



# Beginning DAX with Power BI

The SQL Pro's Guide to  
Better Business Intelligence

—  
Philip Seamark

Apress®

# **Beginning DAX with Power BI**

**The SQL Pro's Guide to Better  
Business Intelligence**

**Philip Seamark**

Apress®

# ***Beginning DAX with Power BI***

Philip Seamark  
UPPER HUTT, New Zealand

ISBN-13 (pbk): 978-1-4842-3476-1  
<https://doi.org/10.1007/978-1-4842-3477-8>

ISBN-13 (electronic): 978-1-4842-3477-8

Library of Congress Control Number: 2018937896

Copyright © 2018 by Philip Seamark

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director, Apress Media LLC: Welmoed Spahr  
Acquisitions Editor: Joan Murray  
Development Editor: Laura Berendson  
Coordinating Editor: Jill Balzano

Cover designed by eStudioCalamar

Cover image designed by Freepik ([www.freepik.com](http://www.freepik.com))

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail [orders-ny@springer-sbm.com](mailto:orders-ny@springer-sbm.com), or visit [www.springeronline.com](http://www.springeronline.com). Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail [rights@apress.com](mailto:rights@apress.com), or visit <http://www.apress.com/rights-permissions>.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at <http://www.apress.com/bulk-sales>.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at [www.apress.com/9781484234761](http://www.apress.com/9781484234761). For more detailed information, please visit <http://www.apress.com/source-code>.

Printed on acid-free paper

*To Grace, Hazel, Emily  
... and of course Rebecca.*

# Table of Contents

<b>About the Author .....</b>	<b>xi</b>
<b>About the Technical Reviewer .....</b>	<b>xiii</b>
<b>Foreword .....</b>	<b>xv</b>
<b>Acknowledgments .....</b>	<b>xvii</b>
<b>Chapter 1: Introduction to DAX .....</b>	<b>1</b>
What Is DAX?.....	2
What Is a Data Model? .....	3
Components of a DAX Data Model .....	4
Data .....	4
Tables .....	4
Columns.....	4
Relationships.....	4
Measures.....	4
Hierarchies .....	5
Your First DAX Calculation.....	5
Your First Calculation.....	6
IntelliSense .....	8
Formatting .....	9
Comments .....	9
Your Second DAX Calculation.....	11
Your Third DAX Calculation.....	12
The CALENDARAUTO Function .....	13
Datatypes .....	14
The Whole Number Datatype .....	15
The Decimal and Fixed Decimal Number Datatype .....	16

TABLE OF CONTENTS

- Date and DateTime ..... 17
- Time..... 18
- Operators ..... 18
  - Arithmetic Operators ..... 19
  - Comparison Operators..... 19
  - Concatenation Operator..... 20
  - Logical Operators ..... 20
  - Operator Precedence..... 20
- Relationships ..... 22
  - Types of Relationships..... 22
- Hierarchies..... 24
- Chapter 2: Variables ..... 27**
  - Variable Structure ..... 28
    - Using Variables with Text..... 29
    - Using Variables in Calculated Columns ..... 31
    - Using Variables in Calculated Measures..... 32
    - Using Variables in Calculated Tables ..... 33
    - Debugging Using Variables..... 36
  - Nesting Variables ..... 37
    - Nested Variables (Complex)..... 38
- Chapter 3: Context ..... 41**
  - Filter Context..... 41
    - Implicit Filter Context ..... 43
    - Explicit Filter Context..... 50
    - Calculated Column..... 53
    - Calculated Measure..... 54
    - Hardcoded Example..... 56
  - Row Context..... 57
    - Iterators ..... 60

How Data Is Stored in DAX.....	62
Column Indexes .....	63
Context Transition .....	65
<b>Chapter 4: Summarizing and Aggregating .....</b>	<b>67</b>
The SUMMARIZE Function.....	69
Relationships.....	72
SUMMARIZE with a Filter.....	74
The SUMMARIZECOLUMNS Function .....	76
SUMMARIZECOLUMNS with a Filter.....	77
The GROUP BY Function .....	78
The CURRENTGROUP() Function .....	79
GROUPBY Iterators.....	80
The GROUPBY Double Aggregation.....	81
GROUPBY with a Filter .....	84
Subtotal and Total Lines.....	85
The ISSUBTOTAL Function .....	88
<b>Chapter 5: Joins .....</b>	<b>93</b>
Joins in DAX.....	93
Standard Relationship .....	93
How to Join Tables Without a Relationship.....	96
The CROSSJOIN and GENERATE Functions .....	96
CROSSJOIN .....	97
GENERATE.....	97
NATURALINNERJOIN and NATURALLEFTOUTERJOIN .....	118
Lineage.....	119
UNION.....	121
LOOKUPVALUE.....	123

