14/10/2023

S23
DataWarehousing & ETL
Project Report

Contents

Introduction

In the process of using data and extracting value, the key step is to integrate various data into a unified IT system. This integration requires the unification and standardization of data from varied sources, making it usable for various applications. One potential solution to this challenge is the implementation of a Data Warehouse.

In this project, we are going to design and implement a Data Warehouse with the call center data of ServiceSpot, a leading IT company.

Our mission is to construct a robust ETL (Extract, Transform, Load) project and merge ServiceSpot's disparate data sources into a cohesive enterprise data warehouse. For this, we will use SQL Server and SSIS.

Data

Our dataset comprises 6 distinct CSV files.

1. The “Employees.csv” provides information about employee records with specific information about their identity, work location, and supervising manager. The data within this file is structured as follows:

- **EmployeeID:** Unique identifier for each employee.
- **EmployeeName:** Full name of the employee.
- **Site:** Name of the site where the employee is stationed.
- **ManagerName:** Name of the employee's manager.

2. The “Call Types.csv”, each entry in this dataset delineates a distinct call type, categorized by its unique identifier, and accompanied by a clear label defining its purpose or category. It contains the following data :

- **CallTypeID:** Represents a unique identifier assigned to each specific call type.
- **CallTypeLabel:** Provides a descriptive label for each call type, indicating the nature or purpose of the call.

The “Call Charges.csv” provides detailed information about the charges applied to customers for different call types, allowing for precise financial analysis based on call duration and type, and is segmented by individual years :

- **Call Type Key:** Represents a unique identifier specific to each call type.
- **Call Type:** Corresponds to the label indicating the nature or purpose of the call.

- **Call Charges / Min (YYYY):** Denotes the monetary amount charged to customers for every minute spent on the phone. This value varies for each specific call type and is specific to the respective year denoted as (YYYY).

The “US States.csv” provides comprehensive information about various states within the United States, aach row in this dataset represents a specific state, with detailed information about its code, full name, and regional classification, allowing for efficient categorization and analysis based on geographical region:

- **StateCD:** This column contains the 2-letter state codes, which are standardized abbreviations representing each state.
- **Name:** This column lists the full names of the states.
- **Region:** The 'Region' column specifies the broader geographical categorization of each state within the United States, such as East, West, and so on.

The Calls Data files, the crucial files “Data 2018.csv”, “Data 2019.csv”and “Data 2020.csv”. The dataset is structured across three separate CSV files, each corresponding to a specific year (2018, 2019, and 2020) and containing detailed call information. Each row within these files represents a specific call record, containing essential details and provide a comprehensive view of customer interactions:

- **CallTimestamp:** This column records the date and time of each call, capturing the precise moment when the call occurred.
- **Call Type:** Specifies the nature or purpose of the call, differentiating between various types of inquiries or interactions.
- **EmployeeID:** Each call is associated with a unique identifier for the employee who handled the call, providing a reference point for staff performance analysis.
- **CallDuration:** Represents the duration of each call in seconds, indicating the total time spent on the call, including conversations and any additional tasks.
- **WaitTime:** Captures the duration customers spent waiting before their call was answered, a key metric for evaluating service efficiency.
- **CallAbandoned:** This binary column indicates whether the call was abandoned by the customer. A value of 1 signifies 'Yes,' indicating the call was abandoned, while 0 denotes 'No,' indicating the call was not abandoned.

Pipeline Design

The pipeline will be done in three steps:

- The Staging area (STA) will allow loading the data as is, or with minimal changes.

- The Operational Data Store area (ODS) will allow cleaning and standardizing the data. If the data don't pass the quality criteriums, they will be put in the "Technical_Rejects" table as technical rejects.

- The Data Warehouse area (DWH) will organize the data in one fact table related to multiple dimensions tables. If records cannot be integrated in the schema, they will be put in the "Functional_Rejects" table as functional reject. Alternatively, some placeholder relations can be created.

There will be one STA and one ODS package per file. Except for the Calls Data files, for the three separate years (2018, 2019 and 2020) the data is consolidated into a single set of STA and ODS packages, optimizing the efficiency of the ETL process while guaranteeing data integrity and consistency of analysis over several years.

Staging database

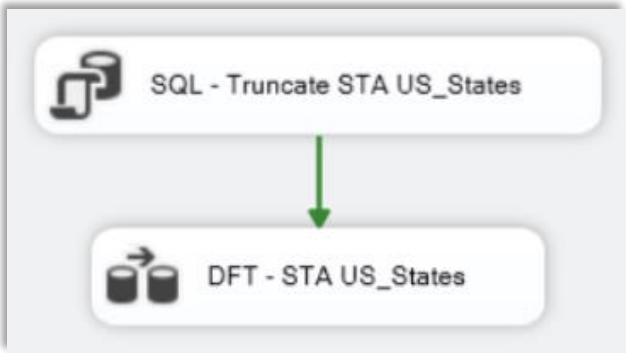
Here, the role of the staging database is to store all the data coming from the different sources. We want to accept all available data.

US States Table

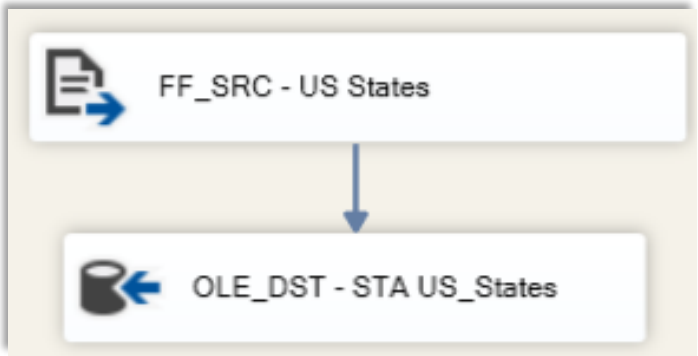
Below is an extract of the data in “US States.csv” file.

StateCD	Name	Region
AK	Alaska	West
AL	Alabama	South
AR	Arkansas	South
AZ	Arizona	West
CA	California	West
CO	Colorado	West
CT	Connecticut	Northeast
DC	District of Columbia	South
DE	Delaware	South
FL	Florida	South
GA	Georgia	South
HI	Hawaii	West
IA	Iowa	Midwest
ID	Idaho	West
IL	Illinois	Midwest

For this CSV file we have only one table and before running the package we have to truncate the table “US States”:



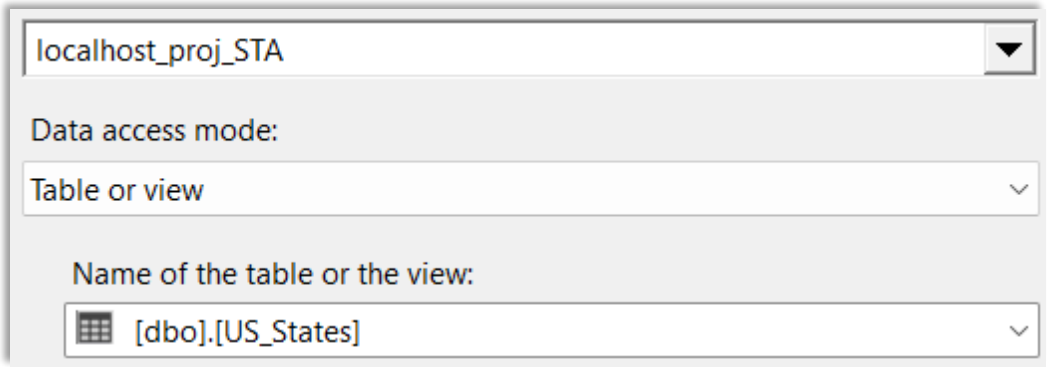
The dataflow is an import of a flat file:



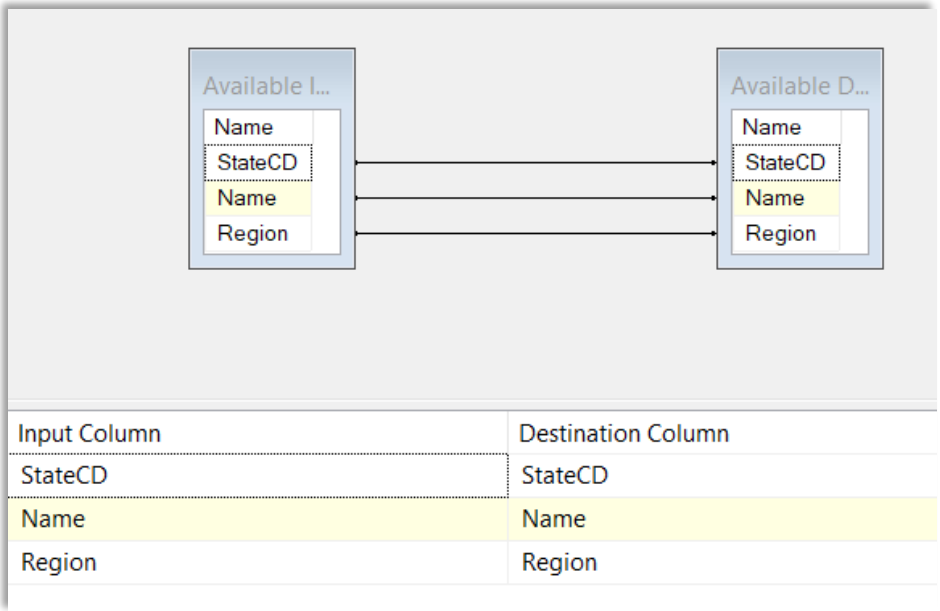
We need to create the destination table with the following command:

```
USE [proj_STA]
GO

/****** Object:  Table [dbo].[US_States]    Script Date: 10/14/2023 5:24:55 PM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[US_States](
    [StateCD] [varchar](255) NULL,
    [Name] [varchar](255) NULL,
    [Region] [varchar](255) NULL
) ON [PRIMARY]
GO
```

The data is then loaded into the table “US States”:



We need to make sure if any changes are needed or my tables are well mapped to avoid any errors:



Input Column	Destination Column
StateCD	StateCD
Name	Name
Region	Region

Here is the first 10 lines of the result:

	StateCD	Name	Region
1	AK	Alaska	West
2	AL	Alabama	South
3	AR	Arkansas	South
4	AZ	Arizona	West
5	CA	California	West
6	CO	Colorado	West
7	CT	Connecticut	Northeast
8	DC	District of Columbia	South
9	DE	Delaware	South
10	FL	Florida	South

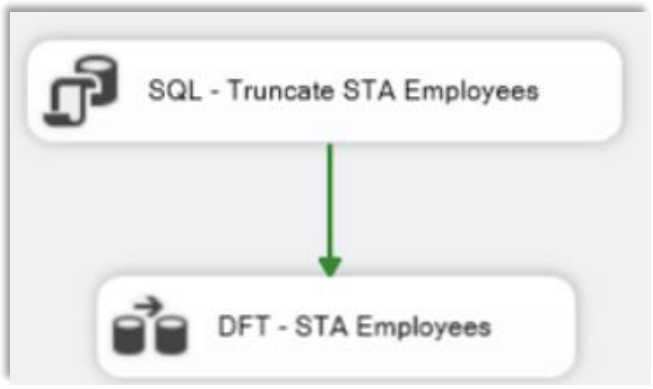
Employees Table

Below is an extract of the data in “Employees.csv” file.

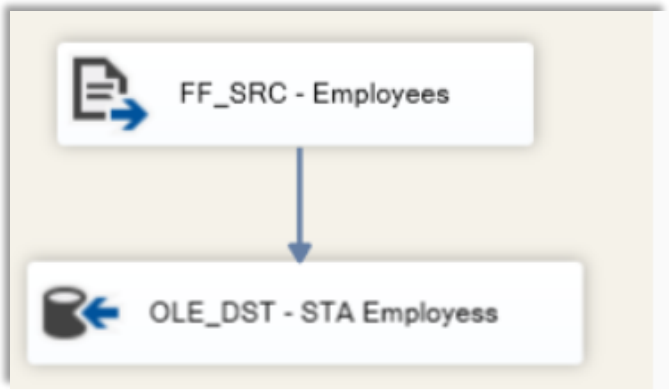
EmployeeID	EmployeeName	Site	ManagerName
N772493	Onita Trojan	Spokane, WA	Deidre Robbs
F533051	Stormy Seller	Aurora, CO	Elsie Taplin
S564705	Mable Ayoub	Aurora, CO	Shala Lion
I281837	Latrisha Buckalew	Aurora, CO	Rana Taub
Y193775	Adrianna Duque	Spokane, WA	Collin Trotman
J632516	Keiko Daulton	Spokane, WA	Jamar Prahl

G727038	Dolores Lundeen	Aurora, CO	Shala Lion

For this CSV file we have only one table and before running the package we have to truncate the table “Employees.csv”:



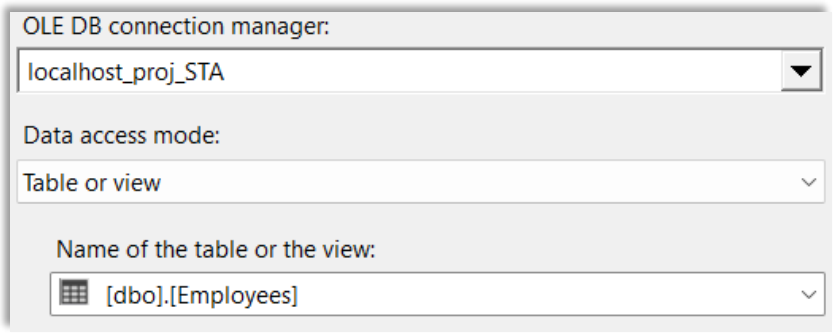
The dataflow is an import of a flat file :



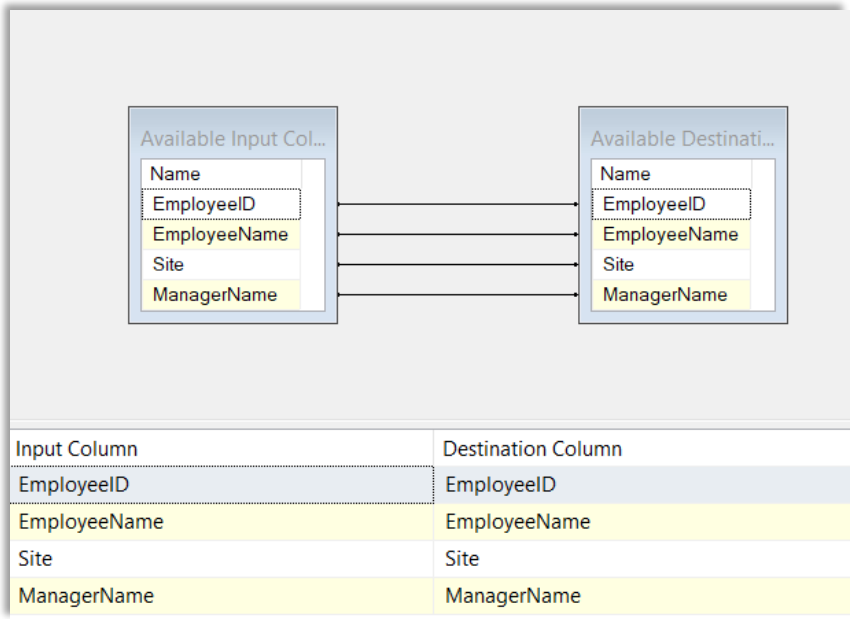
We need to create the destination table with the following command:

```
USE [proj_STA]
GO

/****** Object:  Table [dbo].[Employees]    Script Date: 10/14/2023 5:26:59 PM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Employees](
    [EmployeeID] [nvarchar](255) NULL,
    [EmployeeName] [nvarchar](255) NULL,
    [Site] [nvarchar](255) NULL,
    [ManagerName] [nvarchar](255) NULL
) ON [PRIMARY]
GO
```

The data is then loaded into the table “Employees”:



We need to make sure if any changes are needed or my tables are well mapped to avoid any errors:



Input Column	Destination Column
EmployeeID	EmployeeID
EmployeeName	EmployeeName
Site	Site
ManagerName	ManagerName

Here is the first 10 lines of the result:

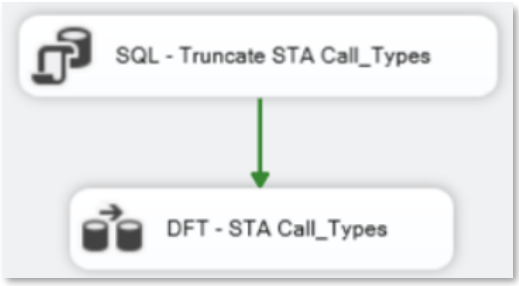
	EmployeeID	EmployeeName	Site	ManagerName
1	N772493	Onita Trojan	Spokane, WA	Deidre Robbs
2	F533051	Stormy Seller	Aurora, CO	Elsie Taplin
3	S564705	Mable Ayoub	Aurora, CO	Shala Lion
4	I281837	Latrisha Buckalew	Aurora, CO	Rana Taub
5	Y193775	Adrianna Duque	Spokane, WA	Collin Trotman
6	J632516	Keiko Daulton	Spokane, WA	Jamar Prahl
7	G727038	Dolores Lundeen	Aurora, CO	Shala Lion
8	V126561	Wilbur Mohl	Jacksonville, FL	Casey Bainbridge
9	E243130	Ileen Bornstein	Jacksonville, FL	Gonzalo Lesage
10	C206355	Janeth Roesler	Spokane, WA	Miyoko Degraw

Call Types Table

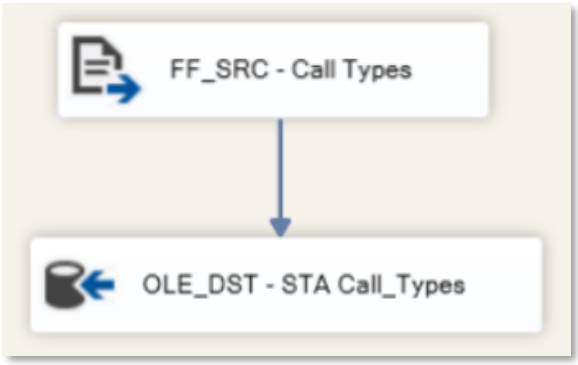
Below is an extract of the data in “Call Types.csv” file.

