Name: Taha Sameer
Student ID: 837500930
Submitted To: Prof. Udayangi Muthupoltotage

Table Of Contents

Descriptive Analysis 3

1. **Understanding Average Customer Order Value for Purchases in 2016** 3
2. **Highest Sales by Stock Group Between January to December.** 7
3. **List of All Suppliers with Total Sales Year to Date.** 11
4. **Understanding Preferred Delivery Methods for Sales and Purchasing** 14
5. **Customer with Most Diverse Purchases in 2016.** 17
6. **Details of Top 5 High-Value Customers.** 20

Exploratory Data Analysis 24

1. **Time Between Receiving an Order to Invoicing the Customer.** 24
   Key Take Away for the Business: 25
2. **List of Redundant Products In the last 2 Years.** 27
   Key Take Away for the Business: 28
3. **Analysing Sales and Quantity Growth by Continent Between 2013 to 2015** 30
   Key Take Away for the Business: 31
4. **Product Category with Highest Growth In Profit Earned 2014/2015** 32
   Key Take Away for the Business: 33

LLM Conclusive Review: 34

Overall LLM Rating: 34

Detailed Review: 34

1. **Deciding EDA questions:** 34
2. **Proficiency in Explaining Concepts:** 35
3. **Proficiency in Correcting Syntax and Debugging:** 36
4. **Proficiency in Providing Business Insights:** 36
5. **Strengths and Weaknesses:** 37

Appendix 38

Descriptive Analysis

1. Understanding Average Customer Order Value for Purchases in 2016.

Question: *What is the average order value for each customer who made purchases in 2016? Include the customer's ID, full name, and average order value (rounded to two decimal places) in the results. Order the results appropriately to quickly identify the customers with the highest average spending per order.*

To answer the question, I first list the requirements:

- Customer's ID, Full Name, Average Order Value rounded to two decimal places,
- For each customer who made a purchase in 2016,
- Order by highest average spending per order.

To understand the flow of information and visualize the columns required to solve the question, I ran exploratory queries. These helped me in understanding the relationships between the tables which were required to join the tables together.

The challenge for this question was to visualize the tables and decide which join is required. Another challenge was to figure out how to aggregate the Order Value by Order ID and then join it with the Customer table to get the average order value by customer.

I solved this problem by breaking down the question into parts. Running exploratory queries helped in visualizing the data. The queries I ran to understand the tables and the flow of information are listed below:

1. SELECT* FROM Sales.Customers

- Running this showed me that all unique customer IDs are listed in this table.

2. SELECT* FROM Sales.Orders ORDER BY OrderID

- The query lists all information from the Orders table. Arranging it by OrderID showed me that one customer can have multiple Order IDs whereas an Order ID is unique to each customer.

3. SELECT* FROM Sales.OrderLines ORDER BY OrderID

- OrderLines table joins the Orders table using the OrderID. Running this query showed me that an order is further broken down by ProductLineID. This means that 1 order can have different types of products in it.

Then I list the tables I need to solve the question, this helps me while I am writing my JOINS.

- Sales.Customers – This table will provide me with Customer ID, Full Name.
- Sales.Orders – Used to get the Order ID of each order placed.
- Sales.Orderlines – Used to get the average order value.

Final Query:

```
SELECT c.CustomerID, c.CustomerName, ROUND(AVG(OrderTotal.TotalOrderValue), 2) AS [Average Order Value]
FROM Sales.Orders o
JOIN Sales.Customers c ON o.CustomerID = c.CustomerID
JOIN (SELECT
o.OrderID, SUM(ol.Quantity * ol.UnitPrice) AS [TotalOrderValue]
FROM Sales.Orders o
JOIN Sales.Orderlines ol ON o.OrderID = ol.OrderID
WHERE YEAR(o.OrderDate) = 2016
GROUP BY o.OrderID) as OrderTotal
ON o.OrderID = OrderTotal.OrderID
GROUP BY c.CustomerID, c.CustomerName
ORDER BY [Average Order Value] DESC
```

Try 1

Since all of the customer ID's were required, I decided to use LEFT JOIN to match the customer table with the orders table.

```
SELECT c.CustomerID , c.CustomerName , COUNT(o.OrderID) AS [ORDER COUNT (CHECK)] ,
ROUND(AVG(ol.Quantity * ol.UnitPrice), 2) AS [Average Order Value]
FROM Sales.Customers c
LEFT JOIN Sales.Orders o ON c.CustomerID = o.CustomerID
JOIN Sales.OrderLines ol ON o.OrderID = ol.OrderID
WHERE o.OrderDate = '2016'
GROUP BY c.CustomerID , c.CustomerName
ORDER BY [Average Order Value] DESC
```

Try 2:

After running the above query, I realized that the results are not averaging based on the order value rather it is based on the orderlines. My next challenge was to aggregate the total order value based on the order ID and then join it to the customer ID and customer name.

```
SELECT o.OrderID, SUM(ol.Quantity * ol.UnitPrice) AS [TotalOrderValue]
FROM Sales.Orders o
JOIN Sales.Orderlines ol ON o.OrderID = ol.OrderID
WHERE YEAR (o.OrderDate) = 2016
GROUP BY o.OrderID
```

This sums order value based on Order ID from the Orders table and the Orderlines table.

Try 3:

Next challenge was to get Customer ID and Customer Name and join it with the table above to get the accurate average order value. I used the query from Try 2 in the JOIN clause to join it with Customers Table. This allowed me to pull the Customer Name and Customer ID.

Ordered it descending by the average order value gave me the customer with the highest average order in 2016.

```
SELECT c.CustomerID, c.CustomerName,
ROUND(AVG(OrderTotal.TotalOrderValue), 2) AS [Average Order Value]
FROM Sales.Orders o
JOIN Sales.Customers c ON o.CustomerID = c.CustomerID
JOIN (SELECT o.OrderID, SUM(ol.Quantity * ol.UnitPrice) AS [Total Order Value]
FROM Sales.Orders o
JOIN Sales.Orderlines ol ON o.OrderID = ol.OrderID
WHERE YEAR(o.OrderDate) = 2016
GROUP BY o.OrderID) AS OrderTotal
ON o.OrderID = OrderTotal.OrderID
GROUP BY c.CustomerID, c.CustomerName
ORDER BY [Average Order Value] DESC
```

2. Highest Sales by Stock Group Between January to December.

Question: *Which stock groups have generated the highest total sales between January 1, 2014, and December 31, 2016? Include the stock group ID, stock group name, and total sales amount in your results. Order the results suitably to identify the top-performing stock groups.*

To answer the question, I first list the requirements:

- Stock groups with highest sales,
- Between January 1, 2014, and December 31, 2016,
- Stock group ID, stock group name, and total sales amount,
- Order by top-performing groups.

I wrote SELECT statements to pull all information in that table to understand the relationship it has with other tables. This helps in identifying the foreign key and the primary key. This also helps in identifying key columns required to answer the question.

The challenge for this question was to find out the tables needed to answer this question. I used the help of Dataedo link provided by the professor to move between different tables to understand. The queries I ran to understand the tables and the flow of information are below.

1. SELECT* FROM Warehouse.StockGroups

- Key information after running this query was the total stock groups in the dataset.

2. SELECT* FROM Warehouse.StockItemStockGroups

- Wanted to see the key that links stock group table and stock item stock groups table.

3. SELECT* FROM Warehouse.StockItems

- This table links the Warehouse tables with the sales tables thus it was important to join it.

4. SELECT* FROM Sales.Invoices

- This table links with Stock Items table directly and has the sales columns required to calculate total sales. This table also has the stock item ID that is required to reach back to the stock group name.

5. SELECT* FROM Sales.InvoiceLines

- This table is required because the question asks us to filter sales between a date. The invoice date is not available in the InvoiceLines table thus it was important to join the Invoices table.

6. SELECT sg.StockGroupID , sg.StockGroupName FROM Warehouse.StockGroups sg ORDER BY 1

- I wanted to check what are the total number of Stock groups so that when I get the results I could verify my answer.

Then I start with listing the tables I need to solve the question, this allows me to keep track of them when I write my JOINS.

1. Warehouse.StockGroups

- This table has the list of Stock Group ID and their Names.

2. Warehouse.StockItemStockGroups

- This table links Stock Items with Stock Groups thus was necessary since there is no direct relationship between Stock Groups and Sales tables.

3. Warehouse.StockItems

- This table links Warehouse tables with Sales tables.

4. Sales.Invoices

- This table has the Invoice Date and is also the common table to join the Sales Lines table.

5. Sales.InvoiceLines

- This table has the sales lines which I need to calculate total sales.

Final Query:

```
SELECT sg.StockGroupID , sg.StockGroupName , ROUND(SUM(il.Quantity *il.UnitPrice),2) AS [Total Sales]

FROM Warehouse.StockGroups sg

JOIN Warehouse.StockItemStockGroups sist ON sist.StockGroupID= sg.StockGroupID

JOIN Warehouse.StockItems si ON si.StockItemID = sist.StockItemID

JOIN Sales.InvoiceLines il ON il.StockItemID = si.StockItemID

JOIN Sales.Invoices i ON i.InvoiceID = il.InvoiceID

WHERE i.InvoiceDate BETWEEN '2014-01-01' AND '2016-12-31'

GROUP BY sg.StockGroupID , sg.StockGroupName

ORDER BY [Total Sales] DESC
```

Try 1:

I joined the required tables needed but I did a mistake of using the wrong key to join the StockGroup table with the StockItemStockGroups. The right key to join these tables should be StockGroupID.

```
SELECT sg.StockGroupID , sg.StockGroupName , ROUND(SUM(il.Quantity *il.UnitPrice),2) AS [Total Sales]
FROM Warehouse.StockGroups sg
JOIN Warehouse.StockItemStockGroups sist ON sist.StockItemStockGroupID= sg.StockGroupID
JOIN Warehouse.StockItems si ON si.StockItemID = sist.StockItemID
JOIN Sales.InvoiceLines il ON il.StockItemID = si.StockItemID
JOIN Sales.Invoices i ON i.InvoiceID = il.InvoiceID
WHERE i.InvoiceDate BETWEEN '2014-01-01' AND '2016-12-31'
GROUP BY sg.StockGroupID , sg.StockGroupName
ORDER BY [Total Sales] DESC
```

Try 2:

This is the final code that I wrote to solve the answer.

```
SELECT sg.StockGroupID , sg.StockGroupName , ROUND(SUM(il.Quantity *il.UnitPrice),2) AS [Total Sales]
FROM Warehouse.StockGroups sg
JOIN Warehouse.StockItemStockGroups sist ON sist.StockGroupID= sg.StockGroupID
JOIN Warehouse.StockItems si ON si.StockItemID = sist.StockItemID
JOIN Sales.InvoiceLines il ON il.StockItemID = si.StockItemID
JOIN Sales.Invoices i ON i.InvoiceID = il.InvoiceID
WHERE i.InvoiceDate BETWEEN '2014-01-01' AND '2016-12-31'
GROUP BY sg.StockGroupID , sg.StockGroupName
ORDER BY [Total Sales] DESC
```

3. List of All Suppliers with Total Sales Year to Date.

***Question:** List all suppliers, displaying the total sales amount for their items (if any), and order the suppliers by the total sales amount in descending order, ensuring that suppliers with no sales are shown with a total sales amount of zero.*

To answer the question, I first list the requirements:

- List of all suppliers,
- Total Sales Amount (if any),
- Order by Total Sales Amount (DESC),
- Suppliers with no sales have a ZERO.

I wrote SELECT statements to pull all information in that table to understand the relationship it has with other tables. This helps in identifying the foreign key and the primary key. This also helps in identifying key columns required to answer the question.

The challenge for this question was to know which join to use and the tables required to join to get the desired information. I referred to my lecture notes to find out that to get the list of all the suppliers I need to use LEFT JOIN. To find out the tables needed to answer this question. I used the help of Dataedo link provided by the professor to move between different tables to understand which tables were needed. The queries I ran to understand the tables and the flow of information are listed below.

1. SELECT* FROM Purchasing.Suppliers

- Running this query gave me two key information. The total number of suppliers, by looking at SupplierID and the key needed to join this table with the next one.

2. SELECT* FROM Warehouse.StockItems

- This table necessary to link sales data with suppliers. StockItems table is the common table to link Purchasing schemas with Sales schemas.

3. SELECT* FROM Sales.InvoiceLines

- This table has the required information to calculate total sales.

Then I start with making a list of all the tables I need to solve the question.

1. Purchasing.Suppliers
2. Warehouse.StockItems
3. Sales.InvoiceLines

Final Query:

```
SELECT s.SupplierID , s.SupplierName , ROUND(COALESCE(SUM(il.Quantity * il.UnitPrice),0),2) AS [Total Sales]

FROM Purchasing.Suppliers s

LEFT JOIN Warehouse.StockItems st ON s.SupplierID = st.SupplierID

LEFT JOIN Sales.InvoiceLines il ON st.StockItemID = il.StockItemID

GROUP BY s.SupplierID , s.SupplierName

ORDER BY [Total Sales] DESC
```

Try 1:

This was the first attempt to get the required information. LEFT JOIN was used because we needed the list of all suppliers.

This query did not work since this did not give me those suppliers which had no sales. Since I knew that the total number of suppliers are 13, discarding the results from this code was easy.

This also made me aware that I used an INNER JOIN to connect StockItems with InvoiceLines, this will not work since I need information of all suppliers, not just of those that have sales.

```
SELECT s.SupplierID , s.SupplierName , ROUND(SUM(il.Quantity * il.UnitPrice),2) AS [Total Sales]

FROM Purchasing.Suppliers s

LEFT JOIN Warehouse.StockItems st ON s.SupplierID = st.SupplierID

JOIN Sales.InvoiceLines il ON st.StockItemID = il.StockItemID

GROUP BY s.SupplierID , s.SupplierName
```

Try 2:

I changed my INNER JOIN to LEFT JOIN and fixed my aggregate function to give me ZERO where the supplier has no sale value.

```
SELECT s.SupplierID , s.SupplierName , ROUND(COALESCE(SUM(il.Quantity * il.UnitPrice),0),2) AS [Total Sales]

FROM Purchasing.Suppliers s

LEFT JOIN Warehouse.StockItems st ON s.SupplierID = st.SupplierID

LEFT JOIN Sales.InvoiceLines il ON st.StockItemID = il.StockItemID

GROUP BY s.SupplierID , s.SupplierName

ORDER BY [Total Sales] DESC
```

4. Understanding Preferred Delivery Methods for Sales and Purchasing

Question: *List all delivery methods and usage counts in sales invoices and purchase orders. Return the delivery method ID, delivery method name, and the counts of their usage in both sales and purchasing.*

To answer this question, I first list the requirements:

- List all delivery methods,
- Usage counts in sales invoices and purchase orders,
- Return the delivery method ID, delivery method name.

I wrote SELECT statements to pull all information in that table to understand the relationship it has with other tables. This helps in identifying the foreign key and the primary key. This also helps in identifying key columns required to answer the question.

The challenge for this question was to know which join to use. Reading the list of requirements helped me in figuring out that I will need LEFT JOIN to solve it. To find out the tables needed to answer this question. I used the help of Dataedo provided by the professor. The queries I ran to understand the tables and the flow of the information are below:

1. SELECT* FROM Application.DeliveryMethods
- I wanted to know what the total number of delivery methods are.
2. SELECT* FROM Purchasing.PurchaseOrders
- This table contains information for the preferred delivery method for purchasing.
3. SELECT* FROM Sales.Invoices
- This contains delivery methods for sales.

Then I start solving the question by listing the tables I need.

1. Application.Deliverymethods
2. Sales.Invoices
3. Purchasing.PurchaseOrders.

Final Query:

```
SELECT dm.DeliveryMethodID , dm.DeliveryMethodName , COUNT(i.DeliveryMethodID) AS [Delivery Method Count - Sales], COUNT(po.DeliveryMethodID) AS [Delivery Method Count - Purchases]

FROM Application.DeliveryMethods dm

LEFT JOIN Sales.Invoices i ON dm.DeliveryMethodID = i.DeliveryMethodID

LEFT JOIN Purchasing.PurchaseOrders po ON dm.DeliveryMethodID = po.DeliveryMethodID

GROUP BY dm.DeliveryMethodID , dm.DeliveryMethodName
```

Try 1:

This query counts the number of times a particular delivery method is used in the Sales table. It took out all NULL values. The query only gave 1 result.

```
SELECT dm.DeliveryMethodID , dm.DeliveryMethodName , COUNT(i.DeliveryMethodID) AS [Delivery Method Count - Sales]

FROM Application.DeliveryMethods dm

LEFT JOIN Sales.Invoices i ON dm.DeliveryMethodID = i.DeliveryMethodID

WHERE i.DeliveryMethodID IS NOT NULL

GROUP BY dm.DeliveryMethodID , dm.DeliveryMethodName
```

I wanted to check whether results from the above query was correct or not thus I wrote the following query. It lists the unique delivery methods used for Sales.Invoices.

```
SELECT DISTINCT DeliveryMethodID

FROM Sales.Invoices
```

The query below lists the number of unique delivery methods used for the purchase order table. This helped me in combining the two insights and counter checking my answer.

```
SELECT DISTINCT DeliveryMethodID

FROM Purchasing.PurchaseOrders
```

Try 2:

This provided me with 2 separate columns where one was counting the unique delivery method ID in sales table and the other was counting from the Purchases table. LEFT JOIN was used because I wanted the complete list of delivery method ID's. LEFT JOIN was also used because I wanted to see which delivery method is not being used in each of the table.

```
SELECT dm.DeliveryMethodID , dm.DeliveryMethodName , COUNT(i.DeliveryMethodID) AS [Delivery Method Count - Sales], COUNT(po.DeliveryMethodID) AS [Delivery Method Count - Purchases]

FROM Application.DeliveryMethods dm

LEFT JOIN Sales.Invoices i ON dm.DeliveryMethodID = i.DeliveryMethodID

LEFT JOIN Purchasing.PurchaseOrders po ON dm.DeliveryMethodID = po.DeliveryMethodID

GROUP BY dm.DeliveryMethodID , dm.DeliveryMethodName
```

5. Customer with Most Diverse Purchases in 2016.

Question: *Identify which customers purchased the most diverse range of products in 2016, and the total amount they spent. Include the number of unique products each customer has bought, and the total amount spent in the results to demonstrate the diversity of products. Order and filter the result set in a suitable manner to find the top 10 high-value customers.*

To answer this question, I first list the requirements:

1. Identify which customers,
2. Purchased the most diverse range of products in 2016,
3. Total amount they spent.

I wrote SELECT statements to pull all information in that table to understand the relationship it has with other tables. This helps in identifying the foreign key and the primary key. This also helps in identifying key columns required to answer the question.