TABLE OF CONTENTS

- Chapter 6: Filtering..... 125**
  - Implicit Filtering ..... 125
  - Explicit Filtering ..... 133
    - The FILTER Function ..... 133
    - Filters and Calculated Tables..... 162
  
- Chapter 7: Dates ..... 165**
  - Date..... 166
  - Time ..... 166
  - Date/Calendar Tables ..... 167
    - Automatic Date Tables ..... 171
  - Quick Measures ..... 172
  - Sorting by Columns ..... 172
  - Including the Year When Using Month ..... 174
  - Time Intelligence..... 174
  - Year to Date..... 175
    - Period Comparisons ..... 178
  - The Rolling Average ..... 181
  - Rolling Your Own Table..... 184
    - CALENDAR ..... 185
    - CALENDARAUTO..... 185
    - Expanding the Date Table ..... 186
    - Adding the Year ..... 188
    - Fiscal Year ..... 189
    - Days from Today ..... 190
    - Weekly Buckets ..... 190
    - Is Working Day..... 192
    - Weekday Name..... 194
    - Rolling Your Own—Summary..... 194
  - Optimizing Dates..... 194



<b>Chapter 8: Debugging and Optimizing</b> .....	<b>197</b>
Debugging in DAX .....	197
Debugging Using Columns .....	199
Debugging Using Variables .....	201
Debugging Calculated Measures .....	202
External Tools .....	206
Optimizing .....	217
Optimizing Data Models .....	217
<b>Chapter 9: Practical DAX</b> .....	<b>231</b>
Creating a Numbers Table .....	231
Adding a Date Table .....	233
Creating the Sales Table .....	234
Optimizing Sales .....	242
Calculated Columns vs. Calculated Measures .....	243
Calculated Columns .....	243
Calculated Measures .....	243
Show All Sales for the Top Ten Products .....	248
Double Grouping .....	252
<b>Index</b> .....	<b>259</b>

# About the Author



**Philip Seamark** is an experienced Data Warehouse and Business Intelligence consultant with a deep understanding of the Microsoft stack and extensive knowledge of Data Warehouse methodologies and enterprise data modeling. He is recognized for his analytical, conceptual, and problem-solving abilities and has more than 25 years of commercial experience delivering business applications across a broad range of technologies. His expertise runs the gamut from project management, dimensional modeling, performance tuning, ETL design, development and optimization, and report and dashboard design to installation and administration.

In 2017 he received a Microsoft Data Platform MVP award for his contributions to the Microsoft Power BI community site, as well as for speaking at many data, analytic, and reporting events around the world. Philip is also the founder and organizer of the Wellington Power BI User Group.

# About the Technical Reviewer

**Jeffrey Wang** stumbled upon Power BI technologies by accident in 2002 in the suburbs of Philadelphia, fell in love with the field, and has stayed in the BI industry ever since. In 2004, he moved his family across the continent to join the Microsoft Analysis Services engine team as a software engineer just in time to catch the tail end of SQL Server 2005 Analysis Services RTM development. Jeffrey started off with performance improvements in the storage engine. After he found that users desperately needed faster MDX (Multidimensional) calculations, he switched to the formula engine and participated in all the improvements to MDX afterward. He was one of the inventors of the DAX programming language in 2009 and has been driving the progress of the DAX language since then. Currently, Jeffrey is a principal engineering manager in charge of the DAX development team and is leading the next phase of BI programming and its modeling evolution.

# Foreword

Power BI has changed the definition of business intelligence (BI) tools. Taking tools that were historically reserved for data scientists and making them easy to use and economically available to business analysts has enabled an entire culture of self-service BI. This book is doing the same for data access programming. It breaks complex concepts into simple, easy-to-understand steps and frames them in a business context that makes it easy for you to start learning the DAX language and helps you solve real-world business challenges on a daily basis.

—Charles “Chuck” Sterling (of Microsoft).

