ETL PROJECT AND DATAWAREHOUSING

Hamza BOUKHALFA
Marie-Laure JOVIAL
Cedric Wilfried KOFFI DJIVO
Sajeevan SOTHIRASA
Naïma TAILEB

Contents

INTRODUCTION - CONTEXT 2

DATASET 2

PIPELINE DESIGN 3

DATA MODEL 3

ETL PROCESS VISUALIZATION 4

STA STAGE 5

- Call Charges Table 5
- Call Type Table 6
- Employees Table 7
- US States Table 8
- Calls Data Table 9

ODS STAGE 11

- ODS - Employee Table 11
- ODS – Site table 13
- ODS - Call Charges table 17
- ODS - Call Data table 22

DWH STAGE 28

Datawarehousing 28

Our Database Design 28

- DimDate Integration 29
- DimSites Integration 30
- DimEmployee Integration 31
- Fact Table Call Data Integration 32

INTRODUCTION - CONTEXT

ServiceSpot, an IT company, has enlisted our team to help them analyze their call center data. They receive daily calls from their customers and wish to gain insights into their operations. The data is currently scattered across multiple files, making it difficult to analyze effectively. Our mission is to develop an Extract, Transform, Load (ETL) project using SSIS to consolidate and load data into a new enterprise data warehouse. As the data engineering team, our primary focus is to provide clean and structured data for the business. Subsequent report generation will be handled by a separate team of data analysts who will utilize the data warehouse.

DATASET

Our data encompasses:

- Data call folder
  The folder contains 3 csv files (Data YYYY) call data, one year per file.

Column Name	Description
CallTimestamp	Date & Time of call
Call Type	Type of call
EmployeeID	Employee unique ID
CallDuration	Duration of call, in seconds
WaitTime	Wait time, in seconds
CallAbandoned	Was the call abandoned by the customer? (1 = Yes, 0 = No)

- Employee CSV

Column Name	Description
EmployeeID	Employee unique ID
EmployeeName	Employee full name
Site	Site name where the employee is working at
ManagerName	Employee's Manager

- Call Type CSV

Column Name	Description
CallTypeID	Unique ID for Call Type
CallTypeLabel	Call type label

- US States CSV
  This file contains information about every state

Column Name	Description
StateCD	2-letter state code
Name	Name of the state
Region	US region name (East, West, etc.)

- Call Charges CSV
  This file contains the call charges (i.e. the amount of money that is charged to customers) per minute, and for each year

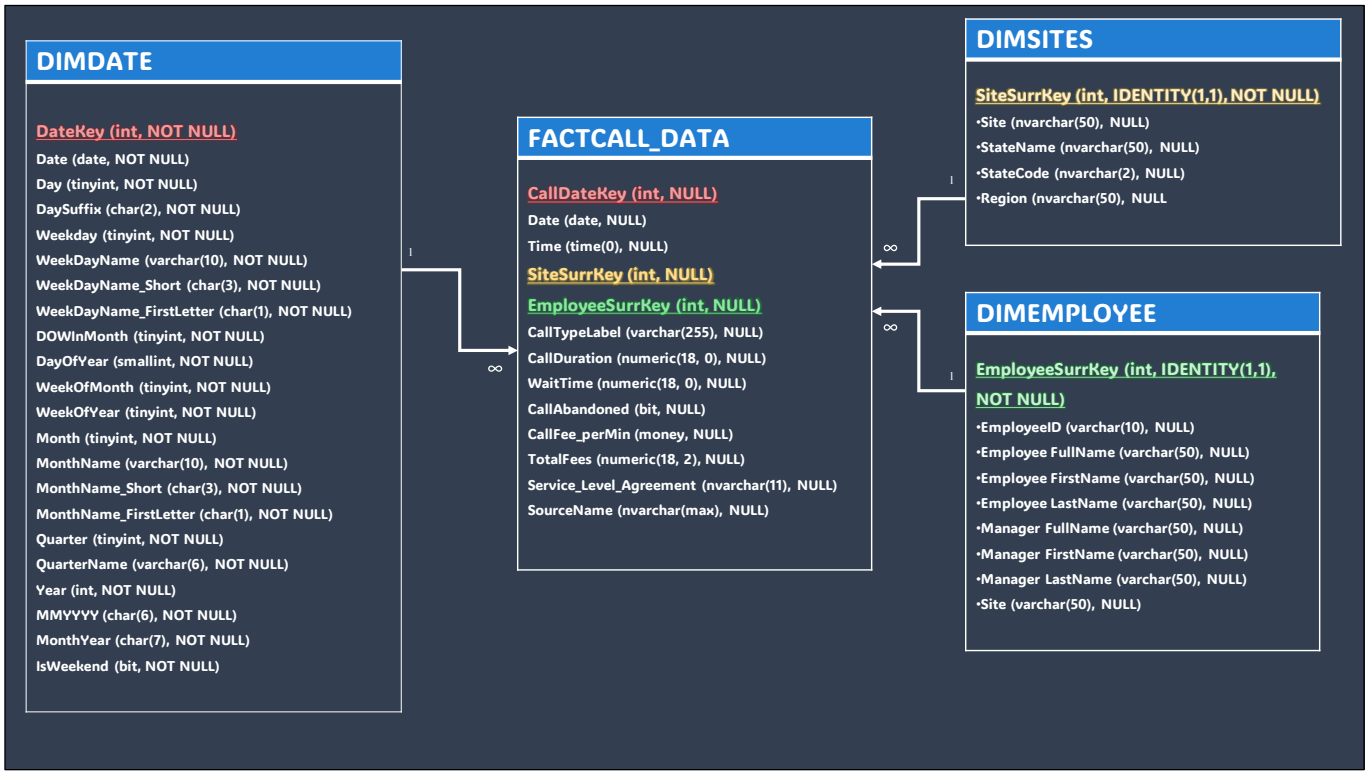
Column Name	Description
Call Type Key	Unique ID for Call Type
Call Type	Call type label
Call Charges / Min (YYYY)	The amount of money that is charged to a customer for each minute spent on the phone, for a specific year (YYYY)

PIPELINE DESIGN

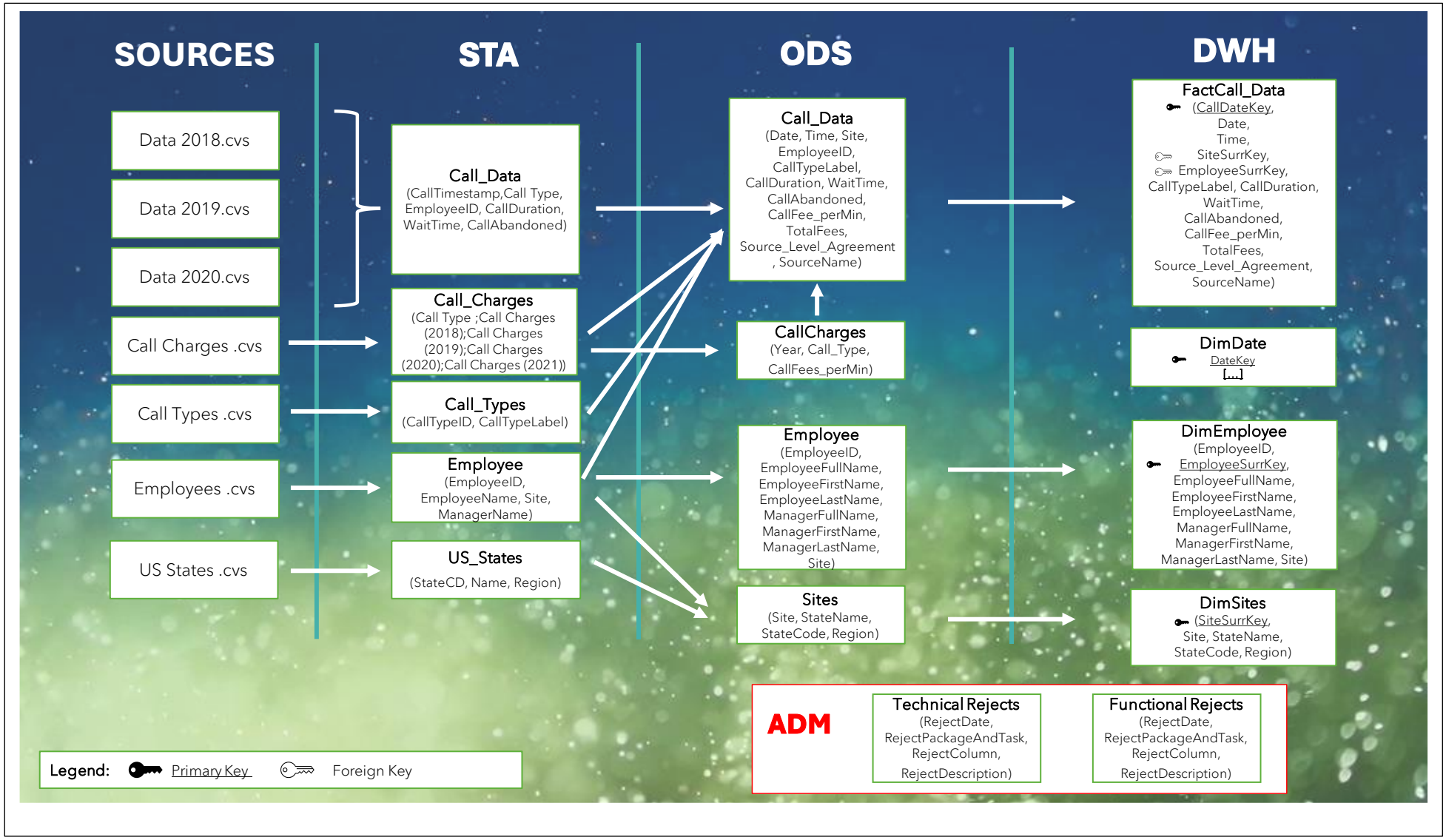
The process will unfold in three phases:

- The Staging Area (STA) will serve to either load the data in its current state or with minimal modifications.
- The Operational Data Store (ODS) area will be used to cleanse and standardize the data. If the data fails to meet the specified quality standards, it will be stored in the "Technical Rejects" table as technical rejections.
- The Data Warehouse (DWH) section will structure the data into one fact table linked to several dimension tables. Records that do not fit the schema will be categorized as functional rejections and placed in the "Functional Rejects" table. In some cases, temporary placeholder relationships may be established.

DATA MODEL



ETL PROCESS VISUALIZATION



STA STAGE

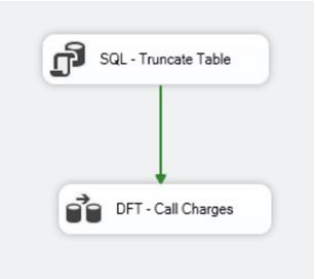
The ingestion of most of our data at this stage is straightforward. It consists in creating a new package in SSIS named according to the data, create a dataflow in which we ingest this data via Flat file to store it in a newly created table into the database created for this stage i.e. ETL_PROJECT_STA.
Each table will also be truncated before the process.

Call Charges Table

Here is an extract of the data from the call charge 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

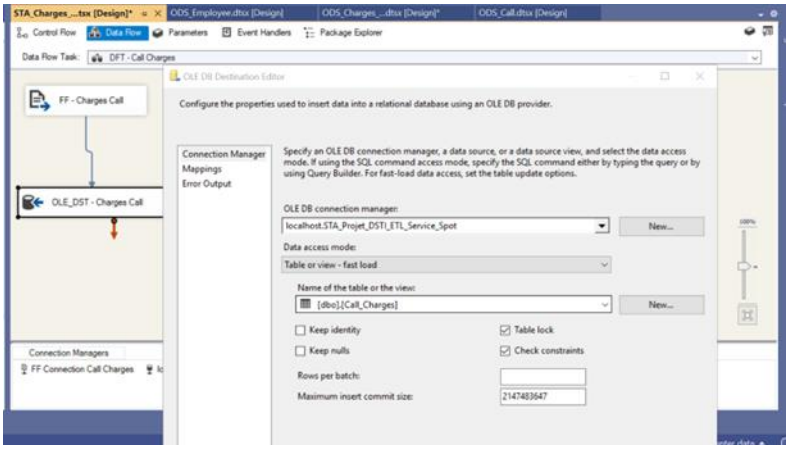
Given the data, we will ingest this data into a table in the STA database without any addition or transformation.



After creating a package in SSIS and creating a Data Flow Task

We will ingest the data via a Flat File to an OLE DB Destination with the newly created table. We previously created our ETL_PROJECT_STA database under SQL SERVER

```
USE [ETL_PROJECT_STA]
GO

/****** Object:  Table [dbo].[Call_Charges]    Script Date: 5/10/2024 2:19:31 AM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

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



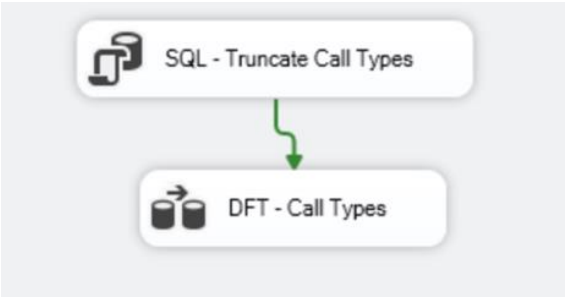
Call Type Table

Here is an extract of the data from the call charge file:

CallTypeID	CallTypeLabel
1	Sales
2	Billing
3	Tech Support

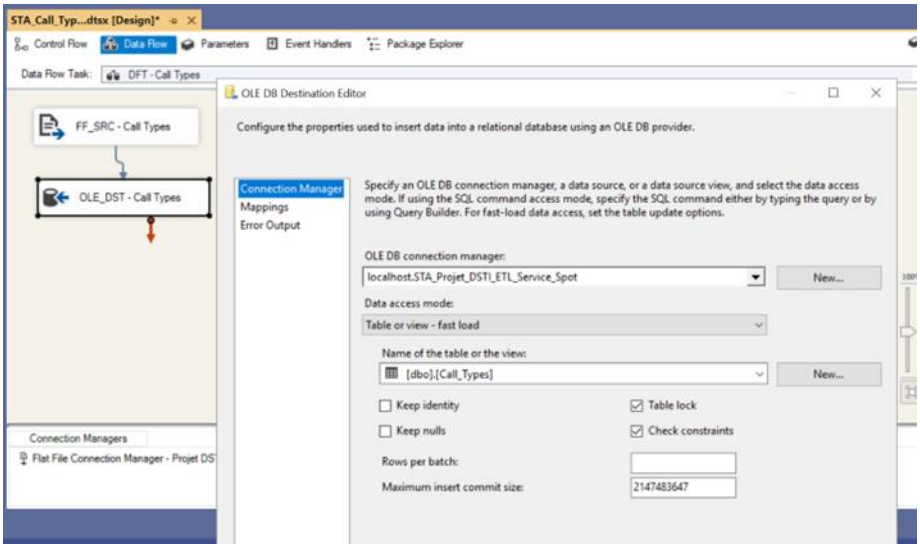
Given the data, we will ingest this data into a table in the STA database without any addition or transformation.

After creating a package in SSIS and creating a Data Flow Task



We will ingest the data via a Flat File to an OLE DB Destination for the newly created table. We first created our ETL_PROJECT_STA database under SQL SERVER

```
USE [ETL_PROJECT_STA]
GO

/****** Object:  Table [dbo].[Call_Types]    Script Date: 5/10/2024 2:18:00 AM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

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



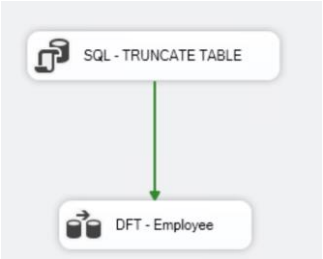
Employees Table

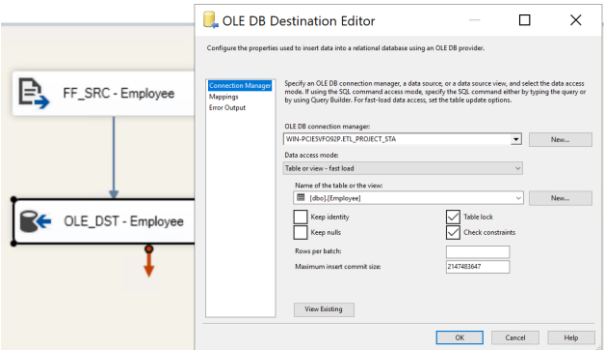
Here is an extract of the data from the employee file (5 first rows out of 65 rows):

EmployeeID	Employee Name	Site	Manager Name
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

Given the data, we will ingest this data into a table in the STA database without any addition or transformation.

