**Amazon Brazil** wants to identify trends and customer behaviours that could be leveraged in the Indian market, given the similarities between Brazil and India—such as large populations and diverse consumer bases—there is an opportunity to replicate success in India. The main objective is to **analyse trends, customer behaviours, and preferences** that could be leveraged in the Indian market which will help Amazon India make informed decisions, enhance customer experience, and seize new opportunities.

This project uses multiple tables: **Customers, Orders, Order Items, Product, Sellers, and Payments**. Through SQL queries, various business-critical questions and actionable insights have been shared.

­­­­­­­­­­­­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ANALYSIS – I**

**Question 1**

**Problem Statement:**

*To simplify its financial reports, Amazon India needs to standardise payment values. Round the average payment values to integer (no decimal) for each payment type and display the results sorted in ascending order. O****utput****: payment\_type, rounded\_avg\_payment*

**Approach:**

1. **Identifying Relevant Tables and Columns:**
   * **Table:** payments
   * **Columns:** payment\_type, payment\_value
2. **Calculating Average Payment Value:**
   * Used the AVG() aggregate function on payment\_value
   * Grouped the results by payment\_type to get the average for each type
3. **Rounding the Averages:**
   * Used the ROUND() function to round the average payment values to the nearest integer.
4. **Ordering the Results:**
   * Ordered the final results in ascending order based on the rounded average payment.

SQL QUERY:

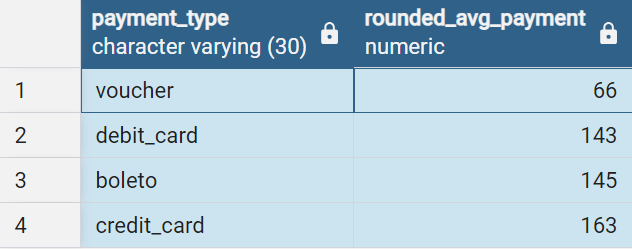
**SELECT** payment\_type, ROUND(AVG(payment\_value)) AS rounded\_avg\_payment

**FROM** amazon\_brazil."Payments"

**WHERE** payment\_value > 0

**GROUP** BY payment\_type

**ORDER BY** rounded\_avg\_payment;

****

**OUTPUT:**

**Recommendations:**

1. **Focus on Popular Payment Methods:**
   * Payment methods like Credit Card and Boleto should be optimised, as their average payment values are higher.
   * Introducing targeted promotions for customers using these methods can lead to higher revenue.
2. **Improve Use of Underperforming Methods:**
   * For payment methods like Debit Card and Voucher, giving promotional discounts or improving transaction processes can be helpful to retain users while increasing usage rate and amount as their average payment values are lower.

**Question 2**

**Problem Statement:**

***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*

**Approach:**

1. **Identifying Relevant Tables and Columns:**
   * **Table:** payments & **Columns:** payment\_type, order\_id
2. **Calculating Total Number of Orders:**
   * Used COUNT() on order\_id
3. **Calculating Orders per Payment Type:**
   * Used COUNT() on payment\_type
4. **Calculating Percentage of Orders and Rounding the result:**
   * Divided the count of orders per payment type by the total number of orders, multiplied by 100, and rounded to one decimal place.
5. **Sorting Results:**
   * Ordered the results in descending order of percentage to highlight the most popular payment methods.

**SQL QUERY:**

**SELECT** payment\_type,

ROUND((COUNT(payment\_type)\*100.0)/(SELECT COUNT(DISTINCT order\_id) FROM amazon\_brazil."Payments"), 1) AS percentage\_orders

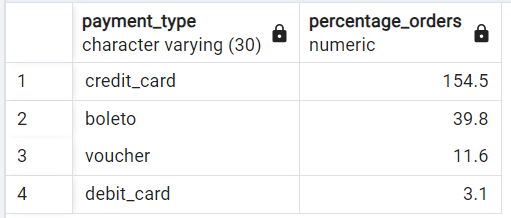
**FROM** amazon\_brazil."Payments"

