Data Analysis Expressions (DAX) Reference

SQL Server 2012 Books Online

Reference

Microsoft

Data Analysis Expressions (DAX) Reference

SQL Server 2012 Books Online

Summary: The Data Analysis Expressions (DAX) language is a library of functions and operators that can be combined to build formulas and expressions.

Category: Reference
Applies to: SQL Server 2012
Source: SQL Server Books Online (link to source content)
E-book publication date: June 2012

Contents

Data Analysis Expressions (DAX) Reference

The Data Analysis Expressions (DAX) language is a library of functions and operators that can be combined to build formulas and expressions. This section provides topics that describe function syntax and other attributes of the DAX language.

For an overview of how you can use DAX formulas, see Getting Started with Data Analysis Expressions (DAX).

In this Section

DAX Syntax Specification

Operator Reference

Function Reference

See Also

Key Concepts in DAX

Data Types Supported in PowerPivot Workbooks

DAX Syntax Specification for PowerPivot

Data Analysis Expressions (DAX) is a library of functions, operators, and constants that can be combined to build formulas and expressions in PowerPivot for Excel. This section provides details about the syntax and requirements of the DAX language.

For examples of the kinds of formulas that you can build, and how you can use expressions to filter tables and change context, see Getting Started with Data Analysis Expressions (DAX). This topic contains the following sections:

Syntax Requirements

Naming Requirements

Functions

Operators and Constants

Data Types

Syntax Requirements

DAX formulas are very similar to the formulas you type in Excel tables, but there are some key differences.

- In Microsoft Excel you can reference individual cells or arrays; in PowerPivot you can reference only complete tables or columns of data. However, If you need to work

with only part of a column, or with unique values from a column, you can achieve similar behavior by using DAX functions that filter the column or return unique values.

- DAX formulas do not support exactly the same data types as Microsoft Excel. In general, DAX provides more data types than Excel does, and DAX performs implicit type conversions on some data when importing. For more information, see Data Types in DAX.

A DAX formula always starts with an equal sign (=). After the equals sign, you can provide any expression that evaluates to a scalar, or an expression that can be converted to a scalar. These include the following:

- A scalar constant, or expression that uses a scalar operator (+,-,*,/,>=,...,&&, ...)
- References to columns or tables. The DAX language always uses tables and columns as inputs to functions, never an array or arbitrary set of values.
- Operators, constants, and values provided as part of an expression.
- The result of a function and its required arguments. Some DAX functions return a table instead of a scalar, and must be wrapped in a function that evaluates the table and returns a scalar; unless the table is a single column, single row table, then it is treated as a scalar value.

  Most PowerPivot functions require one or more arguments, which can include tables, columns, expressions, and values. However, some functions, such as PI, do not require any arguments, but always require parentheses to indicate the null argument. For example, you must always type PI(), not PI. You can also nest functions within other functions.
- Expressions. An expression can contain any or all of the following: operators, constants, or references to columns.

For example, the following are all valid formulas.

Formula	Result
=3	3
="**Sales**"	**Sales**
='Sales'[Amount]	If you use this formula within the Sales table, you will get the value of the column Amount in the Sales table for the current row.
=(0.03 *[Amount]) =0.03 * [Amount]	Three percent of the value in the Amount column of the current table. Although this formula can be used to calculate a percentage, the result is not

Formula	Result
	shown as a percentage unless you apply formatting in the table.
=PI()	The value of the constant pi.

Note
Formulas can behave differently depending on whether they are used in a calculated column, or in a measure within a PivotTable. You must always be aware of the context and how the data that you use in the formula is related to other data that might be used in the calculation. For more information, see Context in DAX Formulas.

Naming Requirements

A PowerPivot window can contain multiple tables, each on its own tab. Together the tables and their columns comprise a database stored in the PowerPivot xVelocity in-memory analytics engine (VertiPaq). Within that database, all tables must have unique names. The names of columns must also be unique within each table. All object names are case-insensitive; for example, the names **SALES** and **Sales** would represent the same table.

Each column and measure that you add to an existing PowerPivot database must belong to a specific table. You specify the table that contains the column either implicitly, when you create a calculated column within a table, or explicitly, when you create a measure and specify the name of the table where the measure definition should be stored.

When you use a table or column as an input to a function, you must generally *qualify* the column name. The *fully qualified* name of a column is the table name, followed by the column name in square brackets: for examples, 'U.S. Sales'[Products]. A fully qualified name is always required when you reference a column in the following contexts:

- As an argument to the function, VALUES
- As an argument to the functions, ALL or ALLEXCEPT
- In a filter argument for the functions, CALCULATE or CALCULATETABLE
- As an argument to the function, RELATEDTABLE
- As an argument to any time intelligence function

An *unqualified* column name is just the name of the column, enclosed in brackets: for example, [Sales Amount]. For example, when you are referencing a scalar value from the same row of the current table, you can use the unqualified column name.

If the name of a table contains spaces, reserved keywords, or disallowed characters, you must enclose the table name in single quotation marks. You must also enclose table names in quotation marks if the name contains any characters outside the ANSI

alphanumeric character range, regardless of whether your locale supports the character set or not. For example, if you open a workbook that contains table names written in Cyrillic characters, such as 'Таблица', the table name must be enclosed in quotation marks, even though it does not contain spaces.

Note
To make it easier to enter the fully qualified names of columns, we recommend that you use the formula AutoComplete feature in the client.

Tables

- Table names are required whenever the column is from a different table than the current table. Table names must be unique within the database.
- Table names must be enclosed in single quotation marks if they contain spaces, other special characters or any non-English alphanumeric characters.

Measures

- Measure names must always be in brackets.
- Measure names can contain spaces.
- Each measure name must be unique within a database. Therefore, the table name is optional in front of a measure name when referencing an existing measure. However, when you create a measure you must always specify a table where the measure definition will be stored.

Columns

Column names must be unique in the context of a table; however, multiple tables can have columns with the same names (disambiguation comes with the table name).

In general, columns can be referenced without referencing the base table that they belong to, except when there might be a name conflict to resolve or with certain functions that require column names to be fully qualified.

Reserved Keywords

If the name that you use for a table is the same as an Analysis Services reserved keyword, an error is raised, and you must rename the table. However, you can use keywords in object names if the object name is enclosed in brackets (for columns) or quotation marks (for tables).

Note
Note that quotation marks can be represented by several different characters, depending on the application. If you paste formulas from an external document or Web page, make sure to check the ASCII code of the character that is used for opening and closing quotes, to ensure that they are the same. Otherwise DAX

may be unable to recognize the symbols as quotation marks, making the reference invalid.

Special Characters

The following characters and character types are not valid in the names of tables, columns, or measures:

- Leading or trailing spaces; unless the spaces are enclosed by name delimiters, brackets, or single apostrophes.
- Control characters
- The following characters that are not valid in the names of PowerPivot objects: .,;':/*|?&%$!+=()[]{}<>

Examples of Object Names

The following table shows examples of some object names:

Object Types	Examples	Comment
Table name	**Sales**	If the table name does not contain spaces or other special characters, the name does not need to be enclosed in quotation marks.
Table name	**'Canada Sales'**	If the name contains spaces, tabs or other special characters, enclose the name in single quotation marks.
Fully qualified column name	**Sales[Amount]**	The table name precedes the column name, and the column name is enclosed in brackets.
Fully qualified measure name	**Sales[Profit]**	The table name precedes the measure name, and the measure name is enclosed in brackets. In certain contexts, a fully qualified name is always required.
Unqualified column name	**[Amount]**	The unqualified name is just

		the column name, in brackets. Contexts where you can use the unqualified name include formulas in a calculated column within the same table, or in an aggregation function that is scanning over the same table.
Fully qualified column in table with spaces	**'Canada Sales'[Qty]**	The table name contains spaces, so it must be surrounded by single quotes.

Note
To make it easier to enter the fully qualified names of columns, we recommend that you use the AutoComplete feature when building formulas. For more information, see Building Formulas for Calculated Columns and Measures.

Miscellaneous Restrictions

The syntax required for each function, and the type of operation it can perform, varies greatly depending on the function. In general, however, the following rules apply to all formulas and expressions:

- DAX formulas and expressions cannot modify or insert individual values in tables.
- You cannot create calculated rows by using DAX. You can create only calculated columns and measures.
- When defining calculated columns, you can nest functions to any level.
- DAX has several functions that return a table. Typically, you use the values returned by these functions as input to other functions, which require a table as input.

Functions in DAX

DAX provides the following types of functions.

- Date and Time Functions
- Filter Functions
- Information Functions
- Logical Functions
- Math and Trigonometric Functions
- Statistical Functions

- Text Functions

DAX Operators and Constants

The following table lists the operators that are supported by DAX. In general, operators in DAX behave the same way that they do in Microsoft Excel, with some minor exceptions. For more information about the syntax of individual operators, see Operator Reference.

Operator Type	Symbol and Use
Parenthesis operator	() precedence order and grouping of arguments
Arithmetic operators	+ (addition) - (subtraction/ sign) * (multiplication) / (division) ^ (exponentiation)
Comparison operators	= (equal to) > (greater than) < (less than) >= (greater than or equal to) <= (less than or equal to) <> (not equal to)
Text concatenation operator	& (concatenation)
Logic operators	&& (and) || (or)

Data Types in DAX

You do not need to cast, convert, or otherwise specify the data type of a column or value that you use in a DAX formula. When you use data in a DAX formula, DAX automatically identifies the data types in referenced columns and of the values that you type in, and performs implicit conversions where necessary to complete the specified operation.

For example, if you try to add a number to a date value, PowerPivot will interpret the operation in the context of the function, like Excel does, and convert the numbers to a common data type, and then present the result in the intended format, a date.

However, there are some limitations on the values that can be successfully converted. If a value or a column has a data type that is incompatible with the current operation, DAX returns an error. Also, DAX does not provide functions that let you explicitly change, convert, or cast the data type of existing data that you have imported into a PowerPivot workbook.

Important
PowerPivot does not support use of the variant data type used in Excel. Therefore, when you load or import data, it is expected that the data in each column is generally of a consistent data type.

Some functions return scalar values, including strings, whereas other functions work with numbers, both integers and real numbers, or dates and times. The data type required for each function is described in the section, Function Reference.

Tables are a new data type in PowerPivot. You can use tables containing multiple columns and multiple rows of data as the argument to a function. Some functions also return tables, which are stored in memory and can be used as arguments to other functions.

For more information about the different numeric and date/time data types, and for details on the handling of nulls and empty strings, see Data Types Supported in PowerPivot Workbooks.

See Also

Build a formula

DAX Operator Reference for PowerPivot

The Data Analysis Expression (DAX) language uses operators to create expressions that compare values, perform arithmetic calculations, or work with strings. This section describes the use of each operator.

Types of Operators

There are four different types of calculation operators: arithmetic, comparison, text concatenation, and logical.

Arithmetic Operators

To perform basic mathematical operations such as addition, subtraction, or multiplication; combine numbers; and produce numeric results, use the following arithmetic operators.

Arithmetic operator	Meaning	Example
+ (plus sign)	Addition	3+3
– (minus sign)	Subtraction or sign	3–1–1
* (asterisk)	Multiplication	3*3
/ (forward slash)	Division	3/3
^ (caret)	Exponentiation	16^4

Note

The plus sign can function both as a *binary operator* and as a *unary operator*. A binary operator requires numbers on both sides of the operator and performs addition. When you use values in a DAX formula on both sides of the binary operator, DAX tries to cast the values to numeric data types if they are not already numbers. In contrast, the unary operator can be applied to any type of argument. The plus symbol does not affect the type or value and is simply ignored, whereas the minus operator creates a negative value, if applied to a numeric value.

Comparison Operators

You can compare two values with the following operators. When two values are compared by using these operators, the result is a logical value, either TRUE or FALSE.

Comparison operator	Meaning	Example
=	Equal to	[Region] = "USA"
>	Greater than	[Sales Date] > "Jan 2009"
<	Less than	[Sales Date] < "Jan 1 2009"
>=	Greater than or equal to	[Amount] >= 20000
<=	Less than or equal to	[Amount] <= 100
<>	Not equal to	[Region] <> "USA"

Text Concatenation Operator

Use the ampersand (&) to join, or concatenate, two or more text strings to produce a single piece of text.

Text operator	Meaning	Example
& (ampersand)	Connects, or concatenates, two values to produce one continuous text value	[Region] & ", " & [City]

Logical Operators

Use logical operators (&&) and (||) to combine expressions to produce a single result.

Text operator	Meaning	Examples
&& (double ampersand)	Creates an AND condition between two expressions that each have a Boolean result. If both expressions return TRUE, the combination of the expressions also returns TRUE; otherwise the combination returns FALSE.	([Region] = "France") && ([BikeBuyer] = "yes"))
|| (double pipe symbol)	Creates an OR condition between two logical expressions. If either expression returns TRUE, the result is TRUE; only when both expressions are FALSE is the result FALSE.	(([Region] = "France") || ([BikeBuyer] = "yes"))

Operators and Precedence Order

In some cases, the order in which calculation is performed can affect the return value; therefore, it is important to understand how the order is determined and how you can change the order to obtain the desired results.

Calculation Order

An expression evaluates the operators and values in a specific order. All expressions always begin with an equal sign (=). The equal sign indicates that the succeeding characters constitute an expression.

Following the equal sign are the elements to be calculated (the operands), which are separated by calculation operators. Expressions are always read from left to right, but the order in which the elements are grouped can be controlled to some degree by using parentheses.

Operator Precedence

If you combine several operators in a single formula, the operations are ordered according to the following table. If the operators have equal precedence value, they are ordered from left to right. For example, if an expression contains both a multiplication and division operator, they are evaluated in the order that they appear in the expression, from left to right.

Operator	Description
^	Exponentiation
–	Sign (as in –1)
* and /	Multiplication and division
!	NOT (unary operator)
+ and –	Addition and subtraction
&	Connects two strings of text (concatenation)
=< ><=>=<>	Comparison

Using Parentheses to Control Calculation Order

To change the order of evaluation, you should enclose in parentheses that part of the formula that must be calculated first. For example, the following formula produces 11 because multiplication is calculated before addition. The formula multiplies 2 by 3, and then adds 5 to the result.

```
=5+2*3
```

In contrast, if you use parentheses to change the syntax, the order is changed so that 5 and 2 are added together, and the result multiplied by 3 to produce 21.

```
=(5+2)*3
```

In the following example, the parentheses around the first part of the formula force the calculation to evaluate the expression `(3 + 0.25)` first and then divide the result by the result of the expression, `(3 - 0.25)`.

```
=(3 + 0.25)/(3 - 0.25)
```

In the following example, the exponentiation operator is applied first, according to the rules of precedence for operators, and then the sign operator is applied. The result for this expression is -4.

```
=-2^2
```

To ensure that the sign operator is applied to the numeric value first, you can use parentheses to control operators, as shown in the following example. The result for this expression is 4.

```
= (-2)^2
```

Compatibility Notes

DAX easily handles and compares various data types, much like Microsoft Excel. However, the underlying computation engine is based on SQL Server Analysis Services and provides additional advanced features of a relational data store, including richer support for date and time types. Therefore, in some cases the results of calculations or the behavior of functions may not be the same as in Excel. Moreover, DAX supports more data types than does Excel. This section describes the key differences.

Coercing Data Types of Operands

In general, the two operands on the left and right sides of any operator should be the same data type. However, if the data types are different, DAX will convert them to a common data type for comparison, as follows:

1. First, both operands are converted to the largest possible common data type.
2. Next, the operands are compared.

For example, suppose you have two numbers that you want to combine. One number results from a formula, such as `=[Price] * .20`, and the result may contain many decimal places. The other number is an integer that has been provided as a string value.

In this case, DAX will convert both numbers to real numbers in a numeric format, using the largest numeric format that can store both kinds of numbers. Then DAX will compare the values.

In contrast, Excel tries to compare values of different types without first coercing them to a common type. For this reason, you may see different results in DAX than in Excel for the same comparison expression.

Data Types used in DAX	Data Types used in Excel
Numbers (I8, R8)	Numbers (R8)
Boolean	Boolean
String	String
DateTime	Variant

Data Types used in DAX	Data Types used in Excel
Currency	Currency

For more information about implicit data type conversion, see Data Types Supported in PowerPivot Workbooks.

Differences in Precedence Order

The precedence order of operations in DAX formulas is basically the same as that used by Microsoft Excel, but some Excel operators are not supported, such as percent. Also, ranges are not supported.

Therefore, whenever you copy and paste formulas from Excel, be sure to review the formula carefully, as some operators or elements in the formulas may not be valid. When there is any doubt about the order in which operations are performed, we recommend that you use parentheses to control the order of operations and remove any ambiguity about the result.

See Also

Basic DAX Syntax

Using Data Analysis Expressions

Parameter-Naming Conventions

Parameter names are standardized in DAX reference to facilitate the usage and understanding of the functions.

Parameter Names

Parameter	Description
expression	Any DAX expression that returns a single scalar value, where the expression is to be evaluated multiple times (for each row/context).
value	Any DAX expression that returns a single scalar value where the expression is to be evaluated exactly once before all other operations.
table	Any DAX expression that returns a table of

	data.
tableName	The name of an existing table using standard DAX syntax. It cannot be an expression.
columnName	The name of an existing column using standard DAX syntax, usually fully qualified. It cannot be an expression.
name	A string constant that will be used to provide the name of a new object.
order	An enumeration used to determine the sort order.
ties	An enumeration used to determine the handling of tie values.
type	An enumeration used to determine the data type for PathItem and PathItemReverse.

Prefixing parameter names or using the prefix only

prefixing	Parameter names may be further qualified with a prefix that is descriptive of how the argument is used and to avoid ambiguous reading of the parameters. For example:

Result_ColumnName	Refers to an existing column used to get the result values in the LOOKUPVALUE() function.
Search_ColumnName	Refers to an existing column used to search for a value in the

	LOOKUPVALUE() function.
omitting	Parameter names will be omitted if the prefix is clear enough to describe the parameter. For example, instead of having the following syntax DATE (Year_Value, Month_Value, Day_Value) it is clearer for the user to read DATE (Year, Month, Day); repeating three times the suffix value does not add anything to a better comprehension of the function and it clutters the reading unnecessarily. However, if the prefixed parameter is Year_columnName then the parameter name and the prefix will stay to make sure the user understands that the parameter requires a reference to an existing column of Years.

Understanding Functions for Parent-Child Hierarchies in DAX

DAX provides five functions to help users manage data that is presented as a parent-child hierarchy in their models. With this functions a user can obtain the entire lineage of parents a row has, how many levels has the lineage to the top parent, who is the parent n-levels above the current row, who is the n-descendant from the top of the current row hierarchy and is certain parent a parent in the current row hierarchy?

Parent-Child functions in DAX

The following table contains a Parent-Child hierarchy on the columns: **EmployeeKey** and **ParentEmployeeKey** that is used in all the functions examples.

EmployeeKey	**ParentEmployeeKey**

112	
14	112
3	14
11	3
13	3
162	3
117	162
221	162
81	162

In the above table you can see that employee 112 has no parent defined, employee 14 has employee 112 as manager (ParentEmployeeKey), employee 3 has employee 14 as manager and employees 11, 13, and 162 have employee 3 as manager. The above helps to understand that employee 112 has no manager above her/him and she/he is the top manager for all employees shown here; also, employee 3 reports to employee 14 and employees 11, 13, 162 report to 3.

The following table presents the available functions, a brief description of the function and an example of the function over the same data shown above.

<table>
<tr><th>Function</th><th>Description / Example</th></tr>
<tr><td>PATH Function (DAX)</td><td>Returns a delimited text with the identifiers of all the parents to the current row, starting with the oldest or top most until current.
In the following example column ‘Path’ is defined as
<code>'=PATH(EmployeeKey, ParentEmployeeKey)'</code>
<table>
<tr><th>EmployeeKey</th><th>ParentEmployeeKey</th><th>Path</th></tr>
<tr><td>112</td><td></td><td>112</td></tr>
<tr><td>14</td><td>112</td><td>112|14</td></tr>
<tr><td>3</td><td>14</td><td>112|14|3</td></tr>
<tr><td>11</td><td>3</td><td>112|14|3|11</td></tr>
<tr><td>13</td><td>3</td><td>112|14|3|13</td></tr>
<tr><td>162</td><td>3</td><td>112|14|3|162</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Function</th><th>Description / Example</th></tr>
<tr><td></td><td>
<table>
<tr><td>117</td><td>162</td><td>112|14|3|162|117</td></tr>
<tr><td>221</td><td>162</td><td>112|14|3|162|221</td></tr>
<tr><td>81</td><td>162</td><td>112|14|3|162|81</td></tr>
</table>
</td></tr>
<tr><td>PATHLENGTH Function (DAX)</td><td>Returns the number of levels in a given PATH(), starting at current level until the oldest or top most parent level.
In the following example column PathLength is defined as '<code>=PATHLENGTH([Path])</code>'; the example includes all data from the Path() example to help understand how this function works.
<table>
<tr><th>EmployeeKey</th><th>ParentEmployeeKey</th><th>Path</th><th>PathLength</th></tr>
<tr><td>112</td><td></td><td>112</td><td>1</td></tr>
<tr><td>14</td><td>112</td><td>112|14</td><td>2</td></tr>
<tr><td>3</td><td>14</td><td>112|14|3</td><td>3</td></tr>
<tr><td>11</td><td>3</td><td>112|14|3|11</td><td>4</td></tr>
<tr><td>13</td><td>3</td><td>112|14|3|13</td><td>4</td></tr>
<tr><td>162</td><td>3</td><td>112|14|3|162</td><td>4</td></tr>
<tr><td>117</td><td>162</td><td>112|14|3|162|117</td><td>5</td></tr>
<tr><td>221</td><td>162</td><td>112|14|3|162|221</td><td>5</td></tr>
<tr><td>81</td><td>162</td><td>112|14|3|162|81</td><td>5</td></tr>
</table>
</td></tr>
<tr><td>PATHITEM Function (DAX)</td><td>Returns the item at the specified position from a PATH() like result, counting from left to right.
In the following example column PathItem - 4th from left is defined as '<code>=PATHITEM([Path], 4)</code>'; this example returns the EmployeKey at fourth position in the Path string from the left, using the same sample data from the Path() example.</td></tr>
</table>

<table>
<tr><th>Function</th><th>Description / Example</th></tr>
<tr><td></td><td>
<table>
<tr><th>EmployeeKey</th><th>ParentEmployeeKey</th><th>Path</th><th>PathItem - 4th from left</th></tr>
<tr><td>112</td><td></td><td>112</td><td></td></tr>
<tr><td>14</td><td>112</td><td>112|14</td><td></td></tr>
<tr><td>3</td><td>14</td><td>112|14|3</td><td></td></tr>
<tr><td>11</td><td>3</td><td>112|14|3|11</td><td>11</td></tr>
<tr><td>13</td><td>3</td><td>112|14|3|13</td><td>13</td></tr>
<tr><td>162</td><td>3</td><td>112|14|3|162</td><td>162</td></tr>
<tr><td>117</td><td>162</td><td>112|14|3|162|117</td><td>162</td></tr>
<tr><td>221</td><td>162</td><td>112|14|3|162|221</td><td>162</td></tr>
<tr><td>81</td><td>162</td><td>112|14|3|162|81</td><td>162</td></tr>
</table>
</td></tr>
<tr><td>PATHITEMREVERSE Function (DAX)</td><td>Returns the item at position from a PATH() like function result, counting backwards from right to left.
In the following example column PathItemReverse - 3rd from right is defined as '<code>=PATHITEMREVERSE([Path], 3)</code>'; this example returns the EmployeKey at third position in the Path string from the right, using the same sample data from the Path() example.
<table>
<tr><th>EmployeeKey</th><th>ParentEmployeeKey</th><th>Path</th><th>PathItemReverse - 3rd from right</th></tr>
<tr><td>112</td><td></td><td>112</td><td></td></tr>
<tr><td>14</td><td>112</td><td>112|14</td><td></td></tr>
<tr><td>3</td><td>14</td><td>112|14|3</td><td>112</td></tr>
<tr><td>11</td><td>3</td><td>112|14|3|11</td><td>14</td></tr>
<tr><td>13</td><td>3</td><td>112|14|3|13</td><td>14</td></tr>
<tr><td>162</td><td>3</td><td>112|14|3|162</td><td>14</td></tr>
<tr><td>117</td><td>162</td><td>112|14|3|162|117</td><td>3</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><th>Function</th><th>Description / Example</th></tr>
<tr><td></td><td>
<table>
<tr><td>221</td><td>162</td><td>112|14|3|162|221</td><td>3</td></tr>
<tr><td>81</td><td>162</td><td>112|14|3|162|81</td><td>3</td></tr>
</table>
</td></tr>
<tr><td>PATHCONTAINS Function (DAX)</td><td>Returns TRUE if the specified item exists within the specified path.
In the following example column PathContains - employee 162 is defined as '<code>=PATHCONTAINS([Path], "162")</code>'; this example returns TRUE if the given path contains employee 162. This example uses the results from the Path() example above.
<table>
<tr><th>EmployeeKey</th><th>ParentEmployeeKey</th><th>Path</th><th>PathContains - employee 162</th></tr>
<tr><td>112</td><td></td><td>112</td><td>FALSE</td></tr>
<tr><td>14</td><td>112</td><td>112|14</td><td>FALSE</td></tr>
<tr><td>3</td><td>14</td><td>112|14|3</td><td>FALSE</td></tr>
<tr><td>11</td><td>3</td><td>112|14|3|11</td><td>FALSE</td></tr>
<tr><td>13</td><td>3</td><td>112|14|3|13</td><td>FALSE</td></tr>
<tr><td>162</td><td>3</td><td>112|14|3|162</td><td>TRUE</td></tr>
<tr><td>117</td><td>162</td><td>112|14|3|162|117</td><td>TRUE</td></tr>
</table>
</td></tr>
</table>

Warning
In SQL Server 2012 Analysis Services, the xVelocity in-memory analytics engine (VertiPaq) does not support the definition of parent-child hierarchies; however, the DAX language provides a set of functions that allows users to explore parent-child hierarchies and to use these hierarchies in formulas.

DAX Function Reference

This section provides detailed syntax for the functions and operators used in Data Analysis Expression formulas, together with examples. For general information about DAX, see DAX Reference.

In this Section

DAX Table Queries

Date and Time Functions (DAX)

Filter Functions (DAX)

Information Functions (DAX)

Logical Functions (DAX)

Math and Trigonometric Functions (DAX)

Statistical Functions (DAX)

Text Functions (DAX)

See Also

Operator Reference (DAX)

DAX Syntax Specification

DAX Queries

The DAX language offers a new syntax to return table data from a query.

In this Section

In this section you will find:

Table Query syntax reference

Parameters for Table Queries

Reference

- Execute Method (XMLA)
- Statement Element (XMLA)

Related Sections

DAX Query Syntax Reference

DAX queries allow the user to retrieve data defined by a table expression from the xVelocity in-memory analytics engine (VertiPaq). The user can create measures as part of the query; these measures exist only for the duration of the query.

Syntax

[DEFINE { MEASURE <tableName>**[**<name>**]** = <expression> }

EVALUATE <table>

[ORDER BY {<expression> [{ASC | DESC}]}[, …]

[START AT {<value>|<parameter>} [, …]]]

Parameters

Parameter	Description
DEFINE clause	An optional clause of the query statement that allows the user to define measures for the duration of the query. Definitions can reference other definitions that appear before or after the current definition.
tableName	The name of an existing table using standard DAX syntax. It cannot be an expression.
name	The name of a new measure. It cannot be an expression.
expression	Any DAX expression that returns a single scalar value.
EVALUATE clause	Contains the table expression that generates the results of the query. The expression can use any of the defined measures. The expression must return a table. If a scalar value is required, the person authoring the measure can wrap their scalar inside a ROW() function to produce a table that contains the required scalar.
ORDER BY clause	Optional clause that defines the expression(s) used to sort the query results. Any expression that can be evaluated for each row of the result is valid.
START AT sub-clause	Optional clause, inside an **ORDER BY** clause, that defines the values at which the query

	results will start. The **START AT** clause is part of the **ORDER BY** clause and cannot be used outside it. In an ordered set of results, the **START AT** clause defines the starting row for the result set. The **START AT** arguments have a one to one correspondence with the columns in the ORDER BY clause; there can be as many arguments in the START AT clause as there are in the ORDER BY clause, but not more. The first argument in the **START AT** defines the starting value in column 1 of the **ORDER BY** columns. The second argument in the **START AT** defines the starting value in column 2 of the **ORDER BY** columns within the rows that meet the first value for column 1.
value	A constant value; it cannot be an expression.
parameter	The name of a parameter in the XMLA statement prefixed with an **@** character. For more information, see Parameters for DAX Queries.

Return Value

A table of data.

Code Examples

For examples of using DAX queries, see this site.

Parameters for DAX Queries

This topic illustrates how to pass parameter values in an XMLA structure to a DAX query statement.

Parameters in XMLA and DAX queries

A well-defined DAX query statement would benefit enormously by being able to be parameterized and then used, over and over, with just changes in the parameter values.

The Execute method, in XMLA, has a Parameters collection element that allows parameters to be defined and assigned a value; within the collection, each Parameter element defines the name of the parameter and a value to it.

The DAX query syntax allows you to reference XMLA parameters by prefixing the name, of the parameter, with an **@** character. Hence, any place in the syntax where a value is

allowed it can be replaced with a parameter call. However, one thing needs to be remembered: all XMLA parameters are typed as text.

Warning
Parameters defined in the parameters section and not used in the **<STATEMENT>** element generate an error response in XMLA.

Warning
Parameters used and not defined in the **<Parameters>** element generate an error response in XMLA.

Code Examples

For examples of using DAX queries, see this site.

Date and Time Functions

Many of the date and time functions in DAX are very similar to the Excel date and time functions. However, DAX functions use a **datetime** data type, and can take values from a column as an argument. DAX also includes a set of *time intelligence functions* that enable you to manipulate data using time periods, including days, months, quarters, and years, and then build and compare calculations over those periods.

In this Section

Time Intelligence Functions (DAX)
DATE Function
DATEVALUE Function (DAX)
DAY Function
EDATE Function
EOMONTH Function
HOUR Function
MINUTE Function
MONTH Function
NOW Function
SECOND Function
TIME Function (DAX)
TIMEVALUE Function (DAX)
TODAY Function
WEEKDAY Function
WEEKNUM Function
YEAR Function

YEARFRAC Function

Reference

Using DAX

Basic DAX Syntax

Related Sections

Date and Time Functions (DAX)

Aggregation Functions (DAX)

Logical Functions (DAX)

Filter and Value Functions (DAX)

Math and Trigonometric Functions (DAX)

See Also

Getting Started with Data Analysis Expressions (DAX)

Time Intelligence Functions

Data Analysis Expressions (DAX) includes time intelligence functions to support the needs of Business Intelligence analysis by enabling you to manipulate data using time periods, including days, months, quarters, and years, and then build and compare calculations over those periods. The following time intelligence functions are available in DAX.

In this Section

CLOSINGBALANCEMONTH Function (DAX)

CLOSINGBALANCEQUARTER Function (DAX)

CLOSINGBALANCEYEAR Function (DAX)

DATEADD Function (DAX)

DATESBETWEEN Function (DAX)

DATESINPERIOD Function (DAX)

DATESMTD Function (DAX)

DATESQTD Function (DAX)

DATESYTD Function (DAX)

ENDOFMONTH Function (DAX)

ENDOFQUARTER Function (DAX)

ENDOFYEAR Function (DAX)

FIRSTDATE Function (DAX)

FIRSTNONBLANK Function (DAX)

LASTDATE Function (DAX)

LASTNONBLANK Function (DAX)
NEXTDAY Function (DAX)
NEXTMONTH Function (DAX)
NEXTQUARTER Function (DAX)
NEXTYEAR Function (DAX)
OPENINGBALANCEMONTH Function (DAX)
OPENINGBALANCEQUARTER Function (DAX)
OPENINGBALANCEYEAR Function (DAX)
PARALLELPERIOD Function (DAX)
PREVIOUSDAY Function (DAX)
PREVIOUSMONTH Function (DAX)
PREVIOUSQUARTER Function (DAX)
PREVIOUSYEAR Function (DAX)
SAMEPERIODLASTYEAR Function (DAX)
STARTOFMONTH Function (DAX)
STARTOFQUARTER Function (DAX)
STARTOFYEAR Function (DAX)
TOTALMTD Function (DAX)
TOTALQTD Function (DAX)
TOTALYTD Function (DAX)

See Also

Function Reference (DAX)
Date and Time Functions (DAX)
Filter and Value Functions (DAX)
Information Functions (DAX)
Logical Functions (DAX)
Math and Trigonometric Functions (DAX)
Statistical Functions (DAX)
Text Functions (DAX)

CLOSINGBALANCEMONTH Function

Evaluates the **expression** at the last date of the month in the current context.

Syntax

CLOSINGBALANCEMONTH(<expression>,<dates>[,<filter>])

Parameters

Parameter	Definition
expression	An expression that returns a scalar value.
dates	A column that contains dates.
filter	(optional) An expression that specifies a filter to apply to the current context.

Return Value

A scalar value that represents the **expression** evaluated at the last date of the month in the current context.

Remarks

Note
To understand more about how context affects the results of formulas, see Context.

The **dates** argument can be any of the following:

- A reference to a date/time column.
- A table expression that returns a single column of date/time values.
- A Boolean expression that defines a single-column table of date/time values.

Note
Constraints on Boolean expressions are described in the topic, CALCULATE Function (DAX).

Note
The **filter** expression has restrictions described in the topic, CALCULATE Function (DAX).

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that calculates the 'Month End Inventory Value' of the product inventory.

To see how this works, create a PivotTable and add the fields, CalendarYear, MonthNumberOfYear and DayNumberOfMonth, to the **Row Labels** area of the PivotTable. Then add a measure, named **Month End Inventory Value**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=CLOSINGBALANCEMONTH(SUMX(ProductInventory,ProductInventory[UnitCost]*P
roductInventory[UnitsBalance]),DateTime[DateKey])
```

See Also

Time intelligence functions
CLOSINGBALANCEYEAR
CLOSINGBALANCEQUARTER
Get Sample Data

CLOSINGBALANCEQUARTER Function

Evaluates the **expression** at the last date of the quarter in the current context.

Syntax

CLOSINGBALANCEQUARTER(<expression>,<dates>[,<filter>])

Parameters

Parameter	Definition
expression	An expression that returns a scalar value.
dates	A column that contains dates.
filter	(optional) An expression that specifies a filter to apply to the current context.

Return Value

A scalar value that represents the **expression** evaluated at the last date of the quarter in the current context.

Remarks

Note
To understand more about how context affects the results of formulas, see Context.

The **dates** argument can be any of the following:

- A reference to a date/time column.
- A table expression that returns a single column of date/time values.
- A Boolean expression that defines a single-column table of date/time values.

Note

Constraints on Boolean expressions are described in the topic, CALCULATE Function (DAX).

Note
The **filter** expression has restrictions described in the topic, CALCULATE Function (DAX).

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models,
see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that calculates the 'Quarter End Inventory Value' of the product inventory.

To see how this works, create a PivotTable and add the fields, CalendarYear, CalendarQuarter and MonthNumberOfYear, to the **Row Labels** area of the PivotTable. Then add a measure, named **Quarter End Inventory Value**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=CLOSINGBALANCEQUARTER(SUMX(ProductInventory,ProductInventory[UnitCost]
*ProductInventory[UnitsBalance]),DateTime[DateKey])
```

See Also

Time intelligence functions
CLOSINGBALANCEYEAR
CLOSINGBALANCEMONTH
Get Sample Data

CLOSINGBALANCEYEAR Function

Evaluates the **expression** at the last date of the year in the current context.

Syntax

CLOSINGBALANCEYEAR(<expression>,<dates>[,<filter>][,<year_end_date>])

Parameters

Parameter	Definition
expression	An expression that returns a scalar value.
dates	A column that contains dates.
filter	(optional) An expression that specifies a filter to apply to the current context.

year_end_date	(optional) A literal string with a date that defines the year-end date. The default is December 31.

Return Value

A scalar value that represents the **expression** evaluated at the last date of the year in the current context.

Remarks

Note
To understand more about how context affects the results of formulas, see Context.

The **dates** argument can be any of the following:

- A reference to a date/time column.
- A table expression that returns a single column of date/time values.
- A Boolean expression that defines a single-column table of date/time values.

Note
Constraints on Boolean expressions are described in the topic, CALCULATE Function (DAX).

Note
The **filter** expression has the restrictions described in the topic, CALCULATE Function (DAX).

The **year_end_date** parameter is a string literal of a date, in the same locale as the locale of the client where the workbook was created. The year portion of the date is ignored.

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that calculates the 'Year End Inventory Value' of the product inventory.

To see how this works, create a PivotTable and add the field, CalendarYear, to the **Row Labels** area of the PivotTable. Then add a measure, named **Year End Inventory Value**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=CLOSINGBALANCEYEAR(SUMX(ProductInventory,ProductInventory[UnitCost]*Pr
oductInventory[UnitsBalance]),DateTime[DateKey])
```

See Also

Time intelligence functions
CLOSINGBALANCEYEAR
CLOSINGBALANCEQUARTER
CLOSINGBALANCEMONTH
Get Sample Data

DATEADD Function

Returns a table that contains a column of dates, shifted either forward or backward in time by the specified number of intervals from the dates in the current context.

Syntax

DATEADD(<dates>,<number_of_intervals>,<interval>)

Parameters

Term	Definition
dates	A column that contains dates.
number_of_intervals	An integer that specifies the number of intervals to add to or subtract from the dates.
interval	The interval by which to shift the dates. The value for interval can be one of the following: `year`, `quarter`, `month`, `day`

Return Value

A table containing a single column of date values.

Remarks

Note
To understand more about how context affects the results of formulas, see Context.

The **dates** argument can be any of the following:

- A reference to a date/time column,
- A table expression that returns a single column of date/time values,
- A Boolean expression that defines a single-column table of date/time values.

Note

Constraints on Boolean expressions are described in the topic, CALCULATE Function (DAX).

If the number specified for **number_of_intervals** is positive, the dates in **dates** are moved forward in time; if the number is negative, the dates in **dates** are shifted back in time.

The **interval** parameter is an enumeration, not a set of strings; therefore values should not be enclosed in quotation marks. Also, the values: `year`, `quarter`, `month`, `day` should be spelled in full when using them.

The result table includes only dates that exist in the **dates** column.

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models,
see http://go.microsoft.com/fwlink/?LinkId=219172.

Example: Shifting a Set of Dates

Description

The following formula calculates dates that are one year before the dates in the current context.

Code

```
=DATEADD(DateTime[DateKey],-1,year)
```

See Also

Time intelligence functions

Date and time functions

Get Sample Data

DATESBETWEEN Function

Returns a table that contains a column of dates that begins with the **start_date** and continues until the **end_date**.

Syntax

DATESBETWEEN(<dates>,<start_date>,<end_date>)

Parameters

Term	Definition
dates	A reference to a date/time column.
start_date	A date expression.
end_date	A date expression.

Return Value

A table containing a single column of date values.

Remarks

If **start_date** is a blank date value, then **start_date** will be the earliest value in the **dates** column.

If **end_date** is a blank date value, then **end_date** will be the latest value in the **dates** column.

The dates used as the **start_date** and **end_date** are inclusive: that is, if the sales occurred on September 1 and you use September 1 as the start date, sales on September 1 are counted.

Note

The DATESBETWEEN function is provided for working with custom date ranges. If you are working with common date intervals such as months, quarters, and years, we recommend that you use the appropriate function, such as DATESINPERIOD.

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models,
see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

Description

The following sample formula creates a measure that calculates the 'Summer 2007 sales' for the Internet sales.

To see how this works, create a PivotTable and add the field, CalendarYear, to the **Row Labels** area of the PivotTable. Then add a measure, named **Summer 2007 Sales**, using the formula as defined in the code section, to the **Values** area of the PivotTable.

Code

```
=CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]),
DATESBETWEEN(DateTime[DateKey],
    DATE(2007,6,1),
    DATE(2007,8,31)
  ))
```

See Also

Time intelligence functions

Date and time functions

DATESINPERIOD

DATESINPERIOD Function

Returns a table that contains a column of dates that begins with the **start_date** and continues for the specified **number_of_intervals**.

Syntax

DATESINPERIOD(<dates>,<start_date>,<number_of_intervals>,<interval>)

Parameters

Term	Definition
dates	A column that contains dates.
start_date	A date expression.
number_of_intervals	An integer that specifies the number of intervals to add to or subtract from the dates.
interval	The interval by which to shift the dates. The value for interval can be one of the following: `year`, `quarter`, `month`, `day`

Return Value

A table containing a single column of date values.

Remarks

Note
To understand more about how context affects the results of formulas, see Context.

The **dates** argument can be any of the following:

- A reference to a date/time column,
- A table expression that returns a single column of date/time values,
- A Boolean expression that defines a single-column table of date/time values.

Note
Constraints on Boolean expressions are described in the topic, CALCULATE Function (DAX).

If the number specified for **number_of_intervals** is positive, the dates are moved forward in time; if the number is negative, the dates are shifted back in time.

The **interval** parameter is an enumeration, not a set of strings; therefore values should not be enclosed in quotation marks. Also, the values: `year`, `quarter`, `month`, `day` should be spelled in full when using them.

The result table includes only dates that appear in the values of the underlying table column.

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models,
see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

Description

The following formula returns the Internet sales for the 21 days prior to August 24, 2007.

Code

```
=
CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]),DATESINPERIOD(DateTim
e[DateKey],DATE(2007,08,24),-21,day))
```

See Also

Time intelligence functions

Date and time functions

DATESBETWEEN

Get Sample Data

DATESMTD Function

Returns a table that contains a column of the dates for the month to date, in the current context.

Syntax

DATESMTD(<dates>)

Parameters

Term	Definition
dates	A column that contains dates.

Property Value/Return Value

A table containing a single column of date values.

Remarks

Note
To understand more about how context affects the results of formulas, see Context.

The **dates** argument can be any of the following:

- A reference to a date/time column.
- A table expression that returns a single column of date/time values.
- A Boolean expression that defines a single-column table of date/time values.

Note
Constraints on Boolean expressions are described in the topic, CALCULATE Function (DAX).

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

Description

The following sample formula creates a measure that calculates the 'Month To Date Total' for the Internet sales.

To see how this works, create a PivotTable and add the fields, CalendarYear, MonthNumberOfYear and DayNumberOfMonth, to the **Row Labels** area of the PivotTable. Then add a measure, named **Month To Date Total**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]),
DATESMTD(DateTime[DateKey]))
```

See Also

Time intelligence functions

Date and time functions

DATESYTD

DATESQTD

Get Sample Data

DATESQTD Function

Returns a table that contains a column of the dates for the quarter to date, in the current context.

