Written by

Ahmed Nassr Mohamed  
Islam Eltayeb Mohamed   
Lara Saad Hessuin   
Abdelrahman Ibrahim Attia

DEPI final project

Customer Data Project

Group Code: ONL1\_AIS4\_G2e

**SQL Introduction:**

SQL, or Structured Query Language, is a powerful programming language specifically designed for managing and manipulating relational databases. It serves as the standard means for querying, updating, and managing data stored in a structured format, allowing users to perform a wide range of operations, from simple data retrieval to complex transactions. SQL's declarative syntax enables users to specify what data they want to retrieve or manipulate without detailing how to accomplish these tasks, making it both intuitive and efficient. Widely adopted across various database systems, including MySQL, PostgreSQL, Microsoft SQL Server, and Oracle, SQL is essential for data analysts, developers, and database administrators. Its versatility and robustness make it a cornerstone of modern data management and a critical skill in the ever-growing field of data science and analytics.

import flat file

In this step, all files related to the project were imported

using database tools, such as Import Flat File. Imported

files include:

Customer\_Data.csv

Order\_Data.csv

Interaction\_Data.csv

Order\_Products.csv

Product\_Data.csv

Product\_Transactions1.csv

Transaction\_Data.csv

A screenshot of a computer

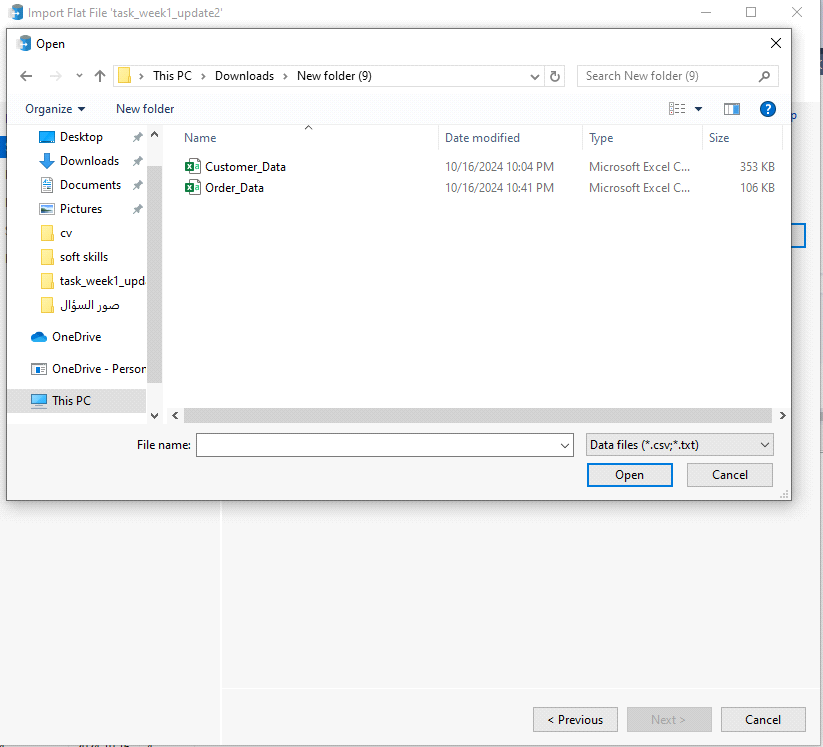
Description automatically generated

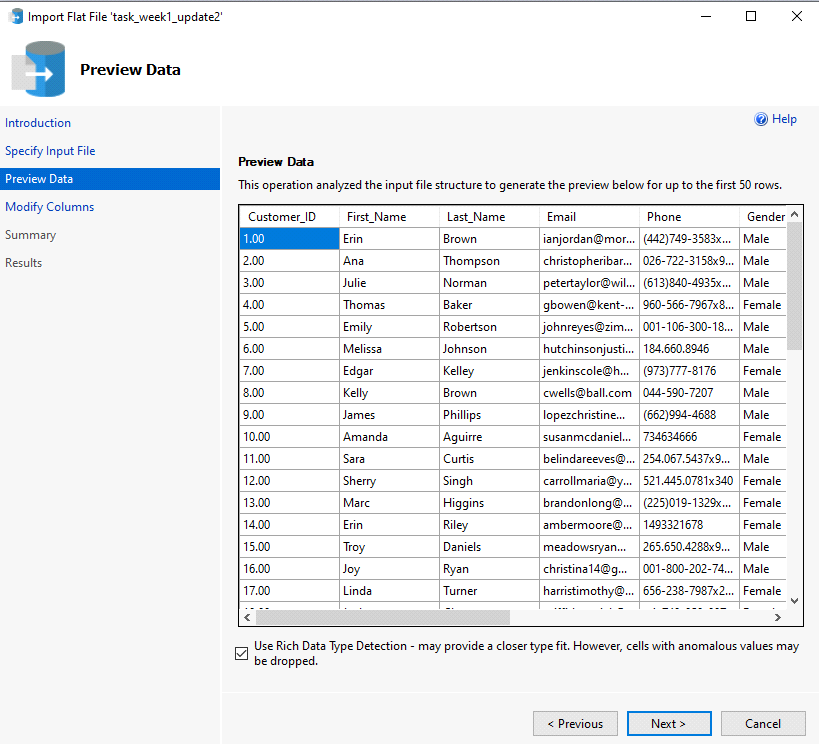
A screenshot of a computer

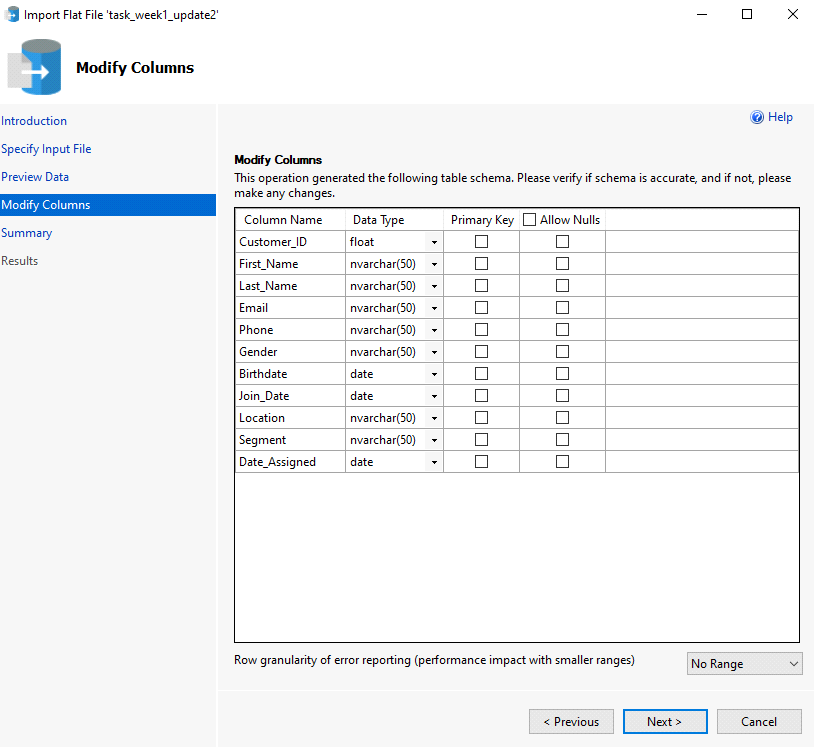
Description automatically generated

A screenshot of a computer

Description automatically generated







A screenshot of a computer

Description automatically generated

**ERD MODEL**

**what is ERD?**

An Entity-Relationship Diagram (ERD) is a visual representation of the structure of a database, showing the entities (tables), their attributes (columns), and the relationships between them.

**relationship between tables**

1)one to many between customer and interactions

A close-up of a computer code

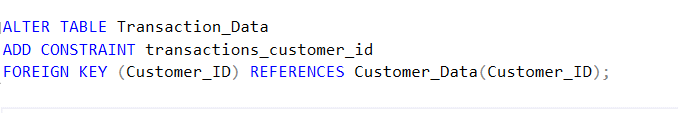
Description automatically generated

2)one to many between customer and Orders

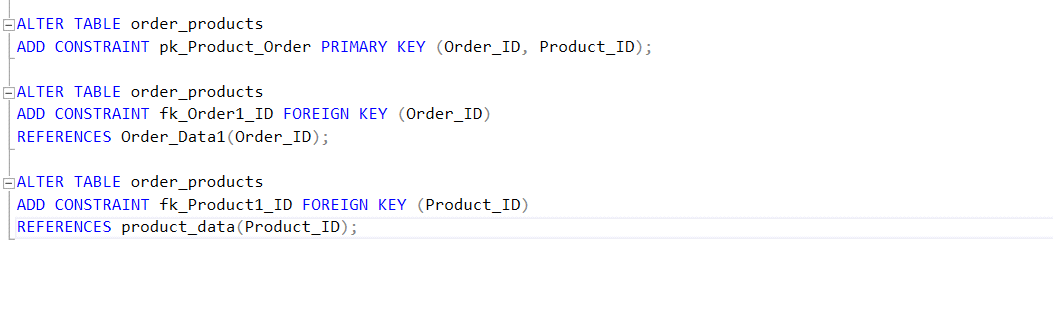
A computer screen shot of a computer

Description automatically generated

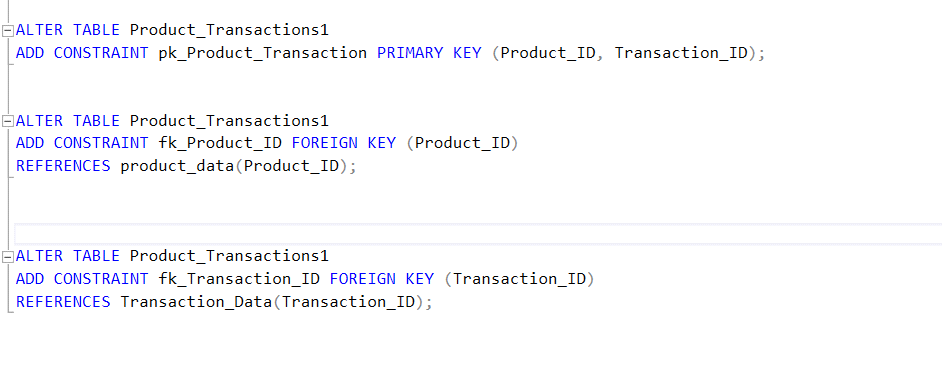
3)one to many between customer and Transaction



4)many to many between products and Orders



5)many to many between products and Transaction



6)one to one between Transaction and Orders

A white background with black text

Description automatically generated

**Alteration**

**what is alter ?**

The ALTER statement in SQL is used to modify the structure of an existing database table, such as adding, deleting, or modifying columns or constraints

1)we set customer id as primary key and change null value to not null

A screen shot of a computer

Description automatically generated

2)we change datatype customer from float to int

A white background with yellow border

Description automatically generated

3)we change datatype interaction\_id as primary key

A white background with black dots

Description automatically generated

**SQL query**

To update the email address of the customer with a Customer\_ID of 1, we made this query,

A screen shot of a computer

Description automatically generated

To update the quantity for the order with order\_id 3000001, we made this query,

A black text on a white background

Description automatically generated

To delete the order with order\_id 3000005, we made this query,

A black text on a white background

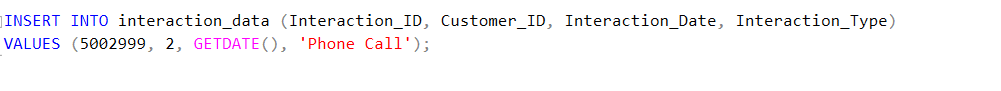
Description automatically generated

To delete the customer with Customer\_ID 3, we made this query,

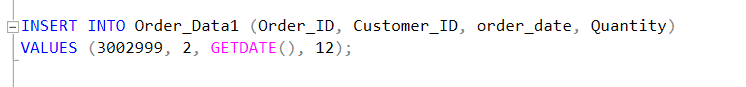
A black text on a white background

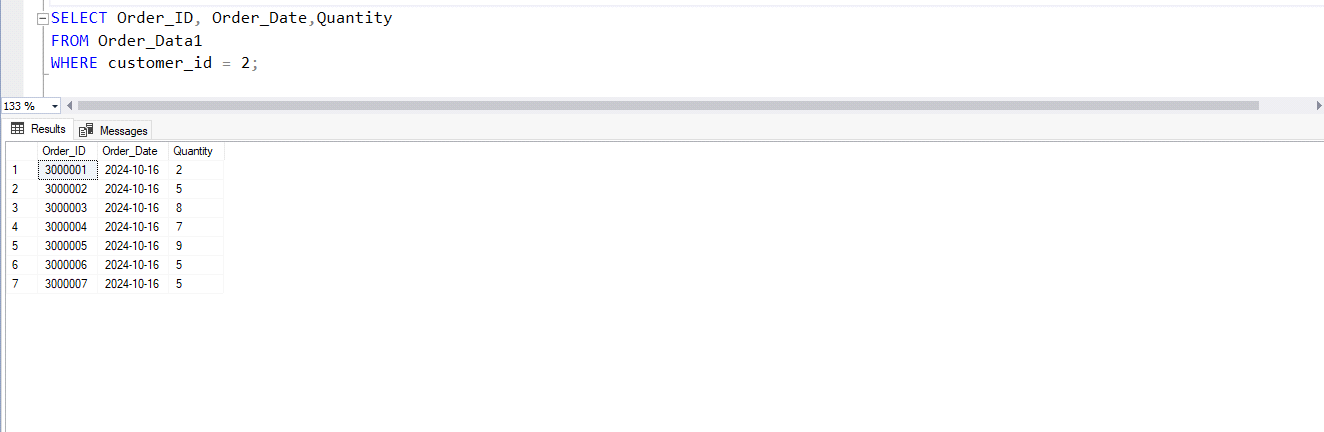
Description automatically generated

To insert a new interaction record into the interaction\_data table, we made this query,



To insert a new order record into the Order\_Data table, we made this query,

  
1)To get the order details (Order\_ID, Order\_Date, and Quantity) for the customer with ID 2, we made this query.



