
Training Courseware

# AN OVERVIEW OF DAX FORMULAS FOR POWER BI

***Take the Fast Track Route to IT Learning***

Contact Us : 01527 836840
E-mail Us : info@fasttrackcomputertraining.co.uk
Our Website : www.fasttrackcomputertraining.co.uk

# CONTENTS



# POWER BI & DAX

DAX stands for **Data Analysis Expressions** i.e. such expressions or formulas that are used for data analysis and calculations. These expressions are a collection and combination of *functions, operators, and constants* that are evaluated as one formula to yield results (value or values). DAX formulas are very useful in BI tools like *Power BI* as they help data analysts to use the datasets they have to the fullest potential.
With the help of the DAX language, analysts can discover new ways to calculate data values they have and come up with fresh insights.
DAX formulas are used in calculated columns, measures, calculated tables, and row-level security.

Have a look at some key points about DAX which will help you understand the concept better.

- DAX is a functional language i.e. its complete code is always a function. An executable DAX expression may contain *conditional statements, nested functions, value references,* etc.
- DAX formulas have two primary data types; **Numeric** and **Non-numeric** or Others. The numeric data type includes *integers, decimals, currency,* etc. Whereas, the non-numeric consists of *strings and binary objects*.
- DAX expressions are evaluated from the innermost function going to the outermost one at the last. This makes formulating of a DAX formula important.

You can use values of mixed data types as inputs in a DAX formula and the conversion will take place automatically during execution of the formula. The output values will be converted into the data type you instructed for the DAX formula.

## Why should you learn DAX?

Learning DAX as a Power BI user is much like being an Excel user and discovering how to use formulas, You were able to structure your tables, add some charts and click the sum/average/… button (Σ) but suddenly you discover the world of VLOOKUP, IF functions etc. However, this comparison is not completely valid since, or at least in my opinion, Power BI is already an incredibly powerful tool even without DAX, as in contrast, anything beyond light use in Excel already requires formulas. The most import feature you will unlock is being able to select, join, filter, etc. data in a dynamic way. This means that dashboards & reports can take input from the users and use it to dynamically generate calculated columns, measures and tables.

## DAX - **It Opens Up A Whole New World!** 

## Importance of DAX in Power BI

A user needs to have basic knowledge of Power BI Desktop to create a decent report with all the available data. But, if you want to level up and use advanced calculations in your Power BI reports, you need DAX.

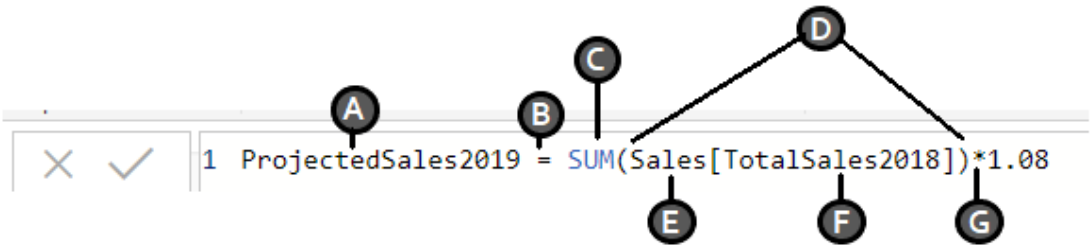
Let's say you want to make a visual to analyse growth percentage across different states of a country or need to compare year-over-year growth/sales. The data fields that you import in a data table are generally not enough to be used for such purposes.

For this, you need to make new measures using DAX language. In this way, you can create new measures, use them for creating exclusive visualizations, and have unique insights into data. With such unique insights into data, you can have fitting solutions for the business problems that you might miss with the usual way of analysis. Thus, DAX makes data analysis using Power BI, a smart and intelligent approach.

DAX formulas are used in calculated columns, measures, calculated tables, and row-level security.

# DAX FORMULA – SYNTAX

The initial and most crucial step in learning any language is to break it down into definitive elements and understand its elements. And that is why we study the syntax of a language. Below is an example of the DAX formula. We will understand this formula and its syntax elements with the help of this example.




