1. **Data type of all columns in the "customers" table.**

01. **Table 1 - customers.csv**

	Field name	Type	Mode
☐	customer_id	STRING	NULLABLE
☐	customer_unique_id	STRING	NULLABLE
☐	customer_zip_code_prefix	INTEGER	NULLABLE
☐	customer_city	STRING	NULLABLE
☐	customer_state	STRING	NULLABLE

02. **Table 2 - sellers.csv**

	Field name	Type	Mode
☐	seller_id	STRING	NULLABLE
☐	seller_zip_code_prefix	INTEGER	NULLABLE
☐	seller_city	STRING	NULLABLE
☐	seller_state	STRING	NULLABLE

03. Table 3 - order_items.csv

	Field name	Type	Mode
☐	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

04. Table 4 - geolocation.csv

	Field name	Type	Mode
☐	geolocation_zip_code_prefix	INTEGER	NULLABLE
☐	geolocation_lat	FLOAT	NULLABLE
☐	geolocation_lng	FLOAT	NULLABLE
☐	geolocation_city	STRING	NULLABLE
☐	geolocation_state	STRING	NULLABLE

05. Table 5 - payments.csv

	Field name	Type	Mode
☐	order_id	STRING	NULLABLE
☐	payment_sequential	INTEGER	NULLABLE
☐	payment_type	STRING	NULLABLE
☐	payment_installments	INTEGER	NULLABLE
☐	payment_value	FLOAT	NULLABLE

06. Table 6 - order_reviews.csv

	Field name	Type	Mode
☐	review_id	STRING	NULLABLE
☐	order_id	STRING	NULLABLE
☐	review_score	INTEGER	NULLABLE
☐	review_comment_title	STRING	NULLABLE
☐	review_creation_date	TIMESTAMP	NULLABLE
☐	review_answer_timestamp	TIMESTAMP	NULLABLE

07. Table 7 - orders.csv

	Field name	Type	Mode
☐	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

4

08. Table 8 - products.csv

	Field name	Type	Mode
☐	product_id	STRING	NULLABLE
☐	product_category	STRING	NULLABLE
☐	product_name_length	INTEGER	NULLABLE
☐	product_description_length	INTEGER	NULLABLE
☐	product_photos_qty	INTEGER	NULLABLE
☐	product_weight_g	INTEGER	NULLABLE
☐	product_length_cm	INTEGER	NULLABLE
☐	product_height_cm	INTEGER	NULLABLE
☐	product_width_cm	INTEGER	NULLABLE

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

QUERY:
```
SELECT
  MIN(order_purchase_timestamp) AS fv,
  MAX(order_purchase_timestamp) AS fr
FROM
  customer.orders;
```

Query results ⬇ SAVE RESULTS ▾

JOB INFORMATION   RESULTS   CHART   JSON   EXECUTION DETAILS   EXECUTION GRAPH

Row	fv ▾	fr ▾
1	2016-09-04 21:15:19 UTC	2018-10-17 17:30:18 UTC

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

QUERY:
```
SELECT
 COUNT(DISTINCT customer_city) AS count_city,
 COUNT(DISTINCT customer_state) AS count_state
```

```
FROM
  customer.customers AS c
INNER JOIN
  customer.orders AS o
ON
  c.customer_id = o.customer_id
WHERE
  order_status = "delivered";
```

Row	count_city ▾	count_state ▾	
1	4085	27	

4. Is there a growing trend in the no. of orders placed over the past years?

```
SELECT
  EXTRACT(YEAR
  FROM
    order_purchase_timestamp) AS year,
  COUNT(order_id) AS count
FROM
  customer.orders
WHERE
  order_status = "delivered"
GROUP BY
  1
ORDER BY
  1;
```

Row	year ▾	count ▾	
1	2016	267	
2	2017	43428	
3	2018	52783	

QUERY:
```sql
 SELECT c.customer_state as states,
         EXTRACT(MONTH FROM o.order_purchase_timestamp) as months,
         count(o.order_id) as total_count
  FROM customer.orders as o inner join
  customer.customers as c
  on o.customer_id = c.customer_id
  group by 1,2
  order by 3 desc
```

Query results   ⬇ SAVE RESULTS ▾

JOB INFORMATION	RESULTS	CHART	JSON	EXECUTION DETAILS	EXECUTION GRAPH

Row	states ▾	years ▾	months ▾	total_count ▾	
1	RR	2016	9	1	
2	RS	2016	9	1	
3	SP	2016	9	2	
4	SP	2016	10	113	
5	RS	2016	10	24	
6	RJ	2016	10	56	
7	MT	2016	10	3	
8	GO	2016	10	9	

