BUSINESS ANALYTICS
Master of Science

Data Management and Business Intelligence

Assignment 2

ELENI RALLI (f2822312)
ANNA PAPADIMITRIOU (f2822311)

Professor: Damianos Hatziantoniou

Contents

1. The Discovery of the Dataset

We visited Kaggle and found the following dataset:
Online Shopping Dataset
Exploring Online Shopping Trends and Patterns
https://www.kaggle.com/datasets/jacksondivakarr/online-shopping-dataset?rvi=1

This dataset has a mix of categorical and numerical data, and is complex enough to benefit from a data warehouse.
We investigate the structure of the dataset, identifying potential facts and dimensions and we understand the business context and goals that the dataset represents.

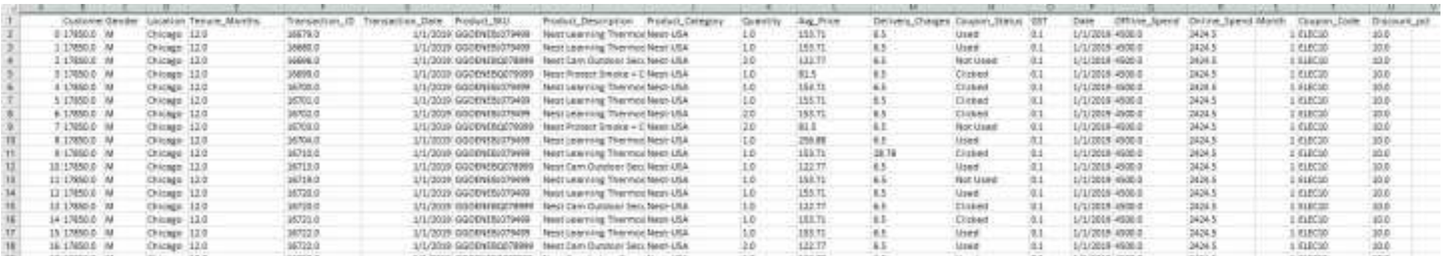
2. Presentation of the columns of the dataset

1) The first column in the excel is just the number of the row, without header.
2) **CustomerID:** This is a unique numeric identifier assigned to each customer. It helps in uniquely identifying and differentiating each customer.
3) **Gender:** A categorical field indicating the gender of the customer, with values like 'M', 'F'.
4) **Location**: Textual information about the customer's location or address. This include city or state.

> 1. Washington DC: This is a city, but it's also a federal district known as the District of Columbia. It serves as the capital of the United States and is not part of any state.
> 2. New York: This can refer to the state of New York
> 3. California: This is a state on the west coast of the United States.
> 4. New Jersey: This is a state
> 5. Chicago: This is a city, specifically the largest city in the state of Illinois.

5) **Tenure_Months**: A numeric field indicating the number of months the customer has been associated with the platform or service.
6) **Transaction_ID:** A unique numeric identifier for each transaction. It's used to uniquely identify every transaction made.
7) **Transaction_Date:** Date of the transaction.
8) **Product_SKU:** Text field representing the Stock Keeping Unit (SKU), which is a unique identifier for each product, used for tracking and inventory management.
9) **Product_Description:** A textual description of the product.
10) **Product_Category:** A categorical field describing the category to which the product belongs.
11) **Quantity:** Numeric field indicating the quantity of the product purchased in the transaction.
12) **Avg_Price:** The average price of the product, represented as a numeric value(inclusive of any discounts, GST, and delivery charges (if applicable))
13) **Delivery_Charges:** Numeric field indicating the charges associated with the delivery of the product.
14) **Coupon_Status:** Categorical field indicating the status of any coupon associated with the transaction
15) **GST:** Goods and Services Tax associated with the transaction, expressed as a numeric value. It represents the tax applied to the purchase.
16) **Offline_Spend:** A numeric field indicating the amount spent by the customer through offline channels (inclusive of any discounts, GST, and delivery charges (if applicable))
17) **Online_Spend:** A numeric field indicating the amount spent by the customer through online channels (inclusive of any discounts, GST, and delivery charges (if applicable))
18) **Month:** Month of the transaction.
19) **Date**: Date of the transaction (redundant with Transaction_Date)
20) **Coupon_Code:** Text field representing the code associated with a coupon.
21) **Discount_pct:** Numeric field indicating the percentage of discount applied to the transaction.

- **The dataset has 52956 rows.**



	Customer	Gender	Location	Tenure_Months	Transaction_ID	Transaction_Date	Product_SKU	Product_Description	Product_Category	Quantity	Avg_Price	Delivery_Charges	Coupon_Status	GST	Date	Offline_Spend	Online_Spend	Month	Coupon_Code	Discount_pct

3. Multidimensional modeling process

In multidimensional data modeling, a fundamental distinction from «standard» data modeling is the approach towards data inclusion. The modeler should focus on incorporating only those data elements and relationships that are critical business drivers, rather than attempting to include all available data and relationships. Another notable aspect is the acceptance of redundancy in specific, strategically chosen areas (primarily in dimensions), when it serves to enhance the model's intuitiveness for users.

Kimball organizes the multidimensional modeling process into four sub-processes:

1. Choose the business processes to model (not all business processes are equally important for the business, we prioritize the one with the largest potential for increasing the profits).

Transaction Analysis is chosen due to its direct impact on understanding revenue generation and customer purchasing behavior. It is a critical process for identifying areas of improvement and growth opportunities within the business. (an emphasis on both product-level sales details and customer behavioral insights.)

Objectives of Transaction Analysis:

- Revenue Tracking: With the dataset, we can monitor sales over time and identify top-performing products or categories, such as those within the Nest-USA category which has high frequency according to the Product_Category statistics.

- Customer Behavior: We can segment customers by Gender, Location, and Tenure_Months to create targeted marketing campaigns and personalized shopping experiences.

- Inventory Management: Utilize Quantity and Product_SKU data to manage stock levels and optimize inventory based on sales velocity.

- Marketing Effectiveness: Analyze Coupon_Status and Coupon_Code usage to determine the success of promotions and refine marketing strategies.

- Price Optimization: Leverage Avg_Price and Discount_pct to find the best pricing strategies that attract customers while maintaining profitability.

- Sales Forecasting: Use historical data like Transaction_Date and Month to predict future sales trends and prepare accordingly.

- Profitability Analysis: Assess the profitability of sales by examining GST, Delivery_Charges, Offline_Spend, and Online_Spend.

- Customer Lifetime Value (CLV): Estimate potential revenue from customer data to focus retention efforts on the most valuable customer segments.

- Cross-Selling and Up-Selling Opportunities: Use transaction records to identify opportunities to promote related products or higher-value alternatives.

- Risk Management: Monitor transaction patterns for potential fraud or irregularities, such as unusual Quantity or Discount_pct values.

2. Choose the granularity of the business process.

Product-Line Level Granularity: Each record represents an individual product within a transaction. This granularity allows for a detailed analysis of sales at the product level, including product preferences, inventory management, and sales performance.

Customer-Level Analysis: In addition to product-line level details, aggregate data at the customer level to understand overall customer purchasing patterns, customer lifetime value, and behavioral segmentation.

3. Design the dimensions.

Given the dataset, we can define the following dimensions:

- Customer Dimension:
  Attributes: Customer_created_ID, CustomerID, Gender, Tenure_Months, Location_created_ID
  Link: Location_created_ID (to LocationSubDim).
  Use: For customer profile analysis, and Customer Lifetime Value estimation.

- Location Sub-Dimension:
  Attributes: Location_created_ID,City, State, Country.
  Use: demographic segmentation.
  Location Sub-Dimension Hierarchy: Days → Months → Quarters → Years.

- Product Dimension:
  Attributes: Product_created_ID,Product_SKU, Product_Description, Product_Category.
  Use: For product performance analysis, inventory management, and cross-selling opportunities.

- Time Dimension:
  Attributes: Transaction_Date_created_ID, Date, Day, Month, Year, Quarter
  Use: For sales forecasting, trend analysis, and temporal patterns.
  Time Dimension Hierarchy: Days → Months → Quarters → Years.

- Promotion Dimension:
  Attributes: Coupon_created_ID, Coupon_Code, Coupon_Status.
  Use: For marketing effectiveness and promotion analysis.

We chose to make a snowflake schema.
Because our primary concern is maintaining data integrity, reducing redundancy, and having a scalable and flexible system.

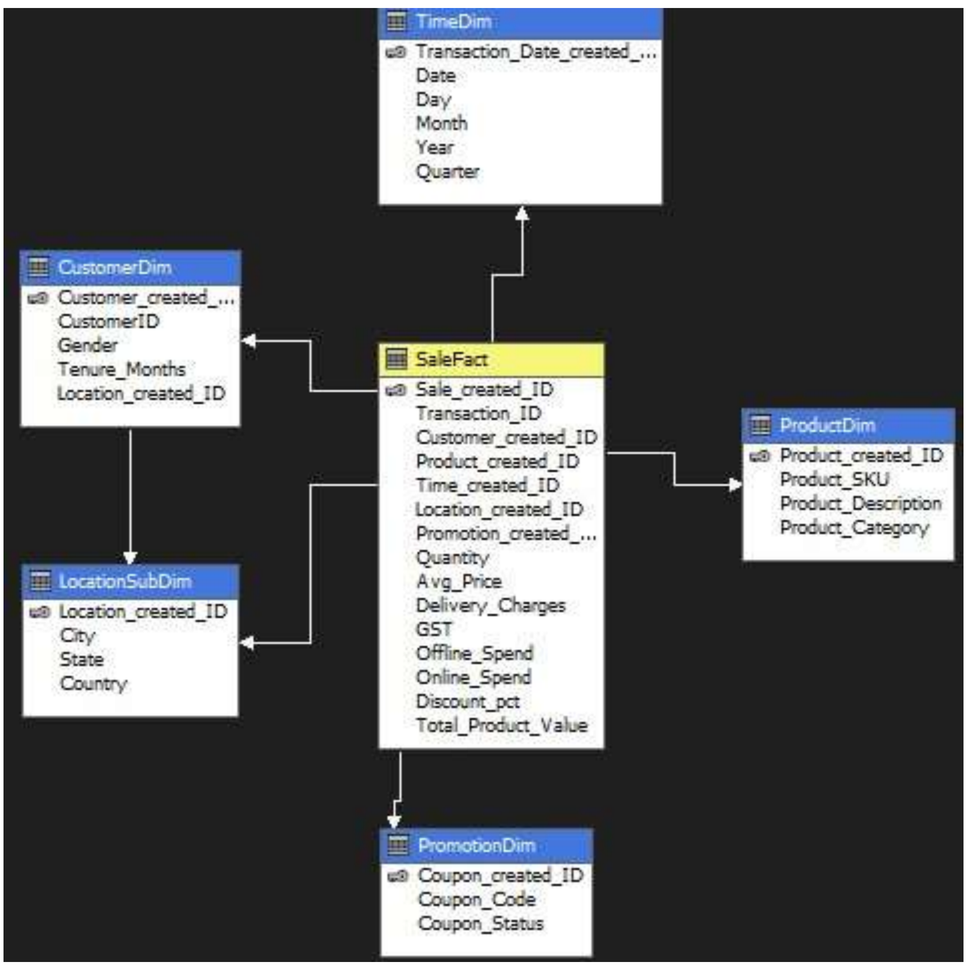
4. Choose the measures.

- **Quantitative Measures**:
  - Include Quantity, Avg_Price, Delivery_Charges, GST, Offline_Spend, Online_Spend,Discount_pct.
  - Use: To calculate total sales, profitability, and financial metrics at both product and transaction levels.

- **Derived Measures**:
  - Total_Product_Value →
    WHEN Coupon_Status = 'used' THEN (Quantity * Avg_Price) - (Quantity * Avg_Price * Discount_pct / 100) ELSE Quantity * Avg_Price
  - Use: For detailed revenue tracking and profitability analysis.

<u>SaleFact Table:</u>
Primary Key: Sale_created_ID.
Foreign Keys: Link to CustomerDim, ProductDim, TimeDim, LocationSubDim, PromotionDim.

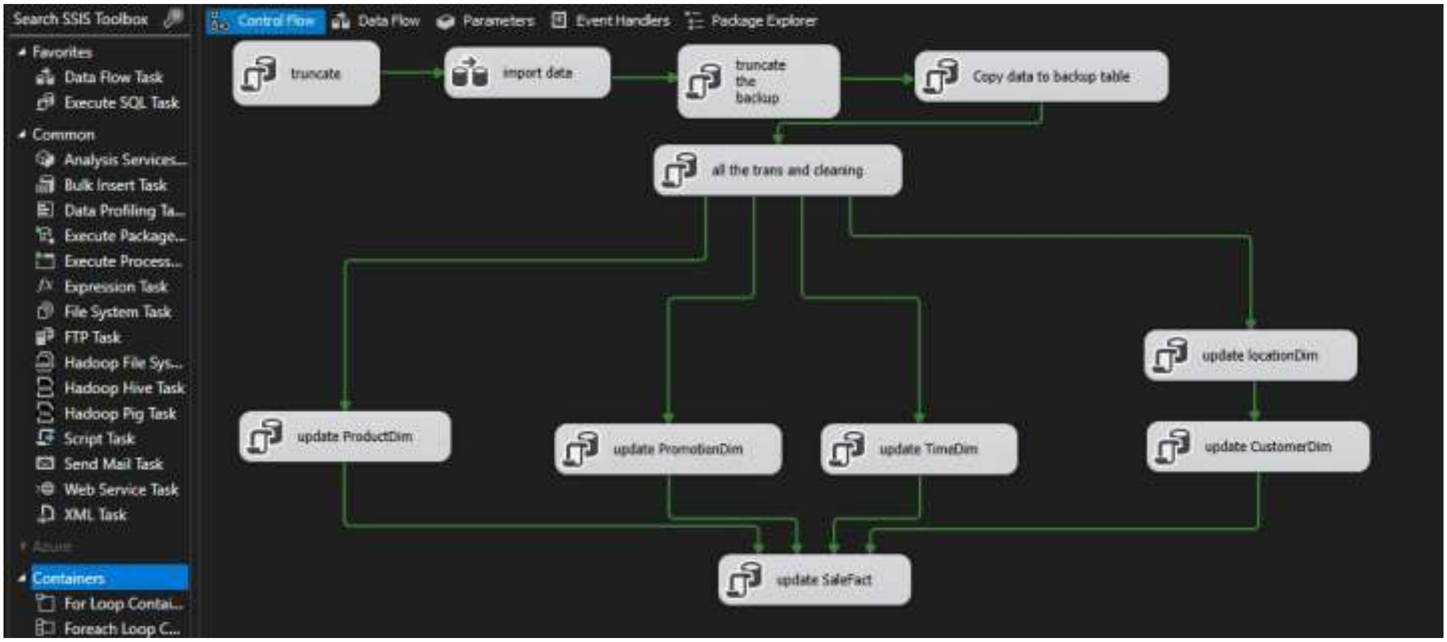
- Measures: Quantity, Avg_Price, Delivery_Charges, GST, Offline_Spend, Online_Spend, Discount_pct, Total_Product_Value



4. ETL (Extract, Transform, Load)

ETL plays a vital role in data-driven decision-making. In this assignment we took data from one source (one csv file) but usually in extract step somebody must consolidating data from various sources into a single repository.

It provides businesses with a holistic view of their operations, enabling better insights and strategic decisions. It also ensures that data used for analysis is reliable, consistent, and up-to-date.

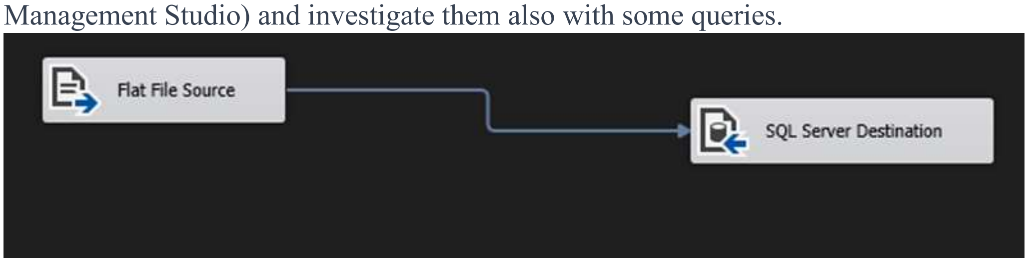


We will explain this control flow with the order that appears above. In this flow we did extract, transform and load of the data:

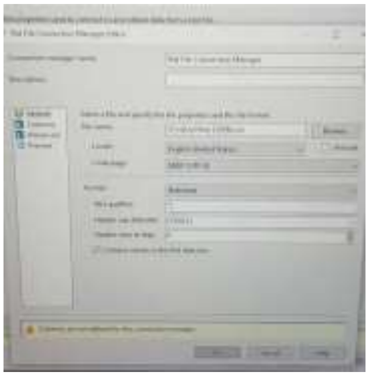
Import Data

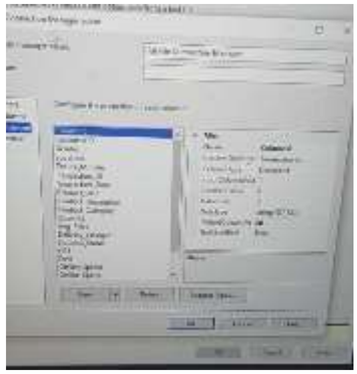
The first phase of the ETL process, where data is gathered from various source systems, in our assignment -one csv file (in the screenshot is the box "import data")

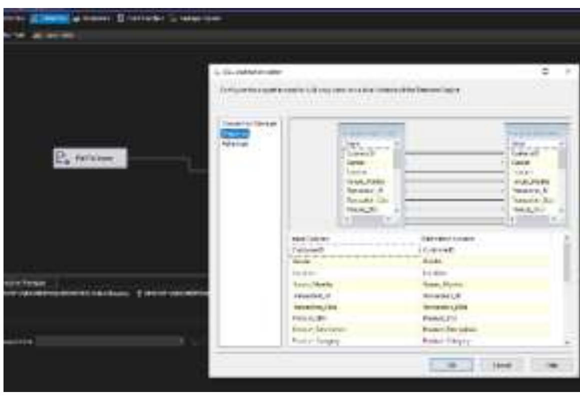
First of all we imported the data in R in order to investigate them and see problems that we will have to deal in cleaning process, in order to choose the business processes to model. In a real situation, the businessman would tell us what he is most interested in being recorded in the company's data, but for this specific assignment we made personal assumptions (the code is in the appendix) . In this step we choose the columns that we will input in order to examine the business processes that we describe in the "3.Multidimensional modeling process" part of this assignment.
After this we use the SSIS (SQL Server Integration Services) in order to import the data in the SSMS (SQL Server Management Studio) and investigate them also with some queries.

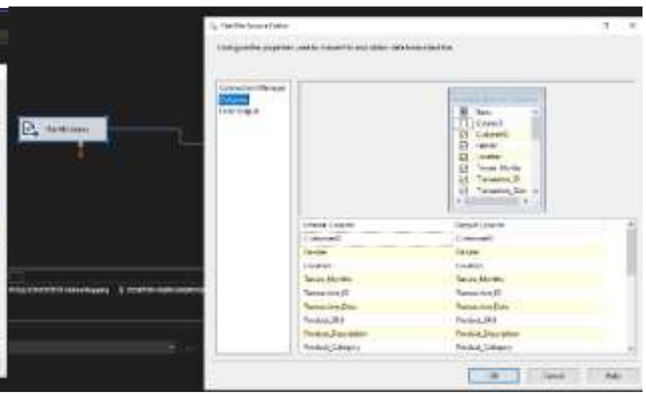


We didn't import all the columns. We didn't include the first column because it was just a count of the rows and the Date: Date of the transaction (redundant with Transaction_Date)

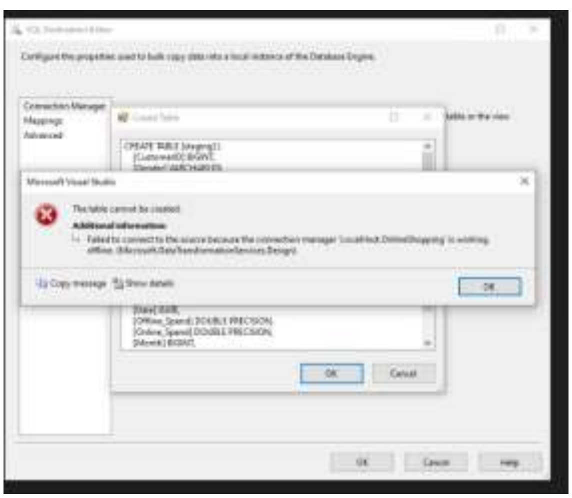




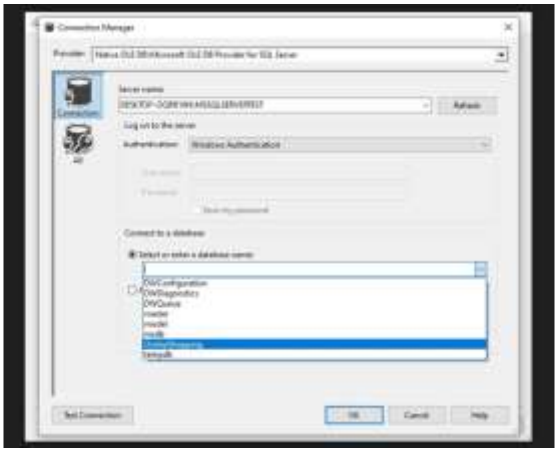


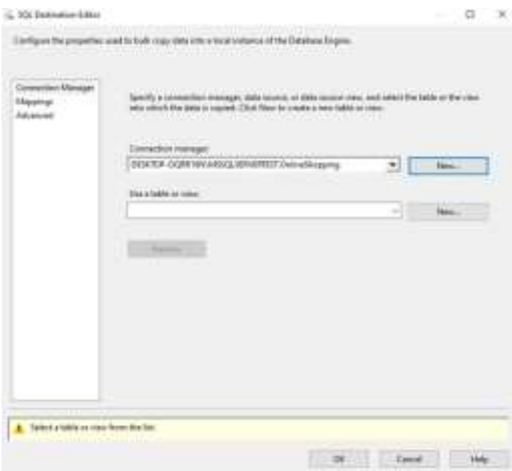


In this step we came across with some errors:

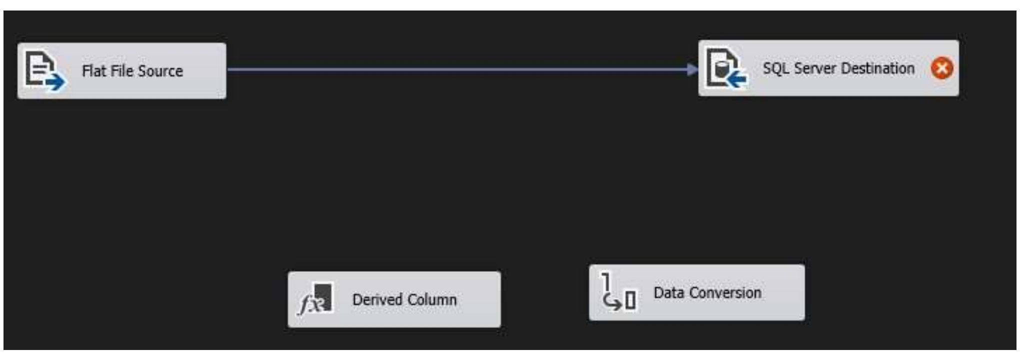


In order to make the connection we chose this provider:





After that the error that we saw was that the Transaction_date column could not be read as a date while we were selecting DT_DBDATE and also we tried to do Data conversion with these 2 ways in the screenshot but the error continued to exist:

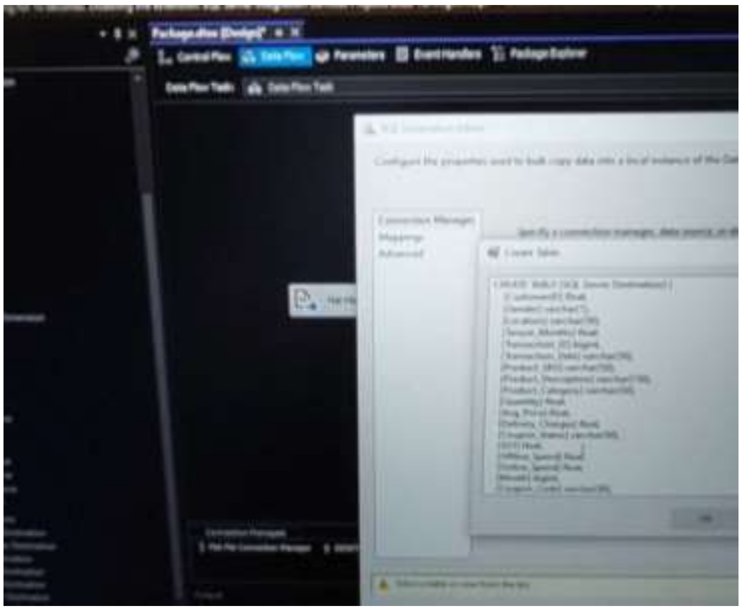


In the end we chose to import this column as string and to transform it after the import.

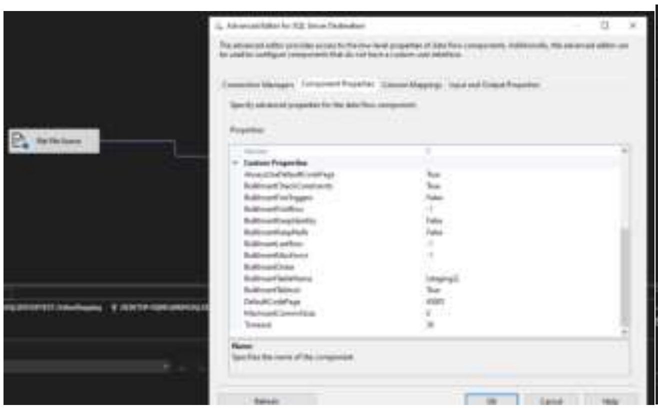
With more details presented in the import, we made this choices for the columns:

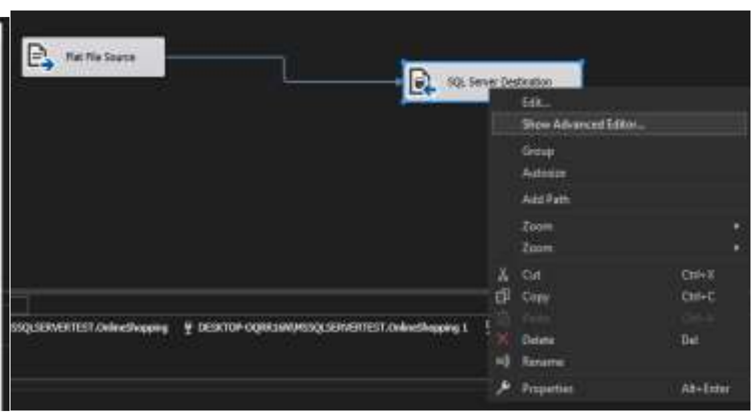
CustomerID: **DT_I8** (eight-byte signed integer)
Gender: This is a string with a single character, **DT_STR** with a length of 1 (but we put 2, for the NA value)
Location: **DT_STR** Width 100
Tenure_Months: **DT_R8** (double-precision floating point).
Transaction_ID: a numeric value **DT_I8**
Transaction_Date: **DT_STR**
Product_SKU: **DT_STR**
Product_Description: **DT_STR**
Product_Category: **DT_STR**
Quantity: **DT_R8**
Avg_Price: As prices often have decimals, **DT_R8**

Delivery_Charges: **DT_R8**
Coupon_Status: **DT_STR**
GST: This is a numeric value that have decimals, so **DT_R8**
Offline_Spend and **Online_Spend**: decimal numbers **DT_R8**
Month: Generally an integer representing the month number **DT_I8** (in order to include and values that are maybe wrong and not see an error)
Coupon_Code: a string, so **DT_STR**
Discount_pct: **DT_R8**



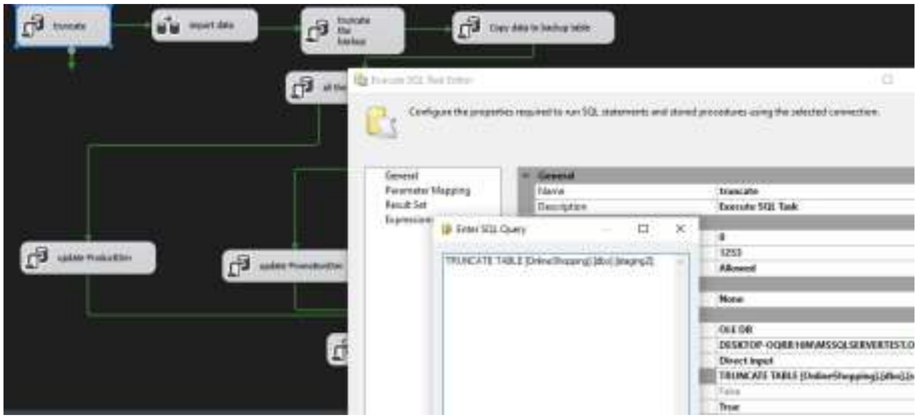
In the end, in order to run the data flow successfully, we opened the following editor in the destination and we changed the default code page and the AlwaysUseDefaultCodePage to <u>TRUE</u>:





Truncate (the table that we imported)

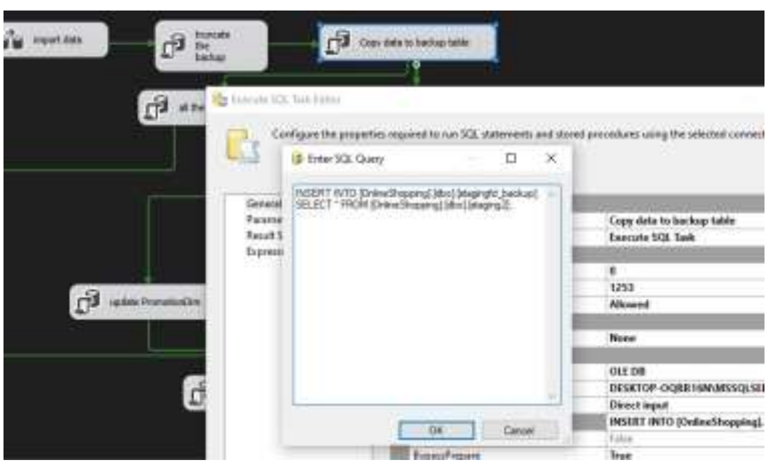
In this step we truncated the table that we entered in the SSMS from the csv file. This is done to ensure that each time the ETL process runs, it starts with an empty table. Truncating the table prevents the addition of duplicate rows (existing lines are not added below) and keeps the data fresh and up-to-date, enhancing the efficiency and accuracy of our data management process.



