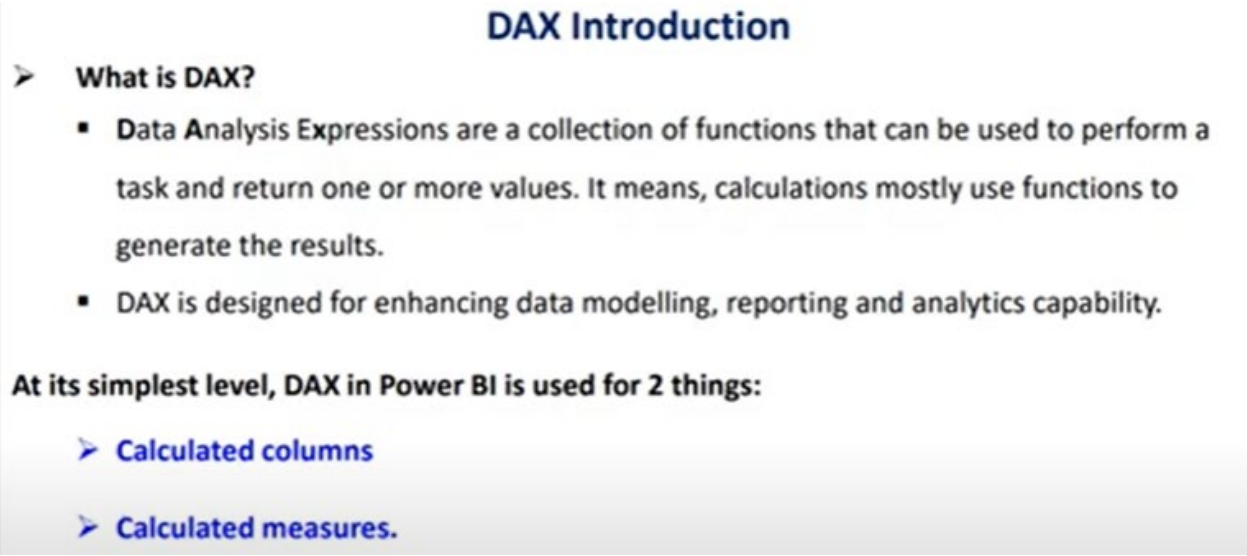
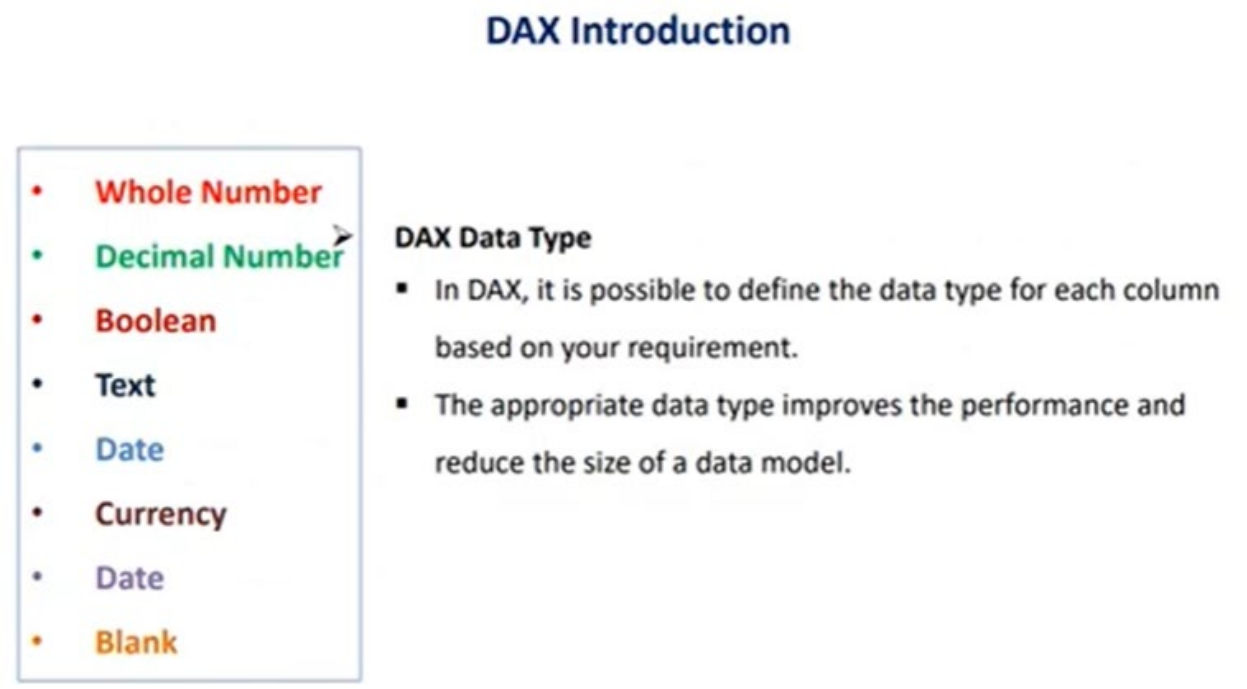
Microsoft DAX Documentation with examples: https://learn.microsoft.com/en-us/dax/

1] DAX - Introduction





DAX Function Category

There are more than 200 DAX functions. There are 9 categories in the DAX function.

- **Aggregation functions**

  Aggregation functions return a scalar value applying an aggregation function to a column or to an expression evaluated by iterating a table expression.

- **Date and Time functions**
  - Date and time functions help creating calculations based on dates and time.
- **Filter functions**
  - Filter functions manipulate table and filter contexts.
- **Information functions**
  - Information functions provide information about data type or filter context of the argument provided.
- **Logical functions**
  - Logical functions act upon an expression to return information about the values or sets in the expression.

DAX Function Category

- **Math and Trig functions**
  - The mathematical functions in DAX are very similar to the Excel mathematical and trigonometric functions.
- **Parent-child functions**
  - These functions helps flattening a parent-child relationship in a regular one.
- **Relationships management functions**
  - These functions manage and manipulate relationships between tables.
- **Table manipulation functions**
  - These functions manipulate and return tables.
- **Text functions**
  - Text functions manipulate strings.
- **Time Intelligence functions**
  - Time intelligence functions support calculations to compare and aggregate data over time periods, supporting days, months, quarters, and years.

DAX Introduction

- Data
- Tables
- Columns
- Relationships
- Measures
- Hierarchies

➢ **Components of DAX Data Model**

- A data model consists of data, calculations and formatting rules and it combines to create an object.
- This object helps to explore and understand the dataset.

DAX Parameter Names

➢ DAX has standard parameter names to facilitate the usage and understanding of the DAX functions.

➢ To understand the **syntax** of the DAX functions and to use data values appropriately for the relevant DAX function parameters, It is important to understand DAX parameter naming conventions.

DAX Parameter Names

Sr. No.	Parameter Name & Description
1	**Expression** : Any DAX expression that returns a single scalar value, where the expression is to be evaluated multiple times (for each row/context).
2	**Value** : Any DAX expression that returns a single scalar value where the expression is to be evaluated exactly once before all other operations.
3	**Table** : Any DAX expression that returns a table of data.
4	**tableName**: The name of an existing table using standard DAX syntax. It cannot be an expression.
5	**columnName**: The name of an existing column using standard DAX syntax, usually fully qualified. It cannot be an expression.
6	**Name** : A string constant that will be used to provide the name of a new object.
7	**Order** : An enumeration used to determine the sort order.
8	**Type** : An enumeration used to determine the data type for Path Item and PathItemReverse.

DAX Functions - Description Structure

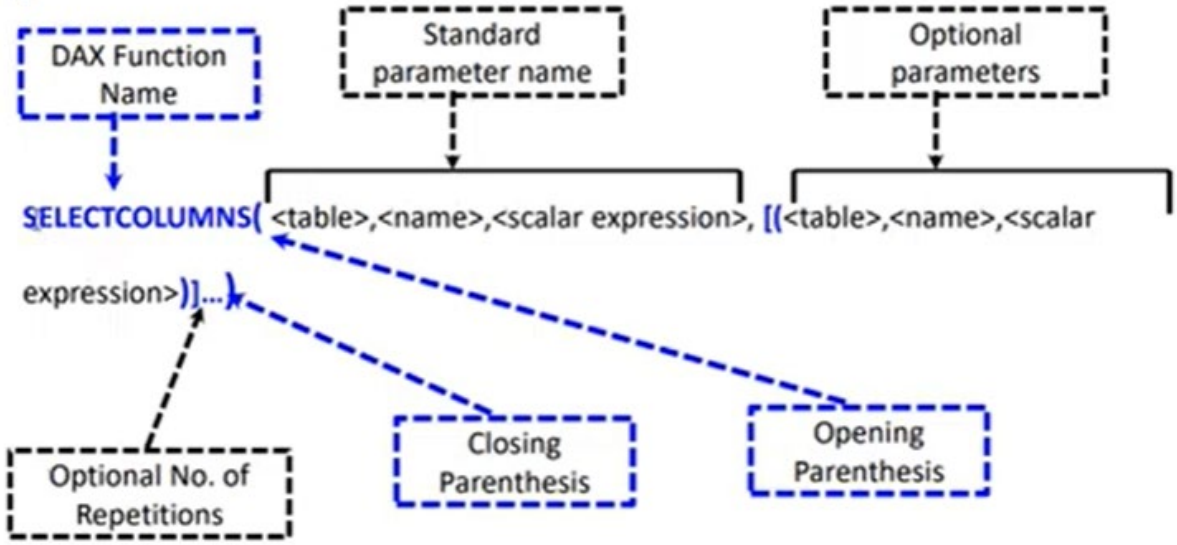
What is the DAX function and how and where to use it.

- **Syntax**
  - 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.**

DAX Functions - Description Structure

What is the DAX function and how and where to use it.

- **Syntax**



DAX Context

The concept of DAX context is essential in gaining a complete understanding of DAX language and how it works in Power BI.

There are two types of DAX contexts; Row context and Filter context.

Mainly there are two types of context in DAX.

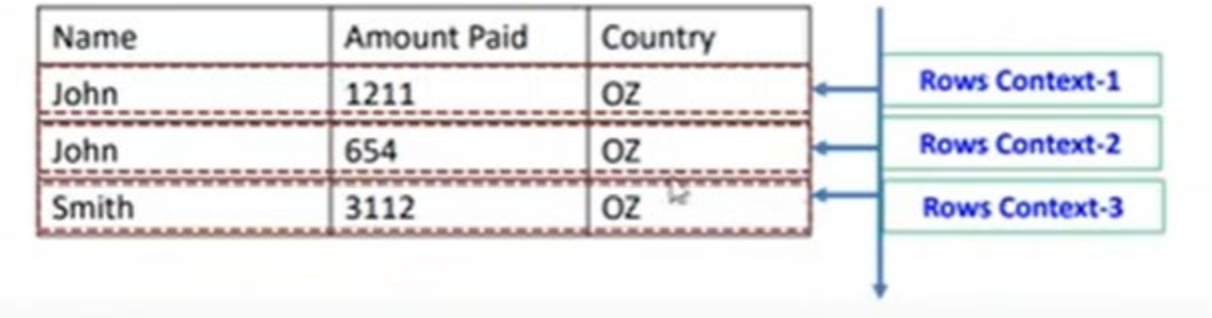
- **Row context**
  - Row context can be thought of as "the current row."
  - The row context consists of the values in each individual row and values in columns that are related to the current row.
- **Filter context**
  - The filter context filters data.
  - It is the set of values allowed in each column, based on filter constraints that were applied to the row or that are defined by filter expressions within the formula.

DAX Context

Name	Amount Paid	Country
John	1211	OZ
Taylor	2433	NZ
John	654	OZ
Taylor	678	NZ
Marina	896	UK
Smith	3112	OZ

Name	Amount Paid	Country
John	1211	OZ
Taylor	2433	NZ
John	654	OZ
Taylor	678	NZ
Marina	896	UK
Smith	3112	OZ

DAX Context



2] DAX - Calculate Function

Power BI DAX CALCULATE Function

- Evaluates an expression in a context modified by filters.
  - **Syntax:**

CALCULATE(<expression>[, <filter1> [, <filter2> [, ...]]])

[Mandatory]
Return the result(value format).
For this reason, the expression is usually an aggregation function like SUM, MIN, MAX, COUNTROWS

[Optional]
Optional arguments are **Filter1**, **Filter2**, **Filter3,...**, **Filter(n).** Up to as many filter conditions as you might need.

So what does the CALCULATE function do exactly?

The CALCULATE function is useful because it can **modify** the *filter context* of the Expression inside:

- If the ***filter context*** specified by a filter condition already exists, it will *override* the already-existing ***filter context*** with the new one that's been specified in the CALCULATE expression.
- If the ***filter context*** does not exist, it will add new filter context according to the filter condition specified.
- Every filter argument can be either a filter removal (such as **ALL, ALLEXCEPT, ALLNOBLANKROW**), a filter restore (**ALLSELECTED**), or a table expression returning a list of values for one or more columns or for an entire expanded table.

Power BI DAX CALCULATE Function

Step-1

Measure_Product_DistinctCount =

DISTINCTCOUNT('Sales Summary'[Product])

Country	Gross Sales	Measure_Product_DistinctCount
Canada	142782	6
France	26686	5
Germany	23235	6
Mexico	13927	5
United States of America	31542	4
Total	**238172**	**7**

1. **Pen**
2. **Pencil**
3. **Mouse**
4. **Pad**
5. **Hard Disk**
6. **Keyboard**
7. **CPU**

26

You can see that the total value is not the sum of all values, right? That is because the total is evaluated in a different filter context. Which filter context? The whole table!

Power BI DAX CALCULATE Function

```
Measure_TotalSales =
        CALCULATE(Sum('Sales Summary'[Gross Sales]))

Measure_TotalSales_Pen =
        CALCULATE(SUM('Sales Summary'[Gross Sales]),
        'Sales Summary'[Product]="Pen")
```

Country	Measure_TotalSales	Measure_TotalSales_Pen
Canada	142782	5700
France	26686	10310
Germany	23235	2609
Mexico	13927	156
United States of America	31542	
Total	**238172**	**18775**

Power BI DAX CALCULATE Function

Step-3

- **Filter modifier functions**

Filter modification functions allow you to do more than simply add filters. They provide you with additional control when modifying filter context.

- ALL
- ALLEXCEPT

Remove filters from one or more columns, or from all columns of a single table.

Power BI DAX CALCULATE Function

Step-4

ALL: Remove filter from table

```
Measure_Pen_Sales =
   CALCULATE(Sum('Sales Summary'[Gross Sales]),
                    All('Sales Summary'), 'Sales Summary'[Product]="Pen"
                    )
```

Country	Gross Sales	Measure_Pen_Sales
Canada	142782	18775
France	26686	18775
Germany	23235	18775
Mexico	13927	18775
United States of America	31542	18775
Total	**238172**	**18775**

Power BI DAX CALCULATE Function

➢ **ALLEXCEPT:**

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

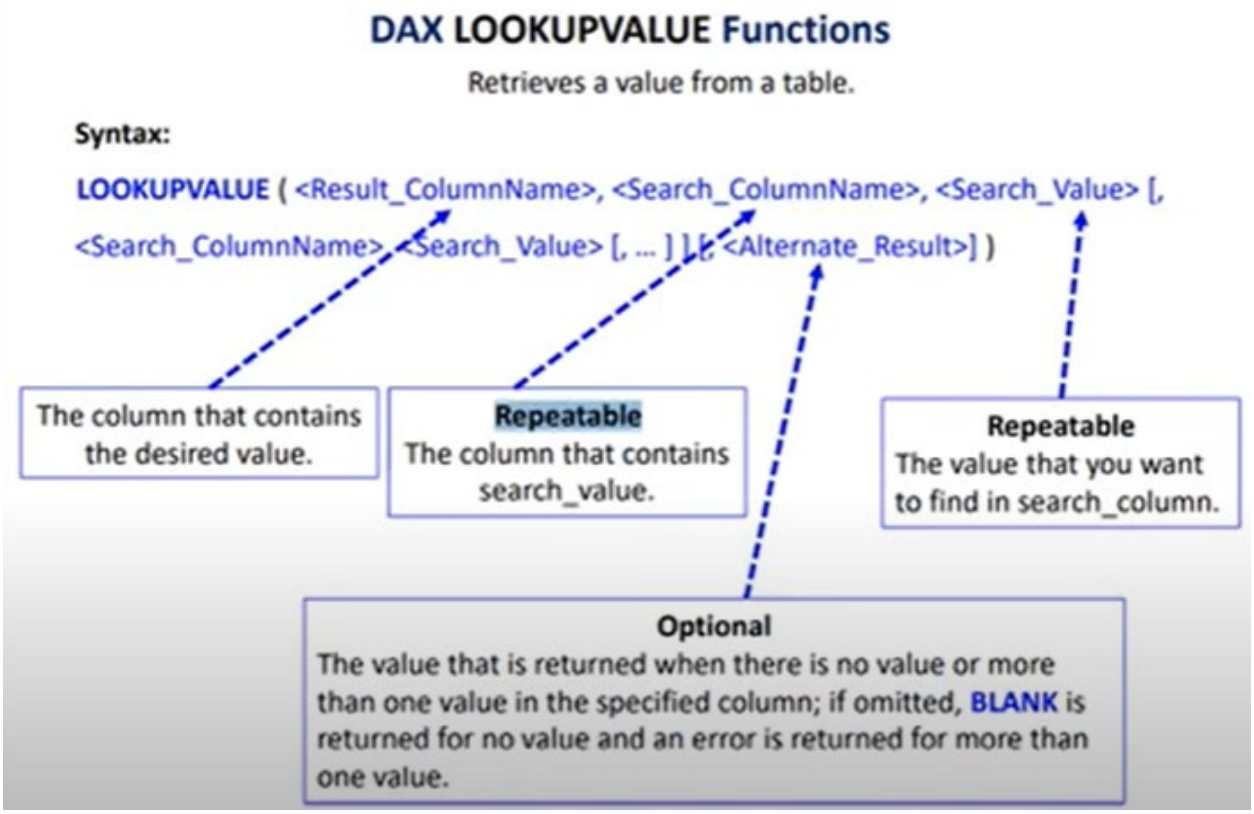
```
Measure_Pen_Sales =
CALCULATE (Sum ('Sales Summary'[Gross Sales]),
ALLEXCEPT ('Sales Summary','Sales Summary'[Country]),'Sales Summary'[Product]="Pen"
)
```

Country	Gross Sales	Measure_Total_Product_Sold	Measure_Pen_Sales
Canada	142782	6	5700
France	26686	5	10310
Germany	23235	6	2609
Mexico	13927	5	156
United States of America	31542	4	
Total	**238172**	**7**	**18775**

```
1] TotalSales_pen = CALCULATE(SUM('Sales Summary'[Gross Sale]), 'Sales Summary'[Product] = "Pen")
```

```
2] TotalSales_pen_1 = CALCULATE(SUM('Sales Summary'[Gross
Sale]),ALL('Sales Summary'), 'Sales Summary'[Product] = "Pen")
3] TotalSales_pen_2 = CALCULATE(SUM('Sales Summary'[Gross
Sale]),ALL('Sales Summary'[Country]),'Sales Summary'[Product] = "Pen")
4] TotalSales_pen_3 = CALCULATE(SUM('Sales Summary'[Gross
Sale]),ALLEXCEPT('Sales Summary','Sales Summary'[Country]), 'Sales
Summary'[Product] = "Pen")
5] TotalSales_pen_4 = CALCULATE(SUM('Sales Summary'[Gross
Sale]),ALLEXCEPT('Sales Summary','Sales Summary'[Product]), 'Sales
Summary'[Product] = "Pen")
```

3] DAX - LookUpValue



DAX LOOKUPVALUE Functions

➢ **Example:**

```
DeptName = LOOKUPVALUE(Depts[Dept_Name], Depts[Dep_ID], Employee[DeptID], "Default")
```

Depts Table

Dep_ID	Dept_Name
1	Sales
2	Purchase
3	Accounts
4	IT
5	HR
6	Developmemt
7	Admin

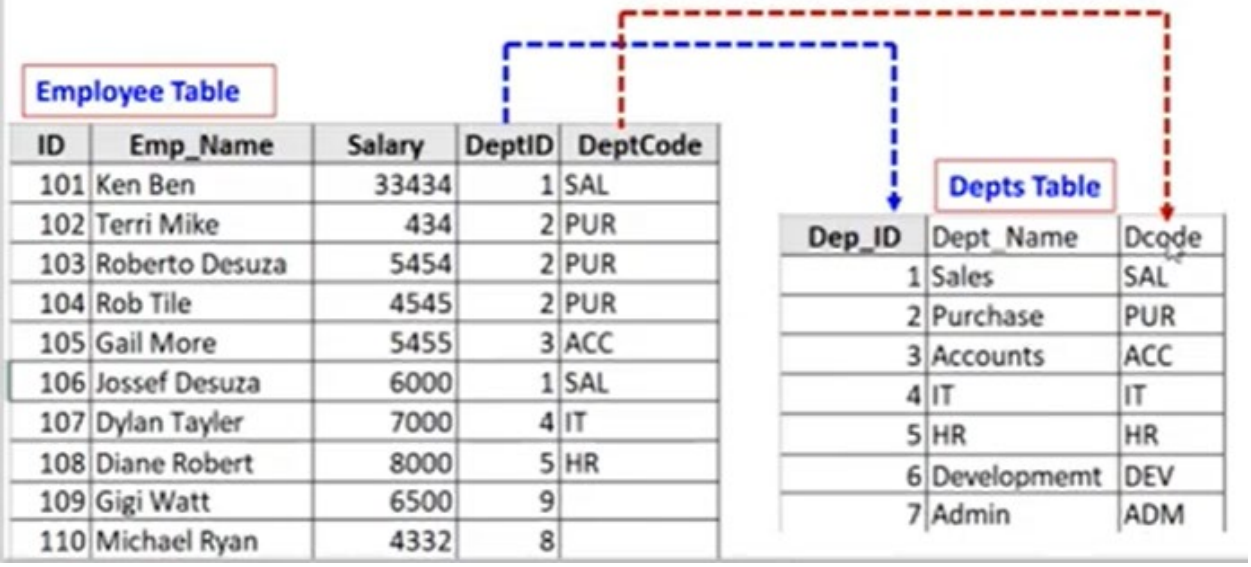
Employee Table

ID	Emp_Name	Salary	DeptID
101	Ken	33434	1
102	Terri	434	2
103	Roberto	5454	2
104	Rob	4545	2
105	Gail	5455	3
106	Jossef	6000	1
107	Dylan	7000	4
108	Diane	8000	5
109	Gigi	6500	9
110	Michael	4332	8

DAX LOOKUPVALUE Functions

➢ **Example: Multiple Values**

```
DeptName = LOOKUPVALUE(Depts[Dept_Name], Depts[Dep_ID], Employee[DeptID],
Depts[Dcode], Employee[DeptCode], "Not Matched")
```



Employee Table

ID	Emp_Name	Salary	DeptID	DeptCode
101	Ken Ben	33434	1	SAL
102	Terri Mike	434	2	PUR
103	Roberto Desuza	5454	2	PUR
104	Rob Tile	4545	2	PUR
105	Gail More	5455	3	ACC
106	Jossef Desuza	6000	1	SAL
107	Dylan Tayler	7000	4	IT
108	Diane Robert	8000	5	HR
109	Gigi Watt	6500	9	
110	Michael Ryan	4332	8	

Depts Table

Dep_ID	Dept_Name	Dcode
1	Sales	SAL
2	Purchase	PUR
3	Accounts	ACC
4	IT	IT
5	HR	HR
6	Developmemt	DEV
7	Admin	ADM

```
Dept_name =
LOOKUPVALUE(Depts[Dept_Name],Depts[Dep_ID],Employee[DeptID],Depts[Dcod
e],Employee[DeptCode],"Not Found")
```

4] DAX | PATH | PATHITEM | PATHCONTAINS | PATHLENGTH

DAX Functions

1. PATH
2. PATHLENGTH
3. PATHITEM
4. PATHITEMREVERSE
5. PATHCONTAINS

DAX PATH Functions

- PATH function returns a delimited text which contains all the parents to the current row, starting with the oldest or topmost until current. In our example, on a table that contains employees, the managers of employees, and the managers of the managers, you can return the path that connects an employee to his or her manager.
- **Syntax:**

PATH(<ID_columnName>, <parent_columnName>)

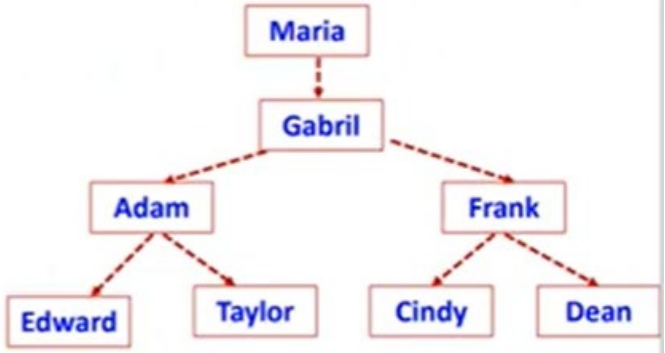
- The name of an existing column containing the unique identifier for rows in the table.
- The data type of the value in ***ID_columnName*** must be text or integer, and must also be the same data type as the column referenced in ***parent_columnName.***

- The name of an existing column containing the unique identifier for the parent of the current row.
- The data type of the value in ***parent_columnName*** data type must be text or integer, and must be the same data type as the value in ***ID_columnName***.

DAX PATH Functions

➢ **Example:**

Employee	Manager
Edward	Adam
Taylor	Adam
Cindy	Frank
Dean	Frank
Adam	Gabaril
Frank	Gabaril
Gabril	Maria
Maria	Maria



- Values in first column must be be unique.
- Values in the second column don´t have to be uniqe - since an item can be superior to multiple items.

```
1 Manager_level = PATH(Employee[Employee],Employee[Manager])
```

Employee	Manager	Manager_level
Edward	Adam	Maria|Gabaril|Adam|Edward
Taylor	Adam	Maria|Gabaril|Adam|Taylor
Cindy	Frank	Maria|Gabaril|Frank|Cindy
Dean	Frank	Maria|Gabaril|Frank|Dean
Adam	Gabaril	Maria|Gabaril|Adam
Frank	Gabaril	Maria|Gabaril|Frank
Gabaril	Maria	Maria|Gabaril
Maria	Maria	Maria

DAX PATHLENGTH Functions

Returns the number of items in a particular path string. This function returns 1 for the path generated for an ID at the top/root of a hierarchy.

Syntax:

PATHLENGTH(<PATH>)

Example:

PATHLENGTH("Maria|Gabaril|Adam|Edward") -> 4

PATHLENGTH("Maria|Gabaril|Adam") -> 3

PATHLENGTH("Maria|Gabaril") -> 2

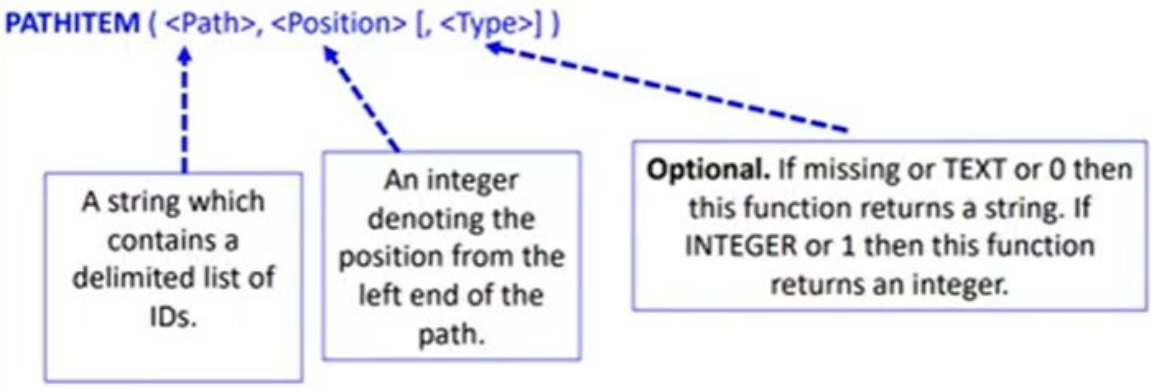
```
1  Path_Lenght = PATHLENGTH(PATH(Employee[Employee],Employee[Manager]))
```

Employee	Manager	Manager_level	Path_Lenght
Edward	Adam	Maria|Gabaril|Adam|Edward	4
Taylor	Adam	Maria|Gabaril|Adam|Taylor	4
Cindy	Frank	Maria|Gabaril|Frank|Cindy	4
Dean	Frank	Maria|Gabaril|Frank|Dean	4
Adam	Gabaril	Maria|Gabaril|Adam	3
Frank	Gabaril	Maria|Gabaril|Frank	3
Gabaril	Maria	Maria|Gabaril	2
Maria	Maria	Maria	1

DAX PATHITEM Functions

Returns the nth item in the delimited list produced by the Path function.

Syntax:



Example:

OUTPUT : **Gabaril**

```
1  PathItem = PATHITEM(Employee[Manager_level],2)
```

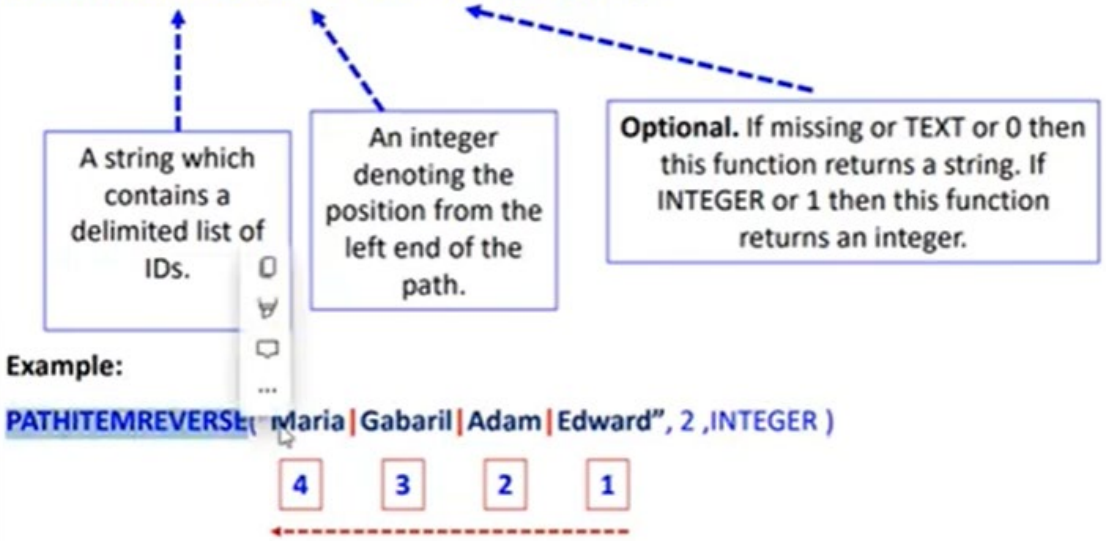
Employee	Manager	Manager_level	Path_Lenght	PathItem
Edward	Adam	Maria|Gabaril|Adam|Edward	4	Gabaril
Taylor	Adam	Maria|Gabaril|Adam|Taylor	4	Gabaril
Cindy	Frank	Maria|Gabaril|Frank|Cindy	4	Gabaril
Dean	Frank	Maria|Gabaril|Frank|Dean	4	Gabaril
Adam	Gabaril	Maria|Gabaril|Adam	3	Gabaril
Frank	Gabaril	Maria|Gabaril|Frank	3	Gabaril
Gabaril	Maria	Maria|Gabaril	2	Gabaril
Maria	Maria	Maria	1	

DAX PATHITEMRVERSE Functions

Returns the nth item in the delimited list produced by the Path function, counting backwards from the last item in the path.

Syntax:

PATHITEMREVERSE (<Path>, <Position> [, <Type>])



Example:

PATHITEMREVERSE("Maria|Gabaril|Adam|Edward", 2 ,INTEGER)

4 3 2 1

OUTPUT : Adam

```
1 PathItemReverse = PATHITEMREVERSE(Employee[Manager_level],2)
```

Employee	Manager	Manager_level	Path_Lenght	PathItem	PathItemReverse
Edward	Adam	Maria|Gabaril|Adam|Edward	4	Adam	Adam
Taylor	Adam	Maria|Gabaril|Adam|Taylor	4	Adam	Adam
Cindy	Frank	Maria|Gabaril|Frank|Cindy	4	Frank	Frank
Dean	Frank	Maria|Gabaril|Frank|Dean	4	Frank	Frank
Adam	Gabaril	Maria|Gabaril|Adam	3	Adam	Gabaril
Frank	Gabaril	Maria|Gabaril|Frank	3	Frank	Gabaril
Gabaril	Maria	Maria|Gabaril	2		Maria
Maria	Maria	Maria	1		

```
1 PathContains = PATHCONTAINS(Employee[Manager_level],"Frank")
```

Employee	Manager	Manager_level	Path_Lenght	PathItem	PathItemReverse	PathContains
Edward	Adam	Maria|Gabaril|Adam|Edward	4	Adam	Adam	False
Taylor	Adam	Maria|Gabaril|Adam|Taylor	4	Adam	Adam	False
Cindy	Frank	Maria|Gabaril|Frank|Cindy	4	Frank	Frank	True
Dean	Frank	Maria|Gabaril|Frank|Dean	4	Frank	Frank	True
Adam	Gabaril	Maria|Gabaril|Adam	3	Adam	Gabaril	False
Frank	Gabaril	Maria|Gabaril|Frank	3	Frank	Gabaril	True
Gabaril	Maria	Maria|Gabaril	2		Maria	False
Maria	Maria	Maria	1			False

```
1  PathContains = PATHCONTAINS(Employee[Manager_level],"ABC")
```

Employee	Manager	Manager_level	Path_Lenght	PathItem	PathItemReverse	PathContains
Edward	Adam	Maria|Gabaril|Adam|Edward	4	Adam	Adam	False
Taylor	Adam	Maria|Gabaril|Adam|Taylor	4	Adam	Adam	False
Cindy	Frank	Maria|Gabaril|Frank|Cindy	4	Frank	Frank	False
Dean	Frank	Maria|Gabaril|Frank|Dean	4	Frank	Frank	False
Adam	Gabaril	Maria|Gabaril|Adam	3	Adam	Gabaril	False
Frank	Gabaril	Maria|Gabaril|Frank	3	Frank	Gabaril	False
Gabaril	Maria	Maria|Gabaril	2		Maria	False
Maria	Maria	Maria	1			False

5] Nested IF Statement

Power BI DAX IF Function

- The Power BI DAX If function checks whether the given expression is True or False.