In layman's terms this DAX formula commands the system to calculate the sum of the values in Total Sales 2018 and 1.08 (8% increase) and store the value in a new field or column known as Projected Sales 2019.

Each syntax element labelled in the image are outlined in the points below:

A: It is the **name** of the new measure (Projected Sales 2019).

B: It is the equals sign (=) which is an operator indicating the **start of the DAX formula** and equating the two sides.

C: It is the DAX function used to **add the values** of a given field (Total Sales 2018) from a table (Sales). The function used here is **SUM.**

D: The parenthesis () is used to **enclose and define arguments** in an expression. Every function must have at least one argument.

E: It is the **name of the table** from which a field or column is taken in the formula (Sales).

F: It is the **name of the field** from which the formula will use the values. For instance, the function SUM will apply itself on the values of the column or field [Total Sales 2018] of the table Sales.

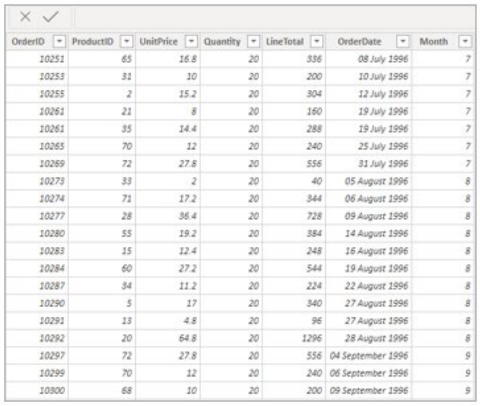
G: It is **another operator** used for multiplication. Although, the syntax elements from A to F constitutes the basic syntax of DAX.

# DAX CALCULATION TYPES

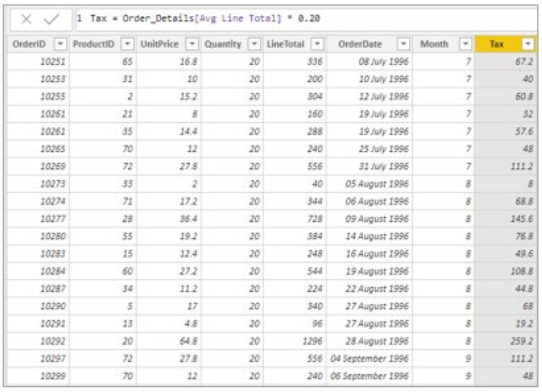
So, apparently, the DAX formulas can also be called as calculations as they calculate an input value and return a resultant value. You can create two types of expressions or calculations using DAX in Power BI; calculated columns and calculated measures.

- **Calculated Columns:** A calculated column is a column that you add to an existing table (in the model designer) and then create a DAX formula that defines the column's values.
  When a calculated column contains a valid DAX formula, values are calculated for each row as soon as the formula is entered.
  Values are then stored in the in-memory data model.

Creating a Calculated columns



| OrderID | ProductID | UnitPrice | Quantity | LineTotal | OrderDate | Month |
|---|---|---|---|---|---|---|
| 10251 | 65 | 16.8 | 20 | 336 | 08 July 1996 | 7 |
| 10253 | 31 | 10 | 20 | 200 | 10 July 1996 | 7 |
| 10255 | 2 | 15.2 | 20 | 304 | 12 July 1996 | 7 |
| 10261 | 21 | 8 | 20 | 160 | 19 July 1996 | 7 |
| 10261 | 35 | 14.4 | 20 | 288 | 19 July 1996 | 7 |
| 10265 | 70 | 12 | 20 | 240 | 25 July 1996 | 7 |
| 10269 | 72 | 27.8 | 20 | 556 | 31 July 1996 | 7 |
| 10273 | 33 | 2 | 20 | 40 | 05 August 1996 | 8 |
| 10274 | 71 | 17.2 | 20 | 344 | 06 August 1996 | 8 |
| 10277 | 28 | 36.4 | 20 | 728 | 09 August 1996 | 8 |
| 10280 | 55 | 19.2 | 20 | 384 | 14 August 1996 | 8 |
| 10283 | 15 | 12.4 | 20 | 248 | 16 August 1996 | 8 |
| 10284 | 60 | 27.2 | 20 | 544 | 19 August 1996 | 8 |
| 10287 | 34 | 11.2 | 20 | 224 | 22 August 1996 | 8 |
| 10290 | 5 | 17 | 20 | 340 | 27 August 1996 | 8 |
| 10291 | 13 | 4.8 | 20 | 96 | 27 August 1996 | 8 |
| 10292 | 20 | 64.8 | 20 | 1296 | 28 August 1996 | 8 |
| 10297 | 72 | 27.8 | 20 | 556 | 04 September 1996 | 9 |
| 10299 | 70 | 12 | 20 | 240 | 06 September 1996 | 9 |
| 10300 | 68 | 10 | 20 | 200 | 09 September 1996 | 9 |



```
1 Tax = Order_Details[Avg Line Total] * 0.20
```

