

Data Analysis Expressions (DAX) In the Tabular BI Semantic Model

SQL Server Technical Article

Authors: Howie Dickerman, Peter Myers

Contributors: Kasper de Jonge, Owen Duncan

Published: December 2011

Applies to: PowerPivot, SQL Server 2012 Analysis Services

Summary: This paper introduces Data Analysis Expressions (DAX), a formula expression language used to define calculations in PowerPivot for Excel® workbooks and tabular model projects authored in SQL Server Data Tools. DAX functions provide extensive filtering to calculate on data across multiple tables, work with relationships, and perform dynamic aggregation. DAX formulas use the highly optimized in-memory engine to rapidly query and calculate values from very large data sets.

The concepts introduced in this paper will help you in learning DAX and how you can use DAX formulas effectively to solve real-world business problems and meet the Business Intelligence needs of your organization.

Copyright

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in examples herein are fictitious. No association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

© 2011 Microsoft Corporation. All rights reserved.

Microsoft, Excel, PowerPivot are trademarks of the Microsoft group of companies.

All other trademarks are property of their respective owners.

Table of Contents

Copyright	2
Executive Summary	5
Samples	6
Companion Workbook	6
Contoso Dataset	6
Data Analysis Expressions (DAX) – The Basics	8
DAX Goals	8
DAX Calculations - Calculated Columns and Measures	8
DAX Syntax	13
DAX Uses Tabular Model Data Types	14
Intro to Context – Row Context and Filter Context	14
Functionality That Does Not Exist In Excel Formulas.....	15
DAX Operators and Constants.....	16
Simple DAX Functions.....	17
BLANK() and Blank Values	17
Functions from Excel	18
FORMAT(Value, Format_text)	19
Functions to Aggregate Expressions – the “X” Functions.....	19
COUNTROWS(Table).....	20
RELATED(Column) and RELATEDTABLE(Table)	20
USERELATIONSHIP(Column1, Column2).....	23
FILTER(Table, Condition) and DISTINCT (Column)	23
Row Context and Filter Context	24
Row Context	24
Filter Context.....	25
Relationships and Filter Context	26
Measures and Filter Context	27

More DAX Functions.....	28
CALCULATE(Expression, SetFilter1, SetFilter2,...).....	28
VALUES(Column)	32
CALCULATETABLE(TableExpression, SetFilter1, SetFilter2,...)	34
ALL(Table) and ALL(Column1 [,Column2]...).....	34
ALLEXCEPT(Table [,Column1] [,Column2]...)	36
RANKX(Table, Expression [,Value] [,Order] [,Ties]...).....	36
RANK.EQ(Value, ColumnName [,Order]).....	38
TOPN(N_Value, Table [,OrderBy_Expression1] [,Order1] [,OrderBy_Expression2] [,Order2]...).....	39
LOOKUPVALUE(Result_ColumnName, Search_ColumnName1, Search_Value1 [,Search_ColumnName2] [,Search_Value2]...)	40
ALLSELECTED()	40
Time Intelligence Functions.....	41
Concepts and Best Practices.....	41
Functions that Return a Single Date	43
Functions that Return a Table of Dates.....	48
Year Over Year Growth.....	49
Calculating Many Time Periods Within a Single Measure Formula.....	51
Functions that Evaluate Expressions Over a Time Period	56
Parent-Child Functions	57
Sample Formulas	63
Calculated Columns.....	63
Measures.....	64
DAX Query	64
Additional Resources.....	69

Executive Summary

With the release of Microsoft SQL Server 2012, Analysis Services introduces a new paradigm in Business Intelligence data modeling, the BI Semantic Model (BISM). BISM includes both traditional multidimensional models developed by BI professionals and deployed to Analysis Services, and the new Tabular Model. There are two flavors of tabular models: PowerPivot for Excel, first introduced with SQL Server 2008 R2, provides self-service data modeling solutions, and the new Tabular Project, authored in SQL Server Data Tools (formerly known as BIDS) and deployed to Analysis Services running in Tabular server mode.

Both PowerPivot and tabular project data modeling approaches include the ability to import data from a wide variety of data sources, the ability to perform calculations on large volumes of in-memory data quickly, the ability to author custom calculations using the **DAX (Data Analysis Expressions) language**, and the ability to use the result of those calculations in a variety of client applications and tools, including Excel, Microsoft Power View, and Microsoft Reporting Services. Whether you choose to author tabular models by using PowerPivot or by using tabular projects in SQL Server Data Tools, the DAX language's operators, functions, and syntax is the same.

There are millions of Microsoft Excel users who are familiar with using Excel formulas to perform calculations. Those calculations may be as simple as adding up a column of numbers, or they may be far more complex simulations of various business models. But in every case, each formula is built using a combination of basic operators and functions that are provided within Excel. Similarly, for professional BI solution developers using Multidimensional Expressions (MDX) in multidimensional model solutions, the fundamental concepts of creating calculations by using formulas is much the same; however, the differences in syntax, operators, and functions between MDX and DAX formulas is greater than it is between Excel and DAX formulas. Whether you are familiar with Excel or MDX formulas, this basic understanding of formulas is important when developing tabular data modeling solutions. While understanding Excel or MDX formulas will give you a head start in learning DAX, it is not necessary.

DAX formulas are very similar to Excel formulas, and there is considerable overlap between the list of DAX functions and Excel functions. But there are also significant differences, and many new functions in DAX that don't exist within Excel. These functions are designed to offer capabilities that focus on data analysis, particularly for related tables of data, and for dynamic analysis.

This paper outlines the use of DAX formulas in tabular BI Semantic Models, in-particular PowerPivot for Excel, and describes the many DAX functions (including many new ones) supported in tabular model solutions. In addition to describing the functions themselves, there is a discussion of the important concepts that any model developer will want to know. It is hoped that this paper might be a good way to become familiar with the basics of the DAX formula language.

If you are authoring tabular model solutions by using the tabular model project templates in SQL Server Data Tools, the DAX concepts, syntax, functions, and formulas described here apply much the same; only the tools user interface and the ability to use DAX formulas in row-level security for Roles is different.

Samples

Companion Workbook

DAX formulas described in this document are included in the Contoso Sample DAX Formulas PowerPivot workbook available on Microsoft Download Center:

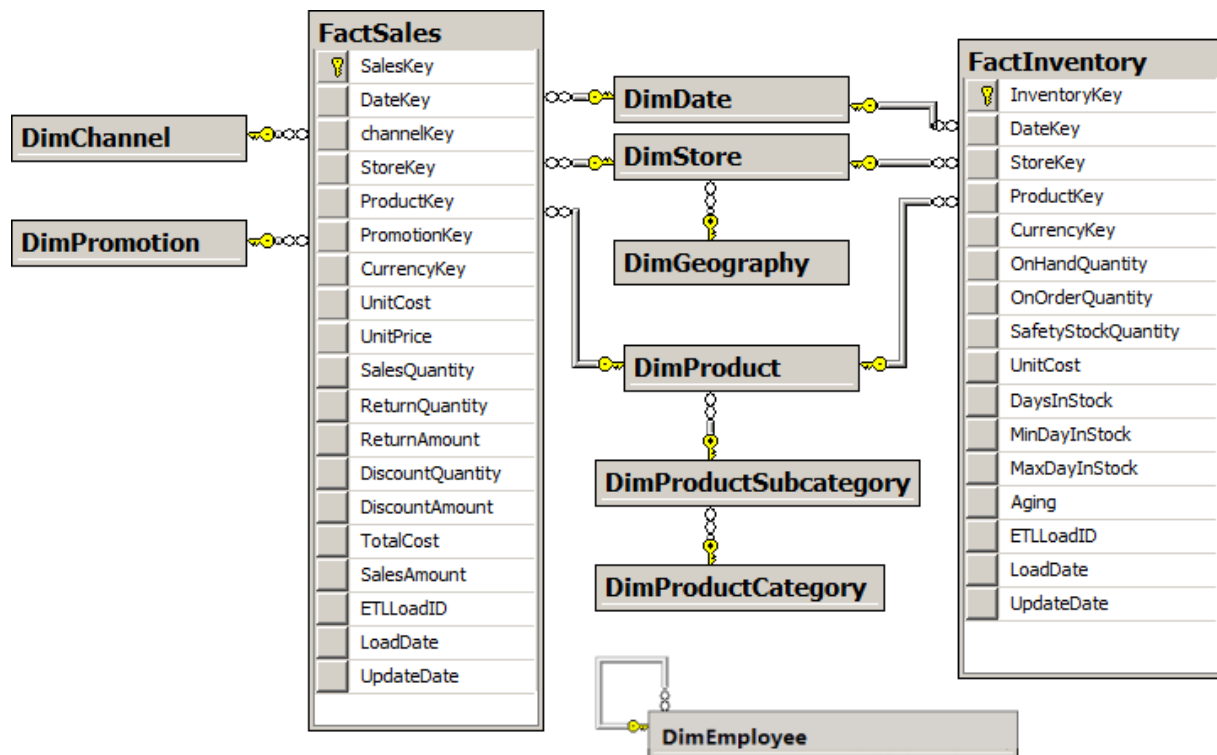
<http://go.microsoft.com/fwlink/?LinkID=237472&clcid=0x409>

Contoso Dataset

The samples used as illustrations in this document all come from an Excel workbook that imported its relational data from the “Contoso” SQL Server relational database that is available from the Microsoft Download Center as part of Microsoft Contoso BI Demo Dataset for Retail Industry:

<http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=18279>

To build the sample workbook, data is imported from eleven (11) of the tables in the Contoso database:



There are two tables of transactions in this data: FactSales and FactInventory. (This explains why the samples in this paper illustrate Sales and/or Inventory scenarios.) Eight tables are related to one or both of the transaction (fact) tables as shown in the diagram above.

The DimEmployee table is stand alone and is used solely to illustrate the parent-child DAX functions.

This sample data set contains approximately 3.4 million sales transactions and more than 8 million inventory transactions. When loaded into an Excel workbook using PowerPivot, it illustrates two very compelling advantages of PowerPivot for data analysis over simple Excel PivotTables in Excel:

- The ability to base a PivotTable on a set of multiple tables (as opposed to a single flattened table).
- The ability to consume far larger data volumes (more rows) than can fit within an Excel worksheet.

Data Analysis Expressions (DAX) – The Basics

DAX Goals

The goal of tabular models is to make data analysis really easy, and in doing so, expand data modeling to more users. The idea is that Excel users and IT professionals should be able to leverage the training and experience they already have and should not be required to learn complex multidimensional concepts nor specialized multidimensional languages like MDX.

With this in mind, DAX was created to achieve the following:

- Ease-Of-Use DAX uses standard Excel formula syntax (and some of Excel's functions)
- Relational Data Simple relational data model based on tables, columns, and relationships
- PivotTables Analysis performed via PivotTables based on the database (multiple tables)

DAX Calculations - Calculated Columns and Measures

A DAX formula is used to define a field that will be placed somewhere on an Excel PivotTable. There are two types of fields that can be placed on a PivotTable: columns and measures.

Calculated Columns are columns that are defined in the PowerPivot Window by providing a column name and a DAX expression. Calculated Columns are populated with values when they are defined and become just like any other column in a table, except that regular (non-calculated) columns are usually imported from an external data source while calculated columns are calculated after the regular columns have been loaded. The values from a column may be placed onto a PivotTable's row labels or column labels or may be placed in a PivotTable's filter/slicer to control the portion of the data that will be used in the analysis. Finally a column may be used as part of the calculation that defines a measure (described below). Here are a couple of simple calculated columns:

1) [Margin] defined as: $\text{=[SalesAmount]-[TotalCost]}$

PowerPivot for Excel - Contoso Sample DAX Formulas.xlsx

Home Design

[Margin] $\text{=[SalesAmount]-[TotalCost]}$

TotalCost	SalesAmount	ETLLoadID	LoadDate	UpdateDate	Margin
\$728.40	\$1,544.40	1	1/1/2010	1/1/2010	\$816.00
\$40.60	\$78.61	1	1/1/2010	1/1/2010	\$38.01
\$1,881.27	\$3,628.50	1	1/1/2010	1/1/2010	\$1,747.23
\$1,063.20	\$2,254.20	1	1/1/2010	1/1/2010	\$1,191.00
\$3,468.48	\$10,207.08	1	1/1/2010	1/1/2010	\$6,738.60

DimChannel DimDate DimGeography DimProduct DimProductCategory FactSales ...

Record: 1 of 3,406,089

2) [CityState] defined as: $\text{=[CityName]\&" " \& [StateProvinceName]}$

PowerPivot for Excel - Contoso Sample DAX Formulas.xlsx

Home Design

[CityState] $\text{=[CityName]\&" " \& [StateProvinceName]}$

CityName	StateProvinceName	RegionCountryName	CityState
Downey	California	United States	Downey, California
Edmonds	Washington	United States	Edmonds, Washington
Grossmont	California	United States	Grossmont, California
Haney	British Columbia	Canada	Haney, British Columbia
Imperial Beach	California	United States	Imperial Beach, California
Issaquah	Washington	United States	Issaquah, Washington
Langley	British Columbia	Canada	Langley, British Columbia

DimChannel DimDate DimGeography DimProduct DimProductCategory DimProductSubcategory DimPromotion ...

Record: 36 of 303

Measures are named formulas that are defined in the PowerPivot Window's Measure Grid, or in Excel, they can also be defined in the PowerPivot Field List (in the Excel window, while focus is on a PivotTable.) Once defined, measures can be added to the Values area of the PivotTable. The formula is evaluated when the measure is placed into the Values area of an Excel PivotTable and then it is evaluated separately for each cell in the Values area. Here are a couple of Measure definitions:

- [Sales] defined as: `=SUM(FactSales[SalesAmount])`

The screenshot shows the 'Measure Settings' dialog box. It has a title bar with a question mark and a close button. The main area contains several input fields and a section for formatting options. The 'Table name' is set to 'FactSales'. The 'Measure name (all PivotTables)' and 'Custom name (this PivotTable)' are both set to 'Sales'. The 'Description' field is empty. The 'Formula' field contains the text '=SUM(FactSales[SalesAmount])'. Below the formula field is a section titled 'Formatting Options'. Under 'Category', a list box shows 'General', 'Number', 'Currency' (which is highlighted), 'Date', and 'TRUE/FALSE'. To the right of the list box, the 'Symbol' is set to '\$', 'Decimal places' is set to '2', and the 'Use 1000 separator (.)' checkbox is checked. At the bottom right, there are 'OK' and 'Cancel' buttons.

- [Average Sale] defined as: `=AVERAGE(FactSales[SalesAmount])`

Measure Settings

Table name: FactSales

Measure name (all PivotTables): Average Sale

Custom name (this PivotTable): Average Sale

Description:

Formula:
 =AVERAGE(FactSales[SalesAmount])

Formatting Options

Category:
 General
 Number
Currency
 Date
 TRUE/FALSE

Symbol: \$

Decimal places: 2

☒ Use 1000 separator (.)

Here's what you get when you add these two measures to a PivotTable with Calendar Year on Row Labels. Observe how we get four results for each measure – one per year plus a total for all years.

Row Labels	Sales	Average Sale
2007	\$4,561,940,955.02	\$3,103.94
2008	\$4,111,233,534.68	\$3,909.73
2009	\$3,740,483,119.18	\$4,227.38
Grand Total	\$12,413,657,608.89	\$3,644.55

In addition to DAX measures that are defined by a DAX formula, the tabular model also provides a simpler way to define measures when all you want to do is take a column and aggregate it. This is known as an implicit measure. You can easily build implicit measures that calculate the SUM, COUNT, MIN, MAX, AVERAGE or DISCOUNTCOUNT of a column using checkboxes and dropdowns in the PowerPivot Field List in Excel. So while it is easy to create a measure that is merely the SUM, COUNT, MIN, MAX, AVERAGE or DISTINCTCOUNT of a column, you can perform calculations that are far more powerful using DAX formulas instead of the six built in aggregations.

To illustrate this, consider the PivotTable shown below. There are two measures in this PivotTable that show precisely the same results. One (named Sales) uses a DAX formula, (as illustrated above) while the other one (named Sum of SalesAmount) does not.

Row Labels	Sales	Sum of SalesAmount
Catalog	\$1,078,007,547.23	\$1,078,007,547.23
Online	\$2,677,599,035.07	\$2,677,599,035.07
Reseller	\$1,715,197,831.44	\$1,715,197,831.44
Store	\$6,942,853,195.14	\$6,942,853,195.14
Grand Total	\$12,413,657,608.89	\$12,413,657,608.89

The Measure Settings dialog window for the measure “Sum of SalesAmount” shows that this measure was defined by simply checking the SalesAmount column in the field list and leaving the default aggregation as “Sum”. This dialog window is slightly different than the dialog window that appears when a measure is created using a DAX formula.

The screenshot shows the 'Measure Settings' dialog box. It has a title bar with a question mark and a close button. The 'Table Name' is 'FactSales' and the 'Source Name' is 'SalesAmount'. The 'Custom Name' is 'Sum of SalesAmount'. Under the heading 'Choose how you want the selected field to be aggregated:', there is a list box with 'Sum' selected, and other options: 'Count', 'Min', 'Max', 'Average', and 'DistinctCount'. Below this, it says 'Measure will use this formula:' followed by a text box containing the DAX formula '=SUM('FactSales'[SalesAmount])'. At the bottom are 'OK' and 'Cancel' buttons.

DAX Syntax

DAX syntax mimics Excel syntax as much as possible. In particular, DAX uses function composition just as Excel does, and also refers to columns in tables using the same syntax as Excel. DAX then extends that syntax to measures as well.

References to Columns and Measures

The name of a column is always unique within a given table, and the name of a measure must be unique across the entire data model. Just as each column belongs to a particular table, each measure also belongs to a particular table. We mimic the syntax used by Excel for columns. When a DAX expression refers to a column or measure, the name of that column or measure must appear within square brackets, and it will sometimes be preceded by the name of the table to which the column or measure belongs. Here are some examples:

Fully qualified names:

- Table1[Column2] as in =SUM(Table1[Column2])
- Table2[Measure1] as in = 2.5 * Table2[Measure1]

Unqualified names:

- [Column2] as in =[Column1] + [Column2]
- [Measure2] as in =[Measure2]/[Measure3]

DAX formulas that refer to columns generally require the references be fully qualified, but an exception is made when a calculated column refers to other columns from within the same table. In these cases, the column references need not be fully qualified. Since measure names are globally unique within a model, we do not generally require measure names to be fully qualified.

Unlike Excel formulas, DAX has no notion of addressing individual cells or ranges. Notation such as B14 or C12:C15 which are valid in Excel will not work in DAX expressions. Instead, DAX always refers to a column of data by providing the (qualified or unqualified) column name. When there is a row context, this reference to a column will be interpreted as the value of that column in the current row.

Note: *It is possible to create a calculated column using the same name as a measure from a different table. When this occurs, it is important to use a fully qualified reference to the column in any formula to avoid invoking the measure when the intention was to invoke the calculated column.*

Split Measures into Separate Measures

To make measures easier to define and maintain, when possible, it is recommended to split measures into separate measures. So instead of writing a single measure that does SUM(FactSales[SalesAmount]) / SUM(Table1[SalesQuantity]) it is recommended to create two measures, and use them in a new measure. You can hide the intermediate measures so they're not exposed to the end users. This will improve readability, improve the ability to debug parts of the measure and can help when defining the measure. This will become increasingly important when your measures become more complex.

