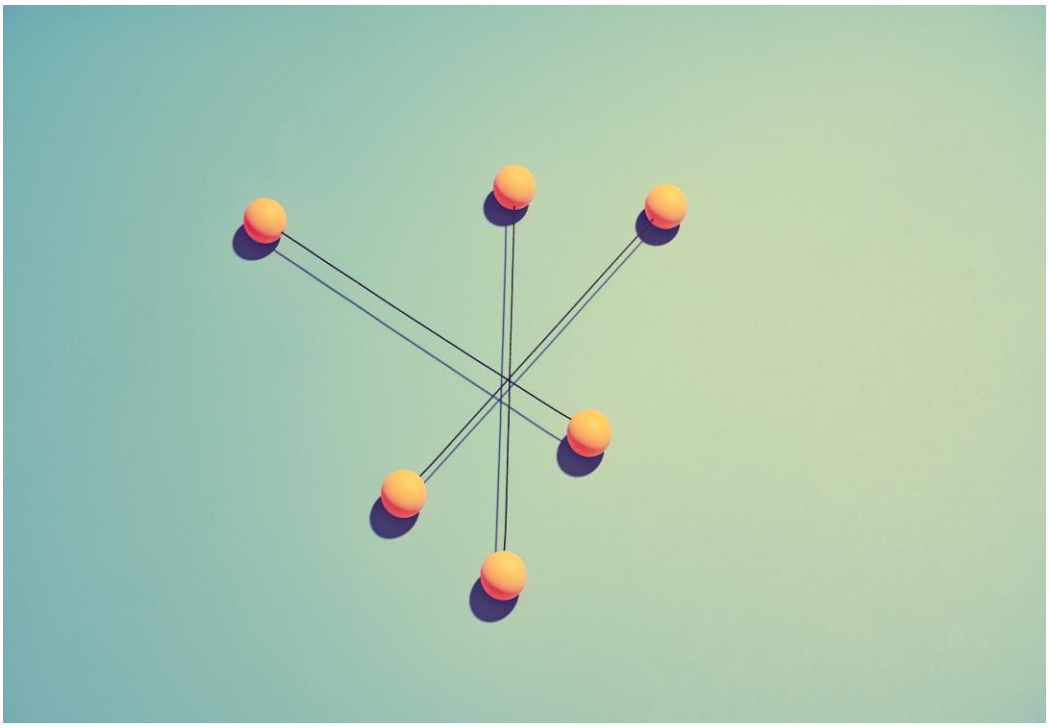
Data ScienceTech Institute

SSIS & SQL Server project report



Group members:

Trung Nguyen

Phuc Nguyen Pham

Mohammed Danish Mustafa

Patrick Warui Githendu

Paris, 19/05/2024

Table of Content

1. Introduction

To effectively utilize and extract value from available data, it must first be integrated into an IT system. This means unifying and standardizing data from various sources so it can be used by other programs. One way to accomplish this is by implementing a Data Warehouse.

In this project, we will design and implement a Data Warehouse using call data from ServiceSpot – an IT Company that has requested an analysis of their call center data. The task is to develop an Extract Transform & Load (ETL) project with SSIS that will load data into a new enterprise data warehouse.

1.1. Data

Our data is composed of 2 kinds of files: Call data (main data) and Lookup data:

CSV Files (Data YYYY)

These files contain call data, with one file per year:

- CallTimestamp: Date & time of the call
- Call Type: Type of call
- EmployeeID: Employee's unique ID
- CallDuration: Duration of the call, in seconds
- WaitTime: Wait time before the call was answered, in seconds
- CallAbandoned: Indicates if the call was abandoned by the customer (1 = Yes, 0 = No)

Lookup Data

These files provide additional information about employees and call types:

Table: Employees

- EmployeeID: Employee's unique ID
- EmployeeName: Employee's full name
- Site: Site where the employee works
- ManagerName: Employee's manager

Table: Call Types

- CallTypeID: Unique ID for the call type
- CallTypeLabel: Label for the call type

Table: US States

This file contains information about each state:

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

Table: Call Charges

This file contains the call charges per minute for each call type and year:

- Call Type Key: Unique ID for the call type
- Call Type: Call type label
- Call Charges / Min (YYYY): Amount charged to customers per minute for a specific year (YYYY)

By integrating and standardizing this data in a Data Warehouse using SQL Server and SSIS, we can perform comprehensive analyses and generate valuable insights for call center operations.

1.2. Pipeline design

The pipeline will be implemented in three steps:

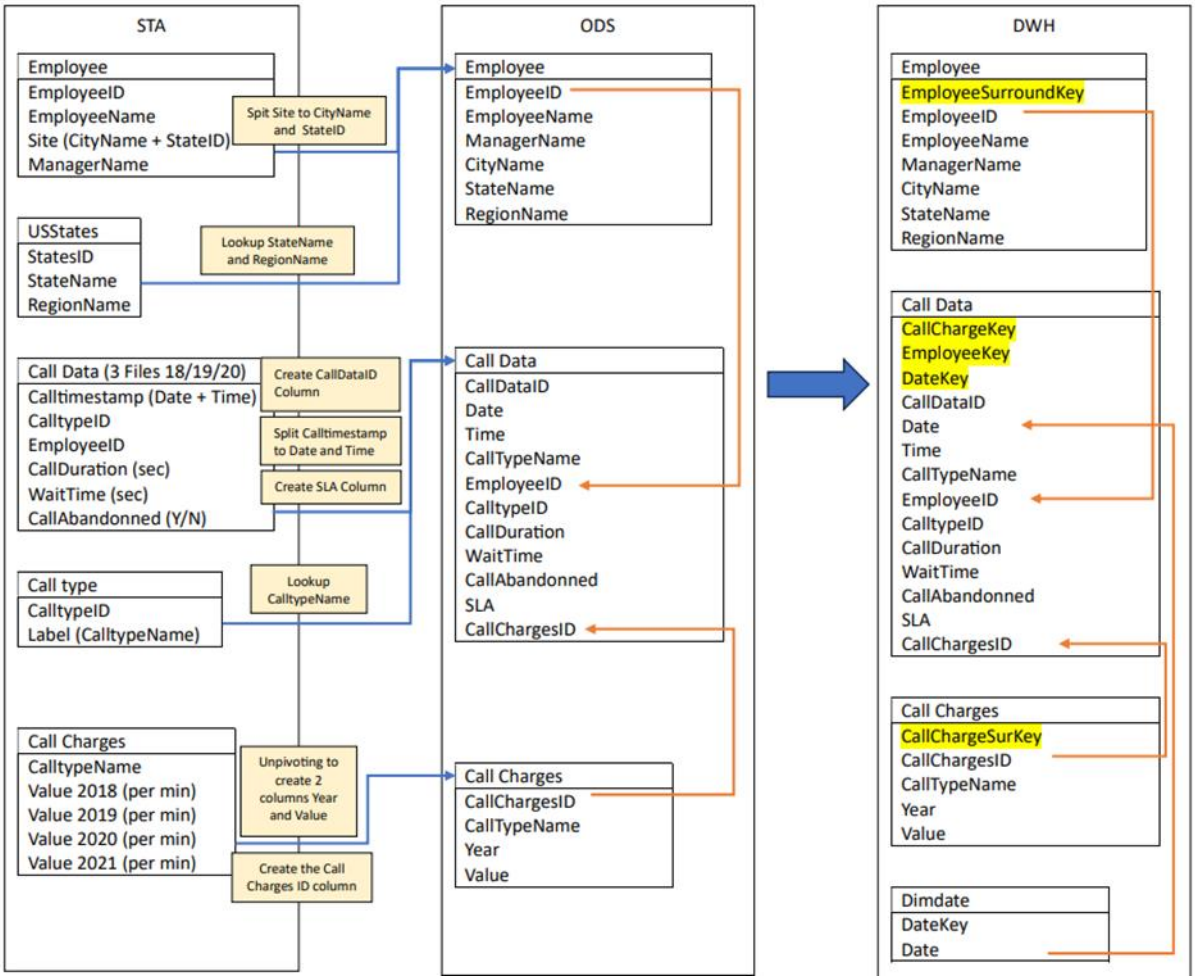
Staging Area (STA): This stage involves loading the data as is or with minimal changes.

Operational Data Store (ODS): This stage involves cleaning and standardizing the data. If the data does not meet quality criteria, it will be placed in the "Technical_Rejects" table as technical rejects.

Data Warehouse (DWH): This stage involves organizing the data into one fact table related to multiple dimension tables. If there is any change in the data, it would be updated on DWH.

1.3. Model

Based on our analysis, the model for design the data warehouse for the requirement of our client as below:



2. Staging

The **Staging Area** is the initial landing zone for raw data from various sources before it undergoes cleaning and transformation. This is the **Extract** step in the ETL process, where data from the CSV files is loaded in its original format.

A database is created as well as the following tables representing the data dictionary given.

a) Create Database A_23_Call_Types_STA
b) Create tables the MSQL queries are below

A decision is made to make most of the values as VARCHAR to accept raw data except those already clear from the datasets provided some of which are INT, CHAR, DATETIME and BIT.

The following Five (5) .dtsx files packages were created in Visual Studio SSIS.

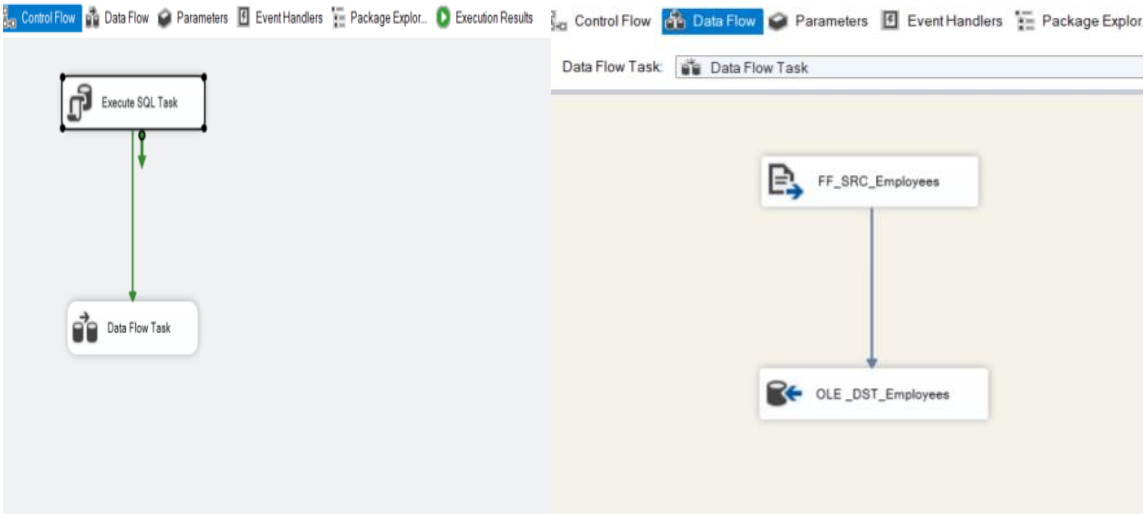
STA_CallData.dtsx, STA_CallCharges.dtsx, STA_CallTypes.dtsx, STA_Employees.dtsx and STA_USStates.dtsx.

2.1 Employees Table

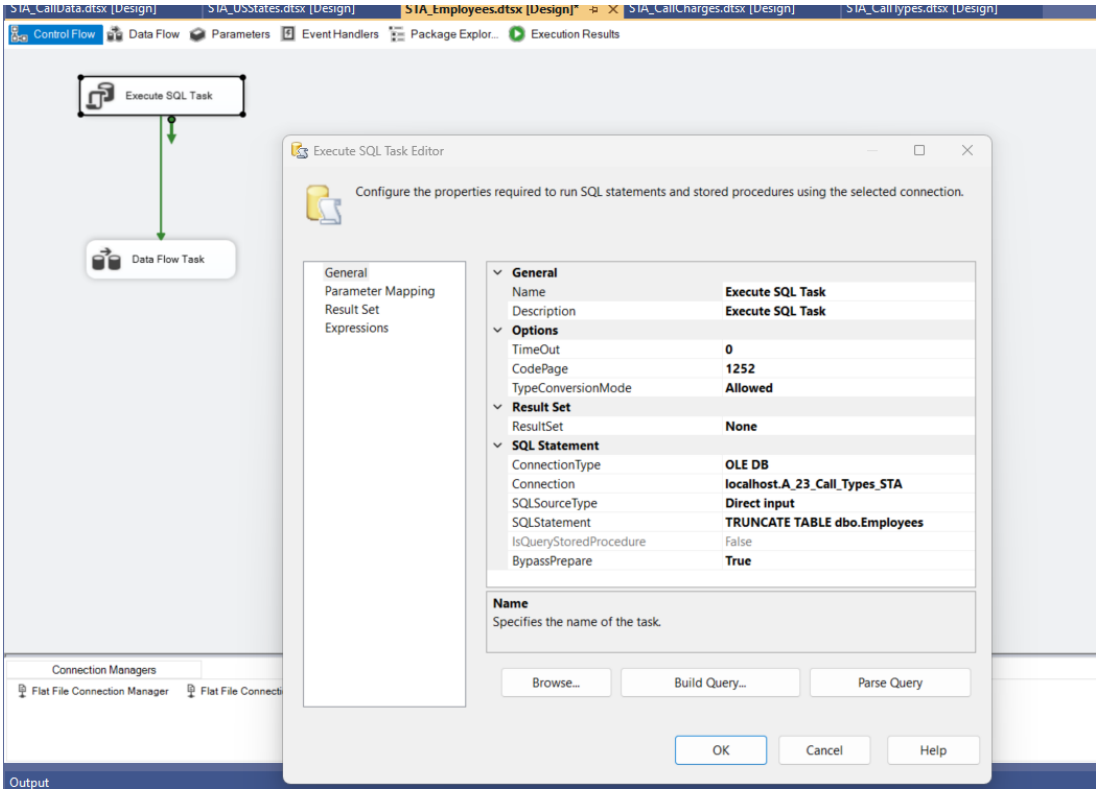
The Employees table is created as follows

```
CREATE TABLE Employees (
    EmployeeID VARCHAR(255),
    EmployeeName VARCHAR(255),
    Site VARCHAR(255),
    ManagerName VARCHAR(255)
);
```

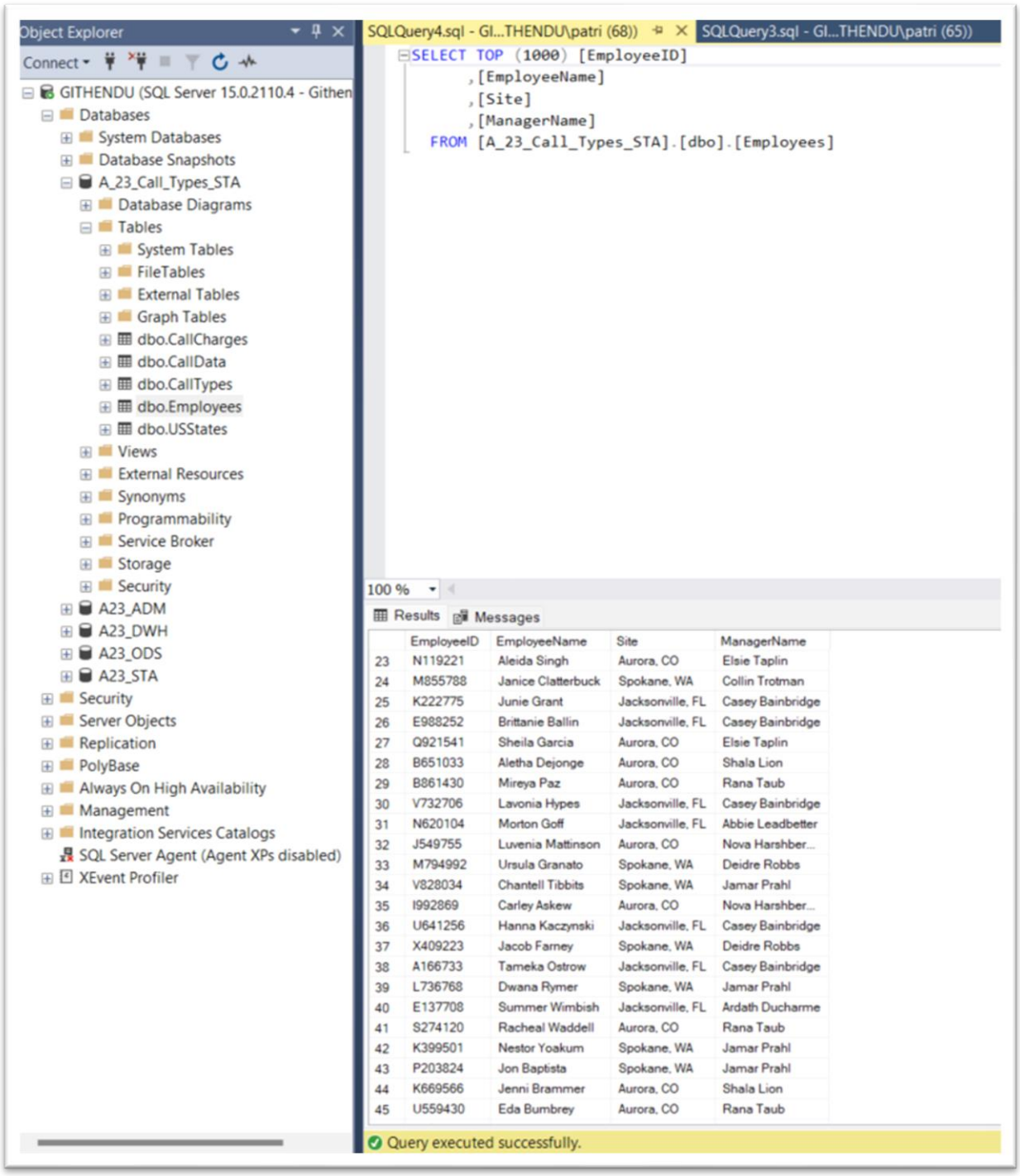
The data is then loaded into the database using the following flow



It is truncated in execute SQL Task to avoid duplication of data when loading other incoming data. **All other .dtsx files look similar**. We will show the Database tables that result from the process in the subsequent pages.



The Employees Database looks as follows



2.2 Call Data Table

The call Data Table is created as follows.

```
CREATE TABLE CallData (
    CallTimestamp DATETIME,
    CallType VARCHAR(255),
    EmployeeID VARCHAR(255),
    CallDuration INT,
    WaitTime INT,
```

```
    CallAbandoned BIT
);
```

In order to load the Calls Data, which is in 3 files as seen below, we use the Foreach Loop Container.

Name	Status	Date modified	Type	Size
Data 2018		4/11/2024 1:41 PM	Microsoft Excel Comma Separated Values File	1,149 KB
Data 2019		4/11/2024 1:41 PM	Microsoft Excel Comma Separated Values File	1,147 KB
Data 2020		4/11/2024 1:41 PM	Microsoft Excel Comma Separated Values File	1,145 KB

e.g. Foreach Loop Container -> Edit -> Collection-> choose Foreach File Enumerator and go to file folder with the files *.* is changed to *csv as we want to import only csv files 2018, 2019 and 2020 data. The Flat File Connection Manager is made dynamic to enable loading all 3 files.

Steps:

1. Add the Foreach Loop Container

Drag and drop a "Foreach Loop Container" into the Control Flow area.

2. Configure the Enumerator for Fully Qualified Filenames

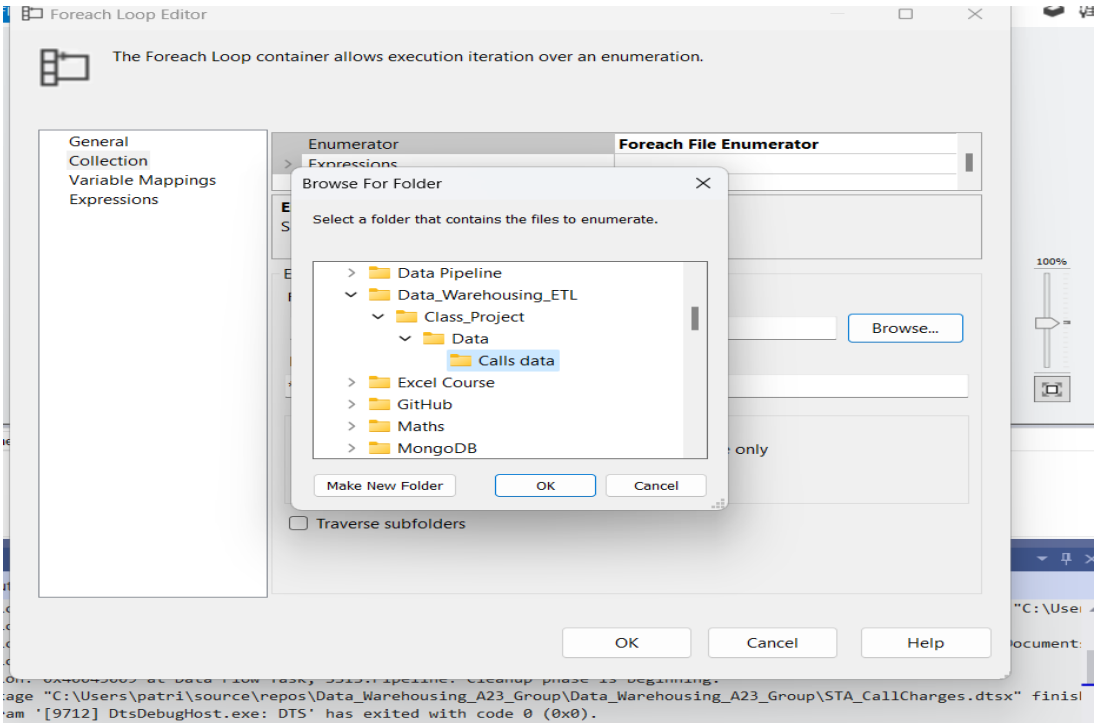
Double-click the Foreach Loop Container to open the editor.

In the Collection pane, choose the "Foreach File Enumerator" as the Enumerator.

Configure the Enumerator properties:

Set the folder to the directory containing the files.

Set the Files property to use *.csv for the 3 files.



3. Map the Variable

Go to the "Variable Mappings" pane.

Select your variable User::Filename from the list.

Set the Index to 0, assuming the Foreach Loop is set to enumerate file names which should be stored in the User::Filename variable during each iteration.

4. Configure the Expression

In the Foreach Loop Editor, switch to the "Expressions" pane.

Click the ellipsis (...) button to open the Property Expressions Editor.

In the Property column, select "Directory" or a relevant property that needs to use the User::Filename.

In the Expression column, set the expression to use your variable, like @[User::Filename].

