Data Analysis using SQL

Analyzed Amazon Brazil's data to identify trends and customer behaviors that could be leveraged in the Indian market. Focused on customer demographics and behavior using the Customers table to understand purchase patterns and preferences. Evaluated regional trends and customer density through the Geolocation table. Track order lifecycles, product preferences, and seller performance using the Orders, Order Items, Product, and Seller tables. Additionally, analyzed payment preferences and transaction details via the Payments table. This comprehensive analysis will help Amazon India enhance customer experience and seize new market opportunities.

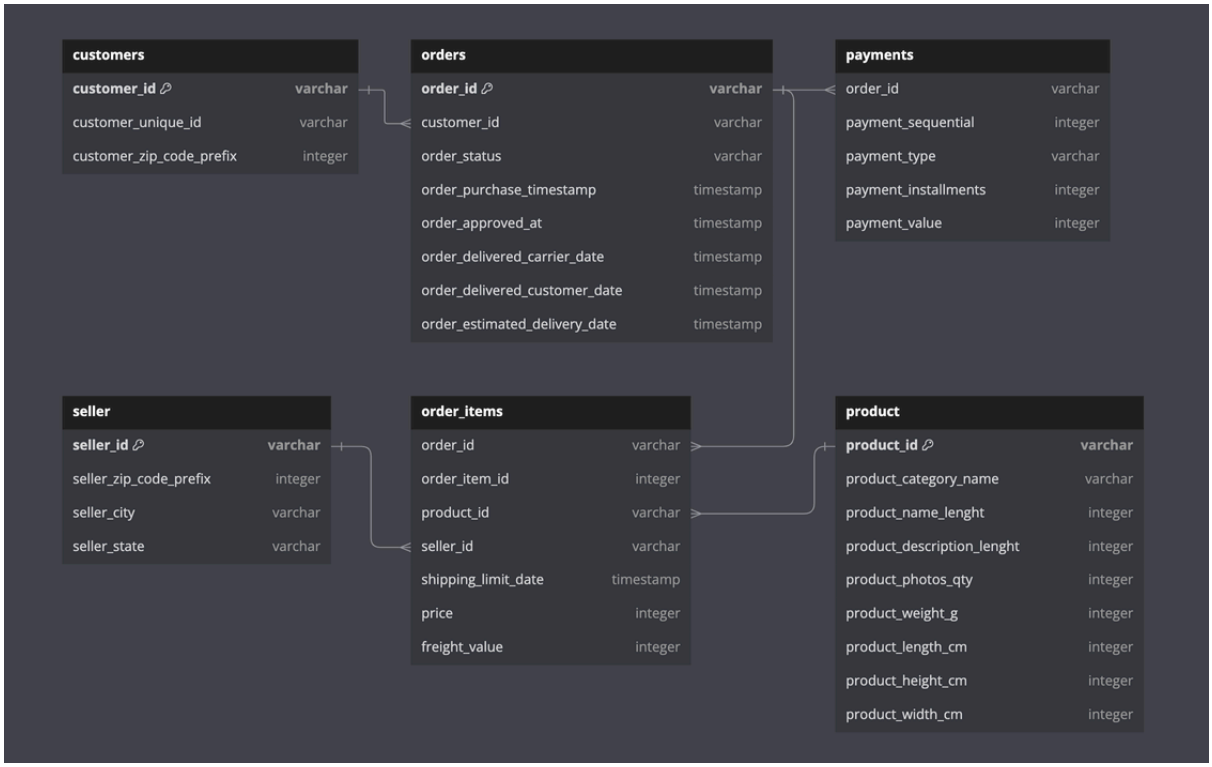
Background

Amazon, a global leader in e-commerce, has achieved significant success in markets like the U.S., Europe, and Asia. In Brazil, Amazon connects small and medium businesses with millions of customers, becoming a key player. Given the similarities between Brazil and India such as large populations and diverse consumer bases there’s an opportunity to replicate success in India.

Overview of the Schema

The schema consists of seven interconnected tables that provide insights into the operations of Amazon Brazil:

- **Customers**: Holds customer information, including unique identifiers and locations, to study customer demographics and behavior.
- **Orders**: Captures details about each order, such as order status and timestamps, essential for tracking the order lifecycle.
- **Order Items**: Lists each item in an order with details like price, seller, and shipping information.
- **Product**: Contains product specifications, such as category, size, and weight, for product-level analysis.
- **Seller**: Provides data about sellers, including their location and unique identifiers, to evaluate seller performance.
- **Payments**: Records transaction details, such as payment type and value, to understand payment preferences and financial performance.



