Data Engineering Report for JADARA Mart

Supervised By:

Eng. Asmaa Yassin

Eng. Muhammed Sanad

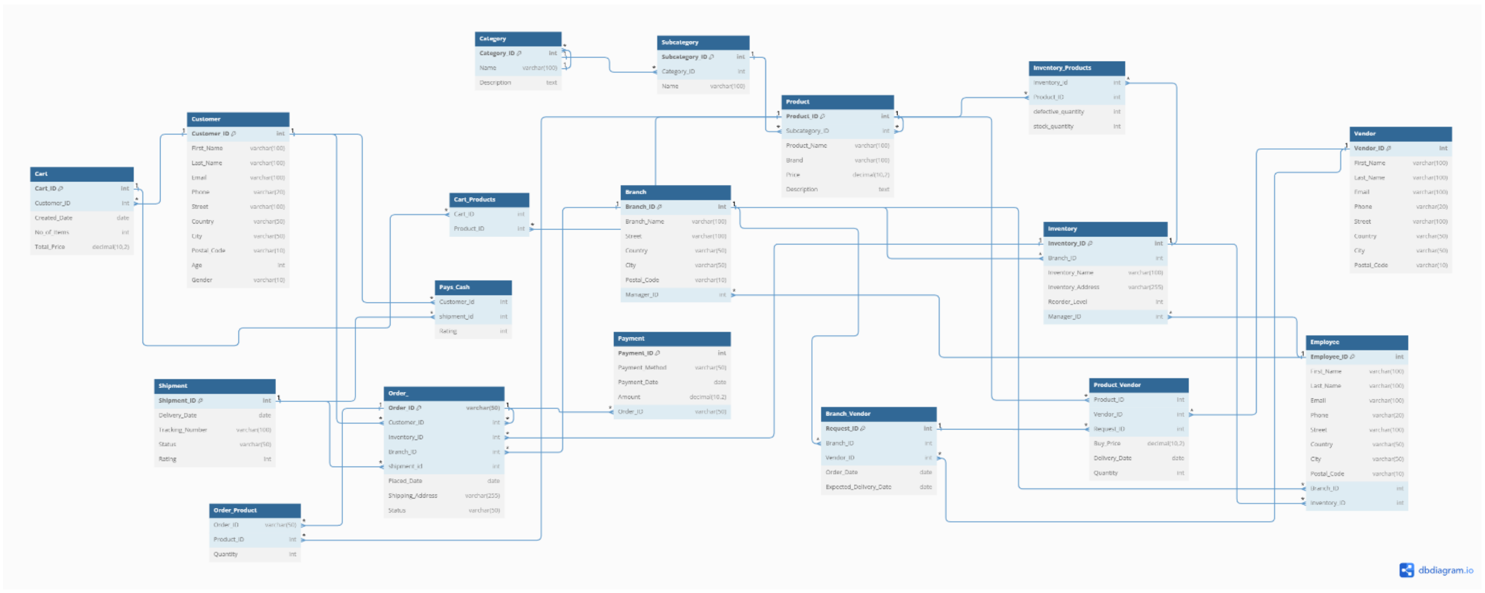
<u>Made by:</u>

Ahmed Said Hammad	**(Data Engineer)**
Mohamed Tamer	**(Data Engineer)**

Data Engineering

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 this process, we encountered a challenge at the destination layer. The two sources had different structures due to variations in their entities and relationships. To address this, we developed a **unified schema** for the destination, designed to accommodate the diverse structures of both source databases



- **Database Name:** Unified_RetailStore

This schema is designed to **standardize data** across multiple source systems.

The inclusion of fields like Hashcode, Source, and Landing_Date in every table allows for:

- **Hashcode:** used for tracking or uniquely identifying records across multiple systems.
- **Source:** Stores metadata about the data's origin, helping to track the source system from which the record was extracted.
- **Landing_Date:** A timestamp for when the data landed in the destination table, crucial for ETL and auditing processes.

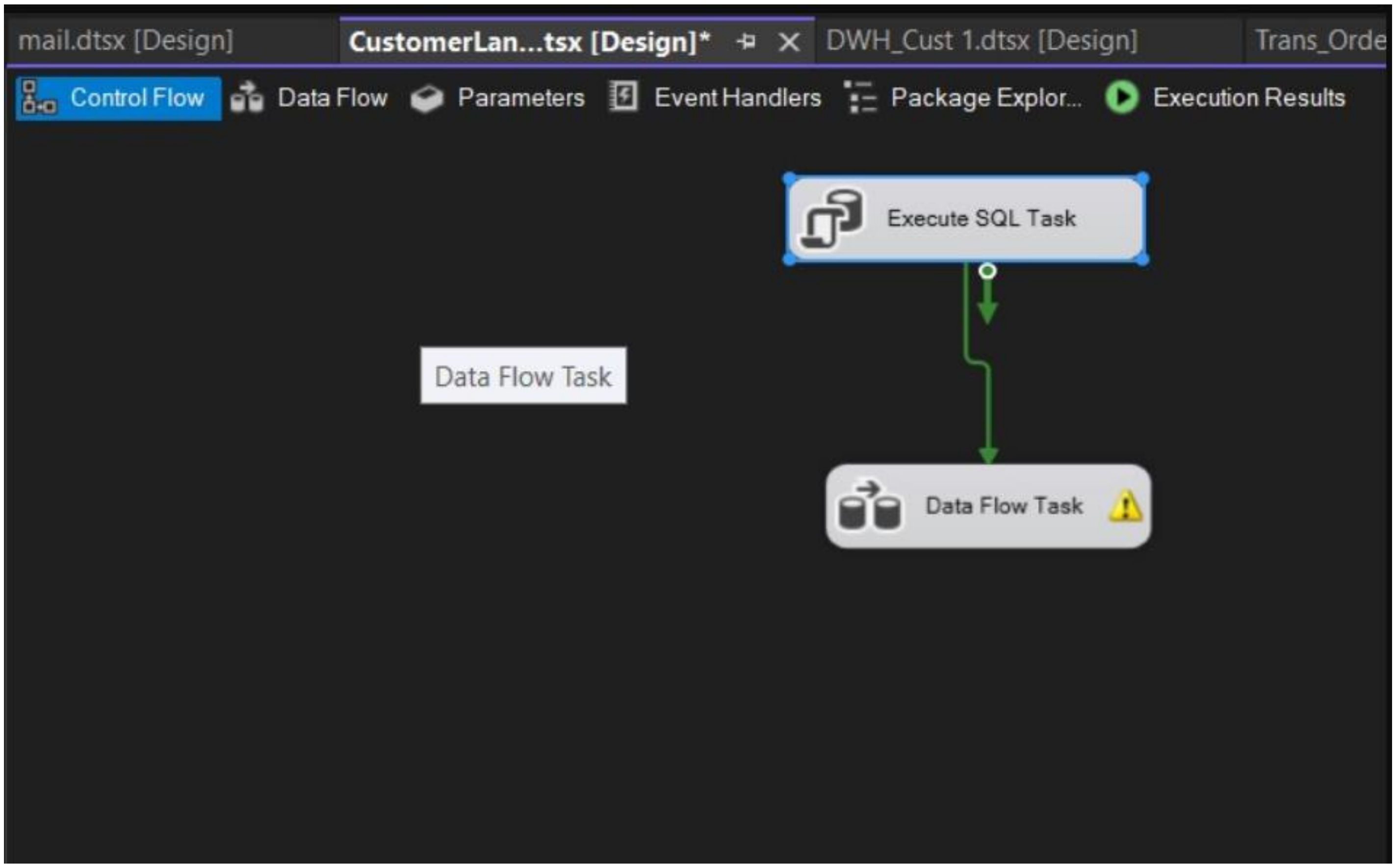
ETL Process for Landing DB

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

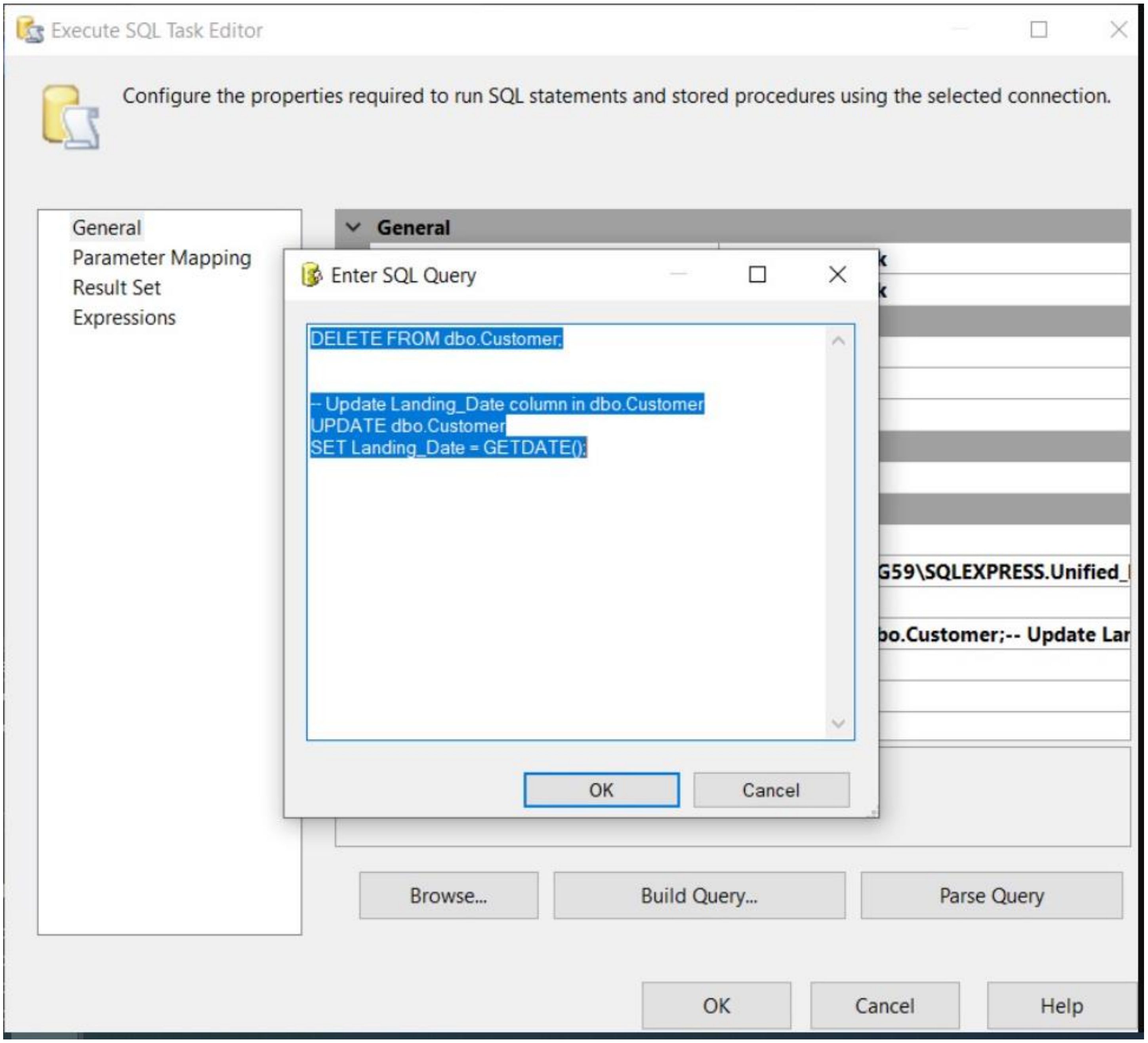
During the initial ETL execution, we encountered a critical issue due to differences in column data types across the two sources, preventing the ETL from running successfully. To resolve this, we used the SSIS (SQL Server Integration Services) Advanced Editor and applied data conversion to align the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



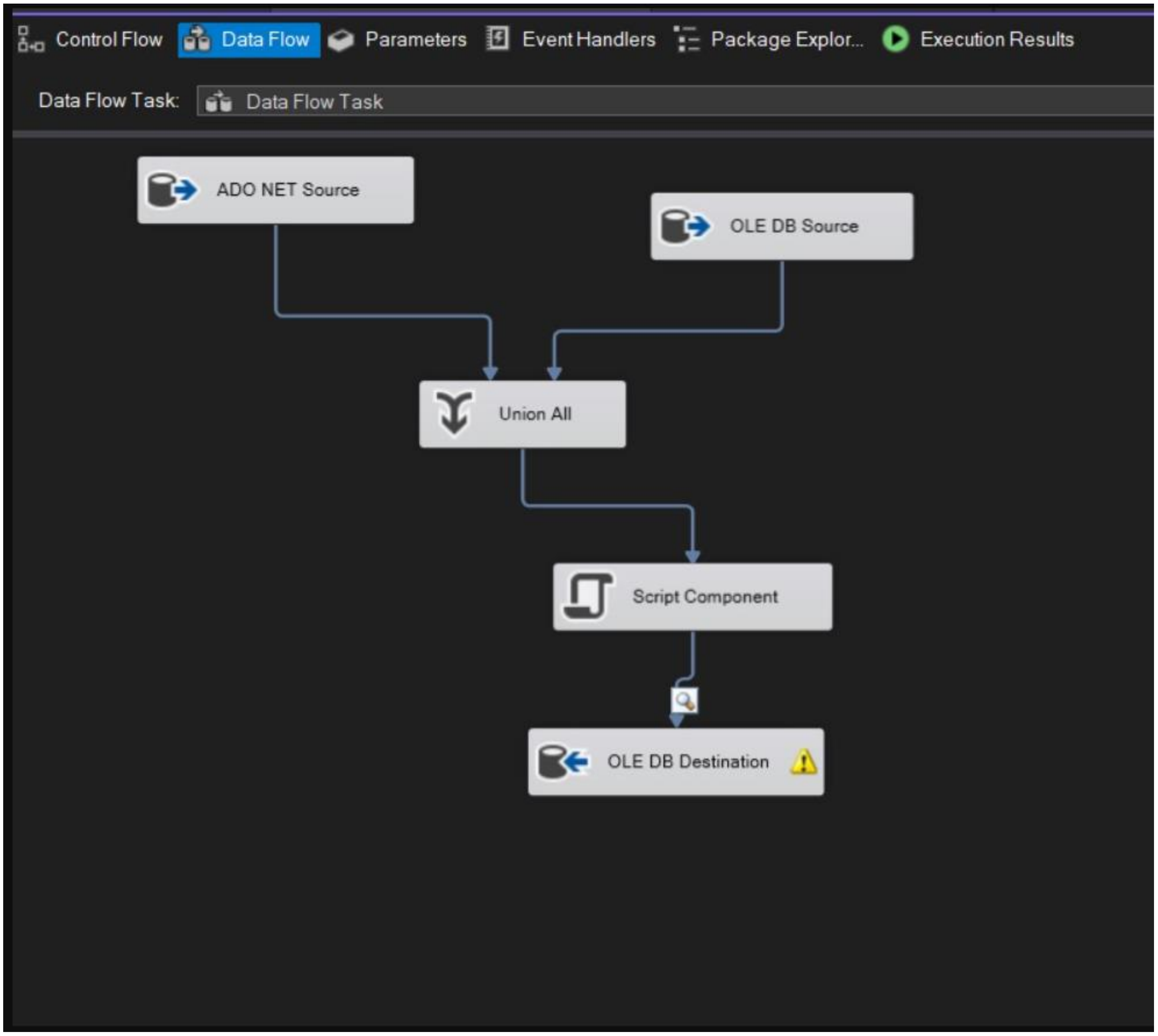
```
DELETE FROM dbo.Customer;


-- Update Landing_Date column in dbo.Customer
UPDATE dbo.Customer
SET Landing_Date = GETDATE();
```

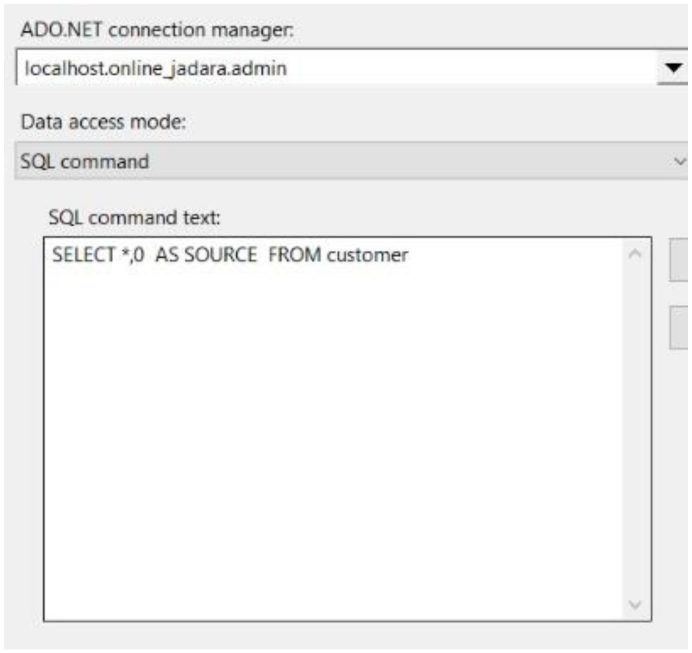
- **Truncate Table:** The process begins by truncating the table before loading, as part of a "truncate and load" strategy, to ensure that all data is refreshed during each run.

- **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.

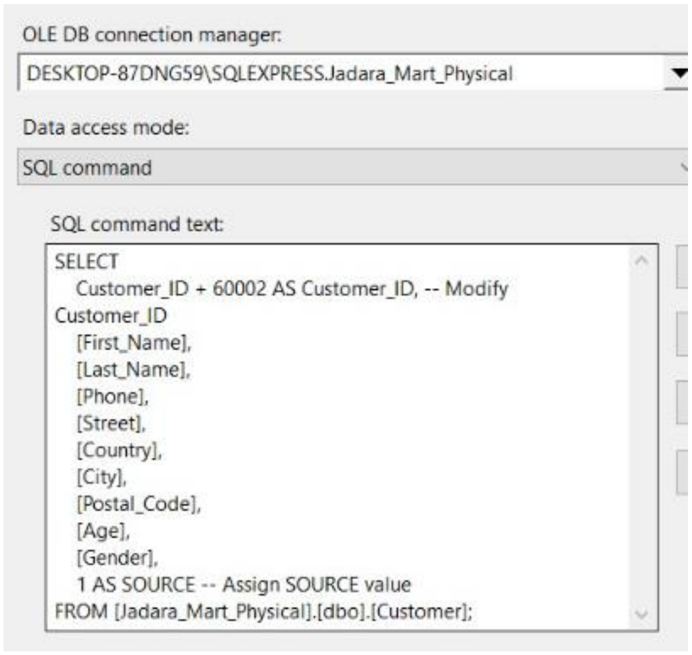


ADO.NET connection manager: localhost.online_jadara.admin

Data access mode: SQL command

SQL command text:

```
SELECT *,0  AS SOURCE  FROM customer
```



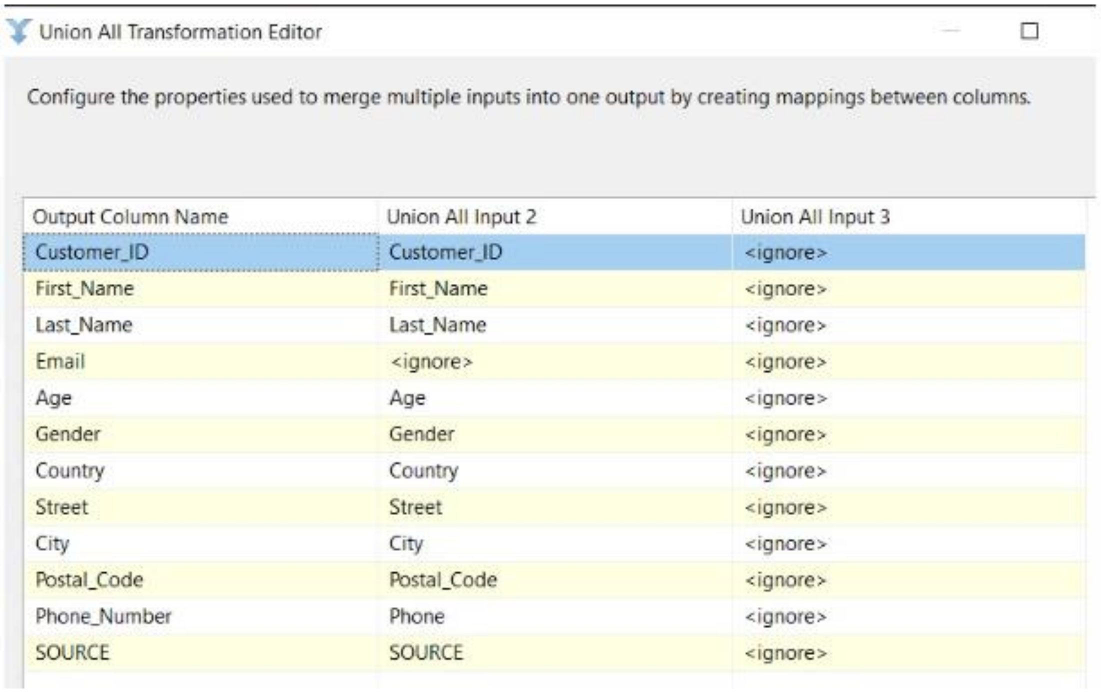
OLE DB connection manager: DESKTOP-87DNG59\SQLEXPRESS.Jadara_Mart_Physical

Data access mode: SQL command

SQL command text:

```
SELECT
   Customer_ID + 60002 AS Customer_ID, -- Modify
Customer_ID
   [First_Name],
   [Last_Name],
   [Phone],
   [Street],
   [Country],
   [City],
   [Postal_Code],
   [Age],
   [Gender],
   1 AS SOURCE -- Assign SOURCE value
FROM [Jadara_Mart_Physical].[dbo].[Customer];
```

- **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.)*

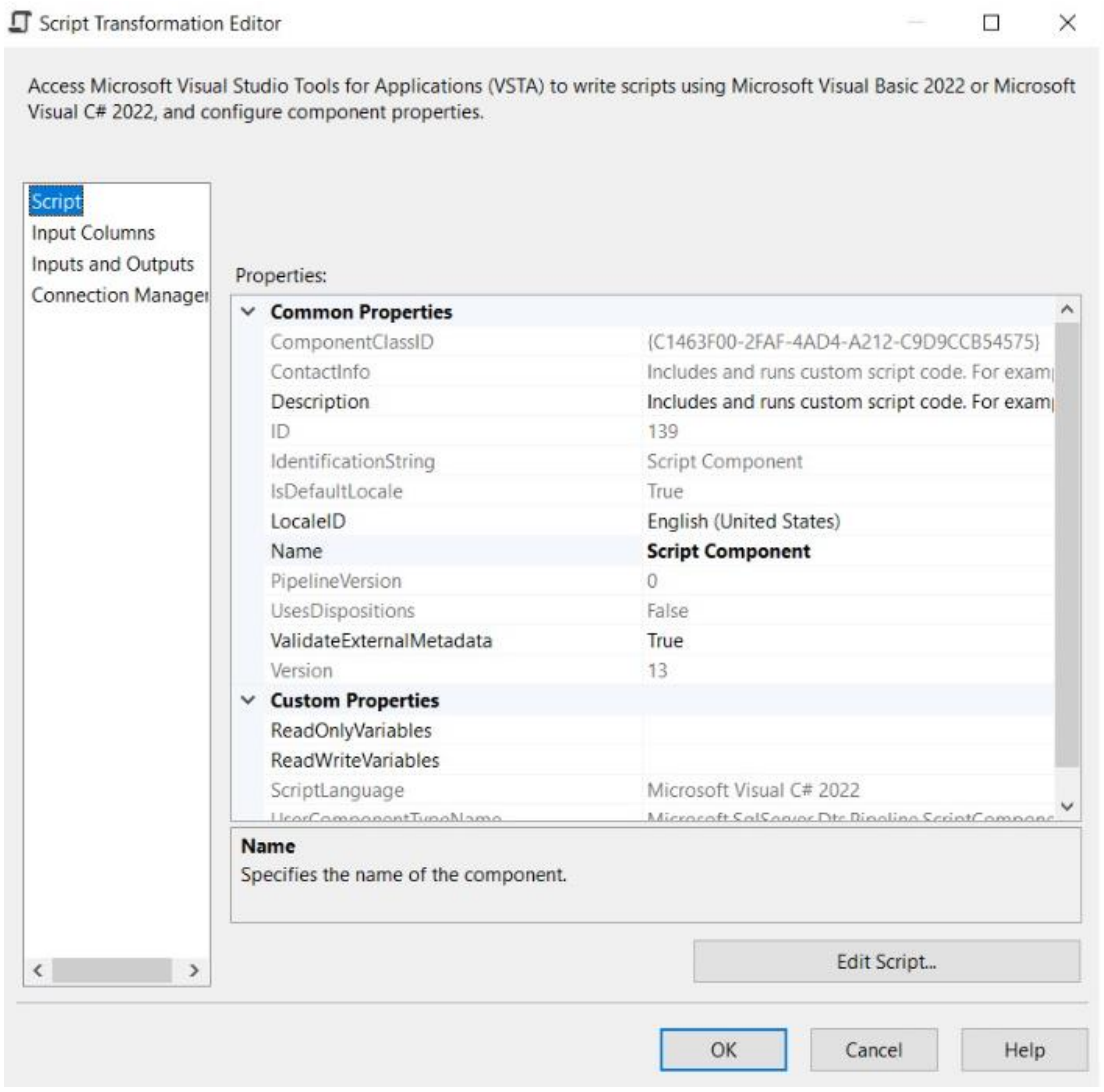


Union All Transformation Editor

Configure the properties used to merge multiple inputs into one output by creating mappings between columns.