Syntax:

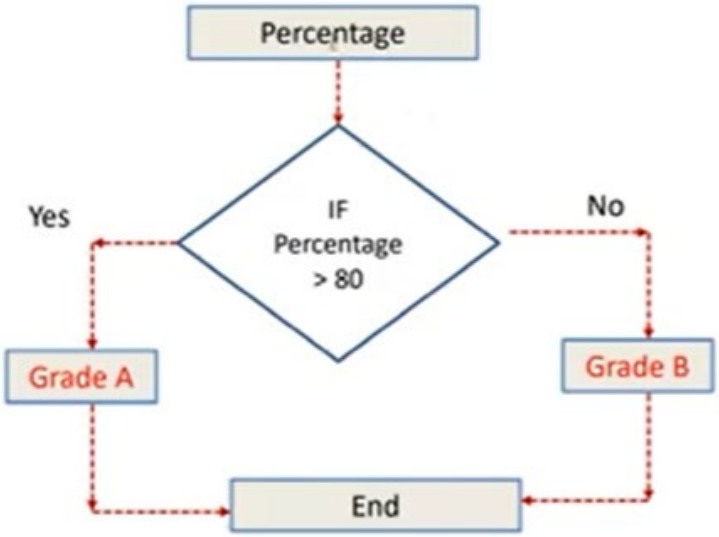
IF(Expression, True_Info, False_Info)

Example:

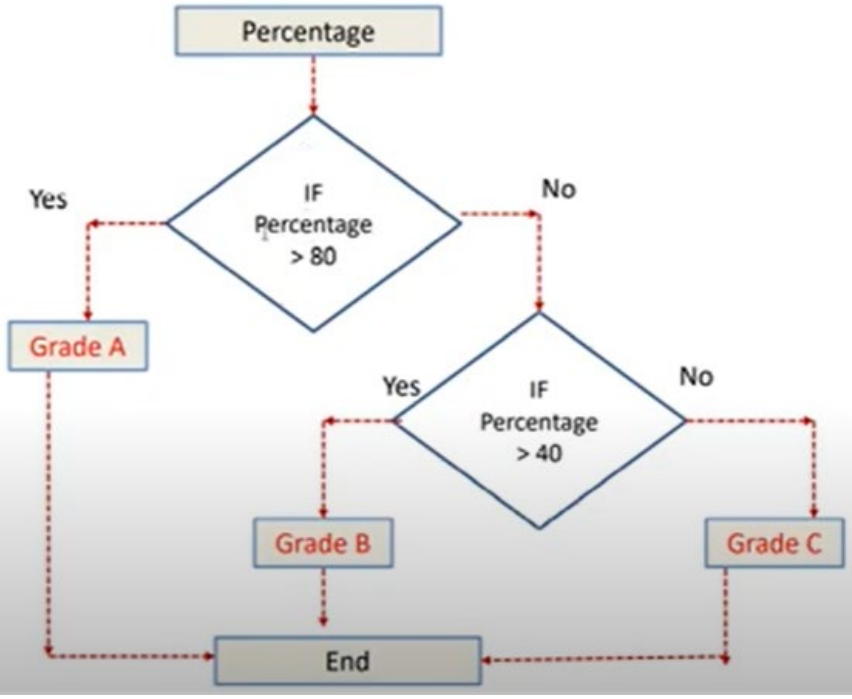
```
Grade=
IF( Percentage > 80 , "A","B")

Nested IF
Grade=
IF( Percentage > 80 , "A",
      IF( Percentage > 40 , "B","C")
)
```

Power BI DAX IF Function



Power BI DAX NESTED IF Function



```
1 Grade = IF('Student Marks'[Percentage]> 80,"A","B")
```

RollNo	St_Name	Class	English	Math	Science	Total	Percentage	Grade
1	Jesica	8th	88	55	77	220	73.33	B
2	Merry	9th	90	66	89	245	81.67	A
3	Mona	4th	67	87	98	252	84.00	A
4	Loki	8th	65	90	65	220	73.33	B
5	Smith	7th	90	87	45	222	74.00	B
6	Kider	3rd	43	54	24	121	40.33	B
7	Mike	6th	78	76	78	232	77.33	B

```
1 Grade = IF('Student Marks'[Percentage]> 80,"A",
2     IF(
3         AND('Student Marks'[Percentage]>60,'Student Marks'[Percentage]<80),"B",
4         IF(
5             AND('Student Marks'[Percentage] > 40 , 'Student Marks'[Percentage] < 60),"C","Fail"
6         )
7     )
8 )
```

RollNo	St_Name	Class	English	Math	Science	Total	Percentage	Grade
1	Jesica	8th	88	55	77	220	73.33	B
2	Merry	9th	90	66	89	245	81.67	A
3	Mona	4th	67	87	98	252	84.00	A
4	Loki	8th	65	90	65	220	73.33	B
5	Smith	7th	90	87	45	222	74.00	B
6	Kider	3rd	43	54	24	121	40.33	C
7	Mike	6th	78	76	78	232	77.33	B

```
1 Sales_status = IF(
2     'Sales Summary'[Net Sales] > 5000, "Green",
3     IF(
4         AND('Sales Summary'[Net Sales] >  3000 , 'Sales Summary'[Net Sales] < 5000),"Yellow","Red"
5         )
6 )
```

Year of Sales	Net Sales	Sales_status
2000	4344	Yellow
2001	430	Red
2002	675	Red
2003	2899	Red
2004	3467	Yellow
2005	6000	Green
2006	6000	Green
2007	3200	Yellow
2008	4000	Yellow
2009	2000	Red
2010	1000	Red

6] DAX Date Functions

Power BI DAX Date Function

CALENDAR	CALENDARAUTO	DATE
DATEDIFF	DATEVALUE	DAY
EOMONTH	HOUR	MINUTE
NOW	QUARTER	SECOND
TIME	TIMEVALUE	TODAY
UTCNOW	WEEKDAY	WEEKDAY
WEEKNUM	YEAR	YEARFRAC

Power BI DAX Date Function

Syntax

CALENDAR(<start_date>, <end_date>)

DAX expression that returns a date time value. Format : yyy,mm,dd

- Returns a table with a single column named "Date" that contains a contiguous set of the dates.
- Now, mark it as a date table by going to **Table Tools > Mark as Date Table** for format operation.
- An error is returned if start_date is greater than end_date.

Example

```
CALENDAR(Date(2020,1,1), Date(2021,12,1))
```

Syntax

YEARFRAC(start_date, end_date)

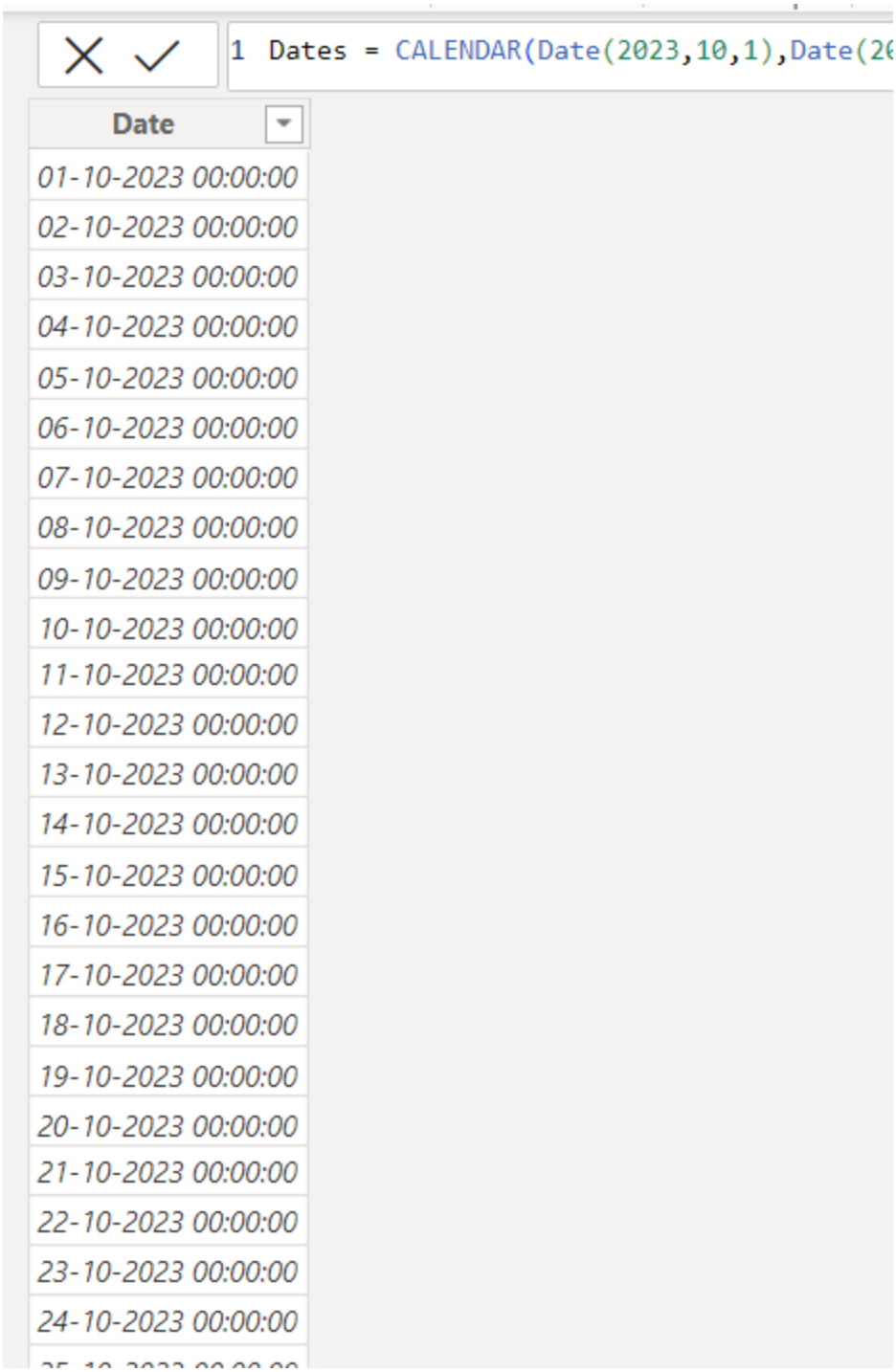
This function returns the fractional difference between the start date and end date in numbers.

Example

Age = **YEARFRAC**('Sales Summary'[Date_of_birth], **NOW()**)

```
1  Age = YEARFRAC('Sales Summary'[Date_of_birth],NOW())
```

Employee	Date_of_birth	Age
John	*12 February 1908*	*113*
King	*12 August 1988*	*33*
Smith	*02 September 1999*	*22*
Mike	*09 September 1955*	*66*
Tonu	*09 September 2000*	*21*
Russel	*08 September 2000*	*21*
Meeka	*08 August 1998*	*23*
Susan	*08 August 1999*	*22*


1 Dates = CALENDAR(Date(2023,10,1),Date(2
Date
01-10-2023 00:00:00
02-10-2023 00:00:00
03-10-2023 00:00:00
04-10-2023 00:00:00
05-10-2023 00:00:00
06-10-2023 00:00:00
07-10-2023 00:00:00
08-10-2023 00:00:00
09-10-2023 00:00:00
10-10-2023 00:00:00
11-10-2023 00:00:00
12-10-2023 00:00:00
13-10-2023 00:00:00
14-10-2023 00:00:00
15-10-2023 00:00:00
16-10-2023 00:00:00
17-10-2023 00:00:00
18-10-2023 00:00:00
19-10-2023 00:00:00
20-10-2023 00:00:00
21-10-2023 00:00:00
22-10-2023 00:00:00
23-10-2023 00:00:00
24-10-2023 00:00:00

Structure Formatting

```
1  Age = YEARFRAC('Sample file'[Date of Birth],NOW())
```

Employee Name	Date of Birth	Date of Joining	Age
John Smith	15 March 1980	20 July 2010	44
Jane Doe	10 August 1992	12 April 2015	31
Michael Brown	05 December 1975	30 September 2008	48
Emily Johnson	20 June 1988	25 March 2012	35
David Lee	18 February 1990	15 October 2017	34
Sarah Wilson	30 November 1985	05 August 2011	38
James Davis	25 April 1982	18 November 2009	42
Olivia Clark	12 July 1995	08 June 2016	28
William Turner	28 September 1978	10 January 2013	45
Sophia Harris	03 October 1989	02 May 2014	34
Robert Evans	08 January 1987	07 December 2018	37
Emma Hall	27 March 1991	14 June 2007	33
Christopher Smith	14 August 1976	20 February 2019	47
Ava Martin	22 May 1994	22 October 2010	29
Daniel White	19 July 1983	28 November 2015	40

Power BI DAX Date Function

Syntax

EDATE(start_date, Months)

returns previous dates or future dates. Adding or subtracting months

Example

Edate_Val = EDATE('Sales Summary'[Date_of_birth], 2)

```
1  EDAte_Val = EDATE('Sales Summary'[Date_of_birth],2)
```

Employee	Date_of_birth	EDAte_Val
John	12 February 1908	12-04-1908 00:00:00
King	12 August 1988	12-10-1988 00:00:00
Smith	02 September 1999	02-11-1999 00:00:00
Mike	09 September 1955	09-11-1955 00:00:00
Tonu	09 September 2000	09-11-2000 00:00:00
Russel	08 September 2000	08-11-2000 00:00:00
Meeka	08 August 1998	08-10-1998 00:00:00
Susan	08 August 1999	08-10-1999 00:00:00

```
1 EDate_value = EDATE('Sample file'[Date of Joining],2)
```

Employee Name	Date of Birth	Date of Joining	Age	EDate_value
John Smith	15 March 1980	20 July 2010	44	20-09-2010 00:00:00
Jane Doe	10 August 1992	12 April 2015	31	12-06-2015 00:00:00
Michael Brown	05 December 1975	30 September 2008	48	30-11-2008 00:00:00
Emily Johnson	20 June 1988	25 March 2012	35	25-05-2012 00:00:00
David Lee	18 February 1990	15 October 2017	34	15-12-2017 00:00:00
Sarah Wilson	30 November 1985	05 August 2011	38	05-10-2011 00:00:00
James Davis	25 April 1982	18 November 2009	42	18-01-2010 00:00:00
Olivia Clark	12 July 1995	08 June 2016	28	08-08-2016 00:00:00
William Turner	28 September 1978	10 January 2013	45	10-03-2013 00:00:00
Sophia Harris	03 October 1989	02 May 2014	34	02-07-2014 00:00:00
Robert Evans	08 January 1987	07 December 2018	37	07-02-2019 00:00:00
Emma Hall	27 March 1991	14 June 2007	33	14-08-2007 00:00:00
Christopher Smith	14 August 1976	20 February 2019	47	20-04-2019 00:00:00
Ava Martin	22 May 1994	22 October 2010	29	22-12-2010 00:00:00
Daniel White	19 July 1983	28 November 2015	40	28-01-2016 00:00:00

Syntax
EOMONTH(start_date, Months)

function (EOMONTH means the end of the month) display the last day in a month. You can use the second argument to return previous dates or future dates.

Example
EMONTH_Val = **EOMONTH**('Sales Summary'[Date_of_birth], **2**)

```
1 EMOnth_Val = EOMONTH('Sales Summary'[Date_of_birth].[Date],2)
```

Employee	Date_of_birth	EMOnth_Val
John	12 February 1908	30-04-1908 00:00:00
King	12 August 1988	31-10-1988 00:00:00
Smith	02 September 1999	30-11-1999 00:00:00
Mike	09 September 1955	30-11-1955 00:00:00
Tonu	09 September 2000	30-11-2000 00:00:00
Russel	08 September 2000	30-11-2000 00:00:00
Meeka	08 August 1998	31-10-1998 00:00:00

Syntax

DATEDIFF(start_date, end_date, difference value)

Function returns the date difference between the start date and end date. Third argument to select the difference value.
Power BI DAX DATEDIFF function accepts **YEAR, MONTH, DAY, SECOND, MINUTE, HOUR, QUARTER, WEEK.**

Example

DateDiff_Val = DATEDIFF('Date Data'[DOJ].[Date], NOW(),YEAR)

```
1 DateDiff_Val = DATEDIFF('Date Data'[DOJ].[Date], NOW(),YEAR )
```

Employee	DOB	DOJ	DateDiff_Val
John	12 February 1908	12 February 1990	31
King	12 August 1988	12 August 1999	22
Smith	02 September 1999	02 September 2001	20
Mike	09 September 1955	09 July 1989	32
Tonu	09 September 2000	09 September 2010	11
Russel	08 September 2000	08 September 2005	16

```
1 Service Year = DATEDIFF('Sample file'[Date of Joining], NOW(),
```

DAY
HOUR
MINUTE
MONTH
QUARTER
SECOND
WEEK
YEAR

Employee Name	Date of Birth	Date of Joining	Age	EDate_value	
John Smith	15 March 1980	20 July 2010	44	20-05-2010 00:00:0	
Jane Doe	10 August 1992	12 April 2015	31	12-02-2015 00:00:0	
Michael Brown	05 December 1975	30 September 2008	48	30-07-2008 00:00:0	
Emily Johnson	20 June 1988	25 March 2012	35	25-01-2012 00:00:0	
David Lee	18 February 1990	15 October 2017	34	15-08-2017 00:00:0	
Sarah Wilson	30 November 1985	05 August 2011	38	05-06-2011 00:00:00	31-10-2011 00:00:00
James Davis	25 April 1982	18 November 2009	42	18-09-2009 00:00:00	31-01-2010 00:00:00
Olivia Clark	12 July 1995	08 June 2016	28	08-04-2016 00:00:00	31-08-2016 00:00:00
William Turner	28 September 1978	10 January 2013	45	10-11-2012 00:00:00	31-03-2013 00:00:00
Sophia Harris	03 October 1989	02 May 2014	34	02-03-2014 00:00:00	31-07-2014 00:00:00
Robert Evans	08 January 1987	07 December 2018	37	07-10-2018 00:00:00	28-02-2019 00:00:00
Emma Hall	27 March 1991	14 June 2007	33	14-04-2007 00:00:00	31-08-2007 00:00:00
Christopher Smith	14 August 1976	20 February 2019	47	20-12-2018 00:00:00	30-04-2019 00:00:00
Ava Martin	22 May 1994	22 October 2010	29	22-08-2010 00:00:00	31-12-2010 00:00:00
Daniel White	19 July 1983	28 November 2015	40	28-09-2015 00:00:00	31-01-2016 00:00:00

```
1 Service Year = DATEDIFF('Sample file'[Date of Joining], NOW(),YEAR)
```

Employee Name	Date of Birth	Date of Joining	Age	EDate_value	EndOfMonth_value	Service Year
John Smith	15 March 1980	20 July 2010	44	20-05-2010 00:00:00	30-09-2010 00:00:00	13
Jane Doe	10 August 1992	12 April 2015	31	12-02-2015 00:00:00	30-06-2015 00:00:00	8
Michael Brown	05 December 1975	30 September 2008	48	30-07-2008 00:00:00	30-11-2008 00:00:00	15
Emily Johnson	20 June 1988	25 March 2012	35	25-01-2012 00:00:00	31-05-2012 00:00:00	11
David Lee	18 February 1990	15 October 2017	34	15-08-2017 00:00:00	31-12-2017 00:00:00	6
Sarah Wilson	30 November 1985	05 August 2011	38	05-06-2011 00:00:00	31-10-2011 00:00:00	12
James Davis	25 April 1982	18 November 2009	42	18-09-2009 00:00:00	31-01-2010 00:00:00	14
Olivia Clark	12 July 1995	08 June 2016	28	08-04-2016 00:00:00	31-08-2016 00:00:00	7
William Turner	28 September 1978	10 January 2013	45	10-11-2012 00:00:00	31-03-2013 00:00:00	10
Sophia Harris	03 October 1989	02 May 2014	34	02-03-2014 00:00:00	31-07-2014 00:00:00	9
Robert Evans	08 January 1987	07 December 2018	37	07-10-2018 00:00:00	28-02-2019 00:00:00	5
Emma Hall	27 March 1991	14 June 2007	33	14-04-2007 00:00:00	31-08-2007 00:00:00	16
Christopher Smith	14 August 1976	20 February 2019	47	20-12-2018 00:00:00	30-04-2019 00:00:00	4
Ava Martin	22 May 1994	22 October 2010	29	22-08-2010 00:00:00	31-12-2010 00:00:00	13
Daniel White	19 July 1983	28 November 2015	40	28-09-2015 00:00:00	31-01-2016 00:00:00	8

Power BI DAX Date Function

Syntax

UTCNOW()

Returns a current date and time in datetime format expressed in coordinated universal time (UTC). Coordinated Universal Time (UTC) is the basis for civil time today.

Example

UTCNow_Date = UTCNOW()

```
1 UTCNow_Date = UTCNOW()
```

Employee	DOB	DOJ	UTCNow_Date
John	12 February 1908	12 February 1990	19-07-2021 10:04:44
King	12 August 1988	12 August 1999	19-07-2021 10:04:44
Smith	02 September 1999	02 September 2001	19-07-2021 10:04:44
Mike	09 September 1955	09 July 1989	19-07-2021 10:04:44
Tonu	09 September 2000	09 September 2010	19-07-2021 10:04:44
Russel	08 September 2000	08 September 2005	19-07-2021 10:04:44
Meeka	08 August 1998	08 August 1990	19-07-2021 10:04:44

Structure | Calendars | Relationships | Calculations

```
1 UTC Now = UTCNOW()
```

Employee Name	Date of Birth	Date of Joining	Age	EDate_value	EndOfMonth_value	Service Year	UTC Now
John Smith	15 March 1980	20 July 2010	44	20-05-2010 00:00:00	30-09-2010 00:00:00	13	29-10-2023 09:37:09
Jane Doe	10 August 1992	12 April 2015	31	12-02-2015 00:00:00	30-06-2015 00:00:00	8	29-10-2023 09:37:09
Michael Brown	05 December 1975	30 September 2008	48	30-07-2008 00:00:00	30-11-2008 00:00:00	15	29-10-2023 09:37:09
Emily Johnson	20 June 1988	25 March 2012	35	25-01-2012 00:00:00	31-05-2012 00:00:00	11	29-10-2023 09:37:09
David Lee	18 February 1990	15 October 2017	34	15-08-2017 00:00:00	31-12-2017 00:00:00	6	29-10-2023 09:37:09
Sarah Wilson	30 November 1985	05 August 2011	38	05-06-2011 00:00:00	31-10-2011 00:00:00	12	29-10-2023 09:37:09
James Davis	25 April 1982	18 November 2009	42	18-09-2009 00:00:00	31-01-2010 00:00:00	14	29-10-2023 09:37:09
Olivia Clark	12 July 1995	08 June 2016	28	08-04-2016 00:00:00	31-08-2016 00:00:00	7	29-10-2023 09:37:09
William Turner	28 September 1978	10 January 2013	45	10-11-2012 00:00:00	31-03-2013 00:00:00	10	29-10-2023 09:37:09
Sophia Harris	03 October 1989	02 May 2014	34	02-03-2014 00:00:00	31-07-2014 00:00:00	9	29-10-2023 09:37:09
Robert Evans	08 January 1987	07 December 2018	37	07-10-2018 00:00:00	28-02-2019 00:00:00	5	29-10-2023 09:37:09
Emma Hall	27 March 1991	14 June 2007	33	14-04-2007 00:00:00	31-08-2007 00:00:00	16	29-10-2023 09:37:09
Christopher Smith	14 August 1976	20 February 2019	47	20-12-2018 00:00:00	30-04-2019 00:00:00	4	29-10-2023 09:37:09
Ava Martin	22 May 1994	22 October 2010	29	22-08-2010 00:00:00	31-12-2010 00:00:00	13	29-10-2023 09:37:09
Daniel White	19 July 1983	28 November 2015	40	28-09-2015 00:00:00	31-01-2016 00:00:00	8	29-10-2023 09:37:09

7] Logical Functions in DAX

Power BI Logical Functions

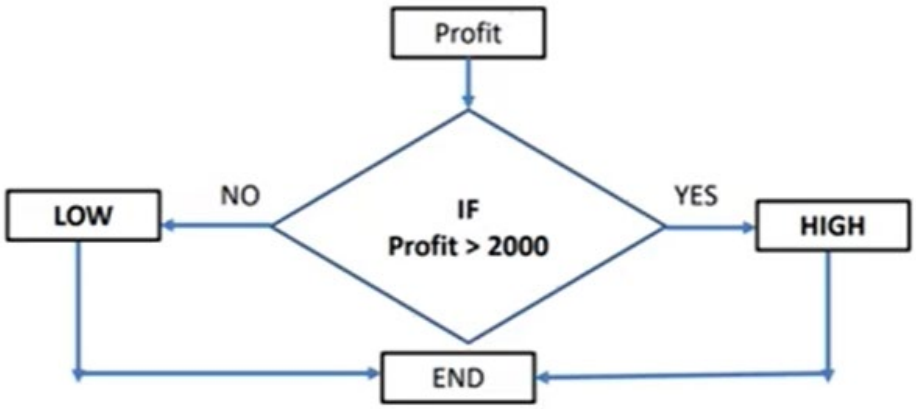
Function	Description
IF	Checks a condition, and returns one value when TRUE, otherwise it returns a second value.
AND	Checks whether both arguments are TRUE, returns TRUE if both arguments are TRUE.
COALESCE	Returns the first expression that does not evaluate to BLANK
TRUE	Returns the logical value TRUE.
FALSE	Returns the logical value FALSE.
IFERROR	Evaluates an expression and returns a value if the expression returns an error
NOT	Changes FALSE to TRUE, or TRUE to FALSE.
OR	Checks whether one of the arguments is TRUE to return TRUE.
SWITCH	Evaluates an expression against a list of values and returns one of multiple possible result expressions.
IN	The Power BI DAX IN function restricts the calculation to specified columns.

IF Function

- Checks a condition, and returns one value when it's TRUE, otherwise it returns a second value.

Syntax

- IF(<logical_test>, <value_if_true>[, <value_if_false>])



- Status = IF(('Sales Summary'[Gross Sales]-'Sales Summary'[Net Sales])> 100,"High Profit","Low Profit")

```
1 Sales_Status = IF('Sales Summary'[Net Sales]>500,"High","Low")
```

Country	Product	Qty Sold	Year of Sales	Gross Sales	Net Sales	Sales_Status
Canada	Pen	20	2019	420	400	Low
Germany	Pen	15	2019	30	15	Low
France	Pen	1999	2018	9995	5997	High
Germany	Pen	350	2019	2450	350	Low
Mexico	Pen	12	2018	156	96	Low
Germany	Pen	7	2020	84	70	Low
Germany	Pen	15	2019	45	15	Low
Canada	Pen	120	2020	5280	4800	High
France	Pen	15	2019	315	165	Low
Germany	Pencel	7	2018	238	140	Low
Mexico	Pencel	22	2020	462	242	Low
Canada	Pencel	12	2019	672	276	Low
Mexico	Pencel	15	2019	825	330	Low
Germany	Pencel	20	2019	1540	660	High
Canada	Pad	800	2018	71200	64800	High
United States of America	Pad	7	2019	63	21	Low
Canada	Pad	125	2020	9500	9375	High
Mexico	Pad	12	2019	540	252	Low
Canada	Pad	150	2018	4800	4650	High
Germany	Mouse	12	2019	672	276	Low
Germany	Mouse	125	2018	9750	5625	High
Mexico	Mouse	125	2019	5500	5250	High
France	Mouse	300	2020	9600	6300	High
Mexico	Mouse	12	2019	144	132	Low

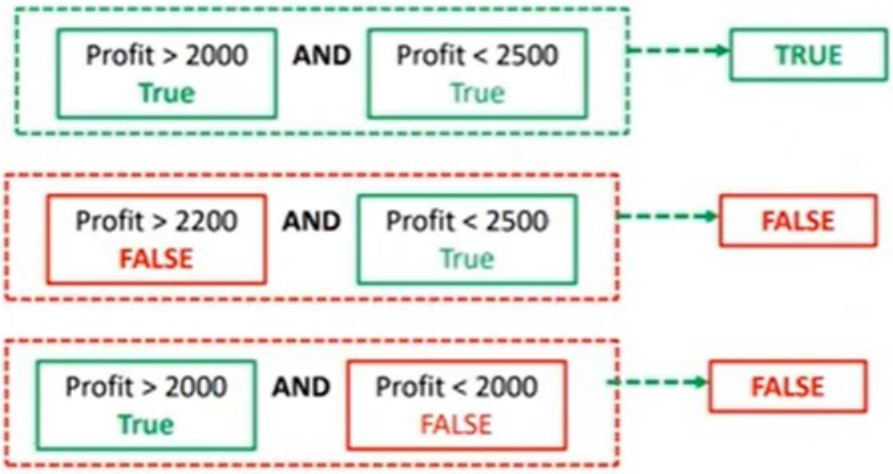
AND Function

- Checks a condition, and returns one value when it's TRUE, otherwise it returns a second value.

Syntax

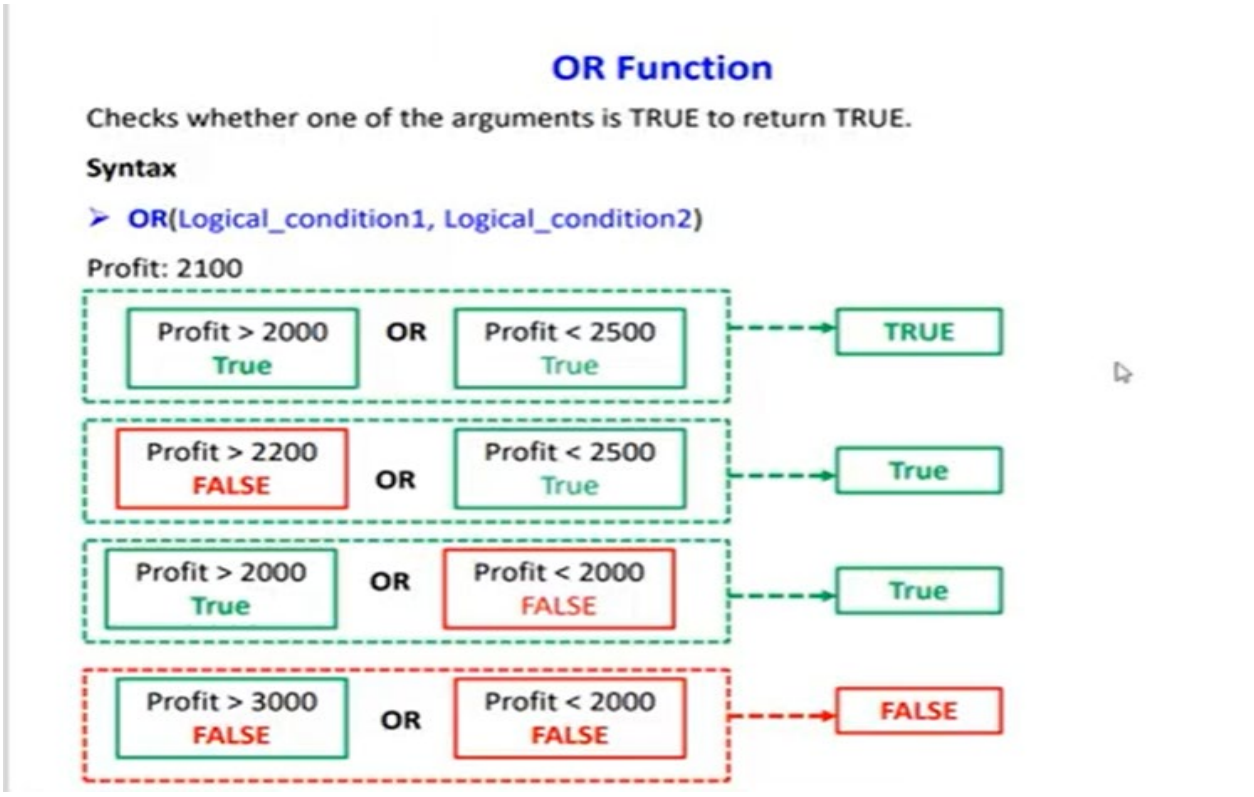
- **AND** (Logical_condition1, Logical_condition2)

Profit: 2100




```
1 Canada Sales = IF(AND('Sales Summary'[Country] = "Canada",'Sales Summary'[Net Sales]>500),"Valid","Not Valid")
```

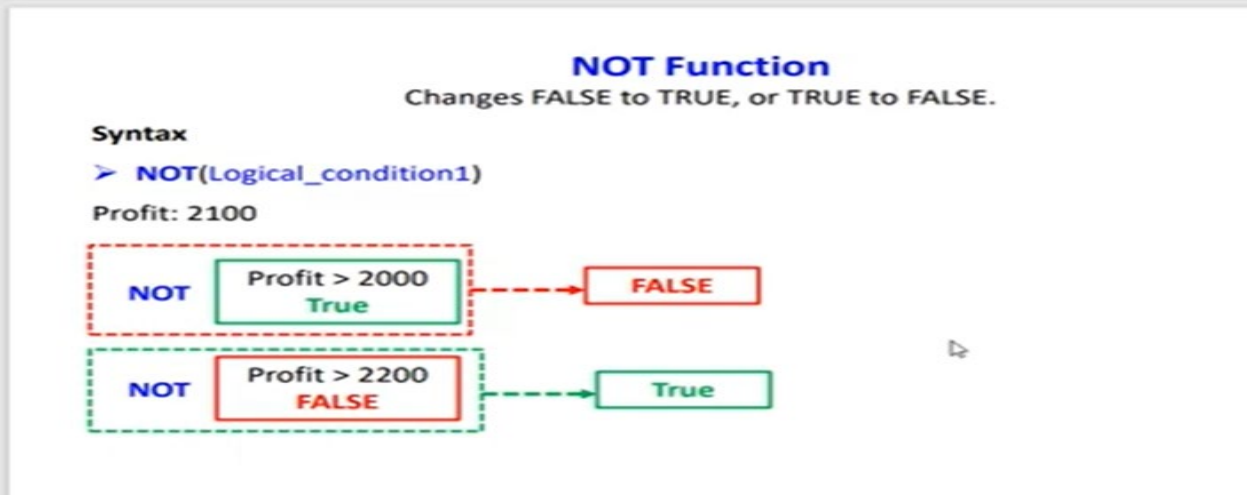
Country	Product	Qty Sold	Year of Sales	Gross Sales	Net Sales	Canada Sales
Canada	Pen	20	2019	420	400	Not Valid
Germany	Pen	15	2019	30	15	Not Valid
France	Pen	1999	2018	9995	5997	Not Valid
Germany	Pen	350	2019	2450	350	Not Valid
Mexico	Pen	12	2018	156	96	Not Valid
Germany	Pen	7	2020	84	70	Not Valid
Germany	Pen	15	2019	45	15	Not Valid
Canada	Pen	120	2020	5280	4800	Valid
France	Pen	15	2019	315	165	Not Valid
Germany	Pencel	7	2018	238	140	Not Valid
Mexico	Pencel	22	2020	462	242	Not Valid
Canada	Pencel	12	2019	672	276	Not Valid
Mexico	Pencel	15	2019	825	330	Not Valid
Germany	Pencel	20	2019	1540	660	Not Valid
Canada	Pad	800	2018	71200	64800	Valid
United States of America	Pad	7	2019	63	21	Not Valid
Canada	Pad	125	2020	9500	9375	Valid
Mexico	Pad	12	2019	540	252	Not Valid
Canada	Pad	150	2018	4800	4650	Valid
Germany	Mouse	12	2019	672	276	Not Valid
Germany	Mouse	125	2018	9750	5625	Not Valid
Mexico	Mouse	125	2019	5500	5250	Not Valid




```
1 Canada Sales = IF(OR('Sales Summary'[Country] = "Canada",'Sales Summary'[Country] = "France"),"Valid","Not Valid")
```