The challenge for this question was to figure out how to count the number of unique products. Defining 'diverse products' was a challenge. To resolve this challenge, I tried to understand the relationship between the sales tables. Looked at customer buying pattern from the Invoices table. Running different queries to get an idea of the flow of information and to find out the tables needed to answer this question. I used the help of Dataedo link provided by the professor to move between different tables to understand which tables were needed. The queries I ran to understand the tables and the flow of the information are below:

1. SELECT* FROM Sales.invoiceLines

- This table had the StockItemID, which is a unique identifier for the type of product ordered. This gave me an idea to count the unique IDs by customer to see what is the count of diverse product range ordered by each customer. This table also has the quantity ordered and the unit price which would give me the total sales value.

2. SELECT* FROM Sales.invoices

- This table had the customer information that was required to join with the InvoiceLines table to arrange the diverse product range by CustomerID.

Then I start solving the question by listing the tables I need.

1. Sales.InvoiceLines
2. Sales.Invoices

Final Query:

```
SELECT TOP 10 i.CustomerID , COUNT(DISTINCT(il.StockItemID)) AS [Product Count] , SUM(il.Quantity * il.UnitPrice) AS [Total Sales]
FROM Sales.InvoiceLines il
JOIN Sales.Invoices i ON il.InvoiceID = i.InvoiceID
WHERE YEAR(i.invoiceDate) = 2016
GROUP BY i.CustomerID
ORDER BY 2 DESC
```

Try 1:

The following query was run to find out the number of orders placed by each customer in 2016.

```
SELECT i.CustomerID , COUNT(DISTINCT(il.StockItemID)) AS [Number of Orders]
FROM Sales.InvoiceLines il
JOIN Sales.Invoices i ON il.InvoiceID = i.InvoiceID
WHERE YEAR(i.invoiceDate) = 2016
GROUP BY i.CustomerID
ORDER BY [Number of Orders] DESC
```

Try 2:

The following query is like the above one with the addition of Total Sales column. This does not provide with the top 10 customers.

```
SELECT i.CustomerID , COUNT(DISTINCT(StockItemID)) AS [Product Count] , SUM(il.Quantity * il.UnitPrice) AS [Total Sales]
FROM Sales.InvoiceLines il
JOIN Sales.Invoices i ON il.InvoiceID = i.InvoiceID
WHERE YEAR(i.invoiceDate) = 2016
GROUP BY i.CustomerID
ORDER BY 2 DESC
```

TRY 3:

This is the final query. Top 10 customers who have ordered unique products and their total sales in 2016.

```
SELECT TOP 10 i.CustomerID , COUNT(DISTINCT(il.StockItemID)) AS [Product Count] , SUM(il.Quantity * il.UnitPrice) AS [Total Sales]

FROM Sales.InvoiceLines il

JOIN Sales.Invoices i ON il.InvoiceID = i.InvoiceID

WHERE YEAR(i.invoiceDate) = 2016

GROUP BY i.CustomerID

ORDER BY 2 DESC
```

6. Details of Top 5 High-Value Customers.

Question: *Modify your query from question 5) above to display the details of these purchases for the top 5 high-value customers. Include in your results the customer's ID and full name, the product IDs and names, the number of orders for each product, the total quantity ordered, and the total amount spent on each product.*

To answer this question, I first list the requirements:

4. Display the details of these purchases for the top 5 high-value customers,
5. Customer's ID and full name, the product IDs and names, number of orders for each product,
6. Total quantity ordered, and the total amount spent on each product.

The challenge for this question was to figure out where the sub query will fit in. Another challenge was to code it properly to get the right results. To solve this problem, I tried breaking down my question into parts and then tried solving it. This helped as it provided me with more clarity, and I was sure of the steps I was taking to answer the question.

Running different queries to get an idea of the flow of information and to find out the tables needed to answer this question. I used the help of Dataedo link provided by the professor to move between different tables to understand which tables were needed. The queries I ran to understand the tables and the flow of the information are below:

1. Sales.Invoices
2. Sales.InvoiceLines
3. Sales.Customers
4. Warehouse.StockItems

Next, I brought my query from question 5 and tried to understand what more is required from this question and how can I modify it for the results required.

Final Query:

```
SELECT tc.CustomerID, tc.CustomerName,si.StockItemID,si.StockItemName,COUNT(il.InvoiceID) AS
NumberOfOrders,SUM(il.Quantity) AS TotalQuantityOrdered,ROUND(SUM(il.Quantity * il.UnitPrice), 2) AS
TotalAmountSpent

FROM (SELECT TOP 5

c.CustomerID, c.CustomerName, COUNT(DISTINCT il.StockItemID) AS UniqueProductsBought,

ROUND(SUM(il.Quantity * il.UnitPrice), 2) AS TotalAmountSpent

FROM Sales.Invoices i

JOIN Sales.InvoiceLines il ON i.InvoiceID = il.InvoiceID

JOIN Sales.Customers c ON i.CustomerID = c.CustomerID

JOIN Warehouse.StockItems si ON il.StockItemID = si.StockItemID

WHERE YEAR(i.InvoiceDate) = 2016

GROUP BY c.CustomerID, c.CustomerName

ORDER BY TotalAmountSpent DESC) AS tc

JOIN Sales.Invoices i ON tc.CustomerID = i.CustomerID

JOIN Sales.InvoiceLines il ON i.InvoiceID = il.InvoiceID
JOIN Warehouse.StockItems si ON il.StockItemID = si.StockItemID

WHERE YEAR(i.InvoiceDate) = 2016

GROUP BY tc.CustomerID, tc.CustomerName, si.StockItemID, si.StockItemName

ORDER BY tc.CustomerName ASC, TotalAmountSpent DESC
```

Try 1:

This is just bringing the above query below and listing the new requirements for the question.

This query will work as a virtual table for the top 5 customers who spent the most in 2016. This is based on their unique purchasing pattern.

```
SELECT TOP 5 c.CustomerID, c.CustomerName, COUNT(DISTINCT il.StockItemID) AS
UniqueProductsBought, ROUND(SUM(il.Quantity * il.UnitPrice), 2) AS TotalAmountSpent

FROM Sales.Invoices i

JOIN Sales.InvoiceLines il ON i.InvoiceID = il.InvoiceID

JOIN Sales.Customers c ON i.CustomerID = c.CustomerID

JOIN Warehouse.StockItems si ON il.StockItemID = si.StockItemID

WHERE YEAR(i.InvoiceDate) = 2016

GROUP BY c.CustomerID, c.CustomerName

ORDER BY TotalAmountSpent DESC
```

Try 2:

Next, I had to JOIN my virtual table with Invoices table to pull in their invoices and join the Invoices table with InvoiceLines to pull in individual product information and lastly joining the StockItems table to pull in each product details like name of the product.

```
SELECT
tc.CustomerID, tc.CustomerName, si.StockItemID, si.StockItemName, COUNT(il.InvoiceID) AS
NumberOfOrders,SUM(il.Quantity) AS TotalQuantityOrdered, ROUND(SUM(il.Quantity * il.UnitPrice), 2) AS
TotalAmountSpent
FROM (SELECT TOP 5
c.CustomerID, c.CustomerName,
COUNT(DISTINCT il.StockItemID) AS UniqueProductsBought,
ROUND(SUM(il.Quantity * il.UnitPrice), 2) AS TotalAmountSpent
FROM Sales.Invoices i
JOIN Sales.InvoiceLines il ON i.InvoiceID = il.InvoiceID
JOIN Sales.Customers c ON i.CustomerID = c.CustomerID
JOIN Warehouse.StockItems si ON il.StockItemID = si.StockItemID
WHERE YEAR(i.InvoiceDate) = 2016
GROUP BY c.CustomerID, c.CustomerName
ORDER BY TotalAmountSpent DESC) AS tc
JOIN Sales.Invoices i ON tc.CustomerID = i.CustomerID
JOIN Sales.InvoiceLines il ON i.InvoiceID = il.InvoiceID
JOIN Warehouse.StockItems si ON il.StockItemID = si.StockItemID
WHERE YEAR(i.InvoiceDate) = 2016
GROUP BY tc.CustomerID, tc.CustomerName, si.StockItemID, si.StockItemName
ORDER BY tc.CustomerName ASC, TotalAmountSpent DESC
```

Exploratory Data Analysis

For my exploratory analysis, I choose to analyze the product domain. Since Worldwide Importers is a trading company, it is important for them to handle their product portfolio efficiently to meet demands. It will also help them understand fast moving products so that they could stay updated with current products and invest in new ones to maintain revenue.

1. Time Between Receiving an Order to Invoicing the Customer.

Question: *What orders take more than 2 days from the date when the order was received to the date when it was invoiced to the customer.*

For my first exploratory analysis, I wanted to understand how long it takes for the company to invoice the customer after receiving an order. This will help in analyzing whether there is a trend to a certain type of products that takes too long to be invoiced. This is an important metric to look at since the company is not a manufacturing company, thus fulfilling demands quickly is important.

Query required to pull in all necessary information is:

```
SELECT i.OrderID, ol.Description, ol.Quantity, i.InvoiceDate , o.OrderDate ,
DATEDIFF (DAY , O.OrderDate, i.InvoiceDate) AS [Days From receiving an Order to Invoicing]
FROM Sales.InvoiceLines il
JOIN Sales.Invoices i ON il.InvoiceID = i.InvoiceID
JOIN Sales.Orders o ON i.OrderID = o.OrderID
JOIN Sales.OrderLines ol ON o.OrderID = ol.OrderID
JOIN Warehouse.StockItems si ON il.StockItemID = si.StockItemID
WHERE DATEDIFF (DAY , O.OrderDate, i.InvoiceDate) > 2 AND YEAR(o.OrderDate) = 2015
GROUP BY i.InvoiceDate , o.OrderDate ,i.OrderID , ol.Description ,ol.Quantity
ORDER BY [Days From receiving an Order to Invoicing] DESC ;
```

Table 1 shows that products which take more than a year to be invoiced are fringe sizes. Such products are uncommon sizes (5XL ,4XL). Secondly, looking at description and quantity, there is some similarity. Such sizes are in demand which is evident since an order is placed almost every month. Another key information derived from this is a possibility of a bottle neck in the supply chain. Since the time difference averages around 400 days, there might be a problem from the supplier side in fulfilling the order.

Key Take Away for the Business:

The company needs to evaluate its cash flow cycle. High cash conversion days might lead to poor cash management which might cause liquidity problem. Either acquiring short-term credit line from the bank or partnering with suppliers who deal with fringe products exclusively would significantly improve the cash conversion cycle of the company. For high demand fringe products, the company might look at keeping some stock at hand, this would reduce order-to-invoice time significantly and allow better inventory turnover.

Table 1: List of products that have the difference between order date and Invoice date more than 2 days. Note: These are partial results from the full results set.

	Order ID	Description	Quantity	Invoice Date	Order Date	Days From Receiving an Order to Invoicing
1	42924	The Gu red shirt XML tag t-shirt (White) 5XL	72	28/04/2016	09/02/2015	444
2	44874	The Gu red shirt XML tag t-shirt (White) 5XL	72	23/05/2016	11/03/2015	439
3	41906	The Gu red shirt XML tag t-shirt (White) 5XL	72	29/03/2016	22/01/2015	432
4	42639	The Gu red shirt XML tag t-shirt (White) 5XL	72	04/04/2016	03/02/2015	426
5	46729	The Gu red shirt XML tag t-shirt (Black) 4XL	72	23/05/2016	09/04/2015	410

6	47279	Shipping carton (Brown) 305x305x305mm	175	30/05/2016	16/04/2015	410
7	43779	The Gu red shirt XML tag t-shirt (White) 5XL	60	07/04/2016	23/02/2015	409
8	46471	The Gu red shirt XML tag t-shirt (Black) 4XL	72	19/05/2016	06/04/2015	409
9	47354	The Gu red shirt XML tag t-shirt (Black) 4XL	72	30/05/2016	18/04/2015	408
10	47075	Shipping carton (Brown) 305x305x305mm	175	26/05/2016	14/04/2015	408
11	43368	The Gu red shirt XML tag t-shirt (White) 5XL	60	21/03/2016	16/02/2015	399
12	45082	The Gu red shirt XML tag t-shirt (White) 5XL	60	13/04/2016	14/03/2015	396
13	47176	The Gu red shirt XML tag t-shirt (Black) 4XL	60	12/05/2016	15/04/2015	393
14	47425	The Gu red shirt XML tag t-shirt (Black) 4XL	60	16/05/2016	20/04/2015	392
15	46724	Shipping carton (Brown) 305x305x305mm	175	02/05/2016	09/04/2015	389
16	46957	The Gu red shirt XML tag t-shirt (White) XS	72	05/05/2016	13/04/2015	388
17	45475	The Gu red shirt XML tag t-shirt (White) XS	72	11/04/2016	21/03/2015	387
18	45763	The Gu red shirt XML tag t-shirt (White) XS	72	14/04/2016	25/03/2015	386
19	47705	The Gu red shirt XML tag t-shirt (White) XS	72	09/05/2016	23/04/2015	382
20	44958	The Gu red shirt XML tag t-shirt (White) XS	72	25/03/2016	12/03/2015	379

2. List of Redundant Products In the last 2 Years.

Question: *List products that have not been ordered in the past 2 years.*

My second question analyses redundant product list in the past 2 years. This will allow me to understand products that are no longer in demand. This is key information since the company would not want to purchase or hold stock that has the potential of becoming redundant in the future.

Query required to pull in all necessary information is:

```
SELECT q.StockItemID , q.StockItemName , [QuantityOrderedin2014] ,[QuantityOrderedin2015]
FROM(SELECTpol.StockItemID, si.StockItemName,
SUM(CASE WHEN po.OrderDate BETWEEN '2013-01-01' AND '2013-12-31' THEN pol.OrderedOuters ELSE
0 END) AS [QuantityOrderedin2013],
SUM(CASE WHEN po.OrderDate BETWEEN '2014-01-01' AND '2014-12-31' THEN pol.OrderedOuters ELSE
0 END) AS [QuantityOrderedin2014],
SUM(CASE WHEN po.OrderDate BETWEEN '2015-01-01' AND '2015-12-31' THEN pol.OrderedOuters ELSE
0 END) AS [QuantityOrderedin2015]
FROM Purchasing.PurchaseOrderLines pol
JOIN Purchasing.PurchaseOrders po ON pol.PurchaseOrderID = po.PurchaseOrderID
JOIN Warehouse.StockItems si ON pol.StockItemID = si.StockItemID
GROUP BY pol.StockItemID , si.StockItemName ) AS [q]
WHERE q.QuantityOrderedin2014 = 0 OR q.QuantityOrderedin2015 = 0
ORDER BY q.StockItemID
```

Table 2 shows a partial snapshot of the list of all redundant products. There are a total of 218 products that have never been ordered since 2014. The list indicates that products are from a wide range of categories indicating a shift in consumer choice.

Key Take Away for the Business:

This is a key insight for the company as this will help in adding different products to their portfolio. This initiates the process to look for newer suppliers that will help the company to adapt to this shift. Most products that have become redundant have some text written on it, this could mean that such products were in demand for that particular year, thus the company needs to research for current trends so that in demand products could be added. This will help in better resource allocation and increased revenue.

Researching current market trends and looking at past redundancy list will help the company in predicting future products that might not be in demand. This will give them a head start in acquiring new suppliers that could help in the shift. This will also lead to improved inventory handling as company will not allocate resources where it predicts that the product will become redundant.

Table 2: List of products not ordered in 2014 and 2015. Note: These are partial results from the full results set.

Stock Item ID	Stock Item Name	Quantity Ordered in 2014	Quantity Ordered in 2015
1	USB missile launcher (Green)	0	0
2	USB rocket launcher (Gray)	0	0
3	Office cube periscope (Black)	0	0
4	USB food flash drive - sushi roll	0	0
5	USB food flash drive - hamburger	0	0
6	USB food flash drive - hot dog	0	0
7	USB food flash drive - pizza slice	0	0
8	USB food flash drive - dim sum 10 drive variety pack	0	0
9	USB food flash drive - banana	0	0
10	USB food flash drive - chocolate bar	0	0
11	USB food flash drive - cookie	0	0
12	USB food flash drive - donut	0	0

13	USB food flash drive - shrimp cocktail	0	0
14	USB food flash drive - fortune cookie	0	0
15	USB food flash drive - dessert 10 drive variety pack	0	0
16	DBA joke mug - mind if I join you? (White)	0	0
17	DBA joke mug - mind if I join you? (Black)	0	0
18	DBA joke mug - daaaaaa-ta (White)	0	0
19	DBA joke mug - daaaaaa-ta (Black)	0	0
20	DBA joke mug - you might be a DBA if (White)	0	0

3. Analysing Sales and Quantity Growth by Continent Between 2013 to 2015.

***Question:** How have sales and quantity sold changed between 2013 to 2015 by continent.*

My third question investigates change in quantity sold and sales volume between 2013 to 2015 by continent. This will show me which region is most popular with our products and help in analysing a continent in isolation in terms to product quantity and sales volumes.