2)To get the number of orders placed by each customer, we made this query,

A screenshot of a computer

Description automatically generated

To retrieve the customers who have placed more than 3 orders, we made this query,

A long white object with a white background

Description automatically generated with medium confidence

To retrieve the product names associated with transaction ID 4000000, we made this query,

A screenshot of a computer

Description automatically generated

To get all the orders placed within the last 30 days, we made this query,

A screenshot of a computer

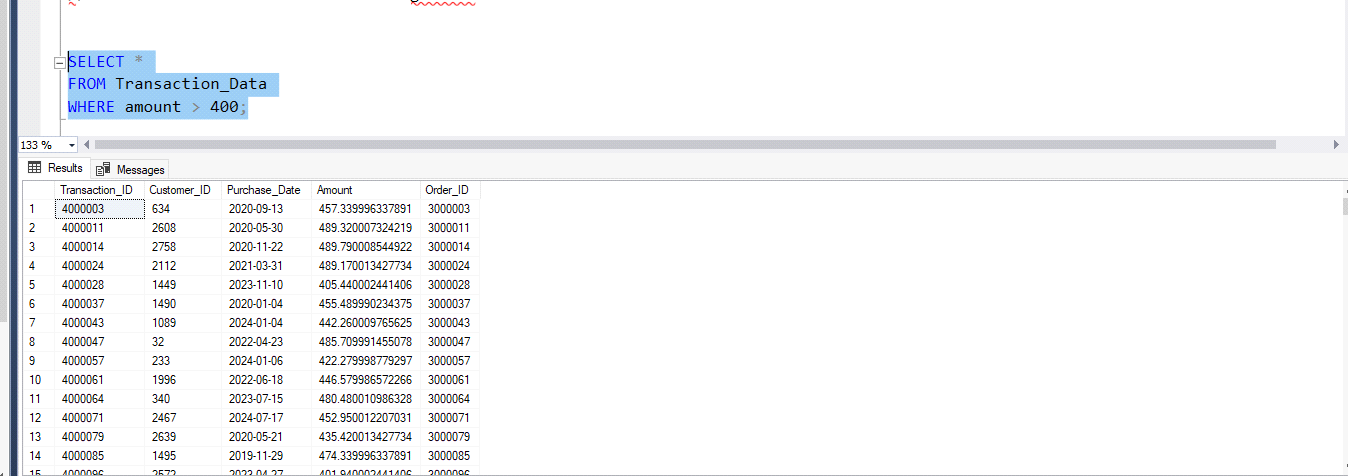
Description automatically generated

To retrieve all interactions that occurred within the last 30 days, we made this query,

A computer screen with a blue and purple text

Description automatically generated

To retrieve all transactions with an amount greater than 400, we made this query,



To get the customers who either have not placed any orders or whose last order was more than six months ago, we made this query,

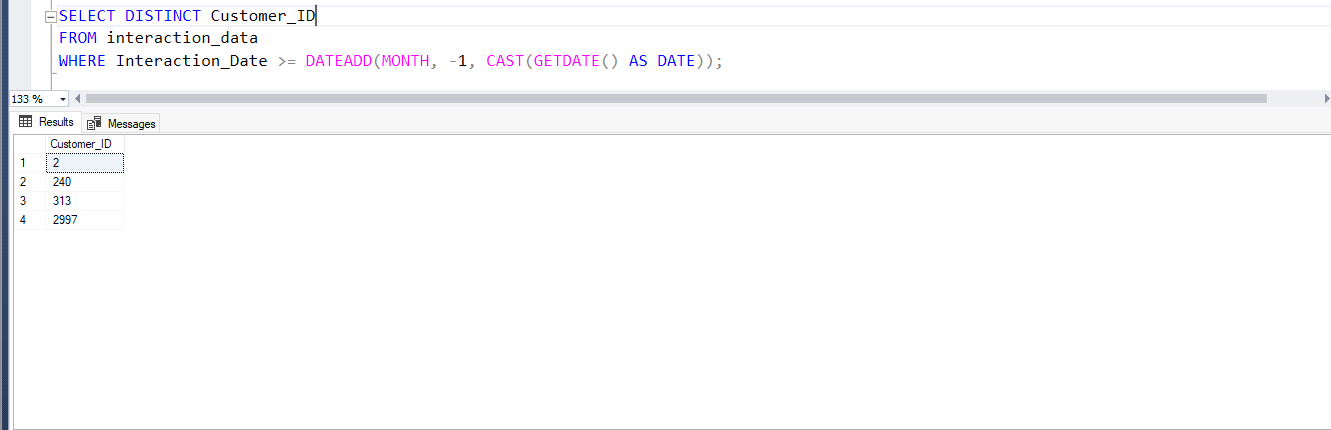
A screen shot of a computer

Description automatically generated

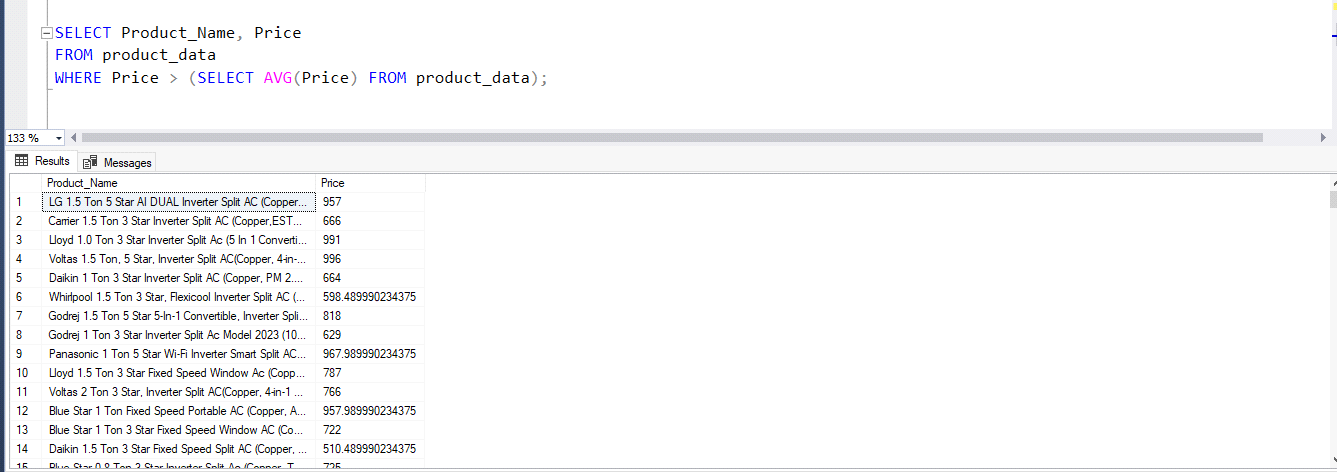
To get the most recent interaction date for each customer, we made this query,



To retrieve the distinct customer IDs of customers who have had interactions within the last month, we made this query,



To retrieve the products that are priced above the average price, we made this query,

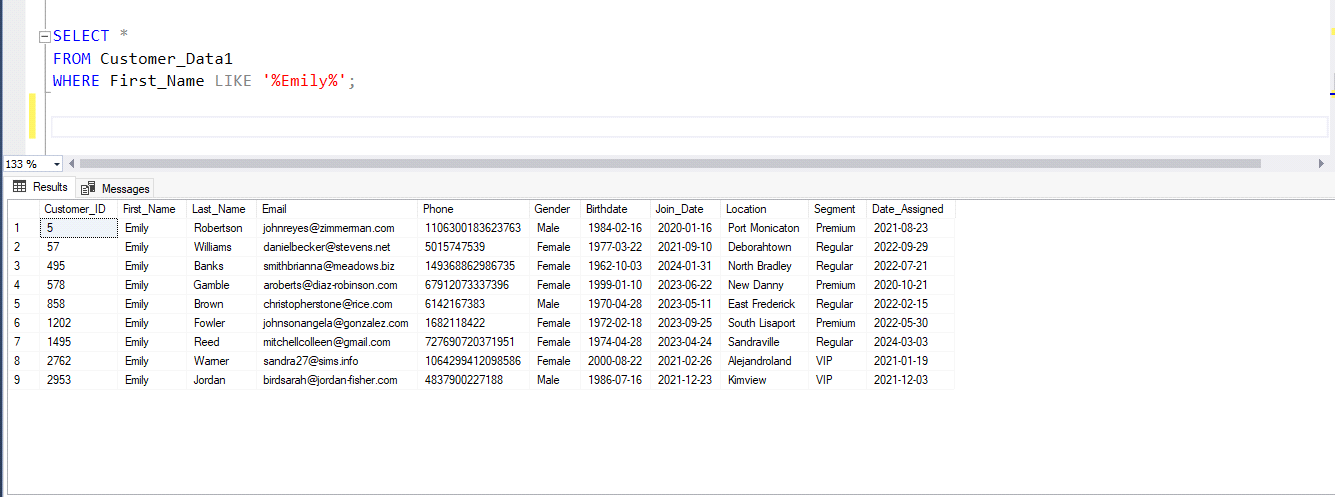


To calculate the average quantity of items per order, we made this query,

A screenshot of a computer

Description automatically generated

To retrieve all customer records where the first name contains "Emily," we made this query,



To retrieve the customer ID along with a concatenation of the first and last names separated by a hyphen, we made this query,

A screen shot of a computer

Description automatically generated

To retrieve the customer ID along with a concatenation of the first letter of the first name and the last letter of the last name, separated by a hyphen, we made this query,

A screenshot of a computer

Description automatically generated

import flat file

In this step, all files related to the project were imported

using database tools, such as Import Flat File. Imported

files include:

