Select Records from the Database Based on Search Criteria

Robert Brown

Northcentral University

Databases and Business Intelligence

Dr. Martin Crossland

March 19, 2023

Select Records from the Database Based on Search Criteria

Introduction

This paper comprehensively analyzes sales data using SQL Server Management Studio Express. Data scientists often need to extract insights from data stored in databases, and SQL (Structured Query Language) is a powerful tool that enables this objective. The paper answers five critical questions related to product sales, customer orders, and geographical distribution by writing and executing SQL queries. The analysis demonstrates the effectiveness of SQL as a tool for data analysis and showcases the benefits of using SQL Server Management Studio Express or similar devices.

Overview of the Database and Tables

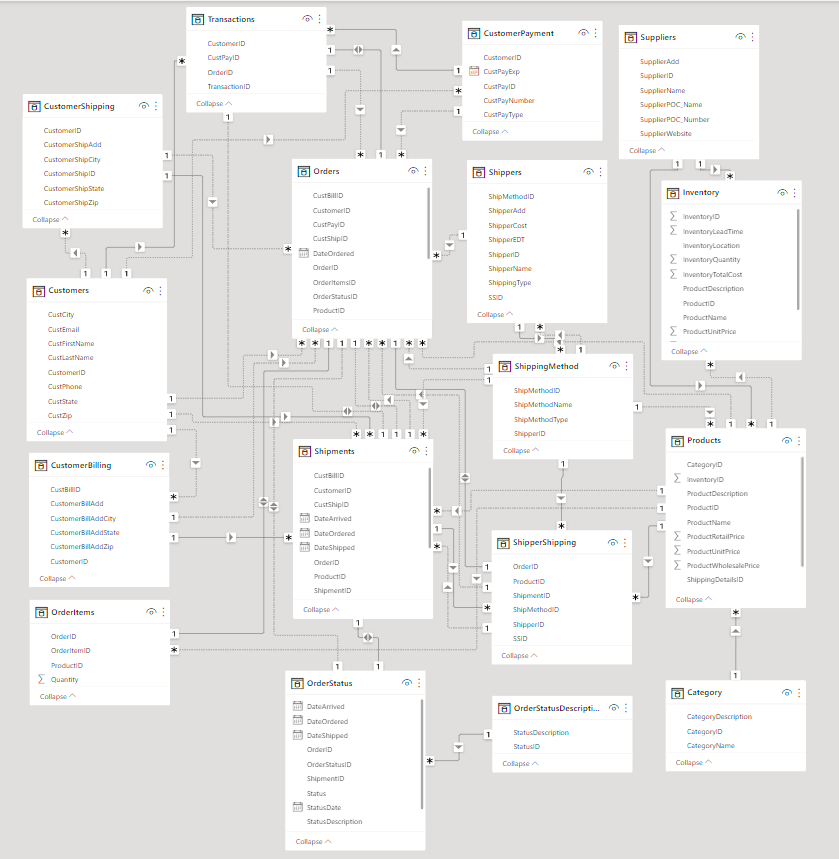
The Pet's n' Paw's database currently contains 18 tables with various data points, including ‘Products’, ‘Customers’, ‘Orders’, ‘OrderItems’, and ‘US\_Cities’. The ‘Products’ table has 11 columns and 75 sample products, while the ‘Customers’ table has eight columns and 125 sample customers. The Orders table has 17 columns and 5413 sample orders, while the ‘OrderItems’ table has four columns and 5413 sample order items.

The amount of sample data generated has allowed the project team to create satisfactory test results to demonstrate that the referential information is working, despite encountering a few gaps in the table updates. Further testing, procedure**,** and trigger creation will eliminate the issues with relational updates. One example is the ‘DateOrdered’ column in the ‘Orders’ table, which is not correctly updating across the database. Minor ‘tweaks’ to the origin and direction of data flow are needed to smooth updates across the tables.

Unfortunately, significant time was spent attempting to populate several tables, making it impossible to keep several previously linked tables together in their Primary and Foreign Key relationships. A new table named **‘**ShipperShipping**’** was created to bridge between Orders and the Shipping tables; A **‘**US\_Cities**’** table was added to allow the Customer table to reference expanded address information. The **‘**ShipperShipping**’** table provides a more efficient path to **specific** transactions and shipping information in a more streamlined manner.The Database Schema Diagram, seen in Figure 1, illustrates these updates.