Query required to pull in all necessary information is:

```
SELECT c.Continent ,
(SUM(CASE WHEN i.InvoiceDate BETWEEN '2014-01-01' AND '2014-12-31'THEN il.Quantity ELSE 0 END)
-
SUM(CASE WHEN i.InvoiceDate BETWEEN '2013-01-01' AND '2013-12-31' THEN il.Quantity ELSE 0 END))
AS [Change in Quantity Sold 2013/2014] ,
(SUM(CASE WHEN i.InvoiceDate BETWEEN '2015-01-01' AND '2015-12-31' THEN il.Quantity ELSE 0 END)
-
SUM(CASE WHEN i.InvoiceDate BETWEEN '2014-01-01' AND '2014-12-31' THEN il.Quantity ELSE 0 END))
AS [Change in Quantity Sold 2014/2015],
(SUM(CASE WHEN i.InvoiceDate BETWEEN '2014-01-01' AND '2014-12-31' THEN (il.Quantity *
il.UnitPrice) ELSE 0 END) -
SUM(CASE WHEN i.InvoiceDate BETWEEN '2013-01-01' AND '2013-12-31' THEN (il.Quantity * il.UnitPrice)
ELSE 0 END)) AS [Change in Sales 2013/2014],
(SUM(CASE WHEN i.InvoiceDate BETWEEN '2015-01-01' AND '2015-12-31' THEN (il.Quantity *
il.UnitPrice) ELSE 0 END) -
SUM(CASE WHEN i.InvoiceDate BETWEEN '2014-01-01' AND '2014-12-31' THEN (il.Quantity * il.UnitPrice)
ELSE 0 END)) AS [Change in Sales 2014/2015]
FROM Application.Countries c
LEFT JOIN Application.People p ON c.LastEditedBy = p.PersonID
JOIN Application.DeliveryMethods dm ON p.PersonID = dm.LastEditedBy
JOIN Sales.Customers sc On dm.DeliveryMethodID = sc.DeliveryMethodID
JOIN Sales.Invoices i ON sc.CustomerID = i.CustomerID
JOIN Sales.InvoiceLines il ON i.InvoiceID = il.InvoiceID
GROUP BY c.Continent
```

Table 3 shows that there was an increase in quantity sold year on year, but when we compare it with the change in sales, it shows that our sales declined. This means that the volume of low profit earning products increased compared to last year.

Key Take Away for the Business:

This key insight will help the company to promote high-profit margin-earning products to these regions. This will also help in budgeting for next year based on historical data. The company can also consider increasing the sales price of high-demand products in each region to boost revenue. There could be a change in product preference; thus, this insight can initiate research work for the company, which would help in acquiring new products. There could also be a supply chain problem within the company. The company could start reevaluating their operations to see why sales decreased when product quantity increased.

Table 3: Change in Quantity sold and Sales Volume between 2013 and 2015

Continent	Change in Quantity Sold 2013-2014	Change in Quantity Sold 2014-2015	Change in Sales 2013-2014	Change in Sales 2014-2015
Africa	7,126,992	7,433,195	181,558,866	174,666,140
Asia	6,795,504	7,087,465	173,114,267	166,542,133
Europe	5,138,064	5,358,815	130,891,275	125,922,101
North America	2,983,392	3,111,570	76,001,386	73,116,059
Oceania	1,325,952	1,382,920	33,778,394	32,496,026
Seven seas (open ocean)	497,232	518,595	12,666,898	12,186,010
South America	1,823,184	1,901,515	46,445,291	44,682,036

4. Product Category with Highest Growth In Profit Earned 2014/2015

Question: *Which product category had the highest increase in profit earned in between 2014 and 2015?*

This question analyses profit growth between 2014 and 2015 by product category. This will show us how each product category performed compared to last year. This can further help in comparing it with budgeted targets by the finance team whereas quantity sold can be compared with targets set for the sales team.

Query required to pull in all necessary information is:

```
SELECT sg.StockGroupName ,
(AVG (CASE WHEN YEAR(i.InvoiceDate) = 2014 THEN il.UnitPrice ELSE 0 END) - (AVG (CASE WHEN
YEAR(i.InvoiceDate) = 2014 THEN sih.LastCostPrice ELSE 0 END))) AS [Profit Margin 2014],
(AVG (CASE WHEN YEAR(i.InvoiceDate) = 2015 THEN il.UnitPrice ELSE 0 END) - (AVG (CASE WHEN
YEAR(i.InvoiceDate) = 2015 THEN sih.LastCostPrice ELSE 0 END))) AS [Profit Margin 2015],
SUM (CASE WHEN YEAR(i.InvoiceDate) = 2014 THEN il.Quantity ELSE 0 END) AS [Total Quantity Sold -
2014] ,
SUM (CASE WHEN YEAR(i.InvoiceDate) = 2015 THEN il.Quantity ELSE 0 END) AS [Total Quantity Sold -
2015] ,
((AVG (CASE WHEN YEAR(i.InvoiceDate) = 2014 THEN il.UnitPrice ELSE 0 END) - (AVG (CASE WHEN
YEAR(i.InvoiceDate) = 2014 THEN sih.LastCostPrice ELSE 0 END)))
* (SUM (CASE WHEN YEAR(i.InvoiceDate) = 2014 THEN il.Quantity ELSE 0 END))) AS [Profit Earned
2014] ,
((AVG (CASE WHEN YEAR(i.InvoiceDate) = 2015 THEN il.UnitPrice ELSE 0 END) - (AVG (CASE WHEN
YEAR(i.InvoiceDate) = 2015 THEN sih.LastCostPrice ELSE 0 END)))
* (SUM (CASE WHEN YEAR(i.InvoiceDate) = 2015 THEN il.Quantity ELSE 0 END))) AS [Profit Earned
2015],
((((AVG (CASE WHEN YEAR(i.InvoiceDate) = 2015 THEN il.UnitPrice ELSE 0 END) - (AVG (CASE WHEN
YEAR(i.InvoiceDate) = 2015 THEN sih.LastCostPrice ELSE 0 END)))
* (SUM (CASE WHEN YEAR(i.InvoiceDate) = 2015 THEN il.Quantity ELSE 0 END))) - ((AVG (CASE
WHEN YEAR(i.InvoiceDate) = 2014 THEN il.UnitPrice ELSE 0 END) - (AVG (CASE WHEN
YEAR(i.InvoiceDate) = 2014 THEN sih.LastCostPrice ELSE 0 END)))
* (SUM (CASE WHEN YEAR(i.InvoiceDate) = 2014 THEN il.Quantity ELSE 0 END)))) / ((AVG (CASE
WHEN YEAR(i.InvoiceDate) = 2014 THEN il.UnitPrice ELSE 0 END) - (AVG (CASE WHEN
YEAR(i.InvoiceDate) = 2014 THEN sih.LastCostPrice ELSE 0 END)))
* (SUM (CASE WHEN YEAR(i.InvoiceDate) = 2014 THEN il.Quantity ELSE 0 END)))) * 100 AS [Percentage
Growth]
FROM Warehouse.StockItemHoldings sih
JOIN Warehouse.StockItems si ON sih.StockItemID = si.StockItemID
JOIN Warehouse.StockItemStockGroups sisg ON si.StockItemID = sisg.StockItemID
JOIN Warehouse.StockGroups  sg ON sisg.StockGroupID = sg.StockGroupID
JOIN Sales.InvoiceLines il ON si.StockItemID = il.StockItemID
JOIN Sales.Invoices i ON il.InvoiceID = i.InvoiceID
GROUP BY StockGroupName
ORDER BY [Percentage Growth] DESC
```

Insights reveal that Furry Footwear had the highest increase in profit margin, 24% whereas T-Shirts experienced the least amount of growth, 9%. Quantity sold for Packaging Material saw an increase of 115,000 from last year but the increase in profit earned only 17%. USB Novelties and Toys saw an increase of just 2000 products but growth in profit earned was 21% each.

Key Take Away for the Business:

This information will help the company push products that contribute higher to their profit margin even when change in quantity sold is low. Such groups are USB Novelties and Toys. With just 2,000 more products sold than last year their profit growth was 21% and 20% respectively.

Packaging Material has over 14 times more sales than Furry Footwear thus it is important to check how sensitive consumers are to the sales price, if they are less sensitive then its increasing sales price would significantly improve profit earned.

Table 4: Average Profit Margins, Total Quantity Sold and Profit Earned Between 2014 and 2015.

Stock Group Name	Profit Margin 2014	Profit Margin 2015	Total Quantity Sold 2014	Total Quantity Sold 2015	Profit Earned 2014	Profit Earned 2015	Percentage Growth
Furry Footwear	5.92	6.59	112,845	125,924	668,108	830,212	24
Mugs	2.45	2.71	70,384	77,268	172,365	209,364	21
USB Novelties	11.06	12.13	23,685	26,048	261,924	316,059	21
Toys	14.88	16.45	35,403	38,303	526,801	630,181	20
Novelty Items	6.72	7.42	306,077	328,677	2,056,133	2,439,706	19
Packaging Materials	8.56	9.36	1,649,422	1,765,233	14,122,986	16,529,721	17
Clothing	3.75	4.10	771,357	817,394	2,891,586	3,353,079	16
Computing Novelties	4.13	4.50	579,099	605,696	2,393,655	2,723,128	14
T-Shirts	3.10	3.24	467,880	486,360	1,452,562	1,577,839	9

LLM Conclusive Review:

Overall LLM Rating:

Overall LLM helped in solving the assignment but with some tasks it took me more time to fix what LLM provided. This delayed my assignment as it took me off-topic. To be able to use LLM efficiently one needs to know the dataset beforehand, it improves the quality of questions you ask and thus can perform faster debugging of the syntax provided by the LLM.

LLM was useful in understanding certain technical concepts. LLM defines a technical element of SQL efficiently which improved my technical knowledge and allowed me to write proper syntax.

LLM was useful in writing business insights once I provided it with the results.

Table 1: *Overall Summary of LLM (ChatGPT):*

Criteria	Rating (Excellent, Average, Poor)
Help in Deciding EDA Questions	Average
Proficiency in Explaining Concepts	Excellent
Proficiency in Correcting Syntax and Debugging	Excellent
Proficiency in Providing Business Insights	Excellent

Detailed Review:

Deciding EDA questions:

LLM was first used to check whether it knows the Worldwide Importers data set. I started with basic questions. **Figure 1** shows screenshots of the conversation.

The database diagram was overwhelming when I loaded it in SSMS thus I wanted LLM to give me a quick glance of it. LLM gave me a very detailed explanation of the dataset as shown in **Figure 2 & 3**, which was helpful in getting an overall understanding of the various tables.

Since it knew what the various tables the dataset had, I asked it to produce some exploratory questions. I provided it with all the types of syntaxes I knew so that it does not provide me with queries that might be too complex. **Figure 4** shows the prompt generated. The generated results did not have much business context thus I refined my question by providing it with examples of potential questions and then asked to produce more questions based on the provided examples. **Figure 5** shows the prompt generated.

LLM was creating dummy tables that were not in the data set, which made it difficult to use the questions for my analysis. Tables like "Return Reasons" and columns like "Return Reason ID" were not part of the dataset, but LLM produced syntax incorporating such dummy tables/columns. **Figure 6** shows one of the questions generated and its syntax. Here you can see the dummy tables produced by it.

Since we were asked to focus on either Customer or Product domain, LLM was asked to produce questions that were targeted to these two domains. **Figure 7** shows the prompt. It was able to produce some good results. The concern with these questions was that they were focusing too much on a particular product. Since the requirement of the assignment was to do a broader exploration which would provide key insights for the business to act on, these generated questions were disregarded.

Reading multiple questions did trigger my thinking of how these questions can be modified to pull key insights that would be meaningful to the business.

Proficiency in Explaining Concepts:

LLM was more useful in explaining concepts than suggesting questions. After deciding my EDA questions, when I started solving them, I came across multiple questions that LLM helped me understand.

Figure 8 shows how I asked LLM to explain whether the order of JOIN matters.

Figure 9 shows the conversation where I asked LLM how to write a SELECT Subquery.

After trying how to write a subquery, I came across a problem where I could not run the subquery in isolation, even though I knew that it should run on its own, I took the help of LLM to clear my concepts. **Figure 10** shows results of the conversation.

LLM is also good at providing you with syntax when you ask it to work with dummy dataset. **Figure 11** shows the results when I asked LLM on show me a code based on a theory I had in mind.

LLM’s deep knowledge of SQL helped me write queries and helped me solve questions better. Its quick response coupled with examples made it clear on how to code the type of error I might be doing while coding.

Proficiency in Correcting Syntax and Debugging:

When provided with a written query, LLM is good at finding out your mistakes and giving you the correct syntax.

Figure 12 shows the first instance where I asked LLM to correct my syntax. It first reads through your code and provides you with the correct syntax. **Figure 13** is the continuation of **Figure 12** where LLM breaks it down and pinpoints your mistake and provides you with the explanation.

LLM is proficient even if you just give it the code and ask it to debug. **Figure 14** shows one of the results. It does not matter how complex or long the query is, **Figure 15** shows LLM’s ability to debug a long and complex query.

Proficiency in Providing Business Insights:

When asked to evaluate a certain problem, LLM provides with a wide range of possible answered. I presented my findings to LLM and asked business context of it. It provided me with lots of scenarios from which I could choose and implement on based on my scenario.

Figure 16 shows the prompt where I asked LLM to provide me with possible reasons why a company would want to explore its product side.

I used LLM’s help while providing business insights for my EDA questions. I found that LLM was better at providing insights than any other part of this assignment. **Figure 17** conversation where I asked LLM to provide me possible reasons why a business would want to look at their redundant product list.

I found that results were better when you define your conditions in detail. LLM provided with specific results when I told it that the company is not a manufacturing company rather a trading company. **Figure 18** shows the conversation. You can see the difference in the recommendations if you compare it with **Figure 17**.

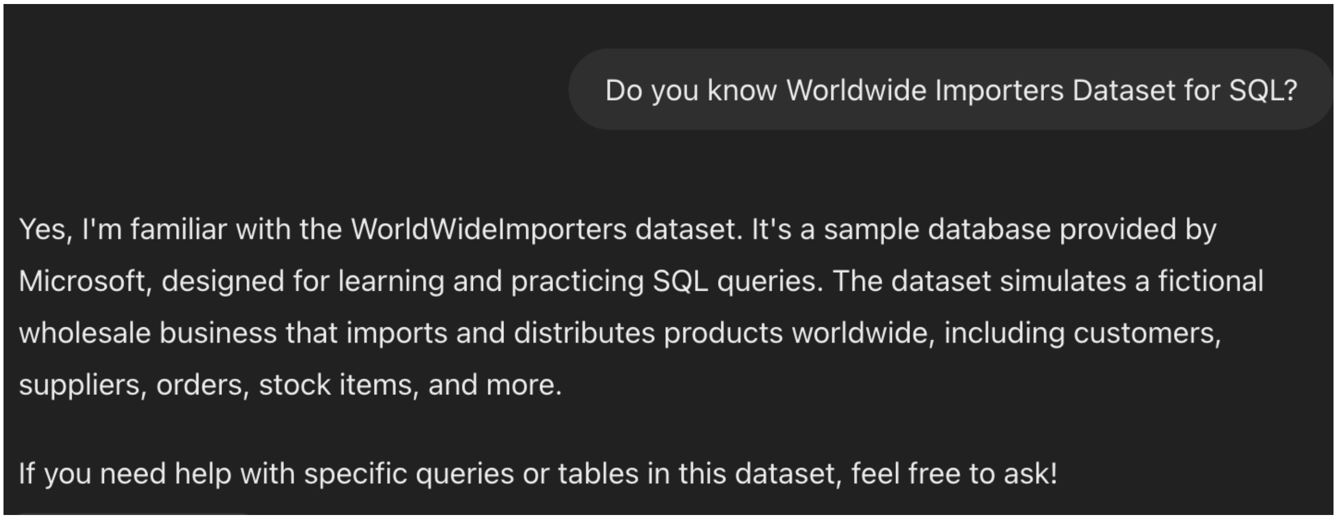
Strengths and Weaknesses:

Following are key strengths and weaknesses of LLM that I experienced while doing my assignment.

Strengths	Weaknesses
1. It has knowledge about the dataset.	1. It will create dummy tables to answer a question.
2. It can explain concepts clearly & with examples.	2. It might not fully understand your command
3. It can debug syntax efficiently.	3. It will join tables that might not exist in the data set.
4. It is extremely helpful in providing business context.	4. Business theories are at times to generic.
	5. It takes time to verify whether a syntax provided is accurate or not.
	6. It cannot write proper queries if you provide it with a question.