Customer\_Data.csv

Order\_Data.csv

Interaction\_Data.csv

Order\_Products.csv

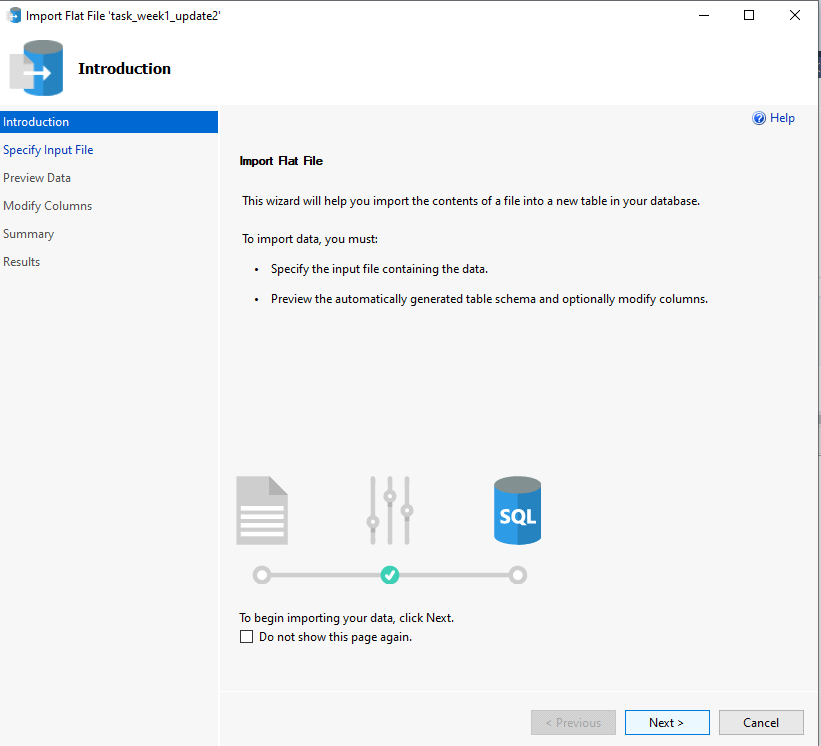
Product\_Data.csv

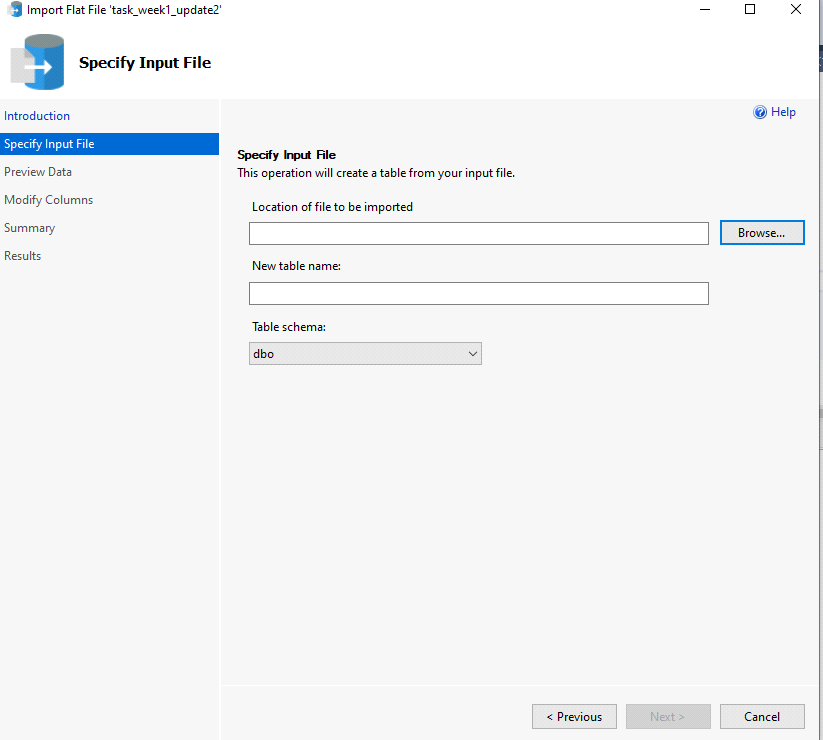
Product\_Transactions1.csv

Transaction\_Data.csv

A screenshot of a computer

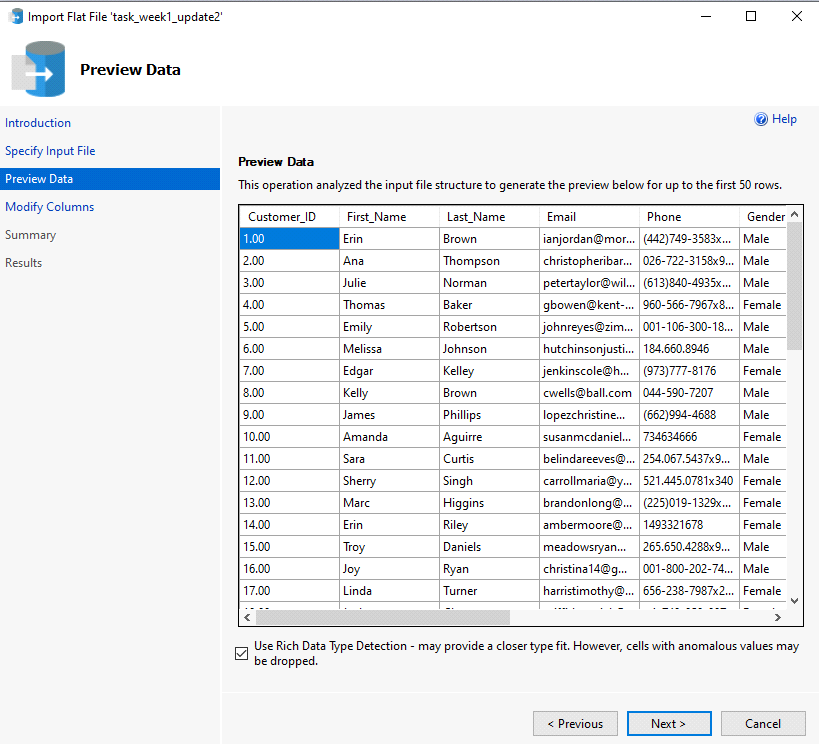
Description automatically generated

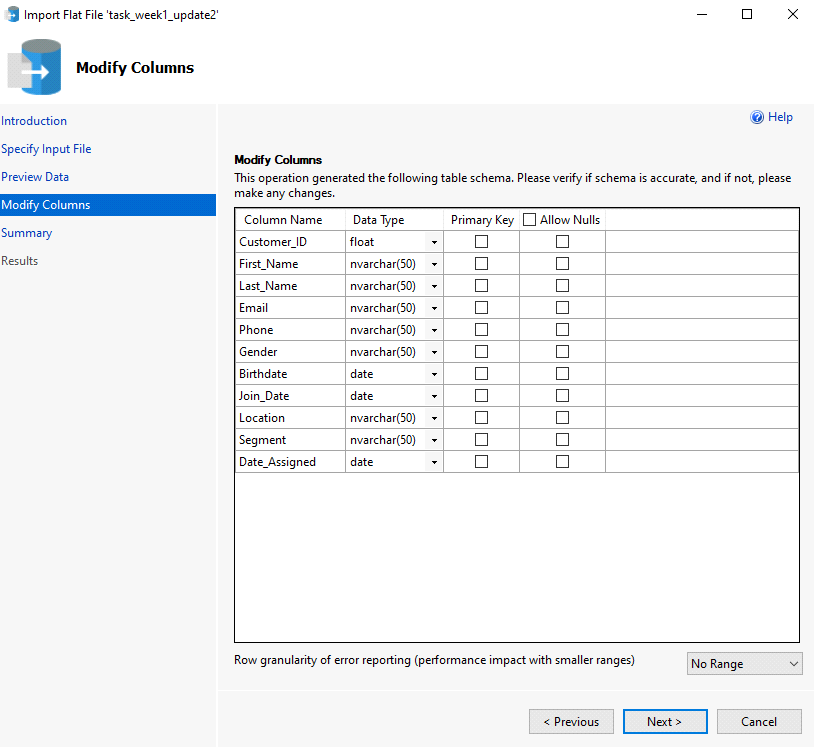


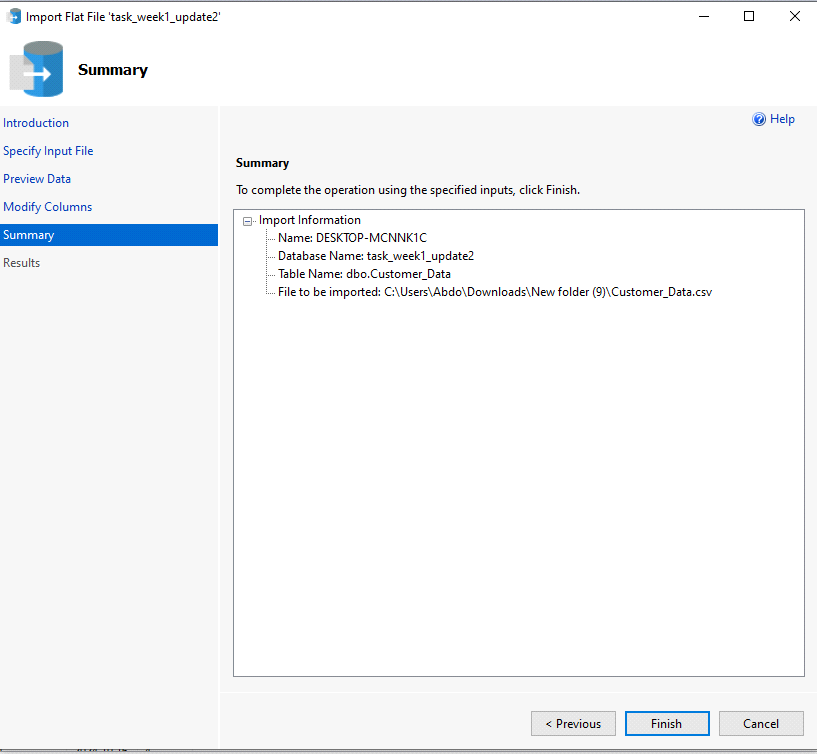


A screenshot of a computer

Description automatically generated







**ERD MODEL**

**what is ERD?**

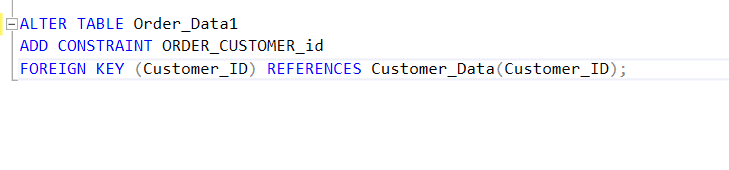
An Entity-Relationship Diagram (ERD) is a visual representation of the structure of a database, showing the entities (tables), their attributes (columns), and the relationships between them.

**relationship between tables**

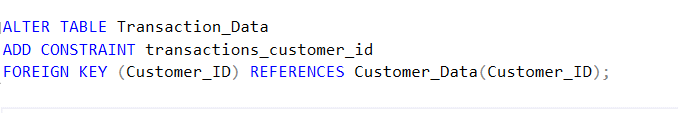
1)one to many between customer and interactions



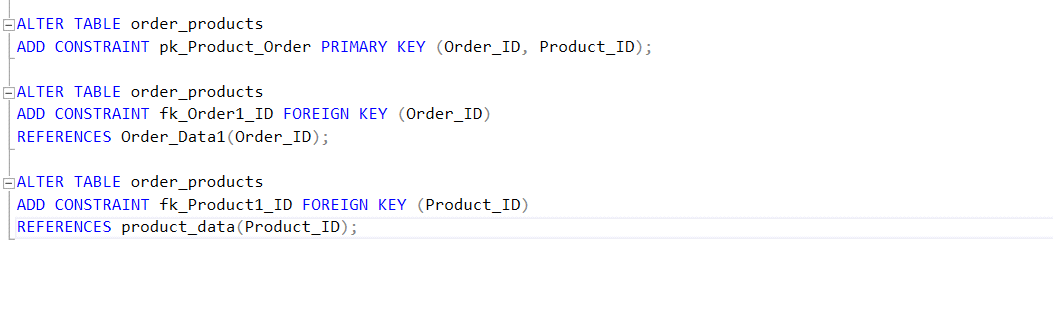
2)one to many between customer and Orders



3)one to many between customer and Transaction



4)many to many between products and Orders



5)many to many between products and Transaction

A screenshot of a computer code

Description automatically generated

6)one to one between Transaction and Orders

A white background with black text

Description automatically generated

**Alteration**

**what is alter ?**

The ALTER statement in SQL is used to modify the structure of an existing database table, such as adding, deleting, or modifying columns or constraints

1)we set customer id as primary key and change null value to not null

A screen shot of a computer

Description automatically generated

2)we change datatype customer from float to int

A white background with yellow border

Description automatically generated

3)we change datatype interaction\_id as primary key

A white background with black dots

Description automatically generated

**sql query**

To update the email address of the customer with a Customer\_ID of 1, we made this query,

A screen shot of a computer

Description automatically generated

To update the quantity for the order with order\_id 3000001, we made this query,

A black text on a white background

Description automatically generated

To delete the order with order\_id 3000005, we made this query,

A black text on a white background

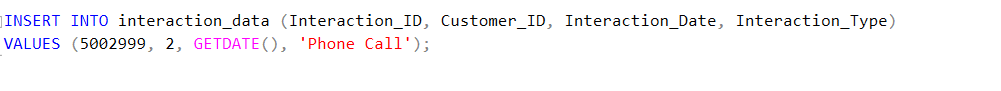
Description automatically generated

To delete the customer with Customer\_ID 3, we made this query,

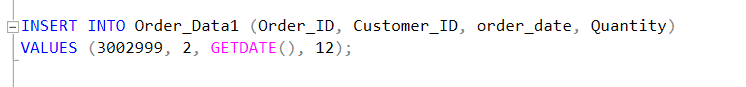
A black text on a white background

Description automatically generated

To insert a new interaction record into the interaction\_data table, we made this query,



To insert a new order record into the Order\_Data table, we made this query,



1)To get the order details (Order\_ID, Order\_Date, and Quantity) for the customer with ID 2, we made this query.

A white rectangular object with a long rectangular object

Description automatically generated

2)To get the number of orders placed by each customer, we made this query,

A screenshot of a computer

Description automatically generated

To retrieve the customers who have placed more than 3 orders, we made this query,

A long white object with a white background

Description automatically generated with medium confidence

To retrieve the product names associated with transaction ID 4000000, we made this query,

A screenshot of a computer

Description automatically generated

To get all the orders placed within the last 30 days, we made this query,

A screenshot of a computer

Description automatically generated

To retrieve all interactions that occurred within the last 30 days, we made this query,

A computer screen with a blue and purple text

Description automatically generated

To retrieve all transactions with an amount greater than 400, we made this query,

A screenshot of a computer

Description automatically generated

To get the customers who either have not placed any orders or whose last order was more than six months ago, we made this query,

A screen shot of a computer

Description automatically generated

To get the most recent interaction date for each customer, we made this query,

A screenshot of a computer

Description automatically generated

To retrieve the distinct customer IDs of customers who have had interactions within the last month, we made this query,

A screen shot of a computer

Description automatically generated

To retrieve the products that are priced above the average price, we made this query,

A screen shot of a computer

Description automatically generated

To calculate the average quantity of items per order, we made this query,

A screenshot of a computer

Description automatically generated

To retrieve all customer records where the first name contains "Emily," we made this query,

A screen shot of a computer

Description automatically generated

To retrieve the customer ID along with a concatenation of the first and last names separated by a hyphen, we made this query,

A screen shot of a computer

Description automatically generated

To retrieve the customer ID along with a concatenation of the first letter of the first name and the last letter of the last name, separated by a hyphen, we made this query,

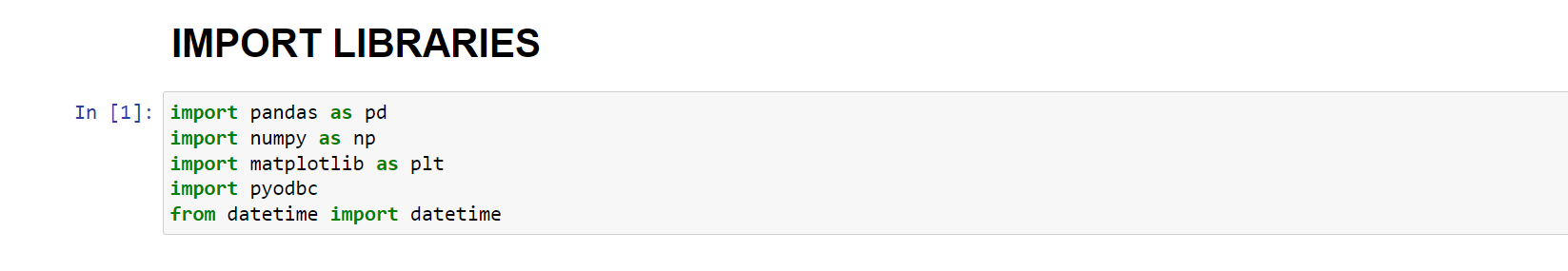
A screenshot of a computer

Description automatically generated

**Introduction to Pandas**

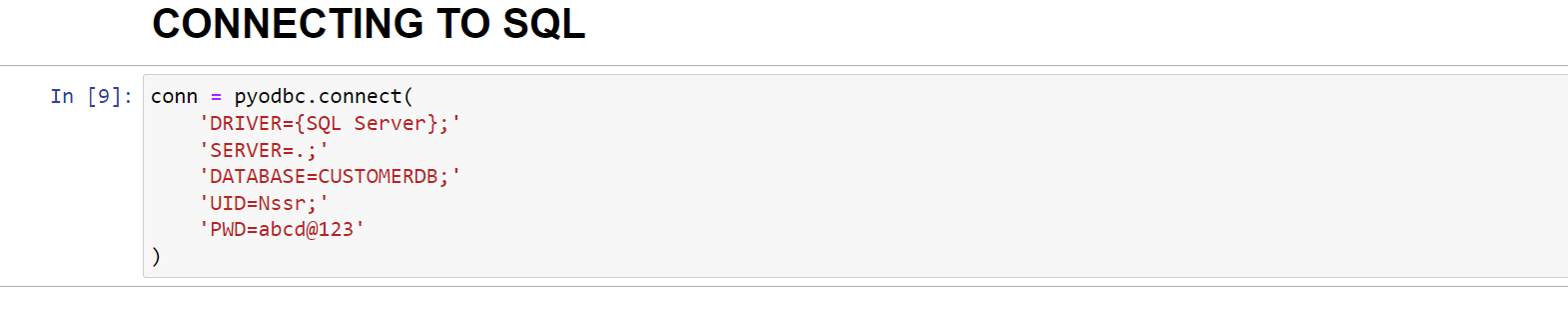
Pandas is a powerful and versatile open-source data analysis and manipulation library for Python. It provides data structures like Series and DataFrame, which facilitate efficient handling of structured data. Originally developed by Wes McKinney in 2008, Pandas has become an essential tool for data scientists and analysts due to its ability to simplify complex data operations, including data cleaning, transformation, and analysis. With features such as intuitive indexing, powerful data alignment, and easy handling of missing data, Pandas enables users to perform a wide range of tasks, from exploratory data analysis to advanced statistical modeling. Its integration with other scientific libraries like NumPy and Matplotlib further enhances its functionality, making it a cornerstone of the Python data science ecosystem.

**Import Libraries:**

****

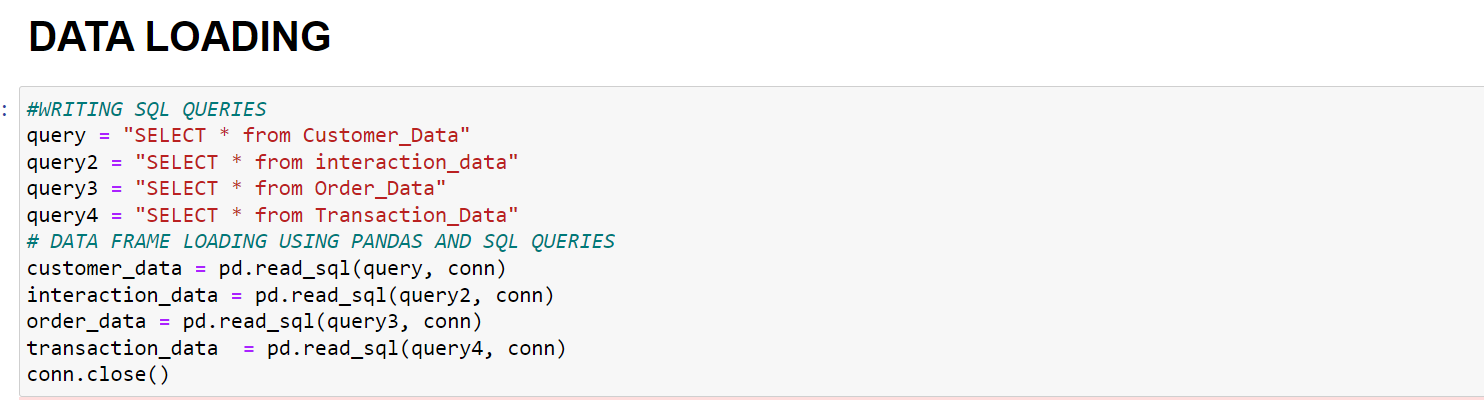
This code snippet imports several essential libraries used for data manipulation, analysis, and visualization in Python. The pandas library, imported as pd, is a powerful tool for data analysis, providing data structures like DataFrames that facilitate the handling of structured data. The numpy library, imported as np, offers support for numerical operations and array manipulations, which are often used alongside pandas for efficient data processing. The matplotlib library, imported incorrectly as plt (it should be import matplotlib.pyplot as plt), is a widely-used library for creating static, animated, and interactive visualizations in Python. The pyodbc library allows for database connectivity, enabling interactions with databases through SQL queries. Lastly, the datetime module is imported to work with date and time objects, making it easier to handle time series data and perform date-related calculations.

**Connecting to SQL:**



This code snippet establishes a connection to a SQL Server database using the pyodbc library, which enables Python to interact with databases through ODBC (Open Database Connectivity). The connect function is called with a connection string that specifies various parameters needed for the connection, including the ODBC driver set to SQL Server, the server location indicated by . for the local machine, and the target database named CUSTOMERDB. Additionally, it includes authentication details with the username Nssr and the password abcd@123. Executing this code attempts to connect to the specified database, allowing for subsequent database operations such as querying or updating records.

Data Loading :



This code snippet defines several SQL queries to retrieve data from different tables in a database and then loads that data into pandas DataFrames for further analysis. Four SQL queries are created: query, query2, query3, and query4, each selecting all records from their respective tables—Customer\_Data, interaction\_data, Order\_Data, and Transaction\_Data. Using the pd.read\_sql() function, each query is executed against the established database connection (conn), and the resulting data is loaded into the pandas DataFrames named customer\_data, interaction\_data, order\_data, and transaction\_data. After the data is successfully retrieved and stored in these DataFrames, the connection to the database is closed with conn.close(), ensuring that resources are properly released.

