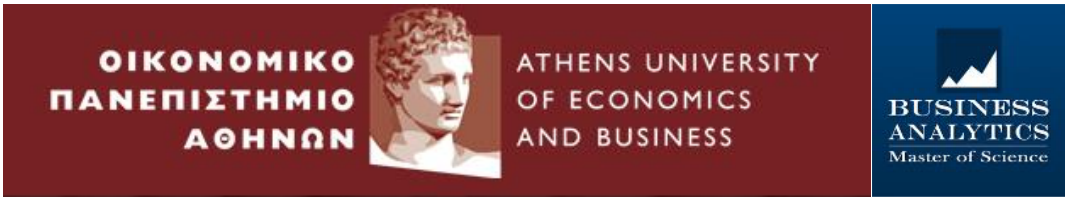


MSc Business Analytics
Data Management & Business Intelligence

COVID-19 Vaccinations in the United States

Lakkas-Pyknis Evangelos
Registration number: f2822306
E-mail: eva.lakkaspyknis@aueb.gr

MESOLORA STAMATOULA-GERASIMOULA
Registration number: f2822308
E-mail: sta.mesolora@aueb.gr

Contents

Introduction

In the wake of a global health crisis, the pursuit of understanding and enhancing public health measures has become paramount. As the world grappled with the unprecedented challenges posed by the COVID-19 pandemic, the pursuit of vaccination emerged as a cornerstone in the fight against the virus. In this pursuit, leveraging comprehensive and reliable data has proven instrumental in strategizing and implementing effective vaccination campaigns. Our project endeavors to delve into this crucial realm, using data-driven insights to examine COVID-19 vaccination rates across different age groups and counties in the United States.

The dataset under scrutiny originates from the Centers for Disease Control and Prevention (CDC), a stalwart institution renowned for its dedication to safeguarding public health through rigorous science and data-driven decision-making. This dataset, encapsulating information from the years 2021 and 2022, provides a nuanced perspective on vaccination trends across the U.S., offering a granular view that spans counties and age demographics. The original dataset in its complete form can be found here: https://data.cdc.gov/Vaccinations/COVID-19-Vaccinations-in-the-United-States-County/8xkx-amqh/data

Our venture into this dataset seeks to address fundamental questions surrounding the distribution, efficacy, and uptake of COVID-19 vaccinations. The primary goal is to discern patterns and variations in vaccination rates, dissecting these insights through the lenses of age groups and geographical distinctions at the county level. By examining these parameters, we aim to derive actionable insights that can inform targeted strategies, interventions, and policy formulations aimed at bolstering vaccination coverage.

The methodology adopted for this project involved a meticulous approach encompassing data collection, cleaning, and analysis across multiple platforms and tools. Leveraging the diverse functionalities of R, Python, and SQL, the initial phase involved comprehensive data cleaning to ensure the dataset's integrity and reliability. Cleaning operations focused on rectifying inconsistencies, handling missing values, and standardizing the data structure to prepare it for downstream processing.

The subsequent phase witnessed the utilization of a suite of tools within the Microsoft ecosystem, including SQL Server Management Studio (SSMS), SQL Server Integration Services (SSIS), and SQL Server Analysis Services (SSAS). SSMS and SSIS were pivotal in constructing and refining our data warehouse, orchestrating the seamless flow of information from raw datasets to well-organized fact and dimension tables. These tables, designed following the principles of snowflake schema, formed the bedrock upon which our analytical endeavors were built.

The culmination of our efforts materialized in the creation of a robust multi-dimensional model—a cube—within SQL Server Analysis Services (SSAS). This cube encapsulates a rich tapestry of dimensions, measures, and calculated metrics derived from the meticulously curated dataset. Harnessing the capabilities of OLAP (Online Analytical Processing), our

exploration within the cube enables sophisticated analyses, facilitating drill-downs, roll-ups actions to extract meaningful insights.
Moreover, to amplify the impact of our findings, we employed Power BI as a potent visualization tool. Power BI transformed our analytical outputs into visually engaging and informative representations. Through interactive charts and visualizations, we endeavor to present our discoveries in a compelling and easily comprehensible manner.

In the subsequent sections of this report, we elaborate on the methodologies, challenges encountered, insights gleaned, and the visual narratives sculpted from the comprehensive analysis of COVID-19 vaccination rates across different demographic strata and geographic regions in the United States.

Data Acquisition and Preparation Methodologies

In the intricate landscape of data analysis, the foundation laid in data acquisition, cleaning, and warehouse design stands as the bedrock upon which insightful analyses are built. This section delves into the technical intricacies and systematic approaches employed in our quest to harness the Centers for Disease Control and Prevention (CDC) dataset regarding COVID-19 vaccination rates. We elucidate the methodological framework comprising data collection, cleaning procedures executed through a synergy of R, Python, and SQL. Our narrative unravels the meticulous steps taken to transform raw data into a structured and reliable format, setting the stage for subsequent analytical endeavors within this enriched data ecosystem.

Data cleaning in R

Package Loading and Data Import:

The script starts by loading the necessary packages: lubridate and dplyr and importing the dataset into the data frame named “covidDF”.

Date Standardization:

Using “mdy()” function from the lubridate package, the “Date” column in covidDF_2021plus is converted into a standardized date format (month-day-year).

Data Cleaning and Transformation:

The next step is the conversion of columns to appropriate data types as follows. The columns related to doses administered (Administered_Dose1_Recip, etc.) are cleaned and converted to the 'raw' data type using “as.raw()” function. Then, the “gsub()” function removes commas (,) from specific columns (Administered_Dose1_Recip, etc.), preparing them for numeric conversion. After removing commas, “as.numeric()” function converts these columns to numeric types, allowing mathematical operations.

Creation of derived columns:

New columns (Administered_Dose1_0to12_group, Administered_Dose1_12to18_group, etc.), are generated by performing arithmetic operations on existing columns. These columns represent grouped age-based doses administered based on existing data.

Validation and Checking:

In the end, unique dates, states, and FIPS are calculated from the cleaned dataset. The function “unique()” extracts the unique values for dates, states, and FIPS codes, validating the data against predefined unique values and “length()” function checks the number of unique elements in specific columns (FIPS, Date, etc.), ensuring completeness and consistency of the data.

R code:

```
install.packages("lubridate")

library(lubridate)
library(dplyr)

covidDF <- read.csv(file = "COVID-
19_Vaccinations_in_the_United_States_County_CDC_dataset.csv", header = TRUE)
covidDF_new<-subset.data.frame(covidDF, select =
c('Date','FIPS','Recip_County','Recip_State','Completeness_pct','Administered_Dose1_Recip',
'Administered_Dose1_Pop_Pct','Administered_Dose1_Recip_12Plus',
'Administered_Dose1_Recip_12PlusPop_Pct','Administered_Dose1_Recip_18Plus','Administ
ered_Dose1_Recip_18PlusPop_Pct',
'Administered_Dose1_Recip_65Plus','Administered_Dose1_Recip_65PlusPop_Pct',
'Series_Complete_Yes','Series_Complete_Pop_Pct',
      'Series_Complete_12Plus','Series_Complete_12PlusPop_Pct','Series_Complete_18Plus'
,'Series_Complete_18PlusPop_Pct','Series_Complete_65Plus','Series_Complete_65PlusPop_
Pct','Booster_Doses','Booster_Doses_Vax_Pct','Booster_Doses_12Plus','Booster_Doses_12Pl
us_Vax_Pct',
'Booster_Doses_18Plus','Booster_Doses_18Plus_Vax_Pct','Booster_Doses_65Plus','Booster_
Doses_65Plus_Vax_Pct',
      'SVI_CTGY','Series_Complete_Pop_Pct_SVI','Series_Complete_12PlusPop_Pct_SVI','Seri
es_Complete_18PlusPop_Pct_SVI',
'Series_Complete_65PlusPop_Pct_SVI','Metro_status','Census2019','Census2019_12PlusPop'
,'Census2019_18PlusPop','Census2019_65PlusPop'))
covidDF_2021plus$Date<-mdy(covidDF_2021plus$Date)
covidDF_2021plus <- covidDF_new[-(1:62377),]
covidDF_2021plus$Date<- mdy(covidDF_2021plus$Date)
covidDF_clean <- subset(covidDF_2021plus, year(covidDF_2021plus$Date)!= 2020)

covidDF_2021plus$Administered_Dose1_Recip <-
as.raw(covidDF_2021plus$Administered_Dose1_Recip)
covidDF_2021plus$Administered_Dose1_Recip_no_com =
gsub(",","",covidDF_2021plus$Administered_Dose1_Recip)
covidDF_2021plus$Administered_Dose1_Recip <-
no_space(covidDF_2021plus$Administered_Dose1_Recip)
covidDF_2021plus$Administered_Dose1_Recip_no_com <-
as.numeric(covidDF_2021plus$Administered_Dose1_Recip_no_com)
```

```
covidDF_clean$Administered_Dose1_Recip <-
gsub(",","",covidDF_clean$Administered_Dose1_Recip)
covidDF_clean$Administered_Dose1_Recip <-
as.numeric(covidDF_clean$Administered_Dose1_Recip)
covidDF_clean$Administered_Dose1_Recip_12Plus <-
gsub(",","",covidDF_clean$Administered_Dose1_Recip_12Plus)
covidDF_clean$Administered_Dose1_Recip_12Plus <-
as.numeric(covidDF_clean$Administered_Dose1_Recip_12Plus)
covidDF_clean$Administered_Dose1_Recip_18Plus <-
gsub(",","",covidDF_clean$Administered_Dose1_Recip_18Plus)
covidDF_clean$Administered_Dose1_Recip_18Plus<-
as.numeric(covidDF_clean$Administered_Dose1_Recip_18Plus)
covidDF_clean$Administered_Dose1_Recip_65Plus<-
gsub(",","",covidDF_clean$Administered_Dose1_Recip_65Plus)
covidDF_clean$Administered_Dose1_Recip_65Plus<-
as.numeric(covidDF_clean$Administered_Dose1_Recip_65Plus)
covidDF_clean$Administered_Dose1_0to12_group <-
covidDF_clean$Administered_Dose1_Recip-
covidDF_clean$Administered_Dose1_Recip_12Plus
covidDF_clean$Administered_Dose1_12to18_group <-
covidDF_clean$Administered_Dose1_Recip_12Plus -
covidDF_clean$Administered_Dose1_Recip_18Plus
covidDF_clean$Administered_Dose1_18to65_group <-
covidDF_clean$Administered_Dose1_Recip_18Plus -
covidDF_clean$Administered_Dose1_Recip_65Plus

covidDF_clean$Series_Complete_Yes <- gsub(",","",covidDF_clean$Series_Complete_Yes)
covidDF_clean$Series_Complete_Yes <- as.numeric(covidDF_clean$Series_Complete_Yes)
covidDF_clean$Series_Complete_12Plus <-
gsub(",","",covidDF_clean$Series_Complete_12Plus)
covidDF_clean$Series_Complete_12Plus<-
as.numeric(covidDF_clean$Series_Complete_12Plus)
covidDF_clean$Series_Complete_18Plus <-
gsub(",","",covidDF_clean$Series_Complete_18Plus)
covidDF_clean$Series_Complete_18Plus <-
as.numeric(covidDF_clean$Series_Complete_18Plus)
covidDF_clean$Series_Complete_65Plus <-
gsub(",","",covidDF_clean$Series_Complete_65Plus)
covidDF_clean$Series_Complete_65Plus <-
as.numeric(covidDF_clean$Series_Complete_65Plus)
covidDF_clean$Series_Complete_12to18_group <- covidDF_clean$Series_Complete_12Plus
- covidDF_clean$Series_Complete_18Plus
covidDF_clean$Series_Complete_0to12_group <- covidDF_clean$Series_Complete_Yes -
covidDF_clean$Series_Complete_12Plus
covidDF_clean$Series_Complete_18to65_group <- covidDF_clean$Series_Complete_18Plus
- covidDF_clean$Series_Complete_65Plus

covidDF_clean$Booster_Doses <- gsub(",","",covidDF_clean$Booster_Doses)
covidDF_clean$Booster_Doses <- as.numeric(covidDF_clean$Booster_Doses)
covidDF_clean$Booster_Doses_12Plus <- gsub(",","",covidDF_clean$Booster_Doses_12Plus)
covidDF_clean$Booster_Doses_12Plus <- as.numeric(covidDF_clean$Booster_Doses_12Plus)
```

```
covidDF_clean$Booster_Doses_18Plus <- gsub(",","",covidDF_clean$Booster_Doses_18Plus)
covidDF_clean$Booster_Doses_18Plus <- as.numeric(covidDF_clean$Booster_Doses_18Plus)
covidDF_clean$Booster_Doses_65Plus <- gsub(",","", covidDF_clean$Booster_Doses_65Plus)
covidDF_clean$Booster_Doses_65Plus <- as.numeric(covidDF_clean$Booster_Doses_65Plus)
covidDF_clean$Booster_Doses_0to12_group <- covidDF_clean$Booster_Doses -
covidDF_clean$Booster_Doses_12Plus
covidDF_clean$Booster_Doses_12to18_group <- covidDF_clean$Booster_Doses_12Plus -
covidDF_clean$Booster_Doses_18Plus
covidDF_clean$Booster_Doses_18to65_group <- covidDF_clean$Booster_Doses_18Plus -
covidDF_clean$Booster_Doses_65Plus

covidDF_clean$Census2019 <- gsub(",","",covidDF_clean$Census2019)
covidDF_clean$Census2019 <- as.numeric(covidDF_clean$Census2019)
covidDF_clean$Census2019_12PlusPop <-
gsub(",","",covidDF_clean$Census2019_12PlusPop)
covidDF_clean$Census2019_12PlusPop <-
as.numeric(covidDF_clean$Census2019_12PlusPop)
covidDF_clean$Census2019_18PlusPop<-gsub(",","",covidDF_clean$Census2019_18PlusPop)
covidDF_clean$Census2019_18PlusPop<-
as.numeric(covidDF_clean$Census2019_18PlusPop)
covidDF_clean$Census2019_65PlusPop<-gsub(",","",covidDF_clean$Census2019_65PlusPop)
covidDF_clean$Census2019_65PlusPop <-
as.numeric(covidDF_clean$Census2019_65PlusPop)
covidDF_clean$Census2019_0to12_group <- covidDF_clean$Census2019 -
covidDF_clean$Census2019_12PlusPop
covidDF_clean$Census2019_12to18_group <- covidDF_clean$Census2019_12PlusPop -
covidDF_clean$Census2019_18PlusPop
covidDF_clean$Census2019_18to65_group <- covidDF_clean$Census2019_18PlusPop -
covidDF_clean$Census2019_65PlusPop

D <- unique(covidDF_clean$Date)
D ##using D variable to compare if there are any missing values from the original dataset
states <- unique(covidDF_clean$Recip_State)
states  ##using states variable to compare if there are any missing values from the original
dataset
fips <- unique(covidDF_clean$FIPS)
length(fips)
length(unique(covidDF_clean$Date)) ##using length to compare if there are any missing
values from the original dataset
length(unique(covidDF_clean$Recip_County)) ##using length to compare if there are any
missing values from the original dataset
```

Data cleaning in Python

The code delineates a sequence of operations crucial for refining and structuring the dataset, priming it for subsequent analytical exploration. The key steps encapsulated within this script are detailed below:

Data Loading and Initial Setup:

The code initiates by importing essential libraries - Pandas, Math, and NumPy. The dataset, stored as a CSV file, is loaded into a Pandas DataFrame, denoted as dataset. The “pd.read_csv()” function facilitates this process, allowing for the specification of the file path, skipping specific rows (skiprows), specifying header rows (header), and the number of rows to read (nrows). The original dataset contains data for years 2020 – 2023. In this survey we focused on the period between years 2021 and 2022 since the need for vaccination intensified between these years, when the pandemic was on the increase.

Cleaning Numeric Columns:

The code identifies columns containing strings that represent numeric values essential for analysis. Columns to be cleaned and converted are specified in the “columns_to_convert” list. A custom-defined function, “clean_and_convert”, employs regular expressions (re.sub()) to remove all non-numeric characters from string values. This function ensures uniformity and consistency in numeric data representation. The “clean_and_convert()” function systematically cleans strings and converts them to floating-point numbers. If a cleaned string is empty, it's replaced with “NaN” (not a number) to denote missing or non-applicable values.

Feature Engineering and Data Transformation:

The script undertakes feature engineering by creating new columns derived from existing data to provide additional insights into vaccination demographics and rates. The creation of demographic subdivisions based on census populations across distinct age groups (0-11, 12-17, 18-64, and 65+) enhances the granularity of the dataset, facilitating a more nuanced analysis. Calculations of vaccination percentages within specific demographic cohorts are computed to derive a deeper understanding of vaccination rates across diverse population segments.

Column Deletion and Missing Value Handling:

The columns referring to the different types of vaccination and the population by age groups (5Plus, 12Plus, 18plus, 65plus) are now redundant for later analysis, since the age groups have been updated and separated into new age strata (0-11, 12-17, 18-64, and 65+). Furthermore, the columns containing percentage calculations and total values have also been removed from the dataset using the "drop()" function, as they can now be calculated whenever necessary.

Data Imputation and Standardization:

The script addresses missing values within certain columns by employing strategies such as replacing missing state information ('Recip_State') with 'Unknown State' to ensure consistency and completeness in the dataset. Values representing 'UNK' or unknown in specific columns ('Recip_State', 'FIPS') are standardized for improved data clarity and consistency.

Datetime Manipulation and Conditional Filling:

The "Date" column is converted to standard datetime format using the function "pd.to_datetime()", ensuring uniformity and facilitating time-based analysis. Conditional

statements are used to strategically fill in missing values in selected columns with zeros based on date conditions to maintain data consistency and accuracy. Specifically, for values where the "Date" is before 15 December 2021, missing values have been replaced with zero, as vaccinations had not yet started before that date

Python code:

```
import pandas as pd
import math
import numpy as np

dataset = pd.read_csv(r"C:\Users\fani_\OneDrive\Υπολογιστής\COVID-
19_Vaccinations_in_the_United_States_County.csv",
                skiprows = range(1, 62378),
                header=0,
                nrows = 1838255)

import re

# Columns to clean and convert to numeric type
columns_to_convert = ['Administered_Dose1_Recip','Administered_Dose1_Pop_Pct'
,'Administered_Dose1_Recip_12Plus' , 'Administered_Dose1_Recip_12PlusPop_Pct',
'Administered_Dose1_Recip_18Plus', 'Administered_Dose1_Recip_18PlusPop_Pct',
'Administered_Dose1_Recip_65Plus', 'Administered_Dose1_Recip_65PlusPop_Pct',
'Booster_Doses', 'Booster_Doses_Vax_Pct', 'Booster_Doses_12Plus',
'Booster_Doses_12Plus_Vax_Pct', 'Booster_Doses_18Plus',
'Booster_Doses_18Plus_Vax_Pct', 'Booster_Doses_65Plus',
'Booster_Doses_65Plus_Vax_Pct', 'Series_Complete_Yes', 'Series_Complete_Pop_Pct',
'Series_Complete_12Plus', 'Series_Complete_12PlusPop_Pct', 'Series_Complete_18Plus',
'Series_Complete_18PlusPop_Pct', 'Series_Complete_65Plus',
'Series_Complete_65PlusPop_Pct', 'Series_Complete_Pop_Pct_SVI',
'Series_Complete_12PlusPop_Pct_SVI', 'Series_Complete_18PlusPop_Pct_SVI',
'Series_Complete_65PlusPop_Pct_SVI', 'Census2019', 'Census2019_12PlusPop',
'Census2019_18PlusPop', 'Census2019_65PlusPop']

# Define a function to clean the strings and convert to numeric
def clean_and_convert(value):
   if isinstance(value, str):
      value = re.sub(r'[^\d.]', '', value)  # Remove all non-numeric characters except dots
      return float(value) if value else np.nan  # Convert the cleaned string to float or NaN if
empty
   else:
      return value  # Return original value if not a string

# Clean strings and convert to numeric
dataset[columns_to_convert]=dataset[columns_to_convert].map(clean_and_convert)
```

```
dataset['Census2019_65Plus'] = dataset['Census2019_65PlusPop']
dataset['Census2019_18_64'] = dataset['Census2019_18PlusPop'] -
dataset['Census2019_65PlusPop']
dataset['Census2019_12_17'] = dataset['Census2019_12PlusPop'] -
dataset['Census2019_18PlusPop']
dataset['Census2019_0_11'] = dataset['Census2019'] - dataset['Census2019_12PlusPop']

dataset['Administered_Dose1_Recip_18_64'] =
dataset['Administered_Dose1_Recip_18Plus'] -
dataset['Administered_Dose1_Recip_65Plus']
dataset['Administered_Dose1_Recip_12_17'] =
dataset['Administered_Dose1_Recip_12Plus'] -
dataset['Administered_Dose1_Recip_18Plus']
dataset['Administered_Dose1_Recip_0_11'] = dataset['Administered_Dose1_Recip'] -
dataset['Administered_Dose1_Recip_12Plus']

dataset['Administered_Dose1_Pop_pct'] = dataset['Administered_Dose1_Recip'] /
dataset['Census2019']
dataset['Administered_Dose1_Recip_65Plus_pct'] =
dataset['Administered_Dose1_Recip_65Plus'] / dataset['Census2019_65PlusPop']
dataset['Administered_Dose1_Recip_12_17_pct'] =
dataset['Administered_Dose1_Recip_12_17'] / dataset['Census2019_12_17']
dataset['Administered_Dose1_Recip_0_11_pct'] =
dataset['Administered_Dose1_Recip_0_11'] / dataset['Census2019_0_11']
dataset['Administered_Dose1_Recip_18_64_pct'] =
dataset['Administered_Dose1_Recip_18_64'] / dataset['Census2019_18_64']

dataset['Series_Complete_18_64'] = dataset['Series_Complete_18Plus'] -
dataset['Series_Complete_65Plus']
dataset['Series_Complete_12_17'] = dataset['Series_Complete_12Plus'] -
dataset['Series_Complete_18Plus']
dataset['Series_Complete_0_11'] = dataset['Series_Complete_Yes'] -
dataset['Series_Complete_12Plus']

dataset['Series_Complete_Pop_pct'] = dataset['Series_Complete_Yes'] /
dataset['Census2019']
dataset['Series_Complete_65Plus_pct'] = dataset['Series_Complete_65Plus'] /
dataset['Census2019_65PlusPop']
dataset['Series_Complete_18_64_pct'] = dataset['Series_Complete_18_64'] /
dataset['Census2019_18_64']
dataset['Series_Complete_12_17_pct'] = dataset['Series_Complete_12_17'] /
dataset['Census2019_12_17']
dataset['Series_Complete_0_11_pct'] = dataset['Series_Complete_0_11'] /
dataset['Census2019_0_11']
```

```
dataset['Booster_Doses_18_64'] = dataset['Booster_Doses_18Plus'] -
dataset['Booster_Doses_65Plus']
dataset['Booster_Doses_12_17'] = dataset['Booster_Doses_12Plus'] -
dataset['Booster_Doses_18Plus']
dataset['Booster_Doses_0_11'] = dataset['Booster_Doses'] -
dataset['Booster_Doses_12Plus']

dataset['Booster_Doses_Vax_pct'] = dataset['Booster_Doses'] /
dataset['Series_Complete_Yes']
dataset['Booster_Doses_65Plus_pct'] = dataset['Booster_Doses_65Plus'] /
dataset['Series_Complete_65Plus']
dataset['Booster_Doses_18_64_pct'] = dataset['Booster_Doses_18_64'] /
dataset['Series_Complete_18_64']
dataset['Booster_Doses_12_17_pct'] = dataset['Booster_Doses_12_17'] /
dataset['Series_Complete_12_17']
dataset['Booster_Doses_0_11_pct'] = dataset['Booster_Doses_0_11'] /
dataset['Series_Complete_0_11']

# List of columns to delete
columns_to_delete = ['Administered_Dose1_Recip_5Plus',
'Administered_Dose1_Recip_12Plus', 'Administered_Dose1_Recip_18Plus',
            'Series_Complete_12Plus', 'Series_Complete_18Plus', 'Booster_Doses_12Plus',
'Booster_Doses_18Plus', 'Booster_Doses_50Plus',
            'Second_Booster_50Plus', 'Census2019_12PlusPop', 'Census2019_18PlusPop',
'Administered_Dose1_Pop_Pct',
            'Administered_Dose1_Recip_5PlusPop_Pct',
'Administered_Dose1_Recip_12PlusPop_Pct', 'Administered_Dose1_Recip_18PlusPop_Pct',
            'Administered_Dose1_Recip_65PlusPop_Pct', 'Series_Complete_5Plus',
'Series_Complete_12PlusPop_Pct', 'Series_Complete_Pop_Pct',
            'Series_Complete_5PlusPop_Pct', 'Series_Complete_18PlusPop_Pct',
'Series_Complete_65PlusPop_Pct', 'Booster_Doses_Vax_Pct',
            'Booster_Doses_5Plus', 'Booster_Doses_5Plus_Vax_Pct',
'Booster_Doses_12Plus_Vax_Pct', 'Booster_Doses_18Plus_Vax_Pct',
            'Booster_Doses_65Plus_Vax_Pct', 'Second_Booster_65Plus_Vax_Pct',
'Booster_Doses_50Plus_Vax_Pct', 'MMWR_week', 'Series_Complete_5to17',
            'Series_Complete_5to17Pop_Pct', 'Series_Complete_5PlusPop_Pct_SVI',
'Series_Complete_5to17Pop_Pct_SVI', 'Series_Complete_Pop_Pct_UR_Equity',
            'Series_Complete_5PlusPop_Pct_UR_Equity',
'Series_Complete_5to17Pop_Pct_UR_Equity', 'Series_Complete_12PlusPop_Pct_UR_Equity',
            'Series_Complete_18PlusPop_Pct_UR_Equity',
'Series_Complete_65PlusPop_Pct_UR_Equity', 'Booster_Doses_Vax_Pct_SVI',
            'Booster_Doses_12PlusVax_Pct_SVI', 'Booster_Doses_18PlusVax_Pct_SVI',
'Booster_Doses_65PlusVax_Pct_SVI', 'Booster_Doses_Vax_Pct_UR_Equity',
            'Booster_Doses_12PlusVax_Pct_UR_Equity',
'Booster_Doses_18PlusVax_Pct_UR_Equity', 'Booster_Doses_65PlusVax_Pct_UR_Equity',
```

```
            'Census2019_5PlusPop', 'Census2019_5to17Pop', 'Census2019_65PlusPop',
'Bivalent_Booster_5Plus', 'Bivalent_Booster_5Plus_Pop_Pct',
            'Bivalent_Booster_12Plus', 'Bivalent_Booster_12Plus_Pop_Pct',
'Bivalent_Booster_18Plus', 'Bivalent_Booster_18Plus_Pop_Pct',
            'Bivalent_Booster_65Plus', 'Bivalent_Booster_65Plus_Pop_Pct',
'Second_Booster_65Plus', 'Second_Booster_65Plus_Vax_Pct']

# Drop the specified columns and assign the result back to dataset
dataset = dataset.drop(columns=columns_to_delete)
# OR: Use inplace=True to modify dataset directly
# dataset.drop(columns=columns_to_delete, inplace=True)
dataset.shape

for column in dataset.columns:
   missing_values = dataset[column].isnull().sum()  # Count missing values for each column
   if missing_values > 0:
      print(f"Column '{column}' has {missing_values} missing values.")

dataset['Recip_State'] = dataset['Recip_State'].fillna('Unknown State')

print(all(dataset['Recip_State'].isnull()))
dataset['Recip_State'].unique()

dataset['Recip_State'].replace('UNK', 'Unknown State', inplace=True)
dataset['Recip_State'].unique()

unique_values = dataset['FIPS'].unique()
# Display unique values in batches of 10
for i in range(0, len(unique_values), 50):
   print(unique_values[i:i+50])

dataset['FIPS'].replace('UNK', 'Unknown', inplace=True)
len(dataset[dataset['FIPS'] == 'Unknown'])

dataset[dataset['FIPS'] == 'UNK']  # Filter the DataFrame
dataset['Metro_status'] = dataset['Metro_status'].fillna('Unknown')
dataset['Metro_status'].unique()

dataset['SVI_CTGY'] = dataset['SVI_CTGY'].fillna('Unknown')

dataset['Date'] = pd.to_datetime(dataset['Date'])
condition = dataset['Date'] < pd.to_datetime('2021-12-15')
columns_to_update = ['Booster_Doses', 'Booster_Doses_Vax_pct', 'Booster_Doses_0_11',
'Booster_Doses_0_11_pct', 'Booster_Doses_12_17', 'Booster_Doses_12_17_pct',
```

```
'Booster_Doses_18_64', 'Booster_Doses_18_64_pct', 'Booster_Doses_65Plus',
'Booster_Doses_65Plus_pct']
dataset.loc[condition, columns_to_update] = dataset.loc[condition,
columns_to_update].fillna(0)

condition2 = dataset['Date'] < pd.to_datetime('2022-01-27')
columns_to_update = ['Booster_Doses_0_11', 'Booster_Doses_0_11_pct',
'Booster_Doses_12_17', 'Booster_Doses_12_17_pct']
dataset.loc[condition2, columns_to_update] = dataset.loc[condition2,
columns_to_update].fillna(0)
```

Data cleaning in SQL

The COALESCE function is utilized in conjunction with the “Population” column, potentially handling null values in that column. If the “Population” column has null values, the COALESCE function tries to fill those null values. It first checks the “Population” column, if it's null, it uses the result of the window function (MAX with OVER) partitioned by “Recip_County” to replace the null value. Essentially, this code ensures that if the “Population” column contains null values, it replaces those nulls with the maximum population value observed within the same “Recip_County” group. The reason it fills the null values this way is due to the fact that that the population is a fixed size for every county and every census value refers to the same date.

SQL code:

```
COALESCE(vu.Census, MAX(vu.Census) OVER (PARTITION BY vu.Recip_County)) AS
'Population'
```

Data Warehouse Design and Implementation

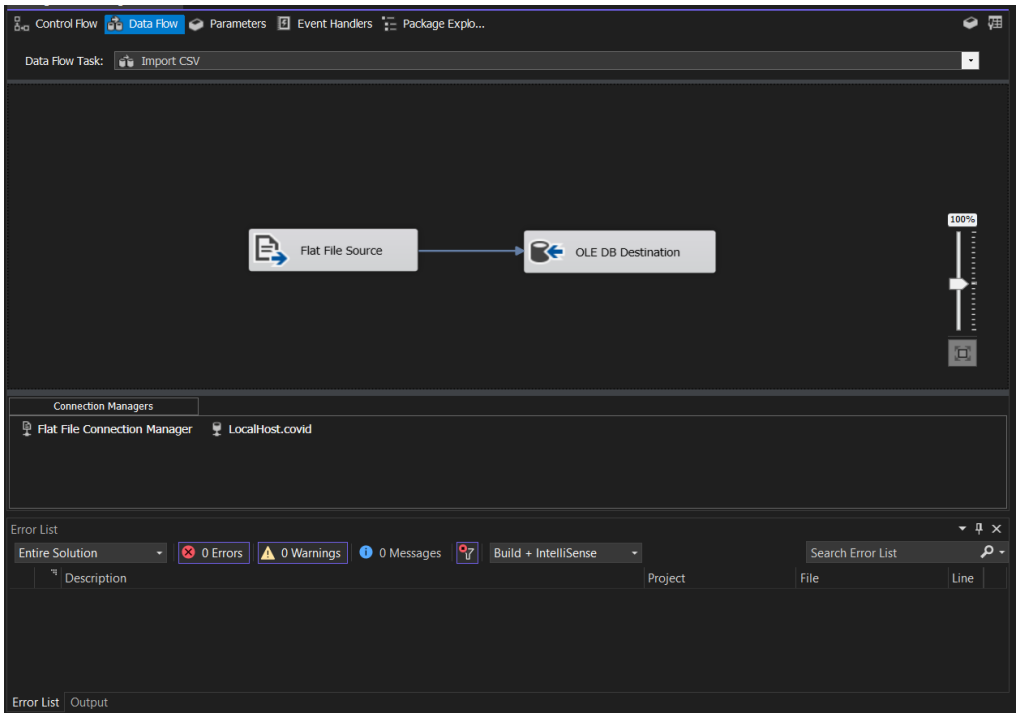
The path from acquiring raw data to information that can be used for action involves the critical step of structuring and organizing the data into a robust framework. In this section, we delve into the complex process of designing and implementing a data warehouse to host the curated COVID-19 vaccination dataset. Leveraging the capabilities of SQL Server Management Studio (SSMS) and SQL Server Integration Services (SSIS) from Microsoft Visual Studio, our goal is to design a robust and efficient data warehouse that harmonizes disparate datasets into a coherent structure.

