Information
Technology
Institute

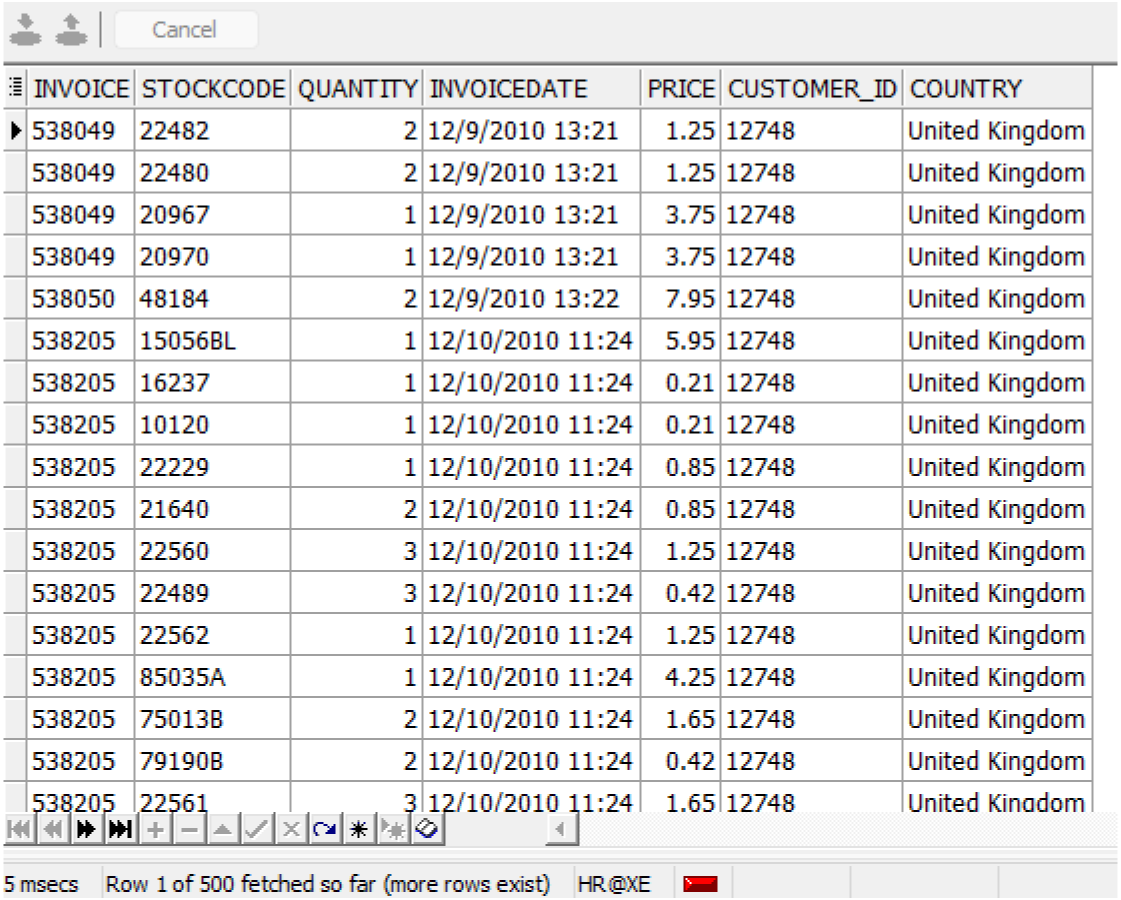
Analytical SQL Project

"Customers Transaction Analysis"

Shorouk Abdelraouf

Background:

Customers have purchasing transactions 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.



INVOICE	STOCKCODE	QUANTITY	INVOICEDATE	PRICE	CUSTOMER_ID	COUNTRY
538049	22482	2	12/9/2010 13:21	1.25	12748	United Kingdom
538049	22480	2	12/9/2010 13:21	1.25	12748	United Kingdom
538049	20967	1	12/9/2010 13:21	3.75	12748	United Kingdom
538049	20970	1	12/9/2010 13:21	3.75	12748	United Kingdom
538050	48184	2	12/9/2010 13:22	7.95	12748	United Kingdom
538205	15056BL	1	12/10/2010 11:24	5.95	12748	United Kingdom
538205	16237	1	12/10/2010 11:24	0.21	12748	United Kingdom
538205	10120	1	12/10/2010 11:24	0.21	12748	United Kingdom
538205	22229	1	12/10/2010 11:24	0.85	12748	United Kingdom
538205	21640	2	12/10/2010 11:24	0.85	12748	United Kingdom
538205	22560	3	12/10/2010 11:24	1.25	12748	United Kingdom
538205	22489	3	12/10/2010 11:24	0.42	12748	United Kingdom
538205	22562	1	12/10/2010 11:24	1.25	12748	United Kingdom
538205	85035A	1	12/10/2010 11:24	4.25	12748	United Kingdom
538205	75013B	2	12/10/2010 11:24	1.65	12748	United Kingdom
538205	79190B	2	12/10/2010 11:24	0.42	12748	United Kingdom
538205	22561	3	12/10/2010 11:24	1.65	12748	United Kingdom