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.



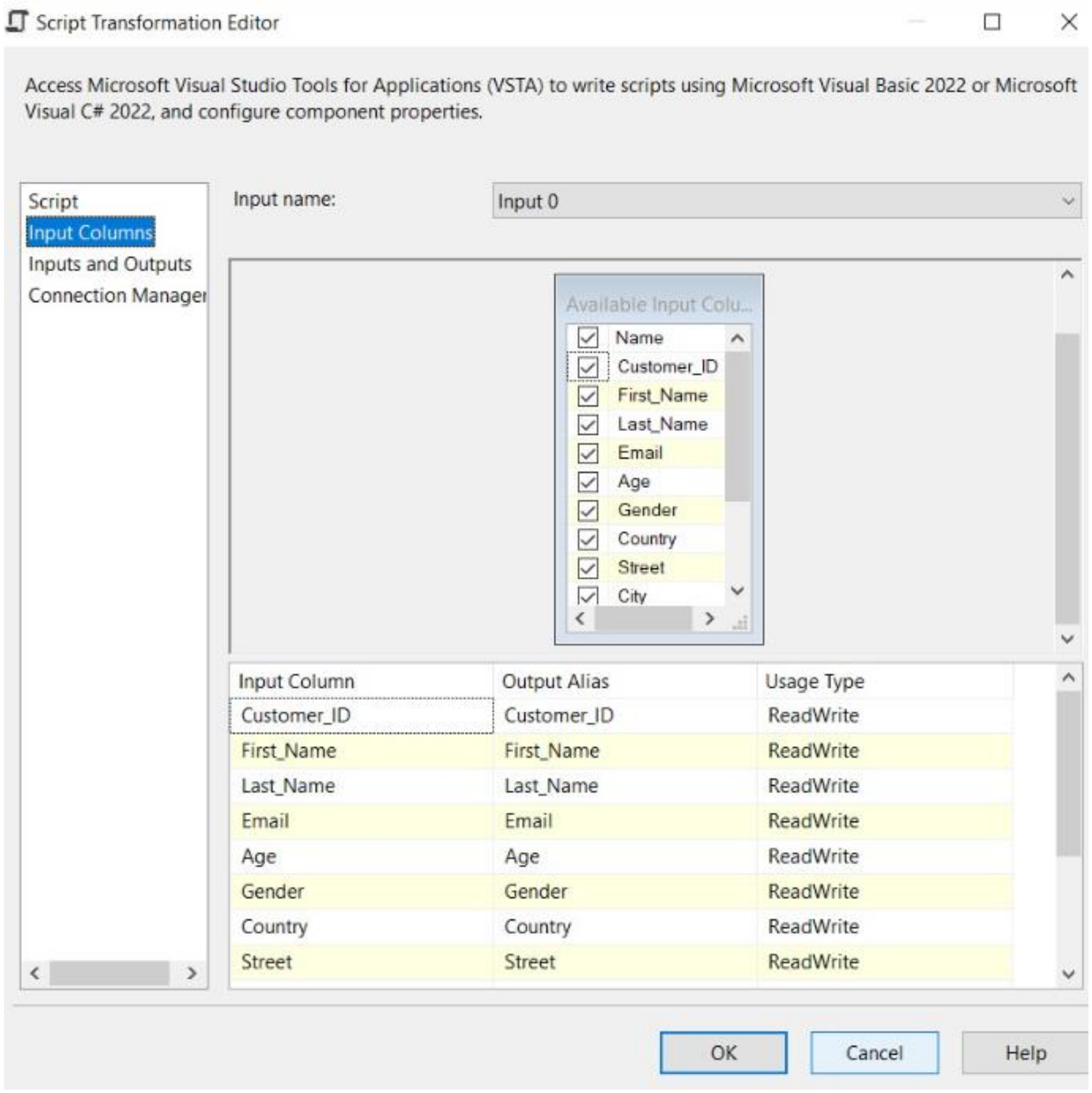
Script Transformation Editor

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.

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

Name
Specifies the name of the component.



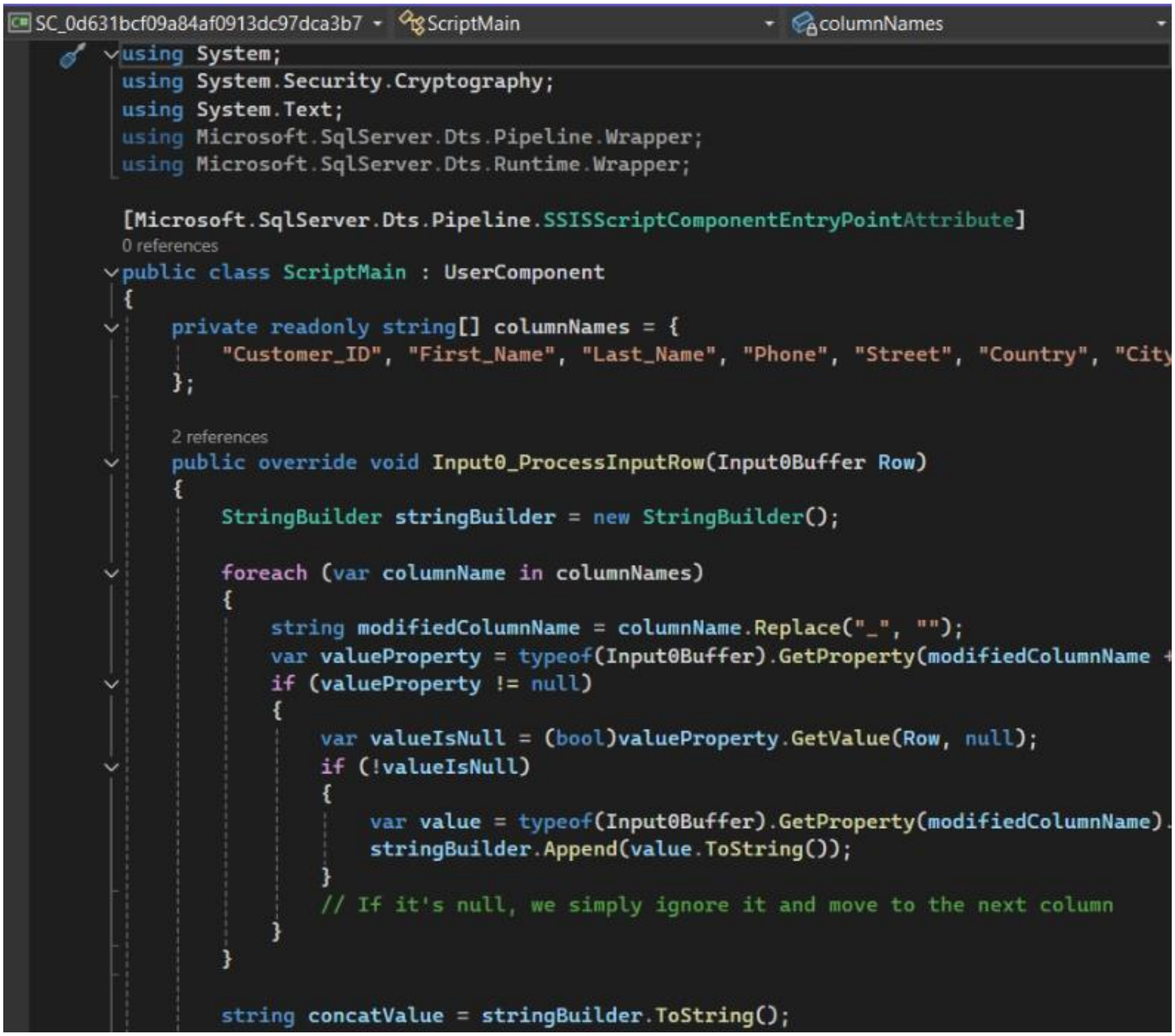
Access Microsoft Visual Studio Tools for Applications (VSTA) to write scripts using Microsoft Visual Basic 2022 or Microsoft Visual C# 2022, and configure component properties.

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



```
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]
public class ScriptMain : UserComponent
{
    private readonly string[] columnNames = {
        "Customer_ID", "First_Name", "Last_Name", "Phone", "Street", "Country", "City
    };

    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();
```

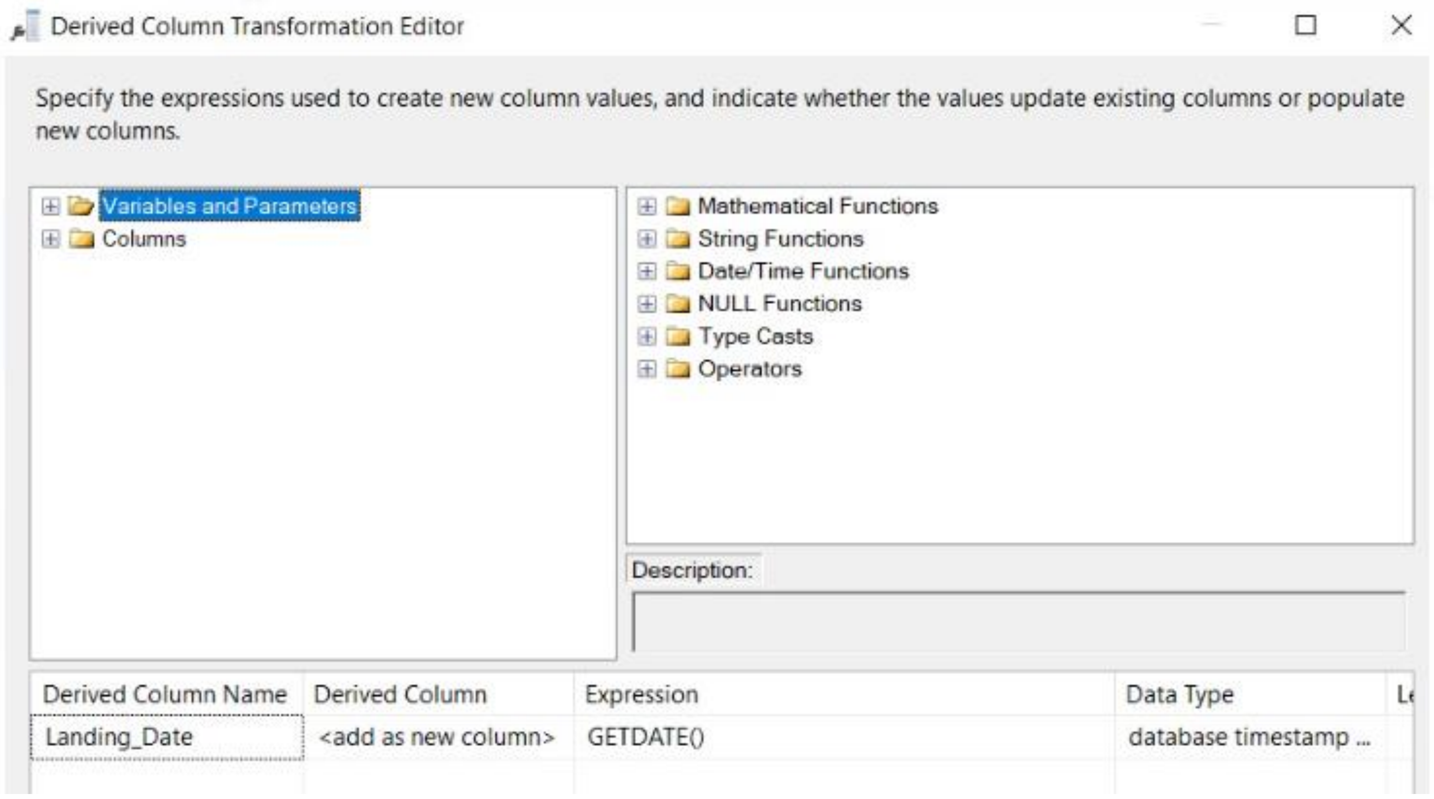
The script is designed to create a hash based on the concatenation of multiple columns, while handling null values appropriately. Each column's value is appended to a string if it's not null, and the resulting string is hashed using the SHA-256 algorithm. The hash is then stored in the Hashoutput column.

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.

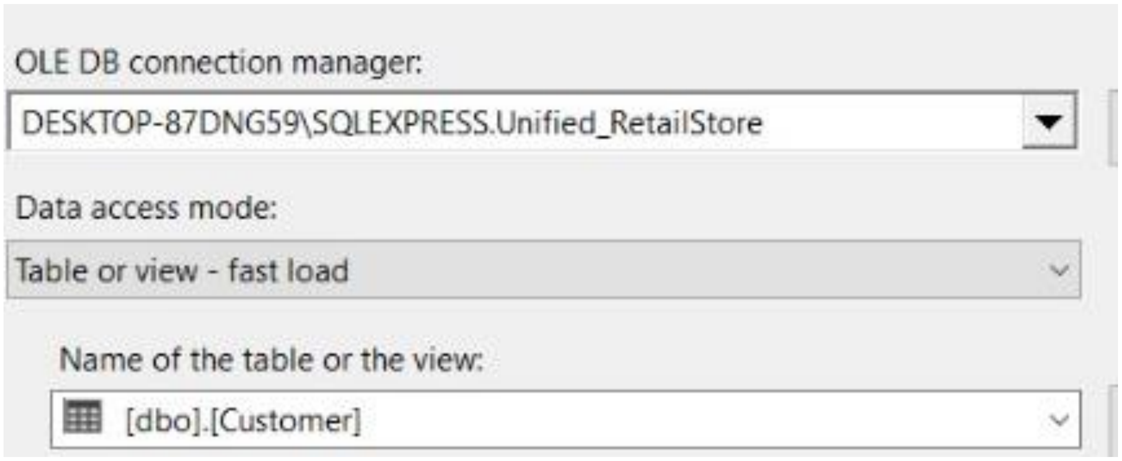
- **Derived Column:** Adds the landingDate column to the destination table.



Specify the expressions used to create new column values, and indicate whether the values update existing columns or populate new columns.

Derived Column Name	Derived Column	Expression	Data Type	Le
Landing_Date	<add as new column>	GETDATE()	database timestamp ...	

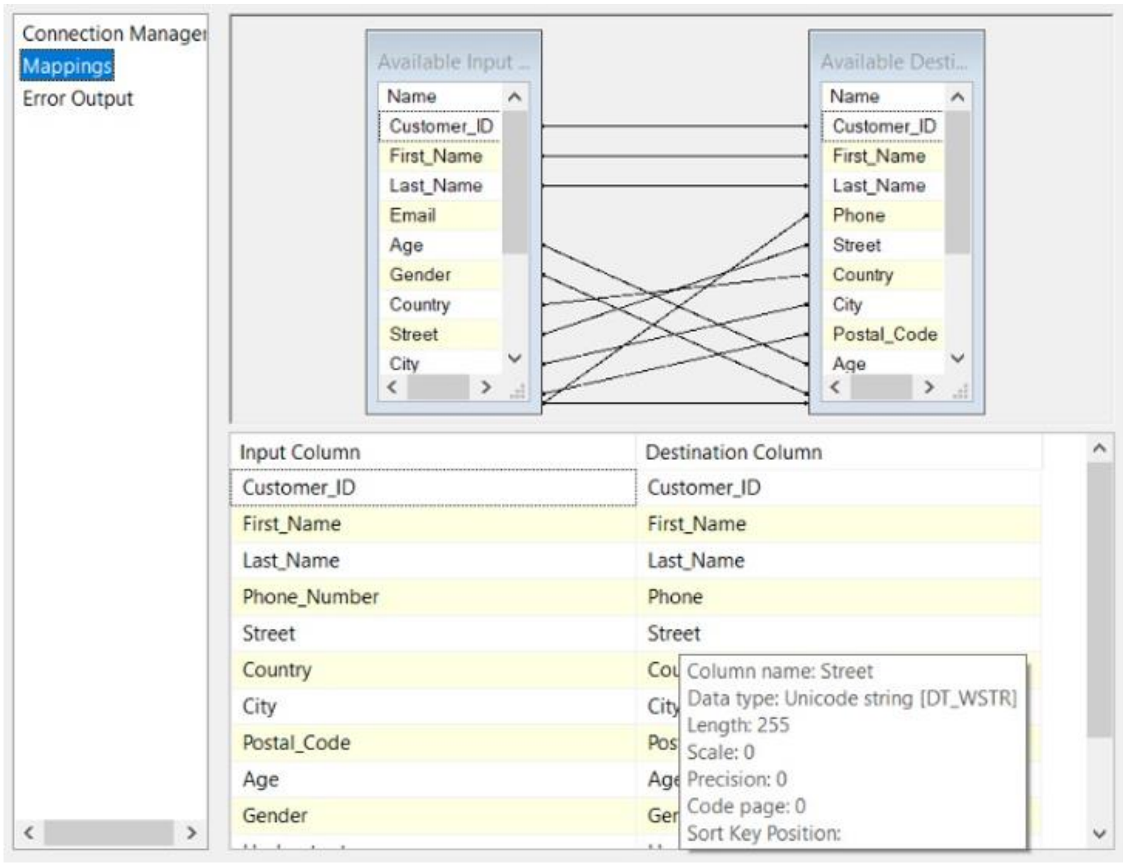
- **OLE DB Destination:** Utilizes "Fast Load" in data access mode to efficiently load data into the target table, with proper mapping between source and destination columns.



OLE DB connection manager: DESKTOP-87DNG59\SQLEXPRESS.Unified_RetailStore

Data access mode: Table or view - fast load

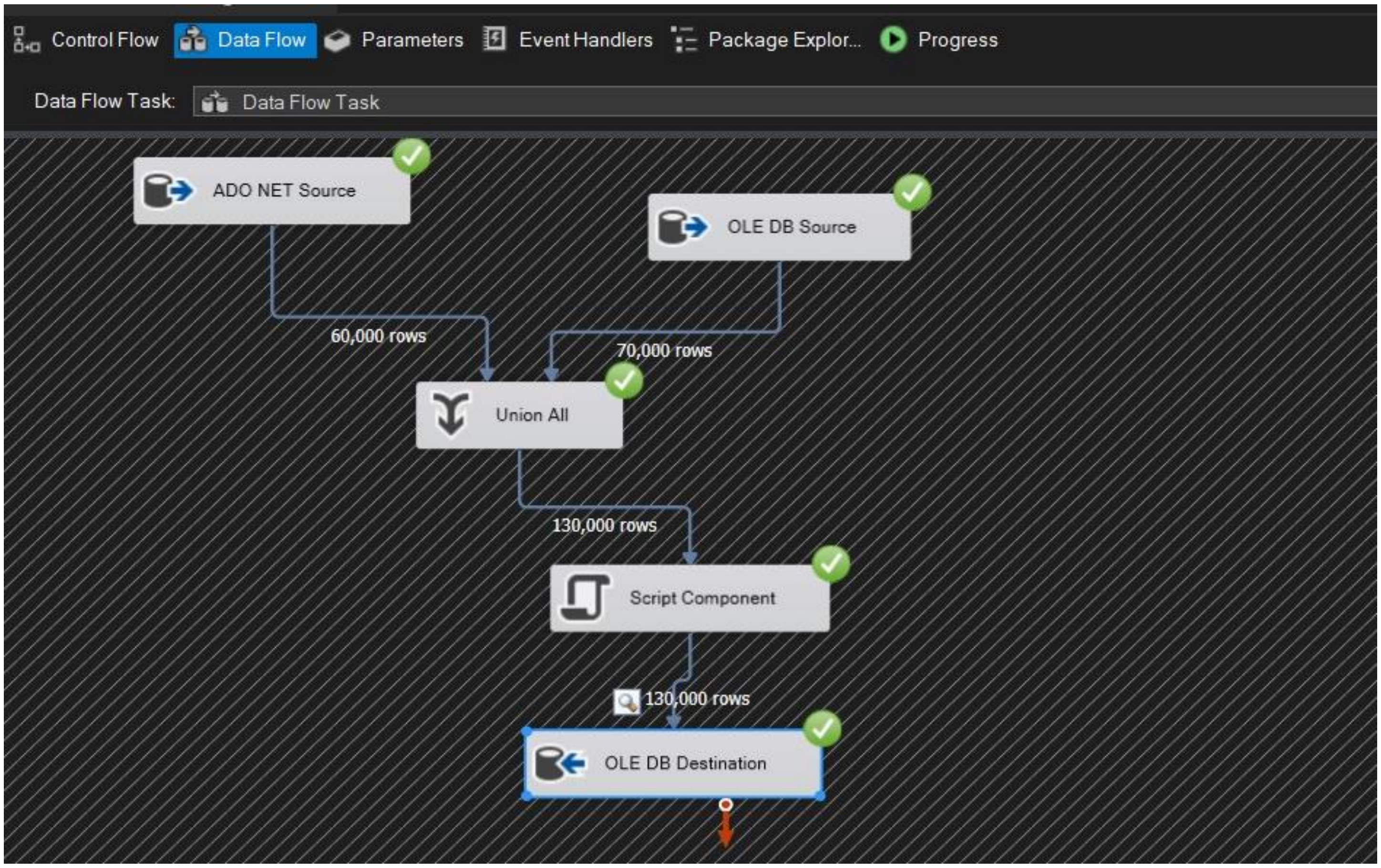
Name of the table or the view: [dbo].[Customer]



Input Column	Destination Column
Customer_ID	Customer_ID
First_Name	First_Name
Last_Name	Last_Name
Phone_Number	Phone
Street	Street
Country	Cou
City	City
Postal_Code	Pos
Age	Age
Gender	Ger

Column name: Street
Data type: Unicode string [DT_WSTR]
Length: 255
Scale: 0
Precision: 0
Code page: 0
Sort Key Position:

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.

