

# Leveraging DAX

**Data analysis expressions (DAX)** is a formula language that made its debut back in 2010 with the release of Power Pivot within Excel. Much of DAX is similar to Excel's functions, and therefore learning DAX is an easy transition for Excel users and power users. In fact, DAX is so similar to Excel that I have seen new students become comfortable with the language and begin writing DAX within minutes.

The goal of this chapter is to introduce you to DAX and give you the confidence to start exploring this language on your own.

.

Now, let's take a look at what is covered in this chapter:

- Building calculated columns
- Calculated measures – the basics
- Calculated measures – time intelligence

# Building calculated columns

There are many use cases for calculated columns, but the two most common are as follows:

- Descriptive attributes
- Concatenated key columns

Click on the Data View—this is located on the left side of the Power BI Desktop screen. Next, click on the customer table from the Fields list. Once the customer table has been selected, click New Column—this is found under the modeling ribbon, as shown in the following screenshot:

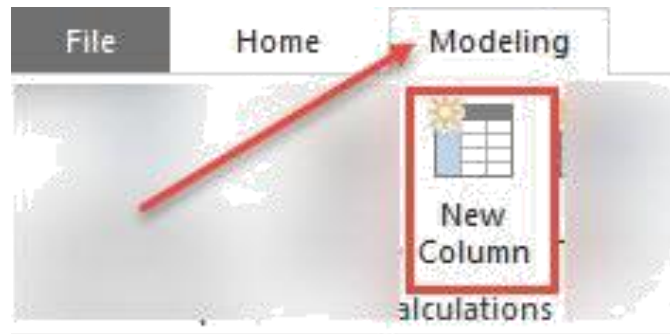


Figure 1- New column

You will now see the text `Column =` in the formula bar. First, name the new column by replacing the default text of `Column` with `Full Name`. Then, move your cursor to after the equals sign and type a single quote character. Immediately after typing the single quote character, a list of autocomplete options will appear preceding the formula bar. This is IntelliSense at work. The first option in this list is the name of the table you currently have selected—`Customer`. Click the `Tab` key and the name of the table will automatically be added to the formula bar, as shown in the following screenshot:

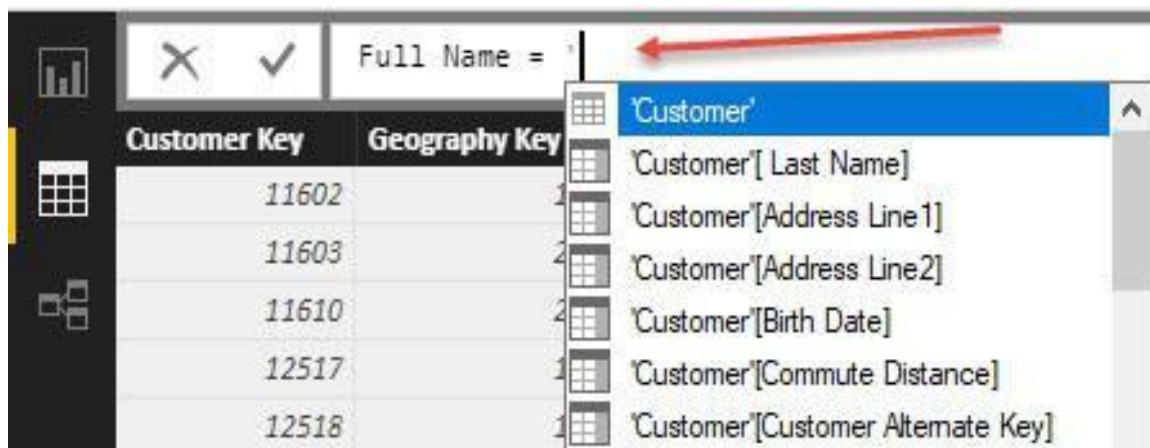


Figure 2-Adding name of the table

*At some point, you will inevitably discover that you can reference just the column name. As a best practice, we recommend always referencing both the table and column name anytime you use a column in your DAX code.*

Next, type an opening square bracket into the formula bar followed by a capital letter F, making `[F`. Once again, you will immediately be presented with autocomplete options. The list of

options has been limited to only columns that contain the letter f, and the first option available from the dropdown is First Name. Click *tab* to autocomplete. The formula bar should now contain the following formula:

```
Full Name = 'Customer'[First Name]
```

The next step is to add a space, followed by the last name. There are two options in DAX for combining string values. The first option is the `concatenate` function. Unfortunately, `concatenate` only accepts two parameters; therefore, if you have more than two parameters, your code will require multiple `concatenate` function calls. On the other hand, you also have the option of using the ampersand sign (&) to combine strings. The ampersand will first take both input parameters and convert them into strings. After this data conversion step, the two strings are then combined into one. Let's continue with the rest of the expression.

