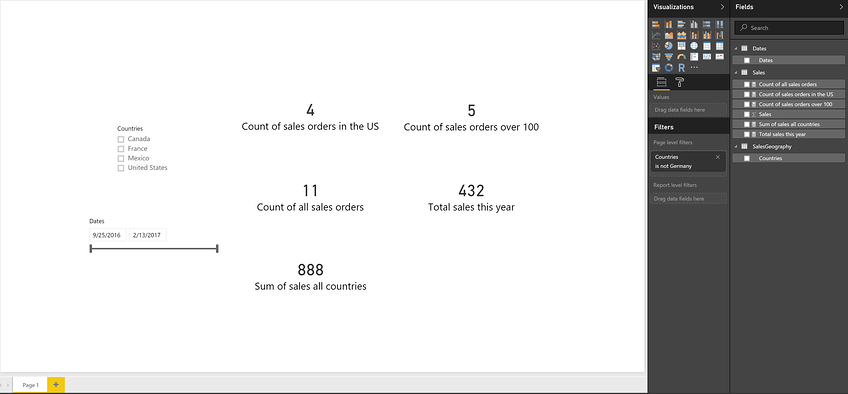
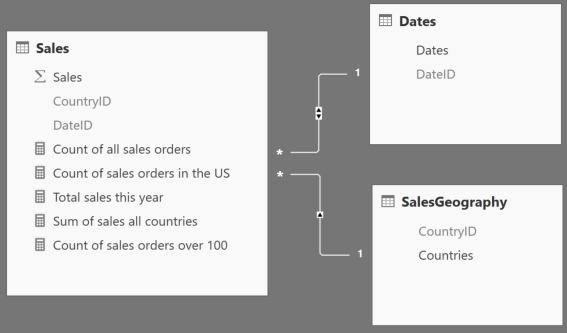
Working with Dax Functions

f you’ve ever run into this problem, then having some basic knowledge of DAX can enable you to create new views of the data in your model. As a relative newcomer to Power BI and DAX, I find myself using the 5 following DAX functions most often. Let’s work through some common business scenarios with the following Power BI file, [available here](https://cdn2.hubspot.net/hubfs/257922/BlueGranite%20DAX%20example.pbix) (note: must have Power BI to open):

**Report View**  


|  |  |  |
| --- | --- | --- |
| **Data View** | | |
| Date Table | Sales Table | SalesGeography Table |
|  |  |  |

**Relationships View**  


## **1.**[**FILTER**](https://msdn.microsoft.com/en-us/library/ee634966(v=sql.130).aspx)

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**](https://msdn.microsoft.com/en-us/library/ee634802(v=sql.130).aspx)

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**](https://msdn.microsoft.com/en-us/library/ee634202(v=sql.130).aspx)

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**](https://msdn.microsoft.com/en-us/library/ee634400(v=sql.130).aspx)**/**[**TOTALQTD**](https://msdn.microsoft.com/en-us/library/ee634579.aspx)**/**[**TOTALMTD**](https://msdn.microsoft.com/en-us/library/ee634560.aspx)

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**](https://msdn.microsoft.com/en-us/library/ee634825(v=sql.130).aspx)

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.

The answer is DAX (short for Data Analysis Expressions). DAX is used to bring some meaningful information hidden inside the raw data. In simple words, DAX is used for data manipulation.

There are two places where we write DAX:

* Calculated Column
* Calculated Measure

Let’s see what both of these stand for:

* **Calculated columns** are very similar to regular columns that we see in most datasets. The difference is that calculated columns are the result of our computations by using two or more columns or using columns from different tables. They can be used when we want to perform row-wise calculations
* **Calculated Measure**, on the other hand, is similar to a calculated column. However, they do not occupy any physical memory and their results cannot be seen in the form of a column. We usually use this when we want to perform dynamic computations on a group of rows or by grouping data together

**LOOKUP( )**

The LOOKUP function is pretty similar to **Vlookup** in Microsoft Excel.

