**ATOM CAMP**

**AI AND DS BOOTCAMP**

SQL PORTFOLIO PROJECT REPORT

|  |  |
| --- | --- |
| **NAME** | Sarah Qasim |
| **MODULE NAME** | SQL |
| **DATE** | 16-Nov-2024 |

**Module 1: Sales Performance Analysis**

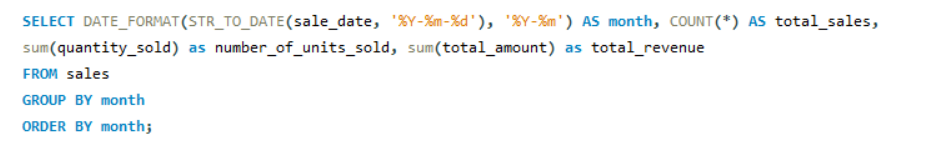
**1. Total Sales per Month:**

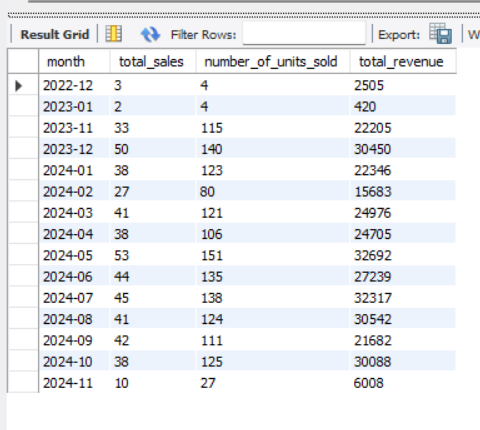
* ***Calculate the total sales amount per month, including the number of units sold and the total revenue generated.***

My sale\_date column is saved with the data type “varchar”, so I format it from to str\_to\_date and use date\_format to save it using month-year to find sales per month.

We print total sales by counting total rows in the sales table, number of units sold by finding the sum of quantity sold column and the revenue by finding sum of total amount column.

Group and order by month column created earlier.



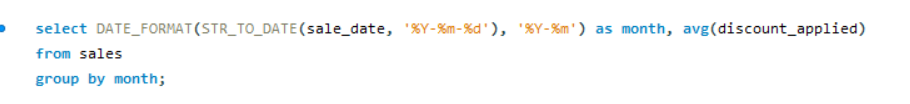


**2. Average Discount per Month:**

* ***Calculate the average discount applied to sales in each month***

My sale\_date column is saved with the data type “varchar”, so I format it from to str\_to\_date and use date\_format to save it using month-year with the column name “month”.

We group by month and print the aggregate function: avg(discount\_applied)



A screenshot of a computer

Description automatically generated

* ***Assess how discounting strategies impact total sales.***

To see relation between each discount offered and total sales, we group by discount\_applied and use the aggregate function: sum(quantity\_sold) as total\_quantity\_sold.

A white background with black text

Description automatically generated

A screenshot of a computer

Description automatically generated

**Module 2: Customer Behaviour and Insights**

**3. Identify high-value customers:**

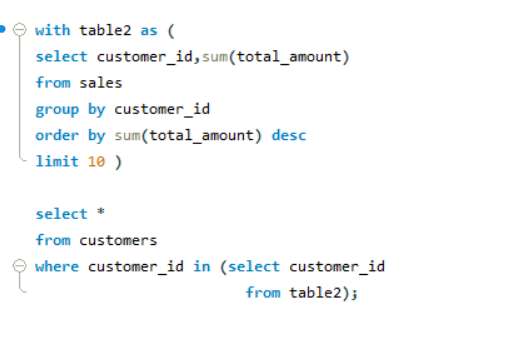
* ***Which customers have spent the most on their purchases? Show their details (showing top10)***

Here we need customer details from the customers table and each customer’s total\_amount spent from the sales table.

The two tables have a common column: customer\_id.

We make a CTE called table2 which uses the sales table and groups by customer\_id to get the sum(total\_amount) thus combining amount spend in each sale by each customer. The resulting rows are ordered by sum(total\_amount) in descending order so customers who have spent the most on their purchases are placed on top of the table. A limit of 10 is set to get the most profitable customers.

Finally, we print all columns from the customers table, setting the where clause to customer\_id column including all customer\_id values in the CTE: table2.

