IT Real-Time training that work for your career.

PROVIDED TRAINING FOR THOUSANDS OF STUDENTS

SUBJECT, MATERIAL & VIDEOS

VIDEOS PROVIDED

POWER BI

DAX INDIVIDUAL FUNCTIONS

MSBI
IS, AS, RS & MDS

POWER BI
SERVER, DESKTOP & DAX

Trainings:

CLASS ROOM

ONLINE

FAST TRACK

ONE ON ONE

PROJECT TRAINING

Address:

Flat No: 506/B
Nilgiri Block
Aditya Enclave
Mytrivanam Area
Hyderabad.

Website & Blog

www.vinaytechhouse.com

www.msbivinay.blogspot.in

Contact Information

+91 9573168449

040 66638869

Sri VINAY TechHouse

DATE FUNCTIONS [Deal with date values and give portion of date / date output]

DATE FUNCTIONS	DESC	
CALENDAR	Generate calendar from the **given dates**	Table / colum
CALENDARAUTO	Automatically generate calendar based on **your data model dates** availability	Table/ Columr
DAY	Day from the date value	Measure/ Col
MONTH	Month from the date value	Measure/ Col
YEAR	Year from the date value	Measure/ Col
HOUR	Hour from Date Time	Measure/ Col
MINUTE	Minute from Date Time	Measure/ Col
SECOND	Second from Date Time	Measure/ Col
YEARFRAC	Fraction of year based on the result, rounded to the upper integer.	Measure/ Col
DATE	Consider three values as date	Measure/ Col
TIME	Consider three values as time	Measure/ Col
DATEVALUE	Consider string as date	Measure/ Col
TIMEVALUE	Consider string as time	Measure/ Col
EDATE	End of date. Add months to the date. Ex: End date of a policy based on period	Measure/ Col
EOMONTH	End of month (After adding months gives you the last day in the month)	Measure/ Col
WEEKDAY	Weekday of the given date { Value between 1-7, 1-Sun…7-Sat}	Measure/ Col
WEEKNUM	Current week month, by considering 52 weeks in a year	Measure/ Col
TODAY	Current date with time 12AM	Measure/ Col
UTCTODAY	Universal Time Coordinator date and time (12AM) London Time	Measure/ Col
NOW	Current date with current time	Measure/ Col
UTCNOW	Universal Time Coordinator date and current time London Time	Measure/ Col
DATEDIFF	Differences between two specified date in the form of interval	Measure/ Col

New column [Calculated field	New measure [Calculated measure]	New Table	
Adding new column	Adding a measure to the query	**One or more colun as new table**	
Row by row operation and retu Multiple rows	Set of rows operation and return sin row		
More memory required due to model operation	More CPU required due to many ro store and Time consuming process		

[Create new measure for each function and place the measure on the card, and preview the result]

a)Click New Measure, Write the below expression, click Save in the left.

Create new measure and place the measure on the card, and preview the result

=Today()

Create new measure and place the measure on the card, and preview the result

=NOW()

Create new measure and place the measure on the card, and preview the result

=UTCTODAY()

Create new measure and place the measure on the card, and preview the result

=UTCNOW()

Create new measure and place the measure on the card, and preview the result

=DAY(TODAY())

Create new measure and place the measure on the card, and preview the result

=MONTH(TODAY())

Create new measure and place the measure on the card, and preview the result

=YEAR(TODAY())

Create new measure and place the measure on the card, and preview the result

=HOUR(NOW())

Create new measure and place the measure on the card, and preview the result

=MINUTE(NOW())

Create new measure and place the measure on the card, and preview the result

=SECOND(NOW())

Create new measure and place the measure on the card, and preview the result

=YEARFRAC("Jan 1 2007","Mar 1 2009")

Create new measure and place the measure on the card, and preview the result

=YEARFRAC("Jan 1 2007","Mar 1 2009",3)

Create new measure and place the measure on the card, and preview the result

=EDATE(NOW(),180) --180 MONTHS ADDING

Create new measure and place the measure on the card, and preview the result

=EOMONTH(NOW(),2.5) --AFTER ADDING 2.5 MONTHS, THAT CURRENT MONTH LAST DATE

Create new measure and place the measure on the card, and preview the result

=DATEDIFF(DATE(2019,01,01),DATE(2019,09,01))

Create new measure and place the measure on the card, and preview the result

=DATE(2019,01,01)

Create new measure and place the measure on the card, and preview the result

=TIME(10, 05, 02)

Create new measure and place the measure on the card, and preview the result

=DATEVALUE("2019-01-01")

Create new measure and place the measure on the card, and preview the result

=TIMEVALUE("10:03:24")

Create new measure and place the measure on the card, and preview the result

=WEEKNUM(Now())

Create new measure and place the measure on the card, and preview the result

=WEEKDAY(NOW(),1)

Note:

You can apply the above functions in your new column operations or existing columns

[Create new table for the below functions, go to data view and see the table data]

Create new table for the below function, go to data view and see the table data

=CALENDAR(date(2019,01,01),date(2019,08,31))

Create new table for the below function, go to data view and see the table data

=CALENDARAUTO()

Practice the above functions in simple way using DAX Query at DAX Studio / SSMS

Run the below queries at DAX Studio and see the result


```
//Calendar will take two dates and give you a list of dates between them

EVALUATE
(
CALENDAR(date(2017,01,01),date(2018,01,01))
)


//Calendar auto will take dates from the model

EVALUATE
(
CALENDARAUTO()

)
```

Run the below DAX Query in the DAX Studio and view the result

```
/*

select Today() as "Today date", NOW() as "Today date and time",UTCTODAY() as "UK Date"
 FROM <tablename>

 */

EVALUATE
(
    ROW
    (
    "Today date",Today(),
     "Today date and time", NOW(),
     "UK Date", UTCTODAY(),
     "UK current date and time", UTCNOW(),
     "Year",year(today()),
     "month",month(today()),
     "Day",day(today()),
     "hour",hour(now()),
     "minute",minute(now()),
     "second", second(now()),
     "Adding months", edate(now(),5),
     "end of month",eoMONTH(now(),5),
     "weekday", weekday(today()),
     "yearfrac",YEARFRAC(date(2017,01,01),date(2019,08,02)),
     "datediff",DATEDIFF(date(2017,01,01),date(2019,08,03),year),
     "timevalue", timevalue("18:37:04"),
     "Date value",DATEVALUE("2020-10-05")
    )
)
```


Note: Type / copy and paste the above query, click Run / Press 'F5'

Create Date Table Manually [Trainer constructured using various functions]

In the below function, you will learn multiple things

a) ADDCOLUMN
b) FORMAT
c) CALENDAR

YYYY- FULL YEAR (EX: 2021)

YY-SHORT YEAR (EX: 21)

MM-SHORT MONTH (EX: 04)

MMM-MONTH TEXT SHORT (EX: APR)

MMMM-MONTH TEXT FULL (EX: APRIL)

DD-DAY SHORT (EX: 01)

DDD-DAY TEXT SHORT (EX: THU)

DDDD-DAY TEXT LONG (EX: THURSDAY)

Create New table in the Power BI Desktop and write the below, save, see the tabe in the data view

DimDate=

ADDCOLUMNS

(

CALENDAR (DATE(2000,1,1), DATE(2025,12,31)),

"DateAsInteger", FORMAT ([Date], "**YYYYMMDD**"),

"Year", YEAR ([Date]),

"Monthnumber", **FORMAT** ([Date], "**MM**"),

"YearMonthnumber", **FORMAT** ([Date], "**YYYY/MM**"),

"YearMonthShort", **FORMAT** ([Date], "**YYYY/mmm**"),

"MonthNameShort", FORMAT ([Date], "**mmm**"),

"MonthNameLong", FORMAT ([Date], "**mmmm**"),

"DayOfWeekNumber", **WEEKDAY** ([Date]),

"DayOfWeek", FORMAT ([Date], "**dddd**"),

"DayOfWeekShort", FORMAT ([Date], "**ddd**"),

"Quarter", **"Q" & FORMAT ([Date], "Q"**),

"YearQuarter", FORMAT ([Date], "**YYYY**") & "/Q" & FORMAT ([Date], "**Q**"

)

Run the below query in the DAX STUDIO to generate a Date Table

```
EVALUATE
(
ADDCOLUMNS
(
CALENDAR (DATE(2000,1,1), DATE(2025,12,31)),

"DateAsInteger", FORMAT ( [Date], "YYYYMMDD" ),

"Year", YEAR ( [Date] ),

"Monthnumber", FORMAT ( [Date], "MM" ),

"YearMonthnumber", FORMAT ( [Date], "YYYY/MM" ),

"YearMonthShort", FORMAT ( [Date], "YYYY/mmm" ),

"MonthNameShort", FORMAT ( [Date], "mmm" ),

"MonthNameLong", FORMAT ( [Date], "mmmm" ),

"DayOfWeekNumber", WEEKDAY ( [Date] ),

"DayOfWeek", FORMAT ( [Date], "dddd" ),

"DayOfWeekShort", FORMAT ( [Date], "ddd" ),

"Quarter", "Q" & FORMAT ( [Date], "Q" ),

"YearQuarter", FORMAT ( [Date], "YYYY" ) & "/Q" & FORMAT ( [Date], "Q" )

)

)
```

PARENT CHILD FUNCTIONS OVERVIEW

What are the advantages of Parent and Child Functions?

To identify the immediate parent / parents for a child, to create hierarchies, for finding the position of a parent etc.

PATH FUNCTIONS	Description
Path	Gives you the hierarchy path starting from higher level
pathlength	Number of levels in the path
pathcontains	Verifying an item availability in the path [Return TRUE/FALSE]
pathItemeverse	Value at a position in the path from right to left (after reversing path)
PathItem	Value at a position in the path from left to right

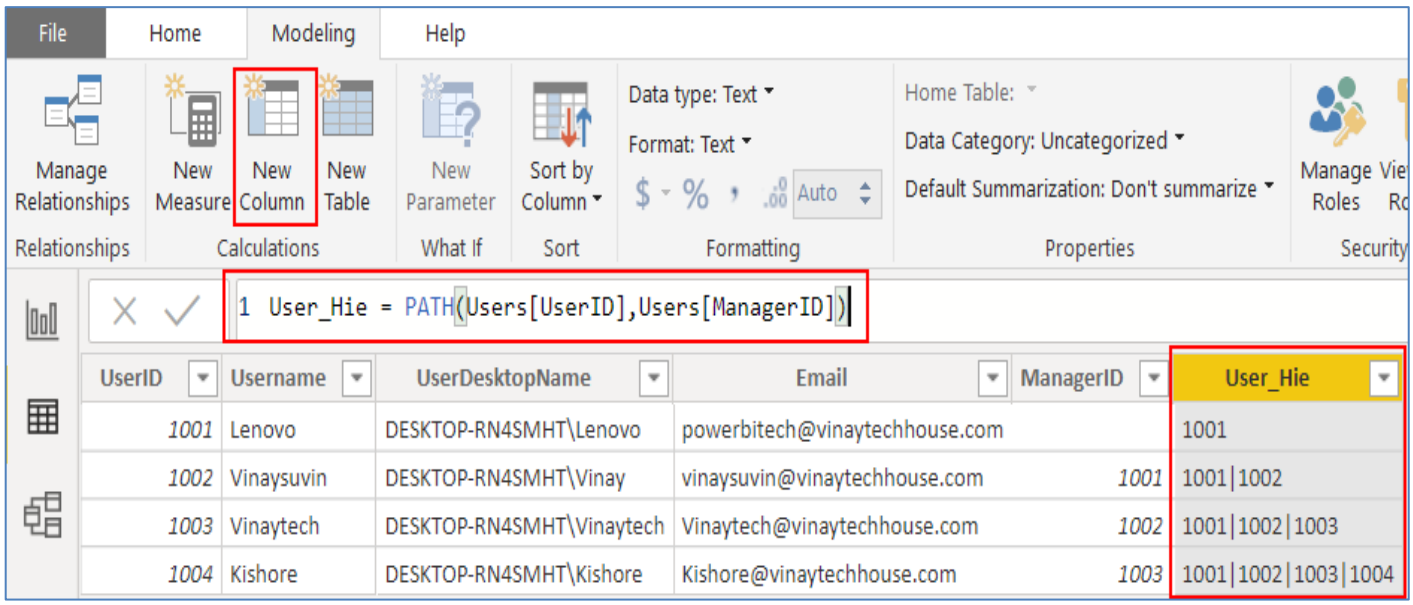
Run the below queries in the DAX Studio

```
EVALUATE
(
ADDCOLUMNS(
DimUsers,
"Parent Path", path(DimUsers[UserID],DimUsers[ManagerID]),

"Path length1", PATHLENGTH(path(DimUsers[UserID],DimUsers[ManagerID])),

"2nd Positioned user left to
right",PATHITEM(path(DimUsers[UserID],DimUsers[ManagerID]),2,1),

"2nd positioned item from right to
left",PATHITEMREVERSE(path(DimUsers[UserID],DimUsers[ManagerID]),2,1),

"Item available", PATHCONTAINS(path(DimUsers[UserID],DimUsers[ManagerID]),1003)
)
)
```

Go to Users table in Dataview

Path Function Practice

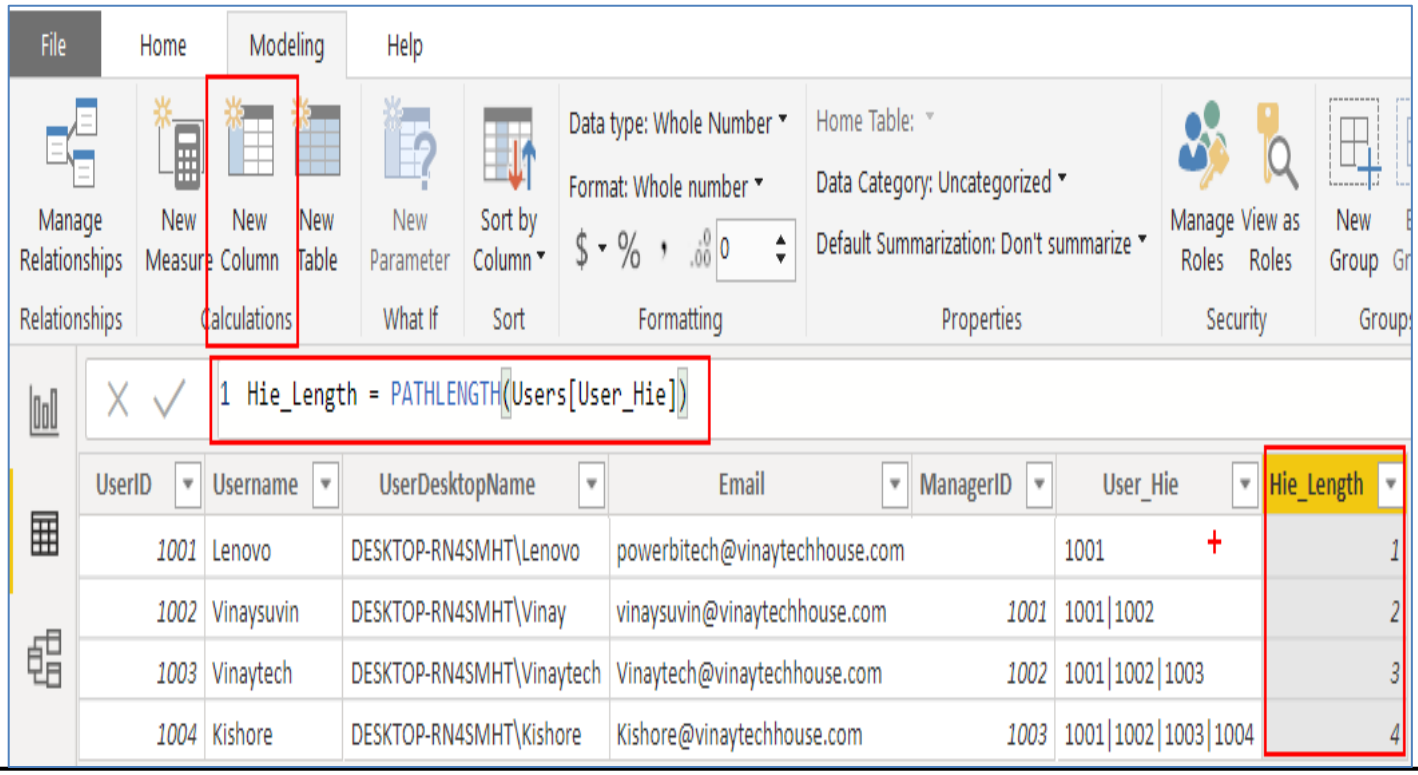
Go to DimCourse Table→Add new column and write the below expression



UserID	Username	UserDesktopName	Email	ManagerID	User_Hie
1001	Lenovo	DESKTOP-RN4SMHT\Lenovo	powerbitech@vinaytechhouse.com		1001
1002	Vinaysuvin	DESKTOP-RN4SMHT\Vinay	vinaysuvin@vinaytechhouse.com	1001	1001|1002
1003	Vinaytech	DESKTOP-RN4SMHT\Vinaytech	Vinaytech@vinaytechhouse.com	1002	1001|1002|1003
1004	Kishore	DESKTOP-RN4SMHT\Kishore	Kishore@vinaytechhouse.com	1003	1001|1002|1003|1004