The third table in our dataset contains the details of all the managers per region. Now here is where **LOOKUP** comes in hand. We can perform a lookup for the ‘Manager’ column in the ‘users’ table against the corresponding ‘Region’ column in the ‘orders’ table.

So how do we perform a lookup in Power BI? There are two things we need to remember:

1. We need a common column in order to perform a lookup
2. We also need unique values in at least one of the matching columns chosen from two different tables. The basic syntax for lookup is:

LOOKUPVALUE(Result Column Name, Search Column Name, Search Column value)

Putting this syntax using our dataset variables:

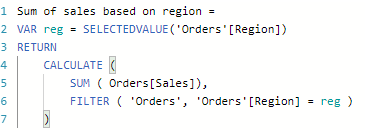
Manager = LOOKUPVALUE(Users[Manager],Users[Region],Orders[Region])

**FILTER( ) & CALCULATE( )**

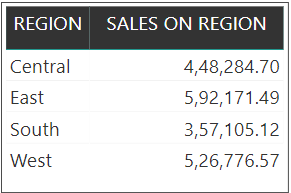
The DAX displayed below is similar to the *group by* function. It dynamically aggregates a column based on the filter. This is helpful when we are creating a table in Power BI dashboards and need to filter only one column (while the remaining column remains unaffected by the filter).

This DAX comes in handy where every column used in the table can have its own filter. Let’s take an example to understand how this works.

We want to calculate the sum of sales by region.  So first, the filter function divides the region column into north, south, east and west. Then, it calculates the sum of sales according to the segregation. *We are using a measure here since a particular region can have any number of rows involved within it.*

[](https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/05/110.png)

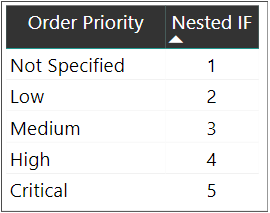
Here, we are declaring a variable **reg**which acts as a key for the filter. We can declare a variable using the keyword **VAR.**The **RETURN** keyword gives us the result of the calculation (sum of sales, in our example). The result we get from the calculated DAX is:

[](https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/05/25.png)

**Nested IF Condition**

Consider the orders table in our dataset. The ‘Order Priority’ column has five values under it. Let’s assume we need some integer values instead of the original values present in that column. The Nested IF statement is our friend here:

Nested IF = IF(Orders[Order Priority] ="Critical", 5, IF(Orders[Order Priority] ="High", 4, IF(Orders[Order Priority] = "Medium", 3, IF(Orders[Order Priority] = "Low", 2, IF(Orders[Order Priority] = "Not Specified", 1)))))

[](https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/05/32.png)

**Conditional Formatting**

Conditional formatting is one of the most commonly used features of Microsoft Excel. And we can leverage that inside Power BI as well!

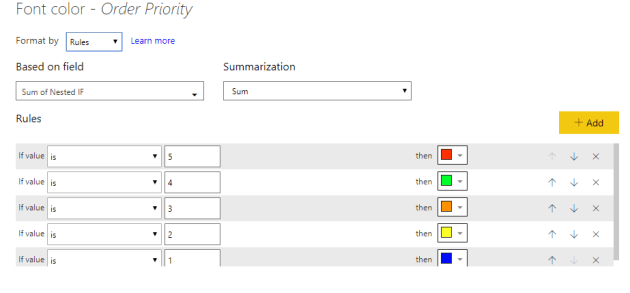
Conditional formatting, for those who haven’t used it before, is the ability to change the font color of a column based on a condition from another column. This can be done by creating a new column as per our condition and then using that column to set rules in the conditional formatting tab.