CallTypeID	CallTypeLabel
1	Sales
2	Billing
3	Tech Support

For this CSV file we have only one table and before running the package we have to truncate the table “Call Types1” :



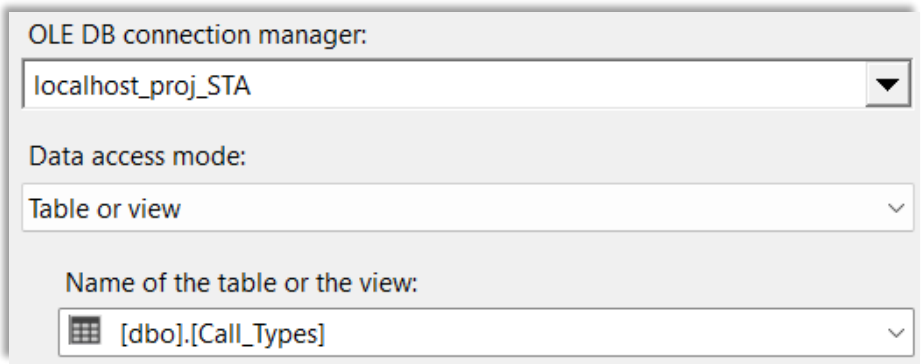
The dataflow is an import of a flat file:



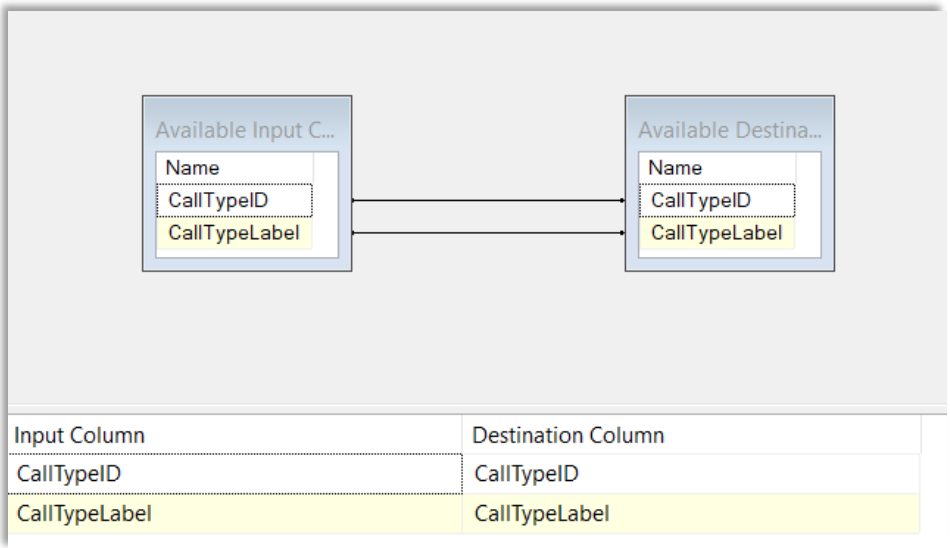
We need to create the destination table with the following command:

```
USE [proj_STA]
GO

/****** Object:  Table [dbo].[Call_Types]    Script Date: 10/14/2023 5:33:18 PM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Call_Types](
    [CallTypeID] [nvarchar](255) NULL,
    [CallTypeLabel] [nvarchar](255) NULL
) ON [PRIMARY]
GO
```

The data is then loaded into the table “Call Types”:



We need to make sure if any changes are needed or my tables are well mapped to avoid any errors:



Here is the result:

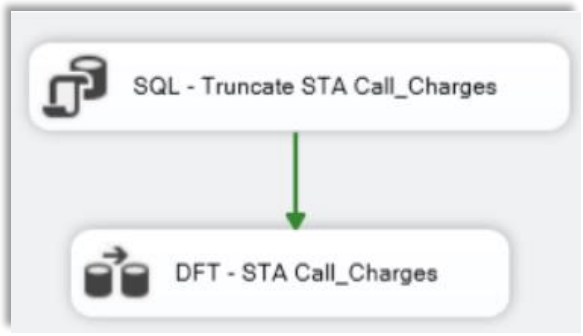
	CallTypeID	CallTypeLabel
1	1	Sales
2	2	Billing
3	3	Tech Support

Call Charges Table

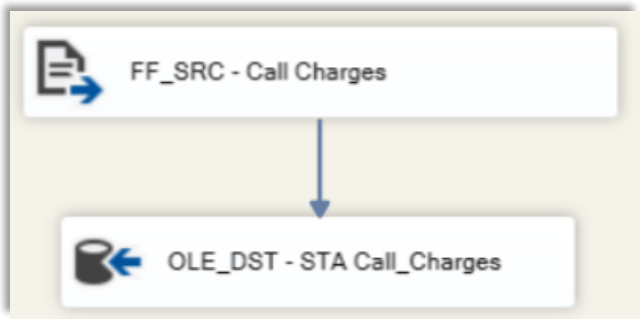
Below is an extract of the data in “Call Charges.csv” file.

Call Type	Call Charges (2018)	Call Charges (2019)	Call Charges (2020)	Call Charges (2021)
Sales	1..52 / min	1.56 / min	1.60 / min	1.71 / min
Billing	1.2 / min	1.32 / min	1.41 / min	1.45 / min
Tech Support	0.95 / min	0.98 / min	1.04 / min	1.12 / min

For this CSV file we have only one table and before running the package we have to truncate the table “Call Charges.csv” :



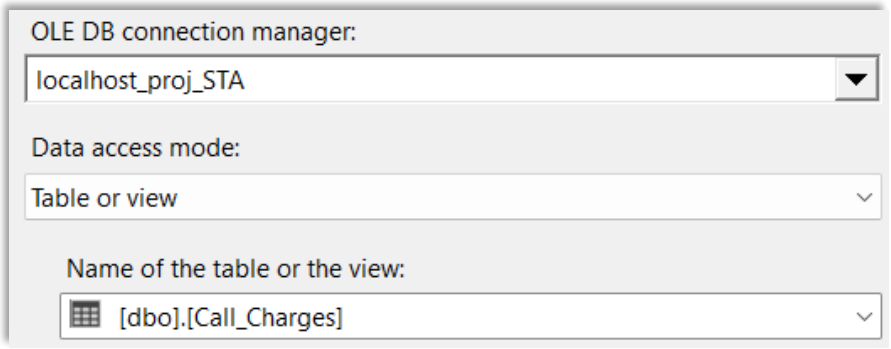
The dataflow is an import of a flat file :



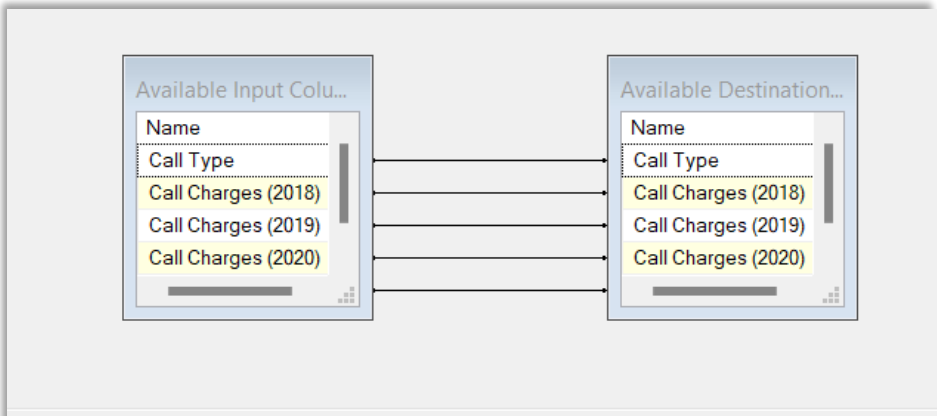
We need to create the destination table with the following command:

```
USE [proj_STA]
GO

/****** Object:  Table [dbo].[Call_Charges]    Script Date: 10/14/2023 5:34:58 PM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Call_Charges](
    [Call Type ] [nvarchar](255) NULL,
    [Call Charges (2018)] [nvarchar](255) NULL,
    [Call Charges (2019)] [nvarchar](255) NULL,
    [Call Charges (2020)] [nvarchar](255) NULL,
    [Call Charges (2021)] [nvarchar](255) NULL
) ON [PRIMARY]
GO
```

The data is then loaded into the table “Call_Charges”:



We need to make sure if any changes are needed or my tables are well mapped to avoid any errors:



Input Column	Destination Column
[Call Type]	Call Type
Call Charges (2018)	Call Charges (2018)
Call Charges (2019)	Call Charges (2019)
Call Charges (2020)	Call Charges (2020)
Call Charges (2021)	Call Charges (2021)

Here is the result:

	Call Type	Call Charges (2018)	Call Charges (2019)	Call Charges (2020)	Call Charges (2021)
1	Sales	1.52 / min	1.56 / min	1.60 / min	1.71 / min
2	Billing	1.2 / min	1.32 / min	1.41 / min	1.45 / min
3	Tech Support	0.95 / min	0.98 / min	1.04 / min	1.12 / min

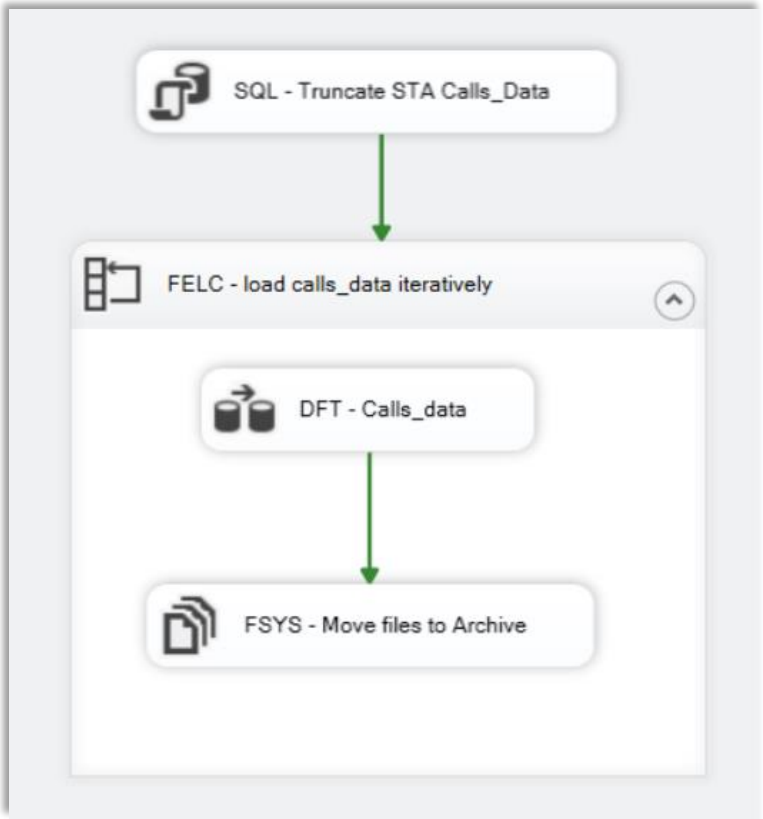
Calls data Table

Below is an example of the data in one of the three seperate “Call Charges” files:

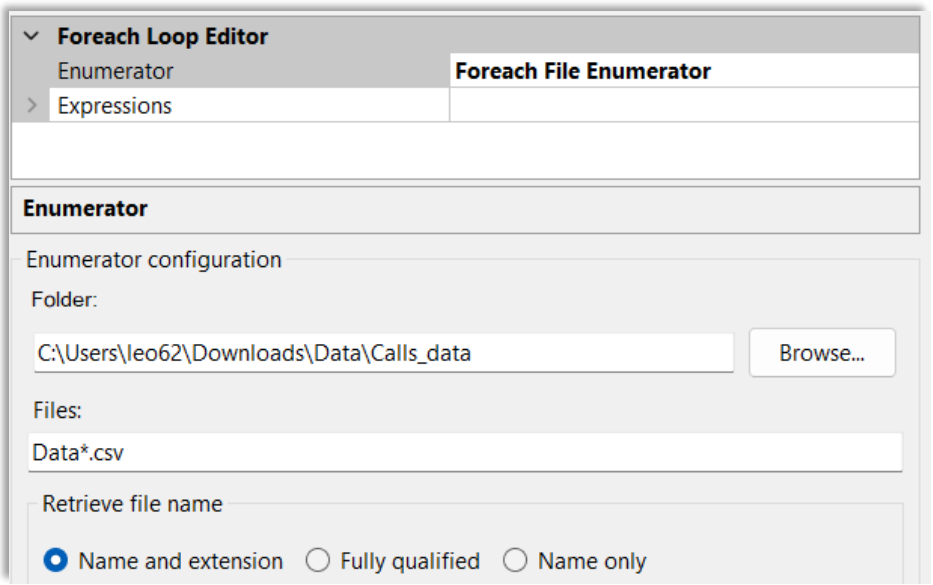
CallTimestamp	Call Type	EmployeeID	CallDuration	WaitTime	CallAbandoned
5/4/2018 16:33	3	U559430	486	2	0
6/21/2018 18:28	3	A166733	945	0	0
6/21/2018 15:13	1	B971624	379	11	0
11/21/2018 13:02	3	U641256	1044	0	0
2/25/2018 13:36	1	P286634	1357	0	0
10/28/2018 10:04	3	M855788	570	23	0

To be able to use the data in the files, we need to put it into one table. This means that we need to extract the data from each Excel file.

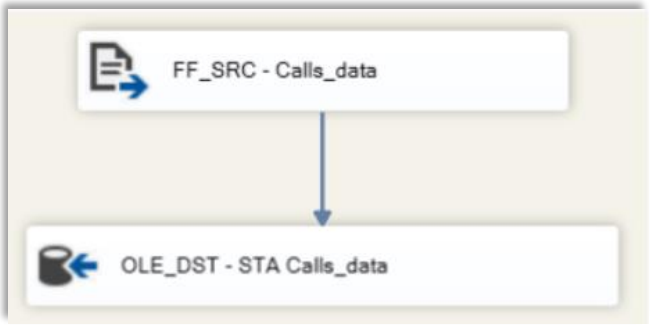
To go into all data sheets, we use a “Foreach Loop Container” where we put the extraction dataflow inside. The data flow in the container will be able to load one sheet at the time. Here, we also truncate the data from the previous runs.



We use the enumerator “Foreach File Enumerator” to be able to iterate over the 3 tables.



Next, the dataflow is defined as follow:

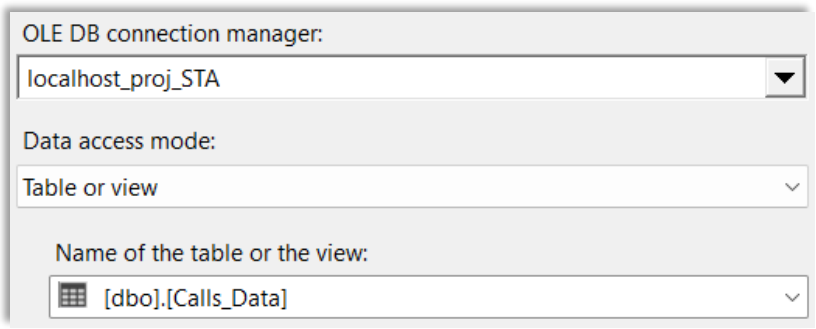


We need to create the destination table with the following command:

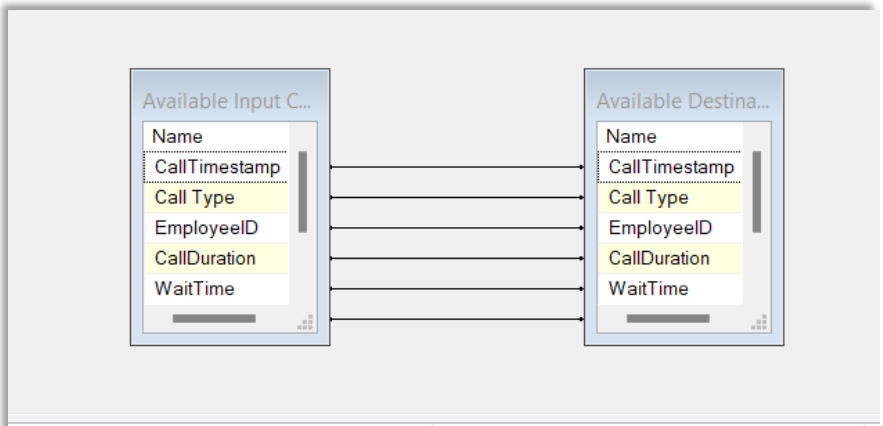
```
USE [proj_STA]
GO

/****** Object:  Table [dbo].[Calls_Data]    Script Date: 10/14/2023 5:36:41 PM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Calls_Data](
    [CallTimestamp] [nvarchar](255) NULL,
    [Call Type] [nvarchar](255) NULL,
    [EmployeeID] [nvarchar](255) NULL,
    [CallDuration] [nvarchar](255) NULL,
    [WaitTime] [nvarchar](255) NULL,
    [CallAbandoned] [nvarchar](255) NULL
) ON [PRIMARY]
GO
```

All values are set to nvarchar(255) for now, to be able to accept all data.

We can now define the target destination.



And finally, we define the columns mapping as follow:



Input Column	Destination Column
CallTimestamp	CallTimestamp
Call Type	Call Type
EmployeeID	EmployeeID
CallDuration	CallDuration
WaitTime	WaitTime
CallAbandoned	CallAbandoned

Here is the result:

	CallTimestamp	Call Type	EmployeeID	CallDuration	WaitTime	CallAbandoned
1	5/4/2018 16:33	3	U559430	486	2	0
2	6/21/2018 18:28	3	A166733	945	0	0
3	6/21/2018 15:13	1	B971624	379	11	0
4	11/21/2018 13:02	3	U641256	1044	0	0
5	2/25/2018 13:36	1	P286634	1357	0	0
6	10/28/2018 10:04	3	M855788	570	23	0
7	12/17/2018 16:39	3	M794992	26	26	0
8	7/28/2018 19:09	2	I281837	8	8	0
9	11/6/2018 18:57	1	J192194	800	0	0
10	6/18/2018 15:32	2	F542348	651	111	0

Operational Data Store

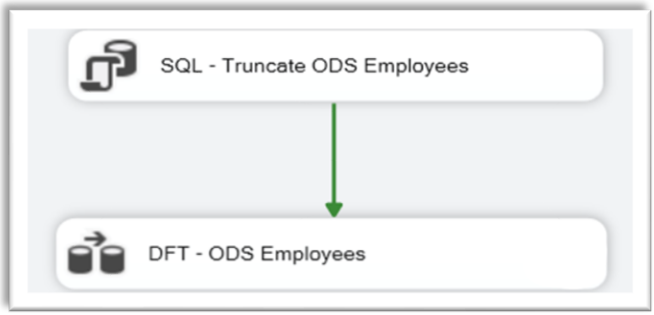
The second stage in the process involves loading usable data into the Operational Data Store (ODS) database. The aim is to transform data into a proper format that can be used efficiently.

Additionally, it requires cleaning and standardizing the data. Any data that fails to meet the established 'quality standards' will be rejected as a technical reject.

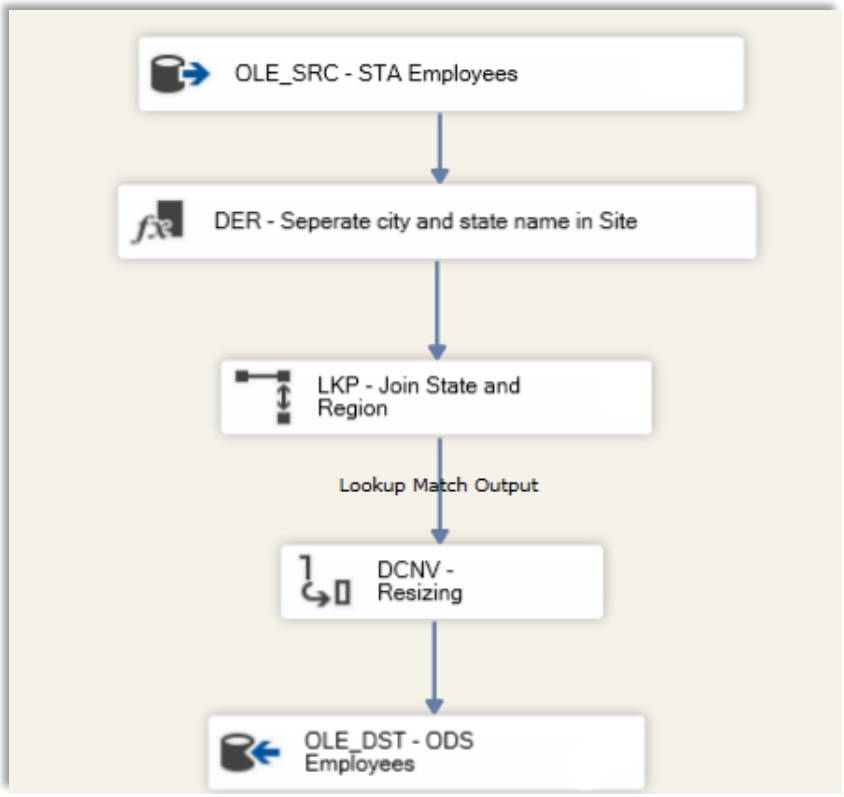
These quality standards are determined based on the accuracy of the data. The resulting data must maintain consistency in terms of data types and values. It is also essential to ensure that the data is suitable for queries, which might necessitate restructuring and enhancing the data.