QUERY:
```sql
WITH
  cte AS (
  SELECT
    EXTRACT(YEAR
    FROM
      orders.order_purchase_timestamp) AS year_,
    SUM(payments.payment_value) AS revenue
  FROM
    customer.orders AS orders
```

```
  INNER JOIN
    customer.payments AS payments
  ON
    orders.order_id = payments.order_id
  WHERE
    EXTRACT(MONTH
    FROM
      orders.order_purchase_timestamp) BETWEEN 0
    AND 8
  GROUP BY
    Year_
  ORDER BY
    Year_)
SELECT
 *,
 LAG(revenue,1) OVER(ORDER BY year_) AS pre_revenue,
 (revenue - (LAG(revenue,1) OVER(ORDER BY year_)))/(LAG(revenue,1) OVER(ORDER BY
year_))*100 AS increse_per
FROM
  Cte
```

Query results

SAVE RESULTS ▾   EXPLO

JOB INFORMATION   **RESULTS**   CHART   JSON   EXECUTION DETAILS   EXECUTION GRAPH

Row	year_ ▾	revenue ▾	pre_revenue ▾	increse_per ▾	
1	2017	3669022.120000...	null	null	
2	2018	8694733.839999...	3669022.120000...	136.9768716466...	

```sql
SELECT
  (
    CASE
      WHEN (EXTRACT(HOUR FROM order_purchase_timestamp) >= 0) AND (EXTRACT(HOUR FROM
order_purchase_timestamp) <= 6) THEN "Dawn"
      WHEN (EXTRACT(HOUR
      FROM
        order_purchase_timestamp) >= 7)
    AND (EXTRACT(HOUR
      FROM
        order_purchase_timestamp) <= 12) THEN "Morning"
      WHEN (EXTRACT(HOUR FROM order_purchase_timestamp) >= 13) AND (EXTRACT(HOUR FROM
order_purchase_timestamp) <= 18) THEN "Afternoon"
    ELSE
    "Night"
  END
    ) AS tod,
  COUNT(order_id) AS total_count
FROM
  customer.orders
GROUP BY
  1
ORDER BY
  2 DESC;
```

Query results

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

Row	tod ▼	total_count ▼
1	Afternoon	38135
2	Night	28331
3	Mornings	27733
4	Dawn	5242

```sql
SELECT
  customer_state,
  COUNT(customer_id) as count_
FROM
  customer.customers
GROUP BY
  1
ORDER BY
  2 DESC;
```

Query results   ⬇ SAV

JOB INFORMATION	RESULTS	CHART	JSON	EXECUTION DETAILS	EXECUTION GRAPH

Row	customer_state ▼	count_ ▼
1	SP	41746
2	RJ	12852
3	MG	11635
4	RS	5466
5	PR	5045
6	SC	3637
7	BA	3380
8	DF	2140

```sql
SELECT
 orders.order_id,
 orders.order_purchase_timestamp,
 orders.order_delivered_customer_date,
 orders.order_estimated_delivery_date,

TIMESTAMP_DIFF(orders.order_delivered_customer_date,orders.order_purchase_timestamp,
DAY) AS time_to_Deliver,
 TIMESTAMP_DIFF(orders.order_estimated_delivery_date,
orders.order_delivered_customer_date, DAY) AS diff_estimated_delivery
```

```sql
FROM
  customer.orders
GROUP BY
  1,
  2,
  3,
  4
```

Query results							

JOB INFORMATION   **RESULTS**   CHART   JSON   EXECUTION DETAILS   EXECUTION GRAPH

Row	order_id ▼	order_purchase_timestamp	order_delivered_customer_date ▼	order_estimated_delivery_date ▼	time_to_Deliver ▼	diff_estimated_delivery ▼
1	1950d777989f6a8...	2018-02-19 19:48:52 UTC	2018-03-21 22:03:51 UTC	2018-03-09 00:00:00 UTC	30	-12
2	2c45c33d2f9cb8ff...	2016-10-09 15:39:56 UTC	2016-11-09 14:53:50 UTC	2016-12-08 00:00:00 UTC	30	28
3	65d1e226dfaeb8cd...	2016-10-03 21:01:41 UTC	2016-11-08 10:58:34 UTC	2016-11-25 00:00:00 UTC	35	16
4	635c894d068ac37...	2017-04-15 15:37:38 UTC	2017-05-16 14:49:55 UTC	2017-05-18 00:00:00 UTC	30	1
5	3b97562c3aee8bd...	2017-04-14 22:21:54 UTC	2017-05-17 10:52:15 UTC	2017-05-18 00:00:00 UTC	32	0
6	68f47f50f04c4cb6...	2017-04-16 14:56:13 UTC	2017-05-16 09:07:47 UTC	2017-05-18 00:00:00 UTC	29	1
7	276e9ec344d3bf0...	2017-04-08 21:20:24 UTC	2017-05-22 14:11:31 UTC	2017-05-18 00:00:00 UTC	43	-4
8	54e1a3c2b97fb08...	2017-04-11 19:49:45 UTC	2017-05-22 16:18:42 UTC	2017-05-18 00:00:00 UTC	40	-4

10. **Find out the top 5 states with the highest & lowest average freight value.**
    Hint: We want you to find the top 5 & the bottom 5 states arranged in increasing order of the average freight value.