Figure

Database Schema Diagram; PNP\_v9



Note. Created with Microsoft PowerBI Relationship Management feature.

Analysis

Top 5 Products by Sales

The script retrieves the top 5 selling products by their retail price and total sales for each product. The SQL commands used in the script allowed the data to be parsed from the tables.

To select the top 5 results that meet specified criteria, the source columns needed to be identified in the ‘Orders’ and ‘Product’ tables. The exact process was accomplished for each script generation. The following columns were isolated for the query; od.ProductID (od. shorthand for ‘Orders’, and p. for ‘Products’), p.ProductName, p.ProductRetailPrice, p.ProductUnitPrice, p.ProductWholesalePrice. Then a COUNT(\*) AS TotalSales: the list was applied to the columns to select from the ‘Orders’ and ‘Products’ tables. This created an alias for the COUNT(\*) function, which will calculate the total sales for each product and store it until needed.

The script then identifies from where to take data and how to put it together with a FROM Orders od JOIN ‘Products’ p ON od.ProductID = p.ProductID. This joins the ‘Orders’ and ‘Products’ tables on the ‘ProductID’ column.

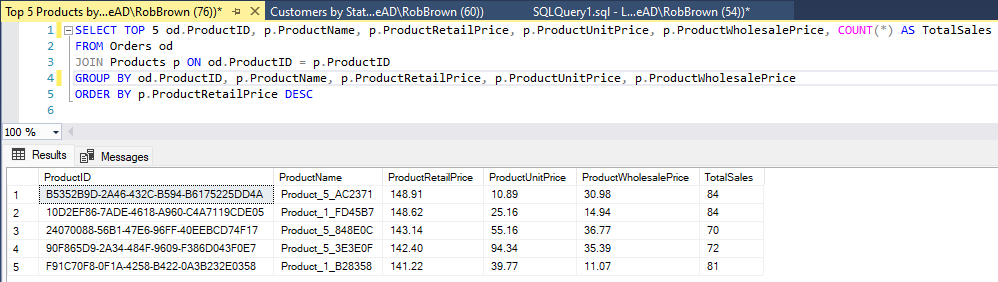
Now that the data is available, it needs to be sorted using a GROUP BY ProductID, ProductName, ProductRetailPrice p.ProductUnitPrice, and ProductWholesalePrice, function. This groups the results by ‘ProductID’, ‘ProductName’, ‘ProductRetailPrice’, ‘ProductUnitPrice’, and ‘ProductWholesalePrice’, respectively. This allows for a count of the sales for each product in each pricing category.

ORDER BY p.ProductRetailPrice DESC: sorts the results by ProductRetailPrice in descending order, so the top 5 selling ‘Products’ by their retail price are displayed first. The p.ProductUnitPrice, and p.ProductWholesalePrice, are added columns that provide fidelity to cost and profit margins. These numbers can help Sales, Marketing, and Management determine where to focus on product development and improvement.

Using JOIN to combine the ‘Orders’ and ‘Products’ tables on the ‘ProductID’ column allows retrieval of the product name and retail price for each product sold. GROUP BY is used to group the results by ‘ProductID’, ‘ProductName’, and ‘ProductRetailPrice’, so that a count of the sales for each product can be obtained. Finally, ORDER BY is used to sort the results by ProductRetailPrice in descending order so that the top 5 selling products by their retail price are displayed first. It will be easy for the different departments to parse the information for deeper analysis in this format. Additionally, the simplicity of the script allows for the adjustment of product numbers for slicing the data further. The result of the script is shown in Figure 2, a screenshot of the query result.

Figure

Top 5 Products by Sales



Note. The results and query were created in and generated by Microsoft SQL Server Management Studio, version 19.0.20200

