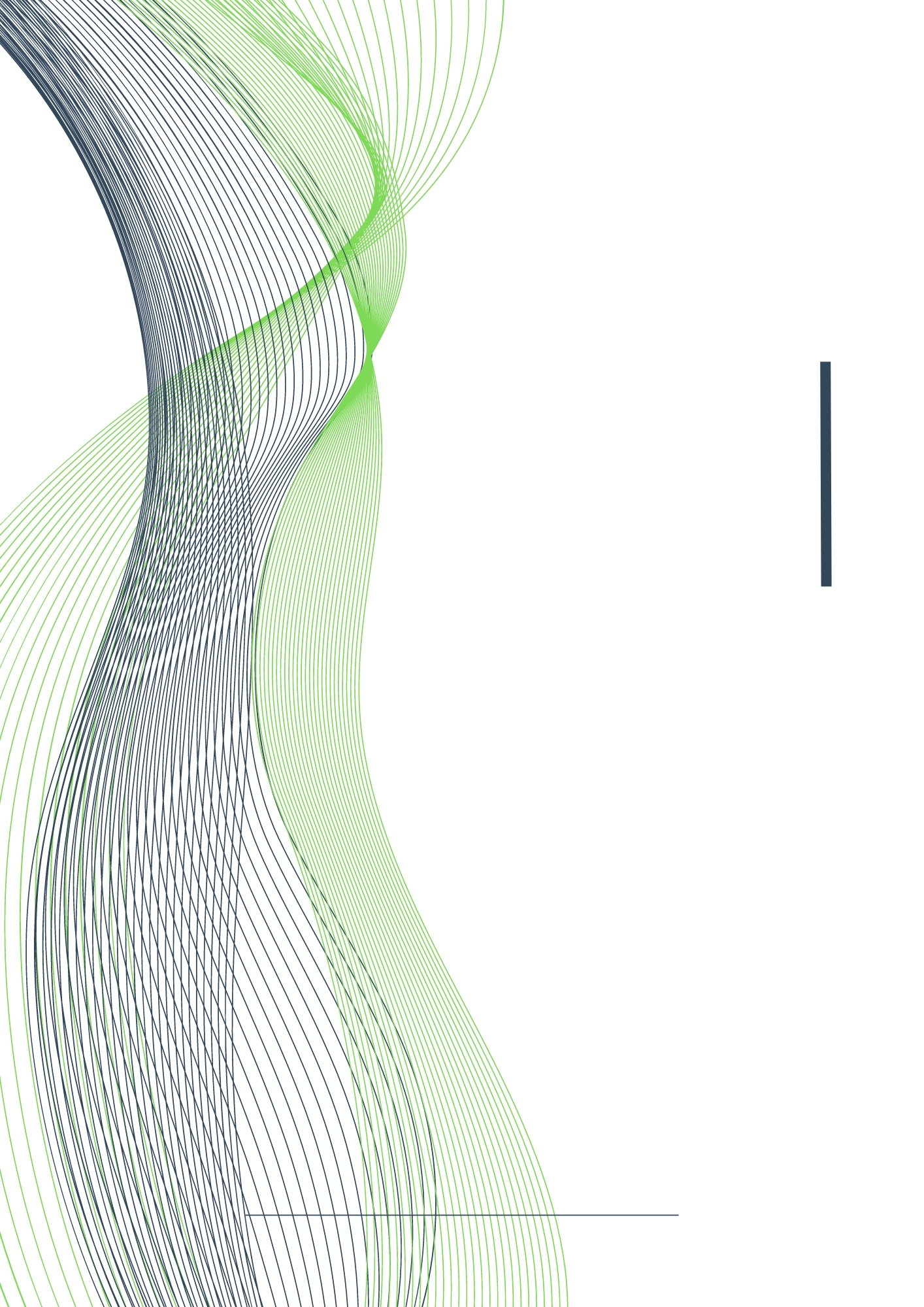
****

REAL-TIME DATA INTEGRATION PROJECT FROM AN API TO SQL SERVER USING SSIS

**ENG. Silva Parraguez Maximo**

1. **UNDERSTANDING THE DATA AND THE PROJECT**

**Real-time data will be obtained from the OpenWeatherMap API** , which provides real-time weather data such as temperature, humidity, wind speed, etc.

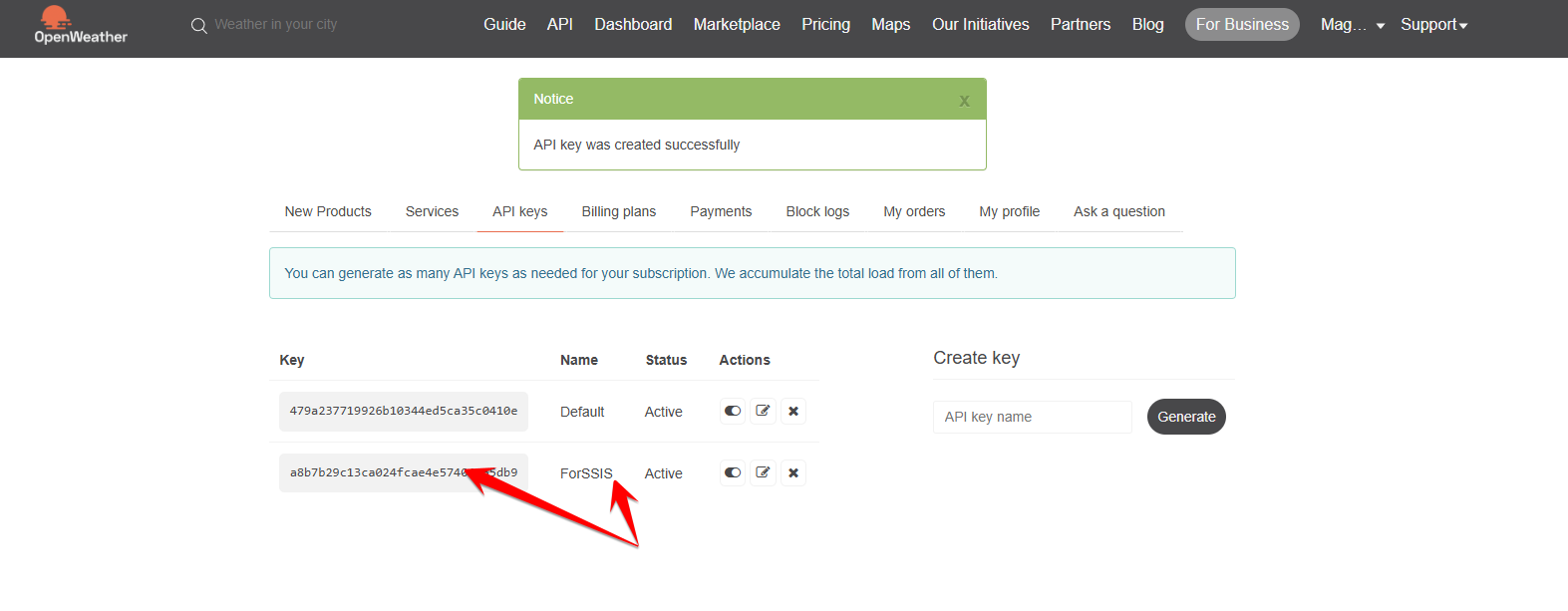
The purpose of this project is to create an ETL process in SSIS that captures, transforms, and loads real-time data from a continuous data source (such as the OpenWeatherMap API) for a specific city (or multiple cities), into a SQL Server database.

1. **PROJECT DEVELOPMENT**

**Step 1: Obtaining an OpenWeatherMap API Key**

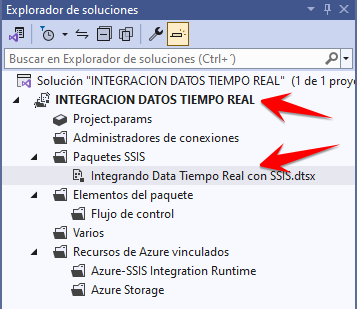
Since I will extract the data from an API that asks for an API Key, I will need to create an account on the Web that will provide it to me. If there were another API that did not ask for that key, for example later, in ***Step 5: Editing the Script with the C# code,*** instead of passing the APIKey to the URL that builds the API, only the API link would be passed directly (For example: <https://api.coindesk.com/v1/bpi/currentprice.json>) (Free API that provides real-time Bitcoin Price Index (BPI) information).

Once registered on the OpenWeatherMap website ( <https://openweathermap.org/api>), we generate a new API Key that you will use to obtain the Data.



**Step 2: Creating Project in SSIS**

The project was created with the name: **“REAL-TIME DATA INTEGRATION”** and an SSIS package called: **“Integrating Real-Time Data with SSIS”.**

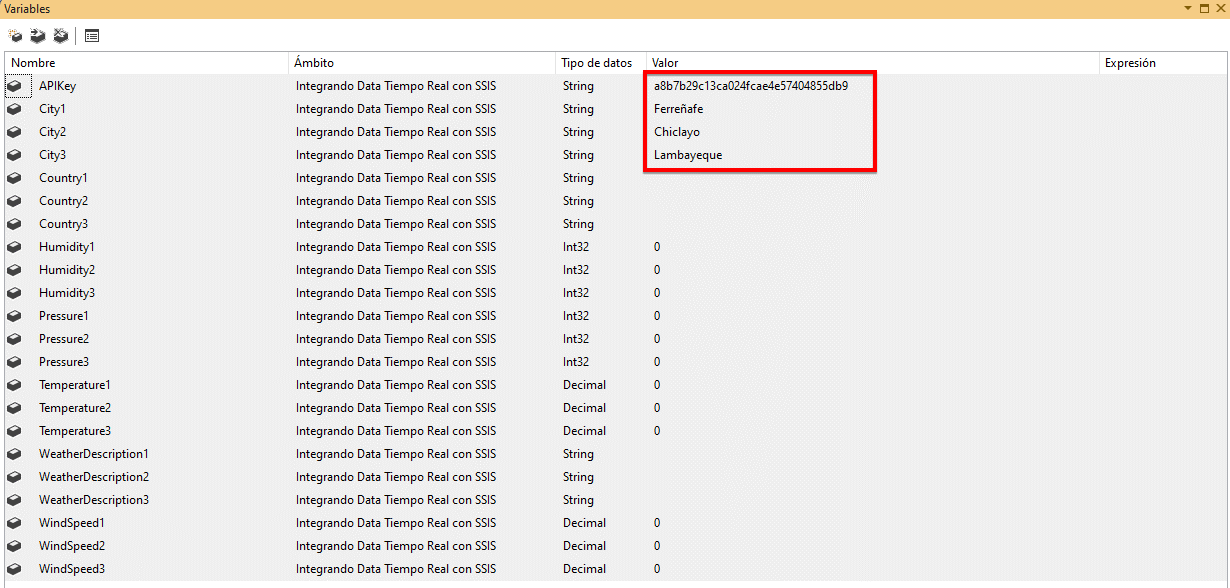


**Step 3: Setting Up Variables in SSIS**

I will create variables that will be responsible for extracting and storing the data obtained from the API. The variables are:

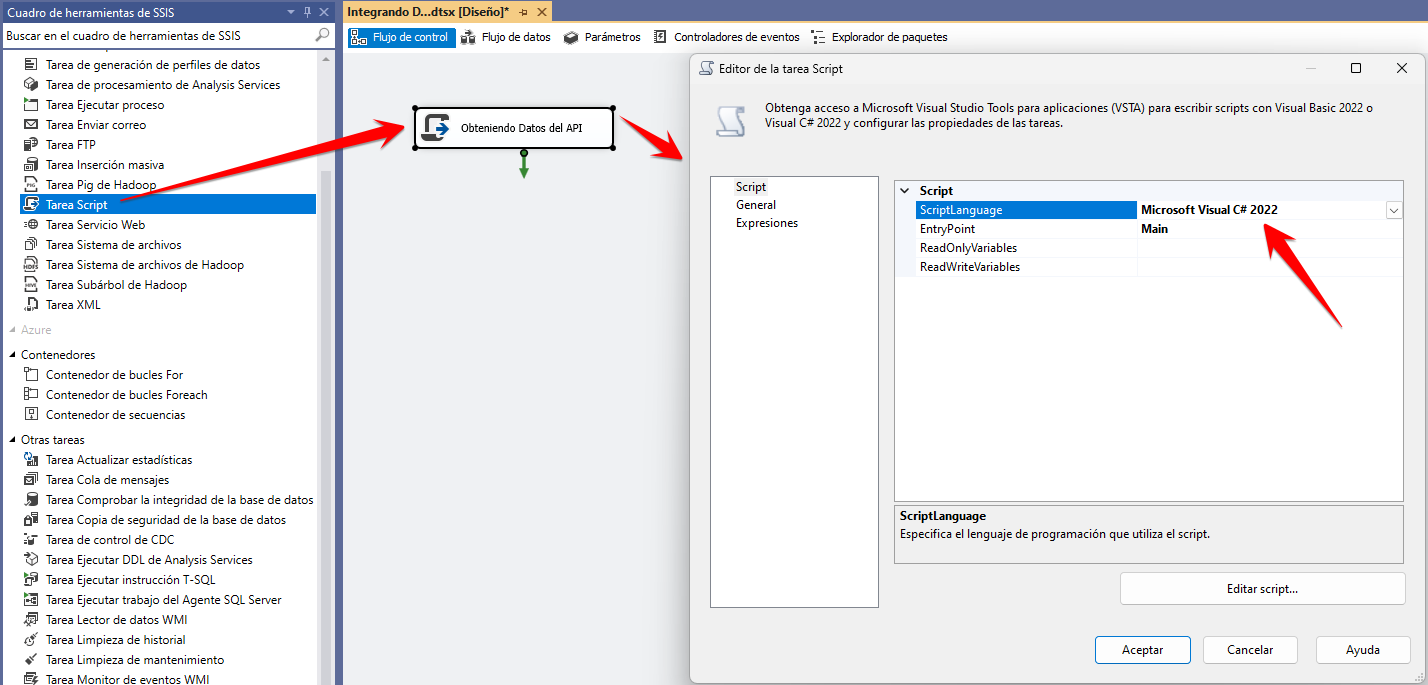
* **APIKey** ( String ): Will store the OpenWeatherMap API Key that I obtained when creating the account.
* **City1** ( String ): Will store the name of a specific city (for this project, "Ferreñafe").
* **City2** ( String ): Will store the name of a specific city (for this project, "Chiclayo").
* **City3** ( String ): Will store the name of a specific city (for this project, "Lambayeque").
* **Country1** ( String ): Will store the country to which city 1 belongs.
* **Country2** ( String ): Will store the country to which city 2 belongs.
* **Country3** ( String ): Will store the country to which city 3 belongs.
* **Humidity1** ( Int ): Will store the humidity obtained from city 1.
* **Humidity2** ( Int ): Will store the humidity obtained from city 2.
* **Humidity3** ( Int ): Will store the humidity obtained from city 3.
* **Pressure1** ( Int ): For atmospheric pressure obtained from city 1.
* **Pressure2** ( Int ): For atmospheric pressure obtained from city 2.
* **Pressure3** ( Int ): For atmospheric pressure obtained from city 3.
* **Temperature1** (Decimal): Will store the temperature obtained from city 1.
* **Temperature2** (Decimal): Will store the temperature obtained from city 2.
* **Temperature3** (Decimal): Will store the temperature obtained from city 3.
* **WeatherDescription1** ( String ): For the weather description obtained from city 1.
* **WeatherDescription2** ( String ): For the weather description obtained from city 2.
* **WeatherDescription3** ( String ): For the weather description obtained from city 3.
* **WindSpeed1** (Decimal): For the wind speed obtained from city 1.
* **WindSpeed2** (Decimal): For the wind speed obtained from city 2.
* **WindSpeed3** (Decimal): For the wind speed obtained from city 3.

**Only** **APIKey , City1, City2 and City3 have a “Value” assigned** , the other variables will be left empty since they will be filled in the Script. If it is necessary to change the cities, they can be changed by modifying this section.



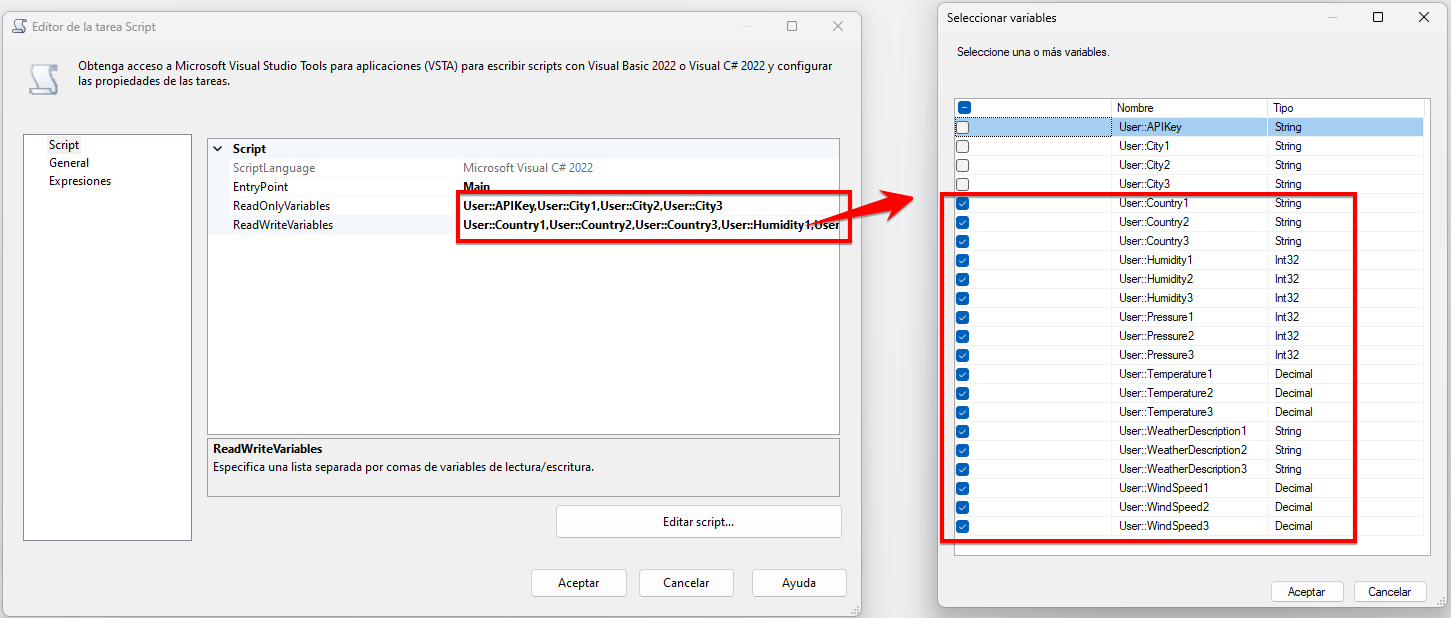
**Step 4: Script Task Component.**

I will use the “Script Task” component and use C# as the programming language to configure the data retrieval from the API.



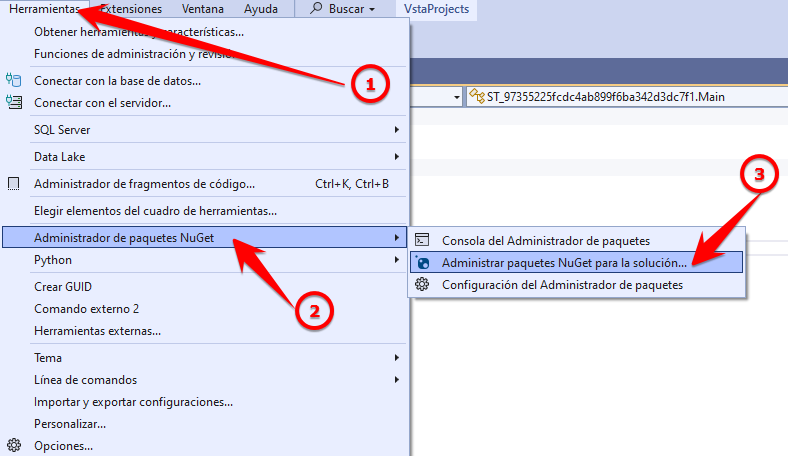
Variables are added to the Script:

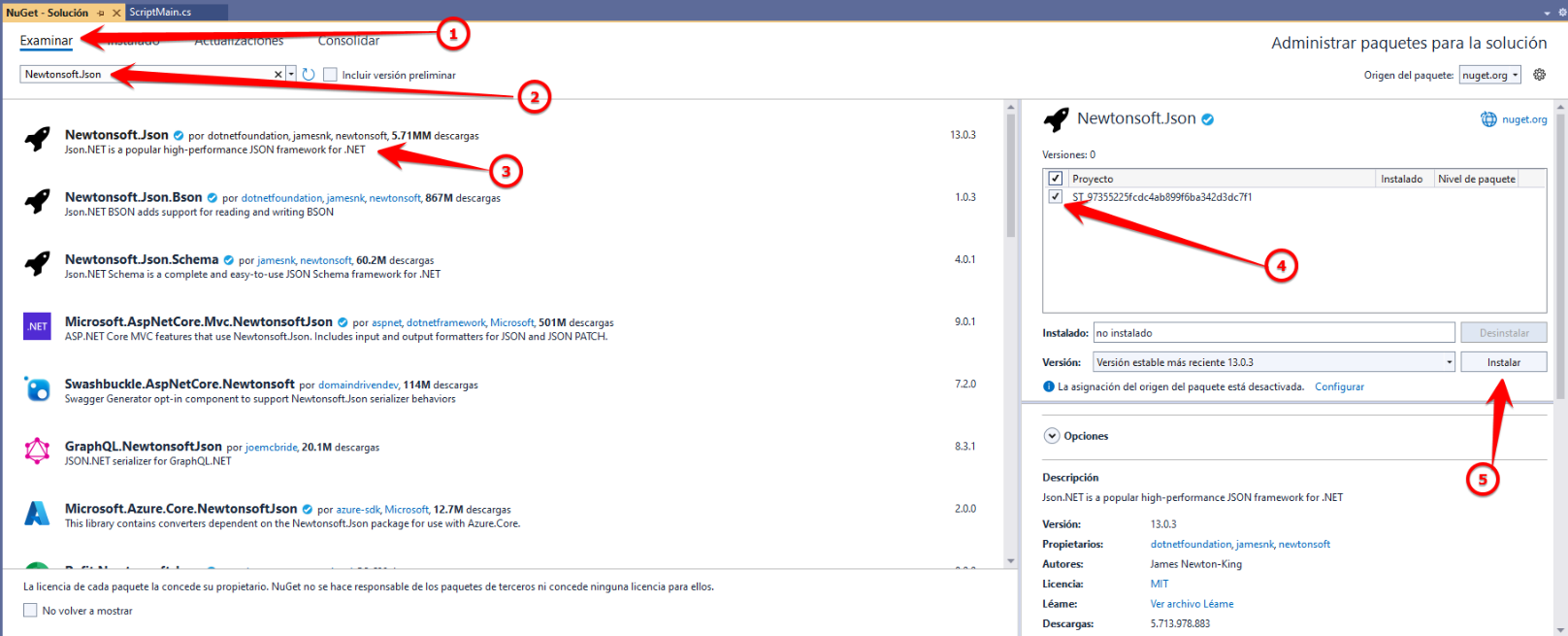
* **ReadOnlyVariables "** tab, I selected the variables User::APIKey , User::City1, User::City2, User::City3. (The variables I set a value for.)
* **ReadWriteVariables "** tab, I selected the other variables (the other variables that were not given any Value).