5 msecs | Row 1 of 500 fetched so far (more rows exist) | HR@XE

Problem_1:

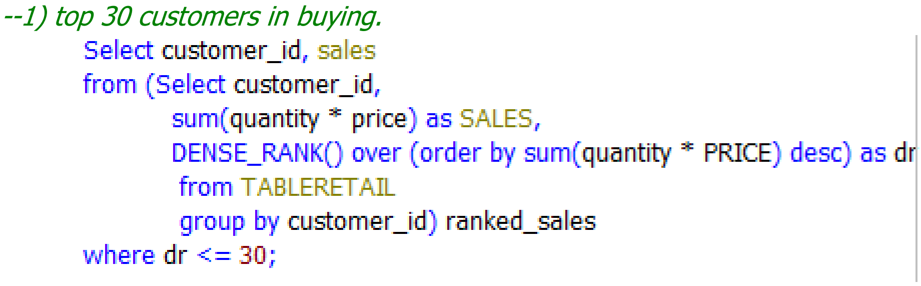
- write at **least 5 analytical SQL** queries that tells a story about the data.
- write small description about the business meaning behind each query.

Query1:

Business Description:

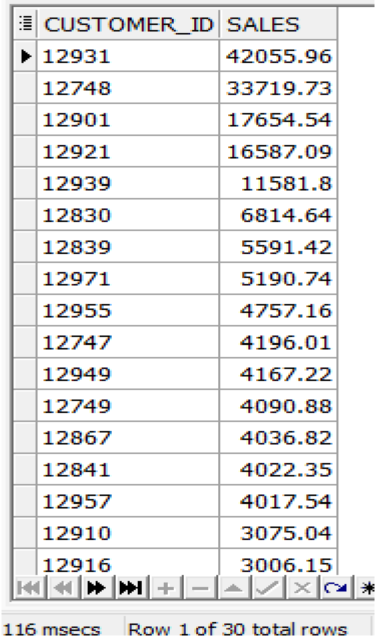
This query identifies the top 30 customers who generate the most sales for the business. Businesses can use this information to target their top customers with specific marketing campaigns and promotions.

Query:



```
--1) top 30 customers in buying.
        Select customer_id, sales
        from (Select customer_id,
                  sum(quantity * price) as SALES,
                  DENSE_RANK() over (order by sum(quantity * PRICE) desc) as dr
                  from TABLERETAIL
                  group by customer_id) ranked_sales
        where dr <= 30;
```

output:



CUSTOMER_ID	SALES
12931	42055.96
12748	33719.73
12901	17654.54
12921	16587.09
12939	11581.8
12830	6814.64
12839	5591.42
12971	5190.74
12955	4757.16
12747	4196.01
12949	4167.22
12749	4090.88
12867	4036.82
12841	4022.35
12957	4017.54
12910	3075.04
12916	3006.15

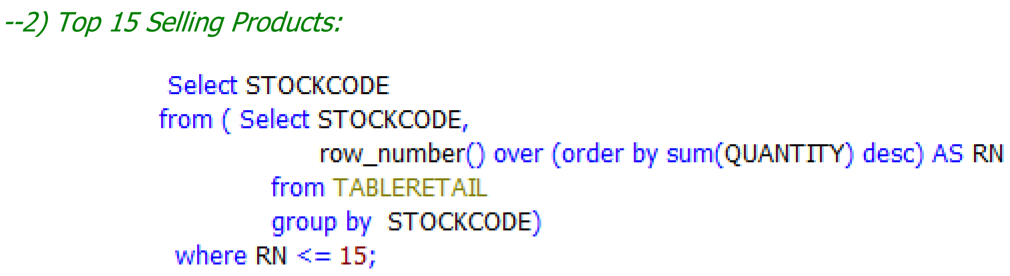
116 msecs   Row 1 of 30 total rows

Query2:

Business Description:

This query retrieves the STOCKCODE for the top 15 selling products from the store's sales data, ranked by their total sales QUANTITY.
Which can help the business for understanding customer preferences and identifying the product categories that are most popular among customers.
.

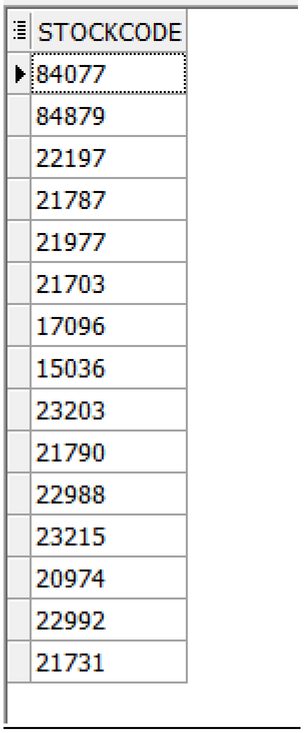
Query:



```
--2) Top 15 Selling Products:

                 Select STOCKCODE
                from ( Select STOCKCODE,
                            row_number() over (order by sum(QUANTITY) desc) AS RN
                     from TABLERETAIL
                     group by  STOCKCODE)
                 where RN <= 15;
```

output:



STOCKCODE
84077
84879
22197
21787
21977
21703
17096
15036
23203
21790
22988
23215
20974
22992
21731

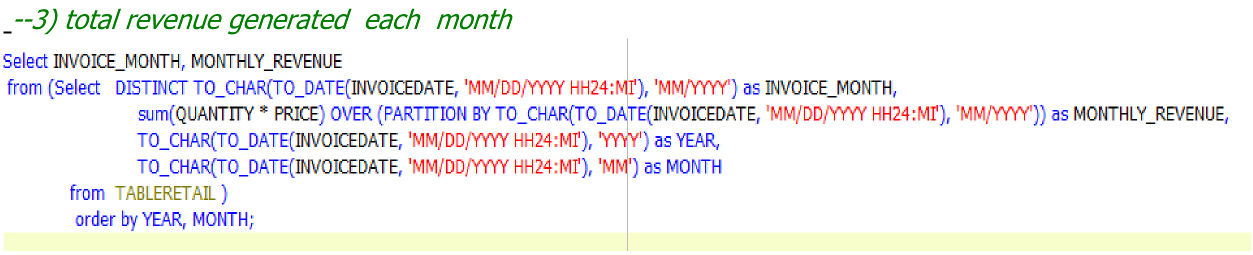
Query3:

Business Description:

This query retrieves the monthly revenue and by analyzing this information ,businesses can :

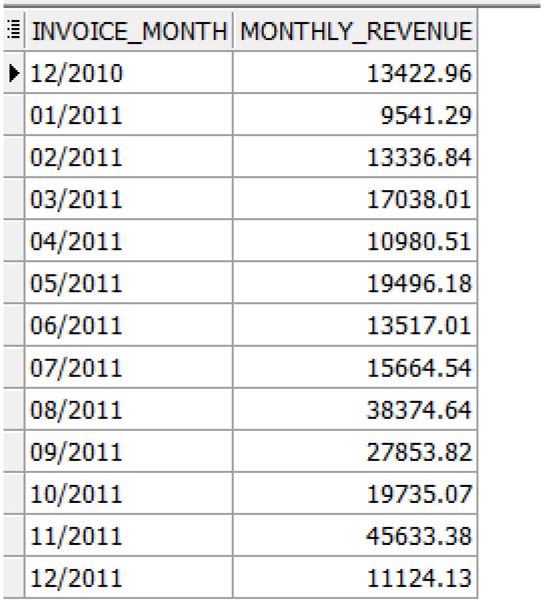
Identify if there are specific months with higher or lower revenue, which can be helpful for planning inventory and marketing(Seasonal trends).

Query:



```
_--3) total revenue generated  each  month
Select INVOICE_MONTH, MONTHLY_REVENUE
 from (Select  DISTINCT TO_CHAR(TO_DATE(INVOICEDATE, 'MM/DD/YYYY HH24:MI'), 'MM/YYYY') as INVOICE_MONTH,
                sum(QUANTITY * PRICE) OVER (PARTITION BY TO_CHAR(TO_DATE(INVOICEDATE, 'MM/DD/YYYY HH24:MI'), 'MM/YYYY')) as MONTHLY_REVENUE,
                TO_CHAR(TO_DATE(INVOICEDATE, 'MM/DD/YYYY HH24:MI'), 'YYYY') as YEAR,
                TO_CHAR(TO_DATE(INVOICEDATE, 'MM/DD/YYYY HH24:MI'), 'MM') as MONTH
        from  TABLERETAIL )
         order by YEAR, MONTH;
```

□ **output:**



INVOICE_MONTH	MONTHLY_REVENUE
12/2010	13422.96
01/2011	9541.29
02/2011	13336.84
03/2011	17038.01
04/2011	10980.51
05/2011	19496.18
06/2011	13517.01
07/2011	15664.54
08/2011	38374.64
09/2011	27853.82
10/2011	19735.07
11/2011	45633.38
12/2011	11124.13

Query4:

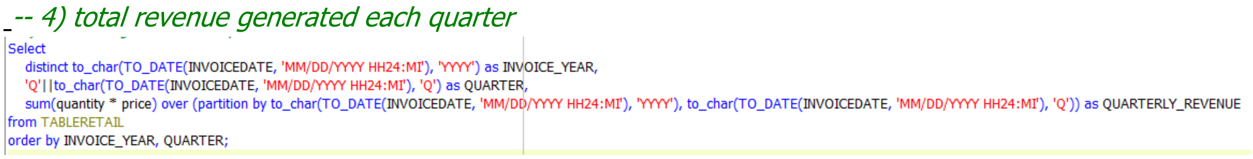
Business Description:

This query retrieves the sales for each quarter in the year and by analyzing this information, businesses can:

if there are specific quarters where sales are higher or lower, indicating potential seasonal trends.

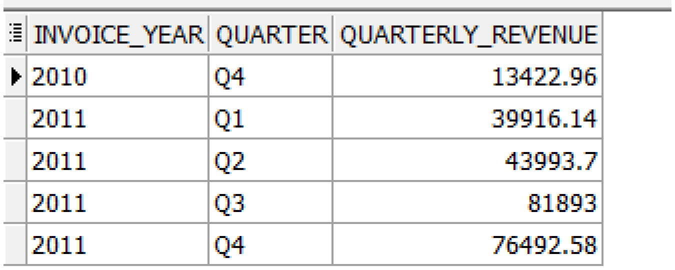
Identifying periods with lower sales might enable the businesses to implement targeted marketing campaigns or promotions to increase sales.