Employees Table:

We start by processing the data in the "Employees" table. We firstly truncate the data from the previous runs:

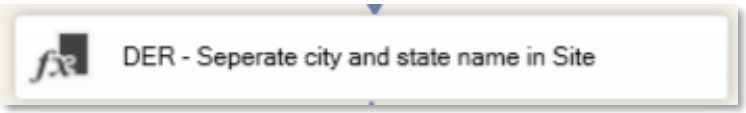


The data flow is defined as follow:

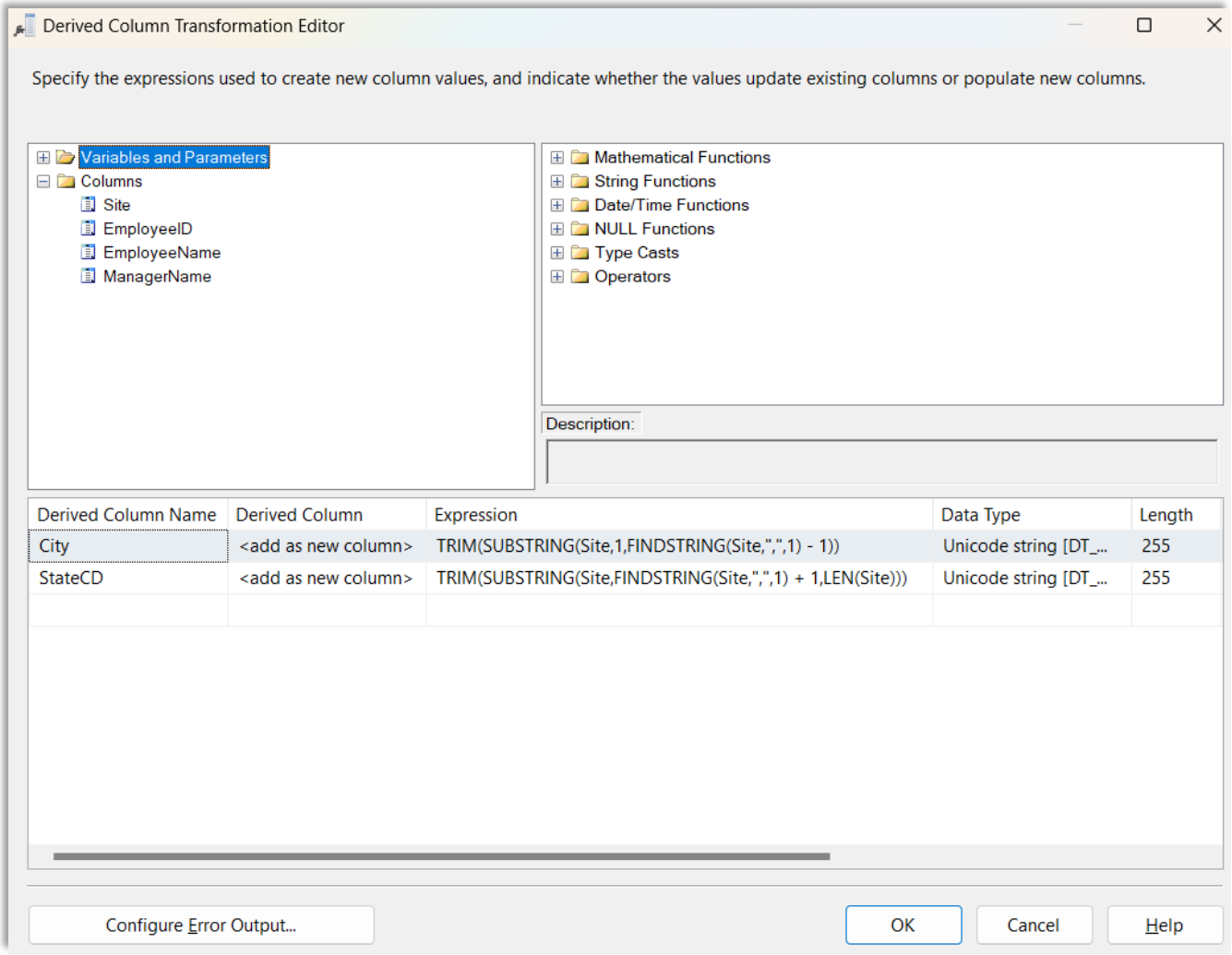


We will detail each block in the screenshot above.

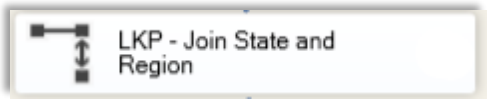
Bloc 1:



To simplify queries and the next steps, we will start by separating the city and state in the Site column. These two pieces of information are separated by a comma in the column of interest in the CSV file. So we use derived columns with the following expressions to perform the processing:

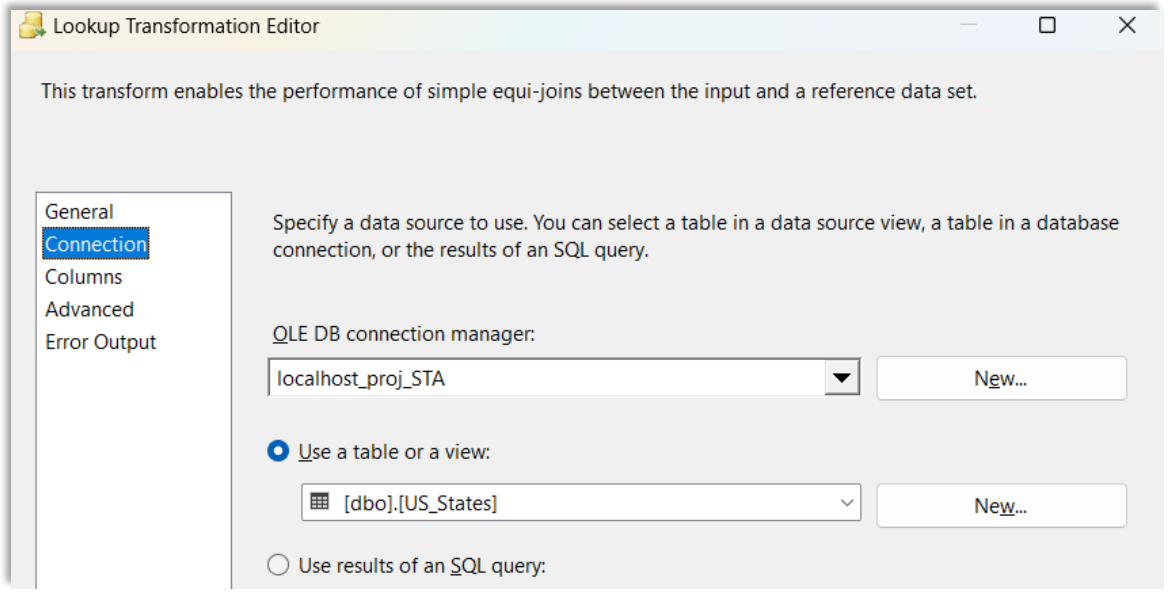


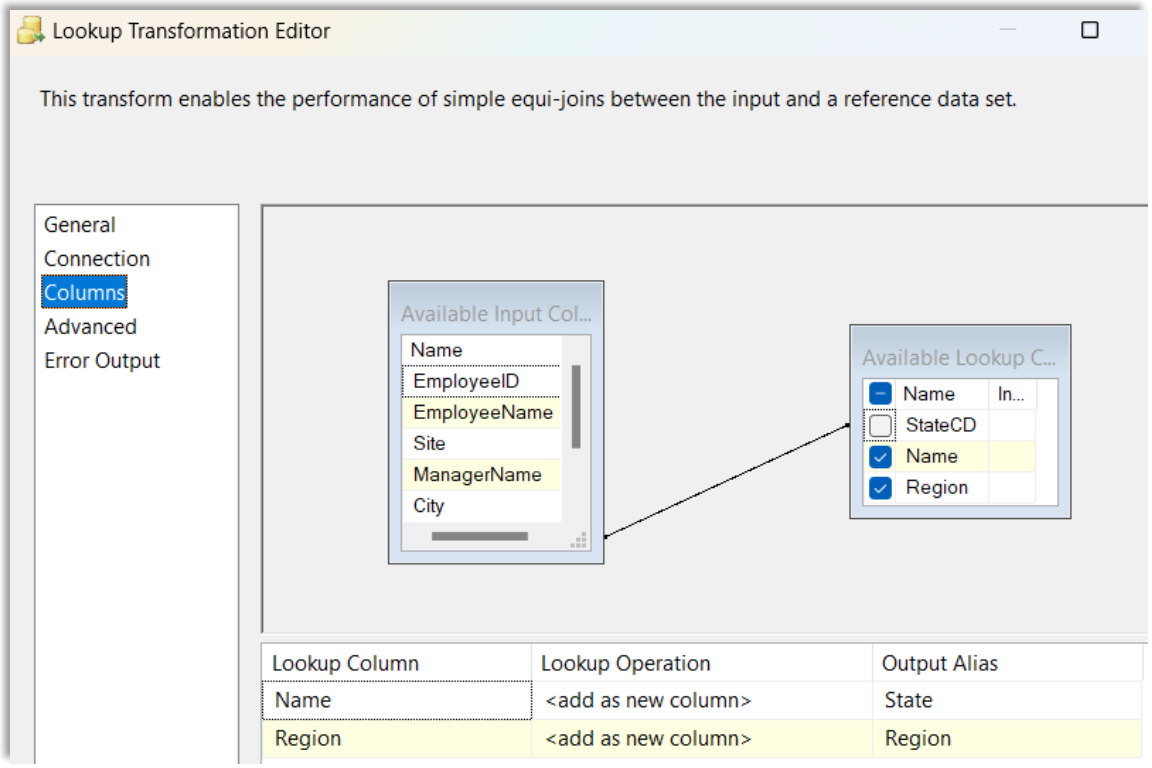
Bloc 2 :



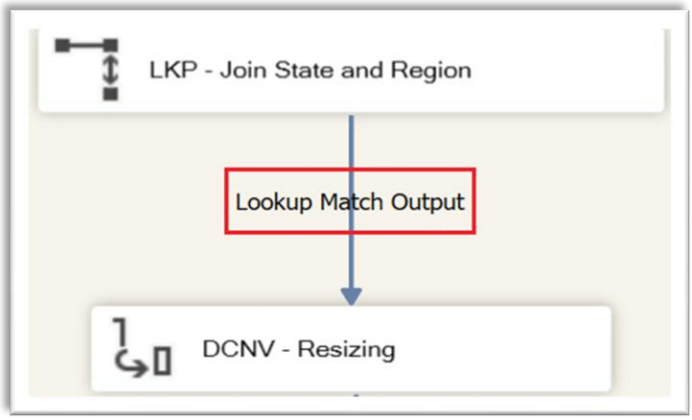
Now that we have a column for the state, we will do a LEFT JOIN operatonn with the US States table to add the full name and region of each state directly to the Employees table.

To achieve this integration, we use the Look Up Transformation. The Look Up Transformation allows us to create a join condition and to match rows from the "Employees" table with corresponding rows in the "US States" table based on the common "StateCD" column.





Before resizing, we make sure to configure the "Look Up Match Output" parameter to direct matching rows to the output arrow, so that data from both tables ("Employees" and "US States") that match on the specified column ("StateCD") will be routed to this output.



Bloc 3:

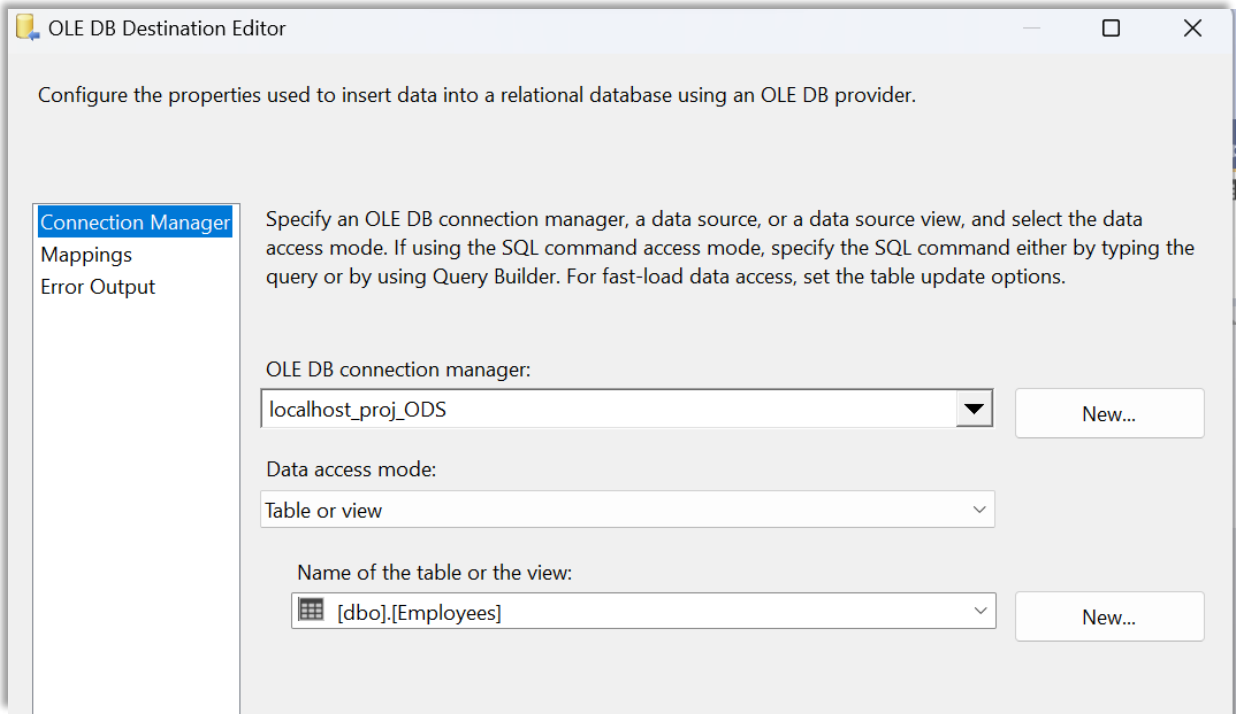
In the next step, before creating the table, we need to resize the data to adjust data types and character lengths to actual data requirements.

We resize each column as follows:

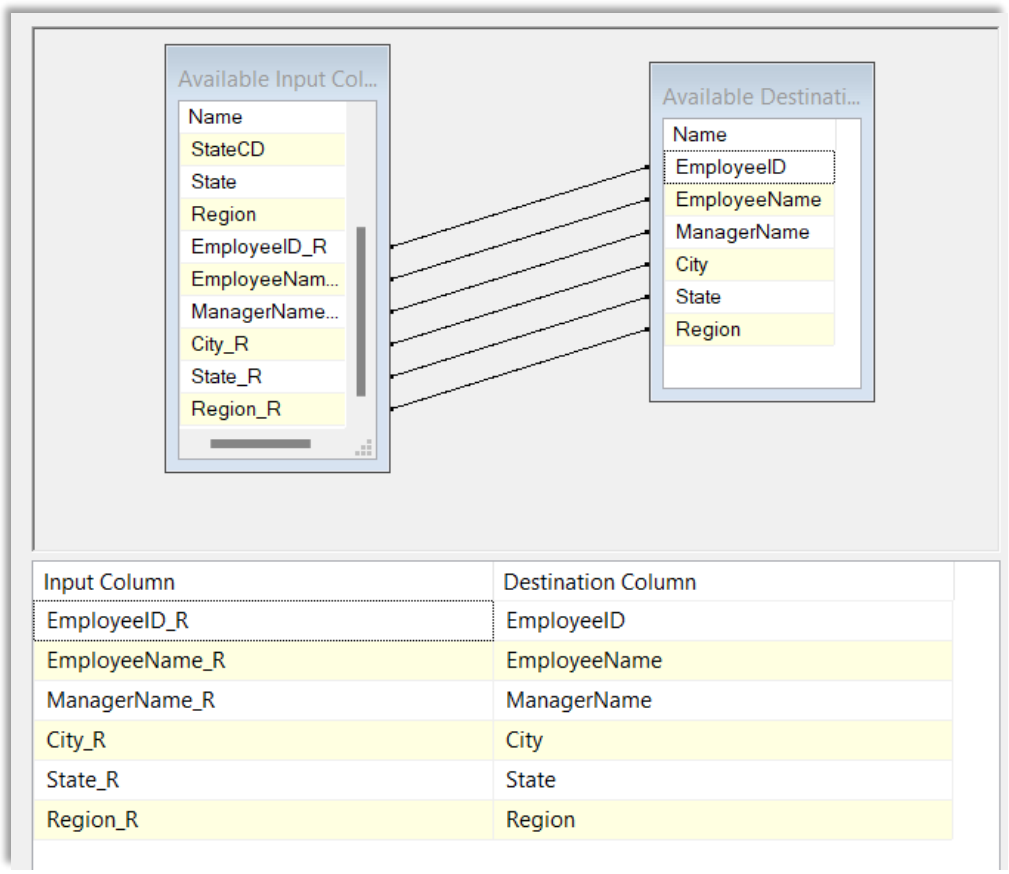
Input Column	Output Alias	Data Type	Length
EmployeeID	EmployeeID_R	chaîne Unicode [DT_WSTR]	10
EmployeeName	EmployeeName_R	chaîne Unicode [DT_WSTR]	50
ManagerName	ManagerName_R	chaîne Unicode [DT_WSTR]	50
City	City_R	chaîne Unicode [DT_WSTR]	50
State	State_R	chaîne Unicode [DT_WSTR]	50
Region	Region_R	chaîne Unicode [DT_WSTR]	20

The last task is to insert the data in the ODS database by creating the ODS “Employees” table as follow:

```
USE [proj_ODS]
GO

/****** Object:  Table [dbo].[Employees]    Script Date: 10/14/2023 3:57:39 PM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Employees](
    [EmployeeID] [nvarchar](10) NULL,
    [EmployeeName] [nvarchar](50) NULL,
    [ManagerName] [nvarchar](50) NULL,
    [City] [nvarchar](50) NULL,
    [State] [nvarchar](50) NULL,
    [Region] [nvarchar](20) NULL
) ON [PRIMARY]
GO
```



The final mapping is:



Input Column	Destination Column
EmployeeID_R	EmployeeID
EmployeeName_R	EmployeeName
ManagerName_R	ManagerName
City_R	City
State_R	State
Region_R	Region

