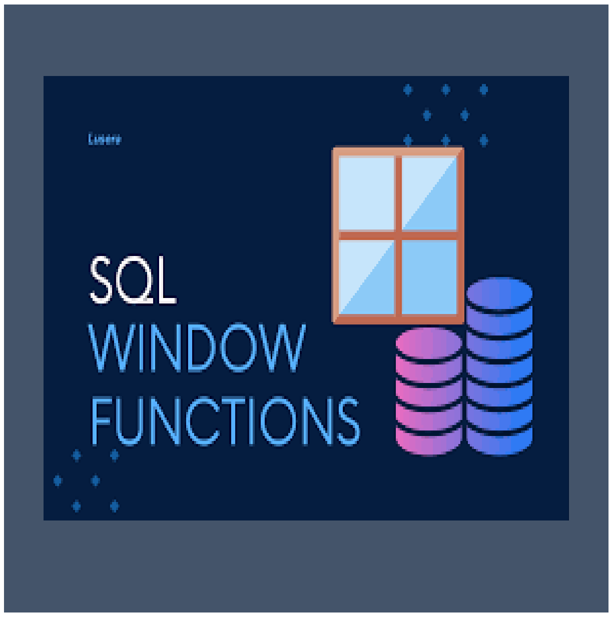


Analytical SQL

Case Study

Israa Ali Ahmed
Data Management - ITI

Information
Technology
Institute

Overview

Customers has purchasing transaction that we shall be monitoring to get intuition behind each customer behavior to target the customers in the most efficient and proactive way, to increase sales/revenue, improve customer retention and decrease churn.

Objective

This case study offers a comprehensive analysis of customer transactions, aiming to provide actionable insights for businesses to improve their marketing strategies, inventory management, customer retention, and pricing.

Dataset

Our dataset consists of 12,858 records, containing information such as invoice numbers, stock codes, quantities, dates, prices, customer IDs, and countries.

Information
Technology
Institute

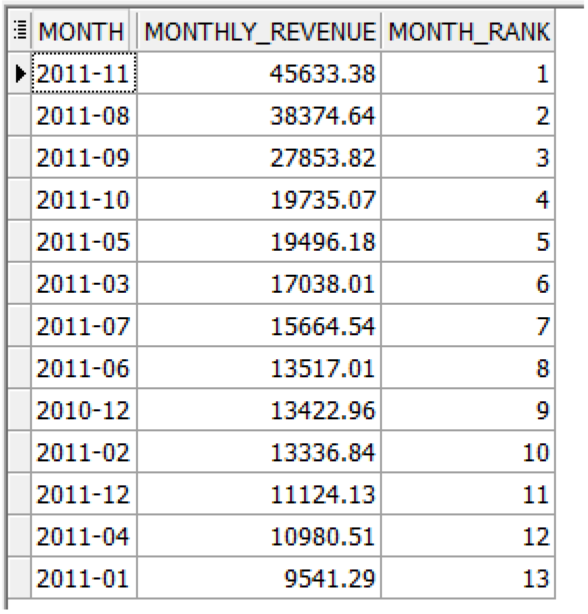
Q1:

1) Identifying Seasonal Trends in Revenue:

This query calculates the monthly revenue for each distinct month in the dataset and ranks them based on their revenue, presenting the result set with the highest revenue months appearing first. Helping in identifying seasonal patterns and assessing the effectiveness of marketing efforts over time.

- **Query:**

```
with cte as (
-- Select distinct month-year combinations from the 'InvoiceDate' column and calculate total revenue for each month-year combination
  select distinct
    to_char(to_date(InvoiceDate, 'MM/DD/YYYY HH24:MI'), 'YYYY-MM') as month,
-- Calculate monthly revenue by summing up Quantity * Price over each month-year partition
    sum(Quantity * Price) over (partition by to_char(to_date(InvoiceDate, 'MM/DD/YYYY HH24:MI'), 'YYYY-MM')) as monthly_revenue
  from tableRetail
)
-- Main Query
Select
  -- Select the month and its corresponding monthly revenue from cte
  month,
  monthly_revenue,
-- Assign a rank to each month based on its revenue (higher revenue gets lower rank)
  rank() over (order by monthly_revenue desc) as month_rank
from cte
-- Order the result set by the month rank in ascending order
order by month_rank;
```



MONTH	MONTHLY_REVENUE	MONTH_RANK
2011-11	45633.38	1
2011-08	38374.64	2
2011-09	27853.82	3
2011-10	19735.07	4
2011-05	19496.18	5
2011-03	17038.01	6
2011-07	15664.54	7
2011-06	13517.01	8
2010-12	13422.96	9
2011-02	13336.84	10
2011-12	11124.13	11
2011-04	10980.51	12
2011-01	9541.29	13