Query:



```
-- 4) total revenue generated each quarter
Select
    distinct to_char(TO_DATE(INVOICEDATE, 'MM/DD/YYYY HH24:MI'), 'YYYY') as INVOICE_YEAR,
    'Q'||to_char(TO_DATE(INVOICEDATE, 'MM/DD/YYYY HH24:MI'), 'Q') as QUARTER,
    sum(quantity * price) over (partition by to_char(TO_DATE(INVOICEDATE, 'MM/DD/YYYY HH24:MI'), 'YYYY'), to_char(TO_DATE(INVOICEDATE, 'MM/DD/YYYY HH24:MI'), 'Q')) as QUARTERLY_REVENUE
from TABLERETAIL
order by INVOICE_YEAR, QUARTER;
```

output:



INVOICE_YEAR	QUARTER	QUARTERLY_REVENUE
2010	Q4	13422.96
2011	Q1	39916.14
2011	Q2	43993.7
2011	Q3	81893
2011	Q4	76492.58

Query5:

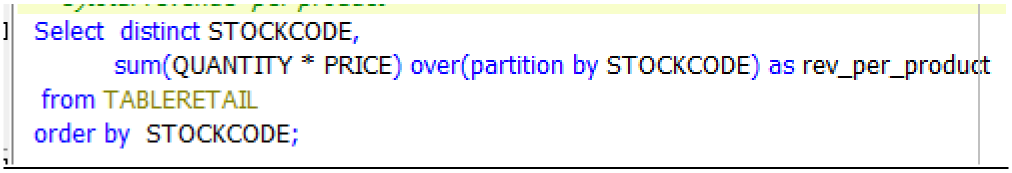
Business Description:

This query retrieves the revenue for each product and by analyzing this information, businesses can:

- Highlighting **products with low revenue** that might need further analysis or marketing efforts.
- Analyzing historical revenue data for different products can aid in forecasting future sales trends.

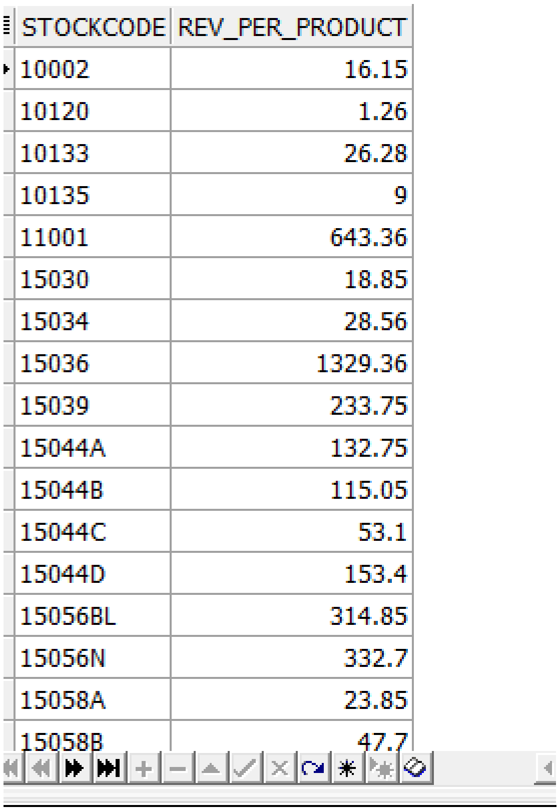
Query:

--5) total revenue per product



```
Select  distinct STOCKCODE,
          sum(QUANTITY * PRICE) over(partition by STOCKCODE) as rev_per_product
 from TABLERETAIL
order by  STOCKCODE;
```

output:



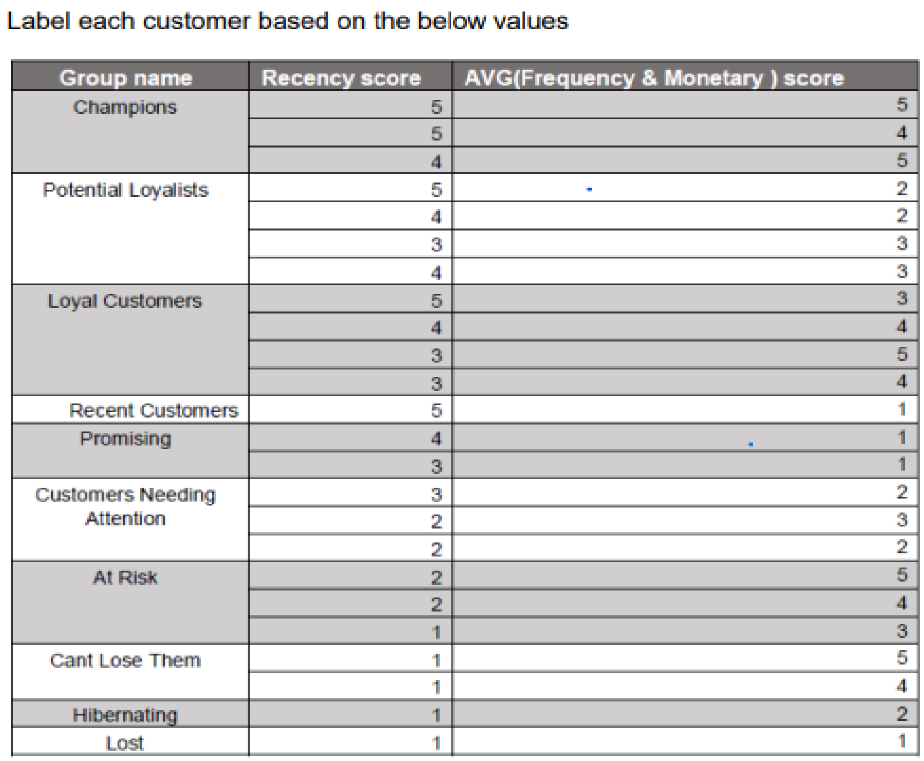
STOCKCODE	REV_PER_PRODUCT
10002	16.15
10120	1.26
10133	26.28
10135	9
11001	643.36
15030	18.85
15034	28.56
15036	1329.36
15039	233.75
15044A	132.75
15044B	115.05
15044C	53.1
15044D	153.4
15056BL	314.85
15056N	332.7
15058A	23.85
15058B	47.7

Problem_2:

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, For this, we would decrease the permutations by getting the average scores of the frequency and monetary



Label each customer based on the below values

Group name	Recency score	AVG(Frequency & Monetary) score
Champions	5	5
	5	4
	4	5
Potential Loyalists	5	2
	4	2
	3	3
	4	3
Loyal Customers	5	3
	4	4
	3	5
	3	4
Recent Customers	5	1
Promising	4	1
	3	1
Customers Needing Attention	3	2
	2	3
	2	2
At Risk	2	5
	2	4
	1	3
Cant Lose Them	1	5
	1	4
Hibernating	1	2
Lost	1	1

Business Description:

This SQL query aims to segment customers based on their recency, frequency, and monetary values and then classifies customers into different segments based on these metrics.

Query:

ri :

returns for each customer his recency, frequency, and the monetary value.

r2:

Assigning Segment Scores

assigns a score (1-5) based on recency, with 5 being the most recent.

assigns a score (1-5) based on purchase frequency, with 5 being the most frequent buyer.

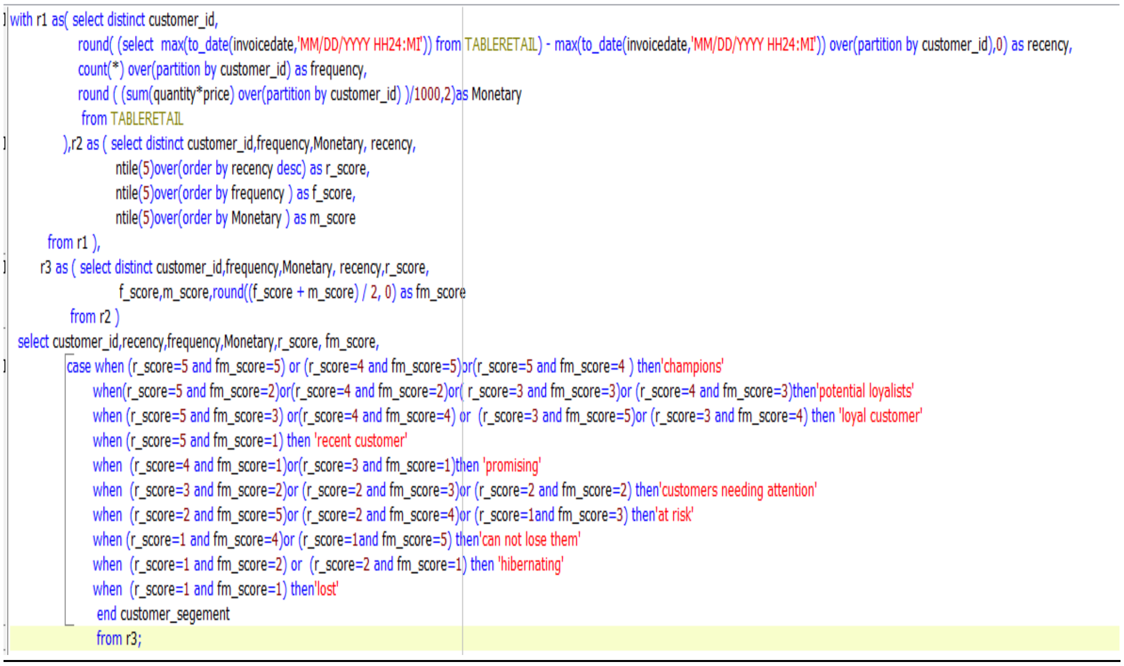
assigns a score (1-5) based on total spending, with 5 being the highest spender.

r3:

the average of the f_score (frequency score) and m_score (monetary score) and then round it to the nearest whole number.

