Project Report - JADARA Mart

Team 6

Made by:

- **Mohamed Ibrahim**
  Business Analyst / Data Modeler
- **Ahmed Said Hammad**
  Data Engineer
- **Mohamed Tamer**
  Data Engineer
- **Mohamed Attia**
  Data Engineer
- **Marwan Amr**
  Data Modeler / Semantic
- **Halla Ibrahim**
  Data Modeler
- **Salma Fathi**
  Semantic / Data Analyst
- **Adeem**
  Data Analyst

Supervised By:

- **Eng. Asmaa Yassin**
- **Eng. Muhammed Sanad**

Compiled by:

- Mohamed Ibrahim

Project Overview and Background

Jadara Mart is a retail chain with multiple physical branches and an online commerce platform. This project aims to develop data analysis dashboards that provide insights into various aspects of the business, such as sales performance, inventory optimization, and logistics performance. The focus is on leveraging data engineering and analysis to improve decision-making and operational efficiency.

The purpose is to provide business with detailed insights, so that decisions are data driven.

As part of requirement gatherings, following activities were covered:

- Understanding of high-level business processes.
- Process understanding and data requirements gathering.
- Understanding current reporting methodology like data gathering, data preparation, business logic, frequency of reporting and schedule for data refresh if any.
- Identification of key KPI's/Metrics /Measures required to address the reporting requirements.

Data Sources:

Below are the data sources identified.

- SQL Server for physical branches.
- MYSQL Server for online commerce.

Business Requirements:

This section describes current process flow, key business definitions and metrics.

1. About current process:

The current business processes at Jadara Mart involve various operational activities, including sales, inventory management, customer interactions, and logistics. These processes are critical for ensuring the smooth functioning of both physical branches and the online commerce platform. However, the existing system lacks a comprehensive analytical framework to provide deep insights into these operations. The data is often siloed and not easily accessible for analysis, leading to challenges in decision-making and operational efficiency.

Key components of the current process include:

Sales: Sales data is collected from POS systems in physical branches and the online platform. This data is used for basic reporting but lacks detailed analysis by product, vendor, and customer segments.

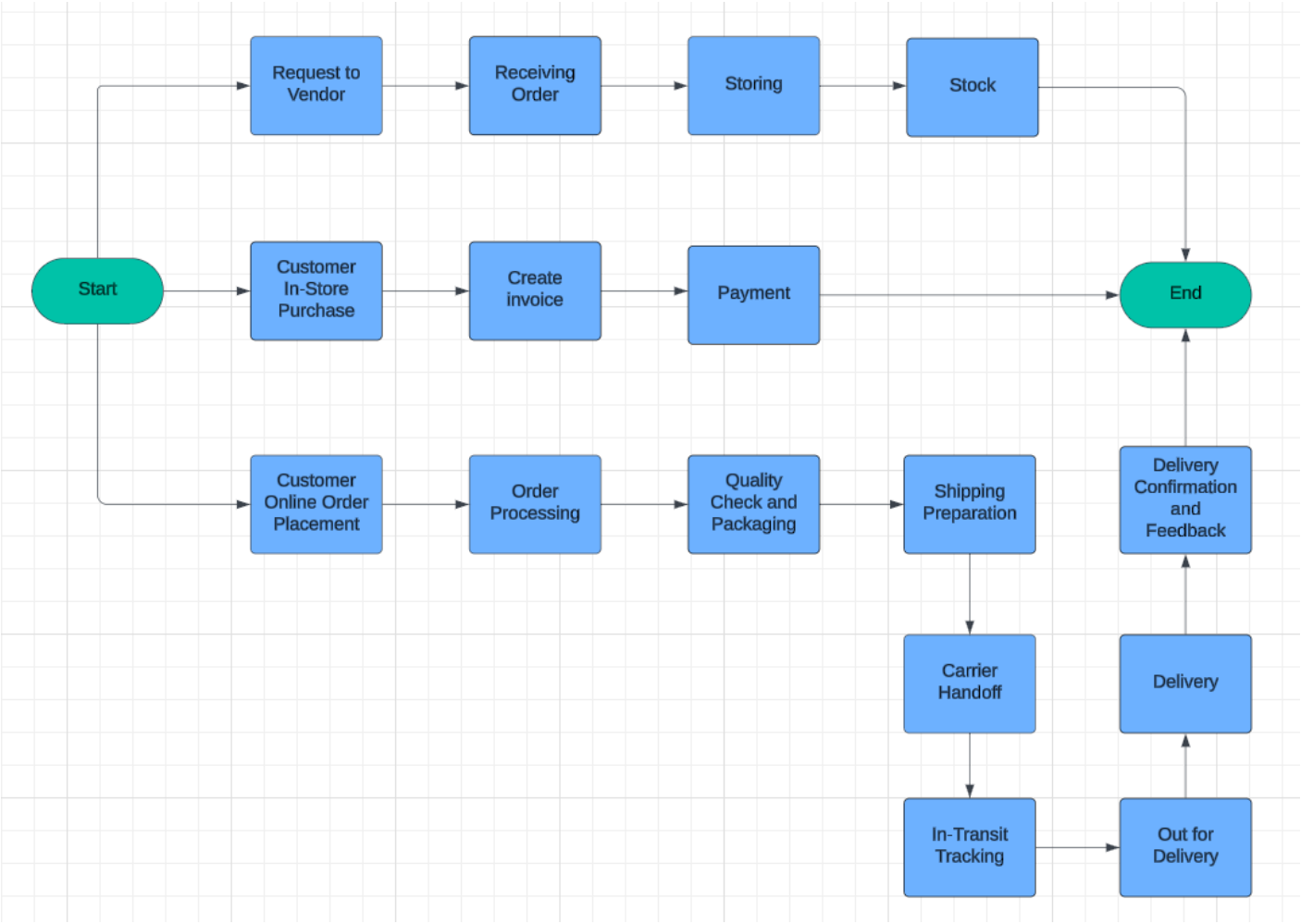
Inventory Management: Inventory levels are tracked manually or through basic systems, with limited forecasting and optimization capabilities. This can lead to overstocking or stockouts.

Customer Management: Customer data is collected but not fully leveraged for personalized marketing or customer segmentation.

Logistics: The logistics process involves managing the supply chain from suppliers to branches and customers. There is a need for better tracking of delivery performance and customer feedback.

2. High Level Process Flow:

Below are the steps in the process along with owners at each step:



Project Steps

1. Business Requirement Document (BRD) and Vision Requirement Document (VRD)

Our project began with the creation of two critical documents: the Business Requirement Document (BRD) and the Vision Requirement Document (VRD). The BRD was developed outlining the key business needs, goals, and objectives.

Meanwhile, the VRD, provided a high-level overview of the project's vision and long-term goals, serving as a foundation for all subsequent technical and design decisions.

The BRD was developed by Mohamed Ibrahim.

The VRD was created by Mohamed Tamer, Salma Fathy, Adeem.

2. Entity Relationship Diagrams (ERD) Creation

Once the requirements were documented, we proceeded with designing the system's architecture. Two distinct Entity Relationship Diagrams (ERDs) were developed:

- One for the **physical stores**: This ERD detailed the structure and relationships specific to the operations in our physical store branches, including inventory management, sales, and customer interactions.

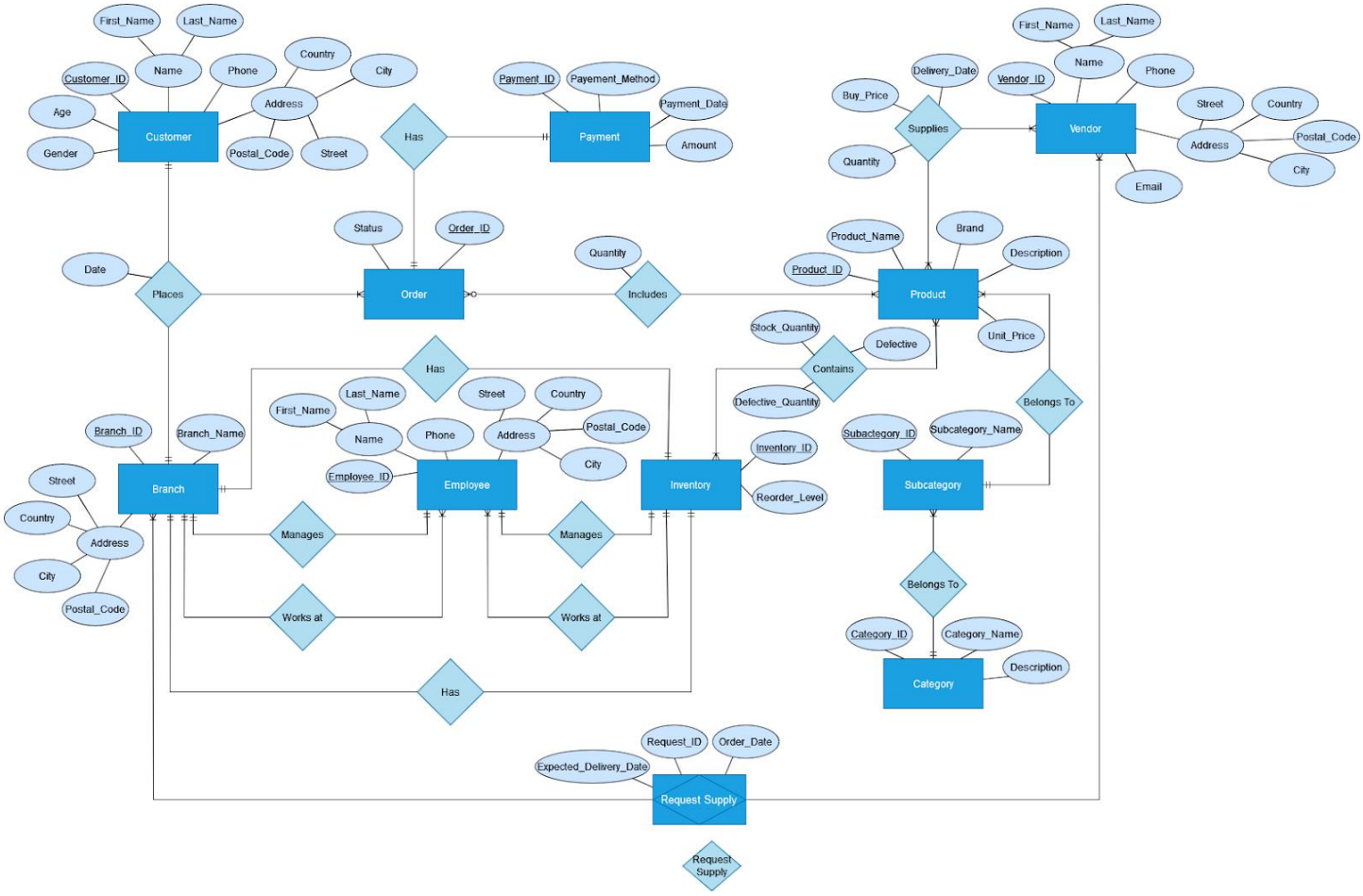
Physical Store ERD Team: Marwan Amr, Mohamed Ibrahim

- One for the **online store**: This ERD was tailored for the online commerce platform, addressing the unique components related to online transactions, customer accounts, and order processing.

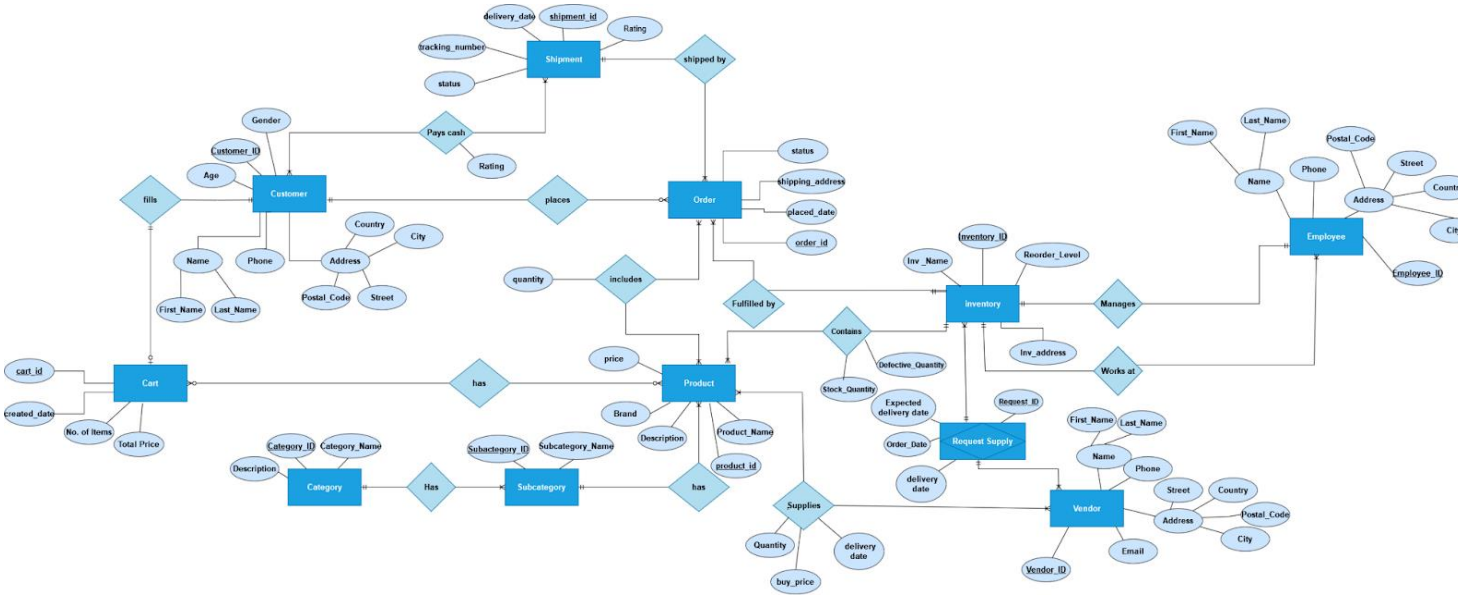
Online Store ERD Team: Halla Ibrahim, Mohamed Ibrahim

Below are the entity relationship diagrams for physical store and online commerce platform:

Physical Store



Online Store



3. Database Design and Implementation

Following the ERD creation, we designed the corresponding databases based on the relationships and entities identified. The databases were created with scalability and performance in mind, ensuring seamless integration between the physical and online systems.

Database Design and Implementation Team: Marwan Amr, Halla Ibrahim, Mohamed Ibrahim

4. Data Insertion with Logical Scripts

After the database structures were in place, we manually created scripts to insert data into the system. These scripts were crafted with careful consideration to ensure the data was as realistic and logical as possible, reflecting real-world scenarios. This approach was vital to produce meaningful and accurate results during the data analysis phase. The inserted data helped simulate real-world operations, providing us with a strong foundation for generating insightful analyses.

Data Script Development Team: All team members actively contributed to this task

Mohamed Ibrahim - 9 Tables

Marwan Amr - 8 Tables

Mohamed Tamer - 8 Tables

Ahmed Hammad - 3 Tables

Halla Ibrahim - 3 Tables

Salma Fathi - 3 Tables

Mohamed Attia - 2 Tables

Adeem - 2 Tables

These scripts were designed to provide meaningful input for accurate analysis and performance tracking.

Some Examples

Customers Data Insertion

```python
male_names = [
    'Ahmed', 'Mohamed', 'Mahmoud', 'Mostafa', 'Amr', 'Youssef', 'Tarek', 'Hassan', 'Ali', 'Omar',
    'Karim', 'Nabil', 'Samir', 'Anwar', 'Fadi', 'Magdy', 'Ehab', 'Ramy', 'Hossam', 'Ibrahim', 'Ayman',
    'Adel', 'Wael', 'Ashraf', 'Sameh', 'Sherif', 'Khaled', 'Essam', 'Galal', 'Hani', 'Nasser', 'Osama',
    'Ismail', 'Yasser', 'Saad', 'Abdelrahman', 'Waleed', 'Fathy', 'Reda', 'Moustafa', 'Basem', 'Ziad',
    'Ragab', 'Zakaria', 'Salah', 'Lotfy', 'Sayed', 'Farouk', 'Fares', 'Bassam', 'Sherif', 'Hatem', 'Magid',
    'Nour', 'Hamza', 'Kamel', 'Rafik', 'Rami', 'Ameen', 'Magdy', 'Ezzat', 'Fekry', 'Gamal', 'Hamed', 'Mahmoud',
    'Hamdy', 'Rady', 'Ashraf', 'Hassan', 'Salem', 'Mazen', 'Sherif', 'Maged', 'Taher', 'Rashed', 'Abdallah',
    'Moataz', 'Sobhy', 'Sayed', 'Ibrahim', 'Rashad', 'Younes', 'Salah', 'Ashraf', 'Maher', 'Lotfy', 'Azmy',
    'Zakaria', 'Fathy', 'Ismail', 'Osama', 'Sherif', 'Rabea', 'Raouf', 'Gamal', 'Helmy', 'Mohamed', 'Mamdouh', 'Hatem',
    'Ezzat', 'Tawfik', 'Salah', 'Waheed', 'Kamel', 'Rashad', 'Khaled', 'Hany', 'Amr', 'Magdy', 'Ayman', 'Ibrahim',
    'Karim', 'Mohey', 'Tawfik', 'Samy', 'Abdelrahman', 'Mostafa', 'Khalil', 'Hamdy', 'Rady', 'Fathy', 'Sameh', 'Lotfy',
    'Sherif', 'Hamed', 'Zaki', 'Magdy', 'Adel', 'Taher', 'Naguib', 'Abdelaziz', 'Kamal', 'Khalaf', 'Ashraf', 'Ramzy',
    'Basem', 'Nabil', 'Mounir', 'Atef', 'Yasser', 'Salah', 'Shady', 'Fares', 'Gamal', 'Ahmed', 'Tarek', 'Farid', 'Lotfy',
    'Mahmoud', 'Mamdouh', 'Hisham', 'Helmy', 'Anwar', 'Hatem', 'Gamal', 'Magdy', 'Khalil', 'Hamdy', 'Rady', 'Fathy',
    'Ismail', 'Osama', 'Sherif', 'Lotfy', 'Salah', 'Mounir', 'Kamal', 'Ashraf', 'Rami', 'Khaled', 'Karim', 'Amr', 'Mohamed',
    'Mostafa', 'Ayman', 'Adel', 'Hani', 'Nabil', 'Samir', 'Anwar', 'Fadi', 'Magdy', 'Ehab', 'Ramy', 'Hossam', 'Ibrahim',
    'Wael', 'Ashraf', 'Sherif', 'Khaled', 'Essam', 'Galal', 'Hani', 'Nasser', 'Osama', 'Ismail', 'Yasser', 'Saad', 'Abdelrahman', 'Waleed'
]

female_names = [
    'Fatima', 'Amina', 'Maryam', 'Sara', 'Nadia', 'Salma', 'Hoda', 'Mona', 'Dalia', 'Leila', 'Yasmin', 'Rania', 'Heba', 'Samira',
    'Nesma', 'Amal', 'Ghada', 'Reem', 'Naglaa', 'Manal', 'Dina', 'Marwa', 'Asmaa', 'Heba', 'Nourhan', 'Amal', 'Shaimaa', 'Maha',
    'Iman', 'Rasha', 'Hend', 'Noha', 'Eman', 'Zeinab', 'May', 'Lamia', 'Esraa', 'Hanan', 'Doaa', 'Sahar', 'Nour', 'Lubna',
    'Hebatallah', 'Menna', 'Yasmine', 'Rania', 'Aisha', 'Nesreen', 'Nesma', 'Mariam', 'Nada', 'Riham', 'Samaa', 'Lobna', 'Nermin',
    'Salma', 'Asmaa', 'Hager', 'Fayrouz', 'Layla', 'Sherine', 'Safaa', 'Lobna', 'Mona', 'Walaa', 'Zeina', 'Hebatallah', 'Noura',
    'Samar', 'Ola', 'Enaam', 'Rawya', 'Farida', 'Enas', 'Hana', 'Lobna', 'Mervat', 'Samah', 'Neveen', 'Riham', 'Dalia', 'Doaa',
    'Hend', 'Gihan', 'Hoda', 'Safaa', 'Sherin', 'Dina', 'Reham', 'Amal', 'Aya', 'Habiba', 'Mervat', 'Rasha', 'Safaa', 'Samar',
    'Salwa', 'Shaimaa', 'Fatma', 'Abeer', 'Fatima', 'Iman', 'Israa', 'Leila', 'Malak', 'Marwa', 'Nada', 'Nevine', 'Nehal',
    'Nisreen', 'Rania', 'Samia', 'Shahira', 'Shereen', 'Zeina', 'Zeinab', 'Amal', 'Asmaa', 'Dina', 'Doaa', 'Fayrouz', 'Ghada',
    'Habiba', 'Hager', 'Hana', 'Hania', 'Heba', 'Hoda', 'Iman', 'Leila', 'Lobna', 'Lubna', 'Maha', 'Manal', 'Mariam', 'May',
    'Mona', 'Nada', 'Naglaa', 'Nabila', 'Nada', 'Nadeen', 'Naira', 'Nermin', 'Nesma', 'Nesreen', 'Nihad', 'Nihal', 'Nisreen',
    'Nour', 'Nourhan', 'Rania', 'Rawya', 'Reem', 'Riham', 'Samar', 'Salma', 'Salwa', 'Samira', 'Shadia', 'Shaimaa', 'Shereen',
    'Shorouk', 'Sohair', 'Souad', 'Tahani', 'Yasmin', 'Yasmina', 'Zahraa', 'Zeinab', 'Zohra', 'Zainab', 'Zeina', 'Amira', 'Ahlam',
    'Amal', 'Dina', 'Doaa', 'Eman', 'Farida', 'Faten', 'Gihan', 'Ghada', 'Habiba', 'Hala', 'Heba', 'Hend', 'Hoda', 'Ibtisam',
    'Inas', 'Jihan', 'Karima', 'Laila', 'Lobna', 'Maha', 'Magda', 'Marwa', 'Mayada', 'Nabila', 'Nahla'
]
```

```python
streets = {
    'Cairo': [
        'Tahrir St', 'Talaat Harb St', 'Ramses St', 'Salah Salem St', 'Al Nasr Rd',
        'Gameat El Dewal St', 'Kasr El Aini St', 'Mohamed Mahmoud St', 'Abdel Khalek Tharwat St',
        'El Merghany St', 'El Tayaran St', 'El Manial St', 'Gezira St', 'Al Haram St',
        'El Batal Ahmed Abdel Aziz St', 'Ahmed Orabi St', 'Al Azhar St', 'Abdel Moneim Riad St',
        'El Dokki St', 'Al Thawra St'
    ],
    'Alexandria': [
        'El-Tahrir St', 'El-Montaza St', 'El-Raml Station St', 'Stanley Bridge St', 'Abu Qir St',
        'Fouad St', 'El-Agamy St', 'Kafr Abdo St', 'Bolkly St', 'San Stefano St',
        'Corniche Rd', 'Safia Zaghloul St', 'El-Horeya Rd', 'El-Asafra St', 'Victoria St',
        'Smouha St', 'Sidi Gaber St', 'Roshdy St', 'Moharam Bek St', 'Gleem St'
    ],
    'Giza': [
        'Pyramids St', 'Faisal St', 'Al Haram St', 'El-Mohandeseen St', 'Sudan St',
        'El-Bahr El-Azam St', 'El-Malek El-Saleh St', 'Tersana St', 'Ahmed Zewail St',
        'Mansouriya Rd', 'Dokki St', 'Omraneya St', 'Zamzam St', 'Sphinx Sq',
        'Giza St', 'Abu Rawash Rd', 'University St', 'Lebanon St', 'Adnan El Malki St', 'Gezeret El Arab St'
    ],
    'Mansoura': [
        'Gomhoria St', 'Suez Canal St', 'El Mashaya El Soflya St', 'El Mashaya El Ola St',
        'El-Gaish St', 'Ahmed Maher St', 'Port Said St', 'El-Qawmiya El-Arabiya St',
        'Shokri El-Qawatly St', 'Aga St', 'El-Sikka El-Jadida St', 'Gamal Abdel Nasser St',
        'Tanta St', 'El-Mansoury St', 'Nile St', 'El-Geish Bridge St', 'El-Bahr St',
        'El-Mashaya El-Olya St', 'El-Tayaran St', 'El-Galaa St'
    ],
    'Assiut': [
        'El-Geish St', 'Port Said St', 'Abu Tig St', 'El-Sultan Hussein St', 'El-Salam St',
        'Karnak St', 'Talaat Harb St', 'El-Thawra St', 'Abdel Kader El Toukhy St',
        'Al Mahata St', 'University St', 'Abo El-Makarem St', 'Corniche El-Nile St',
        'Helwan St', 'El-Gomhoria St', 'Ain Shams St', 'El-Manshiya St', 'Salah Salem St',
        'Al-Nahda St', 'El-Fath St'
    ],
    'Luxor': [
        'Karnak St', 'Corniche El-Nil St', 'Television St', 'Luxor Temple Rd', 'El-Tod Rd',
        'Abu El-Haggag St', 'Madinet Habu St', 'Sphinx St', 'El-Madinah St', 'El-Mahatta St',
```

```python
# Function to generate random customer data
def generate_customer_data(num_customers):
    global counter
    customers = []
    while len(customers) < num_customers:

        # Generate phone number with specific prefixes
        phone_prefix = random.choice(['011', '012', '010', '015'])
        phone = f'+2{phone_prefix}{random.randint(1000000, 9999999)}'

        postal_code = f'{random.randint(10000, 99999)}'

        # Ensure unique phone and postal code
        if phone in existing_phones or postal_code in existing_postal_codes:
            continue

        print('counter =',counter)
        counter+=1
        gender = random.choice(['Male', 'Female'])
        if gender == 'Male':
            first_name = random.choice(male_names)
        else:
            first_name = random.choice(female_names)
        last_name = random.choice(male_names)

        existing_phones.add(phone)
        existing_postal_codes.add(postal_code)

        country = 'Egypt'
        city = random.choice(cities)
        street = random.choice(streets[city])
        age = random.randint(18, 70)
        email = f"{first_name.lower()}.{last_name.lower()}{random.randint(1, 9999)}@onlinejadara.com"
        customers.append((counter, first_name, last_name, phone, country, city, street, postal_code, age, gender, email))

    return customers
```

Shipments Data Insertion with status weights

```python
# Define the shipment statuses and their corresponding weights
shipment_status = ['Pending', 'Processing', 'Out for Delivery', 'Delivered', 'Returned', 'Canceled']
weights = [0.01, 0.03, 0.02, 0.75, 0.1, 0.09]

def random_date(start_date, end_date):
    """Generate a random date between start_date and end_date."""
    delta = end_date - start_date
    random_days = random.randint(0, delta.days)
    return start_date + timedelta(days=random_days)

start_date = datetime(2022, 1, 1)
end_date = datetime(2024, 7, 31)
counter = 0

# Function to generate random shipment data
def generate_shipment_data(num_shipment):
    global counter
    shipments = []
    while len(shipments) < num_shipment:
        counter += 1
        print('counter =',counter)
        delivery_date = random_date(start_date, end_date)
        tracking_number = counter
        status = random.choices(shipment_status, weights=weights, k=1)[0]
        rating = random.randint(1, 5)

        shipments.append((counter, delivery_date, tracking_number, status, rating))

    return shipments
```

Order Products Data Insertion

```python
# Get all Product IDs
cursor.execute("SELECT Product_ID FROM Product")
product_ids = [row.Product_ID for row in cursor.fetchall()]

counter = 0

# Function to generate random order_products data
def generate_order_products_data():
    global counter
    order_products = []
    for Order_ID in range (1, 80001):
        counter += 1
        print('counter =',counter)
        num = random.randint(1, 10)
        existing_products = set()
        for i in range (num):
            Product_ID = random.choice(product_ids)
            # Ensure unique phone and postal code
            if Product_ID in existing_products:
                continue

            existing_products.add(Product_ID)
            Quantity = random.randint(1, 3)

            order_products.append((Order_ID, Product_ID, Quantity))

    return order_products
```

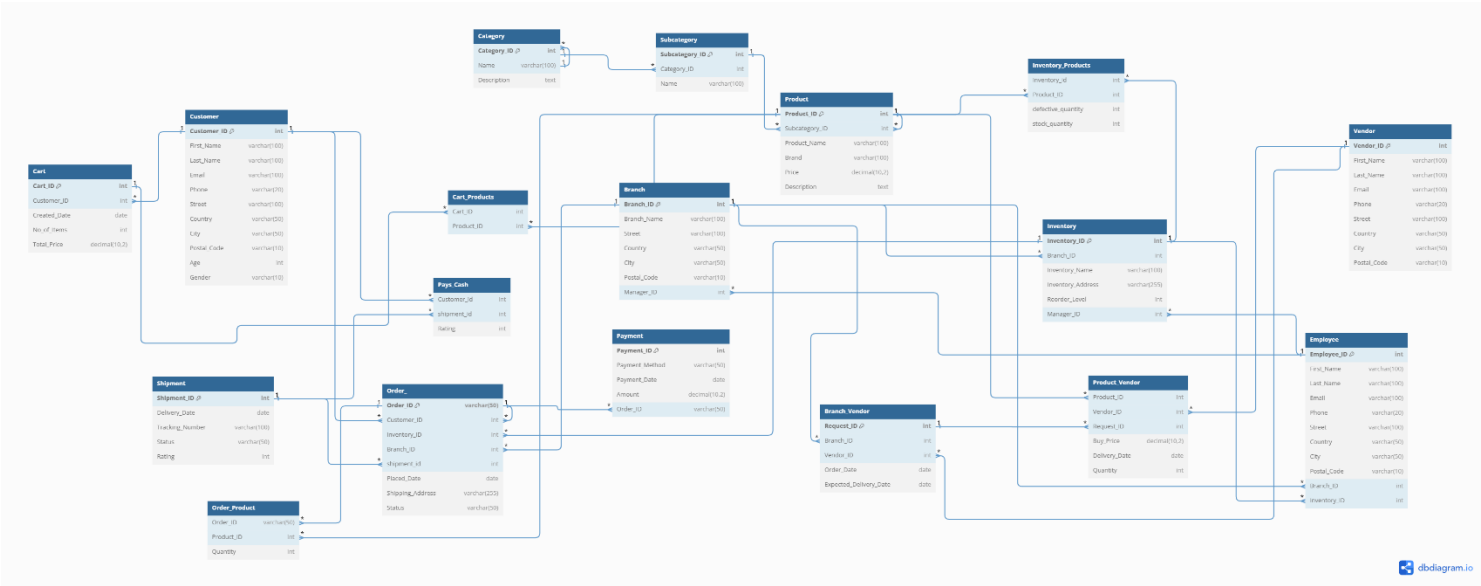
5. Data Engineering

5.1 SSIS

SSIS Development Team: Mohamed Tamer, Ahmed Hammad.

We began by identifying two primary data sources: an SQL Server database for the **physical store** and another SQL Server database for the **online store**. To better simulate a real-world scenario and differentiate between the sources, we **migrated** the online store database from SQL Server to MySQL.

During the process, we faced a challenge at the destination layer due to structural differences between the two source databases, arising from variations in their entities and relationships. To resolve this, we developed a unified schema that harmonized the diverse structures, ensuring compatibility at the destination.



Database Name: Unified_RetailStore

The schema is designed to standardize data across multiple source systems.

Key fields, such as **Hashcode, Source, and Landing Date**, are included in every table to ensure proper data management:

1. **Hashcode:** Uniquely identifies records across different systems, aiding in tracking and de-duplication.

2. **Source:** Captures metadata about the origin of the data, allowing the identification of the source system from which the record was extracted.

3. **Landing_Date:** A timestamp indicating when the data was loaded into the destination table, essential for ETL processes and auditing.

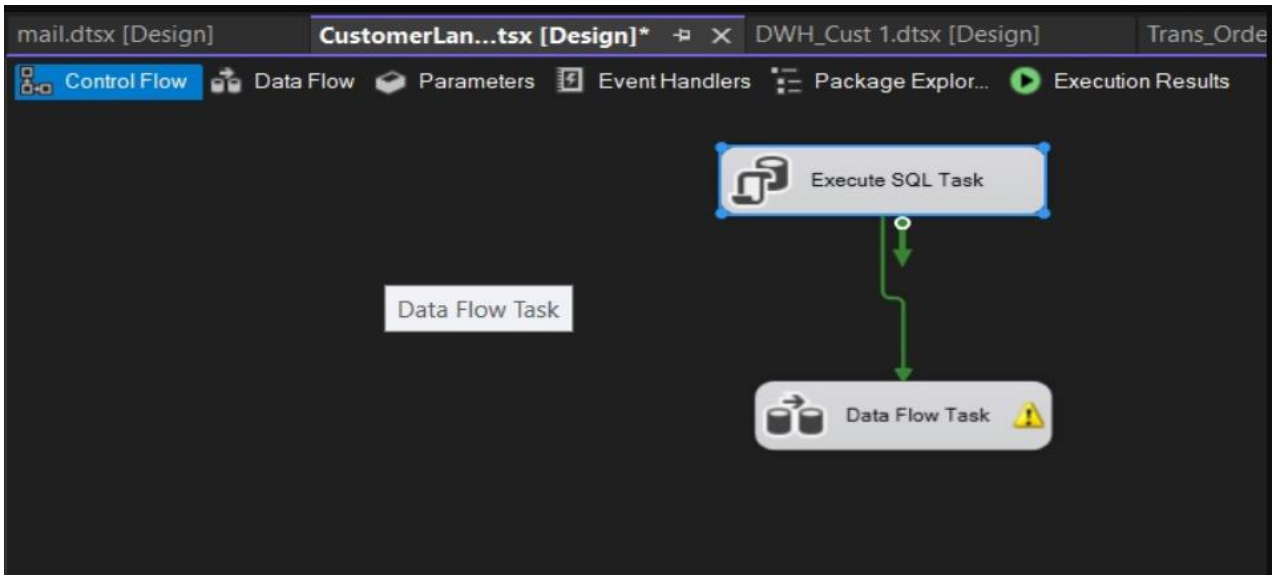
ETL Process for Landing DB

We designed the first ETL process to integrate data from two sources: the SQL Server database for the physical store and the MySQL database for the online store. The destination for this process was the unified schema for the landing database.