Entity Relationship Diagram

The whole analysis is divided into 3 parts and it covers different topics in each analysis starting from basic SQL to Joins to Window and CTE functions.

Analysis - I

Problem Statement: 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*

Approach:

- Selected payments table using payment_type and payment_value column
- Filtered valid transactions with payment_value > 0
- Group by payment_type then calculated average payment value using AVG( ) function
- Converted averages to integers using ROUND( ) to remove decimals
- Ordered results in ascending order

SQL query:

```
select payment_type, ROUND(avg(payment_value)as Integer) as rounded_avg_payment
from payments
where payment_value > 0
group by payment_type
order by rounded_avg_payment;
```

Output:

	payment_type character varying	rounded_avg_payment integer
1	voucher	66
2	debit_card	143
3	boleto	145
4	credit_card	163

Analysis & Findings:

- Credit card has the highest average payment value (163), showing customers prefer credit-based payments for higher-value purchases.
- Voucher payments have the lowest average value (66), they are mostly used for low-value transactions.

Recommendations:

- Amazon India should promote credit card offers such as EMI and cashback on high-value products to increase average order value.
- Voucher-based promotions can be used strategically to increase order volume among price-sensitive customers.

Problem Statement: 1.2

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*

Approach:

- Selected payments table using payment_type and order_id
- Filtered valid transactions with payment_value > 0
- Group by payment_type then counted orders for each payment type using COUNT( )
- Calculated percentage contribution by dividing each payment type count by total distinct orders and multiplying by 100
- Rounded the percentage to one decimal place using ROUND( )
- Ordered the results in descending order to show the most used payment types first

SQL Query:

```
select payment_type,
ROUND(count(payment_type)*100/(select count(distinct(order_id)) from payments), 1)
as percentage_orders
from payments
where payment_value > 0
group by payment_type
order by percentage_orders desc;
```

Output:

	payment_type character varying	percentage_orders numeric
1	credit_card	77.0
2	boleto	19.0
3	voucher	5.0
4	debit_card	1.0

Analysis & Findings:

- Credit card dominates the payment distribution with 77% of total orders, showing strong customer preference for credit-based payments.
- Debit card and voucher payments have very low usage (1% and 5%), indicating lower adoption compared to other payment methods.

Recommendations:

- Amazon India should continue promoting credit card offers and partnerships to maintain high conversion rates and customer convenience.
- Incentives such as cashback or discounts can be introduced for debit card and voucher payments to improve their adoption and balance payment distribution.

Problem Statement: 1.3

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*

Approach:

- Selected columns product_id and price from the products and order_items tables
- Joined products with order_items using product_id to access product details along with pricing
- Filtered records where price lies between 100 and 500 BRL using the BETWEEN
- Applied a text filter using LIKE '%smart%' to select products containing the word Smart in their name.
- Sorted the final result in descending order of price to display higher-priced products first

SQL Query:

```
Select P.product_id, O.price as Price
from products as P
join order_items as O
on P.product_id = O.product_id
where (O.price between '100' and '500')
AND (P.product_category_name like '%smart%')
Order by O.price desc;
```

Output:

	product_id character varying	price numeric
1	1df1a2df8ad2b9d3aa49fd851e314...	439.99
2	7debe59b10825e89c1cbcc8b190c...	349.99
3	ca86b9fe16e12de698c955aedff0ae...	349
4	ca86b9fe16e12de698c955aedff0ae...	349
5	0e52955ca8143bd179b311cc454a...	335
6	7aeaa8f3e592e380c420e8910a717...	329.9
7	7aeaa8f3e592e380c420e8910a717...	329.9

Recommendations:

- Amazon India should focus promotional campaigns and discounts on mid-range smart products to increase conversion and sales volume.

Problem Statement: 1.4

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*

Approach:

- Selected order_purchase_timestamp from orders and payment_value from payments
- Joined orders and payments using order_id to combine order dates with payment amounts
- Extracted the month from order_purchase_timestamp using TO_CHAR( ) to group sales month-wise
- Calculated total sales using SUM(payment_value) and rounded the result to the nearest integer
- Sorted the results in descending order of total sales
- Limited the output to the top 3 months with the highest sales

SQL Query:

```
Select TO_CHAR(O.order_purchase_timestamp, 'Month') as Months,
Round(sum(P.payment_value)) as Total_sales
from orders as O
join payments as P
on O.order_id = P.order_id
group by Months
order by Total_sales desc
limit 3;
```

Output:

	months text	total_sales numeric
1	May	1746901
2	August	1696822
3	July	1658924

Analysis & Findings:

- May recorded the highest total sales (1,746,901), followed by August and July, indicating strong customer purchasing activity during mid-year months.