Pathlength Function Practice

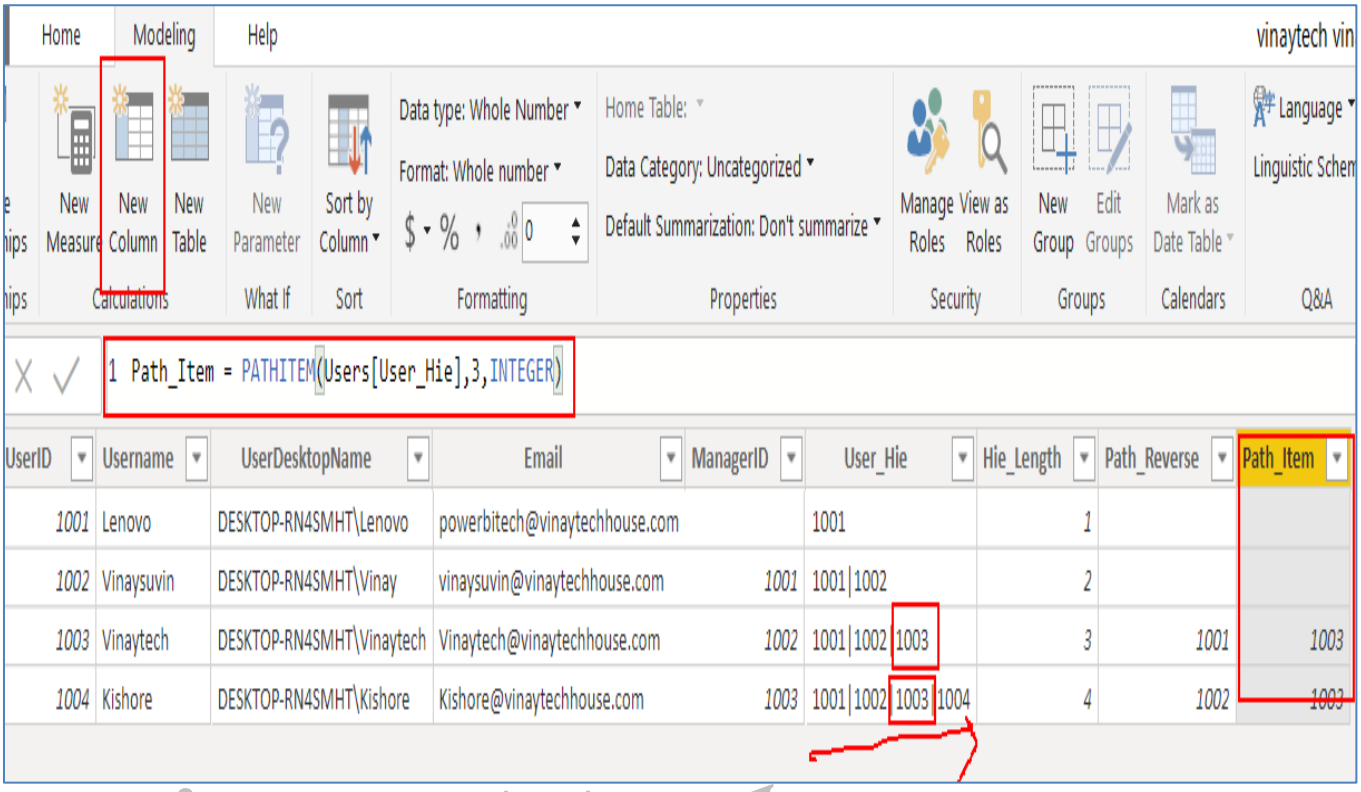
Go to DimCourse Table →Add new column and write the below expression



UserID	Username	UserDesktopName	Email	ManagerID	User_Hie	Hie_Length
1001	Lenovo	DESKTOP-RN4SMHT\Lenovo	powerbitech@vinaytechhouse.com		1001	1
1002	Vinaysuvin	DESKTOP-RN4SMHT\Vinay	vinaysuvin@vinaytechhouse.com	1001	1001|1002	2
1003	Vinaytech	DESKTOP-RN4SMHT\Vinaytech	Vinaytech@vinaytechhouse.com	1002	1001|1002|1003	3
1004	Kishore	DESKTOP-RN4SMHT\Kishore	Kishore@vinaytechhouse.com	1003	1001|1002|1003|1004	4

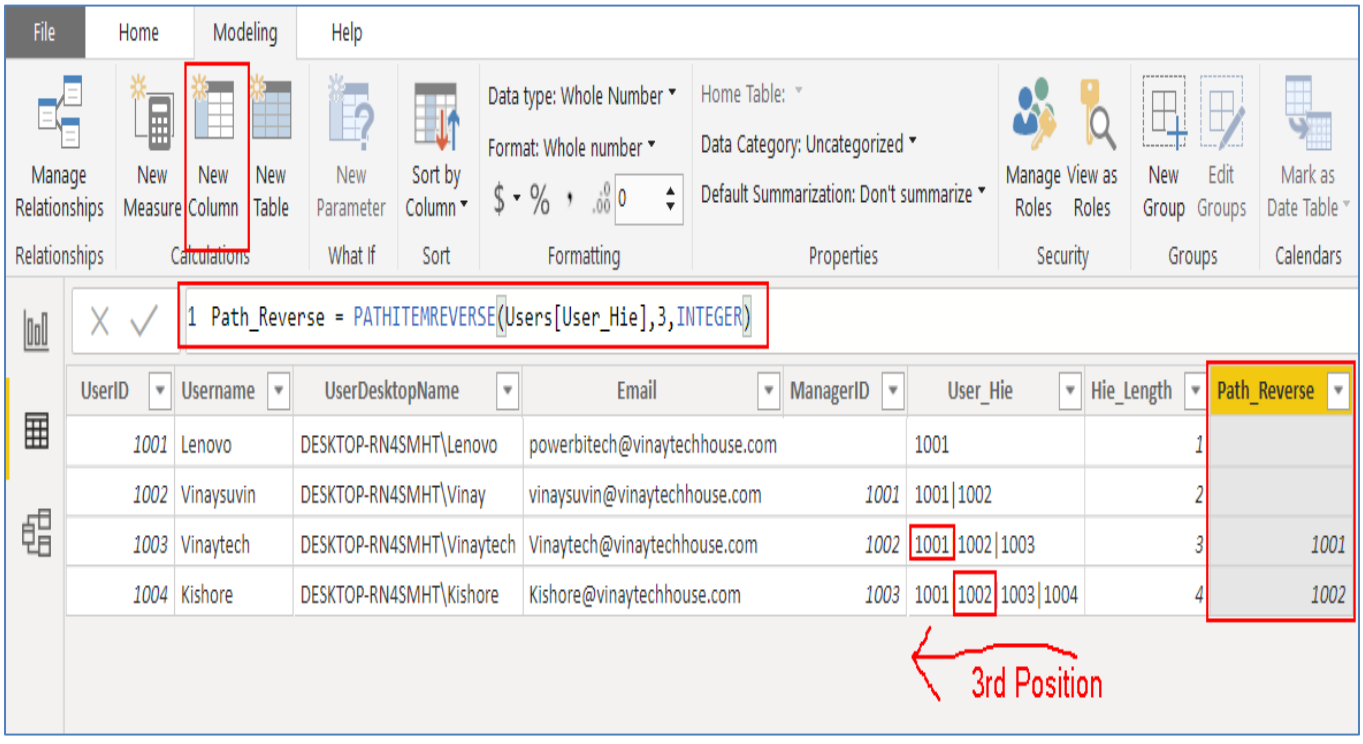
PathItem Practice

Go to DimCourse Table →Add new column and write the below expression



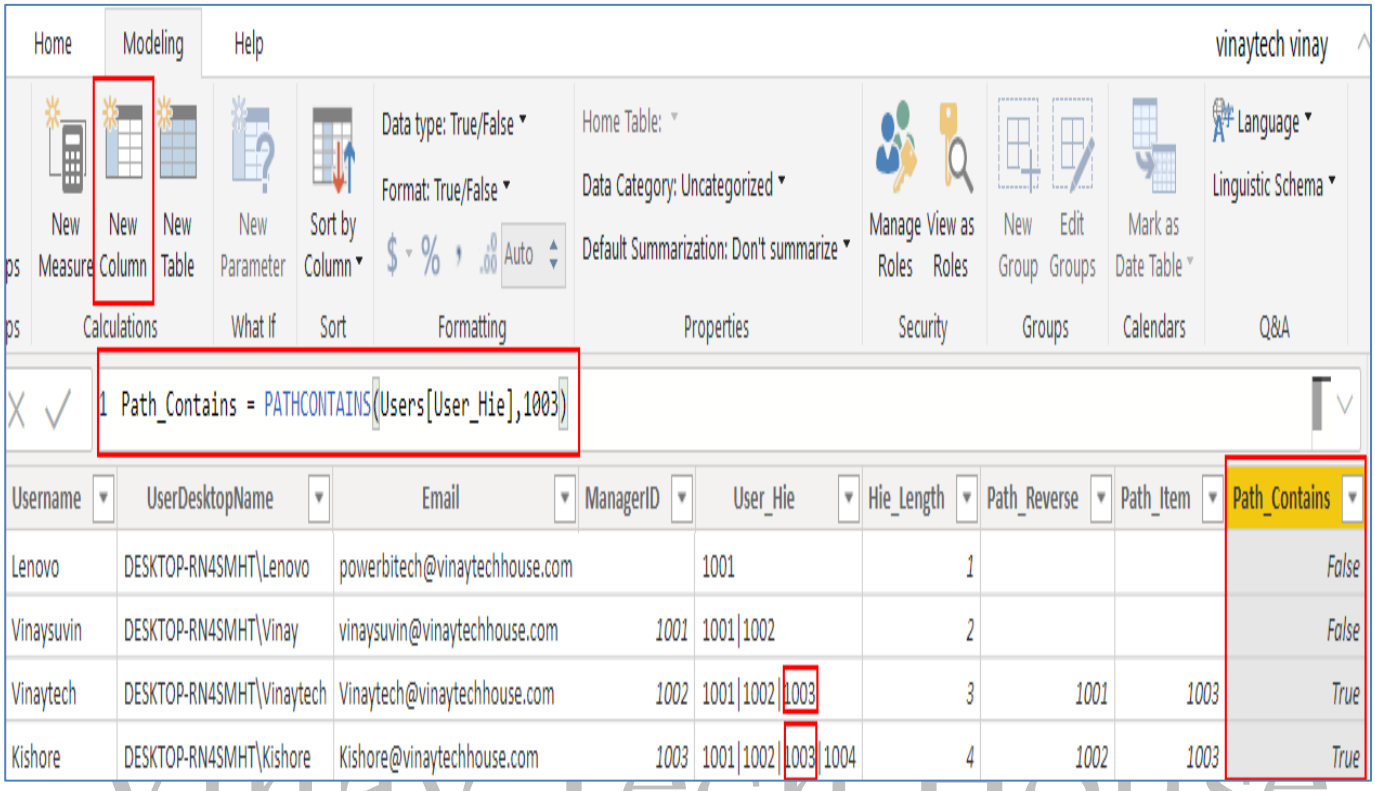
PathItemReverse Function Practice

Go to DimCourse Table →Add new column and write the below expression



PathContains Function Practice

Go to DimCourse Table →Add new column and write the below expression



ADDITIONALS

PATH

PATH function (DAX) -Returns a delimited text with the identifiers of all the parents to the current row, startingwith the oldest or top most until current.

EMPLOYEEKEY	PARENTEMPLOYEEKEY	PATH
14	112	112|14
3	14	112|14|3
11	3	112|14|3|11
13	3	112|14|3|13
162	3	112|14|3|162
117	162	112|14|3|162|117
221	162	112|14|3|162|221
81	162	112|14|3|162|81

Note:

Take values like below

eid ename mgrid

1 x

2 y 1

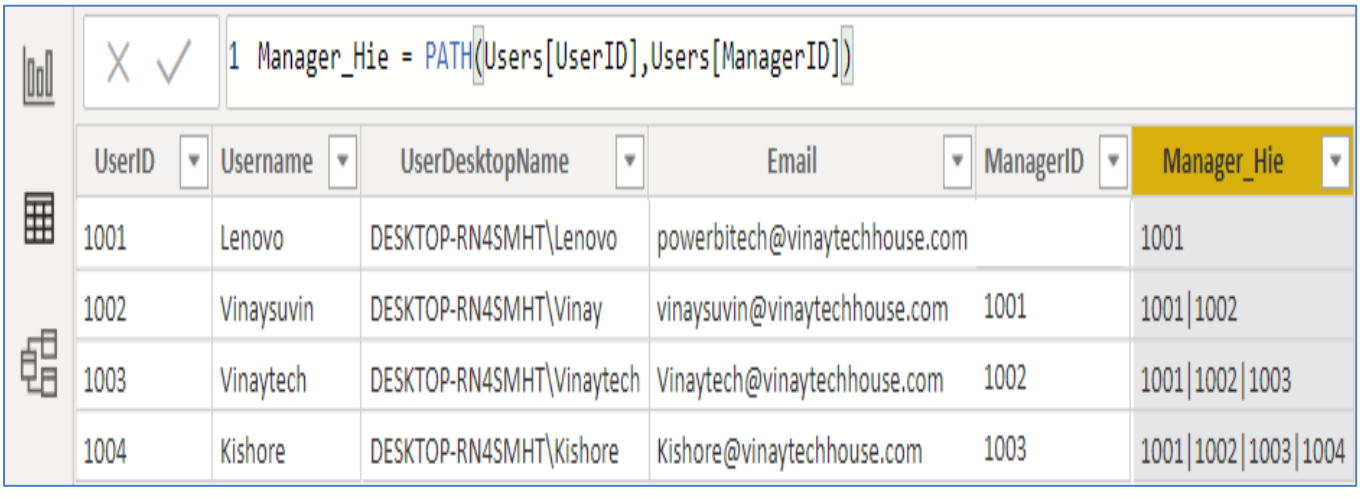
3 z 2

4 k 2

5 r 4

=Path([eid],[mgrid])

Class room: Managers_Hie = path(Users[UserID],Users[ManagerID])



UserID	Username	UserDesktopName	Email	ManagerID	Manager_Hie
1001	Lenovo	DESKTOP-RN4SMHT\Lenovo	powerbitech@vinaytechhouse.com		1001
1002	Vinaysuvin	DESKTOP-RN4SMHT\Vinay	vinaysuvin@vinaytechhouse.com	1001	1001|1002
1003	Vinaytech	DESKTOP-RN4SMHT\Vinaytech	Vinaytech@vinaytechhouse.com	1002	1001|1002|1003
1004	Kishore	DESKTOP-RN4SMHT\Kishore	Kishore@vinaytechhouse.com	1003	1001|1002|1003|1004

PATHCONTAINS

Returns TRUE if the specified item exists within the specified path.

Syntax:

PATHCONTAINS(<path>, <item>)

Ex:New caluclated column

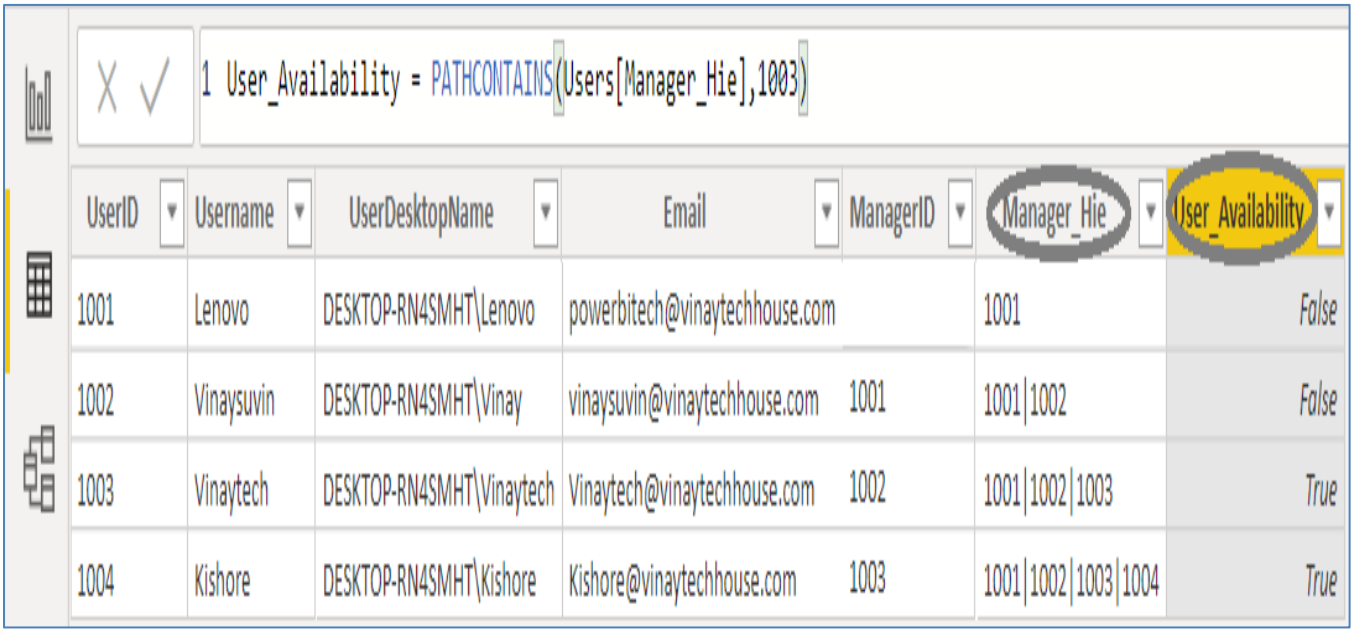
Add new column to the EMP_MGR Query [This query in Business_Details Sheet]

Scenario :The below gives you employee and hierarchical path

Managers hierarchy=path(EMP_MGR[EID],EMP_MGR[MGRID])

Scenario:The below gives you existence of 1003 in the hierarchy

UserID availability = PATHCONTAINS(path(Users[UserID],Users[ManagerID]),1003)



UserID	Username	UserDesktopName	Email	ManagerID	Manager_Hie	User_Availability
1001	Lenovo	DESKTOP-RN4SMHT\Lenovo	powerbitech@vinaytechhouse.com		1001	False
1002	Vinaysuvin	DESKTOP-RN4SMHT\Vinay	vinaysuvin@vinaytechhouse.com	1001	1001|1002	False
1003	Vinaytech	DESKTOP-RN4SMHT\Vinaytech	Vinaytech@vinaytechhouse.com	1002	1001|1002|1003	True
1004	Kishore	DESKTOP-RN4SMHT\Kishore	Kishore@vinaytechhouse.com	1003	1001|1002|1003|1004	True

Scenario:Identifying the length or number of levels managers available.