Create a copy (copy data to backup table)

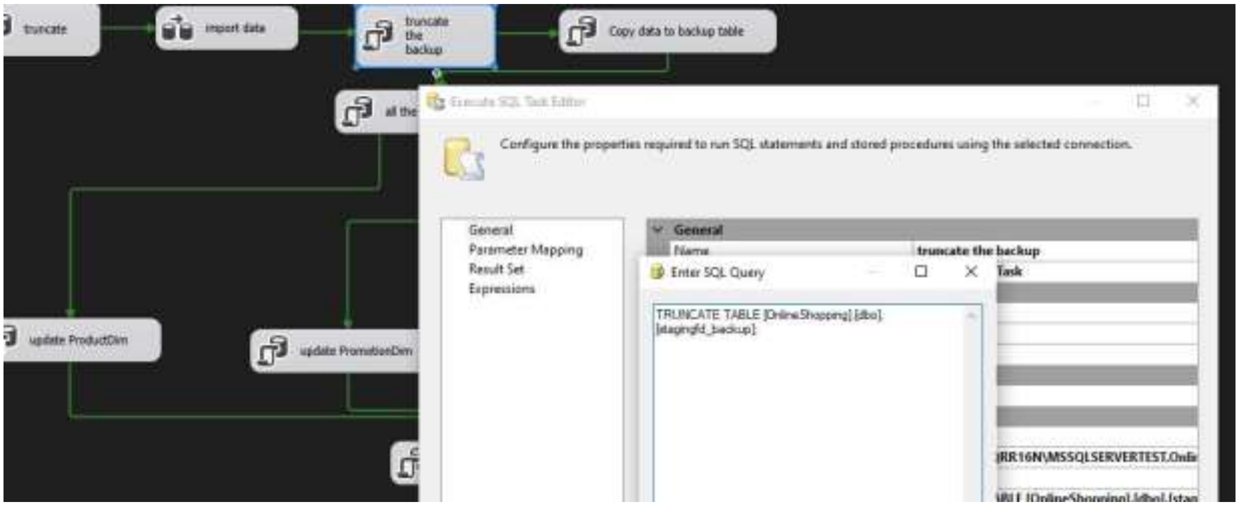
We created a copy of our table to clean it and also keep a backup of our first table for security:
In most cases for data cleaning, especially if you are going to modify the data, creating a copy of the table is the preferred approach. It gives you the flexibility to manipulate the data directly and maintain a safe backup of the original state. Views are better suited for situations where you only need to read and transform data on the fly without the need for direct data manipulation. We first created the backup table in SSMS and after that we included in the SSIS flow.

```
SELECT *
INTO [OnlineShopping].[dbo].[stagingfd_backup]
FROM [OnlineShopping].[dbo].[staging2];
```



Truncate the backup table

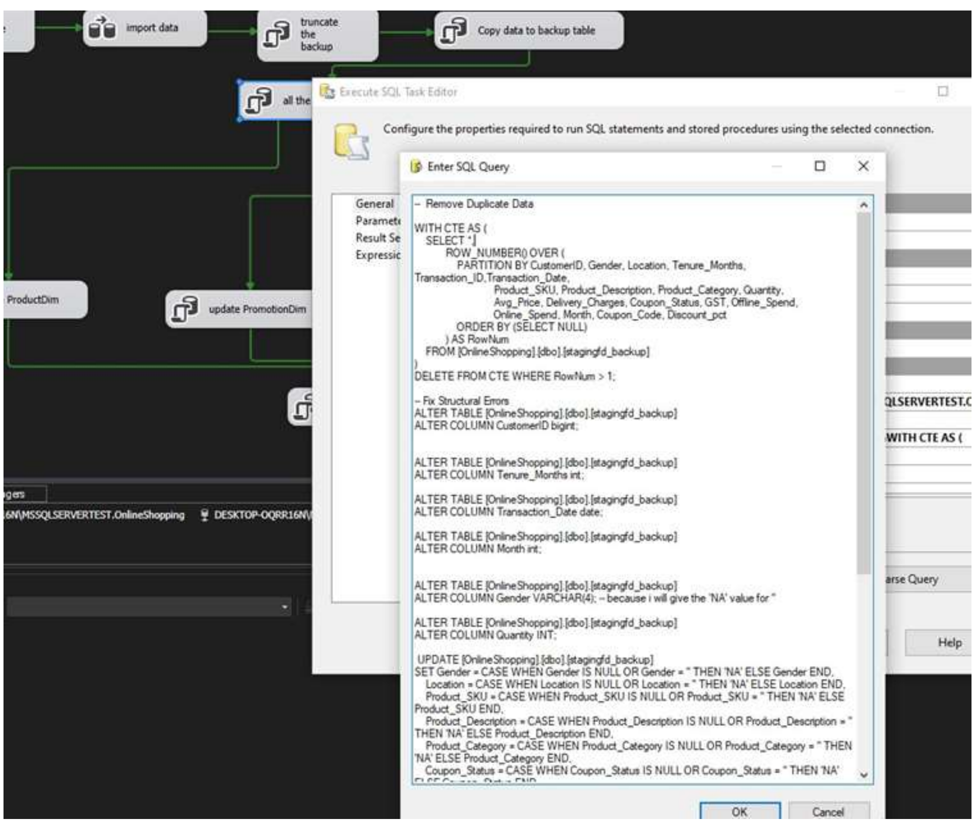
This is done to ensure that each time the ETL process runs, it starts with an empty backup table.



Transformations and Cleaning in the backup table

In this phase, the extracted data is transformed into a format suitable for analysis and reporting. We did this step 2 times, one in the import data step but after that we went in the SSMS and we found that the columns didn't have a proper type again so we did one more SQL task query in order to do to all the transformation and cleaning .

We ran each query first in SSMS and after we added it in our ETL process (as you can see in the picture):



The steps that we followed for the cleaning:

Step 1: Remove Irrelevant Data (we have removed those columns that we don't need, from before in SSIS)

Step 2: Remove Duplicate Data

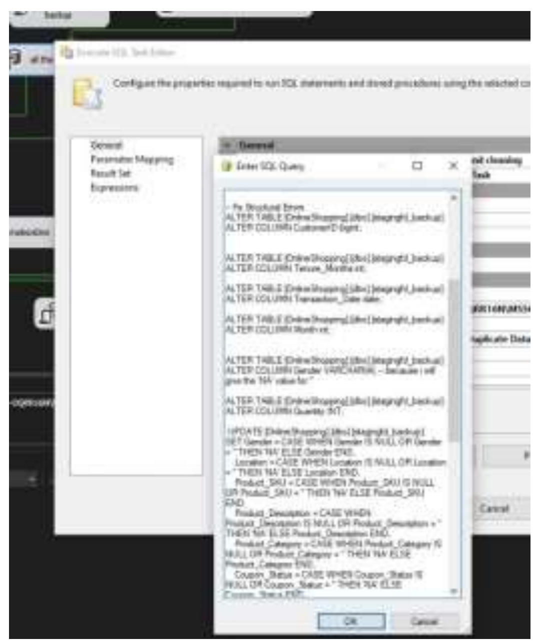
```
WITH CTE AS (
   SELECT *,
         ROW_NUMBER() OVER (
            PARTITION BY CustomerID, Gender, Location, Tenure_Months, Transaction_ID,Transaction_Date,
                     Product_SKU, Product_Description, Product_Category, Quantity,
                     Avg_Price, Delivery_Charges, Coupon_Status, GST, Offline_Spend,
                     Online_Spend, Month, Coupon_Code, Discount_pct
            ORDER BY (SELECT NULL)
         ) AS RowNum
   FROM [OnlineShopping].[dbo].[stagingfd_backup]
)
DELETE FROM CTE WHERE RowNum > 1;        --we don’t have any duplicates 0 rows affected
```

Step 3: Fix Structural Errors

Correct misspellings or wrongly classified data. Specific queries depend on the nature of errors.
We didn’t have structural errors.

Step 4: Convert Types

Every “ALTER TABLE….” query is included in the SSIS process that we have as you can see in the picture.



First we see the types

```
SELECT COLUMN_NAME, DATA_TYPE
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'stagingfd_backup'
AND TABLE_SCHEMA = 'dbo'
AND TABLE_CATALOG = 'OnlineShopping';
```

We changed some columns:
INT or **BIGINT** for whole number identifiers and counts.
DATE or **DATETIME** for dates.
DECIMAL or **NUMERIC** for financial columns
VARCHAR for textual or categorical data.

Change CustomerID to bigint:

```
ALTER TABLE [OnlineShopping].[dbo].[stagingfd_backup]
ALTER COLUMN CustomerID bigint;
```

Change Tenure_Months to int:

```
ALTER TABLE [OnlineShopping].[dbo].[stagingfd_backup]
ALTER COLUMN Tenure_Months int;
```

Change Transaction_Date to datetime:

Check the Current Format of Transaction_Date:

```
SELECT DISTINCT TOP 10 Transaction_Date
```

FROM [OnlineShopping].[dbo].[stagingfd_backup];   --12/22/19 so 'MM/DD/YYYY'

Select rows where conversion to datetime fails:

```
SELECT
   Transaction_Date
FROM
   [OnlineShopping].[dbo].[stagingfd_backup]
WHERE
   TRY_CONVERT(datetime, Transaction_Date) IS NULL
   AND Transaction_Date IS NOT NULL;           --everything is ok so i will move to change Transaction_Date
```

```
ALTER TABLE [OnlineShopping].[dbo].[stagingfd_backup]
ALTER COLUMN Transaction_Date date;
```

Change Month to int (assuming it represents the month number):

```
ALTER TABLE [OnlineShopping].[dbo].[stagingfd_backup]
ALTER COLUMN Month int;
```

```
ALTER TABLE [OnlineShopping].[dbo].[stagingfd_backup]
ALTER COLUMN Gender VARCHAR(2);              -- because i will give the 'NA' value and I don’t have only “F”,”M”
```

If the query returns no rows, it means all Quantity values are integers. If it returns any rows, those are the cases where Quantity contains non-integer values.

```
  SELECT *
FROM [OnlineShopping].[dbo].[stagingfd_backup]
WHERE CAST(Quantity AS INT) != Quantity;          --ok and know i want to alter from float to int
```

```
ALTER TABLE [OnlineShopping].[dbo].[stagingfd_backup]
ALTER COLUMN Quantity INT;
```

So , lets check if everything is ok now lets see the types

```
SELECT COLUMN_NAME, DATA_TYPE
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'stagingfd_backup'
AND TABLE_SCHEMA = 'dbo'
AND TABLE_CATALOG = 'OnlineShopping';     -- everything ok
```

Step 5: Handle Missing Data (Completeness checks)

R Environment:
R differentiates between NA (Not Available) and empty strings (""). NaN(not a number) NULL(a variable that holds no data at all)
The R code we ran checks for both NA values and empty strings, and it seems the dataset contains a mix of these.
SQL Server Environment:
SQL differentiates between NULL values and empty strings, but it does not have a direct equivalent of R's NA.
In the SQL query, we only checked for NULL values. If your data has empty strings (''), they won't be counted as NULL.
So we should adjust the SQL query to check for both NULL and empty strings

Find out if we have NULL or empty

```
SELECT
  COUNT(CASE WHEN CustomerID IS NULL OR CustomerID = '' THEN 1 END) AS MissingCustomerIDs,
-- 31
  COUNT(CASE WHEN Gender IS NULL OR Gender = '' THEN 1 END) AS MissingGender,
-- 31
  COUNT(CASE WHEN Location IS NULL OR Location = '' THEN 1 END) AS MissingLocation,
-- 31
  COUNT(CASE WHEN Tenure_Months IS NULL OR CAST(Tenure_Months AS VARCHAR) = '' THEN 1 END) AS
MissingTenureMonths,        -- 0
  COUNT(CASE WHEN Transaction_ID IS NULL OR CAST(Transaction_ID AS VARCHAR) = '' THEN 1 END) AS
MissingTransactionIDs,    -- 0
  COUNT(CASE WHEN Transaction_Date IS NULL OR Transaction_Date = '' THEN 1 END) AS
MissingTransactionDate,             -- 31
  COUNT(CASE WHEN Product_SKU IS NULL OR Product_SKU = '' THEN 1 END) AS MissingProductSKU,
-- 31
  COUNT(CASE WHEN Product_Description IS NULL OR Product_Description = '' THEN 1 END) AS
MissingProductDescription,        -- 31
  COUNT(CASE WHEN Product_Category IS NULL OR Product_Category = '' THEN 1 END) AS
MissingProductCategory,               --144
  COUNT(CASE WHEN Quantity IS NULL OR CAST(Quantity AS VARCHAR) = '' THEN 1 END) AS
MissingQuantity,                    -- 0
  COUNT(CASE WHEN Avg_Price IS NULL OR CAST(Avg_Price AS VARCHAR) = '' THEN 1 END) AS
MissingAvgPrice,                    -- 0
  COUNT(CASE WHEN Delivery_Charges IS NULL OR CAST(Delivery_Charges AS VARCHAR) = '' THEN 1 END)
AS MissingDeliveryCharges,    -- 0
  COUNT(CASE WHEN Coupon_Status IS NULL OR Coupon_Status = '' THEN 1 END) AS MissingCouponStatus,
--175
  COUNT(CASE WHEN GST IS NULL OR CAST(GST AS VARCHAR) = '' THEN 1 END) AS MissingGST,
-- 0
  COUNT(CASE WHEN Offline_Spend IS NULL OR CAST(Offline_Spend AS VARCHAR) = '' THEN 1 END) AS
MissingOfflineSpend,           -- 0
  COUNT(CASE WHEN Online_Spend IS NULL OR CAST(Online_Spend AS VARCHAR) = '' THEN 1 END) AS
MissingOnlineSpend,            -- 0
  COUNT(CASE WHEN Month IS NULL OR CAST(Month AS VARCHAR) = '' THEN 1 END) AS MissingMonth,
-- 0
  COUNT(CASE WHEN Coupon_Code IS NULL OR Coupon_Code = '' THEN 1 END) AS MissingCouponCode,
--544
  COUNT(CASE WHEN Discount_pct IS NULL OR CAST(Discount_pct AS VARCHAR) = '' THEN 1 END) AS
MissingDiscountPct              --0
FROM [OnlineShopping].[dbo].[stagingfd_backup];
```

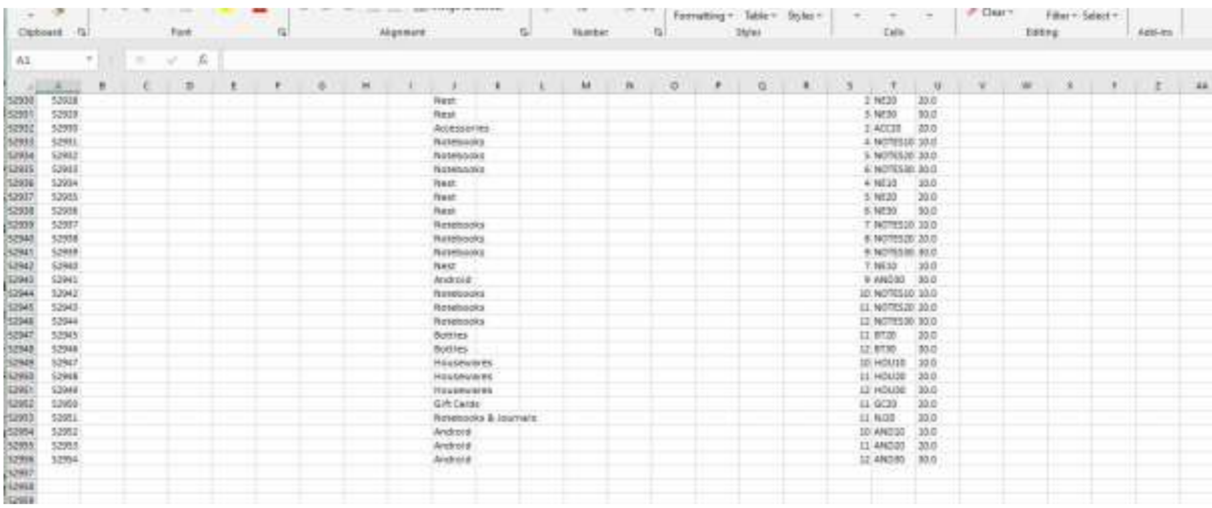
We want to ensure that our data cleaning or imputation strategies do not inadvertently introduce biases or inaccuracies. And also we document any change that we make for future reference and to maintain transparency in our data handling processes

Decision: we will fill with NULL the metrics and with NA the dimensions that have missing values (We have CustomerID and Month as int, so they will not be NA but NULL) :

```
 UPDATE [OnlineShopping].[dbo].[stagingfd_backup]
SET Gender = CASE WHEN Gender IS NULL OR Gender = '' THEN 'NA' ELSE Gender END,
    Location = CASE WHEN Location IS NULL OR Location = '' THEN 'NA' ELSE Location END,
    Product_SKU = CASE WHEN Product_SKU IS NULL OR Product_SKU = '' THEN 'NA' ELSE Product_SKU END,
    Product_Description = CASE WHEN Product_Description IS NULL OR Product_Description = '' THEN 'NA' ELSE
Product_Description END,
```

```
    Product_Category = CASE WHEN Product_Category IS NULL OR Product_Category = '' THEN 'NA' ELSE Product_Category END,
    Coupon_Status = CASE WHEN Coupon_Status IS NULL OR Coupon_Status = '' THEN 'NA' ELSE Coupon_Status END,
    Coupon_Code = CASE WHEN Coupon_Code IS NULL OR Coupon_Code = '' THEN 'NA' ELSE Coupon_Code END;

UPDATE [OnlineShopping].[dbo].[stagingfd_backup]
SET Tenure_Months = CASE WHEN Tenure_Months IS NULL THEN NULL ELSE Tenure_Months END,
    Transaction_ID = CASE WHEN Transaction_ID IS NULL THEN NULL ELSE Transaction_ID END,
    Quantity = CASE WHEN Quantity IS NULL THEN NULL ELSE Quantity END,
    Avg_Price = CASE WHEN Avg_Price IS NULL THEN NULL ELSE Avg_Price END,
    Delivery_Charges = CASE WHEN Delivery_Charges IS NULL THEN NULL ELSE Delivery_Charges END,
    GST = CASE WHEN GST IS NULL THEN NULL ELSE GST END,
    Offline_Spend = CASE WHEN Offline_Spend IS NULL THEN NULL ELSE Offline_Spend END,
    Online_Spend = CASE WHEN Online_Spend IS NULL THEN NULL ELSE Online_Spend END,
    Discount_pct = CASE WHEN Discount_pct IS NULL THEN NULL ELSE Discount_pct END,
    CustomerID = CASE WHEN CustomerID IS NULL OR CustomerID = '' THEN NULL ELSE CustomerID END,
    Month = CASE WHEN Month IS NULL OR Month = '' THEN NULL ELSE Month END;
```

In this excel, the last 31 lines has almost no values in any column so we decided to extract them:



```
DELETE FROM [OnlineShopping].[dbo].[stagingfd_backup]
WHERE CustomerID IS NULL OR CustomerID = '';                --31 rows affected
```

<u>Step 6:</u> Deal with Outliers

Reasons to keep them:

Business Understanding: Sometimes, outliers represent rare but important business scenarios. Understanding whether an outlier is an error or a valid data point is crucial.

Valuable Business Insights: In some cases, outliers are the most interesting part of your data. For example, unusually high sales on certain days might warrant further investigation rather than removal.

<u>Step 7</u>: Standardize/Normalize Data

Standardize: Refers to the process of bringing data into a uniform format. We didn’t have to do this because we had only one source file and it’s already normalized. We just transformed some columns types.

Step 8: Validate Data

Validation is the process of ensuring that the data is accurate and appropriate for the intended use.

This involves checking the data for accuracy, consistency, and completeness. Validation can take many forms:

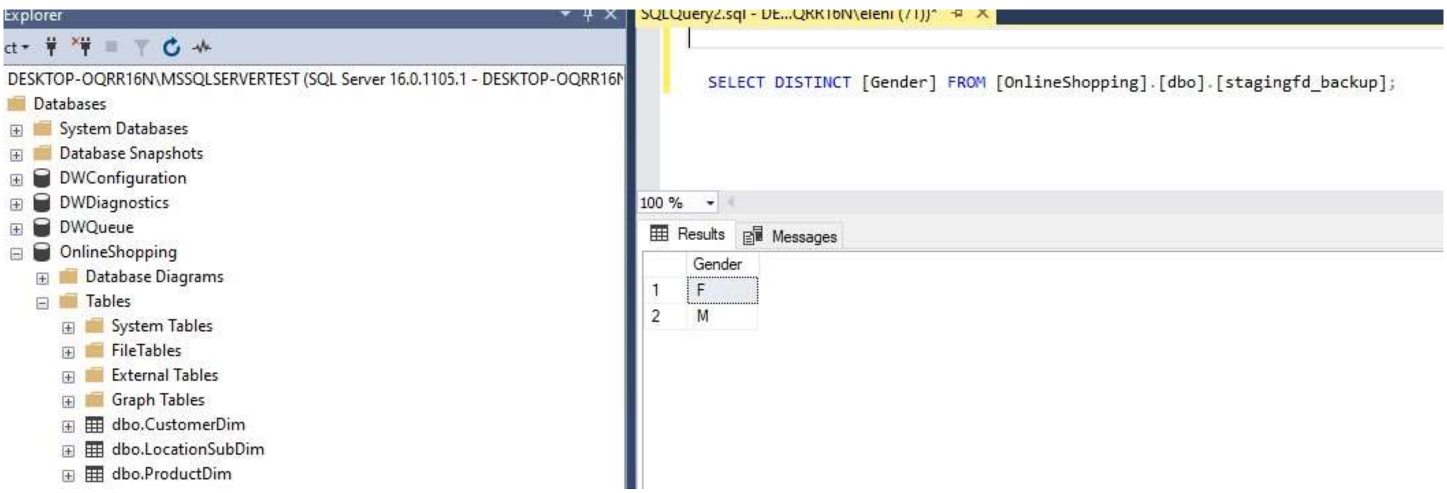
Accuracy checks ensure the data is correctly entered or imported. For example, if you have a dataset of addresses, you might check to ensure the postal codes align with the correct cities.

Consistency checks look for data that doesn't conform to the expected patterns or rules. For example, if you have a column that should only contain positive values, a consistency check would flag any negative values.

Let’s see the distinct values for each column (we did this queries in SSMS):

SELECT DISTINCT [Gender] FROM [OnlineShopping].[dbo].[stagingfd_backup]; -- only F, M (#2)

For example



SELECT DISTINCT [Location] FROM [OnlineShopping].[dbo].[stagingfd_backup]; -- Washington DC , New York , California ,New Jersey , Chicago (#5)

SELECT DISTINCT [Product_SKU] FROM [OnlineShopping].[dbo].[stagingfd_backup]; -- (#1133)

SELECT DISTINCT [Product_Description] FROM [OnlineShopping].[dbo].[stagingfd_backup]; -- (#394)

SELECT DISTINCT [Product_Category] FROM [OnlineShopping].[dbo].[stagingfd_backup]; --(#16)

SELECT DISTINCT [Coupon_Status] FROM [OnlineShopping].[dbo].[stagingfd_backup]; -- not used , used, clicked (#3)

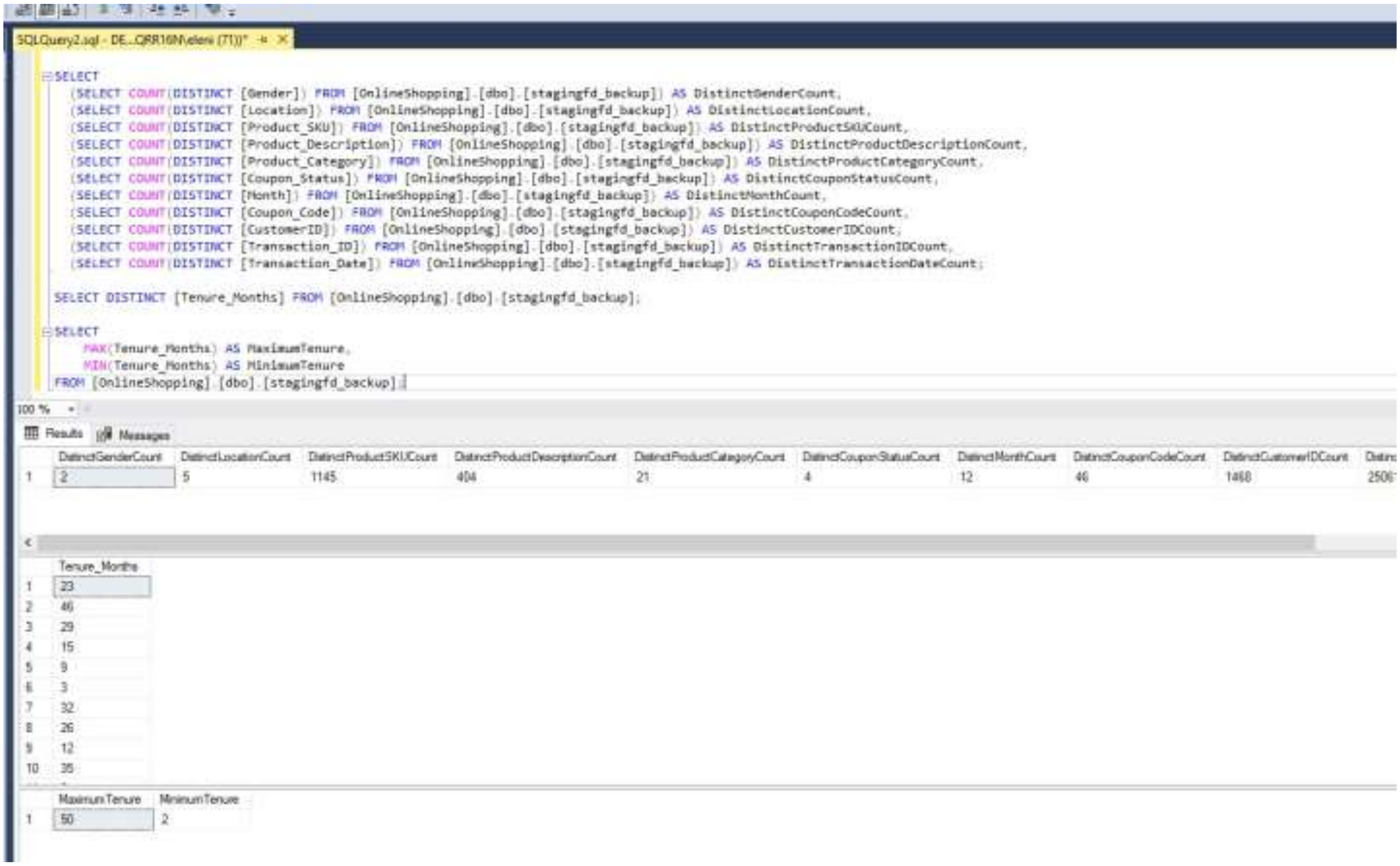
SELECT DISTINCT [Month] FROM [OnlineShopping].[dbo].[stagingfd_backup]; -- (#12)

SELECT DISTINCT [Coupon_Code] FROM [OnlineShopping].[dbo].[stagingfd_backup]; --(#45)

Let’s see how many distinct values each column has:

SELECT

```sql
  (SELECT COUNT(DISTINCT [Gender]) FROM [OnlineShopping].[dbo].[stagingfd_backup]) AS DistinctGenderCount,
  (SELECT COUNT(DISTINCT [Location]) FROM [OnlineShopping].[dbo].[stagingfd_backup]) AS DistinctLocationCount,
  (SELECT COUNT(DISTINCT [Product_SKU]) FROM [OnlineShopping].[dbo].[stagingfd_backup]) AS DistinctProductSKUCount,
  (SELECT COUNT(DISTINCT [Product_Description]) FROM [OnlineShopping].[dbo].[stagingfd_backup]) AS DistinctProductDescriptionCount,
  (SELECT COUNT(DISTINCT [Product_Category]) FROM [OnlineShopping].[dbo].[stagingfd_backup]) AS DistinctProductCategoryCount,
  (SELECT COUNT(DISTINCT [Coupon_Status]) FROM [OnlineShopping].[dbo].[stagingfd_backup]) AS DistinctCouponStatusCount,
  (SELECT COUNT(DISTINCT [Month]) FROM [OnlineShopping].[dbo].[stagingfd_backup]) AS DistinctMonthCount,
  (SELECT COUNT(DISTINCT [Coupon_Code]) FROM [OnlineShopping].[dbo].[stagingfd_backup]) AS DistinctCouponCodeCount,
  (SELECT COUNT(DISTINCT [CustomerID]) FROM [OnlineShopping].[dbo].[stagingfd_backup]) AS DistinctCustomerIDCount,
  (SELECT COUNT(DISTINCT [Transaction_ID]) FROM [OnlineShopping].[dbo].[stagingfd_backup]) AS DistinctTransactionIDCount,
  (SELECT COUNT(DISTINCT [Transaction_Date]) FROM [OnlineShopping].[dbo].[stagingfd_backup]) AS DistinctTransactionDateCount;


SELECT DISTINCT [Tenure_Months] FROM [OnlineShopping].[dbo].[stagingfd_backup];


SELECT
   MAX(Tenure_Months) AS MaximumTenure,
   MIN(Tenure_Months) AS MinimumTenure
FROM [OnlineShopping].[dbo].[stagingfd_backup];
```



Check for negative values in columns expected to have only positive values:

SELECT *

FROM [OnlineShopping].[dbo].[stagingfd_backup]

WHERE Tenure_Months < 0 OR Quantity < 0 OR Avg_Price < 0 OR Delivery_Charges < 0 OR GST < 0 OR Offline_Spend < 0 OR Online_Spend < 0; -- no negatives, everything ok

Check for Invalid Dates:

SELECT *

FROM [OnlineShopping].[dbo].[stagingfd_backup]

WHERE Transaction_Date < '2019-01-01' OR Transaction_Date > GETDATE(); -- empty table the output, everything after 2019 .