**Data Analyzing :**

**Customer-Interaction Analysis:**

A screenshot of a computer code

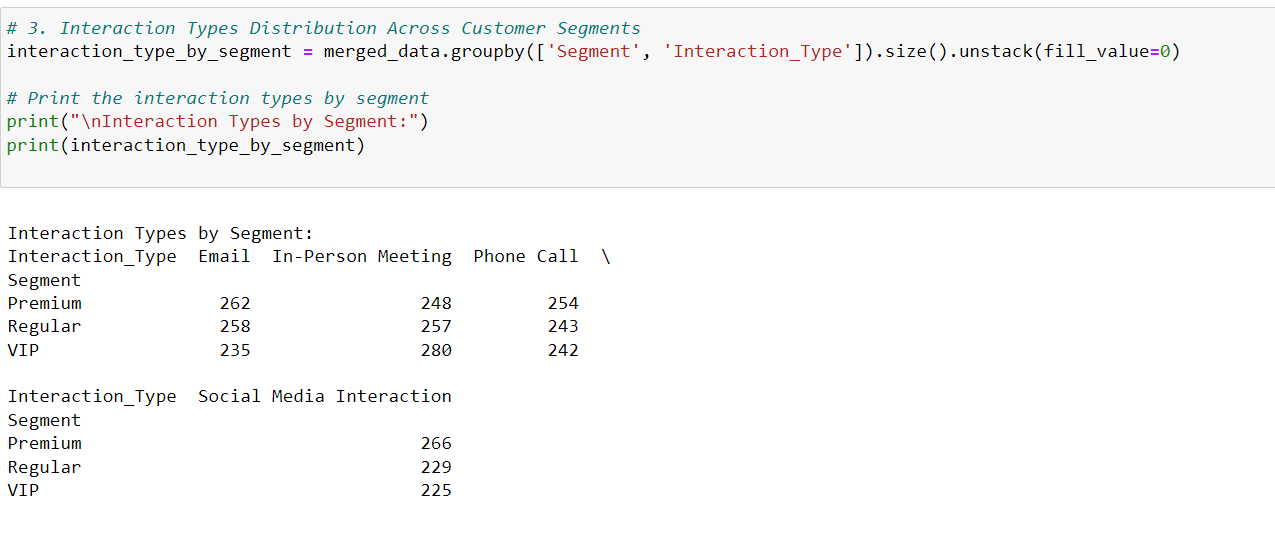
Description automatically generated

This code snippet performs data preprocessing and analysis on the customer\_data DataFrame, specifically focusing on date conversions and age calculation. First, it converts three date columns—Join\_Date, Date\_Assigned, and Birthdate—into pandas datetime objects using pd.to\_datetime(). The errors='coerce' argument ensures that any invalid dates are converted to NaT (Not a Time) rather than raising an error. Next, the code calculates the age of each customer by subtracting their Birthdate from the current date (using pd.Timestamp.now()), resulting in a time difference expressed in days. This difference is then divided by 365.25 to account for leap years, yielding the age in years, which is stored in a new column named Age. The code concludes by generating descriptive statistics for the Age column using the describe() method, which summarizes key metrics such as count, mean, standard deviation, minimum, and quartiles. Finally, it prints the age analysis to the console, providing insights into the age distribution of customers.

A screenshot of a computer code

Description automatically generated

This code snippet analyzes the interaction frequency of customers by their segments. It begins by merging the interaction\_data DataFrame with the customer\_data DataFrame based on the common column Customer\_ID. This merge is performed using an inner join, meaning only records that have matching Customer\_IDs in both DataFrames will be included in the resulting merged\_data DataFrame. Next, the code groups the merged data by the Segment column and counts the number of interactions for each segment by using the count() method on the Interaction\_ID column. This results in a Series that reflects the frequency of interactions categorized by customer segment. Finally, the interaction frequency by segment is printed to the console, providing insights into how often different customer segments engage with the business.



This code snippet examines the distribution of interaction types across different customer segments. It groups the merged\_data DataFrame by both the Segment and Interaction\_Type columns, using the size() method to count the occurrences of each combination. The result is then reshaped using the unstack() method, which transforms the grouped data into a more readable format, with customer segments as rows and interaction types as columns. The fill\_value=0 argument ensures that any missing combinations of segment and interaction type are filled with zeros, indicating no interactions occurred for those combinations. Finally, the distribution of interaction types by segment is printed to the console, providing a clear overview of how different customer segments engage through various types of interactions.

**Customer – Order analysis:**

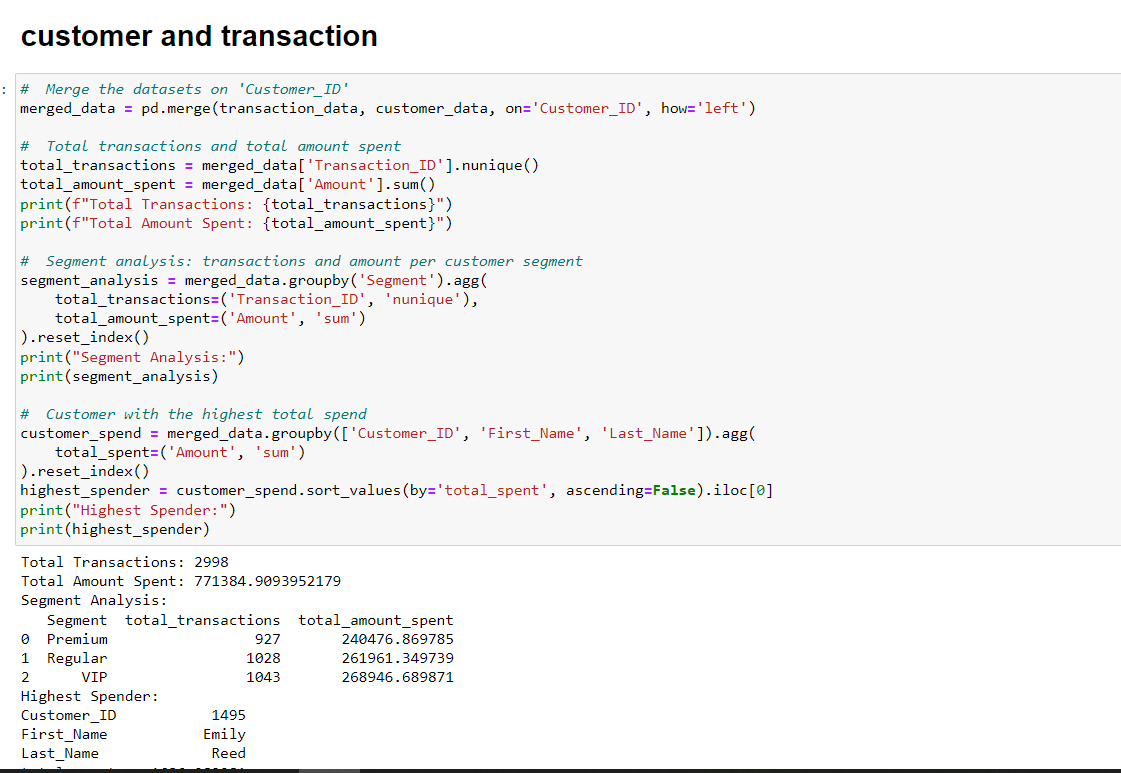


This code snippet performs a series of analyses on customer and order data, focusing on date parsing, summary statistics, and trend analysis. Initially, it converts date columns in the customer\_data and order\_data DataFrames into pandas datetime objects using pd.to\_datetime(), allowing for effective date-based operations and ensuring that any invalid dates are handled gracefully with errors='coerce'. The first part of the code generates basic summary statistics for both datasets using the describe() method, with include='all' for the customer data to capture all data types. These summaries, which include metrics such as count, unique values, mean, and standard deviation, are printed to the console for review.

Next, the code analyzes order trends over time by grouping the order\_data by month, leveraging the dt.to\_period('M') function to create a period representation of the order dates. The size of each group is calculated, resulting in a Series that reflects the number of orders per month, which is then printed.

Lastly, the code merges the order\_data with customer\_data on the Customer\_ID field using an inner join, allowing for the examination of orders in relation to customer segments. It groups the merged data by the Segment column and counts the total number of orders per segment, which is also printed, providing valuable insights into customer purchasing behavior across different segments.

**Customer-Transaction Analysis:**

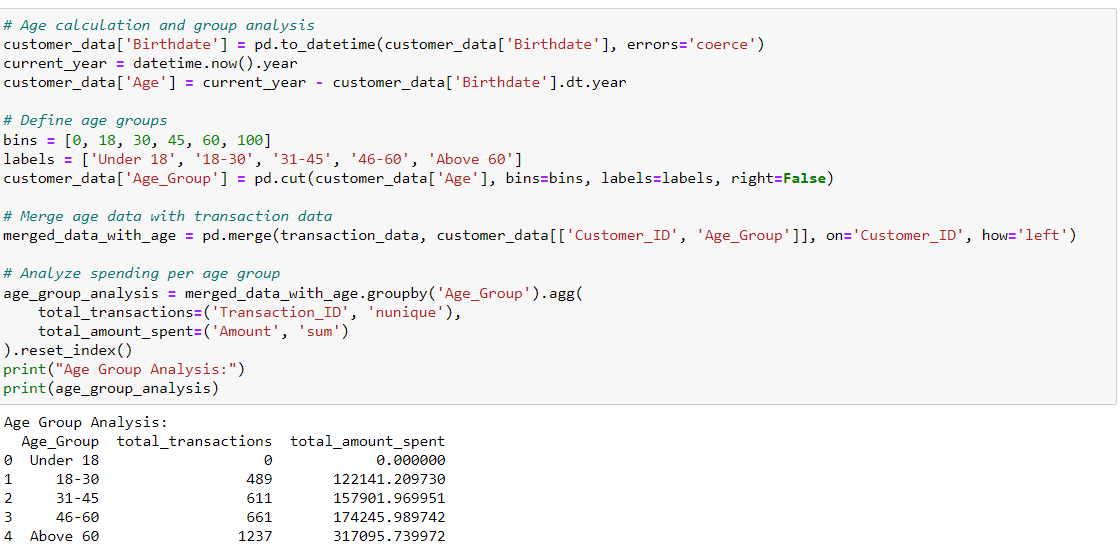


This code snippet analyzes transaction data in relation to customer data, focusing on total transactions, amounts spent, segment analysis, and identifying the highest spender. It begins by merging the transaction\_data DataFrame with the customer\_data DataFrame on the Customer\_ID field using a left join, ensuring that all records from the transaction data are preserved while matching with customer information where available.

Next, the code calculates the total number of unique transactions by counting distinct Transaction\_IDs and the total amount spent by summing the Amount column. These totals are printed to the console, providing a quick overview of transaction activity.

The analysis then extends to customer segments by grouping the merged data by the Segment column. It employs the agg() function to calculate the total number of transactions and the total amount spent for each segment, generating a summarized DataFrame that is printed for review.

Finally, the code identifies the customer with the highest total spend by grouping the merged data by Customer\_ID, First\_Name, and Last\_Name, and aggregating the total spent using the sum() function. The results are sorted in descending order based on total spend, and the customer with the highest total spend is extracted and printed, providing insights into customer purchasing behavior.

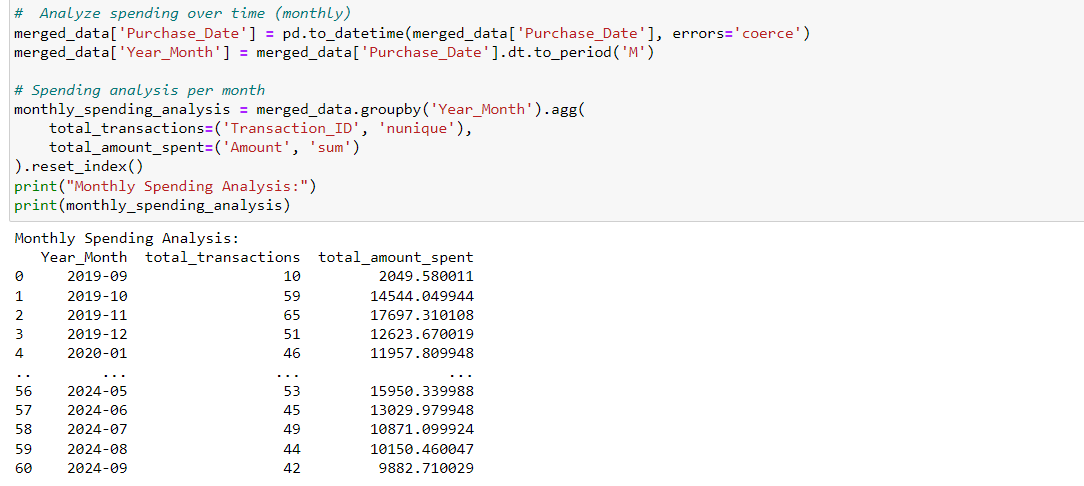


This code snippet focuses on calculating customer ages, categorizing them into age groups, and analyzing spending patterns based on these groups. It starts by converting the Birthdate column in the customer\_data DataFrame into pandas datetime objects to ensure accurate date handling. The current year is retrieved using datetime.now().year, and the age of each customer is calculated by subtracting the birth year from the current year.

Next, the code defines age groups using the pd.cut() function, creating bins that categorize customers into five groups: 'Under 18', '18-30', '31-45', '46-60', and 'Above 60'. This categorization helps in analyzing spending behavior across different age ranges.

Following this, the code merges the age group information with the transaction\_data DataFrame using a left join on the Customer\_ID field. This results in a new DataFrame, merged\_data\_with\_age, which includes both transaction data and corresponding age group data.

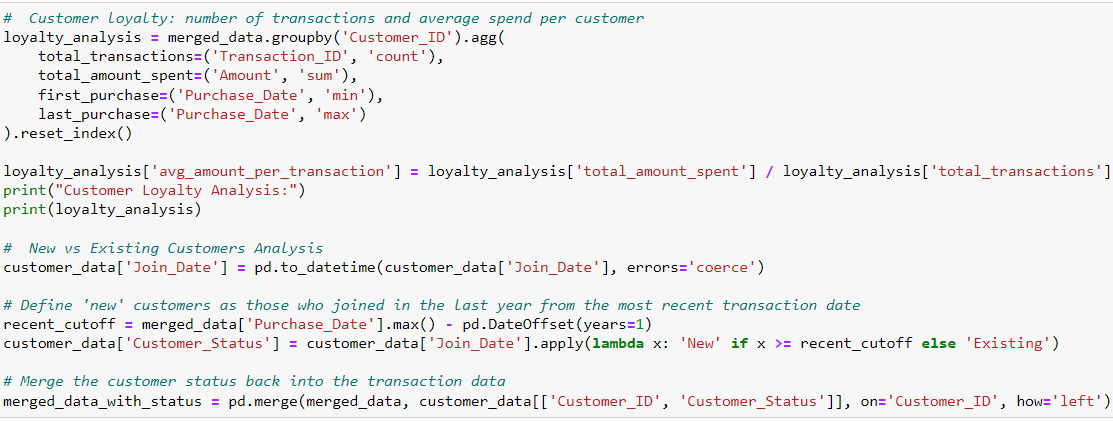
Finally, the code performs an analysis of spending per age group by grouping the merged data by the Age\_Group column. It calculates the total number of unique transactions and the total amount spent for each age group using the agg() function, generating a summarized DataFrame that highlights spending behavior across different age categories. This analysis is then printed to the console, providing insights into how spending varies by age group.



This code snippet analyzes customer spending over time on a monthly basis. It begins by converting the Purchase\_Date column in the merged\_data DataFrame into pandas datetime objects, ensuring accurate date handling. A new column, Year\_Month, is then created using the dt.to\_period('M') function, which formats the purchase dates into a monthly period representation.

Following this, the code groups the data by the Year\_Month column and employs the agg() function to calculate two key metrics: the total number of unique transactions and the total amount spent for each month. This aggregation results in a summarized DataFrame that provides insights into spending trends over time.