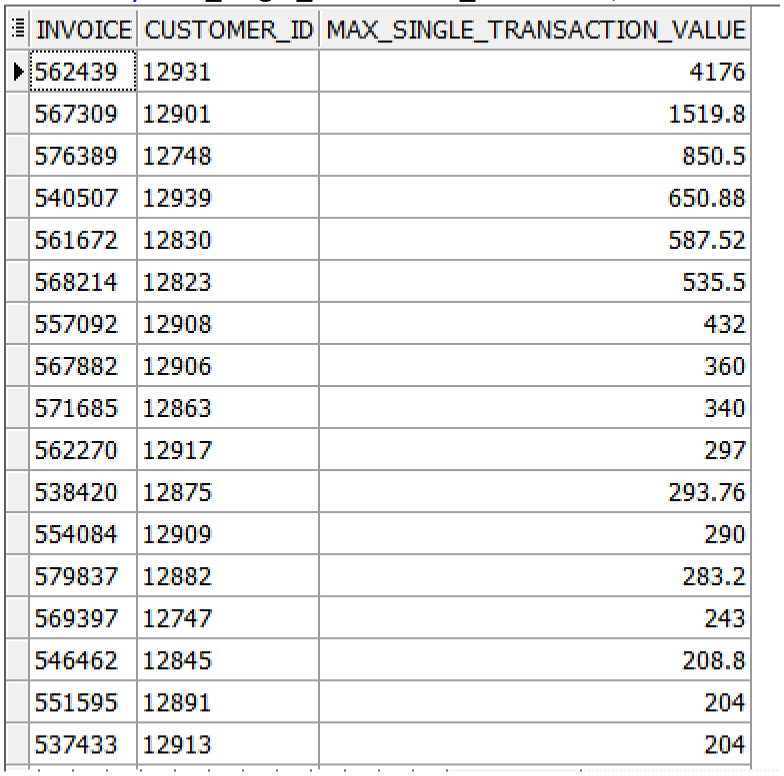
2) The Highest Single Transaction value for each customer

This query retrieves the maximum value of a single transaction for each customer from the tableRetail table and order the result to show customers with the highest single transaction value.

- **Query:**

```
with cte as (
-- select relevant columns and calculate the transaction value for each invoice
   select
      invoice,
      customer_id,
      quantity * price as transaction_value,
      -- assign a row number for each row within each partition of 'customer_id', ordering them
by  the transaction value for each invoice for him in descending order
      row_number() over (partition by customer_id order by quantity * price desc) as row_num
   from tableretail
)
select
   invoice,
   customer_id,
   transaction_value AS max_single_transaction_value
from  cte
 /* filter the results to only include rows where the row number is equal to 1,
meaning it selects only the invoices with the highest  transaction value for each customer
and order the results by the transaction_value desc
```

```
meaning the first row represents the customer with the highest single transaction
values */
where row_num = 1
  order by max_single_transaction_value desc;
```



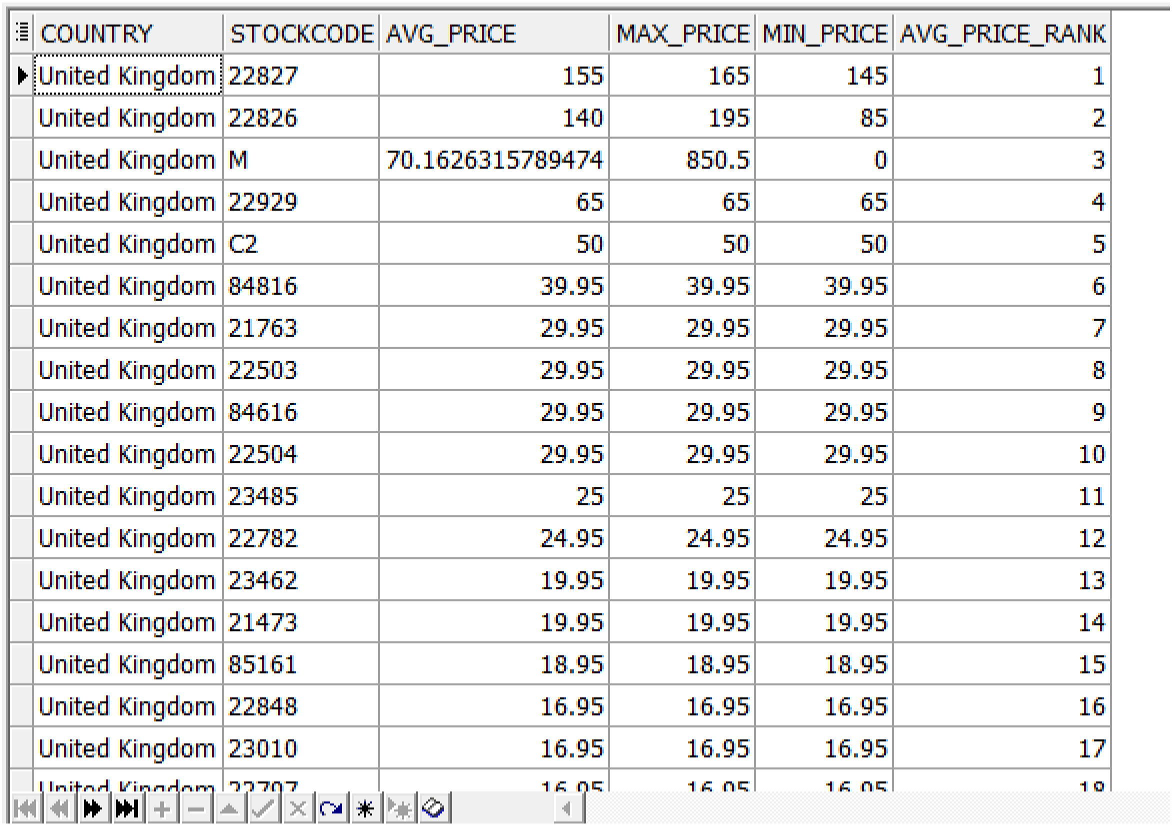
INVOICE	CUSTOMER_ID	MAX_SINGLE_TRANSACTION_VALUE
562439	12931	4176
567309	12901	1519.8
576389	12748	850.5
540507	12939	650.88
561672	12830	587.52
568214	12823	535.5
557092	12908	432
567882	12906	360
571685	12863	340
562270	12917	297
538420	12875	293.76
554084	12909	290
579837	12882	283.2
569397	12747	243
546462	12845	208.8
551595	12891	204
537433	12913	204