This phase transcends mere data storage; it's a meticulously orchestrated symphony where disparate data sources converge, undergo transformation, and find residence within a unified schema. The design decisions made here resonate deeply, shaping the foundation upon which subsequent analytics, querying, and reporting thrive. Through a combination of structural blueprints, integration workflows, and data loading strategies, this section illuminates the meticulous steps taken to mold raw data into a coherent and accessible form, ready to empower insightful analysis and visualization.

In this section, we will address the essence of integrating and enhancing data through the extraction, transformation and loading (ETL) process. Here, we explore the meticulous steps involved in shaping raw data into a powerful asset that fuels business intelligence and informed decision making. At the same time, in the context of Data Warehousing, we focus on the construction of multidimensional structures known as "cubes". This module reveals the intricacies of cube design and development, where different dimensions of data intersect to form an integrated framework. These cubes serve as the foundation for OLAP (Online Analytical Processing) capabilities, facilitating efficient data analysis and exploration.

PART A: ETL

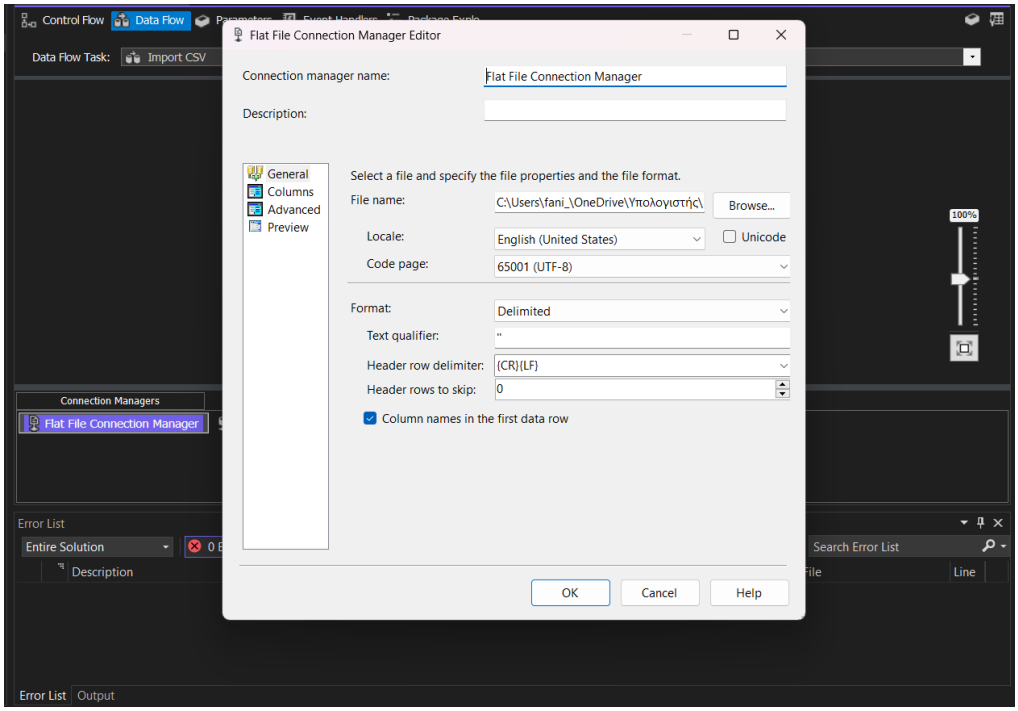
First of all, it is necessary to create a Data Flow which will export the data from the csv file and will import them to a database for further editing in SQL. In the “Control Flow” we select “Data Flow Task” and we name it “Import CSV”. Then, we select “edit ” and choose a Flat File Source and we connect it to OLE DB Destination which will load the required data into the database.



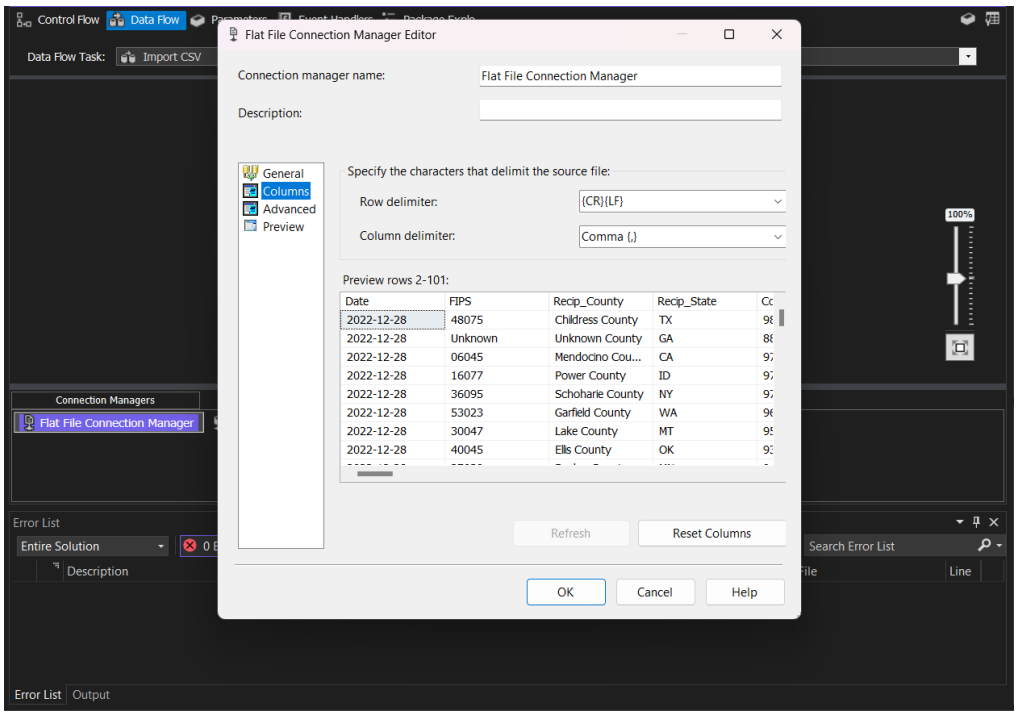
Flat File Source setting

Opening the Flat File Connection Manager, in “General” tab, we have to import the csv file into the empty “File name”, we select the option “Browse” and we select the file which contains the data we are going to use. After that we make the necessary settings so that the file is read correctly. We alter the “Code Page” to 65001 which refers to a specific character encoding called UTF-8 that can represent almost every character in any language in the world. It's flexible because it can represent characters from many different languages and also

includes special characters. Next, we set “Format” to “Delimited” as the delimiter of the csv file is the comma character (,) and the “Text qualifier” is quotation marks (“”).

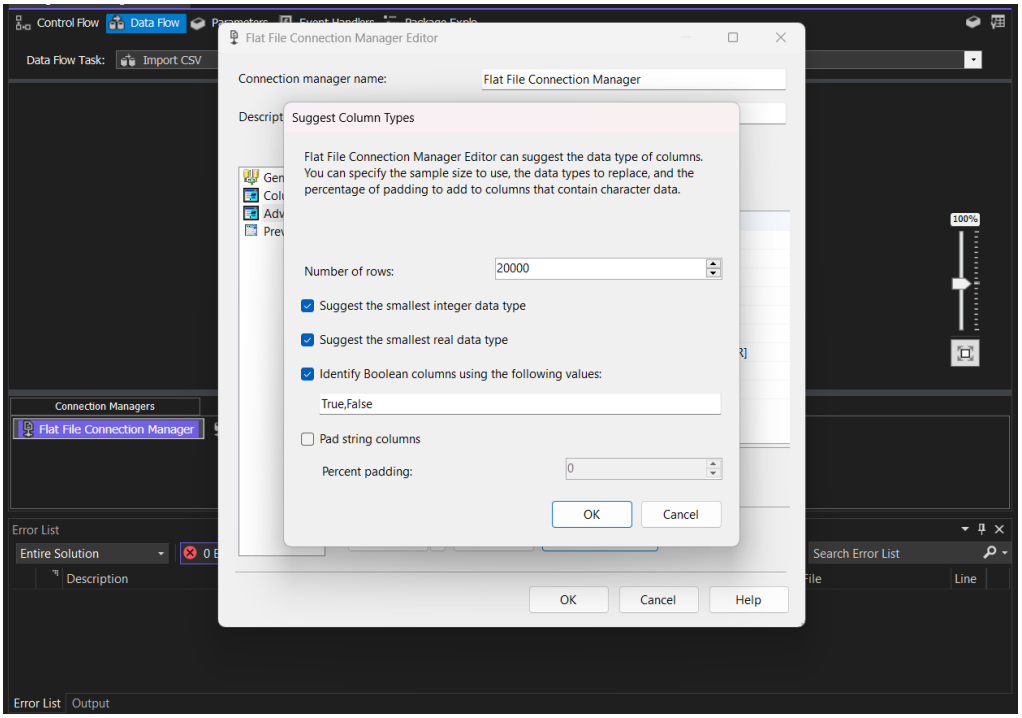


We move to “Columns” tab to check that the data have been read correctly.

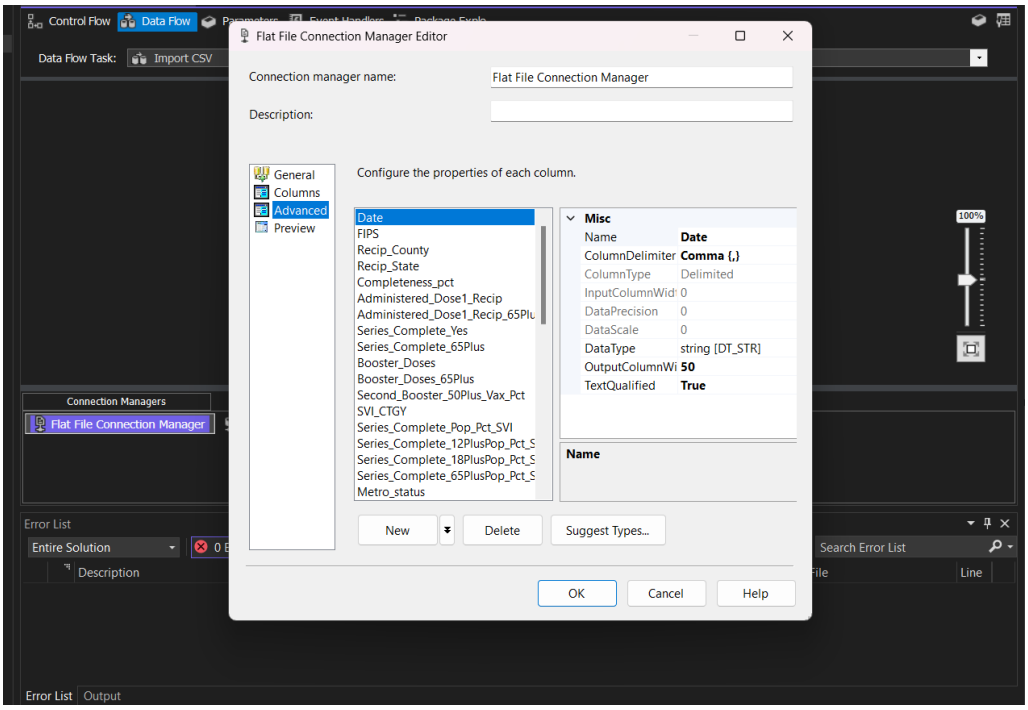


We move to “Advanced” tab. In this tab we are going to set our data types. We have the option to let the importer, Visual Studio, suggest the type of the data and we set its sample

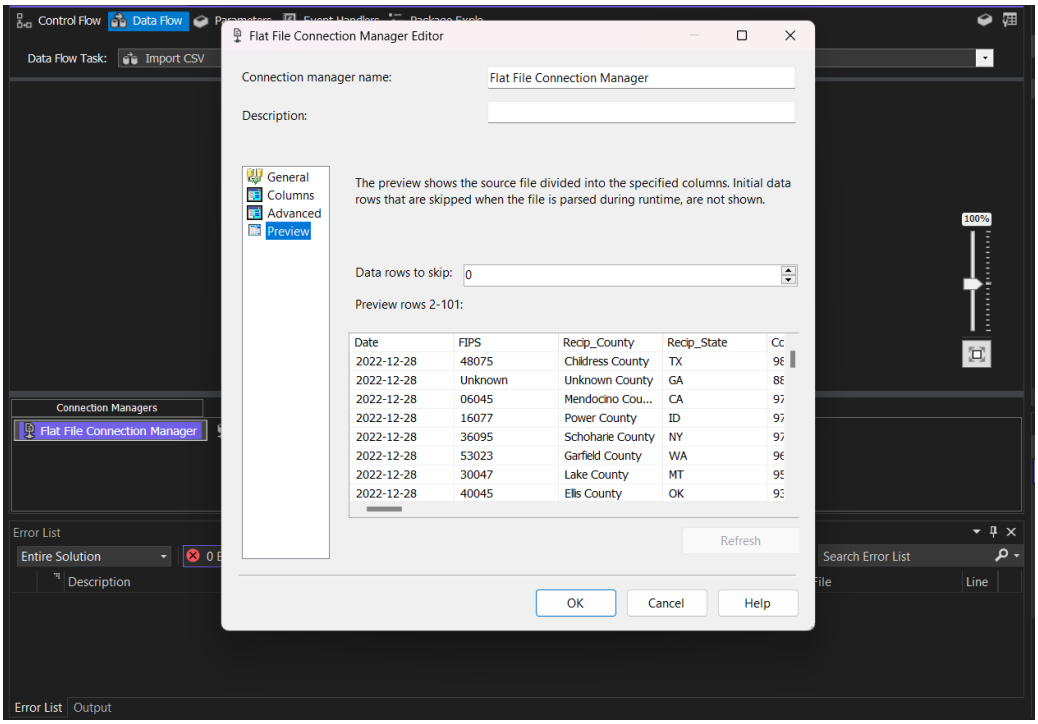
to 20.000 rows. This is something very useful as we save time and we avoid making mistakes.



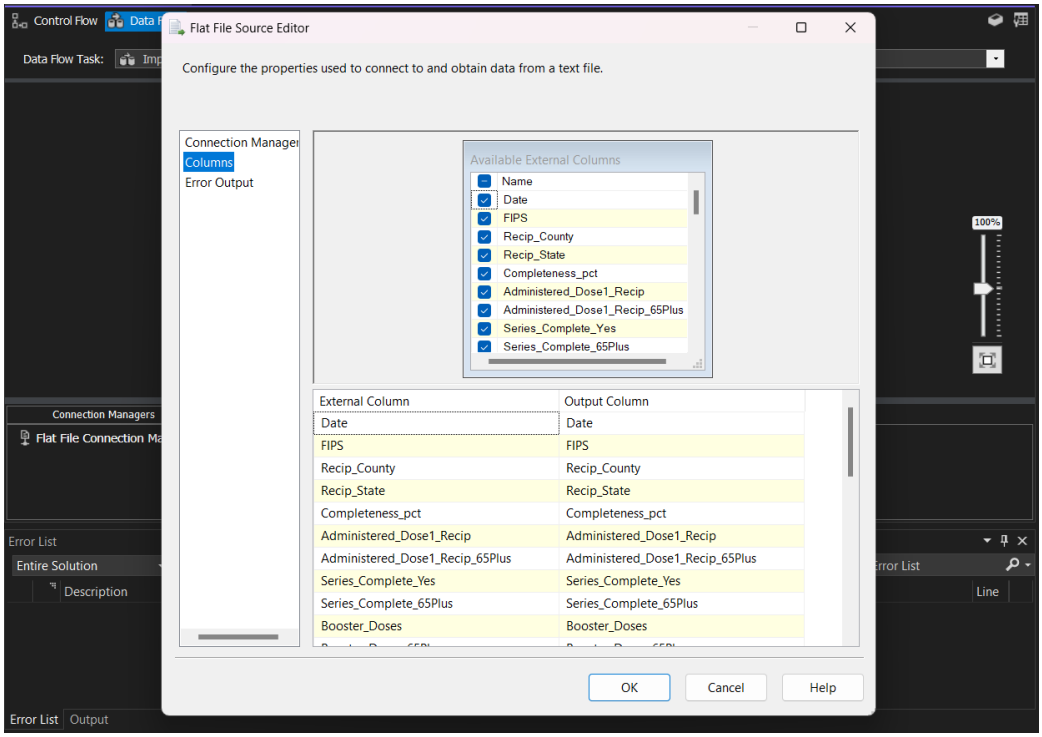
After the suggestions of importer we make the appropriate corrections untill data are ready for editing. One of the main challenges we faced at this part was the type of the variable Date, as it should be set as a string otherwise it was not recognised from Visual Studio and and the output was error. The rest of the importer's suggestions were correct.



Moving to the “Preview” tab we take a final look to the data and we confirm they are correct.

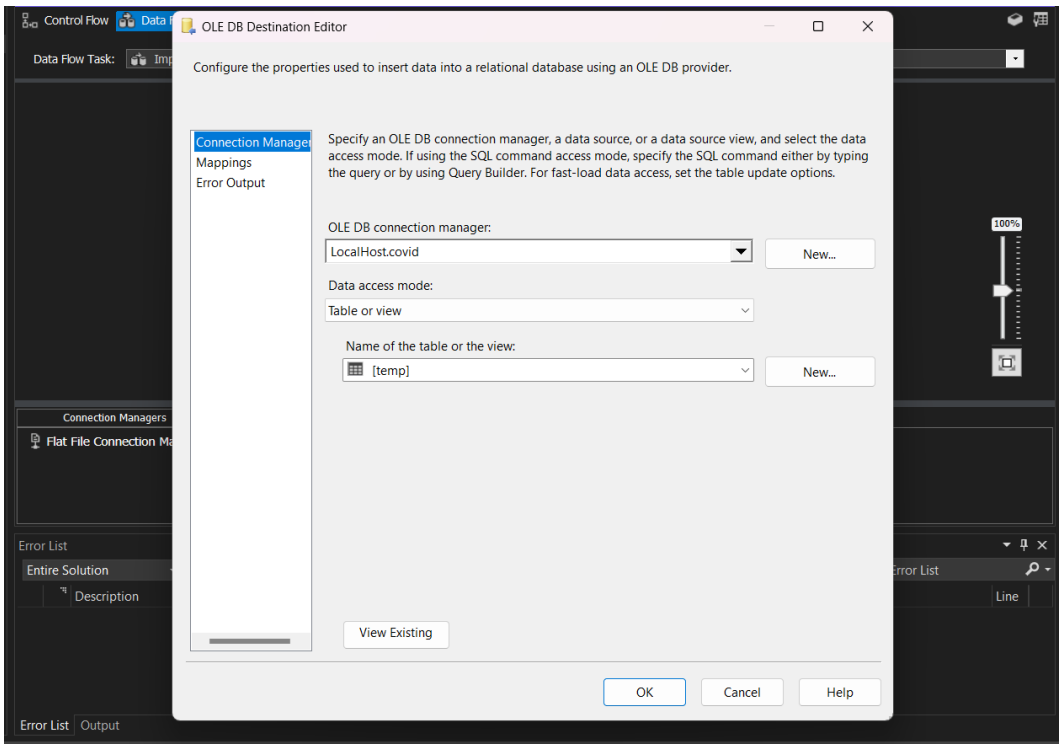


Moving forward from “Connection Manager ” to “Columns” tab we select the columns that we are going to keep for analysis.

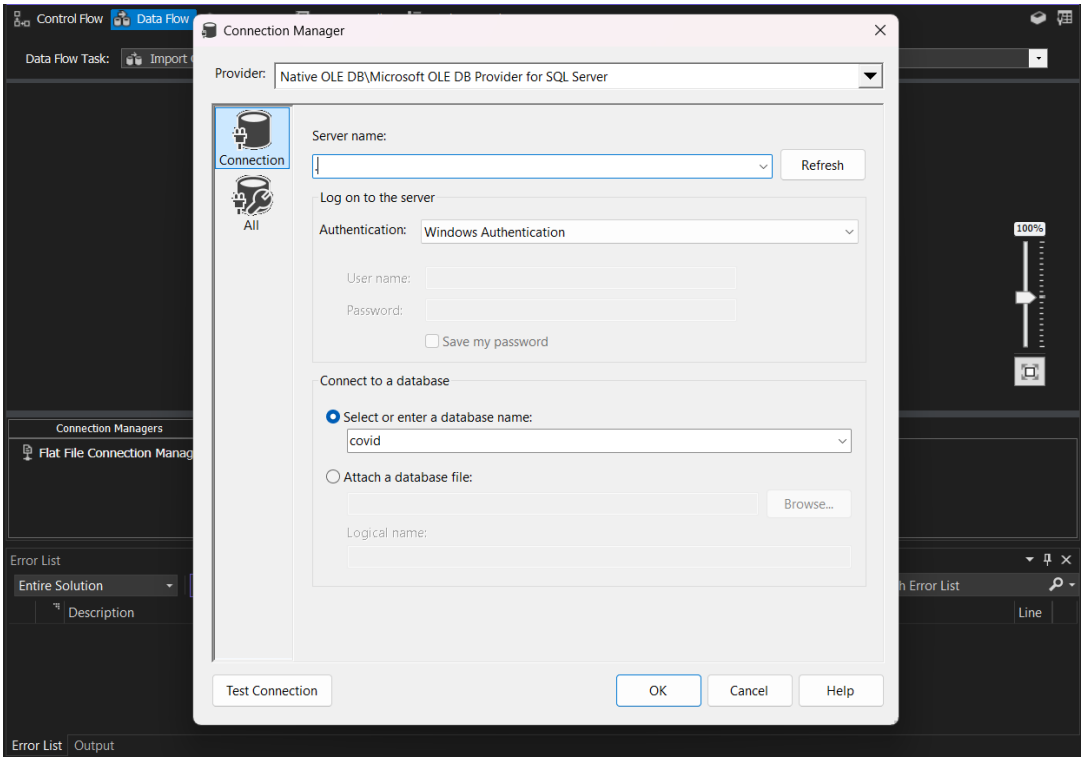


<u>OLE DB Destination setting</u>

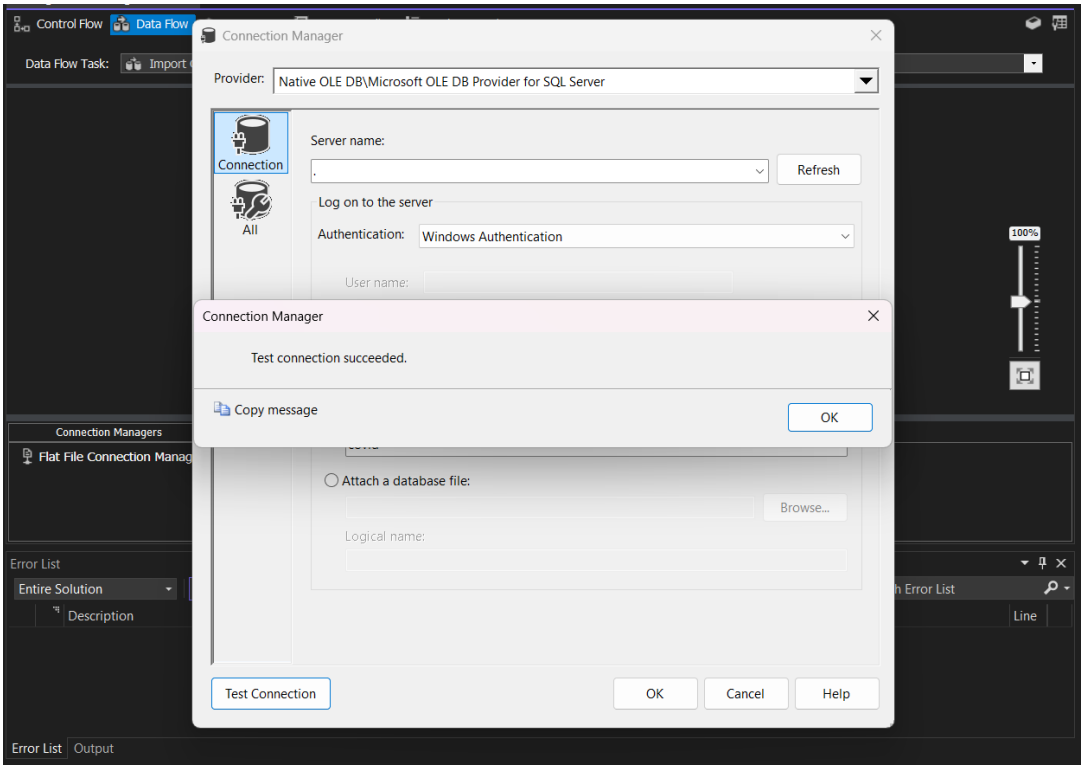
In the “Connection Manager” tab we select “New” in order to connect to the server which will contain the database we will use.



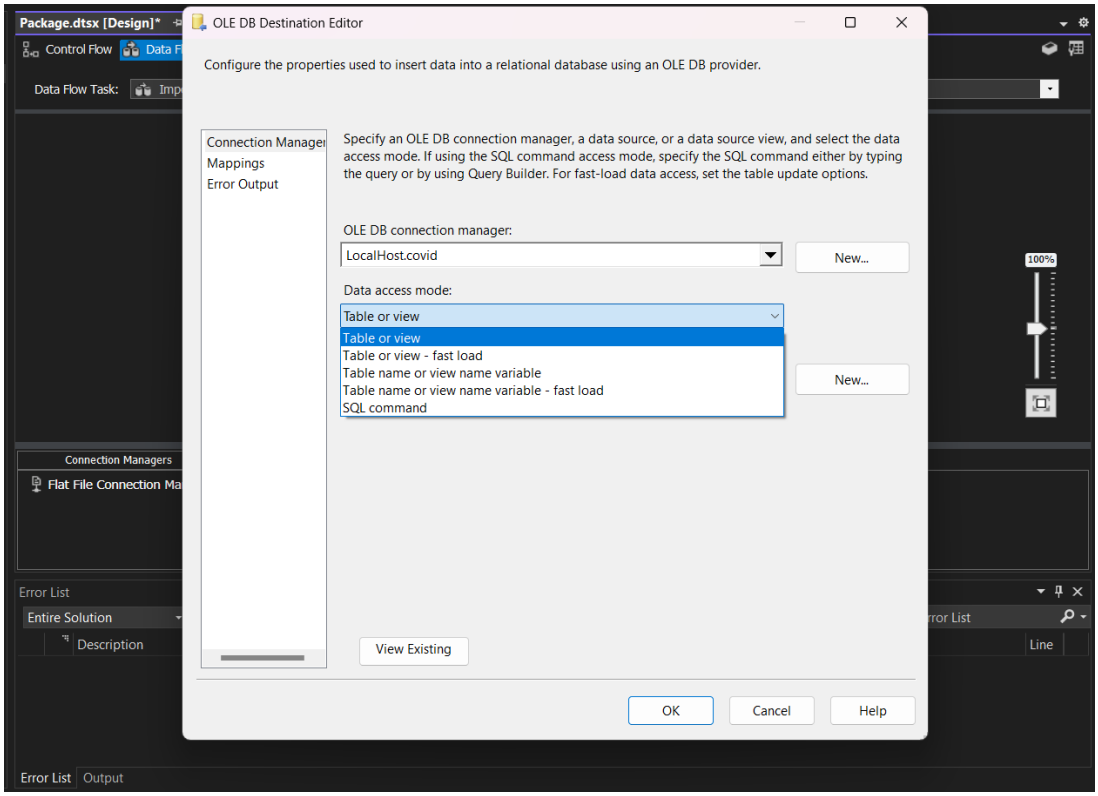
In the empty “Server name” we put the dot (.), which corresponds to the local server we are using and as “Authentication” we select to connect via Windows Authenticotion. Then we enter the database name which is “covid”.



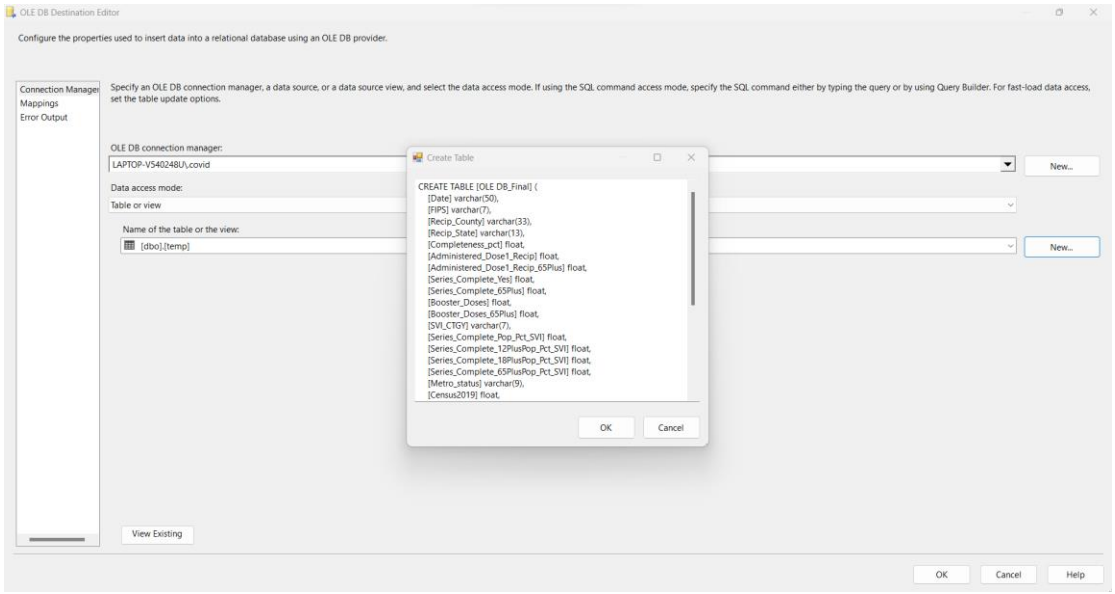
If we test the connection we created we see the message ”Connection succeeded”, so we are able to continue.