******

***A screenshot of a computer

Description automatically generated***

**4. Identify the oldest Customer:**

* ***Find the details of customers born in the 1990s, including their total spending and specific order details.***

We need details of customers from the customers table and each customer’s total spending and specific order details from the sales table, so we join the two tables using inner join on the common column: customer\_id. This gives us all the common customers so we can print information from both tables.

We then use the window function to partition by customer\_id over the aggregate function sum(total\_amount). We use this instead of group by as group by does not allow select clause to contain any other columns except those with aggregate functions or group by.

Finally, the where clause is added filter for values from 1st Jan 1990 to 31st Dec 1999.

***A computer code with text

Description automatically generated with medium confidence***

***A screenshot of a computer

Description automatically generated***

**5. Customer Segmentation:**

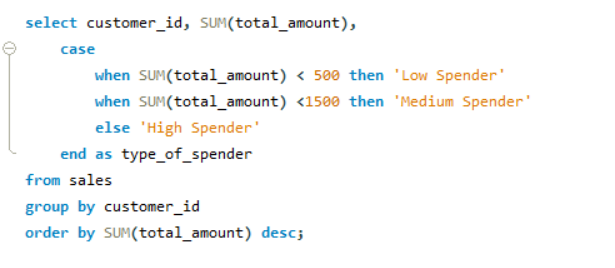
* ***Use SQL to create customer segments based on their total spending (e.g., Low Spenders, High Spenders).***

We select customer\_id, sum(total\_amount) and case statements using sum(total\_amount):

* Less than 500: ‘Low spender’
* Less than 1500: ‘Medium spender’
* Remaining : ‘High spender’

from the sales table.

Finally, we group by customer\_id to compare using case statements for each customer.



A screenshot of a computer

Description automatically generated

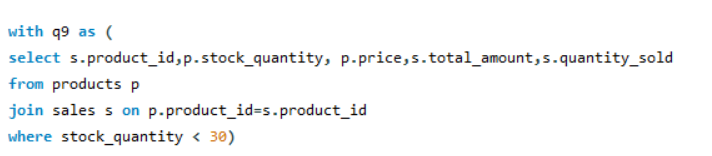
**Module 3: Inventory and Product Management**

**6. Stock Management:**

* ***Write a query to find products that are running low in stock (below a threshold like 10 units) and recommend restocking amounts based on past sales performance.***

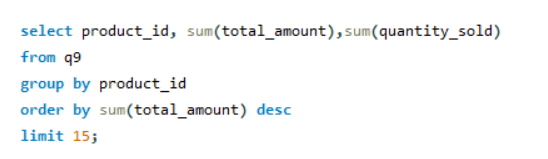
***Threshold = 30***

We join sales and products tables. Then, create a CTE which stores the joined tables with where clause condition: stock\_quantity < 30

******

For sales performance, we look at sum(total\_amount) and sum(quantity\_sold) for each product.

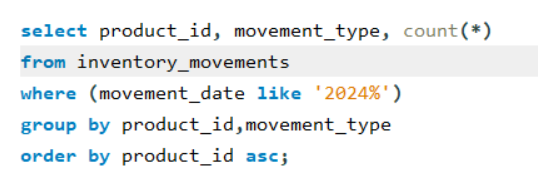
We then order by sum(total\_amount) and set a limit at 15 items out of the 25 initial products that were low in stock.



**7. Inventory Movements Overview:**

***Create a report showing the daily inventory movements (restock vs. sales) for each product over a given period.***

* *Using the latest period: 2024*
* *Out: sale*
* *In: restock*
* *Grouping by product\_id and movement\_type for count of rows*
* *to see each product along with the count of “in” and “out”*

******

***A screenshot of a computer

Description automatically generated***

**8. Rank Products:**

***Rank products in each category by their prices.***

We print category name, prices and product\_id using the window function, dense\_rank over a partition by category to separate by category names and print rankings per category using order by price in descending order so highest price items in each category are ranked 1.

***A close-up of a white background

Description automatically generated***

***A screenshot of a computer

Description automatically generated***

**Module 4: Advanced Analytics**

**9. Average order size:**

***What is the average order size in terms of quantity sold for each product?***

Using sales table, we order by product\_id using the aggregate function: avg(quantity\_sold) and rounding off to 2 decimal places.

***A screenshot of a computer code

Description automatically generated***

***A screenshot of a computer

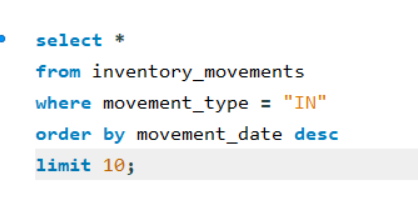
Description automatically generated***

**10. Recent Restock Product:**

***Which products have seen the most recent restocks?***

Most recent: order by movement\_date in descending order to get most recent on top and set limit to 10.