Syntax

DATESQTD(<dates>)

Parameters

Term	Definition
dates	A column that contains dates.

Property Value/Return Value

A table containing a single column of date values.

Remarks

Note
To understand more about how context affects the results of formulas, see Context.

The **dates** argument can be any of the following:

- A reference to a date/time column.
- A table expression that returns a single column of date/time values.
- A Boolean expression that defines a single-column table of date/time values.

Note
Constraints on Boolean expressions are described in the topic, CALCULATE Function (DAX).

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

Description

The following sample formula creates a measure that calculates the 'Quarterly Running Total' of the internet sales.

To see how this works, create a PivotTable and add the fields, CalendarYear, CalendarQuarter and MonthNumberOfYear to the **Row Labels** area of the PivotTable. Then add a measure, named **Quarterly Running Total**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]),
DATESQTD(DateTime[DateKey]))
```

See Also

Time intelligence functions

Date and time functions

DATESYTD

DATESMTD

Get Sample Data

DATESYTD Function

Returns a table that contains a column of the dates for the year to date, in the current context.

Syntax

DATESYTD(<dates> [,<year_end_date>])

Parameters

Term	Definition
dates	A column that contains dates.
year_end_date	(optional) A literal string with a date that defines the year-end date. The default is December 31.

Property Value/Return Value

A table containing a single column of date values.

Remarks

Note
To understand more about how context affects the results of formulas, see Context.

The **dates** argument can be any of the following:

- A reference to a date/time column,
- A table expression that returns a single column of date/time values,
- A Boolean expression that defines a single-column table of date/time values.

Note
Constraints on Boolean expressions are described in the topic, CALCULATE Function (DAX).

The **year_end_date** parameter is a string literal of a date, in the same locale as the locale of the client where the workbook was created. The year portion of the date is ignored.

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

Description

The following sample formula creates a measure that calculates the 'Running Total' for the Internet sales.

To see how this works, create a PivotTable and add the fields, CalendarYear and CalendarQuarter, to the **Row Labels** area of the PivotTable. Then add a measure named **Running Total**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]),
DATESYTD(DateTime[DateKey]))
```

See Also

Time intelligence functions

Date and time functions

DATESMTD

DATESQTD

Get Sample Data

ENDOFMONTH Function

Returns the last date of the month in the current context for the specified column of dates.

Syntax

ENDOFMONTH(<dates>)

Parameters

Term	Definition
dates	A column that contains dates.

Return Value

A table containing a single column and single row with a date value.

Remarks

Note
To understand more about how context affects the results of formulas, see Context.

The **dates** argument can be any of the following:

- A reference to a date/time column.
- A table expression that returns a single column of date/time values.
- A Boolean expression that defines a single-column table of date/time values.

Note
Constraints on Boolean expressions are described in the topic, CALCULATE Function (DAX).

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models,
see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

Description

The following sample formula creates a measure that returns the end of the month, for the current context.

To see how this works, create a PivotTable and add the fields CalendarYear and MonthNumberOfYear to the **Row Labels** area of the PivotTable. Then add a measure, named **EndOfMonth**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=ENDOFMONTH(DateTime[DateKey])
```

See Also

Date and time functions

time intelligence functions

endofyear

endofquarter

ENDOFQUARTER Function

Returns the last date of the quarter in the current context for the specified column of dates.

Syntax

ENDOFQUARTER(<dates>)

Parameters

Term	Definition
dates	A column that contains dates.

Return Value

A table containing a single column and single row with a date value.

Remarks

Note
To understand more about how context affects the results of formulas, see Context.

The **dates** argument can be any of the following:

- A reference to a date/time column,
- A table expression that returns a single column of date/time values,
- A Boolean expression that defines a single-column table of date/time values.

Note
Constraints on Boolean expressions are described in the topic, CALCULATE Function (DAX).

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

Description

The following sample formula creates a measure that returns the end of the quarter, for the current context.

To see how this works, create a PivotTable and add the fields CalendarYear and MonthNumberOfYear to the **Row Labels** area of the PivotTable. Then add a measure, named **EndOfQuarter**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=ENDOFQUARTER(DateTime[DateKey])
```

See Also

Date and time functions

time intelligence functions

endofyear

endofmonth

ENDOFYEAR Function

Returns the last date of the year in the current context for the specified column of dates.

Syntax

ENDOFYEAR(<dates> [,<year_end_date>])

Parameters

Term	Definition
dates	A column that contains dates.
year_end_date	(optional) A literal string with a date that defines the year-end date. The default is December 31.

Return Value

A table containing a single column and single row with a date value.

Remarks

Note
To understand more about how context affects the results of formulas, see Context.

The **dates** argument can be any of the following:

- A reference to a date/time column,
- A table expression that returns a single column of date/time values,
- A Boolean expression that defines a single-column table of date/time values.

Note
Constraints on Boolean expressions are described in the topic, CALCULATE Function (DAX).

The **year_end_date** parameter is a string literal of a date, in the same locale as the locale of the client where the workbook was created. The year portion of the date is ignored.

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

Description

The following sample formula creates a measure that returns the end of the fiscal year that ends on June 30, for the current context.

To see how this works, create a PivotTable and add the field CalendarYear to the **Row Labels** area of the PivotTable. Then add a measure, named **EndOfFiscalYear**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=ENDOFYEAR(DateTime[DateKey],"06/30/2004")
```

See Also

Date and time functions

time intelligence functions

endofmonth

endofquarter

FIRSTDATE Function

Returns the first date in the current context for the specified column of dates.

Syntax

FIRSTDATE(<dates>)

Parameters

Term	Definition
dates	A column that contains dates.

Return Value

A table containing a single column and single row with a date value.

Remarks

Note
To understand more about how context affects the results of formulas, see Context.

The **dates** argument can be any of the following:

- A reference to a date/time column.
- A table expression that returns a single column of date/time values,.
- A Boolean expression that defines a single-column table of date/time values.

Note
Constraints on Boolean expressions are described in the topic, CALCULATE Function (DAX).

When the current context is a single date, the date returned by the FIRSTDATE and LASTDATE functions will be equal.

Technically, the return value is a table that contains a single column and single value. Therefore, this function can be used as an argument to any function that requires a table in its arguments. Also, the returned value can be used whenever a date value is required.

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models,
see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

Description

The following sample formula creates a measure that obtains the first date when a sale was made in the Internet sales channel for the current context.

To see how this works, create a PivotTable and add the field CalendarYear to the **Row Labels** area of the PivotTable. Then add a measure, named **FirstSaleDate**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=FIRSTDATE('InternetSales_USD'[SaleDateKey])
```

See Also

Date and time functions

Time Intelligence functions

LASTDATE

FIRSTNONBLANKDATE

Get Sample Data

FIRSTNONBLANK Function

Returns the first value in the column, **column**, filtered by the current context, where the expression is not blank.

Syntax

FIRSTNONBLANK(<column>,<expression>)

Parameters

Term	Definition
column	A column expression.
expression	An expression evaluated for blanks for each value of **column**.

Property Value/Return Value

A table containing a single column and single row with the computed first value.

Remarks

The **column** argument can be any of the following:

- A reference to any column.
- A table with a single column.
- A Boolean expression that defines a single-column table .

Note
Constraints on Boolean expressions are described in the topic, CALCULATE Function (DAX).

This function is typically used to return the first value of a column for which the expression is not blank. For example, you could get the last value for which there were sales of a product.

Note
To understand more about how context affects the results of formulas, see Context.

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see http://go.microsoft.com/fwlink/?LinkId=219172.

See Also

LASTNONBLANK Function (DAX)

Statistical Functions (DAX)

Function Reference (DAX)

Context in DAX Formulas

Working with Relationships in Formulas

Get Sample Data

LASTDATE Function

Returns the last date in the current context for the specified column of dates.

Syntax

LASTDATE(<dates>)

Parameters

Term	Definition
dates	A column that contains dates.

Return Value

A table containing a single column and single row with a date value.

Remarks

Note
To understand more about how context affects the results of formulas, see Context.

The **dates** argument can be any of the following:

- A reference to a date/time column,
- A table expression that returns a single column of date/time values,
- A Boolean expression that defines a single-column table of date/time values.

Note
Constraints on Boolean expressions are described in the topic, CALCULATE Function (DAX).

When the current context is a single date, the date returned by the FIRSTDATE and LASTDATE functions will be equal.

Technically, the return value is a table that contains a single column and single value. Therefore, this function can be used as an argument to any function that requires a table in its arguments. Also, the returned value can be used whenever a date value is required.

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

Description

The following sample formula creates a measure that obtains the last date, for the current context, when a sale was made in the Internet sales channel.

To see how this works, create a PivotTable and add the field CalendarYear to the **Row Labels** area of the PivotTable. Then add a measure, named **LastSaleDate**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=LASTDATE('InternetSales_USD'[SaleDateKey])
```

See Also

Date and time functions

Time Intelligence functions

FIRSTDATE

LASTNONBLANKDATE

Get Sample Data

LASTNONBLANK Function

Returns the last value in the column, **column**, filtered by the current context, where the expression is not blank.

Syntax

LASTNONBLANK(<column>,<expression>)

Parameters

Term	Definition
column	A column expression.
expression	An expression evaluated for blanks for each value of **column**.

Property Value/Return Value

A table containing a single column and single row with the computed last value.

Remarks

The **column** argument can be any of the following:

- A reference to any column.
- A table with a single column.
- A Boolean expression that defines a single-column table

Note
Constraints on Boolean expressions are described in the topic, CALCULATE Function (DAX).

This function is typically used to return the last value of a column for which the expression is not blank. For example, you could get the last value for which there were sales of a product.

Note
To understand more about how context affects the results of formulas, see Context.

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see http://go.microsoft.com/fwlink/?LinkId=219172.

See Also

FIRSTNONBLANK Function (DAX)

Statistical Functions (DAX)

Function Reference (DAX)
Context in DAX Formulas
Working with Relationships in Formulas
Get Sample Data

NEXTDAY Function

Returns a table that contains a column of all dates from the next day, based on the first date specified in the **dates** column in the current context.

Syntax

NEXTDAY(<dates>)

Parameters

Term	Definition
dates	A column containing dates.

Return Value

A table containing a single column of date values.

Remarks

Note
To understand more about how context affects the results of formulas, see Context.

This function returns all dates from the next day to the first date in the input parameter. For example, if the first date in the **dates** argument refers to June 10, 2009; then this function returns all dates equal to June 11, 2009.

The **dates** argument can be any of the following:

- A reference to a date/time column.
- A table expression that returns a single column of date/time values.
- A Boolean expression that defines a single-column table of date/time values.

Note
Constraints on Boolean expressions are described in the topic, CALCULATE Function (DAX).

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

Description

The following sample formula creates a measure that calculates the 'next day sales' of the internet sales.

To see how this works, create a PivotTable and add the fields, CalendarYear and MonthNumberOfYear, to the **Row Labels** area of the PivotTable. Then add a measure, named **Next Day Sales**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]),
NEXTDAY('DateTime'[DateKey]))
```

See Also

Time intelligence functions

Date and time functions

NEXTQUARTER

NEXTMONTH

NEXTYEAR

Examples

NEXTMONTH Function

Returns a table that contains a column of all dates from the next month, based on the first date in the **dates** column in the current context.

Syntax

NEXTMONTH(<dates>)

Parameters

Term	Definition
dates	A column containing dates.

Return Value

A table containing a single column of date values.

Remarks

Note

To understand more about how context affects the results of formulas, see Context.

This function returns all dates from the next day to the first date in the input parameter. For example, if the first date in the **dates** argument refers to June 10, 2009; then this function returns all dates for the month of July, 2009.

The **dates** argument can be any of the following:

- A reference to a date/time column.
- A table expression that returns a single column of date/time values.
- A Boolean expression that defines a single-column table of date/time values.

Note
Constraints on Boolean expressions are described in the topic, CALCULATE Function (DAX).

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

Description

The following sample formula creates a measure that calculates the 'next month sales' for the Internet sales.

To see how this works, create a PivotTable and add the fields, CalendarYear and MonthNumberOfYear, to the **Row Labels** area of the PivotTable. Then add a measure, named **Next Month Sales**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]),
NEXTMONTH('DateTime'[DateKey]))
```

See Also

Time intelligence functions

Date and time functions

NEXTDAY

NEXTQUARTER

NEXTYEAR

Get Sample Data

NEXTQUARTER Function

Returns a table that contains a column of all dates in the next quarter, based on the first date specified in the **dates** column, in the current context.

Syntax

NEXTQUARTER(<dates>)

Parameters

Term	Definition
dates	A column containing dates.

Return Value

A table containing a single column of date values.

Remarks

Note
To understand more about how context affects the results of formulas, see Context.

This function returns all dates in the next quarter, based on the first date in the input parameter. For example, if the first date in the **dates** column refers to June 10, 2009, this function returns all dates for the quarter July to September, 2009.

The **dates** argument can be any of the following:

- A reference to a date/time column.
- A table expression that returns a single column of date/time values.
- A Boolean expression that defines a single-column table of date/time values.

Note
Constraints on Boolean expressions are described in the topic, CALCULATE Function (DAX).

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

Description

The following sample formula creates a measure that calculates the 'next quarter sales' for the Internet sales.

To see how this works, create a PivotTable and add the fields, CalendarYear and CalendarQuarter, to the **Row Labels** area of the PivotTable. Then add a measure, named **Next Quarter Sales**, using the formula defined in the code section to the **Values** area of the PivotTable.

Code

```
=CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]),
NEXTQUARTER('DateTime'[DateKey]))
```

See Also

Time intelligence functions

Date and time functions

NEXTDAY

NEXTMONTH

NEXTYEAR

Get Sample Data

NEXTYEAR Function

Returns a table that contains a column of all dates in the next year, based on the first date in the **dates** column, in the current context.

Syntax

NEXTYEAR(<dates>[,<year_end_date>])

Parameters

Term	Definition
dates	A column containing dates.
year_end_date	(optional) A literal string with a date that defines the year-end date. The default is December 31.

Return Value

A table containing a single column of date values.

Remarks

Note

To understand more about how context affects the results of formulas, see Context.

This function returns all dates in the next year, based on the first date in the input column. For example, if the first date in the **dates** column refers to the year 2007, this function returns all dates for the year 2008.

The **dates** argument can be any of the following:

- A reference to a date/time column.

- A table expression that returns a single column of date/time values.
- A Boolean expression that defines a single-column table of date/time values.

Note
Constraints on Boolean expressions are described in the topic, CALCULATE Function (DAX).

The **year_end_date** parameter is a string literal of a date, in the same locale as the locale of the client where the workbook was created. The year portion of the date is ignored.

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models,
see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

Description

The following sample formula creates a measure that calculates the 'next year sales' for the Internet sales.

To see how this works, create a PivotTable and add the fields, CalendarYear and CalendarQuarter, to the **Row Labels** area of the PivotTable. Then add a measure, named **Next Year Sales**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]),
NEXTYEAR('DateTime'[DateKey]))
```

See Also

Time intelligence functions
Date and time functions
NEXTDAY
NEXTQUARTER
NEXTMONTH
Get Sample Data

OPENINGBALANCEMONTH Function

Evaluates the **expression** at the first date of the month in the current context.

Syntax

OPENINGBALANCEMONTH(<expression>,<dates>[,<filter>])

Parameters

Parameter	Definition
expression	An expression that returns a scalar value.
dates	A column that contains dates.
filter	(optional) An expression that specifies a filter to apply to the current context.

Return Value

A scalar value that represents the **expression** evaluated at the first date of the month in the current context.

Remarks

Note
To understand more about how context affects the results of formulas, see Context.

The **dates** argument can be any of the following:

- A reference to a date/time column.
- A table expression that returns a single column of date/time values.
- A Boolean expression that defines a single-column table of date/time values.

Note
Constraints on Boolean expressions are described in the topic, CALCULATE Function (DAX).

Note

- The **filter** expression has restrictions described in the topic, CALCULATE Function (DAX).
- This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that calculates the 'Month Start Inventory Value' of the product inventory.

To see how this works, create a PivotTable and add the fields, CalendarYear, MonthNumberOfYear and DayNumberOfMonth, to the **Row Labels** area of the PivotTable. Then add a measure, named **Month Start Inventory Value**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=OPENINGBALANCEMONTH(SUMX(ProductInventory,ProductInventory[UnitCost]*P
roductInventory[UnitsBalance]),DateTime[DateKey])
```

See Also

OPENINGBALANCEYEAR

OPENINGBALANCEQUARTER

Time Intelligence Functions

closingbalancemonth

Get Sample Data

OPENINGBALANCEQUARTER Function

Evaluates the **expression** at the first date of the quarter, in the current context.

Syntax

OPENINGBALANCEQUARTER(<expression>,<dates>[,<filter>])

Parameters

Parameter	Definition
expression	An expression that returns a scalar value.
dates	A column that contains dates.
filter	(optional) An expression that specifies a filter to apply to the current context.

Return Value

A scalar value that represents the **expression** evaluated at the first date of the quarter in the current context.

Remarks

Note
To understand more about how context affects the results of formulas, see Context.

The **dates** argument can be any of the following:

- A reference to a date/time column.
- A table expression that returns a single column of date/time values,
- A Boolean expression that defines a single-column table of date/time values.

Note

Constraints on Boolean expressions are described in the topic, CALCULATE Function (DAX).

Note

- The **filter** expression has restrictions described in the topic, CALCULATE Function (DAX).
- This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that calculates the 'Quarter Start Inventory Value' of the product inventory.

To see how this works, create a PivotTable and add the fields, CalendarYear, CalendarQuarter and MonthNumberOfYear, to the **Row Labels** area of the PivotTable. Then add a measure, named **Quarter Start Inventory Value**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=OPENINGBALANCEQUARTER(SUMX(ProductInventory,ProductInventory[UnitCost]
*ProductInventory[UnitsBalance]),DateTime[DateKey])
```

See Also

OPENINGBALANCEYEAR

OPENINGBALANCEMONTH

Time Intelligence Functions

closingbalancequarter

Get Sample Data

OPENINGBALANCEYEAR Function

Evaluates the **expression** at the first date of the year in the current context.

Syntax

OPENINGBALANCEYEAR(<expression>,<dates>[,<filter>][,<year_end_date>])

Parameters

Parameter	Definition
expression	An expression that returns a scalar value.
dates	A column that contains dates.

filter	(optional) An expression that specifies a filter to apply to the current context.
year_end_date	(optional) A literal string with a date that defines the year-end date. The default is December 31.

Return Value

A scalar value that represents the **expression** evaluated at the first date of the year in the current context.

Remarks

Note
To understand more about how context affects the results of formulas, see Context.

The **dates** argument can be any of the following:

- A reference to a date/time column.
- A table expression that returns a single column of date/time values.
- A Boolean expression that defines a single-column table of date/time values.

Note
Constraints on Boolean expressions are described in the topic, CALCULATE Function (DAX).

Note
The **filter** expression has restrictions described in the topic, CALCULATE Function (DAX).

The **year_end_date** parameter is a string literal of a date, in the same locale as the locale of the client where the workbook was created. The year portion of the date is ignored.

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that calculates the 'Year Start Inventory Value' of the product inventory.

To see how this works, create a PivotTable and add the field, CalendarYear, to the **Row Labels** area of the PivotTable. Then add a measure, named **Year Start Inventory Value**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=OPENINGBALANCEYEAR(SUMX(ProductInventory,ProductInventory[UnitCost]*Pr
oductInventory[UnitsBalance]),DateTime[DateKey])
```

See Also

OPENINGBALANCEQUARTER

OPENINGBALANCEMONTH

Time Intelligence Functions

Closingbalanceyear

Get Sample Data

PARALLELPERIOD Function

Returns a table that contains a column of dates that represents a period parallel to the dates in the specified **dates** column, in the current context, with the dates shifted a number of intervals either forward in time or back in time.

Syntax

PARALLELPERIOD(<dates>,<number_of_intervals>,<interval>)

Parameters

Term	Definition
dates	A column that contains dates.
number_of_intervals	An integer that specifies the number of intervals to add to or subtract from the dates.
interval	The interval by which to shift the dates. The value for interval can be one of the following: `year`, `quarter`, `month`.

Return Value

A table containing a single column of date values.

Remarks

This function takes the current set of dates in the column specified by **dates**, shifts the first date and the last date the specified number of intervals, and then returns all contiguous dates between the two shifted dates. If the interval is a partial range of month, quarter, or year then any partial months in the result are also filled out to complete the entire interval.

Note

To understand more about how context affects the results of formulas, see Context.

The **dates** argument can be any of the following:

- A reference to a date/time column,
- A table expression that returns a single column of date/time values,
- A Boolean expression that defines a single-column table of date/time values.

Note
Constraints on Boolean expressions are described in the topic, CALCULATE Function (DAX).

If the number specified for **number_of_intervals** is positive, the dates in **dates** are moved forward in time; if the number is negative, the dates in **dates** are shifted back in time.

The **interval** parameter is an enumeration, not a set of strings; therefore values should not be enclosed in quotation marks. Also, the values: `year`, `quarter`, `month` should be spelled in full when using them.

The result table includes only dates that appear in the values of the underlying table column.

The PARALLELPERIOD function is similar to the DATEADD function except that PARALLELPERIOD always returns full periods at the given granularity level instead of the partial periods that DATEADD returns. For example, if you have a selection of dates that starts at June 10 and finishes at June 21 of the same year, and you want to shift that selection forward by one month then the PARALLELPERIOD function will return all dates from the next month (July 1 to July 31); however, if DATEADD is used instead, then the result will include only dates from July 10 to July 21.

If the dates in the current context do not form a contiguous interval, the function returns an error.

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models,
see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

Description

The following sample formula creates a measure that calculates the previous year sales for Internet sales.

To see how this works, create a PivotTable and add the fields, CalendarYear and CalendarQuarter, to the **Row Labels** area of the PivotTable. Then add a measure, named **Previous Year Sales**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Note

The above example uses the table DateTime from the DAX sample workbook. For more information about samples, see Get Sample Data .

Code

```
=CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]),
PARALLELPERIOD(DateTime[DateKey],-1,year))
```

See Also

Time intelligence functions

Date and time functions

DATEADD

Get Sample Data

PREVIOUSDAY Function

Returns a table that contains a column of all dates representing the day that is previous to the first date in the **dates** column, in the current context.

Syntax

PREVIOUSDAY(<dates>)

Parameters

Term	Definition
dates	A column containing dates.

Return Value

A table containing a single column of date values.

Remarks

Note
To understand more about how context affects the results of formulas, see Context.

This function determines the first date in the input parameter, and then returns all dates corresponding to the day previous to that first date. For example, if the first date in the **dates** argument refers to June 10, 2009; this function returns all dates equal to June 9, 2009.

The **dates** argument can be any of the following:

- A reference to a date/time column.
- A table expression that returns a single column of date/time values.

- A Boolean expression that defines a single-column table of date/time values.

Note
Constraints on Boolean expressions are described in the topic, CALCULATE Function (DAX).

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models,
see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

Description

The following sample formula creates a measure that calculates the 'previous day sales' for the Internet sales.

To see how this works, create a PivotTable and add the fields, CalendarYear and MonthNumberOfYear, to the **Row Labels** area of the PivotTable. Then add a measure, named **Previous Day Sales**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]),
PREVIOUSDAY('DateTime'[DateKey]))
```

See Also

Time intelligence functions

Date and time functions

PREVIOUSMONTH

PREVIOUSQUARTER

PREVIOUSYEAR

PREVIOUSMONTH Function

Returns a table that contains a column of all dates from the previous month, based on the first date in the **dates** column, in the current context.

Syntax

PREVIOUSMONTH(<dates>)

Parameters

Term	Definition
Dates	A column containing dates.

Return Value

A table containing a single column of date values.

Remarks

Note

To understand more about how context affects the results of formulas, see Context.

This function returns all dates from the previous month, using the first date in the column used as input. For example, if the first date in the **dates** argument refers to June 10, 2009, this function returns all dates for the month of May, 2009.

The **dates** argument can be any of the following:

- A reference to a date/time column.
- A table expression that returns a single column of date/time values.
- A Boolean expression that defines a single-column table of date/time values.

Note

Constraints on Boolean expressions are described in the topic, CALCULATE Function (DAX).

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

Description

The following sample formula creates a measure that calculates the 'previous month sales' for the Internet sales.

To see how this works, create a PivotTable and add the fields, CalendarYear and MonthNumberOfYear, to the **Row Labels** area of the PivotTable. Then add a measure, named **Previous Month Sales**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]),
PREVIOUSMONTH('DateTime'[DateKey]))
```

See Also

Time intelligence functions

Date and time functions

PREVIOUSDAY

PREVIOUSQUARTER

PREVIOUSYEAR

Get Sample Data

PREVIOUSQUARTER Function

Returns a table that contains a column of all dates from the previous quarter, based on the first date in the **dates** column, in the current context.

Syntax

PREVIOUSQUARTER(<dates>)

Parameters

Term	Definition
dates	A column containing dates.

Return Value

A table containing a single column of date values.

Remarks

Note
To understand more about how context affects the results of formulas, see Context.

This function returns all dates from the previous quarter, using the first date in the input column. For example, if the first date in the **dates** argument refers to June 10, 2009, this function returns all dates for the quarter January to March, 2009.

The **dates** argument can be any of the following:

- A reference to a date/time column.
- A table expression that returns a single column of date/time values.
- A Boolean expression that defines a single-column table of date/time values.

Note
Constraints on Boolean expressions are described in the topic, CALCULATE Function (DAX).

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

Description

The following sample formula creates a measure that calculates the 'previous quarter sales' for Internet sales.

To see how this works, create a PivotTable and add the fields, CalendarYear and CalendarQuarter, to the **Row Labels** area of the PivotTable. Then add a measure, named

Previous Quarter Sales, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]),
PREVIOUSQUARTER('DateTime'[DateKey]))
```

See Also

Time intelligence functions

Date and time functions

PREVIOUSMONTH

PREVIOUSDAY

PREVIOUSYEAR

Get Sample Data

PREVIOUSYEAR Function

Returns a table that contains a column of all dates from the previous year, given the last date in the **dates** column, in the current context.

Syntax

PREVIOUSYEAR(<dates>[,<year_end_date>])

Parameters

Term	Definition
dates	A column containing dates.
year_end_date	(optional) A literal string with a date that defines the year-end date. The default is December 31.

Return Value

A table containing a single column of date values.

Remarks

Note
To understand more about how context affects the results of formulas, see Context.

This function returns all dates from the previous year given the latest date in the input parameter. For example, if the latest date in the **dates** argument refers to the year 2009,

then this function returns all dates for the year of 2008, up to the specified **year_end_date**.

The **dates** argument can be any of the following:

- A reference to a date/time column,
- A table expression that returns a single column of date/time values,
- A Boolean expression that defines a single-column table of date/time values.

Note
Constraints on Boolean expressions are described in the topic, CALCULATE Function (DAX).

The **year_end_date** parameter is a string literal of a date, in the same locale as the locale of the client where the workbook was created. The year portion of the date is ignored.

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models,
see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

Description

The following sample formula creates a measure that calculates the previous year sales for the Internet sales.

To see how this works, create a PivotTable and add the fields, CalendarYear and CalendarQuarter, to the **Row Labels** area of the PivotTable. Then add a measure, named **Previous Year Sales**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]),
PREVIOUSYEAR('DateTime'[DateKey]))
```

See Also

Time intelligence functions

Date and time functions

PREVIOUSMONTH

PREVIOUSDAY

PREVIOUSQUARTER

SAMEPERIODLASTYEAR Function

Returns a table that contains a column of dates shifted one year back in time from the dates in the specified **dates** column, in the current context.

Syntax

SAMEPERIODLASTYEAR(<dates>)

Parameters

Term	Definition
dates	A column containing dates.

Property Value/Return Value

A single-column table of date values.

Remarks

Note
To understand more about how context affects the results of formulas, see Context.

The **dates** argument can be any of the following:

- A reference to a date/time column,
- A table expression that returns a single column of date/time values,
- A Boolean expression that defines a single-column table of date/time values.

Note
Constraints on Boolean expressions are described in the topic, CALCULATE Function (DAX).

The dates returned are the same as the dates returned by this equivalent formula:

```
DATEADD(dates, -1, year)
```

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

Description

The following sample formula creates a measure that calculates the previous year sales of the Reseller sales.

To see how this works, create a PivotTable and add the fields, CalendarYear to the **Row Labels** area of the PivotTable. Then add a measure, named **Previous Year Sales**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=CALCULATE(SUM(ResellerSales_USD[SalesAmount_USD]),
SAMEPERIODLASTYEAR(DateTime[DateKey]))
```

See Also

Time intelligence functions
Date and time functions
PREVIOUSYEAR
PARALLELPERIOD

STARTOFMONTH Function

Returns the first date of the month in the current context for the specified column of dates.

Syntax

STARTOFMONTH(<dates>)

Parameters

Term	Definition
dates	A column that contains dates.

Return Value

A table containing a single column and single row with a date value.

Remarks

Note
To understand more about how context affects the results of formulas, see Context.

The **dates** argument can be any of the following:

- A reference to a date/time column.
- A table expression that returns a single column of date/time values.
- A Boolean expression that defines a single-column table of date/time values.

Note
Constraints on Boolean expressions are described in the topic, CALCULATE Function (DAX).

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

Description

The following sample formula creates a measure that returns the start of the month, for the current context.

To see how this works, create a PivotTable and add the fields CalendarYear and MonthNumberOfYear to the **Row Labels** area of the PivotTable. Then add a measure, named **StartOfMonth**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=STARTOFMONTH(DateTime[DateKey])
```

See Also

Date and time functions

time intelligence functions

STARTOFYEAR

STARTOFQUARTER

STARTOFQUARTER Function

Returns the first date of the quarter in the current context for the specified column of dates.

Syntax

STARTOFQUARTER(<dates>)

Parameters

Term	Definition
dates	A column that contains dates.

Return Value

A table containing a single column and single row with a date value.

Remarks

Note
To understand more about how context affects the results of formulas, see Context.

The **dates** argument can be any of the following:

- A reference to a date/time column.
- A table expression that returns a single column of date/time values.
- A Boolean expression that defines a single-column table of date/time values.

Note
Constraints on Boolean expressions are described in the topic, CALCULATE Function (DAX).

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models,
see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

Description

The following sample formula creates a measure that returns the start of the quarter, for the current context.

To see how this works, create a PivotTable and add the fields CalendarYear and MonthNumberOfYear to the **Row Labels** area of the PivotTable. Then add a measure, named **StartOfQuarter**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=STARTOFQUARTER(DateTime[DateKey])
```

See Also

Date and time functions

time intelligence functions

STARTOFYEAR

STARTOFMONTH

STARTOFYEAR Function

Returns the first date of the quarter in the current context for the specified column of dates.

Syntax

STARTOFQUARTER(<dates>)

Parameters

Term	Definition
dates	A column that contains dates.

Return Value

A table containing a single column and single row with a date value.

Remarks

Note
To understand more about how context affects the results of formulas, see Context.

The **dates** argument can be any of the following:

- A reference to a date/time column.
- A table expression that returns a single column of date/time values.
- A Boolean expression that defines a single-column table of date/time values.

Note
Constraints on Boolean expressions are described in the topic, CALCULATE Function (DAX).

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

Description

The following sample formula creates a measure that returns the start of the quarter, for the current context.

To see how this works, create a PivotTable and add the fields CalendarYear and MonthNumberOfYear to the **Row Labels** area of the PivotTable. Then add a measure, named **StartOfQuarter**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=STARTOFQUARTER(DateTime[DateKey])
```

See Also

Date and time functions

time intelligence functions

STARTOFYEAR

STARTOFMONTH

TOTALMTD Function

Evaluates the value of the **expression** for the month to date, in the current context.

Syntax

TOTALMTD(<expression>,<dates>[,<filter>])

Parameters

Parameter	Definition
expression	An expression that returns a scalar value.
dates	A column that contains dates.
filter	(optional) An expression that specifies a filter to apply to the current context.

Return Value

A scalar value that represents the **expression** evaluated for the dates in the current month-to-date, given the dates in **dates**.

Remarks

Note
To understand more about how context affects the results of formulas, see Context.

The **dates** argument can be any of the following:

- A reference to a date/time column.
- A table expression that returns a single column of date/time values.
- A Boolean expression that defines a single-column table of date/time values.

Note
Constraints on Boolean expressions are described in the topic, CALCULATE Function (DAX).

Note
The **filter** expression has restrictions described in the topic, CALCULATE Function (DAX).

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that calculates the 'month running total' or 'month running sum' for the Internet sales.

To see how this works, create a PivotTable and add the fields, CalendarYear, MonthNumberOfYear and DayNumberOfMonth, to the **Row Labels** area of the PivotTable. Then add a measure, named **Month-to-date Total**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=TOTALMTD(SUM(InternetSales_USD[SalesAmount_USD]),DateTime[DateKey])
```

See Also

ALL
CALCULATE
TOTALYTD
TOTALQTD
Get Sample Data

TOTALQTD Function

Evaluates the value of the **expression** for the dates in the quarter to date, in the current context.

Syntax

TOTALQTD(<expression>,<dates>[,<filter>])

Parameters

Parameter	Definition
expression	An expression that returns a scalar value.
dates	A column that contains dates.
filter	(optional) An expression that specifies a filter to apply to the current context.

Return Value

A scalar value that represents the **expression** evaluated for all dates in the current quarter to date, given the dates in **dates**.

Remarks

Note
To understand more about how context affects the results of formulas, see Context.

The **dates** argument can be any of the following:

- A reference to a date/time column.
- A table expression that returns a single column of date/time values.
- A Boolean expression that defines a single-column table of date/time values.

Note

Constraints on Boolean expressions are described in the topic, CALCULATE Function (DAX).

Note

- The **filter** expression has restrictions described in the topic, CALCULATE Function (DAX).
- This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that calculates the 'quarter running total' or 'quarter running sum' for the Internet sales.

To see how this works, create a PivotTable and add the fields, CalendarYear, CalendarQuarter and MonthNumberOfYear, to the **Row Labels** area of the PivotTable. Then add a measure, named **Quarter-to-date Total**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=TOTALQTD(SUM(InternetSales_USD[SalesAmount_USD]),DateTime[DateKey])
```

See Also

ALL

CALCULATE

TOTALYTD

TOTALMTD

Get Sample Data

TOTALYTD Function

Evaluates the year-to-date value of the **expression** in the current context.

Syntax

TOTALYTD(<expression>,<dates>[,<filter>][,<year_end_date>])

Parameters

Parameter	Definition
expression	An expression that returns a scalar value.
dates	A column that contains dates.
filter	(optional) An expression that specifies a

Parameter	Definition
	filter to apply to the current context.
year_end_date	(optional) A literal string with a date that defines the year-end date. The default is December 31.

Return Value

A scalar value that represents the **expression** evaluated for the current year-to-date **dates**.

Remarks

Note
To understand more about how context affects the results of formulas, see Context.

The **dates** argument can be any of the following:

- A reference to a date/time column,
- A table expression that returns a single column of date/time values,
- A Boolean expression that defines a single-column table of date/time values.

Note
Constraints on Boolean expressions are described in the topic, CALCULATE Function (DAX).

Note
The **filter** expression has restrictions described in the topic, CALCULATE Function (DAX).

The **year_end_date** parameter is a string literal of a date, in the same locale as the locale of the client where the workbook was created. The year portion of the date is ignored.

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models, see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

The following sample formula creates a measure that calculates the 'year running total' or 'year running sum' for the Internet sales.

To see how this works, create a PivotTable and add the fields, CalendarYear, CalendarQuarter, and MonthNumberOfYear, to the **Row Labels** area of the PivotTable. Then add a measure, named **Year-to-date Total**, using the formula defined in the code section, to the **Values** area of the PivotTable.

Code

```
=TOTALYTD(SUM(InternetSales_USD[SalesAmount_USD]),DateTime[DateKey])
```

See Also

ALL

CALCULATE

DATESYTD Function (DAX)

TOTALMTD

TOTALQTD

Get Sample Data

DATE Function

Returns the specified date in **datetime** format.

Syntax

DATE(<year>, <month>, <day>)

Parameters

Term	Definition
year	A number representing the year. The value of the **year** argument can include one to four digits. The **year** argument is interpreted according to the date system used by your computer. Dates beginning with March 1, 1900 are supported. If you enter a number that has decimal places, the number is rounded. For values greater than 9999 or less than zero (negative values), the function returns a **#VALUE!** error. If the **year** value is between 0 and 1899, the value is added to 1900 to produce the final value. See the examples below. **Note** You should use four digits for the **year** argument whenever possible to prevent unwanted results. For

Term	Definition
	example, using 07 returns 1907 as the year value.
month	A number representing the month or a calculation according to the following rules: If **month** is a number from 1 to 12, then it represents a month of the year. 1 represents January, 2 represents February, and so on until 12 that represents December. If you enter an integer larger than 12, the following computation occurs: the date is calculated by adding the value of **month** to the **year**. For example, if you have DATE(2008, 18, 1), the function returns a datetime value equivalent to June 1st of 2009, because 18 months are added to the beginning of 2008 yielding a value of June 2009. See examples below. If you enter a negative integer, the following computation occurs: the date is calculated subtracting the value of **month** from **year**. For example, if you have DATE(2008, -6, 15), the function returns a datetime value equivalent to June 15th of 2007, because when 6 months are subtracted from the beginning of 2008 it yields a value of June 2007. See examples below.
day	A number representing the day or a calculation according to the following rules: If **day** is a number from 1 to the last day of the given month then it represents a day of the month. If you enter an integer larger than last day of the given month, the following computation occurs: the date is calculated by adding the value of **day** to **month**. For example, in the formula `DATE( 2008, 3, 32)`, the DATE function returns a **datetime**

Term	Definition
	value equivalent to April 1st of 2008, because 32 days are added to the beginning of March yielding a value of April 1st. If you enter a negative integer, the following computation occurs: the date is calculated subtracting the value of **day** from **month**. For example, in the formula `DATE( 2008, 5, -15)`, the DATE function returns a **datetime** value equivalent to April 15th of 2008, because 15 days are subtracted from the beginning of May 2008 yielding a value of April 2008. If **day** contains a decimal portion, it is rounded to the nearest integer value.

Return Value

Returns the specified date (**datetime)**.

Remarks

The DATE function takes the integers that are input as arguments, and generates the corresponding date. The DATE function is most useful in situations where the year, month, and day are supplied by formulas. For example, the underlying data might contain dates in a format that is not recognized as a date, such as YYYYMMDD. You can use the DATE function in conjunction with other functions to convert the dates to a number that can be recognized as a date.

In contrast to Microsoft Excel, which stores dates as a serial number, DAX date functions always return a **datetime** data type. However, you can use formatting to display dates as serial numbers if you want.

This DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see http://go.microsoft.com/fwlink/?LinkId=219171.

Example: Returning a Simple Date

Description

The following formula returns the date July 8, 2009:

Code

```
=DATE(2009,7,8)
```

Example: Years before 1899

Description

If the value that you enter for the **year** argument is between 0 (zero) and 1899 (inclusive), that value is added to 1900 to calculate the year. The following formula returns January 2, 1908: (1900+08).

Code

```
=DATE(08,1,2)
```

Example: Years before 1899

Description

If the value that you enter for the **year** argument is between 0 (zero) and 1899 (inclusive), that value is added to 1900 to calculate the year. The following formula returns January 2, 3700: (1900+1800).

Code

```
=DATE(1800,1,2)
```

Example: Years after 1899

Description

If **year** is between 1900 and 9999 (inclusive), that value is used as the year. The following formula returns January 2, 2008:

Code

```
=DATE(2008,1,2)
```

Example: Working with Months

Description

If **month** is greater than 12, **month** adds that number of months to the first month in the year specified. The following formula returns the date February 2, 2009:

Code

```
=DATE(2008,14,2)
```

Comment

If the **month** value is less than 1, the DATE function subtracts the magnitude of that number of months, plus 1, from the first month in the year specified. The following formula returns September 2, 2007:

```
=DATE(2008,-3,2)
```

Example: Working with Days

Description

If **day** is greater than the number of days in the month specified, **day** adds that number of days to the first day in the month. The following formula returns the date February 4, 2008:

Code

```
=DATE(2008,1,35)
```

Comment

If **day** is less than 1, **day** subtracts the magnitude that number of days, plus one, from the first day of the month specified. The following formula returns December 16, 2007:

```
=DATE(2008,1,-15)
```

See Also

Date and Time functions

DAY

TODAY

DATEVALUE Function

Converts a date in the form of text to a date in datetime format.

Syntax

DATEVALUE(date_text)

Parameters

Term	Definition
date_text	Text that represents a date.

Property Value/Return Value

A date in **datetime** format.

Remarks

The DATEVALUE function uses the locale and date/time settings of the client computer to understand the text value when performing the conversion. If the current date/time settings represent dates in the format of Month/Day/Year, then the string, "1/8/2009", would be converted to a **datetime** value equivalent to January 8th of 2009. However, if the current date and time settings represent dates in the format of Day/Month/Year, the same string would be converted as a **datetime** value equivalent to August 1st of 2009.

If the year portion of the **date_text** argument is omitted, the DATEVALUE function uses the current year from your computer's built-in clock. Time information in the **date_text** argument is ignored.

Example

Description

The following example returns a different **datetime** value depending on your computer's locale and settings for how dates and times are presented.

- In date/time settings where the day precedes the month, the example returns a **datetime** value corresponding to January 8th of 2009.
- In date/time settings where the month precedes the day, the example returns a **datetime** value corresponding to August 1st of 2009.

Code

