

Analysis - I

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.
 - o **Output:** payment_type, rounded_avg_payment

1. ANSWER:

Object Explorer amazon_analysis/postgres@PostgreSQL 17* x

```
--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 COALESCE(payment_type, 'N/A') AS payment_type,
       ROUND(AVG(payment_value)) AS rounded_avg_payment
FROM Payments
WHERE payment_type != 'not_defined'
      AND payment_value IS NOT NULL
GROUP BY payment_type
ORDER BY rounded_avg_payment DESC;
```

Data Output Messages Notifications

payment_type	rounded_avg_payment
credit_card	163
boleto	145
debit_card	143
voucher	66

Ques2.) 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

- o **Output:** payment_type, percentage_orders

Object Explorer amazon_analysis/postgres@PostgreSQL 17* x

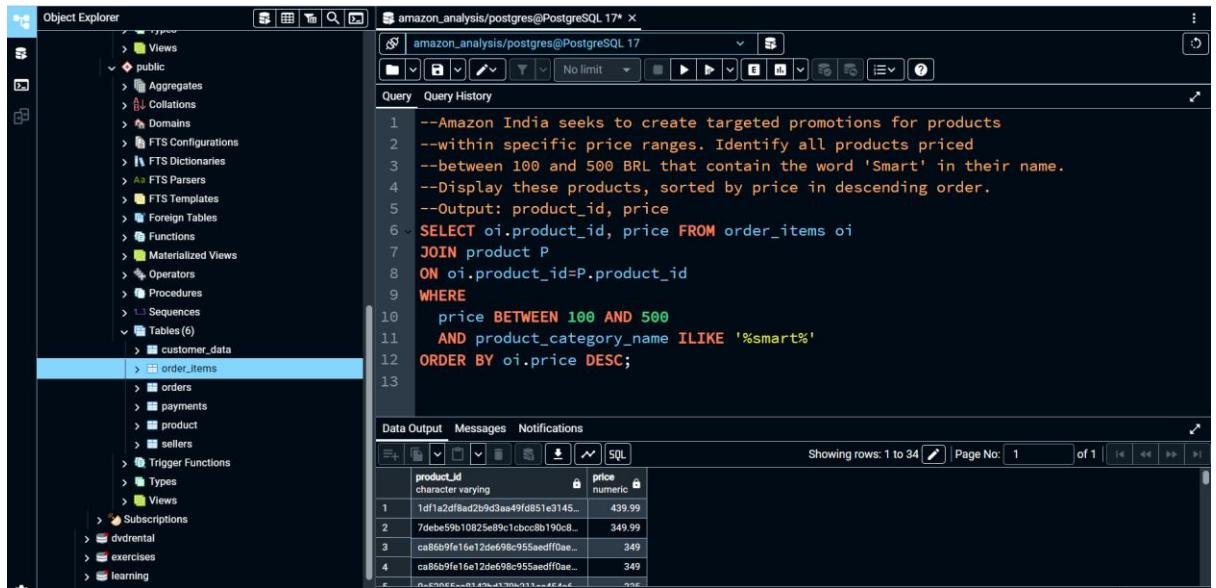
```
--To refine its payment strategy, Amazon India wants to know the distribution
--of orders by payment type. Calculate the percentage of total orders
--for each payment type, rounded to one decimal place, and display them
--in descending order
--Output: payment_type, percentage_orders
SELECT payment_type,
       ROUND((COUNT(*) *100.0)/
             (SELECT COUNT(*) FROM Payments), 1) AS percentage_orders
FROM Payments
WHERE payment_type != 'not_defined'
      AND payment_value IS NOT NULL
GROUP BY payment_type
ORDER BY percentage_orders DESC;
```

Data Output Messages Notifications

payment_type	percentage_orders
credit_card	73.9
boleto	19.0
voucher	5.6
debit_card	1.5

QUES3) 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



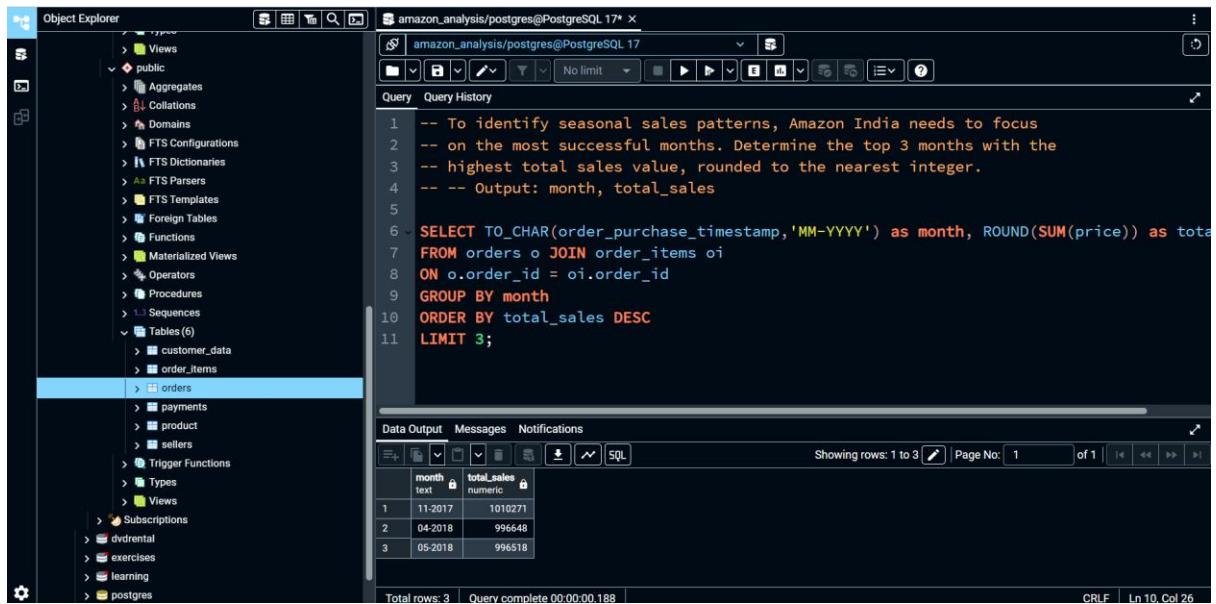
```
--Amazon India seeks to create targeted promotions for products
--within specific price ranges. Identify all products priced
--between 100 and 500 BRL that contain the word 'Smart' in their name.
--Display these products, sorted by price in descending order.
--Output: product_id, price

SELECT oi.product_id, price FROM order_items oi
JOIN product P
ON oi.product_id=P.product_id
WHERE
    price BETWEEN 100 AND 500
    AND product_category_name ILIKE '%smart%'
ORDER BY oi.price DESC;
```

product_id	price
1df1a2df8ad2bb3daa49fd851e3145...	439.99
7dbe59b10825e99c1bcc8b190c8...	349.99
ca86b9f9f16e12de698c955aeddff0ae...	349
ca86b9f9f16e12de698c955aeddff0ae...	349
ca86b9f9f16e12de698c955aeddff0ae...	349