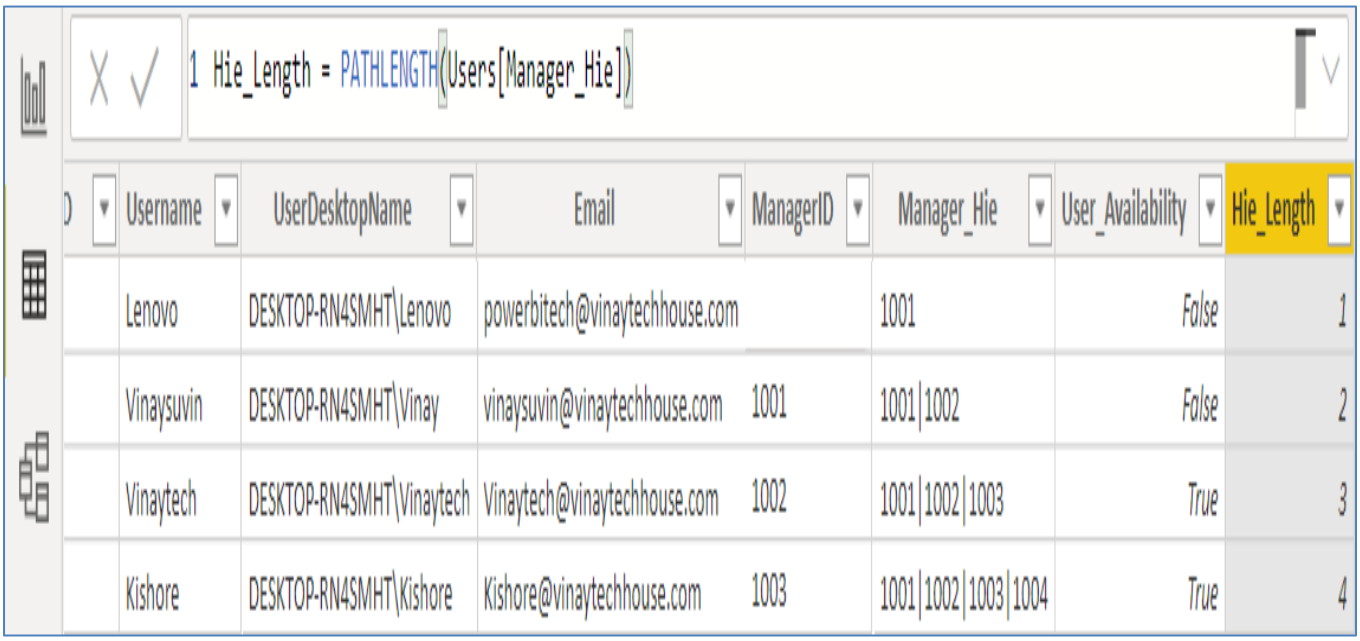
Pathlength= pathlength(path(Users[UserID],Users[ManagerID]))

PATHLENGTH

PATHLENGTH function (DAX) -Returns the number of levels in a given PATH(), starting at current level until theoldest or top most parent level. In the following example column PathLength is defined as ' =PATHLENGTH([Path]) '; the example includes all data from the Path() example to help understand how this function works.

EMPLOYEEKEY	PARENTEMPLOYEEKEY	PATH	PATHLENGTH
112		112	1
14	112	112|14	2
3	14	112|14|3	3
11	3	112|14|3|11	4
13	3	112|14|3|13	4
162	3	112|14|3|162	4
117	162	112|14|3|162|117	5
221	162	112|14|3|162|221	5
81	162	112|14|3|162|81	5

Levelsbelong to = PATHLENGTH(path(Users[UserID],Users[ManagerID]))



Username	UserDesktopName	Email	ManagerID	Manager_Hie	User_Availability	Hie_Length
Lenovo	DESKTOP-RN4SMHT\Lenovo	powerbitech@vinaytechhouse.com		1001	False	1
Vinaysuvin	DESKTOP-RN4SMHT\Vinay	vinaysuvin@vinaytechhouse.com	1001	1001|1002	False	2
Vinaytech	DESKTOP-RN4SMHT\Vinaytech	Vinaytech@vinaytechhouse.com	1002	1001|1002|1003	True	3
Kishore	DESKTOP-RN4SMHT\Kishore	Kishore@vinaytechhouse.com	1003	1001|1002|1003|1004	True	4

PATHITEM

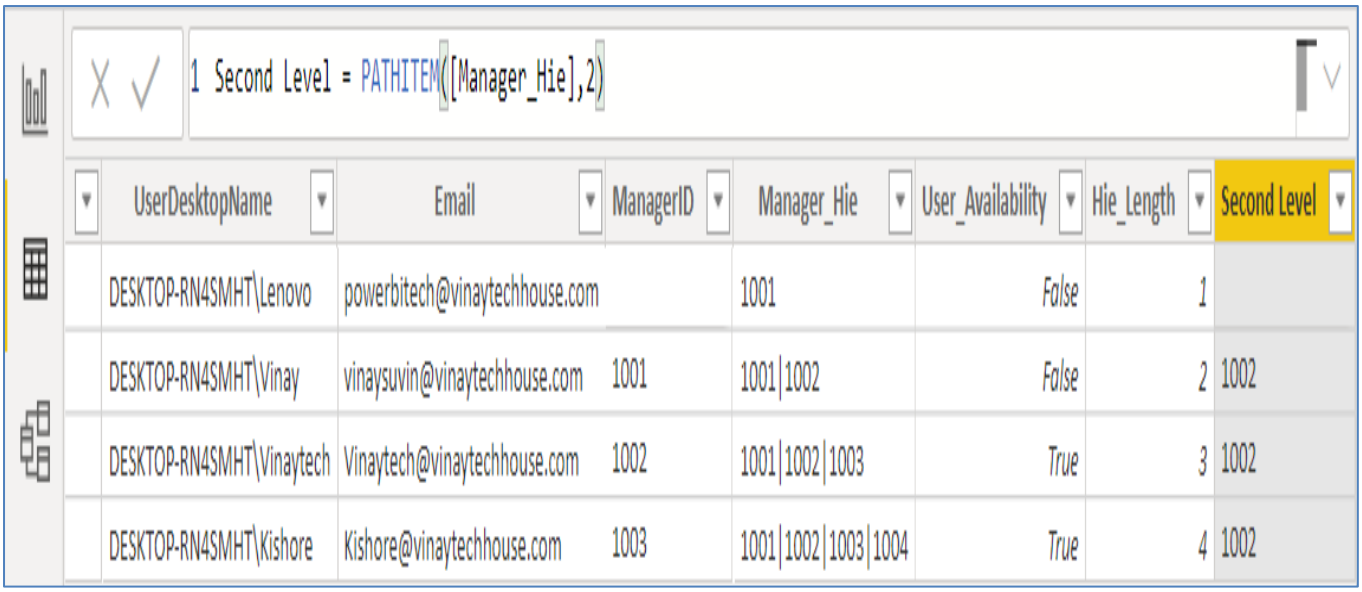
PATHITEM function (DAX) -**Returns the item at the specified position from a PATH()** like result, counting from leftto right. In the following example column PathItem - 4th from left is defined as '

=PATHITEM([Path], 4) ';

this example returns the EmployeKey at fourth position in the Path string from the left, using the same sample data from the Path() example.

EMPLOYEEKEY	PARENTEMPLOYEEKEY	PATH	PATHITEM - 4TH FROM LEFT
112		112	
14	112	112|14	

EMPLOYEEKEY	PARENTEMPLOYEEKEY	PATH	PATHITEM - 4TH FROM LEFT
3	14	112|14|3	
11	3	112|14|3|11	11
13	3	112|14|3|13	13
162	3	112|14|3|162	162
117	162	112|14|3|162|117	162
221	162	112|14|3|162|221	162
81	162	112|14|3|162|81	162



UserDesktopName	Email	ManagerID	Manager_Hie	User_Availability	Hie_Length	Second Level
DESKTOP-RN4SMHT\Lenovo	powerbitech@vinaytechhouse.com		1001	False	1	
DESKTOP-RN4SMHT\Vinay	vinaysuvin@vinaytechhouse.com	1001	1001|1002	False	2	1002
DESKTOP-RN4SMHT\Vinaytech	Vinaytech@vinaytechhouse.com	1002	1001|1002|1003	True	3	1002
DESKTOP-RN4SMHT\Kishore	Kishore@vinaytechhouse.com	1003	1001|1002|1003|1004	True	4	1002

PATHITEMREVERSE

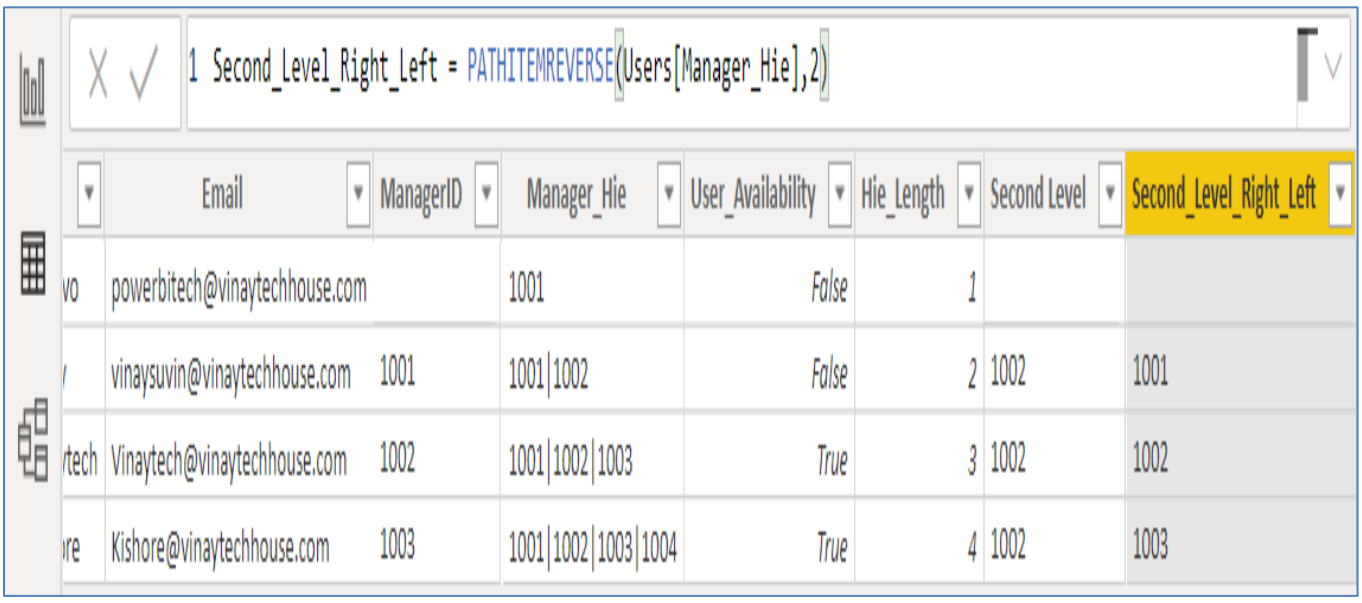
PATHITEMREVERSE function (DAX) -**Returns the item at *position* from a PATH() like function result, counting backwards from right to left.**

In the following example column PathItemReverse - 3rd from right is defined as '

=PATHITEMREVERSE([Path], 3) ';

this example returns the EmployeKey at third position in the Path string from the right, using the same sample data from the Path() example.

EMPLOYEEKEY	PARENTEMPLOYEEKEY	PATH	PATHITEMREVERSE - 3RD FROM RIGHT
112		112	
14	112	112|14	
3	14	112|14|3	112
11	3	112|14|3|11	14
13	3	112|14|3|13	14
162	3	112|14|3|162	14
117	162	112|14|3|162|117	3
221	162	112|14|3|162|221	3
81	162	112|14|3|162|81	3



TEXT FUNCTIONS OVERVIEW

When do we go for Text Functions?

To perform different textual operations (such as Upper, Lower, Trim etc...), these are helpful.

Data Analysis Expressions (DAX) includes a set of text functions that is based on the library of string functions in Excel, but which has been modified to work with tables and columns.

This section lists all text functions available in the DAX language.

TEXT FUNCTIONS	Description
BLANK	Returns blank value
CODE	Return code for the character
UNICHAR	Returns the unichar for the code specified
CONCATENATE	Concatinate two strings only [it will not take delimiter]
COMBINEVALUES	Concatenate the values with the specicifed delimiter
CONCATENATEX	Based on expression [row wise] concatenation
TRIM	Trimming left and right hand side spaces
REPT	Repition of value to the specicifed number of times
REPLACE	Replace the specicifed text (position to the number of chars) with another text {one time only}
SUBSTITUTE	Replace the specified text with the given text {multiple times}
Lower	Convert to lowercase letters
Upper	Convert to uppercase letters
Exact	Returns true if both the arguments match.
FIND	Case sensitive textual values match {return position of the text existed}
SEARCH	Case insensitive, accent sensitive textual values match { return the first position of the character}
FORMAT	Format numerics, strings, date values to another format {it is not conversion of values, simply the appearance will change}
MID	Middle characters {specicifed position to the length}. Equals to SUBSTRING in other applications
LEFT	Specified left hand side characters in the string
RIGHT	Specified right hand side characters in the string
VALUE	Consider string as numeric value

Run the below query in the DAX studio and see the result

```
EVALUATE
(
ROW
(
"Code of char", UNICODE("a"), -- 65

"Char for code",UNICHAR(65), -- a

"left data", left("vinaykumar",3),  --vin (left side 3 chars)

"right data", right("vinaykumar",4), --umar (right side 4 chars)

"trim data", trim("       vinay kumar       "), --vinaykumar (left and right spaces removal)

"replace data", REPLACE("vinaykumar",3,2,"NA"),

--viNAykumar (replace 3rd char to two chars with NA)

"replace all places",SUBSTITUTE("vinay kumar","a","AAA"),

--vinAAAy kumAAAr(replace 'a' with 'AAA')

"string as value", value("123"), --123 (string 123 considered as numeric 123)

"case sensitive search",find("na","VinayKumar"), --3  case sensitive search ('na' first occurance)

"case insensitive search",search("NA","VinayKumar"),

--3 case insensitive search ('na' first occurance)

"format with month name", format(now(),"YYYY MMMM DDD"),-

-"2021 May Tue"  MM, MMM, MMMM, DD, DDD, DDDD,YY, YYYY
```

"format with month short name", format(now(),"MMM, DD, YYYY"),--"May, 11, 2021"

"lower case data", lower("VINAY KUMAR"), --vinay kumar

"upper case data", upper("vinay kumar"), --VINAY KUMAR

"repeting 3 times", rept("vinay",3), --vinayvinayvinay (repeating 3 times)

"concatinating without delimiter", CONCATENATE("vinay","kumar"),

--vinaykumar (combibining without delimiter)

"concating with delimiter", COMBINEVALUES("/","vinay","kumar"),

--vinay/kumar (combining with delimiter)

"taking a part of string like sub string", mid("vinaykumar",3,4),

--nayk (get 3rd position to 4 chars)

"exactly matched or not", exact("vinay","VINAY") --FALSE (because lower case vinay not equal to upper case VINAY)

)

)

/*

DD:14 DDD: Fri DDDD: Friday MM:08 MMM: Aug MMMM: August YY:20 YYYY:2020

*/

Write the below Query under DAX Studio and Run

```
EVALUATE
(

ROW
(
"Code of char", UNICODE("a"),

"Char for code",UNICHAR(65),

"left data", left("vinaykumar",3),

"right data", right("vinaykumar",4),

"trim data", trim("       vinay kumar       "),

"replace data", REPLACE("vinaykumar",3,2,"NA"),

"replace all places",SUBSTITUTE("vinay kumar","a","AAA"),

"string as value", value("123"),

"case sensitive search",find("na","VinayKumar"),

"case insensitive search",search("NA","VinayKumar"),

"format with month name", format(now(),"YYYY MMMM DDD"),--MM, MMM, MMMM, DD, DDD, DDDD,YY, YYYY

"format with month short name", format(now(),"MMM, DD, YYYY"),

"lower case data", lower("VINAY KUMAR"),

"upper case data", upper("vinay kumar"),
```

```
"repeting 3 times", rept("vinay",3),

"concatinating without delimiter", CONCATENATE("vinay","kumar"),

"concating with delimiter", COMBINEVALUES("/","vinay","kumar"),

"taking a part of string like sub string", mid("vinaykumar",3,4),

"exactly matched or not", exact("vinay","VINAY")

)
)



/*

DD:14  DDD: Fri   DDDD: Friday  MM:08  MMM: Aug MMMM: August  YY:20  YYYY:2020

*/
```

Write the below Query under DAX Studio and Run

```
EVALUATE
(
Row(
"Left 3 chars ",left("vinaykuar",3),
"right 3 chars",right("vinay kumar",3),
"trimming left and right '    vinay   '", trim("    vinay    "),
"length of text vinay kumar", LEN("vinay kumar"),
"Code of char 'a'", UNICODE("a"),
"character for code 65", unichar(65),
"part of string from 3rd to 2 char from vinay kumar", mid("vinay kumar", 3,2),
--three arguments (text, start position, number of chars)
"Finding char case sensitive search from second position", find("a","vinaykumar",2),--2nd position
onwards "a" will be searched
"Finding char case insensitive search", search("A","vinay kumar",2),
"exact match check Raju RAJU", exact("Raju","RAJU"),
"exact match Raju Raju", exact("Raju","Raju"),
"Repeat 3 times raju",rept("raju",3),
"Repeat 0 5 times",rept(0,5),
"Replace Text with other from2 position to three chars one time",
 replace("vinay kumar",2,3,"ABC"),
"Replace text multiple times substitute a with 'A'", substitute("vinay kumar","a","A"),
"combine values with delimiter",COMBINEVALUES(",","vinay","kumar"),
"concatenate values without delimiter", CONCATENATE("vinay","kumar"),
"consider string as value '123'",  value("123"),
"formating date YYYY MMM/DD", format(today(),"YYYY MMM/DD"),
"Formating date MMMM DD, YYYY",format(today(),"MMMM DD, YYYY"),
"upper case data vinay kumar", upper("vinay kumar"),
"lower case data VINAY KUMAR", lower ("VINAY KUMAR")
)
)
```

/*

DD-DAY NUMERIC EX: 18

DDD-SHORT DAY STRING EX: MON

DDDD-FULL DAY STRING EX: MONDAY

MM-MONTH NUMERIC EX: 02

MMM-MONTH SHORT STRING EX: JUL

MMMM-MONTH FULL STRING EX: JULY

YY-TWO DIGIT YEAR NUMERIC EX: 20

YYYY-FULL YEAR EX: 2020

*/

Detailed Practice of all functions

BLANK CODE CONCATENATE FORMAT ✓ Pre-Defined Numeric Formats for the FORMAT function ✓ Custom Numeric Formats for the FORMAT function ✓ Pre-defined date and time formats for the FORMAT function ✓ Custom date and time formats for the FORMAT function	LEFT LEN LOWER MID REPLACE FIND FIXED CONCATENATEX EXACT	REPT RIGHT SEARCH SUBSTITUTE TRIM UNICHAR UPPER VALUE

BLANK

Returns Blank

Syntax

BLANK()

Simple Example:

Create a measure and place on a card

= IF(25-25=0,BLANK())

Other Examples:

The following example illustrates how you can work with blanks in formulas. The formula calculates the ratio of sales between the Resellers and the Internet channels. However, before attempting to calculate the ratio the denominator should be checked for zero values. If the denominator is zero then a blank value should be returned; otherwise, the ratio is calculated.

```
=IF( SUM(InternetSales_USD[SalesAmount_USD])= 0    , BLANK()    ,
SUM(ResellerSales_USD[SalesAmount_USD])/SUM(InternetSales_USD[SalesAmount_US
D])

                                                                    )
```

CODE

Returns a numeric code for the first character in a text string. The returned code corresponds to the character set used by your computer.

