Creating tables


```
CREATE TABLE amazon_brazil.customers (
        customer_id VARCHAR PRIMARY KEY,
        customer_unique_id VARCHAR,
        customer_zip_code_prefix INTEGER
);

CREATE TABLE amazon_brazil.orders (
        order_id VARCHAR PRIMARY KEY,
        customer_id VARCHAR,
        order_status VARCHAR,
        order_purchase_timestamp TIMESTAMP,
        order_approved_at TIMESTAMP,
        order_delivered_carrier_date TIMESTAMP,
        order_delivered_customer_date TIMESTAMP,
        order_estimated_delivery_date TIMESTAMP,
        FOREIGN KEY (customer_id) REFERENCES
amazon_brazil.customers(customer_id)
);
```

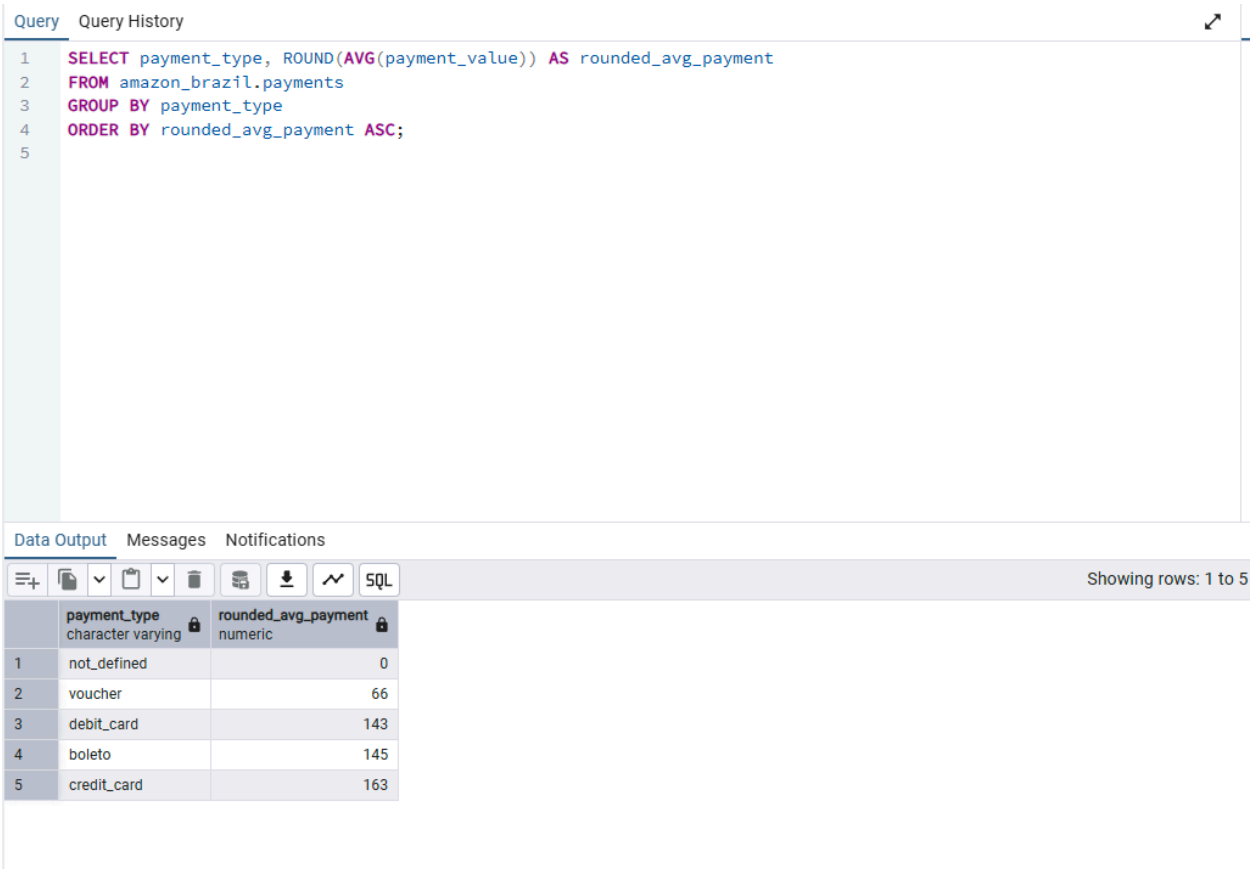
```
CREATE TABLE amazon_brazil.payments (
        order_id VARCHAR,
        payment_sequential INTEGER,
        payment_type VARCHAR,
        payment_installments INTEGER,
        payment_value NUMERIC, # was INTEGER
        PRIMARY KEY (order_id, payment_sequential),
        FOREIGN KEY (order_id) REFERENCES amazon_brazil.orders(order_id)
);

CREATE TABLE amazon_brazil.seller (
        seller_id VARCHAR PRIMARY KEY,
        seller_zip_code_prefix INTEGER
);

CREATE TABLE amazon_brazil.product (
        product_id VARCHAR PRIMARY KEY,
        product_category_name VARCHAR,
        product_name_length INTEGER,
        product_description_length INTEGER,
        product_photos_qty INTEGER,
        product_weight_g INTEGER,
        product_length_cm INTEGER,
        product_height_cm INTEGER,
        product_width_cm INTEGER
);
 CREATE TABLE amazon_brazil.order_items (
        order_id VARCHAR,
        order_item_id INTEGER,
        product_id VARCHAR,
        seller_id VARCHAR,
        shipping_limit_date TIMESTAMP,
        price NUMERIC,
        freight_value NUMERIC,
        PRIMARY KEY (order_id, order_item_id),
        FOREIGN KEY (order_id) REFERENCES amazon_brazil.orders(order_id),
        FOREIGN KEY (product_id) REFERENCES amazon_brazil.product(product_id),
        FOREIGN KEY (seller_id) REFERENCES amazon_brazil.seller(seller_id)
```

);

Analysis 1

1) **To simplify its financial reports, Amazon India needs to standardize payment values.** Round the average payment values to integer (no decimal) for each payment type and display the results sorted in ascending order.

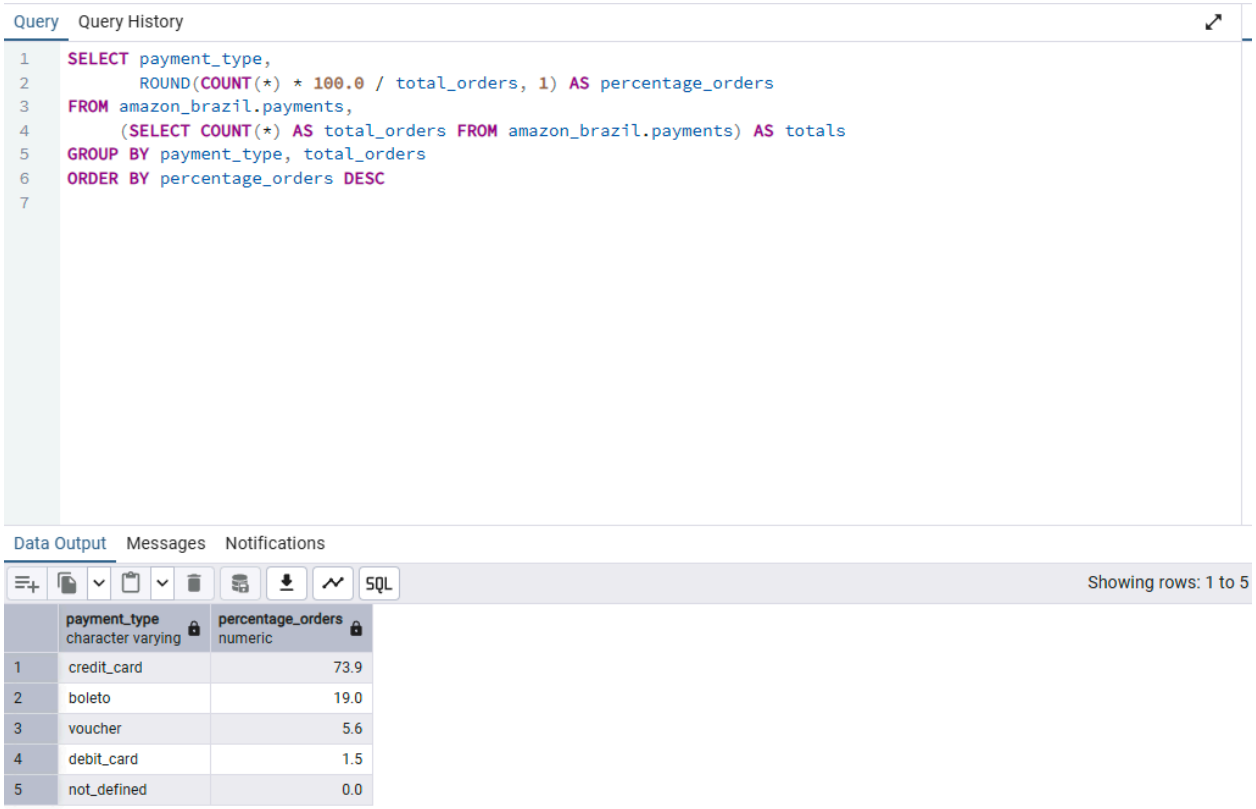
- **Output**: `payment_type, rounded_avg_payment`




```
SELECT payment_type, ROUND(AVG(payment_value)) AS rounded_avg_payment
FROM amazon_brazil.payments
GROUP BY payment_type
ORDER BY rounded_avg_payment ASC;
```