QUES4) 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



```
-- 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 TO_CHAR(order_purchase_timestamp,'MM-YYYY') as month, ROUND(SUM(price)) as total_sales
FROM orders o JOIN order_items oi
ON o.order_id = oi.order_id
GROUP BY month
ORDER BY total_sales DESC
LIMIT 3;
```

month	total_sales
11-2017	1010271
04-2018	996648
05-2018	996518



QUES5) 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

The screenshot shows the pgAdmin interface with the Object Explorer on the left and the Query Editor on the right. The Object Explorer displays various database objects like Views, Aggregates, Domains, FTS Configurations, etc. The Query Editor contains the following SQL code:

```
-- 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
SELECT product_category_name,
       (MAX(price) - MIN(price)) AS price_difference
FROM product p JOIN order_items oi
ON p.product_id = oi.product_id
GROUP BY product_category_name
HAVING (MAX(price) - MIN(price)) > 500
ORDER BY price_difference DESC;
```

The Data Output tab shows the results of the query:

product_category_name	price_difference
utilidades_domesticas	6731.94
pos	6694.5
artes	6495.5
electroportateis	4792.5
instrumentos	4204.07

Total rows: 57 | Query complete 00:00:00.143 | CRLF | Ln 11, Col 32

QUES6) 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_deviatiion

The screenshot shows the pgAdmin interface with the Object Explorer on the left and the Query Editor on the right. The Object Explorer displays various database objects like Views, Aggregates, Domains, FTS Configurations, etc. The Query Editor contains the following SQL code:

```
-- 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, ROUND(STDDEV(payment_value),2) as std_deviatiion
SELECT payment_type, ROUND(STDDEV(payment_value), 2) AS std_deviatiion
FROM payments
GROUP BY payment_type
ORDER BY std_deviatiion ASC;
```

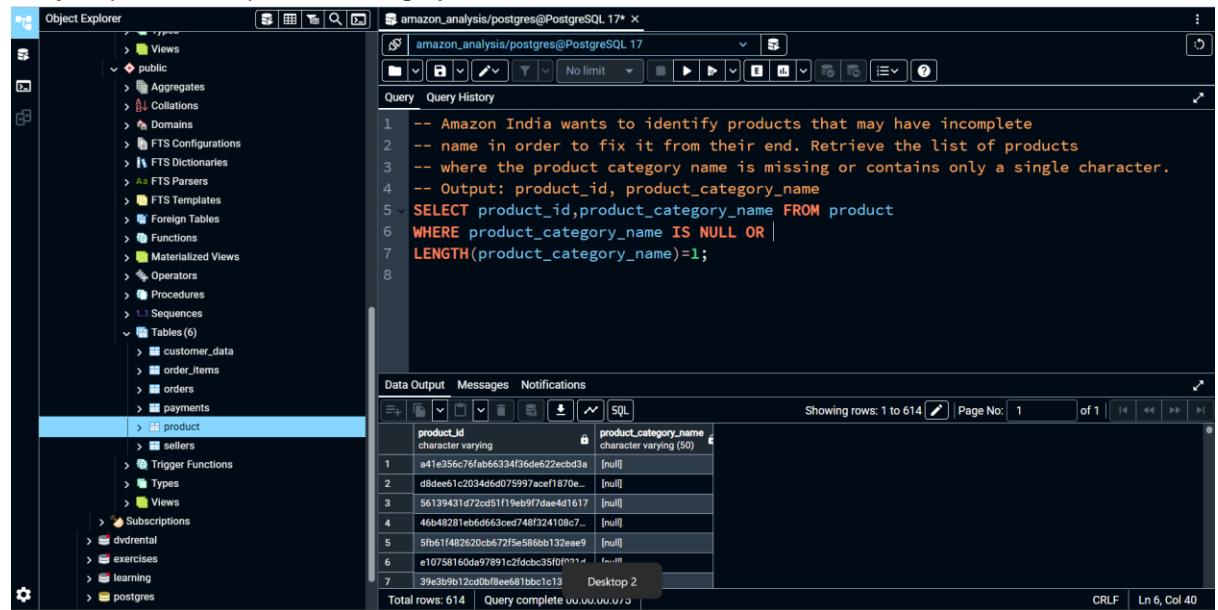
The Data Output tab shows the results of the query:

payment_type	std_deviatiion
not_defined	0.00
voucher	115.52
boleto	213.58
credit_card	222.12
debit_card	245.79

Total rows: 5 | Query complete 00:00:00.551 | CRLF | Ln 9, Col 28

QUES7) 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



```

-- 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
SELECT product_id,product_category_name FROM product
WHERE product_category_name IS NULL OR
LENGTH(product_category_name)=1;

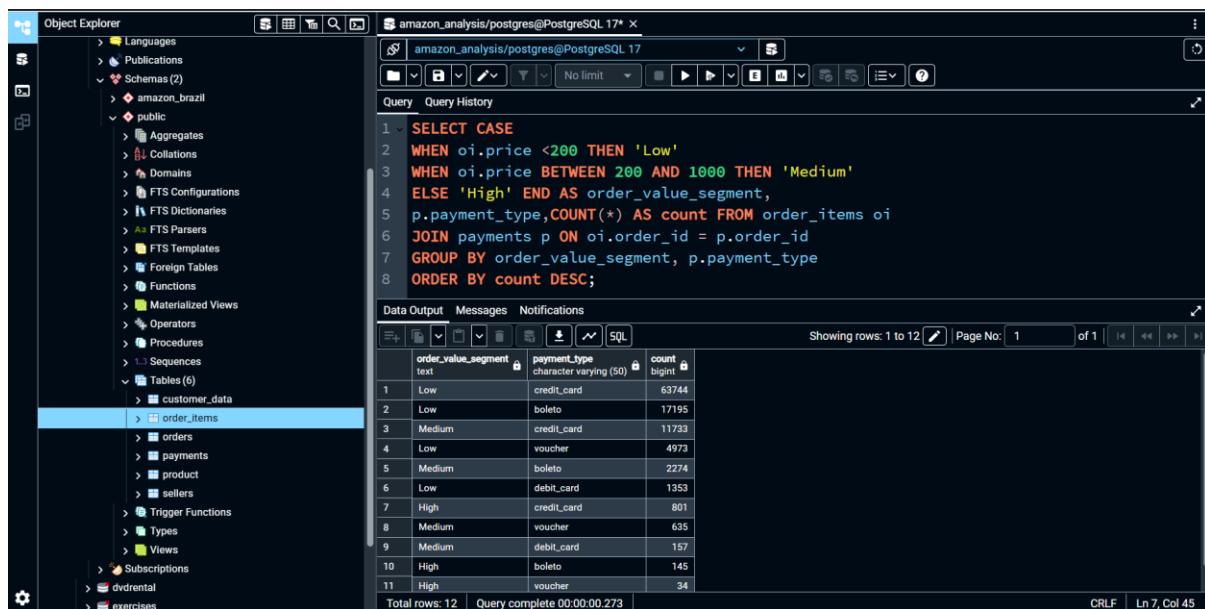
```

product_id	product_category_name
a41e356c76fab66334f36de622cbcd3a	[null]
d8deef1c2034d6d07599acef1870...	[null]
56139431d72cd5119eb97d4ae4d1617	[null]
46648281eb6d663ced748f324108c7...	[null]
5fb61f482620cb672f5e586bb132ee9	[null]
e10758160d97891c2fdabc350f0714...	[null]
39e3b9b12cd0bf8e681b0c1c13	Desktop 2

