PROJECT FOR THE CREATION OF A TRAFFIC ACCIDENT DATAMART IN THE USA (2016 - 2023)

ENG. SILVA PARRAGUEZ MAXIMO

I. Preparing and understanding the Data:

1. Fountain:

The data was obtained from US-wide car accident records covering 49 states. The data is from February 2016 through March 2023. It is a . csv file containing approximately 7.7 million accident records.

This project consisted of using the .CSV data to carry out the entire ETL process, extracting the data, transforming it, and loading it to a final destination, which is a Datamart , which is a STAR schema.

The data source was taken from: https://www.kaggle.com/datasets/sobhanmoosavi/us-accidents

The process that was carried out was:

2. Columns:

The . csv in question contains 46 columns, each of which is detailed below:

Original Column	Description
ID	This is a unique identifier for the accident record.
Source	Raw data source on accidents
Severity	Shows the severity of the accident, a number between 1 and 4, where 1 indicates the least impact on traffic (i.e. short delay as a result of the accident) and 4 indicates a significant impact on traffic (i.e. long delay).
Start_Time	Displays the start time of the accident in the local time zone.
End_Time	Displays the end time of the accident in the local time zone. The end time here refers to the time when the impact of the accident on traffic flow was ruled out.
Start_Lat	Displays the latitude in GPS coordinates of the starting point.
Start_Lng	Displays the longitude in GPS coordinates of the starting point.
End_Lat	Displays the latitude in GPS coordinates of the end point.
End_Lng	Displays the longitude in GPS coordinates of the end point.
Distance (mi)	The length of the road affected by the accident in miles.
Description	Displays a description of the accident provided by a person.
Street	Displays the street name in the address field.
City	Displays the city in the address field.
County	Displays the county in the address field.
State	Displays the status in the address field.
Zipcode	Displays the zip code in the address field.
Country	Displays the country in the address field.
Timezone	Displays the time zone based on the location of the accident (Eastern, Central, etc.).
Airport_Code	Indicates a weather station located at the airport that is closest to the accident site.

Weather_Timestamp	Displays the timestamp of the weather observation record (in local time).
Temperature (F)	Displays the temperature (in Fahrenheit).
Wind_Chill (F)	Displays the wind chill (in degrees Fahrenheit).
Humidity (%)	Displays humidity (in percentage).
Pressure (in)	Displays air pressure (in inches).
Visibility (mi)	Displays visibility (in miles).
Wind_Direction	Shows the direction of the wind.
Wind_Speed (mph)	Displays wind speed (in miles per hour).
Precipitation (in)	Displays the amount of precipitation in inches, if any.
Weather_Condition	Displays weather conditions (rain, snow, storm, fog, etc.)
Amenity	A POI annotation indicating the presence of a service at a nearby location.
Bump	A POI annotation indicating the presence of a speed bump or speed bump at a nearby location.
Crossing	A POI annotation indicating the presence of a junction at a nearby location.
Give_Way	A POI annotation indicating the presence of a yield sign at a nearby location.
Junction	A POI annotation indicating the presence of an intersection at a nearby location.
No_Exit	A POI annotation indicating the presence of a dead end at a nearby location.
Railway	A POI annotation indicating the presence of a railroad at a nearby location.
Roundabout	A POI annotation indicating the presence of a roundabout at a nearby location.
Station	A POI annotation indicating the presence of a station at a nearby location.
Stop	A POI annotation indicating the presence of a stop at a nearby location.
Traffic_Calming	A POI annotation indicating the presence of traffic calming measures at a nearby location.
Traffic_Signal	A POI annotation indicating the presence of a road sign at a nearby location.
Turning_Loop	A POI annotation indicating the presence of a Rotating Loop at a nearby location.
Sunrise_Sunset	Displays the period of the day (i.e. day or night) based on sunrise/sunset.
Civil_Twilight	Displays the period of day (i.e. day or night) based on civil twilight.
Nautical_Twilight	Displays the period of day (i.e. day or night) based on nautical twilight.
Astronomical_Twilight	Displays the period of day (i.e. day or night) based on astronomical twilight.

II. STEP I: FROM CSV to LOAD_ACCIDENTS.

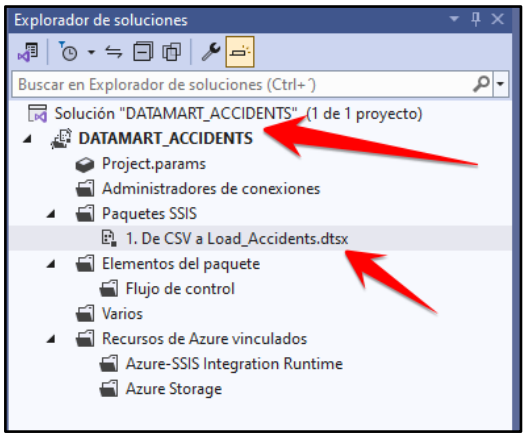
Since we have the data in the . CSV, it has a size of 3 GB, and with approximately 7.7 million records, the ideal thing is to pass it to a Load table in SQL Server, this table will act as a first warehouse where the data will be saved as is from the . csv , which will allow to handle the data in a more efficient, controlled and secure way before being loaded into a Staging database (this DB will be the second intermediary warehouse between the original data and the Datamart) before being passed to the DataMart . Although it is true, two additional steps will be taken, but this will ensure that the final destination receives clean, consistent data that is ready to be used in analysis.

1. Creating a Database in SQL Server.

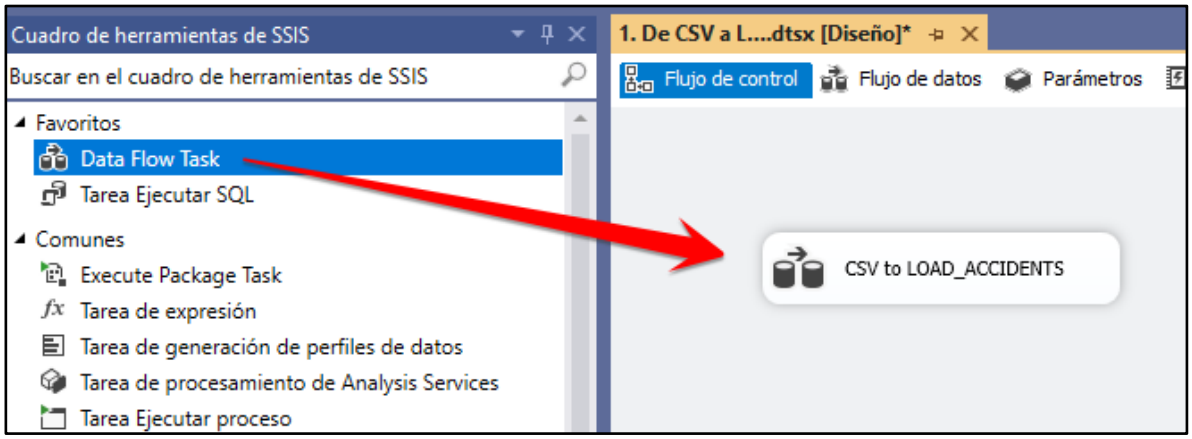
The LOAD_ACCIDENTS database will be responsible for receiving the data from the CSV file in the L_ACCIDENTS table. To do this, we create the DB and the table:

```
CREATE DATABASE LOAD_ACCIDENTS
/*CREANDO LA TABLA L_ACCIDENTS*/

CREATE TABLE L_ACCIDENTS (
    Id_Staging int identity primary key not null,
    ID varchar(50),
    Source VARCHAR(50),
    Severity int,
    Start_Time DATETIME2,
    End_Time DATETIME2,
    Start_Lat FLOAT,
    Start_Lng FLOAT,
    End_Lat FLOAT,
    End_Lng FLOAT,
    Distance_mi FLOAT,
    Description VARCHAR(800),
    Street VARCHAR(500),
    City VARCHAR(100),
    County VARCHAR(100),
    State VARCHAR(50),
    Zipcode VARCHAR(50),
    Country VARCHAR(50),
    Timezone VARCHAR(50),
    Airport_Code VARCHAR(50),
    Weather_Timestamp DATETIME2,
    Temperature_F FLOAT,
    Wind_Chill_F FLOAT,
    Humidity FLOAT,
    Pressure_in FLOAT,
    Visibility_mi FLOAT,
    Wind_Direction VARCHAR(50),
    Wind_Speed_mph FLOAT,
    Precipitation_in FLOAT,
    Weather_Condition VARCHAR(100),
    Amenity VARCHAR(10),
    Bump VARCHAR(10),
    Crossing VARCHAR(10),
    Give_Way VARCHAR(10),
    Junction VARCHAR(10),
    No_Exit VARCHAR(10),
    Railway VARCHAR(10),
    Roundabout VARCHAR(10),
    Station VARCHAR(10),
    Stop VARCHAR(10),
    Traffic_Calming VARCHAR(10),
    Traffic_Signal VARCHAR(10),
    Turning_Loop VARCHAR(10),
    Sunrise_Sunset VARCHAR(10),
    Civil_Twilight VARCHAR(10),
    Nautical_Twilight VARCHAR(10),
    Astronomical_Twilight VARCHAR(10)
);
```

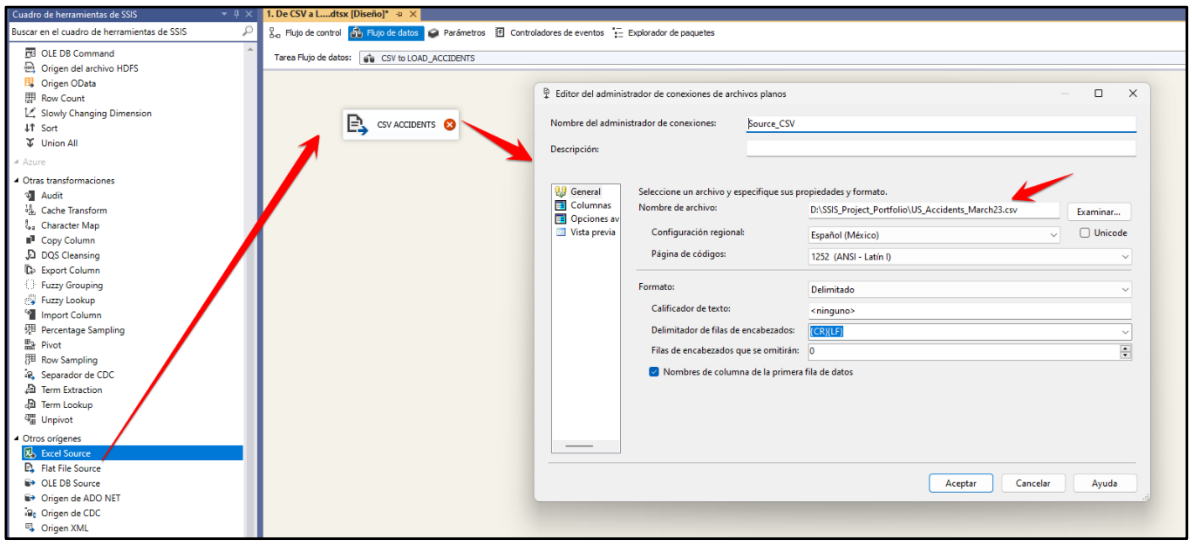
2. **I created a Project in SSIS.**



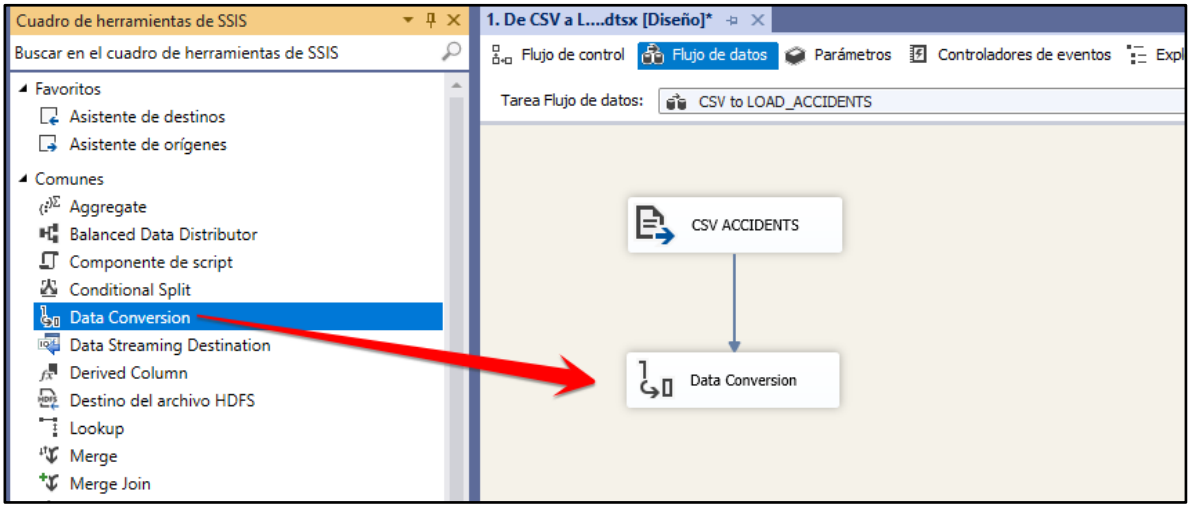
3. **I used a Data Flow Task.**

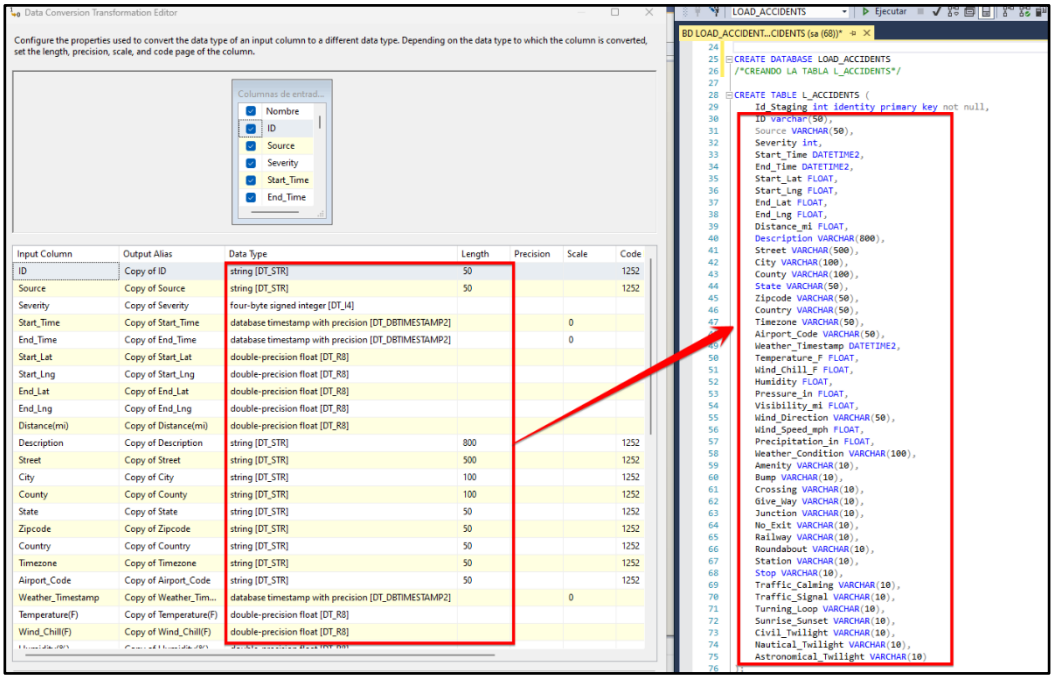


4. **I set the source to a CSV file.**

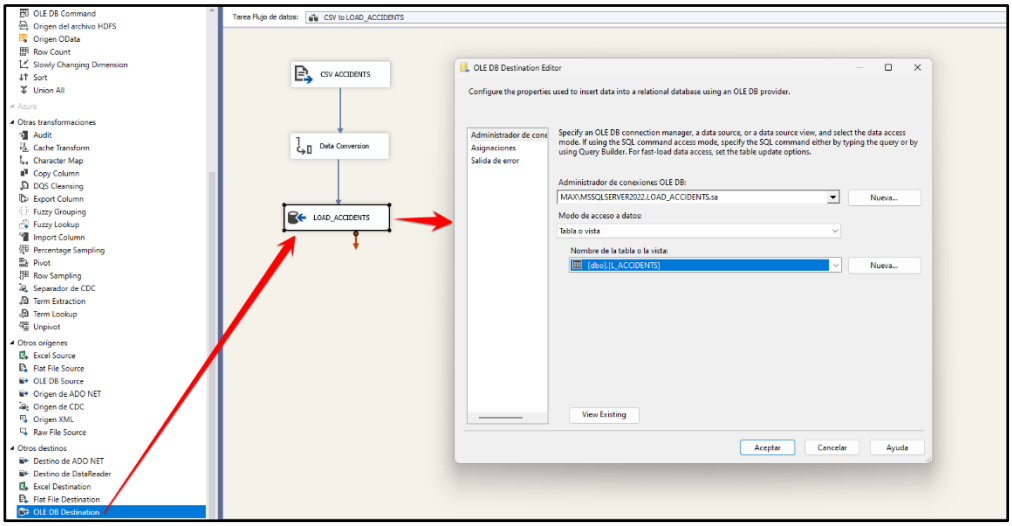


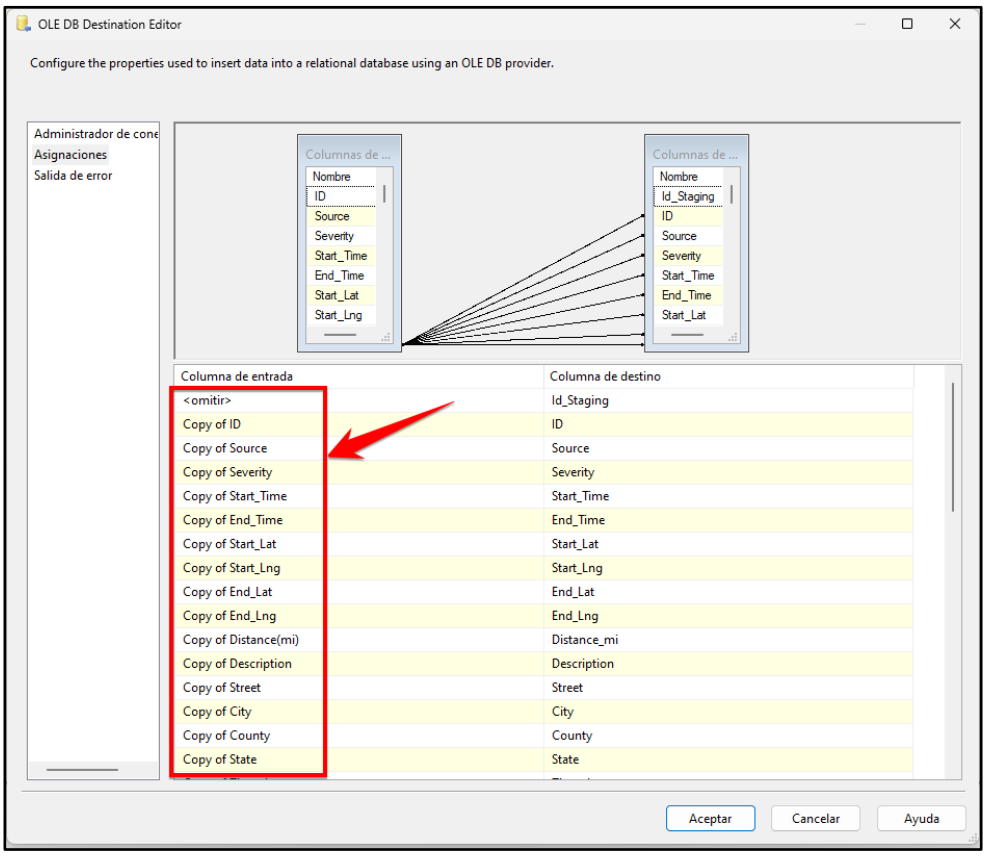
5. converted data types so that there are no errors between the source and destination.