2) **To refine its payment strategy, Amazon India wants to know the distribution of orders by payment type.** Calculate the percentage of total orders for each payment type, rounded to one decimal place, and display them in descending order

- **Output**: `payment_type, percentage_orders`




```
SELECT payment_type,
        ROUND(COUNT(*) * 100.0 / total_orders, 1) AS percentage_orders
FROM amazon_brazil.payments,
        (SELECT COUNT(*) AS total_orders FROM amazon_brazil.payments) AS totals
GROUP BY payment_type, total_orders
ORDER BY percentage_orders DESC
```

3) **Amazon India seeks to create targeted promotions for products within specific price ranges.** Identify all products priced between 100 and 500 BRL that contain the word 'Smart' in their name. Display these products, sorted by price in descending order.

- **Output**: `product_id, price`

Query | Query History

```sql
SELECT distinct
    oi.product_id,
    oi.price
FROM amazon_brazil.order_items oi
JOIN amazon_brazil.product p ON oi.product_id = p.product_id
WHERE oi.price BETWEEN 100 AND 500
  AND p.product_category_name ILIKE '%Smart%'
ORDER BY oi.price DESC;
```

Data Output | Messages | Notifications

Showing rows: 1 to 1

	product_id character varying	price numeric
1	1df1a2df8ad2b9d3aa49fd851e3145...	439.99
2	7debe59b10825e89c1cbcc8b190c8...	349.99
3	ca86b9fe16e12de698c955aedff0ae...	349
4	0e52955ca8143bd179b311cc454a6...	335
5	7aeaa8f3e592e380c420e8910a717...	329.9
6	d1b571cd58267d8cac8b2afd6e288...	299.9
7	66ffe28d0fd53808d0535eee4b90a1...	254
8	f06796447de379a26dde5fcac6a1a2...	239.9
9	d3d5a1d52abe9a7d234908d873fc3...	229.9
10	06ae026e430189633c2fbd0288c86...	217.36
11	49ef750dc5bf23e3788d4f614bc6db...	198
12	33bb7da523efcdef6cd2996cbf72d0...	148
13	6f5795735ab2c629b22669fe889b7...	129.9

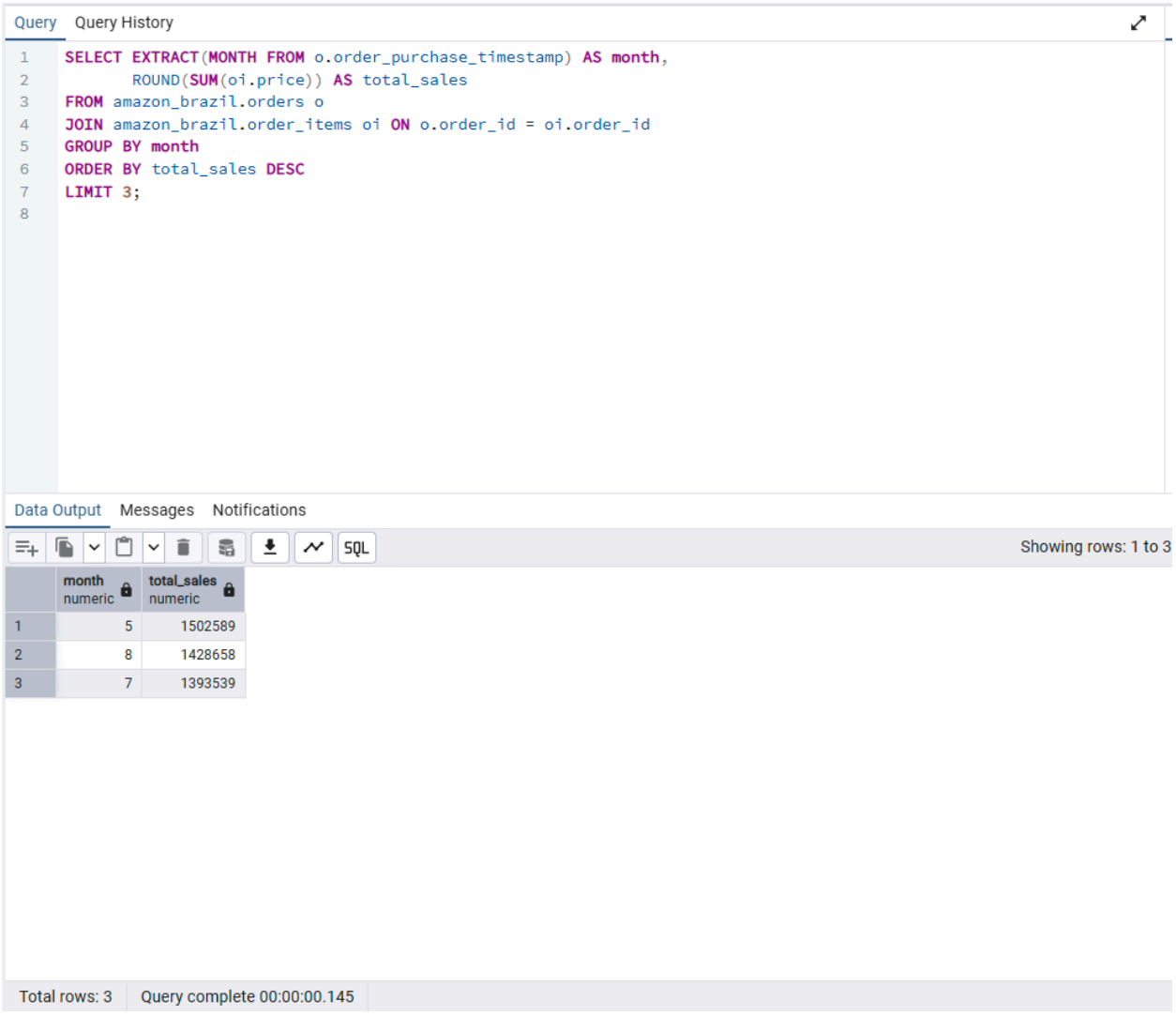
Total rows: 19 | Query complete 00:00:00.058

SELECT distinct
    oi.product_id,
    oi.price
FROM amazon_brazil.order_items oi

```
JOIN amazon_brazil.product p ON oi.product_id = p.product_id
WHERE oi.price BETWEEN 100 AND 500
  AND p.product_category_name ILIKE '%Smart%'
ORDER BY oi.price DESC;
```

4) **To identify seasonal sales patterns, Amazon India needs to focus on the most successful months.** Determine the top 3 months with the highest total sales value, rounded to the nearest integer.

- **Output**: `month, total_sales`



```
SELECT EXTRACT(MONTH FROM o.order_purchase_timestamp) AS month,
       ROUND(SUM(oi.price)) AS total_sales
FROM amazon_brazil.orders o
JOIN amazon_brazil.order_items oi ON o.order_id = oi.order_id
GROUP BY month
ORDER BY total_sales DESC
LIMIT 3;
```

5) **Amazon India is interested in product categories with significant price variations.** Find categories where the difference between the maximum and minimum product prices is greater than 500 BRL.