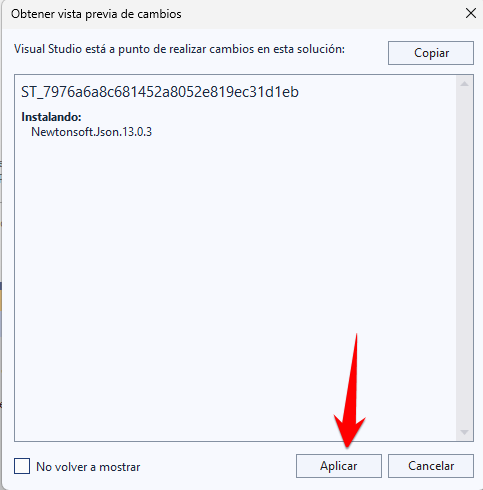


**Step 5: Editing the Script with C# code.**

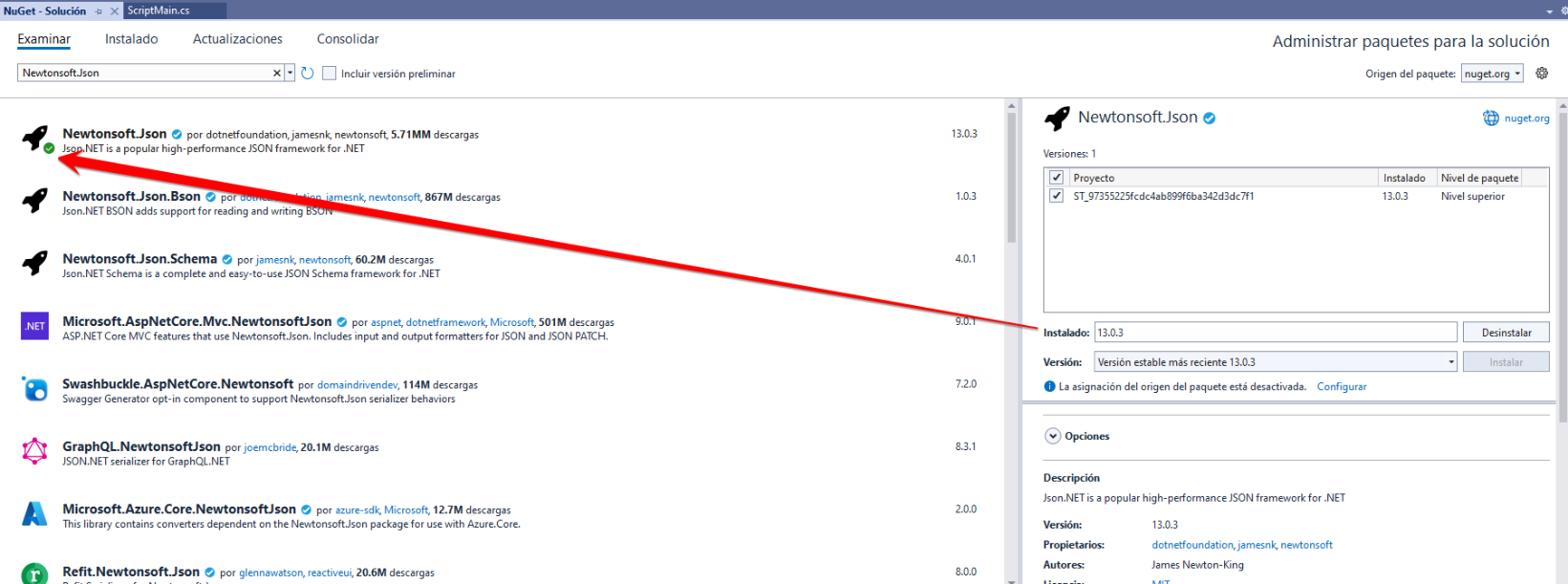
First I will install a library that I will use in the code, the library is **Newtonsoft.Json :**



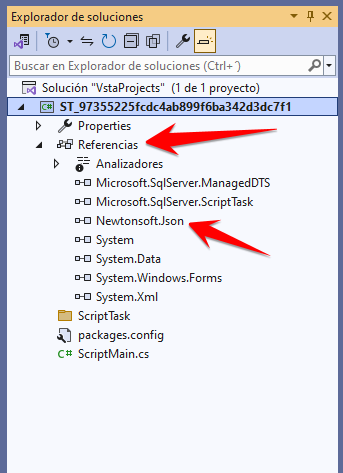




Successfully installed:

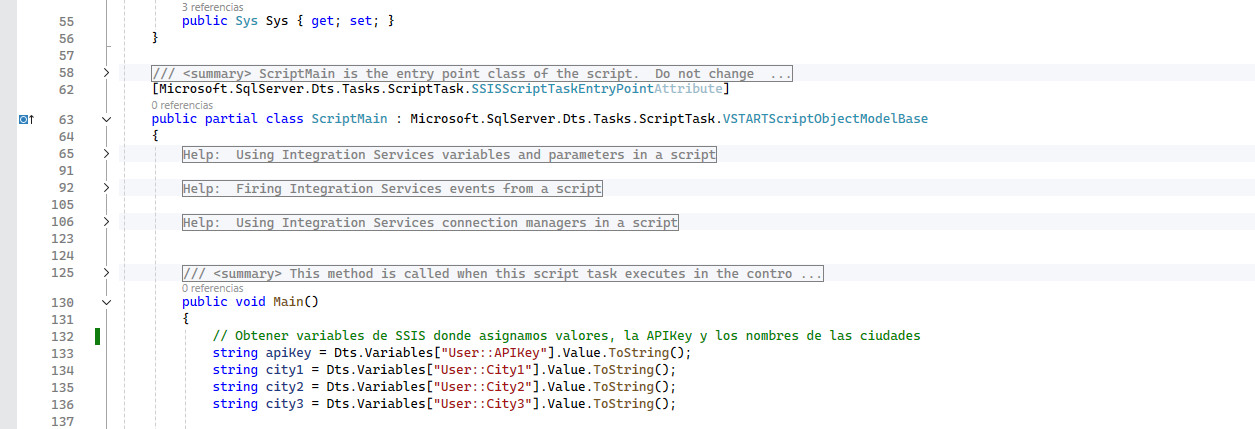


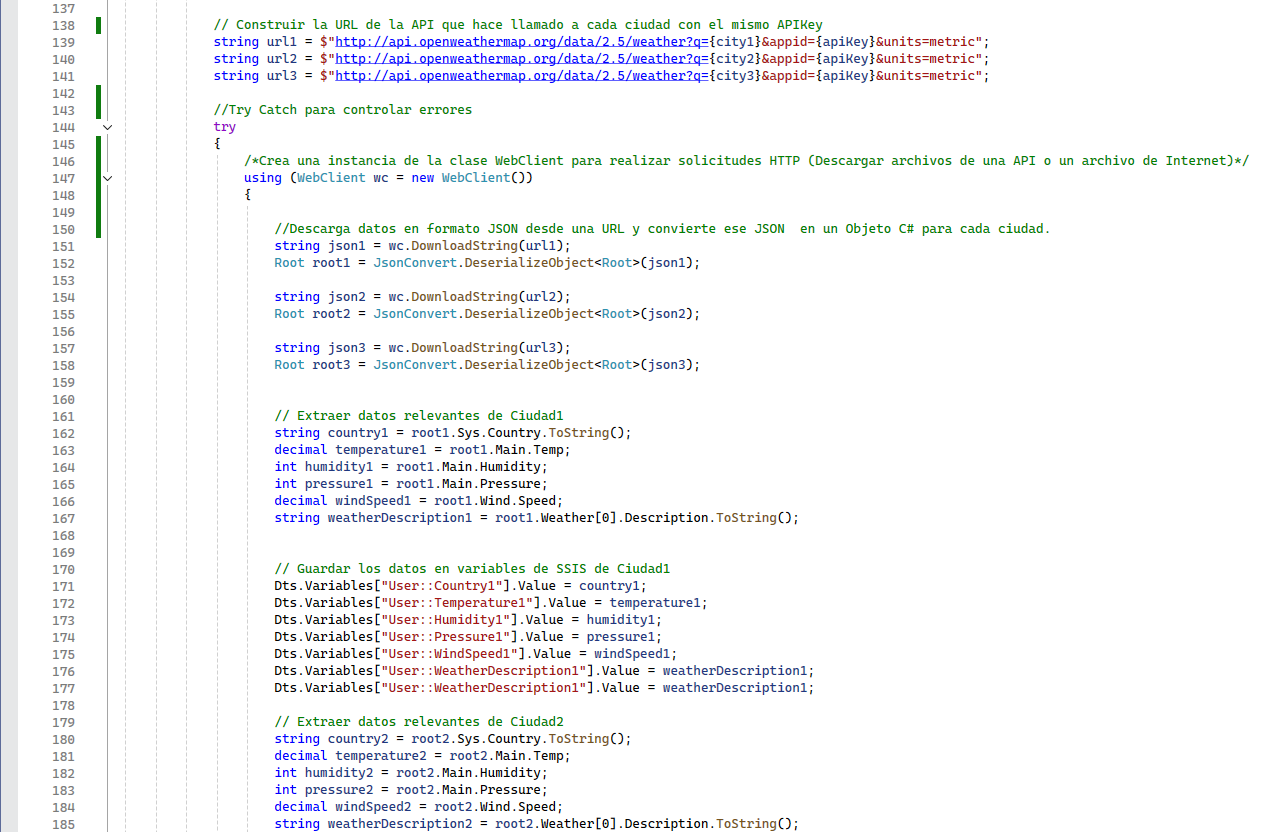
We verify that the library that was added appears referenced:

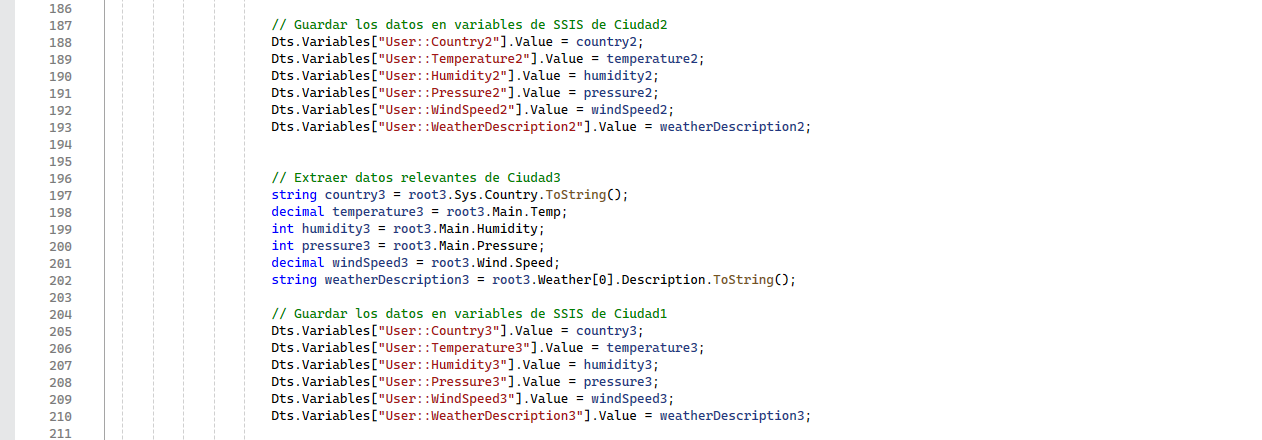


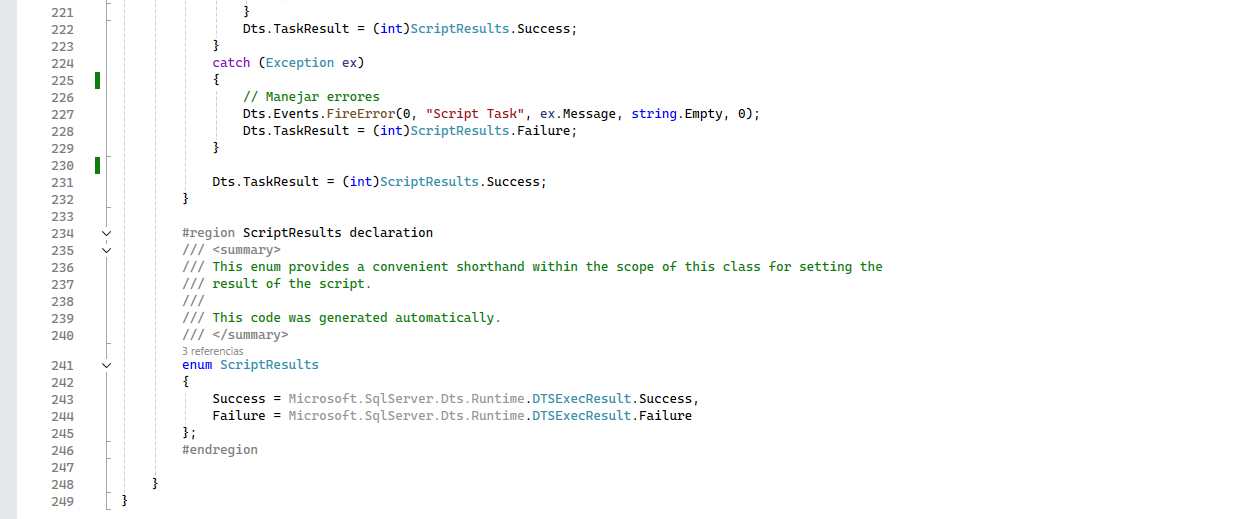
Then I wrote the code that calls the OpenWeatherMap API using the API Key and 3 Cities, we create the classes taking into account the JSON, then it extracts the temperature, humidity, atmospheric pressure, wind speed and weather description of each city provided from the response JSON and saves the values in the variables created in the SSIS.





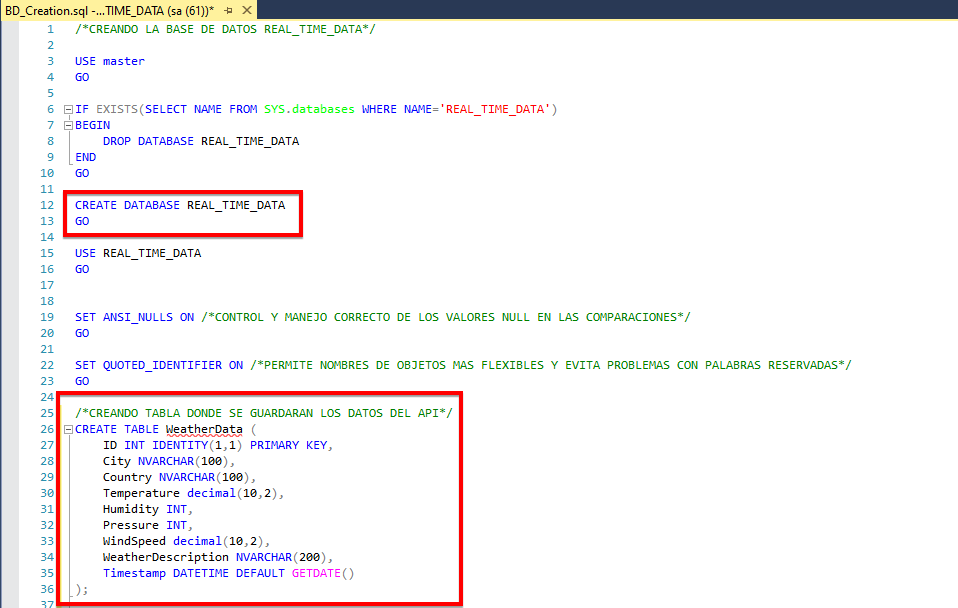






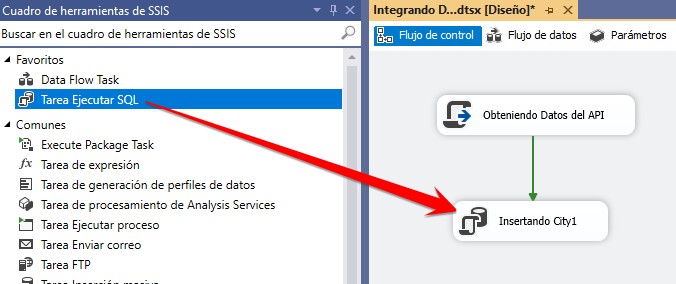
**Step 6: Creating Table to store the Data.**

I will create a database and a table in SQL Server where the data extracted from the API will be stored:

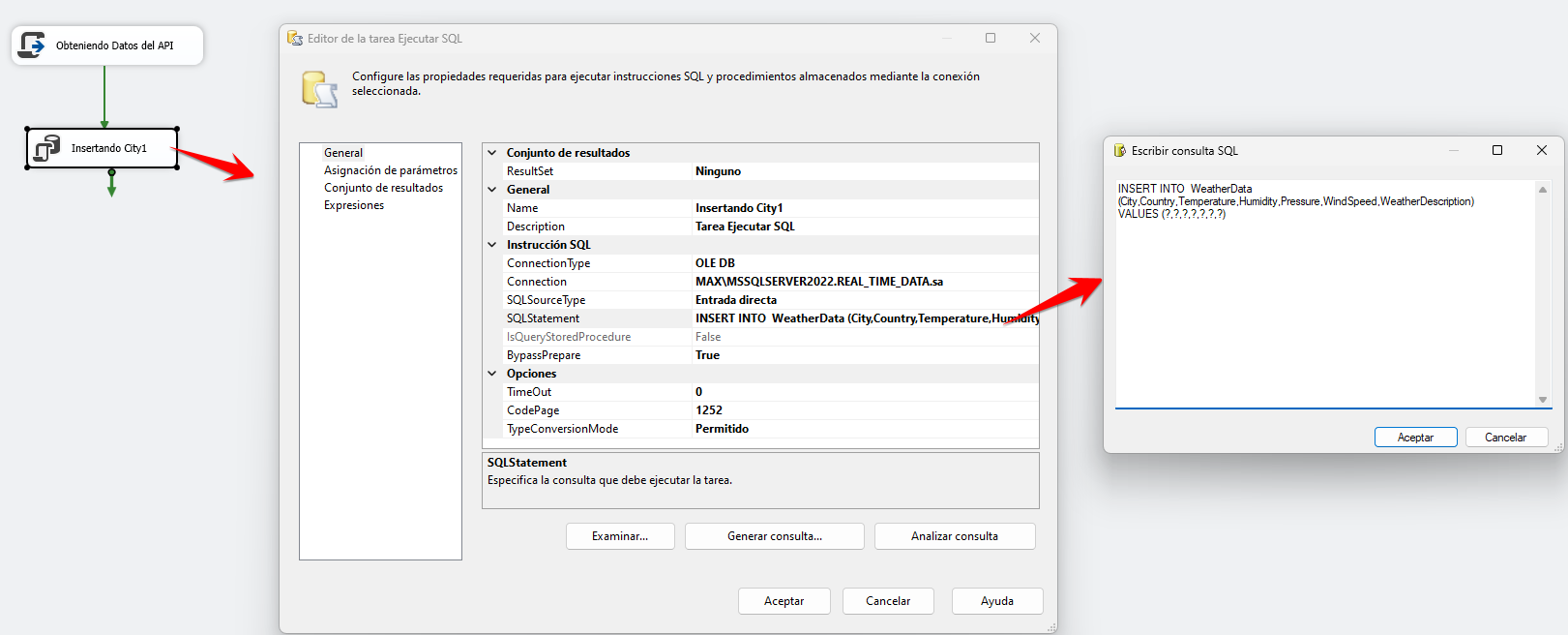


**Step 7: Creating an Execute SQL Task to insert the data into the Database.**

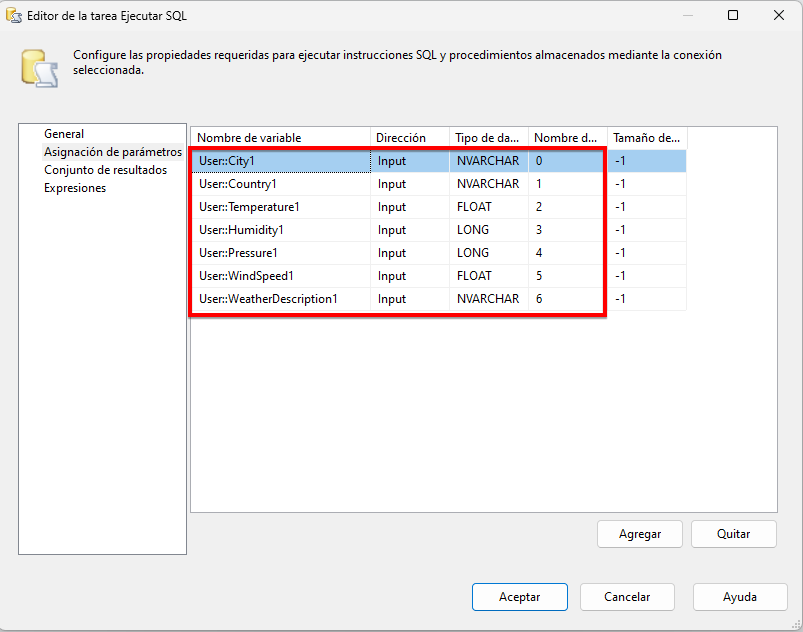
Now for the process of loading JSON data to a table in SQL Server I used the Execute SQL Task component to insert the data corresponding to each City.