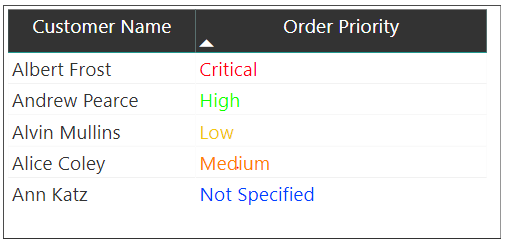
Let’s try this with an example from our Sample Superstore dataset.

We want to change the color of the values displayed in the ‘Order priority’ column of the orders table. For example, all ‘Critical’ values should be in red color, all ‘High’ values should be in green color, etc.

We can use the Nested IF column we created above since the conditions are already specified. Set rules in the conditional formatting tab – if the value of the column created above is 1, then the font color should be red If the value is 2, then it should orange, and so on.

Here’s an image to show how you can do it in Power BI:

**[](https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/05/42.png)**

[](https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/05/52.png)

**Splitting a String Based on Delimiters**

Another common Excel function we can use in Power BI:

customer\_split\_1 = PATHITEM(SUBSTITUTE(Orders[Customer Name]," ","|"),1)

The above **PATHITEM** function returns the resulting string. The **SUBSTITUTE**function replaces the delimiter specified with a particular character and fetches the corresponding word based on the value mentioned.

For example, let’s perform a split on the ‘Customer Name’ field. Here, the ” ” space will be the delimiter and we have replaced it with a pipeline “|”. We need only the first name of a customer, so we have specified 1. You can change this value and see what results you get.

Now let’s perform three split up’s like this and name them as customer\_split\_1, customer\_split\_2, customer\_split\_3 respectively.

**Fetching a Particular Letter from a Word**

What if we wanted to extract only a particular letter from a word? It sounds tricky, but it’s actually quite easy in Power BI:

customer\_initial\_1 = LEFT(Orders[customer\_split\_1],1)

We did it in just one line of code!

Let’s say a particular customer’s name is “Helen Stein” and the split is divided into two parts. We use the above DAX to get the first letter from the first split word (Helen). The **LEFT** function returns the number of characters by positioning to the start of the string. If we give 2 instead of 1, the above DAX will return “He” instead of “H”.

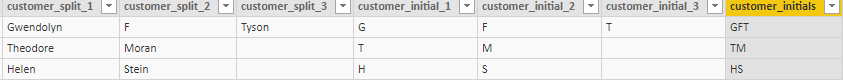
**Concatenating Strings**

We often find ourselves in situations where we need to combine two words together. To see how this works in Power BI, let’s assume we have multiple initials in our data.

We need to specify only the customer initials in the visualization rather than their whole name. This will help keep the table neat and compact:

customer\_initials = CONCATENATE(Orders[customer\_initial\_1],CONCATENATE(Orders[customer\_initial\_2],Orders[customer\_initial\_3]))

The concatenate function joins strings together. Here, we have performed a concatenation of three columns:

[](https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/05/62.png)

**WEEKDAY()**

The **WEEKDAY** function returns an integer number giving us the current day. Let’s figure out the day when a few orders were dispatched:

weekday = WEEKDAY(Orders[Ship Date],2)

The integer number displayed above specifies the start day:

* 1 – Start from Sunday = 1 and ends on Saturday = 7
* 2 – Start from Monday = 1 and ends on Sunday = 7
* 3 – Start from Monday = 0 and ends on Sunday = 6

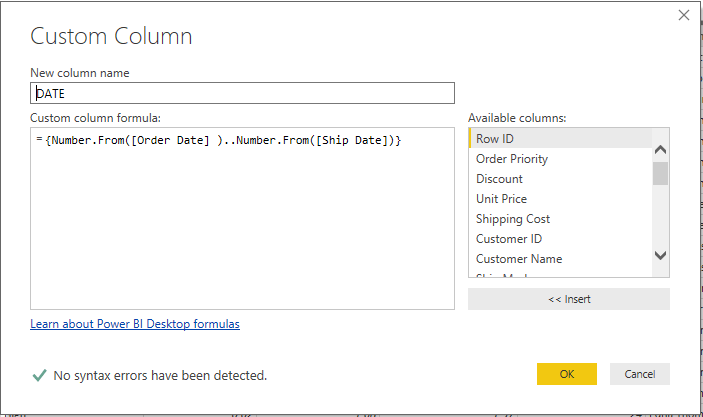
**DATE Split Up**

There are times when we need to un-pivot the date for certain projects. Here’s the good news – we can do this with the help of DAX and the query editor in Power BI.