Remember to use the built-in autocomplete functionality to help you write code.

Next, add a space and the last name column. To add a space—or any string literal value for that matter—into a DAX formula, you will use quotes on both sides of the string. For example, " " inserts a space between the first and last name columns. The completed DAX formula will look like the following:

```
Full Name = 'Customer'[First Name] & " " & 'Customer'[Last Name]
```

# String functions – Month, Year

Now that you have completed your first calculated column, let's build a calculated column that stores the month–year value. The goal is to return a month–year column with the two-digit month and four-digit year separated by a dash, making "MM-YYYY". Let's build this calculation incrementally.

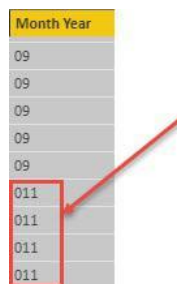
Select the Date (order) table and then click New Column from the modeling ribbon. Write the following code in the formula bar and then hit *Enter*:

```
Month Year = 'Date (Order)'[Month Number of Year]
```

As you begin validating the code, you will notice that this only returns the single-digit month with no leading zero. Your next attempt may look something like the following:

```
Month Year = "0" & 'Date (Order)'[Month Number of Year]
```

This will work for single-digit months; however, double-digit months will now return three digits. Take a look at the following screenshot:



Month Year
09
09
09
09
09
011
011
011
011

Figure 3-Displaying Month Year

To improve upon this and only return the two-digit month, you

can use the `RIGHT` function. The `RIGHT` function returns a specified number of characters from the right side of a string. Modify your existing DAX formula to look like the following:

```
Month Year = RIGHT("0" & 'Date (Order)'[Month Number of Year], 2)
```

*For a full list of text functions in DAX, please go to the following link:*  
<https://tinyurl.com/pbiqs-text>

The rest of this formula can be completed quite easily. First, to add a dash, the following DAX code can be used:

```
Month Year = RIGHT("0" & 'Date (Order)'[Month Number of Year], 2) & "-"
```

Complete the Month Year formula by combining the current string with the calendar year column:

```
RIGHT("0" & 'Date (Order)'[Month Number of Year], 2) & "-" &  
'Date (Order)'[Year])
```

# Format function – Month Year

As with any other language, you will find that there are usually multiple ways to do something. Now you are going to learn how to perform the calculation that we saw in the previous section using the FORMAT function. The FORMAT function allows you to take a number or date column and customize it in a number of ways. A side effect of the FORMAT function is that the resulting data type will be text. Let's perform the preceding calculation again, but this time using the FORMAT function.

Make sure you have the Date (order) table selected, and then click on Create a New Calculated Column by selecting New Column from the modeling ribbon. In the formula bar, write the following expression:

```
Month Year Format = FORMAT('Date (Order)'[Date], "MM-YYYY")
```

# Age calculation

Next, you are going to determine the age of each customer. The Customer table currently contains a column with the birth date for each customer. This column, along with the TODAY function and some DAX, will allow you to determine each customer's age. Your first attempt at this calculation may be to use the `DATEDIFF` function in a calculation that looks something like the following:

```
Customer Age = DATEDIFF('Customer'[Birth Date],  
TODAY(), YEAR)
```

The `TODAY` function returns the current date and time. The `DATEDIFF` function returns the count of the specified interval between two dates; however, it does not look at the day and month, and therefore does not always return the correct age for each customer.

Let's rewrite the previous DAX formula in a different way. In this example, you are going to learn how to use conditional logic and the `FORMAT` function to return the proper customer age. Please keep in mind, that there are many ways to perform this calculation.