Check for Invalid Discount Percentages

SELECT *

FROM [OnlineShopping].[dbo].[stagingfd_backup]

WHERE Discount_pct < 0 OR Discount_pct > 100; --everythink ok

Check for Invalid Month Values

```
SELECT *
FROM [OnlineShopping].[dbo].[stagingfd_backup]
WHERE Month < 1 OR Month > 12;      -- everything ok
```

Checking the nature of our columns

- Check for Uniqueness of Product_SKU and for NULL values

```
SELECT Product_SKU, COUNT(*)
FROM [OnlineShopping].[dbo].[stagingfd_backup]
GROUP BY Product_SKU
HAVING COUNT(*) > 1;
```

If this query returns any records, it means there are duplicate SKUs in the dataset, and we cannot use SKU as a primary key. In our case, it did return records:
For the same Product_ID we have different Product_SKU so we will put a surrogate key with the name Product_created_ID . One more reason to use surrogate key is that we may have missing values in this column (it must be unique and not null for every record in the table)

- Check for Null Values

```
SELECT COUNT(*)
FROM [OnlineShopping].[dbo].[stagingfd_backup]
WHERE Product_SKU IS NULL;                    --this is ok after the remove of the 31 rows in the bottom of excel
```

Surrogate keys are used in database design, especially in data warehousing, for several reasons:

1. Uniqueness: A surrogate key is a unique identifier for each row in a table, which is necessary for maintaining uniqueness when natural keys (business keys) might not be unique or might change over time.

2. Consistency: Surrogate keys are consistent as they are typically auto-incremented numbers assigned by the database system. They don't carry business meaning and are not subject to the same potential volatility as natural keys.

3. Performance: Because surrogate keys are usually integers, they are generally smaller, fixed in size, and faster to join on than natural keys, which may be strings or composite keys. This can lead to performance improvements in query processing.

4. Simplicity: Surrogate keys simplify the handling of changes to natural key values, which might occur due to business operations such as data correction, changes in business terminology, mergers/acquisitions, or re-keying errors.

5. Handling of NULLs and Duplicates: Natural keys can sometimes be NULL or duplicate, which can cause problems in relational integrity and joins. Surrogate keys, being system-generated, are always present and unique.

6. Integration: In scenarios where data comes from multiple sources, surrogate keys can help integrate data that has overlapping or conflicting natural keys, ensuring a consistent key structure in the target database.

7. Abstraction: Surrogate keys abstract away the underlying business or natural keys, which can be beneficial when merging or integrating disparate systems, where business keys could potentially overlap or change.

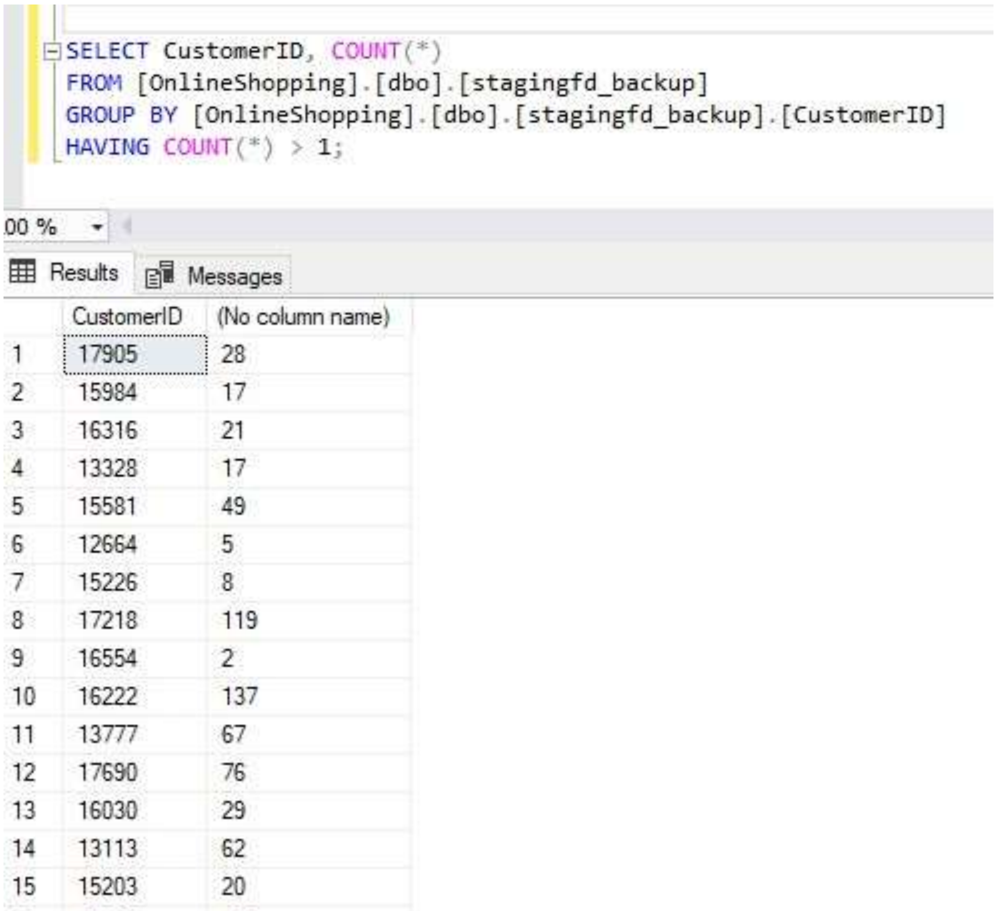
8. Slowly Changing Dimensions: In data warehousing, dimensions often change over time (e.g., a product description or a customer address). Surrogate keys allow for the implementation of slowly changing dimensions, where historical data can be preserved alongside current data without ambiguity.

➢ Check if we can use the CustomerID as Primary key

--Check for Uniqueness

SELECT CustomerID, COUNT(*)
FROM [OnlineShopping].[dbo].[stagingfd_backup]
GROUP BY CrustomerID
HAVING COUNT(*) > 1;

If this query returns any records, it means there are duplicate CustomerID in the dataset, and we cannot use CustomerID as a primary key. As you can see in the bellow screenshot we should use a surrogate key (Customer_created_ID) .



```
SELECT CustomerID, COUNT(*)
FROM [OnlineShopping].[dbo].[stagingfd_backup]
GROUP BY [OnlineShopping].[dbo].[stagingfd_backup].[CustomerID]
HAVING COUNT(*) > 1;
```


	CustomerID	(No column name)
1	17905	28
2	15984	17
3	16316	21
4	13328	17
5	15581	49
6	12664	5
7	15226	8
8	17218	119
9	16554	2
10	16222	137
11	13777	67
12	17690	76
13	16030	29
14	13113	62
15	15203	20

➢ Check for Null Values

SELECT COUNT(*)
FROM [OnlineShopping].[dbo].[stagingfd_backup]
WHERE CustomerID IS NULL; --this is ok

➢ Check for Slowly changing dimensions

If each CustomerID is refer always to the same location.

SELECT
[CustomerID],
COUNT(DISTINCT [Location]) AS DistinctLocationCount
FROM
[OnlineShopping].[dbo].[stagingfd_backup]
GROUP BY

```
    [CustomerID]
HAVING
    COUNT(DISTINCT [Location]) > 1;   --our customers don't change locations , we don't have a Slowly Changing Dimension (SCD)
```

- There are multiple products per transaction , so multiple rows might share the same Transaction_ID

```
SELECT Transaction_ID, COUNT(DISTINCT Product_SKU) as Unique_Products_Per_Transaction
FROM [OnlineShopping].[dbo].[stagingfd_backup]
GROUP BY Transaction_ID;
```

- there are multiple transactions per CustomerID

```
SELECT CustomerID, COUNT(DISTINCT Transaction_ID) as Total_Transactions
FROM [OnlineShopping].[dbo].[stagingfd_backup]
GROUP BY CustomerID;
```

- there multiple products per product category

```
SELECT Product_Category, COUNT(*) as Total_Products
FROM [OnlineShopping].[dbo].[stagingfd_backup]
GROUP BY Product_Category;
```

- there are multiple tranasactions per month

```
SELECT Month, COUNT(DISTINCT Transaction_ID) as Total_Transactions
FROM [OnlineShopping].[dbo].[stagingfd_backup]
GROUP BY Month;
```

- the total sales per Discount_pct

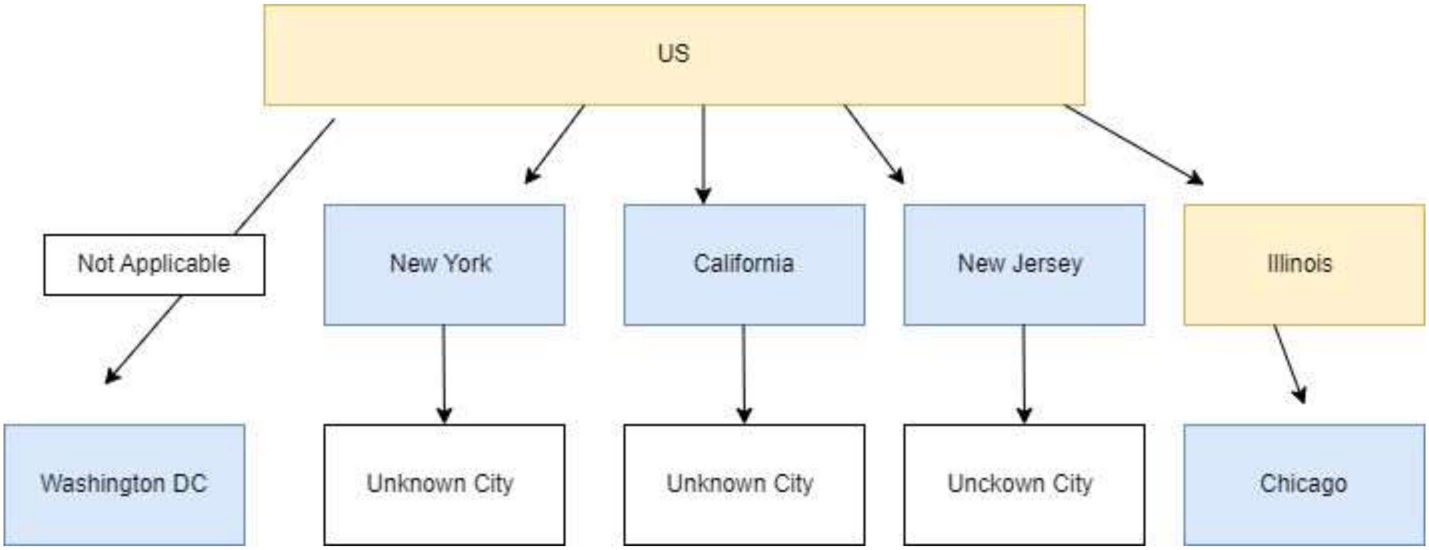
```
SELECT Discount_pct, SUM(Avg_Price * Quantity) as Total_Sales
FROM [OnlineShopping].[dbo].[stagingfd_backup]
GROUP BY Discount_pct;
```

- coupon status frequency

```
SELECT Coupon_Status, COUNT(*) as Frequency
FROM [OnlineShopping].[dbo].[stagingfd_backup]
GROUP BY Coupon_Status;
```

- Handling Unbalanced and Non Covering Location Hierarchy

1. Washington DC: This is a city, but it's also a federal district known as the District of Columbia. It serves as the capital of the United States and is not part of any state.
2. New York: This can refer to the state of New York
3. California: This is a state on the west coast of the United States.
4. New Jersey: This is a state
5. Chicago: This is a city, specifically the largest city in the state of Illinois.



We will handle the Unbalanced Location Hierarchy with Placeholder values. In the Location sub-dimension we will have Country -> State -> City. So “USA” will be the value for Country, Washington DC will have “Not Applicable” as State and for Chicago we will put the value “Illinois” in State.

- 1133 different Product_SKU in our dataset
- 394 different product Descriptions
- 16 different Product categories (one of them is NA)

```
SELECT  DISTINCT [Product_SKU]  FROM [OnlineShopping].[dbo].[stagingfd_backup]   -- 1133 rows

SELECT DISTINCT  [Product_Description] FROM [OnlineShopping].[dbo].[stagingfd_backup]    --394 rows

SELECT DISTINCT  [Product_Category] FROM [OnlineShopping].[dbo].[stagingfd_backup]      --16 rows
```

Update sub-dimension and Dimension tables

- Initially, we use 'CREATE TABLE' statements in SSMS to establish the necessary tables. Following their creation, we populate these tables with the relevant data using 'INSERT' queries. Additionally, we include these 'INSERT' queries in the SSIS control flow. This integration with SSIS is crucial as it allows for the automatic updating of our dimension tables each time the ETL (Extract, Transform, Load) process is executed. By doing this, we ensure that the tables are consistently refreshed with new data and kept up-to-date.

ProductDim

CREATE TABLE ProductDim (

Product_created_ID INT IDENTITY(1,1) PRIMARY KEY, -- surrogate key, are unique identifiers within the data warehouse and are used primarily for internal references.

Product_SKU VARCHAR(300),

Product_Description NVARCHAR(900),

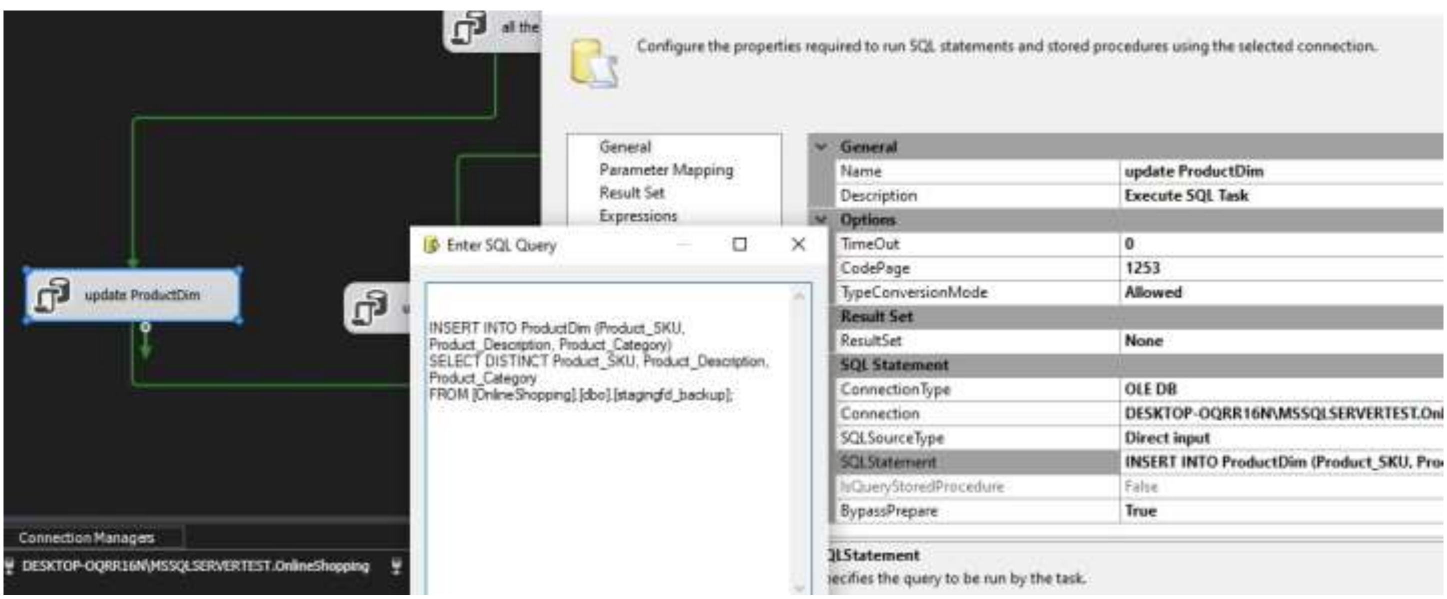
Product_Category NVARCHAR(300)

);

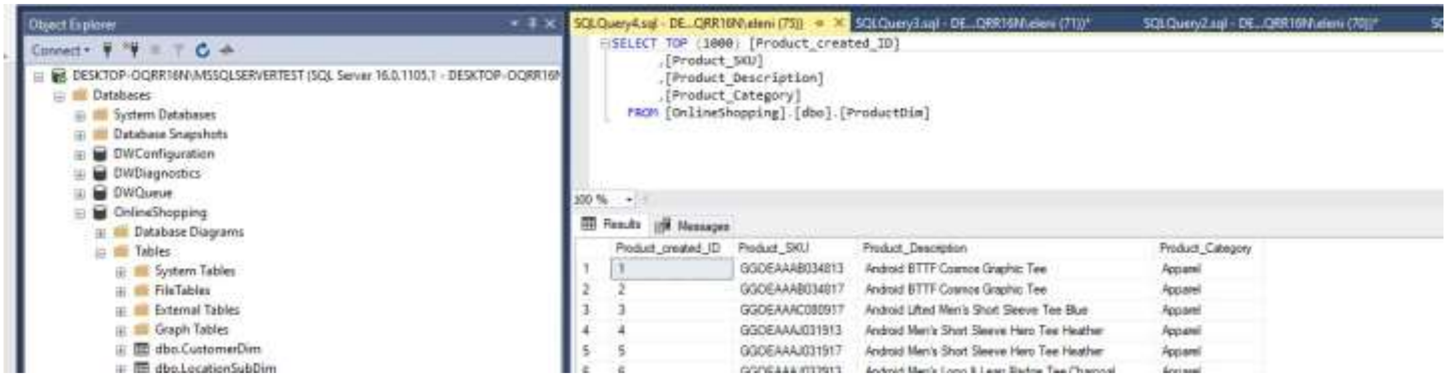
INSERT INTO ProductDim (Product_SKU, Product_Description, Product_Category)

SELECT DISTINCT Product_SKU, Product_Description, Product_Category

FROM [OnlineShopping].[dbo].[stagingfd_backup]; -- 1145 rows affected



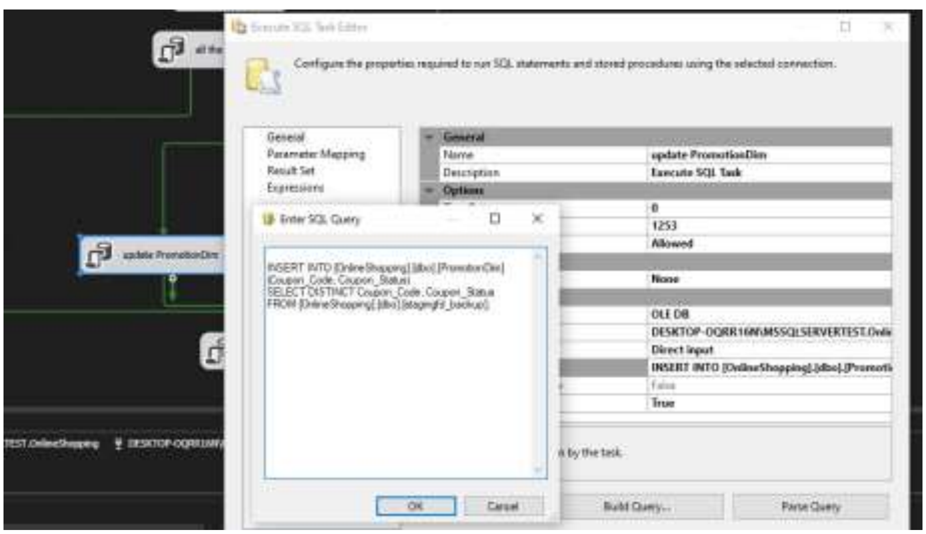
In the SSMS looks like this (we display only the top rows):



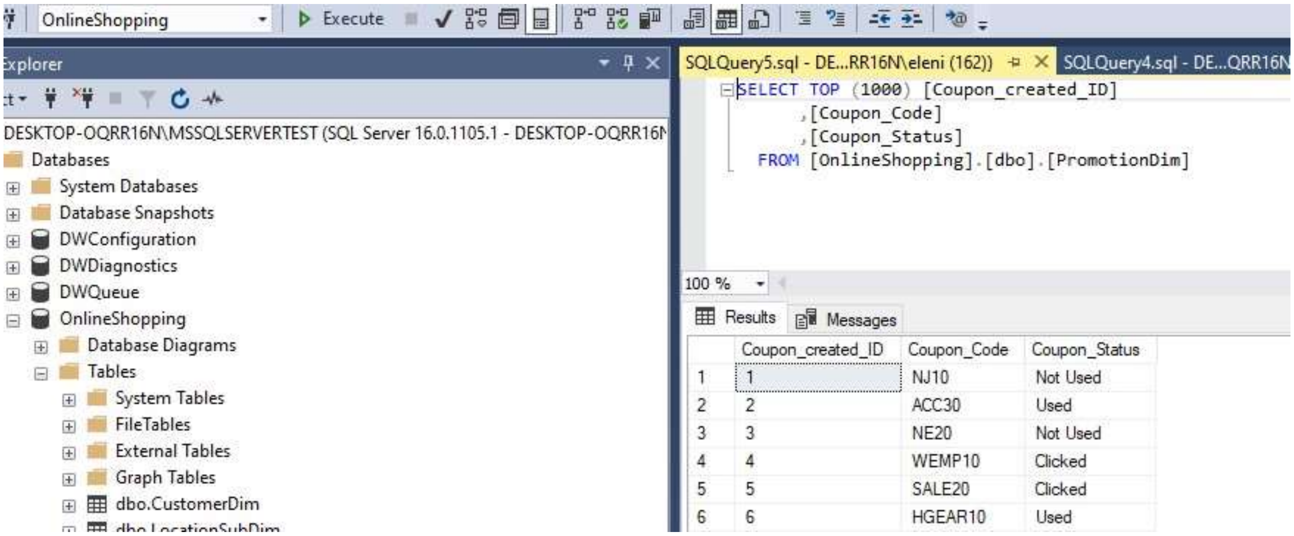
PromotionDim

```
CREATE TABLE [OnlineShopping].[dbo].[PromotionDim] (
    Coupon_created_ID INT IDENTITY(1,1) PRIMARY KEY,
    Coupon_Code VARCHAR(300),
    Coupon_Status VARCHAR(300)
);
```

```
INSERT INTO [OnlineShopping].[dbo].[PromotionDim] (Coupon_Code, Coupon_Status)
SELECT DISTINCT Coupon_Code, Coupon_Status
FROM [OnlineShopping].[dbo].[stagingfd_backup];  -- 156 rows affected
```



In the SSMS looks like this (we display only the top rows):



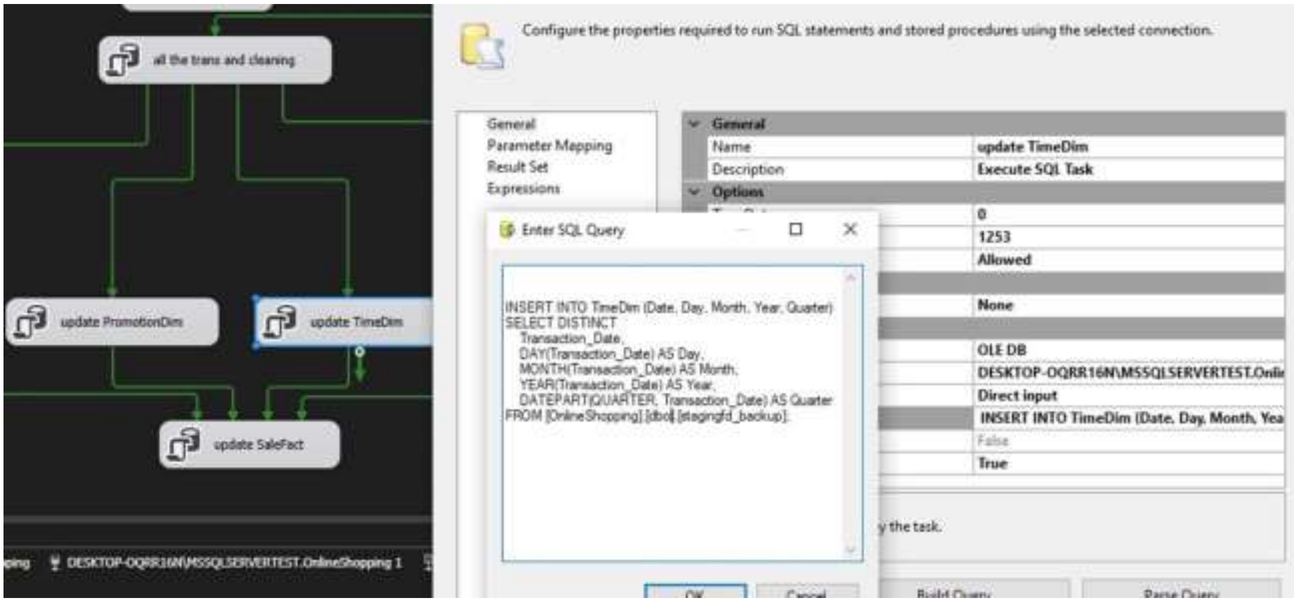
<u>TimeDim</u>

```
CREATE TABLE TimeDim (
    Transaction_Date_created_ID INT IDENTITY(1,1) PRIMARY KEY,
    Date DATE,
    Day INT,
    Month INT,
    Year INT,
    Quarter INT
);
```

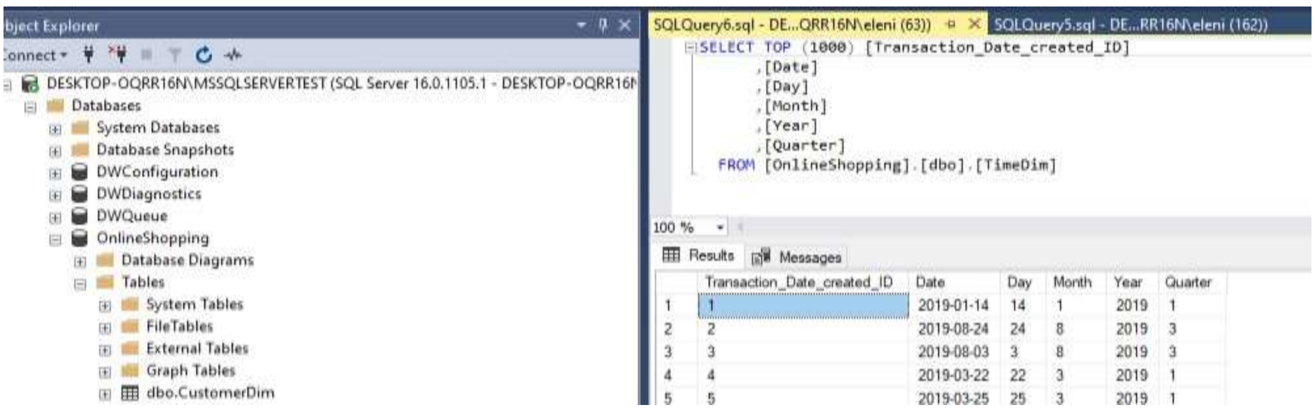
```
INSERT INTO TimeDim (Date, Day, Month, Year, Quarter)
SELECT DISTINCT
    Transaction_Date,              -- Unique dates from your staging table
    DAY(Transaction_Date) AS Day,    -- Extracts the day part from the date
    MONTH(Transaction_Date) AS Month,-- Extracts the month part from the date
    YEAR(Transaction_Date) AS Year,  -- Extracts the year part from the date
    DATEPART(QUARTER, Transaction_Date) AS Quarter -- Extracts the quarter part from the date
FROM [OnlineShopping].[dbo].[stagingfd_backup];       -- 365 rows affected
```

--SELECT DISTINCT: This part of the query selects unique (distinct) rows based on the Transaction_Date. It ensures that each date is only inserted once into the TimeDim table.

-- DAY/MONTH/YEAR/DATEPART(QUARTER, ...): These functions extract the respective date parts from each Transaction_Date. This breakdown allows for detailed time-based analysis in our data warehouse.