# Acknowledgments

Thanks to Marco Russo, Alberto Ferrari, and Chris Webb for providing many years of high-quality material in the world of business intelligence.

## CHAPTER 1

# Introduction to DAX

The aim of this book is to help you learn how you can use the DAX language to improve your data modelling capability using tools such as Microsoft Power BI, Excel Power Pivot, and SSAS Tabular. This book will be particularly useful if you already have a good knowledge of T-SQL, although this is not essential.

Throughout the book, I present and solve a variety of scenarios using DAX and provide equivalent T-SQL statements primarily as a comparative reference to help clarify each solution. My personal background is as someone who has spent many years building solutions using T-SQL, and I would like to share the tips and tricks I have acquired on my journey learning DAX with those who have a similar background. It's not crucial for you to be familiar with T-SQL to get the best out of this book because the examples will still be useful to someone who isn't. I find it can be helpful to sometimes describe an answer multiple ways to help provide a better understanding of the solution.

In this book, I use Power BI Desktop as my primary DAX engine and most samples use data from the WideWorldImportersDW database, which is freely available for download from Microsoft's website. This database can be restored to an instance of Microsoft SQL Server 2016 or later. I am using the Developer edition of SQL Server 2016.

I recommend you download and install the latest version of Power BI Desktop to your local Windows PC. The download is available from [powerbi.microsoft.com/desktop](https://powerbi.microsoft.com/desktop), or you can find it via a quick internet search. The software is free to install and allows you to load data and start building DAX-based data models in a matter of minutes.

The WideWorldImportersDW database is clean, well-organized, and an ideal starting point from which to learn to data model using DAX.

The aim of this first chapter is to cover high-level fundamentals of DAX without drilling into too much detail. Later chapters explore the same fundamentals in much more depth.

# What Is DAX?

Data Analysis Expressions (DAX) is both a query and functional language. It made its first appearance back in 2009 as part of an add-in to Microsoft Excel 2010. The primary objective of DAX is to help organize, analyze, understand, and enhance data for analytics and reporting.

DAX is not a full-blown programming language and does not provide some of the flow-control or state-persistence mechanisms you might expect from other programming languages. It has been designed to enhance data modeling, reporting, and analytics. DAX is constantly evolving with new functions being added on a regular basis.

DAX is described as a functional language, which means calculations primarily use functions to generate results. A wide variety of functions are provided to help with arithmetic, string manipulation, date and time handling, and more. Functions can be nested but *you cannot create your own*. Functions are classified into the following categories:

- DateTime
- Filter
- Info
- Logical
- Mathtrig
- ParentChild
- Statistical
- Text

There are over 200 functions in DAX. Every calculation you write will use one or more of these. Each function produces an output with some returning a single value and others returning a table. Functions use parameters as input. Functions can be nested so the output of one function can be used as input to another function.

Unlike T-SQL, there is no concept of INSERT, UPDATE, or DELETE for manipulating data in a data model. Once a physical table exists in a Power BI, SSAS Tabular, or Excel PowerPivot data model, DAX cannot add, change, or remove data from that table. Data can only be filtered or queried using DAX functions.

# What Is a Data Model?

A *data model* is a collection of data, calculations, and formatting rules that combine to create an object that can be used to explore, query, and better understand an existing dataset. This can include data from many sources.

Power BI Desktop, SSAS Tabular, and PowerPivot for Excel can import data from a wide variety of data sources including databases and flat files or directly from many source systems. Once imported, calculations can be added to the model to help explore and make sense of the data.

Data is organized and stored into tables. Tables are two dimensional and share many characteristics with databases tables. Tables have columns and rows, and relationships can be defined between tables to assist calculations that use data from multiple tables. Calculations can be as simple as providing a row count over a table or providing a sum of values in a column. Well-considered calculations should enhance your data model and support the process of building reports and performing analytical tasks known as *measures*.

It's the combination of data and measures that become your data model.

The Power BI Desktop user interface consists of three core components. First, the *Report View* provides a canvas that lets you create a visual layer of your data model using charts and other visuals. Report View also lets you control the layout by dragging, dropping, and resizing elements on the report canvas. It's the canvas that is presented to the end user when they access the report.

