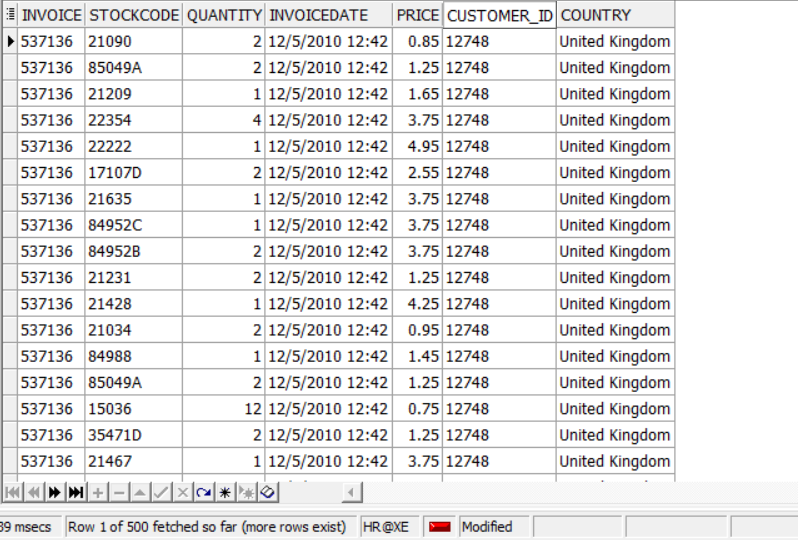
**Analytical SQL Case Study**

****

**we will answer question using SQL Analytical functions we have learnt in the course.**

**Q1- Using OnlineRetail dataset**

**• write at least 5 analytical SQL queries that tells a story about the data**

**• write small description about the business meaning behind each query**

**Query 1**

SELECT StockCode, TotalQuantitySold

FROM (

SELECT

StockCode,

SUM(Quantity) AS TotalQuantitySold,

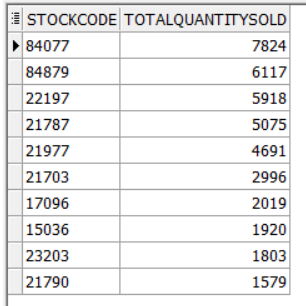
RANK() OVER (ORDER BY SUM(Quantity) DESC) AS Rank

FROM tableRetail

GROUP BY StockCode

) ranked\_data

WHERE Rank <= 10;



**Business Meaning:** Identifying the top-selling products helps the company understand customer preferences and demand. It guides inventory stocking decisions, highlights popular items for promotion, and informs product development strategies.

**Query 2**

SELECT Customer\_ID, TotalPurchases

FROM (

SELECT

Customer\_ID,

COUNT(DISTINCT Invoice) AS TotalPurchases,

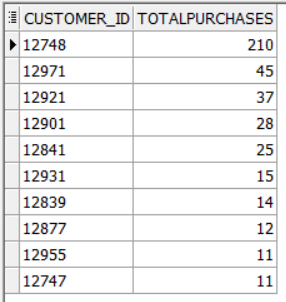
RANK() OVER (ORDER BY COUNT(DISTINCT Invoice) DESC) AS PurchaseRank

FROM tableRetail

GROUP BY Customer\_ID

) ranked\_data

WHERE PurchaseRank <= 10;

****

***Business Meaning:*** *This query identifies the most frequent customers, indicating their loyalty and potential for targeted marketing campaigns, loyalty programs, and personalized offers to increase retention and lifetime value.*

**Query 3**

WITH MonthlyData AS (

SELECT

SUBSTR(TO\_CHAR(TO\_DATE(InvoiceDate, 'MM/DD/YYYY HH24:MI'), 'DD/MM/YYYY'), 4, 9) AS Month,

SUM(Price \* Quantity) AS MonthlyRevenue

FROM tableRetail

GROUP BY SUBSTR(TO\_CHAR(TO\_DATE(InvoiceDate, 'MM/DD/YYYY HH24:MI'), 'DD/MM/YYYY'), 4, 9)

)

SELECT

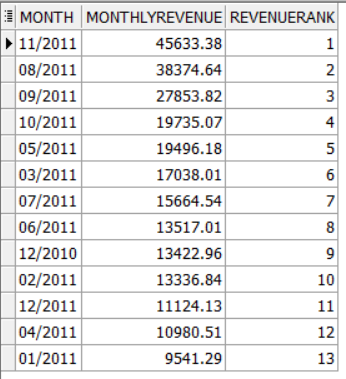
Month,

MonthlyRevenue,

RANK() OVER (ORDER BY MonthlyRevenue DESC) AS RevenueRank

FROM MonthlyData

ORDER BY RevenueRank;

Business Meaning: This query provides insights into the revenue generated by retail transactions monthly. It allows businesses to identify the months with the highest and lowest revenue. By analyzing the monthly sales trend, the company can identify seasonal patterns, peak sales periods, and overall sales performance over time. This information can guide inventory management, staffing, and promotional efforts.