3) Identifying price variance across countries:

This query examines the average, maximum, and minimum prices for each product (StockCode) in different countries and ranks each stock code based on its average price within each country. It helps businesses understand their pricing strategies, segment markets based on price sensitivity, compare pricing competitiveness with competitors, and track product performance over time. Ultimately, it enables businesses to make informed decisions about pricing and product positioning to maximize profitability and market share.

- **Query**

```
-- This cte calculates the average, maximum, and minimum prices for each stock code in each
country
with cte as (
  select
    country,
    stockcode,
    avg(price) as avg_price,
    max(price) as max_price,
          min(price) as min_price
  from tableretail
  group by country, stockcode
)
-- Main query to select country, stock code, average price, maximum price, minimum price, and rank stock codes based on average price within each country
SELECT
  country,
  stockcode,
  avg_price,
  max_price,
  min_price,
  row_number() over (partition by country order by avg_price desc) as avg_price_rank
from cte;
```



COUNTRY	STOCKCODE	AVG_PRICE	MAX_PRICE	MIN_PRICE	AVG_PRICE_RANK
United Kingdom	22827	155	165	145	1
United Kingdom	22826	140	195	85	2
United Kingdom	M	70.1626315789474	850.5	0	3
United Kingdom	22929	65	65	65	4
United Kingdom	C2	50	50	50	5
United Kingdom	84816	39.95	39.95	39.95	6
United Kingdom	21763	29.95	29.95	29.95	7
United Kingdom	22503	29.95	29.95	29.95	8
United Kingdom	84616	29.95	29.95	29.95	9
United Kingdom	22504	29.95	29.95	29.95	10
United Kingdom	23485	25	25	25	11
United Kingdom	22782	24.95	24.95	24.95	12
United Kingdom	23462	19.95	19.95	19.95	13
United Kingdom	21473	19.95	19.95	19.95	14
United Kingdom	85161	18.95	18.95	18.95	15
United Kingdom	22848	16.95	16.95	16.95	16
United Kingdom	23010	16.95	16.95	16.95	17
United Kingdom	22797	16.95	16.95	16.95	18

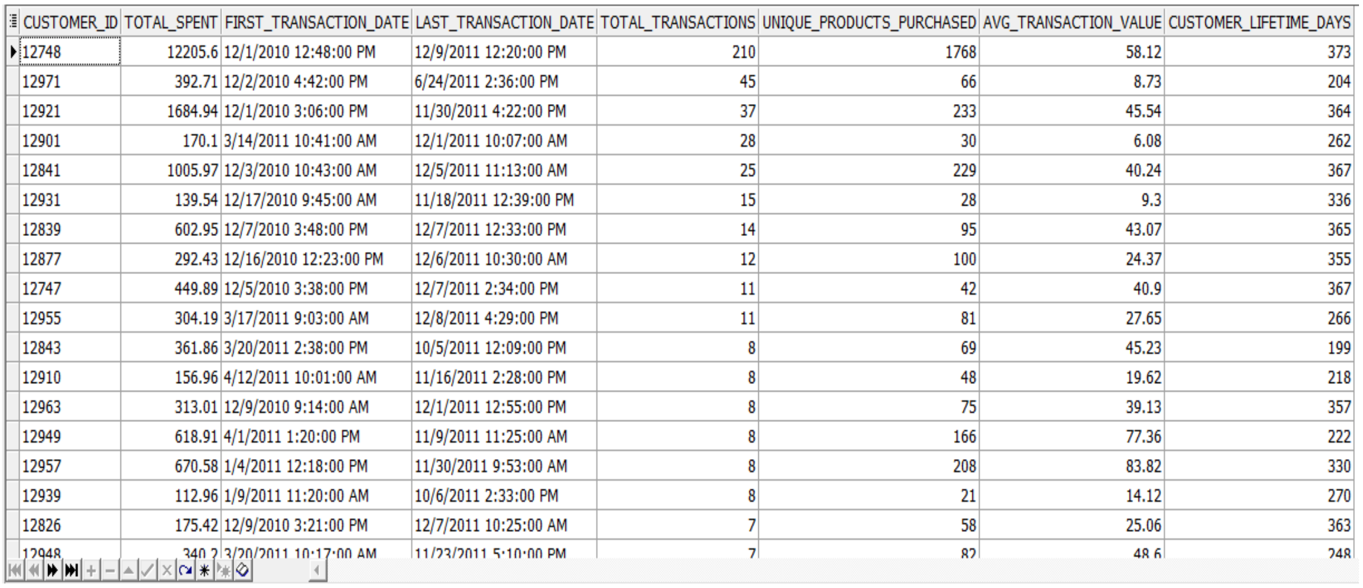
4) Customer Lifetime Cycle:

Customer lifetime insight is a valuable metric for businesses to understand the overall value a customer brings over their entire relationship with the company.

This helps businesses in optimizing their strategies for acquiring, retaining, and maximizing the value of their customer base, ultimately leading to sustainable growth and profitability.

- **Query:**

```
-- CTE to calculate customer transactions and activity
with customer_activity as (
   -- calculate total spent, first purchase date, and last purchase date,total_transactions and unique products for each customer
   select
      customer_id,
      sum(price) as total_spent, -- cumulative amount of money spent by each customer
      min(to_date(invoicedate, 'MM/DD/YYYY HH24:MI')) as first_transaction_date,
      max(to_date(invoicedate, 'MM/DD/YYYY HH24:MI')) as last_transaction_date,
      count(distinct invoice) as total_transactions,
      count(distinct stockcode) as unique_products_purchased
   from tableretail
   group by  customer_id
)
-- Main query
select
   customer_id,
   total_spent,
   first_transaction_date,
   last_transaction_date,
   total_transactions,
   unique_products_purchased,
   round(total_spent / total_transactions, 2) as avg_transaction_value,
   --Duration of the customer's lifetime in days (last purchase date - first purchase date)
  round( (last_transaction_date - first_transaction_date)) as customer_lifetime_days
from customer_activity
-- Order the results by total transactions for each customer desc which means the first row is the most active customer
order by total_transactions desc;
```



CUSTOMER_ID	TOTAL_SPENT	FIRST_TRANSACTION_DATE	LAST_TRANSACTION_DATE	TOTAL_TRANSACTIONS	UNIQUE_PRODUCTS_PURCHASED	AVG_TRANSACTION_VALUE	CUSTOMER_LIFETIME_DAYS
12748	12205.6	12/1/2010 12:48:00 PM	12/9/2011 12:20:00 PM	210	1768	58.12	373
12971	392.71	12/2/2010 4:42:00 PM	6/24/2011 2:36:00 PM	45	66	8.73	204
12921	1684.94	12/1/2010 3:06:00 PM	11/30/2011 4:22:00 PM	37	233	45.54	364
12901	170.1	3/14/2011 10:41:00 AM	12/1/2011 10:07:00 AM	28	30	6.08	262
12841	1005.97	12/3/2010 10:43:00 AM	12/5/2011 11:13:00 AM	25	229	40.24	367
12931	139.54	12/17/2010 9:45:00 AM	11/18/2011 12:39:00 PM	15	28	9.3	336
12839	602.95	12/7/2010 3:48:00 PM	12/7/2011 12:33:00 PM	14	95	43.07	365
12877	292.43	12/16/2010 12:23:00 PM	12/6/2011 10:30:00 AM	12	100	24.37	355
12747	449.89	12/5/2010 3:38:00 PM	12/7/2011 2:34:00 PM	11	42	40.9	367
12955	304.19	3/17/2011 9:03:00 AM	12/8/2011 4:29:00 PM	11	81	27.65	266
12843	361.86	3/20/2011 2:38:00 PM	10/5/2011 12:09:00 PM	8	69	45.23	199
12910	156.96	4/12/2011 10:01:00 AM	11/16/2011 2:28:00 PM	8	48	19.62	218
12963	313.01	12/9/2010 9:14:00 AM	12/1/2011 12:55:00 PM	8	75	39.13	357
12949	618.91	4/1/2011 1:20:00 PM	11/9/2011 11:25:00 AM	8	166	77.36	222
12957	670.58	1/4/2011 12:18:00 PM	11/30/2011 9:53:00 AM	8	208	83.82	330
12939	112.96	1/9/2011 11:20:00 AM	10/6/2011 2:33:00 PM	8	21	14.12	270
12826	175.42	12/9/2010 3:21:00 PM	12/7/2011 10:25:00 AM	7	58	25.06	363
12948	340.2	3/20/2011 10:17:00 AM	11/23/2011 5:10:00 PM	7	82	48.6	248