- **Output**: `product_category_name, price_difference`

Query | Query History

```
SELECT p.product_category_name, MAX(oi.price) - MIN(oi.price) AS price_difference
FROM amazon_brazil.order_items oi
join amazon_brazil.product p on oi.product_id = p.product_id
GROUP BY p.product_category_name
HAVING MAX(oi.price) - MIN(oi.price) > 500;
```

Data Output | Messages | Notifications

SQL | Showing rows: 1 to 57

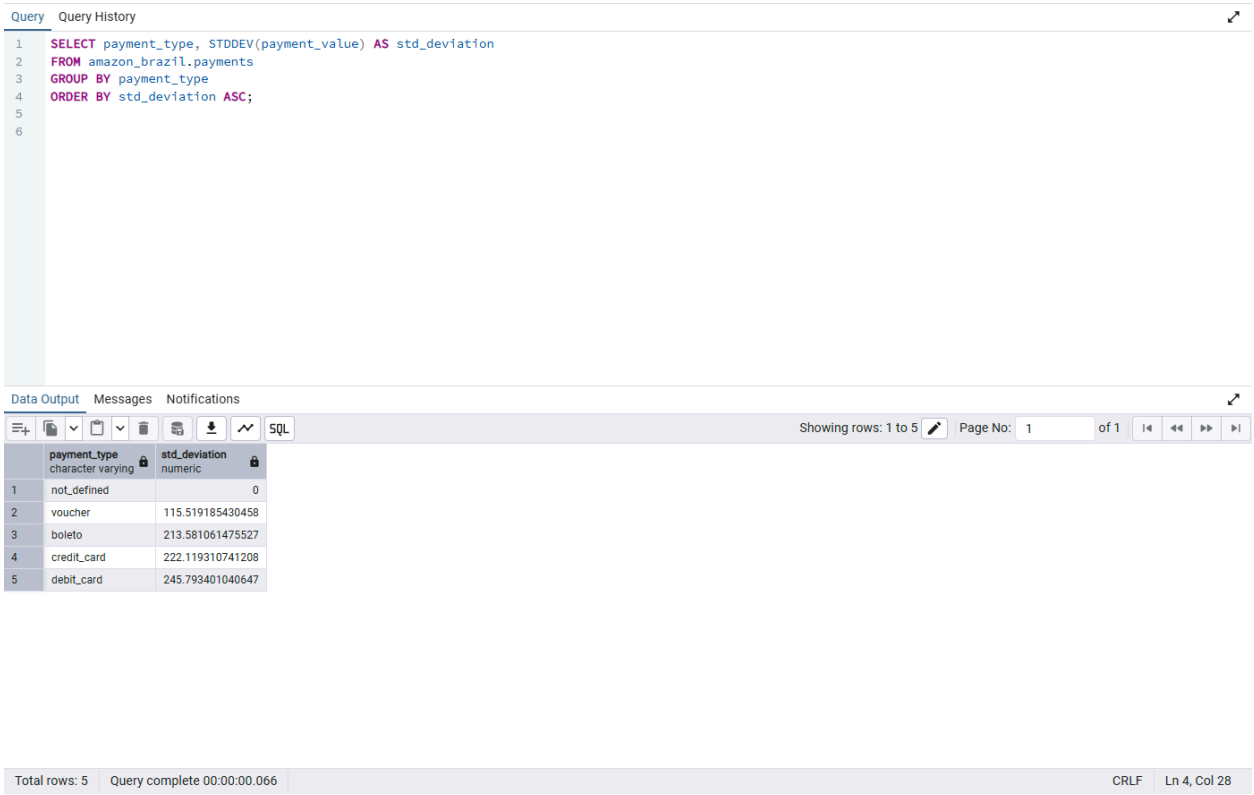
	product_category_name character varying	price_difference numeric
1	climatizacao	1588.1
2	livros_importados	730.01
3	[null]	3977
4	ferramentas_jardim	3923.65
5	dvds_blu_ray	1411.1
6	cine_foto	867.19
7	beleza_saude	3122.8
8	livros_interesse_geral	893.9
9	tablets_impressao_imagem	875.09
10	papelaria	1690.71
11	bebes	3895.46
12	musica	1162.12
13	consoles_games	4094.81

Total rows: 57 | Query complete 00:00:00.096

```
SELECT p.product_category_name, MAX(oi.price) - MIN(oi.price) AS price_difference
FROM amazon_brazil.order_items oi
join amazon_brazil.product p on oi.product_id = p.product_id
GROUP BY p.product_category_name
HAVING MAX(oi.price) - MIN(oi.price) > 500;
```

6) **To enhance the customer experience, Amazon India wants to find which payment types have the most consistent transaction amounts.** Identify the payment types with the least variance in transaction amounts, sorting by the smallest standard deviation first.

- **Output**: `payment_type, std_deviation`




```
SELECT payment_type, STDDEV(payment_value) AS std_deviation
```

FROM amazon_brazil.payments
GROUP BY payment_type
ORDER BY std_deviation ASC;

7) **Amazon India wants to identify products that may have incomplete name in order to fix it from their end**. Retrieve the list of products where the product category name is missing or contains only a single character.

- **Output**: `product_id, product_category_name`

Query | Query History

```
1  SELECT product_id, product_category_name
2  FROM amazon_brazil.product
3  WHERE product_category_name IS NULL
4     OR LENGTH(product_category_name) <= 1;
5
6
7
```

Data Output | Messages | Notifications

SQL | Showing rows: 1 to 614

	product_id [PK] character varying	product_category_name character varying
1	a41e356c76fab66334f36de622ecbd3a	[null]
2	d8dee61c2034d6d075997acef1870e…	[null]
3	56139431d72cd51f19eb9f7dae4d16…	[null]
4	46b48281eb6d663ced748f324108c7…	[null]
5	5fb61f482620cb672f5e586bb132eae9	[null]
6	e10758160da97891c2fdcbc35f0f031d	[null]
7	39e3b9b12cd0bf8ee681bbc1c130feb5	[null]
8	794de06c32a626a5692ff50e4985d36f	[null]
9	7af3e2da474486a3519b0cba9dea8a…	[null]
10	629beb8e7317703dcc5f35b5463fd20e	[null]
11	3a78f64aac654298e4b9aff32fc21818	[null]
12	bcb815bba008d89458e428078c0b92…	[null]
13	6b82874c6b51b92913dcdb364eaaae…	[null]

Total rows: 614 | Query complete 00:00:00.073

SELECT product_id, product_category_name

```
FROM amazon_brazil.product
WHERE product_category_name IS NULL
   OR LENGTH(product_category_name) <= 1;
```

Analysis 2

1) **Amazon India wants to understand which payment types are most popular across different order value segments (e.g., low, medium, high).** Segment order values into three ranges: orders less than 200 BRL, between 200 and 1000 BRL, and over 1000 BRL. Calculate the count of each payment type within these ranges and display the results in descending order of count

- **Output:** `order_value_segment, payment_type, count`

Query | Query History

```
with order_values as (
select o.order_id, sum(oi.price) as total_val from amazon_brazil.orders o
join amazon_brazil.order_items oi on o.order_id = oi.order_id
group by o.order_id
),
order_segments as (
select ov.order_id,
case
when ov.total_val < 200 then 'Low'
when ov.total_val between 200 and 1000 then 'Medium'
else 'High'
end as order_value_segment
from order_values ov
)

select os.order_value_segment, p.payment_type, count(*) as count
from order_segments os join amazon_brazil.payments p
on os.order_id  = p.order_id
group by os.order_value_segment,payment_type
order by count desc
```

Data Output | Messages | Notifications

Showing rows: 1 to 12

	order_value_segment text	payment_type character varying	count bigint
1	Low	credit_card	63744
2	Low	boleto	17195
3	Medium	credit_card	11733
4	Low	voucher	4973
5	Medium	boleto	2274
6	Low	debit_card	1353
7	High	credit_card	801
8	Medium	voucher	635
9	Medium	debit_card	157
10	High	boleto	145
11	High	voucher	34
12	High	debit_card	12

Total rows: 12 | Query complete 00:00:00.227

```
with order_values as (
select o.order_id, sum(oi.price) as total_val from amazon_brazil.orders o
join amazon_brazil.order_items oi on o.order_id = oi.order_id
group by o.order_id
),
order_segments as (
select ov.order_id,
case
when ov.total_val < 200 then 'Low'
when ov.total_val between 200 and 1000 then 'Medium'
else 'High'
end as order_value_segment
from order_values ov
)

select os.order_value_segment, p.payment_type, count(*) as count
from order_segments os join amazon_brazil.payments p
on os.order_id  = p.order_id
group by os.order_value_segment,payment_type
order by count desc
```

2)**Amazon India wants to analyse the price range and average price for each product category.** Calculate the minimum, maximum, and average price for each category, and list them in descending order by the average price.

