**What is DAX?**

DAX is a collection of functions, operators, and constants that can be used in a formula, or expression, to calculate and return one or more values. Stated more simply, DAX helps you create new information from data already in your model.

**Why is DAX so important?**

It’s quite easy to create a new Power BI Desktop file and import some data into it. You can even create reports that show valuable insights without using any DAX formulas at all. But, what if you need to analyze growth percentage across product categories and for different date ranges? Or, you need to calculate year-over-year growth compared to market trends? DAX formulas provide this capability and many other important capabilities as well. Learning how to create effective DAX formulas will help you get the most out of your data. When you get the information you need, you can begin to solve real business problems that affect your bottom line. This is the power in Power BI, and DAX will help you get there.

## Prerequisites

You might already be familiar with creating formulas in Microsoft Excel. That knowledge will be helpful in understanding DAX, but even if you have no experience with Excel formulas, the concepts described here will help you get started creating DAX formulas and solving real-world BI problems right away.

### Syntax

Before you create your own formulas, let’s take a look at DAX formula syntax. Syntax includes the various elements that make up a formula, or more simply, how the formula is written. For example, let’s look at a simple DAX formula for a measure.

This formula includes the following syntax elements:

**A.** The measure name **Total Sales**.

**B.** The equals sign operator (**=**) indicates the beginning of the formula. When calculated, it will return a result.

**C.** The DAX function **SUM** adds up all of the numbers in the **Sales[SalesAmount]** column. You’ll learn more about functions later.

**D.** Parenthesis **()** surround an expression containing one or more arguments. All functions require at least one argument. An argument passes a value to a function.

**E.** The referenced table **Sales**.

**F.** The referenced column **[SalesAmount]** in the Sales table. With this argument, the SUM function knows on which column to aggregate a SUM.

### Task: Create a measure formula

To complete this task, you’ll need to open the Contoso Sales Sample Power BI Desktop file.

1. In Report view, in the field list, right-click on the **Sales** table, and then click **New Measure**.
2. In the formula bar, replace **Measure** by typing a new measure name, **Previous Quarter Sales**.
3. After the equals sign, type the first few letters **CAL**, and then double-click the function you want to use. In this formula, you want to use the **CALCULATE** function.

You’ll use the CALCULATE function to filter the amounts we want to sum by an argument we pass to the CALCULATE function. This is what’s referred to as nesting functions. The CALCULATE function has at least two arguments. The first is the expression to be evaluated, and the second is a filter.

1. After the opening parenthesis **(** for the **CALCULATE** function, type **SUM** followed by another opening parenthesis **(**. Now we need to pass an argument to the SUM function.
2. Begin typing **Sal**, and then select **Sales[SalesAmount]**, followed by a closing parenthesis **)**. This is the first expression argument for our CALCULATE function.
3. Type a comma (**,**) followed by a space to specify the first filter, and then type **PREVIOUSQUARTER**. This will be our filter.

You’ll use the PREVIOUSQUARTER time intelligence function to filter SUM results by the previous quarter.

1. After the opening parenthesis **(** for the PREVIOUSQUARTER function, type **Calendar[DateKey]**.

The PREVIOUSQUARTER function has one argument, a column containing a contiguous range of dates. In our case, that's the DateKey column in the Calendar table.

1. Make sure both the arguments being passed to the PREVIOUSQUARTER function and the CALCULATE function are closed by typing two closing parenthesis **))**.

Your formula should now look like this:

**Previous Quarter Sales = CALCULATE(SUM(Sales[SalesAmount]), PREVIOUSQUARTER(Calendar[DateKey]))**

1. Click the checkmark  in the formula bar or press Enter to validate the formula and add it to the model.

You did it! You just created a measure using DAX, and not an easy one at that. What this formula will do is calculate the total sales for the previous quarter, depending on the filters applied in a report. For example, if we put SalesAmount and our new Previous Quarter Sales measure in a chart, and then added Year and QuarterOfYear as Slicers, we’d get something like this:

### Context

Context is one of the most important DAX concepts to understand. There are two types of context in DAX; row context and filter context. We’ll first look at row context.

**Row context**

Row context is most easily thought of as the current row. It applies whenever a formula has a function that applies filters to identify a single row in a table. The function will inherently apply a row context for each row of the table over which it is filtering. This type of row context most often applies to measures.