Data Type Discrepancies and Resolution

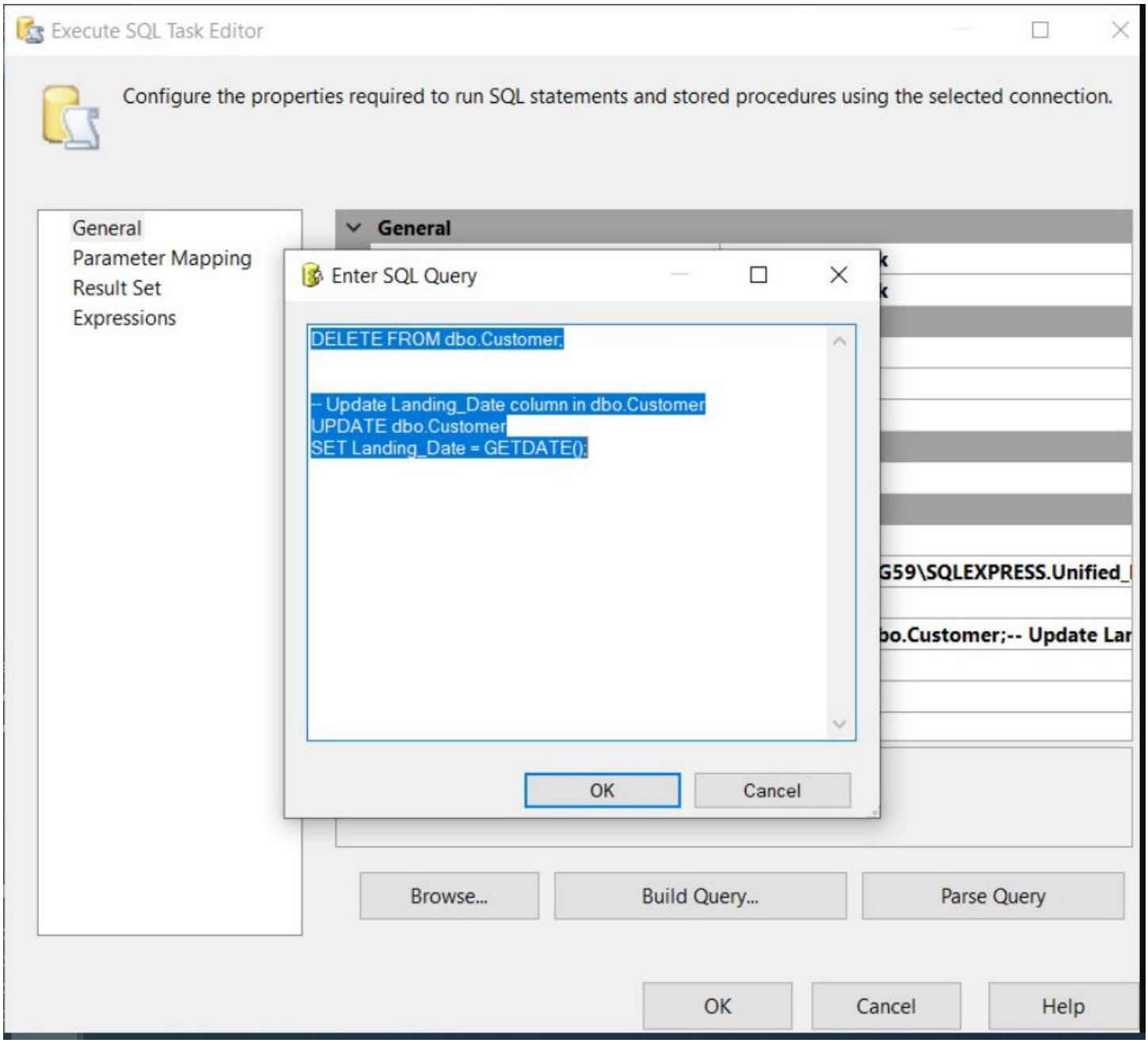
During the initial ETL execution, we encountered a critical issue due to differences in column data types across the two sources, preventing the ETL from running successfully. To resolve this, we used the SSIS (SQL Server Integration Services) Advanced Editor and applied data conversion to align the data types, ensuring compatibility.

ETL Package Structure and Workflow



The ETL process was organized into multiple packages, with each package representing a specific table. Below is an explanation of the components and workflow used in each ETL package:

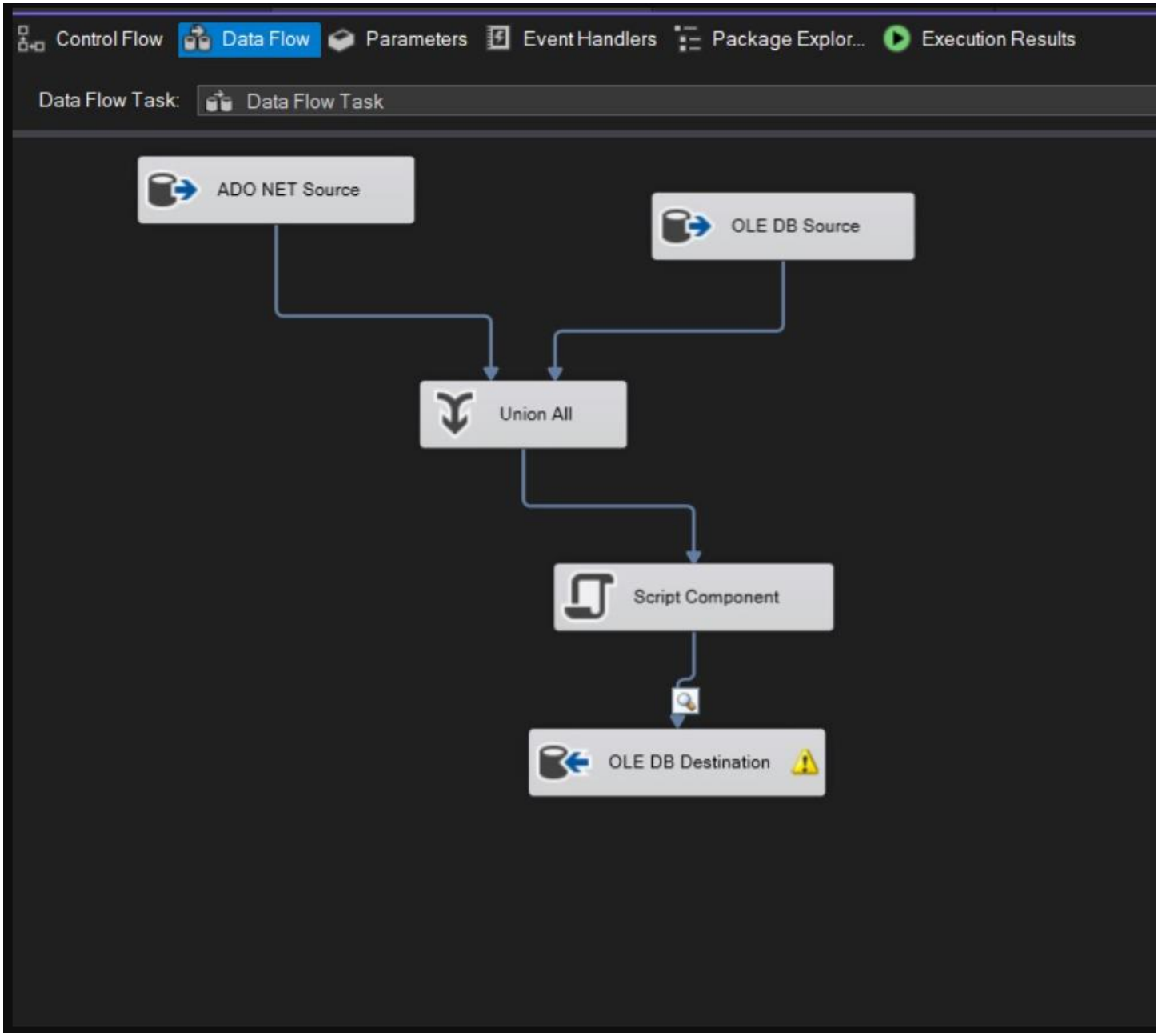
1. SQL Task Component:



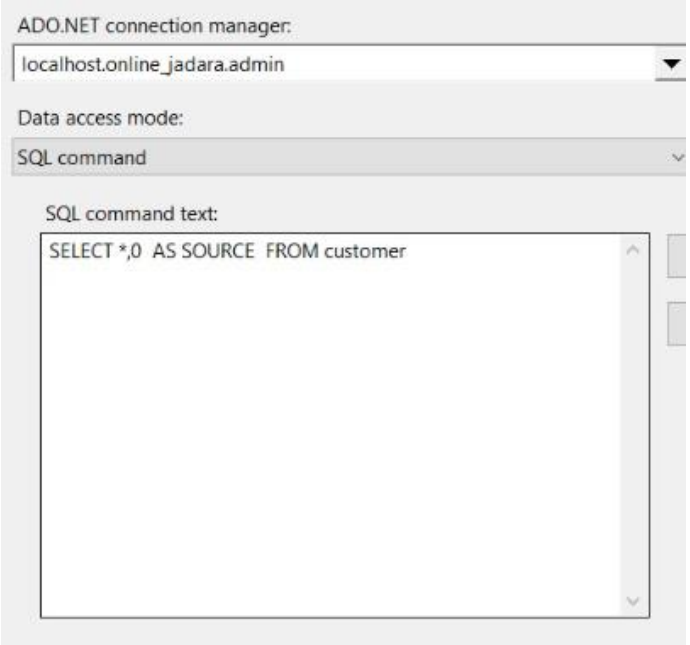
- **Truncate Table:** The process begins by truncating the table before loading, as part of a "truncate and load" strategy, to ensure that all data is refreshed during each run.

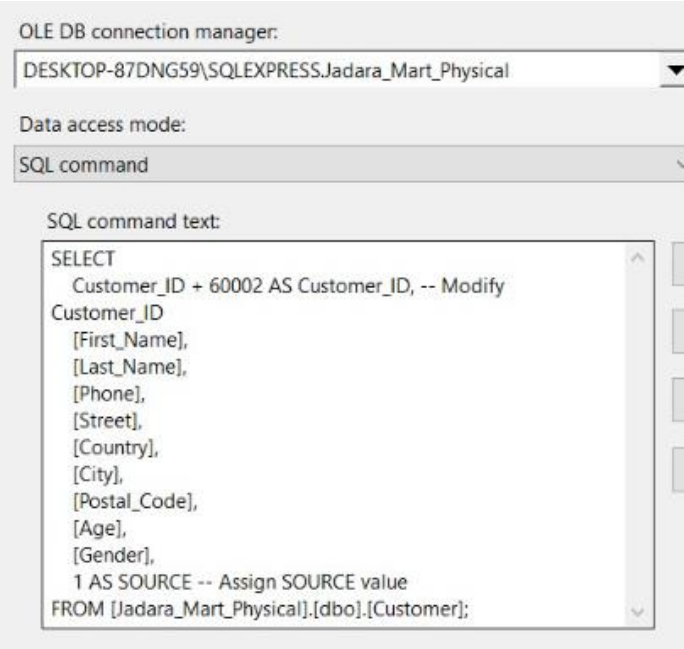
- **Date Update:** The package retrieves the current date and time at runtime to update the landingDate column in the destination table, reflecting the time of data loading.

2. Data Flow Task:

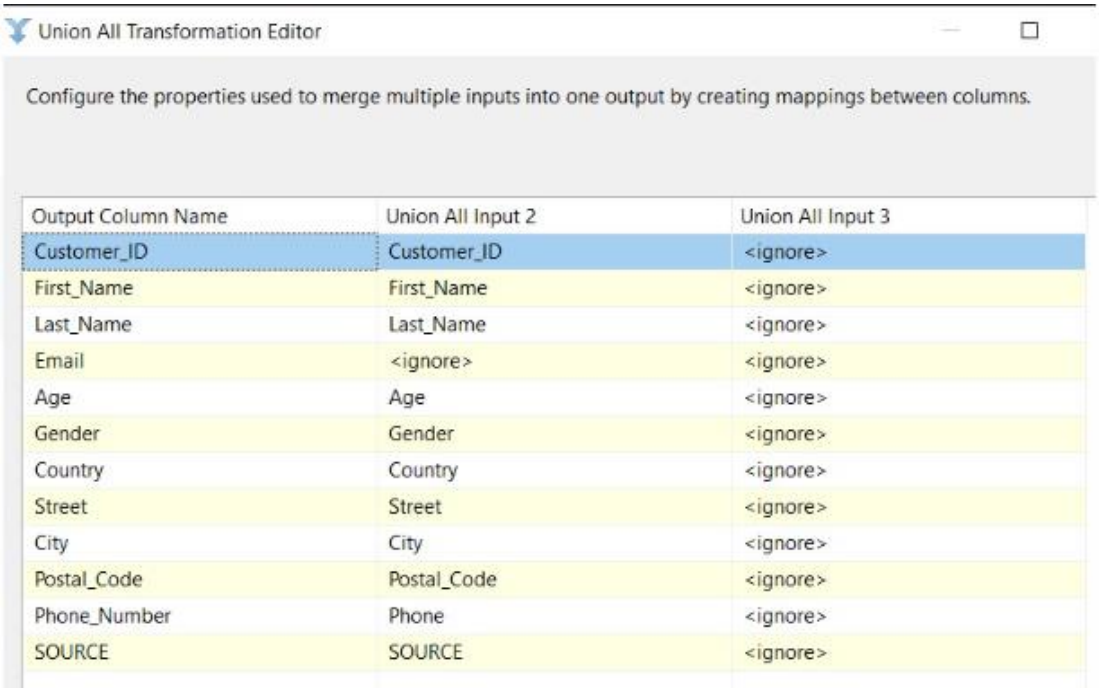


- **This task is composed of several components:**
    - **ADO Source and OLE DB Source:** These components extract data from the MySQL (online store) and SQL Server (physical store) databases, respectively. During extraction, a new column named "Source" is added to flag the data origin.





- **Union All:** Used to combine rows from columns that are common across both sources. *(Note: A Merge Join would be used if the columns were different.)*



Output Column Name	Union All Input 2	Union All Input 3
Customer_ID	Customer_ID	<ignore>
First_Name	First_Name	<ignore>
Last_Name	Last_Name	<ignore>
Email	<ignore>	<ignore>
Age	Age	<ignore>
Gender	Gender	<ignore>
Country	Country	<ignore>
Street	Street	<ignore>
City	City	<ignore>
Postal_Code	Postal_Code	<ignore>
Phone_Number	Phone	<ignore>
SOURCE	SOURCE	<ignore>

- **Script Component:** Generates a hash code for each row to identify changes and track data integrity.

Script Transformation Editor

Access Microsoft Visual Studio Tools for Applications (VSTA) to write scripts using Microsoft Visual Basic 2022 or Microsoft Visual C# 2022, and configure component properties.

- Script
- Input Columns
- Inputs and Outputs
- Connection Manager

Properties:

Common Properties	
ComponentClassID	{C1463F00-2FAF-4AD4-A212-C9D9CCB54575}
ContactInfo	Includes and runs custom script code. For exam
Description	Includes and runs custom script code. For exam
ID	139
IdentificationString	Script Component
IsDefaultLocale	True
LocaleID	English (United States)
Name	**Script Component**
PipelineVersion	0
UsesDispositions	False
ValidateExternalMetadata	True
Version	13
Custom Properties	
ReadOnlyVariables	
ReadWriteVariables	
ScriptLanguage	Microsoft Visual C# 2022
UserComponentTypeName	Microsoft.SqlServer.Dts.Pipeline.ScriptComponent

Name

Specifies the name of the component.

Edit Script...

OK Cancel Help

Script Transformation Editor

Access Microsoft Visual Studio Tools for Applications (VSTA) to write scripts using Microsoft Visual Basic 2022 or Microsoft Visual C# 2022, and configure component properties.

- Script
- Input Columns
- Inputs and Outputs
- Connection Manager

Input name: Input 0

Available Input Colu...

- Name
- Customer_ID
- First_Name
- Last_Name
- Email
- Age
- Gender
- Country
- Street
- City

Input Column	Output Alias	Usage Type
Customer_ID	Customer_ID	ReadWrite
First_Name	First_Name	ReadWrite
Last_Name	Last_Name	ReadWrite
Email	Email	ReadWrite
Age	Age	ReadWrite
Gender	Gender	ReadWrite
Country	Country	ReadWrite
Street	Street	ReadWrite

OK Cancel Help

```
SC_0d631bcf09a84af0913dc97dca3b7 ▾  ScriptMain ▾  columnNames ▾
using System;
using System.Security.Cryptography;
using System.Text;
using Microsoft.SqlServer.Dts.Pipeline.Wrapper;
using Microsoft.SqlServer.Dts.Runtime.Wrapper;

[Microsoft.SqlServer.Dts.Pipeline.SSISScriptComponentEntryPointAttribute]
0 references
public class ScriptMain : UserComponent
{
    private readonly string[] columnNames = {
        "Customer_ID", "First_Name", "Last_Name", "Phone", "Street", "Country", "City
    };

    2 references
    public override void Input0_ProcessInputRow(Input0Buffer Row)
    {
        StringBuilder stringBuilder = new StringBuilder();

        foreach (var columnName in columnNames)
        {
            string modifiedColumnName = columnName.Replace("_", "");
            var valueProperty = typeof(Input0Buffer).GetProperty(modifiedColumnName +
            if (valueProperty != null)
            {
                var valueIsNull = (bool)valueProperty.GetValue(Row, null);
                if (!valueIsNull)
                {
                    var value = typeof(Input0Buffer).GetProperty(modifiedColumnName).
                    stringBuilder.Append(value.ToString());
                }
                // If it's null, we simply ignore it and move to the next column
            }
        }

        string concatValue = stringBuilder.ToString();
```

The script is designed to generate a hash by concatenating multiple columns, ensuring proper handling of null values. For each column, if the value is not null, it is appended to a string. This concatenated string is then hashed using the **SHA-256** algorithm, with the resulting hash stored in the **Hashoutput** column.

Here’s a breakdown of the process:

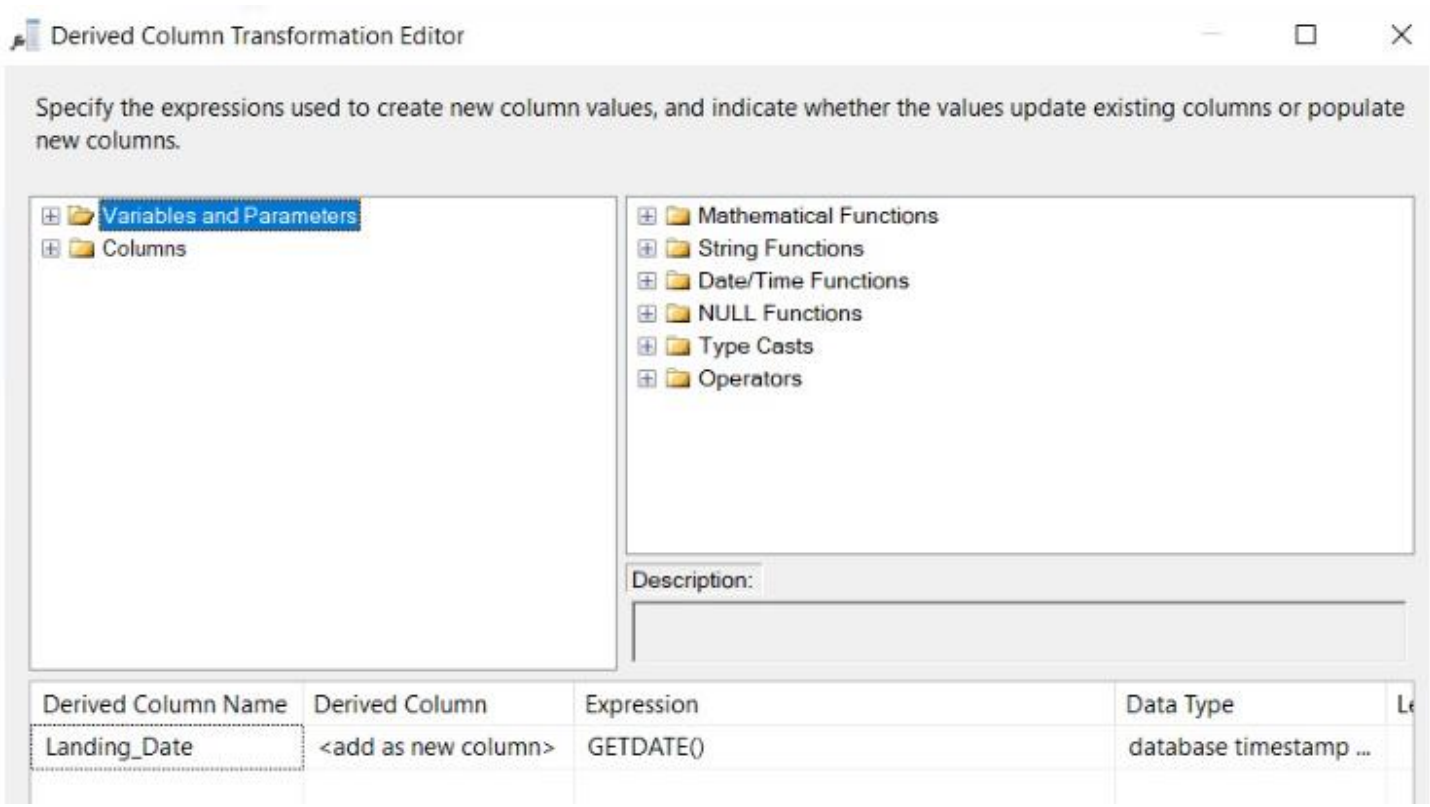
1. **Column Name Mapping**: The underscores are removed from column names when searching for the corresponding properties in the Input0Buffer (e.g., Customer_ID becomes CustomerID).

2. **Null Handling**: For each column, the script checks if the value is null by accessing the _IsNull property. If the column is not null, its value is appended to the string builder.

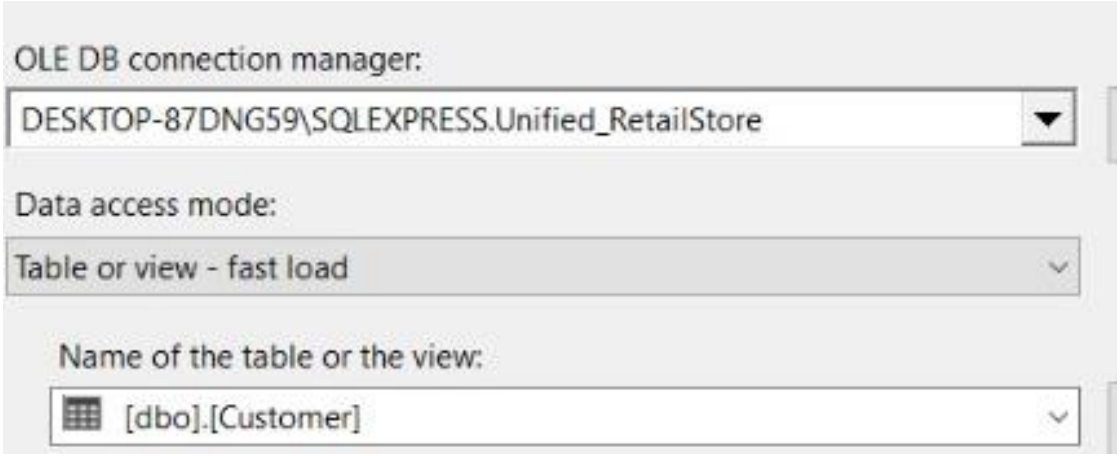
3. **Hashing**: Once all non-null values are concatenated, the SHA-256 algorithm is applied to the result, and the hash is stored in Hashoutput.

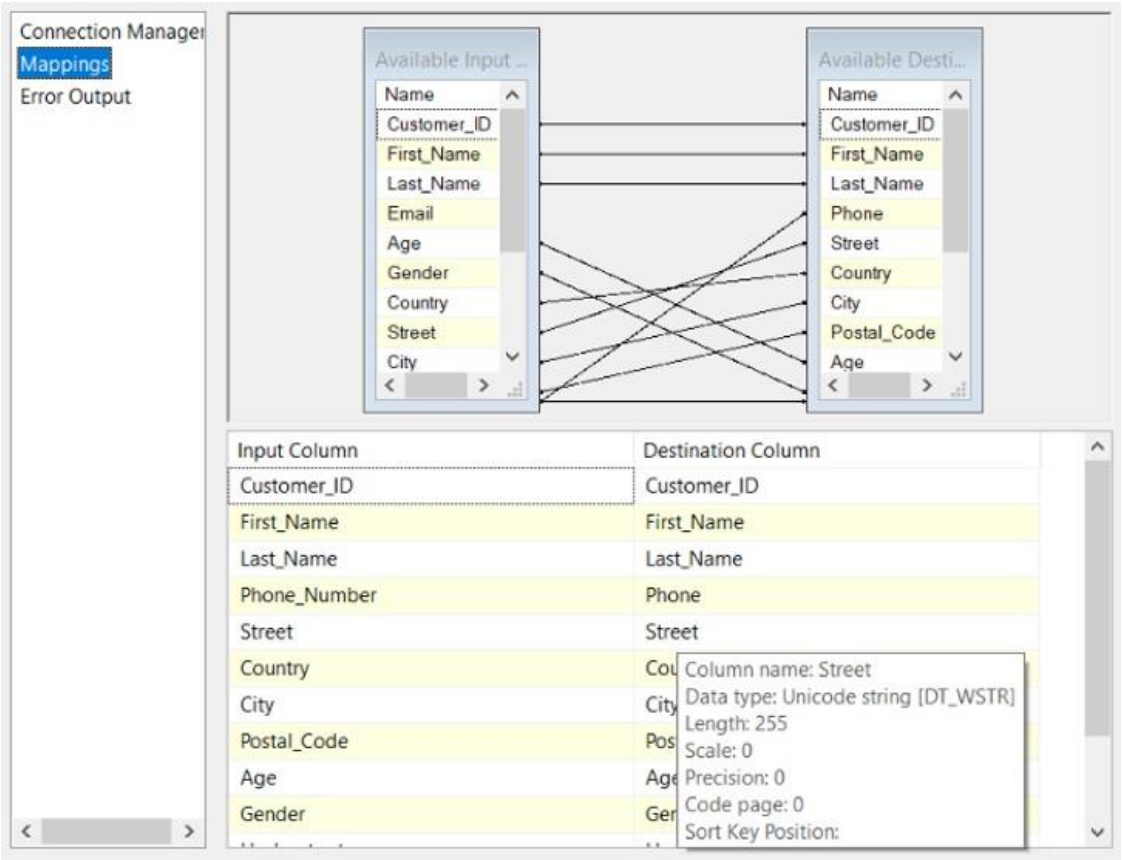
This approach ensures that null values do not disrupt the hashing process while efficiently generating a unique hash for non-null column combinations.

- **Derived Column:** Adds the landingDate column to the destination table.

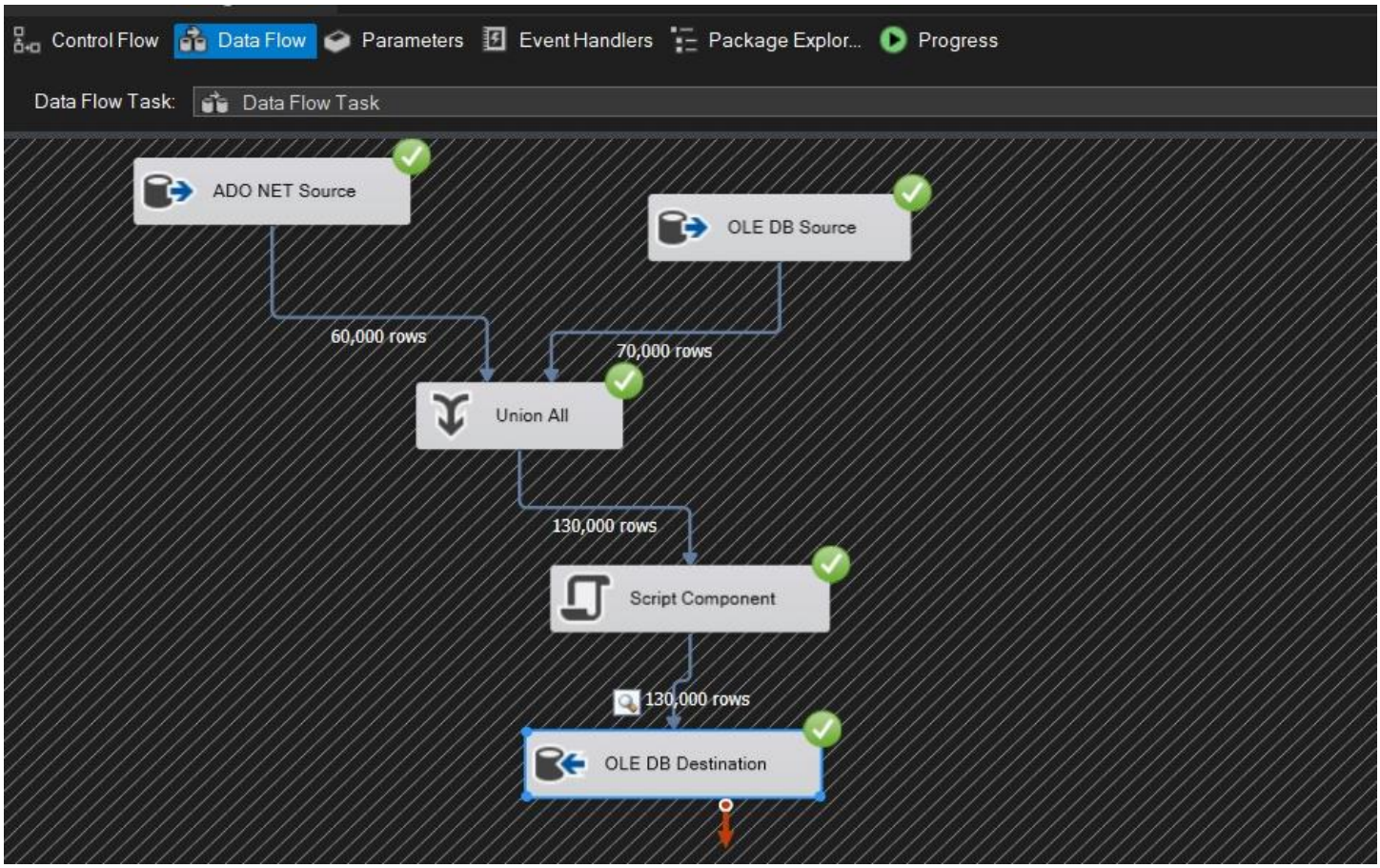


- **OLE DB Destination:** Utilizes "Fast Load" in data access mode to efficiently load data into the target table, with proper mapping between source and destination columns.





- **Package Running**

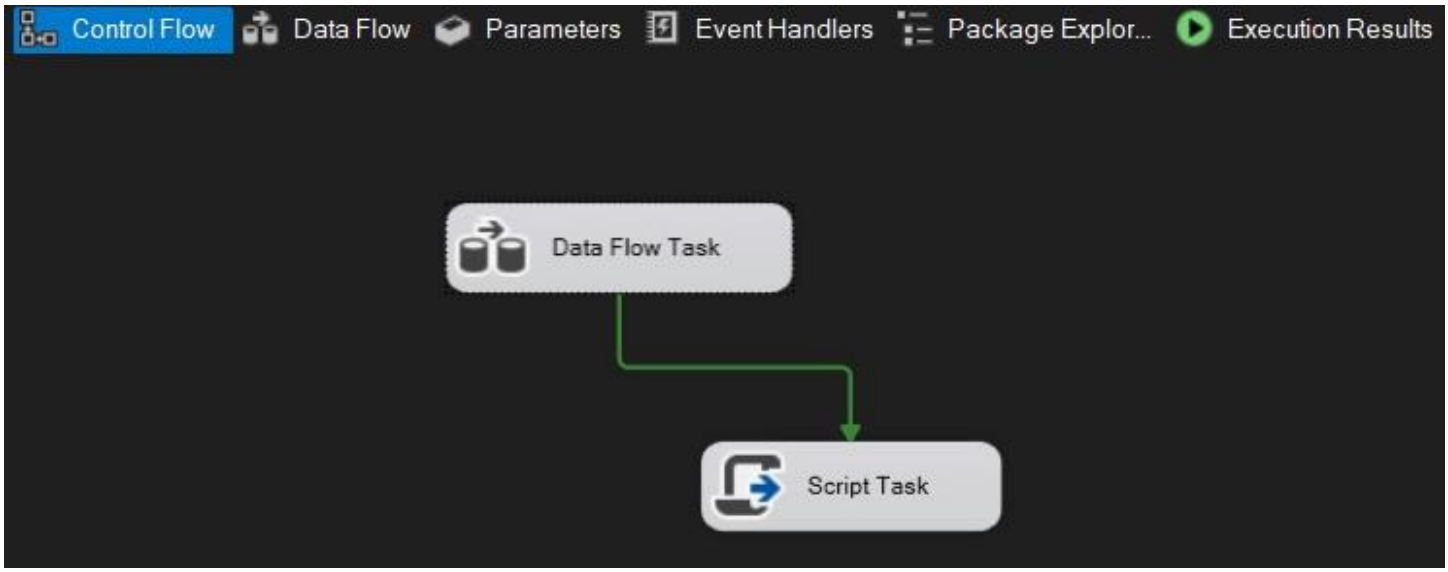


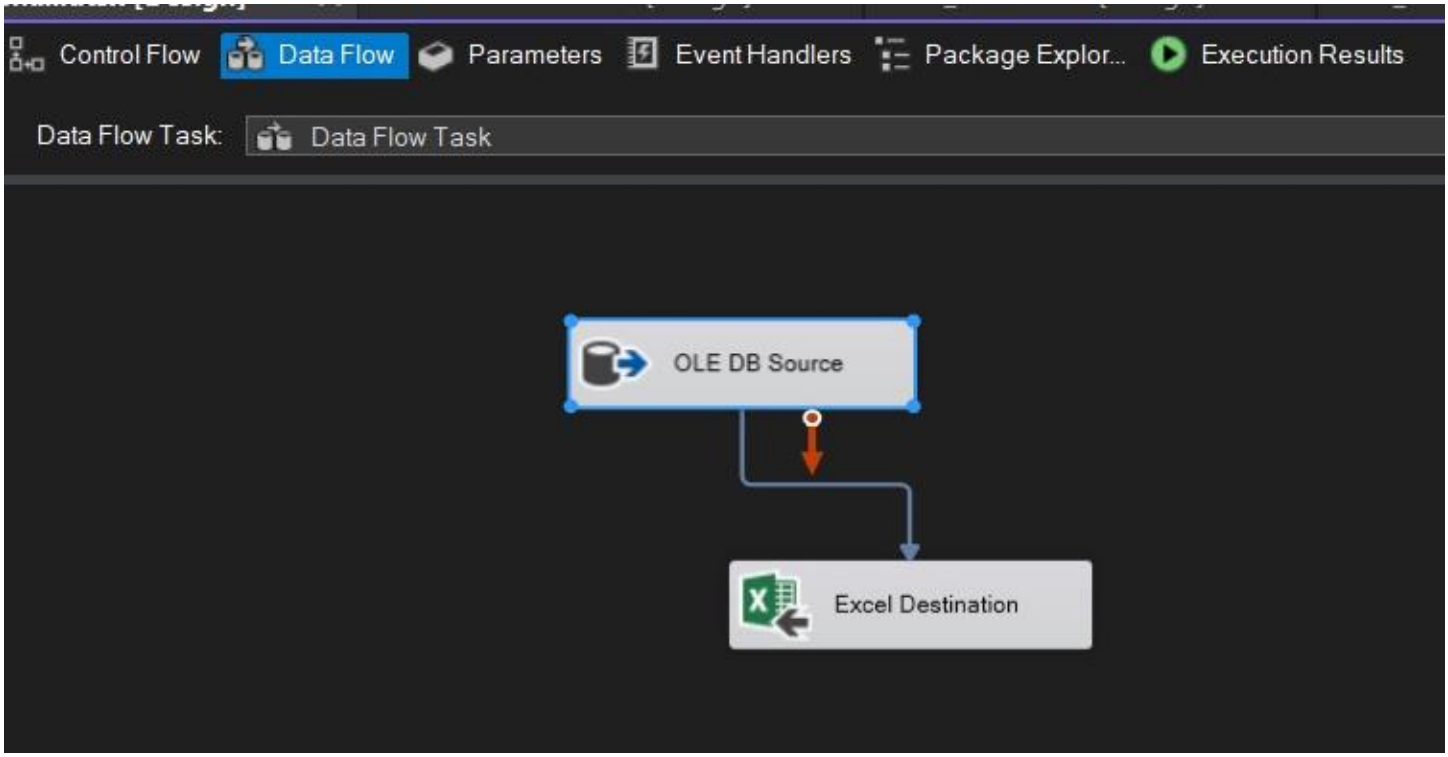
- **Connection Management:**
    - To facilitate smooth data transfer, we configured global connection managers for the entire project, ensuring seamless connectivity between the sources and the destination.
- **Consistency Across Packages:**
    - We maintained a consistent approach across all ETL packages, covering every table in the schema.