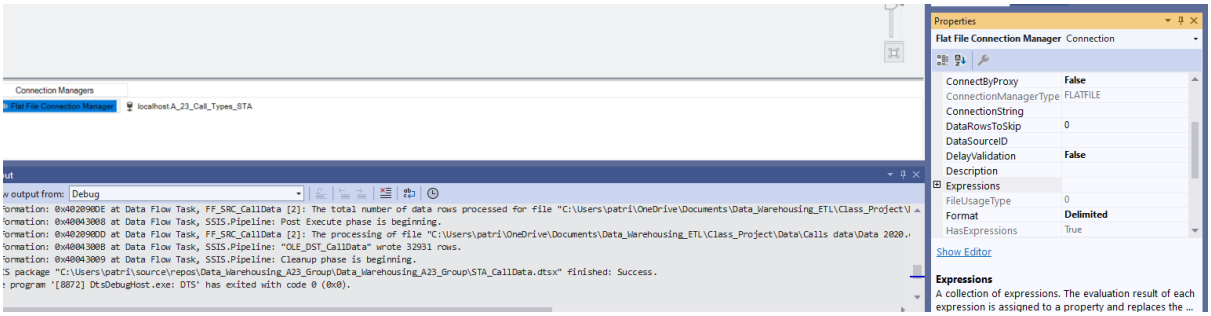
Click "OK" to close the editor.

5. Configure Tasks Inside the Container

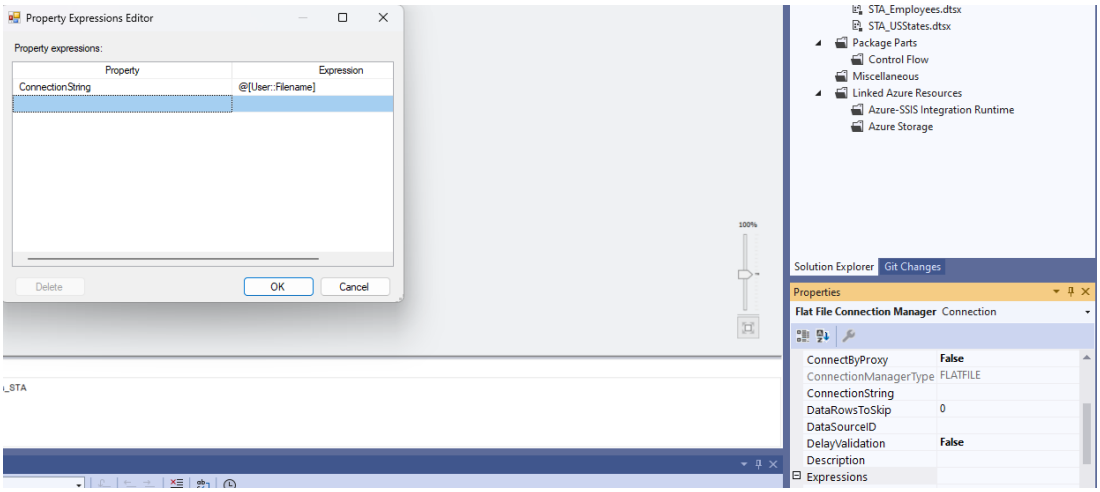
Inside the Foreach Loop Container, a Data Flow Task is dropped.

Set up these tasks to use the User::Filename variable where necessary, such as a file connection's ConnectionString property.

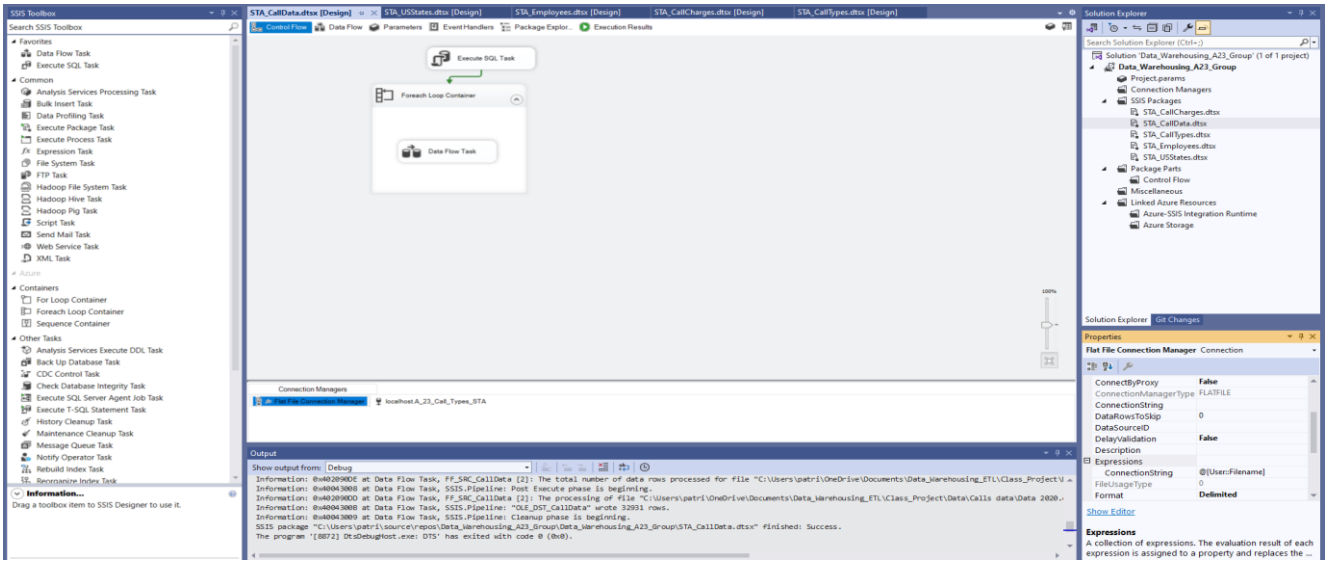
`Flat File Connection Manager -> Expression`



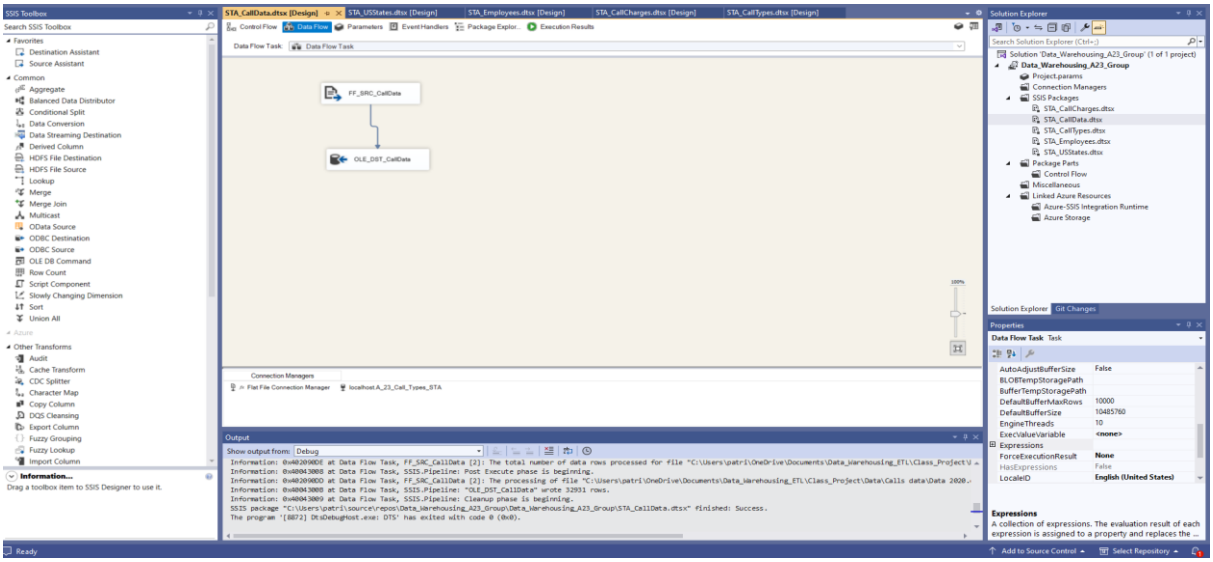
`Then -> Expression`



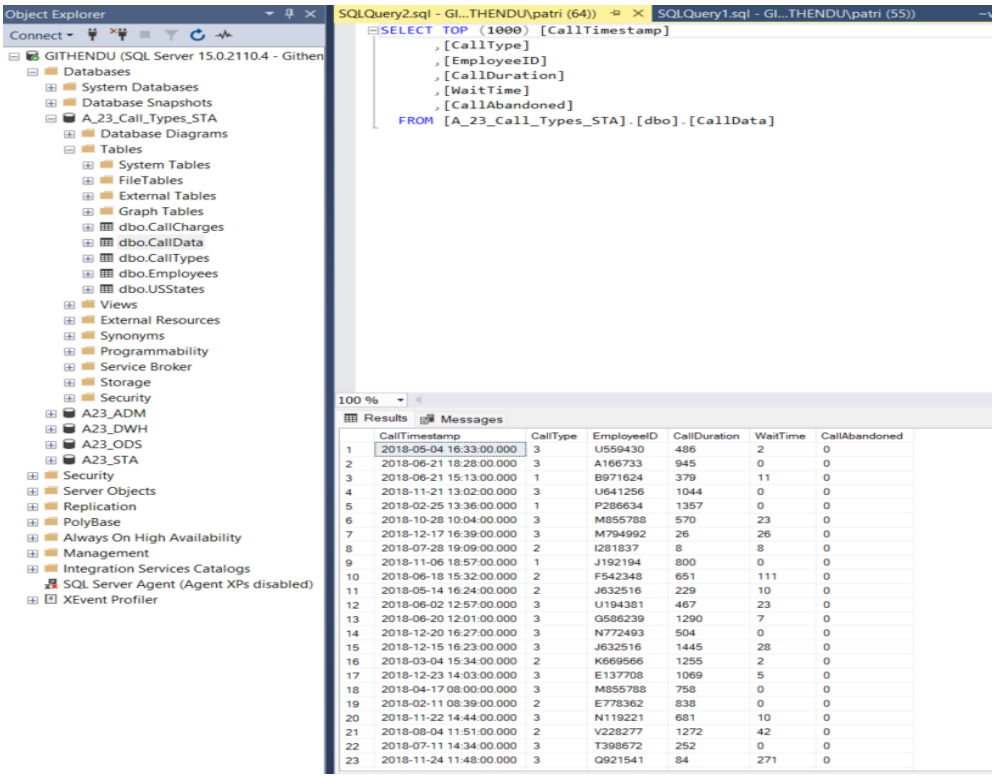
This is the STA_CallData.dtsx. control flow and is truncated to erase previous runs and ensure each time it runs, it inputs new data and avoids duplication.



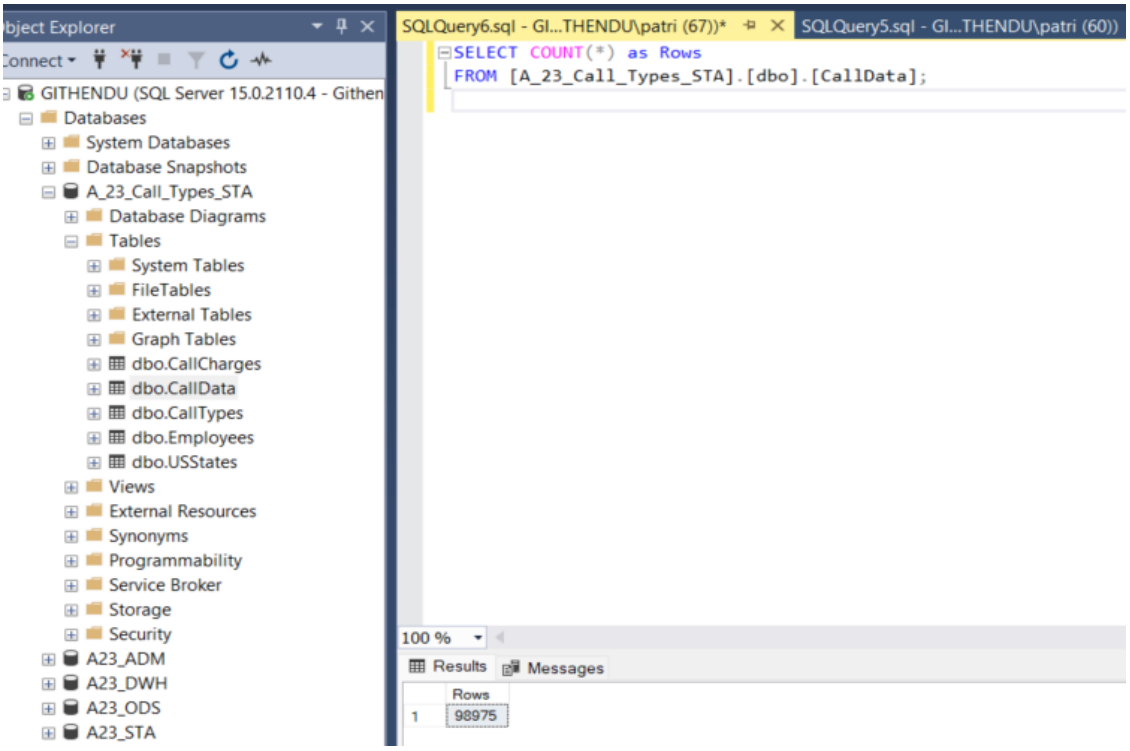
This is the STA_CallData.dtsx. data flow



The Database looks as follows in the Call Data Table after populating it



Only 98975 rows are loaded from the 3 files and there are no duplications since it is truncated as discussed at ForEach Loop Container section.

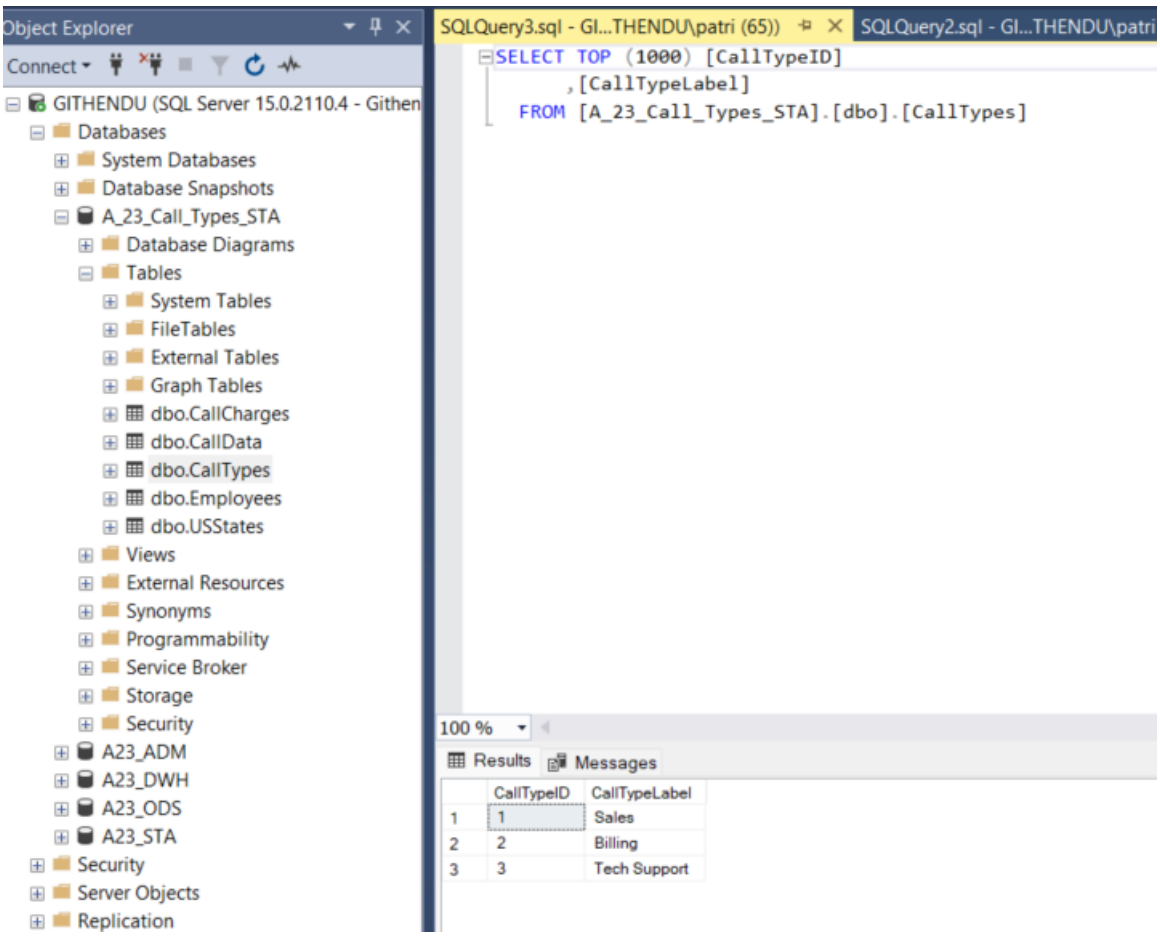


2.3 Call Types Table

The CallTypes Table is created as follows

```
CREATE TABLE CallTypes (
    CallTypeID INT,
    CallTypeLabel VARCHAR(255)
);
```

The Database table looks as follows.

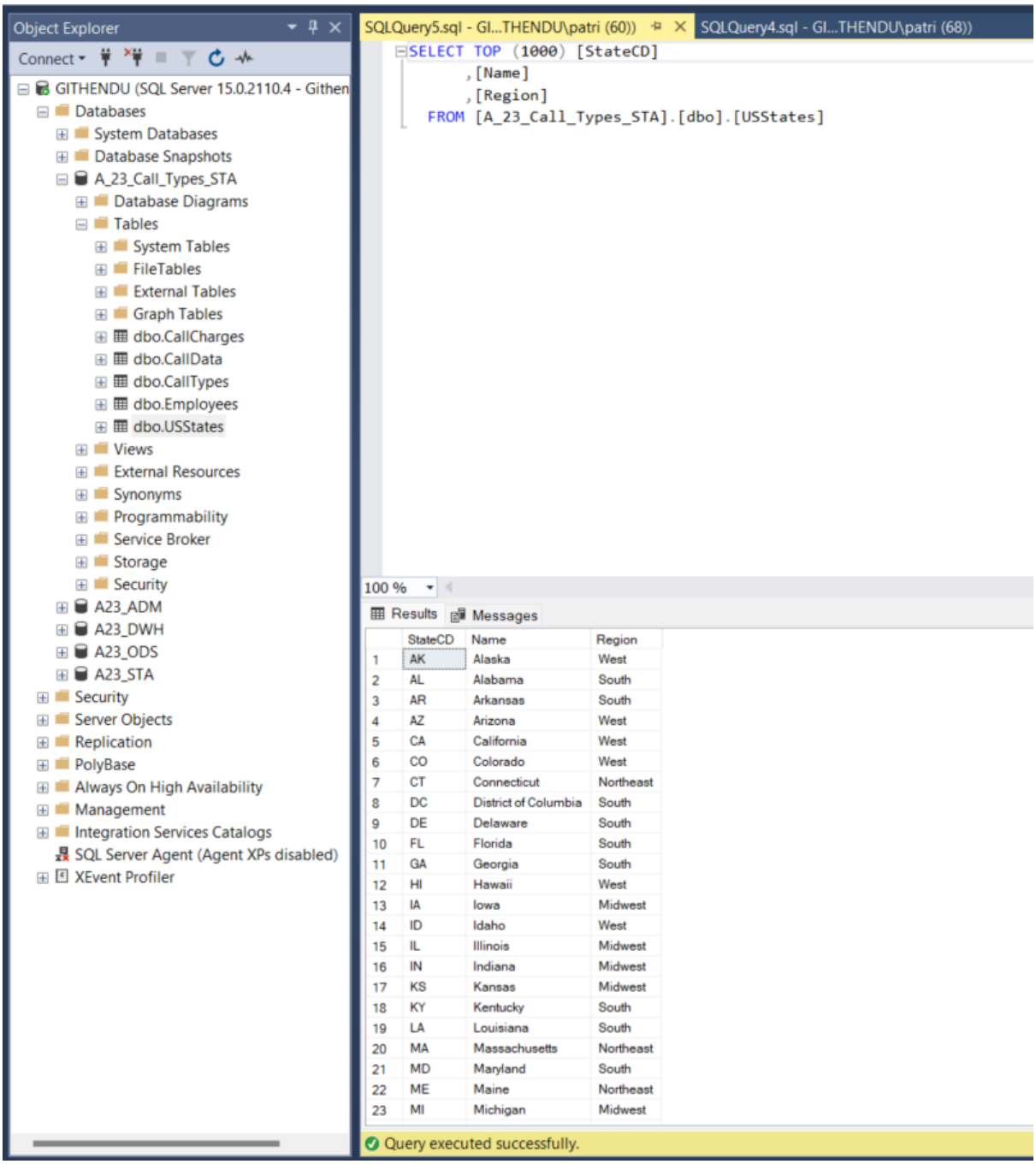


2.4 US States Table

The US States table is created as follows

```
CREATE TABLE USStates (
    StateCD VARCHAR(255),
    Name VARCHAR(255),
    Region VARCHAR(255)
);
```

The following is the US States table after loading the data

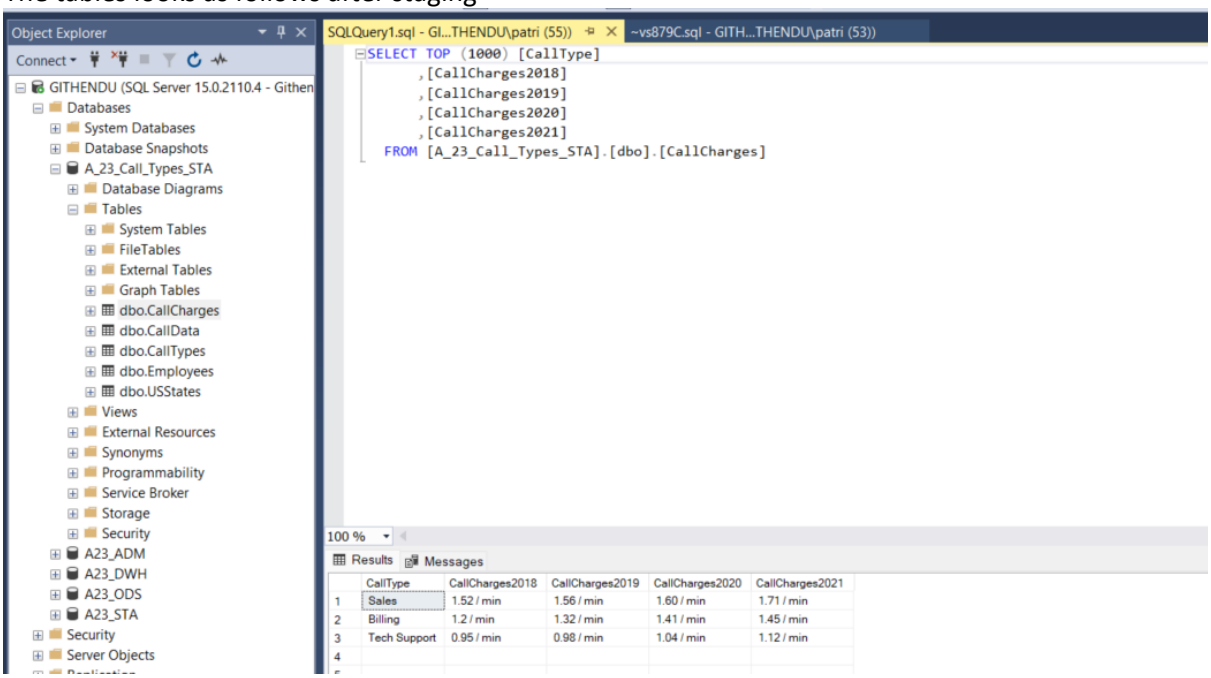


2.5 Call Charges Table

The Call Charges Table is created as follows

```
CREATE TABLE CallCharges (
    CallType VARCHAR(255),
    CallCharges2018 VARCHAR(255),
    CallCharges2019 VARCHAR(255),
    CallCharges2020 VARCHAR(255),
    CallCharges2021 VARCHAR(255)
);
```

The tables looks as follows after staging



The above are the 5 dbo. Tables in the Database A_23_Call_Types_STA

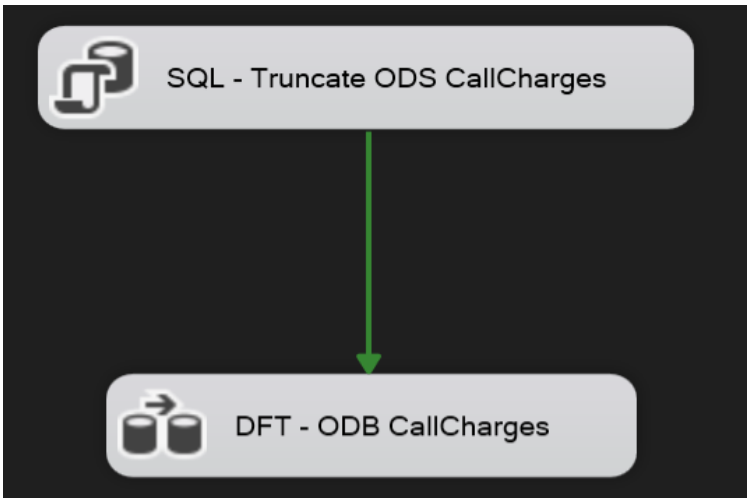
3. Operational Data Store

The second stage is The Operational Data Store (ODS). It serves as an intermediate staging area for data before it is moved to a data warehouse, ensuring data quality, consistency, and accessibility for operational reporting and decision-making.