**Filter context**

Filter context is a little more difficult to understand than row context. You can most easily think of filter context as: One or more filters applied in a calculation that determines a result or value.

Filter context doesn’t exist in-place of row context; rather, it applies in addition to row context. For example, to further narrow down the values to include in a calculation, you can apply a filter context, which not only specifies the row context, but also specifies only a particular value (filter) in that row context.

Filter context is easily seen in your reports. For example, when you add TotalCost to a visualization, and then add Year and Region, you are defining a filter context that selects a subset of data based on a given year and region.

Why is filter context so important to DAX? Because while filter context can most easily be applied by adding fields to a visualization, filter context can also be applied in a DAX formula by defining a filter using functions such as ALL, RELATED, FILTER, CALCULATE, by relationships, and by other measures and columns. For example, let’s look at the following formula in a measure named Store Sales:

To better understand this formula, we can break it down, much like with other formulas.

This formula includes the following syntax elements:

**A.** The measure name **Store Sales**.

**B.** The equals sign operator (**=**) indicates the beginning of the formula.

**C.** The **CALCULATE** function evaluates an expression, as an argument, in a context that is modified by the specified filters.

**D.** Parenthesis **()** surround an expression containing one or more arguments.

**E.** A measure **[Total Sales]** in the same table as an expression. The Total Sales measure has the formula: =SUM(Sales[SalesAmount]).

**F.** A comma (**,**) separates the first expression argument from the filter argument.

**G.** The fully qualified referenced column, **Channel[ChannelName]**. This is our row context. Each row in this column specifies a channel: Store, Online, etc.

**H.** The particular value, **Store** as a filter. This is our filter context.

This formula ensures only sales values defined by the Total Sales measure are calculated only for rows in the Channel[ChannelName] column with the value “Store”, as a filter.

## 1. FILTER

The FILTER function is used to return a subset of a table or expression, as shown below.

**FILTER(<table>,<filter>)**

Let’s say that you want to get a count of items sold at the premium level, which you define as anything over $100. We will use the COUNTROWS function, which counts the number of rows in the specified table, along with the FILTER function to accomplish this:

**Count of sales orders over 100 = COUNTROWS(FILTER('Sales', 'Sales'[Sales] > 100))**

The first parameter, 'Sales', identifies a table or an expression that results in a table. The second parameter, 'Sales'[Sales] > 100, represents a Boolean, or true/false expression that is evaluated for each row of the table. In this expression, we are passing the Sales table to the FILTER function and asking it to return any sales that are over $100. The FILTER function is never used as a standalone function, but is used in conjunction with other functions. In the example above, we use the FILTER function to return a subset and then count the results.

## 2. ALL

The ALL function is used to return all of the rows in a table, or values in a column, ignoring any filters that may have been applied.

**ALL(<table> or <column>)**

In the Report View above, we have a report with multiple cards and a page-level filter that excludes sales in Germany. We would like to keep this filter, but add a card visual that shows the total number of items sold, ignoring any filters placed on the rest of the report. The following expression that incorporates the ALL function can help you to achieve this:

**Count of all sales orders = COUNTROWS(ALL('Sales'))**

In this example, we pass the 'Sales' table to the ALL function, asking it to clear any filters that may have been placed on it. Like the FILTER function, the ALL function is not used standalone but in conjunction with other functions. In this case, we use the ALL function in conjunction with the COUNTROWS function to get a count of all sales records. The ALL function accepts either a table or a column and clears any filters that may have been placed on them.

## 3. RELATED

The RELATED function returns a related value from another table (example shown below).

**RELATED(<column>)**

So far, we’ve worked with functions that can help you to return a subset or clear any filters on a table or column. We would now like to filter our sales for only the United States, but don’t have all of the data we need in one table to accomplish this. Fortunately, we have the RELATED function, which we can use to retrieve values from one table to another through an established relationship. Given that there is a many-to-one relationship between the Sales table and the SalesGeography table, respectively, we can use the following expression that incorporates the RELATED function to return a count of sales orders for only the United States:

**Count of sales orders in the US = COUNTROWS(FILTER(ALL('Sales'), RELATED('SalesGeography'[Countries]) = "United States"))**

## 4. TOTALYTD ****/****TOTALQTD ****/****TOTALMTD

Time intelligence functions in DAX enable you to manipulate data using time periods, including days, months, quarters, and years, and then build and compare calculations over those periods.