TOP 5 STATES WITH HIGHEST freight_value AVERAGE

QUERY:

```sql
SELECT
  c.customer_state,
  ROUND(AVG(oi.freight_value),2) AS AVERAGE_
FROM (customer.orders AS o
  INNER JOIN
    customer.customers AS c
  ON
    o.customer_id = c.customer_id
  INNER JOIN
    customer.order_items AS oi
  ON
    oi.order_id = o.order_id)
GROUP BY
  1
```

```
ORDER BY
 2 DESC
LIMIT
 5
```

Query results

⬇ SAVE RE

| | JOB INFORMATION | **RESULTS** | CHART | JSON | EXECUTION DETAILS | EXECUTION GRAPH |

Row	customer_state ▼	AVERAGE_ ▼
1	RR	42.98
2	PB	42.72
3	RO	41.07
4	AC	40.07
5	PI	39.15

BOTTOM 5 STATES WITH LOWEST freight_value AVERAGE

QUERY:

```
SELECT
 c.customer_state,
 ROUND(AVG(oi.freight_value),2) AS AVERAGE_
FROM (customer.orders AS o
 INNER JOIN
   customer.customers AS c
 ON
   o.customer_id = c.customer_id
 INNER JOIN
   customer.order_items AS oi
 ON
   oi.order_id = o.order_id)
GROUP BY
 1
ORDER BY
 2 asc
LIMIT
 5
```

Query results

JOB INFORMATION	RESULTS	CHART	JSON	EXECUTION DETAILS	EXECUTION GRAPH

Row	customer_state ▾	AVERAGE_ ▾
1	SP	15.15
2	PR	20.53
3	MG	20.63
4	RJ	20.96
5	DF	21.04

11. Find the month on month no. of orders placed using different payment types.
Hint: We want you to count the no. of orders placed using different payment methods in each month over the past years

```sql
SELECT
 EXTRACT(YEAR
 FROM
    o.order_purchase_timestamp) AS Year_,
 EXTRACT(MONTH
 FROM
    o.order_purchase_timestamp) AS Month_,
 p.payment_type AS pay_method,
 COUNT(p.order_id) AS count_
FROM
 customer.payments AS p
INNER JOIN
 customer.orders AS o
ON
 o.order_id = p.order_id
GROUP BY
 1,
 2,
 3
ORDER BY
 4 DESC;
```

Query results

JOB INFORMATION   RESULTS   CHART   JSON   EXECUTION DETAILS   EXECUTION GRAPH

Row	Year_ ▾	Month_ ▾	pay_method ▾	count_ ▾
1	2018	5	credit_card	5497
2	2018	4	credit_card	5455
3	2018	1	voucher	416
4	2017	4	voucher	202
5	2017	10	voucher	291
6	2018	9	not_defined	1
7	2018	8	not_defined	2
8	2017	6	voucher	239

12. Find the no. of orders placed on the basis of the payment installments that have been paid. Hint: We want you to count the no. of orders placed based on the no. of payment installments where at least one installment has been successfully paid.

```sql
SELECT
 p.payment_installments,
 COUNT(p.order_id) AS count_
FROM
 customer.payments AS p
WHERE
 p.payment_installments != 0
GROUP BY
 1
ORDER BY
 2 DESC;
```

Query results

JOB INFORMATION   RESULTS   CHART   JSON   EXECUTION DETAILS   EXECUTION GRAPH

Row	payment_installments ▾	count_ ▾
1	1	52546
2	2	12413
3	3	10461
4	4	7098
5	10	5328
6	5	5239
7	8	4268
8	6	3920

Insights:

- Orders from the dataset were purchased from (2016-09-04) to (2018-10-17).
- Orders volume increased year by year i.e count of orders from 2016 increased from 267 to 52783 in 2018.
- Most of the orders placed in the afternoon, suggesting promotions for Target occur most in this time.
- Orders purchased across 4085 cities and 27 cities of Brazil. There's a variation in average order value across states. Target may explore optimizing pricing strategies based on location and customer segments.
- Understanding popular payment methods (credit card, debit card, installments) can help tailor the checkout process. A significant portion of customers use installments, indicating a preference for spreading purchase costs. Target might explore offering more attractive installment plans or partnering with financing companies.

Recommendations:

- Implement data-driven promotions based on customer location, and order history.
- Focus on improving delivery times in states with slower performance.
- Develop location-specific pricing strategies considering customer segments and average order values.
- Offer a variety of payment options including installments to cater to customer preferences.
- Segment customers based on demographics, purchase behavior, and location to personalize marketing efforts.
- Optimize inventory levels in warehouses across Brazil to ensure faster delivery times.