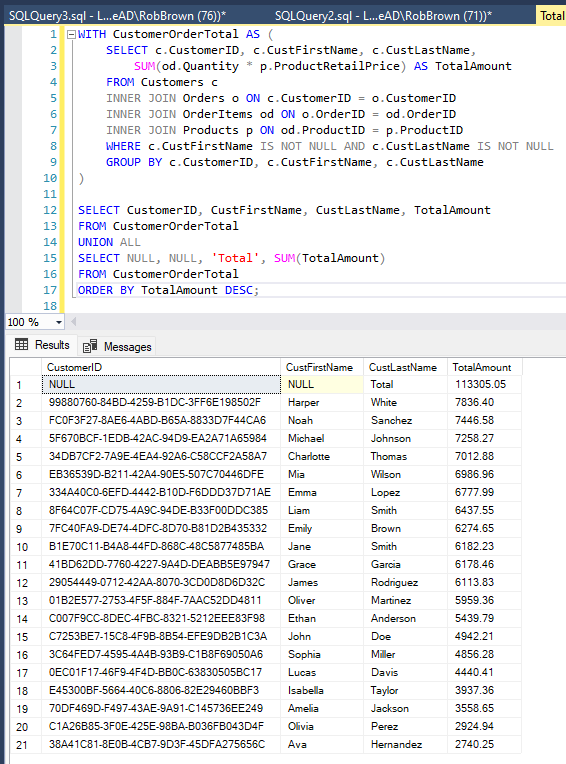
Total Orders for Each Customer

This script calculates the total amount spent by each customer on their orders and then displays the results with a total at the bottom. The query returns the top 20 customers by summoning a common table expression (CTE) called ‘CustomerOrderTotal’, uses inner joins to link the ‘Customers’, ‘Orders’, ‘OrderItems’, and ‘Products’ tables together and then calculates the total amount spent by each customer. Only customers with a first and last name and group the results by customer ID, first name, and last name.

Next, a UNION ALL operator combines the results of the CTE with a second SELECT statement that returns a row with a NULL customer ID, NULL first and last names, and a "Total" label, along with the sum of the ‘TotalAmount’ column from the CTE. This row serves as the total at the top of the results.

Figure

Total Orders for Each Customer



Note. The results and query were created in and generated by Microsoft SQL Server Management Studio, version 19.0.20200

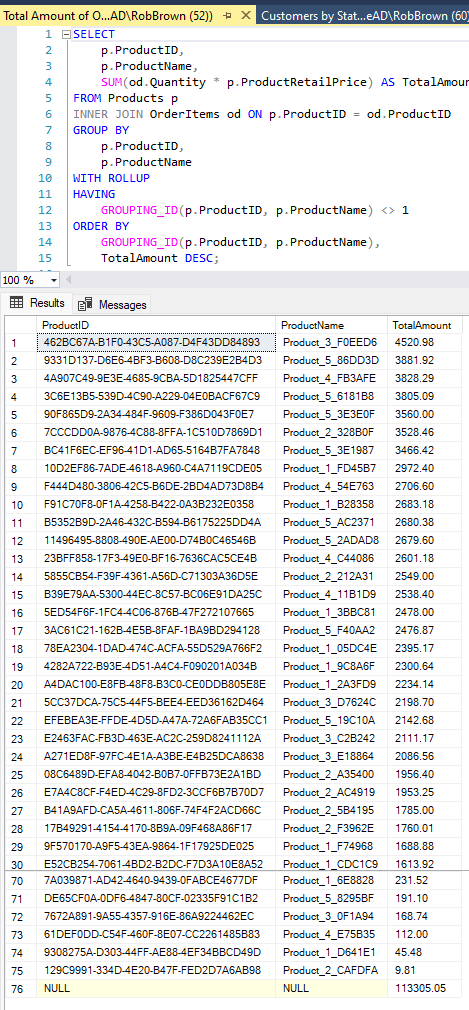
Total Amount of Orders Associated with Each Product

This script is designed to calculate the total amount of sales for each product in the ‘Products’ table. The ‘ProductID’ and ‘ProductName’ columns were selected from the ‘Products’ table and, using an inner join; it was linked with the ‘OrderItems’ table. We then calculate the total amount of sales for each product by multiplying the ‘Quantity’ and ‘ProductRetailPrice’ columns and summing the results. The results are grouped by ‘ProductID’ and ‘ProductName’.

To add a total at the bottom of the results, we use the ROLLUP function, which generates subtotals and a total for the results. The HAVING clause filtered out the total row with a GROUPING\_ID value of 1. Finally, the results were sorted by the GROUPING\_ID value in ascending order, followed by the TotalAmount value in descending order. This places the total row at the bottom of the results. This script provides a snapshot of the total sales for each product and the total for all products. The results of this can be found in Figure 4.