**Query 4**

SELECT

DISTINCT TO\_CHAR(TO\_DATE(InvoiceDate, 'MM/DD/YYYY HH24:MI'), 'YYYY-MM') AS InvoiceMonth,

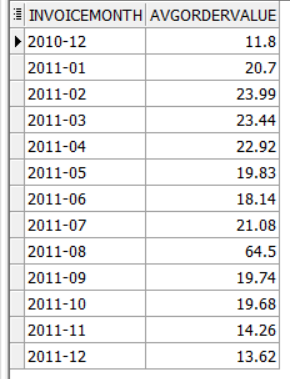
ROUND(AVG(Quantity \* Price) OVER (PARTITION BY TO\_CHAR(TO\_DATE(InvoiceDate, 'MM/DD/YYYY HH24:MI'), 'YYYY-MM')), 2) AS AvgOrderValue

FROM

tableRetail

ORDER BY

TO\_CHAR(TO\_DATE(InvoiceDate, 'MM/DD/YYYY HH24:MI'), 'YYYY-MM');



Business Meaning: This query calculates the average order value (AOV) per month over the dataset. It focuses on understanding the average spending per order over time, which can indicate changes in customer behavior or purchasing patterns. The output is the average order value for each month, providing insights into whether customers are spending more or less per transaction over time.

**Query 5**

SELECT

InvoiceMonth,

MonthlyRevenue,

PreviousMonthRevenue,

GrowthRate

FROM (

SELECT

InvoiceMonth,

SUM(MonthlyRevenue) AS MonthlyRevenue,

LAG(SUM(MonthlyRevenue)) OVER (ORDER BY InvoiceMonth) AS PreviousMonthRevenue,

ROUND(((SUM(MonthlyRevenue) - LAG(SUM(MonthlyRevenue)) OVER (ORDER BY InvoiceMonth))

/ LAG(SUM(MonthlyRevenue)) OVER (ORDER BY InvoiceMonth)) \* 100, 2

) AS GrowthRate

FROM (

SELECT

TO\_CHAR(TO\_DATE(InvoiceDate, 'MM/DD/YYYY HH24:MI'), 'YYYY-MM') AS InvoiceMonth,

Quantity \* Price AS MonthlyRevenue

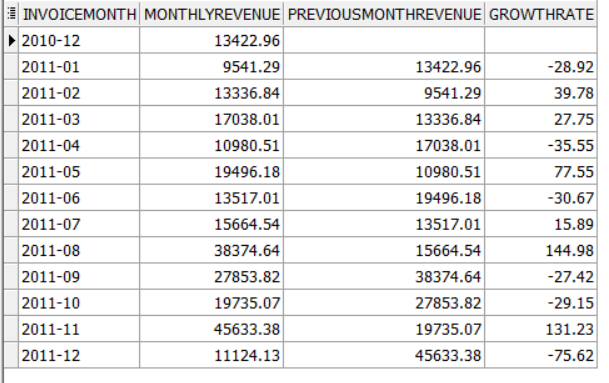
FROM

tableRetail) subquery

GROUP BY

InvoiceMonth) subquery2

ORDER BY InvoiceMonth;



Business Meaning: This query calculates the monthly sales growth rate by comparing the revenue of each month with the revenue of the previous month. It helps in understanding the rate of change in sales over time, which is crucial for making informed business decisions and setting realistic sales targets.

**Query 6**

WITH MonthlyData AS (

SELECT

Customer\_ID,

TO\_CHAR(TO\_DATE(InvoiceDate, 'MM/DD/YYYY HH24:MI'), 'YYYY-MM') AS Month,

SUM(Quantity \* Price) AS MonthlyRevenue,

COUNT(DISTINCT Invoice) AS MonthlyOrders

FROM

tableRetail

GROUP BY

Customer\_ID, TO\_CHAR(TO\_DATE(InvoiceDate, 'MM/DD/YYYY HH24:MI'), 'YYYY-MM')

),

CustomerSummary AS (

SELECT

Customer\_ID,

SUM(MonthlyRevenue) AS TotalRevenue,

SUM(MonthlyOrders) AS TotalOrders,

COUNT(DISTINCT Month) AS TotalMonths

FROM

MonthlyData

GROUP BY

Customer\_ID

)