Recommendations:

- Amazon India should plan major sales campaigns, inventory stocking, and marketing activities during high-performing months to maximize revenue.
- Promotional offers and targeted marketing should be introduced during lower-performing months to improve sales consistency throughout the year.

Problem Statement: 1.5

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*

Approach:

- Selected product_category_name from products and price from order_items
- Joined products with order_items using product_id to link categories with product prices
- Grouped results by product_category_name to compute category-level price differences
- Calculated price variation using the difference between MAX(price) and MIN(price) for each category
- Applied a HAVING condition to keep only categories where the price difference is greater than 500
- Ordered the results in descending order of price difference to highlight categories with the highest

SQL Query:

```
select P.product_category_name, (max(OI.price) - min(OI.price)) as price_difference
from Products as P
Join Order_items as OI
on P.product_id = OI.product_id
group by P.product_category_name
having (max(OI.price) - min(OI.price)) > 500
order by price_difference desc;
```

Output:

	product_category_name character varying	price_difference numeric
1	utilidades_domesticas	6731.94
2	pcs	6694.5
3	artes	6495.5
4	eletroportateis	4792.5
5	instrumentos_musicais	4394.97
6	consoles_games	4094.81
7	esporte_lazer	4054.5
8	relogios_presentes	3990.91

Analysis & Findings:

- Categories such as *utilidades_domesticas, pcs*, and *artes* show very high price differences (above 6000 BRL), indicating presence of both low-priced and premium products within the same category.

Recommendations

- Amazon India should segment high-variation categories into budget, mid-range, and premium groups to improve product discovery and customer decision-making.
- Clear pricing filters and category-level recommendations should be introduced to help customers easily navigate products across different price ranges.

Problem Statement: 1.6

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:

- Selected payments table using payment_type and payment_value
- Filtered valid transactions using WHERE payment_value > 0 to exclude zero-value payments..
- Grouped the data by payment_type to compute standard deviation separately for each payment method by using STDDEV_SAMP(payment_value)
- Ordered the results in ascending order so payment types with the smallest standard deviation appear first, indicating more consistent transaction amounts

SQL Query:

```
select payment_type, STDDEV_SAMP(payment_value) as std_deviation
from payments
where payment_value > 0
group by payment_type
order by std_deviation;
```

Output:

	payment_type character varying	std_deviation numeric
1	voucher	115.559804417648
2	boleto	213.581061475527
3	credit_card	222.119310741208
4	debit_card	245.793401040647

Analysis & Findings:

- Voucher payments have the lowest standard deviation (~115.56), indicating more consistent and predictable transaction amounts.
- Debit card and credit card payments show higher variation, suggesting wider differences in transaction values across purchases.

Recommendations:

- Amazon India can use voucher-based payments for fixed-value promotions or discount campaigns where consistent transaction amounts are preferred.

Problem Statement: 1.7

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*

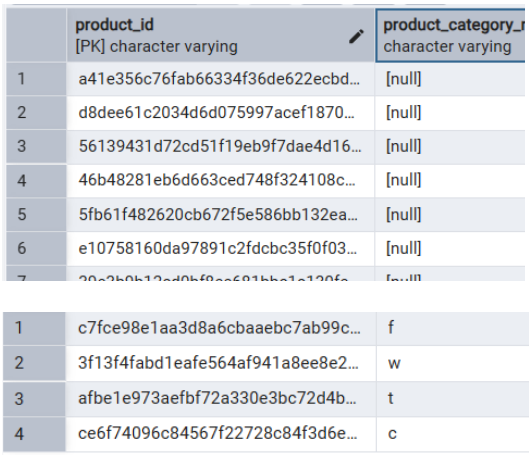
Approach:

- Selected products table using product_id and product_category_name
- Filtered missing category names using product_category_name IS NULL to identify products without a defined category.
- And filtered incomplete category names using product_category_name LIKE '_' to capture entries containing only a single character.
- Returned product_id and product_category_name to view incomplete product data