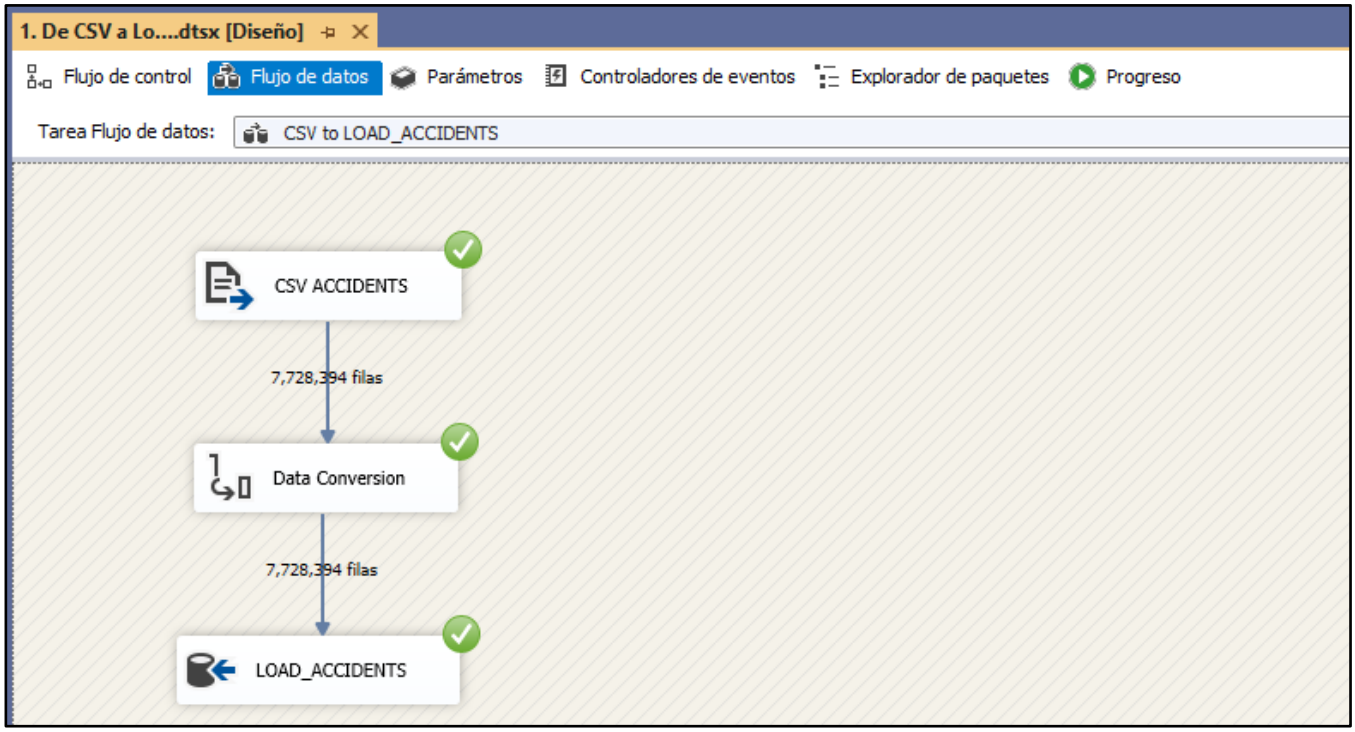


6. I set up an OLE DB destination for our DB in SQL Server LOAD_ACCIDENTS.





7. I proceeded to execute the project:

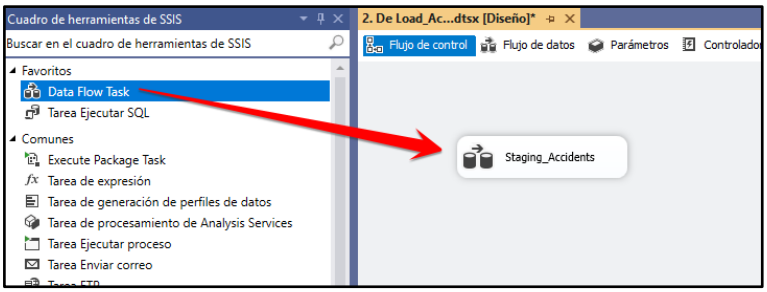


8. **created a new table with the names of each STATE, since the data source is in acronyms, and I want to have the name of the state that will be created later in the cleaning process.**

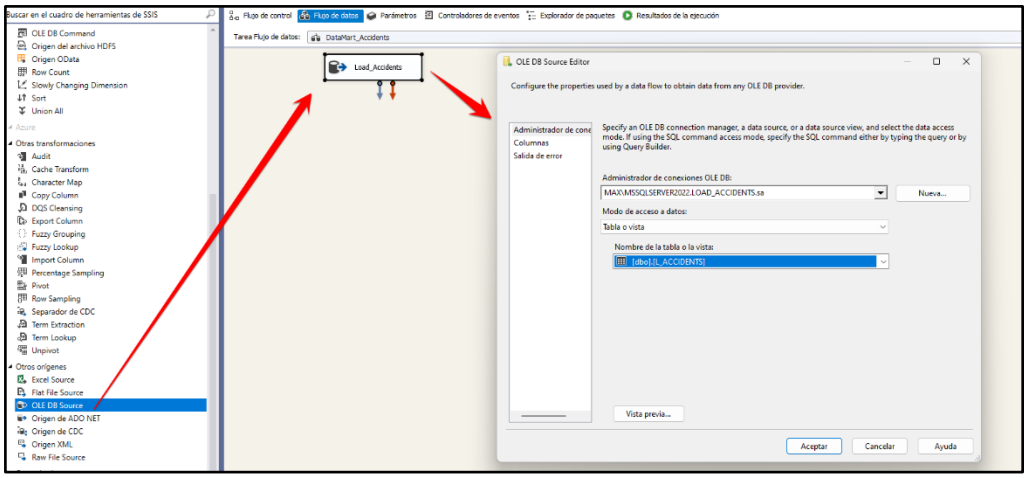
```
CREATE TABLE STATE_REFERENCE (
    State_Abbreviation CHAR(2) PRIMARY KEY,
    State_Name VARCHAR(50)
);

INSERT INTO STATE_REFERENCE (State_Abbreviation, State_Name)
VALUES
('AL', 'Alabama'), ('AK', 'Alaska'), ('AZ', 'Arizona'),('AR', 'Arkansas'),
('CA', 'California'),('CO', 'Colorado'),('CT', 'Connecticut'),
('DE', 'Delaware'),('FL', 'Florida'),('GA', 'Georgia'),
('HI', 'Hawaii'),('ID', 'Idaho'),('IL', 'Illinois'),
('IN', 'Indiana'),('IA', 'Iowa'),('KS', 'Kansas'),
('KY', 'Kentucky'),('LA', 'Louisiana'),('ME', 'Maine'),
('MD', 'Maryland'),('MA', 'Massachusetts'),('MI', 'Michigan'),
('MN', 'Minnesota'),('MS', 'Mississippi'),('MO', 'Missouri'),
('MT', 'Montana'),('NE', 'Nebraska'),('NV', 'Nevada'),
('NH', 'New Hampshire'),('NJ', 'New Jersey'),('NM', 'New Mexico'),
('NY', 'New York'),('NC', 'North Carolina'),('ND', 'North Dakota'),
('OH', 'Ohio'),('OK', 'Oklahoma'),('OR', 'Oregon'),
('PA', 'Pennsylvania'),('RI', 'Rhode Island'),('SC', 'South Carolina'),
('SD', 'South Dakota'),('TN', 'Tennessee'),('TX', 'Texas'),
('UT', 'Utah'),('VA', 'Virginia'),('VT', 'Vermont'),
('WA', 'Washington'),('WI', 'Wisconsin'),('WV', 'West Virginia'),
('WY', 'Wyoming');
```

III. STEP II: FROM LOAD_ACCIDENTS TO STAGING_ACCIDENTS (DATA CLEANING)

We create a new project where the data will be cleaned and loaded and drag a Data Flow Task:

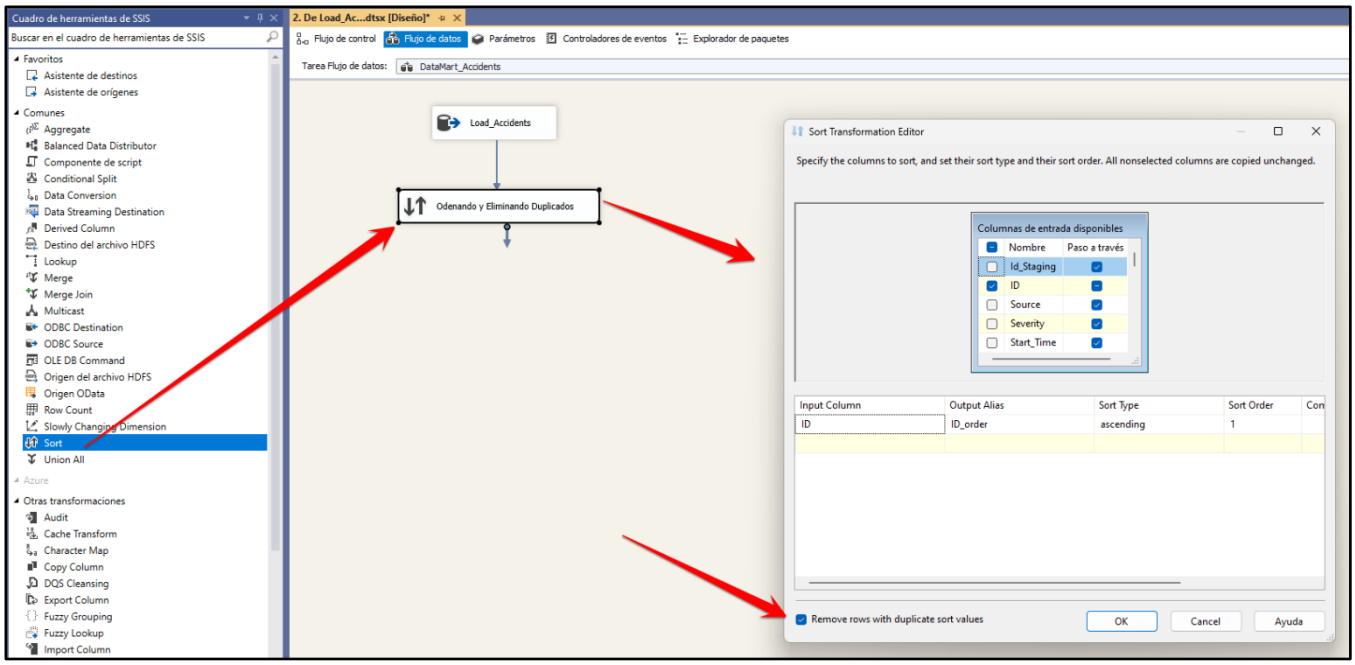


We create an OLE DB data source referencing the LOAD_ACCIDENTS database:



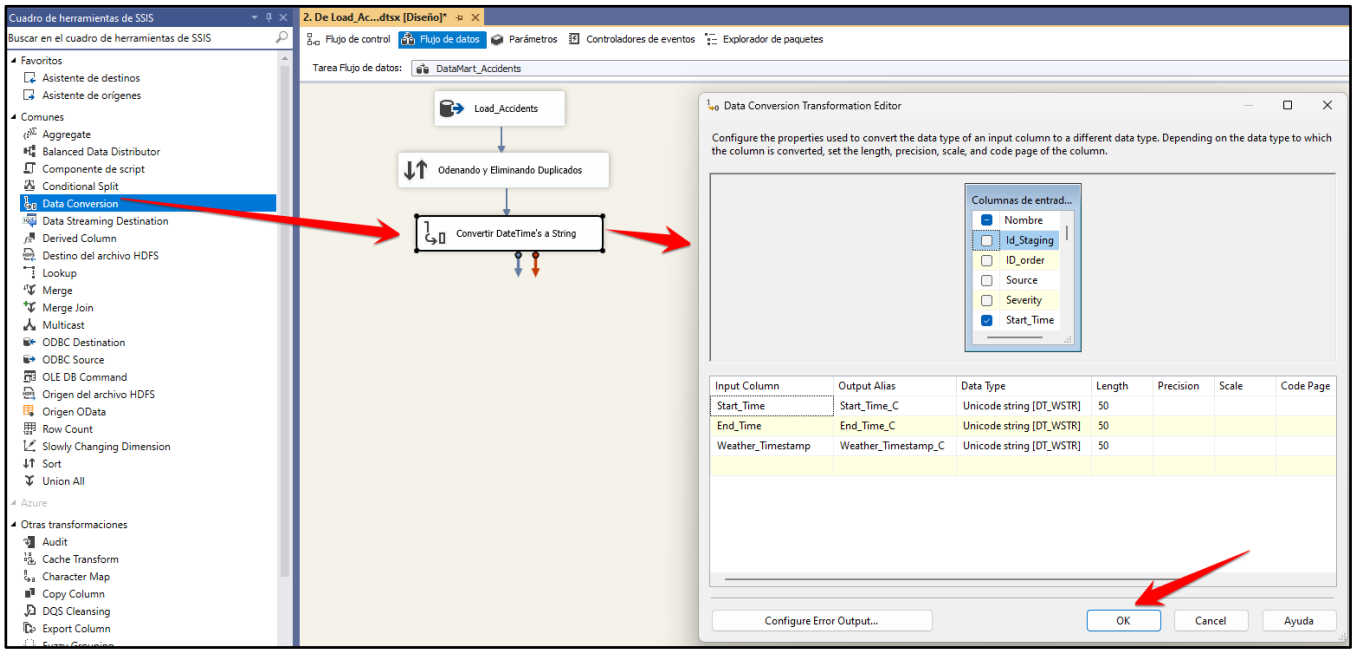
1. Remove Duplicates.

To ensure that there is unique data, we proceed to eliminate records that may be duplicated using the **Sort component** :



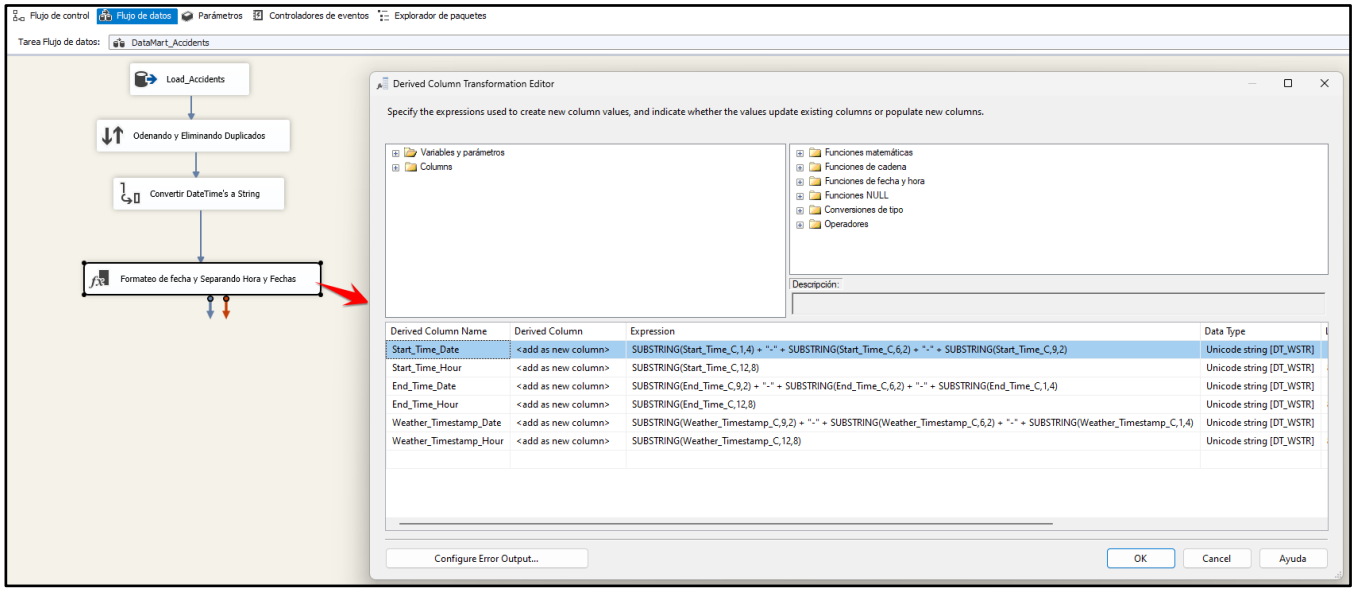
2. Data conversion.

Since we have dates and times in DateTime format , we want to split in the next step the dates and times into different columns, so we will convert them to String and then from that string we will extract some of the text to pass it to different columns, I will use the **Data Conversion** component .



3. Formatting dates and times.

To ensure that the dates are well structured and do not have errors, it is necessary to ensure that all records have the same format and the dates are correct, we will also divide the hours of the dates, all this will be done using the **Derive Column component** :



4. Handling Null/Non-Data Records.

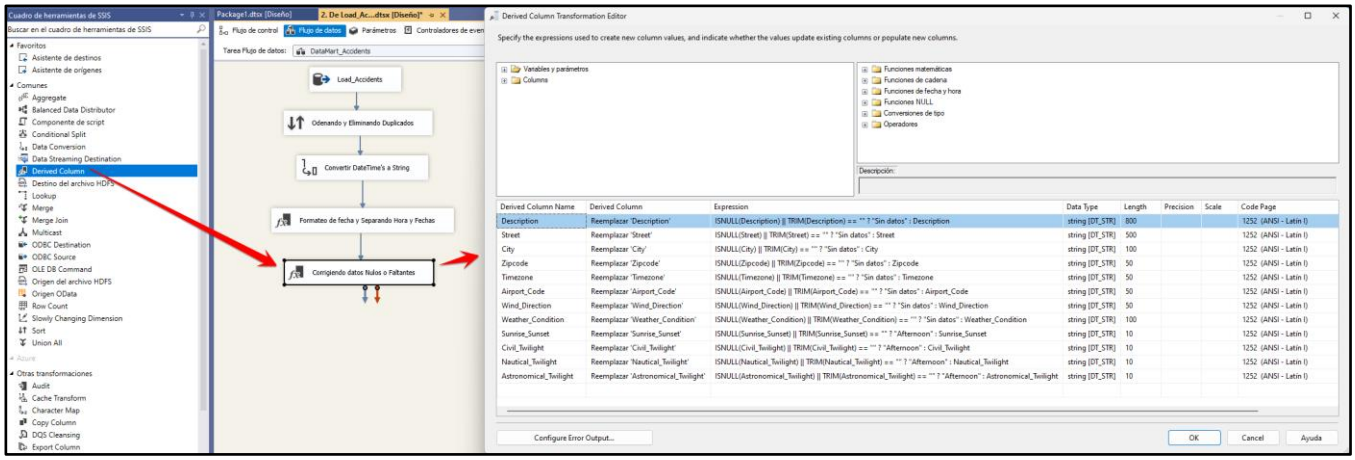
In this step **we check for NULL records or missing data** present in our Data, to do this we explore in SQL Server which columns of the LOAD_ACCIDENTS Table do not have data, to do this I applied the following query:

```
----- Creando Variables para usar una consulta dinámica
DECLARE @TableName NVARCHAR(MAX) = 'L_ACCIDENTS';
DECLARE @SQL NVARCHAR(MAX);

-- Construcción dinámica de la consulta
SET @SQL =
    'SELECT ColumnName, SUM(NullCount) AS NullCount ' +
    'FROM (' +
    STUFF((SELECT
        ' UNION ALL SELECT ''' + c.name + ''' AS ColumnName, COUNT(1) AS NullCount ' +
        'FROM ' + @TableName + ' WHERE [' + c.name + '] IS NULL ' +
        CASE
            -- Validación adicional si es de tipo DATETIME
            WHEN c.system_type_id IN (61, 58) THEN
                'OR TRY_CAST([' + c.name + '] AS DATETIME) IS NULL '
            -- Para columnas de texto
            WHEN c.system_type_id IN (167, 175, 231, 239) THEN
                'OR [' + c.name + '] = '''' '
            ELSE
                '' -- No se agrega validación extra para otros tipos de datos
        END
    FROM sys.columns c
    INNER JOIN sys.tables t ON c.object_id = t.object_id
    WHERE t.name = @TableName
    FOR XML PATH(''), TYPE).value('.', 'NVARCHAR(MAX)'), 1, 11, '') +
    ') AS Results ' +
    'GROUP BY ColumnName ' +
    'ORDER BY NullCount DESC';

-- Ejecución de la consulta resultante
EXEC sp_executesql @SQL;
```