```
=DATEVALUE("8/1/2009")
```

See Also

Date and Time functions

DAY Function

Returns the day of the month, a number from 1 to 31.

Syntax

DAY(<date>)

Parameters

Term	Definition
date	A date in **datetime** format, or a text representation of a date.

Return Value

An integer number indicating the day of the month.

Remarks

The DAY function takes as an argument the date of the day you are trying to find. Dates can be provided to the function by using another date function, by using an expression that returns a date, or by typing a date in a **datetime** format. You can also type a date in one of the accepted string formats for dates.

Values returned by the YEAR, MONTH and DAY functions will be Gregorian values regardless of the display format for the supplied date value. For example, if the display format of the supplied date is Hijri, the returned values for the YEAR, MONTH and DAY functions will be values associated with the equivalent Gregorian date.

When the date argument is a text representation of the date, the day function uses the locale and date/time settings of the client computer to understand the text value in order to perform the conversion. If the current date/time settings represent dates in the format of Month/Day/Year, then the string, "1/8/2009", is interpreted as a **datetime** value equivalent to January 8th of 2009, and the function returns 8. However, if the current date/time settings represent dates in the format of Day/Month/Year, the same

string would be interpreted as a **datetime** value equivalent to August 1st of 2009, and the function returns 1.

Example: Getting the Day from a Date Column

Description

The following formula returns the day from the date in the column, [Birthdate].

Code

```
=DAY([Birthdate])
```

Example: Getting the Day from a String Date

Description

The following formulas return the day, 4, using dates that have been supplied as strings in an accepted text format.

Code

```
=DAY("3-4-1007")
```

```
=DAY("March 4 2007")
```

Example: Using a Day Value as a Condition

Description

The following expression returns the day that each sales order was placed, and flags the row as a promotional sale item if the order was placed on the 10th of the month.

Code

```
=IF( DAY([SalesDate])=10,"promotion","")
```

See Also

Date and Time functions

TODAY

DATE

EDATE Function

Returns the date that is the indicated number of months before or after the start date. Use EDATE to calculate maturity dates or due dates that fall on the same day of the month as the date of issue.

Syntax

EDATE(<start_date>, <months>)

Parameters

Term	Definition
start_date	A date in **datetime** or **text** format that represents the start date.
months	An integer that represents the number of months before or after **start_date**.

Return Value

A date (**datetime**).

Remarks

In contrast to Microsoft Excel, which stores dates as sequential serial numbers, DAX works with dates in a **datetime** format. Dates stored in other formats are converted implicitly.

If **start_date** is not a valid date, EDATE returns an error. Make sure that the column reference or date that you supply as the first argument is a date.

If **months** is not an integer, it is truncated.

When the date argument is a text representation of the date, the EDATE function uses the locale and date time settings of the client computer to understand the text value in order to perform the conversion. If the current date time settings represent a date in the format of Month/Day/Year, then the following string "1/8/2009" is interpreted as a datetime value equivalent to January 8th of 2009. However, if the current date time settings represent a date in the format of Day/Month/Year, the same string would be interpreted as a datetime value equivalent to August 1st of 2009.

If the requested date is past the last day of the corresponding month, then the last day of the month is returned. For example, the following functions: EDATE("2009-01-29", 1), EDATE("2009-01-30", 1), EDATE("2009-01-31", 1) return February 28th of 2009; that corresponds to one month after the start date.

This DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see http://go.microsoft.com/fwlink/?LinkId=219171.

Example

Description

The following example returns the date three months after the order date, which is stored in the column [TransactionDate].

Code

```
=EDATE([TransactionDate],3)
```

See Also

EOMONTH

EOMONTH Function

Returns the date in **datetime** format of the last day of the month, before or after a specified number of months. Use EOMONTH to calculate maturity dates or due dates that fall on the last day of the month.

Syntax

EOMONTH(<start_date>, <months>)

Parameters

Term	Definition
start_date	The start date in **datetime** format, or in an accepted text representation of a date.
months	A number representing the number of months before or after the **start_date**.  **Note** If you enter a number that is not an integer, the number is rounded up or down to the nearest integer.

Return Value

A date (**datetime**).

Remarks

In contrast to Microsoft Excel, which stores dates as sequential serial numbers, DAX works with dates in a **datetime** format. The EOMONTH function can accept dates in other formats, with the following restrictions:

If **start_date** is not a valid date, EOMONTH returns an error.

If **start_date** is a numeric value that is not in a **datetime** format, EOMONTH will convert the number to a date. To avoid unexpected results, convert the number to a **datetime** format before using the EOMONTH function.

If **start_date** plus months yields an invalid date, EOMONTH returns an error. Dates before March 1st of 1900 and after December 31st of 9999 are invalid.

When the date argument is a text representation of the date, the EDATE function uses the locale and date time settings, of the client computer, to understand the text value in order to perform the conversion. If current date time settings represent a date in the

format of Month/Day/Year, then the following string "1/8/2009" is interpreted as a datetime value equivalent to January 8th of 2009. However, if the current date time settings represent a date in the format of Day/Month/Year, the same string would be interpreted as a datetime value equivalent to August 1st of 2009.

This DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see http://go.microsoft.com/fwlink/?LinkId=219171.

Example

Description

The following expression returns May 31, 2008, because the **months** argument is rounded to 2.

Code

```
=EOMONTH("March 3, 2008",1.5)
```

See Also

EDATE

Date and Time functions

HOUR Function

Returns the hour as a number from 0 (12:00 A.M.) to 23 (11:00 P.M.).

Syntax

HOUR(<datetime>)

Parameters

Term	Definition
datetime	A **datetime** value, such as 16:48:00 or 4:48 PM.

Return Value

An integer number from 0 to 23.

Remarks

The HOUR function takes as argument the time that contains the hour you want to find. You can supply the time by using a date/time function, an expression that returns a **datetime**, or by typing the value directly in one of the accepted time formats. Times can also be entered as any accepted text representation of a time.

When the **datetime** argument is a text representation of the date and time, the function uses the locale and date/time settings of the client computer to understand the text value in order to perform the conversion. Most locales use the colon (:) as the time separator and any input text using colons as time separators will parse correctly. Review your locale settings to understand your results.

Example

Description

The following example returns the hour from the **TransactionTime** column of a table named **Orders**.

Code

```
=HOUR('Orders'[TransactionTime])
```

Example

Description

The following example returns 15, meaning the hour corresponding to 3 PM in a 24-hour clock. The text value is automatically parsed and converted to a date/time value.

Code

```
=HOUR("March 3, 2008 3:00 PM")
```

See Also

Date and Time functions

MINUTE

YEAR

SECOND

MINUTE Function

Returns the minute as a number from 0 to 59, given a date and time value.

Syntax

MINUTE(<datetime>)

Parameters

Term	Definition
datetime	A **datetime** value or text in an accepted time format, such as 16:48:00 or 4:48 PM.

Return Value

An integer number from 0 to 59.

Remarks

In contrast to Microsoft Excel, which stores dates and times in a serial numeric format, DAX uses a **datetime** data type for dates and times. You can provide the **datetime** value to the MINUTE function by referencing a column that stores dates and times, by using a date/time function, or by using an expression that returns a date and time.

When the **datetime** argument is a text representation of the date and time, the function uses the locale and date/time settings of the client computer to understand the text value in order to perform the conversion. Most locales use the colon (:) as the time separator and any input text using colons as time separators will parse correctly. Verify your locale settings to understand your results.

Example

Description

The following example returns the minute from the value stored in the **TransactionTime** column of the **Orders** table.

Code

```
=MINUTE(Orders[TransactionTime])
```

Example

Description

The following example returns 45, which is the number of minutes in the time 1:45 PM.

Code

```
=MINUTE("March 23, 2008 1:45 PM")
```

See Also

Date and Time functions

HOUR

YEAR

SECOND

MONTH Function

Returns the month as a number from 1 (January) to 12 (December).

Syntax

MONTH(<datetime>)

Parameters

Term	Definition
date	A date in **datetime** or text format.

Return Value

An integer number from 1 to 12.

Remarks

In contrast to Microsoft Excel, which stores dates as serial numbers, DAX uses a **datetime** format when working with dates. You can enter the date used as argument to the MONTH function by typing an accepted **datetime** format, by providing a reference to a column that contains dates, or by using an expression that returns a date.

Values returned by the YEAR, MONTH and DAY functions will be Gregorian values regardless of the display format for the supplied date value. For example, if the display format of the supplied date is Hijri, the returned values for the YEAR, MONTH and DAY functions will be values associated with the equivalent Gregorian date.

When the date argument is a text representation of the date, the function uses the locale and date time settings of the client computer to understand the text value in order to perform the conversion. If the current date time settings represent a date in the format of Month/Day/Year, then the following string "1/8/2009" is interpreted as a datetime value equivalent to January 8th of 2009, and the function yields a result of 1. However, if the current date time settings represent a date in the format of Day/Month/Year, then the same string would be interpreted as a datetime value equivalent to August 1st of 2009, and the function yields a result of 8.

If the text representation of the date cannot be correctly converted to a datetime value, the function returns an error.

Example

Description

The following expression returns 3, which is the integer corresponding to March, the month in the **date** argument.

Code

```
=MONTH("March 3, 2008 3:45 PM")
```

Example

Description

The following expression returns the month from the date in the **TransactionDate** column of the **Orders** table.

Code

```
=MONTH(Orders[TransactionDate])
```

See Also

Date and Time functions

HOUR

MINUTE

YEAR

SECOND

NOW Function

Returns the current date and time in **datetime** format.

The NOW function is useful when you need to display the current date and time on a worksheet or calculate a value based on the current date and time, and have that value updated each time you open the worksheet.

Syntax

NOW()

Return Value

A date (**datetime)**.

Remarks

In contrast to Microsoft Excel, which stores dates and times as serial numbers, DAX uses a **datetime** format to work with dates. Dates that are not in this format are implicitly converted when you use dates and times in a formula.

The result of the NOW function changes only when the column that contains the formula is refreshed. It is not updated continuously.

The TODAY function returns the same date but is not precise with regard to time; the time returned is always 12:00:00 AM and only the date is updated.

Example

Description

The following example returns the current date and time plus 3.5 days:

Code

```
=NOW()+3.5
```

See Also

Date and Time functions

TODAY

SECOND Function

Returns the seconds of a time value, as a number from 0 to 59.

Syntax

SECOND(<time>)

Parameters

Term	Definition
time	A time in **datetime** format, such as 16:48:23 or 4:48:47 PM.

Return Value

An integer number from 0 to 59.

Remarks

In contrast to Microsoft Excel, which stores dates and times as serial numbers, DAX uses a **datetime** format when working with dates and times. If the source data is not in this format, DAX implicitly converts the data. You can use formatting to display the dates and times as a serial number of you need to.

The date/time value that you supply as an argument to the SECOND function can be entered as a text string within quotation marks (for example, "6:45 PM"). You can also provide a time value as the result of another expression, or as a reference to a column that contains times.

If you provide a numeric value of another data type, such as 13.60, the value is interpreted as a serial number and is represented as a **datetime** data type before extracting the value for seconds. To make it easier to understand your results, you might want to represent such numbers as dates before using them in the SECOND function. For example, if you use SECOND with a column that contains a numeric value such as, **25.56**, the formula returns 24. That is because, when formatted as a date, the value 25.56 is equivalent to January 25, 1900, 1:26:24 PM.

When the **time** argument is a text representation of a date and time, the function uses the locale and date/time settings of the client computer to understand the text value in order to perform the conversion. Most locales use the colon (:) as the time separator and any input text using colons as time separators will parse correctly. Review your locale settings to understand your results.

Example

Description

The following formula returns the number of seconds in the time contained in the **TransactionTime** column of a table named **Orders**.

Code

```
=SECOND('Orders'[TransactionTime])
```

Example

Description

The following formula returns 3, which is the number of seconds in the time represented by the value, **March 3, 2008 12:00:03**.

Code

```
=SECOND("March 3, 2008 12:00:03")
```

See Also

Date and Time functions
HOUR
MINUTE
YEAR

TIME Function

Converts hours, minutes, and seconds given as numbers to a time in **datetime** format.

Syntax

TIME(hour, minute, second)

Parameters

Term	Definition
hour	A number from 0 to 23 representing the hour. Any value greater than 23 will be divided by 24 and the remainder will be treated as the hour value.
minute	A number from 0 to 59 representing the minute. Any value greater than 59 will be converted to hours and minutes.
second	A number from 0 to 59 representing the second. Any value greater than 59 will be converted to hours, minutes, and seconds.

Return Value

A time (**datetime**).

Remarks

In contrast to Microsoft Excel, which stores dates and times as serial numbers, DAX works with date and time values in a **datetime** format. Numbers in other formats are implicitly

converted when you use a date/time value in a DAX function. If you need to use serial numbers, you can use formatting to change the way that the numbers are displayed.

Time values are a portion of a date value, and in the serial number system are represented by a decimal number. Therefore, the **datetime** value 12:00 PM is equivalent to 0.5, because it is half of a day.

You can supply the arguments to the TIME function as values that you type directly, as the result of another expression, or by a reference to a column that contains a numeric value. The following restrictions apply:

- Any value for **hours** that is greater than 23 will be divided by 24 and the remainder will be treated as the hour value.
- Any value for **minutes** that is greater than 59 will be converted to hours and minutes.
- Any value for **seconds** that is greater than 59 will be converted to hours, minutes, and seconds.
- For minutes or seconds, a value greater than 24 hours will be divided by 24 and the reminder will be treated as the hour value. A value in excess of 24 hours does not alter the date portion.

To improve readability of the time values returned by this function, we recommend that you format the column or PivotTable cell that contains the results of the formula by using one of the time formats provided by Microsoft Excel.

This DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see http://go.microsoft.com/fwlink/?LinkId=219171.

Example

Description

The following examples both return the time, 3:00 AM:

Code

```
=TIME(27,0,0)
```

```
=TIME(3,0,0)
```

Example

Description

The following examples both return the time, 12:30 PM:

Code

```
=TIME(0,750,0)
```

```
=TIME(12,30,0)
```

Example

Description

The following example creates a time based on the values in the columns, `intHours`, `intMinutes`, `intSeconds`:

Code

```
=TIME([intHours],[intMinutes],[intSeconds])
```

See Also

DATE

Date and Time functions

TIMEVALUE Function

Converts a time in text format to a time in datetime format.

Syntax

TIMEVALUE(time_text)

Parameters

Term	Definition
time_text	A text string that that represents a certain time of the day. Any date information included in the **time_text** argument is ignored.

Return Value

A date (**datetime**).

Remarks

Time values are a portion of a date value and represented by a decimal number. For example, 12:00 PM is represented as 0.5 because it is half of a day.

When the **time_text** argument is a text representation of the date and time, the function uses the locale and date/time settings of the client computer to understand the text value in order to perform the conversion. Most locales use the colon (:) as the time separator, and any input text using colons as time separators will parse correctly. Review your locale settings to understand your results.

Example

Description

Code

```
=TIMEVALUE("20:45:30")
```

Comments

See Also

Date and Time functions

TODAY Function

Returns the current date.

Syntax

TODAY()

Return Value

A date (**datetime**).

Remarks

The TODAY function is useful when you need to have the current date displayed on a worksheet, regardless of when you open the workbook. It is also useful for calculating intervals.

Note

If the TODAY function does not update the date when you expect it to, you might need to change the settings that control when the column or workbook is refreshed. For more information, see Data Refresh.

The NOW function is similar but returns the exact time, whereas TODAY returns the time value 12:00:00 PM for all dates.

Example

Description

If you know that someone was born in 1963, you might use the following formula to find that person's age as of this year's birthday:

Code

```
=YEAR(TODAY())-1963
```

Comments

This formula uses the TODAY function as an argument for the YEAR function to obtain the current year, and then subtracts 1963, returning the person's age.

See Also

Date and Time functions

NOW

WEEKDAY Function

Returns a number from 1 to 7 identifying the day of the week of a date. By default the day ranges from 1 (Sunday) to 7 (Saturday).

Syntax

WEEKDAY(<date>, <return_type>)

Parameters

<table>
<tr><th>Term</th><th>Definition</th></tr>
<tr><td>date</td><td>A date in datetime format.
Dates should be entered by using the DATE function, by using expressions that result in a date, or as the result of other formulas.</td></tr>
<tr><td>return_type</td><td>A number that determines the return value:

<table>
<tr><td>Return type</td><td>Week begins</td></tr>
<tr><td>1</td><td>Week begins on Sunday (1) and ends on Saturday (7).</td></tr>
<tr><td>2</td><td>Week begins on Monday (1) and ends on Sunday (7).</td></tr>
<tr><td>3</td><td>Week begins on Monday (0) and ends on Sunday (6).</td></tr>
</table>
</td></tr>
</table>

Return Value

An integer number from 1 to 7.

Remarks

In contrast to Microsoft Excel, which stores dates as serial numbers, DAX works with dates and times in a **datetime** format. If you need to display dates as serial numbers, you can use the formatting options in Excel.

You can also type dates in an accepted text representation of a date, but to avoid unexpected results, it is best to convert the text date to a **datetime** format first.

When the date argument is a text representation of the date, the function uses the locale and date/time settings of the client computer to understand the text value in order to perform the conversion. If the current date/time settings represent dates in the format of Month/Day/Year, then the string, "1/8/2009", is interpreted as a **datetime** value equivalent to January 8th of 2009. However, if the current date/time settings represent

dates in the format of Day/Month/Year, then the same string would be interpreted as a **datetime** value equivalent to August 1st of 2009.

Example

Description

The following example gets the date from the [HireDate] column, adds 1, and displays the weekday corresponding to that date. Because the **return_type** argument has been omitted, the default format is used, in which 1 is Sunday and 7 is Saturday. If the result is 4, the day would be Wednesday.

Code

```
=WEEKDAY([HireDate]+1)
```

See Also

Date and Time functions

WEEKNUM

YEARFRAC

WEEKNUM Function

Returns the week number for the given date and year according to the **return_type** value. The week number indicates where the week falls numerically within a year.

Syntax

WEEKNUM(<date>, <return_type>)

Parameters

<table>
<tr><th>Term</th><th>Definition</th></tr>
<tr><td>date</td><td>The date in datetime format.</td></tr>
<tr><td>return_type</td><td>A number that determines the return value: use 1 when the week begins on Sunday; use 2 when the week begins on Monday. The default is 1.

<table>
<tr><td>Return type</td><td>Week begins</td></tr>
<tr><td>1</td><td>Week begins on Sunday. Weekdays are numbered 1 through 7.</td></tr>
</table>
</td></tr>
</table>

Term	Definition	
	2	Week begins on Monday. Weekdays are numbered 1 through 7.

Return Value

An integer number.

Remarks

In contrast to Microsoft Excel, which stores dates as serial numbers, DAX uses a **datetime** data type to work with dates and times. If the source data is in a different format, DAX implicitly converts the data to **datetime** to perform calculations.

By default, the WEEKNUM function uses a calendar convention in which the week containing January 1 is considered to be the first week of the year. However, the ISO 8601 calendar standard, widely used in Europe, defines the first week as the one with the majority of days (four or more) falling in the new year. This means that for years in which there are three days or less in the first week of January, the WEEKNUM function returns week numbers that are different from the ISO 8601 definition.

Example

Description

The following example returns the week number of the date February 14, 2010.

Code

```
=WEEKNUM("Feb 14, 2010", 2)
```

Example

Description

The following example returns the week number of the date stored in the column, **HireDate**, from the table, **Employees**.

Code

```
=WEEKNUM('Employees'[HireDate])
```

See Also

Date and Time functions

YEARFRAC

WEEKDAY

YEAR Function

Returns the year of a date as a four digit integer in the range 1900-9999.

Syntax

YEAR(<date>)

Parameters

Term	Definition
date	A date in **datetime** or text format, containing the year you want to find.

Return Value

An integer in the range 1900-9999.

Remarks

In contrast to Microsoft Excel, which stores dates as serial numbers, DAX uses a **datetime** data type to work with dates and times.

Dates should be entered by using the DATE function, or as results of other formulas or functions. You can also enter dates in accepted text representations of a date, such as March 3, 2007, or Mar-3-2003.

Values returned by the YEAR, MONTH, and DAY functions will be Gregorian values regardless of the display format for the supplied date value. For example, if the display format of the supplied date uses the Hijri calendar, the returned values for the YEAR, MONTH, and DAY functions will be values associated with the equivalent Gregorian date.

When the date argument is a text representation of the date, the function uses the locale and date time settings of the client computer to understand the text value in order to perform the conversion. Errors may arise if the format of strings is incompatible with the current locale settings. For example, if your locale defines dates to be formatted as month/day/year, and the date is provided as day/month/year, then 25/1/2009 will not be interpreted as January 25th of 2009 but as an invalid date.

Example

Description

The following example returns 2007.

Code

```
=YEAR("March 2007")
```

Example: Date as Result of Expression

Description

The following example returns the year for today's date.

Code

```
=YEAR(TODAY())
```

See Also

Date and Time functions

HOUR

MINUTE

YEAR

SECOND

YEARFRAC Function

Calculates the fraction of the year represented by the number of whole days between two dates. Use the YEARFRAC worksheet function to identify the proportion of a whole year's benefits or obligations to assign to a specific term.

Syntax

YEARFRAC(<start_date>, <end_date>, <basis>)

Parameters

Term	Definition
start_date	The start date in **datetime** format.
end_date	The end date in **datetime** format.
basis	(Optional) The type of day count basis to use. All arguments are truncated to integers. Basis: Description 0: US (NASD) 30/360 1: Actual/actual 2: Actual/360 3: Actual/365 4: European 30/360

Term	Definition

Return Value

A decimal number. The internal data type is a signed IEEE 64-bit (8-byte) double-precision floating-point number.

Remarks

In contrast to Microsoft Excel, which stores dates as serial numbers, DAX uses a **datetime** format to work with dates and times. If you need to view dates as serial numbers, you can use the formatting options in Excel.

If **start_date** or **end_date** are not valid dates, YEARFRAC returns an error.

If **basis** < 0 or if **basis** > 4, YEARFRAC returns an error.

Example

Description

The following example returns the fraction of a year represented by the difference between the dates in the two columns, `TransactionDate` and `ShippingDate`:

Code

```
=YEARFRAC(Orders[TransactionDate],Orders[ShippingDate])
```

Example

Description

The following example returns the fraction of a year represented by the difference between the dates, January 1 and March 1:

Code

```
=YEARFRAC("Jan 1 2007","Mar 1 2007")
```

Comments

Use four-digit years whenever possible, to avoid getting unexpected results. When the year is truncated, the current year is assumed. When the date is or omitted, the first date of the month is assumed.

The second argument, **basis**, has also been omitted. Therefore, the year fraction is calculated according to the US (NASD) 30/360 standard.

See Also

Date and Time functions

WEEKNUM

YEARFRAC

WEEKDAY

Filter Functions

The filter and value functions in DAX are some of the most complex and powerful, and differ greatly from Excel functions. The lookup functions work by using tables and relationships, like a database. The filtering functions let you manipulate data context to create dynamic calculations. For a discussion of context, see Context in DAX Formulas.

In this Section

ALL Function
ALLEXCEPT Function
ALLNOBLANKROW Function
ALLSELECTED Function
CALCULATE Function
CALCULATETABLE Function
DISTINCT Function
EARLIER Function
EARLIEST Function
FILTER Function
FILTERS Function
HASONEFILTER Function
HASONEVALUE Function
ISCROSSFILTERED Function
ISFILTERED Function
RELATED Function
RELATEDTABLE Function
VALUES Function

Reference

Using DAX
Basic DAX Syntax

Related Sections

Date and Time Functions (DAX)
Aggregation Functions (DAX)
Logical Functions (DAX)
Filter and Value Functions (DAX)
Math and Trigonometric Functions (DAX)

See Also

ALL Function

Returns all the rows in a table, or all the values in a column, ignoring any filters that might have been applied. This function is useful for clearing filters and creating calculations on all the rows in a table.

Syntax

ALL({<table> | <column>[, <column>[, <column>[,...]]]})

Parameters

Term	Definition
table	The table that you want to clear filters on.
column	The column that you want to clear filters on.

The argument to the ALL function must be either a reference to a base table or a reference to a base column. You cannot use table expressions or column expressions with the ALL function.

Return Value

The table or column with filters removed.

Remarks

This function is not used by itself, but serves as an intermediate function that can be used to change the set of results over which some other calculation is performed.

As described in the following table, you can use the ALL and ALLEXCEPT functions in different scenarios.

Function and Usage	Description
ALL(Table)	Removes all filters from the specified table. In effect, ALL(Table) returns all of the values in the table, removing any filters from the context that otherwise might have been applied. This function is useful when you are working with many levels of grouping, and want to create a calculation that creates a

Function and Usage	Description
	ratio of an aggregated value to the total value. The first example demonstrates this scenario.
ALL (Column[, Column[, ...]])	Removes all filters from the specified columns in the table; all other filters on other columns in the table still apply. All column arguments must come from the same table. The ALL(Column) variant is useful when you want to remove the context filters for one or more specific columns and to keep all other context filters. The second and third examples demonstrate this scenario.
ALLEXCEPT(Table, Column1 [,Column2]...)	Removes all context filters in the table except filters that are applied to the specified columns. This is a convenient shortcut for situations in which you want to remove the filters on many, but not all, columns in a table.

Example: Calculate Ratio of Category Sales to Total Sales

Description

Assume that you want to find the amount of sales for the current cell, in your PivotTable, divided by the total sales for all resellers. To ensure that the denominator is the same regardless of how the PivotTable user might be filtering or grouping the data, you define a formula that uses ALL to create the correct grand total.

The following table shows the results when a new measure, **All Reseller Sales Ratio**, is created using the formula shown in the code section. To see how this works, add the field, CalendarYear, to the **Row Labels** area of the PivotTable, and add the field, ProductCategoryName, to the **Column Labels** area. Then, drag the measure, **All Reseller Sales Ratio**, to the **Values** area of the Pivot Table. To view the results as percentages, use the formatting features of Excel to apply a percentage number formatting to the cells that contains the measure.

All Reseller Sales	Column Labels				
Row Labels	Accessories	Bikes	Clothing	Components	Grand Total
2005	0.02%	9.10%	0.04%	0.75%	9.91%
2006	0.11%	24.71%	0.60%	4.48%	29.90%
2007	0.36%	31.71%	1.07%	6.79%	39.93%
2008	0.20%	16.95%	0.48%	2.63%	20.26%
Grand Total	0.70%	82.47%	2.18%	14.65%	100.00%

Code

```
=SUMX(ResellerSales_USD,
ResellerSales_USD[SalesAmount_USD])/SUMX(ALL(ResellerSales_USD),
ResellerSales_USD[SalesAmount_USD])
```

Comments

The formula is constructed as follows:

1. The numerator, `SUMX(ResellerSales_USD, ResellerSales_USD[SalesAmount_USD])`, is the sum of the values in ResellerSales_USD[SalesAmount_USD] for the current cell in the PivotTable, with context filters applied on CalendarYear and ProductCategoryName.
2. For the denominator, you start by specifying a table, ResellerSales_USD, and use the ALL function to remove all context filters on the table.
3. You then use the SUMX function to sum the values in the ResellerSales_USD[SalesAmount_USD] column. In other words, you get the sum of ResellerSales_USD[SalesAmount_USD] for all resellers sales.

For more information about creating measures, see Create a Measure.

Note
The above example uses the tables, ResellerSales_USD, DateTime, and ProductCategory from the DAX sample workbook. For more information about samples, see Get Sample Data .

Example: Calculate Ratio of Product Sales to Total Sales Through Current Year

Description

Assume that you want to create a table showing the percentage of sales compared over the years for each product category (ProductCategoryName). To obtain the percentage for each year over each value of ProductCategoryName, you need to divide the sum of

sales for that particular year and product category by the sum of sales for the same product category over all years. In other words, you want to keep the filter on ProductCategoryName but remove the filter on the year when calculating the denominator of the percentage.

The following table shows the results when a new measure, **Reseller Sales Year**, is created using the formula shown in the code section. To see how this works, add the field, CalendarYear, to the **Row Labels** area of the PivotTable, and add the field, ProductCategoryName, to the **Column Labels** area. To view the results as percentages, use Excel's formatting features to apply a percentage number format to the cells containing the measure, **Reseller Sales Year**.

Reseller Sales Year	**Column Labels**				
Row Labels	Accessories	Bikes	Clothing	Components	Grand Total
2005	3.48%	11.03%	1.91%	5.12%	9.91%
2006	16.21%	29.96%	27.29%	30.59%	29.90%
2007	51.62%	38.45%	48.86%	46.36%	39.93%
2008	28.69%	20.56%	21.95%	17.92%	20.26%
Grand Total	100.00%	100.00%	100.00%	100.00%	100.00%

Code

```
=SUMX(ResellerSales_USD, ResellerSales_USD[SalesAmount_USD])/CALCULATE(
SUM( ResellerSales_USD[SalesAmount_USD]), ALL(DateTime[CalendarYear]))
```

Comments

The formula is constructed as follows:

1. The numerator, `SUMX(ResellerSales_USD, ResellerSales_USD[SalesAmount_USD])`, is the sum of the values in ResellerSales_USD[SalesAmount_USD] for the current cell in the pivot table, with context filters applied on the columns CalendarYear and ProductCategoryName.
2. For the denominator, you remove the existing filter on CalendarYear by using the ALL(Column) function. This calculates the sum over the remaining rows on the ResellerSales_USD table, after applying the existing context filters from the column labels. The net effect is that for the denominator the sum is calculated over the selected ProductCategoryName (the implied context filter) and for all values in Year.

For more information about creating measures, see Create a Measure.

Note

This example uses the tables, ResellerSales_USD, DateTime, and ProductCategory from the DAX sample workbook. For more information about samples, see Get Sample Data .

Example: Calculate Contribution of Product Categories to Total Sales Per Year

Description

Assume that you want to create a table that shows the percentage of sales for each product category, on a year-by-year basis. To obtain the percentage for each product category in a particular year, you need to calculate the sum of sales for that particular product category (ProductCategoryName) in year , and then divide the resulting value by the sum of sales for the year over all product categories. In other words, you want to keep the filter on year but remove the filter on ProductCategoryName when calculating the denominator of the percentage.

The following table shows the results when a new measure, **Reseller Sales CategoryName**, is created using the formula shown in the code section. To see how this works, add the field, CalendarYear to the **Row Labels** area of the PivotTable, and add the field, ProductCategoryName, to the **Column Labels** area. Then add the new measure to the **Values** area of the PivotTable. To view the results as percentages, use Excel's formatting features to apply a percentage number format to the cells that contain the new measure, **Reseller Sales CategoryName**.

Reseller Sales CategoryName	**Column Labels**				
Row Labels	Accessories	Bikes	Clothing	Components	Grand Total
2005	0.25%	91.76%	0.42%	7.57%	100.00%
2006	0.38%	82.64%	1.99%	14.99%	100.00%
2007	0.90%	79.42%	2.67%	17.01%	100.00%
2008	0.99%	83.69%	2.37%	12.96%	100.00%
Grand Total	0.70%	82.47%	2.18%	14.65%	100.00%

Code

```
=SUMX(ResellerSales_USD, ResellerSales_USD[SalesAmount_USD])/CALCULATE(
SUM( ResellerSales_USD[SalesAmount_USD]),
ALL(ProductCategory[ProductCategoryName]))
```

Comments

The formula is constructed as follows:

1. The numerator, `SUMX(ResellerSales_USD, ResellerSales_USD[SalesAmount_USD])`, is the sum of the values in ResellerSales_USD[SalesAmount_USD] for the current cell in the PivotTable, with context filters applied on the fields, CalendarYear and ProductCategoryName.
2. For the denominator, you use the function, ALL(Column), to remove the filter on ProductCategoryName and calculate the sum over the remaining rows on the ResellerSales_USD table, after applying the existing context filters from the row labels. The net effect is that, for the denominator, the sum is calculated over the selected Year (the implied context filter) and for all values of ProductCategoryName.

For more information about creating measures, see Create a Measure.

Note
This example uses the tables, ResellerSales_USD, DateTime, and ProductCategory from the DAX sample workbook. For more information about samples, see Get Sample Data.

See Also

Filter and value functions

ALL

ALLEXCEPT

FILTER

ALLEXCEPT Function

Removes all context filters in the table except filters that have been applied to the specified columns.

Syntax

ALLEXCEPT(<table>,<column>[,<column>[,...]])

Parameters

Term	Definition
table	The table over which all context filters are removed, except filters on those columns that are specified in subsequent arguments.
column	The column for which context filters must be preserved.

The first argument to the ALLEXCEPT function must be a reference to a base table; all subsequent arguments must be references to base columns. You cannot use table expressions or column expressions with the ALLEXCEPT function.

Return Value

A table with all filters removed except for the filters on the specified columns.

Remarks

This function is not used by itself, but serves as an intermediate function that can be used to change the set of results over which some other calculation is performed.

As described in the following table, you can use the ALL and ALLEXCEPT functions in different scenarios.

Function and Usage	Description
ALL(Table)	Removes all filters from the specified table. In effect, ALL(Table) returns all of the values in the table, removing any filters from the context that otherwise might have been applied. This function is useful when you are working with many levels of grouping, and want to create a calculation that creates a ratio of an aggregated value to the total value.
ALL (Column[, Column[, ...]])	Removes all filters from the specified columns in the table; all other filters on other columns in the table still apply. All column arguments must come from the same table. The ALL(Column) variant is useful when you want to remove the context filters for one or more specific columns and to keep all other context filters.
ALLEXCEPT(Table, Column1 [,Column2]...)	Removes all context filters in the table except filters that are applied to the specified columns. This is a convenient shortcut for situations in which you want to remove the filters on many, but not all, columns in a table.

Example

Description

The following example presents a formula that you can use in a measure. For more information about how to create a measure, see Create a Measure.

The formula sums SalesAmount_USD and uses the ALLEXCEPT function to remove any context filters on the DateTime table except if the filter has been applied to the CalendarYear column.

Note

The above example uses the tables, ResellerSales_USD and DateTime from the DAX sample workbook. For more information about samples, see Get Sample Data .

Code

```
=CALCULATE(SUM(ResellerSales_USD[SalesAmount_USD]), ALLEXCEPT(DateTime,
DateTime[CalendarYear]))
```

Comments

Because the formula uses ALLEXCEPT, whenever any column but CalendarYear from the table DateTime is used to slice the PivotTable, the formula will remove any slicer filters, providing a value equal to the sum of SalesAmount_USD for the column label value, as shown in Table 1.

However, if the column CalendarYear is used to slice the PivotTable, the results are different. Because CalendarYear is specified as the argument to ALLEXCEPT, when the data is sliced on the year, a filter will be applied on years at the row level, as shown in Table 2. The user is encouraged to compare these tables to understand the behavior of ALLEXCEPT().

See Also

Filter and value functions

ALL

FILTER

ALLNOBLANKROW Function

From the parent table of a relationship, returns all rows but the blank row, or all distinct values of a column but the blank row, and disregards any context filters that might exist.

Syntax

ALLNOBLANKROW(<table>|<column>)

Parameters

Term	Definition
table	The table over which all context filters are removed.
column	The column over which all context filters are removed.

Only one parameter must be passed; the parameter is either a table or a column.

Return Value

A table, when the passed parameter was a table, or a column of values, when the passed parameter was a column.

Remarks

The ALLNOBLANKROW function only filters the blank row that a parent table, in a relationship, will show when there are one or more rows in the child table that have non-matching values to the parent column. See the example below for a thorough explanation.

The following table summarizes the variations of ALL that are provided in DAX, and their differences:

Function and Usage	Description
ALL(Column)	Removes all filters from the specified column in the table; all other filters in the table, over other columns, still apply.
ALL(Table)	Removes all filters from the specified table.
ALLEXCEPT(Table,Col1,Col2...)	Overrides all context filters in the table except over the specified columns.
ALLNOBLANK(table|column)	From the parent table of a relationship, returns all rows but the blank row, or all distinct values of a column but the blank row, and disregards any context filters that might exist

For a general description of how the ALL function works, together with step-by-step examples that use ALL(Table) and ALL(Column), see ALL Function (DAX).

Example

Description

In the sample data, the ResellerSales_USD table contains one row that has no values and therefore cannot be related to any of the parent tables in the relationships within the workbook. You will use this table in a PivotTable so that you can see the blank row behavior and how to handle counts on unrelated data.

Step 1: Verify the unrelated data

Open the **PowerPivot window**, then select the ResellerSales_USD table. In the ProductKey column, filter for blank values. One row will remain. In that row, all column values should be blank except for SalesOrderLineNumber.

Step 2: Create a PivotTable

Create a new PivotTable, then drag the column, datetime.[Calendar Year], to the Row Labels pane. The following table shows the expected results:

Row Labels
2005
2006
2007
2008
Grand Total

Note the blank label between **2008** and **Grand Total**. This blank label represents the Unknown member, which is a special group that is created to account for any values in the child table that have no matching value in the parent table, in this example the datetime.[Calendar Year] column.

When you see this blank label in the PivotTable, you know that in some of the tables that are related to the column, datetime.[Calendar Year], there are either blank values or non-matching values. The parent table is the one that shows the blank label, but the rows that do not match are in one or more of the child tables.

The rows that get added to this blank label group are either values that do not match any value in the parent table-- for example, a date that does not exist in the datetime table-- or null values, meaning no value for date at all. In this example we have placed a blank value in all columns of the child sales table. Having more values in the parent table than in the children tables does not cause a problem.

Step 3: Count rows using ALL and ALLNONBLANK

Add the following two measures to the datetime table, to count the table rows: **Countrows ALLNOBLANK of datetime**, **Countrows ALL of datetime**. The formulas that you can use to define these measures are given in the code section following.

On a blank PivotTable add datetime.[Calendar Year] column to the row labels, and then add the newly created measures. The results should look like the following table:

Row Labels	Countrows ALLNOBLANK of datetime	Countrows ALL of datetime
2005	1280	1281
2006	1280	1281
2007	1280	1281
2008	1280	1281
	1280	1281
Grand Total	1280	1281

The results show a difference of 1 row in the table rows count. However, if you open the **PowerPivot window** and select the datetime table, you cannot find any blank row in the table because the special blank row mentioned here is the Unknown member.

Step 4: Verify that the count is accurate

In order to prove that the ALLNOBLANKROW does not count any truly blank rows, and only handles the special blank row on the parent table only, add the following two measures to the ResellerSales_USD table: **Countrows ALLNOBLANKROW of ResellerSales_USD**, **Countrows ALL of ResellerSales_USD**.

Create a new PivotTable, and drag the column, datetime.[Calendar Year], to the Row Labels pane. Now add the measures that you just created. The results should look like the following:

Row Labels	Countrows ALLNOBLANKROW of ResellerSales_USD	Countrows ALL of ResellerSales_USD
2005	60856	60856
2006	60856	60856
2007	60856	60856
2008	60856	60856
	60856	60856
Grand Total	60856	60856

Now the two measures have the same results. That is because the ALLNOBLANKROW function does not count truly blank rows in a table, but only handles the blank row that is a special case generated in a parent table, when one or more of the child tables in the relationship contain non-matching values or blank values.

Code

```
// Countrows ALLNOBLANK of datetime
= COUNTROWS(ALLNOBLANKROW('DateTime'))

// Countrows ALL of datetime
= COUNTROWS(ALL('DateTime'))

// Countrows ALLNOBLANKROW of ResellerSales_USD
=COUNTROWS(ALLNOBLANKROW('ResellerSales_USD'))

// Countrows ALL of ResellerSales_USD
=COUNTROWS(ALL('ResellerSales_USD'))
```

Comments

See Also

Filter and value functions

ALL

FILTER

ALLSELECTED Function

Removes context filters from columns and rows in the current query, while retaining all other context filters or explicit filters.

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.

Syntax

ALLSELECTED([<tableName> | <columnName>])

Parameters

Parameter	Description
	The name of an existing table,

	using standard DAX syntax. This parameter cannot be an expression. This parameter is optional. The name of an existing column using standard DAX syntax, usually fully qualified. It cannot be an expression. This parameter is optional.

Return Value

The context of the query without any column and row filters.

Exceptions

Remarks

- This function takes one or no arguments. If there is one argument, the argument is either tableName or columnName.
- This function is different from ALL() because it retains all filters explicitly set within the query, and it retains all context filters other than row and column filters.

Example

Description

The following example shows how to generate different levels of visual totals in a table report using DAX expressions. In the report two (2) previous filters have been applied to the Reseller Sales data; one on Sales Territory Group = *Europe* and the other on Promotion Type = *Volume Discount*. Once filters have been applied, visual totals can be calculated for the entire report, for All Years, or for All Product Categories. Also, for illustration purposes the grand total for All Reseller Sales is obtained too, removing all filters in the report. Evaluating the following DAX expression results in a table with all the information needed to build a table with Visual Totals.

Code

```
define
measure 'Reseller Sales'[Reseller Sales Amount]=sum('Reseller
Sales'[Sales Amount])
measure 'Reseller Sales'[Reseller Grand Total]=calculate(sum('Reseller
Sales'[Sales Amount]), ALL('Reseller Sales'))
measure 'Reseller Sales'[Reseller Visual Total]=calculate(sum('Reseller
Sales'[Sales Amount]), ALLSELECTED())
```