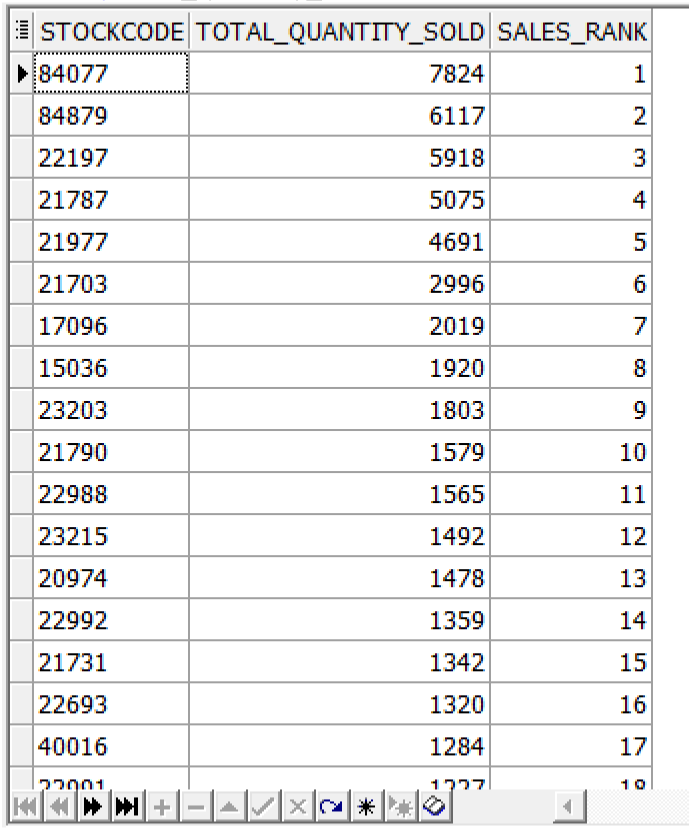
5) Best-Selling Products:

Identifying the best-selling products helps businesses understand customer preferences and demand patterns. By ranking products based on total quantity sold, businesses can prioritize their inventory, focus on popular items, and allocate resources effectively. This insight enables businesses to optimize their product offerings, streamline inventory management, and capitalize on high-demand products to maximize sales and profitability.

- **Query:**

```
-- CTE to calculate total quantity sold for each product
```

```
with product_sales as (
    select
        stockcode,
        sum(quantity) as total_quantity_sold
    from tableretail
    group by stockcode
)
-- main query to rank products based on total quantity sold
select
    stockcode,
    total_quantity_sold,
    rank() over (order by total_quantity_sold desc) as sales_rank
from product_sales
order by total_quantity_sold desc;
```



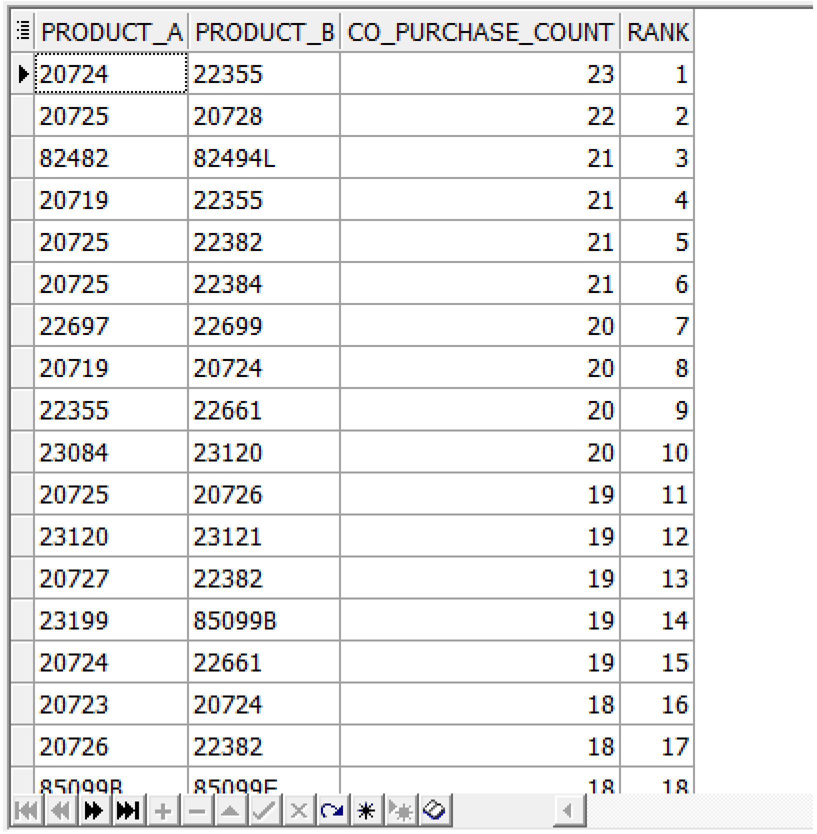
STOCKCODE	TOTAL_QUANTITY_SOLD	SALES_RANK
84077	7824	1
84879	6117	2
22197	5918	3
21787	5075	4
21977	4691	5
21703	2996	6
17096	2019	7
15036	1920	8
23203	1803	9
21790	1579	10
22988	1565	11
23215	1492	12
20974	1478	13
22992	1359	14
21731	1342	15
22693	1320	16
40016	1284	17

6) Cross-Selling Analysis:

This insight aims to identify frequently co-purchased products to optimize cross-selling strategies and improve revenue generation. Understanding which products are frequently purchased together allows businesses to implement effective cross-selling strategies. By recommending complementary products to customers during the purchase process, businesses can increase the average order value and enhance customer satisfaction.