Automated Email Notification Package

We created a unique package to automate email notifications, which includes:

- **Data Flow Task:** Extracts metadata such as step name, run date, job name, and duration from the Integration Services Catalog database and outputs it to an Excel file.
- **Script Task:** Customizes the content of the email to provide a summary of the ETL execution status.





OLE DB Source Editor

Configure the properties used by a data flow to obtain data from any OLE DB provider.

- Connection Manager
- Columns
- Error Output

Specify an OLE DB connection manager, a data source, or a data source view, and select the data access mode. If using the SQL command access mode, specify the SQL command either by typing the query or by using Query Builder.

OLE DB connection manager:

localhost\Tamer.SSISDB

New...

Data access mode:

SQL command

SQL command text:

```
select
 j.name as 'JobName',
 s.step_id as 'Step',
 s.step_name as 'StepName',
 msdb.dbo.agent_datetime(run_date, run_time) as
'RunDateTime',
 ((run_duration/10000*3600 + (run_duration/100)%100*60 +
run_duration%100 + 31 ) / 60)
       as 'RunDurationMinutes'
From msdb.dbo.sysjobs j
INNER JOIN msdb.dbo.sysjobsteps s
 ON j.job_id = s.job_id
INNER JOIN msdb.dbo.sysjobhistory h
 ON s.job_id = h.job_id
```

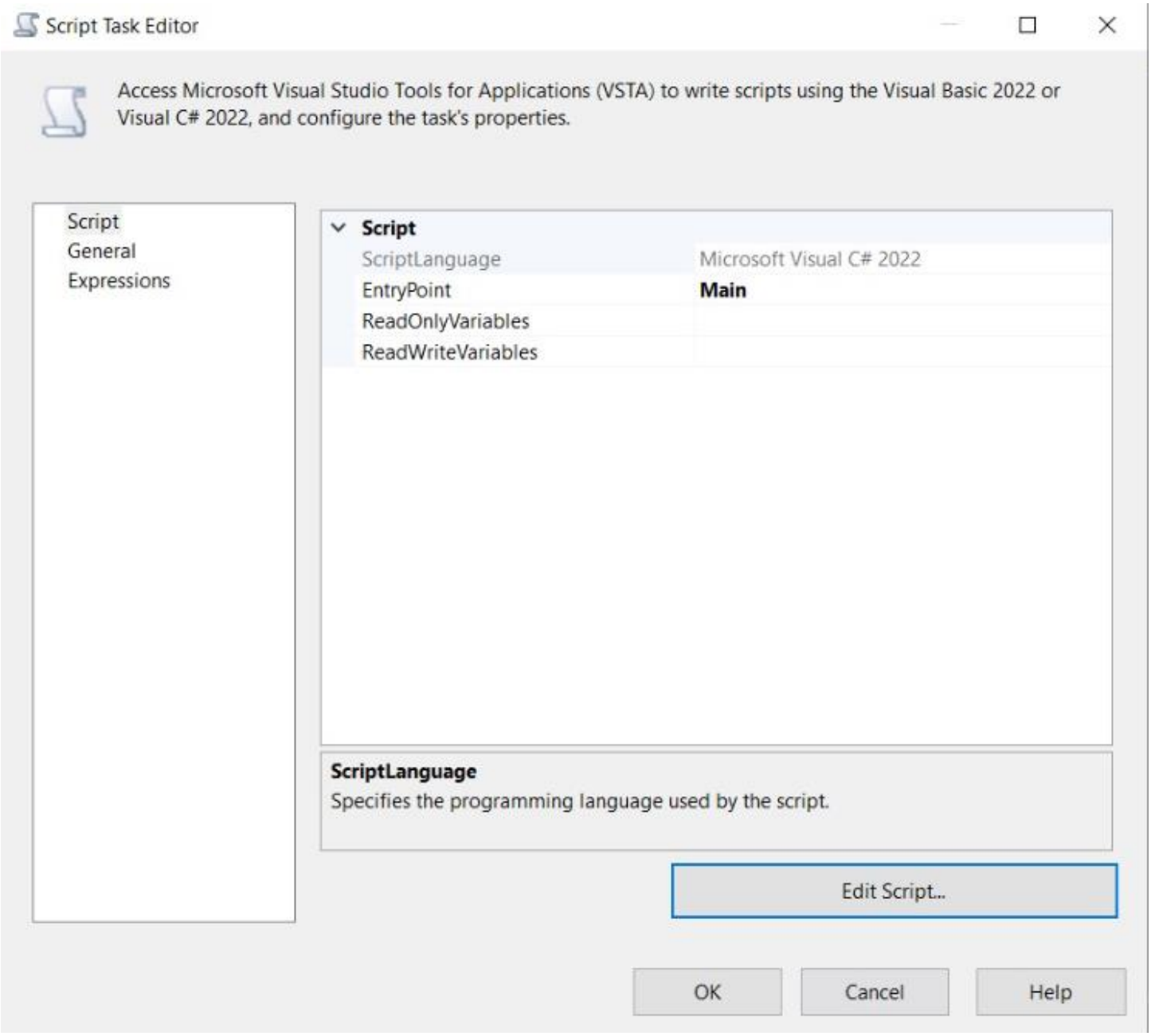
Parameters...

Build Query...

Browse...

Parse Query

Preview...



This script is used for sending email notifications from an SSIS (SQL Server Integration Services) package. It includes the ability to send emails with an attachment and handles error logging if something goes wrong. Let's break it down in detail:

1) Namespaces

The script starts with several important namespaces:

```
using System;
using System.Net.Mail;
using System.Net;
using Microsoft.SqlServer.Dts.Runtime;
using System.IO;
```

- **System.Net.Mail**: Provides classes to construct and send email messages.
- **System.Net**: Used for handling network credentials and sending email securely over SMTP.
- **Microsoft.SqlServer.Dts.Runtime**: Provides access to the SSIS runtime and error logging through the Dts object.
- **System.IO**: Used to check the existence of the file attachment before adding it to the email.

2) Mail Settings

In the Main() method, the script constructs the email:

```
string fromAddress = "KingHammad404@outlook.com";
string appPassword = "puoualsobvhnrvav"; // Replace with your App Password
string toAddress = "7ammad404@gmail.com";
string subject = "Success Notification";
string body = "Yay! The job was successful!";
string attachmentPath = @"D:\Book2.xlsx";
```

1. **fromAddress**: The sender's email address.
2. **appPassword**: The **app-specific password** (important when using email providers like Outlook that require this for secure access, especially when multi-factor authentication is enabled).

3. **toAddress**: The recipient's email address.

4. **subject** and **body**: These define the content of the email.

5. **attachmentPath**: Path to the file that will be attached to the email.

3) MailMessage Construction

Here, the MailMessage object is created, which defines the email:

```
MailMessage mail = new MailMessage(fromAddress, toAddress, subject, body);
mail.IsBodyHtml = false;
```

- MailMessage takes the sender, recipient, subject, and body as parameters.
- IsBodyHtml = false ensures that the email body is treated as plain text (not HTML). If you want to send an HTML-formatted email, you can set this to true.

4) Attachment Handling

The script checks if the file exists before attaching it to the email:

```
if (File.Exists(attachmentPath))
{
    Attachment attachment = new Attachment(attachmentPath);
    mail.Attachments.Add(attachment);
}
else
{
    throw new FileNotFoundException("The specified attachment file was not found.", attachm
}
```

- File.Exists(attachmentPath) ensures that the file exists before attempting to attach it. If it doesn't exist, a FileNotFoundException is thrown, which is caught later in the catch block.
- The attachment is added to the MailMessage via mail.Attachments.Add(attachment).

5) SMTP Client Setup

The SmtpClient is used to send the email:

```
SmtpClient smtpClient = new SmtpClient("smtp.office365.com", 587);
smtpClient.EnableSsl = true;
smtpClient.UseDefaultCredentials = false;
smtpClient.Credentials = new NetworkCredential(fromAddress, appPassword);
smtpClient.DeliveryMethod = SmtpDeliveryMethod.Network;
smtpClient.Timeout = 30000; // 30 seconds
ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
smtpClient.Send(mail);
```

1. **"smtp.office365.com"** and port **587** are used for sending email via Outlook/Office 365.
2. **EnableSsl = true** ensures that SSL/TLS encryption is used for secure communication.
3. **UseDefaultCredentials = false** disables using the machine's default credentials for authentication.
4. **smtpClient.Credentials** sets the sender's email credentials (email address and app password).
5. **smtpClient.Timeout**: The operation will time out after 30 seconds if the email isn't sent.
6. **ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12**: Ensures the script uses TLS 1.2, which is required for modern email servers to encrypt email traffic.
7. **smtpClient.Send(mail)**: Actually sends the email.

6) Handling Success and Failure

The script handles success or failure by setting the SSIS task result:

```
Dts.TaskResult = (int)ScriptResults.Success;
```

- If everything goes well, the SSIS task result is set to Success. This indicates that the script executed correctly.

7) Error Handling

In case something goes wrong, the catch block captures exceptions and logs detailed error messages:

```
catch (Exception ex)
{
    string errorMessage = $"Error sending email: {ex.Message}";
    if (ex.InnerException != null)
    {
        errorMessage += $" Inner exception: {ex.InnerException.Message}";
    }
    Dts.Events.FireError(0, "Email Script Task", errorMessage, String.Empty, 0);
    Dts.TaskResult = (int)ScriptResults.Failure;
}
```

- The error message includes the exception's Message and, if applicable, the InnerException message to give more context about the failure.
- Dts.Events.FireError logs the error within the SSIS framework, so it can be viewed in the package's execution results.
- The task result is set to Failure if an error occurs.

8) ScriptResults Enum

The ScriptResults enum is used to clearly map success and failure states for SSIS:

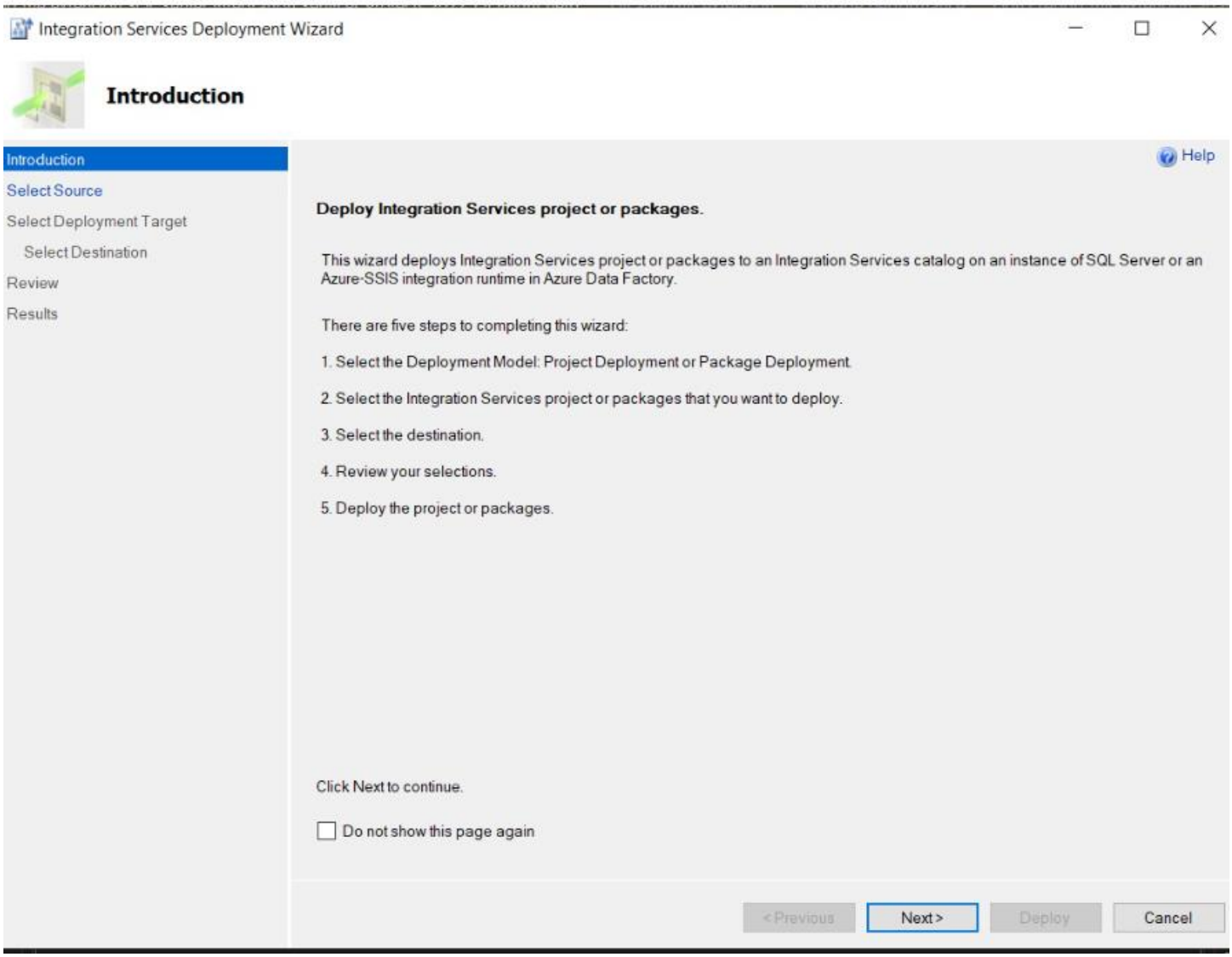
```
enum ScriptResults
{
    Success = Microsoft.SqlServer.Dts.Runtime.DTSExecResult.Success,
    Failure = Microsoft.SqlServer.Dts.Runtime.DTSExecResult.Failure
}
```

- This enum makes the code more readable and ensures that task results are clearly mapped to the corresponding SSIS result types (Success and Failure).

This script effectively handles sending emails with attachments and provides good error reporting, making it suitable for use in an SSIS package for notifications, such as reporting job successes or failures.

Packages Deployment and Job Creation

To automate the notification process, we deployed the ETL packages to the SQL Server Integration Services (SSIS) catalog and configured a SQL Server Agent job for the landing database. This job consists of a step for each ETL package, with the final step designed to send an email notification upon the successful completion of all ETL tasks, confirming the successful transfer of data to the landing database.



Integration Services Deployment Wizard

Select Destination

- Introduction
- Select Source
- Select Deployment Target
- Select Destination
- Review
- Results

Help

Enter the destination server name and where the project will be located in the Integration Services catalog.

Server name: localhost\tamer [Browse...]

Authentication: Windows Authentication [Connect]

User name:

Password:

Path: /SSISDB/JadraMart/Integration Services Project1 [Browse...]

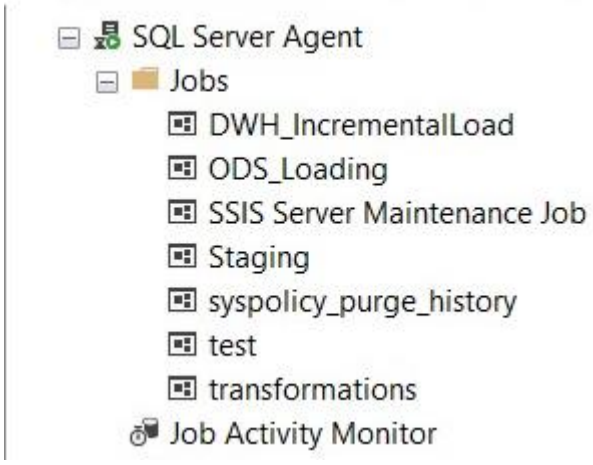
A project with this name already exists. Click Next to replace the existing project during deployment.

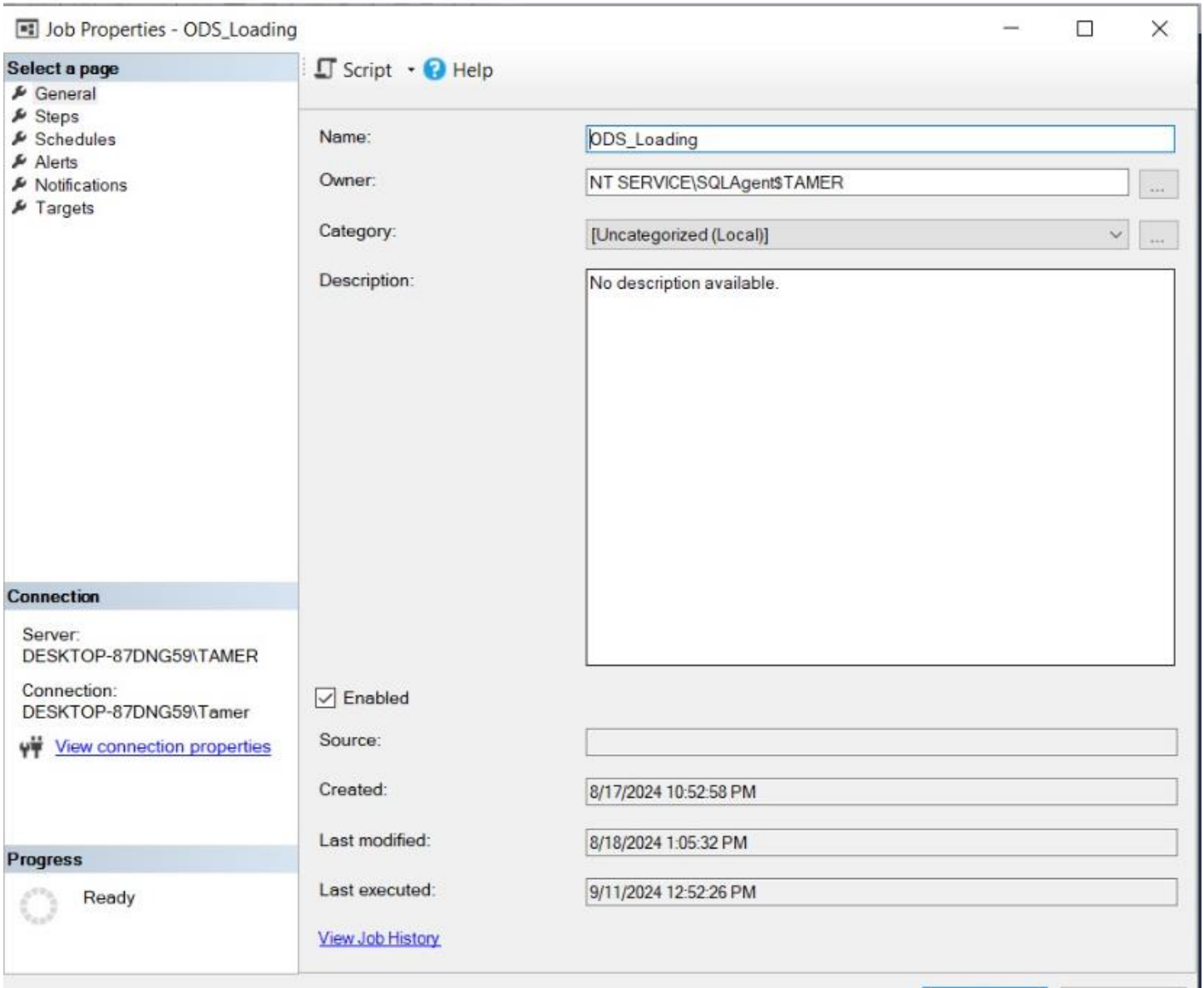
< Previous | Next > | Deploy | Cancel

- Integration Services Catalogs
  - SSISDB
    - JadraMart
      - Projects
        - DWH
        - Integration Services Project1
        - staging
        - Transformations
      - Environments

- Integration Services Catalogs
  - SSISDB
    - JadraMart
      - Projects
        - DWH
        - Integration Services Project1
          - Packages
            - Branch.dtsx
            - branch_vendor.dtsx
            - Cart.dtsx
            - Cart_Product.dtsx
            - Category.dtsx
            - CustomerLanding.dtsx
            - employees.dtsx
            - inventory.dtsx
            - Inventory_Product.dtsx
            - mail.dtsx
            - Order - Copy.dtsx
            - order.dtsx
            - Order_Product 1.dtsx
            - payment.dtsx
            - pays_check.dtsx
            - Product.dtsx
            - Product_vendor.dtsx
            - shipment.dtsx
            - Subcategory.dtsx
            - template 1.dtsx
            - testlookup.dtsx
            - vendor.dtsx
        - staging
        - Transformations
      - Environments

Job Creation





Job Properties - ODS_Loading

Select a page

- General
- Steps
- Schedules
- Alerts
- Notifications
- Targets

Script · Help

Job step list:

S...	Name	Type	On Succ...	On Failure
1	CustomerLanding	SQL Se...	Go to th...	Quit the...
2	Branch	SQL Se...	Go to th...	Go to th...
3	Branch__Vendor	SQL Se...	Go to th...	Quit the...
4	Cart	SQL Se...	Go to th...	Quit the...
5	Cart_Products	SQL Se...	Go to th...	Quit the...
6	Category	SQL Se...	Go to th...	Quit the...
7	Employees	SQL Se...	Go to th...	Quit the...
8	Inventory	SQL Se...	Go to th...	Quit the...
9	Inventory_Products	SQL Se...	Go to th...	Quit the...
10	Orders	SQL Se...	Go to th...	Quit the...
11	Orders_Products	SQL Se...	Go to th...	Quit the...
12	Payment	SQL Se...	Go to th...	Quit the...
13	paysCheck	SQL Se...	Go to th...	Quit the...
14	Products	SQL Se...	Go to th...	Quit the...
15	Products_vendor	SQL Se...	Go to th...	Quit the...
16	shipment	SQL Se...	Go to th...	Quit the...
17	Subcategory	SQL Se...	Go to th...	Quit the...
18	Vendor	SQL Se...	Go to th...	Quit the...
19	mail	SQL Se...	Quit the...	Quit the...

Connection

Server:
DESKTOP-87DNG59\TAMER

Connection:
DESKTOP-87DNG59\Tamer

View connection properties

Progress

Ready

Move step:

Start step:
1:CustomerLanding

New... | Insert... | Edit | Delete

OK | Cancel

Job Step Properties - CustomerLanding

Select a page
- General
- Advanced

Script · Help

Step name:
CustomerLanding

Type:
SQL Server Integration Services Package

Run as:
SQL Server Agent Service Account

Package | Configuration

Package source: SSIS Catalog

Server: localhost\Tamer

Log on to the server
- (●) Use Windows Authentication
- ( ) Use SQL Server Authentication

User name:

Password:

Package:
\SSISDB\JadraMart\Integration Services Project1\CustomerLanding.dtsx ...

Connection

Server:
DESKTOP-87DNG59\TAMER

Connection:
DESKTOP-87DNG59\Tamer

View connection properties

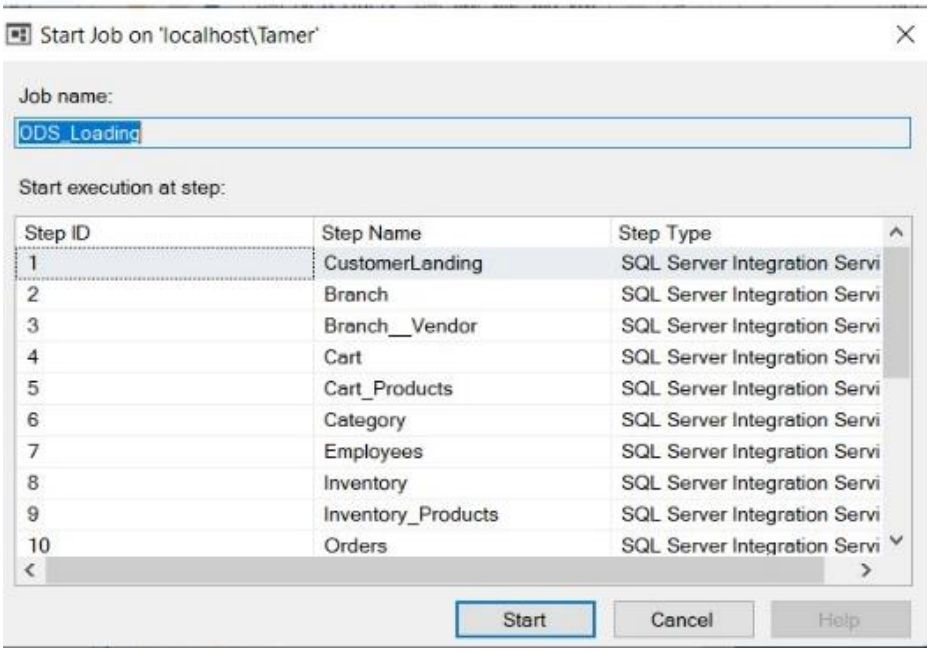
Progress

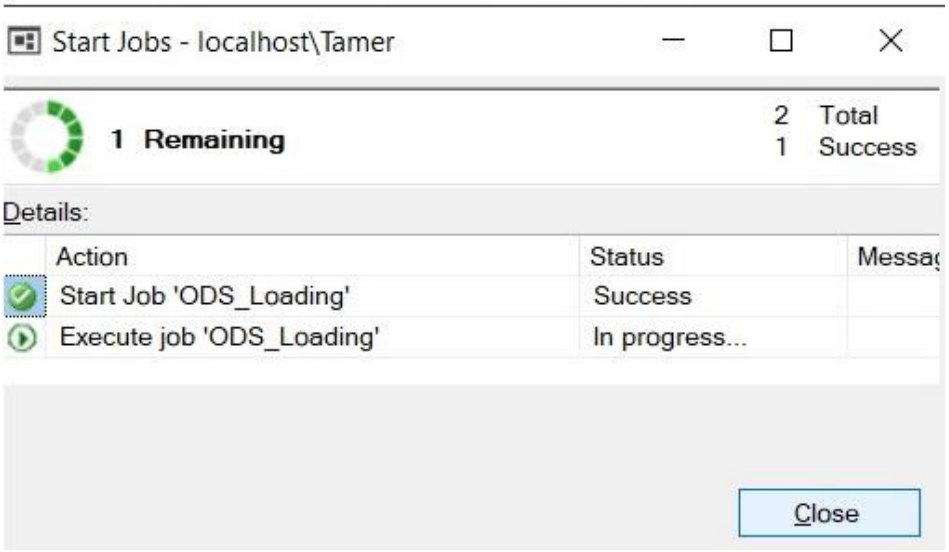
Ready

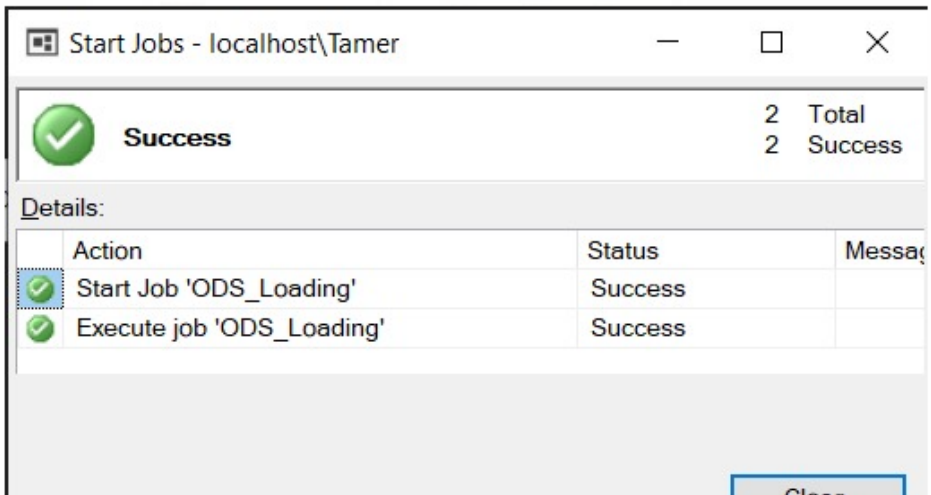
Previous | Next

OK | Cancel

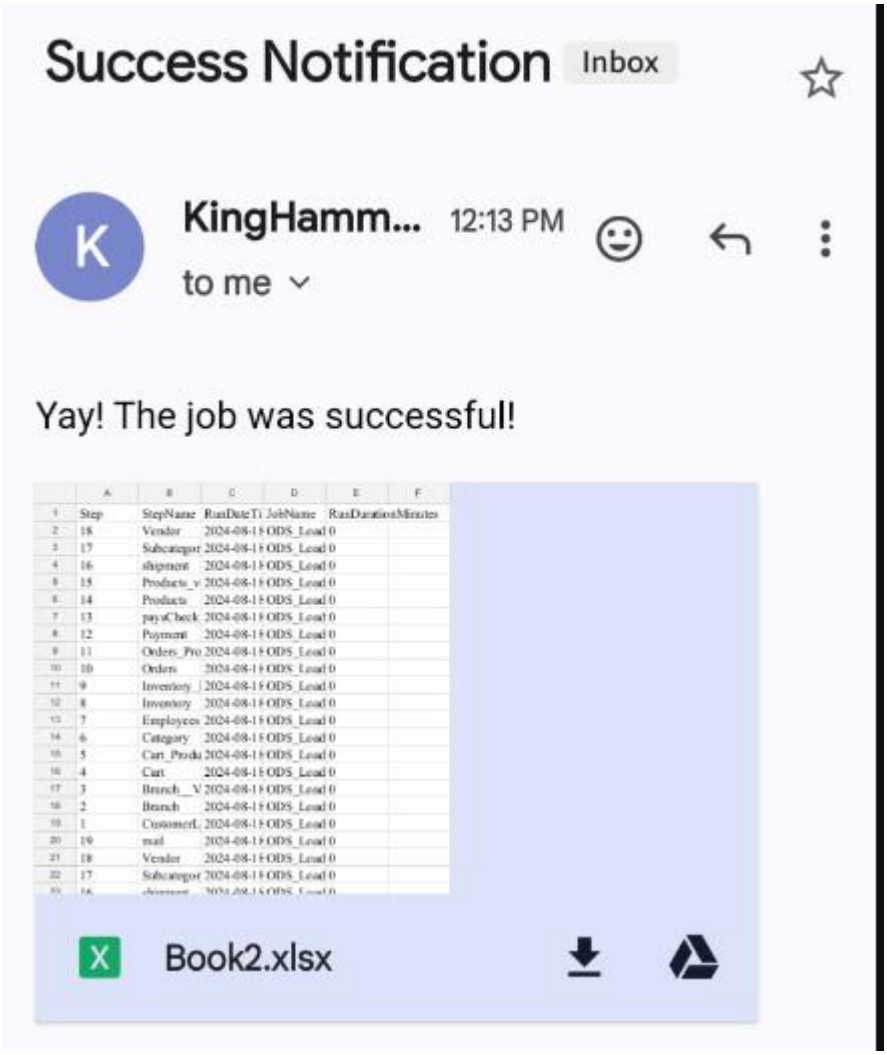
Job Start:


Start Job on 'localhost\Tamer'
Job name:
ODS_Loading
Start execution at step:
Step ID
Step Name
Step Type
1 CustomerLanding SQL Server Integration Servi
2 Branch SQL Server Integration Servi
3 Branch__Vendor SQL Server Integration Servi
4 Cart SQL Server Integration Servi
5 Cart_Products SQL Server Integration Servi
6 Category SQL Server Integration Servi
7 Employees SQL Server Integration Servi
8 Inventory SQL Server Integration Servi
9 Inventory_Products SQL Server Integration Servi
10 Orders SQL Server Integration Servi
Start
Cancel
Help


Start Jobs - localhost\Tamer
1 Remaining
2 Total
1 Success
Details:
Action
Status
Start Job 'ODS_Loading'
Success
Execute job 'ODS_Loading'
In progress...
Close


Start Jobs - localhost\Tamer
Success
2 Total
2 Success
Details:
Action
Status
Start Job 'ODS_Loading'
Success
Execute job 'ODS_Loading'
Success

Email after running the package:



Sent File Content:

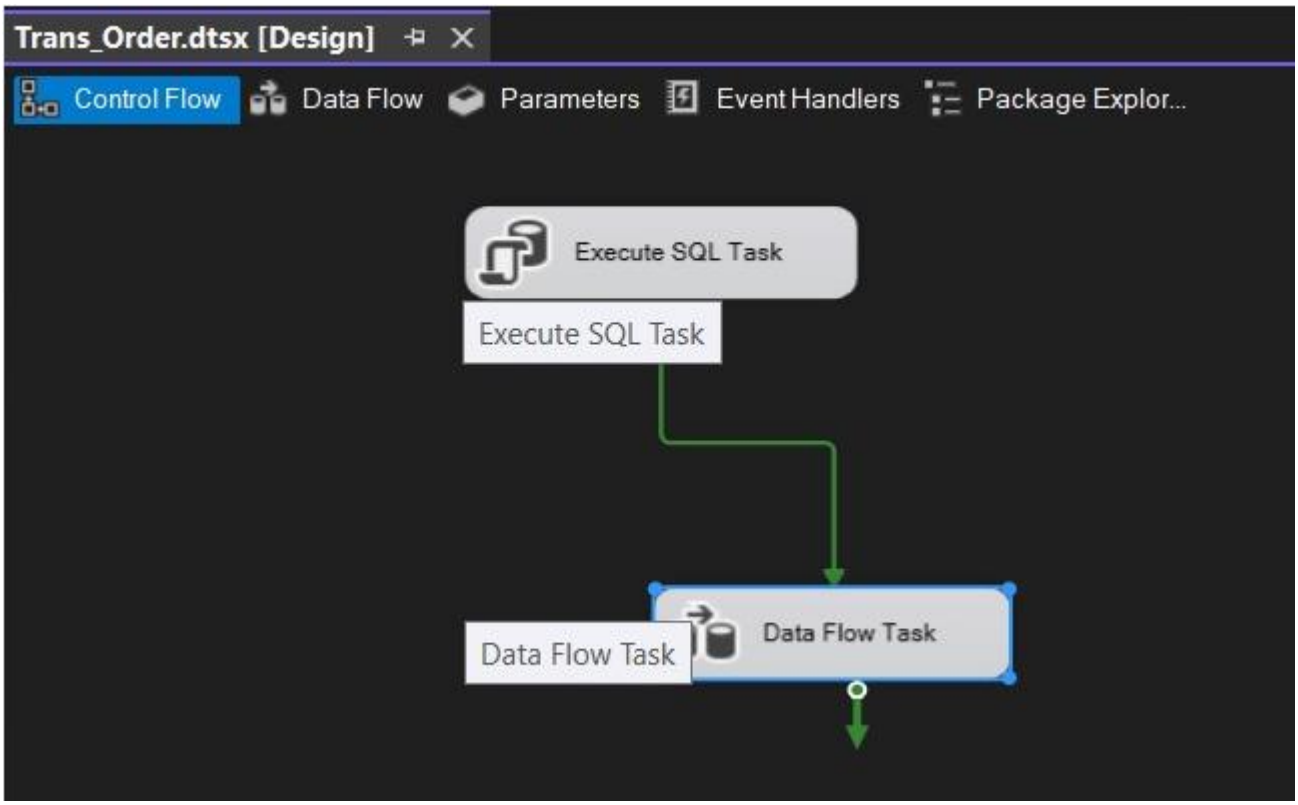
I8

	A	B	C	D	E	F
1	Step	StepName	RunDateTime	JobName	RunDurationMinutes	
2	18	Vendor	2024-08-18 13:23:15	ODS_Loading	0	
3	17	Subcategory	2024-08-18 13:23:09	ODS_Loading	0	
4	16	shipment	2024-08-18 13:23:01	ODS_Loading	0	
5	15	Products_vendor	2024-08-18 13:22:53	ODS_Loading	0	
6	14	Products	2024-08-18 13:22:48	ODS_Loading	0	
7	13	paysCheck	2024-08-18 13:22:41	ODS_Loading	0	
8	12	Payment	2024-08-18 13:22:35	ODS_Loading	0	
9	11	Orders_Products	2024-08-18 13:22:18	ODS_Loading	0	
10	10	Orders	2024-08-18 13:22:09	ODS_Loading	0	
11	9	Inventory_Products	2024-08-18 13:22:05	ODS_Loading	0	
12	8	Inventory	2024-08-18 13:21:59	ODS_Loading	0	
13	7	Employees	2024-08-18 13:21:54	ODS_Loading	0	
14	6	Category	2024-08-18 13:21:48	ODS_Loading	0	
15	5	Cart_Products	2024-08-18 13:21:41	ODS_Loading	0	
16	4	Cart	2024-08-18 13:21:35	ODS_Loading	0	
17	3	Branch__Vendor	2024-08-18 13:21:29	ODS_Loading	0	

ETL Process for Data Transformation

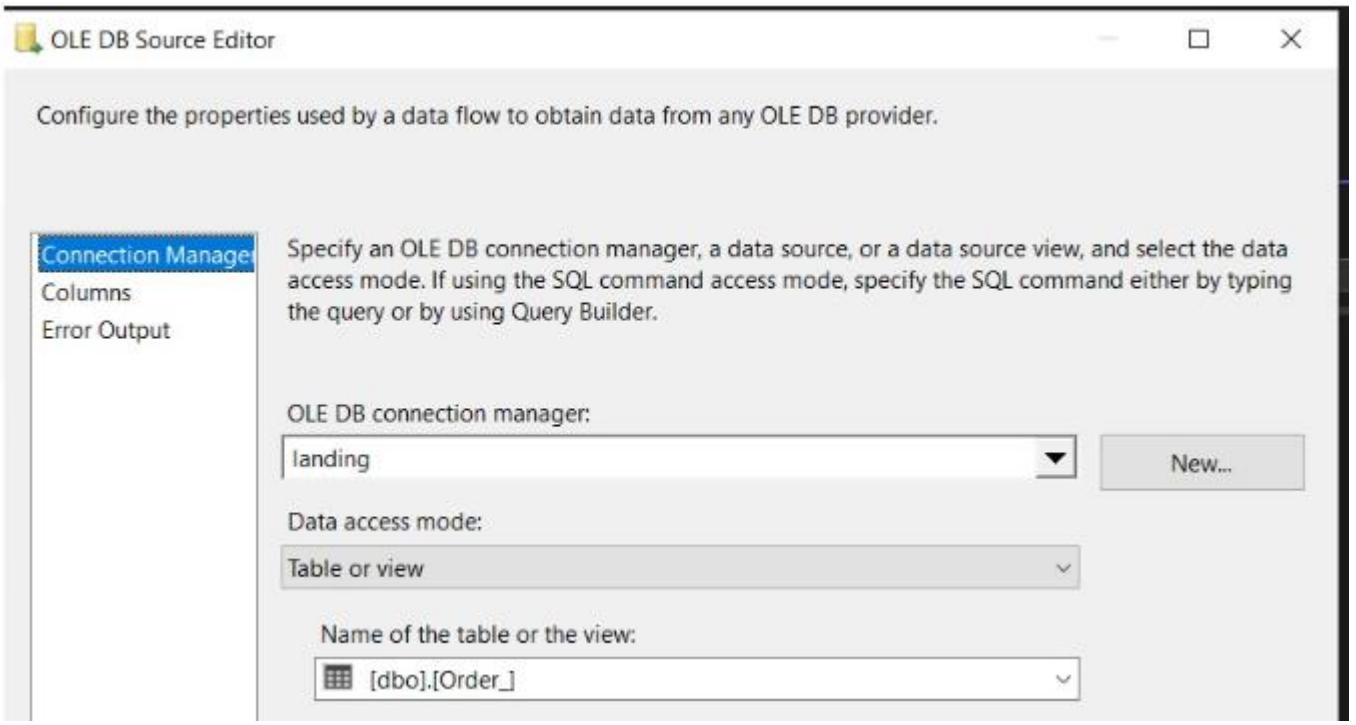
To manage data transformations effectively, we developed a separate project dedicated specifically to this purpose. This approach enables us to isolate and independently execute the transformation logic when needed, ensuring flexibility and modularity in our data pipeline.

ETL Package Design for Data Transformation



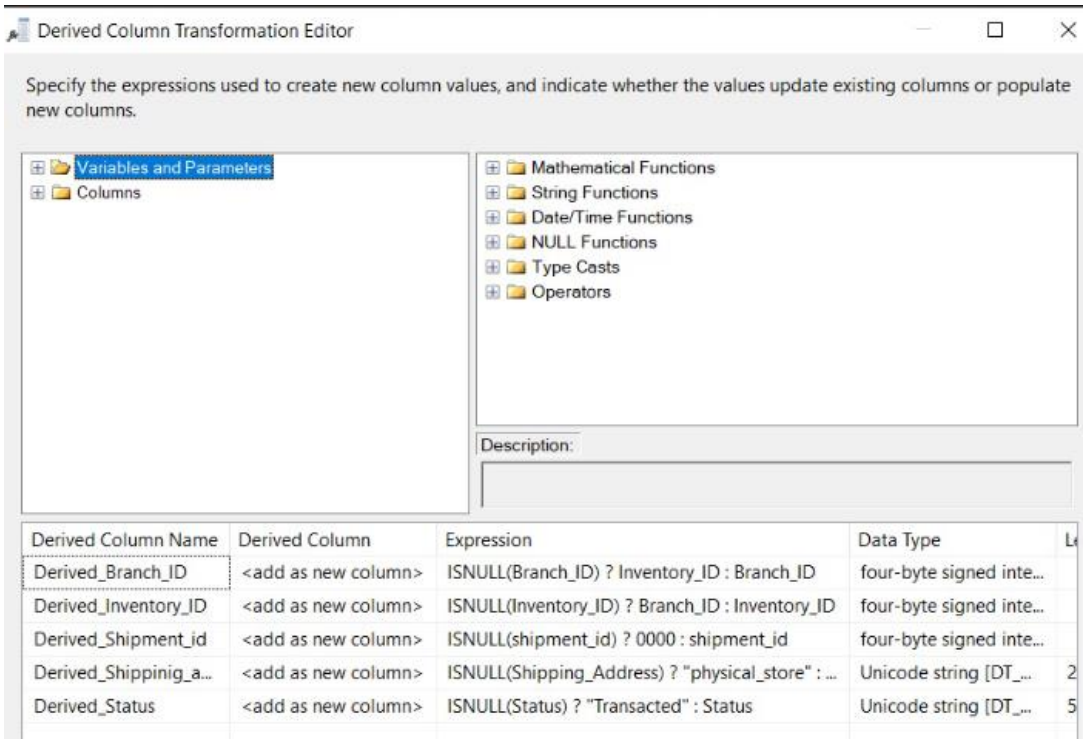
The transformation package structured as follows:

1. **OLE DB Source:**
    - This component extracts data from the relevant table in the landing database, serving as the source for the transformation process.

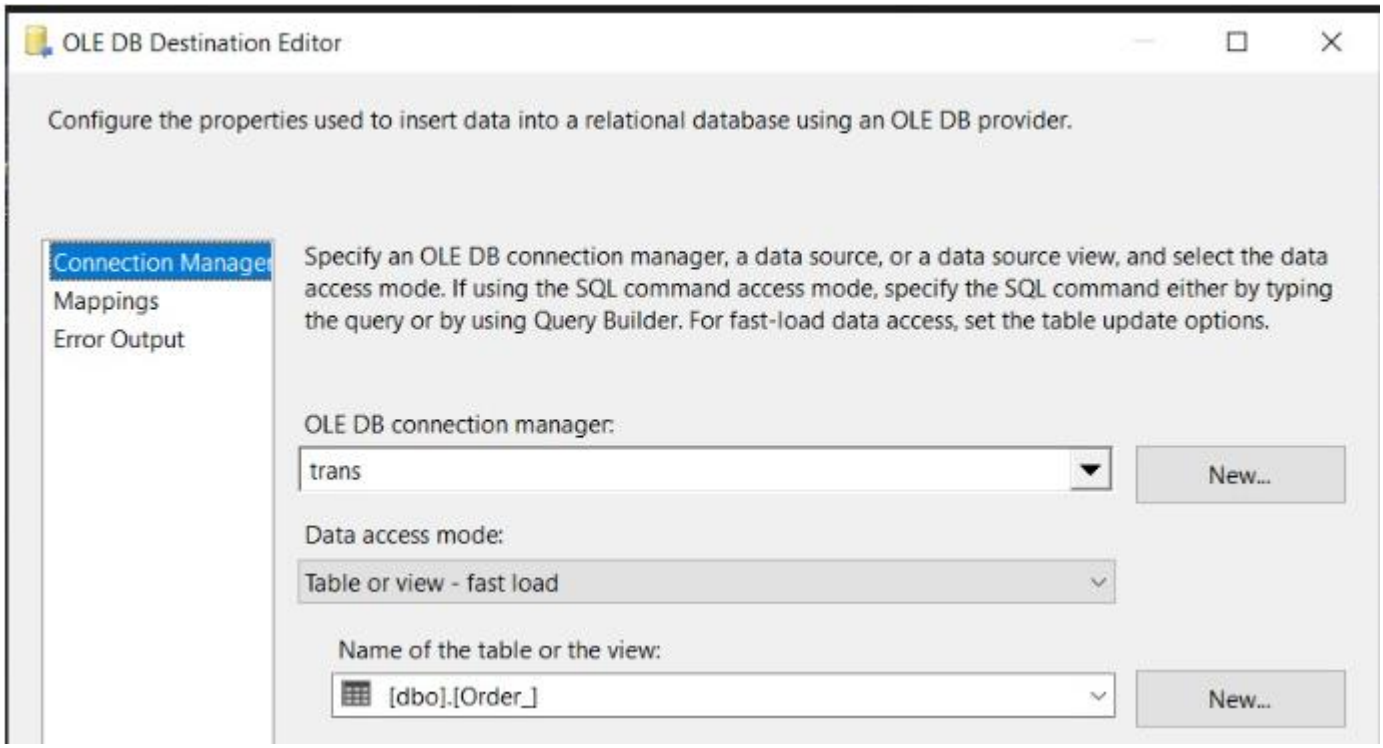


2. **Derived Column Transformation:**

   - A derived column transformation is utilized to handle null values in the data. This step allows us to define new column expressions, replacing any null values with default or computed values as required by the business logic.

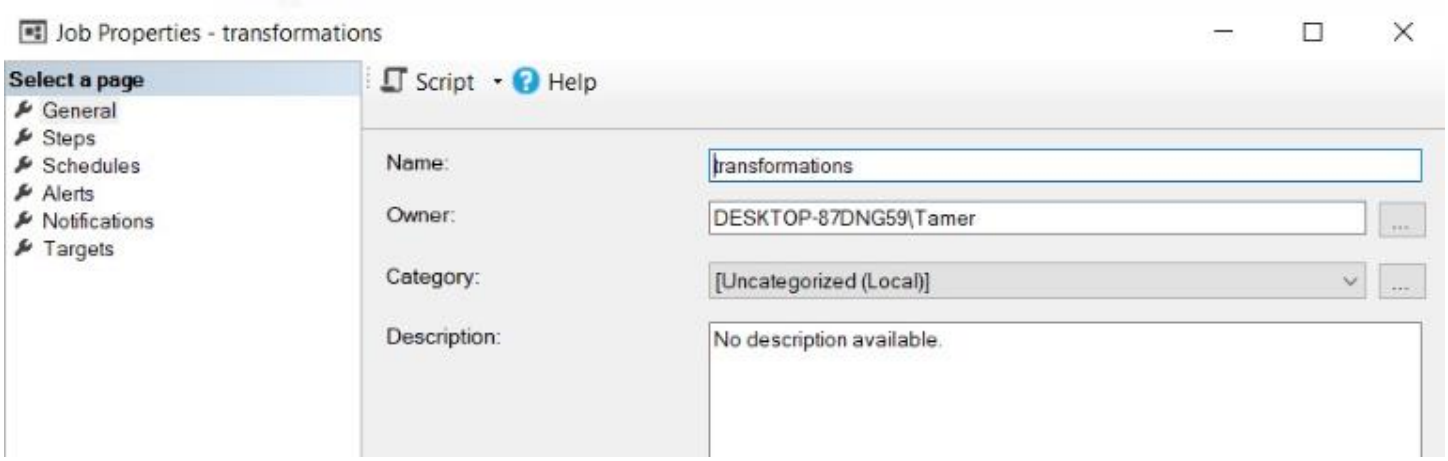


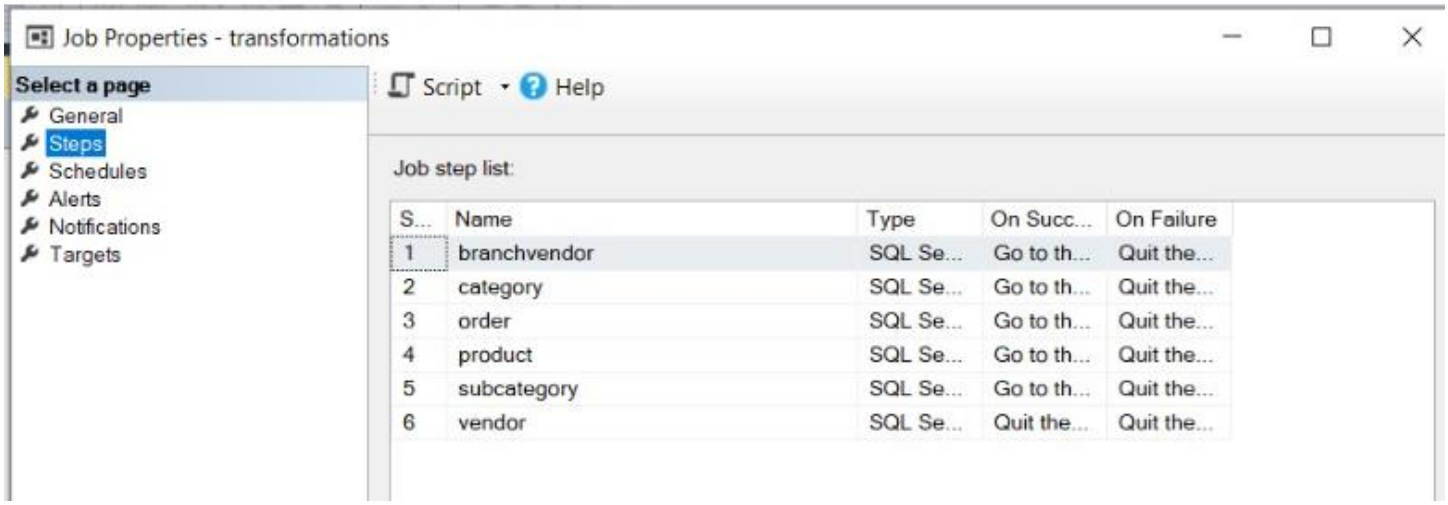
3. **OLE DB Destination:**
    - After processing, the data is loaded into the destination, with careful mapping to ensure that the transformed columns replace the original columns containing null values.

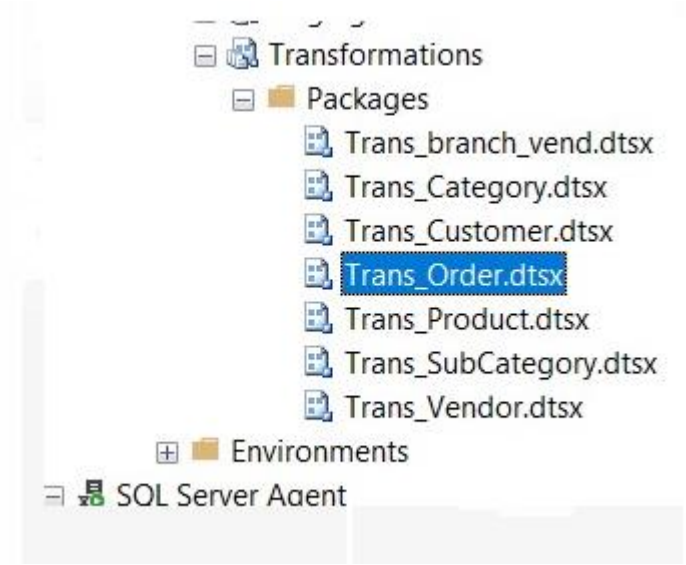


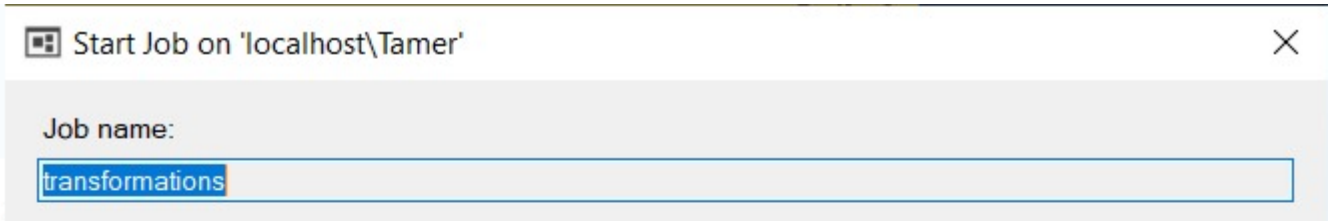
Automation Transformation ETL Processe

To automate the ETL process for the transformation databases, we configured a SQL Server Agent job. This job includes steps for executing the ETL packages, similar to the automation implemented for the landing database.

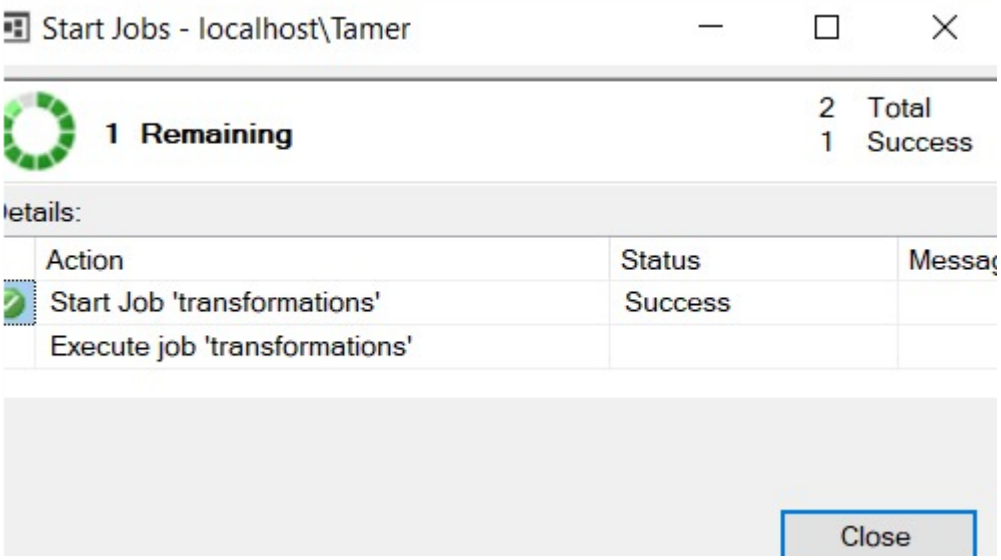

Job Properties - transformations
Select a page
General
Steps
Schedules
Alerts
Notifications
Targets
Script
Help
Name:
transformations
Owner:
DESKTOP-87DNG59\Tamer
Category:
[Uncategorized (Local)]
Description:
No description available.

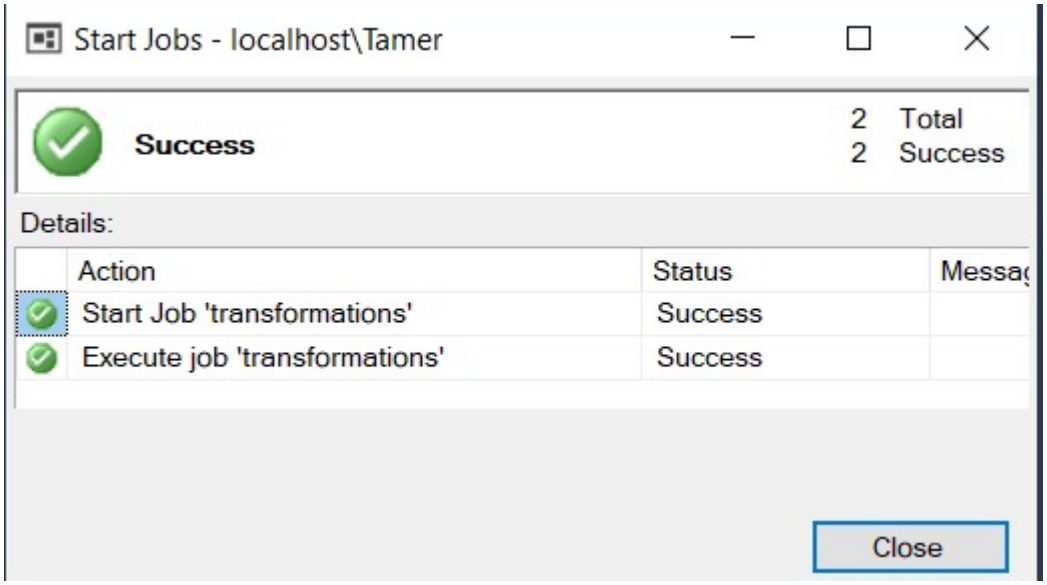

Job Properties - transformations
Select a page
General
Steps
Schedules
Alerts
Notifications
Targets
Script
Help
Job step list:
S... Name Type On Succ... On Failure
1 branchvendor SQL Se... Go to th... Quit the...
2 category SQL Se... Go to th... Quit the...
3 order SQL Se... Go to th... Quit the...
4 product SQL Se... Go to th... Quit the...
5 subcategory SQL Se... Go to th... Quit the...
6 vendor SQL Se... Quit the... Quit the...


Transformations
Packages
Trans_branch_vend.dtsx
Trans_Category.dtsx
Trans_Customer.dtsx
Trans_Order.dtsx
Trans_Product.dtsx
Trans_SubCategory.dtsx
Trans_Vendor.dtsx
Environments
SQL Server Agent


Start Job on 'localhost\Tamer'
Job name:
transformations

Job Start:


Start Jobs - localhost\Tamer
1 Remaining
2 Total
1 Success
etails:
Action
Status
Messag
Start Job 'transformations'
Success
Execute job 'transformations'
Close



Rationale for a Separate Transformation Project

We opted to create a separate transformation project to provide flexibility in managing and running transformations independently. This separation is particularly advantageous when we need to test or execute transformations without affecting other stages of the ETL process. Additionally, transformations could be executed within the staging area or database, depending on the specific data handling requirements.

ETL Process for Staging

The ETL process for the staging database was designed to maintain a historical record of all changes made to the data in the landing database. This was achieved by implementing a sample ETL package with multiple components, including two SQL task components and two data flow tasks.

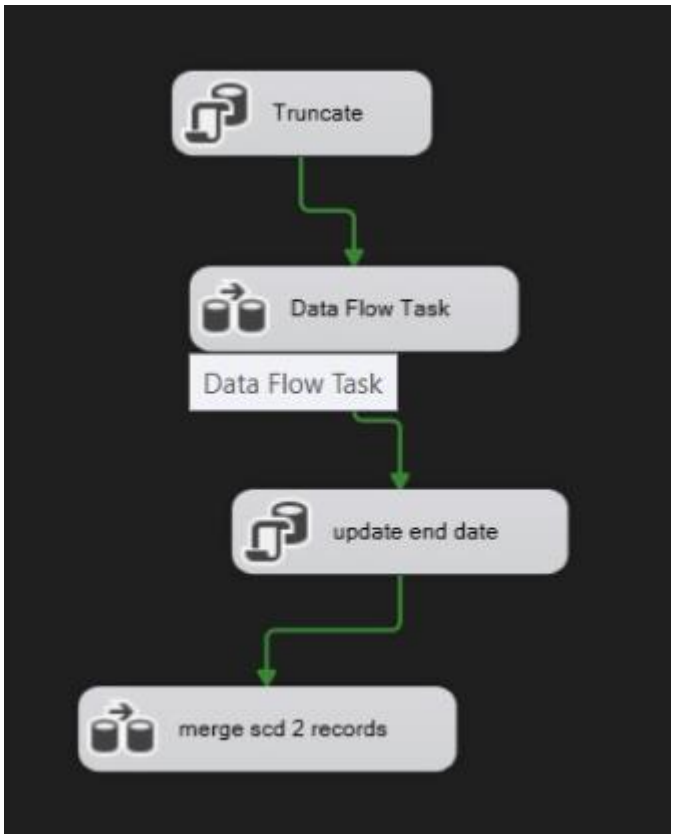
Staging Logic and Historical Data Tracking

To effectively track changes, we added three columns to our staging tables: StartDate, EndDate, and IsCurrent:

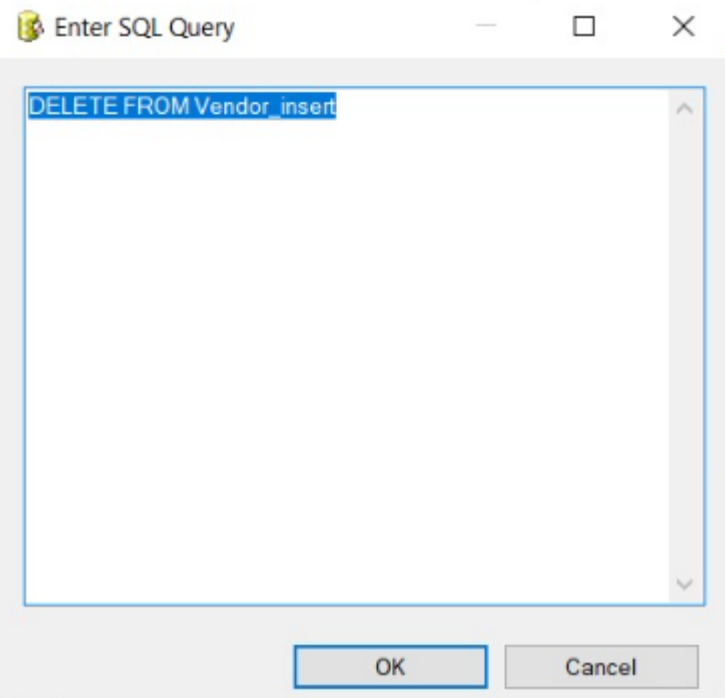
- **StartDate:** Represents the entry date of the data, corresponding to when the data became available. It is set to the staging date (STG Date).

- **EndDate:** Initially set to NULL to indicate that the data is currently valid. When a change occurs, the end date of the old record is updated to match the start date of the new record, signaling that the old record has expired.
- **IsCurrent:** A flag used to indicate whether the record is current (1 for current, 0 for expired)

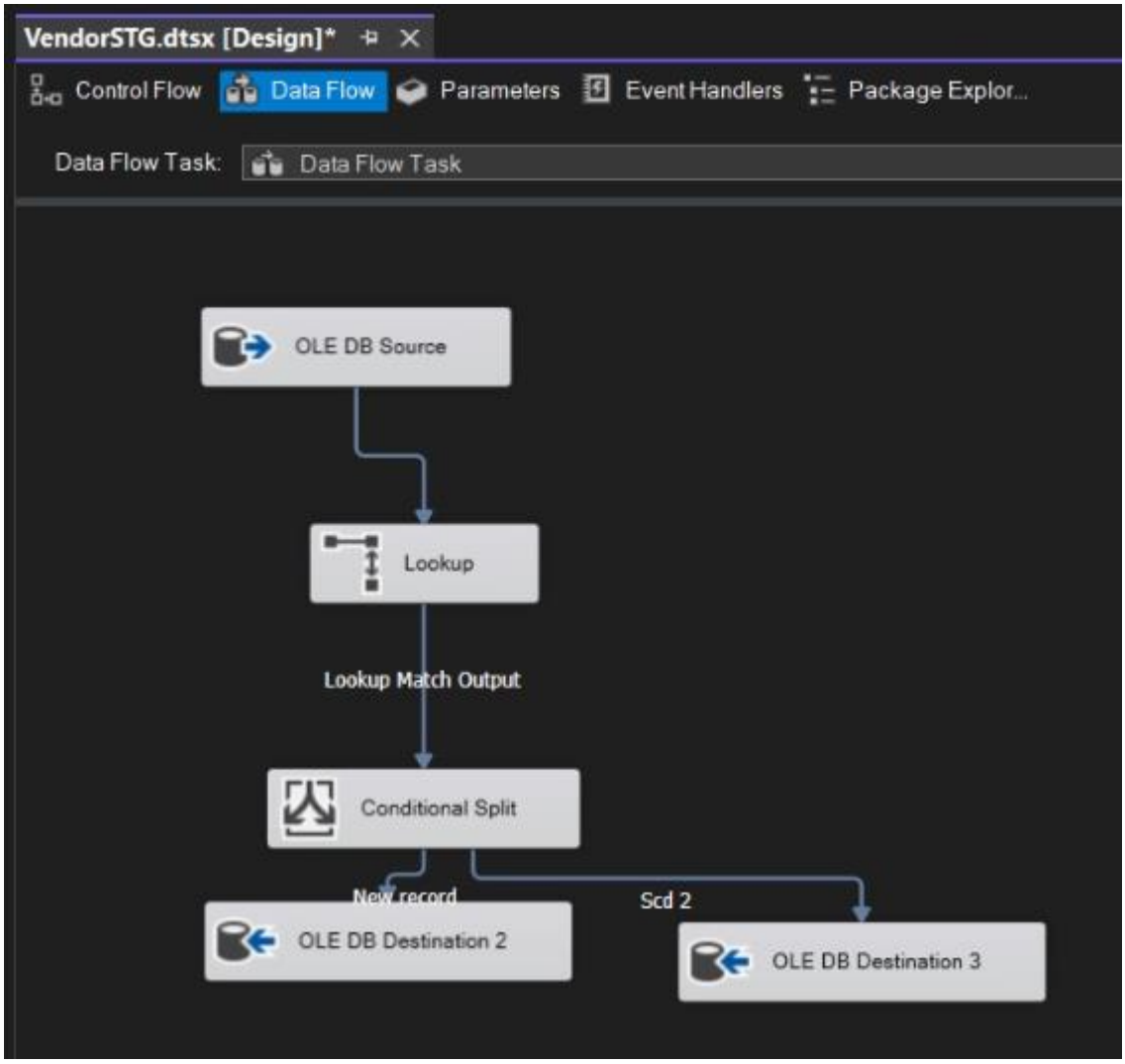
The package follows these steps to implement the logic:



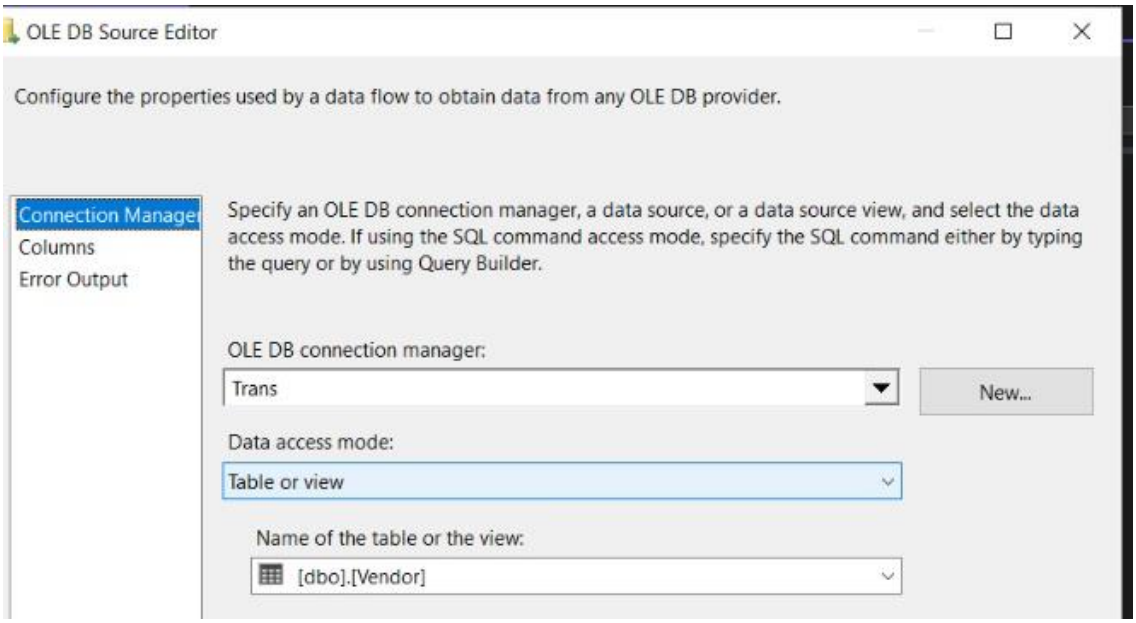
1. **Initial Truncation:**
    - The process begins by truncating a temporary table, table_inserted, which stores records temporarily before updating the staging database.