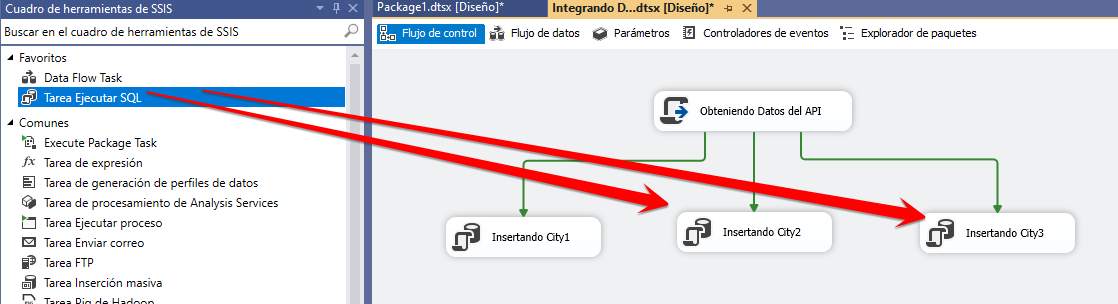
I added the connection to the Database and the SQL command where I will pass the variables.

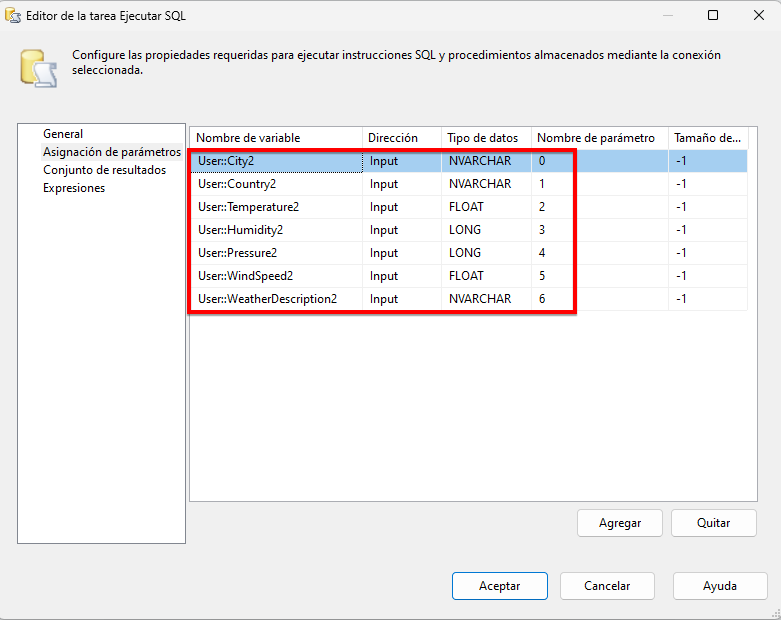
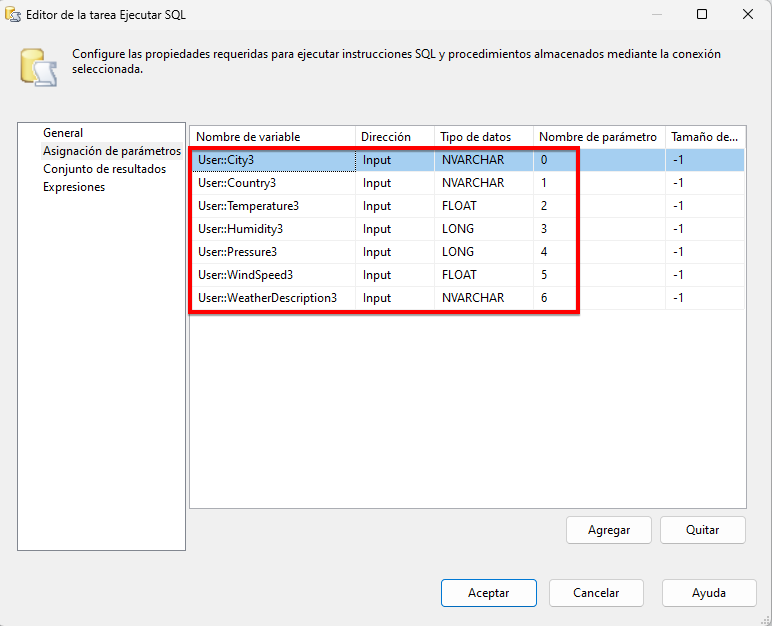


In Variable Assignment we place everything related to data obtained for city 1.



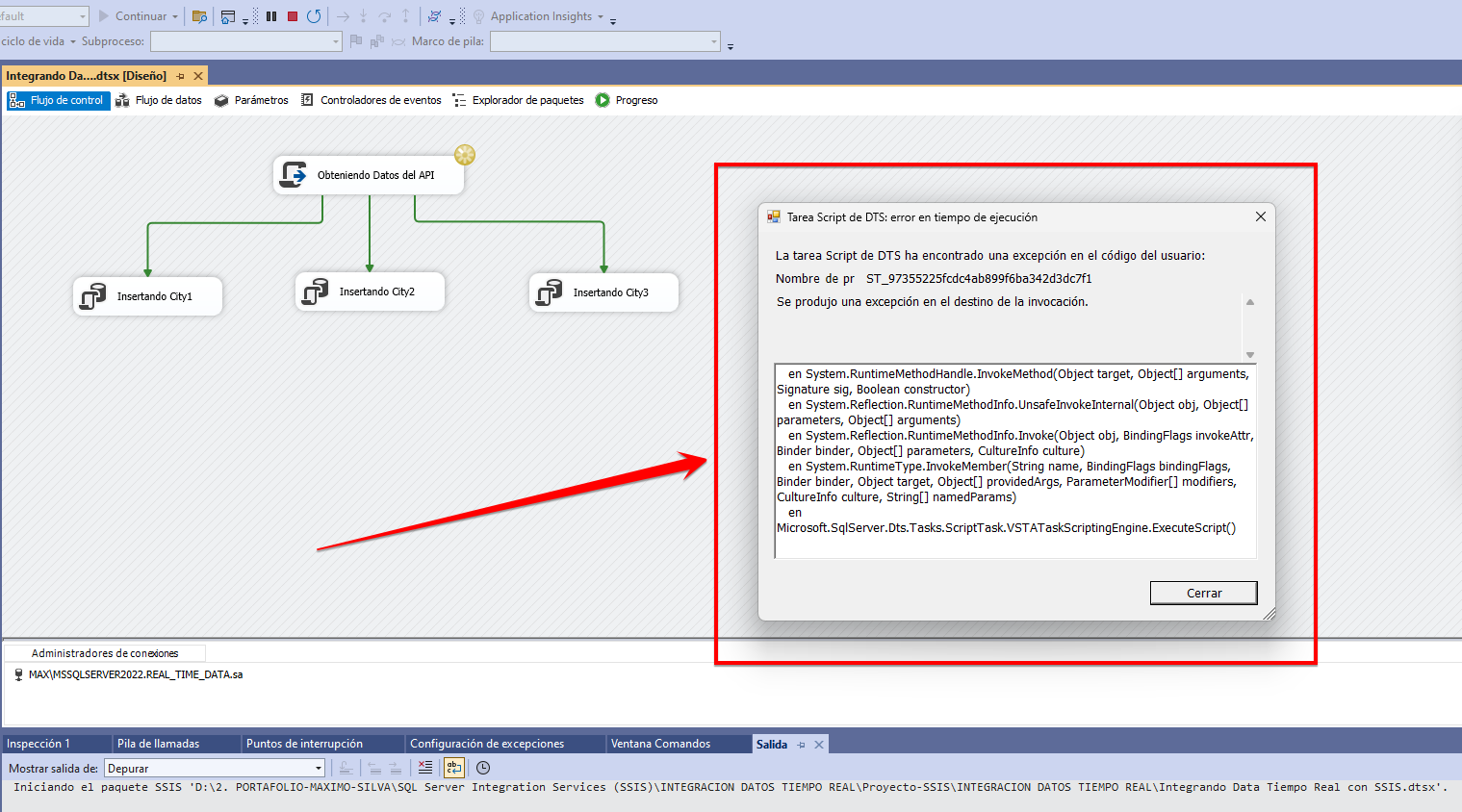
The same process was done for the other cities:

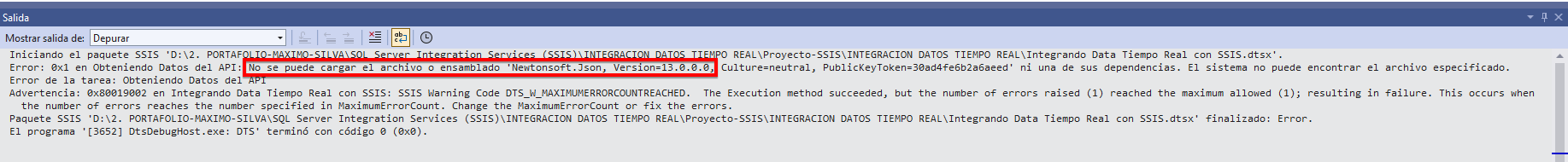


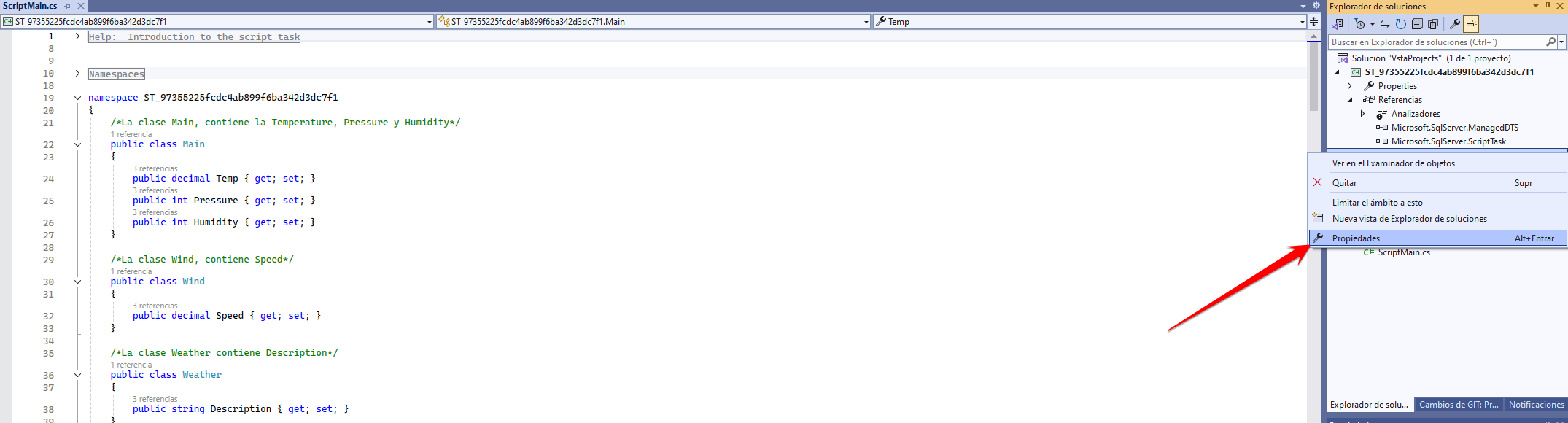
**Step 8: Execution Error and Solution to the Error.**