When executing, we validate that there are indeed columns with null or missing records and when sorted from highest to lowest, we see that there are **12 columns** that we must process the missing data for:

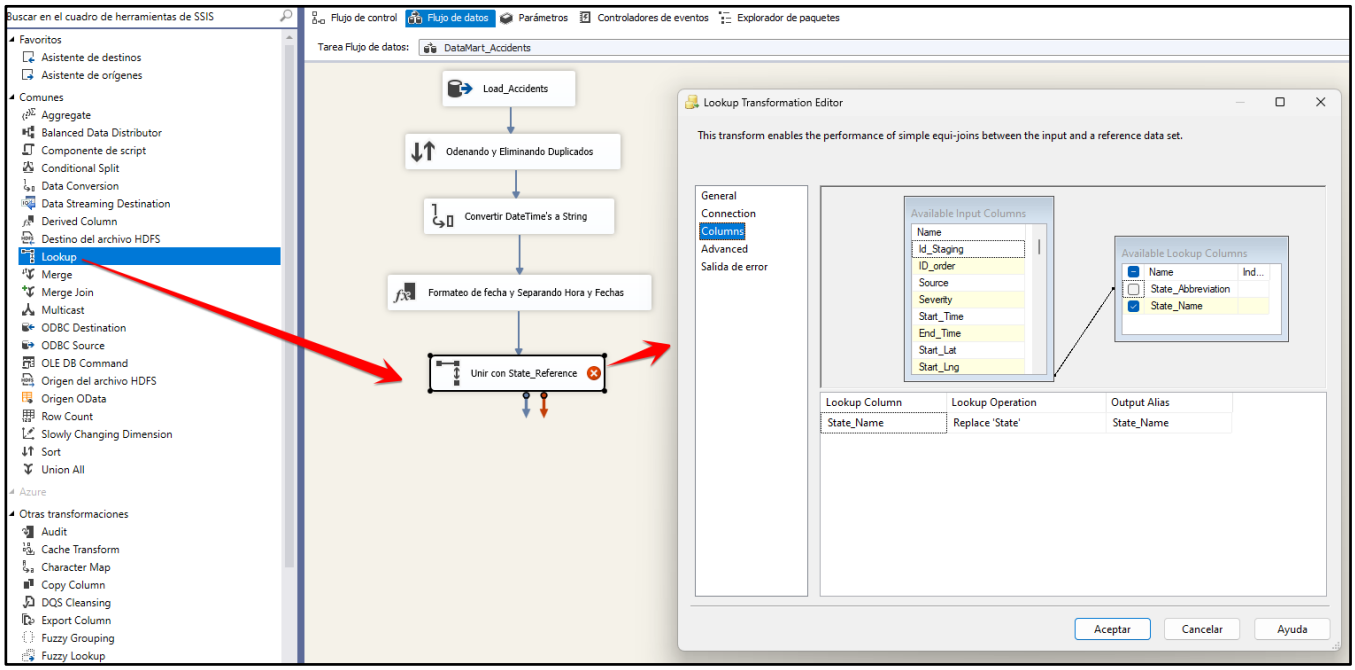
	ColumnName	NullCount
1	Wind_Direction	175206
2	Weather_Condition	173459
3	Sunrise_Sunset	23246
4	Civil_Twilight	23246
5	Nautical_Twilight	23246
6	Astronomical_Twilight	23246
7	Airport_Code	22635
8	Street	10869
9	Timezone	7808
10	Zipcode	1915
11	City	253
12	Description	5
13	Country	0
14	County	0
15	State	0
16	Id_Staging	0
17	ID	0
18	Source	0
19	Severity	0
20	Start_Time	0
21	End_Time	0
22	Start_Lat	0
23	Start_Lng	0
24	End_Lat	0
25	End_Lng	0
26	Distance_mi	0
27	Weather_Timestamp	0
28	Temperature_F	0
29	Wind_Chill_F	0
30	Humidity	0
31	Pressure_in	0
32	Visibility_mi	0
33	Amenity	0
34	Bump	0
35	Crossing	0
36	Give_Way	0
37	Junction	0
38	No_Exit	0
39	Railway	0
40	Roundabout	0
41	Station	0
42	Stop	0
43	Traffic_Calming	0
44	Traffic_Signal	0
45	Turning_Loop	0
46	Wind_Speed_mph	0
47	Precipitation_in	0

After checking, I used the **DeriveColum component** to replace the null or missing values with understandable data:



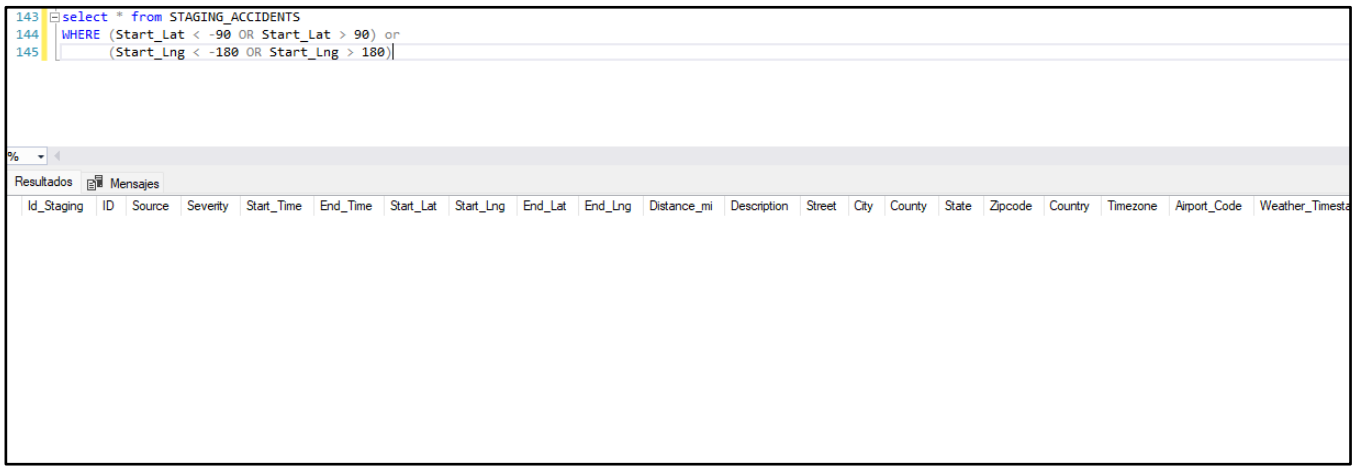
5. Normalize/Replace Values (State Abbreviations):

" State " column contains data of the states in abbreviated form, the full names of the states are required, for this the " **STATE_REFERENCE " table was created that we will use together with the Lookup** component . for this process.



6. Check the validity of the coordinates.

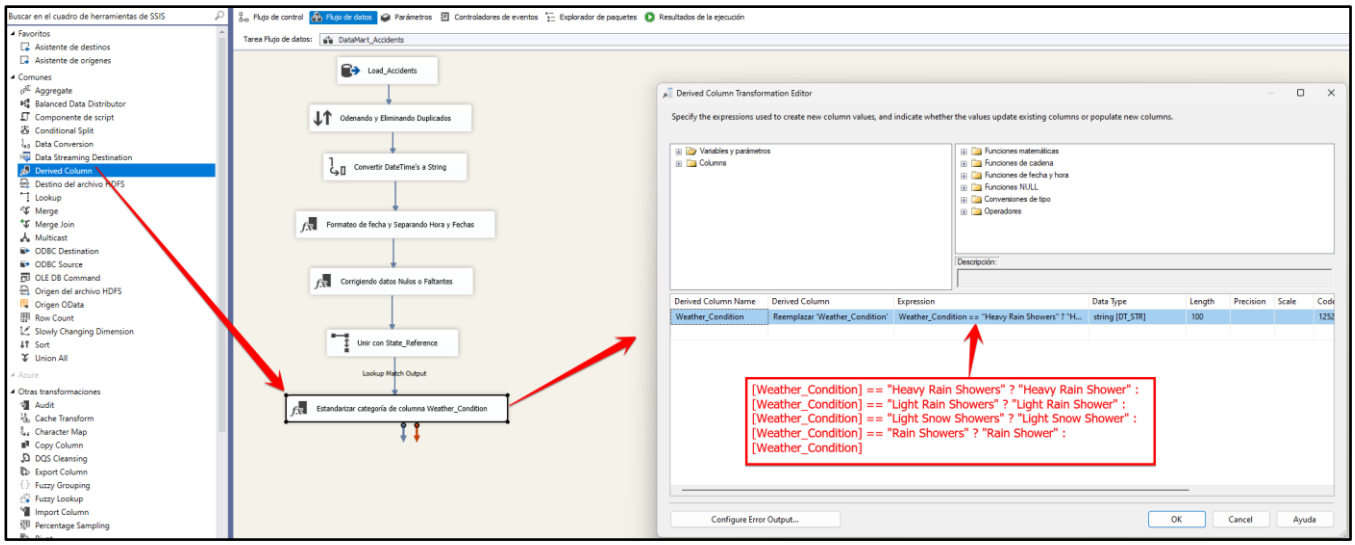
The Latitude and Longitude field must contain valid values, the longitude must be between -90º and +90º, and the Longitude must be between -180º and +180º. To do this, we check in SQL Server if there are values outside that field:



In this case we observe that no value is outside that range, so I will continue to the next step.

7. **Standardize categorical variables in columns.**

In this step I checked if the data in the columns could be standardized, seeing that in the **Weather_Condition column** the same categories existed, only some were singular and others plural, using the **Derive Column component.** Data in plural was replaced by singular to have a standard in the data.



8. **Converting data from String to Date and Time.**

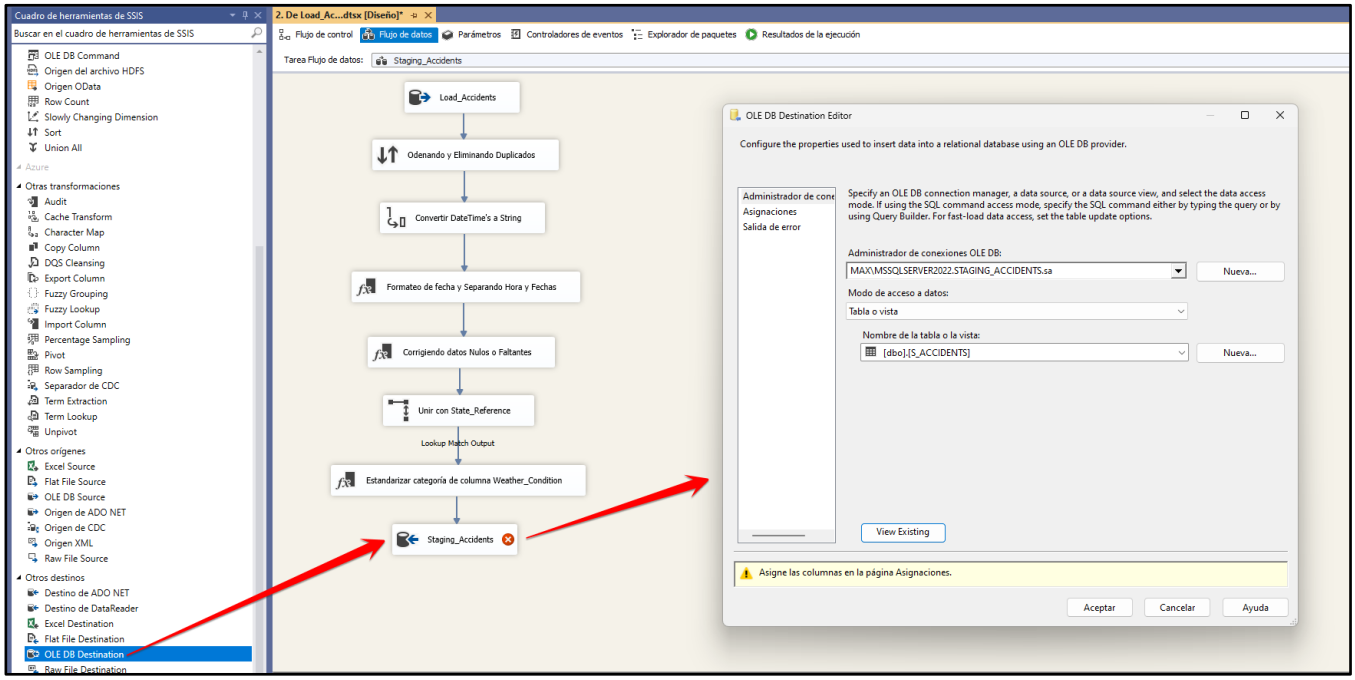
The columns that we separate from date and time are converted to data types: date and time, to then be loaded into the Destination Database.

9. **We created a Destination Database.**

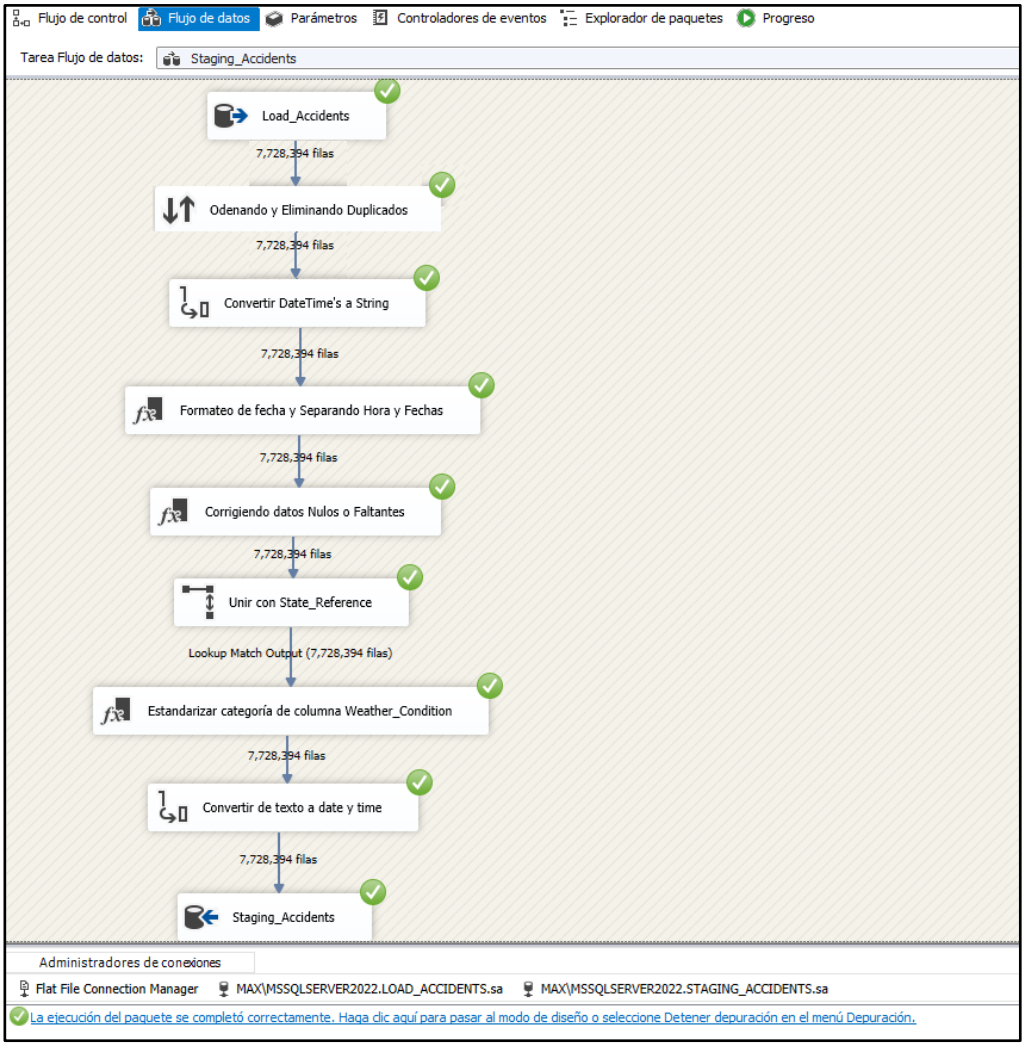
i) I created a Database called **STAGING_ACCIDENTS** , with a table **S_ACCIDENTS** , which will be a copy of the L_Accidents table from the Load_Accidents Database , only I added the 6 columns where we separate the dates and times from 3 columns.

```sql
CREATE DATABASE STAGING_ACCIDENTS
/*CREANDO LA TABLA L_ACCIDENTS*/
CREATE TABLE S_ACCIDENTS (
    Id_Staging int identity primary key not null,
    ID varchar(50),
    Source VARCHAR(50),
    Severity int,
    Start_Time DATETIME2,
    End_Time DATETIME2,
    Start_Lat FLOAT,
    Start_Lng FLOAT,
    End_Lat FLOAT,
    End_Lng FLOAT,
    Distance_mi FLOAT,
    Description VARCHAR(800),
    Street VARCHAR(500),
    City VARCHAR(100),
    County VARCHAR(100),
    State VARCHAR(50),
    Zipcode VARCHAR(50),
    Country VARCHAR(50),
    Timezone VARCHAR(50),
    Airport_Code VARCHAR(50),
    Weather_Timestamp DATETIME2,
    Temperature_F FLOAT,
    Wind_Chill_F FLOAT,
    Humidity FLOAT,
    Pressure_in FLOAT,
    Visibility_mi FLOAT,
    Wind_Direction VARCHAR(50),
    Wind_Speed_mph FLOAT,
    Precipitation_in FLOAT,
    Weather_Condition VARCHAR(100),
    Amenity VARCHAR(10),
    Bump VARCHAR(10),
    Crossing VARCHAR(10),
    Give_Way VARCHAR(10),
    Junction VARCHAR(10),
    No_Exit VARCHAR(10),
    Railway VARCHAR(10),
    Roundabout VARCHAR(10),
    Station VARCHAR(10),
    Stop VARCHAR(10),
    Traffic_Calming VARCHAR(10),
    Traffic_Signal VARCHAR(10),
    Turning_Loop VARCHAR(10),
    Sunrise_Sunset VARCHAR(10),
    Civil_Twilight VARCHAR(10),
    Nautical_Twilight VARCHAR(10),
    Astronomical_Twilight VARCHAR(10),
    Start_Time_Date date,
    Start_Time_Hour time,
    End_Time_Date date,
    End_Time_hour time,
    Weather_Timestamp_Date date,
    Weather_Timestamp_Hour time
);
```

ii) We drag an OLE DB Destination component and choose the S_ACCIDENTS table as the Destination.

