Analytical SQL Business Queries

Q1 :-

Query 1:-

```
select stockcode, count(*) as purchasecount,
     rank() over (order by count(*) desc) as purchaserank
from tableretail
group by stockcode;
```

This SQL query is designed to analyze the purchase frequency of different products in a retail dataset (Top Selling Ptoducts).

STOCKCODE	PURCHASECOUNT	PURCHASERANK
84879	61	1
22086	56	2
85099B	53	3
22197	52	4
85123A	49	5
47566	49	5
23298	49	5

Query 2:-

```
with productpricerange as (
  select stockcode,
       min(price) over (partition by stockcode) as minprice,
       max(price) over (partition by stockcode) as maxprice
  from tableretail
)
select stockcode, minprice, maxprice
from productpricerange
group by stockcode, minprice, maxprice;
```

This SQL query is designed to analyze the price range for each product in a retail dataset. It focuses on understanding the minimum and maximum prices associated with each unique product, providing valuable insights into the pricing variability within the dataset

STOCKCODE	MINPRICE	MAXPRICE
10120	0.21	0.21
11001	1.27	1.69
15044C	2.95	2.95
16008	0.12	0.25
16014	0.42	0.42
16048	0.12	0.12
16258A	0.42	0.42
20616	0.75	0.75
20619	2.1	2.1
20700	3.75	3.75
20727	1.65	1.65
20748	12.75	12.75
20769	2.55	2.55

Query 3:-

```
with customerpurchasepattern as (
  select customer_id,
       to_char(to_date(invoicedate, 'mm/dd/yyyy hh24:mi'), 'mm-yyyy') as purchasemonth,
       count(distinct invoice) as monthlypurchases
  from tableretail
  group by customer_id,  to_char(to_date(invoicedate, 'mm/dd/yyyy hh24:mi'), 'mm-yyyy')
)
select customer_id, purchasemonth,
     monthlypurchases,
     lag(monthlypurchases) over (partition by customer_id order by purchasemonth) as
previousmonthpurchases
from customerpurchasepattern;
```

This SQL query is designed to analyze the purchasing patterns of customers in a retail context, specifically focusing on the monthly trends in their buying behavior

CUSTOMER_ID	PURCHASEMONTH	MONTHLYPURCHASES	PREVIOUSMONTHPURCHASES
12747	01-2011	1	
12747	03-2011	1	1
12747	05-2011	2	1
12747	06-2011	1	2
12747	08-2011	1	1
12747	10-2011	1	1
12747	11-2011	1	1
12747	12-2010	2	1

Query 4:-

```
select customer_id, total_sales
from (
   select customer_id, round(sum(quantity * price)) as total_sales,
        ntile(5) over (order by round(sum(quantity * price)) desc) as percentile
   from tableretail
   group by customer_id
)
where percentile = 1;
```

This query divides the customers into five groups based on their total sales, with group 1 containing the customers with the highest total sales

CUSTOMER_ID	TOTAL_SALES
12931	42056
12748	33720
12901	17655
12921	16587
12939	11582
12830	6815
12839	5591
12971	5191

Query 5:-

```
select distinct year, count(*) over(partition by year, country) number_of_orders
from (select distinct invoice ,to_char(to_date(invoicedate, 'mm/dd/yyyy hh24:mi'), 'yyyy') as
year , country
from tableretail) x
order by 1 , 2;
```

This query will give us the count of total orders for each country for each year

YEAR	NUMBER_OF_ORDERS
2010	68
2011	649

Query 6:-

```
select distinct stockcode , round(sum(quantity ) over(partition by stockcode)) as
quantity_sold
from tableretail
where quantity > 0
order by quantity_sold desc;
```

this simple query is to identify the products that have sold the most quantity in the given period.