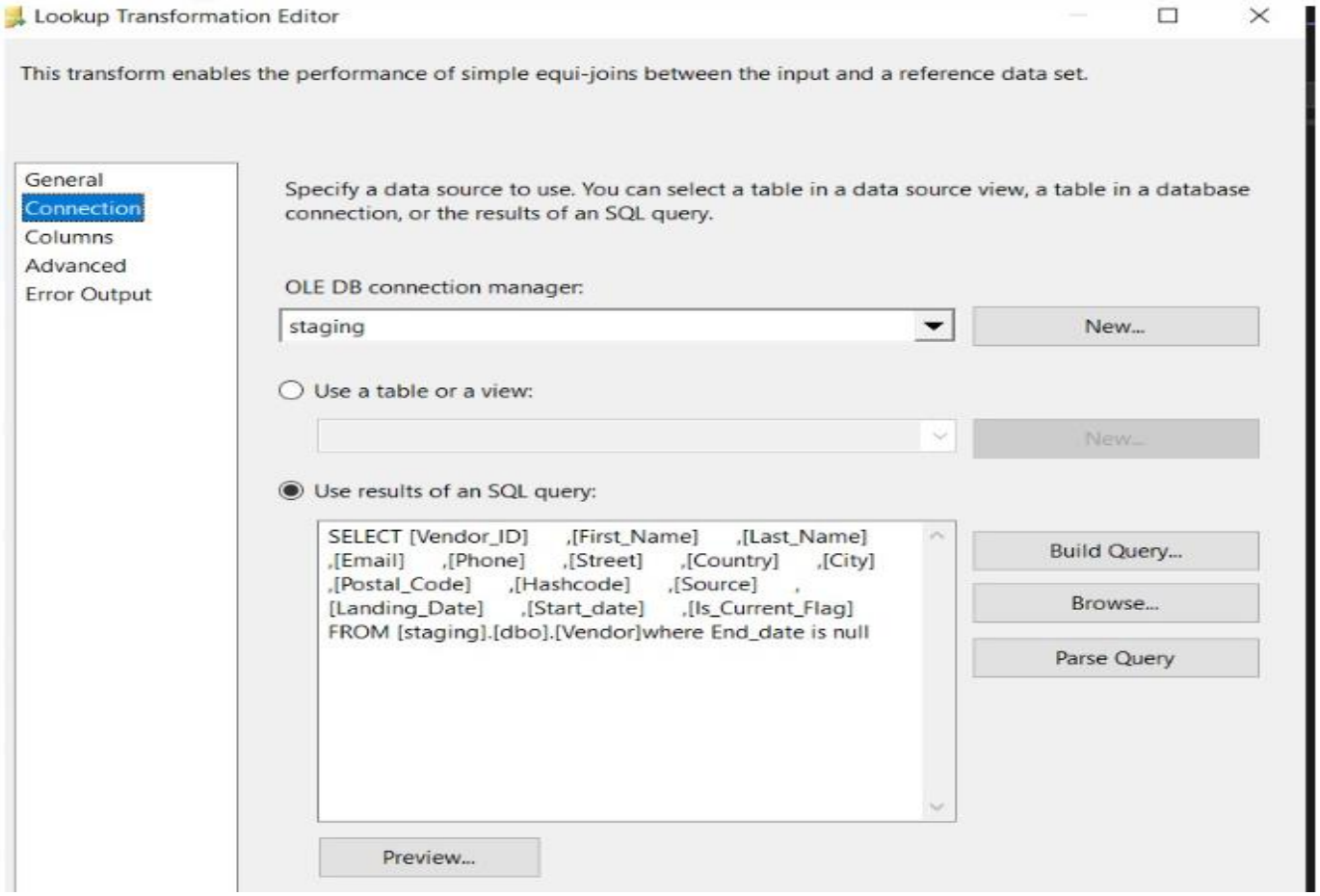
2. **First Data Flow Task:**



- The **OLE DB Source** retrieves data from the transformation database after it has been processed.



- A **Lookup Transformation** compares the data between the staging and transformation databases to detect any changes.



Lookup Transformation Editor

This transform enables the performance of simple equi-joins between the input and a reference data set.

- General
- Connection
- Columns
- Advanced
- Error Output

Available Inpu...

Name
Vendor_ID
First_Name
Last_Name
Email
Phone
Street
Country
City
Postal_C...

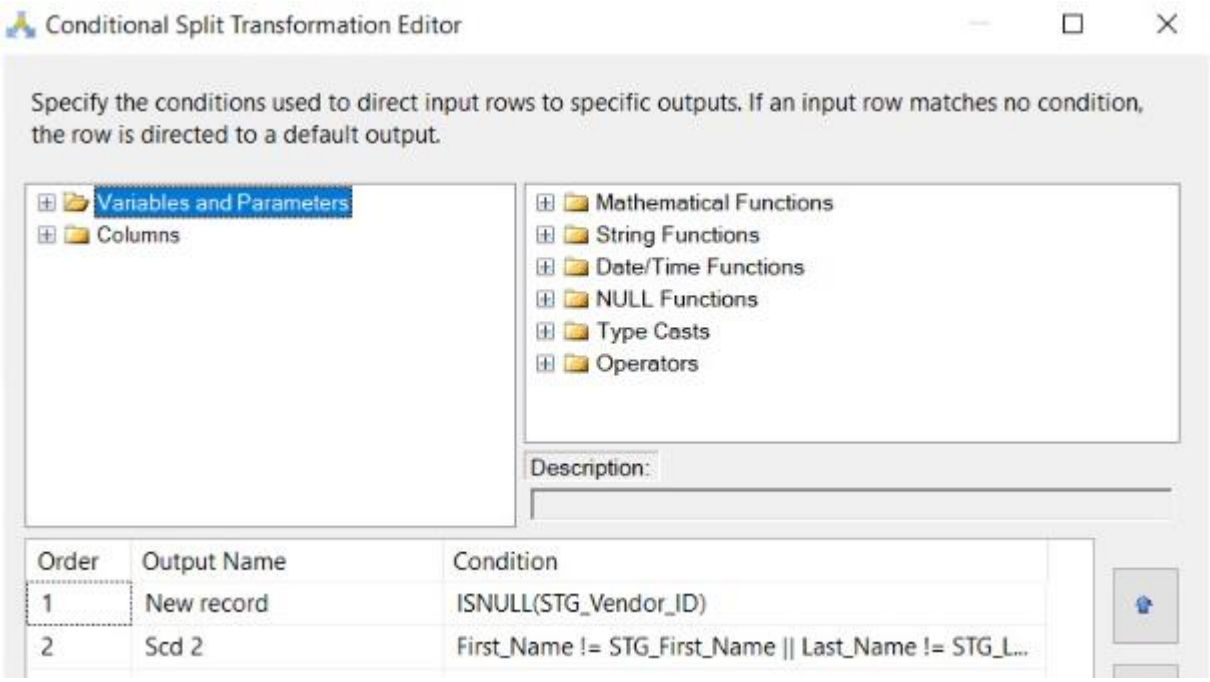
Available Lookup Colu...

Name	In...
Vendor_ID	
First_Name	
Last_Name	
Email	
Phone	
Street	
Country	
City	
Postal_C...	

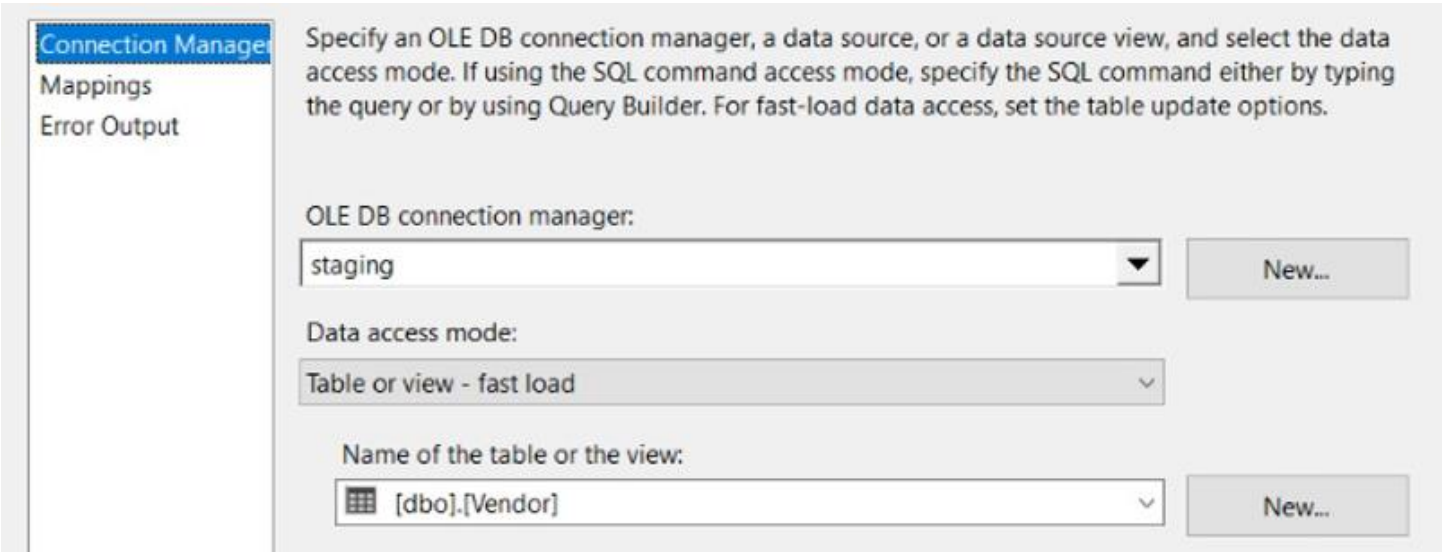
Lookup Column	Lookup Operation	Output Alias
Vendor_ID	<add as new column>	STG_Vendor_ID
First_Name	<add as new column>	STG_First_Name
Last_Name	<add as new column>	STG_Last_Name
Email	<add as new column>	STG_Email
Phone	<add as new column>	STG_Phone
Street	<add as new column>	STG_Street
Country	<add as new column>	STG_Country
City	<add as new column>	STG_City
Postal_Code	<add as new column>	STG_Postal_Code
Hashcode	<add as new column>	STG_Hashcode

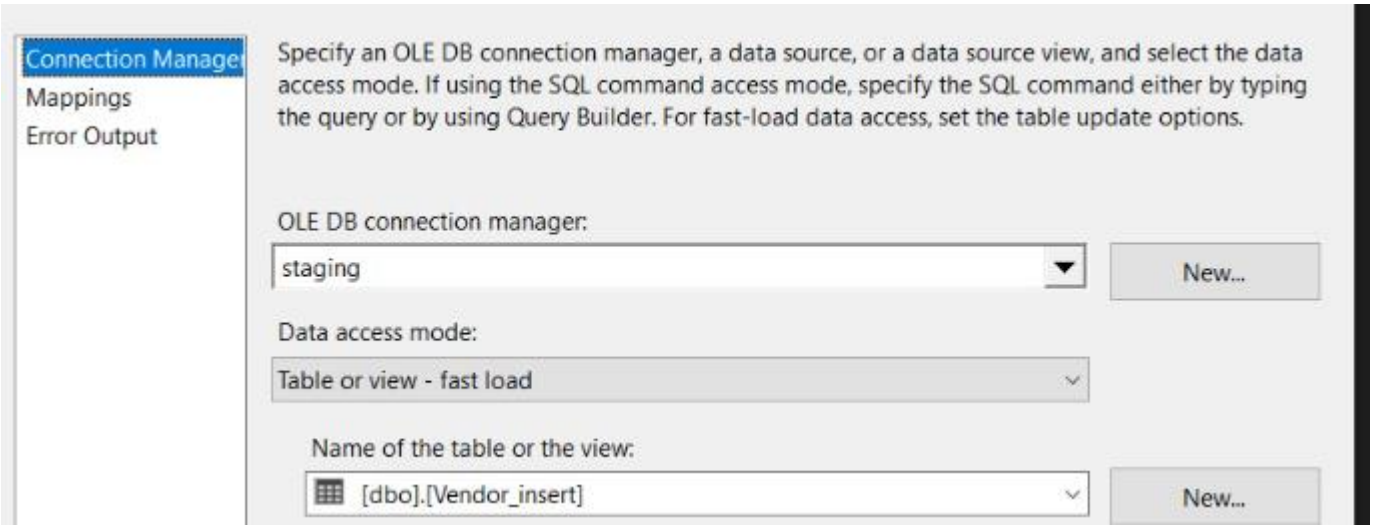
OK Cancel Help

- A **Conditional Split Transformation** evaluates two conditions:

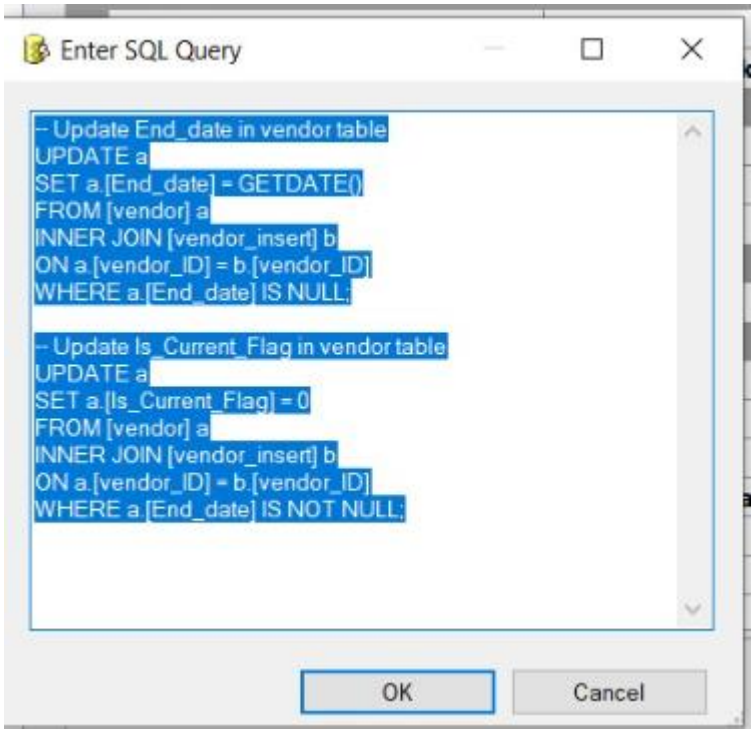


  - **New Records:** If an order ID is not found in the staging database, it is directly inserted.
  - **Updated Records:** If a hash code comparison indicates that a record has changed, it is added to table_inserted.



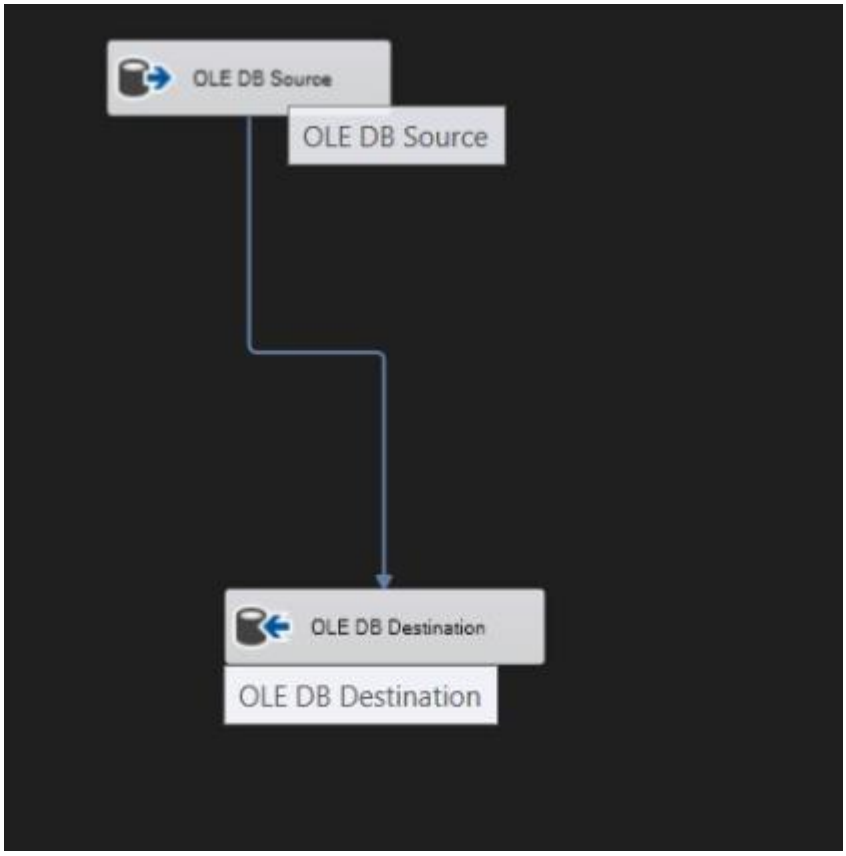


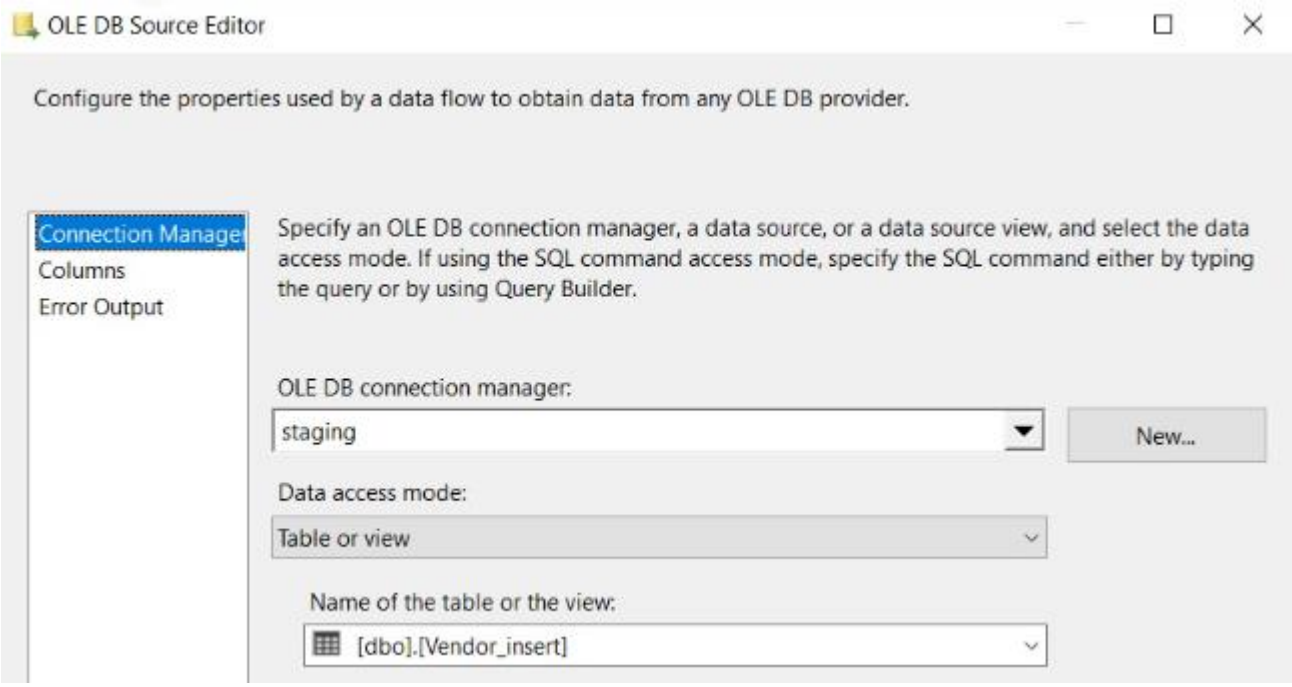
3. **Updating Historical Data:**
   - A **SQL Task Component** updates the old records in the staging table:
     - The EndDate of the existing (now outdated) record is updated to the start date of the new record.
     - The IsCurrent flag is set to 0 to indicate that the record is no longer valid.

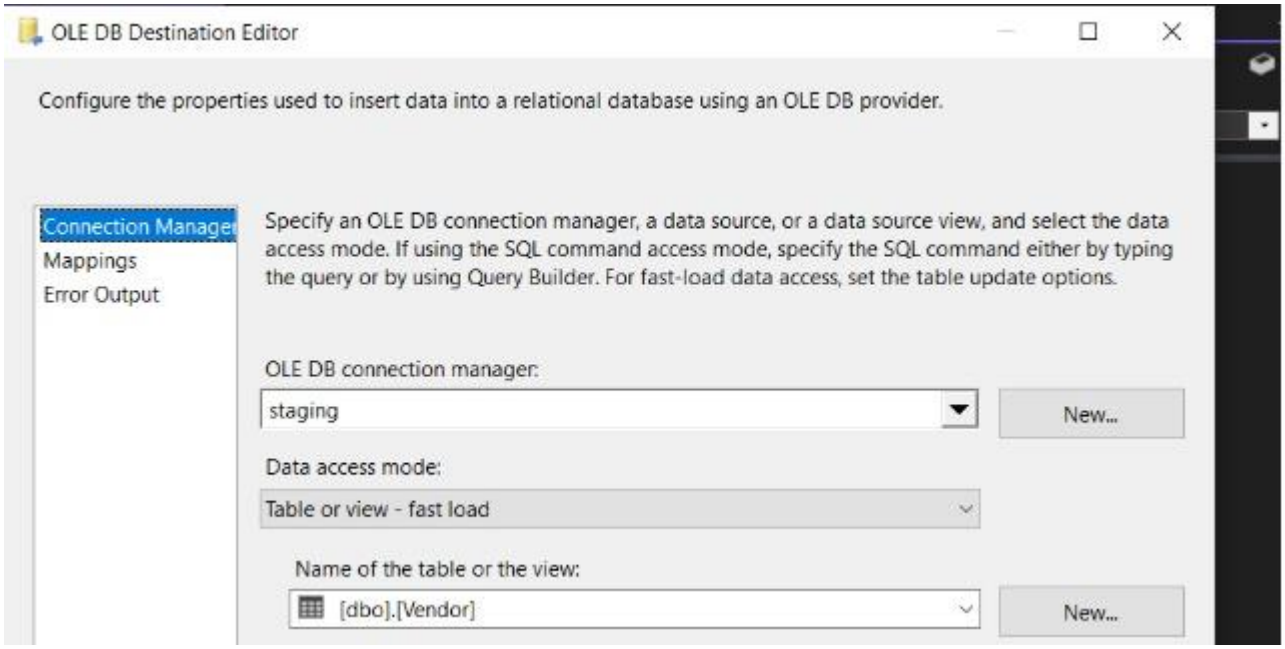


4. **Appending Updated Rows:**

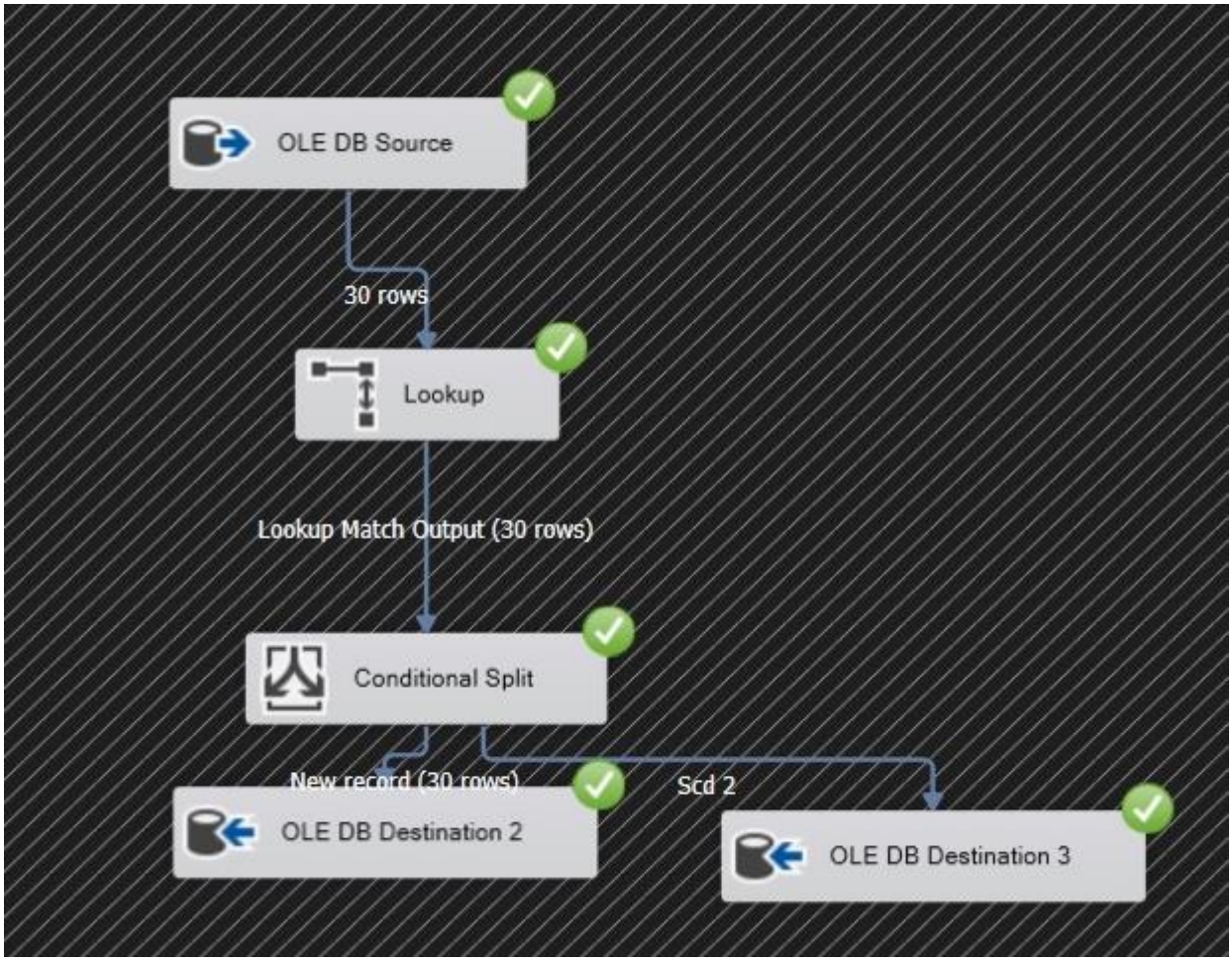
- The second **Data Flow Task** appends the updated rows from table_inserted to the staging table, ensuring that changes are accurately reflected.

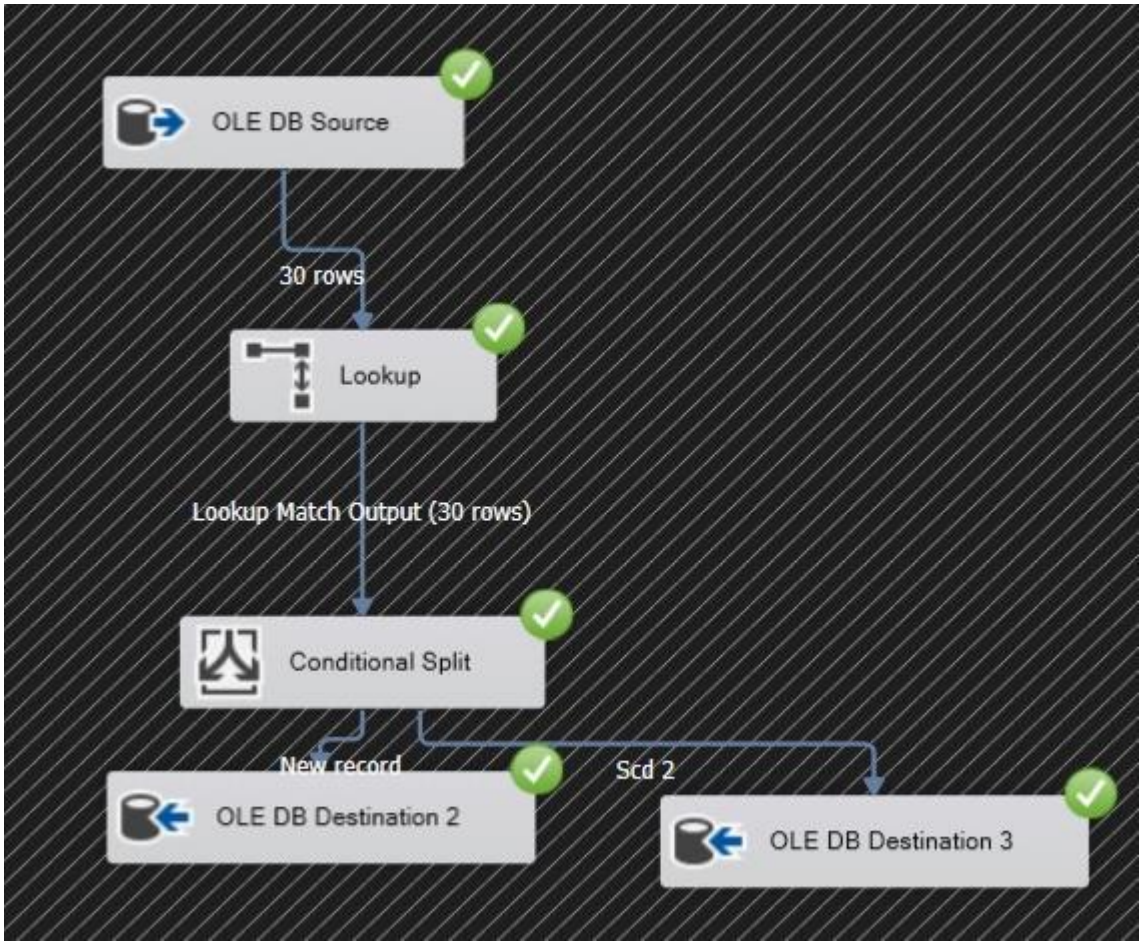


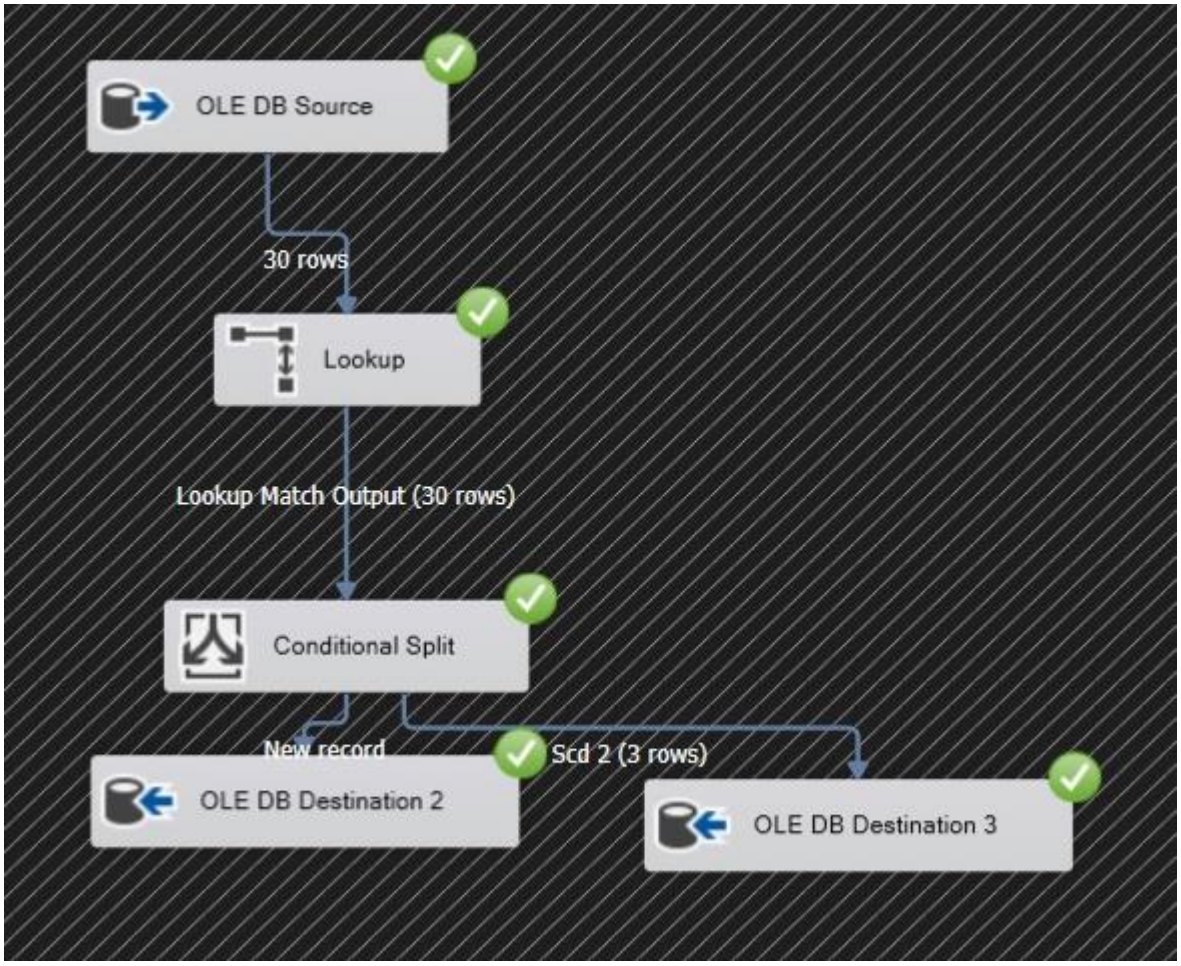




Package Multiple Runs:

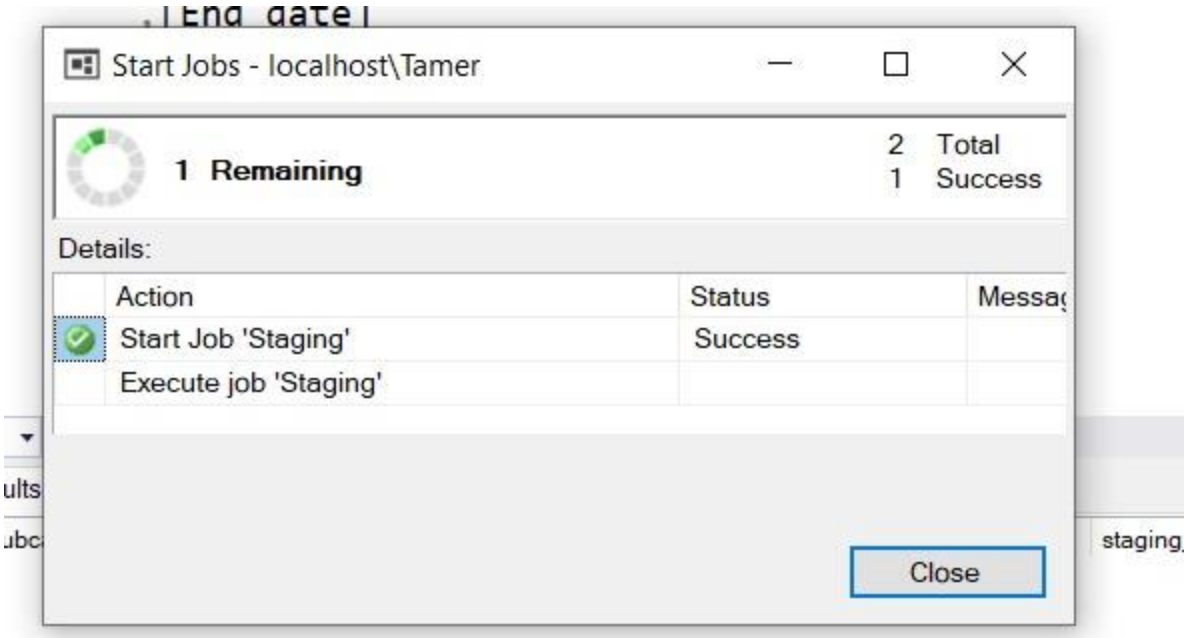


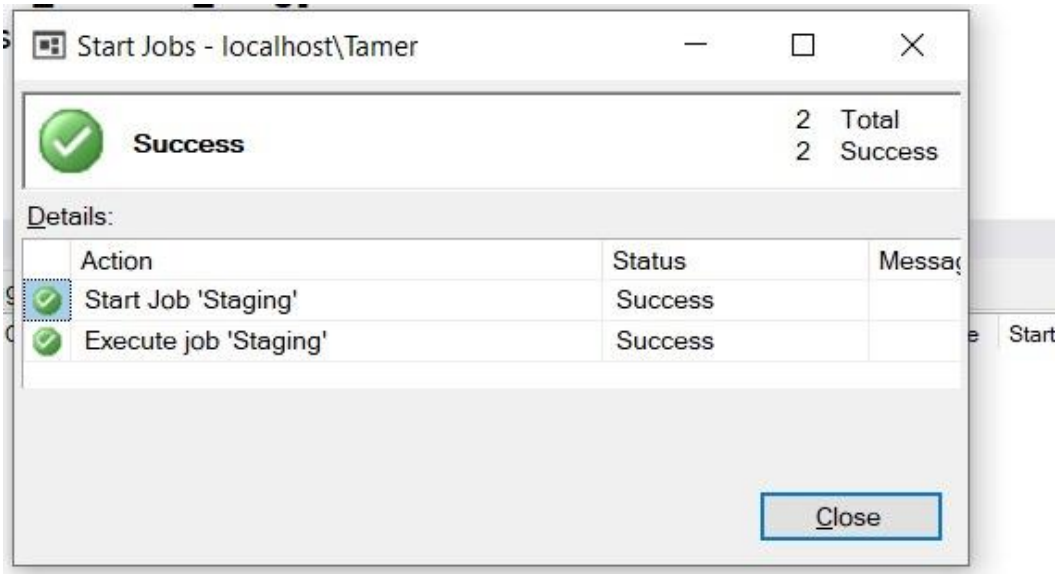

OLE DB Source
30 rows
Lookup
Lookup Match Output (30 rows)
Conditional Split
New record
Scd 2
OLE DB Destination 2
OLE DB Destination 3



Automation of Staging ETL Process

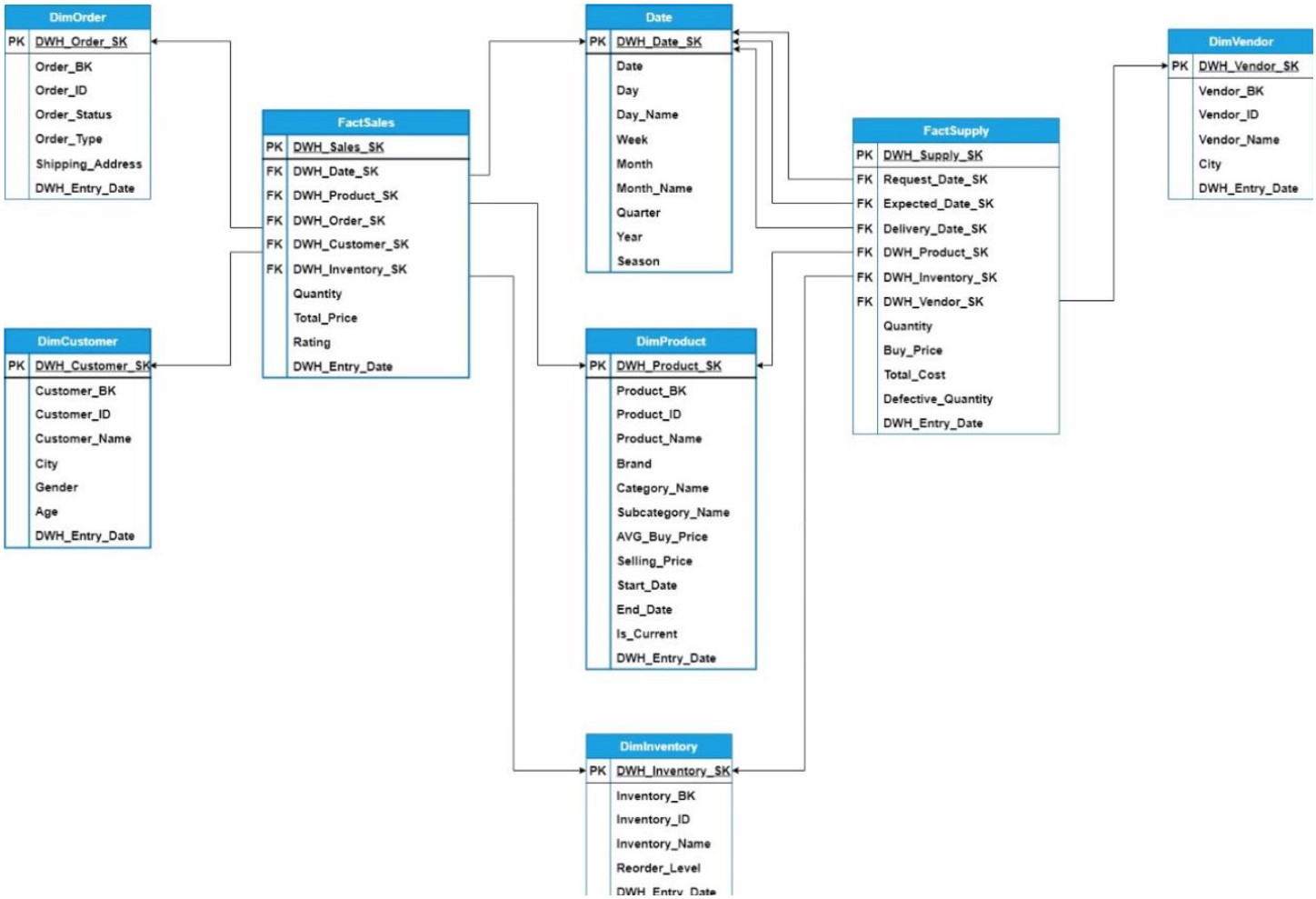
To automate the ETL process for the staging database, we configured a SQL Server Agent job. This job includes steps for executing the ETL packages, similar to the automation implemented for the transformation database.


Start Jobs - localhost\Tamer
1 Remaining
2 Total
1 Success
Details:
Action
Status
Start Job 'Staging'
Success
Execute job 'Staging'
Close


Start Jobs - localhost\Tamer
Success
2 Total
2 Success
Details:
Action
Status
Start Job 'Staging'
Success
Execute job 'Staging'
Success
Close

Data Loading Strategy for the Data Warehouse

The data warehouse for Jadara Mart is structured with six dimensions—**Customer, Inventory, Product, Date, Vendor, and Order**—and two fact tables: **Sales** and **Supply**.



For each of these tables, we developed an ETL package to handle the data loading process, ensuring both accuracy and efficiency.

Dimension Loading: SCD2 vs. Truncate and Load

Initially, we planned to implement **Slowly Changing Dimension Type 2 (SCD2)** across all dimensions, as it is a common practice in data warehousing to maintain historical records. Our data engineering mentors verified this approach, emphasizing the importance of using SCD2 for proper history tracking.

However, according to our data modeler's perspective, applying SCD2 universally might not be necessary. He advised that the **Product dimension** is the only one where SCD2 is crucial because we need to track the product history specifically. For the remaining dimensions—**Customer, Inventory, Date, Vendor, and Order**—SCD2 was deemed redundant since we had already captured historical changes at the **staging** level, and these dimensions did not require the same level of historical tracking.

Based on this recommendation, we opted for a truncate-and-load approach for most dimensions, which simplifies the process but presents challenges related to foreign key constraints. To manage these constraints, we implemented pre- and post-SQL scripts around each ETL package:

1. **Pre-SQL:**

   To manage foreign key constraints during the ETL process, we implemented a dynamic SQL script that temporarily disables all foreign key constraints in the target tables. This allows us to execute the truncate-and-load strategy without encountering constraint violations. The script works by iterating through all foreign keys and generating the necessary SQL statements to disable them.

Pre-SQL Script Logic:

The script follows these steps:

1. **Declare a Variable:** A NVARCHAR(MAX) variable @sql is declared to hold the dynamically generated SQL commands.
2. **Generate SQL for Each Foreign Key:**

- The script queries the sys.foreign_keys, sys.tables, and sys.schemas system views to identify all foreign key constraints in the database.
- For each foreign key, an ALTER TABLE ... NOCHECK CONSTRAINT statement is generated to disable the constraint.

3. **Execute the SQL Commands:**
   - The PRINT statement outputs the generated SQL commands for review.
   - The sp_executesql procedure executes the dynamic SQL, disabling the foreign key constraints.

2. Post-SQL:

Following the data load into the destination tables, it is essential to re-enable all foreign key constraints to ensure data integrity across the database. The script below dynamically re-enables any foreign key constraints that were previously disabled during the ETL process.

Post-SQL Script Logic:

1. **Declare a Variable:** A NVARCHAR(MAX) variable @sql is declared to store the dynamically generated SQL commands.
2. **Generate SQL for Each Disabled Foreign Key:**
   - The script queries the sys.foreign_keys, sys.tables, and sys.schemas system views to identify all foreign keys that are currently disabled.
   - For each disabled foreign key, an ALTER TABLE ... WITH CHECK CHECK CONSTRAINT statement is generated to re-enable the constraint and perform validation checks on existing data.
3. **Execute the SQL Commands:**
   - The PRINT statement outputs the generated SQL commands for review.
   - The sp_executesql procedure executes the generated SQL to enforce the foreign key constraints.

This method allows us to efficiently reload dimension tables without causing foreign key violations.

Fact Table Loading: Incremental Load for Efficiency

For the fact tables—**Sales** and **Supply**—we designed a slightly different approach. In the initial load, we performed a **full load**, using a direct data source-to-destination structure to populate the fact tables.

For subsequent loads, we switched to an **incremental load** strategy to optimize performance and prevent duplicate data. This involved adding a **Lookup transformation** based on the Data Warehouse surrogate key (SK). The lookup checks whether each incoming record already exists in the fact table:

- If the record is present, no action is taken.
- If the record is new, it is inserted into the fact table, preserving data integrity and maintaining a complete history of transactions.

This method ensures that our fact tables contain all relevant historical data while minimizing redundant data processing.

Technical Considerations for Dimension and Fact Loading

The combination of truncate-and-load for non-SCD2 dimensions and incremental loading for fact tables offers an efficient and scalable solution. However, it required careful management of dependencies, particularly foreign key constraints. The **pre/post SQL script approach** played a crucial role in avoiding integrity issues while allowing for simplified dimension updates.

By applying these methods, we achieved a balance between efficient data processing and the need to maintain a clean and accurate data warehouse environment, supporting both historical tracking and future scalability.

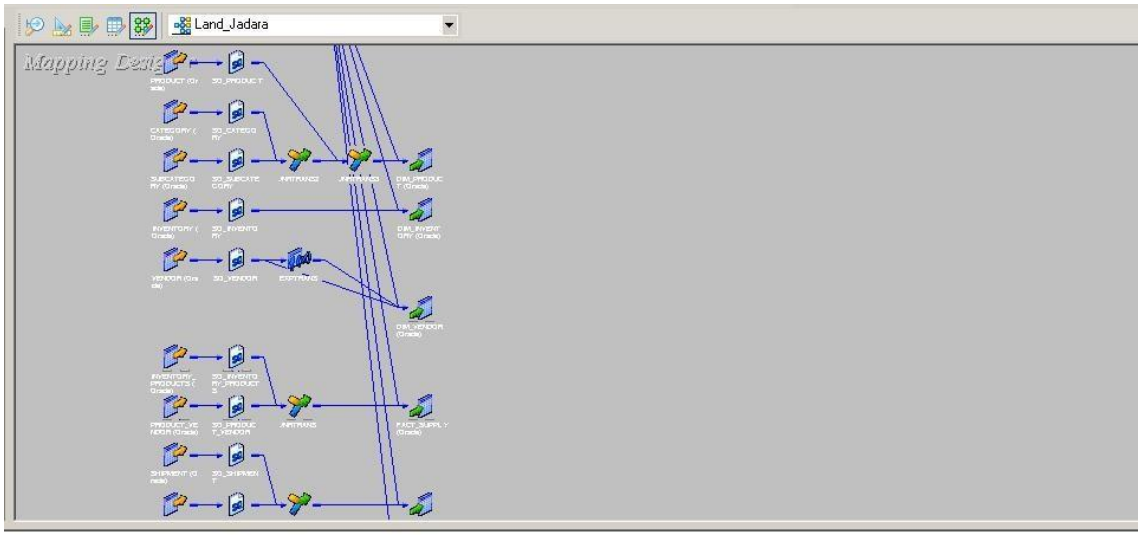
5.2 Informatica

Informatica Development Team: Mohamed Attia.

Landing :

- Took all the tables from the microsoft sql server management studio exported as csv then imported them in the oracle sql developer in the Virtual machine as oracle tables.
- Imported the tables as sources into informatica.
- Created tables to serve as targets in oracle sql developer.
- Mapped every source to its respective target while using expression transformation when needed.
- Successfully landed all sources into their respective targets with no errors.

Mapping :



Workflow results : All data is present in targets.

s_Land_Jadara [9/13/2024 6:23:47 AM] Get Session Log

Task Details

Source/Target Statistics

Transformation Name	Node	Applied Rows	Affected Rows	Rejected Rows	Throughput (Rows/Sec)	Throughput (Bytes/Sec)	Bytes	Last Error Code	Last Error Message
DIM_CUSTOMER1	NODE01_INTD...	130000	130000	0	130000	43940000	43940000	0	No errors encountered.
DIM_INVENTORY	NODE01_INTD...	4	4	0	4	904	904	0	No errors encountered.
DIM_ORDER	NODE01_INTD...	91839	91839	0	91839	56756502	56756502	0	No errors encountered.
DIM_PRODUCT	NODE01_INTD...	293	293	0	293	133901	133901	0	No errors encountered.
DIM_VENDOR	NODE01_INTD...	30	30	0	30	9570	9570	0	No errors encountered.
FACT_SALES	NODE01_INTD...	0	0	0	0	0		0	No errors encountered.
FACT_SUPPLY	NODE01_INTD...	98142	98142	0	98142	13052886	13052886	0	No errors encountered.
SQ_CATEGORY	NODE01_INTD...	4	4	0	4	604	604	0	No errors encountered.
SQ_CUSTOMERS	NODE01_INTD...	130000	130000	0	130000	17030000	17030000	0	No errors encountered.
SQ_INVENTORY	NODE01_INTD...	4	4	0	4	508	508	0	No errors encountered.
SQ_INVENTORY...	NODE01_INTD...	878	878	0	878	137846	137846	0	No errors encountered.
SQ_ORDER_PR...	NODE01_INTD...	464486	464486	0	464486	59454208	59454208	0	No errors encountered.
SQ_ORDERS	NODE01_INTD...	91839	91839	0	91839	10837002	10837002	0	No errors encountered.
SQ_PRODUCT	NODE01_INTD...	293	293	0	293	118665	118665	0	No errors encountered.
SQ_PRODUCT_...	NODE01_INTD...	32721	32721	0	32721	5366244	5366244	0	No errors encountered.
SQ_SHIPMENT	NODE01_INTD...	80000	80000	0	80000	13200000	13200000	0	No errors encountered.
SQ_SUBCATEG...	NODE01_INTD...	37	37	0	37	5328	5328	0	No errors encountered.
SQ_VENDOR	NODE01_INTD...	30	30	0	30	3570	3570	0	No errors encountered.

Example : Customer table

Columns | Data | Constraints | Grants | Statistics | Triggers | Dependencies | Details | Partitions | Indexes | SQL

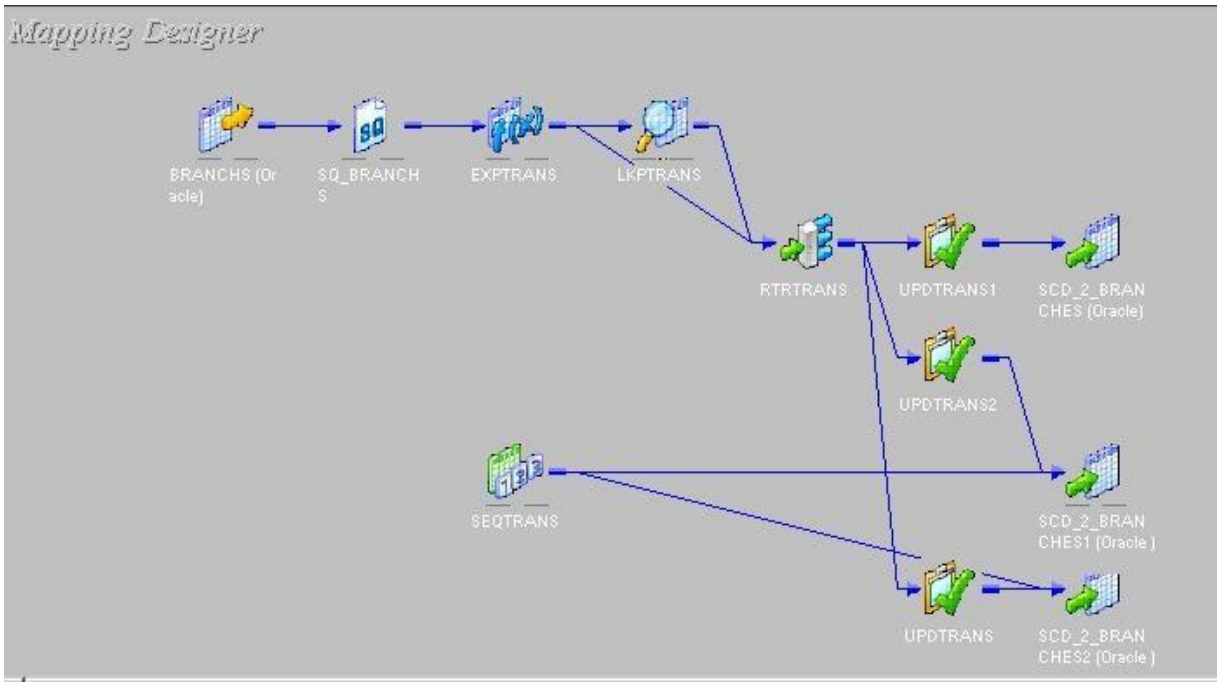
Sort... Filter:

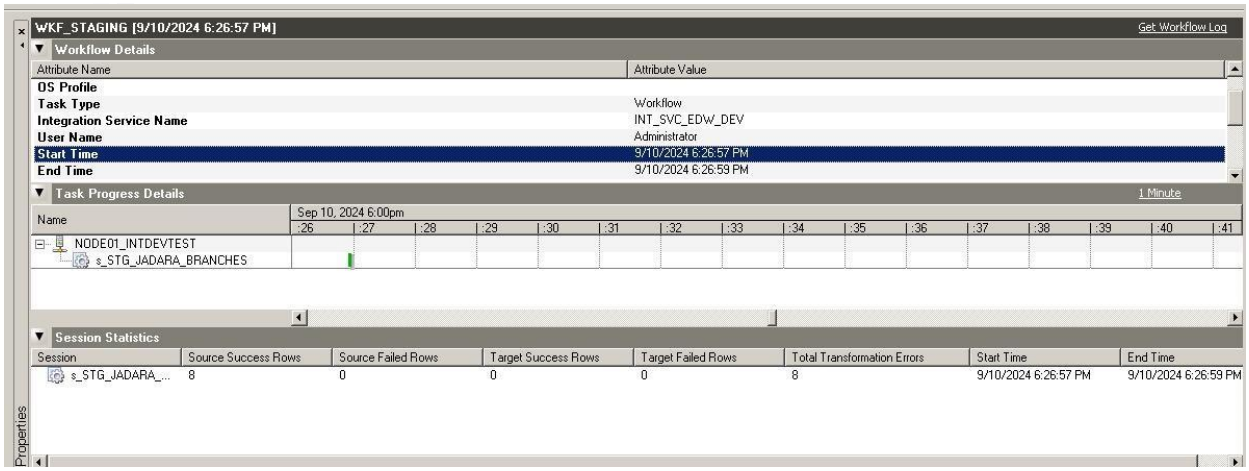
	CUSTOMER_SK	CUSTOMER_ID	CUSTOMER_NAME	CITY	GENDER	AGE	HASHCODE
1	790	162	ZeinaZaki	Matrouh	Female	49	9b433734d3fadcd5d26d2b33234aacb1e8a72c20058b2c0caca816f2f67ac796
2	791	163	RaoufMohamed	Kafr El Sheikh	Male	41	61d86b344a1cd9cb255dbaac08080044d69efa8e60f54045d819f54730fd74fa
3	792	164	AshrafRashad	Aswan	Male	32	e22f895d237969583f4b492ae5f9fd0c555447fe6f292aaf0239b923e9fddb26
4	793	165	FaroukGamal	Cairo	Male	47	c20005b1266f79c0c666c8c784a4afb0457d3d3fe557bcc1857b6eff09f7b456
5	794	166	BasemIsmail	Fayoum	Male	58	8d3699b100a6b6004c5d2128b1bbec1c16c56e2922d7fe6b53018a4ce721b16a
6	795	167	LotfySalah	Ismailia	Male	26	38827f9d6ef32a3cb7ffb72e80310fcd0acc2296f2e7e0d6ed3ec8091c2320d8
7	796	168	ShoroukHelmy	Sohag	Female	18	a3672c126ab31b7c1455c2dce8dace1460db52f5052914a21bd61fc695961e48
8	797	169	InasHamed	Cairo	Female	40	d2e50df5b16ba94fe1109125eb87d6cc75a570cd0b43c39eda3173775fa894a0
9	798	170	NasserSherif	Luxor	Male	31	0de9748344839ba5aa964479fc899b6856e24dd0d51170a3e05483bd1a19a83e
10	799	342	SamiraAdel	New Cairo	Female	27	c9e4bbd4080b46e2682cef267622194af74b92d99e7e2797ecc2891784fd4843
11	800	343	ImanFathy	New Cairo	Female	41	4a0db7cdd4c2df203dfdb2a8eee20566e297a416da45b069c2045284d6c520a0
12	801	344	GamalHatem	Aswan	Male	65	24aa4e9446466000c5b4179721f928e3c6402eee647a372ba5ab1c161cf82b4d
13	802	345	MagdyAshraf	Hurghada	Male	21	ab3ef3325d9622eaf517aec02ff9b8047a2516097a7076cbe70c32736d96ac69
14	803	346	RashadMaged	Gharbia	Male	68	d3fa453637080904fe2db4ae617a7f12e85b9b881512a6c51b31fb4faac65f71
15	804	347	FaridaKhalaf	Cairo	Female	33	6512228ae7de77eb83d0c8fc7d4b26156ad4364f47283dbcd05d2a7f53feac77
16	805	348	OsamaSherif	Sharm El Sheikh	Male	64	14f1435b2f32b4388a8cf4f9b2d4d08c122c7c88e94a169297fac2ee2974ca86
17	806	349	FaridIsmail	Damanhour	Male	67	0fab1fbd6a2e66c1c1cd505cecae037a9a5eccd3572ee8dbdcfb8ca96e3d0b16
18	807	350	LamiaSherif	Alexandria	Female	31	f7493aa342ed7330b3366a080e83e20cf0aed4db85d670a1ba5362ceb782d362
19	808	351	FaridHossam	Beni Suef	Male	70	b20798074a101e5d35baecaae18479674475affa88059c414bbac8eb4fd9df02
20	809	352	MazenKhaled	Aswan	Male	40	f65470eee36686e342a6dfed46595afe08bc1adcddfb6cc881871dd2a0a17705
21	810	353	RamyKhalil	Cairo	Male	34	c8b9c86294b9d354821596e4fb6e26fd511b2695052b4a4b09a903dcf4662e96
22	811	354	GalalSherif	Tanta	Male	25	609019ab73e7d1f786f6b80f2c53ce7383e98a37fb4e23b01b126c8d4016c84c
23	812	355	MahmoudAnwar	Ismailia	Male	50	2c41feef62f1b29bef4b182e8b6ee549706dfe58d1058e1918d0b8be3ffb2805
24	813	356	SamehKhaled	Tanta	Male	60	a0871790d064b9186dee885573a7d1ea5802f22dde81839368152609edd16c31
25	814	357	MoustafaAbdelrahman	Matrouh	Male	48	c1c4f13751685b7edc6c5b1081fe64a9b370f590fb0d30d9587c94431ab9c2a6
26	815	358	SouadKhaled	Fayoum	Female	47	6945d813a5a49b132b32552282e6e6db59f1550243a5e202ac0190f73b3eee0b
27	816	359	MagdyFarouk	Alexandria	Male	61	0c7abaf2ef34d4170f1f68f990c718bc883b97f6a6d5501693cbe251b3c69619
28	817	360	EssamOmar	Damietta	Male	42	22d28b11a74ad2e4899792a9f83de9751d01db568a94478ff5e001c2a797690a

Staging :

- Applied all the necessary transformations on the data to move from landing to staging such as SCD.

Mapping :

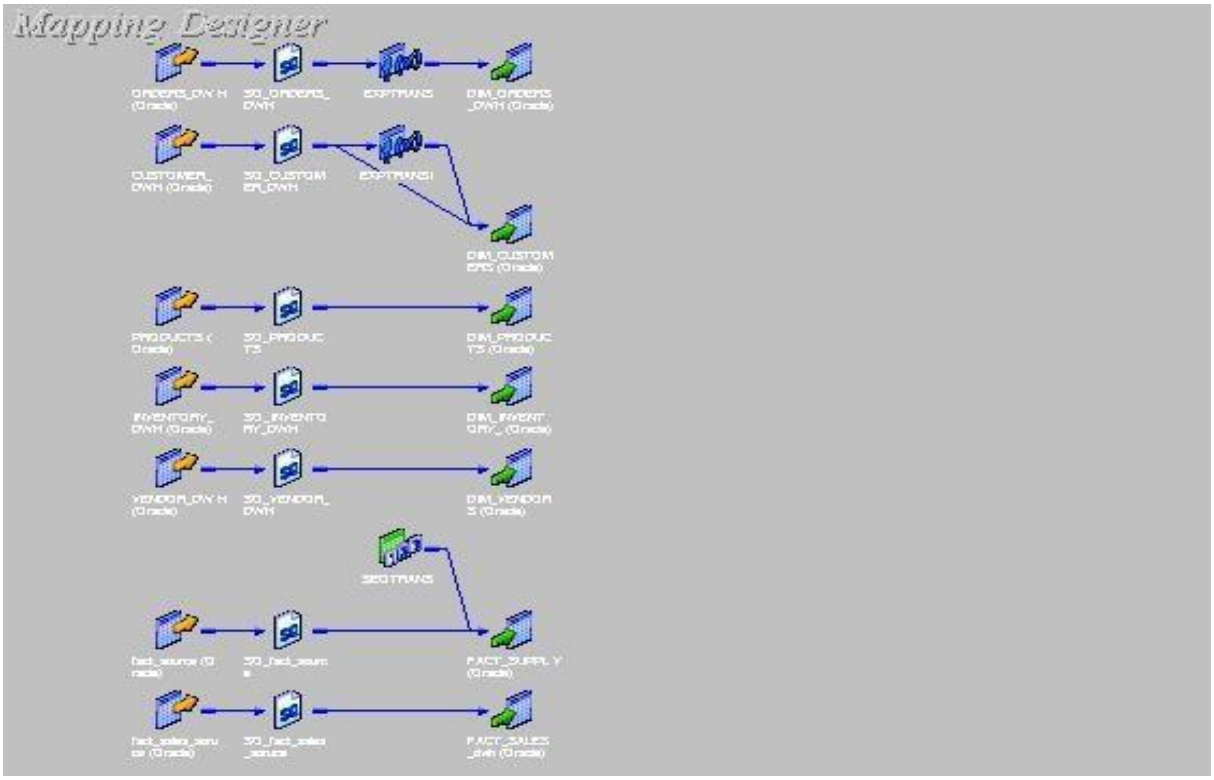




Data WareHouse:

- Took all the tables from the microsoft sql server as csv then imported them in to the oracle sql developer in the virtual machine.
- Used the mapping sheet to correctly map every source to its respective target.
- All data is successfully entered in the DWH schema

Mapping :



s_DWH_JADARA [9/13/2024 4:07:04 AM] Get Session Log

Task Details

Source/Target Statistics

Transformation Name	Node	Applied Rows	Affected Rows	Rejected Rows	Throughput (Rows/Sec)	Throughput (Bytes/Sec)	Bytes	Last Error Code	Last Error Message
DIM_CUSTOME...		99999	99999	0	99999	33799662	33799662	0	No errors encountered.
DIM_INVENTOR...		4	4	0	4	628	628	0	No errors encountered.
DIM_ORDERS_...		91839	91839	0	91839	13224816	13224816	0	No errors encountered.
DIM_PRODUCTS		490	490	0	490	99470	99470	0	No errors encountered.
DIM_VENDORS		30	30	0	30	4110	4110	0	No errors encountered.
FACT_SALES_dwh		5542	5542	0	5542	1679226	1679226	0	No errors encountered.
FACT_SUPPLY		1006	1006	0	1006	190134	190134	0	No errors encountered.
SQ_CUSTOMER...		99999	99999	0	99999	13599864	13599864	0	No errors encountered.
SQ_fact_sales_s...		5542	5542	0	5	495	548658	0	No errors encountered.
SQ_fact_source		1006	1006	0	503	60863	121726	0	No errors encountered.
SQ_INVENTORY...		4	4	0	4	560	560	0	No errors encountered.
SQ_ORDERS_D...		91839	91839	0	91839	11112519	11112519	0	No errors encountered.
SQ_PRODUCTS		490	490	0	490	193550	193550	0	No errors encountered.
SQ_VENDOR_D...		30	30	0	30	3690	3690	0	No errors encountered.