**TOTALYTD(<expression>,<dates>[,<filter>][,<year\_end\_date>])**

Continuing from the examples above, let’s say that you would like to see the total sales to date for this year. The following expression that incorporates the TOTALYTD function can enable you to easily do this:

**Total sales this year = TOTALYTD(SUM('Sales'[Sales]), 'Dates'[Dates])**

The first parameter, 'Sales'[Sales], identifies the column that you would like to aggregate. This could also be an expression that returns a scalar, or singular value. The second parameter, 'Date'[Dates], is a column that contains dates. Time intelligence functions are immensely useful functions that eliminate the need for complex code in calculating aggregations over commonly used periods of time.

## 5. CALCULATE

The CALCULATE function evaluates an expression in a context that is modified by specific filters.

**CALCULATE(<expression>, <filter1>,<filter2>…)**

Let’s say you are now interested in tabulating all sales for all areas. While you could create some piecemeal expressions to accomplish this, you can easily and cleanly accomplish the same thing utilizing the CALCULATE function. The following example, which uses the CALCULATE function, can accomplish this:

**Sum of sales all countries = CALCULATE(SUM('Sales'[Sales]),ALL('SalesGeography'))**

The first parameter, SUM('Sales'[Sales]), identifies the column that you would like to aggregate. The second parameter, ALL('SalesGeography'), represents a Boolean that removes any filters that may have been placed on the SalesGeography table. Notice that this ignores the page-level filter that excludes sales in Germany. The CALCULATE function is one of the most powerful and useful functions in DAX. It is helpful to think of the CALCULATE function as a supercharged “IF” statement. A couple of rules apply to the CALCULATE function: The filter parameters cannot reference measures, and expressions cannot use any functions that scan or return a table. The CALCULATE function is typically used with aggregation functions, and although the filter parameters are optional, at least one is typically used.

Coming back to DAX, these functions will be put in in the Data view. There are 3 things in Power BI where you can use DAX –

* Calculated Columns
* Measures
* Tables

We will go through all of these to see how DAX functions in each. These DAX functions in Power BI will behave differently depending on where you use them.

### Calculated Columns

As the name may suggest, you can create new columns based on the data as needed. For example, there is no ‘net price’ column available in the *List\_Items*table. Only MRP & quantity are available. Let’s use DAX to calculate a new column. This will demonstrate how simple it is to get started with DAX. To do this, in the data view, click on the List\_Items table. Then to add a new calculated column, go to the Modeling Tab and select New Column. A new blank column will be created and your cursor will be in the DAX editor (similar to the formula bar in Excel).

Pressing tab will type it out for you! Next, multiply by the quantity column and press enter. Once done, your ‘formula’ or expression should look like this:

Price = List\_Items[MRP]\*List\_Items[Qty]

In the data, each row will now have the respective calculated price.

Calculated columns are easy to understand and use, but take up more storage space on disk, especially if there are more than a few billion rows of data. This is because all the rows will have to be calculated every time the data is refreshed.

### Measures

Measures allow you to perform a calculation, without actually adding to the data. This is very helpful for reports; where the price can be shown, without needing a whole new column to store it in. Just one major difference is that measures have to be told exactly what to calculate on. If you type in a measure to calculate MRP \* Qty;

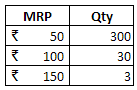
This is indicating that such a calculation is not accepted at all. This is because in order to calculate, we need to first specify what needs to be calculated –

1] Total of the MRP column **\*** Total of Qty column

OR

2] the MRP value in each row **\*** the corresponding Qty value in each row

This difference will be easier to grasp if you try to do both these calculations on this:



The answer to the first calculation will be ₹99,900 and the answer to the second will be ₹18,450!

Calculated columns just assumed and automatically used the second calculation. Using measures, you can calculate for each row by using:

MRPxQty = SUM**X**(List\_Items,List\_Items[MRP]\*List\_Items[Qty])

The ‘X’ after the sum means this calculation will be carried out for each row. This difference is officially called the Row Context.

