**Q1 )**

**1) Total Quantity Sold for Each Stock Code Over Time :**

**Query :**

select stockcode, customer\_id, invoicedate, quantity,

sum(quantity) over(partition by stockcode order by invoicedate) as total\_quantity

from tableretail

order by stockcode,invoicedate;

**Business Meaning :**

This query calculates the total quantity sold for each stock code over time and presents the cumulative total quantity sold using a window function. It helps in analyzing the sales trend of each stock code over time.

-----------------------------------------------------------------------------------------------------------------------------

**2) Ranking Customers by Total Sales :**

**Query :**

with cte as (

select customer\_id, sum(quantity\*price) as sales

from tableretail

group by customer\_id

)

select customer\_id, sales, rank() over(order by sales desc) as "Rank"

from cte;

**Business Meaning :**

This query ranks customers based on their total sales. It provides insights into which customers are the top spenders and helps in identifying high-value customers for targeted marketing or loyalty programs.

**3) Ranking Products by Order Count :**

**Query :**

with cte as (

select stockcode,count(stockcode) as OrderCount

from tableretail

group by stockcode

)

select stockcode, OrderCount, dense\_rank() over(order by OrderCount desc) as "Rank"

from cte;

**Business Meaning :**

This query ranks products based on the number of orders they appear in. It helps in identifying the most popular products and can guide inventory management and marketing strategies.

-----------------------------------------------------------------------------------------------------------------------------

**4) Ranking Products by Quantity Sold :**

**Query :**

with cte as (

select stockcode,sum(quantity) as QuantitySold

from tableretail

group by stockcode

)

select stockcode, QuantitySold, dense\_rank() over(order by QuantitySold desc) as "Rank"

from cte;

**Business Meaning :**

This query ranks products based on the total quantity sold. It provides insights into product popularity and helps in identifying top-selling items for inventory management and marketing efforts.

**5) Ranking Products by Total Sales :**

**Query :**

with cte as (

select stockcode,sum(quantity\*price) as TotalSales

from tableretail

group by stockcode

)

select stockcode, TotalSales, dense\_rank() over(order by TotalSales desc) as "Rank"

from cte;

**Business Meaning :**

This query ranks products based on their total sales revenue. It helps in identifying the most profitable products and can guide pricing strategies and inventory decisions.

-----------------------------------------------------------------------------------------------------------------------------

**6) Average Sales Each Year for Each Customer :**

**Query :**

select invoice,customer\_id,invoicedate,extract(year from to\_date(invoicedate, 'MM/DD/YYYY HH24:MI')) as Year,quantity,price,

round(avg(quantity \* price) over (partition by customer\_id, extract(year from to\_date(invoicedate, 'MM/DD/YYYY HH24:MI'))),2) as AvgSalesEachYear

from tableRetail;

**Business Meaning :**

This query calculates the average sales for each customer each year. It helps in understanding customer purchasing patterns over time and can inform customer relationship management strategies.

**7) Average Quantity Each Year for Each Product :**

**Query :**

select invoice,stockcode,invoicedate,extract(year from to\_date(invoicedate, 'MM/DD/YYYY HH24:MI')) as Year,quantity,price,

avg(quantity) over (partition by stockcode, extract(year from to\_date(invoicedate, 'MM/DD/YYYY HH24:MI'))) as AvgQtyEachYear

from tableRetail;

**Business Meaning :**

This query calculates the average quantity sold for each product each year. It helps in analyzing product demand trends over time and can guide inventory planning and procurement decisions.

-----------------------------------------------------------------------------------------------------------------------------

**8) Total Spent Each Month by Each Customer :**

**Query :**

select invoice, customer\_id, invoicedate,

extract(year from to\_date(invoicedate, 'MM/DD/YYYY HH24:MI')) as Year,

extract(month from to\_date(invoicedate, 'MM/DD/YYYY HH24:MI')) as Month,quantity,price,

sum(quantity \* price) over (partition by customer\_id, extract(year from to\_date(invoicedate, 'MM/DD/YYYY HH24:MI')),

extract (month from to\_date(invoicedate, 'MM/DD/YYYY HH24:MI'))) as TotalSpentEachMonth

from tableRetail;

**Business Meaning :**

This query calculates the total amount spent by each customer each month. It helps in analyzing customer spending patterns over time and can inform marketing campaigns and promotions targeting specific time periods.