Country	Product	Qty Sold	Year of Sales	Gross Sales	Net Sales	Canada Sales
Canada	Pen	20	2019	420	400	Valid
Germany	Pen	15	2019	30	15	Not Valid
France	Pen	1999	2018	9995	5997	Valid
Germany	Pen	350	2019	2450	350	Not Valid
Mexico	Pen	12	2018	156	96	Not Valid
Germany	Pen	7	2020	84	70	Not Valid
Germany	Pen	15	2019	45	15	Not Valid
Canada	Pen	120	2020	5280	4800	Valid
France	Pen	15	2019	315	165	Valid
Germany	Pencel	7	2018	238	140	Not Valid
Mexico	Pencel	22	2020	462	242	Not Valid
Canada	Pencel	12	2019	672	276	Valid
Mexico	Pencel	15	2019	825	330	Not Valid
Germany	Pencel	20	2019	1540	660	Not Valid
Canada	Pad	800	2018	71200	64800	Valid
United States of America	Pad	7	2019	63	21	Not Valid
Canada	Pad	125	2020	9500	9375	Valid
Mexico	Pad	12	2019	540	252	Not Valid
Canada	Pad	150	2018	4800	4650	Valid
Germany	Mouse	12	2019	672	276	Not Valid
Germany	Mouse	125	2018	9750	5625	Not Valid
Mexico	Mouse	125	2019	5500	5250	Not Valid
France	Mouse	300	2020	9600	6300	Valid
Mexico	Mouse	12	2019	144	132	Not Valid
Mexico	Mouse	15	2019	675	450	Not Valid
United States of America	Keyboard	312	2018	20904	15600	Not Valid
Canada	Keyboard	350	2019	15050	14350	Valid
United States of America	Keyboard	300	2018	9600	6300	Not Valid

Sales Summary (49 rows) Column: Canada Sales (2 distinct values)



Structure | Calendars | Relationships | Calculations

```
1 Not_Demo = IF(NOT('Sales Summary'[Country] = "Canada"),"Valid","Not Valid")
```

Country	Product	Qty Sold	Year of Sales	Gross Sales	Net Sales	Canada Sales	Not_Demo
Canada	Pen	20	2019	420	400	Valid	Not Valid
Germany	Pen	15	2019	30	15	Not Valid	Valid
France	Pen	1999	2018	9995	5997	Valid	Valid
Germany	Pen	350	2019	2450	350	Not Valid	Valid
Mexico	Pen	12	2018	156	96	Not Valid	Valid
Germany	Pen	7	2020	84	70	Not Valid	Valid
Germany	Pen	15	2019	45	15	Not Valid	Valid
Canada	Pen	120	2020	5280	4800	Valid	Not Valid
France	Pen	15	2019	315	165	Valid	Valid
Germany	Pencel	7	2018	238	140	Not Valid	Valid
Mexico	Pencel	22	2020	462	242	Not Valid	Valid
Canada	Pencel	12	2019	672	276	Valid	Not Valid
Mexico	Pencel	15	2019	825	330	Not Valid	Valid
Germany	Pencel	20	2019	1540	660	Not Valid	Valid
Canada	Pad	800	2018	71200	64800	Valid	Not Valid
United States of America	Pad	7	2019	63	21	Not Valid	Valid
Canada	Pad	125	2020	9500	9375	Valid	Not Valid
Mexico	Pad	12	2019	540	252	Not Valid	Valid
Canada	Pad	150	2018	4800	4650	Valid	Not Valid
Germany	Mouse	12	2019	672	276	Not Valid	Valid
Germany	Mouse	125	2018	9750	5625	Not Valid	Valid
Mexico	Mouse	125	2019	5500	5250	Not Valid	Valid
France	Mouse	300	2020	9600	6300	Valid	Valid
Mexico	Mouse	12	2019	144	132	Not Valid	Valid
Mexico	Mouse	15	2019	675	450	Not Valid	Valid
United States of America	Keyboard	312	2018	20904	15600	Not Valid	Valid
Canada	Keyboard	350	2019	15050	14350	Valid	Not Valid
United States of America	Keyboard	300	2018	9600	6300	Not Valid	Valid

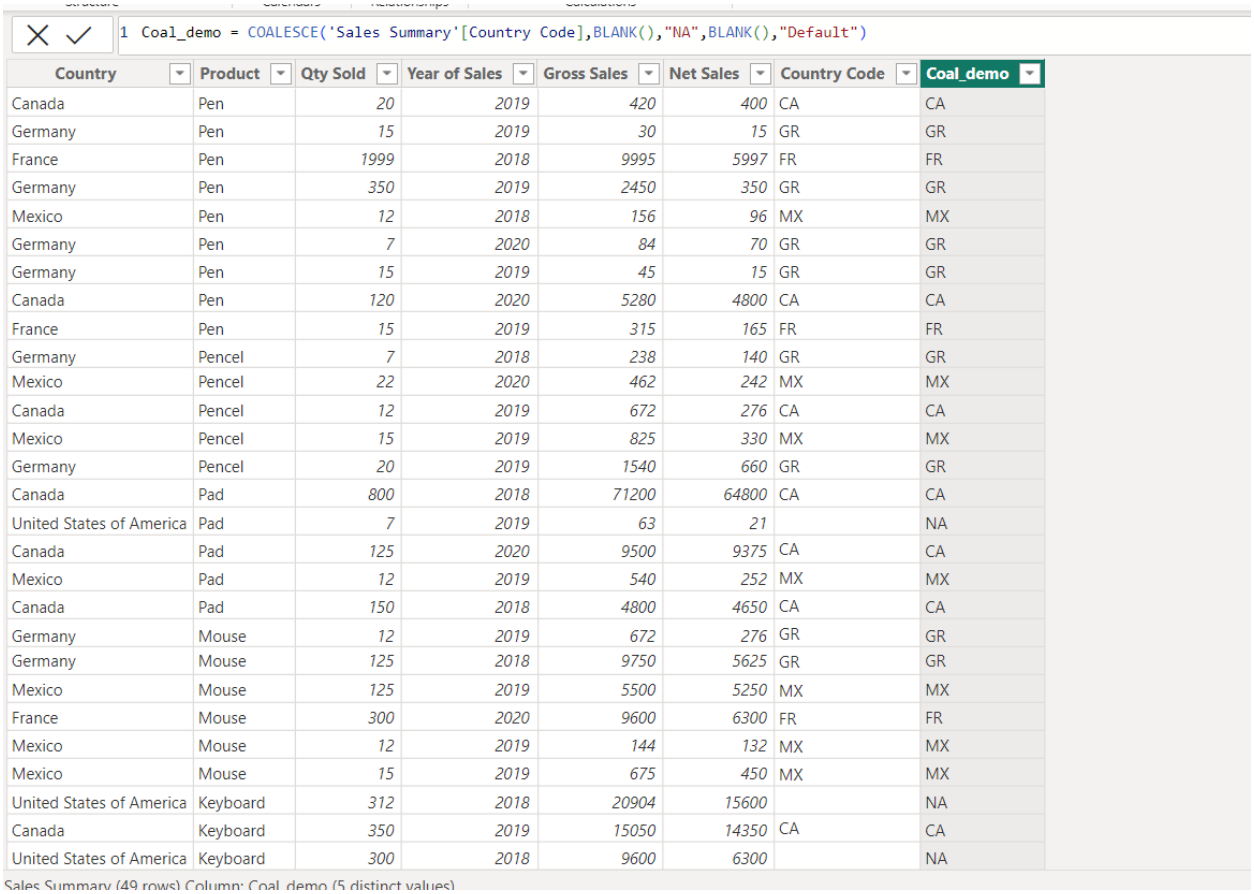
Structure | Formatting | Properties | Sort | Groups | Relationships | Calculations

```
1 Country Code = SWITCH('Sales Summary'[Country], "Canada","CA","Germany","GR","France","FR","Mexico","MX","United States of America","USA")
```

Country	Product	Qty Sold	Year of Sales	Gross Sales	Net Sales	Country Code
Canada	Pen	20	2019	420	400	CA
Germany	Pen	15	2019	30	15	GR
France	Pen	1999	2018	9995	5997	FR
Germany	Pen	350	2019	2450	350	GR
Mexico	Pen	12	2018	156	96	MX
Germany	Pen	7	2020	84	70	GR
Germany	Pen	15	2019	45	15	GR
Canada	Pen	120	2020	5280	4800	CA
France	Pen	15	2019	315	165	FR
Germany	Pencel	7	2018	238	140	GR
Mexico	Pencel	22	2020	462	242	MX
Canada	Pencel	12	2019	672	276	CA
Mexico	Pencel	15	2019	825	330	MX
Germany	Pencel	20	2019	1540	660	GR
Canada	Pad	800	2018	71200	64800	CA
United States of America	Pad	7	2019	63	21	USA
Canada	Pad	125	2020	9500	9375	CA
Mexico	Pad	12	2019	540	252	MX
Canada	Pad	150	2018	4800	4650	CA
Germany	Mouse	12	2019	672	276	GR
Germany	Mouse	125	2018	9750	5625	GR
Mexico	Mouse	125	2019	5500	5250	MX
France	Mouse	300	2020	9600	6300	FR
Mexico	Mouse	12	2019	144	132	MX
Mexico	Mouse	15	2019	675	450	MX
United States of America	Keyboard	312	2018	20904	15600	USA
Canada	Keyboard	350	2019	15050	14350	CA

```
1 Coal_demo = COALESCE('Sales Summary'[Country Code],BLANK(),"Default")
```

Country	Product	Qty Sold	Year of Sales	Gross Sales	Net Sales	Country Code	Coal_demo
Canada	Pen	20	2019	420	400	CA	CA
Germany	Pen	15	2019	30	15	GR	GR
France	Pen	1999	2018	9995	5997	FR	FR
Germany	Pen	350	2019	2450	350	GR	GR
Mexico	Pen	12	2018	156	96	MX	MX
Germany	Pen	7	2020	84	70	GR	GR
Germany	Pen	15	2019	45	15	GR	GR
Canada	Pen	120	2020	5280	4800	CA	CA
France	Pen	15	2019	315	165	FR	FR
Germany	Pencel	7	2018	238	140	GR	GR
Mexico	Pencel	22	2020	462	242	MX	MX
Canada	Pencel	12	2019	672	276	CA	CA
Mexico	Pencel	15	2019	825	330	MX	MX
Germany	Pencel	20	2019	1540	660	GR	GR
Canada	Pad	800	2018	71200	64800	CA	CA
United States of America	Pad	7	2019	63	21		Default
Canada	Pad	125	2020	9500	9375	CA	CA
Mexico	Pad	12	2019	540	252	MX	MX
Canada	Pad	150	2018	4800	4650	CA	CA
Germany	Mouse	12	2019	672	276	GR	GR
Germany	Mouse	125	2018	9750	5625	GR	GR
Mexico	Mouse	125	2019	5500	5250	MX	MX
France	Mouse	300	2020	9600	6300	FR	FR
Mexico	Mouse	12	2019	144	132	MX	MX
Mexico	Mouse	15	2019	675	450	MX	MX
United States of America	Keyboard	312	2018	20904	15600		Default
Canada	Keyboard	350	2019	15050	14350	CA	CA
United States of America	Keyboard	300	2018	9600	6300		Default



```
1 Coal_demo = COALESCE('Sales Summary'[Country Code],BLANK(),"NA",BLANK(),"Default")
```

Country	Product	Qty Sold	Year of Sales	Gross Sales	Net Sales	Country Code	Coal_demo
Canada	Pen	20	2019	420	400	CA	CA
Germany	Pen	15	2019	30	15	GR	GR
France	Pen	1999	2018	9995	5997	FR	FR
Germany	Pen	350	2019	2450	350	GR	GR
Mexico	Pen	12	2018	156	96	MX	MX
Germany	Pen	7	2020	84	70	GR	GR
Germany	Pen	15	2019	45	15	GR	GR
Canada	Pen	120	2020	5280	4800	CA	CA
France	Pen	15	2019	315	165	FR	FR
Germany	Pencel	7	2018	238	140	GR	GR
Mexico	Pencel	22	2020	462	242	MX	MX
Canada	Pencel	12	2019	672	276	CA	CA
Mexico	Pencel	15	2019	825	330	MX	MX
Germany	Pencel	20	2019	1540	660	GR	GR
Canada	Pad	800	2018	71200	64800	CA	CA
United States of America	Pad	7	2019	63	21		NA
Canada	Pad	125	2020	9500	9375	CA	CA
Mexico	Pad	12	2019	540	252	MX	MX
Canada	Pad	150	2018	4800	4650	CA	CA
Germany	Mouse	12	2019	672	276	GR	GR
Germany	Mouse	125	2018	9750	5625	GR	GR
Mexico	Mouse	125	2019	5500	5250	MX	MX
France	Mouse	300	2020	9600	6300	FR	FR
Mexico	Mouse	12	2019	144	132	MX	MX
Mexico	Mouse	15	2019	675	450	MX	MX
United States of America	Keyboard	312	2018	20904	15600		NA
Canada	Keyboard	350	2019	15050	14350	CA	CA
United States of America	Keyboard	300	2018	9600	6300		NA

Sales Summary (49 rows) Column: Coal_demo (5 distinct values)

```
1 If_Error_Demo = IFERROR('Sales Summary'[Net Sales]/0,0)
```

Country	Product	Qty Sold	Year of Sales	Gross Sales	Net Sales	Country Code	Coal_demo	If_Error_Demo
Canada	Pen	20	2019	420	400	CA	CA	0
Germany	Pen	15	2019	30	15	GR	GR	0
France	Pen	1999	2018	9995	5997	FR	FR	0
Germany	Pen	350	2019	2450	350	GR	GR	0
Mexico	Pen	12	2018	156	96	MX	MX	0
Germany	Pen	7	2020	84	70	GR	GR	0
Germany	Pen	15	2019	45	15	GR	GR	0
Canada	Pen	120	2020	5280	4800	CA	CA	0
France	Pen	15	2019	315	165	FR	FR	0
Germany	Pencel	7	2018	238	140	GR	GR	0
Mexico	Pencel	22	2020	462	242	MX	MX	0
Canada	Pencel	12	2019	672	276	CA	CA	0
Mexico	Pencel	15	2019	825	330	MX	MX	0
Germany	Pencel	20	2019	1540	660	GR	GR	0
Canada	Pad	800	2018	71200	64800	CA	CA	0
United States of America	Pad	7	2019	63	21		NA	0
Canada	Pad	125	2020	9500	9375	CA	CA	0
Mexico	Pad	12	2019	540	252	MX	MX	0
Canada	Pad	150	2018	4800	4650	CA	CA	0
Germany	Mouse	12	2019	672	276	GR	GR	0
Germany	Mouse	125	2018	9750	5625	GR	GR	0
Mexico	Mouse	125	2019	5500	5250	MX	MX	0
France	Mouse	300	2020	9600	6300	FR	FR	0
Mexico	Mouse	12	2019	144	132	MX	MX	0
Mexico	Mouse	15	2019	675	450	MX	MX	0
United States of America	Keyboard	312	2018	20904	15600		NA	0
Canada	Keyboard	350	2019	15050	14350	CA	CA	0
United States of America	Keyboard	300	2018	9600	6300		NA	0

Structure | Calendars | Relationships | Calculations

```
1 If_Error_Demo = IFERROR('Sales Summary'[Net Sales],0)
```

Country	Product	Qty Sold	Year of Sales	Gross Sales	Net Sales	Country Code	Coal_demo	If_Error_Demo
Canada	Pen	20	2019	420	400	CA	CA	400
Germany	Pen	15	2019	30	15	GR	GR	15
France	Pen	1999	2018	9995	5997	FR	FR	5997
Germany	Pen	350	2019	2450	350	GR	GR	350
Mexico	Pen	12	2018	156	96	MX	MX	96
Germany	Pen	7	2020	84	70	GR	GR	70
Germany	Pen	15	2019	45	15	GR	GR	15
Canada	Pen	120	2020	528				
France	Pen	15	2019	31				
Germany	Pencel	7	2018	23				
Mexico	Pencel	22	2020	46				
Canada	Pencel	12	2019	672				
Mexico	Pencel	15	2019	825	330	MX	MX	330
Germany	Pencel	20	2019	1540	660	GR	GR	660
Canada	Pad	800	2018	71200	64800	CA	CA	64800
United States of America	Pad	7	2019	63	21		NA	21
Canada	Pad	125	2020	9500	9375	CA	CA	9375
Mexico	Pad	12	2019	540	252	MX	MX	252
Canada	Pad	150	2018	4800	4650	CA	CA	4650
Germany	Mouse	12	2019	672	276	GR	GR	276
Germany	Mouse	125	2018	9750	5625	GR	GR	5625
Mexico	Mouse	125	2019	5500	5250	MX	MX	5250
France	Mouse	300	2020	9600	6300	FR	FR	6300
Mexico	Mouse	12	2019	144	132	MX	MX	132
Mexico	Mouse	15	2019	675	450	MX	MX	450
United States of America	Keyboard	312	2018	20904	15600		NA	15600
Canada	Keyboard	350	2019	15050	14350	CA	CA	14350
United States of America	Keyboard	300	2018	9600	6300		NA	6300

Working on it

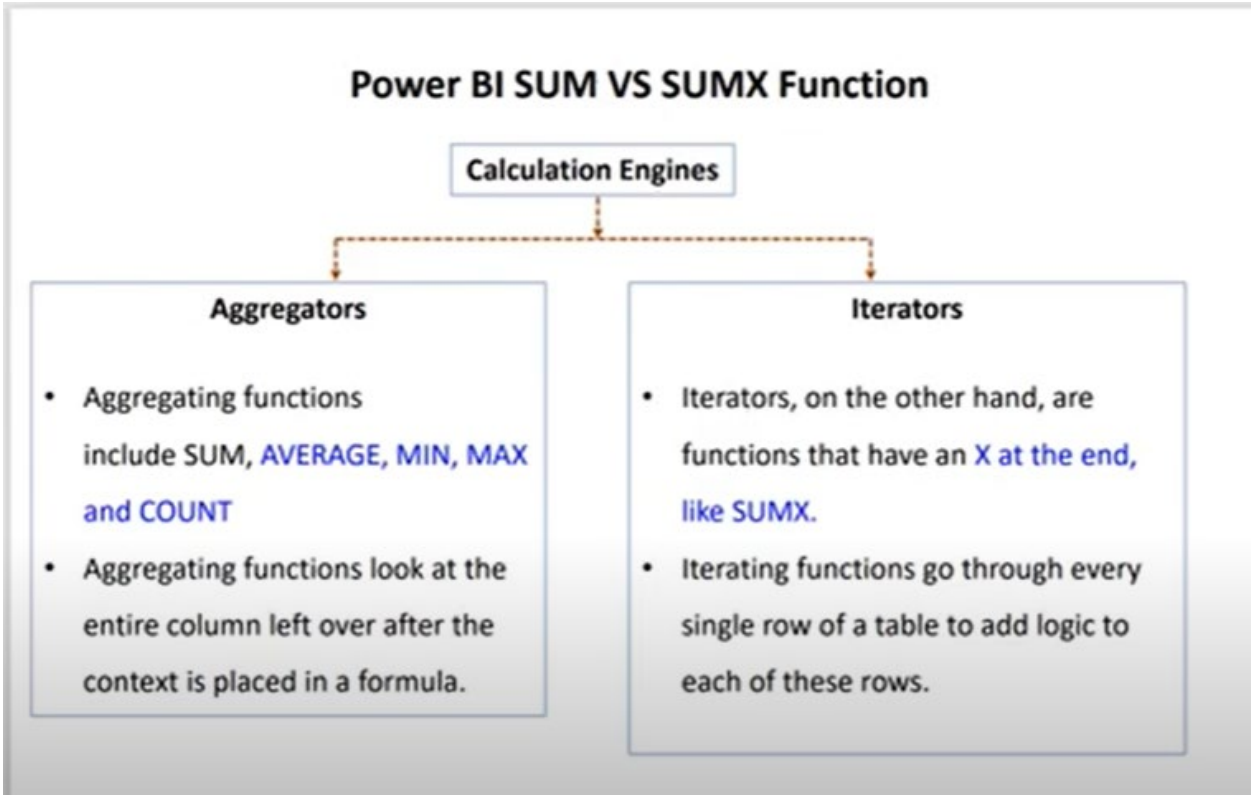
Structure | Calendars | Relationships | Calculations

```
1 InDemo = CALCULATE(SUM('Sales Summary'[Net Sales]), 'Sales Summary'[Country] IN {"Canada", "France","Germany"})
```

Country	Product	Qty Sold	Year of Sales	Gross Sales	Net Sales	Country Code	Coal_demo	If_Error_Demo	InDemo
Canada	Pen	20	2019	420	400	CA	CA	400	400
Germany	Pen	15	2019	30	15	GR	GR	15	15
France	Pen	1999	2018	9995	5997	FR	FR	5997	5997
Germany	Pen	350	2019	2450	350	GR	GR	350	350
Mexico	Pen	12	2018	156	96	MX	MX	96	
Germany	Pen	7	2020	84	70	GR	GR	70	70
Germany	Pen	15	2019	45	15	GR	GR	15	15
Canada	Pen	120	2020	5280	4800	CA	CA	4800	4800
France	Pen	15	2019	315	165	FR	FR	165	165
Germany	Pencel	7	2018	238	140	GR	GR	140	140
Mexico	Pencel	22	2020	462	242	MX	MX	242	
Canada	Pencel	12	2019	672	276	CA	CA	276	276
Mexico	Pencel	15	2019	825	330	MX	MX	330	
Germany	Pencel	20	2019	1540	660	GR	GR	660	660
Canada	Pad	800	2018	71200	64800	CA	CA	64800	64800
United States of America	Pad	7	2019	63	21		NA	21	
Canada	Pad	125	2020	9500	9375	CA	CA	9375	9375
Mexico	Pad	12	2019	540	252	MX	MX	252	
Canada	Pad	150	2018	4800	4650	CA	CA	4650	4650
Germany	Mouse	12	2019	672	276	GR	GR	276	276
Germany	Mouse	125	2018	9750	5625	GR	GR	5625	5625
Mexico	Mouse	125	2019	5500	5250	MX	MX	5250	
France	Mouse	300	2020	9600	6300	FR	FR	6300	6300
Mexico	Mouse	12	2019	144	132	MX	MX	132	
Mexico	Mouse	15	2019	675	450	MX	MX	450	
United States of America	Keyboard	312	2018	20904	15600		NA	15600	
Canada	Keyboard	350	2019	15050	14350	CA	CA	14350	14350
United States of America	Keyboard	300	2018	9600	6300		NA	6300	

Sales Summary (49 rows) Column: InDemo (34 distinct values)

8] Sum vs SumX



Power BI SUM VS SUMX Function

SUM-Aggregators	SUMX-Iterators

• Sum is a aggregating functions.
• This is used for adding numerical data together.
• It takes a column of data and adds the values to give a total.
• **Syntax**:
  • **SUM(Table Column Name)**
	• **SUMX is a** Iterator function.
	• This is also used for adding numerical data together.
	• if you want to filter the data which you are adding, then you need to use the SUMX function.
	• **Syntax:**
	  • **SUMX(Table, Expression)**

Power BI SUM VS SUMX Function

SUM-Aggregators

F	H	I
Gross Sale	**Net Sales**	
420.00	400	
30.00	15	
9995.00	5997	
2450.00	350	
156.00	96	
84.00	70	
45.00	15	
5280.00	4800	
315.00	165	
238.00	140	
462.00	242	
672.00	276	
825.00	330	

SUMX-Iterators

F	H	I
Gross Sale	**Net Sales**	
420.00	400	
30.00	15	
9995.00	5997	
2450.00	350	
156.00	96	
84.00	70	
45.00	15	
5280.00	4800	
315.00	165	
238.00	140	
462.00	242	
672.00	276	
825.00	330	

- **Total of Gross Sales->Product Wise(Filter Context)**

```
Measure_TotalSales = Sum('Sales Summary'[Gross Sale])
```

- **Profit Calculation (Filter Context)**

```
Measure_SUM_Profit = Sum('Sales Summary'[Gross Sale]) - Sum('Sales Summary'[Net Sales])
```

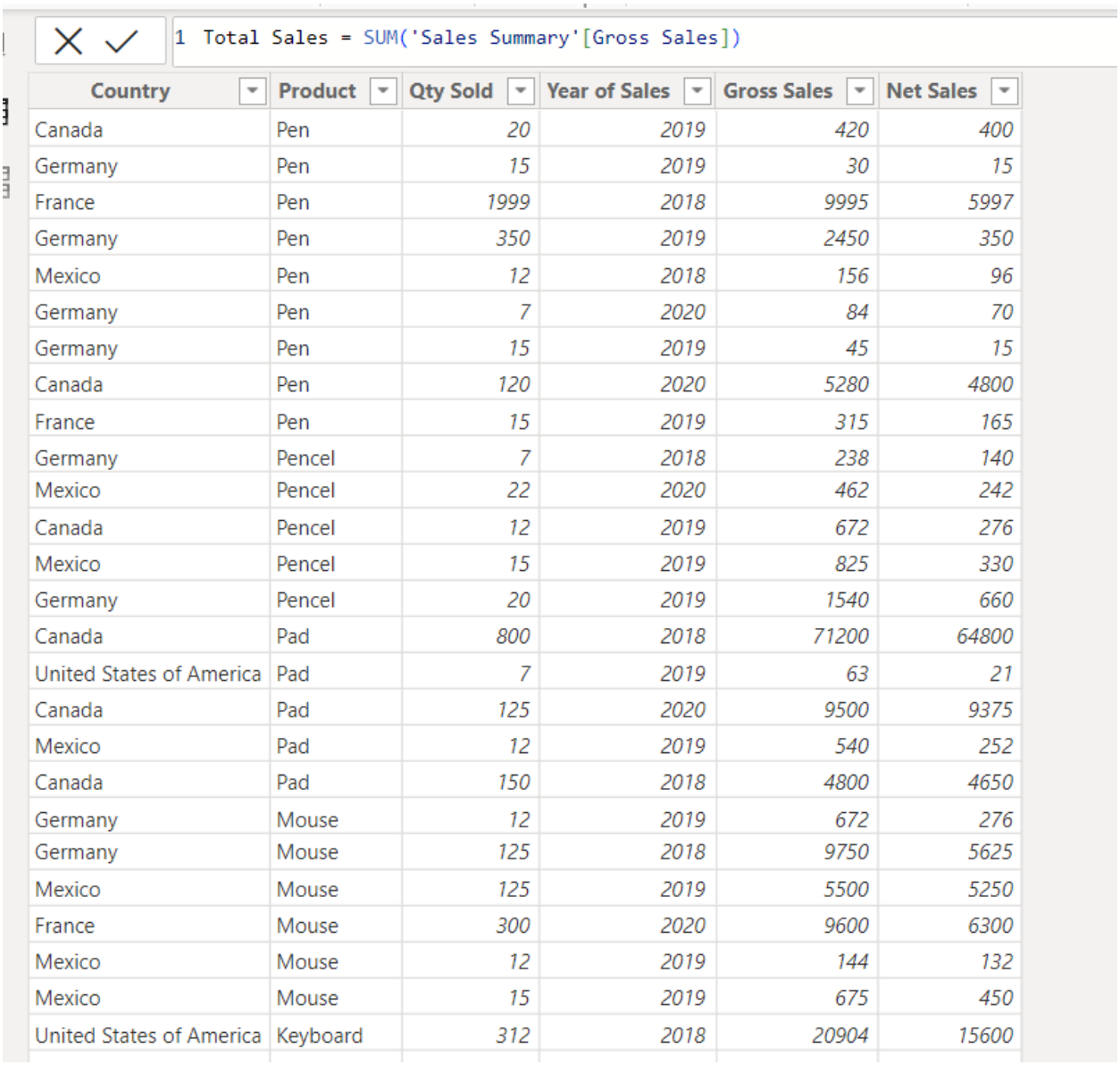
- **Profit Calculation (Filter Context)**

```
Measure_Profit = SUMX('Sales Summary','Sales Summary'[Gross Sale] - 'Sales Summary'[Net Sales])
```

Structure | Calendars | Relationships | Calculations

```
1 Profit = SUMX('Sales Summary','Sales Summary'[Gross Sales]-'Sales Summary'[Net Sales])
```

Country	Product	Qty Sold	Year of Sales	Gross Sales	Net Sales
Canada	Pen	20	2019	420	400
Germany	Pen	15	2019	30	15
France	Pen	1999	2018	9995	5997
Germany	Pen	350	2019	2450	350
Mexico	Pen	12	2018	156	96
Germany	Pen	7	2020	84	70
Germany	Pen	15	2019	45	15
Canada	Pen	120	2020	5280	4800
France	Pen	15	2019	315	165
Germany	Pencel	7	2018	238	140
Mexico	Pencel	22	2020	462	242
Canada	Pencel	12	2019	672	276
Mexico	Pencel	15	2019	825	330
Germany	Pencel	20	2019	1540	660
Canada	Pad	800	2018	71200	64800
United States of America	Pad	7	2019	63	21
Canada	Pad	125	2020	9500	9375
Mexico	Pad	12	2019	540	252
Canada	Pad	150	2018	4800	4650
Germany	Mouse	12	2019	672	276
Germany	Mouse	125	2018	9750	5625
Mexico	Mouse	125	2019	5500	5250
France	Mouse	300	2020	9600	6300
Mexico	Mouse	12	2019	144	132
Mexico	Mouse	15	2019	675	450
United States of America	Keyboard	312	2018	20904	15600
Canada	Keyboard	350	2019	15050	14350
United States of America	Keyboard	300	2018	9600	6300



```
1 Total Sales = SUM('Sales Summary'[Gross Sales])
```

Country	Product	Qty Sold	Year of Sales	Gross Sales	Net Sales
Canada	Pen	20	2019	420	400
Germany	Pen	15	2019	30	15
France	Pen	1999	2018	9995	5997
Germany	Pen	350	2019	2450	350
Mexico	Pen	12	2018	156	96
Germany	Pen	7	2020	84	70
Germany	Pen	15	2019	45	15
Canada	Pen	120	2020	5280	4800
France	Pen	15	2019	315	165
Germany	Pencel	7	2018	238	140
Mexico	Pencel	22	2020	462	242
Canada	Pencel	12	2019	672	276
Mexico	Pencel	15	2019	825	330
Germany	Pencel	20	2019	1540	660
Canada	Pad	800	2018	71200	64800
United States of America	Pad	7	2019	63	21
Canada	Pad	125	2020	9500	9375
Mexico	Pad	12	2019	540	252
Canada	Pad	150	2018	4800	4650
Germany	Mouse	12	2019	672	276
Germany	Mouse	125	2018	9750	5625
Mexico	Mouse	125	2019	5500	5250
France	Mouse	300	2020	9600	6300
Mexico	Mouse	12	2019	144	132
Mexico	Mouse	15	2019	675	450
United States of America	Keyboard	312	2018	20904	15600

9] Split Column in DAX

Power BI-How to Split Column using DAX

- **FIND Function**
  - Returns the starting position of one text string within another text string.

Syntax:

- **FIND(FindText, WithinText, StartPosition, NotFoundValue)**

- **Example:**
  - **FIND("AMAZE", "LOTUSAMAZE")**
  - **Result : 6th Position**

L	O	T	U	S	A	M	A	Z	E
1	2	3	4	5	6	7	8	9	10

Power BI-How to Split Column using DAX

- **LEFT Function**
  - Returns the specified number of characters from the start of a text string.