Finally, the monthly spending analysis is printed to the console, allowing for an easy review of how customer spending varies from month to month. This analysis can be particularly useful for identifying seasonal trends and understanding customer behavior over time.



This code snippet focuses on customer loyalty analysis and the distinction between new and existing customers. It begins by aggregating transaction data for each customer in the merged\_data DataFrame. The code groups the data by Customer\_ID and calculates several metrics: the total number of transactions, the total amount spent, and the dates of the first and last purchases. This aggregation results in a loyalty\_analysis DataFrame that summarizes each customer's transaction behavior. Additionally, it calculates the average amount spent per transaction by dividing the total amount spent by the total number of transactions. The resulting loyalty analysis is printed, providing insights into customer spending patterns and retention.

The analysis then shifts to differentiating between new and existing customers. The Join\_Date column in the customer\_data DataFrame is converted into datetime format to facilitate date comparisons. New customers are defined as those who joined within the last year from the most recent transaction date in the merged data. A cutoff date is established, and a new column, Customer\_Status, is created using a lambda function that labels customers as 'New' or 'Existing' based on their join date.

Finally, the customer status information is merged back into the transaction data, creating a merged\_data\_with\_status DataFrame that includes both transaction details and customer status. This enriched data can now be used for further analysis on spending behaviors between new and existing customers.



This code snippet analyzes customer spending and transaction behavior based on their status as new or existing customers. It begins by grouping the merged\_data\_with\_status DataFrame by the Customer\_Status column and using the agg() function to calculate two key metrics: the total number of unique transactions and the total amount spent for each customer status. The resulting status\_analysis DataFrame summarizes how spending and transaction behaviors differ between new and existing customers, and this analysis is printed for review.

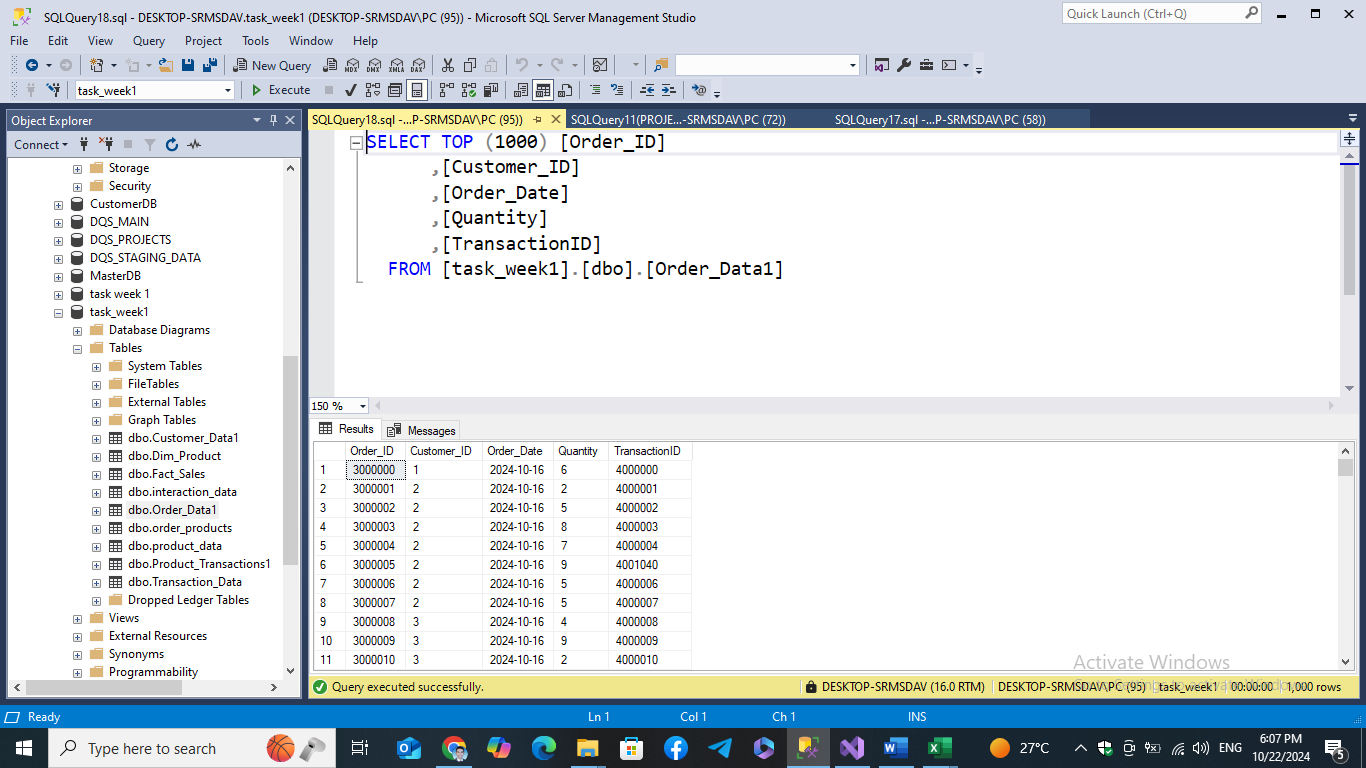
Next, the code identifies inactive customers—those who have not made any transactions in the last six months. It establishes an inactivity cutoff date by subtracting six months from the most recent purchase date found in the merged\_data. The code then filters the loyalty\_analysis DataFrame to find customers whose last\_purchase date is earlier than this cutoff, resulting in a DataFrame of inactive customers. This information is printed, providing insights into customer retention and highlighting individuals who may need to be re-engaged through marketing efforts.

**Week 2:**

**Part 1 : Implement a SQL DATA Warehouse:**

**Aggregating and managing large volumes of customer data for analytical purposes.**

**Step 1:** Studying the data base and check its accuracy and cleanliness and check for any cleansing required.



**Step 2: Make the necessary cleansing processes such as changing data types of some columns to the correct data types and handling nulls.**

A computer screen with a white screen

Description automatically generated

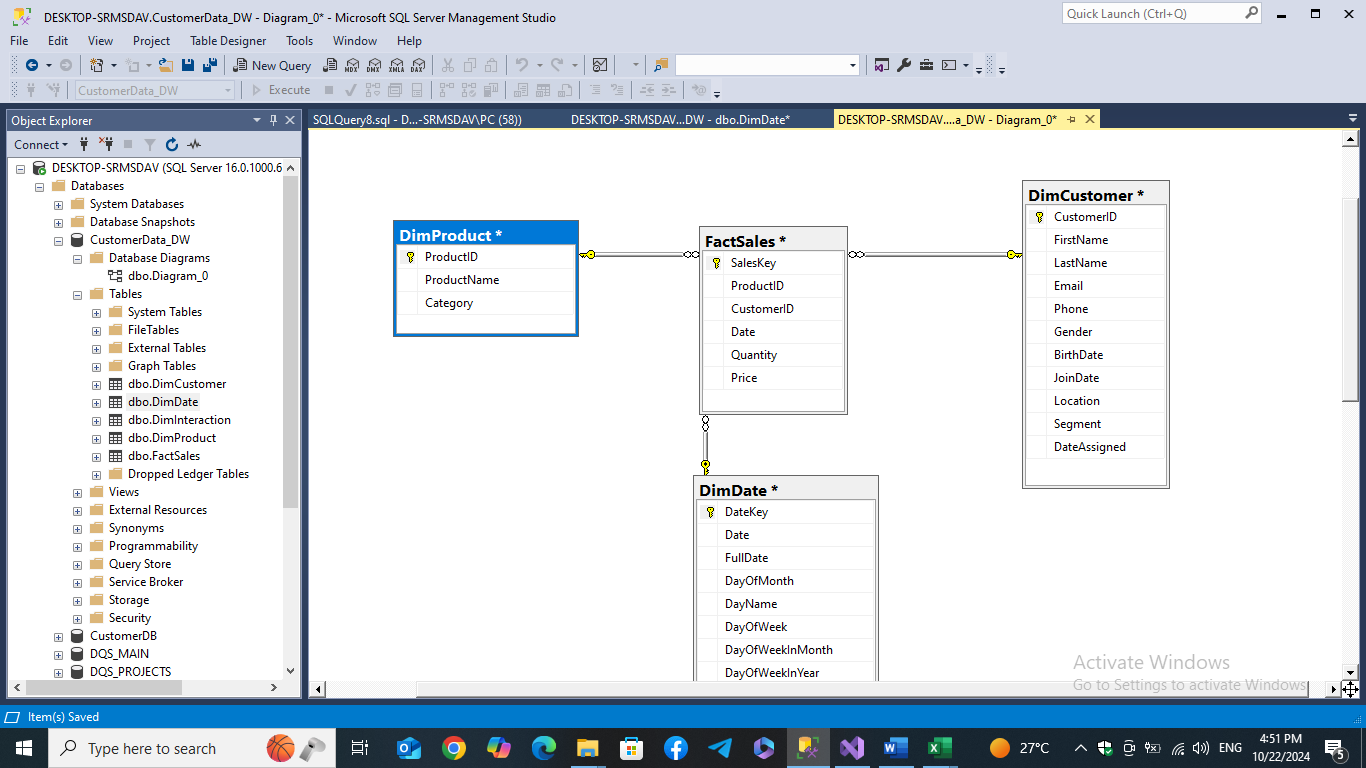
**Step 3: Studying the data base structure and components after being cleaned to identify fact and dimension tables in the data warehouse.**

**Step 4: Identify the columns to be put into fact and dimension tables.**

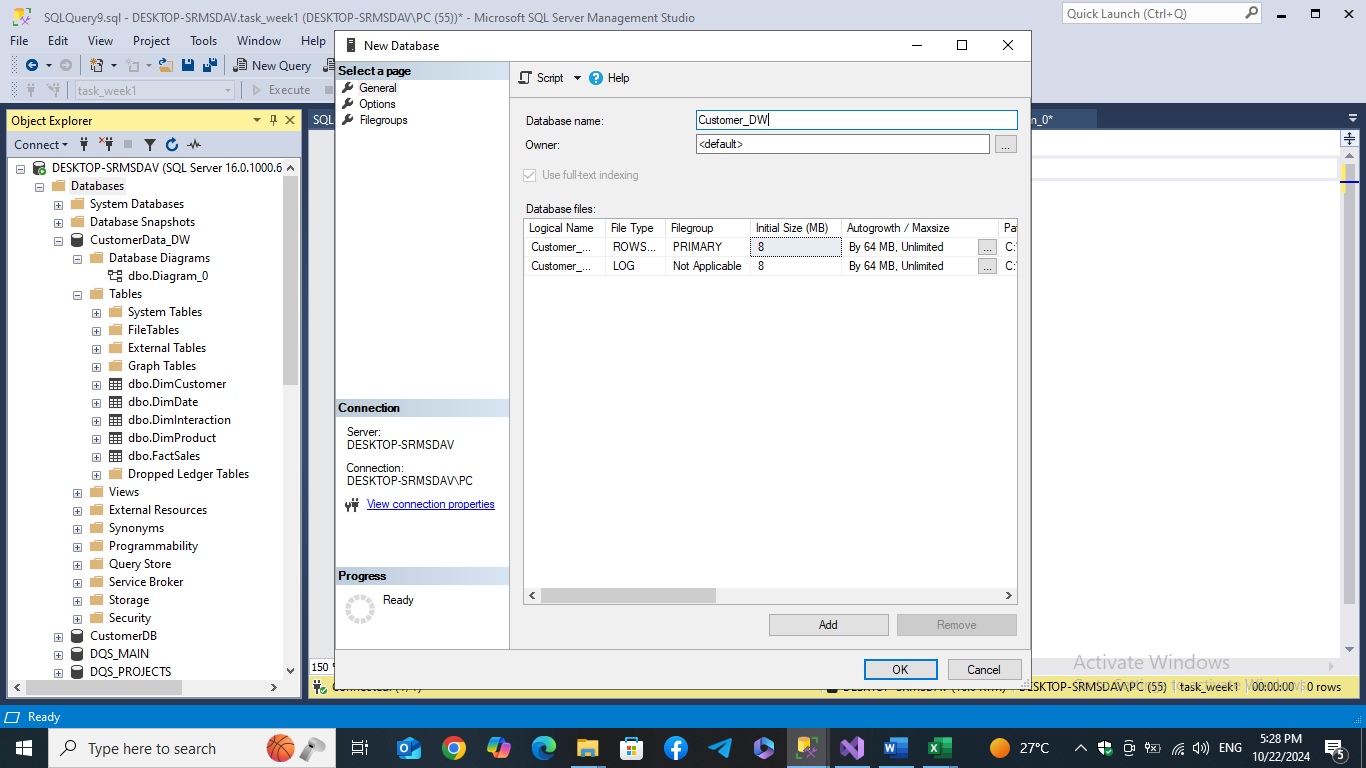
**Step 5: Choosing the suitable schema for the data which is the star schema in our case.**

**Step 6: Create a surrogate-key in the Fact table named (SalesKey) to be the key of the fact table in the data warehouse.**

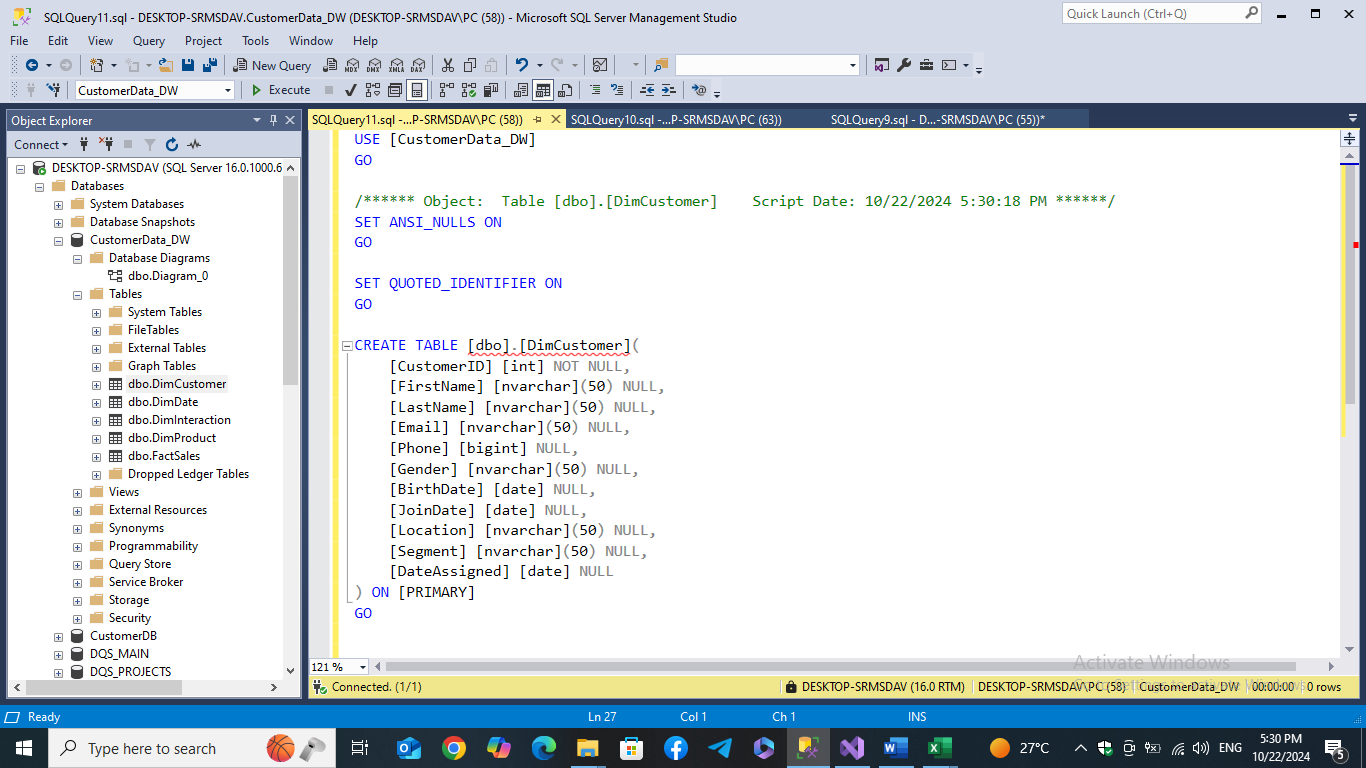
**Step 7: Identify Primary keys in dimension tables.**