After creating a package in SSIS and creating a Data Flow Task, we will ingest the data via a Flat File to an OLE DB Destination for the newly created table.





The SQL query below was used to create the destination table named 'Employee' in ETL_PROJECT_STA.

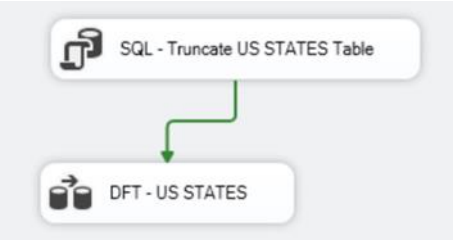
```
USE [ETL_PROJECT_STA]
GO

/****** Object:  Table [dbo].[Employee]    Script Date: 5/10/2024 2:15:51 AM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

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

US States Table

Here is an extract of the data from the US States file:

StateCD	Name	Region
AK	Alaska	West
AL	Alabama	South
AR	Arkansas	South
AZ	Arizona	West

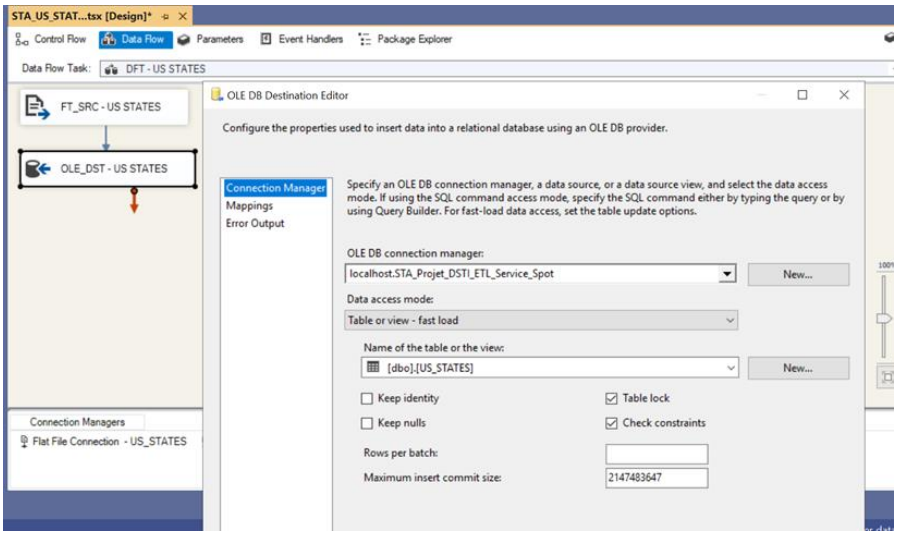
Given the data, we will ingest this data into a table in the STA database without any addition or transformation.



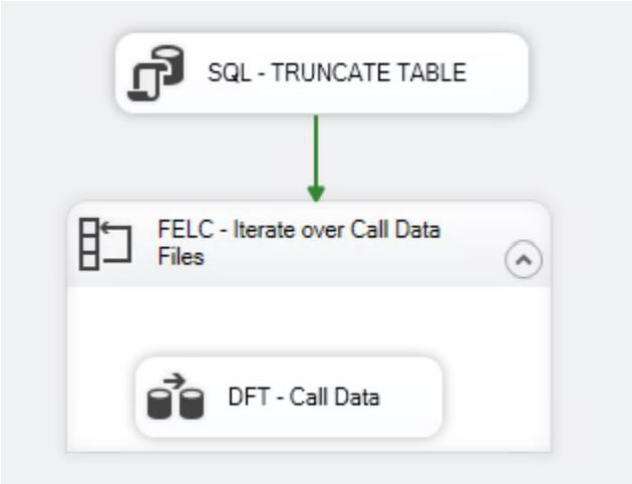
After creating a package in SSIS and creating a Data Flow Task

We will ingest the data via a Flat File to an OLE DB Destination for the newly created table. We first created our ETL_PROJECT_STA database under SQL SERVER

```
USE [ETL_PROJECT_STA]
GO

/****** Object:  Table [dbo].[US_States]    Script Date: 5/10/2024 2:11:09 AM ******/
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
```



Calls Data Table

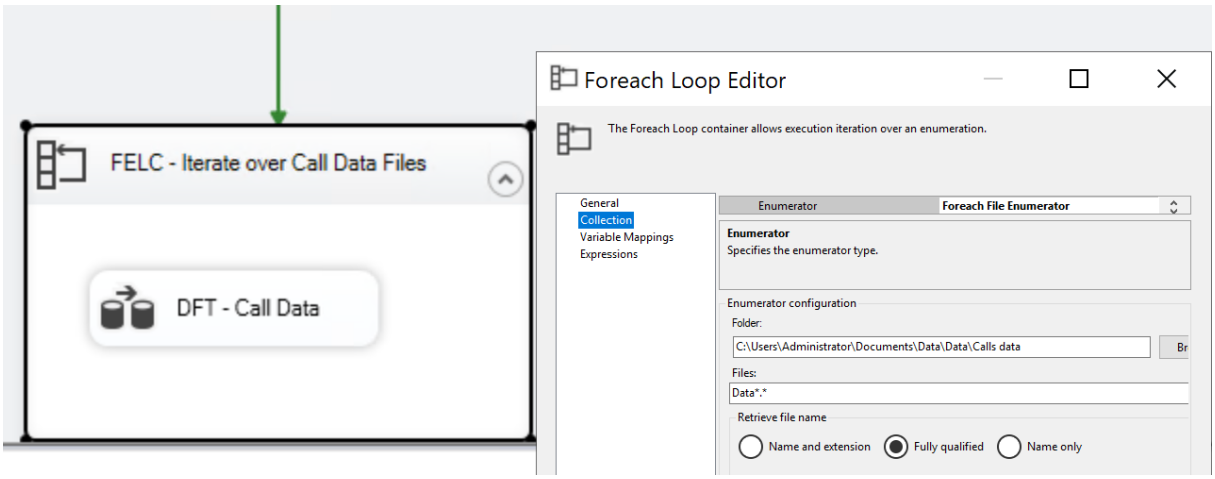


Compared to our previous data, our call data information is split into 3 separate files "*Call Data XXXX*" gathered in one folder.

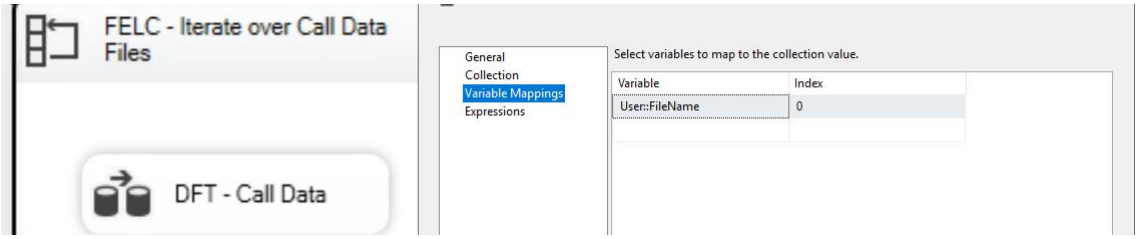
We would like to load these files to one unique table.

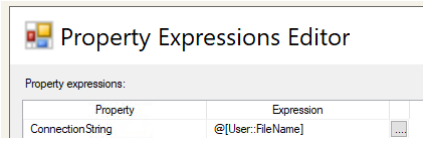
Remark: All files have the same structure.

To do so, after creating our SSIS package, in our dataflow, we add a Foreach Loop Container task that we configure to iterate the flow over each file containing the word "*Data*."



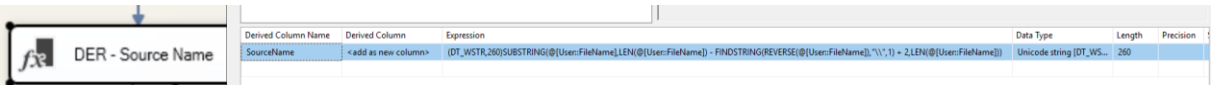
We also create a variable that will be later used to configure our connection manager.



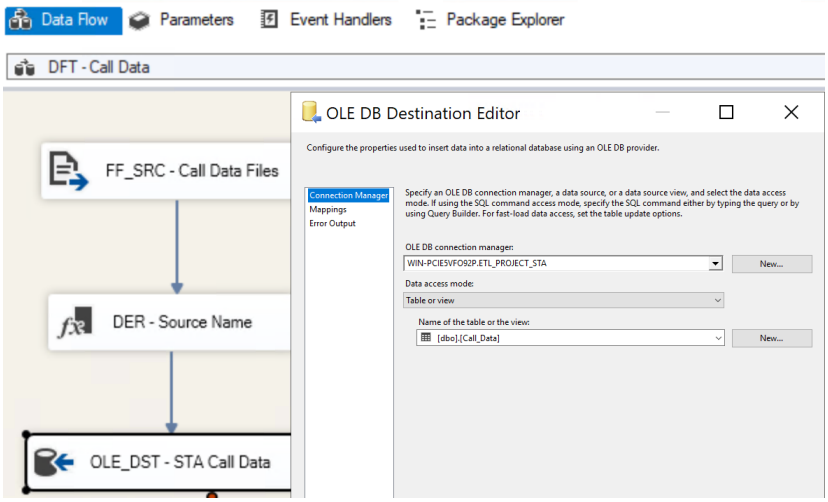


As we are ingesting data from different sources in the same table, we add the specific source name in a new column to keep track of the specific origin of each row. We use the following SQL query to extract the name of the file:

((DT_WSTR,260)SUBSTRING(@[User::FileName],LEN(@[User::FileName]) - FINDSTRING(REVERSE(@[User::FileName]),"\\",1) + 2,LEN(@[User::FileName]))



After creating a package in SSIS and creating a Data Flow Task, we will ingest the data via a Flat File to an OLE DB Destination for the newly created table.



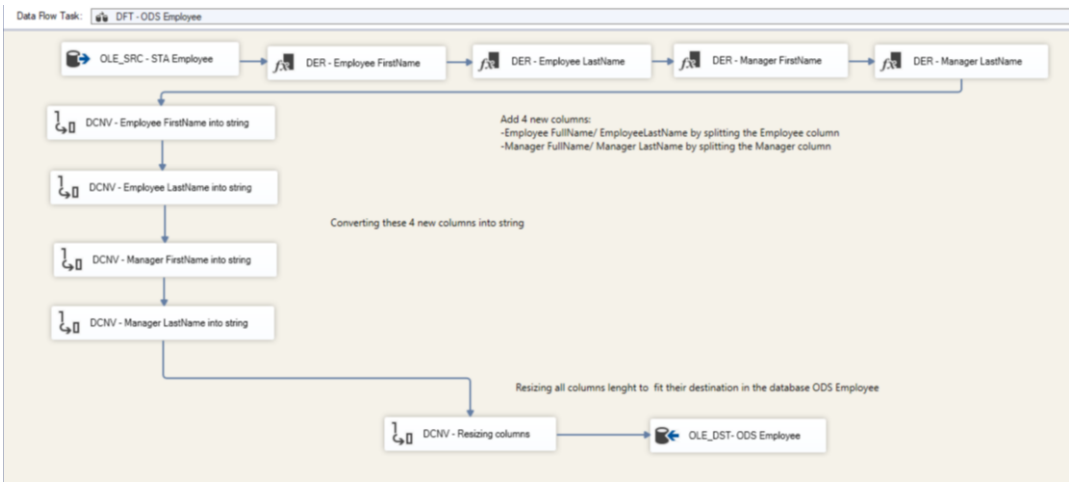
The SQL query below was used to create the destination table named 'Call_Data' in ETL_PROJECT_STA.

```
USE [ETL_PROJECT_STA]
GO

/****** Object:  Table [dbo].[Call_Data]    Script Date: 5/10/2024 2:20:59 AM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Call_Data](
    [CallTimestamp] [varchar](255) NULL,
    [Call Type] [varchar](255) NULL,
    [EmployeeID] [varchar](255) NULL,
    [CallDuration] [varchar](255) NULL,
    [WaitTime] [varchar](255) NULL,
    [CallAbandoned] [varchar](255) NULL,
    [SourceName] [nvarchar](max) NULL
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
```

ODS STAGE

ODS - Employee Table

OBJECTIVE: On this table, the primary alteration involved introducing new columns to separate the first name and last name of both the manager and the employee. An arbitrary choice was made to keep the initial columns and renamed them as *Employee FullName* and *Manager FullName*.



To achieve this transformation 4 derived columns were created from the source data "Employee": *Employee FirstName, Employee LastName, Manager FirstName and Manager LastName.*

Creation of the 4 new columns thanks to the SQL queries below:

Derived Column Name	Derived Column	Expression	Data Type	Length	Precision	Scale	Code Page
Employee FirstName	<add as new column>	TRIM(LEFT(EmployeeName,FINDSTRING(EmployeeName," ",1)))	Unicode string [DT_WS...	255			

Derived Column Name	Derived Column	Expression	Data Type	Length	Precision	Scale	Code Page
Employee LastName	<add as new column>	TRIM(RIGHT(EmployeeName,LEN(EmployeeName) - FINDSTRING(EmployeeName," ",1)))	Unicode string [DT_WS...	255			

Derived Column Name	Derived Column	Expression	Data Type	Length	Precision	Scale	Code Page
Manager FirstName	<add as new column>	TRIM(LEFT(ManagerName,FINDSTRING(ManagerName," ",1)))	Unicode string [DT_WS...	255			

Derived Column Name	Derived Column	Expression	Data Type	Length	Precision	Scale	Code Page
Manager LastName	<add as new column>	TRIM(RIGHT(ManagerName,LEN(ManagerName) - FINDSTRING(ManagerName," ",1)))	Unicode string [DT_WS...	255			

The white space in the full name was used as the separator to extract the first and last names.

Once created, the columns were converted into string then resized to fit the destination database's length.

Input Column	Output Alias	Data Type	Length	Precision	Scale	Code Page
EmployeeID	EmployeeID_R	string [DT_STR]	10			1252 (ANSI - Latin I)
EmployeeName	EmployeeFullName_R	string [DT_STR]	50			1252 (ANSI - Latin I)
Site	Site_R	string [DT_STR]	50			1252 (ANSI - Latin I)
ManagerName	ManagerFullName_R	string [DT_STR]	50			1252 (ANSI - Latin I)
Employee FirstName_str	Employee FirstName_R	string [DT_STR]	50			1252 (ANSI - Latin I)
Employee LastName_str	Employee LastName_R	string [DT_STR]	50			1252 (ANSI - Latin I)
Manager FirstName_str	Manager FirstName_R	string [DT_STR]	50			1252 (ANSI - Latin I)
Manager LastName_str	Manager LastName_R	string [DT_STR]	50			1252 (ANSI - Latin I)

The destination database was created with can be recreated with the SQL Query below:

```sql
USE [ETL_PROJECT_ODS]
GO

/****** Object:  Table [dbo].[Employee]    Script Date: 5/10/2024 2:25:04 AM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Employee](
    [EmployeeID] [varchar](10) NULL,
    [Employee FullName] [varchar](50) NULL,
    [Employee FirstName] [varchar](50) NULL,
    [Employee LastName] [varchar](50) NULL,
    [Manager FullName] [varchar](50) NULL,
    [Manager FirstName] [varchar](50) NULL,
    [Manager LastName] [varchar](50) NULL,
    [Site] [varchar](50) NULL
) ON [PRIMARY]
GO
```

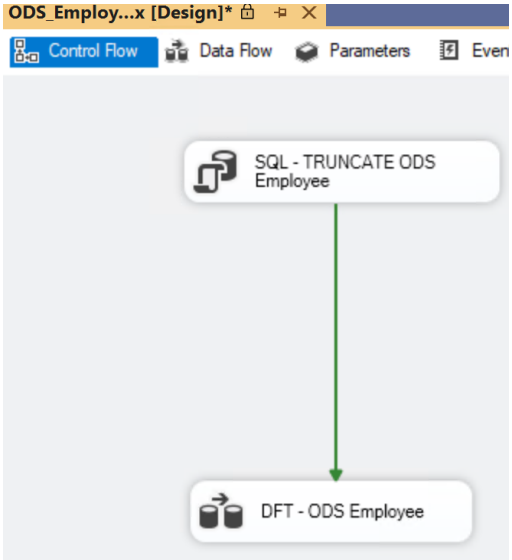
All resized columns were then mapped with their corresponding column in the destination database:

Input Column	Destination Column
Site_R	Site
ManagerFullName_R	Manager FullName
Manager LastName_R	Manager LastName
Manager FirstName_R	Manager FirstName
EmployeeID_R	EmployeeID
EmployeeFullName_R	Employee FullName
Employee LastName_R	Employee LastName
Employee FirstName_R	Employee FirstName

The result obtained in SQL Server:

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

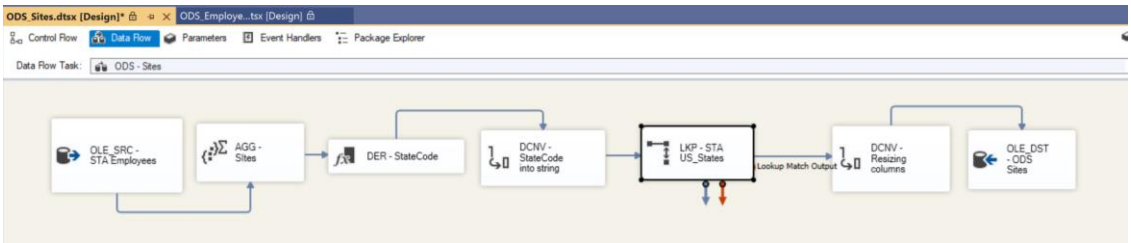
Note that the table is truncated at the beginning of the launch of the dataflow.



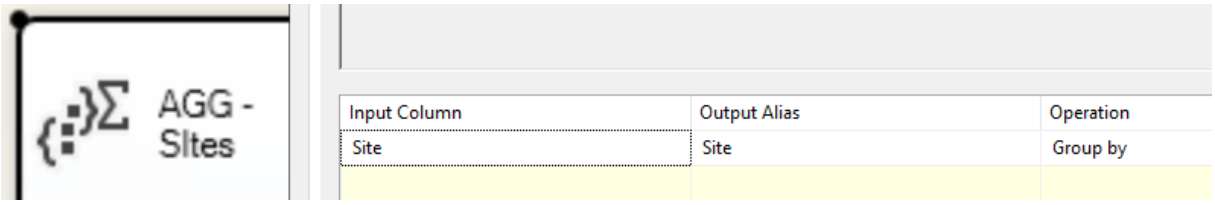
ODS – Site table

OBJECTIVE: Have a table with the different call center sites and their related geographical information.

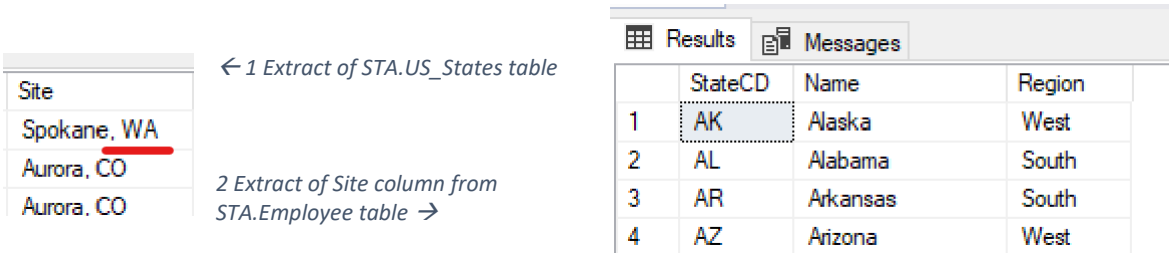
To create this table, we will extract the distinct sites names from STA Employee table and fetch additional information such as the (*Region, Code district and the City name*) on the STA US States table.

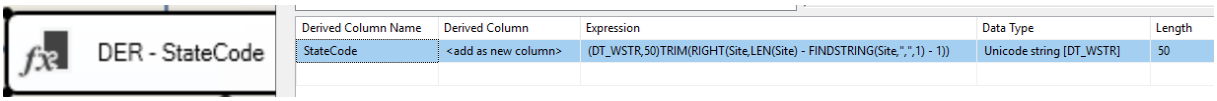


- **STEP 1: We extract the column Site of the STA Employee table and use an aggregation** to avoid duplicate sites.

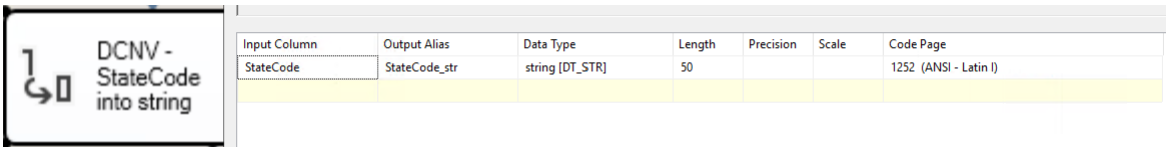


- **STEP 2:** To perform a lookup on the US_States table, we needed a common column between the two tables i.e., *Employee* and *US_State*. As we did not have any common column, we had to **create a new column *StateCode* that is the extraction on the two last characters of the *Site* column (or every character after the comma).**

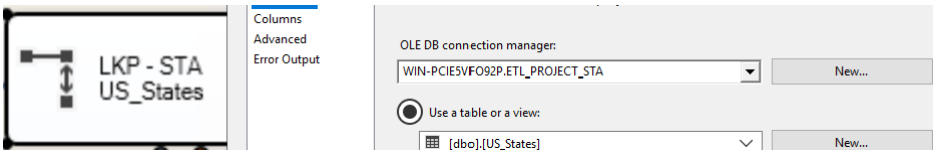


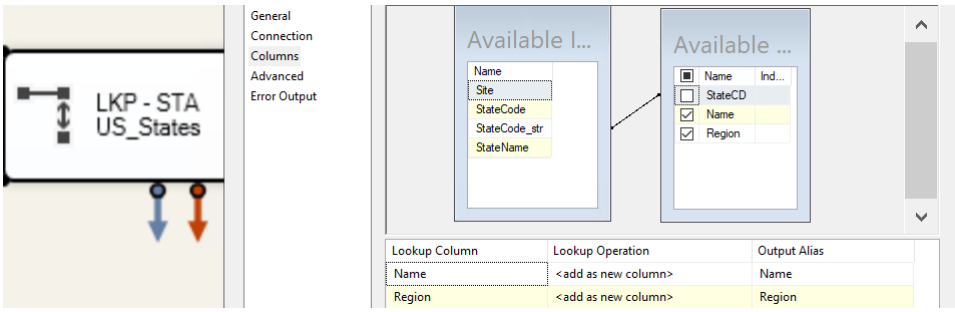


- **STEP 3: Converting the new column into the same datatype to be able to perform the look up.** Indeed, the *StateCode* column on the we just created is in WSTRING whereas its equivalent on the (STA)Employee table is in STRING. To overcome this, we are going to convert the new column to STRING.

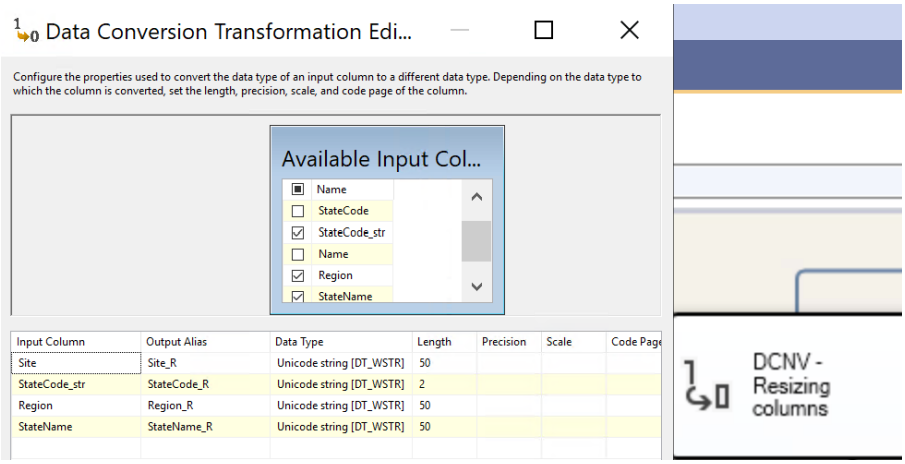


- **STEP 4: Performing the look up on the STA US_States table to retrieve *Name* and *Region* information corresponding to each call center site.**





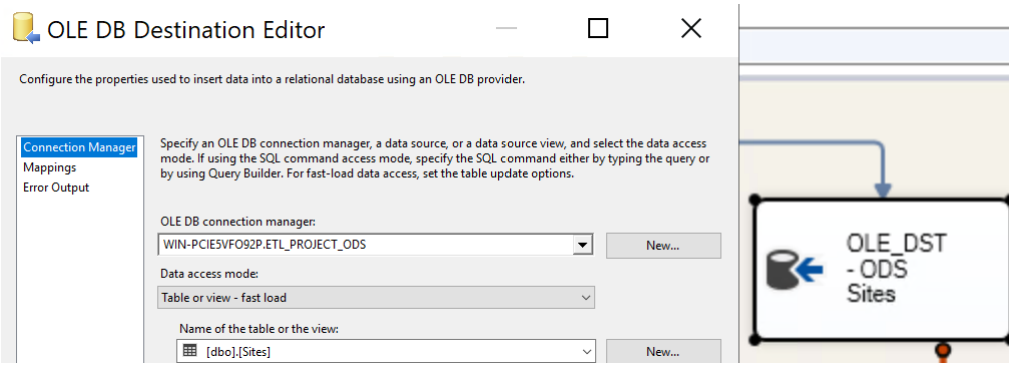
- **STEP 5: The columns we want to import in our database are resized** to match their destination in the database OSD Sites.

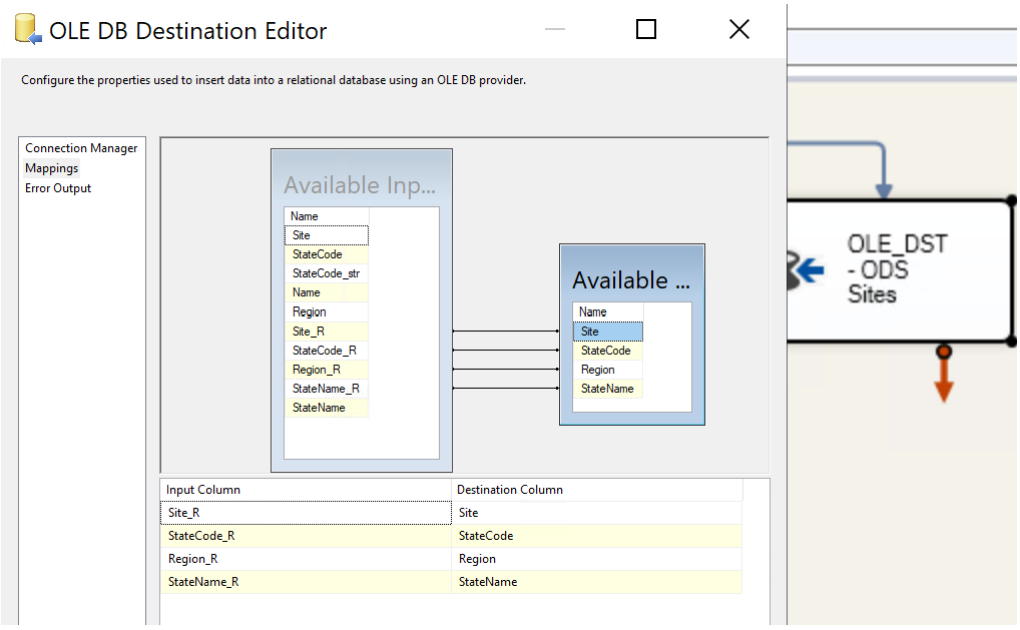


Below the database created to store the site information:

