

# *Group members:*

# Trung Nguyen

# Phuc Nguyen Pham

# Mohammed Danish Mustafa

# Patrick Warui Githendu

Paris, 19/05/2024

SSIS & SQL Server project report

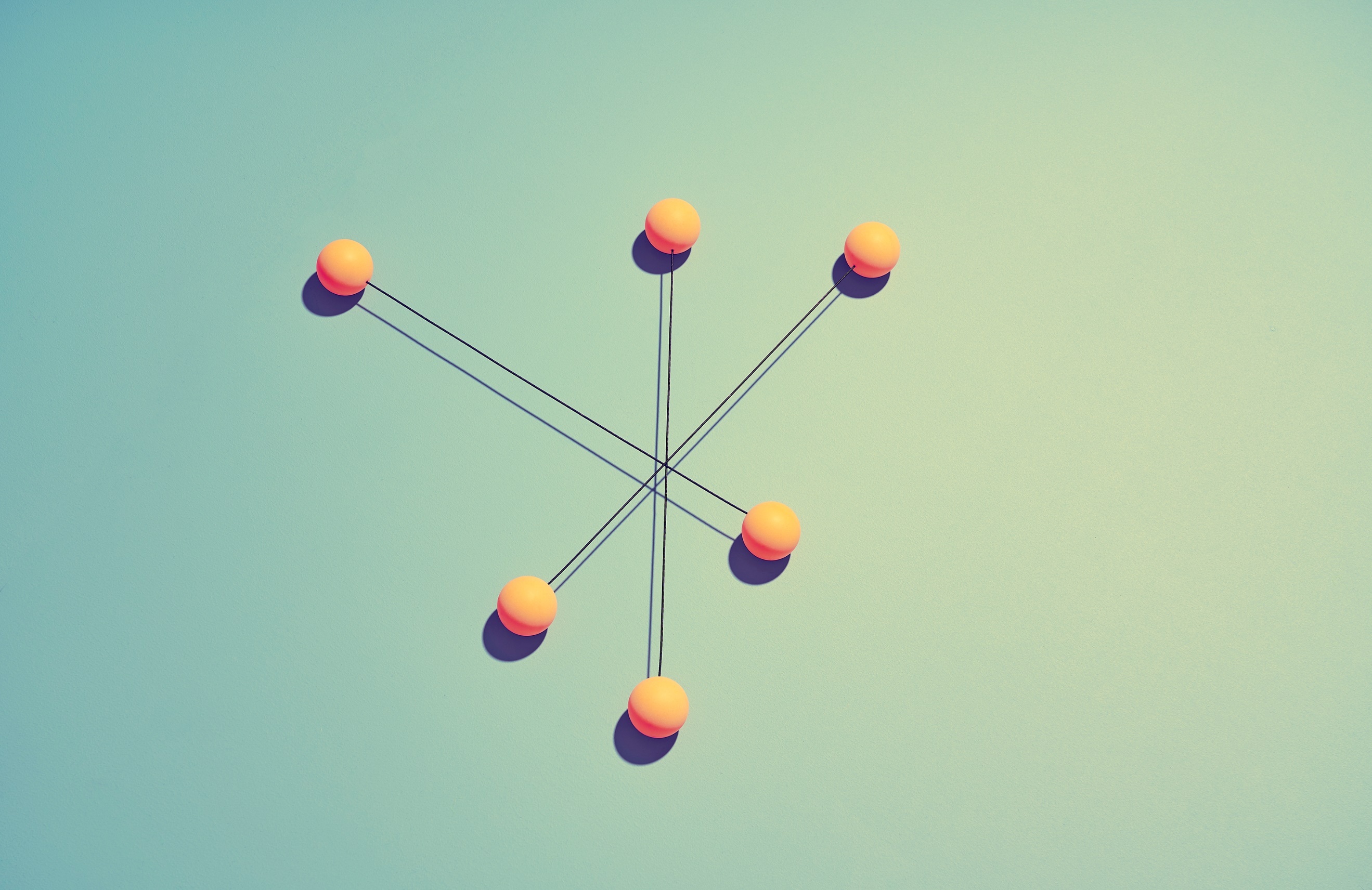


Table of Content

[1. Introduction 2](#_Toc180933361)

[1.1. Data 2](#_Toc1785022748)

[1.2. Pipeline design 3](#_Toc180692845)

[1.3. Model 3](#_Toc556422623)

[2. Staging 4](#_Toc662521316)

[2.1 Employees Table 4](#_Toc1827241061)

[2.2 Call Data Table 6](#_Toc1532918995)

[2.3 Call Types Table 11](#_Toc944623819)

[2.4 US States Table 11](#_Toc409860195)

[2.5 Call Charges Table 12](#_Toc1686255101)

[3. Operational Data Store 13](#_Toc1430410029)

[3.1. Employees Table 13](#_Toc1260597713)

[3.2. Data Charges Table 16](#_Toc1961509983)

[3.3. Data Call Table 22](#_Toc1572798408)

[4. Datawarehouse 30](#_Toc734232505)

[4.1. Database design 30](#_Toc222241487)

[4.2. Integration of the Date dimension 31](#_Toc1410179778)

[4.3. Integration of the Employees dimension 34](#_Toc1201319633)

[4.4. Integration of the Call charge dimension 36](#_Toc176518784)

[4.5. Integration of the Call Data Facts table 39](#_Toc46451461)

[5. Conclusion 47](#_Toc164229389)

# 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.

## 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:

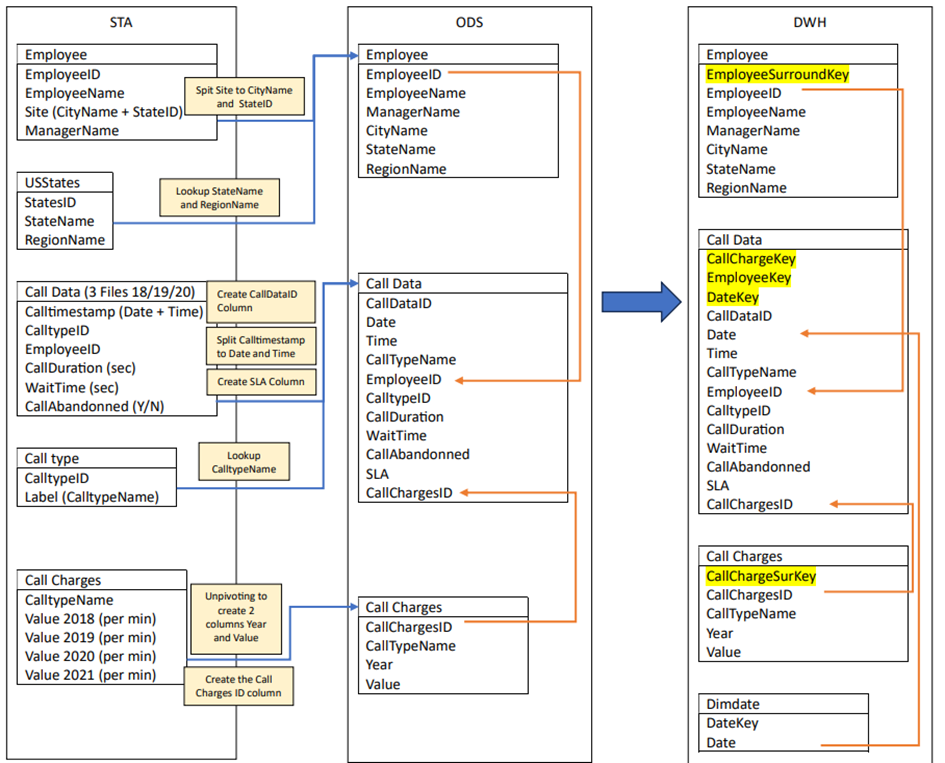
Staging Area (STA): This stage involves loading the data as is or with minimal changes.

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:



# 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.

1. Create Database A\_23\_Call\_Types\_STA
2. 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

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

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.

A screenshot of a computer

Description automatically generated

The Employees Database looks as follows

A screenshot of a computer

Description automatically generated

## 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.

A screenshot of a computer

Description automatically generated

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.

A screenshot of a computer

Description automatically generated

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

A screenshot of a computer

Description automatically generated

Then -> Expression

A screenshot of a computer

Description automatically generated

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. A screenshot of a computer

Description automatically generated

This is the STA\_CallData.dtsx. data flowA screenshot of a computer

Description automatically generated

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

A screenshot of a computer

Description automatically generated

**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.**

A screenshot of a computer screen

Description automatically generated

## 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.

A screenshot of a computer

Description automatically generated

## 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

A screenshot of a computer

Description automatically generated

## 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

A screenshot of a computer

Description automatically generated

The above are the 5 dbo. Tables in the Database A\_23\_Call\_Types\_STA

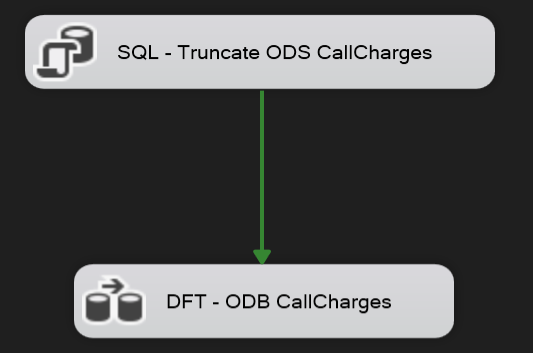
# Operational Data Store

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.

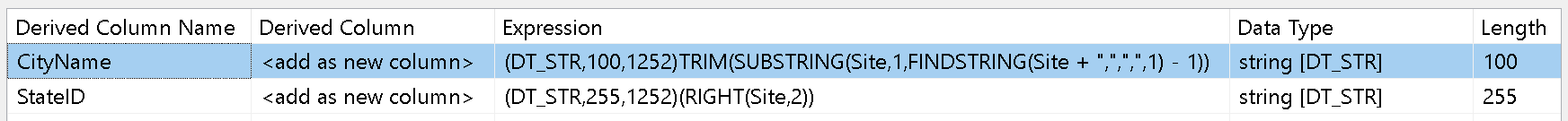


Next, this is the data flow task to make the Employee table in this ODS stage, the process will comprise five steps:

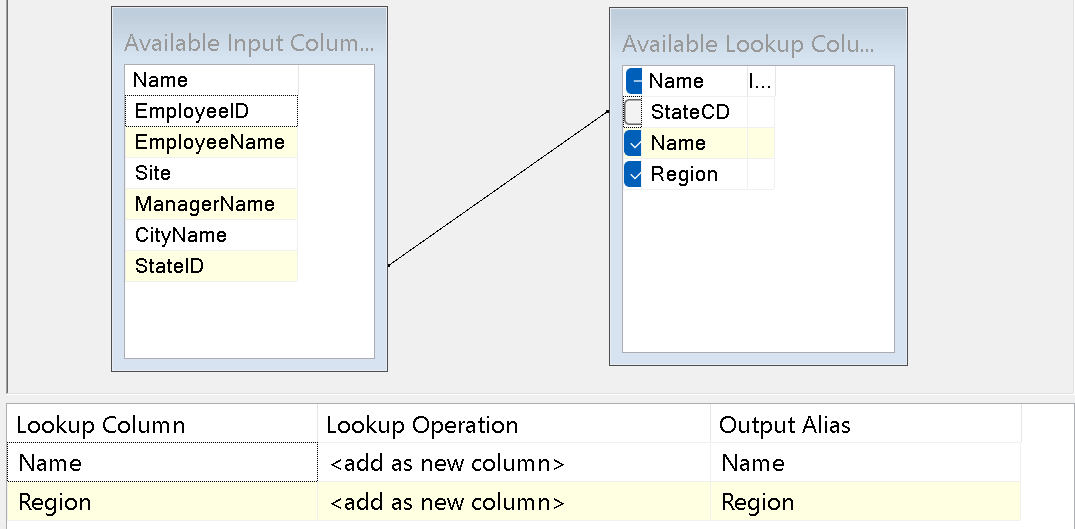
1. Import data from the STA Employees table.
2. Split the values in the Site column into two separate values to create the CityName and StateID columns.
3. Perform a lookup with the STA USStates table to add the StateName and RegionName columns.
4. Resize the length of the values in the columns to ensure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