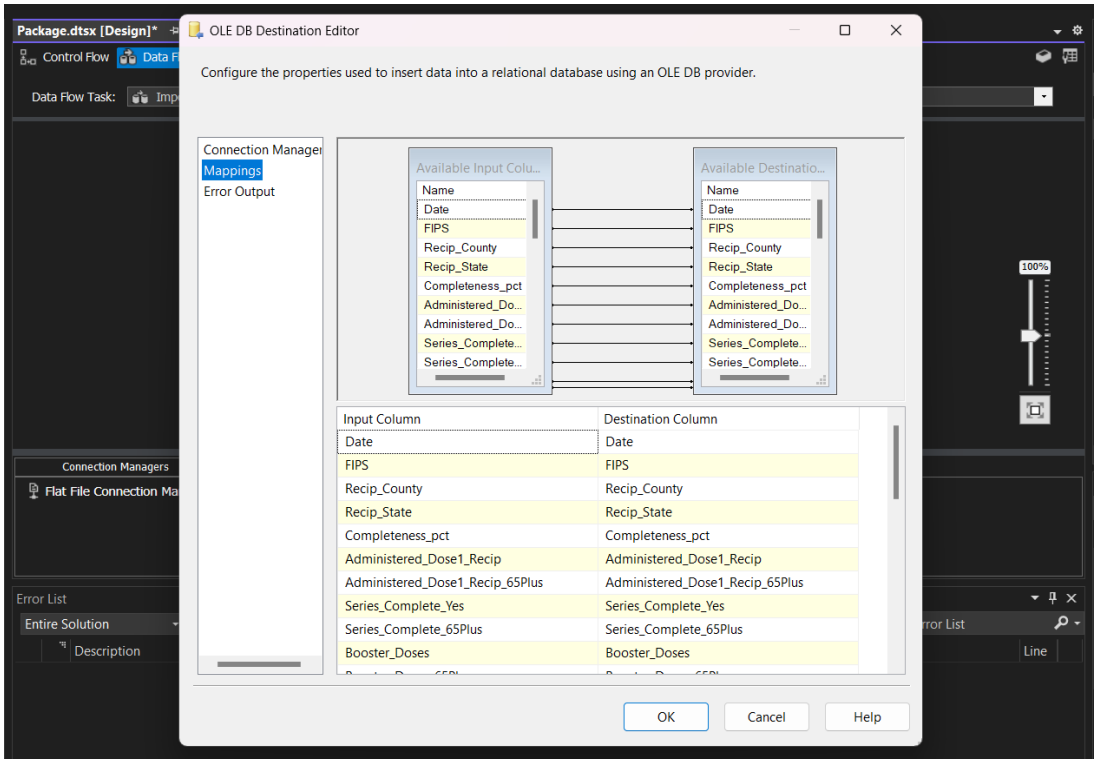
Turning back to the “Connection Manager” tab, we select “Table or View” in the “Data access mode”.



To fill the “Name of the table or the view” we select “New” and we see a query in which is created a Table with the columns we have inserted from our data. We name this Table “OLE DB_Final”.



Moving to tab “Mapping”, we make sure that the input and the destination columns have been assigned to each other and this is very important for the proper export of the data.



To end this part all we have to do is select the control flow, press start and confirm it works. Afterwards, we open Microsoft SQL Server Studio Management (SSMS) we select our database and we refresh the tables expecting the table “OLE DB_Final” we created.

In this part we are going to apply “UNPIVOT” function in table “OLE DB_Final” and then aggregate the data in order to convert the columns which contain vaccination types and census with age groups into rows, achieving better data management. This process is going to be saved as a view named “view_unpivoting”.

In more detail,
In the subquery we select multiple columns related to vaccination data (Date, FIPS, Recip_County, Recip_State, etc.) from table OLE DB_Final. Using the UNPIVOT function is used to convert columns representing different age groups and vaccination types into rows.

The outer Query selects specific columns from the subquery and uses UNPIVOT function to transform the data further, creating new columns (VaccinationCount) by unpivoting different vaccination types based on age groups.
Then, it uses MAX and CASE statements to pivot the data again, this time aggregating the vaccination counts based on the different types (Administered, Series, Booster, Census) and selecting the maximum value for each group. It groups the data by various columns (Date, FIPS, Recip_County, Recip_State, etc.) and uses CASE statements within a COALESCE function to categorize the vaccination data into age groups (0-11, 12-17, 18-64, 65+, Total) based on the vaccination type.

<u>Unpivot code in SQL:</u>

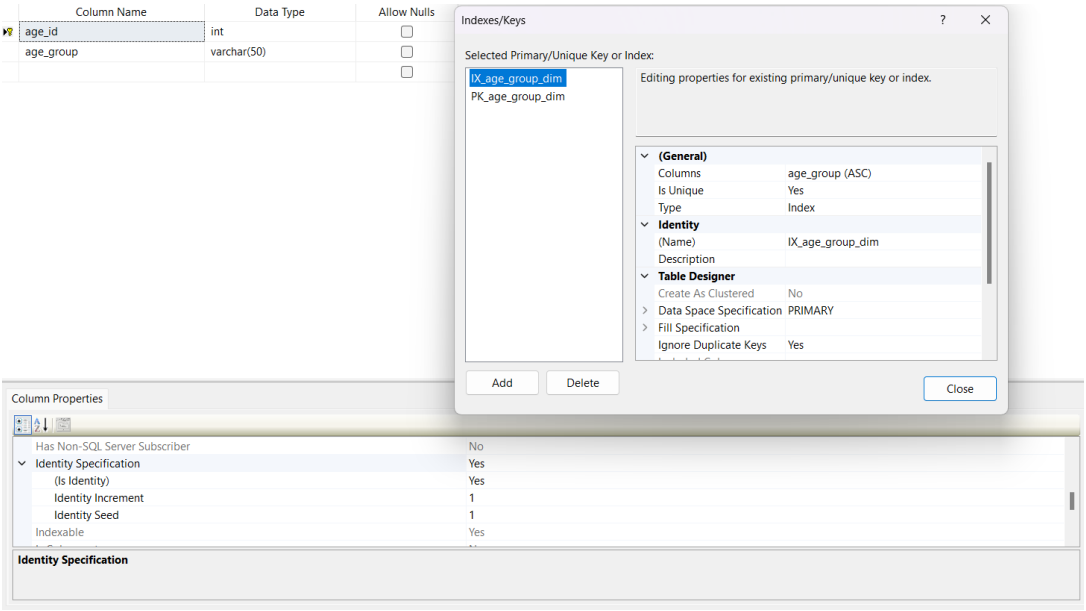
```
SELECT
  Date,
  FIPS,
  Recip_County,
  Recip_State,
  SVI_CTGY,
  Metro_status,
  COALESCE(CASE
    WHEN VaccineType LIKE '%_0_11' THEN '0-11'
    WHEN VaccineType LIKE '%_12_17' THEN '12-17'
    WHEN VaccineType LIKE '%_18_64' THEN '18-64'
    WHEN VaccineType LIKE '%_65Plus' THEN '65+'
    WHEN VaccineType IS NULL THEN 'Total'
    ELSE NULL
  END, 'Total') AS age_group,
  MAX(CASE WHEN VaccineType LIKE 'Administered%' THEN VaccinationCount END) AS
Administered,
  MAX(CASE WHEN VaccineType LIKE 'Series%' THEN VaccinationCount END) AS Series,
  MAX(CASE WHEN VaccineType LIKE 'Booster%' THEN VaccinationCount END) AS Booster,
  MAX(CASE WHEN VaccineType LIKE 'Census%' THEN VaccinationCount END) AS Census
FROM (
  SELECT
    Date,
    FIPS,
    Recip_County,
    Recip_State,
    SVI_CTGY,
    Metro_status,
    Administered_Dose1_Recip_0_11,
    Administered_Dose1_Recip_12_17,
    Administered_Dose1_Recip_18_64,
    Administered_Dose1_Recip_65Plus,
    Series_Complete_0_11,
    Series_Complete_12_17,
    Series_Complete_18_64,
    Series_Complete_65Plus,
    Booster_Doses_0_11,
    Booster_Doses_12_17,
    Booster_Doses_18_64,
    Booster_Doses_65Plus,
    Census2019_0_11,
    Census2019_12_17,
    Census2019_18_64,
    Census2019_65Plus,
```

```
      Administered_Dose1_Recip,
      Series_Complete_Yes,
      Booster_Doses,
      Census2019
   FROM  covid.dbo.OLE DB_Final
) AS Source
UNPIVOT (
   VaccinationCount FOR VaccineType IN (
      Administered_Dose1_Recip_0_11,
      Administered_Dose1_Recip_12_17,
      Administered_Dose1_Recip_18_64,
      Administered_Dose1_Recip_65Plus,
      Series_Complete_0_11,
      Series_Complete_12_17,
      Series_Complete_18_64,
      Series_Complete_65Plus,
      Booster_Doses_0_11,
      Booster_Doses_12_17,
      Booster_Doses_18_64,
      Booster_Doses_65Plus,
      Census2019_0_11,
      Census2019_12_17,
      Census2019_18_64,
      Census2019_65Plus,
      Administered_Dose1_Recip,
      Series_Complete_Yes,
      Booster_Doses,
      Census2019)
) AS Unpivoted
GROUP BY
   Date,
   FIPS,
   Recip_County,
   Recip_State,
   SVI_CTGY,
   Metro_status,
   COALESCE(CASE
      WHEN VaccineType LIKE '%_0_11' THEN '0-11'
      WHEN VaccineType LIKE '%_12_17' THEN '12-17'
      WHEN VaccineType LIKE '%_18_64' THEN '18-64'
      WHEN VaccineType LIKE '%_65Plus' THEN '65+'
      WHEN VaccineType IS NULL THEN 'Total'
      ELSE NULL
   END, 'Total');
```

After having execute the UNPIVOT, the next step is the creation of the dimension tables of our schema.

Creation of dimension "age_group_dim":

This table contains two columns. The first one is "age_id" which is Primary Key, its type is integrate, is unique and set with "Identity Specification" to "Yes", so it increases automatically. The second one is "age_group" which is varchar type and has been set as index, is unique and it ignores duplicand values in insertion. Neither of the columns allows null values.



To insert values in "age_group_dim", we create a code that executes a MERGE operation between two tables "age_group_dim" (referred to as Target) and a derived table from a SELECT DISTINCT query (referred to as Source), within the context of a database schema. The Source table is generated by extracting unique values from the "AgeGroup" column in the "view_unpivoting" table. The merging process is defined by the matching condition specified in the ON clause, which pairs Source.[AgeGroup] with Target.[age_group]. When no match is found based on this condition, the code performs an INSERT operation into the Target table, inserting values from the "AgeGroup" column of the Source table into the "age_group" column of the Target table. This code segment effectively updates the Target table by adding unique values from the Source table that do not already exist in the Target table, maintaining data integrity and completeness within the context of age-related information.

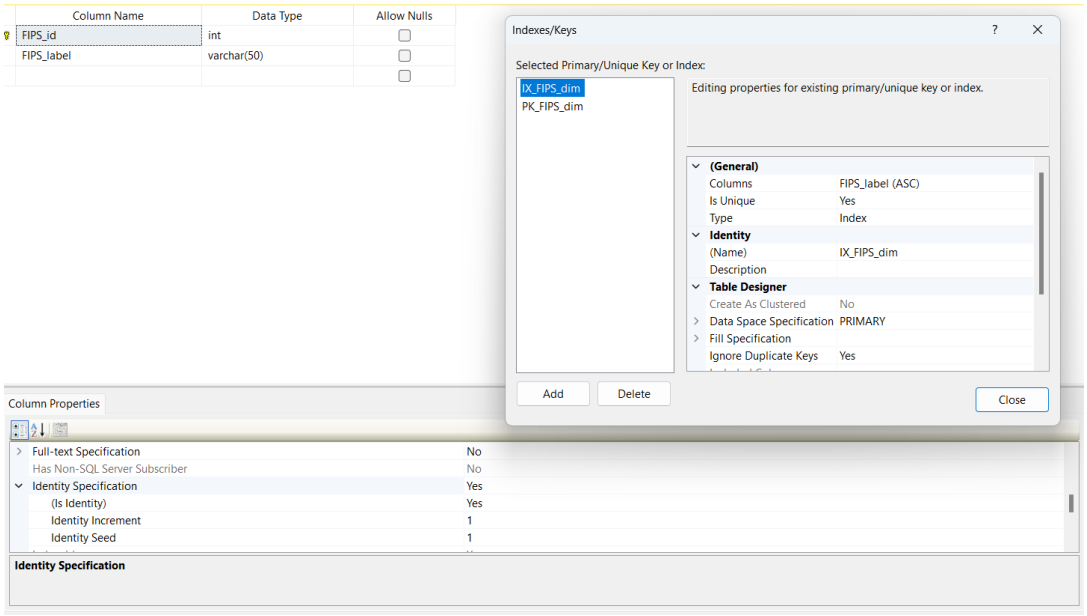
SQL code:

MERGE [covid].[dbo].[age_group_dim] AS Target

```
USING(
SELECT DISTINCT [AgeGroup]
FROM [covid].[dbo].[view_unpivoting]
) AS Source
ON (
        Source.[ AgeGroup] = Target.[age_group]
)
WHEN NOT MATCHED BY Target THEN
INSERT (age_group)
VALUES (Source.[AgeGroup]);
```

Creation of dimension "fips_dim":

This table contains two columns. The first one is "FIPS_id" which is Primary Key, its type is integrate, is unique and set with "Identity Specification" to "Yes", so it increases automatically. The second one is "FIPS_label" which is varchar type and has been set as index, is unique and it ignores duplicand values in insertion. Neither of the columns allows null values.



To insert values in "fips_dim" we create an SQL code that performs a MERGE operation between two tables "fips_dim" (referred to as Target) and a derived table from a SELECT DISTINCT query (referred to as Source) within a specific database schema. The Source table is generated by extracting unique values from the "FIPS" column in the "view_unpivoting" table. The merging process is defined by the matching condition specified in the ON clause, which pairs Source.[FIPS] with Target.[fips_label]. In instances where there's no match based on this condition, the code executes an INSERT operation into the Target table. It inserts values from the "FIPS" column of the Source table into the "fips_label" column of the Target table. This code snippet effectively updates the Target table by adding unique

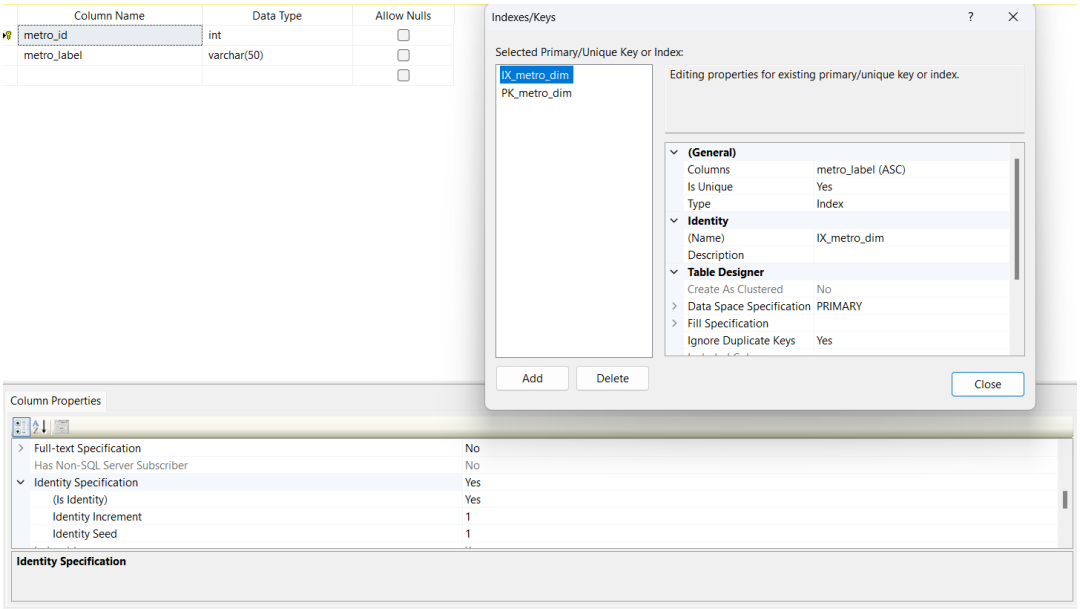
FIPS values from the Source table that do not already exist in the Target table, ensuring data completeness and integrity within the context of FIPS-related information.

SQL code:

```
MERGE [covid].[dbo].[fips_dim] AS Target
USING(
        SELECT DISTINCT [FIPS]
        FROM [covid].[dbo].[view_unpivoting]
) AS Source
ON (
        Source.[FIPS] = Target.[fips_label]
)
WHEN NOT MATCHED BY Target THEN
INSERT (fips_label)
VALUES (Source.[FIPS]);
```

Creation of dimension "metro_dim":

This table contains two columns. The first one is "metro_id" which is Primary Key, its type is integrate, is unique and set with "Identity Specification" to "Yes", so it increases automatically. The second one is "metro_label" which is varchar type and has been set as index, is unique and it ignores duplicand values in insertion. Neither of the columns allows null values.



To insert values in "metro_dim" we create an SQL code that performs a MERGE operation between two tables: "metro_dim" (referred to as Target) and a derived table created by selecting distinct values from the [Metro_status] column in the "view_unpivoting" table (referred to as Source). The merging logic is based on matching conditions defined in the ON

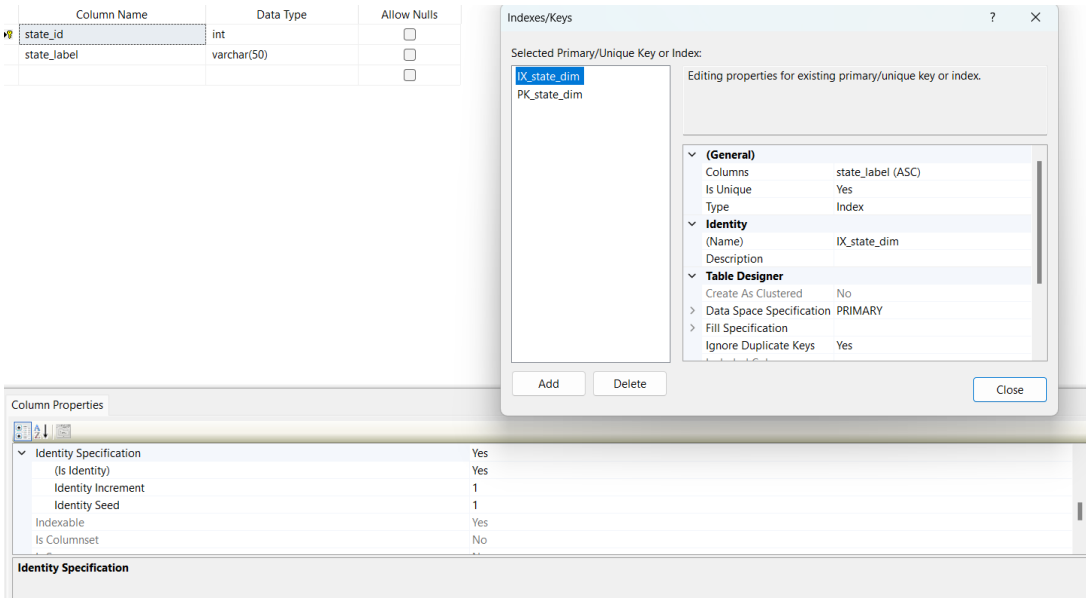
clause, linking Source.[Metro_status] to Target.[metro_label]. Whenever a match based on this condition isn't found, the code executes an INSERT operation into the Target table. It inserts values from the “Metro_status” column of the Source table into the “metro_label” column of the Target table. Essentially, this code snippet updates the Target table by incorporating unique values from the Source table's “Metro_status” column that are absent in the Target table, ensuring that the metro-related labels are comprehensive and complete in the context of the dimensional structure.

SQL code:

```
MERGE [covid].[dbo].[ metro_dim] AS Target
USING(
        SELECT DISTINCT [Metro_status]
        FROM [covid].[dbo].[view_ unpivoting]
) AS Source
ON (
        Source.[Metro_status] = Target.[metro_label]
)
WHEN NOT MATCHED BY Target THEN
INSERT (metro_label)
VALUES(Source.[Metro_status]);
```

Creation of dimension “state_dim”:

This table contains two columns. The first one is “state_id” which is Primary Key, its type is integrate, is unique and set with “Identity Specification” to “Yes”, so it increases automatically. The second one is “state_label” which is varchar type and has been set as index, is unique and it ignores duplicand values in insertion. Neither of the columns allows null values.



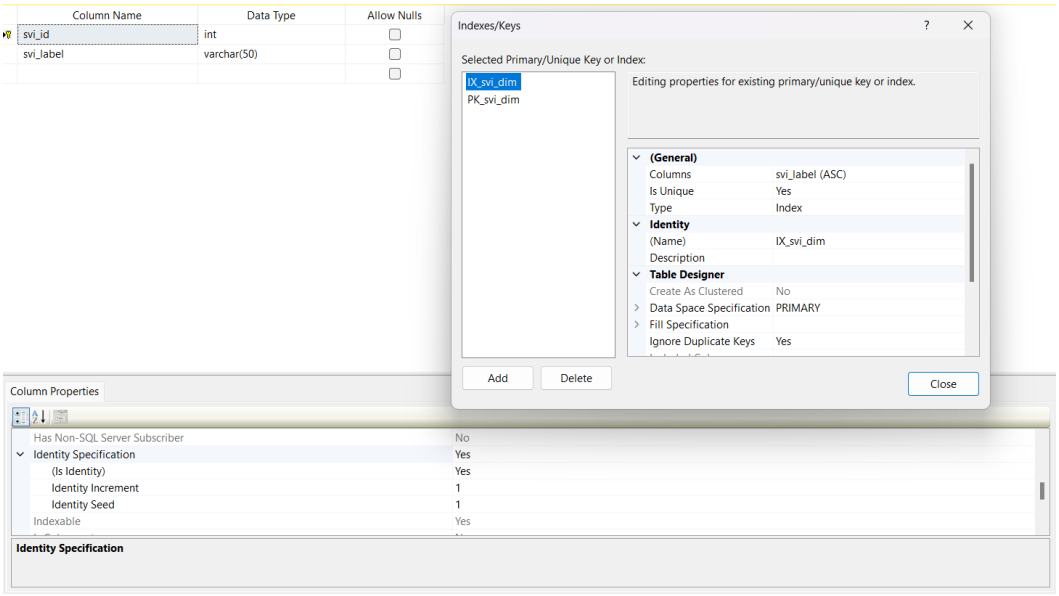
To insert values in “state_dim” we create an SQL code that performs a MERGE operation between two tables: “state_dim” (referred to as Target) and a derived table created by selecting distinct values from the “Recip_State” column in the “view_unpivoting” table (referred to as Source). The merging logic is based on matching conditions defined in the ON clause, linking Source.[Recip_State] to Target.[state_label]. Whenever a match based on this condition isn't found, the code executes an INSERT operation into the Target table. It inserts values from the “Recip_State” column of the Source table into the “state_label” column of the Target table. Essentially, this code snippet updates the Target table by incorporating unique values from the Source table's “Recip_State” column that are absent in the Target table, ensuring that the state-related labels are comprehensive and complete in the context of the dimensional structure.

SQL code:

```
MERGE [covid].[dbo].[state_dim] AS Target
USING(
	SELECT DISTINCT [Recip_State]
	FROM [covid].[dbo].[view_ unpivoting]
) AS Source
ON (
	Source.[Recip_State] = target.[state_label]
)
WHEN NOT MATCHED BY Target THEN
INSERT (state_label)
VALUES (Source.[Recip_State]);
```

Creation of dimension “svi_dim”:

This table contains two columns. The first one is “svi_id” which is Primary Key, its type is integrate, is unique and set with “Identity Specification” to “Yes”, so it increases automatically. The second one is “svi_label” which is varchar type and has been set as index, is unique and it ignores duplicand values in insertion. Neither of the columns allows null values.



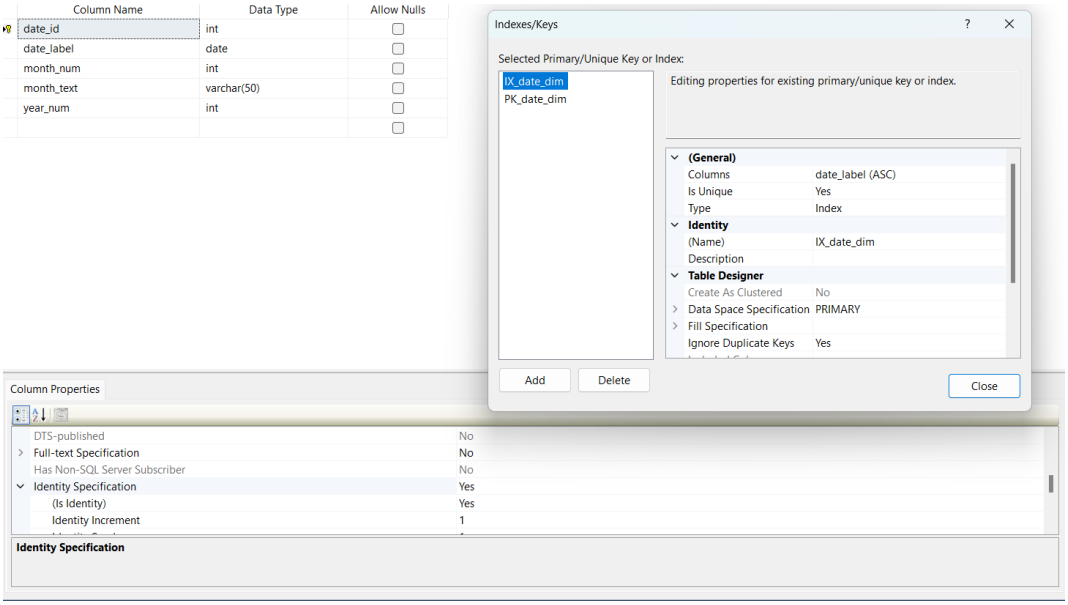
To insert values in “svi_dim” we create an SQL code that involves a MERGE operation between two tables: “svi_dm” (referred to as Target) and a derived table formed by selecting distinct values from the “SVI_CTGY” column in the “view_unpivoting” table (referred to as Source). The merging logic is based on matching conditions specified in the ON clause, connecting Source.[SVI_CTGY] to Target.svi_label. When a match isn't found based on this condition, the code executes an INSERT operation into the Target table. It inserts values from the “SVI_CTGY” column of the Source table into the “svi_label” column of the Target table. This process updates the Target table by incorporating unique values from the Source table's “SVI_CTGY” column that are not present in the Target table, ensuring that the Social Vulnerability Index (SVI) categories or labels are comprehensive and inclusive within the dimensional structure.

SQL code:

```
MERGE [covid].[dbo].[svi_dim] AS Target
USING(
        SELECT DISTINCT [SVI_CTGY]
        FROM [covid].[dbo].[view_ unpivoting]
) AS Source
ON (
        Source.[SVI_CTGY] = target.svi_label
)
WHEN NOT MATCHED BY Target THEN
INSERT (svi_label)
VALUES (Source.[SVI_CTGY]);
```

Creation of dimension “date_dim”:

This table contains five columns. The first one is “date_id” which is Primary Key, its type is integrate, is unique and set with “Identity Specification” to “Yes”, so it increases automatically. The second one is “date_label” which is date type and has been set as index, is unique and it ignores duplicand values in insertion. The other columns are “month_num”, “month_text” and “year_num” which are applied to the number of month in the year, the name of each month and the number of year, respectively. None of the columns allows null values.



To insert values in “date_dim” we create an SQL code that employs a MERGE statement to synchronize the “date_dim” table (Target) with a derived table created by selecting distinct values from the “Date” column in the “view_unpivoting” table (Source). The merging operation is based on the matching conditions defined in the ON clause, associating Source.[Date] with Target.[date_label]. When there's no match based on this condition, the code executes an INSERT operation into the Target table.

The INSERT statement adds records to the Target table by extracting values from the “Date” column of the Source table. Additionally, it extracts the month and year information from the Date column using functions like “MONTH()”, “DATENAME()”, and “YEAR()” functions, populating the corresponding columns in the Target table (month_num, month_text, year_num). This process ensures that the date_dim table remains comprehensive and includes all unique dates while incorporating associated month and year attributes for analytical purposes within the data warehouse structure.

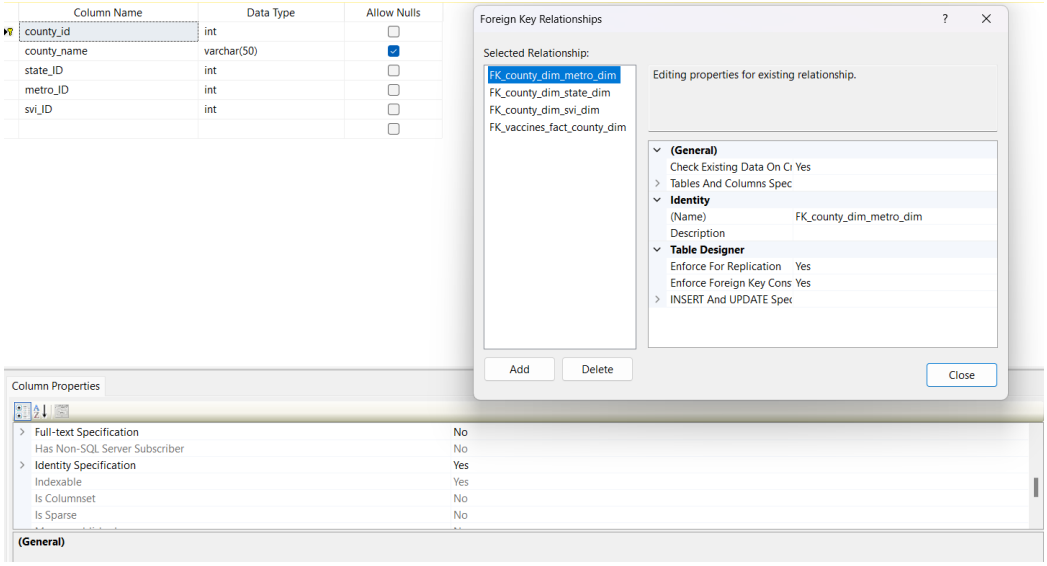
SQL code:

```
MERGE [covid].[dbo].[date_dim] AS Target
USING (
   SELECT DISTINCT [Date]
   FROM [covid].[dbo].[view_ unpivoting]
```

```
) AS Source
ON (
   Source.[Date] = Target.[date_label]
)
WHEN NOT MATCHED BY Target THEN
   INSERT (date_label, month_num, month_text, year_num)
   VALUES (
      Source.[Date],
      MONTH(Source.[Date]),
      DATENAME(month, Source.[Date]),
      YEAR(Source.[Date])
   );
```

Creation of dimension "county_dim":

This table contains five columns. The first one is "county_id" which is Primary Key, its type is integrate, is unique and set with "Identity Specification" to "Yes", so it increases automatically. The second one is "county_name" which is varchar type and the other columns are "state_id", "metro_id" and "svi_id" which are Foreign Keys of tables "state_dim", "metro_dim" and "svi_dim" respectively. None of the columns allows null values, except "county_name".



To insert values in "county_dim" we create an SQL code that utilizes a MERGE statement to synchronize the "county_dim" table (Target) with a derived table (Source) generated by performing an inner join on tables "state_dim", "metro_dim", "svi_dim", and "view_unpivoting". The joined tables are aggregated, grouping the data by "Recip_County", "state_id", "svi_id", and "metro_id".

The merging operation is predicated on multiple matching conditions outlined in the ON clause, linking columns from the Source table to corresponding columns in the Target table

(state_id, metro_id, svi_id, and county_name). Whenever there's no match based on these conditions, the code executes an INSERT operation into the Target table.

The INSERT statement appends records to the Target table by extracting values from the Source table's aggregated result. It populates columns “county_name”, “state_id”, “svi_id”, and “metro_id” in the Target table with the corresponding values from the Source table. This operation ensures that the “county_dim” table remains updated and aligned with the associated state, metro, and svi dimensions, preserving comprehensive and accurate county-level information within the data warehouse structure.

One of the greatest challenges we faced was inserting values into the “county_name” column. This is due to the fact that this specific column cannot be set to get unique values whlie some county values are dublicate. The result of this not getting uniqu values was the rows of the fact table to spread over an extremely wide field of trillions. To face this problem we use the “group by” function nested to “merge” function. We also used special handling on the table “county_dim”, when we linked it to the fact table as you will find out later.

