**More Data Modelling**

A data model is essentially a blueprint that outlines how data is structured, stored, and managed in a database. It's a way of organizing and standardizing data elements and their relationships to ensure consistency, efficiency, and reliability in the management of data. Think of it as a map that guides the design and use of a database, helping to define the data elements, their attributes, and their relationships.

**Attributes of a Good Data Model**

A good data model should have several key attributes:

* **Accuracy**: It should correctly represent the real-world data and its relationships.
* **Simplicity**: The model should be as simple as possible while still being effective. Overly complex models can be difficult to maintain and understand.
* **Flexibility**: It should be adaptable to changes in requirements without requiring a complete overhaul.
* **Scalability**: The model should handle growth in data volume and complexity without significant performance degradation.
* **Consistency**: Ensures that data is consistently represented and standardized across the model.
* **Integrity**: It should maintain the integrity of the data, ensuring accuracy and reliability through constraints and validation rules.
* **Performance**: The model should facilitate efficient querying, data retrieval, and updates to ensure optimal and predictable performance.
* **Documentation**: Comprehensive documentation helps in understanding the structure, design, and purpose of the data model, aiding in future maintenance and updates. Modelling

A well-designed data model can significantly improve the performance of your Power BI reports. It allows for faster data retrieval, efficient query processing, and reduced memory usage, which leads to quicker response times. With a good data model, the process of transforming and preparing data becomes more straightforward. It reduces the need for complex transformations and joins, making the data easier to work with. This in turn help us writing less complex DAX formula. Also, it is easier to enable Row-Level Security in a good data model.

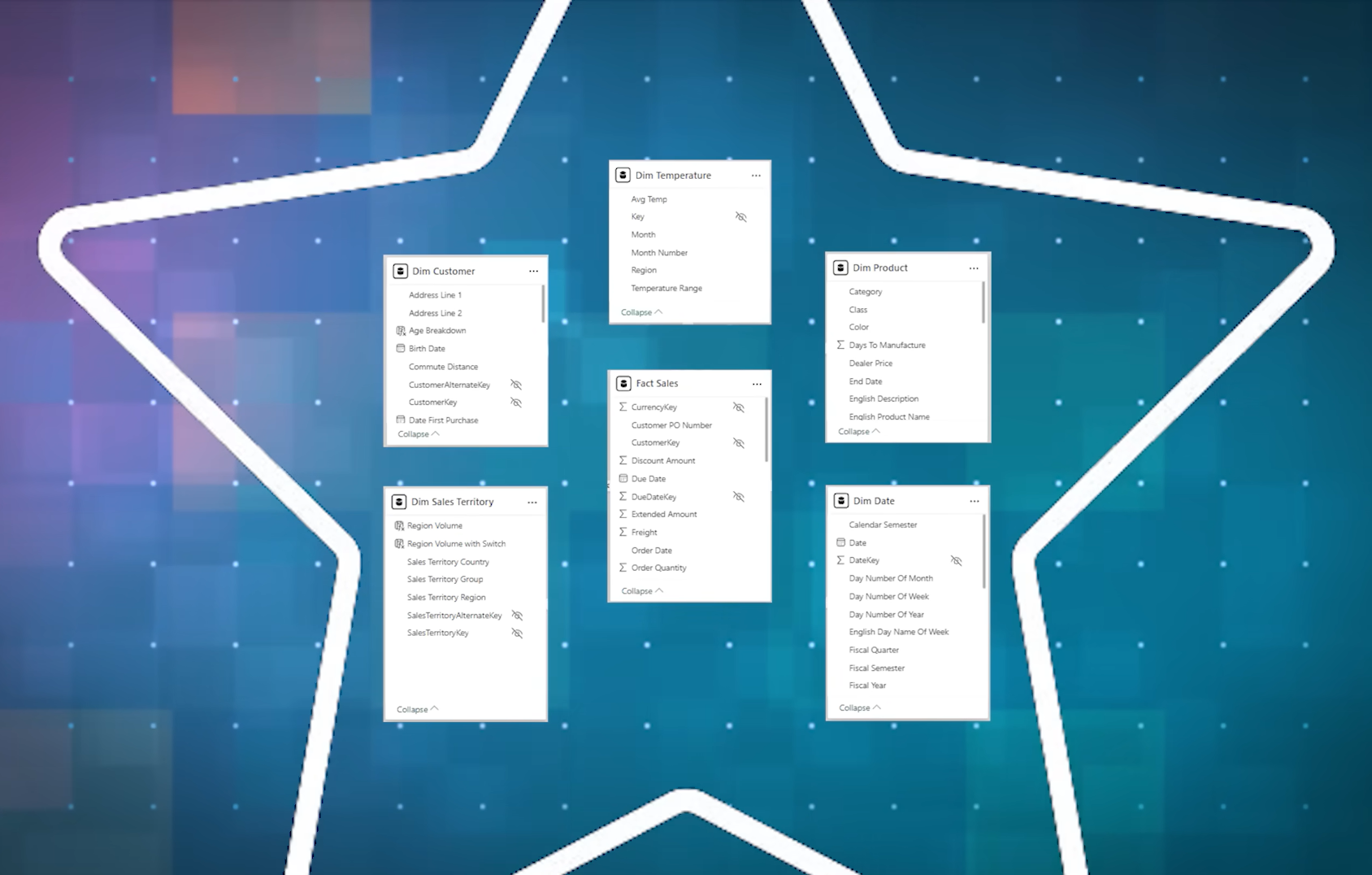
**Dimensional Modelling**

Dimensional modeling is a design concept used to structure data in a way that is easy to understand and analyze. It's particularly popular in data warehousing and business intelligence environments. The main goal of dimensional modeling is to make it simple for users to query and analyze large datasets.

**Key Concepts of Dimensional Modeling**

* **Fact Tables**: These tables store quantitative data for analysis and are often at the center of a star schema or snowflake schema. Fact tables consist of measurable, numerical data like sales, revenue, or quantities.
* **Dimension Tables**: These tables contain descriptive information that provides context to the facts. They typically include attributes such as dates, product names, customer details, or locations. Dimension tables are denormalized to allow for easier querying.
* **Star Schema**: This is the simplest form of a dimensional model. It consists of a central fact table surrounded by dimension tables, resembling a star. Each dimension table is directly connected to the fact table.
* **Snowflake Schema**: This is a more complex form of a dimensional model. It involves normalizing dimension tables to reduce redundancy. In a snowflake schema, dimension tables are connected to one or more related tables, forming a shape similar to a snowflake.

**Why Star Schema**



In a start schema the fact table is generally going to be in the middle of that star and it is going to be surrounded by the dimension tables. The dimension tables are what filters the fact table. Ideally the fact and dimension tables share a one-to-many relationship. The fact table is going to be the “many” side of that relationship.

Star schema is often the most preferred schema in data warehousing and business intelligence environments, particularly for analytical purposes. Here are some reasons why it is so popular:

* **Simplicity**: The star schema is straightforward and easy to understand. Its simple structure, with a central fact table connected to dimension tables, makes it intuitive for users to navigate and query.
* **Query Performance**: Because of its denormalized structure, the star schema allows for faster query performance. The simplicity of the joins between the fact and dimension tables reduces the complexity and execution time of queries.
* **Ease of Maintenance**: The star schema is easier to maintain compared to more complex schemas like the snowflake schema. The denormalized structure means fewer tables and relationships to manage.
* **Optimized for Reporting**: The star schema is designed to facilitate efficient reporting and analysis. It aligns well with the way most business users think about their data, making it easier to create reports and dashboards.
* **Flexibility**: The star schema can accommodate various types of queries and analyses, making it a versatile choice for different analytical needs.
* **Data Integration**: It simplifies the integration of data from multiple sources. The clear separation between fact and dimension tables helps in consolidating data in a consistent manner.
* **Compatibility with BI Tools**: Many business intelligence tools, including Power BI, are optimized to work with star schemas. This compatibility ensures that you can leverage the full capabilities of these tools for data analysis and visualization.

**Conceptual Model, Logical Model and Physical Model**

These three types of data models represent different stages in the data modeling process, each serving a unique purpose in database design.

**Conceptual Model**

* **Purpose**: This model focuses on the high-level organization and relationships of data. It's abstract and often used during the initial planning phase to outline the overall structure of the database without going into technical details.
* **Components**: Entities (major objects or concepts), Relationships (how entities are related), and Attributes (details about each entity).
* **Example**: A conceptual model for a retail business might include entities like Customer, Product, and Order, with relationships indicating that Customers place Orders and Orders include Products.

Basically, this model is only going have the names of the tables in the data model we’re going to build:

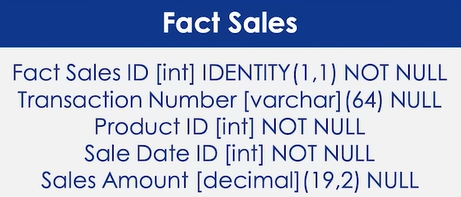
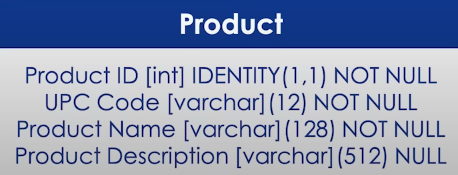
**Logical Model**

* **Purpose**: The logical model provides a more detailed view of the data, including specific entities, attributes, and relationships. It refines the conceptual model by adding more granularity and ensuring that the design adheres to business rules and data requirements.
* **Components**: Detailed Entities, Attributes, Relationships, Primary Keys, and Foreign Keys. This model is still independent of any specific database technology.
* **Example**: In the retail business example, the logical model might specify attributes like CustomerID, ProductName, OrderDate, etc., and define primary keys and foreign keys to establish relationships.

This is where we decide what columns these tables are going to have:

**Physical Model**

* **Purpose**: This model translates the logical model into a specific database implementation. It includes all the technical details required to create the actual database, such as table structures, indexes, constraints, and storage details.
* **Components**: Tables, Columns, Data Types, Indexes, Storage, and other technical specifications. This model is dependent on the chosen database management system (DBMS). The foreign key-primary key relationships are also defined here.
* **Example**: For the retail business, the physical model would detail the exact table structures, data types (e.g., VARCHAR, INT), indexes on CustomerID or ProductID, and how data will be stored and accessed in the database.

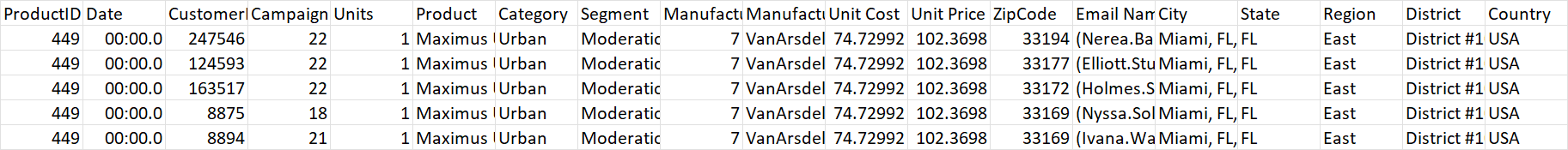
 

These are things that we normally store back in the database.

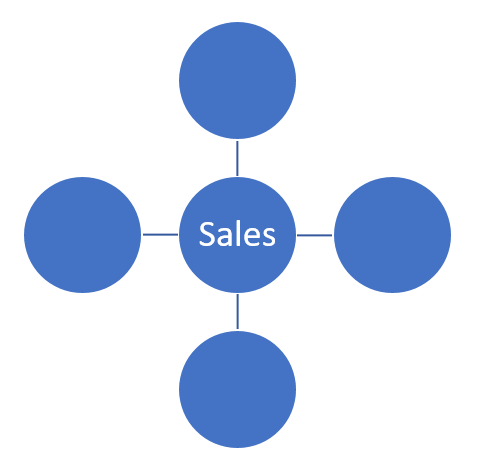
**Create a Conceptual Model**

Project files- C:\Users\sarathchandran\Power BI-YouTube\Data Modelling

Open the Sales.csv file. This is our source data. We’re going to create a conceptual model out of this dataset. Which means, we’re going to figure out what should be the fact table and what should be dimension tables. Here’s a snapshot of the dataset:



Since the data is about sales, let’s name the fact table Sales.



Now, let’s find out our dimension tables:

* There are several fields in the dataset that give us details regarding the product. So, we can create a dimension table to store all product information.
* The next dimension table could be for customer details.
* There are several columns that stores geographic data, like district, city, state etc. Hence, we can create a dimension table for geography.
* If you table has data/time fields it is always a good practice to create a dimension table for date attributes.
* There is one field in the dataset that stores campaign id. We can take this out from the dataset and make it a table of its own.

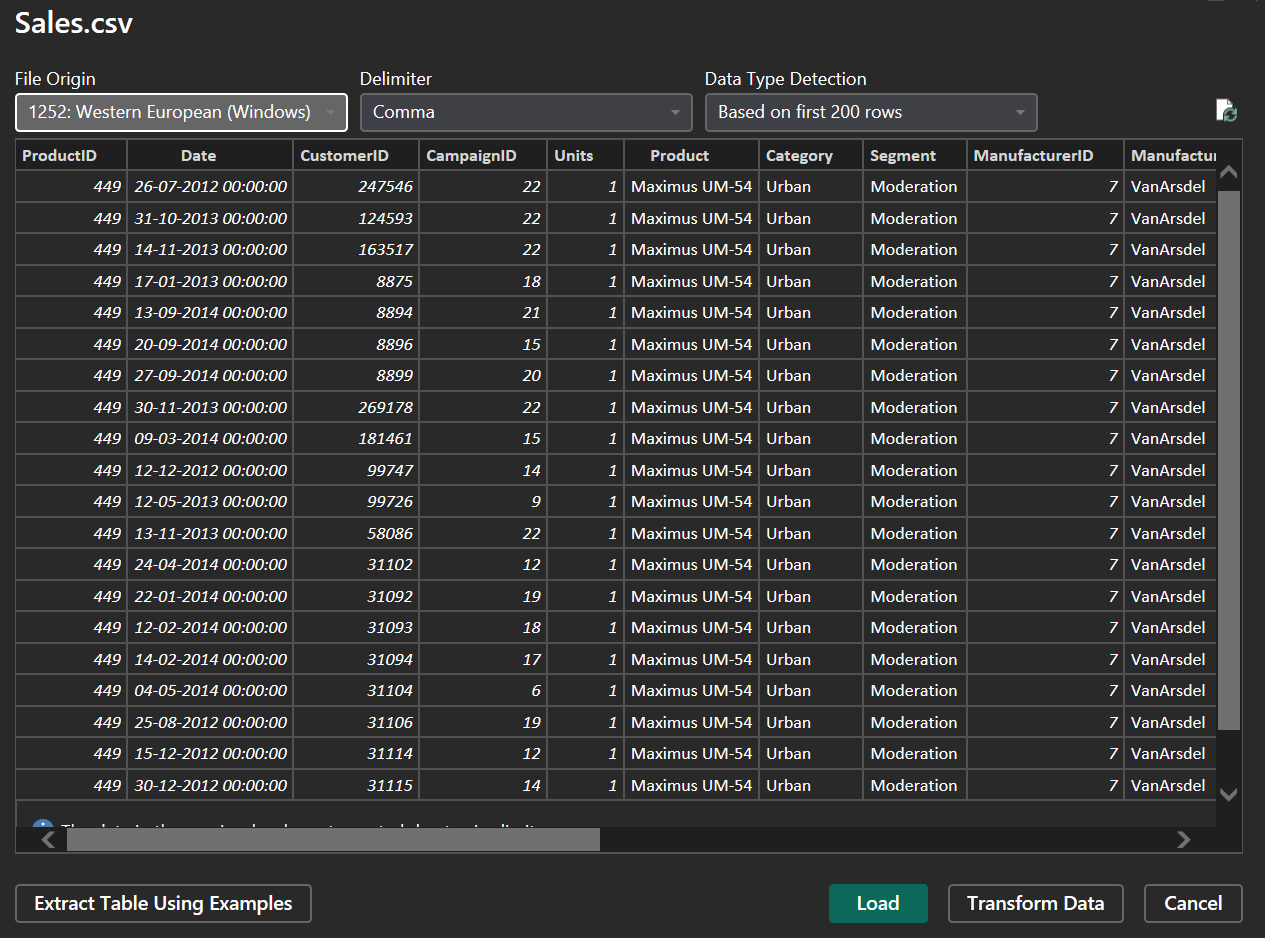
Out star schema now should look like this:



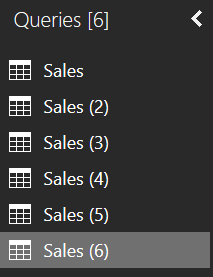
As we have conceived our conceptual model, let’s open Power BI and connect to the Sales.csv file.

In the navigator screen, you can preview the data and in that you will see the full date value. If you look at the snapshot of the Sales.csv file above, you can only see a bunch of zeros in the date field.

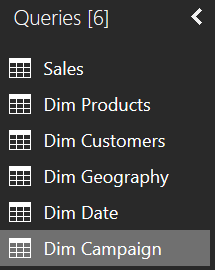
Once you’re connected to the source data, we will click on Transform to and go to Power Query.



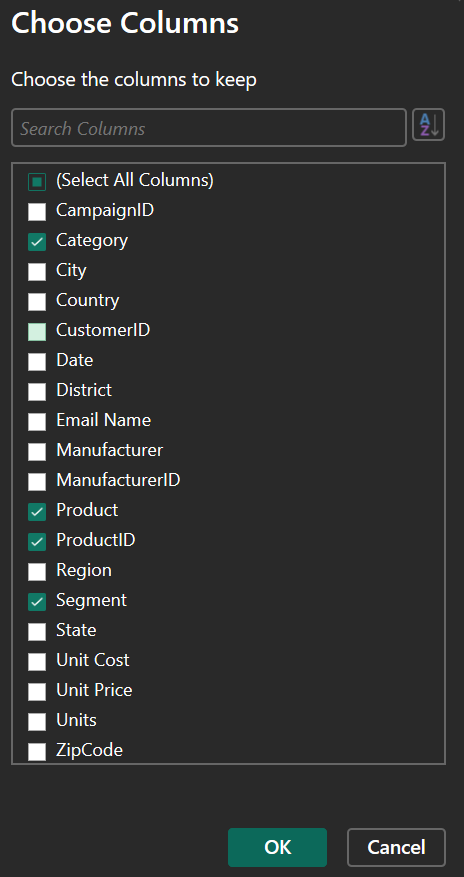
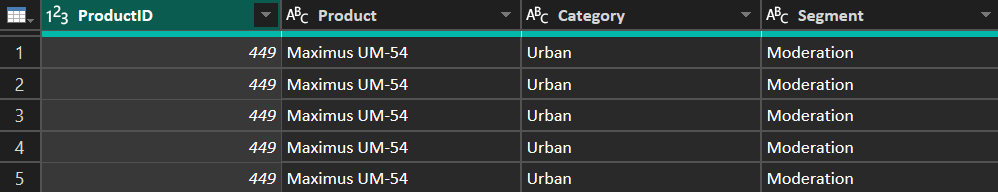
We will not be creating a dimension table for campaign. So, we will be having four dimension tables. Since we have 4 dimensional tables, we’ll create 4 copies of the Sales table and convert each copy in to a dimensional table that contains only specific information about customer, geography, product etc.



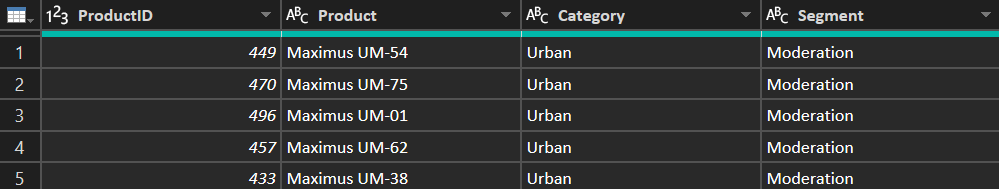
You can rename the queries appropriately.



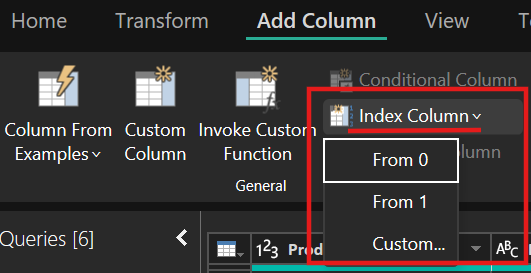
Take the Dim Products for example, we will keep only those columns that stores product details and remove the rest. Go to Choose columns and select Product, ProductID, Category and Segment:

 🡪 

As you can see there are duplicates in this table. Remember what we discussed earlier. The dimension tables are usually at the “One” side of the One-t-Many relationships. Which means, these tables are supposed to contain only unique rows. So, let’s remove the duplicates from Dim Products.

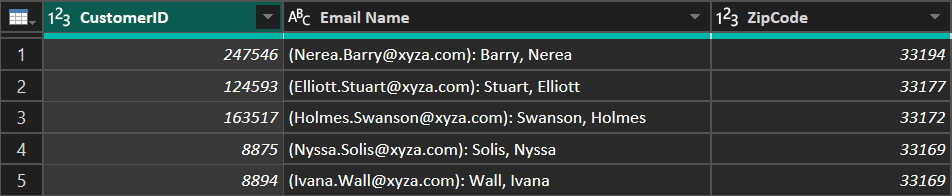


This table has a key column, which is ProductID, that can uniquely identify a row. You might run into situation where your table doesn’t have a unique key column like this. There is a feature in Power BI that helps you create column for this:

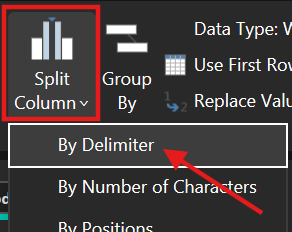


This will add an index column. The indexing can either start from 0 or 1. You can even create a custom index. But one thing to note here is that, when you create an index column you have to make sure that the fact table also contains this index. Only then we can build a connection with the fact table. This is a column that you created, which doesn’t exist in the original dataset. You can merge this column with the Sales table based on the Products name.

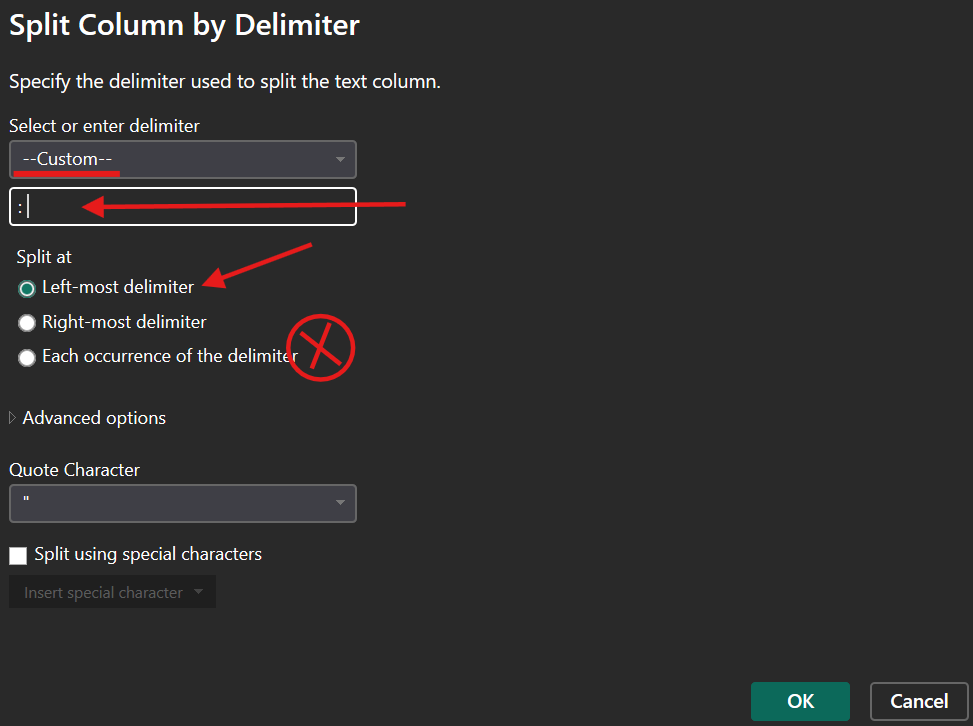
We will do the same thing in the Dim Customers query. We’ll only keep customer related data in this dimension table. These fields are Customer ID, Email Name and ZipCode (because we will relate Dim Customers with Dim Geography).



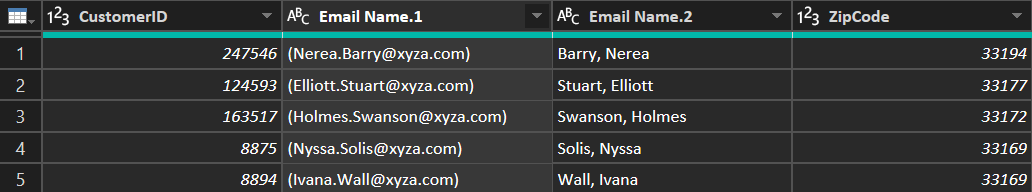
We now have a dimension table for customer detail that contain only unique rows. We need to perform an additional step. If you look at the Email Name column you can see that this column not only contains the email ids but also contains the names of the customers. The email ids and names are separated using a delimiter, semi-colon and space. We can split the column based off of that semi-colon and space. For this, select the EmailName column and click on Split Column in Home tab.



Select the Custom delimiter option and type in “: ” (semi-colon and space):

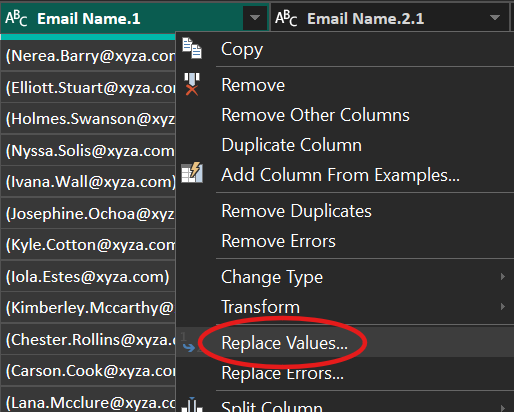
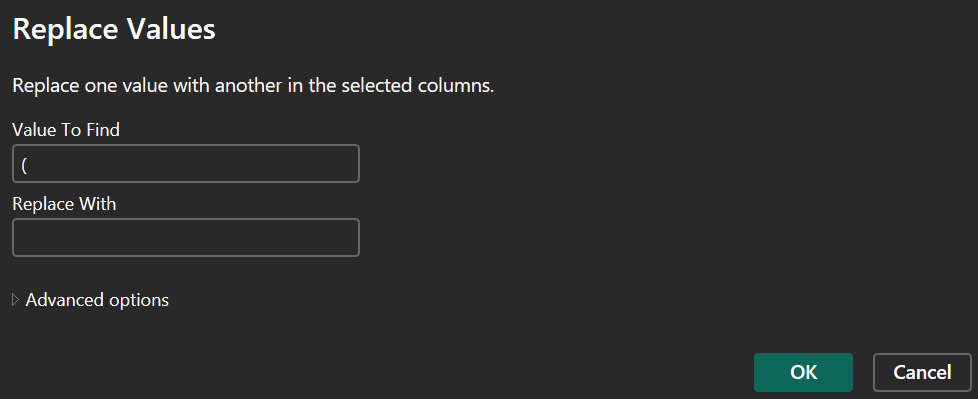


The Split at option will be defaulted to the third option. This will split the text every time it sees “: ”. We don’t want that. We want to split the text only at the first occurrence of “: ”.



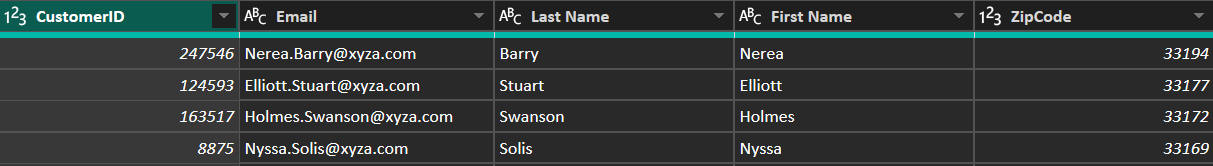
We have successfully split the email and name. If wanted you can split the name to first name and last name. The delimiter in this case would be comma and space “, ”. Since this delimiter option is not present in the Power BI, we will have to choose the Custom option and type “, ” like we did above.

We now need to get rid of the parenthesis around email. For this we can use the replace value option. We can search for the opening parenthesis “(“ and replace it with blank:

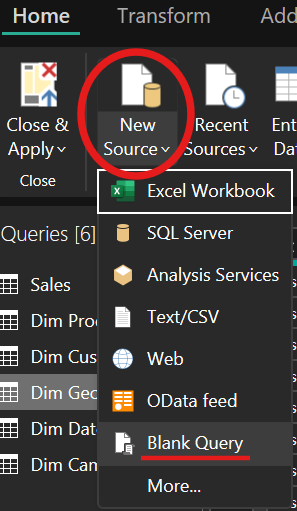
 🡪 

Similarly, we can replace the closing parenthesis “)” and replace it with nothing.

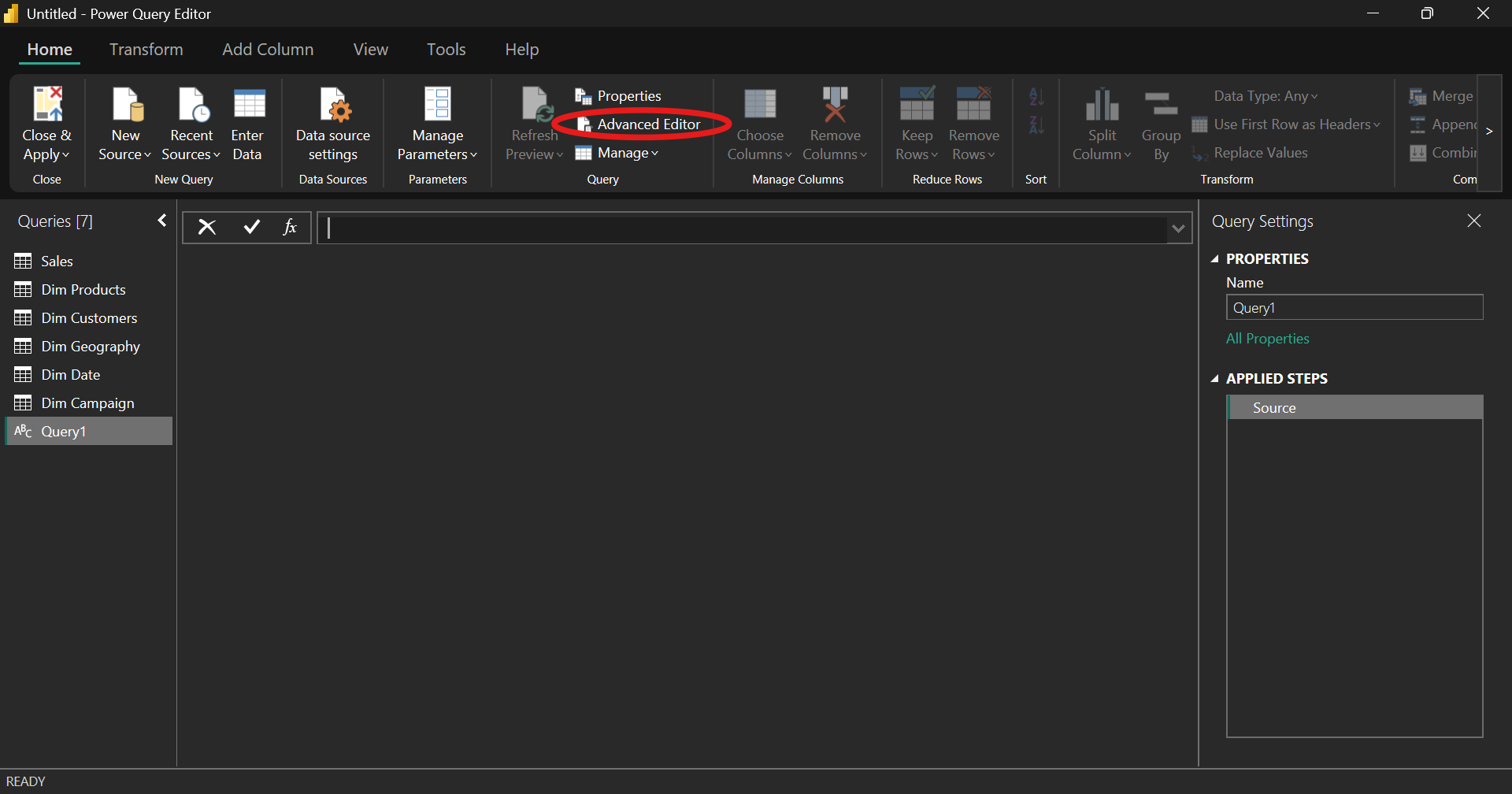
After properly renaming the columns, the table should look like this:



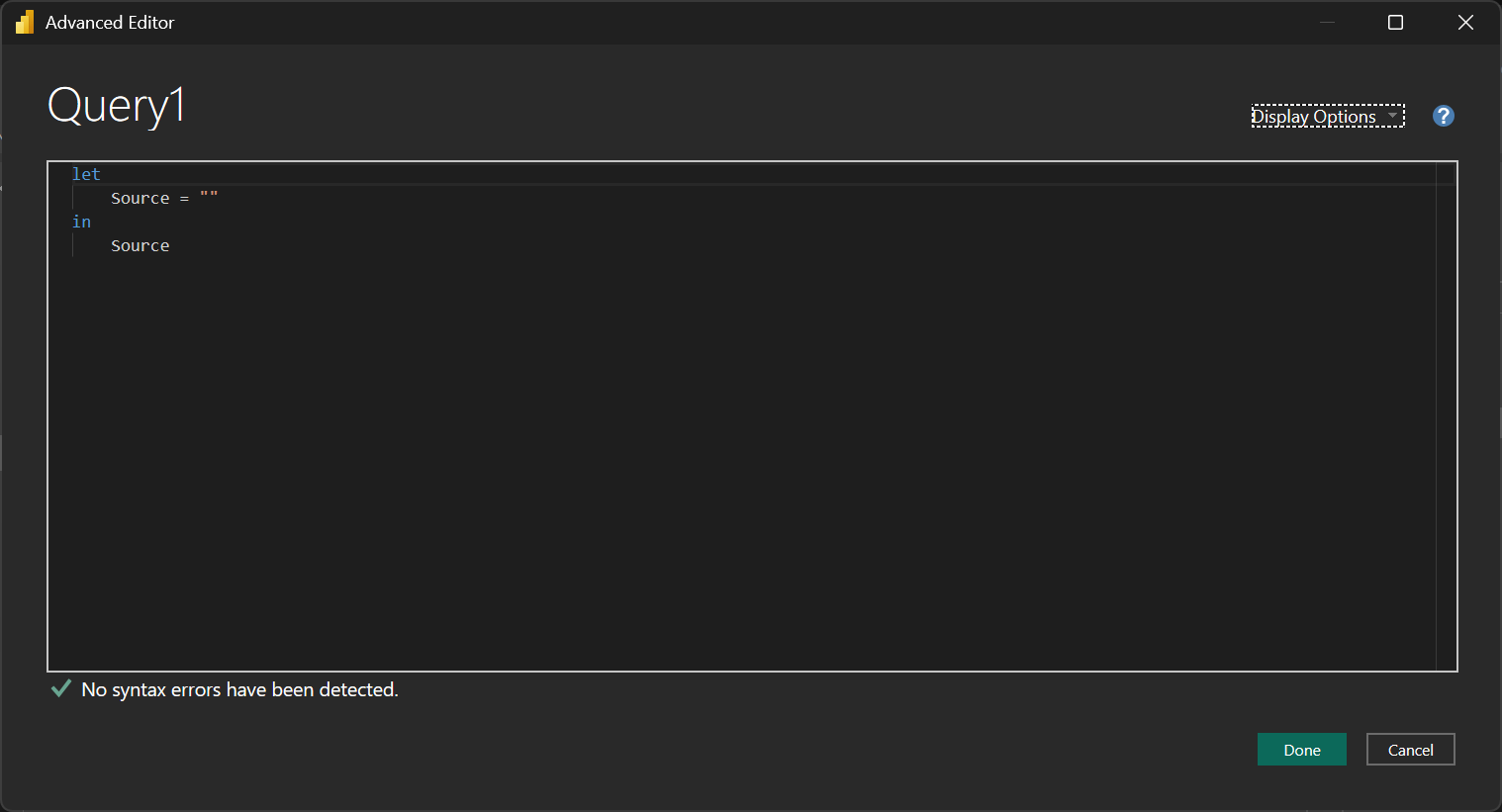
Similarly, we’ll create geography dimension table. We’ll be using ZipCode as the key for this one (You might run into situations where two different locations having the same ZipCode. In such situations you may have to add an index column). And finally, we’ll create a dimension table for date. For creating a table, we are actually following a different path. We’ll create a table using the M Code. So, in the Power Query editor, under the Home tab, in the New Query section, there is an option called New Source. From there select Blank Query:



This will create a blank query, no rows, no column, no data, nothing. From there select Advanced Editor:



This will open a new window:



Delete everything in there and paste the below code and click Done:

//Create Date Dimension

(StartDate as date, EndDate as date)=>

let

//Capture the date range from the parameters

StartDate = #date(Date.Year(StartDate), Date.Month(StartDate),

Date.Day(StartDate)),

EndDate = #date(Date.Year(EndDate), Date.Month(EndDate),

Date.Day(EndDate)),

//Get the number of dates that will be required for the table

GetDateCount = Duration.Days(EndDate - StartDate),

//Take the count of dates and turn it into a list of dates

GetDateList = List.Dates(StartDate, GetDateCount,

#duration(1,0,0,0)),

//Convert the list into a table

DateListToTable = Table.FromList(GetDateList,

Splitter.SplitByNothing(), {"Date"}, null, ExtraValues.Error),

//Create various date attributes from the date column

//Add Year Column

YearNumber = Table.AddColumn(DateListToTable, "Year",

each Date.Year([Date])),

//Add Quarter Column

QuarterNumber = Table.AddColumn(YearNumber , "Quarter",

each "Q" & Number.ToText(Date.QuarterOfYear([Date]))),

//Add Week Number Column

WeekNumber= Table.AddColumn(QuarterNumber , "Week Number",

each Date.WeekOfYear([Date])),

//Add Month Number Column

MonthNumber = Table.AddColumn(WeekNumber, "Month Number",

each Date.Month([Date])),

//Add Month Name Column

MonthName = Table.AddColumn(MonthNumber , "Month",

each Date.ToText([Date],"MMMM")),

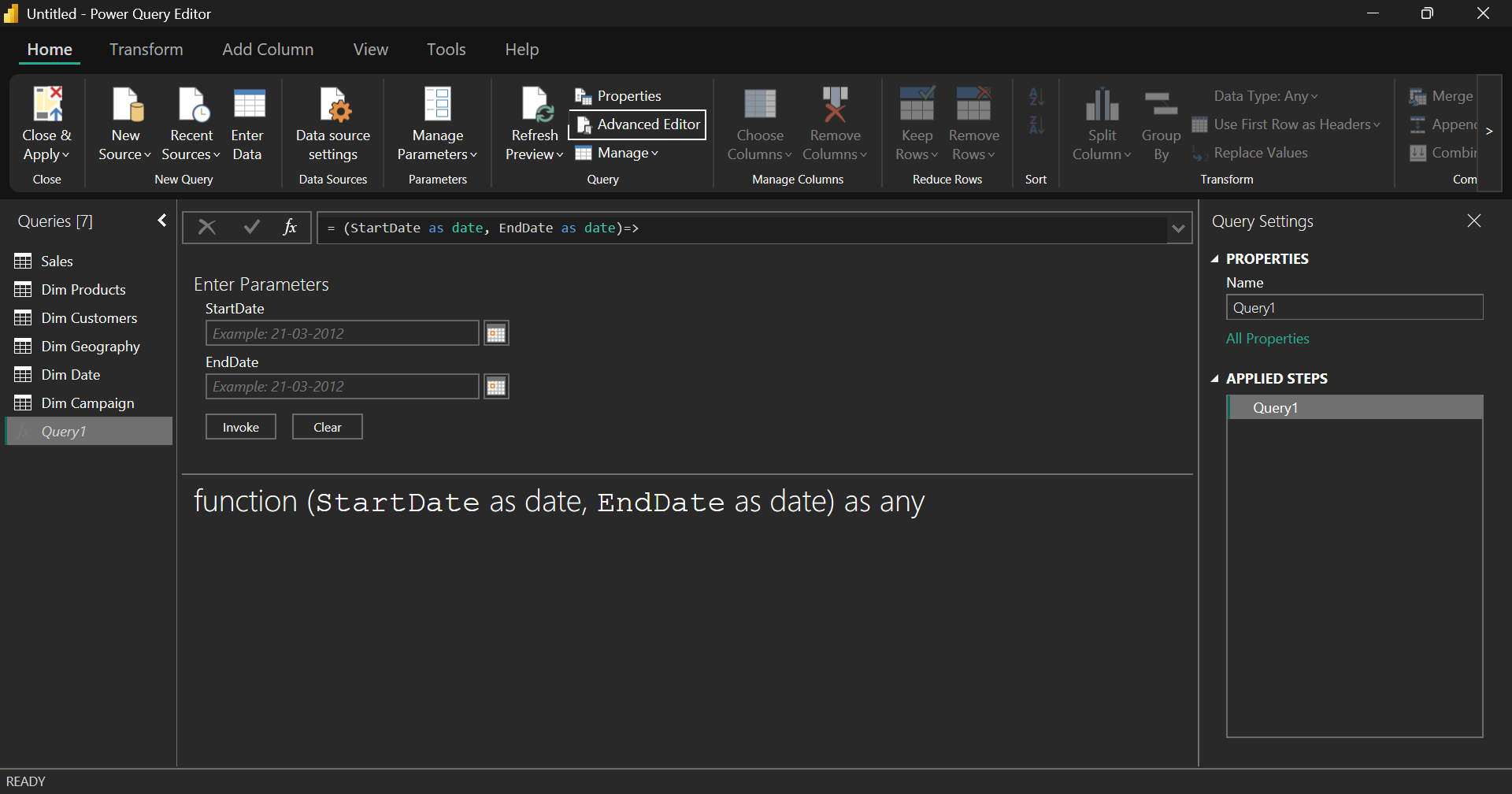
//Add Day of Week Column

DayOfWeek = Table.AddColumn(MonthName , "Day of Week",

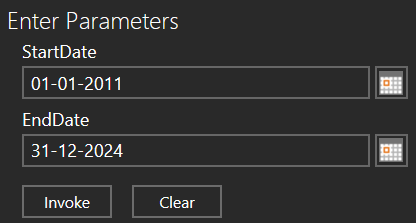
each Date.ToText([Date],"dddd"))

in

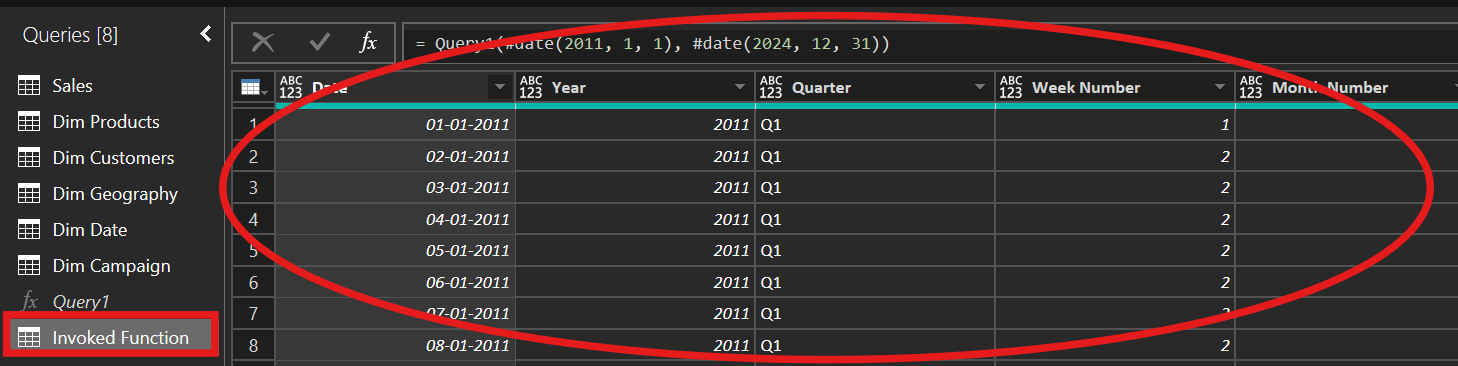
DayOfWeek



We don’t have our Date table yet. Here enter the lower and higher range of the dates you want. We want to create a date table with the date range from 1-1-2011 to 31-12-2024:



Click Invoke, and you have the date table:



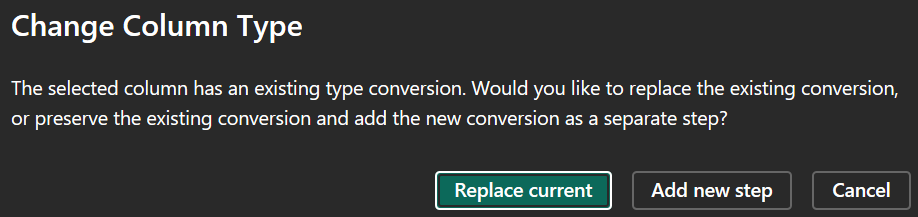
You can rename this table to Dim Date. Delete the Dim Date already existing there. As you can see the above Dim Date table require some formatting, like, the Date column is not in date format, the year column is not a whole number etc. So, let’s format the column correctly.



As we have created all the dimension tables, let’s go to the fact table, Sales. We need to fix something here as well.

Here we have to change the date column format to Date from Date/Time. We’re doing this because the in the Dim Date table, the date field is in Date format. By this doing we can avoid any potential problems that may arise due to inconsistent date formats.

While doing so, you may get this dialogue box:



The dialogue box you're seeing indicates that the column you are trying to change already has an existing type conversion applied to it. Here’s what the options mean:

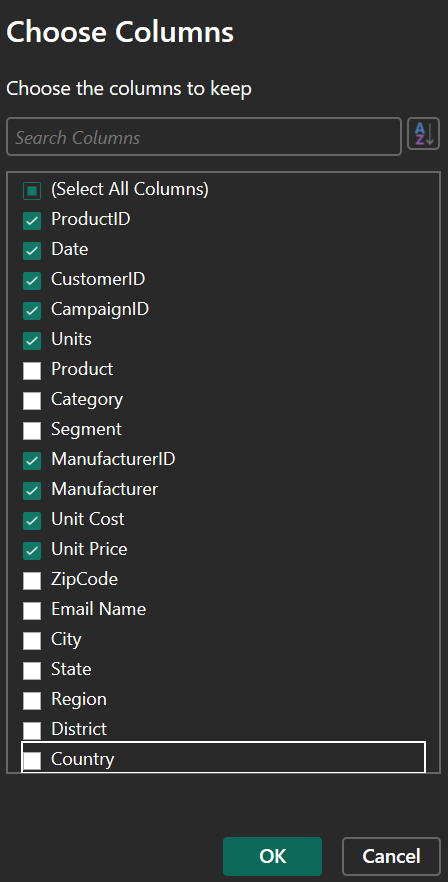
* **Replace current**: This option will overwrite the existing type conversion with your new conversion. It essentially means that the column will be directly converted to the new date format without retaining the previous conversion steps.
* **Add new step**: This option will preserve the existing conversion as a separate step and add the new conversion on top of it. This is useful if you want to keep the history of transformations applied to the column.

**When to Use Each Option**

* **Replace current**: Use this when you are sure that the previous conversion is no longer needed and you want a clean change to the new format.
* **Add new step**: Use this when you want to maintain a record of all transformations for better traceability or if you think you might need to revert to the previous state.

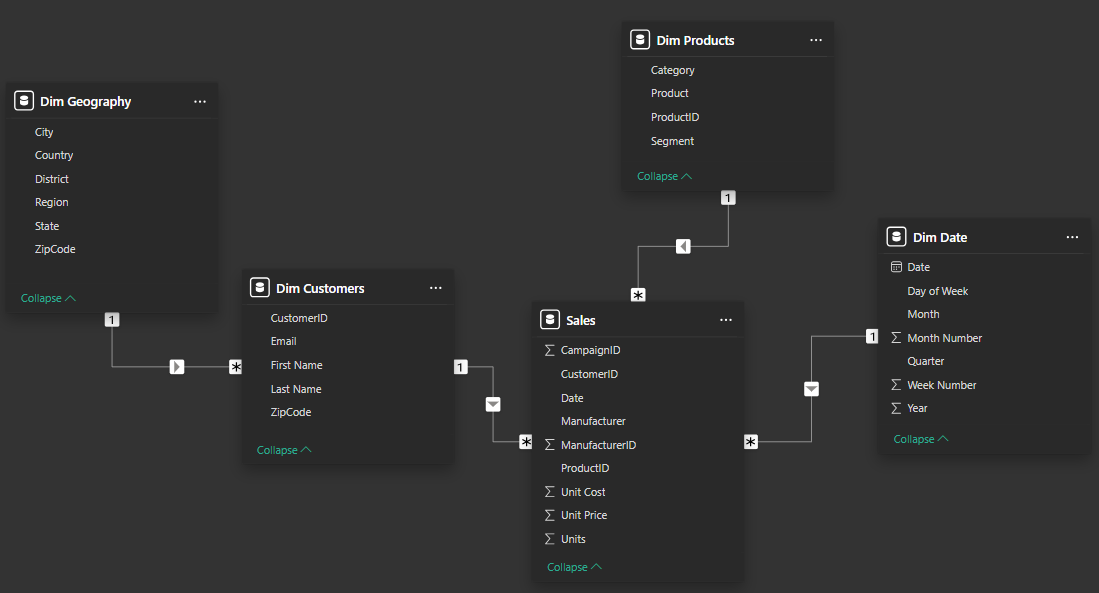
In general, if you're unsure, adding a new step is a safer option as it offers more flexibility and traceability. This can be helpful if you need to troubleshoot or revert to a previous state.

As we have fixed the formatting of the date column, we can remove the redundant fields from the Sales:



These are the columns we are going to keep in Sales table. In some cases, you may want to build a dimension table for storing manufacturer details. Here we’re not doing in anyway.

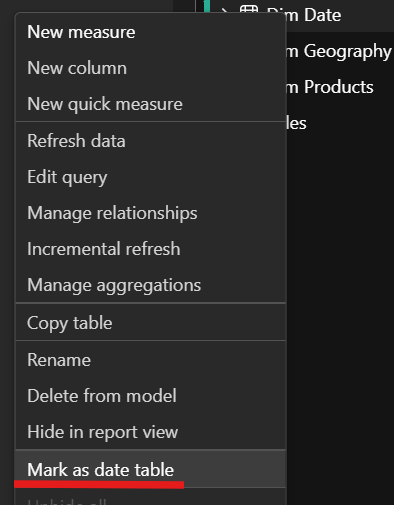
Let’s head over to the model view and see everything is correct:



If you look at the model, it is not a perfect star schema; we do have a little bit of snowflake. This is because the Dim Geography doesn’t share a direct relationship with Sales, rather it is related to Dim Customers. Note that, building a relationship based on a date value is generally not considered a best practice. It always best to build relationships based on integer values. It would have been much better if the Dim Date had an ID field.

There is a crucial step that you must do if you have a date table. If you have a date table, it is highly recommended to mark it as a date table. so, go to Dim Date, and mark it as a date table. Marking date table has several advantages:

* **Automatic Time Intelligence**: When you designate a date table in Power BI, it enables you to take full advantage of Power BI's time intelligence features. This includes the ability to use DAX (Data Analysis Expressions) functions that perform calculations over time periods, such as year-to-date, quarter-to-date, month-to-date, and period comparisons. These built-in functions rely on a properly marked date table to function correctly.
* **Consistent Time-Based Analysis**: A marked date table ensures that all date-related calculations and visualizations are based on the same date structure, providing consistency across your reports. This consistency is vital for accurate trend analysis, forecasting, and time-based metrics.
* **Hierarchies and Sorting**: Marking a table as a date table allows Power BI to recognize date hierarchies (like Year, Quarter, Month, Day) and automatically create them for use in visuals. It also ensures that date values are sorted correctly, which is essential for time-based charts and tables.
* **Performance Optimization**: Using a designated date table can optimize performance. Power BI can more efficiently handle time-based calculations and aggregations when it knows the specific structure and characteristics of the date data.
* **Simplified User Experience**: It provides a simplified user experience by reducing the need for manual configurations and custom calculations. Users can drag and drop date fields into visuals, and Power BI will automatically handle the time intelligence calculations based on the marked date table.

 🡪