Analysis - II

QUES1) 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



The screenshot shows the pgAdmin 4 interface with a PostgreSQL connection named 'amazon_analysis/postgres@PostgreSQL 17*'. The left pane is the Object Explorer, displaying various database objects like Schemas, Tables, and Functions. The right pane shows the SQL editor with the following query:

```

1. SELECT CASE
2. WHEN oi.price < 200 THEN 'Low'
3. WHEN oi.price BETWEEN 200 AND 1000 THEN 'Medium'
4. ELSE 'High' END AS order_value_segment,
5. p.payment_type,COUNT(*) AS count FROM order_items oi
6. JOIN payments p ON oi.order_id = p.order_id
7. GROUP BY order_value_segment, p.payment_type
8. ORDER BY count DESC;

```

The Data Output tab displays the results of the query:

order_value_segment	payment_type	count
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

Total rows: 12 | Query complete 00:00:00.273 | CRLF | Ln 7, Col 45

QUES2) 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

```

Object Explorer      amazon_analysis/postgres@PostgreSQL 17* X
amazon_brazil
  public
    Aggregates
    Collations
    Domains
    FTS Configurations
    FTS Dictionaries
    FTS Parsers
    FTS Templates
    Foreign Tables
    Functions
    Materialized Views
    Operators
    Procedures
    Sequences
    Tables (6)
      customer_data
      order_items
      orders
      payments
      product
      sellers
    Trigger Functions
    Types
    Views
  Subscriptions
  dvrental
  exercises
  learning
  postgres
  school

Query  Query History
1. SELECT p.product_category_name, MIN(oi.price) AS min_price,
2. MAX(oi.price) AS max_price,
3. ROUND(AVG(oi.price), 2) AS avg_price
4. FROM product p JOIN order_items oi
5. ON p.product_id = oi.product_id
6. GROUP BY p.product_category_name
7. ORDER BY avg_price DESC;

Data Output  Messages  Notifications
Showing rows: 1 to 79 | Page No: 1 of 1 | < << > >>
product_category_name | min_price | max_price | avg_price |
pcos | 34.5 | 6729 | 1098.34 |
portateis_casa_forno_e_cafe | 10.19 | 2899 | 624.29 |
eletronicos_2 | 13.9 | 2350 | 476.12 |
agro_industria_e_comercio | 12.99 | 2990 | 341.66 |
instrumentos_musicais | 4.9 | 4399.87 | 281.62 |
eletroportateis | 6.5 | 4799 | 280.78 |
portateis_cozinha_e_preparadores_de_alimentos | 17.42 | 1099 | 264.57 |
telefonia_fixa | 6 | 1790 | 225.69 |
construcao_ferramentas_seguranca | 8.9 | 3099.9 | 208.99 |
relogios_presentes | 8.99 | 3999.9 | 200.91 |
climatizacao | 10.9 | 1599 | 185.27 |

Total rows: 79 | Query complete 00:00:00.238 | CRLF | Ln 3, Col 13

```

QUES3) 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

```

Object Explorer      amazon_analysis/postgres@PostgreSQL 17* X
amazon_brazil
  public
    Aggregates
    Collations
    Domains
    FTS Configurations
    FTS Dictionaries
    FTS Parsers
    FTS Templates
    Foreign Tables
    Functions
    Materialized Views
    Operators
    Procedures
    Sequences
    Tables (6)
      customer_data
        order_items
        orders
        payments
        product
        sellers
      Trigger Functions
      Types
      Views
    Subscriptions
    dvrental
    exercises
    learning
    postgres
    school

Query  Query History
1. SELECT cd.customer_unique_id,
2. COUNT(order_id) AS total_orders
3. FROM orders o
4. JOIN customer_data cd
5. ON cd.customer_id=o.customer_id
6. GROUP BY cd.customer_unique_id
7. HAVING COUNT(order_id) > 1
8. ORDER BY total_orders DESC;

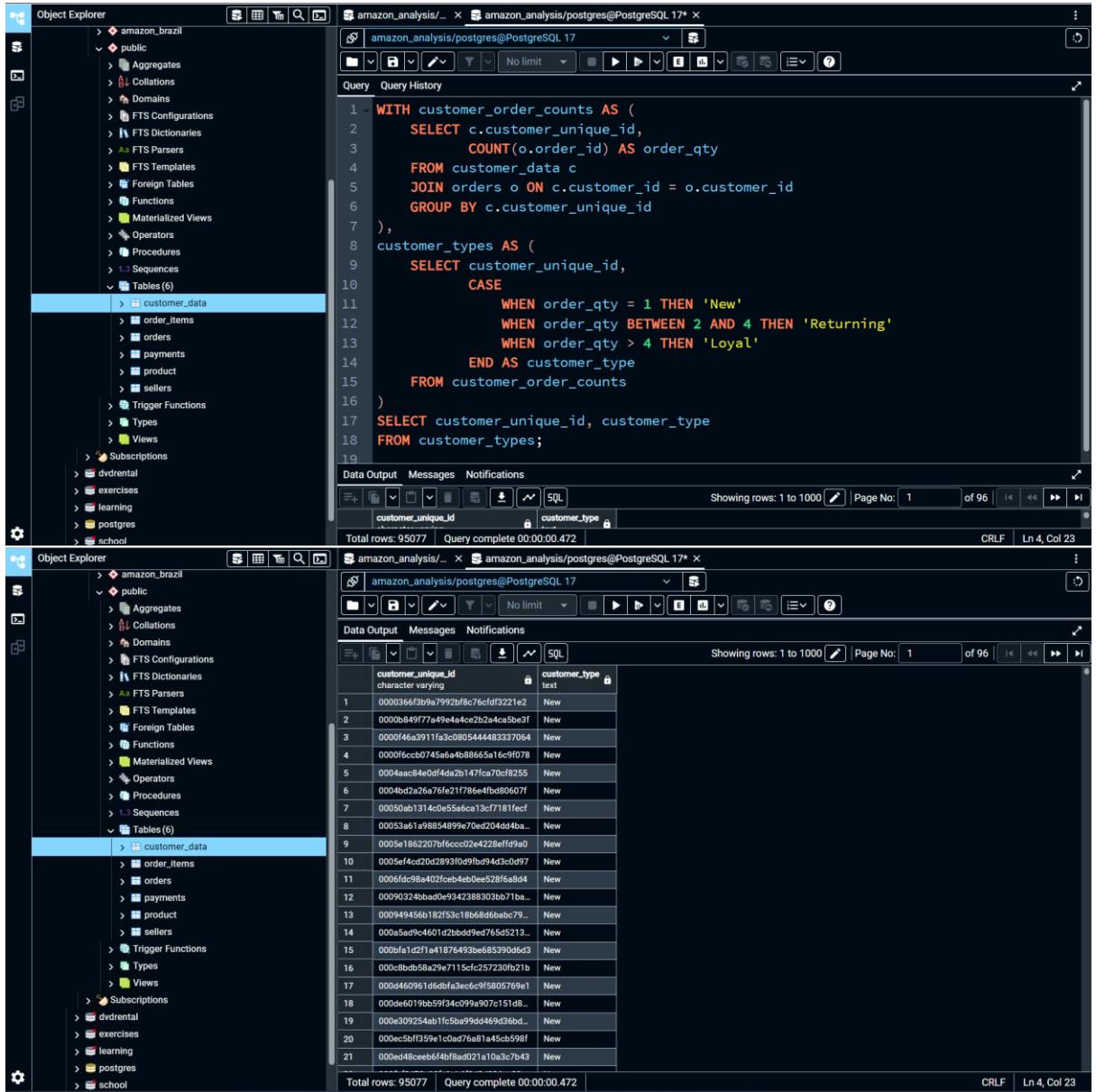
Data Output  Messages  Notifications
Showing rows: 1 to 1000 | Page No: 1 of 4 | < << > >>
customer_unique_id | total_orders |
a91e8fb80ddc07de66a5cf9270293c | 16 |
a6168cd79131e64ace92e3e74dfcc43 | 16 |
3639fc80585bf04c1a88fd986011c52e | 16 |
cbd03504ccba9772eae768d4aa45c... | 16 |
417b909c0962b2610f1cfec1c1478986 | 16 |
5f94af52aaef02e96a2e0f01430864e | 16 |
1b6d29725255a77667a8c639eeb4cc... | 16 |
e4bbc533fd3f917c56deea2c43bf2084 | 16 |
930c4390af58f6734447c3a1cb2a36 | 16 |
5bf4ea2d98005b960ea0dbf652e4e7 | 16 |
9159c04688895d95741dd59b7a5f... | 16 |

Total rows: 3140 | Query complete 00:00:00.484 | CRLF | Ln 7, Col 28