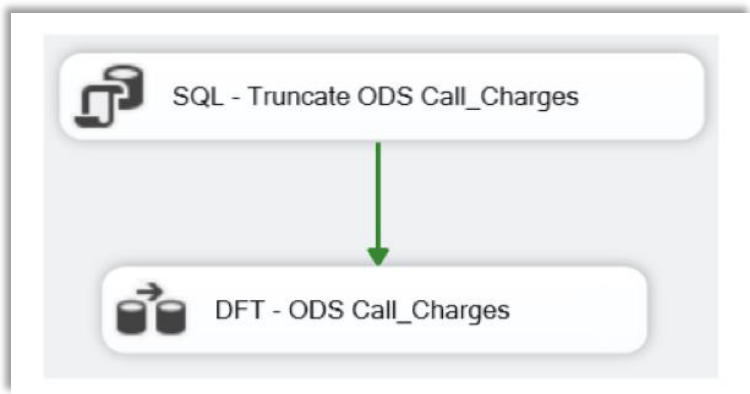
Here is the first ten lines of the results :

	EmployeeID	EmployeeName	ManagerName	City	State	Region
1	N772493	Onita Trojan	Deidre Robbs	Spokane	Washington	West
2	F533051	Stormy Seller	Elsie Taplin	Aurora	Colorado	West
3	S564705	Mable Ayoub	Shala Lion	Aurora	Colorado	West
4	I281837	Latrisha Buckalew	Rana Taub	Aurora	Colorado	West
5	Y193775	Adrianna Duque	Collin Trotman	Spokane	Washington	West
6	J632516	Keiko Daulton	Jamar Prahl	Spokane	Washington	West
7	G727038	Dolores Lundeen	Shala Lion	Aurora	Colorado	West
8	V126561	Wilbur Mohl	Casey Bainbridge	Jacksonville	Florida	South
9	E243130	Ileen Bornstein	Gonzalo Lesage	Jacksonville	Florida	South
10	C206355	Janeth Roesler	Miyoko Degraw	Spokane	Washington	West

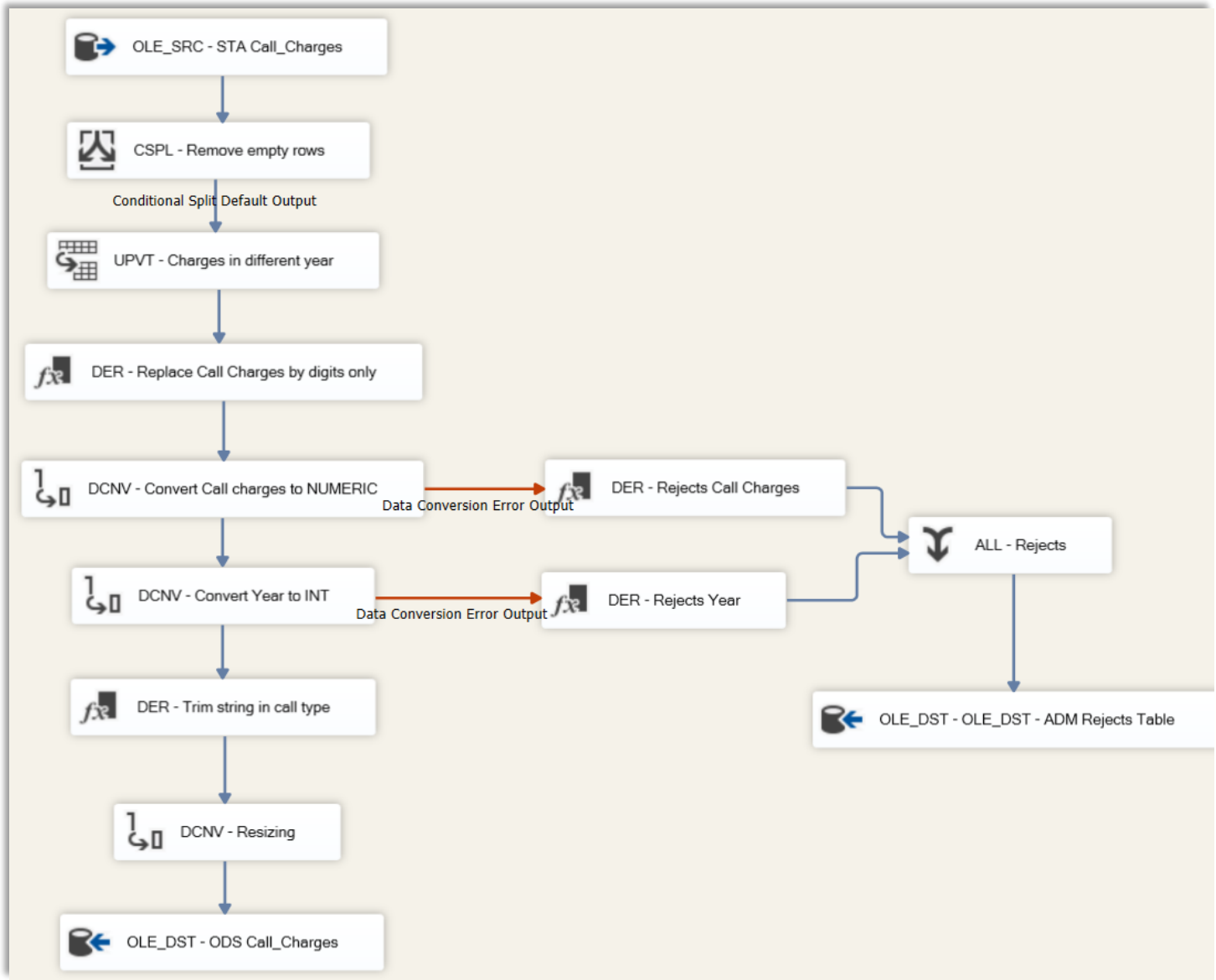
In the case of the Employees Table, rejection tables are unnecessary because no data transformations were applied. Since the original data remained unchanged and unprocessed, there were no discrepancies or errors to filter out, eliminating the need for rejection tables. The absence of transformations ensured data integrity, making rejection handling redundant in this scenario.

Call Charges Table

Now we need to process the data from the “Call Charges” table. For the same reason as the table before, you must first truncate the data from previous executions.

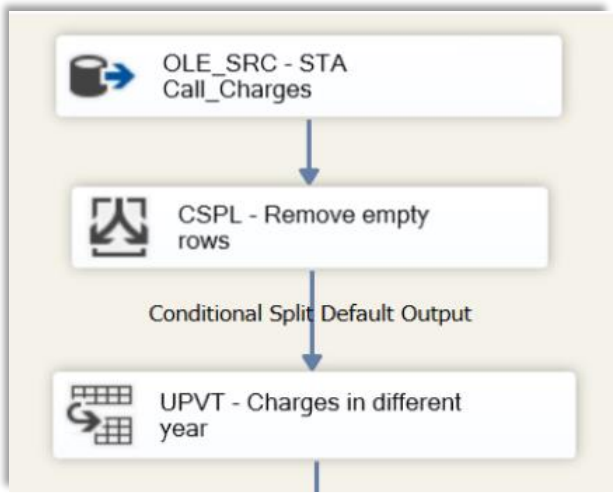


The data flow is defined as follow:

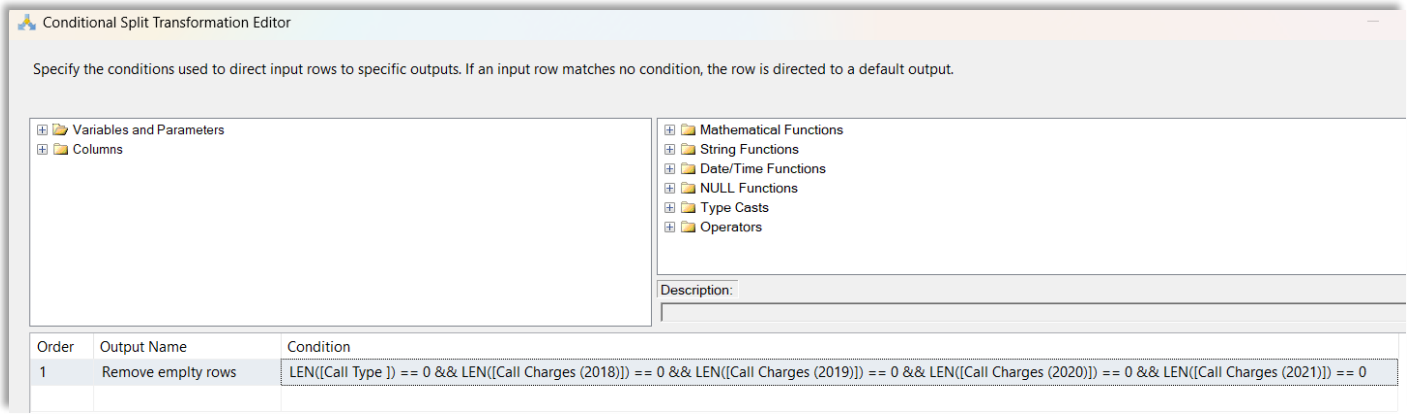


We will detail each major part of this dataflow.

Part 1 :



In the initial step, we import the data from the staging area and then we clean the Call Charges table. Although the table initially contains 4 rows of data, it also includes numerous empty rows. To address this, we employ the conditional split transformation with the specified expression:



The length of the columns 'Call Type', 'Call Charges (2018)', 'Call Charges (2019)', 'Call Charges (2020)', and 'Call Charges (2021)' should be equal to zero. This process enables us to eliminate the empty rows without any valid values from the table.

Then, we change the format of the table into one that is easier to query. In fact, the “Call charges” file is provided in a “easy-to-read” format for a human, but not very efficient to deal with for a database. So we will need to unpivot the data coming from that file. For this we use the “unpivot” transformation in SSIS.

The initial data format of the Call charges table has the year series data organized into columns and the categories are defined in the rows:

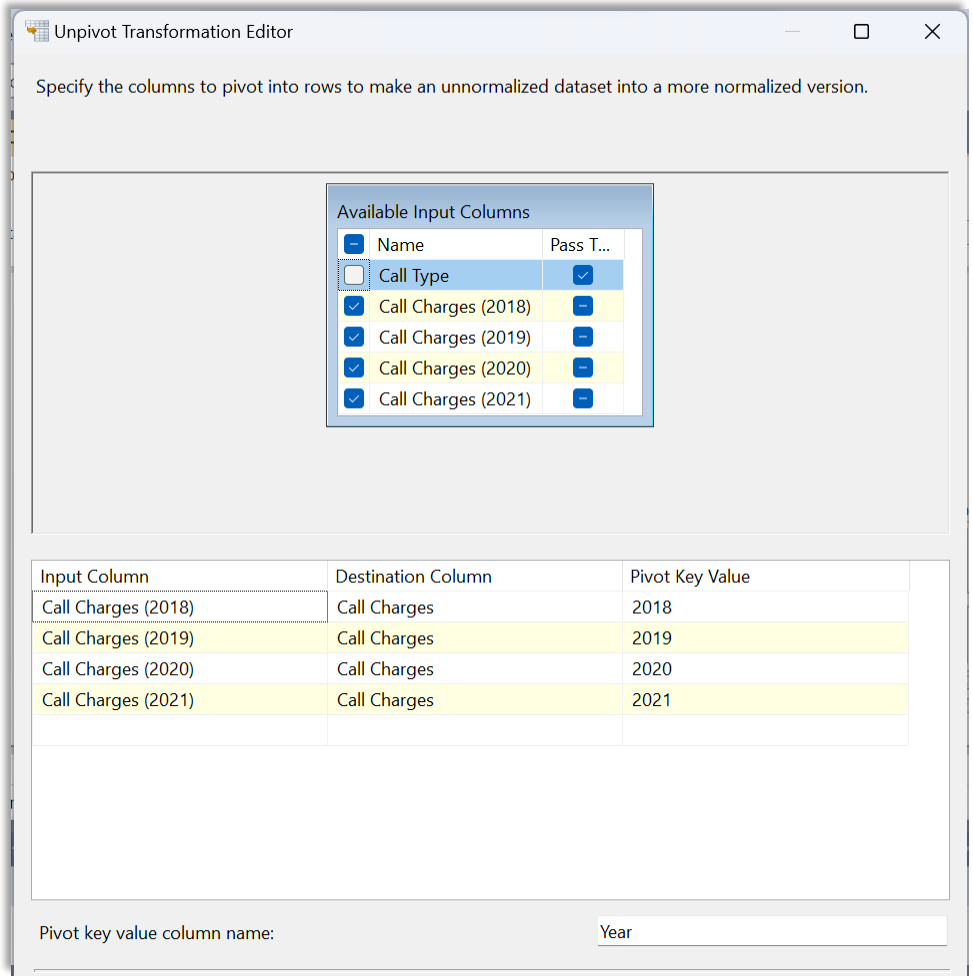
	A	B	C	D	E
1	Call Type	Call Charges (2018)	Call Charges (2019)	Call Charges (2020)	Call Charges (2021)
2	Sales	1.52 / min	1.56 / min	1.60 / min	1.71 / min
3	Billing	1.2 / min	1.32 / min	1.41 / min	1.45 / min
4	Tech Support	0.95 / min	0.98 / min	1.04 / min	1.12 / min

The desired output would have one row per year, with the associated call type and value:

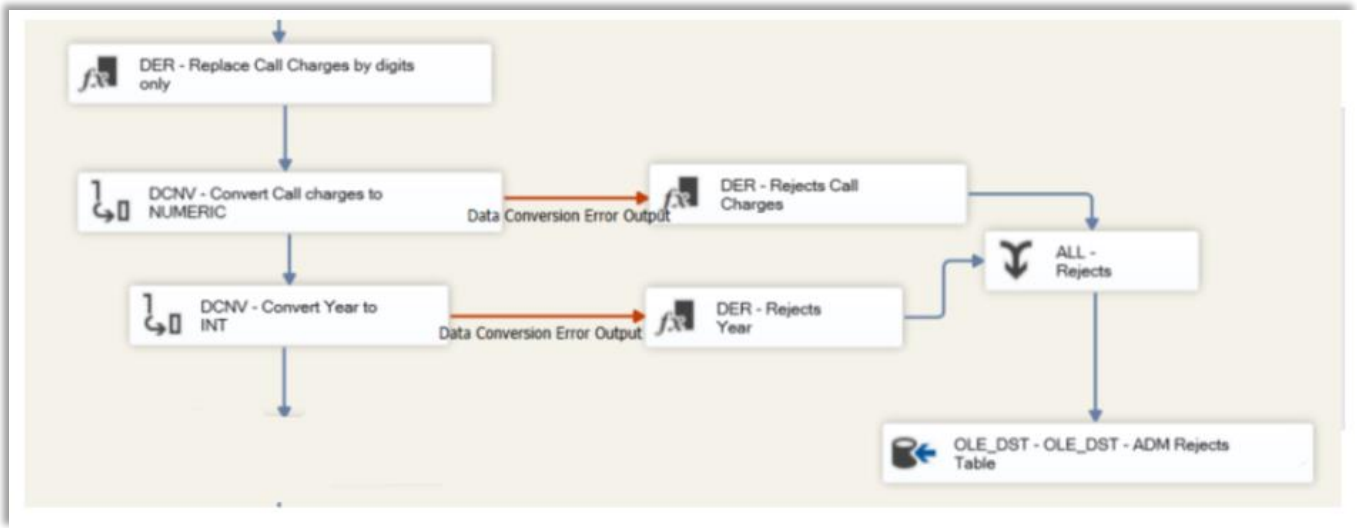
Results Messages

	Year	Type	Call Charges
1	2018	Sales	1.52
2	2019	Sales	1.56
3	2020	Sales	1.6
4	2021	Sales	1.71
5	2018	Billing	1.2
6	2019	Billing	1.32
7	2020	Billing	1.41
8	2021	Billing	1.45
9	2018	Tech Support	0.95
10	2019	Tech Support	0.98
11	2020	Tech Support	1.04
12	2021	Tech Support	1.12

We apply the “unpivot” transformation to the Year columns so we rename the year column “Year” and we move the data to a new column “Call Charges”:

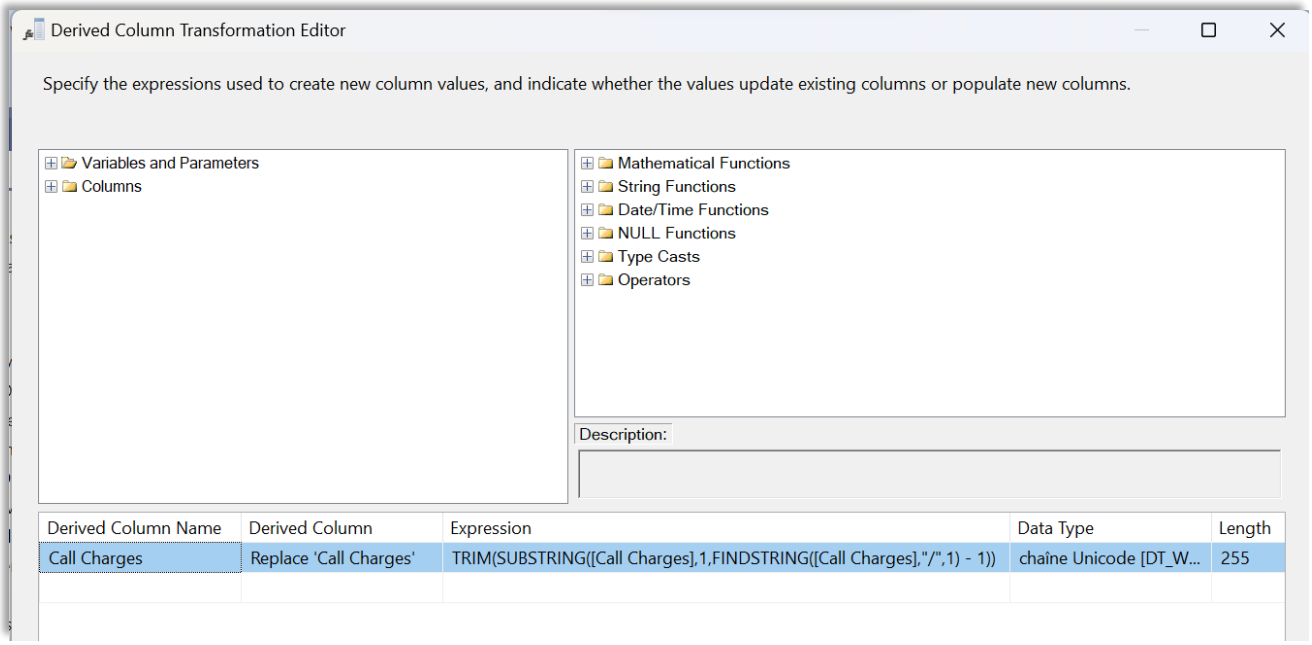


Part 2:



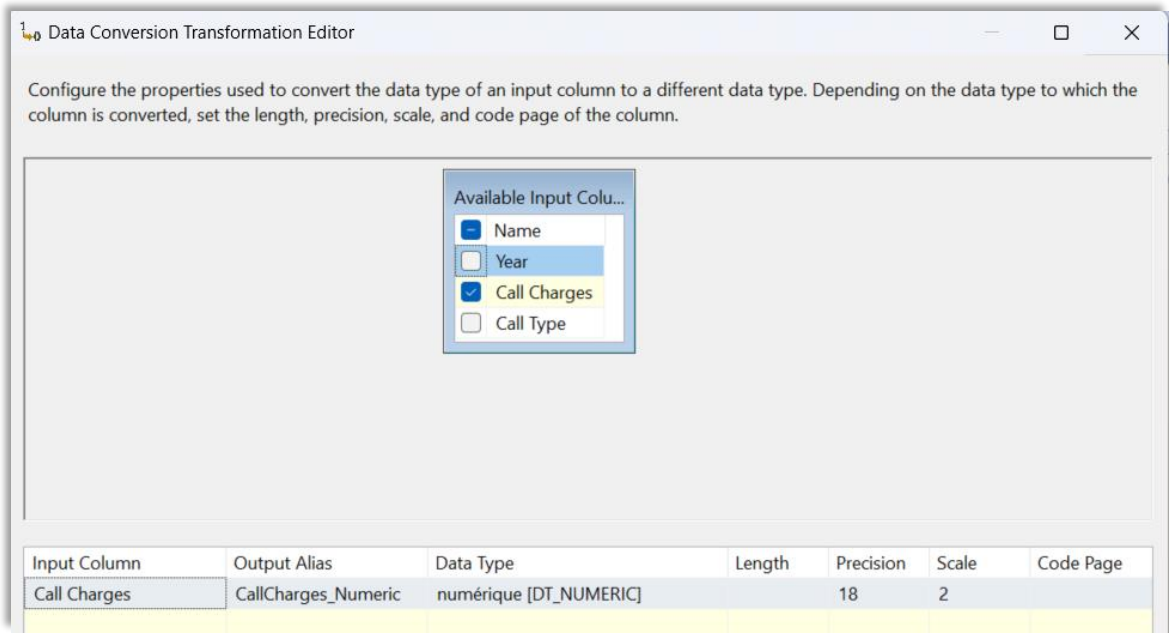
Then we had to take care of the Call Charges column, which corresponds to the price charged to customers for each minute spent on the phone. This column contains

excess characters, and which will pose a problem when we want to write queries. We must be able to have only the value of the call per minute without the '/' and 'min' characters.