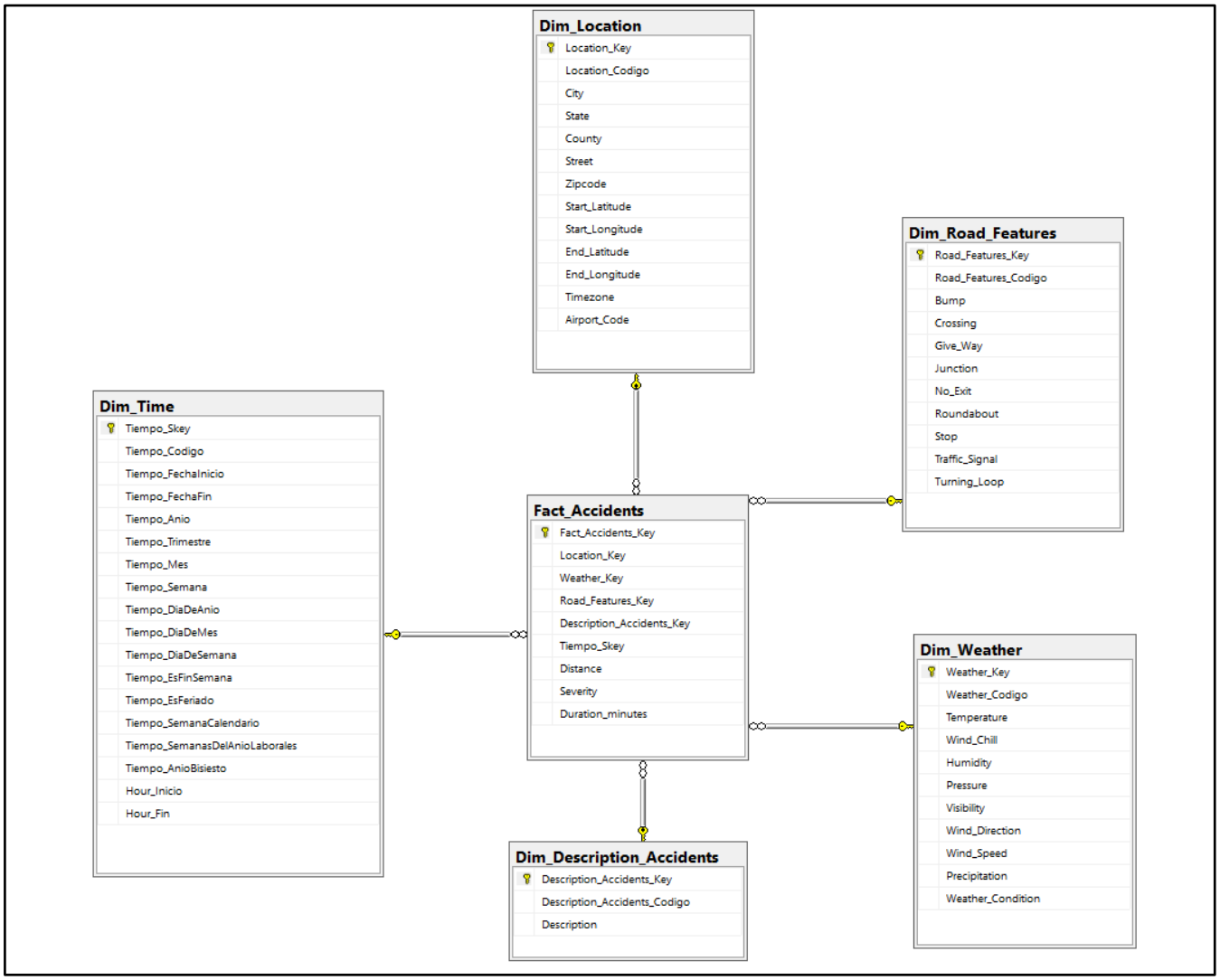


iii) We execute the project.



IV. STEP III: FROM STAGING_ACCIDENTS TO DATAMART_ACCIDENTS.

Datamart schema will be **Star (The model is attached in the image)** . First, I will create **the DATAMART_ACCIDENTS database,** and continue by creating the dimensions and the facts table. Then **I will also create a table for each dimension within the STAGING_ACCIDENTS database** which will store the **modified records. of each dimension** , therefore, in the population of each dimension there will be 2 data sources, the first to extract the data from the S_ACCIDENTS table and the second from the corresponding Dimension that we will populate, this with the purpose of comparing records and saving the modified records in the tables mentioned above.

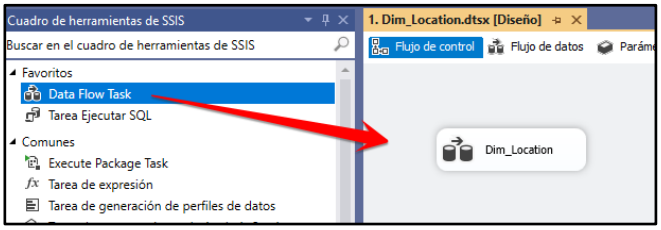


1. Populating the Dimension Location:

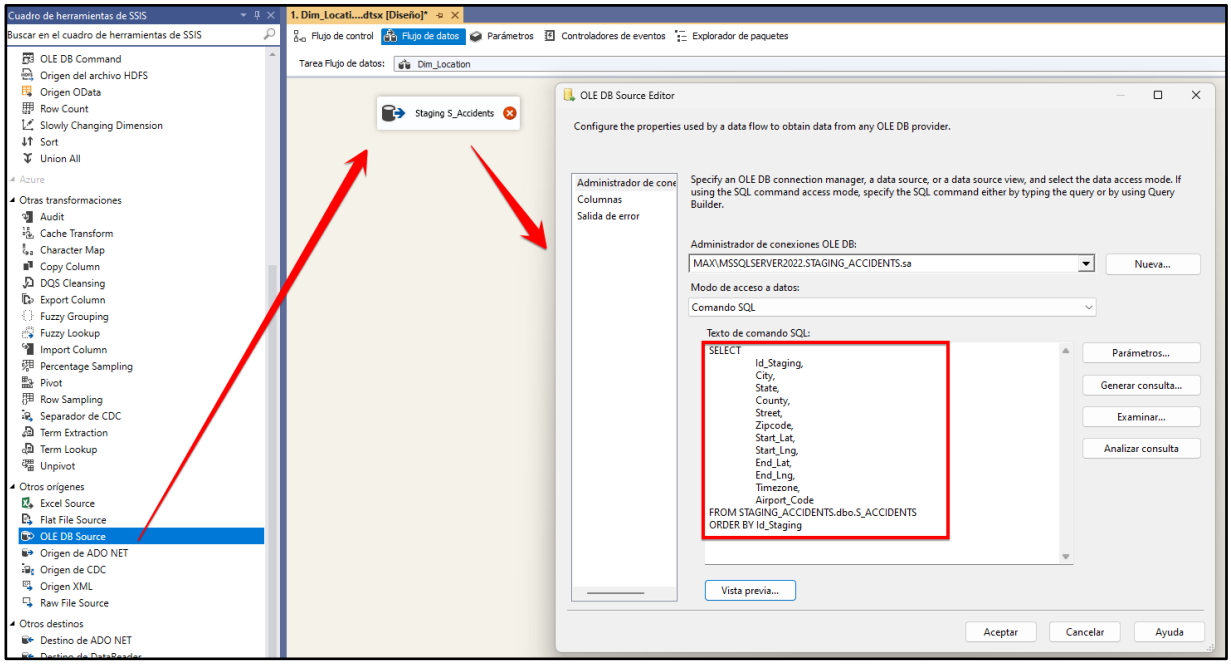
a) Creating the **Dim_Location Table** .

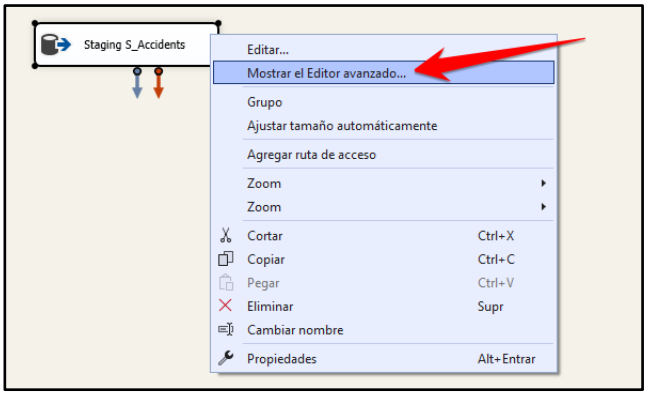
```
-- Dimensión: Ubicación
CREATE TABLE Dim_Location (
    Location_Key INT PRIMARY KEY IDENTITY(1,1),
    Location_Codigo INT,
    City VARCHAR(100),
    State VARCHAR(50),
    County VARCHAR(100),
    Street VARCHAR(500),
    Zipcode VARCHAR(50),
    Start_Latitude FLOAT,
    Start_Longitude FLOAT,
    End_Latitude FLOAT,
    End_Longitude FLOAT,
    Timezone VARCHAR(50),
    Airport_Code VARCHAR(50)
);
```

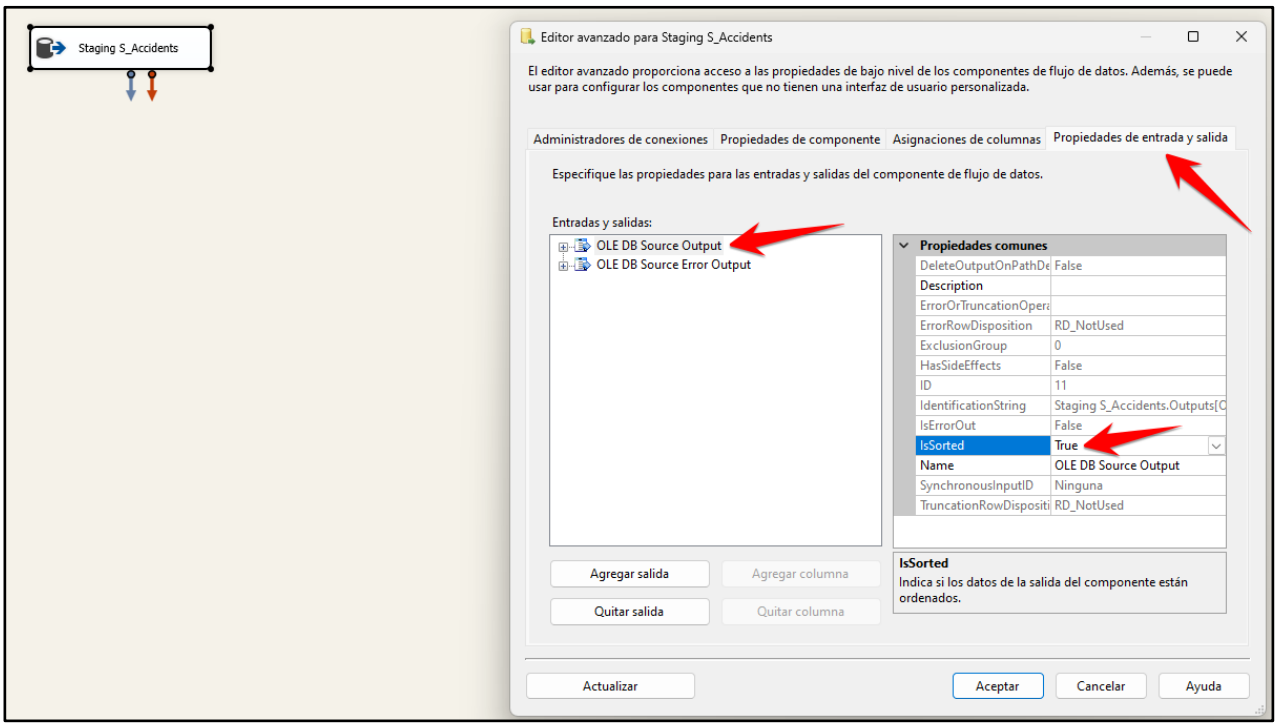
b) Dim_Location project I will use a data flow task.

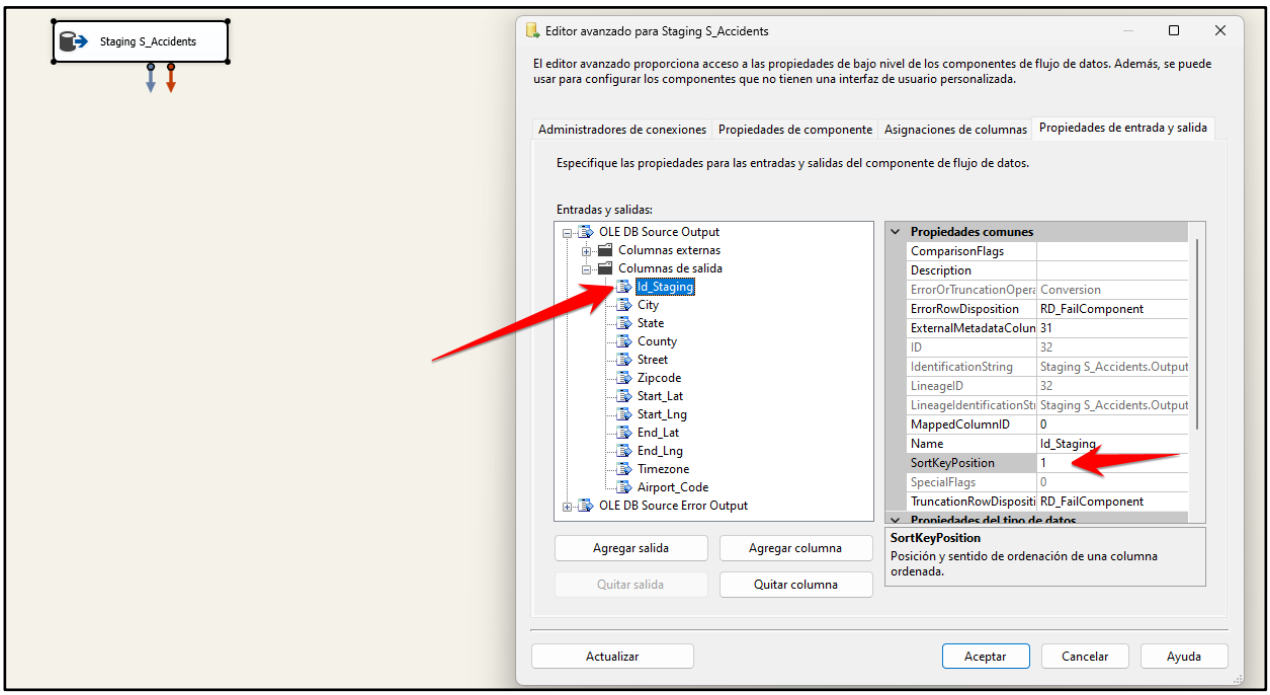


c) **S_ACCIDENTS** Data Source .

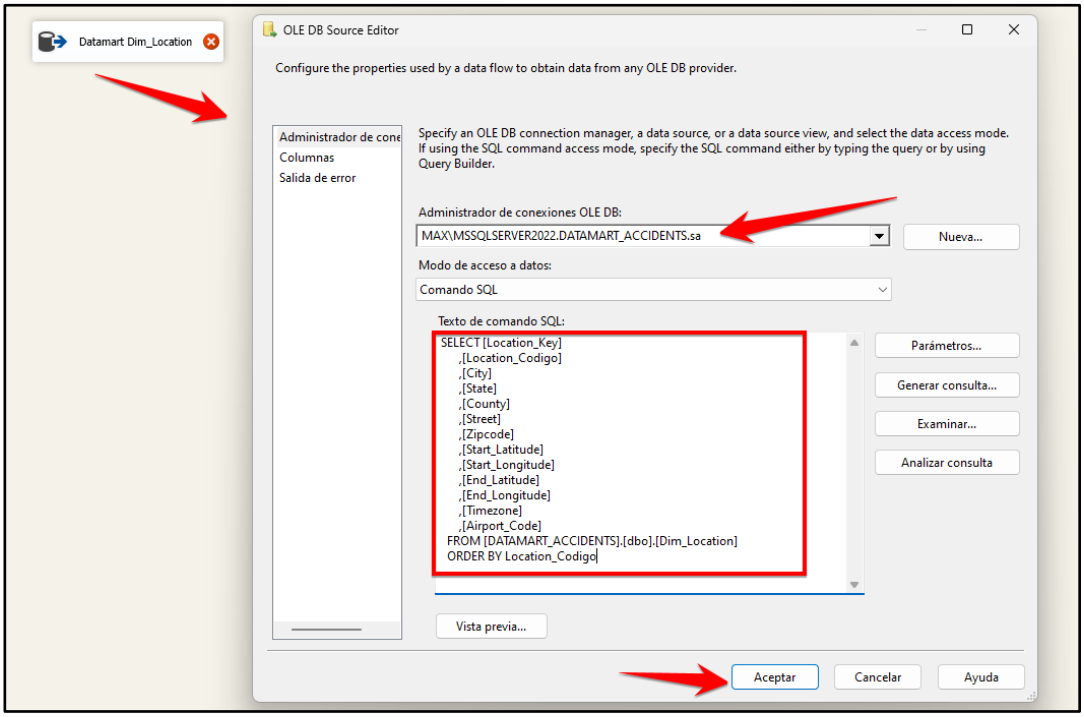


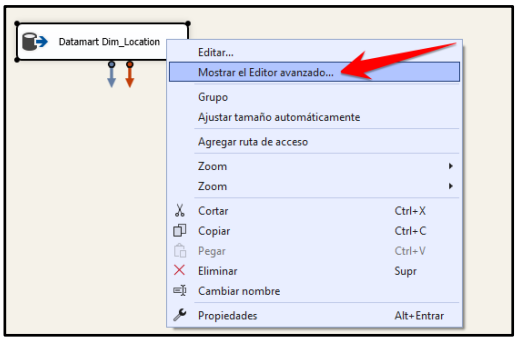


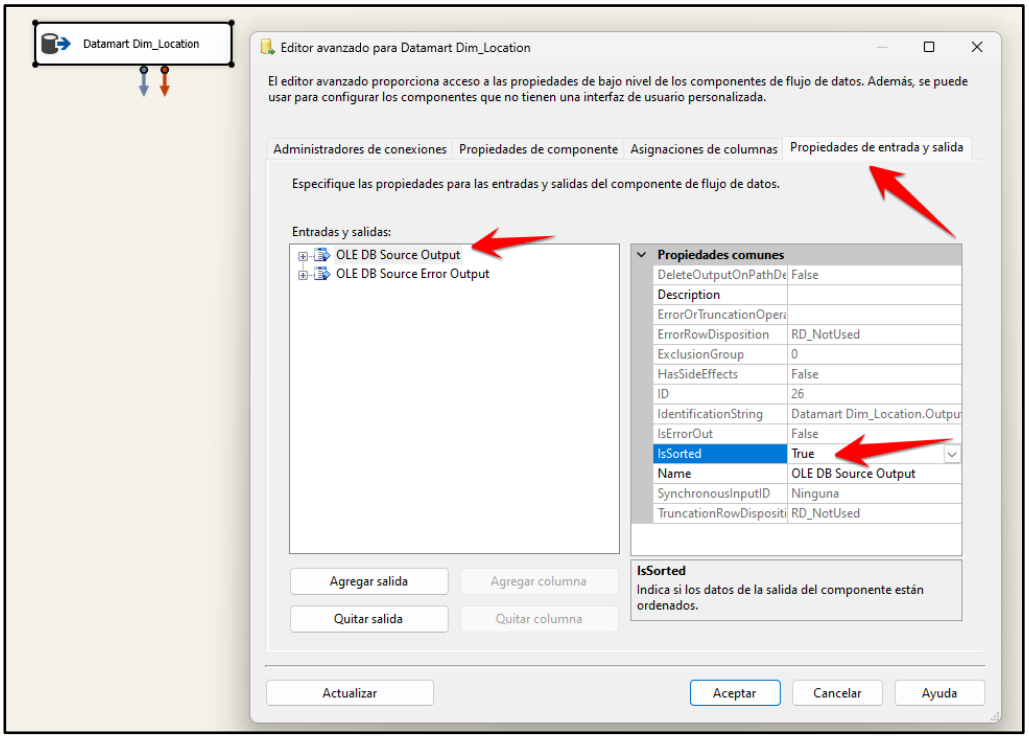

Staging S_Accidents
Editor avanzado para Staging S_Accidents
El editor avanzado proporciona acceso a las propiedades de bajo nivel de los componentes de flujo de datos. Además, se puede usar para configurar los componentes que no tienen una interfaz de usuario personalizada.
Administradores de conexiones
Propiedades de componente
Asignaciones de columnas
Propiedades de entrada y salida
Especifique las propiedades para las entradas y salidas del componente de flujo de datos.
Entradas y salidas:
OLE DB Source Output
OLE DB Source Error Output
Propiedades comunes
DeleteOutputOnPathDe
False
Description
ErrorOrTruncationOper
ErrorRowDisposition
RD_NotUsed
ExclusionGroup
0
HasSideEffects
False
ID
11
IdentificationString
Staging S_Accidents.Outputs[O
IsErrorOut
False
IsSorted
True
Name
OLE DB Source Output
SynchronousInputID
Ninguna
TruncationRowDispositi
RD_NotUsed
Agregar salida
Agregar columna
Quitar salida
Quitar columna
IsSorted
Indica si los datos de la salida del componente están ordenados.
Actualizar
Aceptar
Cancelar
Ayuda

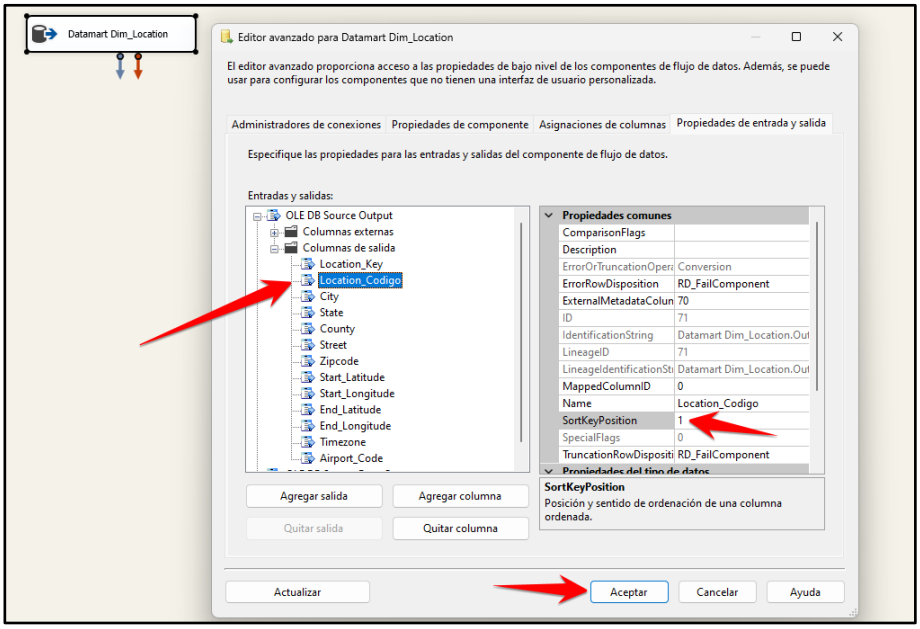

Staging S_Accidents
Editor avanzado para Staging S_Accidents
El editor avanzado proporciona acceso a las propiedades de bajo nivel de los componentes de flujo de datos. Además, se puede usar para configurar los componentes que no tienen una interfaz de usuario personalizada.
Administradores de conexiones
Propiedades de componente
Asignaciones de columnas
Propiedades de entrada y salida
Especifique las propiedades para las entradas y salidas del componente de flujo de datos.
Entradas y salidas:
OLE DB Source Output
Columnas externas
Columnas de salida
Id_Staging
City
State
County
Street
Zipcode
Start_Lat
Start_Lng
End_Lat
End_Lng
Timezone
Airport_Code
OLE DB Source Error Output
Propiedades comunes
ComparisonFlags
Description
ErrorOrTruncationOper
Conversion
ErrorRowDisposition
RD_FailComponent
ExternalMetadataColun
31
ID
32
IdentificationString
Staging S_Accidents.Output
LineageID
32
LineageIdentificationSt
Staging S_Accidents.Output
MappedColumnID
0
Name
Id_Staging
SortKeyPosition
1
SpecialFlags
0
TruncationRowDispositi
RD_FailComponent
Propiedades del tipo de datos
Agregar salida
Agregar columna
Quitar salida
Quitar columna
SortKeyPosition
Posición y sentido de ordenación de una columna ordenada.
Actualizar
Aceptar
Cancelar
Ayuda

d) Setting up second data source **DIM_LOCATION** .