```

QUES4) 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

1. 

```

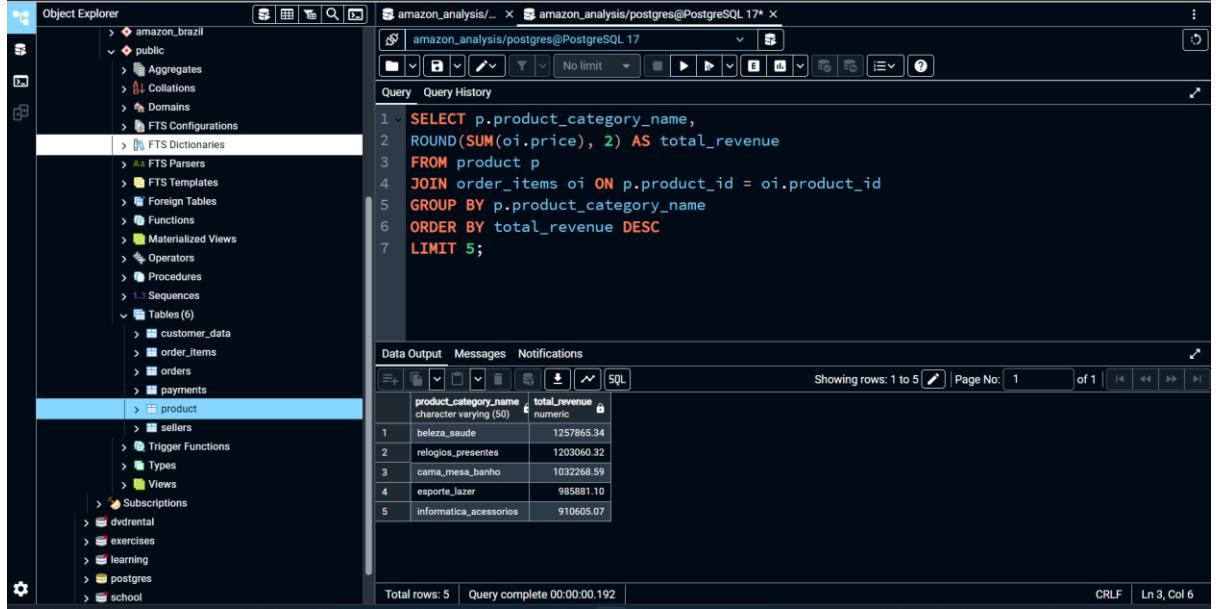
WITH customer_order_counts AS (
    SELECT c.customer_unique_id,
           COUNT(o.order_id) AS order_qty
    FROM customer_data c
    JOIN orders o ON c.customer_id = o.customer_id
    GROUP BY c.customer_unique_id
),
customer_types AS (
    SELECT customer_unique_id,
          CASE
              WHEN order_qty = 1 THEN 'New'
              WHEN order_qty BETWEEN 2 AND 4 THEN 'Returning'
              WHEN order_qty > 4 THEN 'Loyal'
          END AS customer_type
    FROM customer_order_counts
)
SELECT customer_unique_id, customer_type
FROM customer_types;

```

customer_unique_id	customer_type
0000364fb9e7992bf8c76cfdf3221e2	New
0000849f77a49e4ace2b2a4ca5be2f	New
0000f46a3911fa3c0805444483337064	New
0000f6ccb0745a64a88665a1fc9f079	New
0004aac84e0d4da2b147fc70cf8255	New
0004bd2a26a76fe21f78e4fb806077	New
00050ab1314ce05a5aca13cf7181fec1	New
00053a61a98854899e70ed204dd4ba_	New
0005e1862207bfcc0c024228affd9af	New
0005ef4cd20d2893fd9fb4d3cd97	New
0006fd98a402fc4eb0ee528f58d4	New
0009324bbad0e934238303b71ba_	New
000994956b182f53c18b66d6abc79..	New
000a5ad9c460d2bbddfed765d213	New
000bfa12f1a4187493be685390dd6	New
000c8b0b58a29e7115cf257230fb21b	New
000d460961d5dbfa3ec69f5805769e1	New
000de6019bb59f34c099a907c151d8..	New
000e309254b1f1c5b899dd469363bd..	New
000ec5bf359e1c0ad76a81a45cb598f	New
000ed48ceeb6f4bf8a0d21a10a3c7b43	New

QUES5)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



The screenshot shows the pgAdmin 4 interface. On the left, the Object Explorer tree view displays the schema structure of the 'amazon_brazil' database, including FTS Dictionaries, Tables (customer_data, order_items, orders, payments, product, sellers), and other objects like triggers, functions, and types. The 'product' table is selected, highlighted with a blue border. In the center, the Query Editor window contains a SQL query to calculate total revenue by product category. The Data Output tab on the right shows the results of the query, listing five categories with their respective total revenues.

```

SELECT p.product_category_name,
       ROUND(SUM(oi.price), 2) AS total_revenue
  FROM product p
  JOIN order_items oi ON p.product_id = oi.product_id
 GROUP BY p.product_category_name
 ORDER BY total_revenue DESC
 LIMIT 5;