Figure

Total Amount of Orders Associated with Each Product



Note. Items 31 to 69 in the results were truncated from the image for readability. The results and query were created and generated by Microsoft SQL Server Management Studio, version 19.0.20200.

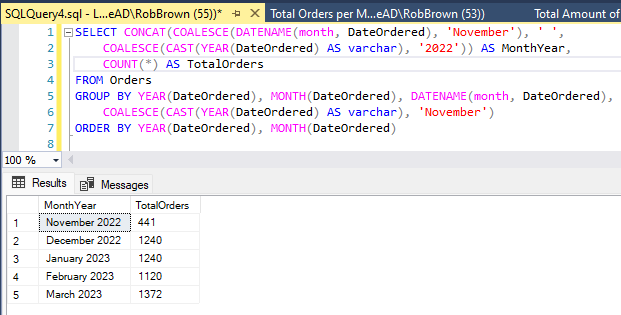
Total Count of Orders per Month by Year

This SQL script retrieves the total number of orders made for each month and year combination from the "Orders" table. The script uses the CONCAT function to combine the month name and year into a single column and the COALESCE function to replace NULL values with "November" for the month and "2022" for the year. The script then uses the GROUP BY clause to group the orders by year, month, and month name and the COALESCE function to group NULL values as November for the month. This was an oversight during the creation of the dataset; when the ‘DateOrdered’ was initially added to the data set, there were only 441 orders. During a later iteration of Sample Data, an additional 4,873 ‘Orders’ were created. Other tables had to be bolstered to maintain data integrity due to the drastic increase in sample sales data. Finally, the script orders the results yearly and monthly in ascending order, starting in November.

Overall, this script allows us to quickly see the total number of orders made for each month and year combination in a concise and organized manner. The CONCAT and COALESCE functions help ensure the results are clear and accurate, even when dealing with NULL values in the original data. The results and code can be seen in Figure 5.

Figure

Total Count of Orders per Month & Year



Note. The results and query were created and generated by Microsoft SQL Server Management Studio, version 19.0.20200.

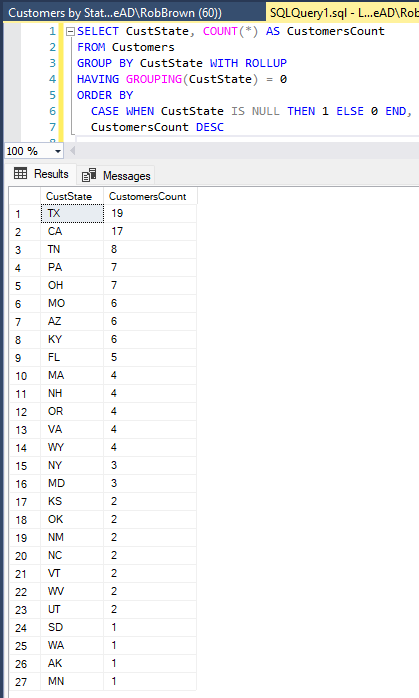
Total Count of Customers per State

The above SQL script calculates the total number of customers per state. The ‘CustState’ column was used to count the number of customers in each state using a COUNT(\*) function. The results were then grouped by state using the GROUP BY clause. From there, a WITH ROLLUP clause to include a subtotal row for each state would typically generate a total at the bottom of the results. In this case, it was felt that a total was not needed. This query intends to derive the states in which most customers live. With that, a HAVING clause filters out the subtotal values generated by the ROLLUP operation to only display the actual state counts and total.

Finally, an ORDER BY clause sorted the results in descending order by the number of customers in each state, with the highest value at the top. A CASE statement was inserted to ensure the total row is always displayed at the bottom. Overall, this SQL script is clear and concise and provides the total number of customers per state in a sorted and organized manner.

Figure

Total Customers per State



Note. The results and query were created and generated by Microsoft SQL Server Management Studio, version 19.0.20200.

Conclusion

The analysis conducted in this paper highlights the potential of SQL Server Management Studio Express, and like tools, for extracting valuable insights from sales data. Albeit simple scripts with a sample data set, the results better understand product sales, customer orders, and geographical distribution by addressing five critical questions through SQL queries. The screenshots and SQL scripts with explanations serve as valuable resources for further analysis and showcase the versatility and effectiveness of SQL for data analysis. This paper aims to highlight the power of SQL in answering complex questions and guiding data-driven decision-making in business and research.