**WHER**E payment\_value > 0

**GROUP** BY payment\_type

**ORDER BY** percentage\_orders DESC;

**Output:**



**Recommendations:**

1. **Maintain efficiency of Popular Payment Methods:**
   * Since Credit Card is very popular, this payment method needs to be highly reliable, secure, and easy to use.
   * Consider cashback/reward or loyalty programs for Credit Card users to incentivize more purchases and to increase average order value.
2. **Improve Less Popular Payment Methods:**
   * Explore ways to make Boleto, Voucher, and Debit Card more attractive by offering discounts or simplifying the payment process.
   * Further analysis needs to be done to know reason behind low usage of certain methods and address any underlying cause.

**Question 3**

**Problem Statement:**

***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*

**Approach:**

1. Identifying relevant Tables and Columns:
   1. **Table:** Product, Order Items
   2. **Columns:** product\_id, price
2. Used the ROUND() function to round the price to the nearest integer.
3. Used INNER JOIN() to join Product Table and Order Items Table on product\_id as common column
4. Used WHERE() to put required conditions
5. Used Order BY clause to order price from highest to lowest

**SQL QUERY:**

**SELECT** pd.product\_id, ROUND(oi.price, 0) AS price

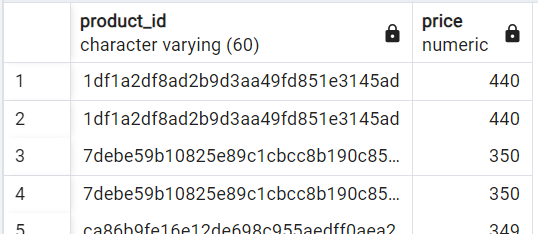
**FROM** amazon\_brazil."Product" pd

**JOIN** amazon\_brazil."Order\_Items" oi **ON** pd.product\_id = oi.product\_id

**WHERE** oi.price BETWEEN '100' AND '500' AND pd.product\_category\_name LIKE '%smart%'

**ORDER BY** oi.price DESC;

**OUTPUT:**



**Question 4**

**Problem Statement:**

*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*

**Approach:**

1. Identifying relevant Tables and Columns:
   1. **Table:** Payments, Orders
   2. **Columns:** payment\_value, order\_purchase\_timestamp, order\_id
2. Used ROUND() to round payment value to nearest integer and extracted month name from timestamp using TO\_CHAR()
3. Used JOIN() to join both tables based on order\_id
4. Used GROUP() to get total\_sales based on each month
5. Used ORDER() to order total\_sales in Descending order to find Top 3 successful month using LIMIT.

**SQL Query:**

**SELECT** ROUND(SUM(pt.payment\_value)) AS total\_sales, To\_CHAR(o.order\_purchase\_timestamp, 'Month') as months

**FROM** amazon\_brazil."Payments" pt

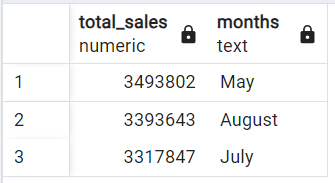
**JOIN** amazon\_brazil."Orders" o **ON** pt.order\_id = o.order\_id

**GROUP BY** months

**ORDER BY** total\_sales DESC

**LIMIT** 3;

**OUTPUT:**



**Question 5**

**Problem Statement:**

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

**Approach:**

1. Identifying relevant Tables and Columns:
   1. **Table:** Product, Order Items
   2. **Columns:** product\_category\_name, price, product\_id
2. Used ROUND() to round difference of Max and Min of price to nearest integer and extracted month name from timestamp using TO\_CHAR()
3. Used JOIN() to join both tables based on product\_id
4. Used GROUP() to group price by product\_category\_name
5. Used HAVING() to put required conditions