### Tables

DAX functions in tabular model simply return entire tables, not just a value or a column of values. For example, to get a list of all the cities the company has customers in, use the function:

CitiesTouched = DISTINCT(Customers[City])

This will create table called ‘CitiesTouched’ and list out all the distinct or unique values in the City column in the Customers Table. This table can then be used independently as just like any other table. It will also be visible with the other tables in the tables section of Power BI.

## Power BI DAX Functions

Some related function in DAX are as follows. Since a lot of them are available in Excel, they function just like you would expect them to. Will elaborate more where necessary. Like in Excel, the required syntax / arguments / format function in DAX will be displayed in the application itself.

### Average

Find the average from a given set of values.  
**Example**– find out the average sales amount across all orders.  
AvgSales = AVERAGE(List\_Items[Price])

### Max

Find the maximum from a given set of values.  
**Example** – Find out the highest order.  
HighestSale = MAX(List\_Items[Price])

### Min

Find the minimum from a given set of values.  
**Example** – Find out the lowest order.  
LowestSale = MIN(List\_Items[Price])

### Count

Count any numerical data.  
**Example** – Count number of invoices generated.  
InvoiceVolume = COUNT(Invoices[Invoice No.])

### CountRows

Count the number of rows.  
**Example** – Count the number of rows in the ‘CitiesTouched’ table.  
RowsOfCitiesTouched = COUNTROWS(CitiesTouched)

### CountA

Count any kind of data; except blanks.  
Example – Count the customer codes to get the number customers the company has catered to.  
CustCount = COUNTA(Customers[Customer Code])

### Concatenate

Is used to join values in calculated columns. Use ConcatenateX if using in measures.  
**Example** – Concatenate the Product names and MRPs will give a unique code for all the price points at which each product is sold. This is helpful because the price of the product determines variant in our specific data model.  
ProMrp = CONCATENATE(List\_Items[Product],List\_Items[MRP])

### TotalYTD

Calculates the sum from the start of current Year To a given Date. It uses calendar year, not financial year. Calculates for each row, i.e. will return single values.  
**Example** – Calculate running / cumulative totals for the price column.  
CumiSales = TOTALYTD(SUM(List\_Items[Price]),Invoices[Date])

### Distinct

Returns unique values as a table. Using this in a calculated column will not work as it returns a whole table.  
**Example** – As used above, in the tables section.

### Filter

The DAX filter function returns a Table based on a criteria. Can be used to create a Sub- Table. Using this in a calculated column will not work as it returns a whole table.  
**Example** – Create a new table showing customers only from the USA.  
CustUSA = FILTER(Customers,Customers[Country]="USA")

### Calculate

The Calculate function in DAX is used to circumvent all existing filters applied to any table and calculate. It can also add new filters before calculating.  
**Example** – Create a new table showing total sales for each product. Filter the data to show only sales from a single country (marked red). Add a new column to show percentage of that country’s sales vis a vis total sales.  
PercentOFGlobal = SUM(List\_Items[Price]) / CALCULATE(SUM(List\_Items[Price]),ALL(List\_Items[Product]),ALL(Customers[Country]))

Cumulative Quantity :=

CALCULATE (

    SUM ( Transactions[Quantity] ),

    FILTER (

        ALL ( 'Date'[Date] ),

        'Date'[Date] <= MAX ( 'Date'[Date] )

    )

)

You can create a cumulative count using CALCULATE function, which lets us calculate the Running %measure:

Cumulative Count =

CALCULATE (

[Event Count],

FILTER (

ALL ( EventTable ),

[Duration\_Bucket] <= MAX ( EventTable[Duration\_Bucket] )

)

)

Now calculate the Running % measure using:

Running % =

DIVIDE (

[Cumulative Count],

CALCULATE ( [Event Count], ALL ( EventTable ) ),

BLANK ()

)

1. Create a calculated column for the date.
2. FullDate =
3. DATE ( 2016, 'Session'[Month of the Year], 1 )
4. Create a measure for 3 months moving average. You can change the number of months if you want.
5. Moving\_Average\_3\_Months =
6. CALCULATE (
7. AVERAGEX ( 'Session', 'Session'[Sessions] ),
8. DATESINPERIOD (
9. 'Session'[FullDate],
10. LASTDATE ( 'Session'[FullDate] ),
11. -3,
12. MONTH
13. )

)