- **Output**: `product_category_name, min_price, max_price, avg_price`

Query | Query History

```
select p.product_category_name, min(oi.price),max(oi.price),avg(oi.price) from amazon_brazil.product p
join amazon_brazil.order_items oi
on p.product_id = oi.product_id
group by p.product_category_name
order by avg(oi.price) desc
```

Data Output | Messages | Notifications

Showing rows: 1 to 79

	product_category_name character varying	min numeric	max numeric	avg numeric
1	pcs	34.5	6729	1098.3405418719211823
2	portateis_casa_forno_e_cafe	10.19	2899	624.2856578947368421
3	eletrodomesticos_2	13.9	2350	476.1249579831932773
4	agro_industria_e_comercio	12.99	2990	341.6610426540284360
5	instrumentos_musicais	4.9	4399.87	281.6160000000000000
6	eletroportateis	6.5	4799	280.7784683357879234
7	portateis_cozinha_e_preparadores_de_alimentos	17.42	1099	264.5686666666666667
8	telefonia_fixa	6	1790	225.6931818181818182
9	construcao_ferramentas_seguranca	8.9	3099.9	208.9923711340206186
10	relogios_presentes	8.99	3999.9	200.9118770875083500
11	climatizacao	10.9	1599	185.2692255892255892
12	moveis_quarto	6.9	650	183.7502752293577982

Total rows: 79 | Query complete 00:00:00.144

select p.product_category_name, min(oi.price),max(oi.price),avg(oi.price) from amazon_brazil.product p
join amazon_brazil.order_items oi
on p.product_id = oi.product_id
group by p.product_category_name
order by avg(oi.price) desc

3) **Amazon India wants to identify the customers who have placed multiple orders over time.** Find all customers with more than one order, and display their customer unique IDs along with the total number of orders they have placed.

- **Output:** `customer_unique_id, total_orders`

Query | Query History

```
select c.customer_unique_id, count(distinct o.order_id) as order_count from amazon_brazil.customers c
join amazon_brazil.orders o
on c.customer_id = o.customer_id
group by customer_unique_id
having count(distinct o.order_id)>1
order by order_count desc
```

Data Output | Messages | Notifications

Showing rows: 1 to 1000

	customer_unique_id character varying	order_count bigint
1	a91e80fbe80ddc07de66a5cf9270293c	16
2	a6168cd79131e64acef92e3c74d6cc...	16
3	363f980585bf04c1a88fdb986011c52e	16
4	cbd0350d4ccba9772e8e768d4a4a5c...	16
5	417b909c0962b2610f1cfeb1c1478986	16
6	5f94af52aef02c968a2e0f01f430864e	16
7	1b6d29725255a77667a8c639eeb4cc...	16
8	e4bbcc533fdf3917c56dea2c43bf2084	16
9	930c4390af58f67334447c3a1cf2ba36	16
10	5bf4ea2d98005b960eea0dbf652ef4e7	16
11	9159c04b88895d995741dd5b9b7a5f...	16
12	4034aa08d48695a538b7030910aae5...	16

Successfully run. T

Total rows: 3140 | Query complete 00:00:00.499

select c.customer_unique_id, count(distinct o.order_id) as order_count from
amazon_brazil.customers c
join amazon_brazil.orders o
on c.customer_id = o.customer_id
group by customer_unique_id
having count(distinct o.order_id)>1
order by order_count desc

4) **Amazon India wants to categorize customers into different types ('New – order qty. = 1' ; 'Returning' –order qty. 2 to 4; 'Loyal' – order qty. >4) based on their purchase history.** Use a temporary table to define these categories and join it with the customers table to update and display the customer types.

- **Output**: `customer_unique_id, customer_type`

Query | Query History

```
with customer_order_count as(
select c.customer_unique_id, count(distinct o.order_id) as order_count from amazon_brazil.customers c
left join amazon_brazil.orders o
on c.customer_id = o.customer_id
group by customer_unique_id
)

select customer_unique_id,
case
when order_count =1 then 'New'
when order_count between 2 and 4 then 'Returning'
when order_count > 4 then 'Loyal'
else 'No Orders'
end  as customer_type
from customer_order_count
```

Data Output | Messages | Notifications

Showing rows: 1 to 1000

	customer_unique_id character varying	customer_type text
1	0000366f3b9a7992bf8c76cfdf3221...	New
2	0000b849f77a49e4a4ce2b2a4ca5b...	New
3	0000f46a3911fa3c0805444483337...	New
4	0000f6ccb0745a6a4b88665a16c9f...	New
5	0004aac84e0df4da2b147fca70cf82...	New
6	0004bd2a26a76fe21f786e4fbd8060...	New
7	00050ab1314c0e55a6ca13cf7181fe...	New
8	00053a61a98854899e70ed204dd4b...	New
9	0005e1862207bf6ccc02e4228effd9...	New
10	0005ef4cd20d2893f0d9fbd94d3c0d...	New
11	0006fdc98a402fceb4eb0ee528f6a8...	New
12	00082cbe03e478190aadbea78542e...	No Orders

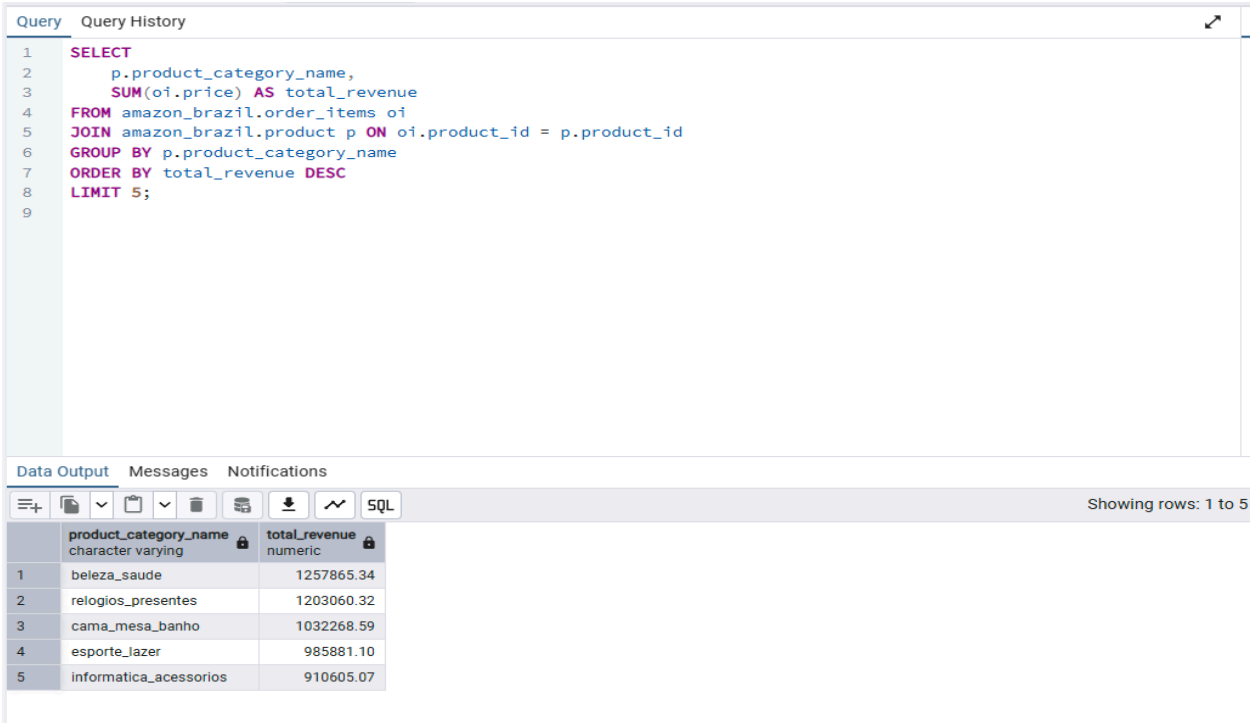
Total rows: 96096 | Query complete 00:00:00.498

with customer_order_count as(
select c.customer_unique_id, count(distinct o.order_id) as order_count from amazon_brazil.customers c

```
left join amazon_brazil.orders o
on c.customer_id = o.customer_id
group by customer_unique_id
)

select customer_unique_id,
case
when order_count =1 then 'New'
when order_count between 2 and 4 then 'Returning'
when order_count > 4 then 'Loyal'
else 'No Orders'
end  as customer_type
from customer_order_count
```

5) **Amazon India wants to know which product categories generate the most revenue.** Use joins between the tables to calculate the total revenue for each product category. Display the top 5 categories.

