

# POWER

# BI

# DAX



ABHISHEK DHATUNDE

## Table of Contents

DAX FUNCTIONS	3
Types of DAX Functions	4
DAX parameter naming convention	7
DAX Variables	11
DAX Aggregation Functions	13
DAX Filter Functions	24
DAX Time Intelligence Functions	33
DAX Date and Time Functions	41
DAX Information Functions	44
DAX Logical Functions	47
DAX Math and Trigonometry Functions	57
DAX Statistical Functions	60
DAX Text Functions	64
DAX Other Functions	68

# DAX FUNCTION

A DAX function is an inbuilt function provided in the DAX language to enable you to perform various actions on the data in the tables in your Data Model.

DAX functions enable you to perform commonly used data calculations on the Data Model. Some of the DAX functions have same names and functionality as that of Excel functions but have been modified to use DAX data types and to work with tables and columns, as highlighted in the next section. DAX has additional functions that are designed to work with relational data and perform dynamic aggregation.

DAX functions play an important role in the usage of DAX for data modelling and reporting.

## Excel Functions vs. DAX Functions

There are certain similarities between the Excel functions and the DAX functions and there are certain differences too. Following are the similarities and differences between Excel functions and DAX functions:

### Similarities Between Excel Functions and DAX Functions

- Certain DAX functions have the same name and the same general behaviour as Excel functions.
- DAX has lookup functions that are similar to the array and vector lookup functions in Excel.

### Differences Between Excel Functions and DAX Functions

1. DAX functions have been modified to take different types of inputs and some of the DAX functions might return a different data type. Hence, you need to understand the usage of these functions separately though they have the same name.
2. You cannot use DAX functions in an Excel formula or use Excel functions in DAX formula, without the required modifications.
3. Excel functions take a cell reference or a range of cells as a reference. DAX functions never take a cell reference or a range of cells as a reference, but instead take a column or table as a reference.
4. Excel date and time functions return an integer that represents a date as a serial number. DAX date and time functions return a datetime data type that is in DAX but not in Excel.
5. Excel has no functions that return a table, but some functions can work with arrays. Many of the DAX functions can easily reference complete tables and columns to perform calculations

and return a table or a column of values. This ability of DAX adds power to the Power Pivot, Power View and Power BI, where DAX is used.

## TYPES OF DAX FUNCTIONS

DAX supports the following types of functions.

1. DAX Table-Valued Functions
  - DAX Filter Functions
  - DAX Aggregation Functions
  - DAX Time Intelligence Functions
2. DAX Date and Time Functions
3. DAX Information Functions
4. DAX Logical Functions
5. DAX Math and Trig Functions
6. DAX Other Functions
7. DAX Parent and Child Functions
8. DAX Statistical Functions
9. DAX Text Functions

### DAX Table-Valued Functions

Many DAX functions take tables as input or output tables or do both. These DAX functions are called DAX table-valued functions. Because a table can have a single column, DAX table valued functions also take single columns as inputs. You have the following types of DAX table-valued functions:

- DAX Aggregation functions
- DAX Filter functions
- DAX Time intelligence functions

### DAX Aggregation Functions

DAX Aggregation functions aggregate any expression over the rows of a table and are useful in calculations.

### DAX Filter Functions

DAX Filter functions return a column or a table or values related to the current row. You can use DAX Filter functions to return specific data types, look up values in related tables and filter by related values. DAX Lookup functions work by using tables and relationships between them. DAX Filter functions enable you to manipulate the data context to create dynamic calculations.

## **DAX Time Intelligence Functions**

DAX Time Intelligence functions return a table of dates or the use a table of dates to calculate an aggregation. These DAX functions help you create calculations that support the needs of Business Intelligence analysis by enabling you to manipulate data using time periods, including days, months, quarters, and years.

## **DAX Date and Time Functions**

DAX Date and Time functions are similar to the Excel date and time functions. However, DAX Date and Time functions are based on the datetime data type of DAX.

## **DAX Information Functions**

DAX Information functions look at the cell or row that is provided as an argument and tell you whether the value matches the expected type.

## **DAX Logical Functions**

DAX Logical Functions return information about values in an expression. For example, DAX TRUE function lets you know whether an expression that you are evaluating returns a TRUE value.

## **DAX Math and Trigonometric Functions**

DAX Mathematical and Trigonometric functions are very similar to the Excel mathematical and trigonometric functions.

## **DAX Parent and Child Functions**

DAX Parent and Child functions are useful in managing data that is presented as a parent/child hierarchy in the Data Model.

## **DAX Statistical Functions**

DAX Statistical functions are very similar to the Excel Statistical functions.

## **DAX Text Functions**

DAX Text functions work with tables and columns. With DAX Text functions, you can return part of a string, search for text within a string or concatenate string values. You can also control the formats for dates, times, and numbers.

## **DAX Other Functions**

These DAX functions perform unique actions that cannot be defined by any of the categories most other functions belong to.

## **DAX Function Description Structure**

If you have to use a DAX function in a DAX formula, you need to understand the function in detail. You should know the syntax of the function, the parameter types, what the function returns, etc.

a common function description structure is used for all the DAX functions so that you can read and interpret the DAX functions effectively.

## DAX PARAMETER NAMING CONVENTIONS

DAX has standard parameter names to facilitate the usage and understanding of the DAX functions. Further, you can use certain prefixes to the parameter names. If the prefix is clear enough, you can use the prefix itself as the parameter name.

To understand the syntax of the DAX functions and to use data values appropriately for the relevant DAX function parameters, you need to understand DAX parameter naming conventions.

### Parameter Names

Following are the DAX standard parameter names –

Parameter Name	Description
Expression	Any DAX expression that returns a single scalar value, where the expression is to be evaluated multiple times (for each row/context).
Value	Any DAX expression that returns a single scalar value where the expression is to be evaluated exactly once before all other operations.
Table	Any DAX expression that returns a table of data.
tableName	The name of an existing table using standard DAX syntax. It cannot be an expression.
columnName	The name of an existing column using standard DAX syntax, usually fully qualified. It cannot be an expression.
Name	A string constant that will be used to provide the name of a new object.

order	An enumeration used to determine the sort order.
ties	An enumeration used to determine the handling of tie values.
type	An enumeration used to determine the data type for Path Item and Path Item Reverse.

### Prefixing Parameter Names or Using the Prefix Only

You can qualify a parameter name with a prefix –

1. The prefix should be descriptive of how the argument is used.
2. The prefix should be in such a way that ambiguous reading of the parameter is avoided.

For example,

1. Result\_ColumnName - Refers to an existing column used to get the result values in the DAX LOOKUPVALUE () function.
2. Search\_ColumnName - Refers to an existing column used to search for a value in the DAX LOOKUPVALUE () function.

You can omit the parameter name and use only the prefix, if the prefix is clear enough to describe the parameter. Omitting the parameter name and using only prefix can sometimes help in avoiding the clutter during reading.

For example, Consider **DATE (Year\_value, Month\_value, Day\_value)**. You can omit the parameter name – value, that is repeated thrice and write it as DATE (Year, Month, Day). As seen, by using only the prefixes, the function is more readable. However, sometimes the parameter name and the prefix have to be present for clarity.

For example, Consider **Year\_columnName**. The parameter name is ColumnName and the prefix is Year. Both are required to make the user understand that the parameter requires a reference to an existing column of years.

If you have to use a DAX function in a DAX formula, you need to understand the function in Detail. You should know the syntax of the function, the parameter types, what the function returns, etc.

To enable you to understand how to read and interpret the DAX functions, a uniform function description structure is used in this tutorial.



1. The different types of DAX functions are grouped by the type name of the DAX functions as chapters.
2. Each of these chapters provides a brief description of the utility of the respective type of DAX functions.
3. The brief description will be followed by the list of DAX functions corresponding to that chapter (Type/Category of DAX functions).
4. Each DAX function name is hyperlinked to DAX function details that have the following

### DAX function description structure:

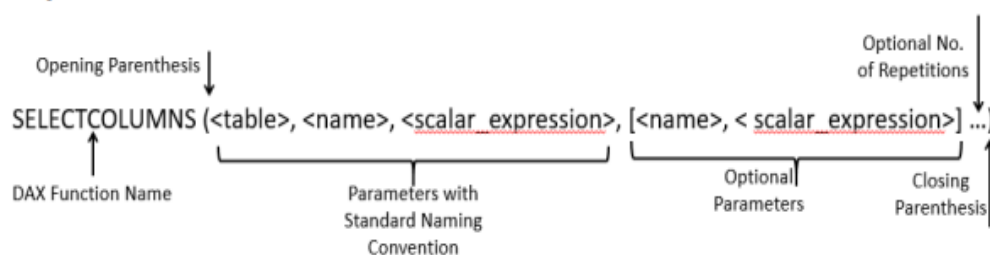
- Description
- Syntax
- Parameters
- Return Value
- Remarks
- Example

### Syntax

In the Syntax section, you will learn the exact function name and the respective parameters.

- DAX function name is given in UPPERCASE letters.
- DAX function name is followed by opening parenthesis.
- Each parameter follows standard DAX parameter naming convention and is enclosed in angle brackets.
- If a parameter is optional, it is further enclosed in square brackets.
- The parameters are separated by commas.
- Ellipses ... are used to show an optional number of repetitions of parameters.
- The function syntax ends with closing parenthesis.
- 

### Example



## Parameters

In the Parameters section, each of the parameters of the specific DAX function is listed in a table with its description. For example, the parameters of the above example DAX function `SELECTCOLUMNS` is listed in the following table.

Parameter	Description
Table	Table or a DAX expression that returns a table.
Name	The name given to the column, enclosed in double quotes.
scalar expression	DAX expression that returns a scalar value like a column reference, integer or string value.

## Return Value

In the Return Value section, you will learn about what value the DAX function will return and its data type.

### Example

An example of the usage of the DAX function is given in this section.

Note: When you write DAX functions with the data values for the parameters, you will follow the naming conventions as given below:

- A Table name is specified as it appears in the Data Model. E.g. Sales.
- A Column name is specified as it appears in the Data Model with square brackets enclosing it.

For example, `[Sales Amount]`

- It is recommended to use fully qualified names for columns, i.e. a column name is prefixed with the table name that contains it.

For example, `Sales[Sales Amount]`.

- If the table name contains spaces, it should be enclosed in single quotes.

For example, 'East Sales'[Sales Amount]

- A DAX function can return a column or table of values, in which case, it needs to be used as a parameter of another DAX function that requires a column or table.

## DAX VARIABLES

- Variables can be defined by using keyword VAR.
- Can be defined within expression.
- If VAR is declared in a measure then we will have to use Return keyword.

### Example

#### **Total Sales =**

VAR S = SUM(Orders [Sales])

Return IF(ISBLANK(S), 0, S)

#### **Test Measure=**

VAR A = 10

VAR B = 15

RETURN A+B

#### **Test Measure=**

VAR A = SUM(orders[sales])

VAR B = SUM(orders[profit])

RETURN A+B

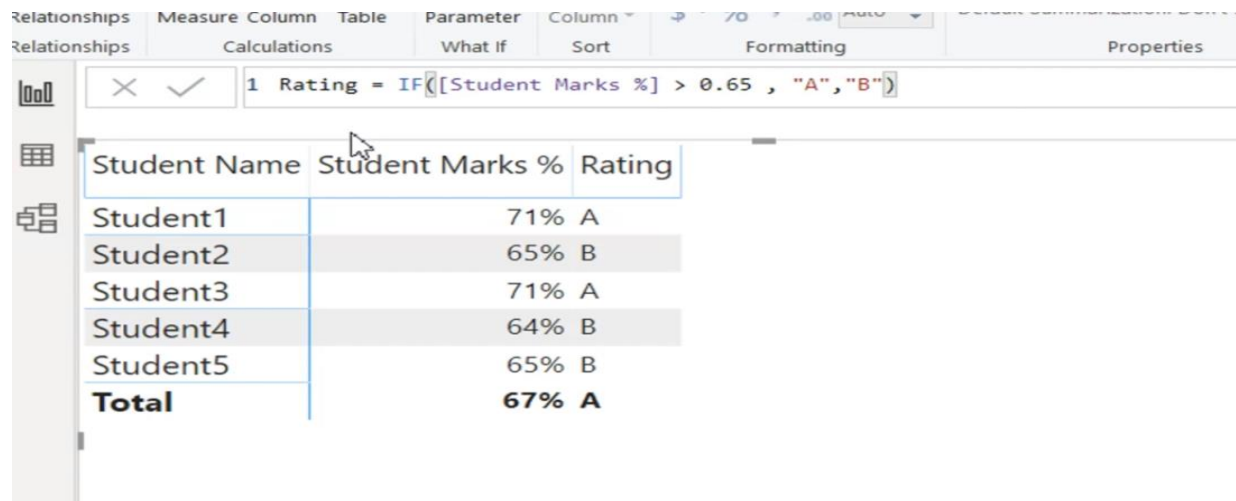
#### **Total Sales =**

VAR A = SUM(Orders[Sales])

VAR B = SUM(Orders[Profit])

RETURN IF(A-B>=0, A-B, 0)

EX. Rating = IF([Student Marks %] > 0.65, "A", "B")



The screenshot shows a software interface with a top menu bar containing 'Relationships', 'Measure Column', 'Table', 'Parameter', 'Column', and 'Auto'. Below the menu bar, there are tabs for 'Relationships', 'Calculations', 'What If', 'Sort', 'Formatting', and 'Properties'. A formula bar displays the formula: `1 Rating = IF([Student Marks %] > 0.65, "A", "B")`. Below the formula bar, a table is displayed with three columns: 'Student Name', 'Student Marks %', and 'Rating'. The table contains five rows of student data and a 'Total' row. The 'Rating' column is calculated based on the formula, showing 'A' for marks above 0.65 and 'B' for marks below 0.65.