```

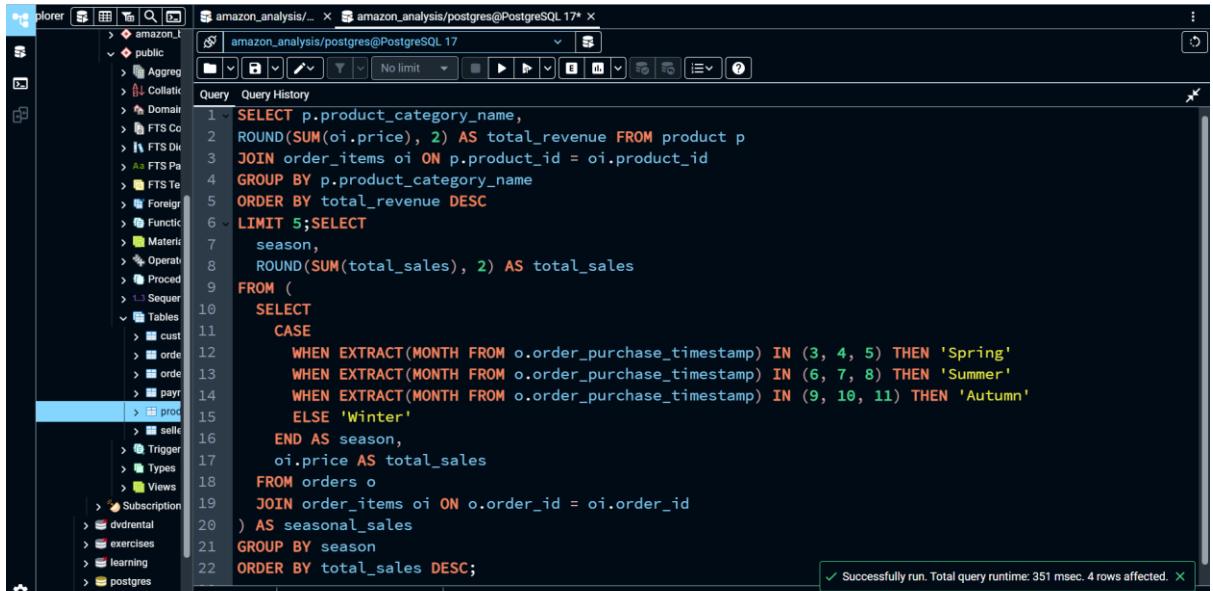
product_category_name	total_revenue
beleza_saude	1257865.34
relogios_presentes	1203060.32
cama_mesa_banho	1032268.59
esporte_lazer	985881.10
informatica_acessories	910605.07

Analysis - III

QUES1)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

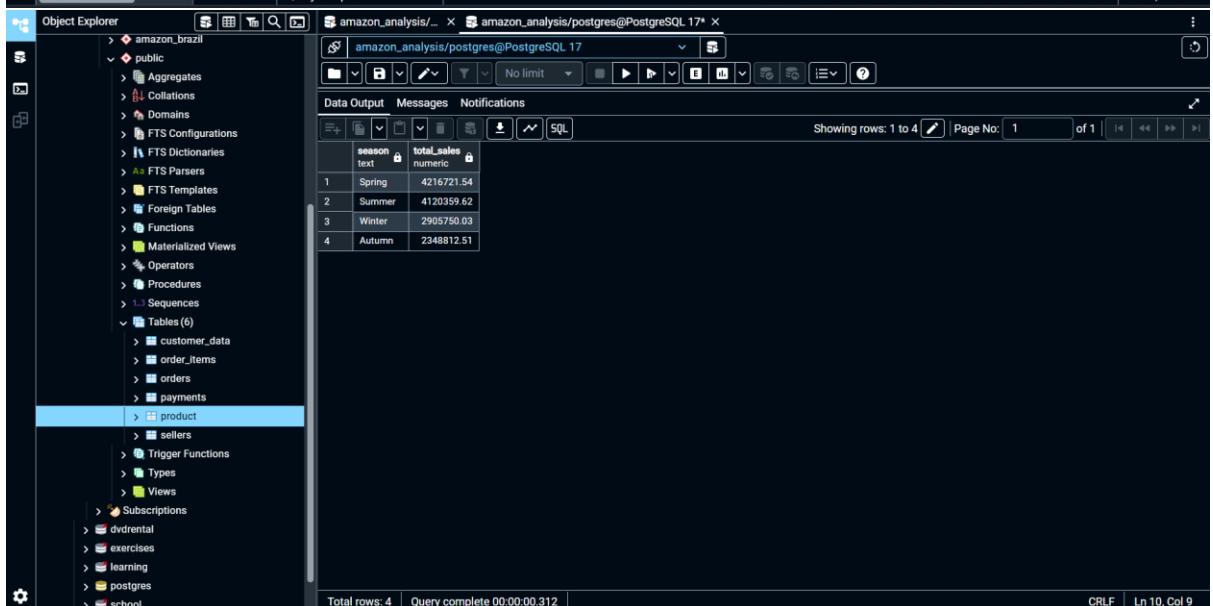
1.



```

SELECT p.product_category_name,
       ROUND(SUM(oi.price), 2) AS total_revenue
  FROM product p
  JOIN order_items oi ON p.product_id = oi.product_id
 GROUP BY p.product_category_name
 ORDER BY total_revenue DESC
 LIMIT 5;
SELECT
    season,
    ROUND(SUM(total_sales), 2) AS total_sales
  FROM (
    SELECT
      CASE
        WHEN EXTRACT(MONTH FROM o.order_purchase_timestamp) IN (3, 4, 5) THEN 'Spring'
        WHEN EXTRACT(MONTH FROM o.order_purchase_timestamp) IN (6, 7, 8) THEN 'Summer'
        WHEN EXTRACT(MONTH FROM o.order_purchase_timestamp) IN (9, 10, 11) THEN 'Autumn'
        ELSE 'Winter'
      END AS season,
      oi.price AS total_sales
    FROM orders o
    JOIN order_items oi ON o.order_id = oi.order_id
  ) AS seasonal_sales
 GROUP BY season
 ORDER BY total_sales DESC;
  
```

Total rows: 4 | Query complete 00:00:00.351 | ✓ Successfully run. Total query runtime: 351 msec. 4 rows affected. CRLF | Ln 16, Col 19



season	total_sales
Spring	4216721.54
Summer	4120359.62
Winter	2905750.03
Autumn	2348812.51

Total rows: 4 | Query complete 00:00:00.312 | CRLF | Ln 10, Col 9



QUES2)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

The screenshot shows the pgAdmin 4 interface. The left pane, titled 'Object Explorer', displays a tree view of database objects under 'amazon_brazil'. The 'Tables' node is expanded, showing 'customer_data', 'order_items', 'orders', 'payments', and 'product'. The 'product' table is currently selected, highlighted with a blue background. The right pane, titled 'amazon_analysis/postgres@PostgreSQL 17*', contains a SQL query editor with the following code:

```
1 SELECT product_id, total_quantity_sold
2 FROM (
3     SELECT
4         product_id,
5         COUNT(*) AS total_quantity_sold
6     FROM order_items
7     GROUP BY product_id
8 ) AS product_sales
9 WHERE total_quantity_sold > (
10     SELECT
11         AVG(product_count)
12     FROM (
13         SELECT
14             COUNT(*) AS product_count
15         FROM order_items
16         GROUP BY product_id
17     ) AS avg_sales
18 )
19 ORDER BY total_quantity_sold DESC;
```

The status bar at the bottom indicates 'Total rows: 6366' and 'Query complete 00:00:00.131'.