Appendix

Figure 1: Asking LLM whether it knows about Worldwide Importers:

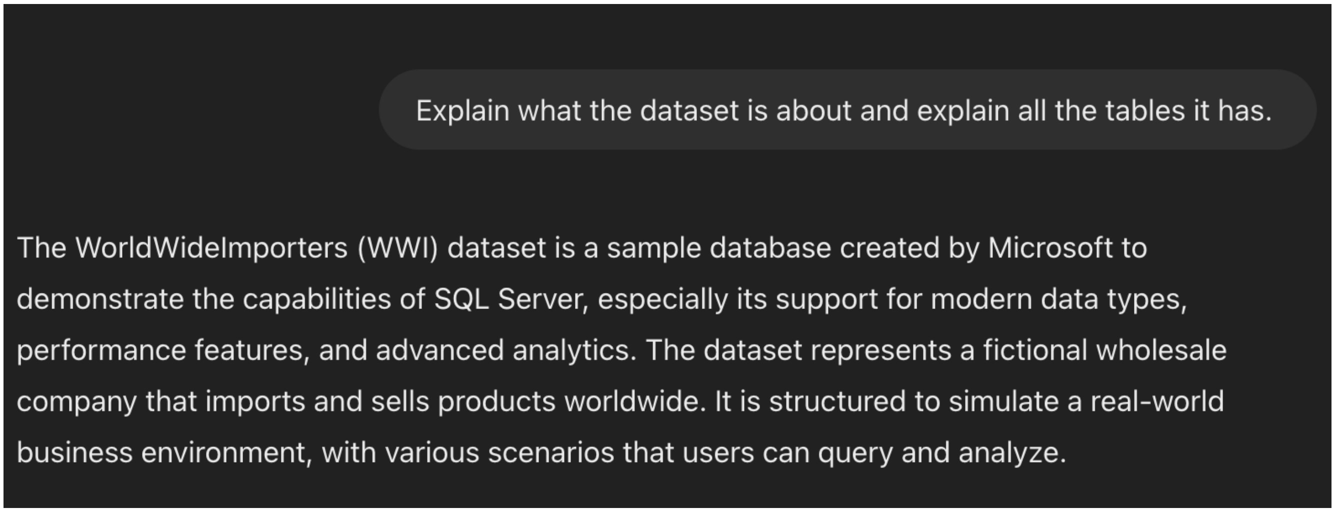


Do you know Worldwide Importers Dataset for SQL?

Yes, I'm familiar with the WorldWideImporters dataset. It's a sample database provided by Microsoft, designed for learning and practicing SQL queries. The dataset simulates a fictional wholesale business that imports and distributes products worldwide, including customers, suppliers, orders, stock items, and more.

If you need help with specific queries or tables in this dataset, feel free to ask!

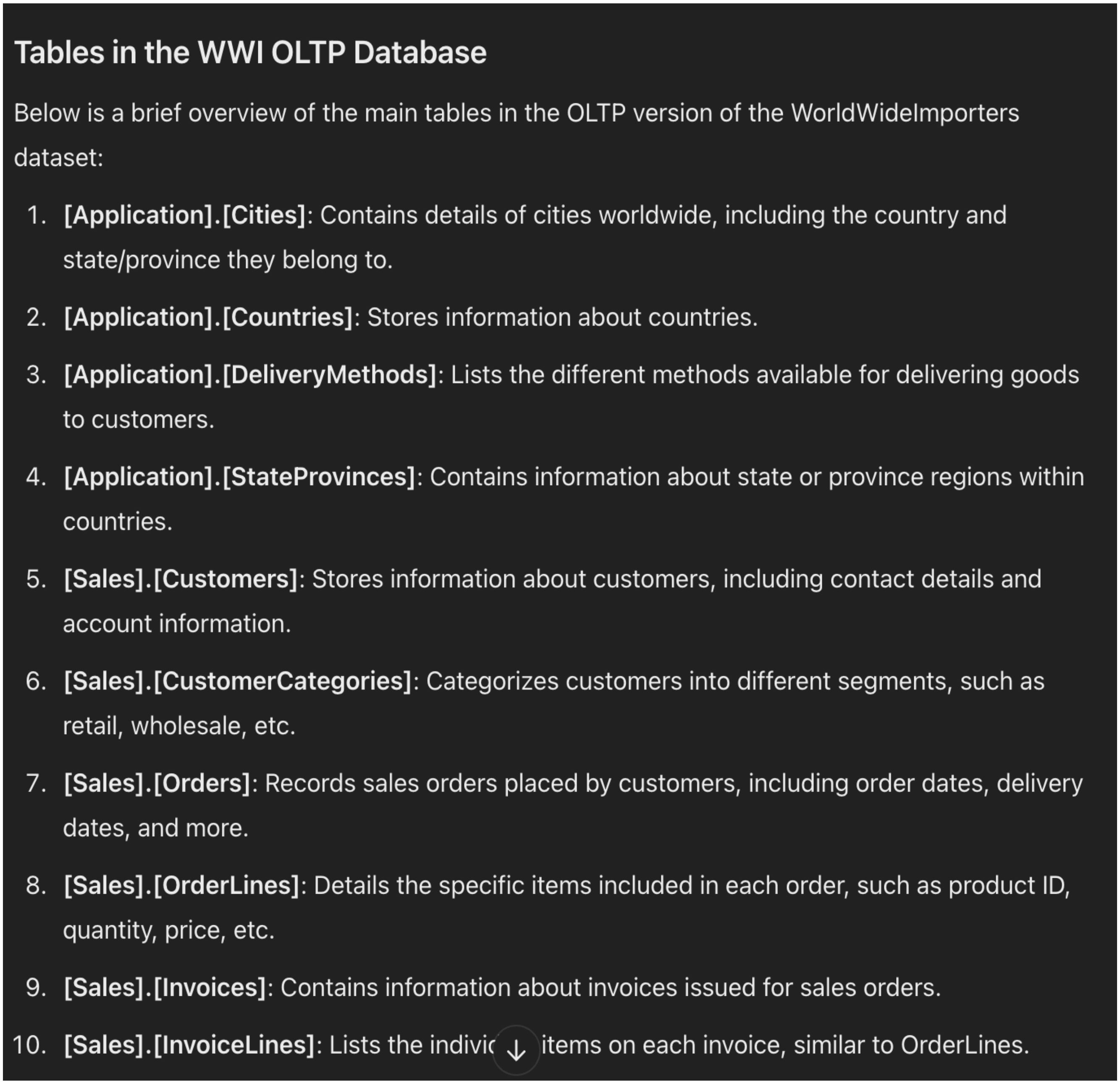
Figure 2: LLM response to its knowledge of the dataset:



Explain what the dataset is about and explain all the tables it has.

The WorldWideImporters (WWI) dataset is a sample database created by Microsoft to demonstrate the capabilities of SQL Server, especially its support for modern data types, performance features, and advanced analytics. The dataset represents a fictional wholesale company that imports and sells products worldwide. It is structured to simulate a real-world business environment, with various scenarios that users can query and analyze.

Figure 3: LLM response to its knowledge of all the tables:

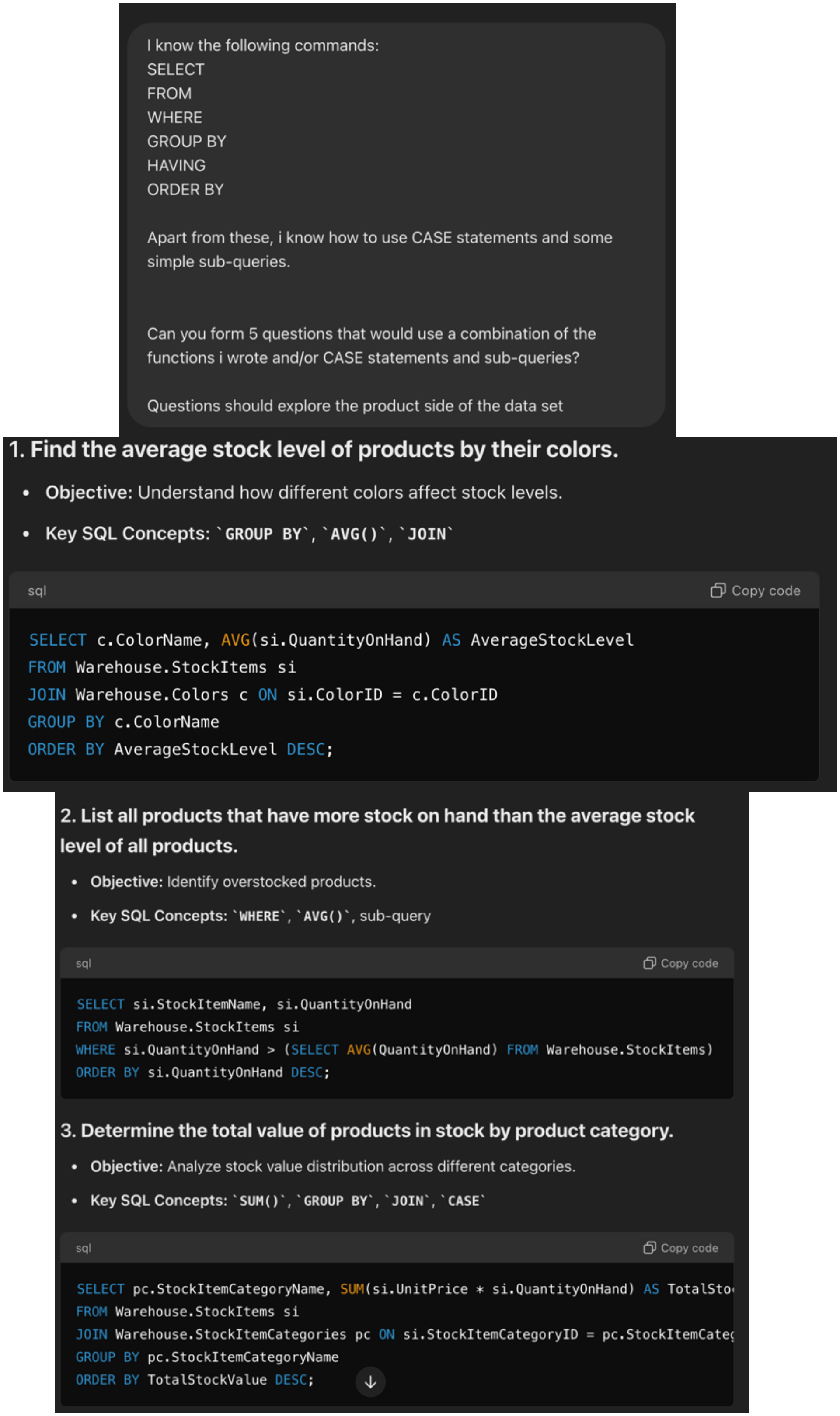


Tables in the WWI OLTP Database

Below is a brief overview of the main tables in the OLTP version of the WorldWideImporters dataset:

1. **[Application].[Cities]**: Contains details of cities worldwide, including the country and state/province they belong to.
2. **[Application].[Countries]**: Stores information about countries.
3. **[Application].[DeliveryMethods]**: Lists the different methods available for delivering goods to customers.
4. **[Application].[StateProvinces]**: Contains information about state or province regions within countries.
5. **[Sales].[Customers]**: Stores information about customers, including contact details and account information.
6. **[Sales].[CustomerCategories]**: Categorizes customers into different segments, such as retail, wholesale, etc.
7. **[Sales].[Orders]**: Records sales orders placed by customers, including order dates, delivery dates, and more.
8. **[Sales].[OrderLines]**: Details the specific items included in each order, such as product ID, quantity, price, etc.
9. **[Sales].[Invoices]**: Contains information about invoices issued for sales orders.
10. **[Sales].[InvoiceLines]**: Lists the indivic items on each invoice, similar to OrderLines.

Figure 4: LLM response for EDA questions:



I know the following commands:
SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY

Apart from these, i know how to use CASE statements and some simple sub-queries.

Can you form 5 questions that would use a combination of the functions i wrote and/or CASE statements and sub-queries?

Questions should explore the product side of the data set

1. Find the average stock level of products by their colors.

- **Objective:** Understand how different colors affect stock levels.
- **Key SQL Concepts:** `GROUP BY`, `AVG()`, `JOIN`

```sql
SELECT c.ColorName, AVG(si.QuantityOnHand) AS AverageStockLevel
FROM Warehouse.StockItems si
JOIN Warehouse.Colors c ON si.ColorID = c.ColorID
GROUP BY c.ColorName
ORDER BY AverageStockLevel DESC;
```

2. List all products that have more stock on hand than the average stock level of all products.

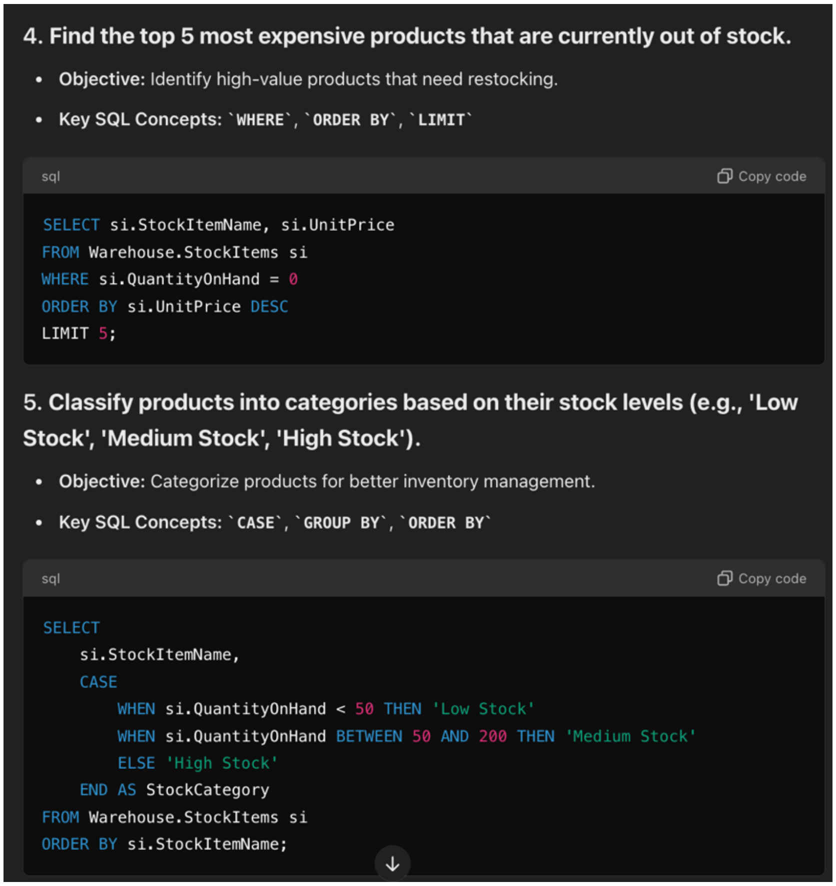
- **Objective:** Identify overstocked products.
- **Key SQL Concepts:** `WHERE`, `AVG()`, sub-query

```sql
SELECT si.StockItemName, si.QuantityOnHand
FROM Warehouse.StockItems si
WHERE si.QuantityOnHand > (SELECT AVG(QuantityOnHand) FROM Warehouse.StockItems)
ORDER BY si.QuantityOnHand DESC;
```

3. Determine the total value of products in stock by product category.

- **Objective:** Analyze stock value distribution across different categories.
- **Key SQL Concepts:** `SUM()`, `GROUP BY`, `JOIN`, `CASE`

```sql
SELECT pc.StockItemCategoryName, SUM(si.UnitPrice * si.QuantityOnHand) AS TotalSto
FROM Warehouse.StockItems si
JOIN Warehouse.StockItemCategories pc ON si.StockItemCategoryID = pc.StockItemCate
GROUP BY pc.StockItemCategoryName
ORDER BY TotalStockValue DESC;
```



4. Find the top 5 most expensive products that are currently out of stock.

- **Objective:** Identify high-value products that need restocking.
- **Key SQL Concepts:** `WHERE`, `ORDER BY`, `LIMIT`

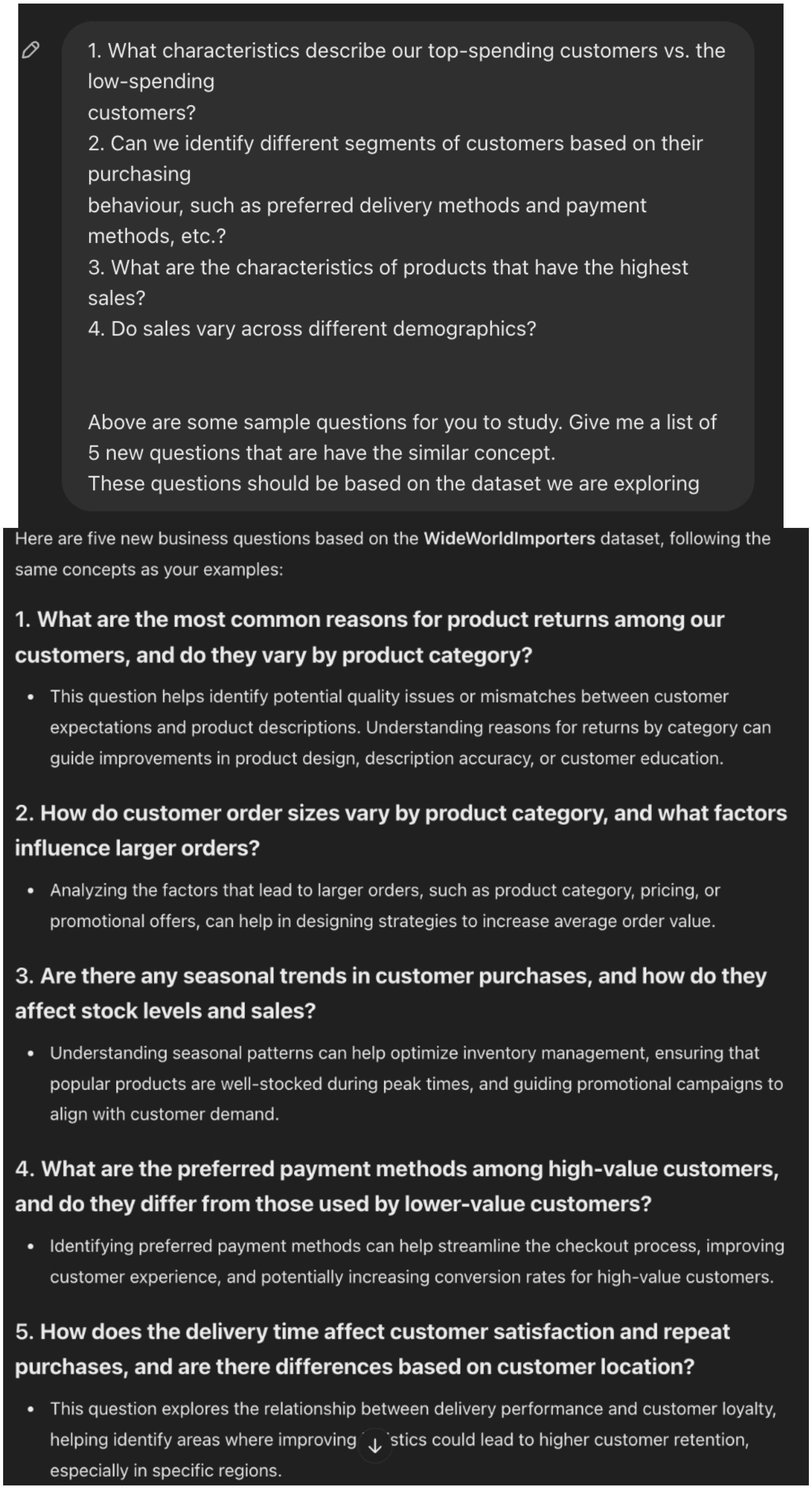
```sql
SELECT si.StockItemName, si.UnitPrice
FROM Warehouse.StockItems si
WHERE si.QuantityOnHand = 0
ORDER BY si.UnitPrice DESC
LIMIT 5;
```

5. Classify products into categories based on their stock levels (e.g., 'Low Stock', 'Medium Stock', 'High Stock').

- **Objective:** Categorize products for better inventory management.
- **Key SQL Concepts:** `CASE`, `GROUP BY`, `ORDER BY`

```sql
SELECT
    si.StockItemName,
    CASE
        WHEN si.QuantityOnHand < 50 THEN 'Low Stock'
        WHEN si.QuantityOnHand BETWEEN 50 AND 200 THEN 'Medium Stock'
        ELSE 'High Stock'
    END AS StockCategory
FROM Warehouse.StockItems si
ORDER BY si.StockItemName;
```

Figure 5: Questions produced by LLM based on examples provided:



1. What characteristics describe our top-spending customers vs. the low-spending
customers?
2. Can we identify different segments of customers based on their purchasing
behaviour, such as preferred delivery methods and payment methods, etc.?
3. What are the characteristics of products that have the highest sales?
4. Do sales vary across different demographics?