SQL Query:

```
select product_id, product_category_name
from products
where product_category_name is null or
product_category_name like '_';
```

Output:



	product_id [PK] character varying	product_category_r character varying
1	a41e356c76fab66334f36de622ecbd...	[null]
2	d8dee61c2034d6d075997acef1870...	[null]
3	56139431d72cd51f19eb9f7dae4d16...	[null]
4	46b48281eb6d663ced748f324108c...	[null]
5	5fb61f482620cb672f5e586bb132ea...	[null]
6	e10758160da97891c2fdcbc35f0f03...	[null]
1	c7fce98e1aa3d8a6cbaaebc7ab99c...	f
2	3f13f4fabd1eafe564af941a8ee8e2...	w
3	afbe1e973aefbf72a330e3bc72d4b...	t
4	ce6f74096c84567f22728c84f3d6e...	c

Recommendations:

- Fix incomplete names and add missing names in product category can give clarity about product to users which will be helpful in increasing sales.

Analysis - II

Problem Statement: 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*

Approach:

- Selected payments table using payment_type and payment_value
- Created order value segments using a CASE statement to classify transactions as Low less than 200, Medium between 200 and 1000, and High greater than 1000
- Grouped the data by both payment_type and order value segment and counted transactions using COUNT( )
- Ordered the results in descending order of transaction count to highlight the most commonly used payment types within each value segment

SQL Query:

```
select order_value_segment, payment_type, count(payment_type) as counts
from (select payment_type, payment_value,
case
when payment_value < 200 then 'low'
when payment_value between 200 and 1000 then 'medium'
Else 'High'
End as order_value_segment
from payments)
group by payment_type, order_value_segment
order by counts desc;
```

Output:

	order_value_segment text	payment_type character varying	counts bigint
1	low	credit_card	60548
2	low	boleto	16444
3	medium	credit_card	15303
4	low	voucher	5476
5	medium	boleto	3162
6	low	debit_card	1287
7	High	credit_card	944
8	medium	voucher	286
9	medium	debit_card	227
10	High	boleto	178
11	High	debit_card	15
12	High	voucher	13
13	low	not_defined	3

Analysis & Findings:

- Credit card is the most used payment type across all order value segments, especially in the low

and medium segments, indicating strong customer preference for credit-based payments.
- High-value orders have comparatively lower transaction counts across all payment types, showing fewer purchases in the high-value segment.

Recommendations:

- Amazon India should continue promoting credit card offers and installment options to maintain high usage across all order segments.
- Promotions or financing options should be introduced for high-value products to encourage more purchases in the high-value segment.

Problem Statement: 2.2

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*

Approach:

- Selected tables, products and order_items, using columns product_category_name, product_id, and price
- Joined both tables using product_id to associate each product category with its corresponding price records
- Grouped the data by product_category_name to organize records at the category level before aggregation
- Applied aggregation functions MIN(price), MAX(price), and AVG(price) to calculate minimum, maximum, and average price for each category
- Ordered the final results in descending order of average price to display categories with higher average prices first

SQL Query:

```
select P.product_category_name,
min(OI.price) as min_price, max(OI.price) as max_price, avg(OI.price) as avg_price
from products as P
join order_items as OI
on P.product_id = OI.product_id
group by P.product_category_name
order by avg_price desc;
```

Output:

	product_category_name character varying	min_price numeric	max_price numeric	avg_price 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_prepar...	17.42	1099	264.5686666666666667
8	telefonia_fixa	6	1790	225.6931818181818182

Analysis & Findings:

- Category *pcs* have the highest average prices, indicating premium product positioning and higher customer spending in this category.

Recommendations:

- Amazon India should prioritize visibility and targeted promotions for high average price categories to maximize revenue contribution.
- Categories with large price variation should be segmented into budget, mid-range, and premium filters to improve customer navigation and purchase decision-making.