- **Output**: `product_category_name, total_revenue`

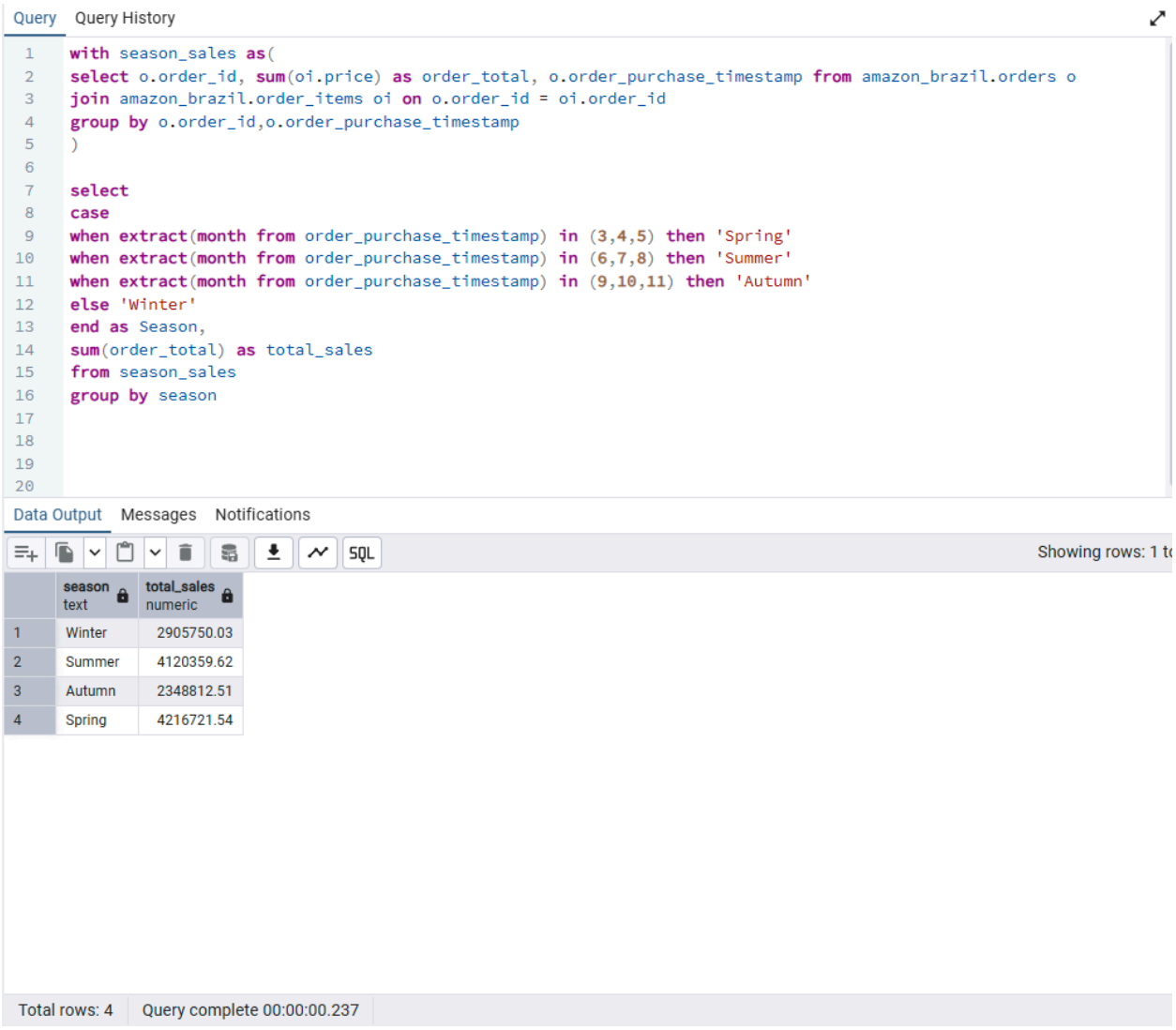


```
SELECT
	p.product_category_name,
	SUM(oi.price) AS total_revenue
FROM amazon_brazil.order_items oi
JOIN amazon_brazil.product p ON oi.product_id = p.product_id
GROUP BY p.product_category_name
ORDER BY total_revenue DESC
LIMIT 5;
```

Analysis 3

1) **The marketing team wants to compare the total sales between different seasons.** Use a subquery to calculate total sales for each season (Spring, Summer, Autumn, Winter) based on order purchase dates, and display the results. Spring is in the months of March, April and May. Summer is from June to August and Autumn is between September and November and rest months are Winter.

- **Output:** `season, total_sales`




```
with season_sales as(
select o.order_id, sum(oi.price) as order_total, o.order_purchase_timestamp from
amazon_brazil.orders o
join amazon_brazil.order_items oi on o.order_id = oi.order_id
group by o.order_id,o.order_purchase_timestamp
)

select
case
when extract(month from order_purchase_timestamp) in (3,4,5) then 'Spring'
when extract(month from order_purchase_timestamp) in (6,7,8) then 'Summer'
when extract(month from order_purchase_timestamp) in (9,10,11) then 'Autumn'
else 'Winter'
end as Season,
```

```
sum(order_total) as total_sales
from season_sales
group by season
```

2) **The inventory team is interested in identifying products that have sales volumes above the overall average.** Write a query that uses a subquery to filter products with a total quantity sold above the average quantity.

- **Output:** `product_id, total_quantity_sold`

Query | Query History

```
with sales as(
select product_id,count(order_item_id) as total_quantity from amazon_brazil.order_items
group by product_id
),

average_sales as (
select avg(total_quantity) as avg_quantity from sales
)

-- AVERAGE QUANTITY = 3
select product_id, total_quantity  from sales
where total_quantity> (select avg_quantity from average_sales)
```

Data Output | Messages | Notifications

Showing rows: 1 to 1000

	product_id character varying	total_quantity bigint
1	3d5837f86205fe83f03fb5f7e4d5b9cf	11
2	afeeea6271148ee1bb15173b8187c431	53
3	434487f82b5c35646bd8155cf1946179	4
4	e5063ce7fff1cf7cd528dc4c1e7dcba8	4
5	b25a0f93e25104798df2d1664495d157	4
6	6639a238ead6779d6ef0b3eea56f9f86	4
7	dceb3f67aef3484498a7caa4ba50f484	6
8	3c4e3782469a0f1ac459dc6c47ebef31	4
9	ecaaaccb5eb3102553f001d62db6389e	6
10	98ad26989524a790f1d29686025b6fcc	4
11	de4fb1ddae276a3503afed39c8227cff	4
12	9048cbd294fe0c1a3ec8c8248bc2cadd	15
13	4464ecc5c8cd38eff5beae1484f80166	11

Total rows: 6366 | Query complete 00:00:00.108


```
with sales as(
select product_id,count(order_item_id) as total_quantity from
amazon_brazil.order_items
group by product_id
),

average_sales as (
select avg(total_quantity) as avg_quantity from sales
)
-- AVERAGE QUANTITY = 3
select product_id, total_quantity from sales
where total_quantity> (select avg_quantity from average_sales)
```

3) **To understand seasonal sales patterns, the finance team is analysing the monthly revenue trends over the past year (year 2018).** Run a query to calculate total revenue generated each month and identify periods of peak and low sales. Export the data to Excel and create a graph to visually represent revenue changes across the months.