Above are some sample questions for you to study. Give me a list of 5 new questions that are have the similar concept.
These questions should be based on the dataset we are exploring

Here are five new business questions based on the **WideWorldImporters** dataset, following the same concepts as your examples:

1. What are the most common reasons for product returns among our customers, and do they vary by product category?

- This question helps identify potential quality issues or mismatches between customer expectations and product descriptions. Understanding reasons for returns by category can guide improvements in product design, description accuracy, or customer education.

2. How do customer order sizes vary by product category, and what factors influence larger orders?

- Analyzing the factors that lead to larger orders, such as product category, pricing, or promotional offers, can help in designing strategies to increase average order value.

3. Are there any seasonal trends in customer purchases, and how do they affect stock levels and sales?

- Understanding seasonal patterns can help optimize inventory management, ensuring that popular products are well-stocked during peak times, and guiding promotional campaigns to align with customer demand.

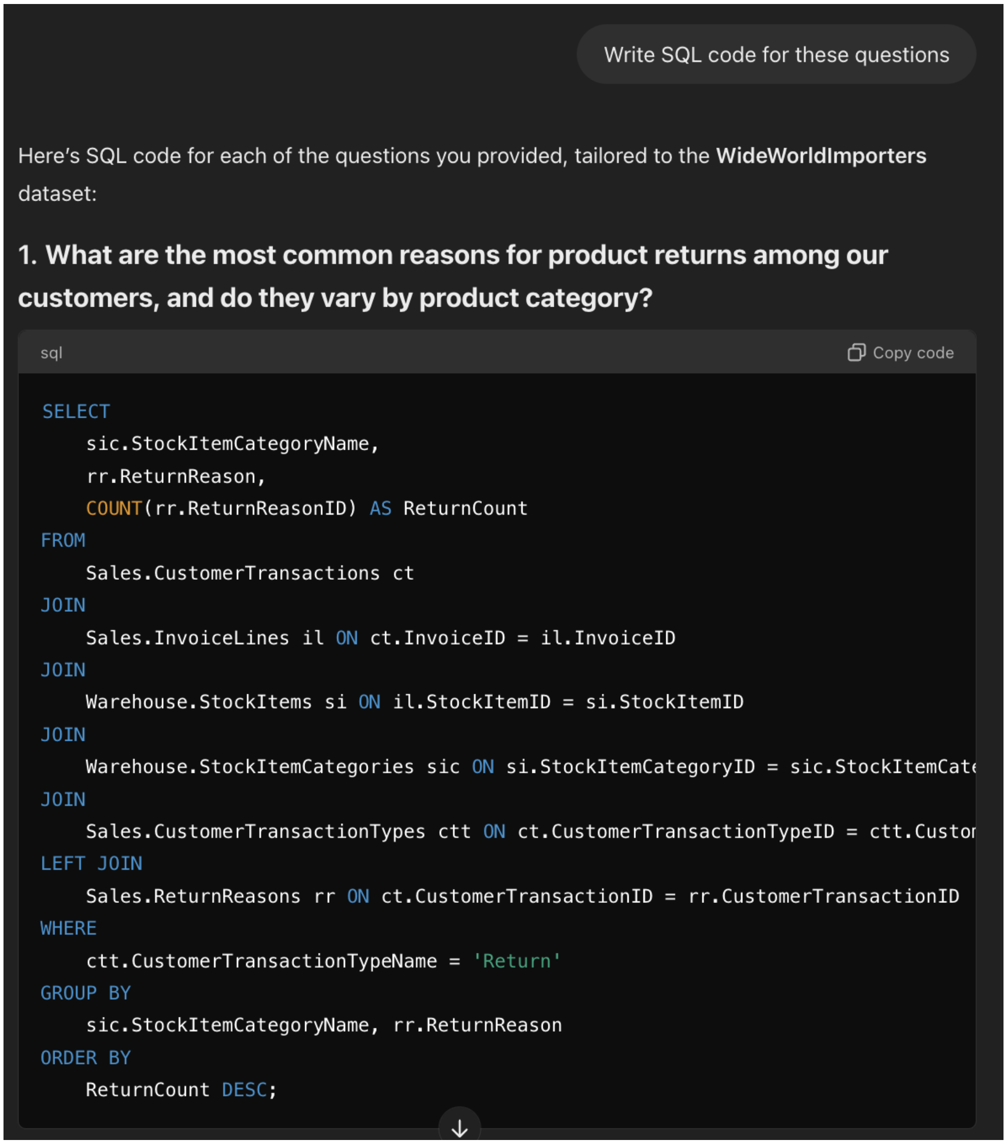
4. What are the preferred payment methods among high-value customers, and do they differ from those used by lower-value customers?

- Identifying preferred payment methods can help streamline the checkout process, improving customer experience, and potentially increasing conversion rates for high-value customers.

5. How does the delivery time affect customer satisfaction and repeat purchases, and are there differences based on customer location?

- This question explores the relationship between delivery performance and customer loyalty, helping identify areas where improving 'stics could lead to higher customer retention, especially in specific regions.

Figure 6: Syntax generated using dummy tables:



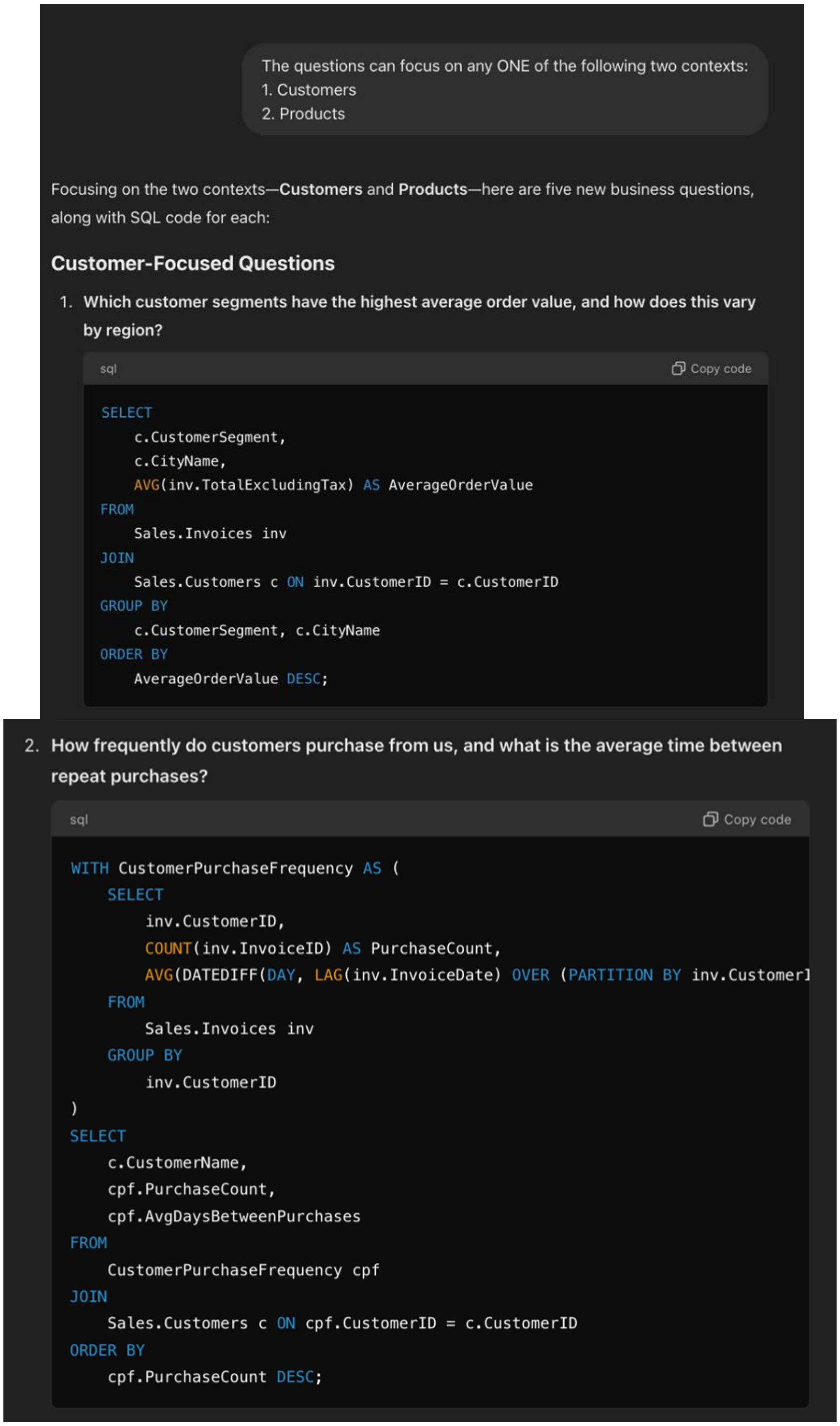
Write SQL code for these questions

Here’s SQL code for each of the questions you provided, tailored to the **WideWorldImporters** dataset:

1. What are the most common reasons for product returns among our customers, and do they vary by product category?

```sql
SELECT
    sic.StockItemCategoryName,
    rr.ReturnReason,
    COUNT(rr.ReturnReasonID) AS ReturnCount
FROM
    Sales.CustomerTransactions ct
JOIN
    Sales.InvoiceLines il ON ct.InvoiceID = il.InvoiceID
JOIN
    Warehouse.StockItems si ON il.StockItemID = si.StockItemID
JOIN
    Warehouse.StockItemCategories sic ON si.StockItemCategoryID = sic.StockItemCate
JOIN
    Sales.CustomerTransactionTypes ctt ON ct.CustomerTransactionTypeID = ctt.Custor
LEFT JOIN
    Sales.ReturnReasons rr ON ct.CustomerTransactionID = rr.CustomerTransactionID
WHERE
    ctt.CustomerTransactionTypeName = 'Return'
GROUP BY
    sic.StockItemCategoryName, rr.ReturnReason
ORDER BY
    ReturnCount DESC;
```

Figure 7: LLM asked to focus on Customer or Product domain.



The questions can focus on any ONE of the following two contexts:
1. Customers
2. Products

Focusing on the two contexts—**Customers** and **Products**—here are five new business questions, along with SQL code for each:

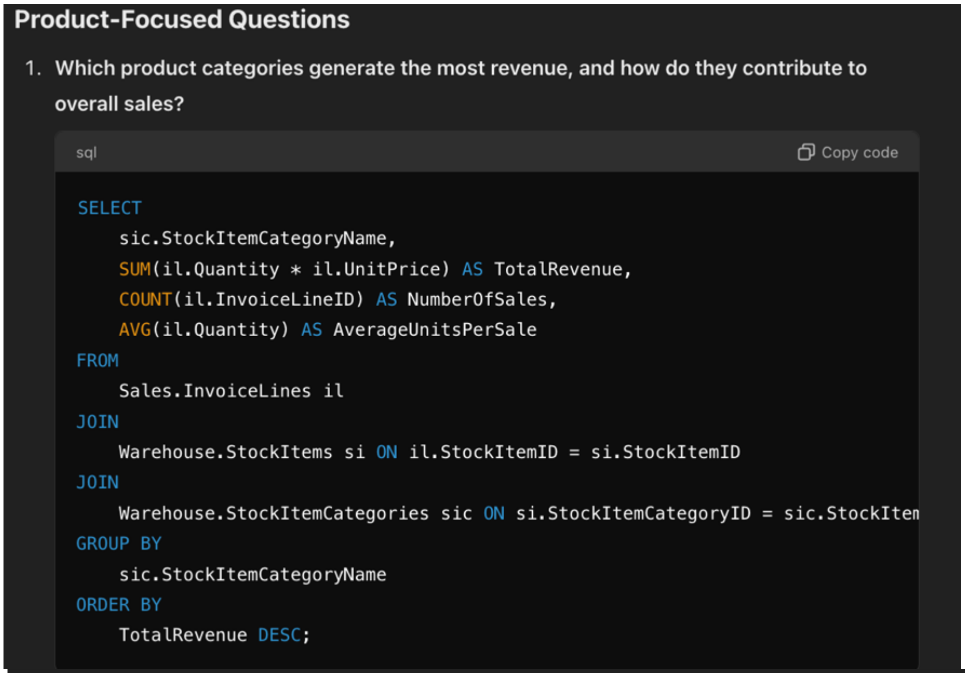
Customer-Focused Questions

1. **Which customer segments have the highest average order value, and how does this vary by region?**

```sql
SELECT
    c.CustomerSegment,
    c.CityName,
    AVG(inv.TotalExcludingTax) AS AverageOrderValue
FROM
    Sales.Invoices inv
JOIN
    Sales.Customers c ON inv.CustomerID = c.CustomerID
GROUP BY
    c.CustomerSegment, c.CityName
ORDER BY
    AverageOrderValue DESC;
```

2. **How frequently do customers purchase from us, and what is the average time between repeat purchases?**

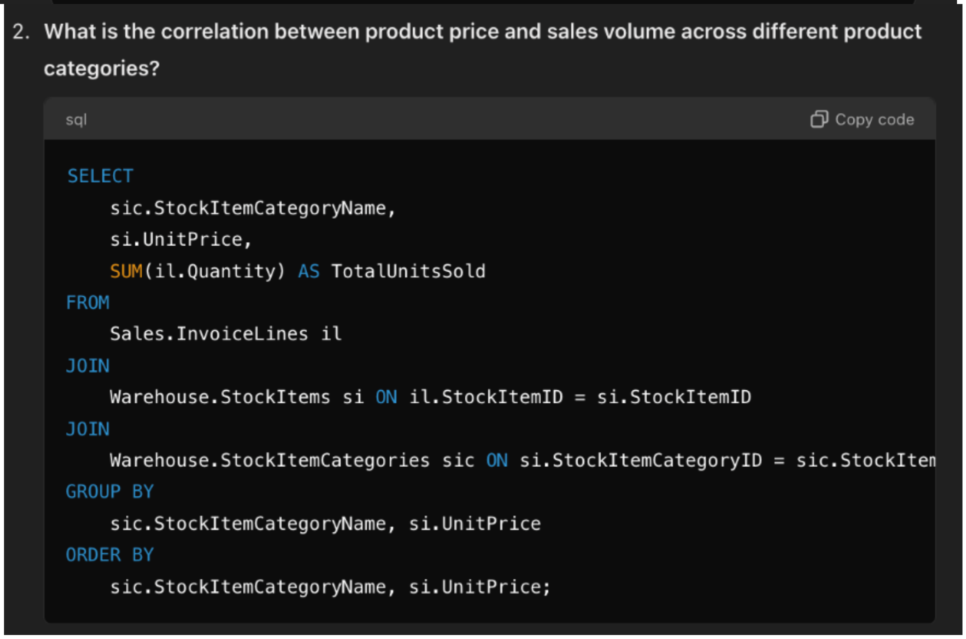
```sql
WITH CustomerPurchaseFrequency AS (
    SELECT
        inv.CustomerID,
        COUNT(inv.InvoiceID) AS PurchaseCount,
        AVG(DATEDIFF(DAY, LAG(inv.InvoiceDate) OVER (PARTITION BY inv.Customer
    FROM
        Sales.Invoices inv
    GROUP BY
        inv.CustomerID
)
SELECT
    c.CustomerName,
    cpf.PurchaseCount,
    cpf.AvgDaysBetweenPurchases
FROM
    CustomerPurchaseFrequency cpf
JOIN
    Sales.Customers c ON cpf.CustomerID = c.CustomerID
ORDER BY
    cpf.PurchaseCount DESC;
```