Student Name	Student Marks %	Rating
Student1	71%	A
Student2	65%	B
Student3	71%	A
Student4	64%	B
Student5	65%	B
<b>Total</b>	<b>67%</b>	<b>A</b>

EX. Rating w Variables -

VAR obtained Marks = SUMX (Marks, Marks[Mid term Marks] + Marks[Final Term Marks])

VAR TotalMarks = SUM(Marks[Total Marks])

VAR MarksPercentage = obtainedMarks / TotalMarks

RETURN

IF(Markspercentage > 0.65, "A", "B")

The screenshot shows the Power BI DAX editor interface. The top ribbon includes tabs for 'relationships', 'Calculations', 'What If', 'Sort', 'Formatting', 'Properties', and 'Security'. The 'Calculations' tab is active, showing the DAX formula for a measure named 'Rating w Variables'.

```

1 Rating w Variables =
2 VAR obtainedMarks = SUMX(Marks, Marks[Mid term Marks] + Marks[Final Term Marks])
3 VAR TotalMarks = SUM(Marks[Total Marks])
4 VAR MarksPercentage = obtainedMarks / TotalMarks
5 RETURN
6 IF(MarksPercentage > 0.65, "A", "B")

```

Below the formula, a table visualization displays the results of the measure. The table has two columns: 'Student Name' and 'Rating'. The data rows are as follows:

Student Name	Rating
Student1	65% B
Student2	71% A
Student3	64% B
Student4	65% B
Student5	67% A
<b>Total</b>	<b>67% A</b>

## DAX AGGREGATE FUNCTIONS

These functions in Data Analysis Expressions (DAX) enable data aggregation and summarization in Power BI and other Microsoft Power Platform tools.

**DAX aggregation functions along with their syntax and usage:**

### 1. ADDCOLUMNS

**Syntax:** ADDCOLUMNS(Table, NewColumnName1, Expression1, [NewColumnName2], [Expression2], ...)

**Usage:** The ADDCOLUMNS function creates a new table with additional calculated columns from the specified expressions.

### Parameters

Parameter	Description
table	Table or a DAX expression that returns a table.
name	The name given to the column, enclosed in double quotes.
expression	DAX expression that returns a scalar expression, evaluated for each row of table.

### Return Value

A table with all its original columns and the added ones.

### Remarks

--

### Example

```
=ADDCOLUMNS (  
    Products,"East_Sales", SUMX (RELATEDTABLE(East_Sales),  
                                IF([Product]=East_Sales[Product],  
                                East_Sales[Sales Amount],0)  
    )  
)
```

## 2. AVERAGE

**Syntax:** AVERAGE(Column)

**Usage:** The AVERAGE function calculates the arithmetic mean of the values in the specified Column.

### Return Value

Returns a decimal number that represents the arithmetic mean of the numbers in the column.

### Remarks

- If the column contains logical values or empty cells, those values are ignored and the rows are not counted.
- Cells with the value zero are included and the rows are counted for the divisor.
- 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.

### Example

=AVERAGE (Sales[Sales Amount])

## 3. AVERAGEA

**Syntax:** AVERAGEA(Column)

**Usage:** The AVERAGEA function calculates the arithmetic mean of the values in the specified Column, including non-numeric values as zeros.

### Return Value

Returns a decimal number.

### Remarks

The AVERAGEA function takes a column and averages the numbers in it and 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).

### Example

=AVERAGEA (East\_Sales[Sales Amount])

## 4. AVERAGEX

**Syntax:** AVERAGEX(Table, Expression)

**Usage:** The AVERAGEX function calculates the arithmetic mean of the Expression evaluated over the rows of the specified Table.

### Return Value

A decimal number.

### Remarks

The AVERAGEX function enables you to evaluate expressions for each row of a table, and then take the resulting set of values and calculate its arithmetic mean. Therefore, the function takes a table as its first argument and an expression as the second argument.

In all other respects, AVERAGEX follows the same rules as AVERAGE. You cannot include non numeric or null cells.

### Example

=AVERAGEX (East\_Sales, East\_Sales[Unit Price] \* East\_Sales[No. of Units])

Sales AVERAGE = AVERAGE(Sales[Price]) \* SUM(Sales[Quantity])

Sales AVERAGEX = AVERAGEX(Sales, Sales[Price]) \* SUM(Sales[Quantity])

## 5. COUNT

**Syntax:** COUNT(Column)

**Usage:** The COUNT function counts the number of non-blank values in the specified Column.

### Return Value

Returns a whole number.

### Remarks

You can use columns containing any type of data, but only numbers are counted. The COUNT function counts the 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

=COUNT (ProductInventory[UnitsBalance])

## 6. COUNTA

**Syntax:** COUNTA(Column)

**Usage:** The COUNTA function counts the number of non-blank values in the specified Column, including text, numbers, and logical values.

### Return Value

Returns a whole number.

### Remarks

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**

=COUNTA (ProductInventory[UsageDate])

## **7. COUNTAX**

Syntax: COUNTAX(Table, Expression)

Usage: The COUNTAX function counts the number of non-blank values of the Expression evaluated over the rows of the specified Table.

### **Return Value**

A whole number.

### **Remarks**

The COUNTAX function counts the 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. 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.

### **Example**

Medal Count Summer Sports:=COUNTAX (  
    FILTER (Results, Results[Season]="Summer"),  
    Results[Medal])

## **8. COUNTBLANK**

Syntax: COUNTBLANK(Column)

Usage: The function counts the number of blank values in the specified Column.

### **Return Value**

A whole number. If there are no blank rows, blank is returned.

### **Example**

=COUNTBLANK(Results[Medal])

## **9. COUNTROWS**

**Syntax:** COUNTROWS(Table)

**Usage:** The COUNTROWS function counts the number of rows in the specified Table.

**Return Value**

Returns a whole number.

**Remarks** - This function can be used to count the number of rows in a base table, but more often is used to count the number of rows that result from filtering a table, or applying a context to a table.

**Example**

=COUNTROWS (CALENDAR (DATE (2016,8,1), DATE (2016,10,31))) returns 92.

=COUNTROWS (Results) returns 34094.

=COUNTROWS (Events) returns 995.

**Remarks**

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.

**Example**

=COUNTBLANK (SalesTarget[SalesTarget])

## 10. COUNTX

**Syntax:** COUNTX(Table, Expression)

**Usage:** The COUNTX function counts the number of non-blank values of the Expression evaluated over the rows of the specified Table.

**Return Value**

Returns a whole number.

**Remarks**

The COUNTX function counts only numeric values or dates. Parameters 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.

**Example**

=COUNTX (RELATEDTABLE (East\_Sales), IF ([Product]=East\_Sales[Product],1,0))

## 11. CROSSJOIN

**Syntax:** CROSSJOIN(Table1, Table2)

**Usage:** The CROSSJOIN function creates a new table by combining all the rows from Table1 with all the rows from Table2.

### Return Value

Returns a table that contains the Cartesian product of all rows from all tables in the parameters. The columns in the new table are all the columns in all the parameter tables.

### Remarks

- Column names from table parameters must all be different in all tables or an error is returned.
- The total number of rows in the result table is the product of the number of rows from all tables in the parameters.
- The total number of columns in the result table is the sum of the number of columns from all tables in the parameters.

For example, if table1 has r1 rows and c1 columns, table2 has r2 rows and c2 columns, and table3 has r3 rows and c3 columns, then the resulting table will have -

**$r1 \times r2 \times r3$  rows and  $c1 + c2 + c3$  columns**

### Example

=CROSSJOIN (Salesperson,Products)

## 12. DISTINCTCOUNT

**Syntax:** DISTINCTCOUNT(Column)

**Usage:** The DISTINCTCOUNT function counts the number of distinct (unique) values in the specified Column.

## 13. GENERATE

**Syntax:** GENERATE(Table, Table2)

**Usage:** The GENERATE function creates a new table by iterating over the rows of Table and for each row, returns the rows of Table2.

### Return Value

A table that can be passed as a parameter to a DAX function.

### Remarks

- If the evaluation of table2 for the current row in table1 returns an empty table, then the result table will not contain the current row from table1. This is different than GENERATEALL () where the current row from table1 will be included in the results, and columns corresponding to table2 will have null values for that row.
- All column names from table1 and table2 must be different or an error is returned.

#### **Example**

```
=GENERATE (
SUMMARIZE(Salesperson,Salesperson[Salesperson]),
SUMMARIZE(SalesTarget,SalesTarget[SalesTarget],"MaxTarget",MAX(SalesTar
get[SalesTarget]))))
```

### **14. GENERATEALL**

**Syntax:** GENERATEALL(Table)

**Usage:** The GENERATEALL function creates a new table by iterating over all the rows of Table, including the blank rows.

#### **Return Value**

Returns a table with the Cartesian product between each row in table1 and the table that results from evaluating table2 in the context of the current row from table1.

#### **Remarks**

- If the evaluation of table2 for the current row in table1 returns an empty table, then the current row from table1 will be included in the results, and columns corresponding to table2 will have null values for that row. This is different than GENERATE () where the current row from table1 will not be included in the results in such a case.
- All column names from table1 and table2 must be different or an error is returned.

#### **Example**

```
=GENERATEALL (
SUMMARIZE(Salesperson,Salesperson[Salesperson]),
SUMMARIZE(SalesTarget,SalesTarget[SalesTarget],"MaxTarget",MAX(SalesTar
get[SalesTarget]))))
```

### **15. MAX**

**Syntax:** MAX(Column)

**Usage:** The MAX function returns the maximum value in the specified Column.

**Return Value**

A decimal number.

**Remarks**

The following types of values in the column are considered:

- Numbers
- Dates

Empty cells, logical values, and text are ignored.

**Example**

=MAX (Sales[Sales Amount])

**16. MAXA**

**Syntax:** MAXA(Column)

**Usage:** The MAXA function returns the maximum value in the specified Column, including non-numeric values.

**Return Value**

Returns a decimal number.

**Remarks**

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

- Numbers
- Dates
- Logical values, such as TRUE and FALSE. Rows that evaluate to TRUE count as 1 and 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).

**Example**

=MAXA (ProductInventory[UsageDate])

**17. MAXX**

**Syntax:** MAXX(Table, Expression)

**Usage:** The MAXX function returns the maximum value of the Expression evaluated over the rows of the specified Table.

**Return Value**

Returns a decimal number.

#### **Remarks**

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.

#### **Example**

=MAXX (East\_Sales,East\_Sales[No. of Units]\*East\_Sales[Unit Price])

Maximum Value = MAX(SUM(Orders[Profit]), 290000)

### **18. MIN**

Syntax: MIN(Column)

Usage: The MIN function returns the minimum value in the specified Column.

Returns the smallest value in column.

Column can have Numeric, String, Date type.

EX. Minimum Profit = MIN(Orders [Profit])

### **19. MINA**

Syntax: MINA(Column)

Usage: The MINA function returns the minimum value in the specified Column, including non-numeric values.

### **20. MINX**

Syntax: MINX(Table, Expression)

Usage: The MINX function returns the minimum value of the Expression evaluated over the rows of the specified Table.

### **21. PRODUCT**

Syntax: PRODUCT(Column)

Usage: The function calculates the product of all the values in the specified Column.

Returns a decimal number, the product of all the numbers in the column.

Column must contain numbers.

EX. 2021 Pop = [1951 Pop] \* PRODUCT('Pop change'[Growth Factor])

2023 Pop = [2023 Pop] \* PRODUCT(India[Division])

## 22. PRODUCTX

Syntax: PRODUCTX(Table, Expression)

Usage: The PRODUCTX function calculates the product of the Expression evaluated over the rows of the specified Table.

## 23. ROW

Syntax: ROW(Column1, [Column2], ...)

Usage: The ROW function returns a single-row table containing the specified columns and their values.

## 24. SELECTCOLUMNS

Syntax: SELECTCOLUMNS(Table, ColumnName1, [ColumnName2], ...)

Usage: The SELECTCOLUMNS function creates a new table with the specified columns from the original Table.

## 25. SUM

Syntax: SUM(Column)

Usage: - The SUM function calculates the sum of the values in the specified Column.

Returns a decimal number, the sum of all the numbers in the column.

Column must contain numbers.

EX. Total Profit = SUM(Orders[profit])

## 26. SUMMARIZE

Syntax: SUMMARIZE(Table, Group\_Column1, [Group\_Column2], ..., [Expression1], [Expression2], ...)

Usage: The SUMMARIZE function groups the rows of the specified Table based on the Group\_Columns and calculates the expressions specified for each group.

## 27. SUMX

Syntax: SUMX(Table, Expression)

Usage: The SUMX function calculates the sum of the Expression evaluated over the rows of the specified Table.

EX.  $\text{Sales SUM} = \text{SUM}(\text{Sales}[\text{Price}]) * \text{SUM}(\text{Sales}[\text{quantity}])$

$\text{Sales SUMX} = \text{SUMX}(\text{Sales}, \text{Sales}[\text{Price}] * \text{Sales}[\text{Quantity}])$

## 28. TOPN

Syntax: TOPN(N, Table, Expression, [Order])

Usage: The TOPN function returns the top N rows from the specified Table based on the Expression, sorted by the specified Order.

These DAX aggregation functions allow you to perform various types of calculations and aggregations on your data in Power BI and other Microsoft Power Platform tools. They are essential for summarizing and analyzing data at different levels of granularity.