DAX Uses Tabular Model Data Types

DAX assumes that all values in columns (and the inputs and outputs of all DAX formulas) are one of the following seven data types:

Data Type	Description
Whole Number	Integer
Decimal Number	Double precision real number
Currency	Four decimal places of precision (integer divided by 10,000)
TRUE/FALSE	Boolean
Text	String
Date	Datetime values beginning with Mar 1, 1900
Binary	Binary data imported from the data source (typically for image data)

This is slightly different from Excel, where everything is either a number (real) or a string. In Excel, data types are handled by formatting numeric values rather than by using different data types. Note that date values in DAX, like Excel, are restricted to dates after March 1, 1900.

DAX also has functions that return tables of data, but those tables may not be the final value in a measure or calculated column. Tables are used only as intermediate results, and are passed as arguments into additional functions, so that a DAX formula eventually returns a scalar value that has one of the six data types mentioned above.

Intro to Context – Row Context and Filter Context

In DAX, one of the most important concepts to understand is the *context* in which each DAX expression is evaluated. Context will be examined in more detail later in this document (after some DAX functions are described, so that meaningful examples can be shown,) but for now, it is enough to be aware that each formula may be evaluated in the context of a specific row of table data (a “row context”) and/or in the context of a specific set of filters (a “filter context”). For example, the formula for a calculated column will be evaluated for each row of a table, using the “row context”. Similarly, a measure that is placed into a PivotTable will be evaluated for each cell in the values area, and each of those cells has its own “filter context” which is the combination of the cell’s row labels, column labels, report filters, and slicers.

Row Context

PowerPivot for Excel - Contoso Sample DAX Formulas.xlsx

[Margin] = [SalesAmount] - [TotalCost]

TotalCost	SalesAmount	ETLLoadID	LoadDate	UpdateDate	Margin
\$728.40	\$1,544.40	1	1/1/2010	1/1/2010	\$816.00
\$40.60	\$78.61				\$38.01
\$1,881.27	\$3,628.50				\$1,747.23
\$1,063.20	\$2,254.20				\$1,191.00
\$3,468.48	\$10,207.08				\$6,738.60

Each row provides a separate Row Context for evaluating the Margin. In other words, each row has a different value for SalesAmount and TotalCost.

DimChannel DimDate DimGeography DimProduct DimProductCategory FactSales

Record: 1 of 3,406,089

Filter Context

Contoso Sample DAX Formulas - Microsoft Excel

PivotTable Tools: Options, Design

C4 = 769217765.999

Sales	Column Labels	2007	2008	2009	Grand Total
Row Labels					
Asia		\$ 726,887,376.46	\$ 944,715,987.80	\$ 1,028,600,621.89	\$ 2,700,203,986.15
Europe		\$ 959,554,446.45	\$ 769,217,766.00	\$ 697,375,893.85	\$ 2,426,148,106.30
North America		\$ 2,875,499,132.11	\$ 2,397,299,780.89	\$ 2,014,506,603.43	\$ 7,287,305,516.43
Grand Total		\$ 4,561,940,955.02	\$ 4,111,233,534.68	\$ 3,740,483,119.18	\$ 12,413,657,608.89

This PivotTable evaluates a single DAX formula (measure) in sixteen (16) different filter contexts. For example, the highlighted cell has a filter context with two filters: Continent=Europe and Year=2008.

SalesByCountry SalesByPeriod Inventory

Ready 100%

Functionality That Does Not Exist In Excel Formulas

PowerPivot includes capabilities such as (relationships between tables, dynamic aggregation, context modification, etc. that do not exist within Excel, and therefore we need to have DAX functions that provide functionality that goes beyond the scope of Excel formulas. The details of DAX functions are described later, but here are some of the DAX functions categories which don't exist in Excel.

Tables and relationships require these classes of functions:

- Functions that navigate relationships between tables (more powerful than VLOOKUP)
- Functions that take tables as arguments (aggregation over a table, filtering a table, etc.)
- Functions that produce a table result (this result must then be an input to another function)

Dynamic measure aggregation requires these classes of functions:

- Functions that discover the current context for a calculation (e.g. has a specific filter been set?)
- Functions that modify the context for a calculation (e.g. calculate formula for all products or years)
- Functions that know about Time manipulation (e.g. parallel period, previous period, YTD, etc.)

For example, consider this PivotTable showing sales by country for the Current Period, month-to-date (MTD), year-to-date (YTD), Last Year, PriorYearMTD, and PriorYearYTD. These results are calculated using DAX time intelligence functions as well as DAX aggregation functions.

FullDateLabel	Current	MTD	YTD	LastYear	PriorYearMTD	PriorYearYTD
Australia	\$ 106,031.77	\$ 520,730.02	\$ 32,380,669.99	\$ 67,067.44	\$ 381,595.22	\$ 25,428,532.05
Canada	\$ 316,637.57	\$ 978,578.23	\$ 64,924,810.91	\$ 326,814.41	\$ 1,592,639.82	\$ 88,737,924.00
China	\$ 1,852,200.40	\$ 7,384,601.08	\$ 470,675,431.89	\$ 1,570,606.35	\$ 6,358,381.08	\$ 370,551,111.02
Denmark	\$ 12,468.73	\$ 53,223.63	\$ 4,932,953.18	\$ 24,890.82	\$ 143,172.62	\$ 9,202,455.69
France	\$ 568,303.96	\$ 2,105,588.04	\$ 189,577,757.71	\$ 596,111.07	\$ 2,546,849.81	\$ 214,538,710.62
Germany	\$ 895,327.02	\$ 3,612,463.27	\$ 292,349,124.98	\$ 940,203.95	\$ 3,545,995.12	\$ 304,361,113.77
Greece	\$ 7,780.05	\$ 66,916.63	\$ 4,977,762.03	\$ 24,761.09	\$ 105,478.00	\$ 9,118,514.16
Japan	\$ 198,429.73	\$ 988,073.32	\$ 66,053,978.75	\$ 287,436.89	\$ 958,141.97	\$ 43,760,610.48
Russia	\$ 87,822.64	\$ 316,123.60	\$ 26,303,454.92	\$ 133,148.08	\$ 484,013.06	\$ 35,542,427.16
Spain	\$ 13,754.17	\$ 47,370.69	\$ 4,968,537.43	\$ 27,078.60	\$ 134,180.07	\$ 9,260,718.19
Sweden	\$ 6,756.33	\$ 34,156.19	\$ 4,925,025.46	\$ 33,651.54	\$ 85,008.48	\$ 9,066,176.49
Switzerland	\$ 18,151.26	\$ 45,186.75	\$ 4,789,504.07	\$ 28,789.53	\$ 87,957.62	\$ 9,018,584.07
United Kingdom	\$ 278,967.96	\$ 950,174.58	\$ 77,063,395.81	\$ 467,785.54	\$ 1,624,756.49	\$ 129,312,192.62
United States	\$ 7,748,194.23	\$ 30,313,156.29	\$ 1,888,068,614.10	\$ 9,786,995.77	\$ 40,065,133.71	\$ 2,255,486,360.92
Grand Total	\$ 13,038,744.96	\$ 51,498,668.60	\$ 3,380,882,316.81	\$ 15,314,604.06	\$ 61,911,531.18	\$ 3,744,241,734.39

DAX Operators and Constants

DAX supports the usual arithmetic operators (+ - * / ^) for addition, subtraction, multiplication, division, and exponentiation. DAX supports the comparison operators greater than (>), less than (<), greater than or equal to (>=), less than or equal to (<=), equals (=), and not equals (<>). DAX uses the ampersand (&) operator for concatenation. DAX also has logical operators that consist of a double vertical bar (||) for logical OR, and double ampersand (&&) for logical AND. DAX does not support the use of "OR" or "AND" as keywords –

users should use “|” and “&&” for these operations. DAX also supports TRUE and FALSE as logical constants.

Simple DAX Functions

This section focuses on some basic DAX functions that do not require a full understanding of filter context. In subsequent sections of this paper, you will find a more detailed explanation of filter context, followed by a section on the DAX functions that interact with the filter context in various ways.

BLANK() and Blank Values

When a field in a traditional database does not contain a value, the empty field is often referred to as a NULL value. In Excel, an empty cell is referred to as a “blank” cell. While NULL and BLANK are two closely related concepts, it is useful to consider the difference between them.

In databases that support NULL, all calculations involving NULL will return NULL. For example, NULL+3 generates a NULL result. Trying to perform arithmetic on any NULL will always return a NULL. This is useful in databases where there is a desire to call out the special case of a missing value and to differentiate that missing value from a zero in the data.

In Excel, when you have a BLANK cell in a column of data, Excel allows you to perform arithmetic on the cell and simply treats the blank cell as a zero (or an empty string, depending on the operation being performed.) For example, BLANK +3 generates a result of 3. This makes sense where you are trying to aggregate values and wish to ignore any missing values in the list. This treatment is different than is used by most database products for missing values, which are referred to as NULL, and where NULL +3 generates NULL. Since DAX follows the Excel logic, we use the term BLANK for missing values rather than NULL.

Note that there is one place where DAX differs from Excel, and this difference is quite intentional. When we have a formula which is adding up a column of numbers, and the entire column contains nothing but blanks, instead of generating a result of zero (which is what Excel does) DAX and PowerPivot will generate a result of BLANK. As long as there are any values in the column being aggregated, we will get a numeric result, but when all the values are BLANK, the result will also be BLANK.

We needed to have this difference in order to properly work in PivotTables. Imagine a database with a couple of thousand cities, and a set of sales that have been filtered to include only eight (8) of those cities. If we build a PivotTable and place [City] on rows, and [Sales] in values, we want to see a PivotTable with only eight rows. If the sum of blanks was zero (as it is in Excel) then we would get a PivotTable with two thousand rows and mostly zeroes everywhere. Allowing “empty” data sets to aggregate to BLANK is incredibly useful for PivotTables, since PivotTables automatically (by default) filter away any rows with blank results.

DAX has a function BLANK() that returns an empty (blank) value. This is useful when you want to test to see if the result of an expression is blank, because you can compare the expression to BLANK(). As you will see in the next section DAX also supports the Excel ISBLANK() function for the same purpose.

Functions from Excel

DAX supports approximately eighty (80) functions from Excel. There are some specific differences from how these functions work in Excel, but generally these functions are so similar that the learning curve is likely to be very small. The description provided for each of these functions is intentionally brief because these functions are essentially the same as have been in Excel for many years. This document's focus is primarily about DAX functions that are not available in Excel.

In places where the Excel function took a range (or a set of ranges,) the DAX function takes a column reference. For example, in DAX the function SUM doesn't take a list of ranges. It takes a column reference and adds the values in that column.

In places where a function used to return the serial number representing a date, DAX will return the date itself using the date data type.

Logical functions AND and OR only allow for two arguments in DAX. CONCATENATE has the same limitation. To get around this limitation, we recommend using the operators instead of these functions as shown here:

AND	&&
OR	
CONCATENATE	&

Here are the Excel functions supported in DAX by Category:

Date and Time	ISBLANK	EXP	TRUNC
DATE	ISERROR	FACT	
DATEVALUE	ISLOGICAL	FLOOR	
DAY	ISNONTEXT	INT	
EDATE	ISNUMBER	LN	Statistical
EOMONTH	ISTEXT	LOG	AVERAGE
HOUR		LOG10	AVERAGEA
MINUTE	Logical	MOD	COUNT
MONTH	AND	MROUND	COUNTA
NOW	IF	PI	COUNTBLANK
SECOND	IFERROR	POWER	MAX
TIME	NOT	QUOTIENT	MAXA
TIMEVALUE	OR	RAND	MIN
TODAY	FALSE	RANDBETWEEN	MINA
WEEKDAY	TRUE	ROUND	RANK.EQ
WEEKNUM		ROUNDDOWN	STDEV.P
YEAR	Math and Trig	ROUNDUP	STDEV.S
YEARFRAC	ABS	SIGN	VAR.P
	CEILING,	SQRT	VAR.S
Information	ISO.CEILING	SUM	Text
			CONCATENATE

EXACT	LEN	REPT	TRIM
FIND	LOWER	RIGHT	UPPER
FIXED	MID	SEARCH	VALUE
LEFT	REPLACE	SUBSTITUTE	

FORMAT(Value, Format_text)

Instead of Excel's TEXT function, DAX has a FORMAT function for converting various numeric and date values to strings. This function works just like the FORMAT function in Visual Basic.

Start with a field named [TestDate] that is defined as: =DATE (2001, 1, 27) + TIME (17, 4, 23). In other words, we start with a date field that contains this date and time: Jan 27, 2001 5:04:23 PM.

The following examples illustrate how FORMAT can produce different strings from this value:

DAX Formula	Result
=FORMAT([TestDate],"mmm dd yyyy hh:mm:ss")	January 27 2001 17:04:23
=FORMAT([TestDate],"mmm dd yyyy ddd hh:mm")	Jan 27 2001 Sat 17:04
=FORMAT([TestDate],"ddd mmm-dd")	Saturday Jan-27

Functions to Aggregate Expressions – the “X” Functions

DAX implements aggregation functions from Excel including SUM, AVERAGE, MIN, MAX, COUNT, STDEV.P, STDEV.S, VAR.P, VAR.S, but instead of taking multiple arguments (a list of ranges,) they may take only a reference to a column. This can be somewhat limiting, so DAX also adds some new aggregation functions which aggregate any expression over the rows of a table. Here are the functions:

- SUMX (Table, Expression)
- AVERAGEX (Table, Expression)
- COUNTAX (Table, Expression)
- MINX (Table, Expression)
- MAXX (Table, Expression)
- STDEVX.P (Table, Expression)
- STDEVX.S (Table, Expression)
- VARX.P (Table, Expression)
- VARX.S (Table, Expression)

In each case, the first argument is a table over which you want to aggregate an expression, and the second argument is the expression to be aggregated.

Consider this example:

= SUMX(FactSales, [UnitPrice] * [SalesQuantity])

This formula says we should start with the FactSales table, and for each row of that table we should evaluate the expression [UnitPrice] * [SalesQuantity]. Then the results should be added up because we are using the SUMX function. If we had used AVERAGEX, we would be taking the average of all the results.

Consider the following three formulas are precisely identical to each other:

- =SUM(FactSales [SalesQuantity])
- =SUMX(FactSales, [SalesQuantity])
- =SUMX(FactSales, FactSales[SalesQuantity])

The first formula takes the sum of a column, the second formula takes the value from a column for each row of the table and adds them up, and the third formula simply provides a fully qualified name for the column.

These aggregation functions are particularly powerful in measures because the context in which they are evaluated will change (filter) the table over which the aggregation is being performed. For example, if we have a PivotTable with years in the column labels, and stores in the row labels, and then we place this formula:

=SUMX(FactSales, [UnitPrice] * [SalesQuantity]) into a measure, for each cell of the PivotTable, we'll get the subtotal for the sales belonging to a particular year and store.

When we use functions like this in a measure we are effectively aggregating a table of data across many slices of the data. This is precisely the sort of data analysis for which PowerPivot and PivotTables are designed. A formula like this in a calculated column is far less interesting because the data set isn't being sliced (unless you also use RELATEDTABLE, which is covered next.☺).

COUNTROWS(Table)

COUNTROWS is similar to the other COUNT functions (COUNT, COUNTA, COUNTX, COUNTAX, COUNTBLANK) but it takes a table as its argument, and returns the count of rows in that table. For example, the formula =COUNTROWS(FactSales) will return the number of sales transactions in the FactSales table.

RELATED(Column) and RELATEDTABLE(Table)

DAX introduces the functions RELATED and RELATEDTABLE for following relationships and retrieving related data from another table. This is more powerful than VLOOKUP in Excel in a couple of ways. First, VLOOKUP only returns the first match – there is no promise of referential integrity and no assurance there was not another row that would also have matched the lookup. Second, instead of requiring the lookup table to be organized in a particular way, with certain columns on the left-hand side, DAX functions rely on relationships between the two tables, and those relationships may involve any of the columns in a table.

RELATED(Column) follows existing many-to-one relationship(s) from the many side to the one side and returns the single matching value from the other table. Consider these examples:

In the FactSales table, we can add two calculated columns using these formulas:

1. =RELATED(DimStore[StoreName])
2. =RELATED(DimGeography[ContinentName])

These formulas will add two columns to the FactSales table, the first containing the name of each store and the second containing the continent name. The first of these examples follows a single relationship from FactSales to DimStore, while the second formula must navigate two relationships: from FactSales to DimStore and then from DimStore to DimGeography. Authoring this formula requires only that you know the name of the column from which you want to get a value.

The screenshot shows the PowerPivot for Excel window for the file 'Contoso Inventory Values.xlsx'. The 'FactSales' table is selected, and the formula bar shows the formula for 'CalculatedColumn2' as `=RELATED(DimGeography[ContinentName])`. The table has columns: SalesKey, DateKey, channelKey, StoreKey, CalculatedColumn1, CalculatedColumn2, and ProductKey. The data rows show various sales transactions with their corresponding store names and continents.

SalesKey	DateKey	channelKey	StoreKey	CalculatedColumn1	CalculatedColumn2	ProductKey
838	10/11/2008	1	77	Contoso Orange Store	North America	1930
1839	5/1/2008	1	158	Contoso Haverhill Store	North America	1930
6120	10/26/2007	1	3	Contoso Kennewick Store	North America	1930
20762	4/5/2007	1	81	Contoso Baccliff Store	North America	1930
43698	4/16/2007	1	77	Contoso Orange Store	North America	1930
46944	9/17/2009	1	278	Contoso Urumqi Store	Asia	1930
48395	4/24/2007	1	5	Contoso Redmond Store	North America	1930

The bottom of the window shows the 'FactSales' table selected in the 'FactSales' group, with a record count of 6 of 3,406,089.

RELATEDTABLE(Table) follows a relationship in either direction (many-to-one or one-to-many) and returns a table containing all the rows that are related to the current row from the specified table. This is very useful when you want to find all the transactions associated with a particular row of a related table.

Note: this function returns a table and not a scalar value. This means that this function cannot be used by itself to define a calculated column or a measure. Instead this function can only be used to provide an intermediate result that is in turn an argument to another function, such as an aggregation function.

Consider this example: `=SUMX(RELATEDTABLE(FactSales), FactSales[SalesAmount])`

This formula says that we want to first construct a table that contains the rows from FactSales that are related to the current row. RELATEDTABLE(FactSales) assumes we have a current row and there is a relationship between the current table and the FactSales table. Once we have the table containing the related rows from the sales transactions, we will then take the SalesAmount from each row, and then add those sales amounts.

Let's go ahead and place this formula into a calculated column in the DimProduct table. Then let's place the exact same formula with no changes whatsoever into a calculated column in the DimStore table. Finally we will place the same formula into the DimDate table. While we get completely different numbers in each of those three tables, we will see that we're getting essentially the same thing: A total of the sales transactions broken down by each product, or by each store, or for each date.

PowerPivot for Excel - 04 - Contoso Inventory Values.xlsx

Home Design

[CalculatedColumn1] \sum = SUMX(RELATEDTABLE(FactSales), FactSales[SalesAmount])

ProductKey	ProductName	CalculatedColumn1
416	Adventure Works Desktop PC2.33 XD233 Silver	\$3,382,255.74
417	Adventure Works Desktop PC2.30 MD230 Silver	\$9,883,188.52
418	Adventure Works Desktop PC1.60 ED160 Silver	\$10,174,720.5435
423	Adventure Works Desktop PC2.30 MD230 Black	\$10,143,106.6
429	Adventure Works Desktop PC2.30 MD230 Brown	\$9,154,258.036
431	Adventure Works Desktop PC1.80 ED180 Brown	\$18,900,028.71
434	Adventure Works Desktop PC2.30 MD230 White	\$9,503,051.14

DimChannel DimDate DimGeography DimProduct DimProductCategory DimProductSubcategory ...