```
USE [ETL_PROJECT_ODS]
GO

/****** Object:  Table [dbo].[Sites]    Script Date: 5/10/2024 2:23:25 AM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Sites](
    [Site] [nvarchar](50) NULL,
    [StateName] [nvarchar](50) NULL,
    [StateCode] [nvarchar](2) NULL,
    [Region] [nvarchar](50) NULL
) ON [PRIMARY]
GO
```

- **STEP 6: Mapping all resized columns to their precise destination in the ODS database on the Site table.**





- **STEP 7: Visualizing the result on SQL Server.**

We only have 3 distinct sites so our table will have 3 rows.

Results | Messages

	Site	StateName	StateCode	Region
1	Spokane, WA	Washington	WA	West
2	Jacksonville, FL	Florida	FL	South
3	Aurora, CO	Colorado	CO	West

ODS - Call Charges table

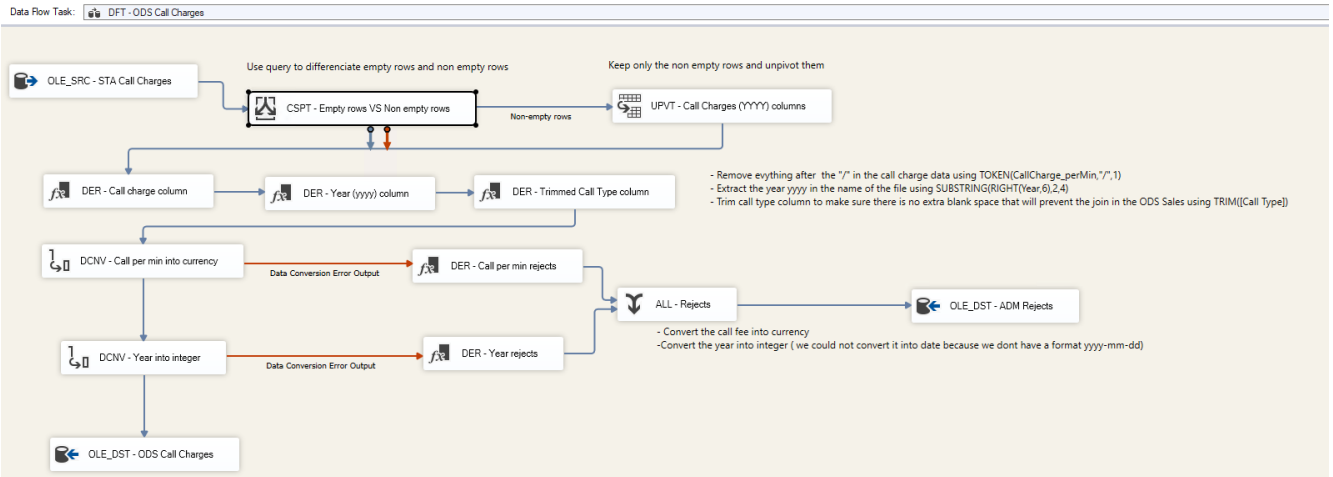
OBJECTIVE: Transform the data into a usable format by unpivoting the columns. The outcome should enable a lookup to extract the call charges value for each year and call type.

	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
4					
5					
6					

3 Extract of original data source.

Year	Call_Type	CallFees_perMin
2018	Sales	1.52
2019	Sales	1.56
2020	Sales	1.60
2021	Sales	1.71
2018	Billing	1.20
2019	Billing	1.32
2020	Billing	1.41
2021	Billing	1.45
2018	Tech Support	0.95
2019	Tech Support	0.98
2020	Tech Support	1.04
2021	Tech Support	1.12

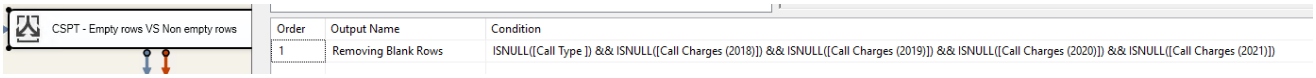
4 Screenshot of the final table in SQL



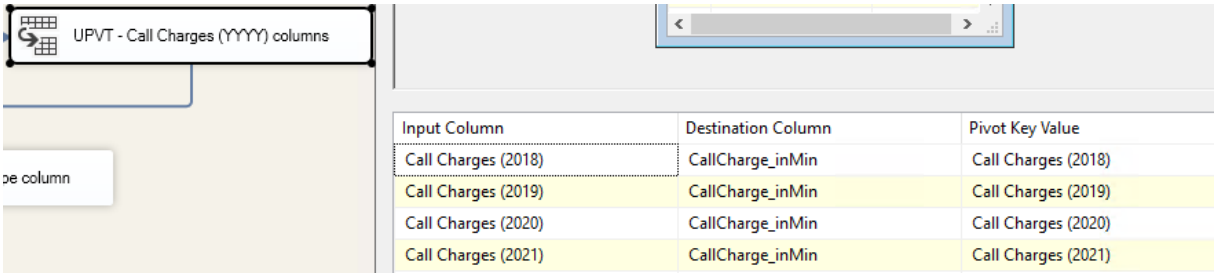
5 Transformation process of the data Call charges.

- **STEP 1: Remove empty rows.**

When analyzing the data, we observe 987 rows whereas only the first 3 rows contain data. To avoid having these blank cells during transformation, we use a conditional split and a SQL query to separate blank rows from non-blank rows and choose to output only the non-blank rows.



- **STEP 2: Unpivoting all the "Call charges (YYYY) columns**.



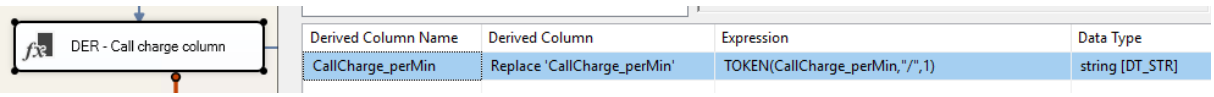
Here is the result after unpivoting the data.

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

To reach the desired result, we still need to remove everything after the "/" in the call charge data and extract the year in the name of the original columns.

- **STEP 3 /4: Replacing the *CallCharge_perMin* column and replacing the year *column.***

We replace *CallCharge_perMin* column using the TOKEN function to extract any substring before the '/' in the column CallCharge_perMin.



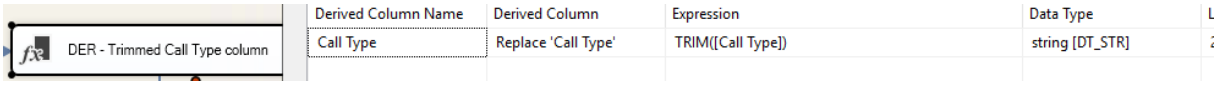
Derived Column Name	Derived Column	Expression	Data Type
CallCharge_perMin	Replace 'CallCharge_perMin'	TOKEN(CallCharge_perMin,"/",1)	string [DT_STR]

We have created a new column that will replace the initial Year column. As the year appears always in the same position, this new column uses a textual SQL query to extract the 4-digit corresponding to the year from the text thanks to a combination of SUBSTRING and RIGHT function.

Derived Column Name	Derived Column	Expression	Data Type
Year	Replace 'Year'	SUBSTRING(RIGHT(Year,6),2,4)	Unicode string [DT_WS...

- **STEP 5: Trimming *Call Type* column.**

The final table of this transformation must be used as a Lookup table with the Call Type column as the Lookup table. During the joint testing, it appears that one of the values could not find its match. To avoid this, the Call Type column was trimmed to remove any invisible extra white space.



Derived Column Name	Derived Column	Expression	Data Type	L
Call Type	Replace 'Call Type'	TRIM([Call Type])	string [DT_STR]	2

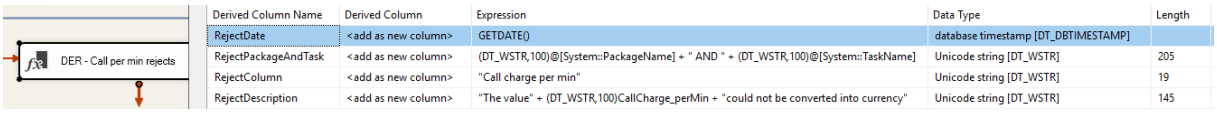
- **STEP 6/7: Choosing datatype for call per min and year column.**

The *CallCharges_perMin* column was converted to currency datatype. The new column was labelled *CallCharges_perMin_CY.*

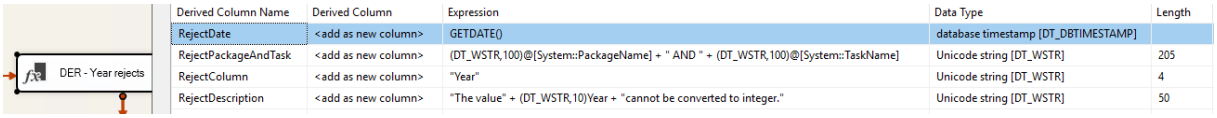
The column *Year* was converted to a four-integer datatype. Converting year into date was not possible as a date must be in a yyyy/mm/dd format to be converted to year. The new column was labelled *Year_int.*

- **STEP 8: Configuring rejects messages in case of conversion error and send them to *ADM Rejects* Database**

We set up these four new columns for storage purposes, aiding us in monitoring any errors that may arise during the process. These columns will store the date of the error occurrence, the package and task affected by the error, the column name, and a customized error message.

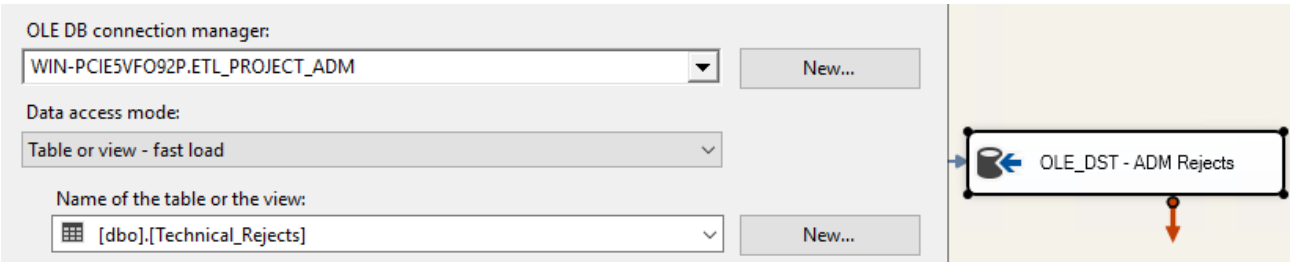


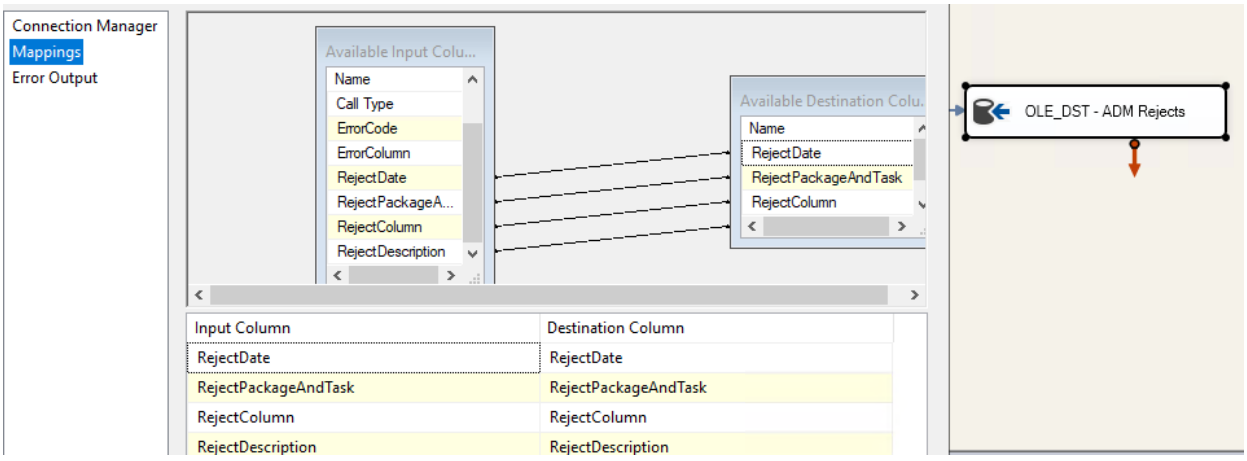
Derived Column Name	Derived Column	Expression	Data Type	Length
RejectDate	<add as new column>	GETDATE()	database timestamp [DT_DBTIMESTAMP]	
RejectPackageAndTask	<add as new column>	(DT_WSTR,100)@[System::PackageName] + " AND " + (DT_WSTR,100)@[System::TaskName]	Unicode string [DT_WSTR]	205
RejectColumn	<add as new column>	"Call charge per min"	Unicode string [DT_WSTR]	19
RejectDescription	<add as new column>	"The value" + (DT_WSTR,100)CallCharge_perMin + "could not be converted into currency"	Unicode string [DT_WSTR]	145



Derived Column Name	Derived Column	Expression	Data Type	Length
RejectDate	<add as new column>	GETDATE()	database timestamp [DT_DBTIMESTAMP]	
RejectPackageAndTask	<add as new column>	(DT_WSTR,100)@[System::PackageName] + " AND " + (DT_WSTR,100)@[System::TaskName]	Unicode string [DT_WSTR]	205
RejectColumn	<add as new column>	"Year"	Unicode string [DT_WSTR]	4
RejectDescription	<add as new column>	"The value" + (DT_WSTR,10)Year + "cannot be converted to integer."	Unicode string [DT_WSTR]	50

These columns are then stored in the Technical Reject table in the ADM database.





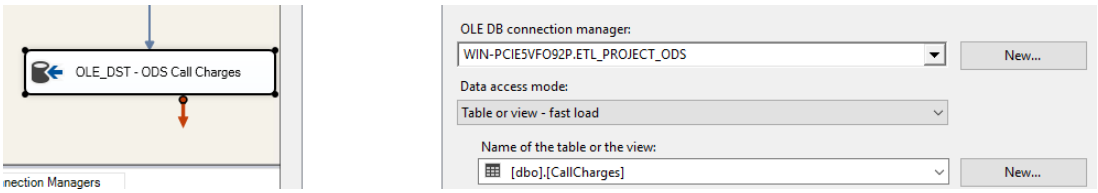
Below the SQL Query used to create the Technical Rejects table in the ADM database.

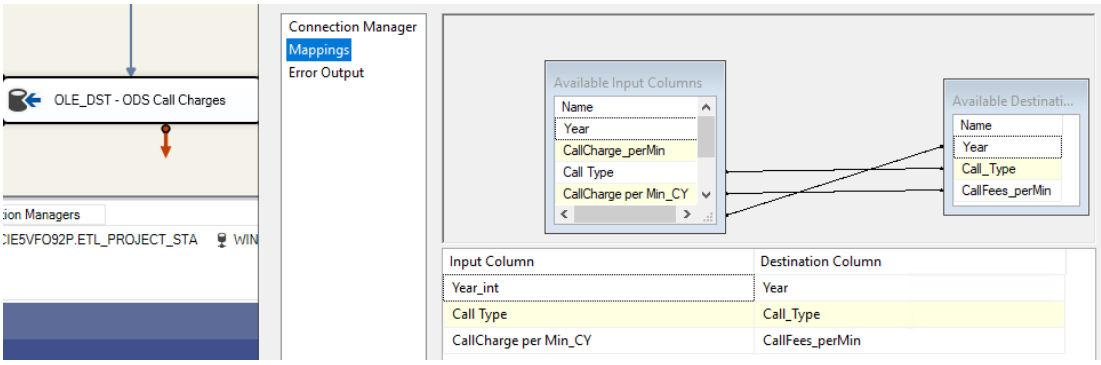
As the column might be of distinct size for each Reject task, we set up the column length to MAX.

```
USE [ETL_PROJECT_ADM]
GO

/****** Object:  Table [dbo].[Technical_Rejects]    Script Date: 5/12/2024 9:56:17 AM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

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

- **STEP 9: Configure the destination database and map the columns.**

The result of the transformation is sent to the PROJECT_ODS database in the table Call Charge. All columns were mapped to their destination in the database.