```
measure 'Reseller Sales'[Reseller Visual Total for All of Calendar
Year]=calculate(sum('Reseller Sales'[Sales Amount]),
ALLSELECTED('Date'[Calendar Year]))
measure 'Reseller Sales'[Reseller Visual Total for All of Product
Category Name]=calculate(sum('Reseller Sales'[Sales Amount]),
ALLSELECTED('Product Category'[Product Category Name]))
evaluate
CalculateTable(
    //CT table expression
    summarize(
//summarize table expression
crossjoin(distinct('Product Category'[Product Category Name]),
distinct('Date'[Calendar Year]))
//First Group by expression
, 'Product Category'[Product Category Name]
//Second Group by expression
, 'Date'[Calendar Year]
//Summary expressions
, "Reseller Sales Amount", [Reseller Sales Amount]
, "Reseller Grand Total", [Reseller Grand Total]
, "Reseller Visual Total", [Reseller Visual Total]
, "Reseller Visual Total for All of Calendar Year", [Reseller Visual
Total for All of Calendar Year]
, "Reseller Visual Total for All of Product Category Name", [Reseller
Visual Total for All of Product Category Name]
)
//CT filters
, 'Sales Territory'[Sales Territory Group]="Europe",
'Promotion'[Promotion Type]="Volume Discount"
)
order by [Product Category Name], [Calendar Year]
```

Comments

After executing the above expression in SQL Server Management Studio against AdventureWorks DW Tabular Model 2012 you obtain the following results:

[Product Category Name]	[Calendar Year]	[Reseller Sales Amount]	[Reseller Grand Total]	[Reseller Visual Total]	[Reseller Visual Total for All of Calendar Year]	[Reseller Visual Total for All of Product Category Name]
Accessories	2000		80450596.9823	877006.7987	38786.018	
Accessories	2001		80450596.9823	877006.7987	38786.018	
Accessories	2002	625.7933	80450596.9823	877006.7987	38786.018	91495.3104
Accessories	2003	26037.3132	80450596.9823	877006.7987	38786.018	572927.0136
Accessories	2004	12122.9115	80450596.9823	877006.7987	38786.018	212584.4747
Accessories	2005		80450596.9823	877006.7987	38786.018	
Accessories	2006		80450596.9823	877006.7987	38786.018	
Bikes	2000		80450596.9823	877006.7987	689287.7939	
Bikes	2001		80450596.9823	877006.7987	689287.7939	
Bikes	2002	73778.938	80450596.9823	877006.7987	689287.7939	91495.3104
Bikes	2003	439771.4136	80450596.9823	877006.7987	689287.7939	572927.0136
Bikes	2004	175737.4423	80450596.9823	877006.7987	689287.7939	212584.4747
Bikes	2005		80450596.98	877006.79	689287.79	

[Product Category Name]	[Calendar Year]	[Reseller Sales Amount]	[Reseller Grand Total]	[Reseller Visual Total]	[Reseller Visual Total for All of Calendar Year]	[Reseller Visual Total for All of Product Category Name]
			23	87	39	
Bikes	2006		80450596.9823	877006.7987	689287.7939	
Clothing	2000		80450596.9823	877006.7987	95090.7757	
Clothing	2001		80450596.9823	877006.7987	95090.7757	
Clothing	2002	12132.4334	80450596.9823	877006.7987	95090.7757	91495.3104
Clothing	2003	58234.2214	80450596.9823	877006.7987	95090.7757	572927.0136
Clothing	2004	24724.1209	80450596.9823	877006.7987	95090.7757	212584.4747
Clothing	2005		80450596.9823	877006.7987	95090.7757	
Clothing	2006		80450596.9823	877006.7987	95090.7757	
Components	2000		80450596.9823	877006.7987	53842.2111	
Components	2001		80450596.9823	877006.7987	53842.2111	
Components	2002	4958.1457	80450596.9823	877006.7987	53842.2111	91495.3104
Components	2003	48884.0654	80450596.9823	877006.7987	53842.2111	572927.0136
Components	2004		80450596.9823	877006.7987	53842.2111	212584.4747
Components	2005		80450596.9823	877006.7987	53842.2111	

[Product Category Name]	[Calendar Year]	[Reseller Sales Amount]	[Reseller Grand Total]	[Reseller Visual Total]	[Reseller Visual Total for All of Calendar Year]	[Reseller Visual Total for All of Product Category Name]
Components	2006		80450596.9823	877006.7987	53842.2111	

The columns in the report are:

Reseller Sales Amount

The actual value of Reseller Sales for the year and product category. This value appears in a cell in the center of your report, at the intersection of year and catergory.

Reseller Visual Total for All of Calendar Year

The total value for a product category across all years. This value appears at the end of a column or row for a given product category and across all years in the report.

Reseller Visual Total for All of Product Category Name

The total value for a year across all product categories. This value appears at the end of a column or row for a given year and across all product categories in the report.

Reseller Visual Total

The total value for all years and product categories. This value usually appears in the bottom rightmost corner of the table.

Reseller Grand Total

This is the grand total for all reseller sales, before any filter has been applied; you should notice the difference with [Reseller Visual Total]. You do remember that this report includes two (2) filters, one on Product Category Group and the other in Promotion Type.

Example

Description

The following example shows the usage of ALLSELECTED() with no arguments to show how to calculate a ratio over the total value shown in a table that has been filtered by using horizontal and vertical slicers. This example uses SQL Server 2012, PowerPivot for Excel and PowerPivot Sample Data (DAX AdventureWorks).

- In the PowerPivot field list, drag the column, ResellerSales[SalesAmount_USD], to the **Values** area.
- Drag Promotion[PromotionType] to the **Slicers Vertical** area. Select the slicers for Discontinued Product, Excess Inventory and Seasonal Discount.

- Drag SalesTerritory[SalesTerritoryGroup] to the **Slicers Horizontal** area. Select the slicer for Europe.
- The value of **Sum Of SalesAmount_USD** should be $19,614.37
- Drag ProductCategory[ProductCategoryName] to the **Row Labels** area and DateTime[CalendarYear] to the **Column Labels** area.

  Your table should look similar to:

	SalesTerritory Group					
	Europe	North America				
	Pacific	NA				
Promotion Type						
Discontinued Product		**Sum of SalesAmount_USD**	**Column Labels**			
Excess Inventory		**Row Labels**	**2006**	**2007**	**2008**	**Grand Total**
New Product		Accessories	$1,111.22	$3,414.43		**$4,525.66**
No Discount		Bikes	$8,834.94		$6,253.78	**$15,088.72**
Seasonal Discount		**Grand Total**	**$9,946.16**	**$3,414.43**	**$6,253.78**	**$19,614.37**
Volume Discount						

- Using a hand calculator, verify that the amount $3,414.43 (for Accessories in 2007) represents 17.41% of $19,614.37.
- Using a hand calculator, verify that the amount $6,253.78 (for Bikes in 2008) represents 31.88% of $19,614.37.
- Using a hand calculator, verify that the amount $15,088.72 (for Bikes Grand Total) represents 76.93% of $19,614.37.

- Remove ResellerSales[SalesAmount_USD] from the **Values** area.
- Create a measure, named **Reseller Sales Ratio**, in the ResellerSales table, using the following formula (format the result as percentage):

Code

```
=SUM(ResellerSales_USD[SalesAmount_USD])/
CALCULATE(SUM(ResellerSales_USD[SalesAmount_USD]), ALLSELECTED())
```

Comments

Your table should look like this:

	SalesTerritoryGroup					
	Europe	North America				
	Pacific	NA				
PromotionType						
Discontinued Product		**Reseller Sales Ratio**	**Column Labels**			
Excess Inventory		**Row Labels**	**2006**	**2007**	**2008**	**Grand Total**
New Product		Accessories	5.67 %	17.41 %		**23.07 %**
No Discount		Bikes	45.04 %		31.88 %	**76.93 %**
Seasonal Discount		**Grand Total**	**50.71 %**	**17.41 %**	**31.88 %**	**100.00 %**
Volume Discount						

- Compare your handheld calculator results with those given by **Reseller Sales Ratio** and they should match; your denominator value is fixed as the value at the bottom of the table.
- Because the CALCULATE formula in the denominator uses the ALLSELECTED function, the denominator represents the grand total of sales, at the bottom of the table, after the vertical, horizontal and page slicers have been applied; but, before the row and column slicers are applied .

Note: if you have explicit filters in your expression, those filters are also applied to the expression.

CALCULATE Function

Evaluates an expression in a context that is modified by the specified filters.

Syntax

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

Parameters

Term	Definition
expression	The expression to be evaluated.
filter1,**filter2**,…	(optional) A comma separated list of Boolean expression or a table expression that defines a filter.

The expression used as the first parameter is essentially the same as a measure.

The following restrictions apply to Boolean expressions that are used as arguments:

- The expression cannot reference a measure.
- The expression cannot use a nested CALCULATE function.
- The expression cannot use any function that scans a table or returns a table, including aggregation functions.

However, a Boolean expression can use any function that looks up a single value, or that calculate a scalar value.

Return Value

The value that is the result of the expression.

Remarks

If the data has been filtered, the CALCULATE function changes the context in which the data is filtered, and evaluates the expression in the new context that you specify. For each column used in a filter argument, any existing filters on that column are removed, and the filter used in the filter argument is applied instead.

Example

Description

To calculate the ratio of current reseller sales to all reseller sales, you add to the PivotTable a measure that calculates the sum of sales for the current cell (the numerator), and then divides that sum by the total sales for all resellers (the denominator). To ensure

that the denominator remains the same regardless of how the PivotTable might be filtering or grouping the data, the part of the formula that represents the denominator must use the ALL function to clear any filters and create the correct total.

The following table shows the results when the new measure, named **All Reseller Sales Ratio**, is created by using the formula in the code section.

To see how this works, add the field, CalendarYear, to the **Row Labels** area of the PivotTable, and add the field, ProductCategoryName, to the **Column Labels** area. Then add the new measure to the **Values** area of the PivotTable. To display the numbers as percentages, apply percentage number formatting to the area of the PivotTable that contains the new measure, **All Reseller Sales Ratio**.

All Reseller Sales	**Column Labels**				
Row Labels	Accessories	Bikes	Clothing	Components	Grand Total
2005	0.02%	9.10%	0.04%	0.75%	9.91%
2006	0.11%	24.71%	0.60%	4.48%	29.90%
2007	0.36%	31.71%	1.07%	6.79%	39.93%
2008	0.20%	16.95%	0.48%	2.63%	20.26%
Grand Total	0.70%	82.47%	2.18%	14.65%	100.00%

Code

```
=( SUM('ResellerSales_USD'[SalesAmount_USD]))
 /CALCULATE( SUM('ResellerSales_USD'[SalesAmount_USD])
           ,ALL('ResellerSales_USD'))
```

Comments

The CALCULATE expression in the denominator enables the sum expression to include all rows in the calculation. This overrides the implicit filters for CalendarYear and ProductCategoryName that exist for the numerator part of the expression.

Related Functions

Whereas the CALCULATE function requires as its first argument an expression that returns a single value, the CALCULATETABLE function takes a table of values.

See Also

CALCULATETABLE Function

Filter and value functions

CALCULATETABLE Function

Evaluates a table expression in a context modified by the given filters.

Syntax

CALCULATETABLE(<expression>,<filter1>,<filter2>,...)

Parameters

Term	Definition
Expression	The table expression to be evaluated
filter1,**filter2**,...	A Boolean expression or a table expression that defines a filter

The expression used as the first parameter must be a function that returns a table.

The following restrictions apply to Boolean expressions that are used as arguments:

- The expression cannot reference a measure.
- The expression cannot use a nested CALCULATE function.
- The expression cannot use any function that scans a table or returns a table, including aggregation functions.

However, a Boolean expression can use any function that looks up a single value, or that calculates a scalar value.

Return Value

A table of values.

Remarks

The CALCULATETABLE function changes the context in which the data is filtered, and evaluates the expression in the new context that you specify. For each column used in a filter argument, any existing filters on that column are removed, and the filter used in the filter argument is applied instead.

This function is a synonym for the RELATEDTABLE function.

Example

Description

The following example uses the CALCULATETABLE function to get the sum of Internet sales for 2006. This value is later used to calculate the ratio of Internet sales compared to all sales for the year 2006.

The following table shows the results from the following formula.

Row Labels	Internet SalesAmount_USD	CalculateTable 2006 Internet Sales	Internet Sales to 2006 ratio
2005	$2,627,031.40	$5,681,440.58	0.46
2006	$5,681,440.58	$5,681,440.58	1.00
2007	$8,705,066.67	$5,681,440.58	1.53
2008	$9,041,288.80	$5,681,440.58	1.59
Grand Total	$26,054,827.45	$5,681,440.58	4.59

Code

```
=SUMX( CALCULATETABLE('InternetSales_USD',
'DateTime'[CalendarYear]=2006)
     , [SalesAmount_USD])
```

See Also

RELATEDTABLE Function (DAX)

Filter and value functions

DISTINCT Function

Returns a one-column table that contains the distinct values from the specified column. In other words, duplicate values are removed and only unique values are returned.

Note
This function cannot be used to return values into a cell or column on a worksheet; rather, you nest the DISTINCT function within a formula, to get a list of distinct values that can be passed to another function and then counted, summed, or used for other operations.

Syntax

DISTINCT(<column>)

Parameters

Term	Definition
column	The column from which unique values are to be returned. Or, an expression that returns a column.

Return Value

A column of unique values.

Remarks

The results of DISTINCT are affected by the current filter context. For example, if you use the formula in the following example to create a measure, the results would change whenever the table was filtered to show only a particular region or a time period.

Related Functions

The VALUES function is similar to DISTINCT; it can also be used to return a list of unique values, and generally will return exactly the same results as DISTINCT. However, in some context VALUES will return one additional special value. For more information, see VALUES Function (DAX).

Example

Description

The following formula counts the number of unique customers who have generated orders over the internet channel. The table that follows illustrates the possible results when the formula is added to a PivotTable.

Code

```
=COUNTROWS(DISTINCT(InternetSales_USD[CustomerKey]))
```

Comments

Note that you cannot paste the list of values that DISTINCT returns directly into a column. Instead, you pass the results of the DISTINCT function to another function that counts, filters, or aggregates values by using the list. To make the example as simple as possible, here the table of distinct values has been passed to the COUNTROWS function.

Unique Internet customers	**Column Labels**			
Row Labels	Accessories	Bikes	Clothing	Grand Total
2005		1013		1013
2006		2677		2677
2007	6792	4875	2867	9309
2008	9435	5451	4196	11377
Grand Total	15114	9132	6852	18484

Also, note that the results are not additive. That is to say, the total number of unique customers in 2007 is not the sum of unique customers of Accessories, Bikes and Clothing for that year. The reason is that a customer can be counted in multiple groups.

See Also

Filter and value functions

FILTER

RELATED

VALUES

EARLIER Function

Returns the current value of the specified column in an outer evaluation pass of the mentioned column.

EARLIER is useful for nested calculations where you want to use a certain value as an input and produce calculations based on that input. In Microsoft Excel, you can do such calculations only within the context of the current row; however, in DAX you can store the value of the input and then make calculation using data from the entire table.

EARLIER is mostly used in the context of calculated columns.

Syntax

EARLIER(<column>, <number>)

Parameters

Term	Definition
column	A column or expression that resolves to a column.
num	(Optional) A positive number to the outer evaluation pass. The next evaluation level out is represented by 1; two levels out is represented by 2 and so on. When omitted default value is 1.

Property Value/Return Value

The current value of row, from **column**, at **number** of outer evaluation passes.

Exceptions

Description of errors

Remarks

EARLIER succeeds if there is a row context prior to the beginning of the table scan. Otherwise it returns an error.

The performance of **EARLIER** might be slow because it theoretically, it might have to perform a number of operations that is close to the total number of rows (in the column) times the same number (depending on the syntax of the expression). For example if you have 10 rows in the column, approximately a 100 operations could be required; if you have 100 rows then close to 10,000 operations might be performed.

Note
In practice, the xVelocity in-memory analytics engine (VertiPaq) performs optimizations to reduce the actual number of calculations, but you should be cautious when creating formulas that involve recursion.

Example

Description

To illustrate the use of EARLIER, it is necessary to build a scenario that calculates a rank value and then uses that rank value in other calculations.

The following example is based on this simple table, **ProductSubcategory**, which shows the total sales for each ProductSubcategory.

The final table, including the ranking column is shown here.

ProductSubcategoryKey	EnglishProductSubcategoryName	TotalSubcategorySales	SubcategoryRanking
18	Bib-Shorts	$156,167.88	18
26	Bike Racks	$220,720.70	14
27	Bike Stands	$35,628.69	30
28	Bottles and Cages	$59,342.43	24
5	Bottom Brackets	$48,643.47	27
6	Brakes	$62,113.16	23
19	Caps	$47,934.54	28
7	Chains	$8,847.08	35
29	Cleaners	$16,882.62	32
8	Cranksets	$191,522.09	15
9	Derailleurs	$64,965.33	22
30	Fenders	$41,974.10	29
10	Forks	$74,727.66	21
20	Gloves	$228,353.58	12

ProductSubcategoryKey	EnglishProductSubcategoryName	TotalSubcategorySales	SubcategoryRanking
4	Handlebars	$163,257.06	17
11	Headsets	$57,659.99	25
31	Helmets	$451,192.31	9
32	Hydration Packs	$96,893.78	20
21	Jerseys	$699,429.78	7
33	Lights		36
34	Locks	$15,059.47	33
1	Mountain Bikes	$34,305,864.29	2
12	Mountain Frames	$4,511,170.68	4
35	Panniers		36
13	Pedals	$140,422.20	19
36	Pumps	$12,695.18	34
2	Road Bikes	$40,551,696.34	1
14	Road Frames	$3,636,398.71	5
15	Saddles	$52,526.47	26
22	Shorts	$385,707.80	10
23	Socks	$28,337.85	31
24	Tights	$189,179.37	16
37	Tires and Tubes	$224,832.81	13
3	Touring Bikes	$13,334,864.18	3
16	Touring Frames	$1,545,344.02	6
25	Vests	$240,990.04	11
17	Wheels	$648,240.04	8

Creating a Rank Value

One way to obtain a rank value for a given value in a row is to count the number of rows, in the same table, that have a value larger (or smaller) than the one that is being compared. This technique returns a blank or zero value for the highest value in the table,

whereas equal values will have the same rank value and next value (after the equal values) will have a non consecutive rank value. See the sample below.

A new calculated column, **SubCategorySalesRanking**, is created by using the following formula.

Code

```
= COUNTROWS(FILTER(ProductSubcategory,
EARLIER(ProductSubcategory[TotalSubcategorySales])<ProductSubcategory[T
otalSubcategorySales]))+1
```

Comments

The following steps describe the method of calculation in more detail.

1. The **EARLIER** function gets the value of TotalSubcategorySales for the current row in the table. In this case, because the process is starting, it is the first row in the table
2. **EARLIER**([TotalSubcategorySales]) evaluates to $156,167.88, the current row in the outer loop.
3. The **FILTER** function now returns a table where all rows have a value of TotalSubcategorySales larger than $156,167.88 (which is the current value for **EARLIER**).
4. The **COUNTROWS** function counts the rows of the filtered table and assigns that value to the new calculated column in the current row plus 1. Adding 1 is needed to prevent the top ranked value from become a Blank.
5. The calculated column formula moves to the next row and repeats steps 1 to 4. These steps are repeated until the end of the table is reached.

The **EARLIER** function will always get the value of the column prior to the current table operation. If you need to get a value from the loop before that, set the second argument to 2.

See Also

EARLIESTfunction

Filter and value functions

EARLIEST Function

Returns the current value of the specified column in an outer evaluation pass of the specified column.

Syntax

EARLIEST(<column>)

Parameters

Term	Definition
column	A reference to a column.

Property Value/Return Value

A column with filters removed.

Remarks

The EARLIEST function is similar to EARLIER, but lets you specify one additional level of recursion.

Example

Description

The current sample data does not support this scenario.

Code

```
=EARLIEST(<column>)
```

See Also

EARLIER function

Filter and value functions

FILTER Function

Returns a table that represents a subset of another table or expression.

Syntax

FILTER(<table>,<filter>)

Parameters

Term	Definition
table	The table to be filtered. The table can also be an expression that results in a table.
filter	A Boolean expression that is to be evaluated for each row of the table. For example, `[Amount] > 0` or `[Region] = "France"`

Return Value

A table containing only the filtered rows.

Remarks

You can use FILTER to reduce the number of rows in the table that you are working with, and use only specific data in calculations. FILTER is not used independently, but as a function that is embedded in other functions that require a table as an argument.

Example

Description

The following example creates a report of Internet sales outside the United States by using a measure that filters out sales in the United States, and then slicing by calendar year and product categories. To create this measure, you filter the table, Internet Sales USD, by using Sales Territory, and then use the filtered table in a SUMX function.

In this example, the expression `FILTER('InternetSales_USD', RELATED('SalesTerritory'[SalesTerritoryCountry])<>"United States")` returns a table that is a subset of Internet Sales minus all rows that belong to the United States sales territory. The RELATED function is what links the Territory key in the Internet Sales table to SalesTerritoryCountry in the SalesTerritory table.

The following table demonstrates the proof of concept for the measure, NON USA Internet Sales, the formula for which is provided in the code section below. The table compares all Internet sales with non- USA Internet sales, to show that the filter expression works, by excluding United States sales from the computation.

To re-create this table, add the field, SalesTerritoryCountry, to the **Row Labels** area of the PivotTable.

Table 1. Comparing total sales for U.S. vs. all other regions

Row Labels	Internet Sales	Non USA Internet Sales
Australia	$4,999,021.84	$4,999,021.84
Canada	$1,343,109.10	$1,343,109.10
France	$2,490,944.57	$2,490,944.57
Germany	$2,775,195.60	$2,775,195.60
United Kingdom	$5,057,076.55	$5,057,076.55
United States	$9,389,479.79	
Grand Total	$26,054,827.45	$16,665,347.67

The final report table shows the results when you create a PivotTable by using the measure, NON USA Internet Sales. Add the field, CalendarYear, to the **Row Labels** area of the PivotTable and add the field, ProductCategoryName, to the **Column Labels** area.

Table 2. Comparing non- U.S. sales by product categories

Non USA Internet Sales	Column Labels			
Row Labels	Accessories	Bikes	Clothing	Grand Total
2005		$1,526,481.95		$1,526,481.95
2006		$3,554,744.04		$3,554,744.04
2007	$156,480.18	$5,640,106.05	$70,142.77	$5,866,729.00
2008	$228,159.45	$5,386,558.19	$102,675.04	$5,717,392.68
Grand Total	$384,639.63	$16,107,890.23	$172,817.81	$16,665,347.67

Code

```
SUMX(FILTER('InternetSales_USD',
RELATED('SalesTerritory'[SalesTerritoryCountry])<>"United States")
     ,'InternetSales_USD'[SalesAmount_USD])
```

See Also

Filter and value functions

ALL

ALLEXCEPT

FILTERS Function

Returns the values that are directly applied as filters to columnName.

Syntax

FILTERS(<columnName>)

Parameters

Parameter	Description
	The name of an existing column, using standard DAX syntax. It cannot be an expression.

Return Value

The values that are directly applied as filters to columnName.

Remarks

Example

Description

The following example shows how to determine the number of direct filters a column has.

Code

```
=COUNTROWS(FILTERS(ResellerSales_USD[ProductKey]))
```

Comments

The example above lets you know how many direct filters on ResellerSales_USD[ProductKey] have been applied to the context where the expression is being evaluated.

HASONEFILTER Function

Returns **TRUE** when the number of directly filtered values on columnName is one; otherwise returns **FALSE**.

Syntax

HASONEFILTER(<columnName>)

Parameters

Parameter	Description
	The name of an existing column, using standard DAX syntax. It cannot be an expression.

Return Value

TRUE when the number of directly filtered values on columnName is one; otherwise returns **FALSE**.

Remarks

1. An equivalent expression for HASONEFILTER() is `COUNTROWS(FILTERS(<columnName>)) = 1`.
2. This function is similar to HASONEVALUE() with the difference that HASONEVALUE() works based on cross-filters while HASONEFILTER() works by a direct filter.

Example

Description

The following example shows how to use HASONEFILTER() to return the filter for ResellerSales_USD[ProductKey]) if there is one filter, or to return BLANK if there are no filters or more than one filter on ResellerSales_USD[ProductKey]).

Code

```
=IF(HASONEFILTER(ResellerSales_USD[ProductKey]),FILTERS(ResellerSales_USD[ProductKey]),BLANK())
```

Comments

HASONEVALUE Function

Returns **TRUE** when the context for columnName has been filtered down to one distinct value only. Otherwise is **FALSE**.

Syntax

HASONEVALUE(<columnName>)

Parameters

Parameter	Description
	The name of an existing column, using standard DAX syntax. It cannot be an expression.

Return Value

TRUE when the context for columnName has been filtered down to one distinct value only. Otherwise is **FALSE**.

Remarks

- An equivalent expression for HASONEVALUE() is `COUNTROWS(VALUES(<columnName>)) = 1`.

Example

Description

In the following example you want to create a formula that verifies if the context is being sliced by one value in order to estimate a percentage against a predefined scenario; in this case you want to compare Reseller Sales against sales in 2007, then you need to know if the context is filtered by single years. Also, if the comparison is meaningless you want to return BLANK.

If you want to follow the scenario, you can download the spreadsheet with the model from PowerPivot Sample Data spreadsheet.

Create a measure named [ResellerSales compared to 2007] using the following expression:

Code

```
=IF(HASONEVALUE(DateTime[CalendarYear]),SUM(ResellerSales_USD[SalesAmou
nt_USD])/CALCULATE(SUM(ResellerSales_USD[SalesAmount_USD]),DateTime[Cal
endarYear]=2007),BLANK())
```

Comments

1. After creating the measure you should have an empty result under [ResellerSales compared to 2007], as shown below:

ResellerSales compared to 2007

   The BLANK cell in the result is because you don't have single year filters anywhere in your context.

2. Drag DateTime[CalendarYear] to the **Column Labels** box; your table should look like this:

	Column Labels			
	2005	**2006**	**2007**	**2008**
ResellerSales compared to 2007	**24.83 %**	**74.88 %**	**100.00 %**	**50.73 %**

3. Drag ProductCategory[ProductCategoryName] to the **Row Labels** box to have something like this:

ResellerSales compared to 2007	Column Labels			
Row Labels	**2005**	**2006**	**2007**	**2008**
Accessories	6.74 %	31.40 %	100.00 %	55.58 %
Bikes	28.69 %	77.92 %	100.00 %	53.46 %
Clothing	3.90 %	55.86 %	100.00 %	44.92 %
Components	11.05 %	65.99 %	100.00 %	38.65 %
Grand Total	**24.83 %**	**74.88 %**	**100.00 %**	**50.73 %**

Did you notice that **Grand Totals** appeared at the bottom of the columns but not for the rows? This is because the context for Grand Totals on rows implies more than one year; but for columns implies a single year.

4. Drag DateTime[CalendarYear] to the **Horizontal Slicers** box, and drag SalesTerritory[SalesTerritoryGroup] to the **Horizontal Labels** box. You should have an empty results set, because your table contains data for multiple years. Select **2006** in the slicer and your table should now have data again. Try other years to see how the results change.
5. In summary, HASONEVALUE() allows you to identify if your expression is being evaluated in the context of a single value for columnName.

ISCROSSFILTERED Function

Returns TRUE when columnName or another column in the same or related table is being filtered.

Syntax

ISCROSSFILTERED(<columnName>)

Parameters

Parameter	Description
	The name of an existing column, using standard DAX syntax. It cannot be an expression.

Return Value

TRUE when columnName or another column in the same or related table is being filtered. Otherwise returns **FALSE**.

Remarks

- A column is said to be cross-filtered when a filter applied to another column in the same table or in a related table affects columnName by filtering it. A column is said to be filtered *directly* when the filter or filters apply over the column.
- The related function ISFILTERED Function (DAX) returns TRUE when columnName is filtered directly.

Example

Description

The following example creates a measure and then presents different scenarios to explain the behavior of ISCROSSFILTERED(). The scenarios can be followed by downloading the DAX_AdventureWorks spreadsheet.

First, create the following measure [Is ProductName Cross Filtered] in the [Product] table.

Code

```
=ISCROSSFILTERED(Product[ProductName])
```

Comments

Understanding ISCROSSFILTERED():

1. After you create the measure, the PivotTable should show that [Is ProductName Cross Filtered] is **FALSE**, because the expression is not being filtered at all. Now, you should have something like this:

Is ProductName Cross Filtered
FALSE

   If nothing appears in the PivotTable, drag the column [Is ProductName Cross Filtered] to the **Values** box.

2. Drag DateTime[CalendarYear] to the **Column Labels** box and drag SalesTerritory[SalesTerritoryGroup] to the **Row Labels** box; all values should appear as **FALSE**. Now, you should have something like this:

Is Product Name CrossFiltered	Column Labels					
Row Labels	2005	2006	2007	2008		Grand Total
Europe	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
NA	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
North America	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Pacific	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Grand Total	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE

   You are seeing the value FALSE in all cells because neither DateTime[CalendarYear] or SalesTerritory[SalesTerritoryGroup] have any filtering effect over Product[ProductName].

3. Drag the column ProductCategory[ProductCategoryName] to the **Slicers Horizontal** box and ProductSubCategory[ProductSubCategoryName] to the **Slicers Vertical** box. All values should still appear as **FALSE**, because when you add a column to the slicers

box you haven't yet selected a slicing set. Therefore, you should have something like this:

	ProductCategoryName						
	Accessories	*Bikes*					
	Clothing	*Components*					
ProductSubcategoryName	Is Product Name CrossFiltered	Column Labels					
Bib-Shorts	Row Labels	2005	2006	2007	2008		Grand Total
Bike-Racks	Europe	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Bike-Stands	NA	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Bottles and Cages	North America	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Bottom Brackets	Pacific	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Brakes		FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Caps	Grand Total	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Chains							
Cleaners							
Cranksets							
Derailleurs							
Fenders							
Forks							

Gloves							
Handlebars							
Headsets							

4. Select any item in the slicers and your table will now turn all cells to TRUE because you are now filtering the Product[ProductName] column through the related tables ProductCategory and ProductSubcategory. Your results should look like this:

	ProductCategory Name						
	Accessories	***Bikes***					
	Clothing	*Components*					
ProductSubcategory Name	Is Product Name CrossFiltered	Column Labels					
Bib-Shorts	Row Labels	2005	2006	2007	2008		Grand Total
Bike-Racks	Europe	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
Bike-Stands	NA	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
Bottles and Cages	North America	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
Bottom Brackets	Pacific	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
Brakes		TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
Caps	Grand Total	TRUE	TRUE	TRUE	TRUE	TRUE	TRUE
Chains							

Cleaners							
Cranksets							
Derailleurs							
Fenders							
Forks							
Gloves							
Handlebars							
Headsets							

5. You can now replace the slicers with Product[ProductName] and Product[ModelName]. As long as you do not select any item on any slicer the measure returns **FALSE**; however, when you select an item in the slicer, the measure now returns **TRUE** because you are filtering Product[ProductName] by itself or by selecting a Product[ModelName].
6. In conclusion, ISCROSSFILTERED() allows you to determine if columnName is being filtered in the context where your DAX expression is being evaluated.

This example uses the DAX AdventureWorks sample spreadsheet model.

See Also

ISFILTERED Function (DAX)

FILTERS Function (DAX)

HASONEFILTER Function (DAX)

HASONEVALUE Function (DAX)

ISFILTERED Function

Returns TRUE when columnName is being filtered directly. If there is no filter on the column or if the filtering happens because a different column in the same table or in a related table is being filtered then the function returns **FALSE**.

Syntax

ISFILTERED(<columnName>)

Parameters

Parameter	Description
	The name of an existing column, using standard DAX syntax. It

cannot be an expression.

Return Value

TRUE when columnName is being filtered directly.

Remarks

- columnName is said to be filtered directly when the filter or filters apply over the column; a column is said to be cross-filtered when a filter applied to another column in the same table or in a related table affects columnName the column by filtering it as well.
- The related function ISCROSSFILTERED Function (DAX) returns TRUE when columnName or another column in the same or related table is being filtered.

Example

Description

The following example creates a measure and then presents different scenarios to explain the behavior of ISFILTERED(). The scenarios can be followed by downloading the Data Analysis Expressions (DAX) Sample Data spreadsheet.

First, create the following measure [Is ProductName Filtered directly] in the [Product] table.

Code

```
=ISFILTERED(Product[ProductName])
```

Comments

Understanding ISFILTERED():

1. After you create the measure, the pivot table should show that [Is ProductName Filtered directly] is FALSE, because the expression is not being filtered at all. Now, you should have something like this:

Is ProductName Filtered directly
FALSE

   If nothing appears in the pivot table, drag [Is ProductName Filtered directly] to the **Values** box.

2. Drag ProductCategory[ProductCategoryName] to the **Row Labels** box; all values should still be **FALSE** and your table should look something like this:

Row Labels	Is ProductName Filtered directly
Accessories	FALSE

Bikes	FALSE
Clothing	FALSE
Components	FALSE
	FALSE
Grand Total	**FALSE**

The reason the table contains values of FALSE is because ProductCategory[ProductCategoryName] filters indirectly or cross-filters Product[ProductName], but is not a direct filter on the column.

3. Drag Product[Status] to the **Column Labels** box; all values should still be **FALSE** and your table should look something like this:

Is ProductName Filtered directly	**Column Labels**		
Row Labels		**Current**	**Grand Total**
Accessories	FALSE	FALSE	FALSE
Bikes	FALSE	FALSE	FALSE
Clothing	FALSE	FALSE	FALSE
Components	FALSE	FALSE	FALSE
	FALSE	FALSE	FALSE
Grand Total	**FALSE**	**FALSE**	**FALSE**

The table still has many values of **FALSE** because Product[Status] doesn't filter Product[Name] directly.

4. The last step in this process is to drag Product[Name] to the **Report Filter** box. At this point nothing should have changed; however, once you select a product name, the table values should change to **TRUE**. Depending on the product you selected your table should look like this:

ProductName	Mountain-400-W Silver, 38	
Is ProductName	**Column Labels**	

Filtered directly		
Row Labels	**Current**	**Grand Total**
Accessories	TRUE	TRUE
Bikes	TRUE	TRUE
Clothing	TRUE	TRUE
Components	TRUE	TRUE
	TRUE	TRUE
Grand Total	**TRUE**	**TRUE**

5. In conclusion, ISFILTERED() allows you to determine if columnName is being filtered directly in the context where your expression is being evaluated.

This example uses the DAX_AdventureWorks sample spreadsheet model

See Also

ISCROSSFILTERED Function (DAX)

FILTERS Function (DAX)

HASONEFILTER Function (DAX)

HASONEVALUE Function (DAX)

KEEPFILTERS Function

Modifies how filters are applied while evaluating a CALCULATE or CALCULATETABLE function.

Syntax

KEEPFILTERS(<expression>)

Parameters

Term	Definition
expression	Any expression.

Return Value

No return value.

Remarks

You use KEEPFILTERS within the context CALCULATE and CALCULATETABLE functions, to override the standard behavior of those functions.

By default, filter arguments s in functions such as CALCULATE are used as the context for evaluating the expression, and as such filter arguments for CALCULATE replace all existing filters over the same columns. The new context effected by the filter argument for CALCULATE affects only existing filters on columns mentioned as part of the filter argument. Filters on columns other than those mentioned in the arguments of CALCULATE or other related functions remain in effect and unaltered.

The KEEPFILTERS function allows you to modify this behavior. When you use KEEPFILTERS, any existing filters in the current context are compared with the columns in the filter arguments, and the intersection of those arguments is used as the context for evaluating the expression. The net effect over any one column is that both sets of arguments apply: both the filter arguments used in CALCULATE and the filters in the arguments of the KEEPFILTER function. In other words, whereas CALCULATE filters replace the current context, KEEPFILTERS adds filters to the current context.

Example

Description

The following example takes you through some common scenarios that demonstrate use of the KEEPFILTERS function as part of a CALCULATE or CALCULATETABLE formula.

The first three expressions obtain simple data to be used for comparisons:

- Internet Sales for the state of Washington.
- Internet Sales for the states of Washington and Oregon (both states combined).
- Internet Sales for the state of Washington and the province of British Columbia (both regions combined).

The fourth expression calculates Internet Sales for Washington and Oregon, while the filter for Washington and British Columbia is applied.

The next expression calculates Internet Sales for Washington and Oregon but uses KEEPFILTERS; the filter for Washington and British Columbia is part of the prior context.

Code

```
EVALUATE ROW(
  "$$ in WA"
    , CALCULATE('Internet Sales'[Internet Total Sales]
                , 'Geography'[State Province Code]="WA"
      )
, "$$ in WA and OR"
    , CALCULATE('Internet Sales'[Internet Total Sales]
                , 'Geography'[State Province Code]="WA"
```

```
                || 'Geography'[State Province Code]="OR"
      )
, "$$ in WA and BC"
    , CALCULATE('Internet Sales'[Internet Total Sales]
              , 'Geography'[State Province Code]="WA"
                || 'Geography'[State Province Code]="BC"
      )
, "$$ in WA and OR ??"
    , CALCULATE(
          CALCULATE('Internet Sales'[Internet Total Sales]
                   ,'Geography'[State Province Code]="WA"
                     || 'Geography'[State Province Code]="OR"
          )
          , 'Geography'[State Province Code]="WA"
            || 'Geography'[State Province Code]="BC"
      )
, "$$ in WA !!"
    , CALCULATE(
          CALCULATE('Internet Sales'[Internet Total Sales]
                  , KEEPFILTERS('Geography'[State Province Code]="WA"
                              || 'Geography'[State Province Code]="OR"
                    )
          )
          , 'Geography'[State Province Code]="WA"
            || 'Geography'[State Province Code]="BC"
      )
)
```

Comments

When this expression is evaluated against the sample database AdventureWorks DW Tabular 2012, the following results are obtained.

Column	Value
[$$ in WA]	$ 2,467,248.34
[$$ in WA and OR]	$ 3,638,239.88
[$$ in WA and BC]	$ 4,422,588.44
[$$ in WA and OR ??]	$ 3,638,239.88
[$$ in WA !!]	$ 2,467,248.34

Note
The above results were formatted to a table, instead of a single row, for educational purposes.

First, examine the expression, **[$$ in WA and OR ??]**. You might wonder how this formula could return the value for sales in Washington and Oregon, since the outer CALCULATE expression includes a filter for Washington and British Columbia. The answer is that the default behavior of CALCULATE overrides the outer filters in 'Geography'[State Province Code] and substitutes its own filter arguments, because the filters apply to the same column.

Next, examine the expression, **[$$ in WA !!]**. You might wonder how this formula could return the value for sales in Washington and nothing else, since the argument filter includes Oregon and the outer CALCULATE expression includes a filter in Washington and British Columbia. The answer is that KEEPFILTERS modifies the default behavior of CALCULATE and adds an additional filter. Because the intersection of filters is used, now the outer filter **'Geography'[State Province Code]="WA" || 'Geography'[State Province Code]="BC")** is added to the filter argument **'Geography'[State Province Code]="WA" || 'Geography'[State Province Code]="OR"**,. Because both filters apply to the same column, the resulting filter **'Geography'[State Province Code]="WA"** is the filter that is applied when evaluating the expression.

See Also

Filter and value functions

CALCULATE

CALCULATETABLE

RELATED Function

Returns a related value from another table.

Syntax

RELATED(<column>)

Parameters

Term	Definition
column	The column that contains the values you want to retrieve.

Return Value

A single value that is related to the current row.

Remarks

The RELATED function requires that a relationship exists between the current table and the table with related information. You specify the column that contains the data that you want, and the function follows an existing many-to-one relationship to fetch the value from the specified column in the related table.

If a relationship does not exist, you must create a relationship. For more information, see Creating a Relationship.

When the RELATED function performs a lookup, it examines all values in the specified table regardless of any filters that may have been applied.

Note

The RELATED function needs a row context; therefore, it can only be used in calculated column expression, where the current row context is unambiguous, or as a nested function in an expression that uses a table scanning function. A table scanning function, such as SUMX, gets the value of the current row value and then scans another table for instances of that value.

Example

Description

In the following example, the measure Non USA Internet Sales is created to produce a sales report that excludes sales in the United States. In order to create the measure, the InternetSales_USD table must be filtered to exclude all sales that belong to the United States in the SalesTerritory table. The United States, as a country, appears 5 times in the SalesTerritory table; once for each of the following regions: Northwest, Northeast, Central, Southwest, and Southeast.

The first approach to filter the Internet Sales, in order to create the measure, could be to add a filter expression like the following:

```
FILTER('InternetSales_USD', 'InternetSales_USD'[SalesTerritoryKey]<>1
&& 'InternetSales_USD'[SalesTerritoryKey]<>2 &&
'InternetSales_USD'[SalesTerritoryKey]<>3 &&
'InternetSales_USD'[SalesTerritoryKey]<>4 &&
'InternetSales_USD'[SalesTerritoryKey]<>5)
```

However, this approach is counterintuitive, prone to typing errors, and might not work if any of the existing regions is split in the future.

A better approach would be to use the existing relationship between InternetSales_USD and SalesTerritory and explicitly state that the country must be different from the United States. To do so, create a filter expression like the following:

```
FILTER( 'InternetSales_USD',
RELATED('SalesTerritory'[SalesTerritoryCountry])<>"United States")
```

This expression uses the RELATED function to lookup the country value in the SalesTerritory table, starting with the value of the key column, SalesTerritoryKey, in the InternetSales_USD table. The result of the lookup is used by the filter function to determine if the InternetSales_USD row is filtered or not.

Note

If the example does not work, you might need to create a relationship between the tables. For more information, see Relationships Between Tables.

Code