**SQL Query:**

**SELECT** pd.product\_category\_name, MAX(oi.price) - MIN(oi.price) AS price\_difference

**FROM** amazon\_brazil."Product" pd

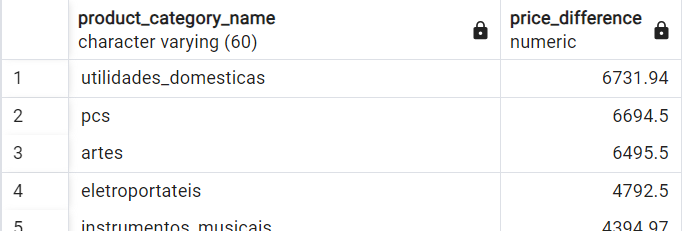
**JOIN** amazon\_brazil."Order\_Items" oi **ON** pd.product\_id = oi.product\_id

**GROUP BY** pd.product\_category\_name

**HAVING** MAX(oi.price) - MIN(oi.price) > 500

**ORDER BY** price\_difference **DESC**;

**OUTPUT:**



**Question 6**

**Problem Statement:**

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

**Approach:**

1. Identifying relevant Tables and Columns:
   1. **Table:** Payments
   2. **Columns:** payment\_type,
2. Used STDDEV() to find variance in payment\_value
3. Used GROUP BY() to get payment\_value for each payment\_type
4. Used ORDER BY() to order std\_deviation in descending order

**SQL Query:**

**SELECT** payment\_type, STDDEV(payment\_value) AS std\_deviation

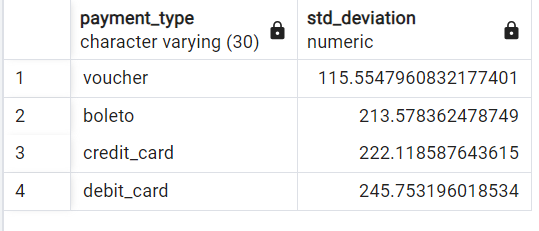
**FROM** amazon\_brazil."Payments"

**WHERE** payment\_value > 0

**GROUP BY** payment\_type

**ORDER BY** std\_deviation;

**OUTPUT:**



**Question 7**

**Problem Statement:**

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

**Approach:**

1. Identifying relevant Tables and Columns:
   1. **Table:** Product
   2. **Columns:** product\_id, product\_category\_name
2. Used WHERE() to to filter under given conditions
3. Used ORDER BY() to order product\_category\_name in descending order

**SQL Query:**

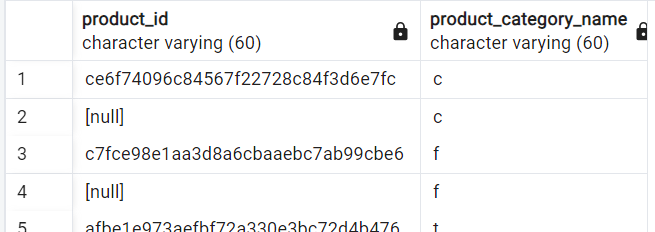
**SELECT** product\_id, product\_category\_name

**FROM** amazon\_brazil."Product"

**WHERE** product\_category\_name IS NULL OR product\_category\_name LIKE '\_'

**ORDER BY** product\_category\_name;

**OUTPUT:**

****

**Recommendations:**

1. **Fix incomplete names**

* Adding missing names in product category can give clarity about product to users which will be helpful in increasing sales.

**ANALYSIS: II**

**Question 1**

**Problem Statement:**

*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*

**Approach:**

1. Identifying relevant Tables and Columns:
   1. **Table:** Payments
   2. **Columns:** payment\_value, payment\_type,
2. Used CASE WHEN() statements to segment customers based on given payment\_value
3. Used COUNT() on payment type to get count of each payment type on given range
4. Used GROUP BY() to get payment\_value for each payment\_type and segment
5. Used ORDER BY() to order payment\_count in ascending order