When starting the project execution **I got the following error** :

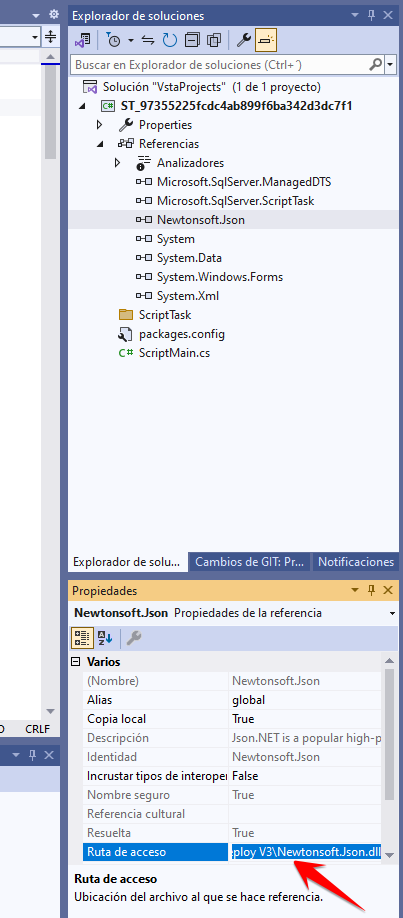




I was investigating and this error indicates that the reference to the **Newtonsoft.Json library** is not registered in the **GAC (Global Assembly Cache)** so it needs to be registered so that it can be located by the project and thus be used, for this we open again where the C# code was written to search for the path where the downloaded library is:



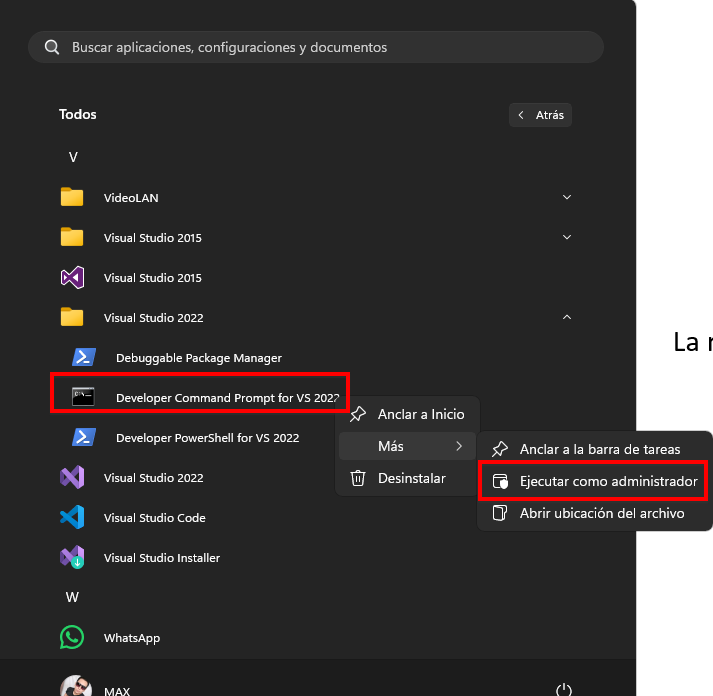
Identify the Library Path and copy it:



The route where it was downloaded was:

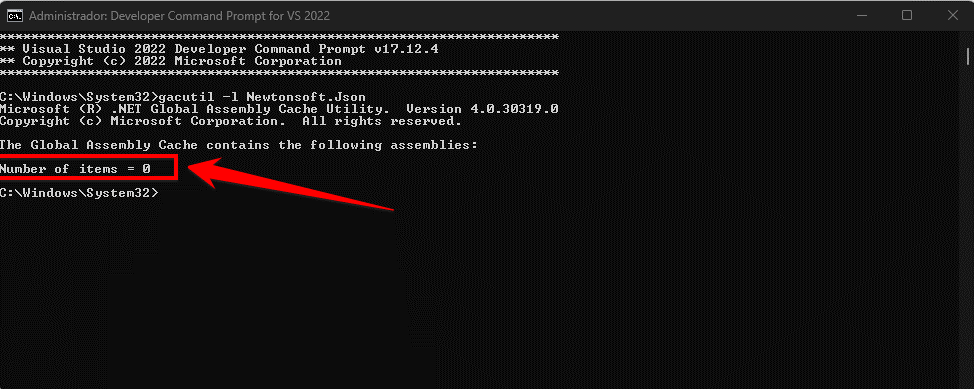
**C:\Program Files\IIS\Microsoft Web Deploy V3\Newtonsoft.Json.dll**

Then I opened **Developer Command Prompt for VS 2022** and ran it as Administrator.



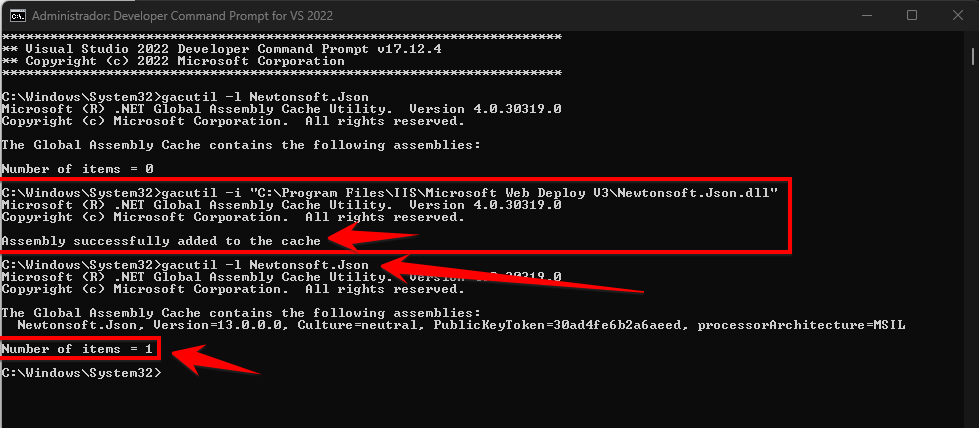
In the command screen we verify that the Library is not registered:

**gacutil -l Newtonsoft.Json**



The number 0 indicates that there are no libraries registered in the **GAC**, to reference it **globally** (Not just for the project, that's why **gacutil** is used):

**gacutil -i "C:\Program Files\IIS\Microsoft Web Deploy V3\Newtonsoft.Json.dll"**



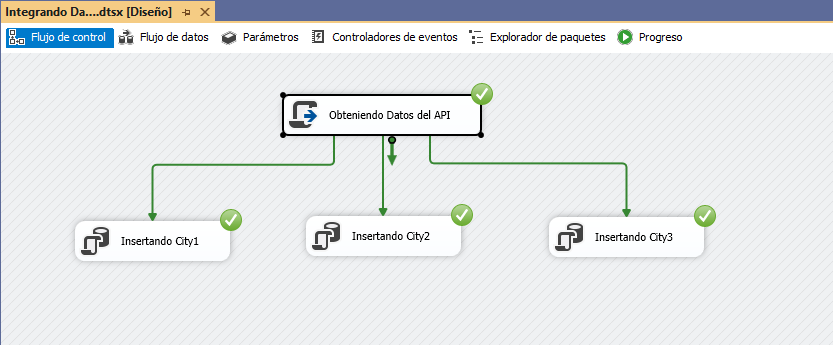
When checking the library again, the number 1 appears, indicating that a library is registered, in this case Newtonsoft.Json . **The problem should now be solved.**

If for some reason we need to delete reference to the library, it is done with the following command:

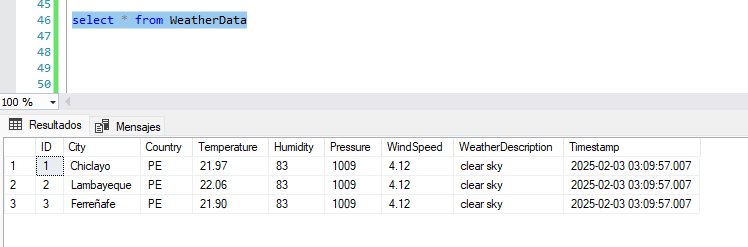
**gacutil -u Newtonsoft.Json**

**Step 9: Running the project.**

Once the error is resolved, when running the project, it does so without any problem:

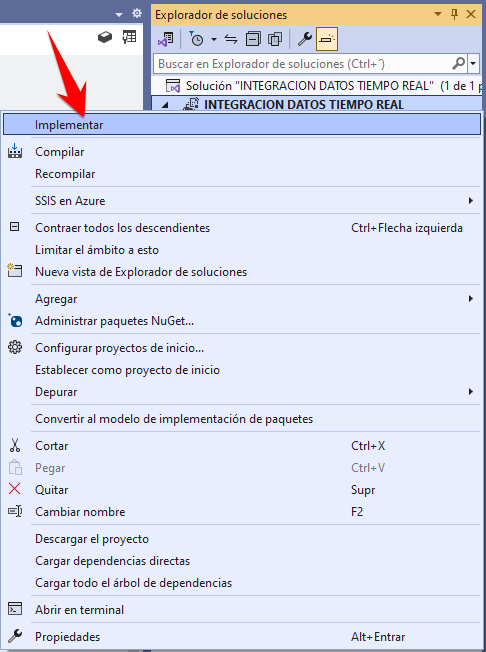
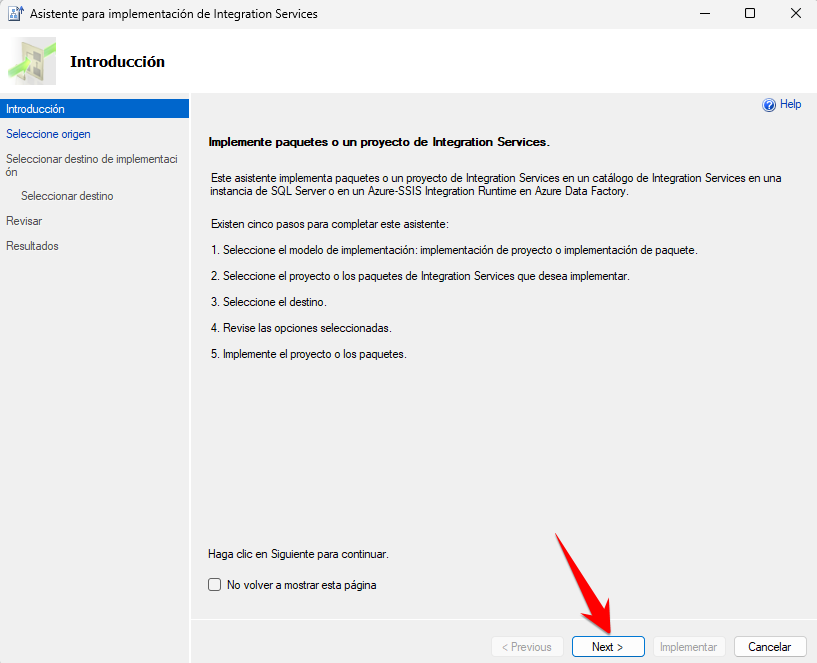