Fact sales table result:

	DATE_SK	PRODUCT_SK	ORDER_SK	CUSTOMER_SK	INVENTORY_SK	QUANTITY	TOTAL_PRICE	RATING	HASHCODE	DWH_FACTSALES_SK
1	1877	423	514	7905	14	1	336.859593	2	P:-OP:d934	1877
2	1901	421	536	50571	14	1	343.222636	1	P:-OP:a5d1	1901
3	2222	471	94	34500	14	1	34.165666	3	P:-OP:914d	2222
4	2223	492	703	8319	14	3	687.438558	3	P:-OP:8b83	2223
5	2278	390	38	25658	14	3	329.379999	1	P:-OP:cf48	2278
6	2385	525	222	21068	14	3	77.028258	4	P:-OP:3895	2385
7	2095	510	605	9652	14	1	2.969393	4	P:-OP:25a2	2095
8	2361	372	643	21592	14	3	2040.09906	3	P:-OP:61a7	2361
9	2544	511	401	46151	14	3	475.870275	1	P:-OP:eee9	2544
10	2004	386	444	16331	14	1	639.819918	4	P:-OP:f209	2004
11	2411	506	531	20387	14	1	165.852831	1	P:-OP:2cae	2411
12	2558	407	469	11438	14	3	616.164816	1	P:-OP:b1e4	2558
13	2009	381	479	45195	14	2	2808.83252	2	P:-OP:a27c	2009
14	2452	435	111	29049	14	3	84.028983	4	P:-OP:41b8	2452
15	2656	385	765	19467	14	3	1870.11224	4	P:-OP:df5e	2656
16	2063	542	142	57805	14	3	4447.48528	5	P:-OP:cb65	2063
17	2490	500	250	58682	14	2	66.767272	2	P:-OP:c272	2490
18	2206	468	570	59145	14	2	68.464806	3	P:-OP:8693	2206
19	2004	542	810	29492	14	3	4447.48528	4	P:-OP:6864	2004
20	2568	434	791	7345	14	2	43.542222	3	P:-OP:ba5c	2568
21	2375	301	555	53063	14	2	60.55339	1	P:-OP:02df	2375
22	1917	403	160	28898	14	2	263.653274	5	P:-OP:b840	1917
23	2486	321	43	52800	14	1	376.132079	2	P:-OP:37d2	2486
24	2643	546	147	59942	14	2	929.18818	3	P:-OP:32b9	2643
25	2666	388	339	36987	14	1	644.522056	1	P:-OP:5de7	2666
26	2333	564	582	39828	14	3	1005.87710	2	P:-OP:f19a	2333
27	2303	563	591	57920	14	1	1057.26809	2	P:-OP:404a	2303
28	2326	424	630	47057	14	2	1625.82517	4	P:-OP:8e75	2326

Fact supply table result:

	DATE_SK	PRODUCT_SK	ORDER_SK	CUSTOMER_SK	INVENTORY_SK	QUANTITY	TOTAL_PRICE	RATING	HASHCODE	DWH_FACTSALES_SK
1	1877	423	514	7905	14	1	336.859593	2	P:-OP:d934	1877
2	1901	421	536	50571	14	1	343.222636	1	P:-OP:a5d1	1901
3	2222	471	94	34500	14	1	34.165666	3	P:-OP:914d	2222
4	2223	492	703	8319	14	3	687.438558	3	P:-OP:8b83	2223
5	2278	390	38	25658	14	3	329.379999	1	P:-OP:cf48	2278
6	2385	525	222	21068	14	3	77.028258	4	P:-OP:3895	2385
7	2095	510	605	9652	14	1	2.969393	4	P:-OP:25a2	2095
8	2361	372	643	21592	14	3	2040.09906	3	P:-OP:61a7	2361
9	2544	511	401	46151	14	3	475.870275	1	P:-OP:eee9	2544
10	2004	386	444	16331	14	1	639.819918	4	P:-OP:f209	2004
11	2411	506	531	20387	14	1	165.852831	1	P:-OP:2cae	2411
12	2558	407	469	11438	14	3	616.164816	1	P:-OP:b1e4	2558
13	2009	381	479	45195	14	2	2808.83252	2	P:-OP:a27c	2009
14	2452	435	111	29049	14	3	84.028983	4	P:-OP:41b8	2452
15	2656	385	765	19467	14	3	1870.11224	4	P:-OP:df5e	2656
16	2063	542	142	57805	14	3	4447.48528	5	P:-OP:cb65	2063
17	2490	500	250	58682	14	2	66.767272	2	P:-OP:c272	2490
18	2206	468	570	59145	14	2	68.464806	3	P:-OP:8693	2206
19	2004	542	810	29492	14	3	4447.48528	4	P:-OP:6864	2004
20	2568	434	791	7345	14	2	43.542222	3	P:-OP:ba5c	2568
21	2375	301	555	53063	14	2	60.55339	1	P:-OP:02df	2375
22	1917	403	160	28898	14	2	263.653274	5	P:-OP:b840	1917
23	2486	321	43	52800	14	1	376.132079	2	P:-OP:37d2	2486
24	2643	546	147	59942	14	2	929.18818	3	P:-OP:32b9	2643
25	2666	388	339	36987	14	1	644.522056	1	P:-OP:5de7	2666
26	2333	564	582	39828	14	3	1005.87710	2	P:-OP:f19a	2333
27	2303	563	591	57920	14	1	1057.26809	2	P:-OP:404a	2303
28	2326	424	630	47057	14	2	1625.82517	4	P:-OP:8e75	2326

6. Data Warehouse Model

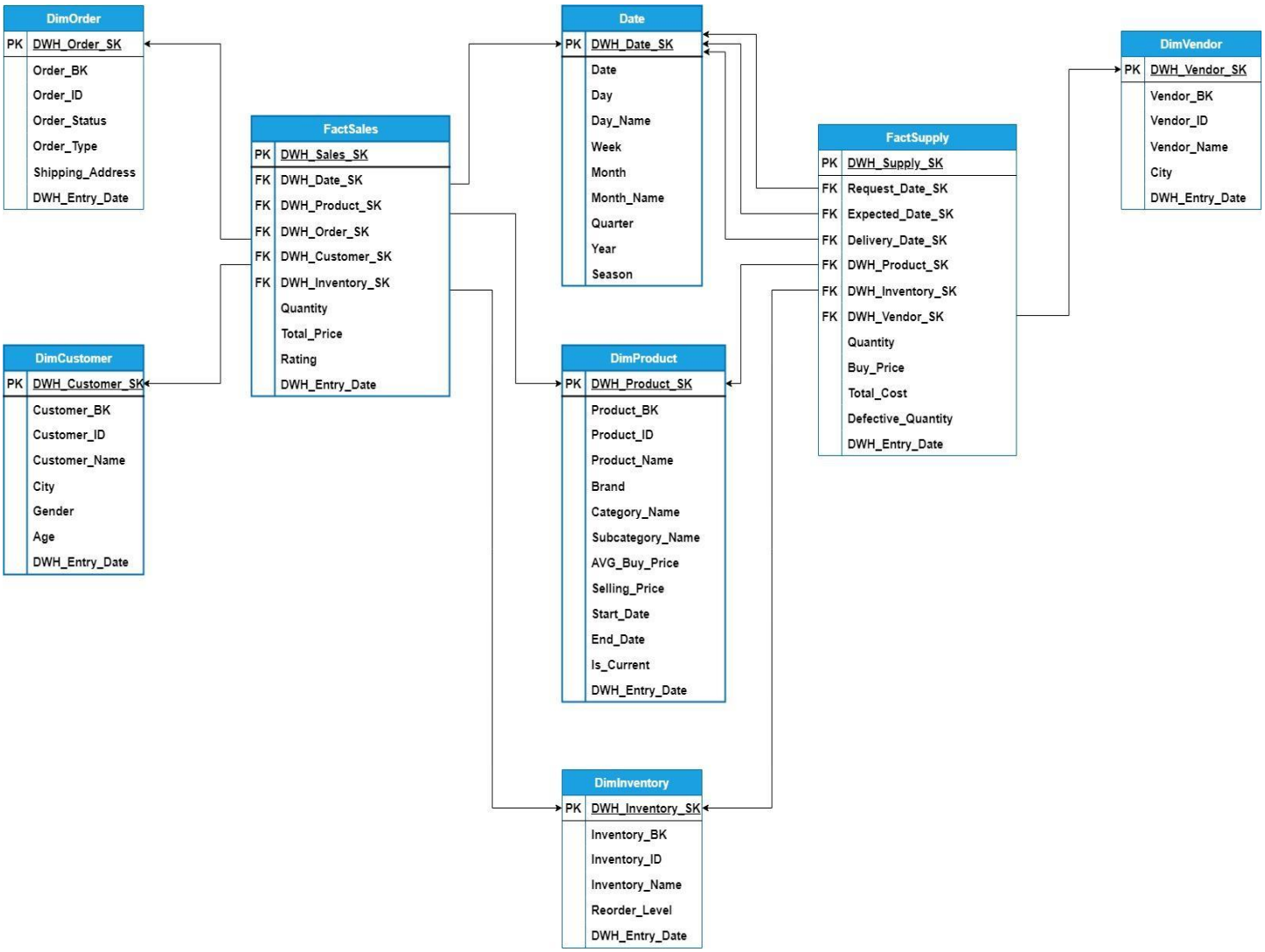
Data Modeling Development Team: Marwan Amr, Mohamed Ibrahim, Hala Ibrahim

Overview

This document provides a detailed description of the Data Warehouse (DWH) schema for **Jadara Mart**, which integrates sales and supply information across various dimensions. This schema supports the company's business intelligence efforts by allowing for comprehensive reporting on sales, customer demographics, product inventory, and vendor supplies.

The schema consists of the following major components:

1. **Dimensions**: Provide the contextual information needed for analysis (e.g., Customers, Products, Vendors).
2. **Fact Tables**: Contain transactional data (e.g., sales, supply details).



Dimensions

1. DimCustomer

- **Purpose**: This table contains detailed information about each customer.

- **Columns**:
  - **DWH_Customer_SK (PK)**: Surrogate key used as the primary key in the data warehouse for customer-related joins.
  - **Customer_BK**: Business key from the source system.
  - **Customer_ID**: The original customer ID from the operational system.
  - **Customer_Name**: Full name of the customer.
  - **City**: The city where the customer resides.
  - **Gender**: Gender of the customer.
  - **Age**: Age of the customer.
  - **DWH_Entry_Date**: The date this record was inserted or updated in the DWH.

2. DimOrder

- **Purpose**: This table holds order-related details, providing context for sales.
- **Columns**:
  - **DWH_Order_SK (PK)**: Surrogate key for orders.
  - **Order_BK**: Business key of the order.
  - **Order_ID**: Order identifier from the transactional system.
  - **Order_Status**: Status of the order (e.g., Pending, Completed).
  - **Order_Type**: Type of order (e.g., Online, In-store).
  - **Shipping_Address**: Shipping address associated with the order.
  - **DWH_Entry_Date**: The entry date for the order into the DWH.

3. DimProduct

- **Purpose**: This table stores product-related data used for sales and inventory analysis.

- **Columns**:
  - **DWH_Product_SK (PK)**: Surrogate key for products.
  - **Product_BK**: Product business key.
  - **Product_ID**: Identifier for the product.
  - **Product_Name**: Name of the product.
  - **Brand**: The brand of the product.
  - **Category_Name**: The main category for the product.
  - **Subcategory_Name**: The subcategory within the category.
  - **AVG_Buy_Price**: The average buying price for the product.

- **Selling_Price**: Selling price of the product.
- **Start_Date**: Date when the product was introduced.
- **End_Date**: Date when the product was discontinued.
- **Is_Current**: Flag indicating if the product is current (active).
- **DWH_Entry_Date**: The date the product data was entered into the DWH.

4. DimVendor

- **Purpose**: This dimension provides information about the vendors supplying products.
- **Columns**:
  - **DWH_Vendor_SK (PK)**: Surrogate key for vendors.
  - **Vendor_BK**: Business key of the vendor.
  - **Vendor_ID**: Vendor's ID from the source system.
  - **Vendor_Name**: Vendor's name.
  - **City**: Vendor's city.
  - **DWH_Entry_Date**: Date the vendor information was loaded into the DWH.

5. DimInventory

- **Purpose**: This table captures information about the product inventory.

- **Columns**:
  - **DWH_Inventory_SK (PK)**: Surrogate key for inventory.
  - **Inventory_BK**: Business key for inventory records.
  - **Inventory_ID**: Inventory ID from the transactional system.
  - **Inventory_Name**: Name or description of the inventory.
  - **Reorder_Level**: Level at which the product needs to be reordered.
  - **DWH_Entry_Date**: The entry date of the inventory record.

6. DimDate

- **Purpose**: A centralized table for time-based analysis, providing a hierarchy of date-related attributes.
- **Columns**:
  - **DWH_Date_SK (PK)**: Surrogate key for date.
  - **Date**: The actual calendar date.
  - **Day**: Day of the month.
  - **Day_Name**: Name of the day (e.g., Monday).
  - **Week**: Week number.

- **Month**: Month number.
- **Month_Name**: Full name of the month.
- **Quarter**: The quarter of the year (1 to 4).
- **Year**: Year of the date.
- **Season**: Season corresponding to the date (e.g., Summer, Winter).

Fact Tables

1. FactSales

- **Purpose**: This table stores data about sales transactions, including product sales, customer purchases, and order information.

- **Grainularity**:

  Transaction level: The granularity is at the individual **sales transaction** level. Each row corresponds to:

  - A single product sold.
  - A single order.
  - A specific customer.
  - On a specific date.

- **Columns**:
  - **DWH_Sales_SK (PK)**: Surrogate key for sales transactions.
  - **DWH_Date_SK**: Foreign key linking to the Date dimension.
  - **DWH_Product_SK**: Foreign key linking to the Product dimension.
  - **DWH_Order_SK**: Foreign key linking to the Order dimension.
  - **DWH_Customer_SK**: Foreign key linking to the Customer dimension.
  - **DWH_Inventory_SK**: Foreign key linking to the Inventory dimension.
  - **Quantity**: The number of units sold in the transaction.
  - **Total_Price**: The total price of the sale transaction.
  - **Rating**: Customer rating for the product.
  - **DWH_Entry_Date**: Entry date of the sales transaction.

2. FactSupply

- **Purpose**: This table tracks supply transactions, recording when products are delivered by vendors to inventory.

- **Grianularity**:

  Supply Transaction level: The granularity is at the individual **supply delivery** level. Each row corresponds to:

  - A specific product supplied.
  - A specific vendor.
  - A specific inventory location.
  - With particular quantities and costs.
  - On specific request, expected, and delivery dates.

- **Columns**:
  - **DWH_Supply_SK (PK)**: Surrogate key for supply transactions.
  - **Request_Date_SK**: Foreign key linking to the request date in the Date dimension.
  - **Expected_Date_SK**: Foreign key linking to the expected delivery date in the Date dimension.
  - **Delivery_Date_SK**: Foreign key linking to the actual delivery date in the Date dimension.
  - **DWH_Product_SK**: Foreign key linking to the Product dimension.
  - **DWH_Inventory_SK**: Foreign key linking to the Inventory dimension.
  - **DWH_Vendor_SK**: Foreign key linking to the Vendor dimension.
  - **Quantity**: The quantity of items supplied.
  - **Buy_Price**: The price at which the product was bought.
  - **Total_Cost**: Total cost of the supplied products.
  - **Defective_Quantity**: The number of defective items in the supply batch.
  - **DWH_Entry_Date**: The date the supply transaction was recorded.

Mapping Sheet

1. Change Tracker

- **Purpose**: Track modifications in the data mapping and database definitions.

2. Mapping Sheet Analysis

The mapping sheet provides a comprehensive view of how source data is transformed and loaded into the target data warehouse. Key components include:

- Change ID: Tracks modifications to the mapping
- Target information: Schema, table, column, and data type
- Stage database information: Schema, table, and column
- Landing database information: Schema, source table, and column
- Source database information: Schema, table, column, and data type
- Transformation logic and comments
- Table definitions and package names
- Source query definitions and DDL statements

A	B	C	D	E	F	G
Change ID	Target Table Type	Target Schema	Target Table	Target Column	Target Column Data Type	Stage DB Schema
1	Dimension	Sales	DimOrder	DWH_Order_SK	Int	Jadara_Stagging
1	Dimension	Sales	DimOrder	Order_BK	Int	Jadara_Stagging
1	Dimension	Sales	DimOrder	Order_ID	NVarChar(50)	Jadara_Stagging
1	Dimension	Sales	DimOrder	Order_Status	NVarChar(50)	Jadara_Stagging
1	Dimension	Sales	DimOrder	Order_Type	NVarChar(50)	Jadara_Stagging
1	Dimension	Sales	DimOrder	Shipping_Address	NVarChar(200)	Jadara_Stagging

Change Tracker | Mapping Sheet | DDL | Data Warehouse Matrix

Example row analysis (DimOrder):

- Target: Dimension table "DimOrder" in DWH schema
- Key column: DWH_Order_SK (Int type)
- Source: Jadara_Stagging.Order_.SK
- Transformation: Selects current records (Is_Current_Flag = 1)
- Additional attributes: Order_ID, Status, Source, Shipping_Address, Hashcode

The mapping covers both fact and dimension tables, ensuring traceability from source to target for all relevant attributes.

3. DDL Sheet Analysis

The DDL sheet defines the structure of target tables in the data warehouse. Key components include:

- Target table type (Dimension/Fact)
- Table and column names
- Data types
- Primary key and index information
- Descriptions
- DDL statements for table creation
- Foreign key relationships
- Extended properties for documentation

A	B	C	D	E	F	G	H	
Target Table Type	Target Table	Target Column	Target Column Data Type	Primary Key	Index	Description	DDL Statement	Description DDL
Dimension	DimOrder	DWH_Order_SK	Int	Y		This is an auto incremental column that acts as a	CREATE TABLE Shared.DimOrder (	EXEC sys.sp_addextende
Dimension	DimOrder	Order_BK	Int			This is an auto incremental column	Alter TABLE Shared.DimOrder add Order_BK Int;	EXEC sys.sp_addextende
Dimension	DimOrder	Order_ID	NVarChar(50)			This column is the unique identifier for the	Alter TABLE Shared.DimOrder add Order_ID	EXEC sys.sp_addextende
Dimension	DimOrder	Order_Status	NVarChar(50)			This column is the order status of the order if it	Alter TABLE Shared.DimOrder add Order_Status	EXEC sys.sp_addextende
Dimension	DimOrder	Order_Type	NVarChar(50)			this column is the order type as if it is online or	Alter TABLE Shared.DimOrder add Order_Type	EXEC sys.sp_addextende
Dimension	DimOrder	Shipping_Address	NVarChar(50)			this column is for the shipping address of the	Alter TABLE Shared.DimOrder add	EXEC sys.sp_addextende
Dimension	DimOrder	Hash_Code	NVarChar(100)			This column stores the hash codes of the table	Alter TABLE Shared.DimOrder add Hash_Code	EXEC sys.sp_addextende
Dimension	DimOrder	DWH_Entry_Date	Datetime			This column stores the date and the time when	Alter TABLE Shared.DimOrder add	EXEC sys.sp_addextende
Dimension	DimCustomer	DWH_Customer_SK	Int	Y		This is an auto incremental column that acts as a	CREATE TABLE Shared.DimCustomer (	EXEC sys.sp_addextende
Dimension	DimCustomer	Customer_BK	Int			This is an auto incremental column	Alter TABLE Shared.DimCustomer add	EXEC sys.sp_addextende
Dimension	DimCustomer	Customer_ID	Int			This column is the unique identifier for the	Alter TABLE Shared.DimCustomer add	EXEC sys.sp_addextende
Dimension	DimCustomer	Cunstomer_Name	NVarChar(100)			This column stores the FullName of each	Alter TABLE Shared.DimCustomer add	EXEC sys.sp_addextende
Dimension	DimCustomer	City	NVarChar(100)			This column stores the City of each customer	Alter TABLE Shared.DimCustomer add City	EXEC sys.sp_addextende
Dimension	DimCustomer	Gender	NVarChar(10)			This column stores the Gender of each customer	Alter TABLE Shared.DimCustomer add Gender	EXEC sys.sp_addextende
Dimension	DimCustomer	Age	Int			This column stores the Age of each customer	Alter TABLE Shared.DimCustomer add Age Int;	EXEC sys.sp_addextende
Dimension	DimCustomer	Hash_Code	NVarChar(255)			This column stores the hash codes of the table	Alter TABLE Shared.DimCustomer add	EXEC sys.sp_addextende
Dimension	DimCustomer	DWH_Entry_Date	Datetime			This column stores the date and the time when	Alter TABLE Shared.DimCustomer add	EXEC sys.sp_addextende
Dimension	DimDate	DWH_Date_SK	INT	Y		This is an auto incremental column that acts as a	CREATE TABLE Shared.DimDate (DWH_Date_SK	EXEC sys.sp_addextende
Dimension	DimDate	Date_BK	INT			This is an auto incremental column	Alter TABLE Shared.DimDate add Date_BK INT;	EXEC sys.sp_addextende
Dimension	DimDate	Date	DATE			This column stores the dates for the DWH	Alter TABLE Shared.DimDate add Date DATE;	EXEC sys.sp_addextende
Dimension	DimDate	Day	INT			This column stores the days of each date	Alter TABLE Shared.DimDate add Day INT;	EXEC sys.sp_addextende
Dimension	DimDate	Day_Name	NVARCHAR(50)			This column stores the day name of eachday in	Alter TABLE Shared.DimDate add Day_Name	EXEC sys.sp_addextende
Dimension	DimDate	Week	INT			This column stores the week of each date	Alter TABLE Shared.DimDate add Week INT;	EXEC sys.sp_addextende
Dimension	DimDate	Month	INT			This column stores the month of each date	Alter TABLE Shared.DimDate add Month INT;	EXEC sys.sp_addextende
Dimension	DimDate	Month_Name	NVARCHAR(50)			This column stores the month name of each date	Alter TABLE Shared.DimDate add Month_Name	EXEC sys.sp_addextende
Dimension	DimDate	Quarter	INT			This column stores the quarters of each date	Alter TABLE Shared.DimDate add Quarter INT;	EXEC sys.sp_addextende
Dimension	DimDate	Year	INT			This column stores the year of each date	Alter TABLE Shared.DimDate add Year INT;	EXEC sys.sp_addextende
Dimension	DimDate	Season	VARCHAR(50)			This column stores the season of each date	Alter TABLE Shared.DimDate add Season	EXEC sys.sp_addextende
Dimension	DimProduct	DWH_Product_SK	INT	Y		This is an auto incremental column that acts as a	CREATE TABLE Shared.DimProduct (	EXEC sys.sp_addextende
Dimension	DimProduct	Product_BK	INT			This is an auto incremental column	Alter TABLE Shared.DimProduct add Product_BK	EXEC sys.sp_addextende
Dimension	DimProduct	Product_ID	INT			This column is the unique identifier for the	Alter TABLE Shared.DimProduct add Product_ID	EXEC sys.sp_addextende
Dimension	DimProduct	Product_Name	VARCHAR(100)			This column is the Product name for the	Alter TABLE Shared.DimProduct add	EXEC sys.sp_addextende
Dimension	DimProduct	Brand	VARCHAR(100)			This column is the Product brand for the	Alter TABLE Shared.DimProduct add Brand	EXEC sys.sp_addextende
Dimension	DimProduct	Category_Name	VARCHAR(100)			This column is the Product category for the	Alter TABLE Shared.DimProduct add	EXEC sys.sp_addextende
Dimension	DimProduct	Subcategory_Name	VARCHAR(100)			This column is the Product subcategory for the	Alter TABLE Shared.DimPro	
Dimension	DimProduct	AVG_Buy_Price	DECIMAL(10, 2)			This column is the Product avg buy proce from	Alter TABLE Shared.DimPr	

Convert to drop-down chips

Change Tracker | Mapping Sheet | DDL | Data Warehouse Matrix

Example DDL analysis (DimOrder):

- Table: Shared.DimOrder
- Primary Key: DWH_Order_SK (Int, auto-incremental)
- DDL includes extended property for description
- No foreign keys in the dimensions they are in facts only

The DDL sheet ensures proper structure and documentation for each table in the data warehouse.

4. Data Warehouse Matrix Analysis

Fact	Dimensions					
	DimOrder	DimCustomer	DimDate	DimInventory	DimProduct	DimVendor
Sales	✓	✓	✓	✓	✓	
Supply			✓	✓	✓	✓

The matrix provides a high-level overview of the relationships between facts and dimensions:

Facts:

1. Sales
2. Supply

Dimensions:

1. DimOrder
2. DimDate
3. DimInventory
4. DimProduct
5. DimVendor
6. DimCustomer

Relationships:

- Sales fact connects to DimOrder, DimDate, DimCustomer, and DimProduct
- Supply fact connects to all dimensions except DimCustomer

7. Database Testing and Validation

Database Testing and Validation Team: Mohamed Ibrahim, Marwan Amr

To ensure the accuracy and integrity of the data, we developed comprehensive test cases for each database involved in the project:

- ***Physical Database***: Test cases were created to validate the data and ensure the operations were accurately reflected.

- ***Online Database***: Similar to the physical database, we designed test cases to ensure data consistency and accuracy for the online store.

- ***Landing Database:*** Specific test cases were implemented to verify that the landing process captured data accurately and correctly.

- ***Staging Database:*** Test cases were designed to confirm the staging database was correctly processing and storing the data.

- ***Data Warehouse:*** Finally, test cases were used to ensure the integrity and accuracy of the data in the Data Warehouse, which serves as the core of our data analysis.

For each database, SQL queries were written to validate the order of the rows within the tables, ensuring consistency. Additionally, the queries were used to verify the accuracy of the KPI's, Metrics and Measures we aimed to calculate during the analysis phase. This rigorous testing ensured that all data was reliable and ready for accurate analysis.

If any discrepancies or inaccuracies were found during testing, we debugged to pinpoint the specific issues. Once the problem was identified, we promptly reported it to the data engineers or data modelers, who were responsible for addressing and correcting the issue. This iterative debugging and reporting process helped maintain the quality and accuracy of our data in all the stages of the process.

A	B	C	D	E	F	G
ID	Measure	Source 1 SQL (Physical)	Source 1 Result	Source 2 SQL (Online)	Source 2 Result	Source 1 + Source 2 Results
1	Count of Orders	SELECT COUNT(Order_ID) FROM Jadara_Mart_Physical.dbo.Order_	11839	SELECT COUNT(Order_ID) FROM Online_Jadara.dbo.[Order]	80000	91839
2	Count of Customers	SELECT COUNT(*) FROM Jadara_Mart_Physical.dbo.Customer	70000	SELECT COUNT(*) FROM Online_Jadara.dbo.Customer	60000	130000
3	Count of Products	SELECT COUNT(Product_ID) FROM Jadara_Mart_Physical.dbo.Product	293			293
4	Count of Vendors	SELECT COUNT(*) FROM Jadara_Mart_Physical.dbo.Vendor	30			30
5	Count of Inventories	SELECT COUNT(*) FROM Jadara_Mart_Physical.dbo.Inventory	3	SELECT COUNT(*) FROM Online_Jadara.dbo.Inventory	1	4
6	Count of Product_Vendor	SELECT COUNT(*) FROM Jadara_Mart_Physical.dbo.Product_Vendor	8790	SELECT COUNT(*) FROM Online_Jadara.dbo.Supplies	23931	32721
7	Count of Order_Product	SELECT COUNT(*) FROM Jadara_Mart_Physical.dbo.Order_Product	29689	SELECT COUNT(*) FROM order_products	434797	464486
8	Number of Orders per Branch	SELECT b.Branch_Name, COUNT(Order_ID) FROM Jadara_Mart_Physical.dbo.Customer_Order_Branch cob JOIN Branch b ON cob.Branch_ID = b.Branch_ID GROUP BY b.Branch_Name	Alexandria - 3956 Cairo - 3963 Port Said - 3920	SELECT I.Inventory_Name COLLATE Latin1_General_CI_AS AS Name, COUNT(O.Order_ID) AS Number_of_Orders, 'Inventory' AS Source FROM Online_Jadara.dbo.[Order] O JOIN Online_Jadara.dbo.Inventory I ON O.Inventory_id = I.Inventory_ID GROUP BY I.Inventory_Name COLLATE Latin1_General_CI_AS	80000	Online - 80000 Alexandria - 3956 Cairo - 3963 Port Said - 3920
9	Sales per Branch	SELECT b.Branch_Name AS Name, SUM(op.Quantity * p.Unit_Price) AS Total_Sales FROM Jadara_Mart_Physical.dbo.Branch b JOIN Jadara_Mart_Physical.dbo.Customer_Order_Branch cob ON b.B JOIN Jadara_Mart_Physical.dbo.Order_ o ON cob.Order_ID = o.Order JOIN Jadara_Mart_Physical.dbo.Order_Product op ON o.Order_ID = c JOIN Jadara_Mart_Physical.dbo.Product p ON op.Product_ID = p.Pro GROUP BY b.Branch_Name;	Alexandria - 10335270.38 Cairo - 10515186.80 Port Said - 10871703.62	SELECT i.Inventory_ID AS ID, i.Inventory_Name AS Name, SUM(op.Quantity * p.Price) AS Total_Sales FROM `Order` o JOIN order_products op ON o.Order_ID = op.Order_ID JOIN product p ON op.Product_ID = p.Product_ID JOIN inventory i ON o.Inventory_ID = i.Inventory_ID GROUP BY i.Inventory_ID, i.Inventory_Name	620532464.4	Alexandria - 10335270.38 Cairo - 10515186.80 Port Said - 10871703.62 Online - 620532464.4
10	Number of Defective Items	SELECT SUM(Defective_Quantity) FROM Jadara_Mart_Physical.dbo.Product_Inventory	8878	SELECT SUM(defective_quantity) FROM Online_Jadara.dbo.Inventory_Products	3978	12856

A	B	C	D	E	F	G
11	Total Expenses	SELECT SUM(Buy_Price * Quantity) FROM Jadara_Mart_Physical.dbo.Product_Vendor	1116822173	SELECT SUM(Buy_Price * Quantity) FROM supplies	2594188642	3711010815
12	Total Revenue	SELECT SUM(op.Quantity * p.Unit_Price) AS Total_Revenue FROM Jadara_Mart_Physical.dbo.Order_Product op JOIN Jadara_Mart_Physical.dbo.Product p ON op.Product_ID = p.F	31722160.8	SELECT SUM(op.Quantity * p.Price) AS Total_Revenue FROM order_products op JOIN product p ON op.Product_ID = p.Product_ID	620532464.4	652254625.2
13	COGS	WITH CombinedProductData AS (-- Combine physical and online product data SELECT Product_ID, Buy_Price, Quantity FROM Jadara_Mart_Physical.dbo.Product_Vendor UNION ALL SELECT Product_ID, Buy_Price, Quantity FROM Online_Jadara.dbo.Supplies), AvgBuyPriceCombined AS (-- Calculate average buy price across both sources SELECT Product_ID, CONVERT(numeric(10,2), ROUND (AVG(Buy_Price),2,1)) AS AvgBuyPrice FROM CombinedProductData GROUP BY Product_ID), CombinedOrderData AS (-- Combine order product data from both sources SELECT Product_ID, Quantity FROM Jadara_Mart_Physical.dbo.Order_Product UNION ALL SELECT Product_ID, Quantity FROM Online_Jadara.dbo.Order_Products), CombinedCOGS AS (-- Calculate COGS using the combined data SELECT SUM(o.Quantity * a.AvgBuyPrice) AS Total_COGS FROM CombinedOrderData o JOIN AvgBuyPriceCombined a ON o.Product_ID = a.Product_ID) -- Final output of combined COGS SELECT				408559148.4

A	B	C	D	E	F	G
14	Gross Profit	WITH AvgBuyPricePhysical AS (SELECT Product_ID, SUM(Quantity * Buy_Price) / SUM(Quantity) AS AvgBuyPrice FROM Jadara_Mart_Physical.dbo.Product_Vendor GROUP BY Product_ID), PhysicalCOGS AS (SELECT SUM(op.Quantity * abpPhysical.AvgBuyPrice) AS Physical_COGS FROM Jadara_Mart_Physical.dbo.Order_Product op JOIN AvgBuyPricePhysical abpPhysical ON op.Product_ID = abpPhysical.Product_ID), TotalRevenue AS (SELECT SUM(op.Quantity * p.Unit_Price) AS Total_Revenue FROM Jadara_Mart_Physical.dbo.Order_Product op JOIN Jadara_Mart_Physical.dbo.Product p ON op.Product_ID = p.Product_ID) SELECT tr.Total_Revenue - cogs.Physical_COGS AS Gross_Profit FROM TotalRevenue tr, PhysicalCOGS cogs;				243695476.8

H	I	J	K	L	M	N
Landing SQL	**Landing Result**	**Staging SQL**	**Staging Result**	**Data Warehouse SQL**	**Data Warehouse Result**	**Flag**
SELECT COUNT(*) AS Total_Orders FROM Order_;	91839	SELECT COUNT(*) AS Total_Orders FROM Order_ WHERE Is_Current_Flag = 1;	91839	SELECT COUNT(Order_ID) AS NumberOfOrders FROM Sales.DimOrder;	91839	Pass
SELECT COUNT(*) AS Number_Of_Customers FROM Customer;	130000	SELECT COUNT(*) AS Number_Of_Customers FROM Customer WHERE Is_Current_Flag = 1;	130000	SELECT COUNT(Customer_ID) AS NumberOfCustomers FROM Sales.DimCustomer;	130000	Pass
SELECT COUNT(*) AS NumberOfProducts FROM Product;	293	SELECT COUNT(Product_ID) AS NumberOfProducts FROM Product WHERE Is_Current_Flag = 1;	293	SELECT COUNT(Product_ID) FROM Shared.DimProduct	293	Pass
SELECT COUNT(*) FROM Unified_RetailStore.dbo.Vendor	30	SELECT COUNT(*) FROM staging.dbo.Vendor	30	SELECT COUNT(*) FROM Supply.DimVendor	30	Pass
SELECT COUNT(*) FROM Unified_RetailStore.dbo.Inventory	4	Select COUNT(*) FROM staging.dbo.Inventory where Is_Current_Flag = 1	4	SELECT COUNT(*) FROM Shared.DimInventory	4	Pass
SELECT COUNT(*) FROM Unified_RetailStore.dbo.Product_Vendor	32721	SELECT COUNT(*) FROM staging.dbo.Product_Vendor	32721	SELECT COUNT(*) FROM DWH.Supply.FactSupply	32721	Pass
SELECT COUNT(*) FROM Order_Product	464486	SELECT COUNT(*) FROM Order_Product WHERE Is_Current_Flag = 1	464486	SELECT COUNT(*) FROM DWH.Sales.FactSales	464486	Pass
SELECT i.Inventory_ID, i.Branch_ID, i.Inventory_Name, COUNT(o.Order_ID) AS NumberOfOrders FROM Order_ o JOIN Inventory i ON o.Inventory_ID = i.Inventory_ID OR i.Branch_ID = o. GROUP BY i.Inventory_ID,i.Branch_ID, i.Inventory_Name;	Online - 80000 Alexandria - 3956 Cairo - 3963 Port Said - 3920	SELECT i.Inventory_ID, i.Inventory_Name, COUNT(o.Order_ID) AS NumberOfOrders FROM Order_ o JOIN Inventory i ON o.Inventory_ID = i.Inventory_ID WHERE o.Is_Current_Flag = 1 AND i.Is_Current_Flag = 1 GROUP BY i.Inventory_ID, i.Inventory_Name	Online - 80000 Alexandria - 3956 Cairo - 3963 Port Said - 3920	SELECT I.Inventory_ID, I.Inventory_Name, COUNT(DISTINCT F.Order_SK) AS NumberOfOrders FROM Shared.DimInventory I JOIN Sales.FactSales F ON I.Inventory_SK = F.Inventory_SK GROUP BY I.Inventory_ID, I.Inventory_Name;	Online - 80000 Alexandria - 3956 Cairo - 3963 Port Said - 3920	Pass
SELECT i.Inventory_ID, i.Inventory_Name, SUM(op.Quantity * p.Price) AS Total_Sales FROM Order_ o JOIN Order_Product op ON o.Order_ID = op.Order_ID JOIN Product p ON op.Product_ID = p.Product_ID JOIN Inventory i ON o.Inventory_ID = i.Inventory_ID GROUP BY i.Inventory_ID, i.Inventory_Name;	Alexandria - 10335270.38 Cairo - 10515186.80 Port Said - 10871703.62 Online - 620532464.4	SELECT i.Inventory_ID, i.Inventory_Name, SUM(op.Quantity * p.Price) AS Total_Sales FROM Order_ o JOIN Order_Product op ON o.Order_ID = op.Order_ID JOIN Product p ON op.Product_ID = p.Product_ID JOIN Inventory i ON o.Inventory_ID = i.Inventory_ID WHERE o.Is_Current_Flag = 1 AND op.Is_Current_Flag = 1 AND p.Is_Current_Flag = 1 AND i.Is_Current_Flag = 1 GROUP BY i.Inventory_ID, i.Inventory_Name	Alexandria - 10335270.38 Cairo - 10515186.80 Port Said - 10871703.62 Online - 620532464.4	SELECT I.Inventory_ID, I.Inventory_Name, SUM(F.Total_Price) AS TotalSales FROM Shared.DimInventory I LEFT JOIN Sales.FactSales F ON I.Inventory_SK = F.Inventory_SK GROUP BY I.Inventory_ID, I.Inventory_Name;	Alexandria - 10335270.38 Cairo - 10515186.80 Port Said - 10871703.62 Online - 620532464.4	Pass
SELECT SUM(ip.defective_quantity) AS Total_Defective_Items FROM Inventory_Products ip JOIN Product p ON ip.Product_ID = p.Product_ID	12856	SELECT SUM(ip.defective_quantity) AS Total_Defective_Items FROM Inventory_Products ip JOIN Product p ON ip.Product_ID = p.Product_ID WHERE ip.Is_Current_Flag = 1 AND p.Is_Current_Flag = 1;	12856	SELECT SUM(TotalDefectiveItems) FROM (SELECT SUM(DISTINCT Defective_Quantity) AS TotalDefectiveItems FROM DWH.Supply.FactSupply GROUP BY Product_SK) AS num	12856	Pass