SELECT

Customer\_ID,

TotalRevenue,

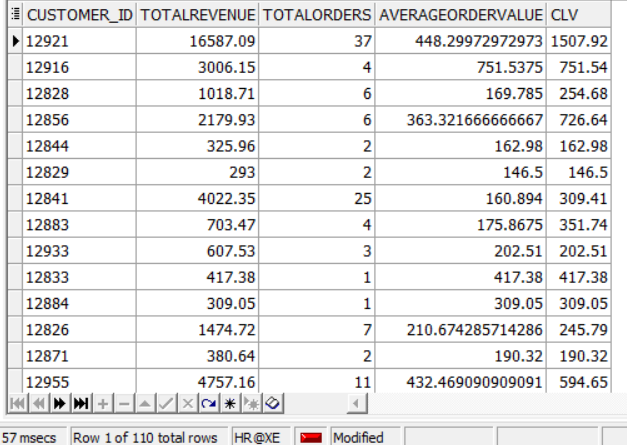
TotalOrders,

TotalRevenue / TotalOrders AS AverageOrderValue,

ROUND((TotalRevenue / TotalOrders) \* (TotalOrders / TotalMonths), 2) AS CLV

FROM

CustomerSummary;



Business Meaning: This query calculates the Customer Lifetime Value (CLV) by estimating the total revenue generated by each customer over their entire relationship with the business. It helps in identifying high-value customers and allocating resources effectively to maximize long-term profitability.

**Query 7**

select customer\_id , invoiceday , customer\_sales , sum(customer\_sales) over (partition by customer\_id order by invoiceday) as total\_sales

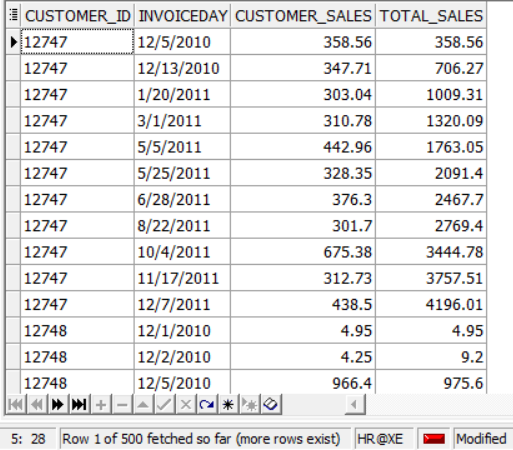
from

(select customer\_id , TRUNC(TO\_DATE(invoicedate, 'MM/DD/YYYY HH24:MI')) as invoiceday , sum(quantity \* price) as customer\_sales

from tableRetail

group by customer\_id , TRUNC(TO\_DATE(invoicedate, 'MM/DD/YYYY HH24:MI') )

order by customer\_id , invoiceday )



Business Meaning: This query track customer sales over dates

**Query 8**

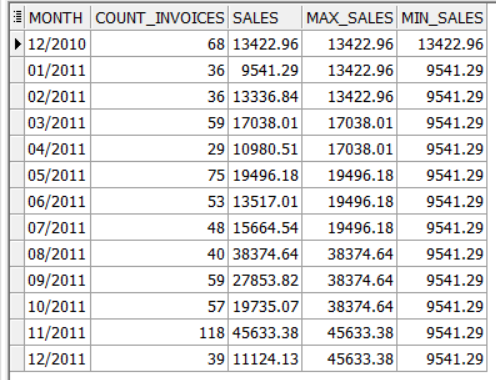
with months\_sales AS (select sum(quantity \* price) as sales , TO\_CHAR(TRUNC(TO\_DATE(invoicedate, 'MM/DD/YYYY HH24:MI')), 'MM/YYYY') as month ,count(distinct invoice) as count\_invoices from tableRetail

group by TO\_CHAR(TRUNC(TO\_DATE(invoicedate, 'MM/DD/YYYY HH24:MI')), 'MM/YYYY'))

select month ,count\_invoices , sales , Max(sales) over(order by to\_date(month , 'MM/YYYY')asc ) as max\_sales , Min(sales) over(order by to\_date(month , 'MM/YYYY')asc ) as min\_sales

from months\_sales

order by to\_date(month , 'MM/YYYY') asc;



Business Meaning: t*rack months sales and invoices and customers count*

Q2- After exploring the data now you are required to implement a Monetary model for customers behavior for product purchasing and segment each customer based on the below groups Champions - Loyal Customers - Potential Loyalists – Recent Customers – Promising - Customers Needing Attention - At Risk - Cant Lose Them – Hibernating – Lost The customers will be grouped based on 3 main values

• Recency => how recent the last transaction is (Hint: choose a reference date, which is the most recent purchase in the dataset )

• Frequency => how many times the customer has bought from our store

• Monetary => how much each customer has paid for our products As there are many groups for each of the R, F, and M features, there are also many potential permutations, this number is too much to manage in terms of marketing strategies.