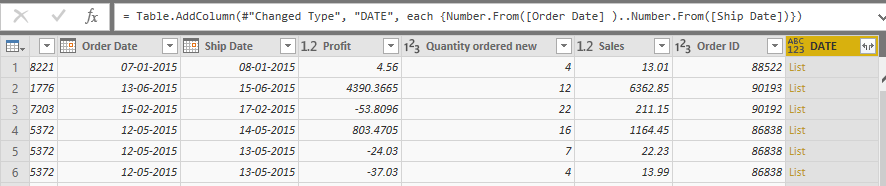
Now, we have two columns – order date and shipping date. We want to delist the dates lying between the two intervals. Let’s take the order date as 01-01-2015 and the shipping date as 03-01-2015. Since there’s a 3-day difference between them, this row will be listed three times.

We can do this through the query editor in Power BI. Select the ‘Custom Column’ option in the ‘Add Column’ Tab. The below window pops up and we can add a column name and use the DAX query:

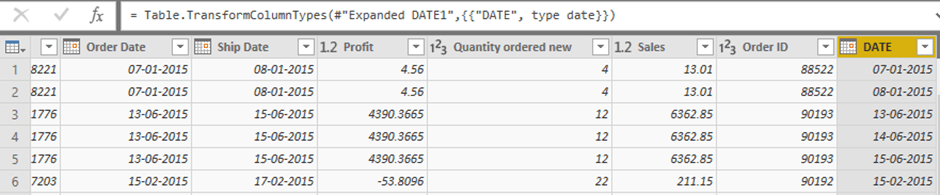
[](https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/05/72.png)

[](https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/05/82.png)

Next, find the DATE column in our dataset. Click on the small box inside the DATE column header and select the ‘Expand to new rows’ option:

[](https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/05/92.png)

Notice how the values are appearing in the form of whole numbers? Now, right-click on the column and change the data type to ‘Date’ format:

[](https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/05/102.png)

**Complex Aggregations Based on DATE**

Working with date data is quite a complicated challenge. There is a lot more to it than just splitting up the different days, months, etc.

In this section, we will work on an example of complex aggregation based on dates. For instance, what is the total number of hours listed for the next 2 weeks or the past 2 weeks?

We’ll switch our dataset for this section. You can download the new dataset, called ‘Weekcal’, from [Forecast](https://github.com/Tejyendloori/Power-Bi/blob/master/Weekcal.xlsx). This dataset is modeled on team time-allocation and a planning tool called Float. Float helps us assign tasks and calculate the estimated hours for team members.

There’s one caveat in Float we should be aware of. A week always starts from Monday in the Float tool. For example, if you want to calculate the assigned hours for next week starting from Thursday, you will find hours assigned from the most recent Monday to Friday and not from this Thursday to next Thursday.

Now, the dataset contains the below features:

* Start day – Start day of the task
* End date – End day of the task
* Hours/day – Number of hours spent on the task per day
* Task Name – Name of the task

Let’s assume we are interested in finding the number of hours allocated for next week. We can do this using the below logic:

*FORECA*ST\_1 WEEK = if (Forecast1[start\_date]>=TODAY()-(WEEKDAY(TODAY(),2)-1) && Forecast1[end\_date]<=TODAY()-(WEEKDAY(TODAY(),2)-1)+7,DATEDIFF(Forecast1[start\_date],Forecast1[end\_date],DAY)+1 \* Forecast1[hours/day],