The SQL table was created with the following query.

```
USE [ETL_PROJECT_ODS]
GO

/****** Object:  Table [dbo].[CallCharges]    Script Date: 5/12/2024 10:09:00 AM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[CallCharges](
    [Year] [int] NULL,
    [Call_Type] [varchar](255) NULL,
    [CallFees_perMin] [money] NULL
) ON [PRIMARY]
GO
```

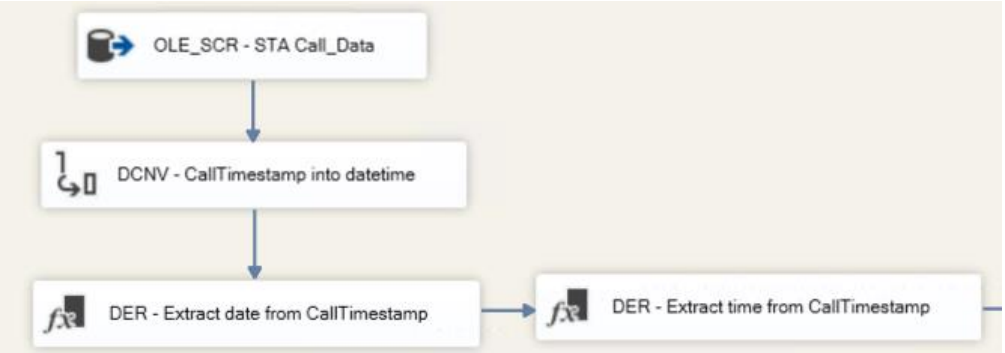
Below is a recall screenshot of the outcome in SQL Server.

Year	Call_Type	CallFees_perMin
2018	Sales	1.52
2019	Sales	1.56
2020	Sales	1.60
2021	Sales	1.71
2018	Billing	1.20
2019	Billing	1.32
2020	Billing	1.41
2021	Billing	1.45
2018	Tech Support	0.95
2019	Tech Support	0.98
2020	Tech Support	1.04
2021	Tech Support	1.12

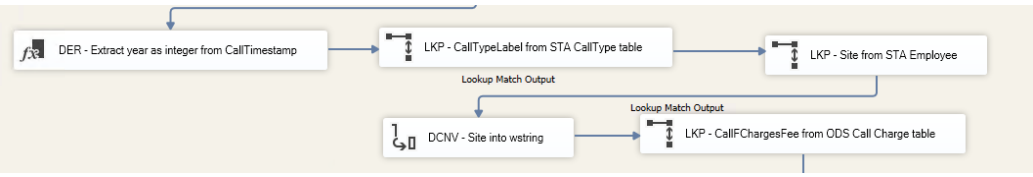
ODS - Call Data table

OBJECTIVE: As the Call Data table serves as our fact table, our primary objective is to ensure that all necessary information from the previously created tables can be retrieved efficiently through lookups, so no information is missing.

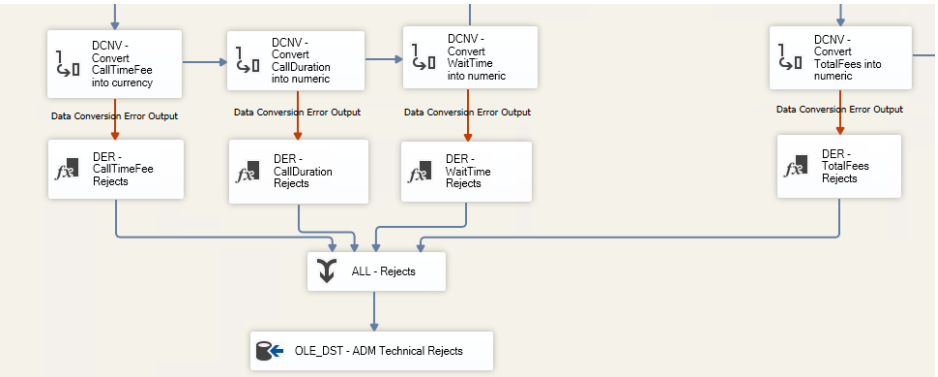
Adhering to best practices, we will begin by separating the time and date components



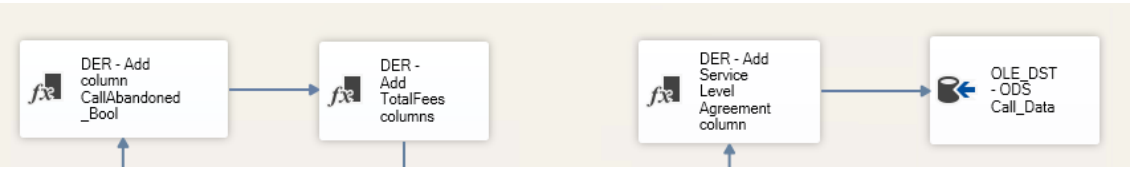
Subsequently, we ensure that all requisite transformations are made, including column additions or datatype conversions, to facilitate lookup tasks on other tables.



Next, all columns undergo conversion to their appropriate data types. It is important to note that data conversion occurs elsewhere in the process. In case of an error during data conversion, derived columns are generated to retain information about the error. This information is then stored in the ADM database within the Technical Rejects table.

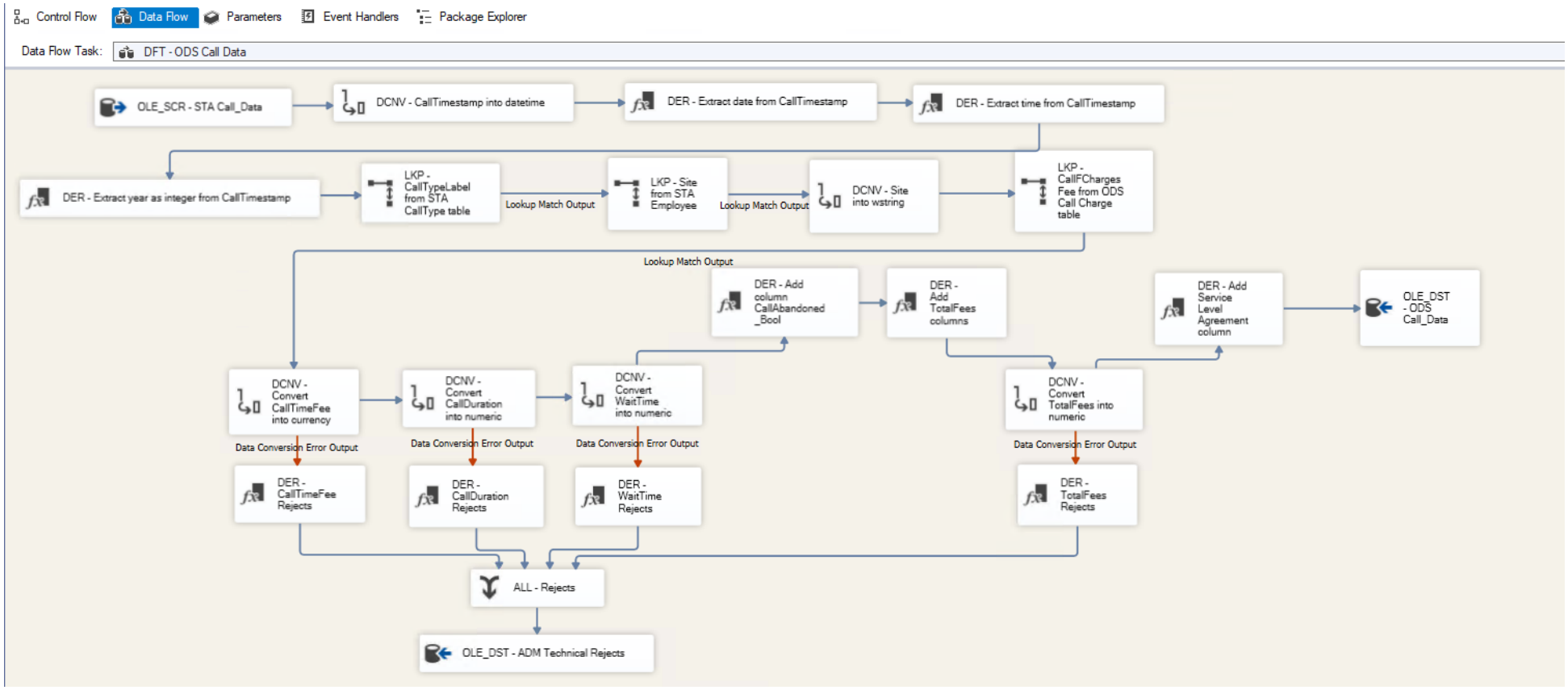


Additional columns to complete the initial business request and other columns for eventual future business analysis are finally added.



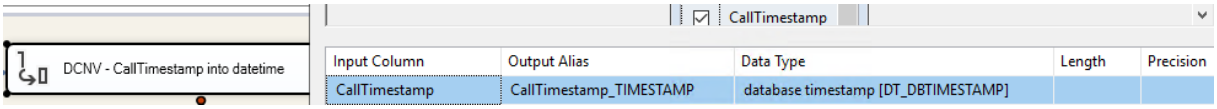
Overview of the ODS Call Data Dataflow

6 Overview of the ODS Call Data Dataflow

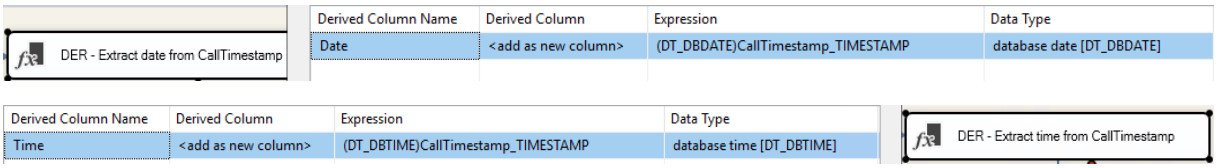


STEP 1 - 4: Splitting Time and Date information

After retrieving the data from the STA database, we first start to convert the *CallTimestamp* column into a datetime format.

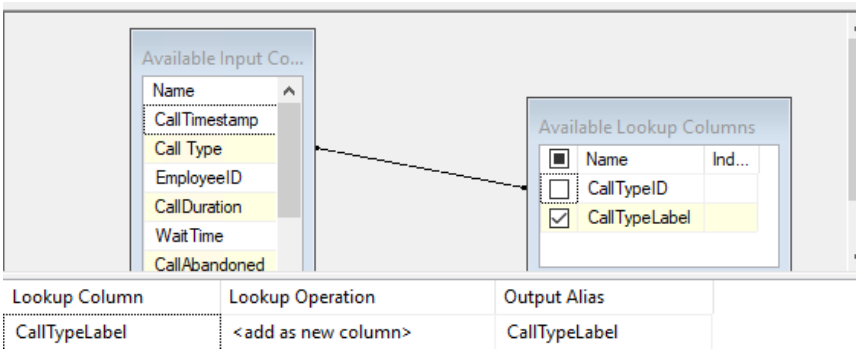


Thanks to this transformation, we used the SQL DATE function and the SQL TIME function to create a date column and a time column.

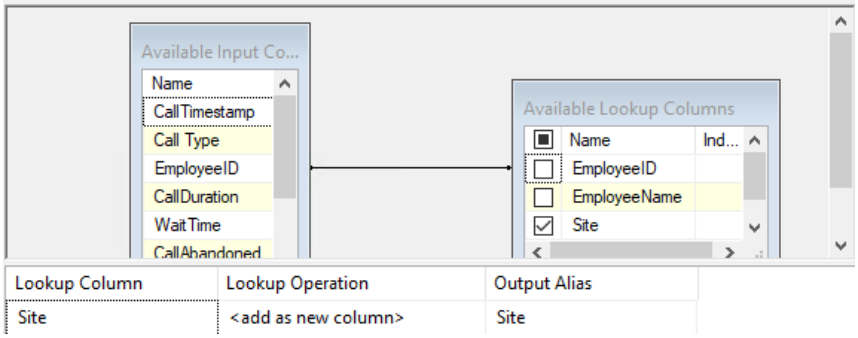


STEP 5 - 9: Performing lookups to retrieve information from other tables.

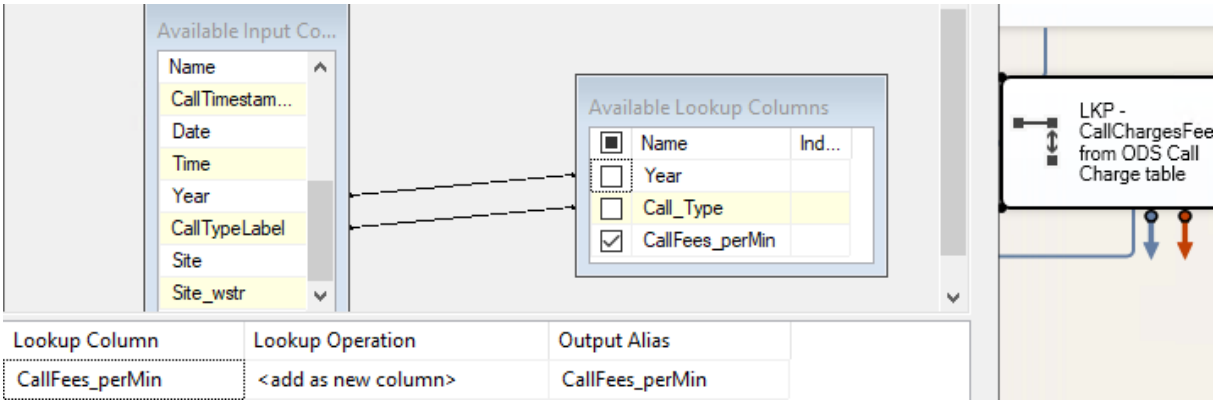
- Step 6: For the Call Data table to be more explicit, we would like to have the *CallTypeLabel* instead of the *CallTypeID* (e.g. "Invoices" instead of "1") so we perform a lookup on the Call Type table in STA database using *CallTypeID* as the matching column.



- Step 7-8: To have a relationship between the site table and the Call Data table as defined in our model, we need one common column. We have chosen to import the site information from the employee table in the STA database. Whenever the corresponding employee is found, the SiteName is retrieved. The SiteName was extracted I then converted to wstring.



- Step 5 and 9: Instead of keeping the call charges information in a separate table, we chose to directly integrate them into the Call Data table. For the lookup on the ODS.Call Charges table, we need to pay attention both to the Year and the *CallTypeLabel* column to have the right *CallCharge_perMin*.



As we had converted *Year* into integer on the Call charges table, we converted year into integer. (step 5). Having done so, we can now match the Year and *CallTypeLabel* of both tables and retrieve the *CallCharges_perMin* corresponding values.

STEP 10 - 17: **Converting column to the right datatype and configure reject columns to keep track of error conversions.** (Tasks included in these steps are the one labeled: "DCNV - Convert *ColumnName* into *datatype"* and "DER – *ColumnName* Rejects")

- *CallTimeFee* (corresponding to the Call Charges in step 9) was converted to currency.
- *CallDuration* was converted to numeric.
- *WaitTime* was converted to numeric.
- *TotalFees* (a column we chose to add- see step 21) was converted to numeric.

For each of these columns, we configure tasks to output customized messages in case of data conversion error. The messages of these rejects' columns are based on the same model as the rejects column configured on step 8 of the Call Charges table. As a recall, these columns will store the date of the error occurrence, the package and task affected by the error, the column name, and a customized error message.

STEP 18 -19: Appending all messages triggered by data conversion error and store them in the ADM database on the Technical Rejects table.

STEP 20-22: Adding columns to fit the (eventual) business analysis need.

- **STEP 20:** In the data dictionary, *CallAbandoned* column is defined as having only two outcomes possible: the client abandoned the call, represented by value 1, or he did not, represented by value 0. It is a Boolean type. So, instead of having these unexplicit values, we create a new column with CallAbandoned as a Boolean column so "0" will now be "false" and "1" will be "true".