For this, we would decrease the permutations by getting the average scores of the frequency and monetary (as both of them are indicative to purchase volume anyway)

Here’s the query:

with maxinvoicedate as (select max(to\_timestamp(invoicedate, 'MM/DD/YYYY HH24:MI')) as max\_date

from tableretail),

customerrfm as (

select customer\_id,

ceil(extract(day from (select max\_date from maxinvoicedate) - max(to\_timestamp(invoicedate, 'MM/DD/YYYY HH24:MI')))) as recency,

count(distinct invoice) as frequency,

round(sum(quantity \* price) / 1000, 2) as monetary

from tableretail

group by customer\_id),

rfmscores as (

select customer\_id, recency,frequency, monetary,

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

ntile(5) over(order by frequency) as f\_score,

ntile(5) over(order by monetary) as m\_score,

round((ntile(5) over(order by frequency) + ntile(5) over(order by monetary))/2,0) as fm\_score

from customerrfm),

customersegment as (

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

case

when (r\_score >= 5 and fm\_score >= 5) or (r\_score >= 5 and fm\_score = 4) or (r\_score = 4 and fm\_score >= 5) then 'champions'

when (r\_score >= 5 and fm\_score = 2) or (r\_score = 4 and fm\_score = 2) or (r\_score = 3 and fm\_score = 3) or (r\_score = 4 and fm\_score >= 3) then 'potential loyalists'

when (r\_score >= 5 and fm\_score = 3) or (r\_score = 4 and fm\_score = 4) or (r\_score = 3 and fm\_score >= 5) or (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) or (r\_score = 3 and fm\_score = 1) then 'promising'

when (r\_score = 3 and fm\_score = 2) or (r\_score = 2 and fm\_score = 3) or (r\_score = 2 and fm\_score = 2) then 'customers needing attention'

when (r\_score = 2 and fm\_score >= 5) or (r\_score = 2 and fm\_score = 4) or (r\_score = 1 and fm\_score = 3) then 'at risk'

when (r\_score = 1 and fm\_score >= 5) or (r\_score = 1 and fm\_score = 4) then 'cant lose them'

when (r\_score = 1 and fm\_score = 2) or (r\_score = 2 and fm\_score = 1) then 'hibernating'

when r\_score = 1 and fm\_score <= 1 then 'lost'

else 'other'

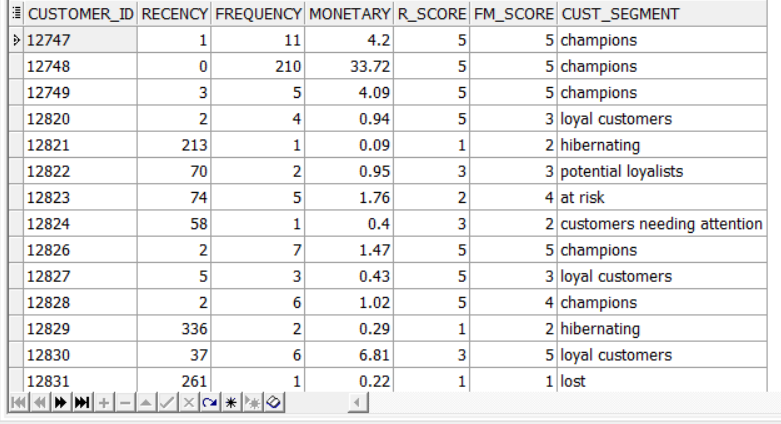
end as cust\_segment

from rfmscores)

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

from customersegment

order by customer\_id;



Here's an alternative explanation:

1. \*\*Finding the Latest Invoice Date:\*\*

- We start by finding the most recent invoice date in our dataset. This helps determine how recent customers' transactions are.

2. \*\*Calculating RFM Metrics:\*\*

- Next, we compute three key metrics for each customer:

- Recency (R): This tells us how recently a customer made a purchase. It's calculated by finding the difference between the latest invoice date and each customer's individual purchase date, rounded up to the nearest day.