```
= SUMX(FILTER( 'InternetSales_USD'
            ,  RELATED('SalesTerritory'[SalesTerritoryCountry])
               <>"United States"
             )
     ,'InternetSales_USD'[SalesAmount_USD])
```

Comments

The following table shows only totals for each region, to prove that the filter expression in the measure, Non USA Internet Sales, works as intended.

Row Labels	Internet Sales	Non USA Internet Sales
Australia	$4,999,021.84	$4,999,021.84
Canada	$1,343,109.10	$1,343,109.10
France	$2,490,944.57	$2,490,944.57
Germany	$2,775,195.60	$2,775,195.60
United Kingdom	$5,057,076.55	$5,057,076.55
United States	$9,389,479.79	
Grand Total	$26,054,827.45	$16,665,347.67

The following table shows the final report that you might get if you used this measure in a PivotTable:

Non USA Internet Sales	Column Labels			
Row Labels	Accessories	Bikes	Clothing	Grand Total
2005		$1,526,481.95		$1,526,481.95
2006		$3,554,744.04		$3,554,744.04
2007	$156,480.18	$5,640,106.05	$70,142.77	$5,866,729.00
2008	$228,159.45	$5,386,558.19	$102,675.04	$5,717,392.68
Grand Total	$384,639.63	$16,107,890.23	$172,817.81	$16,665,347.67

See Also

RELATEDTABLE

Filter and value functions

Relationships between tables

RELATEDTABLE Function

Evaluates a table expression in a context modified by the given filters.

Syntax

RELATEDTABLE(<tableName>)

Parameters

Term	Definition
tableName	The name of an existing table using standard DAX syntax. It cannot be an expression.

Return Value

A table of values.

Remarks

The RELATEDTETABLE function changes the context in which the data is filtered, and evaluates the expression in the new context that you specify.

This function is a shortcut for CALCULATETABLE function with no logical expression.

Example

Description

The following example uses the RELATEDTABLE function to create a calculated column with the Internet Sales in the Product Category table.

The following table shows the results of using the code shown here.

Product Category Key	Product Category AlternateKey	Product Category Name	Internet Sales
1	1	Bikes	$28,318,144.65
2	2	Components	
3	3	Clothing	$339,772.61
4	4	Accessories	$700,759.96

Code

```
= SUMX( RELATEDTABLE('InternetSales_USD')
     , [SalesAmount_USD])
```

See Also

CALCULATETABLE Function (DAX)

Filter and value functions

Relationships between tables

USERELATIONSHIP Function

Specifies the relationship to be used in a specific calculation as the one that exists between columnName1 and columnName2.

Syntax

USERELATIONSHIP(<columnName1>,<columnName2>)

Parameters

Parameter	Description
	The name of an existing column, using standard DAX syntax and fully qualified, that usually represents the many side of the relationship to be used; if the arguments are given in reverse

	order the function will swap them before using them. This argument cannot be an expression.
	The name of an existing column, using standard DAX syntax and fully qualified, that usually represents the one side or lookup side of the relationship to be used; if the arguments are given in reverse order the function will swap them before using them. This argument cannot be an expression.

Return Value

The function returns no value; the function only enables the indicated relationship for the duration of the calculation.

Remarks

1. **USERELATIONSHIP** can only be used in functions that take a filter as an argument, for example: **CALCULATE**, **CALCULATETABLE**, **CLOSINGBALANCEMONTH**, **CLOSINGBALANCEQUARTER**, **CLOSINGBALANCEYEAR**, **OPENINGBALANCEMONTH**, **OPENINGBALANCEQUARTER**, **OPENINGBALANCEYEAR**, **TOTALMTD**, **TOTALQTD** and **TOTALYTD** functions.
2. USERELATIONSHIP uses existing relationships in the model, identifying relationships by their ending point columns.
3. In USERELATIONSHIP, the status of a relationship is not important; that is, whether the relationship is active or not does not affect the usage of the function. Even if the relationship is inactive, it will be used and overrides any other active relationships that might be present in the model but not mentioned in the function arguments.
4. An error is returned if any of the columns named as an argument is not part of a relationship or the arguments belong to different relationships.
5. If multiple relationships are needed to join table A to table B in a calculation, each relationship must be indicated in a different USERELATIONSHIP function.
6. If CALCULATE expressions are nested, and more than one CALCULATE expression contains a USERELATIONSHIP function, then the innermost USERELATIONSHIP is the one that prevails in case of a conflict or ambiguity.
7. Up to 10 USERELATIONSHIP functions can be nested; however, your expression might have a deeper level of nesting, ie. the following sample expression is nested 3 levels deep but only 2 for USEREALTIONSHIP: =CALCULATE(CALCULATE(CALCULATE(<anyExpression>, USERELATIONSHIP(t1[colA], t2[colB])), t99[colZ]=999), USERELATIONSHIP(t1[colA], t2[colA])).

Example

Description

The following sample shows how to override the default, active, relationship between InternetSales and DateTime tables. The default relationship exists between the OrderDate column, in the InternetSales table, and the Date column, in the DateTime table.

To calculate the sum of internet sales and allow slicing by ShippingDate instead of the traditional OrderDate you need to create a measure, [InternetSales by ShippingDate] using the following expression:

Code

```
=CALCULATE(SUM(InternetSales[SalesAmount]),
USERELATIONSHIP(InternetSales[ShippingDate], DateTime[Date]))
```

Comments

In PowerPivot: drag your new measure to the **Values** area in the right pane, drag the InternetSales[ShippingDate] column to the **Row Labels** area; you now have Internet Sales sliced by shipping date instead of by order date as is usually shown in these examples.

For this example to work the relationships between InternetSales[ShipmentDate] and DateTime[Date] must exist and should not be the active relationship; also, the relationship between InternetSales[OrderDate] and DateTime[Date] should exist and should be the active relationship.

VALUES Function

Returns a one-column table that contains the distinct values from the specified column. In other words, duplicate values are removed and only unique values are returned.

Note
This function cannot be used to return values into a cell or column on a worksheet; rather, you use it as an intermediate function, nested in a formula, to get a list of distinct values that can be counted, or used to filter or sum other values.

Syntax

VALUES(<column>)

Parameters

Term	Definition
column	The column from which unique values are

Term	Definition
	to be returned.

Return Value

A column of unique values.

Remarks

When you use the VALUES function in a context that has been filtered, such as in a PivotTable, the unique values returned by VALUES are affected by the filter. For example, if you filter by Region, and return a list of the values for City, the list will include only those cities in the regions permitted by the filter. To return all of the cities, regardless of existing filters, you must use the ALL function to remove filters from the table. The second example demonstrates use of ALL with VALUES.

Related Functions

In most scenarios, the results of the VALUES function are identical to those of the DISTINCT function. Both functions remove duplicates and return a list of the possible values in the specified column. However, the VALUES function can also return an *Unknown member*. This unknown value is useful in cases where you are looking up distinct values from a related table, but a value used in the relationship is missing from one table. In database terminology, this is termed a violation of referential integrity. Such mismatches in data can easily occur when one table is being updated and the related table is not.

The following table summarizes the mismatch between data that can occur in two related tables when referential integrity is not preserved.

MyOrders table	MySales table
June 1	June 1 sales
June 2	June 2 sales
(no order dates have been entered)	June 3 sales

If you used the DISTINCT function to return a list of dates from the PivotTable containing these tables, only two dates would be returned. However, if you use the VALUES function, the function returns the two dates plus an additional blank member. Also, any row from the MySales table that does not have a matching date in the MyOrders table will be "matched" to this unknown member.

Example

Description

The following formula counts the number of unique invoices (sales orders), and produces the following results when used in a report that includes the Product Category Names:

Row Labels	Count Invoices
Accessories	18,208
Bikes	15,205
Clothing	7,461
Grand Total	27,659

Code

```
=COUNTROWS(VALUES('InternetSales_USD'[SalesOrderNumber]))
```

See Also

FILTER

COUNTROWS

Filter and value functions

Information Functions

An information function looks at the cell or row that is provided as an argument and tells you whether the value matches the expected type. For example, the ISERROR function returns TRUE if the value that you reference contains an error.

In this Section

CONTAINS Function

ISBLANK Function

ISERROR Function

ISLOGICAL Function

ISNONTEXT Function

ISNUMBER Function

ISTEXT Function

LOOKUPVALUE Function

PATH Function

PATHCONTAINS Function

PATHITEM Function

PATHITEMREVERSE Function

PATHLENGTH Function

See Also

Function Reference (DAX)

CONTAINS Function

Returns true if values for all referred columns exist, or are contained, in those columns; otherwise, the function returns false.

Syntax

CONTAINS(<table>, <columnName>, <value>[, <columnName>, <value>]...)

Parameters

Parameter	Description
	Any DAX expression that returns a table of data.
	The name of an existing column, using standard DAX syntax. It cannot be an expression.
	Any DAX expression that returns a single scalar value, that is to be sought in columnName. The expression is to be evaluated exactly once and before it is passed to the argument list.

Return Value

A value of **TRUE** if each specified value can be found in the corresponding columnName, or are contained, in those columns; otherwise, the function returns **FALSE**.

Remarks

- The arguments columnName and value must come in pairs; otherwise an error is returned.
- columnName must belong to the specified table, or to a table that is related to table.
- If columnName refers to a column in a related table then it must be fully qualified; otherwise, an error is returned.

Example

Description

The following example creates a calculated measure that tells you whether there were any Internet sales of the product 214 and to customer 11185 at the same time.

Code

```
=CONTAINS(InternetSales, [ProductKey], 214, [CustomerKey], 11185)
```

Comments

CUSTOMDATA Function

Returns the content of the **CustomData** property in the connection string.

Syntax

CUSTOMDATA()

Return Value

The content of the **CustomData** property in the connection string.

Blank, if **CustomData** property was not defined at connection time.

Exceptions

Remarks

Example

Description

The following DAX code verifies if the CustomData property was set to **"OK"**.

Code

```
=IF(CUSTOMDATA()="OK", "Correct Custom data in connection string", "No
custom data in connection string property or unexpected value")
```

Comments

ISBLANK Function

Checks whether a value is blank, and returns TRUE or FALSE.

Syntax

ISBLANK(<value>)

Parameters

Term	Definition
value	The value or expression you want to test.

Return Value

A Boolean value of TRUE if the value is blank; otherwise FALSE.

Example

Description

This formula computes the increase or decrease ratio in sales compared to the previous year. The example uses the IF function to check the value for the previous year's sales in order to avoid a divide by zero error.

Row Labels	Total Sales	Total Sales Previous Year	Sales to Previous Year Ratio
2005	$10,209,985.08		
2006	$28,553,348.43	$10,209,985.08	179.66%
2007	$39,248,847.52	$28,553,348.43	37.46%
2008	$24,542,444.68	$39,248,847.52	-37.47%
Grand Total	$102,554,625.71		

Code

```
//Sales to Previous Year Ratio


=IF( ISBLANK('CalculatedMeasures'[PreviousYearTotalSales])
   , BLANK()
   , ( 'CalculatedMeasures'[Total Sales]-
'CalculatedMeasures'[PreviousYearTotalSales] )
      /'CalculatedMeasures'[PreviousYearTotalSales])
```

See Also

Information functions (DAX)

ISERROR Function

Checks whether a value is an error, and returns TRUE or FALSE.

Syntax

ISERROR(<value>)

Parameters

Term	Definition
value	The value you want to test.

Return Value

A Boolean value of TRUE if the value is an error; otherwise FALSE.

Example

Description

The following example calculates the ratio of total Internet sales to total reseller sales. The ISERROR function is used to check for errors, such as division by zero. If there is an error a blank is returned, otherwise the ratio is returned.

Code

```
= IF( ISERROR(
        SUM('ResellerSales_USD'[SalesAmount_USD])
        /SUM('InternetSales_USD'[SalesAmount_USD])
              )
    , BLANK()
    , SUM('ResellerSales_USD'[SalesAmount_USD])
      /SUM('InternetSales_USD'[SalesAmount_USD])
    )
```

See Also

Information functions (DAX)

IFERROR

IF

ISLOGICAL Function

Checks whether a value is a logical value, (TRUE or FALSE), and returns TRUE or FALSE.

Syntax

ISLOGICAL(<value>)

Parameters

Term	Definition
value	The value that you want to test.

Property Value/Return Value

TRUE if the value is a logical value; FALSE if any value other than TRUE OR FALSE.

Example

Description

The following three samples show the behavior of ISLOGICAL.

Code

```
//RETURNS: Is Boolean type or Logical
=IF(ISLOGICAL(true), "Is Boolean type or Logical", "Is different type")

//RETURNS: Is Boolean type or Logical
=IF(ISLOGICAL(false), "Is Boolean type or Logical", "Is different
type")

//RETURNS: Is different type
=IF(ISLOGICAL(25), "Is Boolean type or Logical", "Is different type")
```

See Also

Information functions (DAX)

ISNONTEXT Function

Checks if a value is not text (blank cells are not text), and returns TRUE or FALSE.

Syntax

ISNONTEXT(<value>)

Parameters

Term	Definition
value	The value you want to check.

Return Value

TRUE if the value is not text or blank; FALSE if the value is text.

Remarks

An empty string is considered text.

Example

Description

The following examples show the behavior of the ISNONTEXT function.

Code

```
//RETURNS: Is Non-Text
=IF(ISNONTEXT(1), "Is Non-Text", "Is Text")

//RETURNS: Is Non-Text
=IF(ISNONTEXT(BLANK()), "Is Non-Text", "Is Text")

//RETURNS: Is Text
=IF(ISNONTEXT(""), "Is Non-Text", "Is Text")
```

See Also

Information functions (DAX)

ISNUMBER Function

Checks whether a value is a number, and returns TRUE or FALSE.

Syntax

ISNUMBER(<value>)

Parameters

Term	Definition
value	The value you want to test.

Property Value/Return Value

TRUE if the value is numeric; otherwise FALSE.

Example

Description

The following three samples show the behavior of ISNUMBER.

Code

```
//RETURNS: Is number
=IF(ISNUMBER(0), "Is number", "Is Not number")
```

```
//RETURNS: Is number
=IF(ISNUMBER(3.1E-1),"Is number", "Is Not number")

//RETURNS: Is Not number
=IF(ISNUMBER("123"), "Is number", "Is Not number")
```

Comments

See Also

Information functions

ISTEXT Function

Checks if a value is text, and returns TRUE or FALSE.

Syntax

ISTEXT(<value>)

Parameters

Term	Definition
value	The value you want to check.

Property Value/Return Value

TRUE if the value is text; otherwise FALSE

Example

Description

The following examples show the behavior of the ISTEXT function.

Code

```
//RETURNS: Is Text
=IF(ISTEXT("text"), "Is Text", "Is Non-Text")

//RETURNS: Is Text
=IF(ISTEXT(""), "Is Text", "Is Non-Text")

//RETURNS: Is Non-Text
=IF(ISTEXT(1), "Is Text", "Is Non-Text")
```

```
//RETURNS: Is Non-Text
=IF(ISTEXT(BLANK()), "Is Text", "Is Non-Text")
```

See Also

Information functions (DAX)

LOOKUPVALUE Function

Returns the value in result_columnName for the row that meets all criteria specified by search_columnName and search_value.

Syntax

LOOKUPVALUE(<result_columnName>, <search_columnName>, <search_value>[, <search_columnName>, <search_value>]...)

Parameters

Parameter	Description
	The name of an existing column that contains the value you want to return. The column must be named using standard DAX syntax, usually, fully qualified. It cannot be an expression.
	The name of an existing column, in the same table as result_columnName or in a related table, over which the look-up is performed. The column must be named using standard DAX syntax, usually, fully qualified. It cannot be an expression.
	A scalar expression that does not refer to any column in the same table being searched.

Return Value

The value of result_column at the row where all pairs of search_column and search_value have a match.

If there is no match that satisfies all the search values, a BLANK is returned. In other words, the function will not return a lookup value if only some of the criteria match.

If multiple rows match the search values and in all cases result_column values are identical then that value is returned. However, if result_column returns different values an error is returned.

Remarks

Example

Description

The following example returns the SafetyStocklLevel for the bike model "Mountain-400-W Silver, 46".

Code

```
=LOOKUPVALUE(Product[SafetyStockLevel], [ProductName], " Mountain-400-W Silver, 46")
```

Comments

PATH Function

Returns a delimited text string with the identifiers of all the parents of the current identifier, starting with the oldest and continuing until current.

Syntax

PATH(<ID_columnName>, <parent_columnName>)

Parameters

Parameter	Description
	The name of an existing column containing the unique identifier for rows in the table. This cannot be an expression. The data type of the value in ID_columnName must be text or integer, and must also be the same data type as the column referenced in parent_columnName.
	The name of an existing column containing the unique identifier for the parent of the current row. This cannot be an expression. The data type of the value in parent_columnName data type must be text or integer, and must

be the same data type as the value in ID_columnName.

Return Value

A delimited text string containing the identifiers of all the parents to the current identifier.

Remarks

This function is used in tables that have some kind of internal hierarchy, to return the items that are related to the current row value. For example, in an Employees table that contains employees, the managers of employees, and the managers of the managers, you can return the path that connects an employee to his or her manager.

The path is not constrained to a single level of parent-child relationships; it can return related rows that are several levels up from the specified starting row.

The delimiter used to separate the ascendants is the vertical bar, '|'.

The values in ID_columnName and parent_columnName must have the same data type, text or integer.

Values in parent_columnName must be present in ID_columnName. That is, you cannot look up a parent if there is no value at the child level.

If parent_columnName is BLANK then PATH() returns ID_columnName value. In other words, if you look for the manager of an employee but the parent_columnName column has no data, the PATH function returns just the employee ID.

If ID_columnName has duplicates and parent_columnName is the same for those duplicates then PATH() returns the common parent_columnName value; however, if parent_columnName value is different for those duplicates then PATH() returns an error. In other words, if you have two listings for the same employee ID and they have the same manager ID, the PATH function returns the ID for that manager. However, if there are two identical employee IDs that have different manager IDs, the PATH function returns an error.

If ID_columnName is BLANK then PATH() returns BLANK.

If ID_columnName contains a vertical bar '|' then PATH() returns an error.

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models,
see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

Description

The following example creates a calculated column that lists all the managers for each employee.

Code

```
=PATH(Employee[EmployeeKey], Employee[ParentEmployeeKey])
```

Comments

PATHCONTAINS Function

Returns **TRUE** if the specified item exists within the specified path.

Syntax

PATHCONTAINS(<path>, <item>)

Parameters

Parameter	Description
	A string created as the result of evaluating a PATH function.
	A text expression to look for in the path result.

Return Value

A value of **TRUE** if item exists in path; otherwise **FALSE**.

Remarks

If item is an integer number it is converted to text and then the function is evaluated. If conversion fails then the function returns an error.

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models,
see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

Description

The following example creates a calculated column that takes a manager ID and checks a set of employees. If the manager ID is among the list of managers returned by the PATH function, the PATHCONTAINS function returns true; otherwise it returns false.

Code

```
=PATHCONTAINS(PATH(Employee[EmployeeKey], Employee[ParentEmployeeKey]),
"23")
```

Comments

PATHITEM Function

Returns the item at the specified position from a string resulting from evaluation of a PATH function. Positions are counted from left to right.

Syntax

PATHITEM(<path>, <position>[, <type>])

Parameters

Parameter	Description
	A text string in the form of the results of a PATH function.
	An integer expression with the position of the item to be returned.
	(Optional)An enumeration that defines the data type of the result:

Enumeration	Alternate Enumeration	Description
TEXT	0	Results are returned with the data type text. (default).
INTEGER	1	Results are returned as integers.

Return Value

The identifier returned by the PATH function at the specified position in the list of identifiers. Items returned by the PATH function are ordered by most distant to current.

Remarks

This function can be used to return a specific level from a hierarchy returned by a PATH function. For example, you could return just the skip-level managers for all employees.

If you specify a number for position that is less than one (1) or greater than the number of elements in path, the PATHITEM function returns BLANK

If type is not a valid enumeration element an error is returned.

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models,
see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

Description

The following example returns the third tier manager of the current employee; it takes the employee and manager IDs as the input to a PATH function that returns a string with the hierarchy of parents to current employee. From that string PATHITEM returns the third entry as an integer.

Code

```
=PATHITEM(PATH(Employee[EmployeeKey], Employee[ParentEmployeeKey]), 3,
1)
```

Comments

PATHITEMREVERSE Function

Returns the item at the specified position from a string resulting from evaluation of a PATH function. Positions are counted backwards from right to left.

Syntax

PATHITEMREVERSE(<path>, <position>[, <type>])

Parameters

Parameter	Description
	A text string resulting from evaluation of a PATH function.
	An integer expression with the position of the item to be returned. Position is counted backwards from right to left.
	(Optional) An enumeration that defines the data type of the result:

Enumeration	Alternate Enumeration	Description
TEXT	0	Results are

		returned with the data type of text. (default)
INTEGER	1	Results are returned with the data type of integer.

Return Value

The n-position ascendant in the given path, counting from current to the oldest.

Remarks

- This function can be used to get an individual item from a hierarchy resulting from a PATH function.
- This function reverses the standard order of the hierarchy, so that closest items are listed first, For example, if the PATh function returns a list of managers above an employee in a hierarchy, the PATHITEMREVERSE function returns the employee's immediate manager in position 2 because position 1 contains the employee's id.
- If the number specified for position is less than one (1) or greater than the number of elements in path, the PATHITEM function returns BLANK.
- If type is not a valid enumeration element an error is returned.

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models,
see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

Description

The following example takes an employee ID column as the input to a PATH function, and reverses the list of grandparent elements that are returned. The position specified is 3 and the return type is 1; therefore, the PATHITEMREVERSE function returns an integer representing the manager two levels up from the employee.

Code

```
=PATHITEMREVERSE(PATH(Employee[EmployeeKey],
Employee[ParentEmployeeKey]), 3, 1)
```

Comments

PATHLENGTH Function

Returns the number of parents to the specified item in a given PATH result, including self.

Syntax

PATHLENGTH(<path>)

Parameters

Parameter	Description
	A text expression resulting from evaluation of a PATH function.

Return Value

The number of items that are parents to the specified item in a given PATH result, including the specified item.

Remarks

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models,
see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

Description

The following example takes an employee ID as input to a PATH function and returns a list of the managers above that employee in the hierarchy, The PATHLENGTH function takes that result and counts the different levels of employees and managers, including the employee you started with.

Code

```
=PATHLENGTH(PATH(Employee[EmployeeKey], Employee[ParentEmployeeKey]))
```

Comments

USERNAME Function

Returns the domain name and username from the credentials given to the system at connection time

Syntax

USERNAME()

Parameters

Return Value

The username from the credentials given to the system at connection time

Exceptions

Remarks

Under Windows authentication, the returned string has the structure: `<domain-name>\<user-name>`

Example

Description

The following code verifies if the user login is part of the UsersTable.

Code

```
=IF(CONTAINS(UsersTable,UsersTable[login], USERNAME()), "Allowed",
BLANK())
```

Comments

Logical Functions

Logical functions act upon an expression to return information about the values or sets in the expression. For example, you can use the IF function to check the result of an expression and create conditional results.

In this Section

AND Function

FALSE Function

IF Function

IFERROR Function

NOT Function

OR Function

SWITCH Function

TRUE Function

Reference

Using DAX

Basic DAX Syntax

Related Sections

Date and Time Functions (DAX)

Aggregation Functions (DAX)

Logical Functions (DAX)

Filter and Value Functions (DAX)

Math and Trigonometric Functions (DAX)

See Also

Getting Started with Data Analysis Expressions (DAX)

AND Function

Checks whether both arguments are TRUE, and returns TRUE if both arguments are TRUE. Otherwise returns false.

Syntax

AND(<logical1>,<logical2>)

Parameters

Term	Definition
logical_1, **logical_2**	The logical values you want to test.

Return Value

Returns true or false depending on the combination of values that you test.

Remarks

The **AND** function in DAX accepts only two (2) arguments. If you need to perform an AND operation on multiple expressions, you can create a series of calculations or, better, use the AND operator (**&&**) to join all of them in a simpler expression.

Example

Description

The following formula shows the syntax of the AND function.

Code

```
=IF(AND(10 > 9, -10 < -1), "All true", "One or more false"
```

Comments

Because both conditions, passed as arguments, to the AND function are true, the formula returns "All True".

Example

Description

The following sample uses the AND function with nested formulas to compare two sets of calculations at the same time. For each product category, the formula determines if

the current year sales and previous year sales of the Internet channel are larger than the Reseller channel for the same periods. If both conditions are true, for each category the formula returns the value, "Internet hit".

AND function	**Column Labels**					
Row Labels	2005	2006	2007	2008		Grand Total
Bib-Shorts						
Bike Racks						
Bike Stands				Internet Hit		
Bottles and Cages				Internet Hit		
Bottom Brackets						
Brakes						
Caps						
Chains						
Cleaners						
Cranksets						
Derailleurs						
Fenders				Internet Hit		
Forks						
Gloves						
Handlebars						
Headsets						
Helmets						
Hydration Packs						
Jerseys						
Lights						

AND function	Column Labels					
Locks						
Mountain Bikes						
Mountain Frames						
Panniers						
Pedals						
Pumps						
Road Bikes						
Road Frames						
Saddles						
Shorts						
Socks						
Tights						
Tires and Tubes				Internet Hit		
Touring Bikes						
Touring Frames						
Vests						
Wheels						
Grand Total						

Code

```
= IF( AND(  SUM( 'InternetSales_USD'[SalesAmount_USD])
           >SUM('ResellerSales_USD'[SalesAmount_USD])
          , CALCULATE(SUM('InternetSales_USD'[SalesAmount_USD]),
PREVIOUSYEAR('DateTime'[DateKey] ))
           >CALCULATE(SUM('ResellerSales_USD'[SalesAmount_USD]),
PREVIOUSYEAR('DateTime'[DateKey] ))
          )
```

```
    , "Internet Hit"
    , ""
    )
```

Comments

See Also

Logical Functions

FALSE Function

Returns the logical value FALSE.

Syntax

FALSE()

Return Value

Always FALSE.

Remarks

The word FALSE is also interpreted as the logical value FALSE.

Example

Description

The formula returns the logical value FALSE when the value in the column, 'InternetSales_USD'[SalesAmount_USD], is less than or equal to 200000.

The following table shows the results when the example formula is used with 'ProductCategory'[ProductCategoryName] in Row Labels and 'DateTime'[CalendarYear] in Column Labels.

True-False	Column Labels					
Row Labels	2005	2006	2007	2008		Grand Total
Accessories	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
Bikes	TRUE	TRUE	TRUE	TRUE	FALSE	TRUE
Clothing	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE
Components	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Grand Total	TRUE	TRUE	TRUE	TRUE	FALSE	TRUE

Code

```
=IF(SUM('InternetSales_USD'[SalesAmount_USD]) >200000, TRUE(), false())
```

See Also

TRUE

NOT

IF

Function Reference

IF Function

Checks if a condition provided as the first argument is met. Returns one value if the condition is TRUE, and returns another value if the condition is FALSE.

Syntax

IF(logical_test>,<value_if_true>, value_if_false)

Parameters

Term	Definition
logical_test	Any value or expression that can be evaluated to TRUE or FALSE.
value_if_true	The value that is returned if the logical test is TRUE. If omitted, TRUE is returned.
value_if_false	The value that is returned if the logical test is FALSE. If omitted, FALSE is returned.

Return Value

Any type of value that can be returned by an expression.

Remarks

If the value of **value_if_true** or **value_if_false** is omitted, IF treats it as an empty string value ("").

If the value referenced in the expression is a column, IF returns the value that corresponds to the current row.

The IF function attempts to return a single data type in a column. Therefore, if the values returned by **value_if_true** and **value_if_false** are of different data types, the IF function will implicitly convert data types to accommodate both values in the column. For example, the formula `IF(<condition>,TRUE(),0)` returns a column of ones and zeros

and the results can be summed, but the formula `IF(<condition>,TRUE(),FALSE())` returns only logical values. For more information about implicit data type conversion, see Data Types in DAX.

Example

Description

The following example uses nested IF functions that evaluate the number in the column, Calls, from the table FactCallCenter in Adventure Works DW Multidimensional 2012 . The function assigns a label as follows: **low** if the number of calls is less than 200, **medium** if the number of calls is less than 300 but not less than 200, and **high** for all other values.

Code

```
=IF([Calls]<200,"low",IF([Calls]<300,"medium","high"))
```

Example

Description

The following example gets a list of cities that contain potential customers in the California area by using columns from the table ProspectiveBuyer in Adventure Works DW Multidimensional 2012 . Because the list is meant to plan for a campaign that will target married people or people with children at home, the condition in the IF function checks for the value of the columns [MaritalStatus] and [NumberChildrenAtHome], and outputs the city if either condition is met and if the customer is in California. Otherwise, it outputs the empty string.

Code

```
=IF([StateProvinceCode]= "CA" && ([MaritalStatus] = "M" ||
[NumberChildrenAtHome] >1),[City])
```

Comments

Note that parentheses are used to control the order in which the AND (&&) and OR (||) operators are used. Also note that no value has been specified for **value_if_false**. Therefore, the function returns the default, which is an empty string.

See Also

TRUE

FALSE

NOT

IF

Function Reference

IFERROR Function

Evaluates an expression and returns a specified value if the expression returns an error; otherwise returns the value of the expression itself.

Syntax

IFERROR(value, value_if_error)

Parameters

Term	Definition
value	Any value or expression.
value_if_error	Any value or expression.

Return Value

A scalar of the same type as **value**

Exceptions

Remarks

You can use the IFERROR function to trap and handle errors in an expression.

If **value** or **value_if_error** is an empty cell, IFERROR treats it as an empty string value ("").

The IFERROR function is based on the IF function, and uses the same error messages, but has fewer arguments. The relationship between the IFERROR function and the IF function as follows:

```
IFERROR(A,B) := IF(ISERROR(A), B, A)
```

Note that the values that are returned for A and B must be of the same data type; therefore, the column or expression used for **value** and the value returned for **value_if_error** must be the same data type.

Example

Description

The following example returns 9999 if the expression 25/0 evaluates to an error. If the expression returns a value other than error, that value is passed to the invoking expression.

Code

```
=IFERROR(25/0,9999)
```

Comments

See Also

Logical Functions

NOT Function

Changes FALSE to TRUE, or TRUE to FALSE.

Syntax

NOT(<logical>)

Parameters

Term	Definition
logical	A value or expression that can be evaluated to TRUE or FALSE.

Return Value

TRUE OR FALSE.

Example

Description

The following example retrieves values from the calculated column that was created to illustrate the IF function. For that example, the calculated column was named using the default name, **Calculated Column1**, and contains the following formula:

```
=IF([Orders]<300,"true","false")
```

The formula checks the value in the column, [Orders], and returns "true" if the number of orders is under 300.

Now create a new calculated column, **Calculated Column2**, and type the following formula.

Code

```
=NOT([CalculatedColumn1])
```

Comments

For each row in **Calculated Column1**, the values "true" and "false" are interpreted as the logical values TRUE or FALSE, and the NOT function returns the logical opposite of that value.

See Also

TRUE

FALSE

IF

Function Reference

OR Function

Checks whether one of the arguments is TRUE to return TRUE. The function returns FALSE if both arguments are FALSE.

Syntax

OR(<logical1>,<logical2>)

Parameters

Term	Definition
logical_1, **logical_2**	The logical values you want to test.

Return Value

A Boolean value. The value is TRUE if any of the two arguments is TRUE; the value is FALSE if both the arguments are FALSE.

Remarks

The **OR** function in DAX accepts only two (2) arguments. If you need to perform an OR operation on multiple expressions, you can create a series of calculations or, better, use the OR operator (**||**) to join all of them in a simpler expression.

The function evaluates the arguments until the first TRUE argument, then returns TRUE.

Example

Description

The following example shows how to use the OR function to obtain the sales people that belong to the Circle of Excellence. The Circle of Excellence recognizes those who have achieved more than a million dollars in Touring Bikes sales or sales of over two and a half million dollars in 2007.

SalesPersonFlag	True					
OR function	Column Labels					
Row Labels	2005	2006	2007	2008		Grand Total
Abbas, Syed E						
Alberts, Amy E						
Ansman-Wolfe, Pamela O						

SalesPersonFlag	True					
Blythe, Michael G	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence
Campbell, David R						
Carson, Jillian	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence
Ito, Shu K						
Jiang, Stephen Y						
Mensa-Annan, Tete A						
Mitchell, Linda C	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence
Pak, Jae B	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence
Reiter, Tsvi Michael						
Saraiva, José Edvaldo	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence
Tsoflias, Lynn N						
Valdez, Rachel B						
Vargas, Garrett R						
Varkey Chudukatil, Ranjit R						Circle of Excellence
Grand Total	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence	Circle of Excellence

Code

```
IF(   OR(   CALCULATE(SUM('ResellerSales_USD'[SalesAmount_USD]),
'ProductSubcategory'[ProductSubcategoryName]="Touring Bikes") > 1000000
         ,   CALCULATE(SUM('ResellerSales_USD'[SalesAmount_USD]),
'DateTime'[CalendarYear]=2007) > 2500000
         )
   , "Circle of Excellence"
   , ""
   )
```

Comments

See Also

Logical functions

SWITCH Function

Evaluates an expression against a list of values and returns one of multiple possible result expressions.

Syntax

SWITCH(<expression>, <value>, <result>[, <value>, <result>]…[, <else>])

Parameters

Parameter	Description
	Any DAX expression that returns a single scalar value, where the expression is to be evaluated multiple times (for each row/context).
	A constant value to be matched with the results of expression.
	Any scalar expression to be evaluated if the results of expression match the corresponding value.
	Any scalar expression to be evaluated if the result of expression doesn't match any of the value arguments.

Return Value

A scalar value coming from one of the result expressions, if there was a match with value, or from the else expression, if there was no match with any value.

Exceptions

Remarks

All result expressions and the else expression must be of the same data type.

Example

Description

The following example creates a calculated column of month names.

Code

```
=SWITCH([Month], 1, "January", 2, "February", 3, "March", 4, "April"
                , 5, "May", 6, "June", 7, "July", 8, "August"
                , 9, "September", 10, "October", 11, "November", 12,
"December"
                , "Unknown month number" )
```

Comments

TRUE Function

Returns the logical value TRUE.

Syntax

TRUE()

Return Value

Always TRUE.

Remarks

The word TRUE is also interpreted as the logical value TRUE.

Example

Description

The formula returns the logical value TRUE when the value in the column, 'InternetSales_USD'[SalesAmount_USD], is greater than 200000.

The following table shows the results when the example formula is used in a PivotTable, with 'ProductCategory'[ProductCategoryName] in Row Labels and 'DateTime'[CalendarYear] in Column Labels.

True-False	Column Labels					
Row Labels	2005	2006	2007	2008		Grand Total
Accessories	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
Bikes	TRUE	TRUE	TRUE	TRUE	FALSE	TRUE
Clothing	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE
Components	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Grand Total	TRUE	TRUE	TRUE	TRUE	FALSE	TRUE

Code

```
= IF(SUM('InternetSales_USD'[SalesAmount_USD]) >200000, TRUE(),
false())
```

See Also

FALSE

NOT

IF

Function Reference

Math and Trig Functions

The mathematical functions in Data Analysis Expressions (DAX) are very similar to the Excel mathematical and trigonometric functions. This section lists the mathematical functions provided by DAX.

The numeric data types used by DAX functions are described in the section, Data Types in DAX.

In this Section

ABS Function

CEILING Function

CURRENCY Function

EXP Function

FACT Function

FLOOR Function

INT Function
ISO.CEILING Function
LN Function
LOG Function
LOG10 Function
MOD Function
MROUND Function
PI Function
POWER Function
QUOTIENT Function
RAND Function (DAX)
RANDBETWEEN Function (DAX)
ROUND Function
ROUNDDOWN Function
ROUNDUP Function
SIGN Function
SQRT Function
SUM Function
SUMX Function
TRUNC Function

Reference

Using DAX
Basic DAX Syntax

Related Sections

Date and Time Functions (DAX)
Aggregation Functions (DAX)
Logical Functions (DAX)
Filter and Value Functions (DAX)
Math and Trigonometric Functions (DAX)

See Also

Getting Started with Data Analysis Expressions (DAX)

ABS Function

Returns the absolute value of a number.

Syntax

ABS(<number>)

Parameters

Term	Definition
number	The number for which you want the absolute value.

Return Value

A decimal number.

Remarks

The absolute value of a number is a decimal number, whole or decimal, without its sign. You can use the ABS function to ensure that only non-negative numbers are returned from expressions when nested in functions that require a positive number.

Example

Description

The following example returns the absolute value of the difference between the list price and the dealer price, which you might use in a new calculated column, **DealerMarkup**.

Code

```
=ABS([DealerPrice]-[ListPrice])
```

See Also

Scalar math and trig functions

SIGN function

CEILING Function

Rounds a number up, to the nearest integer or to the nearest multiple of significance.

Syntax

CEILING(<number>, <significance>)

Parameters

Term	Definition
number	The number you want to round, or a reference to a column that contains

Term	Definition
	numbers.
significance	The multiple of significance to which you want to round. For example, to round to the nearest integer, type 1.

Return Value

A number rounded as specified.

Remarks

There are two CEILING functions in DAX, with the following differences:

- The CEILING function emulates the behavior of the CEILING function in Excel.
- The ISO.CEILING function follows the ISO-defined behavior for determining the ceiling value.

The two functions return the same value for positive numbers, but different values for negative numbers. When using a positive multiple of significance, both CEILING and ISO.CEILING round negative numbers upward (toward positive infinity). When using a negative multiple of significance, CEILING rounds negative numbers downward (toward negative infinity), while ISO.CEILING rounds negative numbers upward (toward positive infinity).

The return type is usually of the same type of the significant argument, with the following exceptions:

- If the number argument type is currency, the return type is currency.
- If the significance argument type is Boolean, the return type is integer.
- If the significance argument type is non-numeric, the return type is real.

Example

Description

The following formula returns 4.45. This might be useful if you want to avoid using smaller units in your pricing. If an existing product is priced at $4.42, you can use CEILING to round prices up to the nearest unit of five cents.

Code

```
=CEILING(4.42,0.05)
```

Example

Description

The following formula returns similar results as the previous example, but uses numeric values stored in the column, **ProductPrice**.

Code

```
=CEILING([ProductPrice],0.05)
```

See Also

Math and Trig functions
FLOOR function
ISO.CEILING Function (DAX)
ROUNDUP function

CURRENCY Function

Evaluates the argument and returns the result as currency data type.

Syntax

CURRENCY(<value>)

Parameters

Parameter	Description
	Any DAX expression that returns a single scalar value where the expression is to be evaluated exactly once before all other operations.

Return Value

The value of the expression evaluated and returned as a currency type value.

Remarks

- The CURRENCY function rounds up the 5th significant decimal, in value, to return the 4th decimal digit; rounding up occurs if the 5th significant decimal is equal or larger than 5. For example, if value is 3.6666666666666 then converting to currency returns $3.6667; however, if value is 3.0123456789 then converting to currency returns $3.0123.
- If the data type of the expression is TrueFalse then CURRENCY(<TrueFalse>) will return $1.0000 for True values and $0.0000 for False values.
- If the data type of the expression is Text then CURRENCY(<Text>) will try to convert text to a number; if conversion succeeds the number will be converted to currency, otherwise an error is returned.
- If the data type of the expression is DateTime then CURRENCY(<DateTime>) will convert the datetime value to a number and that number to currency. DateTime values have an integer part that represents the number of days between the given date and 1900-03-01 and a fraction that represents the fraction of a day (where 12 hours or noon is 0.5 day). If the value of the expression is not a proper DateTime value an error is returned.

Example

Description

Convert number 1234.56 to currency data type.

Code

```
=CURRENCY(1234.56)
```

Comments

Returns the value $1234.5600.

EXP Function

Returns raised to the power of a given number. The constant e equals 2.71828182845904, the base of the natural logarithm.

Syntax

EXP(<number>)

Parameters

Term	Definition
number	The exponent applied to the base . The constant equals 2.71828182845904, the base of the natural logarithm.

Return Value

A decimal number.

Exceptions

Remarks

EXP is the inverse of LN, which is the natural logarithm of the given number.

To calculate powers of bases other than , use the exponentiation operator (^). For more information, see Operator Reference.

Example

Description

The following formula calculates raised to the power of the number contained in the column, `[Power]`.

Code

```
=EXP([Power])
```

See Also

Scalar math and trig functions
LN Function
EXP
LOG10
LOG

FACT Function

Returns the factorial of a number, equal to the series 1*2*3*...* , ending in the given number.

Syntax

FACT(<number>)

Parameters

Term	Definition
number	The non-negative number for which you want to calculate the factorial.

Return Value

A decimal number.

Remarks

If the number is not an integer, it is truncated and an error is returned. If the result is too large, an error is returned.

Example

Description

The following formula returns the factorial for the series of integers in the column, `[Values]`.

Code

```
=FACT([Values])
```

Comments

The following table shows the expected results:

Values	Results
0	1
1	1
2	2
3	6
4	24
5	120
170	7.257415615308E+306

See Also

Scalar math and trig functions

TRUNC

FLOOR Function

Rounds a number down, toward zero, to the nearest multiple of significance.

Syntax

FLOOR(<number>, <significance>)

Parameters

Term	Definition
number	The numeric value you want to round.
significance	The multiple to which you want to round. The arguments **number** and **significance** must either both be positive, or both be negative.

Return Value

A decimal number.

Remarks

If either argument is nonnumeric, FLOOR returns **#VALUE!** error value.

If number and significance have different signs, FLOOR returns the **#NUM!** error value.

Regardless of the sign of the number, a value is rounded down when adjusted away from zero. If the number is an exact multiple of significance, no rounding occurs.

Example

Description

The following formula takes the values in the [Total Product Cost] column from the table, InternetSales.and rounds down to the nearest multiple of .1.

Code