Record: 1 of 2,517

PowerPivot for Excel - 04 - Contoso Inventory Values.xlsx

Home Design

[CalculatedColumn1] \sum = SUMX(RELATEDTABLE(FactSales), FactSales[SalesAmount])

StoreKey	StoreType	StoreName	CalculatedColumn1
15	Store	Contoso Renton Store	\$22,856,635.2115
17	Store	Contoso Spokane Store	\$22,598,548.465
20	Store	Contoso Englewood Store	\$22,514,748.192
22	Store	Contoso Westminster Store	\$22,238,424.413
25	Store	Contoso Greeley No.1 Store	\$22,463,731.9295
27	Store	Contoso Lafayette Store	\$21,925,370.068
30	Store	Contoso Fort Collins Store	\$23,397,040.109

DimChannel DimDate DimGeography DimProduct DimProductCategory DimProductSubcategory DimStore ...

Record: 1 of 306

The screenshot shows the PowerPivot for Excel window with the following data table:

Dat...	FullDateLabel	DateDescription	CalculatedColumn1	CalendarYear
9/7/2008	2008-09-07	2008/09/07	\$11,290,894.4466	2008
9/13/2008	2008-09-13	2008/09/13	\$12,205,002.9174	2008
9/14/2008	2008-09-14	2008/09/14	\$11,587,079.629	2008
9/20/2008	2008-09-20	2008/09/20	\$11,946,227.665	2008
9/21/2008	2008-09-21	2008/09/21	\$12,208,076.163	2008
9/27/2008	2008-09-27	2008/09/27	\$11,872,826.99	2008
9/28/2008	2008-09-28	2008/09/28	\$11,696,474.007	2008

The formula bar shows: `=SUMX(RELATEDTABLE(FactSales), FactSales[SalesAmount])`

At the bottom, the record count is shown as: Record: 615 of 2,556

The construct of aggregating across a related table of transactions is very common and provides a powerful way to break down transactions across the rows of a related table.

USERELATIONSHIP(Column1, Column2)

USERELATIONSHIP specifies the relationship to be used in a specific calculation. The function uses existing relationships in the model, identifying relationships by their ending point columns.

Multiple relationships can exist between two tables in the data model, however only one active path can exist, directly or indirectly. This function allows you to specify the use of a non-active relationship. Active relationships are used by default in DAX expressions. The USERELATIONSHIP function can only be used in functions that take a filter as an argument, like CALCULATE which will be described later in this paper.

For example: `=CALCULATE([Sales], USERELATIONSHIP(FactSales[ShipDateKey], 'Date'[DateKey]))`

Note: There is no ShipDateKey in the Contoso sample database.

FILTER(Table, Condition) and DISTINCT (Column)

FILTER and DISTINCT are two more DAX functions that return a table of results. Because each of these functions returns a table, they cannot be used as the outermost part of a formula placed in a calculated column or measure. Instead these functions will be used as arguments to other functions, typically aggregation functions.

FILTER (Table, Condition) returns a subset of the specified table where the condition is TRUE. It will always return all of the columns of the specified table, and some subset of the rows in that table. For example, the expression `FILTER (FactSales, RELATED(DimGeography[CityName])="Baltimore")` will return a table containing all

the rows from FactSales that took place in the city of Baltimore. To get the sales total for the city of Baltimore, we would simply use this as the table argument to SUMX as follows:

```
=SUMX(FILTER(FactSales, RELATED(DimGeography[CityName])="Baltimore"), [SalesAmount])
```

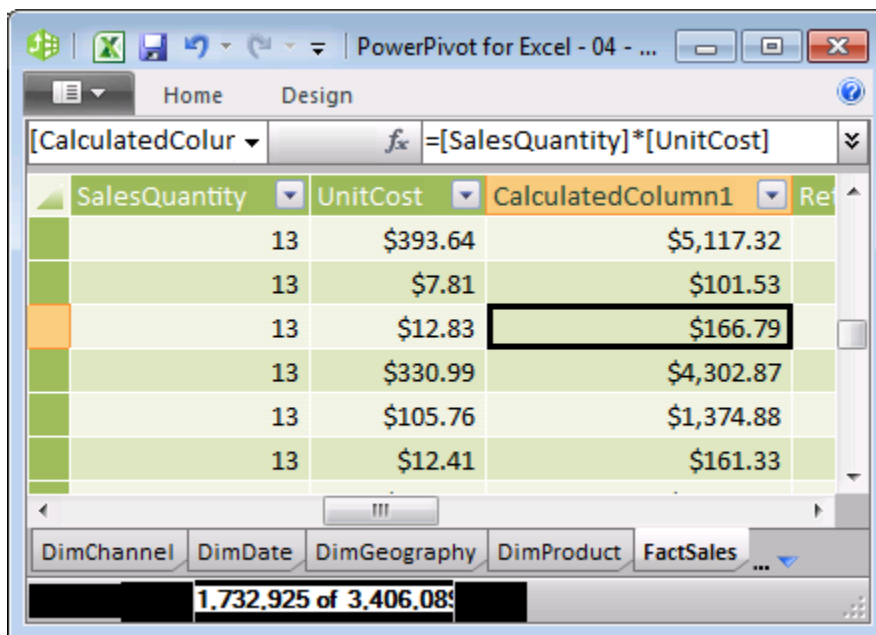
DISTINCT(Column) returns a table containing the unique values within the specified column. In other words, a table of the values, with any duplicate values removed from the list. For example, the expression DISTINCT(FactSales[StoreKey]) will return a table containing the different Store IDs for which there were sales transactions.

Row Context and Filter Context

Now that many of the basic DAX functions have been described, let's look at some examples of what we mean by "Row Context" and "Filter Context" because they represent concepts that you need to understand in order to work with some of the more advanced DAX functions such as CALCULATE, described later in this document.

Row Context

Row context is most easily thought of as the "current row". This is most obvious in cases where a formula is being entered in a calculated column. For example, within the FactSales table, we can author a formula for a calculated column that looks like this: =[SalesQuantity] * [UnitCost]



The screenshot shows the PowerPivot for Excel window. The formula bar at the top displays the formula `= [SalesQuantity] * [UnitCost]` for a calculated column named 'CalculatedColumn1'. Below the formula bar, a table is displayed with the following data:

SalesQuantity	UnitCost	CalculatedColumn1
13	\$393.64	\$5,117.32
13	\$7.81	\$101.53
13	\$12.83	\$166.79
13	\$330.99	\$4,302.87
13	\$105.76	\$1,374.88
13	\$12.41	\$161.33

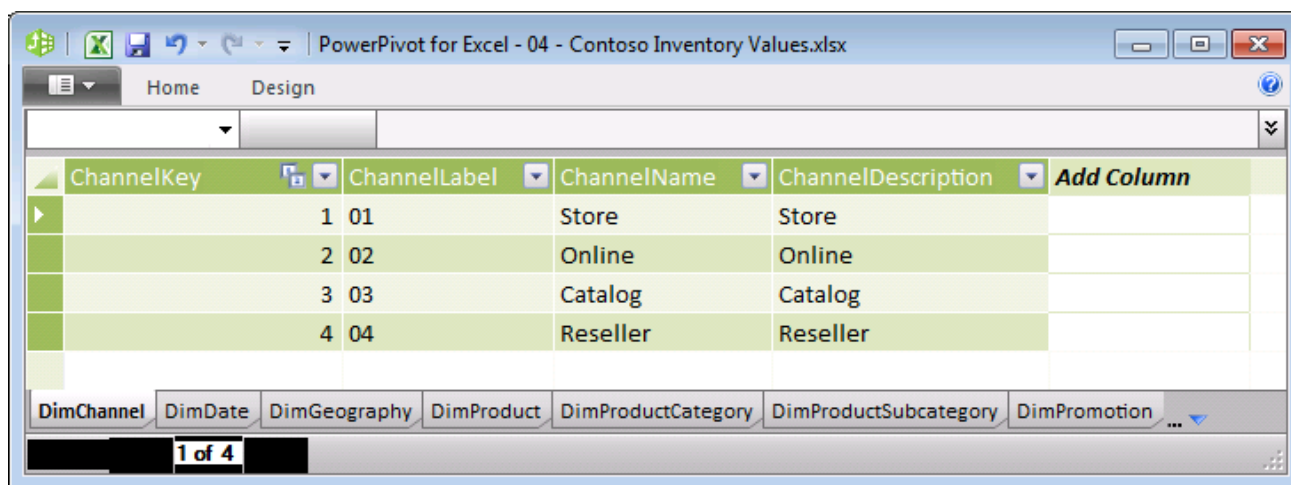
At the bottom of the window, the 'FactSales' table is selected, showing a total of 1,732,925 of 3,406,088 rows.

In this scenario, we are taking the values of the [SalesQuantity] and the [UnitCost] columns from the current row and multiplying the two numbers together. This seems both obvious and intuitive because the same approach works in Excel. But it is important to observe DAX takes the name of a column ([UnitCost]) and interprets it as a single value (\$12.83) when there is a row context. When there is no row context, DAX cannot interpret the name of a column as a value. If we try to enter precisely the same formula in a measure, we will get an error message: "The value for 'SalesQuantity' cannot be determined. Either 'SalesQuantity' doesn't exist, or there is no current

row for a column named 'SalesQuantity'." In this case, the problem is there is no current row (or no row context), so there is no way to know what number to use for [SalesQuantity].

Let's consider a different sort of row context that is based on scanning a table, rather than filling in values in a calculated column. Whenever a formula has a function that scans a table, that function will inherently apply a row context for each row of the table over which it is scanning. One example of this is the SUMX function. You may recall that SUMX takes two arguments – a table over which to aggregate, and an expression to be evaluated for each row in that table. This means that the expression will be evaluated using a row context for each row of the specified table. For example, we can define a measure as follows: =SUMX(FactSales, [SalesQuantity] * [UnitCost]) and the measure will evaluate the total cost for each transaction (in the context of that row) and then add up the results.

Another way to think about row context is to think of it as a set of filters on a table that result in a single row remaining. For example, consider this table:



ChannelKey	ChannelLabel	ChannelName	ChannelDescription	Add Column
1	01	Store	Store	
2	02	Online	Online	
3	03	Catalog	Catalog	
4	04	Reseller	Reseller	

DimChannel DimDate DimGeography DimProduct DimProductCategory DimProductSubcategory DimPromotion ...

1 of 4

The first row of this table is the row where [ChannelKey]=1, [ChannelLabel]="01", [ChannelName]="Store", and [ChannelDescription]="Store". Similarly, the fourth row of this table is the row where [ChannelKey]=4, [ChannelLabel]="04", [ChannelName]="Reseller", and [ChannelDescription]="Reseller". Thinking about applying the filters of each column in a table to identify a single row is a great way to think of row context. Note that in this example, it's enough to filter any one of the columns, but that will not always be the case.

Filter Context

Filter context is more complex than row context and can most easily be described as the set of filters associated with one of the cells in the values area of a PivotTable. Consider the following PivotTable:

04 - Contoso Inventory Values - Microsoft Excel

File Home Insert Page Layout Formulas Data Review View PowerPivot Options Design

C8 58967169.3539

	A	B	C	D	E
1	ChannelName	Store			
2	ContinentName	Europe			
3					
4	Sales	Column Labels			
5	Row Labels	2007	2008	2009	Grand Total
6	Denmark	\$10,711,449.83	\$5,693,881.55	\$5,130,625.49	\$21,535,956.86
7	France	\$64,294,935.44	\$38,294,242.87	\$33,245,400.64	\$135,834,578.94
8	Germany	\$98,916,125.19	\$58,967,169.35	\$53,134,680.49	\$211,017,975.03
9	Greece	\$10,662,152.52	\$5,744,113.96	\$5,049,809.47	\$21,456,075.94
10	Holland	\$10,700,622.10	\$5,916,178.50	\$4,781,201.86	\$21,398,002.46
11	Ireland	\$10,570,530.05	\$5,916,233.23	\$4,962,315.77	\$21,449,079.05
12	Italy	\$34,719,790.81	\$23,229,095.53	\$20,541,708.18	\$78,490,594.52
13	Malta	\$10,572,691.21	\$5,781,747.66	\$4,921,696.16	\$21,276,135.02
14	Poland	\$10,733,699.68	\$6,023,341.08	\$4,693,341.12	\$21,450,381.88
15	Portugal	\$10,629,143.02	\$5,911,127.59	\$4,910,978.97	\$21,451,249.58
16	Romania	\$10,900,048.39	\$5,827,158.89	\$4,950,364.81	\$21,677,572.09
17	Russia	\$42,189,724.14	\$31,289,351.16	\$27,870,158.03	\$101,349,233.33
18	Slovenia	\$10,552,250.44	\$5,702,422.82	\$4,862,182.32	\$21,116,855.58
19	Spain	\$10,775,770.69	\$5,713,579.13	\$4,896,696.91	\$21,386,046.72
20	Sweden	\$10,511,963.13	\$5,688,176.23	\$4,817,243.12	\$21,017,382.48
21	Switzerland	\$10,515,133.82	\$5,496,527.34	\$4,785,769.58	\$20,797,430.75
22	United Kingdom	\$151,571,280.23	\$89,253,182.30	\$73,230,693.92	\$314,055,156.45
23	Grand Total	\$519,527,310.66	\$310,447,529.20	\$266,784,866.83	\$1,096,759,706.69

PowerPivot Field List

Choose fields to add to report:

Search

- DimChannel
- DimDate
- DimGeography
- DimProduct
- DimProductCategory
- DimProductSubcategory
- DimPromotion
- DimStore
- FactSales
- FactInventory

Slicers Vertical Slicers Horizontal

Report Filter

- ChannelName
- ContinentName

Column Labels

- CalendarYear

Row Labels

- RegionCountryName

Σ Values

- Sales

This PivotTable has only one measure, with a single DAX formula defined as `=SUM(FactSales[SalesAmount])`. That single formula is being evaluated 72 distinct times, with 72 different results, and each of those 72 results has a distinct **Filter Context**. The context for each cell in the values area includes the items on the row labels, the column labels, the report filter, (and slicers if they were being used.) The cell that is highlighted above has the following **Filter Context**.

- [RegionCountryName] = "Germany"
- [CalendarYear] = 2008
- [ChannelName] = "Store"
- [ContinentName] = "Europe"

This can be thought of as applying four filters to the FactSales table prior to evaluating the formula, which is simply the sum of the [SalesAmount] values for the rows that are left after applying those four filters. This is where the name "Filter Context" comes from.

Relationships and Filter Context

When there is a relationship between two tables in PowerPivot, that relationship is always considered a "many-to-one" relationship that goes from the table containing "many" records for each instance of the key to the table containing "one" record for each instance of the key. For example, there is a many-to-one relationship

from the table FactSales to the table DimProducts, with many rows for each product in FactSales, and only one row for each product in DimProducts.

It turns out that applying a filter to a table on the “one” side of the relationship will also cause the table on the “many” side of the relationship to be filtered. For example, if we filter DimProducts so only one product remains, then FactSales will also be filtered so it only contains the sales transactions for that one product. Applying a filter to the table on the “many” side of a relationship (filter FactSales to include only certain products) will not have any impact on the table on the “one” side of the relationship.

This is how PivotTables are designed to work, and if you look at some examples, it makes perfect sense. Consider the PivotTable below where DimGeography has been filtered to include only two countries. It is quite natural that FactSales is constrained by the same filter, even though [RegionCountryName] isn’t a field in the FactSales table. It is a field in a table that is on the “one” side of a relationship to FactSales.

04 - Contoso Inventory Values - Microsoft Excel

File Home Insert Page Layout Formulas Data Review View PowerPivot Options Design

A4 fx Row Labels

	A	B	C	D	E	F
1	ChannelName	Store				
2						
3	Sales	Column Labels				
4	Row Labels	2007	2008	2009	Grand Total	
5	Canada	\$108,076,080.27	\$79,570,342.19	\$63,002,636.49	\$250,649,058.95	
6	Denmark	\$10,711,449.83	\$5,693,881.55	\$5,130,625.49	\$21,535,956.86	
7	Grand Total	\$118,787,530.10	\$85,264,223.74	\$68,133,261.97	\$272,185,015.81	
8						
9						

PowerPivot Field List

Choose fields to add to report:

Slicers Vertical Slicers Horizontal

Report Filter Column Labels

ChannelName CalendarYear

Row Labels Values

RegionCountryName Sales

Measures and Filter Context

The formula that defines a measure is always evaluated multiple times – once for each cell in the values area. And each of those evaluations has its own filter context. Even the grand total in the PivotTable above for all countries and for all years is evaluated in its own context, namely the context where [RegionCountryName] = ALL, [CalendarYear] = ALL, and [ChannelName] = “Store”.

The presence of filter context for all measure evaluations (and the presence of row context for calculated columns), is one of the reasons why you often cannot use the same formula to define a measure as you would use to define a calculated column. Generally, because a measure must be evaluated many times, including for any totals rows or columns, measure formulas must do some sort of aggregation. Most measure formulas will either be the aggregation (Sum, Count, Average, Min, Max, or DistinctCount) of an expression, or a formula involving other measures, or some combination of the two.

More DAX Functions

CALCULATE(Expression, SetFilter1, SetFilter2,...)

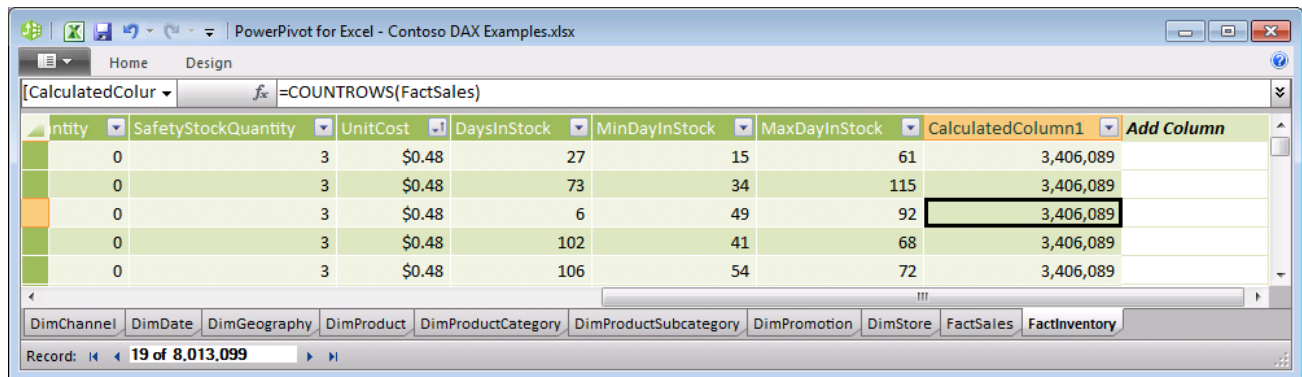
The CALCULATE function is very powerful and very useful, but is conceptually a bit more difficult than any of the other DAX functions described up to this point. CALCULATE allows any DAX expression to be evaluated in a specified filter context. This is equivalent to defining a measure and also defining a set of filters that will be used to provide a context in which the measure is to be evaluated. CALCULATE does the following:

1. Using the SetFilter arguments, it modifies the filter context
2. If there is a row context, it moves that row context onto the filter context
3. It evaluates the Expression in the newly modified filter context

Here are some examples that illustrate how this works.