Problem Statement: 2.3

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*

Approach:

- Selected tables, customers and orders, using columns customer_unique_id, customer_id, and order_id
- Joined both tables using customer_id to associate each order with its corresponding customer
- Grouped the data by customer_unique_id to organize records at the customer level before aggregation
- Applied COUNT(order_id) to calculate the total number of orders placed by each customer
- Filtered results using HAVING COUNT(order_id) > 1 to retain only customers with multiple orders
- Ordered the final output in descending order of total orders to show customers with the highest order count first

SQL Query:

```
select C.customer_unique_id, count(O.order_id) as total_orders
from customers as C
join orders as O
on C.customer_id = O.customer_id
group by C.customer_unique_id
having count(O.order_id) > 1
order by total_orders desc;
```

Output:

	customer_unique_id character varying	total_orders bigint
1	a91e80fbe80ddc07de66a5cf927029...	16
2	a6168cd79131e64acef92e3c74d6cc...	16
3	363f980585bf04c1a88fdb986011c5...	16
4	cbd0350d4ccba9772e8e768d4a4a5...	16
5	417b909c0962b2610f1cfeb1c14789...	16
6	5f94af52aef02c968a2e0f01f430864e	16
7	1b6d29725255a77667a8c639eeb4c...	16
8	e4bbcc533fdf3917c56dea2c43bf20	16

Problem Statement: 2.4

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_id, customer_type*

Approach:

- Selected tables customers and orders using columns customer_id and order_id
- Created a temporary table using a CTE to calculate total orders for each customer by joining both tables on customer_id
- Grouped the data by customer_id to organize records at the customer level before aggregation
- Applied COUNT(order_id) to compute total orders placed by each customer inside the CTE
- Used a CASE statement on the aggregated result to classify customers as New for 1 order Returning for 2-4 orders, and Loyal for more than 4 orders
- Selected customer_id and the derived customer_type as the final output

SQL Query:

```
WITH cte_temp as (
select  C.customer_id, count(O.order_id) as total_orders
from customers as C
join orders as O
on C.customer_id = O.customer_id
group by C.customer_id
order by total_orders desc
)
select customer_id,
case
when total_orders = 1 then 'New'
when total_orders between 2 and 4 then 'Returning'
Else 'loyal'
END AS customer_type
From cte_temp
```

Output:

	customer_id [PK] character varying	customer_type text
1	3922e40a232b96383899bb8b6f111...	loyal
2	1d5c5823a964c7891fca921536fb2...	loyal
3	64e37599865347bc3329d0d79faa8...	loyal
4	3249a3fb2ccd4b5f7103a6e759c0d...	loyal
5	120cfe05591775d14f4852a9080a2...	loyal
6	32a079c19bb9adf352e0e93c4fb9b...	loyal
7	751e579ed6c435ddbe018ccc5b68e...	loyal
8	827d7240a35889668129a7b10aa0...	loyal

Problem Statement: 2.5

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*

Approach

- Selected tables products, order_items, and payments using columns product_category_name, product_id, order_id, and payment_value.
- Joined products with order_items using product_id to associate products with their orders
- Joined order_items with payments using order_id to link payment values with each ordered product
- Grouped the data by product_category_name to organize records at the category level before aggregation
- Applied SUM(payment_value) to calculate total revenue generated by each product category
- Ordered the results in descending order of total revenue and limited the output to the top 5 categories

SQL Query:

```
select P.product_category_name, sum(payment_value) as total_revenue
from products as P
join order_items as OI
on P.product_id = OI.product_id
join payments as PM
ON OI.order_id = PM.order_id
group by P.product_category_name
order by sum(payment_value) desc
limit 5;
```

Output:

	product_category_name character varying	total_revenue numeric
1	cama_mesa_banho	1706393.64
2	beleza_saude	1656408.68
3	informatica_acessorios	1583703.92
4	moveis_decoracao	1429063.39
5	relogios_presentes	1427210.78

Analysis & Findings:

- Categories such as *cama_mesa_banho, beleza_saude,* and *informatica_acessorios* generate the highest total revenue, indicating strong customer demand and consistent sales volume in these categories.

Recommendations:

- Amazon India should prioritize inventory availability, promotions, and marketing investment in top revenue-generating categories to maximize overall sales.

Analysis - III

Problem Statement: 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*

Approach:

- Selected tables orders and payments using columns order_id, order_purchase_timestamp, and payment_value.
- Extracted the month from order_purchase_timestamp using EXTRACT(MONTH) in a subquery to prepare data for seasonal classification
- Created a CTE to assign seasons using a CASE statement, mapping months to Spring, Summer, Autumn, and Winter
- Joined the seasonal data with the payments table using order_id to associate payment values with each season
- Grouped the data by season to organize records before aggregation
- Applied SUM(payment_value) to calculate total sales for each season
- Ordered the results in descending order of total sales to display seasons with higher sales first