Syntax:

- **LEFT(FindText, NumofChars)**

- **Example:**
  - **LEFT("LOTUSAMAZE", 5)**
  - **Result : LOTUS**

L	O	T	U	S	A	M	A	Z	E
1	2	3	4	5	6	7	8	9	10

Power BI-How to Split Column using DAX

- **RIGHT Function**
  - RIGHT returns the last character or characters in a text string
  - **Syntax**:
    - **RIGHT(FindText, NumofChars)**
  - **Example:**
    - **RIGHT("LOTUSAMAZE", 5)**
    - **Result : AMAZE**

L	O	T	U	S	A	M	A	Z	E
1	2	3	4	5	6	7	8	9	10

Power BI-How to Split Column using DAX

- **LEN Function**
  - Returns the number of characters in a text string.
  - **Syntax:**
    - **LEN(FindText)**
  - **Example:**
    - **LEN("LOTUSAMAZE")**
    - **Result : 10**

L	O	T	U	S	A	M	A	Z	E
1	2	3	4	5	6	7	8	9	10

Power BI-How to Split Column using DAX

Emp_Name
Ken Ming
Terri Fernadiz
Roberto Milo
Rob Tyle
Gail Sisaliya
Jossef Fernadiz
Dylan Talor
Diane Sonya
Gigi Viz
Michael Jacson

Find First Name

LEFT(Employee[Emp_Name], **FIND**(" ",Employee[Emp_Name])-1)

Ken Ming

Step-1

Find " "(Space)

K	E	N		M	I	N	g
1	2	3	4	5	6	7	8

Step-2

LEFT("Ken Ming", **4-1**)

Step-3

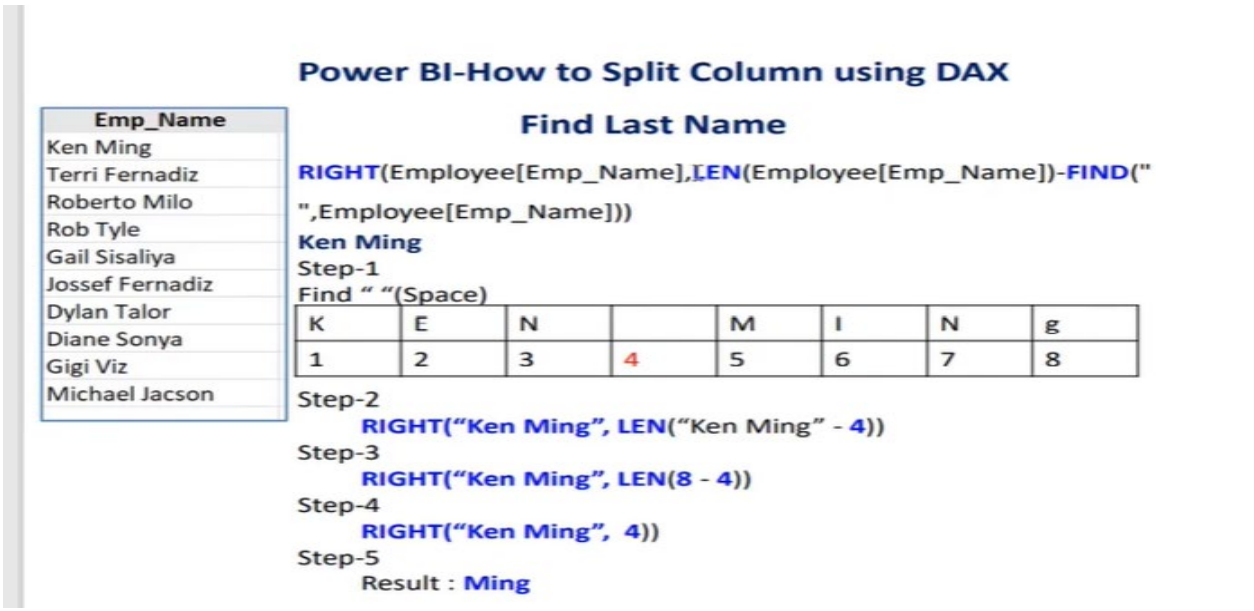
LEFT("**Ken** Ming", **3**)

Step-4

Result : Ken

```
1  First Name = LEFT('Sample file'[Employee Name],FIND(" ",'Sample file'[Employee Name])-1)
```

Employee Name	Date of Birth	Date of Joining	First Name
John Smith	*15 March 1980*	*20 July 2010*	John
Jane Doe	*10 August 1992*	*12 April 2015*	Jane
Michael Brown	*05 December 1975*	*30 September 2008*	Michael
Emily Johnson	*20 June 1988*	*25 March 2012*	Emily
David Lee	*18 February 1990*	*15 October 2017*	David
Sarah Wilson	*30 November 1985*	*05 August 2011*	Sarah
James Davis	*25 April 1982*	*18 November 2009*	James
Olivia Clark	*12 July 1995*	*08 June 2016*	Olivia
William Turner	*28 September 1978*	*10 January 2013*	William
Sophia Harris	*03 October 1989*	*02 May 2014*	Sophia
Robert Evans	*08 January 1987*	*07 December 2018*	Robert
Emma Hall	*27 March 1991*	*14 June 2007*	Emma
Christopher Smith	*14 August 1976*	*20 February 2019*	Christopher
Ava Martin	*22 May 1994*	*22 October 2010*	Ava
Daniel White	*19 July 1983*	*28 November 2015*	Daniel



Employee Name	Date of Birth	Date of Joining	First Name	Last Name
John Smith	15 March 1980	20 July 2010	John	Smith
Jane Doe	10 August 1992	12 April 2015	Jane	Doe
Michael Brown	05 December 1975	30 September 2008	Michael	Brown
Emily Johnson	20 June 1988	25 March 2012	Emily	Johnson
David Lee	18 February 1990	15 October 2017	David	Lee
Sarah Wilson	30 November 1985	05 August 2011	Sarah	Wilson
James Davis	25 April 1982	18 November 2009	James	Davis
Olivia Clark	12 July 1995	08 June 2016	Olivia	Clark
William Turner	28 September 1978	10 January 2013	William	Turner
Sophia Harris	03 October 1989	02 May 2014	Sophia	Harris
Robert Evans	08 January 1987	07 December 2018	Robert	Evans
Emma Hall	27 March 1991	14 June 2007	Emma	Hall
Christopher Smith	14 August 1976	20 February 2019	Christopher	Smith
Ava Martin	22 May 1994	22 October 2010	Ava	Martin
Daniel White	19 July 1983	28 November 2015	Daniel	White

10] Calendar vs CalendarAuto

Power BI DAX Calendar VS Calendarauto

Syntax

CALENDAR(<start_date>, <end_date>)

DAX expression that returns a date time value. Format : yyyy,mm,dd

- Returns a table with a single column named "Date" that contains a contiguous set of the dates.
- Now, mark it as a date table by going to **Table Tools > Mark as Date Table** for format operation.
- An error is returned if start_date is greater than end_date.

Example

```
CALENDAR(Date(2020,1,1), Date(2021,12,1))
```

Example

```
CALENDAR(Date(2021,1,1), Date(2021,1,15))
```

Year, month, day

```
1  Table 2 = CALENDAR(Date(2021,1,1),Date(2021,1,15))
```

Date
01-01-2021 00:00:00
02-01-2021 00:00:00
03-01-2021 00:00:00
04-01-2021 00:00:00
05-01-2021 00:00:00
06-01-2021 00:00:00
07-01-2021 00:00:00
08-01-2021 00:00:00
09-01-2021 00:00:00
10-01-2021 00:00:00
11-01-2021 00:00:00
12-01-2021 00:00:00
13-01-2021 00:00:00
14-01-2021 00:00:00
15-01-2021 00:00:00

Start Date 2021,1,1

End Date 2021,1,15

Structure | Calendars | Relationships | Calculations

```
1  Calendar = CALENDAR(DATE(2023,10,1), DATE(2023,10,15))
```

Date
01-10-2023 00:00:00
02-10-2023 00:00:00
03-10-2023 00:00:00
04-10-2023 00:00:00
05-10-2023 00:00:00
06-10-2023 00:00:00
07-10-2023 00:00:00
08-10-2023 00:00:00
09-10-2023 00:00:00
10-10-2023 00:00:00
11-10-2023 00:00:00
12-10-2023 00:00:00
13-10-2023 00:00:00
14-10-2023 00:00:00
15-10-2023 00:00:00

Power BI DAX Date Function

Syntax

CALENDARAUTO([FiscalYearEndMonth])

Returns a table with one column of dates calculated from the model automatically.

- <FiscalYearEndMonth>Any DAX expression that returns an integer from 1 to 12.
- If you mention the number then it will start from that number to 12 by default.
- If omitted then it will start from minimum value to maximum value in calendar table of current user.

CALENDARAUTO()- returns all dates from Jan-01-2019 to Dec-31-2020
CALENDARAUTO(3)- returns dates from Mar-01-2019 to Feb-28-2021.

Table1 = **CALENDARAUTO()**

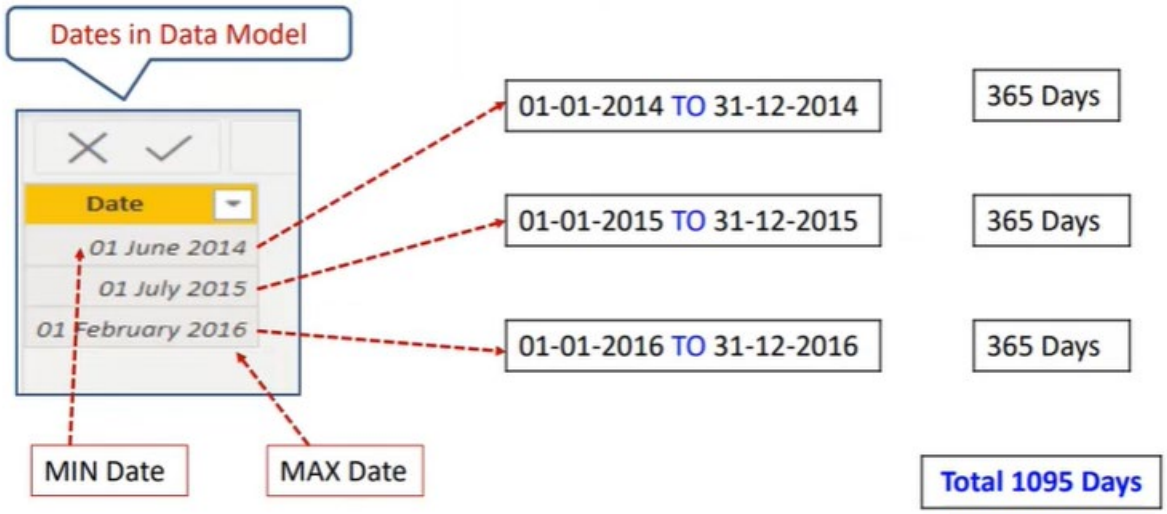
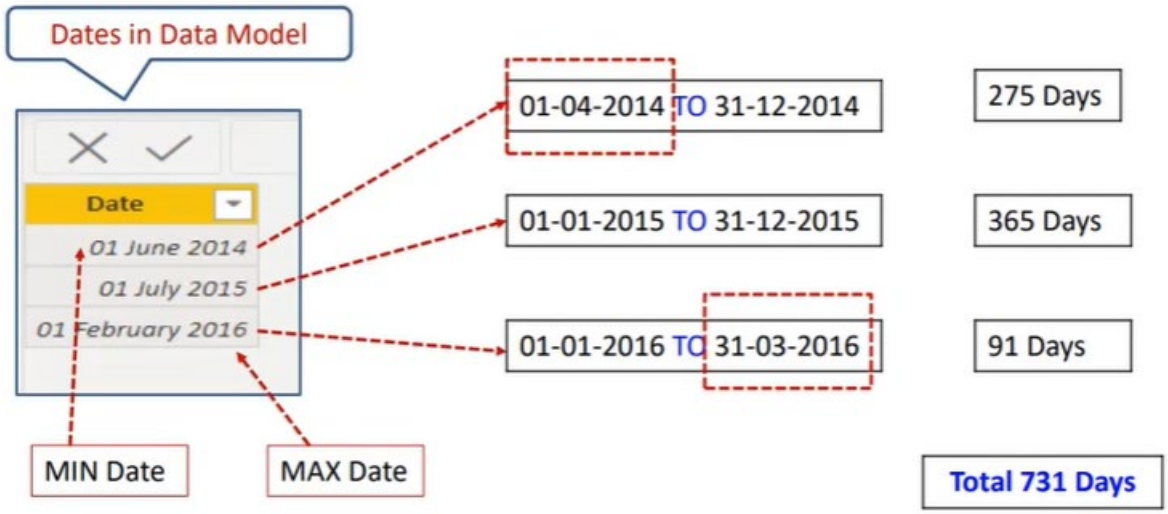
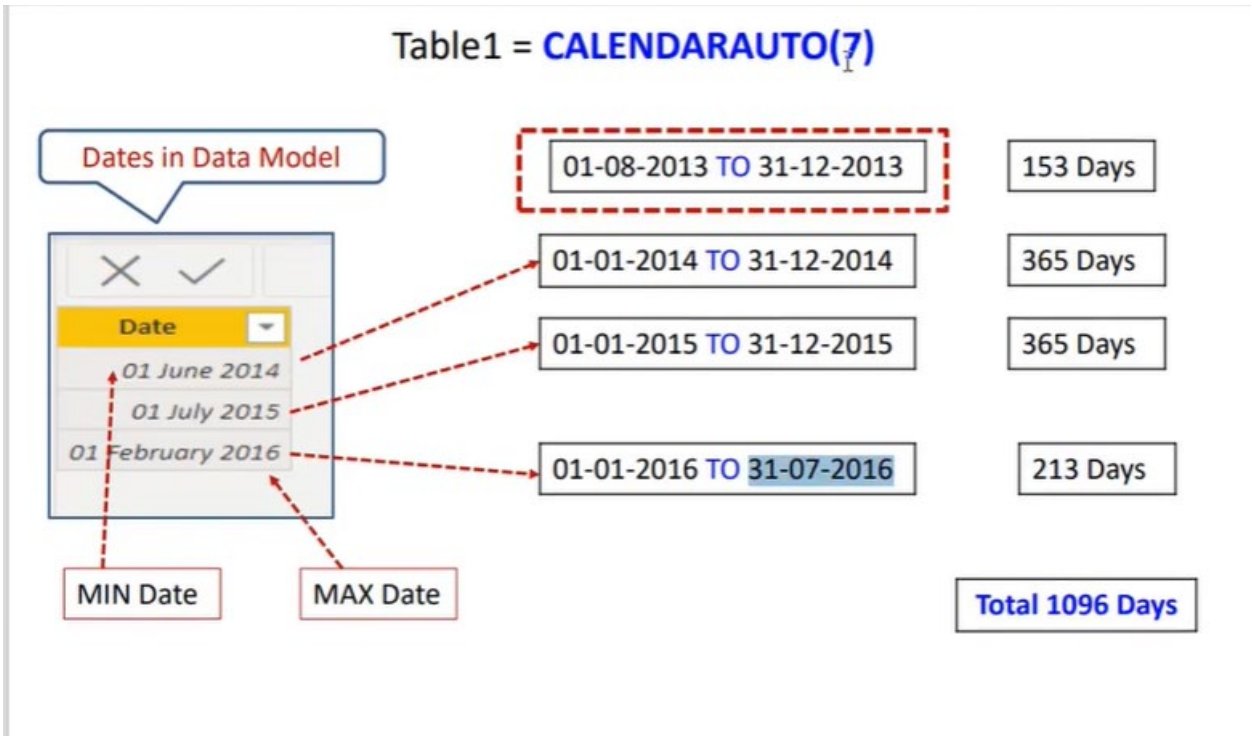


Table1 = **CALENDARAUTO(3)**





1 Calendar Auto = CALENDARAUTO()

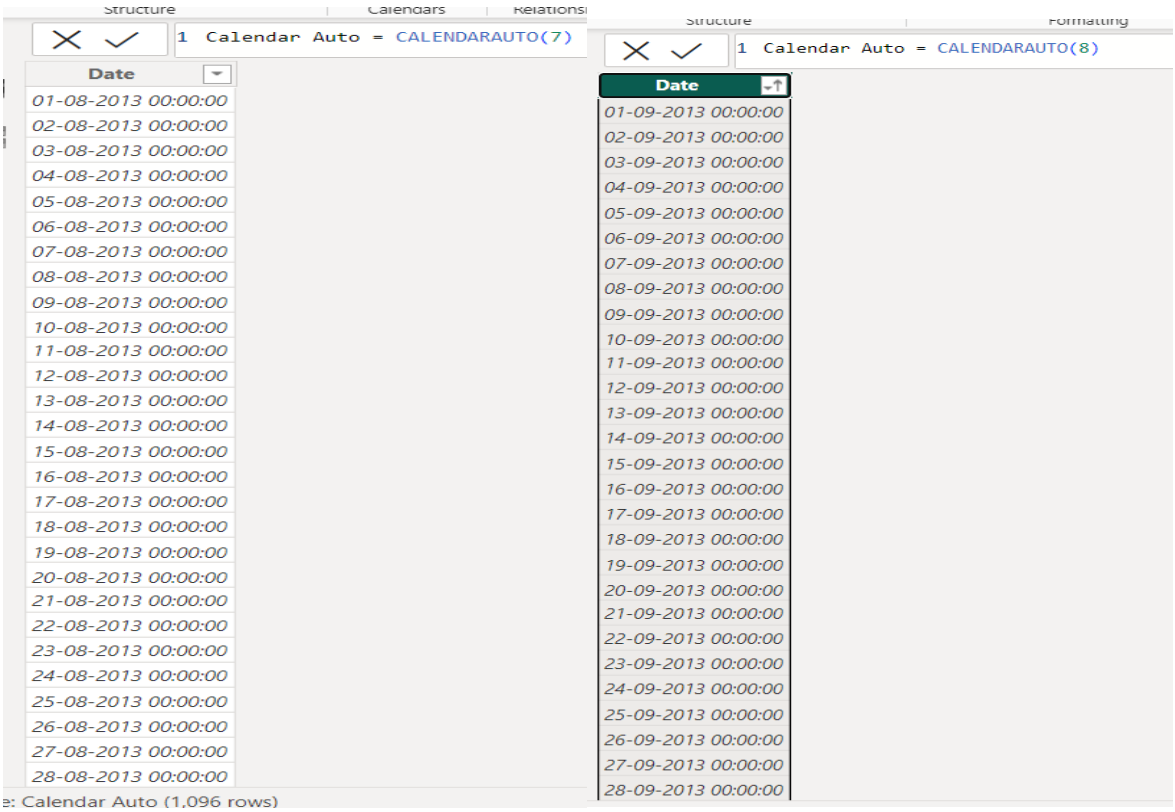
Date
01-01-2014 00:00:00
02-01-2014 00:00:00
03-01-2014 00:00:00
04-01-2014 00:00:00
05-01-2014 00:00:00
06-01-2014 00:00:00
07-01-2014 00:00:00
08-01-2014 00:00:00
09-01-2014 00:00:00
10-01-2014 00:00:00
11-01-2014 00:00:00
12-01-2014 00:00:00
13-01-2014 00:00:00
14-01-2014 00:00:00
15-01-2014 00:00:00
16-01-2014 00:00:00
17-01-2014 00:00:00
18-01-2014 00:00:00
19-01-2014 00:00:00
20-01-2014 00:00:00
21-01-2014 00:00:00
22-01-2014 00:00:00
23-01-2014 00:00:00
24-01-2014 00:00:00
25-01-2014 00:00:00
26-01-2014 00:00:00
27-01-2014 00:00:00
28-01-2014 00:00:00

e: Calendar Auto (1,096 rows)

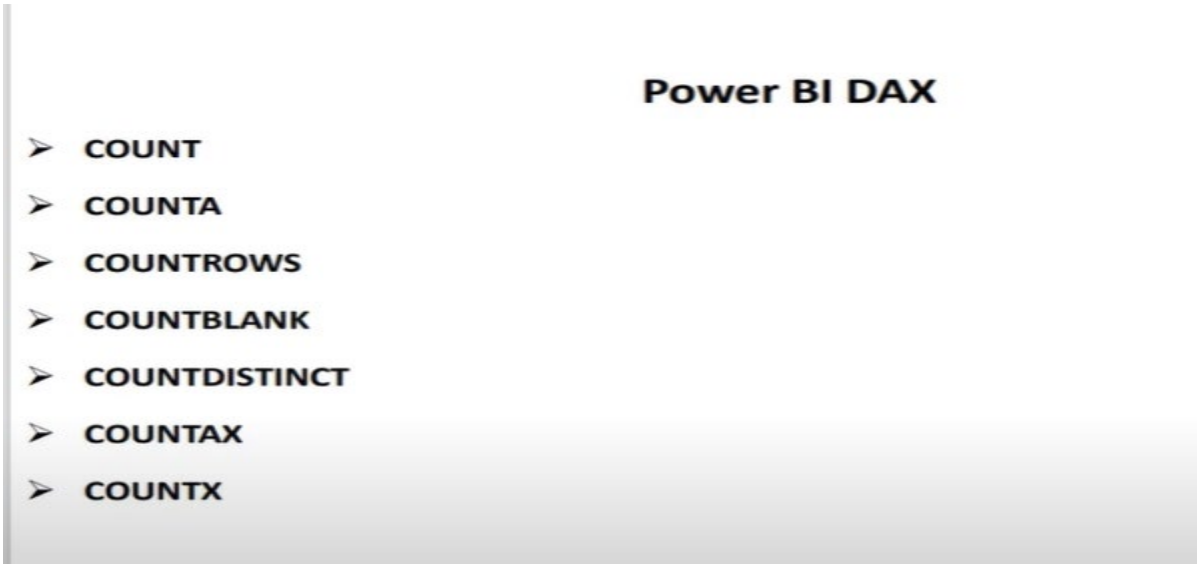
Structure | Calendars | Relationsh

1 Calendar Auto = CALENDARAUTO(3)

Date
01-04-2014 00:00:00
02-04-2014 00:00:00
03-04-2014 00:00:00
04-04-2014 00:00:00
05-04-2014 00:00:00
06-04-2014 00:00:00
07-04-2014 00:00:00
08-04-2014 00:00:00
09-04-2014 00:00:00
10-04-2014 00:00:00
11-04-2014 00:00:00
12-04-2014 00:00:00
13-04-2014 00:00:00
14-04-2014 00:00:00
15-04-2014 00:00:00
16-04-2014 00:00:00
17-04-2014 00:00:00
18-04-2014 00:00:00
19-04-2014 00:00:00
20-04-2014 00:00:00
21-04-2014 00:00:00
22-04-2014 00:00:00
23-04-2014 00:00:00
24-04-2014 00:00:00
25-04-2014 00:00:00
26-04-2014 00:00:00
27-04-2014 00:00:00
28-04-2014 00:00:00



11] Count, Counta, Countblank, Countrows, Countx, Countax



Power BI DAX

➢ COUNT

The COUNT function counts the number of cells in a column that contain non-blank values.

Syntax

COUNT(<column>)

Return value

- A whole number.

- The only argument allowed to this function is a column. The COUNT function counts rows that contain the following kinds of values:
  - Numbers
  - Dates
  - Strings
- When the function finds no rows to count, it returns a blank.

Power BI DAX

➢ COUNTA

The COUNTA function counts the number of cells in a column that are not empty.

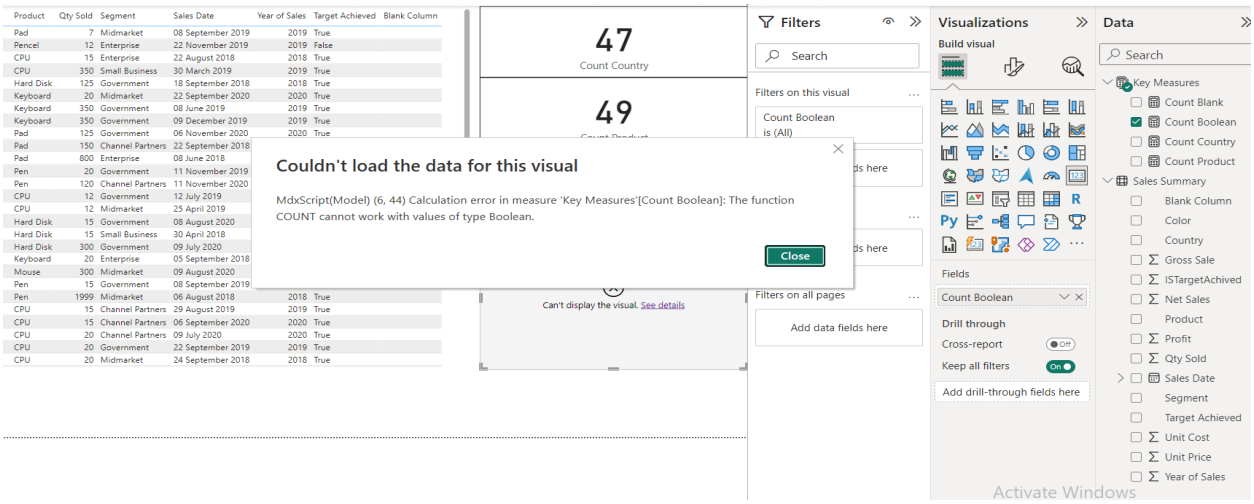
Syntax

COUNTA(<column>)

Return value

- A whole number.

- When the function finds no rows to count, it returns a blank.
- **COUNT** and **COUNTA** are identical in DAX for all the data types except Boolean.
- **COUNTA** can operate on a Boolean data type, whereas **COUNT** cannot do that.



Power BI DAX

➢ COUNTX

- Counts the number of rows that contain a non-blank value or an expression that evaluates to a non-blank value, when evaluating an expression over a table.
- That is, it works just like the COUNT function, but is used to iterate through the rows in a table and count rows where the specified expressions results in a non-blank result.

Syntax

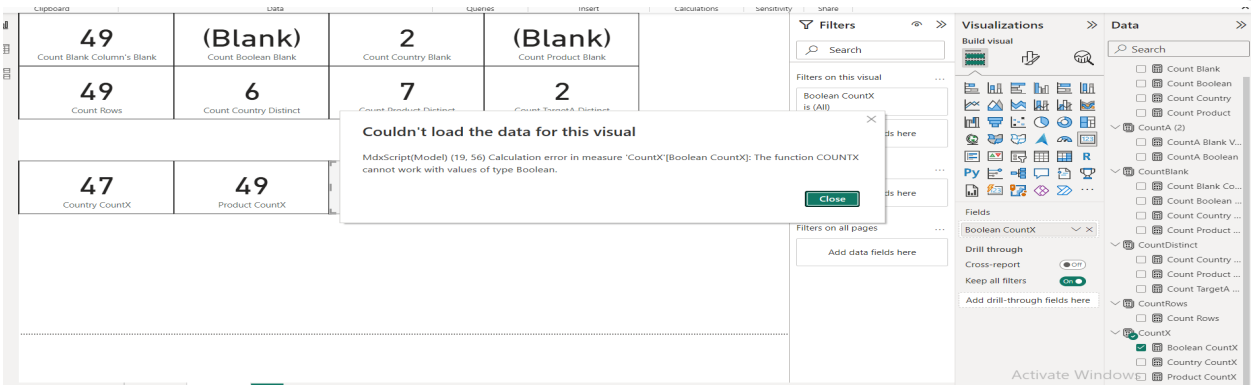
COUNTX(<table><expression>)

Return value

- A whole number.

Remarks

- When the function finds no rows producing a non-blank value, it returns a blank.
- **COUNTX** and **COUNTAX** are identical in DAX for all the data types except Boolean.
- **COUNTAX** can operate on a Boolean data type, whereas **COUNTX** cannot do that.



Count Blank = COUNT('Sales Summary'[Blank Column])

CountA Boolean = COUNTA('Sales Summary'[Target Achieved])

Country CountAX = COUNTAX('Sales Summary','Sales Summary'[Country])

Count Country Blank = COUNTBLANK('Sales Summary'[Country])

Count Country Distinct = DISTINCTCOUNT('Sales Summary'[Country])

Count Rows = COUNTROWS('Sales Summary')

Country CountX = COUNTX('Sales Summary','Sales Summary'[Country])

12] All vs Allselected vs AllExcept

DAX Functions (Filter)

- ALL
- ALLSELECTED
- ALLEXCEPT

DAX ALL Function (Filter)

- Returns all the rows in a table, or all the values in a column, ignoring any filters that might have been applied.
- This function removes the corresponding filters from the filter context, just as **REMOVEFILTERS** does.

Syntax

ALL ([<TableNameOrColumnName>] [, <ColumnName> [, <ColumnName> [, ...]]])

- **TableNameOrColumnName(Optional)**
  - The name of an existing table or column.
- **ColumnName(Optional,Repeatable)**
  - A column in the same base table. The column can be specified in optional parameters only when a column is used in the first argument, too

Return values

- TABLE An entire table or a table with one or more columns.

```
All total Sum = CALCULATE(SUM(Sheet1[Net Sales]), ALL())
AllProductSales = CALCULATE(SUM(Sheet1[Net Sales]),ALL(Sheet1[Product]))
```

All parameters in ALL() are optional.

DAX ALLEXCEPT Function(Filter)

Returns all the rows in a table **except** for those rows that are affected by the specified column filters.

Syntax

ALLEXCEPT (<TableName> <ColumnName> [, <ColumnName> [, ...]])

- **TableName :** The name of an existing table or column.
- **ColumnName(Repeatable) :** A column or a table whose filtering is to be retained when ALLEXCEPT is used as a CALCULATE modifier. The column/table must be part of the expanded table specified in the first parameter.

Return values

- TABLE An entire table or a table with one or more columns.

```
AllExcept Sales = CALCULATE(SUM(Sheet1[Net Sales]), ALLEXCEPT(Sheet1,Sheet1[Country]))
```

DAX ALLSELECTED Function(Filter)

Returns all the rows in a table, or all the values in a column, ignoring any filters that might have been applied inside the query, but keeping filters that come from outside.

Syntax

ALLSELECTED([<TableNameOrColumnName>] [, <ColumnName> [, <ColumnName> [, ...]]])

- **TableNameOrColumnName(Optional)**
  - Remove all filters on the specified table or column applied within the query.
- **ColumnName(Optional, Repeatable)**
  - A column in the same base table.

Return values

- TABLE An entire table or a table with one or more columns.

```
AllSelected Sales = CALCULATE(SUM(Sheet1[Net Sales]), ALLSELECTED(Sheet1[Product]))
```

13] Min vs MinA vs Minx

DAX Functions

- MIN
- MINA
- MINX

MIN-DAX Function(Aggregation)

- Returns the smallest value in a column, or the smaller value between two scalar expressions. **Ignores logical values**. Strings are compared according to alphabetical order.
- **Syntax**:

MIN(<column>) — column in which you want to find the smallest value.

MIN(<expression1>, <expression2>) — Any DAX expression which returns a single value.

Return values

- **SCALAR** A single value of any type.

- The following types of values in the columns are counted:
  - **Numbers, Texts, Dates, Blanks**
- When comparing expressions, blank is treated as 0 when comparing. That is, Min(1,Blank()) returns 0, and Min(-1, Blank()) returns -1.
- If both arguments are blank, MIN returns a blank.
- If you want to evaluate a column of TRUE/FALSE values, use the MINA function.

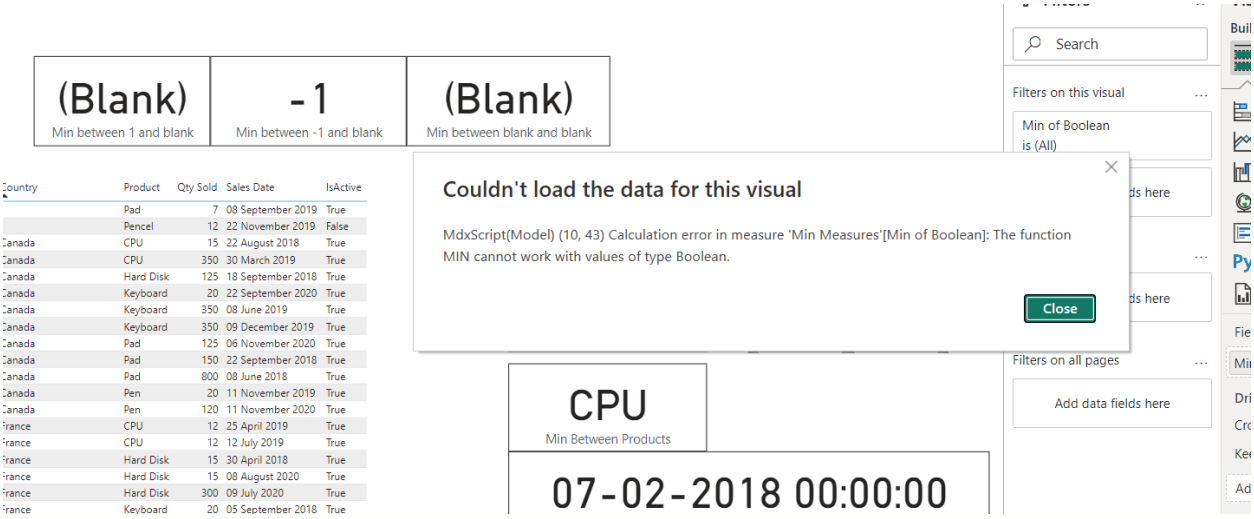
MIN-DAX Function(Aggregation)

Works on Numbers, Texts, Dates, Blanks, ignore logical values(error)

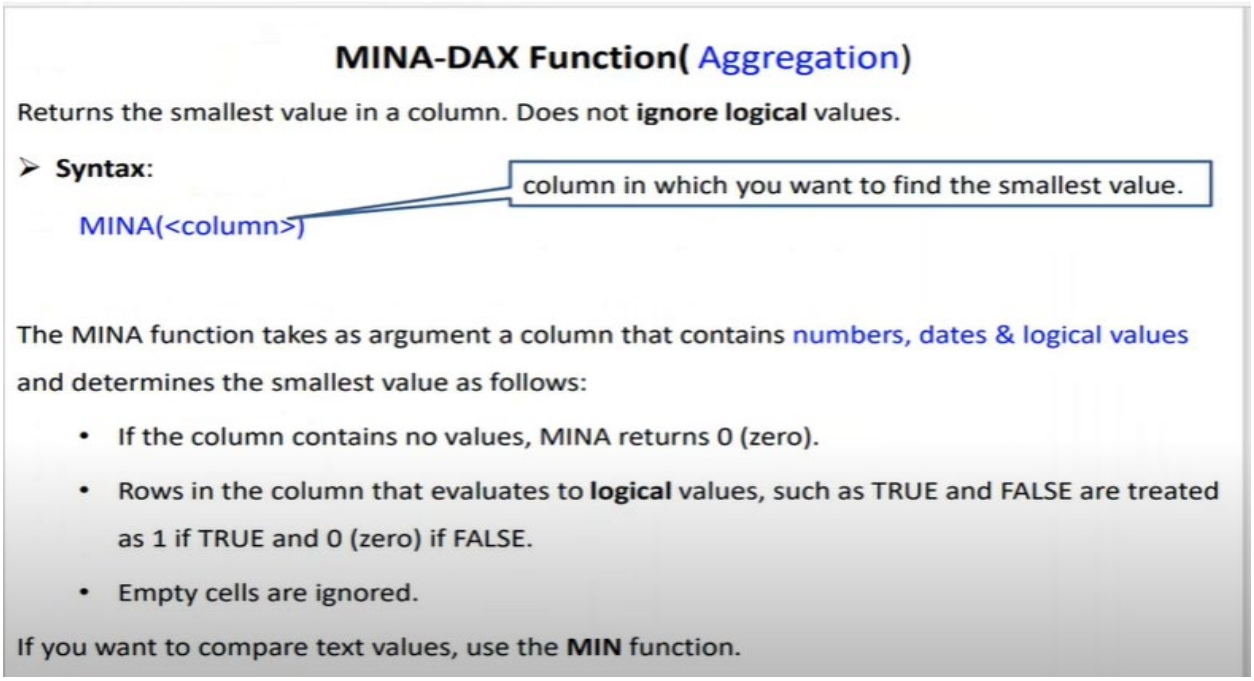
Country	Product	Qty Sold	Year of Sales	Sales Date	Gross Sale	Net Sales	IsActive
Canada	Pen	20	2019	11-11-2019	500.00	400	1
Germany	CPU	15	2019	07-07-2019	432.00	300	1
France	Pencil	1999	2018	06-08-2018	554.00	500	1
Germany	Pad	350	2019	09-02-2019	696.00	200	0
Mexico	Pad	12	2018	21-08-2019	44.00	455	0
Germany	Pencil	7	2020	11-09-2020	789.00	309	0
Germany	Pad	15	2019	22-11-2019	54.00	454	1
Canada	Pen	120	2020	11-11-2020	675.00	145	1
France	Pen	15	2019	08-09-2019	342.00	521	0
Germany	Pencil	45	2018	11-11-2018	568.00	300	0
Mexico	Pencil	22	2020	01-02-2020	432.00	200	1
Germany	Pad	12	2019	22-11-2019	678.00	60	0
Mexico	Mouse	15	2019	11-07-2019	545.00	432	1
Germany	Mouse	20	2019	22-11-2019	555.00	321	0
Canada	Mouse	800	2018	08-06-2018	678.00	654	1
France	Pad	7	2019	08-09-2019	908.00	400	1

```
Min Between Country = MIN('Sales Summary'[Country]) - Canada
Min between -1 and blank = MIN(-1, BLANK()) = -1
Min between 1 and blank = MIN(1, BLANK()) = blank
Min between blank and blank = MIN(BLANK(), BLANK()) = blank
Min Between dates = MIN('Sales Summary'[Sales Date]) = smallest date like  7-2-2018
Min Between Quantity Sold = MIN('Sales Summary'[Qty Sold])= smallest number
Min of Boolean = MIN('Sales Summary'[IsActive]) - doesn't work
```

For boolean values we get this error:



Country	Product	Qty Sold	Sales Date	IsActive
	Pad	7	08 September 2019	True
	Pencel	12	22 November 2019	False
Canada	CPU	15	22 August 2018	True
Canada	CPU	350	30 March 2019	True
Canada	Hard Disk	125	18 September 2018	True
Canada	Keyboard	20	22 September 2020	True
Canada	Keyboard	350	08 June 2019	True
Canada	Keyboard	350	09 December 2019	True
Canada	Pad	125	06 November 2020	True
Canada	Pad	150	22 September 2018	True
Canada	Pad	800	08 June 2018	True
Canada	Pen	20	11 November 2019	True
Canada	Pen	120	11 November 2020	True
France	CPU	12	25 April 2019	True
France	CPU	12	12 July 2019	True
France	Hard Disk	15	30 April 2018	True
France	Hard Disk	15	08 August 2020	True
France	Hard Disk	300	09 July 2020	True
France	Keyboard	20	05 September 2018	True



MINA-DAX Function(Aggregation)

Works Numbers, Dates, Blanks, logical values, ignore text

Country	Product	Qty Sold	Year of Sales	Sales Date	Gross Sale	Net Sales	IsActive
Canada	Pen	20	2019	11-11-2019	500.00	400	1
Germany	CPU	15	2019	07-07-2019	432.00	300	1
France	Pencil	1999	2018	06-08-2018	554.00	500	1
Germany	Pad	350	2019	09-02-2019	696.00	200	0
Mexico	Pad	12	2018	21-08-2019	44.00	455	0
Germany	Pencil	7	2020	11-09-2020	789.00	309	0
Germany	Pad	15	2019	22-11-2019	54.00	454	1
Canada	Pen	120	2020	11-11-2020	675.00	145	1
France	Pen	15	2019	08-09-2019	342.00	521	0
Germany	Pencil	45	2018	11-11-2018	568.00	300	0
Mexico	Pencil	22	2020	01-02-2020	432.00	200	1
Germany	Pad	12	2019	22-11-2019	678.00	60	0
Mexico	Mouse	15	2019	11-07-2019	545.00	432	1
Germany	Mouse	20	2019	22-11-2019	555.00	321	0
Canada	Mouse	800	2018	08-06-2018	678.00	654	1
France	Pad	7	[illegible]	[illegible]	908.00	400	1


```
MinA Boolean = MINA('Sales Summary'[IsActive]) - 0 as 0 is for false
MinA Country = MINA('Sales Summary'[Country]) - doesn't work for text
MinA Dates = MINA('Sales Summary'[Sales Date]) - smallest date in column
MinA Qty Sold = MINA('Sales Summary'[Qty Sold]) - smallest Number in column
```

MINX-DAX Function(Aggregation)

Returns the smallest value that results **from evaluating an expression** for each row of a table.