Example1 – CALCULATE Without Any SetFilter Arguments

Using the sample data from the Contoso database, let's start by adding a calculated column to the FactInventory table with this formula: =COUNTROWS(FactSales)

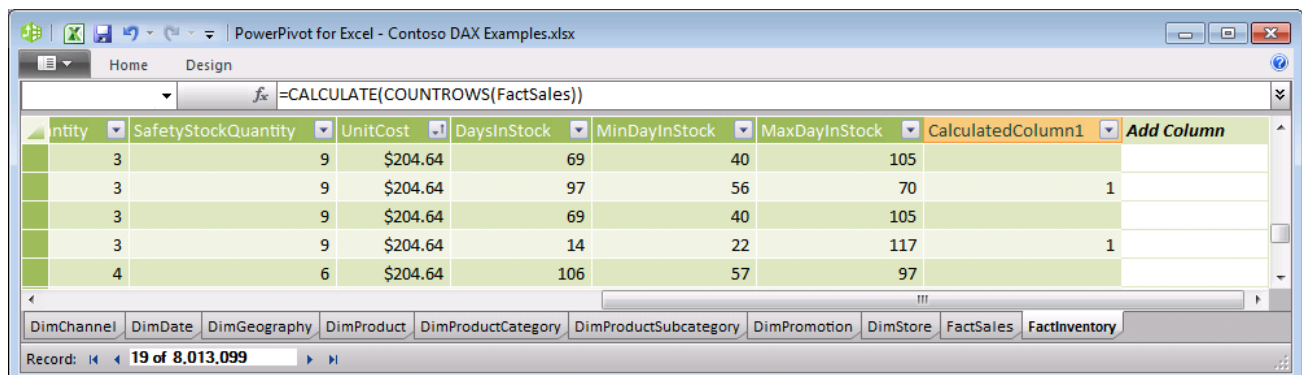


The screenshot shows the PowerPivot for Excel interface. The formula bar displays the formula `=COUNTROWS(FactSales)`. The FactInventory table is shown with columns: Intity, SafetyStockQuantity, UnitCost, DaysInStock, MinDayInStock, MaxDayInStock, and CalculatedColumn1. The CalculatedColumn1 column contains the value 3,406,089 for all rows. The bottom status bar indicates "Record: 19 of 8,013,099".

Intity	SafetyStockQuantity	UnitCost	DaysInStock	MinDayInStock	MaxDayInStock	CalculatedColumn1
0	3	\$0.48	27	15	61	3,406,089
0	3	\$0.48	73	34	115	3,406,089
0	3	\$0.48	6	49	92	3,406,089
0	3	\$0.48	102	41	68	3,406,089
0	3	\$0.48	106	54	72	3,406,089

We should not be surprised when we get the number of rows (approx 3.4 million) in the FactSales table, and we get the same result for every cell in the new calculated column.

Now let's change the formula to this: `=CALCULATE(COUNTROWS(FactSales))`. Observe that we get two different values in the calculated column, either a blank or the value 1. Why did this happen?



The screenshot shows the PowerPivot for Excel interface. The formula bar displays the formula `=CALCULATE(COUNTROWS(FactSales))`. The FactInventory table is shown with columns: Intity, SafetyStockQuantity, UnitCost, DaysInStock, MinDayInStock, MaxDayInStock, and CalculatedColumn1. The CalculatedColumn1 column contains the value 1 for some rows and is blank for others. The bottom status bar indicates "Record: 19 of 8,013,099".

Intity	SafetyStockQuantity	UnitCost	DaysInStock	MinDayInStock	MaxDayInStock	CalculatedColumn1
3	9	\$204.64	69	40	105	
3	9	\$204.64	97	56	70	1
3	9	\$204.64	69	40	105	
3	9	\$204.64	14	22	117	1
4	6	\$204.64	106	57	97	

First, CALCULATE caused the current row context to be placed onto the filter context. This means that the filter context got a set of filters that look like this:

- [InventoryKey] = <value in this row>
- [DateKey] = <value in this row>
- [StoreKey] = <value in this row>
- [ProductKey] = <value in this row>
- [CurrencyKey] = <value in this row>
- [OnHandQuantity] = <value in this row>
- [OnOrderQuantity] = <value in this row>
- Etc.... for every field in the FactInventory table (every field in the row context)

What makes this a bit more complicated is the row context is not merely the current row from that table, but actually the current row from that table as well as the current row from all tables on the one side of a many-to-one relationship from that table. Until now, it has been simpler to think of the row context as being from a single table, but it actually includes the current row from the entire set of related tables. This means the row context is the current row from FactInventory, as well as the current row from DimDate, the current row from DimProduct, and the current row from DimStore. This means the filter context also got a set of filters that look like this:

- [DateKey] = <value in the related row in DimDate>
- [FullDateLabel] = <value in the related row in DimDate>
- Etc.... for every field in the DimDate table
- [ProductKey] = <value in related row in DimProduct>
- [ProductLabel] = <value in related row in DimProduct>
- Etc.... for every field in the DimProduct table
- [StoreKey] = <value in related row in DimStore>
- [StoreType] = <value in related row in DimStore>
- Etc.... for every field in the DimStore table

Now, using that modified filter context, we want to count the rows in the FactSales table using the modified filter context. This is just like performing a calculation for a measure in a PivotTable where all of these filters have been placed on the PivotTable. Note that many of the fields above do not have any impact on the FactSales table, but some of them do. The filters from the DimDate, DimStore, and DimProduct tables are all fields for which there is a relationship to FactSales, so this filter context has effectively filtered the FactSales table down to the rows that have a specific date, store, and product. It turns out with this data set, the filtered FactSales table has either no rows, or a single row. Therefore the formula results in a value of BLANK or 1 depending on the values in the FactInventory row.

Example2 – SetFilter Arguments

The SetFilter argument is used to modify the context in which an expression is going to be evaluated.

Let's say we want to do some analysis on sales data. Let's define a simple measure on the FactSales table named [Sales] that is defined by this DAX formula: =SUM(FactSales[SalesAmount]). [Sales] is nothing more than the sum of the [SalesAmount] field for all the sales transactions contained in any particular slice of the data. But there's a twist because the Contoso sales data includes sales made via four distinct channels: There are "Catalog" sales, "Online" sales, "Reseller" sales, and "Store" sales. Each of these is a separate row in the DimChannel table, and every FactSales row is identified as belonging to one of these channels by the [ChannelKey] field in that table. (There is of course a many-to-one relationship from FactSales to DimChannel.)

Suppose the sales analysis we want to perform is to be based on Store Sales worldwide. We expect various people in our department to slice the sales data a variety of ways – by country, by time, by store, by product, and by any of the fields available in the database. But we want to make sure that no matter what they choose, they are only looking at sales in the "Store" channel. If we don't do anything special, we run the risk the analysis will inadvertently combine store sales with other kinds of sales, because they might forget to place a Channel filter on their PivotTable.

Note this would be an easy mistake to make because the sales data for Contoso includes Store sales in 34 countries. In 30 of those countries, Store sales is all there is. In this data set, Catalog sales are limited to the United States, Online sales are limited to three countries (US, Germany, China), and Reseller sales are also limited to three countries (US, France, China). Someone analyzing sales in Canada would not see any difference between Store Sales and Sales for all channels, but someone analyzing data for the US or China would see a dramatic difference.

To force the values to be from the "Store" channel, we will define a new measure named [StoreSales] which is defined as: =CALCULATE([Sales], DimChannel[ChannelName] = "Store"). The SetFilter argument within CALCULATE ensures that it will override any filter (if any) on the Channel Name column with a new filter that says we only want sales in the Store channel. By using this measure, we are guaranteed to be looking at Store sales, and we don't have to rely on PivotTable authors to place the Channel somewhere on their PivotTable.

The screenshot shows the Microsoft Excel interface with the 'Contoso DAX Examples' workbook. A PivotTable is displayed with 'StoreSales' as the data source. The PivotTable has 'Row Labels' (countries) and 'Column Labels' (years 2007, 2008, 2009, and Grand Total). The 'Measure Settings' dialog box is open, showing the formula `=CALCULATE([Sales], DimChannel[ChannelName]="Store")` and the 'Currency' category selected for formatting. The 'PowerPivot Field List' pane on the right shows the available fields, including 'StoreSales' and 'FactInventory'.

Row Labels	2007	2008	2009	Grand Total
Armenia	\$10,697,642.89	\$12,802,000.68	\$13,768,349.90	\$37,267,993.47
Australia	\$31,698,751.16	\$39,734,710.75	\$41,128,813.37	\$112,562,275.29
Bhutan	\$11,011,949.91	\$12,899,605.66	\$19,962,257.73	\$43,873,813.31
Canada	\$108,076,080.27	\$79,570,342.19	\$63,002,636.49	\$250,649,058.95
China	\$51,524,018.51	\$71,923,190.97	\$86,167,763.67	\$209,614,973.15
Denmark	\$10,711,449.83	\$5,693,881.55	\$5,130,625.49	\$21,535,956.86
France	\$64,294,935.44	\$38,294,242.87	\$33,245,400.64	\$135,834,578.94
Germany	\$98,916,125.19	\$58,967,169.35	\$53,134,680.49	\$211,017,975.03
Greece	\$10,662,152.52	\$5,744,113.96	\$5,049,809.47	\$21,456,075.94
Holland	\$10,700,622.10	\$5,916,178.50	\$4,781,201.86	\$21,398,002.46
India	\$31,834,758.18	\$39,621,943.29	\$40,462,377.31	\$111,919,078.79
Iran	\$21,314,190.11	\$26,804,288.38	\$27,098,480.82	\$75,216,959.32
Ireland	\$10,570,530.05	\$5,916,233.23	\$4,962,315.77	\$21,449,079.05
Grand Total	\$2,783,180,890.20	\$2,228,460,305.60	\$1,931,211,999.35	\$6,942,853,195.14

Syntax Shortcut for CALCULATE(Expression [,SetFilter1] [,SetFilter2]...)

When you have a CALCULATE function where the expression to be evaluated is simply a measure, then there is a convenient shortcut available. Let's consider the example above.

`= CALCULATE([Sales], DimChannel[ChannelName] = "Store")`

In order to make certain common cases easier to author and easier to read, we offer the following shortcut:

When the expression to be evaluated is a measure, a shortcut for `CALCULATE(<measure>, <optional SetFilters>)` is to use the name of the measure as if it were a function name and write this instead:

=<MeasureName>(<optional SetFilters>)

When we apply this to our example, we find that StoreSales can be defined as:

=[Sales](DimChannel[ChannelName] = "Store")

This is easier to read and author and we will use this notation in several places going forward.

More Information About SetFilter Arguments

A SetFilter argument in CALCULATE may be either of the following:

1. A boolean expression that refers to a single column such as ***DimChannel [ChannelName] = "Store"*** or ***(DimGeography[CityName] = "Seattle" || DimGeography[CityName] = "Portland")***. Note that the boolean expression may not refer to measures and may not invoke the CALCULATE function nor any table scanning functions including aggregation functions or table valued functions.
2. A table expression such as ***Filter(Table1, Condition)***
Let's consider how a table SetFilter should be interpreted. Consider a table which contains three columns named "City", "Product", and "Month". The rows in the table specify the valid combinations of values for those columns. Any combination of values not present in the table is filtered away. In other words, we can pass a table like this as a SetFilter argument to the CALCULATE function, and it will filter away any combinations of city, product, and month that aren't in this table.

[City]	[Product]	[Month]
Seattle	Bike	Jan 2008
Seattle	Car	Feb 2008
Boston	Bike	Mar 2008

The most common scenario for using a table as a SetFilter argument to the CALCULATE function is the ALL function which will be illustrated later in this paper.

VALUES(Column)

The VALUES function will return all the valid values for that column in the current filter context. The result comes back as a table of values, even though the table may have only one value (or even no values). This function is an easy way to ask about the current filter context. This function will often be used within an IF function. Imagine a scenario where you want to do one calculation when the country is the United States, and a different calculation when the country is not the United States. This can be accomplished by testing to see if ***VALUES(DimGeography[RegionCountryName]) = "United States"***. Of course there might be a PivotTable where the country isn't specified, or a situation where multiple countries have been selected and we can only compare a table to a single value when the table has only one value.

This forces us to write something like this to determine what the current country is:

```
=IF(HASONEVALUE(DimGeography[RegionCountryName]), VALUES(DimGeography[RegionCountryName]), "No single country selected")
```

The HASONEVALUE function returns TRUE when the context for the column has been filtered down to one distinct value only. This function is very convenient for testing whether the VALUES function will return a single value in which case it can be referenced as a scalar value.

This formula says that if there is only one country selected in the current context, return the name of that country, else return the expression “No single country selected”. The result of placing this measure in a PivotTable where the country is on Row Labels looks like this:

The screenshot displays the Microsoft Excel interface with a PivotTable and the Measure Settings dialog box open.

PivotTable Data:

Row Labels	Measure 1
Armenia	Armenia
Australia	Australia
Bhutan	Bhutan
Canada	Canada
China	China
Denmark	Denmark
France	France
Germany	Germany
Greece	Greece
Holland	Holland
India	India
Iran	Iran
Ireland	Ireland
Italy	Italy
Japan	Japan
Kyrgyzstan	Kyrgyzstan
Malta	Malta
Pakistan	Pakistan
Poland	Poland
Portugal	Portugal
Romania	Romania
Russia	Russia
Singapore	Singapore
Slovenia	Slovenia
South Korea	South Korea
Spain	Spain
Sweden	Sweden
Switzerland	Switzerland
Syria	Syria
Taiwan	Taiwan
Thailand	Thailand
Turkmenistan	Turkmenistan
United Kingdom	United Kingdom
United States	United States
Grand Total	No single country selected

PowerPivot Field List:

- DimChannel
- DimDate
- DimGeography
 - GeographyKey
 - GeographyType
 - ContinentName
 - CityName
 - StateProvinceName
 - ☒ RegionCountryName
 - ETLLoadID
 - LoadDate
 - UpdateDate

Measure Settings Dialog Box:

- Table name: DimGeography
- Measure name (all PivotTables): Measure 1
- Custom name (this PivotTable): Measure 1
- Description:
- Formula: `=IF(HASONEVALUE(DimGeography[RegionCountryName]),
VALUES(DimGeography[RegionCountryName]),
"No single country selected")`
- Formatting Options:
 - Category: General (selected), Number, Currency, Date, TRUE/FALSE
- Buttons: OK, Cancel

CALCULATETABLE(TableExpression, SetFilter1, SetFilter2,...)

CALCULATETABLE is just like CALCULATE, except the expression and the result are both tables. This would be used when you want to change the filter context and then construct a table in that new context.

ALL(Table) and ALL(Column1 [,Column2]...)

The ALL function is a table-valued function which causes the filter context for the specified columns to be ignored. This function is particularly useful when used as a table-valued SetFilter argument within the CALCULATE function. Consider some examples:

ALL(DimProduct) returns a table containing all the rows within DimProduct, ignoring any filters that may be present in the filter context. Duplicate rows will be removed. Because the argument to ALL is a table, the filter context is effectively removed from all columns in that table.

ALL(Column1) returns a table containing all the values from Column1 (without any duplicates) after removing/ignoring any filters from the filter context that may have been present on that column.

When multiple column arguments are supplied to the ALL function, it is required that all of those columns belong to the same table. For example, ALL (Column1, Column2) returns a table with two columns from a source table, consisting of all the combinations of Column1 and Column2 that were present in the original data removing/ignoring any filters from the filter context that may have been present on those columns.

Consider the measure FactSales[Sales] that has been defined as =SUM(FactSales[SalesAmount]). By itself, the measure [Sales] is the sum of the SalesAmount column for whatever the current context may be. Looking at the PivotTable below, you see that this is a different number in each cell of the PivotTable because each cell has a different context (a different product category and year).

The screenshot shows a Microsoft Excel window titled 'Contoso DAX Examples - Microsoft Excel'. The 'PivotTable Tools' ribbon is active, showing 'Options' and 'Design' tabs. The PivotTable is titled 'Sales' and is located in the range A1:E11. The PivotTable has 'Sales' as the PivotTable Name and 'Column Labels' as the field name. The PivotTable is structured with 'Row Labels' as the row field and '2007', '2008', '2009', and 'Grand Total' as the column fields. The data is as follows:

Row Labels	2007	2008	2009	Grand Total
Audio	\$29,734,671.91	\$52,932,396.05	\$68,947,296.35	\$151,614,364.31
Cameras and camcorders	\$1,102,693,917.48	\$817,903,832.17	\$641,426,024.41	\$2,562,023,774.06
Cell phones	\$363,847,591.08	\$254,833,662.07	\$273,552,011.15	\$892,233,264.30
Computers	\$1,146,469,996.57	\$990,173,504.69	\$1,072,783,640.15	\$3,209,427,141.42
Games and Toys	\$42,429,666.08	\$41,929,029.20	\$65,337,761.59	\$149,696,456.86
Home Appliances	\$1,375,117,996.81	\$1,426,891,288.45	\$1,120,727,501.92	\$3,922,736,787.19
Music, Movies and Audio Books	\$74,975,760.83	\$53,303,972.44	\$37,524,972.71	\$165,804,705.98
TV and Video	\$426,671,354.26	\$473,265,849.61	\$460,183,910.90	\$1,360,121,114.76
Grand Total	\$4,561,940,955.02	\$4,111,233,534.68	\$3,740,483,119.18	\$12,413,657,608.89

Now, let's add another measure to the PivotTable. Let's add RatioAllProd which is the sales for a given context divided by sales for all products (in the same year). There are two ways to write this, one using CALCULATE, and another using our shortcut syntax:

- `= [Sales] / CALCULATE([Sales], ALL(DimProduct))`
- `= [Sales] / [Sales](ALL(DimProduct))`

Note that by removing/ignoring the filter context on DimProducts, we are also removing/ignoring the filter context on all tables on the "one" side of a relationship with DimProducts (SubCategory and Category tables).

	2007		2008		2009	
Row Labels	Sales	RatioAllProd	Sales	RatioAllProd	Sales	RatioAllProd
Audio	\$29,734,671.91	0.7%	\$52,932,396.05	1.3%	\$68,947,296.35	1.8%
Cameras and camcorders	\$1,102,693,917.48	24.2%	\$817,903,832.17	19.9%	\$641,426,024.41	17.1%
Cell phones	\$363,847,591.08	8.0%	\$254,833,662.07	6.2%	\$273,552,011.15	7.3%
Computers	\$1,146,469,996.57	25.1%	\$990,173,504.69	24.1%	\$1,072,783,640.15	28.7%
Games and Toys	\$42,429,666.08	0.9%	\$41,929,029.20	1.0%	\$65,337,761.59	1.7%
Home Appliances	\$1,375,117,996.81	30.1%	\$1,426,891,288.45	34.7%	\$1,120,727,501.92	30.0%
Music, Movies and Audio Books	\$74,975,760.83	1.6%	\$53,303,972.44	1.3%	\$37,524,972.71	1.0%
TV and Video	\$426,671,354.26	9.4%	\$473,265,849.61	11.5%	\$460,183,910.90	12.3%
Grand Total	\$4,561,940,955.02	100.0%	\$4,111,233,534.68	100.0%	\$3,740,483,119.18	100.0%

There is one important thing to note about this formula: `= [Sales] / [Sales](ALL(DimProduct))`. We have changed the filter context for the denominator of this ratio without changing it for the numerator. This is a very common BI scenario. We need to calculate something compared to that same thing but for a different set of products, or for a different year, or for a different region, etc. Changing some part of the filter context for some portion of the formula is very powerful.

Let's look at one more example, this time comparing sales for each of our four channels to store sales and also to all sales. We'll define two new measures:

- `RatioStore: = [Sales] / [StoreSales]`
- `RatioAllChan: = [Sales] / [Sales](ALL(DimChannel))`

Then I'll place these measures in a couple of different PivotTables that have different row and column headers to illustrate how a measure can make sense no matter what PivotTable it is placed on. This makes it important to define measures carefully, because once defined, users are free to place them on PivotTables that might be organized quite differently from what was originally anticipated.