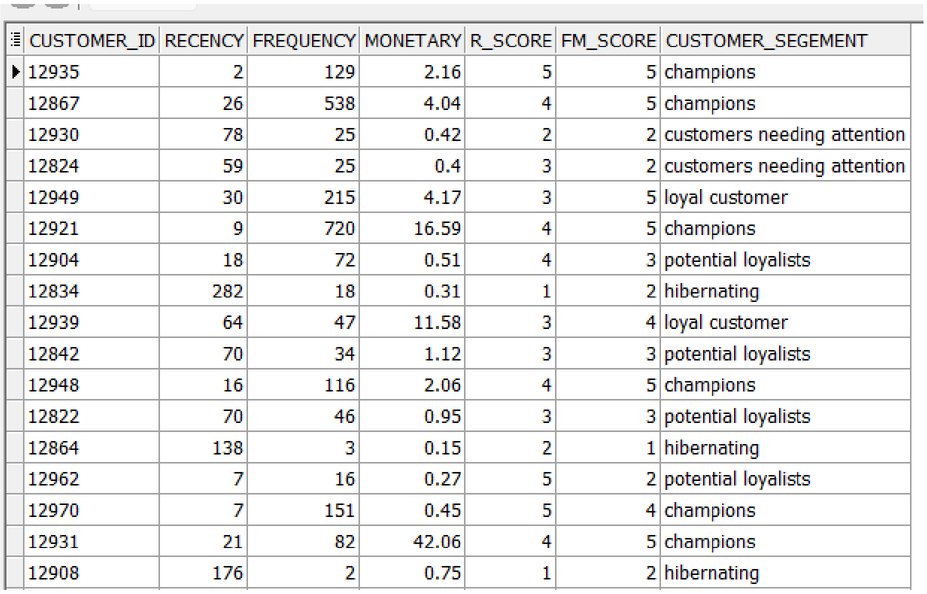
Final sselect :

Based on the recency and fm score , assigns each customer to a segment based on their purchase behaviour.



```
with r1 as( select distinct customer_id,
            round( (select  max(to_date(invoicedate,'MM/DD/YYYY HH24:MI')) from TABLERETAIL) - max(to_date(invoicedate,'MM/DD/YYYY HH24:MI')) over(partition by customer_id),0) as recency,
            count(*) over(partition by customer_id) as frequency,
            round ( (sum(quantity*price) over(partition by customer_id) )/1000,2)as Monetary
             from TABLERETAIL
          ),r2 as ( select distinct customer_id,frequency,Monetary, recency,
                   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 r1 ),
      r3 as ( select distinct customer_id,frequency,Monetary, recency,r_score,
                   f_score,m_score,round((f_score + m_score) / 2, 0) as fm_score
            from r2 )
 select customer_id,recency,frequency,Monetary,r_score, fm_score,
          case when (r_score=5 and fm_score=5) or (r_score=4 and fm_score=5)or(r_score=5 and fm_score=4 ) 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 customer'
              when (r_score=5 and fm_score=1) then 'recent customer'
              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=1and fm_score=3) then'at risk'
              when (r_score=1 and fm_score=4)or (r_score=1and fm_score=5) then'can not 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'
               end customer_segement
               from r3;
```

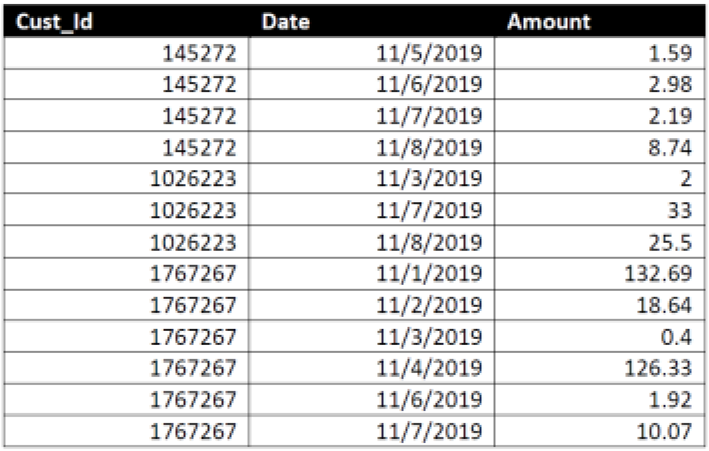
Output:



CUSTOMER_ID	RECENCY	FREQUENCY	MONETARY	R_SCORE	FM_SCORE	CUSTOMER_SEGEMENT
12935	2	129	2.16	5	5	champions
12867	26	538	4.04	4	5	champions
12930	78	25	0.42	2	2	customers needing attention
12824	59	25	0.4	3	2	customers needing attention
12949	30	215	4.17	3	5	loyal customer
12921	9	720	16.59	4	5	champions
12904	18	72	0.51	4	3	potential loyalists
12834	282	18	0.31	1	2	hibernating
12939	64	47	11.58	3	4	loyal customer
12842	70	34	1.12	3	3	potential loyalists
12948	16	116	2.06	4	5	champions
12822	70	46	0.95	3	3	potential loyalists
12864	138	3	0.15	2	1	hibernating
12962	7	16	0.27	5	2	potential loyalists
12970	7	151	0.45	5	4	champions
12931	21	82	42.06	4	5	champions
12908	176	2	0.75	1	2	hibernating

Problem_3:

given the below dataset this dataset contains 10000 rows, which is the daily purchasing transactions for customers.