- **Output:** `month, total_revenue`

Query | Query History

```
select extract(month from o.order_purchase_timestamp) as month, sum(oi.price) as total_revenue from amazon_brazil.or
join amazon_brazil.order_items oi
on o.order_id = oi.order_id
where extract(year from o.order_purchase_timestamp) = 2018
group by month
order by month
```

Data Output | Messages | Notifications

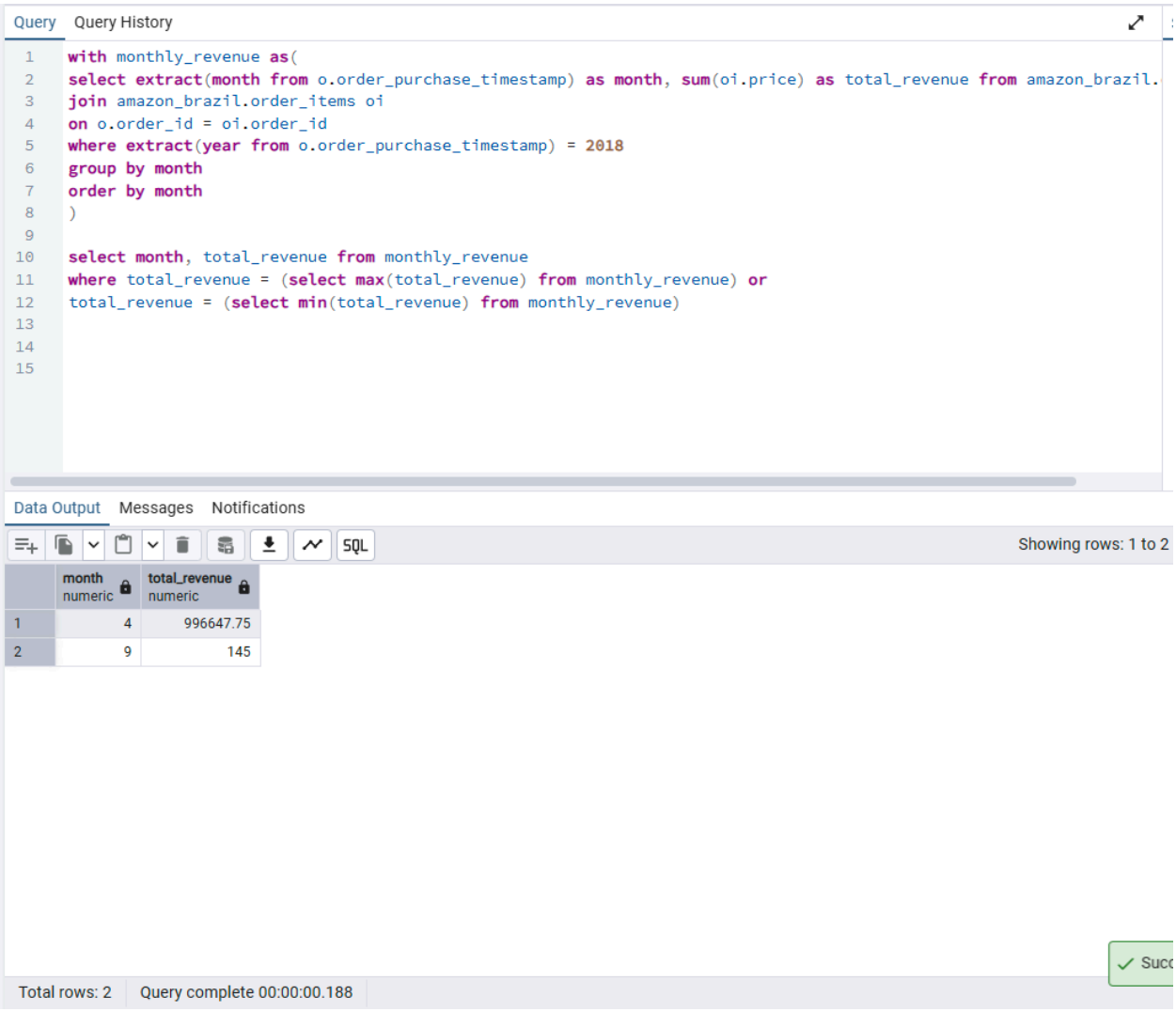
Showing rows: 1 to 9

	month numeric	total_revenue numeric
1	1	950030.36
2	2	844178.71
3	3	983213.44
4	4	996647.75
5	5	996517.68
6	6	865124.31
7	7	895507.22
8	8	854686.33
9	9	145

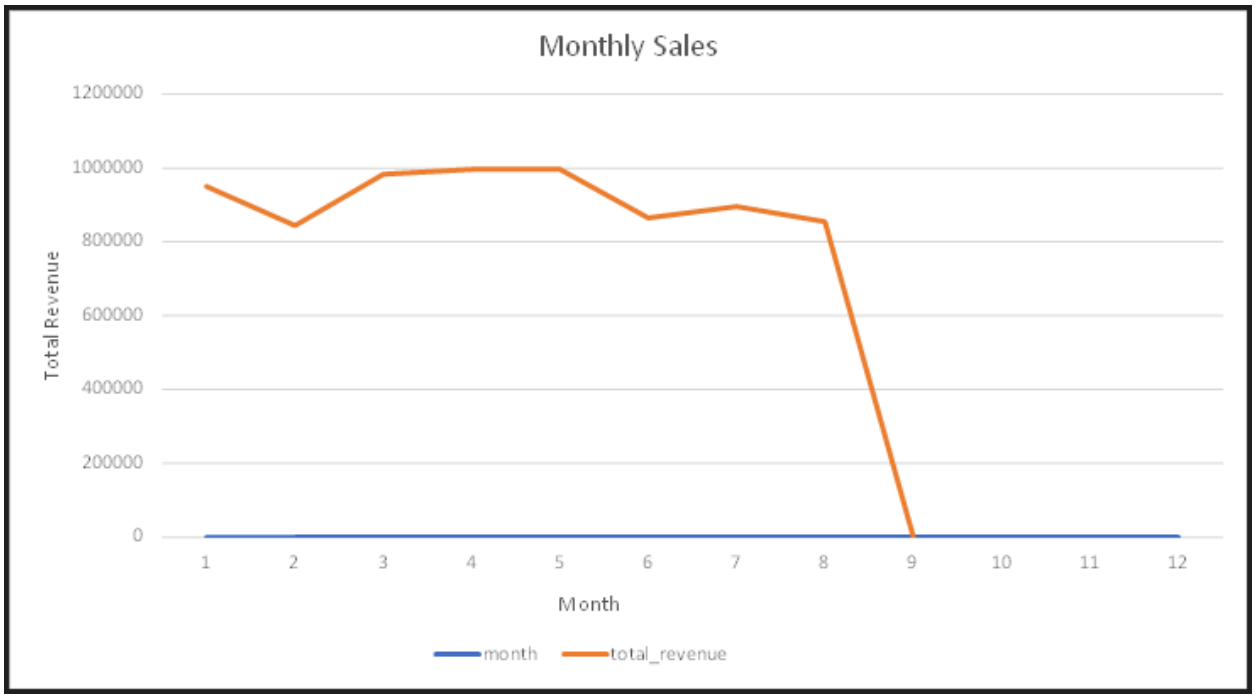
Total rows: 9 | Query complete 00:00:00.220

```
select extract(month from o.order_purchase_timestamp) as month, sum(oi.price) as
total_revenue from amazon_brazil.orders o
join amazon_brazil.order_items oi
on o.order_id = oi.order_id
where extract(year from o.order_purchase_timestamp) = 2018
group by month
order by month
```

3b)




```
with monthly_revenue as(
select extract(month from o.order_purchase_timestamp) as month, sum(oi.price) as
total_revenue from amazon_brazil.orders o
join amazon_brazil.order_items oi
on o.order_id = oi.order_id
where extract(year from o.order_purchase_timestamp) = 2018
group by month
order by month
)

select month, total_revenue from monthly_revenue
where total_revenue = (select max(total_revenue) from monthly_revenue) or
total_revenue = (select min(total_revenue) from monthly_revenue)
```



4)**A loyalty program is being designed for Amazon India.** Create a segmentation based on purchase frequency: ‘Occasional’ for customers with 1-2 orders, ‘Regular’ for 3-5 orders, and ‘Loyal’ for more than 5 orders. Use a CTE to classify customers and their count and generate a chart in Excel to show the proportion of each segment.