| OrderID | ProductID | UnitPrice | Quantity | LineTotal | OrderDate | Month | Tax |
|---|---|---|---|---|---|---|---|
| 10251 | 65 | 16.8 | 20 | 336 | 08 July 1996 | 7 | 67.2 |
| 10253 | 31 | 10 | 20 | 200 | 10 July 1996 | 7 | 40 |
| 10255 | 2 | 15.2 | 20 | 304 | 12 July 1996 | 7 | 60.8 |
| 10261 | 21 | 8 | 20 | 160 | 19 July 1996 | 7 | 32 |
| 10261 | 35 | 14.4 | 20 | 288 | 19 July 1996 | 7 | 57.6 |
| 10265 | 70 | 12 | 20 | 240 | 25 July 1996 | 7 | 48 |
| 10269 | 72 | 27.8 | 20 | 556 | 31 July 1996 | 7 | 111.2 |
| 10273 | 33 | 2 | 20 | 40 | 05 August 1996 | 8 | 8 |
| 10274 | 71 | 17.2 | 20 | 344 | 06 August 1996 | 8 | 68.8 |
| 10277 | 28 | 36.4 | 20 | 728 | 09 August 1996 | 8 | 145.6 |
| 10280 | 55 | 19.2 | 20 | 384 | 14 August 1996 | 8 | 76.8 |
| 10283 | 15 | 12.4 | 20 | 248 | 16 August 1996 | 8 | 49.6 |
| 10284 | 60 | 27.2 | 20 | 544 | 19 August 1996 | 8 | 108.8 |
| 10287 | 34 | 11.2 | 20 | 224 | 22 August 1996 | 8 | 44.8 |
| 10290 | 5 | 17 | 20 | 340 | 27 August 1996 | 8 | 68 |
| 10291 | 13 | 4.8 | 20 | 96 | 27 August 1996 | 8 | 19.2 |
| 10292 | 20 | 64.8 | 20 | 1296 | 28 August 1996 | 8 | 259.2 |
| 10297 | 72 | 27.8 | 20 | 556 | 04 September 1996 | 9 | 111.2 |
| 10299 | 70 | 12 | 20 | 240 | 06 September 1996 | 9 | 48 |

- **Calculated Measures:** A calculated measure creates a field having aggregated values such a *sum, ratios, percentages, averages,* etc. Measures are dynamic calculation formulas where the results change depending on context.

  - Measures are used in reporting that support combining and filtering model data by using multiple attributes such as a Power BI report or Excel PivotTable or PivotChart. Measures are created by using the DAX formula bar in the model designer.
  - A formula in a measure can use standard aggregation functions automatically created by using the AutoSum feature (Excel), such as COUNT or SUM, or you can define your own formula by using the DAX formula bar. Named measures can be passed as an argument to other measures.

  - The result of a measure only becomes 'visible' when the measure is applied to a visual (Excel PivotTable or PivotChart, a Power BI report).

- Calculated tables: A calculated table is a computed object, based on a formula expression, derived from all or part of other tables in the same model. Instead of querying and loading values into your new table's columns from a data source, a DAX formula defines the table's values.

- Row-Level Security (RLS): With Row-Level Security, a DAX formula must evaluate to a Boolean TRUE/FALSE condition, defining which rows can be returned by the results of a query by members of a particular role.
  For example, for members of the Sales role, the Customers table with the following DAX formula: *= Customers[Country] = "USA".*[1]
  Members of the Sales role will only be able to view data for customers in the USA, and aggregates, such as SUM are returned only for customers in the USA. Row-level security is not available in Power Pivot in Excel.