## DAX FILTER FUNCTIONS

- Provides additional control when modifying filter context
- ALL(), ALLEXCEPT(), KEEPFILTERS(), REMOVEFILTERS() and more.
- ALL function variants returns a table object so can also consider as TEF

EX.  $\text{WEST SALES} = \text{CALCULATE}([\text{Total Sales}], \text{FILTER}(\text{Orders}, \text{Orders}[\text{Region}] = \text{"west"})$

$\text{WEST SALES} = \text{CALCULATE}([\text{Total Sales}], \text{KEEPFILTERS}(\text{Orders}, \text{Orders}[\text{Region}] = \text{"west"}))$

### Filter Functions:

$\text{CALCULATE}(\langle \text{expression} \rangle, [\langle \text{filter1} \rangle], [\langle \text{filter2} \rangle], [\langle \text{filter3} \rangle] \dots)$



- Evaluates an Expression in a Modified Filter Context.
- Filters can be Boolean expression, Table expression, or Filter modifier functions.
- By default, all the filters separated by comma are connected with && operators.

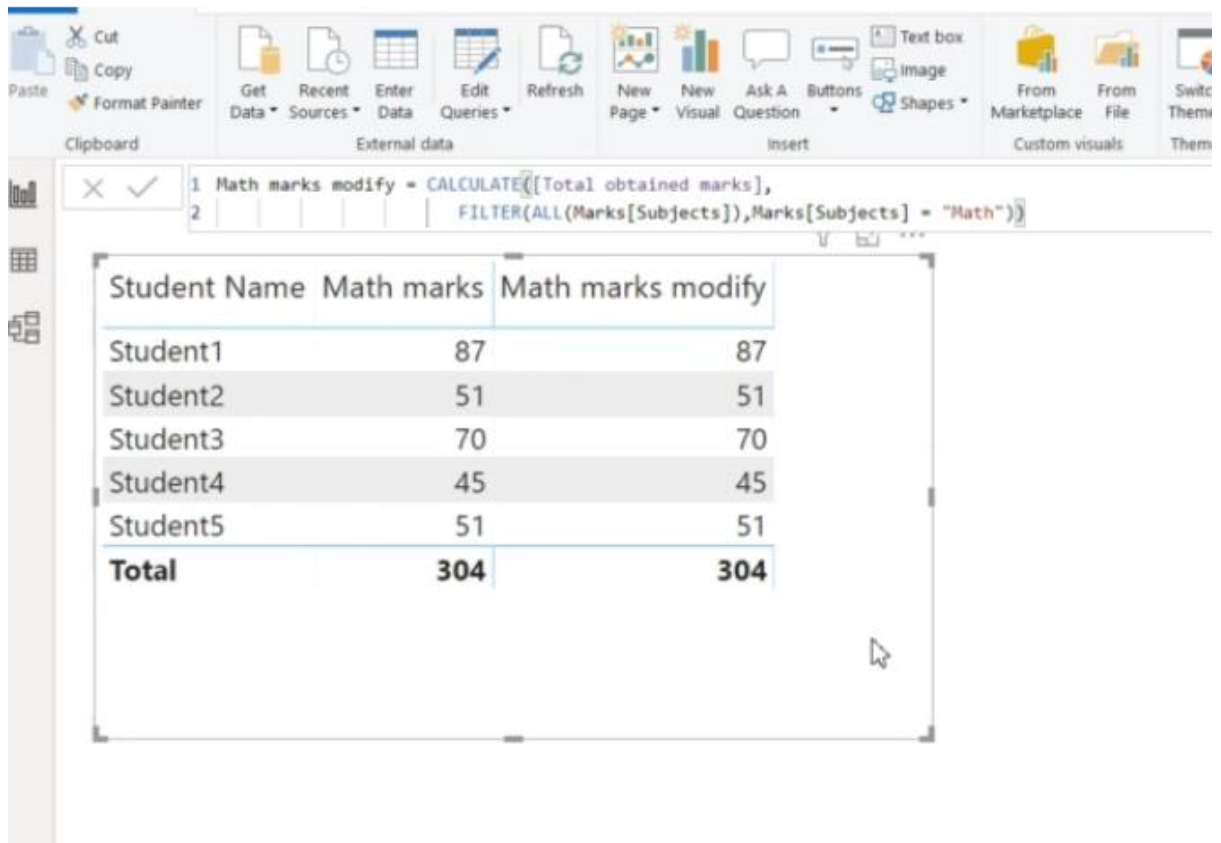
#### Boolean expression filter:

- Expression that evaluates to TRUE or FALSE.
- Can reference columns from single table
- Can not reference Measures.

The screenshot shows the Power BI Desktop interface. The formula bar at the top contains the DAX measure: `1 Math marks = CALCULATE([Total obtained marks], Marks[Subjects] = "Math")`. Below the formula bar, a table is displayed with the following data:

Student Name	Math marks
Student1	87
Student2	51
Student3	70
Student4	45
Student5	51
<b>Total</b>	<b>304</b>

EX. Math marks modify = `CALCULATE([Total obtained marks],  
 FILTER(ALL(Marks[Subjects]), Marks[Subjects] = "MATH"))`



### Table expression filter:

- Uses a table object as a filter
- A table name, or a function that returns a table object
- Mostly FILTER() function is used as Table expression filter

EX. WEST SALES = CALCULATE([Total Sales], Orders[Region] = "west")

WEST SALES = CALCULATE([Total Sales], Orders[Region] = "west" || Orders[Region] = "south")

WEST TECH SALES = CALCULATE([Total Sales],

Orders[Region] = "west" ||

Orders[Category] = "Technology")

EX. Red color Products = FILTER(Products, Products[Color] = "RED")

hips	Measure Column	Table	Parameter	Column	\$ - % + .00 Auto	Default Summarization: Don't summarize	Roles	Roles
hips	Calculations		What If	Sort	Formatting	Properties	Security	

✕ ✓
1 Red color products = `FILTER(Products,Products[Color] = "Red")`

ProductKey	ProductSubcategoryKey	ProductName	StandardCost	Color	SafetyStockLevel	ListPrice
212	31	Sport-100 Helmet, Red	12.0278	Red	4	33.6442
213	31	Sport-100 Helmet, Red	13.8782	Red	4	33.6442
214	31	Sport-100 Helmet, Red	13.0863	Red	4	34.99
314	2	Road-150 Red, 56	2171.2942	Red	100	3578.27
315	2	Road-450 Red, 58	884.7083	Red	100	1457.99
310	2	Road-150 Red, 62	2171.2942	Red	100	3578.27
311	2	Road-150 Red, 44	2171.2942	Red	100	3578.27
312	2	Road-150 Red, 48	2171.2942	Red	100	3578.27
313	2	Road-150 Red, 52	2171.2942	Red	100	3578.27
316	2	Road-450 Red, 60	884.7083	Red	100	1457.99
317	2	Road-450 Red, 44	884.7083	Red	100	1457.99
318	2	Road-450 Red, 48	884.7083	Red	100	1457.99
319	2	Road-450 Red, 52	884.7083	Red	100	1457.99
320	2	Road-650 Red, 58	413.1463	Red	100	699.0982

EX. Filter Sales = SUMX(  
 FILTER(Sales, Sales[SalesAmount] > 3000),  
 Sales[SalesAmount] + Sales[Freight]  
 )

Painter	Get Data	Recent Sources	Enter Data	Edit Queries	Refresh	New Page	New Visual	Ask A Question	Buttons	Shapes	From Marketplace	From File	Switch Theme	Manage Relationships
	External data					Insert					Custom visuals		Themes	Relationships

1 Filter Sales = SUMX(  
2     FILTER(Sales, Sales[SalesAmount] > 3000),  
3     Sales[SalesAmount] + Sales[Freight]  
4 )

Australia	106,782.04	21,823.62
Canada	100,309.99	25,257.36
France	94,191.86	18,155.90
Germany	143,341.86	53,868.23
NA	120,696.66	36,311.79
United Kingdom	123,241.40	39,380.06
United States	517,443.91	159,190.50
<b>Total</b>	<b>15,183,960.47</b>	<b>3,561,011.16</b>

Gender  
☒ F  
☐ M

## 1. ADDMISSINGITEMS

**Syntax:** ADDMISSINGITEMS(Column, Value1, [Value2], ...)

**Usage:** The ADDMISSINGITEMS function returns a new table that includes the missing items from the specified column. It is commonly used when creating relationships between tables with missing values, ensuring that all possible combinations are considered.

## 2. ALL

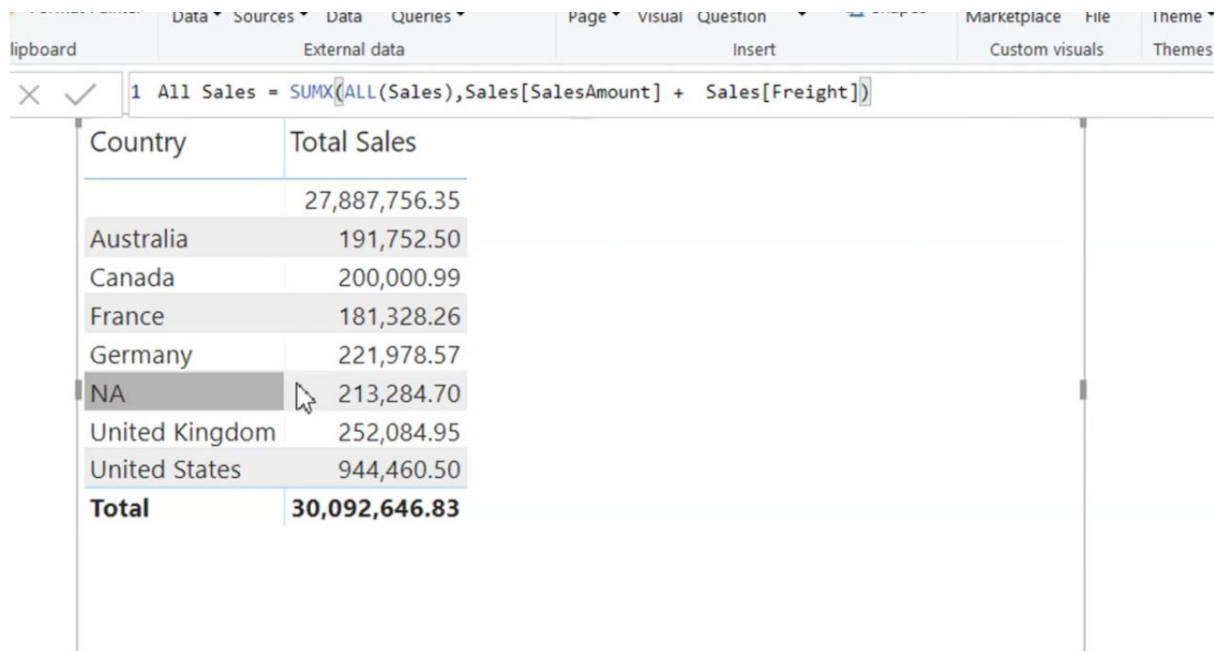
**Syntax:** ALL([TableNameOrColumn], [ColumnName], ...)

**Usage:** The ALL function removes filters from the specified table or columns. It returns all the rows from the specified table, disregarding any filter context from the visual or report level.

**ALL <table name>**

- Clears all applied filters
- Helps in the calculation without any filter
- Does not Return anything, just modifies the filter context
- Removes all filter from the specified table
- Used in CALCULATE function as Table expression filter or Filter modifier function

**EX. All Sales = SUMX(ALL(Sales), Sales[SalesAmount] + Sales[Freight])**



Country	Total Sales
	27,887,756.35
Australia	191,752.50
Canada	200,000.99
France	181,328.26
Germany	221,978.57
NA	213,284.70
United Kingdom	252,084.95
United States	944,460.50
<b>Total</b>	<b>30,092,646.83</b>

## 3. ALLEXCEPT

**Syntax:** ALLEXCEPT([TableName], [ColumnName1], ...)

ALLEXCEPT(<table\_name>, <column\_name1>,<column\_name2>...)

**Usage:** The ALLEXCEPT function removes filters from all columns in the specified table except for the columns listed. This is useful when you want to maintain the filter context of specific columns while removing filters from others.

- Removes all FILTERS from the specified table, except filters on specified columns
- Used in CALCULATE function as Table expression filter or filter modifier function.  
EX. Sales ALLEx = CALCULATE(SUM(Orders[Sales]), ALLEXCEPT(Orders, Orders[Region]))

#### 4. ALLNOBLANKROW

**Syntax:** ALLNOBLANKROW([TableName])

**Usage:** The ALLNOBLANKROW function removes the blank row from the specified table. It is helpful when you want to exclude the automatically added blank row that appears when creating relationships in Power BI.

#### 5. ALLSELECTED

**Syntax:** ALLSELECTED([ColumnName], ...)

**Usage:** The ALLSELECTED function returns all the values from the specified columns in the current filter context. It is useful for capturing the selected values when using visual or slicer filters.

#### 6. SELECTEDVALUE

**Syntax:** SELECTEDVALUE<column\_name>, [<alternate results >])

**Usage:** The ALLSELECTED function returns all the values from the specified columns in the current filter context. It is useful for capturing the selected values when using visual or slicer filters.

- Returns the One DISTINCT value that column has been filtered down to.
- In case of zero or more than one distinct value, it returns the alternate result.
- Default value for the alternate result is BLANK.
- Mostly used with IF or SWITCH functions.