Select the Customer Age column from the previous step and rewrite the formula to look like the following:



Customer Age =

IF(

```
    FORMAT('Customer'[Birth Date], "MMDD") <= FORMAT(TODAY(), "MMDD"), //Logical Test  
    DATEDIFF('Customer'[Birth Date], TODAY(), YEAR), //Result If True  
    DATEDIFF('Customer'[Birth Date], TODAY(), YEAR) -1) //Result If False
```

Figure 4-Select Customer age and rewrite the formula

*Formatting code is very important for readability and maintaining code. Power BI Desktop has a built-in functionality to help out with code formatting. When you type Shift + Enter to navigate down to the next line in your formula bar, your code will be indented automatically where applicable.*

When completed, the preceding code returns the correct age for each customer. The `FORMAT` function is used to return the two-digit month and two-digit day for each date (the birth date and today's date). Following the logical test portion of the `IF` statement are two expressions. The first expression is triggered if the logical test evaluates to `true`, and the second expression is triggered if the result of the test is `false`. Therefore, if the customer's month and day combo is less than or equal to today's month and day, then their birthday has already occurred this year, and the logical test will evaluate to `true`, which will trigger the first expression. If the customer's birthday has not yet occurred this year, then the second expression will execute.

# SWITCH() – age breakdown

Now that you have the customer's age, it's time to put each customer into an age bucket. For this example, there will be four separate age buckets:

- 18-34
- 35-44
- 45-54
- 55 +

The `SWITCH` function is preferable to the `IF` function when performing multiple logical tests in a single DAX formula. This is because the `SWITCH` function is easier to read and makes debugging code much easier.

With the Customer table selected, click `New Column` from the modeling ribbon. Type in the completed DAX formula for the following example:

```
Age Breakdown =  
SWITCH(TRUE(),  
    'Customer'[Customer Age] >= 55, "55 +", //If 55 or older then 55 +  
    'Customer'[Customer Age] >= 45, "45-54", //If 45-54 then 45-54  
    'Customer'[Customer Age] >= 35, "35-44", //If 35-44 then 35-44  
    "18-34") //ELSE, 18-34
```

Figure 5-Completed DAX formula

The preceding formula is very readable and understandable. There are three logical tests, and if a customer age does not evaluate to `true` on any of those logical tests, then that customer is automatically put into the 18-34 age bucket.

The astute reader may have noticed that the second and third logical tests do not have an upper range assigned. For example, the second test simply checks whether the customer's age is 45 or greater. Naturally, you may assume that a customer whose age is 75 would be incorrectly assigned to the 45–54 age bucket. However, once a row evaluates to `true`, it is no longer available for subsequent logical tests. Someone who is 75 would have evaluated to `true` on the first logical test (55 +) and would no longer be available for any further tests.

## Navigation functions – RELATED

It's finally time to create a relationship between the temperature table and internet sales table. The key on the Temperature table is a combination of the region name and the month number of the year. This column combination makes a single row unique in this table, as shown in the following screenshot:

Region	Month	MonthNumber	Key	Avg Temp	Temperature Range
Northeast	Jan	1	Northeast1	26.3	Cold
Northeast	Feb	2	Northeast2	25.4	Cold
Northeast	Mar	3	Northeast3	31.4	Cold
Northeast	Apr	4	Northeast4	48.1	Cool

Figure 6-Column combination that makes a single row unique

Unfortunately, neither of those two columns currently exist in the Internet Sales table. However, the Internet Sales table has a relationship to the Sales Territory table, and the Sales Territory table has the region. Therefore, you can determine the region for each sale by doing a simple `lookup` operation. Well, it should be that simple, but it's not quite that easy. Let's take a look at why.

Calculated columns do not automatically use the existing relationships in the data model. This is a unique characteristic of calculated columns; calculated measures automatically see and interact with all relationships in the data model. Now let's take a look at why this is important.

In the following screenshot, I have created a new column on the Internet Sales table and I am trying to return the region name from the Sales Territory table. Take a look at the following

screenshot:


```
Temperature Key =  
'Sales T
```

Figure 7-Sales Territory table

Note that there is no IntelliSense, and that the autocomplete functionality is unavailable as I type in "Sales Territory". The reason for this is because the calculated column cannot see the existing relationships in the data model, and therefore does not automatically return the column you want from another table. There is a much more complicated explanation behind all this, but for now, suffice to say that navigation functions (`RELATED` and `RELATEDTABLE`) allow calculated columns to interact with and use existing relationships.