The second component is the *Data View*, which provides the ability to see raw data for each table in the model. Data View can show data for one table at a time and is controlled by clicking the name of a table from the list of tables in the right-hand panel. Columns can be sorted in this view, but sorting here has no impact on any sorting by visuals on the report canvas. Columns can be renamed, formatted, deleted, hidden, or have their datatype defined using the Report View. A hidden column will always appear in the Report View but not in any field list in the report.

It's possible to add or change calculations from both Report and Data View.

The last component of the Power BI Desktop user interface is the *Relationship View*. This section shows every table in the data model and allows you to add, change, or remove relationships between tables.



# Components of a DAX Data Model

The DAX data modeling engine is made up of six key components.

## Data

The first step of building a data model is importing data. A wide variety of data sources are available, and once they are imported, they will be stored in two-dimensional tables. Sources that are not two dimensional can be used, but these will need to be converted to a two-dimensional format before or during import. The query editor provides a rich array of functions that help with this type of transformation.

## Tables

Tables are objects used to store and organize data. Tables consist of columns that are made up of source data or results of DAX calculations.

## Columns

Each table can have one or more columns. The underlying data engine stores data from the same column in its own separate index. Unlike T-SQL, DAX stores data in columns rather than in rows. Once data has been loaded to a column, it is considered static and cannot be changed. Columns can also be known as *fields*.

## Relationships

Two tables can be connected via a relationship defined in the model. A single column from each table is used to define the relationship. Only *one-to-many* and *one-to-one* relationships are supported. Many-to-many relationships cannot be created. In DAX, the most common use of relationships is to provide filtering rather than to mimic normalization of data optimized for OLTP operations.

## Measures

A *measure* is a DAX calculation that returns a single value that can be used in visuals in reports or as part of calculations in other measures. A measure can be as simple as a row count of a table or sum over a column. Measures react and respond to user interaction

and recalculate as a report is being used. Measures can return new values based on updates to the selection of filters and slicers.

## Hierarchies

*Hierarchies* are groupings of two or more columns into levels that can be drilled up/down through by interactive visuals and charts. A common hierarchy might be over date data that creates a three-level hierarchy over year, month, and day. Other common hierarchies might use geographical data (country, city, suburb), or structures that reflect organizational groupings in HR or Product data.

## Your First DAX Calculation

It's possible to import data and have no need to write in DAX. If data is clean and simple and report requirements are basic, you can create a model that needs no user-created calculations. Numeric fields dragged to the report canvas will produce a number.

If you then drag a non-numeric field to the same visual, it automatically assumes you would like to group your numeric field by the distinct values found in your nonnumeric field. The default aggregation over your numeric field will be SUM. This can be changed to another aggregation type using the properties of your visual. Other aggregation types include AVERAGE, COUNT, MAX, MIN and so on.

In this approach, the report creates a DAX-calculated measure on your behalf. These are known as *implicit measures*. Dragging the 'Fact Sale'[Quantity] field to the canvas automatically generates the following DAX statement for you:

```
CALCULATE(SUM('Fact Sale'[Quantity]))
```

This calculation recomputes every time a slicer or filter is changed and should show values relevant for any filter settings in your report.

Most real-world scenarios require at least some basic enhancements to raw data, and this is where adding DAX calculations can improve your model. When you specifically create calculated measures, these are known as *explicit measures*.

Some of the most common enhancements are to provide the ability to show a count of the number of records in a table or to sum values in a column.

Other enhancements might be to create a new column using values from other columns in the same row, or from elsewhere in the model. A simple example is a column that multiplies values from columns such as Price and Qty together to produce

a total. A more complicated example might use data from other tables in a calculation to provide a value that has meaning to that row and table.

Once basic count or sum calculations have been added, more sophisticated calculations that provide cumulative totals, period comparisons, or ranking can be added.

These are the three types of calculations in DAX:

- Calculated columns
- Calculated measures
- Calculated tables

We explore each of these calculations in more detail later in this book and I include hints on how and when you might choose one type over another.


---

**Note** *Calculated tables* (tables that are the result of a DAX calculation) can only be created in the DAX engine used by Power BI Desktop and SSAS Tabular.

---

## Your First Calculation

The first example I cover creates a simple calculated measure using data from the WideWorldImportersDW database (Figure 1-1). The dataset has a table called 'Fact Sale' that has a column called [Total Including Tax]. The calculation produces a value that represents a sum using values stored in this column.