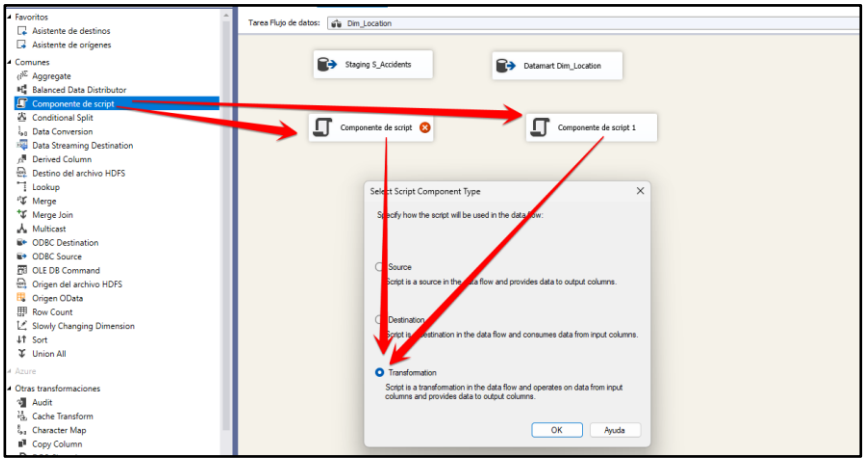




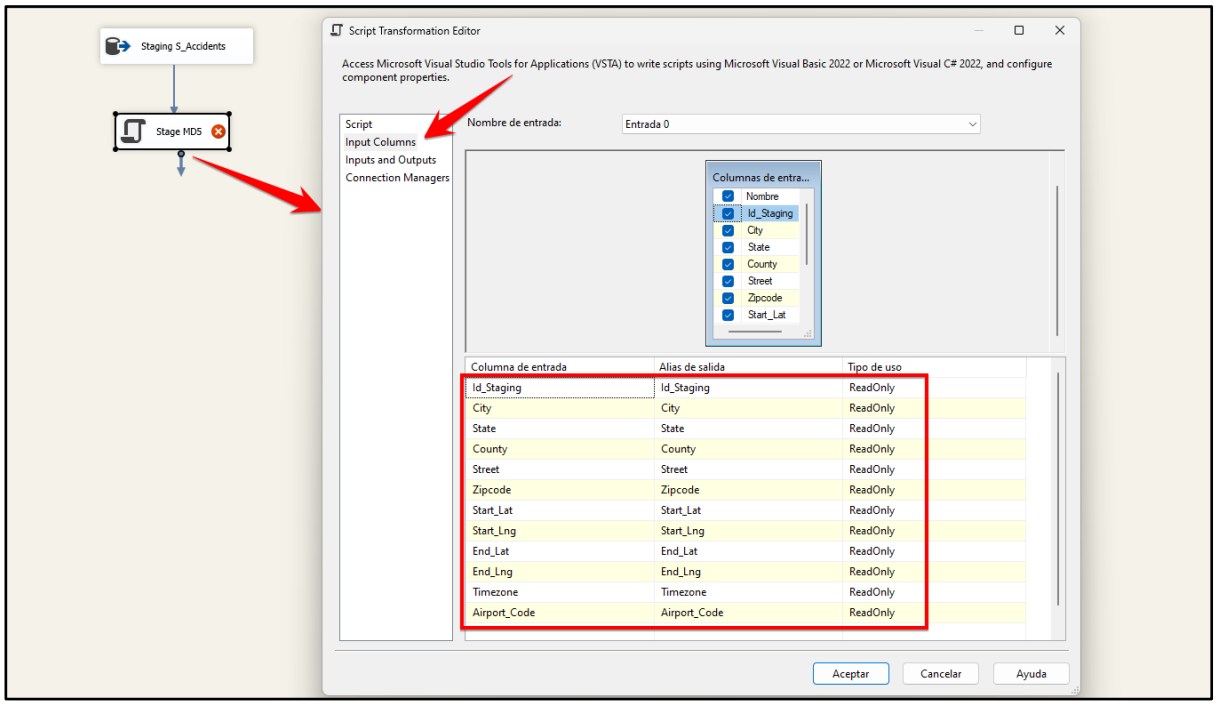


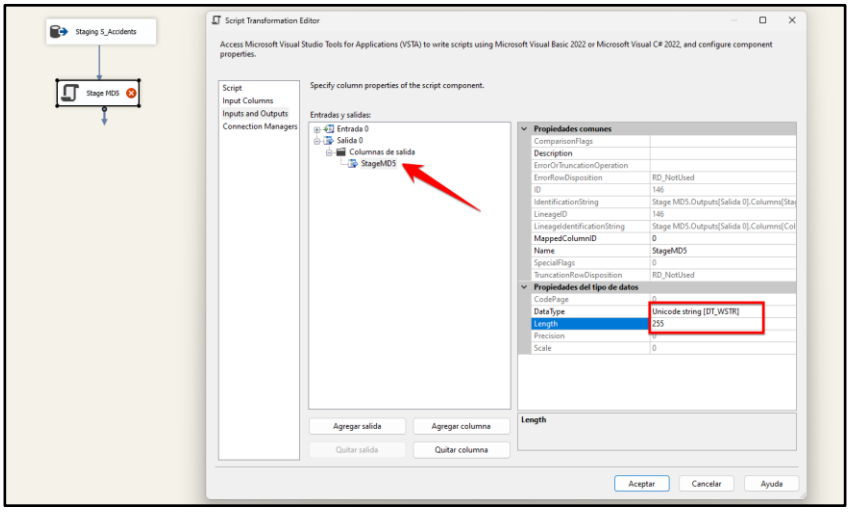


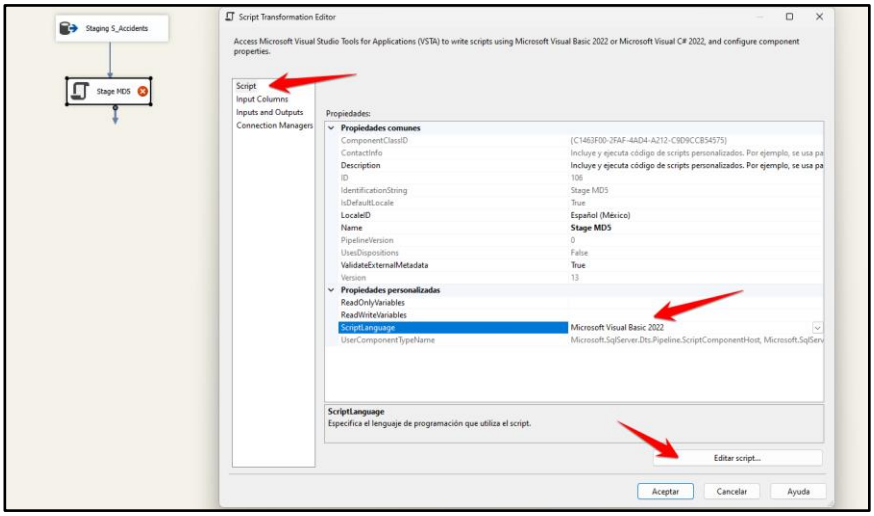
e) I will use two **Script Components** with the Transformation type.

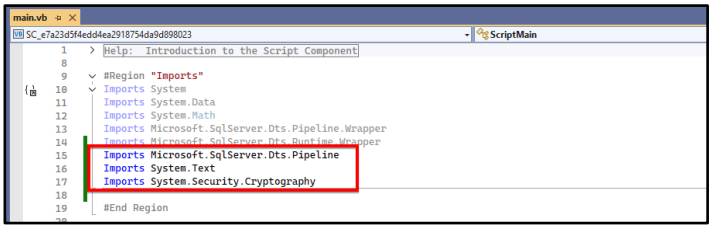


f) Setting Up the First Component:





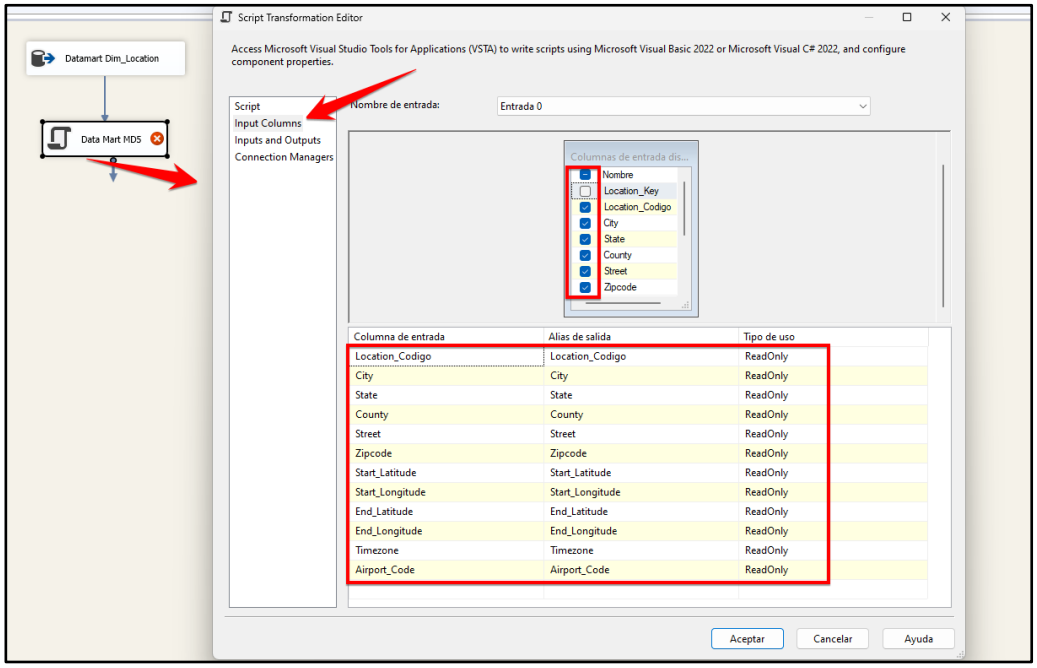


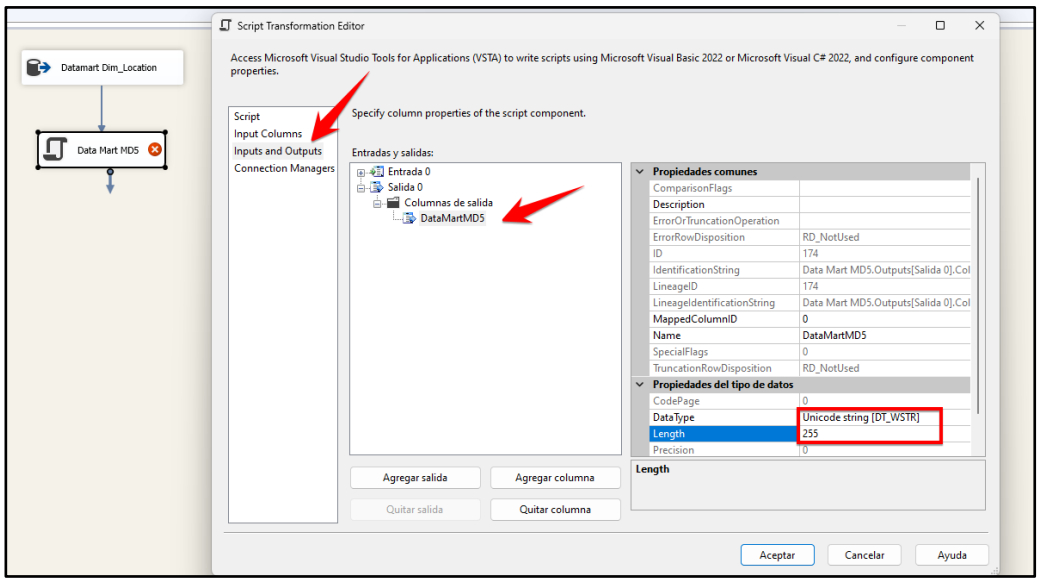


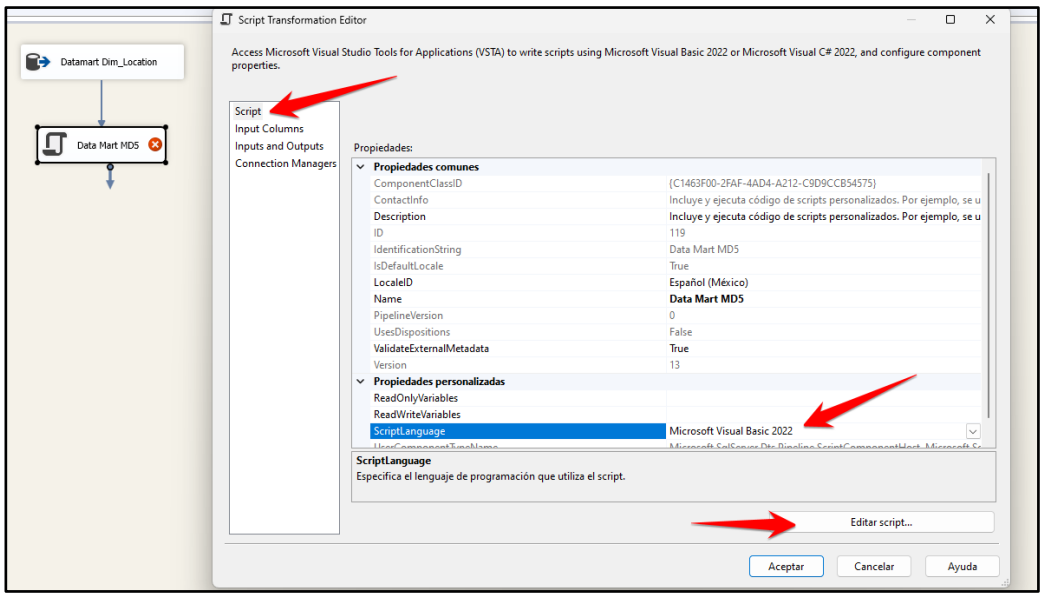

```
    Private inputBuffer As PipelineBuffer

    0 referencias
    Public Overrides Sub ProcessInput(ByVal InputID As Integer, ByVal Buffer As Microsoft.SqlServer.Dts.Pipeline.PipelineBuffer)

        inputBuffer = Buffer

        MyBase.ProcessInput(InputID, Buffer)

    End Sub

    1 referencia
    Public Shared Function CreateHash(ByVal data As String) As String
        Dim dataToHash As Byte() = (New UnicodeEncoding()).GetBytes(data)
        Dim md5 As MD5 = New MD5CryptoServiceProvider()
        Dim hashedData As Byte() = md5.ComputeHash(dataToHash)
        RNGCryptoServiceProvider.Create().GetBytes(dataToHash)
        Dim s As String = Convert.ToBase64String(hashedData, Base64FormattingOptions.None)
        Return s
    End Function

    2 referencias
    Public Overrides Sub Entrada0_ProcessInputRow(ByVal Row As Entrada0Buffer)
        Dim counter As Integer = 0
        Dim values As New StringBuilder
        For counter = 0 To inputBuffer.ColumnCount - 1
            Dim value As Object
            value = inputBuffer.Item(counter)
            values.Append(value)
        Next

        'CAMBIAR EL VALOR VariableSalida A SU COLUMNA DE SALIDA
        Row.StageMD5 = CreateHash(values.ToString())
    End Sub

End Class
```

Eng. Maximo Silva Parraguez

g) Setting up the Second Component.