**SQL Query:**

**SELECT**

**CASE**

**WHEN** payment\_value < 200 **THEN** 'LOW'

**WHEN** payment\_value BETWEEN 200 AND 1000 **THEN** 'MEDIUM'

**ELSE** 'HIGH'

**END** **AS** order\_value\_segment,

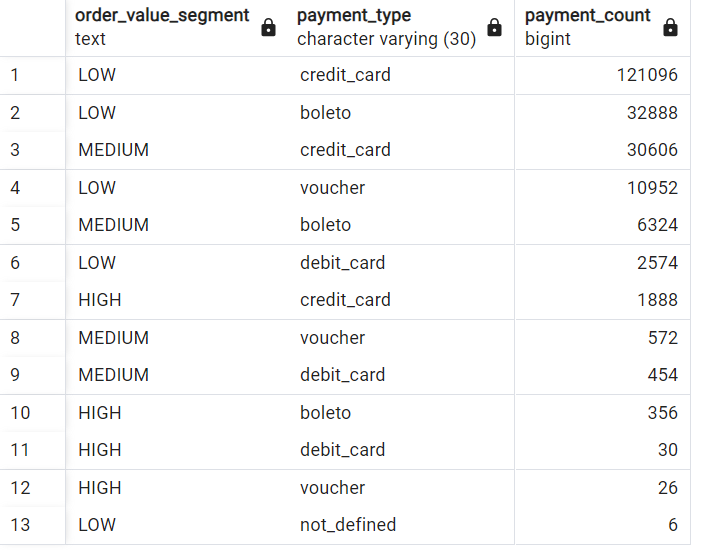
payment\_type, COUNT(payment\_type) **AS** payment\_count

**FROM** amazon\_brazil."Payments"

**GROUP BY** payment\_type, order\_value\_segment

**ORDER BY** payment\_count DESC;

**OUTPUT:**

****

**Question 2**

**Problem Statement:**

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

**Approach:**

1. Identifying relevant Tables and Columns:
   1. **Table:** Product, Order\_Items
   2. **Columns:** product\_category\_name, price
2. **Calculating MIN, MAX and Average Payment Value:**
   * Used the MIN(), MAX(), AVG() aggregate function on price
   * Grouped the results by product\_category \_name to get rounded value for each function
3. Used JOIN() to join Product table with Order\_Items table on product\_id
4. Used ORDER BY() to order price in ascending order

**SQL Query:**

**SELECT** pd.product\_category\_name, ROUND(MIN(oi.price), 2) AS min\_price, ROUND(MAX(oi.price), 2) **AS** max\_price,

ROUND(AVG(oi.price), 2) **AS** avg\_price

**FROM** amazon\_brazil."Product" pd

**JOIN** amazon\_brazil."Order\_Items" oi **ON** pd.product\_id = oi.product\_id

**GROUP** **BY** pd.product\_category\_name

**ORDER** **BY** avg\_price **DESC**;

**OUTPUT:**



**Question 3**

**Problem Statement:**

*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*

**Approach:**

1. Identifying relevant Tables and Columns:
   1. **Table:** Payments
   2. **Columns:** payment\_value, payment\_type,
2. Used COUNT() on order\_id to get number of total orders
3. Used GROUP BY() with HAVING to group customers based on their unique id and more than 1 orders
4. Used ORDER BY() to order total orders in ascending order

**SQL Query:**

**SELECT** cs.customer\_unique\_id, COUNT(o.order\_id) **AS** total\_orders

**FROM** amazon\_brazil."Orders" o

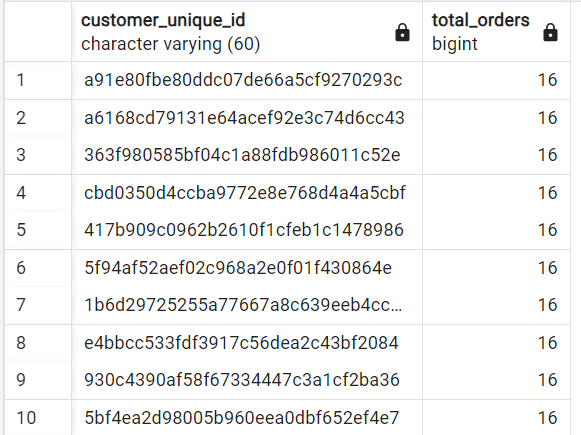
**JOIN** amazon\_brazil."Customers" cs **ON** o.customer\_id = cs.customer\_id