[1] The full application of RLS is not covered in this course.

# DAX FUNCTIONS

A DAX function is a predefined formula which performs calculations on values provided to it in arguments. The arguments in a function need to be in a particular order and can be a *column reference, numbers, text, constants, another formula or function, or a logical value* such as TRUE or FALSE. Every function performs a particular operation on the values enclosed in an argument. You can use more than one argument in a DAX formula.

## Key Points about DAX Functions

Here are some unique facts about DAX functions that you must know in order to understand them better:

- Any DAX function always refers to a complete column/field or a table. It will never refer to individual values. If you want to use the functions on separate values within a column, you need to apply filters in a DAX formula.
- DAX functions provide the flexibility to create a formula that is applied on a row-by-row basis. The calculations or formulas get applied as per the context of the values in each row.
- In some cases, DAX functions return a full table which can be used in other DAX formulas that need a complete set of values. However, you cannot display this table's contents.
- DAX functions have a category known as time intelligence functions. Such functions are used to calculate time/date ranges and periods.

## Date and Time Functions

The date time functions carry out calculations on the date and time values. The data type of these values is always datetime data type. They include functions such as; DAY, MONTH and CALENDAR

## Time Intelligence Functions

The time-intelligence functions are used to evaluate values over a fixed period such as *days, weeks, months, quarter, years,* etc. You can specify a time period using these functions and compare two scenarios in your report. They include functions such as; DATESBETWEEN, DATESMTD, LASTDATE.

## Information Functions

The information functions are used to provide certain information on the data values contained in rows and columns. It evaluates the given condition in a function for the value given and return TRUE or FALSE. For instance, the function ISERROR will return TRUE if the value evaluated contains an error. They include functions such as; ISBLANK, CONTAINS, USERNAME.

## Logical Functions

The logical functions are used to evaluate an expression or argument logically and return TRUE or FALSE if the condition is met or not. They include functions such as; IF, AND, OR, TRUE.

## Mathematical and Trigonometric Functions

The mathematical and trig functions are used to perform all sorts of mathematical functions on the referred values. They include functions such as; ABS, DIVIDE, FLOOR.

## Statistical Functions

These functions carry out statistical and aggregation functions on data values in a DAX expression in Power BI. They include functions such as; AVERAGE, COUNT, MAX.

## Text Functions

The text functions in Power BI are very similar to the string functions of Excel. These functions evaluate string values. They include functions such as; BLANK, LEFT, UPPER.

## Parent-Child functions

The parent and child functions are used for data values that are a part of a parent-child hierarchy. They include functions such as; PATH, PATHCONTAINS.

## Table functions

The table functions in DAX formulas for Power BI are used to apply operations and conditions on entire tables. The output of table functions is used as inputs in other expressions or arguments in a DAX formula. The results of these functions retain the relationships between columns of that table. They include functions such as RELATEDTABLE.

## Filter Functions

DAX has powerful filter functions that are quite different from Excel functions. The lookup functions work by using tables and relationships, like a database. The filtering functions let you manipulate data context to create dynamic calculations.

Note − DAX filter functions that return a table do not add the table to the Data Model. The resulting table is used as an argument in another DAX function. That is, such DAX functions are used as nested functions with other DAX functions. They include functions such as; ALL, ALLEXCEPT, CALCULATE.

## Financial functions

Financial functions in DAX are used in formulas that perform financial calculations, such as net present value and rate of return. These functions are similar to financial functions used in Microsoft Excel. They include the functions XIRR and XNPV.