Quantity	Unit Price	Tax Rate	Total Excluding Tax	Tax Amount	Profit	Total Including Tax	Total
1	13	15	13	1.95	8.5	14.95	
1	13	15	13	1.95	8.5	14.95	
1	13	15	13	1.95	8.5	14.95	
1	13	15	13	1.95	8.5	14.95	
1	13	15	13	1.95	8.5	14.95	
1	13	15	13	1.95	8.5	14.95	
1	13	15	13	1.95	8.5	14.95	
1	13	15	13	1.95	8.5	14.95	
1	13	15	13	1.95	8.5	14.95	
1	13	15	13	1.95	8.5	14.95	
1	13	15	13	1.95	8.5	14.95	
1	13	15	13	1.95	8.5	14.95	
1	13	15	13	1.95	8.5	14.95	
1	13	15	13	1.95	8.5	14.95	

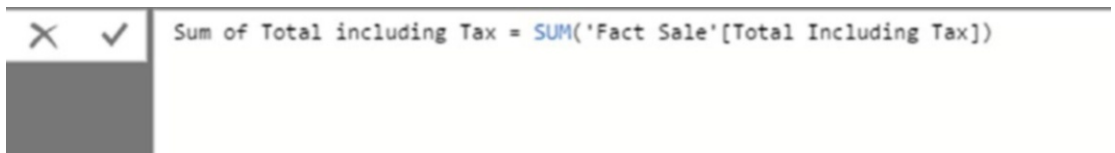
**Figure 1-1.** A sample of data from the 'Fact Sale' table

When viewing this table in Data View, we see the unsummarized value for each row in the [Total Including Tax] column. A calculated measure is required to show a single value that represents a sum of every row in this column.

In Power BI, you can create a calculated measure using the ribbon, or by right-clicking the table name in the Report View or Data View. This presents an area below the ribbon where you can type the DAX code for your calculation. The text for this calculated measure should be

```
Sum of Total including Tax = SUM('Fact Sales'[Total Including Tax])
```

This should look as it does in Figure 1-2.



**Figure 1-2.** DAX for the first calculated measure. This calculation uses the SUM function to return a single value anywhere the calculated measure is used in the report. When dragged and dropped to the report canvas using a visual with no other fields or filters, the value should show as \$198,043,493.45.

The structure of the formula can be broken down as follows: starting from the left, the first part of the text sets the name of the calculated measure. In this case, the name is determined by all text to the left of the = operator. The name of this calculated measure is [Sum of Total including Tax]. Names of calculated measures should be unique across the model including column names.

This name is how you will see the measure appear in the field list as well as how it may show in some visuals and charts.

---

**Note** Spaces between words are recommended when creating names for calculated measures and columns. Avoid naming conventions that use the underscore character or remove spaces altogether. Use natural language as much as possible. Use names that are brief and descriptive. This will be especially helpful for Power BI features such as Q&A.

---

The = sign separates the calculation name from the calculation itself. A calculated measure can only return a single value and never a list or table of values. In more advanced scenarios, steps involving groups of values can be used, but the result must be a single value.

All text after the = sign is the DAX code for the calculated measure. This calculation uses the SUM function and a single parameter, which is a reference to a column. The single number value that is returned by the SUM function represents values from every row from the [Total Including Tax] column added together. The datatype for the column passed to the SUM function needs to be numeric and cannot be either the Text or DateTime datatypes.

The notation for the column reference is *fully qualified*, meaning it contains both the name of the table and name of the column. The table name is encapsulated inside single quotations ( ' '). This is optional when your table name doesn't contain spaces. The column name is encapsulated inside square brackets ( [ ] ).

Calculated measures belong to a single table but you can move them to a new home table using the Home Table option on the Modeling tab. Calculated measures produce the same result regardless of which home table they reside on.

---

**Note** When making references to other calculated measures in calculations, never prefix them with the table name. However, you should always include the name of a table when referencing a column.

---

## IntelliSense

*IntelliSense* is a form of predictive text for programmers. Most modern programming development environments offer some form of IntelliSense to help guide you as you write your code. If you haven't encountered this before, it is an incredibly useful way to help avoid syntax errors and keep calculations well-formed.

DAX is no different, and as you start typing your calculation, you should notice suggestions appearing as to what you might type next. Tooltips provide short descriptions about functions along with details on what parameters might be required.