In the ssms looks like (we dispay only the top rows) :

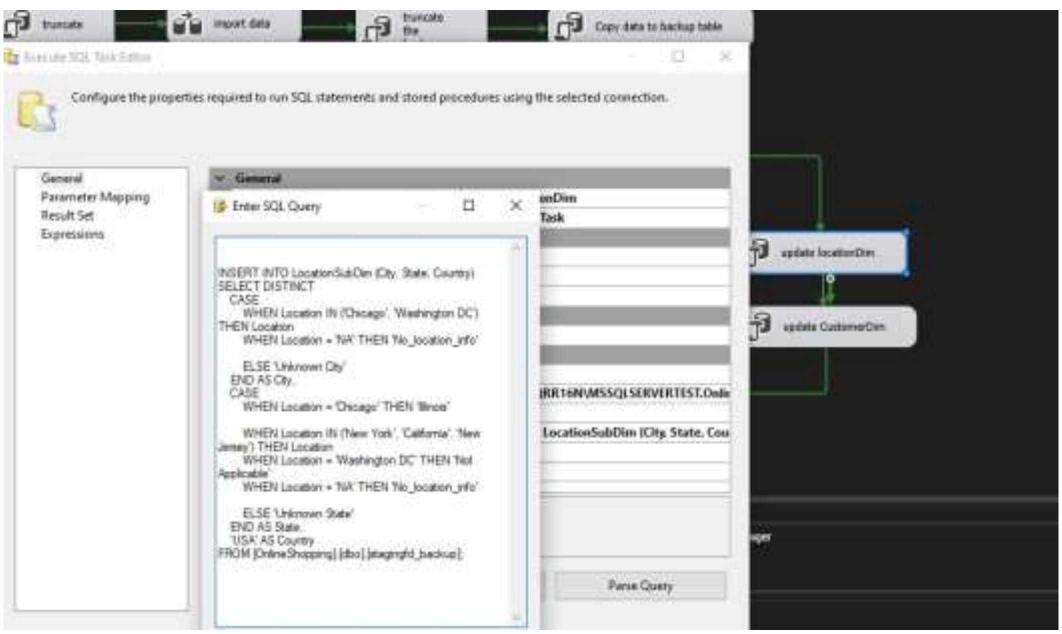


<u>LocationSubDim</u>

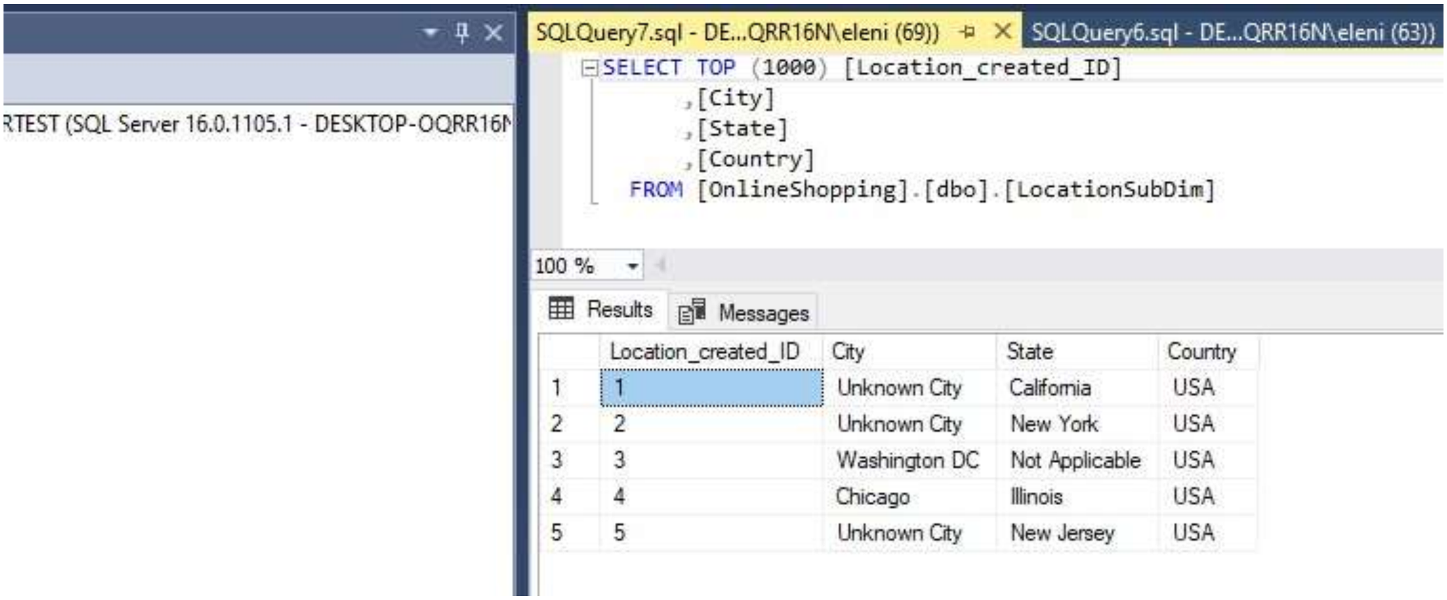
```
CREATE TABLE LocationSubDim (
    Location_created_ID INT IDENTITY(1,1) PRIMARY KEY,
    City VARCHAR(300) DEFAULT 'Unknown City',          -- Placeholder for customers who we don't know the city
    State VARCHAR(300) DEFAULT 'Not Applicable',       -- Placeholder for cities without a State
    Country VARCHAR(300) DEFAULT 'USA'                 -- Assuming all locations are in the USA
);
```

```
INSERT INTO LocationSubDim (City, State, Country)
SELECT DISTINCT
    CASE
        WHEN Location IN ('Chicago', 'Washington DC') THEN Location
        WHEN Location = 'NA' THEN 'No_location_info'                -- Handling 'NA'
```

```
    ELSE 'Unknown City'
  END AS City,
  CASE
    WHEN Location = 'Chicago' THEN 'Illinois'                                    -- Setting State to Illinois when city is Chicago
    WHEN Location IN ('New York', 'California', 'New Jersey') THEN Location
    WHEN Location = 'Washington DC' THEN 'Not Applicable'                        -- Washington DC is not part of any State
    WHEN Location = 'NA' THEN 'No_location_info'                                 -- Handling 'NA'
    ELSE 'Unknown State'
  END AS State,
  CASE
    WHEN Location = 'NA' THEN 'No_location_info'                                 -- Handling 'NA' for Country
    ELSE 'USA'                                                                   -- Default country
  END AS Country
FROM [OnlineShopping].[dbo].[stagingfd_backup];


-- 5 rows affected
```



In the SSMS looks like this:

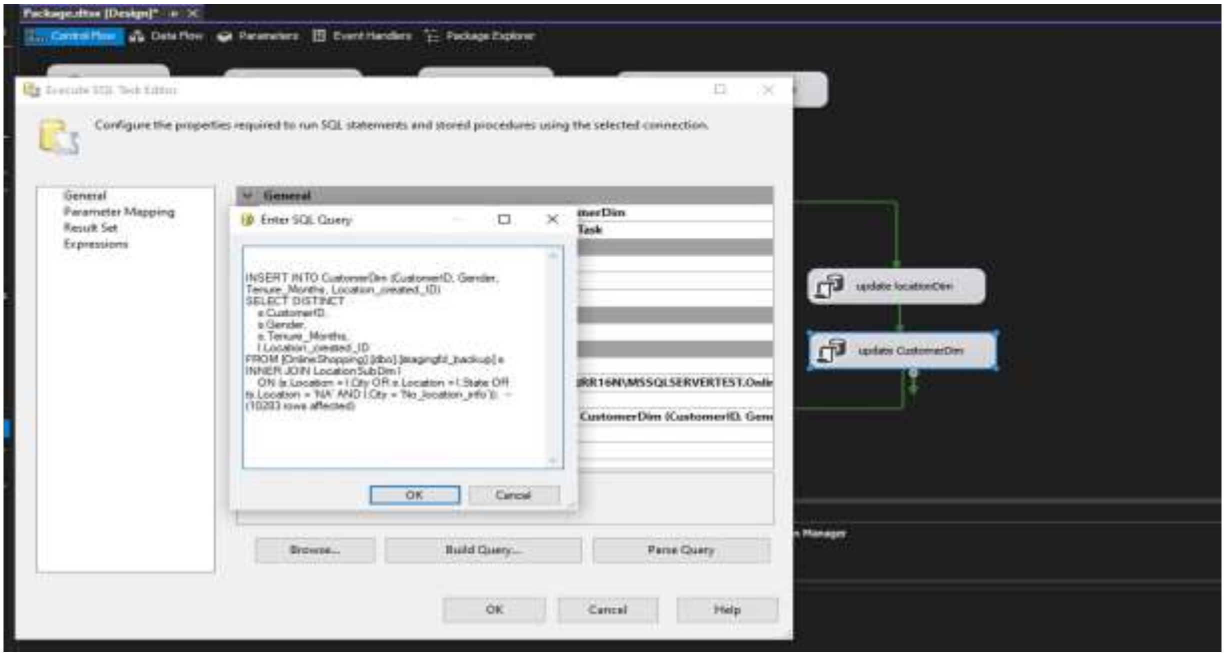


<u>CustomerDim</u>

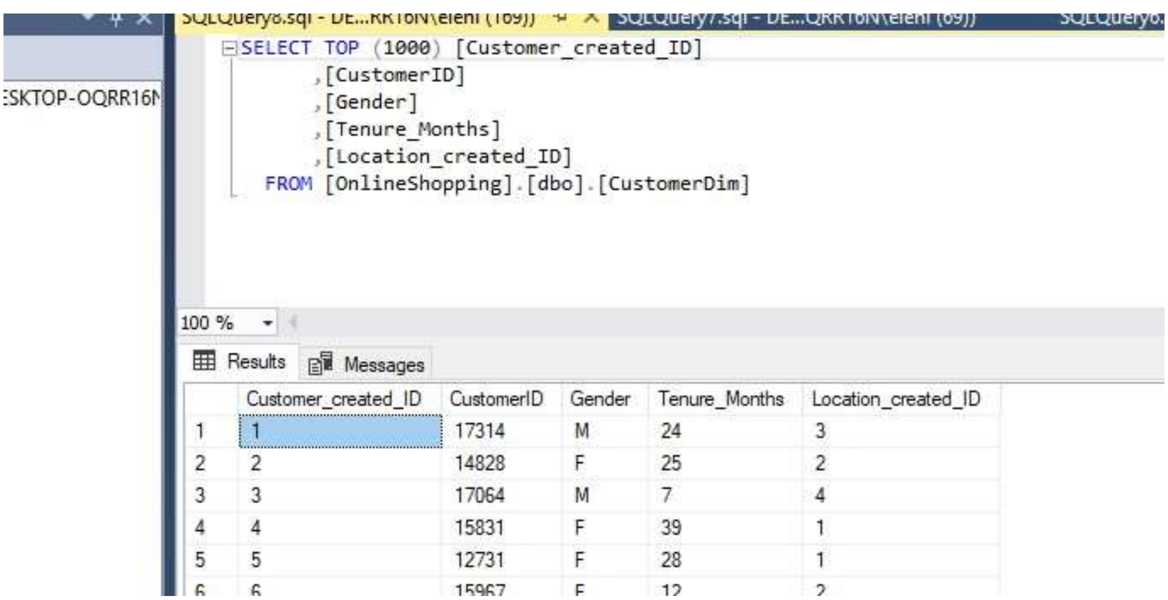
```
CREATE TABLE CustomerDim (
    Customer_created_ID INT IDENTITY(1,1) PRIMARY KEY,                 -- Auto-incrementing primary key
    CustomerID INT,
    Gender VARCHAR(4),
    Tenure_Months INT,
     Location_created_ID INT                          -- This will link to the Location Sub-Dimension Table

FOREIGN KEY (Location_created_ID) REFERENCES LocationSubDim(Location_created_ID)
);
```

```
INSERT INTO CustomerDim (CustomerID, Gender, Tenure_Months, Location_created_ID)
SELECT DISTINCT
    s.CustomerID,
    s.Gender,
    s.Tenure_Months,
    l.Location_created_ID
FROM [OnlineShopping].[dbo].[stagingfd_backup] s
INNER JOIN LocationSubDim l
```

ON (s.Location = l.City OR s.Location = l.State OR (s.Location = 'NA' AND l.City = 'No_location_info')); --(1468 rows affected)



In the SSMS looks like this (we display only the top rows) :



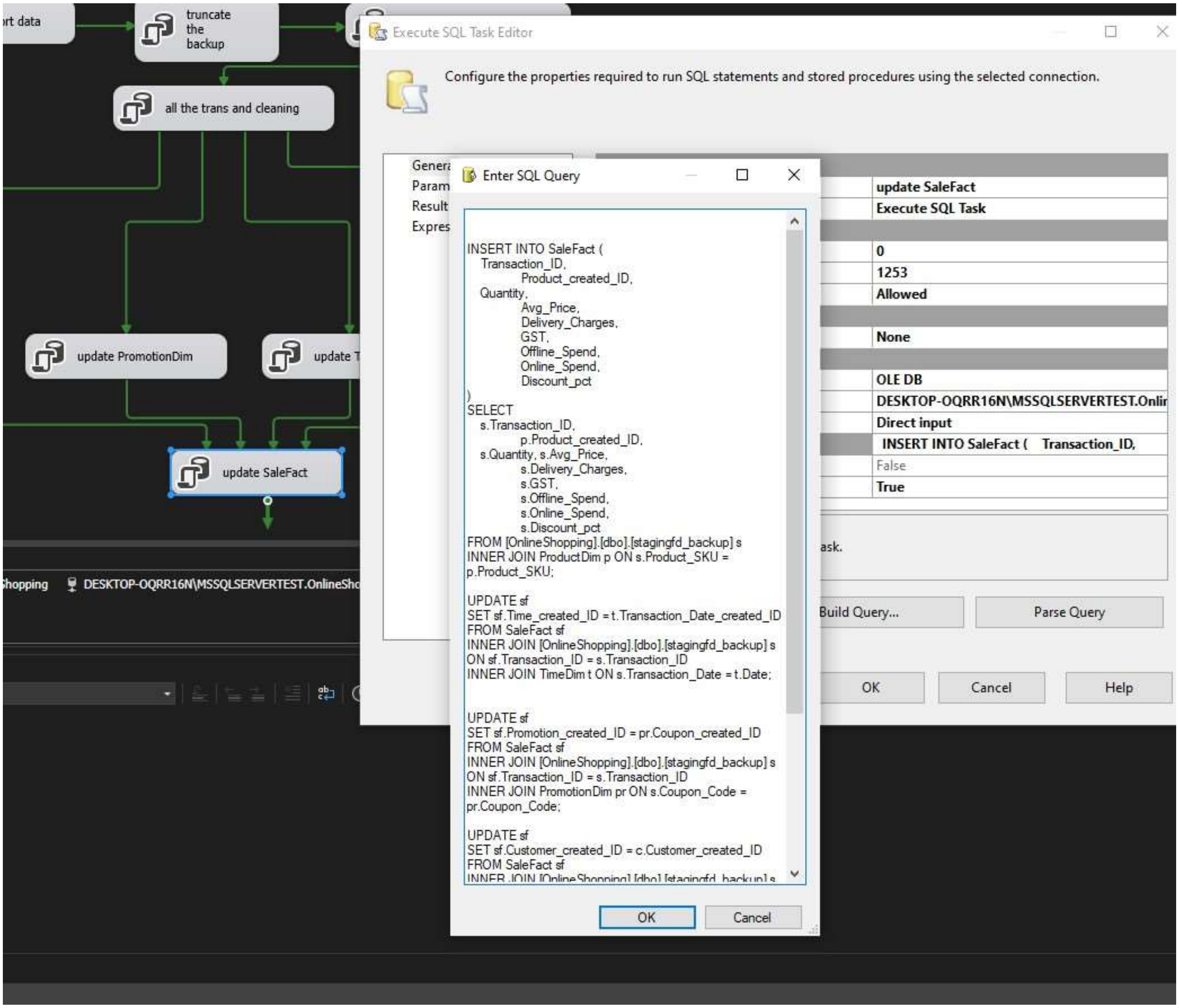
Update fact table

We want to have 1 column with derivable measures (we need them for the Power BI, because when you connect with a cube from SSIS the "modeling tab" is unable). We will update the table with this derivable measure.

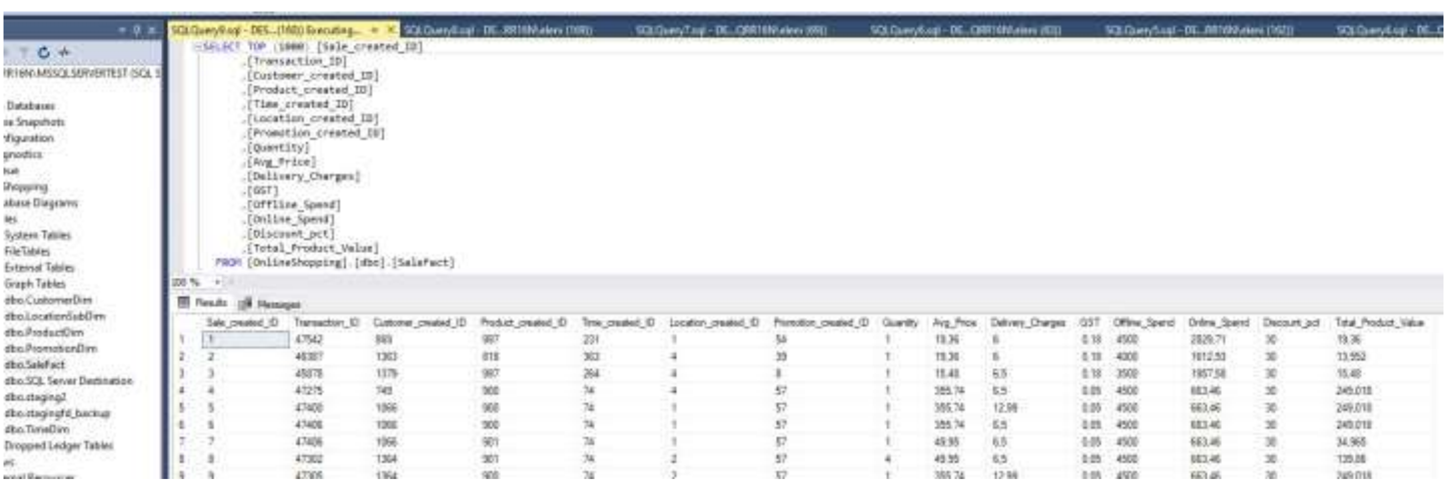
```
CREATE TABLE SaleFact (

    Sale_created_ID INT IDENTITY(1,1) PRIMARY KEY,
    Transaction_ID INT,

    Customer_created_ID INT,
    Product_created_ID INT,
    Time_created_ID INT,
    Location_created_ID INT,
    Promotion_created_ID INT,

    Quantity INT,        -- we need to make sure that it is the same type as the ones in the [stagingfd_backup]
    Avg_Price FLOAT,
    Delivery_Charges FLOAT,
    GST  FLOAT,
    Offline_Spend FLOAT,
    Online_Spend FLOAT,
    Discount_pct FLOAT,

    FOREIGN KEY (Customer_created_ID) REFERENCES CustomerDim(Customer_created_ID),
    FOREIGN KEY (Product_created_ID) REFERENCES ProductDim(Product_created_ID),
    FOREIGN KEY (Time_created_ID) REFERENCES TimeDim(Transaction_Date_created_ID),
    FOREIGN KEY (Location_created_ID) REFERENCES LocationSubDim(Location_created_ID),
    FOREIGN KEY (Promotion_created_ID) REFERENCES PromotionDim(Coupon_created_ID)
);
```

```
INSERT INTO SaleFact (
    Transaction_ID,
        Product_created_ID,
    Quantity,
        Avg_Price,
        Delivery_Charges,
        GST,
        Offline_Spend,
        Online_Spend,
        Discount_pct
)
SELECT
    s.Transaction_ID,
        p.Product_created_ID,
    s.Quantity, s.Avg_Price,
        s.Delivery_Charges,
        s.GST,
        s.Offline_Spend,
        s.Online_Spend,
        s.Discount_pct
FROM [OnlineShopping].[dbo].[stagingfd_backup] s
```

```sql
INNER JOIN ProductDim p ON s.Product_SKU = p.Product_SKU;                    --(52924 rows affected)

UPDATE sf
SET sf.Time_created_ID = t.Transaction_Date_created_ID
FROM SaleFact sf
INNER JOIN [OnlineShopping].[dbo].[stagingfd_backup] s ON sf.Transaction_ID = s.Transaction_ID
INNER JOIN TimeDim t ON s.Transaction_Date = t.Date;                          --(52924 rows affected)

UPDATE sf
SET sf.Promotion_created_ID = pr.Coupon_created_ID
FROM SaleFact sf
INNER JOIN [OnlineShopping].[dbo].[stagingfd_backup] s ON sf.Transaction_ID = s.Transaction_ID
INNER JOIN PromotionDim pr ON s.Coupon_Code = pr.Coupon_Code;                 --(52924 rows
affected)

UPDATE sf
SET sf.Customer_created_ID = c.Customer_created_ID
FROM SaleFact sf
INNER JOIN [OnlineShopping].[dbo].[stagingfd_backup] s ON sf.Transaction_ID = s.Transaction_ID
INNER JOIN CustomerDim c ON s.CustomerID = c.CustomerID;                      --(52924 rows affected)

UPDATE sf
SET sf.Location_created_ID = l.Location_created_ID
FROM SaleFact sf
INNER JOIN [OnlineShopping].[dbo].[stagingfd_backup] s ON sf.Transaction_ID = s.Transaction_ID
INNER JOIN LocationSubDim l
ON (s.Location = l.City OR s.Location = l.State OR (s.Location = 'NA' AND l.City = 'No_location_info'));     --(52924
rows affected)


ALTER TABLE [OnlineShopping].[dbo].[SaleFact]
ADD Total_Product_Value FLOAT;

UPDATE sf
SET sf.Total_Product_Value = CASE
                          WHEN p.Coupon_Status = 'used' THEN (sf.Quantity * sf.Avg_Price) - (sf.Quantity * sf.Avg_Price *
sf.Discount_pct / 100)
                          ELSE sf.Quantity * sf.Avg_Price
                        END
FROM [OnlineShopping].[dbo].[SaleFact] sf
INNER JOIN [OnlineShopping].[dbo].[PromotionDim] p ON sf.Promotion_created_ID = p.Coupon_created_ID;
```



In the SSMS looks like this (we display only the top rows):



Matching: Use business keys to match records in the staging table to records in the dimension tables.

Inserting into Fact Table: Insert surrogate keys (like Customer_created_ID) from dimension tables into the fact table, along with the measures from the staging table.

By using this approach, we leverage the detailed and descriptive nature of the natural keys for accurate matching and the efficiency and consistency of surrogate keys for storage and querying in the data warehouse.

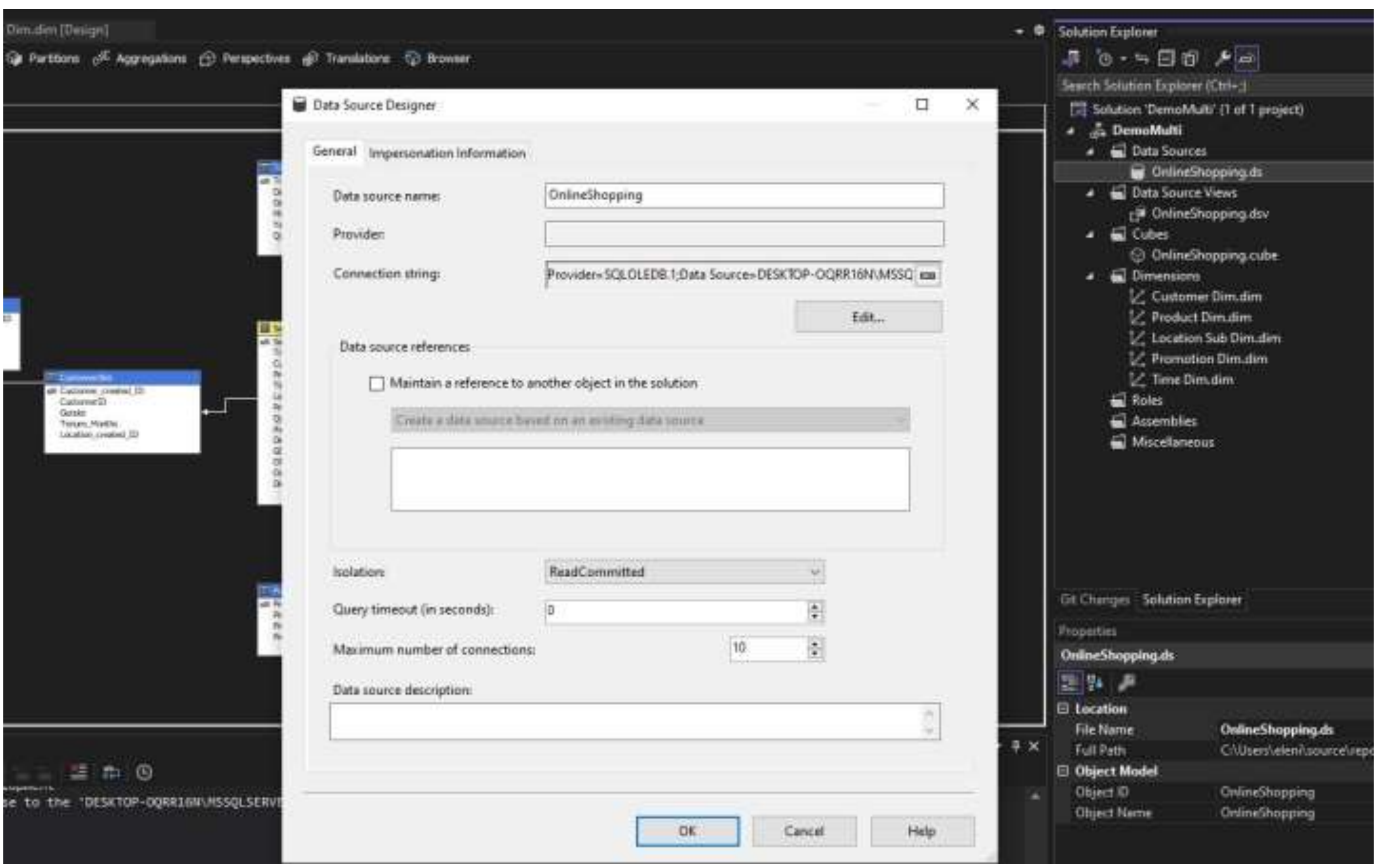
5. Design the Cube

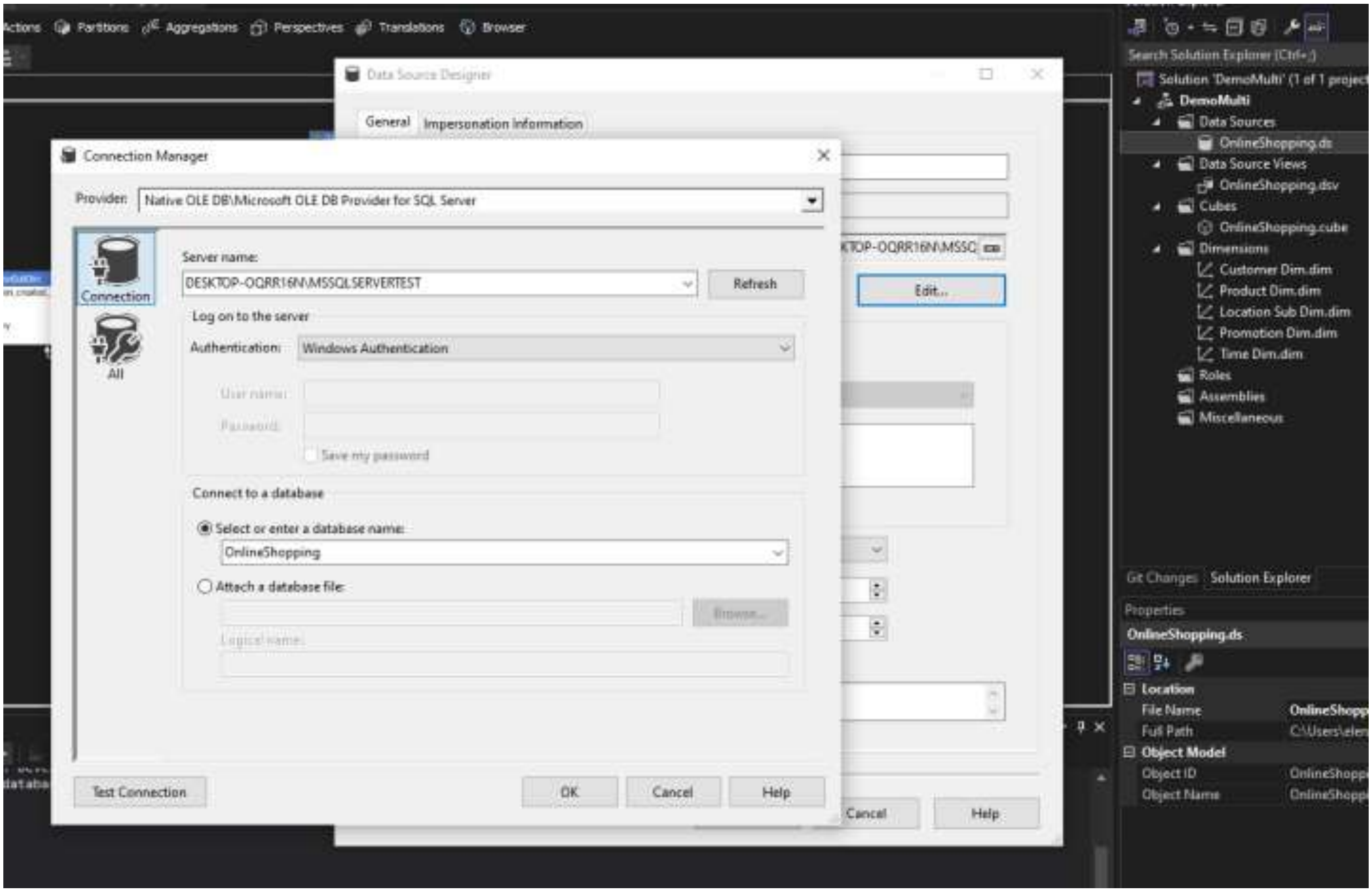
OLAP servers (in Microsoft, this is SSAS) are tools through which we define a cube and then utilize this cube through various reporting tools, such as Power BI, Tableau, and directly from Excel, among others.