- Frequency (F): This indicates how often a customer buys from us. We simply count the number of distinct invoices for each customer.

- Monetary (M): This reflects the value of a customer's purchases. It's calculated as the total monetary value of a customer's purchases, represented as a percentage of the total monetary value across all customers.

3. \*\*Assigning RFM Scores:\*\*

- Using the RFM metrics we calculated, we assign scores to each customer. We divide customers into five groups (quintiles) based on their recency, frequency, and monetary values.

4. \*\*Segmenting Customers:\*\*

- With the RFM scores in hand, we segment customers into different categories based on their scores. Each segment represents a distinct group of customers with similar purchasing behaviors.

- For example, "champions" are customers with high recency and high monetary/frequency scores, indicating they are highly engaged and valuable to the business. On the other hand, "at-risk" customers may have low recency and monetary/frequency scores, suggesting they need attention to prevent them from disengaging.

5. \*\*Final Output:\*\*

- Finally, we present the results by selecting customer IDs along with their recency, frequency, monetary values, RFM scores, and assigned customer segments. The output is sorted by customer ID.

In essence, this SQL code enables businesses to understand and categorize their customers based on their purchasing patterns, helping them tailor marketing strategies and customer engagement efforts for improved satisfaction and retention.

Q3: Daily purchasing transactions for customers

1) What is the maximum number of consecutive days a customer made purchases?

with ranked\_transactions as (

select cust\_id,to\_date(calendar\_dt, 'YYYY-MM-DD') as calendar\_dt,

row\_number() over (partition by cust\_id order by to\_date(calendar\_dt, 'YYYY-MM-DD')) as rn

from transactions),

consecutive\_days as (

select cust\_id, calendar\_dt, calendar\_dt - rn as grp

from ranked\_transactions)

select cust\_id, max(count\_consecutive\_days) as max\_consecutive\_days

from (

select cust\_id, count(\*) as count\_consecutive\_days from consecutive\_days

group by cust\_id, grp ) max\_consecutive\_days\_per\_group

group by cust\_id

order by cust\_id;

A screenshot of a computer

Description automatically generated

Description:

This SQL query harnesses Common Table Expressions (CTEs) to delve into transactional data, pinpointing the longest streak of consecutive days each customer engaged in purchases. By meticulously assigning row numbers and crafting groupings that encapsulate uninterrupted sequences of transactions, it unveils patterns in customer behavior. Armed with these revelations, businesses can craft laser-focused marketing campaigns, fine-tune inventory strategies, and fortify customer loyalty initiatives. Ultimately, such insights pave the way for sustained business expansion and heightened profitability.

2) On average, How many days/transactions does it take a customer to reach a spent threshold of 250 L.E?

WITH cte\_1 AS (

SELECT

cust\_id,

"CALENDAR\_DT",

AMT\_L,

SUM(AMT\_L) OVER (PARTITION BY cust\_id ORDER BY "CALENDAR\_DT") AS total\_amount

FROM transactions

),

cte\_2 AS (

SELECT

cust\_id,

"CALENDAR\_DT",

AMT\_L,

total\_amount,

CASE WHEN total\_amount >= 250 THEN 1 ELSE 0 END AS reached\_threshold

FROM cte\_1

WHERE total\_amount > 250

),

cte\_3 AS (

SELECT

cust\_id,

"CALENDAR\_DT",

AMT\_L,

total\_amount,

CASE WHEN total\_amount < 250 THEN 1 ELSE 0 END AS before\_reached\_threshold

FROM cte\_1

WHERE total\_amount <= 250

),

final AS (

SELECT

c.cust\_id,

COUNT(c.before\_reached\_threshold) AS before\_reaching\_threshold

FROM cte\_3 c

WHERE c.cust\_id IN (SELECT cust\_id FROM cte\_2)

GROUP BY c.cust\_id

)

SELECT

AVG(before\_reaching\_threshold) AS avg\_before\_reaching\_threshold

FROM final;

A close up of a message

Description automatically generated

Description

1. \*\*cte\_1\*\*: This CTE calculates the cumulative total amount spent by each customer over time.

2. \*\*cte\_2\*\*: Filters out the transactions where the total amount spent by the customer exceeds 250 and assigns a flag (1 or 0) accordingly.

3. \*\*cte\_3\*\*: Filters out the transactions where the total amount spent by the customer is less than or equal to 250 and assigns a flag (1 or 0) accordingly.

4. \*\*final\*\*: Counts the number of transactions made by each customer before reaching the spending threshold.

5. Finally, the main query calculates the average count of transactions made by customers before reaching the spending threshold.