If I rewrite the following DAX formula with the `RELATED` function, then you will notice that IntelliSense has returned, along with the autocomplete functionality that was previously discussed:

```
Temperature Key =  
RELATED('Sales T
```



'Sales Territory'
'Sales Territory'[Sales Territory Country]
'Sales Territory'[Sales Territory Group]
'Sales Territory'[Sales Territory Image]
'Sales Territory'[Sales Territory Key]
'Sales Territory'[Sales Territory Region]
'Sales Territory'[Sales Territory AlternateKey]

Figure 8-Temperature key column

Now it's time to create a Temperature Key column on the Internet Sales table. Create a new column on the Internet Sales table and then type in the following DAX formula:

Temperature Key =

```
RELATED('Sales Territory'[Sales Territory Region]) & //Return the region from Sales Territory table  
RELATED('Date (Order)'[Month Number Of Year]) //Return the Month number of year from the Date table
```

Figure 9-Temperature Key column on the Internet Sales table

Now that the temperature key has been created on the Internet Sales table, let's create the relationship. Click Manage Relationships from the home ribbon and then click New... to open the Create Relationship window. Then complete the following steps to create a new relationship. The relevant fields and entries for each step are marked out on the following screenshot:

1. Select Internet Sales from the first drop-down selection list
2. Select the Temperature Key from the list of columns
3. Select Temperature from the second drop-down selection list (scroll right)
4. Select Key from the list of columns
5. Click OK to save your new relationship:

Internet Sales

1

Number	Order Date	DueDate	ShipDate	Temperature Key
	Saturday, December 29, 2007	Thursday, January 10, 2008	Saturday, January 5, 2008	Southwest12
	Monday, September 10, 2007	Saturday, September 22, 2007	Monday, September 17, 2007	Southwest9
	Monday, June 23, 2008	Saturday, July 5, 2008	Monday, June 30, 2008	Southwest6

2

Temperature

3

Region	Month	MonthNumber	Key	Avg Temp	Temperature Range
Northeast	Jan	1	Northeast1	26.3	Cold
Northeast	Feb	2	Northeast2	25.4	Cold
Northeast	Mar	3	Northeast3	31.4	Cold

4

Figure 10-Creating new relationship

# Calculated measures – the basics

Calculated measures are very different than calculated columns. Calculated measures are not static, and operate within the current filter context of a report; therefore, calculated measures are dynamic and ever-changing as the filter context changes. You were introduced to filter context in the previous chapter. The concept of the filter context will be slightly expanded on later in this chapter. Calculated measures are powerful analytical tools, and because of the automatic way that measures work with filter contexts they are surprisingly simple to author.

Before you start learning about creating measures, let's first discuss the difference between implicit and explicit measures.

Calculated measures can do the following:

- They can be assigned to any table
- They interact with all the relationships in the data model automatically, unlike calculated columns.



# Calculated measure – basic aggregations

In this section, you are going to create four simple calculated measures:

- Total Sales
- Total Cost
- Profit
- Profit Margin

# Total Sales

To create your first measure, select the Internet Sales table and then click New Measure... from the modeling ribbon. In the formula bar, type the following code and hit *Enter*:

```
Total Sales = SUM('Internet Sales'[Sales Amount])
```

One of the benefits of creating explicit measures is the ability to centralize formatting. Once the measure has been created, navigate to the modeling ribbon and change the formatting to **\$ English (United States)**, as shown in the following screenshot:



Figure 11-Change formatting to \$ English(United States)

# Total Cost

Now let's create the Total Cost measure. Once again, this is a simple `SUM` operation. Click **New Measure...** from the modeling ribbon and type in the following DAX formula:

```
Total Cost = SUM('Internet Sales'[Total Product Cost])
```

Remember to apply formatting to this new measure; it is easy to miss this step when learning to create measures. The formatting should be `$ English (United States)`.

# Profit

**Profit** is the next measure you will create. You may attempt to write something such as the following:

```
Profit = SUM('Internet Sales'[Sales Amount]) - SUM('Internet Sales'[Total Product Cost])
```

This calculation would be technically correct; however, it's not the most efficient way to write code. In fact, another benefit of building explicit measures is that they can be built on top of each other.