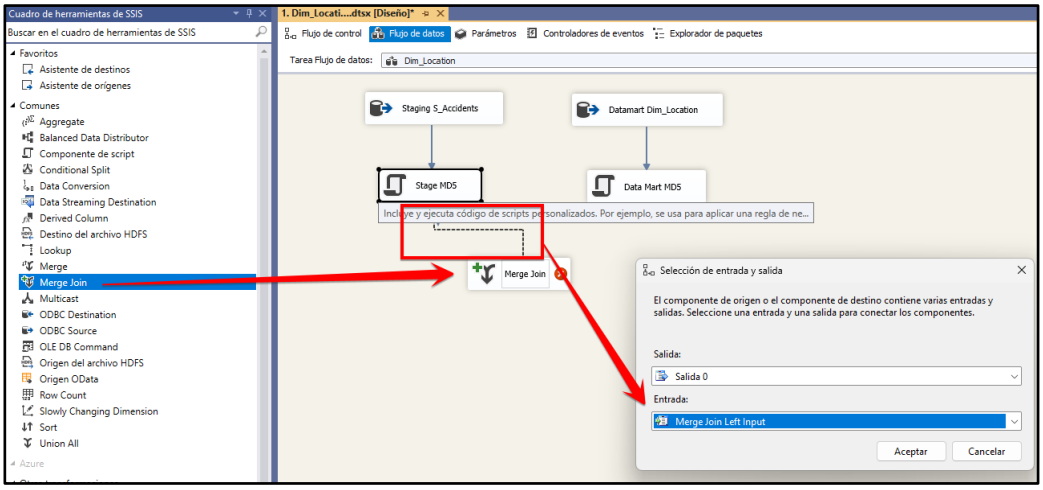


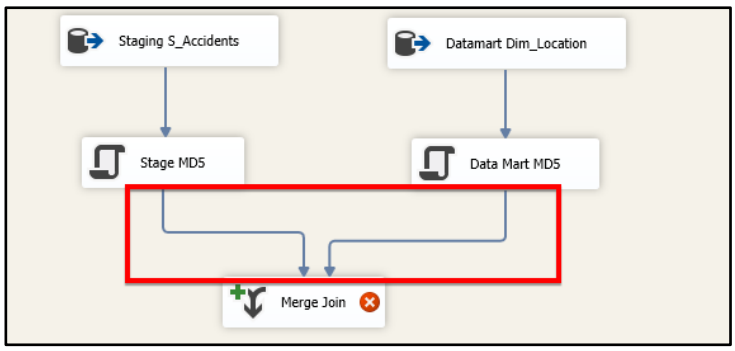


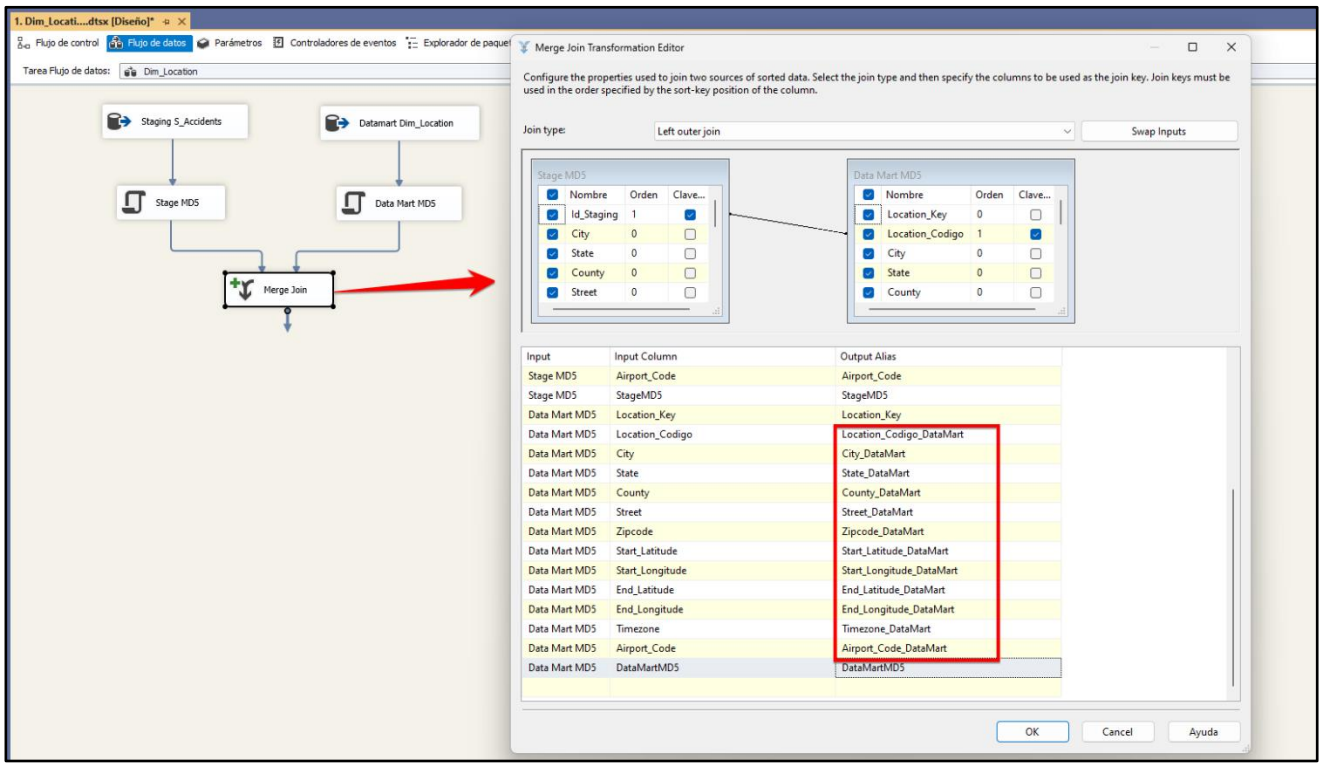
```
#Region "Imports"
Imports System
Imports System.Data
Imports System.Math
Imports Microsoft.SqlServer.Dts.Pipeline.Wrapper
Imports Microsoft.SqlServer.Dts.Runtime.Wrapper
Imports Microsoft.SqlServer.Dts.Pipeline
Imports System.Text
Imports System.Security.Cryptography
#End Region
```

```
    ' Row.ZipCode = zipCode

    Private inputBuffer As PipelineBuffer

    0 referencias
    Public Overrides Sub ProcessInput(ByVal InputID As Integer, ByVal Buffer As Microsoft.SqlServer.Dts.Pipeline.PipelineBuffer)

        inputBuffer = Buffer

        MyBase.ProcessInput(InputID, Buffer)

    End Sub

    1 referencia
    Public Shared Function CreateHash(ByVal data As String) As String
        Dim dataToHash As Byte() = (New UnicodeEncoding()).GetBytes(data)
        Dim md5 As MD5 = New MD5CryptoServiceProvider()
        Dim hashedData As Byte() = md5.ComputeHash(dataToHash)
        RNGCryptoServiceProvider.Create().GetBytes(dataToHash)
        Dim s As String = Convert.ToBase64String(hashedData, Base64FormattingOptions.None)
        Return s
    End Function

    2 referencias
    Public Overrides Sub Entrada0_ProcessInputRow(ByVal Row As Entrada0Buffer)
        Dim counter As Integer = 0
        Dim values As New StringBuilder
        For counter = 0 To inputBuffer.ColumnCount - 1
            Dim value As Object
            value = inputBuffer.Item(counter)
            values.Append(value)
        Next

        'CAMBIAR EL VALOR VariableSalida A SU COLUMNA DE SALIDA
        Row.DataMartMD5 = CreateHash(values.ToString())
    End Sub

End Class
```

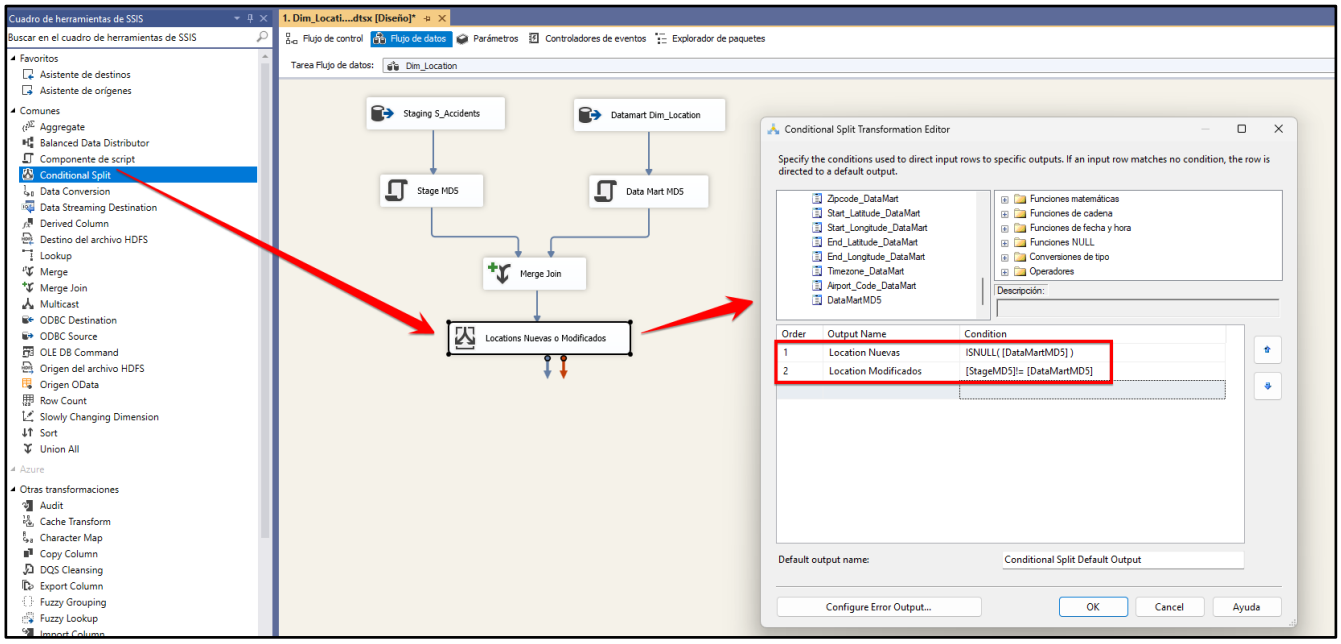
h) **We use the Merge Combination** component **Join .**



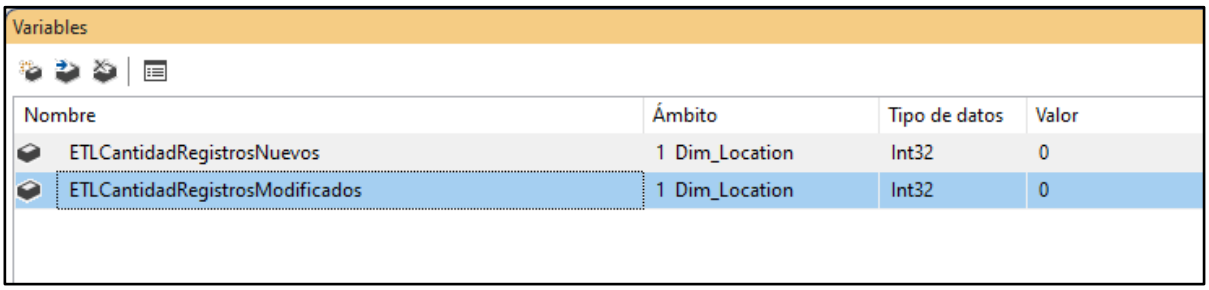




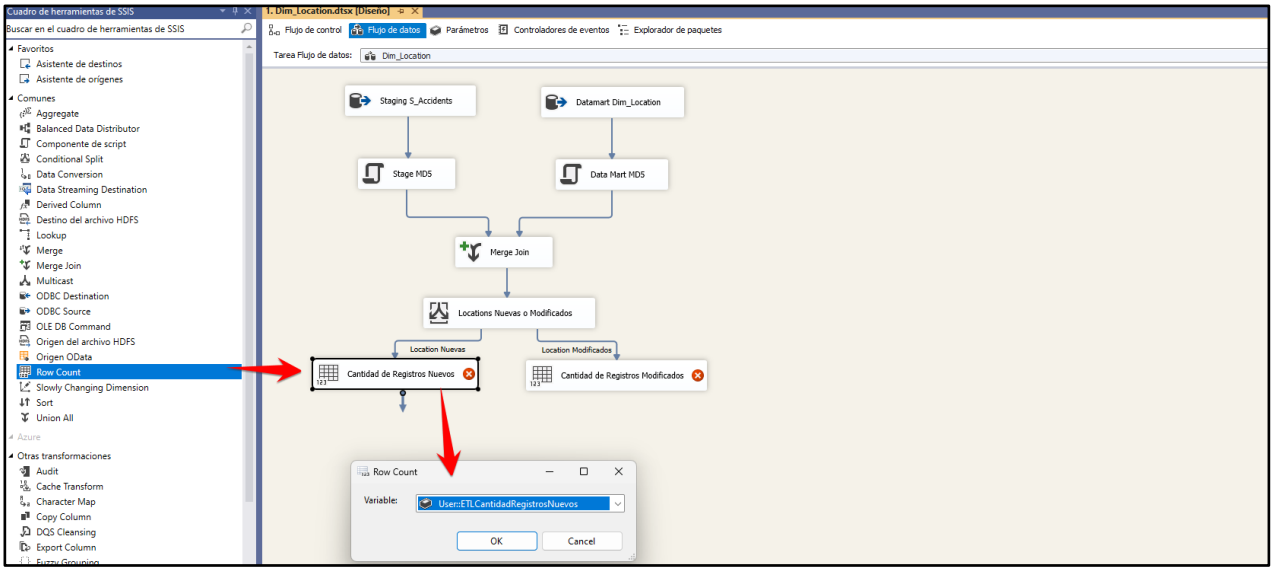
i) I will use the **Conditional Division component** to compare the data and add it to New Records or Modified Records as appropriate.

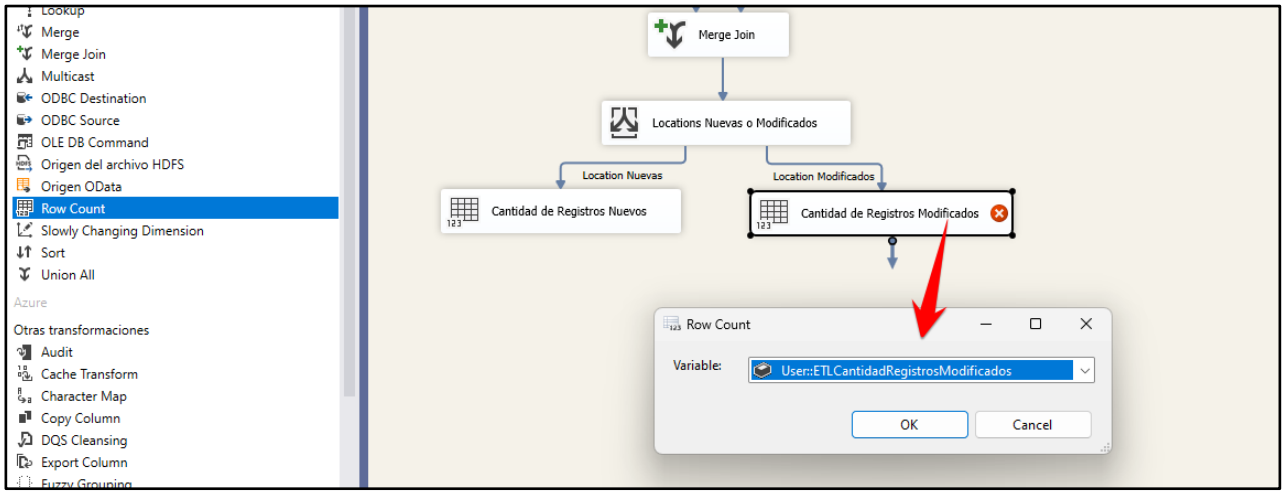


j) I created two variables to use in the next step:

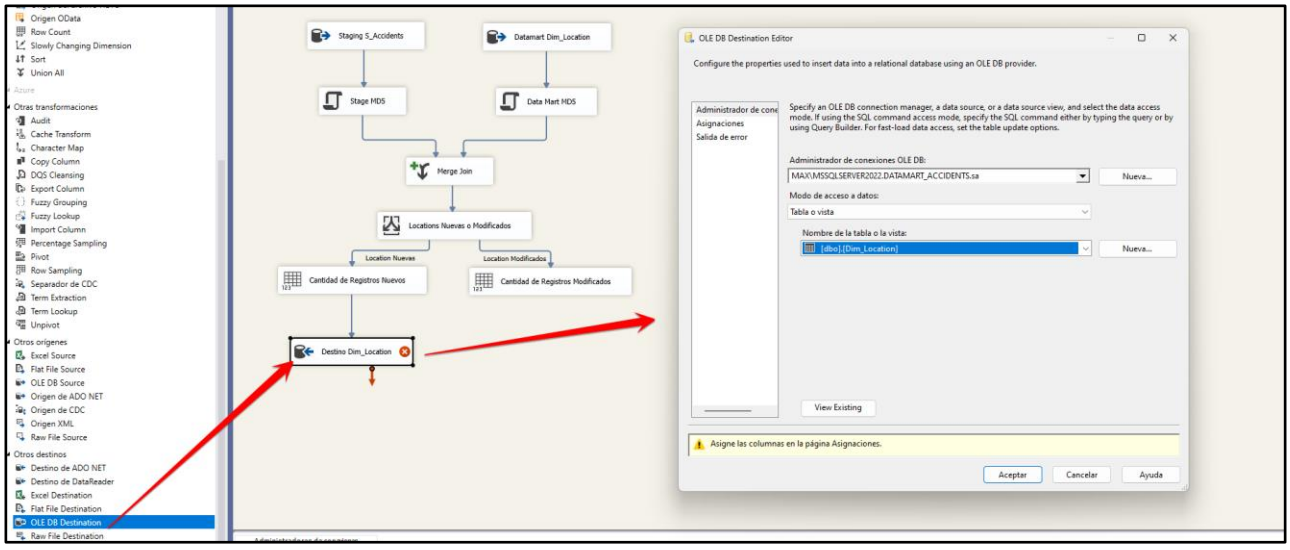


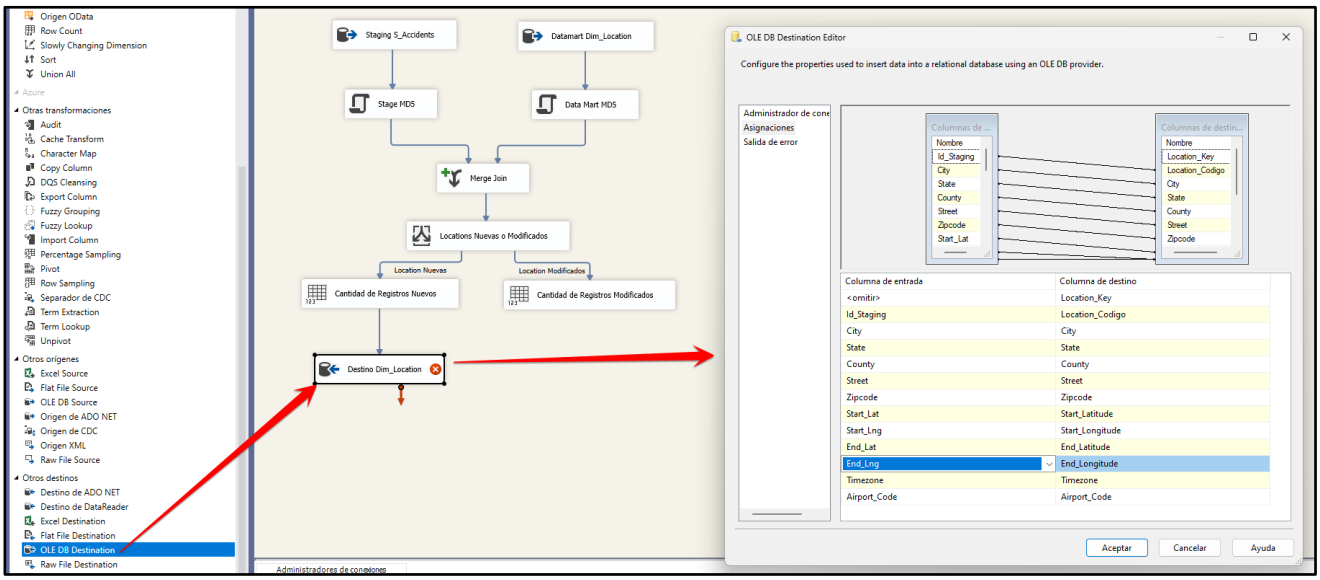
k) I used the **Row component Count** to be able to count the number of new or modified records that will be added.





l) Now we configure the **destination OLE DB to the Dim_Location table** where non-new data will be stored.