Cust_Id	Date	Amount
145272	11/5/2019	1.59
145272	11/6/2019	2.98
145272	11/7/2019	2.19
145272	11/8/2019	8.74
1026223	11/3/2019	2
1026223	11/7/2019	33
1026223	11/8/2019	25.5
1767267	11/1/2019	132.69
1767267	11/2/2019	18.64
1767267	11/3/2019	0.4
1767267	11/4/2019	126.33
1767267	11/6/2019	1.92
1767267	11/7/2019	10.07

answer two questions:

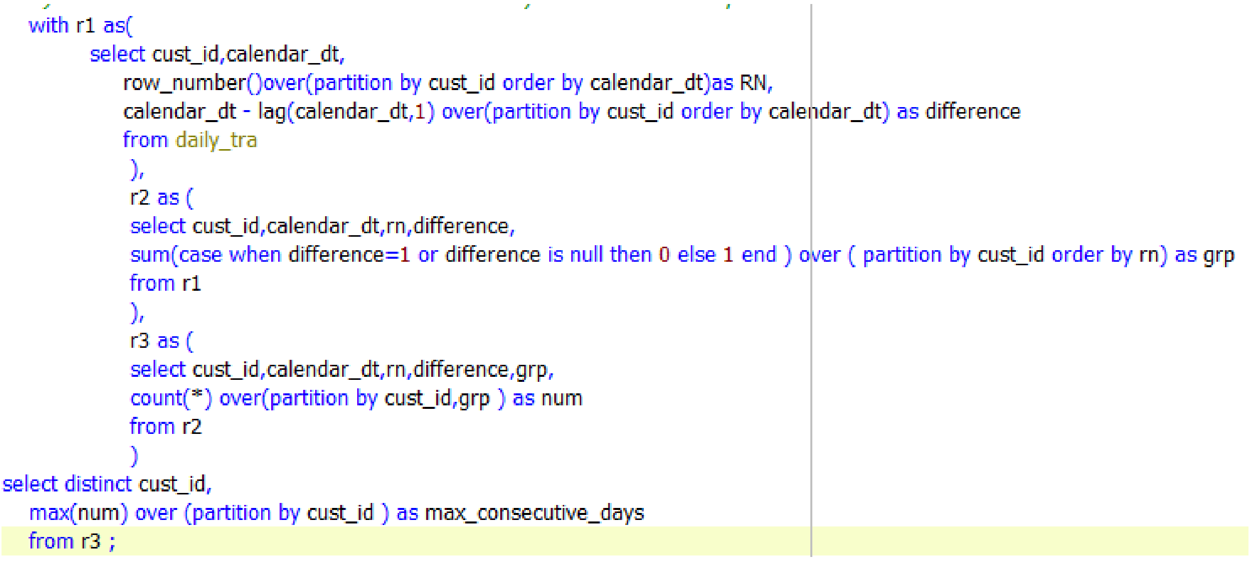
a- What is the maximum number of consecutive days a customer makes purchases?

Query:

r2:
It assigns a group (grp) to the consecutive days based on the calculated differences.
If the difference between consecutive days is 1 or null, it treats them as part of the same group otherwise, it starts a new group.

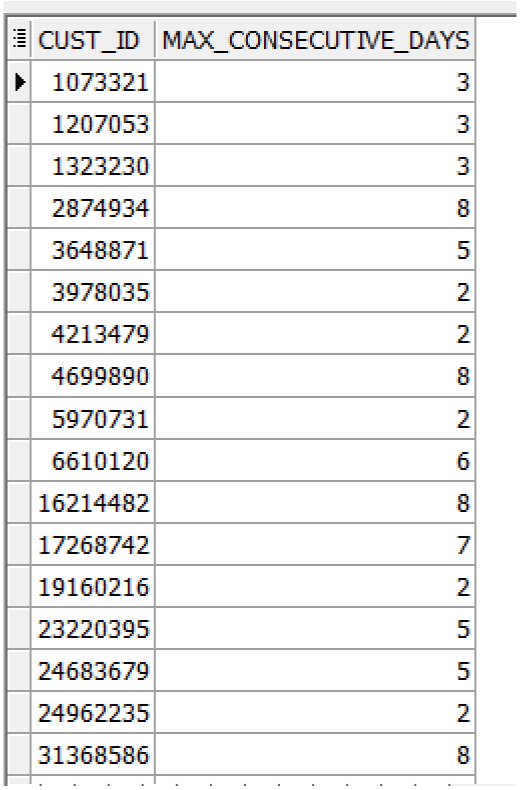
r3:
This counts the consecutive days of the purchase for each customer in the same group.