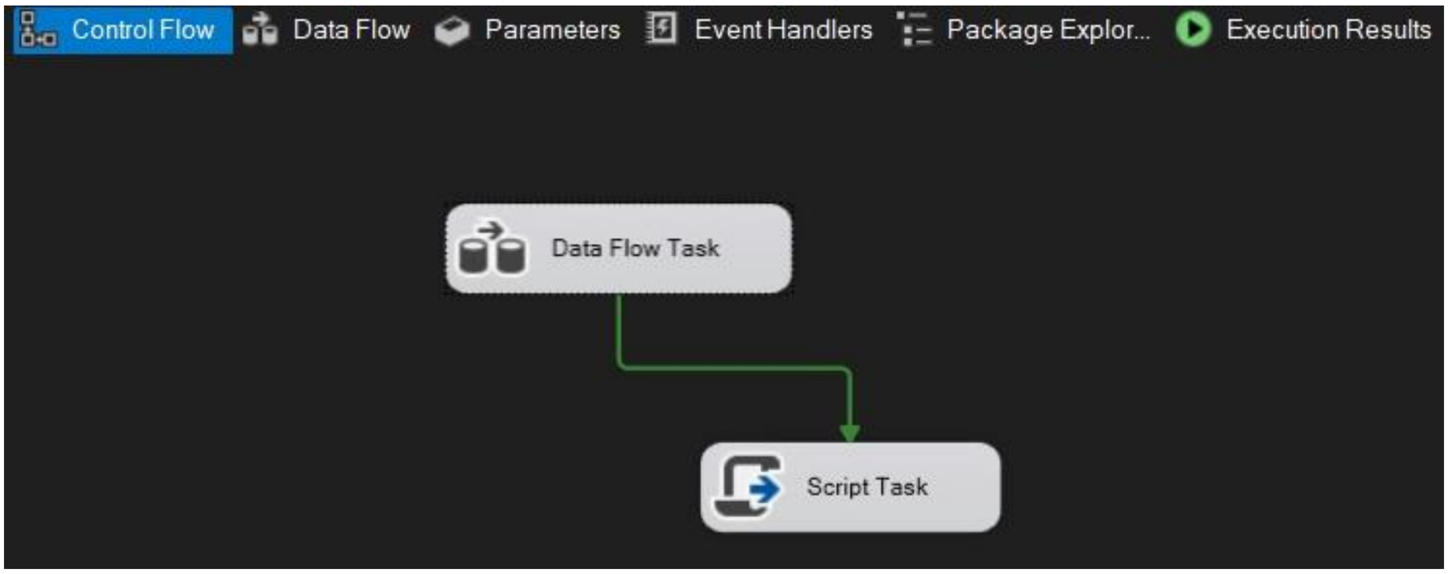
Consistency Across Packages:

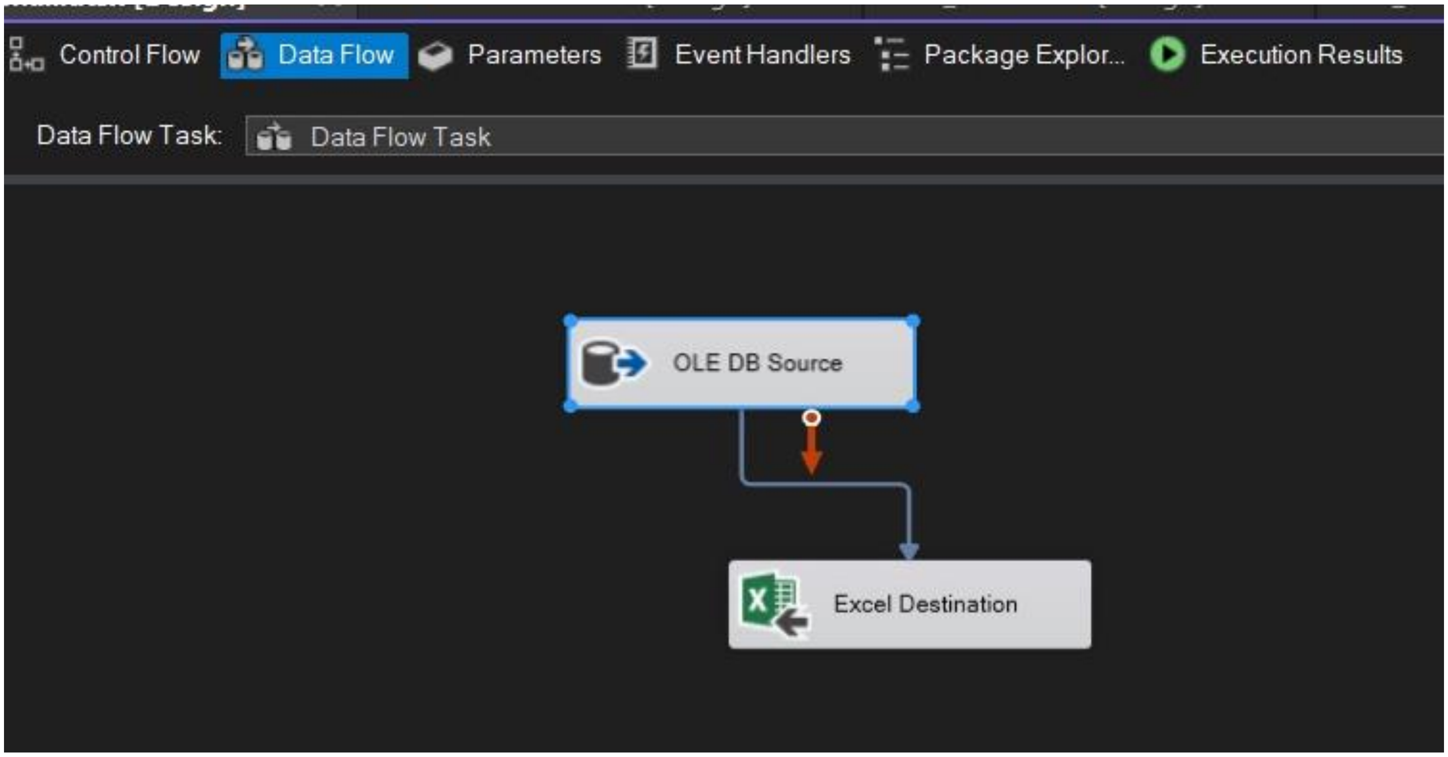
- We maintained a consistent approach across all ETL packages, covering every table in the schema.

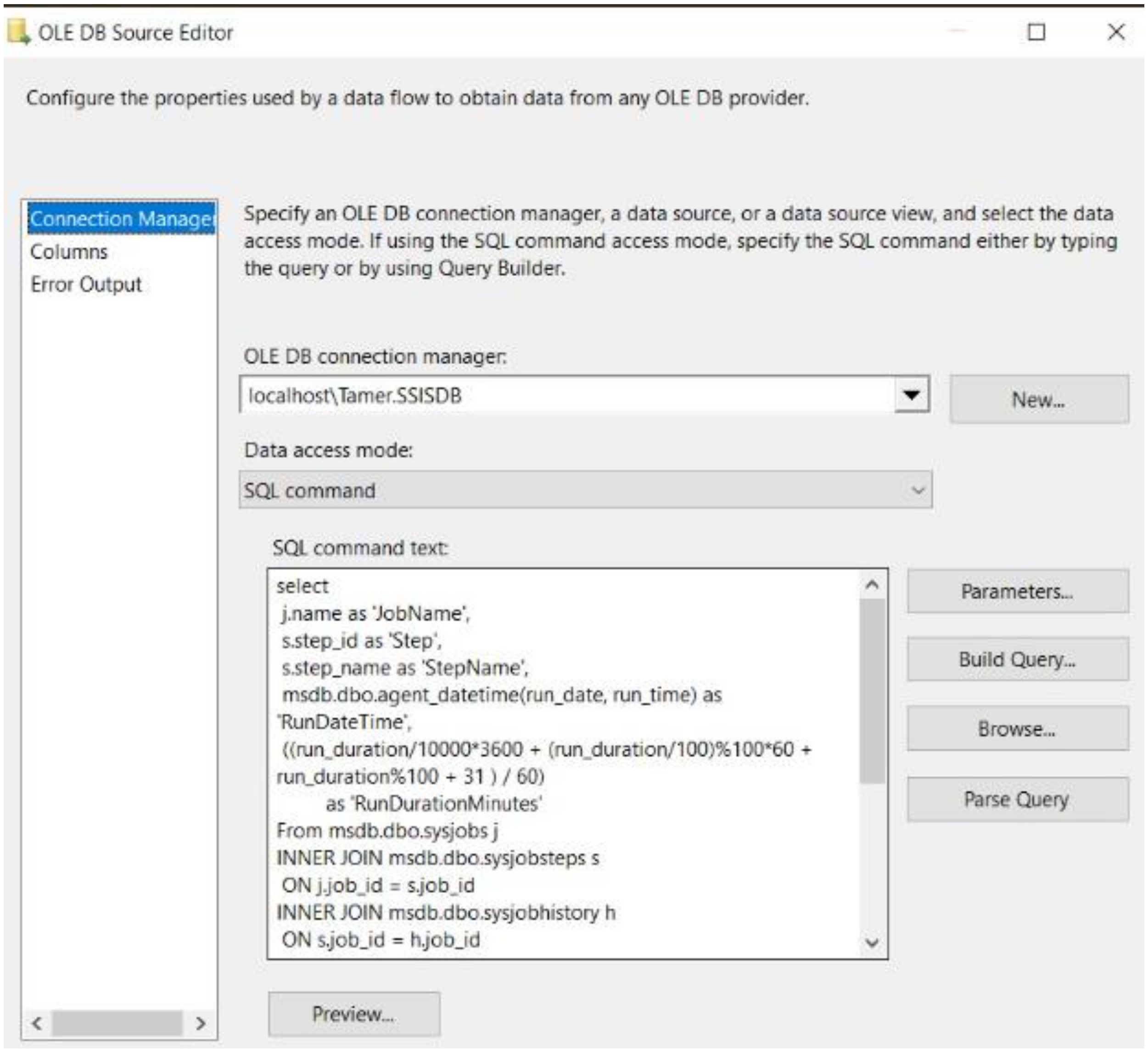
Automated Email Notification Package

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.







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

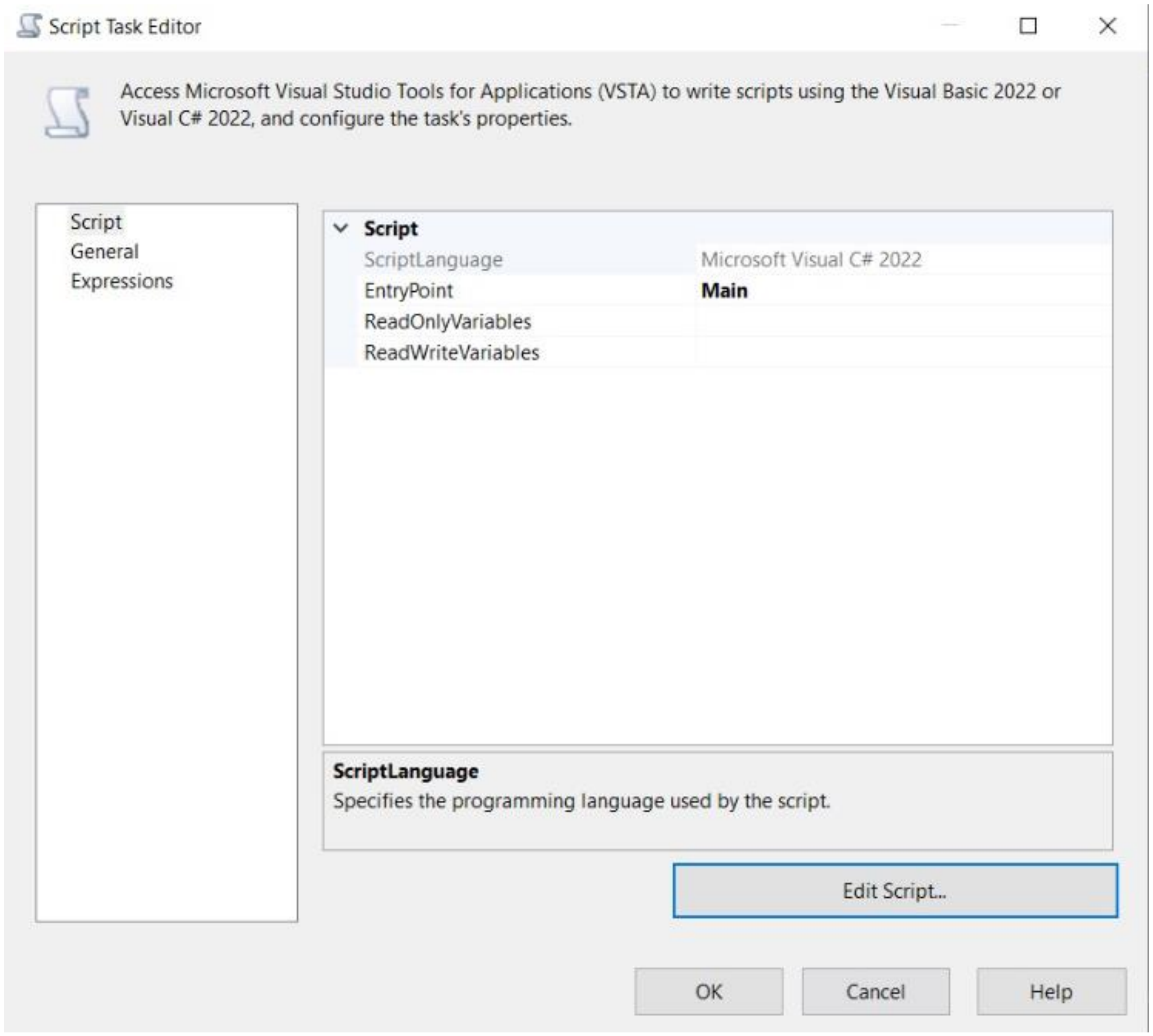
Specify an OLE DB connection manager, a data source, or a data source view, and select the data access mode. If using the SQL command access mode, specify the SQL command either by typing the query or by using Query Builder.

OLE DB connection manager: localhost\Tamer.SSISDB

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
```


Script Task Editor

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

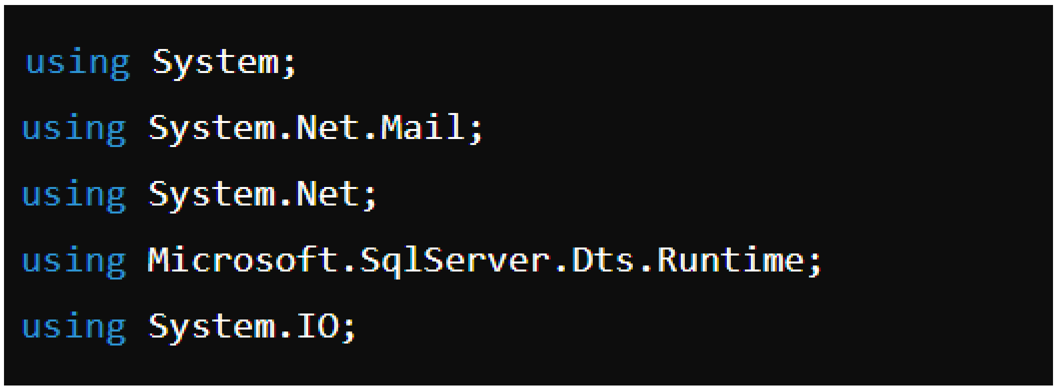
Script	
ScriptLanguage	Microsoft Visual C# 2022
EntryPoint	**Main**
ReadOnlyVariables	
ReadWriteVariables	

ScriptLanguage
Specifies the programming language used by the script.

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:

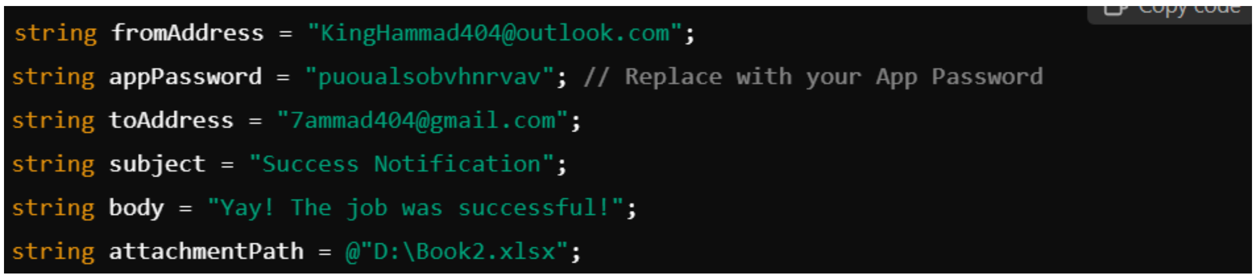


```
using System;
using System.Net.Mail;
using System.Net;
using Microsoft.SqlServer.Dts.Runtime;
using System.IO;
```

- **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";
```

- **fromAddress**: The sender's email address.
- **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).
- **toAddress**: The recipient's email address.

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

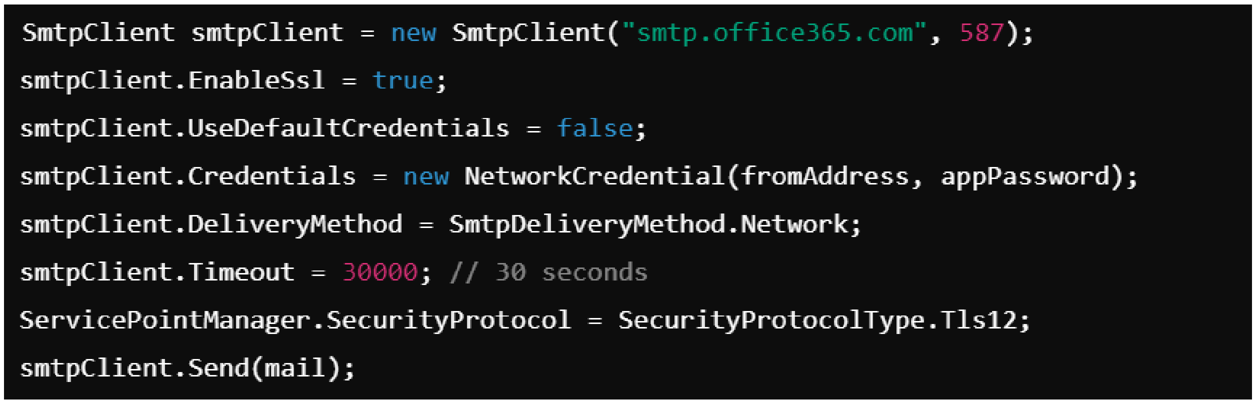
The script checks if the file exists before attaching it to the email:

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

- 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);
```

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

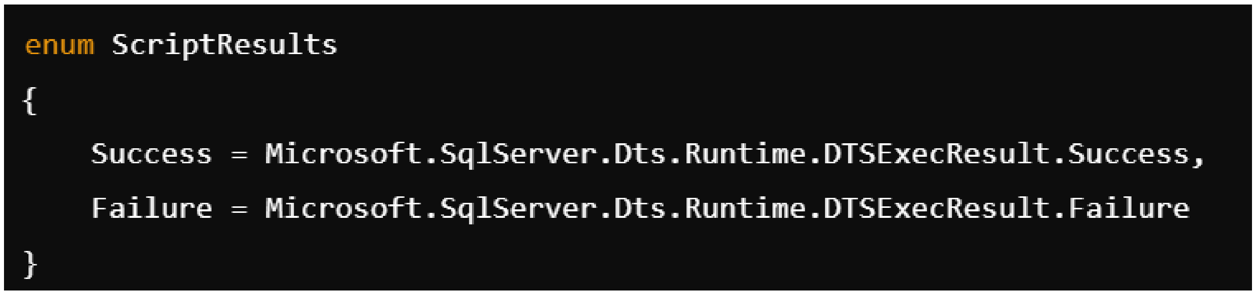
In case something goes wrong, the catch block captures exceptions and logs detailed error messages:

```
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:



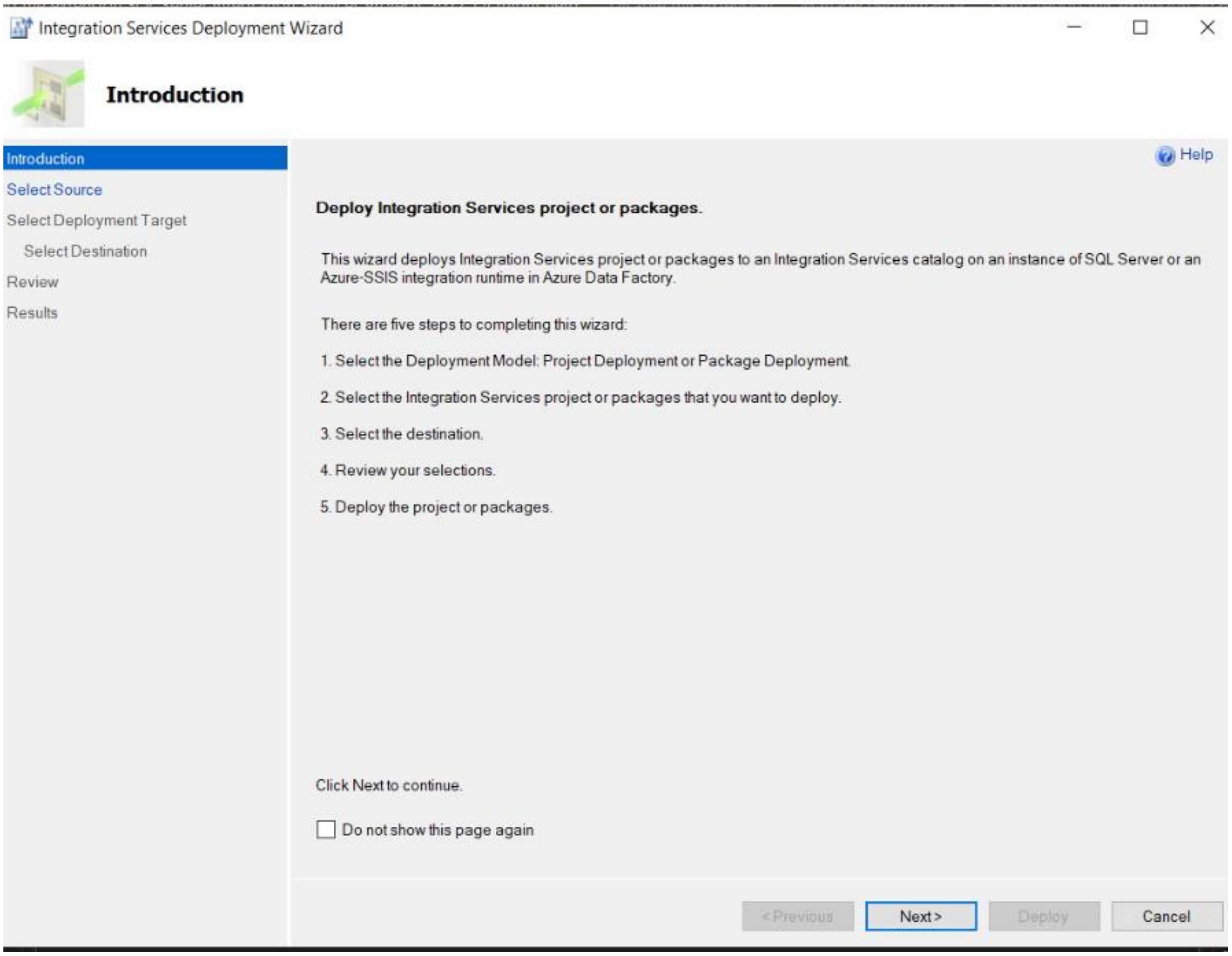
```
enum ScriptResults
{
    Success = Microsoft.SqlServer.Dts.Runtime.DTSExecResult.Success,
    Failure = Microsoft.SqlServer.Dts.Runtime.DTSExecResult.Failure
}
```

- 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.