IntelliSense also helps you ensure you have symmetry with your brackets, although this can sometimes be confusing if you are not aware it is happening.

IntelliSense suggestions can take the form of relevant functions, or tables or columns that can be used by the current function. IntelliSense is smart enough to only offer suggestions valid for the current parameter. It does not offer tables to a parameter that only accepts columns.

In the case of our formula, when we type in the first bracket of the SUM function, IntelliSense offers suggestions of columns that can be used. It does not offer tables or calculated measures as options because the SUM function is only designed to work with columns.

## Formatting

As with T-SQL and pretty much any programming language, making practical use of line spacing, carriage returns, and tabs greatly improves readability and, more importantly, understanding of the code used in the calculation. Although it's possible to construct a working calculation using complex code on just a single line, it is difficult to maintain. Single-line calculations also lead to issues playing the *bracket game*.

---

**Note** The *bracket game* is where you try to correctly pair open/close brackets in your formula to produce the correct result. Failure to pair properly means you lose the game and your formula doesn't work.

---

A good tip is to extend the viewable area where you edit your calculation before you start by repeating the Shift-Enter key combination multiple times or by clicking the down arrow on the right-hand side.

## Comments

Comments can also be added to any DAX calculation using any of the techniques in Table 1-1.

**Table 1-1.** *How to Add Comments*

Comment Characters	Effect
//	Text to the right is ignored by DAX until the next carriage return.
--	Text to the right is ignored by DAX until the next carriage return.
/* */	Text between the two stars is ignored by DAX and comments can span multiple lines.

Examples of ways you can add comments are shown in Figure 1-3 as is an example of how spacing DAX over multiple lines helps increase readability.

```

Measure = DATEDIFF(
    -- This is the first argument
    TODAY() ,
    // This is the second argument
    DATE(2017,12,24),
    /* This is the final argument */
    DAY
)

```

**Figure 1-3.** *Commented text styles and usage of line breaks*

By formatting code and adding comments you help to make the logic and intent of the function easier to understand and interpret for anyone looking at the calculation later.

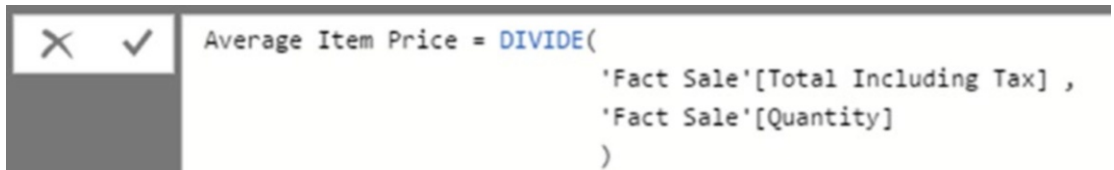
A nice alternative to the formula bar in Power BI Desktop and SSAS Tabular, is DAX Studio, which is a free product full of features designed to help you develop and debug your calculations. I provide a more detailed look at DAX Studio in Chapter 8.

## Your Second DAX Calculation

In this second example, I create a calculated column as opposed to a calculated measure. A Chapter 9 provides more detailed advice on when you should consider using a calculated column instead of a calculated measure, but in short, a calculated column adds a column in an existing table in which the values are generated using a DAX formula.

A simple calculation might use values from other columns in the same row. This example uses two columns from the 'Fact Sale' table to perform a simple division to return a value that represents an average unit price.

To add this calculation to your data model, select the 'Fact Sale' table in the Fields menu so it is highlighted. Then use the New Column button on the Modeling tab and enter the text shown in Figure 1-4.



**Figure 1-4.** A calculated column for [Average Item Price]

---

**Note** This formula works without the table name preceding each column name; however, it is highly recommended that whenever you reference a column, you always include the table name. This makes it much easier to differentiate between columns and measures when you are debugging longer DAX queries.

---

The code in Figure 1-4 adds a new column to the 'Fact Sale' table called [Average Item Price]. The value in each cell of the new column (see Figure 1-5) is the output of this calculation when it is executed once for every row in the table.