SELECTEDVALUE<column\_name>, <alternate results >) =>

IF(HASONEVALUE(<column name >), VALUES<column name>), <alternate result>)

EX. SELECTED STATE = SELECTEDVALUE(Orders[State], "Multiple states")

#### 7. CALCULATE

**Syntax:** CALCULATE(Expression, Filter1, Filter2, ...)

**Usage:** The CALCULATE function evaluates the expression within the specified filter context. It allows you to dynamically change or add filters to modify the calculations in your DAX formulas.

## 8. CALCULATETABLE

**Syntax:** CALCULATETABLE(Table, Filter1, Filter2, ...)

**Usage:** The CALCULATETABLE function filters the specified table based on the provided filters. It returns a new table that contains only the rows that meet the given conditions.

## 9. CROSSFILTER

**Syntax:** CROSSFILTER([TableName], [RelatedTableName], CrossFilterType)

**Usage:** The CROSSFILTER function defines a filtering behavior between two related tables. CrossFilterType can be either "Both", "Single", or "None," indicating how filters are propagated between the tables.

## 10. DISTINCT

**Syntax:** DISTINCT([ColumnName])

**Usage:** The DISTINCT function returns a table with unique values from the specified column. It is often used in combination with other functions to remove duplicates.

## 11. EARLIER

**Syntax:** EARLIER(Expression)

**Usage:** The EARLIER function allows you to reference a value from a previous row within an iteration. It is primarily used in calculations involving iterative functions like SUMX and AVERAGEX.

## 11. EARLIER Function

**Syntax:** EARLIER(ColumnOrMeasure)

**Usage:** The EARLIER function is used in iterative calculations to refer to the value of a column or measure from a previous row within the current iteration context.

## 12. EARLIEST

**Syntax:** EARLIEST([TableName])

**Usage:** The EARLIEST function returns the earliest date from the specified column in the filter context. It is commonly used in time intelligence calculations to get the start date of a selected period.

### 13. FILTER

**Syntax:** FILTER(Table, Condition)

**Usage:** The FILTER function filters the specified table based on the given condition and returns a new table with the filtered rows. It is commonly used to apply dynamic filters to tables.

### 14. FILTERS

**Syntax:** FILTERS([ColumnName], ...)

**Usage:** The FILTERS function returns a table containing distinct values from the specified columns in the current filter context. It is helpful for understanding the filter context during query debugging.

### 15. HASONEFILTER

**Syntax:** HASONEFILTER([ColumnName])

**Usage:** The HASONEFILTER function checks if the specified column has a single filter applied to it. It returns TRUE if there is only one active filter on the column.

### 16. HASONEVALUE

**Syntax:** HASONEVALUE([ColumnName])

**Usage:** The HASONEVALUE function checks if the specified column has a single distinct value in the current filter context. It returns TRUE if there is only one distinct value for the column.

### 17. ISCROSSFILTERED

**Syntax:** ISCROSSFILTERED([TableName])

**Usage:** The ISCROSSFILTERED function checks if the specified table is actively cross-filtered by a related table. It returns TRUE if cross-filtering is happening.

### 18. ISFILTERED

**Syntax:** ISFILTERED([ColumnName])

**Usage:** The ISFILTERED function checks if the specified column is actively filtered in the current context. It returns TRUE if the column has a filter applied to it.

## 19. KEEPFILTERS

**Syntax:** KEEPFILTERS(Expression)

KEEPFILTERS(<boolean\_expression\_filters>) Same as FILTER, but better performance

**Usage:** The KEEPFILTERS function preserves the existing filters in the filter context when evaluating the expression. It is useful when you want to apply additional filters while keeping the current ones.

- Adds a new Filter context, keeping all other filters active,
- Used CALCULATE Function as Filter modifier function

**EX. Female Marks = CALCULATE([Total Marks], KEEPFILTERS(Students[Gender] = "F"))**

Subjects	Total marks	Female marks	Marks %
Bio	180	180	100.00%
Chemistry	217	217	100.00%
Computer	196	196	100.00%
Math	172	172	100.00%
Physics	240	240	100.00%
<b>Total</b>	<b>1005</b>	<b>1005</b>	<b>100.00%</b>

**EX. East sales with keepFilter = CALCULATE(SUM(Orders[Sales]), KEEPFILTERS(Orders[Region] = "East"))**

## 20. REMOVEFILTERS

**Syntax:** REMOVEFILTERS(Expression)



**Usage:**

- Remove all filters from the specified table
  - Used in CALCULATE function as Filter modifier function
- EX. Sales RF T = CALCULATE(SUM(Orders[Sales]), REMOVEFILTERS(Orders))

**REMOVEFILTERS()**

- Clears all applied filters
- Helps in the calculation without any filters,
- Does not return anything, just modifies the filter context

ALL functions and REMOVEFILTERS functions are same, but REMOVEFILTERS perform better.

**21. RELATED**

**Syntax:** RELATED(TableName[ColumnName])

**Usage:** The RELATED function returns a single value from the related table based on a relationship established between tables.

**22. RELATEDTABLE**

**Syntax:** RELATEDTABLE(TableName)

**Usage:** The RELATEDTABLE function returns a table with all related rows from the related table based on an established relationship.

**23. USERELATIONSHIP**

**Syntax:** USERELATIONSHIP([TableName1][ColumnName1], [TableName2][ColumnName2])

**Usage:** The USERELATIONSHIP function overrides the automatic relationship detection between two columns and allows you to define a specific relationship between tables.

**24. VALUES**

**Syntax:** VALUES([ColumnName])

**Usage:** The VALUES function returns a table with unique values from the specified column. It is commonly used to create new calculated columns or measures based on unique values.

These DAX filter functions play a crucial role in data analysis and manipulation within Power BI and other Microsoft Power Platform tools.

## **DAX TIME INTELLIGENCE FUNCTIONS**

DAX time intelligence functions along with their syntax and usage:

**1. CLOSINGBALANCEMONTH**

**Syntax:** CLOSINGBALANCEMONTH(Expression, DateColumn)

**Usage:** The CLOSINGBALANCEMONTH function calculates the closing balance for a measure at the end of each month in the specified DateColumn.

**2. CLOSINGBALANCEQUARTER**

**Syntax:** CLOSINGBALANCEQUARTER(Expression, DateColumn)

**Usage:** The CLOSINGBALANCEQUARTER function calculates the closing balance for a measure at the end of each quarter in the specified DateColumn.

**3. CLOSINGBALANCEYEAR**

**Syntax:** CLOSINGBALANCEYEAR(Expression, DateColumn)

**Usage:** The CLOSINGBALANCEYEAR function calculates the closing balance for a measure at the end of each year in the specified DateColumn.

**4. DATEADD**

**Syntax:** DATEADD(DateColumn, NumberOfIntervals, Interval)

**Usage:** The DATEADD function returns a new date shifted by the specified number of intervals (days, months, quarters, or years) from the original DateColumn.

**5. DATESBETWEEN**

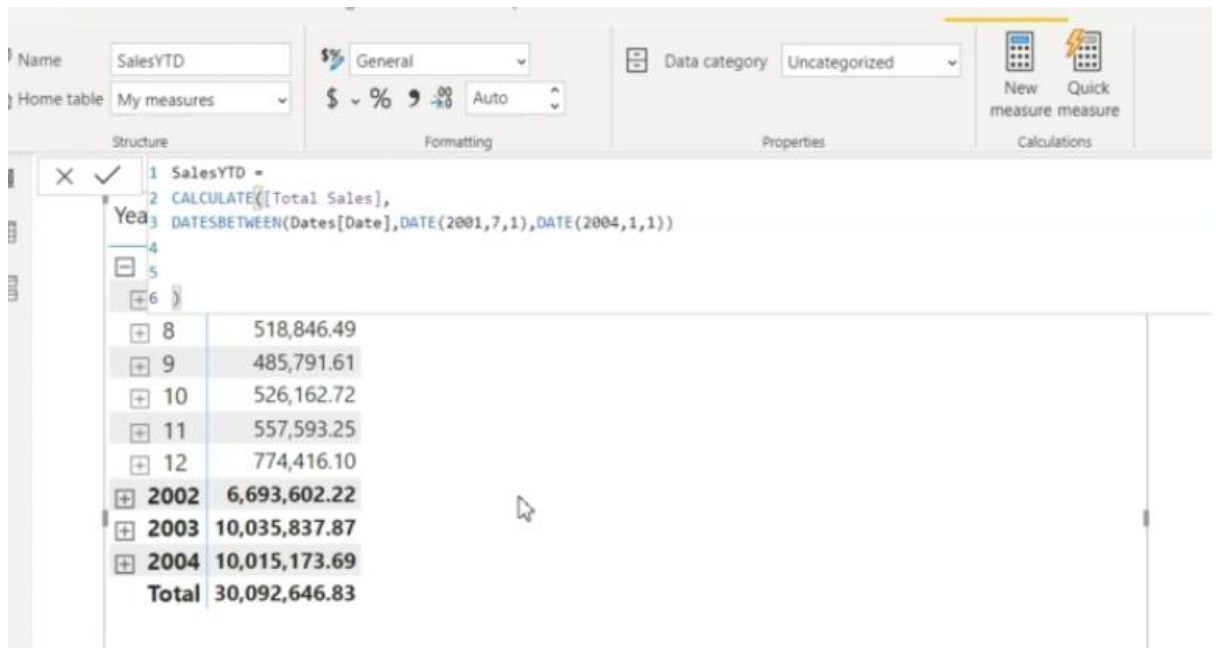
**Syntax:** DATESBETWEEN(DateColumn, StartDate, EndDate)

**Usage:** The DATESBETWEEN function filters a table to include only the dates between the specified StartDate and EndDate in the specified DateColumn.

**EX. SalesYTD =**

**CALCULATE([Total Sales],**

**DATESBETWEEN(Dates[Date], DATE(2001,7,1), DATE(2004,1,1))**



## 6. DATESINPERIOD

**Syntax:** DATESINPERIOD(DateColumn, StartDate, NumberOfIntervals, Interval)

**Usage:** The DATESINPERIOD function returns a table with dates from the specified DateColumn that fall within the period defined by the StartDate and NumberOfIntervals (e.g., 1 for one month, 2 for two months) and the Interval (e.g., "month" or "quarter").

## 7. DATESMTD

**Syntax:** DATESMTD(DateColumn)

**Usage:** The DATESMTD function returns a table with dates from the specified DateColumn that are in the same month-to-date period as the current filter context.

## 8. DATESQTD

**Syntax:** DATESQTD(DateColumn)

**Usage:** The DATESQTD function returns a table with dates from the specified DateColumn that are in the same quarter-to-date period as the current filter context.

## 9. DATASYTD

**Syntax:** DATASYTD(DateColumn)

**Usage:** The DATESYTD function returns a table with dates from the specified DateColumn that are in the same year-to-date period as the current filter context.

**EX. YTD = CALCULATE([Total Sales], DATESYTD(Dates[Date]))**

The screenshot shows the Microsoft Excel interface with the following data in the PivotTable:

Year	Total Sales	SalesYTD	YTD
<b>2001</b>	<b>3,348,033.05</b>	<b>3,348,033.05</b>	<b>3,348,033.05</b>
7	485,222.87	485,222.87	485,222.87
8	518,846.49	1,004,069.37	1,004,069.37
9	485,791.61	1,489,860.98	1,489,860.98
10	526,162.72	2,016,023.70	2,016,023.70
11	557,593.25	2,573,616.95	2,573,616.95
12	774,416.10	3,348,033.05	3,348,033.05
<b>2002</b>	<b>6,693,602.22</b>	<b>10,041,635.27</b>	<b>6,693,602.22</b>
1	611,665.23	3,959,698.28	611,665.23
2	564,587.12	4,524,285.40	1,176,252.35
3	660,238.59	5,184,523.99	1,836,490.94
4	680,284.60	5,864,808.59	2,516,775.55
5	690,395.11	6,555,203.71	3,207,170.66
6	693,682.75	7,248,886.46	3,900,853.41
7	512,874.29	7,761,760.75	4,413,727.70
8	559,651.52	8,321,412.27	4,973,379.22
9	359,228.67	8,680,640.94	5,332,607.89
<b>Total</b>	<b>30,092,646.83</b>	<b>30,092,646.83</b>	<b>10,015,173.69</b>

**EX. YTD = CALCULATE([Total Sales], DATESYTD(Dates[Date], "8/30"))**

File Home Insert Modeling View Help Format Data / Drill Table tools Measure tools

Name: YTD    Format: General    Data category: Uncategorized    New measure    Quick measure

Home table: My measures    Structure    Formatting    Properties    Calculations

1 YTD = CALCULATE([Total Sales], DATESYTD(Dates[Date], "8/30"))

Year	Total Sales	YTD
2001	3,348,033.05	2,365,344.96
7	485,222.87	485,222.87
1	14,839.27	14,839.27
2	14,279.81	29,119.08
3	15,387.48	44,506.56
4	7,335.45	51,842.02
5	15,387.48	67,229.50
6	14,670.91	81,900.41
7	8,052.03	89,952.44
8	8,052.03	98,004.47
9	21,432.52	119,436.99
10	10,820.44	130,257.43
11	14,670.91	144,928.34
12	14,488.17	159,416.51
13	7,335.45	166,751.96
14	25,674.09	192,426.05
15	11,511.39	203,937.45
<b>Total</b>	<b>30,092,646.83</b>	

## 10. ENDOFMONTH

**Syntax:** ENDOFMONTH(DateColumn)

**Usage:** The ENDOFMONTH function returns the last day of the month for each date in the specified DateColumn.

## 11. ENDOFQUARTER

**Syntax:** ENDOFQUARTER(DateColumn)

**Usage:** The ENDOFQUARTER function returns the last day of the quarter for each date in the specified DateColumn.

## 12. ENDOFYEAR

**Syntax:** ENDOFYEAR(DateColumn)

**Usage:** The ENDOFYEAR function returns the last day of the year for each date in the specified DateColumn.

### 13. FIRSDATE

**Syntax:** FIRSDATE(DateColumn)

**Usage:** The FIRSDATE function returns the earliest date from the specified DateColumn.

### 14. FIRSTNONBLANK

**Syntax:** FIRSTNONBLANK(Column, Expression)

**Usage:** The FIRSTNONBLANK function returns the first non-blank value from the specified Column for which the Expression evaluates to TRUE. It is commonly used to find the first non-blank value in a column based on certain conditions.