```
=FLOOR(InternetSales[Total Product Cost],.5)
```

Comments

The following table shows the expected results for some sample values.

Values	Expected Result
10.8423	10.8
8.0373	8
2.9733	2.9

See Also

Scalar math and trig functions

INT Function

Rounds a number down to the nearest integer.

Syntax

INT(<number>)

Parameters

Term	Definition
number	The number you want to round down to an integer

Return Value

A whole number.

Remarks

TRUNC and INT are similar in that both return integers. TRUNC removes the fractional part of the number. INT rounds numbers down to the nearest integer based on the value of the fractional part of the number. INT and TRUNC are different only when using negative numbers: `TRUNC(-4.3)` returns -4, but `INT(-4.3)` returns -5 because -5 is the lower number.

Example

Description

The following expression rounds the value to 1. If you use the ROUND function, the result would be 2.

Code

```
=INT(1.5)
```

See Also

math and trig functions

ROUND

ROUNDUP

ROUNDDOWN

MROUND

ISO.CEILING Function

Rounds a number up, to the nearest integer or to the nearest multiple of significance.

Syntax

ISO.CEILING(<number>[, <significance>])

Parameters

Term	Definition
number	The number you want to round, or a reference to a column that contains numbers.
significance	(optional) The multiple of significance to which you want to round. For example, to round to the nearest integer, type 1. If the unit of significance is not specified, the number is rounded up to the nearest integer.

Return Value

A number, of the same type as the number argument, rounded as specified.

Remarks

There are two CEILING functions in DAX, with the following differences:

- The CEILING function emulates the behavior of the CEILING function in Excel.
- The ISO.CEILING function follows the ISO-defined behavior for determining the ceiling value.

The two functions return the same value for positive numbers, but different values for negative numbers. When using a positive multiple of significance, both CEILING and ISO.CEILING round negative numbers upward (toward positive infinity). When using a negative multiple of significance, CEILING rounds negative numbers downward (toward negative infinity), while ISO.CEILING rounds negative numbers upward (toward positive infinity).

The result type is usually the same type of the significance used as argument with the following exceptions:

- If the first argument is of currency type then the result will be currency type.
- If the optional argument is not included the result is of integer type.
- If the significance argument is of Boolean type then the result is of integer type.
- If the significance argument is non-numeric type then the result is of real type.

Example: Positive Numbers

Description

The following formula returns 4.45. This might be useful if you want to avoid using smaller units in your pricing. If an existing product is priced at $4.42, you can use ISO.CEILING to round prices up to the nearest unit of five cents.

Code

```
=ISO.CEILING(4.42,0.05)
```

Example: Negative Numbers

Description

The following formula returns the ISO ceiling value of -4.40.

Code

```
=ISO.CEILING(-4.42,0.05)
```

See Also

Math and Trig functions

FLOOR function

CEILING Function (DAX)

ROUNDUP function

LN Function

Returns the natural logarithm of a number. Natural logarithms are based on the constant (2.71828182845904).

Syntax

LN(<number>)

Parameters

Term	Definition
number	The positive number for which you want the natural logarithm.

Return Value

A decimal number.

Remarks

LN is the inverse of the EXP function.

Example

Description

The following example returns the natural logarithm of the number in the column, `[Values]`.

Code

```
=LN([Values])
```

See Also

Scalar math and trig functions

EXP Function

LOG Function

Returns the logarithm of a number to the base you specify.

Syntax

LOG(<number>,<base>)

Parameters

Term	Definition
number	The positive number for which you want the logarithm.
base	The base of the logarithm. If omitted, the base is 10.

Return Value

A decimal number.

Remarks

You might receive an error if the value is too large to be displayed.

The function LOG10 is similar, but always returns the common logarithm, meaning the logarithm for the base 10.

Example

Description

The following formulas return the same result, 2.

Code

```
=LOG(100,10)
=LOG(100)
=LOG10(100)
```

See Also

Scalar math and trig functions

EXP

LOG10

LOG

LOG10 Function

Returns the base-10 logarithm of a number.

Syntax

LOG10(<number>)

Parameters

Term	Definition
number	A positive number for which you want the

Term	Definition
	base-10 logarithm.

Return Value

A decimal number.

Remarks

The LOG function lets you change the base of the logarithm, instead of using the base 10.

Example

Description

The following formulas return the same result, 2:

Code

```
=LOG(100,10)

=LOG(100)

=LOG10(100)
```

See Also

Scalar math and trig functions

EXP

LOG10

LOG

MOD Function

Returns the remainder after a number is divided by a divisor. The result always has the same sign as the divisor.

Syntax

MOD(<number>, <divisor>)

Parameters

Term	Definition
number	The number for which you want to find the remainder after the division is performed.
divisor	The number by which you want to divide.

Return Value

A whole number.

Remarks

If the divisor is 0 (zero), MOD returns an error. You cannot divide by 0.

The MOD function can be expressed in terms of the INT function:

Example

Description

The following formula returns 1, the remainder of 3 divided by 2.

Code

```
=MOD(3,2)
```

Example

Description

The following formula returns -1, the remainder of 3 divided by 2. Note that the sign is always the same as the sign of the divisor.

Code

```
=MOD(-3,-2)
```

See Also

math and trig functions

ROUND

ROUNDUP

ROUNDDOWN

MROUND

INT

MROUND Function

Returns a number rounded to the desired multiple.

Syntax

MROUND(<number>, <multiple>)

Parameters

Term	Definition
number	The number to round.
multiple	The multiple of significance to which you

Term	Definition
	want to round the number.

Return Value

A decimal number.

Remarks

MROUND rounds up, away from zero, if the remainder of dividing **number** by the specified **multiple** is greater than or equal to half the value of **multiple**.

Example: Decimal Places

Description

The following expression rounds 1.3 to the nearest multiple of .2. The expected result is 1.4.

Code

```
=MROUND(-1.3,0.2)
```

Example: Negative Numbers

Description

The following expression rounds -10 to the nearest multiple of -3. The expected result is -9.

Code

```
=MROUND(-10,-3)
```

Example: Error

Description

The following expression returns an error, because the numbers have different signs.

Code

```
=MROUND(5,-2)
```

See Also

math and trig functions

ROUND

ROUNDUP

ROUNDDOWN

MROUND

INT

PI Function

Returns the value of Pi, 3.14159265358979, accurate to 15 digits.

Syntax

PI()

Return Value

A decimal number with the value of Pi, 3.14159265358979, accurate to 15 digits.

Remarks

Pi is a mathematical constant. In DAX, Pi is represented as a real number accurate to 15 digits, the same as Excel.

Example

Description

The following formula calculates the area of a circle given the radius in the column, `[Radius]`.

Code

```
=PI()*([Radius]*2)
```

See Also

math and trig functions

POWER Function

Returns the result of a number raised to a power.

Syntax

POWER(<number>, <power>)

Parameters

Term	Definition
number	The base number, which can be any real number.
power	The exponent to which the base number is raised.

Return Value

A decimal number.

Example

Description

The following example returns 25.

Code

```
=POWER(5,2)
```

See Also

Scalar math and trig functions

QUOTIENT Function

Performs division and returns only the integer portion of the division result. Use this function when you want to discard the remainder of division.

Syntax

QUOTIENT(<numerator>, <denominator>)

Parameters

Term	Definition
numerator	The dividend, or number to divide.
denominator	The divisor, or number to divide by.

Return Value

A whole number.

Remarks

If either argument is non-numeric, QUOTIENT returns the **#VALUE!** error value.

You can use a column reference instead of a literal value for either argument. However, if the column that you reference contains a 0 (zero), an error is returned for the entire column of values.

Example

Description

The following formulas return the same result, 2.

Code

```
=QUOTIENT(5,2)
```

```
=QUOTIENT(10/2,2)
```

See Also

Scalar math and trig functions

RAND Function

Returns a random number greater than or equal to 0 and less than 1, evenly distributed. The number that is returned changes each time the cell containing this function is recalculated.

Syntax

RAND()

Return Value

A decimal number.

Remarks

In PowerPivot workbooks, recalculation depends on various factors, including whether the workbook is set to **Manual** or **Automatic** recalculation mode, and whether data has been refreshed. This is different from Microsoft Excel, where you can control when RAND generates a new random number by turning off recalculation.

For more information, see Data Refresh and Recalculation.

RAND and other volatile functions that do not have fixed values are not always recalculated. For example, execution of a query or filtering will usually not cause such functions to be re-evaluated. However, the results for these functions will be recalculated when the entire column is recalculated. These situations include refresh from an external data source or manual editing of data that causes re-evaluation of formulas that contain these functions.

Moreover, RAND is always recalculated if the function is used in the definition of a measure.

Also, in such contexts the RAND function cannot return a result of zero, to prevent errors such as division by zero.

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models,
see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

Description

To generate a random real number between two other numbers, you can use a formula like the following:

Code

```
= RAND()*(int1-int2)+int1
```

See Also

Scalar math and trig functions

Relational math and trig functions

RANDBETWEEN Function

Returns a random number in the range between two numbers you specify.

Syntax

RANDBETWEEN(<bottom>,<top>)

Parameters

Term	Definition
Bottom	The smallest integer the function will return.
Top	The largest integer the function will return.

Return Value

A whole number.

Remarks

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models,
see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

Description

The following formula returns a random number between 1 and 10.

Code

```
=RANDBETWEEN(1,10)
```

See Also

Scalar math and trig functions

Relational math and trig functions

ROUND Function

Rounds a number to the specified number of digits.

Syntax

ROUND(<number>, <num_digits>)

Parameters

Term	Definition
number	The number you want to round.
num_digits	The number of digits to which you want to round. A negative value rounds digits to the left of the decimal point; a value of zero rounds to the nearest integer.

Return Value

A decimal number.

Remarks

If **num_digits** is greater than 0 (zero), then number is rounded to the specified number of decimal places.

If **num_digits** is 0, the number is rounded to the nearest integer.

If **num_digits** is less than 0, the number is rounded to the left of the decimal point.

Related Functions

To always round up (away from zero), use the ROUNDUP function.

To always round down (toward zero), use the ROUNDDOWN function.

To round a number to a specific multiple (for example, to round to the nearest multiple of 0.5), use the MROUND function.

You can use the functions TRUNC and INT to obtain the integer portion of the number.

Example

Description

The following formula rounds 2.15 up, to one decimal place. The expected result is 2.2.

Code

```
=ROUND(2.15,1)
```

Example

Description

The following formula rounds 21.5 to one decimal place to the left of the decimal point. The expected result is 20.

Code

```
=ROUND(21.5,-1)
```

See Also

Math and Trig functions

ROUND

ROUNDDOWN

MROUND
INT
TRUNC

ROUNDDOWN Function

Rounds a number down, toward zero.

Syntax

ROUNDDOWN(<number>, <num_digits>)

Parameters

Term	Definition
number	A real number that you want rounded down.
num_digits	The number of digits to which you want to round. Negative rounds to the left of the decimal point; zero to the nearest integer.

Return Value

A decimal number.

Remarks

If **num_digits** is greater than 0 (zero), then the value in **number** is rounded down to the specified number of decimal places.

If **num_digits** is 0, then the value in **number** is rounded down to the nearest integer.

If **num_digits** is less than 0, then the value in **number** is rounded down to the left of the decimal point.

Related Functions

ROUNDDOWN behaves like ROUND, except that it always rounds a number down. The INT function also rounds down, but with INT the result is always an integer, whereas with ROUNDDOWN you can control the precision of the result.

Example

Description

The following example rounds 3.14159 down to three decimal places. The expected result is 3.141.

Code

```
=ROUNDDOWN(3.14159,3)
```

Example

Description

The following example rounds the value of 31415.92654 down to 2 decimal places to the left of the decimal. The expected result is 31400.

Code

```
=ROUNDDOWN(31415.92654, -2)
```

See Also

Math and Trig functions

ROUND

ROUNDUP

ROUNDDOWN

MROUND

INT

ROUNDUP Function

Rounds a number up, away from 0 (zero).

Syntax

ROUNDUP(<number>, <num_digits>)

Parameters

Term	Definition
number	A real number that you want to round up.
num_digits	The number of digits to which you want to round. A negative value for **num_digits** rounds to the left of the decimal point; if **num_digits** is zero or is omitted, **number** is rounded to the nearest integer.

Return Value

A decimal number.

Remarks

ROUNDUP behaves like ROUND, except that it always rounds a number up.

- If **num_digits** is greater than 0 (zero), then the number is rounded up to the specified number of decimal places.

- If **num_digits** is 0, then number is rounded up to the nearest integer.
- If **num_digits** is less than 0, then number is rounded up to the left of the decimal point.

Related Functions

ROUNDUP behaves like ROUND, except that it always rounds a number up.

Example

Description

The following formula rounds Pi to four decimal places. The expected result is 3.1416.

Code

```
=ROUNDUP(PI(),4)
```

Example: Decimals as Second Argument

Description

The following formula rounds 1.3 to the nearest multiple of 0.2. The expected result is 1.4.

Code

```
=ROUNDUP(1.3,0.2)
```

Example: Negative Number as Second Argument

Description

The following formula rounds the value in the column, **FreightCost**, with the expected results shown in the following table:

Code

```
=ROUNDUP([Values],-1)
```

Comments

When **num_digits** is less than zero, the number of places to the left of the decimal sign is increased by the value you specify.

FreightCost	Expected Result
13.25	20
2.45	10
25.56	30
1.34	10
345.01	350

See Also

[math and trig functions]
[ROUND]
[ROUNDDOWN]
[MROUND]
[INT]

SIGN Function

Determines the sign of a number, the result of a calculation, or a value in a column. The function returns 1 if the number is positive, 0 (zero) if the number is zero, or -1 if the number is negative.

Syntax

SIGN(<number>)

Parameters

Term	Definition
number	Any real number, a column that contains numbers, or an expression that evaluates to a number.

Return Value

A whole number. The possible return values are 1, 0, and -1.

Return Value	Description
1	The number is positive
0	The number is zero
-1	The number is negative

Example

Description

The following formula returns the sign of the result of the expression that calculates sale price minus cost.

Code

```
=SIGN( ([Sale Price] - [Cost]) )
```

See Also

math and trig functions

SQRT Function

Returns the square root of a number.

Syntax

SQRT(<number>)

Parameters

Term	Definition
number	The number for which you want the square root, a column that contains numbers, or an expression that evaluates to a number.

Return Value

A decimal number.

Remarks

If the number is negative, the SQRT function returns an error.

Example

Description

The following formula returns 5.

Code

```
=SQRT(25)
```

See Also

Scalar math and trig functions

SUM Function

Adds all the numbers in a column.

Syntax

SUM(<column>)

Parameters

Term	Definition
column	The column that contains the numbers to sum.

Return Value

A decimal number.

Remarks

If any rows contain non-numeric values, blanks are returned.

If you want to filter the values that you are summing, you can use the SUMX function and specify an expression to sum over.

Example

Description

The following example adds all the numbers that are contained in the column, Amt, from the table, Sales.

Code

```
=SUM(Sales[Amt])
```

See Also

SUMX

Aggregation functions

SUMX Function

Returns the sum of an expression evaluated for each row in a table.

Syntax

SUMX(<table>, <expression>)

Parameters

Term	Definition
table	The table containing the rows for which the expression will be evaluated.
expression	The expression to be evaluated for each row of the table.

Return Value

A decimal number.

Remarks

The SUMX function takes as its first argument a table, or an expression that returns a table. The second argument is a column that contains the numbers you want to sum, or an expression that evaluates to a column.

Only the numbers in the column are counted. Blanks, logical values, and text are ignored.

To see some more complex examples of SUMX in formulas, see ALL and CALCULATETABLE.

Example

Description

The following example first filters the table, InternetSales, on the expression, `ShippingTerritoryID = 5`, and then returns the sum of all values in the column, Freight. In other words, the expression returns the sum of freight charges for only the specified sales area.

Code

```
=SUMX(FILTER(InternetSales,
InternetSales[SalesTerritoryID]=5),[Freight])
```

Comments

If you do not need to filter the column, use the SUM function. The SUM function is similar to the Excel function of the same name, except that it takes a column as a reference.

See Also

SUM

Aggregation functions

TRUNC Function

Truncates a number to an integer by removing the decimal, or fractional, part of the number.

Syntax

TRUNC(<number>,<num_digits>)

Parameters

Term	Definition
number	The number you want to truncate.
num_digits	A number specifying the precision of the

Term	Definition
	truncation; if omitted, 0 (zero)

Return Value

A whole number.

Remarks

TRUNC and INT are similar in that both return integers. TRUNC removes the fractional part of the number. INT rounds numbers down to the nearest integer based on the value of the fractional part of the number. INT and TRUNC are different only when using negative numbers: `TRUNC(-4.3)` returns -4, but `INT(-4.3)` returns -5 because -5 is the smaller number.

Example

Description

The following formula returns 3, the integer part of pi.

Code

```
=TRUNC(PI())
```

Example

Description

The following formula returns -8, the integer part of -8.9.

Code

```
=TRUNC(-8.9)
```

See Also

math and trig functions

ROUND

ROUNDUP

ROUNDDOWN

MROUND

INT

Statistical Functions

Data Analysis Expressions (DAX) provides many functions for creating aggregations such as sums, counts, and averages. These functions are very similar to aggregation functions used by Microsoft Excel. This section lists the statistical and aggregation functions provided in DAX.

In this Section

ADDCOLUMNS
AVERAGE
AVERAGEA
AVERAGEX
COUNT
COUNTA
COUNTAX
COUNTBLANK
COUNTROWS
COUNTX
CROSSJOIN
DISTINCTCOUNT
GENERATE
GENERATEALL
MAX
MAXA
MAXX
MINA
MIN
MINX
RANK.EQ
RANKX
ROW
STDEV.P
STDEV.S
STDEVX.P
STDEVX.S
SUMMARIZE
TOPN
VAR.P
VAR.S
VARX.P
VARX.S

Reference

Using DAX

Basic DAX Syntax

Related Sections

Date and Time Functions (DAX)

Aggregation Functions (DAX)

Logical Functions (DAX)

Filter and Value Functions (DAX)

Math and Trigonometric Functions (DAX)

See Also

Getting Started with Data Analysis Expressions (DAX)

ADDCOLUMNS Function

Adds calculated columns to the given table or table expression.

Syntax

ADDCOLUMNS(<table>, <name>, <expression>[, <name>, <expression>]...)

Parameters

Parameter	Description
table	Any DAX expression that returns a table of data.
name	The name given to the column, enclosed in double quotes.
expression	Any DAX expression that returns a scalar expression, evaluated for each row of table.

Return Value

A table with all its original columns and the added ones.

Remarks

Example

Description

The following example returns an extended version of the Product Category table that includes total sales values from the reseller channel and the internet sales.

Code

```
ADDCOLUMNS(ProductCategory,
```

```
                , "Internet Sales",
SUMX(RELATEDTABLE(InternetSales_USD),
InternetSales_USD[SalesAmount_USD])
                , "Reseller Sales",
SUMX(RELATEDTABLE(ResellerSales_USD),
ResellerSales_USD[SalesAmount_USD]))
```

Comments

The following table shows a preview of the data as it would be received by any function expecting to receive a table:

ProductCategory[ProductCategoryName]	ProductCategory[ProductCategoryAlternateKey]	ProductCategory[ProductCategoryKey]	[Internet Sales]	[Reseller Sales]
Bikes	1	1	25107749.77	63084675.04
Components	2	2		11205837.96
Clothing	3	3	306157.5829	1669943.267
Accessories	4	4	640920.1338	534301.9888

AVERAGE Function

Returns the average (arithmetic mean) of all the numbers in a column.

Syntax

AVERAGE(<column>)

Parameters

Term	Definition
column	The column that contains the numbers for which you want the average.

Return Value

Returns a decimal number that represents the arithmetic mean of the numbers in the column.

Remarks

This function takes the specified column as an argument and finds the average of the values in that column. If you want to find the average of an expression that evaluates to a set of numbers, use the AVERAGEX function instead.

Nonnumeric values in the column are handled as follows:

- If the column contains text, no aggregation can be performed, and the functions returns blanks.
- If the column contains logical values or empty cells, those values are ignored.
- Cells with the value zero are included.
- When you average cells, you must keep in mind the difference between an empty cell and a cell that contains the value 0 (zero). When a cell contains 0, it is added to the sum of numbers and the row is counted among the number of rows used as the divisor. However, when a cell contains a blank, the row is not counted.

Whenever there are no rows to aggregate, the function returns a blank. However, if there are rows, but none of them meet the specified criteria, the function returns 0. Excel also returns a zero if no rows are found that meet the conditions.

Example

Description

The following formula returns the average of the values in the column, ExtendedSalesAmount, in the table, InternetSales.

Code

```
=AVERAGE(InternetSales[ExtendedSalesAmount])
```

Related Functions

The AVERAGEX function can take as its argument an expression that is evaluated for each row in a table. This enables you to perform calculations and then take the average of the calculated values.

The AVERAGEA function takes a column as its argument, but otherwise is like the Excel function of the same name. By using the AVERAGEA function, you can calculate a mean on a column that contains empty values.

See Also

AVERAGEA
AVERAGEX
Aggregation functions

AVERAGEA Function

Returns the average (arithmetic mean) of the values in a column. Handles text and non-numeric values.

Syntax

AVERAGEA(<column>)

Parameters

Term	Definition
column	A column that contains the values for which you want the average.

Return Value

A decimal number.

Remarks

The AVERAGEA function takes a column and averages the numbers in it, but also handles non-numeric data types according to the following rules:

- Values that evaluates to TRUE count as 1.
- Values that evaluate to FALSE count as 0 (zero).
- Values that contain non-numeric text count as 0 (zero).
- Empty text ("") counts as 0 (zero).

If you do not want to include logical values and text representations of numbers in a reference as part of the calculation, use the AVERAGE function.

Whenever there are no rows to aggregate, the function returns a blank. However, if there are rows, but none of them meet the specified criteria, the function returns 0. Microsoft Excel also returns a zero if no rows are found that meet the conditions.

Example

Description

The following example returns the average of non-blank cells in the referenced column, given the following table. If you used the AVERAGE function, the mean would be 21/2; with the AVERAGEA function, the result is 22/5.

Transaction ID	Amount	Result
0000123	1	Counts as 1
0000124	20	Counts as 20
0000125	n/a	Counts as 0
0000126		Counts as 0
0000126	TRUE	Counts as 1

Code

```
=AVERAGEA([Amount])
```

See Also

AVERAGE

AVERAGEX

Aggregation functions

AVERAGEX Function

Calculates the average (arithmetic mean) of a set of expressions evaluated over a table.

Syntax

AVERAGEX(<table>,<expression>)

Parameters

Term	Definition
table	Name of a table, or an expression that specifies the table over which the aggregation can be performed.
expression	An expression with a scalar result, which will be evaluated for each row of the table in the first argument.

Return Value

A decimal number.

Remarks

The AVERAGEX function enables you to evaluate expressions for each row of a table, and then take the resulting set of values and calculate its arithmetic mean. Therefore, the function takes a table as its first argument, and an expression as the second argument.

In all other respects, AVERAGEX follows the same rules as AVERAGE. You cannot include non-numeric or null cells. Both the table and expression arguments are required.

When there are no rows to aggregate, the function returns a blank. When there are rows, but none of them meet the specified criteria, then the function returns 0.

Example

Description

The following example calculates the average freight and tax on each order in the InternetSales table, by first summing Freight plus TaxAmt in each row, and then averaging those sums.

Code

```
=AVERAGEX(InternetSales, InternetSales[Freight]+ InternetSales[TaxAmt])
```

Comments

If you use multiple operations in the expression used as the second argument, you must use parentheses to control the order of calculations. For more information, see Basic DAX Syntax.

See Also

AVERAGE

AVERAGEA

Aggregation functions

COUNT Function

The COUNT function counts the number of cells in a column that contain numbers.

Syntax

COUNT(<column>)

Parameters

Term	Definition
column	The column that contains the numbers to be counted

Return Value

A whole number.

Remarks

The only argument allowed to this function is a column. You can use columns containing any type of data, but only numbers are counted. The COUNT function counts rows that contain the following kinds of values:

- Numbers
- Dates

If the row contains text that cannot be translated into a number, the row is not counted.

When the function finds no rows to count, it returns a blank. When there are rows, but none of them meet the specified criteria, then the function returns 0.

Example

Description

The following example shows how to count the number of numeric values in the column, ShipDate.

Code

```
=COUNT([ShipDate])
```

Comments

To count logical values or text, use the COUNTA or COUNTAX functions.

See Also

COUNT

COUNTA

COUNTAX

COUNTX

Aggregation functions

COUNTA Function

The COUNTA function counts the number of cells in a column that are not empty. It counts not just rows that contain numeric values, but also rows that contain nonblank values, including text, dates, and logical values.

Syntax

COUNTA(<column>)

Parameters

Term	Definition
column	The column that contains the values to be counted

Return Value

A whole number.

Remarks

If you do not need to count cells that contain logical values or text (in other words, if you want to count only cells that contain numbers), use the COUNT or COUNTX functions.

When the function does not find any rows to count, the function returns a blank. When there are rows, but none of them meet the specified criteria, then the function returns 0.

Example

Description

The following example returns all rows in the `Reseller` table that have any kind of value in the column that stores phone numbers. Because the table name does not contain any spaces, the quotation marks are optional.

Code

```
=COUNTA('Reseller'[Phone])
```

See Also

COUNT

COUNTA

COUNTAX

COUNTX

Aggregation functions

COUNTAX Function

The COUNTAX function counts nonblank results when evaluating the result of an expression over a table. That is, it works just like the COUNTA function, but is used to iterate through the rows in a table and count rows where the specified expressions results in a nonblank result.

Syntax

COUNTAX(<table>,<expression>)

Parameters

Term	Definition
table	The table containing the rows for which the expression will be evaluated.
expression	The expression to be evaluated for each

Term	Definition
	row of the table.

Return Value

A whole number.

Remarks

Like the COUNTA function, the COUNTAX function counts cells containing any type of information, including other expressions.

For example, if the column contains an expression that evaluates to an empty string, the COUNTAX function treats that result as nonblank. Usually the COUNTAX function does not count empty cells but in this case the cell contains a formula, so it is counted.

If you do not need to count logical values or text, use the COUNTX function instead.

Whenever the function finds no rows to aggregate, the function returns a blank. However, if there are rows, but none of them meet the specified criteria, the function returns 0. Microsoft Excel also returns 0 if no rows are found that meet the condition.

Example

Description

The following example counts the number of nonblank rows in the column, Phone, using the table that results from filtering the Reseller table on [Status] = **Active**.

Code

```
=COUNTAX(FILTER('Reseller',[Status]="Active"),[Phone])
```

See Also

COUNT

COUNTA

COUNTAX

COUNTX

Aggregation functions

COUNTBLANK Function

Counts the number of blank cells in a column.

Syntax

COUNTBLANK(<column>)

Parameters

Term	Definition
column	The column that contains the blank cells to be counted.

Return Value

A whole number. If no rows are found that meet the condition, blanks are returned.

Remarks

The only argument allowed to this function is a column. You can use columns containing any type of data, but only blank cells are counted. Cells that have the value zero (0) are not counted, as zero is considered a numeric value and not a blank.

Whenever there are no rows to aggregate, the function returns a blank. However, if there are rows, but none of them meet the specified criteria, the function returns 0. Microsoft Excel also returns a zero if no rows are found that meet the conditions.

In other words, if the COUNTBLANK function finds no blanks, the result will be zero, but if there are no rows to check, the result will be blank.

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models,
see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

Description

The following example shows how to count the number of rows in the table Reseller that have blank values for BankName.

Code

```
=COUNTBLANK(Reseller[BankName])
```

Comments

To count logical values or text, use the COUNTA or COUNTAX functions.

See Also

COUNT

COUNTA

COUNTAX

COUNTX

Aggregation functions

COUNTROWS Function

The COUNTROWS function counts the number of rows in the specified table, or in a table defined by an expression.

Syntax

COUNTROWS(<table>)

Parameters

Term	Definition
table	The name of the table that contains the rows to be counted, or an expression that returns a table.

Return Value

A whole number.

Remarks

This function can be used to count the number of rows in a base table, but more often is used to count the number of rows that result from filtering a table, or applying context to a table.

Whenever there are no rows to aggregate, the function returns a blank. However, if there are rows, but none of them meet the specified criteria, the function returns 0. Microsoft Excel also returns a zero if no rows are found that meet the conditions.

Example

Description

The following example shows how to count the number of rows in the table Orders. The expected result is 52761.

Code

```
=COUNTROWS('Orders')
```

Example

Description

The following example demonstrates how to use COUNTROWS with a row context. In this scenario, there are two sets of data that are related by order number. The table Reseller contains one row for each reseller; the table ResellerSales contains multiple rows for each order, each row containing one order for a particular reseller. The tables are connected by a relationship on the column, ResellerKey.

The formula gets the value of ResellerKey and then counts the number of rows in the related table that have the same reseller ID. The result is output in the column, **CalculatedColumn1**.

Code

```
=COUNTROWS(RELATEDTABLE(ResellerSales))
```

Comments

The following table shows a portion of the expected results:

ResellerKey	CalculatedColumn1
1	73
2	70
3	394

See Also

COUNT
COUNTA
COUNTAX
COUNTX
Aggregation functions

COUNTX Function

Counts the number of rows that contain a number or an expression that evaluates to a number, when evaluating an expression over a table.

Syntax

COUNTX(<table>,<expression>)

Parameters

Term	Definition
table	The table containing the rows to be counted.
expression	An expression that returns the set of values that contains the values you want to count.

Return Value

An integer.

Remarks

The COUNTX function takes two arguments. The first argument must always be a table, or any expression that returns a table. The second argument is the column or expression that is searched by COUNTX.

The COUNTX function counts only numeric values, or dates. Arguments that are logical values or text that cannot be translated into numbers are not counted. If the function finds no rows to count, it returns a blank. When there are rows, but none meets the specified criteria, then the function returns 0.

If you want to count logical values, or text, use the COUNTA or COUNTAX functions.

Example

Description

The following formula returns a count of all rows in the Product table that have a list price.

Code

```
=COUNTX(Product,[ListPrice])
```

Example

Description

The following formula illustrates how to pass a filtered table to COUNTX for the first argument. The formula uses a filter expression to get only the rows in the Product table that meet the condition, ProductSubCategory = "Caps", and then counts the rows in the resulting table that have a list price. The FILTER expression applies to the table Products but uses a value that you look up in the related table, ProductSubCategory.

Code

```
=COUNTX(FILTER(Product,RELATED(ProductSubcategory[EnglishProductSubcategoryName])="Caps", Product[ListPrice])
```

See Also

COUNT

COUNTA

COUNTAX

COUNTX

Aggregation functions

CROSSJOIN Function

Returns a table that contains the Cartesian product of all rows from all tables in the arguments. The columns in the new table are all the columns in all the argument tables.

Syntax

CROSSJOIN(<table>, <table>[, <table>]…)

Parameters

Parameter	Description
	Any DAX expression that returns a table of data

Return Value

A table that contains the Cartesian product of all rows from all tables in the arguments.

Remarks

- Column names from table arguments must all be different in all tables or an error is returned.
- The total number of rows returned by CROSSJOIN() is equal to the product of the number of rows from all tables in the arguments; also, the total number of columns in the result table is the sum of the number of columns in all tables. For example, if **TableA** has **rA** rows and **cA** columns, and **TableB** has **rB** rows and **cB** columns, and **TableC** has **rC** rows and **cC** column; then, the resulting table has **rA × rb × rC** rows and **cA + cB + cC** columns.

Example

Description

The following example shows the results of applying CROSSJOIN() to two tables: **Colors** and **Stationery**.

The table **Colors** contains colors and patterns:

Color	Pattern
Red	Horizontal Stripe
Green	Vertical Stripe
Blue	Crosshatch

The table **Stationery** contains fonts and presentation:

Font	Presentation
serif	embossed
sans-serif	engraved

The expression to generate the cross join is presented below:

Code

```
CROSSJOIN( Colors, Stationery)
```

Comments

When the above expression is used wherever a table expression is expected, the results of the expression would be as follows:

Red	Horizontal Stripe	serif	embossed
Green	Vertical Stripe	serif	embossed
Blue	Crosshatch	serif	embossed
Red	Horizontal Stripe	sans-serif	engraved
Green	Vertical Stripe	sans-serif	engraved
Blue	Crosshatch	sans-serif	engraved

DISTINCTCOUNT Function

The DISTINCTCOUNT function counts the number of different cells in a column of numbers.

Syntax

DISTINCTCOUNT(<column>)

Parameters

Parameter	Description
	The column that contains the numbers to be counted

Return Value

The number of distinct values in column.

Remarks

The only argument allowed to this function is a column. You can use columns containing any type of data. When the function finds no rows to count, it returns a BLANK, otherwise it returns the count of distinct values.

Example

Description

The following example shows how to count the number of distinct sales orders in the column ResellerSales_USD[SalesOrderNumber].

Code

```
=DISTINCTCOUNT(ResellerSales_USD[SalesOrderNumber])
```

Comments

Using the above measure in a table with calendar year in the side and product category on top gives the following results:

Distinct Reseller Orders count	**Column Labels**					
Row Labels	**Accessories**	**Bikes**	**Clothing**	**Components**		**Grand Total**
2005	135	345	242	205		366
2006	356	850	644	702		1015
2007	531	1234	963	1138		1521
2008	293	724	561	601		894
					1	1
Grand Total	**1315**	**3153**	**2410**	**2646**	**1**	**3797**

In the above example the user should be able to note that the rows Grand Total numbers do not add up, this happens because the same order might contain line items, in the same order, from different product categories.

See Also

COUNT

COUNTA

COUNTAX

COUNTX

Aggregation functions

GENERATE Function

Returns a table with the Cartesian product between each row in table1 and the table that results from evaluating table2 in the context of the current row from table1.

Syntax

GENERATE(<table1>, <table2>)

Parameters

Parameter	Description
	Any DAX expression that returns a table.
	Any DAX expression that returns a table.

Return Value

A table with the Cartesian product between each row in table1 and the table that results from evaluating table2 in the context of the current row from table1

Remarks

- If the evaluation of table2 for the current row in table1 returns an empty table, then the result table will not contain the current row from table1. This is different than GENERATEALL() where the current row from table1 will be included in the results and columns corresponding to table2 will have null values for that row.
- All column names from table1 and table2 must be different or an error is returned.

Example

Description

In the following example the user wants a summary table of the sales by Region and Product Category for the Resellers channel, like the following table:

SalesTerritory[SalesTerritoryGroup]	ProductCategory[ProductCategoryName]	[Reseller Sales]
Europe	Accessories	$ 142,227.27
Europe	Bikes	$ 9,970,200.44
Europe	Clothing	$ 365,847.63
Europe	Components	$ 2,214,440.19
North America	Accessories	$

		379,305.15
North America	Bikes	$ 52,403,796.85
North America	Clothing	$ 1,281,193.26
North America	Components	$ 8,882,848.05
Pacific	Accessories	$ 12,769.57
Pacific	Bikes	$ 710,677.75
Pacific	Clothing	$ 22,902.38
Pacific	Components	$ 108,549.71

The following code produces the above table:

Code

```
GENERATE(
SUMMARIZE(SalesTerritory, SalesTerritory[SalesTerritoryGroup])
,SUMMARIZE(ProductCategory
, [ProductCategoryName]
, "Reseller Sales", SUMX(RELATEDTABLE(ResellerSales_USD),
ResellerSales_USD[SalesAmount_USD])
)
)
```

Comments

1. The first SUMMARIZE statement, `SUMMARIZE(SalesTerritory, SalesTerritory[SalesTerritoryGroup])`, produces a table of territory groups, where each row is a territory group, as shown below:

SalesTerritory[SalesTerritoryGroup]
North America
Europe
Pacific
NA

2. The second SUMMARIZE statement, `SUMMARIZE(ProductCategory, [ProductCategoryName], "Reseller Sales", SUMX(RELATEDTABLE(ResellerSales_USD), ResellerSales_USD[SalesAmount_USD]))`, produces a table of Product Category groups with the Reseller sales for each group, as shown below:

ProductCategory[ProductCategoryName]	[Reseller Sales]
Bikes	$ 63,084,675.04
Components	$ 11,205,837.96
Clothing	$ 1,669,943.27
Accessories	$ 534,301.99

3. However, when you take the above table and evaluate it under the context of each row from the territory groups table, you obtain different results for each territory.

GENERATEALL Function

Returns a table with the Cartesian product between each row in table1 and the table that results from evaluating table2 in the context of the current row from table1.

Syntax

GENERATEALL(<table1>, <table2>)

Parameters

Parameter	Description
	Any DAX expression that returns a table.
	Any DAX expression that returns a table.

Return Value

A table with the Cartesian product between each row in table1 and the table that results from evaluating table2 in the context of the current row from table1

Remarks

- If the evaluation of table2 for the current row in table1 returns an empty table, then the current row from table1 will be included in the results and columns corresponding to table2 will have null values for that row. This is different than GENERATE() where the current row from table1 will **not** be included in the results.
- All column names from table1 and table2 must be different or an error is returned.

Example

Description

In the following example, the user wants a summary table of the sales by Region and Product Category for the Resellers channel, like the following table:

SalesTerritory[SalesTerritoryGroup]	ProductCategory[ProductCategoryName]	[Reseller Sales]
Europe	Accessories	$ 142,227.27
Europe	Bikes	$ 9,970,200.44
Europe	Clothing	$ 365,847.63
Europe	Components	$ 2,214,440.19
NA	Accessories	
NA	Bikes	
NA	Clothing	
NA	Components	
North America	Accessories	$ 379,305.15
North America	Bikes	$ 52,403,796.85
North America	Clothing	$ 1,281,193.26

SalesTerritory[SalesTerritoryGroup]	ProductCategory[ProductCategoryName]	[Reseller Sales]
North America	Components	$ 8,882,848.05
Pacific	Accessories	$ 12,769.57
Pacific	Bikes	$ 710,677.75
Pacific	Clothing	$ 22,902.38
Pacific	Components	$ 108,549.71

The following code produces the above table:

Code

```
GENERATEALL(
SUMMARIZE(SalesTerritory, SalesTerritory[SalesTerritoryGroup])
,SUMMARIZE(ProductCategory
, [ProductCategoryName]
, "Reseller Sales", SUMX(RELATEDTABLE(ResellerSales_USD),
ResellerSales_USD[SalesAmount_USD])
)
)
```

Comments

1. The first SUMMARIZE produces a table of territory groups, where each row is a territory group, like those listed below:

SalesTerritory[SalesTerritoryGroup]
North America
Europe
Pacific
NA

2. The second SUMMARIZE produces a table of Product Category groups with the Reseller sales for each group, as shown below:

ProductCategory[ProductCategoryName]	[Reseller Sales]
Bikes	$ 63,084,675.04
Components	$ 11,205,837.96
Clothing	$ 1,669,943.27
Accessories	$ 534,301.99

3. However, when you take the above table and evaluate the table under the context of each row from the territory groups table, you obtain different results for each territory.

MAX Function

Returns the largest numeric value in a column.

Syntax

MAX(<column>)

Parameters

Term	Definition
column	The column in which you want to find the largest numeric value.

Property Value/Return Value

A decimal number.

Remarks

The MAX function takes as an argument a column that contains numeric values. If the column contains no numbers, MAX returns a blank. If you want to evaluate values that are not numbers, use the MAXA function.

Example

Description

The following example returns the largest value found in the ExtendedAmount column of the InternetSales table.

Code

```
=MAX(InternetSales[ExtendedAmount])
```

See Also

MAX

MAXA

MAXX

Aggregation functions

MAXA Function

Returns the largest value in a column. Logical values and blanks are counted.

Syntax

MAXA(<column>)

Parameters

Term	Definition
column	The column in which you want to find the largest value.

Return Value

A decimal number.

Remarks

The MAXA function takes as argument a column, and looks for the largest value among the following types of values:

- Numbers
- Dates
- Logical values, such as TRUE and FALSE. Rows that evaluate to TRUE count as 1; rows that evaluate to FALSE count as 0 (zero).

Empty cells are ignored. If the column contains no values that can be used, MAXA returns 0 (zero).

If you do not want to include logical values and blanks as part of the calculation, use the MAX function.

Example

Description

The following example returns the greatest value from a calculated column, named **ResellerMargin**, that computes the difference between list price and reseller price.

Code

```
=MAXA([ResellerMargin])
```

Example

Description

The following example returns the largest value from a column that contains dates and times. Therefore, this formula gets the most recent transaction date.

Code

```
=MAXA([TransactionDate])
```

See Also

MAX

MAXA

MAXX

Aggregation functions

MAXX Function

Evaluates an expression for each row of a table and returns the largest numeric value.

Syntax

MAXX(<table>,<expression>)

Parameters

Term	Definition
table	The table containing the rows for which the expression will be evaluated.
expression	The expression to be evaluated for each row of the table.

Return Value

A decimal number.

Remarks

The **table** argument to the MAXX function can be a table name, or an expression that evaluates to a table. The second argument indicates the expression to be evaluated for each row of the table.

Of the values to evaluate, only the following are counted:

- Numbers. If the expression does not evaluate to a number, MAXX returns 0 (zero).
- Dates.

Empty cells, logical values, and text values are ignored. If you want to include non-numeric values in the formula, use the MAXA function.

If a blank cell is included in the column or expression, MAXX returns an empty column.

Example

Description

The following formula uses an expression as the second argument to calculate the total amount of taxes and shipping for each order in the table, InternetSales. The expected result is 375.7184.

Code

```
=MAXX(InternetSales, InternetSales[TaxAmt]+ InternetSales[Freight])
```

Example

Description

The following formula first filters the table InternetSales, by using a FILTER expression, to return a subset of orders for a specific sales region, defined as [SalesTerritory] = 5. The MAXX function then evaluates the expression used as the second argument for each row of the filtered table, and returns the highest amount for taxes and shipping for just those orders. The expected result is 250.3724.

Code