In this stage, we will clean and standardize the data to ensure smooth data flow even when we get new data the future. Any "non-standard" data or data with an inappropriate type will be directed to a separate database (ADM database) for further analysis and handling. Not only that, new tables or columns will be created or merged which aims to create a "star" schema database model, which facilitates efficient querying in the future. Additionally, we also alter data types and lengths help standardized and streamlined outputs that are good for later analysis. This restructuring stage enhances the database's usability, allowing for more efficient data manipulation and interpretation.

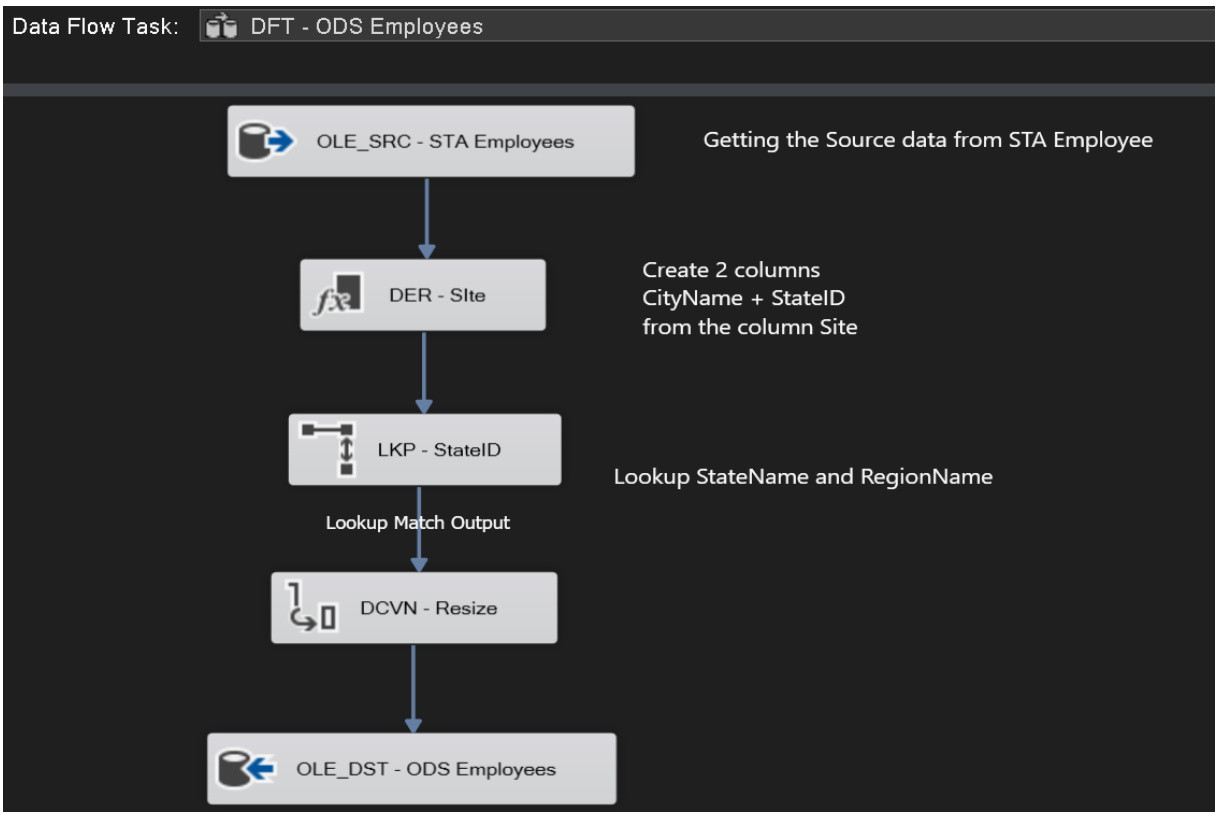
3.1. Employees Table

First of all, we truncate the table to avoid any duplication each time we run the process.



Next, this is the data flow task to make the Employee table in this ODS stage, the process will comprise five steps:

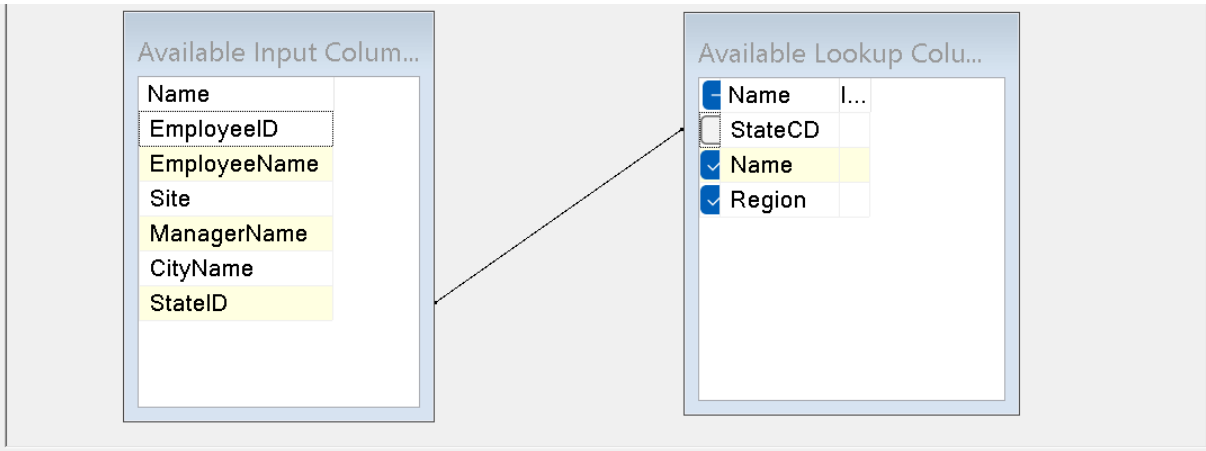
1. Import data from the STA Employees table.
2. Split the values in the Site column into two separate values to create the CityName and StateID columns.
3. Perform a lookup with the STA USStates table to add the StateName and RegionName columns.
4. Resize the length of the values in the columns to ensure consistency and standardization.
5. Load the data into the server and create the ODS Employees table.



In the second step, we found out that the Site column contains the information of the city name and the StateID, so we decided to split this column into 2 column CityName and StateID. We do it by using the Deriver column in SSIS toolbox. Here is the expression for each column:

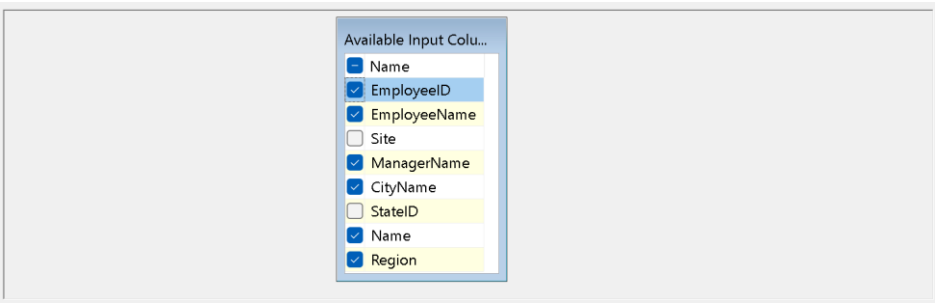
Derived Column Name	Derived Column	Expression	Data Type	Length
CityName	<add as new column>	(DT_STR,100,1252)TRIM(SUBSTRING(Site,1,FINDSTRING(Site + ",",",",1) - 1))	string [DT_STR]	100
StateID	<add as new column>	(DT_STR,255,1252)(RIGHT(Site,2))	string [DT_STR]	255

Then, we do the lookup with the STA USStates table by connecting the StateID of each table



Lookup Column	Lookup Operation	Output Alias
Name	<add as new column>	Name
Region	<add as new column>	Region

Before going to the last step, we resize the length of the column, which helps to save the memory and accelerate the loading time in the long run.



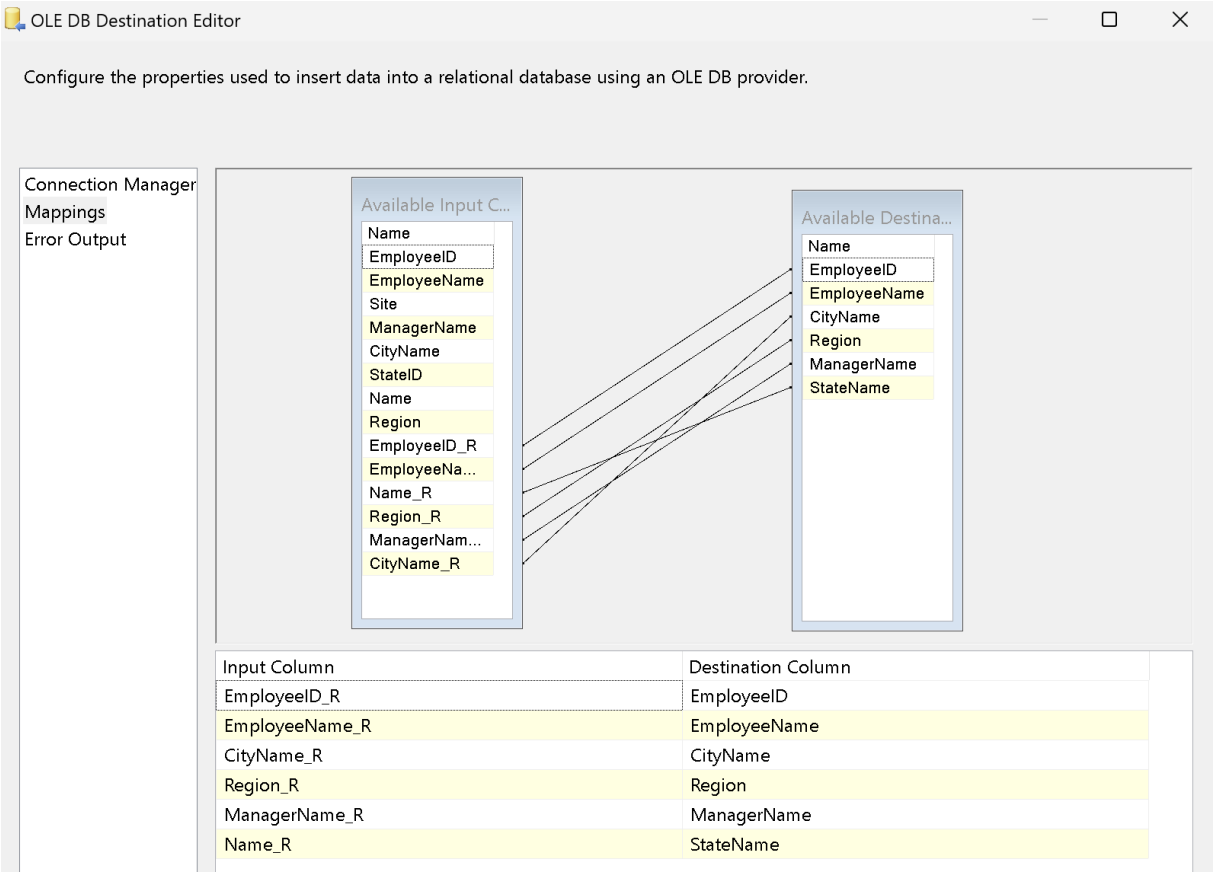
Input Column	Output Alias	Data Type	Length	Precision	Scale	Code Page
EmployeeID	EmployeeID_R	string [DT_STR]	20			1252 (ANSI - Latin I)
EmployeeName	EmployeeName_R	string [DT_STR]	100			1252 (ANSI - Latin I)
Name	Name_R	string [DT_STR]	50			1252 (ANSI - Latin I)
Region	Region_R	string [DT_STR]	20			1252 (ANSI - Latin I)
ManagerName	ManagerName_R	string [DT_STR]	100			1252 (ANSI - Latin I)
CityName	CityName_R	string [DT_STR]	50			1252 (ANSI - Latin I)

And finally, we created a DB destination to put the data into the ODS database. We create the Employees table with a following script:

```
CREATE TABLE Employees (
    [EmployeeID] varchar(20),
    [EmployeeName] varchar(100),
    [CityName] varchar(50),
    [StateName] varchar(50),
```

```
    [Region] varchar(20),
    [ManagerName] varchar(100)
)
```

And here is the mapping:

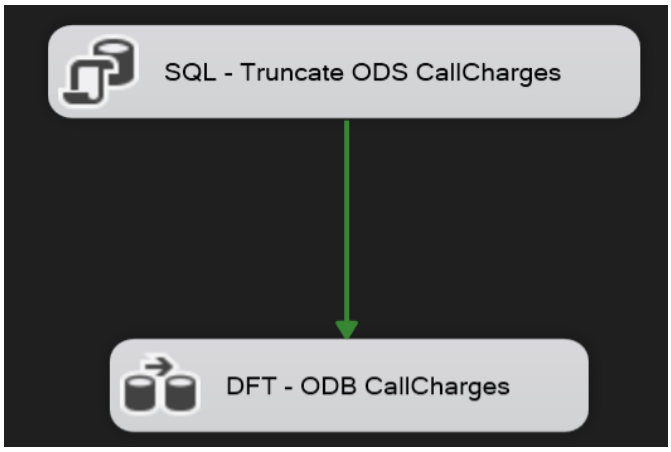


In the ODS Employees table, we got the result like this:

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

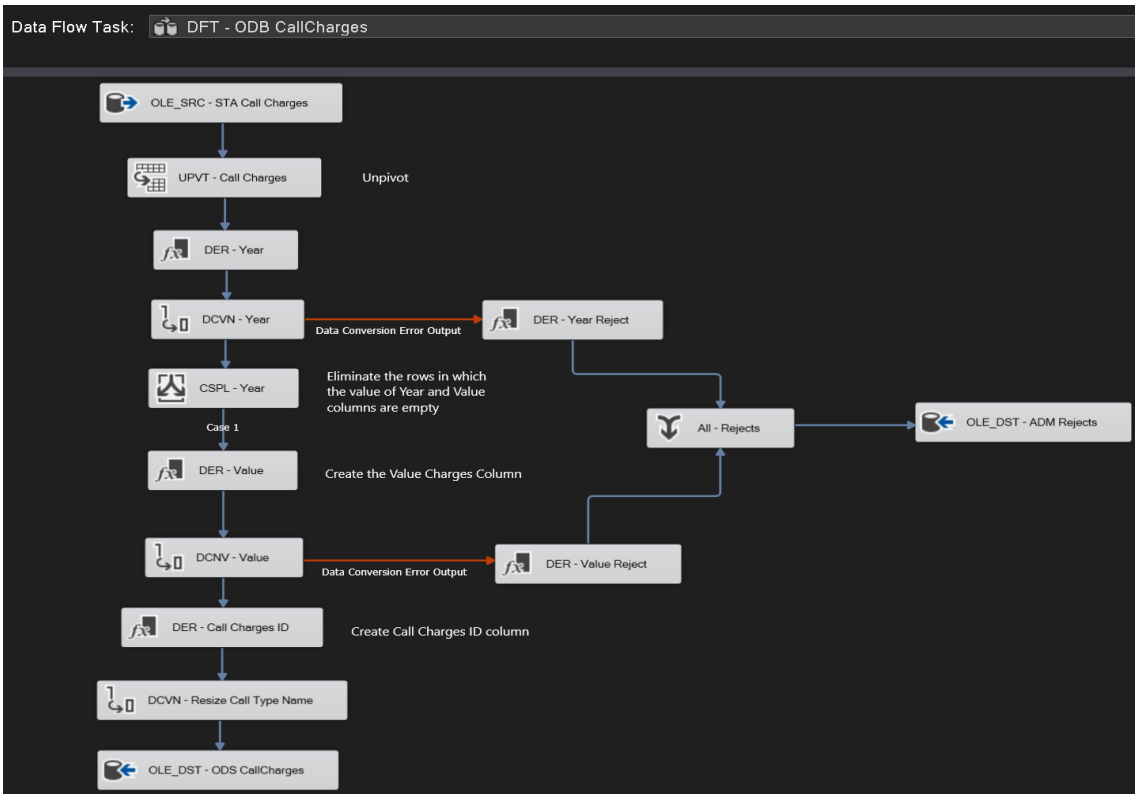
3.2. Call Charges Table

The first step is always to truncate the table:

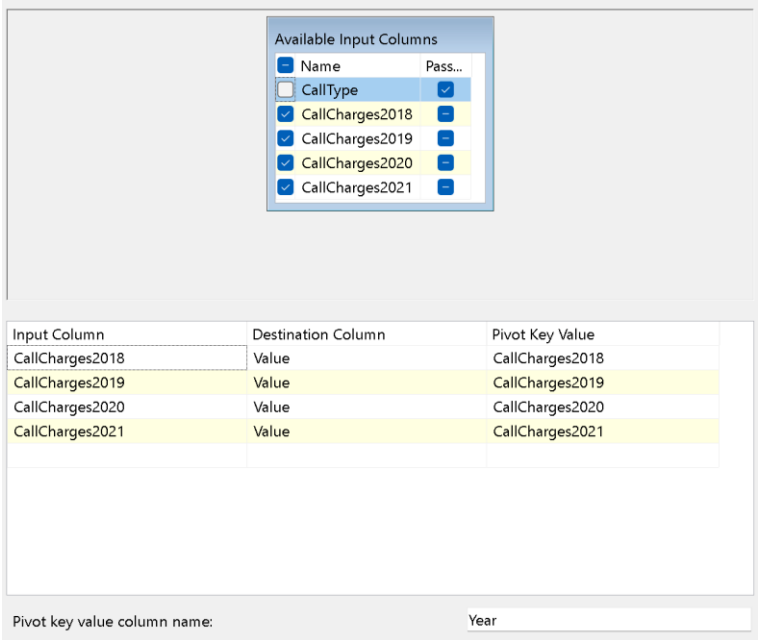


Then, we created the data flow. The process encompasses a total of 10 steps (excluding those related to handling or rejecting erroneous data):

1. Extract data from the STA Call Charges table.
2. Unpivot the Call Charges table to create two columns: Year and Charge Value.
3. Extract and retain only the year data (e.g., extracting "2018" from "CallCharges2018").
4. Transform the Year column to a string type with a length of 4 characters to facilitate the creation of the Call Charges ID column later.
5. Remove rows with empty values in the Year and Value columns.
6. Extract and retain only the charge values (e.g., extracting "1.52" from "1.52 / min").
7. Convert the Value column to a numeric type.
8. Create the Call Charges ID column.
9. Resize the columns to standardize their lengths.
10. Load the data into the server and create the ODS Call Charges table.