OPERATING ENVIRONMENT	CHARACTER SET
Macintosh	Macintosh character set
Windows	ANSI

Syntax

CODE(text)

Parameters

TERM	DEFINITION
text	The text for which you want the code of the first character.

Return value

A numeric code for the first character in a text string.

Example

Create a measure and place on a card

FORMULA	DESCRIPTION	RESULT
=CODE("A")	Displays the numeric code for A	65
=CODE("!")	Displays the numeric code for !	33

COMBINEVALUES

The COMBINEVALUES function joins two or more text strings into one text string. The primary purpose of this function is to support multi-column relationships in DirectQuery models, see **Remarks** for details.

Syntax

COMBINEVALUES(<delimiter>, <expression>, <expression>[, <expression>]...)

Examples:

Create a new column and see in the Data View

=Combinevalues(“,”, [CourseID],[Courname])

Result: 1,MSBI-F

Create a measure and place on a card

=COMBINEVALUES(",","ramana","reddy")

Result: ramana,reddy

Run the following DAX query in DAX Studio:

```
EVALUATE DISTINCT(SELECTCOLUMNS(DimDate, "Month", COMBINEVALUES(",",

[MonthName], [CalendarYear])))
```

CONCATENATE

Joins two text strings into one text string.

Syntax

CONCATENATE(<text1>, <text2>)

Example: Concatenation of Literals

Description

The sample formula creates a new string value by combining two string values that you provide as arguments.

Code

Create a measure and place on a card

```
=CONCATENATE("Hello ", "World")
```


Example: Concatenation of Strings in Columns

Description

The sample formula returns the customer's full name as listed in a phone book. Note how a nested function is used as the second argument. This is one way to concatenate multiple strings, when you have more than two values that you want to use as arguments.

Code

```
=CONCATENATE(Customer[LastName], CONCATENATE(", ", Customer[FirstName]))
```

Example: Conditional Concatenation of Strings in Columns

Description

The sample formula creates a new calculated column in the Customer table with the full customer name as a combination of first name, middle initial, and last name. If there is no middle name, the last name comes directly after the first name. If there is a middle name, only the first letter of the middle name is used and the initial letter is followed by a period.

Code

```
=CONCATENATE( [FirstName]&" ", CONCATENATE( IF(

LEN([MiddleName])>1, LEFT([MiddleName],1)&" ", ""), [LastName]))
```

Comments

This formula uses nested CONCATENATE and IF functions, together with the ampersand (&) operator, to conditionally concatenate three string values and add spaces as separators.

Example: Concatenation of Columns with Different Data Types

The following example demonstrates how to concatenate values in columns that have different data types. If the value that you are concatenating is numeric, the value will be implicitly converted to text. If both values are numeric, both values will be cast to text and concatenated as if they were strings.

PRODUCT DESCRIPTION	PRODUCT ABBREVIATION (COLUMN 1 OF COMPOSITE KEY)	PRODUCT NUMBER (COLUMN 2 OF COMPOSITE KEY)	NEW GENERATED KEY COLUMN
Mountain bike	MTN	40	MTN40
Mountain bike	MTN	42	MTN42

Code

```
=CONCATENATE('Products'[Product abbreviation],'Products'[Product number])
```

Comments

The CONCATENATE function in DAX accepts only two arguments, whereas the Excel CONCATENATE function accepts up to 255 arguments. If you need to add more arguments, you can use the ampersand (&) operator. For example, the following formula produces the results, MTN-40 and MTN-42.

```
=[Product abbreviation] & "-" & [Product number]
```

CONCATENATEX

Concatenates the result of an expression evaluated for each row in a table.

Syntax

CONCATENATEX(<table>, <expression>, [delimiter])

Example

Employees table

FIRSTNAME	LASTNAME
Alan	Brewer
Michael	Blythe

Examples:

Create a new measure and place on a card

=CONCATENATEX(DimCourse,DimCourse[Coursename],",")

CONCATENATEX(Employees, [FirstName] & " " & [LastName], ",")

Returns "Alan Brewer, Michael Blythe"

EXACT

Compares two text strings and returns TRUE if they are exactly the same, FALSE otherwise.

EXACT is case-sensitive but ignores formatting differences. You can use EXACT to test text being entered into a document.

Syntax

EXACT(<text1>,<text2>)

Parameters

TERM	DEFINITION
text1	The first text string or column that contains text.
text2	The second text string or column that contains text.

Return value

True or false. (Boolean)

Simple Example:

Create a new measure and place on a card

=EXACT(50,50)

Output:

True

Example

The following formula checks the value of Column1 for the current row against the value of Column2 for the current row, and returns TRUE if they are the same, and returns FALSE if they are different.

=EXACT([Column1],[Column2])

FIND

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

Syntax

Difference between FIND and SEARCH?

FIND(<find_text>, <within_text>[, [<start_num>][, <NotFoundValue>]])

Example

The following formula finds the position of the first letter of the product designation, BMX, in the string that contains the product description.

Simple Example:

Create a new measure and place on a card

=FIND("BMX","line of BMX racing goods")

Return: 9

Create a new measure and place on a card

FIND("bMX","line of BMX racing goods")

Return: Error

FIXED

Rounds a number to the specified number of decimals and returns the result as text. You can specify that the result be returned with or without commas.

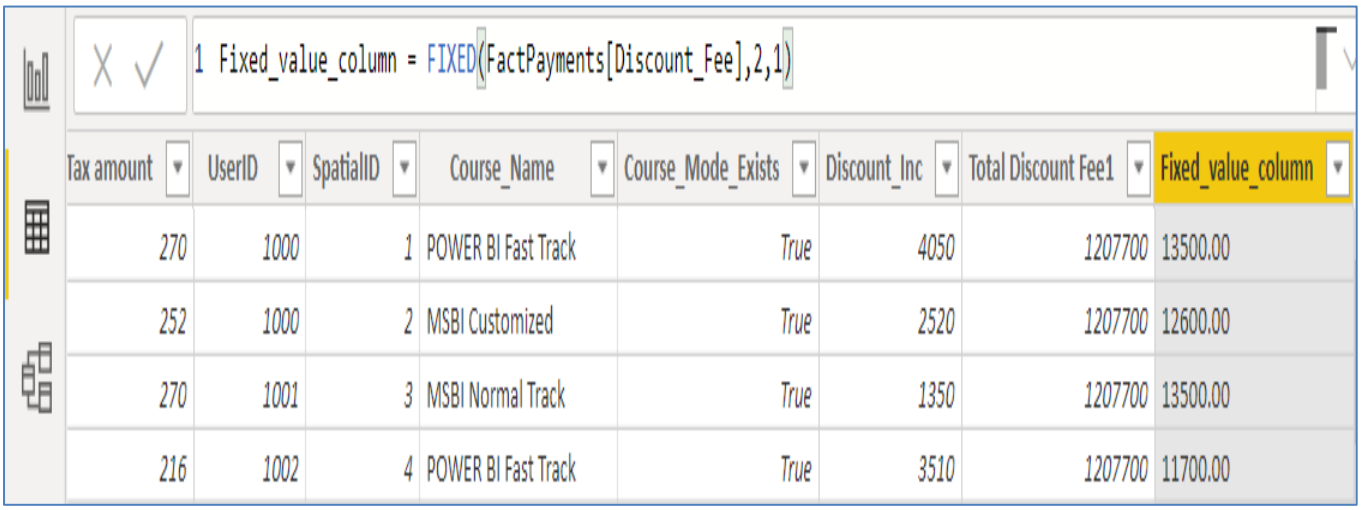
Syntax

FIXED(<number>, <decimals>, <no_commas>)

scenario: Create two columns in the Fact Payments table an d observe the result

Fixed_value_column=FIXED(FACTPAYMENTS[DISCOUNT_FEE],2,1)

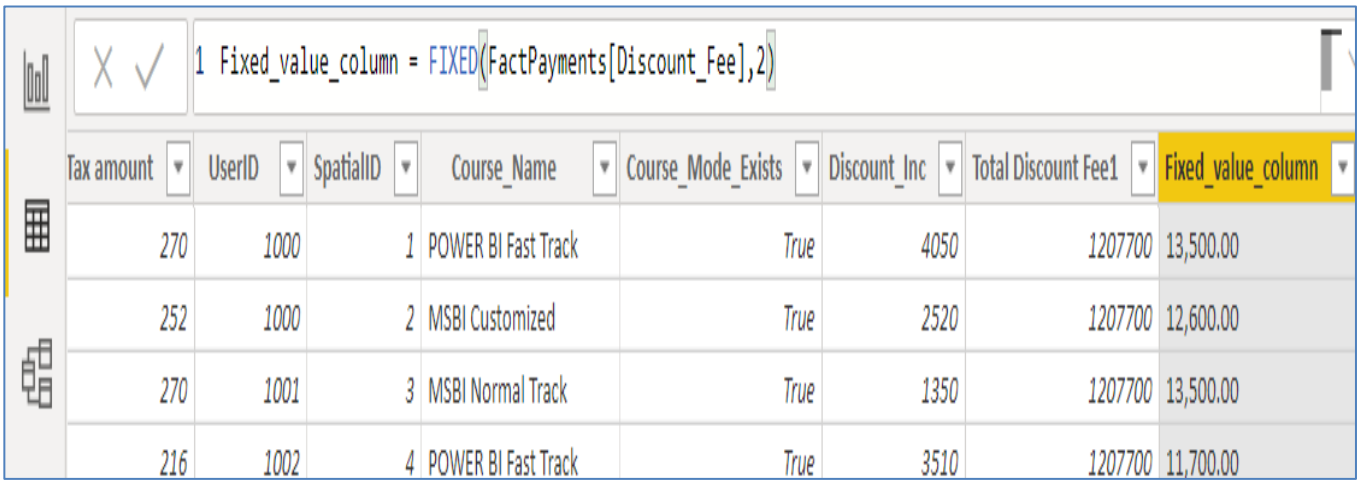
The below result without commas as we mentioned 1 as the third argument



```
1 Fixed_value_column = FIXED(FactPayments[Discount_Fee],2,1)
```

Tax amount	UserID	SpatialID	Course_Name	Course_Mode_Exists	Discount_Inc	Total Discount Fee1	Fixed_value_column
270	1000	1	POWER BI Fast Track	True	4050	1207700	13500.00
252	1000	2	MSBI Customized	True	2520	1207700	12600.00
270	1001	3	MSBI Normal Track	True	1350	1207700	13500.00
216	1002	4	POWER BI Fast Track	True	3510	1207700	11700.00

Fixed_value_column=FIXED(FACTPAYMENTS[DISCOUNT_FEE],2)



```
1 Fixed_value_column = FIXED(FactPayments[Discount_Fee],2)
```

Tax amount	UserID	SpatialID	Course_Name	Course_Mode_Exists	Discount_Inc	Total Discount Fee1	Fixed_value_column
270	1000	1	POWER BI Fast Track	True	4050	1207700	13,500.00
252	1000	2	MSBI Customized	True	2520	1207700	12,600.00
270	1001	3	MSBI Normal Track	True	1350	1207700	13,500.00
216	1002	4	POWER BI Fast Track	True	3510	1207700	11,700.00

Example:

The following example gets the numeric value for the current row in column, **PctCost, and returns it as text with 4 decimal places and no commas.**

=FIXED([PctCost],3,1)

Numbers can never have more than 15 significant digits, but decimals can be as large as 127.

FORMAT [FULL PRACTICE REQUIRED ****]

Converts a value to text according to the specified format.

Syntax

FORMAT(<VALUE>,<FORMAT STRING>)

Pre-Defined Numeric Formats for the FORMAT

The following table identifies the predefined numeric format names. These may be used by name as the style argument for the Format function.

Example

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

Practice the below by creating a measure and placing on card

FORMAT(12345.67, "General Number")

FORMAT(12345.67, "Currency")

FORMAT(12345.67, "Fixed")

FORMAT(12345.67, "Standard")

FORMAT(12345.67, "Percent")

FORMAT(12345.67, "Scientific")

Pre-defined date and time formats for the FORMAT function

The following table identifies the predefined date and time format names. If you use strings other than these predefined strings, they will be interpreted as a custom date and time format.

FORMAT SPECIFICATION	DESCRIPTION
"General Date"	Displays a date and/or time. For example, 3/12/2008 11:07:31 AM. Date display is determined by your application's current culture value.
"Long Date" or "Medium Date"	Displays a date according to your current culture's long date format. For example, Wednesday, March 12, 2008.
"Short Date"	Displays a date using your current culture's short date format. For example, 3/12/2008.
"Long Time" or	Displays a time using your current culture's long time format; typically includes hours, minutes, seconds. For example, 11:07:31 AM.
"Medium Time"	Displays a time in 12 hour format. For example, 11:07 AM.
"Short Time"	Displays a time in 24 hour format. For example, 11:07.

Custom date and time formats for the FORMAT

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

FORMAT SPECIFICATION	DESCRIPTION
(:)	Time separator. In some locales, other characters may be used to represent the time separator. The time separator separates hours, minutes, and seconds when time values are formatted. The actual character that is used as the time separator in formatted output is determined by your application's current culture value.
(/)	Date separator. In some locales, other characters may be used to represent the date separator. The date separator separates the day, month, and year when date values are formatted. The actual character that is used as the date separator in

	formatted output is determined by your application's current culture.
(%)	Used to indicate that the following character should be read as a single-letter format without regard to any trailing letters. Also used to indicate that a single-letter format is read as a user-defined format. See what follows for additional details.
d	Displays the day as a number without a leading zero (for example, 1). Use %d if this is the only character in your user-defined numeric format.
dd	Displays the day as a number with a leading zero (for example, 01).
ddd	Displays the day as an abbreviation (for example, Sun).
dddd	Displays the day as a full name (for example, Sunday).
M	Displays the month as a number without a leading zero (for example, January is represented as 1). Use %M if this is the only character in your user-defined numeric format.
MM	Displays the month as a number with a leading zero (for example, 01/12/01).
MMM	Displays the month as an abbreviation (for example, Jan).
MMMM	Displays the month as a full month name (for example, January).

gg	Displays the period/era string (for example, A.D.).

FORMAT SPECIFICATION	DESCRIPTION
h	Displays the hour as a number without leading zeros using the 12-hour clock (for example, 1:15:15 PM). Use %h if this is the only character in your user-defined numeric format.
hh	Displays the hour as a number with leading zeros using the 12-hour clock (for example, 01:15:15 PM).
H	Displays the hour as a number without leading zeros using the 24-hour clock (for example, 1:15:15). Use %H if this is the only character in your user-defined numeric format.
HH	Displays the hour as a number with leading zeros using the 24-hour clock (for example, 01:15:15).
m	Displays the minute as a number without leading zeros (for example, 12:1:15). Use %m if this is the only character in your user-defined numeric format.
mm	Displays the minute as a number with leading zeros (for example, 12:01:15).
s	Displays the second as a number without leading zeros (for example, 12:15:5). Use %s if this is the only character in your user-defined numeric format.
ss	Displays the second as a number with leading zeros (for example, 12:15:05).
AM/PM	Use the 12-hour clock and display an uppercase AM with any hour before noon; display an uppercase PM with any hour

	between noon and 11:59 P.M.
am/pm	Use the 12-hour clock and display a lowercase AM with any hour before noon; display a lowercase PM with any hour between noon and 11:59 P.M.
A/P	Use the 12-hour clock and display an uppercase A with any hour before noon; display an uppercase P with any hour between noon and 11:59 P.M.
a/p	Use the 12-hour clock and display a lowercase A with any hour before noon; display a lowercase P with any hour between noon and 11:59 P.M.
AMPM	Use the 12-hour clock and display the AM string literal as defined by your system with any hour before noon; display the PM string literal as defined by your system with any hour between noon and 11:59 P.M. AMPM can be either uppercase or lowercase, but the case of the string displayed matches the string as defined by your system settings. The default format is AM/PM.
y	Displays the year number (0-9) without leading zeros. Use %y if this is the only character in your user-defined numeric format.

FORMAT SPECIFICATION	DESCRIPTION
yy	Displays the year in two-digit numeric format with a leading zero, if applicable.
yyy	Displays the year in four-digit numeric format.

yyyy	Displays the year in four-digit numeric format.
z	Displays the timezone offset without a leading zero (for example, -8). Use %z if this is the only character in your user-defined numeric format.
zz	Displays the timezone offset with a leading zero (for example, -08)
zzz	Displays the full timezone offset (for example, -08:00)

Scenario: create a date table [PBI Desktop] like below with various formatting options

DimDate=

```
ADDCOLUMNS (
CALENDAR (DATE(2000,1,1), DATE(2025,12,31)),
"DateAsInteger", FORMAT ( [Date], "YYYYMMDD" ),
"Year", YEAR ( [Date] ),
"Monthnumber", FORMAT ( [Date], "MM" ),
"YearMonthnumber", FORMAT ( [Date], "YYYY/MM" ),
"YearMonthShort", FORMAT ( [Date], "YYYY/mmm" ),
"MonthNameShort", FORMAT ( [Date], "mmm" ),
"MonthNameLong", FORMAT ( [Date], "mmmm" ),
"DayOfWeekNumber", WEEKDAY ( [Date] ),
"DayOfWeek", FORMAT ( [Date], "dddd" ),
"DayOfWeekShort", FORMAT ( [Date], "ddd" ),
"Quarter", "Q" & FORMAT ( [Date], "Q" ),
"YearQuarter", FORMAT ( [Date], "YYYY" ) & "/Q" & FORMAT ( [Date], "Q" )
```

LEFT

Returns the specified number of characters from the start of a text string.