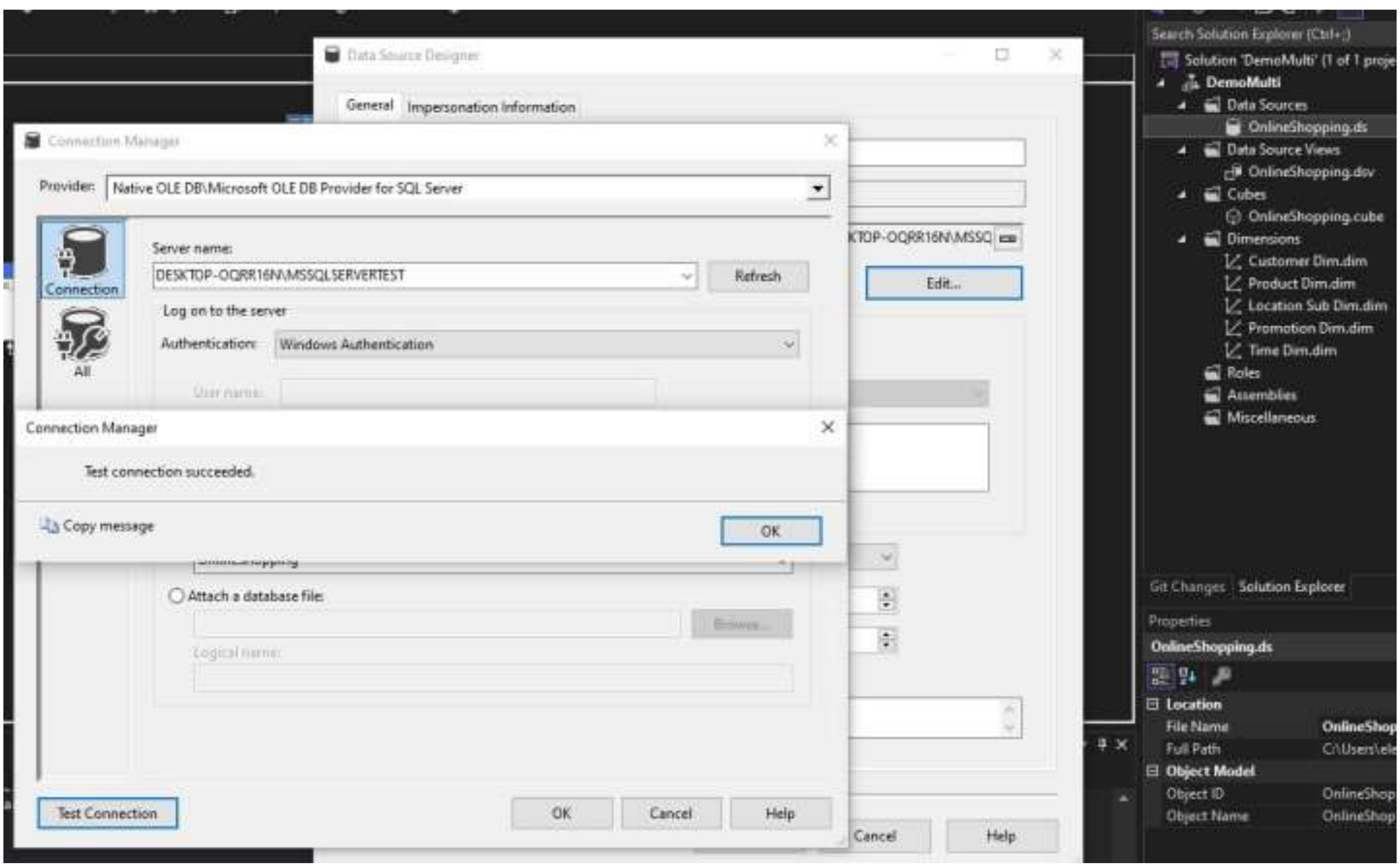
We open Data Tools (in Visual Studio) and, instead of starting an SSIS, we initiate an SSAS Analysis Services Multidimensional Project. On the right-hand panel, you essentially see the first four steps that need to be undertaken:

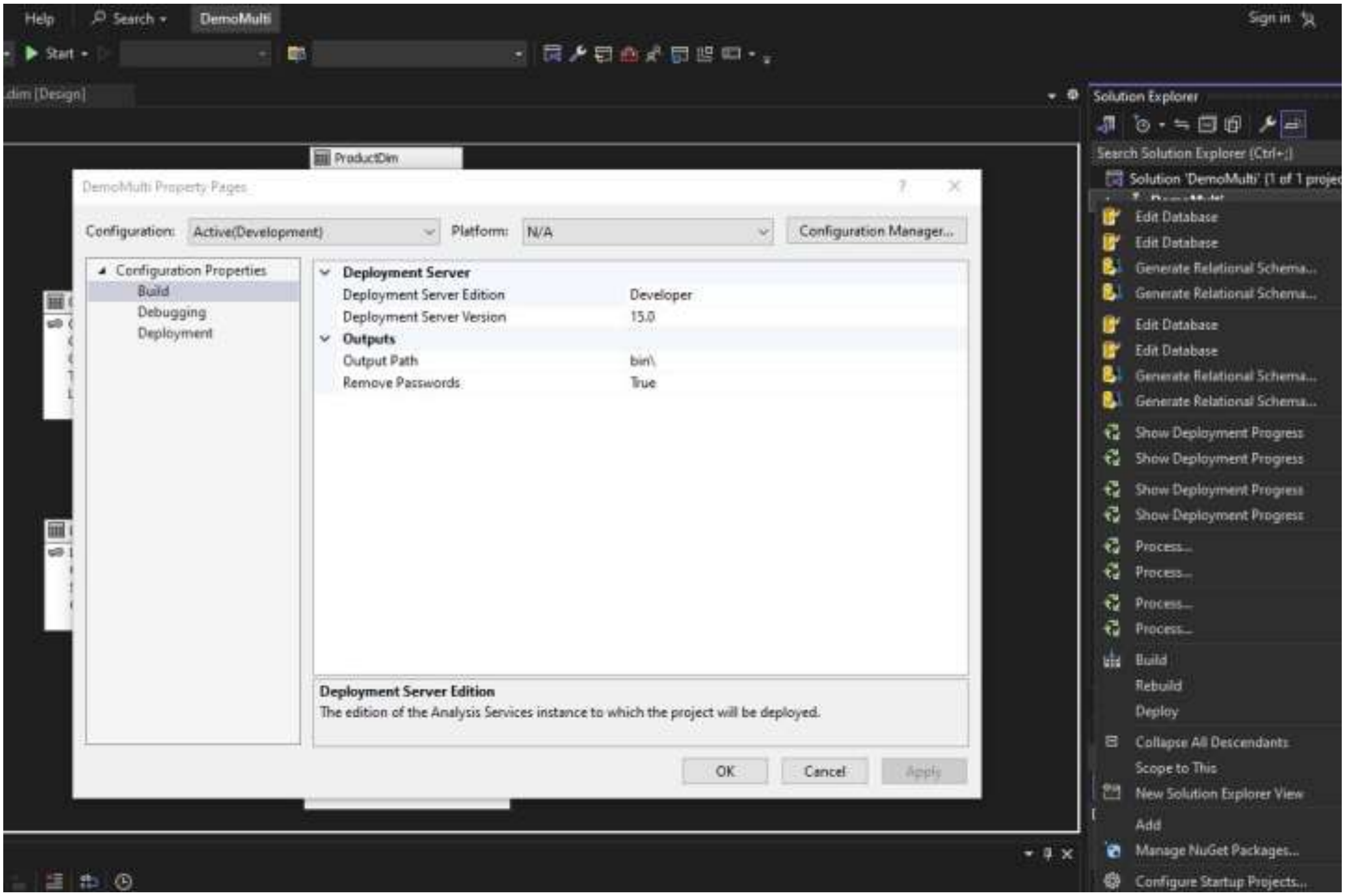
- Define the data source or sources to be used (for us, it's the SQL Server running locally). These are the sources where we will find the data we can use to set up our cubes.
- Define the data source views. After having already defined a data source in step one, it specifically asks which parts of this data source you will need to proceed. Select the specific tables needed to set up the cube.
- In the cube and dimensions section, we need to define which (or what) is the fact table we want to use along with the dimensions.

First we did the connection in connection manager:

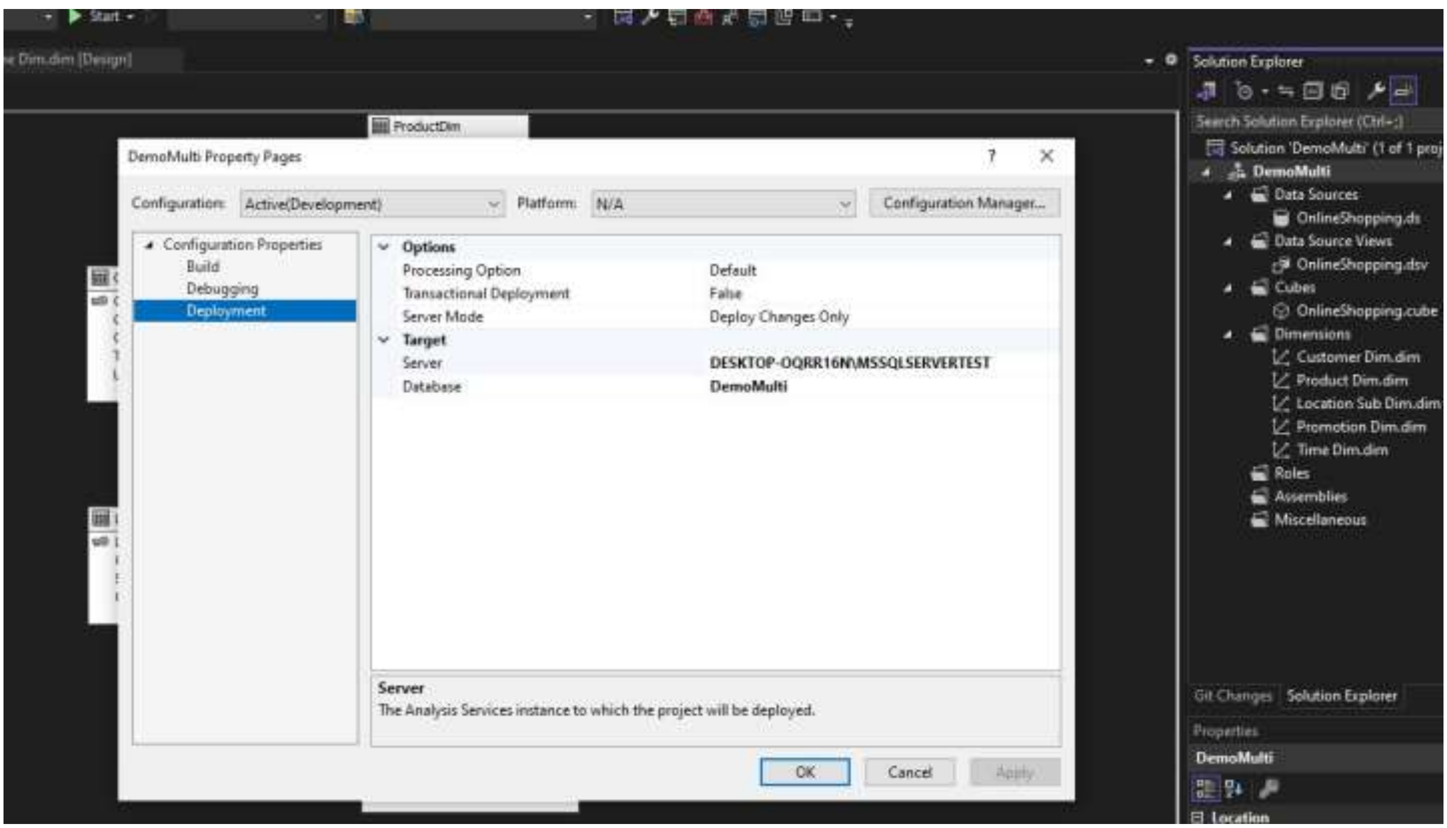



Data Source Designer
General
Impersonation Information
Connection Manager
Provider: Native OLE DB\Microsoft OLE DB Provider for SQL Server
Connection
All
Server name:
DESKTOP-OQRR16N\MSSQLSERVERTEST
Refresh
Log on to the server
Authentication: Windows Authentication
User name:
Password:
Save my password
Connect to a database
Select or enter a database name:
OnlineShopping
Attach a database file:
Browse...
Logical name:
Test Connection
OK
Cancel
Help
Edit...
Solution 'DemoMulti' (1 of 1 project)
DemoMulti
Data Sources
OnlineShopping.ds
Data Source Views
OnlineShopping.dsv
Cubes
OnlineShopping.cube
Dimensions
Customer Dim.dim
Product Dim.dim
Location Sub Dim.dim
Promotion Dim.dim
Time Dim.dim
Roles
Assemblies
Miscellaneous
Git Changes
Solution Explorer
Properties
OnlineShopping.ds
Location
File Name
Full Path
Object Model
Object ID
Object Name

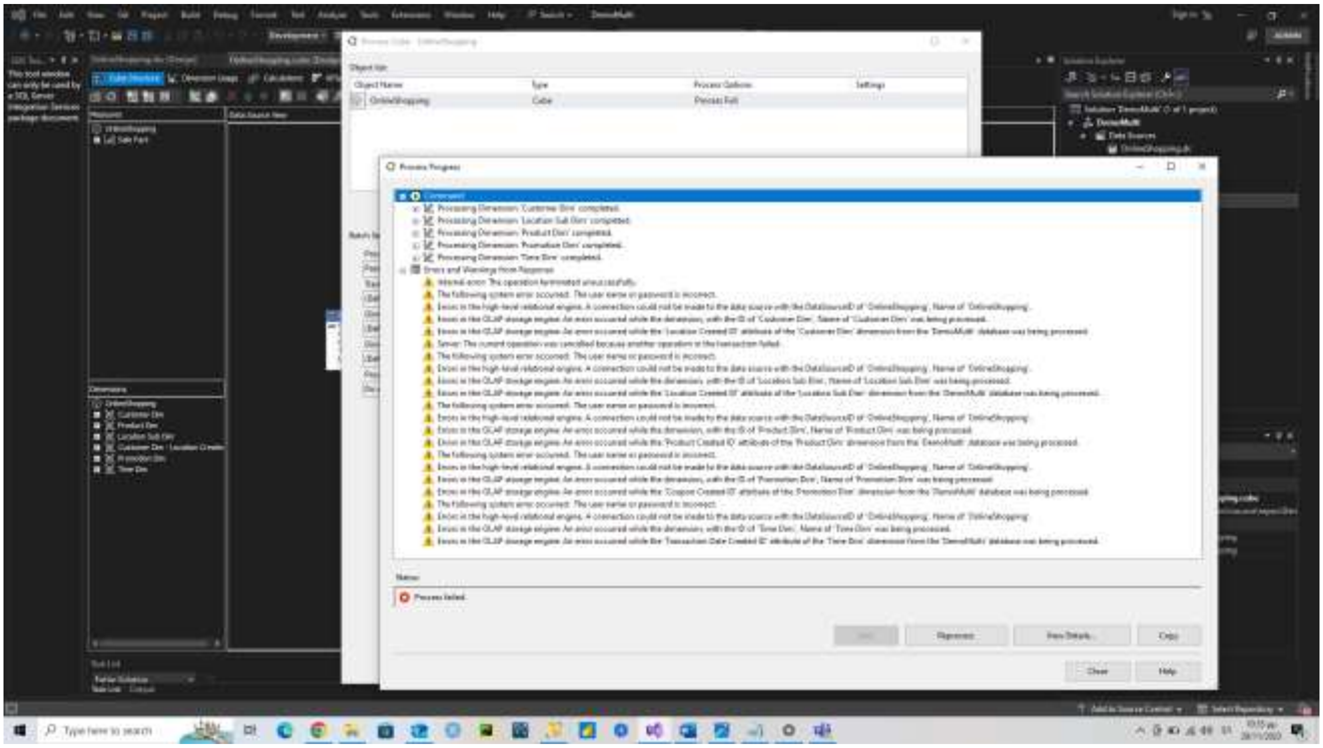

Data Source Designer
General
Impersonation Information
Connection Manager
Provider: Native OLE DB\Microsoft OLE DB Provider for SQL Server
Connection
All
Server name:
DESKTOP-OQRR16N\MSSQLSERVERTEST
Refresh
Log on to the server
Authentication: Windows Authentication
Connection Manager
Test connection succeeded.
Copy message
OK
Attach a database file:
Browse...
Logical name:
Test Connection
OK
Cancel
Help
Edit...
Solution 'DemoMulti' (1 of 1 project)
DemoMulti
Data Sources
OnlineShopping.ds
Data Source Views
OnlineShopping.dsv
Cubes
OnlineShopping.cube
Dimensions
Customer Dim.dim
Product Dim.dim
Location Sub Dim.dim
Promotion Dim.dim
Time Dim.dim
Roles
Assemblies
Miscellaneous
Git Changes
Solution Explorer
Properties
OnlineShopping.ds
Location
File Name
Full Path
Object Model
Object ID
Object Name



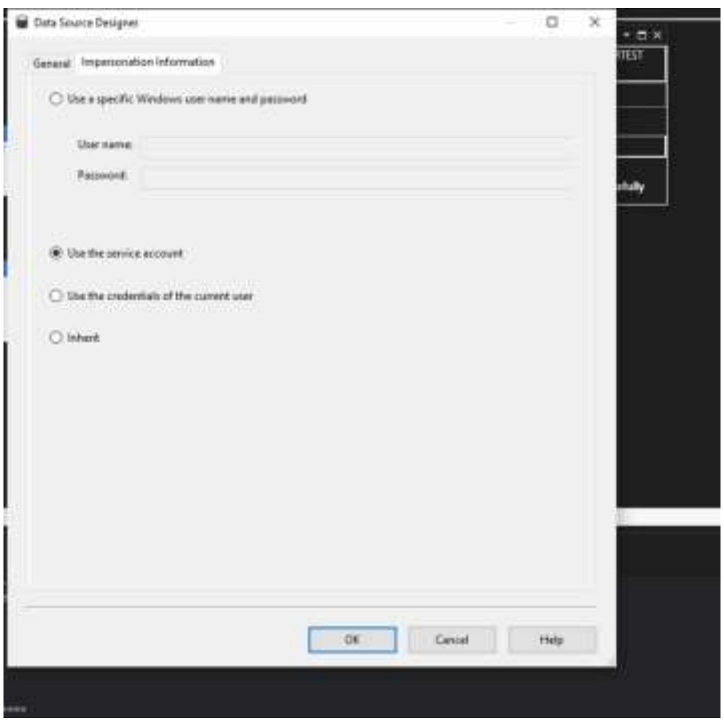
In order to deploy the project with suceess we change the server from local host to our Server :



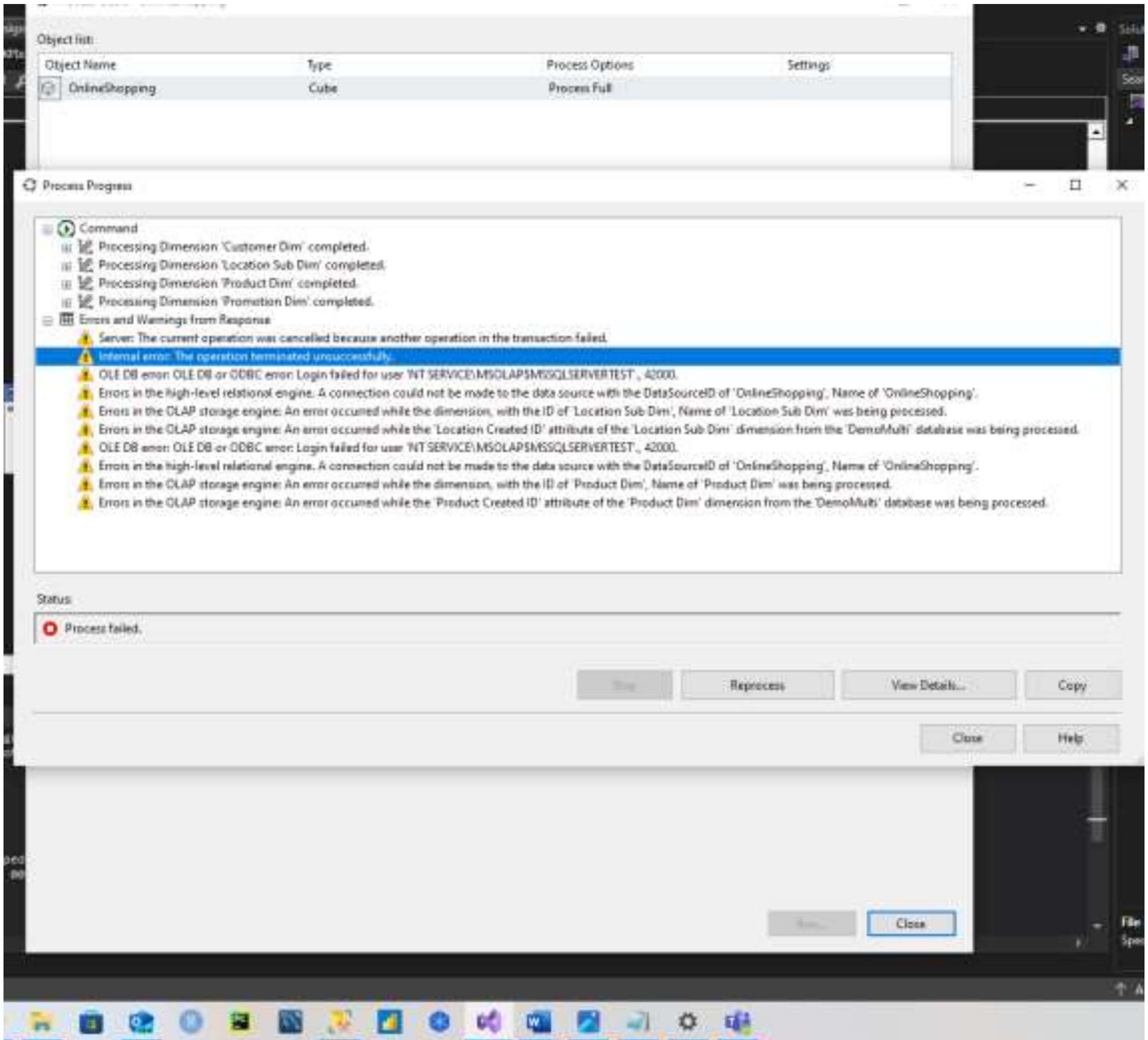
However we have errors again :



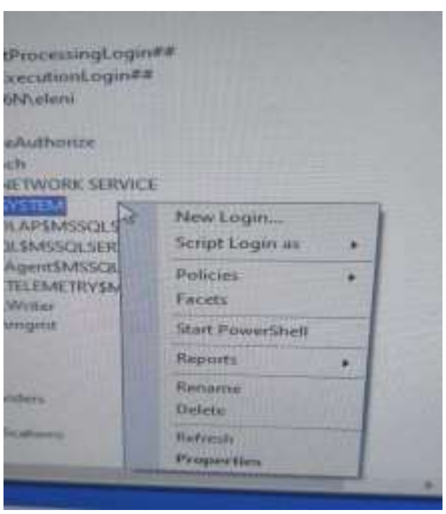
So, first we changed our choice in order to use the service account for the connection :

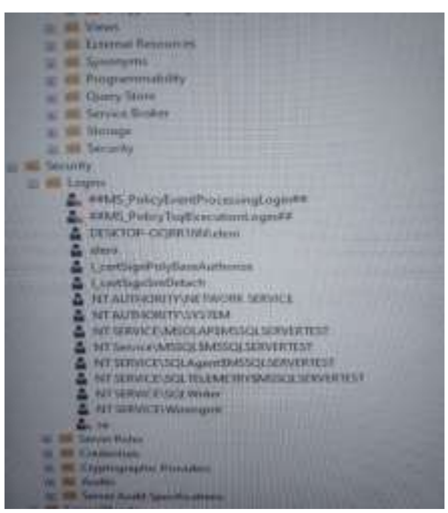


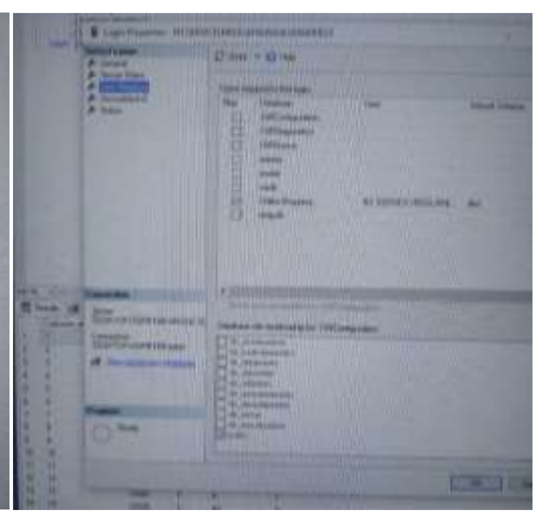
However we have errors again:



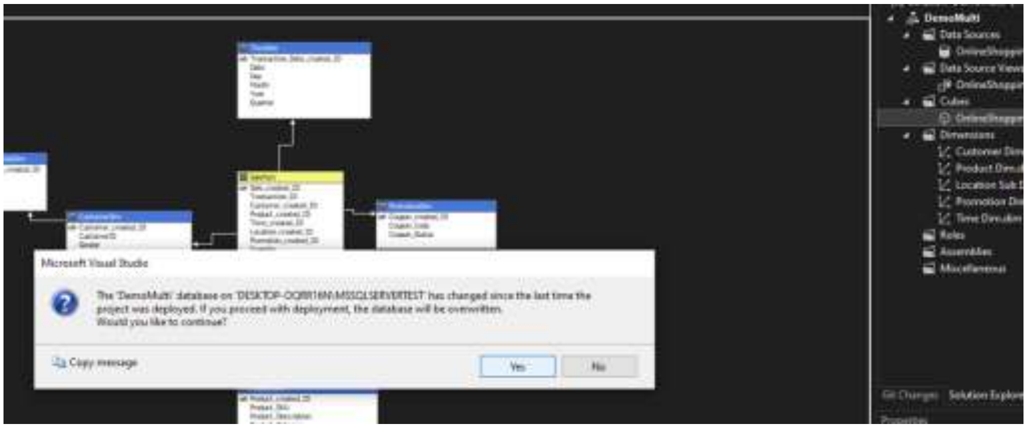
So we went and copy the log fail and we went and create in the SSM a new login and we went to user mapping in this new loggin and choose our database and public:

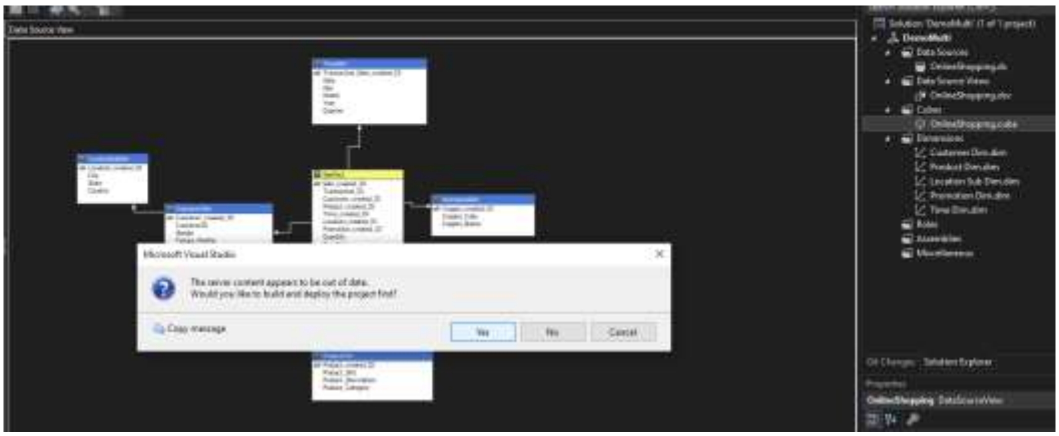


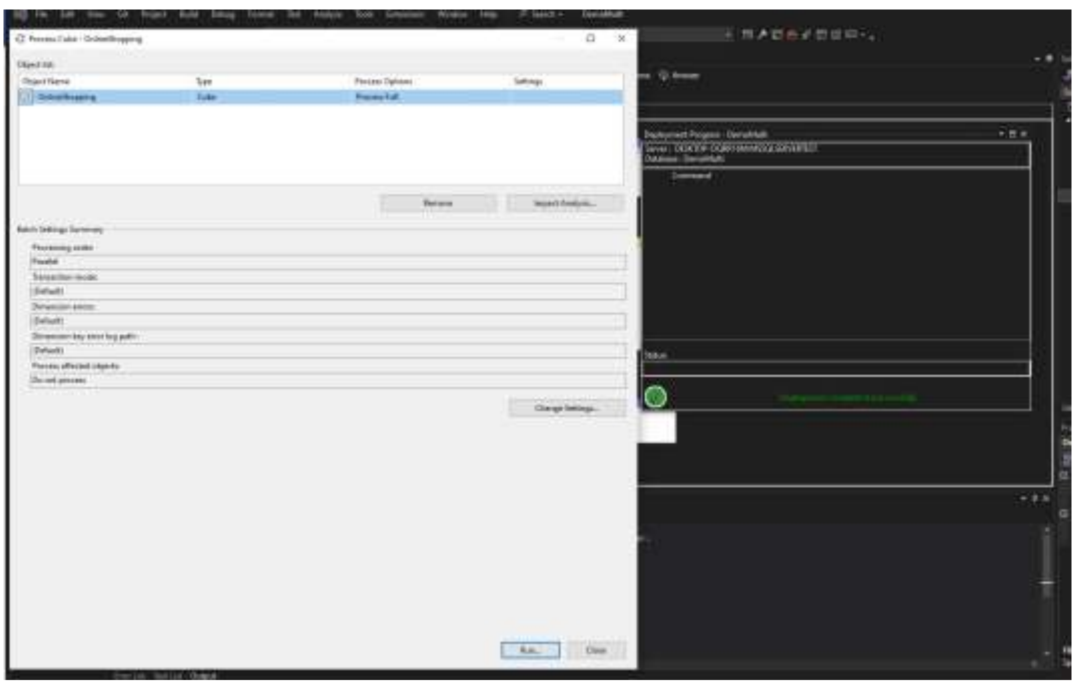


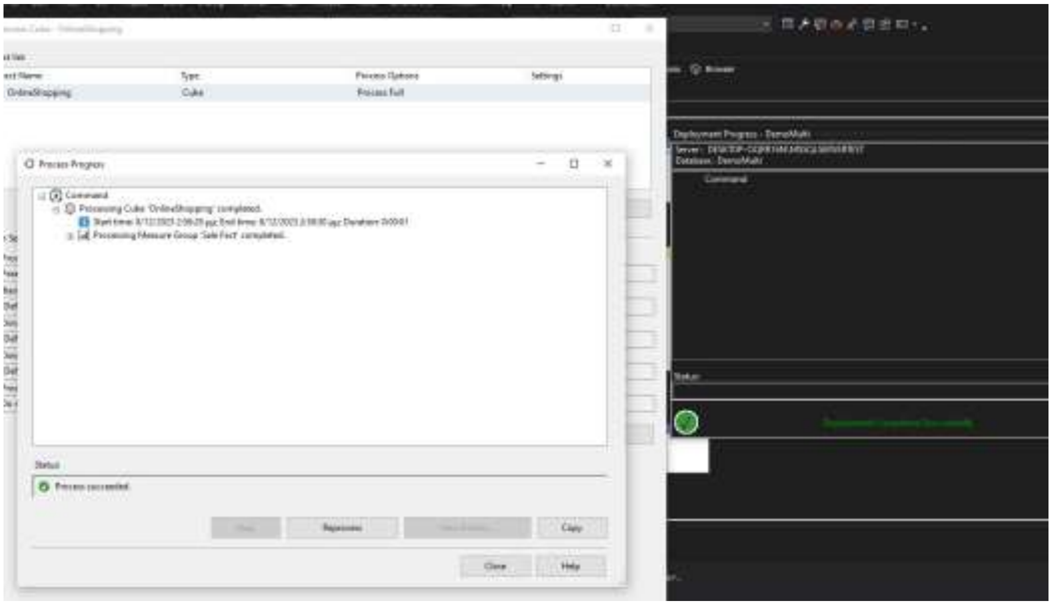


After this everything was ok and the process succeeded:


Microsoft Visual Studio
The 'DemoMulti' database on 'DESKTOP-OQRR16N\MSSQLSERVERTEST' has changed since the last time the project was deployed. If you proceed with deployment, the database will be overwritten.
Would you like to continue?
Copy message
Yes
No
DemoMulti
Data Sources
Data Source Views
Cubes
Dimensions
Roles
Assemblies
Miscellaneous