```
=MAXX(FILTER(InternetSales,[SalesTerritoryCode]="5"),
InternetSales[TaxAmt]+ InternetSales[Freight])
```

See Also

MAX

MAXA

MAXX

Aggregation functions

MIN Function

Returns the smallest numeric value in a column. Ignores logical values and text.

Syntax

MIN(<column>)

Parameters

Term	Definition
column	The column in which you want to find the smallest numeric value.

Return Value

A decimal number.

Remarks

The MIN function takes a column as an argument, and returns the smallest numeric value in the column. The following types of values in the columns are counted:

- Numbers
- Dates
- If the column contains no numerical data, MIN returns blanks.

Empty cells, logical values, and text are ignored. If you want to include logical values and text representations of numbers in a reference as part of the calculation, use the MINA function.

Example

Description

The following example returns the smallest value from the calculated column, ResellerMargin.

Code

```
=MIN([ResellerMargin])
```

Example

Description

The following example returns the smallest value from a column that contains dates and times, TransactionDate. This formula therefore returns the date that is earliest.

Code

```
=MIN([TransactionDate])
```

See Also

MIN

MINA

MINX

Aggregation functions

MINA Function

Returns the smallest value in a column, including any logical values and numbers represented as text.

Syntax

MINA(<column>)

Parameters

Term	Definition
column	The column for which you want to find the minimum value.

Return Value

A decimal number.

Remarks

The MINA function takes as argument a column that contains numbers, and determines the smallest value as follows:

- If the column contains no numeric values, MINA returns 0 (zero).
- Rows in the column that evaluates to logical values, such as TRUE and FALSE are treated as 1 if TRUE and 0 (zero) if FALSE.
- Empty cells are ignored.

If you do not want to include logical values and text as part of the calculation, use the MIN function instead.

Example

Description

The following expression returns the minimum freight charge from the table, InternetSales.

Code

```
=MINA(InternetSales[Freight])
```

Example

Description

The following expression returns the minimum value in the column, PostalCode. Because the data type of the column is text, the function does not find any numeric values, and the formula returns zero (0).

Code

```
=MINA([PostalCode])
```

See Also

MIN

MINA

MINX Function

Returns the smallest numeric value that results from evaluating an expression for each row of a table.

Syntax

MINX(<table>, < expression>)

Parameters

Term	Definition
table	The table containing the rows for which the expression will be evaluated.
expression	The expression to be evaluated for each row of the table.

Return Value

A decimal number.

Remarks

The MINX function takes as its first argument a table, or an expression that returns a table. The second argument contains the expression that is evaluated for each row of the table.

The MINX function evaluates the results of the expression in the second argument according to the following rules:

- Only numbers are counted. If the expression does not result in a number, MINX returns 0 (zero).
- Empty cells, logical values, and text values are ignored. Numbers represented as text are treated as text.

If you want to include logical values and text representations of numbers in a reference as part of the calculation, use the MINA function.

Example

Description

The following example filters the table, InternetSales, and returns only rows for a specific sales territory. The formula then finds the minimum value in the column, Freight.

Code

```
=MINX( FILTER(InternetSales, [SalesTerritoryKey] = 5),[Freight])
```

Example

Description

The following example uses the same filtered table as in the previous example, but instead of merely looking up values in the column for each row of the filtered table, the function calculates the sum of two columns, Freight and TaxAmt, and returns the smallest value resulting from that calculation.

Code

```
=MINX( FILTER(InternetSales, InternetSales[SalesTerritoryKey] = 5),
InternetSales[Freight] + InternetSales[TaxAmt])
```

Comments

In the first example, the names of the columns are unqualified. In the second example, the column names are fully qualified.

See Also

MIN

MINA

MINX

Aggregation functions

RANK.EQ Function

Returns the ranking of a number in a list of numbers.

Syntax

RANK.EQ(<value>, <columnName>[, <order>])

Parameters

Parameter	Description
	Any DAX expression that returns a single scalar value whose rank is to be found. The expression is to be evaluated exactly once, before the function is evaluated, and it's value passed to the argument list.
	The name of an existing column against which ranks will be determined. It cannot be an expression or a column created using these functions:

ADDCOLUMNS, ROW or SUMMARIZE.

(Optional) A value that specifies how to rank number, low to high or high to low:

value	alternate value	Description
0 (zero)	FALSE	Ranks in descending order of columnName. If value is equal to the highest number in columnName then **RANK.EQ** is 1.
1	TRUE	Ranks in ascending order of columnName. If value is equal to the lowest number in columnName then **RANK.EQ** is 1.

Return Value

A number indicating the rank of value among the numbers in columnName.

Exceptions

Remarks

- columnName cannot refer to any column created using these functions: ADDCOLUMNS, ROW or SUMMARIZE.I
- If value is not in columnName or value is a blank, then RANK.EQ returns a blank value.

- Duplicate values of value receive the same rank value; the next rank value assigned will be the rank value plus the number of duplicate values. For example if five (5) values are tied with a rank of 11 then the next value will receive a rank of 16 (11 + 5).

Example

Description

The following example creates a calculated column that ranks the values in SalesAmount_USD, from the InternetSales_USD table, against all numbers in the same column.

Code

```
=RANK.EQ(InternetSales_USD[SalesAmount_USD],
InternetSales_USD[SalesAmount_USD])
```

Comments

Example

Description

The following example ranks a subset of values against a given sample. Assume that you have a table of local students with their performance in a specific national test and, also, you have the entire set of scores in that national test. The following calculated column will give you the national ranking for each of the local students.

Code

```
=RANK.EQ(Students[Test_Score], NationalScores[Test_Score])
```

Comments

RANKX Function

Returns the ranking of a number in a list of numbers for each row in the table argument.

Syntax

RANKX(<table>, <expression>[, <value>[, <order>[, <ties>]]][, <expression>[, <value>[, <order>[, <ties>]]]]...)

Parameters

Parameter	Description
	Any DAX expression that returns a table of data over which the expression is evaluated.
	Any DAX expression that returns a single scalar value. The expression

is evaluated for each row of table, to generate all possible values for ranking. See the remarks section to understand the function behavior when expression evaluates to BLANK.

(Optional) Any DAX expression that returns a single scalar value whose rank is to be found. See the remarks section to understand the function's behavior when value is not found in the expression.

When the value parameter is omitted, the value of expression at the current row is used instead.

(Optional) A value that specifies how to rank value, low to high or high to low:

value	alternate value	Description
0 (zero)	FALSE	Ranks in descending order of values of expression. If value is equal to the highest number in expression then RANKX returns 1. This is the default value when order parameter is omitted.
1	TRUE	Ranks in ascending order of

		expression. If value is equal to the lowest number in expression then RANKX returns 1.

(Optional) An enumeration that defines how to determine ranking when there are ties.

enumeration	Description
Skip	The next rank value, after a tie, is the rank value of the tie plus the count of tied values. For example if five (5) values are tied with a rank of 11 then the next value will receive a rank of 16 (11 + 5). This is the default value when ties parameter is omitted.
Dense	The next rank value, after a tie, is the next rank value. For example if five (5) values are tied with a rank of 11 then the next value will receive a rank of 12.

Return Value

The rank number of value among all possible values of expression evaluated for all rows of table numbers.

Exceptions

Remarks

- If expression or value evaluates to BLANK it is treated as a 0 (zero) for all expressions that result in a number, or as an empty text for all text expressions.
- If value is not among all possible values of expression then RANKX temporarily adds value to the values from expression and re-evaluates RANKX to determine the proper rank of value.
- Optional arguments might be skipped by placing an empty comma (,) in the argument list, i.e. RANKX(Inventory, [InventoryCost],,,"Dense")

Example

Description

The following calculated column in the Products table calculates the sales ranking for each product in the Internet channel.

Code

```
=RANKX(ALL(Products), SUMX(RELATEDTABLE(InternetSales), [SalesAmount]))
```

Comments

ROW Function

Returns a table with a single row containing values that result from the expressions given to each column.

Syntax

ROW(<name>, <expression>][,<name>, <expression>]...])

Parameters

Parameter	Description
name	The name given to the column, enclosed in double quotes.
expression	Any DAX expression that returns a single scalar value to populate. name.

Return Value

A single row table

Remarks

Arguments must always come in pairs of name and expression.

Example

Description

The following example returns a single row table with the total sales for internet and resellers channels.

Code

```
ROW("Internet Total Sales (USD)",
SUM(InternetSales_USD[SalesAmount_USD]),
         "Resellers Total Sales (USD)",
SUM(ResellerSales_USD[SalesAmount_USD]))
```

Comments

The code is split in two lines for readability purposes

STDEV.S Function

Returns the standard deviation of a sample population.

Syntax

STDEV.S(<ColumnName>)

Parameters

Parameter	Description
columnName	The name of an existing column using standard DAX syntax, usually fully qualified. It cannot be an expression.

Return Value

A number that represents the standard deviation of a sample population.

Exceptions

Remarks

1. STDEV.S assumes that the column refers to a sample of the population. If your data represents the entire population, then compute the standard deviation by using STDEV.P.
2. STDEV.S uses the following formula:

$\sqrt{[\sum(x - \tilde{x})^2/(n-1)]}$

where $\tilde{x}$ is the average value of x for the sample population

and n is the population size

3. Blank rows are filtered out from columnName and not considered in the calculations.
4. An error is returned if columnName contains less than 2 non-blank rows.

This DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see http://go.microsoft.com/fwlink/?LinkId=219171.

Example

Description

The following example shows the formula for a measure that calculates the standard deviation of the column, SalesAmount_USD, when the table InternetSales_USD is the sample population.

Code

```
=STDEV.S(InternetSales_USD[SalesAmount_USD])
```

STDEV.P Function

Returns the standard deviation of the entire population.

Syntax

STDEV.P(<ColumnName>)

Parameters

Parameter	Description
	The name of an existing column using standard DAX syntax, usually fully qualified. It cannot be an expression.

Return Value

A number representing the standard deviation of the entire population.

Exceptions

Remarks

1. STDEV.P assumes that the column refers to the entire population. If your data represents a sample of the population, then compute the standard deviation by using STDEV.S.

2. STDEV.P uses the following formula:
   $\sqrt{[\sum(x - \tilde{x})^2/n]}$
   where $\tilde{x}$ is the average value of x for the entire population
   and n is the population size
3. Blank rows are filtered out from columnName and not considered in the calculations.
4. An error is returned if columnName contains less than 2 non-blank rows

This DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see http://go.microsoft.com/fwlink/?LinkId=219171.

Example

Description

The following example shows the formula for a measure that calculates the standard deviation of the column, SalesAmount_USD, when the table InternetSales_USD is the entire population.

Code

```
=STDEV.P(InternetSales_USD[SalesAmount_USD])
```

STDEVX.S Function

Returns the standard deviation of a sample population.

Syntax

STDEVX.S(<table>, <expression>)

Parameters

Parameter	Description
	Any DAX expression that returns a single scalar value, where the expression is to be evaluated multiple times (for each row/context).
	Any DAX expression that returns a single scalar value, where the expression is to be evaluated multiple times (for each row/context).

Return Value

A number with the standard deviation of a sample population.

Exceptions

Remarks

1. STDEVX.S evaluates expression for each row of table and returns the standard deviation of expression assuming that table refers to a sample of the population. If table represents the entire population, then compute the standard deviation by using STDEVX.P.
2. STDEVX.S uses the following formula:

   $\sqrt{[\sum(x - \tilde{x})^2/(n-1)]}$

   where $\tilde{x}$ is the average value of x for the entire population

   and n is the population size
3. Blank rows are filtered out from columnName and not considered in the calculations.
4. An error is returned if columnName contains less than 2 non-blank rows.

This DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see http://go.microsoft.com/fwlink/?LinkId=219171.

Example

Description

The following example shows the formula for a calculated column that estimates the standard deviation of the unit price per product for a sample population, when the formula is used in the Product table.

Code

```
=STDEVX.S(RELATEDTABLE(InternetSales_USD),
InternetSales_USD[UnitPrice_USD] -
(InternetSales_USD[DiscountAmount_USD]/InternetSales_USD[OrderQuantity]
))
```

STDEVX.P Function

Returns the standard deviation of the entire population.

Syntax

STDEVX.P(<table>, <expression>)

Parameters

Parameter	Description
table	Any DAX expression that returns a table of data.

expression	Any DAX expression that returns a single scalar value, where the expression is to be evaluated multiple times (for each row/context).

Return Value

A number that represents the standard deviation of the entire population.

Remarks

1. STDEVX.P evaluates expression for each row of table and returns the standard deviation of expression assuming that table refers to the entire population. If the data in table represents a sample of the population, you should compute the standard deviation by using STDEVX.S instead.
2. STDEVX.P uses the following formula:

   $\sqrt{[\sum(x - \tilde{x})^2/n]}$

   where $\tilde{x}$ is the average value of x for the entire population

   and n is the population size
3. Blank rows are filtered out from columnName and not considered in the calculations.
4. An error is returned if columnName contains less than 2 non-blank rows

This DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see http://go.microsoft.com/fwlink/?LinkId=219171.

Example

Description

The following example shows the formula for a calculated column that calculates the standard deviation of the unit price per product, when the formula is used in the Product table.

Code

```
=STDEVX.P(RELATEDTABLE(InternetSales_USD),
InternetSales_USD[UnitPrice_USD] -
(InternetSales_USD[DiscountAmount_USD]/InternetSales_USD[OrderQuantity]
))
```

SUMMARIZE Function

Returns a summary table for the requested totals over a set of groups.

Syntax

SUMMARIZE(<table>, <groupBy_columnName>[, <groupBy_columnName>]…[, <name>, <expression>]…)

Parameters

Parameter	Description
	Any DAX expression that returns a table of data.
	The qualified name of an existing column to be used to create summary groups based on the values found in it. This parameter cannot be an expression.
	The name given to a total or summarize column, enclosed in double quotes.
	Any DAX expression that returns a single scalar value, where the expression is to be evaluated multiple times (for each row/context).

Return Value

A table with the selected columns for the groupBy_columnName arguments and the summarized columns designed by the name arguments.

Remarks

1. Each column for which you define a name must have a corresponding expression; otherwise, an error is returned. The first argument, name, defines the name of the column in the results. The second argument, expression, defines the calculation performed to obtain the value for each row in that column.
2. groupBy_columnName must be either in table or in a related table to table.
3. Each name must be enclosed in double quotation marks.
4. The function groups a selected set of rows into a set of summary rows by the values of one or more groupBy_columnName columns. One row is returned for each group.

Example

Description

The following example returns a summary of the reseller sales grouped around the calendar year and the product category name, this result table allows you to do analysis over the reseller sales by year and product category.

Code

```
SUMMARIZE(ResellerSales_USD
      , DateTime[CalendarYear]
      , ProductCategory[ProductCategoryName]
      , "Sales Amount (USD)", SUM(ResellerSales_USD[SalesAmount_USD])
      , "Discount Amount (USD)", SUM(ResellerSales_USD[DiscountAmount])
      )
```

Comments

The following table shows a preview of the data as it would be received by any function expecting to receive a table:

DateTime[CalendarYear]	ProductCategory[ProductCategoryName]	[Sales Amount (USD)]	[Discount Amount (USD)]
2008	Bikes	12968255.42	36167.6592
2005	Bikes	6958251.043	4231.1621
2006	Bikes	18901351.08	178175.8399
2007	Bikes	24256817.5	276065.992
2008	Components	2008052.706	39.9266
2005	Components	574256.9865	0
2006	Components	3428213.05	948.7674
2007	Components	5195315.216	4226.0444
2008	Clothing	366507.844	4151.1235
2005	Clothing	31851.1628	90.9593

2006	Clothing	455730.9729	4233.039
2007	Clothing	815853.2868	12489.3835
2008	Accessories	153299.924	865.5945
2005	Accessories	18594.4782	4.293
2006	Accessories	86612.7463	1061.4872
2007	Accessories	275794.8403	4756.6546

Advanced SUMMARIZE options

SUMMARIZE with ROLLUP

The addition of the ROLLUP() syntax modifies the behavior of the SUMMARIZE function by adding roll-up rows to the result on the groupBy_columnName columns.

SUMMARIZE(<table>, <groupBy_columnName>[, <groupBy_columnName>]…[, **ROLLUP(<groupBy_columnName>**[,< groupBy_columnName>…])][, <name>, <expression>]…)

ROLLUP parameters

groupBy_columnName

The qualified name of an existing column to be used to create summary groups based on the values found in it. This parameter cannot be an expression.

Note: All other SUMMARIZE parameters are explained before and not repeated here for brevity.

Remarks

- The columns mentioned in the ROLLUP expression cannot be referenced as part of a groupBy_columnName columns.

Example

The following example adds roll-up rows to the Group-By columns of the SUMMARIZE function call.

```
SUMMARIZE(ResellerSales_USD
      , ROLLUP( DateTime[CalendarYear],
ProductCategory[ProductCategoryName])
      , "Sales Amount (USD)", SUM(ResellerSales_USD[SalesAmount_USD])
```

```
      , "Discount Amount (USD)", SUM(ResellerSales_USD[DiscountAmount])
)
```

The following table shows a preview of the data as it would be received by any function expecting to receive a table:

DateTime[CalendarYear]	ProductCategory[ProductCategoryName]	[Sales Amount (USD)]	[Discount Amount (USD)]
2008	Bikes	12968255.42	36167.6592
2005	Bikes	6958251.043	4231.1621
2006	Bikes	18901351.08	178175.8399
2007	Bikes	24256817.5	276065.992
2008	Components	2008052.706	39.9266
2005	Components	574256.9865	0
2006	Components	3428213.05	948.7674
2007	Components	5195315.216	4226.0444
2008	Clothing	366507.844	4151.1235
2005	Clothing	31851.1628	90.9593
2006	Clothing	455730.9729	4233.039
2007	Clothing	815853.2868	12489.3835
2008	Accessories	153299.924	865.5945

2005	Accessories	18594.4782	4.293
2006	Accessories	86612.7463	1061.4872
2007	Accessories	275794.8403	4756.6546
2008		15496115.89	41224.3038
2005		7582953.67	4326.4144
2006		22871907.85	184419.1335
2007		30543780.84	297538.0745
		76494758.25	527507.9262

SUMMARIZE with ISSUBTOTAL

Enables the user to create another column, in the Summarize function, that returns True if the row contains sub-total values for the column given as argument to ISSUBTOTAL, otherwise returns False.

SUMMARIZE(<table>, <groupBy_columnName>[, <groupBy_columnName>]…[, ROLLUP(<groupBy_columnName>[,< groupBy_columnName>…])][, <name>, {<expression>|**ISSUBTOTAL(<columnName>)**}]…)

ISSUBTOTAL parameters

columnName

The name of any column in table of the SUMMARIZE function or any column in a related table to table.

Return Value

A **True** value if the row contains a sub-total value for the column given as argument, otherwise returns **False**

Remarks

- ISSUBTOTAL can only be used in the expression part of a SUMMARIZE function.
- ISSUBTOTAL must be preceded by a matching name column.

Example

The following sample generates an ISSUBTOTAL() column for each of the ROLLUP() columns in the given SUMMARIZE() function call.

```
SUMMARIZE(ResellerSales_USD
      , ROLLUP( DateTime[CalendarYear],
ProductCategory[ProductCategoryName])
      , "Sales Amount (USD)", SUM(ResellerSales_USD[SalesAmount_USD])
      , "Discount Amount (USD)", SUM(ResellerSales_USD[DiscountAmount])
      , "Is Sub Total for DateTimeCalendarYear",
ISSUBTOTAL(DateTime[CalendarYear])
      , "Is Sub Total for ProductCategoryName",
ISSUBTOTAL(ProductCategory[ProductCategoryName])
)
```

The following table shows a preview of the data as it would be received by any function expecting to receive a table:

[Is Sub Total for DateTimeCalendarYear]	[Is Sub Total for ProductCategoryName]	DateTime[CalendarYear]	ProductCategory[ProductCategoryName]	[Sales Amount (USD)]	[Discount Amount (USD)]
FALSE	FALSE				
FALSE	FALSE	2008	Bikes	12968255.42	36167.6592
FALSE	FALSE	2005	Bikes	6958251.043	4231.1621
FALSE	FALSE	2006	Bikes	18901351.08	178175.8399
FALSE	FALSE	2007	Bikes	24256817.5	27606 5.992
FALSE	FALSE	2008	Components	2008052.706	39.9266
FALSE	FALSE	2005	Components	574256.9865	0
FALSE	FALSE	2006	Components	34282	948.76

				13.05	74
FALSE	FALSE	2007	Components	51953 15.216	4226.0 444
FALSE	FALSE	2008	Clothing	36650 7.844	4151.1 235
FALSE	FALSE	2005	Clothing	31851. 1628	90.959 3
FALSE	FALSE	2006	Clothing	45573 0.9729	4233.0 39
FALSE	FALSE	2007	Clothing	81585 3.2868	12489. 3835
FALSE	FALSE	2008	Accessories	15329 9.924	865.59 45
FALSE	FALSE	2005	Accessories	18594. 4782	4.293
FALSE	FALSE	2006	Accessories	86612. 7463	1061.4 872
FALSE	FALSE	2007	Accessories	27579 4.8403	4756.6 546
FALSE	TRUE				
FALSE	TRUE	2008		15496 115.89	41224. 3038
FALSE	TRUE	2005		75829 53.67	4326.4 144
FALSE	TRUE	2006		22871 907.85	18441 9.1335
FALSE	TRUE	2007		30543 780.84	29753 8.0745
TRUE	TRUE			76494 758.25	52750 7.9262

TOPN Function

Returns the top N rows of the specified table.

Syntax

TOPN(<n_value>, <table>, <orderBy_expression>, [<order>[, <orderBy_expression>, [<order>]]…])

Parameters

Parameter	Description
	The number of rows to return. It is any DAX expression that returns a single scalar value, where the expression is to be evaluated multiple times (for each row/context). See the remarks section to understand when the number of rows returned could possible be larger than n_value. See the remarks section to understand when an empty table is returned.
	Any DAX expression that returns a table of data from where to extract the top 'n' rows.
	Any DAX expression where the result value is used to sort the table and it is evaluated for each row of table.
	(Optional) A value that specifies how to sort orderBy_expression values, ascending or descending:

value	alternate value	Description
0 (zero)	FALSE	Sorts in descending order of values of *order_by*. This is the

		default value when *order* parameter is omitted.
1	TRUE	Ranks in ascending order of *order_by*.

Return Value

A table with the top N rows of table or an empty table if n_value is 0 (zero) or less. Rows are not necessarily sorted in any particular order.

Remarks

- If there is a tie, in order_by values, at the N-th row of the table, then all tied rows are returned. Then, when there are ties at the N-th row the function might return more than n rows.
- If n_value is 0 (zero) or less then TOPN returns an empty table.
- TOPN does not guarantee any sort order for the results.

Example

Description

The following sample creates a measure with the sales of the top 10 sold products.

Code

```
=SUMX(TOPN(10, SUMMARIZE(Product, [ProductKey], "TotalSales",
SUMX(RELATED(InternetSales_USD[SalesAmount_USD]),
InternetSales_USD[SalesAmount_USD]) +
SUMX(RELATED(ResellerSales_USD[SalesAmount_USD]),
ResellerSales_USD[SalesAmount_USD]))
```

Comments

VAR.S Function

Returns the variance of a sample population.

Syntax

VAR.S(<columnName>)

Parameters

Parameter	Description
	The name of an existing column using standard DAX syntax, usually fully qualified. It cannot be an expression.

Return Value

A number with the variance of a sample population.

Exceptions

Remarks

1. VAR.S assumes that the column refers to a sample of the population. If your data represents the entire population, then compute the variance by using VAR.P.
2. VAR.S uses the following formula:

   $\sum(x - \tilde{x})^2/(n-1)$

   where $\tilde{x}$ is the average value of x for the sample population

   and n is the population size
3. Blank rows are filtered out from columnName and not considered in the calculations.
4. An error is returned if columnName contains less than 2 non-blank rows.

This DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see http://go.microsoft.com/fwlink/?LinkId=219171.

Example

Description

The following example shows the formula for a measure that calculates the variance of the SalesAmount_USD column from the InternetSales_USD for a sample population.

Code

```
=VAR.S(InternetSales_USD[SalesAmount_USD])
```

VAR.P Function

Returns the variance of the entire population.

Syntax

VAR.P(<columnName>)

Parameters

Parameter	Description

	The name of an existing column using standard DAX syntax, usually fully qualified. It cannot be an expression.

Return Value

A number with the variance of the entire population.

Remarks

1. VAR.P assumes that the column refers the entire population. If your data represents a sample of the population, then compute the variance by using VAR.S.
2. VAR.P uses the following formula:

   $\sum(x - \tilde{x})^2/n$

   where $\tilde{x}$ is the average value of x for the entire population

   and n is the population size
3. Blank rows are filtered out from columnName and not considered in the calculations.
4. An error is returned if columnName contains less than 2 non-blank rows

This DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see http://go.microsoft.com/fwlink/?LinkId=219171.

Example

Description

The following example shows the formula for a measure that estimates the variance of the SalesAmount_USD column from the InternetSales_USD table, for the entire population.

Code

```
=VAR.P(InternetSales_USD[SalesAmount_USD])
```

VARX.S Function

Returns the variance of a sample population.

Syntax

VARX.S(<table>, <expression>)

Parameters

Parameter	Description
	Any DAX expression that returns a table of data.

	Any DAX expression that returns a single scalar value, where the expression is to be evaluated multiple times (for each row/context).

Return Value

A number that represents the variance of a sample population.

Exceptions

Remarks

1. VARX.S evaluates expression for each row of table and returns the variance of expression; on the assumption that table refers to a sample of the population. If table represents the entire population, then you should compute the variance by using VARX.P.
2. VAR.S uses the following formula:

   $\sum(x - \tilde{x})^2/(n-1)$

   where $\tilde{x}$ is the average value of x for the sample population

   and n is the population size
3. Blank rows are filtered out from columnName and not considered in the calculations.
4. An error is returned if columnName contains less than 2 non-blank rows.

This DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see http://go.microsoft.com/fwlink/?LinkId=219171.

Example

Description

The following example shows the formula for a calculated column that estimates the variance of the unit price per product for a sample population, when the formula is used in the Product table.

Code

```
=VARX.S(InternetSales_USD, InternetSales_USD[UnitPrice_USD] –
(InternetSales_USD[DiscountAmount_USD]/InternetSales_USD[OrderQuantity]
))
```

VARX.P Function

Returns the variance of the entire population.

Syntax

VARX.P(<table>, <expression>)

Parameters

Parameter	Description
table	Any DAX expression that returns a table of data.
expression	Any DAX expression that returns a single scalar value, where the expression is to be evaluated multiple times (for each row/context).

Return Value

A number with the variance of the entire population.

Exceptions

Remarks

1. VARX.P evaluates <expression> for each row of <table> and returns the variance of <expression> assuming that <table> refers to the entire population.. If <table> represents a sample of the population, then compute the variance by using VARX.S.
2. VARX.P uses the following formula:

   $\sum(x - \tilde{x})^2/n$

   where $\tilde{x}$ is the average value of x for the entire population

   and n is the population size
3. Blank rows are filtered out from columnName and not considered in the calculations.
4. An error is returned if columnName contains less than 2 non-blank rows

This DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see http://go.microsoft.com/fwlink/?LinkId=219171.

Example

Description

The following example shows the formula for a calculated column that calculates the variance of the unit price per product, when the formula is used in the Product table

Code

```
=VARX.P(InternetSales_USD, InternetSales_USD[UnitPrice_USD] -
(InternetSales_USD[DiscountAmount_USD]/InternetSales_USD[OrderQuantity]
))
```

Text Functions

Data Analysis Expressions (DAX) includes a set of text functions that is based on the library of string functions in Excel, but which has been modified to work with tables and columns. This section lists all text functions available in the DAX language.

In this Section

BLANK Function (DAX)

CONCATENATE Function (DAX)

EXACT Function (DAX)

FIND Function (DAX)

FIXED Function (DAX)

FORMAT Function (DAX)

- Pre-defined numeric formats for the FORMAT function (DAX)
- Custom numeric formats for the FORMAT function (DAX)
- Pre-defined Date and Time formats for the FORMAT function (DAX)
- Custom Date and Time formats for the FORMAT function (DAX)

LEFT Function (DAX)

LEN Function (DAX)

LOWER Function (DAX)

MID Function (DAX)

REPLACE Function (DAX)

REPT Function (DAX)

RIGHT Function (DAX)

SEARCH Function (DAX)

SUBSTITUTE Function (DAX)

TRIM Function (DAX)

UPPER Function (DAX)

VALUE Function (DAX)

See Also

Function Reference (DAX)

Date and Time Functions (DAX)

Filter and Value Functions (DAX)

Information Functions (DAX)

Logical Functions (DAX)

Math and Trigonometric Functions (DAX)

Statistical Functions (DAX)

BLANK Function

Returns a blank.

Syntax

BLANK()

Return Value

A blank.

Remarks

Blanks are not equivalent to nulls. DAX uses blanks for both database nulls and for blank cells in Excel. For more information, see Data Types in DAX.

Some DAX functions treat blank cells somewhat differently from Microsoft Excel. Blanks and empty strings ("") are not always equivalent, but some operations may treat them as such. For details on the behavior of an individual function or operator, see Function Reference.

Example

Description

The following example illustrates how you can work with blanks in formulas. The formula calculates the ratio of sales between the Resellers and the Internet channels. However, before attempting to calculate the ratio the denominator should be checked for zero values. If the denominator is zero then a blank value should be returned; otherwise, the ratio is calculated.

Code

```
=IF( SUM(InternetSales_USD[SalesAmount_USD])= 0    , BLANK()    ,
SUM(ResellerSales_USD[SalesAmount_USD])/SUM(InternetSales_USD[SalesAmou
nt_USD])    )
```

Comments

The table shows the expected results when this formula is used to create a PivotTable.

Reseller to Internet sales ratio	Column Labels			
Row Labels	Accessories	Bikes	Clothing	Grand Total
2005		2.65		2.89
2006		3.33		4.03

Reseller to Internet sales ratio	Column Labels			
2007	1.04	2.92	6.63	3.51
2008	0.41	1.53	2.00	1.71
Grand Total	0.83	2.51	5.45	2.94

Note that, in the original data source, the column evaluated by the BLANK function might have included text, empty strings, or nulls. If the original data source was a SQL Server database, nulls and empty strings are different kinds of data. However, for this operation an implicit type cast is performed and DAX treats them as the same.

See Also

Text Functions

ISBLANK

CONCATENATE Function

Joins two text strings into one text string.

Syntax

CONCATENATE(<text1>, <text2>)

Parameters

Term	Definition
text1, **text2**	The text strings to be joined into a single text string. Strings can include text or numbers. You can also use column references.

Return Value

The concatenated string.

Remarks

The CONCATENATE function joins two text strings into one text string. The joined items can be text, numbers or Boolean values represented as text, or a combination of those items. You can also use a column reference if the column contains appropriate values.

The CONCATENATE function in DAX accepts only two arguments, whereas the Excel CONCATENATE function accepts up to 255 arguments. If you need to concatenate multiple columns, you can create a series of calculations or, better, use the concatenation operator (**&**) to join all of them in a simpler expression.

If you want to use text strings directly, rather than using a column reference, you must enclose each string in double quotation marks.

This DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see http://go.microsoft.com/fwlink/?LinkId=219171.

Example: Concatenation of Literals

Description

The sample formula creates a new string value by combining two string values that you provide as arguments.

Code

```
=CONCATENATE("Hello ", "World")
```

Example: Concatenation of Strings in Columns

Description

The sample formula returns the customer's full name as listed in a phone book. Note how a nested function is used as the second argument. This is one way to concatenate multiple strings, when you have more than two values that you want to use as arguments.

Code

```
=CONCATENATE(Customer[LastName], CONCATENATE(", ",
Customer[FirstName]))
```

Example: Conditional Concatenation of Strings in Columns

Description

The sample formula creates a new calculated column in the Customer table with the full customer name as a combination of first name, middle initial, and last name. If there is no middle name, the last name comes directly after the first name. If there is a middle name, only the first letter of the middle name is used and the initial letter is followed by a period.

Code

```
=CONCATENATE( [FirstName]&" ", CONCATENATE( IF( LEN([MiddleName])>1,
LEFT([MiddleName],1)&" ", ""), [LastName]))
```

Comments

This formula uses nested CONCATENATE and IF functions, together with the ampersand (&) operator, to conditionally concatenate three string values and add spaces as separators.

Example: Concatenation of Columns with Different Data Types

The following example demonstrates how to concatenate values in columns that have different data types. If the value that you are concatenating is numeric, the value will be implicitly converted to text. If both values are numeric, both values will be cast to text and concatenated as if they were strings.

Product description	Product abbreviation (column 1 of composite key)	Product number (column 2 of composite key)	New generated key column
Mountain bike	MTN	40	MTN40
Mountain bike	MTN	42	MTN42

Code

```
=CONCATENATE('Products'[Product abbreviation],'Products'[Product number])
```

Comments

The CONCATENATE function in DAX accepts only two arguments, whereas the Excel CONCATENATE function accepts up to 255 arguments. If you need to add more arguments, you can use the ampersand (&) operator. For example, the following formula produces the results, MTN-40 and MTN-42.

```
=[Product abbreviation] & "-" & [Product number]
```

See Also

Text functions

EXACT Function

Compares two text strings and returns TRUE if they are exactly the same, FALSE otherwise. EXACT is case-sensitive but ignores formatting differences. You can use EXACT to test text being entered into a document.

Syntax

EXACT(<text1>,<text2>)

Parameters

Term	Definition
text1	The first text string or column that contains text.

Term	Definition
text2	The second text string or column that contains text.

Property Value/Return Value

True or false. (Boolean)

Example

Description

The following formula checks the value of Column1 for the current row against the value of Column2 for the current row, and returns TRUE if they are the same, and returns FALSE if they are different.

Code

```
=EXACT([Column1],[Column2])
```

See Also

Text functions

FIND Function

Returns the starting position of one text string within another text string. FIND is case-sensitive.

Syntax

FIND(<find_text>, <within_text>[, [<start_num>][, <NotFoundValue>]])

Parameters

Term	Definition
find_text	The text you want to find. Use double quotes (empty text) to match the first character in **within_text**. You can use wildcard characters — the question mark (?) and asterisk (*) — in **find_text**. A question mark matches any single character; an asterisk matches any sequence of characters. If you want to find an actual question mark or asterisk, type a tilde (~) before the character.
within_text	The text containing the text you want to

Term	Definition
	find.
start_num	(optional) The character at which to start the search; if omitted, **start_num** = 1. The first character in **within_text** is character number 1.
NotFoundValue	(optional) The value that should be returned when the operation does not find a matching substring, typically 0, -1, or BLANK().

Property Value/Return Value

Number that shows the starting point of the text string you want to find.

Remarks

Whereas Microsoft Excel has multiple versions of the FIND function to accommodate single-byte character set (SBCS) and double-byte character set (DBCS) languages, DAX uses Unicode and counts each character the same way; therefore, you do not need to use a different version depending on the character type.

This DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see http://go.microsoft.com/fwlink/?LinkId=219171.

Example

Description

The following formula finds the position of the first letter of the product designation, BMX, in the string that contains the product description.

Code

```
=FIND("BMX","line of BMX racing goods")
```

See Also

Text functions

FIXED Function

Rounds a number to the specified number of decimals and returns the result as text. You can specify that the result be returned with or without commas.

Syntax

FIXED(<number>, <decimals>, <no_commas>)

Parameters

Term	Definition
number	The number you want to round and convert to text, or a column containing a number.
decimals	(optional) The number of digits to the right of the decimal point; if omitted, 2.
no_commas	(optional) A logical value: if 1, do not display commas in the returned text; if 0 or omitted, display commas in the returned text.

Property Value/Return Value

A number represented as text.

Remarks

If the value used for the **decimals** parameter is negative, **number** is rounded to the left of the decimal point.

If you omit **decimals**, it is assumed to be 2.

If **no_commas** is 0 or is omitted, then the returned text includes commas as usual.

The major difference between formatting a cell containing a number by using a command and formatting a number directly with the FIXED function is that FIXED converts its result to text. A number formatted with a command from the formatting menu is still a number.

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models,
see http://go.microsoft.com/fwlink/?LinkId=219172.

Example

Description

The following example gets the numeric value for the current row in column, PctCost, and returns it as text with 4 decimal places and no commas.

Code

```
=FIXED([PctCost],3,1)
```

Comments

Numbers can never have more than 15 significant digits, but decimals can be as large as 127.

See Also

Text functions
Math and trig functions

FORMAT Function

Converts a value to text according to the specified format.

Syntax

FORMAT(<value>, <format_string>)

Parameters

Term	Definition
value	A value or expression that evaluates to a single value.
format_string	A string with the formatting template.

Return Value

A string containing **value** formatted as defined by **format_string**.

Important

- If value is BLANK() the function returns an empty string.
- If format_string is BLANK(), the value is formatted with a "General Number" or "General Date" format (according to **value** type).

Remarks

For information on how to use the **format_string** parameter, see the appropriate topic listed below:

To Format	Follow these instructions
Numbers	Use predefined numeric formats or create user-defined numeric formats.
Dates and times	Use predefined date/time formats or create user-defined date/time formats.

All predefined formatting strings use the current user locale when formatting the result.

Caution

The format strings supported as an argument to the DAX FORMAT function are based on the format strings used by Visual Basic (OLE Automation), not on the format strings used by the .NET Framework. Therefore, you might get unexpected results or an error if the argument does not match any defined format strings. For example, “p” as an abbreviation for “Percent” is not supported. Strings that you provide as an argument to the FORMAT function that are not included in the list of predefined format strings are handled as part of a custom format string, or as a string literal.

This DAX function is not supported for use in DirectQuery mode. For more information about limitations in DirectQuery models,
see http://go.microsoft.com/fwlink/?LinkId=219172.

See Also

predefined numeric formats

user-defined numeric formats

predefined date/time formats

user-defined date/time formats

VALUE

Pre-Defined Numeric Formats for the FORMAT Function

The following table identifies the predefined numeric format names. These may be used by name as the style argument for the Format function.

Format specification	Description
`"General Number"`	Displays number with no thousand separators.
`"Currency"`	Displays number with thousand separators, if appropriate; displays two digits to the right of the decimal separator. Output is based on system locale settings.
`"Fixed"`	Displays at least one digit to the left and two digits to the right of the decimal separator.
`"Standard"`	Displays number with thousand separators, at least one digit to the left and two digits to the right of the decimal separator.
`"Percent"`	Displays number multiplied by 100 with a percent sign (%) appended immediately to

Format specification	Description
	the right; always displays two digits to the right of the decimal separator.
"Scientific"	Uses standard scientific notation, providing two significant digits.
"Yes/No"	Displays No if number is 0; otherwise, displays Yes.
"True/False"	Displays False if number is 0; otherwise, displays True.
"On/Off"	Displays Off if number is 0; otherwise, displays On.

Remarks

Note that format strings are based on Visual Basic (OLE Automation) and therefore might have slightly different behavior than the format strings used by the .NET Framework. Abbreviations such as "P" and "x" are not supported. Any other strings that you provide as an argument to the FORMAT function are interpreted as defining a custom format.

Important

- If value is BLANK() the function returns an empty string.
- If format_string is BLANK(), the value is formatted with a "General Number" format.

Example

Description

The following samples show the usage of different predefined formatting strings to format a numeric value.

Code