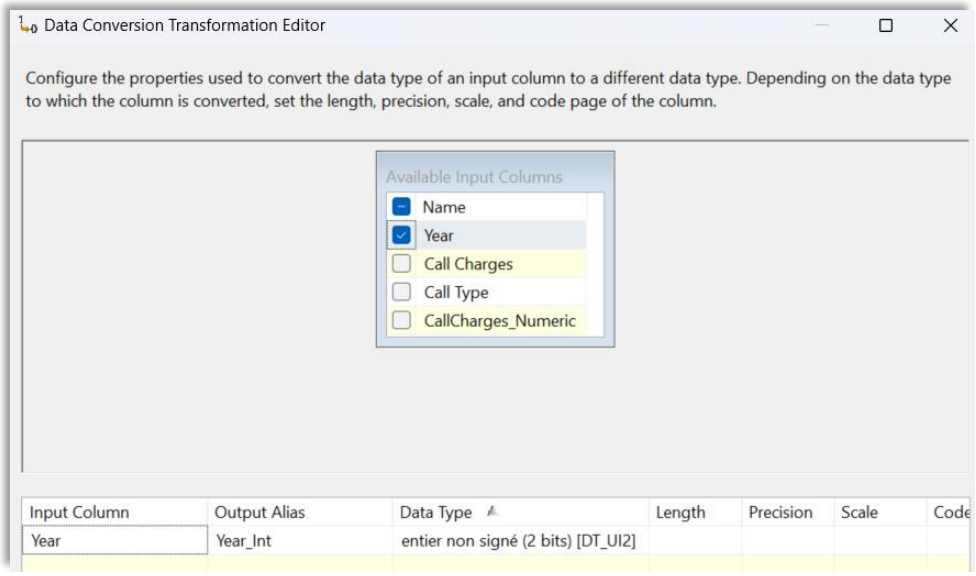


For this, a derived column was used to manipulate the [Call Charges] column. An expression was crafted to extract the numerical values before the '/' symbol. This was achieved using the combination of functions such as FINDSTRING, SUBSTRING, and TRIM:

Subsequently, a Data Conversion transformation was employed to convert the extracted string into a numeric data type, ensuring the data's consistency and usability:



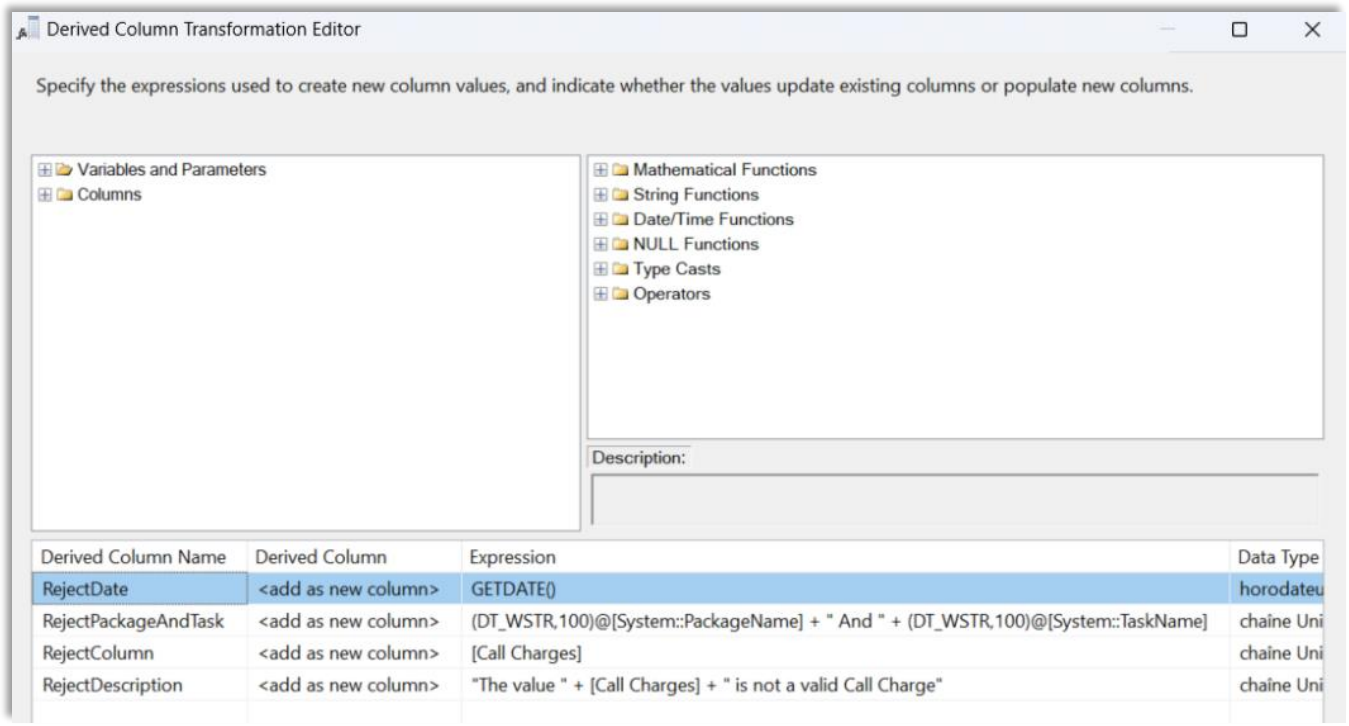
And in the same way, we must perform a data conversion transformation to convert the "Year" column into INT (integer):



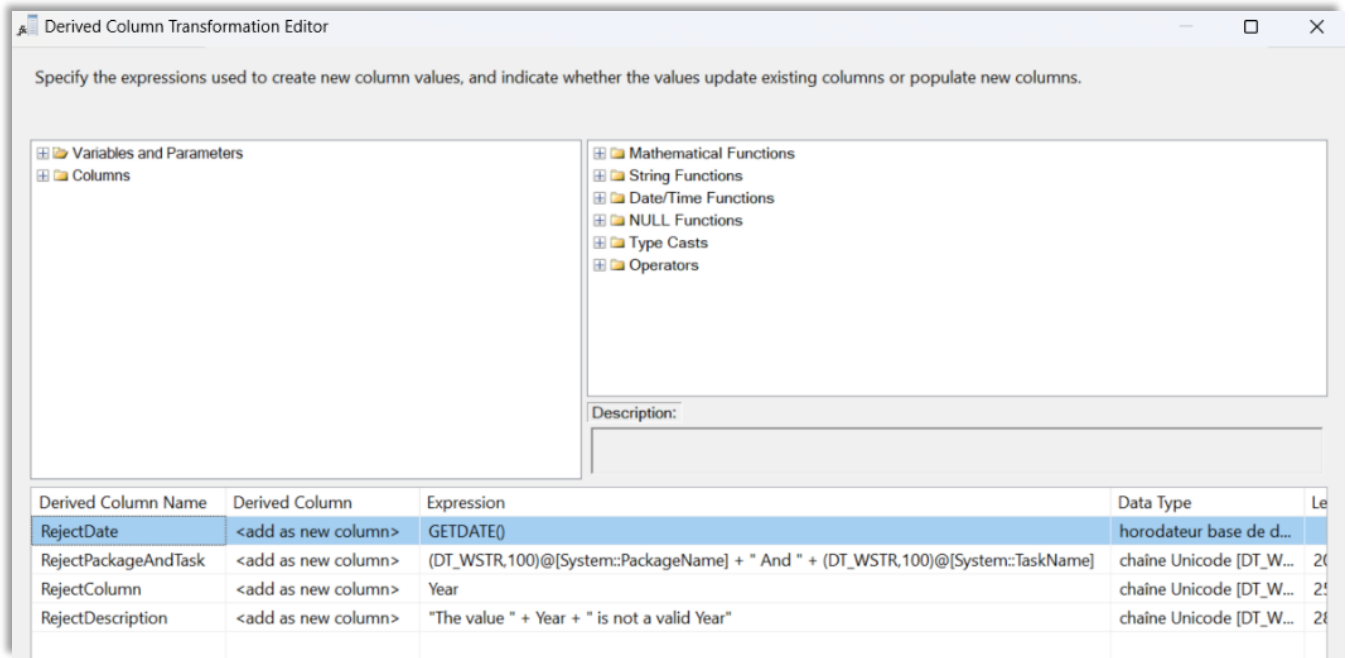
Finally, if these last two conversion steps fail, we track the errors and place them in the “Technical_Rejects” table. In fact, we have added multiple checks, and the errors are redirected to the “Technical_Rejects” Table created as follow:

```
USE [proj_ADM]
GO

/****** Object:  Table [dbo].[Technical_Rejects]    Script Date: 10/14/2023 4:32:43 PM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Technical_Rejects](
    [RejectDate] [datetime] NULL,
    [RejectPackageAndTask] [nvarchar](205) NULL,
    [RejectColumn] [nvarchar](255) NULL,
    [RejectDescription] [nvarchar](300) NULL
) ON [PRIMARY]
GO
```

Here is how we track the error when the values of the Call_Charges column are not numeric:



Here is how we track the error when the values in the Year column are not integers:



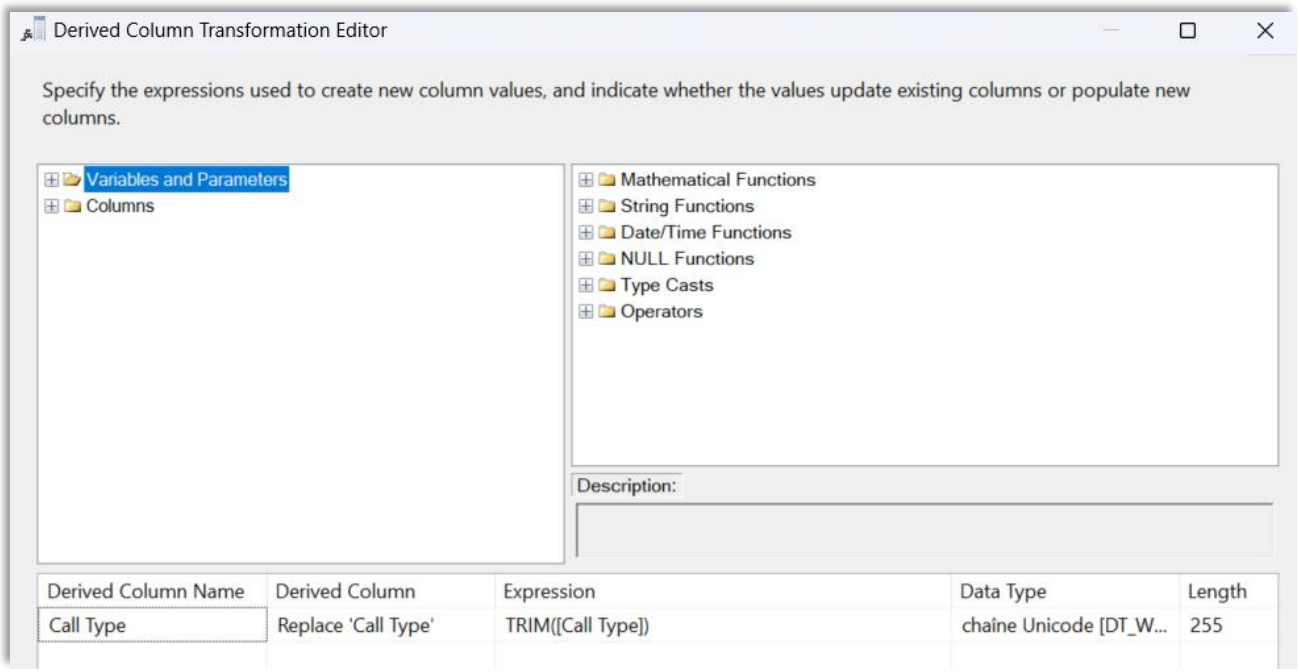
As seen above the data inserted in the technical rejects are:

- ✓ The date of the error.
- ✓ The package and the task causing the error.
- ✓ The column making the error.

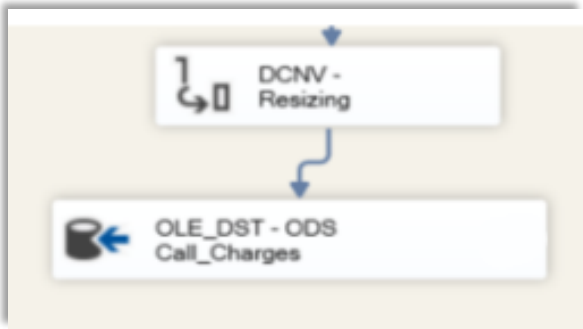
✓ An error message.

Part 3 :

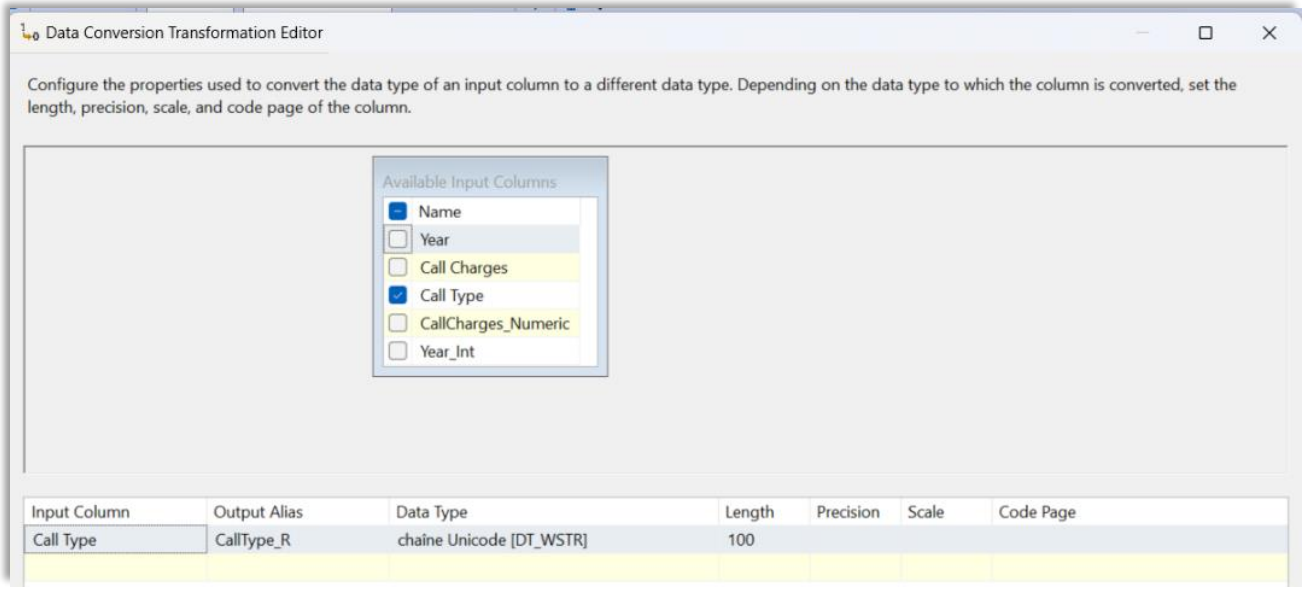
In this part, we create a derived column to remove the spaces at the end of the field for each CallType value in the CallCharges table because we noticed that a space was causing issues in the "Tech Support" value:



Part 4 :



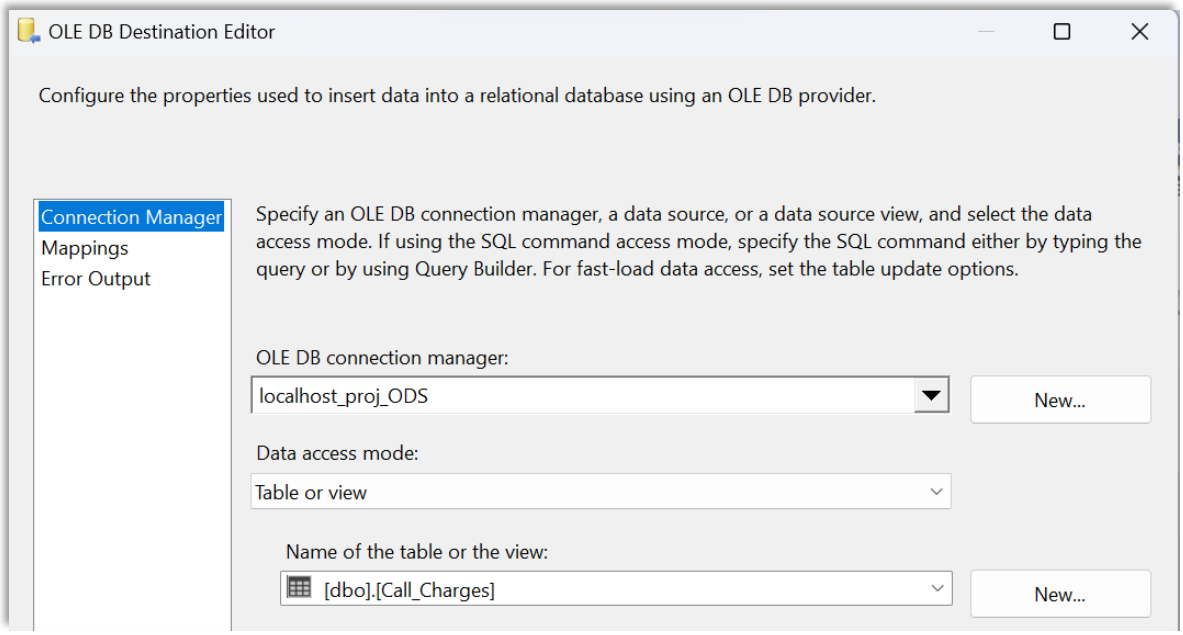
In the next step, we need to resize the data to adjust data types and character lengths to actual data requirements. We rename and resize the column of interest as follows:



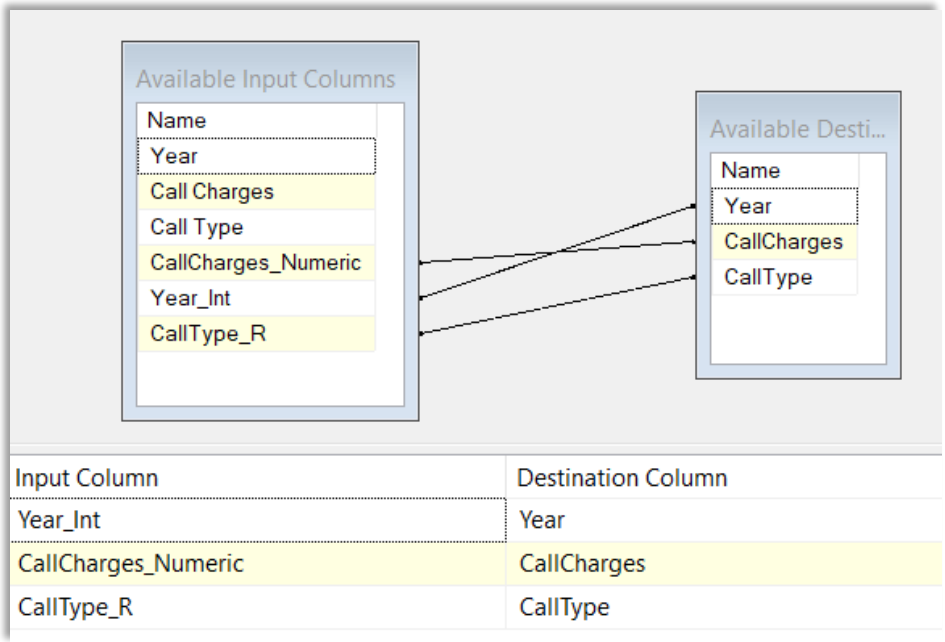
Now we can create the ODS “Call Charges” table as follow:

```
USE [proj_ODS]
GO

/****** Object:  Table [dbo].[Call_Charges]    Script Date: 10/14/2023 4:12:26 PM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Call_Charges](
    [CallCharges] [numeric](18, 2) NULL,
    [Year] [int] NULL,
    [CallType] [nvarchar](100) NULL
) ON [PRIMARY]
GO
```

The last task is to insert the data in the ODS database:



The final mapping is:



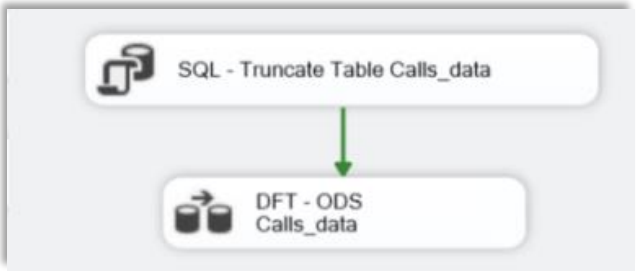
Input Column	Destination Column
Year_Int	Year
CallCharges_Numeric	CallCharges
CallType_R	CallType

And finally, here is the first ten lines of the results:

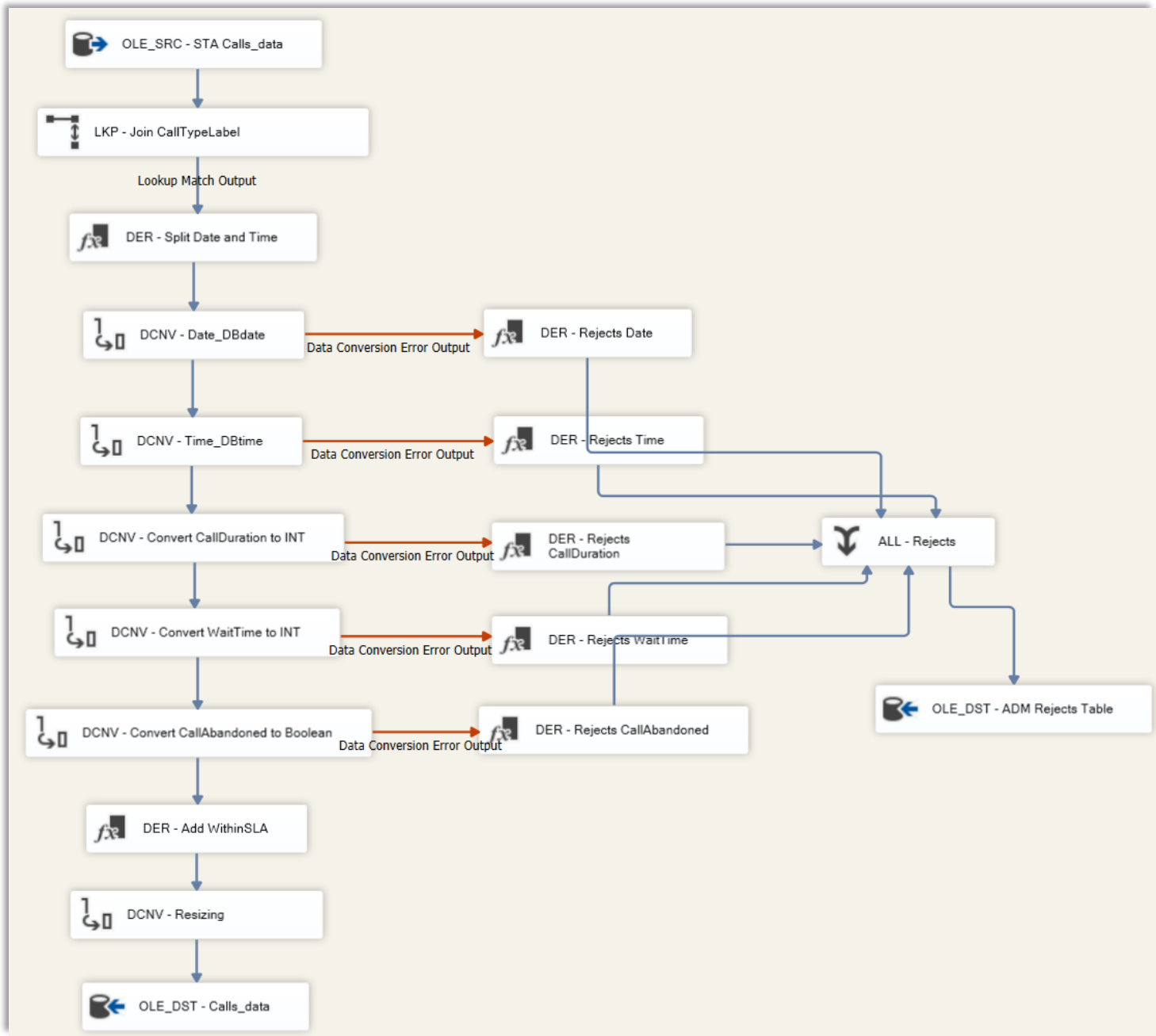
	CallCharges	Year	CallType
1	1.52	2018	Sales
2	1.56	2019	Sales
3	1.60	2020	Sales
4	1.71	2021	Sales
5	1.20	2018	Billing
6	1.32	2019	Billing
7	1.41	2020	Billing
8	1.45	2021	Billing
9	0.95	2018	Tech Support
10	0.98	2019	Tech Support
11	1.04	2020	Tech Support
12	1.12	2021	Tech Support

Calls Data Table :

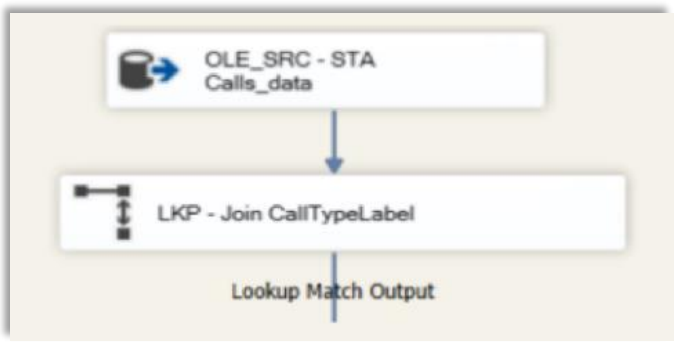
Finally, we process the main table: Calls Data. As usual, we truncate the data from previous executions and we can see a warning for the same reason as the previous table.



The data flow is defined as follow :

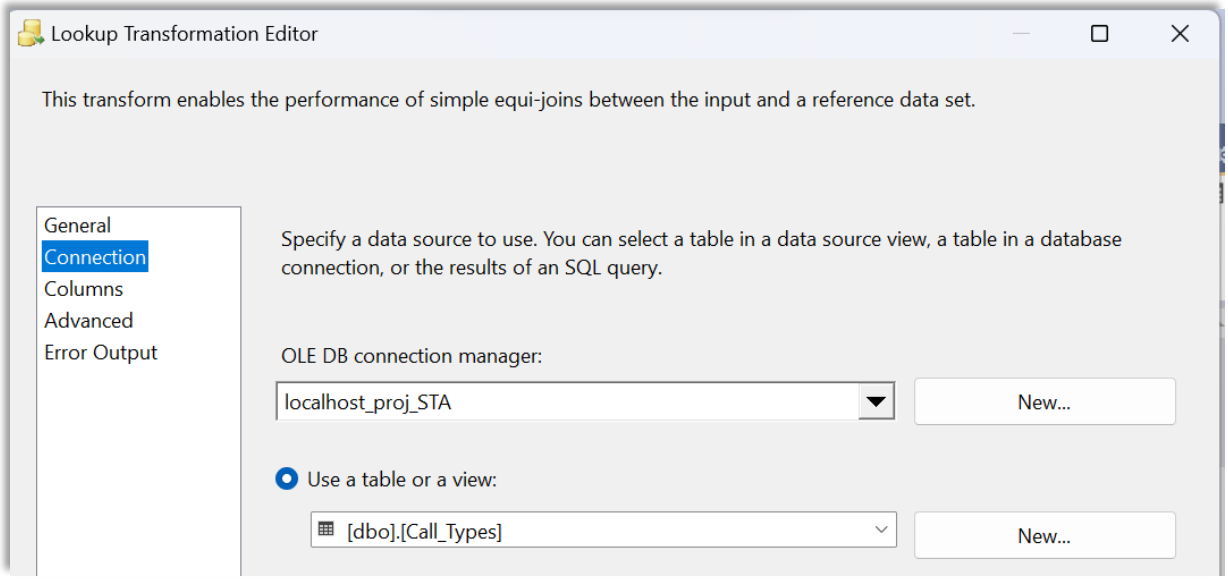

OLE_SRC - STA Calls_data
LKP - Join CallTypeLabel
Lookup Match Output
DER - Split Date and Time
DCNV - Date_DBdate
Data Conversion Error Output
DER - Rejects Date
DCNV - Time_DBtime
Data Conversion Error Output
DER - Rejects Time
DCNV - Convert CallDuration to INT
Data Conversion Error Output
DER - Rejects CallDuration
ALL - Rejects
DCNV - Convert WaitTime to INT
Data Conversion Error Output
DER - Rejects WaitTime
OLE_DST - ADM Rejects Table
DCNV - Convert CallAbandoned to Boolean
Data Conversion Error Output
DER - Rejects CallAbandoned
DER - Add WithinSLA
DCNV - Resizing
OLE_DST - Calls_data

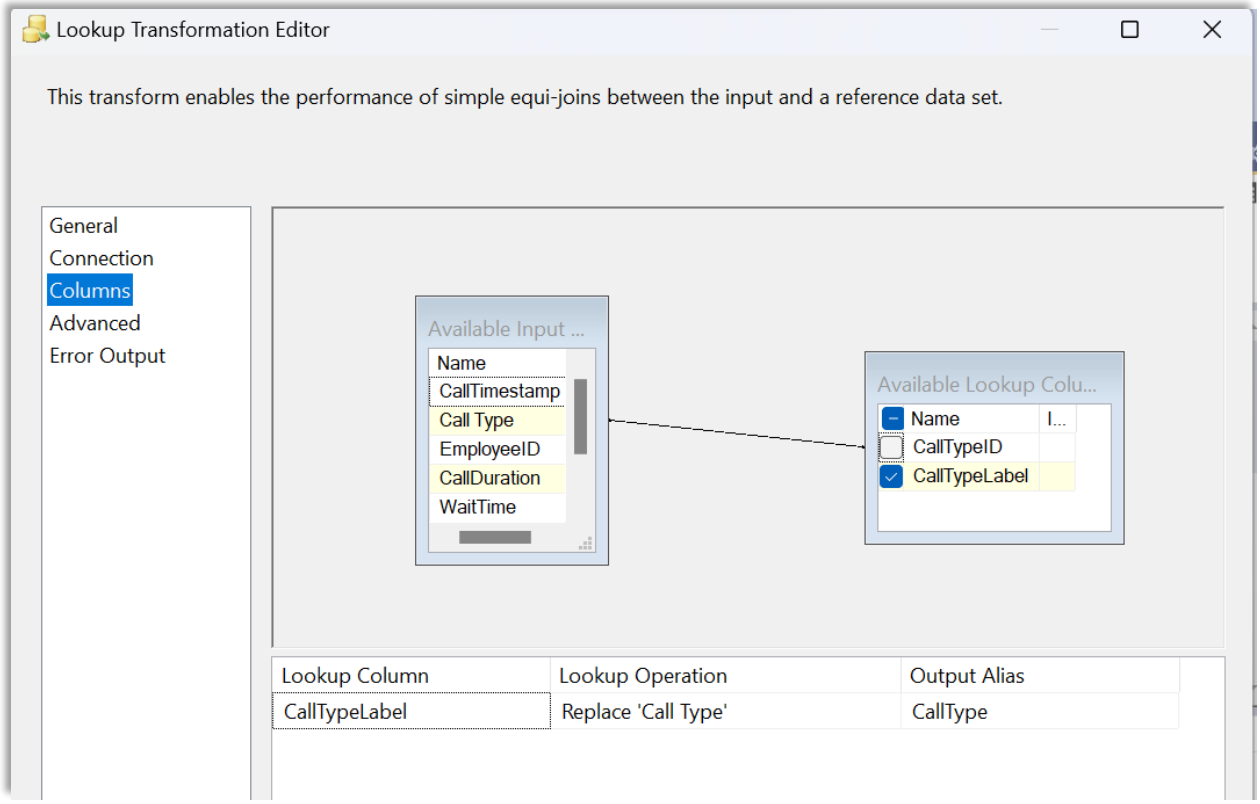
Part 1 :



After importing the data from the staging area, we start by making a join with the Call Type table in order to have the original names of the Call types directly in the main Call data table rather than the numbers, which will save us additional queries later when we want to analyze the data.

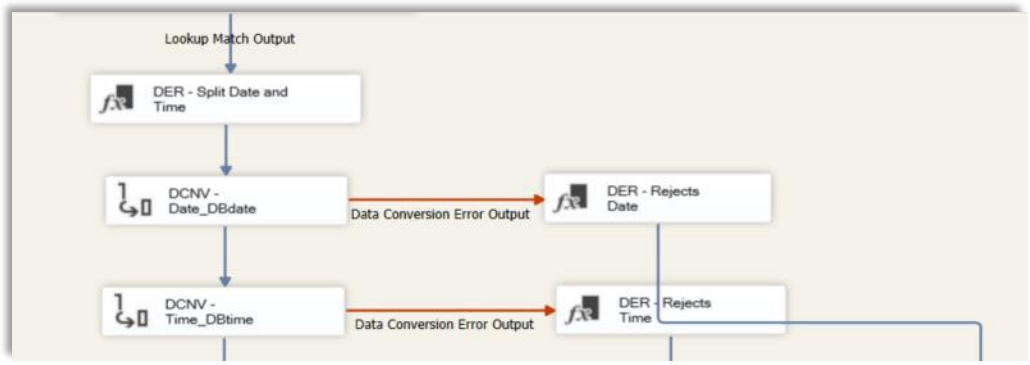
As for the Employees table, to achieve this integration, we use the Look Up transformation. The Look Up transformation allows us to create a join condition and match the rows in the “Call Type” table with the corresponding rows in the “Calls Data” table based on the common “CallTypeID” column to keep only the column CallTypeLabel for call type name:



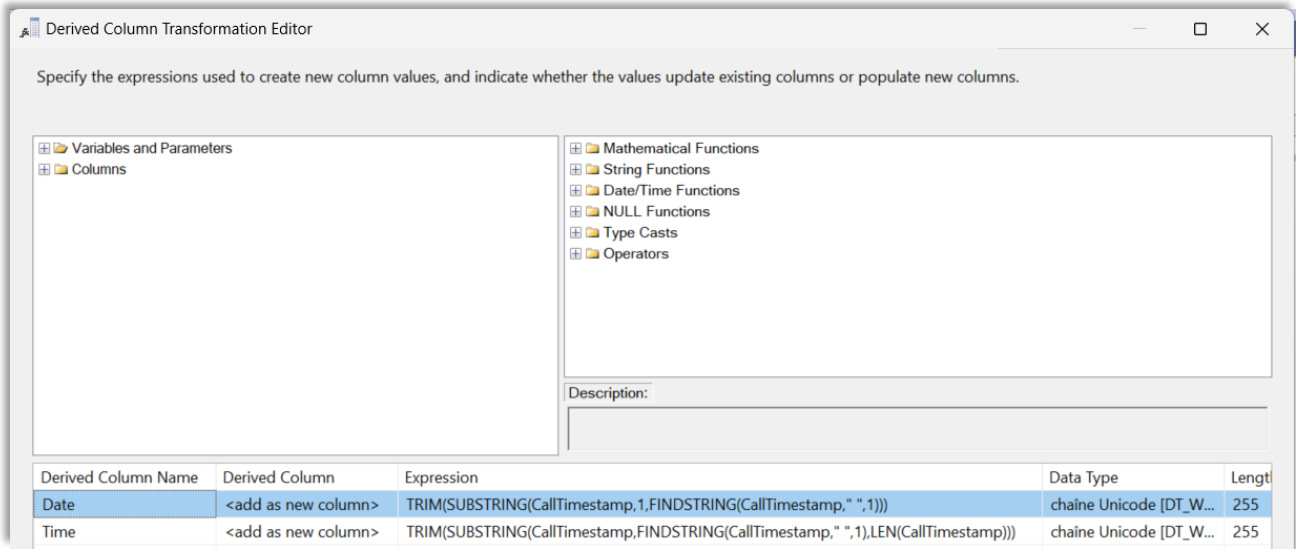


And once again, we have to make sure to configure the “Look Up Match Output” parameter to direct matching lines to the output arrow.

