B

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 scenarios. The DAX reference content in this appendix is obtained from Microsoft documentation. In this appendix you will learn about the following:

- ➤ DAX Syntax Specification
- ➤ Operator Reference
- ➤ Function Reference

DAX SYNTAX SPECIFICATION

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

A DAX formula always starts with an equals 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 (+,-,*, /, >=,...,&&, and so on)

➤ 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.

TABLE B-1: DAX 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]]	Three percent of the value is the Amount column of the current table.

FORMULA	RESULT
=0.03 * [Amount]	Although this formula can be used to calculate a percentage, the result is not shown as a percentage unless you apply formatting in the table.
=PI()	The value of the constant pi.

> *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.*

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 VertiPaq engine. 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 example, '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.

> *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 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:

TABLE B-2: Examples of 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.

To make it easier to enter the fully qualified names of columns, we recommend that you use the AutoComplete feature when building formulas.

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. You will learn about each function and usage in the "DAX Functions" section in this chapter.

➤ 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 the "DAX Operator Reference" section.

TABLE B-3: DAX Operators

OPERATOR TYPE	SYMBOL AND USE
Parenthesis operator	() (precedence order and grouping of arguments)
Arithmetic operators	+ (addition)
	- (subtraction/negation)
	* (multiplication)
	/ (division)

OPERATOR TYPE	SYMBOL AND USE
	^ (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)
	! (negation)

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.

> *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."

DAX OPERATOR REFERENCE

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:

TABLE B-4: Arithmetic Operator Examples

ARITHMETIC OPERATOR	MEANING	EXAMPLE
+ (plus sign)	Addition	3+3
- (minus sign)	Subtraction	3-1
	Negation	-1
* (asterisk)	Multiplication	3*3
/ (forward slash)	Division	3/3
^ (caret)	Exponentiation	16^4

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.

TABLE B-5: Comparison Operators Examples

COMPARISON OPERATOR	MEANING	EXAMPLE
=	Equal to	[Region]="USA"
>	Greater than	[Sales Date] > "Jan 2009"
<	Less than	[Sales Date] < "Jan 2009"
>=	Greater than or equal to	
<=	Less than or equal to	
<>	Not equal to	

Text Concatenation Operator

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

TABLE B-6: Text Concatenation Operator Example

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.

TABLE B-7: Logical Operators Examples

LOGICAL 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")

TABLE B-7 *(continued)*

LOGICAL OPERATOR	MEANING	EXAMPLES
`||` (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"))`
`!(NOT)`	Returns the complement of the condition defined by the expression that follows.	`!([Region] = "U.S.A."`

Operators and Precedence Order

In some cases, the order in which a 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 equals sign (=). The equals sign indicates that the succeeding characters constitute an expression.

Following the equals 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.

TABLE B-8: Operator Precedence

OPERATOR	DESCRIPTION
^	Exponentiation
-	Negation (as in -1)
* and /	Multiplication and division
!	NOT (unary operator)

OPERATOR	DESCRIPTION
+ 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 is 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 negation operator is applied. The result for this expression is -4.

    =-2^2

To ensure that the negation 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 into a common type. For this reason, you may see different results in DAX than in Excel for the same comparison expression.

TABLE B-9: Data Types supported in DAX and Excel

DATA TYPES USED IN DAX	DATA TYPES USED IN EXCEL
Numbers (I8, R8)	Numbers (R8)
Boolean	Boolean
String	String
DateTime	Variant
Currency	Currency

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.

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 and its uses in a PowerPivot workbook please refer to Chapter 4. In this section you will learn the DAX functions in the seven broad categories of DAX functions.

Date and Time Functions (DAX)

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.

DATE Function (DAX)

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 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.

continues

(continued)

TERM	DEFINITION
	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 the 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` 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, `PowerPivot` date functions always return a `datetime` data type. However, you can use formatting to display dates as serial numbers if you want.

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 of 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)
```

DAY Function (DAX)

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

A number indicating the day of the month (I8).

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("3-4-2007")
=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","")
```

DATEVALUE Function (DAX)

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")
```

EDATE Function (DAX)

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.

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)
```

EOMONTH Function (DAX)

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). An appropriate error message is returned when there is an exception.

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.

Example

Description:

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

Code:

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

HOUR Function (DAX)

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

A number from 0 to 23 (I8). An appropriate error message is returned when there is an exception.

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 countries in the world 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])
```

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")
```

MINUTE Function (DAX)

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

A number from 0 to 59 (I8). An appropriate error message is returned when there is an exception.

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 countries 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])
```

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")
```

MONTH Function (DAX)

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

A number from 1 to 12 (I8). An appropriate error message is returned when there is an exception.

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")
```

Description:

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

Code:

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

NOW Function (DAX)

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
```

SECOND Function (DAX)

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

A number from 0 to 59 (I8). An appropriate error message is returned when there is an exception.

Remarks

In contrast to Microsoft Excel, which stores dates and times as serial numbers, DAX uses a `date-time` 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 if 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"`) or as a DAX expression. 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 countries in the world 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])
```

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")
```

TIME Function (DAX)

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 remainder 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.

Example

Description:

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

Code:

```
=TIME(27,0,0)
=TIME(3,0,0)
```

Description:

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

Code:

```
=TIME(0,750,0)
=TIME(12,30,0)
```

Description:

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

Code:

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

TIMEVALUE Function (DAX)

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 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 countries in the world 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 time 8:45:30 PM:

Code:

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

TODAY Function (DAX)

Returns the current date.

Syntax

```
TODAY()
```

Return Value

A date (datetime). An appropriate error message is returned when there is an exception.

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.

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.

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.