Syntax

LEFT(<text>, <num_chars>)

Simple Examples:

Create a measure and place on a card

=LEFT("vinay kumar",3)

Result:vin

Example:

The following example returns the first five characters of the company name in the column [ResellerName] and the first five letters of the geographical code in the column [GeographyKey] and concatenates them, to create an identifier.

=CONCATENATE(LEFT('Reseller'[ResellerName],LEFT(GeographyKey,3))

If the **num_chars** argument is a number that is larger than the number of characters available, the function returns the maximum characters available and does not raise an error. For example, the column [GeographyKey] contains numbers such as 1, 12 and 311; therefore the result also has variable length.

LEN

Returns the number of characters in a text string.

Syntax

LEN(<text>)

Simple Example:

Create a measure and place on a card

=LEN("vinay kumar")

Result:11

Example

The following formula sums the lengths of addresses in the columns, [AddressLine1] and [AddressLine2].

=LEN([AddressLine1])+LEN([AddressLin2])

LOWER

Converts all letters in a text string to lowercase.

Syntax

LOWER(<TEXT>)

Return value

Text in lowercase.

TERM	DEFINITION
text	The text you want to convert to lowercase, or a reference to a column that contains text.

Remarks

Characters that are not letters are not changed. For example, the formula =LOWER("123ABC") returns **123abc**.

Simple Example:

Create a measure and place on a card

=LOWER("VINAYKUMAR")

Result: vinaykumar

Example

The following formula gets each row in the column, [ProductCode], and converts the value to all lowercase.
Numbers in the column are not affected.

=LOWER('New Products'[ProductCode])

MID

Returns a string of characters from the middle of a text string, **given a starting position and length.**

Syntax

MID(<text>, <start_num>, <num_chars>)

Simple Example:

Create a measure and place on a card

=MID("vinay kumar",2,3)

Note: It is like SUBSTRING in other languages

Result: ina

Example

The following examples return the same results, the first 5 letters of the column, [ResellerName]. The first example uses the fully qualified name of the column and specifies the starting point; the second example omits the table name and the parameter, **num_chars**.

```
=MID('Reseller'[ResellerName],5,1))
=MID([ResellerName,5])
```

The results are the same if you use the following formula:

```
=LEFT([ResellerName],5)
```

REPLACE

REPLACE replaces part of a text string**, based on the number of characters** you specify, with a different text string.

Syntax

REPLACE(<old_text>, <start_num>, <num_chars>, <new_text>)

Simple Example:

Create a measure and place on a card

=REPLACE("vinay kumar",7,5,"house")

Here 7th position to 5 chars replacing

Result: vinay house

Example

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

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

=REPLACE('DIMCOURSE'[COURSE DESC], 3,9,"VINAYTECH")

REPT [REPITITION]

Repeats text a given number of times. Use REPT to fill a cell with a number of instances of a text string.

Syntax

REPT(<text>, <num_times>)

Simple Example:

Create a measure and place on a card

=REPT("85",3)

Result:

858585

Example: Repeating Column Values

Remarks

If **number_times** is 0 (zero), REPT returns a blank.

If **number_times** is not an integer, it is truncated.

The result of the REPT function cannot be longer than 32,767 characters, or REPT returns an error.

Example: Repeating Literal Strings

Description

The following example returns the string, 85, repeated three times.

Description

The following example returns the string in the column, [MyText], repeated for the number of times in the column, [MyNumber]. Because the formula extends for the entire column, the resulting string depends on the text and number value in each row.

Code

```
=REPT([MyText],[MyNumber])
```

Comments

MYTEX T	MYNUMBER	CALCULATEDCOLUMN1
Text	2	TextText
Number	0	
85	3	858585

RIGHT

RIGHT returns the last character or characters in a text string, based on the number of characters you specify.

Syntax

RIGHT(<text>, <num_chars>)

Example: Returning a Fixed Number of Characters

Simple Example:

Create a measure and place on a card

=RIGHT("vinay kumar",3)

Result: mar

Description

The following formula returns the last two digits of the product code in the New Products table.

Code

=RIGHT('New Products'[ProductCode],2)

Example: Using a Column Reference to Specify Character Count

Description

The following formula returns a variable number of digits from the product code in the New Products table, depending on the number in the column, MyCount. If there is no value in the column, MyCount, or the value is a blank, RIGHT also returns a blank.

Code

=RIGHT('New Products'[ProductCode],[MyCount])

SEARCH

Differences between Find and Search?

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

Syntax

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

Example: Search within a String

Simple Examples:

Create a measure and place on a card

=SEARCH(“nay”,”vinay kumar”)

Result: 3

Create a measure and place on a card

SEARCH("NAY","vinay kumar")

Result: Success

3

Description

The following formula finds the position of the letter "n" in the word "printer".

Code

=SEARCH("n","printer")

Comments

The formula returns 4 because "n" is the fourth character in the word "printer."

Example: Search within a Column

Description

You can use a column reference as an argument to SEARCH. The following formula finds the position of the character "-" (hyphen) in the column, [PostalCode].

Code

```
=SEARCH("-",[PostalCode])
```

Comments

The return result is a column of numbers, indicating the index position of the hyphen.

Example: Error-Handling with SEARCH

Description

The formula in the preceding example will fail if the search string is not found in every row of the source column. Therefore, the next example demonstrates how to use IFERROR with the SEARCH function, to ensure that a valid result is returned for every row.

The following formula finds the position of the character "-" within the column, and returns -1 if the string is not found.

Code

```
= IFERROR(SEARCH("-",[PostalCode]),-1)
```

Comments

Note that the data type of the value that you use as an error output must match the data type of the non-error output type. In this case, you provide a numeric value to be output in case of an error because SEARCH returns an integer value.

However, you could also return a blank (empty string) by using BLANK() as the second argument to IFERROR.

SUBSTITUTE

Difference between REPLACE and SUBSTITUTE?

Replaces existing text with new text in a text string.

Syntax

SUBSTITUTE(<text>, <old_text>, <new_text>, <instance_num>)

Example: Substitution within a String

Simple Example:

Create a measure and place on a card

= substitute("vinay kuinmarin","in","UN")

Result: vUNay kuUNmarUN

Description

The following formula creates a copy of the column [Product Code] that substitutes the new product code **NW** for the old product code **PA** wherever it occurs in the column.

Code

```
=SUBSTITUTE([Product Code], "NW", "PA")
```

TRIM

Removes all spaces from text except for single spaces between words.

Syntax

TRIM(<text>)

Simple Example:

Create a measure and place on a card

=TRIM(" VINAY KUMAR ")

Result: VINAY KUMAR

Example

following formula creates a new string that does not have trailing white space.

=TRIM("A column with trailing spaces.")

When you create the formula, the formula is propagated through the row just as you typed it, so that you see the original string in each formula and the results are not apparent. However, when the formula is evaluated the string is trimmed.
You can verify that the formula produces the correct result by checking the length of the calculated column created by the previous formula, as follows:

=LEN([Calculated Column 1])

UNICHAR

Returns the Unicode character referenced by the numeric value.

Syntax

UNICHAR(number)

Simple Example:

Create a measure and place on a card

=UNICHAR(9733)

Example

The following example returns the character represented by the Unicode number 66 (uppercase

Create a measure and place on a card

=UNICHAR(65)

Result: A

The following example returns the character represented by the Unicode number 32 (space character).

=UNICHAR(32)

Result: !

The following example returns the character represented by the Unicode number 9733 (★ character).

UPPER

Converts a text string to all uppercase letters

Syntax

UPPER(<TEXT>)

Simple Example:

Create a measure and place on a card

UPPER("vinay kumar")

Result: VINAY KUMAR

Example

The following formula converts the string in the column, [ProductCode], to all uppercase. Non-alphabetic characters are not affected.

=UPPER(['New Products'[Product Code])

VALUE

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

Syntax

Differences between values and value?

VALUE(<text>)

Simple Example:

Create a newcolumn and see the data view

=VALUE (Factpayments[Discount_Fee])

Example

The following formula converts the typed string, "3", into the numeric value 3.

Create a measure and place on a card

=VALUE("320")

Result: 320

Assume Discount Fee comes from file, then system treat as text, to consider as numeric for operations.

LOGICAL FUNCTIONS OVERVIEW

LOGICAL FUNCTIONS [They work on conditions, TRUE OR FALSE values, and logical evaluation]

AND	Multiple conditions to be satisfied, ALL TRUE, THEN RESULT TRUE.	Measure / New Column
OR	Single condition to be satisfied, IF ANY TRUE, RESULT TRUE.	Measure / New Column
NOT	Negating a condition	Measure / New Column
IFERROR	Error validation	Measure / New Column
IF	Single condition validation	Measure / New Column
SWITCH	Multiple conditions validation	Measure / New Column
IN	Validate column against list of values specified	Measure / New Column
TRUE	Returns explicity True	Measure / New Column
FALSE	Returns explicitly False	Measure / New Column

Run the below query under DAX Studio and see the result

```
evaluate
(
addcolumns
(
DimCourse,

"and && validation",  if(DimCourse[CourseID]="MSBI-F" && DimCourse[Duration]=30,true(),false()),

"or || validation",

if(DimCourse[CourseID]="MSBI-F" || DimCourse[Duration]>30,"Proper Duration", "Fast Track"),

"Switch multi condition validation",

switch (DimCourse[Duration],20,"Twenty", 30,"Thirty",40,"Fourty", 50,"Fifty"),

"Switch multi condition validation 1",

switch (DimCourse[Duration],20,DimCourse[Duration]+10,

30,DimCourse[Duration]+8,40,DimCourse[Duration]+6, 50,DimCourse[Duration]+5),

"If error validation",  iferror(25/0, "wrong calculation"),

"If not an error validation",  iferror(25/5, "Not a wrong calculation"),

"Required Courses Fee",

CALCULATE(SUM(FactPayments[Discount_Fee]), DimCourse[CourseID] IN {"MSBI-F","MSBI-N","POWER
BI-F"})
)

)
```

Detailed Practice

AND:

Create a new measure and place on a card

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

OR:

Create a new measure and place on a card

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

TRUE:

Create a new measure and place on a card

```
=IF(SUM(Factpayments[Discount_Fee]) >200000, TRUE(), FALSE())
```

IF:

Create a new column in the factpayments and see the result

```
=IF([Taxamount]<200,"low",IF([Taxamount]<250,"medium","high"))
```


Note: This is Nested IF clause

```
Taxrate = IF([Tax amount]<200,"low",IF([Tax amount]<250,"medium","high"))
```

Actual_Fee	Discount_Fee	Tax amount	UserID	SpatialID	Join_Date	Course_End_Date	Taxrate
15000	,13,500.00	230	1002	4	12/31/2019 12:00:00 AM	12/31/2019 12:00:00 AM	medium
15000	,13,500.00	270	1000	1	03/18/2019 12:00:00 AM	03/18/2019 12:00:00 AM	high
14000	,12,600.00	252	1000	2	03/17/2019 12:00:00 AM	03/17/2019 12:00:00 AM	high
15000	,13,500.00	270	1001	3	03/16/2019 12:00:00 AM	03/16/2019 12:00:00 AM	high
13000	,11,700.00	216	1002	4	03/15/2019 12:00:00 AM	03/15/2019 12:00:00 AM	medium
15000	,13,500.00	216	1003	5	03/14/2019 12:00:00 AM	03/14/2019 12:00:00 AM	medium

IFERROR:

Create a new measure and place on a card

```
=IFERROR(25/0,99999)
```

Note:

No extra conditional expression required, if it is error, automatically handle.

This is equals to Previous Informational Function

IF(iserror(25/x), 99999, 25/x)

Iserror returns true/ false, so conditional handling required to implement the above IFERROR functionality.

IN:

Scenario: Find out the discount fee for the courses MSBI-F, MSBI-N, and POWER BI-F

Example

Create a new measure and place on a card

```
RequiredVal = CALCULATE(SUM(FactPayments[Discount_Fee]), DimCourse[CourseID]
IN {"MSBI-F","MSBI-N","POWER BI-F"})
Note: IN clause is different from ContainsRow because IN clause returns
values whereas containsrow return True / False (informational function)
```

DAX Studio / SSMS:

```
Evaluate
(
Row("Required Courses Fee",
CALCULATE(SUM(FactPayments[Discount_Fee]), DimCourse[CourseID] IN
{"MSBI-F","MSBI-N","POWER BI-F"})
)
```

SQL Query:

```
SELECT SUM(DISCOUNT_FEE) FROM FACTPAYMENTS
WHERE COURSEID IN ("MSBI-F","MSBI-N","POWER BI-F")
```

Note:

IN--Multiple values against single column

CONTAINS--Single value against single column

CONTAINSROW--Multiple values against multiple columns in a row

SWITCH

This is helpful to prevent nested if clause (if inside another if).

Scenario: Based on course mode provide hike to the courses

Goto FactPayments →New column→

```
Discount_Inc =
switch(FactPayments[ModeID],"Online",FactPayments[Discount_Fee]
*10/100,"Classroom",FactPayments[Discount_Fee] * 20/100,"Customized",
FactPayments[Discount_Fee]* 30/100)
```

```
Discount_Inc =
switch(FactPayments[ModeID],"Online",FactPayments[Discount_Fee] *10/100,"Classroom",FactPayments[Discount_Fee] * 20/100,
"Customized", FactPayments[Discount_Fee]* 30/100)
```

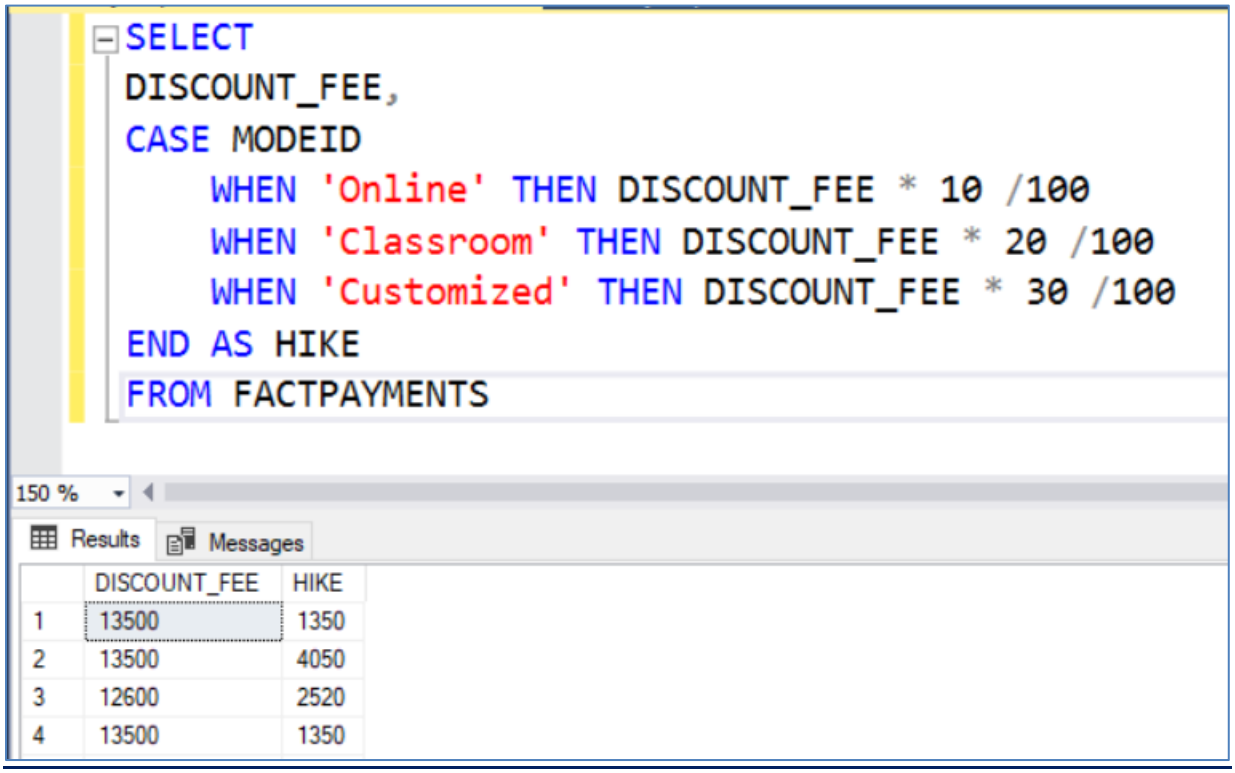
	StudentID	Actual_Fee	Discount_Fee	Tax amount	UserID	SpatialID	Course_Name	Course_Mode_Exists	Discount_Inc
0:00 AM	1098	15000	13500	270	1000	1	POWER BI Fast Track	True	4050
0:00 AM	1097	14000	12600	252	1000	2	MSBI Customized	True	2520
0:00 AM	1096	15000	13500	270	1001	3	MSBI Normal Track	True	1350
0:00 AM	1095	13000	11700	216	1002	4	POWER BI Fast Track	True	3510
0:00 AM	1094	15000	13500	216	1003	5	POWER BI Fast Track	True	2700
0:00 AM	1093	13000	11700	234	1000	6	MSBI Customized	True	1170

DAX Studio Query:

```
EVALUATE
(

ADDCOLUMNS(FactPayments,"New DF",switch(FactPayments[ModeID],"Online",FactPayments[Discount_Fee]

*10/100,"Classroom",FactPayments[Discount_Fee] * 20/100,"Customized", FactPayments[Discount_Fee]*

30/100))

)
```

SQL QUERY

```
SELECT

Discount_Fee,

CASE

when ModeID='Online'THEN Discount_Fee+ Discount_Fee * 10/100

when ModeID='Classroom' THEN Discount_Fee+ Discount_Fee * 20/100

when ModeID='Customized' THEN Discount_Fee+ Discount_Fee * 30/100

END as DiscountFee_Hike

FROM FactPayments
```

	DISCOUNT_FEE	HIKE
1	13500	1350
2	13500	4050
3	12600	2520
4	13500	1350

NOT

Create a new measure and place on a card

=not (True)

Result: False

MATHEMATICAL AND TRIG FUNCTIONS

Perform mathematical operations on data [numeric values supported]

MATH AND TRIG FUNCTIONS		
ABS	Absolute Value [Negative to Positive, round to the nearest integer]	Measure/New Col
COMBIN	Returns the number of combinations for a given number of items.	Measure/New Col
COMBINA	Returns the number of combinations (with repetitions) for a given number of items.	Measure/New Col
CURRENCY	Evaluates the argument and returns the result as currency data type.	Measure/New Col
DIVIDE	Performs division and returns alternate result or BLANK() on division by 0.	Measure/New Col
SUM	Sum of column values [column wise operation]	Measure/New Col
SUMX	Sum of column values based on expression [row wise operation]	Measure/New Col
ROUND	Rounding to the nearest integer	Measure/New Col
MROUND	Rounding to the neartest multiple of integer	Measure/New Col
ROUNDUP	Rounding to the upper integer	Measure/New Col
ROUNDDOWN	Rounding to the lower integer	Measure/New Col
RAND	Random value between 0 and 1	Measure/New Col
RANDBETWEEN	Random value between the user values specified	Measure/New Col
SQRT	Square root value	Measure/New Col
TRUNC	Truncation of values	Measure/New Col
PI	PI result of a value	Measure/New Col
LOG	Take base value to perform LOG	Measure/New Col
LOG10	Log10 algorithm value	Measure/New Col
LN	Log Natural Algorithm value	Measure/New Col
ISO.CEILING		Measure/New Col
CEILING	Rounds a number up, to the nearest integer or to the nearest multiple of significance.	Measure/New Col
FLOOR	Rounding to the lower value	Measure/New Col
ISO.FLOOR		Measure/New Col
GCD	Greatest Common Divisor	Measure/New Col
LCM	Least Common Multiple	Measure/New Col
EXP	Exponentiation of value	Measure/New Col
FACT	Factorial Value	Measure/New Col
Even	Shows nearest even value	Measure/New Col
ODD	Shows nearest ODD Value	Measure/New Col
PRODUCT	Product join with other table column. Product Operation (multiplication) of all values in a column.	Measure/New Col
PRODUCTX	Product join with other table column based on expression. Product of (multiplication) all values in a column based on expression.	Measure/New Col

Note: Where ever "X " appears, that is expression based operation.