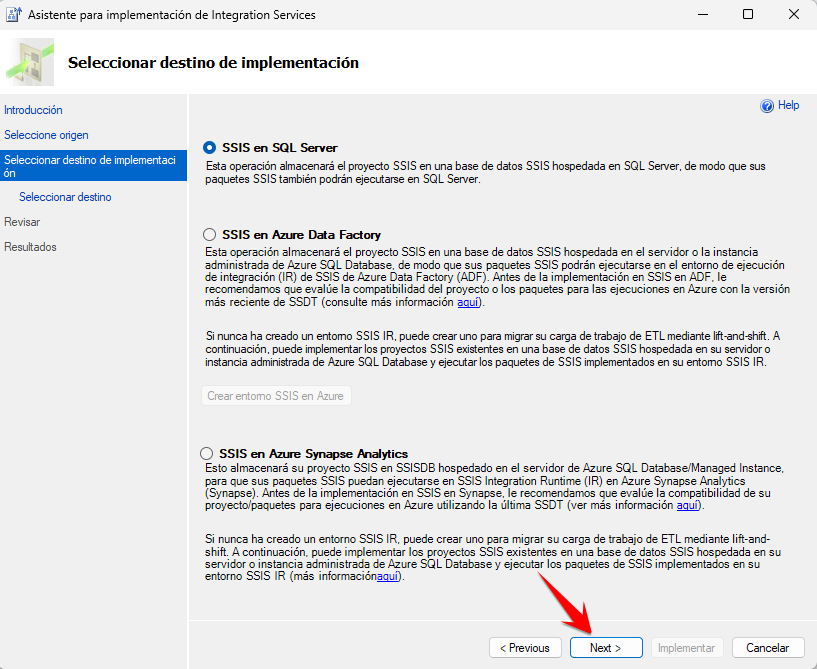
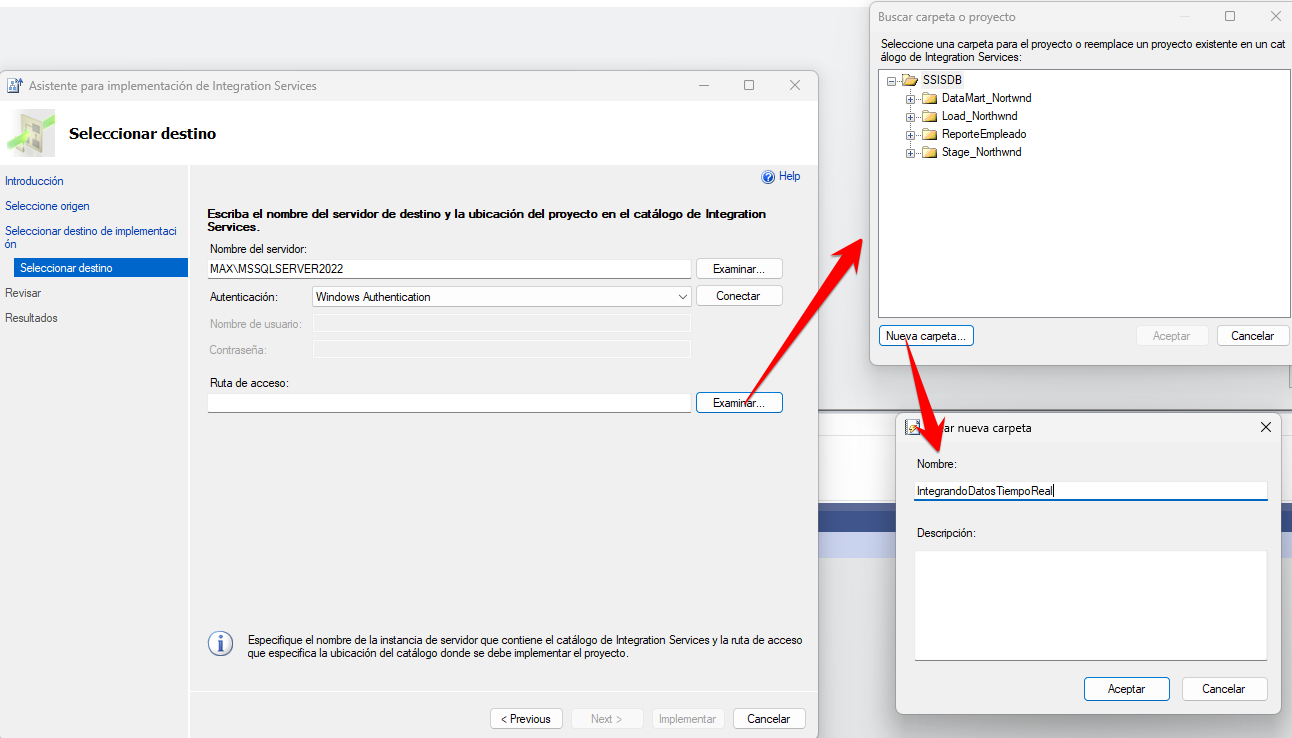
I verify that the data was recorded in the Database:

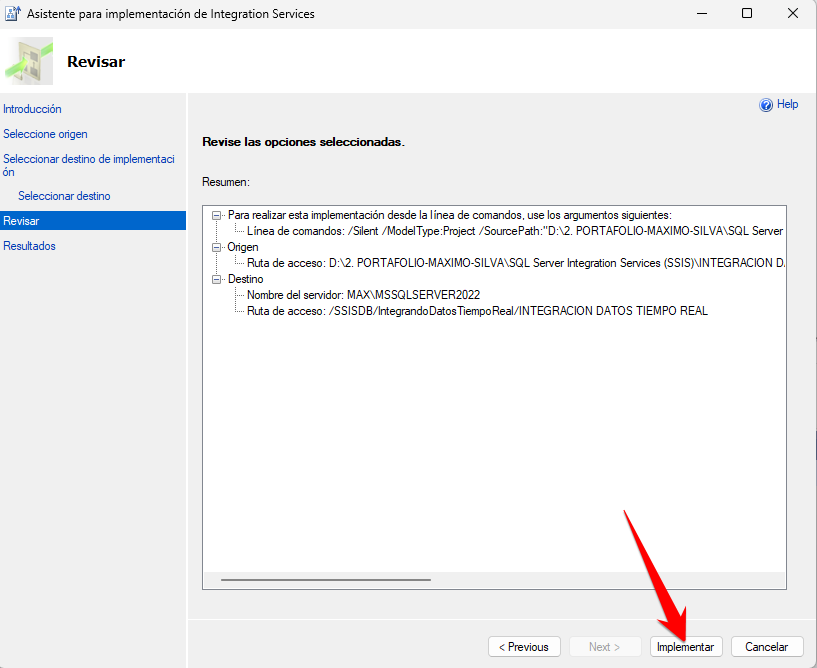
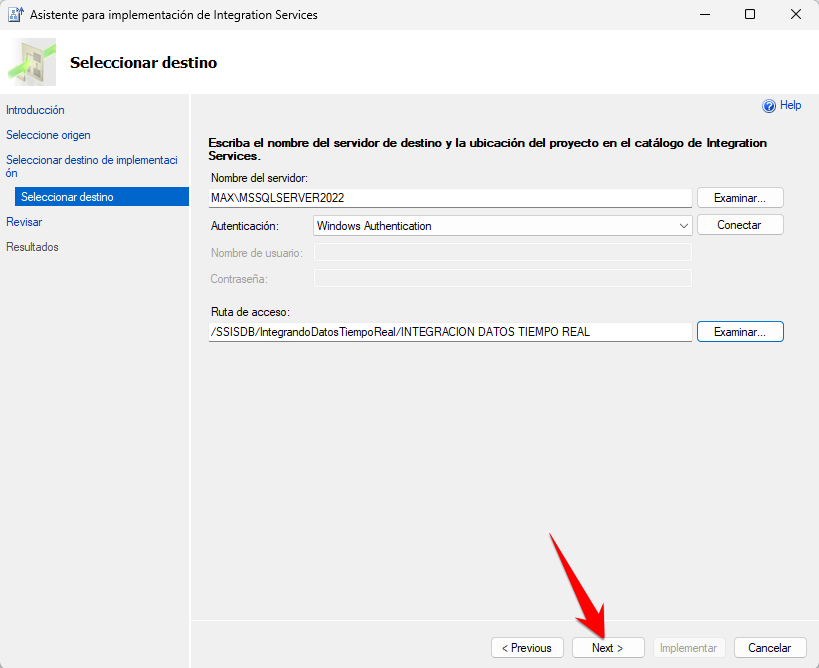


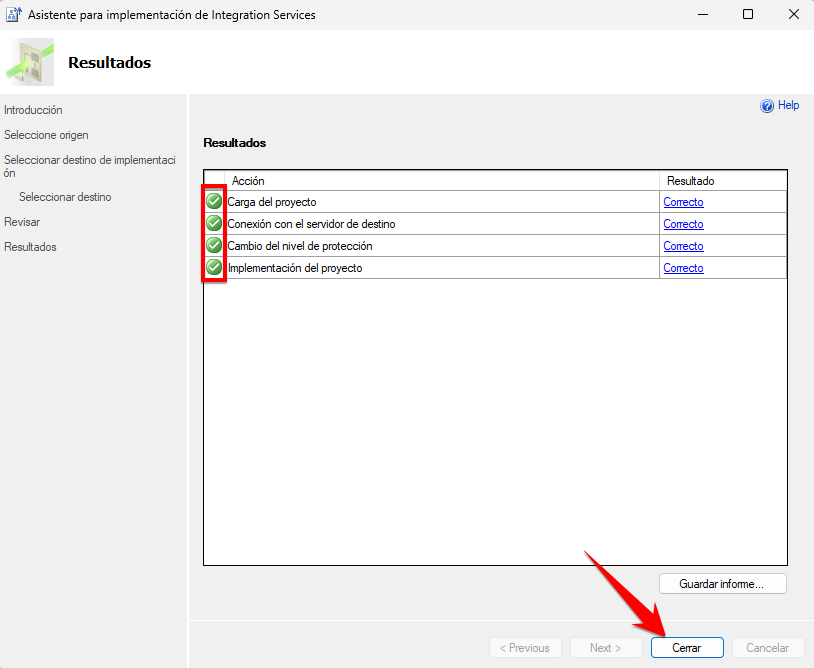
**Step 10: Implementing the Integration project Services .**

Once we have the project, I will start with the deployment so that it appears in SQL Server and I can create a Scheduled Task in the SQL Server Agent, for this I implemented it:

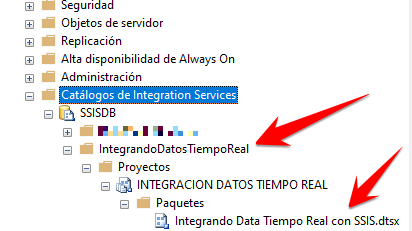
 



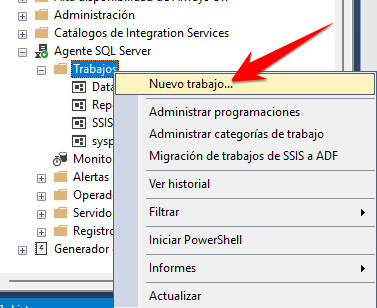
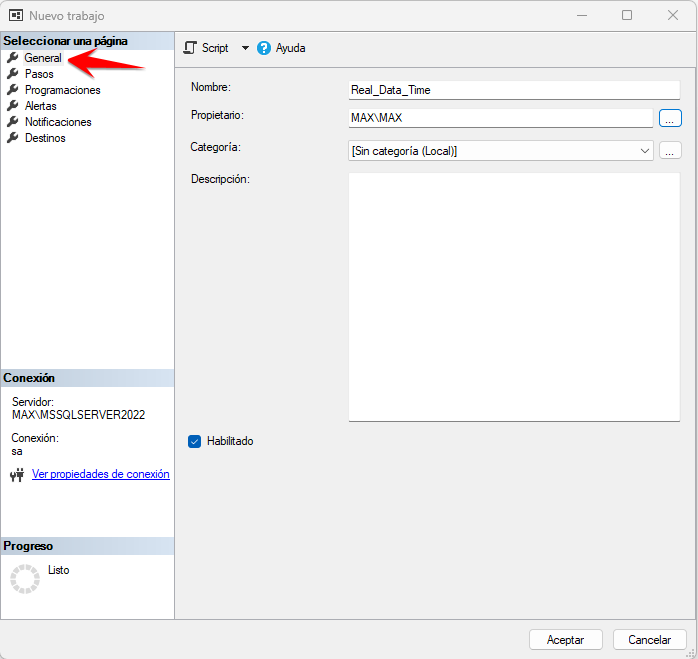


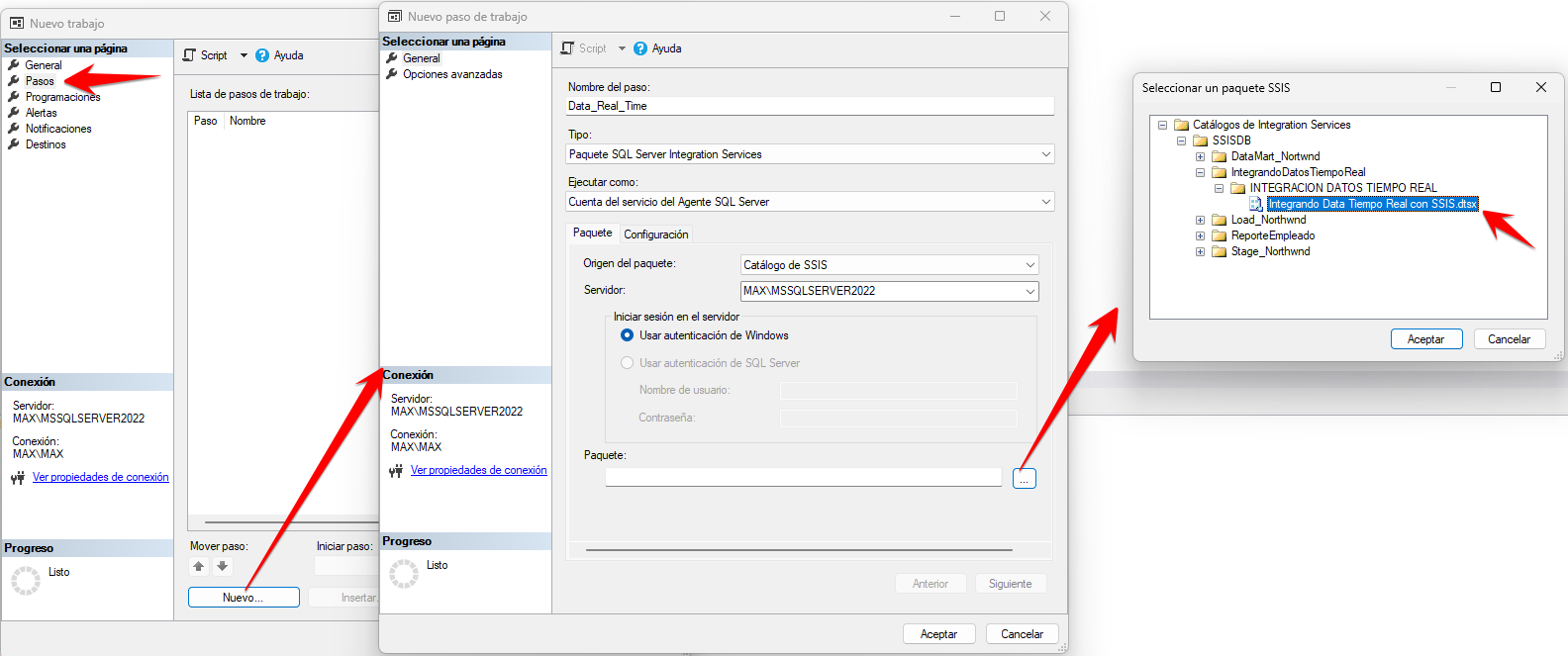
In SQL Server, in the Integration Services Catalogs section (a new catalog had to be created previously), the project now appears:



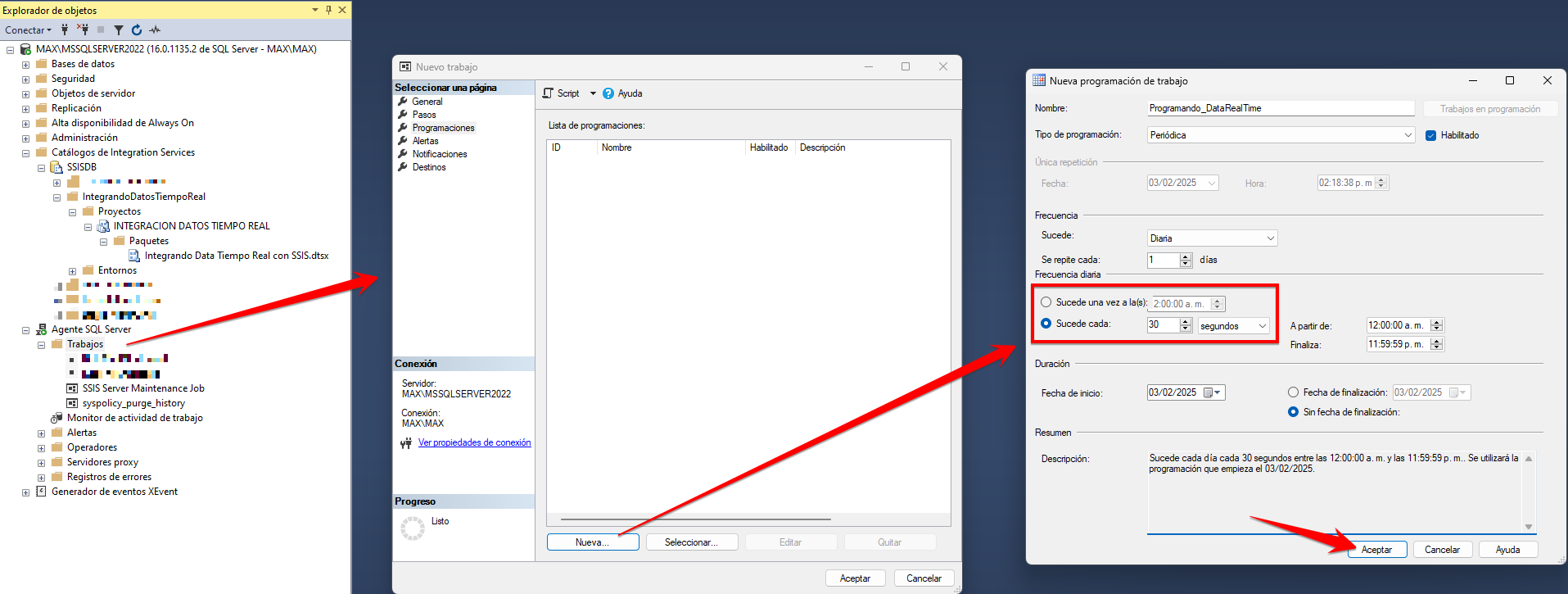
**Step 11: Creating an automatic scheduled task with SQL Server Agent.**

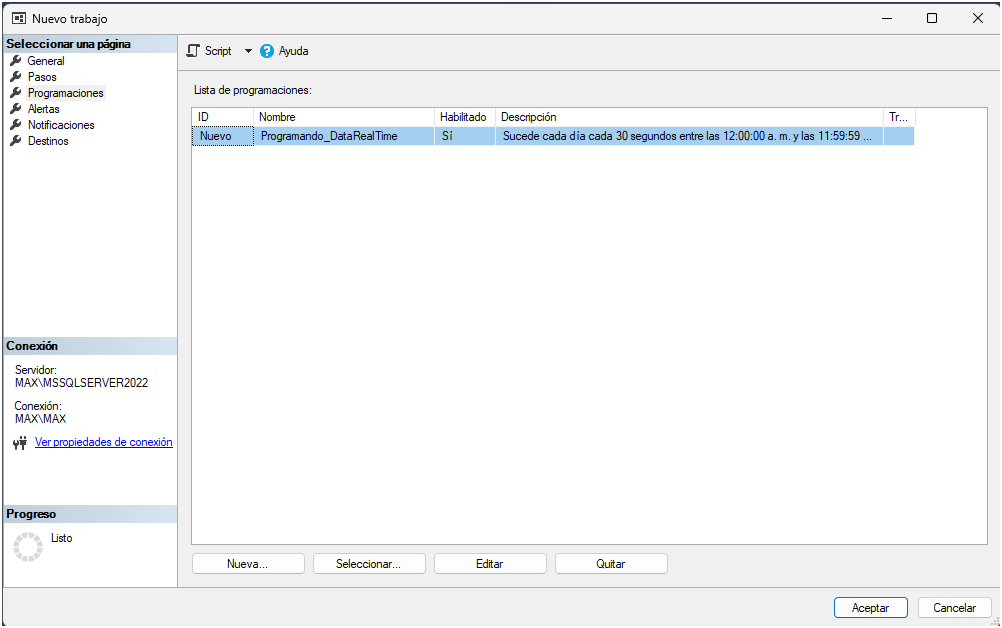
Create a new Job in SQL Server that runs the previously deployed project:



Scheduling to run automatically every 30 seconds:

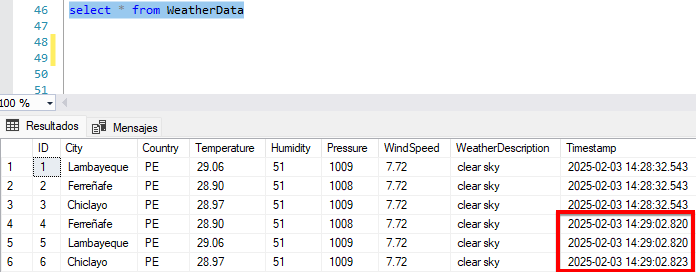




It will be executed automatically, the time of the first execution is observed:



On the second run it shows that it happened 30 seconds later just as I programmed it, but you can reduce that time if you want:



In this way I conclude the project where information was extracted from an API and inserted into a database in SQL Server using SSIS and which will be executed every 30 seconds automatically.