WEEKNUM Function (DAX)

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

TERM	DEFINITION
date	The date in `datetime` format.
return_type	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.
	1 — Week begins on Sunday. Weekdays are numbered 1 through 7.
	2 — Week begins on Monday. Weekdays are numbered 1 through 7.

Return Value

A number (I8). An appropriate error message is returned when there is an exception.

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)
```

Description:

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

Code:

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

WEEKDAY Function (DAX)

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

TERM	DEFINITION
date	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.
return_type	A number that determines the return value.

Return Value

A number from 1 to 7 (I8). An appropriate error message is returned when there is an exception.

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)
```

YEAR Function (DAX)

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 (I8).

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 the results of other formulas or functions. You can also enter dates in accepted text representations of a date, such as March 3, 2003, 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 2003.

Code:

```
=YEAR("March 2003")
```

Description:

The following example shows of a return type date which is a result of a DAX expression. The following expression returns the year for today's date.

Code:

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

YEARFRAC Function (DAX)

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.
	0 — US (NASD 30/360
	1 — Actual/actual
	2 — Actual/360
	3 — Actual/365
	4 — European 30/360

Return Value

A decimal number. The internal data type is a signed IEEE 64-bit (8-byte) double-precision floating-point number (R8).

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])
```

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 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.

Filter Functions (DAX)

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.

ALL Function (DAX)

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.

TABLE B-10: Scenarios for ALL and ALLEXCEPT

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. The first example demonstrates this scenario.
	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.
`ALL (Column[, Column[, …]])`	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 PivotTable. 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 RATIO	COLUMN LABELS				
ROW LABELS	ACCESSORIES	BIKES	CLOTHING	COMPONENTS	GRAND TOTAL
2001	0.02%	9.10%	0.04%	0.75%	9.91%
2002	0.11%	24.71%	0.60%	4.48%	29.90%
2003	0.36%	31.71%	1.07%	6.79%	39.93%
2004	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:

➤ 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`.

➤ For the denominator, you start by specifying a table, `ResellerSales_USD`, and use the `ALL` function to remove all context filters on the table.

➤ 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 reseller sales.

The above example uses the tables, `ResellerSales_USD`, `DateTime`, and `ProductCategory` from the DAX sample workbook.

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**
2001	3.48%	11.03%	1.91%	5.12%	9.91%
2002	16.21%	29.96%	27.29%	30.59%	29.90%
2003	51.62%	38.45%	48.86%	46.36%	39.93%
2004	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:

➤ 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 columns `CalendarYear` and `ProductCategoryName`.

➤ 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.

This example uses the tables, `ResellerSales_USD`, `DateTime`, and `ProductCategory` from the DAX sample workbook.

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 *n*, and then divide the resulting value by the sum of sales for the year *n* 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**
2001	0.25%	91.76%	0.42%	7.57%	100.00%
2002	0.38%	82.64%	1.99%	14.99%	100.00%
2003	0.90%	79.42%	2.67%	17.01%	100.00%
2004	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:

➤ 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 to the fields, CalendarYear and ProductCategoryName.

➤ 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.

This example uses the tables ResellerSales_USD, DateTime, and ProductCategory from the DAX sample workbook.

ALLEXCEPT Function (DAX)

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.

FUNCTION AND USAGE	DESCRIPTION
	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.

Example

Description:

The following example presents a formula that you can use in 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.

The above example uses the tables `ResellerSales_USD` and `DateTime` from the DAX sample workbook.

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()`.

Results with no filters:

The table below shows the results when a new measure, `All Sales Yearly`, is created using the example formula. To see how this works, add the field `CalendarQuarter` to the `Row Labels` area of the PivotTable, and add the field `ProductCategoryName` to the `Column Labels` area. Then, add the new measure, `All Sales Yearly`, to the `Values` area of the PivotTable.

ALL SALES YEARLY	COLUMN LABELS				
ROW LABELS	ACCESSORIES	BIKES	CLOTHING	COMPONENTS	GRAND TOTAL
1	$534,301.99	$63,084,675.05	$1,669,943.26	$11,205,837.96	$76,494,758.25
2	$534,301.99	$63,084,675.05	$1,669,943.26	$11,205,837.96	$76,494,758.25
3	$534,301.99	$63,084,675.05	$1,669,943.26	$11,205,837.96	$76,494,758.25
4	$534,301.99	$63,084,675.05	$1,669,943.26	$11,205,837.96	$76,494,758.25
Grand Total	**$534,301.99**	**$63,084,675.05**	**$1,669,943.26**	**$11,205,837.96**	**$76,494,758.25**

The currency format is obtained by applying currency number formatting to `All Sales Yearly`.

The table below shows how the results differ when the measure `All Sales Yearly` is used in the PivotTable, but `CalendarQuarter` is replaced by `CalendarYear` in the `Row Labels`. The context filters on `CalendarYear` that are created by the PivotTable are preserved in the results.

ALL SALES YEARLY	COLUMN LABELS				
ROW LABELS	ACCESSORIES	BIKES	CLOTHING	COMPONENTS	GRAND TOTAL
2001	$18,594.48	$6,958,251.04	$31,851.16	$574,256.99	$7,582,953.67
2002	$86,612.75	$18,901,351.08	$455,730.97	$3,428,213.05	$22,871,907.85
2003	$275,794.84	$24,256,817.51	$815,853.29	$5,195,315.22	$30,543,780.85
2004	$153,299.92	$12,968,255.41	$366,507.84	$2,008,052.70	$15,496,115.88
Grand Total	**$534,301.99**	**$63,084,675.05**	**$1,669,943.26**	**$11,205,837.96**	**$76,494,758.25**

These results demonstrate how you can create totals but selectively keep filters on a column. You use ALLEXCEPT to remove filters in general, and add back in the columns that you want to keep as filters by using the column names as arguments to the ALLEXCEPT function.

ALLNOBLANKROW Function (DAX)

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	DESCRIPTION
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.
ALLEXCEPT(Table,Col1,Col2...)	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
2001
2002
2003
2004
Grand Total

Note the blank label between 2004 and Grand Total. This blank label represents the Unknown member, which is a special group created by PowerPivot 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 ALLNOBLANK* — 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 the `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
2001	1280	1281
2002	1280	1281
2003	1280	1281
2004	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, 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
2001	60856	60856
2002	60856	60856
2003	60856	60856
2004	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'))
```

BLANK Function (DAX)

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.