Write and run the below queries under DAX Studio

```
EVALUATE
(

row(
"ABsolute value -99",abs(-99), --a) Negative to positive b) Rounding to the nearest integer

"Nearest even number 3", EVEN(3), --result: 4

"Nearest Odd number 2", ODD(2),--result: 3

"Truncating decimal value 2.82", trunc(2.82), --result : 2

"Rounding down 2.82,1", rounddown(2.82,1), --result: 2.8

"Rounding up 2.82,1", roundup(2.82,1),--result: 2.9

"General ceiling value 2.82,1", ceiling(2.82,1),--upper integer value Result: 3

"ceiling with significance 2.82,0.5", ceiling(2.82, .05), --result: 2.85

"Log value log(20)", LOG(20),

"Log 10 value log(10)",log10(20),

"log with base value", log(20,10),

"exponentiation value ,exp(3)", exp(3),

"Factorial value fact(10)", Fact(10),

" 2 items combination from 8 items", combin(8,2),

"Divide two valaues 5,2", DIVIDE(5,2),  --Result: 2.5

"Mod of two values 5,2", MOD(5,2), --Result : 1

"Quotient of values 5,2", QUOTIENT(5,2), --Result : 2

"Square root of value 16", SQRT(16), --result: 4

"Pi value", PI() --result: 3.14

)
)
```

```
--Scenario 1: Adding two columns to the DimCourse Table with Rand and Randbetween functions

EVALUATE
(

ADDCOLUMNS(

DimCourse,

"Random value", rand(),--Between 0 and 1 any value

"Random between", RANDBETWEEN(200,300) --between 200 and 300 any value

)

)

evaluate
(

row(
"product of all duration values", product(DimCourse[Duration])
)

)
--Scenario 2 : Multiplying all values of EHIKE using Product function

EVALUATE
(

row(
"Product of duration", product(TEST[EHIKE])
)

)

--
```

```
EVALUATE
(
Row
(
"Least common multiple", LCM(2,4),

"Greatest Common Divisor", GCD(2,4),

"EVen", even(3),

"Odd",Odd(2),

"Sum of DF", Sum(FactPayments[Discount_Fee]),

"Division", Divide(25,5),

"Division 1", IFERROR(Divide(25,0),99),

"Currency presentation", CURRENCY(1234.56)
)
)
```

Detailed Functions Practice

ABS:

Example1:

Create a new measure and place on a card

=ABS(-99)

Result: 99 [positive]

Example1:

Create a new measure and place on a card

=ABS(-99.56)

Result: 100 [Positive and rounding]

CEILING:

Syntax

CEILING(<number>, <significance>)

Example1

The following formula returns 4.45. This might be useful if you want to avoid using smaller units in your pricing. If an existing product is priced at $4.42, you can use CEILING to round prices up to the nearest unit of five cents.

Create a new measure and place on a card

=CEILING(4.42,0.05)

Result: 4.45

Note:

On Card if you are not able to see properly, then add new column and place the above expression

Example2

=CEILING([TaxAmount],0.05)

COMBIN

Returns the number of combinations for a given number of items. Use COMBIN to determine the total possible number of groups for a given number of items.

Syntax

COMBIN(number, number_chosen)

Example

Create a new measure and place on a card

FORMULA	DESCRIPTION	RESULT
=COMBIN(8,2)	Possible two-person teams that can be formed from 8 candidates.	28

COMBINA

Returns the number of combinations (with repetitions) for a given number of items.

Syntax

COMBINA(number, number_chosen)

Example

Create a new measure for each formula and place on a card

FORMULA	DESCRIPTION	RESULT
=COMBINA(4,3)	Returns the number of combinations (with repetitions) for 4 and 3.	20
=COMBINA(10,3)	Returns the number of combinations (with repetitions) for 10 and 3.	220

CURRENCY

Evaluates the argument and returns the result as currency data type.

Syntax

CURRENCY(<value>)

Example 1:

Convert number 1234.56 to currency data type.

Create a measure like below and place on a card

=CURRENCY(1234.56)

Returns the value $1234.5600.

Example 2:

Display currency format of values for Discount Fee in FactPayments

Add a new column to the FactPayments table with the below expression and preview the data

=CURRENCY(FactPayments[Discount_Fee])

Note:

Incase it is not shoing in currency format, go to Modeling Menu, change Format of the column→Currency

DIVIDE

Performs division and returns alternate result or BLANK() on division by 0.

Syntax

DIVIDE(<numerator>, <denominator> [,<alternateresult>])

Example

Create a measure like below and place on a card

The following example returns 2.5.

=DIVIDE(5,2)

Example

Create a measure like below and place on a card

The following example returns BLANK.

=DIVIDE(5,0)

Example

Create a measure like below and place on a card

=DIVIDE(25,0,1)

Output: 1

EVEN

Returns number rounded up to the nearest even integer. You can use this function for processing items that come in twos. For example, a packing crate accepts rows of one or two items. The crate is full when the number of items, rounded up to the nearest two, matches the crate's capacity.

Syntax

EVEN(number)

Example

Create a measure for each formula and place on a card

FORMULA	DESCRIPTION	RESULT
=EVEN(1.5)	Rounds 1.5 to the nearest even integer	2
=EVEN(3)	Rounds 3 to the nearest even integer	4
=EVEN(2)	Rounds 2 to the nearest even integer	2
=EVEN(-1)	Rounds -1 to the nearest even integer	-2

EXP

Returns e raised to the power of a given number. The constant e equals 2.71828182845904, the base of the natural logarithm.

Syntax

EXP(<number>)

Simple Example:

Create a measure like below and place on a card

=EXP(3)

Example

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

=EXP([Power])

FACT

Returns the factorial of a number, equal to the series 1*2*3*...* , ending in the given number.

Syntax

FACT(<number>)

Simple Example:

Create a measure like below and place on a card

=FACT(60)

Example

The following formula returns the factorial for the series of integers in the column, [Values] .

=FACT([Values])

FLOOR

Rounds a number down, toward zero, to the nearest multiple of significance.

Syntax

FLOOR(<number>, <significance>)

Simple Scenario:

Create a measure like below and place on a card

=FLOOR(23442.234,.5)

Example

The following formula takes the values in the [Total Product Cost] column from the table, InternetSales.and rounds down to the nearest multiple of .1.

=FLOOR(InternetSales[Total Product Cost],.5)

GCD

Returns the greatest common divisor of two or more integers. The greatest common divisor is the largest integer that divides both number1 and number2 without a remainder.

Syntax

GCD(number1, [number2], ...)

Example

Create a measure for each formula and place on a card

FORMULA	DESCRIPTION	RESULT
=GCD(5, 2)	Greatest common divisor of 5 and 2.	1
=GCD(24, 36)	Greatest common divisor of 24 and 36.	12

INT

Rounds a number down to the nearest integer.

Syntax

INT(<number>)

Example

The following expression rounds the value to 1. If you use the ROUND function, the result would be 2.

Create a measure like below and place on a card

=INT(1.5)

Result: 2

ISO.CEILING

Rounds a number up, to the nearest integer or to the nearest multiple of significance.

Syntax

ISO.CEILING(<number>[, <significance>])

Example: Positive Numbers

Description

The following formula returns 4.45. This might be useful if you want to avoid using smaller units in your pricing. If an existing product is priced at $4.42, you can use ISO.CEILING to round prices up to the nearest unit of five cents.

Create a measure like below and place on a card

=ISO.CEILING(4.42,0.05)

Example: Negative Numbers

The following formula returns the ISO ceiling value of -4.40.

Create a measure like below and place on a card

=ISO.CEILING(-4.42,0.05)

LCM

Returns the least common multiple of integers. The least common multiple is the smallest positive integer that is a multiple of all integer arguments number1, number2, and so on. Use LCM to add fractions with different denominators.

Syntax

LCM(number1, [number2], ...)

Example

Create a measure for each formula and place on a card

FORMULA	DESCRIPTION	RESULT
=LCM(5, 2)	Least common multiple of 5 and 2.	10
=LCM(24, 36)	Least common multiple of 24 and 36.	72

LN

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

Syntax

LN(<number>)

Example

The following example returns the natural logarithm of the number in the column, [Values] .

Create a measure like below and place on a card

=LN([Values])

=LN(20)

LOG

Returns the logarithm of a number to the base you specify.

Syntax

LOG(<number>,<base>)

Example

The following formulas return the same result, 2.

Create a measure like below and place on a card

=LOG(100,10)

=LOG(100)

LOG10

Returns the base-10 logarithm of a number

Syntax

LOG10(<NUMBER>)

Example

The following formulas return the same result, 2:

Create a measure like below and place on a card

=LOG(100,10)

=LOG(100)

=LOG10(100)

MOD

Returns the remainder after a number is divided by a divisor. The result always has the same sign as the divisor.

Syntax

MOD(<number>, <divisor>)

Example

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

=MOD(3,2)

Example

The following formula returns -1, the remainder of 3 divided by 2. Note that the sign is always the same as the sign of the divisor.

Create a measure like below and place on a card

=MOD(-3,-2)

MROUND

eturns a number rounded to the desired multiple

Syntax

MROUND(<NUMBER>,<MULTIPLE>)

Return value

A decimal number.

Description

The following expression rounds 1.3 to the nearest multiple of .2. The expected result is 1.4.

Create a measure like below and place on a card

=MROUND(1.3,0.2)

Result: 1.4

Example: Negative Numbers

Description

The following expression rounds -10 to the nearest multiple of -3. The expected result is -9.

Create a measure like below and place on a card

=MROUND(-10,-3)

Create a measure like below and place on a card

Example: Error

Description

The following expression returns an error, because the numbers have different signs.

Create a measure like below and place on a card

=MROUND(5,-2)

ODD

Returns number rounded uo to the nearest odd integer.

Syntax

ODD(NUMBER)

Example

Create measures for each formula and place on cards

FORMULA	DESCRIPTION	RESULT
=ODD(1.5)	Rounds 1.5 up to the nearest odd integer	**3**
=ODD(3)	Rounds 3 up to the nearest odd integer	**3**
=ODD(-1)	Rounds -1 up to the nearest odd integer	**-3**

PI

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

Syntax

PI()

Example

The following formula calculates the area of a circle given the radius in the column, [Radius] .

Create a measure like below and place on a card

=PI()*([Radius]*2)

POWER

Returns the result of number raised to power.

Syntax

POWER(<NUMBER,<POWER>)

Return value

A decimal number.

Example

The following example returns 25.

Create a measure like below and place on a card

=POWER(5,2)

Result: 25

PRODUCT

Returns the product of the numbers in a column.

To return the product of an expression evaluated for each row in a table, use PRODUCTX function (DAX).

Syntax

PRODUCT(<column>)

Example

The following computes the product of the AdjustedRates column in an Annuity table:

=PRODUCT(Annuity[AdjustedRates])

Simple Example :

If we have a table of four rows, then the product of duration is

=Product(Samp([Duration]))

= 5* 4* 3 * 2=120

EID	Duration
1	5
2	4
3	3
4	2

Example: For each course display the total discount fee generated

1.Create a DiscountFee Total measure [Total Discount Fee = sum(FactPayments[Discount_Fee])]

2.Use in the DimCourse table, add column→ write the below expression, so that for each course it will show you the total discountfee value

Course DiscountFee = PRODUCT(FactPayments[Discount total])

Note: Other way using Relatedtable

Total business value = sumx(**RELATEDTABLE**(FactPayments), FactPayments[Discount_Fee])

PRODUCTX

Returns the product of an expression evaluated for each row in a table.

To return the product of the numbers in a column, use PRODUCT function (DAX).

Syntax

PRODUCTX(<table>, <expression>)

Example

The following computes the future value of an investment:

= [PresentValue] * PRODUCTX(AnnuityPeriods, 1+[FixedInterestRate])

QUOTIENT

Performs division and returns only the integer portion of the division result. Use this function when you want to discard the remainder of division.

Syntax

QUOTIENT(<numerator>, <denominator>)

Example

The following formulas return the same result, 2.

Create a measure like below and place on a card

=QUOTIENT(5,2)

Create a measure like below and place on a card

=QUOTIENT(10/2,2)

RADIANS

Converts degrees to radians.

Syntax

RADIANS(angle)

Example

Create a measure like below and place on a card

FORMULA	DESCRIPTION	RESULT
=RADIANS(270)	270 degrees as radians (4.712389 or 3π/2 radians)	4.712389

RAND

Returns a random number greater than or equal to 0 and less than 1, evenly distributed.

The number that is returned changes each time the cell containing this function is recalculated.

Syntax

RAND()

Example

To generate a random real number between two other numbers, you can use a formula like the following:

= RAND()*(int1-int2)+int1

Scenario:

Create a new column on any query with the below logic

=RAND()

RANDBETWEEN

Returns a random number in the range between two numbers you specify.

Syntax

RANDBETWEEN(<bottom>,<top>)

Example

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

=RANDBETWEEN(1,10)

Scenario:

Create a new column on any query with the below logic

=RANDBETWEEN(200,300)

ROUND

Rounds a number to the specified number of digits.

Syntax

ROUND(<NUMBER>,<NUM_DIGITS>)

Related functions

To always round up (away from zero), use the ROUNDUP function.

To always round down (toward zero), use the ROUNDDOWN function.

To round a number to a specific multiple (for example, to round to the nearest multiple of 0.5), use the MROUND function.

You can use the functions TRUNC and INT to obtain the integer portion of the number.

The following formula rounds 2.15 up, to one decimal place. The expected result is 2.2.

=ROUND(2.15,1)

Example

The following formula rounds 21.5 to one decimal place to the left of the decimal point. The expected result is 20.

Create a measure like below and place on a card

=ROUND(21.5,-1)

ROUNDDOWN

Rounds a number down, toward zero

Syntax

ROUNDDOWN(<number>, <num_digits>)

Related functions

ROUNDDOWN behaves like ROUND, except that it always rounds a number down. The INT function also rounds down, but with INT the result is always an integer, whereas with ROUNDDOWN you can control the precision of the result.

Example

The following example rounds 3.14159 down to three decimal places. The expected result is 3.141.

=ROUNDDOWN(3.14159,3)

Example

The following example rounds the value of 31415.92654 down to 2 decimal places to the left of the decimal. The expected result is 31400.

Create a measure like below and place on a card

=ROUNDDOWN(31415.92654, -2)

ROUNDUP

Rounds a number up, away from 0 (zero).

Syntax

ROUNDUP(<number>, <num_digits>)

Example

The following formula rounds Pi to four decimal places. The expected result is 3.1416.

Create a measure like below and place on a card

=ROUNDUP(PI(),4)

Example: Decimals as Second Argument

Description

The following formula rounds 1.3 to the nearest multiple of 0.2. The expected result is 2.

Create a measure like below and place on a card

=ROUNDUP(1.3,0.2)

Example: Negative Number as Second Argument

Description

The following formula rounds the value in the column, **FreightCost**, with the expected results shown in the following table:

Create a measure like below and place on a card

=ROUNDUP([Values],-1)

Comments

When **num_digits** is less than zero, the number of places to the left of the decimal sign is increased by the value you specify.

FREIGHTCOST	EXPECTED RESULT
13.25	20
2.45	10
25.56	30
1.34	10
345.01	350

SIGN

Determines the sign of a number, the result of a calculation, or a value in a column. The function returns 1 if the number is positive, 0 (zero) if the number is zero, or -1 if the number is negative.

Syntax

```
SIGN(<number>)
```

Parameters

TERM	DEFINITION
number	Any real number, a column that contains numbers, or an expression that evaluates to a number.

Return value

A whole number. The possible Return values are 1, 0, and -1.

Create a measure like below and place on a card

RETURN VALUE	DESCRIPTION
1	The number is positive
0	The number is zero
-1	The number is negative

Example

The following formula returns the sign of the result of the expression that calculates sale price minus cost.

```
=SIGN( ([Sale Price] - [Cost]) )
```

SQRT

Return the square root of a number

Syntax

SQRT(<number>)

Example

The following formula returns 5.

Create a measure like below and place on a card

=SQRT(25)

SUM

Adds all the numbers ina column.