# RULES/GUIDELINES OF DAX FORMULAS

1. Never use the shortened CALCULATE syntax.
   Don't use [measure](filter) but CALCULATE( [measure], filter ) instead.
2. Always put a space before parenthesis '(' and ')'.
3. Always put a space before any operand and operator in an expression
4. If an expression is to be split into multiple rows, the operator is the first character in a new line.
5. A function call in an expression split across rows has to be always in a new row, preceded by an operator.
6. Never put a space between table name and column name.
7. Only use single quotes for table name if it is required.
8. Never use table names for measures.
9. Always use table names for column reference; Even when you define a calculated column within a table.
10. Always put a space before an argument, if it is in the same line.
11. Write a function inline only if it has a single argument that is not a function call.
12. Always put arguments on a new line if the function call has 2 or more arguments.
13. If the function is written on more lines:
    1. The opening parenthesis '(' is on the same line of the function call.
    2. The arguments are in new lines, indented 4 spaces from the beginning of the function call.
    3. The closing parenthesis is aligned with the beginning of the function call.
2. The comma separating two arguments is on the same line of the previous argument (no spaces before).
14. Definition of calculated column/measure is in the row before, including the assignment.
15. Use '=' to define a calculated column, Use ':=' to define a measure (Excel).

# ROW CONTEXT & FILTER CONTEXT - WHAT DO THE TERMS MEAN?

These two terms crop up a lot when referring to DAX formulas and can sound really mystifying; here I will try to explain the concepts as simply as possible.

## Row Context

When you use a column reference to retrieve the value of a column in a given row, you need a way to tell DAX which row to use, out of the table, to compute the value. In other words, you need a way to define the current row of a table. This concept of "current row" defines the Row Context.

You have a row context whenever you iterate a table, either explicitly (using an iterator) or implicitly (in a calculated column):

When you write an expression in a calculated column, the expression is evaluated for each row of the table, creating row context for each row.

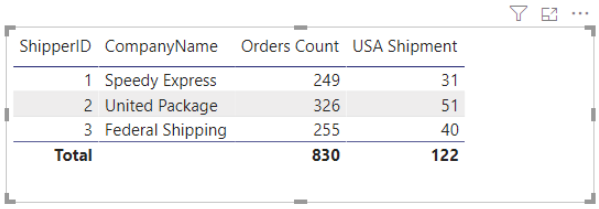
When you use an iterator like FILTER, SUMX, AVERAGEX, ADDCOLUMNS, or any one of the DAX functions that iterate over a table expression; it is "aware" of which row it is acting on in each stage of the formula evaluation.

## Filter Context

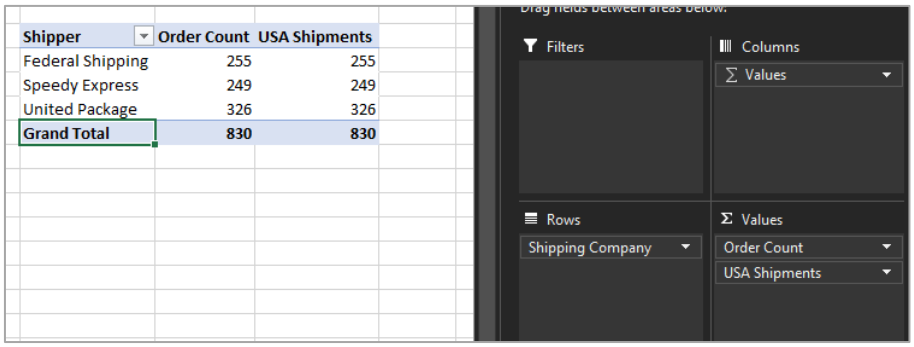
The filter context is the set of filters applied to the data model ***before*** the evaluation of a DAX expression starts. For example, when you use a measure in a table, it produces different results for each 'cell' because the same expression is evaluated over a different subset of the data.

