Information Technology Institute
Analytical SQL project
Track Data Management
Mariam Marcos

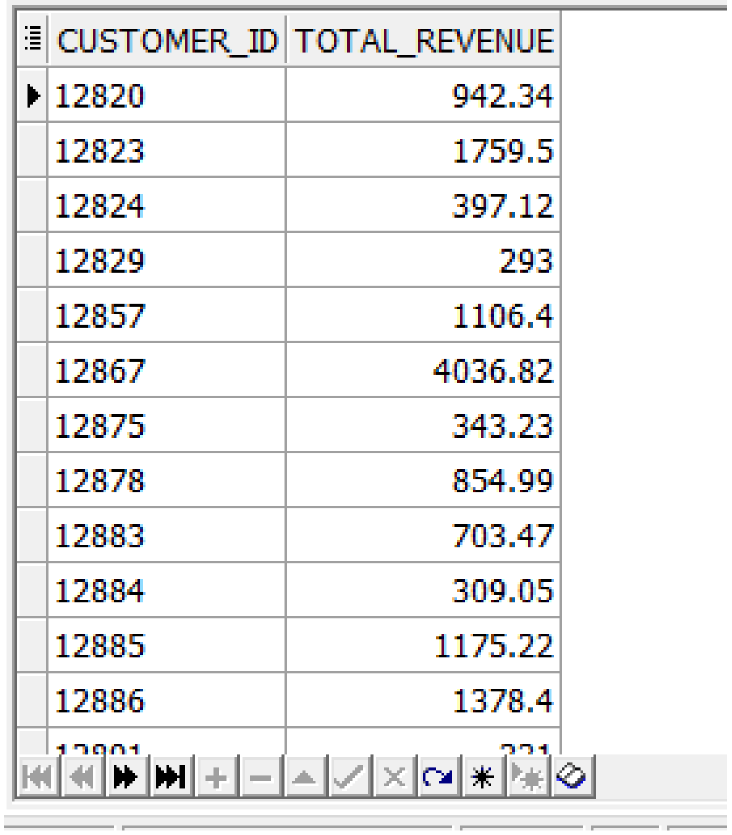
A Journey Through Business Queries

1-First Query:

1. Total Revenue by Customer:

In their quest to understand the financial landscape of their business, our entrepreneur utilized a query to calculate the total revenue generated by each customer. By multiplying the quantity of each purchased item by its price and summing up those values, they gained valuable insights into the contribution of individual customers to overall revenue.

```
    select distinct customer_id,sum(quantity*price) over(partition by customer_id) as total_revenue
from tableretail;
```

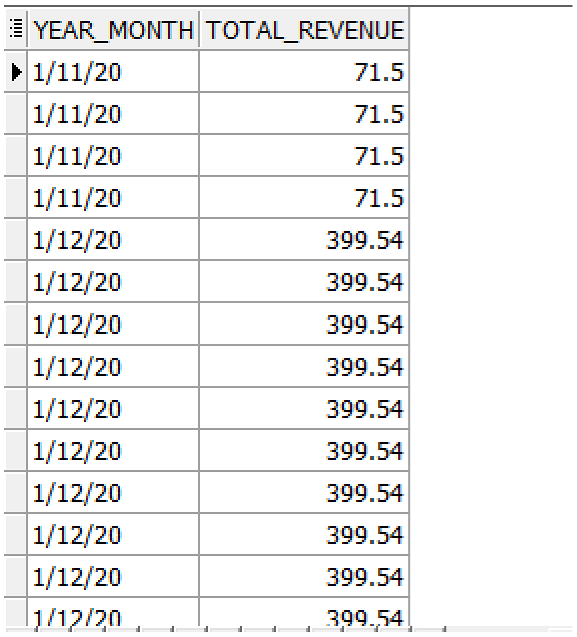


CUSTOMER_ID	TOTAL_REVENUE
12820	942.34
12823	1759.5
12824	397.12
12829	293
12857	1106.4
12867	4036.82
12875	343.23
12878	854.99
12883	703.47
12884	309.05
12885	1175.22
12886	1378.4

2. Monthly Sales Trend:

Seeking to analyze revenue trends over time, our protagonist employed a query to break down sales data into monthly increments. This query provided a comprehensive overview of sales performance, allowing them to identify patterns and fluctuations in revenue on a month-by-month basis.

```
     select substr(invoicedate,1,7)as year_month,
sum(quantity*price)over (partition by substr(invoicedate,1,7)) as total_revenue from tableretail;
```

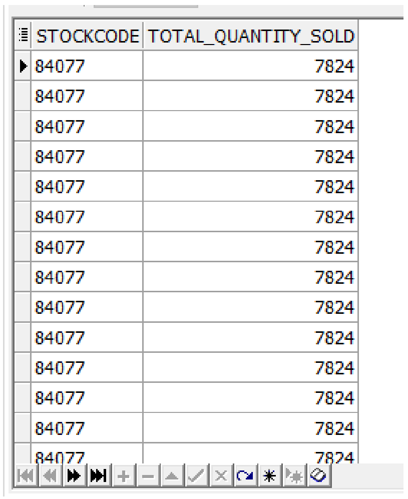


YEAR_MONTH	TOTAL_REVENUE
1/11/20	71.5
1/11/20	71.5
1/11/20	71.5
1/11/20	71.5
1/12/20	399.54
1/12/20	399.54
1/12/20	399.54
1/12/20	399.54
1/12/20	399.54
1/12/20	399.54
1/12/20	399.54
1/12/20	399.54
1/12/20	399.54
1/12/20	399.54

3. Top Selling Products:

To identify the best-performing products in their inventory, our entrepreneur crafted a query to aggregate the total quantity sold for each stock code. By sorting the results in descending order, they uncovered the top-selling products that drove profitability and customer satisfaction.

```
        select stockcode,sum(quantity) over(partition by stockcode) as total_quantity_sold from
tableretail order by total_quantity_sold desc;
```

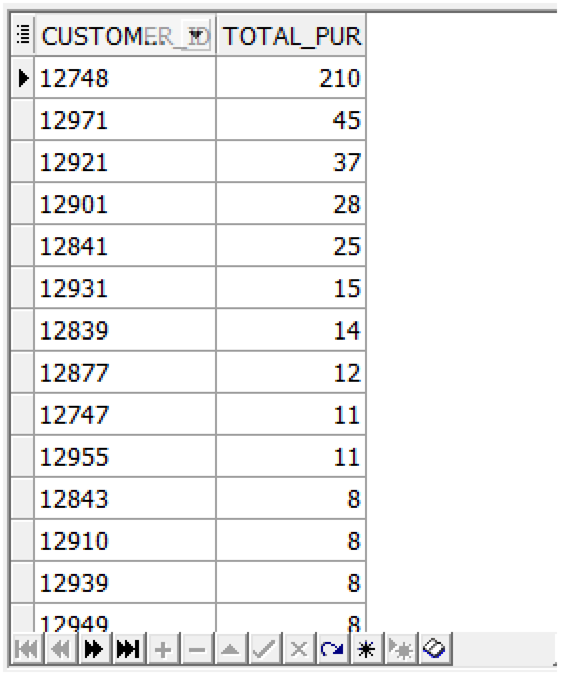


STOCKCODE	TOTAL_QUANTITY_SOLD
84077	7824
84077	7824
84077	7824
84077	7824
84077	7824
84077	7824
84077	7824
84077	7824
84077	7824
84077	7824
84077	7824
84077	7824
84077	7824
84077	7824

4. Customer Purchase Frequency:

Understanding the purchasing habits of their clientele was paramount for our entrepreneur. By calculating the total number of unique invoices per customer, they gained insights into customer loyalty and engagement, enabling targeted marketing strategies and personalized offerings.

```
        select distinct
    customer_id ,
    count(distinct invoice) over (partition by customer_id)as total_pur
    from tableretail
    order by total_pur desc;
```

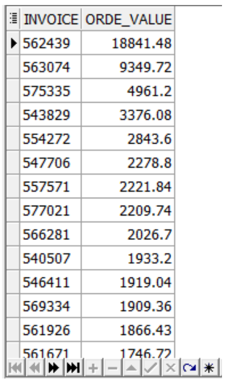


CUSTOMER_ID	TOTAL_PUR
12748	210
12971	45
12921	37
12901	28
12841	25
12931	15
12839	14
12877	12
12747	11
12955	11
12843	8
12910	8
12939	8
12949	8

5. Average Order Value:

In their pursuit of optimizing sales strategies, our protagonist utilized a query to calculate the average order value. By summing the total value of each invoice (order), they gained a deeper understanding of customer spending patterns and transactional behaviors.

```
  select distinct invoice,sum(quantity*price) over (partition by invoice) as orde_value
    from tableretail
    order by orde_value desc;
```

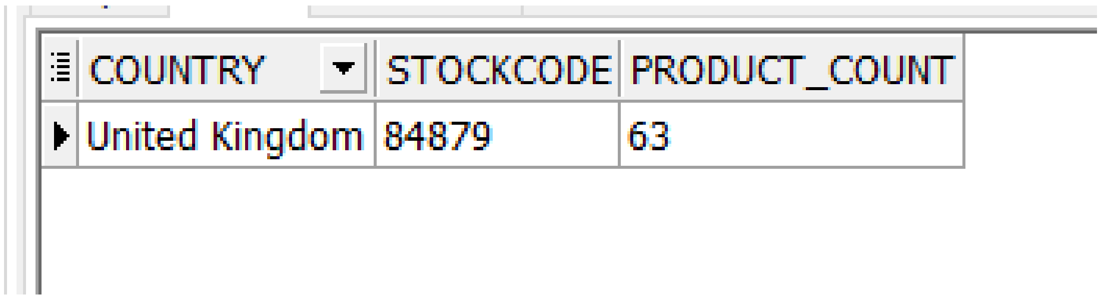


INVOICE	ORDE_VALUE
562439	18841.48
563074	9349.72
575335	4961.2
543829	3376.08
554272	2843.6
547706	2278.8
557571	2221.84
577021	2209.74
566281	2026.7
540507	1933.2
546411	1919.04
569334	1909.36
561926	1866.43
561671	1746.72

6. The Most Frequent Product Sold in Each Country:

Venturing into the realm of international sales, our entrepreneur sought to identify the most popular products in each country. Through a meticulously crafted query, they analyzed sales data by country and stock code, shedding light on cross-border preferences and market trends.

```
with CountryProductCounts as (
    select
        country,
        stockcode,
        count(*) as product_count,
        row_number() over(partition by country order by count(*) desc) as rn
    from
        tableretail
    group by
        country, stockcode
)
select
    country,
    stockcode,
    product_count
from
    CountryProductCounts
where
    rn = 1;
```

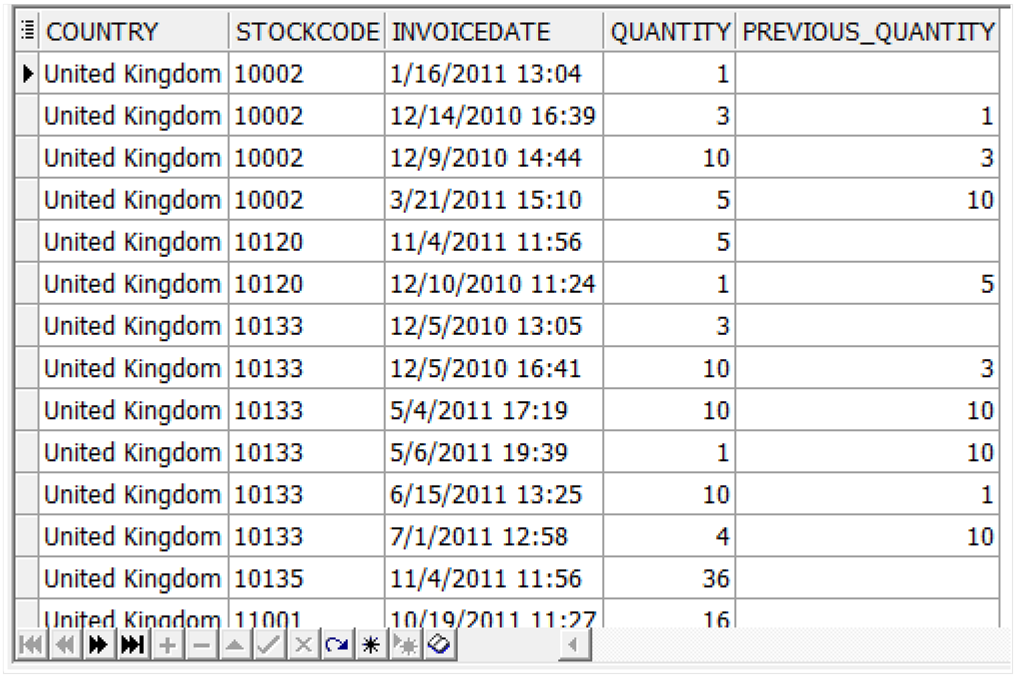


COUNTRY	STOCKCODE	PRODUCT_COUNT
United Kingdom	84879	63

7. Analyzing Sales Trends Using Lag:

Employing a sophisticated query with lag functions, our protagonist delved into the nuances of sales dynamics over time. By examining changes in quantity sold relative to previous transactions, they discerned patterns and fluctuations in demand across different product categories and regions.

```
select
    country,
    stockcode,
    invoicedate,
    quantity,
    lag(quantity) over (partition by stockcode order by invoicedate) as previous_quantity
from
    tableretail
order by
    country, stockcode, invoicedate;
```



COUNTRY	STOCKCODE	INVOICEDATE	QUANTITY	PREVIOUS_QUANTITY
United Kingdom	10002	1/16/2011 13:04	1	
United Kingdom	10002	12/14/2010 16:39	3	1
United Kingdom	10002	12/9/2010 14:44	10	3
United Kingdom	10002	3/21/2011 15:10	5	10
United Kingdom	10120	11/4/2011 11:56	5	
United Kingdom	10120	12/10/2010 11:24	1	5
United Kingdom	10133	12/5/2010 13:05	3	
United Kingdom	10133	12/5/2010 16:41	10	3
United Kingdom	10133	5/4/2011 17:19	10	10
United Kingdom	10133	5/6/2011 19:39	1	10
United Kingdom	10133	6/15/2011 13:25	10	1
United Kingdom	10133	7/1/2011 12:58	4	10
United Kingdom	10135	11/4/2011 11:56	36	
United Kingdom	11001	10/19/2011 11:27	16	

Customer Segmentation:

Objective:

The objective of this document is to outline the implementation of a Monetary model for customer behavior in product purchasing and segmenting each customer based on specific groups.

Overview:

Customer segmentation is a crucial aspect of marketing strategy, allowing businesses to tailor their approaches to different customer groups. In this implementation, we utilize the RFM (Recency, Frequency, Monetary) model to segment customers based on their recent transaction history, purchase frequency, and monetary value.

IMPLEMENTATION:

1. **DATA PREPARATION:**
   - THE FIRST STEP INVOLVES PREPARING THE DATA, ENSURING IT IS CLEAN, STRUCTURED, AND RELEVANT TO THE ANALYSIS. THE DATASET USED CONTAINS INFORMATION ON CUSTOMER TRANSACTIONS, INCLUDING CUSTOMER ID, PURCHASE DATES, QUANTITIES, AND PRICES.
2. **RFM CALCULATION:**
   - RECENCY (R): CALCULATE HOW RECENT THE LAST TRANSACTION IS FOR EACH CUSTOMER. THIS IS DETERMINED BY FINDING THE DIFFERENCE BETWEEN THE REFERENCE DATE (MOST RECENT PURCHASE IN THE DATASET) AND THE DATE OF THE LAST PURCHASE.
   - FREQUENCY (F): COUNT THE NUMBER OF TIMES EACH CUSTOMER HAS MADE A PURCHASE FROM THE STORE.
   - MONETARY (M): CALCULATE THE TOTAL AMOUNT EACH CUSTOMER HAS SPENT ON PRODUCTS PURCHASED FROM THE STORE.
3. **RFM SCORE CALCULATION:**
   - AFTER CALCULATING THE RFM VALUES FOR EACH CUSTOMER, ASSIGN SCORES TO THEM BASED ON THEIR QUARTILE RANKINGS.
   - FOR RECENCY AND MONETARY, HIGHER SCORES INDICATE BETTER PERFORMANCE, WHILE FOR FREQUENCY, LOWER SCORES INDICATE HIGHER FREQUENCY.
4. **AVERAGE RFM SCORE CALCULATION:**
   - TO REDUCE PERMUTATIONS AND SIMPLIFY SEGMENTATION, CALCULATE THE AVERAGE SCORES FOR FREQUENCY AND MONETARY. THIS PROVIDES A MORE MANAGEABLE SET OF SCORES FOR SEGMENTATION.
5. **CUSTOMER SEGMENTATION:**
   - BASED ON THE RFM SCORES AND AVERAGE SCORES, SEGMENT CUSTOMERS INTO DISTINCT GROUPS:
     - CHAMPIONS: CUSTOMERS WITH HIGH RECENCY AND HIGH AVERAGE SCORES (FM).
     - Loyal Customers: Customers with high recency and frequency.

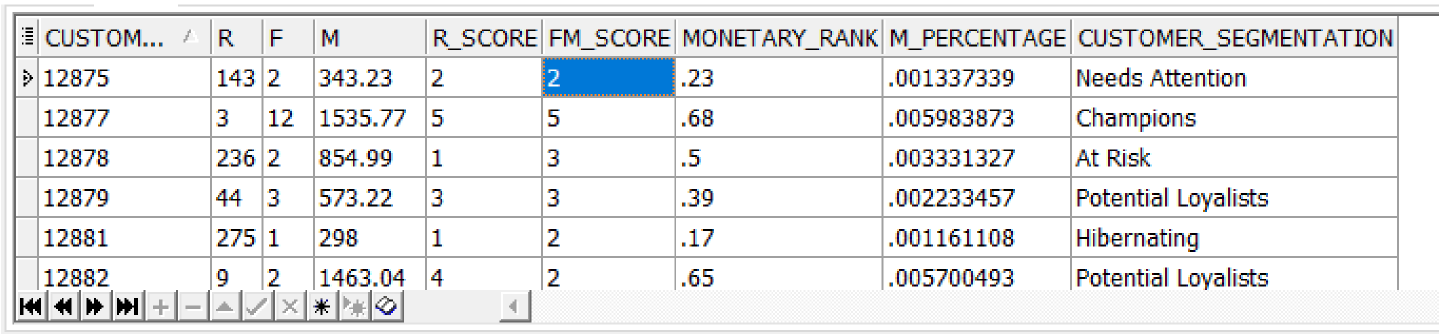
- Potential Loyalists: Customers with high recency and moderate average scores.
- Recent Customers: Customers with recent purchases and moderate average scores.
- Lost: Customers with low recency and low average scores.

6. **Implementation Query:**
   - Utilize SQL queries to implement the RFM model and segment customers based on the specified criteria.

Expected Outcome: The expected outcome of this implementation is a segmented customer dataset containing customer IDs, RFM values, RFM scores, average scores, and customer segments. This dataset will provide valuable insights into customer behavior and enable targeted marketing strategies tailored to different customer groups.

```
WITH customer_purchase_data AS (
 SELECT
 customer_id,
 Quantity,
 Price,
 Invoice,
 ROUND(MAX(TO_DATE(InvoiceDate, 'MM/DD/YYYY HH24:MI')) OVER () -
MAX(TO_DATE(InvoiceDate,'MM/DD/YYYY HH24:MI')) OVER (PARTITION BY customer_id)) AS
recency
 FROM tableRetail
),
customer_frequency_monetary AS (
 SELECT
 customer_id,
 COUNT(DISTINCT invoice) AS frequency,
 SUM(price * quantity) AS monetary,
 recency
 FROM customer_purchase_data
 GROUP BY customer_id, recency
),
customer_score AS (
 SELECT
 customer_id,
 recency,
 frequency,
 NTILE(5) OVER (ORDER BY frequency ) AS F_score,
 monetary,
 ROUND(PERCENT_RANK() OVER (ORDER BY monetary ), 2) AS monetary_rank
 FROM customer_frequency_monetary
),
customer_rfm_score AS (
 SELECT
 customer_id,
 recency,
 frequency,
 monetary,
 monetary_rank,
 NTILE(5) OVER (ORDER BY recency DESC) AS R_score,
 NTILE(5) OVER (ORDER BY (F_score + monetary_rank) / 2) AS FM_score
 FROM customer_score
)
SELECT
 customer_id,
 recency AS R,
 frequency AS F,
 monetary AS M,
 R_score,
 FM_score,
 monetary_rank ,
 monetary / SUM(monetary) OVER () AS M_percentage,
 CASE
   WHEN R_score = 5 AND FM_score IN (5, 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 IN (2 , 3) THEN 'Potential Loyalists'
   WHEN R_score = 3 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 IN (5, 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 = 2 AND FM_score IN (3, 2) THEN 'Needs Attention'
   WHEN R_score = 3 AND FM_score = 2 THEN 'Needs Attention'
   WHEN R_score = 2 AND FM_score IN (5, 4) THEN 'At Risk'
   WHEN R_score = 1 AND FM_score = 3 THEN 'At Risk'
   WHEN R_score = 1 AND FM_score IN (5, 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 'Inactive'
END AS Customer_segmentation

FROM customer_rfm_score ;
```



CUSTOM...	R	F	M	R_SCORE	FM_SCORE	MONETARY_RANK	M_PERCENTAGE	CUSTOMER_SEGMENTATION
12875	143	2	343.23	2	2	.23	.001337339	Needs Attention
12877	3	12	1535.77	5	5	.68	.005983873	Champions
12878	236	2	854.99	1	3	.5	.003331327	At Risk
12879	44	3	573.22	3	3	.39	.002233457	Potential Loyalists
12881	275	1	298	1	2	.17	.001161108	Hibernating
12882	9	2	1463.04	4	2	.65	.005700493	Potential Loyalists

Analysis of Customer Purchase Patterns

Objective: The objective of this document is to provide documentation for a SQL query that analyzes customer purchase patterns, specifically focusing on identifying the maximum consecutive days between purchases for each customer.

Overview: Understanding customer purchase behavior is crucial for businesses to tailor their marketing strategies and enhance customer engagement. This SQL query utilizes the concept of consecutive days between purchases to gain insights into customer activity and loyalty.

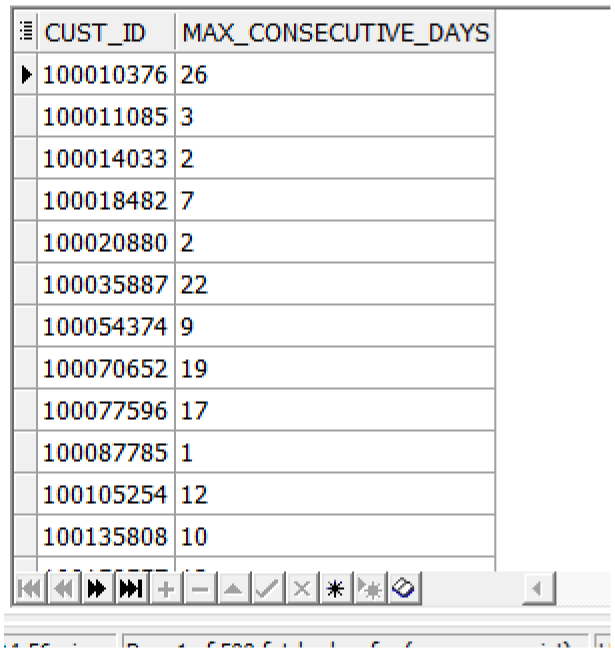
Query Explanation:

1. **Purchase Dates Subquery (purchase_dates):**
   - This subquery selects the customer ID (`cust_id`), purchase date (`calendar_dt`), and assigns a row number (`rn`) to each purchase date within each customer's transaction history.
   - The `row_number()` function partitions the data by customer ID and orders it by calendar date, assigning a sequential number to each row.
2. **Date Differences Subquery (date_diffs):**
   - Building upon the previous subquery, this subquery calculates the difference in days between consecutive purchase dates for each customer.
   - It subtracts the previous purchase date (using the `LAG()` function) from the current purchase date to determine the time lapse between purchases.
   - The resulting difference is stored in the `diff` column.
3. **Main Query:**
   - This query retrieves data from the `date_diffs` subquery.
   - It selects the customer ID (`cust_id`) and calculates the maximum consecutive days between purchases (`max(diff)`).
   - The data is grouped by customer ID (`group by cust_id`), ensuring that the maximum consecutive days are calculated for each individual customer.

Expected Outcome: The expected outcome of this query is a dataset containing each customer's ID along with the maximum consecutive days between their purchases. This information provides valuable insights into customer behavior, indicating how frequently customers make purchases and the gaps between their transactions.

```
with purchase_dates as (
select
cust_id,
calendar_dt,
row_number() over (partition by cust_id order by calendar_dt) as rn
from
my_table
),
date_diffs as (
select
cust_id,
calendar_dt,
calendar_dt - lag(calendar_dt) over (partition by cust_id order by calendar_dt) as diff
from
purchase_dates
)
select
cust_id,
max(diff) as max_consecutive_days
from
date_diffs

group by
cust_id;
```



CUST_ID	MAX_CONSECUTIVE_DAYS
100010376	26
100011085	3
100014033	2
100018482	7
100020880	2
100035887	22
100054374	9
100070652	19
100077596	17
100087785	1
100105254	12
100135808	10

Query Purpose:

The purpose of this SQL query is to calculate the average number of days it takes for customers to reach a spending threshold of 250 units or more from their first purchase date.

Data Sources:

The data used in this query is sourced from a table named `my_table`, which presumably contains information about customer transactions including their identification (`cust_id`), the date of each transaction (`calendar_dt`), and the amount spent (`amt_le`).

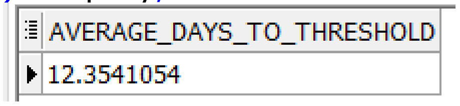
Common Table Expressions (CTEs):

1. `purchase_dates`:
    - Calculates the first purchase date for each customer.
    - Columns:
        - `cust_id`: Customer identification.
        - `first_purchase_date`: The earliest transaction date for each customer.
2. `cumulative_spending`:
    - Calculates the cumulative spending for each customer over time.
    - Columns:
        - `cust_id`: Customer identification.
        - `calendar_dt`: Transaction date.
        - `total_spent`: Cumulative spending for each customer up to the current transaction date.
3. `threshold_dates`:
    - Determines the date when each customer's spending reaches or exceeds 250 units for the first time.
    - Columns:
        - `cust_id`: Customer identification.
        - `threshold_date`: The date when the spending threshold is first met.

Main Query:

The main query computes the average number of days it takes for each customer to reach the spending threshold of 250 units from their first purchase date.

- It calculates the difference in days between the `threshold_date` and `first_purchase_date` for each customer.
- Then, it calculates the average of these differences across all customers.
- If a customer does not reach the spending threshold, their contribution to the average is ignored.

```sql
with purchase_dates as (
    select
        cust_id,
        min(calendar_dt) as first_purchase_date
    from
        my_table
    group by
        cust_id
),
cumulative_spending as (
    select
        cust_id,
        calendar_dt,
        sum(amt_le) over (partition by cust_id order by calendar_dt) as total_spent
    from
        my_table
),
threshold_dates as (
    select
        cust_id,
        min(case when total_spent >= 250 then calendar_dt end) as threshold_date
    from
        cumulative_spending
    group by
        cust_id
)
select
    avg(
        case
            when threshold_date is not null then threshold_date - first_purchase_date + 1
            else null
        end
    ) as average_days_to_threshold
from (
    select
        fp.cust_id,
        min(fp.first_purchase_date) as first_purchase_date,
        tr.threshold_date
    from
        purchase_dates fp
    left join
        threshold_dates tr on fp.cust_id = tr.cust_id
    group by
        fp.cust_id, tr.threshold_date
) subquery;
```


```
AVERAGE_DAYS_TO_THRESHOLD
12.3541054
```