Deploy Integration Services project or packages.

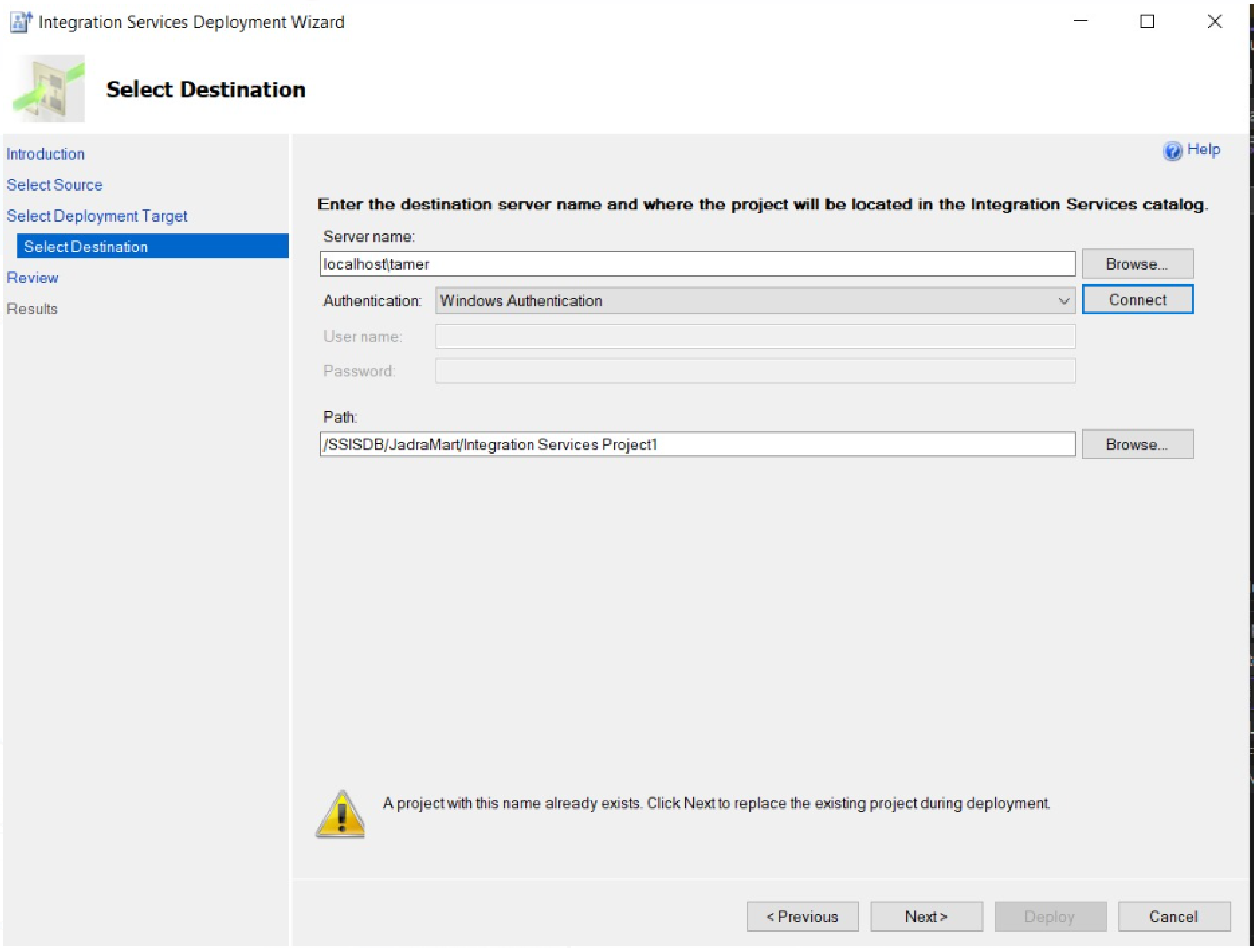
This wizard deploys Integration Services project or packages to an Integration Services catalog on an instance of SQL Server or an Azure-SSIS integration runtime in Azure Data Factory.

There are five steps to completing this wizard:

1. Select the Deployment Model: Project Deployment or Package Deployment.
2. Select the Integration Services project or packages that you want to deploy.
3. Select the destination.
4. Review your selections.
5. Deploy the project or packages.

Click Next to continue.

☐ Do not show this page again



Integration Services Deployment Wizard

Select Destination

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

Server name: localhost\tamer

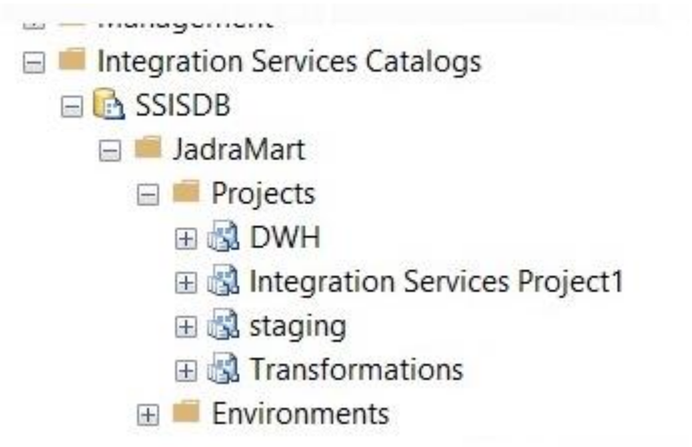
Authentication: Windows Authentication

User name:

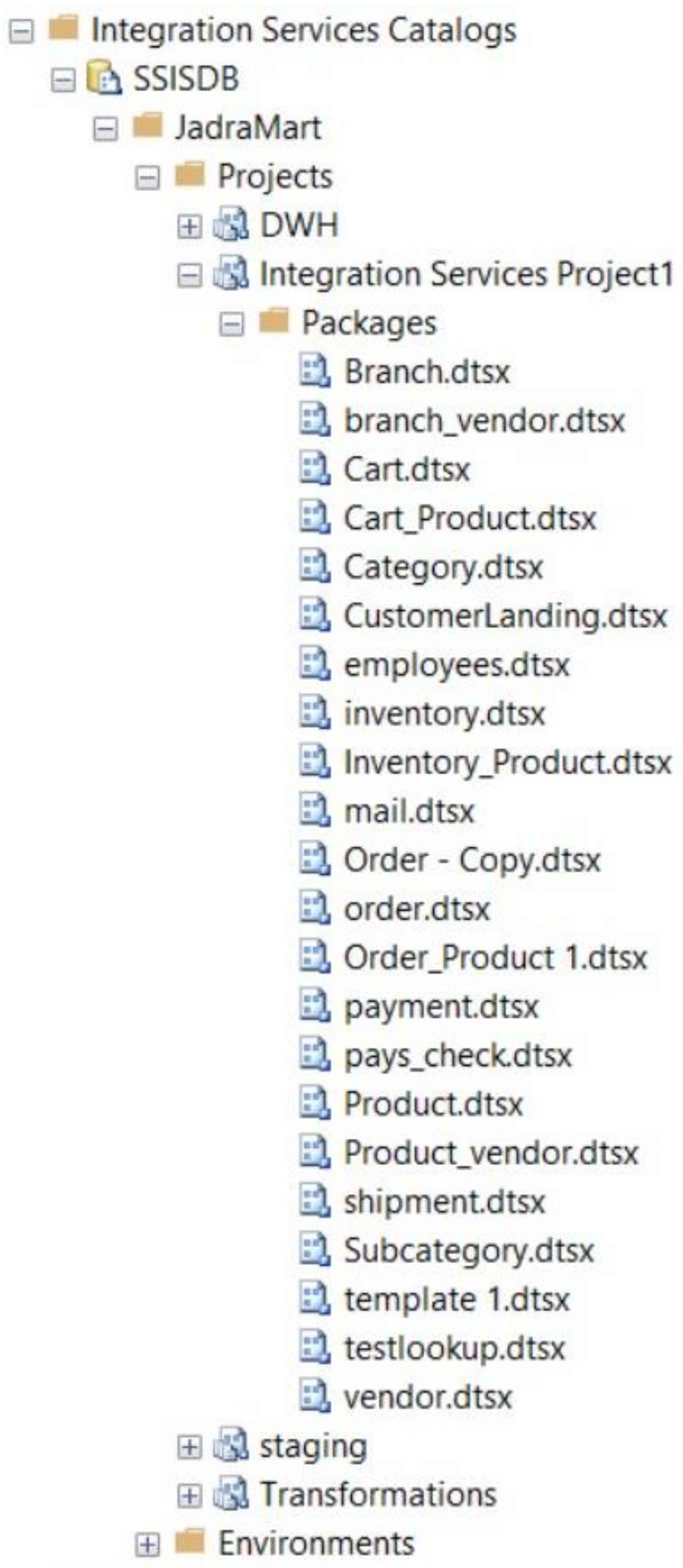
Password:

Path: /SSISDB/JadraMart/Integration Services Project1

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

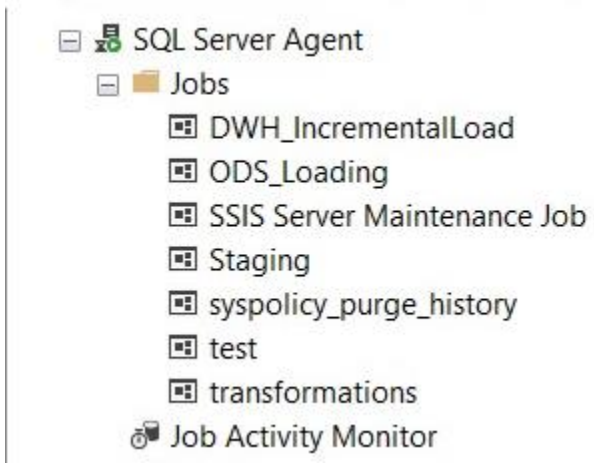


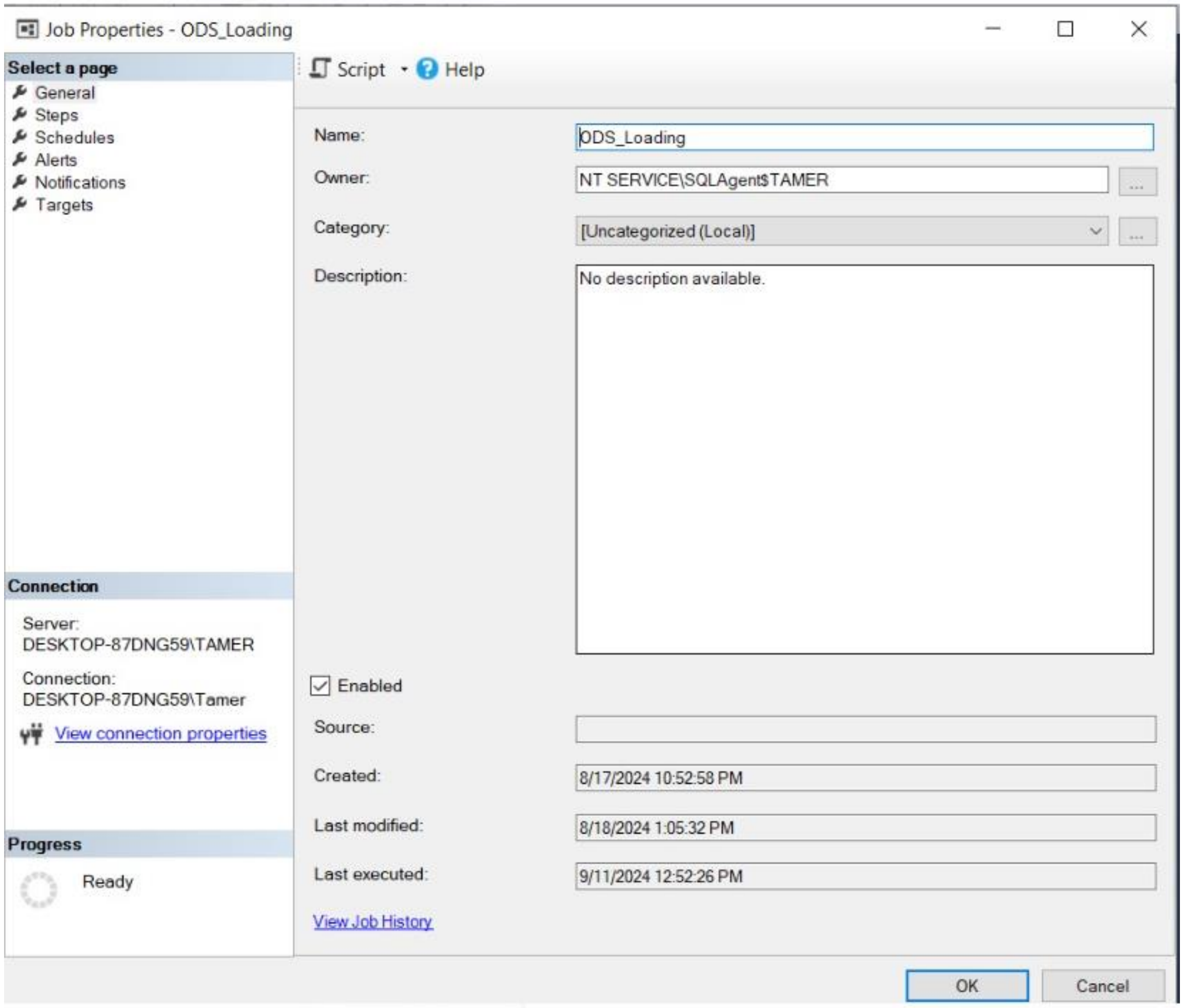
```
⊟ Integration Services Catalogs
  ⊟ SSISDB
    ⊟ JadraMart
      ⊟ Projects
        ⊞ DWH
        ⊞ Integration Services Project1
        ⊞ staging
        ⊞ Transformations
      ⊞ Environments
```



- 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

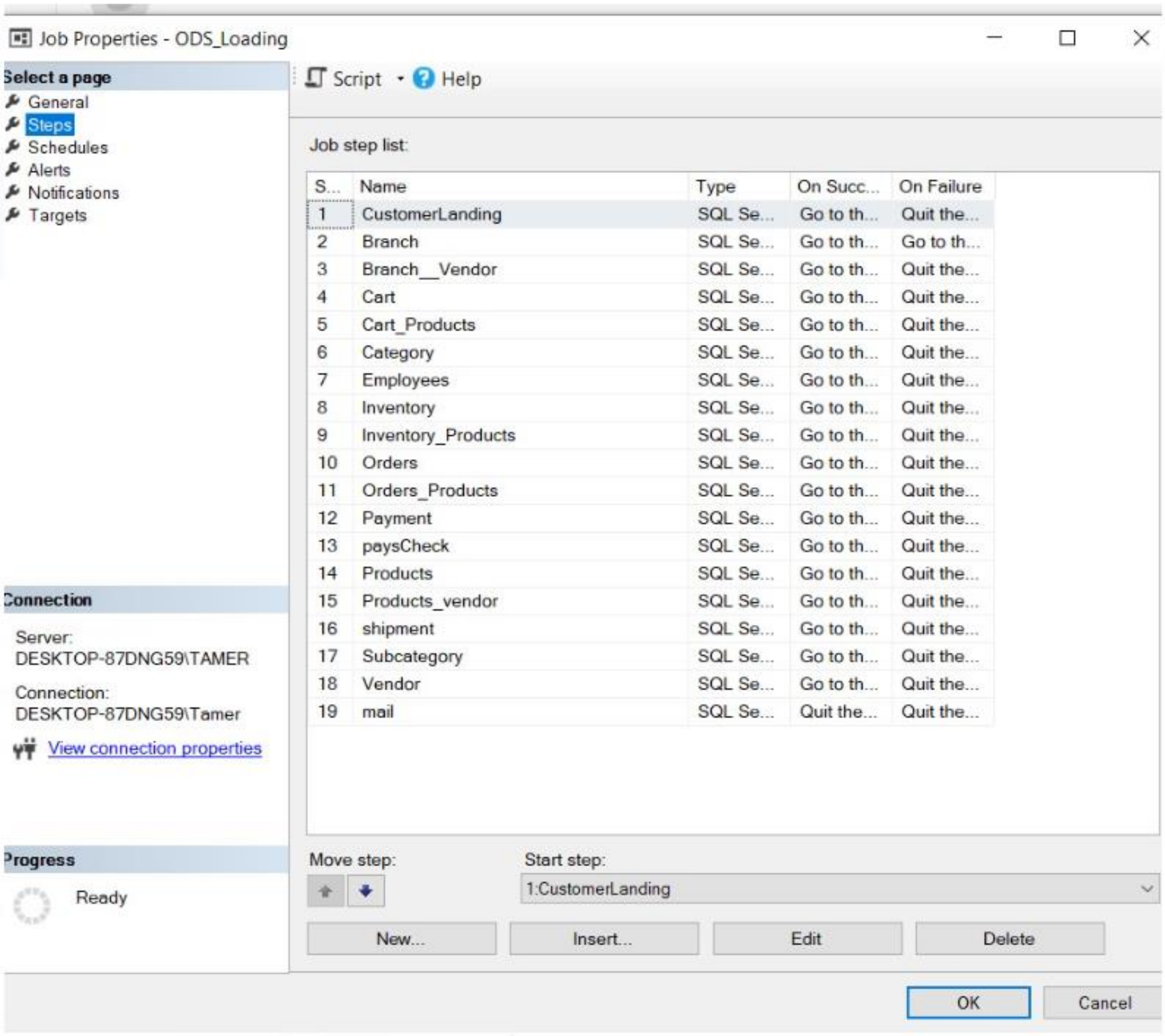
Job Creation


SQL Server Agent
Jobs
DWH_IncrementalLoad
ODS_Loading
SSIS Server Maintenance Job
Staging
syspolicy_purge_history
test
transformations
Job Activity Monitor


Job Properties - ODS_Loading

Name:	ODS_Loading
Owner:	NT SERVICE\SQLAgent$TAMER
Category:	[Uncategorized (Local)]
Description:	No description available.
☑ Enabled	
Source:	
Created:	8/17/2024 10:52:58 PM
Last modified:	8/18/2024 1:05:32 PM
Last executed:	9/11/2024 12:52:26 PM

Connection
Server:
DESKTOP-87DNG59\TAMER
Connection:
DESKTOP-87DNG59\Tamer


Job Properties - ODS_Loading

Select a page: General, Steps, Schedules, Alerts, Notifications, Targets

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

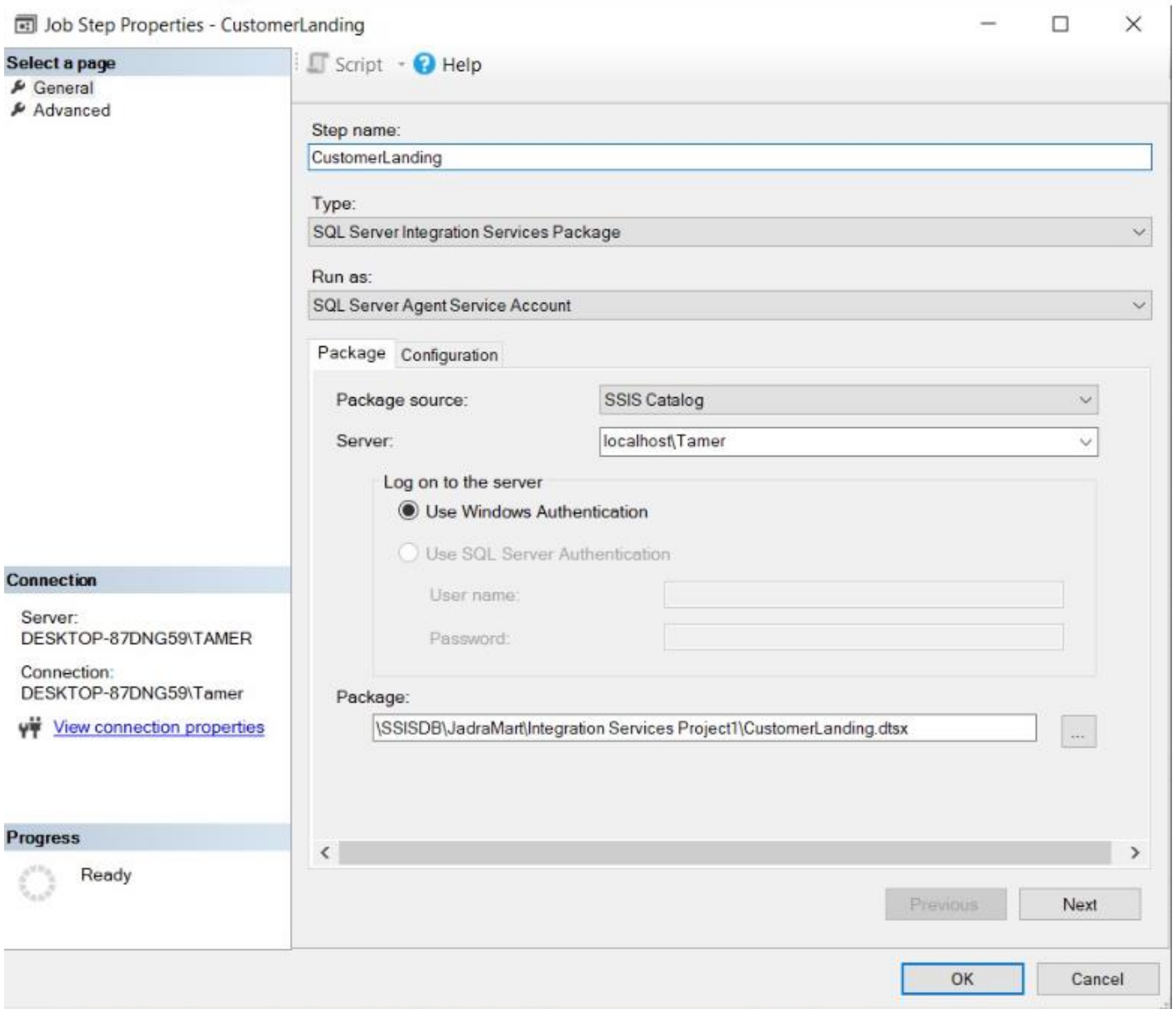
Ready

Move step:

Start step: 1:CustomerLanding

New... Insert... Edit Delete

OK Cancel



Step name: CustomerLanding

Type: SQL Server Integration Services Package

Run as: SQL Server Agent Service Account

Package source: SSIS Catalog

Server: localhost\Tamer

Log on to the server

Use Windows Authentication [✓]