- **Output**: `customer_type, count`

Query | Query History

```
with customer_order_count as(
select c.customer_unique_id, count(distinct o.order_id) as order_count from amazon_brazil.customers c
left join amazon_brazil.orders o
on c.customer_id = o.customer_id
group by customer_unique_id
)

select
case
when order_count BETWEEN 1 AND 2 then 'Occasional'
when order_count between 3 and 5 then 'Regular'
when order_count > 5 then 'Loyal'
else 'No Orders'
end  as customer_type, count(*) as customer_count
from customer_order_count
group by customer_type
```

Data Output | Messages | Notifications

Showing rows: 1 to

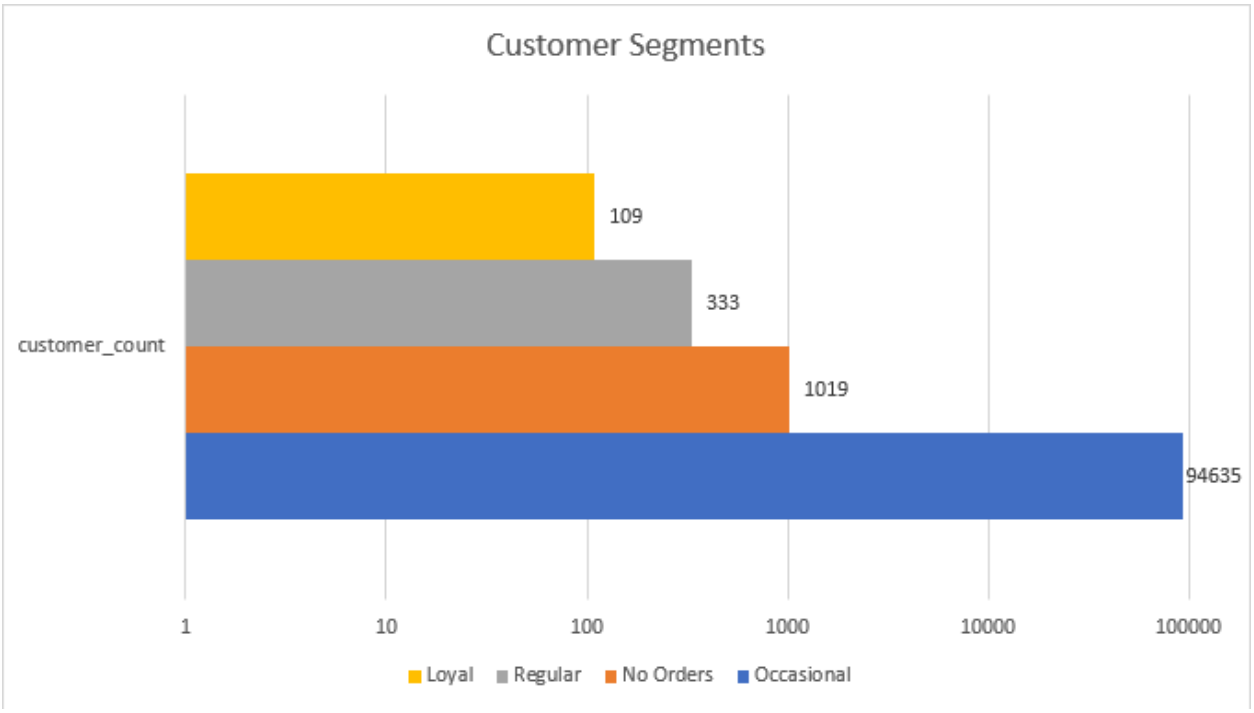
	customer_type text	customer_count bigint
1	Occasional	94635
2	No Orders	1019
3	Regular	333
4	Loyal	109

Total rows: 4 | Query complete 00:00:00.474

with customer_order_count as(
select c.customer_unique_id, count(distinct o.order_id) as order_count from
amazon_brazil.customers c
left join amazon_brazil.orders o
on c.customer_id = o.customer_id
group by customer_unique_id
)

select
case
when order_count BETWEEN 1 AND 2 then 'Occasional'
when order_count between 3 and 5 then 'Regular'
when order_count > 5 then 'Loyal'

```
else 'No Orders'
end  as customer_type, count(*) as customer_count
from customer_order_count
group by customer_type
```



5) **Amazon wants to identify high-value customers to target for an exclusive rewards program.** You are required to rank customers based on their average order value (avg_order_value) to find the top 20 customers.

- **Output:** `customer_id, avg_order_value, and customer_rank`

Query | Query History

```sql
WITH order_totals AS (
  SELECT
    o.order_id,
    o.customer_id,
    SUM(oi.price) AS order_value
  FROM amazon_brazil.orders o
  JOIN amazon_brazil.order_items oi ON o.order_id = oi.order_id
  GROUP BY o.order_id, o.customer_id
),
customer_avg_order_value AS (
  SELECT
    c.customer_unique_id,
    AVG(ot.order_value) AS avg_order_value
  FROM amazon_brazil.customers c
  JOIN order_totals ot ON c.customer_id = ot.customer_id
  GROUP BY c.customer_unique_id
)
SELECT
  customer_unique_id AS customer_id,
  avg_order_value,
  RANK() OVER (ORDER BY avg_order_value DESC) AS customer_rank
FROM customer_avg_order_value
ORDER BY customer_rank
LIMIT 20;
```

Data Output | Messages | Notifications

Showing rows: 1 to 2

	customer_id character varying	avg_order_value numeric	customer_rank bigint
1	0a0a92112bd4c708ca5fde585afaa8...	13440.0000000000000000	1
2	763c8b1c9c68a0229c42c9fc6f662b...	7160.0000000000000000	2
3	dc4802a71eae9be1dd28f5d788ceb5...	6735.0000000000000000	3
4	459bef486812aa25204be022145caa...	6729.0000000000000000	4
5	ff4159b92c40ebe40454e3e6a7c35e...	6499.0000000000000000	5
6	4007669dec559734d6f53e029e360...	5934.6000000000000000	6
7	eebb5dda148d3893cdaf5b5ca3040c...	4690.0000000000000000	7
8	5d0a2980b292d049061542014e896...	4599.9000000000000000	8
9	48e1ac109decbb87765a3eade6854...	4590.0000000000000000	9
10	a229eba70ec1c2abef51f04987deb7...	4400.0000000000000000	10

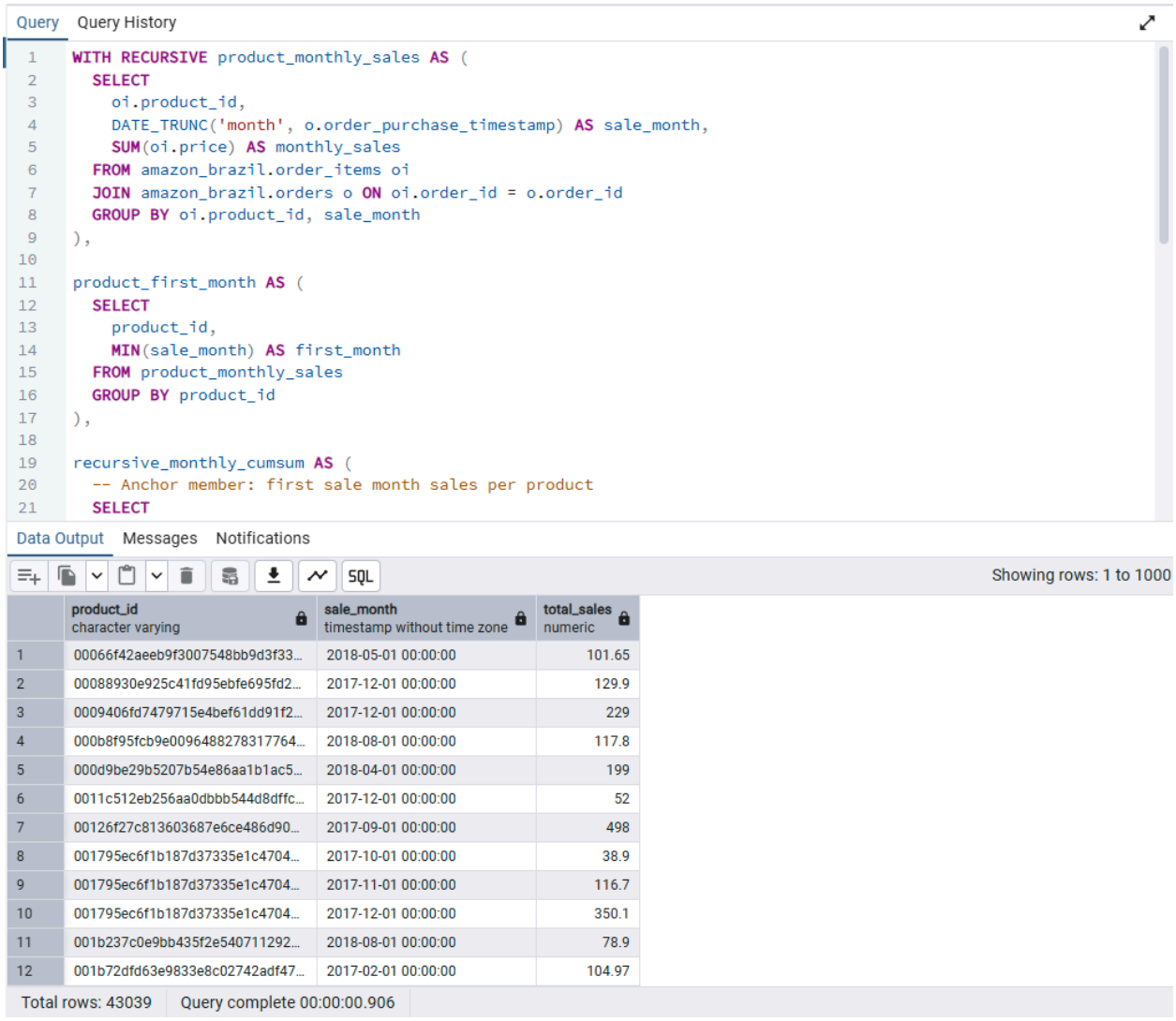
Total rows: 20 | Query complete 00:00:00.759


```sql
WITH order_totals AS (
  SELECT
        o.order_id,
        o.customer_id,
        SUM(oi.price) AS order_value
  FROM amazon_brazil.orders o
  JOIN amazon_brazil.order_items oi ON o.order_id = oi.order_id
  GROUP BY o.order_id, o.customer_id
),
customer_avg_order_value AS (
  SELECT
        c.customer_unique_id,
        AVG(ot.order_value) AS avg_order_value
```

```
  FROM amazon_brazil.customers c
  JOIN order_totals ot ON c.customer_id = ot.customer_id
  GROUP BY c.customer_unique_id
)
SELECT
  customer_unique_id AS customer_id,
  avg_order_value,
  RANK() OVER (ORDER BY avg_order_value DESC) AS customer_rank
FROM customer_avg_order_value
ORDER BY customer_rank
LIMIT 20;
```