Contoso DAX Examples - Microsoft Excel

PivotTable Tools: Options, Design

Column Labels: 2007, 2008, 2009

Row Labels: Sales, RatioStore, RatioAllChan

	2007	2008	2009
Catalog	\$418,176,088.51	\$345,528,219.49	\$314,303,239.23
Online	\$789,379,825.21	\$937,069,110.93	\$951,150,098.93
Reseller	\$571,204,151.10	\$600,175,898.67	\$543,817,781.67
Store	\$2,783,180,890.20	\$2,228,460,305.60	\$1,931,211,999.35
Grand Total	\$4,561,940,955.02	\$4,111,233,534.68	\$3,740,483,119.18

Contoso DAX Examples - Microsoft Excel

Column Labels: Online, Reseller

Row Labels: Sales, RatioStore, RatioAllChan

	Online	Reseller
China	\$887,049,174.43	\$563,941,179.69
France	\$806,300,456.39	\$523,087,943.23
Germany	\$984,249,404.25	\$628,168,708.52
United States	\$2,677,599,035.07	\$1,715,197,831.44
Grand Total	\$2,677,599,035.07	\$1,715,197,831.44

ALLEXCEPT(Table [,Column1] [,Column2]...)

ALLEXCEPT is a shorthand notation for a series of ALL functions. Imagine a table that has 43 columns, and let's say you want to remove/ignore the filter context from 40 of those columns, and retain the filter context on three of them. You can accomplish this in two ways:

- ALL(Column1, Column2,... <repeated for 40 columns>) removes the context for the forty columns specified here.
- ALLEXCEPT(Table, Column1, Column2, Column3) removes the context for all the columns in the specified table except the three columns that are specified here.

This is nothing more than a convenient shortcut for specific situations where you want to remove/ignore the context on many (but not all) columns in a table.

RANKX(Table, Expression [,Value] [,Order] [,Ties]...)

The RANKX function returns a ranking of a number in a list of numbers for each row in the Table argument. The function can optionally take a Value argument that represents a scalar value whose rank is to be found. The optional Order argument specifies how to rank Value, descending (0) or ascending (1). The optional Ties argument defines how to determine ranking when there are ties. Skip (default) will use the next rank value after a tie, and Dense will use the next rank value (i.e. there will be no gaps in the rank numbers).

The following example illustrates using the RANKX function to rank the sales by product color. The two measure expressions are:

- Sales:=SUM(FactSales[SalesAmount])
- ColorRank:=RANKX(ALL(DimProduct[ColorName]), [Sales])

	A	B	C
1	Row Labels	Sales	ColorRank
2	Azure	\$51,328,936.53	13
3	Black	\$2,721,015,591.94	1
4	Blue	\$1,077,572,553.46	5
5	Brown	\$580,192,494.76	6
6	Gold	\$214,798,389.83	11
7	Green	\$545,364,759.33	7
8	Grey	\$1,243,103,636.21	4
9	Orange	\$374,884,081.18	9
10	Pink	\$274,631,376.67	10
11	Purple	\$4,245,247.46	15
12	Red	\$478,436,368.89	8
13	Silver	\$2,152,354,742.10	3
14	Silver Grey	\$121,228,292.98	12
15	Transparent	\$1,173,767.25	16
16	White	\$2,533,782,878.17	2
17	Yellow	\$39,544,492.14	14
18	Grand Total	\$12,413,657,608.89	1

In this example the RANKX function compares the value of the Sales measures for each row in the PivotTable against the values of the Sales measure for all rows of DimProduct[ProductColor] thus producing the rank for the current row.

When the PivotTable rows are sorted by Sales, the rank calculation is clearer.

	A	B	C
1	Row Labels	Sales	ColorRank
2	Black	\$2,721,015,591.94	1
3	White	\$2,533,782,878.17	2
4	Silver	\$2,152,354,742.10	3
5	Grey	\$1,243,103,636.21	4
6	Blue	\$1,077,572,553.46	5
7	Brown	\$580,192,494.76	6
8	Green	\$545,364,759.33	7
9	Red	\$478,436,368.89	8
10	Orange	\$374,884,081.18	9
11	Pink	\$274,631,376.67	10
12	Gold	\$214,798,389.83	11
13	Silver Grey	\$121,228,292.98	12
14	Azure	\$51,328,936.53	13
15	Yellow	\$39,544,492.14	14
16	Purple	\$4,245,247.46	15
17	Transparent	\$1,173,767.25	16
18	Grand Total	\$12,413,657,608.89	1

RANK.EQ(Value, ColumnName [,Order])

Returns a ranking of a number in a list of numbers with its size relative to other values in the list. The last argument is an optional value (0 or 1) that specifies how to rank number, low to high (ascending) or high to low (descending). By default the order is descending (0).

The following example is a calculated column representing the rank of the each product (row) in descending order.

[PriceRank]		fx =RANK.EQ([UnitPrice], [UnitPrice])	
UnitPrice	PriceRank	AvailableForSaleDate	
\$0.95	2514	8/23/2007 12:00:00 AM	
\$0.95	2514	8/23/2007 12:00:00 AM	
\$0.95	2514	8/23/2007 12:00:00 AM	
\$0.95	2514	8/23/2007 12:00:00 AM	
\$1.99	2510	3/15/2008 12:00:00 AM	
\$1.99	2510	3/15/2008 12:00:00 AM	
\$1.99	2510	3/15/2008 12:00:00 AM	
\$1.99	2510	3/15/2008 12:00:00 AM	
\$2.94	2509	3/4/2007 12:00:00 AM	
\$3.35	2506	7/28/2006 12:00:00 AM	

This example is the same calculation, except the Order argument is set to 1 (ascending).

[PriceRank]		f_x	=RANK.EQ([UnitPrice], [UnitPrice], 1)
UnitPrice	PriceRank	AvailableForSaleDate	
\$0.95	1	8/23/2007 12:00:00 AM	
\$0.95	1	8/23/2007 12:00:00 AM	
\$0.95	1	8/23/2007 12:00:00 AM	
\$0.95	1	8/23/2007 12:00:00 AM	
\$1.99	5	3/15/2008 12:00:00 AM	
\$1.99	5	3/15/2008 12:00:00 AM	
\$1.99	5	3/15/2008 12:00:00 AM	
\$1.99	5	3/15/2008 12:00:00 AM	
\$2.94	9	3/4/2007 12:00:00 AM	
\$3.35	10	7/28/2006 12:00:00 AM	

TOPN(N_Value, Table [,OrderBy_Expression1] [,Order1] [,OrderBy_Expression2] [,Order2]...)

The TOPN function returns the top N rows of a specific table, optionally ordered by expressions, ascending or descending. The Expression arguments can be based on columns or measures. The argument following the OrderBy_Expression is a value (0 or 1) that specifies how to order the table. 0 sorts in descending order of values of the OrderBy_Expression, and 1 sorts them in ascending order.

As the function returns a table, it will need to be aggregated when used in a calculated column or measure. The following examples will sum the table returned by the TOPN function to calculate the top three and bottom three products by sales.

The three measure expressions are:

- Sales =SUM(FactSales[SalesAmount])
- Top3ProductSales =SUMX(TOPN(3, DimProduct, [Sales]), [Sales])
- Bottom3ProductSales =SUMX(TOPN(3, DimProduct, [Sales], 1), [Sales])

In the following example, the PivotTable presents Sales by StockTypeName on the rows, and the three measures Sales, Top3ProductSales and Bottom3ProductSales as Values.

	A	B	C	D
1	Row Labels	Sales	Top3ProductSales	Bottom3ProductSales
2	High	\$6,734,689,763.77	\$106,075,463.69	\$131,925.19
3	Low	\$2,579,528,111.57	\$86,256,616.55	\$533,626.60
4	Mid	\$3,099,439,733.55	\$152,651,739.87	\$46,696.90
5	Grand Total	\$12,413,657,608.89	\$152,651,739.87	\$30,607.21

LOOKUPVALUE(Result_ColumnName, Search_ColumnName1, Search_Value1 [Search_ColumnName2] [Search_Value2]...)

The LOOKUPVALUE function returns the value in Result_Column for the row that meets all criteria specified in the search arguments. If no match satisfies the search values a BLANK is returned. If multiple rows match the search values, and all Result_Column values are identical, it will be returned, otherwise an error is returned.

This function is useful in a number of different scenarios. It is commonly used in conjunction with the parent-child functions to lookup the member name against a key. Refer to the example of the PATHITEM function in the Parent-Child section in this paper.

ALLSELECTED()

The ALLSELECTED function gets the context that represents all rows and columns in the query, while keeping explicit filters and contexts other than row and column filters. This function can be used to obtain visual totals in queries. It takes either no arguments, or either a table or column reference.

In the following example, the PivotTable presents Sales by StockTypeName on the rows, and includes the ability to filter the result by using the ClassName slicer. When all ClassName values are selected (i.e. there is no filter) this report works fine. The three measure expressions are:

- Sales =SUM(FactSales[SalesAmount])
- Sales_ALL =CALCULATE([Sales], ALL(FactSales))
- SalesRatio_ALL =[Sales]/[Sales_ALL]

	A	B	C	D	E	F
1			Row Labels	Sales	Sales_ALL	SalesRatio_ALL
2		ClassName	High	\$6,734,689,763.77	\$12,413,657,608.89	54.25 %
3		Deluxe	Low	\$2,579,528,111.57	\$12,413,657,608.89	20.78 %
4		Economy	Mid	\$3,099,439,733.55	\$12,413,657,608.89	24.97 %
5		Regular	Grand Total	\$12,413,657,608.89	\$12,413,657,608.89	100.00 %
6						
7						

When a ClassName value is selected, notice the Sales_ALL measure is evaluated using all rows in the FactSales table. The Grand Total values for Sales and Sales_ALL differ, with the latter calculation based on all FactSales rows. The ClassName filter does not apply to the Sales_ALL calculation.

	A	B	C	D	E	F
1			Row Labels	Sales	Sales_ALL	SalesRatio_ALL
2	ClassName		High	\$1,246,004,902.20	\$12,413,657,608.89	10.04 %
3	Deluxe		Low	\$656,379,560.65	\$12,413,657,608.89	5.29 %
4	Economy		Mid	\$1,201,237,128.45	\$12,413,657,608.89	9.68 %
5	Regular		Grand Total	\$3,103,621,591.29	\$12,413,657,608.89	25.00 %
6						
7						

Now two additional measures are added and the Sales_ALLSELECTED measure uses the ALLSELECTED function.

- Sales_ALLSELECTED =CALCULATE([Sales], ALLSELECTED())
- SalesRatio_ALLSELECTED =[Sales]/[Sales_ALLSELECTED]

	A	B	C	D	E	F	G	H
1			Row Labels	Sales	Sales_ALL	SalesRatio_ALL	Sales_ALLSELECTED	SalesRatio_ALLSELECTED
2	ClassName		High	\$1,246,004,902.20	\$12,413,657,608.89	10.04 %	\$3,103,621,591.29	40.15 %
3	Deluxe		Low	\$656,379,560.65	\$12,413,657,608.89	5.29 %	\$3,103,621,591.29	21.15 %
4	Economy		Mid	\$1,201,237,128.45	\$12,413,657,608.89	9.68 %	\$3,103,621,591.29	38.70 %
5	Regular		Grand Total	\$3,103,621,591.29	\$12,413,657,608.89	25.00 %	\$3,103,621,591.29	100.00 %
6								
7								

Notice the Sales_ALLSELECTED measure takes into consideration the ClassName filter, and that the Grand Total values for Sales and Sales_ALLSELECTED are the same. Also notice the SalesRatio_ALLSELECTED values sum to 100%. These additional calculations using the ALLSELECTED function produce the effect of visual totals.

When defining measures that should display the visual total (and not explicitly override a specific table or column) you should use the ALLSELECTED function rather than the ALL function. When using ALLSELECTED function, the function will always return the visual total regardless of what is on rows and column.

Time Intelligence Functions

One of the most common calculations performed in data analysis is to compare some number to a comparable number for a different time period. Calculations that make comparisons to last month or to the same period from a year ago are very important for any business intelligence tool. Toward that end, DAX includes 35 functions expressly for the purpose of working with time based data.

Concepts and Best Practices

Whenever you are defining measures and working with time periods, there are many opportunities to make bad assumptions about what the PivotTable in which the measure will be used might look like. To avoid a lot of those problems, DAX makes certain assumptions and we recommend certain best practices for anyone planning to use the time intelligence functions described in this section.

We recommend you create a table in your PowerPivot data that contains one row for every date that might exist in your data. Many people think of this as a date table or a time table or a time dimension. The notion is that this table has one row for each date, and there should be a many to one relationship from any date columns in the database to this date table. Building such a table in Excel is fairly trivial to do, and it might look like the table below, or like the DimDate table in Contoso.

	Date	Year	Month	Day	MonthFull	MonthShort	DayFull	DayShort
362	12/27/07	2007	12	27	December	Dec	Thursday	Thu
363	12/28/07	2007	12	28	December	Dec	Friday	Fri
364	12/29/07	2007	12	29	December	Dec	Saturday	Sat
365	12/30/07	2007	12	30	December	Dec	Sunday	Sun
366	12/31/07	2007	12	31	December	Dec	Monday	Mon
367	01/01/08	2008	1	1	January	Jan	Tuesday	Tue
368	01/02/08	2008	1	2	January	Jan	Wednesday	Wed
369	01/03/08	2008	1	3	January	Jan	Thursday	Thu
370	01/04/08	2008	1	4	January	Jan	Friday	Fri
371	01/05/08	2008	1	5	January	Jan	Saturday	Sat
372	01/06/08	2008	1	6	January	Jan	Sunday	Sun
373	01/07/08	2008	1	7	January	Jan	Monday	Mon
374	01/08/08	2008	1	8	January	Jan	Tuesday	Tue
375	01/09/08	2008	1	9	January	Jan	Wednesday	Wed

DAX uses this table to construct a set of dates for each calculation. The only column that matters is the date column itself. There are no requirements whatsoever about any of the other columns, and they aren't needed by DAX, although you may find them useful when you build PivotTables. You need to provide a reference to the Date column as an argument to every one of the 35 time intelligence functions in DAX.

The model developer can mark the Date table and define which Date type column in the table expresses the date. This allows the model tables to use a non-Date type key, which is common in data warehouse fact tables that use integer keys (typically the date in ISO format, e.g. 20110913). The model developer must ensure that the Date table contains one row per date and that there are no date gaps in the table data. Also, the model developer must ensure the table is marked as a date table.

This simple configuration will ensure that the Time Intelligence functions will function correctly, and also Excel will be aware that columns in the Data table can use time-related filters.

In DAX, we always calculate a set of dates as a table and then use that as if it were a SetFilter argument to the CALCULATE function. Consider that a user might have multi-selected some dates within a PivotTable, so that the context for a calculation might be any of the following:

- A single date
- A set of contiguous dates
- A set of non-contiguous dates
- Dates that happen to correspond to a calendar month or quarter or year (very common)

Then consider that we might need to shift those dates to find the following:

- The dates that make up the previous day, month, quarter, or year
- The same dates shifted to a previous month, quarter, or year
- The same dates shifted some interval of time (14 days, 30 days, etc.)
- Dates calculated by shifting an arbitrary interval forward or backward in time

In DAX, we accomplish all of these by working with sets of dates for everything. We don't try to know anything about month or quarter or year columns, but we do know the dates for any given month.

In the initial release of PowerPivot, we did not try to handle the many custom calendars we knew exist for financial analysis, but we do allow folks to build those custom calendars by authoring custom formulas to handle such things as 13 four week months in a year, or 4-4-5 quarters, etc.

The built-in functions handle calendar or fiscal years where fiscal year is defined as having a yearend date other than Dec 31. They also know about months, so when we shift the period April 1-30 back one month, we know we want March 1-31, and not merely March 1-30. This requires snapping to the end of a month in a variety of situations. In general DAX handles calendar quarters as 3 months, and years as 12 months, so all of the internal calculations are really based on days or months.

In the next section(s) of this paper, we look at the specific time intelligence functions in DAX and how they work.

Functions that Return a Single Date

The functions in this category return a single date. The result of these functions might then be used as arguments to other functions.

The first two functions simply return the first (or last) date in the Date_Column in the current context. This can be useful when you want to find the first (or last) date on which you sold each product, or the first (or last) date on which you had a transaction of a particular type. These functions take only one argument, the name of the date column in your date (or time) table.

- FIRSTDATE(Date_Column)
- LASTDATE(Date_Column)

The next two functions aren't strictly time intelligence functions, because they can be used for other purposes, but they will most often be used for time calculations. They are used to find the first (or last) date (or any other column value as well) where an expression has a non-blank value. This is most often used in situations like inventory, where you want to get the last inventory amount, and you don't happen to know when the last inventory was taken.

- FIRSTNONBLANK(Date_Column, Expression)
- LASTNONBLANK(Date_Column, Expression)

Here is an example from Contoso that shows how to calculate inventory using LASTNONBLANK. Note this is a fairly complex calculation, so we're going to take the time to describe it.

Let's look first at the FactInventory table in Contoso. This table contains a set of transactions where inventory was counted for various products in various stores. Each transaction identifies:

- a specific date
- a specific store
- a specific product
- an on-hand quantity

Note inventory quantities do not add. If we count 3 widgets one day, and 2 widgets a week later, the net result is we have 2 widgets (until another transaction occurs to change the quantity on hand.) At any moment in time, the on-hand-qty is the last count that was taken of that product in that store.

While inventories do not add within a store and within a product, they do add across stores and across products. If we have 3 apples and 2 oranges in one store, and 5 apples and 4 oranges in another, then we can say we have a total of 8 apples and 6 oranges, and we can even say we have a total of 14 products.

So how do we author this formula using DAX? (Warning – we are going to intentionally make a mistake in this process, and will point it out later, along with the necessary adjustment. Please bear with us.)

First, we need to know the quantity on hand for each product in each store. This is where LASTNONBLANK comes in. We also need to add these numbers up across a list of stores and a list of products, which are available to us as the DimStore and DimProduct tables. Consider this pair of formulas:

```
[BaseQty] = SUM(FactInventory[OnHandQuantity])
```

```
[QtyOnHand] = SUMX(VALUES(DimStore[StoreKey]),  
    SUMX(VALUES(DimProduct[ProductKey]),  
        CALCULATE([BaseQty],  
            LASTNONBLANK(DimDate[Datekey],[BaseQty]))))
```

These formulas say the following:

Define [BaseQty] as the sum of the OnHandQuantity amounts in FactInventory

Define [QtyOnHand] as a nested aggregation (nested sum) where

For each Store,

For each Product,

For the last date where there is a non-blank [BaseQty],

Calculate the [BaseQty],

Then add up those results across products and stores.

A PivotTable with this measure appears (at first glance) to give the right results. Let's look at this using just a couple of products in a couple of Canadian stores as examples. The last date is used for each product, and the quantities add up across multiple products and stores.