Use SQL Server Authentication

User name:

Password:

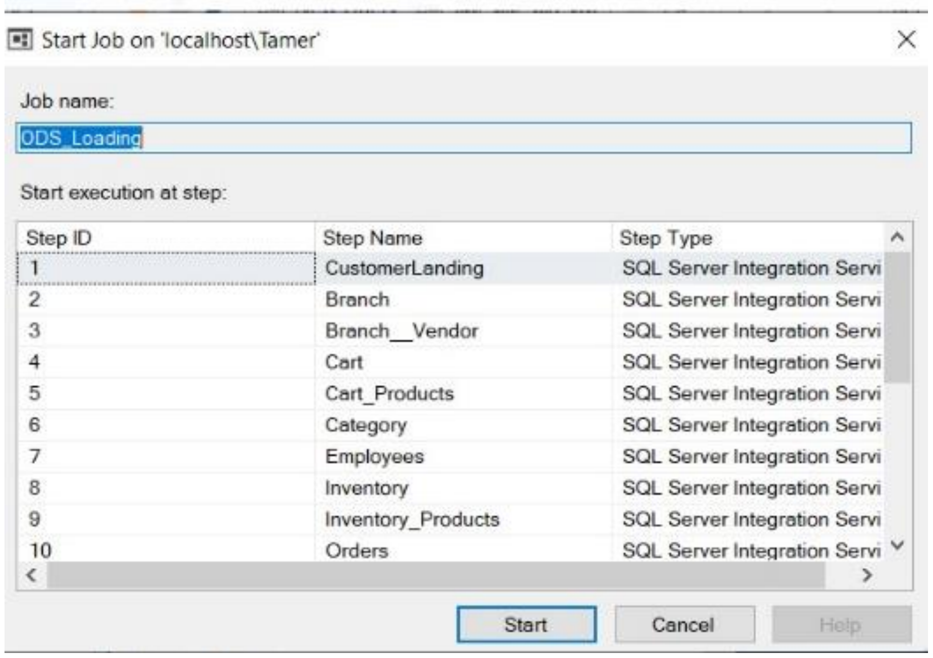
Package: \SSISDB\JadraMart\Integration Services Project1\CustomerLanding.dtsx

Connection

Server: DESKTOP-87DNG59\TAMER

Connection: DESKTOP-87DNG59\Tamer

Job Start:

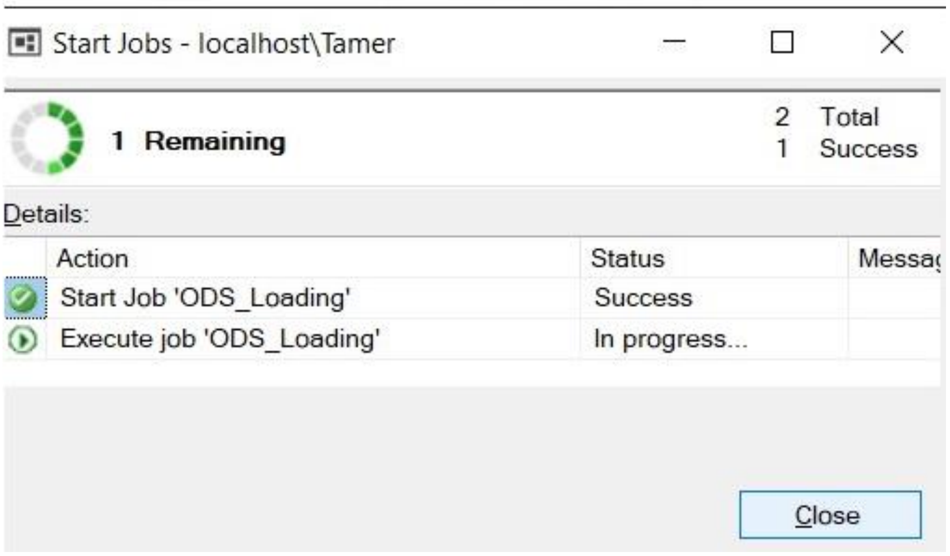

Start Job on 'localhost\Tamer'

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

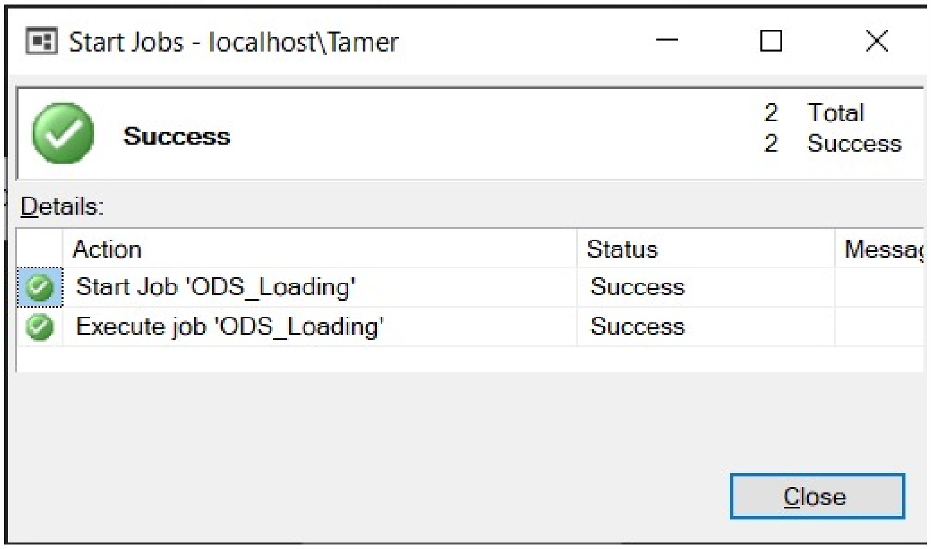

Start Jobs - localhost\Tamer

1 Remaining

2 Total
1 Success

Details:

	Action	Status	Messa
✓	Start Job 'ODS_Loading'	Success	
▶	Execute job 'ODS_Loading'	In progress...	


Start Jobs - localhost\Tamer

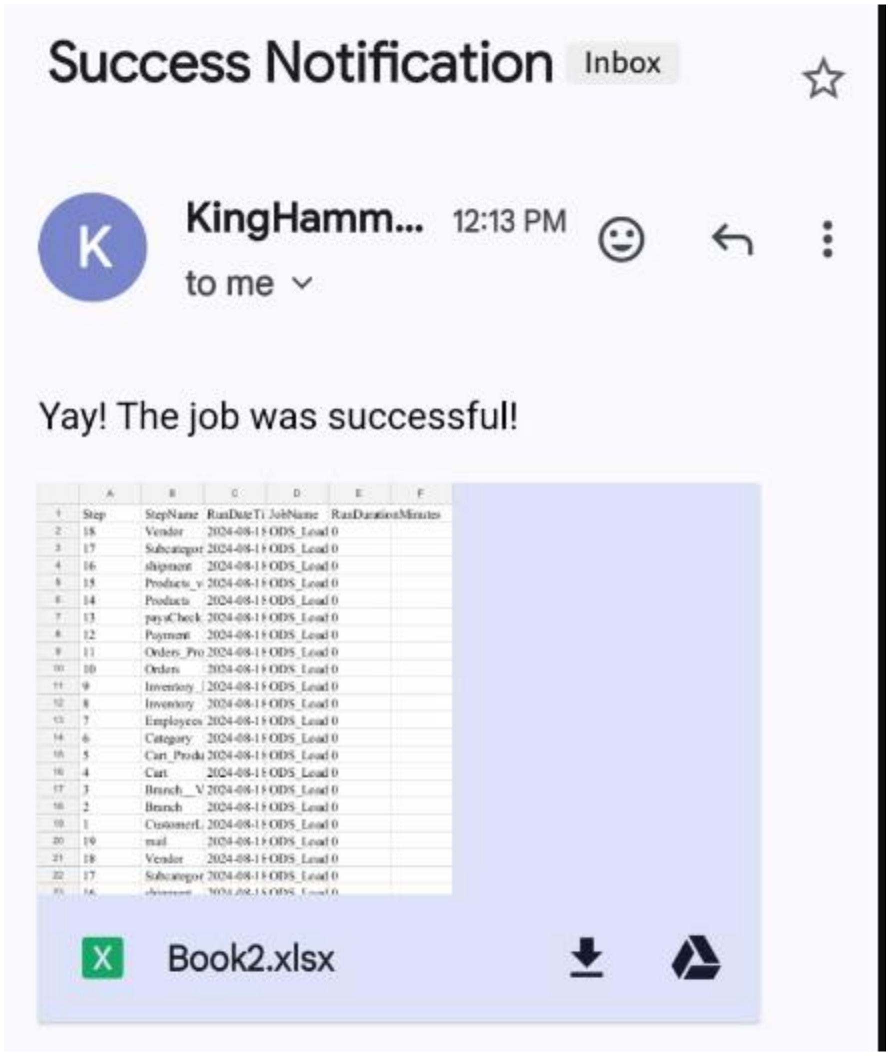
Success

2 Total
2 Success

Details:

	Action	Status	Messa
✓	Start Job 'ODS_Loading'	Success	
✓	Execute job 'ODS_Loading'	Success	

Email after running the package:



Success Notification Inbox

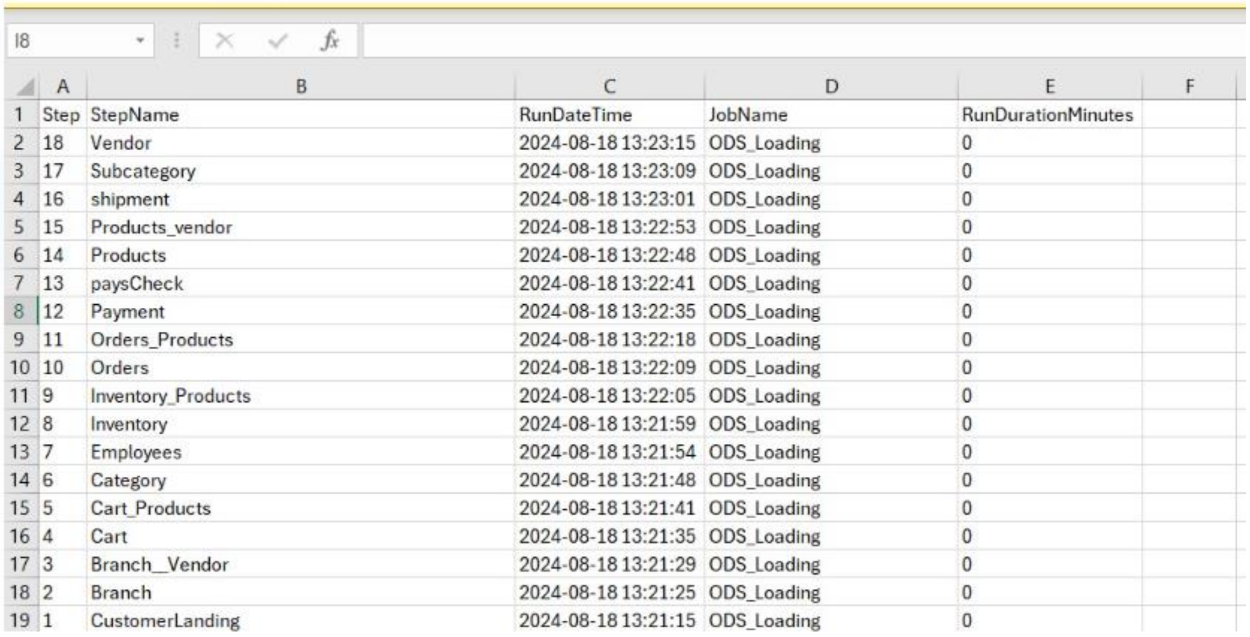
KingHamm... 12:13 PM

to me

Yay! The job was successful!

Book2.xlsx

Sent File Content:

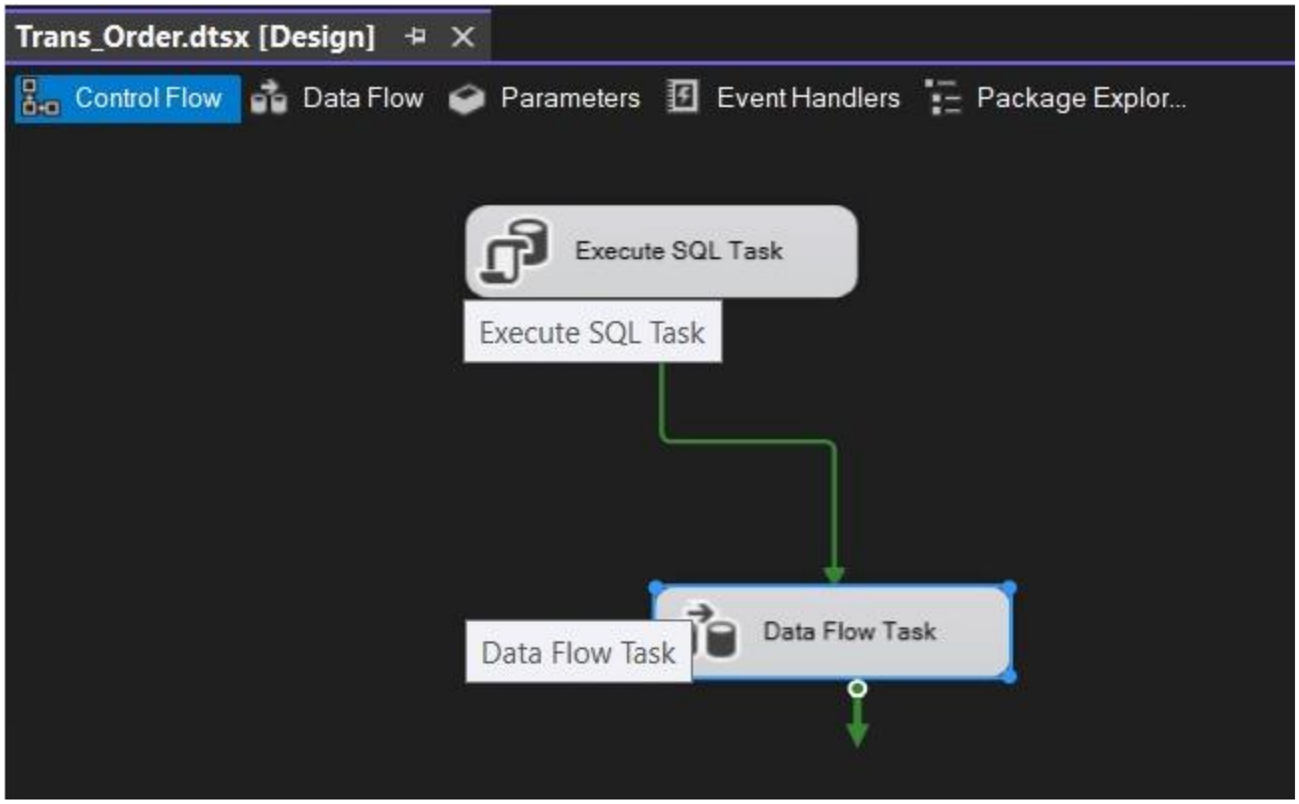


	A	B	C	D	E	F
1	Step	StepName	RunDateTime	JobName	RunDurationMinutes	
2	18	Vendor	2024-08-18 13:23:15	ODS_Loading	0	
3	17	Subcategory	2024-08-18 13:23:09	ODS_Loading	0	
4	16	shipment	2024-08-18 13:23:01	ODS_Loading	0	
5	15	Products_vendor	2024-08-18 13:22:53	ODS_Loading	0	
6	14	Products	2024-08-18 13:22:48	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	
18	2	Branch	2024-08-18 13:21:25	ODS_Loading	0	
19	1	CustomerLanding	2024-08-18 13:21:15	ODS_Loading	0	

ETL Process for Data Transformation

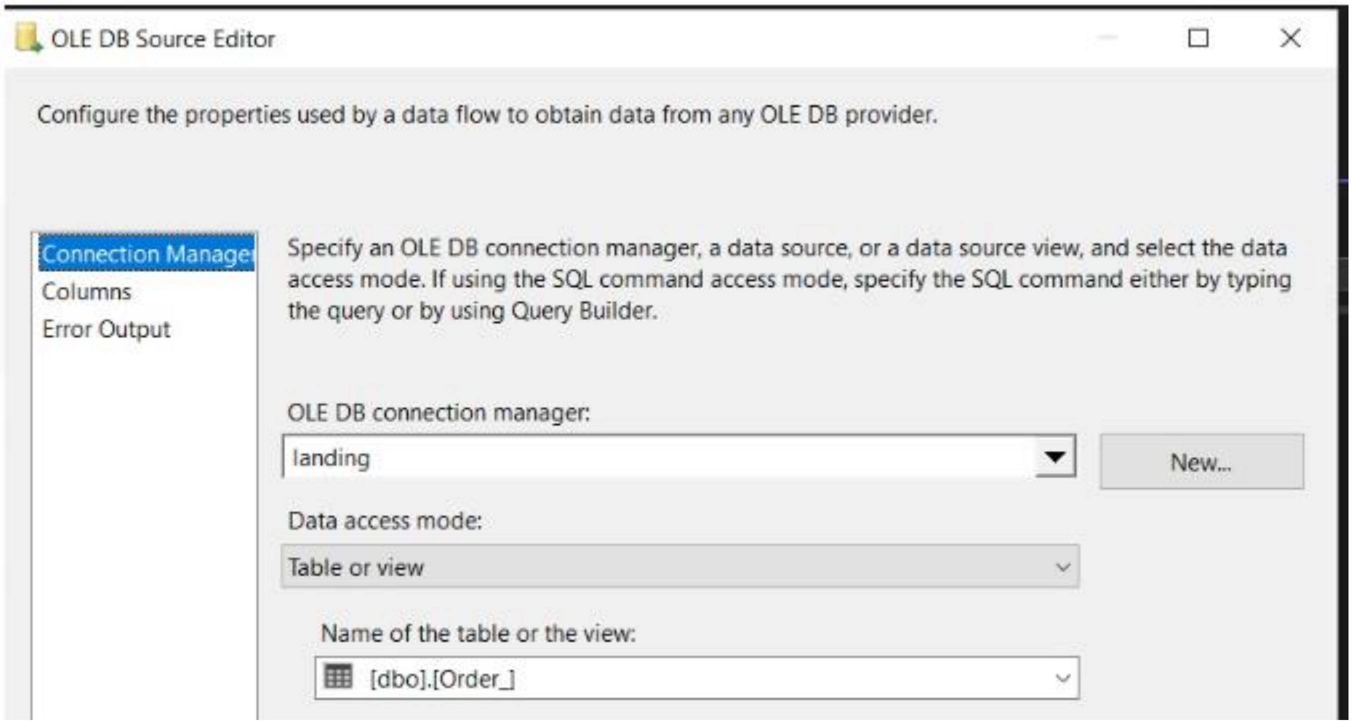
To manage data transformations effectively, we developed a separate project dedicated specifically to this purpose. This approach enables us to isolate and independently execute the transformation logic when needed, ensuring flexibility and modularity in our data pipeline.

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.



OLE DB Source Editor

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

Connection Manage
Columns
Error Output

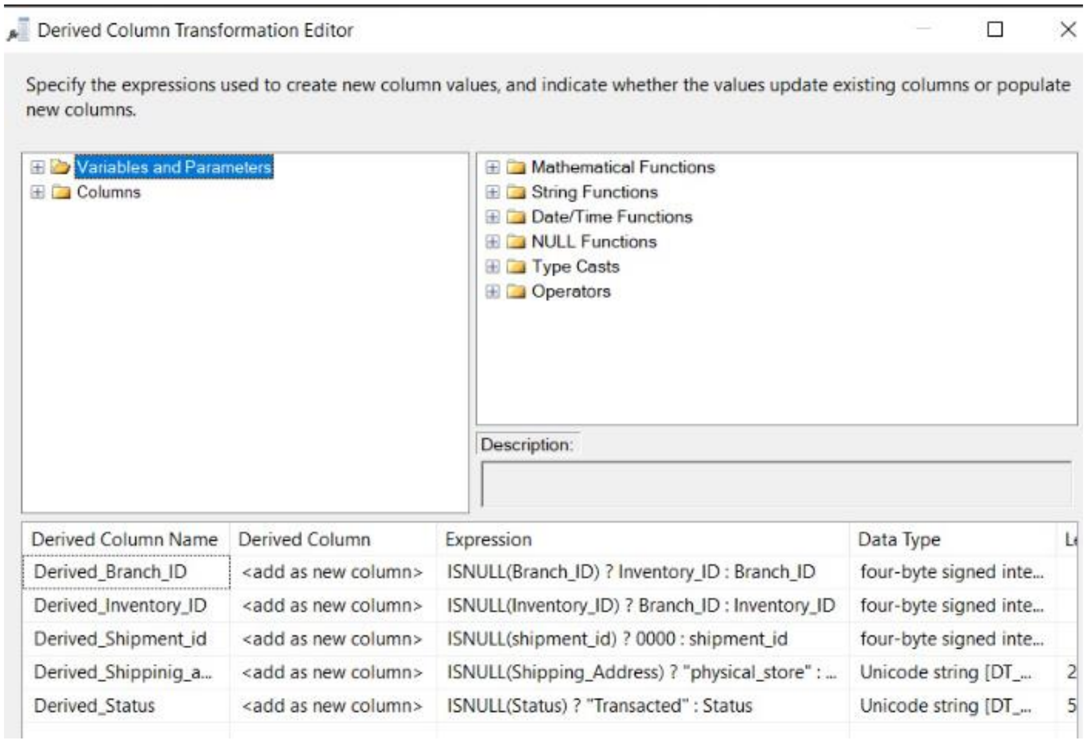
Specify an OLE DB connection manager, a data source, or a data source view, and select the data access mode. If using the SQL command access mode, specify the SQL command either by typing the query or by using Query Builder.

OLE DB connection manager: landing

Data access mode: Table or view

Name of the table or the view: [dbo].[Order_]

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.