**9) Cumulative Rank of Customers by Total Sales :**

**Query :**

with cte as (

select customer\_id, sum(quantity\*price) as sales

from tableretail

group by customer\_id

)

select customer\_id, sales, round(cume\_dist() over(order by sales desc),3) as "CumulativeRank"

from cte;

**Business Meaning :**

This query calculates the cumulative rank of customers based on their total sales. It provides a cumulative distribution of customer sales and helps in understanding the distribution of customer spending across different segments.

**Q2)**

**Code :**

with

dates\_cte as (

select customer\_id,count(distinct(invoice)) over(partition by customer\_id) as frequency,

sum(quantity\*price) over (partition by customer\_id) as total\_sales ,

first\_value(to\_date(invoicedate, 'MM/DD/YYYY HH24:MI')) over (order by to\_date(invoicedate, 'MM/DD/YYYY HH24:MI') desc) as recent\_date,

first\_value(to\_date(invoicedate, 'MM/DD/YYYY HH24:MI')) over (partition by customer\_id order by to\_date(InvoiceDate, 'MM/DD/YYYY HH24:MI') desc) as last\_date

from tableretail),

r\_cte as (

select customer\_id, frequency, total\_sales,

round(recent\_date - last\_date,0) as recency

from dates\_cte

group by customer\_id,frequency,total\_sales,round(recent\_date - last\_date,0)),

fm\_cte as (

select customer\_id, recency, frequency,

round(percent\_rank() over (order by total\_sales),2) as monetary, (frequency+round(percent\_rank() over (order by total\_sales),2))/2 as avg\_fm from r\_cte),

rfm\_scores\_cte as (

select customer\_id, recency, frequency, monetary,

ntile(5) over(order by recency desc) as r\_score,

ntile(5) over(order by avg\_fm) as fm\_score

from fm\_cte)

select customer\_id,recency,frequency,monetary,r\_score,fm\_score,

case

when r\_score = 5 and fm\_score = 5 then ' Champions'

when r\_score = 5 and fm\_score = 4 then ' Champions'

when r\_score = 4 and fm\_score = 5 then ' Champions'

when r\_score = 5 and fm\_score = 2 then 'Potential Loyalists'

when r\_score = 4 and fm\_score = 2 then 'Potential Loyalists'

when r\_score = 3 and fm\_score = 3 then 'Potential Loyalists'

when r\_score = 4 and fm\_score = 3 then 'Potential Loyalists'

when r\_score = 5 and fm\_score = 3 then 'Loyal Customers'

when r\_score = 4 and fm\_score = 4 then 'Loyal Customers'

when r\_score = 3 and fm\_score = 5 then 'Loyal Customers'

when r\_score = 3 and fm\_score = 4 then 'Loyal Customers'

when r\_score = 5 and fm\_score = 1 then 'Recent Customers'

when r\_score = 4 and fm\_score = 1 then 'Promising'

when r\_score = 3 and fm\_score = 1 then 'Promising'

when r\_score = 3 and fm\_score = 2 then 'Customers Needing Attention'

when r\_score = 2 and fm\_score = 3 then 'Customers Needing Attention'

when r\_score = 2 and fm\_score = 2 then 'Customers Needing Attention'

when r\_score = 2 and fm\_score = 5 then 'At Rsik'

when r\_score = 2 and fm\_score = 4 then 'At Risk'

when r\_score = 1 and fm\_score = 3 then 'At Risk'

when r\_score = 1 and fm\_score = 5 then 'Can not Lose Them'

when r\_score = 1 and fm\_score = 4 then 'Can not Lose Them'

when r\_score = 1 and fm\_score = 2 then 'Hibernating'

when r\_score = 1 and fm\_score = 1 then 'Lost'

else 'UnCategorized'

end as cust\_segment

from rfm\_scores\_cte order by customer\_id;

**Explanation :**

dates\_cte:

This CTE calculates the frequency and total sales for each customer, along with their most recent and last purchase dates.

r\_cte:

This CTE calculates the recency for each customer by finding the difference between the most recent purchase and their last purchase dates.

fm\_cte:

This CTE calculates the monetary scores for each customer, and average score between frequency and monetary.

rfm\_scores\_cte:

This CTE assigns RFM scores to each customer based on their recency, frequency, and monetary scores. These scores are divided into quintiles (five equal groups) to rank customers.