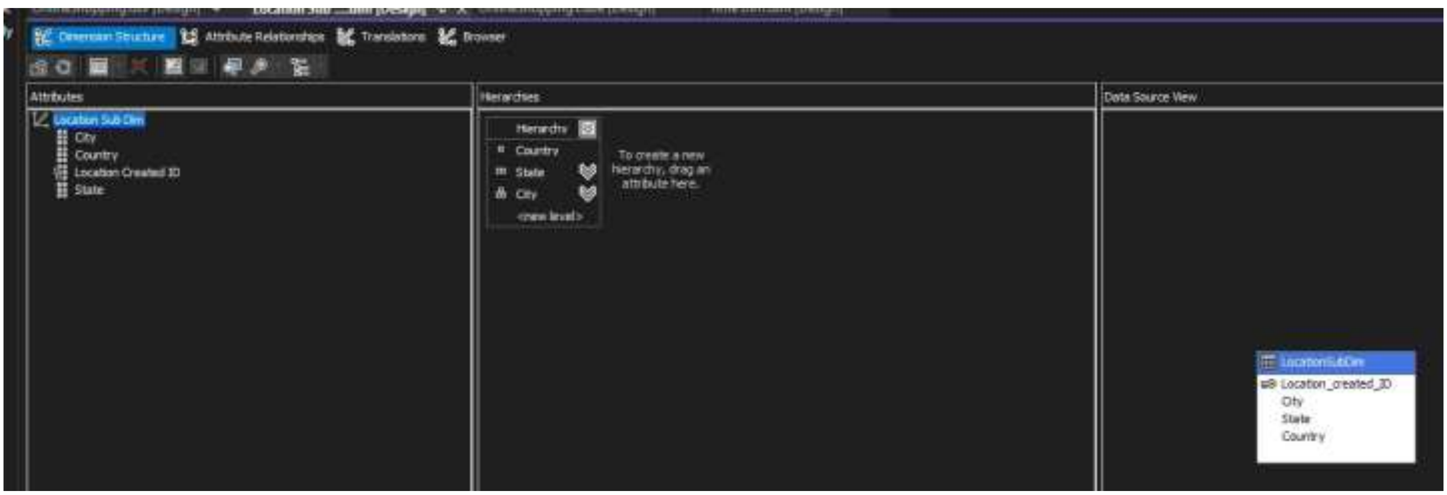

Microsoft Visual Studio
The server content appears to be out of date.
Would you like to build and deploy the project first?
Copy message
Yes
No
Cancel
DemoMulti
Data Sources
Data Source Views
Cubes
Dimensions
Roles
Assemblies
Miscellaneous

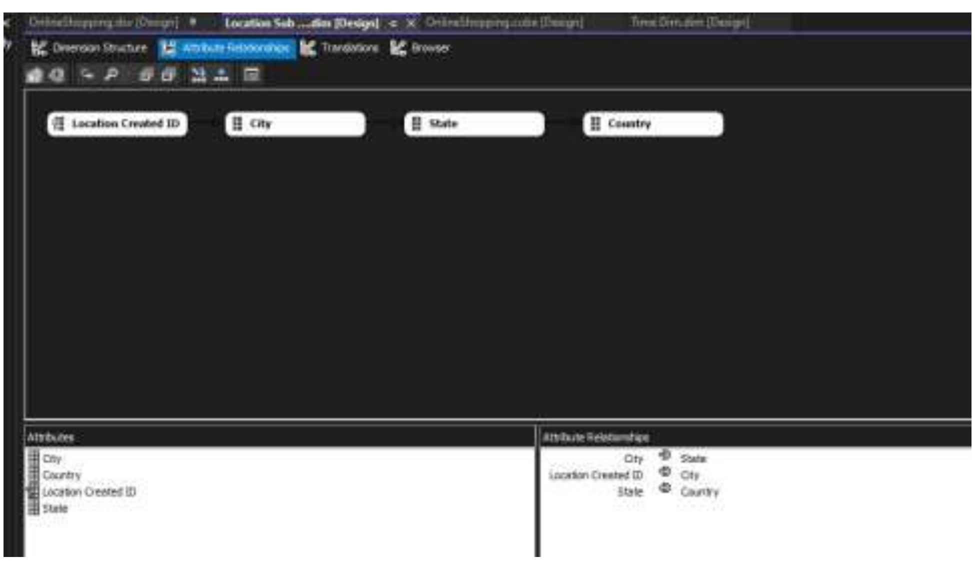

Process Cube - OnlineShopping
Object list:
Object Name
Type
Process Options
Settings
OnlineShopping
Cube
Process Full
Remove
Impact Analysis...
Batch Settings Summary
Change Settings...
Run...
Close
Deployment Progress - DemoMulti
Status



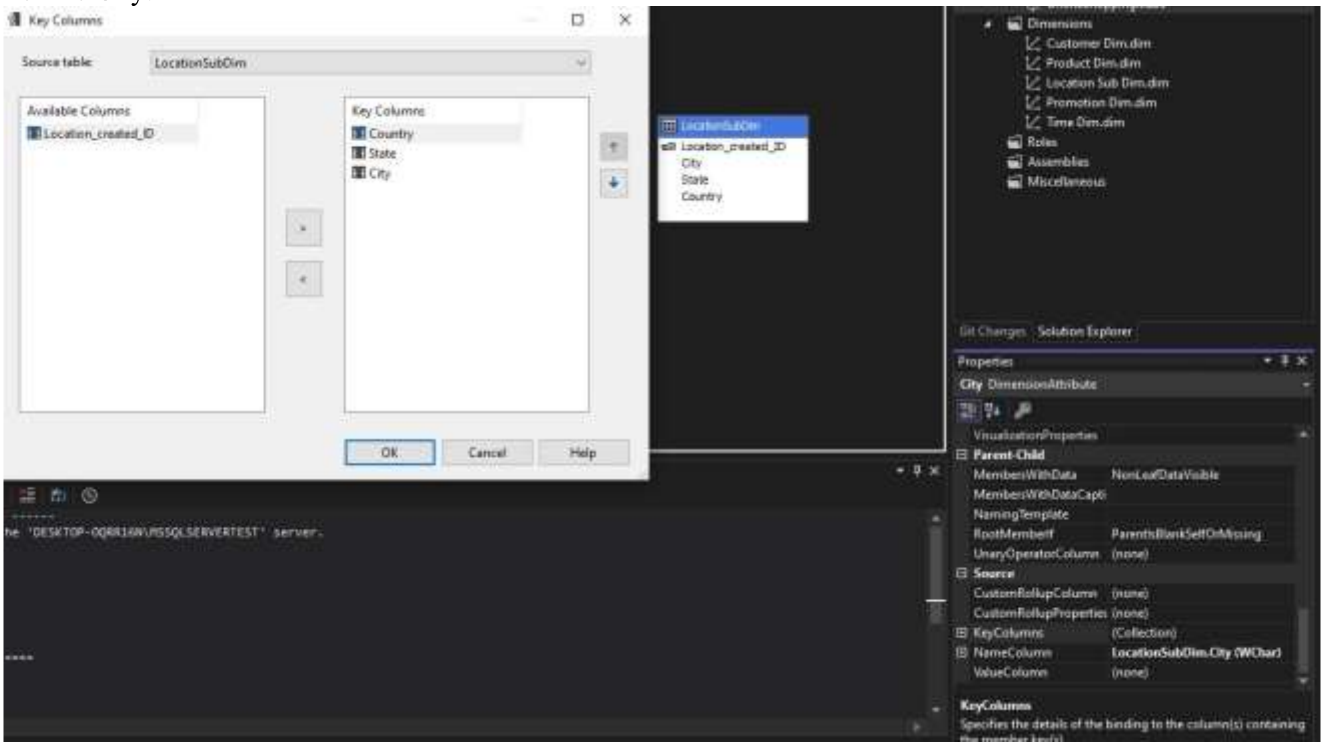
Hierarchies in Locacation Sub-dimension

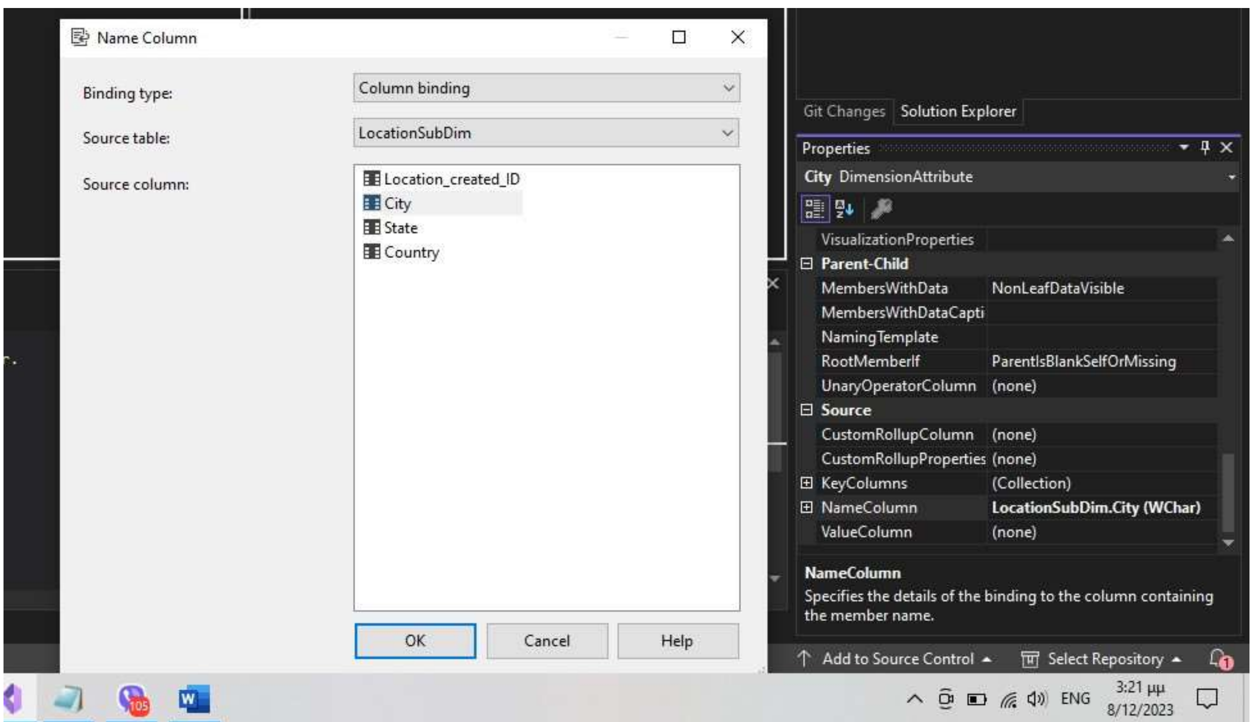
In order to create the Hierarchies in Locacation Sub-dimension we did the followings in the SSIS (natural Hierarchies):



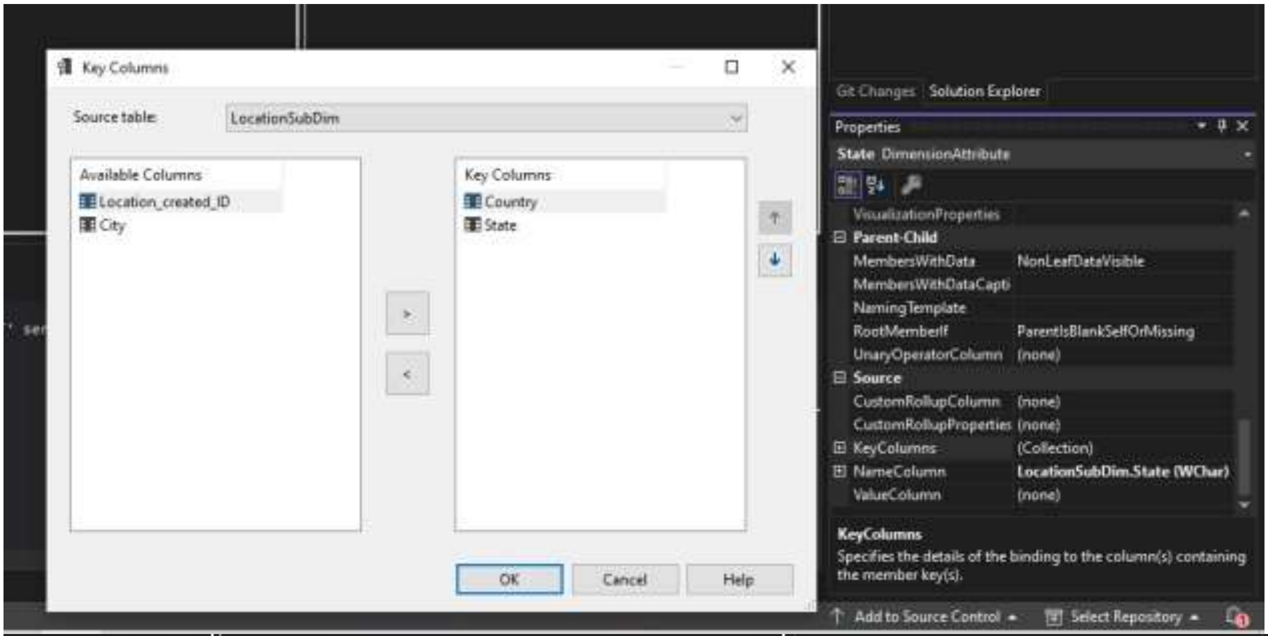


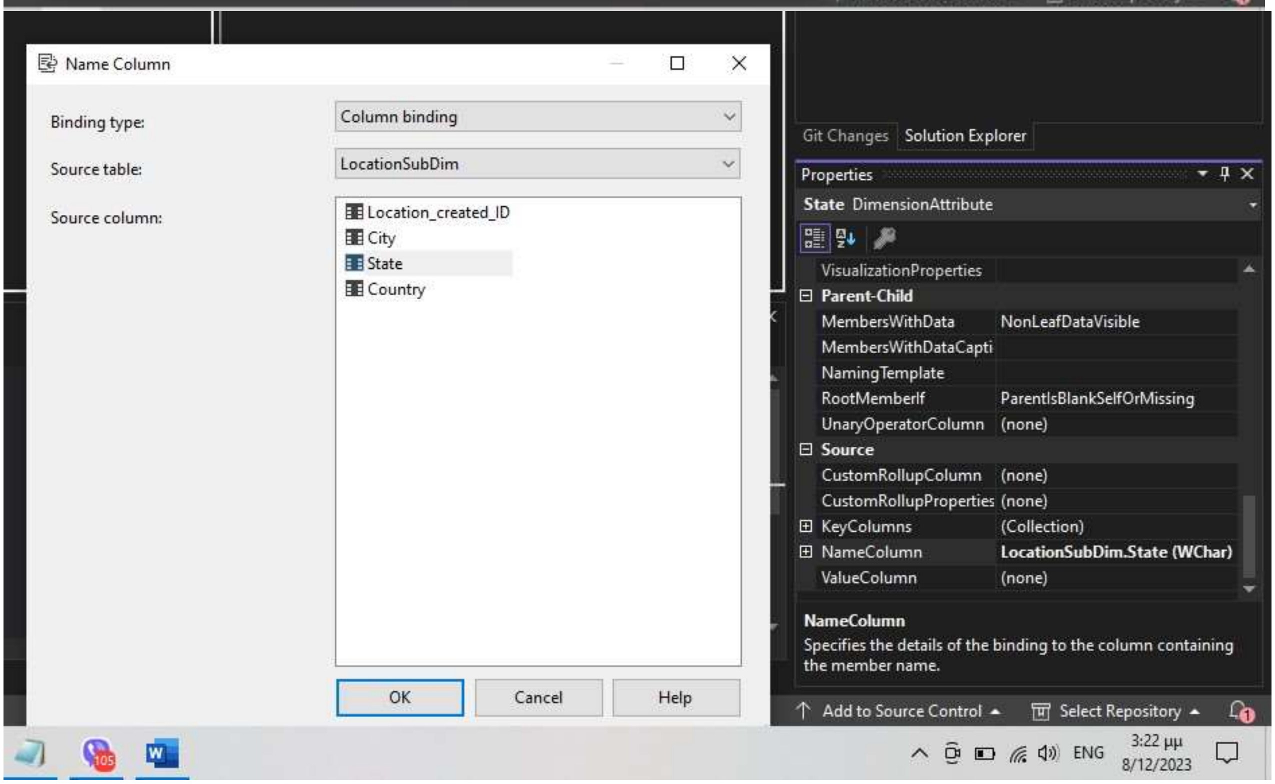
For the City:





And for the State:


Key Columns
Source table:
LocationSubDim
Available Columns
Location_created_ID
City
Key Columns
Country
State
OK
Cancel
Help
Git Changes
Solution Explorer
Properties
State DimensionAttribute
VisualizationProperties
Parent-Child
MembersWithData
NonLeafDataVisible
MembersWithDataCapti
NamingTemplate
RootMemberIf
ParentIsBlankSelfOrMissing
UnaryOperatorColumn
(none)
Source
CustomRollupColumn
(none)
CustomRollupProperties
(none)
KeyColumns
(Collection)
NameColumn
LocationSubDim.State (WChar)
ValueColumn
(none)
KeyColumns
Specifies the details of the binding to the column(s) containing the member key(s).
Add to Source Control
Select Repository

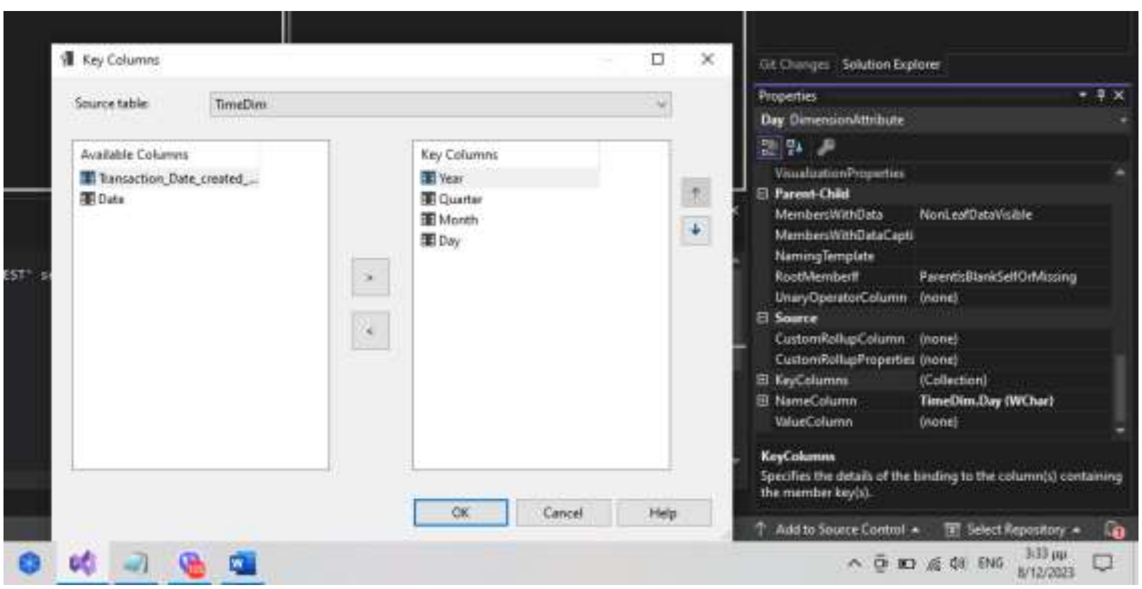

Name Column
Binding type:
Column binding
Source table:
LocationSubDim
Source column:
Location_created_ID
City
State
Country
OK
Cancel
Help
Git Changes
Solution Explorer
Properties
State DimensionAttribute
VisualizationProperties
Parent-Child
MembersWithData
NonLeafDataVisible
MembersWithDataCapti
NamingTemplate
RootMemberIf
ParentIsBlankSelfOrMissing
UnaryOperatorColumn
(none)
Source
CustomRollupColumn
(none)
CustomRollupProperties
(none)
KeyColumns
(Collection)
NameColumn
LocationSubDim.State (WChar)
ValueColumn
(none)
NameColumn
Specifies the details of the binding to the column containing the member name.
Add to Source Control
Select Repository
ENG
3:22 μμ
8/12/2023

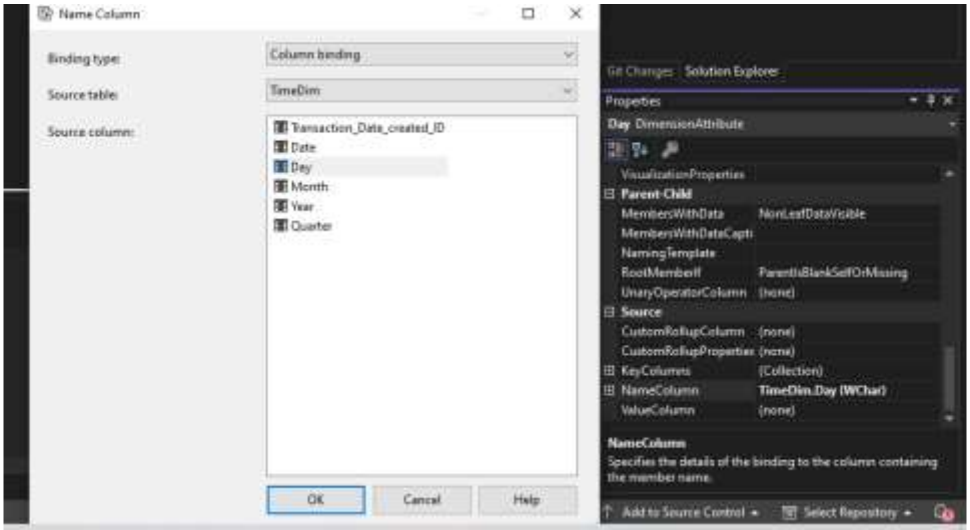
Hierarchies in Time dimension

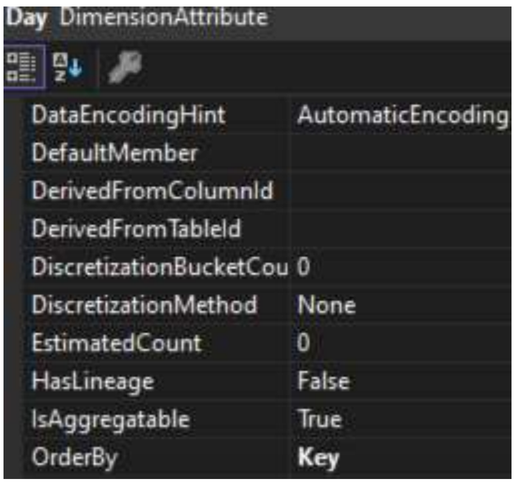
In order to create the Hierarchies in Locacation Sub-dimension we did the followings in the SSIS (natural Hierarchies):

After this we went and did the natural hierarchy:

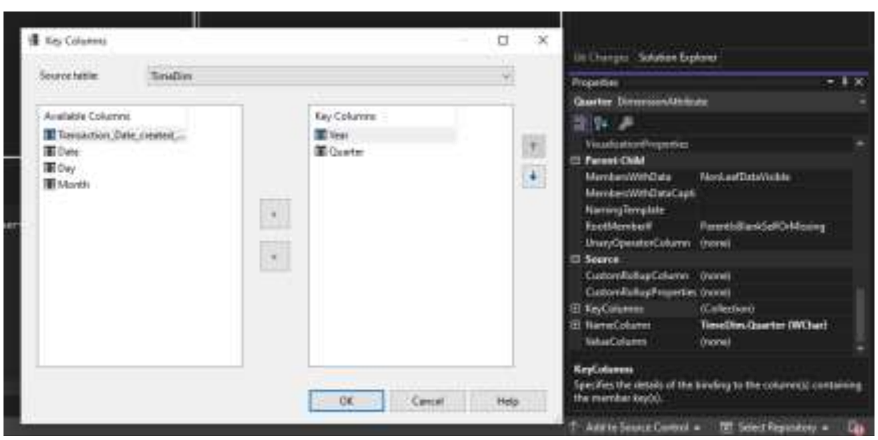
And in the Properties to Fix the Key columns, the name column and the Order:

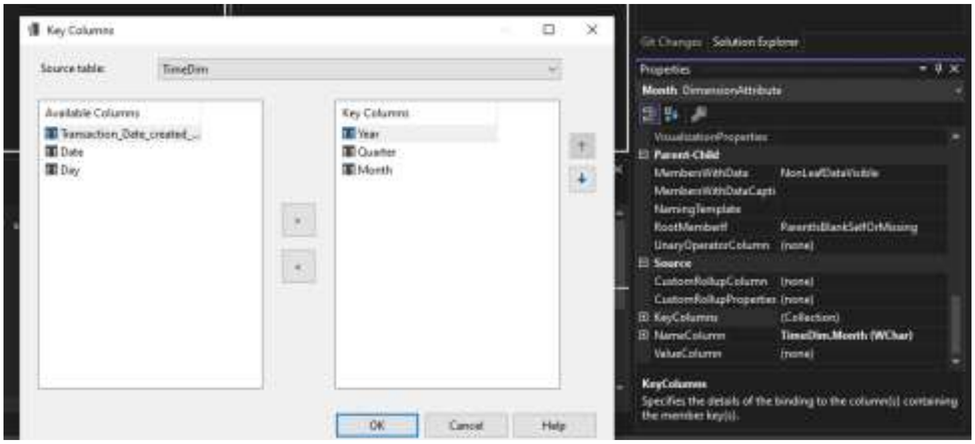






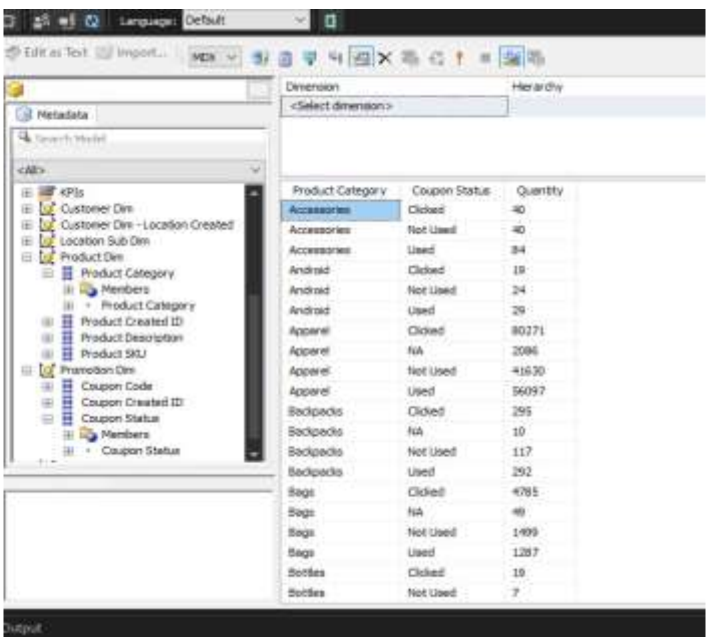
We did this steps for the month and the quarter as well:



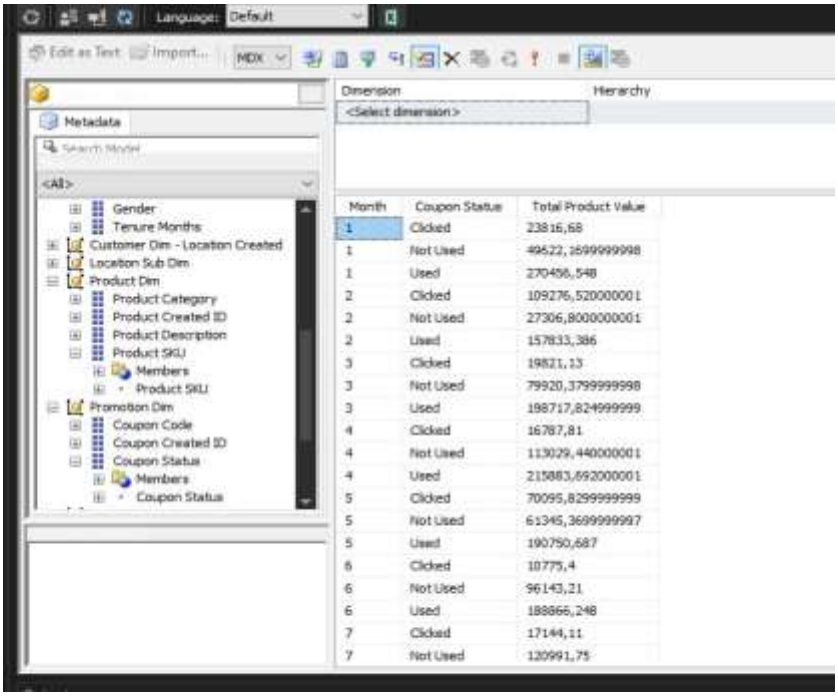


<u>Example of olap queries in SSIS</u>

The query displays the product category alongside the coupon status and quantity for each item, this query is appropriate to evaluate the effectiveness of coupon campaigns on product sales over time:

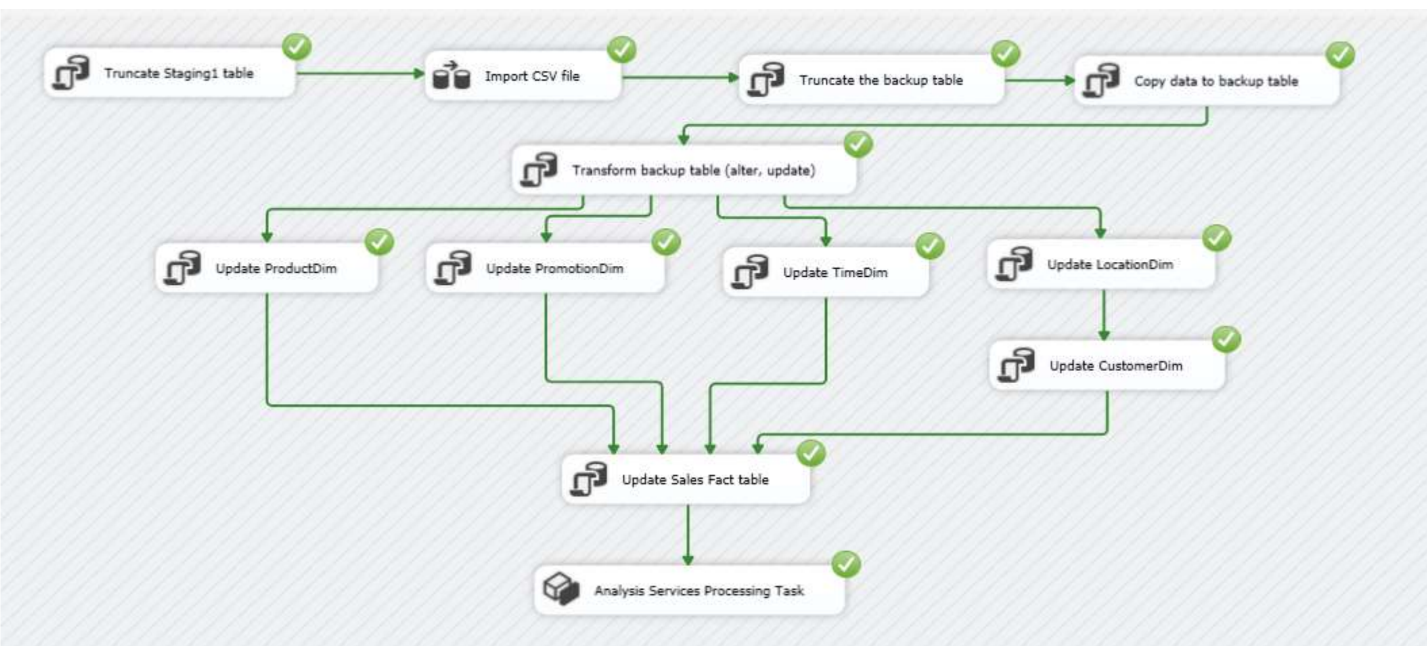


The following query breakdown the total product value by month and coupon status. It allows for an analysis of how different coupon statuses (Clicked, Used, Not Used) correlate with the total value of products sold in each month. This can provide insights into consumer behavior regarding the use of coupons and how it impacts sales:



6. Execute the entire package in SSIS

After adding the analysis services processing task to our work flow in SSIS, we now have the complete package for creating our data warehouse project.