DER - Add column CallAbandoned _Bool

Derived Column Name	Derived Column	Expression	Data Type
CallAbandoned_Bool	<add as new column>	(CallAbandoned == "1") ? TRUE : FALSE	Boolean [DT_BOOL]

- **STEP 21:** We add a column TotalFees which is the amount the client will be charged for the entire call. Based on real life experience, we consider that the client starts to be charged at the first tone. Therefore, the call duration taken for calculation will include *WaitTime* and *CallDuration***.** We note that call charges are expressed per min and WaitTime and CallDuration are in minutes.

Derived Column Name	Derived Column	Expression	Data Type
TotalFees	<add as new column>	(DT_NUMERIC,6,2)(CallDuration_num + WaitTime_num) * CallFees_perMin_currency / 60	numeric [DT_NUMERIC]

DER - Add TotalFees columns

- **STEP 22:** To answer the stakeholder's request, we add the conditional column *Service Level Agreement'* identify the calls that meet the requirement (i.e., *WaitTime* < 35 seconds) and those who do not.

Derived Column Name	Derived Column	Expression	Data Type	Length
Service_Level_Agreeme...	<add as new column>	WaitTime_num < 35 ? "Within SLA" : "Outside SLA"	Unicode string [DT_WSTR]	11

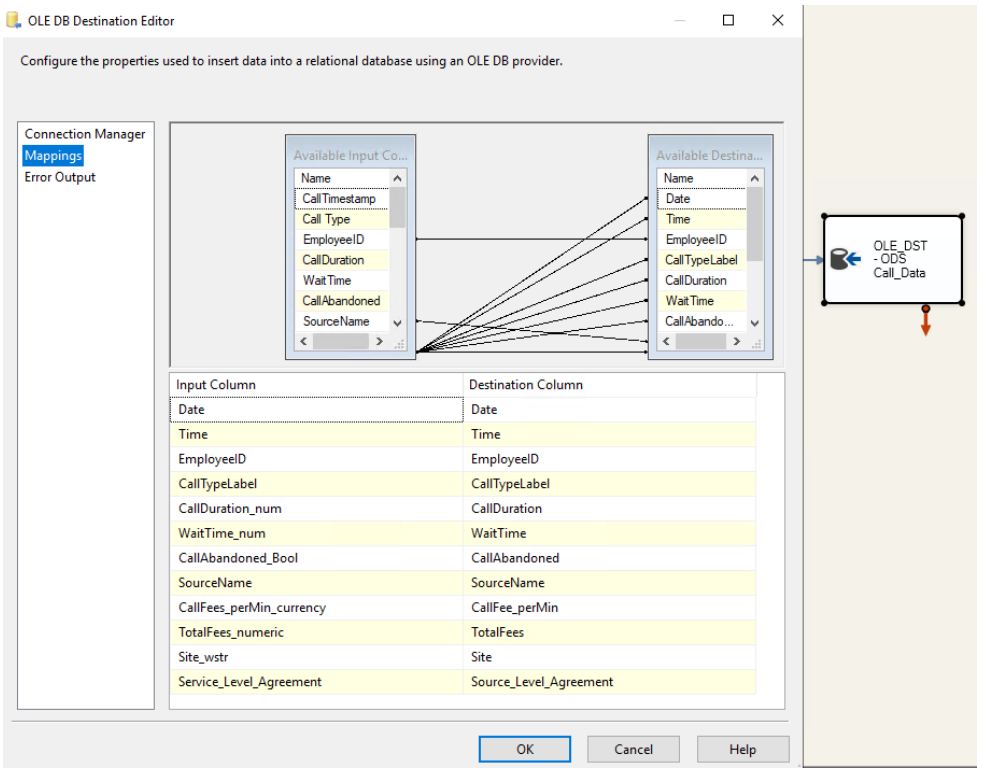
DER - Add Service Level Agreement column

STEP 23: Creating a Call_Data table in ODS database to store the column that we need. We map them to their destination in ODS database on Call_Data table.

Below is the SQL Query to create the Call_Data table in the ODS database.

```
USE [ETL_PROJECT_ODS]
GO

/****** Object:  Table [dbo].[Call_Data]    Script Date: 5/10/2024 2:30:11 AM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Call_Data](
    [Date] [date] NULL,
    [Time] [time](0) NULL,
    [Site] [nvarchar](50) NULL,
    [EmployeeID] [varchar](255) NULL,
    [CallTypeLabel] [varchar](255) NULL,
    [CallDuration] [numeric](18, 0) NULL,
    [WaitTime] [numeric](18, 0) NULL,
    [CallAbandoned] [bit] NULL,
    [CallFee_perMin] [money] NULL,
    [TotalFees] [numeric](18, 2) NULL,
    [Source_Level_Agreement] [nvarchar](11) NULL,
    [SourceName] [nvarchar](max) NULL
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
```

We map the columns accordingly.



Here is an extract of the Call Table in ODS database in SQL Server.

	Date	Time	Site	EmployeeID	CallTypeLabel	CallDuration	WaitTime	CallAbandoned	CallFee_perMin	TotalFees	Source_Level_Agreement	SourceName
1	2019-12-07	14:23:00	Aurora, CO	A475155	Billing	285	16	0	1.32	6.62	Within SLA	Data 2019.csv
2	2019-05-21	19:47:00	Spokane, WA	H438047	Billing	248	15	0	1.32	5.78	Within SLA	Data 2019.csv
3	2019-10-24	12:10:00	Aurora, CO	J419954	Tech Support	639	26	0	0.98	10.86	Within SLA	Data 2019.csv
4	2019-11-11	16:13:00	Spokane, WA	Y193775	Tech Support	273	293	0	0.98	9.24	Outside SLA	Data 2019.csv
5	2019-02-25	12:25:00	Spokane, WA	E778362	Tech Support	221	293	0	0.98	8.39	Outside SLA	Data 2019.csv
6	2019-03-17	18:34:00	Aurora, CO	Q921541	Sales	769	0	0	1.56	19.99	Within SLA	Data 2019.csv
7	2019-07-02	19:19:00	Spokane, WA	N772493	Tech Support	810	0	0	0.98	13.23	Within SLA	Data 2019.csv
8	2019-02-26	18:12:00	Jacksonville, FL	F542348	Tech Support	1150	0	0	0.98	18.78	Within SLA	Data 2019.csv
9	2019-09-27	18:51:00	Spokane, WA	D411745	Billing	669	13	0	1.32	15.00	Within SLA	Data 2019.csv
10	2019-12-09	15:41:00	Aurora, CO	P286634	Sales	322	14	0	1.56	8.73	Within SLA	Data 2019.csv
11	2019-11-25	13:34:00	Jacksonville, FL	V732706	Billing	1035	150	0	1.32	26.07	Outside SLA	Data 2019.csv
12	2019-01-14	13:58:00	Aurora, CO	S274120	Tech Support	88	7	0	0.98	1.55	Within SLA	Data 2019.csv
13	2019-08-08	08:02:00	Aurora, CO	B861430	Tech Support	542	0	0	0.98	8.85	Within SLA	Data 2019.csv

DWH STAGE

Datawarehousing

In the final phase of our journey through the data warehouse, we gather all our insights into the Data Warehouse, a crucial point where information is transformed into exploitable knowledge. Adopting the popular Star schema, our database architecture is built around a core entity: the "fact table", surrounded by "dimension tables". The fact table serves as a repository for our key data, the indisputable facts, while the dimension tables enrich our understanding with contextual details. Our database design is carefully adapted to exploit the power of this schema, so that our data speaks for itself and enables us to make smart decisions.

Our Database Design

For our data, we can consider that the important data are the call data records. Consequently, we choose to build the fact table with the "Call_Data" table. As far as dimensions are concerned, a common dimension is "Date". It allows us to describe dates using several aggregated temporal categories (year, month, quarter). The second dimension we have chosen is " Sites ". It is used to describe geographic data. We have also added an "Employee" dimension to describe the employee in a more detailed way.

DimDate Integration

This dimension provides a complete timeline and all the necessary details. As we don't need to use external data to build this dimension, we'll use a TSQL script to create it.

This is how we proceeded:

```sql
USE [ETL_PROJECT_DWH]
GO

/****** Object:  Table [dbo].[DimDate]    Script Date: 5/10/2024 2:42:30 AM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[DimDate](
    [DateKey] [int] NOT NULL,
    [Date] [date] NOT NULL,
    [Day] [tinyint] NOT NULL,
    [DaySuffix] [char](2) NOT NULL,
    [Weekday] [tinyint] NOT NULL,
    [WeekDayName] [varchar](10) NOT NULL,
    [WeekDayName_Short] [char](3) NOT NULL,
    [WeekDayName_FirstLetter] [char](1) NOT NULL,
    [DOWInMonth] [tinyint] NOT NULL,
    [DayOfYear] [smallint] NOT NULL,
    [WeekOfMonth] [tinyint] NOT NULL,
    [WeekOfYear] [tinyint] NOT NULL,
    [Month] [tinyint] NOT NULL,
    [MonthName] [varchar](10) NOT NULL,
    [MonthName_Short] [char](3) NOT NULL,
    [MonthName_FirstLetter] [char](1) NOT NULL,
    [Quarter] [tinyint] NOT NULL,
    [QuarterName] [varchar](6) NOT NULL,
    [Year] [int] NOT NULL,
    [MMYYYY] [char](6) NOT NULL,
    [MonthYear] [char](7) NOT NULL,
    [IsWeekend] [bit] NOT NULL,
PRIMARY KEY CLUSTERED
(
    [DateKey] 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
```

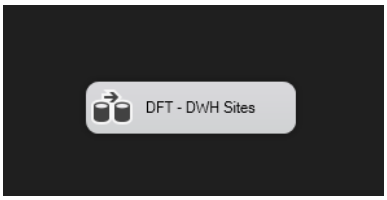
There are different date formats, from month to year, for more details.

Here is an extract from the opening lines:

	DateKey	Date	Day	DaySuffix	Weekday	WeekDayName	WeekDayName_Short	WeekDayName_FirstLetter	DOWInMonth	DayOfYear	WeekOfMonth	WeekOfYear	Month	MonthName	MonthName_Short	MonthName_FirstLetter	Quarter	QuarterName	Year	MMYYYY	MonthYear	IsWeekend
1	20180101	2018-01-01	1	st	2	Monday	MON	M	1	1	1	1	1	January	JAN	J	1	First	2018	012018	2018JAN	0
2	20180102	2018-01-02	2	nd	3	Tuesday	TUE	T	2	2	1	1	1	January	JAN	J	1	First	2018	012018	2018JAN	0
3	20180103	2018-01-03	3	rd	4	Wednesday	WED	W	3	3	1	1	1	January	JAN	J	1	First	2018	012018	2018JAN	0
4	20180104	2018-01-04	4	th	5	Thursday	THU	T	4	4	1	1	1	January	JAN	J	1	First	2018	012018	2018JAN	0
5	20180105	2018-01-05	5	th	6	Friday	FRI	F	5	5	1	1	1	January	JAN	J	1	First	2018	012018	2018JAN	0
6	20180106	2018-01-06	6	th	7	Saturday	SAT	S	6	6	1	1	1	January	JAN	J	1	First	2018	012018	2018JAN	1
7	20180107	2018-01-07	7	th	1	Sunday	SUN	S	7	7	2	2	1	January	JAN	J	1	First	2018	012018	2018JAN	1
8	20180108	2018-01-08	8	th	2	Monday	MON	M	8	8	2	2	1	January	JAN	J	1	First	2018	012018	2018JAN	0
9	20180109	2018-01-09	9	th	3	Tuesday	TUE	T	9	9	2	2	1	January	JAN	J	1	First	2018	012018	2018JAN	0
10	20180110	2018-01-10	10	th	4	Wednesday	WED	W	10	10	2	2	1	January	JAN	J	1	First	2018	012018	2018JAN	0

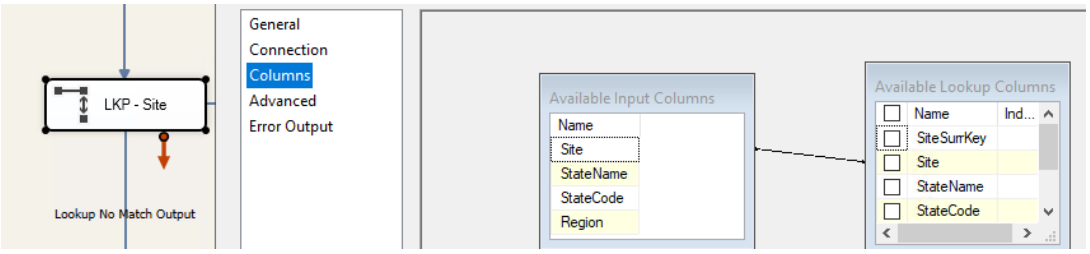
DimSites Integration

We have already cleaned the data in the previous stage.

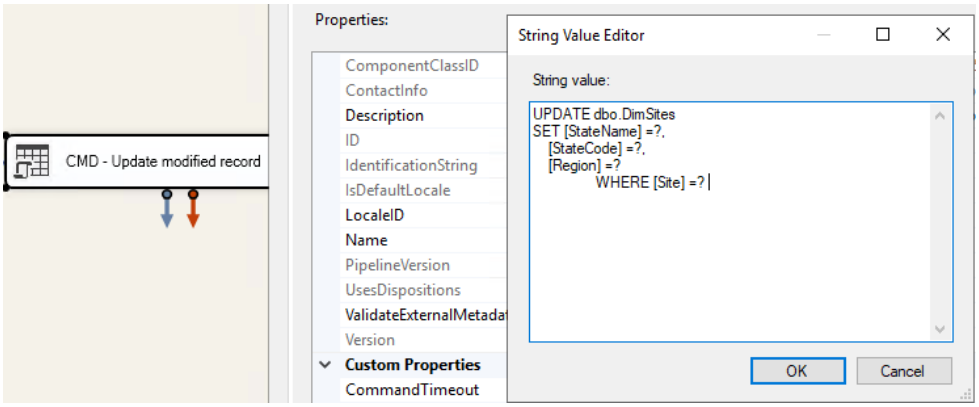


We loaded the data as follows:

We need to join the table of facts and the table of dimensions with “Site“



Next, we search to discover if there are any changes to existing sites. If so, the changes will be applied via CMD.