Object Explorer

- > amazon_brazil
 - > public
 - > Aggregates
 - > Collations
 - > Domains
 - > FTS Configurations
 - > FTS Dictionaries
 - > FTS Parsers
 - > FTS Templates
 - > Foreign Tables
 - > Functions
 - > Materialized Views
 - > Operators
 - > Procedures
 - > Sequences
 - > Tables (6)
 - > customer_data
 - > order_items
 - > orders
 - > payments
 - > product
 - > sellers
 - > Trigger Functions
 - > Types
 - > Views
 - > Subscriptions
 - > dvrental
 - > exercises
 - > learning
 - > postgres
 - > school

amazon_analysis/postgres@PostgreSQL 17+ x

Data Output Messages Notifications

	product_id	character varying	total_quantity_sold	bigint
1	aca2eb7d00ea1eb8ebd4e68314663...		527	
2	9944788cb2485695c36a24e39960...		488	
3	422879e1046682990de24d770e78...		484	
4	389d119b4cf3043d31133e4999c...		392	
5	368cc5c3084247d016ad823897e37...		388	
6	53759a2ecddad2b887a079a1f1519f...		373	
7	d1c427060a0f73fb8995c7c12f2ac4...		343	
8	53b36df67eb7c41585e8d54d6772e...		323	
9	154e731ebfa092203795c972e5804...		281	
10	3dd2a17168ec895c781a9191c1e95a...		274	
11	2b4609f8948be18874494203496c3...		260	
12	7c1b920ddbf2247068bde975d3c...		231	
13	a62e25e09e05ef6faf31d90c6ec1aa3d1...		226	
14	5a848e4ab52fd545cd07ab1c40e...		197	
15	bb50f2e236e5ee0010068013765468...		195	
16	e0d64dcfa3b3db5c5c4ca298a101d...		194	
17	42a2c92a0979a949ca4ea89ec5c7b9...		183	
18	e53e557d5a159f5aa1c5e995df2d44b...		183	
19	b532349fe46b38fbc5b3914c1bdae...		169	
20	35af973633a8eb5877ff5f62793310...		165	
21	a92930c327948861c015c919a0bc4...		160	

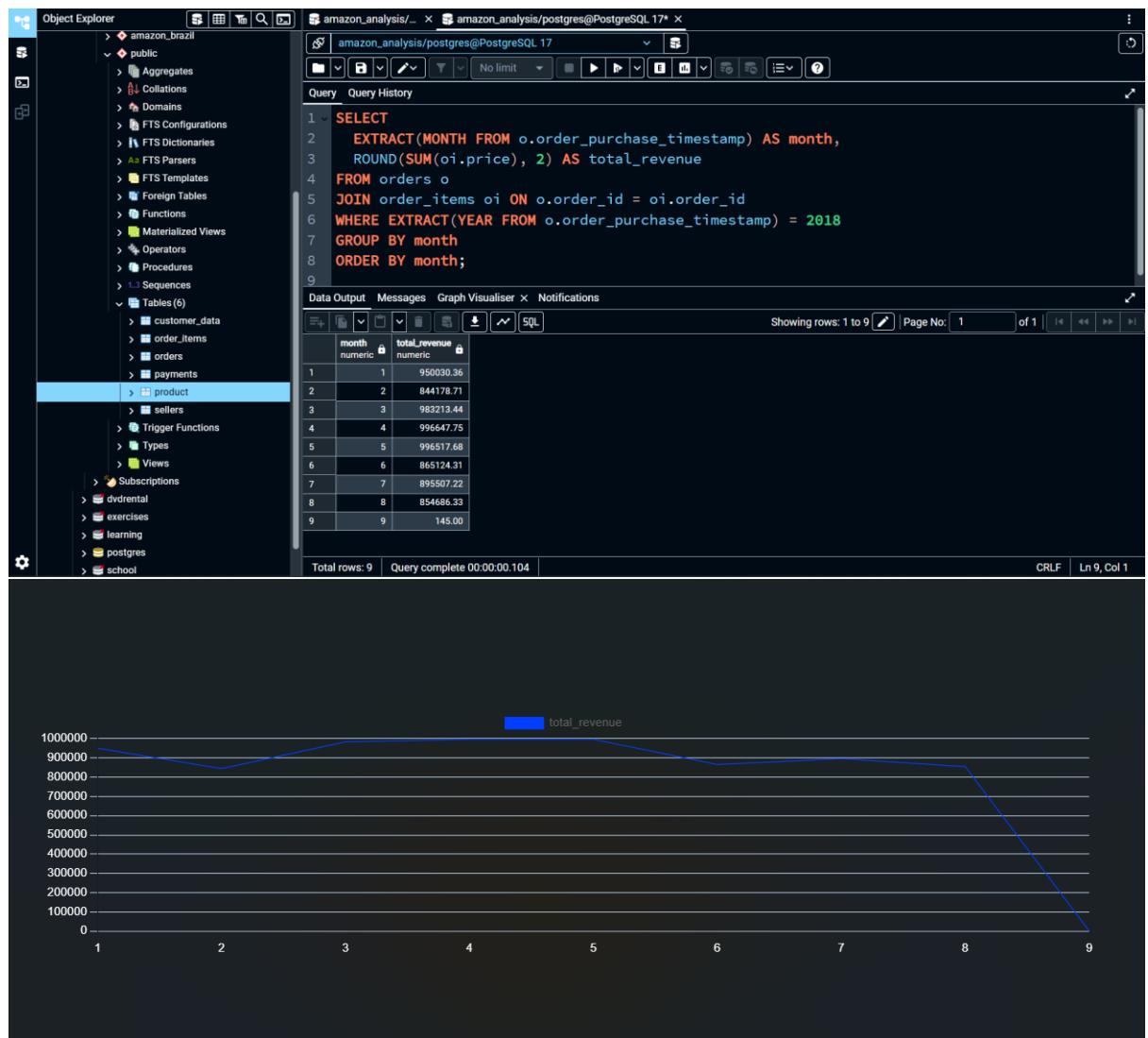
Total rows: 6366 | Query complete 00:00:00.131

Showing rows: 1 to 1000 | Page No: 1 of 7 | < << > >> >

QUES3) 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

1.



The screenshot shows a PostgreSQL database interface with the following details:

- Object Explorer:** On the left, it lists various database objects including tables like `amazon_brazil`, `public`, `customer_data`, `order_items`, `orders`, `payments`, `product`, `sellers`, `Trigger Functions`, `Types`, `Views`, `Subscriptions`, `dvrental`, `exercises`, `learning`, `postres`, and `school`.
- Query Editor:** The main area contains a SQL query to calculate monthly revenue for 2018:

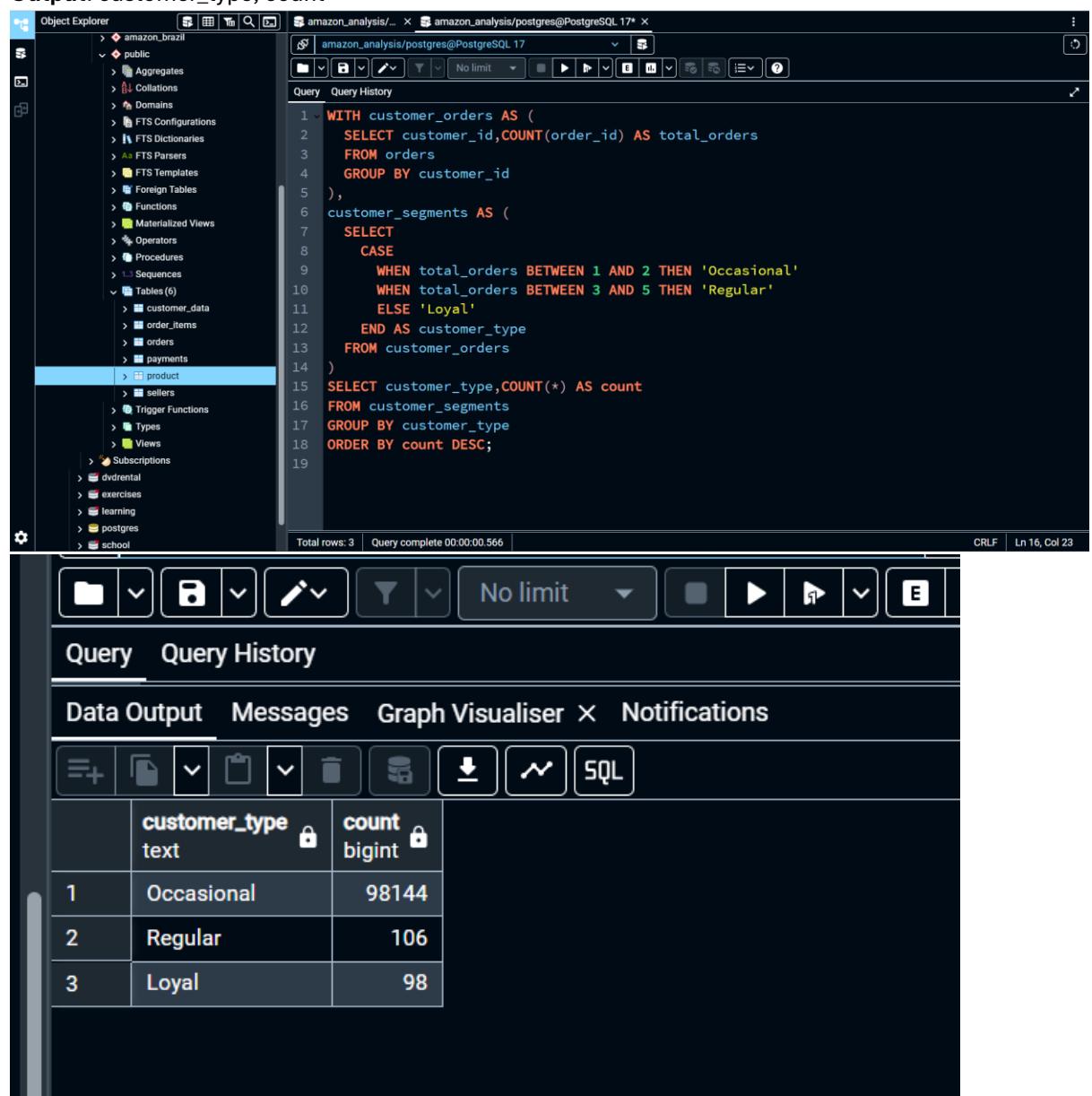

```

1 SELECT
2   EXTRACT(MONTH FROM o.order_purchase_timestamp) AS month,
3   ROUND(SUM(oi.price), 2) AS total_revenue
4   FROM orders o
5   JOIN order_items oi ON o.order_id = oi.order_id
6   WHERE EXTRACT(YEAR FROM o.order_purchase_timestamp) = 2018
7   GROUP BY month
8   ORDER BY month;
9
      
```
- Data Output:** Below the query editor is a table showing the results of the query:

month	total_revenue
1	950030.36
2	844178.71
3	983213.44
4	996647.75
5	996517.68
6	865124.31
7	895507.22
8	854686.33
9	145.00
- Graph Visualiser:** Below the data output is a line chart titled "total_revenue" showing the monthly revenue trend from month 1 to 9. The Y-axis ranges from 0 to 1,000,000. The X-axis shows months 1 through 9. The revenue starts at approximately 950,000 in month 1, dips slightly in month 2, rises to a peak of about 996,000 in month 4, and then gradually declines to around 145,000 by month 9.

QUES4) 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



```

WITH customer_orders AS (
  SELECT customer_id, COUNT(order_id) AS total_orders
  FROM orders
  GROUP BY customer_id
),
customer_segments AS (
  SELECT
    CASE
      WHEN total_orders BETWEEN 1 AND 2 THEN 'Occasional'
      WHEN total_orders BETWEEN 3 AND 5 THEN 'Regular'
      ELSE 'Loyal'
    END AS customer_type
  FROM customer_orders
)
SELECT customer_type, COUNT(*) AS count
FROM customer_segments
GROUP BY customer_type
ORDER BY count DESC;
  
```

Total rows: 3 | Query complete 00:00:00.566 | CRLF | Ln 16, Col 23

	customer_type	count
	text	bigint
1	Occasional	98144
2	Regular	106
3	Loyal	98



QUES5)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

The screenshot shows the pgAdmin interface with two panes. The left pane is the Object Explorer, displaying the schema structure of the 'amazon_brazil' database, including tables like 'customer_data', 'order_items', and 'orders'. The right pane is the Query Editor, showing the execution of a complex SQL query. The query uses common table expressions (CTEs) to calculate total order value, average order value, and finally ranks the customers. The results are displayed in a table at the bottom of the editor.

```
WITH customer_order_totals AS (
    SELECT o.customer_id, o.order_id, SUM(oi.price) AS order_total
    FROM orders o
    JOIN order_items oi ON o.order_id = oi.order_id
    GROUP BY o.customer_id, o.order_id
),
customer_avg_order_value AS (
    SELECT customer_id, ROUND(AVG(order_total), 2) AS avg_order_value
    FROM customer_order_totals
    GROUP BY customer_id
),
ranked_customers AS (
    SELECT customer_id, avg_order_value,
    RANK() OVER (ORDER BY avg_order_value DESC) AS customer_rank
    FROM customer_avg_order_value
)
SELECT customer_id, avg_order_value, customer_rank
FROM ranked_customers
ORDER BY customer_rank
LIMIT 20;
```

	customer_id	avg_order_value	customer_rank
1	1617b1357756262bfa56ab541c47bc...	13440.00	1
2	ec5b2ba62e574342386871631fafd3fc	7160.00	2
3	c6e2731c5b391845f6800c97401a43...	6735.00	3
4	f48d464a0baaea338cb25f816991ab1f	6729.00	4
5	3fd6777bbce08a352fddd04e4a7cc8f6	6499.00	5
6	05455dfa7cd02f13d132aa7a6a9729...	5934.60	6
7	df55c14d1476a9a3467f131269c2477f	4799.00	7
8	24bbf5fd2f2e1b359ee7de94defc4a15	4690.00	8
9	e0a2412720e9ea4f26c1ac985f6a7358	4599.90	9
10	3d979689f636322c62418b6346b1c6...	4590.00	10
11	cc803a2c412833101651d3f90ca7de...	4400.00	11
12	1afc82cd60e303ef09b4ef9837c9505c	4399.87	12
13	35a413c7ca3c69756cb75867d6311c...	4099.99	13
14	e9b0d0eb3015ef1c9ce6cf5b9dcbee9f	4059.00	14
15	c6695e3b1e48680db36b487419fb03...	3999.90	15
16	926b6a6fb8b6081e00b335edef578d...	3999.00	16
17	3be2c536886b2ea4668eced3a80dd0...	3980.00	17
18	31e83c01fce824d0ff786fc48dad009	3930.00	18
19	eb7a157e8da9c488cd4ddc48711f10...	3899.00	19
20	19b32919fa1198aefc0773ee2e46e693	3700.00	20