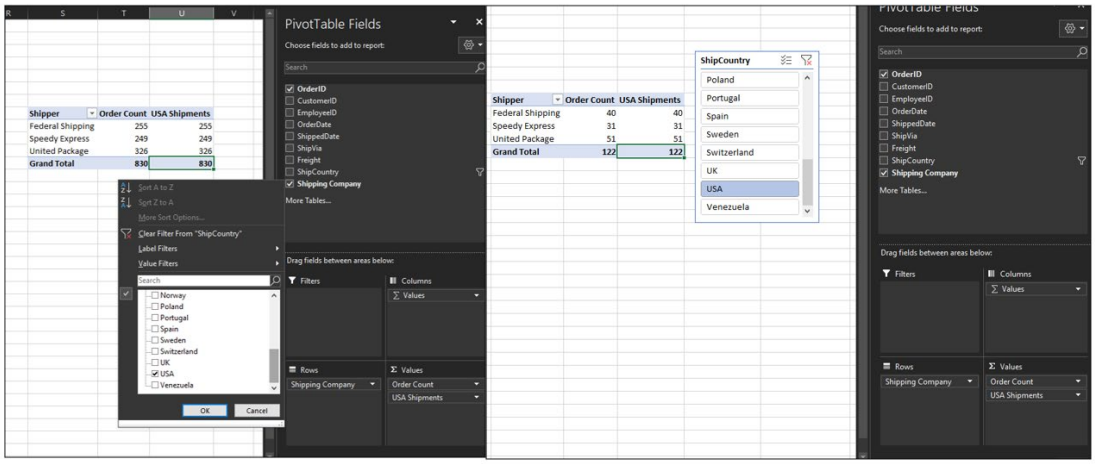
*USA Shipment = COUNTAX(FILTER(Orders,Orders[ShipCountry] = "USA"),Orders[OrderID])*

The above formula would allow me to create a new result that just looks at the rows where the ShipCountry is "USA", and summarizes just those rows. This means, unlike an excel pivot table I can have 2 different results sets in the same visual.



| ShipperID | CompanyName | Orders Count | USA Shipment |
|---|---|---|---|
| 1 | Speedy Express | 249 | 31 |
| 2 | United Package | 326 | 51 |
| 3 | Federal Shipping | 255 | 40 |
| **Total** | | **830** | **122** |




## Filter Propagation Concept

We need to understand the process of filter propagation to really help understand what is going on 'under the hood'.

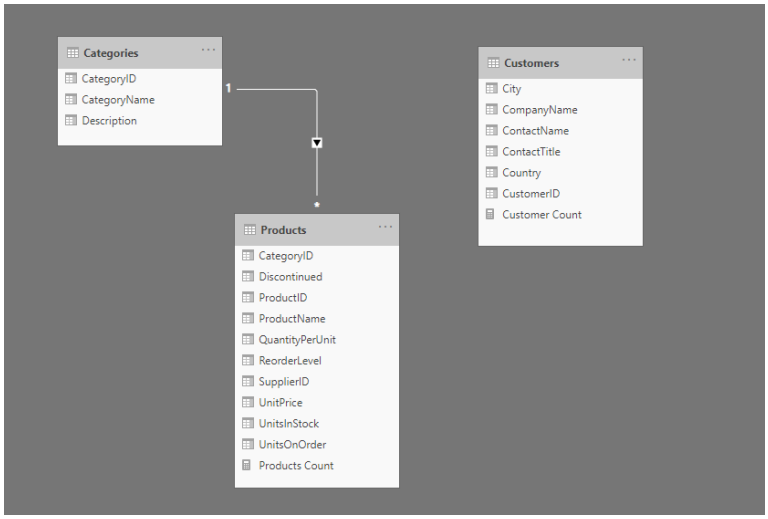
| CategoryName | Products Count | Customer Count |
|---|---|---|
| Beverages | 12 | 91 |
| Condiments | 12 | 91 |
| Confections | 13 | 91 |
| Dairy Products | 10 | 91 |
| Grains/Cereals | 7 | 91 |
| Meat/Poultry | 6 | 91 |
| Produce | 5 | 91 |
| Seafood | 12 | 91 |
| **Total** | **77** | **91** |

The result *Products Count* in the matrix is displaying a different value for each product category; but the result *Customer Count* is the same value for each category. The technical reason this is happening is because the row labels in the visual are "filtering" the products in Product table in the model *before* this measure is evaluated. But these same rows are not "filtering" the Customers table.

A matrix (or any) visual "filters" data and then displays subtotals for each row; that's what it is designed to do. This "filtering" is known as the *initial filter context*; initial because it is possible to change the filter context later by using the CALCULATE function (covered later). So, the initial filter context is the standard filtering coming from the visual before any possible modifications are applied from DAX formulas using CALCULATE.