Strings are compared according to alphabetical order.

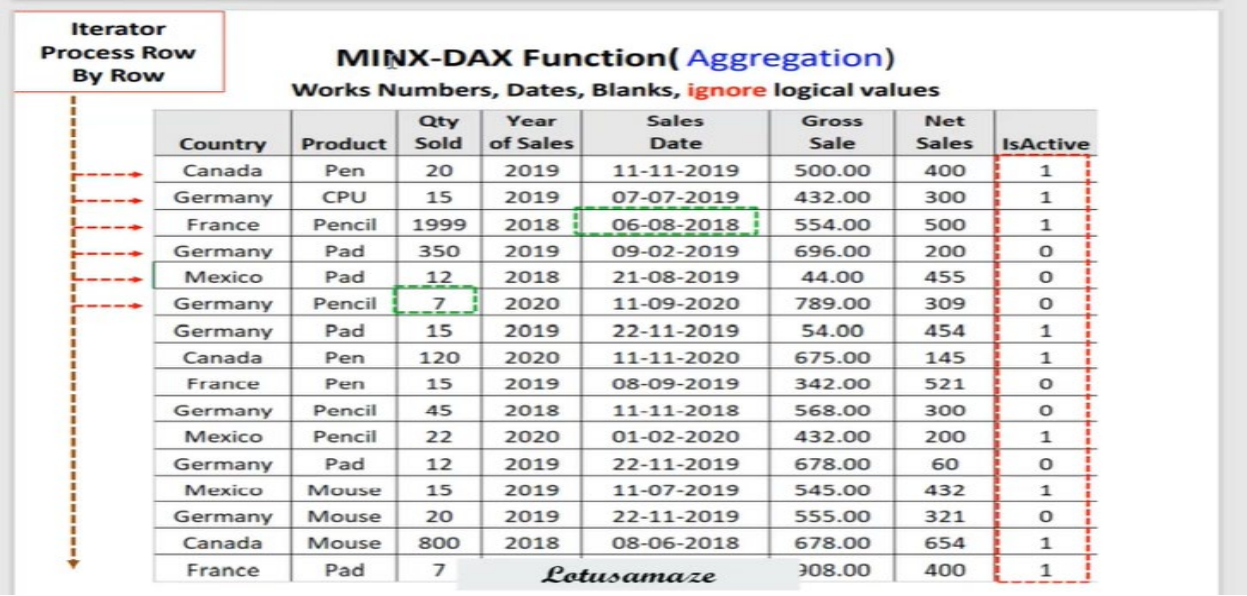
Syntax:

MINX(<table>,<expression>)

Iterator: The table containing the rows for which the expression will be evaluated.

The expression to be evaluated for each row of the table.

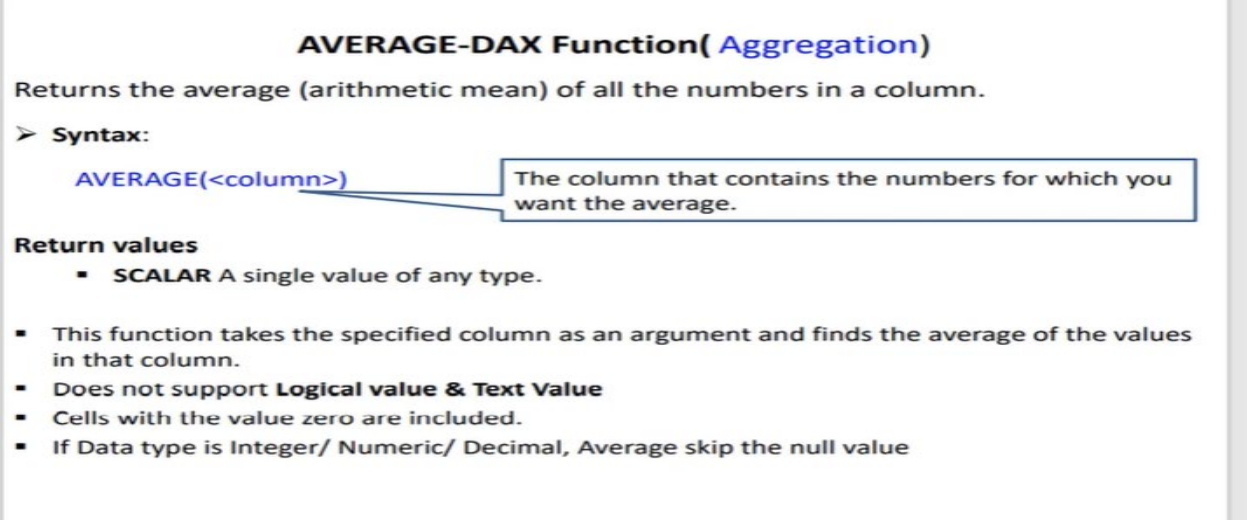
- The **MINX** function takes as its first argument a table, or an expression that returns a table. The second argument contains the expression that is evaluated for each row of the table.
- Blank values are skipped. **TRUE/FALSE values are not supported.**



Country	Product	Qty Sold	Year of Sales	Sales Date	Gross Sale	Net Sales	IsActive
Canada	Pen	20	2019	11-11-2019	500.00	400	1
Germany	CPU	15	2019	07-07-2019	432.00	300	1
France	Pencil	1999	2018	06-08-2018	554.00	500	1
Germany	Pad	350	2019	09-02-2019	696.00	200	0
Mexico	Pad	12	2018	21-08-2019	44.00	455	0
Germany	Pencil	7	2020	11-09-2020	789.00	309	0
Germany	Pad	15	2019	22-11-2019	54.00	454	1
Canada	Pen	120	2020	11-11-2020	675.00	145	1
France	Pen	15	2019	08-09-2019	342.00	521	0
Germany	Pencil	45	2018	11-11-2018	568.00	300	0
Mexico	Pencil	22	2020	01-02-2020	432.00	200	1
Germany	Pad	12	2019	22-11-2019	678.00	60	0
Mexico	Mouse	15	2019	11-07-2019	545.00	432	1
Germany	Mouse	20	2019	22-11-2019	555.00	321	0
Canada	Mouse	800	2018	08-06-2018	678.00	654	1
France	Pad	7			308.00	400	1

```
MinX Qty Sold for Canada = MINX(FILTER('Sales Summary','Sales Summary'[Country] = "Canada"), 'Sales Summary'[Qty Sold])
MinX Product = MINX(FILTER('Sales Summary','Sales Summary'[Qty Sold] > 100), 'Sales Summary'[Product])
```

14] Average vs AverageA vs Averagex



AVERAGEX-DAX Function(Aggregation)

Calculates the average (arithmetic mean) of a set of expressions evaluated over a table.

Syntax:

AVERAGEX(<table>,<expression>)

Iterator: The table containing the rows for which the expression will be evaluated.

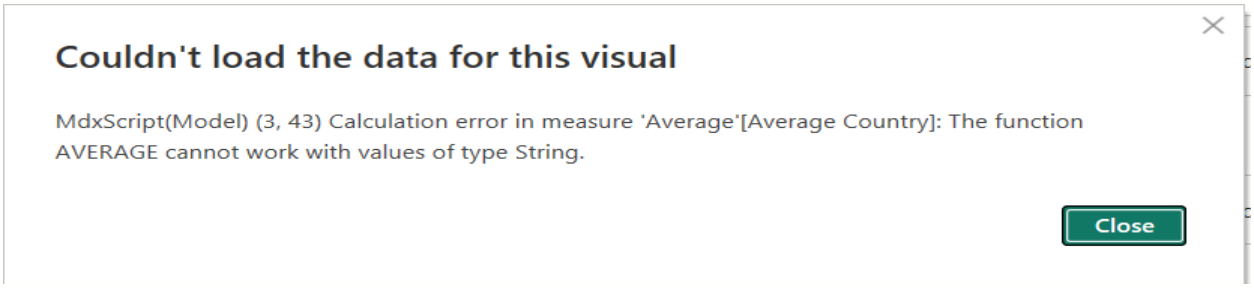
The expression to be evaluated for each row of the table.

- The function takes a table as its first argument, and an expression as the second argument.
- 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.
- You cannot include non-numeric or null cells. Both the table and expression arguments are required.
- When there are no rows to aggregate, the function returns a blank. When there are rows, but none of them meet the specified criteria, then the function returns 0.
- Does not support Logical/ boolean values

1]Average:

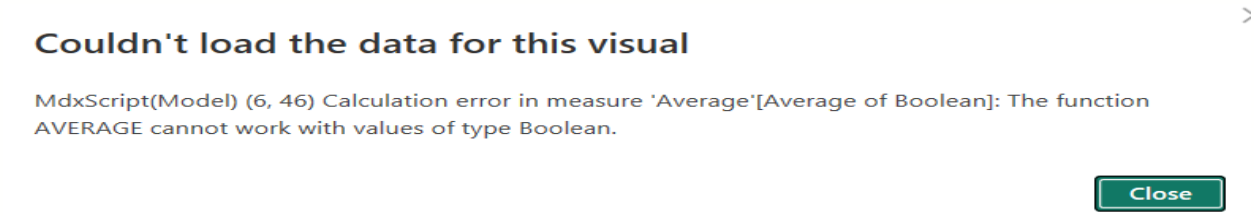
```
Average Country = AVERAGE('Sales Summary'[Country])
```

Doesn't work, get error as :




```
1] Average QTY Sold = AVERAGE('Sales Summary'[Qty Sold])- works properly
2] Average Sales Date = AVERAGE('Sales Summary'[Sales Date]) - works but not an
correct way to calculate as it convert dae in numbers and then adds them.
3] Average of Boolean = AVERAGE('Sales Summary'[IsActive]) - doesn't work
```

Doesn't work, get error as :



2] AverageA

1] `AverageA Country = AVERAGEA('Sales Summary'[Country])- answer is 0`

2] `AverageA QTy Sold = AVERAGEA('Sales Summary'[Qty Sold]) - correct answer`

3] `AverageA Dates = AVERAGEA('Sales Summary'[Sales Date]) - same as average`

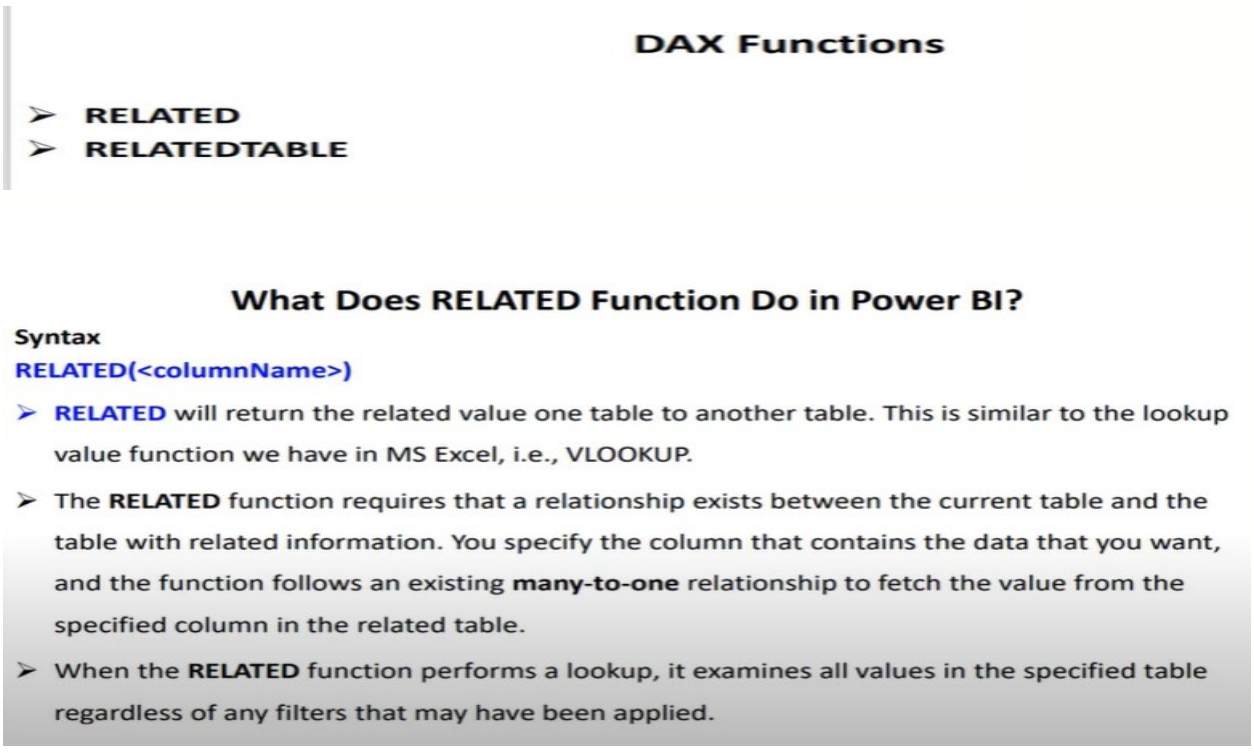
4] `AverageA of Boolean = AVERAGEA('Sales Summary'[IsActive]) - count all 1's and then divide by total rows`

2] AverageX

1]
```
AverageX Qty Sold for Canada = AVERAGEX(FILTER('Sales Summary','Sales Summary'[Country] = "Canada"), 'Sales Summary'[Qty Sold])
```

2]
```
AverageX Qty Sold for Mouse = AVERAGEX(FILTER('Sales Summary','Sales Summary'[Product] = "Mouse"), 'Sales Summary'[Qty Sold])
```

15] Related vs Relatedtable



DAX RELATED Function

One-to-many relationship

Table: Product
Unique Records product column

Product	Price
CPU	89
Mouse	789
Pencil	234
Pad	321
Keyboard	789
Monitor	322

Table: Sales
Many Records product column

Product	Qty Sold	Sold By
Pen	20	John
CPU	15	Smith
Pencil	1999	Peter
Pad	350	Roger
Pad	12	Lienda
Pencil	7	Smith
Pad	15	Susan
Keyboard	120	Susan
Pen	15	Roger
Monitor	45	Peter
Pencil	22	Smith
Keyboard	12	Smith
Mouse	15	Susan
Monitor	20	Roger
Mouse	800	Lienda
Pad	7	Moorn

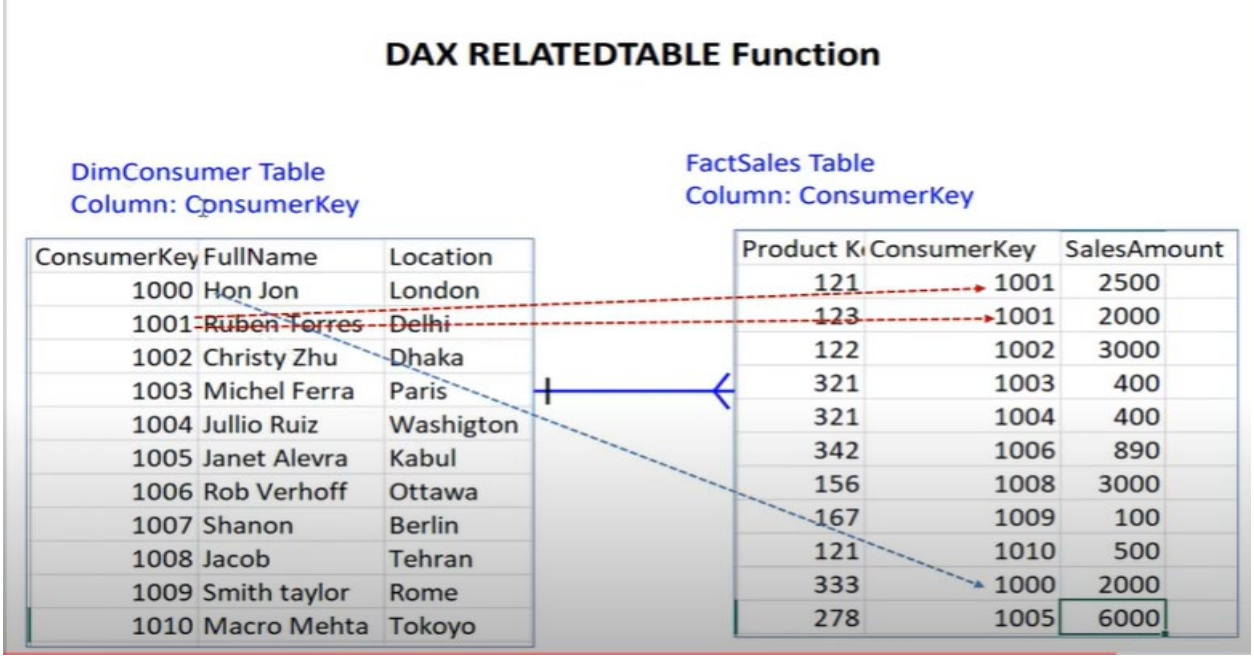
```
Total Sales = 'Sales Summary'[Qty Sold] * RELATED('Product'[Price])
Price = RELATED('Product'[Price])
```

What Does RELATEDTABLE Function Do in Power BI?

- Similar to **RELATED** function, the **RELATEDTABLE** function also works with the relationship.
- The **RELATEDTABLE** returns a table as the output.
- That makes the **RELATEDTABLE** function a tabular function.
- Returns the related tables filtered so that it only includes the related rows.
- The returned table is a sub-table of the given table for all records that match the key in the current row of the current table. This is how the function works;
- This function is a shortcut for CALCULATETABLE function with no logical expression.

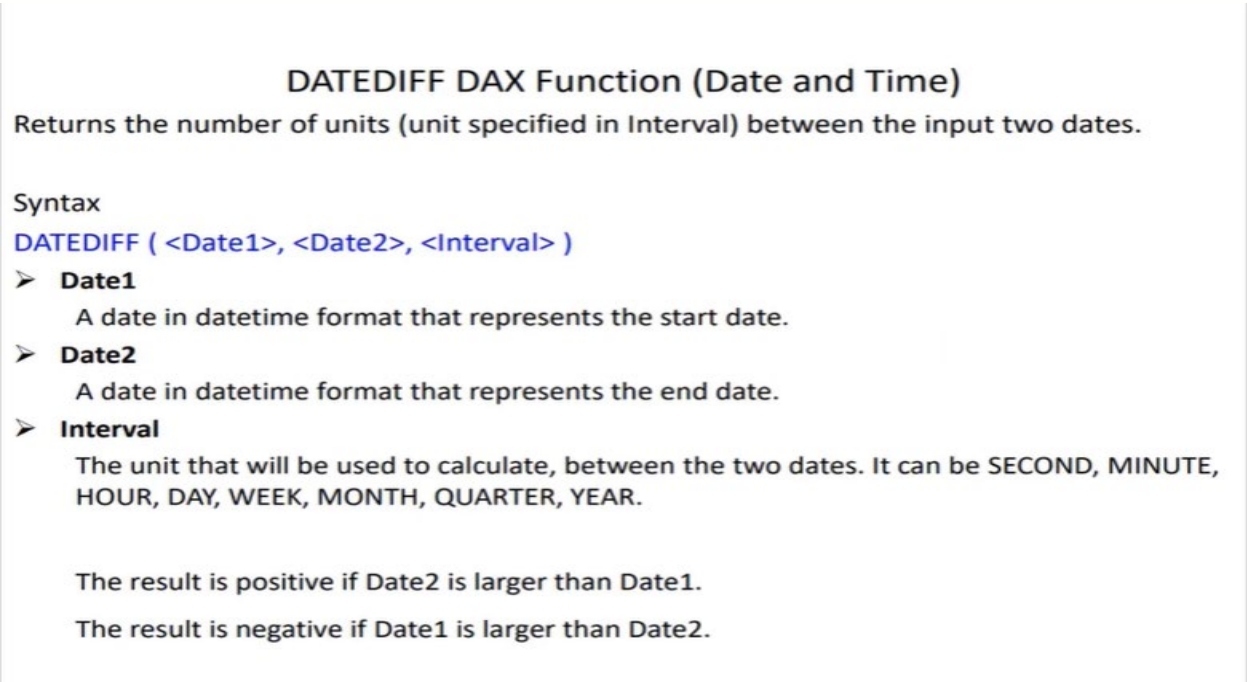
Syntax

RELATEDTABLE(<table>)




```
Total Sales = SUMX(RELATEDTABLE(FactSales),FactSales[SalesAmount])
```

16] DateDiff




```
Age = DATEDIFF('Sales Summary'[BirthDate], TODAY(), YEAR)
Service Period = DATEDIFF('Sales Summary'[Joining Date],TODAY(),YEAR)
Bonus Eligibility = IF(DATEDIFF('Sales Summary'[Joining
Date],TODAY(),YEAR)>4,"Eligible","Not Eligible")
```

16.1] DatesBetween vs DatesInPeriod

DATESBETWEEN DAX Function (Date and Time)

Returns the dates between two given dates.

Syntax

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

Return values: TABLE A table with a single column.

dates : A column reference containing dates.

start_date : A date in datetime format that represents the start date.

end_date : A date in datetime format that represents the end date.

- If StartDate is a blank date value, then StartDate will be the earliest value in the dates column.
- If EndDate is a blank date value, then EndDate will be the latest value in the dates column.
- The dates used as the StartDate and EndDate are inclusive.
- If StartDate is larger than EndDate, the result is an empty table.

```
QTY Sold for Sept = CALCULATE(SUM('Sales Summary'[Qty Sold]), DATESBETWEEN('Sales Summary'[Sales Date],DATE(2019,9,1),DATE(2019,9,30)))
```

DATESINPERIOD DAX Function (Date and Time)

- Returns the dates from the given period.
- **DATESINPERIOD** is suitable if you have regular intervals in your date column.

Syntax

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

Return values: TABLE A table with a single column.

- **Dates :** A column reference containing dates.
- **StartDate** : Start date.
- **NumberOfIntervals:** The number of intervals.
- **Interval** : One of: Day, Month, Quarter, Year.

```
1] Qty sold for first 15 day of sep = CALCULATE(SUM('Sales Summary'[Qty Sold]),DATESINPERIOD('Sales Summary'[Sales Date],DATE(2019,9,1),15,DAY)
```

```
2] Qty sold for month of sep = CALCULATE(SUM('Sales Summary'[Qty
Sold]),DATESINPERIOD('Sales Summary'[Sales Date],DATE(2019,9,1),1,MONTH))
3] Qty sold for Whole Year = CALCULATE(SUM('Sales Summary'[Qty
Sold]),DATESINPERIOD('Sales Summary'[Sales Date],DATE(2019,9,1),1,YEAR))
4] Similar for Quarter.
```

16.2]Datesmtd Vs Datesqtd Vs Datesytd

DATESMTD DAX Function (Time Intelligence)

- Returns a set of dates in the month up to the **last date visible** in the filter context.
- The **DATESMTD** function can be used to calculate the **running total** for the month for the given day number.

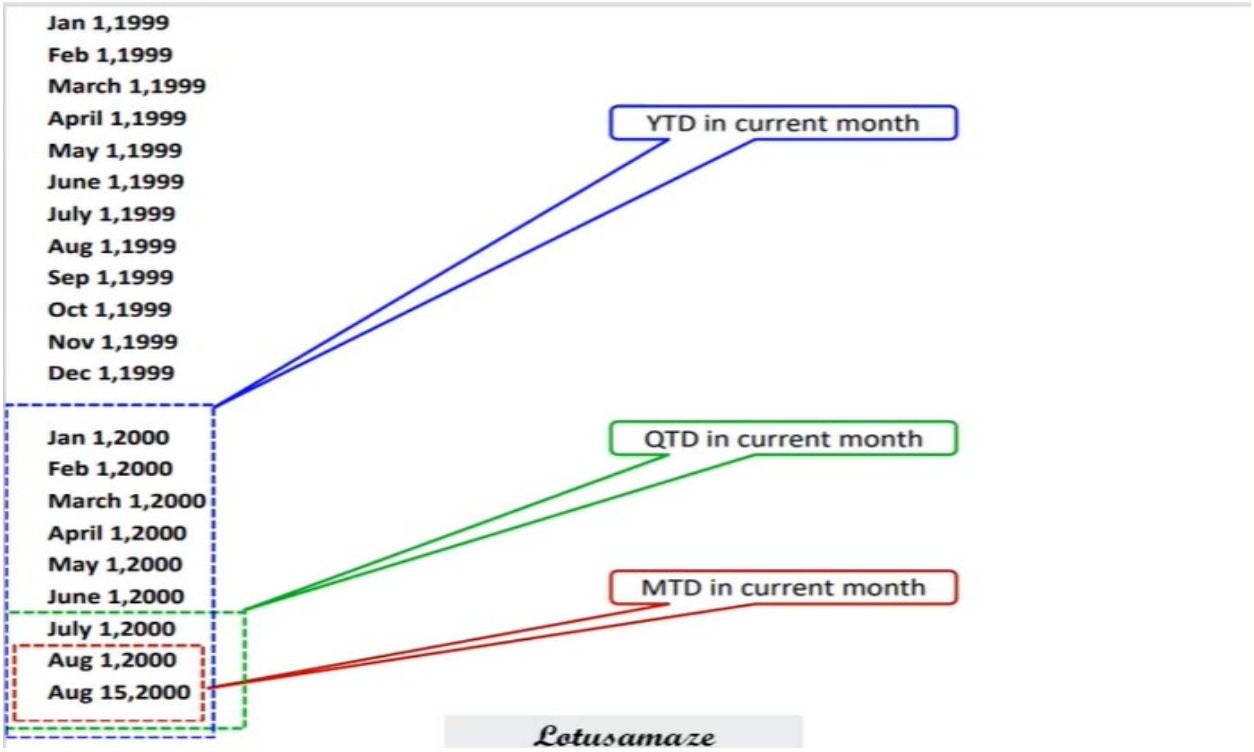
Syntax

DATESMTD(<dates)

- **Dates:** The name of a column containing dates or a one column table containing dates.

Return values

- TABLE A table with a single column.



DATESQTD DAX Function (Time Intelligence)

➢ Returns a set of dates in the quarter up to the **last date visible** in the filter context.

Syntax

DATESQTD(<dates)

- **Dates:** The name of a column containing dates or a one column table containing dates.

Return values

- TABLE A table with a single column.

DATESYTD DAX Function (Time Intelligence)

➢ Returns a set of dates in the **year up to the last date visible** in the filter context.

➢ **Syntax**

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

- **Dates:** The name of a column containing dates or a one column table containing dates.
- **YearEndDate(Optional) :** End of year date.

Return values

- TABLE A table with a single column.

```
1] DateMTD = CALCULATE(SUM('Sales Summary'[Qty Sold]),DATESMTD('Sales Summary'[Sales Date])) - Calculates for Last visible month like(dec 2020)
2] DateQTD = CALCULATE(SUM('Sales Summary'[Qty Sold]),DATESQTD('Sales Summary'[Sales Date])) - Calculates for Last visible quarter like(oct, nov, dec 2020)
3] DateYTD = CALCULATE(SUM('Sales Summary'[Qty Sold]),DATESYTD('Sales Summary'[Sales Date]))- Calculates for Last visible Year like(2020)
```

16.3] Endofmonth, Endofquarter And Endofyear

ENDOFMONTH DAX Function (Time Intelligence)

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

Syntax

ENDOFMONTH(<dates>)

- **Dates:** The name of a column containing dates or a one column table containing dates.

Return values

- TABLE A table with a single column.

ENDOFQUARTER DAX Function (**Time Intelligence**)

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

Syntax

ENDOFQUARTER(<dates>)

- **Dates:** The name of a column containing dates or a one column table containing dates.

Return values

- TABLE A table with a single column.

ENDOFYEAR DAX Function (**Time Intelligence**)

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

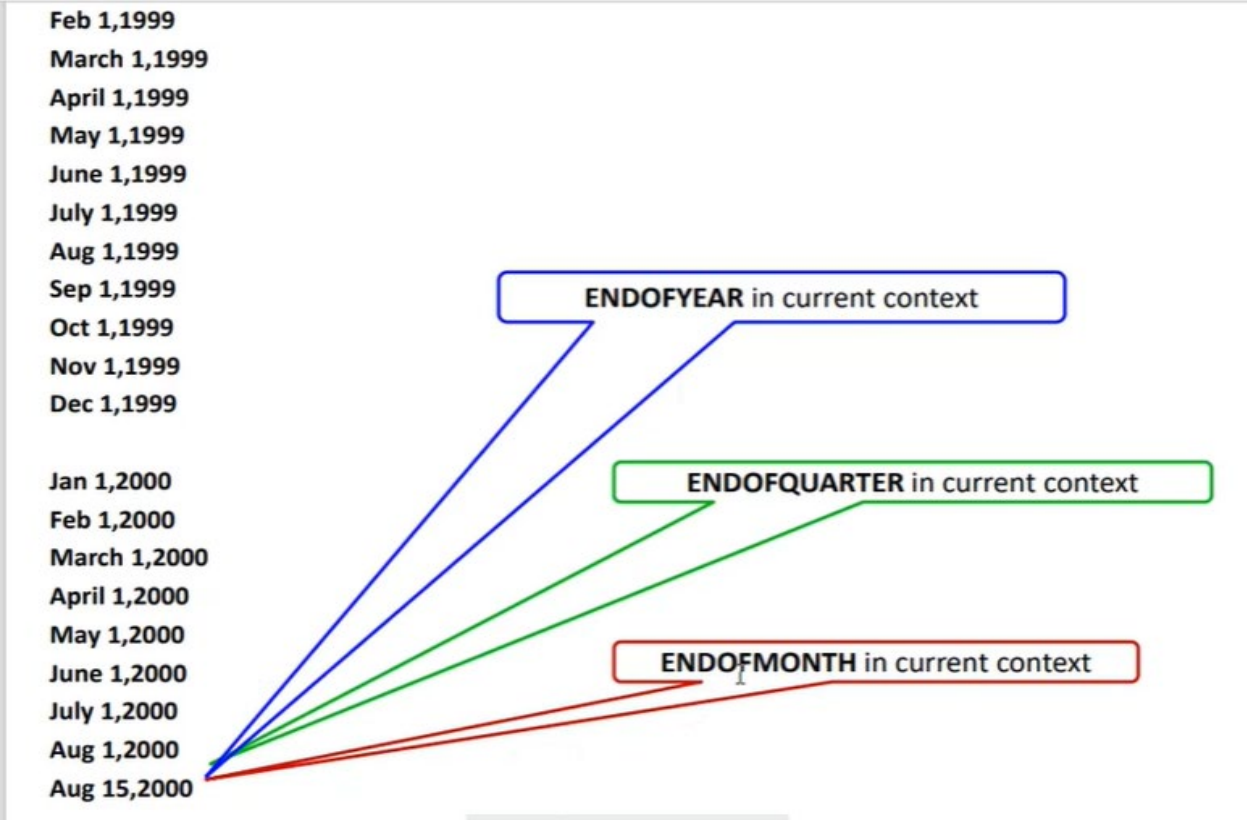
Syntax

ENDOFYEAR(<dates>)

- **Dates:** The name of a column containing dates or a one column table containing dates.

Return values

- TABLE A table with a single column.



All Measures will be calculated for date which is at last in data like 19 dec 2022

```
1] QTY Sold for ENDOFMONTH = CALCULATE(SUM('Country (Sales Summary)'[Qty Sold]),ENDOFMONTH('Country (Sales Summary)'[Sales Date])
2] QTY Sold for ENDOFQUARTER = CALCULATE(SUM('Country (Sales Summary)'[Qty Sold]),ENDOFQUARTER('Country (Sales Summary)'[Sales Date]))
3] QTY Sold for ENDOFYEAR = CALCULATE(SUM('Country (Sales Summary)'[Qty Sold]),ENDOFYEAR('Country (Sales Summary)'[Sales Date]))
```

16.4] TotalMTD, TotalQTD, TotalYTD

TOTALMTD DAX Function (Time Intelligence)

- Evaluates the value of the expression for the month to date, in the current context.
- TOTALMTD function, which calculates the cumulative subtotals in a month.

Syntax

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

- **Expression** : An expression that returns a scalar value.
- **Dates** : The name of a column containing dates or a one column table containing dates.
- **Filter(Optional)** : A boolean (True/False) expression or a table expression that defines a filter.

➢ **Return values**

SCALAR A single value of any type

TOTALQTD DAX Function (Time Intelligence)

- Evaluates the value of the **expression** for the dates in the quarter to date, in the current context.
- TOTALQTD function, which calculates the cumulative subtotals in a quarter.

Syntax

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

- **Expression** : An expression that returns a scalar value.
- **Dates** : The name of a column containing dates or a one column table containing dates.
- **Filter(Optional)** : A boolean (True/False) expression or a table expression that defines a filter.

Syntax

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

- **Expression** : An expression that returns a scalar value.
- **Dates** : The name of a column containing dates or a one column table containing dates.
- **Filter(Optional)** : A boolean (True/False) expression or a table expression that defines a filter.

➢ **Return values**

SCALAR A single value of any type

TOTALYTD DAX Function (**Time Intelligence**)

- Evaluates the year-to-date value of the **expression** in the current context.
- TOTALYTD function, which calculates the cumulative subtotals in a year.

Syntax

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

- **Expression** : An expression that returns a scalar value.
- **Dates** : The name of a column containing dates or a one column table containing dates.
- **Filter(Optional)** : A boolean (True/False) expression or a table expression that defines a filter.
- **year_end_date(optional) :** A literal string with a date that defines the year-end date. The default is December 31.

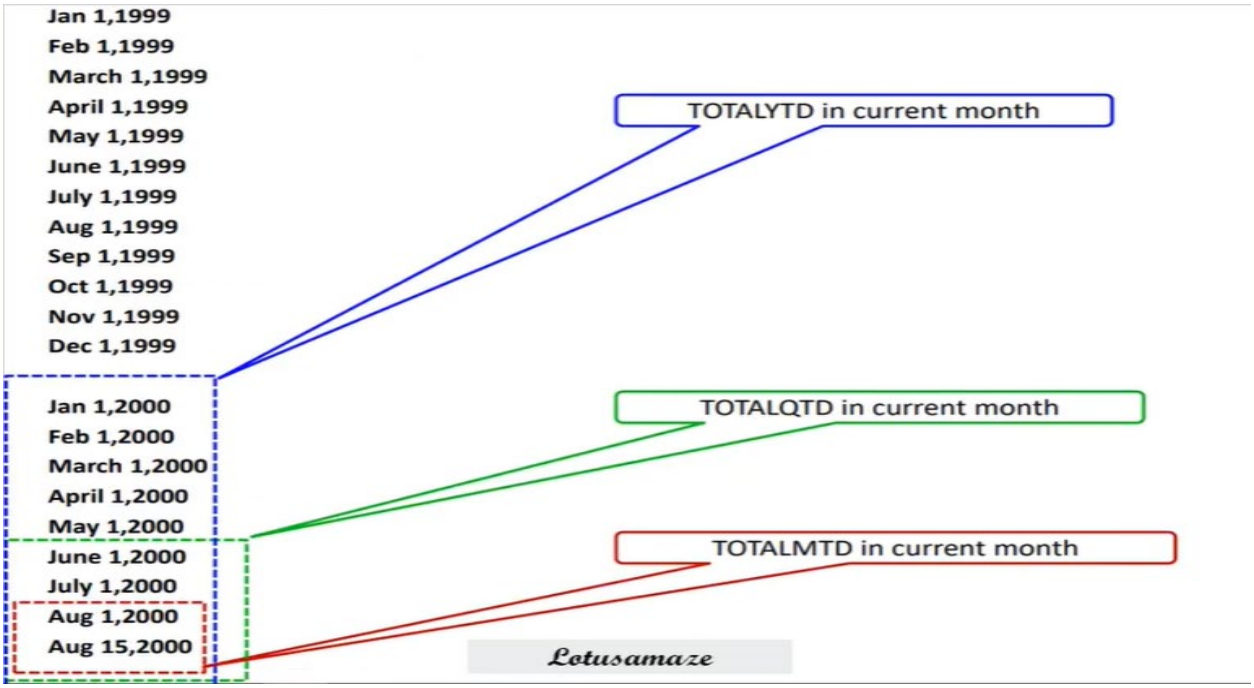
Syntax

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

- **Expression** : An expression that returns a scalar value.
- **Dates** : The name of a column containing dates or a one column table containing dates.
- **Filter(Optional)** : A boolean (True/False) expression or a table expression that defines a filter.
- **year_end_date(optional) :** A literal string with a date that defines the year-end date. The default is December 31.

➢ **Return values**

SCALAR A single value of any ty




```
1] Total_MTD for QTY Sold = TOTALMTD(SUM('Country (Sales Summary)'[Qty Sold]),'Country (Sales Summary)'[Sales Date])
2] Total_MTD-QTY Sold(Pen) = TOTALMTD(SUM('Country (Sales Summary)'[Qty Sold]),'Country (Sales Summary)'[Sales Date], 'Country (Sales Summary)'[Product] = "Pen")
3] Total_QTD for QTY Sold = TOTALQTD(SUM('Country (Sales Summary)'[Qty Sold]),'Country (Sales Summary)'[Sales Date])
4] Total_QTD-QTY Sold(Mouse) = TOTALQTD(SUM('Country (Sales Summary)'[Qty Sold]),'Country (Sales Summary)'[Sales Date], 'Country (Sales Summary)'[Product] = "Mouse")
5] Total_YTD for QTY Sold = TOTALYTD(SUM('Country (Sales Summary)'[Qty Sold]),'Country (Sales Summary)'[Sales Date])
6] Total_YTD-QTY Sold(Pencil) = TOTALYTD(SUM('Country (Sales Summary)'[Qty Sold]),'Country (Sales Summary)'[Sales Date], 'Country (Sales Summary)'[Product] = "Pencil")
7] Total_YTD-QTY Sold(Pencil) for march = TOTALYTD(SUM('Country (Sales Summary)'[Qty Sold]),'Country (Sales Summary)'[Sales Date], 'Country (Sales Summary)'[Product] = "Pencil","3/31")
```

17] CrossJoin

CROSSJOIN

DAX Function (Table manipulation)

Returns a table that is a crossjoin of the specified tables.

Syntax

CROSSJOIN(<table>, <table>[, <table>]...)

- **Table(Repeatable)**
  A table that will participate in the crossjoin.
- **Return values**
  TABLE An entire table or a table with one or more columns.

- Column names from table arguments must all be different in all tables or an error is returned.
- The total number of rows returned by CROSSJOIN() is equal to the product of the number of rows from all tables in the arguments; also,
- the total number of columns in the result table is the sum of the number of columns in all tables.

CASE STUDY-CROSSJOIN

- ABC Limited has six branches across USA and sells 6 products.
- Each product has same target across all the branches.
- You are to generate a new table that will combine these two tables into one — having all products with assigned targets showing for all the branches.

Branch Table

Branch	Manager
Washinton	Jacab
Athens	Mario
Atmore	Smith
Palmer	Junto
Cordova	Leenda
Mesa	Mike

Product Table

Product	Sales(Target)
Hard Disk	200
CPU	300
Pencil	1200
Pad	350
Keyboard	2000
Mouse	2100

Result=Total rows in Branch Table * Total rows in product table
Result = 6*6=36

Lotusamaze

```
1] Branch and Product = CROSSJOIN('Branch Table','Product Table') - Total rows after execution 36(6*6).
2] Branch Product  and Category = CROSSJOIN('Branch Table','Product Table','Category') - Total rows after execution 72(36*2).
```

18] Union, Intersect and Except(Table Manipulation)

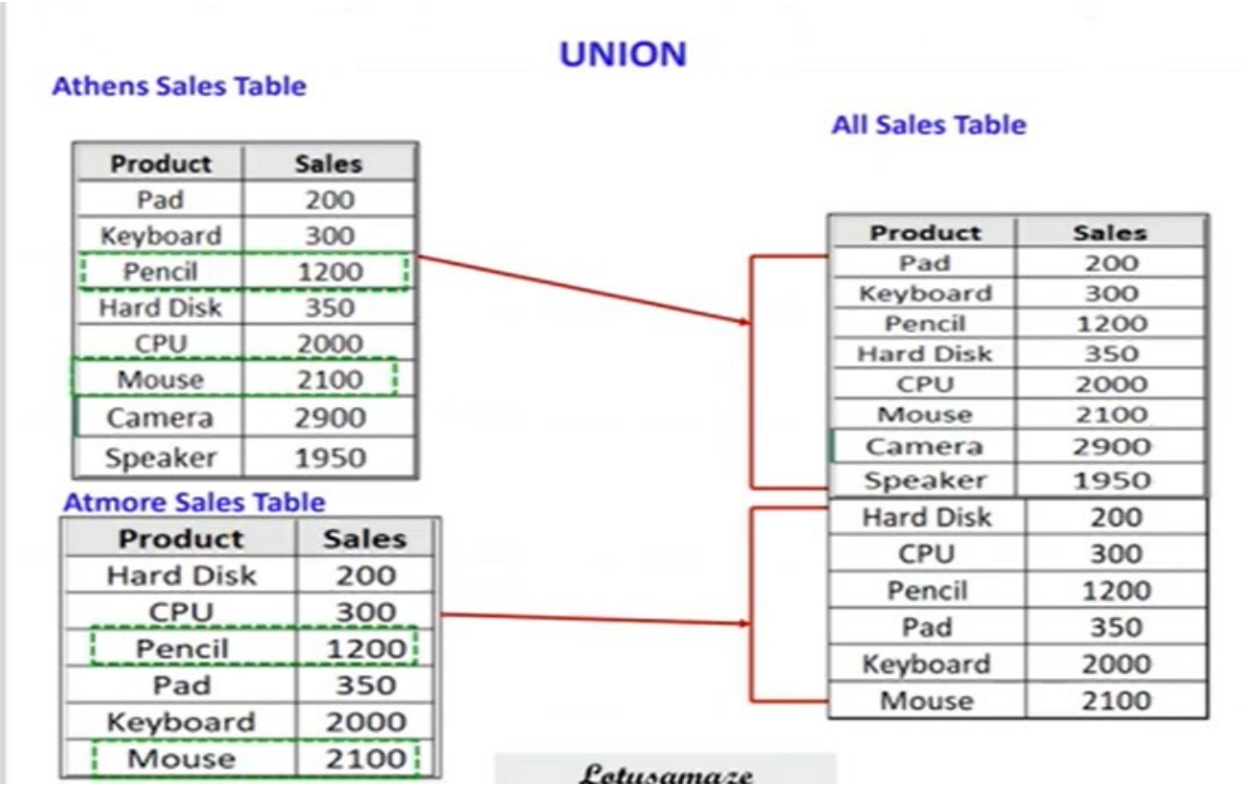
UNION DAX Function (Table manipulation)

Returns the union of the tables whose columns match.

UNION(<table_expression1>, <table_expression2> [,<table_expression>]...)

- **Table(Repeatable)**
  A table that will participate in the union.

- The tables must have the same number of columns.
- Columns are combined by position in their respective tables.
- The column names in the return table will match the column names in the first Table.
- Duplicate rows are retained.




```
Union of Tables = UNION('Athens Sales','Atmore Sales')
```

INTERSECT DAX Function (Table manipulation)

Returns the rows of left-side table which appear in right-side table.

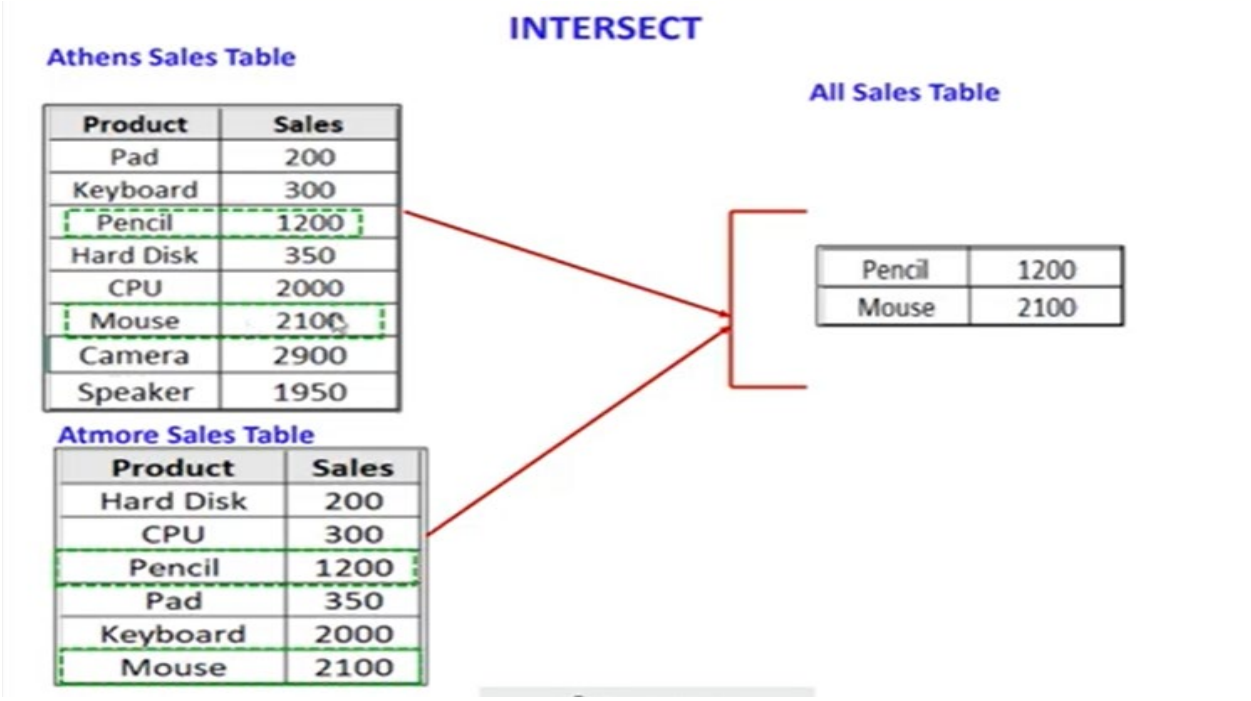
INTERSECT(<left_table>, <right_table>)

- **left_Table & right_table**
  A table that will participate in the intersect.

- **Return values**

  TABLE An entire table or a table with one or more columns.

- A table that contains all the rows in LeftTable that are also in RightTable.
- Duplicate rows are retained.




```
Intersect of Tables = INTERSECT('Athens Sales','Atmore Sales')
```

EXCEPT DAX Function (Table manipulation)

Returns the rows of left-side table which do not appear in right-side table.

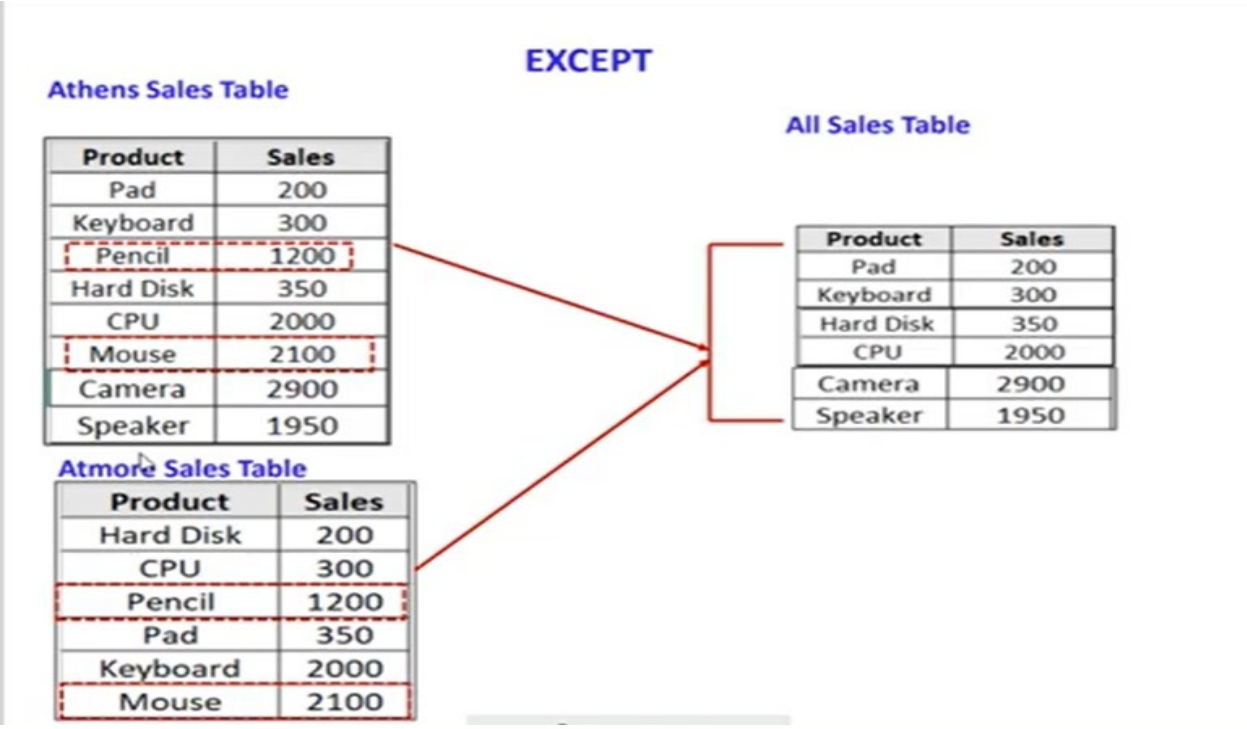
EXCEPT(<left_table>, <right_table>)

- **left_Table & right_table**
  A table that will participate in the except.

- **Return values**

  TABLE An entire table or a table with one or more columns.

- If a row appears at all in both tables, it and its duplicates are not present in the result set. If a row appears in only LeftTable, it and its duplicates will appear in the result set.
- The column names will match the column names in LeftTable.

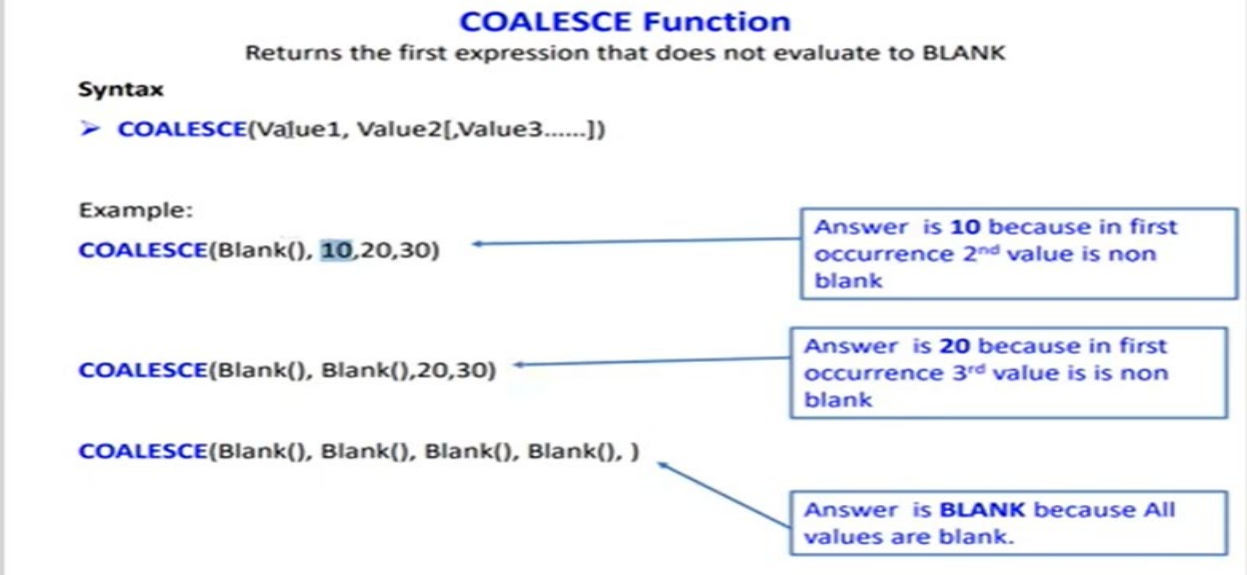



```
Except of Tables = EXCEPT('Athens Sales','Atmore Sales')
```

19] Coalesce(Logical)

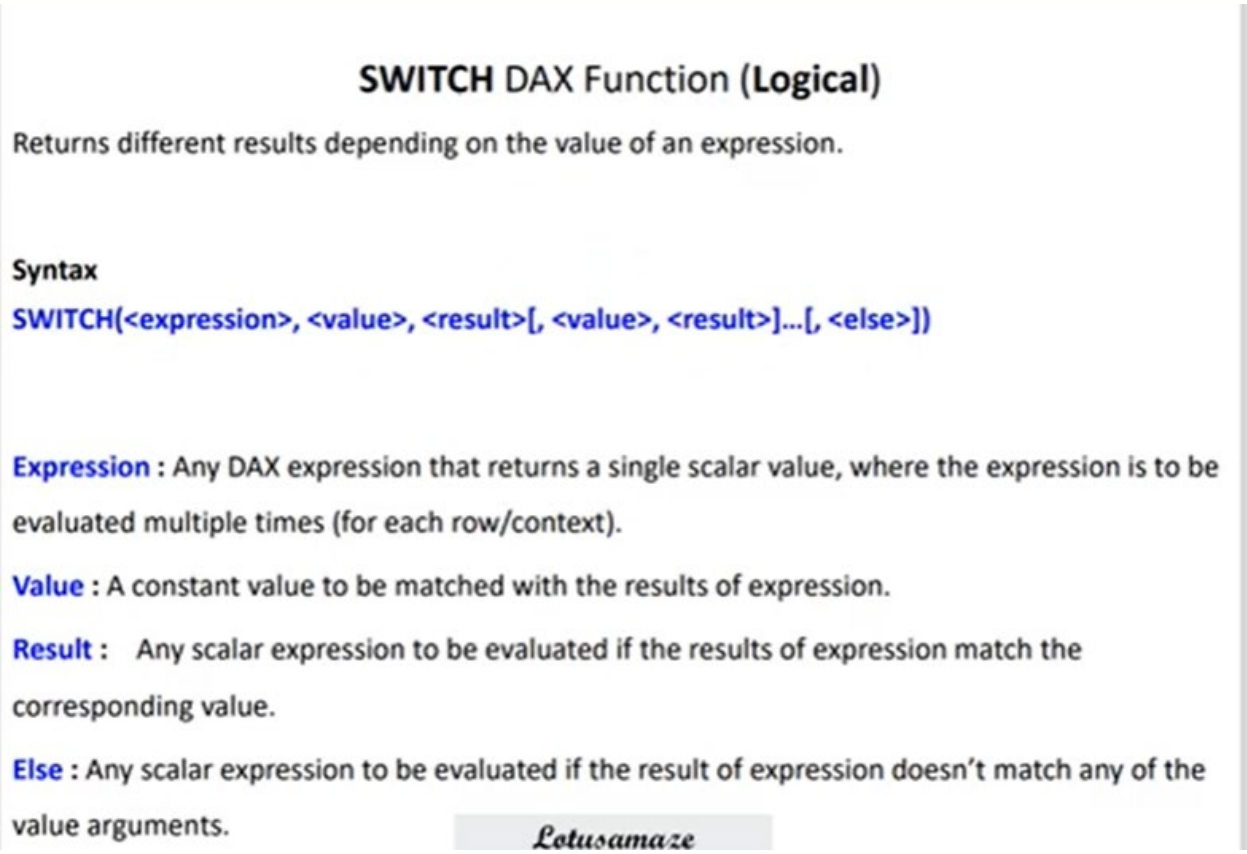
Sample:

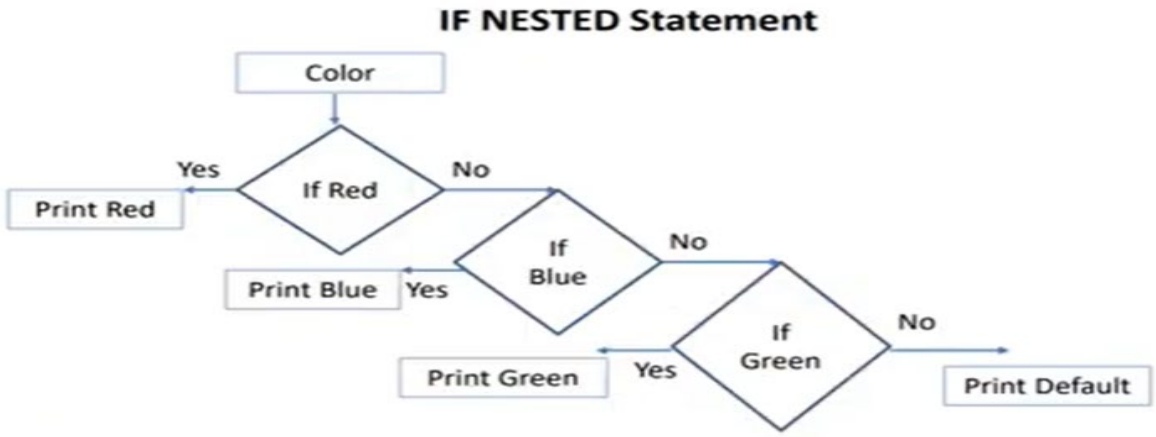
```
1] One Blank = COALESCE(Blank(), 10,20) - answer is 10
2] Two Blank = COALESCE(Blank(), Blank(), "Shivam", "Patil") - answer is Shivam
3] All Blank = COALESCE(BLANK(),BLANK(),BLANK(),BLANK())-answer is Blank
```

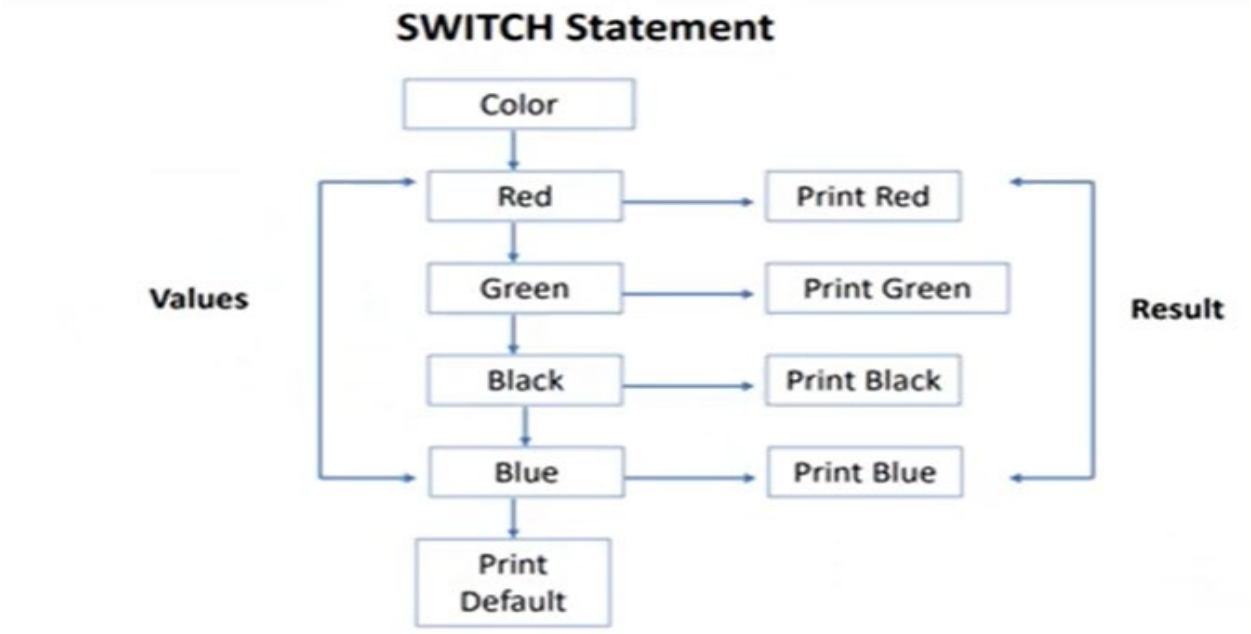



```
1] New Product = COALESCE('Sales Summary'[Product],"Key Board")
2] New Sales Date = COALESCE('Sales Summary'[Sales Date],TODAY())
3] New Net Sales = COALESCE('Sales Summary'[Net Sales],0)
```

20] Switch(Logical)








```
Country Code = SWITCH('Sales Summary'[Country],"Canada", "CA","France","FR",
"Germany","Ge","Defalut Code")
```

21] RankX(Statistical)

New Columns Created:

With Skip:

```
RankX for Net sales with Skip = RANKX('Sales Summary','Sales Summary'[Net
Sales],,DESC,Skip)
```

With Dense:

```
RankX for Net sales with Dense = RANKX('Sales Summary','Sales Summary'[Net
Sales],,DESC,Dense)
```

Power BI DAX RANKX

Returns the ranking of a number in a list of numbers for each row in the *table* argument.

Syntax:

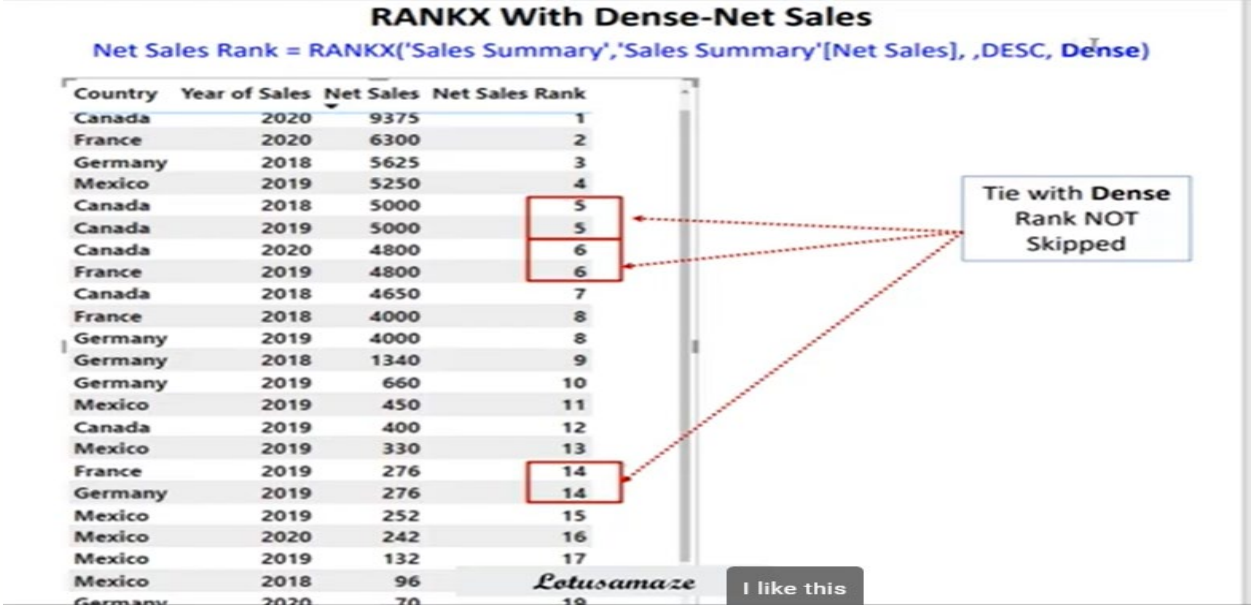
RANKX(<Table>, <Expression>, <Value>, <Order>, <Ties>)

- **Table(Iterator)**: The table (or virtual table) that is used as the source of items for the ranking.
- **Expression(Row Context)**: The expression that the ranking is calculated based on it. This DAX expression should return a single scalar value. The expression will iterate through each and every record in the table and return the values to be able to rank them accordingly.
- **value (optional)**; This is an optional parameter.
- **order(optional);** Based on what order the ranking is calculated. default is descending.
- **ties(optional);** In case of tie, it accepts two parameters - **SKIP and DENSE. SKIP** is used to skip the rank values in case of a tie whereas the **DENSE** assigns an exact count for each tie in the column. It is an optional parameter with a default value for **SKIP**.

Net Sales Rank = RANKX('Sales Summary','Sales Summary'[Net Sales],,DESC)

Country	Year of Sales	Net Sales	Net Sales Rank
Canada	2020	9375	1
France	2020	6300	2
Germany	2018	5625	3
Mexico	2019	5250	4
Canada	2018	5000	5
Canada	2019	5000	5
Canada	2020	4800	7
France	2019	4800	7
Canada	2018	4650	9
France	2018	4000	10
Germany	2019	4000	10
Germany	2018	1340	12
Germany	2019	660	13
Mexico	2019	450	14
Canada	2019	400	15
Mexico	2019	330	16
France	2019	276	17
Germany	2019	276	17
Mexico	2019	252	19
Mexico	2020	242	20

Tie
Rank Skipped
By Default



New Measures Created:

1] Due to Filter on Country it will give 1 ranking to all countries as shown in below snapshot.

```
RankX_Measure1 = RANKX('Sales Summary',SUM('Sales Summary'[Net Sales]),,)
```

Country	Sum of Net Sales	RankX_Measure1
Canada	29225	1
France	15376	1
Germany	12001	1
Mexico	6752	1
Total	**63354**	**1**

2] Now we remove all filters on Country using ALL() and then we get the following result.

```
RankX_Measure2 = RANKX(ALL('Sales Summary'[Country]),'Sales Summary'[Sum_NetSales],,)
```

Country	Sum of Net Sales	RankX_Measure1	RankX_Measure2
Canada	29225	1	1
France	15376	1	2
Germany	12001	1	3
Mexico	6752	1	4
Total	**63354**	**1**	**1**

2] Now again when we add Products in table, rankings gets misjumbled like below screenshot and then again we add Products in All() like below and update RankX_Measure2 like this:

```
RankX_Measure2 = RANKX(ALL('Sales Summary'[Country],'Sales Summary'[Product]),'Sales Summary'[Sum_NetSales],,)
```

Country	Product	Sum of Net Sales	RankX_Measure1	RankX_Measure2
Canada	Pad	24025	1	1
Canada	Pen	5200	1	2
France	Mouse	6300	1	1
France	Pen	8800	1	1
France	Pencel	276	1	3
Germany	Mouse	5901	1	2
Germany	Pen	4100	1	3
Germany	Pencel	2000	1	1
Mexico	Mouse	5832	1	3
Mexico	Pad	252	1	2
Mexico	Pen	96	1	4
Mexico	Pencel	572	1	2
Total		**63354**	**1**	**1**

Country	Product	Sum of Net Sales	RankX_Measure1	RankX_Measure2
Canada	Pad	24025	1	1
France	Pen	8800	1	2
France	Mouse	6300	1	3
Germany	Mouse	5901	1	4
Mexico	Mouse	5832	1	5
Canada	Pen	5200	1	6
Germany	Pen	4100	1	7
Germany	Pencel	2000	1	8
Mexico	Pencel	572	1	9
France	Pencel	276	1	10
Mexico	Pad	252	1	11
Mexico	Pen	96	1	12
Total		**63354**	**1**	**1**

22] TopN, Dynamic TopN(Table Manipulation)

Power BI DAX TOPN

Returns a given number of top rows according to a specified expression.

Syntax

TOPN(<n_value>, <table>, <orderBy_expression>, [<order>[, <orderBy_expression>, [<order>]]...])

- n_value: The count of items you want to pick either from the top or bottom of the input table.
- table: the input table to get the subset of items from it
- orderBy_expression: The expression used to order the items of the table so that the top or bottom items be picked.
- order (optional): The selection should be from top or bottom of the table? default is DESC which means TOP, if it changes to ASC it would act like bottom.

NOTE: TOPN function does not guarantee any sort order for the results.

Table Creation

1] Top_5_QTY Sold = TOPN(5,'Sales Summary')

Country	Product	Qty Sold
France	Pen	1999
Canada	Pad	800
Germany	Pen	350
France	Mouse	300
Canada	Pad	150

2] Top_5_QTY Sold = TOPN(5,'Sales Summary', 'Sales Summary'[Qty Sold])

Country	Product	Qty Sold
Canada	Pad	150
France	Mouse	300
Canada	Pad	800
France	Pen	1999
Germany	Pen	350

3] Top_5_QTY Sold = TOPN(5,'Sales Summary', 'Sales Summary'[Qty Sold],ASC)

Country	Product	Qty Sold
Mexico	Pen	12
Mexico	Pad	12
France	Pencil	12
Mexico	Mouse	12
Germany	Mouse	12
Germany	Pen	7
Canada	Pad	7

Ties in the Sorted Column: When you use the "asc" parameter to sort the data in ascending order, if there are tied values in the column you're sorting by, DAX may return all tied values. In your case, you have the value "12" occurring multiple times and the value "7" occurring multiple times. DAX may include all the tied values in the result set, resulting in more rows than you expected.

To address this and get only the distinct bottom 5 values, you may need to apply additional filtering or use a different DAX expression. You can use a combination of functions like SUMMARIZE and TOPN to get the distinct values before applying the "asc" sorting and then retrieve the bottom 5.

3] Top_5_QTY Sold = TOPN(5,'Sales Summary', 'Sales Summary'[Product])

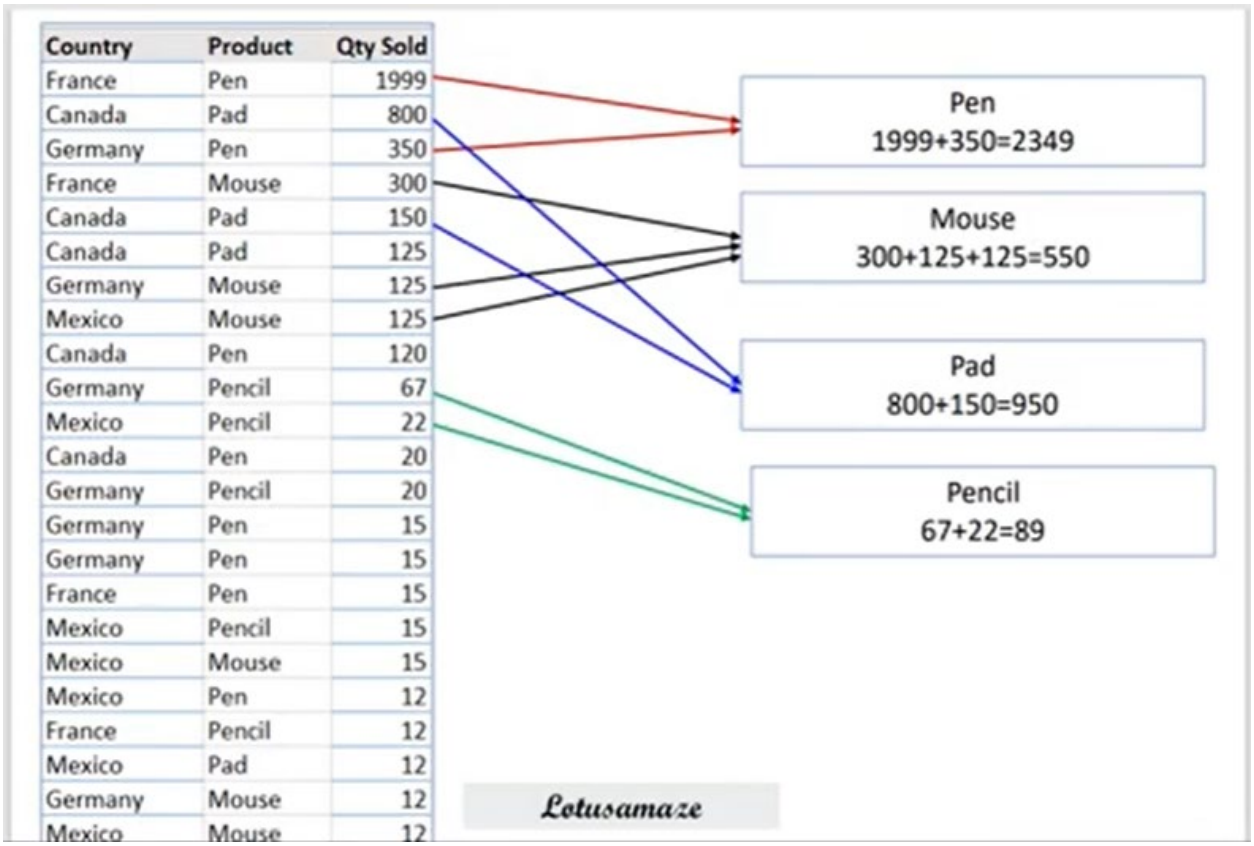
Country	Product	Qty Sold
Germany	Pencil	67
France	Pencil	12
Germany	Pencil	20
Mexico	Pencil	15
Mexico	Pencil	22

Measure Creation

```
1]Top_5_QTY SOld by sum = CALCULATE(SUM('Sales Summary'[Qty Sold]),TOPN(2,'Sales Summary','Sales Summary'[Qty Sold]))
```

Product	SalesSum	Top_5_QTY SOld by sum
Pen	2553	2349
Pad	1094	950
Mouse	589	550
Pencil	136	89
Total	**4372**	**2799**

Why Such Difference????



Dynamic Top N

1] First Create one table with 5 values in it as 1,2,3 4,5 and then create one measure as below.

```
Dynamic Measure = var setValue = SELECTEDVALUE(Numbers_table[Column1])
RETURN
CALCULATE(SUM('Sales Summary'[Qty Sold]),TOPN(setValue,'Sales Summary','Sales Summary'[Qty Sold]))
```

2] Drop this newly created measure in the table but it will show blank as we haven't passed the value through the slicer.

Product	SalesSum	Top_5_QTY SOld by sum	Dynamic Measure
Pen	2553	2349	
Pad	1094	950	
Mouse	589	550	
Pencil	136	89	
Total	**4372**	**2799**	

3] When we use slicer we will get result as:

Product	SalesSum	Top_5_QTY SOld by sum	Dynamic Measure
Pen	2553	2349	1999
Pad	1094	950	800
Mouse	589	550	300
Pencil	136	89	67
Total	**4372**	**2799**	**1999**

Column1
- [x] 1
- [ ] 2
- [ ] 3
- [ ] 4
- [ ] 5

Product	SalesSum	Top_5_QTY SOld by sum	Dynamic Measure
Pen	2553	2349	2469
Pad	1094	950	1075
Mouse	589	550	550
Pencil	136	89	109
Total	**4372**	**2799**	**3149**

Column1
- [ ] 1
- [ ] 2
- [x] 3
- [ ] 4
- [ ] 5

23] DAX - TreatAS(Table Manipulation)

TREATAS DAX Function (Table Manipulation)

Applies the result of a table expression as filters to columns from an unrelated table.

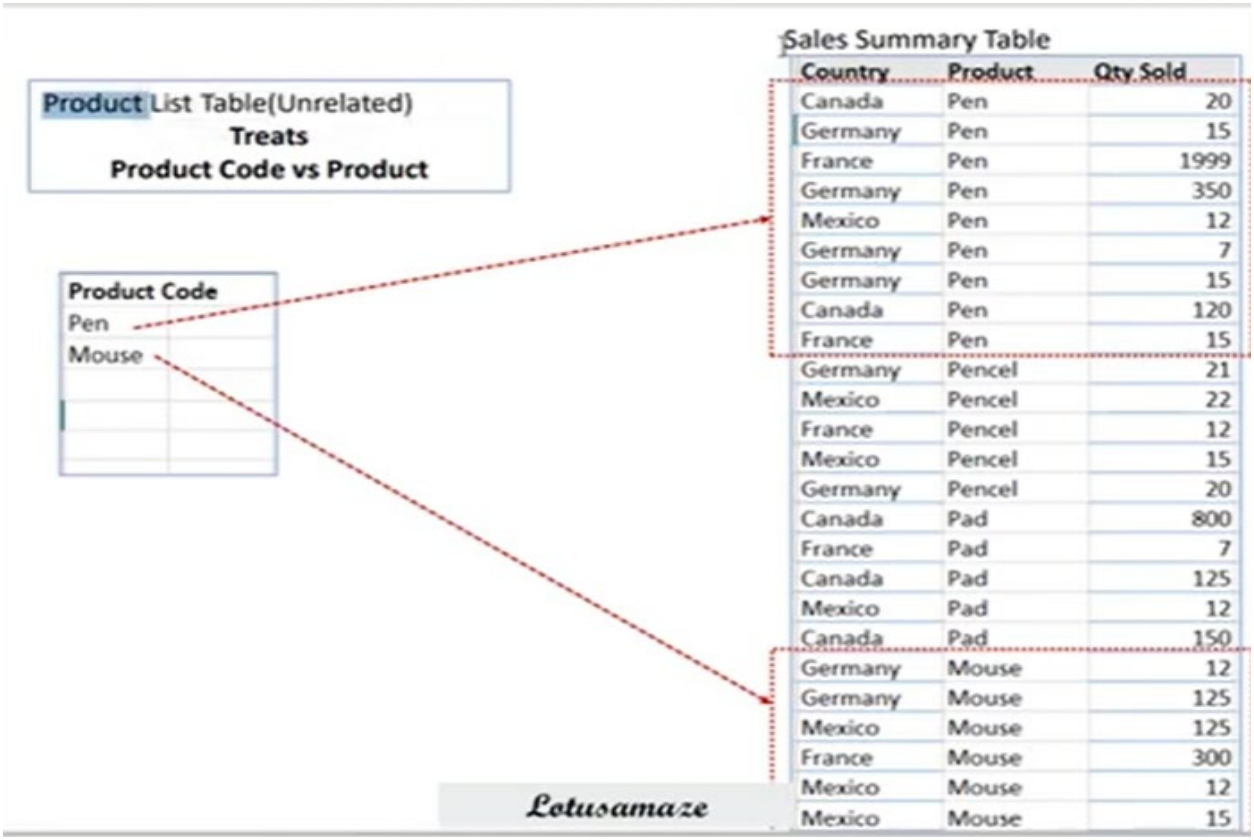
Syntax

TREATAS(table_expression, <column>[, <column>[, <column>[,...]]]})

- **Expression** : An expression that results in a table.
- **column**: One or more existing columns. It cannot be an expression.

➢ **Return values**

A table that contains all the rows in column(s) that are also in table_expression.



1] TreatAS_Measure = CALCULATE(SUM('Sales Summary'[Qty Sold]),TREATAS(VALUES('Product List'[Product Code]),'Sales Summary'[Product])

Country	TreatAS_Measure
Canada	140
France	2314
Germany	524
Mexico	164
Total	**3142**

24] DAX - Earlier

EARLIER DAX Function

- Returns the current value of the specified column in an outer evaluation pass of the mentioned column.
- *The **EARLIER** function allows us to jump from one context to a prior row context, so we can grab the result that we want.*

Syntax

EARLIER(<column name>,[<number>])

- **ColumnName**: The column that contains the desired value.
- **Number(optional)**: number of table scan

➢ **Return values**

A single value of any type

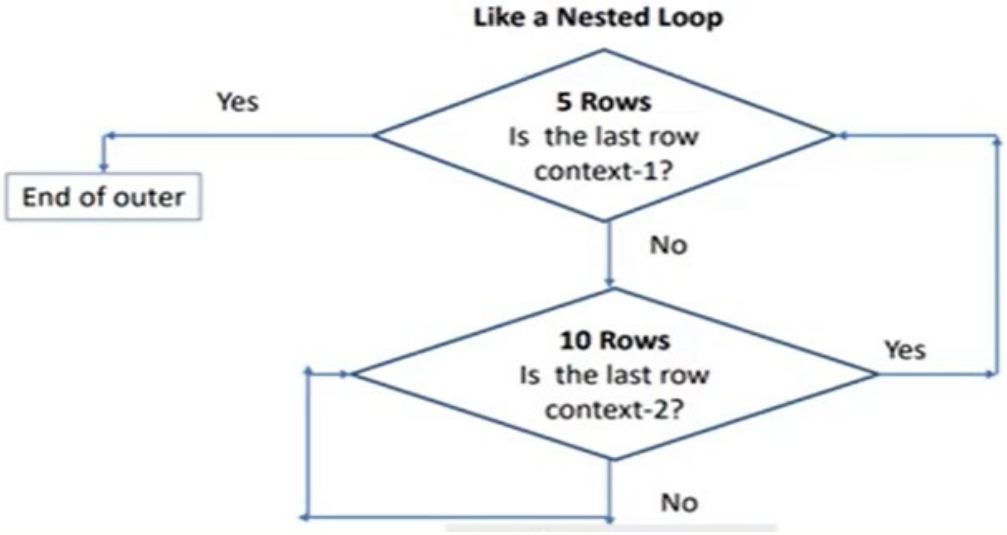
ROW Context

Row context *means that an evaluation is going to happen at every single row. And every single row in a calculated column has a result with the evaluation that's happening at every single row.*

Row Context 1	100
Row Context 2	200
Row Context 3	500
Row Context 4	567
Row Context 5	321
Row Context 6	678
Row Context 7	432
Row Context 8	231

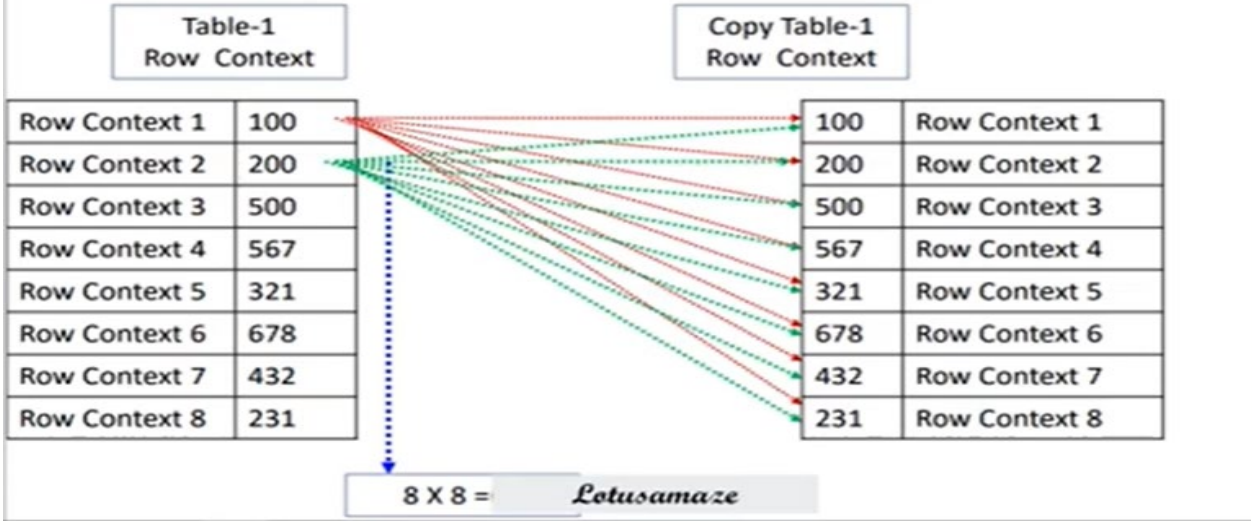
EARLIER DAX Function

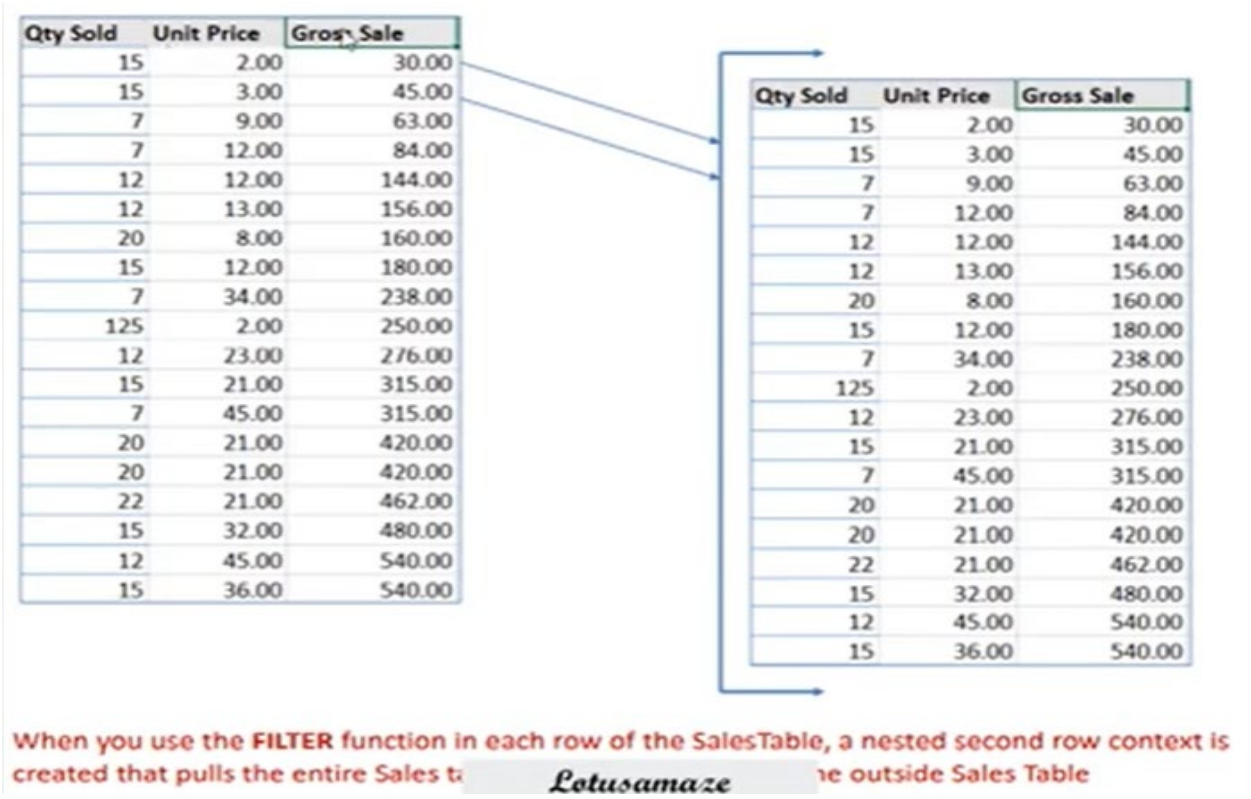
The Earlier Function returns the value of the Current Row in the Column specified and the **EARLIER function at-least needs to have 2 nested ROW Contexts.**



EARLIER DAX Function

The Earlier Function returns the value of the Current Row in the Column specified and the EARLIER function at-least needs to have 2 nested ROW Contexts.






```
1] Cumulative Gross Sale Sum = CALCULATE(SUM('Sales Summary'[Gross Sale]),
FILTER('Sales Summary','Sales Summary'[Gross Sale] <= EARLIER('Sales Summary'[Gross
Sale])))
```

```
2] Count Row = COUNTROWS(FILTER('Sales Summary','Sales Summary'[Gross Sale] <=
EARLIER('Sales Summary'[Gross Sale])))
```

25] SelectedColumns vs AddColumns(Table Manipulation)

SELECTCOLUMNS DAX Function

- Adds calculated columns to the given table or table expression.

➢ **SELECTCOLUMNS** is a tabular function in DAX, which means it returns a table as a result.

Syntax

SELECTCOLUMNS(<table>, <name>, <expression>[, <name>, <expression>]...)

- **Table**: the table that we want to add columns to it
- **Name**: the name of the new calculated column
- **Expression**: the expression of the new calculated column
- you can add multiple columns using this function.

➢ **Return values**

- An entire table or a table with one or more columns.
- **SELECTCOLUMNS** has the same signature as ADDCOLUMNS, and has the same behavior except that instead of starting with the <Table> specified, SELECTCOLUMNS starts with an empty table before adding columns.

ADDCOLUMNS DAX Function

- Adds calculated columns to the given table or table expression.

➢ **ADDCOLUMNS** is a tabular function in DAX, which means it returns a table as a result.

Syntax

ADDCOLUMNS(<table>, <name>, <expression>[, <name>, <expression>]...)

- **Table**: the table that we want to add columns to it
- **Name**: the name of the new calculated column
- **Expression**: the expression of the new calculated column
- you can add multiple columns using this function.

➢ **Return values**

- An entire table or a table with one or more columns.

SelectColumns Vs. AddColumns

One big difference of **SelectColumns** and **AddColumns** is that AddColumns keep all the existing columns in the table and adds more columns to that, But the SelectColumns starts with no columns from the table, and builds a custom subset of columns or with additional calculated columns on it.

1] For SelectColumns we have to create new table and then run this formula

```
SelectedColumns Sample = SELECTCOLUMNS('Sales Summary',
"Country",'Sales Summary'[Country],
"Product", 'Sales Summary'[Product],
"Unit Price",'Sales Summary'[Unit Price],
"QTY Sold",'Sales Summary'[Qty Sold],
"Total Sales",CALCULATE(SUM('Sales Summary'[Unit Price]) * SUM('Sales Summary'[Qty Sold])))
```

2] AddColumns append columns in existing table

```
AddColumns = ADDCOLUMNS('Sales Summary',
"Total Sales",CALCULATE(SUM('Sales Summary'[Unit Price]) * SUM('Sales Summary'[Qty Sold])))
```

26] DAX - Filter(Filter)

FILTER DAX Function

- Returns a table that has been filtered.

Syntax

FILTER(<table>,<filter>)

- **Table(ITERATOR)** : The table to be filtered.
- **FilterExpression(ROW CONTEXT):** A boolean (True/False) expression that is to be evaluated for each row of the table.

Return values

TABLE An entire table or a table with one or more columns.

Common Uses of the FILTER() Function

- **FILTER()** is most often used as a **filter parameter** within the CALCULATE() function.
- It is getting confusing. **FILTER()** is a function. There is another use of the word filter in DAX, and that is as a parameter in CALCULATE(). Let's look at the

FILTER Syntax:

- FILTER(<table>, <true/false test>)

CALCULATE Syntax:

- CALCULATE (<measure expression>, <filter parameter>, <filter parameter>,...)

- As shown above, the CALCULATE() function can take one or more filter parameters, but those filter *parameters* may or may not be FILTER() functions.
- Don't confuse the filter *parameters* inside CALCULATE() with the FILTER() function – they are two separate things.

```
1] Table1 Filter = FILTER('Sales Summary','Sales Summary'[Country]="Canada")
2] Table2 Filter = FILTER('Sales Summary','Sales Summary'[Product]="Pen")
3] Table2 Filter = FILTER('Sales Summary',AND('Sales Summary'[Product]="Pen", 'Sales Summary'[Country]="Canada"))
4] Total Canada = COUNTROWS(FILTER('Sales Summary','Sales Summary'[Country] = "Canada"))
5] Total Canada Sales = CALCULATE(SUM('Sales Summary'[Qty Sold]),FILTER('Sales Summary','Sales Summary'[Country] = "Canada"))
```

27] Row Context VS Filter Context

Power BI

Row Context

Filter Context

in DAX

- The situation in which something happens.
- The context is the "**environment**" within which the DAX calculates the expression.

Row Context

- A Row context is evaluated whenever an expression iterates over a table. Each individual row in a table has its own row context.
- A row context exists in either a calculated column or an iterating DAX functions such as **SUMX, AVERAGEX, FILTER & ADDCOLUMNS etc.**
- If you have created a calculated column, the row context consists of the values in each individual row and values in columns that are related to the current row.

```
1 Profit = 'Sales Summary'[Gross Sale] - 'Sales Summary'[Net Sales]
```

Country	Product	Qty Sold	Year of Sales	Sales Date	Gross Sale	Net Sales	Profit	
Canada	Pen	20	2019	11 November 2019	500	400	100	Row Context-1
Germany	CPU	15	2019	07 July 2019	432	300	132	Row Context-2
France	Pencil	1999	2018	06 August 2018	554	500	54	
Germany	Pad	350	2019	09 February 2019	696	200	496	
Mexico	Pad	12	2018	21 August 2019	44	455	-411	
Germany	Pencil	7	2020	11 September 2020	789	309	480	
Germany	Pad	15	2019	22 November 2019	54	454	-400	

Filter Context

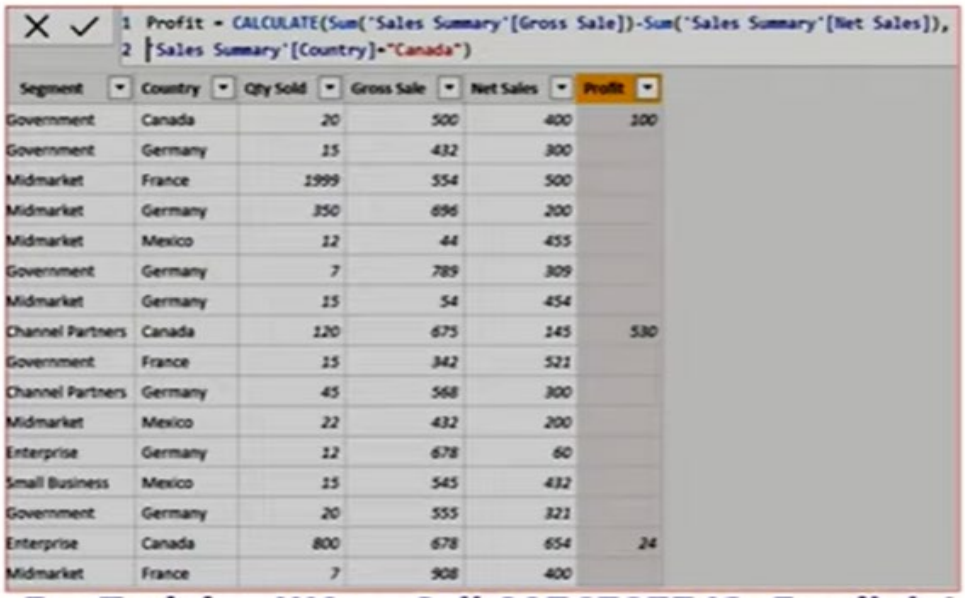
- The filter context is the set of filters applied to the data model before the evaluation of a DAX expression starts.
- Filter Context always exist before a DAX expression been evaluated.
- Filter context applies on top of other contexts, such as row context or query context.

Some initial level filters context applied on Power BI report page in form of:-

- Slicer
- Filter through other Visual
- In Matrix visual based on Row and column data fields
- Power BI filters

Filter Context

- Profit = **CALCULATE** (Sum('Sales Summary'[Gross Sale])-Sum('Sales Summary'[Net Sales]), 'Sales Summary'[Country]="Canada")



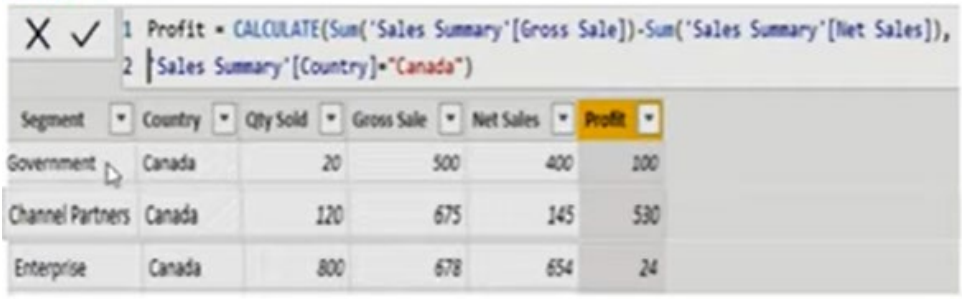
```
1 Profit = CALCULATE(Sum('Sales Summary'[Gross Sale])-Sum('Sales Summary'[Net Sales]),
2 'Sales Summary'[Country]="Canada")
```

Segment	Country	Qty Sold	Gross Sale	Net Sales	Profit
Government	Canada	20	500	400	100
Government	Germany	15	432	300	
Midmarket	France	1999	554	500	
Midmarket	Germany	350	696	200	
Midmarket	Mexico	12	44	455	
Government	Germany	7	789	309	
Midmarket	Germany	15	54	454	
Channel Partners	Canada	120	675	145	530
Government	France	15	342	521	
Channel Partners	Germany	45	568	300	
Midmarket	Mexico	22	432	200	
Enterprise	Germany	12	678	60	
Small Business	Mexico	15	545	432	
Government	Germany	20	555	321	
Enterprise	Canada	800	678	654	24
Midmarket	France	7	908	400	

Filter Context

- Profit = **CALCULATE** (Sum('Sales Summary'[Gross Sale])-Sum('Sales Summary'[Net Sales]), 'Sales Summary'[Country]="Canada")

Step-1



```
1 Profit = CALCULATE(Sum('Sales Summary'[Gross Sale])-Sum('Sales Summary'[Net Sales]),
2 'Sales Summary'[Country]="Canada")
```

Segment	Country	Qty Sold	Gross Sale	Net Sales	Profit
Government	Canada	20	500	400	100
Channel Partners	Canada	120	675	145	530
Enterprise	Canada	800	678	654	24

ROW CONTEXT

- Row context does calculation for each row, with the values within that row.

```
Profit = Sumx( Sales,
                [Sales Amount ] - [Cost Amount ]
        )
```

FILTER CONTEXT

- Filter context is, set of Filters that applied before that table arrives for use.

```
Total Sales = Sum( Sales [Sales Amount ] )
```

Applying filters from Filter pane / Visuals / Slicers (Year = 2017 , Country = India , City = Chennai)

28] DAX - SamePeriodLastYear

Power BI

SAMEPERIODLASTYEAR

in DAX

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

Syntax

SAMEPERIODLASTYEAR(<dates>)

Dates : A column containing dates.

Return value : A single-column table of date values.

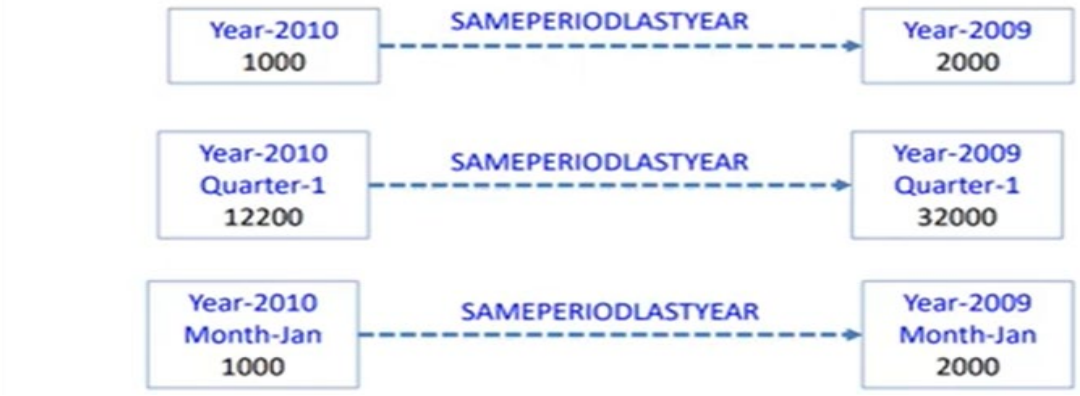
NOTE: This is not returning one single value. Means you cannot use it directly in a measure.

You have to use this function as a filter function.

SAMEPERIODLASTYEAR in DAX

How and Why to use

- When you want to Calculate and compare the difference between the current year data and the previous year's data.
- You can use this DAX function.




```
Gross Sale Previous Year = CALCULATE([Total Gross Sale], SAMEPERIODLASTYEAR('Sales Summary'[Sales Date].[Date]))
```

29] DAX - ParallelPeriod(Context Transition)

DAX PARLLELPERIOD

Returns a parallel period of dates by the given set of dates and a specified interval.

Syntax

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, month.

Return value

A table containing a single column of date values.

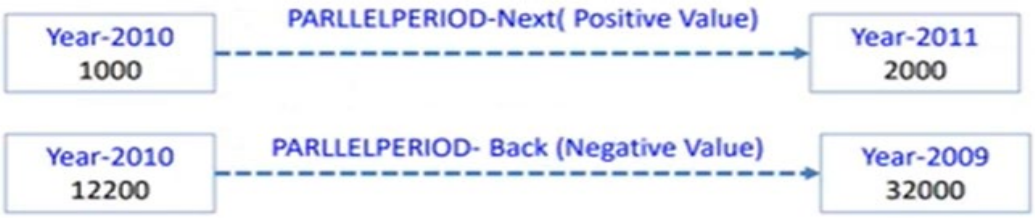
NOTE: This is not returning one single value. Means you cannot use it directly in a measure.

You have to use this function as a filter function.

DAX PARLLELPERIOD

How and Why to use

- When you want to Calculate and compare the difference between the current year data and the previous or next year's data.
- You can use this DAX function.




```
1] ParallelPeriod Sales = CALCULATE('Sales Summary'[Total Gross Sale],PARALLELPERIOD('Sales Summary'[Sales Date].[Date],1,YEAR))
2] ParallelPeriod Sales = CALCULATE('Sales Summary'[Total Gross Sale],PARALLELPERIOD('Sales Summary'[Sales Date].[Date],-1,YEAR))
3] ParallelPeriod Sales = CALCULATE('Sales Summary'[Total Gross Sale],PARALLELPERIOD('Sales Summary'[Sales Date].[Date],2,YEAR))
```

30] DAX - Variables

DAX Variables

- Stores the result of an expression.
- Which can then be passed as an argument to other measure expressions.
- Once resultant values have been calculated for a variable expression, those values do not change, even if the variable is referenced in another expression.
- Variables are a major feature that makes writing DAX code easier. Moreover, it greatly increases the readability and reusability of your code.
- Syntax:

```
VAR <name> = <expression>
[VAR <name2> = <expression2> [...]]
RETURN <result_expression>
```

The RETURN keyword consumes variables defined in previous VAR statements. The result_expression has access to all the variables (name, name2, ...) defined in the VAR statement(s) before RETURN.

```
1] Country Name with Var =
```

```
var canadaSales = CALCULATE(SUM('Sales Summary'[Gross Sale]),'Sales Summary'[Country] = "Canada")
var germanySales = CALCULATE(SUM('Sales Summary'[Gross Sale]),'Sales Summary'[Country] = "Germany")
var germanyProductsCount = CALCULATE(COUNT('Sales Summary'[Gross Sale]),'Sales Summary'[Country] = "Germany")
RETURN
IF(canadaSales > germanySales, "Canada:" & canadaSales,"Germany Sales and Product Counts:" & germanySales & "," & germanyProductsCount)
```

2] Country Name with Var =

```
var canadaSales = CALCULATE(SUM('Sales Summary'[Gross Sale]),'Sales Summary'[Country] = "Canada")
var germanySales = CALCULATE(SUM('Sales Summary'[Gross Sale]),'Sales Summary'[Country] = "Germany")
var germanyProductsCount = CALCULATE(COUNT('Sales Summary'[Gross Sale]),'Sales Summary'[Country] = "Germany")
RETURN
IF(canadaSales > germanySales, "Canada:" & canadaSales,"Germany:" & germanySales)
```

3] Difference of Sales =

```
var totalSales = SUM('Sales Summary'[Gross Sale])
var lastYearSales = CALCULATE(SUM('Sales Summary'[Gross Sale]),SAMEPERIODLASTYEAR('Sales Summary'[Sales Date].[Date]))
RETURN
totalSales - lastYearSales
```

31] IsFiltered And IsCrossFiltered

Filter Types

There are three types of filters that we need to consider while working wit **Isfiltered** & **Iscrossfiltered** in DAX.

1. **Direct Filter**
   - Direct filters are those filters that effect directly on the column that is being used.
2. **Indirect Filter**
   - The indirect filter is that filter, which is not the direct filter.
3. **Cross Filter**
   - CrossFilter function is neither tabular nor scalar function, It is a specific type of function that changes the direction of a relationship.

Filter Types

➢ **ISFILTERED DAX Function (Information)**

Returns true when there are direct filters on the specified column.

Syntax:

ISFILTERED(<TableNameOrColumnName>)

1. **Create Two measures**
   1. **IsCityFiltered = ISFILTERED(DimCity[CityName])**
   2. **IsProductFiltered = ISFILTERED(DimProduct[ProductName])**

```
1] isCityFiltered = ISFILTERED(DimCity[CityName])
2] isProductFiltered = ISFILTERED(DimProduct[ProductName])
Return True or False only.
```

ISCROSSFILTERED

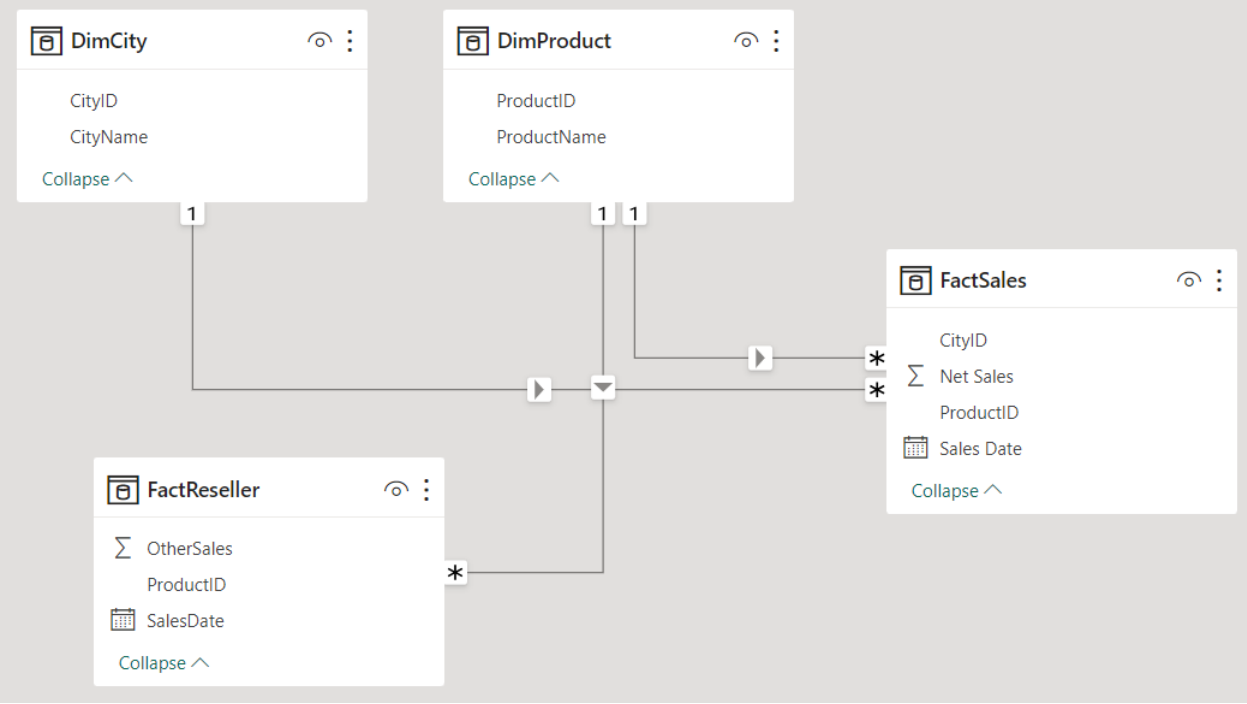
- Returns TRUE when the specified table or column is cross-filtered.
- A column or table is said to be cross-filtered when a filter is applied to ColumnName, any column of TableName, or to any column of a related table.

Syntax
ISCROSSFILTERED(<TableNameOrColumnName>)

TableNameOrColumnName : The name of an existing table or column. It cannot be an expression.

Return value
TRUE when ***ColumnName*** or a column of ***TableName*** is being cross-filtered. Otherwise returns FALSE.

As you can see there is no relation between Dim City and Fact Reseller.



And when we put CityName from DimCity, Net Sales from FactSales and OtherSales from FactReseller then we don't get correct sum of Othersalesand also when we put isCrossFiltered measure in table we get false.

CityName	Sum of Net Sales	Sum of OtherSales
Delhi	700	549
Jaipur	900	549
Lucknow	600	549
Mumbai	2000	549
Nagpur	400	549
Noida	1500	549
Pune	200	549
Total	**6300**	**549**

Now when we go to model View and change cross filter direction between DimProduct and factSales to both we get the following output.

CityName	Sum of Net Sales	Sum of OtherSales	isProductCrossFiltered
Delhi	700	482	True
Jaipur	900	305	True
Lucknow	600	328	True
Mumbai	2000	372	True
Nagpur	400	44	True
Noida	1500	328	True
Pune	200	177	True
Total	**6300**	**549**	**False**

32] Summarize and SummarizeColumns(Table Manipulation)

SUMMARIZE

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

Syantax

SUMMARIZE (<table>, <groupBy_columnName>[, <groupBy_columnName>]…[, <name>, <expression>]…)

Term	Definition
table	Any DAX expression that returns a table of data.
groupBy_ColumnName	(Optional) The qualified name of an existing column used to create summary groups based on the values found in it. This parameter cannot be an expression.
name	The name given to a total or summarize column, enclosed in double quotes.
expression	Any DAX expression that returns a single scalar value, where the expression is to be evaluated multiple times (for each row/context).

Return value

A table with the selected columns for the **groupBy_columnName** arguments and the summarized columns designed by the name arguments.

```
1] Product Summary Table =
SUMMARIZE('Product','Product'[ProductName],"Total
Sales",SUM('Product'[Sales]),"Count of
Product",COUNT('Product'[Sales]))
2] Product Summary Table =
SUMMARIZE('Product','Product'[ProductName],ROLLUP('Product'[Color])
,"Total Sales",SUM('Product'[Sales]),"Count of
Product",COUNT('Product'[Sales]))
```

SUMMARIZECOLUMNS

- Returns a summary table over a set of groups.

Syantax

SUMMARIZECOLUMNS(<groupBy_columnName> [, < groupBy_columnName >]..., [<filterTable>]...[, <name>, <expression>]...)

Term	Definition
groupBy_columnName	A fully qualified column reference (Table[Column]) to a base table for which the distinct values are included in the returned table. Each groupBy_columnName column is cross-joined (different tables) or auto-existed (same table) with the subsequent specified columns.
filterTable	A table expression which is added to the filter context of all columns specified as groupBy_columnName arguments. The values present in the filter table are used to filter before cross-join/auto-exist is performed.
name	A string representing the column name to use for the subsequent expression specified.
expression	Any DAX expression that returns a single value (not a table).

```
Return values:
TABLE: An entire table or table with one or more columns
Product Summary Column =
SUMMARIZECOLUMNS(ProductCode[ProductName],FILTER('Product','Prod
uct'[Color] = "Red"),"Total Sales",SUM('Product'[Sales]))
```

33] DAX - NetworkDays

NETWORKDAYS

Returns the number of whole workdays between two dates (inclusive). Parameters specify which and how many days are weekend days. Weekend days and days specified as holidays are not considered as workdays.

Syntax

NETWORKDAYS(<start_date>, <end_date>[, <weekend>, <holidays>])

Term	Definition
start_date	A date that represents the start date. The dates for which the difference is to be computed. The start_date can be earlier than, the same as, or later than the end_date.
end_date	A date that represents the end date. The dates for which the difference is to be computed. The start_date can be earlier than, the same as, or later than the end_date.
weekend	Indicates the days of the week that are weekend days and are not included in the number of whole working days between start_date and end_date. Weekend is a weekend number that specifies when weekends occur. Weekend number values indicate the following weekend days: 1 or omitted: Saturday, Sunday 2: Sunday, Monday 3: Monday, Tuesday 4: Tuesday, Wednesday

weekend	Indicates the days of the week that are weekend days and are not included in the number of whole working days between start_date and end_date. Weekend is a weekend number that specifies when weekends occur. Weekend number values indicate the following weekend days: 1 or omitted: Saturday, Sunday 2: Sunday, Monday 3: Monday, Tuesday 4: Tuesday, Wednesday

NETWORKDAYS

Term	Definition
weekend	4: Tuesday, Wednesday 5: Wednesday, Thursday 6: Thursday, Friday 7: Friday, Saturday 11: Sunday only 12: Monday only 13: Tuesday only 14: Wednesday only 15: Thursday only 16: Friday only 17: Saturday only
holidays	A column table of one or more dates that are to be excluded from the working day calendar.

Return Value

An integer number of whole workdays.

```
1] Sample of Net_Work_Days =
NETWORKDAYS(DATE(2023,11,01),DATE(2023,11,10),1) - Total Count 8 days
without counting holiday on 07-11-2023.
2] Sample of Net_Work_Days =
NETWORKDAYS(DATE(2023,11,01),DATE(2023,11,10),1,{DATE(2023,11,07)}) -
Total Count 7 days with counting holiday on 07-11-2023.
3] Employees Net_Work_Days = NETWORKDAYS('Employee Attendance
Data'[Attendance From],'Employee Attendance Data'[Attendance To],1) - with
sat and sun as weekend.
4] Employees Net_Work_Days = NETWORKDAYS('Employee Attendance
Data'[Attendance From],'Employee Attendance Data'[Attendance To],11) - with
sun as weekend only.
5] Employees Net_Work_Days = NETWORKDAYS('Employee Attendance
Data'[Attendance From],'Employee Attendance Data'[Attendance To],1,
VALUES(Holidays[DATE]))
6] Employees Net_Work_Days1 = NETWORKDAYS('Employee Attendance
Data'[Attendance From],'Employee Attendance Data'[Attendance To],1)
```

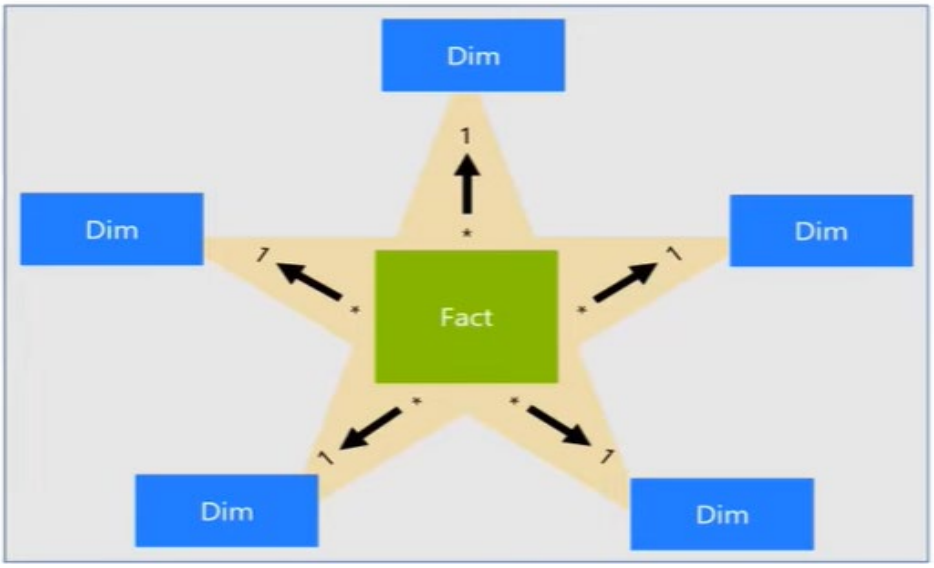
34] Dimension Table, Fact Table, Star Schema

Star schema overview

- **Star schema** is a mature modeling approach widely adopted by relational data warehouses.
- It requires **modelers** to classify their model tables as either **dimension or fact.**
  - **Dimension tables** describe business entities—the *things* you model. Entities can include products, people, places, and concepts including time itself. The most consistent table you'll find in a star schema is a date dimension table. A dimension table contains a key column (or columns) that acts as a unique identifier, and descriptive columns.
  - **Fact tables** store observations or events, and can be sales orders, stock balances, exchange rates, temperatures, etc. A fact table contains dimension key columns that relate to dimension tables, and numeric measure columns.

Star schema overview

- **Star schema** is a mature modeling approach widely adopted by relational data warehouses.
- It requires **modelers** to classify their model tables as either **dimension or fact.**



- **Dimension Table**

1. A dimension table contains descriptive attribute fields.
2. It contains attributes on which truth table calculates the metric values.
3. It contains more attributes in comparison to fact table.
4. It contains less rows in comparison to dimension table.
5. It forms a horizontal table.
6. The attribute format is in text form.
7. It occurs before the fact table.
8. The dimension table helps store information about business and its process.
9. **Examples**:
    - DimCustomer – attributes of a customer
    - DimEmployee – attributes of an employee
    - DimInvoice – invoice attributes
    - DimOrganization – attributes of an organization
    - DimDate – attributes of a date (ex. fiscal year, month, day)

➢ Dimension Table

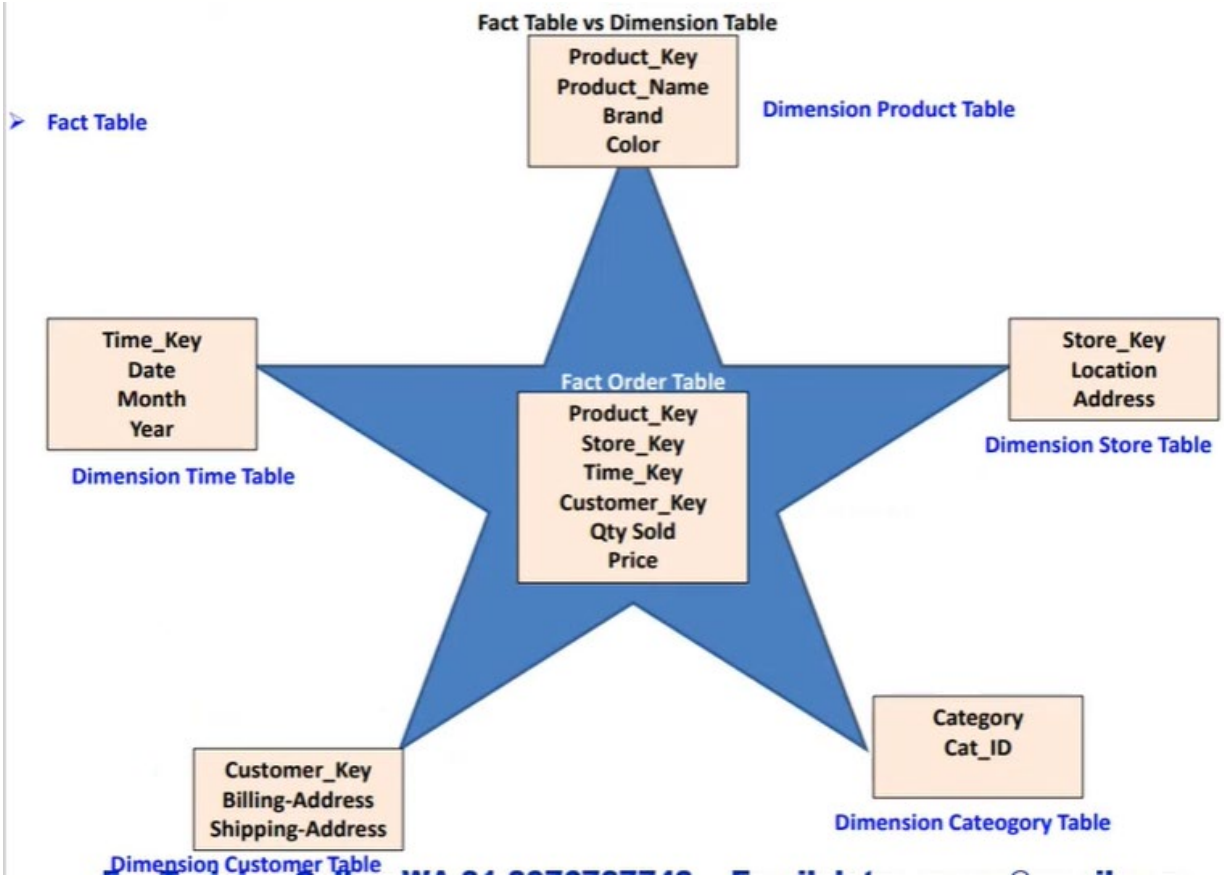
1. **Examples**:

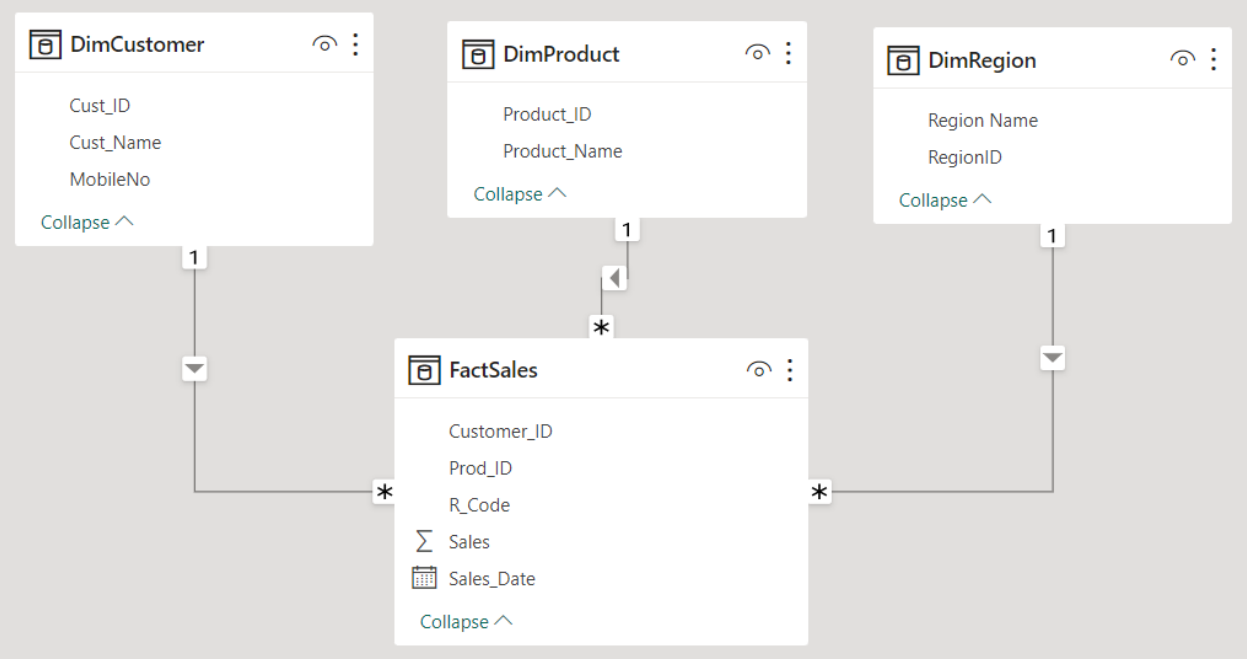
DimCustomer	CustomrID, Fisrt_Name, Last-Name, MobileNo, Address
DimEmployee	EmployeeID, Address, Designation, City, BirthDate

➢ Fact Table

1. It is also known as a reality table.
2. It can be a combination of attributes from completely different dimension table.
3. It helps the user to understand the business dimensions that would help improve the business.
4. It contains less attributes.
5. It contains more records.
6. It forms a vertical table.
7. It comes after the dimension table.
8. The attributes in a fact table can be in numerical or text format.
9. It can be used for analysis, and decision-making process.

➢ Fact Table





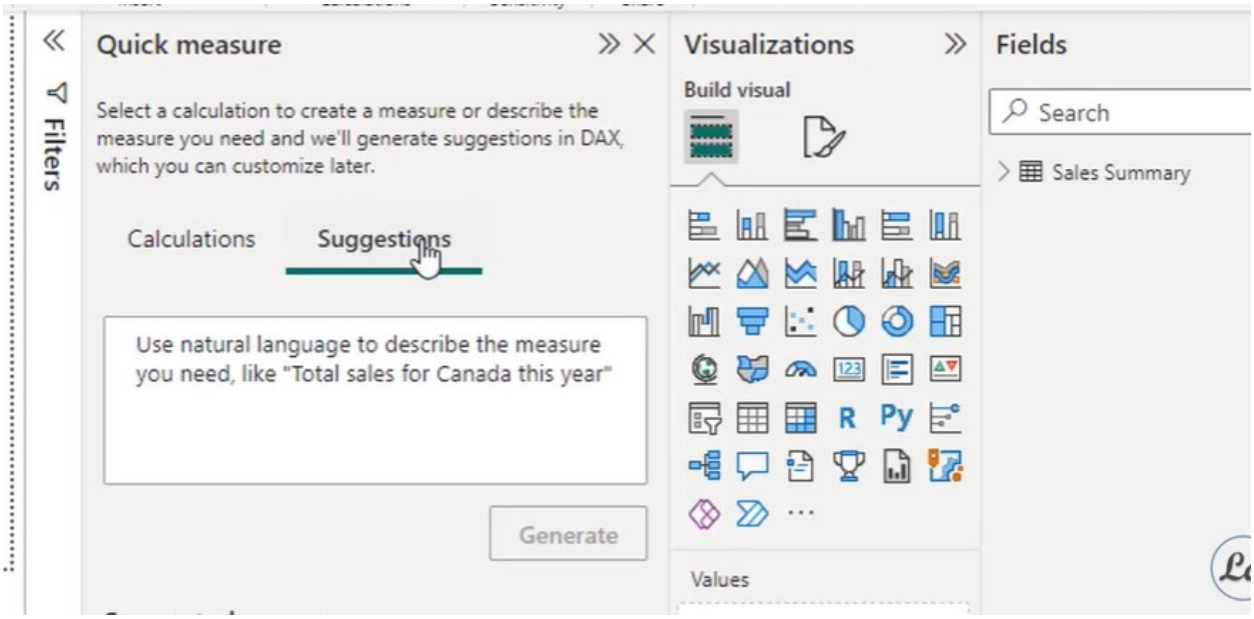
35] Quick Measure in Power BI(suggestions)

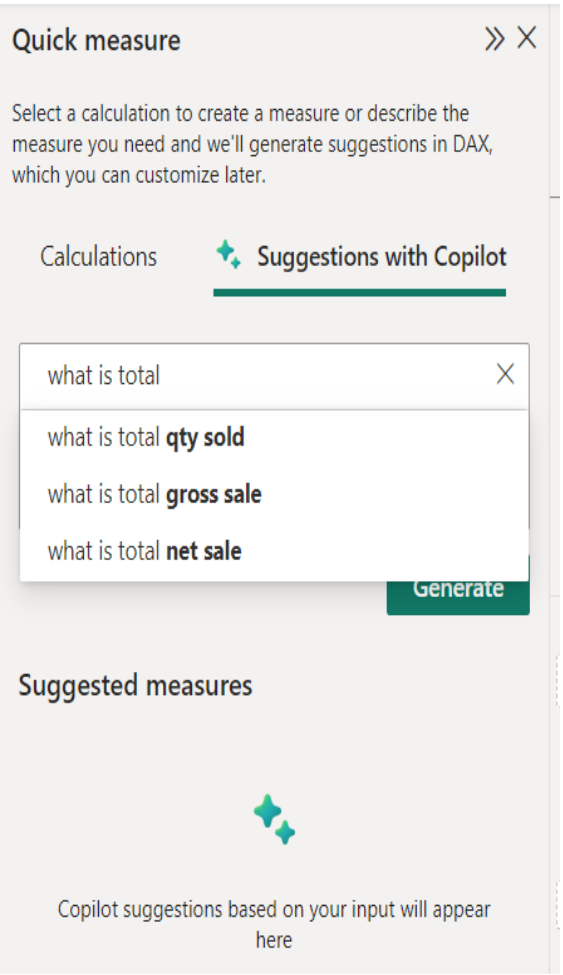
Quick measure suggestions (experimental preview)

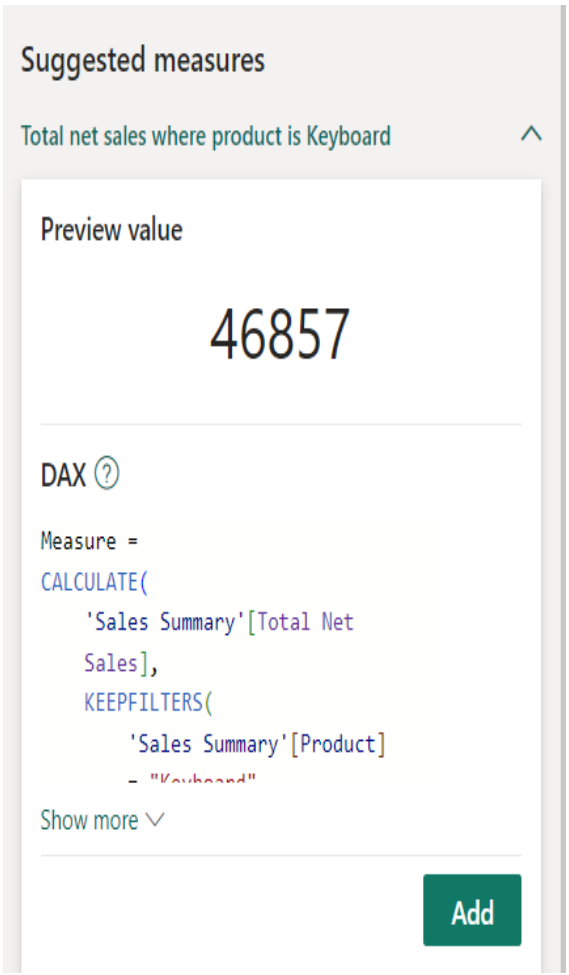
- **Quick measure suggestions assist creation of DAX measures using natural language instead of using templates or writing DAX from scratch.**
- **This feature can be used to jump-start creation of common DAX measures scenarios such as:**
  - **Aggregated columns (Optional filters)**
  - **Count of rows (Optional filters)**
  - **Aggregate per category**
  - **Mathematical operations**
  - **Selected value**
  - **Top N filtered value**
  - **Top N values for a category**
  - **Information functions**

- **Limitations and considerations**

1. Quick measure suggestions are NOT a replacement for learning DAX.
2. The suggestions provided by the feature are meant to help fast track measure creation; however, you will still need to validate the DAX suggestions because they can be wrong or not match your intent.
3. This feature is in experimental preview for users to test and give feedback. Note that the design and functionality may go through significant changes.
4. The feature is powered by a machine learning model that is currently only deployed to US datacenters (East US and West US). If your data is outside the US, the feature will be disabled by default unless your tenant admin enables Allow user data to leave their geography tenant setting:







37] DAX KeepFilters

Modifies how filters are applied while evaluating a CALCULATE or CALCULATETABLE function.

Syntax

DAX

Copy

```
KEEPFILTERS(<expression>)
```

Parameters

Term	Definition
expression	Any expression.

Return value

A table of values.

Remarks

- You use KEEPFILTERS within the context CALCULATE and CALCULATETABLE functions, to override the standard behavior of those functions.
- By default, filter arguments in functions such as CALCULATE are used as the context for evaluating the expression, and as such filter arguments for CALCULATE replace all existing filters over the same columns. The new context effected by the filter argument for CALCULATE affects only existing filters on columns mentioned as part of the filter argument. Filters on columns other than those mentioned in the arguments of CALCULATE or other related functions remain in effect and unaltered.
- The KEEPFILTERS function allows you to modify this behavior. When you use KEEPFILTERS, any existing filters in the current context are compared with the columns in the filter arguments, and the intersection of those arguments is used as the context for evaluating the expression. The net effect over any one column is that both sets of arguments apply: both the filter arguments used in CALCULATE and the filters in the arguments of the KEEPFILTER function. In other words, whereas CALCULATE filters replace the current context, KEEPFILTERS adds filters to the current context.

- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

```
Formal Sales with KF = CALCULATE(SUM(Data[Sales
Amount]),KEEPFILTERS(Data[Product Category] = "Formal"))
```

37] DAX - RemoveFilters

Clear filters from the specified tables or columns.

Syntax

DAX

Copy

```
REMOVEFILTERS([<table> | <column>[, <column>[, <column>[,…]]]])
```

Parameters

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

Return value

N/A. See remarks.

Remarks

- REMOVEFILTERS can only be used to clear filters but not to return a table.
- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.

Example 1

DAX query

DAX

Copy

```
DEFINE
MEASURE FactInternetSales[TotalSales] =
SUM(FactInternetSales[SalesAmount])
MEASURE FactInternetSales[%Sales] = DIVIDE([TotalSales],
CALCULATE([TotalSales],REMOVEFILTERS()))

EVALUATE
    SUMMARIZECOLUMNS(

ROLLUPADDISSUBTOTAL(DimProductCategory[EnglishProductCategoryNam
e], "IsGrandTotal"),
      "TotalSales", [TotalSales],
      "%Sales", [%Sales]
    )
ORDER BY
  [IsGrandTotal] DESC, [TotalSales] DESC
```

Returns

DimProductCategory[EnglishProductCategoryName]	[IsGrandTotal]	[TotalSales]	[%Sales]

Row1	True	29358677.2207	1
Bikes	False	28318144.6507	0.964557920570538
Accessories	False	700759.96	0.023868921434441
Clothing	False	339772.61	0.011573157995021 5

Example 2

DAX query

DAX

Copy

```
DEFINE
MEASURE FactInternetSales[TotalSales] =
SUM(FactInternetSales[SalesAmount])
MEASURE FactInternetSales[%Sales] = DIVIDE([TotalSales],
CALCULATE([TotalSales],REMOVEFILTERS(DimProductSubcategory[Engli
shProductSubcategoryName])))

EVALUATE
    SUMMARIZECOLUMNS(
      DimProductCategory[EnglishProductCategoryName],
      DimProductSubcategory[EnglishProductSubcategoryName],
      "TotalSales", [TotalSales],
      "%Sales", [%Sales]
    )
```

```
ORDER BY
  DimProductCategory[EnglishProductCategoryName] ASC,
  DimProductSubcategory[EnglishProductSubcategoryName] ASC
```

Returns

DimProductCategory [EnglishProductCategoryName]	DimProductSubcategory [EnglishProductSubcategoryName]	[TotalSales]	[%Sales]
Accessories	Bike Racks	39360	0.05616759
Accessories	Bike Stands	39591	0.05649723
Accessories	Bottles and Cages	56798.19	0.08105228
Accessories	Cleaners	7218.6	0.0103011
Accessories	Fenders	46619.58	0.06652717
Accessories	Helmets	225335.6	0.3215589
Accessories	Hydration Packs	40307.67	0.05751994
Accessories	Tires and Tubes	245529.32	0.35037578
Bikes	Mountain Bikes	9952759.564	0.35146228
Bikes	Road Bikes	14520584.04	0.51276608
Bikes	Touring Bikes	3844801.05	0.13577164
Clothing	Caps	19688.1	0.05794493

Clothing	Gloves	35020.7	0.10307099
Clothing	Jerseys	172950.68	0.5090189
Clothing	Shorts	71319.81	0.20990453
Clothing	Socks	5106.32	0.01502864
Clothing	Vests	35687	0.10503201

```
Formal Sales with KF = CALCULATE(SUM(Data[Sales
Amount]),REMOVEFILTERS(Data[Product Category] = "Formal"))
```

38] Moving Average

Go to Quick Measure and select rolling average in time intelligence and in Rolling Average put data column in base data and date in date pane and then select period as per your convenience and in periods before put 2 if you want to find out 3 month moving average.

In the period after put 0.

39] DAX - Values Function

When the input parameter is a column name, returns a one-column table that contains the distinct values from the specified column. Duplicate values are removed and only unique values are returned. A BLANK value can be added. When the input parameter is a table name, returns the rows from the specified table. Duplicate rows are preserved. A BLANK row can be added.

Note

This function cannot be used to Return values into a cell or column on a worksheet; rather, you use it as an intermediate function, nested in a formula, to

get a list of distinct values that can be counted or used to filter or sum other values.

Syntax

DAX

Copy

```
VALUES(<TableNameOrColumnName>)
```

Parameters

Term	Definition
TableName or ColumnName	A column from which unique values are to be returned, or a table from which rows are to be returned.

Return value

When the input parameter is a column name, a single column table. When the input parameter is a table name, a table of the same columns is returned.

Remarks

- **When you use the VALUES function in a context that has been filtered, the unique values returned by VALUES are affected by the filter. For example, if you filter by Region, and return a list of the values for City, the list will include only those cities in the regions permitted by the filter. To return all of the cities, regardless of existing filters, you must**

use the ALL function to remove filters from the table. The second example demonstrates use of ALL with VALUES.

- This function is not supported for use in DirectQuery mode when used in calculated columns or row-level security (RLS) rules.
- For best practices when using VALUES, see Use SELECTEDVALUE instead of VALUES.

Related functions

In most scenarios, when the argument is a column name, the results of the VALUES function are identical to those of the DISTINCT function. Both functions remove duplicates and return a list of the possible values in the specified column. However, the VALUES function can also return a blank value. This blank value is useful in cases where you are looking up distinct values from a related table, but a value used in the relationship is missing from one table. In database terminology, this is termed a violation of referential integrity. Such mismatches in data can occur when one table is being updated and the related table is not.

When the argument is a table name, the result of the VALUES function returns all rows in the specified table plus a blank row, if there is a violation of referential integrity. The DISTINCT function removes duplicate rows and returns unique rows in the specified table.

Note

The DISTINCT function allows a column name or any valid table expression to be its argument but the VALUES function only accepts a column name or a table name as the argument.

The following table summarizes the mismatch between data that can occur in two related tables when referential integrity is not preserved.

MyOrders table	MySales table

June 1	June 1 sales
June 2	June 2 sales
(no order dates have been entered)	June 3 sales

If you use the DISTINCT function to return a list of dates, only two dates would be returned. However, if you use the VALUES function, the function returns the two dates plus an additional blank member. Also, any row from the MySales table that does not have a matching date in the MyOrders table will be "matched" to this unknown member.

Example

The following formula counts the number of unique invoices (sales orders), and produces the following results when used in a report that includes the Product Category Names:

DAX

Copy

```
= COUNTROWS(VALUES('InternetSales_USD'[SalesOrderNumber]))
```

Returns

Row Labels	Count Invoices
Accessories	18,208

Bikes	**15,205**
Clothing	**7,461**
Grand Total	**27,659**