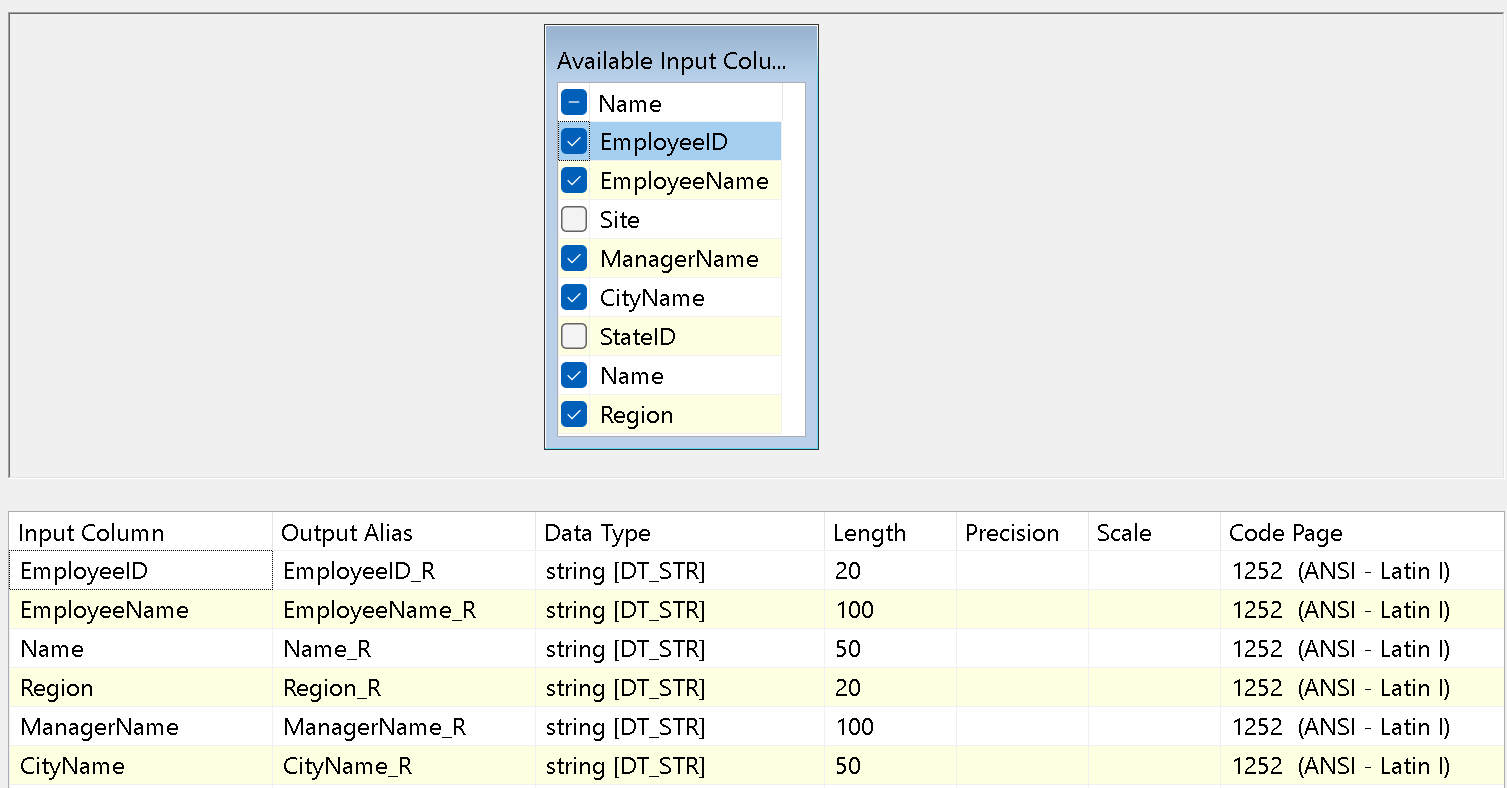
In the second step, we found out that the Site column contains the information of the city name and the StateID, so we decided to split this column into 2 column CityName and StateID. We do it by using the Deriver column in SSIS toolbox. Here is the expression for each column:



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



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.



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:

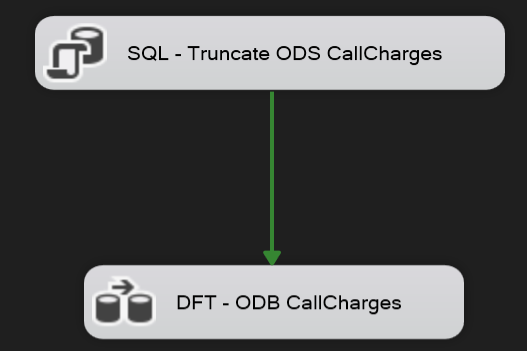


In the ODS Employees table, we got the result like this:



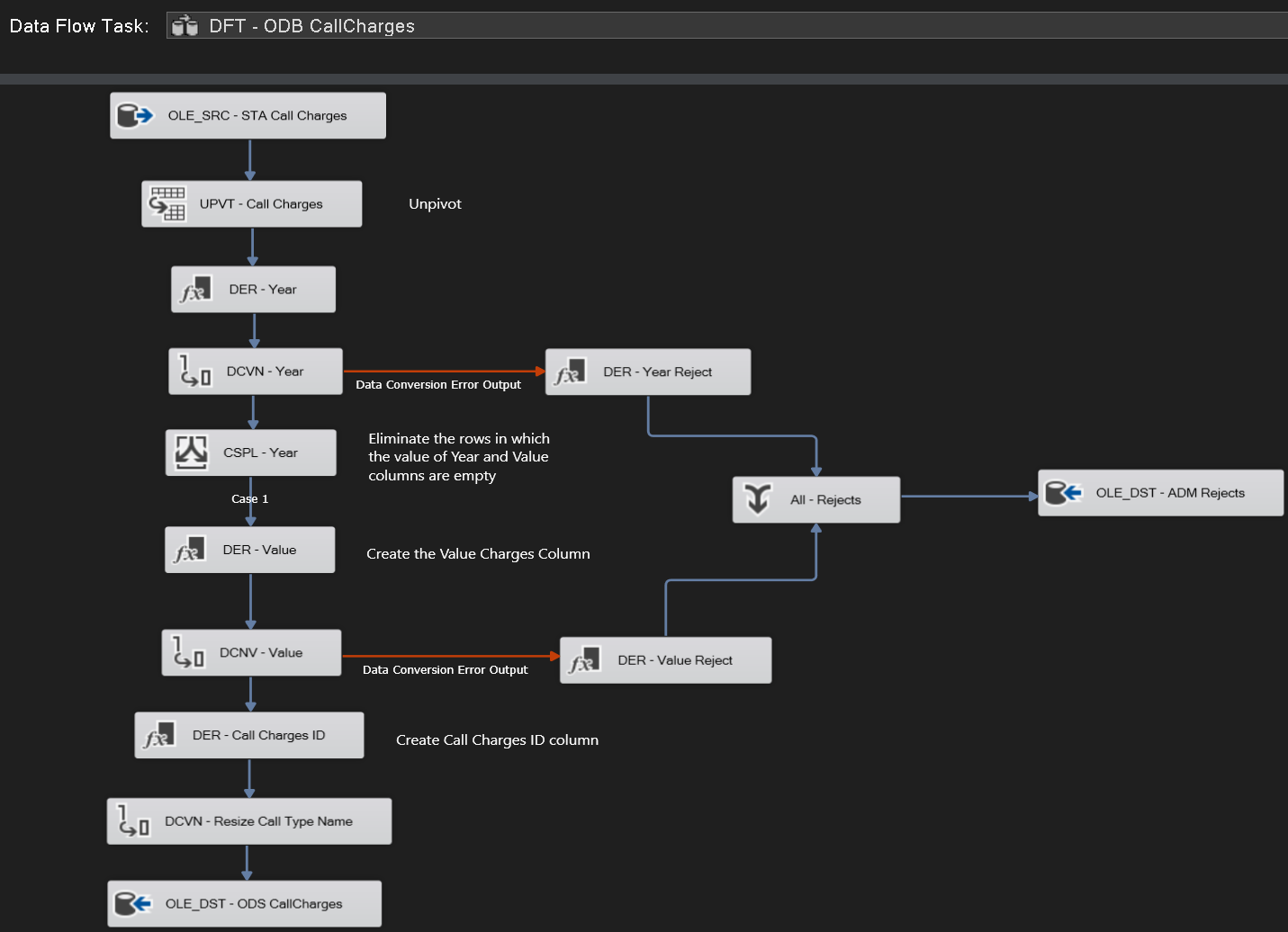
## 3.2. Call Charges Table

The first step is always to truncate the table:

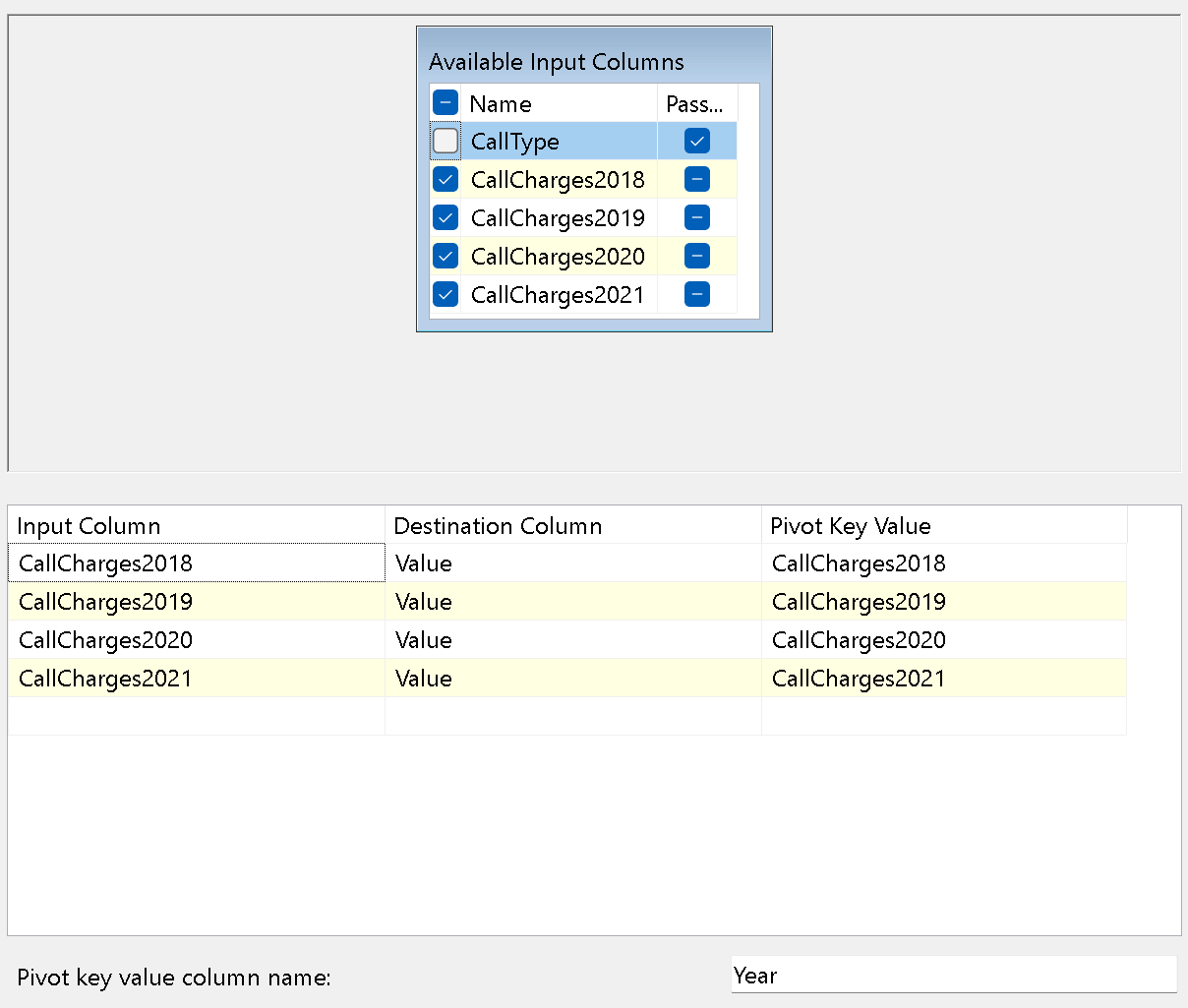


Then, we created the data flow. The process encompasses a total of 10 steps (excluding those related to handling or rejecting erroneous data):