# UNDERSTANDING THE FLOW OF THE INITIAL FILTER CONTEXT

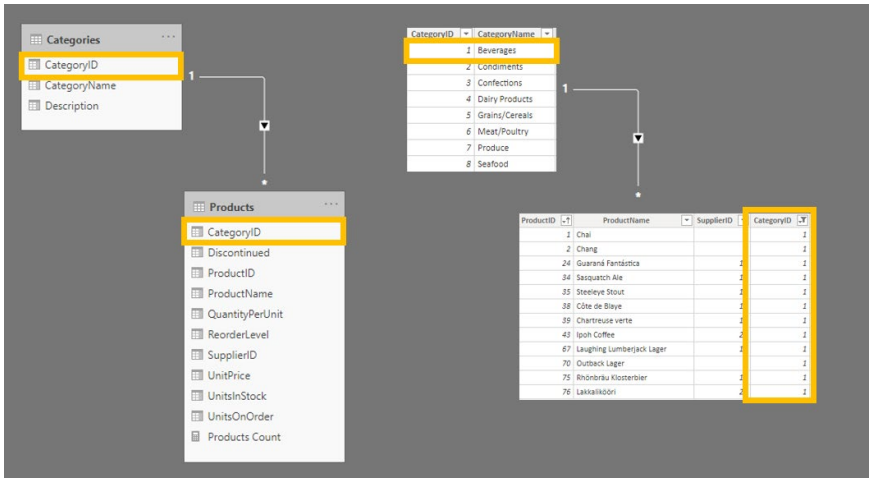
Once you know what initial filter context is, you will be able to mentally apply the following steps in your data model and track how the filters flow. Firstly, let's review the tables being used in the visual:




*In respect of the initial filter context coming from the visual being applied to the underlying tables there is just one table involved – Products, i.e. Categories[CategoryName]="Beverages" ; The Products table is filtered so that only rows in the table that are equal to Beverages remain.*

*The filter applied to the Products table is automatically propagated through the relationship between the tables, flowing downhill (from the One side to the Many).*

*Then the DAX formula is processed against the remaining rows in the Products table Products* $Count = COUNT(Products[ProductID])$




*The Customers table is not filtered; therefore, the result the result of the measure Customer Count = COUNT(Customers[CustomerID]) is the total count of customers in the Customers table.*

# VARIABLES IN DAX FORMULAS

DAX variables are a powerful feature within DAX that allow you to store and reuse intermediate results in your calculations, making your formulas more readable, efficient, and maintainable.

Here's how DAX variables work and how they can be used in Power Pivot:

Syntax: DAX variables are defined using the `VAR` keyword followed by a variable name, an equal sign (`=`), and an expression. The expression can be any valid DAX expression, including calculations, aggregations, and functions.

*VAR VariableName = Expression*
*RETURN ResultExpression*

Benefits:

1. Readability: DAX calculations can become complex and difficult to understand. By using variables, you can break down your calculations into smaller, meaningful steps. This makes it easier for you and others to comprehend the logic behind the calculations.
2. Efficiency: Without variables, you might need to repeat the same subexpression multiple times within a larger expression. This can impact performance because the subexpression is recalculated every time it appears. Variables store the result of a subexpression once and reuse it, reducing the need for redundant calculations.
3. Debugging: Variables allow you to inspect intermediate results while troubleshooting complex formulas. You can see what values are stored in variables at different steps of the calculation.
4. Scope: DAX variables are local to the formula where they are defined. They are not accessible outside of the expression in which they are declared. This means you can reuse variable names in different measures without conflicts.

To use variables in DAX:

1. Begin the formula by using the VAR keyword, followed by the variable name and its assigned value.
2. Define additional variables, if needed, using the same VAR syntax.
3. Use the RETURN statement to specify the final result of your DAX expression, using the variables you've defined.

Here's how you can use variables in DAX formulas, along with examples:

**Basic Variable Usage**: Variables in DAX are declared using the `VAR` keyword. They can store scalar values, table expressions, or even more complex calculations. Here's a basic example:

*Total Sales =*
*VAR TotalAmount = SUM('Sales'[Amount])*
*RETURN*
In this example, `TotalAmount` is a variable that stores the result of the `SUM` aggregation on the 'Sales' table's 'Amount' column. The `RETURN` statement returns the value stored in the variable.