Input Column	Destination Column
StateName	Param_0
StateCode	Param_1
Region	Param_2
Site	Param_3

The first results lines are shown below:

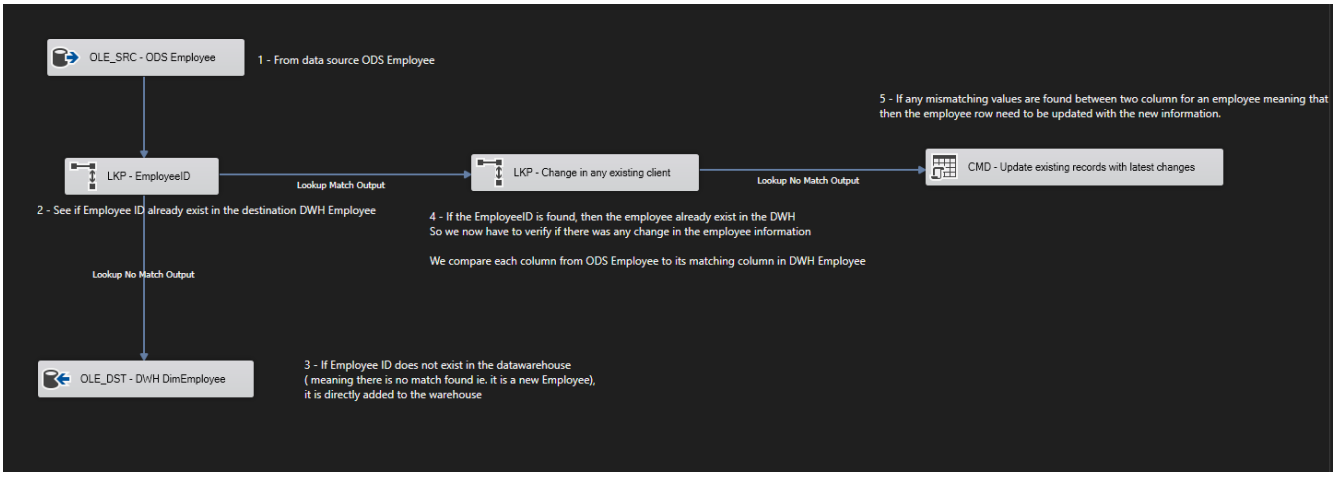
Results | Messages

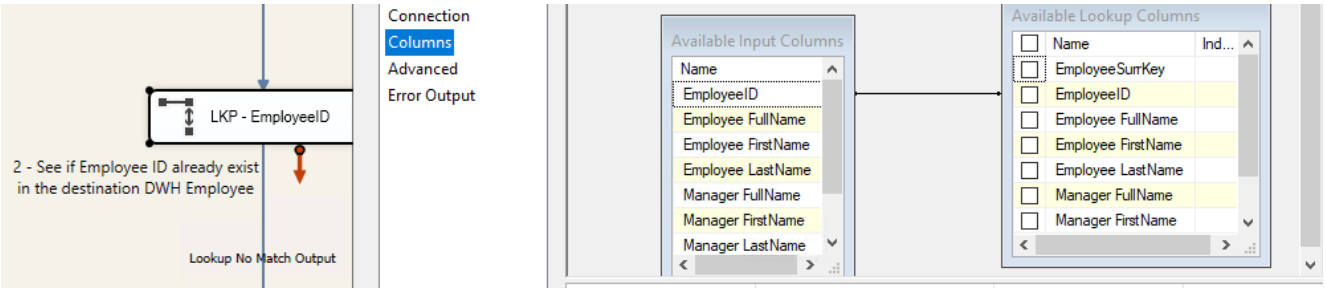
	SiteSurrKey	Site	StateName	StateCode	Region
1	1	Spokane, WA	Washington	WA	West
2	2	Jacksonville, FL	Florida	FL	South
3	3	Aurora, CO	Colorado	CO	West

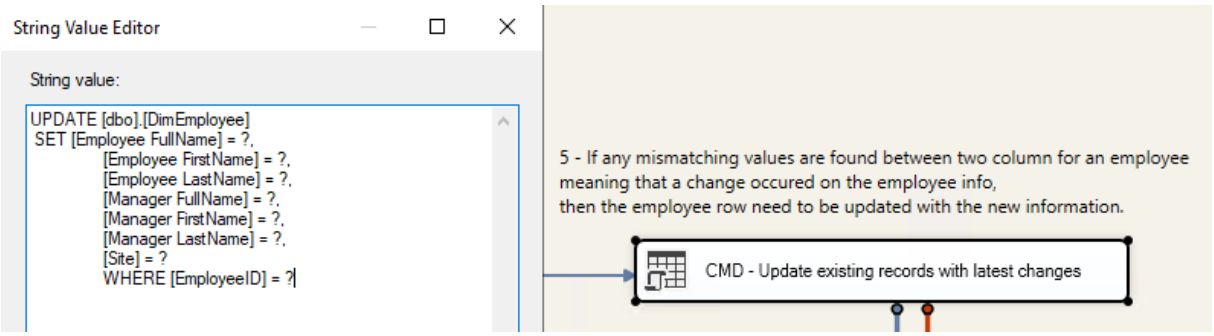
DimEmployee Integration

To have more details on employees, we decided to opt for a dimension table so that we could target calls according to the employee.

We proceeded in the same way as for the sites, using lookups to see if there were any new employees. If there are we'll add them directly to our dimensions and if there is any change on existing employee's information , we'll make the necessary updates.







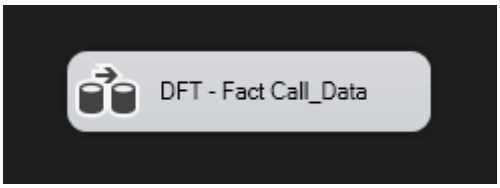
Input Column	Destination Column
Employee FullName	Param_0
Employee FirstName	Param_1
Employee LastName	Param_2
Manager FullName	Param_3
Manager FirstName	Param_4
Manager LastName	Param_5
Site	Param_6
EmployeeID	Param_7

The first results lines are shown below:

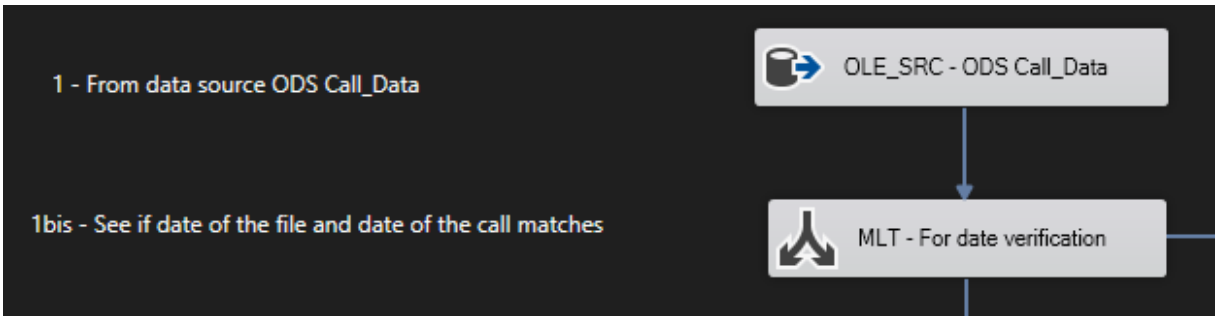
	EmployeeSurrKey	EmployeeID	Employee FullName	Employee FirstName	Employee LastName	Manager FullName	Manager FirstName	Manager LastName	Site
1	1	N772493	Onita Trojan	Onita	Trojan	Deidre Robbs	Deidre	Robbs	Spokane, WA
2	2	F533051	Stormy Seller	Stormy	Seller	Elsie Taplin	Elsie	Taplin	Aurora, CO
3	3	S564705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

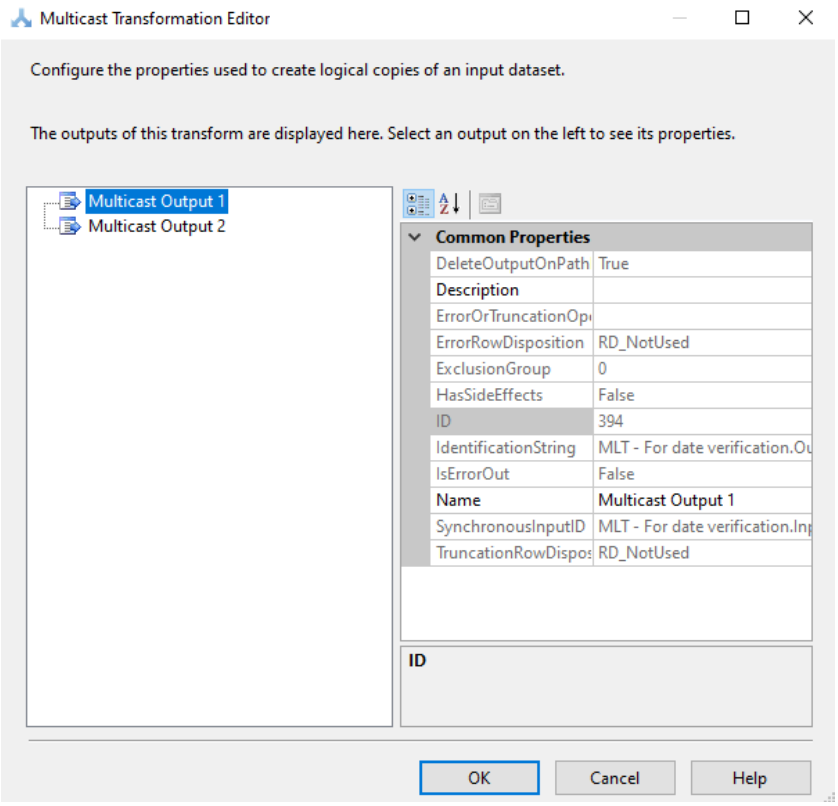
Fact Table Call Data Integration

We now have our dimension tables, so we can construct our fact table and check the validity of the relationships with the dimensions. We also have cleaned the data in the previous stage.

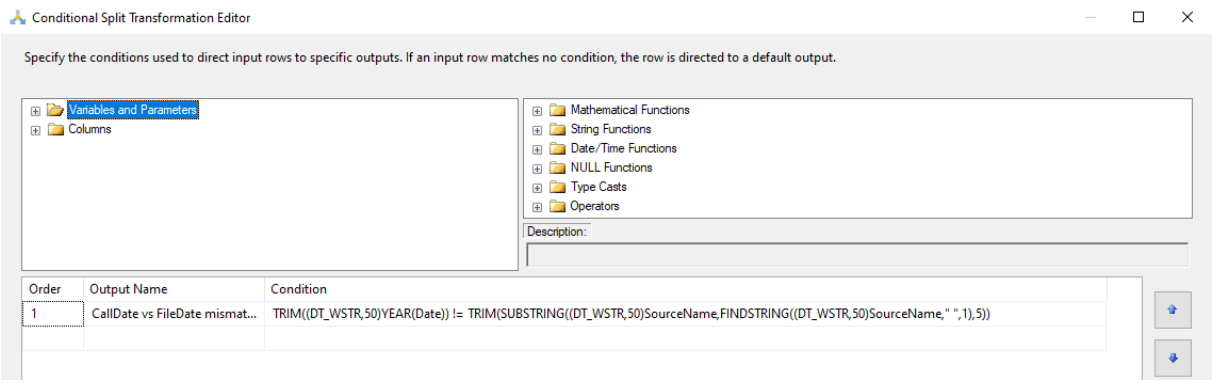


We have added a step to check that the dates of the files and the dates of the calls match. It allows us to be sure that this each call is registerd in the correct file.

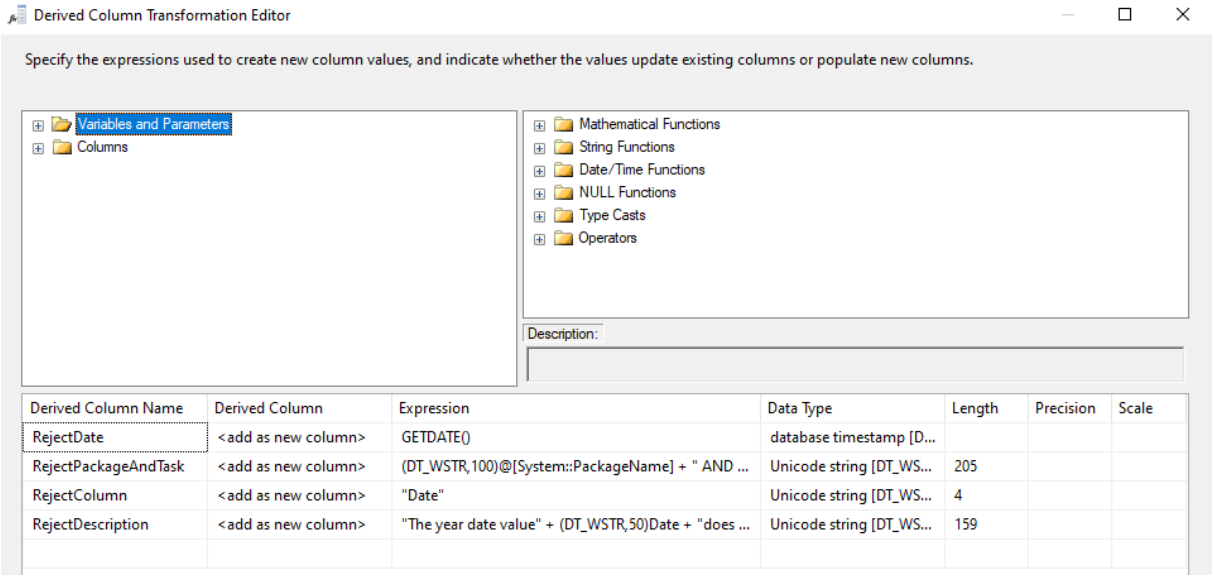


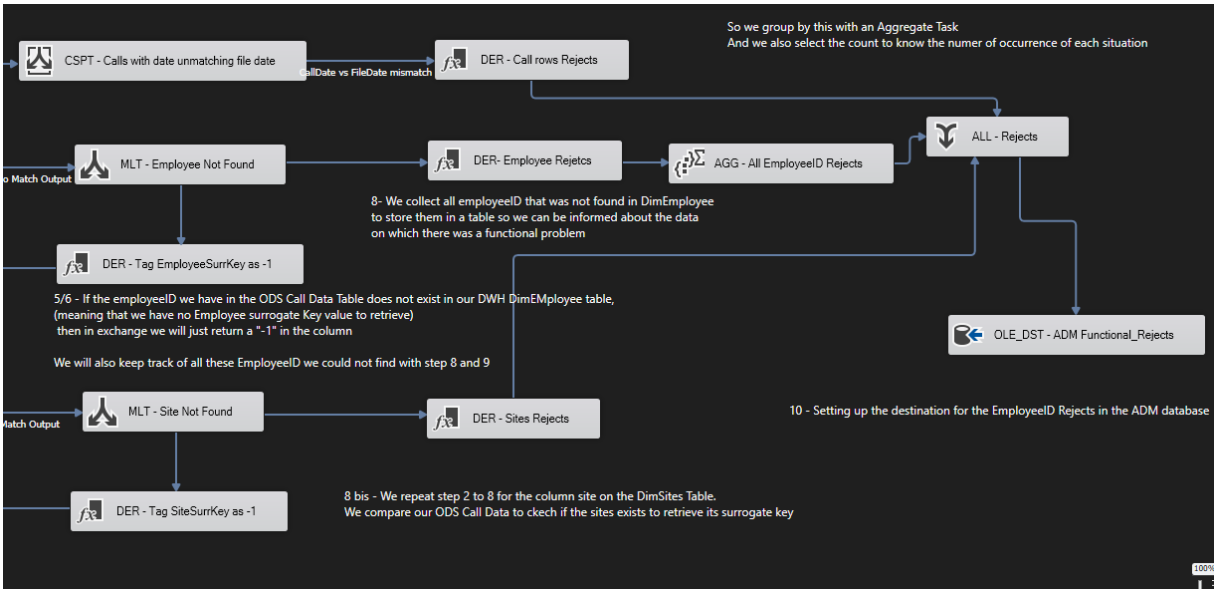


If we have unmatching date, we will proceed a transformation. On one of the outputs, we will separate the rows where the date is not matching from the other rows



Then we will redirect it to our Functional Rejects to keep track and look into the errors:





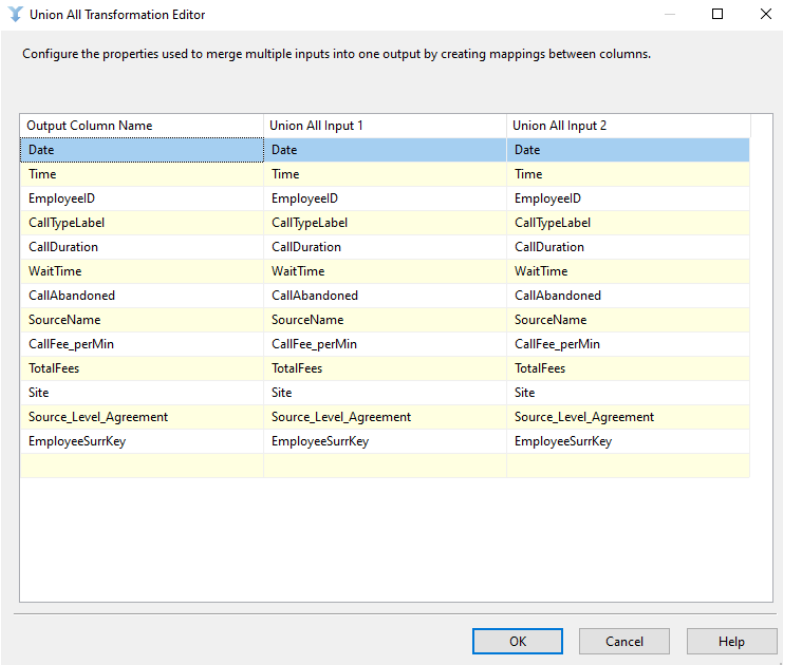
As you can see, we did the same procedure for both employee and Site: whenever we could find a match for EmployeeID/SiteID on DimEmployee table/ DimSites table, we redirect the information to our Rejects Database, but we also tag the missing value as -1 before redirecting it to the final destination database i.e. ETL_PROJECT_DWH.