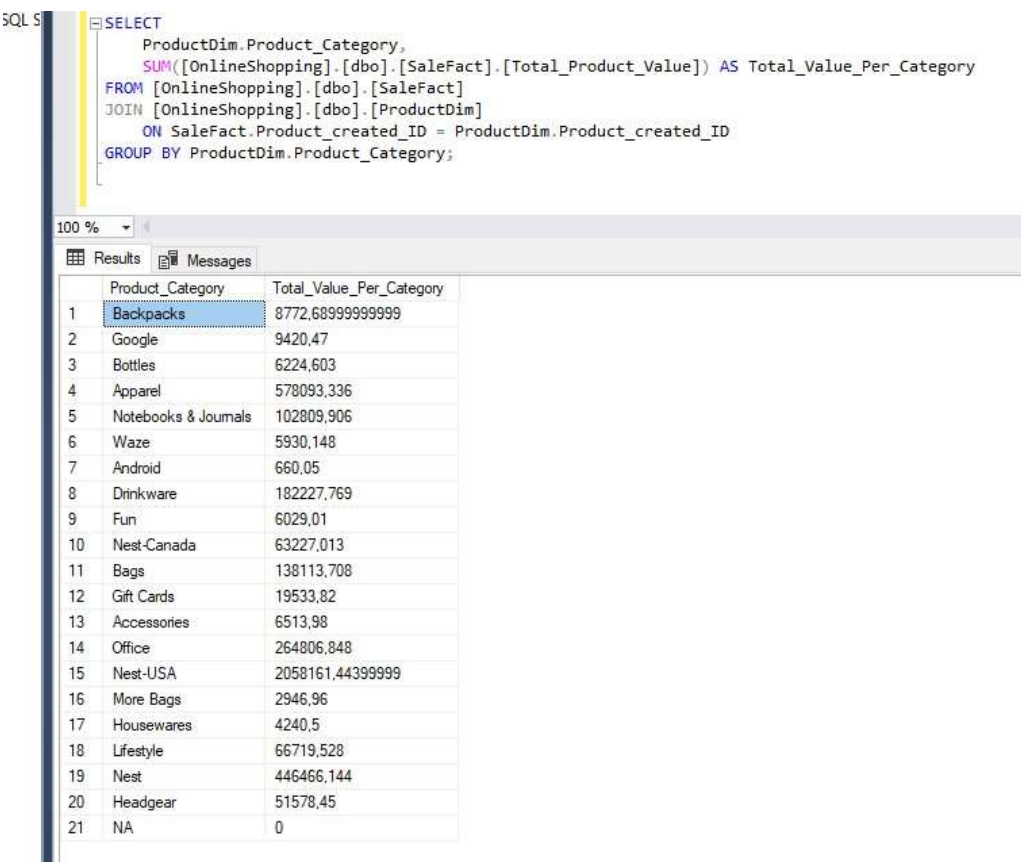
7. OLAP reports

Roll-up (Συγκεντρωτική Ανάλυση)
Drill-down (Λεπτομερής Ανάλυση)
Drill-out (Επέκταση Ανάλυσης)
Slicing (Διατομή)
Dicing (Κοπή σε Κύβους)

i. Sales Performance by Product Category and Gender:

The query summarizes the total sales value for each product category. The result of this query provides an insight into how well sales are performing in each category. For instance, if a category has a significantly higher total value compared to others, this might indicate that the products in this category are more popular.

```
SELECT ProductDim.Product_Category,
      SUM([OnlineShopping].[dbo].[SaleFact].[Total_Product_Value]) AS Total_Value_Per_Category
FROM [OnlineShopping].[dbo].[SaleFact]
JOIN  [OnlineShopping].[dbo].[ProductDim] ON SaleFact.Product_created_ID = ProductDim.Product_created_ID
GROUP BY ProductDim.Product_Category;
```



```
SELECT
    ProductDim.Product_Category,
    SUM([OnlineShopping].[dbo].[SaleFact].[Total_Product_Value]) AS Total_Value_Per_Category
FROM [OnlineShopping].[dbo].[SaleFact]
JOIN [OnlineShopping].[dbo].[ProductDim]
    ON SaleFact.Product_created_ID = ProductDim.Product_created_ID
GROUP BY ProductDim.Product_Category;
```

100 %

Results | Messages

	Product_Category	Total_Value_Per_Category
1	Backpacks	8772,68999999999
2	Google	9420,47
3	Bottles	6224,603
4	Apparel	578093,336
5	Notebooks & Journals	102809,906
6	Waze	5930,148
7	Android	660,05
8	Drinkware	182227,769
9	Fun	6029,01
10	Nest-Canada	63227,013
11	Bags	138113,708
12	Gift Cards	19533,82
13	Accessories	6513,98
14	Office	264806,848
15	Nest-USA	2058161,44399999
16	More Bags	2946,96
17	Housewares	4240,5
18	Lifestyle	66719,528
19	Nest	446466,144
20	Headgear	51578,45
21	NA	0

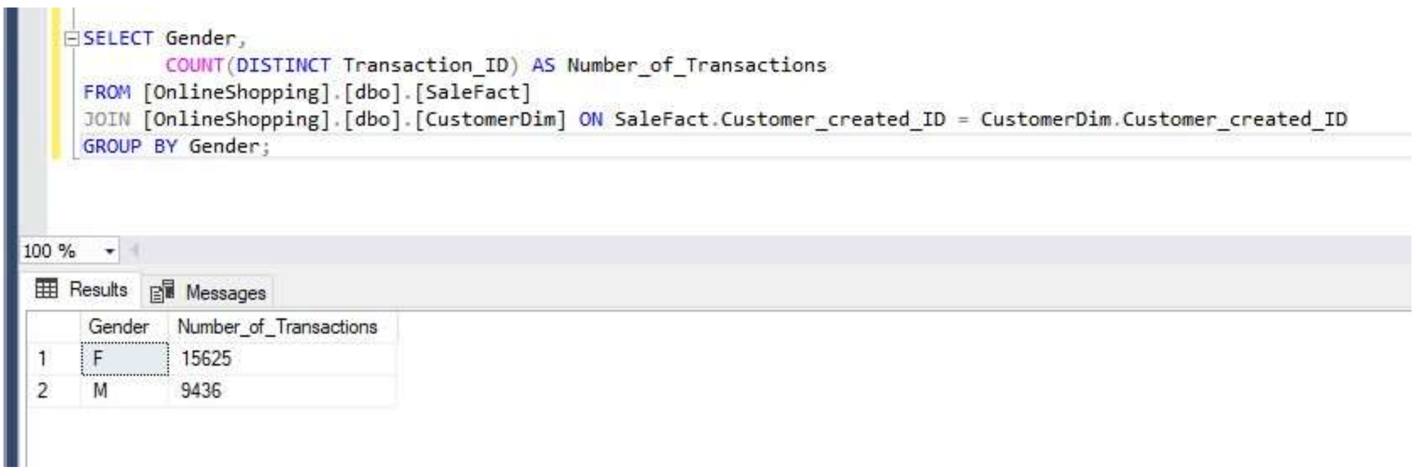
ii. Customer Purchasing Patterns by Gender, Location, and Combined Gender/Location:

Analyzing transactions by gender and location (city, state, or country) provides insights for targeted marketing and demographic analysis.

For Gender:

Analyzing transactions by gender is very useful in businesses. Gender-based segmentation is a common practice in marketing. It allows businesses to categorize customers into groups and tailor their strategies to each segment. For example, certain promotions or discounts may be more appealing to one gender over another.

```
SELECT Gender,
    COUNT(DISTINCT Transaction_ID) AS Number_of_Transactions
FROM [OnlineShopping].[dbo].[SaleFact]
JOIN [OnlineShopping].[dbo].[CustomerDim ]  ON SaleFact.Customer_created_ID =
CustomerDim.Customer_created_ID
GROUP BY Gender;
```



For Location:

Analyzing the number of transactions made by customers depending on the city is important for several reasons. Cities often have diverse demographics and cultural characteristics. Analyzing transactions on a city level enables businesses to create localized marketing strategies that resonate with the unique attributes of each location.

```
SELECT City, -- Replace with State or Country if needed
    COUNT(DISTINCT Transaction_ID) AS Number_of_Transactions
FROM [OnlineShopping].[dbo].[SaleFact]
JOIN [OnlineShopping].[dbo].[CustomerDim ] ON SaleFact.Customer_created_ID =
CustomerDim.Customer_created_ID
JOIN [OnlineShopping].[dbo].[ LocationSubDim] ON CustomerDim.Location_created_ID =
LocationSubDim.Location_created_ID
GROUP BY City; -- Replace with State or Country
```

```
SELECT City, -- Replace with State or Country if needed
       COUNT(DISTINCT Transaction_ID) AS Number_of_Transactions
FROM [OnlineShopping].[dbo].[SaleFact]
JOIN [OnlineShopping].[dbo].[CustomerDim] ON SaleFact.Customer_created_ID = CustomerDim.Customer_created_ID
JOIN [OnlineShopping].[dbo].[LocationSubDim] ON CustomerDim.Location_created_ID = LocationSubDim.Location_created_ID
GROUP BY City; -- Replace with State or Country
```

100 %

Results | Messages

	City	Number_of_Transactions
1	Chicago	8762
2	Unknown City	14971
3	Washington DC	1328

Combined Gender and Location:

Businesses should recognize that gender is just one aspect of a customer's identity, and other factors, such as age, interests, and location, may also play significant roles in shaping purchasing behavior. For example, businesses can optimize their inventory by stocking products that are popular among specific gender groups in particular locations. This prevents overstocking or understocking of items and enhances overall inventory management efficiency. Moreover, using gender and location data for analysis can provide a competitive advantage. Businesses that understand and respond to the diverse preferences of customers in different regions and gender groups are better positioned to capture market share.

```
SELECT
   Gender,
   City, -- or State or Country, depending on which level of location detail I need
   COUNT(DISTINCT Transaction_ID) AS Number_of_Transactions
FROM [OnlineShopping].[dbo].[SaleFact]
JOIN [OnlineShopping].[dbo].[CustomerDim] ON SaleFact.Customer_created_ID = CustomerDim.Customer_created_ID
JOIN [OnlineShopping].[dbo].[LocationSubDim] ON CustomerDim.Location_created_ID =
LocationSubDim.Location_created_ID
GROUP BY Gender, City; -- Replace City with State or Country
```

```
SELECT
    Gender,
    City, -- or State or Country, depending on which level of location detail I need
    COUNT(DISTINCT Transaction_ID) AS Number_of_Transactions
FROM [OnlineShopping].[dbo].[SaleFact]
JOIN [OnlineShopping].[dbo].[CustomerDim] ON SaleFact.Customer_created_ID = CustomerDim.Customer_created_ID
JOIN [OnlineShopping].[dbo].[LocationSubDim] ON CustomerDim.Location_created_ID = LocationSubDim.Location_created_ID
GROUP BY Gender, City; -- Replace City with State or Country
```

100 %

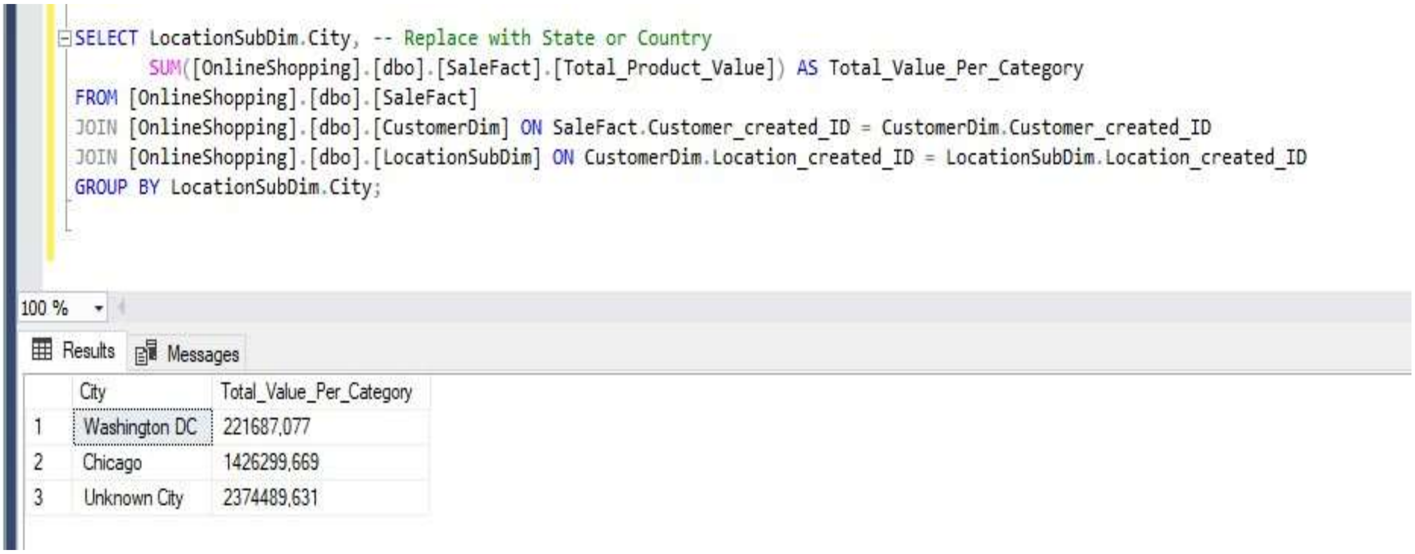
Results | Messages

	Gender	City	Number_of_Transactions
1	F	Chicago	5483
2	M	Chicago	3279
3	F	Unknown City	9522
4	M	Unknown City	5449
5	F	Washington DC	620
6	M	Washington DC	708

iii. Total Sales per Location:

Location analysis is crucial for businesses considering expansion. It provides insights into the potential success of opening new locations in specific areas based on historical transaction data and customer behavior. In addition, analyzing customer locations, businesses can strategically position warehouses in areas with high customer density. This minimizes shipping distances, reduces delivery times, and lowers shipping costs.

```
SELECT LocationSubDim.City, -- Replace with State or Country if needed
    SUM([OnlineShopping].[dbo].[SaleFact].[Total_Product_Value]) AS Total_Value_Per_Category
FROM [OnlineShopping].[dbo].[SaleFact]
JOIN [OnlineShopping].[dbo].[CustomerDim] ON SaleFact.Customer_created_ID = CustomerDim.Customer_created_ID
JOIN [OnlineShopping].[dbo].[LocationSubDim] ON CustomerDim.Location_created_ID =
LocationSubDim.Location_created_ID
GROUP BY LocationSubDim.City;
```



iv. Monthly Sales Trends:

Understanding seasonal sales patterns and inventory planning.

Seasonal variations can significantly impact sales. Monthly analysis helps businesses recognize and understand these trends, allowing for better preparation, inventory management, and marketing strategies tailored to specific seasons or months. Businesses can assess the effectiveness of marketing campaigns by correlating monthly sales with promotional activities. This helps in determining the return on investment (ROI) for marketing efforts and refining future campaigns based on what resonates with customers.

```
SELECT Year,
    Month,
    SUM([OnlineShopping].[dbo].[SaleFact].[Total_Product_Value]) AS Monthly_Sales
FROM [OnlineShopping].[dbo].[SaleFact]
JOIN [OnlineShopping].[dbo].[TimeDim] ON SaleFact.Time_created_ID = TimeDim.Transaction_Date_created_ID
GROUP BY Year, Month
ORDER BY Year, Month;
```

```sql
SELECT Year,
       Month,
       SUM([OnlineShopping].[dbo].[SaleFact].[Total_Product_Value]) AS Monthly_Sales
FROM [OnlineShopping].[dbo].[SaleFact]
JOIN [OnlineShopping].[dbo].[TimeDim] ON SaleFact.Time_created_ID = TimeDim.Transaction_Date_created_ID
GROUP BY Year, Month
ORDER BY Year, Month;
```

100 %

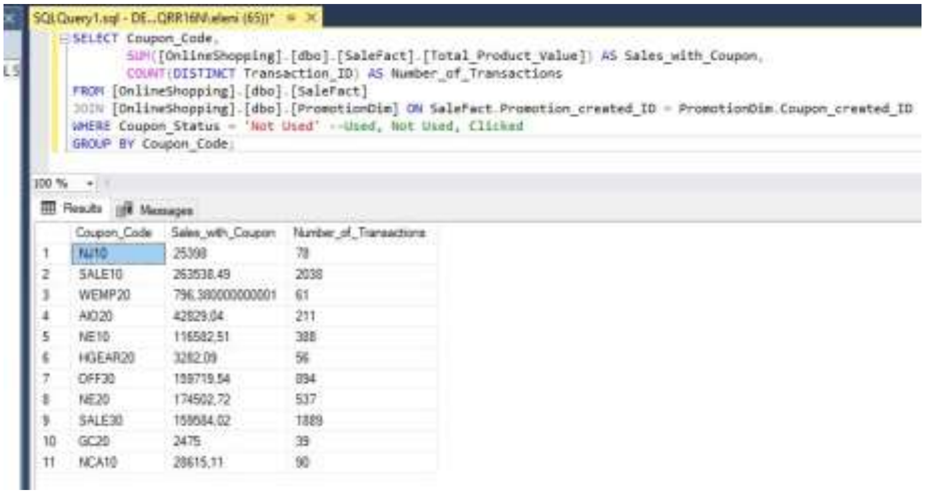
Results | Messages

	Year	Month	Monthly_Sales
1	2019	1	343895,398
2	2019	2	294416,706
3	2019	3	298459,335
4	2019	4	345700,942
5	2019	5	322191,887
6	2019	6	295784,858
7	2019	7	370963,654
8	2019	8	319966,022
9	2019	9	301939,453
10	2019	10	345369,984
11	2019	11	438166,293
12	2019	12	345621,845

v. Effectiveness of Promotional Campaigns:

Businesses should analyze the effectiveness of different promotional offers or discounts. This insight helps in refining future promotions by identifying which types of offers resonate most with the target audience and drive desired actions. Furthermore, campaign analysis provides insights into customer behavior during and after the promotional period. This includes understanding whether promotions drive new customer acquisition, repeat purchases, or changes in buying patterns.

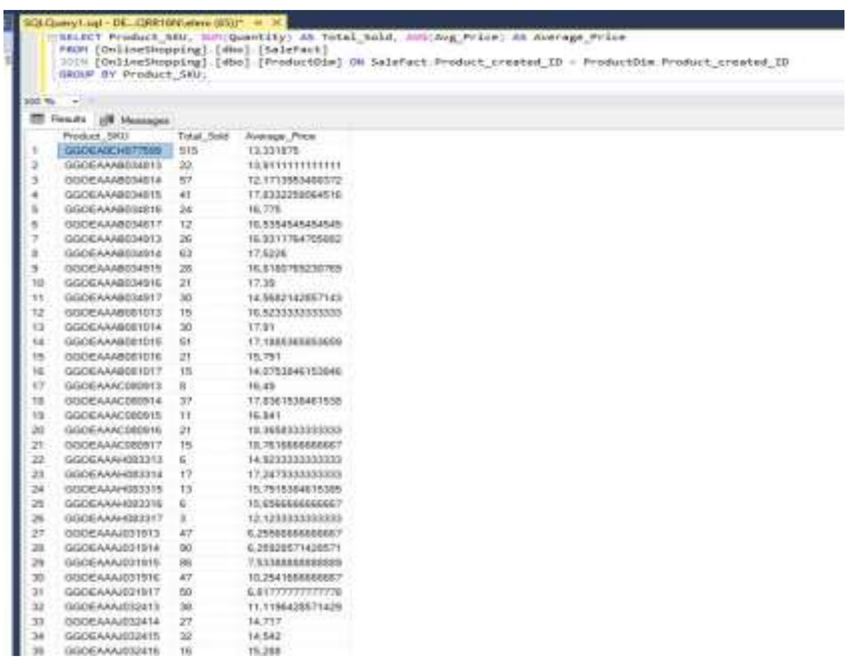
```sql
SELECT Coupon_Code,
    SUM([OnlineShopping].[dbo].[SaleFact].[Total_Product_Value]) AS Sales_with_Coupon,
    COUNT(DISTINCT Transaction_ID) AS Number_of_Transactions
FROM [OnlineShopping].[dbo].[SaleFact]
JOIN [OnlineShopping].[dbo].[ PromotionDim] ON SaleFact.Promotion_created_ID =
PromotionDim.Coupon_created_ID
WHERE Coupon_Status = 'Not Used' --Used, Not Used, Clicked
GROUP BY Coupon_Code;
```



vi. Inventory Management Insights:

Identifying the most saleable products helps in optimizing inventory levels. Businesses can focus on stocking and replenishing items that have higher demand, reducing the risk of overstocking slow-moving products. Understanding which products are more saleable provides valuable insights for marketing strategies. Businesses can tailor promotional efforts, advertising campaigns, and discounts to highlight popular products, attracting more customers and driving sales.

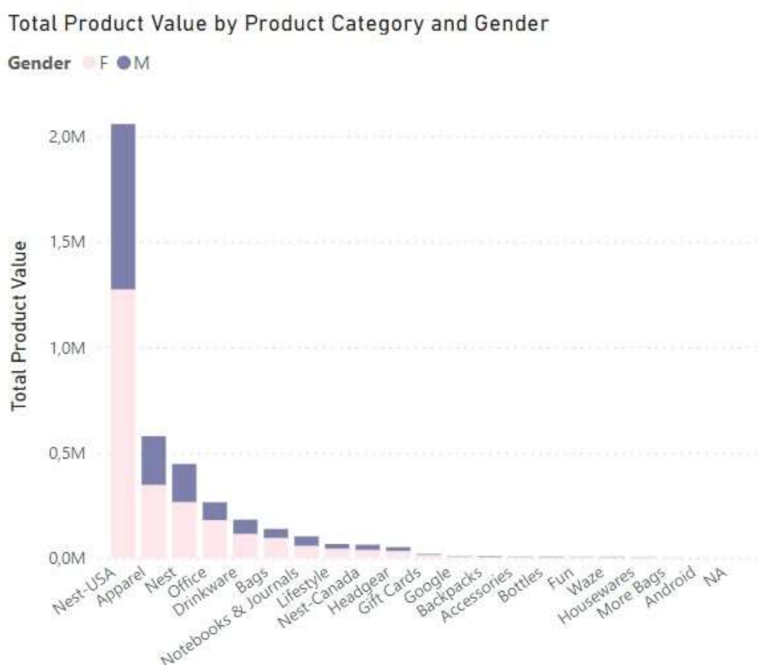
SELECT Product_SKU, SUM(Quantity) AS Total_Sold, AVG(Avg_Price) AS Average_Price
FROM [OnlineShopping].[dbo].[SaleFact]
JOIN [OnlineShopping].[dbo].[ProductDim] ON SaleFact.Product_created_ID = ProductDim.Product_created_ID
GROUP BY Product_SKU;



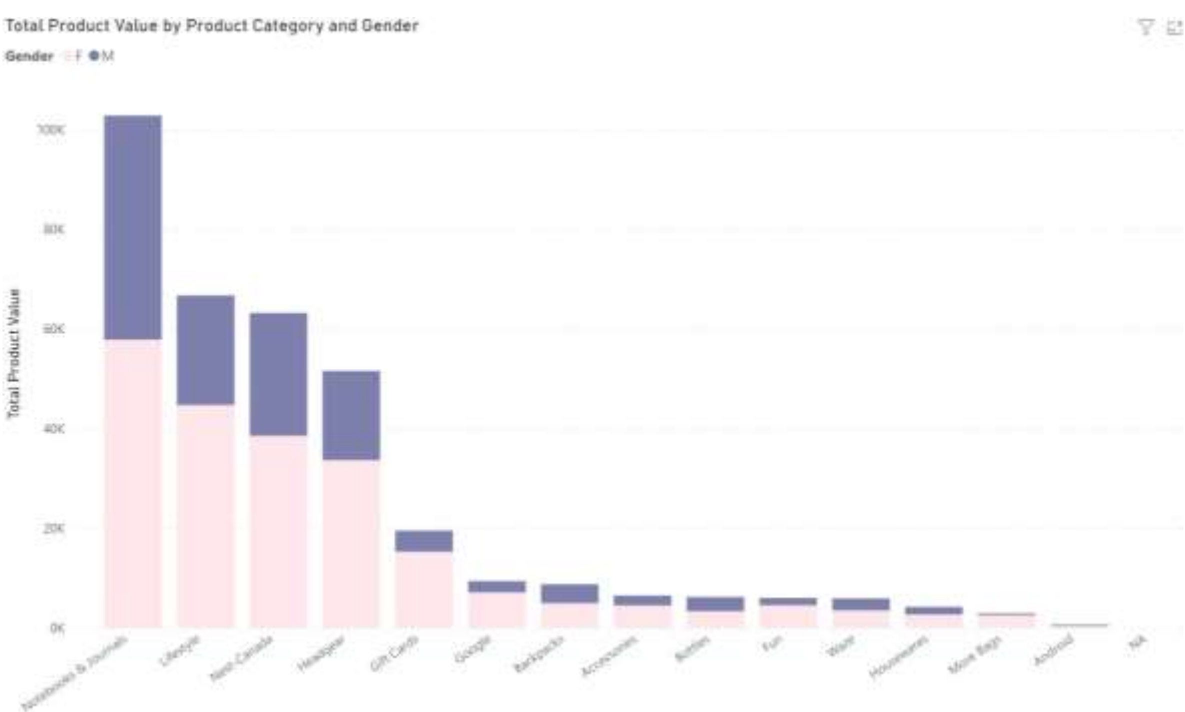
8. Visualizations in Power Bi

Stacked bar chart showing the Total Product Value by Product Category and Gender.

how product preference or spending might differ between genders across different categories, useful for marketing analysis or inventory management.
We see which categories are the most valuable to the company and could be a focus for future business strategies.

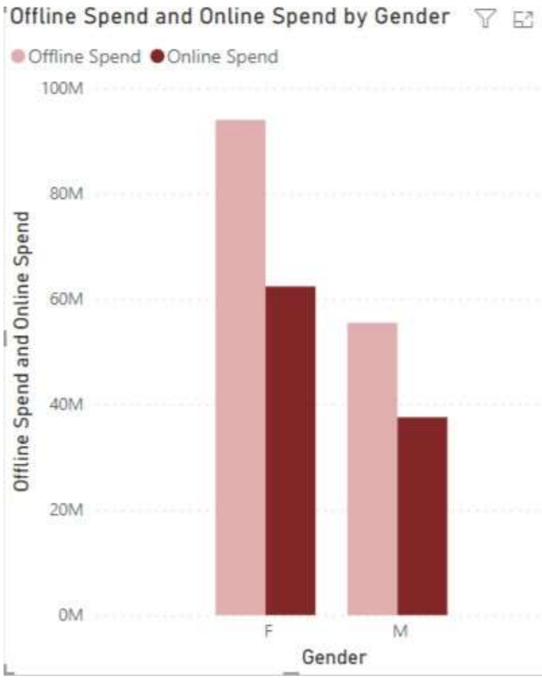


We remove the first product categories to display the categories with smaller total prodict value better, because some categories have very small values, their segments are not be visible and is hard to compare accurately.



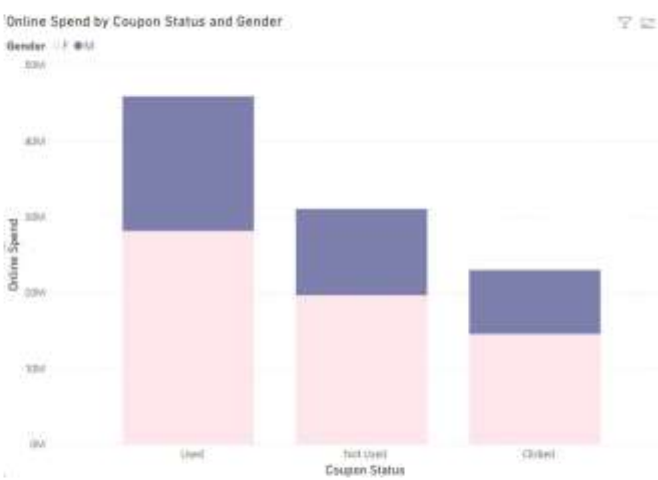
Grouped bar chart comparing Offline Spend and Online Spend by Gender

It shows the differnecies between offline and online spending for each gender, which can be crucial for understanding consumer behavior in different sales channels.