**GROUP BY** cs.customer\_unique\_id

**HAVING** COUNT(o.order\_id) > 1

**ORDER BY** total\_orders DESC;

**OUTPUT:**

****

**Question 4**

**Problem Statement:**

*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\_id, customer\_type*

**Approach:**

1. Identifying relevant Tables and Columns:
   1. **Table:** Orders, Customers
   2. **Columns:** order\_id, customer\_id
2. Used CASE WHEN() statements to segment customers based on number of orders placed by each customer

* Used COUNT() on order\_id to count number of orders
* Used GROUP BY() TO group number of orders by customer\_id

1. Used JOIN() on ct and customers table to find required values

**SQL Query:**

**WITH** ct **AS** (

**SELECT** o.customer\_id,

**CASE**

**WHEN** COUNT(order\_id) = 1 **THEN** 'NEW'

**WHEN** COUNT(order\_id) BETWEEN 2 AND 4 **THEN** 'RETURNING'

**WHEN** COUNT(order\_id) > 4 **THEN** 'LOYAL'

**ELSE** 'NO ORDER'

**END** **AS** customer\_type

**FROM** amazon\_brazil."Orders" o

**GROUP BY** customer\_id

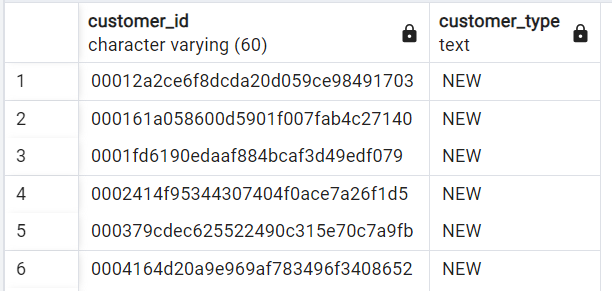
)

**SELECT** cs.customer\_id, ct.customer\_type

**FROM** amazon\_brazil."Customers" cs

**LEFT JOIN** ct on cs.customer\_id = ct.customer\_id

**OUTPUT:**

****

**Question 5**

**Problem Statement:**

*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*

**Approach:**

1. Identifying relevant Tables and Columns:
   1. **Table:** Product, Order\_Items
   2. **Columns:** product\_category\_name, price, product\_id
2. Using aggregate function

* Used SUM() to find total amount of product ordered
* Used GROUP Y() TO group it by product\_category\_name

1. Used INNER JOIN() on product\_id to join Product table with Order\_Items table
2. Used ORDER BY() to order total\_revenue in descending order
3. Used LIMIT clause to find Top 5 categories

**SQL Query:**

**SELECT** pd.product\_category\_name, SUM(oi.price) **AS** total\_revenue

**FROM** amazon\_brazil."Product" pd

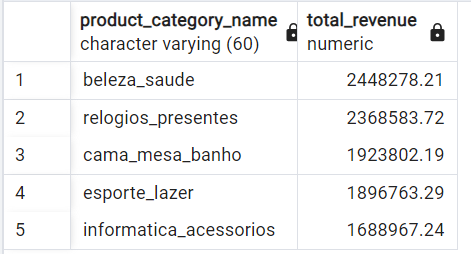
**JOIN** amazon\_brazil."Order\_Items" oi **ON** pd.product\_id = oi.product\_id

**GROUP BY** pd.product\_category\_name

**ORDER BY** total\_revenue **DESC**

**LIMIT** 5;

**OUTPUT:**

****

**ANALYSIS: III**

**Question 1**

**Problem Statement:**

*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*

**Approach:**

1. Identifying relevant Tables and Columns:
   1. **Table:** Orders, Payments
   2. **Columns:** order\_purchase\_timestamp, payment\_value, order\_id
2. Used CASE WHEN() statements to segment customers based on months

* Used EXTRACT() to get month values and segmented it into season of year
* Used SUM() on payment\_value to get value of total sales