Product-Focused Questions

1. **Which product categories generate the most revenue, and how do they contribute to overall sales?**

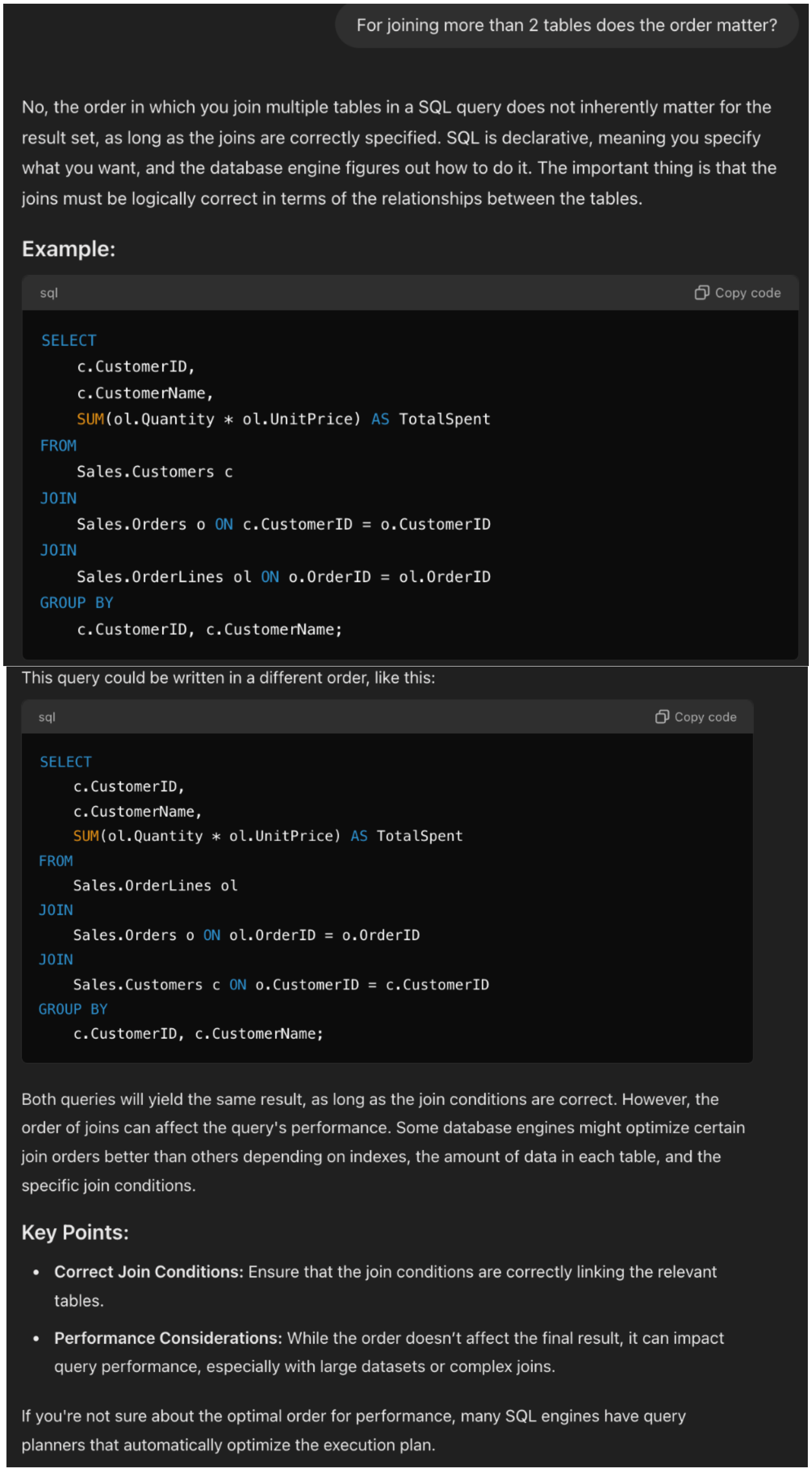
```sql
SELECT
    sic.StockItemCategoryName,
    SUM(il.Quantity * il.UnitPrice) AS TotalRevenue,
    COUNT(il.InvoiceLineID) AS NumberOfSales,
    AVG(il.Quantity) AS AverageUnitsPerSale
FROM
    Sales.InvoiceLines il
JOIN
    Warehouse.StockItems si ON il.StockItemID = si.StockItemID
JOIN
    Warehouse.StockItemCategories sic ON si.StockItemCategoryID = sic.StockItem
GROUP BY
    sic.StockItemCategoryName
ORDER BY
    TotalRevenue DESC;
```



2. **What is the correlation between product price and sales volume across different product categories?**

```sql
SELECT
    sic.StockItemCategoryName,
    si.UnitPrice,
    SUM(il.Quantity) AS TotalUnitsSold
FROM
    Sales.InvoiceLines il
JOIN
    Warehouse.StockItems si ON il.StockItemID = si.StockItemID
JOIN
    Warehouse.StockItemCategories sic ON si.StockItemCategoryID = sic.StockItem
GROUP BY
    sic.StockItemCategoryName, si.UnitPrice
ORDER BY
    sic.StockItemCategoryName, si.UnitPrice;
```

Figure 8: Understanding Concepts Using LLM:



For joining more than 2 tables does the order matter?

No, the order in which you join multiple tables in a SQL query does not inherently matter for the result set, as long as the joins are correctly specified. SQL is declarative, meaning you specify what you want, and the database engine figures out how to do it. The important thing is that the joins must be logically correct in terms of the relationships between the tables.

Example:

```sql
SELECT
    c.CustomerID,
    c.CustomerName,
    SUM(ol.Quantity * ol.UnitPrice) AS TotalSpent
FROM
    Sales.Customers c
JOIN
    Sales.Orders o ON c.CustomerID = o.CustomerID
JOIN
    Sales.OrderLines ol ON o.OrderID = ol.OrderID
GROUP BY
    c.CustomerID, c.CustomerName;
```

This query could be written in a different order, like this:

```sql
SELECT
    c.CustomerID,
    c.CustomerName,
    SUM(ol.Quantity * ol.UnitPrice) AS TotalSpent
FROM
    Sales.OrderLines ol
JOIN
    Sales.Orders o ON ol.OrderID = o.OrderID
JOIN
    Sales.Customers c ON o.CustomerID = c.CustomerID
GROUP BY
    c.CustomerID, c.CustomerName;
```

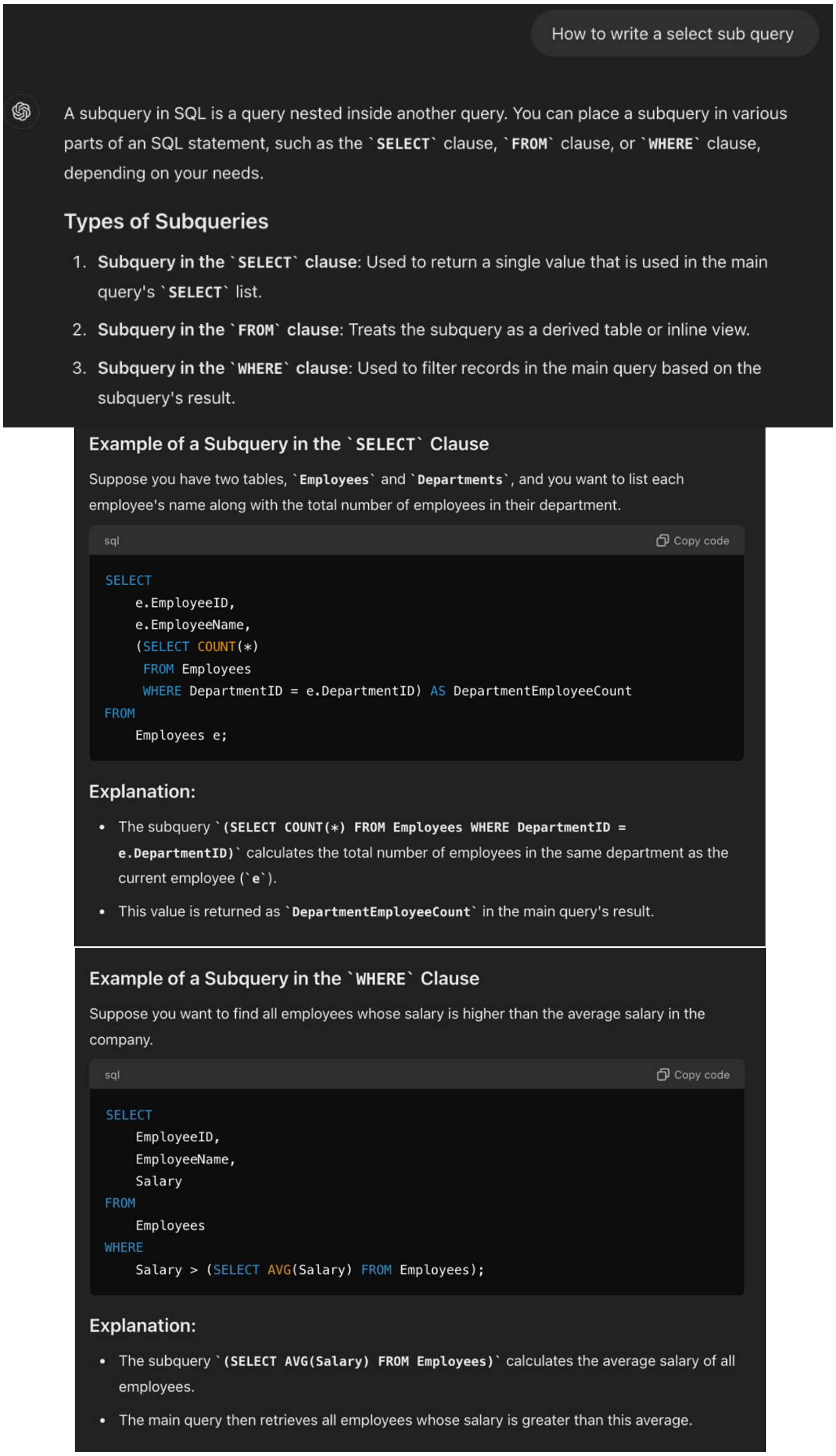
Both queries will yield the same result, as long as the join conditions are correct. However, the order of joins can affect the query's performance. Some database engines might optimize certain join orders better than others depending on indexes, the amount of data in each table, and the specific join conditions.

Key Points:

- **Correct Join Conditions:** Ensure that the join conditions are correctly linking the relevant tables.
- **Performance Considerations:** While the order doesn't affect the final result, it can impact query performance, especially with large datasets or complex joins.

If you're not sure about the optimal order for performance, many SQL engines have query planners that automatically optimize the execution plan.

Figure 9: Asking LLM how to write a SELECT Subquery:



How to write a select sub query

A subquery in SQL is a query nested inside another query. You can place a subquery in various parts of an SQL statement, such as the `SELECT` clause, `FROM` clause, or `WHERE` clause, depending on your needs.

Types of Subqueries

1. **Subquery in the `SELECT` clause**: Used to return a single value that is used in the main query's `SELECT` list.
2. **Subquery in the `FROM` clause**: Treats the subquery as a derived table or inline view.
3. **Subquery in the `WHERE` clause**: Used to filter records in the main query based on the subquery's result.

Example of a Subquery in the `SELECT` Clause

Suppose you have two tables, `Employees` and `Departments`, and you want to list each employee's name along with the total number of employees in their department.

```sql
SELECT
    e.EmployeeID,
    e.EmployeeName,
    (SELECT COUNT(*)
     FROM Employees
     WHERE DepartmentID = e.DepartmentID) AS DepartmentEmployeeCount
FROM
    Employees e;
```

Explanation:

- The subquery `(SELECT COUNT(*) FROM Employees WHERE DepartmentID = e.DepartmentID)` calculates the total number of employees in the same department as the current employee (`e`).
- This value is returned as `DepartmentEmployeeCount` in the main query's result.

Example of a Subquery in the `WHERE` Clause

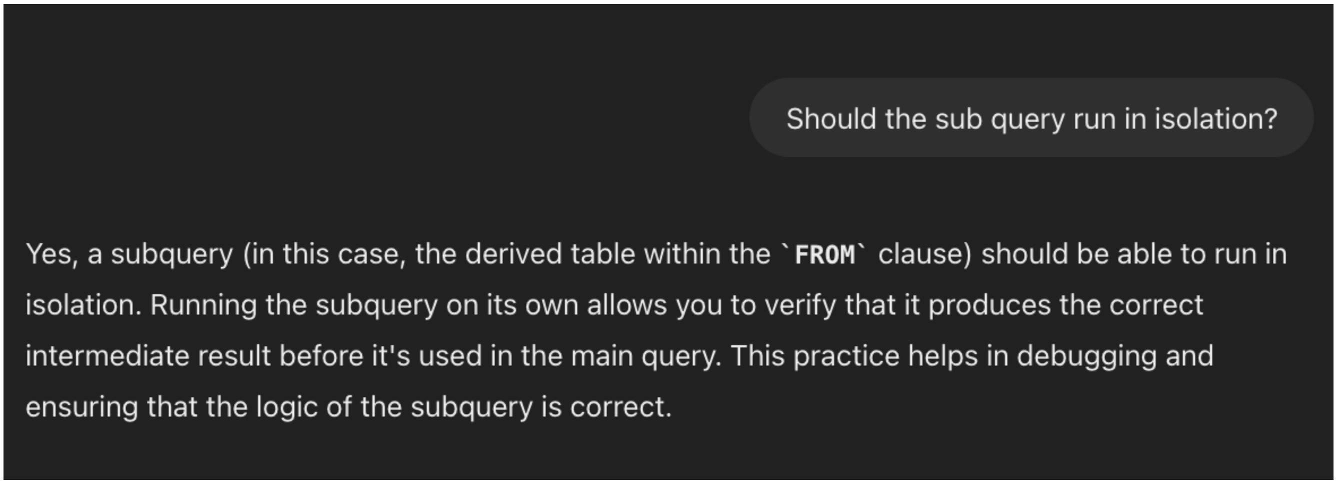
Suppose you want to find all employees whose salary is higher than the average salary in the company.

```sql
SELECT
    EmployeeID,
    EmployeeName,
    Salary
FROM
    Employees
WHERE
    Salary > (SELECT AVG(Salary) FROM Employees);
```

Explanation:

- The subquery `(SELECT AVG(Salary) FROM Employees)` calculates the average salary of all employees.
- The main query then retrieves all employees whose salary is greater than this average.

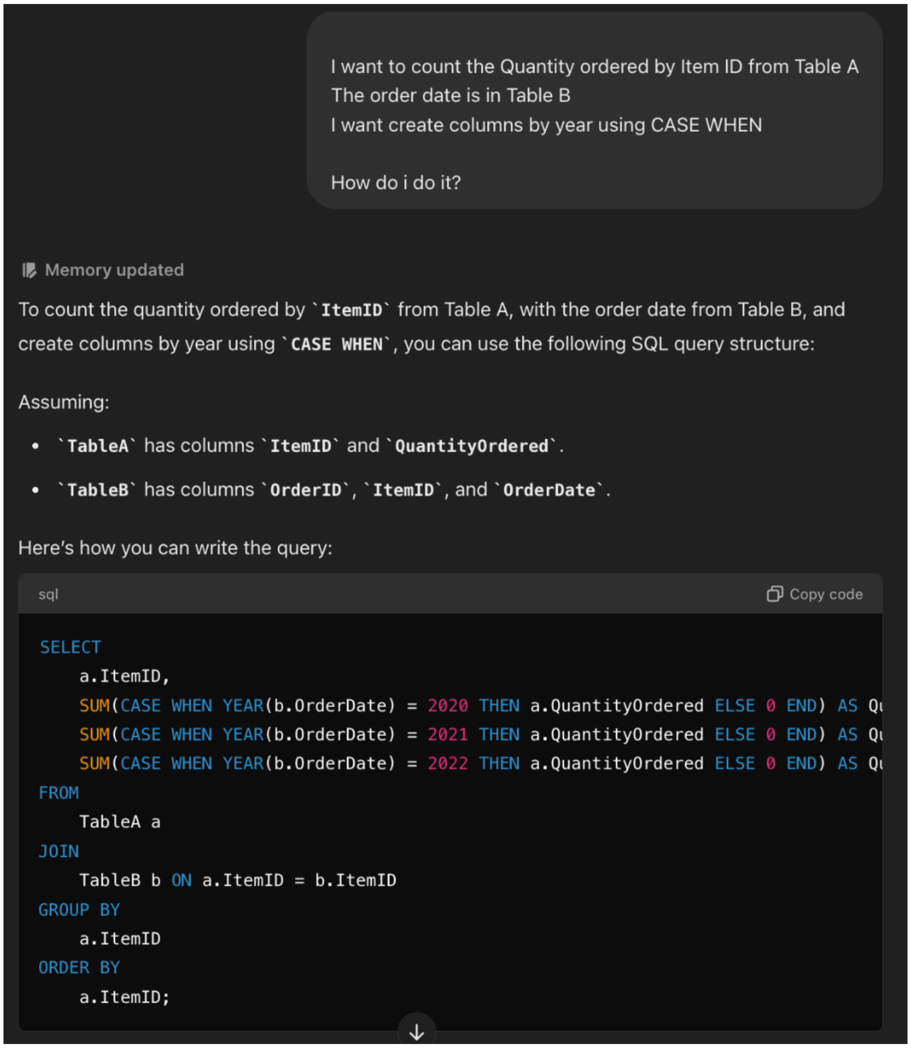
Figure 10: Asking LLM whether a subquery should execute in isolation:



Should the sub query run in isolation?