6) **Amazon wants to analyze sales growth trends for its key products over their lifecycle.** Calculate monthly cumulative sales for each product from the date of its first sale. Use a recursive CTE to compute the cumulative sales (total_sales) for each product month by month.

- Output: `product_id, sale_month, and total_sales`



Query | Query History

```
WITH RECURSIVE product_monthly_sales AS (
  SELECT
    oi.product_id,
    DATE_TRUNC('month', o.order_purchase_timestamp) AS sale_month,
    SUM(oi.price) AS monthly_sales
  FROM amazon_brazil.order_items oi
  JOIN amazon_brazil.orders o ON oi.order_id = o.order_id
  GROUP BY oi.product_id, sale_month
),

product_first_month AS (
  SELECT
    product_id,
    MIN(sale_month) AS first_month
  FROM product_monthly_sales
  GROUP BY product_id
),

recursive_monthly_cumsum AS (
  -- Anchor member: first sale month sales per product
  SELECT
```

Data Output | Messages | Notifications

Showing rows: 1 to 1000

	product_id character varying	sale_month timestamp without time zone	total_sales numeric
1	00066f42aeeb9f3007548bb9d3f33...	2018-05-01 00:00:00	101.65
2	00088930e925c41fd95ebfe695fd2...	2017-12-01 00:00:00	129.9
3	0009406fd7479715e4bef61dd91f2...	2017-12-01 00:00:00	229
4	000b8f95fcb9e0096488278317764...	2018-08-01 00:00:00	117.8
5	000d9be29b5207b54e86aa1b1ac5...	2018-04-01 00:00:00	199
6	0011c512eb256aa0dbbb544d8dffc...	2017-12-01 00:00:00	52
7	00126f27c813603687e6ce486d90...	2017-09-01 00:00:00	498
8	001795ec6f1b187d37335e1c4704...	2017-10-01 00:00:00	38.9
9	001795ec6f1b187d37335e1c4704...	2017-11-01 00:00:00	116.7
10	001795ec6f1b187d37335e1c4704...	2017-12-01 00:00:00	350.1
11	001b237c0e9bb435f2e540711292...	2018-08-01 00:00:00	78.9
12	001b72dfd63e9833e8c02742adf47...	2017-02-01 00:00:00	104.97

Total rows: 43039 | Query complete 00:00:00.906

```
WITH RECURSIVE product_monthly_sales AS (
  SELECT
        oi.product_id,
        DATE_TRUNC('month', o.order_purchase_timestamp) AS sale_month,
        SUM(oi.price) AS monthly_sales
  FROM amazon_brazil.order_items oi
  JOIN amazon_brazil.orders o ON oi.order_id = o.order_id
  GROUP BY oi.product_id, sale_month
),

product_first_month AS (
  SELECT
        product_id,
        MIN(sale_month) AS first_month
```

```
  FROM product_monthly_sales
  GROUP BY product_id
),

recursive_monthly_cumsum AS (
  -- Anchor member: first sale month sales per product
  SELECT
        pms.product_id,
        pms.sale_month,
        pms.monthly_sales AS total_sales
  FROM product_monthly_sales pms
  JOIN product_first_month pfm
        ON pms.product_id = pfm.product_id
        AND pms.sale_month = pfm.first_month

  UNION ALL

  -- Recursive member: cumulative sales by adding next month sales
  SELECT
        pms.product_id,
        pms.sale_month,
        rms.total_sales + pms.monthly_sales AS total_sales
  FROM recursive_monthly_cumsum rms
  JOIN product_monthly_sales pms
        ON rms.product_id = pms.product_id
        AND pms.sale_month = rms.sale_month + INTERVAL '1 month'
)

SELECT
  product_id,
  sale_month,
  total_sales
FROM recursive_monthly_cumsum
ORDER BY product_id, sale_month;
```

7) **To understand how different payment methods affect monthly sales growth, Amazon wants to compute the total sales for each payment method and calculate the month-over-month growth rate for the past year (year 2018).** Write query to first calculate total monthly sales for each payment method, then compute the percentage change from the previous month.

- Output: `payment_type, sale_month, monthly_total, monthly_change.`

Query | Query History

```
WITH monthly_sales AS (
  SELECT
    p.payment_type,
    DATE_TRUNC('month', o.order_purchase_timestamp) AS sale_month,
    SUM(p.payment_value) AS monthly_total
  FROM amazon_brazil.payments p
  JOIN amazon_brazil.orders o ON p.order_id = o.order_id
  WHERE EXTRACT(YEAR FROM o.order_purchase_timestamp) = 2018
  GROUP BY p.payment_type, sale_month
),
monthly_sales_with_lag AS (
  SELECT
    payment_type,
    sale_month,
    monthly_total,
    LAG(monthly_total) OVER (PARTITION BY payment_type ORDER BY sale_month) AS prev_month_total
  FROM monthly_sales
)
SELECT
  payment_type,
  sale_month,
```

Data Output | Messages | Notifications

SQL | Showing rows: 1 to

	payment_type character varying	sale_month timestamp without time zone	monthly_total numeric	monthly_change numeric
1	boleto	2018-01-01 00:00:00	204844.66	[null]
2	boleto	2018-02-01 00:00:00	183112.72	-10.60898536481253648500
3	boleto	2018-03-01 00:00:00	191538.02	4.60115496072583051600
4	boleto	2018-04-01 00:00:00	193547.09	1.04891446617230354600
5	boleto	2018-05-01 00:00:00	195378.93	0.94645700950605870600
6	boleto	2018-06-01 00:00:00	153350.28	-21.51135232442925140400
7	boleto	2018-07-01 00:00:00	198041.24	29.14305731949103712100
8	boleto	2018-08-01 00:00:00	143805.90	-27.38588184965919219700
9	credit_card	2018-01-01 00:00:00	868880.38	[null]
10	credit_card	2018-02-01 00:00:00	778803.00	-10.36706341556475242300
11	credit_card	2018-03-01 00:00:00	933770.10	19.89811287321697528100
12	credit_card	2018-04-01 00:00:00	934306.00	0.05739100020444004400

Total rows: 36 | Query complete 00:00:00.186

```
WITH monthly_sales AS (
  SELECT
        p.payment_type,
        DATE_TRUNC('month', o.order_purchase_timestamp) AS sale_month,
        SUM(p.payment_value) AS monthly_total
  FROM amazon_brazil.payments p
  JOIN amazon_brazil.orders o ON p.order_id = o.order_id
  WHERE EXTRACT(YEAR FROM o.order_purchase_timestamp) = 2018
  GROUP BY p.payment_type, sale_month
),
monthly_sales_with_lag AS (
  SELECT
        payment_type,
```

```
        sale_month,
        monthly_total,
        LAG(monthly_total) OVER (PARTITION BY payment_type ORDER BY
sale_month) AS prev_month_total
  FROM monthly_sales
)
SELECT
  payment_type,
  sale_month,
  monthly_total,
  CASE
        WHEN prev_month_total IS NULL OR prev_month_total = 0 THEN NULL
        ELSE ((monthly_total - prev_month_total) / prev_month_total) * 100
  END AS monthly_change
FROM monthly_sales_with_lag
ORDER BY payment_type, sale_month;
```