1. Used INNER JOIN() ON ORDER\_ID to join Orders table with Payments table
2. Used GROUP BY() on season to get sales values on each season

**SQL Query:**

**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,

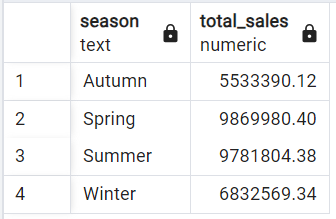
SUM(pt.payment\_value) **AS** total\_sales

**FROM** amazon\_brazil."Orders" O

**JOIN** amazon\_brazil."Payments" pt **ON** o.order\_id = pt.order\_id

**GROUP BY** season;

**OUTPUT:**

****

**Question 2**

**Problem Statement:**

*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*

**Approach:**

1. Identifying relevant Tables and Columns:
   1. **Table:** Order\_Items
   2. **Columns:** product id, order\_id
2. Used CTE()

* Used COUNT() on order\_item\_id to calculate total\_quantity\_sold
* Used GROUP BY() to group it by product\_id

1. Used Subquery in WHERE clause to filter products based on given conditions
2. Used GROUP BY() to get payment\_value for each payment\_type and segment
3. Used ORDER BY() to order total\_quantity\_sold in descending order

**SQL Query:**

**WITH** coi **AS** (

**SELECT** product\_id, COUNT(order\_item\_id) **AS** total\_quantity\_sold

**FROM** amazon\_brazil."Order\_Items"

**GROUP BY** product\_id

)

**SELECT** product\_id, total\_quantity\_sold

**FROM** coi

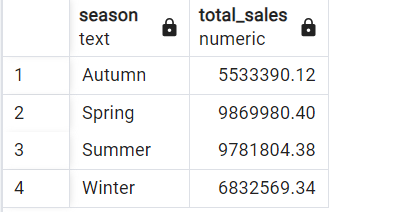
WHERE total\_quantity\_sold > (

**SELECT** AVG(total\_quantity\_sold) **FROM** coi

)

**ORDER BY** total\_quantity\_sold **DESC**;

**OUTPUT:**

****

**Question 3**

**Problem Statement:**

*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*

**Approach:**

1. Identifying relevant Tables and Columns:
   1. **Table:** Orders, Payments
   2. **Columns:** payment\_value, order\_purchase\_timestamp
2. Used LEFT JOIN() to get relevant columns

* Used TO\_CHAR() and DATE\_TRUNC() in timestamp to calculate month values
* Used EXTRACT() in timestamp to fix year as 2018
* Used GROUP BY() to group above data by Months
* Usesd OORDER BY() to order above data by Months

1. Used SUM() to get sum of payment\_value and grouped it by months

**SQL Query:**

**SELECT** TO\_CHAR(DATE\_TRUNC('MONTH', o.order\_purchase\_timestamp), 'YYYY-MM') AS month,

SUM(pt.payment\_value) AS total\_revenue

**FROM** amazon\_brazil."Orders" o

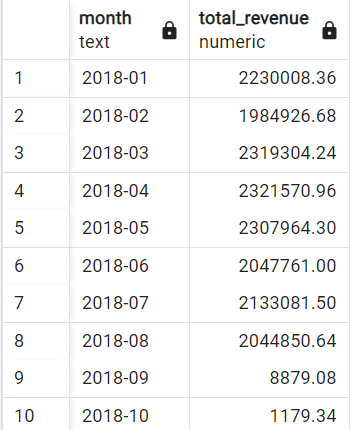
**JOIN** amazon\_brazil."Payments" pt ON o.order\_id = pt.order\_id

**WHERE** EXTRACT(YEAR FROM o.order\_purchase\_timestamp) = 2018

**GROUP** **BY** month

**ORDER** **BY** month;

**OUTPUT:**

****

**Question 4**

**Problem Statement:**

*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*

**Approach:**

1. Identifying relevant Tables and Columns:
   1. **Table:** Orders
   2. **Columns:** order\_id, customer\_id
2. Used CASE WHEN() statements to segment customers based on number of orders
3. Used COUNT() on order\_id to calculate total orders for each customer
4. Used GROUP BY() to group customer\_type based on customer\_id