In the unpivoting step, here is our configuration:



Then, we extracted the year:

Derived Column Name	Derived Column	Expression	Data Type	Length
Year	Replace 'Year'	SUBSTRING(Year,12,4)	Unicode string [DT_WSTR]	255

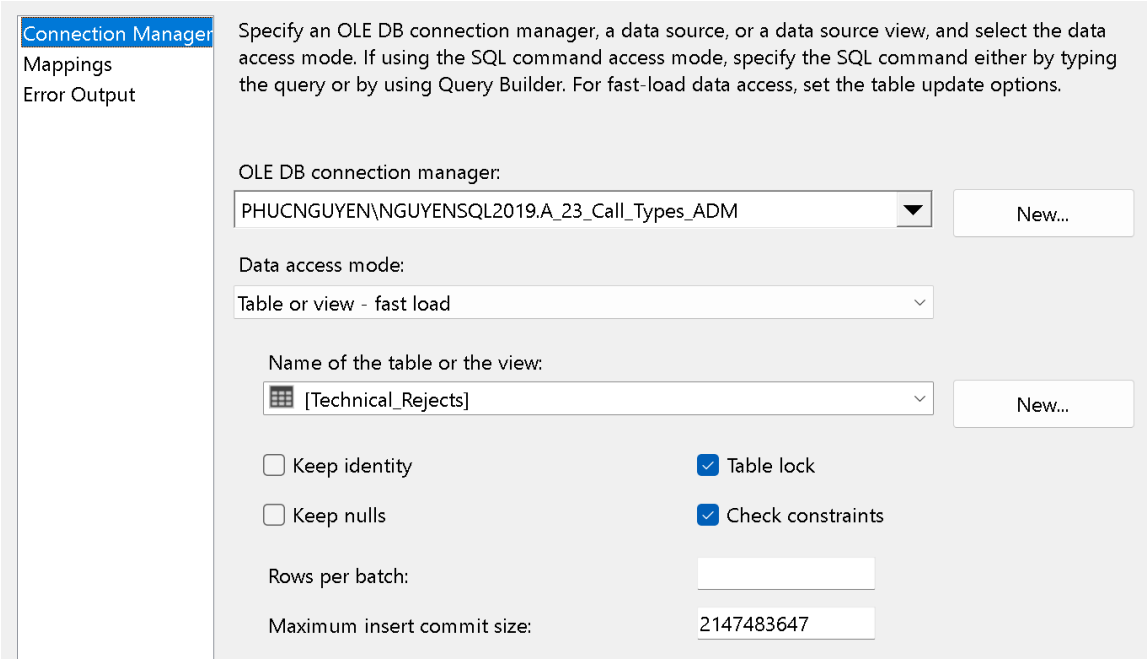
In the 4th step, its type is also changed:

Input Column	Output Alias	Data Type	Length	Precision	Scale	Code Page
Year	Year	string [DT_STR]	4			1252 (ANSI

For the rejected value of Year, we created a flow:

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] + "AN...	Unicode string [DT_WSTR]	203
RejectColumn	<add as new column>	"Year"	Unicode string [DT_WSTR]	4
RejectDescription	<add as new column>	"The value " + Year + "is not a valid Year"	Unicode string [DT_WSTR]	284

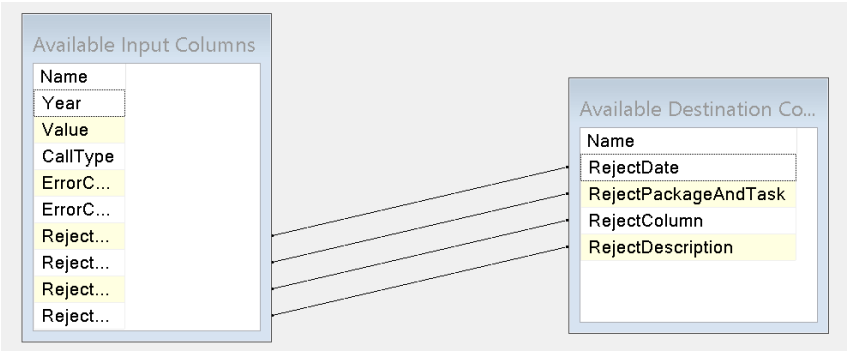
We also made an ADM database which will contain all the error values in the future:



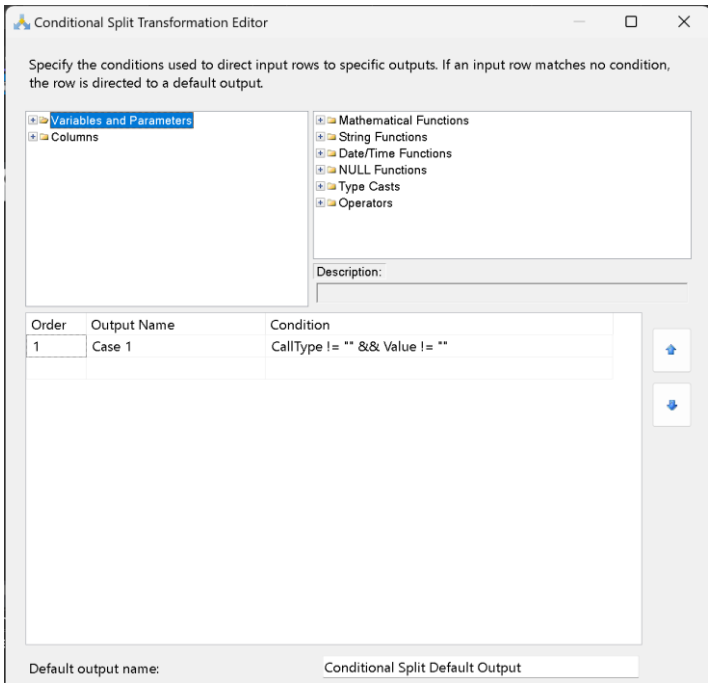
Here is the SQL code to create the ADM table:

```sql
CREATE TABLE Technical_Rejects (
    [RejectDate] datetime,
    [RejectPackageAndTask] nvarchar(203),
    [RejectColumn] nvarchar(50),
    [RejectDescription] nvarchar(287)
)
```

The mapping for the ADM database:



Then, we found a lot of empty values for the Year and Value columns because there were empty rows from the data in STA which from the data in STA which is still not cleaned yet. So, we decided to eliminate those rows by using Conditional Split Transformation Editor. Here is the configuration:



In the 6th step, we also extracted the value from the Value column, so we can have standardized rows with only the number instead of number and text (e.g., extracting "1.52" from "1.52 / min"):

Derived Column Name	Derived Column	Expression	Data Type	Length
Value	Replace 'Value'	LEFT(Value,FINDSTRING(Value," ",1) - 1)	string [DT_STR]	255

Next step, we converted it to numeric type, making the scale at 2:

Input Column	Output Alias	Data Type	Length	Precision	Scale
Value	Value	numeric [DT_NUMERIC]		18	2

In case of error values, we created a flow to put them in the ADM database:

Derived Column Name	Derived Column	Expression	Data Type	Length
RejectDate	<add as new column>	GETDATE()	database timestamp [...	
RejectPackageAndTask	<add as new column>	(DT_WSTR,100)@[System::PackageName] + "AN...	Unicode string [DT_W...	203
RejectColumn	<add as new column>	"Value"	Unicode string [DT_W...	5
RejectDescription	<add as new column>	"The value " + Value + "is not a valid Numeric"	Unicode string [DT_W...	287

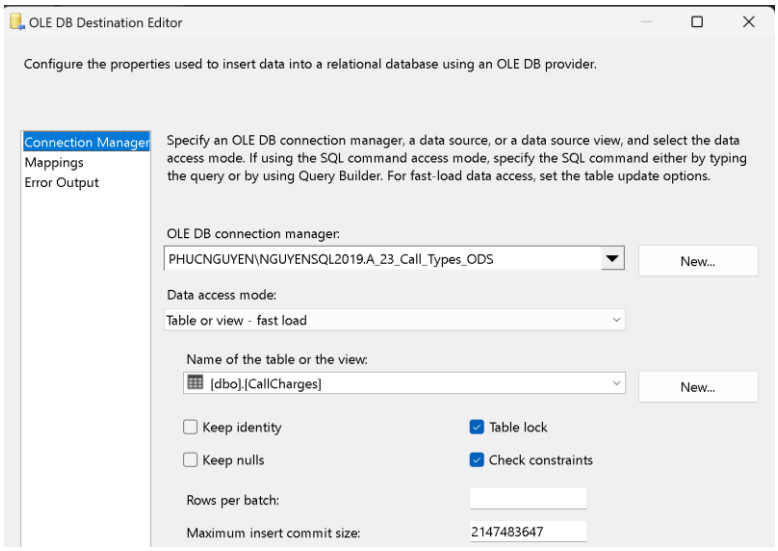
At the 8th step, we created a new column, “CallChargesID,” which is a combination of the Call Type column and the Year column. This will help create a primary key for this table, which is very useful for connecting with the Call Data table in the future by using this key. It is also an effective method when there are more charges by type in the future because the new IDs will be generated automatically. We did it by creating a derived column transformation editor, here is the expression:

Derived Column Name	Derived Column	Expression	Data Type	Leng
CallChargesID	<add as new column>	(DT_STR,50,1252)REPLACE((UPPER(CallType) + [UPVT - Call Charges].Year)," ","")	string [DT_STR]	50

Before putting the data in the ODS table, we resized the CallType column:

Input Column	Output Alias	Data Type	Length	Precision	Scale	Code Page
CallType	CallTypeName	string [DT_STR]	50			1252 (ANSI

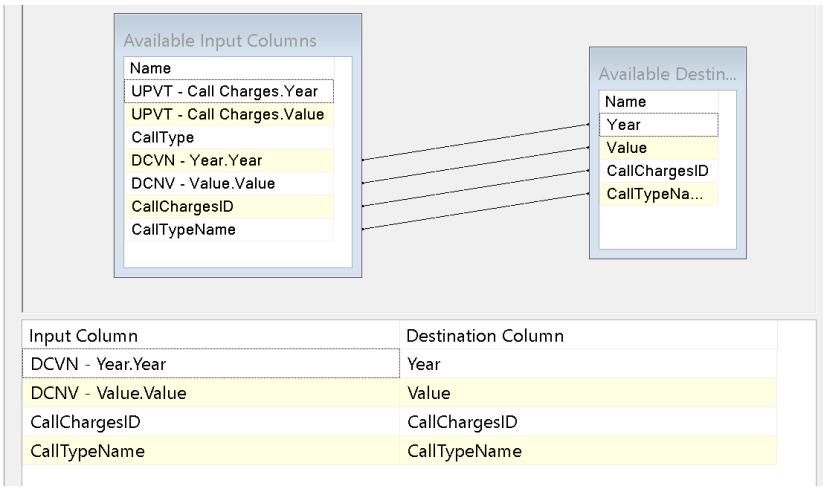
Last but not least, we put the data in the ODS Call Charges table:



Here is the SQL code to create the table:

```
CREATE TABLE CallCharges (
  [CallChargesID] varchar(50),
  [CallTypeName] varchar(50),
  [Year] varchar(4),
  [Value] numeric(18,2)
)
```

The mapping is described as below:

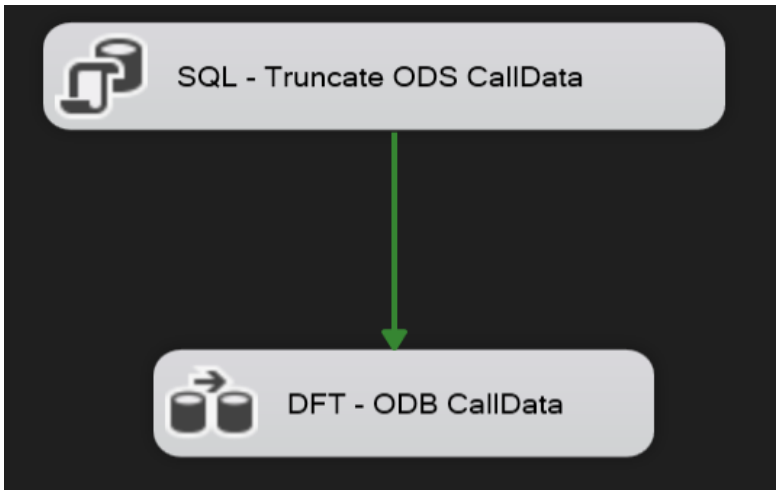


This is 10 first lines of the ODS Call Charges Table:

	CallChargesID	CallTypeName	Year	Value
1	SALES2018	Sales	2018	1.52
2	SALES2019	Sales	2019	1.56
3	SALES2020	Sales	2020	1.60
4	SALES2021	Sales	2021	1.71
5	BILLING2018	Billing	2018	1.20
6	BILLING2019	Billing	2019	1.32
7	BILLING2020	Billing	2020	1.41
8	BILLING2021	Billing	2021	1.45
9	TECHSUPPORT2018	Tech Support	2018	0.95
10	TECHSUPPORT2019	Tech Support	2019	0.98

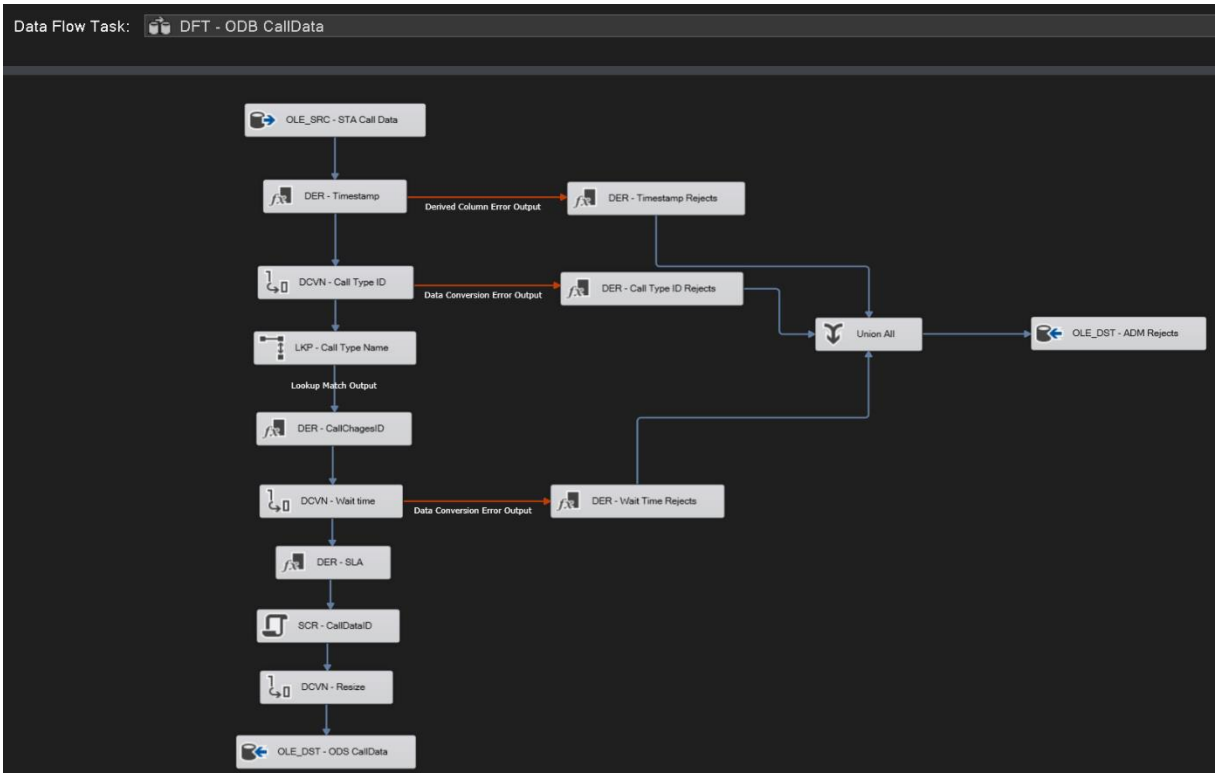
3.3. Data Call Table

Like the 2 previous table, we did the truncation to avoid any case of duplication:



This is the data flow for this table. A total of 10 steps will be executed (excluding steps for handling erroneous data):

1. Input data from the STA Call Data table.
2. Split the Timestamp column into two separate columns: Date and Time.
3. Change the type of the Call Type column (serving as Call Type ID) to do a lookup at the next step.
4. Perform a lookup with the STA Call Type table to create a new column, Call Type Name, which contains the names of the call types.
5. Create a Call Charges ID column to facilitate future connection with the ODS Call Charges table.
6. Convert the type of the Wait Time column to support calculations and to set up the SLA table in the next step.
7. Create an SLA table to meet project requirements.
8. Create a new column, Call Data ID, which will serve as the primary key for the ODS Call Data table.
9. Resize the columns to ensure consistency.
10. Load the data into the server and create the ODS Call Data table.



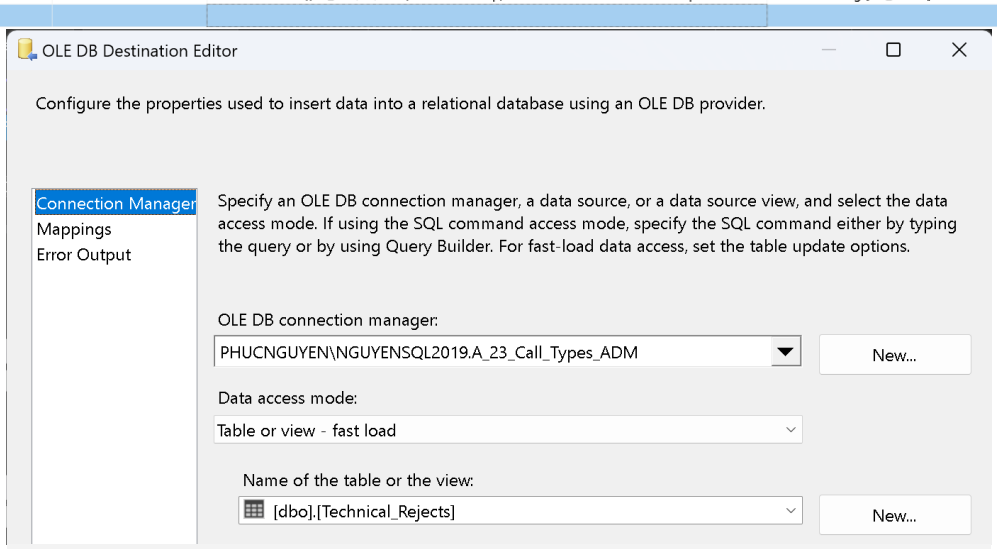
To split the timestamp into 2 columns, here is the expression for the derived column:

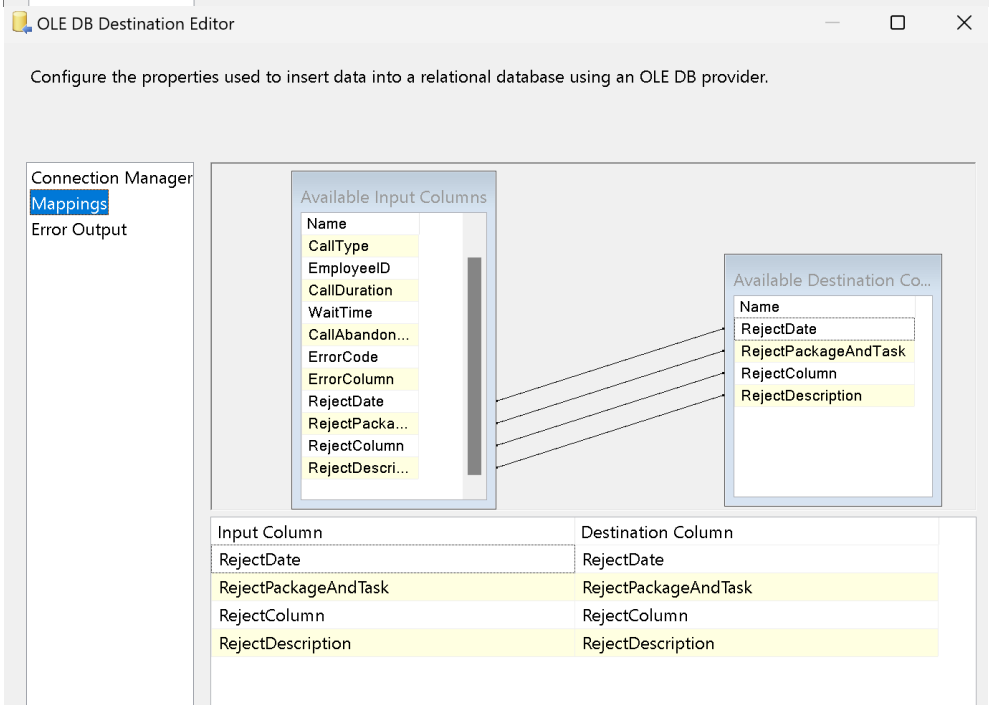
Derived Column Name	Expression
Date	(DT_DBDATE)((DT_STR,4,1252)YEAR(CallTimestamp) + "-" + RIGHT("0" + (DT_STR,2,1252)MONTH(CallTimestamp),2) + "-" + RIGHT("0" + (DT_STR,2,1252)DAY(CallTimestamp),2))
Time	(DT_DBTIME2,0)((DT_STR,2,1252)RIGHT("0" + (DT_STR,2,1252)DATEPART("hh",CallTimestamp),2) + ":" + RIGHT("0" +

	(DT_STR,2,1252)DATEPART("mi",CallTimestamp),2) + ":" + RIGHT("0" + (DT_STR,2,1252)DATEPART("ss",CallTimestamp),2))

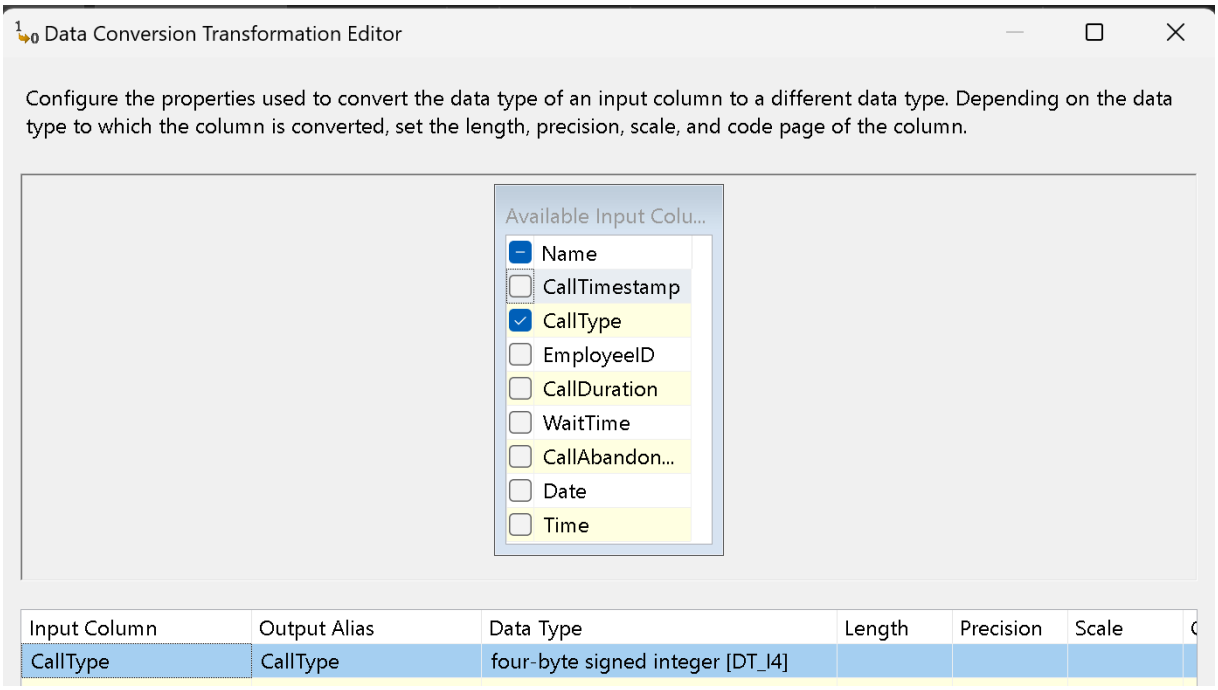
This is the expression in case there are errors for the Timestamp:

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]	203
RejectColumn	<add as new column>	"CallTimestamp "	Unicode string [DT_WSTR]	14
RejectDescription	<add as new column>	"The value " + ((DT_STR,50,1252)CallTimestamp) + "is not a valid CallTimestamp"	Unicode string [DT_WSTR]	88

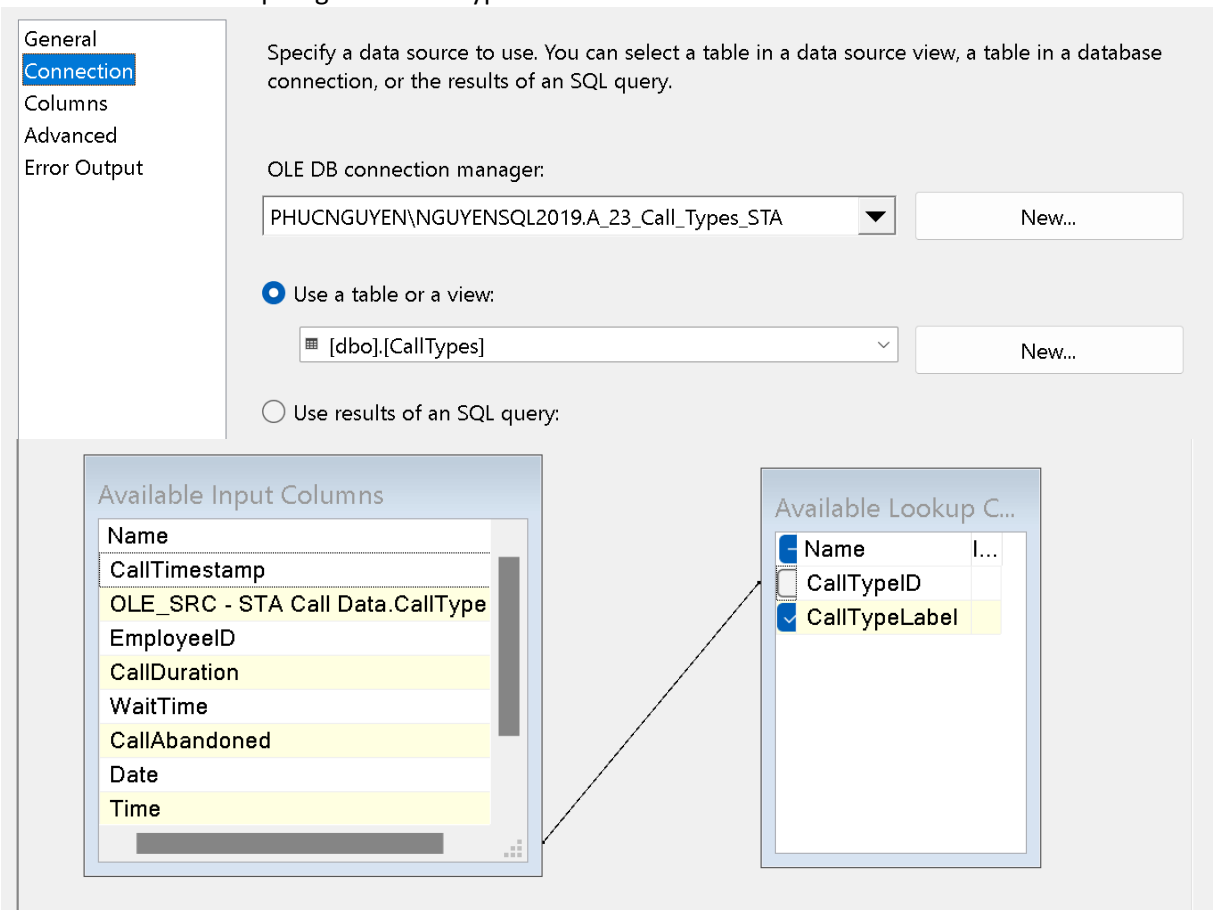




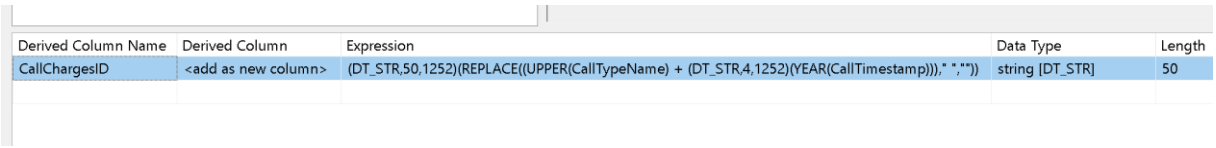
In the 3rd step, we had to change the Call Type (which is the Call Type ID) column so we can do the lookup in the next step:



Then we do the lookup to get the Call Type's name:

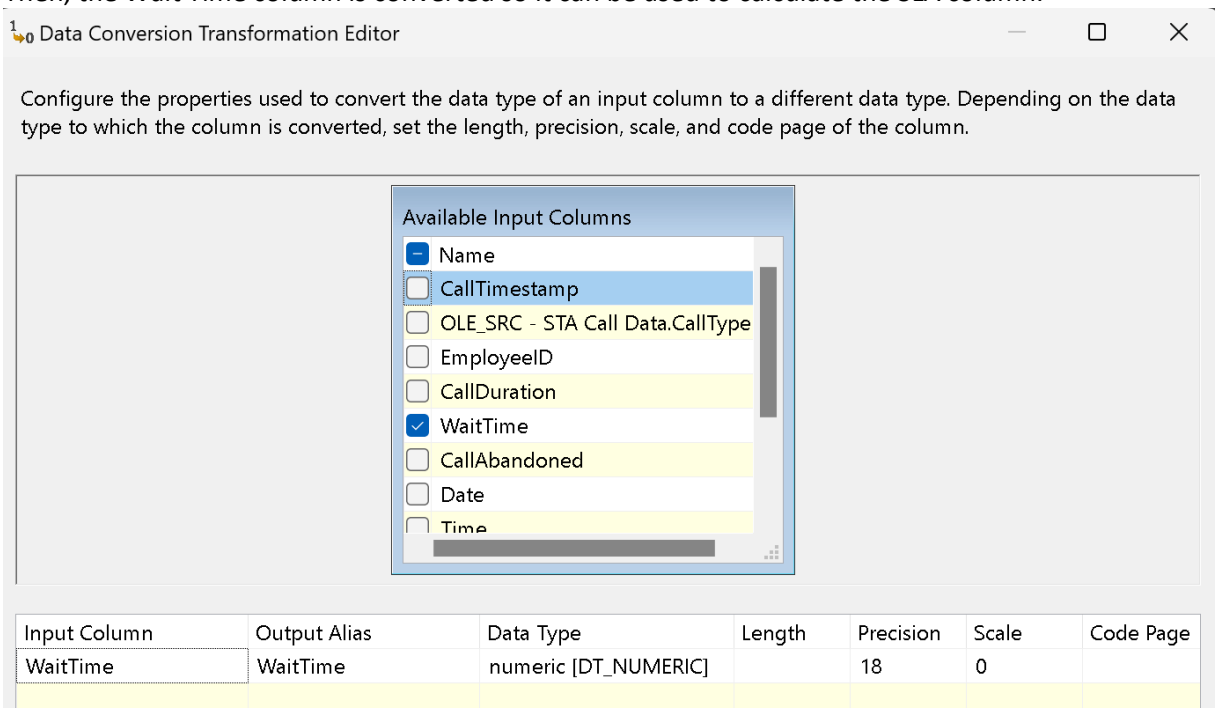


In the next step, we need to create the Call Charges ID Column so we can connect to to Call Charges table:

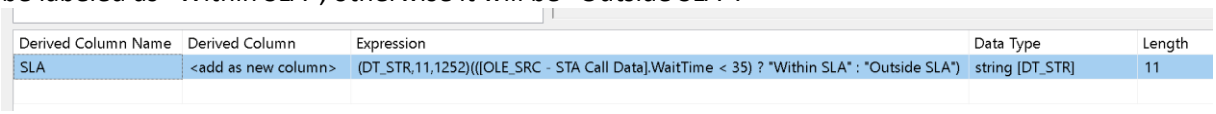


Derived Column Name	Derived Column	Expression	Data Type	Length
CallChargesID	<add as new column>	(DT_STR,50,1252)(REPLACE((UPPER(CallTypeName) + (DT_STR,4,1252)(YEAR(CallTimestamp))),"",""))	string [DT_STR]	50

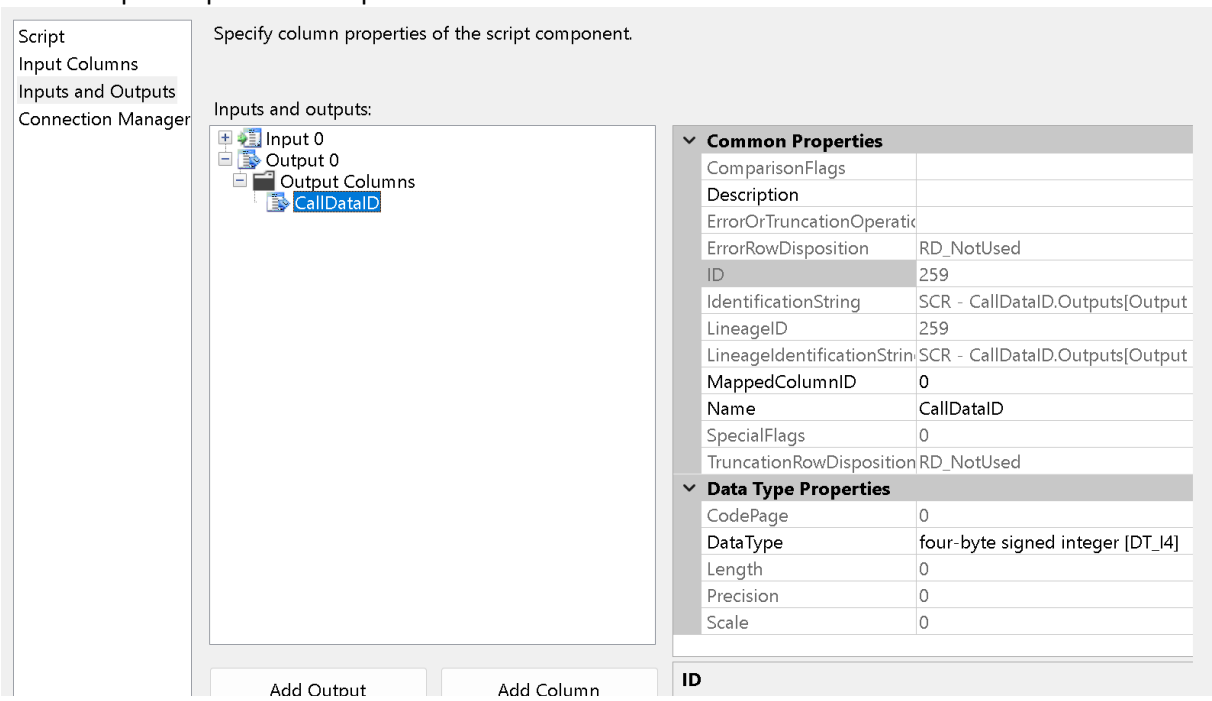
Then, the Wait Time column is converted so it can be used to calculate the SLA column:



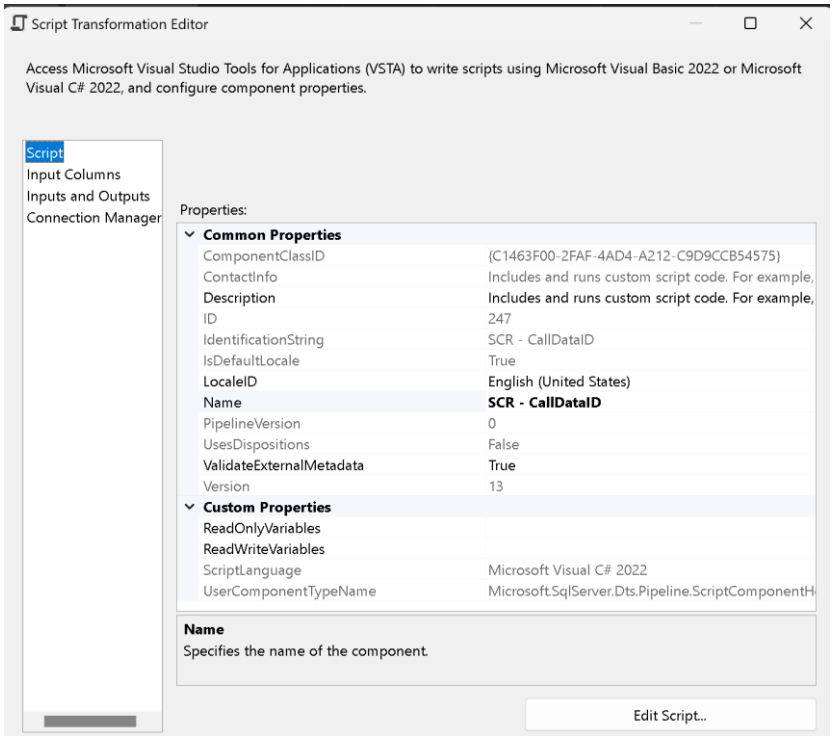
The SLA is created to meet the requirements of the project, if the wait time is below 35 seconds, it will be labeled as “Within SLA”, otherwise it will be “Outside SLA”:

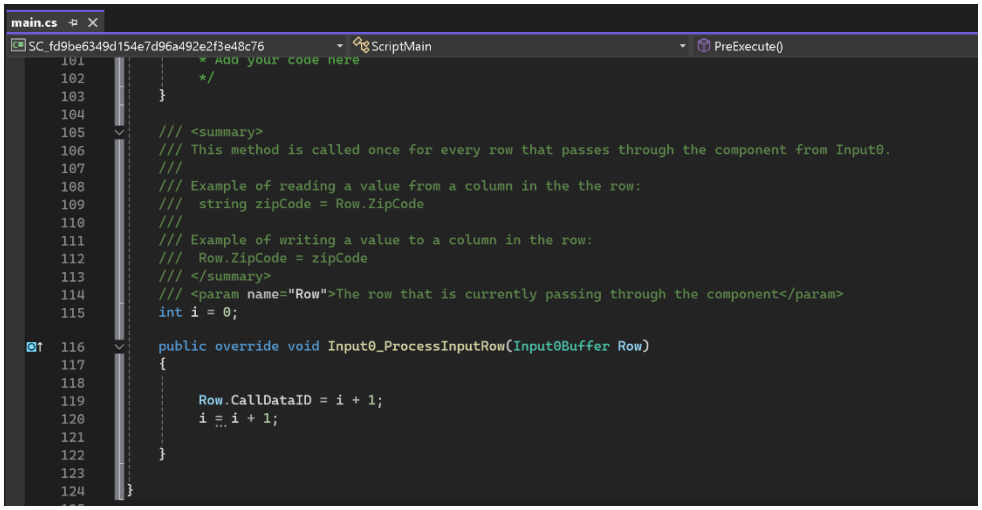


In the 8th step, we created the Call Data ID column, which ensure that this table will have a primary key. To do it, we use the Script Transformation Editor in the toolbox.
In the Script -> Inputs and Outputs -> Create the CallDataID as below:

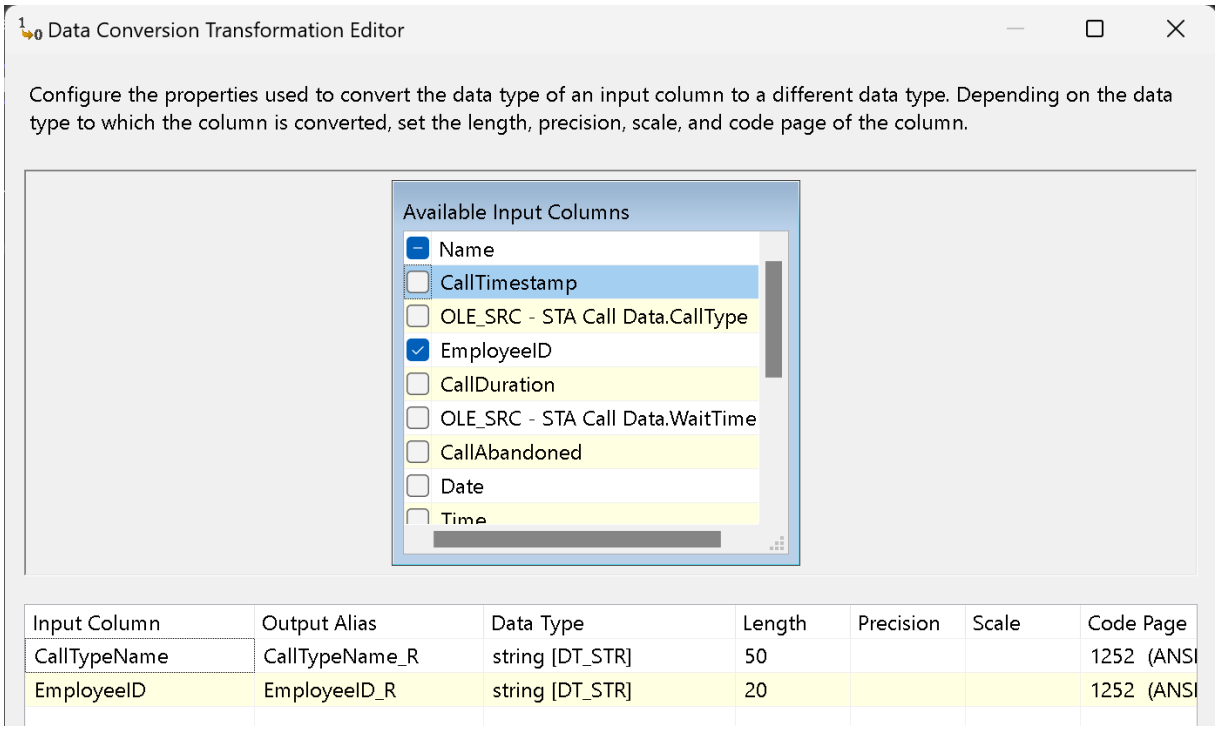


In the Script -> Edit Script -> At the end of the script, we wrote the code to generate the row by counting the row:

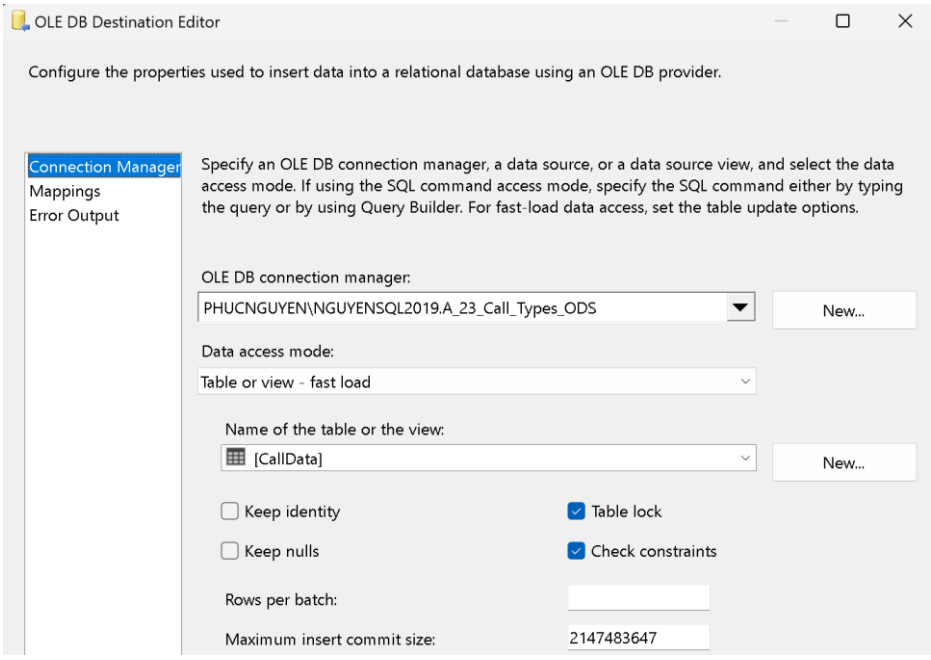




At the 9th step, we resize the Call Type Name and Employee ID:



Last step, we put the data to the ODS Call Data table:

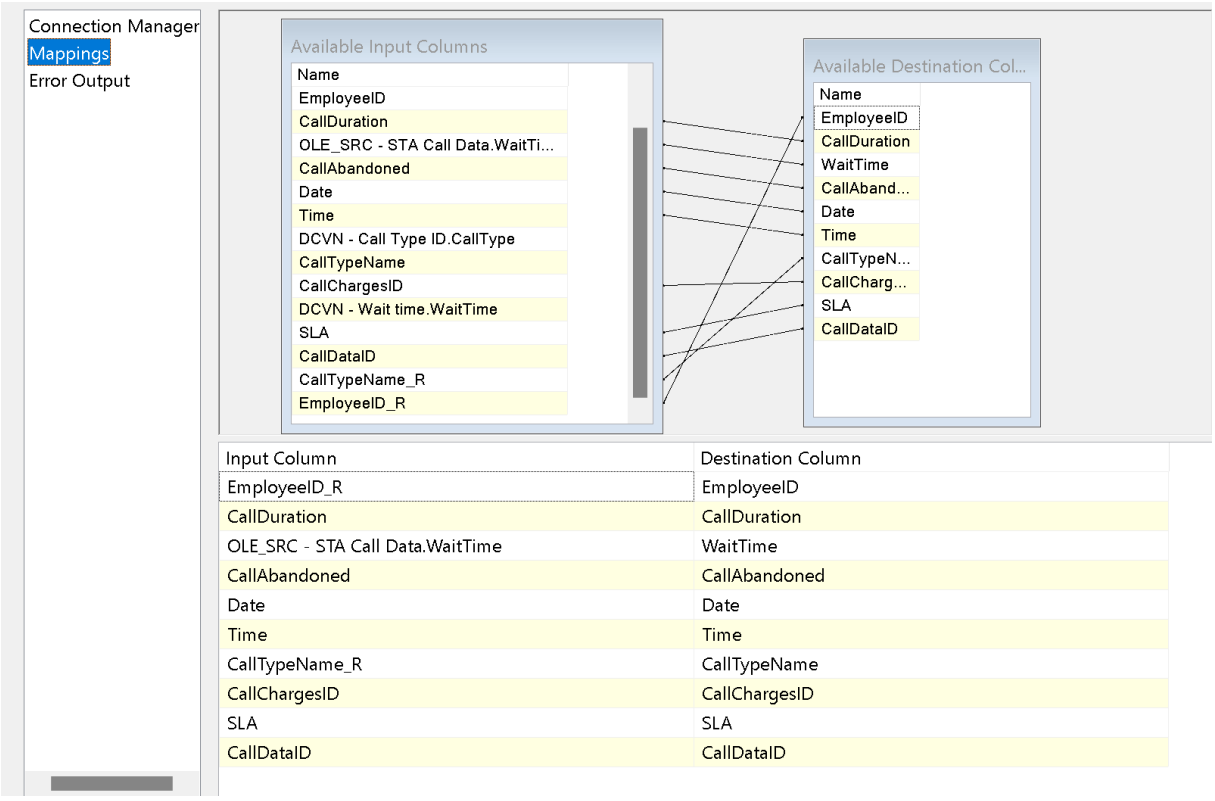


Here is the SQL code:

```
CREATE TABLE CallData (
    [CallDataID] int,
    [Date] date,
    [Time] time(0),
    [CallTypeName] varchar(50),
    [EmployeeID] varchar(20),
    [CallDuration] int,
```

```
    [WaitTime] int,
    [CallAbandoned] bit,
    [SLA] varchar(11),
    [CallChargesID] varchar(50)
)
```

Here is the mapping:



And this is the first 10 results:

	CallDataID	Date	Time	CallTypeName	EmployeeID	CallDuration	WaitTime	CallAbandoned	SLA	CallChargesID
1	1	2018-05-04	16:33:00	Tech Support	U559430	486	2	0	Within SLA	TECHSUPPORT2018
2	2	2018-06-21	18:28:00	Tech Support	A166733	945	0	0	Within SLA	TECHSUPPORT2018
3	3	2018-06-21	15:13:00	Sales	B971624	379	11	0	Within SLA	SALES2018
4	4	2018-11-21	13:02:00	Tech Support	U641256	1044	0	0	Within SLA	TECHSUPPORT2018
5	5	2018-02-25	13:36:00	Sales	P286634	1357	0	0	Within SLA	SALES2018
6	6	2018-10-28	10:04:00	Tech Support	M855788	570	23	0	Within SLA	TECHSUPPORT2018
7	7	2018-12-17	16:39:00	Tech Support	M794992	26	26	0	Within SLA	TECHSUPPORT2018
8	8	2018-07-28	19:09:00	Billing	I281837	8	8	0	Within SLA	BILLING2018
9	9	2018-11-06	18:57:00	Sales	J192194	800	0	0	Within SLA	SALES2018
10	10	2018-06-18	15:32:00	Billing	F542348	651	111	0	Outside SLA	BILLING2018

All the errors are sent to the same “Technical_Rejects” table in ADM database.