### 15. LASTDATE

**Syntax:** LASTDATE(DateColumn)

**Usage:** The LASTDATE function returns the latest date from the specified DateColumn.

### 16. LASTNONBLANK

**Syntax:** LASTNONBLANK(Column, Expression)

**Usage:** The LASTNONBLANK function returns the last non-blank value from the specified Column for which the Expression evaluates to TRUE. It is commonly used to find the last non-blank value in a column based on certain conditions.

### 17. NEXTDAY

**Syntax:** NEXTDAY(DateColumn)

**Usage:** The NEXTDAY function returns the date that follows each date in the specified DateColumn by one day.

### 18. NEXTMONTH

**Syntax:** NEXTMONTH(DateColumn)

**Usage:** The NEXTMONTH function returns the date that represents the first day of the month that follows each date in the specified DateColumn.

### 19. NEXTQUARTER

**Syntax:** NEXTQUARTER(DateColumn)

**Usage:** The NEXTQUARTER function returns the date that represents the first day of the quarter that follows each date in the specified DateColumn.

## 20. NEXTYEAR

**Syntax:** NEXTYEAR(DateColumn)

**Usage:** The NEXTYEAR function returns the date that represents the first day of the year that follows each date in the specified DateColumn.

## 21. OPENINGBALANCEMONTH

**Syntax:** OPENINGBALANCEMONTH(Expression, DateColumn)

**Usage:** The OPENINGBALANCEMONTH function calculates the opening balance for a measure at the start of each month in the specified DateColumn.

## 22. OPENINGBALANCEQUARTER

**Syntax:** OPENINGBALANCEQUARTER(Expression, DateColumn)

**Usage:** The OPENINGBALANCEQUARTER function calculates the opening balance for a measure at the start of each quarter in the specified DateColumn.

## 23. OPENINGBALANCEYEAR

**Syntax:** OPENINGBALANCEYEAR(Expression, DateColumn)

**Usage:** The OPENINGBALANCEYEAR function calculates the opening balance for a measure at the start of each year in the specified DateColumn.

## 24. PARALLELPERIOD

**Syntax:** PARALLELPERIOD(DateColumn, NumberOfIntervals, Interval)

**Usage:** The PARALLELPERIOD function returns a date from the same period as the current date but shifted by the specified number of intervals (days, months, quarters, or years) in the past or future.

## 25. PREVIOUSDAY

**Syntax:** PREVIOUSDAY(DateColumn)

**Usage:** The PREVIOUSDAY function returns the date that precedes each date in the specified DateColumn by one day.

## 26. PREVIOUSMONTH

**Syntax:** PREVIOUSMONTH(DateColumn)

**Usage:** The PREVIOUSMONTH function returns the date that represents the first day of the month that precedes each date in the specified DateColumn.

## 27. PREVIOUSQUARTER

**Syntax:** PREVIOUSQUARTER(DateColumn)

**Usage:** The PREVIOUSQUARTER function returns the date that represents the first day of the quarter that precedes each date in the specified DateColumn.

## 28. PREVIOUSYEAR

**Syntax:** PREVIOUSYEAR(DateColumn)

**Usage:** The PREVIOUSYEAR function returns the date that represents the first day of the year that precedes each date in the specified DateColumn.

## 29. SAMEPERIODLASTYEAR

**Syntax:** SAMEPERIODLASTYEAR(DateColumn)

**Usage:** The SAMEPERIODLASTYEAR function returns the date from the same period as the current date but in the previous year.

## 30. STARTOFMONTH

**Syntax:** STARTOFMONTH(DateColumn)

**Usage:** The STARTOFMONTH function returns the first day of the month for each date in the specified DateColumn.

## 31. STARTOFQUARTER

**Syntax:** STARTOFQUARTER(DateColumn)



**Usage:** The STARTOFQUARTER function returns the first day of the quarter for each date in the specified DateColumn.

### 32. STARTOFYEAR

**Syntax:** STARTOFYEAR(DateColumn)

**Usage:** The STARTOFYEAR function returns the first day of the year for each date in the specified DateColumn.

### 33. TOTALMTD

**Syntax:** TOTALMTD(Expression, DateColumn)

**Usage:** The TOTALMTD function calculates the year-to-date total for a measure up to the maximum date in the specified DateColumn.

### 34. TOTALQTD

**Syntax:** TOTALQTD(Expression, DateColumn)

**Usage:** The TOTALQTD function calculates the quarter-to-date total for a measure up to the maximum date in the specified DateColumn.

### 35. TOTALYTD

**Syntax:** TOTALYTD(Expression, DateColumn)

**Usage:** The TOTALYTD function calculates the year-to-date total for a measure up to the maximum date in the specified DateColumn.

## DAX DATE AND TIME FUNCTIONS

## 1. CALENDAR

**Syntax:** CALENDAR(StartDate, EndDate)

**Usage:** The CALENDAR function generates a table with a single date column containing all dates from the StartDate to the EndDate.

## 2. CALENDARAUTO

**Syntax:** CALENDARAUTO()

**Usage:** The CALENDARAUTO function generates a table with a single date column containing all unique dates from the entire data model.

## 3. DATE

**Syntax:** DATE(Year, Month, Day)

**Usage:** The DATE function returns a date value based on the specified Year, Month, and Day.

## 4. DATEDIFF

**Syntax:** DATEDIFF(StartDate, EndDate, Interval)

**Usage:** The DATEDIFF function calculates the difference between two dates in the specified Interval (e.g., "day", "month", "quarter", "year").

## 5. DATEVALUE

**Syntax:** DATEVALUE(DateString)

**Usage:** The DATEVALUE function converts a date represented as a text string into a date value.

## 6. DAY

**Syntax:** DAY(Date)

**Usage:** The DAY function returns the day of the month for a given Date.

## 7. EDATE

**Syntax:** EDATE(StartDate, NumberOfMonths)

**Usage:** The EDATE function returns a date that is a specified number of months before or after the StartDate.

## 8. EOMONTH

**Syntax:** EOMONTH(StartDate, NumberOfMonths)

**Usage:** The EOMONTH function returns the last day of the month that is a specified number of months before or after the StartDate.

## 9. HOUR

**Syntax:** HOUR(Time)

**Usage:** The HOUR function returns the hour component of a given Time value.

## 10. MINUTE

**Syntax:** MINUTE(Time)

**Usage:** The MINUTE function returns the minute component of a given Time value.

## 11. MONTH

**Syntax:** MONTH(Date)

**Usage:** The MONTH function returns the month number for a given Date.

## 12. NOW

**Syntax:** NOW()

**Usage:** The NOW function returns the current date and time as a datetime value.

## 13. SECOND

**Syntax:** SECOND(Time)

**Usage:** The SECOND function returns the seconds component of a given Time value.

## 14. TIME

**Syntax:** TIME(Hour, Minute, Second)

**Usage:** The function returns a time value based on the specified Hour, Minute, and Second

## 15. TIMEVALUE

**Syntax:** TIMEVALUE(TimeString)

**Usage:** The function converts a time represented as a text string into a time value.

## 16. TODAY

**Syntax:** TODAY()

**Usage:** The function returns the current date (without the time component) as a date value.

## 17. WEEKDAY

**Syntax:** WEEKDAY(Date, [ReturnType])

**Usage:** The WEEKDAY function returns the day of the week for a given Date, where 1 represents Sunday and 7 represents Saturday. The optional ReturnType parameter allows you to customize the day numbering.

## 18. WEEKNUM

**Syntax:** WEEKNUM(Date, [ReturnType])

**Usage:** The WEEKNUM function returns the week number for a given Date, according to the ISO standard. The optional ReturnType parameter allows you to adjust the week numbering system.

## 19. YEAR

**Syntax:** YEAR(Date)

**Usage:** The YEAR function returns the year component of a given Date.

## 20. YEARFRAC

**Syntax:** YEARFRAC(StartDate, EndDate, [DayCountConvention])

**Usage:** The YEARFRAC function calculates the fraction of a year between two dates. The optional DayCountConvention parameter allows you to choose the method for counting days.

These DAX date and time functions provide various ways to work with dates, times, and datetime values in Power BI and other Microsoft Power Platform tools. They are useful for time-based calculations, extracting components from date values, and handling date-related data.

# DAX INFORMATION FUNCTIONS

DAX information functions along with their syntax and usage:

### 1. CONTAINS

**Syntax:** CONTAINS(TableOrColumn, SearchValue, [Expression])

**Usage:** The CONTAINS function checks if the specified TableOrColumn contains the SearchValue. If the optional Expression is provided, it filters the table before checking for the presence of the SearchValue.

### 2. CustomData

**Syntax:** CustomData(Key)

**Usage:** The CustomData function retrieves custom data associated with the Key from the data model. CustomData is typically used in Power BI paginated reports or in specific cases where custom data is required.

### 3. ISBLANK

**Syntax:** ISBLANK(Value)

**Usage:** The ISBLANK function checks if the specified Value is blank (empty or NULL). It returns TRUE if the Value is blank; otherwise, it returns FALSE.

### 4. ISERROR

**Syntax:** ISERROR(Value)

**Usage:** The ISERROR function checks if the specified Value contains an error. It returns TRUE if the Value is an error; otherwise, it returns FALSE.

### 5. ISEMPY

**Syntax:** ISEMPY(Value)

**Usage:** The ISEMPY function checks if the specified Value is empty (empty string or blank). It returns TRUE if the Value is empty; otherwise, it returns FALSE.

### 6. ISEVEN

**Syntax:** ISEVEN(Number)

**Usage:** The ISEVEN function checks if the specified Number is even. It returns TRUE if the Number is even; otherwise, it returns FALSE.

## 7. ISLOGICAL

**Syntax:** ISLOGICAL(Value)

**Usage:** The ISLOGICAL function checks if the specified Value is a logical (Boolean) value. It returns TRUE if the Value is logical; otherwise, it returns FALSE.

## 8. ISNONTEXT

**Syntax:** ISNONTEXT(Value)

**Usage:** The ISNONTEXT function checks if the specified Value is not a text (string) value. It returns TRUE if the Value is not text; otherwise, it returns FALSE.

## 9. ISNUMBER

**Syntax:** ISNUMBER(Value)

**Usage:** The ISNUMBER function checks if the specified Value is a numeric value. It returns TRUE if the Value is numeric; otherwise, it returns FALSE.

## 10. ISODD

**Syntax:** ISODD(Number)

**Usage:** The ISODD function checks if the specified Number is odd. It returns TRUE if the Number is odd; otherwise, it returns FALSE.

## 11. ISONORAFTER

**Syntax:** ISONORAFTER(Expression, AlternateResult)

**Usage:** The ISONORAFTER function returns the value of the Expression if it exists; otherwise, it returns the AlternateResult. It is useful for providing default values or handling missing values.

## 12. ISTEXT

**Syntax:** ISTEXT(Value)

**Usage:** The ISTEXT function checks if the specified Value is a text (string) value. It returns TRUE if the Value is text; otherwise, it returns FALSE.

## 13. LOOKUPVALUE

**Syntax:** LOOKUPVALUE(ResultColumn, SearchColumn, SearchValue, [AlternateResult])

**Usage:** The LOOKUPVALUE function looks up a value from a table based on the SearchValue in the SearchColumn and returns the corresponding value from the ResultColumn. If the optional AlternateResult is provided, it returns it when no match is found.

#### 14. USERNAME

**Syntax:** USERNAME()

**Usage:** The USERNAME function returns the current user's name who is viewing the report or data model. It is commonly used for implementing row-level security or auditing features.

These DAX information functions help you retrieve and analyze data characteristics and handle missing or erroneous values in Power BI and other Microsoft Power Platform tools.

## DAX LOGICAL FUNCTIONS

DAX logical functions along with their syntax and usage:

## 1. AND

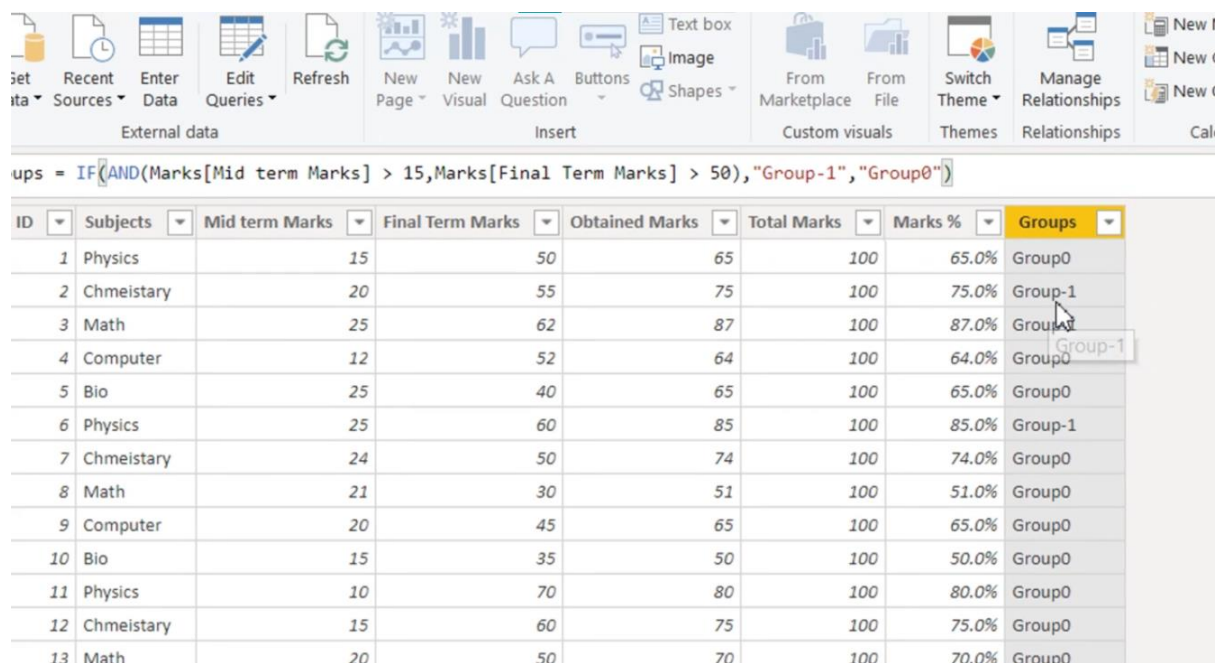
**Syntax:** AND(Condition1, Condition2, ...)

AND(<logical\_expression1>,<logical expression2>)

**Usage:** The AND function returns TRUE if all the specified conditions are TRUE; otherwise, it returns FALSE.

- Returns TRUE if both arguments are TRUE, otherwise FALSE.
- Use nested AND functions for multiple logical expressions, or use AND operator &&

**EX. GROUPS = IF(AND(Marks[Mid term marks] > 15, Marks[Final Term Marks] > 50), "Group-1", "Group0")**



ID	Subjects	Mid term Marks	Final Term Marks	Obtained Marks	Total Marks	Marks %	Groups
1	Physics	15	50	65	100	65.0%	Group0
2	Chmeistary	20	55	75	100	75.0%	Group-1
3	Math	25	62	87	100	87.0%	Group0
4	Computer	12	52	64	100	64.0%	Group0
5	Bio	25	40	65	100	65.0%	Group0
6	Physics	25	60	85	100	85.0%	Group-1
7	Chmeistary	24	50	74	100	74.0%	Group0
8	Math	21	30	51	100	51.0%	Group0
9	Computer	20	45	65	100	65.0%	Group0
10	Bio	15	35	50	100	50.0%	Group0
11	Physics	10	70	80	100	80.0%	Group0
12	Chmeistary	15	60	75	100	75.0%	Group0
13	Math	20	50	70	100	70.0%	Group0

**EX. Groups = IF(OR(AND(Marks[Mid term Marks] > 15, Marks[Final Term Marks] > 50), Marks[Subjects] = "Math"), "Group-1", "Group0")**



1 Groups = IF(OR(AND(Marks[Mid term Marks] > 15,Marks[Final Term Marks] > 50),Marks[Subjects] = "Math"),"Group-1","Group0")									
Student ID	Marks ID	Subjects	Mid term Marks	Final Term Marks	Obtained Marks	Total Marks	Marks %	Groups	
1	1	Physics	15	50	65	100	65.0%	Group0	
1	2	Chmeistary	20	55	75	100	75.0%	Group-1	
1	3	Math	25	62	87	100	87.0%	Group-1	
1	4	Computer	12	52	64	100	64.0%	Group0	
1	5	Bio	25	40	65	100	65.0%	Group0	
2	6	Physics	25	60	85	100	85.0%	Group-1	
2	7	Chmeistary	24	50	74	100	74.0%	Group0	
2	8	Math	21	30	51	100	51.0%	Group-1	
2	9	Computer	20	45	65	100	65.0%	Group0	
2	10	Bio	15	35	50	100	50.0%	Group0	
3	11	Physics	10	70	80	100	80.0%	Group0	
3	12	Chmeistary	15	60	75	100	75.0%	Group0	
3	13	Math	20	50	70	100	70.0%	Group-1	
3	14	Computer	25	40	65	100	65.0%	Group0	
3	15	Bio	30	35	65	100	65.0%	Group0	
4	16	Physics	30	60	90	100	90.0%	Group-1	
4	17	Chmeistary	25	65	90	100	90.0%	Group-1	
4	18	Math	20	25	45	100	45.0%	Group-1	
4	19	Computer	15	35	50	100	50.0%	Group0	
4	20	Bio	25	20	45	100	45.0%	Group0	
5	21	Physics	25	50	75	100	75.0%	Group0	
5	22	Chmeistary	23	45	68	100	68.0%	Group0	
5	23	Math	25	26	51	100	51.0%	Group-1	
5	24	Computer	26	40	66	100	66.0%	Group0	
5	25	Bio	20	45	65	100	65.0%	Group0	

## 2. IF Function

**Syntax:** IF(Condition, ResultIfTrue, ResultIfFalse)

**Usage:** The IF function checks a Condition, and if it evaluates to TRUE, it returns ResultIfTrue; otherwise, it returns ResultIfFalse.

**EX.**

1 Groups = IF(Marks[Subjects] IN {"Computer", "Math", "Physics"}, "Group-1", "Group-0")									
Marks ID	Subjects	Mid term Marks	Final Term Marks	Obtained Marks	Total Marks	Marks %	Abbreviation		
1	Physics	15	50	65	100	65.0%	Py		
2	Chmeistary	20	55	75	100	75.0%	Che		
3	Math	25	62	87	100	87.0%	MH		
4	Computer	12	52	64	100	64.0%	CS		
5	Bio	25	40	65	100	65.0%	Bio		
6	Physics	25	60	85	100	85.0%	Py		
7	Chmeistary	24	50	74	100	74.0%	Che		
8	Math	21	30	51	100	51.0%	MH		
9	Computer	20	45	65	100	65.0%	CS		
10	Bio	15	35	50	100	50.0%	Bio		
11	Physics	10	70	80	100	80.0%	Py		
12	Chmeistary	15	60	75	100	75.0%	Che		
13	Math	20	50	70	100	70.0%	MH		
14	Computer	25	40	65	100	65.0%	CS		

## ERROR HANDLING

Data Analysis Expressions (DAX) is a formula language primarily used in Microsoft Power BI, Power Pivot, and Analysis Services for creating custom calculations in data models. Error handling in DAX is essential to ensure that your calculations and expressions handle unexpected situations gracefully and provide meaningful results or messages to users. Here are some techniques for error handling in DAX:

**IFERROR Function:** The IFERROR function can be used to handle errors in DAX expressions. It allows you to specify an expression to evaluate and a value to return if the expression results in an error.

**Syntax:** IFERROR(Value, ValueIfError)

IFERROR(<expression> <value if error>)

**Usage:** The IFERROR function checks if the specified Value contains an error, and if it does, it returns ValueIfError; otherwise, it returns the original Value.

- Returns a specified value if expression gives Error, otherwise returns expression value.
- Used for error-handling.

IFERROR(A, B) => IF(ISERROR (A), B, A)

**IFERROR(expression, value\_if\_error)**

For example, you can use it to handle division by zero errors:

**DivisionResult = IFERROR(100 / [Denominator], 0)**

**ISERROR Function:** The ISERROR function can be used to check if an expression results in an error and return a Boolean value indicating whether an error occurred.

**ISERROR(expression)**

This can be used in combination with other functions to handle errors conditionally.

**EX. ISERROR = IF(ISERROR(MARKS[Obtained Marks]/ Marks[TestValue]), “There is an error”, “No Error”)**

Student ID	Marks ID	Subjects	Mid term Marks	Final Term Marks	Obtained Marks	Total Marks	Marks %	Abbreviation	Groups	Testvalue	Iserror
1	1	Physics	15	50	65	100	65.0%	Py	Group-1	2	No error
1	2	Chmeistary	20	55	75	100	75.0%	Che	Group-0	2	No error
1	3	Math	25	62	87	100	87.0%	MH	Group-1	2	No error
1	4	Computer	12	52	64	100	64.0%	CS	Group-1	0	There is an error
1	5	Bio	25	40	65	100	65.0%	Bio	Group-0	2	No error
2	6	Physics	25	60	85	100	85.0%	Py	Group-1	2	No error
2	7	Chmeistary	24	50	74	100	74.0%	Che	Group-0	2	No error
2	8	Math	21	30	51	100	51.0%	MH	Group-1	2	No error
2	9	Computer	20	45	65	100	65.0%	CS	Group-1	0	There is an error
2	10	Bio	15	35	50	100	50.0%	Bio	Group-0	2	No error
3	11	Physics	10	70	80	100	80.0%	Py	Group-1	2	No error
3	12	Chmeistary	15	60	75	100	75.0%	Che	Group-0	2	No error
3	13	Math	20	50	70	100	70.0%	MH	Group-1	2	No error
3	14	Computer	25	40	65	100	65.0%	CS	Group-1	0	There is an error
3	15	Bio	30	35	65	100	65.0%	Bio	Group-0	2	No error
4	16	Physics	30	60	90	100	90.0%	Py	Group-1	2	No error
4	17	Chmeistary	25	65	90	100	90.0%	Che	Group-0	2	No error
4	18	Math	20	25	45	100	45.0%	MH	Group-1	2	No error
4	19	Computer	15	35	50	100	50.0%	CS	Group-1	0	There is an error
4	20	Bio	25	20	45	100	45.0%	Bio	Group-0	2	No error
5	21	Physics	25	50	75	100	75.0%	Py	Group-1	2	No error
5	22	Chmeistary	23	45	68	100	68.0%	Che	Group-0	2	No error
5	23	Math	25	26	51	100	51.0%	MH	Group-1	2	No error
5	24	Computer	26	40	66	100	66.0%	CS	Group-1	0	There is an error
5	25	Bio	20	45	65	100	65.0%	Bio	Group-0	2	No error

**ISBLANK Function:** The ISBLANK function checks if a given value or expression is blank (NULL).

**ISBLANK(expression)**

This can be useful to handle cases where you want to perform calculations only if a specific column or measure is not blank.

**DIVIDE Function:** The DIVIDE function is useful for handling division operations, as it performs the division and handles division by zero errors gracefully.

**DIVIDE(numerator, denominator, alternative\_result)**

The **alternative\_result** parameter specifies the value to return when the denominator is zero.

**TRY Function:** The TRY function is available in some versions of Power BI and DAX. It allows you to attempt a calculation and gracefully handle errors by returning a default value if an error occurs.

**TRY(expression, alternative\_result)**

This function is particularly useful for handling complex calculations that might encounter errors.

**Custom Error Messages:** You can also use DAX functions like CONCATENATEX, FILTER, and COUNTROWS to create custom error messages or alerts based on certain conditions.

Remember that effective error handling not only ensures your calculations work correctly but also provides a better user experience by displaying meaningful messages or values when unexpected situations arise. Always consider the potential error scenarios that might occur in your data and create appropriate handling mechanisms.

### 3. NOT Function

**Syntax:** NOT(Value)

**Usage:** The NOT function reverses the logical value of the specified Value. If Value is TRUE, it returns FALSE, and if Value is FALSE, it returns TRUE.

### 4. OR Function

**Syntax:** OR(Condition1, Condition2, ...)

OR(<logical expression 1>, <logical expression2>)

**Usage:** The OR function returns TRUE if at least one of the specified conditions is TRUE; otherwise, it returns FALSE.

- Returns TRUE if at least one of the two arguments is TRUE, otherwise FALSE.
- Use nested OR functions for multiple logical expressions, or use OR operator | |

**EX. Groups = IF(OR(Marks[Subjects] = "Math", Marks[Subjects] = "Physics"), "Group-1", "Group-0")**

ID	Marks ID	Subjects	Mid term Marks	Final Term Marks	Obtained Marks	Total Marks	Marks %	Groups
1	1	Physics	15	50	65	100	65.0%	Group-1
1	2	Chemistry	20	55	75	100	75.0%	Group-0
1	3	Math	25	62	87	100	87.0%	Group-1
1	4	Computer	12	52	64	100	64.0%	Group-0
1	5	Bio	25	40	65	100	65.0%	Group-0
2	6	Physics	25	60	85	100	85.0%	Group-1
2	7	Chemistry	24	50	74	100	74.0%	Group-0
2	8	Math	21	30	51	100	51.0%	Group-1
2	9	Computer	20	45	65	100	65.0%	Group-0
2	10	Bio	15	35	50	100	50.0%	Group-0
3	11	Physics	10	70	80	100	80.0%	Group-1
3	12	Chemistry	15	60	75	100	75.0%	Group-0
3	13	Math	20	50	70	100	70.0%	Group-1

EX. Groups = IF(  
 OR(Marks[Subjects] = "Math",  
 Marks[Subjects] = "Physics"),  
 Marks[Subjects] = "Computer"), "Group-1","Group0")

File

Home

Modeling

Help

Cut

Copy

Format Painter

Get Data

Recent Sources

Enter Data

Edit Queries

Refresh

New Page

New Visual

Ask A Question

Buttons

Text box

Image

Shapes

From Marketplace

From File

Switch Theme

Manage Relationships

New Measure

New Column

New Quick Measure

Clipboard

External data

Insert

Custom visuals

Themes

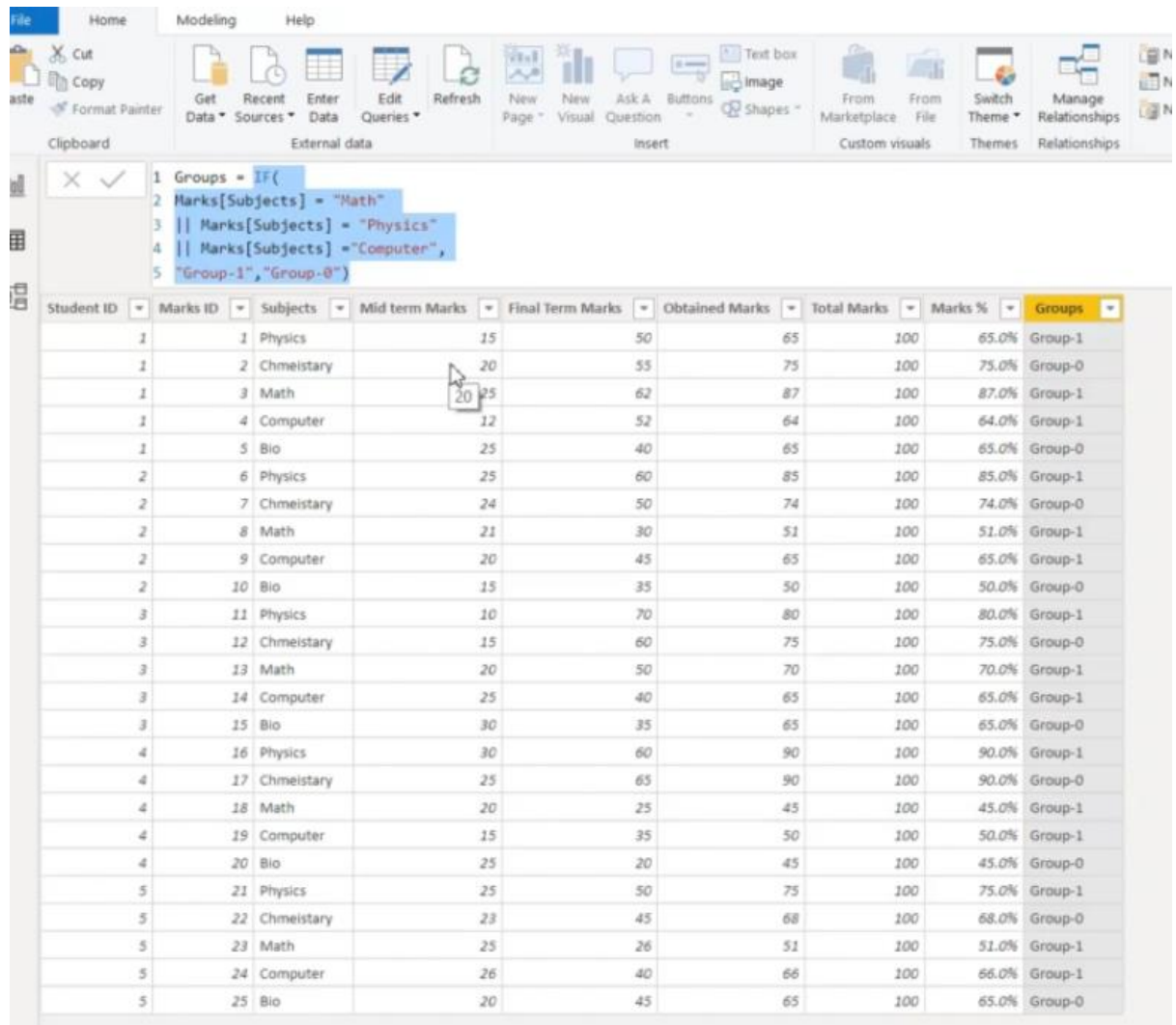
Relationships

Calculations

1 Groups = IF( OR(  
2 OR(Marks[Subjects] = "Math", Marks[Subjects] = "Physics"),Marks[Subjects] ="Computer"), "Group-1","Group-0")

Student ID	Marks ID	Subjects	Mid term Marks	Final Term Marks	Obtained Marks	Total Marks	Marks %	Groups
1	1	Physics	15	50	65	100	65.0%	Group-1
1	2	Chmeistary	20	55	75	100	75.0%	Group-0
1	3	Math	25	62	87	100	87.0%	Group-1
1	4	Computer	12	52	64	100	64.0%	Group-1
1	5	Bio	25	40	65	100	65.0%	Group-0
2	6	Physics	25	60	85	100	85.0%	Group-1
2	7	Chmeistary	24	50	74	100	74.0%	Group-0
2	8	Math	21	30	51	100	51.0%	Group-1
2	9	Computer	20	45	65	100	65.0%	Group-1
2	10	Bio	15	35	50	100	50.0%	Group-0
3	11	Physics	10	70	80	100	80.0%	Group-1
3	12	Chmeistary	15	60	75	100	75.0%	Group-0
3	13	Math	20	50	70	100	70.0%	Group-1
3	14	Computer	25	40	65	100	65.0%	Group-1
3	15	Bio	30	35	65	100	65.0%	Group-0
4	16	Physics	30	60	90	100	90.0%	Group-1
4	17	Chmeistary	25	65	90	100	90.0%	Group-0
4	18	Math	20	25	45	100	45.0%	Group-1
4	19	Computer	15	35	50	100	50.0%	Group-1
4	20	Bio	25	20	45	100	45.0%	Group-0
5	21	Physics	25	50	75	100	75.0%	Group-1
5	22	Chmeistary	23	45	68	100	68.0%	Group-0
5	23	Math	25	26	51	100	51.0%	Group-1
5	24	Computer	26	40	66	100	66.0%	Group-1
5	25	Bio	20	45	65	100	65.0%	Group-0

EX. Groups = IF(  
 Marks[Subjects] = "Math",  
 || Marks[Subjects] = "Physics",  
 || Marks[Subjects] = "Computer",  
 "Group-1","Group0")



The screenshot shows the Microsoft Excel interface. The formula bar at the top contains the following DAX formula:

```
1 Groups = IF(
2 Marks[Subjects] = "Math"
3 || Marks[Subjects] = "Physics"
4 || Marks[Subjects] = "Computer",
5 "Group-1", "Group-0")
```

Below the formula bar is a table with the following columns: Student ID, Marks ID, Subjects, Mid term Marks, Final Term Marks, Obtained Marks, Total Marks, Marks %, and Groups. The table contains 25 rows of data, showing marks for various subjects across different students. The 'Groups' column is populated based on the formula in the formula bar, with values like 'Group-1' and 'Group-0'.

Student ID	Marks ID	Subjects	Mid term Marks	Final Term Marks	Obtained Marks	Total Marks	Marks %	Groups
1	1	Physics	15	50	65	100	65.0%	Group-1
1	2	Chemistry	20	55	75	100	75.0%	Group-0
1	3	Math	25	62	87	100	87.0%	Group-1
1	4	Computer	12	52	64	100	64.0%	Group-1
1	5	Bio	25	40	65	100	65.0%	Group-0
2	6	Physics	25	60	85	100	85.0%	Group-1
2	7	Chemistry	24	50	74	100	74.0%	Group-0
2	8	Math	21	30	51	100	51.0%	Group-1
2	9	Computer	20	45	65	100	65.0%	Group-1
2	10	Bio	15	35	50	100	50.0%	Group-0
3	11	Physics	10	70	80	100	80.0%	Group-1
3	12	Chemistry	15	60	75	100	75.0%	Group-0
3	13	Math	20	50	70	100	70.0%	Group-1
3	14	Computer	25	40	65	100	65.0%	Group-1
3	15	Bio	30	35	65	100	65.0%	Group-0
4	16	Physics	30	60	90	100	90.0%	Group-1
4	17	Chemistry	25	65	90	100	90.0%	Group-0
4	18	Math	20	25	45	100	45.0%	Group-1
4	19	Computer	15	35	50	100	50.0%	Group-1
4	20	Bio	25	20	45	100	45.0%	Group-0
5	21	Physics	25	50	75	100	75.0%	Group-1
5	22	Chemistry	23	45	68	100	68.0%	Group-0
5	23	Math	25	26	51	100	51.0%	Group-1
5	24	Computer	26	40	66	100	66.0%	Group-1
5	25	Bio	20	45	65	100	65.0%	Group-0

## 5. SWITCH

**Syntax:** SWITCH(Expression, Value1, Result1, Value2, Result2, ..., [DefaultResult])

SWITCH(<expression>, <value> <result value2>, <result2>] [<else\_result>])

**Usage:** The SWITCH function evaluates the Expression against multiple values and returns the corresponding Result for the matching Value. If none of the Values match the Expression and the DefaultResult is provided, it returns DefaultResult.

**EX.** REG = SWITCH(Orders[Region],

    "EAST", "E",  
    "WEST", "W",  
    "SOUTH", "S",  
    "C")

- Evaluates an expression against a list of values and returns corresponding result.
- All the results including else result must be of same data type.
- Default value for else result is BLANK.

**Gender Full =**

SWITCH(People[Gender],  
    "F", "FEMALE",  
    "M", "MALE",  
    "T", "TRANSGENDER",  
    "OTHERS")

**Measure 4 =**

SWITCH(TRUE(),  
    SUM(Orders[Profit])>=20000, "High Profit",  
    SUM(Orders[Profit])>=0, "Low Profit",  
    "Loss")

EX. Groups =

```
SWITCH( Marks[Subjects],
"Physics", "py",
"Chemistry", "che",
"Math", "MH",
"Computer", "CS",
"Bio", "Bio", "unknown"
)
```

ent ID	Marks ID	Subjects	Mid term Marks	Final Term Marks	Obtained Marks	Total Marks	Marks %	Group
1	1	Physics	15	50	65	100	65.0%	Py
1	2	Chmeistary	20	55	75	100	75.0%	Che
1	3	Math	25	62	87	100	87.0%	MH
1	4	Computer	12	52	64	100	64.0%	CS
1	5	Bio	25	40	65	100	65.0%	Bio
2	6	Physics	25	60	85	100	85.0%	Py
2	7	Chmeistary	24	50	74	100	74.0%	Che
2	8	Math	21	30	51	100	51.0%	MH
2	9	Computer	20	45	65	100	65.0%	CS

## 6. TRUE Function

**Syntax:** TRUE()

**IF(<logical\_condition>, <value\_if\_true>, [<value\_if\_false>])**

**Usage:** The TRUE function returns the logical value TRUE.

## 7. FALSE

**Syntax:** FALSE()

**Usage:** The FALSE function returns the logical value FALSE.

- Returns one of the two values depending upon logical condition.
- Any Boolean expression in place of logical condition.
- Default value for value if false is BLANK
- In the case of nested IF functions, better to use the SWITCH function.



The IF function can return a variant data type if value\_if\_true and value\_if\_false are of different data types. But if both value if true and value if false are of numeric data types (whole number and decimal number), it will implicitly convert data types to accommodate both values

EX. PROFIT/LOSS = IF(SUM(Orders[Profit]) < 0, "LOSS", "PROFIT")

Measure = IF(SUM(Orders[Profit]) >= 200000, "HIGH PROFIT", IF(SUM(Orders[Profit]) >= 0, "LOW PROFIT", "LOSS"))

These DAX logical functions allow you to perform logical operations and build complex conditions and expressions in Power BI and other Microsoft Power Platform tools. They are essential for decision-making and flow control in DAX calculations and formulas.

## DAX MATHEMATICAL & TRIGONOMETRIC FUNCTIONS

DAX mathematical and trigonometric functions along with their syntax and usage:

### 1. CEILING Function

**Syntax:** CEILING(Number, Significance)

**Usage:** The CEILING function rounds a Number up to the nearest multiple of Significance.

### 2. CURRENCY Function

**Syntax:** CURRENCY(Number, DecimalPlaces)

**Usage:** The CURRENCY function formats a Number as a currency value with the specified DecimalPlaces.

### 3. DEGREES Function

**Syntax:** DEGREES(Angle)

**Usage:** The DEGREES function converts an Angle in radians to degrees.

### 4. DIVIDE Function

**Syntax:** DIVIDE(Numerator, Denominator, [Alternate Result])

**Usage:** The DIVIDE function divides the Numerator by the Denominator. If the Denominator is zero, it returns the optional Alternate Result or NULL if Alternate Result is not provided.

### 5. EVEN Function

**Syntax:** EVEN(Number)

**Usage:** The EVEN function rounds a Number up to the nearest even integer.

### 6. EXP Function

**Syntax:** EXP(Number)

**Usage:** The EXP function returns the exponential value of a given Number ( $e^{\text{Number}}$ ).

## 7. FACT Function

**Syntax:** FACT(Number)

**Usage:** The FACT function returns the factorial of a given Number.

## 8. FLOOR Function

**Syntax:** FLOOR(Number, Significance)

**Usage:** The FLOOR function rounds a Number down to the nearest multiple of Significance.

## 9. MROUND Function

**Syntax:** MROUND(Number, Multiple)

**Usage:** The MROUND function rounds a Number to the nearest multiple of the specified Multiple.

## 10. MOD Function

**Syntax:** MOD(Number, Divisor)

**Usage:** The MOD function returns the remainder of the division of Number by Divisor.

## 11. ODD Function

**Syntax:** ODD(Number)

**Usage:** The ODD function rounds a Number up to the nearest odd integer.

## 12. POWER Function

**Syntax:** POWER(Base, Exponent)

**Usage:** The POWER function raises the Base to the power of Exponent.

## 13. RAND Function

**Syntax:** RAND()

**Usage:** The RAND function returns a random decimal number between 0 and 1.

#### 14. RANDBETWEEN

**Syntax:** RANDBETWEEN(Bottom, Top)

**Usage:** The RANDBETWEEN function returns a random integer between the specified Bottom and Top values.

#### 15. ROUND

**Syntax:** ROUND(Number, NumDigits)

**Usage:** The ROUND function rounds a Number to the specified number of decimal NumDigits.

#### 16. ROUNDDOWN

**Syntax:** ROUNDDOWN(Number, NumDigits)

**Usage:** The ROUNDDOWN function rounds a Number down to the specified number of decimal NumDigits.

#### 17. ROUNDUP

**Syntax:** ROUNDUP(Number, NumDigits)

**Usage:** The ROUNDUP function rounds a Number up to the specified number of decimal NumDigits.

#### 18. SQRT Function

**Syntax:** SQRT(Number)

**Usage:** The SQRT function returns the square root of a given Number.

#### 19. SQRTPI

**Syntax:** SQRTPI(Number)

**Usage:** The SQRTPI function returns the square root of (Number \*  $\pi$ ).

These DAX mathematical and trigonometric functions allow you to perform various numerical and trigonometric calculations in Power BI and other Microsoft Power Platform tools. They are essential for working with numeric data and performing complex mathematical operations.

## DAX STATISTICAL FUNCTIONS

DAX statistical functions along with their syntax and usage:

### 1. MEDIAN

**Syntax:** MEDIAN(Number1, [Number2], ...)

**Usage:** The MEDIAN function returns the median value from a list of numbers.

### 2. MEDIANX

**Syntax:** MEDIANX(Table, Expression)

**Usage:** The MEDIANX function returns the median value of the Expression evaluated over the rows of the specified Table.

### 3. PERCENTILE.EXC

**Syntax:** PERCENTILE.EXC(Column, K)

**Usage:** The PERCENTILE.EXC function returns the Kth percentile of a column, excluding the top and bottom values.

### 4. PERCENTILE.INC

**Syntax:** PERCENTILE.INC(Column, K)

**Usage:** The PERCENTILE.INC function returns the Kth percentile of a column, including the top and bottom values.

### 5. PERCENTILEX.EXC

**Syntax:** PERCENTILEX.EXC(Table, Expression, K)

**Usage:** The PERCENTILEX.EXC function returns the Kth percentile of the Expression evaluated over the rows of the specified Table, excluding the top and bottom values.

### 6. PERCENTILEX.INC

**Syntax:** PERCENTILEX.INC(Table, Expression, K)

**Usage:** The PERCENTILEX.INC function returns the Kth percentile of the Expression evaluated over the rows of the specified Table, including the top and bottom values.

## 7. POISSON.DIST

**Syntax:** POISSON.DIST(X, Mean, Cumulative)

**Usage:** The POISSON.DIST function returns the Poisson distribution for a given value of X, with the specified Mean. If Cumulative is TRUE, it returns the cumulative Poisson distribution.

## 8. RANK.EQ

**Syntax:** RANK.EQ(Number, Ref, [Order])

**Usage:** The RANK.EQ function returns the rank of a Number in a list of numbers, considering the specified Order.

**BY ANOTHER WAY WE CAN ACHIEVE RANKING**

**EX.**

The screenshot shows the Power BI DAX editor interface. The formula bar contains the following DAX code:

```
1 Ranking =  
2 VAR var_exp = Employee[Experience]  
3 Return  
4 COUNTROWS(  
5     FILTER(Employee, Employee[Experience] > var_exp)  
6 ) + 1
```

Below the formula bar, a table is displayed with the following columns: EmpID, Name, Experience, and Ranking. The table contains 24 rows of data, sorted by Experience in descending order. The Ranking column shows the rank of each employee based on their experience, with the highest experience (23) receiving a rank of 1.

EmpID	Name	Experience	Ranking
20	Emp- 20	23	1
7	Emp- 7	23	1
21	Emp- 21	21	1
17	Emp- 17	21	1
18	Emp- 18	20	5
14	Emp- 14	15	6
22	Emp- 22	14	7
13	Emp- 13	13	8
12	Emp- 12	12	9
15	Emp- 15	10	10
16	Emp- 16	9	11
8	Emp- 8	9	11
11	Emp- 11	7	13
6	Emp- 6	6	14
9	Emp- 9	5	15
2	Emp- 2	5	15
23	Emp- 23	4	17
4	Emp- 4	4	17
24	Emp- 24	3	19
10	Emp- 10	3	19
3	Emp- 3	3	19
5	Emp- 5	2	22
1	Emp- 1	2	22
19	Emp- 19	1	24

## 9. RANKX

**Syntax:** RANKX(Table, Expression, Measure, [Order], [Ties])

**Usage:** The RANKX function returns the rank of the Expression evaluated over the rows of the specified Table, using the Measure, Order, and Ties parameters.

## 10. SAMPLE

**Syntax:** SAMPLE(Number1, [Number2], ...)

**Usage:** The SAMPLE function returns a random sample value from a list of numbers.

## 11. STDEV.P

**Syntax:** STDEV.P(Number1, [Number2], ...)

**Usage:** The STDEV.P function returns the standard deviation for a population based on a list of numbers.

## 12. STDEV.S

**Syntax:** STDEV.S(Number1, [Number2], ...)

**Usage:** The STDEV.S function returns the standard deviation for a sample based on a list of numbers.

## 13. STDEVX.P

**Syntax:** STDEVX.P(Table, Expression)

**Usage:** The STDEVX.P function returns the standard deviation for a population of the Expression evaluated over the rows of the specified Table.

## 14. STDEVX.S

**Syntax:** STDEVX.S(Table, Expression)

**Usage:** The STDEVX.S function returns the standard deviation for a sample of the Expression evaluated over the rows of the specified Table.

## 15. VAR.P

**Syntax:** VAR.P(Number1, [Number2], ...)

**Usage:** The VAR.P function returns the variance for a population based on a list of numbers.

## 16. VAR.S

**Syntax:** VAR.S(Number1, [Number2], ...)

**Usage:** The VAR.S function returns the variance for a sample based on a list of numbers.

## 17. VARX.P

**Syntax:** VARX.P(Table, Expression)

**Usage:** The VARX.P function returns the variance for a population of the Expression evaluated over the rows of the specified Table.

## 18. VARX.S

**Syntax:** VARX.S(Table, Expression)

**Usage:** The VARX.S function returns the variance for a sample of the Expression evaluated over the rows of the specified Table.

These DAX statistical functions allow you to perform statistical calculations and analysis in Power BI and other Microsoft Power Platform tools. They are essential for data exploration and understanding the distribution and variability of data.



## DAX TEXT FUNCTIONS

DAX text functions along with their syntax and usage:

### 1. BLANK Function

**Syntax:** BLANK()

**Usage:** The BLANK function returns a blank value, equivalent to a NULL value.

### 2. CODE Function

**Syntax:** CODE(Text)

**Usage:** The CODE function returns the ASCII value of the first character in the Text.

### 3. CONCATENATE

**Syntax:** CONCATENATE(Text1, [Text2], ...)

**Usage:** The CONCATENATE function combines multiple Text strings into a single text string.

### 4. CONCATENATEX

**Syntax:** CONCATENATEX(Table, Expression, [Delimiter])

**Usage:** The CONCATENATEX function concatenates the values resulting from the Expression, evaluated over the rows of the specified Table, using the optional Delimiter to separate the values.

### 5. EXACT

**Syntax:** EXACT(Text1, Text2)

**Usage:** The EXACT function checks if Text1 is exactly equal to Text2, including letter case.

EX. Measure = EXACT("SUMIT", "SUMIT")

### 6. FIND

**Syntax:** FIND(FindText, WithinText, [StartNum])

**Usage:** The FIND function returns the starting position of FindText within WithinText. Optionally, you can specify the StartNum to begin the search from a particular character position.

EX. Measure = FIND("E", SELECTEDVALUE(Orders[Category]), 1, -999)

Measure = FIND("DA", SELECTEDVALUE(Orders[Customer Name]))

## 7. FIXED

**Syntax:** FIXED(Value, [DecimalPlaces], [NoCommas])

**Usage:** The FIXED function formats a numeric Value as text with a fixed number of DecimalPlaces. If NoCommas is TRUE, it does not include commas as thousands separators.

## 8. FORMAT

**Syntax:** FORMAT(Value, FormatText)

**Usage:** The FORMAT function formats a Value using a custom FormatText as specified.

## 9. LEFT Function

**Syntax:** LEFT(Text, NumChars)

**Usage:** The LEFT function returns the leftmost NumChars characters from the Text.

EX. Measure = LEFT(SELECTEDVALUE(Orders[Customer Name]))

## 10. LEN Function

**Syntax:** LEN(Text)

**Usage:** The LEN function returns the number of characters in the Text.

EX. Measure = LEN("Python")

Measure = LEN(SELECTEDVALUE(Orders[Customer Name]))

## 11. LOWER

**Syntax:** LOWER(Text)

**Usage:** The LOWER function converts the Text to lowercase.

EX. Measure = LOWER(SELECTEDVALUE(Orders[Customer Name]))

## 12. MID Function

**Syntax:** MID(Text, StartNum, NumChars)

**Usage:** The MID function returns a substring from the Text, starting at StartNum, and containing NumChars characters.

EX. Measure = MID(SELECTEDVALUE(Orders[Customer Name]), 3, 5)

### 13. REPLACE Function

**Syntax:** REPLACE(OldText, StartNum, NumChars, NewText)

**Usage:** The REPLACE function replaces a portion of the OldText with NewText, starting at StartNum, and replacing NumChars characters.

EX. Test\_Measure = REPLACE(SELECTEDVALUE (Orders(Customer Name)), "Da", "AAAAA")

### 14. REPT

**Syntax:** REPT(Text, NumberTimes)

**Usage:** The REPT function repeats the Text for NumberTimes.

### 15. RIGHT

**Syntax:** RIGHT(Text, NumChars)

**Usage:** The RIGHT function returns the rightmost NumChars characters from the Text.

EX. Measure = RIGHT(SELECTEDVALUE(Orders[Customer Name]), 5)

### 16. SEARCH

**Syntax:** SEARCH(FindText, WithinText, [StartNum])

**Usage:** The SEARCH function returns the starting position of FindText within WithinText, ignoring letter case. Optionally, you can specify the StartNum to begin the search from a particular character position.

### 17. SUBSTITUTE

**Syntax:** SUBSTITUTE(Text, OldText, NewText, [Occurrence])

**Usage:** The SUBSTITUTE function replaces occurrences of OldText with NewText in the Text. Optionally, you can specify the Occurrence to replace only a specific occurrence.

EX. Test\_Measure = SUBSTITUTE(SELECTEDVALUE (Orders(Customer Name)), "Da", "AAAAA")  
Test\_Measure = SUBSTITUTE("Sumit Suman", "Sum", "Am", 1)

### 18. TRIM

**Syntax:** TRIM(Text)

**Usage:** The TRIM function removes leading and trailing spaces from the Text.

## 19. UPPER Function

**Syntax:** UPPER(Text)

**Usage:** The UPPER function converts the Text to uppercase.

EX. Measure = UPPER(SELECTEDVALUE(Orders[Customer Name]))

## 20. VALUE Function

**Syntax:** VALUE(Text)

**Usage:** The VALUE function converts the Text to a numeric value.

These DAX text functions allow you to manipulate and format text values in Power BI and other Microsoft Power Platform tools. They are essential for working with text-based data and customizing report outputs.

## DAX OTHER FUNCTIONS

DAX other functions along with their syntax and usage:

### 1. EXCEPT

**Syntax:** EXCEPT(Table1, Table2)

**Usage:** The EXCEPT function returns the rows from Table1 that are not present in Table2.

### 2. GROUPBY

**Syntax:** GROUPBY(Table, Group\_Column1, [Group\_Column2], ..., [Expression1], [Expression2], ...)

**Usage:** The GROUPBY function groups the rows of the specified Table based on the Group\_Columns and calculates the expressions specified for each group.

### 3. INTERSECT

**Syntax:** INTERSECT(Table1, Table2)

**Usage:** The function returns the rows that are common in both Table1 and Table2.

### 4. NATURALINNERJOIN

**Syntax:** NATURALINNERJOIN(Table1, Table2)

**Usage:** The NATURALINNERJOIN function returns the rows that are common in both Table1 and Table2, based on common column names.

### 5. NATURALLEFTOUTERJOIN

**Syntax:** NATURALLEFTOUTERJOIN(Table1, Table2)

**Usage:** The NATURALLEFTOUTERJOIN function returns all the rows from Table1 and the matching rows from Table2, based on common column names.

### 6. SUMMARIZECOLUMNS

**Syntax:** SUMMARIZECOLUMNS(Column1, [Column2], ..., [Expression1], [Expression2], ...)

**Usage:** The SUMMARIZECOLUMNS function creates a summary table with the specified columns and calculated expressions.

## 7. UNION

**Syntax:** UNION(Table1, Table2)

**Usage:** The UNION function returns the combination of rows from both Table1 and Table2.

## 8. VAR

**Syntax:** VAR(VariableName, Expression)

**Usage:** The VAR function creates a variable named VariableName and stores the result of the Expression in that variable. It can be used within other DAX expressions.

These DAX other functions provide additional functionalities for data manipulation and analysis in Power BI and other Microsoft Power Platform tools. They are useful for complex queries and aggregations when working with multiple tables and datasets.