Restocks: condition in where clause: movement\_type = “IN”.

******

***A screenshot of a computer

Description automatically generated***

**Advanced Features to Challenge Students**

**Dynamic Pricing Simulation:**

* ***Challenge students to analyse how price changes for products impact sales volume, revenue, and customer behaviour.***

**Dynamic pricing:** pricing strategy where prices are adjusted in real-time based on various factors, such as market demand, customer behaviour, and competitor pricing.

**Impact on sales volume and revenue:** By adjusting prices in real-time based on demand, market conditions, and customer behaviour, businesses can maximize their earnings.

**Impact on customer behaviour:** Dynamic pricing has a profound effect on how customers perceive value and make purchasing decisions.

The fluctuating nature of prices can create a sense of urgency, which often triggers impulse buying, as customers rush to secure what they perceive as a bargain.

However, dynamic pricing can also lead to negative emotions, such as frustration or distrust, particularly if customers notice frequent or unpredictable price changes.

**Simulation:** based on these factors, some assumptions are made in the simulation:

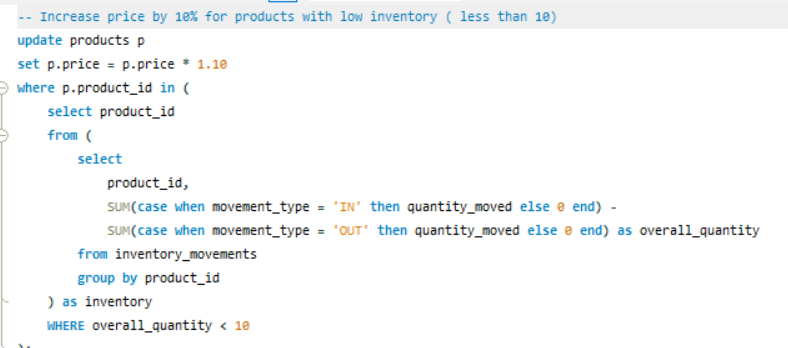
***Demand:*** Products with high sales volume in the sales table will see a price increase in products table.

A computer code with text

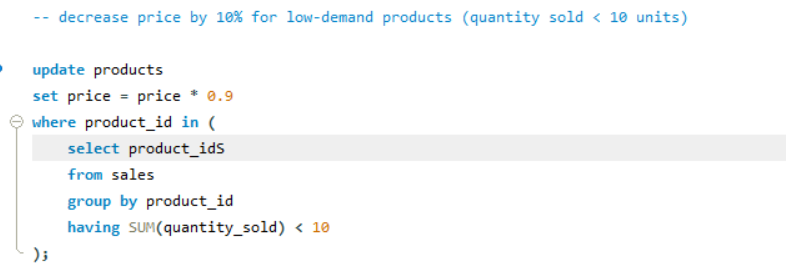
Description automatically generated

Here we use the group by and having clauses along with a subquery in the where clause.

***Scarcity:*** Products with low inventory in the inventory table will also see a price increase in products table.



***Clear stock:*** Products with low sales volume will see a price decrease.



**Customer Purchase Patterns:**

* ***Analyse purchase patterns using time-series data and window functions to find high-frequency buying behaviour.***

We need to find the time difference between purchases for each customer from “sales\_date” in sales table using the lag() window function.

We print the customer\_id, sale\_date, previous\_sale\_date (using lag()) and days\_between\_sales (using datediff on the previous\_sale\_date column)