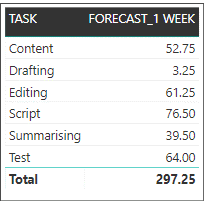
IF(Forecast1[start\_date] > TODAY()-(WEEKDAY(TODAY(),2)-1) ,0,IF(Forecast1[start\_date]<=TODAY()-(WEEKDAY(TODAY(),2)-1) && Forecast1[end\_date]<=TODAY()-(WEEKDAY(TODAY(),2)-1)+7 && Forecast1[end\_date]>=TODAY()-(WEEKDAY(TODAY(),2)-1),DATEDIFF(TODAY()-(WEEKDAY(TODAY(),2)-1),Forecast1[end\_date],DAY)\* Forecast1[hours/day],

IF(Forecast1[start\_date]>=TODAY()-(WEEKDAY(TODAY(),2)-1) && Forecast1[end\_date] >=TODAY()-(WEEKDAY(TODAY(),2)-1)+7,DATEDIFF(Forecast1[start\_date],TODAY()-(WEEKDAY(TODAY(),2)-1)+7,DAY)\* Forecast1[hours/day],

IF(Forecast1[start\_date]<=TODAY()-(WEEKDAY(TODAY(),2)-1)&& Forecast1[end\_date]>=TODAY()-(WEEKDAY(TODAY(),2)-1)+7,DATEDIFF(TODAY()-(WEEKDAY(TODAY(),2)-1),TODAY()-(WEEKDAY(TODAY(),2)-1)+7,DAY) \* Forecast1[hours/day],BLANK())))))

* **TODAY() – (WEEKDAY(TODAY(), 2) – 1** function returns the current Monday of the week
* **DATEDIFF** function returns the difference between two dates

Check out the table below to see the estimated hours for each task for the following week:

[](https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2019/05/112.png)

DAX Table-Valued Functions [<https://www.tutorialspoint.com/dax_functions/dax_functions_quick_guide.htm>

* DAX Filter Functions
* DAX Aggregation Functions
* DAX Time Intelligence Functions

Sum and Sumx

Example if the data in

|  |  |  |
| --- | --- | --- |
| **Product** | **Qty** | **Rate** |
| **a** | **5** | **10** |
| **b** | **3** | **20** |
| **c** | **4** | **30** |
|  |  |  |

Sum : Total = sum([Qty])\*sum([rate]) out put will be 720 [ total of qty \*total of Rate 12,60]

Syntax: = SUMX(<Table>, <expression> )

Sumx: Total 2= SUMX(Sheet1,Sheet1[Qty]\*Sheet1[Rate]) which means it will do cal row wise 5\*10.. for each and every row and then total example of Array function in Excel

Working with Calculation

The CALCULATE function is a very important function in Power BI as it is used to apply your own filters to data, that can add to existing filters or even replace them.

The following syntax is followed to write a CALCULATE function.

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

Example : Total Sales for East Region : Calculate(sum([salary]),[region]=”East”)… it is example o if function in Excel

Working with Filter ()

The filter function works the same way as CALCULATE; however, its major difference is that the FILTER functions are not mutable; It can only subset the data. FILTER is an expression that can be used in unison with an existing function such as CALCULATEr the SUMX, we have used Filter to FILTER the unit price which is greater than 300 to calculate sales of the product which are expensive (greater than 300)

Following syntax is followed to write a Filter function

(<expression>,<filter1>,<filter2>…)

Formula =Sumx(filter(table1,table1[unitprice]>300),table1([Qty]\*table1[price])

ALL Function

ALL function returns all the rows in a table or column, regardless of applied filters. Its simply ignores all filters and clears them. This function is very useful when we calculate aggregations, for s on all the rows in a table. This function is usually nested inside CALCULATE or AVERAGEX.

Syntax

ALL(<table>[<column>])

GrandTotal = CALCULATE(SUM('Demo Data'[Value]), ALL('Demo Data'[Category]))