4. Datawarehouse

The final stage of the data pipeline involves integrating the data into the Data Warehouse. A commonly used database schema for this purpose is the Star schema. In this schema, there is a central table known as the "Fact Table," which is surrounded by "Dimension Tables." The Fact Table holds the critical data, referred to as "facts," while the Dimension Tables provide additional descriptive information. Our database will be designed based on this schema.

4.1. Database design

Based on our data, the important data are the calls data. Therefore, we choose to build the fact table with the “Calls Data” table.

The common dimension is the “Employees” dimension. It will help describe the geographical data and employee information. We choose to use an incremental indices technology key (EmployeeSurroundKey) for the relation to the fact table.

The “Date” dimension allows you to describe dates with multiple temporal aggregate categories (year, month, quarter). The table will contain multiple variations of those three data to adapt to various query styles. We define the relation to the fact table with a technical key with the format “YYYYMMDD.”

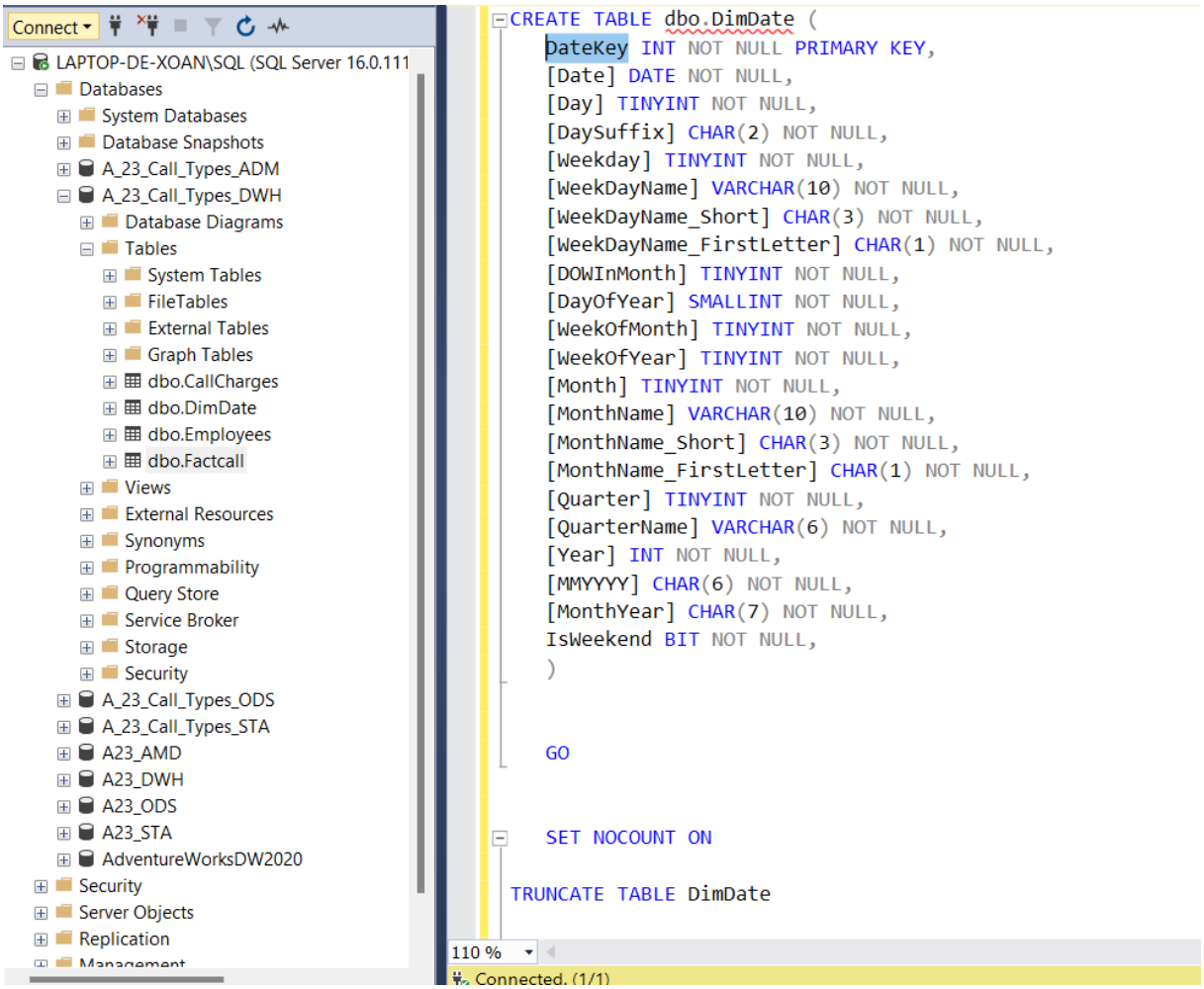
The “Call Charge dimension” will help describe the call's value based on year and CallTypeName. We choose to use an incremental indices technical key (CallChargesKey)for the relation to the fact table.

4.2. Integration of the Date dimension

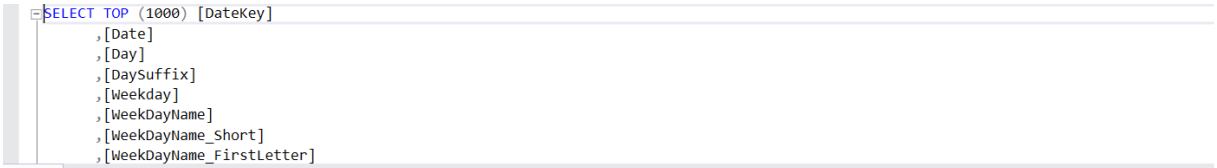
For this dimension, we used the external data “DimDate” to create the table and DateKey is the primary key of this.

```
CREATE TABLE dbo.DimDate (
   DateKey INT NOT NULL PRIMARY KEY,
   [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,
   )


   GO


   SET NOCOUNT ON

TRUNCATE TABLE DimDate

DECLARE @CurrentDate DATE = '2018-01-01'
DECLARE @EndDate DATE = '2021-12-31'

WHILE @CurrentDate < @EndDate
BEGIN
   INSERT INTO [dbo].[DimDate] (
      [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]
      )
   SELECT DateKey = YEAR(@CurrentDate) * 10000 + MONTH(@CurrentDate) * 100 +
DAY(@CurrentDate),
      DATE = @CurrentDate,
      Day = DAY(@CurrentDate),
      [DaySuffix] = CASE
         WHEN DAY(@CurrentDate) = 1
            OR DAY(@CurrentDate) = 21
            OR DAY(@CurrentDate) = 31
            THEN 'st'
         WHEN DAY(@CurrentDate) = 2
            OR DAY(@CurrentDate) = 22
            THEN 'nd'
         WHEN DAY(@CurrentDate) = 3
            OR DAY(@CurrentDate) = 23
            THEN 'rd'
         ELSE 'th'
         END,
      WEEKDAY = DATEPART(dw, @CurrentDate),
      WeekDayName = DATENAME(dw, @CurrentDate),
      WeekDayName_Short = UPPER(LEFT(DATENAME(dw, @CurrentDate), 3)),
      WeekDayName_FirstLetter = LEFT(DATENAME(dw, @CurrentDate), 1),
      [DOWInMonth] = DAY(@CurrentDate),
      [DayOfYear] = DATENAME(dy, @CurrentDate),
      [WeekOfMonth] = DATEPART(WEEK, @CurrentDate) - DATEPART(WEEK, DATEADD(MM,
DATEDIFF(MM, 0, @CurrentDate), 0)) + 1,
      [WeekOfYear] = DATEPART(wk, @CurrentDate),
      [Month] = MONTH(@CurrentDate),
      [MonthName] = DATENAME(mm, @CurrentDate),
      [MonthName_Short] = UPPER(LEFT(DATENAME(mm, @CurrentDate), 3)),
      [MonthName_FirstLetter] = LEFT(DATENAME(mm, @CurrentDate), 1),
      [Quarter] = DATEPART(q, @CurrentDate),
      [QuarterName] = CASE
         WHEN DATENAME(qq, @CurrentDate) = 1
            THEN 'First'
         WHEN DATENAME(qq, @CurrentDate) = 2
            THEN 'second'
         WHEN DATENAME(qq, @CurrentDate) = 3
            THEN 'third'
         WHEN DATENAME(qq, @CurrentDate) = 4
            THEN 'fourth'
         END,
      [Year] = YEAR(@CurrentDate),
      [MMYYYY] = RIGHT('0' + CAST(MONTH(@CurrentDate) AS VARCHAR(2)), 2) +
CAST(YEAR(@CurrentDate) AS VARCHAR(4)),
      [MonthYear] = CAST(YEAR(@CurrentDate) AS VARCHAR(4)) + UPPER(LEFT(DATENAME(mm,
@CurrentDate), 3)),
      [IsWeekend] = CASE
         WHEN DATENAME(dw, @CurrentDate) = 'Sunday'
            OR DATENAME(dw, @CurrentDate) = 'Saturday'
            THEN 1
         ELSE 0
         END

   SET @CurrentDate = DATEADD(DD, 1, @CurrentDate)
END
```



Below is the table Dimdate:



	DateKey	Date	Day	DaySuffix	Weekday	WeekDayName	WeekDayName_Short	WeekDayName_FirstLetter	DOWInMonth	DayOfYear	WeekOfMonth	WeekOfYear	Month	MonthName	MonthName_Short	Month
4	20180104	2018-01-04	4	th	5	Thursday	THU	T	4	4	1	1	1	January	JAN	J
5	20180105	2018-01-05	5	th	6	Friday	FRI	F	5	5	1	1	1	January	JAN	J
6	20180106	2018-01-06	6	th	7	Saturday	SAT	S	6	6	1	1	1	January	JAN	J
7	20180107	2018-01-07	7	th	1	Sunday	SUN	S	7	7	2	2	1	January	JAN	J
8	20180108	2018-01-08	8	th	2	Monday	MON	M	8	8	2	2	1	January	JAN	J
9	20180109	2018-01-09	9	th	3	Tuesday	TUE	T	9	9	2	2	1	January	JAN	J
10	20180110	2018-01-10	10	th	4	Wednesday	WED	W	10	10	2	2	1	January	JAN	J
11	20180111	2018-01-11	11	th	5	Thursday	THU	T	11	11	2	2	1	January	JAN	J
12	20180112	2018-01-12	12	th	6	Friday	FRI	F	12	12	2	2	1	January	JAN	J
13	20180113	2018-01-13	13	th	7	Saturday	SAT	S	13	13	2	2	1	January	JAN	J
14	20180114	2018-01-14	14	th	1	Sunday	SUN	S	14	14	3	3	1	January	JAN	J
15	20180115	2018-01-15	15	th	2	Monday	MON	M	15	15	3	3	1	January	JAN	J
16	20180116	2018-01-16	16	th	3	Tuesday	TUE	T	16	16	3	3	1	January	JAN	J
17	20180117	2018-01-17	17	th	4	Wednesday	WED	W	17	17	3	3	1	January	JAN	J
18	20180118	2018-01-18	18	th	5	Thursday	THU	T	18	18	3	3	1	January	JAN	J
19	20180119	2018-01-19	19	th	6	Friday	FRI	F	19	19	3	3	1	January	JAN	J
20	20180120	2018-01-20	20	th	7	Saturday	SAT	S	20	20	3	3	1	January	JAN	J
21	20180121	2018-01-21	21	st	1	Sunday	SUN	S	21	21	4	4	1	January	JAN	J
22	20180122	2018-01-22	22	nd	2	Monday	MON	M	22	22	4	4	1	January	JAN	J
23	20180123	2018-01-23	23	rd	3	Tuesday	TUE	T	23	23	4	4	1	January	JAN	J
24	20180124	2018-01-24	24	th	4	Wednesday	WED	W	24	24	4	4	1	January	JAN	J
25	20180125	2018-01-25	25	th	5	Thursday	THU	T	25	25	4	4	1	January	JAN	J
26	20180126	2018-01-26	26	th	6	Friday	FRI	F	26	26	4	4	1	January	JAN	J
27	20180127	2018-01-27	27	th	7	Saturday	SAT	S	27	27	4	4	1	January	JAN	J

4.3. Integration of the Employees dimension

In ODS we have clean data about the Employees. However, in fact, the information of the employee would may not be stable. Some will leave, and some will be replaced in their positions. Therefore, we need to have a method to record the new information of employees.

First, we need to create the table. In order to have a surroundkey which relates to the Fact table which name is [EmployeeSurroundKey]

```
CREATE TABLE [dbo].[Employees](
	[EmployeeSurroundKey] int primary key identity (1,1),
	[EmployeeID] [varchar](20) NULL,
	[EmployeeName] [varchar](100) NULL,
	[CityName] [varchar](50) NULL,
	[StateName] [varchar](50) NULL,
	[Region] [varchar](20) NULL,
	[ManagerName] [varchar](100) NULL
) ON [PRIMARY]
GO
```

The table below shows the information about the employee with geographic:

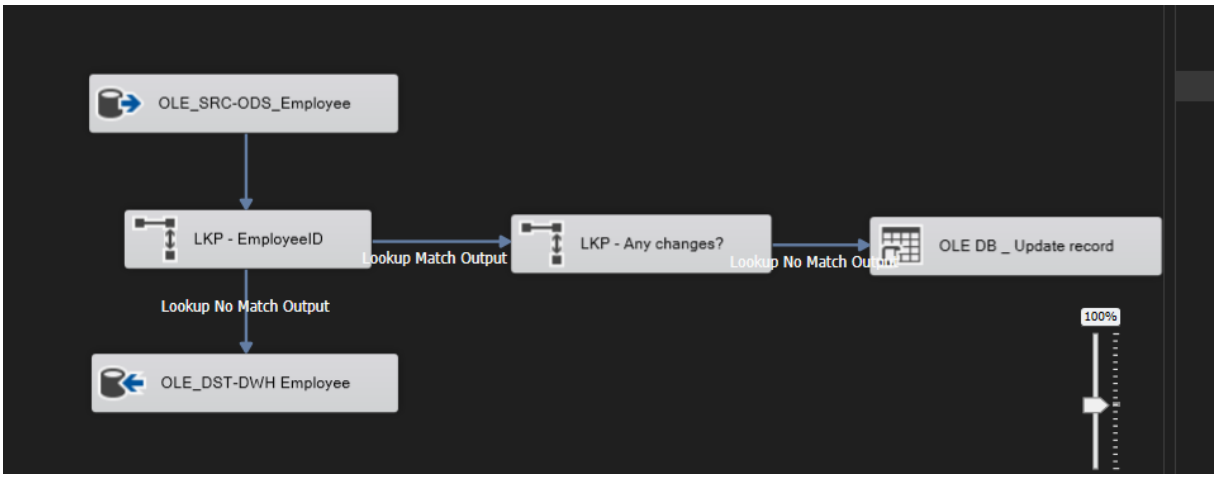
```
SELECT TOP (1000) [EmployeesurrogateKey]
      ,[EmployeeID]
      ,[EmployeeName]
      ,[CityName]
      ,[StateName]
      ,[Region]
      ,[ManagerName]
  FROM [A_23_Call_Types_DWH].[dbo].[Employees]
```

.0 %

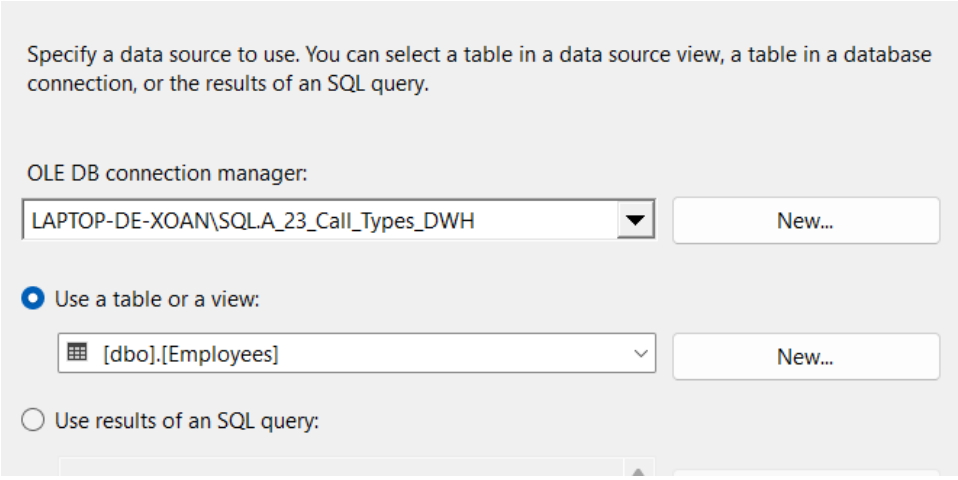
Results | Messages

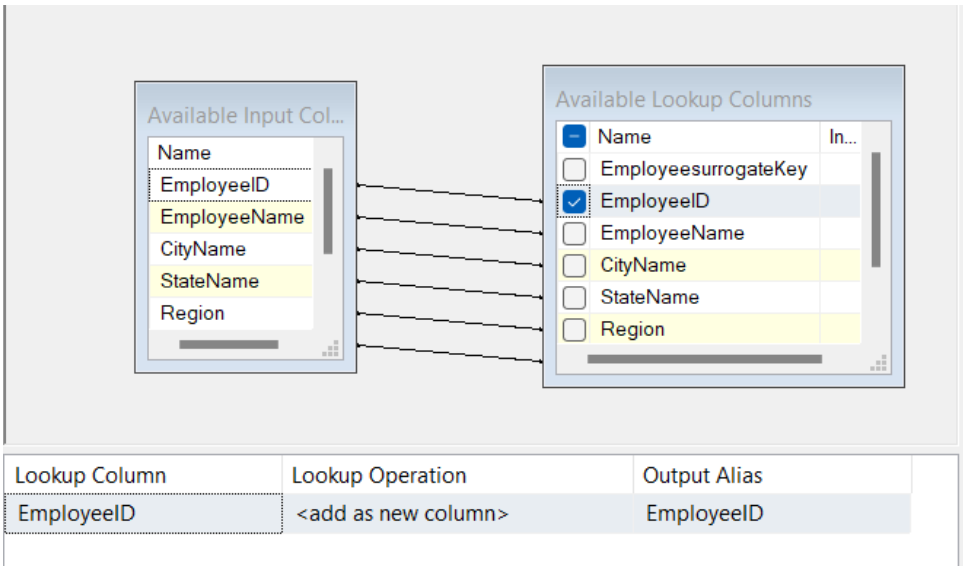
EmployeesurrogateKey	EmployeeID	EmployeeName	CityName	StateName	Region	ManagerName
1	N772493	Onita Trojan	Spokane	Washington	West	Deidre Robbs
2	F533051	Stormy Seller	Aurora	Colorado	West	Elsie Taplin
3	S564705	Mable Ayoub	Aurora	Colorado	West	Shala Lion
4	I281837	Latrisha Buckalew	Aurora	Colorado	West	Rana Taub
5	Y193775	Adrianna Duque	Spokane	Washington	West	Collin Trotman
6	J632516	Keiko Daulton	Spokane	Washington	West	Jamar Prahl
7	G727038	Dolores Lundeen	Aurora	Colorado	West	Shala Lion
8	V126561	Wilbur Mohl	Jacksonville	Florida	South	Casey Bainbridge
9	E243130	Ileen Bornstein	Jacksonville	Florida	South	Gonzalo Lesage
10	C206355	Janeth Roesler	Spokane	Washington	West	Miyoko Degraw
11	G586239	Shery Hover	Aurora	Colorado	West	Elsie Taplin
12	M163408	Trevor Cerda	Spokane	Washington	West	Collin Trotman
13	H438047	Debora Wilker	Spokane	Washington	West	Collin Trotman
14	T659609	Kelvin Nicoletti	Spokane	Washington	West	Jamar Prahl
15	P286634	Rafael Langton	Aurora	Colorado	West	Elsie Taplin
16	T387561	Alla Winkel	Jacksonville	Florida	South	Ardath Ducharme
17	J419954	Harrison Finlayson	Aurora	Colorado	West	Rana Taub
18	N470942	Caterina Jantz	Spokane	Washington	West	Miyoko Degraw
19	B971624	Agripina Snively	Aurora	Colorado	West	Elsie Taplin
20	O856292	Rosenda Korus	Jacksonville	Florida	South	Gonzalo Lesage
21	T664783	Ivette Rodenberg	Jacksonville	Florida	South	Gonzalo Lesage
22	U194381	Beulah Aubert	Jacksonville	Florida	South	Ardath Ducharme
23	N119221	Aleida Singh	Aurora	Colorado	West	Elsie Taplin
24	M855788	Janice Clatterbuck	Spokane	Washington	West	Collin Trotman
25	K222775	Junie Grant	Jacksonville	Florida	South	Casey Bainbridge
26	E988252	Brittanie Ballin	Jacksonville	Florida	South	Casey Bainbridge

Then we can load the data with the process bellow:

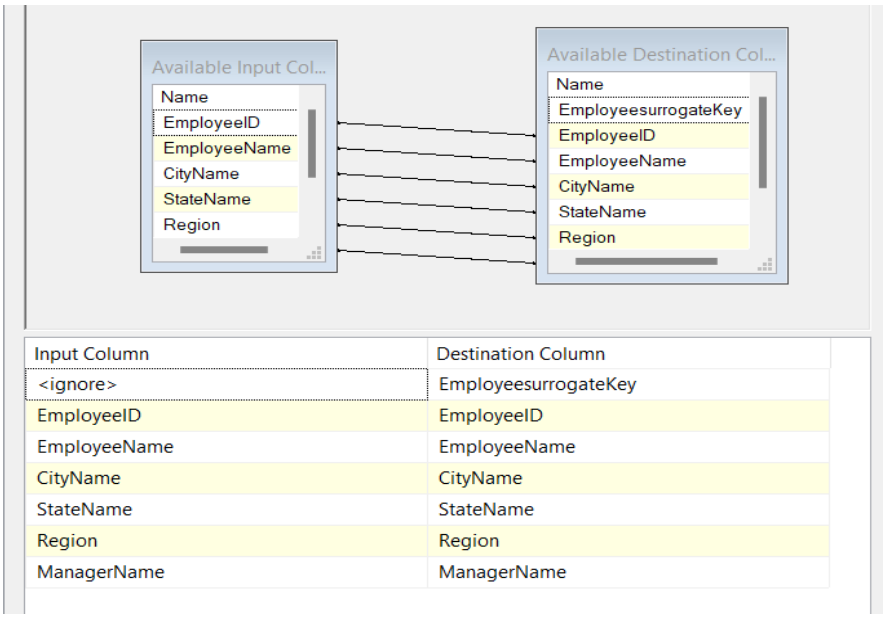


We need to make sure that we can make joins between the fact table and the dimension table, so we check the link with “EmployeeSurroundKey]”.