Syntax

SUM(<COLUMN>)

Create a measure like below and place on a card

Total Discount Fee=sum(

Example

The following example adds all the numbers that are contained in the column, Amt, from the table, Sales.

```
=SUM(Sales[Amt])
```

SUMX

Returns the sum of an expression evaluated for each row in a table.

Syntax

SUMX(<table>, <expression>)

Example: Display Online total discount fee

Create a measure like below and place on a card

Online discount fee =

SUMX(FILTER(FactPayments,FactPayments[ModeID]="Online"),FactPayments[Total Discount Fee])

Example

The following example first filters the table, InternetSales, on the expression, ShippingTerritoryID = 5 , and then returns the sum of all values in the column, Freight. In other words, the expression returns the sum of freight charges for only the specified sales area.

=SUMX(FILTER(InternetSales, InternetSales[SalesTerritoryID]=5),[Freight])

If you do not need to filter the column, use the SUM function. The SUM function is similar to the Excel function of the same name, except that it takes a column as a reference.

TRUNC

Truncates a number to an integer by removing the decimal, or fractional, part of the number.

Syntax

TRUNC(<number>,<num_digits>)

Example

The following formula returns 3, the integer part of pi.

Create a measure like below and place on a card

=TRUNC(PI())

Example

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

Create a measure like below and place on a card

=TRUNC(-8.9)

TIME INTELLIGENCE FUNCTIONS OVERVIEW

TIME INTELLIGENCE FUNCTIONS PRACTICE

These functions play with intelligence of Date and Time operations.They always work on context and suitable for better decision making.

TIME INTELLIGENCE FUNCTIONS	
DATESBETWEEN	Return dates between the **starting** and end dates [2 dates required]
DATESINPERIOD	Return dates between the **starting date and the interval** [1 date +1 interval required]
ClosingBalanceofmonth	In the current content [year] closing balance of the last month or a specific month closing balance
ClosingBalanceOfQuarter	In the current content [year] closing balance of the last quarter or specific quarter closing balance
ClosingBalanceOfYear	In the current context closing balance
OpeningBalanceOfMonth	In the current content opening balance of the first month
OpeningBalanceOfQuarter	In the current context opening balance of the first month
OpeningBalanceOfyear	In the current context opening balance
DATESYTD	Year to date date values , Dates are returned
DATESQTD	Quarter to date date values,Dates are returned
DATESMTD	Month to date date values, Dates are returned
TOTALQTD	Total value for the quarter to date, Total value returned
TOTALMTD	Total value for the month to date, Total value returned
TOTALYTD	Total value for the year to date, Total value returned
FIRSTDATE	First date in the current content
FIRSTNONBLANKDATE	First non blank date in the current content
LASTDATE	Last date in the current content
LASTNONBLANKDATE	Last non blank date in the current content
NEXTDAY	Next day in the current context
NEXTMONTH	Next month in the current context
NEXTQUARTER	Next quarter in the current context
NEXTYEAR	Next year in the current context
PARALLEL PERIOD	Previous years, quarters, months displayed based on the date and interval. Close to previous functions.
SAMEPERIODLASTYEAR	Last year same period in the current context
PREVIOUSDAY	Previous day in the current context
PREVIOUSMONTH	Previous month in the current context [Complete month values]
PREVIOUSQUARTER	Previous quarter in the current context [Complete Quarter values]
PREVIOUSYEAR	Previous year in the current context [Complete year values]
STARTOFMONTH	Month starting in the current context
STARTOFQUARTER	Quarter starting in the current context
STARTOFYEAR	Start year of the business

Context not applied functions	Equavalent Function
Next Year/ Quarter / Month	ParallelPeriod (<Dates>,1, Year/ Quarter /Month)
Previous Year / Quarter / Month	ParallelPeriod(<Dates>,-1,Year/ Quarter / Month)
Note: Negative value for previous and positive value for Next operations	
DatesInPeriod	DatesBetween
DatesBetween	DatesInPeriod
DatesBetween(DimDate[Date],Date(2019,01,01),Date(2019,01,20)))	DatesInPeriod(DimDate[Date],Date(2019,01,01),20, Day)
Context applied	
SameperiodLastyear	
Calculate(sum(fact[Discount_Fee]),DATESYTD(DimDate[Date]))	TOTALYTD(sum(fact[Discount_Fee]), DimDate[Date])
Note: DATESYTD / DATESMTD/ DATESQTD These return Dates Supetset function, dates can be used for any calculation purpose [Ex: Average, MAX, MIN, etc.]	TOTLYTD/ TOTALMTD/ TOTALQTD These return summarized value based on date period. Subset function, suitable for returning summarized value.

Run the below queries in the DAX Studio

```
--Dates between: Returns the dates between specified dates
--I want the total discount fee between the dates given

EVALUATE
(

row (
"Discount fee between dates",
CALCULATE(sum(FactPayments[Discount_Fee]),DATESBETWEEN(DimDate[Date],date(2019,01,01),date(
2019,01,20))
)
)
)

--Dates Inperiod: Return dates in the give period
--I want the total discount fee for the given date interval

EVALUATE
(

row (
"Discount fee dates in period",
CALCULATE(sum(FactPayments[Discount_Fee]),DATESINPERIOD(DimDate[Date],date(2019,01,01),20,D
AY)
)
)
)

/* What is the difference between DateDiff and DatesBetween?

DateDiff gives the difference between dates in numrics
DatesBetween return set of dates.

*/
```

```
/* Finding 23 days data in the month of January 2018 */

EVALUATE
(

ROW(
"Dates between calculation",

calculate(sum(FactPayments[Discount_Fee]),
DATESBETWEEN(DimDate[Date],date(2018,01,01),date(2018,01,23))),

"Dates between calculation for 23 days",

calculate(sum(FactPayments[Discount_Fee]), DATESINPERIOD(DimDate[date],date(2018,01,01),23,
DAY)


)
)
)
```

Practice the below functions for better understanding of Time Intelligence

Create a parent measure like below

Total DF=sum(Factpayments[Discount_Fee])

Create child measure like below [Modeling->New Measure→] and place on card. Slice Year (in slicer) value and see the previous year value on card.

Last Year Complete Sales=calculate([Total Df],PreviousYear(DimDate[Date]))

Create child measure like below [Modeling->New Measure→] and place on card. Slice Quarter (in slicer) value and see previous quarter value on card.

Last Quarter Complete Sales=calculate([Total Df],PreviousQuarter(DimDate[Date]))

Create child measure like below [Modeling->New Measure→] and place on card. Slice Month (in slicer) value and see previous month value on card.

Last Month Complete Sales=calculate([Total Df],PreviousMonth(DimDate[Date]))

Create child measure like below [Modeling->New Measure→] and place on card. Slice Year (in slicer) value and see the Next year value on card.

Next Year Complete Sales=calculate([Total Df],NextYear(DimDate[Date]))

Create child measure like below [Modeling->New Measure→] and place on card. Slice Quarter (in slicer) value and see the Next Quarter value on card.

Next Quarter Complete Sales=calculate([Total Df],NextQuarter(DimDate[Date]))

Create child measure like below [Modeling->New Measure→] and place on card. Slice Month (in slicer) value and see the Next Month value on card.

Next Month Complete Sales=calculate([Total Df],NextMonth(DimDate[Date]))

Create child measure like below [Modeling->New Measure→] and place on card. Slice Year (in slicer) value and see the two previous years value on card.

Two Years Back Complete Sales

=calculate([Total Df],PreviousYear(PreviousYear(DimDate[Date])))

N Years Back Complete Sales

=calculate([Total Df],PreviousYear(PreviousYear....(DimDate[Date]))

Note: This is difficult to do, so recommended for ParallelPeriod

Create child measure like below [Modeling->New Measure→] and place on card. Slice Year (in slicer) value and see the two years back value on card.

Two Years Back Complete Sales

=calculate([Total Df],ParallelPeriod(DimDate[Date],-2,Year))

Note: N Years Back Data: In the above function replace -2 with -N

Create child measure like below [Modeling->New Measure→] and place on card. Slice Month (in slicer) value and see the two months back value on card.

Two Months Back Complete Sales

=calculate([Total Df],PreviousMonth(PreviousMonth(DimDate[Date])))

N Months Back Complete Sales

=calculate([Total Df],PreviousMonth(PreviousMonth...(DimDate[Date]))

Note: This is difficult to do, so recommended for ParallelPeriod

Create child measure like below [Modeling->New Measure→] and place on card. Slice Month (in slicer) value and see the two months back value on card.

Two Months Back Complete Sales

=calculate([Total Df],ParallelPeriod(DimDate[Date],-2,MONTH))

Note: N Months Back Data: In the above function replace -2 with -N

Create child measure like below [Modeling->New Measure→] and place on card. Slice Quarter (in slicer) value and see the two quarters back value on card.

Two Quarters Back Complete Sales

=calculate([Total Df],PreviousQuarter(PreviousQuarter(DimDate[Date])))

N Quarter Back Complete Sales

=calculate([Total Df],PreviousQuarter(PreviousQuarter...(DimDate[Date]))

Note: This is difficult to do, so recommended for ParallelPeriod

Create child measure like below [Modeling->New Measure→] and place on card. Slice Quarter (in slicer) value and see the two quarters back value on card.

Two Quarter Back Complete Sales

=calculate([Total Df],ParallelPeriod(DimDate[Date],-2,Quarter))

Note: N Quarter Back Data: In the above function replace -2 with -N

Create child measure like below [Modeling->New Measure→] and place on card. Slice Year (in slicer) value and see the two previous years value on card.

Two Years Back Complete Sales

=calculate([Total Df],PreviousYear(PreviousYear(DimDate[Date])))

N Years Back Complete Sales

=calculate([Total Df],PreviousYear(PreviousYear....(DimDate[Date]))

Note: This is difficult to do, so recommended for ParallelPeriod

Create child measure like below [Modeling->New Measure→] and place on card. Slice Year (in slicer) value and see the two years back value on card.

Two Years Back Complete Sales

=calculate([Total Df],ParallelPeriod(DimDate[Date],-2,Year))

Note: N Years Back Data: In the above function replace -2 with -N

Create child measure like below [Modeling->New Measure→] and place on card. Slice Month (in slicer) value and see the two months back value on card.

Two Months Back Complete Sales

=calculate([Total Df],PreviousMonth(PreviousMonth(DimDate[Date])))

N Months Back Complete Sales

=calculate([Total Df],PreviousMonth(PreviousMonth...(DimDate[Date]))

Note: This is difficult to do, so recommended for ParallelPeriod

Create child measure like below [Modeling->New Measure→] and place on card. Slice Month (in slicer) value and see the two months back value on card.

Two Months Back Complete Sales

=calculate([Total Df],ParallelPeriod(DimDate[Date],-2,MONTH))

Note: N Months Back Data: In the above function replace -2 with -N

Create child measure like below [Modeling->New Measure→] and place on card. Slice Quarter (in slicer) value and see the two quarters back value on card.

Two Quarters Back Complete Sales

=calculate([Total Df],PreviousQuarter(PreviousQuarter(DimDate[Date])))

N Quarter Back Complete Sales

=calculate([Total Df],PreviousQuarter(PreviousQuarter...(DimDate[Date]))

Note: This is difficult to do, so recommended for ParallelPeriod

Create child measure like below [Modeling->New Measure→] and place on card. Slice Quarter (in slicer) value and see the two quarters back value on card.

Two Quarter Back Complete Sales

=calculate([Total Df],ParallelPeriod(DimDate[Date],-2,Quarter))

Note: N Quarter Back Data: In the above function replace -2 with -N

Create child measure like below [Modeling->New Measure→] and place on card.

a)Slice Year (in slicer) value and see the same period last year value (complete year total) on card.

b)Slice Year (in slicer), Quarter (in slicer) value and see the same period last year and quarter value on card.

c)Slice Year (in slicer), Quarter (in slicer), Month (in slicer) value and see the same period last year, quarter, and month value on card.

d)Slice Year (in slicer), Quarter (in slicer), Month (in slicer), Day (in Slicer) value and see the same period last year, quarter, month, and day value on card.

Same Period Last Year Data

=Calculate([Total DF], sameperiodlastyear(Dimdate[Date]))

Create child measure like below [Modeling->New Measure→] and place on card.

a)Slice Year (in slicer) value and see the same period two last years value (complete year total) on card.

b)Slice Year (in slicer), Quarter (in slicer) value and see the same periods two last years and quarter value on card.

c)Slice Year (in slicer), Quarter (in slicer), Month (in slicer) value and see the same period two last years, quarter, and month value on card.

d)Slice Year (in slicer), Quarter (in slicer), Month (in slicer), Day (in Slicer) value and see the same period two last years, quarter, month, and day value on card.

Same Period Two Last Years Data

=Calculate([Total DF], sameperiodlastyear(sameperiodlastyear(Dimdate[Date]))

Same Period N Last Years Data=Calculate([Total DF],

sameperiodlastyear(sameperiodlastyear....(Dimdate[Date]))

PreviousYear=Complete Year **---> No Context Based {Filters not applied except year filter}**

SamePeriodLastYear--> **Based on the context{day/month/ qtr/ year last year same period data}**

2021-05-12--> sameperiodlastyear (day, month, and year specified)--> 2020-05-12

2021-05--> sameperiodlastyear (month and year specified)-->2020-05

2021--> sameperiodlastyear (year specified)-->2020

Create a measure and place on a card. Slice year value and see the start of year on the card.

Start of year= Startofyear(dimdate[Date])

Create a measure and place on a card. Slice year value and see the start of year value on the card.

start of year discount fee=calculate([Total DF], Startofyear(dimdate[Date]))

Create a measure and place on a card. Slice quarter value and see the start of quarter on the card.

Start of quarter= Startofquarter(dimdate[Date])

Create a measure and place on a card. Slice quarter value and see the start of quarter value on the card.

start of quarter discount fee=calculate([Total DF], Startofquarter(dimdate[Date]))

Create a measure and place on a card. Slice month value and see the start of month on the card.

Start of month= Startofmonth(dimdate[Date])

Create a measure and place on a card. Slice month value and see the start of month value on the card.

start of month discount fee=calculate([Total DF], Startofmonth(dimdate[Date]))

Create measures (for each function one measure) and place on cards. Slice year value and see the result on cards.

Last Date=lastdate(dimdate[Date])

LastNonBlankDate=lastnonblank(dimdate[date],factpayments[discount_fee])

LastNonBlankValue=lastnonblankvalue(dimdate[date],factpayments[discount_fee])

Create measures (for each function one measure) and place on cards. Slice year value and see the result on cards.

FirstDate=Firstdate(dimdate[Date])

FirstNonBlankDate=Firstnonblank(dimdate[date],factpayments[discount_fee])

FirstNonBlankValue=Firstnonblankvalue(dimdate[date],factpayments[discount_fee])

Create child measure like below [Modeling->New Measure→] and place on card. Slice Year (in slicer) value and see the ClosingBalanceYear value on card.

Closing Balance Year

=CLOSINGBALANCEYEAR(sum(payments[salesincome]),payments[salesdate])

Create child measure like below [Modeling->New Measure→] and place on card. Slice Quarter (in slicer) value and see the ClosingBalanceQuarter value on card.

Closing Balance Quarter

=CLOSINGBALANCEQUARTER(sum(payments[salesincome]),payments[salesdate])

Create child measure like below [Modeling->New Measure→] and place on card. Slice Month (in slicer) value and see the ClosingBalanceMonth value on card.

Closing Balance Month

=CLOSINGBALANCEMONTH(sum(payments[salesincome]),payments[salesdate])

Create child measure like below [Modeling->New Measure→] and place on card. Slice Year (in slicer) value and see the OpeningBalanceYear value on card.

Opening Balance Year

=OPENINGBALANCEYEAR(sum(payments[salesincome]),payments[salesdate])

Create child measure like below [Modeling->New Measure→] and place on card. Slice Quarter (in slicer) value and see the OpeningBalanceQuarter value on card.

Opening Balance Quarter

=OPENINGBALANCEQUARTER(sum(payments[salesincome]),payments[salesdate])

Create child measure like below [Modeling->New Measure→] and place on card. Slice Month (in slicer) value and see the OpeningBalanceMonth value on card.

Opening Balance Month

=OPENINGBALANCEMONTH(sum(payments[salesincome]),payments[salesdate])

Year To Date, Quarter To Date, and Month To Date Functions

Year To Date (YTD): Start of year to till date

Quarter To Date(QTD): Start of Quarter to till date

Month To Date(MTD): Start of Month to till date

Example: Today is 2021-05-13

YTD=2021-01-01 to 2021-05-13

QTD=2021-04-01 to 2021-05-13

MTD=2021-05-01 to 2021-05-13

DatesYTD= Return Dates for YTD

DatesQTD=Return Dates for QTD

DatesMTD=Return Dates for MTD

Note: We can use the above three functions result for any kind of aggregation [Total, Avg, Min, Max etc...]

TotalYTD=Return Total Value for YTD

TotalQTD=Return Total Value for QTD

TotalMTD=Return Total Value for MTD

Note: We can use the above three functions result only for Total [no other aggregation]

Practice:

Create four slicers

a) Year Slicer

b) Quarter Slicer

c) Month Slicer

d) Day Slicer

Creating Year to date total value measure

1st Way:

Create a measure and place on a card. Slice the values and see the result.

Total YTD= totalytd([Total DF],DimDate[Date])

2nd Way:

Total YTD1 =Calculate([Total DF, DatesYTD(DimDate[Date]))

Note: Specify Filter and End date if required

Creating Quarter to date total value measure

1st Way:

Create a measure and place on a card. Slice the values and see the result.

Total QTD= totalytd([Total DF],DimDate[Date])

2nd Way:

Total QTD1 =Calculate([Total DF, DatesQTD(DimDate[Date]))

Note: Specify Filter and End date if required

Creating Month to date total value measure

1st Way:

Create a measure and place on a card. Slice the values and see the result.

Total MTD= totalmtd([Total DF],DimDate[Date])

2nd Way:

Total MTD1 =Calculate([Total DF, DatesMTD(DimDate[Date]))

Note: Specify Filter and End date if required

Creating Year to date average value measure

Create a measure and place on a card. Slice the values and see the result.

Avg YTD

=Calculate(Average(FactPayments[Discount_Fee]), DatesYTD(DimDate[Date]))

Note: Similarly Quarter to date and Month to date we can implement

Creating Year to date Max value measure

Create a measure and place on a card. Slice the values and see the result.