SQL Query:

```
With my_cte as (
select order_id,
case
when months between 3 and 5 then 'Spring'
when months between 6 and 8 then 'Summer'
when months between 9 and 11 then 'Autum'
else 'Winter'
End as season
from (select order_id, EXTRACT(MONTH FROM order_purchase_timestamp) as months
from orders)
)
select T1.season as seasons, sum(payment_value) as total_sales
from my_cte as T1
join payments as P
on T1.order_id = P.order_id
group by T1.season
order by sum(payment_value) desc;
```

Output:

	seasons text	total_sales numeric
1	Spring	4934990.20
2	Summer	4890902.19
3	Winter	3416284.67
4	Autum	2766695.06

Analysis & Findings:

- Spring and Summer seasons generate the highest total sales, indicating stronger customer purchasing activity during these periods.
- Autumn shows the lowest sales contribution, suggesting comparatively lower seasonal demand.

Recommendations:

- Amazon India should schedule major sales campaigns, product launches, and marketing activities during Spring and Summer to maximize revenue.
- Targeted discounts and promotional campaigns should be introduced during Autumn and Winter to balance seasonal demand and improve sales consistency.

Problem Statement: 3.2

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*

Approach:

- Selected the order_items table using columns product_id and order_id to analyze product-level sales volume
- Created a CTE to calculate total quantity sold for each product using COUNT(order_id)
- Grouped the data by product_id to organize records at the product level before aggregation
- Used a subquery to calculate the overall average of total_quantity_sold from the CTE
- Filtered products where total quantity sold is greater than the overall average
- Ordered the results in descending order of total quantity sold to display higher-selling products first

SQL Query:

```
with my_cte as (select product_id, count(order_id) as total_quantity_sold
from order_items
group by product_id)
select product_id, total_quantity_sold
from my_cte
where total_quantity_sold > (select avg(total_quantity_sold) from my_cte)
order by total_quantity_sold desc;
```

Output:

	product_id character varying	total_quantity_sold bigint
1	aca2eb7d00ea1a7b8ebd4e6831466...	527
2	99a4788cb24856965c36a24e339b6...	488
3	422879e10f46682990de24d770e7f...	484
4	389d119b48cf3043d311335e499d9...	392
5	368c6c730842d78016ad823897a37...	388
6	53759a2ecddad2bb87a079a1f1519f...	373
7	d1c427060a0f73f6b889a5c7c61f2a...	343
8	53b36df67ebb7c41585e8d54d6772...	323

Recommendations:

- Amazon India should ensure sufficient inventory and faster replenishment for high-selling products to avoid stock-outs and lost sales.
- Low-performing products should be reviewed for pricing, visibility, or promotional improvements to increase sales performance.

Problem Statement: 3.3

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*

Approach

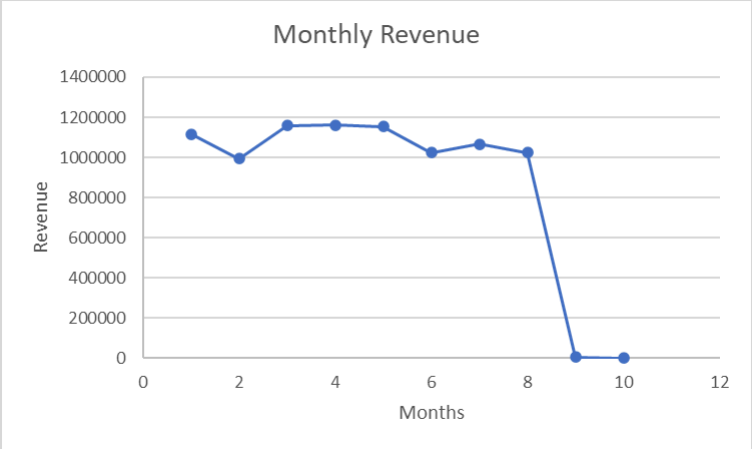
- Selected tables orders and payments using columns order_id, order_purchase_timestamp, and payment_value
- Filtered orders date for the year 2018 using EXTRACT(YEAR FROM order_purchase_timestamp)
- Created a CTE to extract the month from the order purchase timestamp using TO_CHAR( ) for monthly aggregation
- Joined the monthly order data with the payments table using order_id to associate revenue with each order
- Grouped the data by month to organize records before aggregation
- Applied SUM(payment_value) to calculate total revenue generated each month
- Ordered the results by total revenue to identify peak and low sales periods for further visualization in Excel