**Step 8: Create the data warehouse(Customer\_DW):**



**Step 9: Create: Fact And Dimension Tables of the data warehouse:**



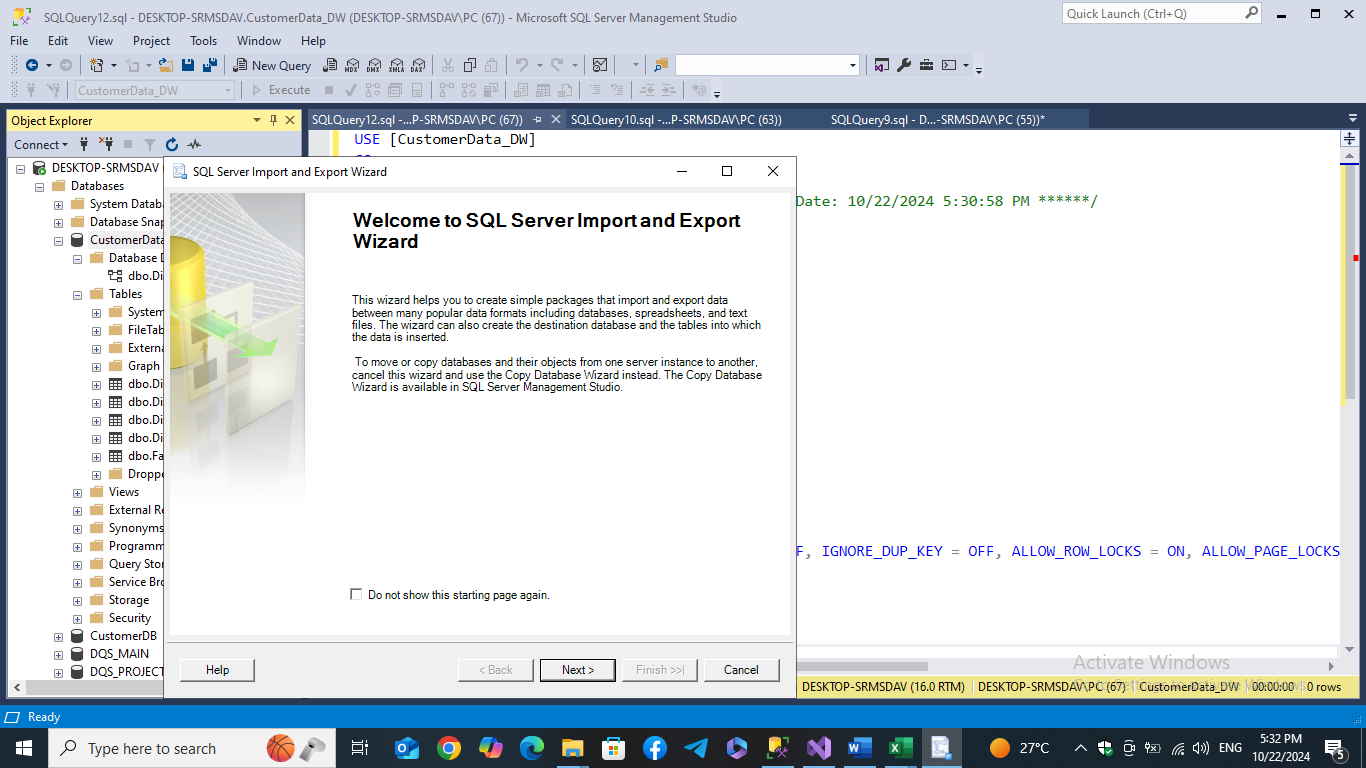
A computer screen with text

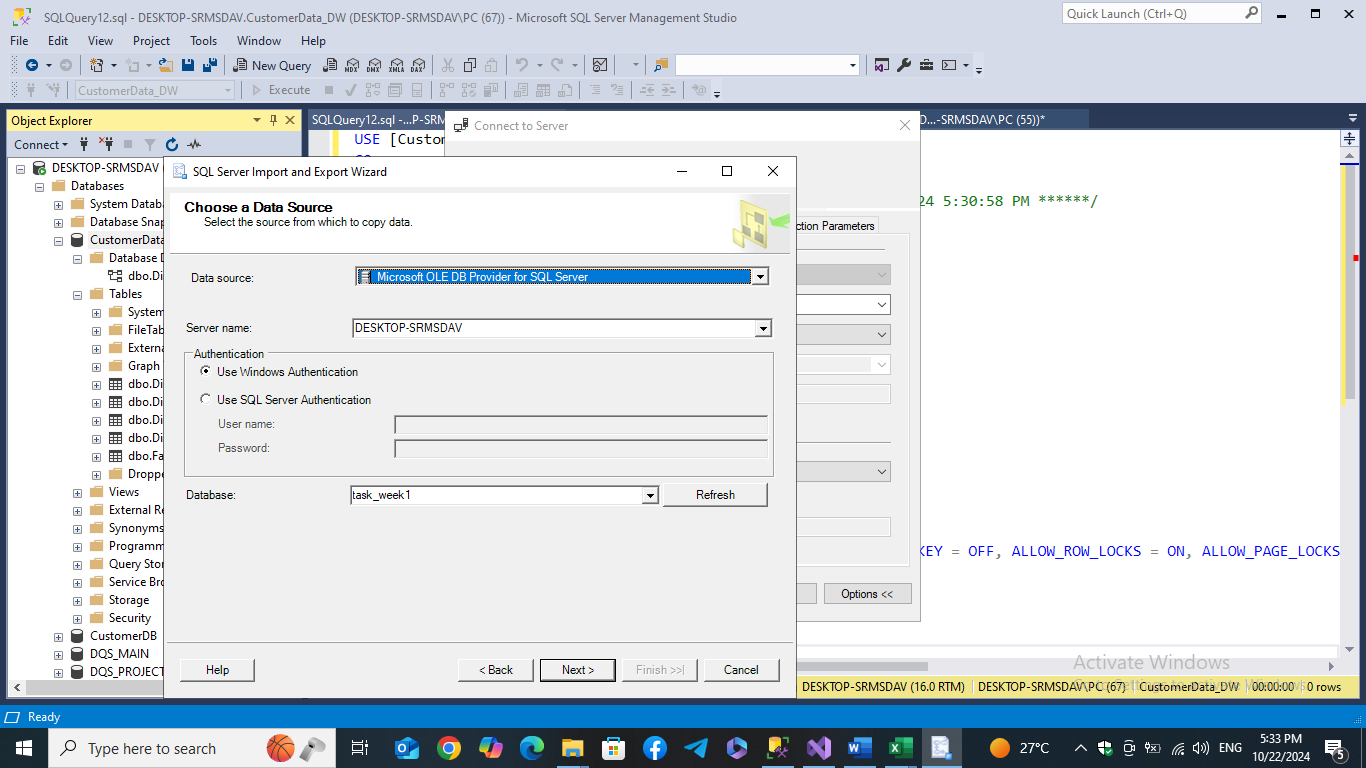
Description automatically generated

**Part 2: Load Data From Various Resources Into Data Warehouse**

**Step 1: Populate Customer and product Dimension Tables (Using Import and Export Wizard Tool):**

**Choosing source and destination databases, choosing tables, mapping columns, and finishing the import prosess.**



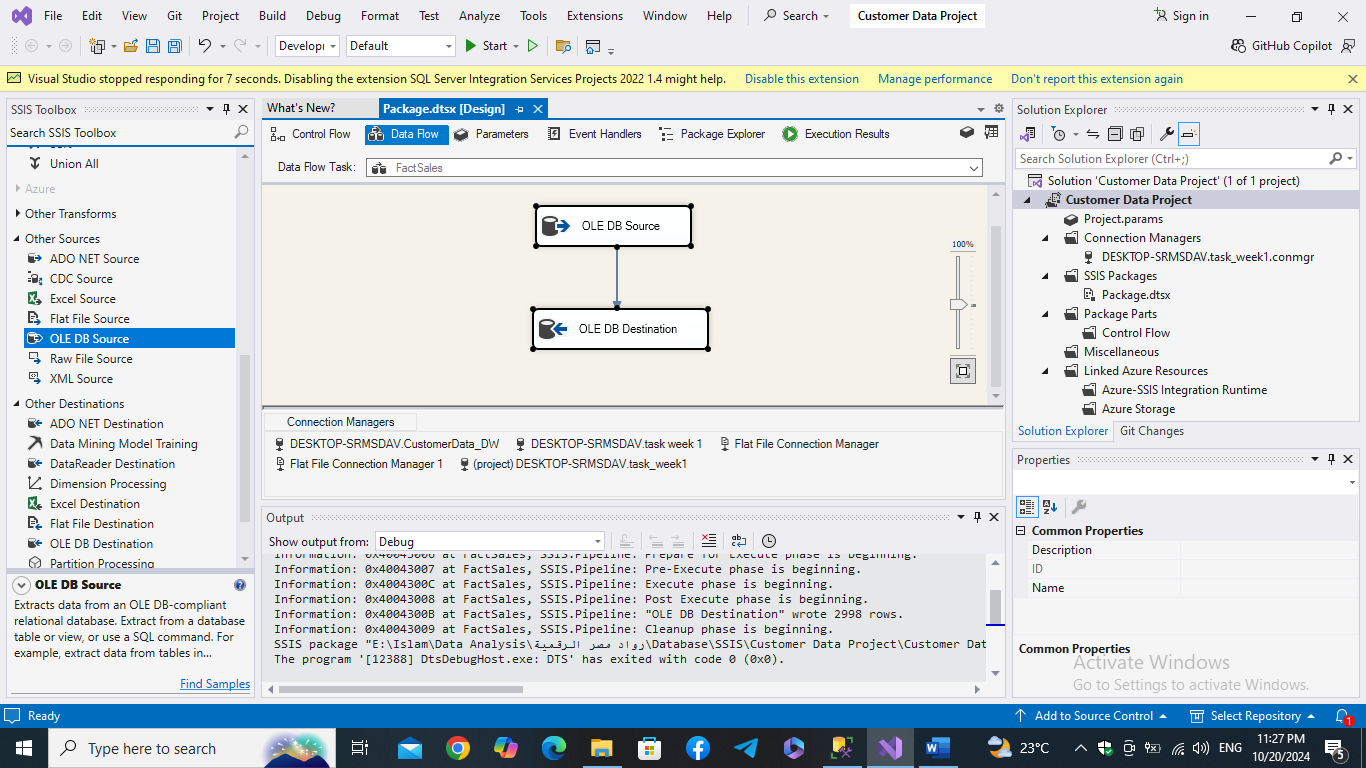