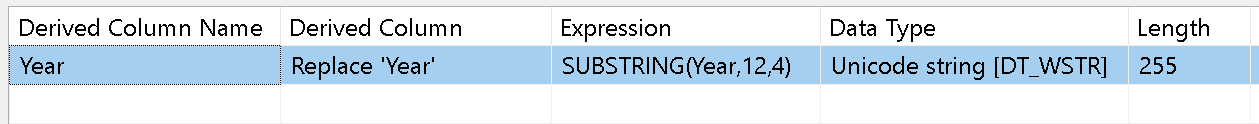
1. Extract data from the STA Call Charges table.
2. Unpivot the Call Charges table to create two columns: Year and Charge Value.
3. Extract and retain only the year data (e.g., extracting "2018" from "CallCharges2018").
4. Transform the Year column to a string type with a length of 4 characters to facilitate 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.



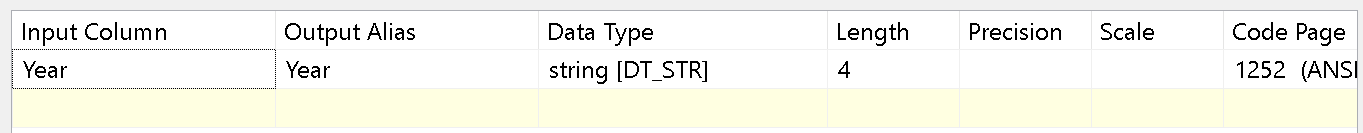
In the unpivoting step, here is our configuration:



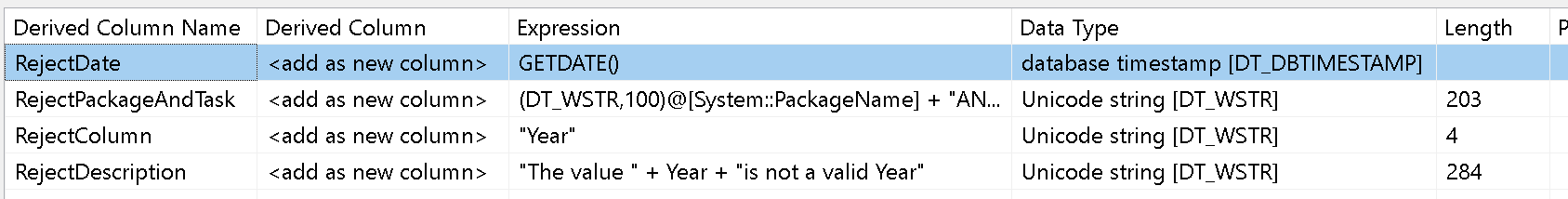
Then, we extracted the year:



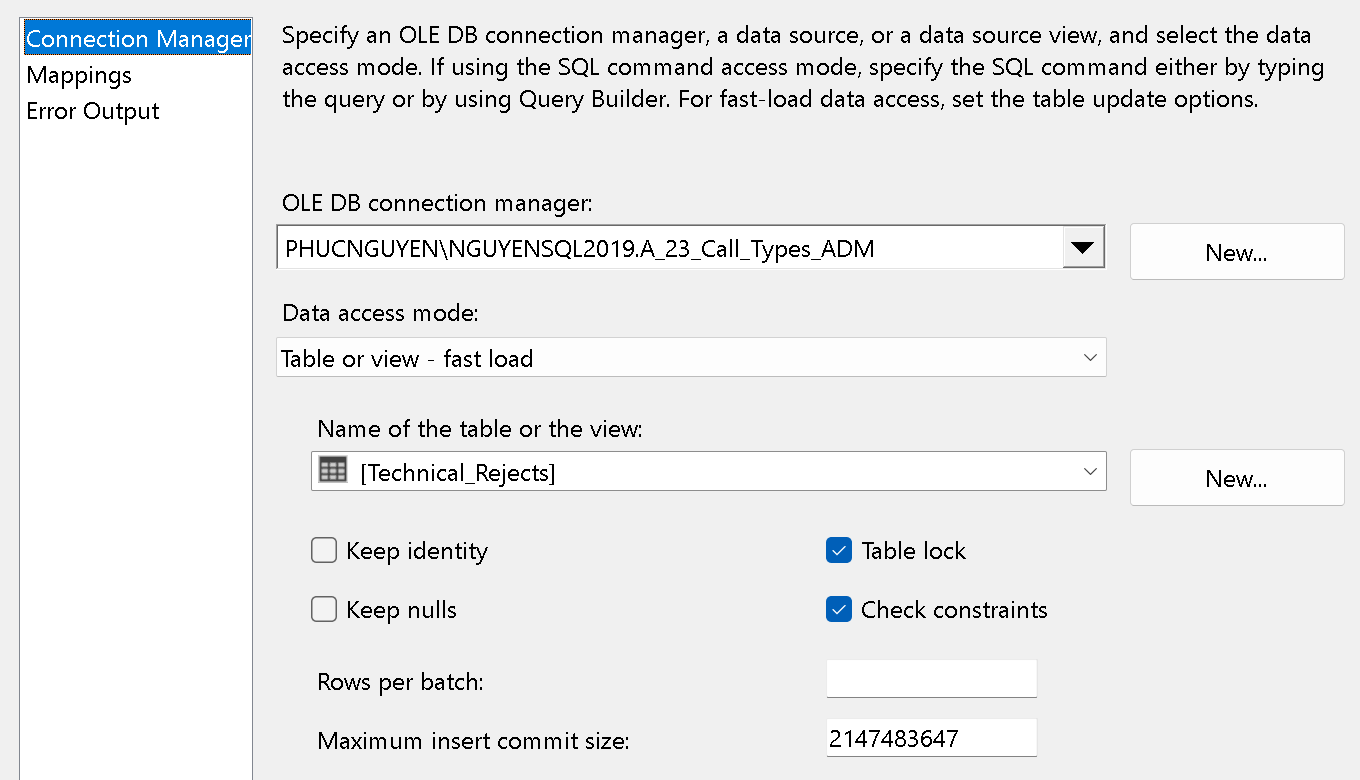
In the 4th step, its type is also changed:



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



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:

CREATE TABLE Technical\_Rejects (

[RejectDate] datetime,

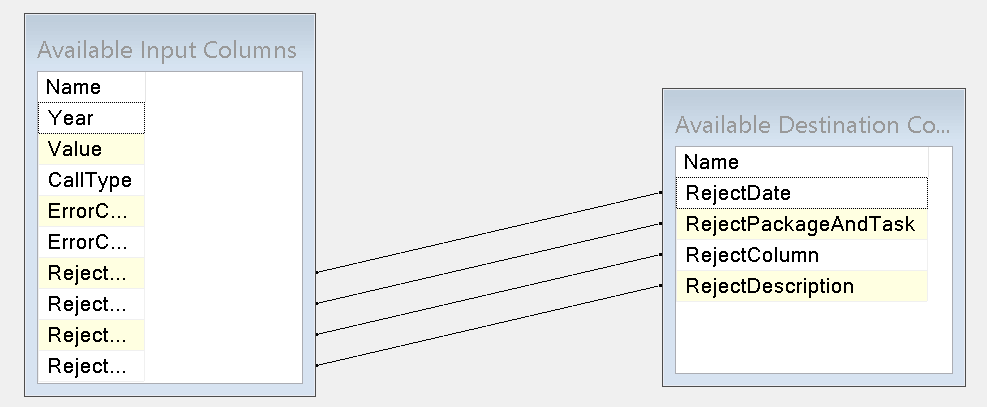
[RejectPackageAndTask] nvarchar(203),

[RejectColumn] nvarchar(50),

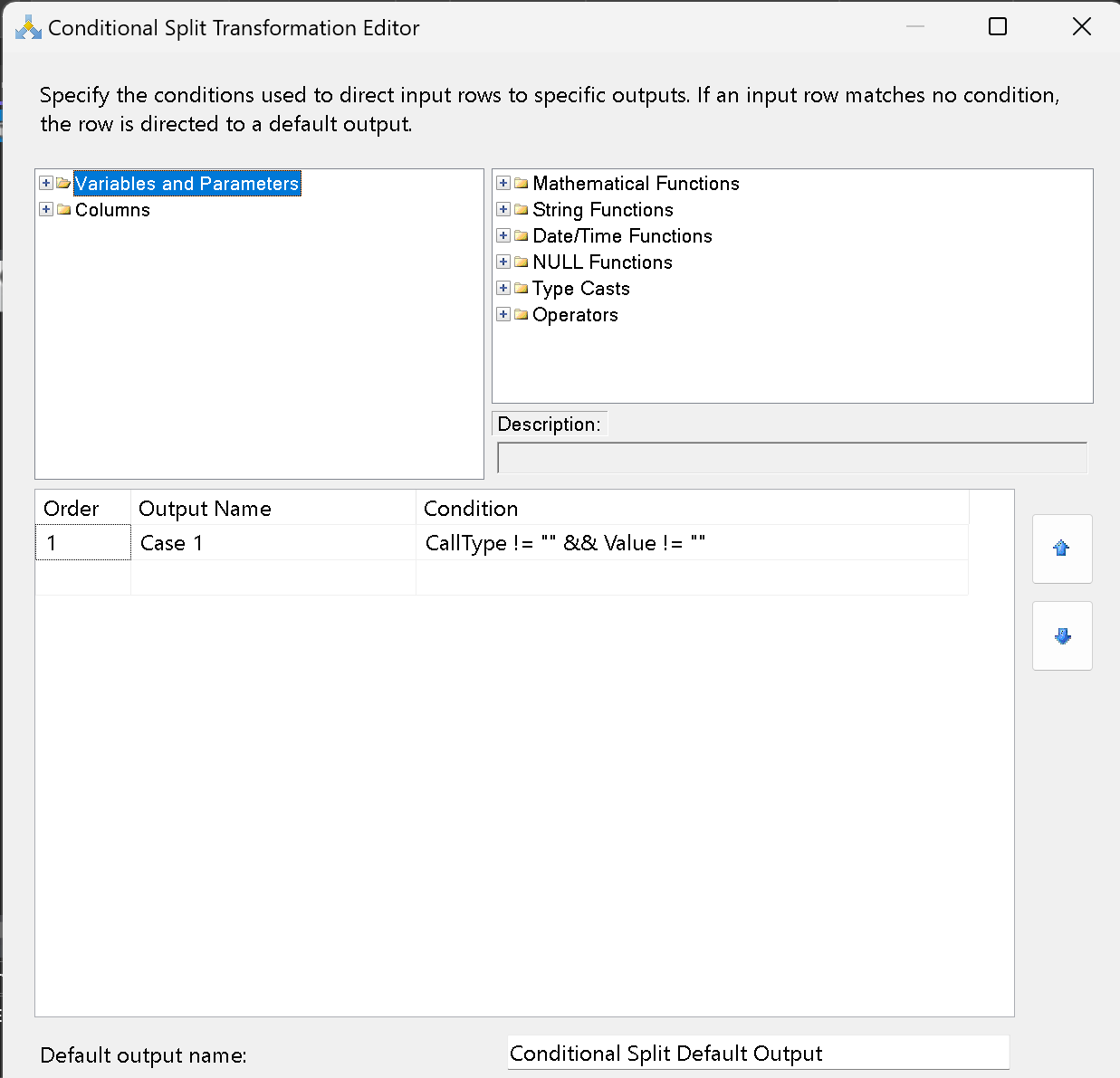
[RejectDescription] nvarchar(287)

