6068965517...
5	RR	-16.4146341463...

#### Q.5.Answer.1.

##### Query

```

SELECT
    EXTRACT(YEAR FROM o.Order_purchase_timestamp) AS Year, EXTRACT(MONTH FROM
o.Order_purchase_timestamp) AS Month,
    p.Payment_Type, COUNT(o.Order_ID) AS NumberOfOrders
FROM `Target.orders` o
JOIN `Target.payments` as p

```

```

ON o.Order_ID = p.Order_ID
GROUP BY Year, Month, Payment_Type
ORDER BY Year, Month, Payment_Type;

```

## Explanation

As asked in the question, we will use 'extract' years and month from the table orders and in order to find payment type we will use 'Payment' table and use the inner join to connect both tables using 'order\_id' because month we have to find on month no. of orders placed.

## Screenshot

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

```

1 SELECT
2   EXTRACT(YEAR FROM o.Order_purchase_timestamp) AS Year, EXTRACT(MONTH FROM o.Order_purchase_timestamp) AS Month,
3   p.Payment_Type, COUNT(o.Order_ID) AS NumberOfOrders
4 FROM `Target.orders` o
5 JOIN `Target.payments` as p
6 ON o.Order_ID = p.Order_ID
7 GROUP BY Year, Month, Payment_Type
8 ORDER BY Year, Month, Payment_Type;

```

The query results are displayed in a table with the following columns: Row, Year, Month, Payment\_Type, and NumberOfOrders.

Row	Year	Month	Payment_Type	NumberOfOrders
1	2016	9	credit_card	3
2	2016	10	UPI	63
3	2016	10	credit_card	254
4	2016	10	debit_card	2
5	2016	10	voucher	23
6	2016	12	credit_card	1
7	2017	1	UPI	197
8	2017	1	credit_card	583

## Question.5.Answer.2

### Query

```

SELECT p.payment_installments,COUNT(o.Order_ID) AS NumberOfOrders FROM
`Target.orders` as o LEFT JOIN `Target.payments` p ON o.Order_ID = p.Order_ID
GROUP BY p.payment_installments
ORDER BY p.payment_installments

```

## Screenshot

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

```

1 #Find the no. of orders placed on the basis of the payment installments that have been paid.
2
3 SELECT p.payment_installments,COUNT(o.Order_ID) AS NumberOfOrders FROM `Target.orders` as o LEFT JOIN `Target.payments` p ON o.Order_ID = p.Order_ID
4 GROUP BY p.payment_installments
5 ORDER BY p.payment_installments

```

The query results are displayed in a table with the following columns: Row, payment\_installment, and NumberOfOrders.

Row	payment_installment	NumberOfOrders
1	null	1
2	0	2
3	1	52546
4	2	12413
5	3	10461
6	4	7098
7	5	5239
8	6	3920
9	7	1626
10	8	4268

## Valuable Insights

- ☐ Here we have the range of all orders placed and purchased, this customized data can help 'Target' to understand the historical time of business operations. Also, this data will also help understanding in what hours the company should be proactive when it comes to showcase customized products to the users of brazil.
- ☐ From this result, we could do extensive research on how many users from all cities are purchasing and how the purchasing values of users could be enhanced.
- ☐ These timestamps can be used to calculate various performance metrics, such as the average order frequency or the total number of orders within the specified time frame.