For the dimensions tables, we also need to have an update policy. We choose the SCD1 strategy, implemented by checking is there is any change and updating the table if necessary.



4.4. Integration of the Call charge dimension

Similar to the Employee dimension, the value of the call charge is based on the year and the call charge ID. Therefore, in order to keep the value of the data to be updated in the years after, we need to create a function that we can update this information.

```
CREATE TABLE [dbo].[CallCharges](
       [CallChargesSurroundKey] int primary key identity (1,1),
       [CallChargesID] [varchar](50) NULL,
       [CallTypeName] [varchar](50) NULL,
       [Year] [varchar](4) NULL,
       [Value] [numeric](18, 2) NULL
) ON [PRIMARY]
GO
```

The table below shows the information about the employee with geographic:

SQLQuery5.sql - L...RUNG NGUYEN (52)) | SQLQuery4.sql - L...RUNG NGUYEN (73))

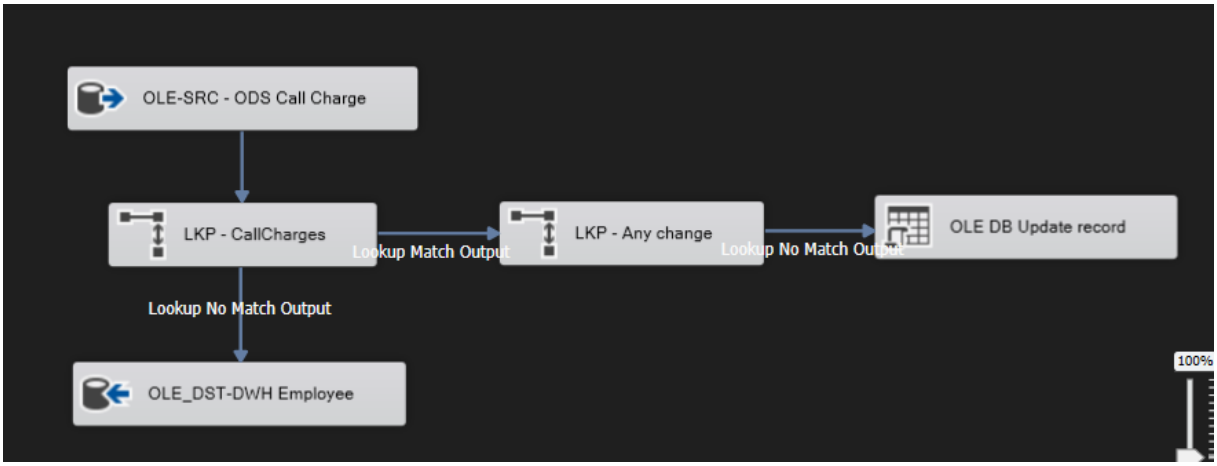
```sql
SELECT TOP (1000) [CallChargesSurrogateKey]
      ,[CallChargesID]
      ,[CallTypeName]
      ,[Year]
      ,[Value]
  FROM [A_23_Call_Types_DWH].[dbo].[CallCharges]
```

110 %

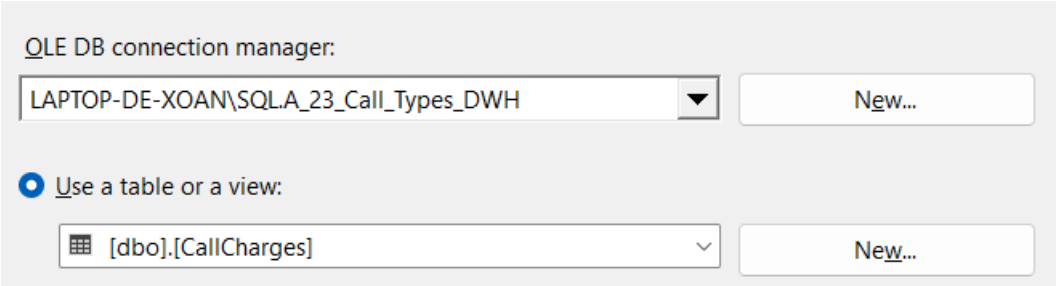
Results | Messages

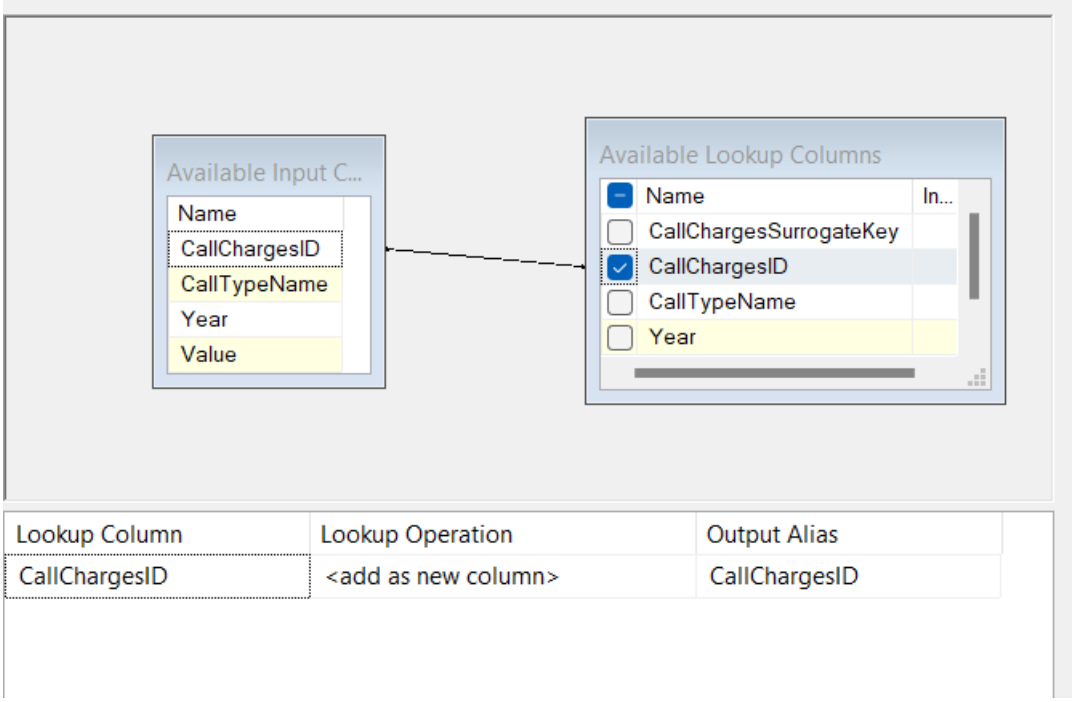
	CallChargesSurrogateKey	CallChargesID	CallTypeName	Year	Value
1	1	SALES2018	Sales	2018	1.52
2	2	SALES2019	Sales	2019	1.56
3	3	SALES2020	Sales	2020	1.60
4	4	SALES2021	Sales	2021	1.71
5	5	BILLING2018	Billing	2018	1.20
6	6	BILLING2019	Billing	2019	1.32
7	7	BILLING2020	Billing	2020	1.41
8	8	BILLING2021	Billing	2021	1.45
9	9	TECHSUPPORT2018	Tech Support	2018	0.95
10	10	TECHSUPPORT2019	Tech Support	2019	0.98
11	11	TECHSUPPORT2020	Tech Support	2020	1.04
12	12	TECHSUPPORT2021	Tech Support	2021	1.12

Then we can load the data with the process bellow:

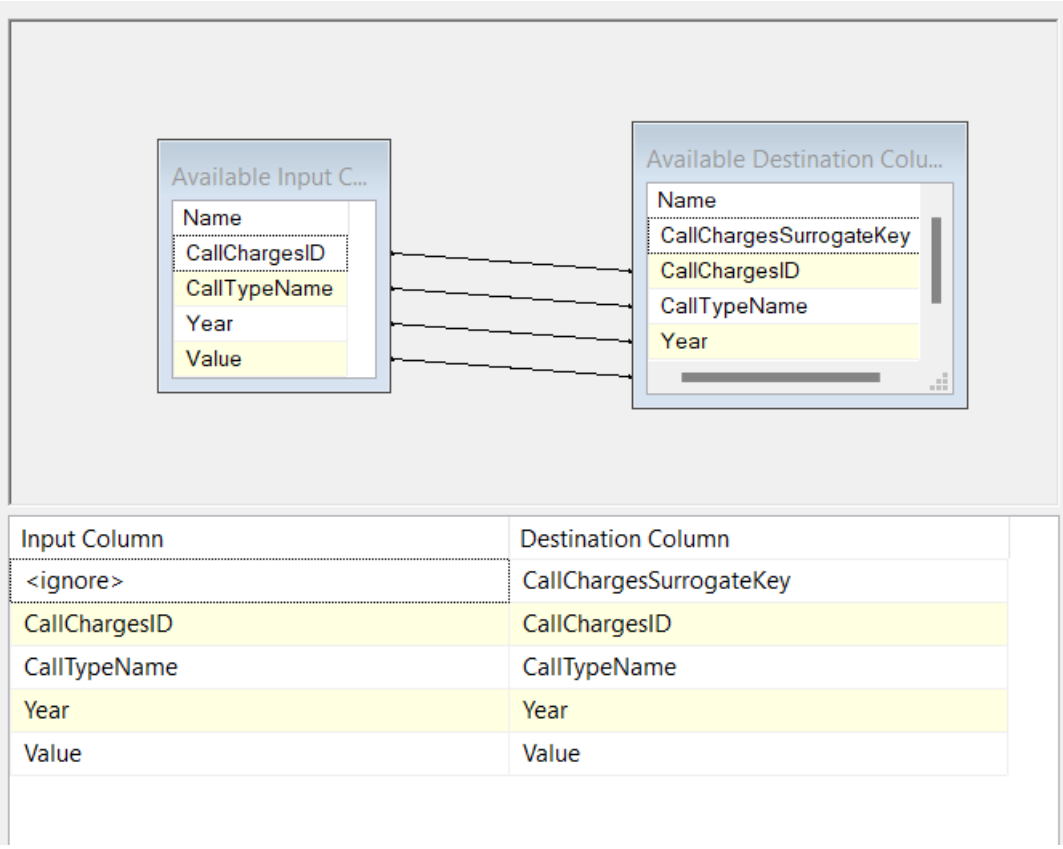


We need to make sure that we can make joins between the fact table and the dimension table, so we check the link with “Call chargeID”





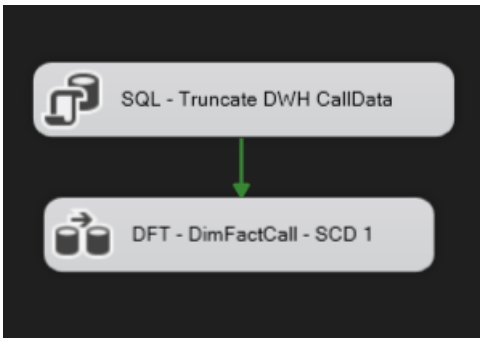
For the dimensions tables, we also need to have an update policy. We choose the SCD1 strategy, implemented by checking is there is any change and updating the table if necessary.



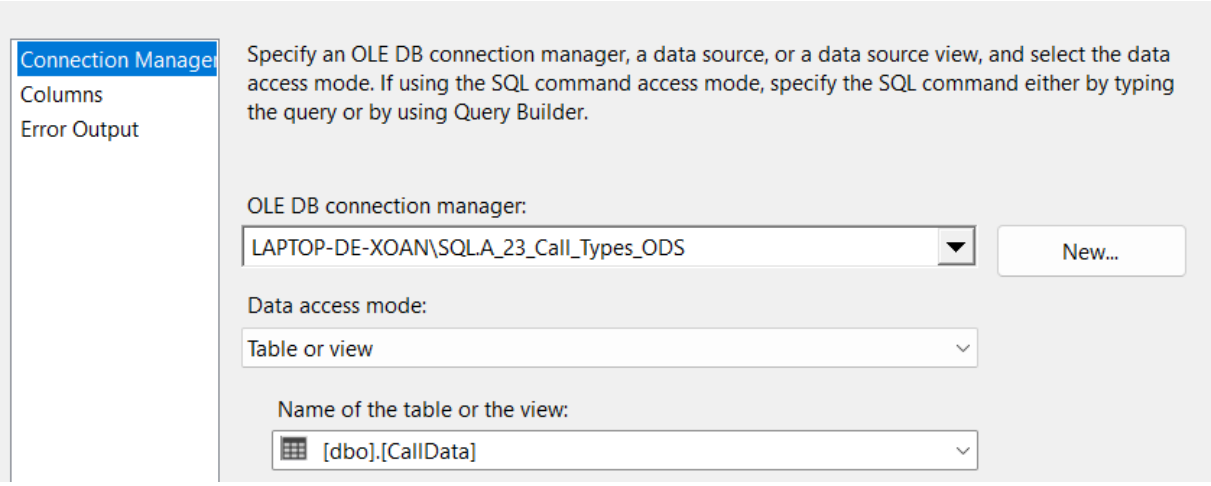
4.5. Integration of the Call Data Facts table

Now that we finally have our dimensions tables, we can build our fact table while checking valid relations with the dimensions.

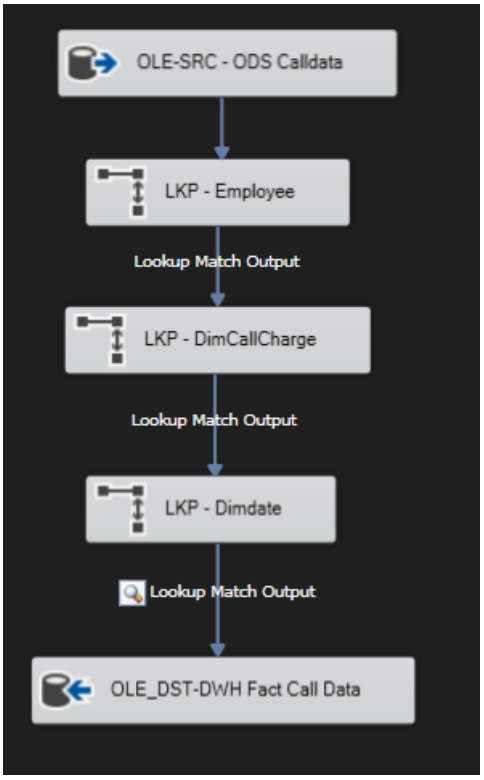
We applied Truncate for the DWH Calldata table for solve the duplication data issue.



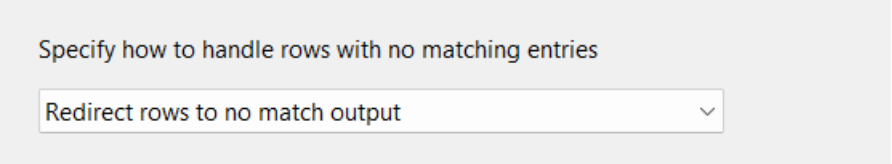
We created the OLE source from ODS Calldata, and we connect with other dimensions table via the Lookup functions. Below is the connection in OLE Source which we connect data from [dbo].[Calldata].

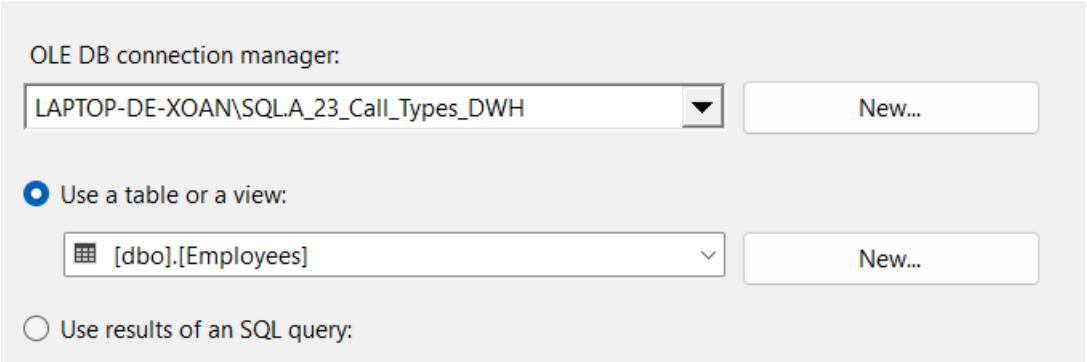


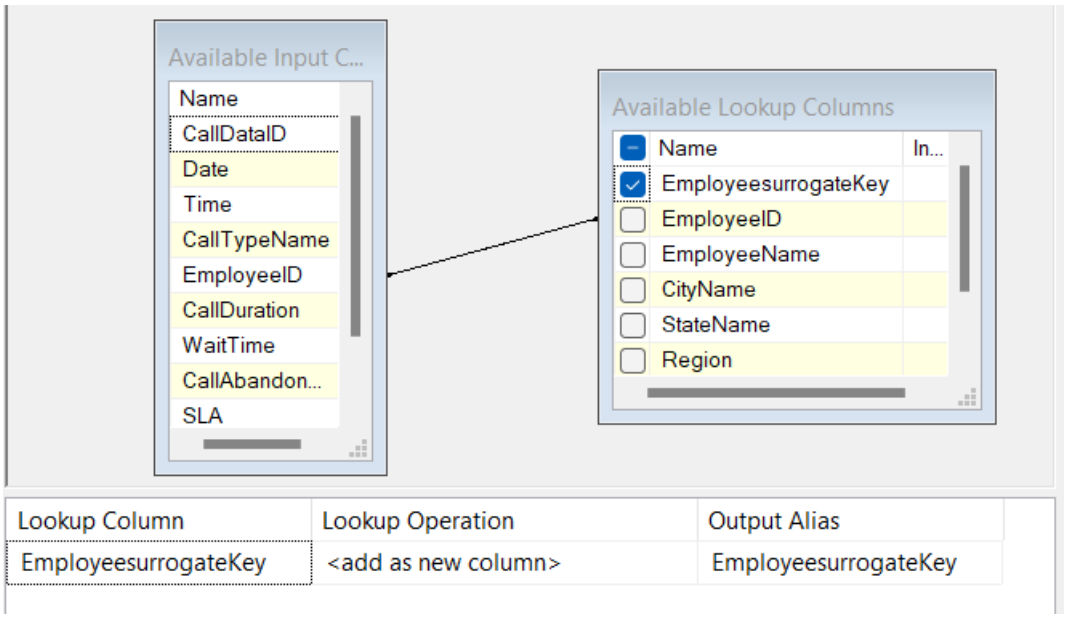
For connecting the data with other dimension tables, we used Lookup functions and connected follow the process of the picture below.