)

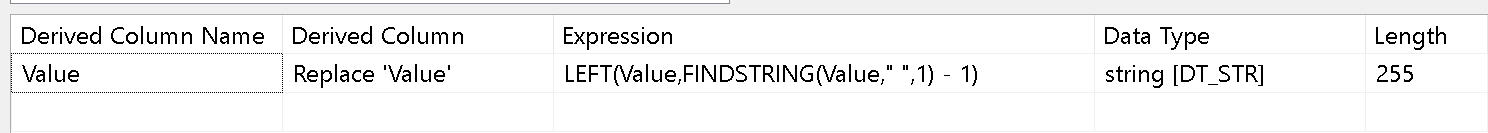
The mapping for the ADM database:



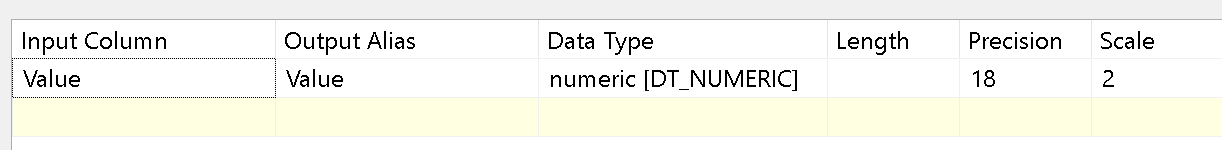
Then, we found a lot of empty values for the Year and Value columns because there were empty rows from the data in STA which from the data in STA which is still not cleaned yet. So, we decided to eliminate those rows by using Conditional Split Transformation Editor. Here is the configuration:



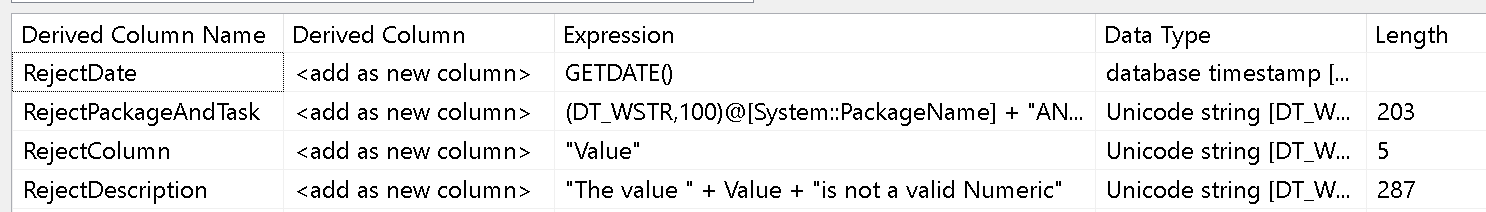
In the 6th step, we also extracted the value from the Value column, so we can have standardized rows with only the number instead of number and text (e.g., extracting "1.52" from "1.52 / min"):



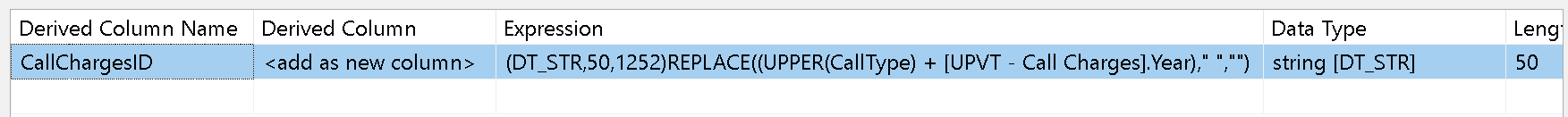
Next step, we converted it to numeric type, making the scale at 2:



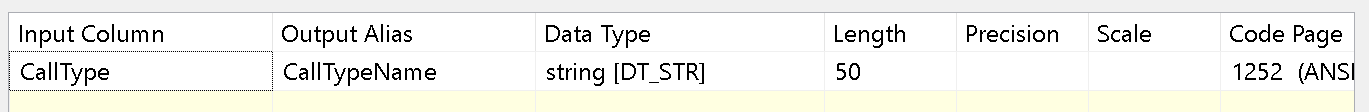
In case of error values, we created a flow to put them in the ADM database:



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:



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



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),

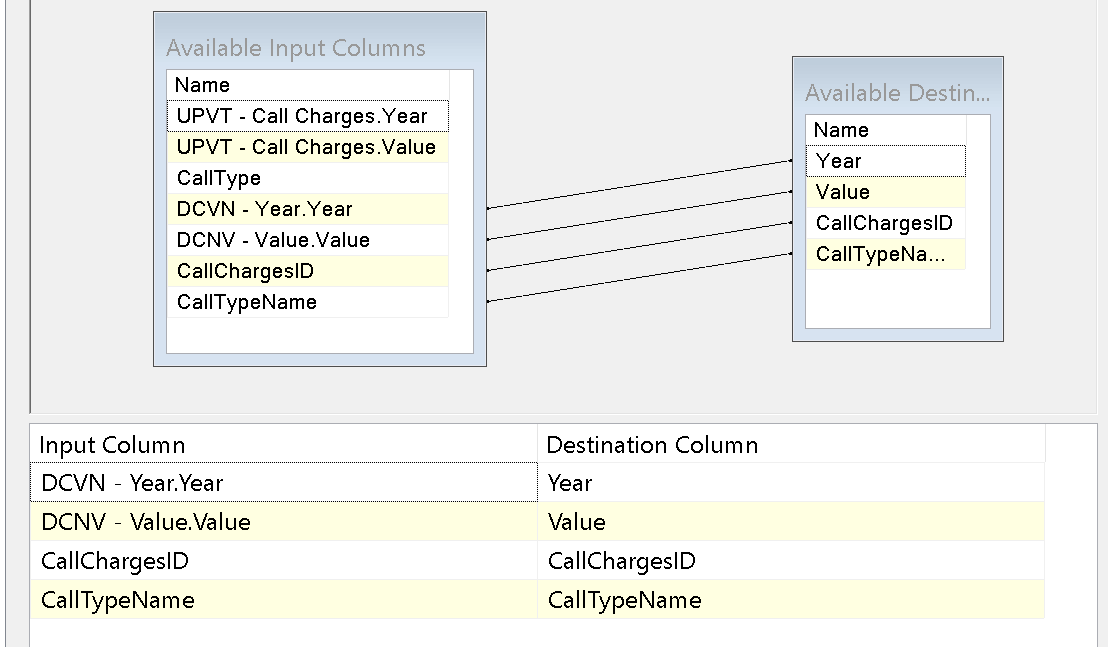
[CallTypeName] varchar(50),

[Year] varchar(4),

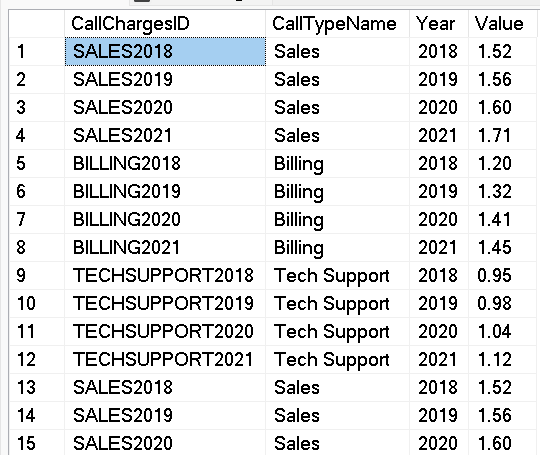
[Value] numeric(18,2)

)

The mapping is described as below:

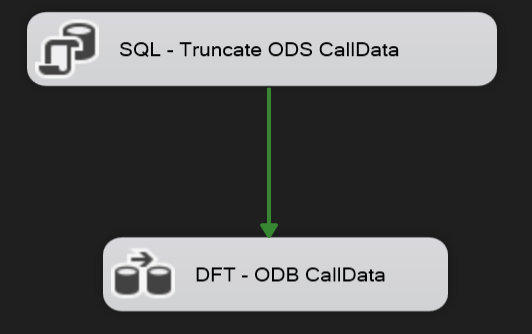


This is 10 first lines of the ODS Call Charges Table:



## 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):

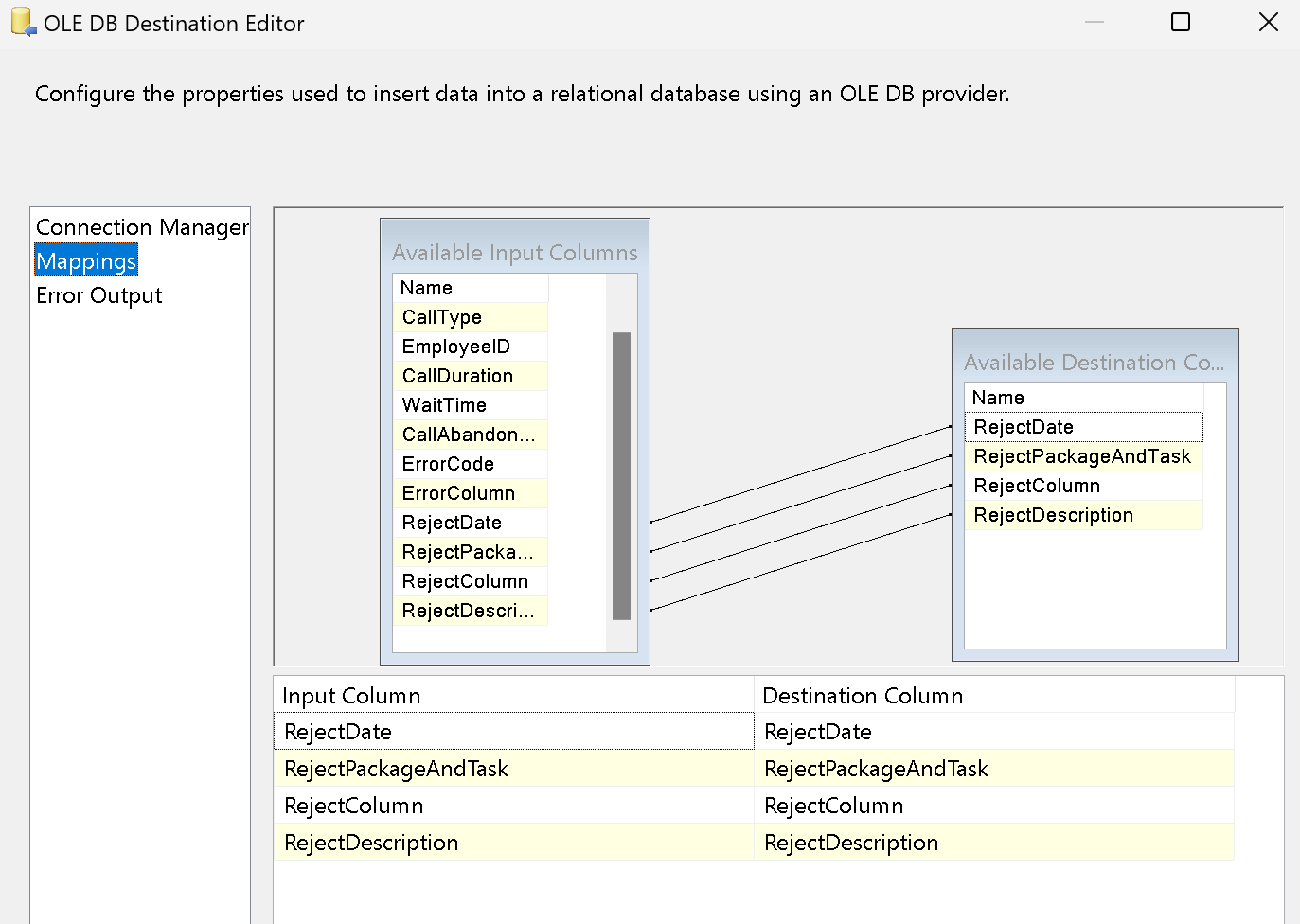
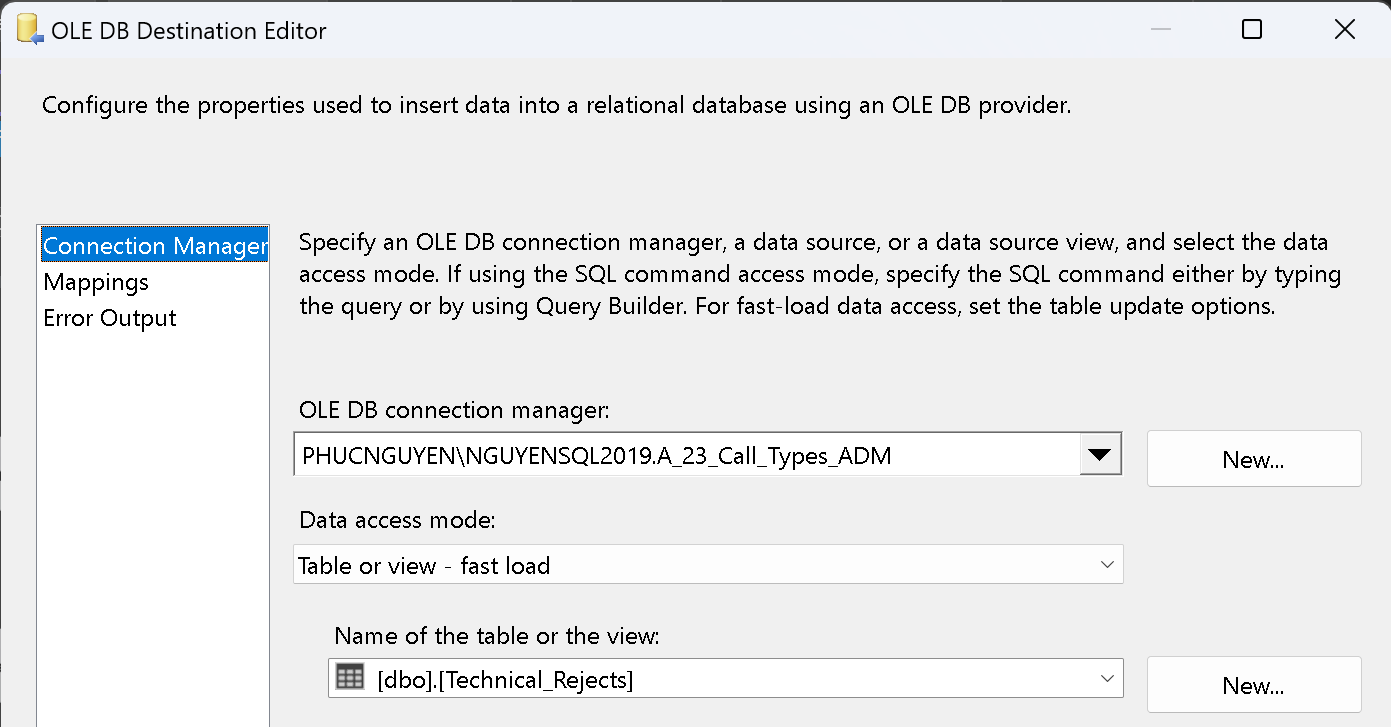
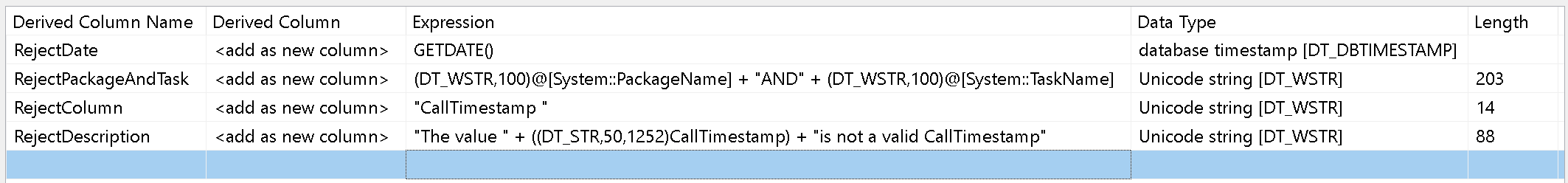
1. Input data from the STA Call Data table.
2. Split the Timestamp column into two separate columns: Date and Time.
3. Change the type of the Call Type column (serving as Call Type ID) to do a lookup at the next step.
4. Perform a lookup with the STA Call Type table to create a new column, Call Type 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.



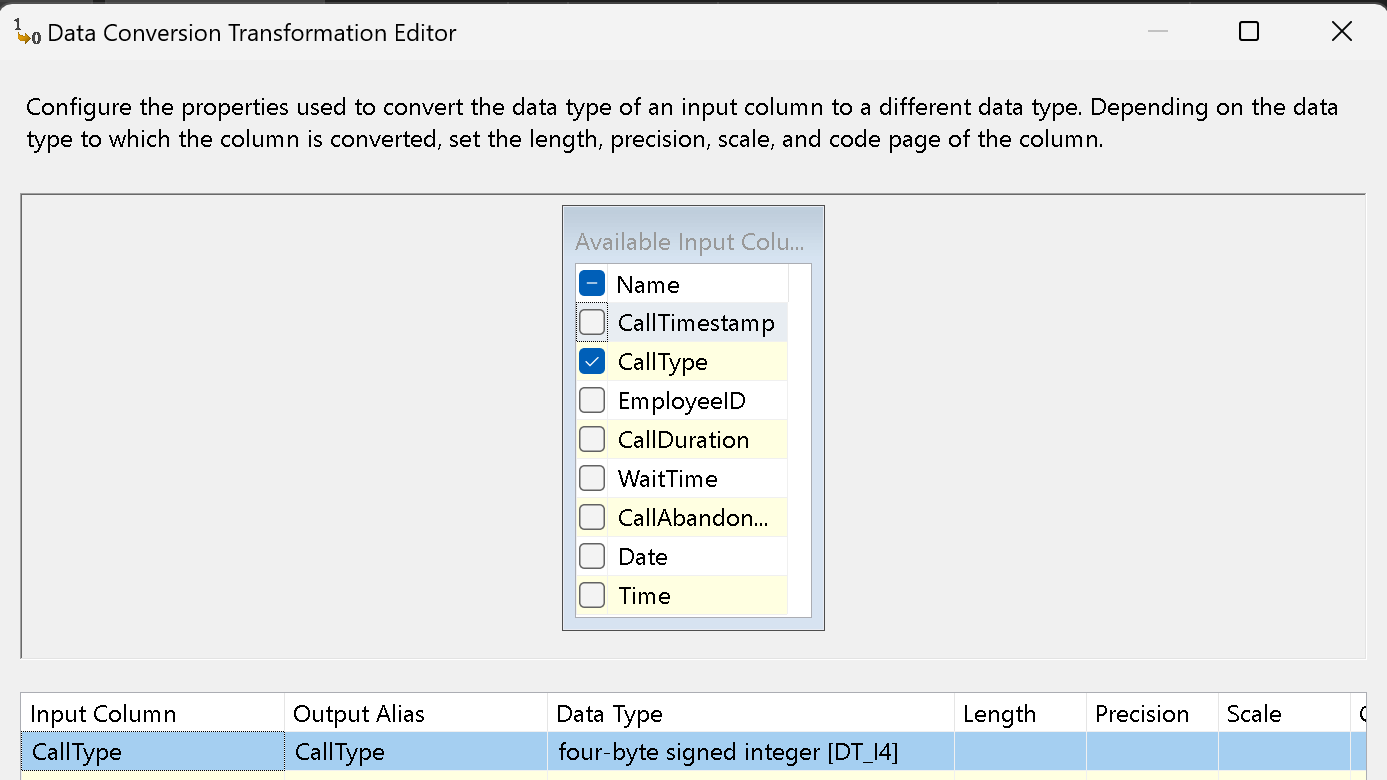
To split the timestamp into 2 columns, here is the expression for the derived column:

|  |  |
| --- | --- |
| 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)) |

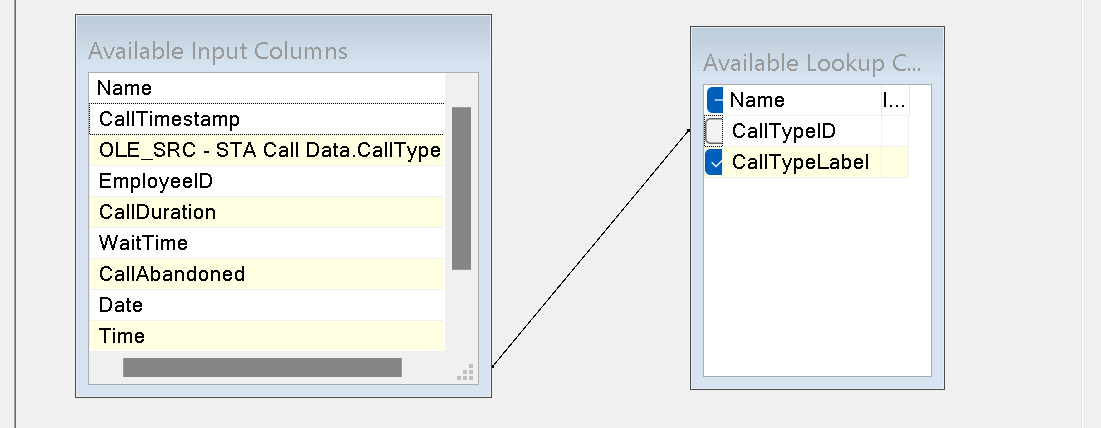
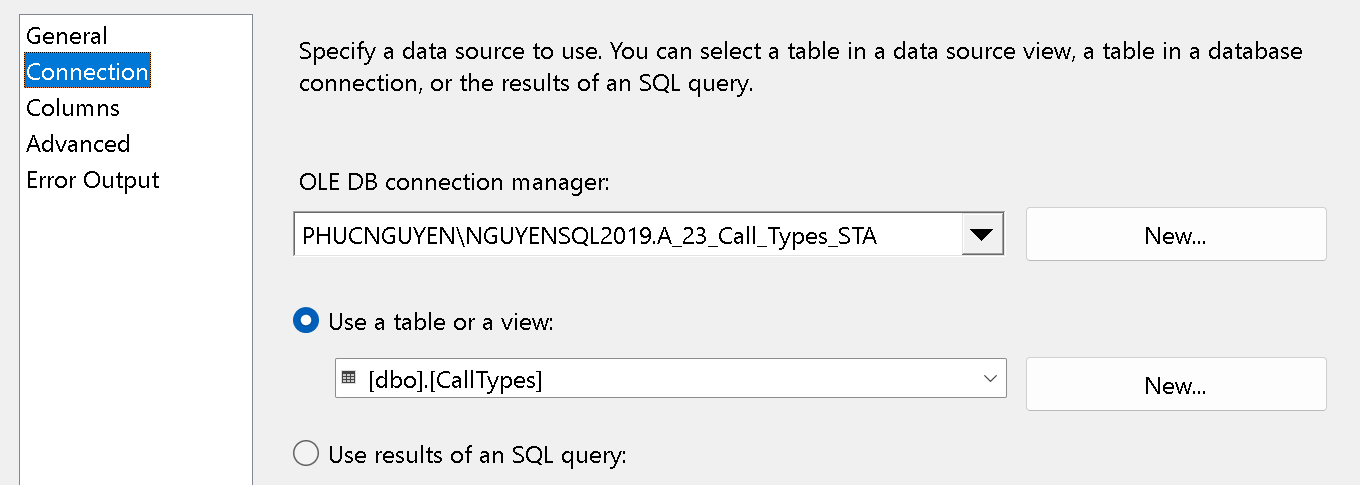
This is the expression in case there are errors for the Timestamp:



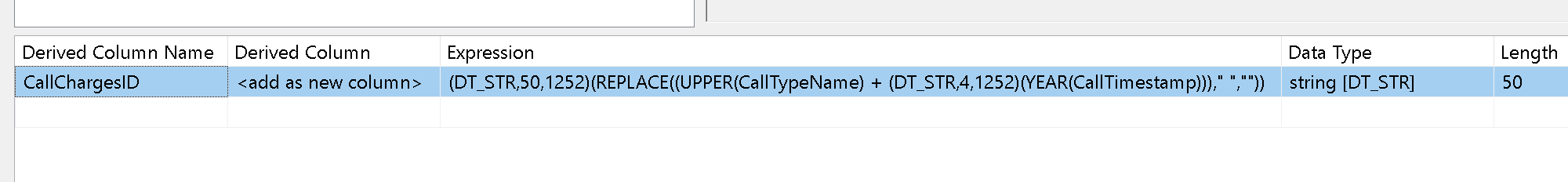
In the 3rd step, we had to change the Call Type (which is the Call Type ID) column so we can do the lookup in the next step:



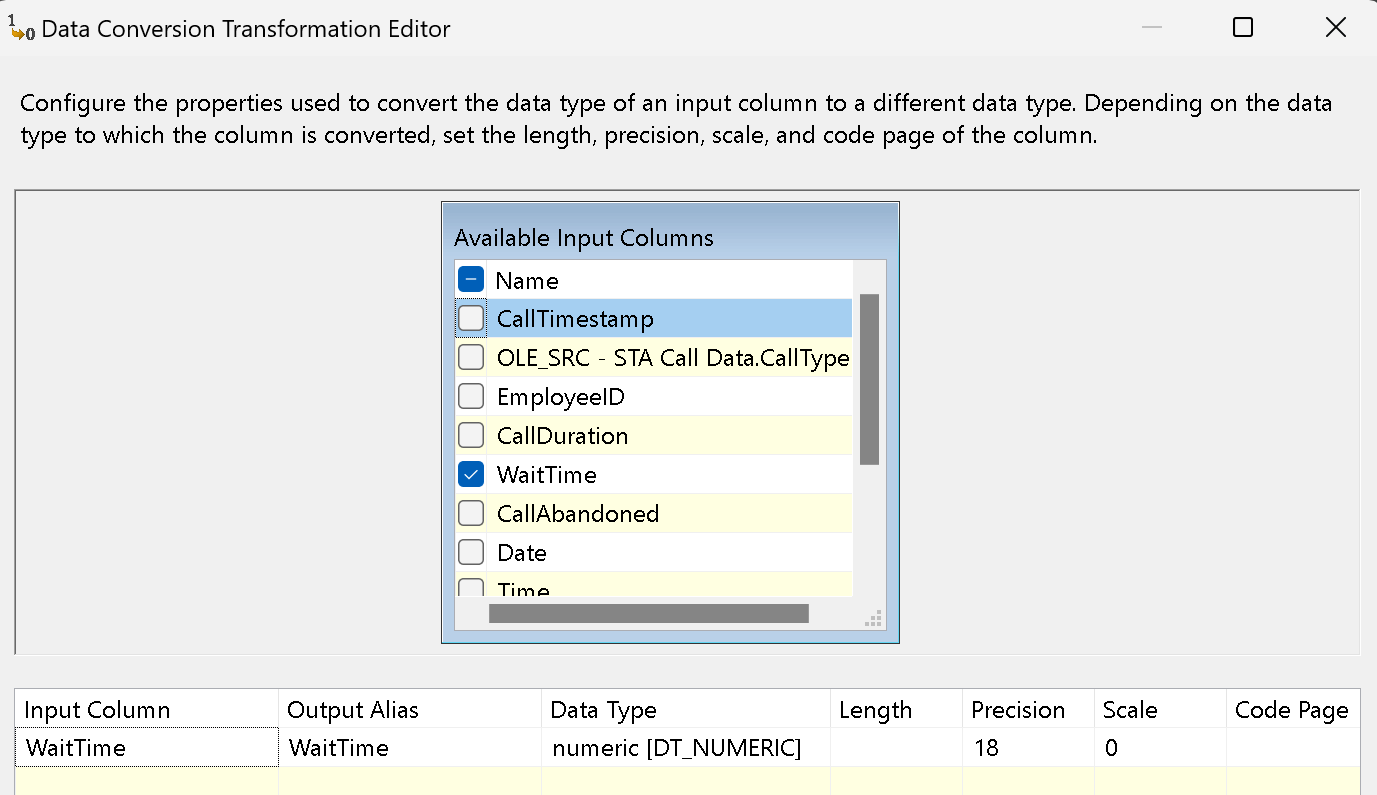
Then we do the lookup to get the Call Type’s name:



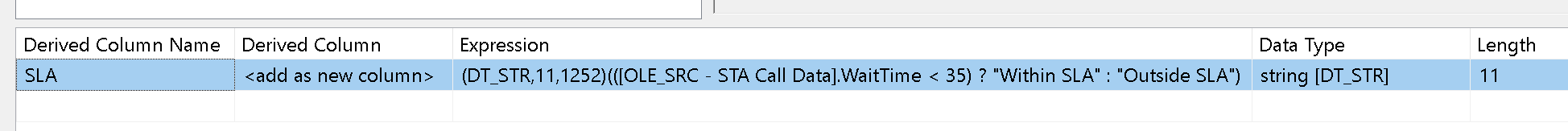
In the next step, we need to create the Call Charges ID Column so we can connect to to Call Charges table:



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”:



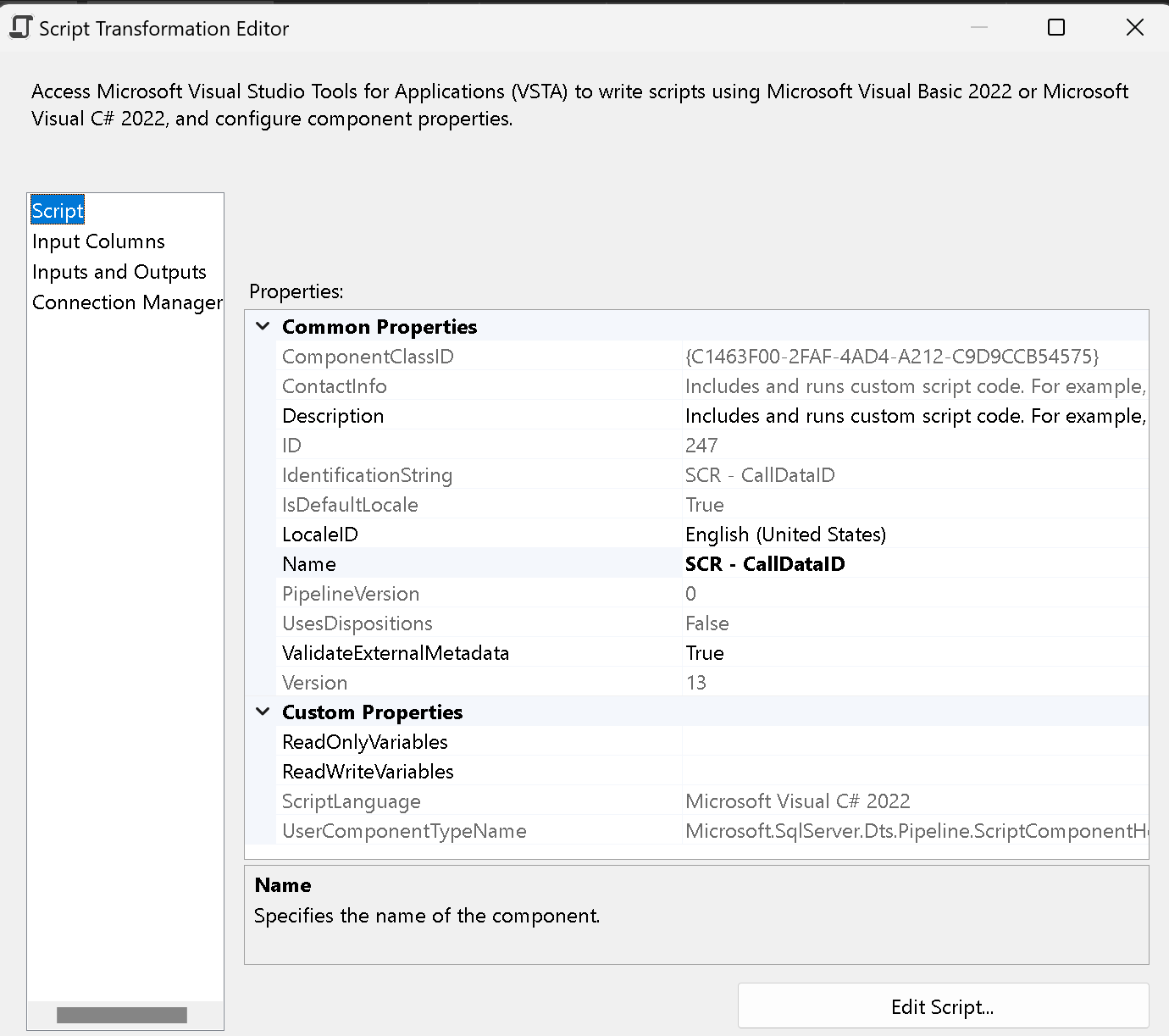
In the 8th step, we created the Call Data ID column, which ensure that this table will have a primary key. To do it, we use the Script Transformation Editor in the toolbox.