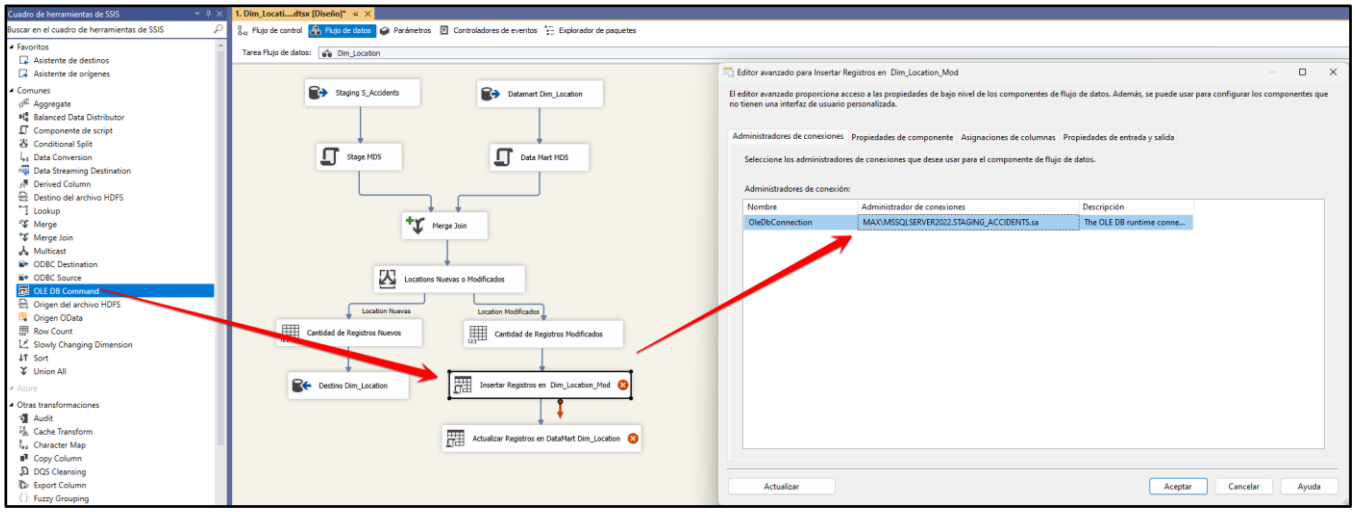


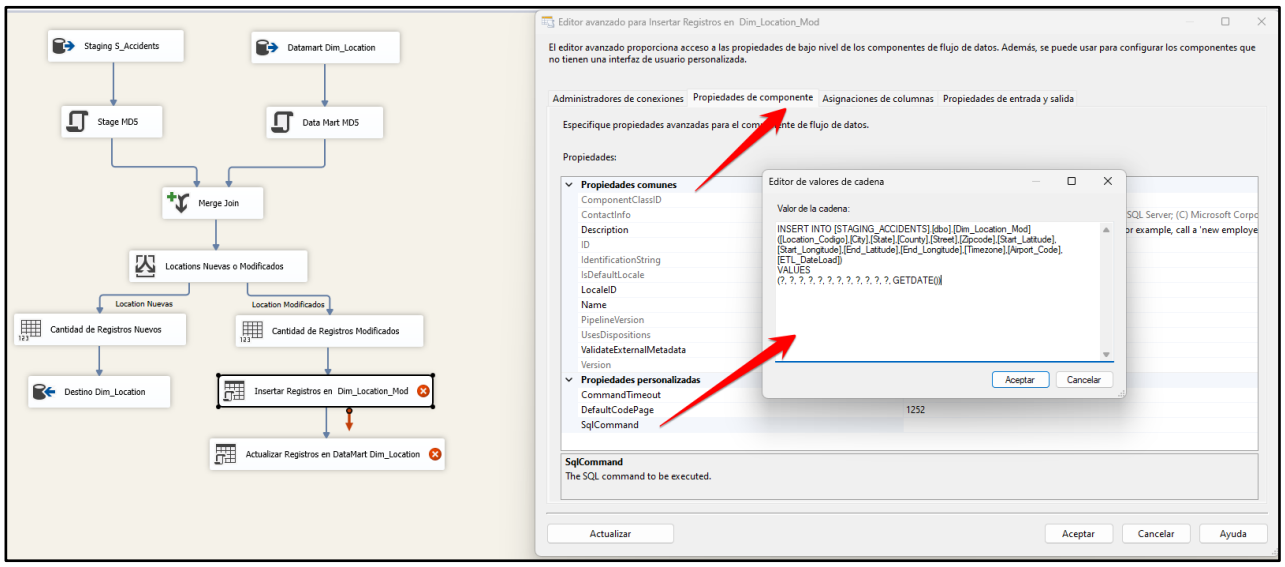


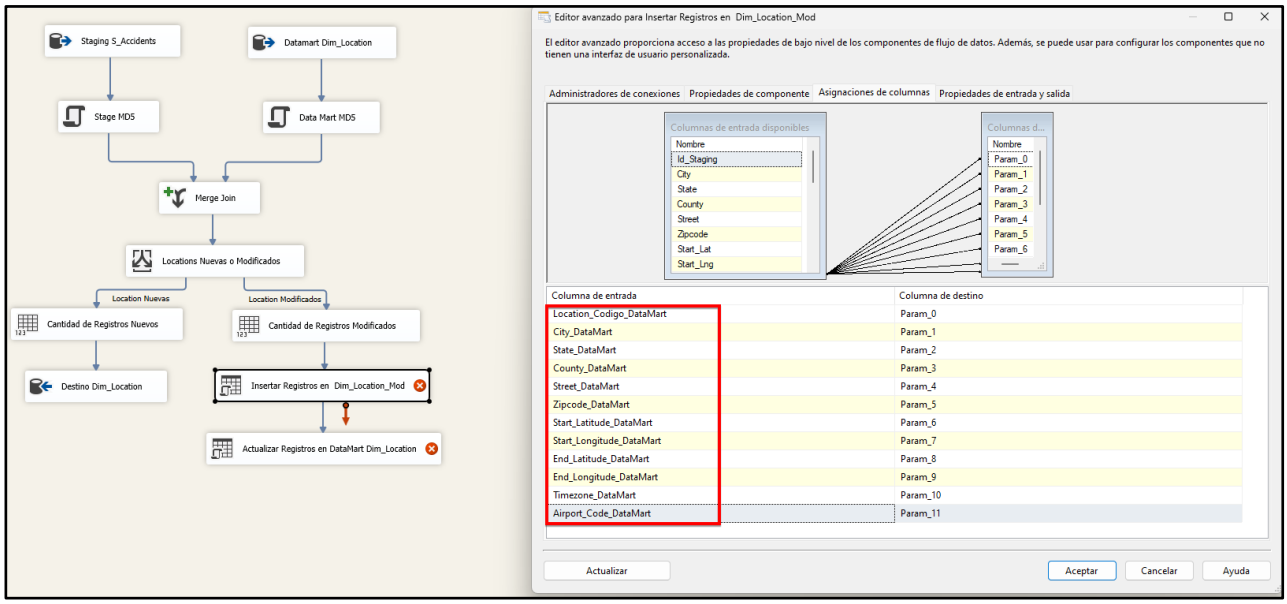
m) **I created a table in the STAGING_ACCIDENTS Database, the Dim_Location_Mod table** , where the records that have been modified will be recorded.

```
CREATE TABLE [STAGING_ACCIDENTS].[dbo].[Dim_Location_Mod] (
    Location_Key INT PRIMARY KEY IDENTITY(1,1),
    Location_Codigo INT,
    City VARCHAR(100),
    State VARCHAR(50),
    County VARCHAR(100),
    Street VARCHAR(500),
    Zipcode VARCHAR(50),
    Start_Latitude FLOAT,
    Start_Longitude FLOAT,
    End_Latitude FLOAT,
    End_Longitude FLOAT,
    Timezone VARCHAR(50),
    Airport_Code VARCHAR(50),
    ETL_DateLoad DATETIME
);
```

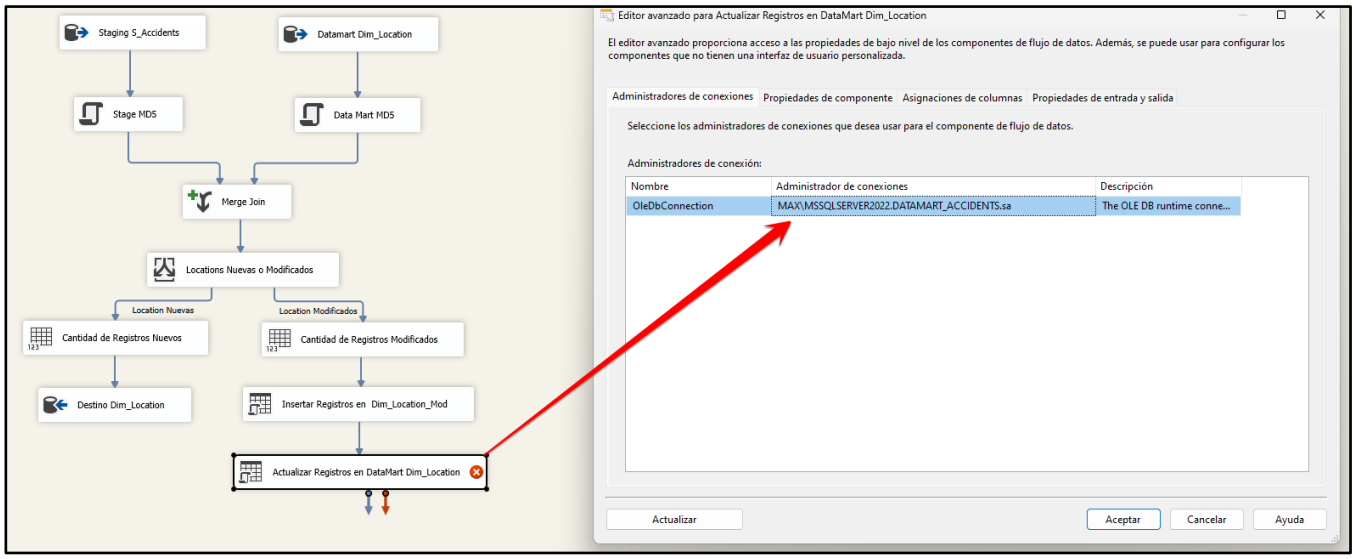
n) I inserted **two OLE DB Command components** , the first one will add the modified records into the Dim_Location_Mod table .

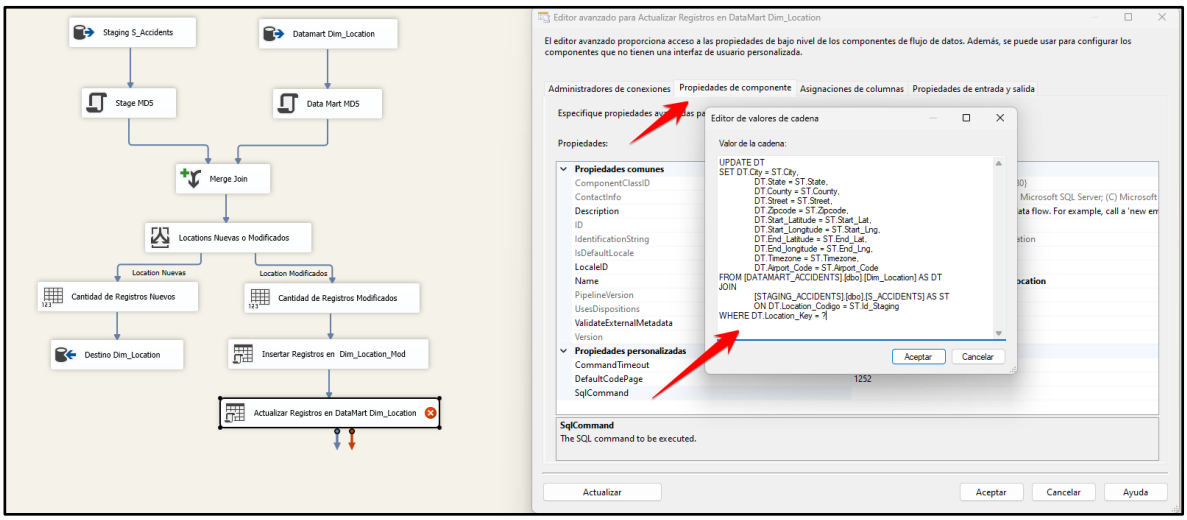


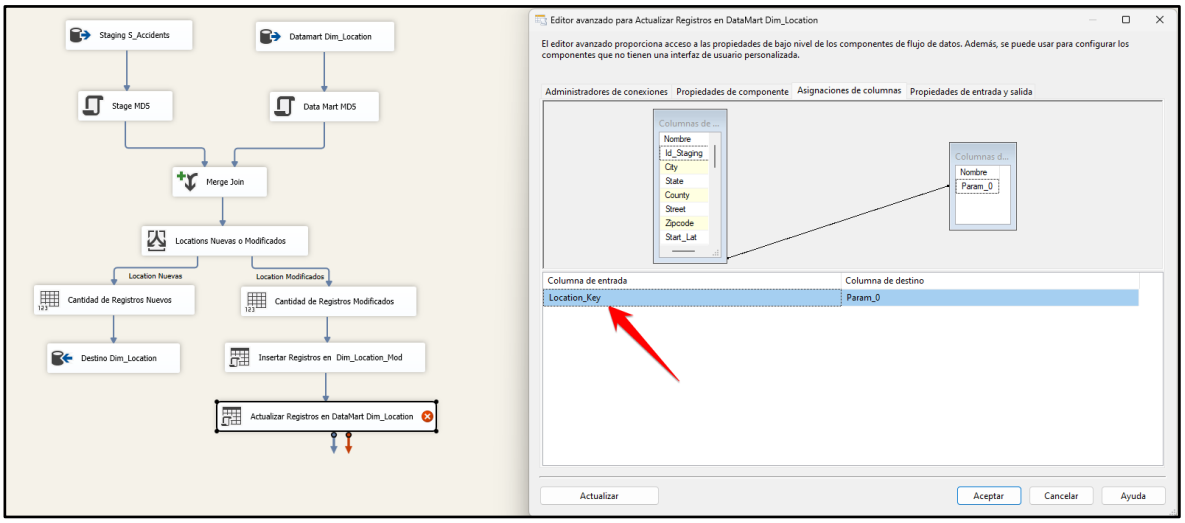




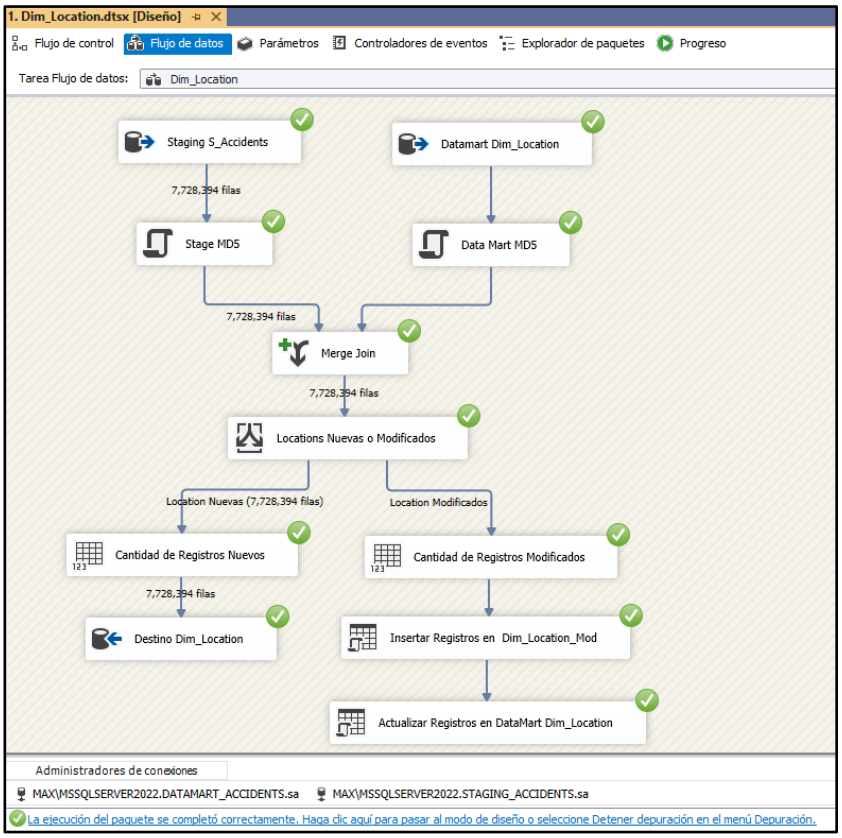
o) The second OLE DB Component will update the records in **Dim_Location .**







p) **Dim_Location** project , and **repeat the same process with the other dimensions.**



2. Populating the Climate Dimension:

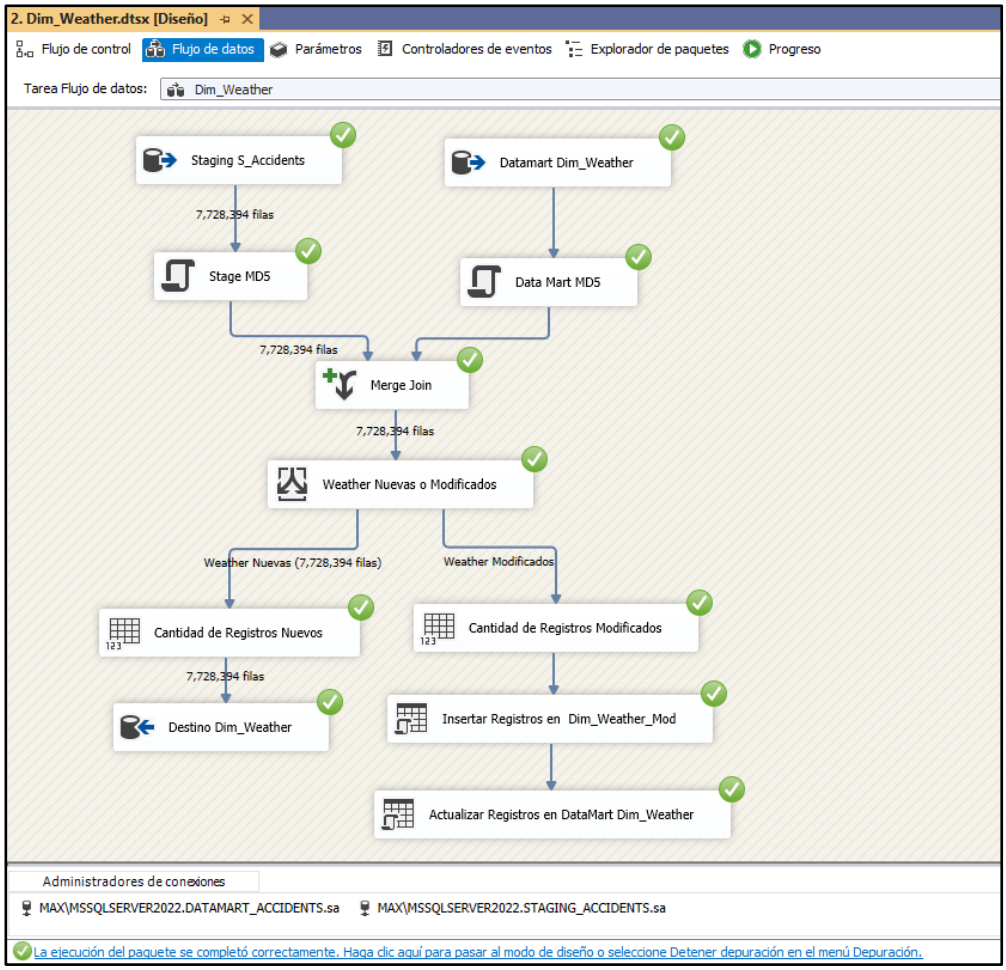
a) Creating the Table **Dim_Weather .**

```
-- Dimensión: Clima
CREATE TABLE Dim_Weather (
    Weather_Key INT PRIMARY KEY IDENTITY(1,1),
    Weather_Codigo INT,
    Temperature FLOAT,
    Wind_Chill FLOAT,
    Humidity FLOAT,
    Pressure FLOAT,
    Visibility FLOAT,
    Wind_Direction VARCHAR(50),
    Wind_Speed FLOAT,
    Precipitation FLOAT,
    Weather_Condition VARCHAR(100)
);
```

b) Creating Table **Dim_Weather_Mod .**

```
CREATE TABLE [STAGING_ACCIDENTS].[dbo].[Dim_Weather_Mod] (
    Weather_Key INT PRIMARY KEY IDENTITY(1,1),
    Weather_Codigo INT,
    Temperature FLOAT,
    Wind_Chill FLOAT,
    Humidity FLOAT,
    Pressure FLOAT,
    Visibility FLOAT,
    Wind_Direction VARCHAR(50),
    Wind_Speed FLOAT,
    Precipitation FLOAT,
    Weather_Condition VARCHAR(100),
    ETL_DateLoad DATETIME
);
```

c) **Running the project** .