Stacked bar chart showing Online Spend by Coupon Status and Gender

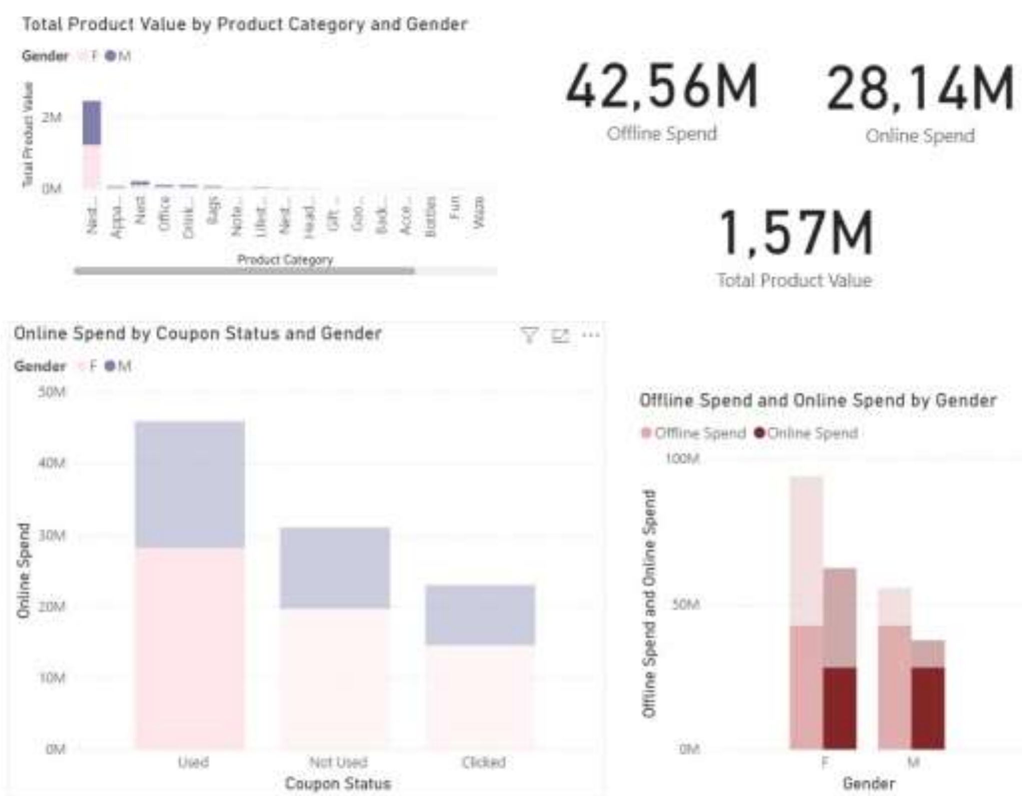
information about gender on top of the coupon status, which allows for an in-depth comparison of how each gender interacts with coupons during their online spending.



In total we have :

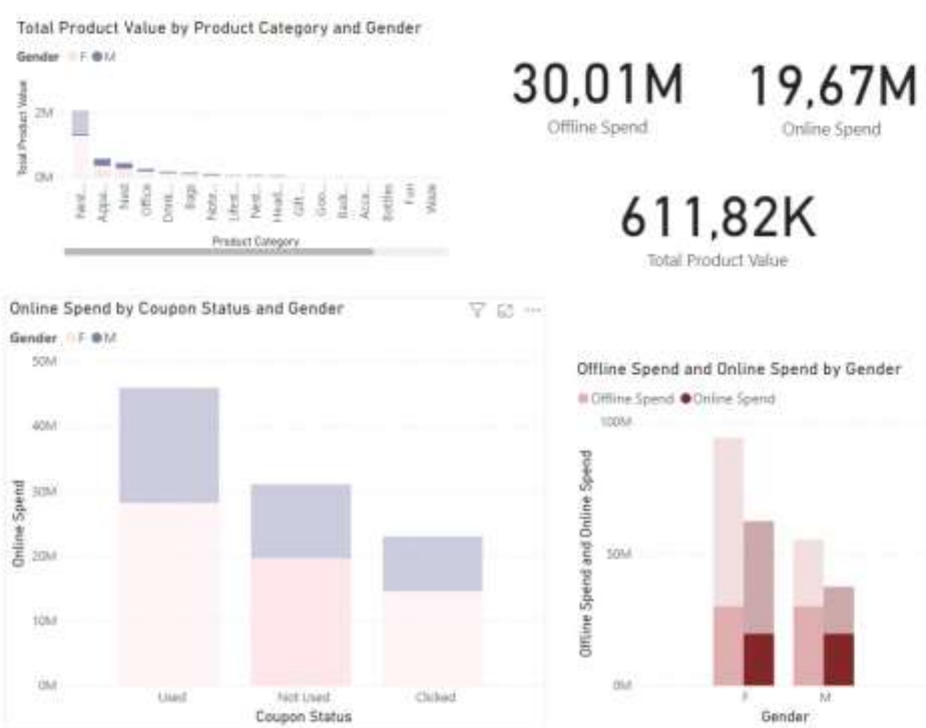
Here if we choose to see all this praphs only if there is used coupon :

We look only at the product categories where a coupon has been used and show the amount of online spend where coupons were used, segmented by gender.Also , we will see accordingly te offline spend , online spend and total product value .



Total Product Value	Product Category	Gender
	Nest-USA	F
104.156,30	Nest	F
54.714,21	Office	F
51.837,04	Drinkware	F
38.079,62	Bags	F
34.749,20	Apparel	F
21.800,67	Lifestyle	F
10.653,77	Nest-Canada	F
9.683,81	Notebooks & Journals	F
5.243,97	Headgear	F
2.916,69	Fun	F
2.398,92	Accessories	F
1.718,21	Housewares	F
1.693,69	More Bags	F
1.161,98	Bottles	F
993,13	Google	F
915,03	Backpacks	F
615,00	Waze	F
89,61	Android	F
0,00	NA	F
1.569.281,44		

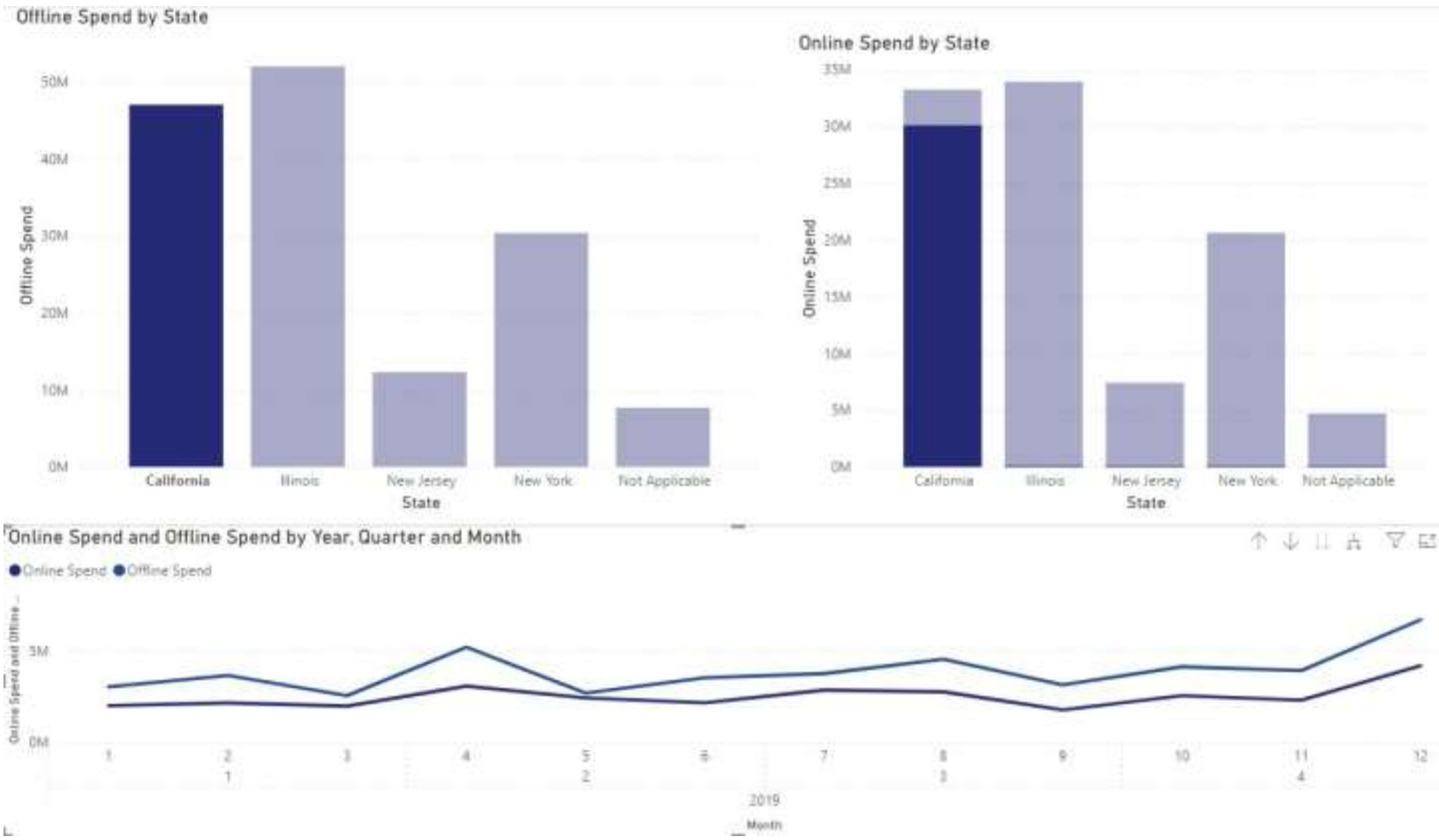
If we click on the coupon status “not used “ we see :



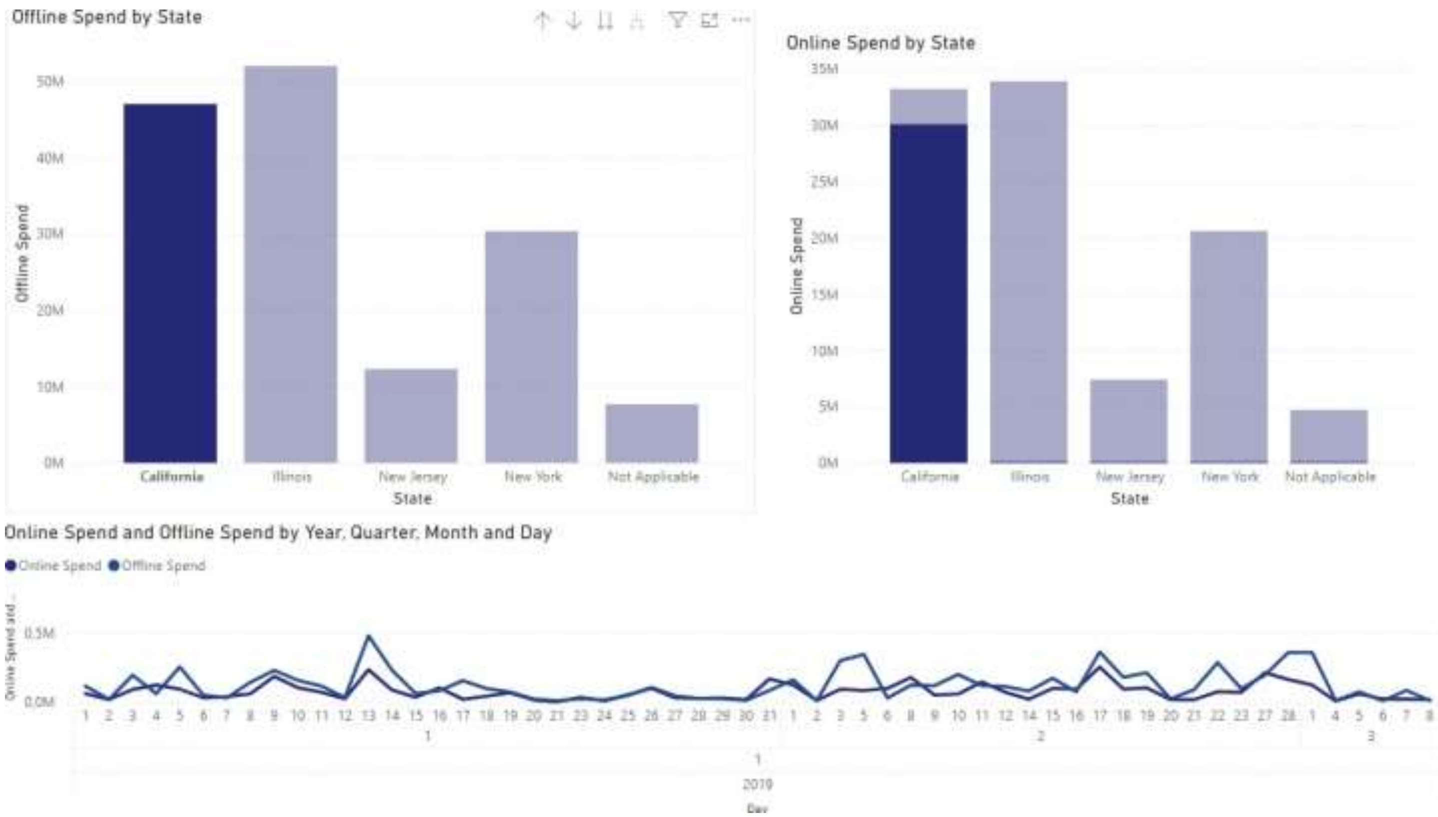
Total Product Value	Product Category	Gender
	Apparel	F
[illegible]	Nest	F
61.586,76	Office	F
37.982,03	Nest-USA	F
32.953,23	Bags	F
32.680,90	Notebooks & Journals	F
27.414,59	Drinkware	F
16.296,95	Nest-Canada	F
15.129,43	Headgear	F
9.524,94	Lifestyle	F
1.997,53	Google	F
1.455,00	Gift Cards	F
1.302,24	Backpacks	F
1.271,24	Waze	F
937,84	Bottles	F
721,27	Accessories	F
651,28	Housewares	F
356,94	Fun	F
217,07	Android	F
123,13	More Bags	F
0,00	NA	F
611.817,21		

The visualization presents three distinct charts comparing offline and online spending habits. The first two **<u>are bar charts showing the offline and online spend by state,</u>** highlighting spending differences across California, Illinois, New Jersey, New York, and unspecified locations. The third is **<u>a line graph comparing online and offline spending over time, broken down by year, quarter, and month.</u>**

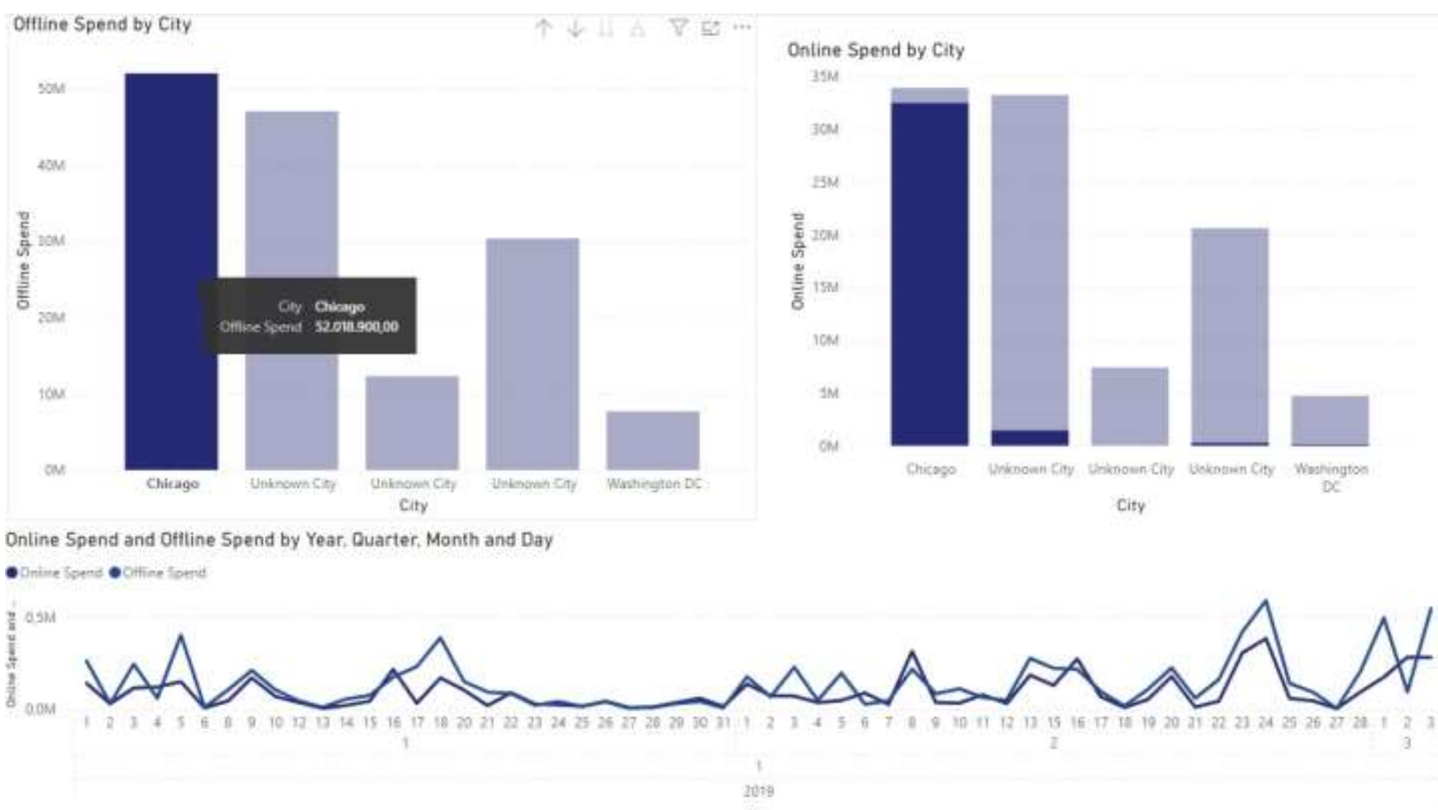
This visualization is useful because it provides a clear comparison of spending habits in different states and trends over time. Are good to understand geographical and temporal patterns in customer behavior. For instance, we can decide from these graphs where to focus marketing efforts or where to consider expanding or reducing physical or digital presence .Also, the temporal spend trends can help in forecasting sales, managing inventory, and planning promotional activities around specific times when spend levels fluctuate.



Here we drill down to show the line graph for the days :

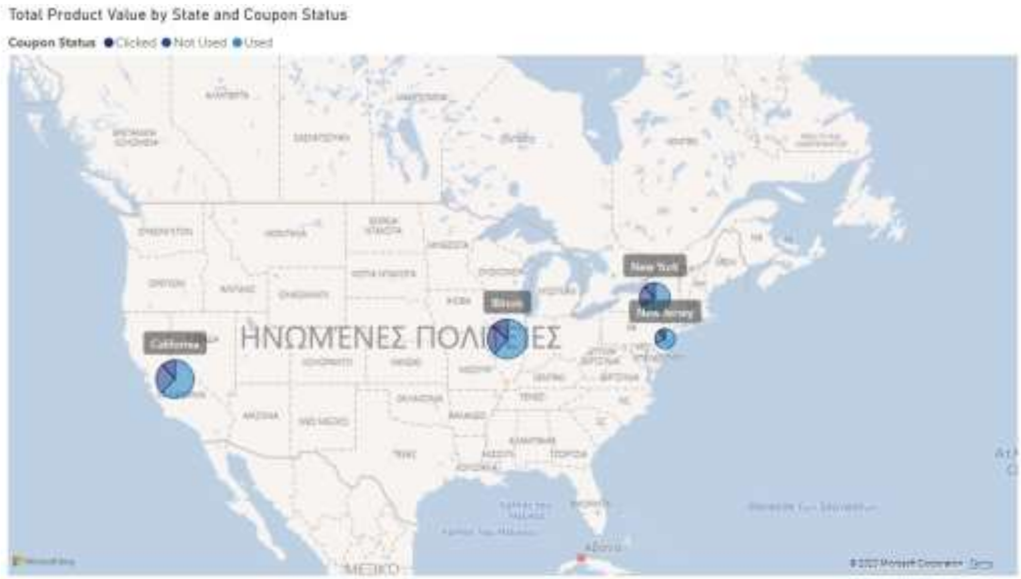


And here we drill down in the location hierarchie to show the offline and online spend by city :



The visualization is a map overlay with pie charts placed over California, Illinois, New Jersey, and New York, indicating the Total Product Value by State and Coupon Status. Each pie is divided into segments representing the proportion of total product value (larger pies indicating a higher total value) that is associated with different statuses of coupon usage: clicked, not used, and used.

This type of visualization is useful for geographically presenting data to quickly identify regional patterns in coupon usage and product value. It allows us to see which states have higher engagement with coupons and where the marketing strategies involving coupons are more or less effective. This can inform decisions on where to allocate marketing resources or how to adjust coupon strategies to maximize product value and customer engagement. The map provides a quick visual reference that can be more intuitive than raw numbers or tables for spatial distribution and regional trends.



Next , we show a pie chart and a bar graph, accompanied by a table, all representing sales data by product category and SKU (Stock Keeping Unit).

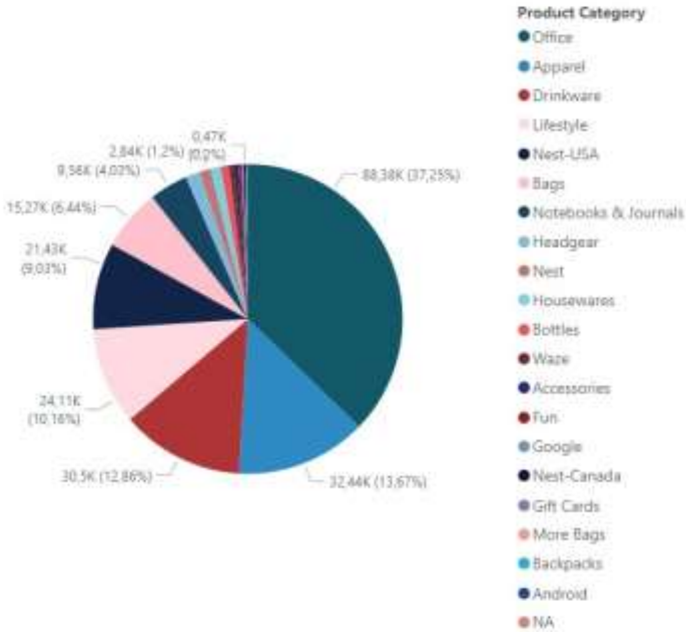
Pie Chart (Quantity by Product Category): This chart shows the distribution of quantities sold across different product categories. Categories like 'Nest-USA' and 'Bags' appear to have a larger share of the total quantity sold, indicating their popularity or high sales volume within the dataset.

Bar Graph (Quantity by Product SKU): The bar graph ranks product SKUs by the quantity sold. The longer bars represent SKUs with higher sales volumes, making it easy to identify top-selling products.

Table (Product Details): The table lists product SKUs with corresponding quantities and average prices. It provides a detailed numerical breakdown, useful for more in-depth analysis.

This visualization is useful for identifying best-selling items and understanding product performance across different categories. It can inform inventory management, purchasing decisions, and marketing strategies. For instance, SKUs with high quantities sold might be prioritized in promotional campaigns, while those with lower sales might be evaluated to determine if they should be discontinued or promoted differently.

Quantity by Product Category



Product SKU	Quantity	Avg Price
GGOEYOLR080599	361	189,07
GGOEYOLR018699	1443	790,70
GGOEYOCR078099	445	252,33
GGOEYOCR077799	1488	775,09
GGOEYOCR077399	246	232,73
GGOEYOCB092699	53	129,99
GGOEYOBR078599	168	149,78
GGOEYHPB072210	1084	1.887,70
GGOEYHPA003610	38	352,50
GGOEYHPA003510	94	514,76
GGOEYFKQ020699	2132	573,09
GGOEYDHJ056099	1080	533,76
GGOEYDHJ019399	779	696,31
GGOEYAYR068656	39	422,64
GGOEYAYR068626	47	384,18
GGOEYAYR068625	79	651,71
GGOEYAYR068624	50	550,67
GGOEYAXR066155	35	295,60
GGOEYAXR066130	10	142,71
GGOEYAXR066129	22	218,42
GGOEYAXR066128	13	146,10
GGOEYAXQ089555	14	97,49
GGOEYAXQ089529	6	68,97
GGOEYAXQ089528	14	81,50
GGOEYAXB089655	3	48,90
GGOEYAXB089630	1	16,30
GGOEYAXB089629	1	16,30
GGOEYAWR062350	2	31,12
GGOEYAWR062349	4	44,60
Total	**237266**	**2.764.427,53**

Quantity by Product SKU

Appendix

R code

We run this code only to take a look of our data and decide what we will do in ssis

```
file.choose()                                          # file path
as1 <-read.csv("D:\\sql\\archive (3)\\file.csv", sep = ";")  # reading the file into a DataFrame

# Assuming your dataframe is named as1
as1 <- subset(as1, select = -c(X, Date))

str(as1)      #it gives me the basic structure of the data
head(as1)     #it gives me the top rows of the data
summary(as1)  #it gives me a summary or statistical description of the mean,median generally the quartiles
describe(as1)
#___________clean___________________________________________________________________________
# Initialize an empty list to store the results
resultsas1 <- list()

# Loop through each column in the dataset
for (col in names(as1)) {
  na_count <- sum(is.na(as1[[col]]))        # Count NA values
  blank_count <- sum(as1[[col]] == "")      # Count blank values
  resultsas1[[col]] <- list(NA_Count = na_count, Blank_Count = blank_count)
}

# Display the results
resultsas1

# Remove records
as1 <- as1[!(is.na(as1$CustomerID) | as1$CustomerID == ""), ]
as1 <- as1[!(is.na(as1$Transaction_Date) | as1$Transaction_Date == ""), ]
as1 <- as1[!(is.na(as1$Product_SKU) | as1$Product_SKU == ""), ]
as1 <- as1[!(is.na(as1$Product_Description) | as1$Product_Description == ""), ]
as1 <- as1[!(is.na(as1$Coupon_Code) | as1$Coupon_Code == ""), ]


as1$CustomerID <- as.factor(as1$CustomerID)
as1$Gender <- as.factor(as1$Gender)
as1$Location <- as.factor(as1$Location)  # Keep as character or convert to factor if there's a limited set of locations
as1$Tenure_Months <- as.numeric(as1$Tenure_Months)  # Already numeric, ensure it's treated as such
as1$Transaction_ID <- as.factor(as1$Transaction_ID)  # Convert to factor for categorical analysis
as1$Transaction_Date <- as.Date(as1$Transaction_Date, format="%m/%d/%Y")  # Adjust format as per your data
as1$Product_SKU <- as.character(as1$Product_SKU)  # Keep as character
as1$Product_Description <- as.character(as1$Product_Description)  # Keep as character
as1$Product_Category <- as.factor(as1$Product_Category)  # Convert to factor
as1$Quantity <- as.numeric(as1$Quantity)  # Keep as numeric
as1$Avg_Price <- as.numeric(as1$Avg_Price)  # Keep as numeric
as1$Delivery_Charges <- as.numeric(as1$Delivery_Charges)  # Keep as numeric
as1$Coupon_Status <- as.factor(as1$Coupon_Status)  # Convert to factor
as1$GST <- as.numeric(as1$GST)  # Keep as numeric
as1$Offline_Spend <- as.numeric(as1$Offline_Spend)  # Keep as numeric
as1$Online_Spend <- as.numeric(as1$Online_Spend)  # Keep as numeric
as1$Month <- as.factor(as1$Month)  # Convert to factor
```

```
as1$Coupon_Code <- as.character(as1$Coupon_Code)  # Keep as character or convert to factor if it's categorical
as1$Discount_pct <- as.numeric(as1$Discount_pct)  # Keep as numeric


str(as1)      #it gives me the basic structure of the data
head(as1)     #it gives me the top rows of the data
summary(as1)  #it gives me a summary or statistical description of the mean,median generally the quartiles
describe(as1)

#______________________________________________________________________________________________

print_unique_values <- function(data) {
  lapply(data, function(column) {
    if (is.character(column)) {
      unique(column)
    }
  })
}

# Call the function with your DataFrame
unique_values <- print_unique_values(as1)
unique_values

length(unique(as1$CustomerID))

unique(as1$Tenure_Months)

sum(is.na(as1$Product_SKU)) ;sum(as1$Product_SKU == "")

sum(is.na(as1$Product_Description));sum(as1$Product_SKU == "")



#___________________statistics for categorical data_____________________
calculate_categorical_stats <- function(data) {

  # Extracting categorical columns (factors and characters)
  cat_columns <- sapply(data, function(x) is.factor(x) || is.character(x))
  categorical_data <- data[, cat_columns]

  # Initialize a list to store results
  results <- list()

  # Loop through each categorical column
  for (column in names(categorical_data)) {
    cat_stats <- list()

    # Frequency and relative frequency
    freq_counts <- table(categorical_data[[column]])
    rel_freq <- prop.table(freq_counts)

    # Mode
    mode_val <- Mode(categorical_data[[column]])

    # Descriptive statistics using psych package
    desc_stats <- describe(categorical_data[[column]])
```

```
    # Store results
    cat_stats$Frequency <- freq_counts
    cat_stats$RelativeFrequency <- rel_freq
    cat_stats$Mode <- mode_val
    cat_stats$Description <- desc_stats

    results[[column]] <- cat_stats
  }

  return(results)
}

calculate_categorical_stats(as1)


calculate_categorical_stats_for_column <- function(data, column_name) {
  # Check if the column name is valid
  if (!column_name %in% names(data)) {
    stop("Column name not found in the dataframe")
  }

  # Check if the column is categorical
  if (!is.factor(data[[column_name]]) && !is.character(data[[column_name]])) {
    stop("The specified column is not categorical")
  }

  # Initialize a list to store results
  cat_stats <- list()

  # Frequency and relative frequency
  freq_counts <- table(data[[column_name]])
  rel_freq <- prop.table(freq_counts)

  # Mode
  mode_val <- Mode(data[[column_name]])

  # Descriptive statistics using psych package
  desc_stats <- describe(data[[column_name]])

  # Store results
  cat_stats$Frequency <- freq_counts
  cat_stats$RelativeFrequency <- rel_freq
  cat_stats$Mode <- mode_val
  cat_stats$Description <- desc_stats

  return(cat_stats)
}


calculate_categorical_stats_for_column(as1,"Product_SKU")

calculate_categorical_stats_for_column(as1,"Product_Category")

calculate_categorical_stats_for_column(as1,"Coupon_Status")
```