STOCKCODE	QUANTITY_SOLD
84077	7824
84879	6117
22197	5918
21787	5075
21977	4691
21703	2996
17096	2019
15036	1920
23203	1803
21790	1579

Query 7:-

```
with customer_sales as (
  select
    customer_id,
    country,
    round(sum(quantity * price), 2) as total_sales
  from tableretail
  group by customer_id, country
),
ranked_sales as (
  select
    customer_id,
    country,
    total_sales,
    row_number() over(partition by country order by total_sales desc) as sales_rank
  from customer_sales
)
select
    customer_id,
  total_sales, sales_rank
from ranked_sales
where sales_rank <= 3
order by country, total_sales desc;
```

This query selects the top 3 customers based on their sales rank and returns customer ID and total sales.

CUSTOMER_ID	TOTAL_SALES	SALES_RANK
12931	42055.96	1
12748	33719.73	2
12901	17654.54	3

Query 8:-

```
select customer_id, max(invoicedate) as lastpurchasedate
from tableretail
group by customer_id;
```

This query describes Customer Purchase Recency (Determine how recently customers made a purchase to identify active customers) .

CUSTOMER_ID	LASTPURCHASEDATE
12828	9/1/2011 17:14
12829	12/14/2010 14:54
12833	7/17/2011 13:46
12841	9/11/2011 12:13
12844	8/1/2011 13:11
12856	12/2/2011 10:52
12883	9/11/2011 15:18
12884	9/12/2011 13:10
12916	7/24/2011 10:35
12921	9/2/2011 13:01
12933	6/21/2011 16:13
12826	9/29/2011 10:55
12830	9/9/2011 15:02
12840	7/19/2011 9:29
12849	3/16/2011 12:26

Q2 :-

```
with cte_1 as (
  select customer_id, quantity, price ,invoice,
       to_date(invoicedate, 'mm/dd/yyyy hh24:mi') as resent_date
  from tableretail
), cte_2 as (
  select distinct
       customer_id,
       round(max(resent_date) over () - max(resent_date) over (partition by customer_id)) as
recency,
       count(distinct(invoice)) over (partition by customer_id) as frequency,
       sum(price * quantity) over (partition by customer_id) as monetary,
       case
          when round(max(resent_date) over () - max(resent_date) over (partition by
customer_id)) <= 30 then 5
          when round(max(resent_date) over () - max(resent_date) over (partition by
customer_id)) <= 60 then 4
          when round(max(resent_date) over () - max(resent_date) over (partition by
customer_id)) <= 90 then 3
          when round(max(resent_date) over () - max(resent_date) over (partition by
customer_id)) <= 180 then 2
          else 1
```

```
        end as r_score,
        case
          when count(*) over (partition by customer_id) > 100 and sum(price * quantity) over
(partition by customer_id) > 1000 then 5
          when count(*) over (partition by customer_id) > 50 and sum(price * quantity) over
(partition by customer_id) > 500 then 4
          when count(*) over (partition by customer_id) > 25 and sum(price * quantity) over
(partition by customer_id) > 250 then 3
          when count(*) over (partition by customer_id) > 10 and sum(price * quantity) over
(partition by customer_id) > 100 then 2
          else 1
        end as fm_score
  from cte_1 c
)
select customer_id  , recency, frequency ,  monetary ,  r_score ,  fm_score,
  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 loyalist'
    when r_score = 4 and fm_score in ( 2 , 3)    then 'potential loyalist'
    when r_score = 3 and fm_score    = 3        then 'potential loyalist'

    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 customer'

    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 'cannot lose them'
    when r_score  =  1  and fm_score = 2  then 'hibernating'
    when r_score  =  1  and fm_score = 1  then 'lost'

else 'about to sleep'

end as customer_segmentation
from cte_2;
```

First Identifies the earliest date (final_date) when each customer reached the spending threshold then Filters customers who have not reached the spending threshold then Identifies the earliest date (first_date) for customers who did not reach the spending threshold then Joins the CTEs cte11 and cte22 to calculate the number of days it took for each customer to reach the spending threshold and finallyComputes the average number of days it takes for customers to reach the spending threshold.