3. Populating the Road Features Dimension:

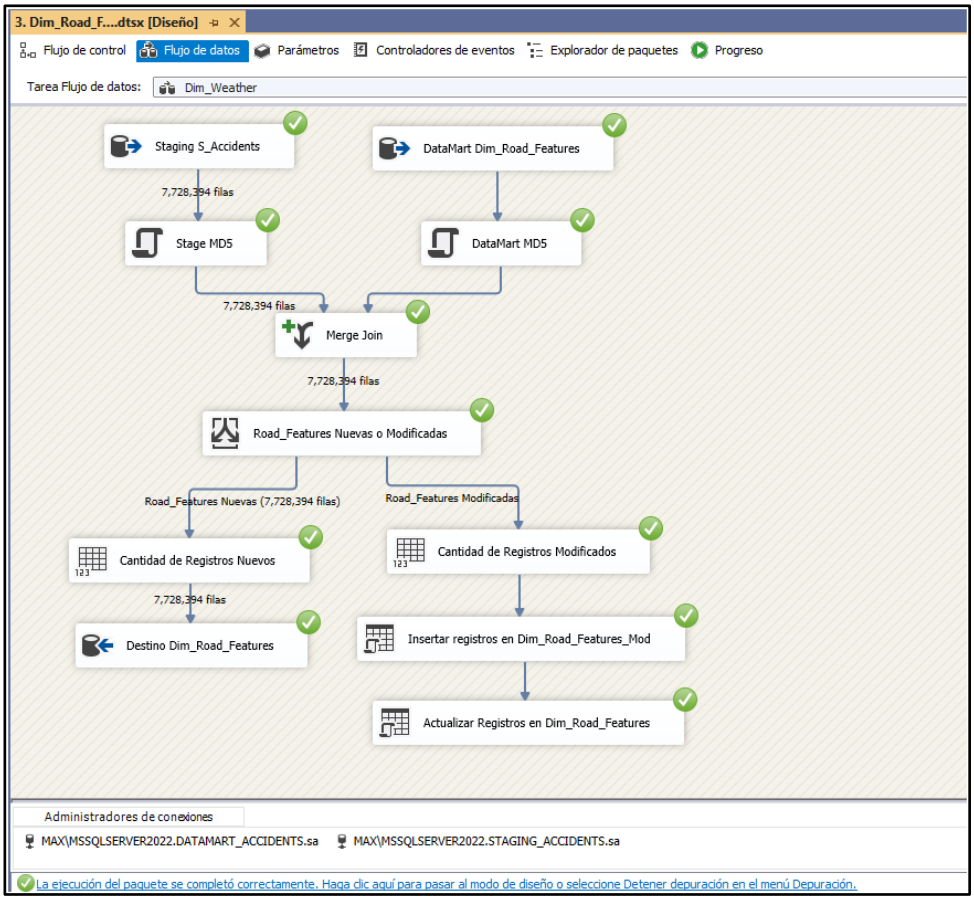
a) Creating the Table **Dim_Road_Features .**

```
CREATE TABLE Dim_Road_Features (
    Road_Features_Key INT PRIMARY KEY IDENTITY(1,1),
    Road_Features_Codigo INT,
    Bump VARCHAR(10),
    Crossing VARCHAR(10),
    Give_Way VARCHAR(10),
    Junction VARCHAR(10),
    No_Exit VARCHAR(10),
    Roundabout VARCHAR(10),
    Stop VARCHAR(10),
    Traffic_Signal VARCHAR(10),
    Turning_Loop VARCHAR(10)

);
```

b) Creating Table **Dim_Road_Features_Mod .**

```
CREATE TABLE [STAGING_ACCIDENTS].[dbo].[Dim_Road_Features_Mod] (
    Road_Features_Key INT PRIMARY KEY IDENTITY(1,1),
    Road_Features_Codigo INT,
    Bump VARCHAR(10),
    Crossing VARCHAR(10),
    Give_Way VARCHAR(10),
    Junction VARCHAR(10),
    No_Exit VARCHAR(10),
    Roundabout VARCHAR(10),
    Stop VARCHAR(10),
    Traffic_Signal VARCHAR(10),
    Turning_Loop VARCHAR(10),
    ETL_DateLoad DATETIME
);
```

c) **Running the project** .



4. Populating the Accident Description Dimension:

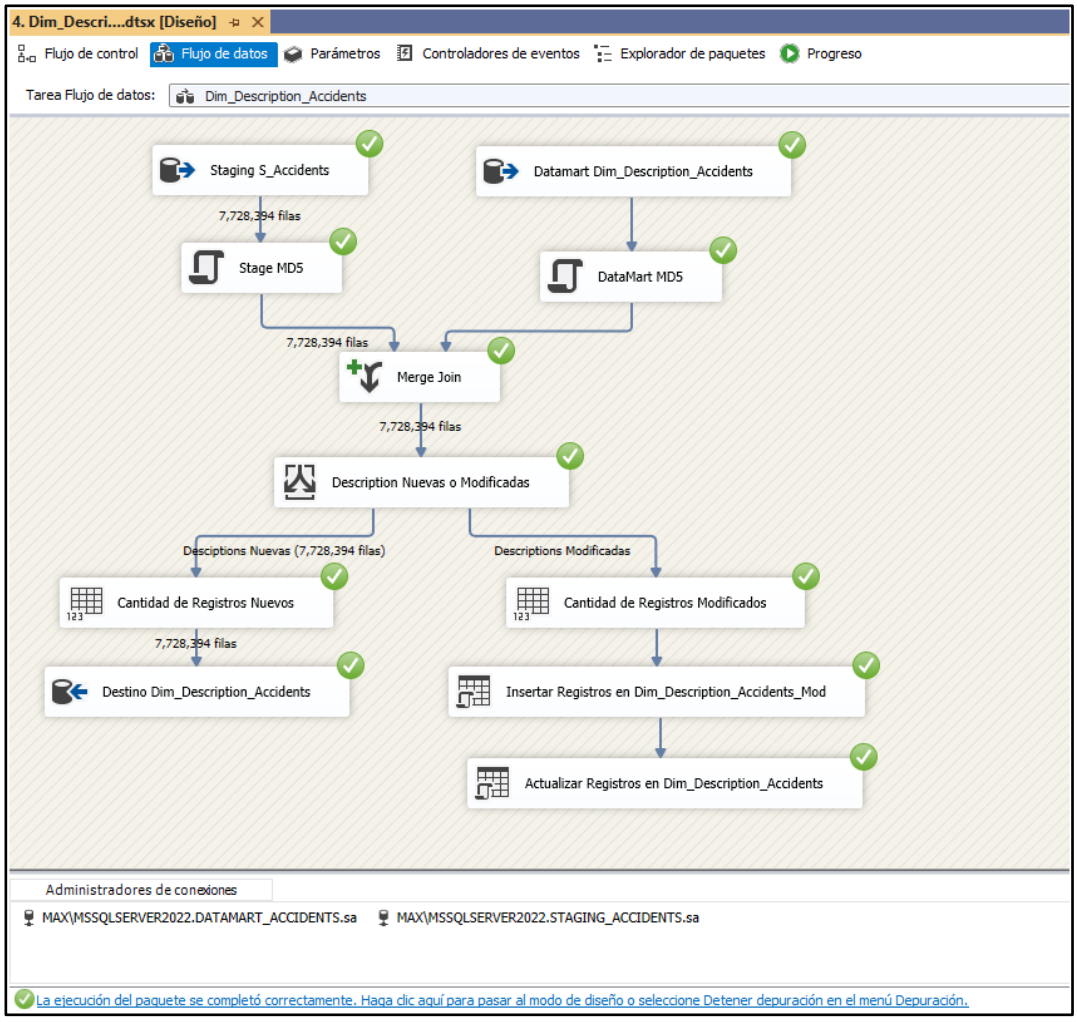
a) Creating the Table **Dim_Description_Accidents .**

```
CREATE TABLE Dim_Description_Accidents (
    Description_Accidents_Key INT PRIMARY KEY IDENTITY(1,1),
    Description_Accidents_Codigo INT,
    Description VARCHAR(800)
);
```

b) Creating Table **Dim_Description_Accidents_Mod .**

```
CREATE TABLE [STAGING_ACCIDENTS].[dbo].[Dim_Description_Accidents_Mod] (
    Description_Accidents_Key INT PRIMARY KEY IDENTITY(1,1),
    Description_Accidents_Codigo INT,
    Description VARCHAR(800),
    ETL_DateLoad DATETIME
);
```

c) **Running the project** .



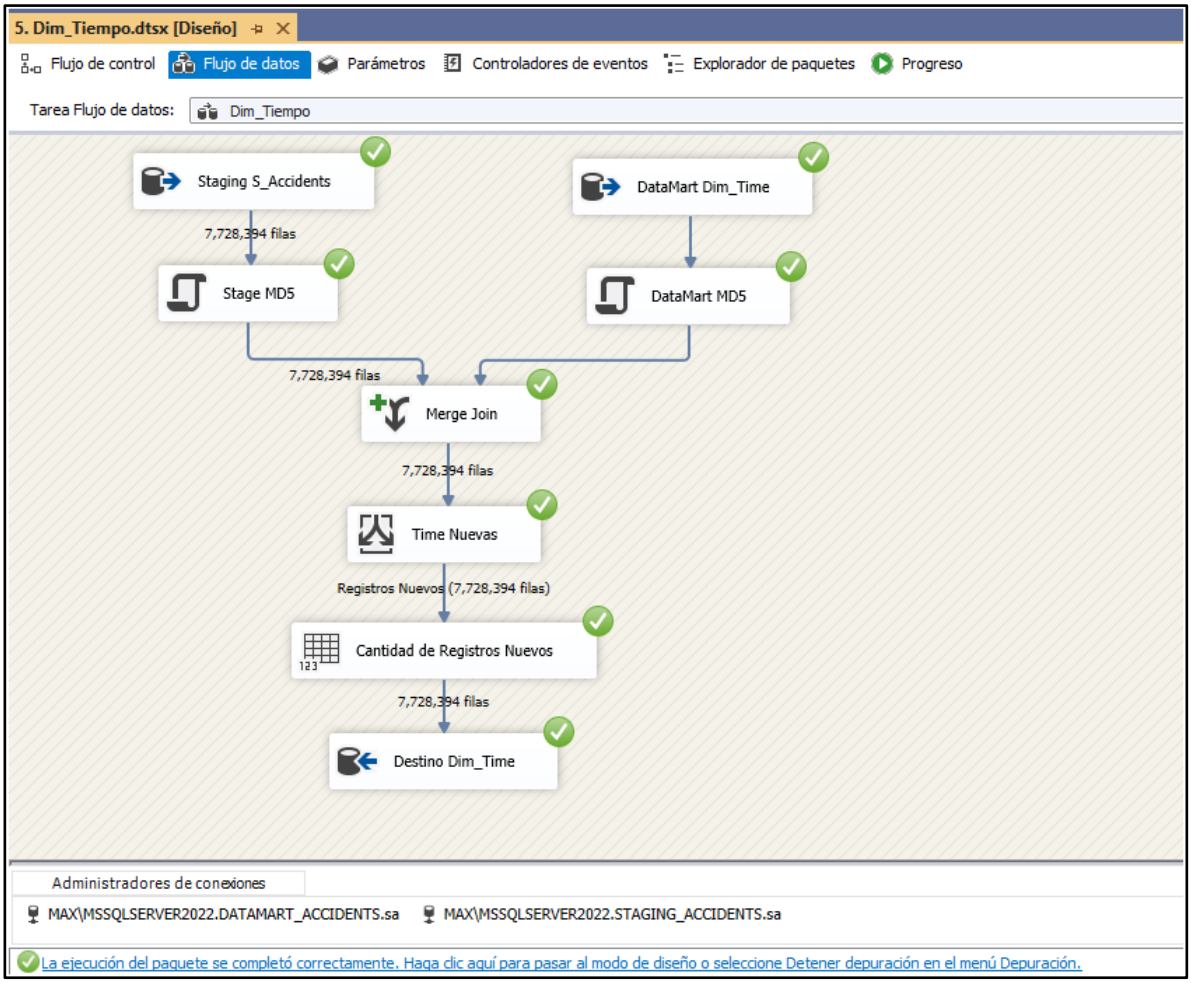
5. Populating the Time Dimension:

a) Creating the Table **Dim_Time .**

```
CREATE TABLE [DATAMART_NORTHWND].[dbo].[Dim_Time](
    [Tiempo_Skey] INT IDENTITY(1,1) NOT NULL,
    [Tiempo_Codigo] INT,
    [Tiempo_FechaInicio] DATETIME,
    [Tiempo_FechaFin] DATETIME,
    [Tiempo_Anio] INT,
    [Tiempo_Trimestre] INT,
    [Tiempo_Mes] INT,
    [Tiempo_Semana] INT,
    [Tiempo_DiaDeAnio] INT,
    [Tiempo_DiaDeMes] INT,
    [Tiempo_DiaDeSemana] INT,
    [Tiempo_EsFinSemana] INT,
    [Tiempo_EsFeriado] INT,
    [Tiempo_Comentarios] VARCHAR(20),
    [Tiempo_SemanaCalendario] INT,
    [Tiempo_SemanasDelAnioLaborales] INT,
    [Tiempo_AnioBisiesto] INT,
    [Hour_Inicio] TIME,
    [Hour_Fin] TIME
);
```

b) In the Time dimension there will only be New Records and NOT Modified Records so I will not create a table for the modified ones **.**

c) **Running the project** .



6. Populating the Table Facts:

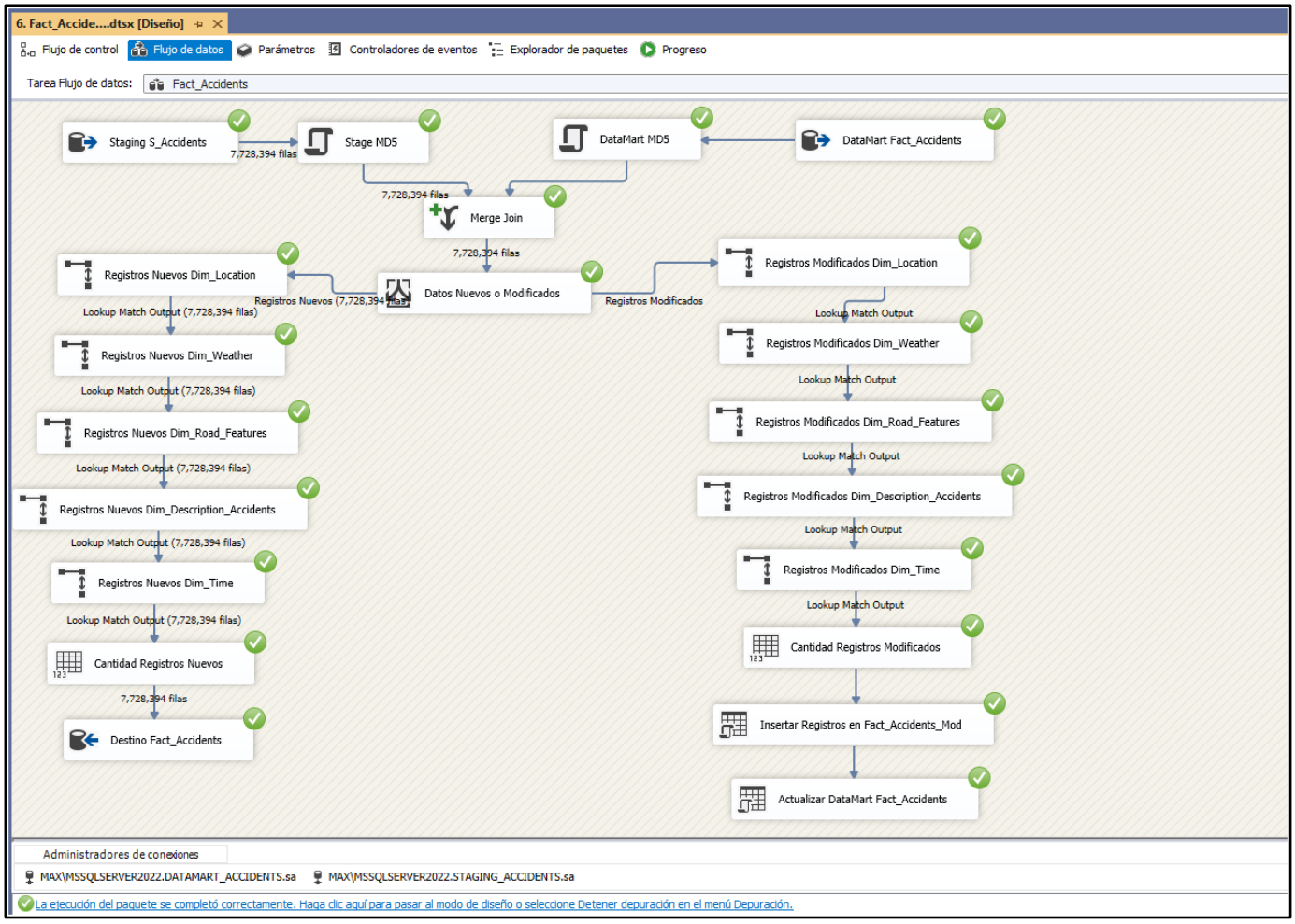
a) Creating the Table **Fact_Accidents .**

```
CREATE TABLE Fact_Accidents (
    Fact_Accidents_Key INT PRIMARY KEY IDENTITY(1,1),
    Location_Key INT NOT NULL,
    Weather_Key INT NOT NULL,
    Road_Features_Key INT NOT NULL,
    Description_Accidents_Key INT NOT NULL,
    Tiempo_Skey INT NOT NULL,
    Distance FLOAT,
    Severity INT,
    Duration_minutes INT
    FOREIGN KEY (Location_Key) REFERENCES Dim_Location(Location_Key),
    FOREIGN KEY (Weather_Key) REFERENCES Dim_Weather(Weather_Key),
    FOREIGN KEY (Road_Features_Key) REFERENCES Dim_Road_Features(Road_Features_Key),
    FOREIGN KEY (Description_Accidents_Key) REFERENCES Dim_Description_Accidents(Description_Accidents_Key),
    FOREIGN KEY (Tiempo_Skey) REFERENCES Dim_Time(Tiempo_Skey)
);
```

b) Creating Table **Fac_Accidents_Mod** .

```
CREATE TABLE [STAGING_ACCIDENTS].[dbo].[Fact_Accidents_Mod] (
    Fact_Accidents_Key INT PRIMARY KEY IDENTITY(1,1),
    Location_Key INT NOT NULL,
    Weather_Key INT NOT NULL,
    Road_Features_Key INT NOT NULL,
    Description_Accidents_Key INT NOT NULL,
    Tiempo_Skey INT NOT NULL,
    Distance FLOAT,
    Severity INT,
    Duration_minutes INT,
    ETL_DateLoad DATETIME
);
```

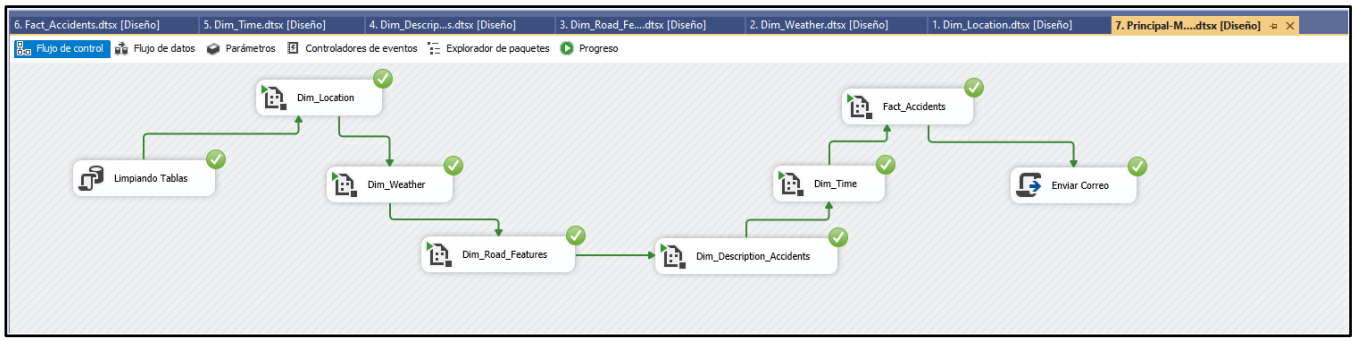
c) **Running the project** .



7. Main Project and Mail Sending.

In this last step, I will create a **Project called PRINCIPAL-MASTER** , which will be in charge of **Cleaning the Tables** (Considering that it will be the first time that the data will be passed and since I had test data, I prefer to delete it and run everything from the beginning **), then it will execute each Dimension Population and from the Facts table** , and **at the end it will send me an email indicating if everything was done correctly** . It is important to mention that I sent the email using the **Script Task component,** because it will be done locally, in a production environment the Send Mail Task Component would be used using a mail server and its respective configuration.

- We executed the entire project.



- It can be seen that everything was executed correctly, therefore, I received the email indicating that the entire process was completed successfully.