Contoso DAX Examples - Microsoft Excel

File Home Insert Page Layout Formulas Data Review View PowerPivot Options Design

A1 RegionCountryName

	A	B	C
1	RegionCountryName	Canada	
2			
3	Product Key	ProductName / Store Name / Date	QtyOnHand
4	7	Contoso 2G MP3 Player E200 Blue	27
5		Contoso Vancouver No.1 Store	11
6		5/5/2007	7
7		6/9/2007	7
8		6/16/2007	7
9		6/23/2007	7
10		10/18/2008	17
11		11/1/2008	11
12		11/8/2008	11
13		11/22/2008	11
14		Contoso Vancouver No.2 Store	16
15		6/30/2007	7
16		7/7/2007	5
17		7/14/2007	5
18		7/21/2007	5
19		11/3/2007	6
20		12/8/2007	5
21		12/15/2007	5
22		12/22/2007	5
23		10/4/2008	18
24		11/8/2008	15
25		11/15/2008	15
26		11/22/2008	15
27		1/10/2009	27
28		1/31/2009	16
29		2/14/2009	16
30		2/21/2009	16
31		2/28/2009	16
32	8	Contoso 4G MP3 Player E400 Silver	13
33	9	Contoso 4G MP3 Player E400 Black	15
34	Grand Total		55

PowerPivot Field List

Choose fields to add to report:

Search

- FactInventory
 - ☐ InventoryKey
 - ☐ DateKey
 - ☐ StoreKey
 - ☐ ProductKey
 - ☐ CurrencyKey
 - ☐ OnHandQuantity
 - ☐ OnOrderQuantity
 - ☐ SafetyStockQuantity
 - ☐ UnitCost
 - ☐ DaysInStock
 - ☐ MinDayInStock
 - ☐ MaxDayInStock
 - ☐ Aging
 - ☐ ETLLoadID
 - ☐ LoadDate
 - ☐ UpdateDate
 - ☐ InventoryQty

Slicers Vertical Slicers Horizontal

Report Filter Column Labels

RegionCountryName

Row Labels Values

ProductKey ProductName StoreName Datekey QtyOnHand

Ready Sales by Country Sales by Product 100%

So what's the problem? We haven't anticipated what might happen when someone slices this PivotTable using Time. For example, let's add Years to column labels and take another look at the results:

The screenshot shows a PivotTable in Excel with the following data:

Product Key	Product Name / Store Name / Date	2007	2008	2009
7	Contoso 2G MP3 Player E200 Blue	12	26	16
	Contoso Vancouver No.1 Store	7	11	
	5/5/2007	7		
	6/9/2007	7		
	6/16/2007	7		
	6/23/2007	7		
	10/18/2008		17	
	11/1/2008		11	
	11/8/2008		11	
	11/22/2008		11	
	Contoso Vancouver No.2 Store	5	15	16
	6/30/2007	7		
	7/7/2007	5		
	7/14/2007	5		
	7/21/2007	5		
	11/3/2007	6		
	12/8/2007	5		
	12/15/2007	5		
	12/22/2007	5		
	10/4/2008		18	
	11/8/2008		15	
	11/15/2008		15	
	11/22/2008		15	
	1/10/2009			27
	1/31/2009			16
	2/14/2009			16
	2/21/2009			16
	2/28/2009			16
8	Contoso 4G MP3 Player E400 Silver	8	21	13
9	Contoso 4G MP3 Player E400 Black	10	15	
Grand Total		30	62	29

The PowerPivot Field List on the right shows the following configuration:

- Report Filter:** RegionCountryName
- Column Labels:** CalendarYear
- Row Labels:** ProductKey, ProductName, StoreName, Datekey
- Values:** QtyOnHand

For Product 7 (Contoso 2G MP3 Player E200 Blue) the last inventory transaction occurs in 2008. When we are doing inventory calculations for 2009, since there are no inventory transactions, we report there is no inventory. But that's not true, because we still have the inventory left over from 2008.

This means we need to change our calculation, so that we change the context for time to include all the transactions from the beginning of time up until the current context. In order to do this, we need another time intelligence function we haven't gotten to yet. We'll come back to this example once we've described the function we'll need.

Six more functions that return a single date are the functions that return the first or last date of a month, quarter, or year within the current context of the calculation.

- STARTOFMONTH(Date_Column)
- STARTOFQUARTER(Date_Column)
- STARTOFYEAR(Date_Column [,YE_Date])
- ENDOFMONTH(Date_Column)
- ENDOFQUARTER(Date_Column)
- ENDOFYEAR(Date_Column [,YE_Date])

Functions that Return a Table of Dates

There are sixteen time intelligence functions that return a table of dates. Most often, these functions will be used as a SetFilter argument to CALCULATE. Just like all Time Intelligence functions in DAX, each function takes a column of dates as one of its arguments.

The first eight functions in this category are reasonably straightforward. Each of these functions starts with a date column in a current context. For example, if we are calculating a measure in a PivotTable, there might be a month or year on either the column labels or row labels. The net effect is the date column is filtered to include only the dates for the current context. Starting from that current context, these eight functions then calculate the previous (or next) day, month, quarter, or year and return those dates in the form of a single column table. Note the “previous” functions work backward from the first date in the current context, and the “next” functions move forward from the last date in the current context.

- PREVIOUSDAY(Date_Column)
- PREVIOUSMONTH(Date_Column)
- PREVIOUSQUARTER(Date_Column)
- PREVIOUSYEAR(Date_Column [,YE_Date])
- NEXTDAY(Date_Column)
- NEXTMONTH(Date_Column)
- NEXTQUARTER(Date_Column)
- NEXTYEAR(Date_Column [,YE_Date])

The next four functions in this category are similar, but instead of calculating a previous (or next) period, they calculate the set of dates in the period that is “month-to-date” (or quarter-to-date, or year-to-date, or in the same period of the previous year). These functions all perform their calculations using the last date in the current context. Note that SAMEPERIODLASTYEAR requires the current context contain a contiguous set of dates. If the current context is not a contiguous set of dates, then SAMEPERIODLASTYEAR will return an error.

- DATESMTD(Date_Column)
- DATESQTD(Date_Column)
- DATSYTD(Date_Column [,YE_Date])
- SAMEPERIODLASTYEAR(Date_Column)

The last four functions in this category are a bit more complex, and also a bit more powerful. These functions are used to shift from the set of dates that are in the current context to a new set of dates.

- DATEADD(Date_Column, Number_of_Intervals, Interval)

- DATESBETWEEN(Date_Column, Start_Date, End_Date)
- DATESINPERIOD(Date_Column, Start_Date, Number_of_Intervals, Interval)
- PARALLELPERIOD(Date_Column, Number_of_Intervals, Interval)

DATESBETWEEN calculates the set of dates between the specified start date and end date. The remaining three functions shift some number of time intervals from the current context. The interval can be day, month, quarter or year. These functions make it very easy to shift the time interval for a calculation by any of the following:

- Go back two years
- Go back one month
- Go forward three quarters
- Go back 14 days
- Go forward 28 days

In each case, we only need to specify which interval, and how many of those intervals we want to shift. A positive interval will move forward in time, while a negative interval will move back in time. The interval itself is specified by a keyword of DAY, MONTH, QUARTER, or YEAR. Note these keywords are not strings, so they should not be in quotation marks. It's also useful to note that auto-complete doesn't yet help to fill in these keywords, so you simply have to type them in.

Let's look at some examples of how these functions might be used with the Contoso data set:

Year Over Year Growth

The problem we want to solve is to calculate year-over-year growth for Store Sales. Although we've had formulas for StoreSales earlier in this paper (based on a [Sales] measure) let's assume that formula is no longer handy and we want to start over with the Contoso database. A formula that might have seemed complicated before is now pretty straightforward:

```
[StoreSales] = CALCULATE(SUM(FactSales[SalesAmount]), DimChannel[ChannelName] = "Store")
```

In other words, Store Sales is defined as the sum of the SalesAmount column in the FactSales table, with the context modified to include only sales where the Channel Name in the DimChannel table is "Store".

Now we want to calculate StoreSales for the previous year.

```
[StoreSalesPrevYr] = CALCULATE([StoreSales], DATEADD(DimDate[DateKey], -1, YEAR))
```

Since we're calculating a single measure in a modified context, this can be written using a syntax shortcut.

```
[StoreSalesPrevYr] = [StoreSales](DATEADD(DimDate[DateKey], -1, YEAR))
```

Now we can calculate Year over Year Growth by simply subtracting last year's sales from this year's sales and showing that difference as a percentage of last year's sales.

```
[YOYGrowth] = ([StoreSales] - [StoreSalesPrevYr]) / [StoreSalesPrevYr]
```

Finally we need to wrap this in an IF statement, so we don't get division by zero in the first year.

[YOYGrowth] = IF([StoreSalesPrevYr], ([StoreSales] - [StoreSalesPrevYr]) / [StoreSalesPrevYr], BLANK())

Place [StoreSales] and [YOYGrowth] in a PivotTable with countries on rows, and years on columns:

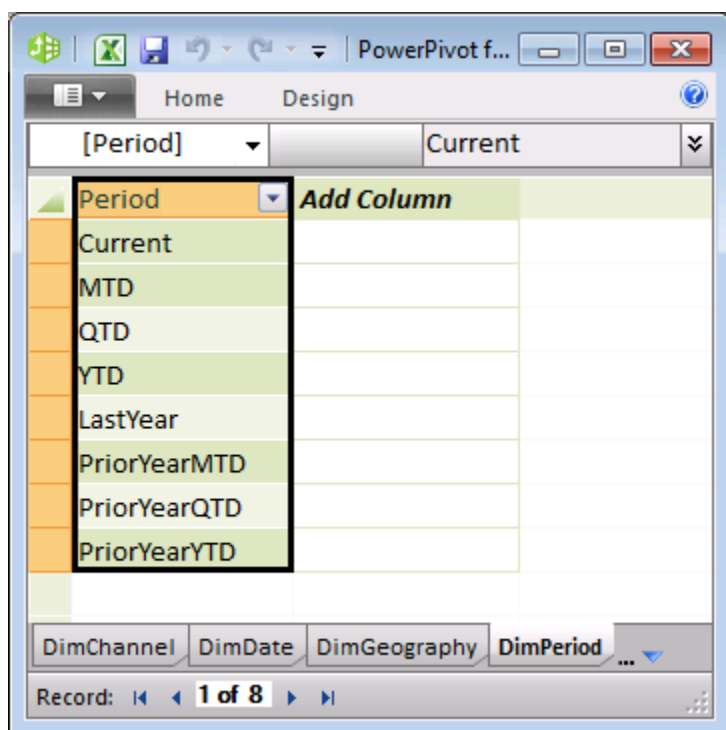
	Column Labels					
	2007		2008		2009	
Row Labels	StoreSales	YOYGrowth	StoreSales	YOYGrowth	StoreSales	YOYGrowth
Armenia	\$10,697,642.89		\$12,802,000.68	19.7%	\$13,768,349.90	7.5%
Australia	\$31,698,751.16		\$39,734,710.75	25.4%	\$41,128,813.37	3.5%
Bhutan	\$11,011,949.91		\$12,899,605.66	17.1%	\$19,962,257.73	54.8%
Canada	\$108,076,080.27		\$79,570,342.19	-26.4%	\$63,002,636.49	-20.8%
China	\$51,524,018.51		\$71,923,190.97	39.6%	\$86,167,763.67	19.8%
Denmark	\$10,711,449.83		\$5,693,881.55	-46.8%	\$5,130,625.49	-9.9%
France	\$64,294,935.44		\$38,294,242.87	-40.4%	\$33,245,400.64	-13.2%
Germany	\$98,916,125.19		\$58,967,169.35	-40.4%	\$53,134,680.49	-9.9%
Greece	\$10,662,152.52		\$5,744,113.96	-46.1%	\$5,049,809.47	-12.1%
Holland	\$10,700,622.10		\$5,916,178.50	-44.7%	\$4,781,201.86	-19.2%
India	\$31,834,758.18		\$39,621,943.29	24.5%	\$40,462,377.31	2.1%
Iran	\$21,314,190.11		\$26,804,288.38	25.8%	\$27,098,480.82	1.1%
Ireland	\$10,570,530.05		\$5,916,233.23	-44.0%	\$4,962,315.77	-16.1%
Italy	\$34,719,790.81		\$23,229,095.53	-33.1%	\$20,541,708.18	-11.6%
Japan	\$56,801,779.57		\$82,804,472.58	45.8%	\$96,818,245.96	16.9%
Kyrgyzstan	\$10,913,951.79		\$12,880,870.84	18.0%	\$13,804,578.11	7.2%
Malta	\$10,572,691.21		\$5,781,747.66	-45.3%	\$4,921,696.16	-14.9%
Pakistan	\$11,992,037.52		\$26,044,576.10	117.2%	\$26,279,344.58	0.9%
Poland	\$10,733,699.68		\$6,023,341.08	-43.9%	\$4,693,341.12	-22.1%
Portugal	\$10,629,143.02		\$5,911,127.59	-44.4%	\$4,910,978.97	-16.9%
Romania	\$10,900,048.39		\$5,827,158.89	-46.5%	\$4,950,364.81	-15.0%
Russia	\$42,189,724.14		\$31,289,351.16	-25.8%	\$27,870,158.03	-10.9%
Singapore	\$10,262,476.73		\$12,832,977.51	25.0%	\$13,167,637.07	2.6%
Slovenia	\$10,552,250.44		\$5,702,422.82	-46.0%	\$4,862,182.32	-14.7%
South Korea	\$11,322,555.08		\$16,008,692.46	41.4%	\$25,809,600.08	61.2%
Spain	\$10,775,770.69		\$5,713,579.13	-47.0%	\$4,896,696.91	-14.3%
Sweden	\$10,511,963.13		\$5,688,176.23	-45.9%	\$4,817,243.12	-15.3%
Switzerland	\$10,515,133.82		\$5,496,527.34	-47.7%	\$4,785,769.58	-12.9%
Syria	\$14,010,740.05		\$26,405,490.76	88.5%	\$26,299,843.29	-0.4%
Taiwan	\$10,582,040.28		\$13,364,330.92	26.3%	\$13,663,685.18	2.2%
Thailand	\$11,138,010.00		\$15,767,761.44	41.6%	\$25,970,356.17	64.7%
Turkmenistan	\$20,877,985.35		\$25,693,638.02	23.1%	\$27,240,861.29	6.0%
United Kingdom	\$151,571,280.23		\$89,253,182.30	-41.1%	\$73,230,693.92	-18.0%
United States	\$1,839,594,612.14		\$1,402,853,883.84	-23.7%	\$1,103,782,301.49	-21.3%
Grand Total	\$2,783,180,890.20		\$2,228,460,305.60	-19.9%	\$1,931,211,999.35	-13.3%

Calculating Many Time Periods Within a Single Measure Formula

Consider the following eight calculations. Each of these formulas calculates a sales amount (in the Contoso database) for a distinct time period.

Time Period	Formula
Current Period	=Sales
MTD	=Sales(DATESMTD(DimDate[Datekey]))
QTD	=Sales(DATESQTD(DimDate[Datekey]))
YTD	=Sales(DATESYTD(DimDate[Datekey]))
Current Period Last Year	=Sales(DATEADD(DimDate[Datekey],-1,YEAR))
PriorYearMTD	=Sales(DATEADD(DATESMTD(DimDate[Datekey]),-1,YEAR))
PriorYearQTD	=Sales(DATEADD(DATESQTD(DimDate[Datekey]),-1,YEAR))
PriorYearYTD	=Sales(DATEADD(DATESYTD(DimDate[Datekey]),-1,YEAR))

Let's create a new table named DimPeriod in PowerPivot that has a single column named Period, by pasting in a set of text strings like this from Excel:



Now let's build a single measure formula that does the following:

1. Use an IF function, to see if the DimPeriod[Period] column has been placed into the Filter Context by placing this column onto the PivotTable's column labels, or row labels, or perhaps on a slicer. The best way is to count the number of values that are applicable from this column. If this isn't in the filter context, all 8 values will be applicable. But if this is on columns or rows, then only one of the values will be applicable for each cell in the values area (except for the Total row/column).

2. If only one of these values is applicable, then use a SWITCH statement nested in an IF statement to see which one it is, and apply the appropriate formula.
3. If this column is not on rows or columns (or selected to a single value in a slicer) then assume the user wants Current sales figure instead of one of the more elaborate calculations.

Such a formula would look like this:

```
=IF(HASONEVALUE(DimPeriod[Period]),  
    SWITCH(VALUES(DimPeriod[Period]),  
        "Current", [Sales],  
        "MTD", [Sales](DATESMTD(DimDate[Datekey])),  
        "QTD", [Sales](DATESQTD(DimDate[Datekey])),  
        "YTD", [Sales](DATESYTD(DimDate[Datekey])),  
        "LastYear", [Sales](DATEADD(DimDate[Datekey],-1,YEAR)),  
        "PriorYearMTD", [Sales](DATEADD(DATESMTD(DimDate[Datekey]),-1,YEAR)),  
        "PriorYearQTD", [Sales](DATEADD(DATESQTD(DimDate[Datekey]),-1,YEAR)),  
        "PriorYearYTD", [Sales](DATEADD(DATESYTD(DimDate[Datekey]),-1,YEAR)),  
        BLANK()))
```

Now if we build a PivotTable with DimGeography[RegionCountryName] on row labels, DimPeriod[Period] on column labels, a single date from DimDate[FullDateLabel] in the Report Filter, and my new formula as the only measure, we will get a PivotTable that looks like this:

Contoso Sample DAX Formulas - Microsoft Excel

PivotTable Tools: Options, Design

File Home Insert Page Layout Formulas Data Review View PowerPivot

B1 2008-11-04

FullDateLabel	2008-11-04								
PeriodSales	Column Labels								
Row Labels	Current	MTD	QTD	YTD	LastYear	PriorYearMTD	PriorYearQTD	PriorYearYTD	
Armenia	\$ 53,497.42	\$ 191,578.86	\$ 1,099,703.76	\$ 10,573,876.86	\$ 19,165.30	\$ 177,866.99	\$ 1,086,386.59	\$ 8,515,932.02	
Australia	\$ 106,031.77	\$ 520,730.02	\$ 3,599,380.22	\$ 32,380,669.99	\$ 67,067.44	\$ 381,595.22	\$ 3,062,176.57	\$ 25,428,532.05	
Bhutan	\$ 40,873.43	\$ 199,269.64	\$ 1,255,113.64	\$ 10,615,904.29	\$ 29,604.01	\$ 161,807.34	\$ 1,051,978.09	\$ 8,816,133.57	
Canada	\$ 316,637.57	\$ 978,578.23	\$ 7,411,803.93	\$ 64,924,810.91	\$ 326,814.41	\$ 1,592,639.82	\$ 13,117,433.12	\$ 88,737,924.00	
China	\$ 1,852,200.40	\$ 7,384,601.08	\$ 54,143,496.92	\$ 470,675,431.89	\$ 1,570,606.35	\$ 6,358,381.08	\$ 45,460,612.05	\$ 370,551,111.02	
Denmark	\$ 12,468.73	\$ 53,223.63	\$ 522,998.30	\$ 4,932,953.18	\$ 24,890.82	\$ 143,172.62	\$ 1,006,029.81	\$ 9,202,455.69	
France	\$ 568,303.96	\$ 2,105,588.04	\$ 18,089,705.87	\$ 189,577,757.71	\$ 596,111.07	\$ 2,546,849.81	\$ 20,969,888.63	\$ 214,538,710.62	
Germany	\$ 895,327.02	\$ 3,612,463.27	\$ 28,157,408.97	\$ 292,349,124.98	\$ 940,203.95	\$ 3,545,995.12	\$ 29,755,662.33	\$ 304,361,113.77	
Greece	\$ 7,780.05	\$ 66,916.63	\$ 476,927.62	\$ 4,977,762.03	\$ 24,761.09	\$ 105,478.00	\$ 958,498.05	\$ 9,118,514.16	
India	\$ 146,564.51	\$ 715,337.29	\$ 3,901,481.19	\$ 32,442,927.34	\$ 69,272.74	\$ 358,661.67	\$ 3,015,426.22	\$ 25,305,128.16	
Iran	\$ 99,183.72	\$ 341,172.05	\$ 2,537,238.55	\$ 22,034,210.52	\$ 79,490.79	\$ 314,454.47	\$ 2,217,963.02	\$ 16,835,672.09	
Ireland	\$ 12,506.43	\$ 51,615.30	\$ 454,870.50	\$ 5,112,067.20	\$ 32,900.12	\$ 121,275.90	\$ 864,113.35	\$ 9,001,740.91	
Italy	\$ 42,053.08	\$ 210,739.40	\$ 1,901,601.41	\$ 20,243,442.19	\$ 132,541.56	\$ 472,312.52	\$ 3,565,127.34	\$ 28,556,745.51	
Japan	\$ 198,429.73	\$ 988,073.32	\$ 8,477,059.82	\$ 66,053,978.75	\$ 287,436.89	\$ 958,141.97	\$ 6,409,316.72	\$ 43,760,610.48	
Kyrgyzstan	\$ 66,765.88	\$ 178,790.54	\$ 1,119,671.44	\$ 10,478,278.49	\$ 22,909.06	\$ 135,002.53	\$ 1,121,717.43	\$ 8,842,890.25	
Malta	\$ 4,629.29	\$ 74,922.72	\$ 457,996.84	\$ 4,932,064.47	\$ 24,402.42	\$ 85,404.37	\$ 832,052.61	\$ 8,997,363.85	
Pakistan	\$ 89,397.86	\$ 326,509.47	\$ 2,331,272.17	\$ 21,174,561.17	\$ 124,593.47	\$ 256,309.48	\$ 1,165,001.58	\$ 8,985,349.93	
Poland	\$ 4,585.73	\$ 52,711.28	\$ 484,525.67	\$ 5,263,947.17	\$ 21,977.97	\$ 94,920.43	\$ 857,675.11	\$ 9,260,790.88	
Portugal	\$ 8,867.00	\$ 45,434.28	\$ 447,525.07	\$ 5,026,198.31	\$ 31,197.09	\$ 116,211.31	\$ 865,943.77	\$ 9,083,677.34	
Romania	\$ 12,428.98	\$ 39,587.89	\$ 499,427.21	\$ 5,101,389.11	\$ 19,214.67	\$ 109,623.71	\$ 907,530.86	\$ 9,314,105.45	
Russia	\$ 87,822.64	\$ 316,123.60	\$ 2,685,764.27	\$ 26,303,454.92	\$ 133,148.08	\$ 484,013.06	\$ 3,587,749.35	\$ 35,542,427.16	
Singapore	\$ 40,322.79	\$ 153,497.42	\$ 1,280,590.72	\$ 10,514,459.29	\$ 17,181.49	\$ 142,172.49	\$ 873,797.94	\$ 8,088,998.43	
Slovenia	\$ 14,397.92	\$ 58,888.39	\$ 507,037.76	\$ 4,934,985.68	\$ 23,955.59	\$ 73,068.50	\$ 743,555.94	\$ 8,977,944.06	
South Korea	\$ 92,524.64	\$ 309,753.40	\$ 1,919,097.00	\$ 11,187,231.44	\$ 54,672.82	\$ 134,113.07	\$ 1,111,224.97	\$ 9,077,284.77	
Spain	\$ 13,754.17	\$ 47,370.69	\$ 528,007.37	\$ 4,968,537.43	\$ 27,078.60	\$ 134,180.07	\$ 890,486.24	\$ 9,260,718.19	
Sweden	\$ 6,756.33	\$ 34,156.19	\$ 486,691.15	\$ 4,925,025.46	\$ 33,651.54	\$ 85,008.48	\$ 894,653.62	\$ 9,066,176.49	
Switzerland	\$ 18,151.26	\$ 45,186.75	\$ 508,401.51	\$ 4,789,504.07	\$ 28,789.53	\$ 87,957.62	\$ 897,995.86	\$ 9,018,584.07	
Syria	\$ 73,148.10	\$ 365,446.15	\$ 2,586,373.35	\$ 21,547,830.87	\$ 136,207.60	\$ 347,805.48	\$ 2,075,293.88	\$ 9,969,292.28	
Taiwan	\$ 20,154.74	\$ 169,150.21	\$ 1,101,863.91	\$ 10,848,676.26	\$ 26,918.22	\$ 139,294.07	\$ 1,048,450.52	\$ 8,532,719.98	
Thailand	\$ 32,260.55	\$ 178,407.71	\$ 1,227,962.91	\$ 10,873,764.90	\$ 26,648.87	\$ 149,130.50	\$ 1,175,006.90	\$ 9,095,576.86	
the Netherlands	\$ 11,311.25	\$ 62,641.82	\$ 470,692.53	\$ 5,041,981.59	\$ 30,913.97	\$ 124,677.60	\$ 1,039,466.11	\$ 9,174,420.97	
Turkmenistan	\$ 62,445.85	\$ 356,872.43	\$ 2,572,119.23	\$ 20,943,498.41	\$ 75,495.21	\$ 284,115.68	\$ 2,075,169.63	\$ 16,424,535.84	
United Kingdom	\$ 278,967.96	\$ 950,174.58	\$ 7,444,000.24	\$ 77,063,395.81	\$ 467,785.54	\$ 1,624,756.49	\$ 13,221,126.58	\$ 129,312,192.62	
United States	\$ 7,748,194.23	\$ 30,313,156.29	\$ 217,067,225.39	\$ 1,888,068,614.10	\$ 9,786,995.77	\$ 40,065,133.71	\$ 317,200,261.24	\$ 2,255,486,360.92	
Grand Total	\$ 13,038,744.96	\$ 51,498,668.60	\$ 377,755,036.45	\$ 3,380,882,316.81	\$ 15,314,604.06	\$ 61,911,531.18	\$ 485,124,772.02	\$ 3,744,241,734.39	

Ready SalesByCountry SalesByPeriod Inventory

Finishing Up Our Inventory Scenario

We now have the functions we need to finish up the Inventory problem we were looking at.

The formula we had before was:

```
[QtyOnHand] = SUMX(VALUES(DimStore[StoreKey]),
    SUMX(VALUES(DimProduct[ProductKey]),
        CALCULATE([BaseQty],
            LASTNONBLANK(DimDate[Datekey],[BaseQty]))))
```

And the problem was LASTNONBLANK was getting the last non-blank date from within the current context, when it should have been considering a larger time period. For example, when we are calculating inventory for 2009, we need to consider the last time inventory was taken from the beginning of time up through 2009. When in the context of the year 2009, the formula above only considers inventory transactions within 2009, and doesn't consider transactions that may have occurred prior to 2009.

We can fix the formula by using DATESBETWEEN whose function signature looks like this:

DATESBETWEEN(Date_Column, Start_Date, End_Date)

We'll use a blank start date (to signify that we want to start at the absolute beginning), and we'll use the last date from the current context as our end date. This means that we want this expression:

DATESBETWEEN(DimDate[DateKey], BLANK(), LASTDATE(DimDate[DateKey]))

And we'll use this expression in place of the date column in LASTNONBLANK, since we want LASTNONBLANK to consider the expanded range of dates:

```
[QtyOnHand] = SUMX(VALUES(DimStore[StoreKey]),  
    SUMX(VALUES(DimProduct[ProductKey]),  
        CALCULATE([BaseQty],  
            LASTNONBLANK(  
                DATESBETWEEN(DimDate[DateKey], BLANK(), LASTDATE(DimDate[DateKey])),  
                [BaseQty])))
```

Note that we could have used our syntax shortcut for CALCULATE(<measure>) here as well:

```
[QtyOnHand] = SUMX(VALUES(DimStore[StoreKey]),  
    SUMX(VALUES(DimProduct[ProductKey]),  
        [BaseQty](LASTNONBLANK(  
            DATESBETWEEN(DimDate[DateKey], BLANK(), LASTDATE(DimDate[DateKey])),  
            [BaseQty])))
```

The good news is that our inventory now looks correct across dates, and items last counted in 2008 appear in the inventory for 2009 as well as for 2010 and 2011, where we know we have no inventory transactions.

Whitepaper Inventory Example - Microsoft Excel						
PivotTable Tools						
Options Design						
F14 11						
A	B	C	D	E	F	G
1	RegionCountryName	Canada				
2						
3	QtyOnHand					
4	Product Key	Product Name / Store Name / Date	2007	2008	2009	2010
5	7	Contoso 2G MP3 Player E200 Blue	68	111	112	112
6		Contoso Calgary Store	5	5	5	5
7		Contoso Montreal No.1 Store	7	18	18	18
8		Contoso Montreal No.2 Store	7	7	7	7
9		Contoso Ottawa No.1 Store	8	8	8	8
10		Contoso Ottawa No.2 Store	6	13	13	13
11		Contoso Toronto No.1 Store	5	5	5	5
12		Contoso Toronto No.2 Store	5	18	18	18
13		Contoso Toronto No.3 Store	8	6	6	6
14		Contoso Vancouver No.1 Store	7	11	11	11
15		Contoso Vancouver No.2 Store	5	15	16	16
16		Contoso Westminster Store	5	5	5	5
17	8	Contoso 4G MP3 Player E400 Silver	55	135	214	214
18		Contoso Calgary Store	8	8	8	8
19		Contoso Montreal No.1 Store	6	21	40	40
20		Contoso Montreal No.2 Store		15	15	15
21		Contoso Ottawa No.1 Store		15	15	15
22		Contoso Ottawa No.2 Store	8	14	14	14
23		Contoso Toronto No.1 Store	7	18	39	39
24		Contoso Toronto No.2 Store	8	13	13	13
25		Contoso Toronto No.3 Store	5	5	5	5
26		Contoso Vancouver No.1 Store	8	21	13	13
27		Contoso Westminster Store	5	5	52	52
28	9	Contoso 4G MP3 Player E400 Black	75	146	193	193
29		Contoso Calgary Store	8	24	19	19
30		Contoso Montreal No.1 Store	7	21	44	44
31		Contoso Montreal No.2 Store	6	15	15	15
32		Contoso Ottawa No.1 Store	8	8	15	15
33		Contoso Ottawa No.2 Store	6	6	6	6
34		Contoso Toronto No.1 Store	12	21	15	15
35		Contoso Toronto No.2 Store	8	8	8	8
36		Contoso Toronto No.3 Store		18	46	46
37		Contoso Vancouver No.2 Store	10	15	15	15
38		Contoso Westminster Store	10	10	10	10
39	Grand Total		198	392	519	519
40						

Functions that Evaluate Expressions Over a Time Period

There is one more set of Time Intelligence functions in DAX – these are the functions that evaluate an expression over a specified time period. These functions are all provided as a convenience. You can accomplish the same thing using CALCULATE and other Time Intelligence functions. For example,

```
=TOTALMTD(Expression, Date_Column [, SetFilter])
```

This expression is precisely the same as this combination of functions:

```
=CALCULATE(Expression, DATESMTD (Date_Column)[, SetFilter])
```

But it will be simpler and easier for users to use these DAX functions when they are a good fit for the problem that needs to be solved. The DAX functions that do this are:

- TOTALMTD(Expression, Date_Column [, SetFilter])
- TOTALQTD(Expression, Date_Column [, SetFilter])
- TOTALYTD(Expression, Date_Column [, SetFilter] [,YE_Date])

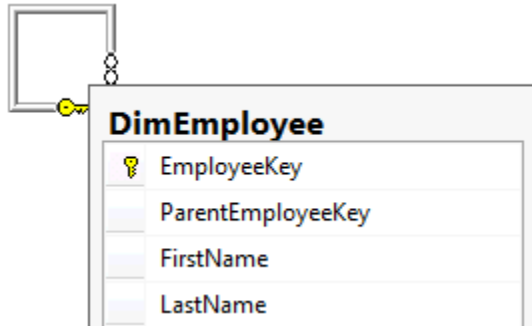
For the functions that calculate opening and closing balances, there are certain concepts that are useful. First, as you might think obvious, the opening balance for any period is the same as the closing balance for the previous period. The closing balance includes all data through the end of the period, while the opening balance does not include any data from within the current period.

These functions always return the value of an expression evaluated for a specific point in time. The point in time we care about is always the last possible date value in a calendar period. The opening balance is based on the last date of the previous period, while the closing balance is based on the last date in the current period. The current period is always determined by the last date in the current date context.

- OPENINGBALANCEMONTH(Expression, Date_Column [,SetFilter])
- OPENINGBALANCEQUARTER(Expression, Date_Column [,SetFilter])
- OPENINGBALANCEYEAR(Expression, Date_Column [,SetFilter] [,YE_Date])
- CLOSINGBALANCEMONTH(Expression, Date_Column [,SetFilter])
- CLOSINGBALANCEQUARTER(Expression, Date_Column [,SetFilter])
- CLOSINGBALANCEYEAR(Expression, Date_Column [,SetFilter] [,YE_Date])

Parent-Child Functions

DAX includes a set of functions specifically to work with data stored in a table with a self-referencing relationship. Tables with this type of relationship are designed to store relationships between the records in the same table. The examples used to illustrate these functions in this paper are based on the DimEmployee table which includes a primary key column named EmployeeKey, and the ParentEmployeeKey column which is a foreign key constraint referencing the primary key.



Note tabular models cannot define self-referencing relationships; however, by using DAX the implied relationships can be materialized and used in calculations. Oftentimes this type of relationship is used to create a hierarchy referred to as a parent-child hierarchy, and common scenarios that can benefit from this type of hierarchy include general ledger and organizational structures.

A useful first step when working with the parent-child functions is to create a calculated column to store the full path between the current record and its ancestors. Typically this column will be hidden from end users. The PATH function is used to produce a pipe (|) delimited string with the identifiers of all the parents of the current identifier, starting with the oldest and continuing until current. The function requires the input of the table ID and the parent ID. Using the DimEmployee table example, the Path column based on the expression `PATH([EmployeeKey], [ParentEmployeeKey])` will produce the following result.

EmployeeKey	ParentEmployeeKey	EmployeeName	Path
1	18	Kim Abercrombie	18 1
2	18	Sagiv Hadaya	18 2
3	18	Luka Abrus	18 3
4	18	Kirk Nason	18 4
5	18	Humberto Acevedo	18 5
6	1	Yoichiro Okada	18 1 6
7	2	Pilar Ackerman	18 2 7
8	3	Aaron Painter	18 3 8
9	4	Terry Adams	18 4 9
10	5	David Probst	18 5 10
11	1	Manoj Agarwal	18 1 11
12	2	Michael Raheem	18 2 12
13	3	David Ahs	18 3 13
14	8	Miguel Saenz	18 3 8 14
15	9	Kim Akers	18 4 9 15
16	1	Kate Taneyhill	18 1 16
17	2	David Alexander	18 2 17
18		Pieter Uittenbogaard	18
19	4	Michelle Alexander	18 4 19

Notice the row for EmployeeKey 18 (Pieter Uittenbogaard). A blank ParentEmployeeKey value means this employee is the root of the hierarchy – in fact he is the Group Manager (there is no higher position). Next, notice EmployeeKey 1 (Kim Abercrombie) who has a ParentEmployeeKey value of 18. She reports to Pieter Uittenbogaard, and is a Region Manager. Next, notice the row for EmployeeKey 6 (Yoichiro Okada). He reports to Kim Abercrombie and is a State Manager. By now it should be clear what the path represents the chain of relationships for each row in the table.

The PATHITEM function is used to extract an item from the path at a particular position. This is useful for naturalizing the parent-child related data into fixed columns, and is necessary if you want to construct a hierarchy to simulate a parent-child relationship in the data model. The PATHITEM function requires a path produced by the PATH function, and the position to return. Positions are counted from left to right.

In the above example the PATHITEM([Path], 1, 1) will return the value 18 for each row. The second argument, in this case, 1, specifies the type INT. The column Region Manager based on the expression PATHITEM([Path], 2, 1) will produce the following result.

EmployeeName ▾	Path ▾	State Manager ▾
Kim Abercrombie	18 1	1
Sagiv Hadaya	18 2	2
Luka Abrus	18 3	3
Kirk Nason	18 4	4
Humberto Acevedo	18 5	5
Yoichiro Okada	18 1 6	1
Pilar Ackerman	18 2 7	2
Aaron Painter	18 3 8	3
Terry Adams	18 4 9	4
David Probst	18 5 10	5
Manoj Agarwal	18 1 11	1
Michael Raheem	18 2 12	2
David Ahs	18 3 13	3
Miguel Saenz	18 3 8 14	3
Kim Akers	18 4 9 15	4
Kate Taneyhill	18 1 16	1
David Alexander	18 2 17	2
Pieter Uittenbogaard	18	
Michelle Alexander	18 4 19	4

Notice the State Manager column returns the second item from the Path column. Notice in the row for Pieter Uittenbogaard the request for an item that does not exist in the path will return BLANK.

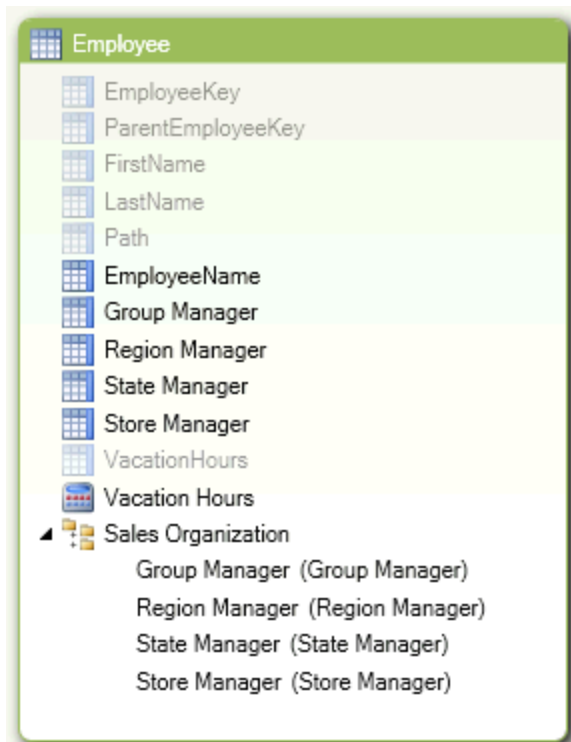
When the item is a key, it is useful to perform a lookup to retrieve the employee name. The DAX expression used in the Region Manager column is updated to LOOKUPVALUE([EmployeeName], [EmployeeKey], PATHITEM([Path], 2, 1)), and it produces the following result.

EmployeeName	Path	Region Manager
Kim Abercrombie	18 1	Kim Abercrombie
Sagiv Hadaya	18 2	Sagiv Hadaya
Luka Abrus	18 3	Luka Abrus
Kirk Nason	18 4	Kirk Nason
Humberto Acevedo	18 5	Humberto Acevedo
Yoichiro Okada	18 1 6	Kim Abercrombie
Pilar Ackerman	18 2 7	Sagiv Hadaya
Aaron Painter	18 3 8	Luka Abrus
Terry Adams	18 4 9	Kirk Nason
David Probst	18 5 10	Humberto Acevedo
Manoj Agarwal	18 1 11	Kim Abercrombie
Michael Raheem	18 2 12	Sagiv Hadaya
David Ahs	18 3 13	Luka Abrus
Miguel Saenz	18 3 8 14	Luka Abrus
Kim Akers	18 4 9 15	Kirk Nason
Kate Taneyhill	18 1 16	Kim Abercrombie
David Alexander	18 2 17	Sagiv Hadaya
Pieter Uittenbogaard	18	
Michelle Alexander	18 4 19	Kirk Nason

This calculation can be adapted to produce the Group Manager (position 1), State Manager (position 3) and Store Manager (position 4).