Final Query:

The final query selects customer ID along with their recency, frequency, monetary, RFM scores, and segments them based on their RFM scores into different categories such as "Champions", "Potential Loyalists", "Loyal Customers", etc.

**Q3 )**

1. **Code :**

with group\_cte as (

select cust\_id, calendar\_dt, amt\_le,

row\_number() over (partition by cust\_id order by calendar\_dt) as "Rank", calendar\_dt - row\_number() over (partition by cust\_id order by calendar\_dt) as "Group"

from daily\_transactions

where amt\_le > 0

),

consecutive\_cte as (

select cust\_id,"Group",count(\*) as Consecutive\_Days

from group\_cte group by cust\_id,"Group"

)

select cust\_id, max(Consecutive\_Days) as Max\_Consecutive\_Days

from consecutive\_cte

group by cust\_id

order by cust\_id;

**Explanation :**

group\_cte CTE:

* + This CTE selects data from the table daily\_transactions.
  + It filters the transactions to include only those where amt\_le (amount) is greater than 0, which means we are excluding transactions with zero or negative amounts.
  + It uses the row\_number() window function to assign a sequential rank to each transaction within each cust\_id, ordered by calendar\_dt. This rank is stored in the column named "Rank".
  + It calculates a "Group" value for each transaction. This is done by subtracting the row\_number() from the calendar\_dt, effectively grouping transactions that happened on consecutive days together. This helps us identify consecutive days of transactions for each customer.

consecutive\_cte CTE:

* This CTE further processes the data from the group\_cte.
* It groups the data by cust\_id and the calculated "Group" value.
* For each group, it counts the number of consecutive days (Consecutive\_Days) a customer made transactions.

Main Query:

* In the main query, we select data from the consecutive\_cte.
* We group the data by cust\_id.
* We find the maximum value of Consecutive\_Days for each cust\_id, which gives us the maximum number of consecutive days a customer made transactions.
* Finally, we order the result by cust\_id.

1. **Code :**

with cte as (

select cust\_id, calendar\_dt, amt\_le,

sum(amt\_le) over(partition by cust\_id order by calendar\_dt) as "SUM", row\_number() over(partition by cust\_id order by calendar\_dt) as "Rank",

min(calendar\_dt) over (partition by cust\_id order by calendar\_dt) as first\_purchase

from daily\_transactions

order by cust\_id

),

cte2 as (

select cte.\*,

case

when "SUM" >= 250 then "Rank"

end as transactions,

case

when "SUM" >= 250 then calendar\_dt - first\_purchase

end as days

from cte

),

cte3 as (

select cust\_id, min(transactions) as chosen\_transaction ,

min(days) as chosen\_day

from cte2

group by cust\_id

)

select round(avg(chosen\_transaction),0) as avg\_transactions , round(avg(chosen\_day),0) as avg\_days

from cte3;

**Explanation :**

First CTE (cte):

* + This CTE selects data from the table daily\_transactions.
  + It calculates the cumulative sum of amt\_le (amount) over each partition of cust\_id, ordered by calendar\_dt, and names it "SUM".
  + It assigns a row number to each transaction within each cust\_id partition, ordered by calendar\_dt, and names it "Rank".
  + It also determines the first purchase date (first\_purchase) for each cust\_id.

Second CTE (cte2):

* + This CTE extends the first CTE (cte).
  + It incorporates conditional logic (CASE statements) to determine values for transactions and days columns based on the cumulative sum ("SUM") being greater than or equal to 250.
  + If the sum is greater than or equal to 250, it assigns the value of "Rank" to transactions and calculates the difference between the current calendar\_dt and the first\_purchase date, storing it in days.

Third CTE (cte3):

* + This CTE further processes the data from cte2.
  + It groups the data by cust\_id.
  + For each group, it selects the minimum value of transactions as chosen\_transaction and the minimum value of days as chosen\_day.

The minimum is selected here to get the first value that reached the thresold

Main Query:

* + In the main query, we select data from cte3.
  + We calculate the average of chosen\_transaction as avg\_transactions and the average of chosen\_day as avg\_days, rounding them to zero decimal places.