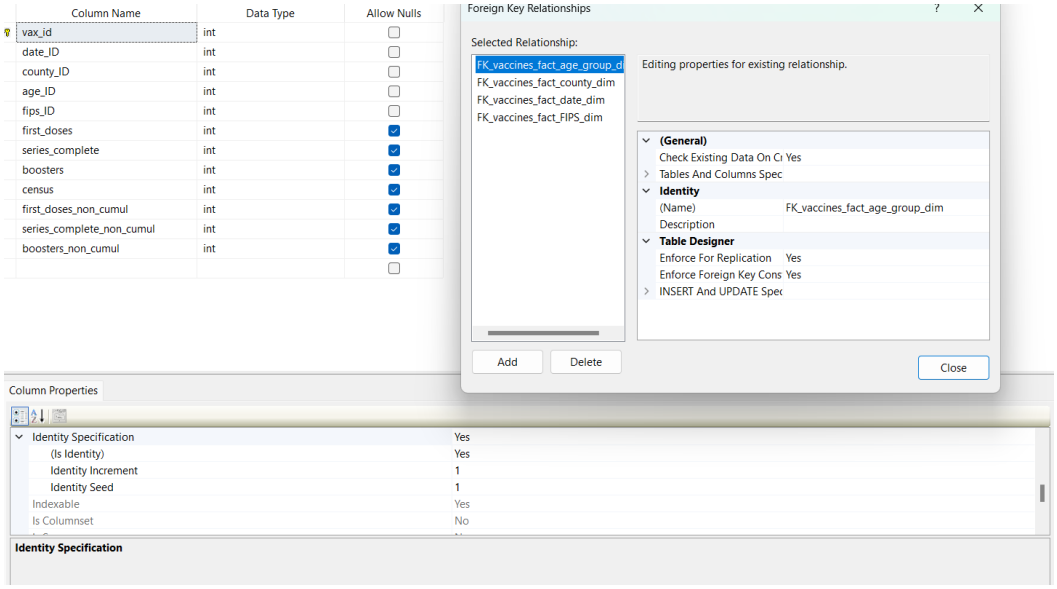
SQL code:

```
MERGE [covid].[dbo].[county_dim] AS Target
USING (
  SELECT
    [state_dim].[state_id],
    [svi_dim].[svi_id],
    [metro_dim].[metro_id],
    [view_ unpivoting].[Recip_County] as [county_name]
  FROM [covid].[dbo].[ view_ unpivoting]
  INNER JOIN [covid].[dbo].[ state_dim] ON ([covid].[dbo].[view_ unpivoting].[Recip_State] =
[covid].[dbo].[state_dim].state_label)
  INNER JOIN [covid].[dbo].[ metro_dim] ON ([covid].[dbo].[ view_
unpivoting].[Metro_status] = [covid].[dbo].[metro_dim].metro_label)
  INNER JOIN [covid].[dbo].[svi_dim] ON ([covid].[dbo].[view_ unpivoting].[SVI_CTGY] =
[covid].[dbo].[ svi_dim].svi_label)
        GROUP BY
              [view_ unpivoting].[Recip_County],
              [state_dim].[state_id],
              [svi_dim].[svi_id],
              [metro_dim].[metro_id]
) AS Source
ON (
  Source.[state_id] = Target.[state_id]
  AND Source.[metro_id] = Target.[metro_id]
  AND Source.[svi_id] = Target.[svi_id]
  AND Source.[county_name] = Target.[county_name]
)
```

```
WHEN NOT MATCHED BY Target THEN
   INSERT ([county_name], [state_id], [svi_id], [metro_id])
   VALUES (Source.[county_name], Source.[state_id], Source.[svi_id], Source.[metro_id]);
```

Creation of fact table "vaccines_fact":

This table contains twelve columns. The first one is "vax_id" which is Primary Key, its type is integrate, is unique and set with "Identity Specification" to "Yes", so it increases automatically. The next four columns, "date_ID", "county_ID", "age_ID", and "fips_ID" are are integrate type and are Foreign Keys of tables "date_dim", "county_dim", "age_dim" and "fips_dim" respectively. The rest of the columns refer to vaccination, the first three contain cumulative values and the rest have been calculated as with non-cumulative values. None of the columns allows null values, except "county_name".
Referring to the overcount challenge with the table "county_dim", in the insertion of values to this table we have used logical operator "and" in "inner join" function between the "county_dim" table and the view_unpivoting view to ensure that counties' values will be aligned to the id's of the state's, svi's and metro's values.

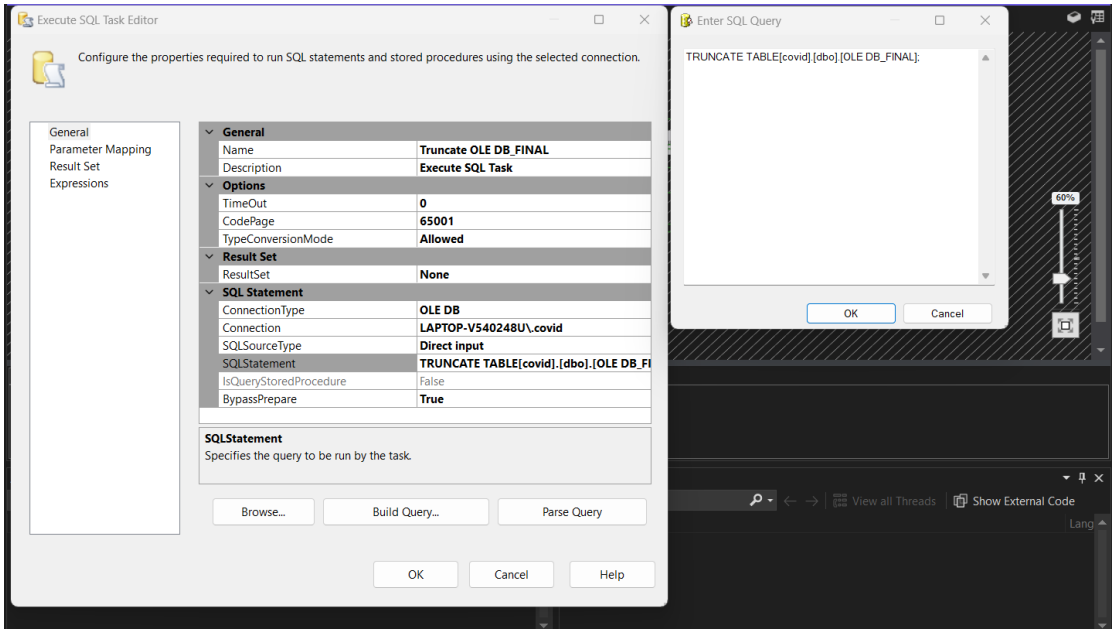


SQL code:

```
INSERT INTO [Covid].[dbo].[vaccines_fact] (
   date_ID, county_ID, age_ID, fips_ID,
   first_doses, series_complete, boosters, census,
   first_doses_non_cumul, series_complete_non_cumul, boosters_non_cumul
)
SELECT
   dd.[date_id] AS 'Date ID',
   cd.county_id AS 'County ID',
   agd.[age_id] AS 'Age ID',
```

```
   fd.FIPS_id AS 'FIPS ID',
   vu.Administered AS 'First Doses',
   vu.Series AS 'Series Complete',
   vu.Booster AS 'Boosters',
   COALESCE(vu.Census, MAX(vu.Census) OVER (PARTITION BY vu.Recip_County)) AS
'Population',
   COALESCE(vu.Administered - LAG(vu.Administered) OVER (PARTITION BY cd.county_id,
agd.age_id ORDER BY vu.[Date]), vu.Administered) AS 'First Doses Non-Cumul',
   COALESCE(vu.Series - LAG(vu.Series) OVER (PARTITION BY cd.county_id, agd.age_id
ORDER BY vu.[Date]), vu.Series) AS 'Series Complete Non-Cumul',
   COALESCE(vu.Booster - LAG(vu.Booster) OVER (PARTITION BY cd.county_id, agd.age_id
ORDER BY vu.[Date]), vu.Booster) AS 'Boosters Non-Cumul'
FROM
   view_unpivoting AS vu
INNER JOIN [Covid].[dbo].[date_dim] AS dd ON vu.[Date] = dd.date_label
INNER JOIN [Covid].[dbo].[age_group_dim] AS agd ON vu.AgeGroup = agd.age_group
INNER JOIN [Covid].[dbo].[FIPS_dim] AS fd ON vu.FIPS = fd.FIPS_label
INNER JOIN [Covid].[dbo].[state_dim] AS sd ON vu.Recip_State = sd.state_label
INNER JOIN [Covid].[dbo].svi_dim AS sv ON vu.SVI_CTGY = sv.svi_label
INNER JOIN [Covid].[dbo].metro_dim AS md ON vu.Metro_status = md.metro_label
INNER JOIN [Covid].[dbo].[county_dim] AS cd ON (vu.Recip_County = cd.county_name
AND cd.state_ID = sd.state_id AND cd.svi_ID = sv.svi_id AND cd.metro_ID = md.metro_id);
```

Now, all tables are ready, so we can continue by setting up processes to update them in Visual Studio. The first step is to create an SQL Task in tab “Control Flow”, which we are going to name “Truncate OLE DB_FINAL”. This task is going to truncate table “OLE DB_FINAL” in order to be empty every time we insert data from the updated csv. Then, we connect this task with the “Import csv” task we created in the beginning, setting the order of tasks to start from “Truncate OLE DB_FINAL”.



In the same way, we are going to create the update tasks for all the tables we created, from the dimensions to the fact table.In all the Update tasks we set the name, according to the name and the content of each table and set CodePage to 65001, as in the begging when we created the task “Import csv”. We set the connection to the server we use and in the end we put in the SQLStatement the query which inserts values into the tables.

Table “state_dim”:

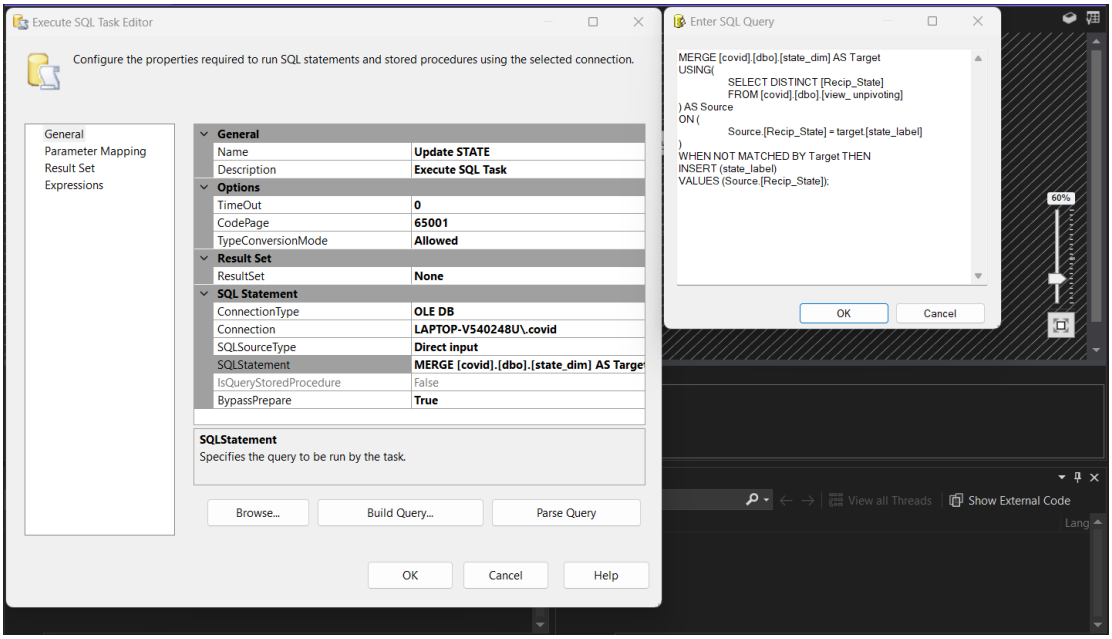


Table “svi_dim”:

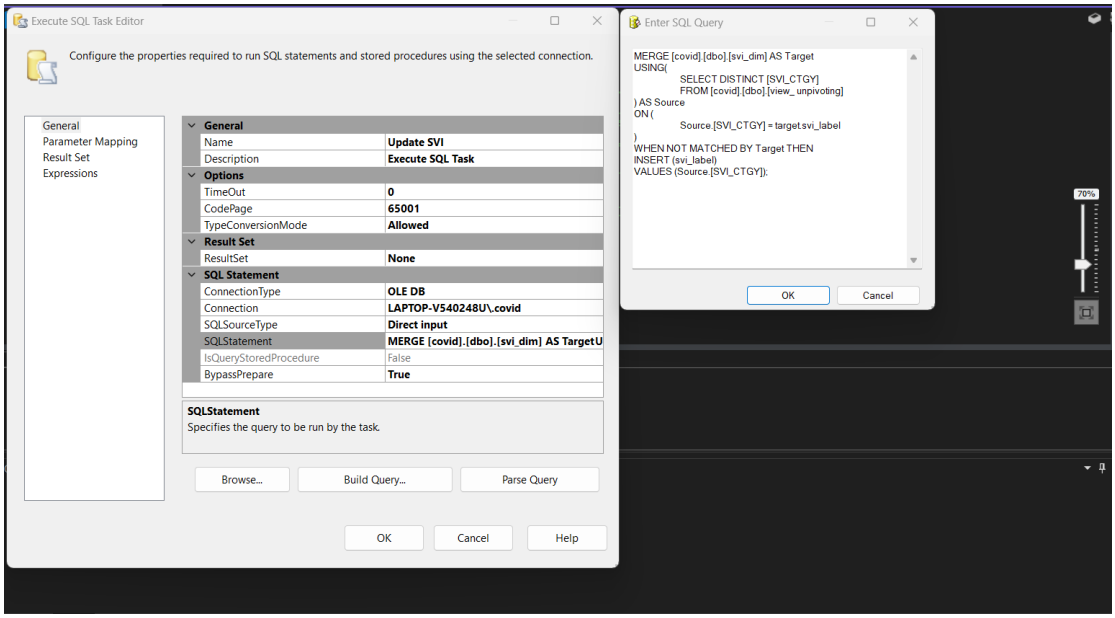


Table “metro_dim”:

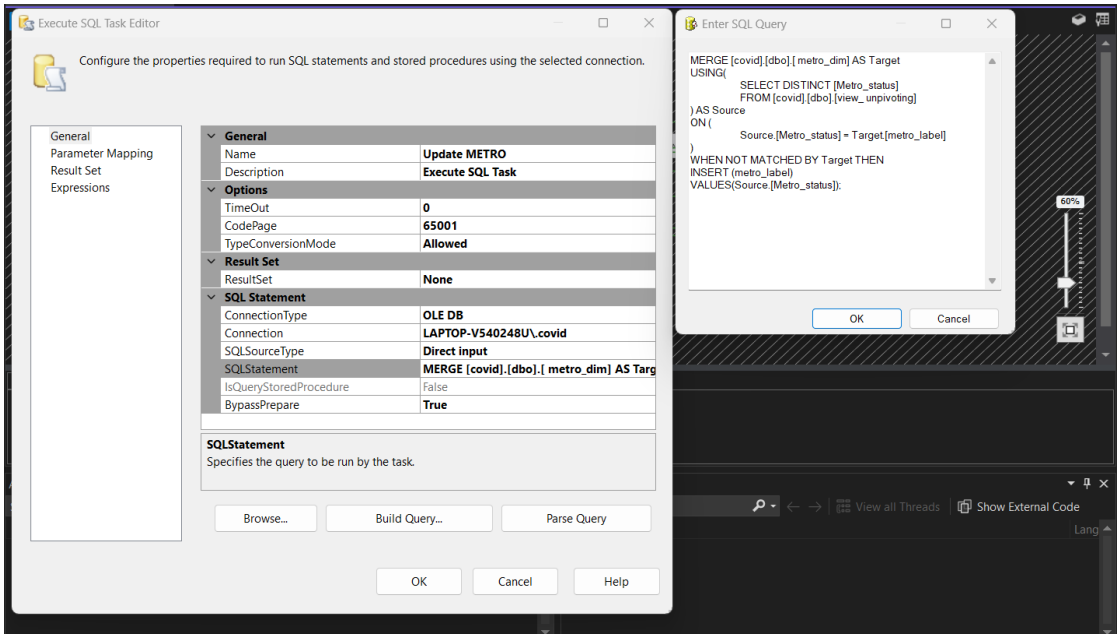


Table “age_dim”:

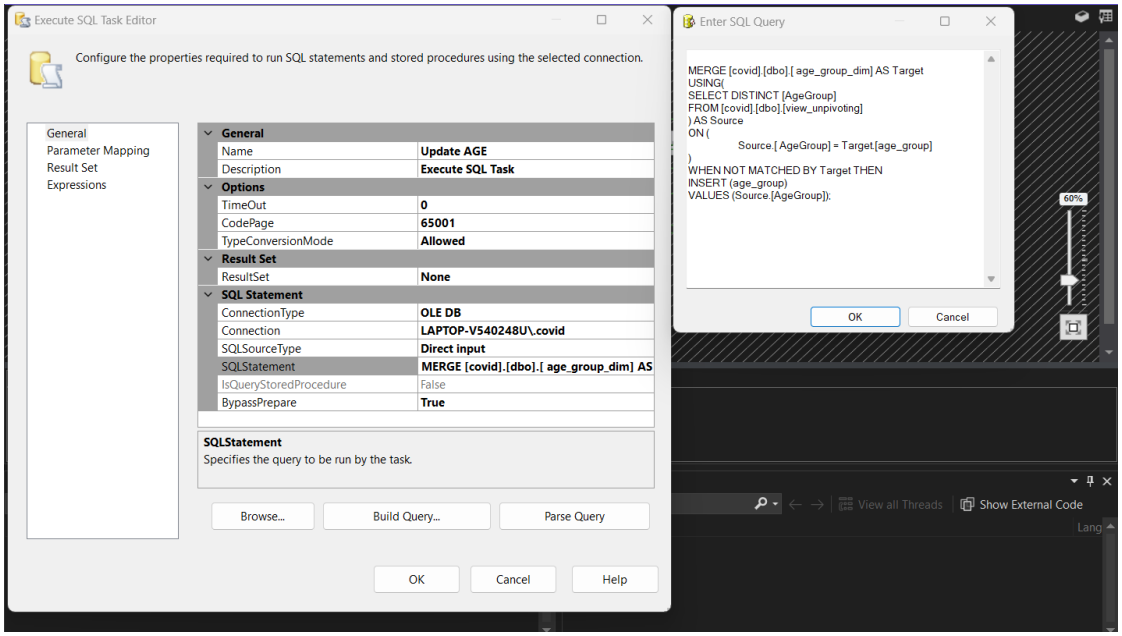


Table “date_dim”:

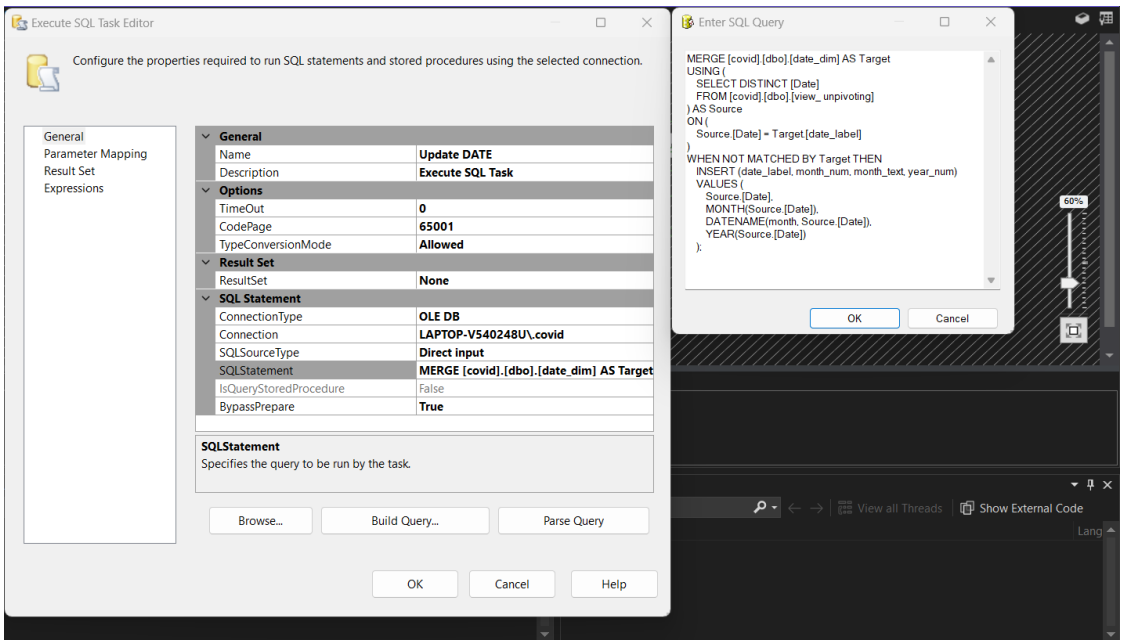


Table “fips_dim”:

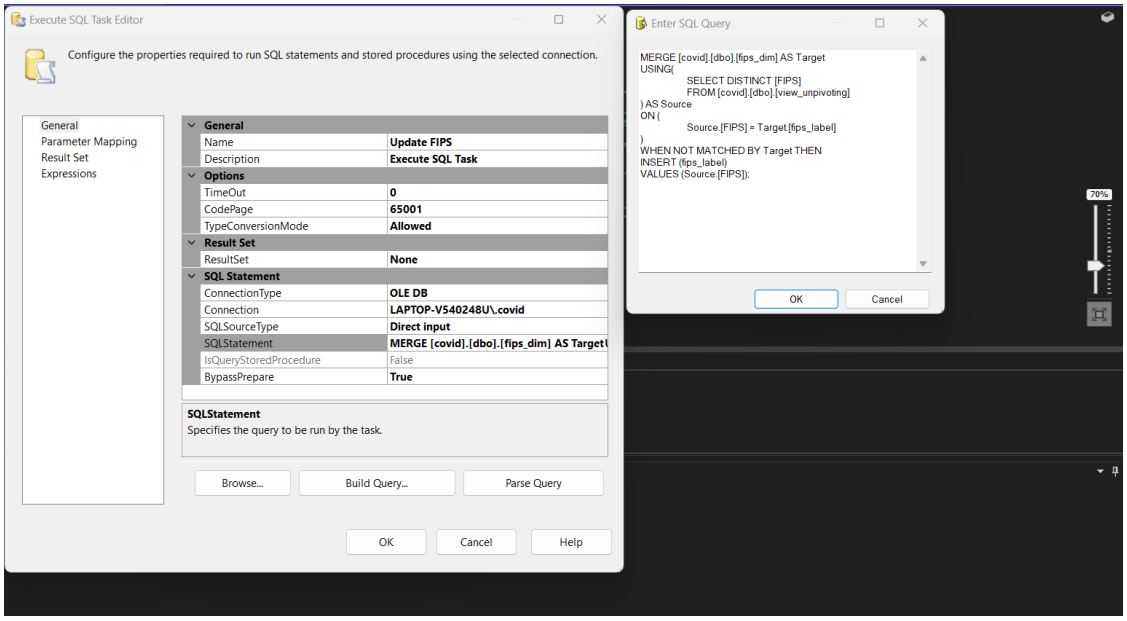


Table “county_dim”:

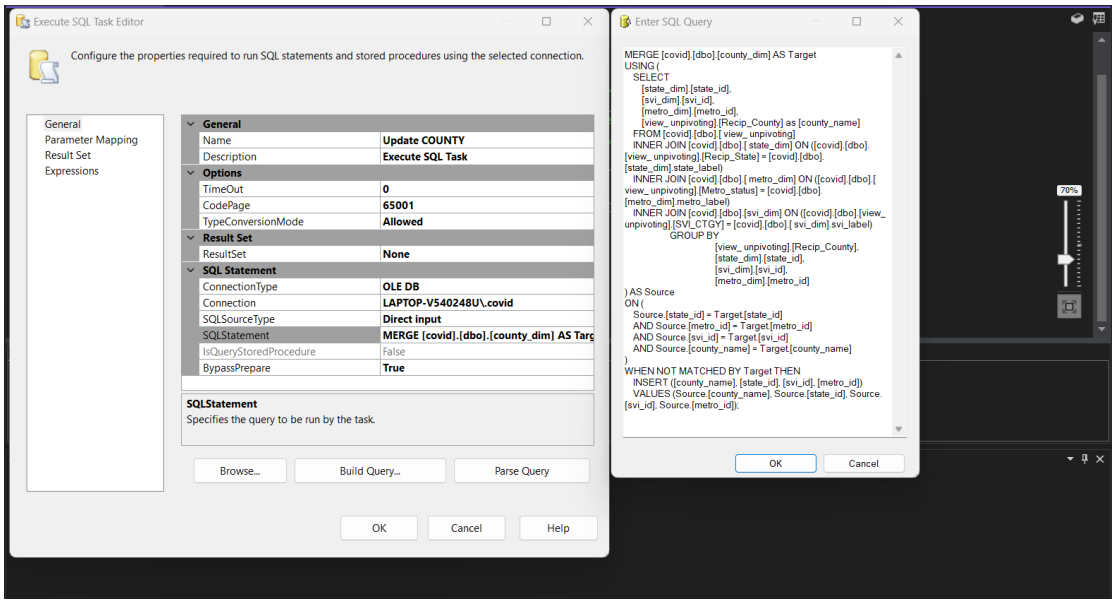
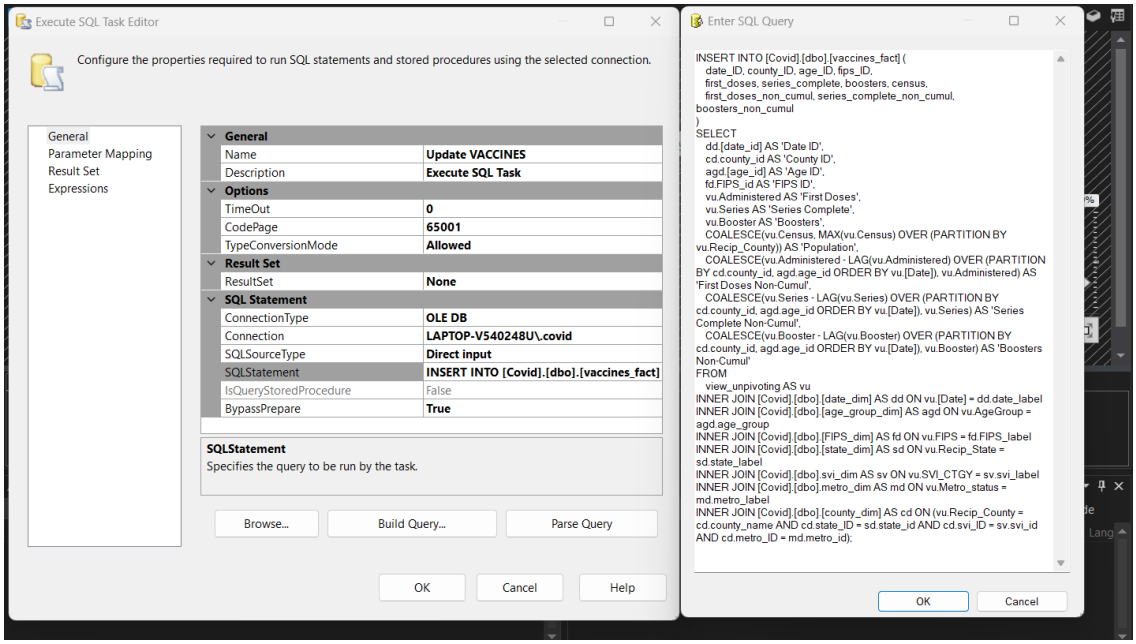
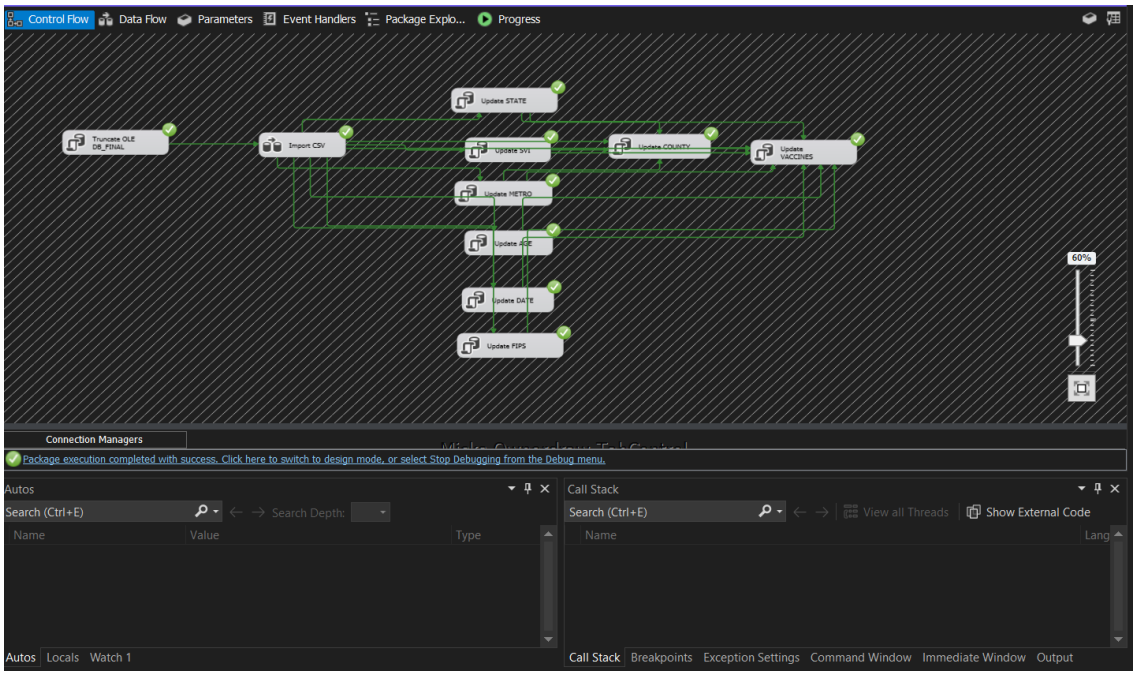


Table “vaccines_fact”:



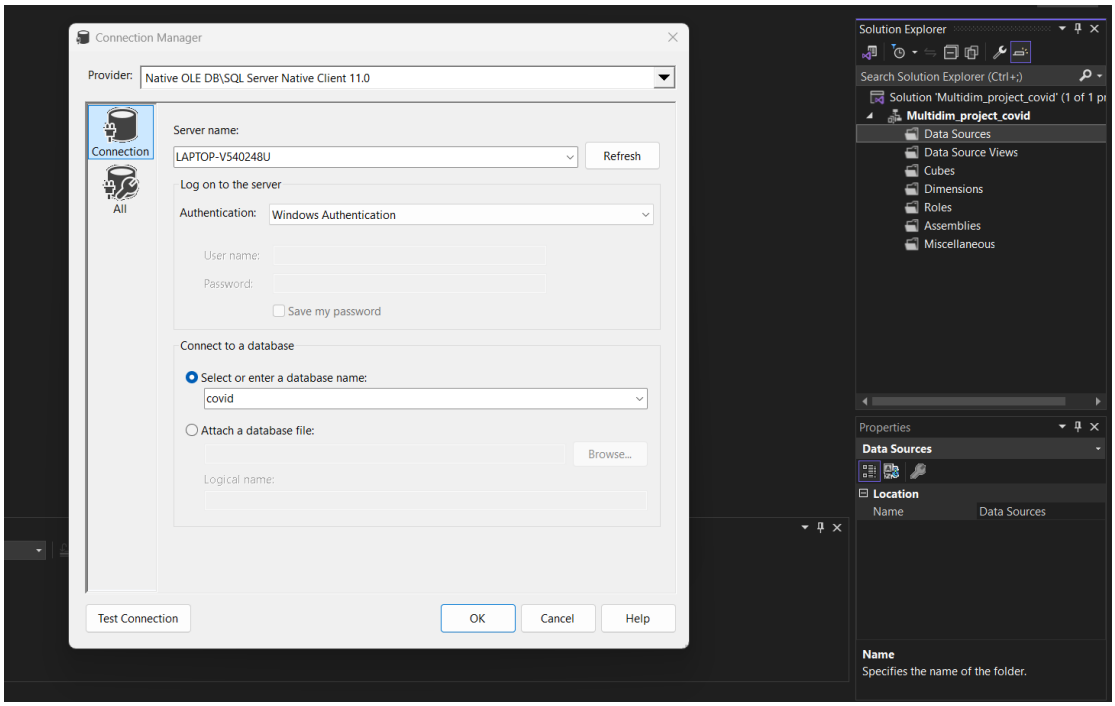
Completing the ETL part, we connect all the update tasks to the task “Import csv”, so that they will be executed after the data importation. Accodrting to the row of the update execution, we put first all dimension tables which do not contain any foreign keys, then the dimension table “county_dim”, which contains foreign keys and then the fact table which contains foreign keys from every other table. We start the tasks execution and we see that everythng has been executed without any problem.



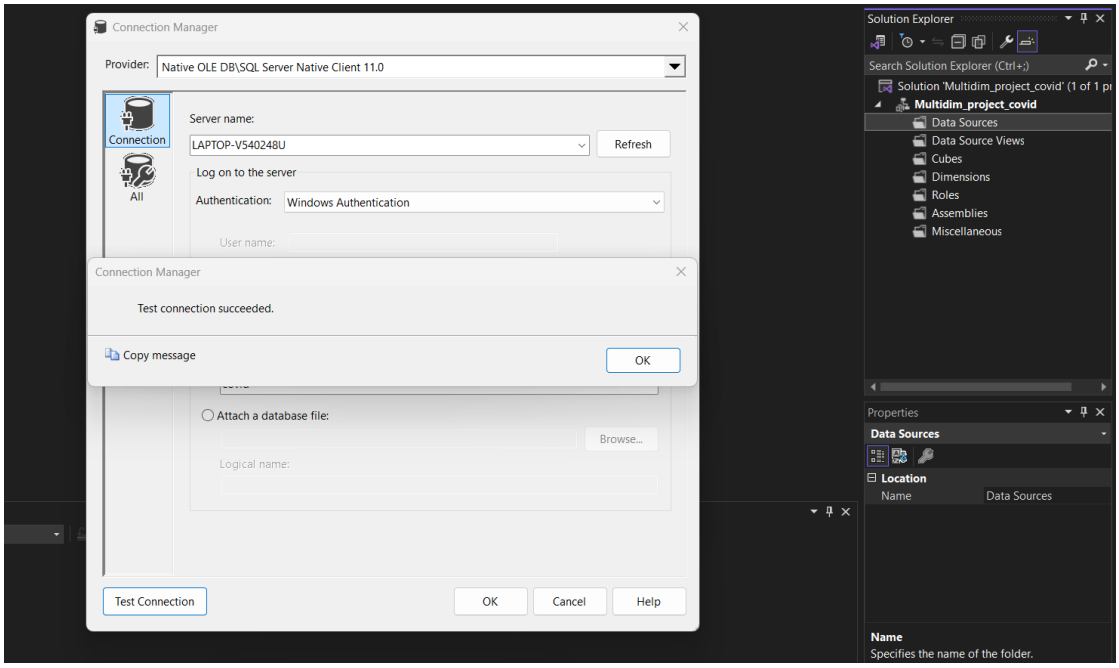
PART B: OLAP (CUBES)

In order to make a cube with the dimensions of the fact table “vaccines_fact” we create a multidimensional project in Visual Studio. Below we will describe the steps we followed until we finalize the cube.

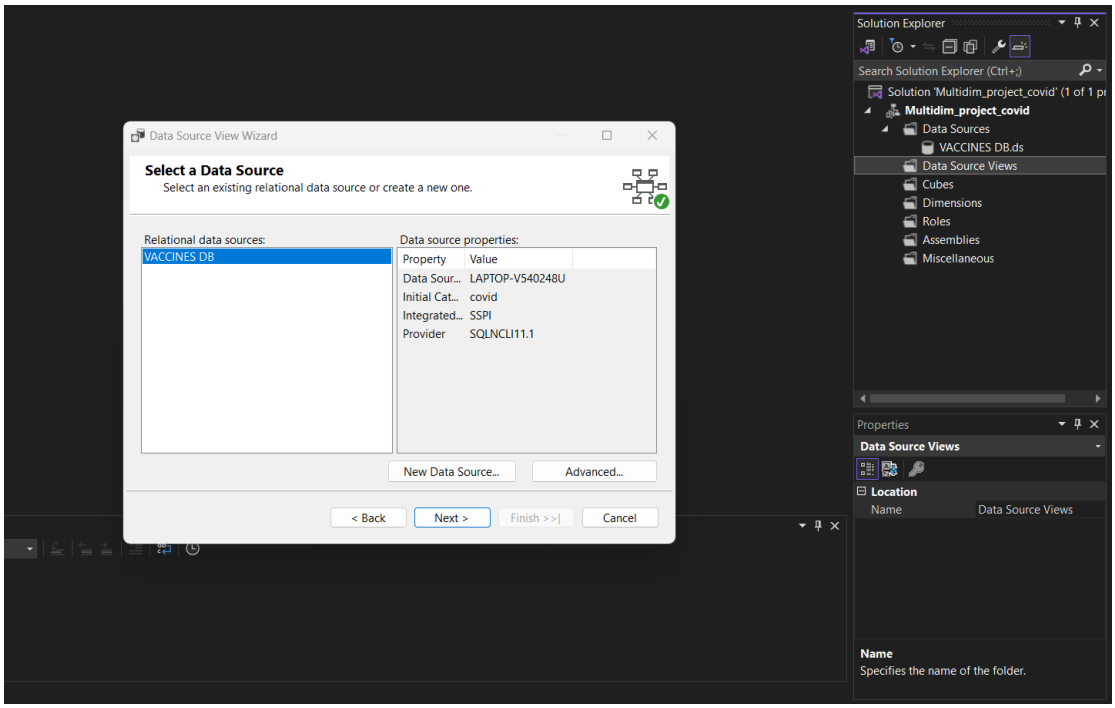
In the folder “Data Source” we create a new data source in which we connect to our database so we can reach the vaccination data.



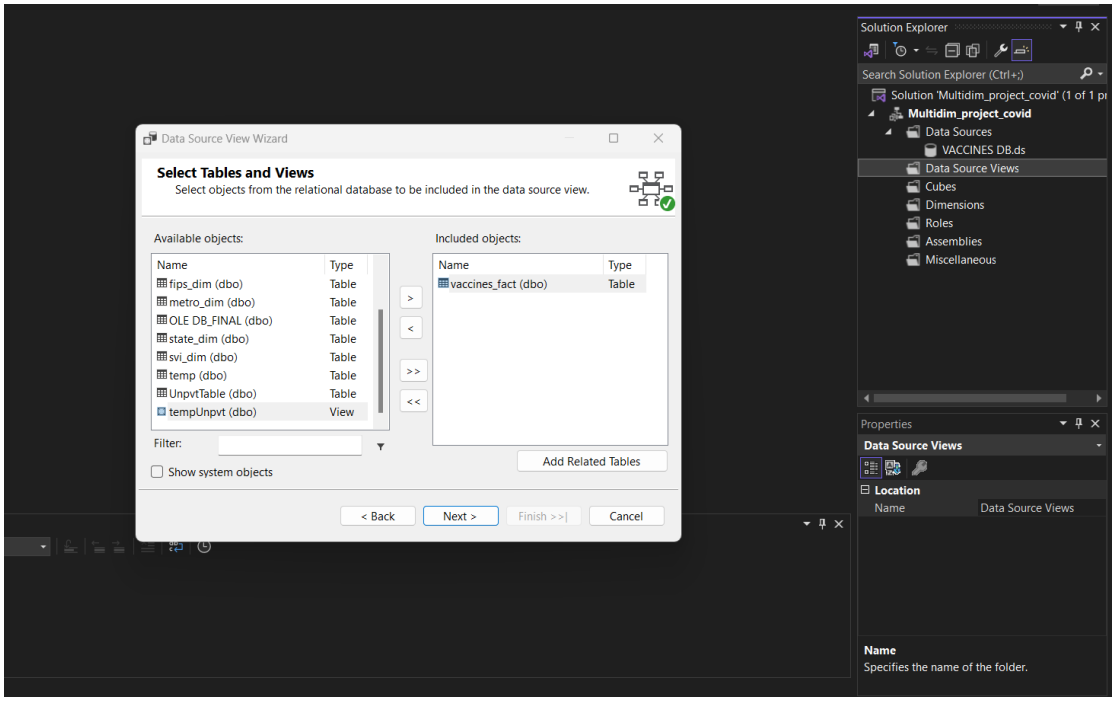
Then we test the connection to affirm it has succeed.



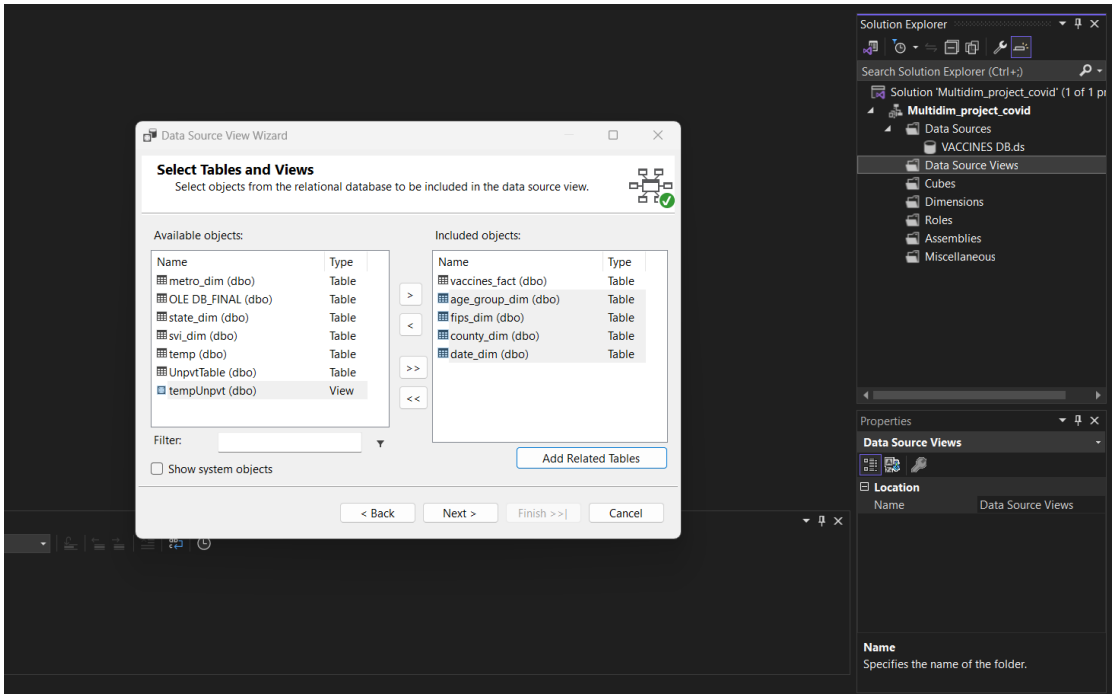
We use the Service connect with our credentials and name our data source “Vaccines DB”. Moving on to the “Data Source Views” folder, we create a new data source view by selecting the data source we just made.



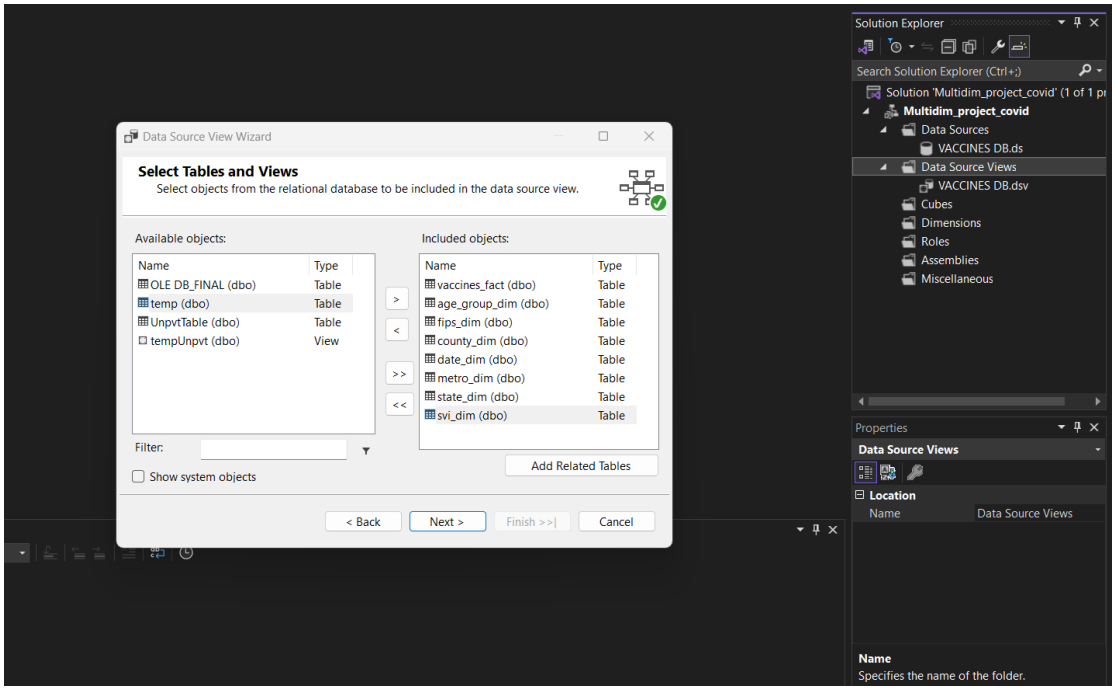
We select the fact table to be included in the view.



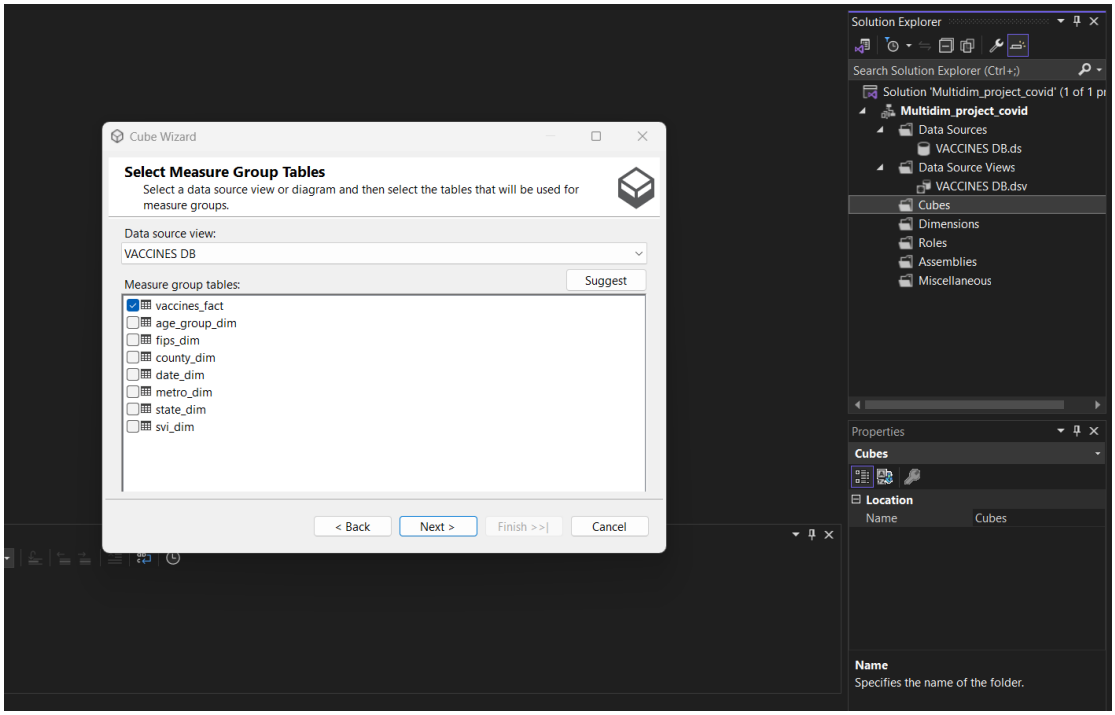
Then, by selecting “Add Related Tables” the dimension tables connected to the fact table are selected automatically.



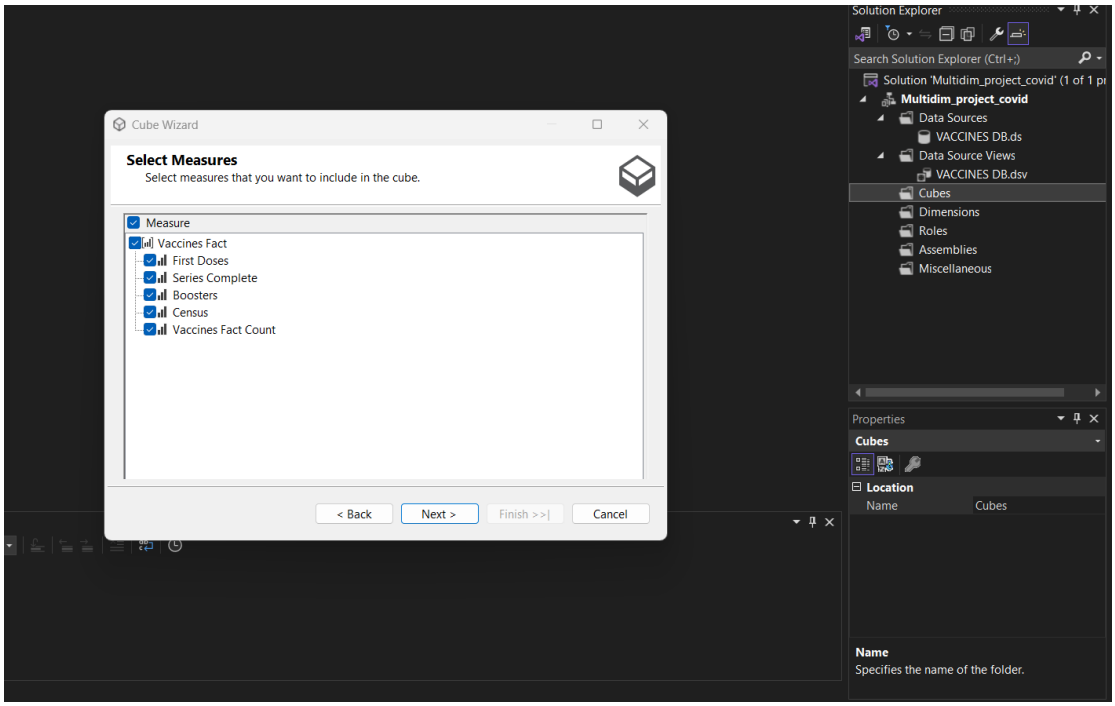
Continuing, we also select the tables that are not connected directly to the fact table, but are connected to table “county_dim” which is a fact table’s dimension.

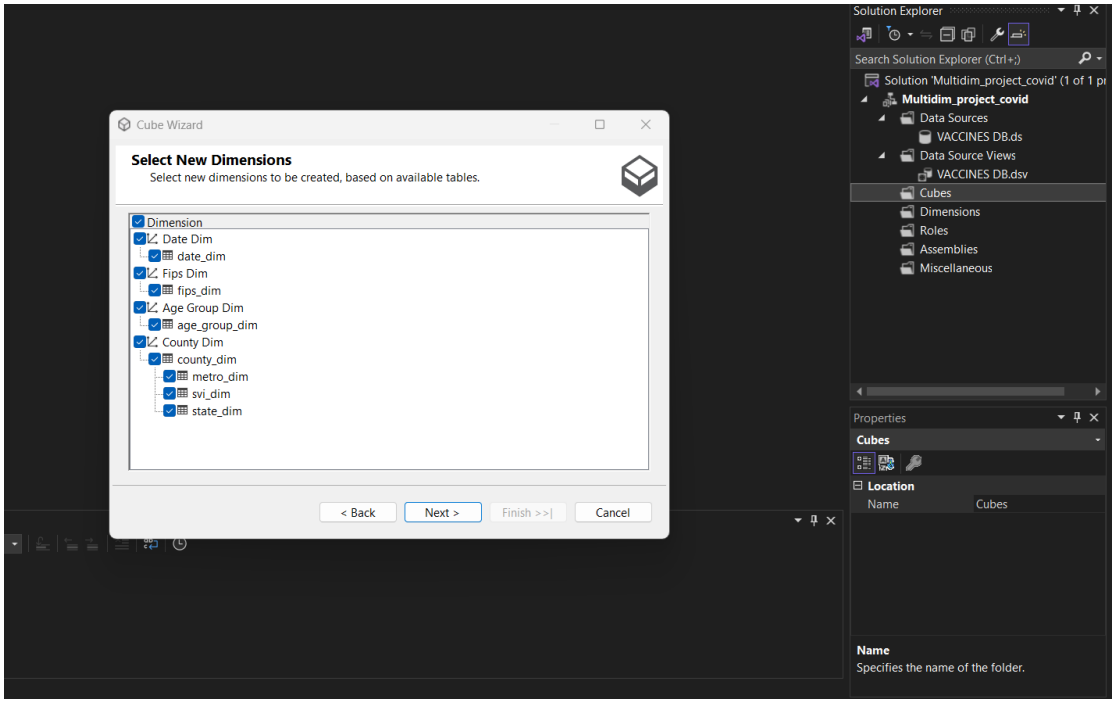


Now, we are ready to go on to the cube’s creation. In the “Cube” folder we select “New Cube” and then “Use existing Table”. Ater that, we select the fact table, which is “vaccines_fact”.

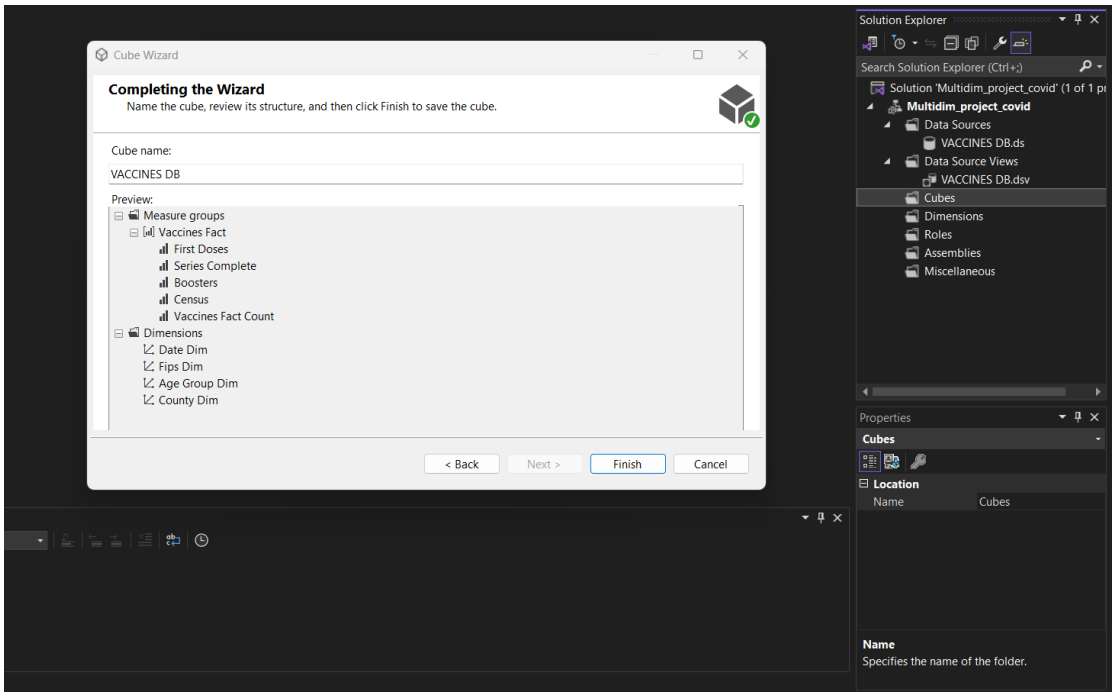


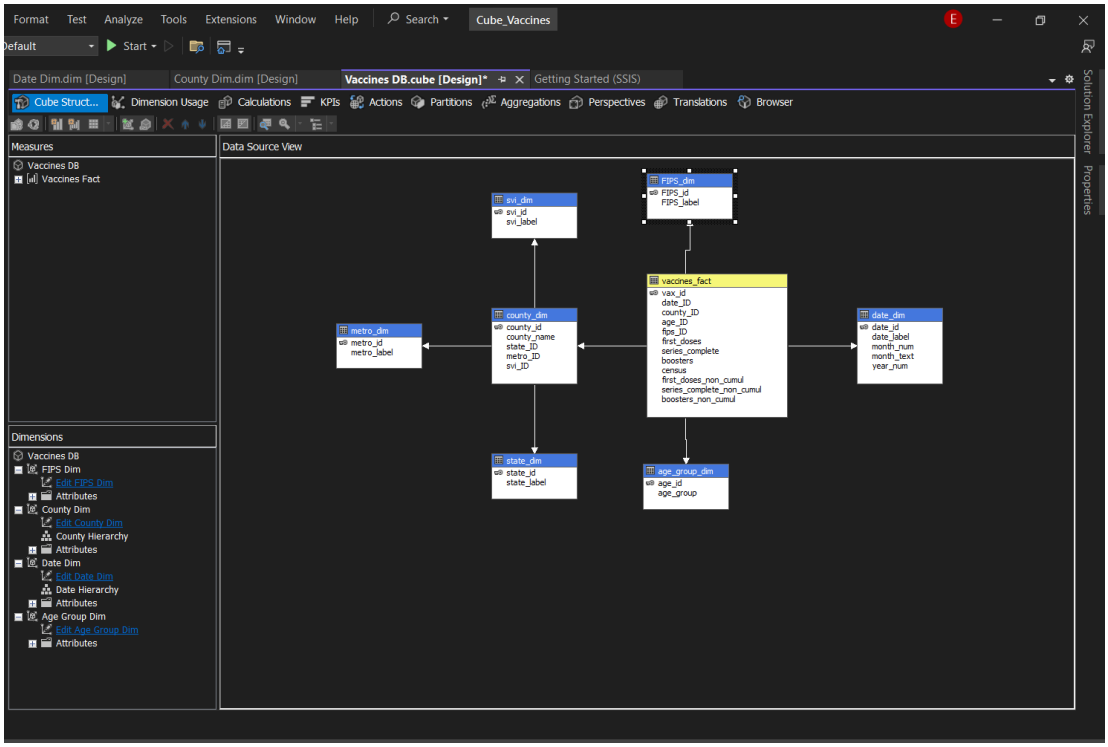
Then, we also select its dimensions.



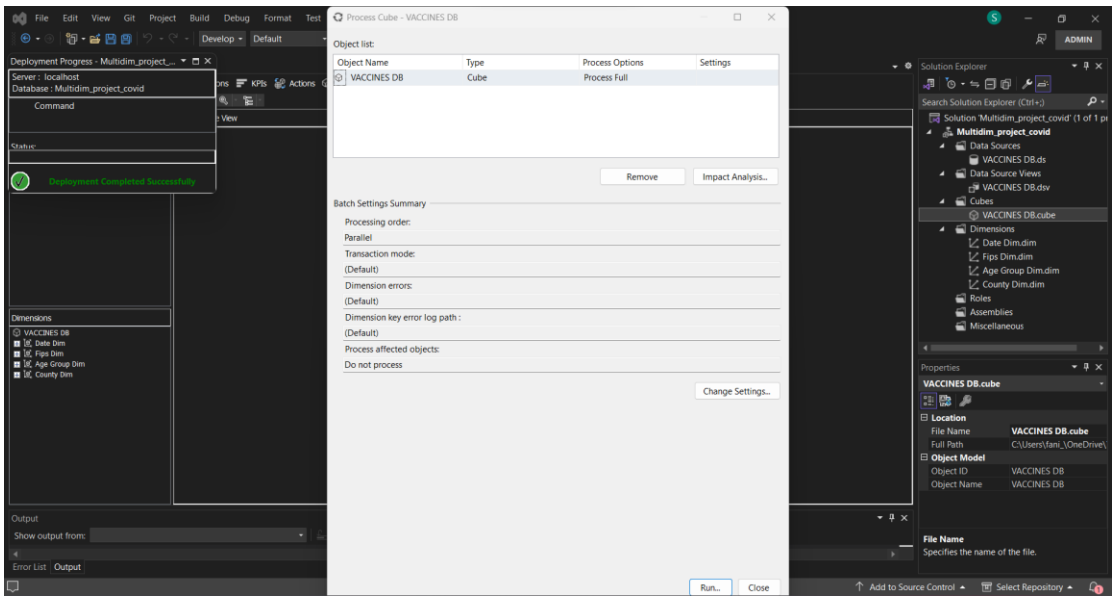


We affirm that the measures and the dimensions have been set correct and we finish the cube settings.

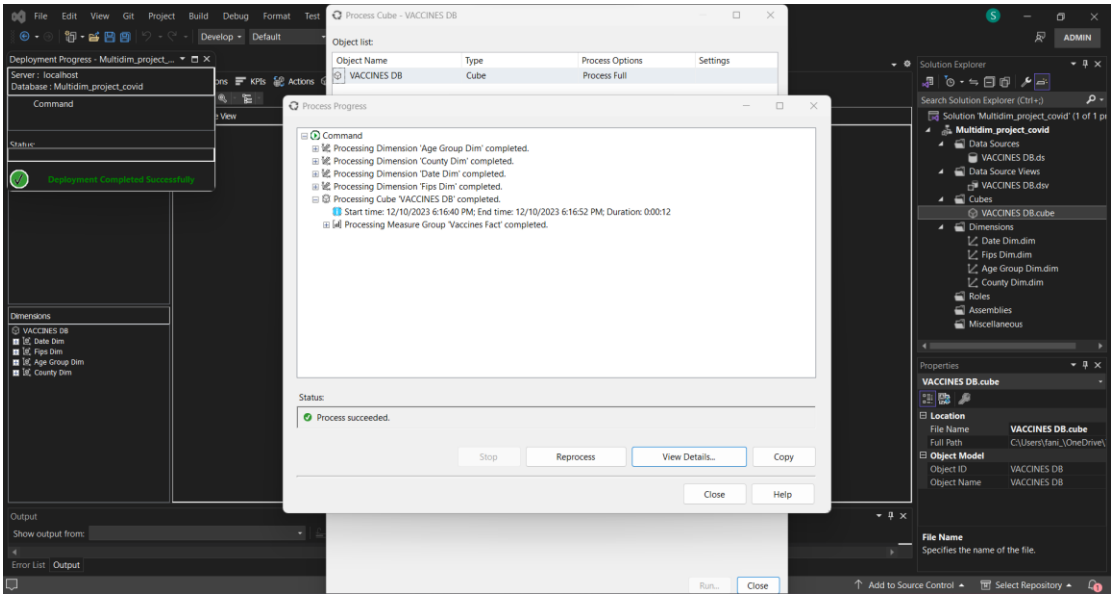




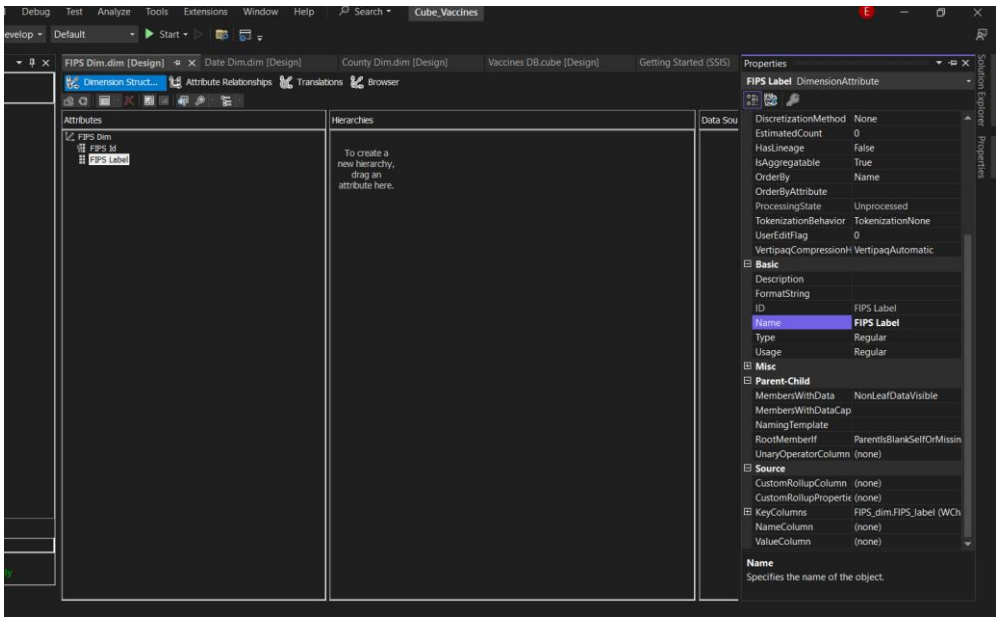
It is now time to process the cube and we see the deployment has succeeded.

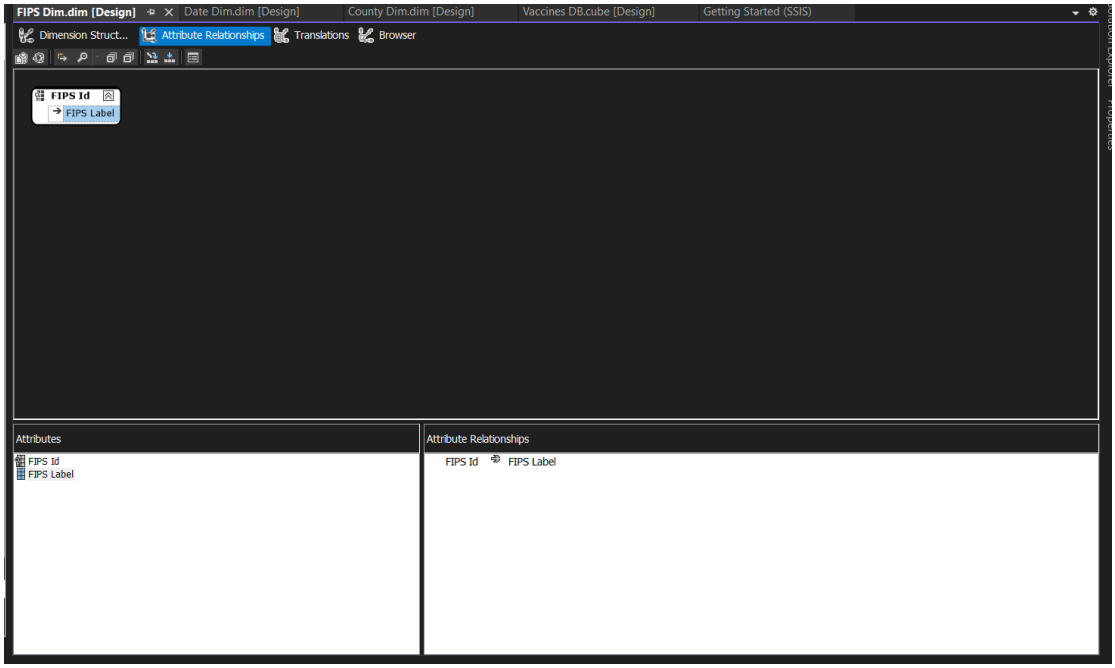


We select “Run” and we receive the message that the cube has been processed.

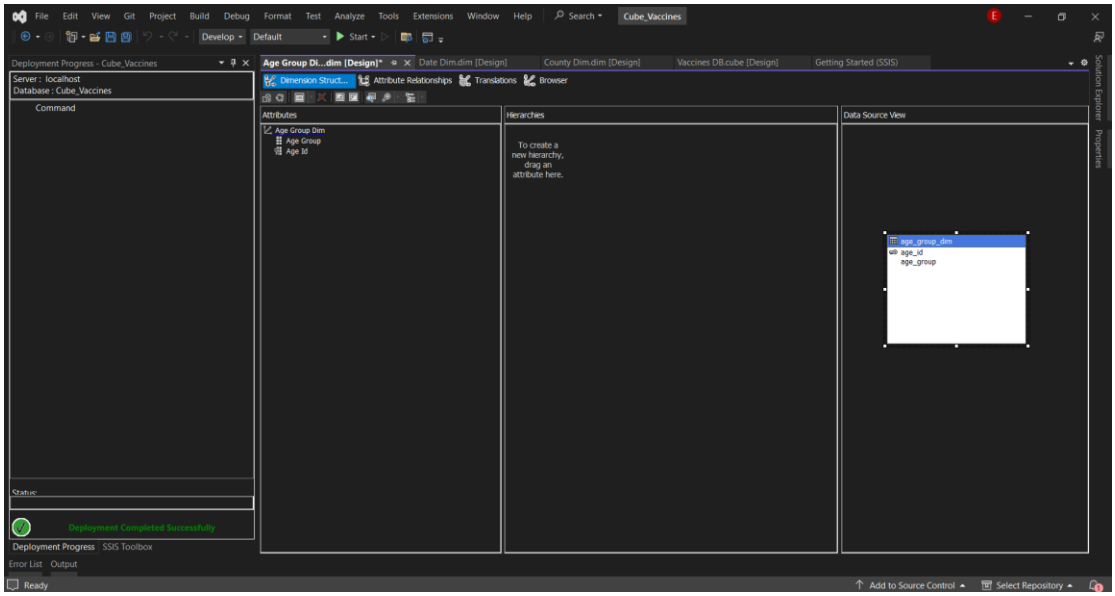


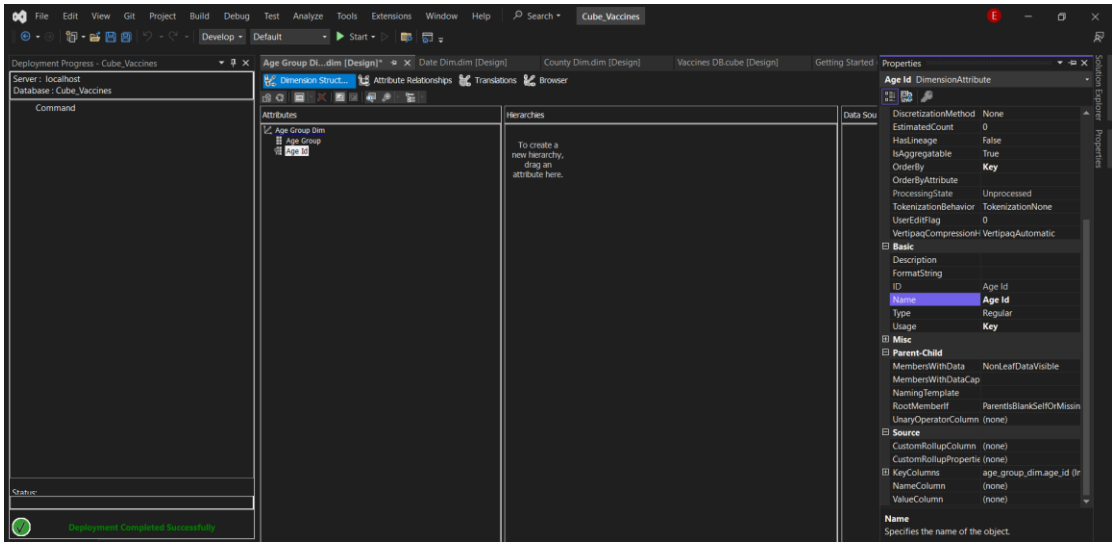
Then we proceed with the editing of the dimensions of our cube. Firstly, we edited the FIPS_dim by adding the FIPS_label in the attributes of the dimension.

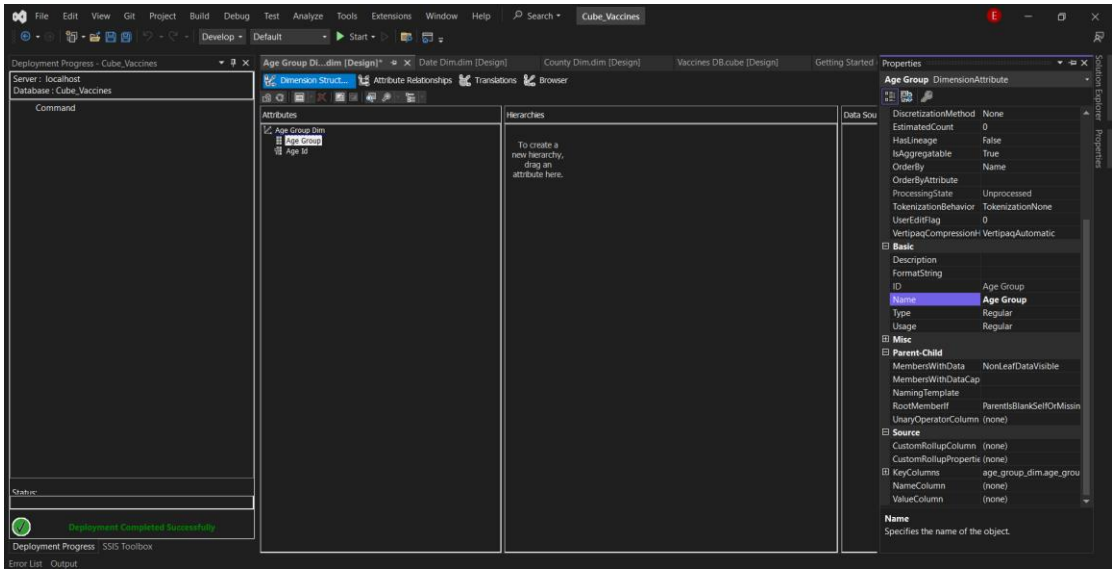


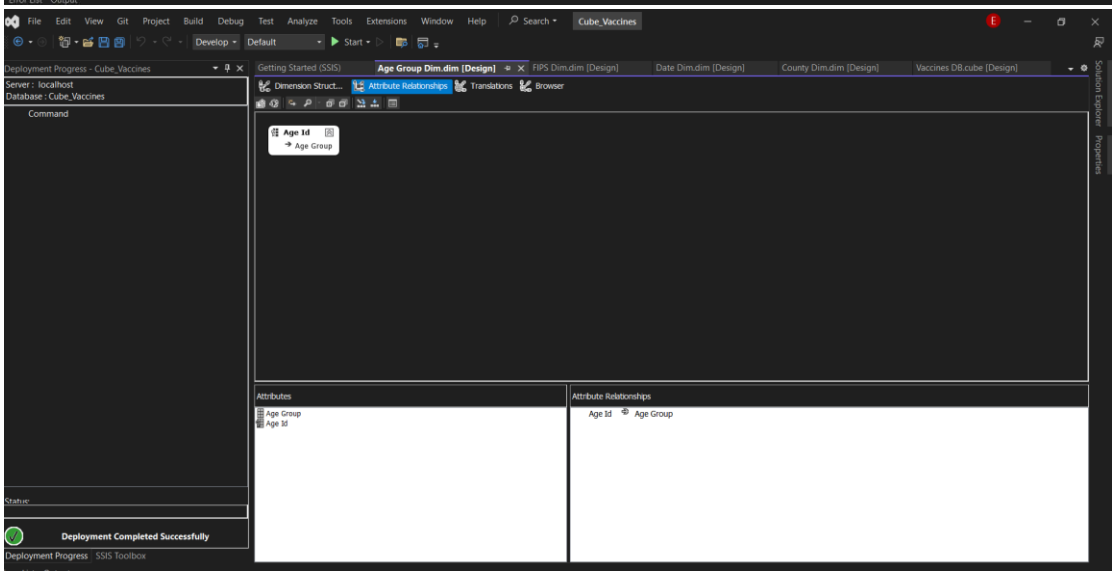


Then we did the same for the label of age_group_dim (in order to be able to show the specific age group that we want to refer to in the Power BI visualizations and not just their IDs).

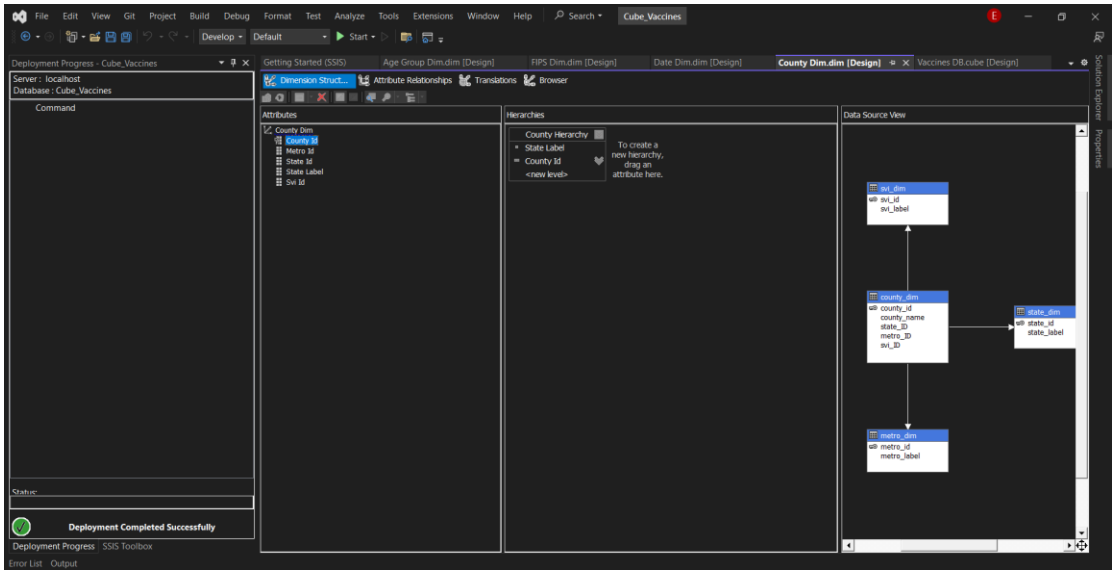


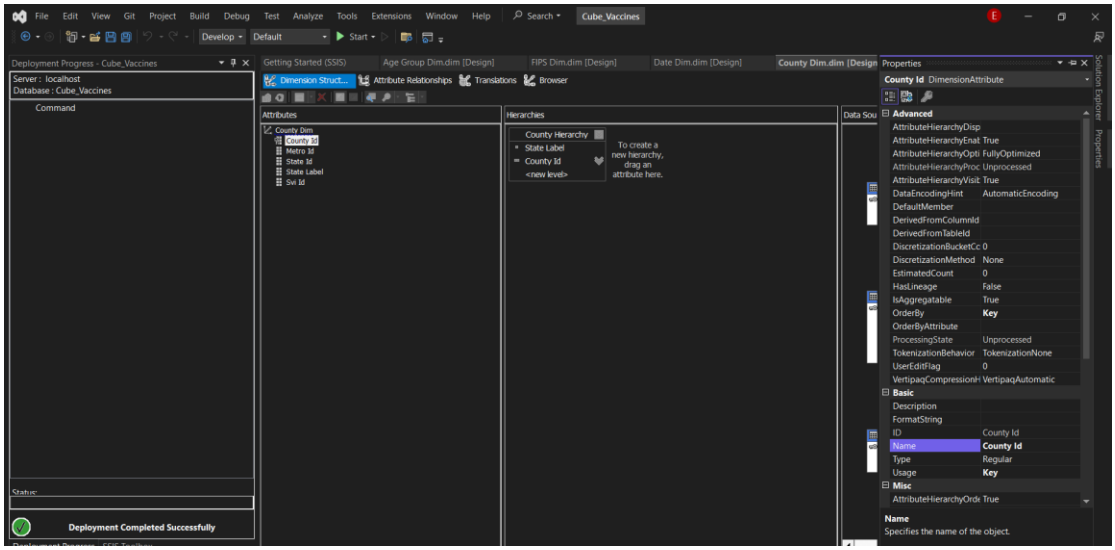

Deployment Progress - Cube_Vaccines
Server : localhost
Database : Cube_Vaccines
Command
Status:
Deployment Completed Successfully
Age Group Di...dim [Design]*
Date Dim.dim [Design]
County Dim.dim [Design]
Vaccines DB.cube [Design]
Getting Started
Dimension Struct...
Attribute Relationships
Translations
Browser
Attributes
Age Group Dim
Age Group
Age Id
Hierarchies
To create a new hierarchy, drag an attribute here.
Properties
Age Id DimensionAttribute
DiscretizationMethod None
EstimatedCount 0
HasLineage False
IsAggregatable True
OrderBy Key
OrderByAttribute
ProcessingState Unprocessed
TokenizationBehavior TokenizationNone
UserEditFlag 0
Basic
Description
FormatString
ID Age Id
Name Age Id
Type Regular
Usage Key
Misc
Parent-Child
MembersWithData NonLeafDataVisible
MembersWithDataCap
NamingTemplate
RootMemberIf ParentIsBlankSelfOrMissin
UnaryOperatorColumn (none)
Source
CustomRollupColumn (none)
KeyColumns
NameColumn (none)
ValueColumn (none)
Name
Specifies the name of the object.

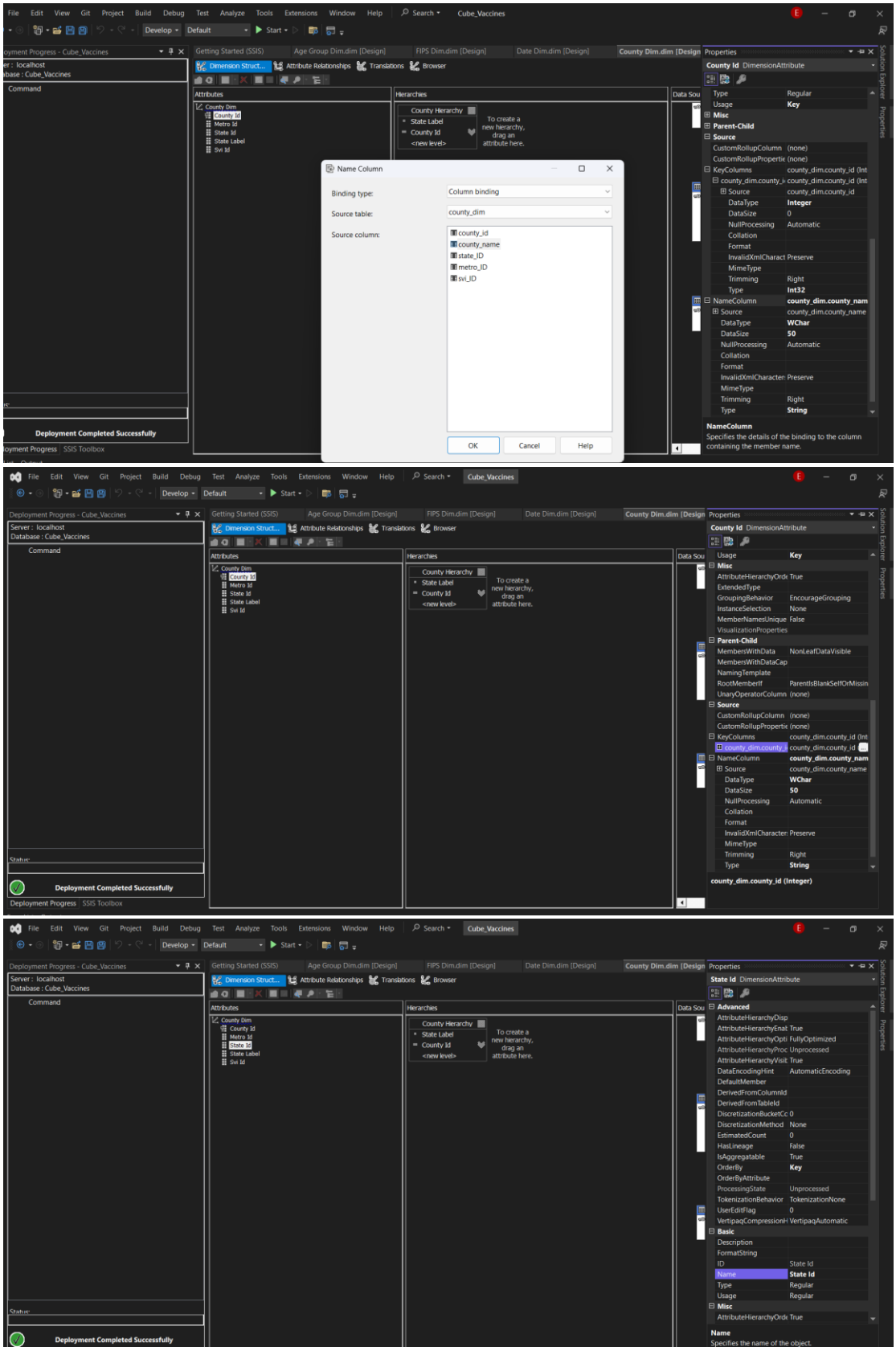

Deployment Progress - Cube_Vaccines
Server : localhost
Database : Cube_Vaccines
Command
Status:
Deployment Completed Successfully
Deployment Progress
SSIS Toolbox
Error List
Output
Age Group Di...dim [Design]*
Date Dim.dim [Design]
County Dim.dim [Design]
Vaccines DB.cube [Design]
Getting Started
Dimension Struct...
Attribute Relationships
Translations
Browser
Attributes
Age Group Dim
Age Group
Age Id
Hierarchies
To create a new hierarchy, drag an attribute here.
Properties
Age Group DimensionAttribute
DiscretizationMethod None
EstimatedCount 0
HasLineage False
IsAggregatable True
OrderBy Name
OrderByAttribute
ProcessingState Unprocessed
TokenizationBehavior TokenizationNone
UserEditFlag 0
Basic
Description
FormatString
ID Age Group
Name Age Group
Type Regular
Usage Regular
Misc
Parent-Child
MembersWithData NonLeafDataVisible
MembersWithDataCap
NamingTemplate
RootMemberIf ParentIsBlankSelfOrMissin
UnaryOperatorColumn (none)
Source
CustomRollupColumn (none)
KeyColumns
NameColumn (none)
ValueColumn (none)
Name
Specifies the name of the object.

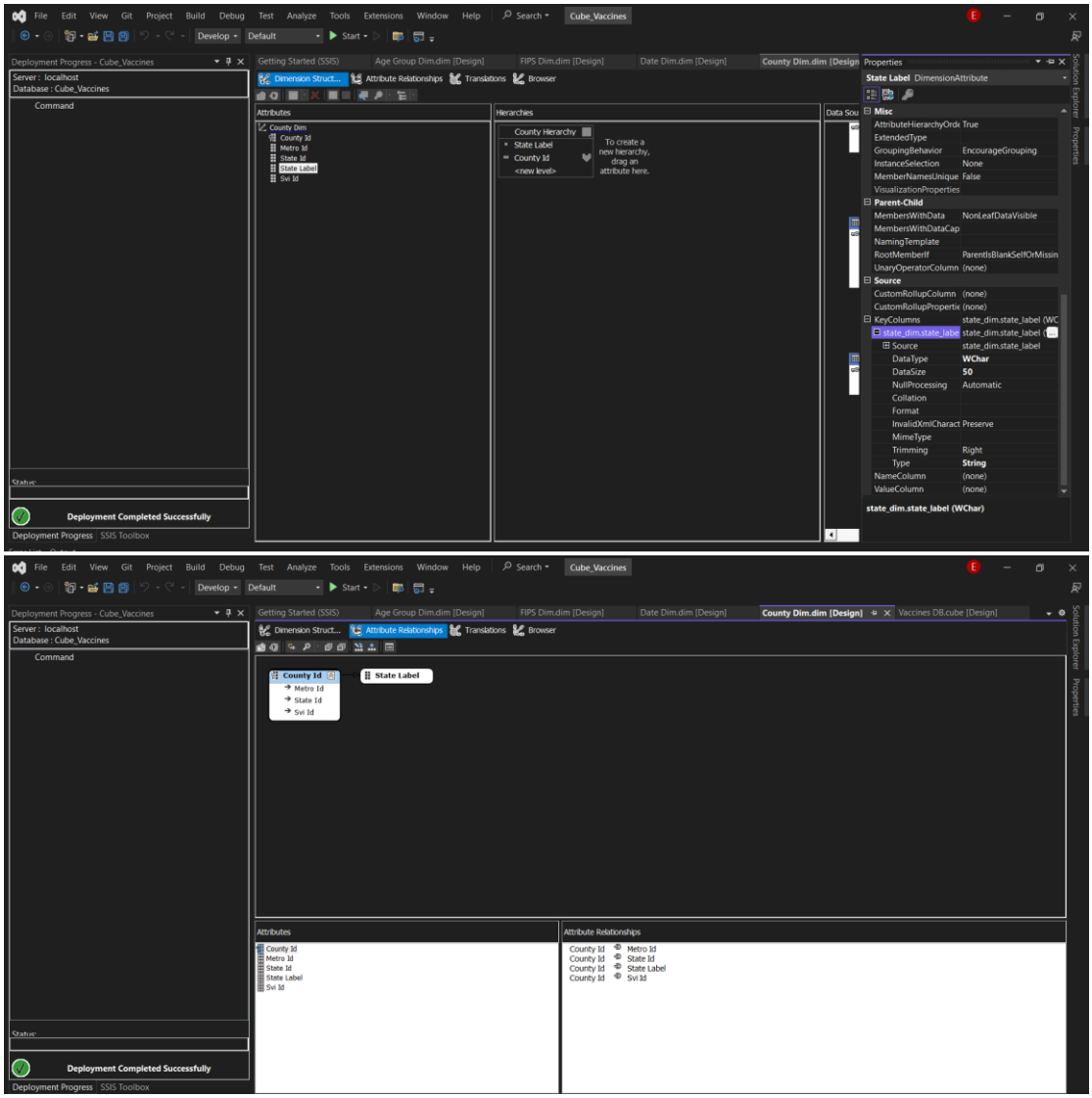

Deployment Progress - Cube_Vaccines
Server : localhost
Database : Cube_Vaccines
Command
Status:
Deployment Completed Successfully
Deployment Progress
SSIS Toolbox
Getting Started (SSIS)
Age Group Dim.dim [Design]
FIPS Dim.dim [Design]
Date Dim.dim [Design]
County Dim.dim [Design]
Vaccines DB.cube [Design]
Dimension Struct...
Attribute Relationships
Translations
Browser
Age Id
Age Group
Attributes
Age Group
Age Id
Attribute Relationships
Age Id
Age Group

Then we deployed and processed the cube once more to update these two dimensions with their attributes. After that we edited county_dim dimension, by adding the State Label into its attributes and by creating a natural hierarchy between County and State. We added state label in order to use the name of the State and not their IDs in our visualizations for the same reason we modified county_id by setting NameColumn field to take values from the county_name in the properties tab of the county_id attribute.

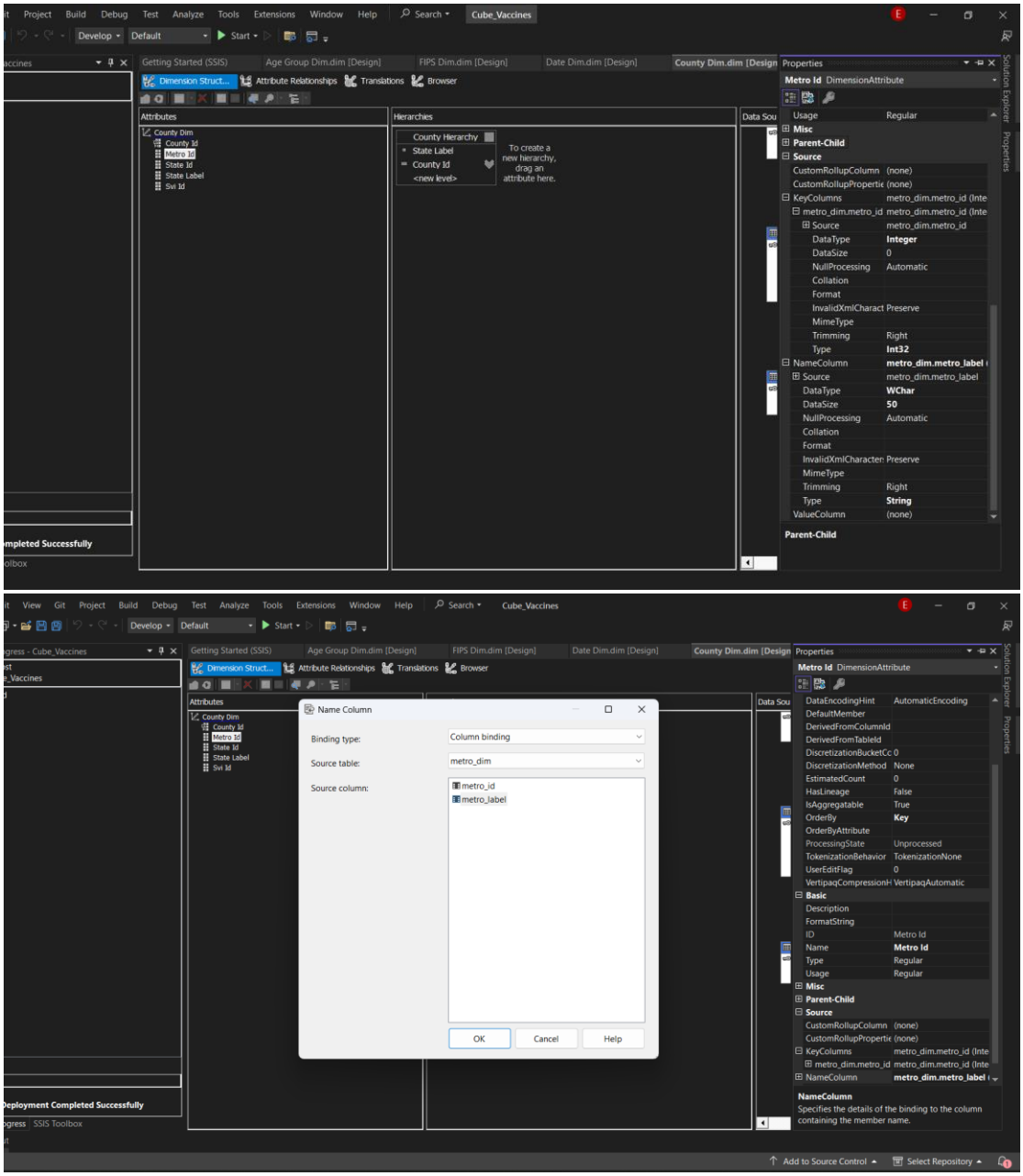




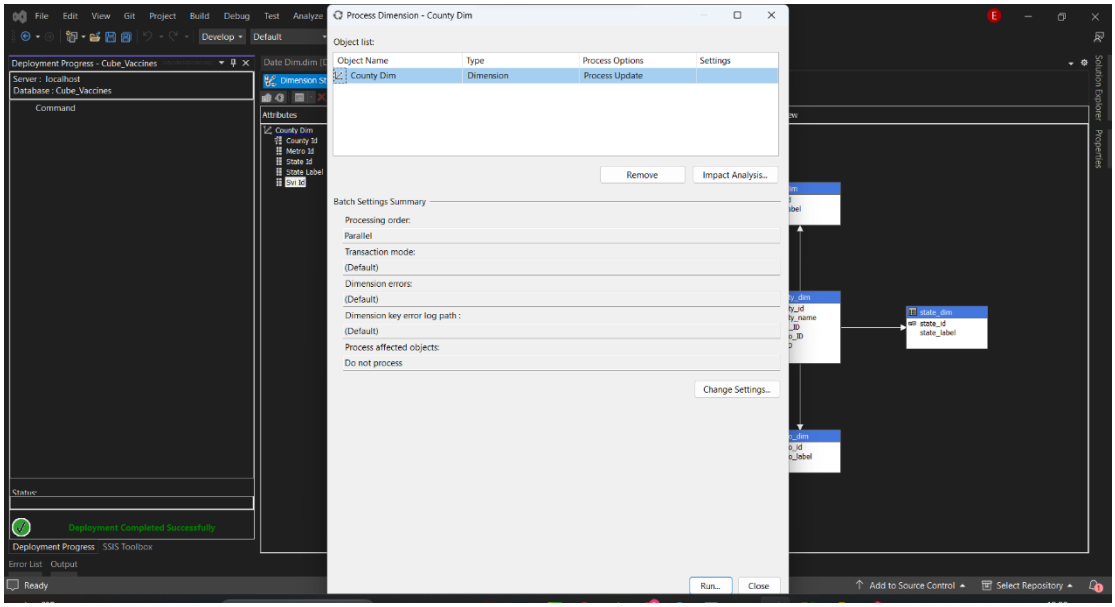

Name Column
Binding type:
Column binding
Source table:
county_dim
Source column:
county_id
county_name
state_ID
metro_ID
svi_ID
OK
Cancel
Help
Deployment Completed Successfully

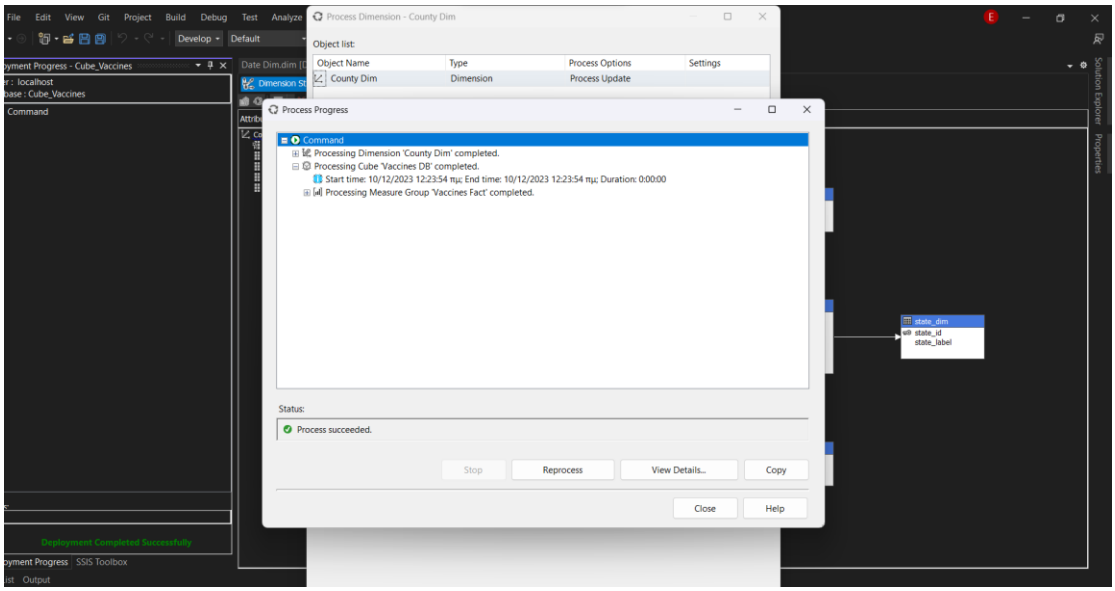


Also we modified the metro_ID in the same way we did with county_id (setting NameColumn field to take values from the metro_label in the properties tab of the metro_ID attribute) to use it later in a Power BI chart in a nice way.

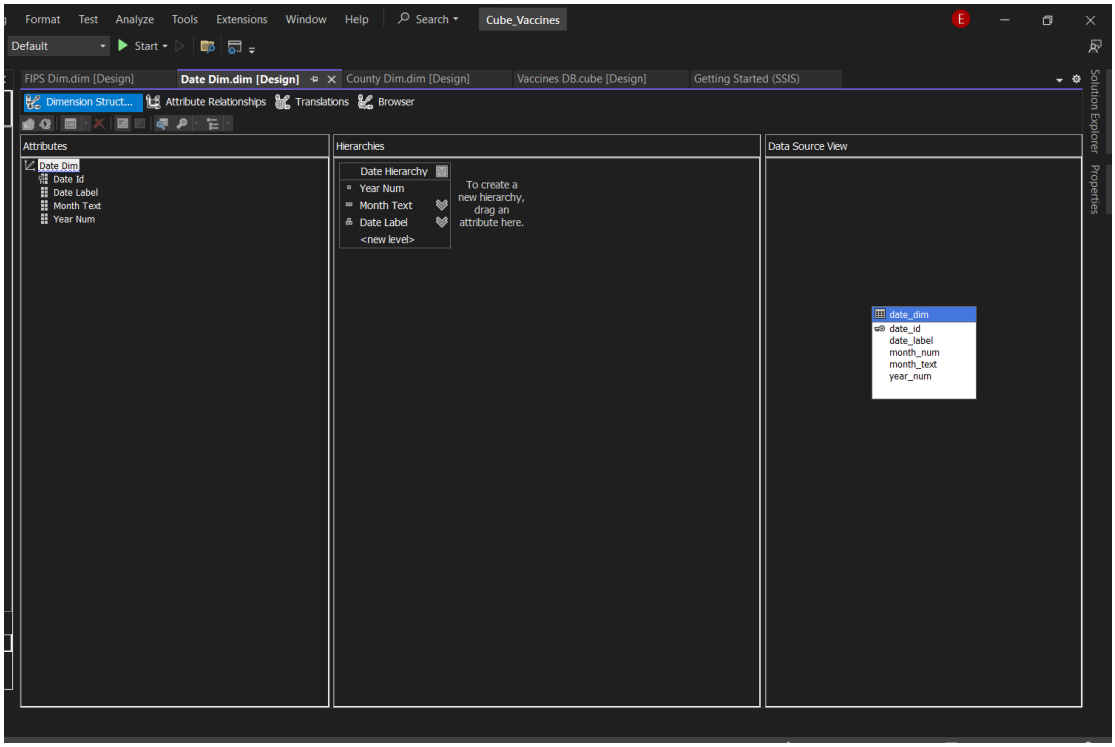


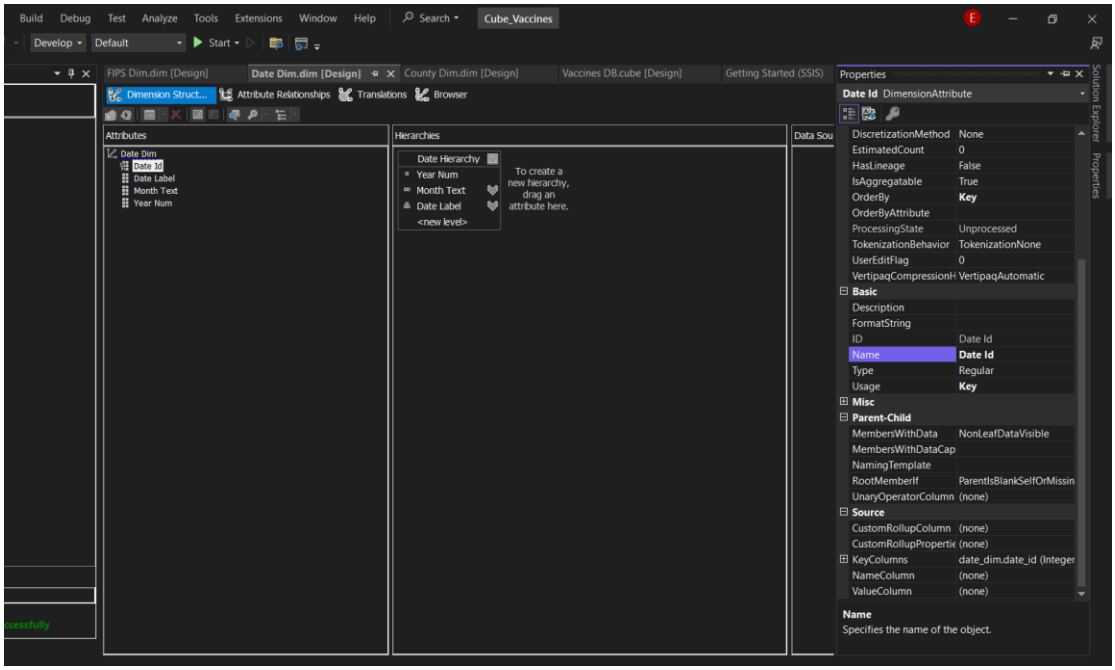
Finally we deployed and processed again the dimension and the cube to be able to use the hierarchy and finalize the alterations we did at the metro_ID and county_id.

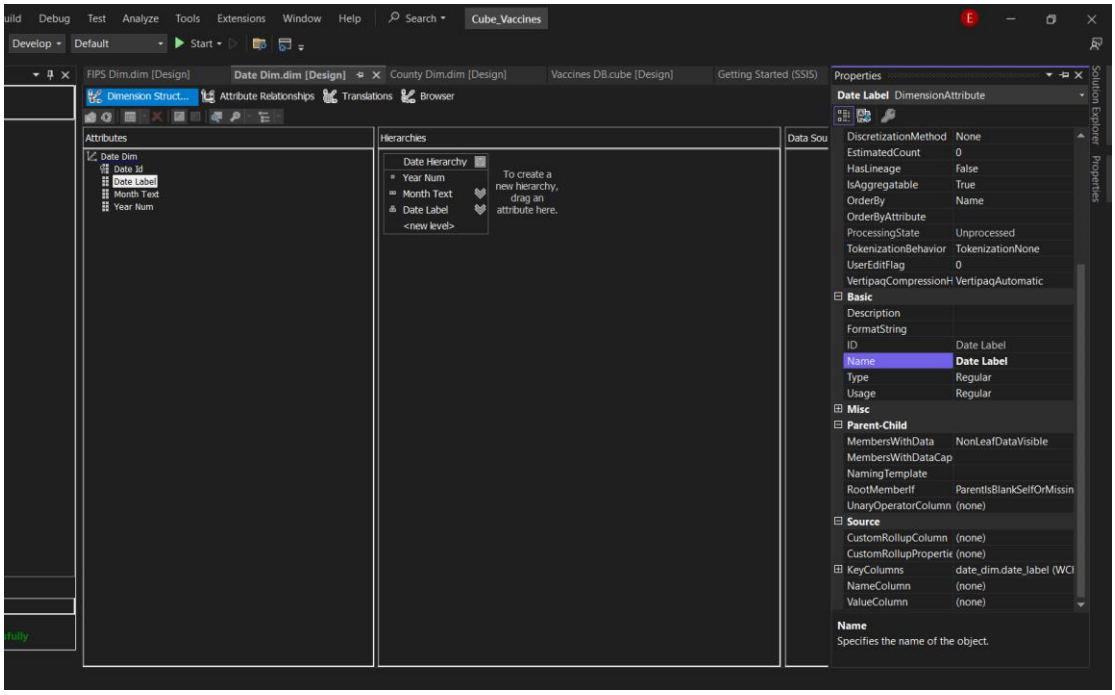




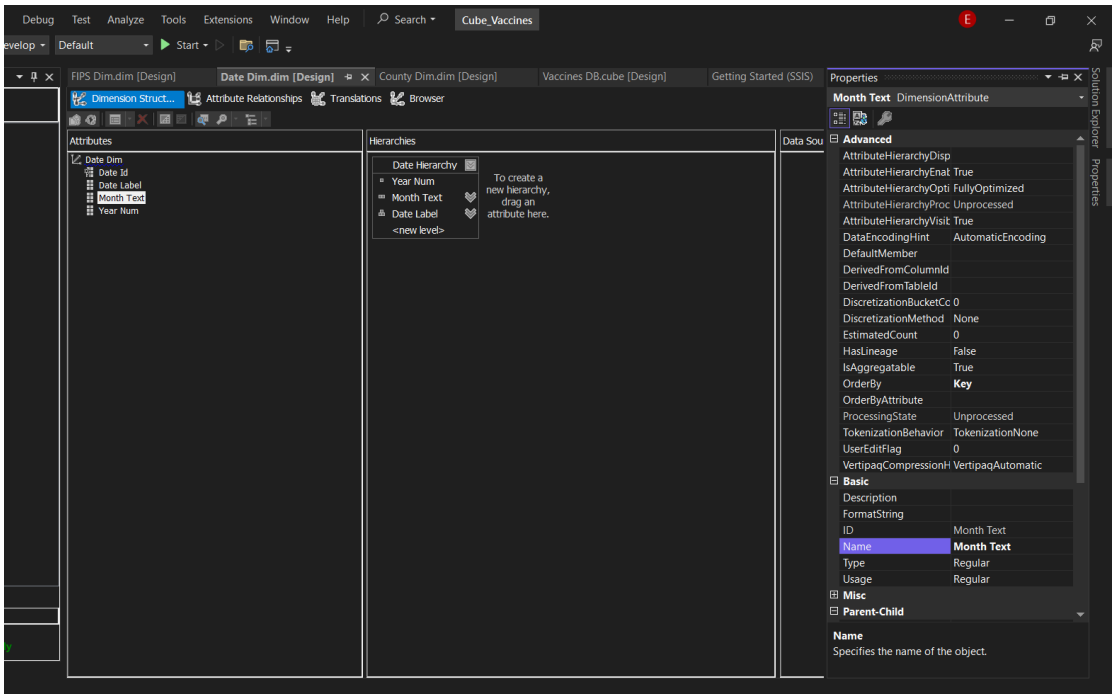
Last but not least we edited the date_dim table to form a natural hierarchy between full date (Date Label attribute), Month (Month Text attribute) and Year (Year Num attribute) and add the full dates in the attributes of the dimension to use it later in our visualizations.

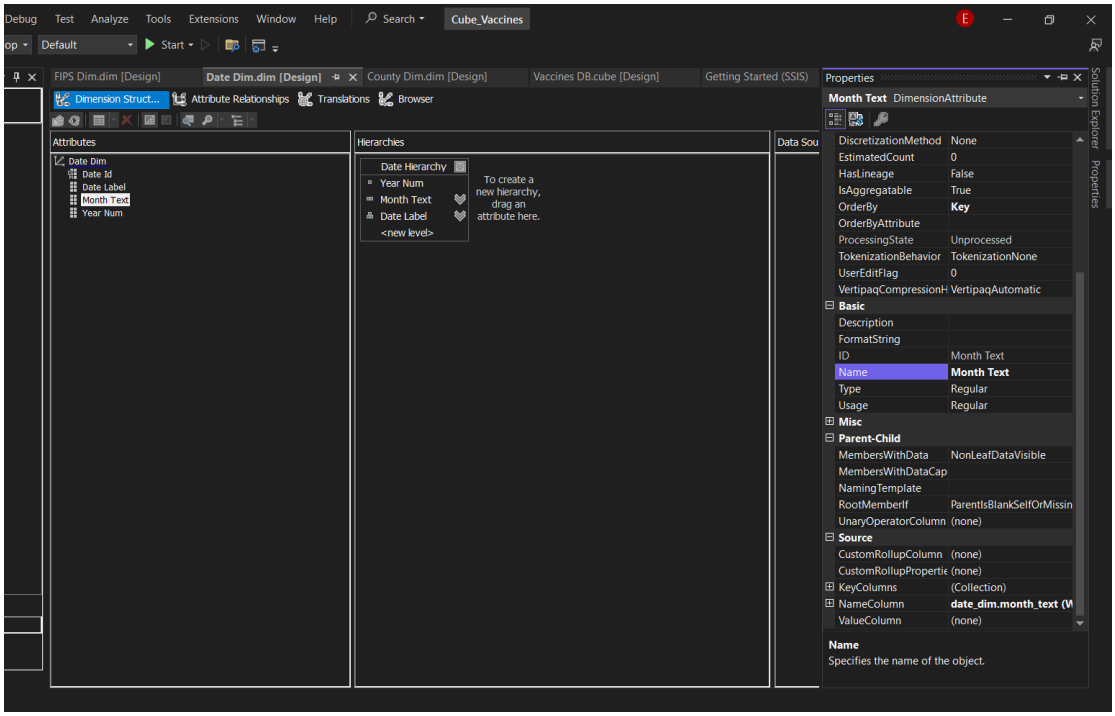

Format Test Analyze Tools Extensions Window Help Search Cube_Vaccines
Default Start
FIPS Dim.dim [Design] Date Dim.dim [Design] County Dim.dim [Design] Vaccines DB.cube [Design] Getting Started (SSIS)
Dimension Struct... Attribute Relationships Translations Browser
Attributes
Date Dim
Date Id
Date Label
Month Text
Year Num
Hierarchies
Date Hierarchy
Year Num
Month Text
Date Label
<new level>
To create a new hierarchy, drag an attribute here.
Data Source View
date_dim
date_id
date_label
month_num
month_text
year_num
Solution Explorer
Properties

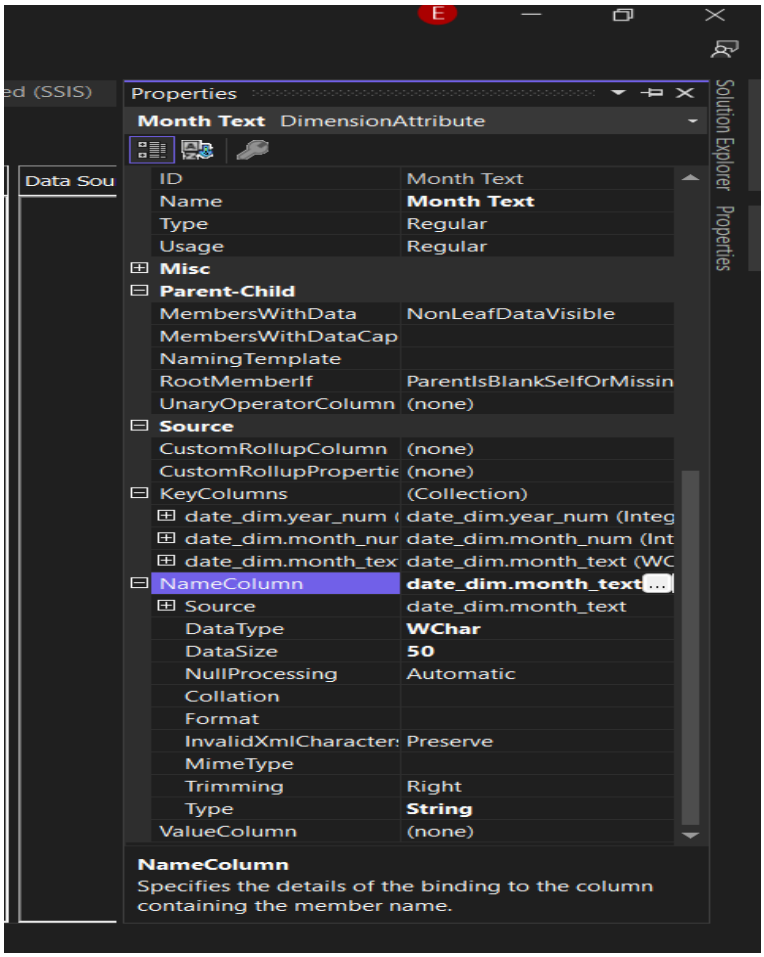

Build Debug Test Analyze Tools Extensions Window Help Search Cube_Vaccines
Develop Default Start
FIPS Dim.dim [Design] Date Dim.dim [Design] County Dim.dim [Design] Vaccines DB.cube [Design] Getting Started (SSIS)
Dimension Struct... Attribute Relationships Translations Browser
Attributes
Date Dim
Date Id
Date Label
Month Text
Year Num
Hierarchies
Date Hierarchy
Year Num
Month Text
Date Label
<new level>
To create a new hierarchy, drag an attribute here.
Properties
Date Id DimensionAttribute
DiscretizationMethod None
EstimatedCount 0
HasLineage False
IsAggregatable True
OrderBy Key
OrderByAttribute
ProcessingState Unprocessed
TokenizationBehavior TokenizationNone
UserEditFlag 0
VertipaqAutomatic
Basic
Description
FormatString
ID Date Id
Name Date Id
Type Regular
Usage Key
Misc
Parent-Child
MembersWithData NonLeafDataVisible
NamingTemplate
RootMemberIf
UnaryOperatorColumn (none)
Source
CustomRollupColumn (none)
KeyColumns date_dim.date_id (Integer
NameColumn (none)
ValueColumn (none)
Name
Specifies the name of the object.
Solution Explorer
Properties

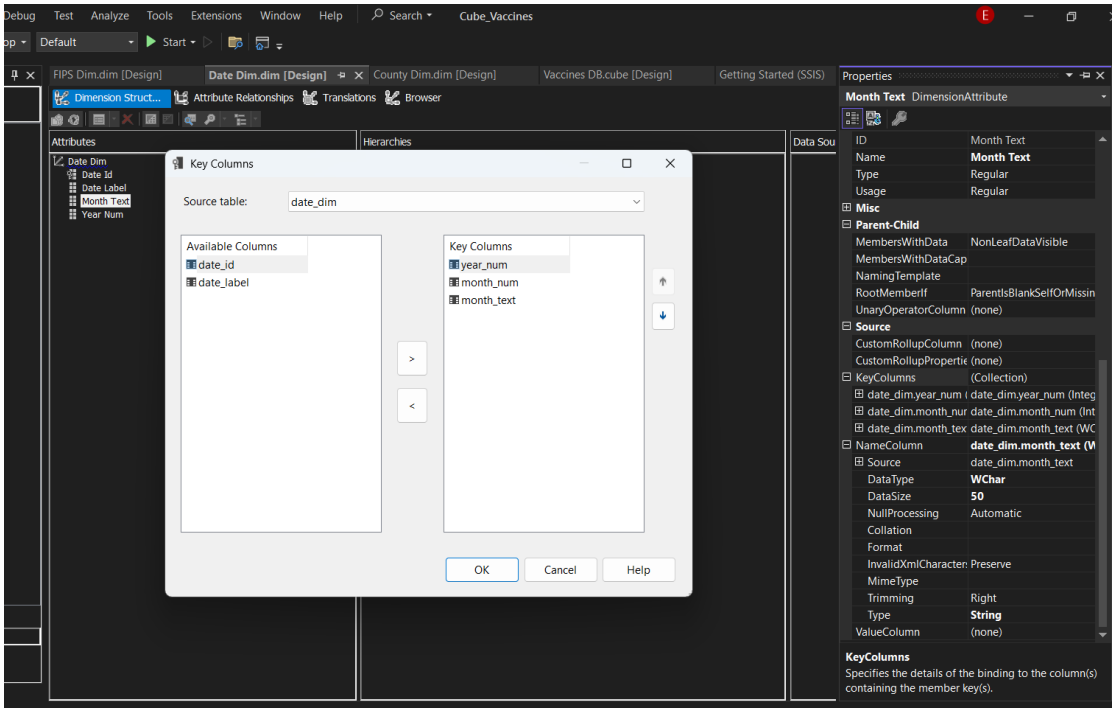


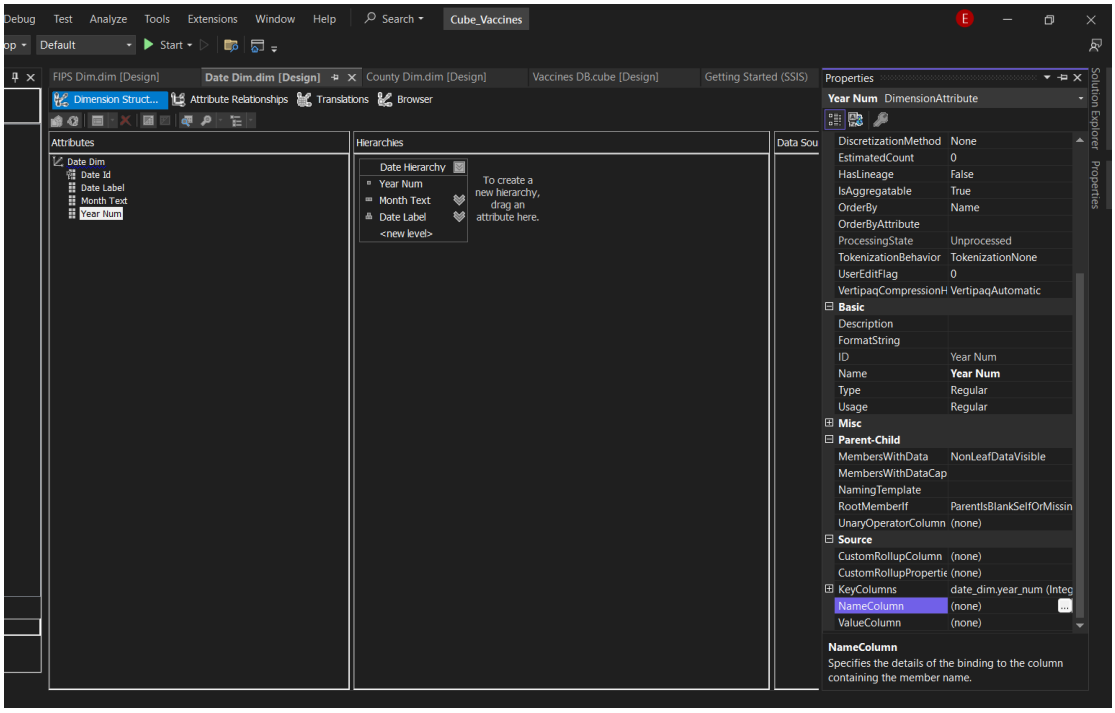
Also we modified the Month Text attribute in order to display the month in a textual form but be identified as a key by the combination of the month (in a numeric form) and the year.



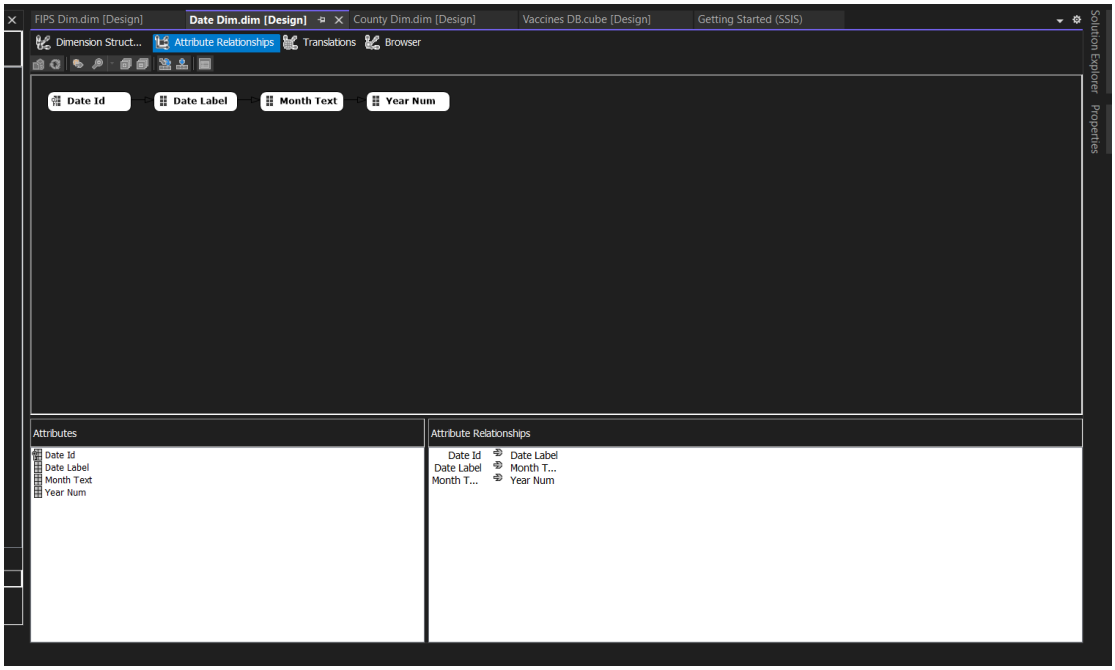

Debug Test Analyze Tools Extensions Window Help Search Cube_Vaccines
Default Start
FIPS Dim.dim [Design] Date Dim.dim [Design] County Dim.dim [Design] Vaccines DB.cube [Design] Getting Started (SSIS)
Dimension Struct... Attribute Relationships Translations Browser
Attributes
Date Dim
Date Id
Date Label
Month Text
Year Num
Hierarchies
Date Hierarchy
Year Num
Month Text
Date Label
<new level>
To create a new hierarchy, drag an attribute here.
Properties
Month Text DimensionAttribute
DiscretizationMethod None
EstimatedCount 0
HasLineage False
IsAggregatable True
OrderBy Key
OrderByAttribute
ProcessingState Unprocessed
TokenizationBehavior TokenizationNone
UserEditFlag 0
VertipaqCompressionH VertipaqAutomatic
Basic
Description
FormatString
ID Month Text
Name Month Text
Type Regular
Usage Regular
Misc
Parent-Child
MembersWithData NonLeafDataVisible
MembersWithDataCap
NamingTemplate
RootMemberIf ParentIsBlankSelfOrMissin
UnaryOperatorColumn (none)
Source
CustomRollupColumn (none)
CustomRollupPropertie (none)
KeyColumns (Collection)
NameColumn date_dim.month_text (W
ValueColumn (none)
Name
Specifies the name of the object.
Solution Explorer Properties


Properties
Month Text DimensionAttribute
ID Month Text
Name Month Text
Type Regular
Usage Regular
Misc
Parent-Child
MembersWithData NonLeafDataVisible
MembersWithDataCap
NamingTemplate
RootMemberIf ParentIsBlankSelfOrMissin
UnaryOperatorColumn (none)
Source
CustomRollupColumn (none)
CustomRollupPropertie (none)
KeyColumns (Collection)
date_dim.year_num (date_dim.year_num (Integ
date_dim.month_nur date_dim.month_num (Int
date_dim.month_tex date_dim.month_text (WC
NameColumn date_dim.month_text...
Source date_dim.month_text
DataType WChar
DataSize 50
NullProcessing Automatic
Collation
Format
InvalidXmlCharacter: Preserve
MimeType
Trimming Right
Type String
ValueColumn (none)
NameColumn
Specifies the details of the binding to the column containing the member name.
Solution Explorer Properties

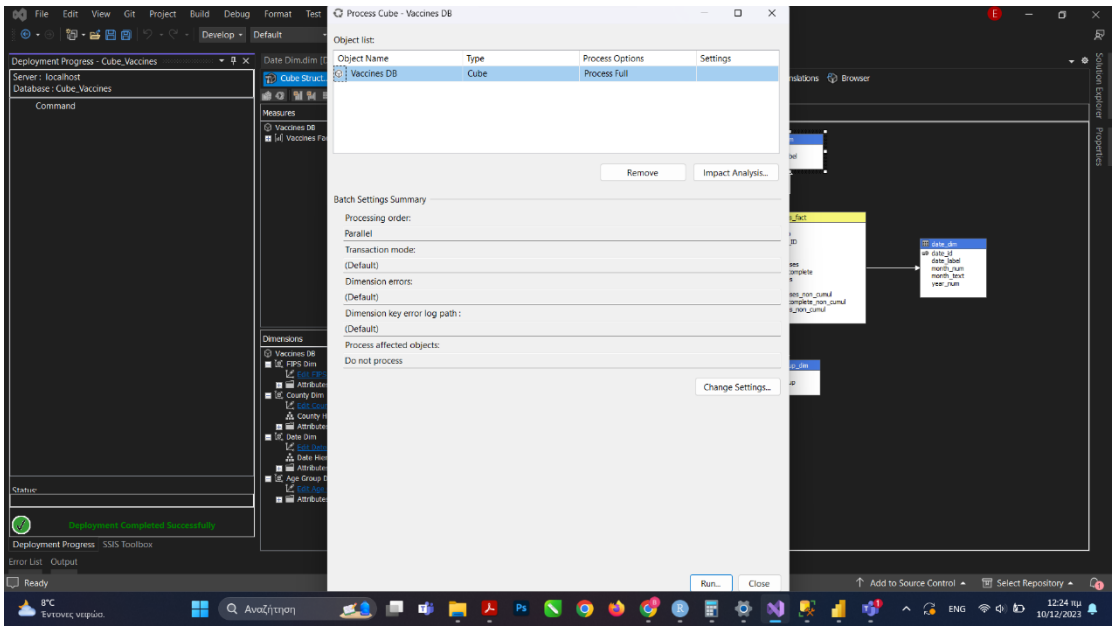




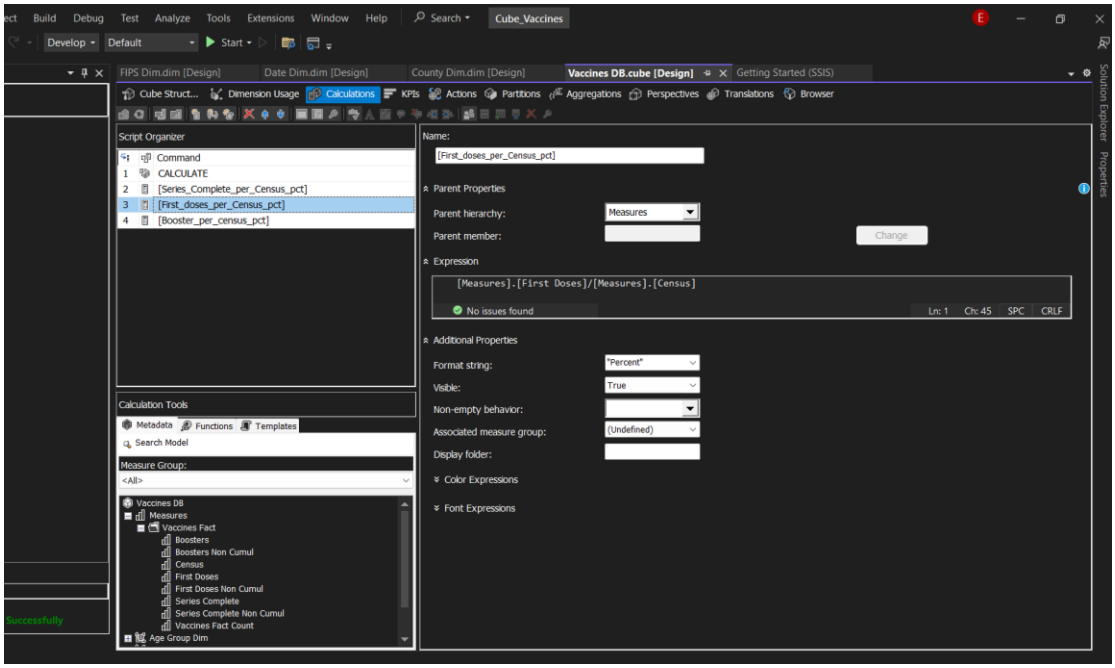
The relationships of the attributes that structure the hierarchy as natural is the displayed like this:

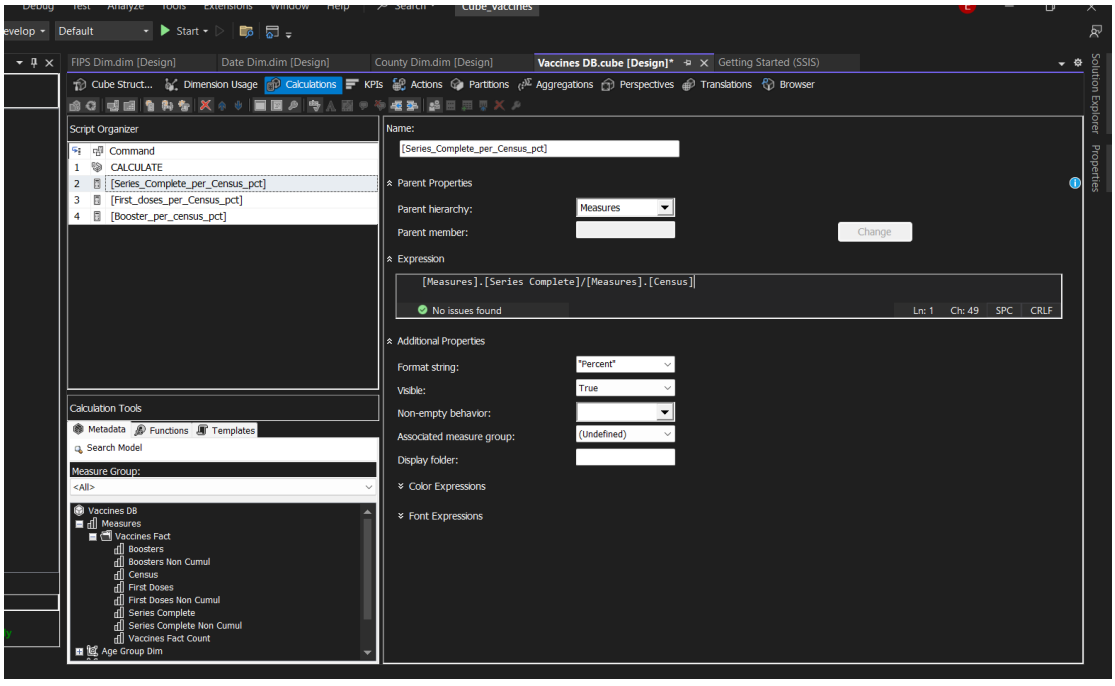


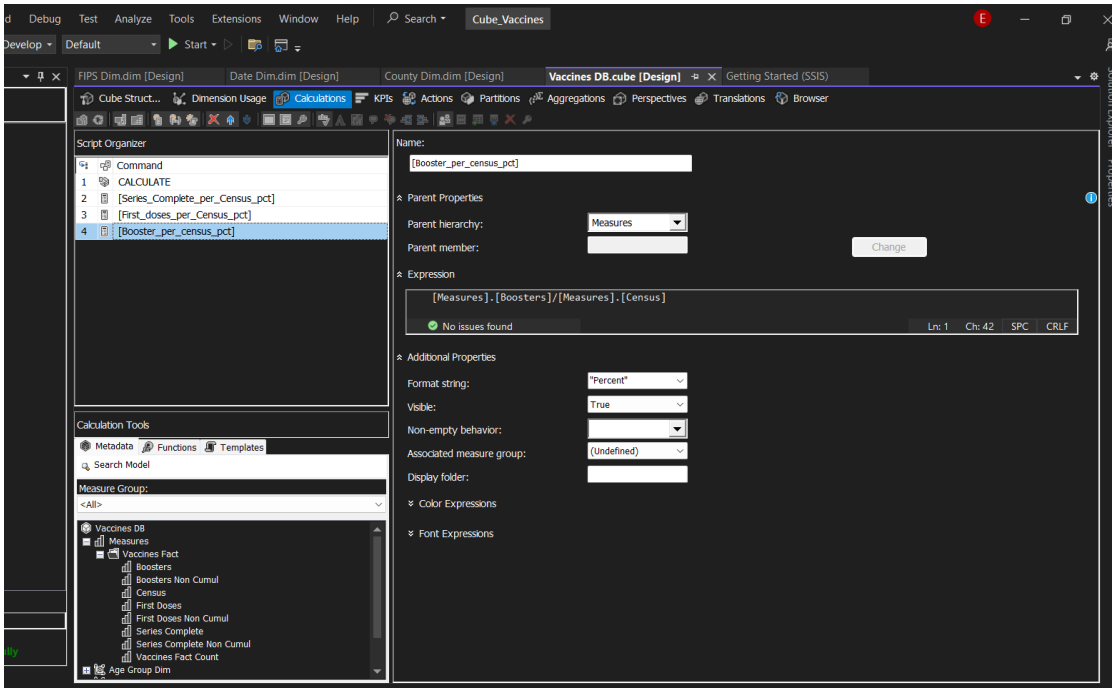
Finally we deployed and processed again the date dimension and the cube.



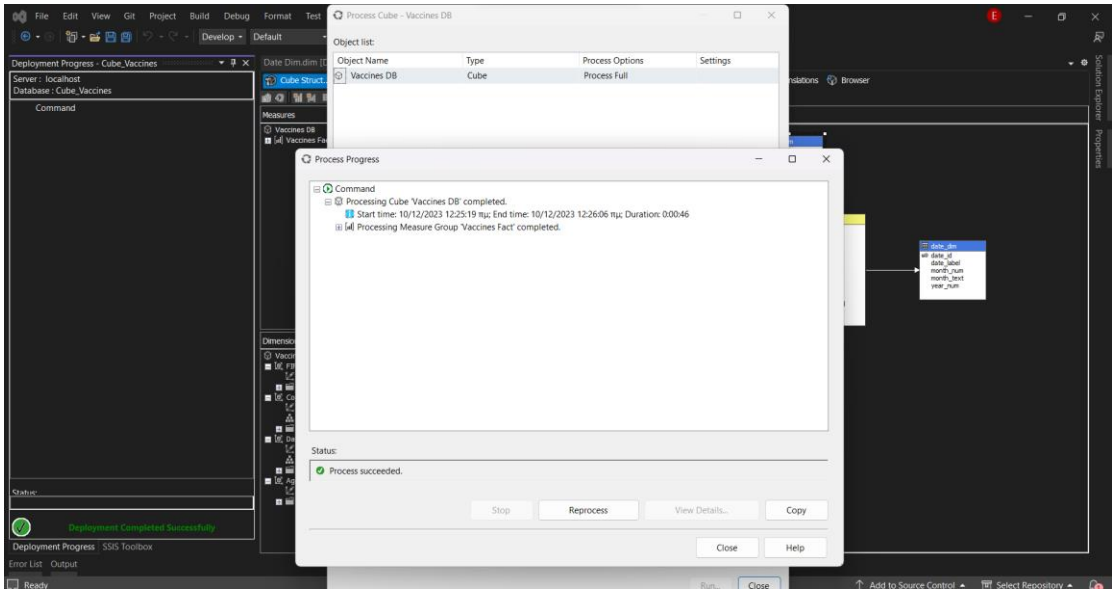
Before the creation of our charts and visualizations in the Power BI environment we created three calculated measures to use them to gain the insights we wanted, those measures are the percentages of the three vaccine types, hence the number of the people who did each vaccine type divided by the population that interests us (according to the county and age group). We named these three percentages: First Doses per Census pct ,Series Complete per Census pct and Boosters per Census pct. We calculated them in the way shown below in the calculated measures tab:


Build Debug Test Analyze Tools Extensions Window Help Search Cube_Vaccines
Develop Default Start
FIPS Dim.dim [Design] Date Dim.dim [Design] County Dim.dim [Design] Vaccines DB.cube [Design] Getting Started (SSIS)
Cube Struct... Dimension Usage Calculations KPIs Actions Partitions Aggregations Perspectives Translations Browser
Script Organizer
Command
1 CALCULATE
2 [Series_Complete_per_Census_pct]
3 [First_doses_per_Census_pct]
4 [Booster_per_census_pct]
Name:
[First_doses_per_Census_pct]
Parent Properties
Parent hierarchy: Measures
Parent member:
Change
Expression
[Measures].[First Doses]/[Measures].[Census]
No issues found
Ln: 1 Ch: 45 SPC CRLF
Additional Properties
Format string: "Percent"
Visible: True
Non-empty behavior:
Associated measure group: (Undefined)
Display folder:
Color Expressions
Font Expressions
Calculation Tools
Metadata Functions Templates
Search Model
Measure Group:
<All>
Vaccines DB
Measures
Vaccines Fact
Boosters
Boosters Non Cumul
Census
First Doses
First Doses Non Cumul
Series Complete
Series Complete Non Cumul
Vaccines Fact Count
Age Group Dim
Solution Explorer
Properties


Default Start
FIPS Dim.dim [Design] Date Dim.dim [Design] County Dim.dim [Design] Vaccines DB.cube [Design]* Getting Started (SSIS)
Cube Struct... Dimension Usage Calculations KPIs Actions Partitions Aggregations Perspectives Translations Browser
Script Organizer
Command
1 CALCULATE
2 [Series_Complete_per_Census_pct]
3 [First_doses_per_Census_pct]
4 [Booster_per_census_pct]
Name:
[Series_Complete_per_Census_pct]
Parent Properties
Parent hierarchy: Measures
Parent member:
Change
Expression
[Measures].[Series Complete]/[Measures].[Census]
No issues found
Ln: 1 Ch: 49 SPC CRLF
Additional Properties
Format string: "Percent"
Visible: True
Non-empty behavior:
Associated measure group: (Undefined)
Display folder:
Color Expressions
Font Expressions
Calculation Tools
Metadata Functions Templates
Search Model
Measure Group:
<All>
Vaccines DB
Measures
Vaccines Fact
Boosters
Boosters Non Cumul
Census
First Doses
First Doses Non Cumul
Series Complete
Series Complete Non Cumul
Vaccines Fact Count
Age Group Dim
Solution Explorer
Properties



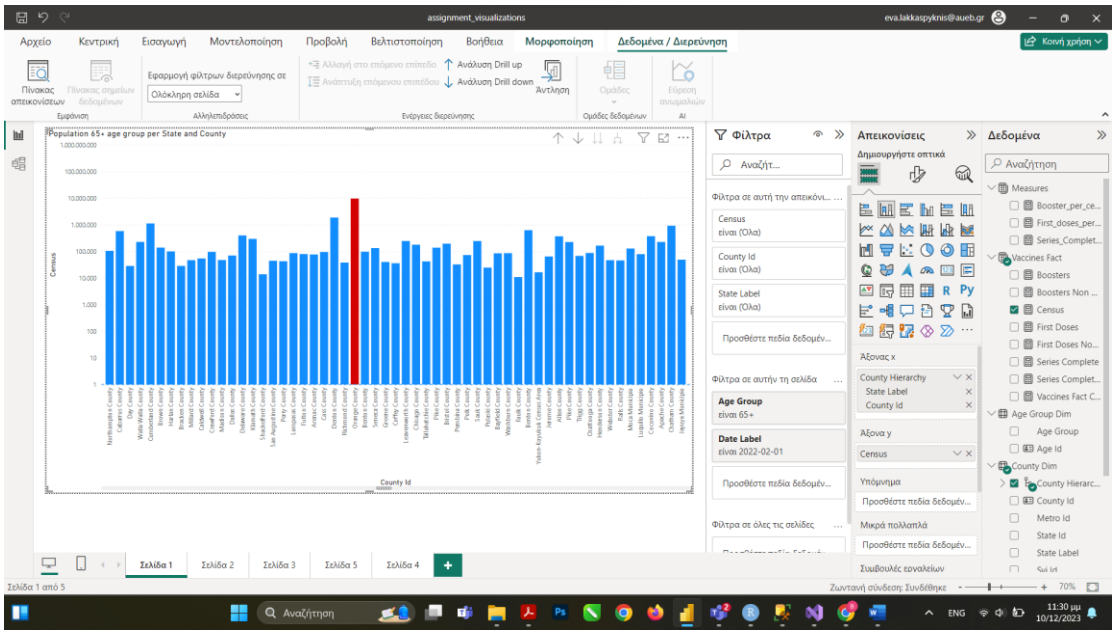
Then we deployed the cube so we can have these measures available in the Power BI environment.

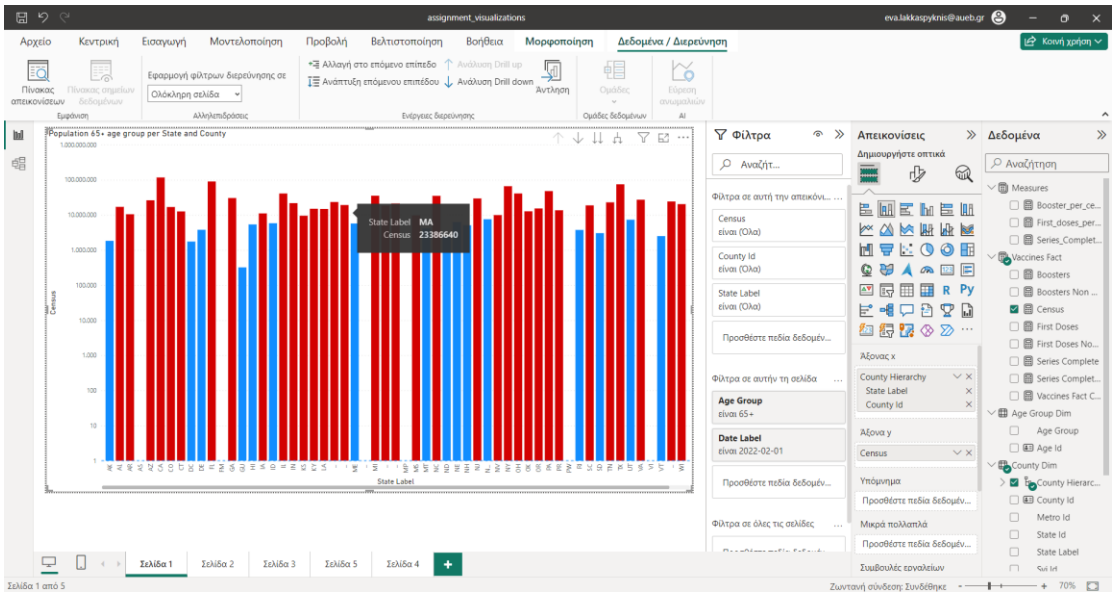


PART C: Power BI Visualizations

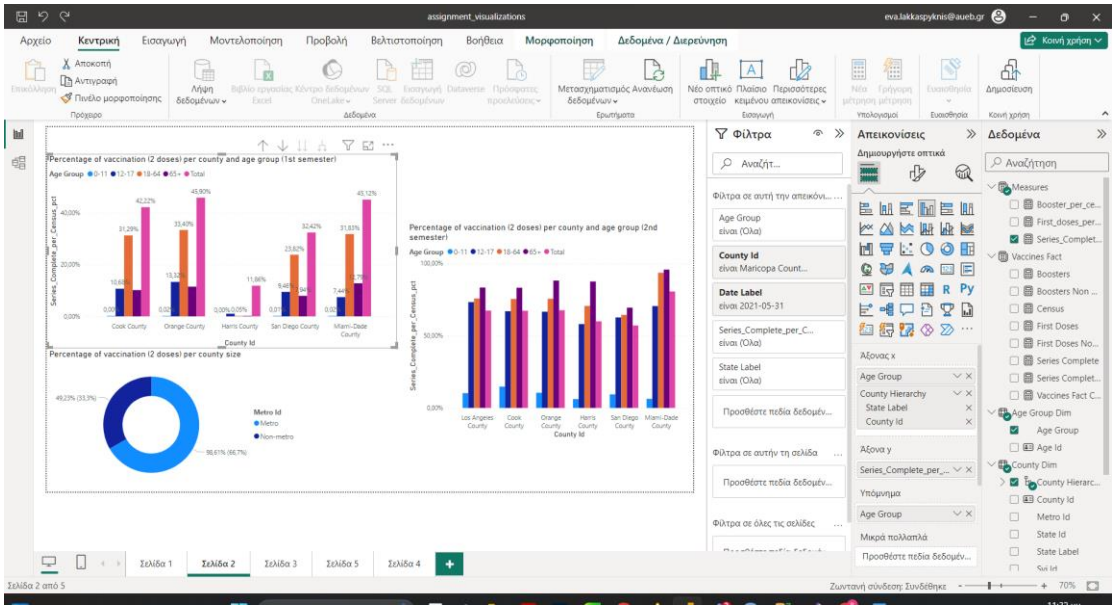
Important note: Because the visualizations may not be proberly displayed here we will also upload a pdf file that has been exported from Power BI Desktop.

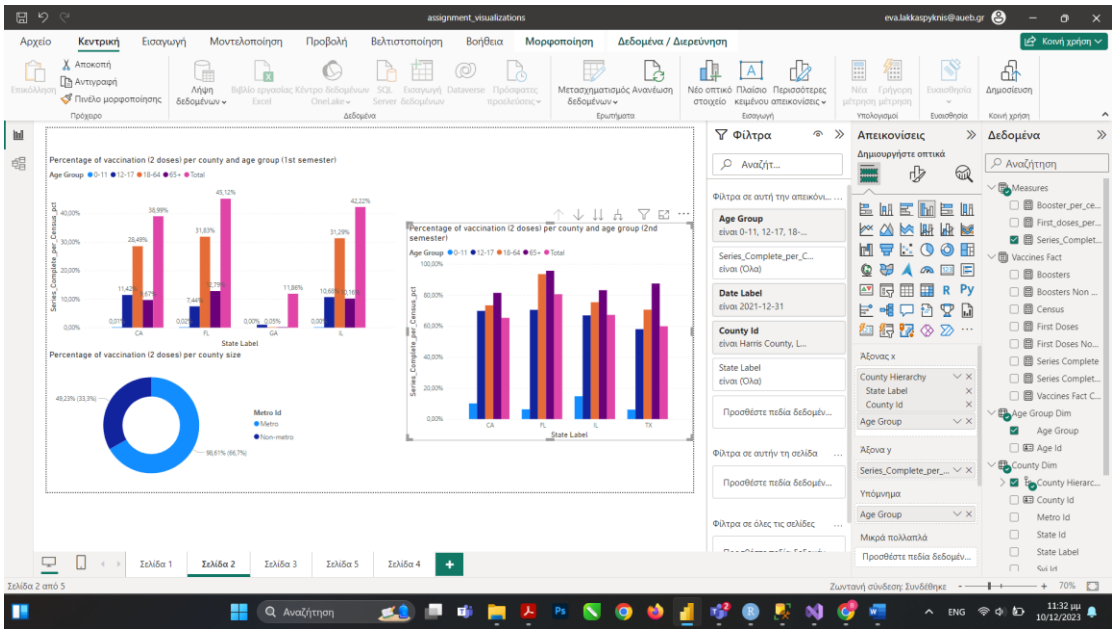
Firstly, we created a bar plot which depicts the population of 65plus age group in each county in order to find which counties have the higher population of senior citizens and hence the bigger need for proper vaccination policies. In the diagrams that follow we see with red bars the counties with very big population of senior citizens. We used a filter for the age group and date (because are data were cumulative) and the county hierarchy that we created in SSAS environment so we can perform also drill downs(first bar plot) and roll ups(2nd bar plot).

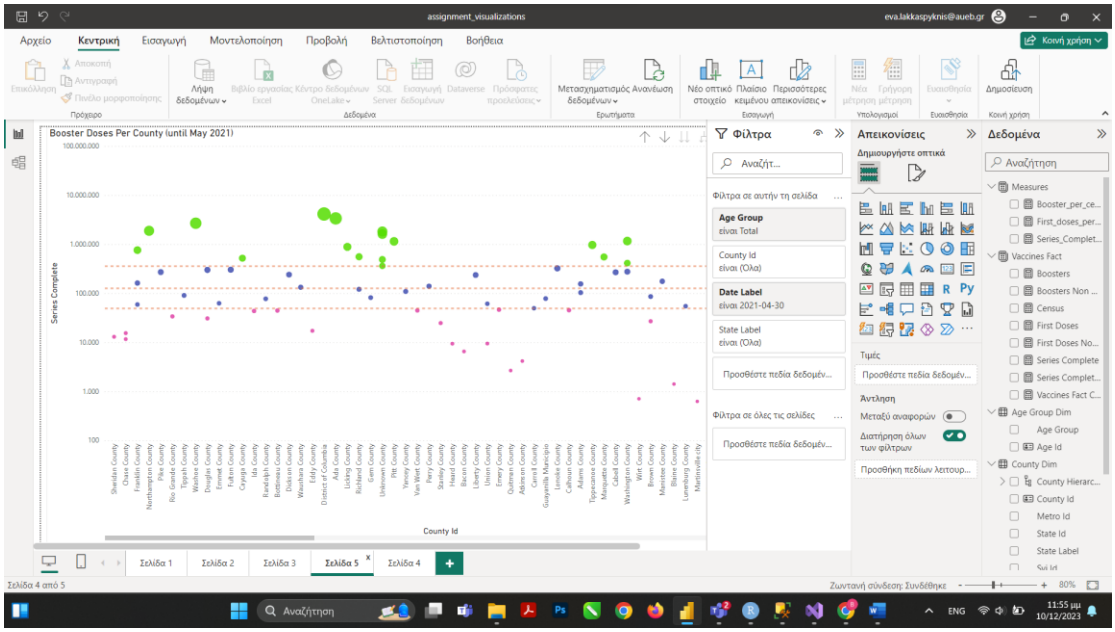


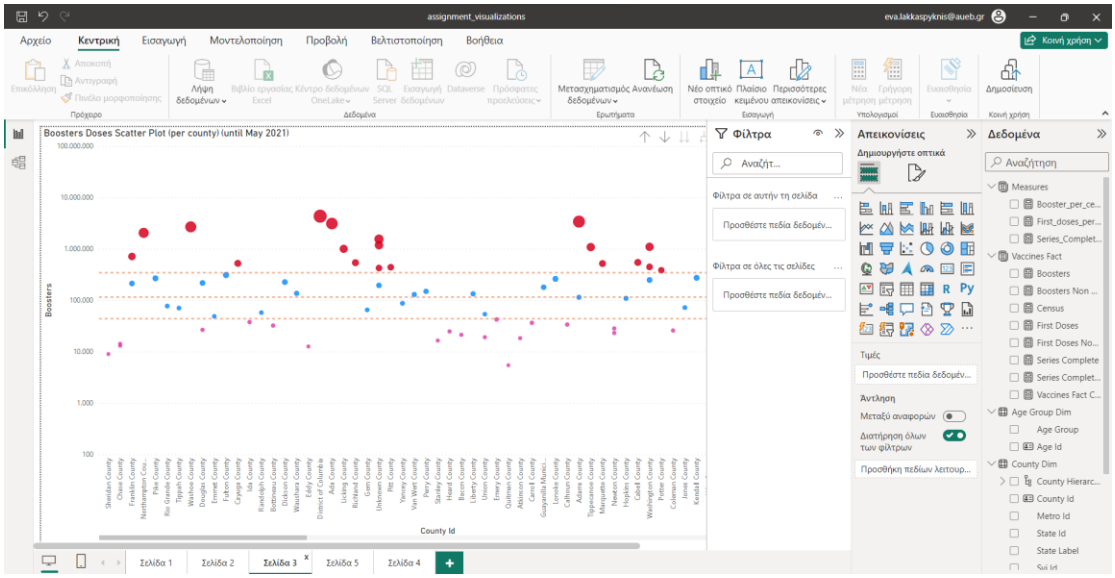


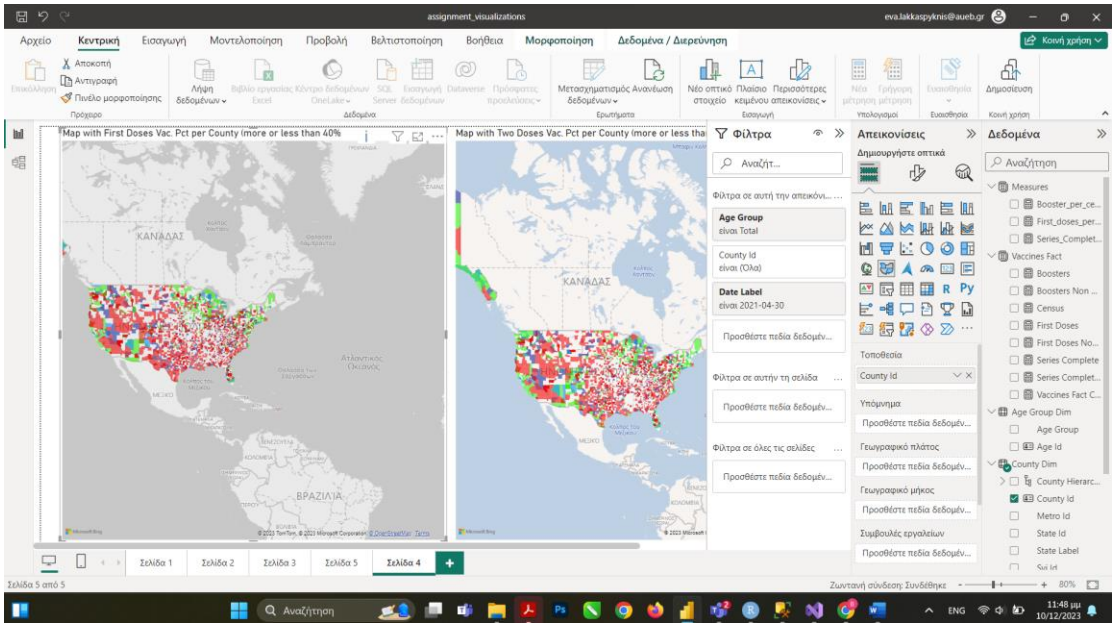
Here we see the vaccination percentages (left for only the first dose, right for the 2 doses) for the counties that have been captured from the previous barplot) by using a stacked barplot for each county and age group. We can perform again drill downs and roll ups by using county hierarchies.






assignment_visualizations
Booster Doses Per County (until May 2021)
Series Complete
County Id
Φίλτρα
Απεικονίσεις
Δεδομένα
Σελίδα 1
Σελίδα 2
Σελίδα 3
Σελίδα 5
Σελίδα 4


assignment_visualizations
Boosters Doses Scatter Plot (per county) (until May 2021)
Boosters
County Id
Φίλτρα
Απεικονίσεις
Δεδομένα
Σελίδα 1
Σελίδα 2
Σελίδα 3
Σελίδα 5
Σελίδα 4


assignment_visualizations
eva.lakkaspyknis@aueb.gr
Αρχείο
Κεντρική
Εισαγωγή
Μοντελοποίηση
Προβολή
Βελτιστοποίηση
Βοήθεια
Μορφοποίηση
Δεδομένα / Διερεύνηση
Κοινή χρήση
Map with First Doses Vac. Pct per County (more or less than 40%
Map with Two Doses Vac. Pct per County (more or less tha
ΚΑΝΑΔΑΣ
ΒΡΑΖΙΛΙΑ
Φίλτρα
Αναζήτ...
Age Group
είναι Total
County Id
είναι (Όλα)
Date Label
είναι 2021-04-30
Προσθέστε πεδία δεδομέν...
Απεικονίσεις
Δημιουργήστε οπτικά
Τοποθεσία
County Id
Υπόμνημα
Γεωγραφικό πλάτος
Γεωγραφικό μήκος
Συμβουλές εργαλείων
Δεδομένα
Measures
Booster_per_ce...
First_doses_per...
Series_Complet...
Vaccines Fact
Boosters
Boosters Non ...
Census
First Doses
First Doses No...
Series Complete
Series Complet...
Vaccines Fact C...
Age Group Dim
Age Group
Age Id
County Dim
County Hierarc...
County Id
Metro Id
State Id
State Label
Σελίδα 1
Σελίδα 2
Σελίδα 3
Σελίδα 5
Σελίδα 4
Σελίδα 5 από 5
Αναζήτηση
ENG
11:48 μμ
10/12/2023