Reusing existing calculated measures will make the code more readable, and make code changes easier and less time consuming. Imagine for a moment that you discovered that the Total Sales calculation is not correct. If you encapsulated all this logic in a single measure and reused that measure in your other measures, then you need only change the original measure, and any updates would be pushed to all other measures.

Now it's time to create the Profit measure. select your Internet Sales table and then click on New Measure... from the modeling ribbon. Type the following into the formula bar—remember to format it:

```
Profit = [Total Sales] - [Total Cost]
```

This calculation returns the same results as the original attempt. The difference is that now you are reusing measures that were already created in the data model. You may have noticed that I referenced the name of the measure without the table name. When referencing explicit measures in your code, it is considered

a best practice to exclude the table name.

# Profit Margin

Now it's time to create the Profit Margin calculation (the profit margin is simply profit divided by sales). For this measure, you are going to use the `DIVIDE` function. The `DIVIDE` function is recommended over the divide operator (`/`) because the `DIVIDE` function automatically handles divide by zero occurrences. In the case of divide by zero occurrences, the `DIVIDE` function returns blank.

Create a new measure on the Internet Sales table using the following code:

```
Profit Margin = DIVIDE([Profit], [Total Sales])
```

Next, set the formatting as a percentage. From the modeling ribbon, click on the % icon, as shown in the following screenshot:

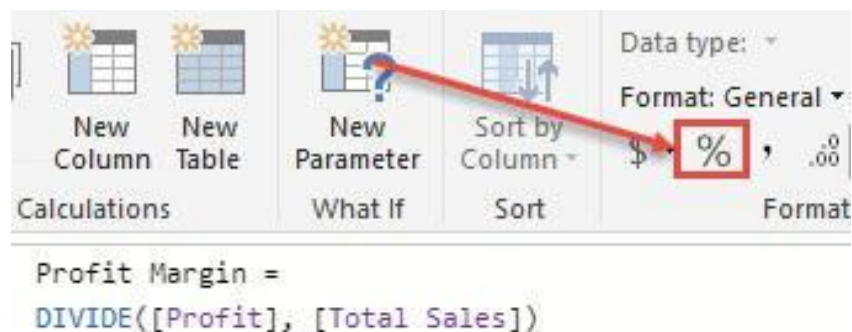


Figure 12-Setting formatting as a percentage

# Optional parameters

You may have noticed that the `DIVIDE` function accepted three parameters and you only provided two. The third parameter allows you to set an alternative result for divide by zero occurrences. This alternate result is optional. Optional parameters are denoted by square brackets on both sides of the parameter. These optional parameters are prevalent in many DAX functions. Take a look at the following screenshot:

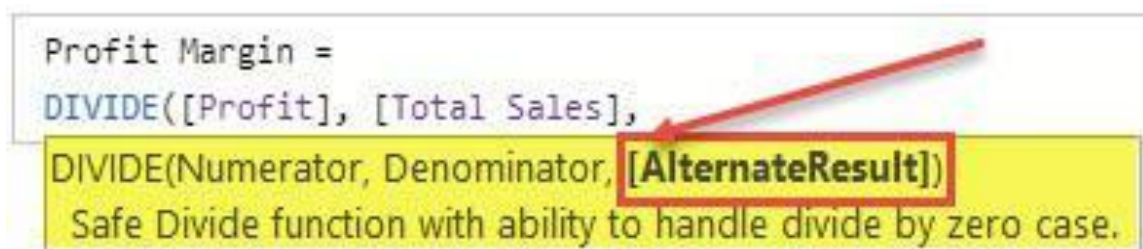
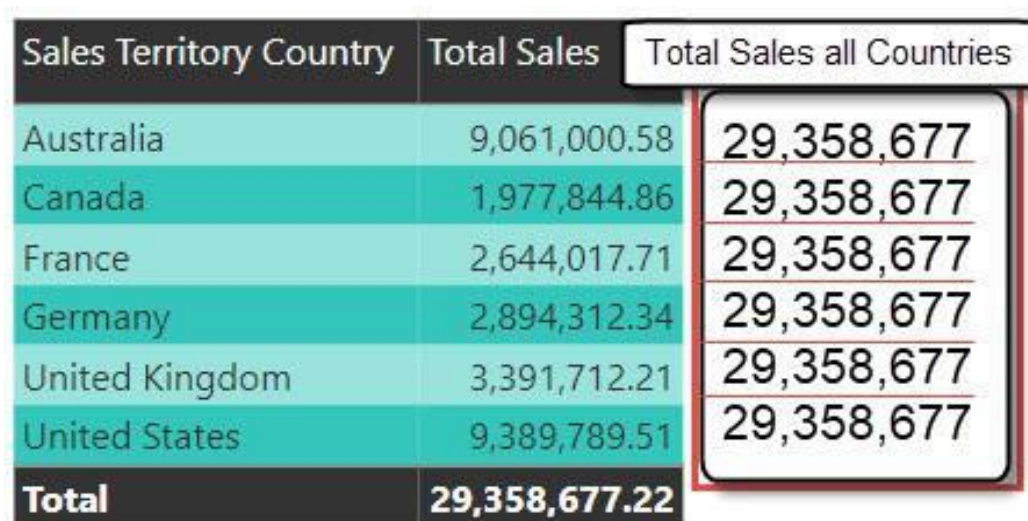