**SQL Query:**

**WITH** cust\_det AS(

**SELECT**

**CASE**

**WHEN** COUNT(order\_id) BETWEEN 1 AND 2 THEN 'Occasional'

**WHEN** COUNT(order\_id) BETWEEN 3 AND 5 THEN 'Regular'

**ELSE** 'Loyal'

**END** AS customer\_type,

COUNT(order\_id) AS Order\_Count

**FROM** amazon\_brazil."Orders"

**GROUP BY** customer\_id

)

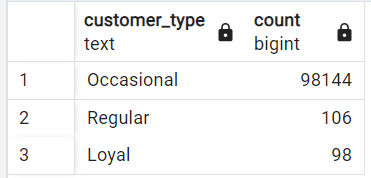
**SELECT** customer\_type, COUNT(\*) AS count

**FROM** cust\_det

**GROUP BY** customer\_type

**ORDER BY** count **DESC**;

**OUTPUT:**

****

**Question 5**

**Problem Statement:**

*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*

**Approach:**

1. Identifying relevant Tables and Columns:
   1. **Table:** Orders, Payments
   2. **Columns:** customer\_id, payment\_value
2. Used CTE() to calculate required columns
3. Used the AVG() aggregate function on payment\_value. Grouped the result by customer\_id to get payment value for each type
4. Used the ROUND() function to round the average payment values to two decimal place.
5. Used GROUP BY() to group avg\_order\_value based on customer\_id
6. Used RANK() FROM WINDOW FUNCTION to find Top 20 customers using LIMIT clause
7. Used ORDER BY() to order the final results in descending order based on the rounded average payment.
8. Used LIMIT to get Top 20 customers.

**SQL Query:**

**WITH** Top\_20 AS(

**SELECT** o.customer\_id, ROUND(AVG(pt.payment\_value),2) AS avg\_order\_value

**FROM** amazon\_brazil."Orders" o

**JOIN** amazon\_brazil."Payments" pt **ON** o.order\_id = pt.order\_id

**GROUP BY** o.customer\_id

)

**SELECT** customer\_id, avg\_order\_value,

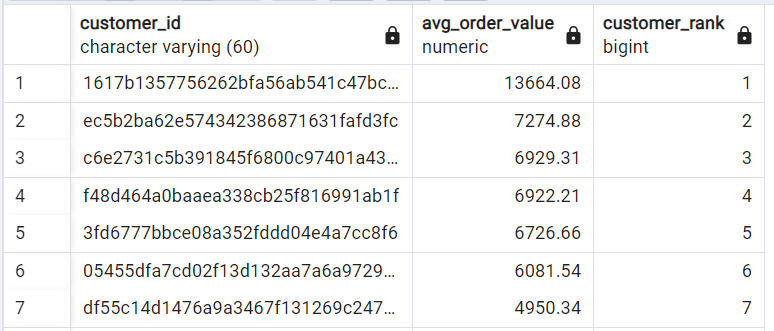
RANK() OVER(ORDER BY avg\_order\_value DESC) AS "customer\_rank"

FROM Top\_20

**ORDER BY** avg\_order\_value DESC;

**LIMIT** 20;

**OUTPUT:**

****

**Question 6**

**Problem Statement:**

*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*

**Approach:**

1. Identifying relevant Tables and Columns:
   1. **Table:** Orders, Order\_Items
   2. **Columns:** product\_id, order\_purchase\_timestamp, order\_id
2. Used CTE to calculate relevant columns

* Used To\_CHAR() and DATE\_TRUNC to convert given timestamp to month value
* Used SUM() on price to get monthly sales
* Used LEFT JOIN() to join Orders table with Order Items table on order id
* Used GROUP BY() to get values based on product id and sale month

1. Used SUM() through window function on product id and ordered by sale month to get total sales
2. Used ORDER BY() to order it by product\_id and sale month