SQL Query:

```
with my_cte as (select order_id, TO_CHAR(order_purchase_timestamp, 'MM') as months
from orders
where EXTRACT(YEAR FROM order_purchase_timestamp) = 2018
)
select months, sum(P.payment_value) as total_revenue
from my_cte as T1
join payments as P
on T1.order_id = P.order_id
group by months
order by sum(P.payment_value) desc;
```

Output:

	months text	total_revenue numeric
1	01	1115004.18
2	02	992463.34
3	03	1159652.12
4	04	1160785.48
5	05	1153982.15
6	06	1023880.50
7	07	1066540.75
8	08	1022425.32
9	09	4439.54
10	10	589.67



Monthly Revenue

Analysis & Findings:

- Months like April, March, and May generating the highest revenue, indicating peak purchasing periods.
- Certain months (September, October) show significantly lower revenue, highlighting periods of weak customer demand and slower sales activity.

Recommendations:

- Amazon India should align major promotional campaigns, product launches, and inventory planning during peak revenue months to maximize sales.
- Targeted discounts and marketing efforts should be introduced during low-revenue months to stabilize monthly revenue and reduce seasonal fluctuations.

Problem Statement: 3.4

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*

Approach:

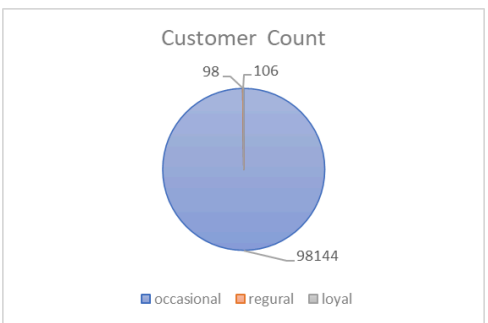
- Selected orders table using customer_id and order_id
- Created an inner CTE to calculate total orders per customer using COUNT(order_id) and grouped the data by customer_id
- Used an outer CTE with a CASE statement to classify customers as Occasional for 1 to 2 orders, Regular for 3 to 5 orders, and Loyal for more than 5 orders
- Grouped the classified data by customer_type to organize records before aggregation
- Applied COUNT(*) to calculate the number of customers in each segment
- Ordered the results in descending order of customer count to show the largest segments first

SQL Query:

```
with T1 as ( with T2 as (select customer_id, count(order_id) as order_counts
from orders group by customer_id) select customer_id, case
when order_counts between 1 and 2 then 'occasional'
when order_counts between 3 and 5 then 'regural'
else 'loyal'
end as customer_type
from T2) select customer_type, count(*)
from T1 group by customer_type order by count(*) desc;
```

Output:

	customer_type text	count bigint
1	occasional	98144
2	regural	106
3	loyal	98



Customer Count

Problem Statement: 3.5

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***

Approach:

- Selected tables orders and payments using columns customer_id, order_id, and payment_value
- Joined both tables using order_id to associate each customer order with its payment amount
- Grouped the data by customer_id to organize records at the customer level before aggregation
- Applied AVG(payment_value) and used ROUND( ) to calculate and standardize the average order value for each customer
- Used the window function DENSE_RANK( ) ordered by average order value in descending order to rank customers based on spending
- Limited the final output to the top 20 customers to identify high-value customers for the rewards program

SQL Query:

```
with my_cte as (select customer_id, Round(avg(payment_value)) as avg_order_value
from orders as O
join payments as P
on O.order_id = P.order_id
group by customer_id)
select customer_id, avg_order_value,
DENSE_RANK () OVER (order by avg_order_value desc)
as customer_rank
from my_cte
limit 20;
```

Output:

	customer_id character varying	avg_order_value numeric	customer_rank bigint
1	1617b1357756262bfa56ab541c47...	13664	1
2	ec5b2ba62e574342386871631fafd...	7275	2
3	c6e2731c5b391845f6800c97401a4...	6929	3
4	f48d464a0baaea338cb25f816991a...	6922	4
5	3fd6777bbce08a352fddd04e4a7cc...	6727	5
6	05455dfa7cd02f13d132aa7a6a972...	6082	6
7	df55c14d1476a9a3467f131269c24...	4950	7
8	e0a2412720e9ea4f26c1ac985f6a7...	4809	8

Problem Statement: 3.6

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 CTE to compute the cumulative sales (total_sales) for each product month by month.* ***Output****: product_id, sale_month, and total_sales*

Approach:

- Selected tables order_items, orders, and payments using columns product_id, order_purchase_timestamp, and payment_value
- Joined order_items with orders using order_id to associate products with their purchase dates, and joined payments using order_id to attach payment values
- Extracted year and month from order_purchase_timestamp using TO_CHAR( ) to create a monthly sales period
- Grouped the data by product_id and sales month to organize records before aggregation
- Applied SUM(payment_value) to calculate total monthly sales for each product
- Used a window function SUM(monthly_sales) OVER (PARTITION BY product_id ORDER BY sales_month) to calculate cumulative sales for each product over time

SQL Query:

```
with table1 as (select OI.product_id,
TO_CHAR(O.order_purchase_timestamp, 'YYYY-MM') AS
sales_month, sum(P.payment_value) as monthly_sales
from order_items as OI
join orders as O
on OI.order_id = O.order_id
join payments as P
on O.order_id = P.order_id
group by product_id, sales_month)
select product_id, sales_month, monthly_sales,
sum(monthly_sales)over(partition by (product_id) order by sales_month)
as Total_sales
from table1;
```

Output:

	product_id character varying	sales_month text	monthly_sales numeric	total_sales numeric
1	00066f42aeeb9f3007548bb9d3f33c38	2018-05	120.24	120.24
2	00088930e925c41fd95ebfe695fd2655	2017-12	143.83	143.83
3	0009406fd7479715e4bef61dd91f2462	2017-12	242.1	242.1
4	000b8f95fcb9e0096488278317764d...	2018-08	157.0	157.0
5	000d9be29b5207b54e86aa1b1ac54...	2018-04	218.27	218.27
6	0011c512eb256aa0dbbb544d8dffcf6e	2017-12	166.4	166.4
7	00126f27c813603687e6ce486d909d...	2017-09	527.73	527.73
8	001795ec6f1b187d37335e1c470476	2017-10	47.62	47.62

Analysis & Findings:

- Cumulative sales increase month by month for active products, indicating steady sales progression after the product's first sale.
- Some products show limited growth after initial months, suggesting weaker long-term demand or shorter product lifecycle.

Recommendations:

- Amazon India should focus promotions and inventory planning on products showing consistent cumulative growth to maximize long-term revenue.
- Products with slow cumulative growth should be evaluated for pricing adjustments, better visibility, or promotional support to improve lifecycle performance.

Problem Statement: 3.7

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.*

Approach:

- Selected tables payments and orders using columns payment_type, order_purchase_timestamp, and payment_value
- Joined both tables using order_id to associate payment information with order dates
- Filtered records for the year 2018 using EXTRACT(YEAR FROM order_purchase_timestamp)
- Extracted year and month using TO_CHAR( ) and grouped the data by payment_type and sale month to organize records before aggregation
- Applied SUM(payment_value) to calculate total monthly sales for each payment type
- Used the window function LAG(monthly_sales) partitioned by payment_type and ordered by month to obtain previous month sales
- Calculated month-over-month percentage change and used NULLIF() to avoid division by zero errors

SQL Query:

```
with table1 as (select P.payment_type, TO_CHAR(order_purchase_timestamp, 'YYYY-MM') as
sale_month, sum(payment_value) as monthly_sales
from payments as P
Join orders as O
On O.order_id = P.order_id
where EXTRACT(YEAR FROM order_purchase_timestamp) = 2018
group by payment_type, sale_month
order by payment_type, sale_month),
table2 as (
select payment_type, sale_month, monthly_sales,
LAG(monthly_sales) over(partition by payment_type order by sale_month) as
previous_month_sales
```

```
from table1)
select payment_type, sale_month, monthly_sales,
ROUND((monthly_sales - previous_month_sales)*100/NULLIF(previous_month_sales,0), 2) AS
monthly_change
FROM table2;
```

Output:

	payment_type character varying	sale_month text	monthly_sales numeric	monthly_change numeric
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	143805.90	-27.39
9	credit_card	2018-01	868880.38	[null]

Analysis & Findings:

- Monthly sales growth varies across payment types, with both positive and negative changes observed during the year, indicating fluctuating customer spending behavior.
- Credit card payments contribute higher monthly sales values, while other payment methods show sharper month-to-month variations.

Recommendations:

- Amazon India should continue promoting high-performing payment methods through cashback or EMI offers to maintain stable sales growth.
- Payment methods showing frequent negative growth should be supported with targeted offers or incentives during weaker months to stabilize overall revenue trends.