Figure 13-Optional parameters in DAX functions

# Calculate

The `CALCULATE` function is an extremely powerful tool in the arsenal of any DAX author. This is because the `CALCULATE` function can be used to ignore, overwrite, or change the existing filter context. You may be asking yourself why—why would anyone want to ignore the default behavior of Power BI? Let's take a look at an example.

Let's assume you want to return the total sales of each country as a percentage of all countries. This is a very basic percent of total calculation: Total Sales per country divided by Total Sales for all countries. However, how do you get the total sales of all the countries so that you can perform this calculation? This is where the `CALCULATE` function comes into the picture. Take a look at the following screenshot:



Sales Territory Country	Total Sales	Total Sales all Countries
Australia	9,061,000.58	29,358,677
Canada	1,977,844.86	29,358,677
France	2,644,017.71	29,358,677
Germany	2,894,312.34	29,358,677
United Kingdom	3,391,712.21	29,358,677
United States	9,389,789.51	29,358,677
<b>Total</b>	<b>29,358,677.22</b>	

Figure14-Calculating total sales of all the countries

To do the percent of total calculation, you need to get Total Sales all Countries on the same row as Total Sales. This means you



need to create a new calculated measure that ignores any filters that come from the country attribute. Create a new calculated measure on your Internet Sales table using the following DAX formula:

```
Total Sales all Countries =  
CALCULATE(  
    [Total Sales],  
    ALL(  
        'Sales Territory'[Sales Territory Country]))
```

Figure 15-Create a new calculated measure on Internet sales table using DAX formula

The preceding calculation will return all sales for all countries, explicitly ignoring any filters that come from the Country column. Let's briefly discuss why this works.

The first parameter of the `CALCULATE` function is an expression, and you can think of this as an aggregation of some kind. In this example, the aggregation is simply Total Sales. The second parameter is a filter that allows the current filter context to be modified in some way. In the preceding example, the filter context is modified by *ignoring any filters* that come from the country attribute. Let's take a look at the definition for the `ALL` function used in the second parameter of the `CALCULATE` function:

**ALL:** Returns all the rows in a table, or all the values in a column, *ignoring any filters* that may have been applied.

# Percentage of total calculation

Now, create another calculated measure on the Internet Sales table using the following code. Make sure that you format the measure as a percentage:

```
% of All Countries = DIVIDE([Total Sales], [Total Sales all Countries])
```

In the following screenshot, you can see the completed example with both of the new measures created in this section. Without a basic understanding of the `CALCULATE` function, this type of percent of total calculation would be nearly impossible:

Sales Territory Country	Total Sales	Total Sales all Countries	% of All Countries
Australia	9,061,000.58	1 \$29,358,677.22	2 30.86%
Canada	1,977,844.86	\$29,358,677.22	6.74%
France	2,644,017.71	\$29,358,677.22	9.01%
Germany	2,894,312.34	\$29,358,677.22	9.86%
NA		\$29,358,677.22	
United Kingdom	3,391,712.21	\$29,358,677.22	11.55%
United States	9,389,789.51	\$29,358,677.22	31.98%
<b>Total</b>	<b>29,358,677.22</b>	<b>\$29,358,677.22</b>	<b>100.00%</b>