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.

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[SalesAmount_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
2001		2.65		2.89
2002		3.33		4.03
2003	1.04	2.92	6.63	3.51
2004	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.

CALCULATE Function (DAX)

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 calculates 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 RATIO	COLUMN LABELS				
ROW LABELS	ACCESSORIES	BIKES	CLOTHING	COMPONENTS	GRAND TOTAL
2001	0.02%	9.10%	0.04%	0.75%	9.91%
2002	0.11%	24.71%	0.60%	4.48%	29.90%
2003	0.36%	31.71%	1.07%	6.79%	39.93%
2004	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.

CALCULATETABLE Function (DAX)

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 2002. This value is later used to calculate the ratio of Internet sales compared to all sales for the year 2002.

The following table shows the results from the following formula.

ROW LABELS	INTERNET SALESAMOUNT_USD	CALCULATETABLE 2002 INTERNET SALES	INTERNET SALES TO 2002 RATIO
2001	$2,627,031.40	$5,681,440.58	0.46
2002	$5,681,440.58	$5,681,440.58	1.00
2003	$8,705,066.67	$5,681,440.58	1.53
2004	$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]=2002)
    , [SalesAmount_USD])
```

DISTINCT Function (DAX)

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.

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. If you want to prevent filtering from affecting the items in the list, use the ALL function to remove filters from the specified column and table, like this:

```
=COUNTROWS(DISTINCT(ALL( InternetSales_USD[CustomerKey])))
```

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 contexts, 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**
2001		1013		1013
2002		2677		2677
2003	6792	4875	2867	9309
2004	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 2003 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.

EARLIER Function (DAX)

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 PowerPivot you can store the value of the input and then make calculations 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 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 100 operations could be required; if you have 100 rows then close to 10,000 operations might be performed.

In practice, the VertiPaq engine 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.

PRODUCTSUB-CATEGORYKEY	ENGLISHPRODUCT-SUBCATEGORYNAME	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
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

PRODUCTSUB-CATEGORYKEY	ENGLISHPRODUCT-SUBCATEGORYNAME	TOTALSUBCATEGORYSALES	SUBCATEGORYRANKING
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 values (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[TotalSubcategorySales]))+1
```

Comments:

The following steps describe the method of calculation in more detail.

➤ 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.

➤ EARLIER([TotalSubcategorySales]) evaluates to $156,167.88, the current row in the outer loop.

➤ 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).

➤ The COUNTROWS function counts the rows of the filtered table and assigns that value to the newly calculated column in the current row plus 1. Adding 1 is needed to prevent the top ranked value from become a Blank.

➤ 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.

EARLIEST Function (DAX)

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>)
```

FILTER Function (DAX)

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.

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

ROW LABELS	INTERNET SALES	NON-USA INTERNET SALES
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.

Comparing non-U.S. sales by product categories

NON-USA INTERNET SALES	COLUMN LABELS			
ROW LABELS	ACCESSORIES	BIKES	CLOTHING	GRAND TOTAL
2001		$1,526,481.95		$1,526,481.95
2002		$3,554,744.04		$3,554,744.04
2003	$156,480.18	$5,640,106.05	$70,142.77	$5,866,729.00
2004	$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])
```

FIRSTNONBLANK Function (DAX)

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.

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.

RELATED Function (DAX)

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 Chapter 4.

When the RELATED function performs a lookup, it examines all values in the specified table regardless of any filters that may have been applied.

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.

If the example does not work, you might need to create a relationship between the 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
2001		$1,526,481.95		$1,526,481.95
2002		$3,554,744.04		$3,554,744.04
2003	$156,480.18	$5,640,106.05	$70,142.77	$5,866,729.00
2004	$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**

RELATEDTABLE Function (DAX)

Evaluates a table expression in a context modified by the given filters.

Syntax

```
RELATEDTABLE(<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 table or an expression 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 RELATEDTABLE 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 CALCULATETABLE function.

Example

Description:

The following example uses the RELATEDTABLE function to get Internet Sales for 2002; this value is later used to calculate a ratio of sales compared to the sales in year 2002.

The following table shows the results of using the code shown here.

ROW LABELS	INTERNET SALESAMOUNT_USD	RELATEDTABLE 2002 INTERNET SALES	INTERNET SALES TO 2002 RATIO
2001	$2,627,031.40	$5,681,440.58	0.46
2002	$5,681,440.58	$5,681,440.58	1.00
2003	$8,705,066.67	$5,681,440.58	1.53
2004	$9,041,288.80	$5,681,440.58	1.59
Grand Total	**$26,054,827.45**	**$5,681,440.58**	**4.59**

Code:

```
= SUMX( RELATEDTABLE('InternetSales_USD', 'DateTime'[CalendarYear]=2002)
    , [SalesAmount_USD])
```

VALUES Function (DAX)

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.

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 to be returned.

Return Value

A column of unique values. An appropriate error message is returned when there is an exception.

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]))
```

Information Functions (DAX)

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.

ISBLANK Function (DAX)

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.

Property Value/Return Value

TRUE if the value is blank; otherwise FALSE (BOOL).

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
2001	$10,209,985.08		
2002	$28,553,348.43	$10,209,985.08	179.66%
2003	$39,248,847.52	$28,553,348.43	37.46%
2004	$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])
```

ISLOGICAL Function (DAX)

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 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")
```

ISNONTEXT Function (DAX)

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.

Property Value/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")
```

ISNUMBER Function (DAX)

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")
```

ISERROR Function (DAX)

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.

Property Value/Return Value

TRUE if the value is an error; otherwise FALSE. (BOOL)

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])
      )
```

ISTEXT Function (DAX)

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")
```

Logical Functions (DAX)

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.

AND Function (DAX)

Checks whether all arguments are TRUE, and returns TRUE if all arguments are TRUE.

Syntax

```
AND(<logical1>,<logical2>,…)
```

Parameters

TERM	DEFINITION
`logical_1, logical_2, …`	The logical values you want to test.

Property Value/Return Value

Returns `True` or `False` depending on the combination of values that you test.

Example

Description:

The following formula shows the syntax of the AND function.

Code:

```
=IF(AND( 10 > 9, –10 < –1, true), "All true", "One or more false"
```

Comments:

Because all three conditions passed as arguments to the ALL function are true, the formula returns `"All True"`.

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	2001	2002	2003	2004	GRAND TOTAL
Bib-Shorts					
Bike Racks					
Bike Stands				Internet Hit	
Bottles and Cages				Internet Hit	
Bottom Brackets					
Brakes					
Caps					
Chains					
Cleaners					

AND FUNCTION	COLUMN LABELS				
ROW LABELS	2001	2002	2003	2004	GRAND TOTAL
Cranksets					
Derailleurs					
Fenders				Internet Hit	
Forks					
Gloves					
Handlebars					
Headsets					
Helmets					
Hydration Packs					
Jerseys					
Lights					
Locks					
Mountain Bikes					
Mountain Frames					
Panniers					
Pedals					
Pumps					
Road Bikes					
Road Frames					
Saddles					
Shorts					
Socks					
Tights					
Tires and Tubes				Internet Hit	

continues

(continued)

AND FUNCTION	COLUMN LABELS				
ROW LABELS	2001	2002	2003	2004	GRAND TOTAL
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"
        , ""
        )
```

FALSE Function (DAX)

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 200,000.

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	**2001**	**2002**	**2003**	**2004**		**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())
```

IFERROR Function (DAX)

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`.

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 is 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)
```

IF Function (DAX)

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.

Example

Description:

The following example uses nested IF functions that evaluate the number in the column, Calls, from the table FactCallCenter in ssAWDWsp. 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 ssAWDWsp. 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.

NOT Function (DAX)

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.

OR Function (DAX)

Checks whether one of the arguments is TRUE to return TRUE. The function returns FALSE if all 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 arguments is TRUE; the value is FALSE if all the arguments are FALSE.

Remarks

The function evaluates the arguments until the first TRUE argument, then returns TRUE.

Example

SALESPERSONFLAG	TRUE				
OR FUNCTION	COLUMN LABELS				
ROW LABELS	2001	2002	2003	2004	GRAND TOTAL
Abbas, Syed E					
Alberts, Amy E					
Ansman-Wolfe, Pamela O					
Blythe, Michael G	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
Ito, Shu K					
Jiang, Stephen Y					
Mensa-Annan, Tete A					

SALESPERSONFLAG	TRUE				
OR FUNCTION	COLUMN LABELS				
ROW LABELS	2001	2002	2003	2004	GRAND TOTAL
Mitchell, Linda C	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
Reiter, Tsvi Michael					
Saraiva, José Edvaldo	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**

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 2003.

Code:

```
IF(   OR(   CALCULATE(SUM('ResellerSales_USD'[SalesAmount_USD]),
       'ProductSubcategory'[ProductSubcategoryName]="Touring Bikes") > 1000000
           ,    CALCULATE(SUM('ResellerSales_USD'[SalesAmount_USD]),
                  'DateTime'[CalendarYear]=2003) > 2500000
           )
    , "Circle of Excellence"
    , ""
    )
```

TRUE Function (DAX)

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 200,000.

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	2001	2002	2003	2004		GRAND TOTAL
Accessories	FALSE	FALSE	TRUE	TRUE	FALSE	TRUE
Bikes	TRUE	TRUE	TRUE	TRUE	FALSE	TRUE
Clothing	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE

(continued)

TRUE-FALSE	COLUMN LABELS					
ROW LABELS	2001	2002	2003	2004		GRAND TOTAL
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())
```

Math and Trigonometric Functions (DAX)

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.

ABS Function (DAX)

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 number (R8). An appropriate error message is returned when there is an exception.

Remarks

The absolute value of a number is a real 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])
```

CEILING Function (DAX)

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 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)
```

Description:

The following formula returns similar results to the previous example, but uses numeric values stored in the column ProductPrice.

Code:

```
=CEILING([ProductPrice],0.05)
```

EXP Function (DAX)

Returns *e* 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 *e*. The constant *e* equals 2.71828182845904, the base of the natural logarithm.

Return Value

A number (R8).

Remarks

EXP is the inverse of LN, which is the natural logarithm of the given number.

To calculate powers of bases other than *e,* use the exponentiation operator (^). For more information, see the "DAX Operators and Constants" section.

Example

Description:

The following formula calculates *e* raised to the power of the number contained in the column, [Power].

Code:

```
=EXP([Power])
```

FACT Function (DAX)

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 number (I8). An appropriate error message is returned when there is an exception.

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

FLOOR Function (DAX)

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 number (R8). An appropriate error message is returned when there is an exception.

Remarks

If either argument is nonnumeric, FLOOR returns an error.

If number and significance have different signs, FLOOR returns an error.

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

INT Function (DAX)

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 non-negative number for which you want to calculate the factorial.

Return Value

A number (I8). An appropriate error message is returned when there is an exception.

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)
```

ISO.CEILING Function (DAX)

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 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 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 of 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)
```

LN Function (DAX)

Returns the natural logarithm of a number. Natural logarithms are based on the constant e (2.71828182845904).

Syntax

```
LN(<number>)
```

Parameters

TERM	DEFINITION
number	The positive number for which you want the natural logarithm.

Return Value

A number (R8). An appropriate error message is returned when there is an exception.

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])
```

LOG Function (DAX)

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 number (R8). An appropriate error message is returned when there is an exception.

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)
```

LOG10 Function (DAX)

Returns the base-10 logarithm of a number.

Syntax

```
LOG10(<number>)
```

Parameters

TERM	DEFINITION
number	A positive number for which you want the base-10 logarithm.

Property Value/Return Value

A number (R8). An appropriate error message is returned when there is an exception.

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)
```

MOD Function (DAX)

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 number (I8). An appropriate error message is returned when there is an exception.

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:

```
MOD(n, d) = n - d*INT(n/d)
```

Example

Description:

The following formula returns 1, the remainder of 3 divided by 2.

Code:

```
=MOD(3,2)
```

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)
```

MROUND Function (DAX)

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 want to round the number.

Return Value

A number (R8). An appropriate error message is returned when there is an exception.

Remarks

MROUND rounds up, away from zero, if the remainder of the 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)
```

PI Function (DAX)

Returns the value of Pi, 3.14159265358979, accurate to 15 digits.

Syntax

```
PI()
```

Return Value

A number (R8).

Remarks

Pi is a mathematical constant. In PowerPivot, 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)
```

POWER Function (DAX)

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 number (R8). An appropriate error message is returned when there is an exception.

Remarks

Instead of using the comma and the second argument, power, you can use the exponentiation operator between the numbers like a usual mathematical equation.

Example

Description:

The following example returns 25.

Code:

```
=POWER(5,2)
```

QUOTIENT Function (DAX)

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 number (I8). An appropriate error message is returned when there is an exception.

Remarks

If either argument is non-numeric, QUOTIENT returns an error.

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)
```

RAND Function (DAX)

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()
```

Property Value/Return Value

A number (R8).

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.

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 refreshing 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.

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
```

RANDBETWEEN Function (DAX)

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.

Property Value/Return Value

A number (I8)

Example

Description:

The following formula returns a random number between 1 and 10.

Code:

```
=RANDBETWEEN(1,10)
```

ROUND Function (DAX)

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 number (R8). An appropriate error message is returned when there is an exception.

Remarks

If num_digits is greater than 0 (zero), then the 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)
```

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)
```

ROUNDDOWN Function (DAX)

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 number (R8). An appropriate error message is returned when there is an exception.

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)
```

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)
```

ROUNDUP Function (DAX)

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 number (R8). An appropriate error message is returned when there is an exception.

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

SIGN Function (DAX)

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 number (I8). The possible return values are 1, 0, and -1. An appropriate error message is returned when there is an exception.

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]) )
```

SQRT Function (DAX)

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 number (R8). An appropriate error message is returned when there is an exception.

Remarks

If the number is negative, the SQRT function returns an error.

Example

Description:

The following formula returns 5.

Code:

```
=SQRT(25)
```

SUM Function (DAX)

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 number (R8). An appropriate error message is returned when there is an exception.

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])
```

SUMX Function (DAX)

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 number (R8).

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.

TRUNC Function (DAX)

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 truncation, 0 (zero) if omitted.
	A number specifying the precision of the truncation; if omitted, 0 (zero).

Return Value

A number (I8). An appropriate error message is returned when there is an exception.

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())
```

Description:

The following formula returns -8, the integer part of -8.9.

Code:

```
=TRUNC(-8.9)
```

Statistical Functions (DAX)

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.

AVERAGE Function (DAX)

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 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 function 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 for a column that contains empty values.

AVERAGEA Function (DAX)

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 number (R8). An appropriate error message is returned when there is an exception.

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 evaluate 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])
```

AVERAGEX Function (DAX)

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 number (R8). An appropriate error message is returned when there is an exception.

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.

COUNT Function (DAX)

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 number (I8). An appropriate error message is returned when there is an exception.

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. To count the number of distinct values in a column use the DISTINCT function.

COUNTA Function (DAX)

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 number (I8). An appropriate error message is returned when there is an exception.

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])
```

COUNTAX Function (DAX)

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 result 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 row of the table.

Return Value

A number (I8). An appropriate error message is returned when there is an exception.

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])
```

COUNTBLANK Function (DAX)

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 number (I8). If no rows are found that meet the condition, blanks are returned. An appropriate error message is returned when there is an exception.

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.

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.

COUNTROWS Function (DAX)

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 number (I8). An appropriate error message is returned when there is an exception.

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')
```

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

COUNTX Function (DAX)

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. An appropriate error message is returned when there is an exception.

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])
```

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])
```

MAX Function (DAX)

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 number (R8). An appropriate error message is returned when there is an exception.

Remarks

The MAX function takes as an argument a column that evaluates to numeric values (numbers or dates). If the column contains no numbers, MAX returns a blank. If you want to evaluate values that are not numbers, use the MAXA function. The MAX function does not take a DAX expression as an argument.

Example

Description:

The following example returns the largest value found in the ExtendedAmount column of the InternetSales table.

Code:

```
=MAX(InternetSales[ExtendedAmount])
```

MAXA Function (DAX)

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.

Property Value/Return Value

A number (R8). An appropriate error message is returned when there is an exception.

Remarks

The MAXA function takes as argument a column, and looks for the largest value among the following types of values:

➤ Numbers or Text representation of 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])
```

Description:

The following example returns the largest value from a column that contains dates and times. This formula therefore gets the most recent transaction date.

Code:

```
=MAXA([TransactionDate])
```

MAXX Function (DAX)

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.

Property Value/Return Value

A number (R8). An appropriate error message is returned when there is an exception.

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])
```

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 cost for tax and shipping for just those orders. The expected result is 250.3724.

Code:

```
=MAXX(FILTER(InternetSales,[SalesTerritoryCode]="5"),
      InternetSales[TaxAmt]+ InternetSales[Freight])
```

MIN Function (DAX)

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.

Property Value/Return Value

A number (R8). An appropriate error message is returned when there is an exception.

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])
```

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])
```

MINA Function (DAX)

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.

Property Value/Return Value

A number (R8). An appropriate error message is returned when there is an exception.

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 evaluate 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])
```

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])
```

MINX Function (DAX)

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.

Property Value/Return Value

A number (R8). An appropriate error message is returned when there is an exception.

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])
```

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.

Text Functions (DAX)

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.

CONCATENATE Function (DAX)

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.

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]
```

EXACT Function (DAX)

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.
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])
```

FIND Function (DAX)

Returns the starting position of one text string within another text string. FIND is case-sensitive.

Syntax

```
FIND(<find_text, within_text, start_num)
```

Parameters

TERM	DEFINITION
find_text	The text you want to find. Use double quotes (empty text) to match the first character in within_text; wildcard characters are not allowed.
within_text	The text containing the text you want to find.
start_num	The character at which to start the search; if omitted, start_num = 1. The first character in within_text is character number 1.

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, PowerPivot uses Unicode and counts each character the same way; therefore, you do not need to use a different version depending on the character type.

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")
```

FIXED Function (DAX)

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.

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.

FORMAT Function (DAX)

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`.

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.

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 casing to `"d"` you get the date in the short format. Also, you might get unexpected results or an error if the intended formatting does not match the case of any defined format string.

Pre-Defined Numeric Formats for the FORMAT Function (DAX)

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", "G", or "g"	Displays number with no thousand separators.
"Currency", "C", or "c"	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", "F", or "f"	Displays at least one digit to the left and two digits to the right of the decimal separator.
"Standard", "N", or "n"	Displays number with thousand separators, at least one digit to the left and two digits to the right of the decimal separator.

continues

(continued)

FORMAT SPECIFICATION	DESCRIPTION
"Percent"	Displays number multiplied by 100 with a percent sign (%) appended immediately to the right; always displays two digits to the right of the decimal separator.
"P", or "p"	Displays number with thousandths separator multiplied by 100 with a percent sign (%) appended to the right and separated by a single space; always displays two digits to the right of the decimal separator.
"Scientific"	Uses standard scientific notation, providing two significant digits.
"E", or "e"	Uses standard scientific notation, providing six significant digits.
"D", or "d"	Displays number as a string that contains the value of the number in Decimal (base 10) format. This option is supported for integral types (Byte, Short, Integer, Long) only.
"X", or "x"	Displays number as a string that contains the value of the number in Hexadecimal (base 16) format. This option is supported for integral types (Byte, Short, Integer, Long) only.
"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

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); but, if you change the casing 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.

Example

Description:

The following samples show the usage of different predefined formatting strings to format a numeric value.

Code:

```
FORMAT( 12345.67, "G")
FORMAT( 12345.67, "C")
FORMAT( 12345.67, "F")
FORMAT( 12345.67, "N")
FORMAT( 12345.67, "P")
FORMAT( 12345.67, "Scientific")
```

Comments:

The above expressions return the following results:

➤ `12345.67` — `"G"` displays the number with no formatting.

➤ `$12,345.67` — `"C"` displays the number with your currency locale formatting. The sample here shows the default United States currency formatting.

➤ `12345.67` — `"F"` 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` — `"N"` 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 %` — `"P"` 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.

Pre-Defined Date and Time Formats for the FORMAT Function (DAX)

The following table identifies the predefined date and time format names.

TABLE B-11: Date and Time Formats for FORMAT Function

FORMAT SPECIFICATION	DESCRIPTION
`"General Date"`, or `"G"`	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"`, `"Medium Date"`, or `"D"`	Displays a date according to your current culture's long date format. For example, Wednesday, March 12, 2008.
`"Short Date"`, or `"d"`	Displays a date using your current culture's short date format. For example, 3/12/2008.
`"Long Time"`, `"Medium Time"`, or `"T"`	Displays a time using your current culture's long time format; typically includes hours, minutes, seconds. For example, 11:07:31 AM.
`"Short Time"` or `"t"`	Displays a time using your current culture's short time format. For example, 11:07 AM.
	The `t` character displays AM or PM values for locales that use a 12-hour clock in a user-defined time format. For more information, see "User-Defined Date/Time Formats (Format Function)."
`"f"`	Displays the long date and short time according to your current culture's format. For example, Wednesday, March 12, 2008 11:07 AM.

(continued)

FORMAT SPECIFICATION	DESCRIPTION
"F"	Displays the long date and long time according to your current culture's format. For example, Wednesday, March 12, 2008 11:07:31 AM.
"g"	Displays the short date and short time according to your current culture's format. For example, 3/12/2008 11:07 AM.
"M","m"	Displays the month and the day of a date. For example, March 12. The M character displays the month in a user-defined date format. The m character displays the minutes in a user-defined time format. For more information, see "User-Defined Date/Time Formats (Format Function)."
"R", "r"	Formats the date according to the RFC1123 Pattern property. For example, Wed, 12 Mar 2008 11:07:31 GMT. The formatted date does not adjust the value of the date and time. You must adjust the Date/Time value to GMT before calling the Format function.
"s"	Formats the date and time as a sortable index. For example, 2008-03-12T11:07:31.
	The s character displays the seconds in a user-defined time format. For more information, see "User-Defined Date/Time Formats (Format Function)."
"u"	Formats the date and time as a GMT sortable index. For example, 2008-03-12 11:07:31Z.
"U"	Formats the date and time with the long date and long time as GMT. For example, Wednesday, March 12, 2008 6:07:31 PM.
"Y", "y"	Formats the date as the year and month. For example, March, 2008.
	The Y and y characters display the year in a user-defined date format. For more information, see "User-Defined Date/Time Formats (Format Function)."

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.

Custom Numeric Formats for the FORMAT Function (DAX)

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.

TABLE B-12: Custom Numeric Formats for FORMAT Function

FORMAT SPECIFICATION	DESCRIPTION
None	Displays the number with no formatting.
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.

TABLE B-12 *(continued)*

FORMAT SPECIFICATION	DESCRIPTION
. (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 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 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"`.

FORMAT SPECIFICATION	DESCRIPTION
	`"#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 `#`) 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 (")).

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.

Custom Date and Time Formats for the FORMAT Function (DAX)

The following table shows characters you can use to create user-defined date/time formats.

TABLE B-13: Custom Date and Time Formats for FORMAT Function

TERM	DEFINITION
(:)	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.

TERM	DEFINITION
(%)	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 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.
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.

TERM	DEFINITION
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 three-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.
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 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'.

LEFT Function (DAX)

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, PowerPivot Client works with Unicode and stores all characters as the same length; therefore, a single function is enough.

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.

LEN Function (DAX)

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.

Property Value/Return Value

A number indicating the number of characters in the text string. (I8)

Remarks

Whereas Microsoft Excel has different functions for working with single-byte and double-byte character languages, PowerPivot Client 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)
```

LOWER Function (DAX)

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])
```

MID Function (DAX)

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 character languages, PowerPivot 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)
```

REPLACE Function (DAX)

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.
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, PowerPivot Client uses Unicode and therefore stores all characters as the same length.

Example

Description:

The following formula creates a new calculate column that replaces the first two characters of the product code in the column [ProductCode] with a new two-letter code, OB.

Code:

```
=REPLACE('New Products'[Product Code],1,2,"OB")
```

REPT Function (DAX)

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.
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

RIGHT Function (DAX)

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; if omitted, 1. You can also use a reference to a column that contains numbers.

If the column references do not contain text, they are 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.

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])
```

SEARCH Function (DAX)

Returns the number of the character at which a specific character or text string is first found, reading left to right. Search is case-sensitive.

Syntax

```
SEARCH(<find_text>, <within_text>, [start_num])
```

Parameters

TERM	DEFINITION
find_text	The text that you want to find.
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.

Property Value/Return Value

The number of the starting position of the first text string from the first character of the second text string.

Remarks

By using this function, you can locate one text string within a second text string, and return the position where the first string starts.

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.

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.

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.

SUBSTITUTE Function (DAX)

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.

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")
```

TRIM Function (DAX)

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])
```

UPPER Function (DAX)

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])
```

VALUE Function (DAX)

Converts a text string that represents a number to a number.

Syntax

```
VALUE(<text>)
```

Parameters

TERM	DEFINITION
text	The text to be converted.

Property Value/Return Value

The converted number (R8).

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.

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")
```

Time Intelligence Functions (DAX)

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.

CLOSINGBALANCEMONTH Function (DAX)

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

Syntax

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

Parameters

TERM	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

To understand more about how context affects the results of formulas, see Chapters 4 and 5.

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.

Constraints on Boolean expressions are described in the topic "CALCULATE Function (DAX)." The filter expression has restrictions described in the topic "CALCULATE Function (DAX)."

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]*
    ProductInventory[UnitsBalance]),DateTime[DateKey])
```

CLOSINGBALANCEQUARTER Function (DAX)

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

Syntax

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

Parameters

TERM	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

To understand more about how context affects the results of formulas, see Chapters 4 and 5.

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.

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

The filter expression has restrictions described in the topic "CALCULATE Function (DAX)."

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])
```

CLOSINGBALANCEYEAR Function (DAX)

Evaluates the expression on the last date of the year in the current context.

Syntax

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

Parameters

TERM	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 on the last date of the year in the current context.

Remarks

To understand more about how context affects the results of formulas, see Chapters 4 and 5.

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.

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

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.

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]
        *ProductInventory[UnitsBalance]),DateTime[DateKey])
```

DATEADD Function (DAX)

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

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.

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.

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)
```

DATESBETWEEN Function (DAX)

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.

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.

Example

Description:

The following sample formula creates a measure that calculates the 'Summer 2003 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 2003 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(2003,6,1),
    DATE(2003,8,31)
  ))
```

DATESINPERIOD Function (DAX)

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 the interval can be one of the following: year, quarter, month, day.

Return Value

A table containing a single column of date values.

Remarks

To understand more about how context affects the results of formulas, see Chapters 4 and 5.

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.

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.

Example

Description:

The following formula returns the Internet sales for the 21 days prior to August 24, 2003.

Code:

```
= CALCULATE(SUM(InternetSales_USD[SalesAmount_USD]),
    DATESINPERIOD(DateTime[DateKey], DATE(2003,08,24),-21,day))
```

DATESMTD Function (DAX)

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

To understand more about how context affects the results of formulas, see Chapters 4 and 5.

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.

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

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]))
```

DATESQTD Function (DAX)

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

To understand more about how context affects the results of formulas, see Chapters 4 and 5.