EmployeeName	Path	Group Manager	Region Manager	State Manager	Store Manager
Kim Abercrombie	18 1	Pieter Uittenbogaard	Kim Abercrombie		
Sagiv Hadaya	18 2	Pieter Uittenbogaard	Sagiv Hadaya		
Luka Abrus	18 3	Pieter Uittenbogaard	Luka Abrus		
Kirk Nason	18 4	Pieter Uittenbogaard	Kirk Nason		
Humberto Acevedo	18 5	Pieter Uittenbogaard	Humberto Acevedo		
Yoichiro Okada	18 1 6	Pieter Uittenbogaard	Kim Abercrombie	Yoichiro Okada	
Pilar Ackerman	18 2 7	Pieter Uittenbogaard	Sagiv Hadaya	Pilar Ackerman	
Aaron Painter	18 3 8	Pieter Uittenbogaard	Luka Abrus	Aaron Painter	
Terry Adams	18 4 9	Pieter Uittenbogaard	Kirk Nason	Terry Adams	
David Probst	18 5 10	Pieter Uittenbogaard	Humberto Acevedo	David Probst	
Manoj Agarwal	18 1 11	Pieter Uittenbogaard	Kim Abercrombie	Manoj Agarwal	
Michael Raheem	18 2 12	Pieter Uittenbogaard	Sagiv Hadaya	Michael Raheem	
David Ahs	18 3 13	Pieter Uittenbogaard	Luka Abrus	David Ahs	
Miguel Saenz	18 3 8 14	Pieter Uittenbogaard	Luka Abrus	Aaron Painter	Miguel Saenz
Kim Akers	18 4 9 15	Pieter Uittenbogaard	Kirk Nason	Terry Adams	Kim Akers
Kate Taneyhill	18 1 16	Pieter Uittenbogaard	Kim Abercrombie	Kate Taneyhill	
David Alexander	18 2 17	Pieter Uittenbogaard	Sagiv Hadaya	David Alexander	
Pieter Uittenbogaard	18	Pieter Uittenbogaard			
Michelle Alexander	18 4 19	Pieter Uittenbogaard	Kirk Nason	Michelle Alexander	

These columns can then be used to create a hierarchy named Sales Organization in this example. Notice the key columns and the Path column are hidden. These do not need to be surfaced to the model consumers.



The following PivotTable report uses the Sales Organization to report Vacation Hours.

	A	B
1	Row Labels	Vacation Hours
2	Pieter Uittenbogaard	14,041
3	+	80
4	+ Humberto Acevedo	2,569
5	- Kim Abercrombie	3,218
6	+	40
7	+ David Yalovsky	40
8	+ Diogo Andrade	40
9	+ Kate Taneyhill	40
10	+ Manoj Agarwal	40
11	+ Michael Allen	40
12	+ Yoichiro Okada	2,978
13	+ Kirk Nason	4,611
14	+ Luka Abrus	1,874
15	+ Sagiv Hadaya	1,689
16	Grand Total	14,041

The Sales Organization hierarchy reveals blank members that represent the values for parent member. For example, the 80 Vacation Hours in cell B3 are Pieter Uittenbogaard's. To provide meaningful labels to the blank values, the DAX expression can be adapted to test for a blank value returned from the ITEMPATH function, and where it is blank, the value can be expressed as the last item in the path. The PATHITEMREVERSE is helpful here. This function returns the item at the specified position from a path, and positions are counted backwards from right to left. The request for the item at position 1 will provide the lowest (leaf) item.

The Region Manager column is updated to:

```
IF(ISBLANK(PATHITEM([Path], 2, 1)), "(" & LOOKUPVALUE([EmployeeName], [EmployeeKey],  
PATHITEMREVERSE([Path], 1, 1)) & " data)", LOOKUPVALUE([EmployeeName], [EmployeeKey],  
PATHITEM([Path], 2, 1)))
```

The State Manager and Store Manager columns are similarly updated, and it produces the following result.

EmployeeName	Path	Group Manager	Region Manager	State Manager	Store Manager
Kim Abercrombie	18 1	Pieter Uittenbogaard	Kim Abercrombie	(Kim Abercrombie data)	(Kim Abercrombie data)
Sagiv Hadaya	18 2	Pieter Uittenbogaard	Sagiv Hadaya	(Sagiv Hadaya data)	(Sagiv Hadaya data)
Luka Abrus	18 3	Pieter Uittenbogaard	Luka Abrus	(Luka Abrus data)	(Luka Abrus data)
Kirk Nason	18 4	Pieter Uittenbogaard	Kirk Nason	(Kirk Nason data)	(Kirk Nason data)
Humberto Acevedo	18 5	Pieter Uittenbogaard	Humberto Acevedo	(Humberto Acevedo d...	(Humberto Acevedo d...
Yoichiro Okada	18 1 6	Pieter Uittenbogaard	Kim Abercrombie	Yoichiro Okada	(Yoichiro Okada data)
Pilar Ackerman	18 2 7	Pieter Uittenbogaard	Sagiv Hadaya	Pilar Ackerman	(Pilar Ackerman data)
Aaron Painter	18 3 8	Pieter Uittenbogaard	Luka Abrus	Aaron Painter	(Aaron Painter data)
Terry Adams	18 4 9	Pieter Uittenbogaard	Kirk Nason	Terry Adams	(Terry Adams data)
David Probst	18 5 10	Pieter Uittenbogaard	Humberto Acevedo	David Probst	(David Probst data)
Manoj Agarwal	18 1 11	Pieter Uittenbogaard	Kim Abercrombie	Manoj Agarwal	(Manoj Agarwal data)
Michael Raheem	18 2 12	Pieter Uittenbogaard	Sagiv Hadaya	Michael Raheem	(Michael Raheem data)
David Ahs	18 3 13	Pieter Uittenbogaard	Luka Abrus	David Ahs	(David Ahs data)
Miguel Saenz	18 3 8 14	Pieter Uittenbogaard	Luka Abrus	Aaron Painter	Miguel Saenz
Kim Akers	18 4 9 15	Pieter Uittenbogaard	Kirk Nason	Terry Adams	Kim Akers
Kate Taneyhill	18 1 16	Pieter Uittenbogaard	Kim Abercrombie	Kate Taneyhill	(Kate Taneyhill data)
David Alexander	18 2 17	Pieter Uittenbogaard	Sagiv Hadaya	David Alexander	(David Alexander data)
Pieter Uittenbogaard	18	Pieter Uittenbogaard	(Pieter Uittenbogaard ...	(Pieter Uittenbogaard ...	(Pieter Uittenbogaard ...
Michelle Alexander	18 4 19	Pieter Uittenbogaard	Kirk Nason	Michelle Alexander	(Michelle Alexander d...

Notable there are no more blank values. A refresh of the PivotTable report reveals the following.

	A	B
1	Row Labels	Vacation Hours
2	[-] Pieter Uittenbogaard	14,041
3	+ (Pieter Uittenbogaard data)	80
4	+ Humberto Acevedo	2,569
5	[-] Kim Abercrombie	3,218
6	+ (Kim Abercrombie data)	40
7	+ David Yalovsky	40
8	+ Diogo Andrade	40
9	+ Kate Taneyhill	40
10	+ Manoj Agarwal	40
11	+ Michael Allen	40
12	+ Yoichiro Okada	2,978
13	+ Kirk Nason	4,611
14	+ Luka Abrus	1,874
15	+ Sagiv Hadaya	1,689
16	Grand Total	14,041

Notice cells B3 and B6 which now meaningfully label the parent member.

Finally, and for completeness, there are two additional DAX functions to support working with self-referencing relationships. They are the PATHLENGTH function which returns the number of parents to the specified item in a given path result, including self, and the PATHCONTAINS function which returns TRUE if a specified item exists within the specified path.

Sample Formulas

Here are some sample formulas meant to illustrate how DAX formulas might be used to address specific business problems. By no means is list meant to be complete, but simply to illustrate specific scenarios. All of these formulas will work with the Contoso sample database, once you have defined a simple measure named Sales using this formula: =SUM(FactSales[SalesAmount]). Many of these samples are formulas that have already been described in this document.

Calculated Columns

Formulas in calculated columns aren't conceptually different from formulas in Excel. DAX does add some new functions, especially in the areas of aggregation and getting data from related tables.

Sample Formula	Comments
=FactSales[SalesQuantity] * FactSales[UnitCost]	Same formula as in Excel
=RELATED(DimProduct[Size])	Get value from related table
=FactSales[SalesQuantity] * RELATED(DimProduct[UnitCost])	Get part of calculation from other table
=SUMX(RELATEDTABLE(FactSales),FactSales[SalesAmount])	Sum over related rows in other table
=COUNTROWS(RELATEDTABLE(FactSales))	Count related rows in other table
=DimProduct[UnitCost] /	Ratio of this product's unit cost to the

AVERAGEX(ALL(DimProduct),DimProduct[UnitCost])	average unit cost over all the products
--	---

Measures

Formulas in measures are conceptually different from anything in Excel because the formula will be evaluated many times using a different context for each evaluation. It can be challenging to anticipate the layout of the PivotTables in which the formula will eventually be evaluated.

Sample Formula	Comments
=SUM(FactSales[SalesAmount])	Simple aggregation can always be sliced
=CALCULATE(SUM(FactSales[SalesAmount]), DimChannel[ChannelName]="Store")	Set context, then evaluate expression
=CALCULATE([Sales], DimChannel[ChannelName]="Store")	Set context, then evaluate measure that is a simple aggregation
=[Sales] (DimChannel[ChannelName]="Store")	Shorthand for CALCULATE (measure)
=[Sales] - CALCULATE([Sales],PREVIOUSYEAR(DimDate[DateKey]))	Growth of [Sales] from last year
=[Sales] - [Sales] (PREVIOUSYEAR(DimDate[DateKey]))	Same calc using shorthand syntax
=[Sales]/[Sales] (ALL(DimProduct))	Ratio to sales of all products
=TOTALQTD([Sales],DimDate[Datekey])	QTD Sales
=SUMX(VALUE(Summary[StoreKey]), SUMX(VALUE(Summary[ProductKey]), CALCULATE(SUM(FactInventory[OnHandQuantity]), LASTNONBLANK(DATESBETWEEN(DimDate[Datekey],BLANK(),MAX(DimDate[Datekey])), CALCULATE(SUM(FactInventory[OnHandQuantity]))))))	Nested aggregation across store and product to find inventory quantity based on last inventory taken for period up through current context

DAX Query

DAX Query is a new capability of DAX added in the SQL Server 2012 release. It allows the user (or client application) to retrieve data defined by a table expression from the tabular model. With an understanding of how to create a table expression you are well on your way to writing your own table queries.

Note that table queries cannot be run in PowerPivot. Table queries can only be executed against a tabular model deployed either to SharePoint (PowerPivot) or to a tabular instance of Analysis Services. The table query can be written and executed within SQL Server Management Studio, or executed programmatically by an application. Note, Power View (the new report authoring tool delivered in SQL Server 2012) dynamically constructs and sends table queries to the model.

Model developers may find it helpful to test table queries in SQL Server Management Studio, and then embed the results into their model calculations.

In its most basic form, a table query is expressed as `EVALUATE <TableExpression>`. `TableExpression` is simply a table in the model, or any expression that returns a table. It is possible to extend this with clauses to define temporary measures (for use during the query execution) and an `ORDER BY` clause.

For further information on DAX Table Queries, refer to the TechNet topic [DAX Table Queries](#).

The following examples provide simple and increasingly sophisticated uses of table queries. (The queries in this paper have been executed in SQL Server Management Studio.)

The first example is a simple table query expression to query the `Store` table.

`EVALUATE`
`Store;`

Store[StoreKey]	Store[Store]	Store[City]	Store[State]	Store[Country]	Store[Continent]
173	Contoso Bridge...	Bridgeport	Connecticut	United States	North America
179	Contoso New L...	New London	Connecticut	United States	North America
256	Contoso Mumba...	Mumbai	Maharashtra	India	Asia
253	Contoso Sydney...	Sydney	New South Wales	Australia	Asia
236	Contoso Baumh...	Baumholder	Rhineland-Palati...	Germany	Europe
221	Contoso Strasb...	Strasbourg	Bas-Rhin	France	Europe

This next example uses the `FILTER` function to filter the rows of the table with an expression. Any valid table expression can be evaluated to produce a query result.

`EVALUATE`
`FILTER(Store, [Sales] > 100000000);`

Store[StoreKey]	Store[Store]	Store[City]	Store[State]	Store[Country]	Store[Continent]
308	Contoso North ...	Seattle	Washington	United States	North America
199	Contoso North ...	Bethesda	Maryland	United States	North America
307	Contoso Asia O...	Beijing	Beijing	China	Asia
200	Contoso Catalo...	North Harford	Maryland	United States	North America
310	Contoso Asia R...	Beijing	Beijing	China	Asia
306	Contoso Europe...	Berlin	Berlin	Germany	Europe
309	Contoso Europe...	Paris	Seine (Paris)	France	Europe

In the following example, the expression uses the `CROSSJOIN` function to perform a cross join of all `Store` and `Product` rows, and then returns a filtered result based on the expression.

```

EVALUATE
FILTER(
    CROSSJOIN(Store, Product)
    ,[Sales] > 5000000);

```

Store[StoreKey]	Store[Store]	Store[City]	Store[State]	Store[Country]	Store[Continent]	Product[Product...]	Product[Product]
200	Contoso Catalo...	North Harford	Maryland	United States	North America	1857	NT Washer & D..
200	Contoso Catalo...	North Harford	Maryland	United States	North America	1847	NT Washer & D..

This example uses the CROSSJOIN function to cross join the result of two VALUES functions. This way, only the Store table's Continent column and the Product table's Product table are returned in the cross joined result. The result of the cross join is filtered by the expression, and then then ADDCOLUMNS function is used to add the Sales measure as a column to the table with the alias of "Total Sales", and the measure will be evaluated for each row in this table.

```

EVALUATE
ADDCOLUMNS(
    FILTER(
        CROSSJOIN(VALUES(Store[Continent]))
        ,VALUES(Product[Product]))
        ,[Sales] > 25000000)
    ,"Total Sales", [Sales]);

```

Store[Continent]	Product[Product]	[Total Sales]
North America	Proseware Proje...	28787332.5
North America	Proseware Proje...	30305373
North America	Proseware Proje...	29064453.75
North America	Proseware Proje...	31421926.2
North America	Proseware Proje...	27902610
North America	Proseware Proje...	31144162.35
North America	Contoso Project...	28040310

This example uses the GENERATE function in combination with the VALUES and TOPN function. The result is each Continent and its top five products by sales. For each row in the first argument, the GENERATE function will execute the function in the second argument.

```

EVALUATE
GENERATE(
    VALUES(Store[Continent]),
    TOPN(5, VALUES(Product[Product]), [Sales]));

```

Store[Continent]	Product[Product]
North America	Proseware Projector 1080p LCD86 Black
North America	Proseware Projector 1080p LCD86 White
North America	Proseware Projector 1080p DLP86 Silver
North America	Proseware Projector 1080p DLP86 Black
North America	Proseware Projector 1080p DLP86 White
Europe	Proseware Projector 1080p DLP86 Black
Europe	Contoso Projector 1080p X980 Black
Europe	Proseware Projector 1080p DLP86 Silver
Europe	Proseware Projector 1080p DLP86 White
Europe	Contoso Projector 1080p X980 White
Asia	Litware Refrigerator 24.7CuFt X980 Grey

This example uses the SUMMARIZE function based on the Sales table to “group by” Category and Year, and then produces a summary using the Sales measure aliased as “Total Sales”. The SUMMARIZE function returns a summary table for the requested totals over a set of groups. It can help prepare data for sophisticated calculations. For those readers familiar with T-SQL SELECT statement, this is the equivalent of writing a query using the GROUP BY clause. Notice you do not need to define the relationships between tables when using SUMMARIZE because the relationship is defined in the model.

```

EVALUATE
SUMMARIZE(
    Sales
    ,Product[Category]
    ,'Date'[Year]
    ,"Total Sales", [Sales]);

```

Product[Category]	Date[Year]	[Total Sales]
Audio	Year 2007	29734671.9068
Audio	Year 2008	52932396.0524
Audio	Year 2009	68947296.3517
TV and Video	Year 2007	426671354.2612
TV and Video	Year 2008	473265849.6053

This example is a variation of the previous example, and includes an ORDER BY clause.

```

EVALUATE
SUMMARIZE(
    Sales
    ,Product[Category]
    ,'Date'[Year]
    ,"Total Sales", [Sales])
ORDER BY [Year], [Total Sales] DESC;

```

Product[Category]	Date[Year]	[Total Sales]
Home Appliances	Year 2007	1375117996.8146
Computers	Year 2007	1146469996.5735
Cameras and camcorders	Year 2007	1102693917.4784
TV and Video	Year 2007	426671354.2612
Cell phones	Year 2007	363847591.0821

This final example defines a measure on the Sales table named Profit that is then used in the table query. Note the measure exists only for the duration of the table query and are not stored in the data model.

```

DEFINE MEASURE
    Sales[Profit] = [Sales] - [Cost]
EVALUATE
SUMMARIZE(
    Sales
    ,Product[Category]
    ,'Date'[Year]
    ,"Total Sales", [Sales]
    ,"Total Profit", [Profit])
ORDER BY [Year], [Category];

```

Product[Category]	Date[Year]	[Total Sales]	[Total Profit]
Audio	Year 2007	29734671.9068	17336920.4468
Cameras and camcorders	Year 2007	1102693917.4784	666989743.5784
Cell phones	Year 2007	363847591.0821	203458975.5321
Computers	Year 2007	1146469996.5735	651122146.3635
Games and Toys	Year 2007	42429666.0755	23746689.5655
Home Appliances	Year 2007	1375117996.8146	759744791.8546

Additional Resources

This paper introduced many of the fundamental concepts and functions included in the DAX formula language. DAX, however, has immense capabilities in performing highly advanced dynamic calculations that are beyond the scope of what can be described here.

There are many additional resources you can use to learn about DAX formulas and how they can be used effectively in your tabular model solutions.

Information resources provided and maintained by Microsoft:

[Books Online for SQL Server 2012](http://go.microsoft.com/fwlink/?LinkID=181772&clcid=0x409) at <http://go.microsoft.com/fwlink/?LinkID=181772&clcid=0x409>.

[Data Analysis Expressions \(DAX Reference\)](http://go.microsoft.com/fwlink/?LinkID=181772&clcid=0x409) on MSDN at <http://go.microsoft.com/fwlink/?LinkID=181772&clcid=0x409>.

The [Analysis Services and PowerPivot Team Blog](http://go.microsoft.com/fwlink/?LinkID=220949&clcid=0x409) at (<http://go.microsoft.com/fwlink/?LinkID=220949&clcid=0x409>) provides information, tips, news and announcements about SQL Server 2012 Analysis Services (SSAS) Release Candidate 0 (RC 0) and PowerPivot.

The [DAX Resource Center Wiki](http://go.microsoft.com/fwlink/?LinkID=220966&clcid=0x409) at (<http://go.microsoft.com/fwlink/?LinkID=220966&clcid=0x409>) provides both internal and external information on DAX, including numerous DAX solutions submitted by leading Business Intelligence professionals.

Microsoft PowerPivot for Excel 2010: Give Your Data Meaning published by Microsoft Press at <http://www.microsoft.com/learning/en/us/Book.aspx?ID=14569&locale=en-us>.

The [Tabular Modeling \(Adventure Works Tutorial\)](http://msdn.microsoft.com/en-us/library/hh231691(v=SQL.110).aspx) at [http://msdn.microsoft.com/en-us/library/hh231691\(v=SQL.110\).aspx](http://msdn.microsoft.com/en-us/library/hh231691(v=SQL.110).aspx) provides step-by-step instructions on how to create a tabular model project in SSDT that includes many calculations in calculated columns, measures, and row filters. For most formulas, a description about what the formula is meant to do is provided.

Information and resources provided by BI professionals outside of Microsoft:

<http://www.powerpivotpro.com/>

http://sqlblog.com/blogs/alberto_ferrari/default.aspx

http://sqlblog.com/blogs/marco_russo/

<http://cwebbbi.wordpress.com/>

<http://paultebraak.wordpress.com/>

<http://www.bp-msbi.com/>

<http://prologika.com/CS/blogs/blog/default.aspx>

<http://www.powerpivotblog.nl/>