1. **QUES6)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

The screenshot shows the pgAdmin interface with two windows. The left window is 'Object Explorer' showing database schema, and the right window is a query editor for 'amazon_analysis/postgres@PostgreSQL 17*'.

```
WITH RECURSIVE monthly_sales AS (
  SELECT oi.product_id, DATE_TRUNC('month', o.order_purchase_timestamp) AS sale_month,
  SUM(oi.price) AS monthly_sales FROM orders o
  JOIN order_items oi ON o.order_id = oi.order_id
  GROUP BY oi.product_id, sale_month
),
recursive_cumulative AS (
  SELECT product_id, sale_month, monthly_sales AS total_sales FROM monthly_sales
  UNION ALL
  SELECT ms.product_id, ms.sale_month, rc.total_sales + ms.monthly_sales AS total_sales
  FROM recursive_cumulative rc
  JOIN monthly_sales ms ON rc.product_id = ms.product_id
  AND ms.sale_month = rc.sale_month + INTERVAL '1 month'
)
SELECT product_id, TO_CHAR(sale_month, 'YYYY-MM') AS sale_month,
ROUND(total_sales, 2) AS total_sales
FROM recursive_cumulative
ORDER BY product_id, sale_month;
```

The bottom window shows the results of the query:

product_id	sale_month	total_sales
1	2018-05	101.65
2	2017-12	129.90
3	2017-12	229.00
4	2018-08	117.80
5	2018-04	199.00
6	2017-12	52.00
7	2017-09	498.00
8	2017-10	38.90
9	2017-11	77.80
10	2017-11	116.70
11	2017-12	350.10
12	2017-12	311.20
13	2017-12	233.40
14	2018-08	78.90
15	2017-02	104.97
16	2017-03	34.99
17	2017-03	139.96
18	2017-07	104.97
19	2017-08	69.98
20	2017-08	174.95
21	2017-09	244.93



QUES7) 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
1 WITH monthly_sales AS (
2   SELECT p.payment_type, DATE_TRUNC('month', o.order_purchase_timestamp) AS sale_month,
3   SUM(p.payment_value) AS monthly_total FROM orders o
4   JOIN payments p ON o.order_id = p.order_id
5   WHERE EXTRACT(YEAR FROM o.order_purchase_timestamp) = 2018
6   GROUP BY p.payment_type, sale_month
7 ),
8 growth_calc AS (
9   SELECT payment_type, sale_month, monthly_total,
10   LAG(monthly_total) OVER (
11     PARTITION BY payment_type |
12     ORDER BY sale_month
13   ) AS previous_month_total
14   FROM monthly_sales
15 )
16 SELECT payment_type, TO_CHAR(sale_month, 'YYYY-MM') AS sale_month,
17   ROUND(monthly_total, 2) AS monthly_total,
18   ROUND(
19     CASE
20       WHEN previous_month_total IS NULL THEN NULL
21       WHEN previous_month_total = 0 THEN NULL
22       ELSE ((monthly_total - previous_month_total) / previous_month_total) * 100
23     END, 2
24   ) AS monthly_change
25   FROM growth_calc
26   ORDER BY payment_type, sale_month;
27

Total rows: 36 | Query complete 00:00:00.198 | CRLF | Ln 11, Col 33
```

```
23
24
25
26
27

Total rows: 36 | Query complete 00:00:00.198 | CRLF | Ln 11, Col 33
```

amazon_analysis... × amazon_analysis/postgres@PostgreSQL 17* X

amazon_analysis/postgres@PostgreSQL 17

No limit

Data Output Messages Notifications

Showing rows: 1 to 36 | Page No: 1

	payment_type	sale_month	monthly_total	monthly_change
1	boleto	2018-01	204844.66	[null]
2	boleto	2018-02	183112.72	-10.61
3	boleto	2018-03	191538.02	4.60
4	boleto	2018-04	193547.09	1.05
5	boleto	2018-05	195378.93	0.95
6	boleto	2018-06	153350.28	-21.51
7	boleto	2018-07	198041.24	29.14
8	boleto	2018-08	143805.50	-27.39
9	credit_card	2018-01	868880.38	[null]
10	credit_card	2018-02	778803.00	-10.37
11	credit_card	2018-03	933770.10	19.90
12	credit_card	2018-04	934306.00	0.06
13	credit_card	2018-05	927556.35	-0.72
14	credit_card	2018-06	811508.56	-12.51
15	credit_card	2018-07	803674.49	-0.97
16	credit_card	2018-08	797648.89	-0.75
17	debit_card	2018-01	11543.55	[null]
18	debit_card	2018-02	7469.53	-35.29
19	debit_card	2018-03	8375.11	12.12
20	debit_card	2018-04	10782.53	28.74
21	debit_card	2018-05	9710.74	-9.94

Total rows: 36 | Query complete 00:00:00.112 |

Key Insights & Action Points: ANALYSIS-1

Insights:

- Average order value differs across payment types—some are more suited for higher spenders.
- A few payment methods account for most transactions, showing clear user preference.
- "Smart" products priced between 100–500 BRL are trending.
- Revenue fluctuates by month, showing seasonal patterns.
- Wide price gaps within categories point to varied customer budgets.
- Some payment methods show consistent spend, indicating trust and stability.
- Several product categories have incomplete or inconsistent names—potential data quality issue.

Recommendations:

- Prioritize top-performing, consistent payment types to enhance checkout.
- Push mid-range "Smart" items with targeted campaigns.
- Plan stock and marketing around peak sales months.
- Clean and standardize product category data.
- Use category price ranges to personalize offers for different customer segments.

Key Insights & Action Points: ANALYSIS-2

Key Insights:

- Customers use different payment types based on how much they spend.
- High revenue comes from a few key product categories.
- Most buyers return—some multiple times.
- Products fall into distinct price segments: Low, Medium, High.

Recommendations:

- Tailor payment options and offers to customer spend levels.
- Prioritize top-performing categories for promotions and inventory.
- Segment customers (New, Returning, Loyal) and personalize engagement.
- Encourage repeat purchases through loyalty programs.
- Reassess low-selling categories for improvement or discontinuation.

Key Insights & Action Points: ANALYSIS-3

Key Insights:

- Sales patterns shift with seasons and festivals, reflecting local trends.
- A few standout products sell significantly above average—crucial for business focus.
- Monthly sales show a clear rhythm, with predictable highs and lows.
- Most buyers are either occasional or regular; only a few are truly loyal.
- A small group of top customers contributes a large share of total spending.
- Product sales evolve over time, highlighting clear life cycle stages.
- Payment methods show varying growth rates across months.

Recommendations:

- Align marketing and inventory with Brazil's seasonal and festive sales cycles.
- Prioritize high-performing products in stock and promotions.
- Design loyalty programs that motivate frequent purchases.
- Offer tailored benefits to high-value customers.
- Monitor trends in payment preferences to keep checkout options optimized.
- Use product lifecycle data to time new launches or phase out declining items.
- Localize these insights to fit Indian customer behaviour for better results.