final query:
It calculates the maximum consecutive days of purchases (max_consecutive_days) for each customer using the MAX function over the num column, which represents the count of consecutive days in each group.



```
with r1 as(
        select cust_id,calendar_dt,
             row_number()over(partition by cust_id order by calendar_dt)as RN,
             calendar_dt - lag(calendar_dt,1) over(partition by cust_id order by calendar_dt) as difference
             from daily_tra
              ),
              r2 as (
              select cust_id,calendar_dt,rn,difference,
              sum(case when difference=1 or difference is null then 0 else 1 end ) over ( partition by cust_id order by rn) as grp
              from r1
              ),
              r3 as (
              select cust_id,calendar_dt,rn,difference,grp,
              count(*) over(partition by cust_id,grp ) as num
              from r2
              )
select distinct cust_id,
    max(num) over (partition by cust_id ) as max_consecutive_days
    from r3 ;
```

Output:

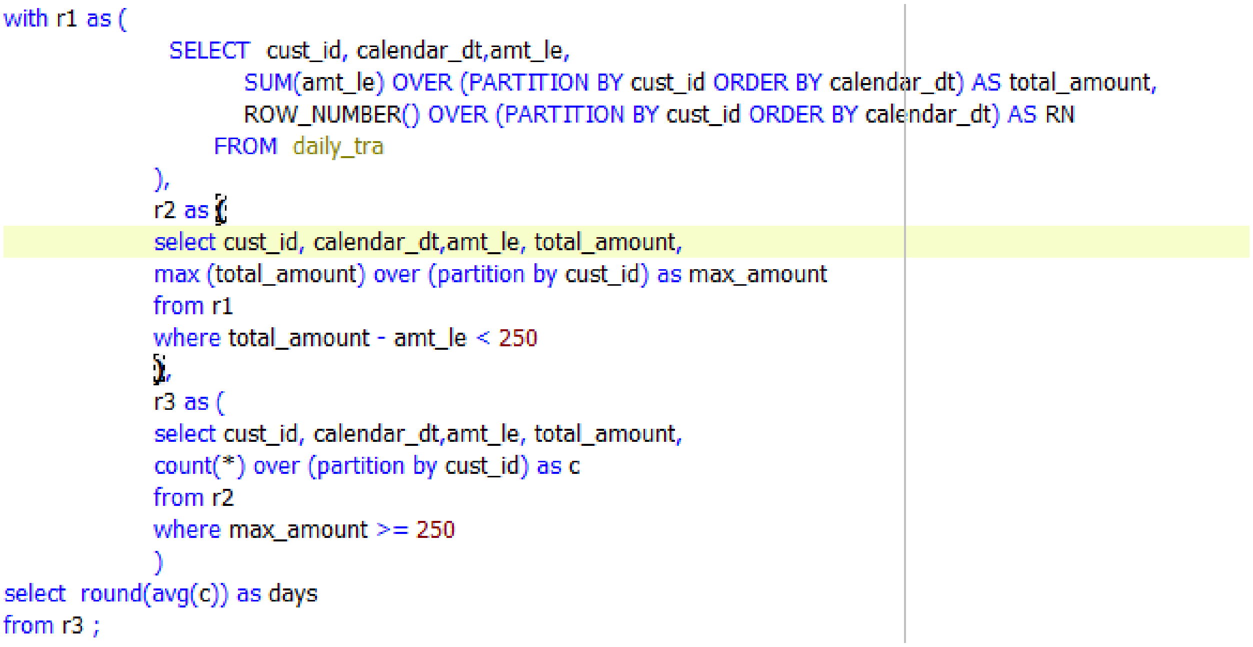


CUST_ID	MAX_CONSECUTIVE_DAYS
1073321	3
1207053	3
1323230	3
2874934	8
3648871	5
3978035	2
4213479	2
4699890	8
5970731	2
6610120	6
16214482	8
17268742	7
19160216	2
23220395	5
24683679	5
24962235	2
31368586	8

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

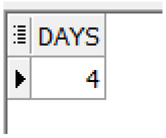
Query:

R1:
It returns the running total of amt_le for each customer by grouping by cust_id and ordering by calendar_dt.
It also assigns a row number (within each customer's data)
R2:
keep rows where the difference between total_amount and amt_le is less than 250.
It then calculates the maximum total_amount reached for each customer.
R3:
keeping only rows where max_amount is greater than or equal to 250. Then we will get the count of customers who had a running total exceeding the threshold .
Finally:
it calculates the count of days for each customer that met the condition in r3



```
with r1 as (
                SELECT  cust_id, calendar_dt,amt_le,
                       SUM(amt_le) OVER (PARTITION BY cust_id ORDER BY calendar_dt) AS total_amount,
                       ROW_NUMBER() OVER (PARTITION BY cust_id ORDER BY calendar_dt) AS RN
                    FROM  daily_tra
              ),
              r2 as (
              select cust_id, calendar_dt,amt_le, total_amount,
              max (total_amount) over (partition by cust_id) as max_amount
              from r1
              where total_amount - amt_le < 250
              ),
              r3 as (
              select cust_id, calendar_dt,amt_le, total_amount,
              count(*) over (partition by cust_id) as c
              from r2
              where max_amount >= 250
              )
select  round(avg(c)) as days
from r3 ;
```

output:



	DAYS
▶	4