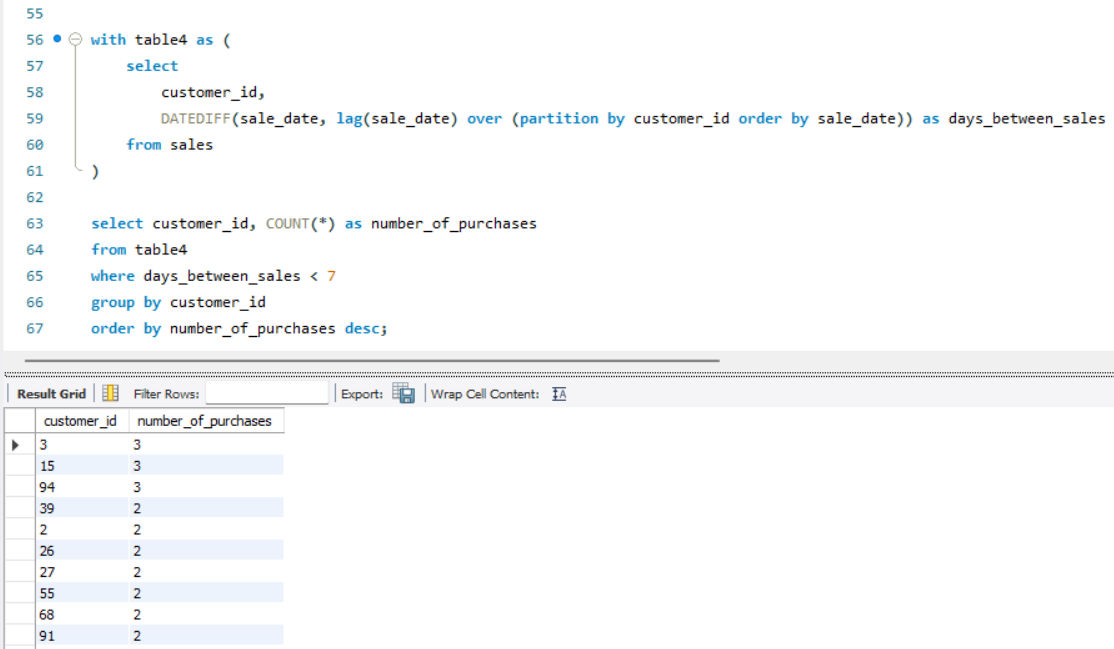
A screenshot of a computer

Description automatically generated

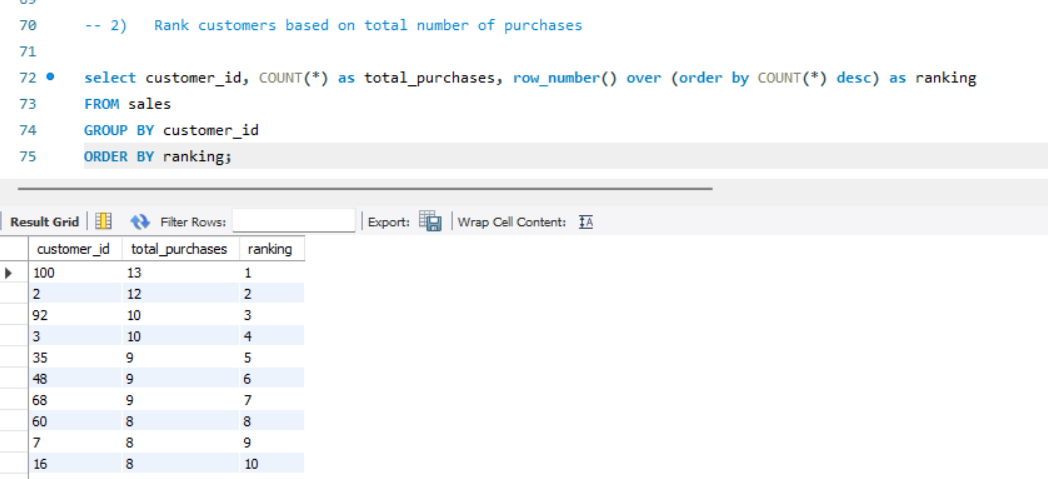
Then, we identify high frequency customers with two methods:

1. **Shortest intervals between purchases**

* We create a CTE of the earlier defined query called table4 that has customer\_id and datediff columns.
* We then call on this table, filtering using the where clause for shortest intervals being defined as a new purchase within a week so less than 7 days.
* Finally, we group by customer\_id and order by the number of purchases.



1. **Rank customers based on total number of purchases**

******

**Predictive Analytics:**

1. ***Use past data to predict which customers are most likely to churn and recommend strategies to retain them.***

**Churn:** a customer who has not made a purchase in a period of time.

**Assumptions:** Based on when was the last time the customer bought something. Customers likely to churn are set at more than 6 months ago.

We use the function: timestampdiff() to specifiy difference to be in months.

A screenshot of a computer

Description automatically generated

We get the output of 5 customers who have not purchased an item in more than 6 months.

**To retain them**, we could use a few strategies like:

* Use their personal contact information like email id and phone number to send information about new products, discount offers and sale dates.
* Reach out for customer feedback to understand why they have discontinued their purchases and try to improve performance using any negative feedback given.