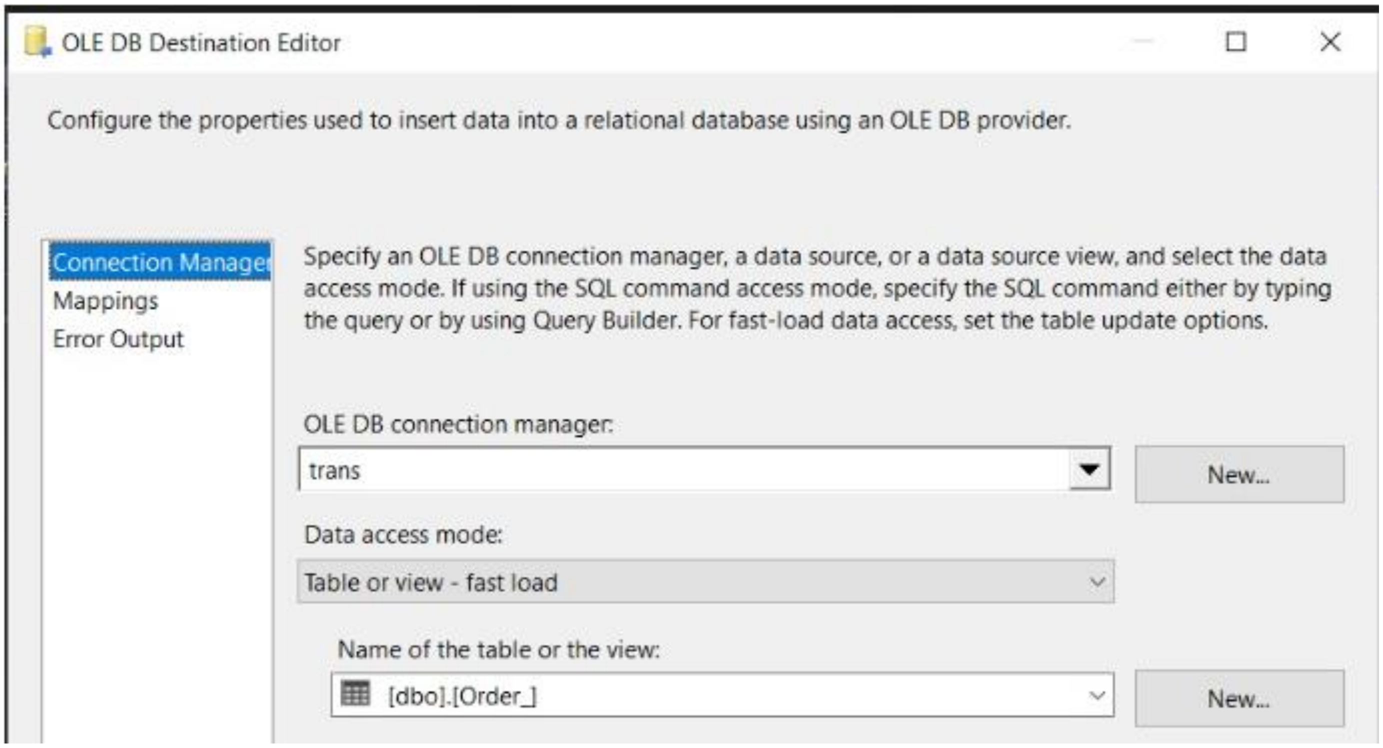


Derived Column Transformation Editor

Specify the expressions used to create new column values, and indicate whether the values update existing columns or populate new columns.

Derived Column Name	Derived Column	Expression	Data Type	Le
Derived_Branch_ID	<add as new column>	ISNULL(Branch_ID) ? Inventory_ID : Branch_ID	four-byte signed inte...	
Derived_Inventory_ID	<add as new column>	ISNULL(Inventory_ID) ? Branch_ID : Inventory_ID	four-byte signed inte...	
Derived_Shipment_id	<add as new column>	ISNULL(shipment_id) ? 0000 : shipment_id	four-byte signed inte...	
Derived_Shippinig_a...	<add as new column>	ISNULL(Shipping_Address) ? "physical_store" : ...	Unicode string [DT_...	2
Derived_Status	<add as new column>	ISNULL(Status) ? "Transacted" : Status	Unicode string [DT_...	5

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.


OLE DB Destination Editor

Configure the properties used to insert data into a relational database using an OLE DB provider.

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. For fast-load data access, set the table update options.

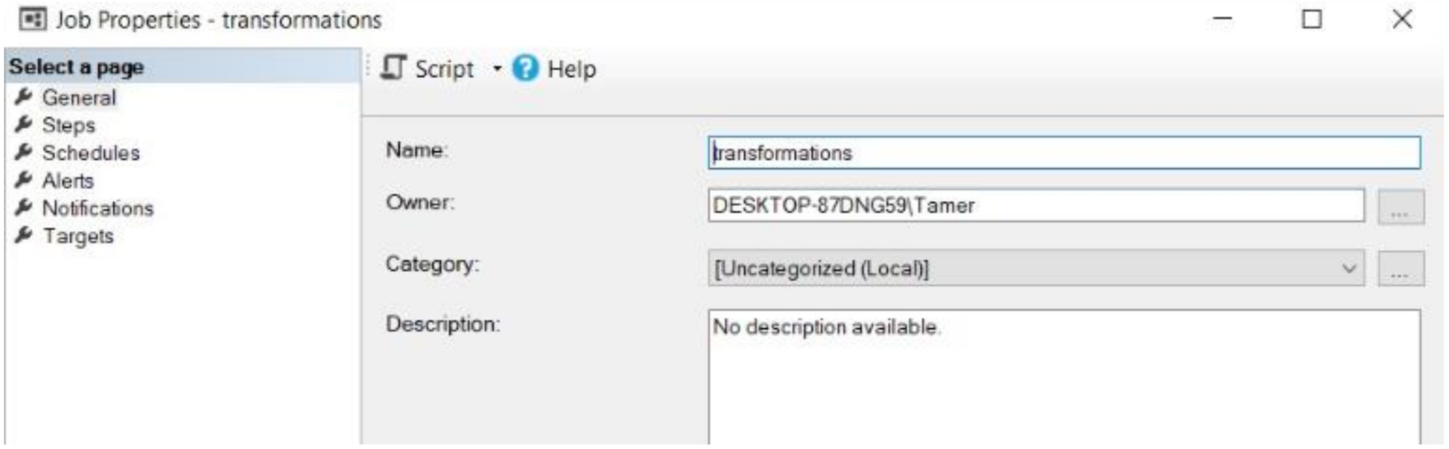
OLE DB connection manager: trans

Data access mode: Table or view - fast load

Name of the table or the view: [dbo].[Order_]

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.

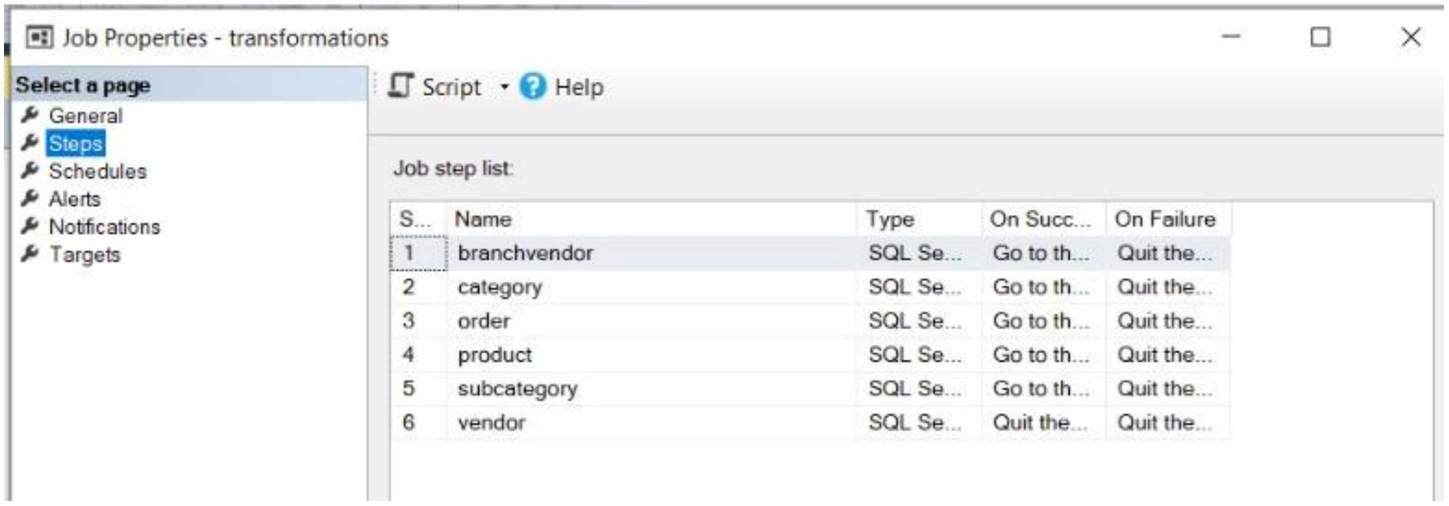

Job Properties - transformations

Name: transformations

Owner: DESKTOP-87DNG59\Tamer

Category: [Uncategorized (Local)]

Description: No description available.



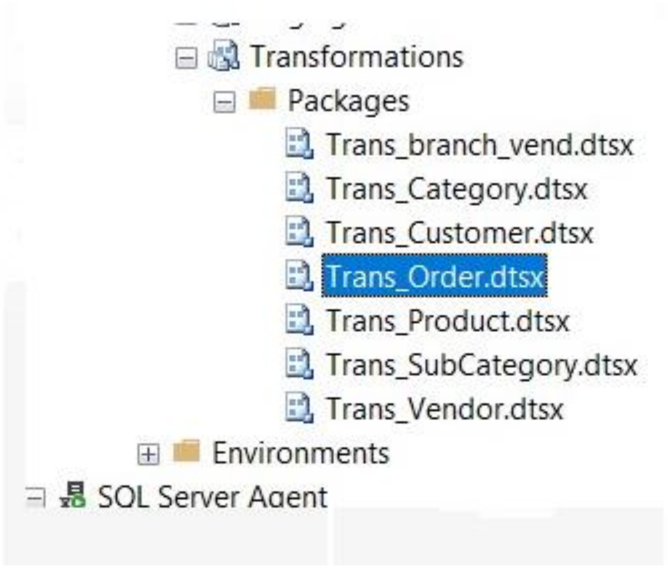
Job Properties - transformations

Select a page
- General
- Steps
- Schedules
- Alerts
- Notifications
- Targets

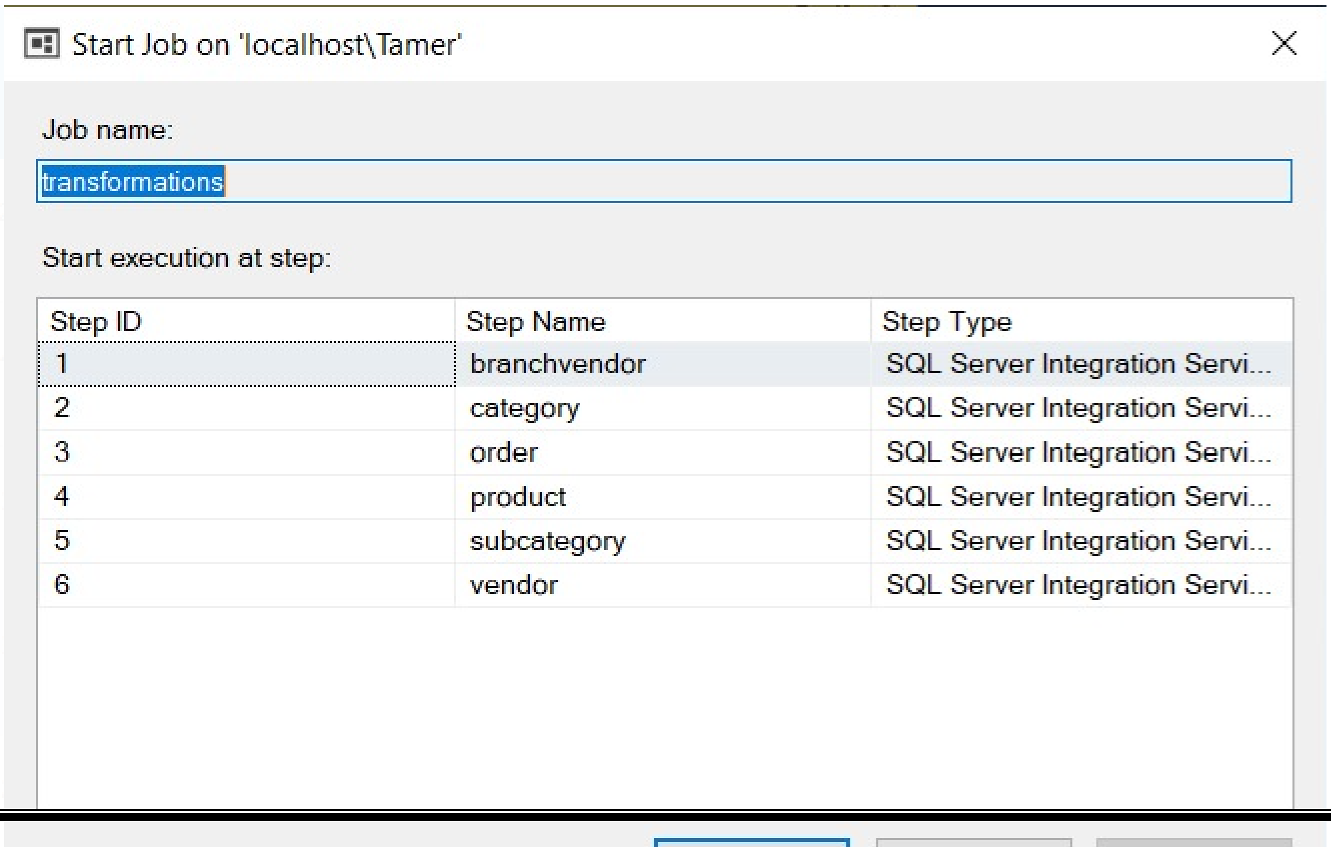
Script · Help

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'

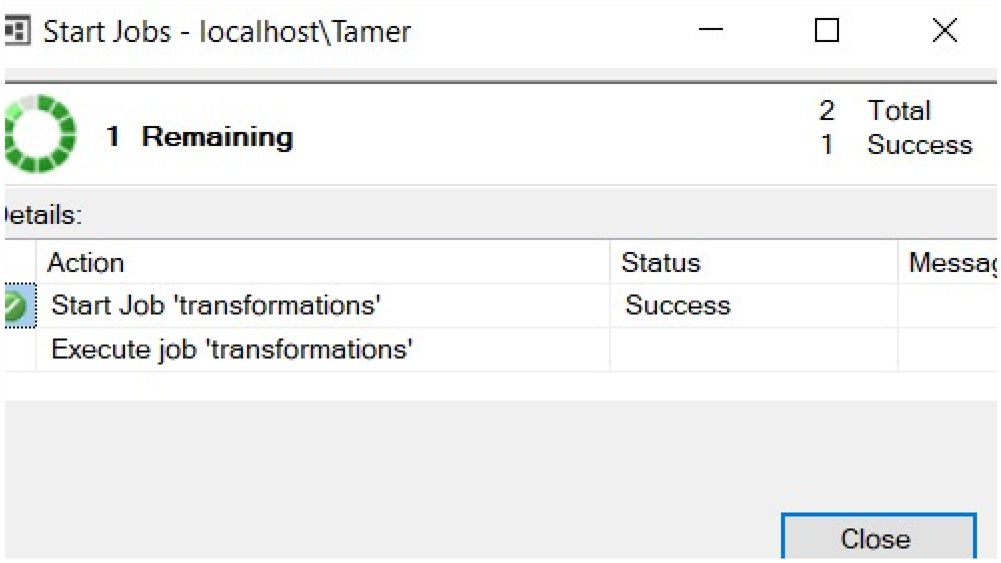
Job name:

transformations

Start execution at step:

Step ID	Step Name	Step Type
1	branchvendor	SQL Server Integration Servi...
2	category	SQL Server Integration Servi...
3	order	SQL Server Integration Servi...
4	product	SQL Server Integration Servi...
5	subcategory	SQL Server Integration Servi...
6	vendor	SQL Server Integration Servi...

Job Start:

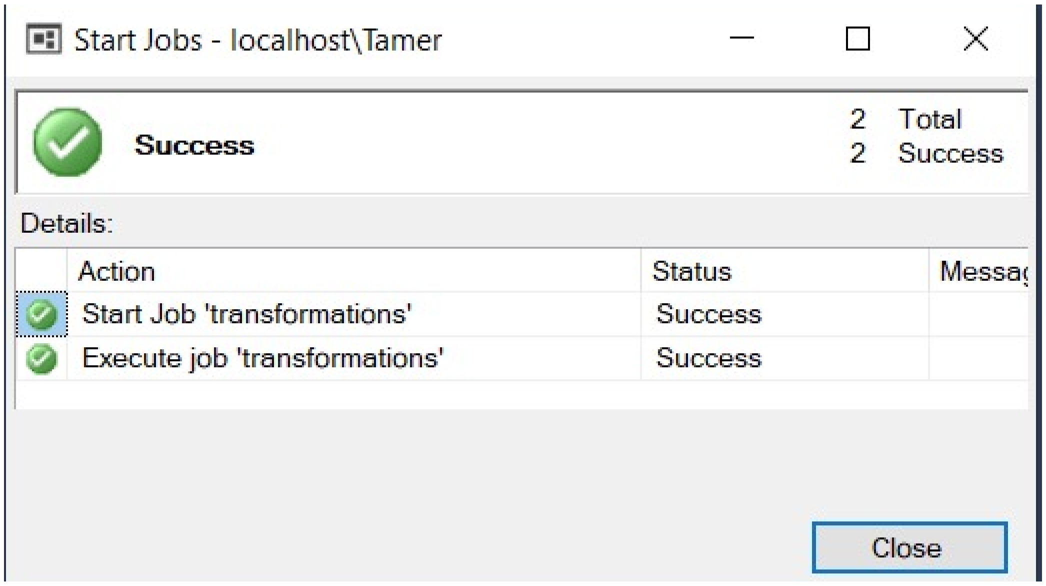

Start Jobs - localhost\Tamer

1 Remaining

2 Total
1 Success

etails:

	Action	Status	Messa
✓	Start Job 'transformations'	Success	
	Execute job 'transformations'		


Start Jobs - localhost\Tamer

Success

2 Total
2 Success

Details:

	Action	Status	Messa
✓	Start Job 'transformations'	Success	
✓	Execute job 'transformations'	Success	

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

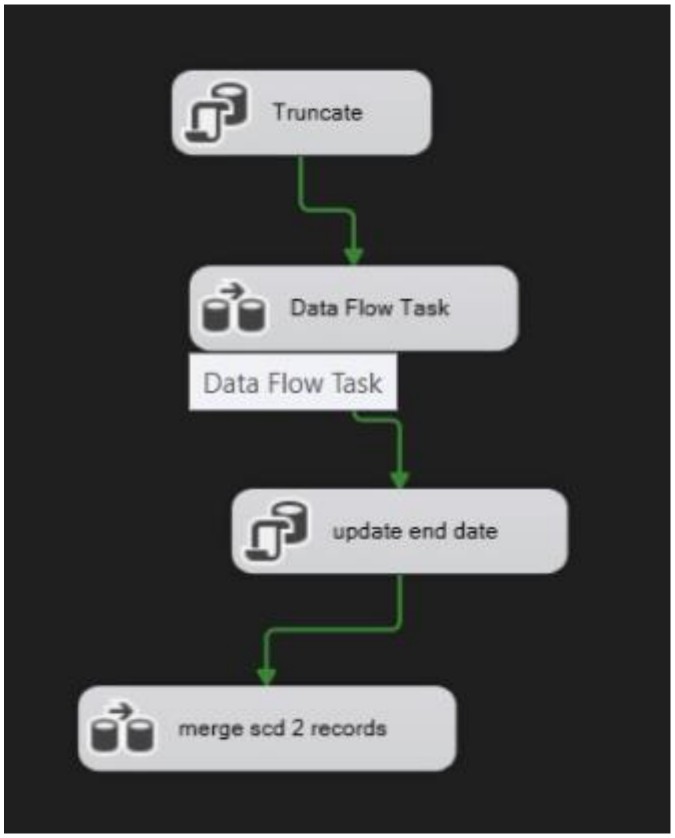
The ETL process for the staging database was designed to maintain a historical record of all changes made to the data in the landing database. This was achieved by implementing a sample ETL package with multiple components, including two SQL task components and two data flow tasks.

Staging Logic and Historical Data Tracking

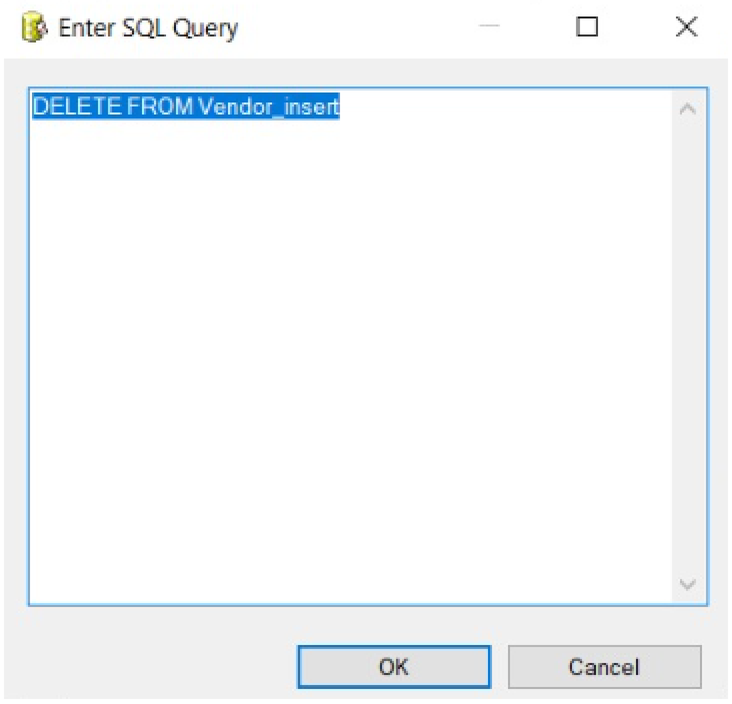
To effectively track changes, we added three columns to our staging tables: StartDate, EndDate, and IsCurrent:

- **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:

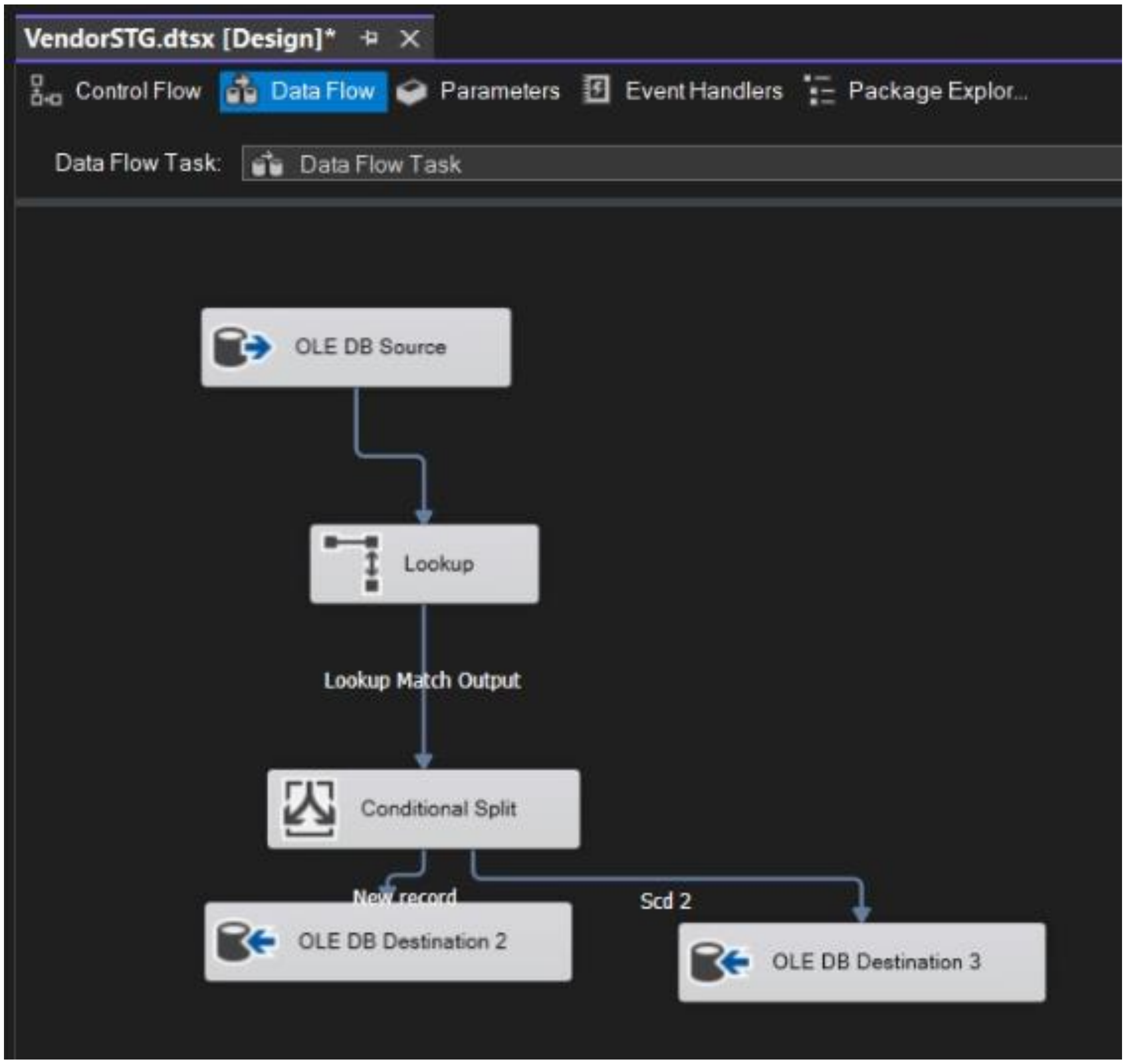


1. **Initial Truncation:**
   - The process begins by truncating a temporary table, table_inserted, which stores records temporarily before updating the staging database.