CUSTOMER_ID	RECENCY	FREQUENCY	MONETARY	R_SCORE	FM_SCORE	CUSTOMER_SEGMENTATION
12749	3	5	4090.88	5	5	Champions
12826	2	7	1474.72	5	4	Champions
12830	37	6	6814.64	4	3	Potential Loyalist
12833	145	1	417.38	2	2	Needs Attention
12849	31	2	1050.89	4	4	Loyal Customers
12851	96	1	135.18	2	2	Needs Attention
12879	44	3	573.22	4	1	Promising
12884	88	1	309.05	3	1	Promising
12908	176	2	750	2	1	About to sleep
12920	17	1	164.23	5	2	Potential Loyalist
12945	288	1	462.95	1	2	Hibernating
12950	2	3	1843	5	2	Potential Loyalist

Q3 :-

a-

```
with consecutive_days as (
  select
    cust_id,
    calendar_dt,
    amt_le,
    row_number() over (partition by cust_id order by calendar_dt) as seq,
    calendar_dt - row_number() over (partition by cust_id order by calendar_dt) as grp
  from customers
),
max_consecutive_days as (
  select
    cust_id,
    calendar_dt,
    amt_le,
    seq,
    grp,
    count(*) over (partition by cust_id, grp) as consecutive_days_count
  from consecutive_days
  where amt_le > 0
)
select cust_id, max(consecutive_days_count)
from max_consecutive_days
group by cust_id;
```

Consecutive_days, a CTE, groups the purchases by customer and, using the difference between the row number and the calendar date, generates a new grouping column grp. We utilize this grouping field to figure out how many days in a row each consumer purchased. The max_consecutive_days CTE aggregates by customer, removes zero purchase amounts, and counts the number of unique groups of consecutive days on which each customer made purchases.

The maximum number of days in a row that each customer has made is chosen by the outer query.

CUST_ID	MAX(CONSECUTIVE_DAYS_COUNT)
150488	9
259866	8
480780	11
505790	22
533068	3
535101	2
811892	9
839622	61
999683	7
1311280	2
1327831	8
1331618	9
1519955	61

b-

```
with cte_1 as(
select cust_id,calendar_dt,amt_le,sum(amt_le) over(partition by cust_id order by
calendar_dt) as total_amount
from customers),

cte_2 as(
select cte_1.*, case when total_amount >= 250 then 1 else 0 end as reached_threshold
from cte_1
where total_amount >= 250
),
cte_3 as(
select cte_1.*, case when total_amount >= 250 then 1 else 0 end as
before_reached_threshold
from cte_1
where total_amount <250
),
cte_4 as (select c3.cust_id, count(before_reached_threshold) as
transactions_before_threshold
from cte_3 c3
where c3.cust_id in (select cust_id from cte_2)
group by c3.cust_id)

select trunc(avg(transactions_before_threshold)) as transactions_before_threshold
from cte_4;
```

TRANSACTIONS_BEFORE_THRESHOLD
6

```sql
with cte_1 as (
  select
    cust_id,
    calendar_dt,
    amt_le,
    sum(amt_le) over (partition by cust_id order by calendar_dt) as total_amount
  from customers
),
cte_2 as (
  select
    cte_1.*,
    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,
    min(calendar_dt) as final_date
  from cte_2
  group by cust_id
),
cte_4 as (
  select
    cte_1.*,
    case when total_amount < 250 then 1 else 0 end as reached_threshold
  from cte_1
  where total_amount < 250
),
cte_5 as (
  select
    cust_id,
    min(calendar_dt) as first_date
  from cte_4
  group by cust_id
),
cte_6 as (
  select
    c5.cust_id,
    (c3.final_date - c5.first_date) as days_before_reaching_threshold
  from cte_5 c5
  join cte_3 c3 on c5.cust_id = c3.cust_id
)

select trunc(avg(days_before_reaching_threshold)) as avg_days_before_threshold
from cte_6;
```

AVG_DAYS_BEFORE_THRESHOLD
13