Yes, a subquery (in this case, the derived table within the `FROM` clause) should be able to run in isolation. Running the subquery on its own allows you to verify that it produces the correct intermediate result before it's used in the main query. This practice helps in debugging and ensuring that the logic of the subquery is correct.

Figure 11: Asking LLM to write a query based on dummy dataset:



I want to count the Quantity ordered by Item ID from Table A
The order date is in Table B
I want create columns by year using CASE WHEN

How do i do it?

To count the quantity ordered by `ItemID` from Table A, with the order date from Table B, and create columns by year using `CASE WHEN`, you can use the following SQL query structure:

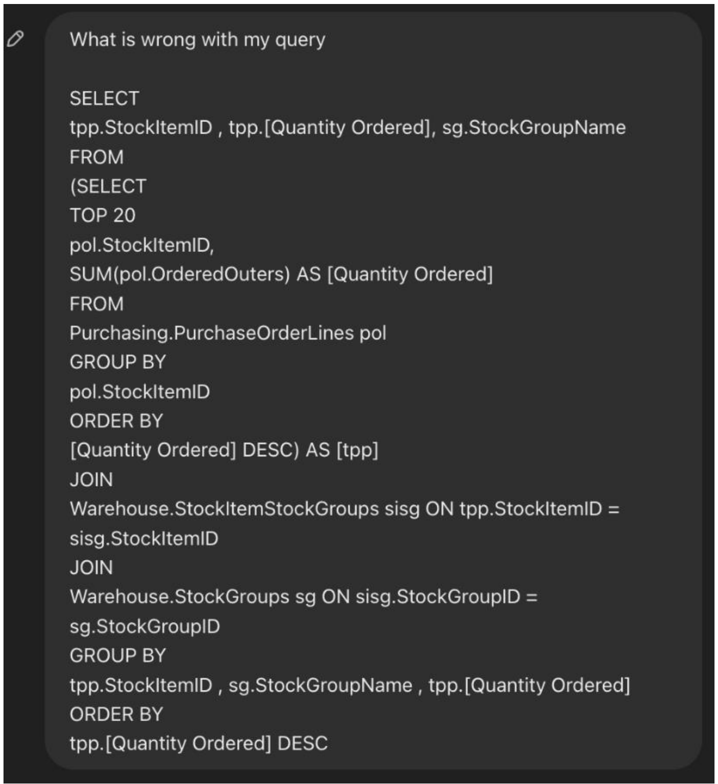
Assuming:

- `TableA` has columns `ItemID` and `QuantityOrdered`.
- `TableB` has columns `OrderID`, `ItemID`, and `OrderDate`.

Here's how you can write the query:

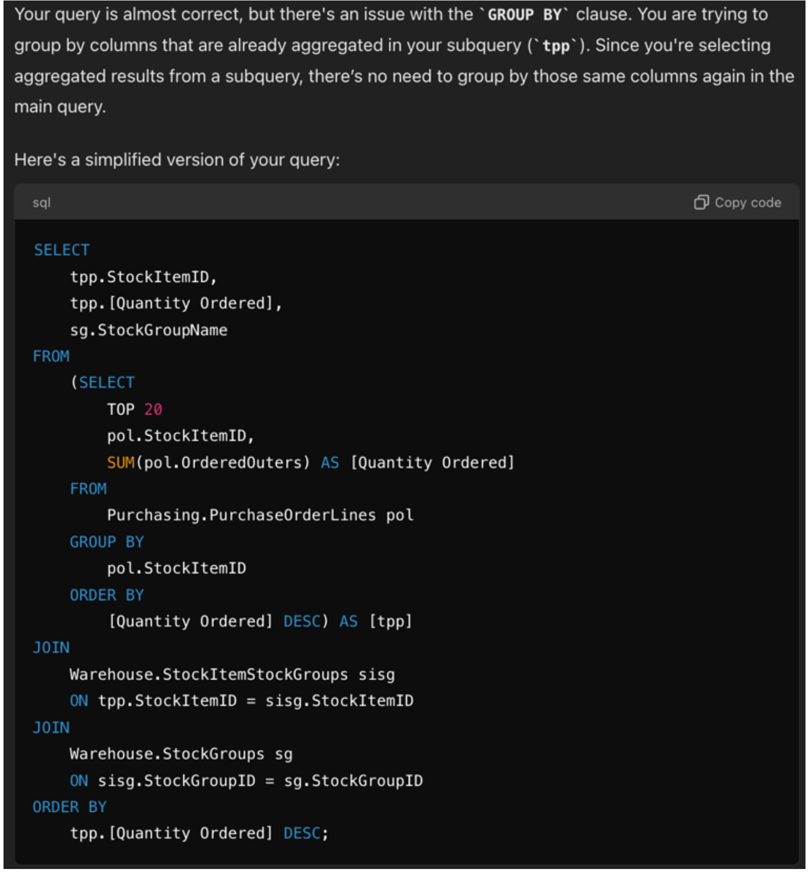
```sql
SELECT
    a.ItemID,
    SUM(CASE WHEN YEAR(b.OrderDate) = 2020 THEN a.QuantityOrdered ELSE 0 END) AS Qu
    SUM(CASE WHEN YEAR(b.OrderDate) = 2021 THEN a.QuantityOrdered ELSE 0 END) AS Qu
    SUM(CASE WHEN YEAR(b.OrderDate) = 2022 THEN a.QuantityOrdered ELSE 0 END) AS Qu
FROM
    TableA a
JOIN
    TableB b ON a.ItemID = b.ItemID
GROUP BY
    a.ItemID
ORDER BY
    a.ItemID;
```

Figure 12: Asking LLM to Fix My Code:



What is wrong with my query

```
SELECT
tpp.StockItemID , tpp.[Quantity Ordered], sg.StockGroupName
FROM
(SELECT
TOP 20
pol.StockItemID,
SUM(pol.OrderedOuters) AS [Quantity Ordered]
FROM
Purchasing.PurchaseOrderLines pol
GROUP BY
pol.StockItemID
ORDER BY
[Quantity Ordered] DESC) AS [tpp]
JOIN
Warehouse.StockItemStockGroups sisg ON tpp.StockItemID =
sisg.StockItemID
JOIN
Warehouse.StockGroups sg ON sisg.StockGroupID =
sg.StockGroupID
GROUP BY
tpp.StockItemID , sg.StockGroupName , tpp.[Quantity Ordered]
ORDER BY
tpp.[Quantity Ordered] DESC
```

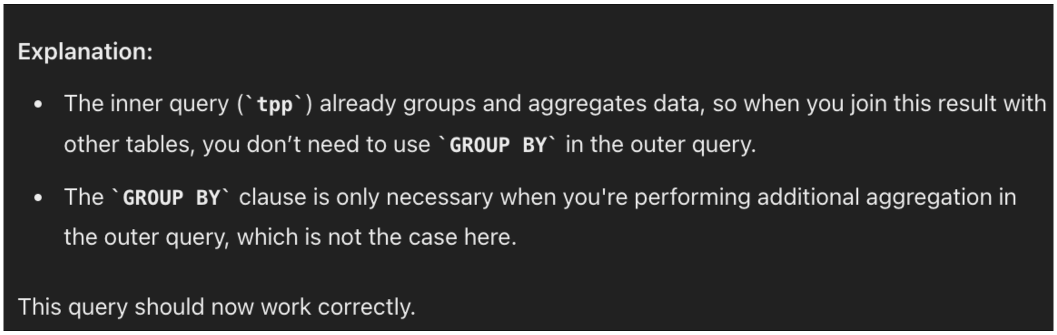


Your query is almost correct, but there's an issue with the `GROUP BY` clause. You are trying to group by columns that are already aggregated in your subquery (`tpp`). Since you're selecting aggregated results from a subquery, there's no need to group by those same columns again in the main query.

Here's a simplified version of your query:

```sql
SELECT
    tpp.StockItemID,
    tpp.[Quantity Ordered],
    sg.StockGroupName
FROM
    (SELECT
        TOP 20
        pol.StockItemID,
        SUM(pol.OrderedOuters) AS [Quantity Ordered]
    FROM
        Purchasing.PurchaseOrderLines pol
    GROUP BY
        pol.StockItemID
    ORDER BY
        [Quantity Ordered] DESC) AS [tpp]
JOIN
    Warehouse.StockItemStockGroups sisg
    ON tpp.StockItemID = sisg.StockItemID
JOIN
    Warehouse.StockGroups sg
    ON sisg.StockGroupID = sg.StockGroupID
ORDER BY
    tpp.[Quantity Ordered] DESC;
```

Figure 13: Explanation of the Error from Figure 12 Code:

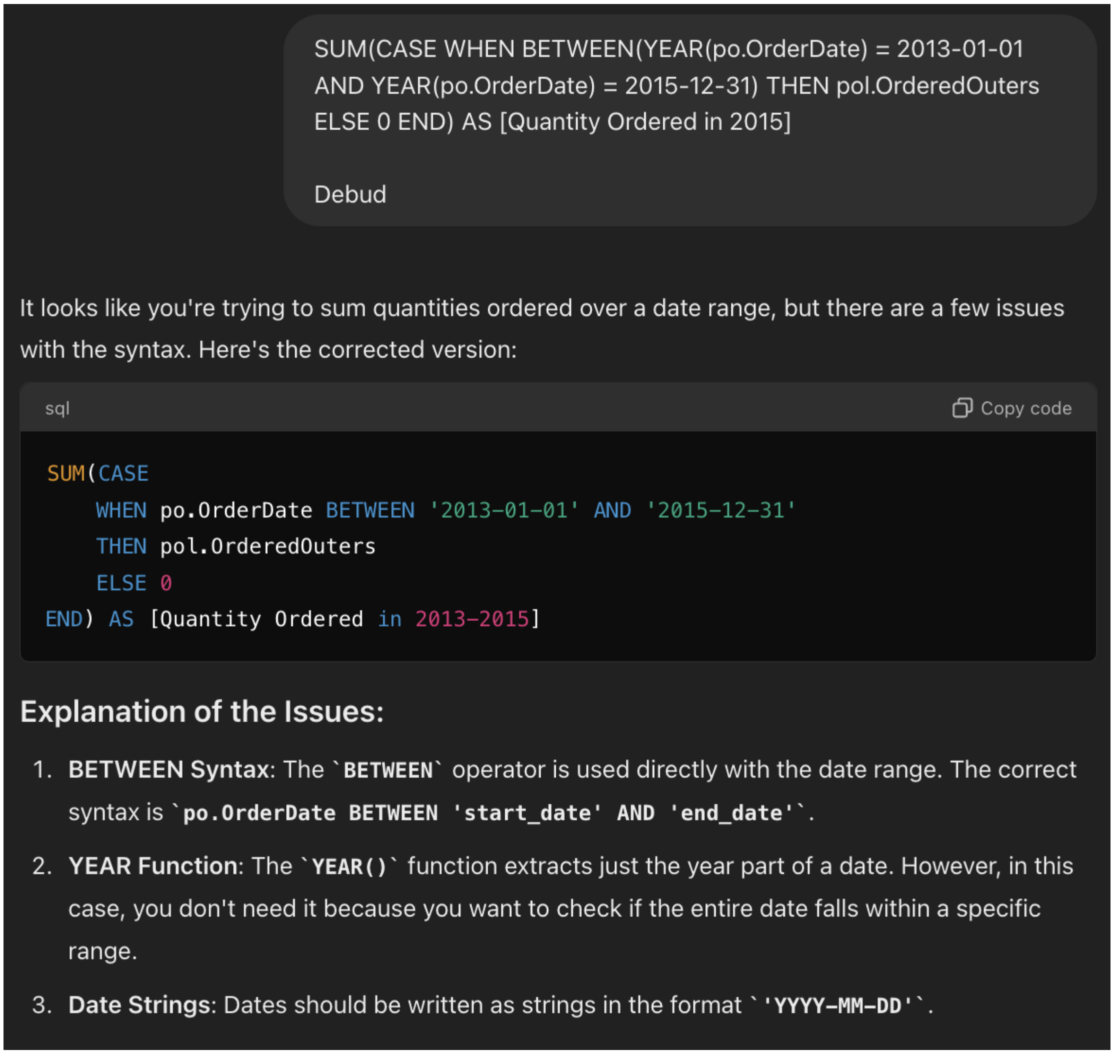


Explanation:

- The inner query (`tpp`) already groups and aggregates data, so when you join this result with other tables, you don't need to use `GROUP BY` in the outer query.
- The `GROUP BY` clause is only necessary when you're performing additional aggregation in the outer query, which is not the case here.

This query should now work correctly.

Figure 14: Debugging a query:



SUM(CASE WHEN BETWEEN(YEAR(po.OrderDate) = 2013-01-01 AND YEAR(po.OrderDate) = 2015-12-31) THEN pol.OrderedOuters ELSE 0 END) AS [Quantity Ordered in 2015]

Debud

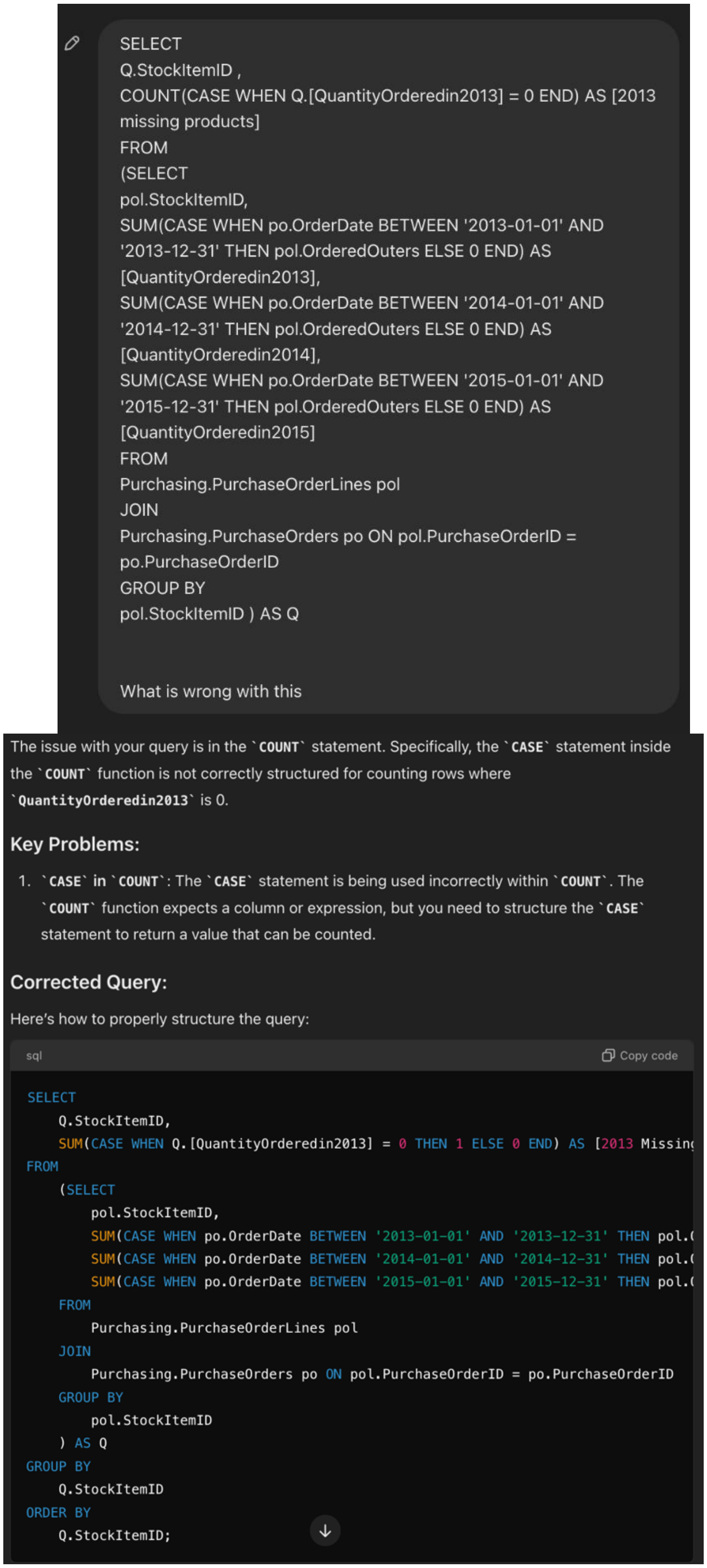
It looks like you're trying to sum quantities ordered over a date range, but there are a few issues with the syntax. Here's the corrected version:

```sql
SUM(CASE
    WHEN po.OrderDate BETWEEN '2013-01-01' AND '2015-12-31'
    THEN pol.OrderedOuters
    ELSE 0
END) AS [Quantity Ordered in 2013-2015]
```

Explanation of the Issues:

1. **BETWEEN Syntax**: The `BETWEEN` operator is used directly with the date range. The correct syntax is `po.OrderDate BETWEEN 'start_date' AND 'end_date'`.
2. **YEAR Function**: The `YEAR()` function extracts just the year part of a date. However, in this case, you don't need it because you want to check if the entire date falls within a specific range.
3. **Date Strings**: Dates should be written as strings in the format `'YYYY-MM-DD'`.