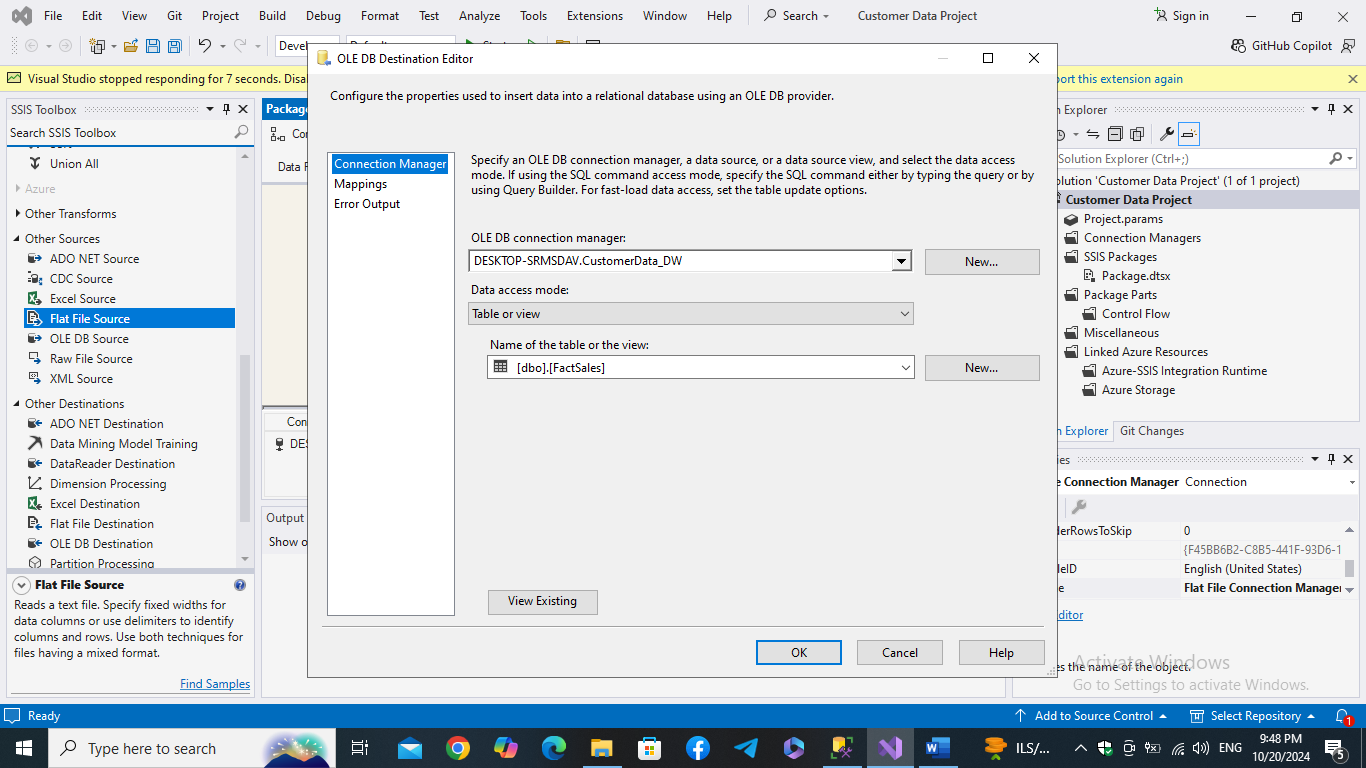
A screenshot of a computer

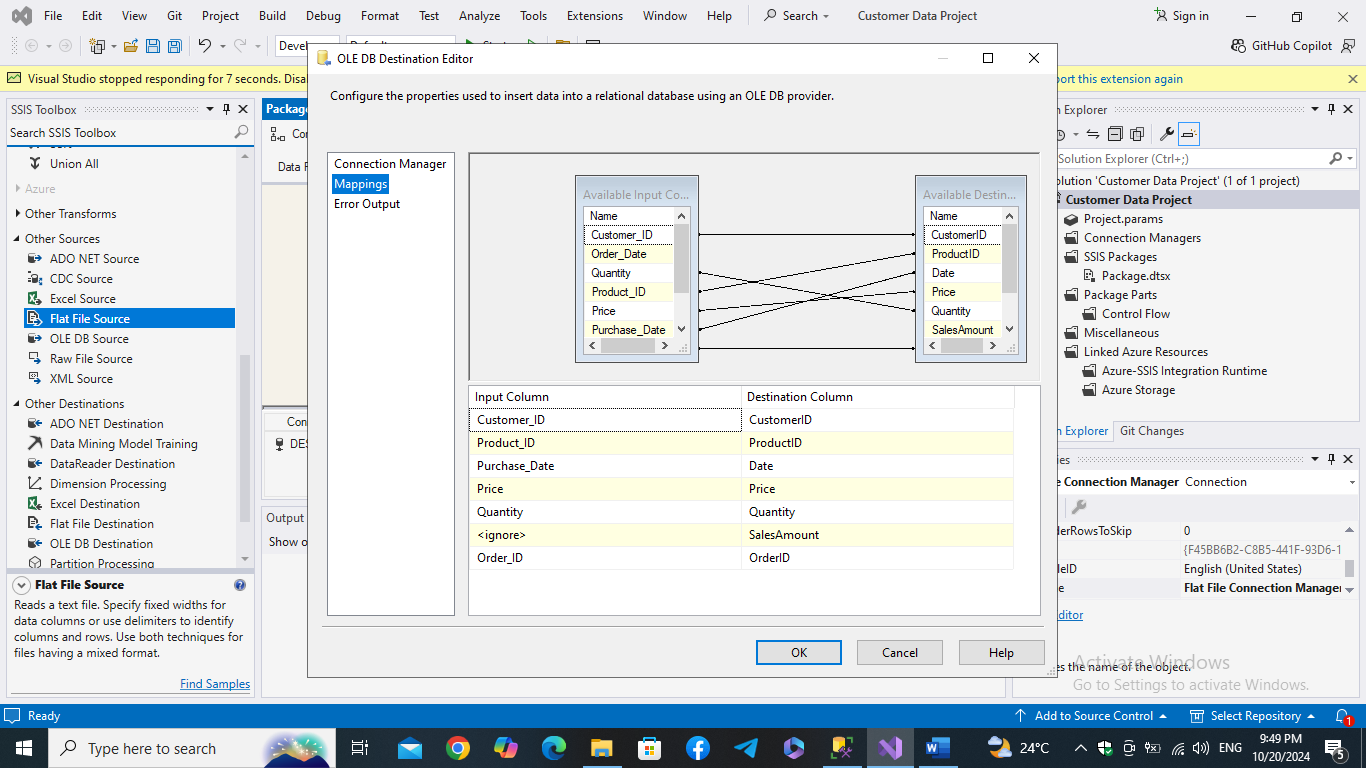
Description automatically generated

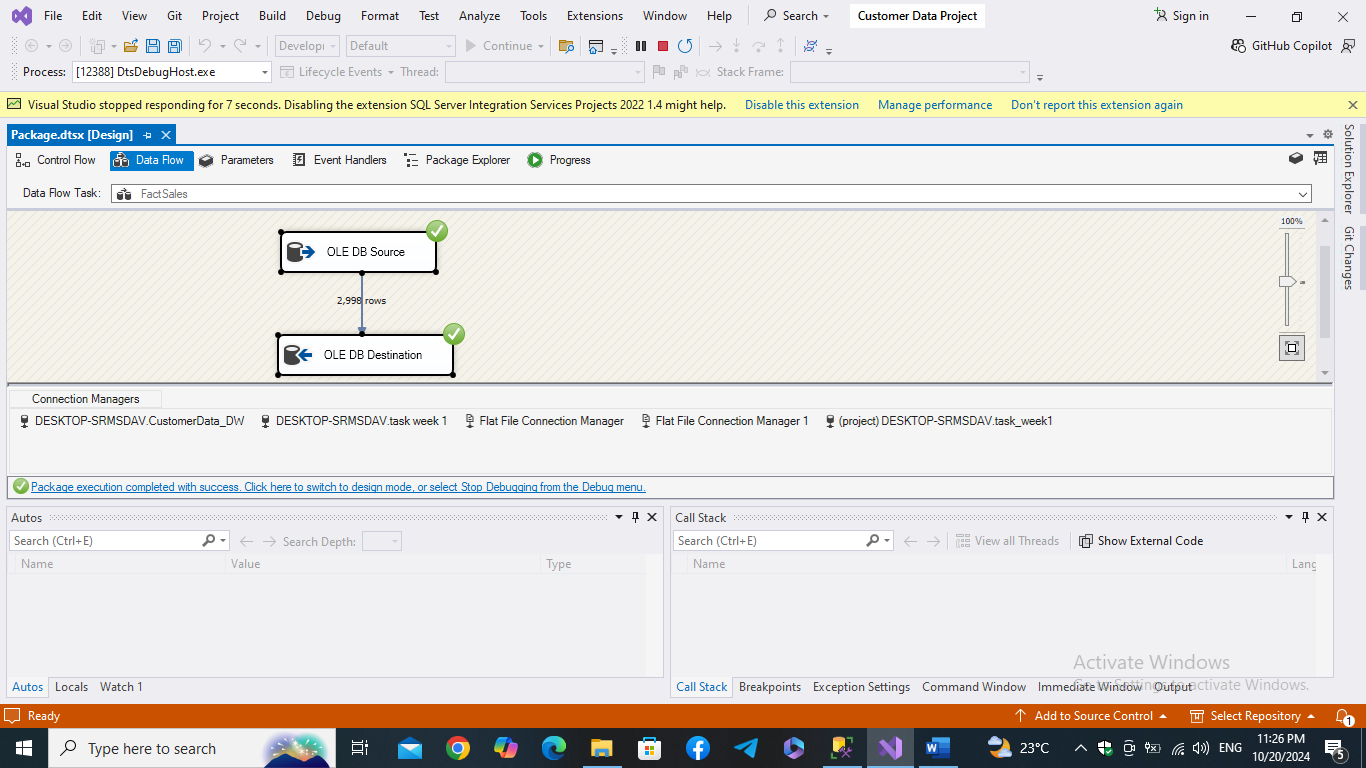
**Step 2: Populate Product Dimension Table (Using SSIS and Visual Studio):**

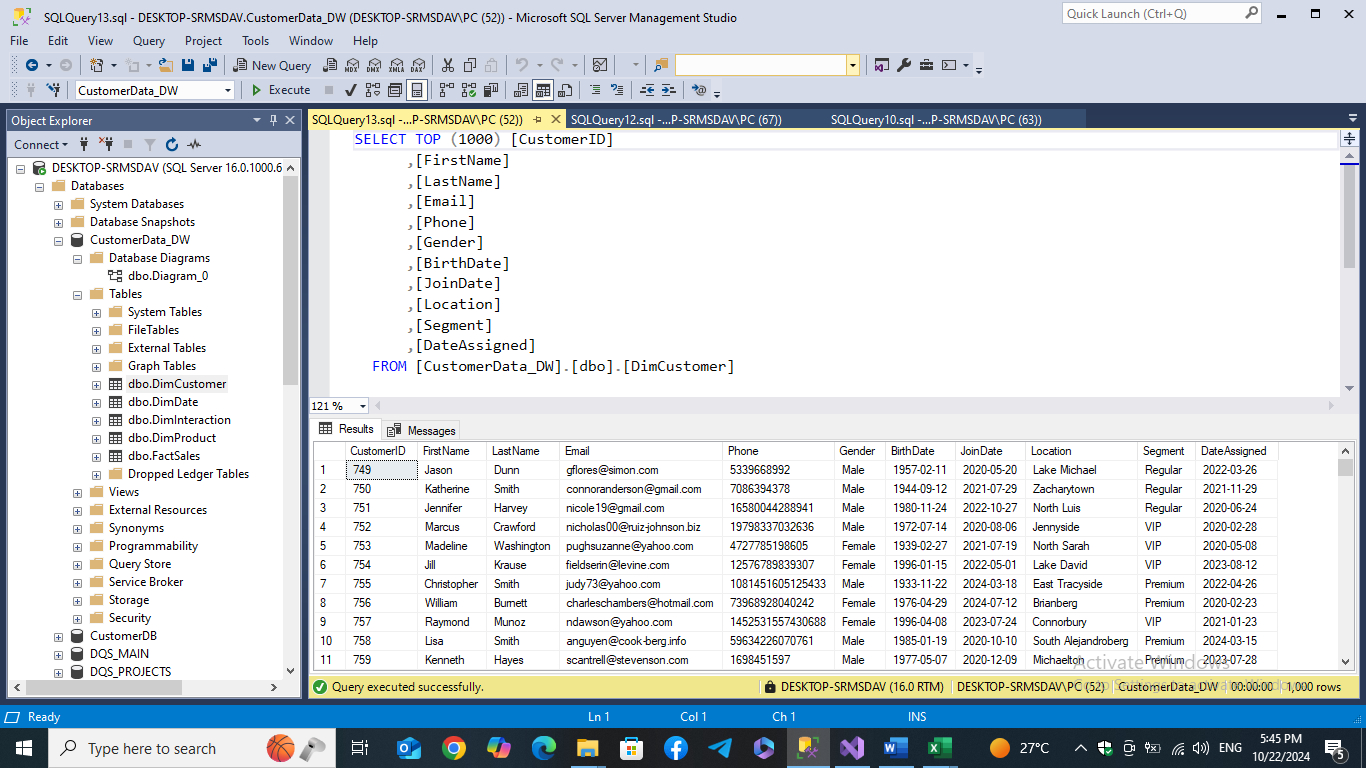
**Set the connection manager, Create a data flow, then the work flow in which we choose source and destination and map the columns and the to run the SSIS package to affect the data warehous**









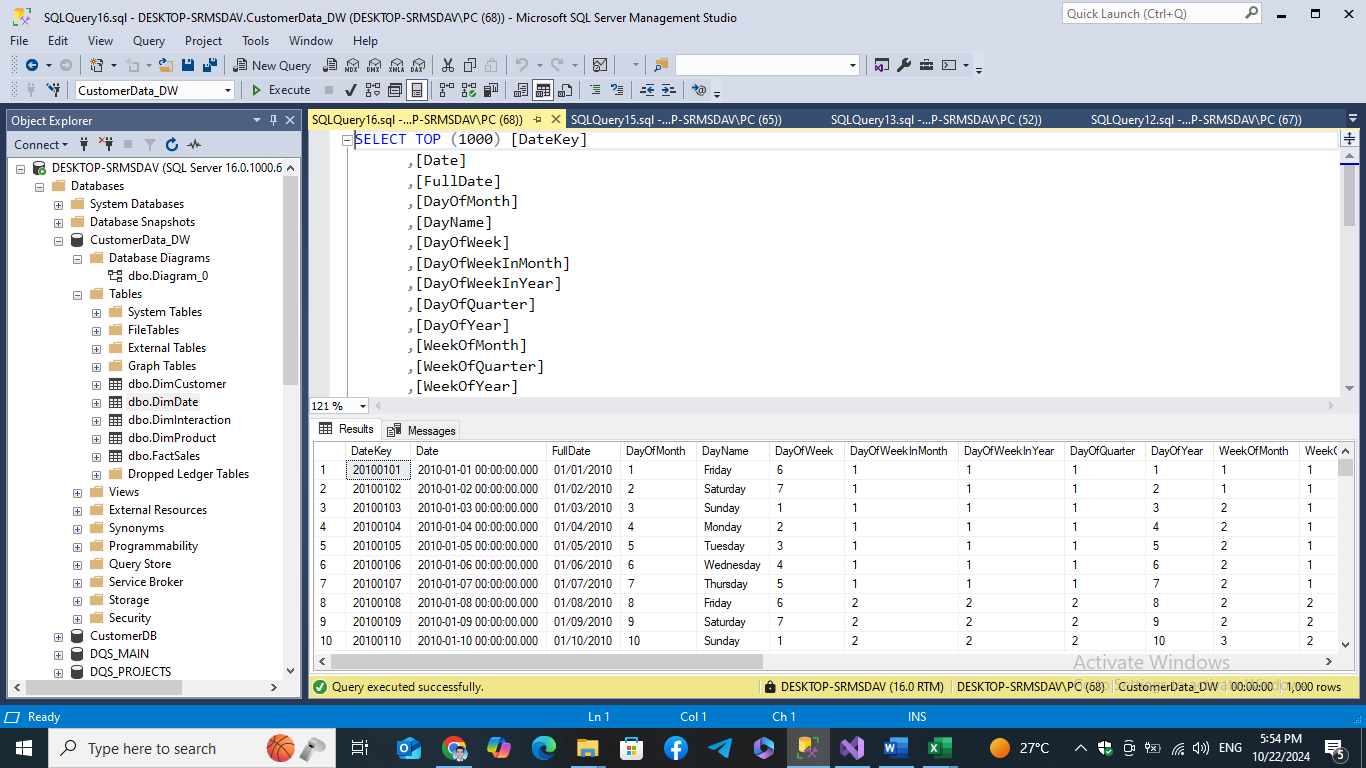
**Step 3: Check the populated customer Dimension table on the warehouse**