```
FORMAT( 12345.67, "General Number")
FORMAT( 12345.67, "Currency")
FORMAT( 12345.67, "Fixed")
FORMAT( 12345.67, "Standard")
FORMAT( 12345.67, "Percent")
FORMAT( 12345.67, "Scientific")
```

Comments

The above expressions return the following results:

12345.67 "General Number" displays the number with no formatting.

$12,345.67 "Currency" displays the number with your currency locale formatting. The sample here shows the default United States currency formatting.

12345.67 "Fixed" displays at least one digit to the left of the decimal separator and two digits to the right of the decimal separator.

12,345.67 " Standard " displays at least one digit to the left of the decimal separator and two digits to the right of the decimal separator, and includes thousand separators. The sample here shows the default United States number formatting.

1,234,567.00 % "Percent" displays the number as a percentage (multiplied by 100) with formatting and the percent sign at the right of the number separated by a single space.

1.23E+04 "Scientific" displays the number in scientific notation with two decimal digits.

See Also

FORMAT

Predefined date formats

Custom numeric formats

Custom Numeric Formats for the FORMAT Function

A user-defined format expression for numbers can have from one to three sections separated by semicolons. If the Style argument of the Format function contains one of the predefined numeric formats, only one section is allowed.

If you use	This is the result
One section only	The format expression applies to all values.
Two sections	The first section applies to positive values and zeros; the second applies to negative values.
Three sections	The first section applies to positive values, the second applies to negative values, and the third applies to zeros.

Format Specifications

The following table identifies characters you can use to create user-defined number formats.

Format specification	Description
None	Displays the number with no formatting.

Format specification	Description
0 (zero character)	Digit placeholder. Displays a digit or a zero. If the expression has a digit in the position where the zero appears in the format string, displays the digit; otherwise, displays a zero in that position. If the number has fewer digits than there are zeros (on either side of the decimal) in the format expression, displays leading or trailing zeros. If the number has more digits to the right of the decimal separator than there are zeros to the right of the decimal separator in the format expression, rounds the number to as many decimal places as there are zeros. If the number has more digits to the left of the decimal separator than there are zeros to the left of the decimal separator in the format expression, displays the extra digits without modification.
#	Digit placeholder. Displays a digit or nothing. If the expression has a digit in the position where the # character appears in the format string, displays the digit; otherwise, displays nothing in that position. This symbol works like the 0 digit placeholder, except that leading and trailing zeros aren't displayed if the number has fewer digits than there are # characters on either side of the decimal separator in the format expression.
. (dot character)	Decimal placeholder. The decimal placeholder determines how many digits are displayed to the left and right of the decimal separator. If the format expression contains only # characters to the left of this symbol; numbers smaller than 1 begin with a decimal separator. To display a leading zero displayed with fractional numbers, use zero as the first digit placeholder to the left of the decimal separator. In some locales, a

Format specification	Description
	comma is used as the decimal separator. The actual character used as a decimal placeholder in the formatted output depends on the number format recognized by your system. Thus, you should use the period as the decimal placeholder in your formats even if you are in a locale that uses a comma as a decimal placeholder. The formatted string will appear in the format correct for the locale.
%	Percent placeholder. Multiplies the expression by 100. The percent character (%) is inserted in the position where it appears in the format string.
, (comma character)	Thousand separator. The thousand separator separates thousands from hundreds within a number that has four or more places to the left of the decimal separator. Standard use of the thousand separator is specified if the format contains a thousand separator surrounded by digit placeholders (0 or #). A thousand separator immediately to the left of the decimal separator (whether or not a decimal is specified) or as the rightmost character in the string means "scale the number by dividing it by 1,000, rounding as needed." Numbers smaller than 1,000 but greater or equal to 500 are displayed as 1, and numbers smaller than 500 are displayed as 0. Two adjacent thousand separators in this position scale by a factor of 1 million, and an additional factor of 1,000 for each additional separator. Multiple separators in any position other than immediately to the left of the decimal separator or the rightmost position in the string are treated simply as specifying the use of a thousand separator. In some

Format specification	Description
	locales, a period is used as a thousand separator. The actual character used as the thousand separator in the formatted output depends on the Number Format recognized by your system. Thus, you should use the comma as the thousand separator in your formats even if you are in a locale that uses a period as a thousand separator. The formatted string will appear in the format correct for the locale. For example, consider the three following format strings: "#,0.", which uses the thousands separator to format the number 100 million as the string "100,000,000". "#0,.", which uses scaling by a factor of one thousand to format the number 100 million as the string "100000". "#,0,.", which uses the thousands separator and scaling by one thousand to format the number 100 million as the string "100,000".
: (colon character)	Time separator. In some locales, other characters may be used to represent the time separator. The time separator separates hours, minutes, and seconds when time values are formatted. The actual character used as the time separator in formatted output is determined by your system settings.
/ (forward slash character)	Date separator. In some locales, other characters may be used to represent the date separator. The date separator separates the day, month, and year when date values are formatted. The actual character used as the date separator in formatted output is determined by your system settings.
E- , **E+** , **e-** , **e+**	Scientific format. If the format expression contains at least one digit placeholder (0 or

Format specification	Description
	#) to the left of E-, E+, e-, or e+, the number is displayed in scientific format and E or e is inserted between the number and its exponent. The number of digit placeholders to the left determines the number of digits in the exponent. Use E- or e- to place a minus sign next to negative exponents. Use E+ or e+ to place a minus sign next to negative exponents and a plus sign next to positive exponents. You must also include digit placeholders to the right of this symbol to get correct formatting.
- + $ ( )	Literal characters. These characters are displayed exactly as typed in the format string. To display a character other than one of those listed, precede it with a backslash (\) or enclose it in double quotation marks (" ").
**** (backward slash character)	Displays the next character in the format string. To display a character that has special meaning as a literal character, precede it with a backslash (\). The backslash itself isn't displayed. Using a backslash is the same as enclosing the next character in double quotation marks. To display a backslash, use two backslashes (\\). Examples of characters that can't be displayed as literal characters are the date-formatting and time-formatting characters (a, c, d, h, m, n, p, q, s, t, w, y, /, and :), the numeric-formatting characters (#, 0, %, E, e, comma, and period), and the string-formatting characters (@, &, <, >, and !).
"ABC"	Displays the string inside the double quotation marks (" "). To include a string in the style argument from within code, you must use Chr(34) to enclose the text (34 is the character code for a quotation mark

Format specification	Description
	(")).

The following table contains some sample format expressions for numbers. (These examples all assume that your system's locale setting is English-U.S.) The first column contains the format strings for the Format function; the other columns contain the resulting output if the formatted data has the value given in the column headings.

Format (Style)	"5" formatted as	"-5" formatted as	"0.5" formatted as	"0" formatted as
Zero-length string ("")	5	-5	0.5	0
0	5	-5	1	0
0.00	5.00	-5.00	0.50	0.00
#,##0	5	-5	1	0
$#,##0;($#,##0)	$5	($5)	$1	$0
$#,##0.00;($#,##0.00)	$5.00	($5.00)	$0.50	$0.00
0%	500%	-500%	50%	0%
0.00%	500.00%	-500.00%	50.00%	0.00%
0.00E+00	5.00E+00	-5.00E+00	5.00E-01	0.00E+00
0.00E-00	5.00E00	-5.00E00	5.00E-01	0.00E00
"$#,##0;;\Z\e\r\o"	$5	$-5	$1	Zero

Remarks

If you include semicolons with nothing between them, the missing section is printed using the format of the positive value.

See Also

FORMAT

Predefined numeric formats

Custom date-time formats

Pre-defined Date and Time formats for the FORMAT Function

The following table identifies the predefined date and time format names. If you use strings other than these predefined strings, they will be interpreted as a custom date and time format.

Format specification	Description
`"General Date"`	Displays a date and/or time. For example, 3/12/2008 11:07:31 AM. Date display is determined by your application's current culture value.
`"Long Date"` or `"Medium Date"`	Displays a date according to your current culture's long date format. For example, Wednesday, March 12, 2008.
`"Short Date"`	Displays a date using your current culture's short date format. For example, 3/12/2008.
`"Long Time"` or `"Medium Time"`	Displays a time using your current culture's long time format; typically includes hours, minutes, seconds. For example, 11:07:31 AM.
`"Short Time"`	Displays a time using your current culture's short time format. For example, 11:07 AM.

Remarks

The formatting strings are based on Visual Basic (OLE Automation) and not the .NET Framework formatting strings; therefore, your results might be slightly different than what you expect from .NET format strings. Note that abbreviations such as “D” for Long Date and “t” for Short Time are not supported.

Important

- If value is BLANK() the function returns an empty string.
- If format_string is BLANK(), the value is formatted with a "General Date" format.

See Also

Custom Date Formats

Custom Date and Time formats for the FORMAT Function

The following table shows characters you can use to create user-defined date/time formats.

Format specification	Description
(:)	Time separator. In some locales, other characters may be used to represent the time separator. The time separator separates hours, minutes, and seconds when time values are formatted. The actual character that is used as the time separator in formatted output is determined by your application's current culture value.
(/)	Date separator. In some locales, other characters may be used to represent the date separator. The date separator separates the day, month, and year when date values are formatted. The actual character that is used as the date separator in formatted output is determined by your application's current culture.
(%)	Used to indicate that the following character should be read as a single-letter format without regard to any trailing letters. Also used to indicate that a single-letter format is read as a user-defined format. See what follows for additional details.
d	Displays the day as a number without a leading zero (for example, 1). Use %d if this is the only character in your user-defined numeric format.
dd	Displays the day as a number with a leading zero (for example, 01).
ddd	Displays the day as an abbreviation (for example, Sun).
dddd	Displays the day as a full name (for example, Sunday).
M	Displays the month as a number without a leading zero (for example, January is represented as 1). Use %M if this is the only

Format specification	Description
	character in your user-defined numeric format.
MM	Displays the month as a number with a leading zero (for example, 01/12/01).
MMM	Displays the month as an abbreviation (for example, Jan).
MMMM	Displays the month as a full month name (for example, January).
gg	Displays the period/era string (for example, A.D.).
h	Displays the hour as a number without leading zeros using the 12-hour clock (for example, 1:15:15 PM). Use %h if this is the only character in your user-defined numeric format.
hh	Displays the hour as a number with leading zeros using the 12-hour clock (for example, 01:15:15 PM).
H	Displays the hour as a number without leading zeros using the 24-hour clock (for example, 1:15:15). Use %H if this is the only character in your user-defined numeric format.
HH	Displays the hour as a number with leading zeros using the 24-hour clock (for example, 01:15:15).
m	Displays the minute as a number without leading zeros (for example, 12:1:15). Use %m if this is the only character in your user-defined numeric format.
mm	Displays the minute as a number with leading zeros (for example, 12:01:15).
s	Displays the second as a number without leading zeros (for example, 12:15:5). Use %s if this is the only character in your user-defined numeric format.

Format specification	Description
ss	Displays the second as a number with leading zeros (for example, 12:15:05).
f	Displays fractions of seconds. For example ff displays hundredths of seconds, whereas ffff displays ten-thousandths of seconds. You may use up to seven f symbols in your user-defined format. Use %f if this is the only character in your user-defined numeric format.
t	Uses the 12-hour clock and displays an uppercase A for any hour before noon; displays an uppercase P for any hour between noon and 11:59 P.M. Use %t if this is the only character in your user-defined numeric format.
tt	For locales that use a 12-hour clock, displays an uppercase AM with any hour before noon; displays an uppercase PM with any hour between noon and 11:59 P.M. For locales that use a 24-hour clock, displays nothing.
y	Displays the year number (0-9) without leading zeros. Use %y if this is the only character in your user-defined numeric format.
yy	Displays the year in two-digit numeric format with a leading zero, if applicable.
yyy	Displays the year in four-digit numeric format.
yyyy	Displays the year in four-digit numeric format.
z	Displays the timezone offset without a leading zero (for example, -8). Use %z if this is the only character in your user-defined numeric format.

Format specification	Description
zz	Displays the timezone offset with a leading zero (for example, -08)
zzz	Displays the full timezone offset (for example, -08:00)

Remarks

Formatting strings are case sensitive. Different formatting can be obtained by using a different case. For example, when formatting a date value with the string "D" you get the date in the long format (according to your current locale). However, if you change the case to "d" you get the date in the short format. Also, unexpected results or an error might occur if the intended formatting does not match the case of any defined format string.

Date/Time formatting uses the current user locale to determine the ultimate format of the string. For example, to format the date March 18, 1995, with the following format string "M/d/yyyy", if the user locale is set to the United States of America (en-us) the result is '3/12/1995', but if the user locale is set to Germany (de-de) the result is '18.03.1995'.

See Also

FORMAT

Custom numeric formats

Predefined date and time formats

LEFT Function

Returns the specified number of characters from the start of a text string.

Syntax

LEFT(<text>, <num_chars>)

Parameters

Term	Definition
text	The text string containing the characters you want to extract, or a reference to a column that contains text.
num_chars	(optional) The number of characters you want LEFT to extract; if omitted, 1.

Property Value/Return Value

A text string.

Remarks

Whereas Microsoft Excel contains different functions for working with text in single-byte and double-byte character languages, DAX works with Unicode and stores all characters as the same length; therefore, a single function is enough.

This DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see http://go.microsoft.com/fwlink/?LinkId=219171.

Example

Description

The following example returns the first five characters of the company name in the column [ResellerName] and the first five letters of the geographical code in the column [GeographyKey] and concatenates them, to create an identifier.

Code

```
=CONCATENATE(LEFT('Reseller'[ResellerName],LEFT(GeographyKey,3))
```

Comments

If the **num_chars** argument is a number that is larger than the number of characters available, the function returns the maximum characters available and does not raise an error. For example, the column [GeographyKey] contains numbers such as 1, 12 and 311; therefore the result also has variable length.

See Also

Text functions

LEN Function

Returns the number of characters in a text string.

Syntax

LEN(<text>)

Parameters

Term	Definition
text	The text whose length you want to find, or a column that contains text. Spaces count as characters.

Return Value

A whole number indicating the number of characters in the text string.

Exceptions

Remarks

Whereas Microsoft Excel has different functions for working with single-byte and double-byte character languages, DAX uses Unicode and stores all characters with the same length.

Therefore, LEN always counts each character as 1, no matter what the default language setting is.

If you use LEN with a column that contains non-text values, such as dates or Booleans, the function implicitly casts the value to text, using the current column format.

Example

Description

The following formula sums the lengths of addresses in the columns, [AddressLine1] and [AddressLine2].

Code

```
=LEN([AddressLine1])+LEN([AddressLine2)
```

See Also

Text functions

LOWER Function

Converts all letters in a text string to lowercase.

Syntax

LOWER(<text>)

Parameters

Term	Definition
text	The text you want to convert to lowercase, or a reference to a column that contains text.

Property Value/Return Value

Text in lowercase.

Remarks

Characters that are not letters are not changed. For example, the formula `=LOWER("123ABC")` returns **123abc**.

Example

Description

The following formula gets each row in the column, [ProductCode], and converts the value to all lowercase. Numbers in the column are not affected.

Code

```
=LOWER('New Products'[ProductCode])
```

See Also

Text functions

MID Function

Returns a string of characters from the middle of a text string, given a starting position and length.

Syntax

MID(<text>, <start_num>, <num_chars>)

Parameters

Term	Definition
text	The text string from which you want to extract the characters, or a column that contains text.
start_num	The position of the first character you want to extract. Positions start at 1.
num_chars	The number of characters to return.

Property Value/Return Value

A string of text of the specified length.

Remarks

Whereas Microsoft Excel has different functions for working with single-byte and double-byte characters languages, DAX uses Unicode and stores all characters with the same length.

Example

Description

The following examples return the same results, the first 5 letters of the column, [ResellerName]. The first example uses the fully qualified name of the column and specifies the starting point; the second example omits the table name and the parameter, **num_chars**.

Code

```
=MID('Reseller'[ResellerName],5,1))
```

```
=MID([ResellerName,5])
```

Comments

The results are the same if you use the following formula:

```
=LEFT([ResellerName],5)
```

See Also

Text functions

REPLACE Function

REPLACE replaces part of a text string, based on the number of characters you specify, with a different text string.

Syntax

REPLACE(<old_text>, <start_num>, <num_chars>, <new_text>)

Parameters

Term	Definition
old_text	The string of text that contains the characters you want to replace, or a reference to a column that contains text.
start_num	The position of the character in **old_text** that you want to replace with **new_text**.
num_chars	The number of characters that you want to replace. **Warning** If the argument, num_chars, is a blank or references a column that evaluates to a blank, the string for new_text is inserted at the position, start_num, without replacing any characters. This is the same

Term	Definition
	behavior as in Excel.
new_text	The replacement text for the specified characters in **old_text**.

Property Value/Return Value

A text string.

Remarks

Whereas Microsoft Excel has different functions for use with single-byte and double-byte character languages, DAX uses Unicode and therefore stores all characters as the same length.

This DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see http://go.microsoft.com/fwlink/?LinkId=219171.

Example

Description

The following formula creates a new calculated column that replaces the first two characters of the product code in column, [ProductCode], with a new two-letter code, OB.

Code

```
=REPLACE('New Products'[Product Code],1,2,"OB")
```

See Also

Text functions

SUBSTITUTE (DAX)

REPT Function

Repeats text a given number of times. Use REPT to fill a cell with a number of instances of a text string.

Syntax

REPT(<text>, <num_times>)

Parameters

Term	Definition
text	The text you want to repeat.

Term	Definition
num_times	A positive number specifying the number of times to repeat text.

Property Value/Return Value

A string containing the changes.

Remarks

If **number_times** is 0 (zero), REPT returns a blank.

If **number_times** is not an integer, it is truncated.

The result of the REPT function cannot be longer than 32,767 characters, or REPT returns an error.

Example: Repeating Literal Strings

Description

The following example returns the string, 85, repeated three times.

Code

```
=REPT("85",3)
```

Example: Repeating Column Values

Description

The following example returns the string in the column, [MyText], repeated for the number of times in the column, [MyNumber]. Because the formula extends for the entire column, the resulting string depends on the text and number value in each row.

Code

```
=REPT([MyText],[MyNumber])
```

Comments

MyText	MyNumber	CalculatedColumn1
Text	2	TextText
Number	0	
85	3	858585

See Also

Text functions

RIGHT Function

RIGHT returns the last character or characters in a text string, based on the number of characters you specify.

Syntax

RIGHT(<text>, <num_chars>)

Parameters

Term	Definition
text	The text string that contains the characters you want to extract, or a reference to a column that contains text.
num_chars	(optional) The number of characters you want RIGHT to extract; is omitted, 1. You can also use a reference to a column that contains numbers.

If the column reference does not contain text, it is implicitly cast as text.

Property Value/Return Value

A text string containing the specified right-most characters.

Remarks

RIGHT always counts each character, whether single-byte or double-byte, as 1, no matter what the default language setting is.

This DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see http://go.microsoft.com/fwlink/?LinkId=219171

Example: Returning a Fixed Number of Characters

Description

The following formula returns the last two digits of the product code in the New Products table.

Code

```
=RIGHT('New Products'[ProductCode],2)
```

Example: Using a Column Reference to Specify Character Count

Description

The following formula returns a variable number of digits from the product code in the New Products table, depending on the number in the column, MyCount. If there is no value in the column, MyCount, or the value is a blank, RIGHT also returns a blank.

Code

```
=RIGHT('New Products'[ProductCode],[MyCount])
```

See Also

Text functions

LEFT

MID

SEARCH Function

Returns the number of the character at which a specific character or text string is first found, reading left to right. Search is case-insensitive and accent sensitive.

Syntax

SEARCH(<find_text>, <within_text>[, [<start_num>][, <NotFoundValue>]])

Parameters

Term	Definition
find_text	The text that you want to find. You can use wildcard characters — the question mark (?) and asterisk (*) — in **find_text**. A question mark matches any single character; an asterisk matches any sequence of characters. If you want to find an actual question mark or asterisk, type a tilde (~) before the character.
within_text	The text in which you want to search for **find_text**, or a column containing text.
start_num	(optional) The character position in **within_text** at which you want to start searching. If omitted, 1.
NotFoundValue	(optional) The value that should be returned when the operation does not find a matching substring, typically 0, -1, or BLANK().

Return Value

The number of the starting position of the first text string from the first character of the second text string.

Remarks

1. The search function is case insensitive. Searching for "N" will find the first occurrence of 'N' or 'n'.
2. The search function is accent sensitive. Searching for "á" will find the first occurrence of 'á' but no occurrences of 'a', 'à', or the capitalized versions 'A', 'Á'.
3. By using this function, you can locate one text string within a second text string, and return the position where the first string starts.
4. You can use the SEARCH function to determine the location of a character or text string within another text string, and then use the MID function to return the text, or use the REPLACE function to change the text.
5. If the **find_text** cannot be found in **within_text**, the formula returns an error. This behavior is like Excel, which returns #VALUE if the substring is not found. Nulls in **within_text** will be interpreted as an empty string in this context.

This DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see http://go.microsoft.com/fwlink/?LinkId=219171.

Example: Search within a String

Description

The following formula finds the position of the letter "n" in the word "printer".

Code

```
=SEARCH("n","printer")
```

Comments

The formula returns 4 because "n" is the fourth character in the word "printer."

Example: Search within a Column

Description

You can use a column reference as an argument to SEARCH. The following formula finds the position of the character "-" (hyphen) in the column, [PostalCode].

Code

```
=SEARCH("-",[PostalCode])
```

Comments

The return result is a column of numbers, indicating the index position of the hyphen.

Example: Error-Handling with SEARCH

Description

The formula in the preceding example will fail if the search string is not found in every row of the source column. Therefore, the next example demonstrates how to use IFERROR with the SEARCH function, to ensure that a valid result is returned for every row.

The following formula finds the position of the character "-" within the column, and returns -1 if the string is not found.

Code

```
= IFERROR(SEARCH("-",[PostalCode]),-1)
```

Comments

Note that the data type of the value that you use as an error output must match the data type of the non-error output type. In this case, you provide a numeric value to be output in case of an error because SEARCH returns an integer value.

However, you could also return a blank (empty string) by using `BLANK()` as the second argument to IFERROR.

See Also

MID

REPLACE

TEXT functions

SUBSTITUTE Function

Replaces existing text with new text in a text string.

Syntax

SUBSTITUTE(<text>, <old_text>, <new_text>, <instance_num>)

Parameters

Term	Definition
text	The text in which you want to substitute characters, or a reference to a column containing text.
old_text	The existing text that you want to replace.
new_text	The text you want to replace **old_text** with.
instance_num	(optional) The occurrence of **old_text** you want to replace. If omitted, every instance of **old_text** is replaced

Property Value/Return Value

A string of text.

Remarks

Use the SUBSTITUTE function when you want to replace specific text in a text string; use the REPLACE function when you want to replace any text of variable length that occurs in a specific location in a text string.

The SUBSTITUTE function is case-sensitive. If case does not match between **text** and **old_text**, SUBSTITUTE will not replace the text.

This DAX function may return different results when used in a model that is deployed and then queried in DirectQuery mode. For more information about semantic differences in DirectQuery mode, see http://go.microsoft.com/fwlink/?LinkId=219171.

Example: Substitution within a String

Description

The following formula creates a copy of the column [Product Code] that substitutes the new product code **NW** for the old product code **PA** wherever it occurs in the column.

Code

```
=SUBSTITUTE([Product Code], "NW", "PA")
```

See Also

Text functions

REPLACE (DAX)

TRIM Function

Removes all spaces from text except for single spaces between words.

Syntax

TRIM(<text>)

Parameters

Term	Definition
text	The text from which you want spaces removed, or a column that contains text.

Property Value/Return Value

The string with spaces removed.

Remarks

Use TRIM on text that you have received from another application that may have irregular spacing.

The TRIM function was originally designed to trim the 7-bit ASCII space character (value 32) from text. In the Unicode character set, there is an additional space character called the nonbreaking space character that has a decimal value of 160. This character is commonly used in Web pages as the HTML entity, . By itself, the TRIM function does not remove this nonbreaking space character. For an example of how to trim both space characters from text, see Remove spaces and nonprinting characters from text.

Example

Description

The following formula creates a new string that does not have trailing white space.

Code

```
=TRIM("A column with trailing spaces.   ")
```

Comments

When you create the formula, the formula is propagated through the row just as you typed it, so that you see the original string in each formula and the results are not apparent. However, when the formula is evaluated the string is trimmed.

You can verify that the formula produces the correct result by checking the length of the calculated column created by the previous formula, as follows:

```
=LEN([Calculated Column 1])
```

See Also

Text functions

UPPER Function

Converts a text string to all uppercase letters

Syntax

UPPER (<text>)

Parameters

Term	Definition
text	The text you want converted to uppercase, or a reference to a column that contains text.

Property Value/Return Value

Same text, in uppercase.

Example

Description

The following formula converts the string in the column, [ProductCode], to all uppercase. Non-alphabetic characters are not affected.

Code

```
=UPPER(['New Products'[Product Code])
```

See Also

Text functions

LOWER (DAX)

VALUE Function

Converts a text string that represents a number to a number.

Syntax

VALUE(<text>)

Parameters

Term	Definition
text	The text to be converted.

Return Value

The converted number in decimal data type.

Remarks

The value passed as the **text** parameter can be in any of the constant, number, date, or time formats recognized by Microsoft Excel and the PowerPivot Add-in. If **text** is not in one of these formats, an error is returned. For more information about PowerPivot data types, see Data Types.

You do not generally need to use the VALUE function in a formula because the PowerPivot add-in implicitly converts text to numbers as necessary.

You can also use column references. For example, if you have a column that contains mixed number types, VALUE can be used to convert all values to a single numeric data type. However, if you use the VALUE function with a column that contains mixed numbers and text, the entire column is flagged with an error, because not all values in all rows can be converted to numbers.

Example

Description

The following formula converts the typed string, "3", into the numeric value 3.

Code

```
=VALUE("3")
```

Comments

See Also

Text functions

Formula Compatibility in DirectQuery Mode

The Data Analysis Expression language (DAX) can be used to create measures and other custom formulas that you use in tabular models and in PowerPivot workbooks. In almost every respect, the models that you create in these two environments are identical, and you can use the same measures, relationships, and KPIs, etc. However, if you build a tabular model and deploy it in DirectQuery mode, there are some restrictions on the formulas that you can use. This topic provides an overview of the differences, lists the functions that are not supported in DirectQuery mode, and lists the functions that are supported but might return different results.

Within this topic, we use the term *in-memory model* to refer to both PowerPivot models, which use a local cache, as well as tabular models, which are fully hosted in memory data on an Analysis Services server running in Tabular mode. We use *DirectQuery models* to refer to tabular models that have been deployed in DirectQuery mode. For information about DirectQuery mode, see DQ Intro.

Semantic Differences

Describes the types of differences that might arise when the same formula is used to DirectQuery mode.

- Comparisons
- Casts
- Mathematical functions and arithmetic operations
- Supported Numeric and Date-Time Ranges
- Currency
- Aggregation Functions
- Text Functions

Functions Supported in DirectQuery Mode

This section lists functions that can be used in DirectQuery mode, but which might return different results.

Functions Not Supported in DirectQuery Mode

The section lists functions that cannot be used in models deployed in DirectQuery mode.

Functions that are not in either of these lists are expected to behave identically regardless of the model storage or query mode.

Overview of Differences between In-Memory and DirectQuery Mode

Queries on a model deployed in DirectQuery mode can return different results than when the same model is deployed in-memory, because data is fetched directly from a relational data store and aggregations required by formulas are performed using the relevant relational engine, rather than using the xVelocity in-memory analytics engine (VertiPaq) for storage and calculation.

For example, there are differences in the way that certain relational data stores handle numeric values, dates, nulls, and so forth.

In contrast, the DAX language is intended to emulate as closely as possible the behavior of functions in Microsoft Excel. For example, when handling nulls, empty strings and zero values, Excel attempts to provide the best answer regardless of the precise data type, and therefore the xVelocity engine does the same. However, when a tabular model is deployed in DirectQuery mode and passes formulas to a relational data source for evaluation, the data must be handled according to the semantics of the relational data source, which typically require distinct handling of empty strings vs. nulls. For this reason, the same formula might return a different result when evaluated against cached data and against data fetched solely from the relational store.

Additionally, some functions cannot be used at all in DirectQuery mode because the calculation would require that the data in the current context be sent to the relational data source as a parameter. For example, measures in a PowerPivot workbook often use time intelligence functions that reference date ranges available within the workbook. Such formulas generally cannot be used in DirectQuery mode.

List of Semantic Differences

This section lists the types of semantic differences that you can expect, and describes any limitations that might apply to the usage of functions or to query results.

Comparisons

DAX in in-memory models supports comparisons of two expressions that resolve to scalar values of different data types. However, models that are deployed in DirectQuery mode use the data types and comparison operators of the relational engine, and therefore might return different results.

The following comparisons will always generate an error when used in a calculation on a DirectQuery data source:

- Numeric data type compared to any string data type
- Numeric data type compared to a Boolean value
- Any string data type compared to a Boolean value

In general, DAX is more forgiving of data type mismatches in in-memory models, and will attempt an implicit cast of values up to two times, as described in this section. However, formulas sent to a relational data store in DirectQuery mode are evaluated more strictly, following the rules of the relational engine, and are more likely to fail.

Comparisons of strings and numbers

EXAMPLE: `"2" < 3`

The formula compares a text string to a number. The expression is **true** in both DirectQuery mode and in-memory models.

In an in-memory model, the result is **true** because numbers as strings are implicitly cast to a numerical data type for comparisons with other numbers. SQL also implicitly casts text numbers as numbers for comparison to numerical data types.

Note that this represents a change in behavior from the first version of PowerPivot, which would return **false**, because the text "2" would always be considered larger than any number.

Comparison of text with Boolean

EXAMPLE: `"VERDADERO" = TRUE`

This expression compares a text string with a Boolean value. In general, for DirectQuery or In-Memory models, comparing a string value to a Boolean value results in an error. The only exceptions to the rule are when the string contains the word **true** or the word **false**; if the string contains any of true or false values, a conversion to Boolean is made and the comparison takes place giving the logical result.

Comparison of nulls

EXAMPLE: `EVALUATE ROW("X", BLANK() = BLANK())`

This formula compares the SQL equivalent of a null to a null. It returns **true** in in-memory and DirectQuery models; a provision is made in DirectQuery model to guarantee similar behavior to in-memory model.

Note that in Transact-SQL, a null is never equal to a null. However, in DAX, a blank is equal to another blank. This behavior is the same for all in-memory models. It is important to note that DirectQuery mode uses, most of, the semantics of SQL Server; but, in this case it separates from it giving a new behavior to NULL comparisons.

Casts

There is no cast function as such in DAX, but implicit casts are performed in many comparison and arithmetic operations. It is the comparison or arithmetic operation that determines the data type of the result. For example,

- Boolean values are treated as numeric in arithmetic operations, such as TRUE + 1, or the function MIN applied to a column of Boolean values. A NOT operation also returns a numeric value.

- Boolean values are always treated as logical values in comparisons and when used with EXACT, AND, OR, &&, or ||.

Cast from string to Boolean

In in-memory and DirectQuery models, casts are permitted to Boolean values from these strings only: **""** (empty string), **"true"**, **"false"**; where an empty string casts to false value.

Casts to the Boolean data type of any other string results in an error.

Cast from string to date/time

In DirectQuery mode, casts from string representations of dates and times to actual **datetime** values behave the same way as they do in SQL Server.

For information about the rules governing casts from string to **datetime** data types in PowerPivot models, see the DAX Syntax Specification.

Models that use the in-memory data store support a more limited range of text formats for dates than the string formats for dates that are supported by SQL Server. However, DAX supports custom date and time formats. For more information, see DAX Predefined Date formats and Custom date formats.

Cast from string to other non Boolean values

When casting from strings to non-Boolean values, DirectQuery mode behaves the same as SQL Server. For more information, see Cast and Convert.

Cast from numbers to string not allowed

EXAMPLE: `CONCATENATE(102,",345")`

Casting from numbers to strings is not allowed in SQL Server.

This formula returns an error in tabular models and in DirectQuery mode; however, the formula produces a result in PowerPivot.

No support for two-try casts in DirectQuery

In-memory models often attempt a second cast when the first one fails. This never happens in DirectQuery mode.

EXAMPLE: `TODAY() + "13:14:15"`

In this expression, the first parameter has type **datetime** and second parameter has type **string**. However, the casts when combining the operands are handled differently. DAX will perform an implicit cast from **string** to **double**. In in-memory models, the formula engine attempts to cast directly to **double**, and if that fails, it will try to cast the string to **datetime**.

In DirectQuery mode, only the direct cast from **string** to **double** will be applied. If this cast fails, the formula will return an error.

Math Functions and Arithmetic Operations

Some mathematical functions will return different results in DirectQuery mode, because of differences in the underlying data type or the casts that can be applied in operations. Also, the restrictions described above on the allowed range of values might affect the outcome of arithmetic operations.

Order of addition

When you create a formula that adds a series of numbers, an in-memory model might process the numbers in a different order than a DirectQuery model. Therefore, when you have many very large positive numbers and very large negative numbers, you may get an error in one operation and results in another operation.

Use of the POWER function

EXAMPLE: `POWER(-64, 1/3)`

In DirectQuery mode, the POWER function cannot use negative values as the base when raised to a fractional exponent. This is the expected behavior in SQL Server.

In an in-memory model, the formula returns -4.

Numerical overflow operations

In Transact-SQL, operations that result in a numerical overflow return an overflow error; therefore, formulas that result in an overflow also raise an error in DirectQuery mode.

However, the same formula when used in an in-memory model returns an eight-byte integer. That is because the formula engine does not perform checks for numerical overflows.

LOG functions with blanks return different results

SQL Server handles nulls and blanks differently than the xVelocity engine. As a result, the following formula returns an error in DirectQuery mode, but return infinity in in-memory mode.

`EXAMPLE: LOG(blank())`

The same limitations apply to the other logarithmic functions: LOG10 and LN.

For more information about the **blank** data type in DAX, see DAX Syntax Specification.

Division by 0 and division by Blank

In DirectQuery mode, division by zero (0) or division by BLANK will always result in an error. SQL Server does not support the notion of infinity, and because the natural result of any division by 0 is infinity, the result is an error. However, SQL Server supports division by nulls, and the result must always equal null.

Rather than return different results for these operations, in DirectQuery mode, both types of operations (division by zero and division by null) return an error.

Note that, in Excel and in PowerPivot models, division by zero also returns an error. Division by a blank returns a blank.

The following expressions are all valid in in-memory models, but will fail in DirectQuery mode:

```
1/BLANK
1/0
0.0/BLANK
0/0
```

The expression `BLANK/BLANK` is a special case that returns `BLANK` in both for in-memory models, and in DirectQuery mode.

Supported Numeric and Date-Time Ranges

Formulas in PowerPivot and tabular models in memory are subject to the same limitations as Excel with regard to maximum allowed values for real numbers and dates. However, differences can arise when the maximum value is returned from a calculation or query, or when values are converted, cast, rounded, or truncated.

- If values of types **Currency** and **Real** are multiplied, and the result is larger than the maximum possible value, in DirectQuery mode, no error is raised, and a null is returned.
- In in-memory models, no error is raised, but the maximum value is returned.

In general, because the accepted date ranges are different for Excel and SQL Server, results can be guaranteed to match only when dates are within the common date range, which is inclusive of the following dates:

- Earliest date: March 1, 1990
- Latest date: December 31, 9999

If any dates used in formulas fall outside this range, either the formula will result in an error, or the results will not match.

Floating point values supported by CEILING

EXAMPLE: `EVALUATE ROW("x", CEILING(-4.398488E+30, 1))`

The Transact-SQL equivalent of the DAX CEILING function only supports values with magnitude of 10^19 or less. A rule of thumb is that floating point values should be able to fit into **bigint**.

Datepart functions with dates that are out of range

Results in DirectQuery mode are guaranteed to match those in in-memory models only when the date used as the argument is in the valid date range. If these conditions are not satisfied, either an error will be raised, or the formula will return different results in DirectQuery than in in-memory mode.

EXAMPLE: `MONTH(0)` or `YEAR(0)`

In DirectQuery mode, the expressions return 12 and 1899, respectively.

In in-memory models, the expressions return 1 and 1900, respectively.

EXAMPLE: `EOMONTH(0.0001, 1)`

The results of this expression will match only when the data supplied as a parameter is within the valid date range.

EXAMPLE: `EOMONTH(blank(), blank())` or `EDATE(blank(), blank())`

The results of this expression should be the same in DirectQuery mode and in-memory mode.

Truncation of time values

EXAMPLE: `SECOND(1231.04097222222)`

In DirectQuery mode, the result is truncated, following the rules of SQL Server, and the expression evaluates to 59.

In in-memory models, the results of each interim operation are rounded; therefore, the expression evaluates to 0.

The following example demonstrates how this value is calculated:

1. The fraction of the input (0.04097222222) is multiplied by 24.
2. The resulting hour value (0.98333333328) is multiplied by 60.
3. The resulting minute value is 58.9999999968.
4. The fraction of the minute value (0.9999999968) is multiplied by 60.
5. The resulting second value (59.999999808) rounds up to 60.
6. 60 is equivalent to 0.

SQL Time data type not supported

In-memory models do not support use of the new SQL **Time** data type. In DirectQuery mode, formulas that reference columns with this data type will return an error. Time data columns cannot be imported into an in-memory model.

However, in PowerPivot and in cached models, sometimes the engine casts the time value to an acceptable data type, and the formula returns a result.

This behavior affects all functions that use a date column as a parameter.

Currency

In DirectQuery mode, if the result of an arithmetic operation has the type **Currency**, the value must be within the following range:

- Minimum:
- Maximum:

Combining currency and REAL data types

EXAMPLE: `Currency sample 1`

If **Currency** and **Real** types are multiplied, and the result is larger than 9223372036854774784 (), DirectQuery mode will not raise an error.

In an in-memory model, an error is raised if the absolute value of the result is larger

than .

Operation results in an out-of-range value

EXAMPLE: `Currency sample 2`

If operations on any two currency values result in a value that is outside the specified range, an error is raised in in-memory models, but not in DirectQuery models.

Combining currency with other data types

Division of currency values by values of other numeric types can result in different results.

Aggregation Functions

Statistical functions on a table with one row return different results. Aggregation functions over empty tables also behave differently in in-memory models than they do in DirectQuery mode.

Statistical functions over a table with a single row

If the table that is used as argument contains a single row, in DirectQuery mode, statistical functions such as STDEV and VAR return null.

In an in-memory model, a formula that uses STDEV or VAR over a table with a single row returns a division by zero error.

Text Functions

Because relational data stores provide different text data types than does Excel, you may see different results when searching strings or working with substrings. The length of strings also can be different.

In general, any string manipulation functions that use fixed-size columns as arguments can have different results.

Additionally, in SQL Server, some text functions support additional arguments that are not provided in Excel. If the formula requires the missing argument you can get different results or errors in the in-memory model.

Operations that return a character using LEFT, RIGHT, etc. may return the correct character but in a different case, or no results

EXAMPLE: `LEFT(["text"], 2)`

In DirectQuery mode, the case of the character that is returned is always exactly the same as the letter that is stored in the database. However, the xVelocity engine uses a different algorithm for compression and indexing of values, to improve performance.

By default, the Latin1_General collation is used, which is case-insensitive but accent-sensitive. Therefore, if there are multiple instances of a text string in lower case, upper case, or mixed case, all instances are considered the same string, and only the first instance of the string is stored in the index. All text functions that operate on stored strings will retrieve the specified portion of the indexed form. Therefore, the example

formula would return the same value for the entire column, using the first instance as the input.

String Storage and Collation in Tabular Models

This behavior also applies to other text functions, including RIGHT, MID, and so forth.

String length affects results

EXAMPLE: `SEARCH("within string", "sample target  text", 1, 1)`

If you search for a string using the SEARCH function, and the target string is longer than the within string, DirectQuery mode raises an error.

In an in-memory model, the searched string is returned, but with its length truncated to the length of <within text>.

EXAMPLE: `EVALUATE ROW("X", REPLACE("CA", 3, 2, "California"))`

If the length of the replacement string is greater than the length of the original string, in DirectQuery mode, the formula returns null.

In in-memory models, the formula follows the behavior of Excel, which concatenates the source string and the replacement string, which returns CACalifornia.

Implicit TRIM in the middle of strings

EXAMPLE: `TRIM(" A sample sentence with leading white space")`

DirectQuery mode translates the DAX TRIM function to the SQL statement `LTRIM(RTRIM(<column>))`. As a result, only leading and trailing white space is removed.

In contrast, the same formula in an in-memory model removes spaces within the string, following the behavior of Excel.

Implicit RTRIM with use of LEN function

EXAMPLE: `LEN('string_column')`

Like SQL Server, DirectQuery mode automatically removes white space from the end of string columns: that is, it performs an implicit RTRIM. Therefore, formulas that use the LEN function can return different values if the string has trailing spaces.

In-memory supports additional parameters for SUBSTITUTE

EXAMPLE: `SUBSTITUTE([Title],"Doctor","Dr.")`

EXAMPLE: `SUBSTITUTE([Title],"Doctor","Dr.", 2)`

In DirectQuery mode, you can use only the version of this function that has three (3) parameters: a reference to a column, the old text, and the new text. If you use the second formula, an error is raised.

In in-memory models, you can use an optional fourth parameter to specify the instance number of the string to replace. For example, you can replace only the second instance,

etc.

Restrictions on string lengths for REPT operations

In in-memory models, the length of a string resulting from an operation using REPT must be less than 32,767 characters.

This limitation does not apply in DirectQuery mode.

Substring operations return different results depending on character type

EXAMPLE: `MID([col], 2, 5)`

If the input text is **varchar** or **nvarchar**, the result of the formula is always the same.

However, if the text is a fixed-length character and the value for <num_chars> is greater than the length of the target string, in DirectQuery mode, a blank is added at the end of the result string.

In an in-memory model, the result terminates at the last string character, with no padding.

Functions Supported in DirectQuery Mode

The following DAX functions can be used in DirectQuery mode, but with the qualifications as described in the preceding section.

Text functions

CONCATENATE

FIND

LEFT

LEN

MID

REPLACE

REPT

RIGHT

SUBSTITUTE

TRIM

Statistical functions

COUNT

STDEV.P

STDEV.S

STDEVX.P

STDEVX.S

VAR.P

VAR.S

VARX.P

VARX.S

Date/time functions

DATE

EDATE

EOMONTH

DATE

TIME

SECOND

Math and number functions

CEILING

LN

LOG

LOG10

POWER

DAX Table queries

There are some limitations when you evaluate formulas against a DirectQuery model by using DAX Table queries. DirectQuery does not support referring to the same column twice in an ORDER BY clause. The equivalent Transact-SQL statement cannot be created and the query fails.

In an in-memory model, repeating the ORDER by clause has no effect on the results.

Functions Not Supported in DirectQuery Mode

Some DAX functions are not supported in models that are deployed in DirectQuery mode. The reasons that a particular function is not supported can include any or a combination of these reasons:

- The underlying relational engine cannot perform calculations equivalent to those performed by the xVelocity engine.
- The formula cannot be converted to en equivalent SQL expression.
- The performance of the converted expression and the resulting calculations would be unacceptable.

The following DAX functions cannot be used in DirectQuery models.

Path functions

PATH

PATHCONTAINS

PATHITEM

PATHITEMREVERSE

PATHLENGTH

Misc functions

COUNTBLANK

FIXED

FORMAT

RAND

RANDBETWEEN

Time intelligence functions: Start and end dates

DATESQTD

DATESYTD

DATESMTD

DATESQTD

DATESINPERIOD

TOTALMTD

TOTALQTD

TOTALYTD

DATESINPERIOD

SAMEPERIODLASTYEAR

PARALLELPERIOD

Time intelligence functions: Balances

OPENINGBALANCEMONTH

OPENINGBALANCEQUARTER

OPENINGBALANCEYEAR

CLOSINGBALANCEMONTH

CLOSINGBALANCEQUARTER

CLOSINGBALANCEYEAR

Time intelligence functions: Previous and next periods

PREVIOUSDAY

PREVIOUSMONTH

PREVIOUSQUARTER

PREVIOUSYEAR

NEXTDAY

NEXTMONTH

NEXTQUARTER

NEXTYEAR

Time intelligence functions: Periods and calculations over periods

STARTOFMONTH
STARTOFQUARTER
STARTOFYEAR
ENDOFMONTH
ENDOFQUARTER
ENDOFYEAR
FIRSTDATE
LASTDATE
DATEADD

See Also

Direct Query Intro