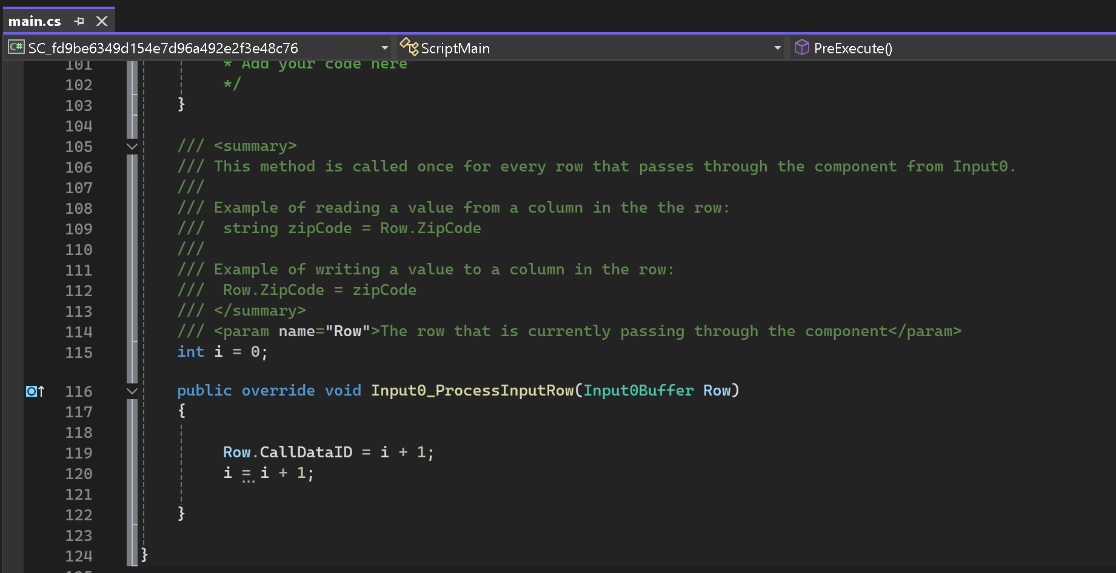
In the Script -> Inputs and Outputs -> Create the CallDataID as below:

A screenshot of a computer

Description automatically generated

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:

A screenshot of a computer

Description automatically generated

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

A screenshot of a computer

Description automatically generated

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)

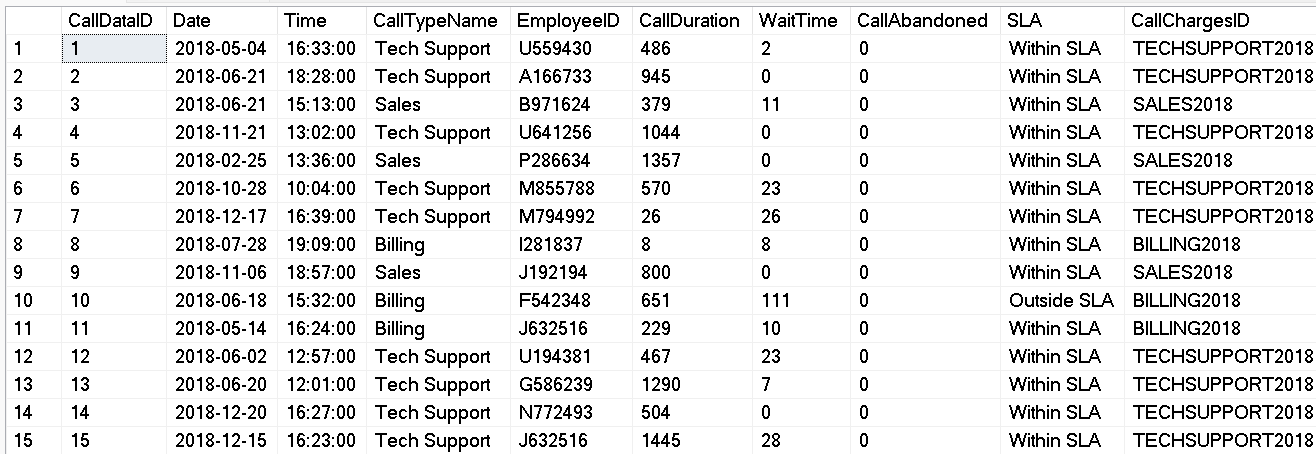
)

Here is the mapping:

A screenshot of a computer

Description automatically generated

And this is the first 10 results:



All the errors are sent to the same “Technical\_Rejects” table in ADM database.

# 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'

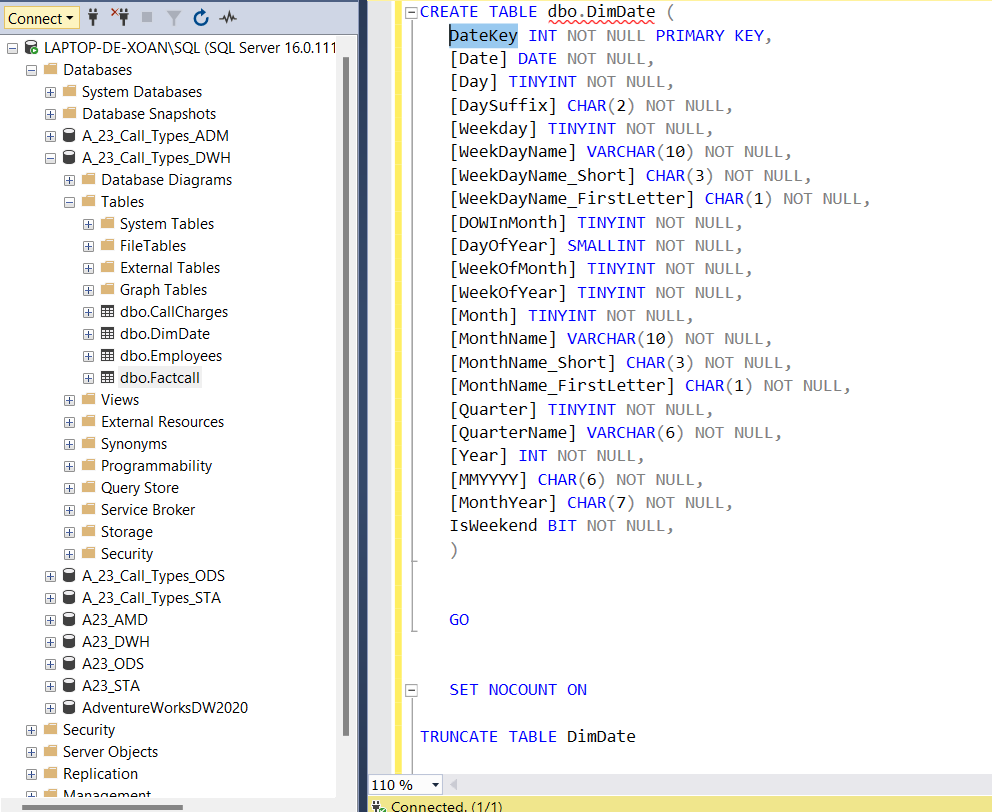
THEN 1

ELSE 0

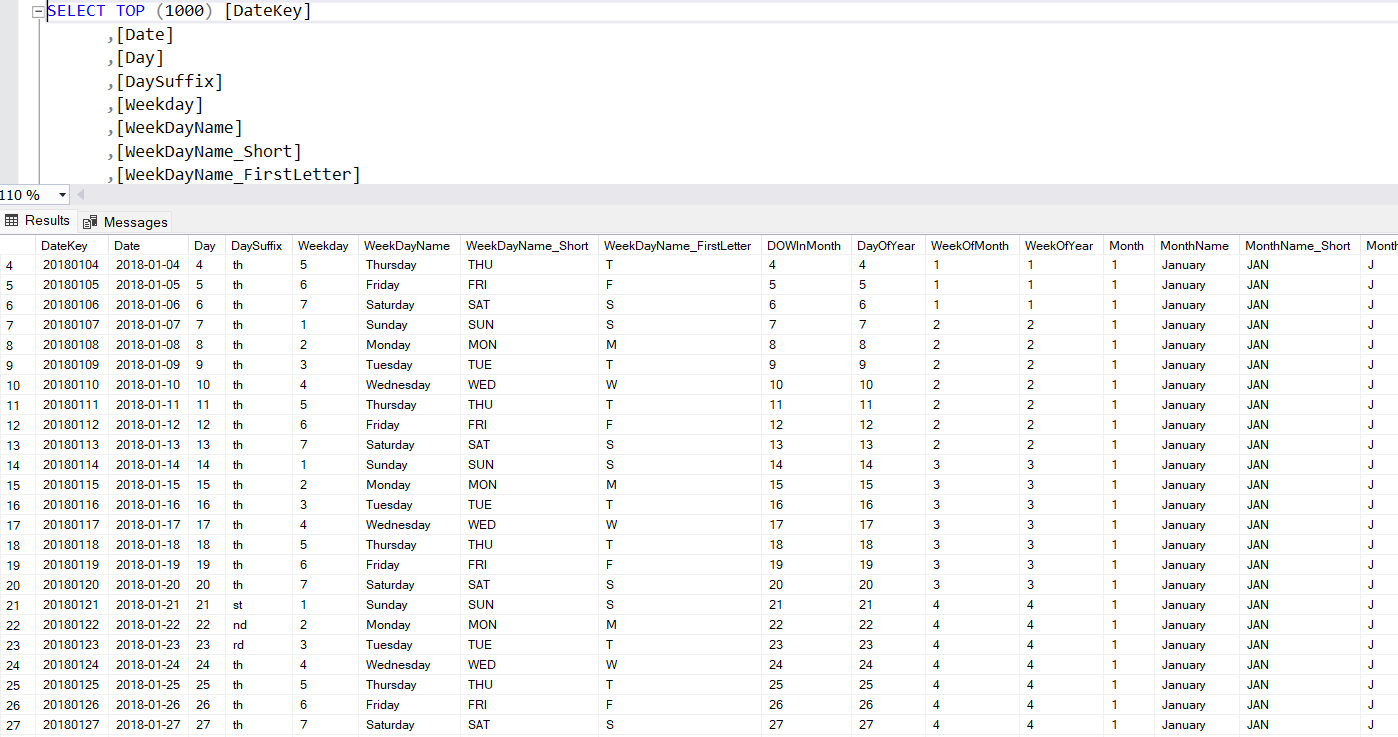
END

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

END



Below is the table Dimdate:



## 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,

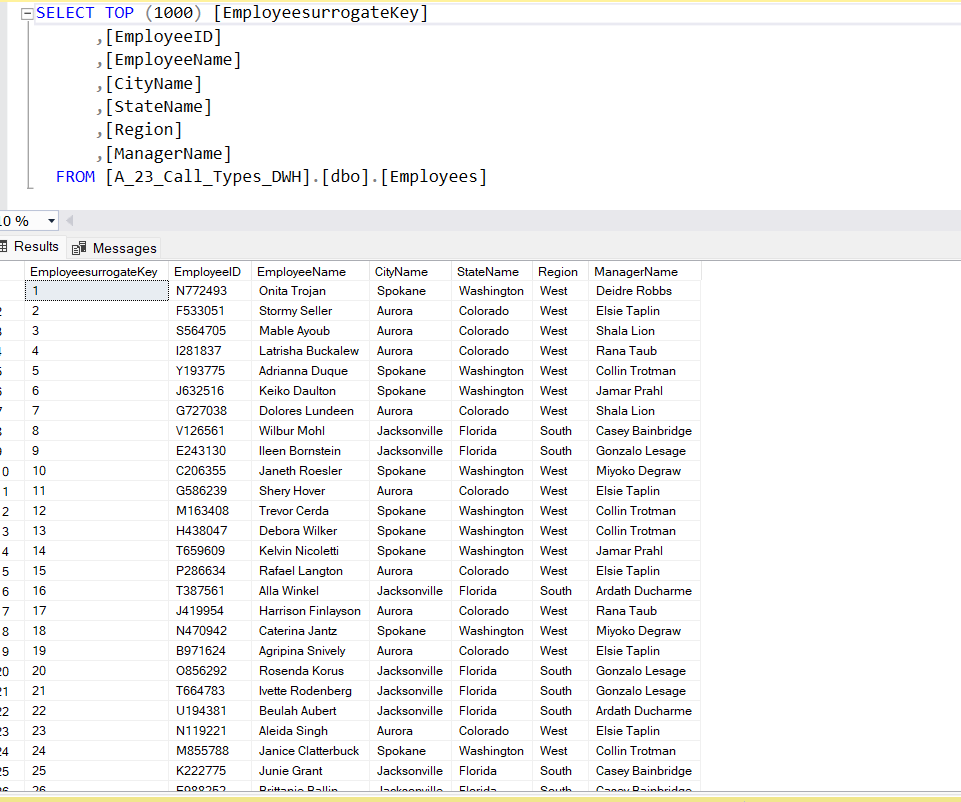
[Region] [varchar](20) NULL,

[ManagerName] [varchar](100) NULL

) ON [PRIMARY]

GO

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

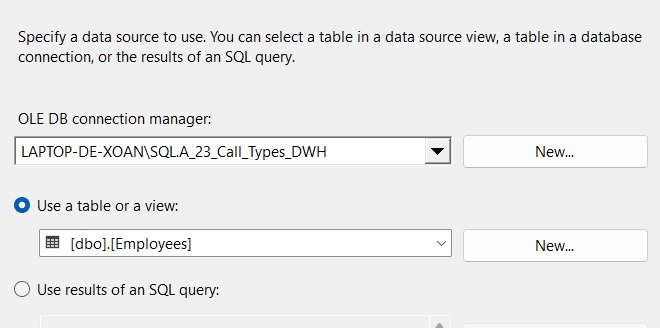


Then we can load the data with the process bellow:

A screenshot of a computer

Description automatically generated

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]”.