- **Query:**

```
-- CTE to generate pairs of co-purchased products
with cte as (
   -- selects pairs of products that are purchased together, avoiding duplicate pairs
   select
      a.stockcode as product_A,  -- The first product in the pair
      b.stockcode AS product_B,  -- The second product in the pair
      count(*) as co_purchase_count  -- Counts the number of times the pair purchased  together
   from  tableretail a
   join tableretail b
on a.invoice = b.invoice and a.stockcode < b.stockcode
      /* Joins the table with itself to find co-purchased items, ensuring product_1 < product_2
      using a.stockcode < b.stockcode ensures that each pair of products is only counted once.
      This condition prevents duplicate pairs where the same two products are swapped in
positions
       (a.StockCode and b.StockCode are interchanged) */
   group by a.stockcode, b.stockcode  -- group the results by the two products in each pair
)
select
   product_A,
   product_B,
   co_purchase_count,
   row_number() over (order by co_purchase_count desc) as rank
      -- assigns a rank to each pair based on their frequency
from cte
order by co_purchase_count desc;  -- orders the results by the frequency of co-purchase counts
```



PRODUCT_A	PRODUCT_B	CO_PURCHASE_COUNT	RANK
20724	22355	23	1
20725	20728	22	2
82482	82494L	21	3
20719	22355	21	4
20725	22382	21	5
20725	22384	21	6
22697	22699	20	7
20719	20724	20	8
22355	22661	20	9
23084	23120	20	10
20725	20726	19	11
23120	23121	19	12
20727	22382	19	13
23199	85099B	19	14
20724	22661	19	15
20723	20724	18	16
20726	22382	18	17
85099B	85099F	18	18

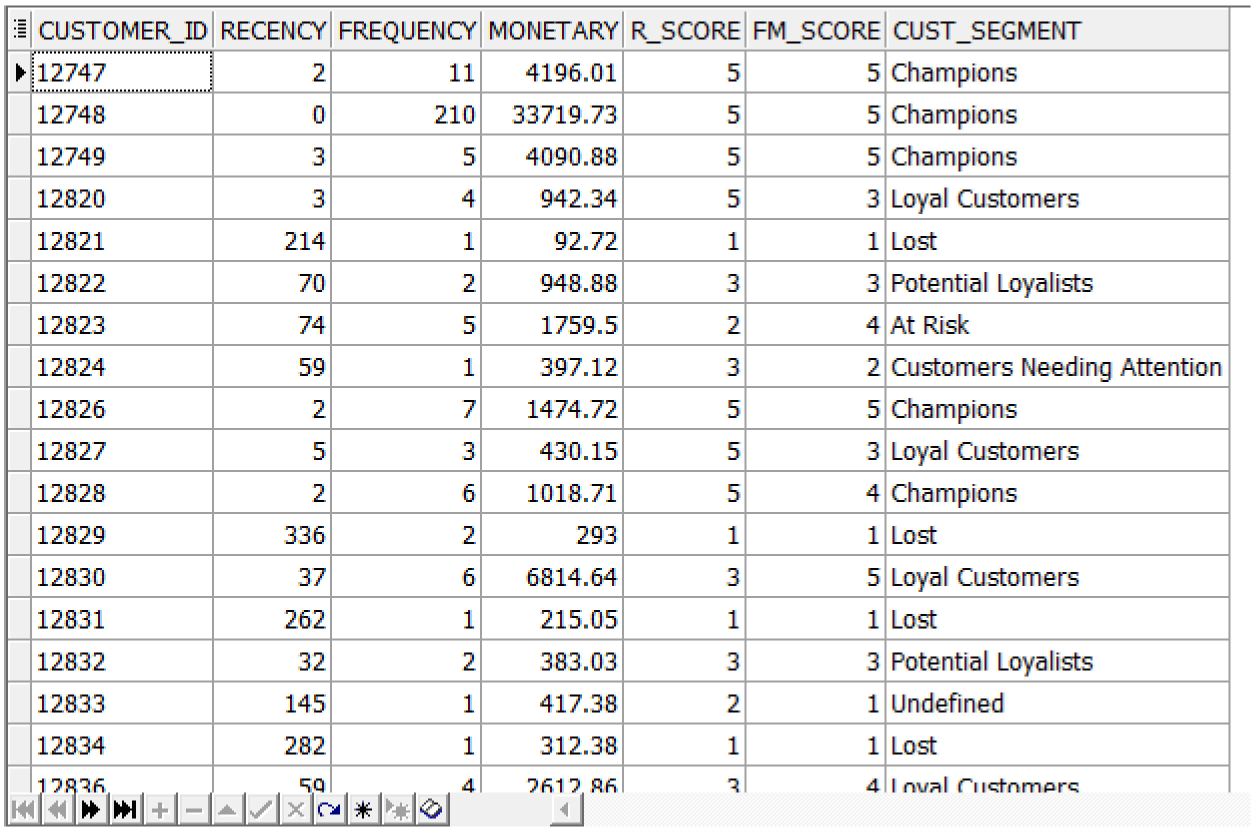
Q2: A Monetary model for customers behavior:

A monetary model in customer analytics is used to segment customers based on their monetary value or spending behavior. It focuses on how much money each customer has spent over a certain period and helps identify high-value customers, low-value customers, and everything in between.

- **Query:**

```
-- cte to calculate RFM (Recency, Frequency, Monetary) metrics for each customer
with RFM as (
  select
    customer_id,
                 -- calculate recency as the difference between the maximum invoice date and the
maximum invoice date for each individual customer(last transaction).
     round((select max(to_date(invoicedate, 'mm/dd/yyyy hh24:mi')) from tableretail) -
max(to_date(invoicedate, 'mm/dd/yyyy hh24:mi'))) as recency,
     -- calculate frequency as the count of distinct invoices for each customer
     count(distinct invoice)  as frequency,
     -- calculate monetary as the sum of quantity multiplied by price for each customer
     sum(quantity * price) as monetary
   from tableretail
   group by customer_id
),
--CTE to calculate scores
rfm_scores as (
select
    customer_id,
    recency,
    frequency,
    monetary,
  -- assign r_score based on the recency, dividing customers into 5 groups
     ntile(5) over (order by recency desc) as r_score,
     ntile(5) over (order by Frequency ) as f_score,
     ntile(5) over (order by Monetary ) as m_score
     from rfm
),
-- CTE to calculate the FM_score, which is the average of f_score and m_score
FM as (
   select
     customer_id,
     ntile(5) over (order by (f_score + m_score) / 2) as FM_score
   from rfm_scores
)
-- Main query to select RFM metrics, scores, and customer segments
select
  fm.customer_id,
  recency,
  frequency,
 monetary,
  r_score,
  FM.FM_score,
   -- assign customer segments based on r_score and FM_score combinations
   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) then 'Hibernating'
      when (r_score = 1 and FM_score = 1) then 'Lost'
      else 'Undefined'
   end as Cust_Segment
from rfm_scores
join FM
on    rfm_scores.customer_id= FM.customer_id
order by customer_id;
```



CUSTOMER_ID	RECENCY	FREQUENCY	MONETARY	R_SCORE	FM_SCORE	CUST_SEGMENT
12747	2	11	4196.01	5	5	Champions
12748	0	210	33719.73	5	5	Champions
12749	3	5	4090.88	5	5	Champions
12820	3	4	942.34	5	3	Loyal Customers
12821	214	1	92.72	1	1	Lost
12822	70	2	948.88	3	3	Potential Loyalists
12823	74	5	1759.5	2	4	At Risk
12824	59	1	397.12	3	2	Customers Needing Attention
12826	2	7	1474.72	5	5	Champions
12827	5	3	430.15	5	3	Loyal Customers
12828	2	6	1018.71	5	4	Champions
12829	336	2	293	1	1	Lost
12830	37	6	6814.64	3	5	Loyal Customers
12831	262	1	215.05	1	1	Lost
12832	32	2	383.03	3	3	Potential Loyalists
12833	145	1	417.38	2	1	Undefined
12834	282	1	312.38	1	1	Lost
12836	59	4	2612.86	3	4	Loyal Customers

Q3:

A) The maximum number of consecutive days a customer made purchases:

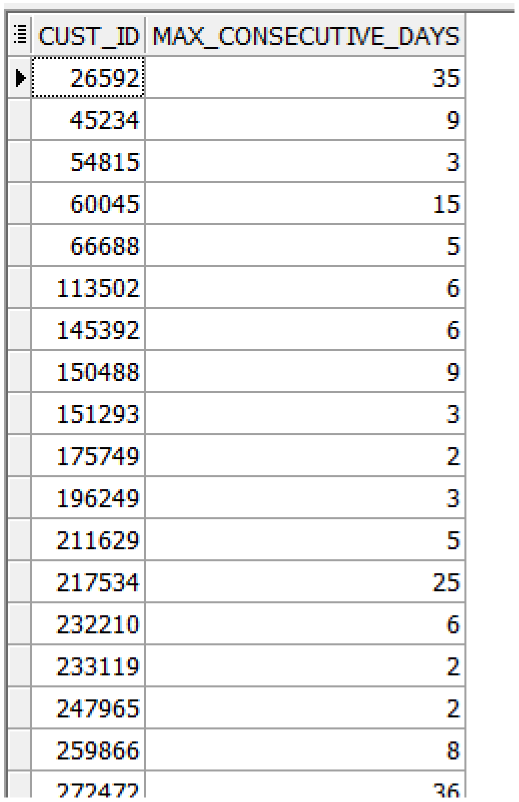
This SQL query aims to find the maximum number of consecutive days on which each customer made purchases. It starts by ranking customer purchases by calendar date using the CTE. Then, it calculates the consecutive days of purchases for each customer in the Consecutive_Days CTE and put every consecutive day at the same group and then group them using cust_id,group_id. Finally, it selects the maximum consecutive days for each customer and orders the results by customer id.

- **Query:**

```
with CTE as (
  --  a row number to each purchase for every customer, ordered by purchase date
  select
    cust_id,
    calendar_dt ,
    row_number() over (partition by  cust_id order by calendar_dt) as rankk
```

```
        from customers
),
consecutive_days as (
   -- Calculate the difference between the purchase date and its row number to identify
consecutive purchases(the consective purchases will have the same group id )
   select
      cust_id,
      calendar_dt,
      rankk,
      calendar_dt - rankk AS group_id
   from CTE
),
grouped_consecutive_days as (
   -- group consecutive purchases by customer and group id
   select
      cust_id,
      group_id,
      min(calendar_dt) as start_date,
      max(calendar_dt) as end_date,
      count(*) as consecutive_days
   from consecutive_days
   group by cust_id, group_id
)
-- Select the maximum consecutive days for each customer
select
   cust_id,
   max(consecutive_days) as max_consecutive_days
from grouped_consecutive_days
group by cust_id
order by cust_id ;
```

Information
Technology
Institute



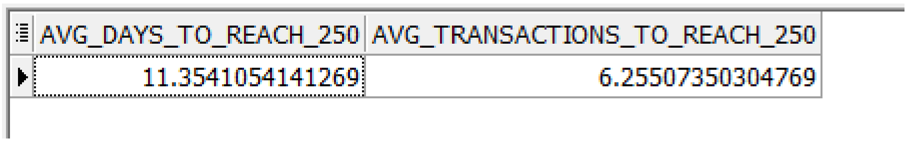
CUST_ID	MAX_CONSECUTIVE_DAYS
26592	35
45234	9
54815	3
60045	15
66688	5
113502	6
145392	6
150488	9
151293	3
175749	2
196249	3
211629	5
217534	25
232210	6
233119	2
247965	2
259866	8
272472	36

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

- **Query:**

```
-- CTE to calculate cumulative spending for each customer over time
with cumulative_spending as (
  select
     cust_id,
     calendar_dt,
     amt_LE,
     -- Calculate the total spending for each customer at each transaction
     sum(amt_LE) over (partition by  cust_id order by calendar_dt) as total_spending,
     -- Identify the first transaction date for each customer
     first_value(calendar_dt) over (partition by  cust_id order by calendar_dt) as 
first_transaction_date,
     -- Calculate the cumulative count of transactions for each customer
    count(*) over(partition by cust_id order by calendar_dt) as total_transactions
  from customers
),
-- CTE to calculate the number of days it took for each customer to reach a total spending of 250
to_reach as (
        select
  distinct cust_id ,
  min(total_transactions) over (partition by cust_id) as  transactions_to_reach_250,
  min(calendar_dt - first_transaction_date) over (partition by cust_id) as days_to_reach_250
 from cumulative_spending
 where total_spending >= 250
 order by cust_id asc

)
-- Main query to calculate the avg number of days and transactions it took each customer to reach 250 in total spending
select
 avg(days_to_reach_250) as avg_days_to_reach_250,
 avg(transactions_to_reach_250) as avg_transactions_to_reach_250
from to_reach;
```



AVG_DAYS_TO_REACH_250	AVG_TRANSACTIONS_TO_REACH_250
11.3541054141269	6.25507350304769

- Cumulative Spending CTE (cumulative_spending):

It computes the cumulative spending for each customer over time.
The SUM window function calculates the total spending for each customer at each transaction, partitioned by cust_id and ordered by calendar_dt.
The FIRST_VALUE function identifies the first transaction date for each customer.
The count window function computes the cumulative count of transactions for each customer, also partitioned by cust_id and ordered by calendar_dt.

- To Reach CTE (to_reach):

It calculates the number of days and transactions it took for each customer to reach a total spending of 250.
The DISTINCT keyword ensures that only unique cust_id values are considered.
The MIN window function is used to find the minimum value of total_transactions and calendar_dt - first_transaction_date for each cust_id, effectively identifying the number of transactions and days it took to reach a spending of 250.
The WHERE clause filters out records where the total spending is less than 250.

- Main Query:

It computes the average number of days (avg_days_to_reach_250) and transactions (avg_transactions_to_reach_250) for each customer to reach a spending of 250.