Figure 16- Completed example with both of the new measures

# Time intelligence

Another advantage of Power BI is how easily time intelligence can be added to your data model. Within **data analysis expressions (DAX)**, you have a comprehensive list of built-in time intelligence functions to make this very easy. In this section, you are going to use these built-in functions to create the following measures:

- Year to Date Sales
  - Year to Date Sales (Fiscal Calendar)
  - Prior Year Sales
-

# Year to Date Sales

Create a new calculated measure on your Internet Sales table using the following DAX formula. Remember to format the measure as **\$ English (United States)**:

```
YTD Sales = TOTALYTD([Total Sales], 'Date (Order)'[Date])
```

# YTD Sales (Fiscal Calendar)

Maybe your requirement is slightly more complex, and you need to see the year-to-date sales based on your fiscal year end rather than the calendar year end date. The `TOTALYTD` function has an optional parameter that allows you to change the default year end date from "12/31" to a different date. Create a new calculated measure on your Internet Sales table using the following DAX formula:

```
Fiscal YTD Sales = TOTALYTD([Total Sales], 'Date (Order)'[Date], "03/31")
```

Now, let's take a look at both of these new measures in a table in Power BI:

Year	English Month Name	Total Sales	YTD Sales	Fiscal YTD Sales
2005	July	473,388.16	\$473,388	\$473,388
2005	August	506,191.69	\$979,580	\$979,580
2005	September	473,943.03	\$1,453,523	\$1,453,523
2005	October	513,329.47	\$1,966,852	\$1,966,852
2005	November	543,993.41	\$2,510,846	\$2,510,846
2005	December	755,527.89	1 \$3,266,374	\$3,266,374
2006	January	596,746.56	2 \$596,747	\$3,863,120
2006	February	550,816.69	\$1,147,563	\$4,413,937
2006	March	644,135.20	\$1,791,698	3 \$5,058,072
2006	April	663,692.29	\$2,455,391	4 \$663,692
2006	May	673,556.20	\$3,128,947	\$1,337,248
<b>Total</b>		<b>29,358,677.22</b>		

Figure 17-Both the new measure in a table

The newly created measures YTD Sales and Fiscal YTD Sales have both been added to the preceding table. Let's take a closer look at how these two measures are different; the relevant sections in the table are annotated with the numbers one to four, corresponding to the following notes:

1. The amount displayed for December 2005 is \$3,266,374. This is the cumulative total of all sales from January 1, 2005 to December, 2005.
2. As expected, the cumulative total starts over as the year switches from 2005 to 2006; therefore, the YTD Sales amount for January 2006 is \$596,747.
3. In the Fiscal YTD Sales column, the cumulative total works slightly differently. The displayed amount of \$5,058,072 is the cumulative total of all sales from April 1st, 2005 to March 31, 2006.
4. Unlike the YTD Sales measure, the Fiscal YTD Sales measure does not start over until April 1. The amount displayed for April 2006 of \$663,692 is the cumulative total for April. This number will grow each month until May 31, at which point the number will reset again.

# Prior Year Sales

A lot of time series analysis consists of comparing current metrics to the previous month or previous year. There are many functions in DAX that work in conjunction with the `CALCULATE` function to make these types of calculations easy. You are going to create a new measure to return the total sales for the prior year.

Create a new calculated measure on your Internet Sales table using the following DAX formula:

```
Prior Year Sales =  
CALCULATE(  
    [Total Sales],           // SUM('Internet Sales'[Sales Amount])  
    SAMEPERIODLASTYEAR(     // Change the filter context to go back one year  
        'Date (Order)'[Date])  
)
```

Figure 18-Create a new calculated measure on your Internet sales

`CALCULATE` allows you to ignore or even change the current filter context. In the preceding formula, `CALCULATE` is used to take the current filter context and change it to one year ago. This calculated measure also works at the day, month, quarter, and year level of the hierarchy. For example, if you are looking at sales for June 15, 2018, then the Prior Year Sales measure would return sales for June 15, 2017. However, if you were simply analyzing your sales aggregated at the month level for June 2018, then the measure would return the sales for June 2017.

---