A screenshot of a computer

Description automatically generated

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.

A screenshot of a computer

Description automatically generated

## 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:

A screenshot of a computer

Description automatically generated

Then we can load the data with the process bellow:

A screenshot of a computer

Description automatically generated

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”

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

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.

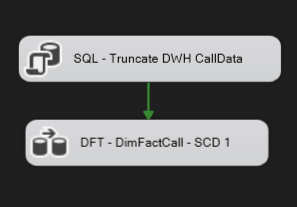
A screenshot of a computer

Description automatically generated

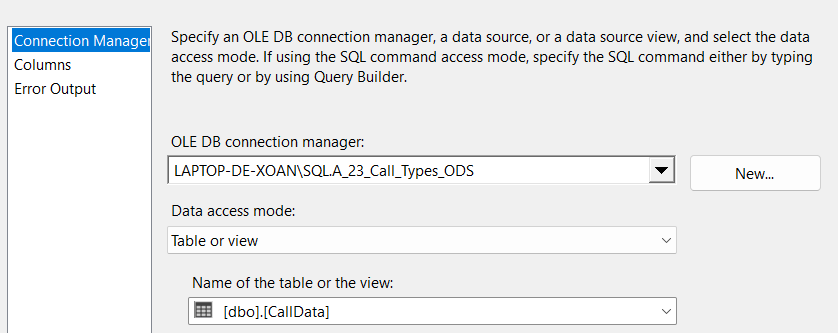
## 4.5. Integration of the Call Data Facts table

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

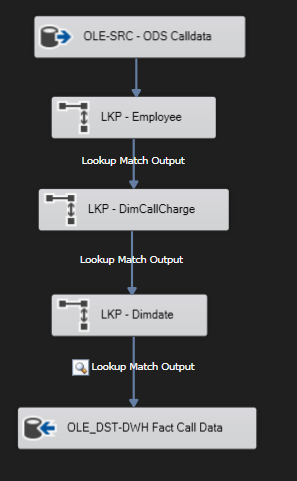
We applied Truncate for the DWH Calldata table for solve the duplication data issue.



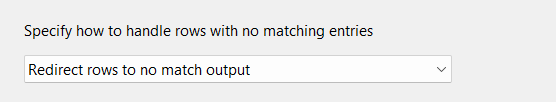
We created the OLE source from ODS Calldata, and we connect with other dimensions table via the Lookup functions. Below is the connection in OLE Source which we connect data from [dbo].[Calldata].



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:



A screenshot of a computer

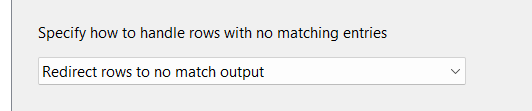
Description automatically generated

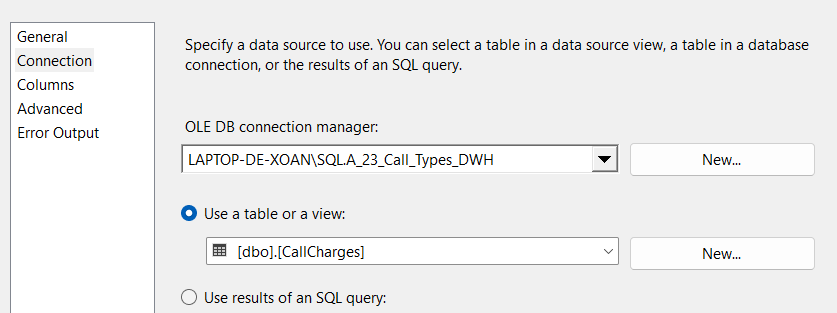
A screenshot of a computer

Description automatically generated

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

* DimCallCharge





A screenshot of a computer

Description automatically generated

* Dimdate

A screenshot of a computer

Description automatically generated

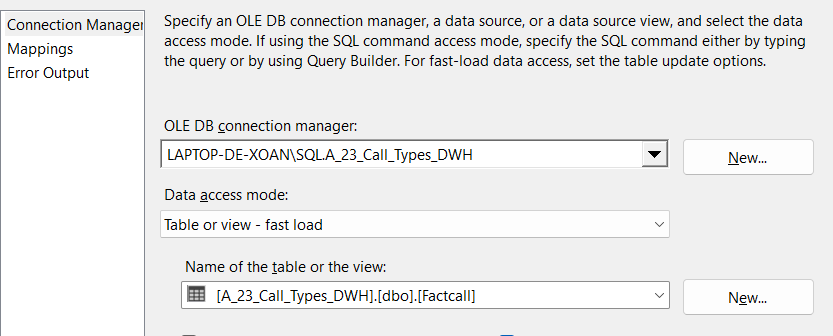
A screenshot of a computer

Description automatically generated

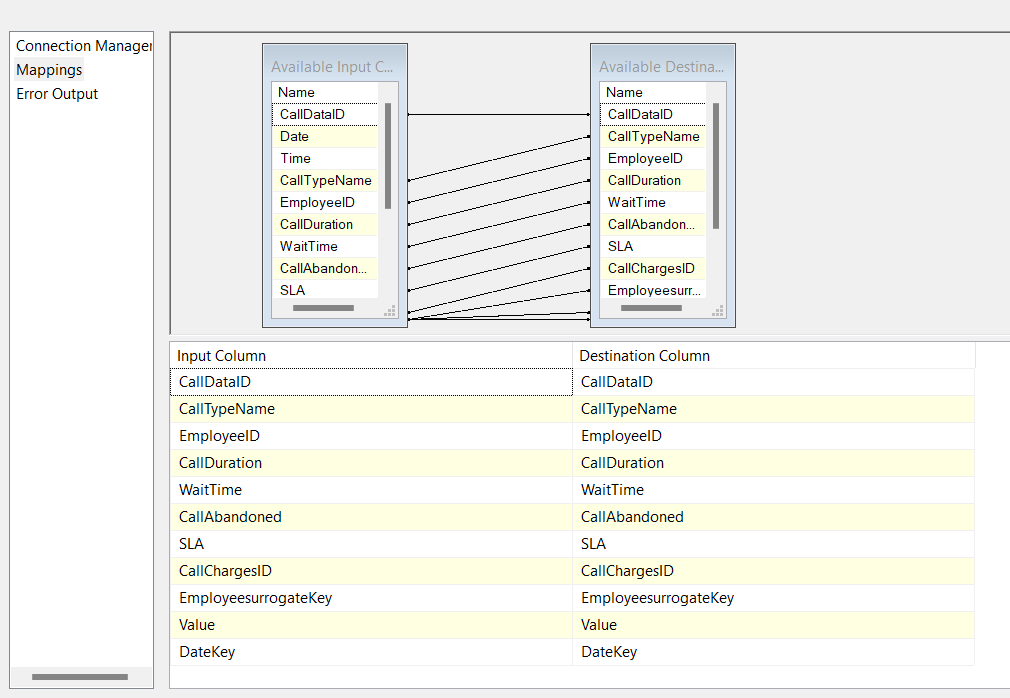
A screenshot of a computer

Description automatically generated

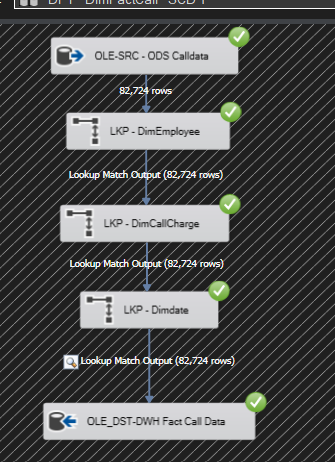
Lastly, we created the OLE-DST for FactCall data in DWH. We created the FactCall Table. The code created this table as below:



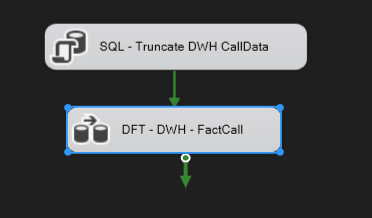
We mapped the column based on the picture below:



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.



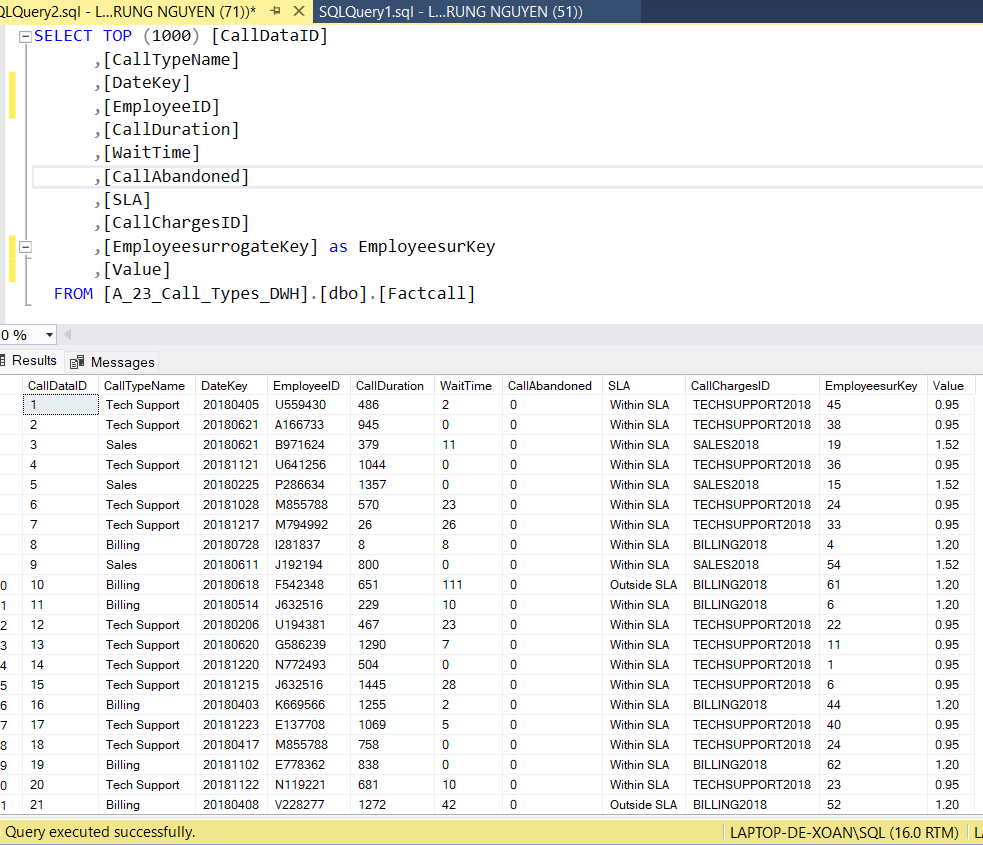
In order to avoid any duplicating risk, we applied Truncate for this table.



A screenshot of a computer

Description automatically generated

We check the result on SQL Server Management as below”



# 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.