Quantity	Total Including Tax	Average Item Price
90	1552.5	\$17.25
90	3001.5	\$33.35
90	3829.5	\$42.55
90	10246.5	\$113.85
90	3881.25	\$43.13
90	3881.25	\$43.13
90	2277	\$25.30
90	3001.5	\$33.35
90	4347	\$48.30
90	3415.5	\$37.95
90	2277	\$25.30
90	1552.5	\$17.25

**Figure 1-5.** Sample of data for new calculated column

## Your Third DAX Calculation

This last example creates a calculated table. This option is only available in Power BI Desktop and SSAS Tabular, not in PowerPivot for Excel 2016 (or earlier).

You can create calculated tables using any DAX function that returns a table or by simply referencing another table in the model. The simplest syntax allows you to create a clone of another DAX table. The example shown in Figure 1-6 creates a new calculated table called 'Dates2', which is a full copy of the 'Dates' table. Modifications made to the 'Dates' table, such as adding or modifying columns, automatically flow through to the 'Dates2' table.



**Figure 1-6.** Creating a calculated table

Filters, measures, columns, and relationships can be added to the 'Dates2' table without effecting 'Dates' table. The beauty of this is that because the base table ('Dates') is reading from a physical data source, any changes to data in the 'Dates' table are reflected in the 'Dates2' table.

To extend the example so the new table only shows some rows from the original table, you can use the FILTER function as shown in Figure 1-7.



**Figure 1-7.** Calculated table created using FILTER

The 'Dates2' calculated table still has the same number of columns as 'Dates', but it only has rows that match the filter expression. This results in the 'Dates2' calculated table having 365 rows that represent a row per day for the calendar year of 2016.

It is common to use calculated tables to create summary tables that can be used as faster alternatives for calculated measures. Using calculated tables to produce an aggregated version of a sales table can provide considerable performance gains for any calculation using the aggregated version.

## The CALENDARAUTO Function

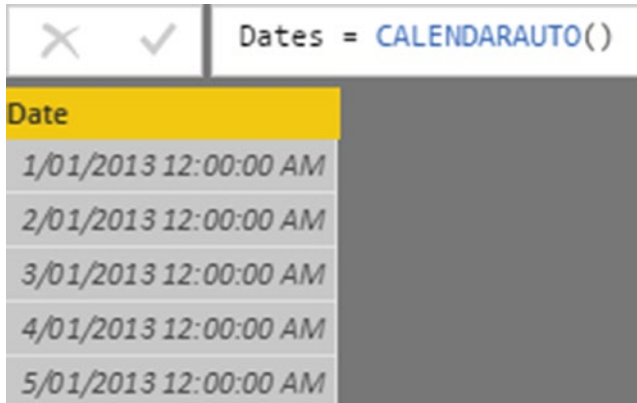
A handy DAX function that generates a calculated table without using an existing table is the CALENDARAUTO function. This function returns a table with a single column called [Date]. When called, the CALENDARAUTO function inspects every table in the model looking for columns that use either the Date or DateTime datatype.

The oldest and newest date values that appear in any column using these datatypes are used to generate a row for every day between the oldest and newest values. The dates are rounded to the start and end of the calendar year.

Date tables are invaluable for data models, particularly when you add measures designed to show period comparison and running totals accurately. There are several DAX functions that allow you to add time-intelligence logic to your model. These often rely on a date table that has contiguous dates to work properly. In the WideWorldImportersDW dataset, no rows exist in the 'Fact Sale' table with an [Invoice Date Key] that falls on a

Sunday. This might cause problems for some functions when they are processing period comparison logic. Date/Calendar tables help make calculations behave more reliably when there are gaps in dates in non-date table data.

Once you create a calculated table using the CALENDARAUTO function as shown in Figure 1-8, you can start adding calculated columns and measures to it. You can also start creating one-to-many relationships to other tables in your model.



Date
1/01/2013 12:00:00 AM
2/01/2013 12:00:00 AM
3/01/2013 12:00:00 AM
4/01/2013 12:00:00 AM
5/01/2013 12:00:00 AM

**Figure 1-8.** A sample output of the CALENDARAUTO function

## Datatypes

In DAX, it is possible to define datatypes for individual columns. The different datatypes are listed momentarily in Table 1-2 and fall into three main categories, with the exception of True/False (Boolean).

- Text
- Numeric
- DateTime

By select the best datatype, you help reduce the size of your model as well as improve performance when refreshing data and using your report.

When importing new data, the data modeling engine guesses at what the datatype for each column should be. This is something worth checking on as there may be opportunities to adjust to make sure that appropriate data types are used for each column.