# COLUMNS & MEASURES IN POWER BI

**In general, we use calculated columns and measures to perform arithmetic operations on top of the data that we already have in our model.**

**Calculated columns can compute aggregate values, the evaluation context is performed by default for each row, which suits calculated columns for some computations, but not others.**

**Measures are used to calculate aggregates, such as the sum or average of a column.**

***Measures are calculated at the time of your query, which means that they aren’t stored in your database, but use processing power to execute a query at the time of your request.***

**A simple example could be having a sales table with a Unit Price column and an Order Quantity column, and the desire to know the Order Total for each line, or in aggregate.**

|  |  |  |
| --- | --- | --- |
| OLTP / DB | SSAS - / CUBE | POWER BI / POWER PIVOT |
| SQL EXPRESSION | MDX | DAX |

## What is DAX (DATA ANALYSIS EXPRESSIONS)

***Data Analysis Expression language or DAX, was originally released with Power Pivot in Excel in 2010 by Microsoft. It’s a functional language that is very similar to excel, as you would expect as it was part of the excel tools. Its initial basic function was to allow excel users solve more complex business problems.***

## USING DAX

*DAX is used to create Calculated Columns, Measures and Tables in Power BI.*

*DAX Consists of Functions, Expressions, and Filter Conditions etc.*

|  |  |  |
| --- | --- | --- |
| COLUMNS | MEASURES | TABLE |
|  |  |  |

*Now, not only is DAX used in Power Pivot but it is also the language used in Analysis Services tabular and Power BI.*

***AGGREGRATE FUNCTIONS OF DAX***

|  |  |
| --- | --- |
| **Function** | **Description** |
| **AVERAGE(<column>)** | Returns the average (arithmetic mean) of all the numbers in a column. |
| **COUNT(<column>)** | The COUNT function counts the number of cells in a column that contain values.  COUNT(CANDIDATES[RESUME]) |
| **COUNTBLANK(<column>)** | Counts the number of blank cells in a column. (blank + null) values  COUNTBLANK(CANDIDATES[RESUME]) |
| **COUNTROWS(<table>)** | The COUNTROWS function counts the number of rows in the specified table, or in a table defined by an expression. |
| COUNT and [COUNTA](https://dax.guide/counta/) are identical in DAX for all the data types except Boolean.  [COUNTA](https://dax.guide/counta/) can operate on a Boolean data type, whereas COUNT cannot do that. | |
| **MAX(<column>)** | Returns the largest numeric value in a column. |
| **MIN(<column>)** | Returns the smallest numeric value in a column. Ignores logical values and text. |
| **SUM(<column>)** | Adds all the numbers in a column. |
|  |  |

### Filter and Value Functions (Table Functions)

|  |  |
| --- | --- |
| **Function** | **Description** |
| **ALL(<table\_or\_column>)** | Returns all the rows in a table, or all the values in a column, ignoring any filters that might have been applied. |
| **Table2 = ALL(DimSalesTerritory)**  **ALL(<table> or <column>)**  **ALL(<columnname1>,<columnname2>)**  **Count of all sales orders = COUNTROWS(ALL('Sales'))**  NewCourses = ALL(COURSES\_DETAIL[COURSENAME],COURSES\_DETAIL[FEES]) | |
| **ALLEXCEPT(<table>,column1>,<column2>,…)** | Overrides all context filters in the table except filters that have been applied to the specified columns. |
| **Table 2 = ALLEXCEPT(DimSalesTerritory,DimSalesTerritory[SalesTerritoryKey],DimSalesTerritory[SalesTerritoryRegion])**  **Returns a Table except the column specified**  NewSales = ALLEXCEPT(LocationSales,LocationSales[Projected Gross Margin ],LocationSales[Projected Gross Profit ]) | |
| BLANK() | Returns a blank. |
| **CALCULATE(<expression>,<filter1>,<filter2>…)** | Evaluates an expression in a context that is modified by the specified filters. |

|  |  |
| --- | --- |
| CALCULATETABLE( <expression>, <filter1>, <filter2>,…) | CALCULATETABLE( <table>, <filter1>, <filter2>,…)  Evaluates a table expression in a context modified by filters.  NewTable1 = CALCULATETABLE(LocationSales,Location[Country]="INDIA")  SQLCandidatesCourses = CALCULATETABLE(CANDIDATES,COURSES\_DETAIL[COURSENAME]="SQL SERVER") |
| DISTINCT(<column>) | Returns a one-column table that contains the distinct values from the specified column. |
| Table 2 = DISTINCT(DimSalesTerritory[SalesTerritoryKey]) | |
| **FILTER(<table>,<filter>)** | Returns a table that represents a subset of another table or expression. |
| Table 2 = FILTER(DimSalesTerritory,DimSalesTerritory[SalesTerritoryKey]>5)  **Count of sales orders over 100 = COUNTROWS(FILTER('Sales', 'Sales'[Sales] > 100))**  Count of sales greater than 3000= COUNTROWS(FILTER(FactInternetSales, FactInternetSales[SalesAmount] > 3000)) | |

|  |  |
| --- | --- |
| VALUES(<column>) | Returns a one-column table that contains the distinct values from the specified column. This function is similar to DISTINCT function, but VALUES function can also return Unknown member |
| **Table = VALUES(DimCustomer[AddressLine2])**  CourseId = VALUES(CANDIDATES[COURSEID]) | |
| ALLNONBLANKROW(?) | Returns all the rows, except for blank rows, in a table or column, and disregards any context filters that might exist. |
| Table = ALLNOBLANKROW(DimCustomer[AddressLine1]) | |
| FIRSTNONBLANK(<column>,<expression>) | Returns the first non-blank values in column, filtered by expression.  DateofFirstRegistration = CALCULATE(FIRSTNONBLANK(CANDIDATES[DateofRegistration],TRUE()))  FirstOrderDate = CALCULATE(FIRSTNONBLANK(FactSales[Order\_Date],TRUE()),                                                   FILTER(FactSales,YEAR(FactSales[Order\_Date])=2014))  FirstOrderValue = CALCULATE(SUM(FactSales[Net\_Sales]),                                                     FILTER(FactSales,YEAR(FactSales[Order\_Date])=2014),                                                     FIRSTNONBLANK(FactSales[Order\_Date],TRUE()))  https://excelkingdom.blogspot.com/2019/02/how-to-use-firstnonblank-and.html |
| LASTNONBLANK(<column>,<expression>) | The LASTNONBLANK() Function returns the last value in the column, filtered by the current context, where the expression is not blank. Syntax : LASTNONBLANK(<Column>,<Expression>)  Returns the LAST non-blank values in column, filtered by expression.  LastDateofRegistration = CALCULATE(LASTNONBLANK(CANDIDATES[DateofRegistration],TRUE())) |

**RELATED**

|  |
| --- |
| RELATED as the name itself says it will return the related value one table to another table. **This is similar to the lookup value function we have in MS Excel i.e. VLOOKUP.**  However in Power BI before we apply this DAX function we need to create a data model relationship between tables we are referring to.  Below is the syntax of the RELATED DAX function in Power BI.  Power BI RELATED Syntax  **We just need to select the result column from the other table and it will fetch the details for us.**  **EXAMPLE**  UnitPrice=RELATED(ProductPrice[Price])  Add a New Column CourseFees to Candidate Table in Power BI  CourseFees = RELATED(COURSES\_DETAIL[FEES])  It will add a new column **CourseFees to the Candidates Table.** |

## **Logical functions**

|  |  |
| --- | --- |
| **Function** | **Description** |
| **AND(<logical1>,<logical2>,…)** | **Checks whether all arguments are TRUE, and returns TRUE if all arguments are TRUE.**  **And(1,1) = true**  **And(1,0) = false** |
| **FALSE()** | **Returns the logical value FALSE.** |
| **IF( logical\_test>, <value\_if\_true>, value\_if\_false)** | **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.**  **If(tax> 100,”Higher”,”Lower”)**  CandidateStatus = if(CANDIDATES[CourseFees] > 10000,"High","Low") |
| **IFERROR(value, value\_if\_error)** | **Returns value\_if\_error if the first expression is an error and the value of the expression itself if otherwise.** |
| **NOT(<logical>)** | **Changes FALSE to TRUE, or TRUE to FALSE.** |
| **TRUE()** | **Returns the logical value TRUE.** |
| **OR(<logical1>,<logical2>,…)** | **Checks whether one of the arguments is TRUE to return TRUE. The function returns FALSE if all arguments are FALSE**  **OR(1,0) = true** |

## **Information functions**

|  |  |
| --- | --- |
| **Function** | **Description** |
| **ISBLANK(<value>)** | **Checks whether a value is blank, and returns TRUE or FALSE.** |
| **ISERROR(<value>)** | **Checks whether a value is an error, and returns TRUE or FALSE.** |
| **ISLOGICAL(<value>)** | **Checks whether a value is a logical value, (TRUE or FALSE), and returns TRUE or FALSE.** |
| **ISNONTEXT(<value>)** | **Checks whether a value is not text (blank cells are not text), and returns TRUE or FALSE.** |
| **ISNUMBER(<value>)** | **Checks whether a value is a number, and returns TRUE or FALSE.**  **Column 2 = ISNUMBER("test") = false** |
| **ISTEXT(<value>)** | **Checks whether a value is text, and returns TRUE or FALSE.** |

## 

## **Math and Trig Functions**

|  |  |
| --- | --- |
| **Function** | **Description** |
| **ABS(<number>)** | **Returns the absolute value of a number.**  **COL1 = ABS(101.11) = 101.11** |
| **CEILING(<number>, <significance>)** | **Rounds a number up, to the nearest integer or to the nearest multiple of significance.**  **COL1 = CEILING(101.1111,1) = 102**  = CEILING(4.6,1) returns 5. |
| **EXP(<number>)** | **Returns e raised to the power of a given number. The constant e equals 2.71828182845904, the base of the natural logarithm.** |
| **FACT(<number>)** | **Returns the factorial of a number, equal to the series 1\*2\*3\*...\* , ending in the given number.**  **COL1 = FACT(4) | OUTPUT : 24**  **4 \* 3\*2**  **5 \* 4\*3\*2\*1** |
| **FLOOR(<number>, <significance>)** | **Rounds a number down, toward zero, to the nearest multiple of significance.**  FLOOR(101.1,1) = 101 |
| **INT(<number>)** | **Rounds a number down to the nearest integer.** |
| **LN(<number>)** | **Returns the natural logarithm of a number. Natural logarithms are based on the constant e (2.71828182845904).** |
| **LOG(<number>,<base>)** | **Returns the logarithm of a number to the base you specify.** |
| **LOG10(<number>)** | **Returns the base-10 logarithm of a number.** |
| **MOD(<number>, <divisor>)** | **Returns the remainder after a number is divided by a divisor. The result always has the same sign as the divisor.**  **MOD(14,5) = 4**  **MOD(14,2) = 0** |
| **PI()** | **Returns the value of Pi, 3.14159265358979, accurate to 15 digits.**  **pi() \* 2 = 6.28** |
| **POWER(<number>, <power>)** | **Returns the result of a number raised to a power.**  **POWER(10,3) = 1000**  **POWER(10,4)=10000** |
| **QUOTIENT(<numerator>, <denominator>)** | **Performs division and returns only the integer portion of the division result. Use this function when you want to discard the remainder of division.**  **QUOTIENT(10,2)= 5** |
| **ROUND(<number>, <num\_digits>)** | **Rounds a number to the specified number of digits.**  **ROUND(10.111,2)=10.11**  **ROUND(10.567,2) = 10.57** |
| **ROUNDDOWN(<number>, <num\_digits>)** | **Rounds a number down, toward zero.**  = ROUNDDOWN (5.55555,2) returns 5.55.  = ROUNDDOWN (5.55555,0) returns 5. |
| **ROUNDUP(<number>, <num\_digits>)** | **Rounds a number up, away from 0 (zero).**  = ROUNDUP (5.55555,2) returns 5.56.  = ROUNDUP (5.55555,0) returns 6. |
| **SIGN(<number>)** | **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.** |
| **SQRT(<number>)** | **Returns the square root of a number.** |
| **TRUNC(<number>)** | **Truncates a number to an integer by removing the decimal, or fractional, part of the number.**  TRUNC (5.8) returns 5. |
| **[vb] RAND()** | **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.** |
| **[vb] RANDBETWEEN(<bottom>,<top>)** | **Returns a random number between the numbers you specify** |

## 

## **Text Functions**

|  |  |  |
| --- | --- | --- |
| **Function** | | **Description** |
| **UNICODE(<text>)** | | **Returns a numeric code for the first character in a text string, in the character set used by your computer.** |
| **EXAMPLE : NEWNAME = UNICODE(DimCustomer[FirstName])** | | |
| **CONCATENATE(<text1>, <text2>,…)** | | **The CONCATENATE function joins multiple 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.** |
| **EXAMPLE : NEWNAME = CONCATENATE( DimCustomer[FirstName], DimCustomer[LastName] )** | | |
| **EXACT(<text1>,<text2>)** | | **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.** |
| **EXAMPLE : NEWNAME = EXACT(DimCustomer[FirstName],DimCustomer[LastName])** | | |
| **FIND(<find\_text, within\_text, start\_num)** | | **Returns the starting position of one text string within another text string. FIND is case-sensitive.** |
| **EXAMPLE : NEWNAME = FIND("AND",DimCustomer[LastName],1,100)**  **NEWNAME = FIND("AND",DimCustomer[LastName],11,20)** | | |
| **FIXED(<number>, <decimals>, <no\_commas>)** | | **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.** |
| **Column 2 = FIXED(100,2,TRUE()) = 100.00** | | |
| **LEFT(<text>, <num\_chars>)** | | **Returns the specified number of characters from the start of a text string.** |
| **Column 2 = LEFT("amitsharma",4) = amit** |  | |
| **LEN(<text>)** | | **Returns the number of characters in a text string.** |
| **LOWER(<text>)** | | **Converts all letters in a text string to lowercase.** |
| **MID(<text>, <start\_num>, <num\_chars>)** | | **Returns a string of characters from the middle of a text string, given a starting position and length.** |
| **MID("amitsharma",3,3) = its MID("amitsharma",5,5) = sharm** | | |
| **REPLACE(<old\_text>, <start\_num>, <num\_chars>, <new\_text>)** | | **REPLACE replaces part of a text string, based on the number of characters you specify, with a different text string.** |
| **REPLACE("dillipparhi", 7, 2, "das") = dillipdasrhi** | | |
| **REPT(<text>, <num\_times>)** | | **Repeats text a given number of times. Use REPT to fill a cell with a number of instances of a text string** |
| **RIGHT(<text>, <num\_chars>)** | | **RIGHT returns the last character or characters in a text string, based on the number of characters you specify.** |
| **SEARCH(<search\_text>, <within\_text>, [start\_num])** | | **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.** |
| **SUBSTITUTE(<text>, <old\_text>, <new\_text>, <instance\_num>)** | | **Replaces existing text with new text in a text string.** |
| **TRIM(<text>)** | | **Removes all spaces from text except for single spaces between words.** |
| **UPPER (<text>)** | | **Converts a text string to all uppercase letters** |
| **VALUE(<text>)** | | **Converts a text string that represents a number to a number.** |
| **Column 2 = VALUE("11")=11** | |  |
| **FORMAT(<value>, <format\_string>)** | | **Converts a value to text according to the specified format.** |
| FORMAT ( 0.742, "Percent" )                   -- Returns 74.20%  FORMAT ( 1234.567, "$#,0.00" )                -- Returns $1,234.57  FORMAT ( 1234.567, """US$"" #,0.00" )         -- Returns US$ 1,234.57  FORMAT ( DATE ( 2019, 3, 28 ), "yyyy-mm-dd" ) -- Returns 2019-03-28  FORMAT ( DATE ( 2019, 3, 28 ), "m/d/yy" )     -- Returns 3/28/19  FORMAT(TableName[Duration]/60, "HH:mm:ss") -- | | |

## **Date and Time functions**

|  |  |
| --- | --- |
| **Function** | **Description** |
| **DATE(<year>, <month>, <day>)** | **Returns the specified date in datetime format.** |
| **DD = DATE("1990","01","10")** | |
| **DATEVALUE(date\_text)** | **Converts a date in the form of text to a date in datetime format** |
| **DAY(<date>)** | **Returns the day of the month, a number from 1 to 31.** |
| **EDATE(<start\_date>, <months>)**  **Add months** | **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** |
| **EDATE("01/10/2017",3) = 04/10/2017** | |
| **EOMONTH(<start\_date>, <months>)**  **Add months and give the last day of month** | **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.** |
| **EOMONTH("01/10/2017",3)** | **4/30/2017** |
| **HOUR(<datetime>)** | **Returns the hour as a number from 0 (12:00 A.M.) to 23 (11:00 P.M.).** |
| **MINUTE(<datetime>)** | **Returns the minute as a number from 0 to 59, given a date and time value.** |
| **MONTH(<datetime>)** | **Returns the month as a number from 1 (January) to 12 (December).** |
| **NOW()** | **Returns the current date and time in datetime format.** |
| **SECOND(<time>)** | **Returns the seconds of a time value, as a number from 0 to 59** |
| **TIME(hour, minute, second)** | **Converts hours, minutes, and seconds given as numbers to a time in datetime format.** |
| **TIMEVALUE(time\_text)** | **Converts a time in text format to a time in datetime format.** |
| **TODAY()** | **Returns the current date.** |
| **WEEKDAY(<date>, <return\_type>)** | **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).** |
| **Column 2 = WEEKDAY("10/10/2018",2) = 3** | |
| **WEEKNUM(<date>, <return\_type>)** | **Returns the week number for the given date and year according to the specified convention. The week number indicates where the week falls numerically within a year.** |
| **YEAR(<date>)** | **Returns the year of a date as a four digit integer in the range 1900-9999.** |
| **YEARFRAC(<start\_date>, <end\_date>, <basis>)** | **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.** |
| **YEARFRAC("10/10/2018","10/10/2020",4)=2** | |

Calendar(startdate,enddate)

## **Time Intelligence Functions**

|  |  |
| --- | --- |
| **Function** | **Description** |
| **TotalMTD(<expression>,<dates>,<filter>)**  **QTDTotalsalesthisyear = TOTALMTD(SUM(LocationSales[Profit]),LocationSales[Date])** | **Evaluates the specified expression for the interval that starts at the first day of the month and ends at the latest date in the specified dates column, after applying all filters.** |
| **TotalQTD(<expression>,<dates>,<filter>)**  **QTDTotalsalesthisyear = TOTALQTD(SUM(LocationSales[Profit]),LocationSales[Date])** | **Evaluates the specified expression for the interval that starts at the first day of the quarter and ends at the latest date in the specified dates column, after applying all filters.** |
| **TotalYTD(<expression>,<dates>,<filter>)** | **Evaluates the specified expression for the interval that starts at the first day of the year and ends at the latest date in the specified dates column, after applying all filters.** |
| **Totalsalesthisyear = TOTALYTD(SUM(FactInternetSales[SalesAmount]), DimDate[FullDateAlternateKey],FILTER(DimDate,DimDate[CalendarYear]=2008))**  **MTotalsalesthisyear = TOTALYTD(SUM(LocationSales[Profit]),LocationSales[Date])** | |
| **SAMEPERIOD-LAST YEAR : Return the last year**  PreviousYear = CALCULATE(sum(factTable[Sales]),SAMEPERIODLASTYEAR(factTable[Date].[Date])) | |

## **Time Intelligence Functions**

|  |  |
| --- | --- |
| **Function** | **Description** |
| **CLOSINGBALANCEYEAR(<expression>,<dates>,<filter>)**  **Last day of the year** | **Evaluates the specified expression at the calendar end of the given year. The given year is calculated as the year of the latest date in the dates argument, after applying all filters.**  **Last year from the current date.** |
| **Example :**  **Revenue on the last day of the year**  **closingBalanceYear = CLOSINGBALANCEYEAR(SUM(TransactionData[Revenue]),CalenderDate[Date])**  **LastYear = CLOSINGBALANCEYEAR(COUNT(CANDIDATES[CANDIDATE\_CODE]),CANDIDATES[DateofRegistration])**  **LastYearClosingProfit = CLOSINGBALANCEYEAR(SUM(LocationSales[Profit]),LocationSales[Date])** | |
| **CLOSINGBALANCEQUARTER : Last day of the Quarter**  **CLOSINGBALANCEQUARTER(<expression>,<dates>,<filter>)**  **Last day of the quarter** | |
| **CLOSINGBALANCEMONTH : Last day of the Month**  **CLOSINGBALANCEMONTH(<expression>,<dates>,<filter>)**  **Last day of the month** | |
| **DATESBETWEEN(<column>,<start\_date>,<end\_date>)**  **Calendar = DATESBETWEEN(LocationSales[Date],FIRSTDATE(LocationSales[Date]),LASTDATE(LocationSales[Date]))**  **Returns a table.**  datesdata = DATESBETWEEN(CANDIDATES[DateofRegistration].[Date],FIRSTDATE(CANDIDATES[DateofRegistration].[Date]),LASTDATE(CANDIDATES[DateofRegistration].[Date]))  **returns all dates between the starting date to the ending date** | **Returns a table of dates that can be found in the specified date column beginning with the start date and ending with the end date.** |
| **DATEADD(<date\_column>,<number\_of\_intervals>,<interval>)** | **Returns a table that contains a column of dates, shifted either forward in time or back in time from the dates in the specified date column.** |
| **FIRSTDATE (<datecolumn>)** | **Returns the first date in the current context for the specified Date\_Column.**  firstdate = FIRSTDATE(CANDIDATES[DateofRegistration].[Date])  **“01/01”2016”** |
| **FIRSTDATE : WILL GIVE THE FIRST DATE DEPENDING UPON CONTEXT , FILTERS**  **LASTDATE : WILL GIVE THE END DATE DEPENDING UPON CONTEXT , FILTERS**  **DATESBETWEEN : WILL RETURN SET OF DATES BETWEEN START RANGE AND END RANGE**  **STARTOFYEAR,STARTOFQUARTER,STARTOFMONTH :**  **WILL GIVE THE STARTYEARDATE,STARTQUATERDATE AND STARTMONTH DATE OF PARTICULAR YEAR,QUARTER AND MONTH**  **ENDOFYEAR, ENDOFQUARTER,ENDOFMONTH :**  **WILL GIVE THE ENDYEARDATE,ENDQUATERDATE AND ENDMONTH DATE OF PARTICULAR YEAR,QUARTER AND MONTH** | |
| **Column = FIRSTDATE (CANDIDATES[DOB]) = “01/01/2015”**  **Returns the first date of the month, year depending upon filter selection. For a year it will the first date of the year.** | |
| **LASTDATE (<datecolumn>)** | **Returns the last date in the current context for the specified Date\_Column.** |
| **Column = LASTDATE (CANDIDATES[DOB])** | |
| **FIRSTNONBLANK (<datecolumn>,<expression>)** | **Returns the firstdate from datatable**  CALCULATE(FIRSTNONBLANK(CANDIDATES[DateofRegistration],TRUE())) |
| **LASTNONBLANK (<datecolumn>,<expression>)** | **Returns the lastdate from datatable**  LastDateofRegistration = CALCULATE(LASTNONBLANK(CANDIDATES[DateofRegistration],TRUE())) |
| **STARTOFMONTH (<date\_column>)** | **Returns the first day of the month in the specified date column.** |
| **Column = STARTOFMONTH(CANDIDATES[DOB].[Date])** |  |
| **STARTOFQUARTER (<date\_column>)** | **Returns the first day of the quarter in the specified date column.** |
| **STARTOFQUARTER (CANDIDATES[DOB].[Date])** |  |
| **STARTOFYEAR(<date\_column>)** | **Returns the first day of the year in the specified date column.** |
| **Column = STARTOFYEAR(CANDIDATES[DOB].[Date])** |  |
| **ENDOFMONTH(<date\_column>)** | **Returns the last day of the month in the specified date column.** |
| **Column = ENDOFMONTH(CANDIDATES[DOB].[Date])** |  |
| **ENDOFQUARTER(<date\_column>)** | **Returns the last day of the quarter in the specified date column.** |
| **ENDOFYEAR(<date\_column>)** | **Returns the last day of the year in the specified date column.** |
| DATESYTD ( <Dates>[, <StartDate>] ) | Returns a set of dates in the year up to the last date visible in the filter context. |
| Measure 2 = CALCULATE ( [Total], DATESYTD ( 'Calendar'[Date] ) )  YearCount = CALCULATE ( COUNT(CANDIDATES[CANDIDATE\_CODE]), DATESYTD (CANDIDATES[DateofRegistration].[Date] ) )  TableYear = DATESYTD(CANDIDATES[DateofRegistration].[Date])  **TableYear = DATESYTD(CANDIDATES[DateofRegistration].[Date],"05/15/2020")** | |
| DATESMTD ( <Dates> )  Returns a set of dates in the month up to the last date visible in the filter context.  MonthYear = DATESMTD(CANDIDATES[DateofRegistration].[Date]) | |

|  |
| --- |
| PARALLELPERIOD 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.  PARALLELPERIOD(<dates>,<number\_of\_intervals>,<interval>)   * 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 or month. |

|  |
| --- |
| StudentMonthParallelCount = CALCULATE(COUNT(CANDIDATES[CANDIDATE\_CODE]),PARALLELPERIOD(CANDIDATES[DateofRegistration].[Date],-1,MONTH))  StudentYearParallelCount=  CALCULATE(COUNT(CANDIDATES[CANDIDATE\_CODE]),PARALLELPERIOD(CANDIDATES[DateofRegistration].[Date],-1,YEAR))  LastYearStudentCount = CALCULATE(COUNT(CANDIDATES[CANDIDATE\_CODE]),SAMEPERIODLASTYEAR(CANDIDATES[DateofRegistration].[Date])) |
| DATEADD = CALCULATE([Total Sales], DATEADD(‘Calendar’ [Date], -1, YEAR))  SAMEPERIODLASTYEAR = CALCULATE([Total Sales], SAMEPERIODLASTYEAR(‘Calendar’ [Date]))  https://databear.com/power-bi-dax-sameperiodlastyear-paralellperiod-and-dateadd/ |

CALENDAR

|  |
| --- |
| The following formula returns a table with dates between January 1st, 2005 and December 31st, 2015.  DAX  = CALENDAR (DATE (2005, 1, 1), DATE (2015, 12, 31))  For a data model which includes actual sales data and future sales forecasts. The following expression returns the date table covering the range of dates in these two tables.  DAX  = CALENDAR (MINX (Sales, [Date]), MAXX (Forecast, [Date])) |

# **EXAMPLES OF TABLE FUNCTIONS IN DAX / POWER BI**

DataTable Function

Syntax

=DataTable("ColumName", DataType,

"ColumnName", DataType

,{

{"col1value","col2value” },

{" col1value "," col2value "}

}

)

=DataTable("Name", STRING,

"Region", STRING

,{

{" User1","East"},

{" User1","East"},

{" User2","East"},

{" User3","West"},

{" User4","West"},

{" User4","East"}

}

)

UNION

COMIBINES THE ROWS OF TWO OR MORE TABLES

UNION(UsaInventory, IndInventory)

FinalProductsData = UNION(Product1,Product2)

tblUnionResults = UNION(Country1,Country2)

**EXCEPT (SIMILAR TO MINUS )**

TABLE1 MINUS TABLE2

|  |
| --- |
| **tblExceptResults = EXCEPT(Country1,Country2)** |

Except(States1, States2)

Except(States1, States2)

**INTERSECT**

COMMON RECORDS IN TWO TABLES

INTERSECT(table1,table2)

**tblIntersectResults = INTERSECT(Country1,Country2)**

**NATURALINNERJOIN**

tblJoins = NATURALINNERJOIN(DEPTS,Employee)

tblJoins = NATURALLEFTOUTERJOIN(DEPTS,Employee)

**EXAMPLES OF OR , IN & CONTAINROWS**

And, using the calculate function, the goal has been achieved…

* For example, consider the following syntax:

|  |
| --- |
| RedOrBlack Sales OR :=CALCULATE ([Sales Amount],  Products[Color] = "Red" || Products[Color] = "Black" ) |

* With the new IN syntax, you can write:

|  |
| --- |
| RedOrBlack Sales IN :=  CALCULATE ( [Sales Amount], Products[Color] IN { "Red", "Black" }  ) |

WEEKEND SALES

|  |
| --- |
| WeekDaySales = CALCULATE(sum(factTable[Sales]),WEEKDAY(factTable[Date]) in{1,2,3,4,5}) |

WEEKDAYS SALES

|  |
| --- |
| WeekEndSales = CALCULATE(SUM(factTable[Sales]),WEEKDAY(factTable[Date]) in{6,7}) |

SWITCH – CASE IN POWER BI

|  |
| --- |
| SWITCH  (  COLNAME,  WHENVALUE1,CALVALUE1,  WHENVALUE2,CALVALUE2  ) |
| ConvertedIndianPrice =  SWITCH(  'Product'[CURRENCY],"DOLLAR",'Product'[UNIT PRICE] \* 72,  "POUND",'Product'[UNIT PRICE] \* 90,  "RUPPEE",'Product'[UNIT PRICE] \* 1,  "RAND",'Product'[UNIT PRICE] \* 7  ) |

DYNAMIC FORMATTING IN POWER-BI

|  |  |
| --- | --- |
| |  | | --- | | This would be the measure I would write:  Measure = if(HASONEVALUE(Device[DeviceName]), SWITCH(VALUES(Device[DeviceName]) ,”Aircard 1″, FORMAT([Sum of Revenue], “0.0”) ,”Aircard 2″, FORMAT([Sum of Revenue], “0.00”) ,”Aircard 3″, FORMAT([Sum of Revenue], “$0.000″) ,”Aircard 4”, FORMAT([Sum of Revenue], “0.0000”) ,FORMAT([Sum of Revenue], “0”) ) ) | |

## Iterator Functions in DAX

Iterators in DAX are functions that iterate through all rows of the given table, apply the expression, and then aggregate the result. One of the known functions in this category is [SUMX](https://radacad.com/sum-vs-sumx-what-is-the-difference-of-the-two-dax-functions-in-power-bi). because an iterator function goes through every row and apply the transformation, then the result is materialized (temporary) before aggregating it, and that causes the totals to be actual sum (or other aggregations) of values in a column.

BudgetProfit = CALCULATE(SUMX(factTable,factTable[Budget Profit]))

# SUM() vs SUMX()

First let me start with an overview of both of these functions.

* SUM() is an aggregator function.  It adds up all the values in a column you specify **in the current filter context.**SUM() has no knowledge of the existence of rows – all it can do is add everything in the single column it is presented with.
* SUMX() is an iterator function.  It works through a table, row by row to complete the evaluation **in the current filter context.**SUMX() has awareness of rows in a table, and hence can reference the intersection of each row with any columns in the table.

|  |
| --- |
| * SUMX (<table>, <expression>)   TotalSales = CALCULATE(Sumx(Sales,[Price]\*[Quantity]))  **Add a Profit Column to Product Table in CoffeeShop Data**  **Profit = CALCULATE(SUMX(factTable,factTable[Profit]))**  **Profit = CALCULATE(SUMX(factTable,factTable[Profit]))**  BudgetProfit = CALCULATE(SUMX(factTable,factTable[Budget Profit]))  Status = IF((('Product'[Profit] > 'Product'[BudgetProfit]) && ('Product'[Profit] > 20000)),"HighProfitProduct","LowProfitProduct")  Status = IF((('Product'[Profit] > 'Product'[BudgetProfit]) && ('Product'[Profit] > 20000)),"HighProfitProduct","LowProfitProduct") |

**SUMMARIZE**

Returns a summary table for the requested totals over a set of groups.

**Syntax**

DAX

**SUMMARIZE(<table>, <groupBy\_columnName>[, <groupBy\_columnName>]…[, <name>, <expression>]…)**

**Parameters**

|  |
| --- |
| SummResults = SUMMARIZE(factTable,'Product'[Product],"SalesAmount" ,SUM(factTable[Sales]),"Profit",sum(factTable[Profit])) |
| DAX  SUMMARIZE(ResellerSales\_USD  , DateTime[CalendarYear]  , ProductCategory[ProductCategoryName]  , "Sales Amount (USD)", SUM(ResellerSales\_USD[SalesAmount\_USD])  , "Discount Amount (USD)", SUM(ResellerSales\_USD[DiscountAmount])  ) |
| SalesTerritory\_SummResults =  SUMMARIZE  (  FactInternetSales,  DimSalesTerritory[SalesTerritoryCountry],  DimDate[CalendarYear],  "SalesTotal",SUM(FactInternetSales[SalesAmount]),  "Freight Total",SUM(FactInternetSales[Freight])  ) |

## Advanced SUMMARIZE options

### SUMMARIZE with ROLLUP

The addition of the ROLLUP() syntax modifies the behavior of the SUMMARIZE function by adding roll-up rows to the result on the groupBy\_columnName columns.

DAX

SUMMARIZE(<table>, <groupBy\_columnName>[, <groupBy\_columnName>]…[, ROLLUP(<groupBy\_columnName>[,< groupBy\_columnName>…])][, <name>, <expression>]…)

|  |
| --- |
| Summ\_WithRollUp = SUMMARIZE(FactInternetSales,ROLLUP(DimSalesTerritory[SalesTerritoryCountry],DimDate[CalendarYear]),"TotalSales",SUM(FactInternetSales[SalesAmount]),"TotalFreight",sum(FactInternetSales[Freight])) |

## **RANKX DAX function in Power BI**

The Power BI DAX RANKX function is a powerful sortation function. We will give an overview of what the RANKX function is capable of and make sure that you understand the basics. This is a super useful function so we will also explore some more complex methods of using RANX with practical examples.

RANKX(<table>, <expression>, <value>, <order>, <ties>)

|  |
| --- |
| RankId = RANKX('Product','Product'[ConvertedIndianPrice],,ASC) |

RANX Total Sales =

RANKX(

‘Sales Table’,

‘Sales Table'[Total Sales]

)

|  |
| --- |
| RankColOrder = RANKX(    Country,  [TotalSales]  ) |

|  |
| --- |
| RANX Total Sales ASC =  RANKX(  ‘Sales Table’,  ‘Sales Table'[Total Sales],  ,  ASC  ) |

**Note here that there will be a ‘gap’ in the rank values in RANKX’s default state or if we pass SKIP into the ties parameter. For this example, the next rank value will be 5, as our rank values are 1, 2, 3, 3, 5.**

**If we pass DENSE in the ties parameter, it will not allow for gaps in the value of RANKX, meaning that in our used example the next rank value after our tied RANKX value of 3, will be 4.**

**RANK.EQ**

Returns the rank of a number in a column of numbers. If more than one value has the same rank, the top rank of that set of values is returned.

## Remarks

The following RANK.EQ calls:

|  |
| --- |
| RANK.EQ ( <value>, table[column] )   RANK.EQ ( <value>, table[column], <order> ) |
| |  | | --- | | RankValue = RANK.EQ(44,'Product'[UNIT PRICE],DESC) | |

|  |
| --- |
| example: An order quantity for 174,728 units is received. Find the rank of this order in terms of quantity compare to all other order quantities. Answer Deinfe the following measures in Power BI:  rank\_it = RANK.EQ(174728,TransactionData[Quantity]) |

**TOPN syntax**

TOPN(<n\_value>, <table>, <orderBy\_expression>, [<order>[, <orderBy\_expression>, [<order>]]…])

|  |
| --- |
| Top3Products = TOPN(3,VALUES('Product'[Product]),CALCULATE(SUM('Product'[ConvertedIndianPrice])),DESC) |

TopNProducts = TOPN (10,VALUES(DimProduct[EnglishProductName]),CALCULATE (SUM(FactInternetSales[SalesAmount])))

TopNSubCategory = TOPN(3,VALUES(DimProductSubcategory[EnglishProductSubcategoryName]),CALCULATE(SUM(FactInternetSales[SalesAmount])))

**RETURN**

The RETURN keyword consumes variables defined in previous VAR statements.

|  |
| --- |
| VAR <name> = <expression>  [VAR <name2> = <expression2> [...]]  RETURN <result\_expression> |

|  |
| --- |
| SalesIncrease =  var sales2019 = TOTALYTD(SUM(factTable[Sales]),factTable[Date].[Date])  var sales2018 = CALCULATE(SUM(factTable[Sales]),PARALLELPERIOD(factTable[Date].[Date],-1,YEAR))  return (sales2019-sales2018) |

|  |
| --- |
| GrowthPer =  VAR  TotalStudents = COUNT( CANDIDATES[CANDIDATE\_CODE] )  VAR  TotalStudentsLastYear2019 = CALCULATE (  COUNT ( CANDIDATES[CANDIDATE\_CODE] ),  SAMEPERIODLASTYEAR ( CANDIDATES[DateofRegistration].[Date])  )  VAR  TotalStudents2020 =  TOTALYTD(COUNT ( CANDIDATES[CANDIDATE\_CODE] ),  CANDIDATES[DateofRegistration].[Date])  RETURN  DIVIDE (  TotalStudents2020,  TotalStudentsLastYear2019  ) |

### COUNTX By Calculated Column

Now let’s look at COUNTX. Remember we said COUNTX is an iterator.  That means it will work its way through the table and evaluates an expression for each row. it will then store that value and aggregation of these values will happen at the end.

NegProfitCount = COUNTX(FILTER(factTable,factTable[Profit] < 0),factTable[Profit])

What we will try to do next is count the number of a set of values in the products table. But we will filter the table for the product category Shoes.

First, we will create a calculated column and we will enter the following DAX function:

CCcountshoes =COUNTX (FILTER(products,products[Product Name]=”Shoes”),products[Cost Price])

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| DAX  COUNTX(<table>,<expression>) Parameters  | Parameters | | | --- | --- | | **Term** | **Definition** | | table | The table containing the rows to be counted. | | expression | An expression that returns the set of values that contains the values you want to count. |  Return value An integer. Remarks The COUNTX function takes two arguments. The first argument must always be a table, or any expression that returns a table. The second argument is the column or expression that is searched by COUNTX.  **The COUNTX function counts only values, dates, or strings. If the function finds no rows to count, it returns a blank.**  **If you want to count logical values, use the COUNTAX function.** Example 1 The following formula returns a count of all rows in the Product table that have a list price.  DAX  = COUNTX(Product,[ListPrice]) Example 2 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.  = COUNTX(FILTER(Product,RELATED(ProductSubcategory[EnglishProductSubcategoryName])="Caps", Product[ListPrice]) |