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.

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

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]))
```

DATESYTD Function (DAX)

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

To understand more about how context affects the results of formulas, see Chapters 4 and 5.

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.

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.

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]))
```

ENDOFMONTH Function (DAX)

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

To understand more about how context affects the results of formulas, see Chapters 4 and 5.

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.

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

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])
```

ENDOFQUARTER Function (DAX)

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

To understand more about how context affects the results of formulas, see Chapters 4 and 5.

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.

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

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])
```

ENDOFYEAR Function (DAX)

Introduction

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

To understand more about how context affects the results of formulas, see Chapters 4 and 5.

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.

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.

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")
```

FIRSTDATE Function (DAX)

Introduction

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

To understand more about how context affects the results of formulas, see Chapters 4 and 5.

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.

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.

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])
```

LASTDATE Function (DAX)

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

To understand more about how context affects the results of formulas, see Chapters 4 and 5.

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.

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.

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])
```

LASTNONBLANK Function (DAX)

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.

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.

To understand more about how context affects the results of formulas, see Chapters 4 and 5.

NEXTDAY Function (DAX)

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

To understand more about how context affects the results of formulas, see Chapters 4 and 5.

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.

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

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]))
```

NEXTMONTH Function (DAX)

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

To understand more about how context affects the results of formulas, see Chapters 4 and 5.

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.

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

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]))
```

NEXTQUARTER Function (DAX)

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

To understand more about how context affects the results of formulas, see Chapters 4 and 5.

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.

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

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]))
```

NEXTYEAR Function (DAX)

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

To understand more about how context affects the results of formulas, see Chapters 4 and 5.

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.

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.

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]))
```

OPENINGBALANCEMONTH Function (DAX)

Evaluates the `expression` on the first date of the month in the current context.

Syntax

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

Parameters

TERM	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

To understand more about how context affects the results of formulas, see Chapters 4 and 5.

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.

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

The `filter` expression has restrictions described in the topic "CALCULATE Function (DAX)."

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]
      *ProductInventory[UnitsBalance]),DateTime[DateKey])
```

OPENINGBALANCEQUARTER Function (DAX)

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

Syntax

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

Parameters

TERM	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

To understand more about how context affects the results of formulas, see Chapters 4 and 5.

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.

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

The `filter` expression has restrictions described in the topic "CALCULATE Function (DAX)."

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])
```

OPENINGBALANCEYEAR Function (DAX)

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

Syntax

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

Parameters

TERM	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

To understand more about how context affects the results of formulas, see Chapters 4 and 5.

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.

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

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.

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]
    *ProductInventory[UnitsBalance]),DateTime[DateKey])
```

PARALLELPERIOD Function (DAX)

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.

To understand more about how context affects the results of formulas, see Chapters 4 and 5.

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.

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.

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))
```

PREVIOUSDAY Function (DAX)

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

To understand more about how context affects the results of formulas, see Chapters 4 and 5.

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.

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

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]))
```

PREVIOUSMONTH Function (DAX)

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

To understand more about how context affects the results of formulas, see Chapters 4 and 5.

This function returns all dates from the previous month, using the first date in the column used as input in the current context. 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.

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

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]))
```

PREVIOUSQUARTER Function (DAX)

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

To understand more about how context affects the results of formulas, see Chapters 4 and 5.

This function returns all dates from the previous quarter, using the first date in the input column in the current context. 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.

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

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]))
```

PREVIOUSYEAR Function (DAX)

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

To understand more about how context affects the results of formulas, see Chapters 4 and 5.

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 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.

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.

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]))
```

SAMEPERIODLASTYEAR Function (DAX)

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

To understand more about how context affects the results of formulas, see Chapters 4 and 5.

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.

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)
```

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]))
```

STARTOFMONTH Function (DAX)

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 a single row with a date value.

Remarks

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.

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

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])
```

STARTOFQUARTER Function (DAX)

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 a single row with a date value.

Remarks

To understand more about how context affects the results of formulas, see Chapters 4 and 5.

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.

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

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])
```

STARTOFYEAR Function (DAX)

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

Syntax

```
STARTOFYEAR(<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 a single row with a date value.

Remarks

To understand more about how context affects the results of formulas, see Chapters 4 and 5.

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.

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.

Example

Description:

The following sample formula creates a measure that returns the start 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 `StartOfFiscalYear`, using the formula defined in the code section, to the `Values` area of the PivotTable.

Code:

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

TOTALMTD Function (DAX)

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

Syntax

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

Parameters

TERM	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

To understand more about how context affects the results of formulas, see Chapters 4 and 5.

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.

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

The `filter` expression has restrictions described in the topic "CALCULATE Function (DAX)."

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])
```

TOTALQTD Function (DAX)

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

Syntax

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

Parameters

TERM	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

To understand more about how context affects the results of formulas, see Chapters 4 and 5.

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.

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

The `filter` expression has restrictions described in the topic "CALCULATE Function (DAX)."

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])
```

TOTALYTD Function (DAX)

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

Syntax

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

Parameters

TERM	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 for the current year-to-date `dates`.

Remarks

To understand more about how context affects the results of formulas, see Chapters 4 and 5.

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.

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

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.

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])
```