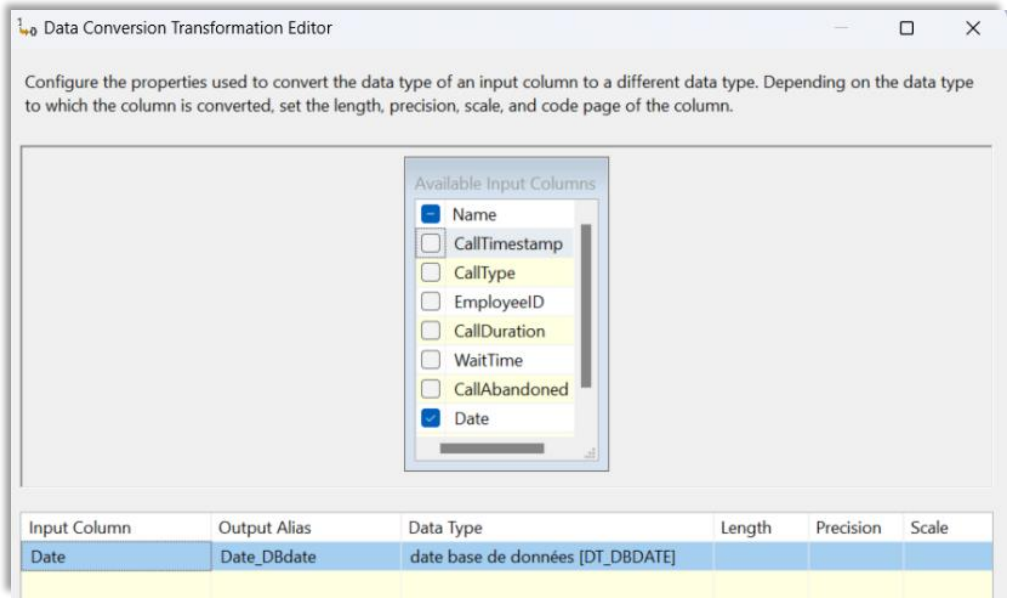
Part 2 :

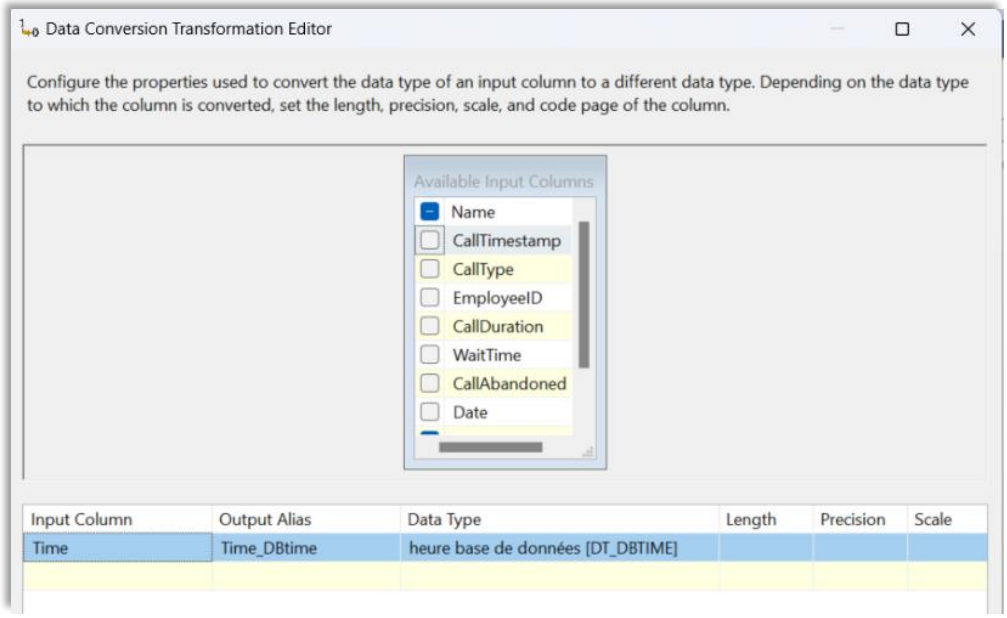


Then, we process the columns one by one, the "CallTimestamp" column includes data on the date and data on the time. Thus, we must separate these two data because it is a good practice to do. For this we use two derived columns using the following expressions:



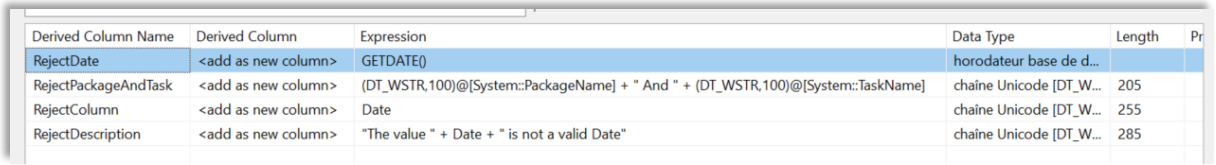
Then, we must convert the data in these two columns because they are in STRING by default. We use Data Conversion transformation blocks to convert the data type of the Date column to DBDATE and the data type of the Time column to DBTIME :





And in the same way as for the Call_Charges table, we must anticipate the case where these two conversion steps fail, we track the error and place it in the “Technical_Rejects” table with the following 4 columns:

- **Date Technical rejects :**

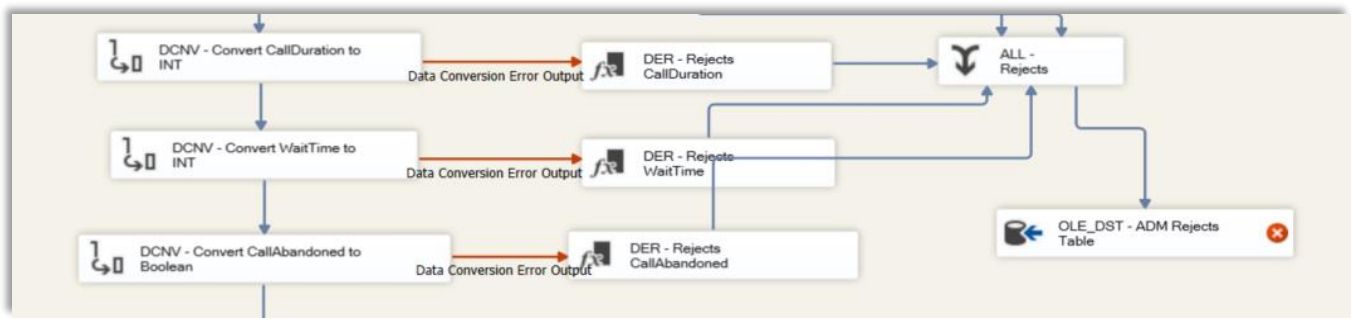


Derived Column Name	Derived Column	Expression	Data Type	Length	Pr
RejectDate	<add as new column>	GETDATE()	horodateur base de d...		
RejectPackageAndTask	<add as new column>	(DT_WSTR,100)@[System::PackageName] + " And " + (DT_WSTR,100)@[System::TaskName]	chaîne Unicode [DT_W...	205	
RejectColumn	<add as new column>	Date	chaîne Unicode [DT_W...	255	
RejectDescription	<add as new column>	"The value " + Date + " is not a valid Date"	chaîne Unicode [DT_W...	285	

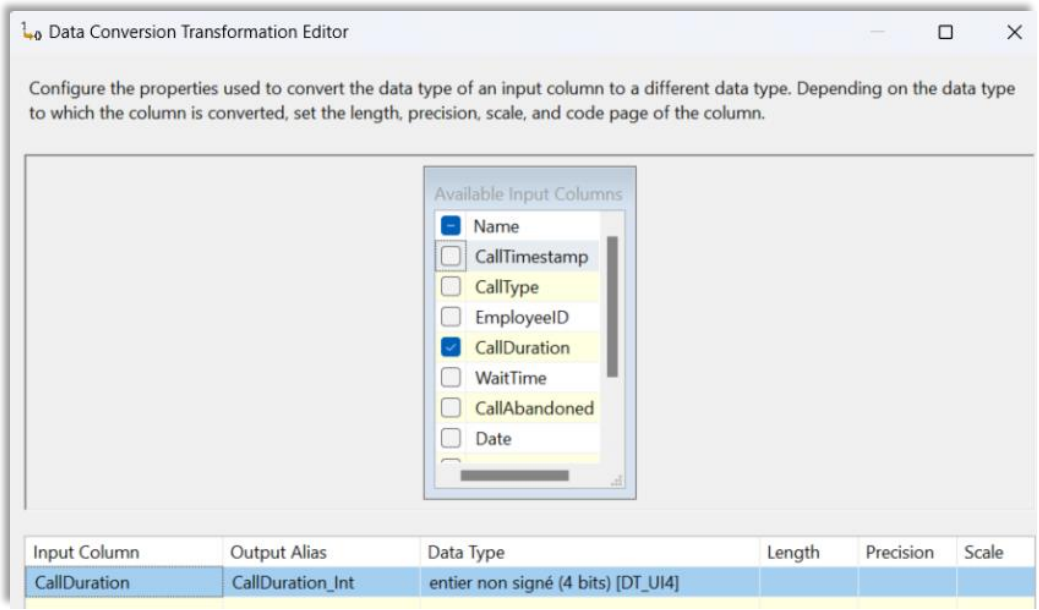
- **Time Technical rejects :**

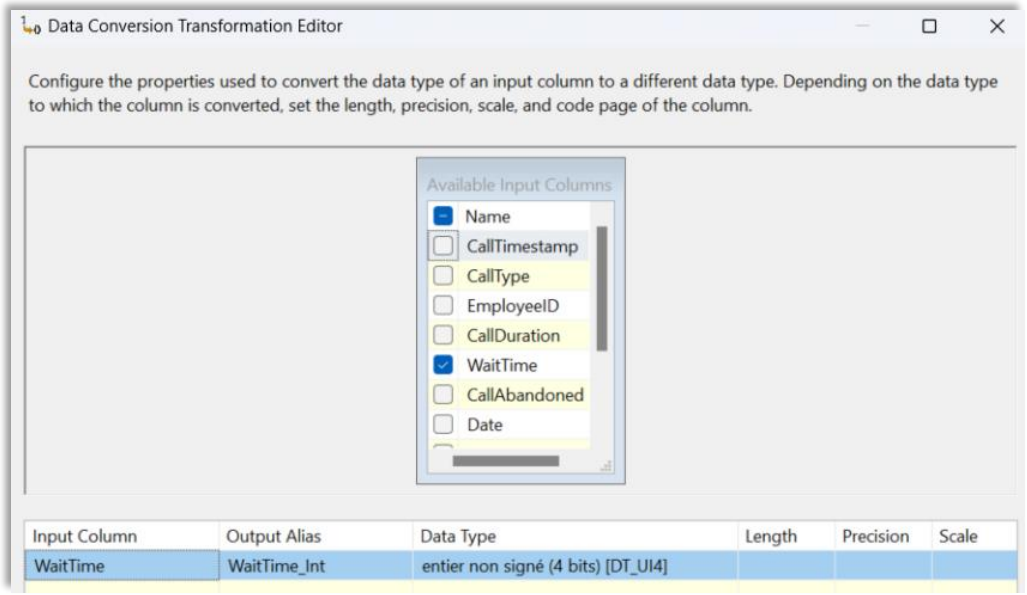
Derived Column Name	Derived Column	Expression	Data Type	Le
RejectDate	<add as new column>	GETDATE()	horodateur base de d...	
RejectPackageAndTask	<add as new column>	(DT_WSTR,100)@[System::PackageName] + " And " + (DT_WSTR,100)@[System::TaskName]	chaîne Unicode [DT_W...	2(
RejectColumn	<add as new column>	Time	chaîne Unicode [DT_W...	2
RejectDescription	<add as new column>	"The value " + Time + " is not a valid Time"	chaîne Unicode [DT_W...	2

Part 3 :

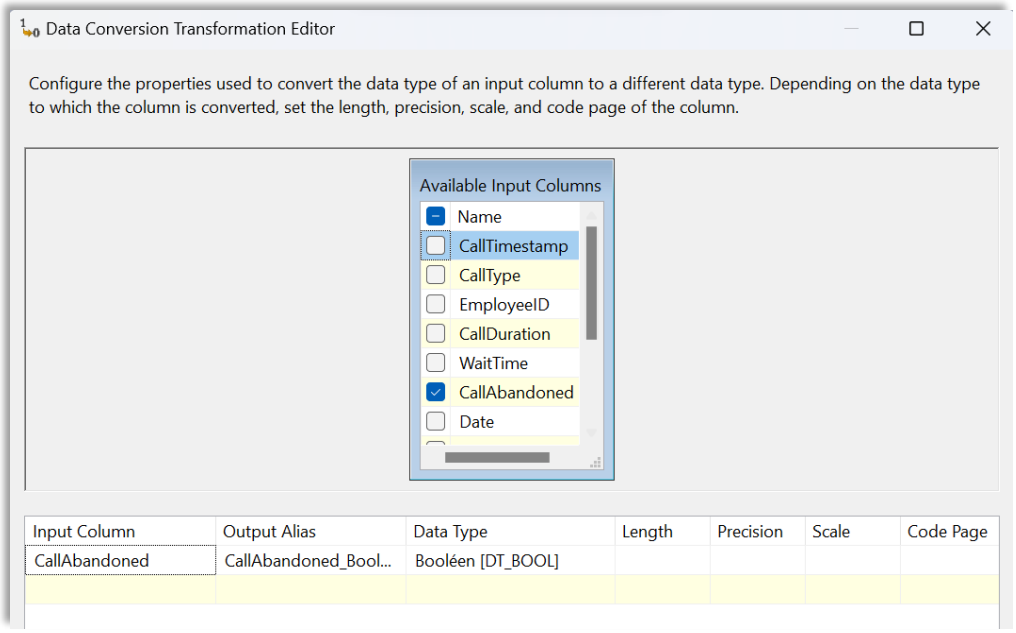


In this part, we focus on 3 columns of the table: CallDuration, WaitTime and CallAbandoned. To have a valid CallDuration and a valid WaitTime, which are in seconds, they must be integers. Then, we use a data conversion transformation to change the datatype:



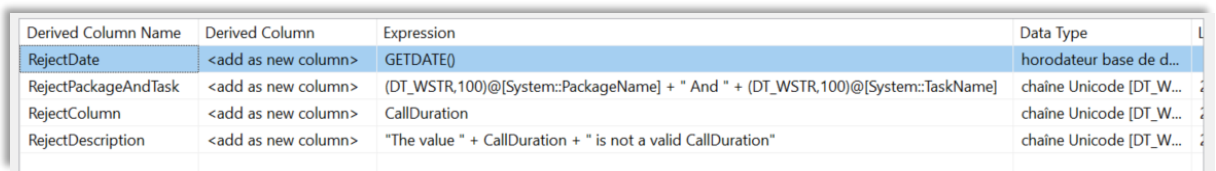


In the same way, we must change the datatype of CallAbandonedto boolean. In fact, this column allows us to know if the call was abandoned by the customer or not (1 = Yes, 0 = No):



In case of an error, we track these 3 conversions like before:

- **CallDuration Technical rejects :**



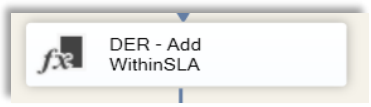
- **WaitTime Technical rejects :**

Derived Column Name	Derived Column	Expression	Data Type	L
RejectDate	<add as new column>	GETDATE()	horodateur base de d...	
RejectPackageAndTask	<add as new column>	(DT_WSTR,100)@[System::PackageName] + " And " + (DT_WSTR,100)@[System::TaskName]	chaîne Unicode [DT_W...	2
RejectColumn	<add as new column>	WaitTime	chaîne Unicode [DT_W...	2
RejectDescription	<add as new column>	"The value " + WaitTime + " is not a valid WaitTime"	chaîne Unicode [DT_W...	2

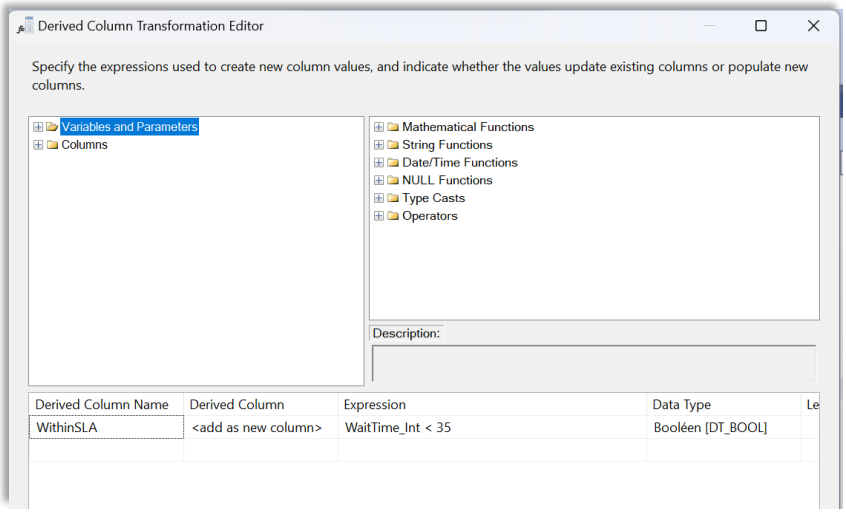
- **CallAbandoned Technical rejects :**

Derived Column Name	Derived Column	Expression	Data Type	L
RejectDate	<add as new column>	GETDATE()	horodateur base de d...	
RejectPackageAndTask	<add as new column>	(DT_WSTR,100)@[System::PackageName] + " And " + (DT_WSTR,100)@[System::TaskName]	chaîne Unicode [DT_W...	2
RejectColumn	<add as new column>	CallAbandoned	chaîne Unicode [DT_W...	2
RejectDescription	<add as new column>	"The value " + CallAbandoned + " is not a valid CallAbandoned value"	chaîne Unicode [DT_W...	3

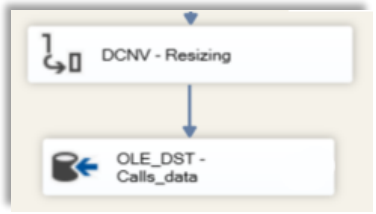
Part 4:



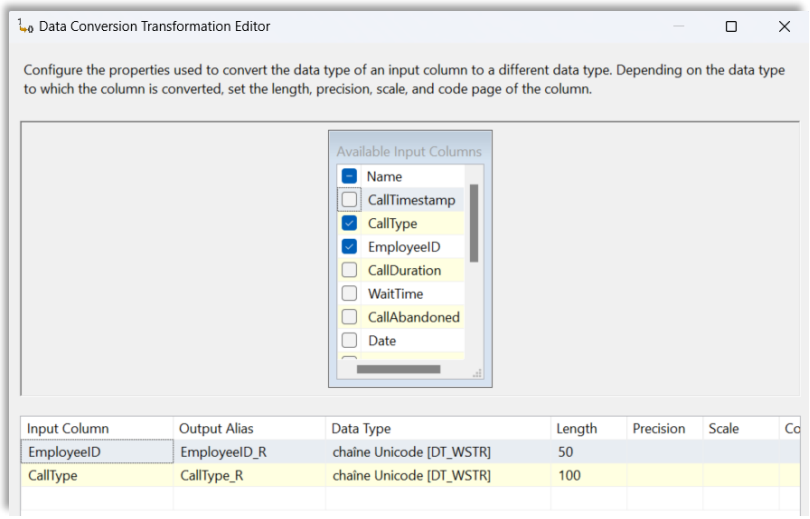
Among the project requirements, there is the need for the company to know if a call was answered before 35 seconds of waiting or not in order to respect the SLA. Thus, we must create a derived column which will consider the WaitTime column which will have a Boolean datatype, if WaitTime is less than 35 seconds then the value will be true, otherwise the value will be false. For this we used a derived column block:



Part 5:



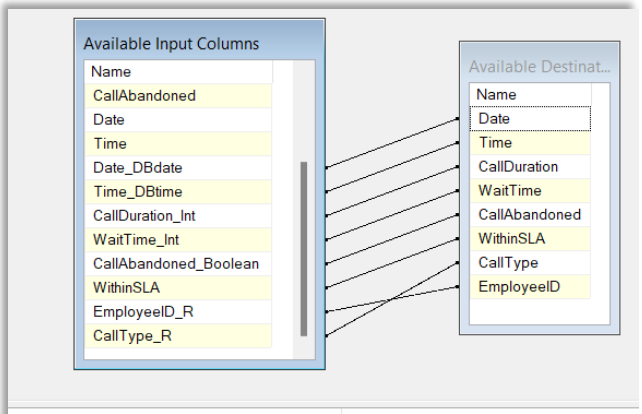
Finally, we rename and resize the “EmployeeID” column and the “CallType” column :



We can then create the ODS “Calls_data” table as follow:

```
USE [proj_ODS]
GO

/****** Object:  Table [dbo].[Calls_data]    Script Date: 10/14/2023 4:08:56 PM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Calls_data](
    [Date] [date] NULL,
    [Time] [time](7) NULL,
    [CallDuration] [bigint] NULL,
    [WaitTime] [bigint] NULL,
    [CallAbandoned] [bit] NULL,
    [WithinSLA] [bit] NULL,
    [EmployeeID] [nvarchar](50) NULL,
    [CallType] [nvarchar](100) NULL
) ON [PRIMARY]
GO
```

And as for the previous cases, the last task is to insert the data in the ODS database and here is he final mapping :



Input Column	Destination Column
Date_DBdate	Date
Time_DBtime	Time
CallDuration_Int	CallDuration
WaitTime_Int	WaitTime
CallAbandoned_Boolean	CallAbandoned
WithinSLA	WithinSLA
CallType_R	CallType
EmployeeID_R	EmployeeID

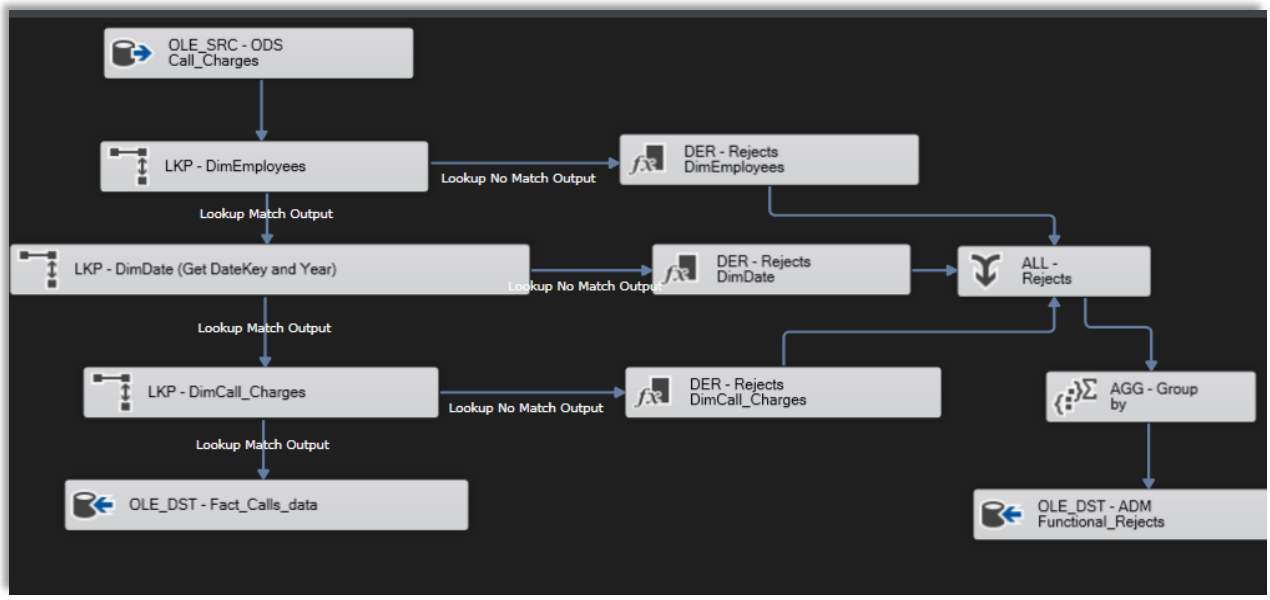
To finish with the last and main table of the ODS phase, here is the first ten lines of the results:

	Date	Time	CallDuration	WaitTime	CallAbandoned	WithinSLA	EmployeeID	CallType
1	2018-05-04	16:33:00.0000000	486	2	0	1	U559430	Tech Support
2	2018-06-21	18:28:00.0000000	945	0	0	1	A166733	Tech Support
3	2018-06-21	15:13:00.0000000	379	11	0	1	B971624	Sales
4	2018-11-21	13:02:00.0000000	1044	0	0	1	U641256	Tech Support
5	2018-02-25	13:36:00.0000000	1357	0	0	1	P286634	Sales
6	2018-10-28	10:04:00.0000000	570	23	0	1	M855788	Tech Support
7	2018-12-17	16:39:00.0000000	26	26	0	1	M794992	Tech Support
8	2018-07-28	19:09:00.0000000	8	8	0	1	I281837	Billing
9	2018-11-06	18:57:00.0000000	800	0	0	1	J192194	Sales
10	2018-06-18	15:32:00.0000000	651	111	0	0	F542348	Billing

Data Warehousing

In this phase we finally present the fact table. The fact table will be “Fact_Calls_data.” Before making dimentional tables (linking this with various other tables). Let’s look at the columns of this table in ODS phase:

1. [Date] - This column will is a date type and can be linked to a Dimensional table called DimDate.
2. [Time] - This column as well can be linked to Dimensional Table called DimDate.
3. [CallDuration]- It is an integer value.
4. [WaitTime] - It is an integer value.
5. [CallAbandoned]- It is a boolean value.
6. [WithinSLA] - It is a booean value.
7. [EmployeeID] - It can be linked to Dimensional table DimEmployees.
8. [CallType] - It can be linked to Dimensional Table DimCall_Charges.



As we can see it does Lookup in three Tables i.e., DimEmployees, DimCall_Charges and DimDate (This is the date table generated using SQL script) If we do not find any matches we will send the records to functional rejects.

We will need the following SQL script for functional rejects:

```
USE [proj_ADM]
GO

/****** Object:  Table [dbo].[Functional_Rejects]    Script Date: 10/14/2023 4:33:50 PM ******
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Functional_Rejects](
    [Nb_Rejects] [numeric](20, 0) NULL,
    [RejectDate] [datetime] NULL,
    [RejectPackageAndTask] [nvarchar](205) NULL,
    [RejectColumn] [nvarchar](10) NULL,
    [RejectDescription] [nvarchar](47) NULL
) ON [PRIMARY]
GO
```

As a result, we get the following fact table.

	CallDuration	WaitTime	CallAbandoned	WithinSLA	CallType	EmployeesKey	DateKey	CallChargesKey
1	1116	8	0	1	Tech Support	60	20180513	51
2	411	0	0	1	Tech Support	24	20180311	51
3	1145	15	0	1	Tech Support	13	20181127	51
4	435	0	0	1	Tech Support	52	20180705	51
5	8	0	1	1	Tech Support	16	20180216	51
6	1153	27	0	1	Tech Support	45	20181005	51
7	1378	27	0	1	Billing	52	20180209	47
8	95	24	0	1	Tech Support	35	20180705	51
9	1416	24	0	1	Tech Support	30	20181129	51
10	1439	0	0	1	Tech Support	7	20180915	51

Dimension Table DimDate

We will execute our own SQL script to generate the Dimensional date table called DimDate.

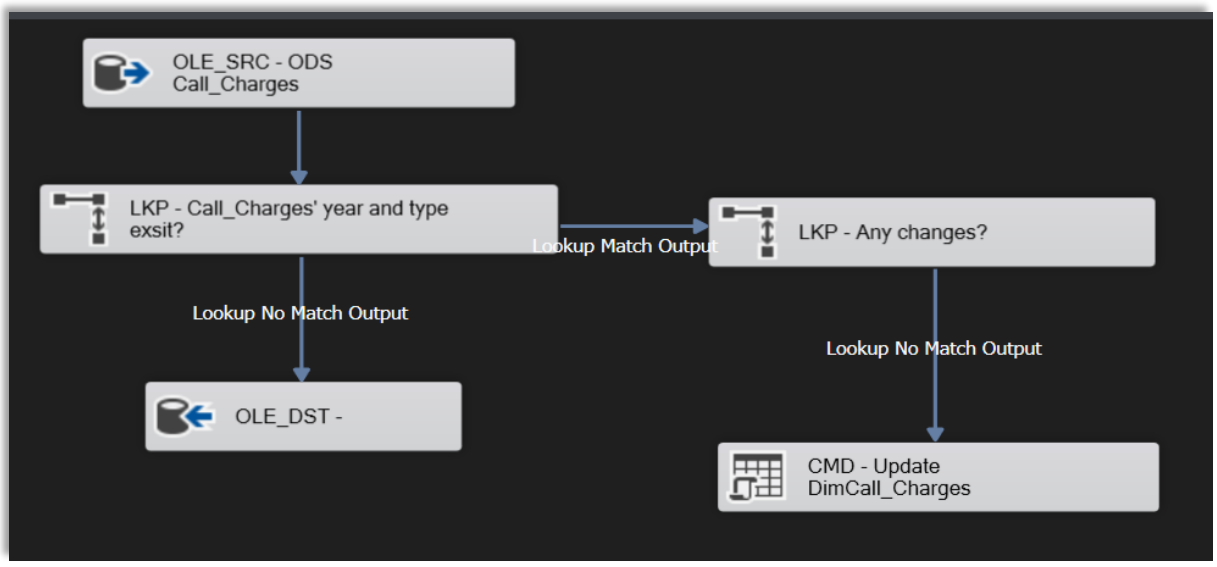
As a result, we will get the following dimension table DimDate in DWH database.

	DateKey	Date	Day	DaySuffix	Weekday	WeekDayName	WeekDayName_Short	WeekDayName_FirstLetter	DOWInMonth
1	20180101	2018-01-01	1	st	2	Monday	MON	M	1
2	20180102	2018-01-02	2	nd	3	Tuesday	TUE	T	2
3	20180103	2018-01-03	3	rd	4	Wednesday	WED	W	3
4	20180104	2018-01-04	4	th	5	Thursday	THU	T	4
5	20180105	2018-01-05	5	th	6	Friday	FRI	F	5
6	20180106	2018-01-06	6	th	7	Saturday	SAT	S	6
7	20180107	2018-01-07	7	th	1	Sunday	SUN	S	7
8	20180108	2018-01-08	8	th	2	Monday	MON	M	8
9	20180109	2018-01-09	9	th	3	Tuesday	TUE	T	9
10	20180110	2018-01-10	10	th	4	Wednesday	WED	W	10

	DayOfYear	WeekOfMonth	WeekOfYear	Month	MonthName	MonthName_Short	MonthName_FirstLetter	Quarter	QuarterName
1	1	1	1	1	January	JAN	J	1	First
2	2	1	1	1	January	JAN	J	1	First
3	3	1	1	1	January	JAN	J	1	First
4	4	1	1	1	January	JAN	J	1	First
5	5	1	1	1	January	JAN	J	1	First
6	6	1	1	1	January	JAN	J	1	First
7	7	2	2	1	January	JAN	J	1	First
8	8	2	2	1	January	JAN	J	1	First
9	9	2	2	1	January	JAN	J	1	First
10	10	2	2	1	January	JAN	J	1	First

Year	MMYYYY	MonthYear	IsWeekend
2018	012018	2018JAN	0
2018	012018	2018JAN	0
2018	012018	2018JAN	0
2018	012018	2018JAN	0
2018	012018	2018JAN	0
2018	012018	2018JAN	1
2018	012018	2018JAN	1
2018	012018	2018JAN	0
2018	012018	2018JAN	0
2018	012018	2018JAN	0

Dimension Table Call Charges



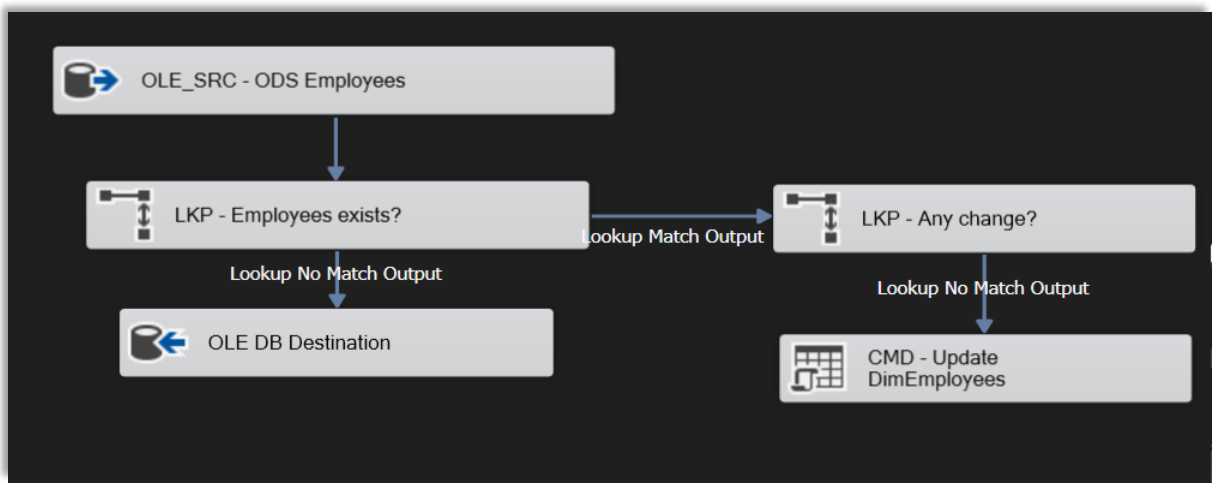
Here is what Dimension Table Call Charges will look in SSIS.
The SQL script to generate the Call Charges table is

```
USE [proj_DWH]
GO

/****** Object:  Table [dbo].[DimCall_Charges]    Script Date: 10/14/2023 4:03:17 PM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[DimCall_Charges](
    [CallChargesKey] [int] IDENTITY(1,1) NOT NULL,
    [CallCharges] [numeric](18, 2) NULL,
    [Year] [int] NULL,
    [CallType] [nvarchar](100) NULL,
PRIMARY KEY CLUSTERED
(
    [CallChargesKey] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

We will get the following table for the Call Charges Dimension table.

	CallChargesKey	CallCharges	Year	CallType
1	43	1.52	2018	Sales
2	44	1.56	2019	Sales
3	45	1.60	2020	Sales
4	46	1.71	2021	Sales
5	47	1.20	2018	Billing
6	48	1.32	2019	Billing
7	49	1.41	2020	Billing
8	50	1.45	2021	Billing
9	51	0.95	2018	Tech Support
10	52	0.98	2019	Tech Support
11	53	1.04	2020	Tech Support

Dimension Table Employees



Here is what Employees Dimension table will look like in SSIS
This table has been generated with the following SQL script.

```
USE [proj_DWH]
GO

/****** Object:  Table [dbo].[DimEmployees]    Script Date: 10/14/2023 4:06:05 PM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[DimEmployees](
    [EmployeesKey] [int] IDENTITY(1,1) NOT NULL,
    [EmployeeID] [nvarchar](10) NULL,
    [EmployeeName] [nvarchar](50) NULL,
    [ManagerName] [nvarchar](50) NULL,
    [City] [nvarchar](50) NULL,
    [State] [nvarchar](50) NULL,
    [Region] [nvarchar](20) NULL,
PRIMARY KEY CLUSTERED
(
    [EmployeesKey] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

The following Dimension table called DimEmployees will be generated in DWH database.

	EmployeesKey	EmployeeID	EmployeeName	ManagerName	City	State	Region
1	1	N772493	Onita Trojan	Deidre Robbs	Spokane	Washington	West
2	2	F533051	Stormy Seller	Elsie Taplin	Aurora	Colorado	West
3	3	S564705	Mable Ayoub	Shala Lion	Aurora	Colorado	West
4	4	I281837	Latrisha Buckalew	Rana Taub	Aurora	Colorado	West
5	5	Y193775	Adrianna Duque	Collin Trotman	Spokane	Washington	West
6	6	J632516	Keiko Daulton	Jamar Prahl	Spokane	Washington	West
7	7	G727038	Dolores Lundeen	Shala Lion	Aurora	Colorado	West
8	8	V126561	Wilbur Mohl	Casey Bainbridge	Jacksonville	Florida	South
9	9	E243130	Ileen Bornstein	Gonzalo Lesage	Jacksonville	Florida	South
10	10	C206355	Janeth Roesler	Miyoko Degraw	Spokane	Washington	West
11	11	G586239	Shery Hover	Elsie Taplin	Aurora	Colorado	West

Use Case

On successful completion of Data warehouse phase we can think of one query.

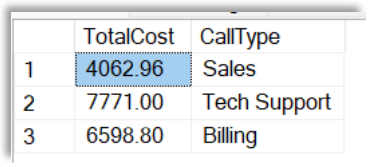
Suppose we want to know the total cost of every call type for the year 2018.

Approach - It is the Call Charges table that contains the cost of each call type. We will join this with fact table and date table.

Now we can filter out records for year 2018 and group by call type. Now we can sum them to know total cost.

```
  SELECT SUM(ch.CallCharges) AS TotalCost, ch.CallType AS CallType FROM
[proj_DWH].[dbo].[Fact_Calls_data] AS ft
    INNER JOIN [proj_DWH].[dbo].DimDate AS dt ON dt.DateKey = ft.DateKey
       INNER JOIN [proj_DWH].[dbo].DimCall_Charges AS ch ON ch.CallChargesKey =
ft.CallChargesKey
       WHERE dt.Year = '2018'
       GROUP BY ch.CallType
```

This results in following output.



	TotalCost	CallType
1	4062.96	Sales
2	7771.00	Tech Support
3	6598.80	Billing

Conclusion

Some choice we made:

- **Call_Type table**

  This is a very small table, so we didn't make it become a dimensional table. at in the final data warehouse. We chose to replace the index in the fact table by the full call type name.

- **Call_Charges table**

  In this table we have 3 colums [CallCharges], [Year] and [CallType]. We chose to select both [CallCharges] and [Year] as source key in DWH phase.

- **Calls_data**

We chose this table to be the fact table at DWH phase. We split the data and time into 2 columns. In our case we have only one fact table with two dimensional table linked to it, and there is no dimensional table of dimensional table.

In the culmination of our recent data warehousing project, we have not only delivered a robust solution tailored to the IT company's requirements but also gleaned invaluable insights along the way. The complexity of collating data from disparate sources and engineering it into a comprehensible and efficient data warehouse underscored the significance of meticulous planning, adept technological prowess, and fluid collaboration.

Throughout this journey, our team witnessed the transformative power of SQL Server Integration Services (SSIS) and SQL server. It served as a reminder that, while tools are essential, it is the innovative application of these tools that creates real value. Designing a suitable data warehouse structure required us to think critically about the company's present needs while anticipating potential future requirements. As we applied transformations using SSIS, the versatility and adaptability of the tool became evident, facilitating seamless data integration.

Our success in this project can be attributed to the cohesive collaboration of our team members. Every challenge encountered was met with collective brainstorming and proactive problem-solving. The culmination of our efforts is evident in the comprehensive solution delivered, which includes every SQL script and a detailed documentation encapsulating our strategic decisions and warehouse schema.

In reflection, this project was as much about team constructive collaboration and growth as it was about data engineering. It reinforced our belief that when technology meets teamwork, even the most intricate challenges can be overcome with finesse.