Firstly, we made the connection between DimEmployee table with the FactTable. For case no matching entries, we choose “Redirect rows to no match output”. Then, we connected with the data [dbo].[Employees] in DWH connection. We took EmployeeID in Employee table connect with EmployeeID in FactCall table. From that, we looked up column EmployeesurrogateKey. The process as the pictures below:

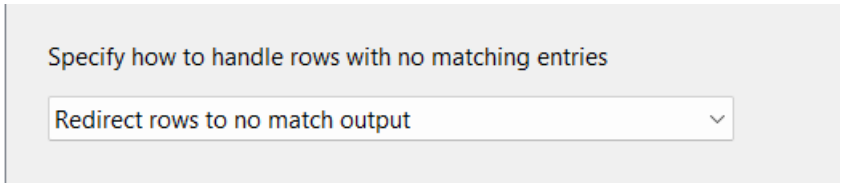


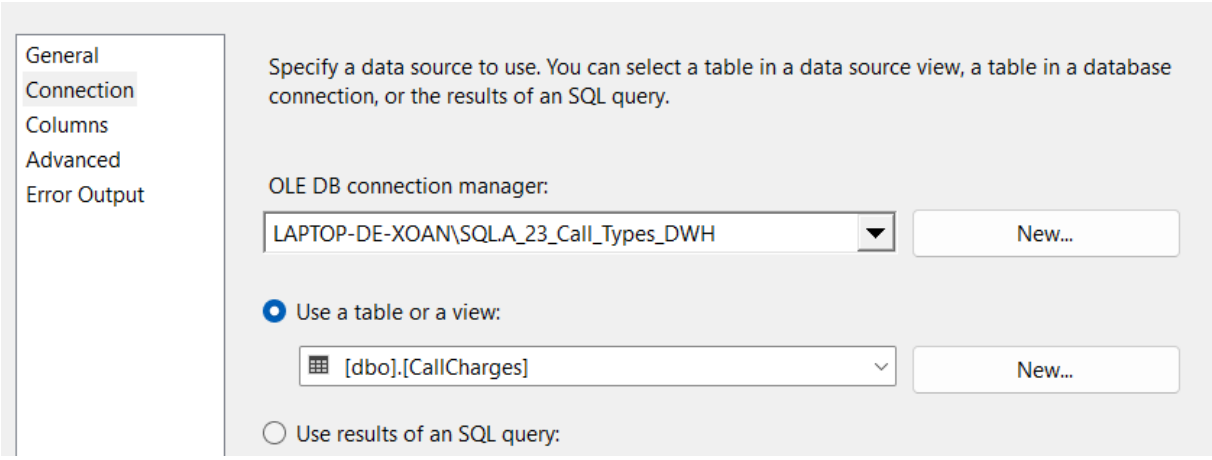


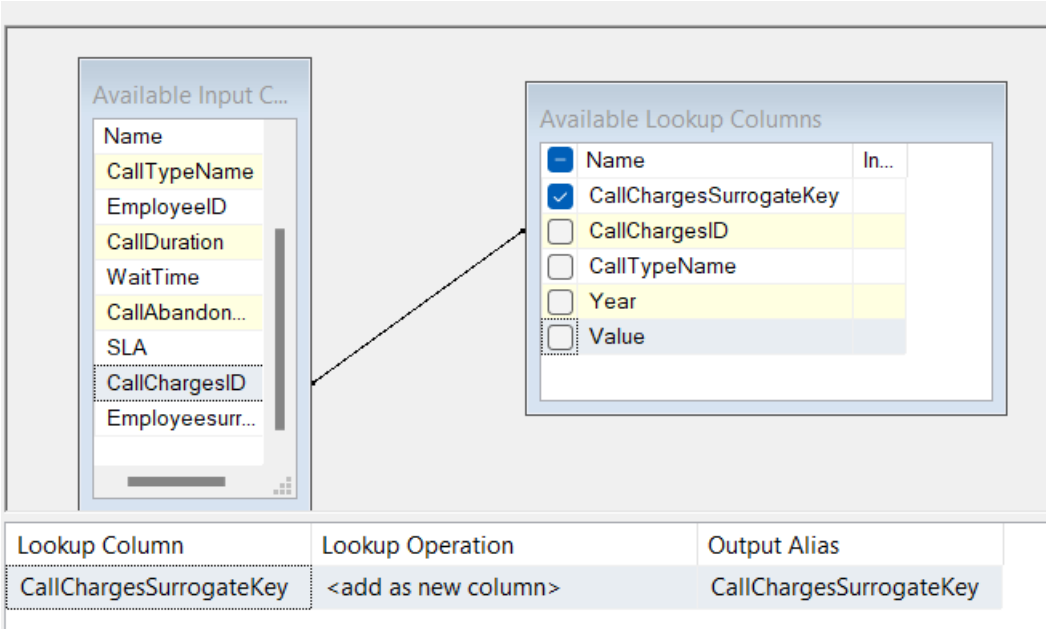


Secondly, we do the same process with the DimCallCharge and Dimdate. The detail of them as below:

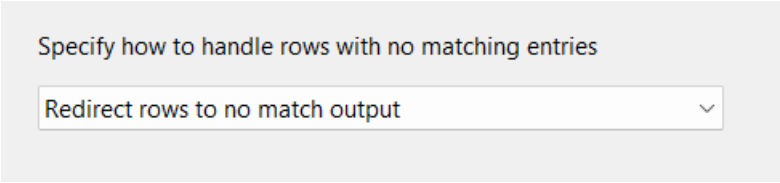
- DimCallCharge

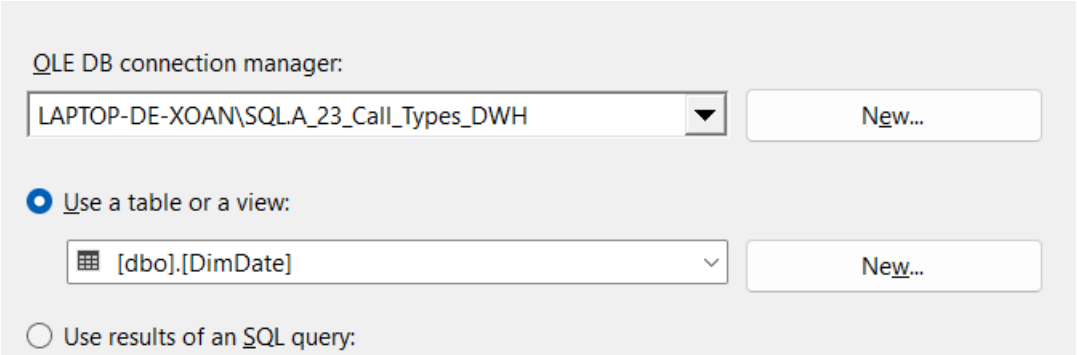


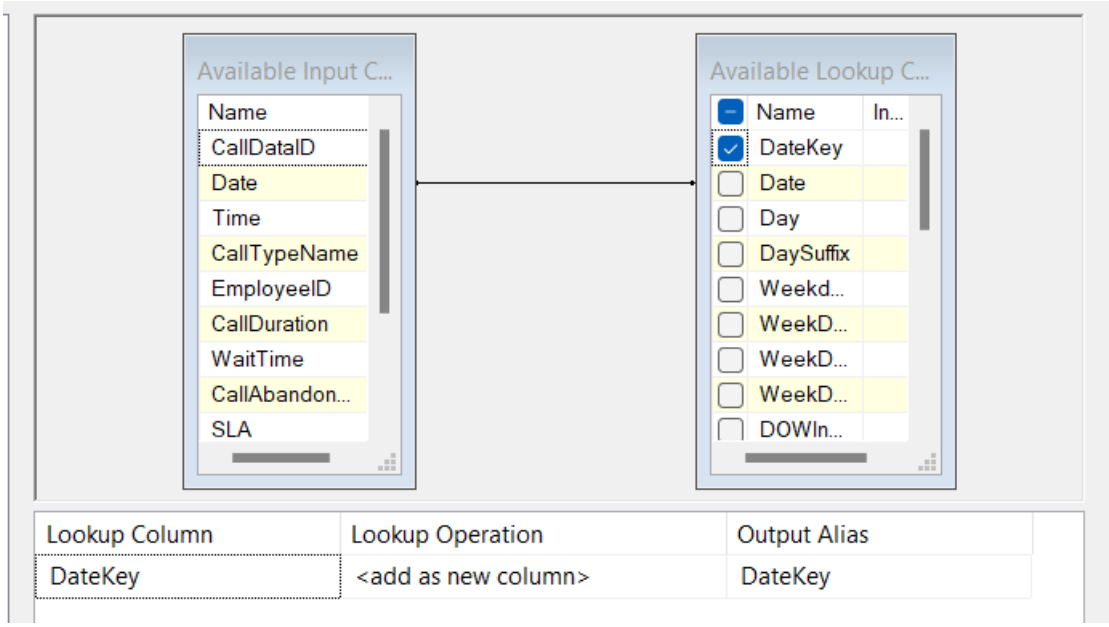



Available Input C...
Name
CallTypeName
EmployeeID
CallDuration
WaitTime
CallAbandon...
SLA
CallChargesID
Employeesurr...
Available Lookup Columns
Name
In...
CallChargesSurrogateKey
CallChargesID
CallTypeName
Year
Value
Lookup Column
Lookup Operation
Output Alias
CallChargesSurrogateKey
<add as new column>
CallChargesSurrogateKey

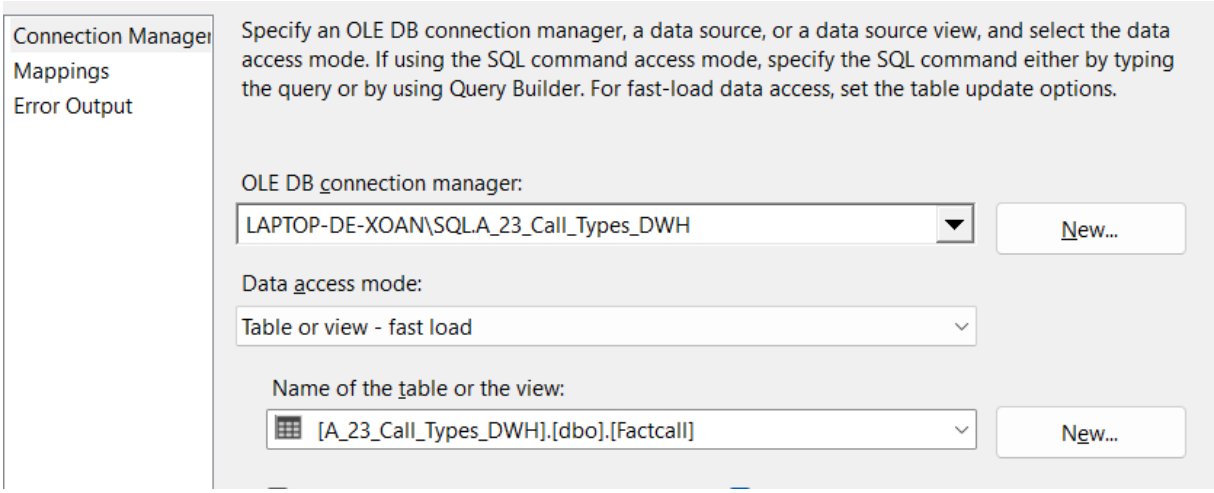
- Dimdate


Specify how to handle rows with no matching entries
Redirect rows to no match output

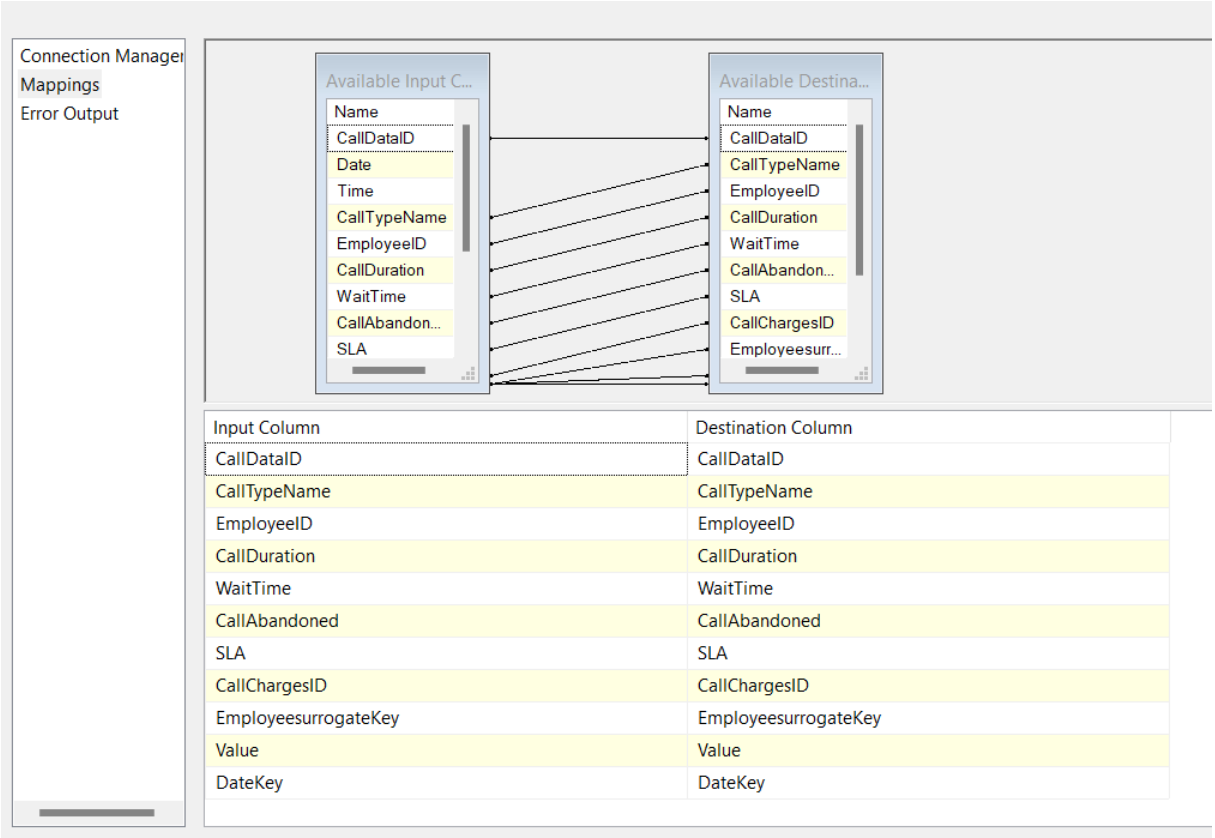

OLE DB connection manager:
LAPTOP-DE-XOAN\SQL.A_23_Call_Types_DWH
New...
Use a table or a view:
[dbo].[DimDate]
New...
Use results of an SQL query:



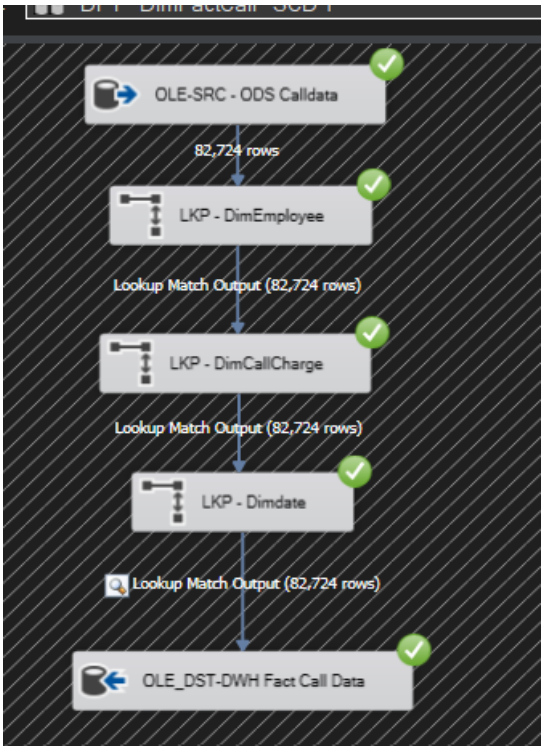
Lastly, we created the OLE-DST for FactCall data in DWH. We created the FactCall Table. The code created this table as below:



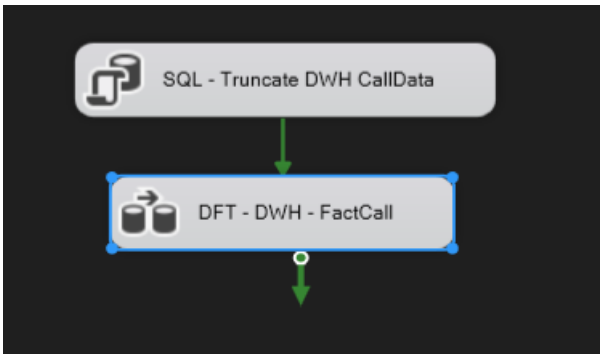
We mapped the column based on the picture below:

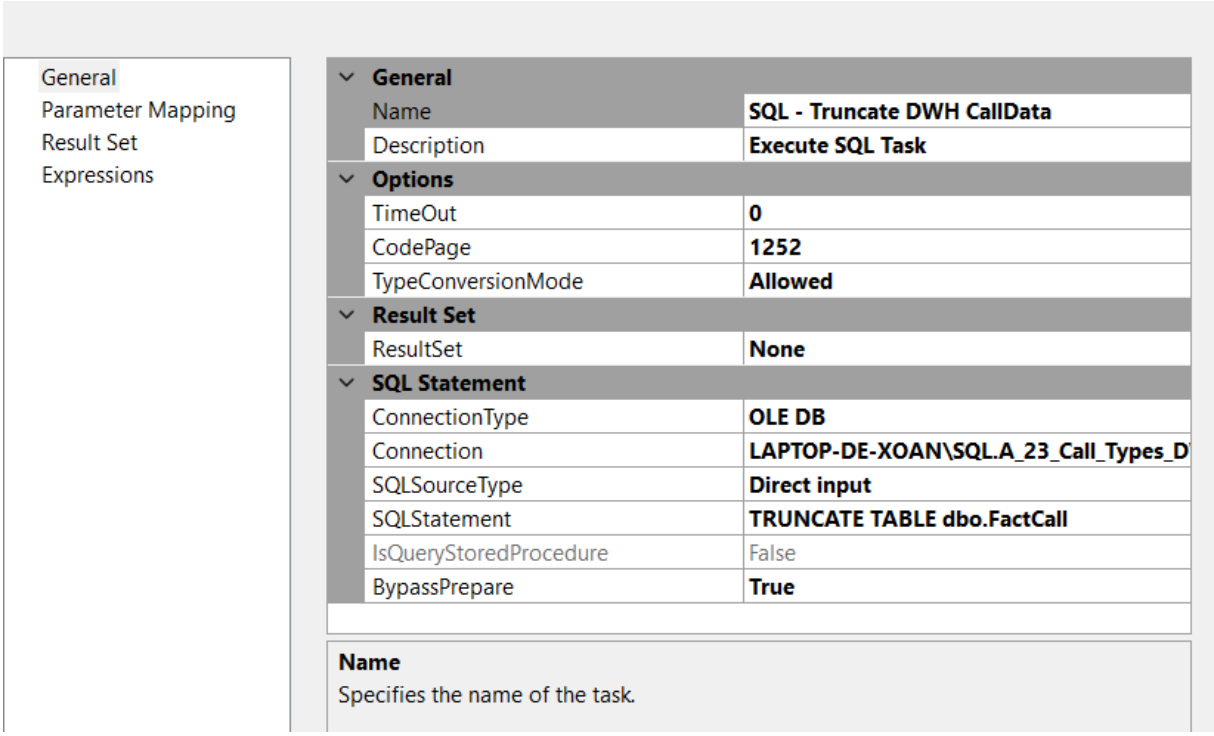


After that, we pressed on the “Start” which can help us review and there is any problem in our process. Below is the result of the processing, we do not miss any rows on our data, which are 82724 rows.



In order to avoid any duplicating risk, we applied Truncate for this table.





We check the result on SQL Server Management as below”

QLQuery2.sql - L...RUNG NGUYEN (71))* | SQLQuery1.sql - L...RUNG NGUYEN (51))

```sql
SELECT TOP (1000) [CallDataID]
      ,[CallTypeName]
      ,[DateKey]
      ,[EmployeeID]
      ,[CallDuration]
      ,[WaitTime]
      ,[CallAbandoned]
      ,[SLA]
      ,[CallChargesID]
      ,[EmployeesurrogateKey] as EmployeesurKey
      ,[Value]
  FROM [A_23_Call_Types_DWH].[dbo].[Factcall]
```

0 %

Results | Messages

	CallDataID	CallTypeName	DateKey	EmployeeID	CallDuration	WaitTime	CallAbandoned	SLA	CallChargesID	EmployeesurKey	Value
	1	Tech Support	20180405	U559430	486	2	0	Within SLA	TECHSUPPORT2018	45	0.95
	2	Tech Support	20180621	A166733	945	0	0	Within SLA	TECHSUPPORT2018	38	0.95
	3	Sales	20180621	B971624	379	11	0	Within SLA	SALES2018	19	1.52
	4	Tech Support	20181121	U641256	1044	0	0	Within SLA	TECHSUPPORT2018	36	0.95
	5	Sales	20180225	P286634	1357	0	0	Within SLA	SALES2018	15	1.52
	6	Tech Support	20181028	M855788	570	23	0	Within SLA	TECHSUPPORT2018	24	0.95
	7	Tech Support	20181217	M794992	26	26	0	Within SLA	TECHSUPPORT2018	33	0.95
	8	Billing	20180728	I281837	8	8	0	Within SLA	BILLING2018	4	1.20
	9	Sales	20180611	J192194	800	0	0	Within SLA	SALES2018	54	1.52
0	10	Billing	20180618	F542348	651	111	0	Outside SLA	BILLING2018	61	1.20
1	11	Billing	20180514	J632516	229	10	0	Within SLA	BILLING2018	6	1.20
2	12	Tech Support	20180206	U194381	467	23	0	Within SLA	TECHSUPPORT2018	22	0.95
3	13	Tech Support	20180620	G586239	1290	7	0	Within SLA	TECHSUPPORT2018	11	0.95
4	14	Tech Support	20181220	N772493	504	0	0	Within SLA	TECHSUPPORT2018	1	0.95
5	15	Tech Support	20181215	J632516	1445	28	0	Within SLA	TECHSUPPORT2018	6	0.95
6	16	Billing	20180403	K669566	1255	2	0	Within SLA	BILLING2018	44	1.20
7	17	Tech Support	20181223	E137708	1069	5	0	Within SLA	TECHSUPPORT2018	40	0.95
8	18	Tech Support	20180417	M855788	758	0	0	Within SLA	TECHSUPPORT2018	24	0.95
9	19	Billing	20181102	E778362	838	0	0	Within SLA	BILLING2018	62	1.20
0	20	Tech Support	20181122	N119221	681	10	0	Within SLA	TECHSUPPORT2018	23	0.95
1	21	Billing	20180408	V228277	1272	42	0	Outside SLA	BILLING2018	52	1.20

Query executed successfully. | LAPTOP-DE-XOAN\SQL (16.0 RTM) | L

5. Conclusion

Based on the initial database, we have several data tables which include differential data and if there is any update, it would be difficult to connect and update those data. Thanks to the integration of SSIS and SQL, we had created the FactCall Table which can update data for years to come and include almost the important information within only one table.

IN STA stage:

We created the STA databases and loaded the files in their raw format. However, as the Call Data files are 3 by year, we used a Foreach Loop Container to load the files into the database. This ensures that the files are all loaded sequentially, and any subsequent files too can be loaded for years to come.

In the ODS step:

From the initial table, we do lookup to combine the data from the table Employee and USState. From that, we have a table named “Employee” with information about the geography. For 3 tables of CallData, we fused, split call time stamp to stamp to Date and Time, and n SLA Column. For Calltype and Call Charge, we did look up and unpivoted the data into 2 columns, from that, we have Call Charge with value based on Calltype and Year.

In the DWH Step:

We created some dimension tables which help us connect data into the FactCall table. We created DimEmployee Table, Dimdate and DimCall Charge. From that, we connected to the FactCall data and applied Truncate to avoid any duplication in our data.