**Step 4: Populate the Date Dimension table:**

**Using SQL query populated the time** A computer screen with text on it

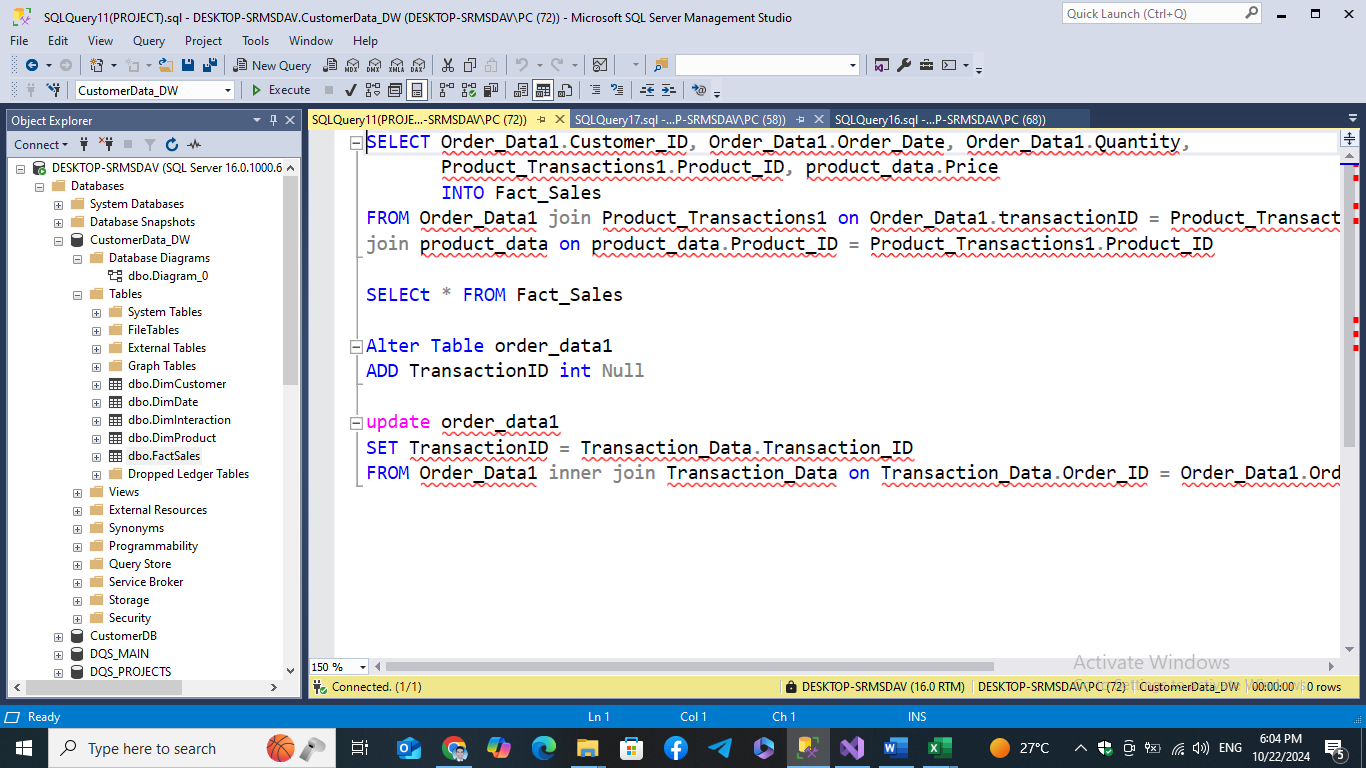
Description automatically generated

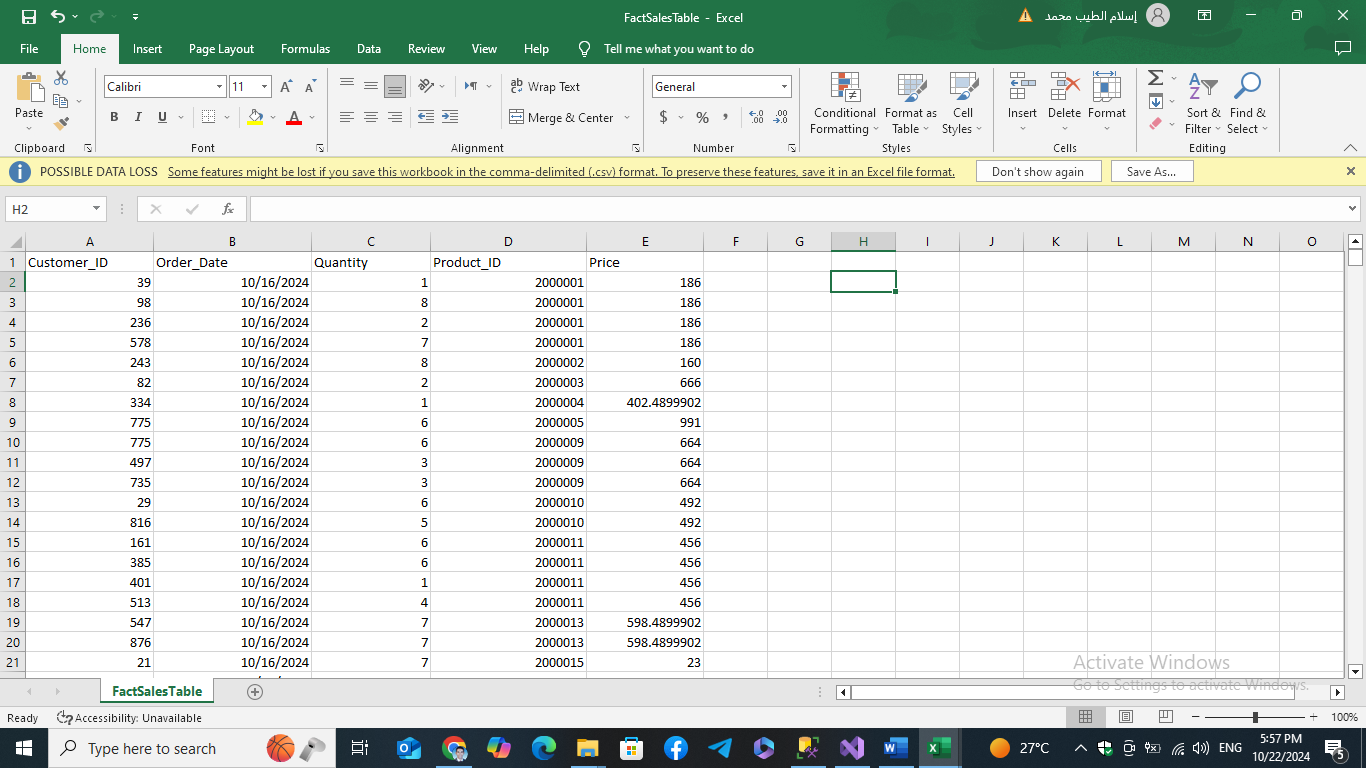
**dimension table with all its columns**



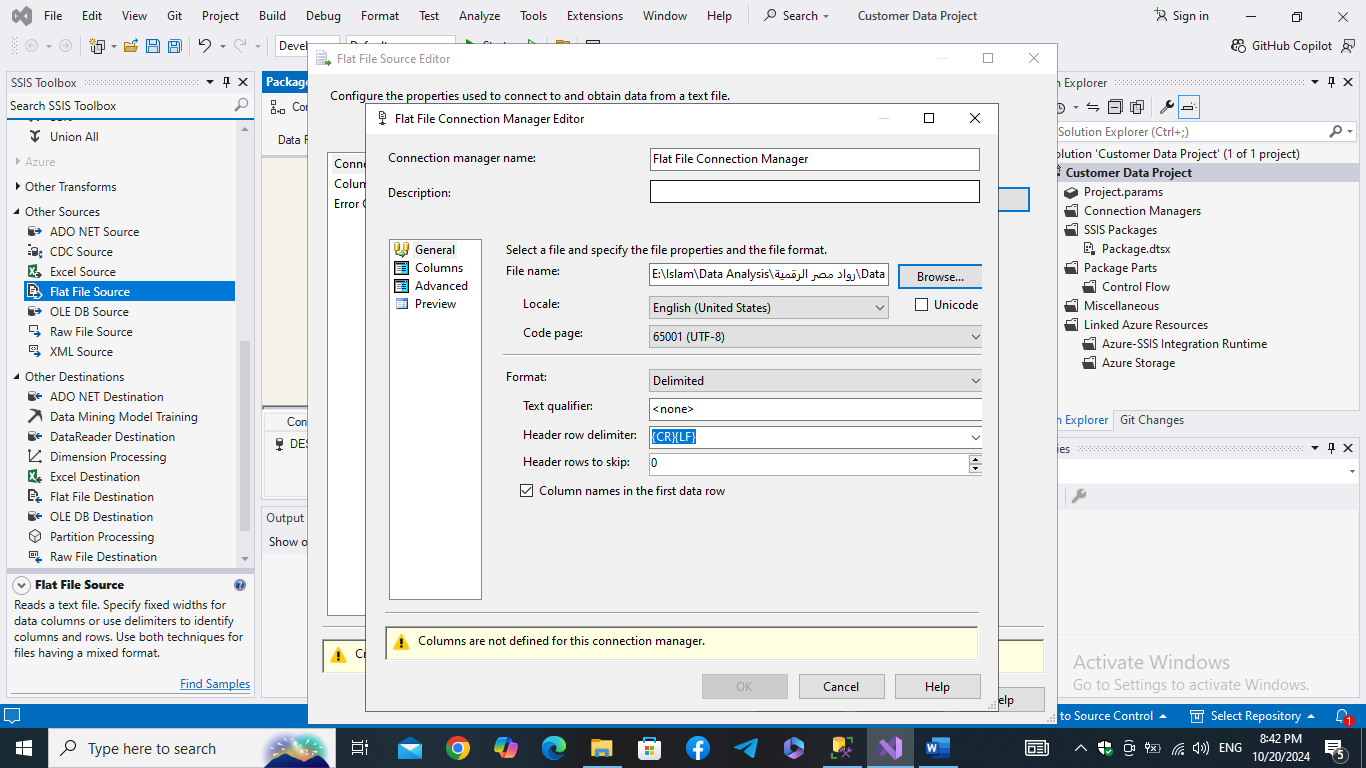
**Step 5: Check the populated data in date dimension table:**

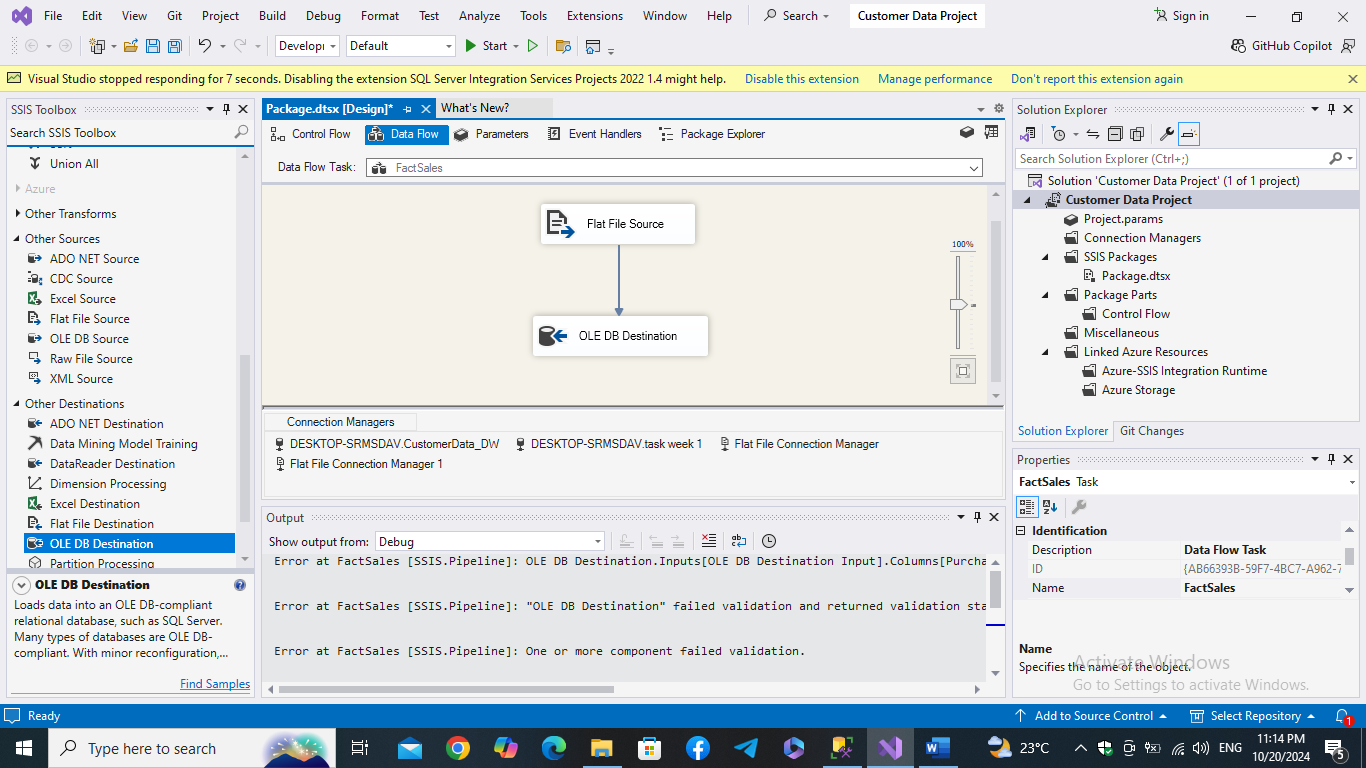
**Step 6: Populating the fact table (using a csv file created from the database using SQL and ETL this table using SSIS Tool and Visual Studio)**



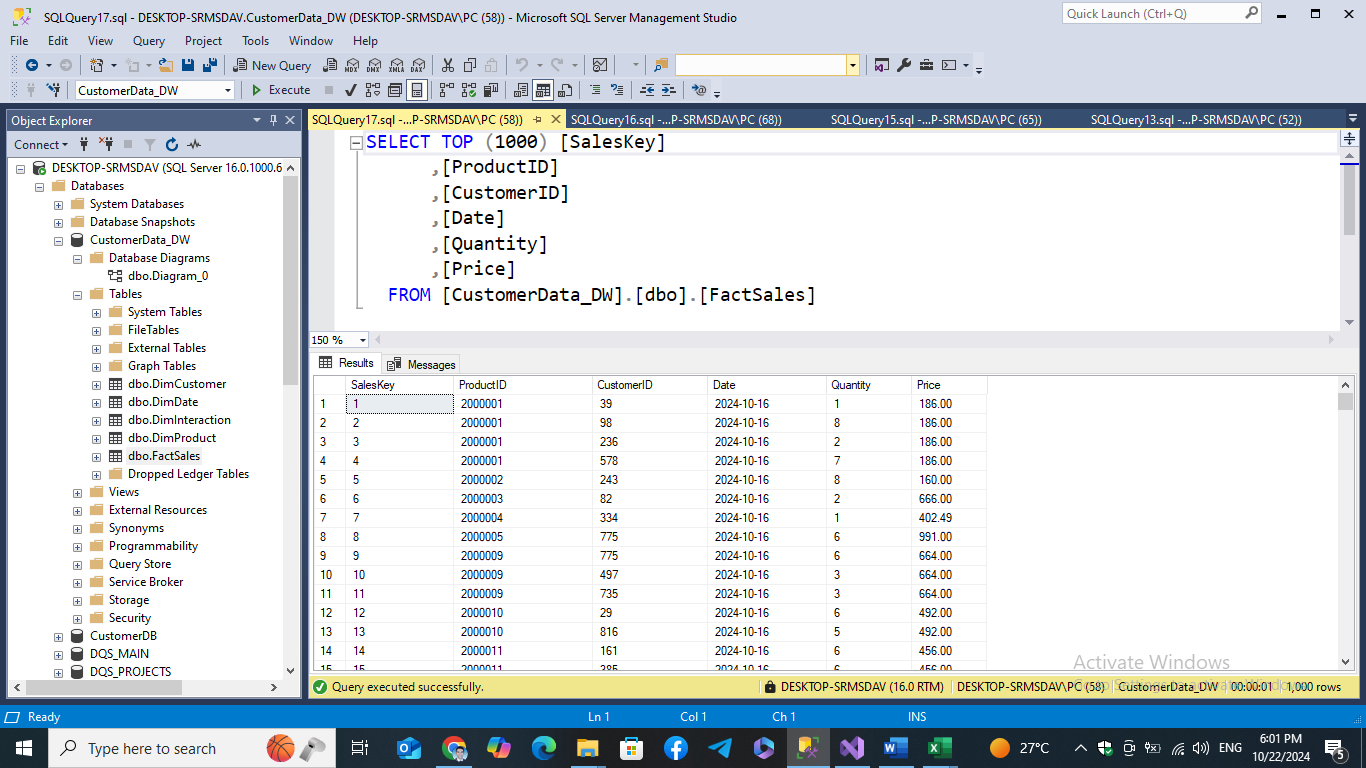








**Step 7: Check the populated data of Fact table in the data warehouse:**



**Now data warehouse is ready for conducting the required analytics!**

**Data Science and Azure Integration**

**Week 3: Data Science and Azure Integration**

**Tasks**

Data Science with Python**:**

**Performed data analysis and built predictive models using Python for customer segmentation and lifetime value prediction.**

Azure Data Fundamentals**:**

**Utilized Azure Data services to manage and analyze customer data effectively.**

Model Development**:**

**Developed and evaluated a Linear Regression model using Python and Azure Machine Learning to predict customer lifetime value.**

**Tools and Libraries**

Python**:**

**Libraries:**

pandas**: For data manipulation and analysis.**

scikit-learn**: For building and evaluating machine learning models.**

numpy**: For numerical computations.**

joblib**: For saving and loading model files.**

Azure Data Studio**: For managing and querying Azure data services.**

Azure Machine Learning**: For developing, training, and evaluating machine learning models.**

**Deliverables**

Analysis Report**:**

**Insights and predictive models for customer lifetime value were documented in a comprehensive analysis report.**

Integrated Azure Data Services**:**

**Setup and documentation for integrating Azure Data services with data analysis tasks.**

**Week 4: MLOps, Deployment, and Final Presentation**

**Tasks**

MLOps Implementation**:**

**Implemented MLOps using MLflow to track experiments and manage machine learning models.**

Model Deployment**:**

**Deployed the machine learning model using Azure services. Created an online endpoint for model predictions.**

**Tools and Libraries**

MLflow**: For tracking machine learning experiments and managing models.**

Azure Machine Learning**: For deploying models and managing online endpoints.**

Azure Blob Storage**: For storing and managing model files.**

**Deliverables**

Tracked Experiments**:**

**All experiments were tracked using MLflow, providing a comprehensive record of model development and evaluation.**

Deployed Model**:**

**Had issues with the deployment giving the below code,**