Note that, we also decided to aggregate our EmployeeID rejects to avoid duplicates.

If the data match, then we proceed to retrieve the Employee surrogate key /Site surrogate key and there is unmatching data we proceed like this:

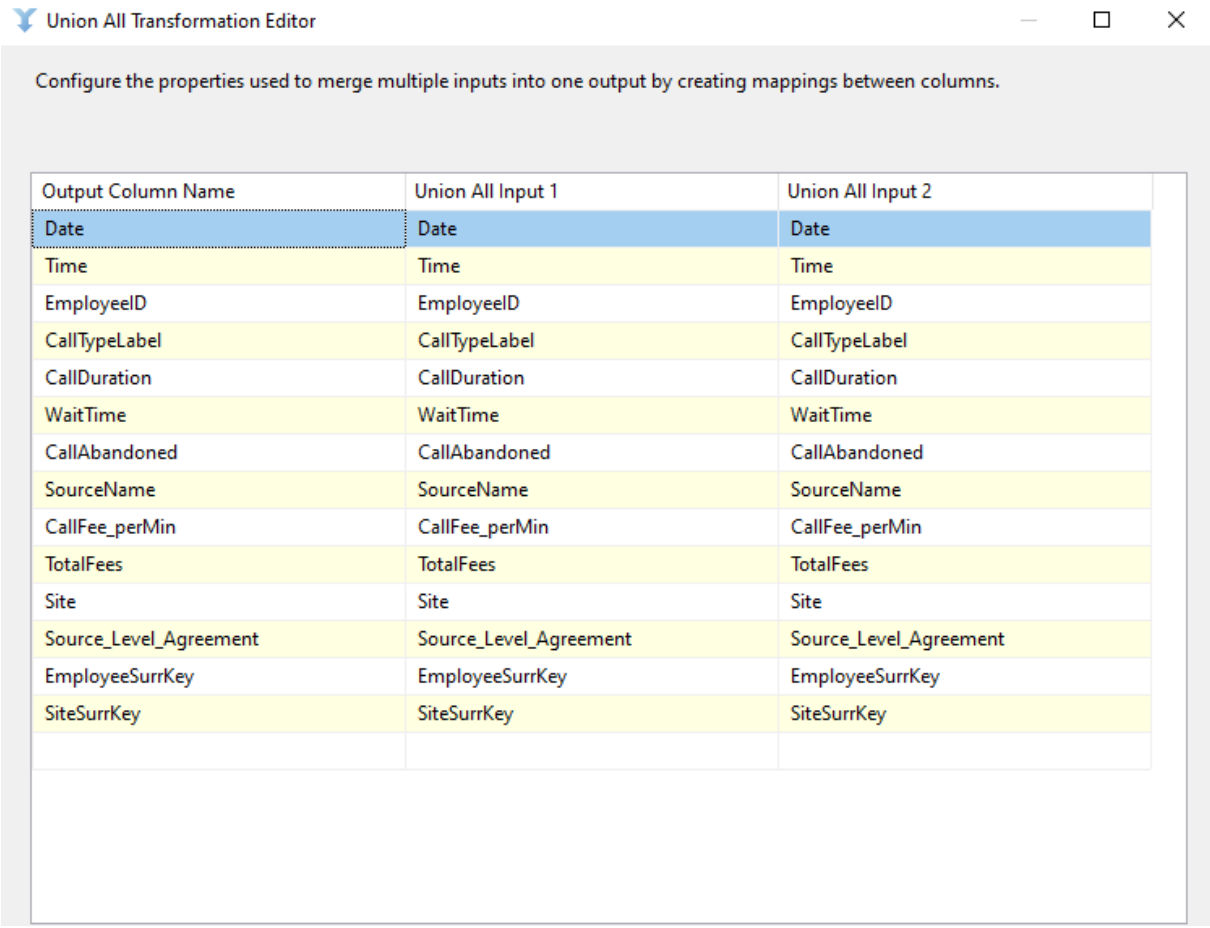
So, we have matching data, we are now going to compare our Dimtable with our actual FactTable. We add these steps also to manage surrogate key. We have chosen to go for SCD 1, so we will not have duplicate primary key.

The next step is to make a union of all the employee surrogate key: the one retrieved from the employee table in the ETL_PROJECT_DWH Warehouse and the one tagged as -1. The same is also done for Site. We make sure the column added to tag the missing value as -1, is merged with the surrogate Key values.



Output Column Name	Union All Input 1	Union All Input 2
Date	Date	Date
Time	Time	Time
EmployeeID	EmployeeID	EmployeeID
CallTypeLabel	CallTypeLabel	CallTypeLabel
CallDuration	CallDuration	CallDuration
WaitTime	WaitTime	WaitTime
CallAbandoned	CallAbandoned	CallAbandoned
SourceName	SourceName	SourceName
CallFee_perMin	CallFee_perMin	CallFee_perMin
TotalFees	TotalFees	TotalFees
Site	Site	Site
Source_Level_Agreement	Source_Level_Agreement	Source_Level_Agreement
EmployeeSurrKey	EmployeeSurrKey	EmployeeSurrKey

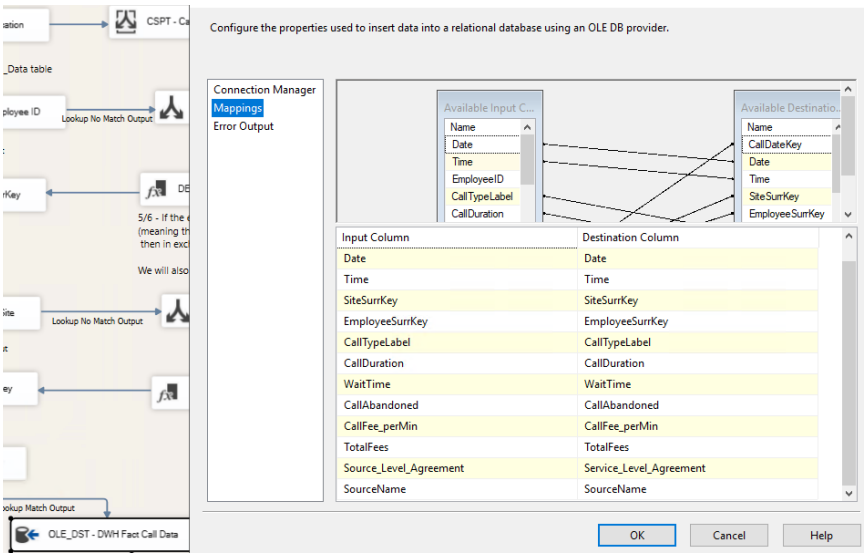
The next step is similar to the previous one we are going to compare our Dimsites table with our Fact table. To do that we think about matching the date column from our ODS Call_Date with our DimDate Table. In order to retrieve the Date Primary Key from our DimDate table and add the date primary key as a new column into our Fact Table



Output Column Name	Union All Input 1	Union All Input 2
Date	Date	Date
Time	Time	Time
EmployeeID	EmployeeID	EmployeeID
CallTypeLabel	CallTypeLabel	CallTypeLabel
CallDuration	CallDuration	CallDuration
WaitTime	WaitTime	WaitTime
CallAbandoned	CallAbandoned	CallAbandoned
SourceName	SourceName	SourceName
CallFee_perMin	CallFee_perMin	CallFee_perMin
TotalFees	TotalFees	TotalFees
Site	Site	Site
Source_Level_Agreement	Source_Level_Agreement	Source_Level_Agreement
EmployeeSurrKey	EmployeeSurrKey	EmployeeSurrKey
SiteSurrKey	SiteSurrKey	SiteSurrKey

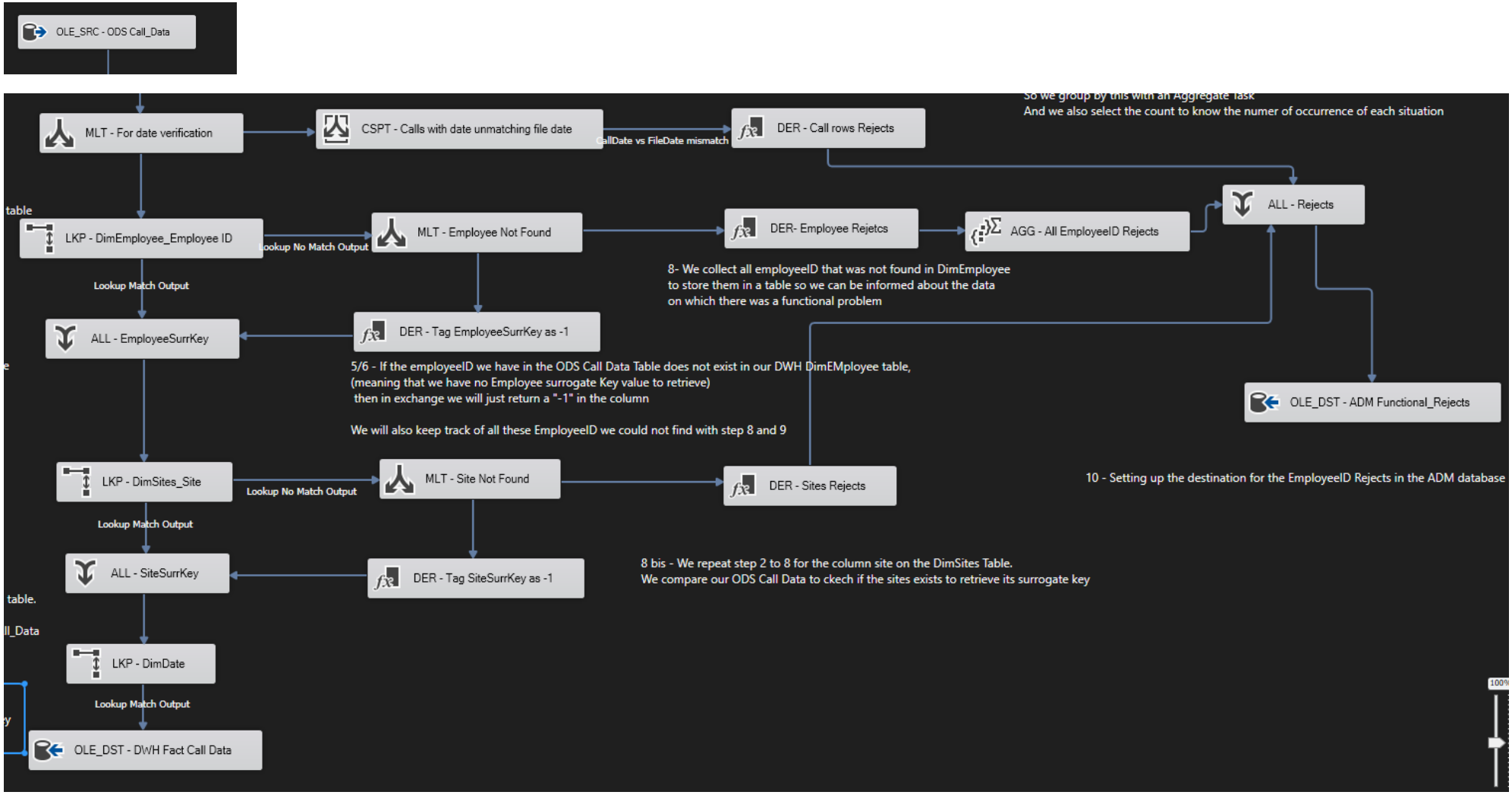
Last but not least, we create the destination table called Call_Data (our Fact Table).

```sql
USE [ETL_PROJECT_DWH]
GO

/****** Object:  Table [dbo].[FactCall_Data]    Script Date: 5/10/2024 2:40:17 AM ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[FactCall_Data](
    [CallDateKey] [int] NULL,
    [Date] [date] NULL,
    [Time] [time](0) NULL,
    [SiteSurrKey] [int] NULL,
    [EmployeeSurrKey] [int] NULL,
    [CallTypeLabel] [varchar](255) NULL,
    [CallDuration] [numeric](18, 0) NULL,
    [WaitTime] [numeric](18, 0) NULL,
    [CallAbandoned] [bit] NULL,
    [CallFee_perMin] [money] NULL,
    [TotalFees] [numeric](18, 2) NULL,
    [Service_Level_Agreement] [nvarchar](11) NULL,
    [SourceName] [nvarchar](max) NULL
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
```



After mapping the necessary column, you can see below an extract of the final fact table *'FactCall_Data'*

Results | Messages

	CallDateKey	Date	Time	SiteSurrKey	EmployeeSurrKey	CallTypeLabel	CallDuration	WaitTime	CallAbandoned	CallFee_perMin	TotalFees	Service_Level_Agreement	SourceName
1	20191207	2019-12-07	14:23:00	3	56	Billing	285	16	0	1.32	6.62	Within SLA	Data 2019.csv
2	20190521	2019-05-21	19:47:00	1	13	Billing	248	15	0	1.32	5.78	Within SLA	Data 2019.csv
3	20191024	2019-10-24	12:10:00	3	17	Tech Support	639	26	0	0.98	10.86	Within SLA	Data 2019.csv
4	20191111	2019-11-11	16:13:00	1	5	Tech Support	273	293	0	0.98	9.24	Outside SLA	Data 2019.csv
5	20190225	2019-02-25	12:25:00	1	62	Tech Support	221	293	0	0.98	8.39	Outside SLA	Data 2019.csv
6	20190317	2019-03-17	18:34:00	3	27	Sales	769	0	0	1.56	19.99	Within SLA	Data 2019.csv
7	20190702	2019-07-02	19:19:00	1	1	Tech Support	810	0	0	0.98	13.23	Within SLA	Data 2019.csv

Here is all the steps that we implemented in our fact table: