# Calculated column vs Measure

## Option 1

When you insert Columns or Measures using the **Modeling** tab, they are assigned to whichever *table is currently selected*, or the *first table in the field list* by default.

* Measures can be reassigned to new “Home” tables (under the “*Properties*” options in the **Modeling** tab), but the Option 2 allows you to be more deliberate about placing them

## Option 2

* Right-click within the table (in the Data view) or the Field List (in either the Data or Report view)

## Option 3

* New quick measure
  + Quick measures are pre-built formula templates that allow you to drag and drop fields, rather than write DAX from scratch
* Note: Assigning measures to specific tables doesn’t have any impact on functionality – it’s just a way to keep them organized.

# Implicit vs Explicit Measures

## Implicit Measures

Only accessible within the specific visualization in which it was created, and cannot be referenced elsewhere

## Explicit Measures

Can be used anywhere in the report and referenced within other DAX calculations to create “measure trees”

Each measure value in a report is ***like an island***, and calculates according to its own filter context (even *Totals* and *Grand Totals*)

Measure Name = function name ( table name [ column name ] )

Calculated columns do not always use functions, but measures always use functions

* In a Calculated Column, **=Transactions[quantity]** returns the value from the quantity column in each row (since it evaluates one row at a time)
* In a Measure, **=Transactions[quantity]** will return an error since PowerBI doesn’t know how to translate that as a single value (you need some sort of aggregation)

For Column references, use the fully qualified name (i.e. Table[Column])

For measure references, use the measure name (i.e. [Measure])

*Referenced*

*Table Name*

**Total Quantity**: =SUM(*Transactions*[***quantity***])

**Measure Name** Function Name ***Referenced Column name***

# RELATED()

Returns related values in each row of a table based on relationships with other tables

=RELATED(ColumnName)

e.g. ColumnName

Product\_Lookup(ProductName)

Territory\_Lookup(Country)

RELATED works almost exactly like a VLOOKUP function – it uses the relationship between tables (defined by primary key and foreign keys) to pull values from one table into a new column of another

Since this function requires row context, it can only be used as a Calculated Column or as part of an iterator function that cycles through all the rows in a table (FILTER, SUMX, MAXX, etc.)

Avoid using RELATED to create redundant calculated columns unless you absolutely need them, since those extra columns increase file size. Instead, use RELATED within a measure like FILTER or SUMX.

COUNT() – Counts the number of cells in a column that contains numbers

COUNTA() – Counts the number of non-empty cells in a column (numerical and non-numerical)

DISTINCTCOUNT() – Counts the number of distinct or unique values in a column

COUNTROWS() – Counts the number of rows in the specified table or a table defined by an expression

# CALCULATE()

=CALCULATE(Expression,[Filter1],[Filter2],..)

Expression should contain name of an existing measure or a DAX formula for a valid measure

e.g.

[Total Orders]

SUM(Returns\_Data[ReturnQuantity])

[Filter1][Filter2]…

List of simple Boolean (True/False) filter expressions

(note: these require simple, fixed values; you cannot create filters based on measures or reference measures)

e.g.

Territory\_Lookup[Country] = “USA”

Calendar[Year] > 1988

CALCULATE works just like SUMIF or COUNTIF in excel, except it can evaluate measures based on ANY sort of calculation (not just a sum, count, etc.); it may help to think of it like **CALCULATEIF**

CALCULATE ***modifies*** and ***overrules*** any competing filter contexts

Step1

Filter context is detected and applied

Step 1.5 – Filters modified by CALCULATE

Step 2 – Filters flow “downstream” to all related tables

Step 3 – Measure formula evaluates against the filtered table

# ALL()

Returns all rows in a table or all values in a column, ignoring any filters that have been applied

Does not return a single number, returns a table or a subset of a table. Almost never used by itself, but is used as a component of other functions, most commonly CALCULATE.

=ALL(Table or ColumnName, [ColumnName1],[ColumnName2],…)

The table or column that you want to clear filters on

e.g.

Transactions

Products[ProductCategory]

[ColumnName1],[ColumnName2],…

List of columns that you want to clear filters on (optional)

**Notes**: If your first parameter is a table, you can’t specify additional columns. All columns must include the table name, and come from the same table.

e.g.

Customer\_Lookup[CustomerCity], Customer\_Lookup[CustomerCountry)

Products[ProductName]

Instead of adding filter context, ALL **removes it**. This is often used when you need unfiltered values that won’t react to changes in filter context (e.g. **% of Total** where the denominator needs to remain fixed)

# FILTER()

=FILTER(Table, FilterExpression)

Table – Table to be filtered

e.g.

Territory\_Lookup

Customer\_Lookup

FilterExpression – A Boolean (True/False) filter expression to be evaluated for each row of the table

e.g.

Territory\_Lookup[Country]=”USA”

Calendar[Year]=1988

Product\_Lookup[ProductPrice]>[Overall Avg Price]

FILTER is used to add new filter context and can handle ***more complex filter expressions*** than CALCULATE (by referencing measures, for example). Since FILTER returns an entire table, it is almost always used as an input to other functions like CALCULATE or SUMX

Since FILTER iterates through each row in a table, it can be slow and processor-intensive; don’t use FILTER if a CALCULATE function will accomplish the same thing

# Iterator (“X”) Functions

=SUMX(Table,Expression)

SUMX

Aggregation to apply to calculated rows

e.g.

SUMX

MAXX

COUNTX

AVERAGEX

RANKX

MINX

Table – Table in which the expression will be evaluated

e.g.

Sales

FILTER(Sales,RELATED(Products[Category])=”Clothing”)

Expression – To be evaluated for each row of the given table

e.g.

[Total Orders]

Sales[RetailPrice]\*Sales[Quantity]

Imagine the function ***adding a temporary new column*** to the table, calculating the value in each row (based on the expression) and then applying the aggregation to that new column (like SUMPRODUCT)

# Time Intelligence Formulas

Performance To-Date = CALCULATE(Measure, DATESYTD(Calendar[Date]))

Use DATESQTD for quarters or DATESMTD for months

Previous Period = CALCULATE(Measure, DATEADD(Calendar[Date],-1,MONTH))

, -1, MONTH)

Select an interval (DAY, MONTH, QUARTER, or YEAR) and the # of intervals to compare (i.e. previous month, rolling 10-day)

Running Total = CALCULATE(Measure, DATESINPERIOD(Calendar[Date], MAX(Calendar[Date]), -10, DAY))

, -10, DAY)

Select an interval (DAY, MONTH, QUARTER, or YEAR) and the # of intervals to compare (i.e. previous month, rolling 10-day)

To calculate a ***moving average***, use the running total calculation above and divide by the number of intervals

# What-If Parameters

**“What-if” Parameters** are essentially pre-set measures that produce values within a given range, based on user-inputs (data-type, min/max, increment, and default).

These can be great tools for forecasting or scenario testing; here we’ve created a “**Price Adjustment %**” parameter in order to compare **Total Revenue** (based on the actual price) against **Adjusted Revenue** (based on the *parameter-adjusted* price).

NOTE: When you create a parameter, a new table is automatically added with DAX calculations for ***“Parameter” and “Parameter Value”***, which look something like this:

Parameter = GENERATESERIES(-1, 1, 0.1)

Parameter Value = SELECTEDVALUE(Parameter[Parameter], 0)