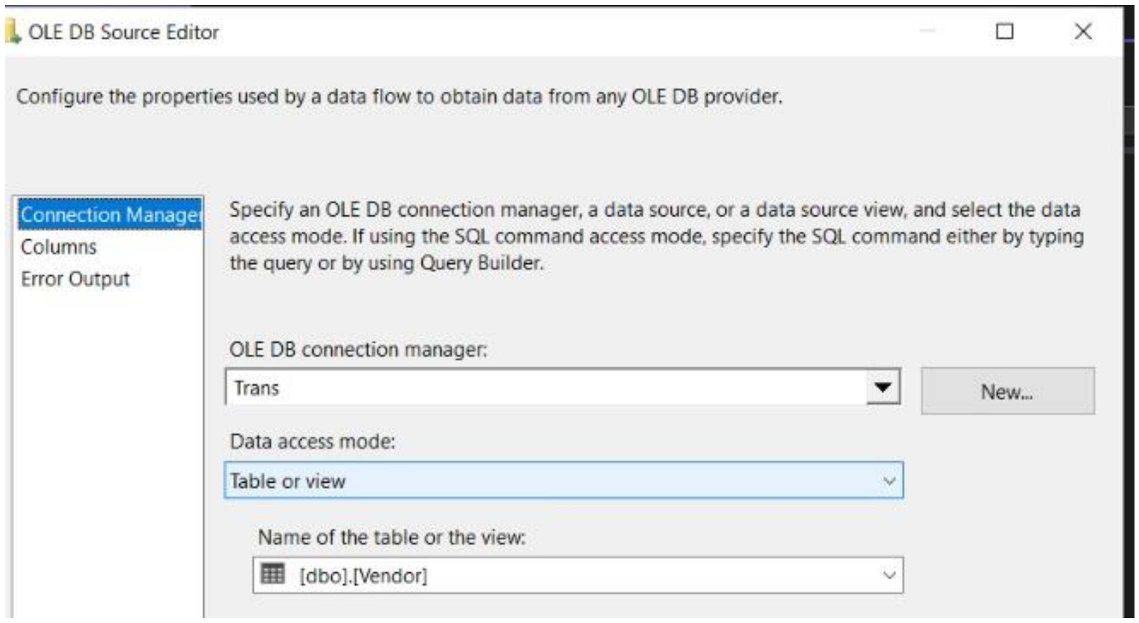


```
DELETE FROM Vendor_insert
```

2. **First Data Flow Task:**



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



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

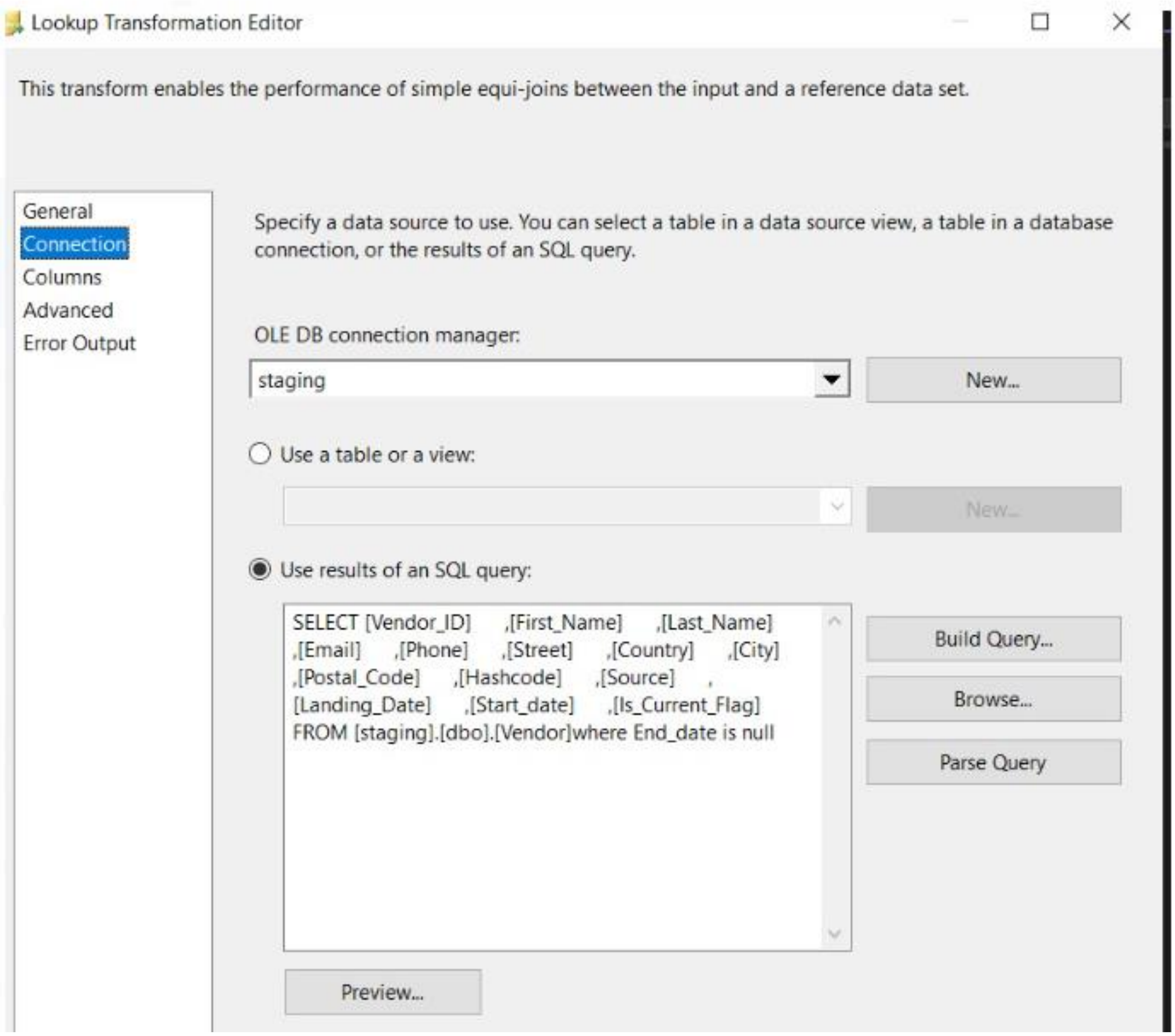
Specify an OLE DB connection manager, a data source, or a data source view, and select the data access mode. If using the SQL command access mode, specify the SQL command either by typing the query or by using Query Builder.

OLE DB connection manager: Trans

Data access mode: Table or view

Name of the table or the view: [dbo].[Vendor]

- 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.

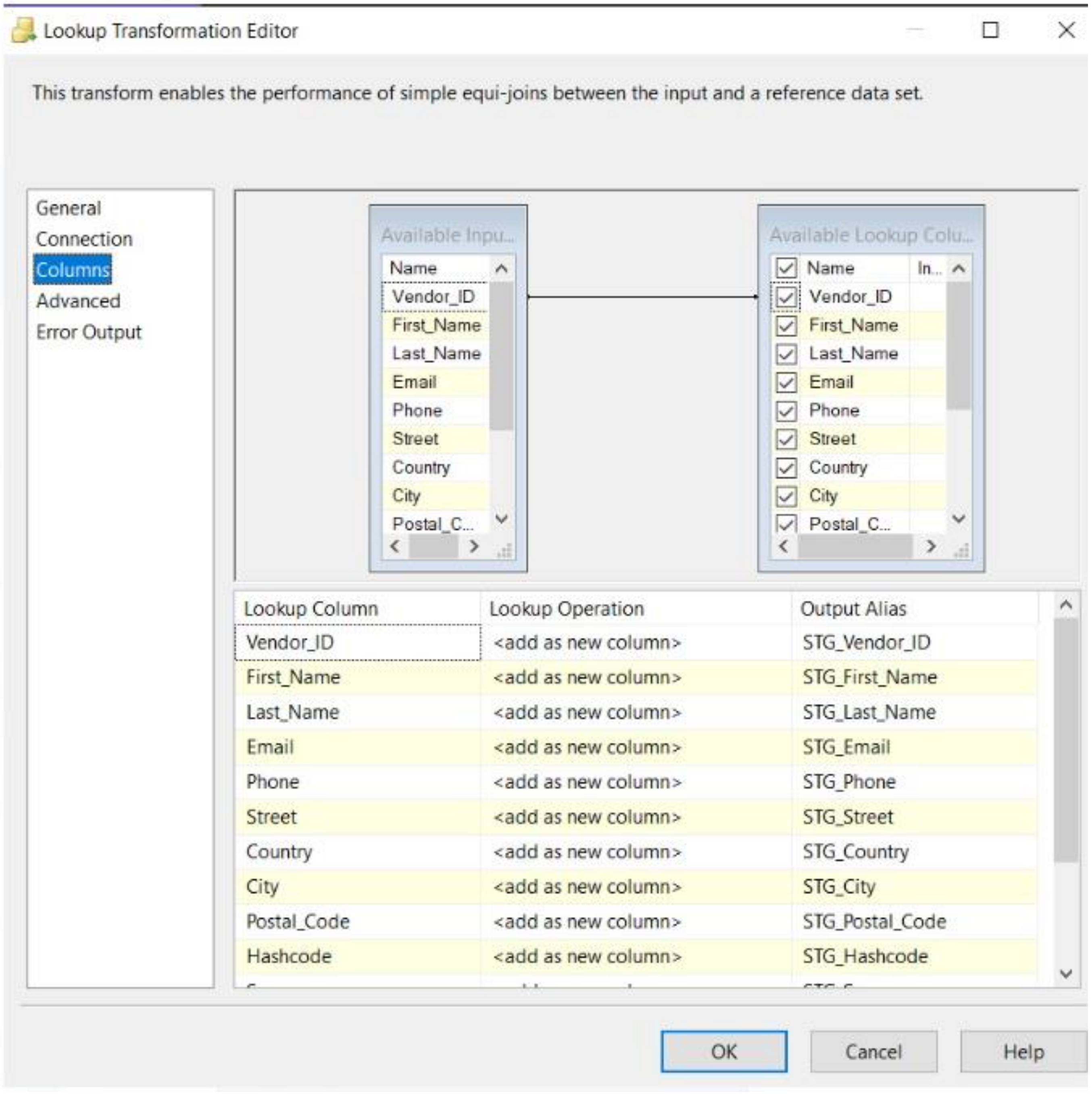
Specify a data source to use. You can select a table in a data source view, a table in a database connection, or the results of an SQL query.

OLE DB connection manager: staging

[ ] Use a table or a view:

[✓ Use results of an SQL query:]

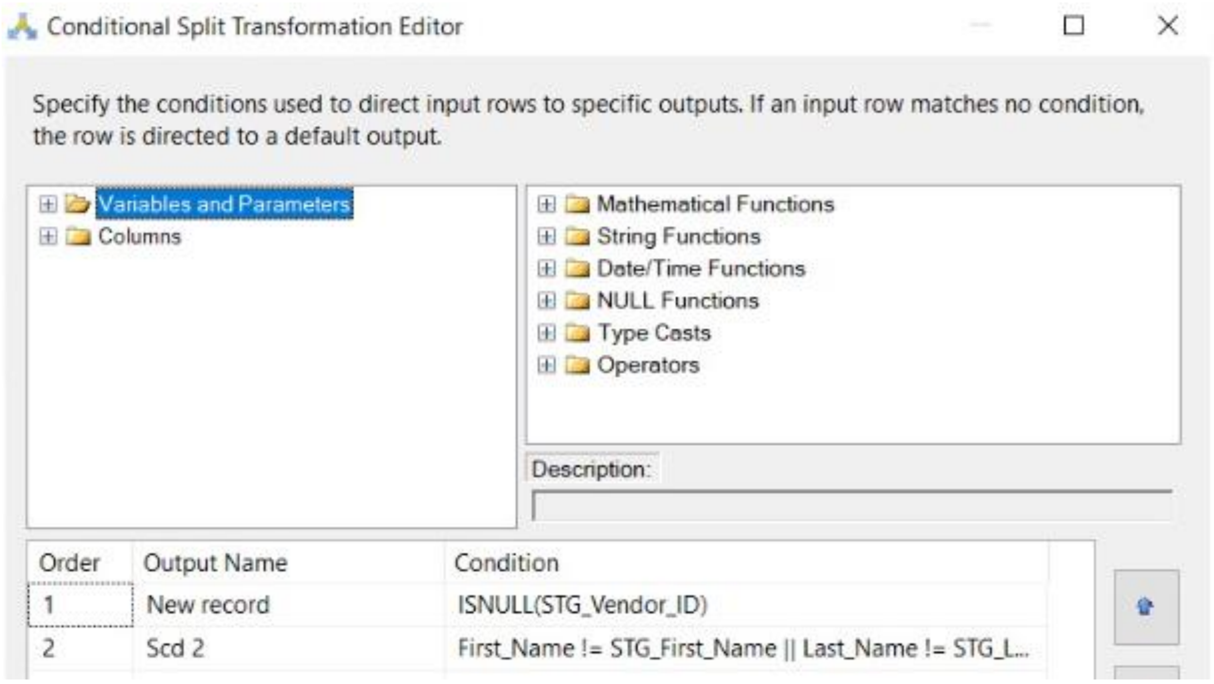
```
SELECT [Vendor_ID]      ,[First_Name]      ,[Last_Name]
,[Email]      ,[Phone]      ,[Street]      ,[Country]      ,[City]
,[Postal_Code]      ,[Hashcode]      ,[Source]      ,
[Landing_Date]      ,[Start_date]      ,[Is_Current_Flag]
FROM [staging].[dbo].[Vendor]where End_date is null
```


Lookup Transformation Editor

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

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

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

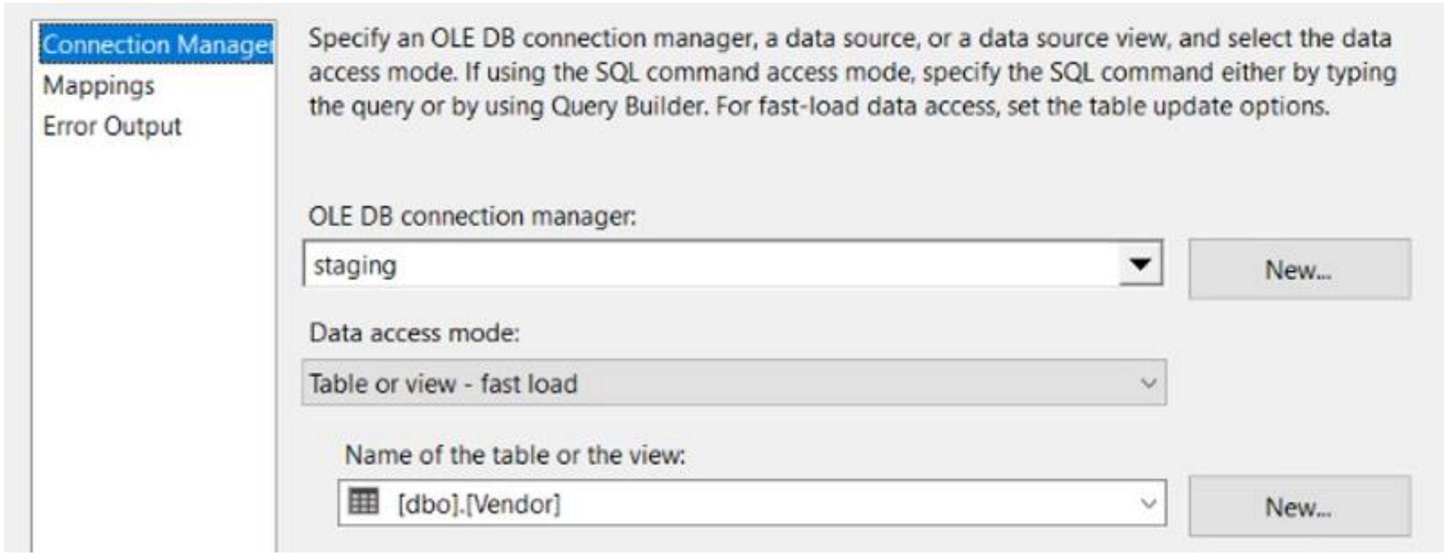


Conditional Split Transformation Editor

Specify the conditions used to direct input rows to specific outputs. If an input row matches no condition, the row is directed to a default output.

Order	Output Name	Condition
1	New record	ISNULL(STG_Vendor_ID)
2	Scd 2	First_Name != STG_First_Name || Last_Name != STG_L...

  - **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.

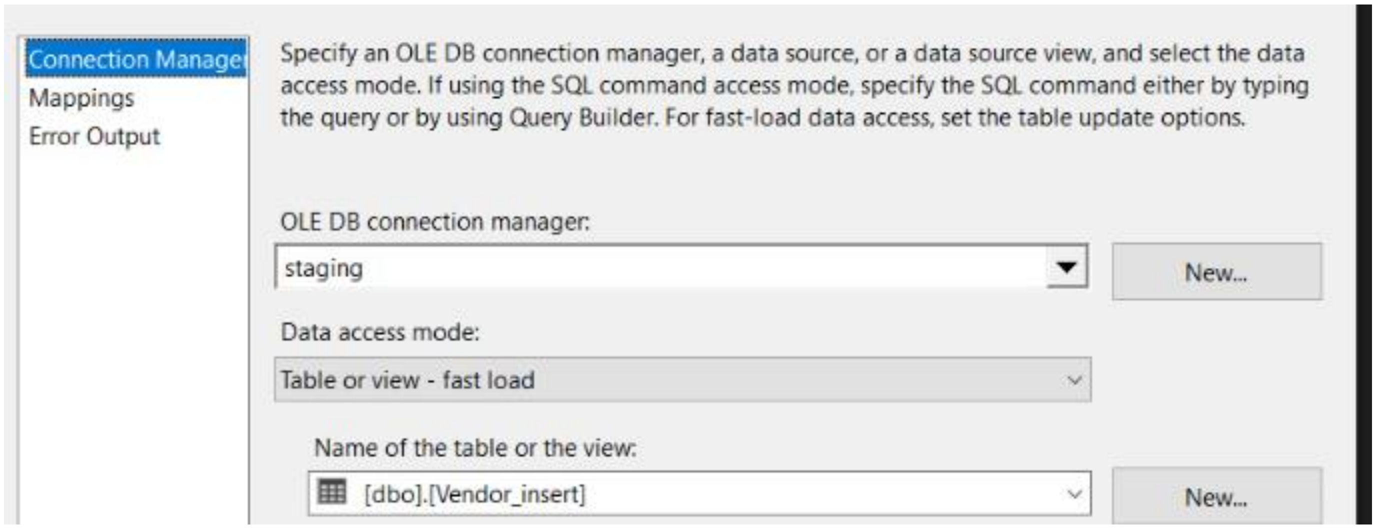


Specify an OLE DB connection manager, a data source, or a data source view, and select the data access mode. If using the SQL command access mode, specify the SQL command either by typing the query or by using Query Builder. For fast-load data access, set the table update options.

OLE DB connection manager: staging

Data access mode: Table or view - fast load

Name of the table or the view: [dbo].[Vendor]



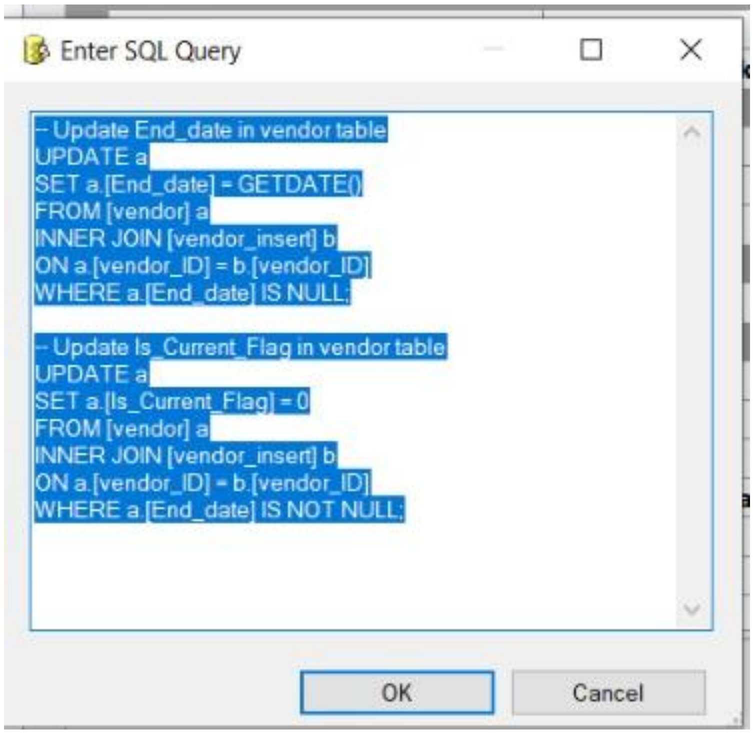
Specify an OLE DB connection manager, a data source, or a data source view, and select the data access mode. If using the SQL command access mode, specify the SQL command either by typing the query or by using Query Builder. For fast-load data access, set the table update options.

OLE DB connection manager: staging

Data access mode: Table or view - fast load

Name of the table or the view: [dbo].[Vendor_insert]

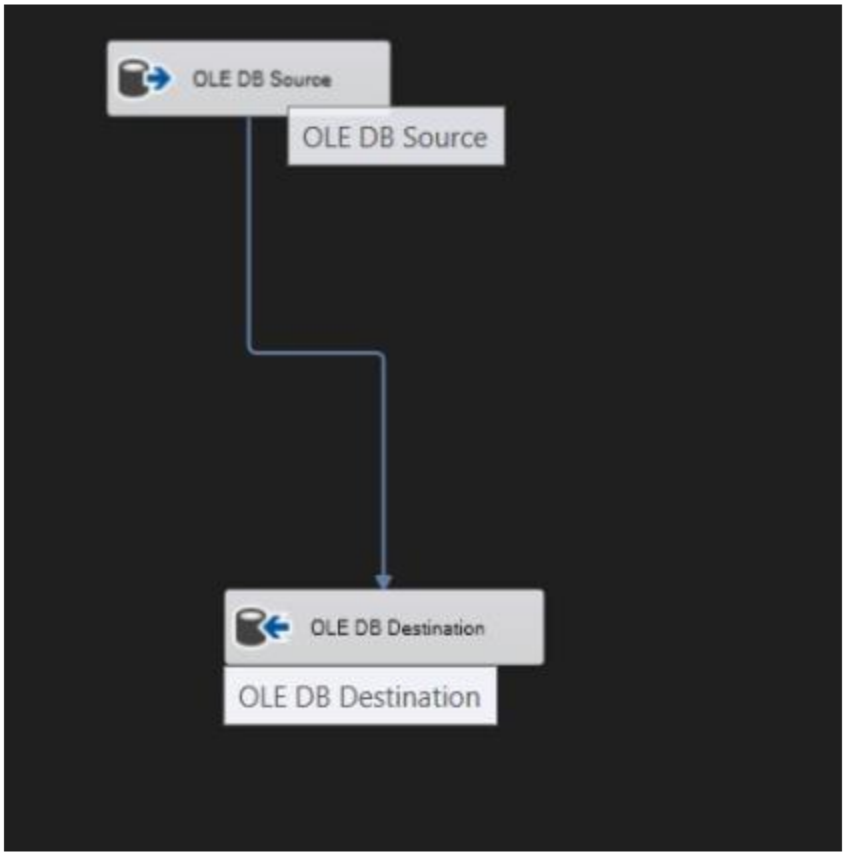
3. **Updating Historical Data:**
   - A **SQL Task Component** updates the old records in the staging table:
     - The EndDate of the existing (now outdated) record is updated to the start date of the new record.
     - The IsCurrent flag is set to 0 to indicate that the record is no longer valid.

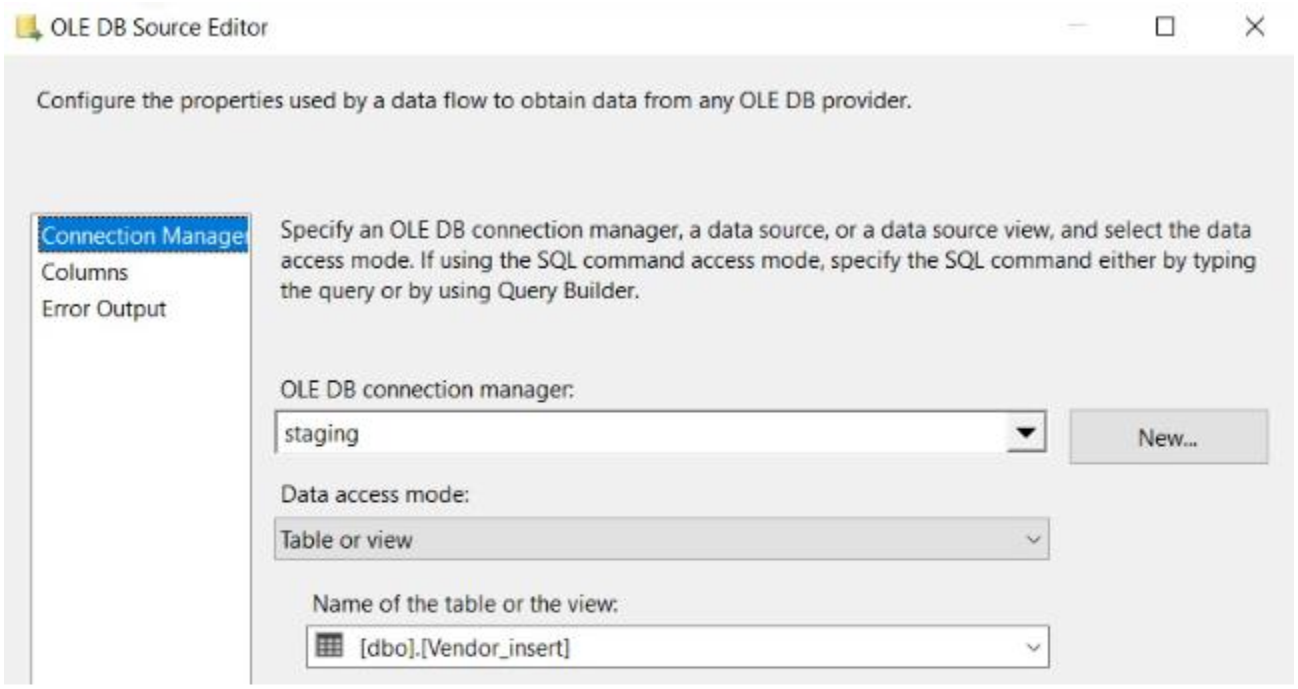


```
-- Update End_date in vendor table
UPDATE a
SET a.[End_date] = GETDATE()
FROM [vendor] a
INNER JOIN [vendor_insert] b
ON a.[vendor_ID] = b.[vendor_ID]
WHERE a.[End_date] IS NULL;

-- Update Is_Current_Flag in vendor table
UPDATE a
SET a.[Is_Current_Flag] = 0
FROM [vendor] a
INNER JOIN [vendor_insert] b
ON a.[vendor_ID] = b.[vendor_ID]
WHERE a.[End_date] IS NOT NULL;
```

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.




OLE DB Source Editor

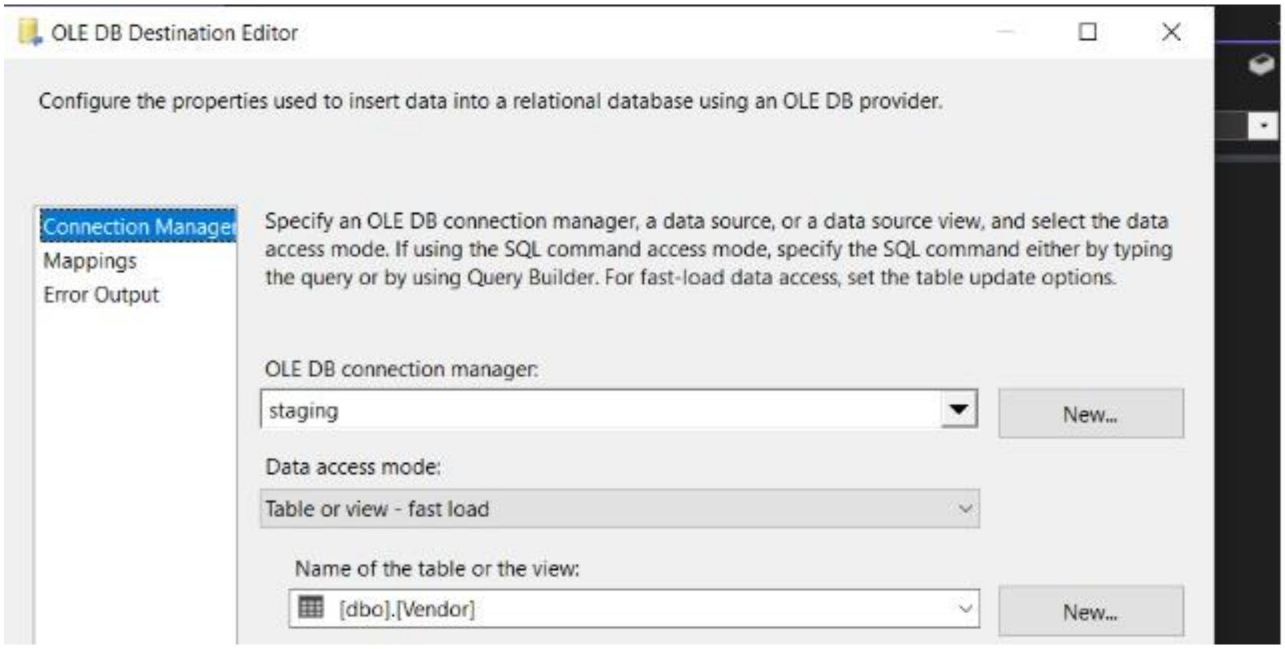
Configure the properties used by a data flow to obtain data from any OLE DB provider.

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: staging

Data access mode: Table or view

Name of the table or the view: [dbo].[Vendor_insert]



OLE DB Destination Editor

Configure the properties used to insert data into a relational database using an OLE DB provider.

Connection Manager
Mappings
Error Output

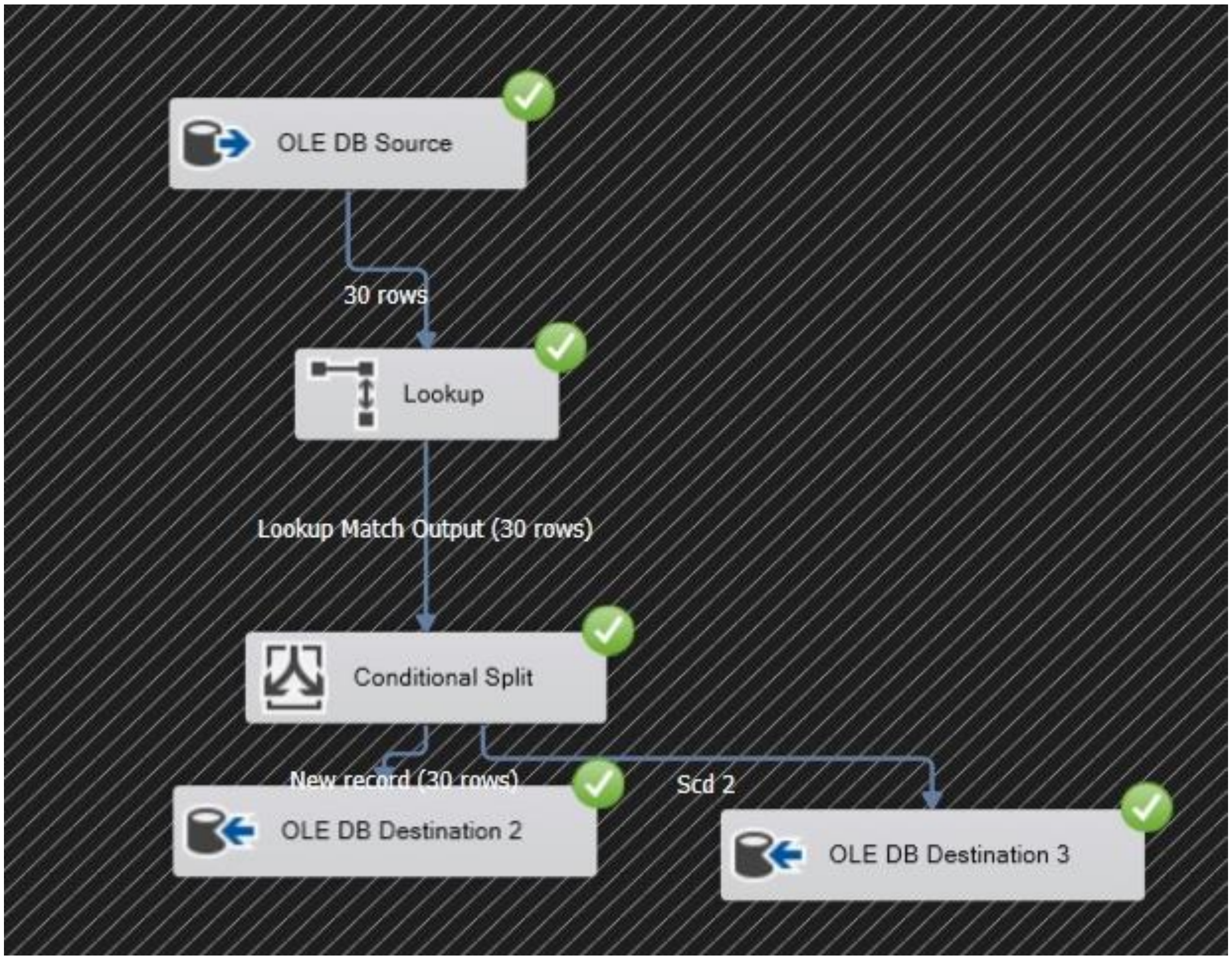
Specify an OLE DB connection manager, a data source, or a data source view, and select the data access mode. If using the SQL command access mode, specify the SQL command either by typing the query or by using Query Builder. For fast-load data access, set the table update options.

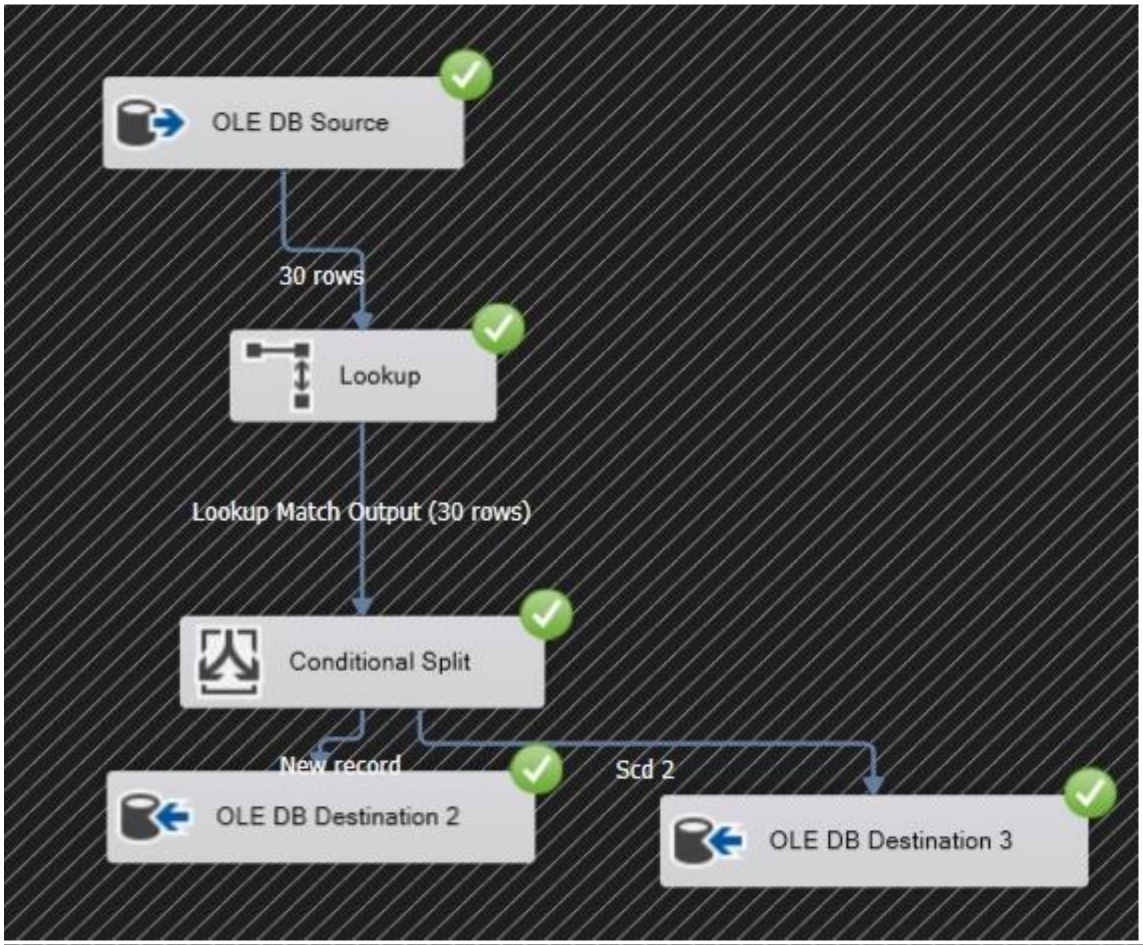
OLE DB connection manager: staging

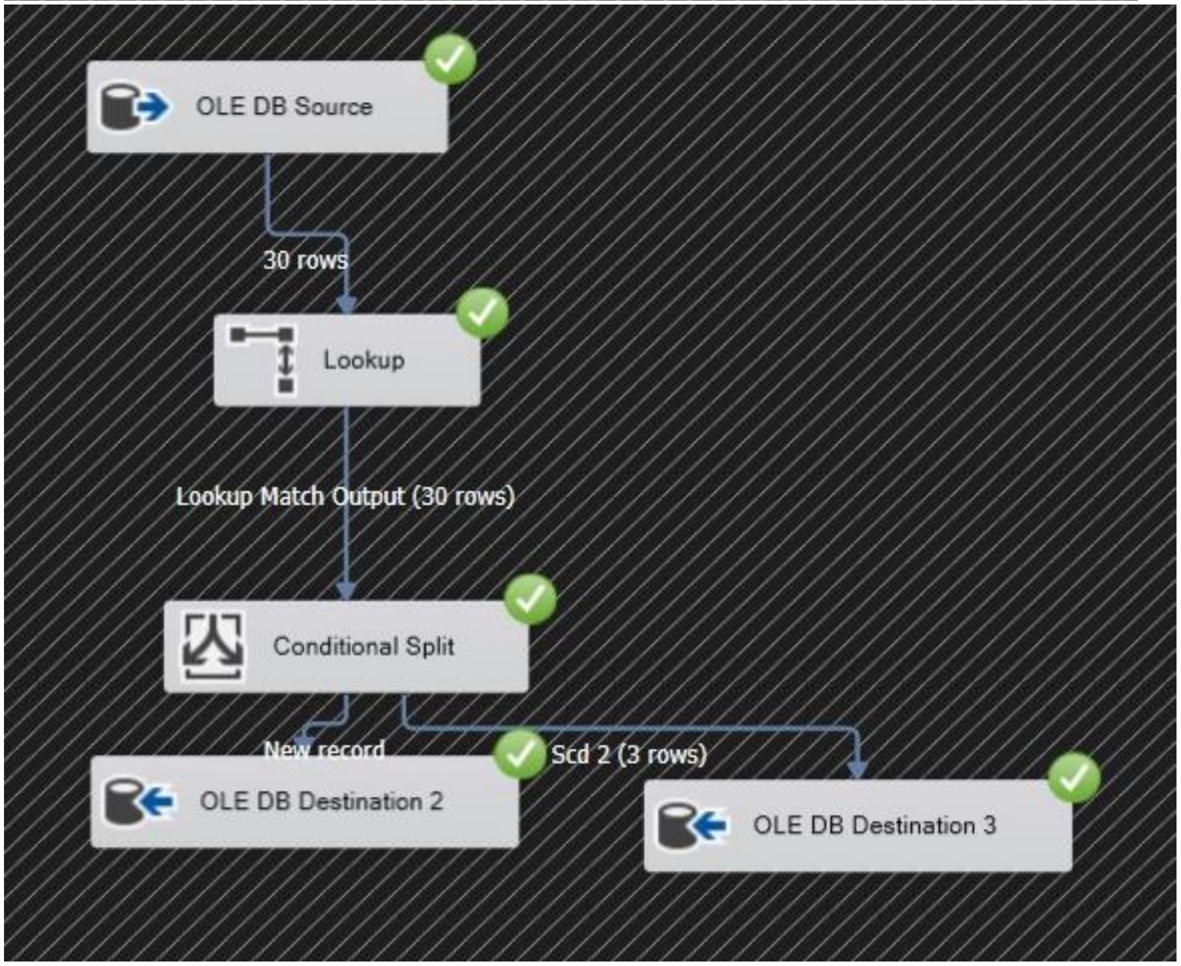
Data access mode: Table or view - fast load

Name of the table or the view: [dbo].[Vendor]

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

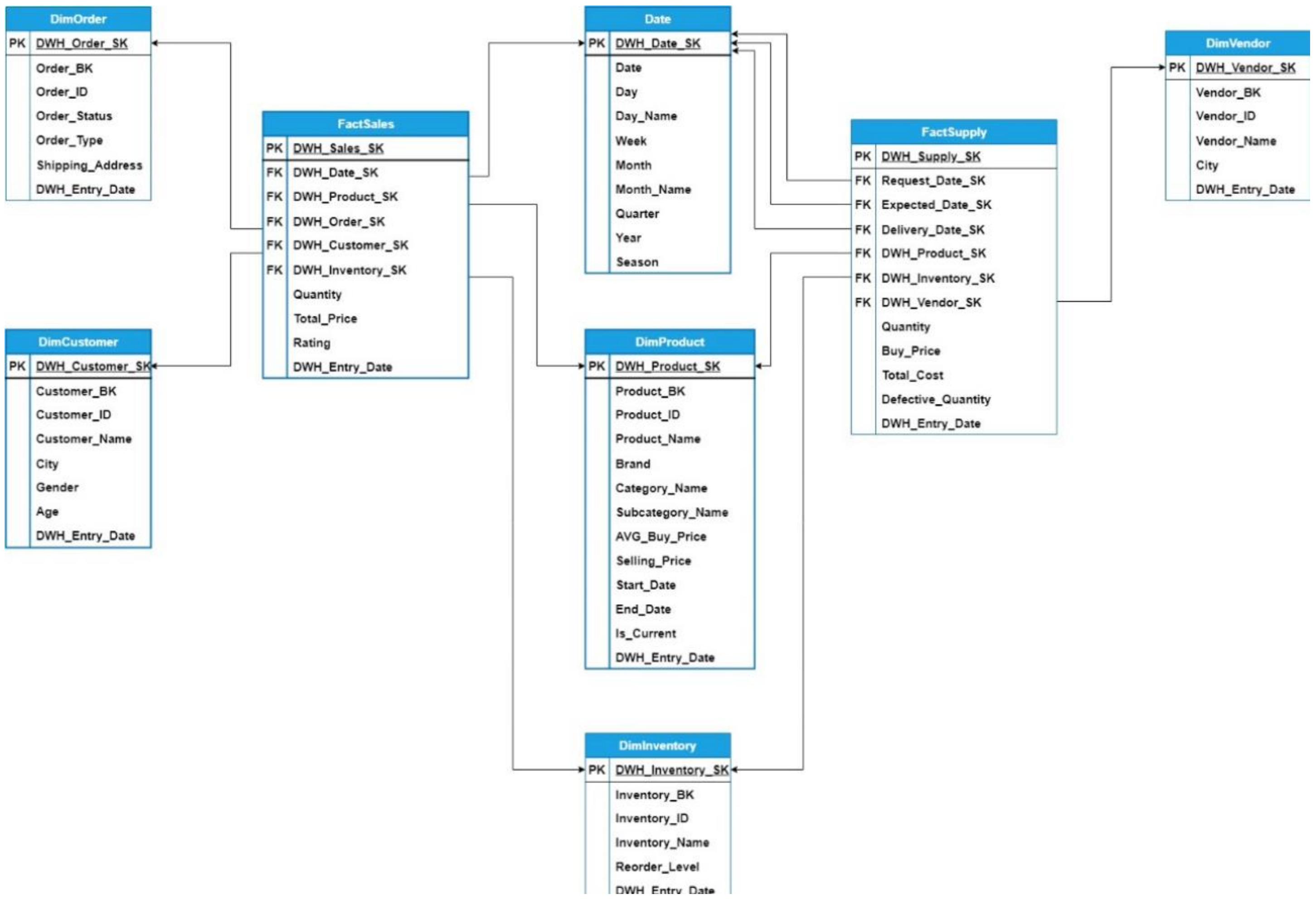

OLE DB Source
30 rows
Lookup
Lookup Match Output (30 rows)
Conditional Split
New record
Scd 2 (3 rows)
OLE DB Destination 2
OLE DB Destination 3

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.

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.