Figure 15: LLM's Ability to Debug a Long and Complex Query:



SELECT
Q.StockItemID ,
COUNT(CASE WHEN Q.[QuantityOrderedin2013] = 0 END) AS [2013 missing products]
FROM
(SELECT
pol.StockItemID,
SUM(CASE WHEN po.OrderDate BETWEEN '2013-01-01' AND '2013-12-31' THEN pol.OrderedOuters ELSE 0 END) AS [QuantityOrderedin2013],
SUM(CASE WHEN po.OrderDate BETWEEN '2014-01-01' AND '2014-12-31' THEN pol.OrderedOuters ELSE 0 END) AS [QuantityOrderedin2014],
SUM(CASE WHEN po.OrderDate BETWEEN '2015-01-01' AND '2015-12-31' THEN pol.OrderedOuters ELSE 0 END) AS [QuantityOrderedin2015]
FROM
Purchasing.PurchaseOrderLines pol
JOIN
Purchasing.PurchaseOrders po ON pol.PurchaseOrderID = po.PurchaseOrderID
GROUP BY
pol.StockItemID) AS Q

What is wrong with this

The issue with your query is in the `COUNT` statement. Specifically, the `CASE` statement inside the `COUNT` function is not correctly structured for counting rows where `QuantityOrderedin2013` is 0.

Key Problems:

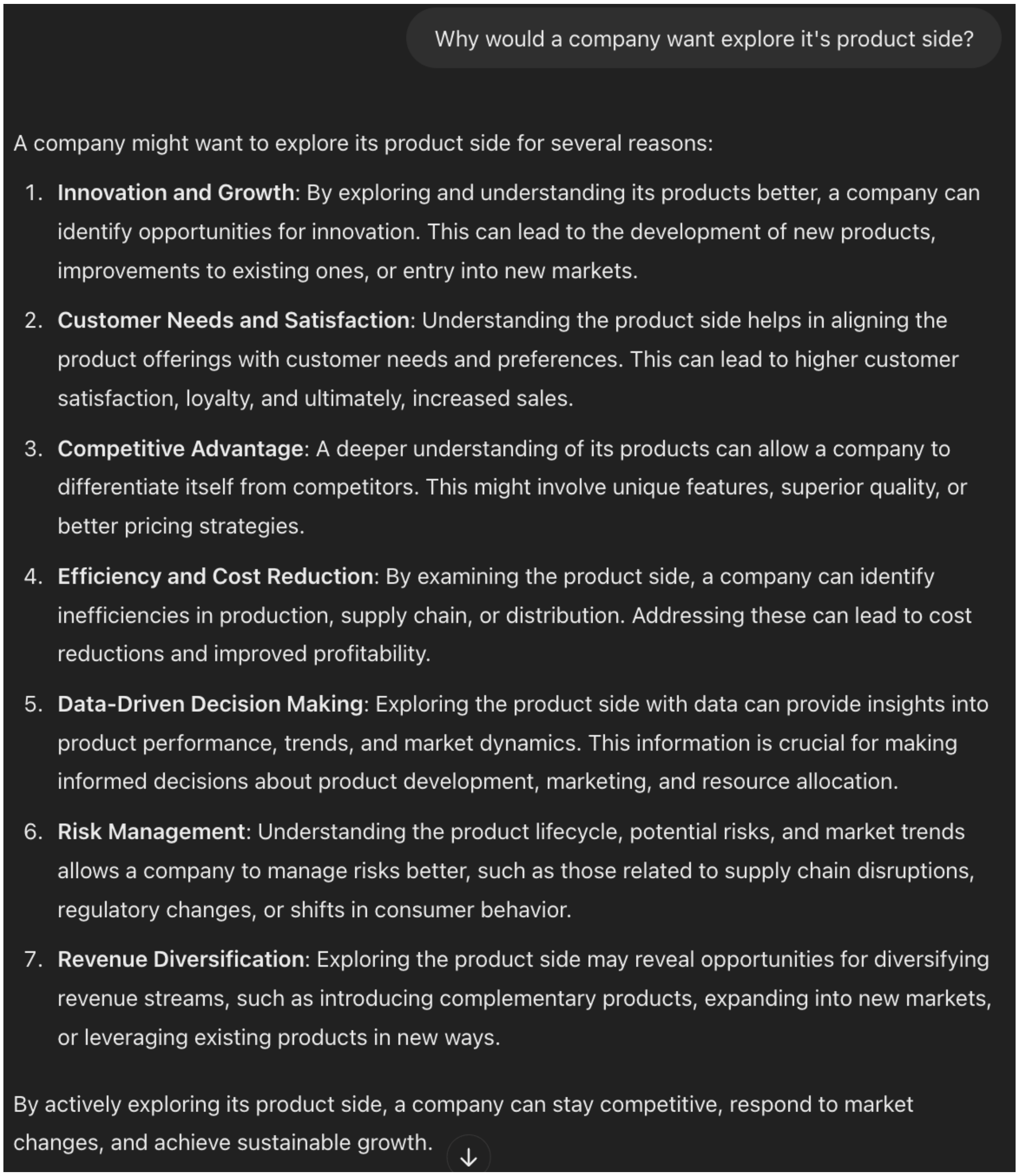
1. **`CASE` in `COUNT`**: The `CASE` statement is being used incorrectly within `COUNT`. The `COUNT` function expects a column or expression, but you need to structure the `CASE` statement to return a value that can be counted.

Corrected Query:

Here's how to properly structure the query:

```sql
SELECT
    Q.StockItemID,
    SUM(CASE WHEN Q.[QuantityOrderedin2013] = 0 THEN 1 ELSE 0 END) AS [2013 Missin
FROM
    (SELECT
        pol.StockItemID,
        SUM(CASE WHEN po.OrderDate BETWEEN '2013-01-01' AND '2013-12-31' THEN pol.
        SUM(CASE WHEN po.OrderDate BETWEEN '2014-01-01' AND '2014-12-31' THEN pol.
        SUM(CASE WHEN po.OrderDate BETWEEN '2015-01-01' AND '2015-12-31' THEN pol.
    FROM
        Purchasing.PurchaseOrderLines pol
    JOIN
        Purchasing.PurchaseOrders po ON pol.PurchaseOrderID = po.PurchaseOrderID
    GROUP BY
        pol.StockItemID
    ) AS Q
GROUP BY
    Q.StockItemID
ORDER BY
    Q.StockItemID;
```

Figure 16: LLM's Response to Business Problems:



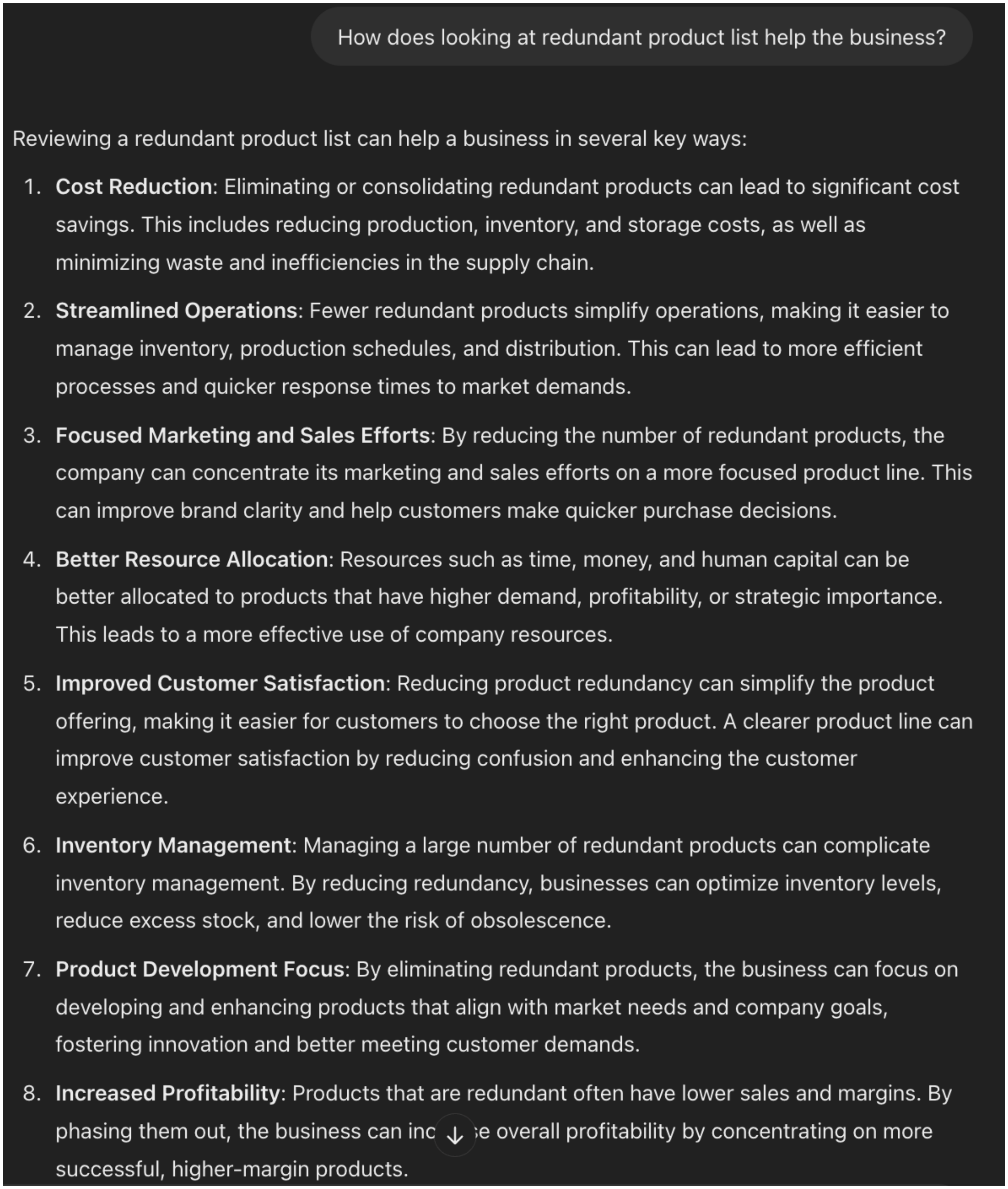
Why would a company want explore it's product side?

A company might want to explore its product side for several reasons:

1. **Innovation and Growth**: By exploring and understanding its products better, a company can identify opportunities for innovation. This can lead to the development of new products, improvements to existing ones, or entry into new markets.
2. **Customer Needs and Satisfaction**: Understanding the product side helps in aligning the product offerings with customer needs and preferences. This can lead to higher customer satisfaction, loyalty, and ultimately, increased sales.
3. **Competitive Advantage**: A deeper understanding of its products can allow a company to differentiate itself from competitors. This might involve unique features, superior quality, or better pricing strategies.
4. **Efficiency and Cost Reduction**: By examining the product side, a company can identify inefficiencies in production, supply chain, or distribution. Addressing these can lead to cost reductions and improved profitability.
5. **Data-Driven Decision Making**: Exploring the product side with data can provide insights into product performance, trends, and market dynamics. This information is crucial for making informed decisions about product development, marketing, and resource allocation.
6. **Risk Management**: Understanding the product lifecycle, potential risks, and market trends allows a company to manage risks better, such as those related to supply chain disruptions, regulatory changes, or shifts in consumer behavior.
7. **Revenue Diversification**: Exploring the product side may reveal opportunities for diversifying revenue streams, such as introducing complementary products, expanding into new markets, or leveraging existing products in new ways.

By actively exploring its product side, a company can stay competitive, respond to market changes, and achieve sustainable growth.

Figure 17: Business Insights With LLM's Help:

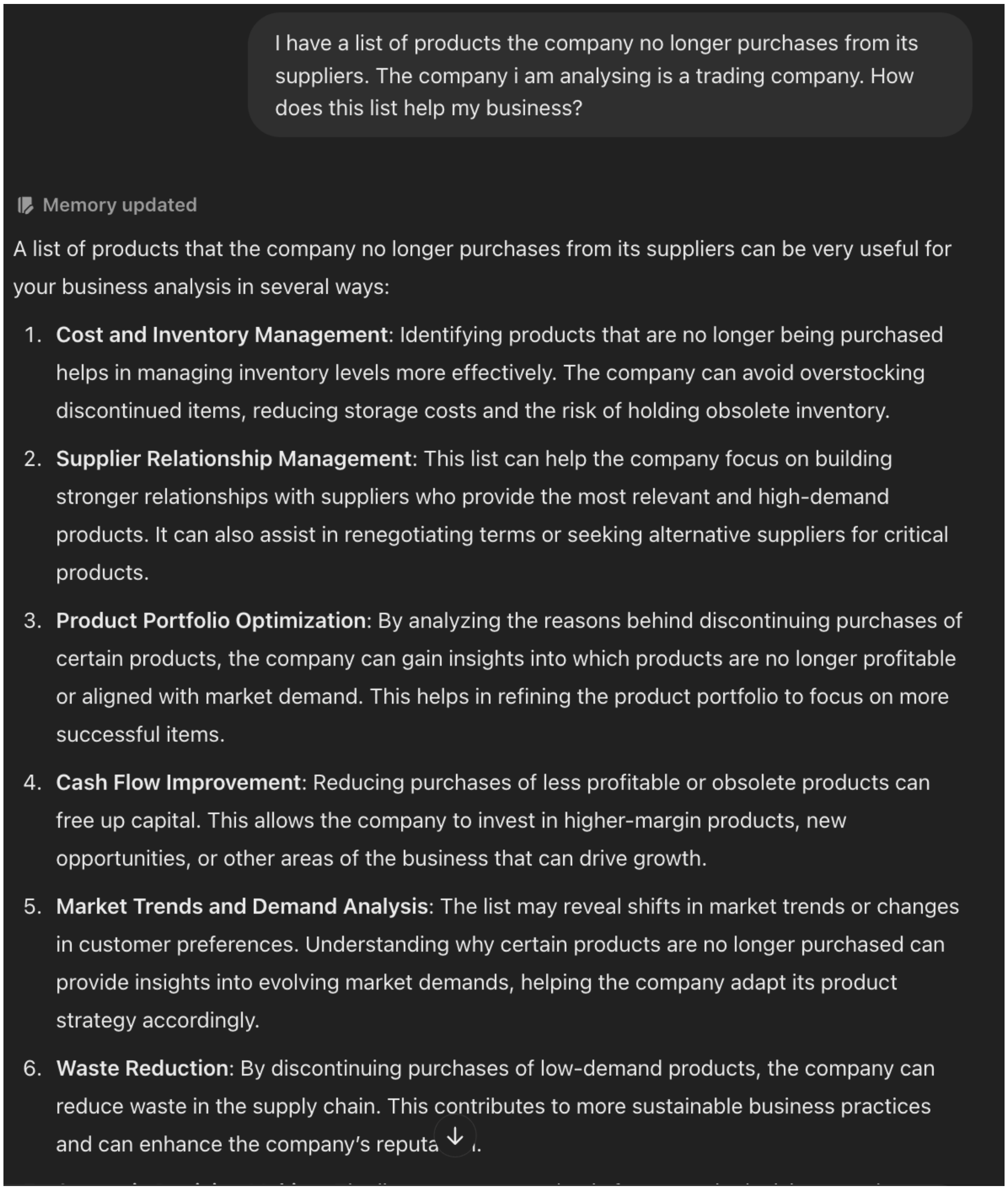


How does looking at redundant product list help the business?

Reviewing a redundant product list can help a business in several key ways:

1. **Cost Reduction**: Eliminating or consolidating redundant products can lead to significant cost savings. This includes reducing production, inventory, and storage costs, as well as minimizing waste and inefficiencies in the supply chain.
2. **Streamlined Operations**: Fewer redundant products simplify operations, making it easier to manage inventory, production schedules, and distribution. This can lead to more efficient processes and quicker response times to market demands.
3. **Focused Marketing and Sales Efforts**: By reducing the number of redundant products, the company can concentrate its marketing and sales efforts on a more focused product line. This can improve brand clarity and help customers make quicker purchase decisions.
4. **Better Resource Allocation**: Resources such as time, money, and human capital can be better allocated to products that have higher demand, profitability, or strategic importance. This leads to a more effective use of company resources.
5. **Improved Customer Satisfaction**: Reducing product redundancy can simplify the product offering, making it easier for customers to choose the right product. A clearer product line can improve customer satisfaction by reducing confusion and enhancing the customer experience.
6. **Inventory Management**: Managing a large number of redundant products can complicate inventory management. By reducing redundancy, businesses can optimize inventory levels, reduce excess stock, and lower the risk of obsolescence.
7. **Product Development Focus**: By eliminating redundant products, the business can focus on developing and enhancing products that align with market needs and company goals, fostering innovation and better meeting customer demands.
8. **Increased Profitability**: Products that are redundant often have lower sales and margins. By phasing them out, the business can inc ↓ se overall profitability by concentrating on more successful, higher-margin products.

Figure 18: Business Insights After Defining The Company as a Trading Company.



I have a list of products the company no longer purchases from its suppliers. The company i am analysing is a trading company. How does this list help my business?

A list of products that the company no longer purchases from its suppliers can be very useful for your business analysis in several ways:

1. **Cost and Inventory Management**: Identifying products that are no longer being purchased helps in managing inventory levels more effectively. The company can avoid overstocking discontinued items, reducing storage costs and the risk of holding obsolete inventory.
2. **Supplier Relationship Management**: This list can help the company focus on building stronger relationships with suppliers who provide the most relevant and high-demand products. It can also assist in renegotiating terms or seeking alternative suppliers for critical products.
3. **Product Portfolio Optimization**: By analyzing the reasons behind discontinuing purchases of certain products, the company can gain insights into which products are no longer profitable or aligned with market demand. This helps in refining the product portfolio to focus on more successful items.
4. **Cash Flow Improvement**: Reducing purchases of less profitable or obsolete products can free up capital. This allows the company to invest in higher-margin products, new opportunities, or other areas of the business that can drive growth.
5. **Market Trends and Demand Analysis**: The list may reveal shifts in market trends or changes in customer preferences. Understanding why certain products are no longer purchased can provide insights into evolving market demands, helping the company adapt its product strategy accordingly.
6. **Waste Reduction**: By discontinuing purchases of low-demand products, the company can reduce waste in the supply chain. This contributes to more sustainable business practices and can enhance the company's reputa n.