**SQL Query:**

**WITH** growth\_trends AS(

**SELECT** oi.product\_id,

TO\_CHAR(DATE\_TRUNC('month', o.order\_purchase\_timestamp), 'YYYY-MM') AS sale\_month,

SUM(oi.price) AS monthly\_sales

**FROM** amazon\_brazil."Orders" o

JOIN amazon\_brazil."Order\_Items" oi **ON** o.order\_id = oi.order\_id

**GROUP BY** oi.product\_id, sale\_month

)

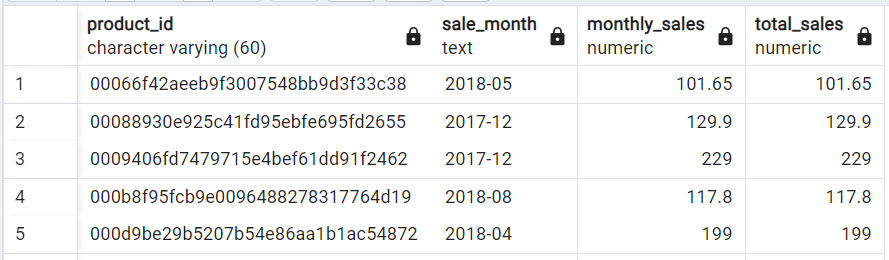
**SELECT** product\_id, sale\_month, monthly\_sales,

SUM(monthly\_sales) OVER(PARTITION BY product\_id ORDER BY sale\_month) AS total\_sales

**FROM** growth\_trends

**ORDER BY** product\_id, sale\_month;

**OUTPUT:**

****

**Question 7**

**Problem Statement:**

*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.*

**Approach:**

1. Identifying relevant Tables and Columns:
   1. **Table:** Payments, Orders
   2. **Columns:** payment\_type, payment\_value, order\_purchase\_timestamp, order\_id
2. Used CTE to get relevant columns

* Used TO\_CHAR() and DATE\_TRUNC() in timestamp to calculate month values
* Used SUM() on payment\_value to get monthly\_total
* Used LEFT JOIN() on Orders table to join with Payments table
* Used DATE\_PART() in timestamp to fix year as 2018
* Used GROUP BY() to group above data by payment type and month

1. Used LAG() window function and ROUND() to get month on month change
2. Used ORDER BY() to order above data by payment type, and sale month

**SQL Query:**

**WITH** monthly\_sales AS(

**SELECT** pt.payment\_type,

TO\_CHAR(DATE\_TRUNC('month', o.order\_purchase\_timestamp), 'YYYY-MM') AS sale\_month,

SUM(payment\_value) AS monthly\_total

**FROM** amazon\_brazil."Payments" pt

**JOIN** amazon\_brazil."Orders" o **ON** pt.order\_id = o.order\_id

**WHERE** DATE\_PART('Year', o.order\_purchase\_timestamp) = 2018 AND pt.payment\_value > 0

**GROUP BY** pt.payment\_type, sale\_month

)

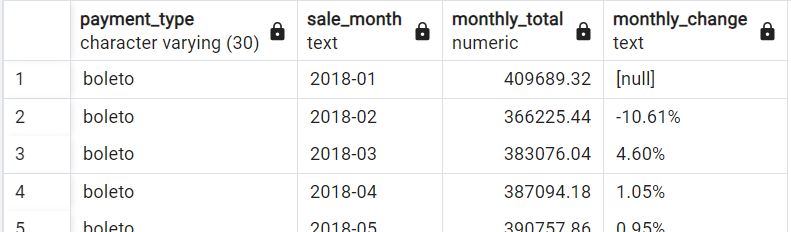
**SELECT** payment\_type, sale\_month, monthly\_total,

ROUND(((monthly\_total - LAG(monthly\_total) OVER(PARTITION BY payment\_type ORDER BY sale\_month))\*100/ (LAG(monthly\_total) OVER(PARTITION BY payment\_type ORDER BY sale\_month))), 2) || '%' AS monthly\_change

**FROM** monthly\_sales

**ORDER BY** payment\_type, sale\_month;

**OUTPUT:**

****