H	I	J	K	L	M	N
SELECT SUM(Buy_Price * Quantity) AS Total_Expenses FROM Product_Vendor;	3711010815	SELECT SUM(Buy_Price * Quantity) AS Total_Expenses FROM Product_Vendor WHERE Is_Current_Flag = 1	3711010815	SELECT SUM(Total_Cost) AS TotalCost FROM Supply.FactSupply;	3711010815	Pass
SELECT SUM(op.Quantity * p.Price) AS Total_Revenue FROM Order_Product op JOIN Product p ON op.Product_ID = p.Product_ID;	652254625.2	SELECT SUM(op.Quantity * p.Price) AS Total_Revenue FROM Order_Product op JOIN Product p ON op.Product_ID = p.Product_ID WHERE op.Is_Current_Flag = 1 AND p.Is_Current_Flag = 1	652254625.2	SELECT SUM(Total_Price) FROM DWH.Sales.FactSales	652254625.2	Pass
WITH AvgBuyPrice AS (SELECT Product_ID, CONVERT(numeric(10,2), ROUND (AVG(Buy_Price),2,1)) AS FROM Product_Vendor GROUP BY Product_ID) SELECT SUM(op.Quantity * abp.AvgBuyPrice) AS Total_COGS FROM Order_Product op JOIN AvgBuyPrice abp ON op.Product_ID = abp.Product_ID;	408559148.4	WITH AvgBuyPrice AS (SELECT Product_ID, CONVERT(numeric(10,2), ROUND (AVG(Buy_Price),2,1))AS AvgBuyPrice FROM Product_Vendor WHERE Is_Current_Flag = 1 GROUP BY Product_ID) SELECT SUM(op.Quantity * abp.AvgBuyPrice) AS Total_COGS FROM Order_Product op JOIN AvgBuyPrice abp ON op.Product_ID = abp.Product_ID WHERE op.Is_Current_Flag = 1	408559148.4	WITH SalesData AS (-- Combine sales data with product average buy price SELECT f.Date_SK, f.Product_SK, f.Quantity, f.Total_Price, p.AVG_Buy_Price FROM Sales.FactSales f JOIN Shared.DimProduct p ON f.Product_SK = p.Product_SK), COGS_Calculation AS (-- Calculate COGS based on quantity sold and average buy price SELECT SUM(s.Quantity * s.AVG_Buy_Price) AS Total_COGS FROM SalesData s) -- Output total COGS SELECT Total_COGS FROM COGS_Calculation;	408559148.4	Pass

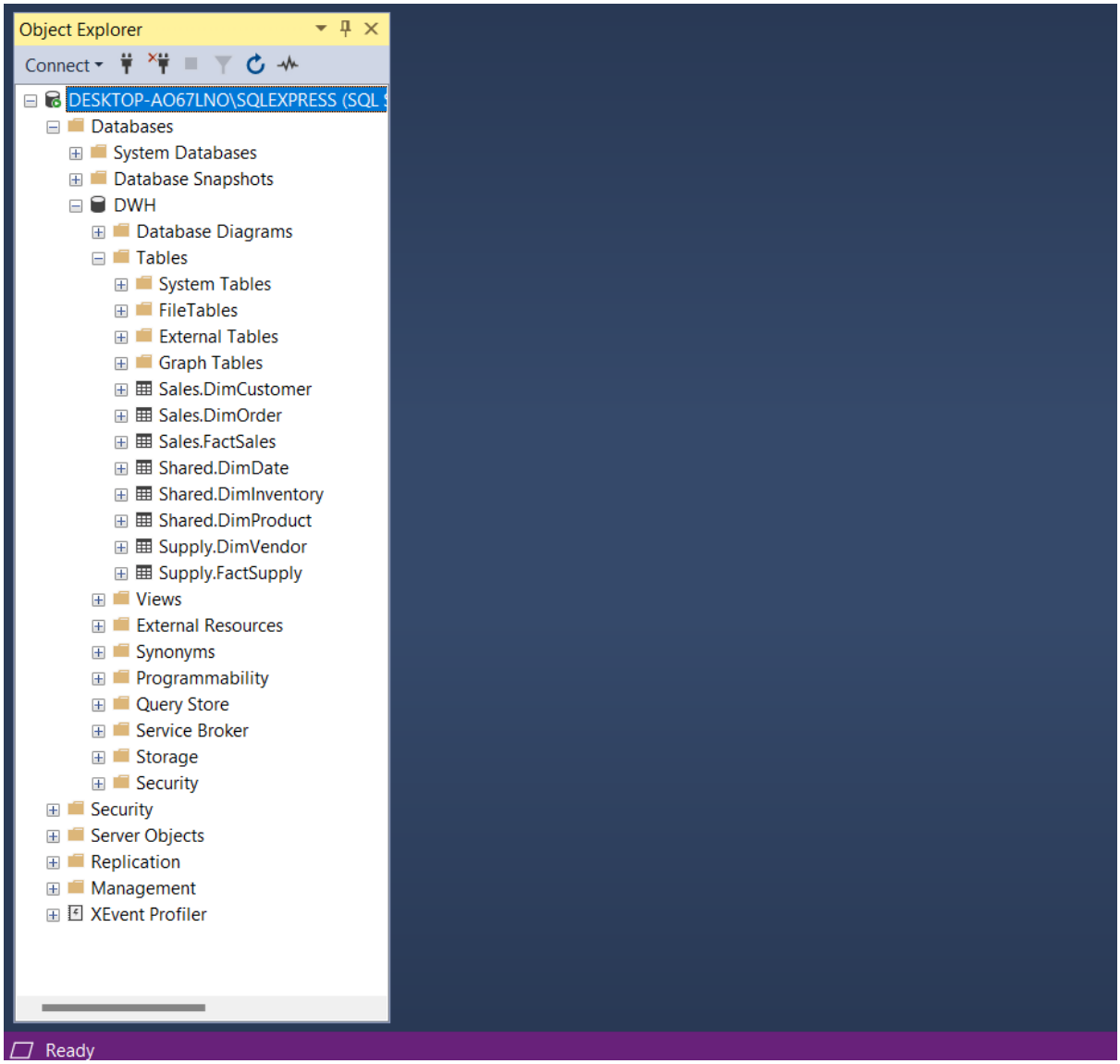
H	I	J	K	L	M	N
WITH Total_Revenue_CTE AS (-- Calculate Total Revenue SELECT SUM(op.Quantity * p.Price) AS Total_Revenue FROM Order_Product op JOIN Product p ON op.Product_ID = p.Product_ID), AvgBuyPrice_CTE AS (-- Calculate Average Buy Price per Product SELECT Product_ID, CONVERT(numeric(10,2), ROUND(AVG(Buy_Price), 2, 1)) AS FROM Product_Vendor GROUP BY Product_ID), Total_COGS_CTE AS (-- Calculate Total COGS SELECT SUM(op.Quantity * abp.AvgBuyPrice) AS Total_COGS FROM Order_Product op JOIN AvgBuyPrice_CTE abp ON op.Product_ID = abp.Product_ID) -- Calculate Gross Profit SELECT tr.Total_Revenue - tc.Total_COGS AS Gross_Profit FROM Total_Revenue_CTE tr CROSS JOIN Total_COGS_CTE tc;	243695476.8	WITH Total_Revenue_CTE AS (-- Calculate Total Revenue SELECT SUM(op.Quantity * p.Price) AS Total_Revenue FROM Order_Product op JOIN Product p ON op.Product_ID = p.Product_ID WHERE op.Is_Current_Flag = 1 AND p.Is_Current_Flag = 1), AvgBuyPrice_CTE AS (-- Calculate Average Buy Price per Product SELECT Product_ID, CONVERT(numeric(10,2), ROUND(AVG(Buy_Price), 2, 1)) AS AvgBuyPrice FROM Product_Vendor WHERE Is_Current_Flag = 1 GROUP BY Product_ID), Total_COGS_CTE AS (-- Calculate Total COGS SELECT SUM(op.Quantity * abp.AvgBuyPrice) AS Total_COGS FROM Order_Product op JOIN AvgBuyPrice_CTE abp ON op.Product_ID = abp.Product_ID WHERE op.Is_Current_Flag = 1)	243695476.8	WITH Total_Sales_CTE AS (-- Calculate Total Sales Price SELECT SUM(Total_Price) AS Total_Sales_Price FROM DWH.Sales.FactSales), SalesData_CTE AS (-- Combine sales data with product average buy price SELECT f.Date_SK, f.Product_SK, f.Quantity, f.Total_Price, p.AVG_Buy_Price FROM Sales.FactSales f JOIN Shared.DimProduct p ON f.Product_SK = p.Product_SK), COGS_Calculation_CTE AS (-- Calculate COGS based on quantity sold and average buy price SELECT SUM(s.Quantity * s.AVG_Buy_Price) AS Total_COGS FROM SalesData_CTE s) -- Calculate and select Gross Profit only SELECT ts.Total_Sales_Price - c.Total_COGS AS Gross_Profit FROM Total_Sales_CTE ts CROSS JOIN	243695476.8	Pass

8. Semantic

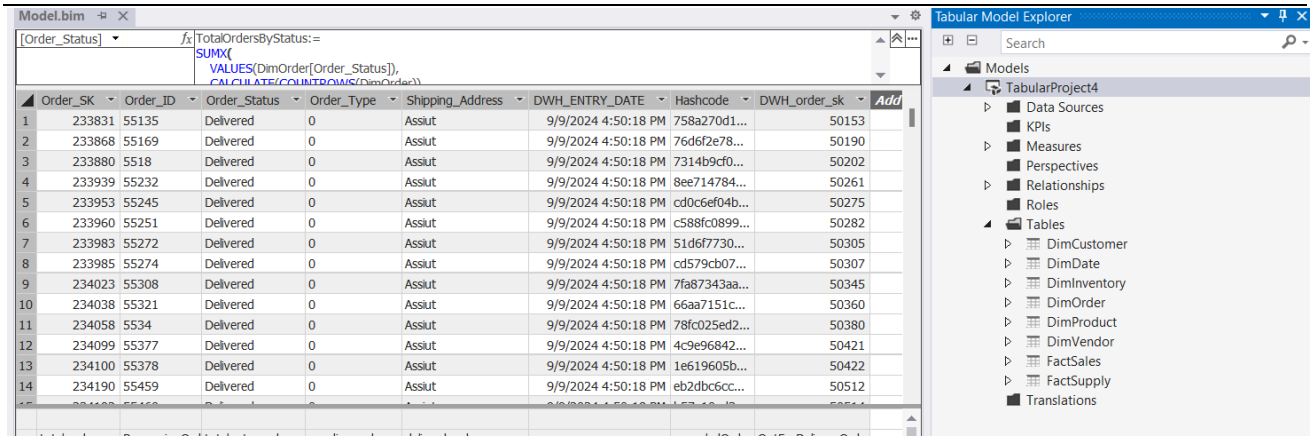
Semantic: Salma Fathy, Marwan Amr

** Still working on it

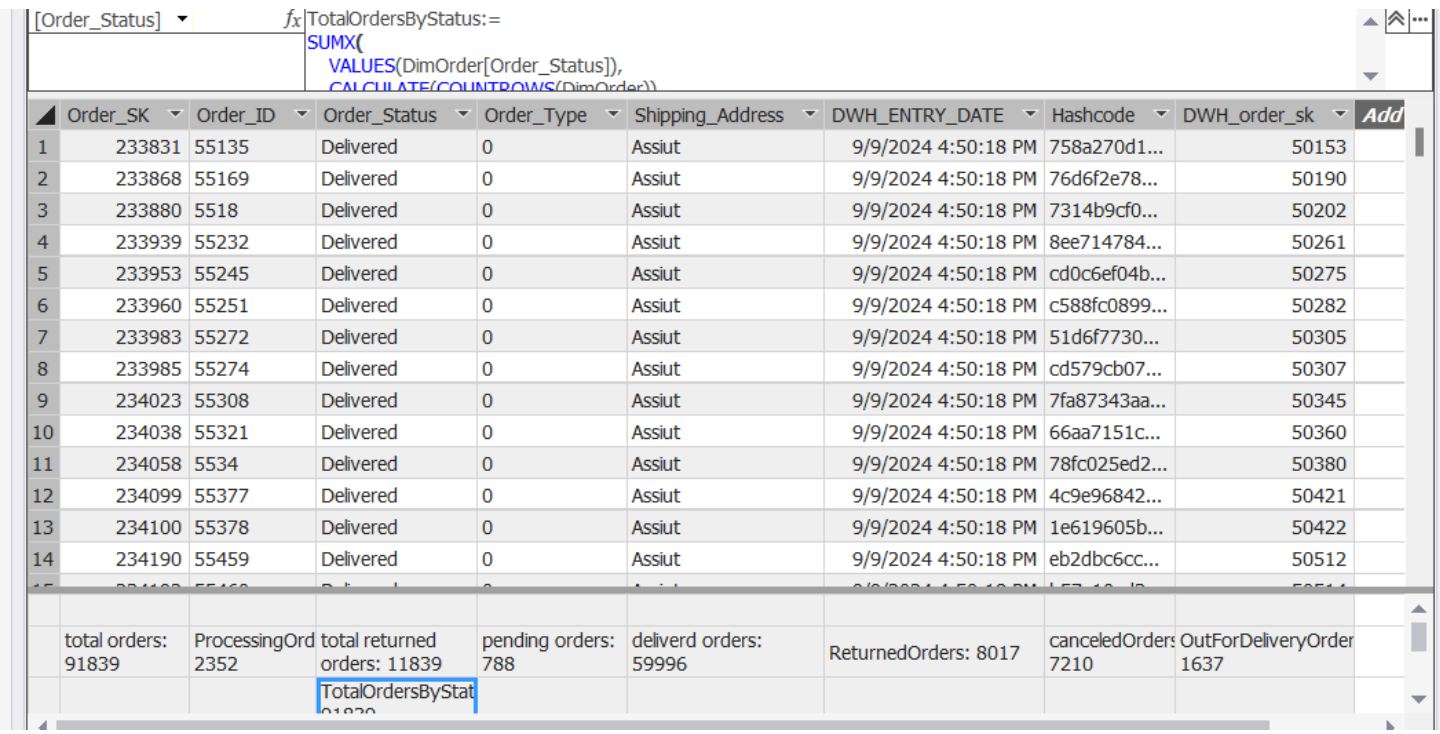
1. Restore the Backup of datawarhouse



2. Import from data source for SSAS project



3. Make measures for DashBoard

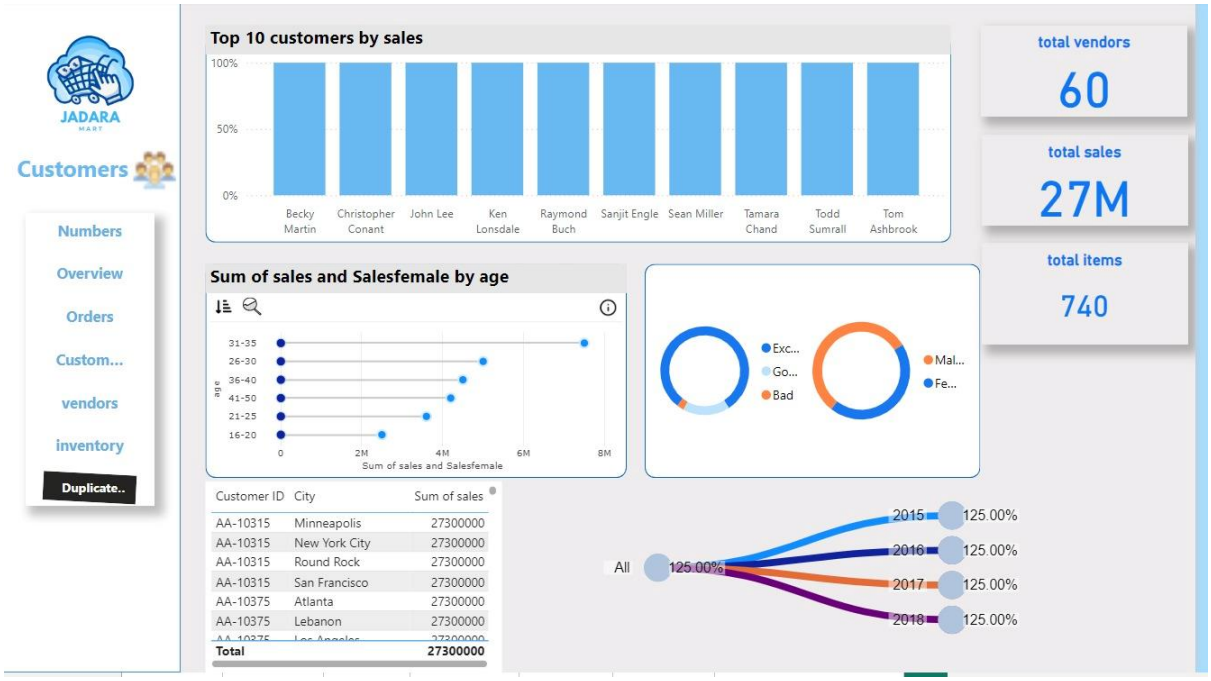


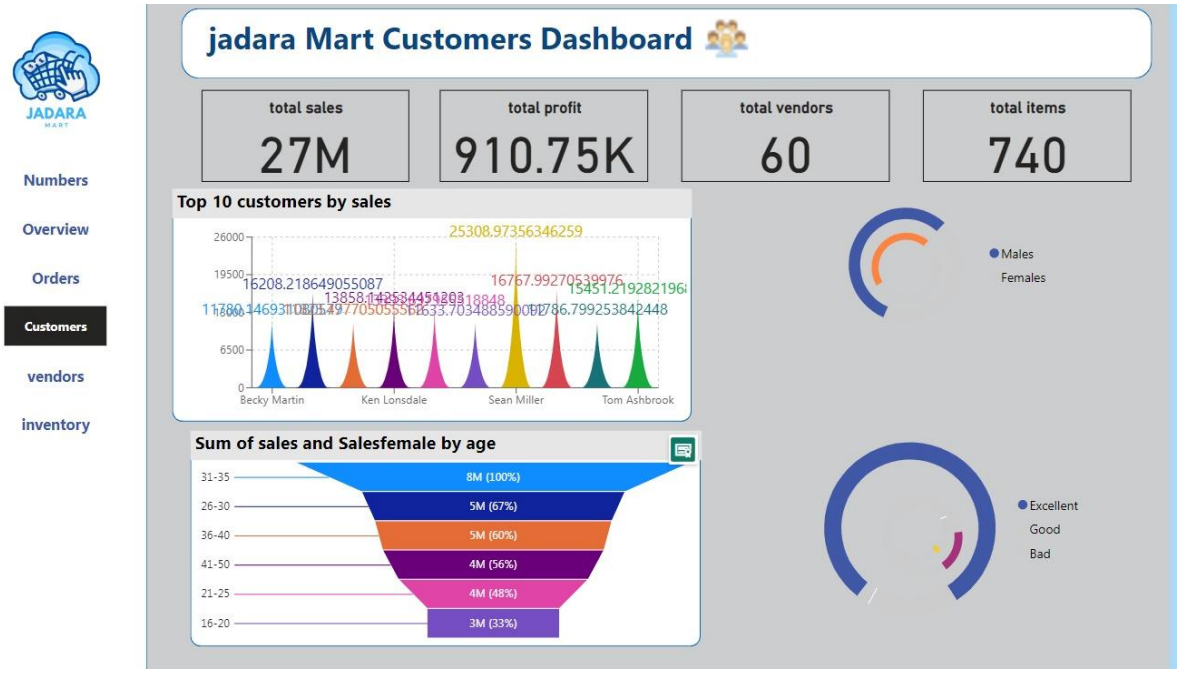
9. Data Analysis : BI

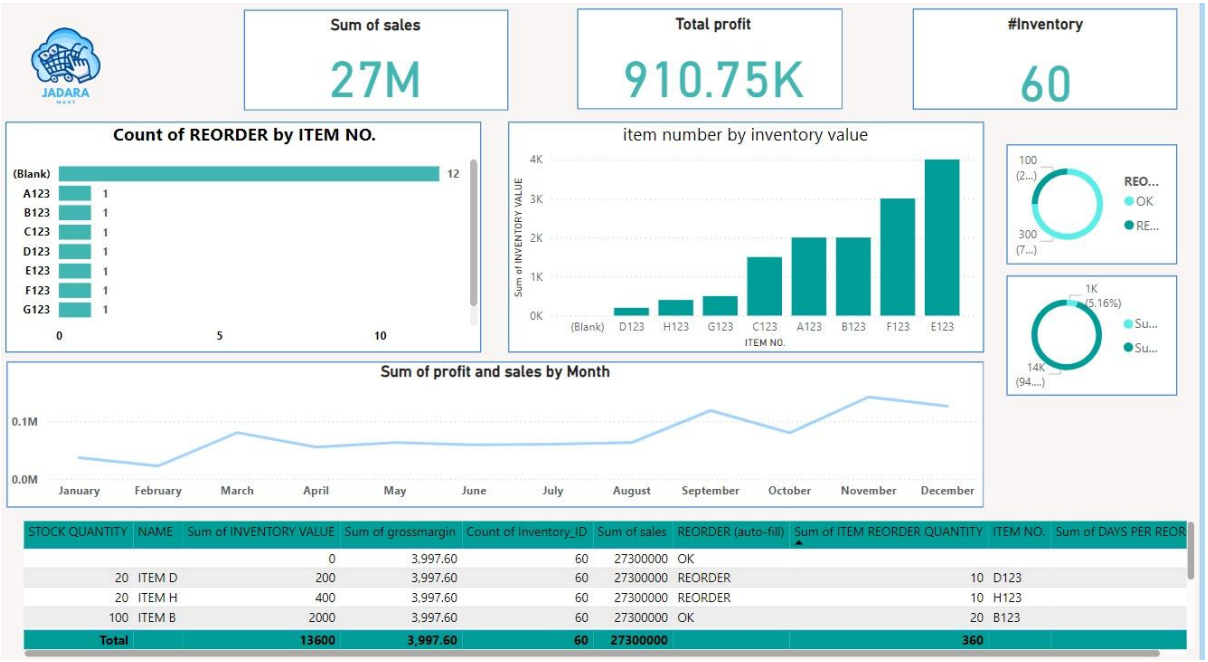
BI Team: Salma Fathy

We created several versions of the dashboard and discussed them with the team to reach the final form and we agreed on it.

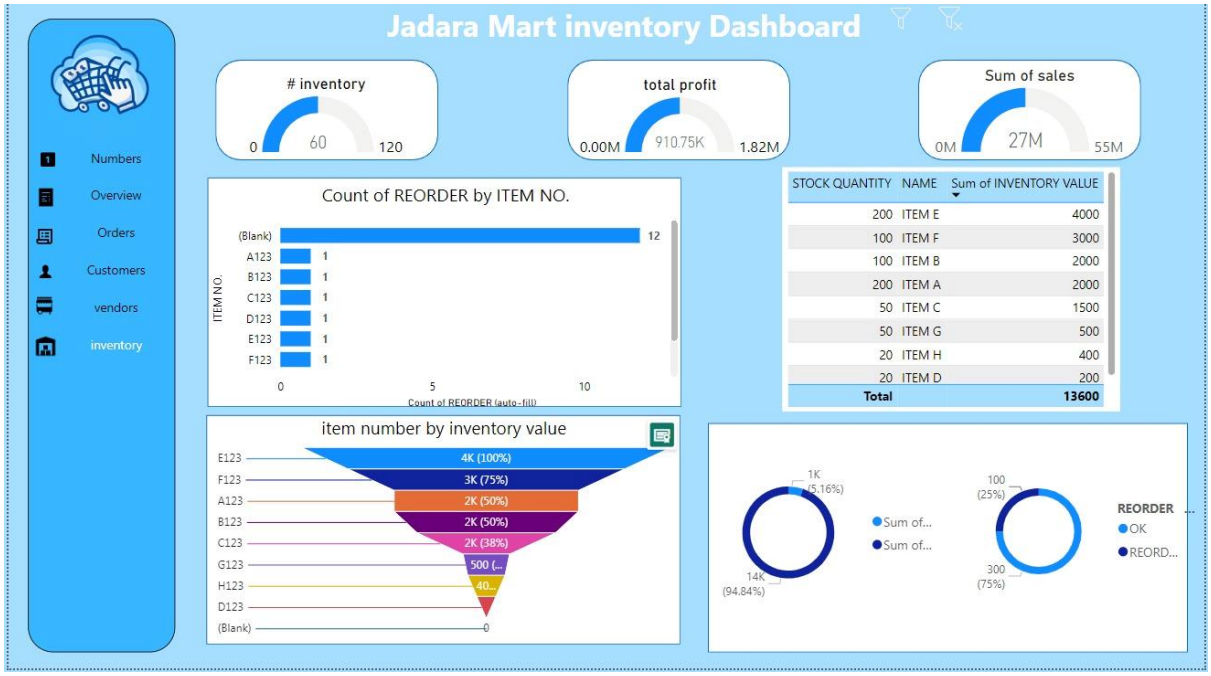
First dashboards for VRD

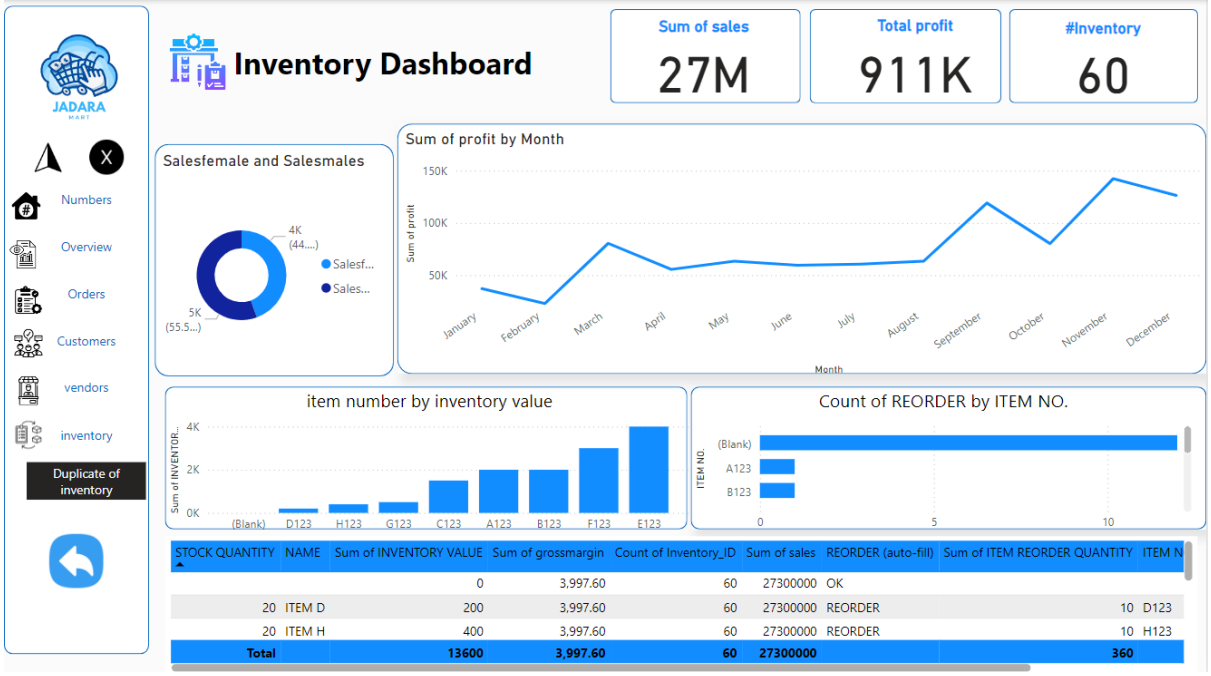




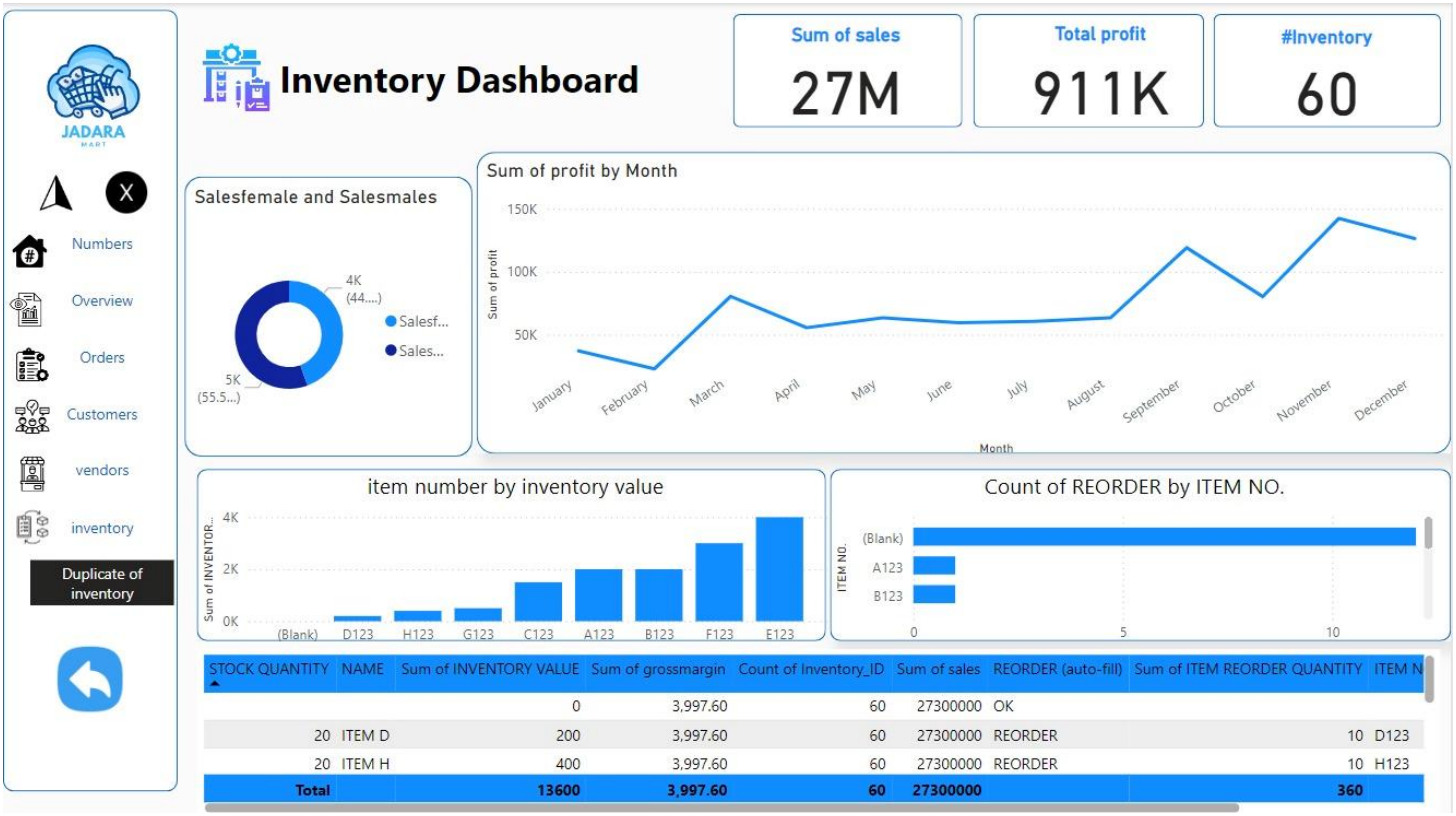


STOCK QUANTITY	NAME	Sum of INVENTORY VALUE	Sum of grossmargin	Count of Inventory_ID	Sum of sales	REORDER (auto-fill)	Sum of ITEM REORDER QUANTITY	ITEM NO.	Sum of DAYS PER REOR
		0	3,997.60	60	27300000	OK			
20	ITEM D	200	3,997.60	60	27300000	REORDER	10	D123	
20	ITEM H	400	3,997.60	60	27300000	REORDER	10	H123	
100	ITEM B	2000	3,997.60	60	27300000	OK	20	B123	
Total		**13600**	**3,997.60**	**60**	**27300000**		**360**		


Jadara Mart inventory Dashboard
Numbers
Overview
Orders
Customers
vendors
inventory
inventory
0
60
120
total profit
0.00M
910.75K
1.82M
Sum of sales
0M
27M
55M
Count of REORDER by ITEM NO.
ITEM NO.
(Blank)
A123
B123
C123
D123
E123
F123
12
1
0
5
10
Count of REORDER (auto-fill)
STOCK QUANTITY NAME Sum of INVENTORY VALUE
200 ITEM E 4000
100 ITEM F 3000
100 ITEM B 2000
200 ITEM A 2000
50 ITEM C 1500
50 ITEM G 500
20 ITEM H 400
20 ITEM D 200
Total 13600
item number by inventory value
E123
F123
A123
B123
C123
G123
H123
D123
(Blank)
4K (100%)
3K (75%)
2K (50%)
2K (50%)
2K (38%)
500 (...
40...
0
1K
(5.16%)
14K
(94.84%)
Sum of...
Sum of...
100
(25%)
300
(75%)
REORDER
OK
REORD...


JADARA
MART
Numbers
Overview
Orders
Customers
vendors
inventory
Duplicate of inventory
Inventory Dashboard
Sum of sales
27M
Total profit
911K
#Inventory
60
Salesfemale and Salesmales
4K
(44....)
5K
(55.5...)
Salesf...
Sales...
Sum of profit by Month
150K
100K
50K
Sum of profit
January
February
March
April
May
June
July
August
September
October
November
December
Month
item number by inventory value
4K
2K
0K
Sum of INVENTOR...
(Blank)
D123
H123
G123
C123
A123
B123
F123
E123
Count of REORDER by ITEM NO.
ITEM NO.
(Blank)
A123
B123
0
5
10
STOCK QUANTITY NAME Sum of INVENTORY VALUE Sum of grossmargin Count of Inventory_ID Sum of sales REORDER (auto-fill) Sum of ITEM REORDER QUANTITY ITEM N
0 3,997.60 60 27300000 OK
20 ITEM D 200 3,997.60 60 27300000 REORDER 10 D123
20 ITEM H 400 3,997.60 60 27300000 REORDER 10 H123
Total 13600 3,997.60 60 27300000 360

The last version for DashBoard



STOCK QUANTITY	NAME	Sum of INVENTORY VALUE	Sum of grossmargin	Count of Inventory_ID	Sum of sales	REORDER (auto-fill)	Sum of ITEM REORDER QUANTITY	ITEM N
		0	3,997.60	60	27300000	OK		
20	ITEM D	200	3,997.60	60	27300000	REORDER	10	D123
20	ITEM H	400	3,997.60	60	27300000	REORDER	10	H123
Total		**13600**	**3,997.60**	**60**	**27300000**		**360**	