**Using Variables for Reusability**: Variables are especially useful when you want to reuse a specific calculation within your formula. For instance, calculating a percentage based on a total:

```
Gross Margin % =
VAR TotalSales = SUM('Sales'[Amount])
VAR TotalCost = SUM('Sales'[Cost])
RETURN
   DIVIDE(TotalSales - TotalCost, TotalSales)
```

Here, both `TotalSales` and `TotalCost` are variables that store intermediate results, making the formula easier to understand and maintain.

**Using Variables with FILTER**: You can use variables with the `FILTER` function to create more complex calculations involving filtered subsets of data:

```
Average Revenue for Top Products =

VAR TopProducts =
   FILTER(
      'Products',
      'Products'[Revenue] > 100000
   )
RETURN
   AVERAGEX(TopProducts, 'Products'[Revenue])
```

In this example, the `TopProducts` variable stores a filtered subset of the 'Products' table containing products with revenue greater than 100,000. The `AVERAGEX` function then calculates the average revenue for this filtered subset.

Bear in mind that DAX variables are scoped within the formula in which they are defined. They can't be reused across different formulas or within different measures/calculations. Additionally, while DAX variables can enhance the readability of your formulas, it's important not to overuse them excessively, as too many variables might also make the formula harder to understand.

Using variables in DAX formulas can make your calculations more organized and easier to understand, especially when dealing with complex logic or repetitive calculations. They contribute to improved code readability and maintenance in your Power BI projects.

# POWER BI AND DATE DIMENSION TABLES

Power BI is an amazing business intelligence tool that gives us the ability to calculate many time-intelligent calculations based on the available underlying data. Few in-built functions allow the business users to calculate month-over-month, or month-to-date, etc. calculations just out of the box. The only requirement for Power BI to calculate such functions is to have a date dimension table in the Power BI data model on which it can make the calculations.

A date dimension table, also referred to as date table or calendar table can either be imported to the model, created in Power Query or built using DAX.

## Using Date Tables

Power BI Desktop works behind the scenes to automatically identify columns that represent dates, and then creates date hierarchies and other enabling metadata for your model, on your behalf. You can then use those built-in hierarchies when creating report features like visuals, tables, quick measures, slicers, and so on. Power BI Desktop does this by creating hidden tables on your behalf, which you can then use for your reports and DAX expressions.

## Using Power Query or DAX to build a date table: What is the difference?

The answer is that for many scenarios these are similar. So, it may make no difference to use Power Query or DAX for it. However, there is a key difference.

Power Query can fetch data from live web APIs. This functionality gives you the power to fetch public holidays live from an API. You cannot do this with DAX! Apart from this big difference, majority of other requirements can be done with both, you can write calculations in both M or DAX to get calendar columns as well as fiscal columns. In many scenarios public holidays plays an important role in analysing data. You would like to know how the sales was in holidays compared to other holidays and etc.

Why DAX? Most people find it easier to create a date table using DAX as this feels more like Excel.

## DAX Date Dimension Table

A date dimension written in DAX is created in the analysis layer directly within the report. The date dimension will recalculate a new date dimension upon report refresh after the queries of the report ETL layer are done processing. A date dimension can be created in a variety of ways in DAX, combining different functions to create and populate the dimension.

- DAX comes with useful built-in functions that can be leveraged to build the date dimension quickly.
- The date dimension written in DAX can be automatically created based on the range of dates in the data model.
- A date dimension calculated in DAX does not benefit from the compression of the ETL layer. It can result in more calculations and computing run in the data model than necessary.
- It can be confusing for other authors to understand and maintain the report when dimensions are created in different layers of the report.

# SUMMARY

Remember that DAX expressions can be complex, involving functions, filtering, and iterating over data. It's essential to understand DAX syntax and the underlying data model to create accurate and efficient DAX measures in Power BI.

Just like anything learning DAX will take time and a little perseverance too!

## What to know more?

Why not attend the Power BI Education Track training days: a set of 6 x 1-day training courses which will take even a novice user through the key components of managing data and creating reports using Power BI.