MAX YTD

=Calculate(MAX(FactPayments[Discount_Fee]), DatesYTD(DimDate[Date]))

Note: Similarly Quarter to date and Month to date we can implement

Creating Year to date Min value measure

Min YTD

=Calculate(MIN(FactPayments[Discount_Fee]), DatesYTD(DimDate[Date]))

Note: Similarly Quarter to date and Month to date we can implement

Three simple SQL Queries to work with data validation at source

```
--Total Discount Fee value

select sum(f.discount_fee) from factpayments f

inner join dimdate d on f.date=d.date


--Year wise total discount fee value

select d.year,sum(f.discount_fee) from factpayments f

inner join dimdate d on f.date=d.date

group by d.year

--Total discount fee value between two dates

select sum(f.discount_fee) from factpayments f

inner join dimdate d on f.date=d.date

where d.datekey between datekey1  and datekey2

--where d.month = monthvalue --month validation
--where d.day=dayvalue  --day validation
--where d.year=yearvalue --year validation
--where d.qtr=quartervalue --qtr validation
```

CASE STUDY 1:

SSMS-->DATABASE ENGINE QUERIES

```
SELECT * FROM FactPayments;
```

Scenario: Display dates between '2017-01-01' and '2018-01-01'

DAX

Modelling menu→New table

1st way [using DATESBETWEEN]:

```
=DATESBETWEEN(FactPayments[Date],DATE(2017,01,01),DATE(2018,01,01))
```

1st way [using DATESINPERIOD]:


```
=DATESINPERIOD(FactPayments[Date],DATE(2017,01,01),366,DAY))
```

SQL QUERY

```
SELECT DATE FROM FACTPAYMENTS

WHERE DATE BETWEEN '2017-01-01' AND '2018-01-01'
```

Scenario:

Display the total discount fee for the dates between

'2017-01-01' and '2018-01-01'

DAX

1st way [using DATESBETWEEN]:

```
Total_Discount_Fee =
CALCULATE(sum(FactPayments[Discount_Fee]),
DATESBETWEEN(FactPayments[Date],DATE(2017,01,01),DATE(2018,01,01)))
```

2nd way [using DATESINPERIOD]:

```
Total_Discount_Fee =
CALCULATE(sum(FactPayments[Discount_Fee]),
DATESINPERIOD(FactPayments[Date],DATE(2017,01,01),366,DAY))
```


SQL QUERY

```
SELECT sum(discount_fee) FROM FACTPAYMENTS
WHERE DATE BETWEEN '2017-01-01' AND '2018-01-01'
```

SQL QUERIES FOR "YEAR TO DATE DATA"

What are the ways available to findout Year To Date / any interval to date values?

There are four ways [read below]

Create four slicers in the report and test the below by browsing the slicer values.

To get proper result, always use four slicers for testing

a) Year

b) Quarter

c) Month

d) Day

Note:

If we ignore a particular slicer and evaluate, then all values in the respective interval selected.

CASE STUDY-1

DAX EXPRESSIONS FOR "YEAR TO DATE DATA"

Scenario: Total discount fee for the year 2019

SQL QUERY

1ST WAY:

```
SELECT SUM(F.DISCOUNT_FEE) FROM FactPayments F
INNER JOIN DIMDATE D ON F.DATE=D.DATE
WHERE F.DATE BETWEEN '2019-01-01' AND '2019-12-31'
```

2ND WAY:

```
SELECT SUM(F.DISCOUNT_FEE) FROM FactPayments F
INNER JOIN DIMDATE D ON F.DATE=D.DATE
--WHERE F.DATE BETWEEN '2019-01-01' AND '2019-12-31'

WHERE D.YEAR=2019
```


Note: Change dates based on your selection (year, quarter, month, and day)

There are Three to Four ways to do this

a) DATESYTD : Return dates for the year to date

b) TOTALYTD : Return total value for the year to date

c) DATESINPERIOD: Return dates

d) DATESBETWEEN : Return dates

e)

Note: Options a and b are simple than the other two.

DAX FUNCTIONS, take measure and write the function, place it on cards

1st WAY [DATESYTD]:

year to date _1st way =

CALCULATE(sum(FactPayments[Discount_Fee]),**DATESYTD**(DimDate[Date]))

2nd Way [TOTALYTD]

year to date _ 2ndway = **TOTALYTD**(sum(FactPayments[Discount_Fee]),DimDate[Date])

3rd Way [DATESBETWEEN]

Year To Date _3way =

CALCULATE(sum(FactPayments[Discount_Fee]),

DATESBETWEEN(DimDate[Date],MIN(DimDate[Date]),max(DimDate[Date])))

4th Way [DATESINPERIOD]

Year To Date _3way = CALCULATE(sum(FactPayments[Discount_Fee]),

DATESINPERIOD(DimDate[Date],DATE(2018,01,01),365,DAY))

Note: The interval and the value should be changed according to the context of data.

Year to date value 2=

TOTALYTD(sum(FactPayments[Discount_Fee]),DimDate[Date],**ALL('DimDate'),"02/28"**)

CASE STUDY 2:

Scenario: Total discount fee for January month 2019

DAX AND SQL QUERIES FOR "MONTH TO DATE" DATA

1ST WAY:

```
SELECT SUM(F.DISCOUNT_FEE) FROM FactPayments F

INNER JOIN DIMDATE D ON F.DATE=D.DATE

WHERE F.DATE BETWEEN '2019-01-01' AND '2019-01-31'
```

2ND WAY:

```
SELECT SUM(F.DISCOUNT_FEE) FROM FactPayments F

INNER JOIN DIMDATE D ON F.DATE=D.DATE
WHERE D.MONTH=1 AND D.YEAR=2019
```


Note: Change dates based on your selection (year, quarter, month, and day)

1st WAY [DATESMTD]:

year to date _1st way =

```
CALCULATE(sum(FactPayments[Discount_Fee]),DATESMTD(DimDate[Date]))
```

2nd Way [TOTALMTD]

```
year to date _ 2ndway = TOTALMTD(sum(FactPayments[Discount_Fee]),DimDate[Date])
```

3rd Way [DATESBETWEEN]

```
Year To Date _3way =

CALCULATE(sum(FactPayments[Discount_Fee]),DATESBETWEEN(DimDate[Date],MIN(DimDate[D

ate]),max(DimDate[Date])))
```

4rd Way [DATESINPERIOD] [Month Contect of March]

```
Year To Date _3way = CALCULATE(sum(FactPayments[Discount_Fee]),

DATESINPERIOD(DimDate[Date],DATE(2019,03,31),-30,DAY ))
```

CASE STUDY 3:

Scenario: Total discount fee for the first quarter in 2019

DAX AND SQL QUERIES FOR " QUARTER TO DATE DATA"

1ST WAY:

```
SELECT SUM(F.DISCOUNT_FEE) FROM FactPayments F

INNER JOIN DIMDATE D ON F.DATE=D.DATE

WHERE F.DATE BETWEEN '2019-01-01' AND '2019-03-31'
```

Note: Change dates based on your selection (year, quarter, month, and day)

2ND WAY:

```
SELECT SUM(F.DISCOUNT_FEE) FROM FactPayments F
INNER JOIN DIMDATE D ON F.DATE=D.DATE

WHERE D.QUARER=1 AND D.YEAR=2019
```


DAX EXPRESSIONS FOR QUARTER TO DATE DATA

1st WAY [DATESQTD]:

```
year to date _1st way =

CALCULATE(sum(FactPayments[Discount_Fee]),DATESQTD(DimDate[Date]))
```

2nd Way [TOTALQTD]

```
year to date _ 2ndway = TOTALQTD(sum(FactPayments[Discount_Fee]),DimDate[Date])
```

3rd Way [DATESBETWEEN]

```
Year To Date _3way =

CALCULATE(sum(FactPayments[Discount_Fee]),

DATESBETWEEN(DimDate[Date],MIN(DimDate[Date]),max(DimDate[Date])))
```

3rd Way [DATESINPERIOD]

```
Year To Date _3way = CALCULATE(sum(FactPayments[Discount_Fee]),

DATESINPERIOD(DimDate[Date],DATE(2019,03,31),-90,DAY))
```

CASE STUDY 4:

[Create New table with the below expression, preview data in data view]

= ADDCOLUMNS(DimDate,"Dateadd",DATEADD(DimDate[Date],-1,year))

[Create New table with the below expression, preview data in data view]

date_Table_More_Columns = ADDCOLUMNS(Calendar_auto,"YEAR",

YEAR(Calendar_auto[Date]),"MONTH",MONTH(Calendar_auto[Date]))

CASE STUDY 5:

SQL QUERY :

```
SELECT F.DATE FROM FactPayments F

INNER JOIN DIMDATE D ON F.DATE=D.DATE

WHERE F.DATE BETWEEN '2019-01-01' AND '2019-01-20'
```


Note: Change dates based on your selection (year, quarter, month, and day)

DAX:

[Create New table with the below expression, preview data in data view]

Partial_Calendar = DATESBETWEEN(DimDate[Date],date(2019,01,01),date(2019,01,20))

[Create New table with the below expression, preview data in data view]

Partial Calendar= DATESINPERIOD(DimDate[Date],DATE(2019,01,20),-20,DAY)

CASE STUDY 6:

SQL QUERY :

```
SELECT SUM(F.DISCOUNT_FEE) FROM FactPayments F

INNER JOIN DIMDATE D ON F.DATE=D.DATE

WHERE F.DATE BETWEEN '2019-02-04' AND '2019-03-31'
```

Note: Change dates based on your selection (year, quarter, month, and day)

DAX:

[Create new measure and place on a card]

```
Specific dates total = CALCULATE(SUM(FactPayments[Discount_Fee]),

DATESBETWEEN(DimDate[Date],date(2019,02,04),date(2019,03,31) ))
```


[Create new measure and place on a card]

```
Specific dates total = CALCULATE(SUM(FactPayments[Discount_Fee]),

DATESINPERIOD(DimDate[Date],date(2019,03,31),-55,DAY ))
```

CASE STUDY 7: Last periods [last year]

There are three ways

A) **Sameperiod last year**

B) **Parallelperiods**

C) **Previousyear**

Finding Last Year Data

1st way:

[Create New Measure with the below expression, Place it on a card see the data]

Last year data

```
= CALCULATE(SUM(FactPayments[Discount_Fee]), SAMEPERIODLASTYEAR(DimDate[Date]))
```

2nd way:

[Create New Measure with the below expression, Place it on a card see the data]

Last year data

```
= CALCULATE(SUM(FactPayments[Discount_Fee]), PARALLELPERIOD(DimDate[Date],-1,year))
```

Note: One year data considered in parallel period, it has other options to filter more.

3rd Way:

[Create New Measure with the below expression, Place it on a card see the data]

Last year data

```
= CALCULATE(SUM(FactPayments[Discount_Fee]), PREVIOUSYEAR(DimDate[Date]))
```

Same result in the Year context [year selection]

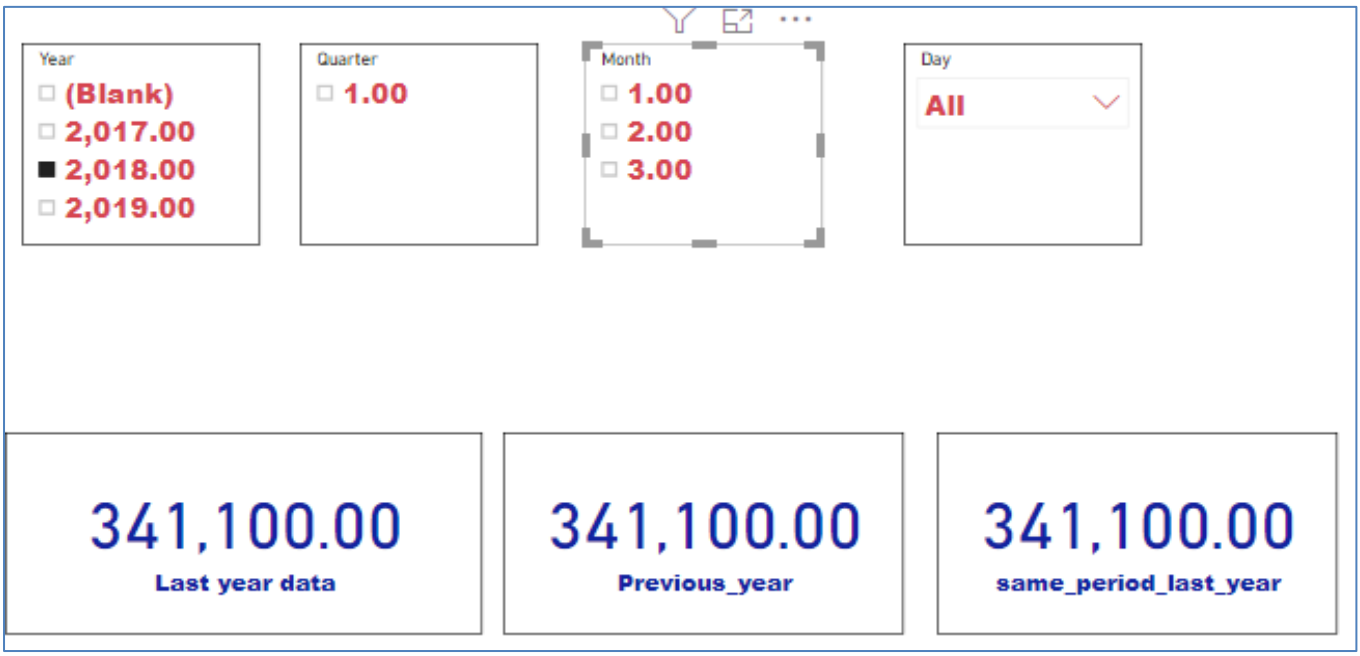
PARALLELPERIOD(DimDate[Date],-1,year)

SAMEPERIODLASTYEAR(DimDate[Date])

PREVIOUSYEAR(DimDate[Date])

SQL:

```
SELECT SUM(F.DISCOUNT_FEE) FROM FactPayments F

INNER JOIN DIMDATE D ON F.DATE=D.DATE

WHERE D.YEAR=2017
```



They differ with result when other intervals also added [Quarter, Month, Day etc...]

Specially Sameperiodlastyear will be different with other intervals.

CASE STUDY 8: Two years back data

PARALLELPERIOD(DimDate[Date],-2,year)

SAMPERIODLASTYEAR(SAMEPERIODLASTYEAR(DimDate[Date]))

PREVIOUSYEAR(PREVIOUSYEAR(DimDate[Date]))

In the "Year Context" the below three give you same result

```
SELECT SUM(F.DISCOUNT_FEE) FROM FactPayments F
INNER JOIN DIMDATE D ON F.DATE=D.DATE
WHERE D.YEAR=2017
```

Three ways

1st Way:

[Create New Measure with the below expression, Place it on a card see the data]

Two years back data = CALCULATE(SUM(FactPayments[Discount_Fee]),

SAMEPERIODLASTYEAR(SAMEPERIODLASTYEAR(DimDate[Date])))

2nd Way:

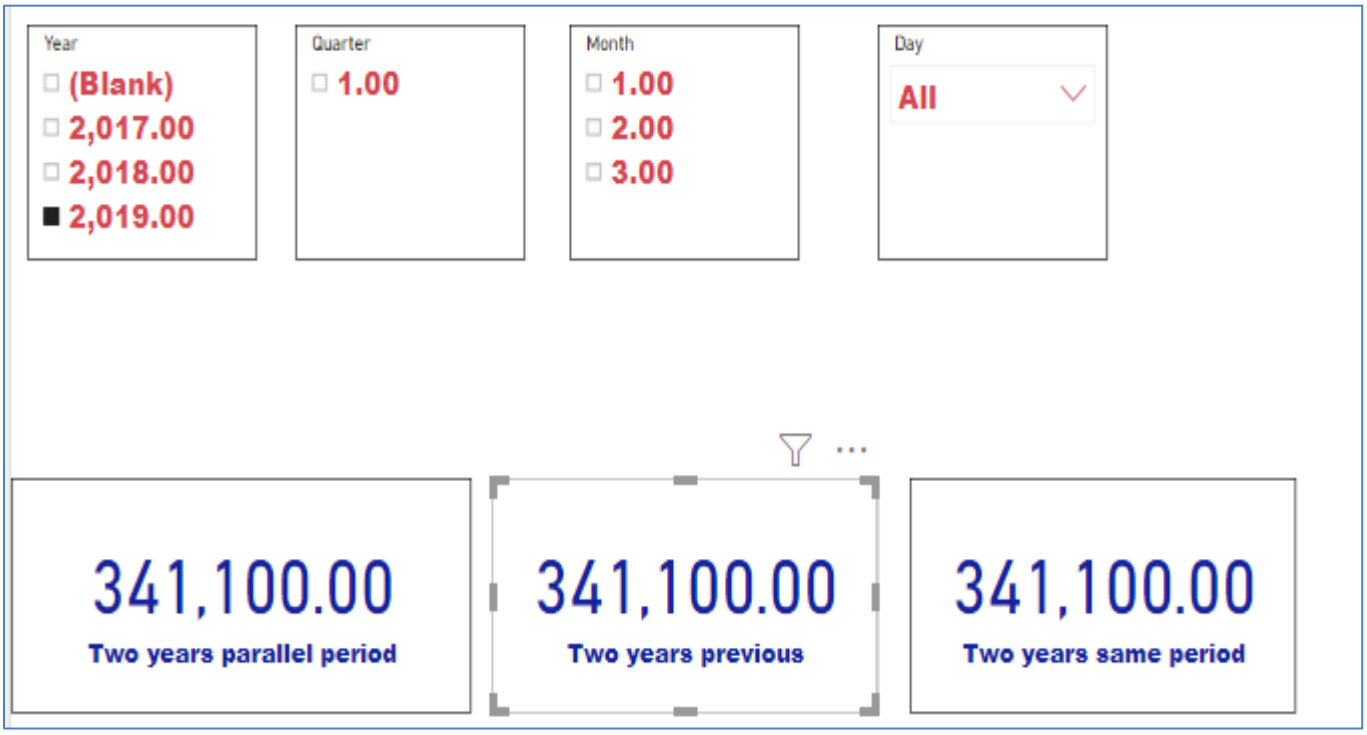
Two years back data = CALCULATE(SUM(FactPayments[Discount_Fee]),

PARALLELPERIOD(DimDate[Date],-2,year))

3nd Way:

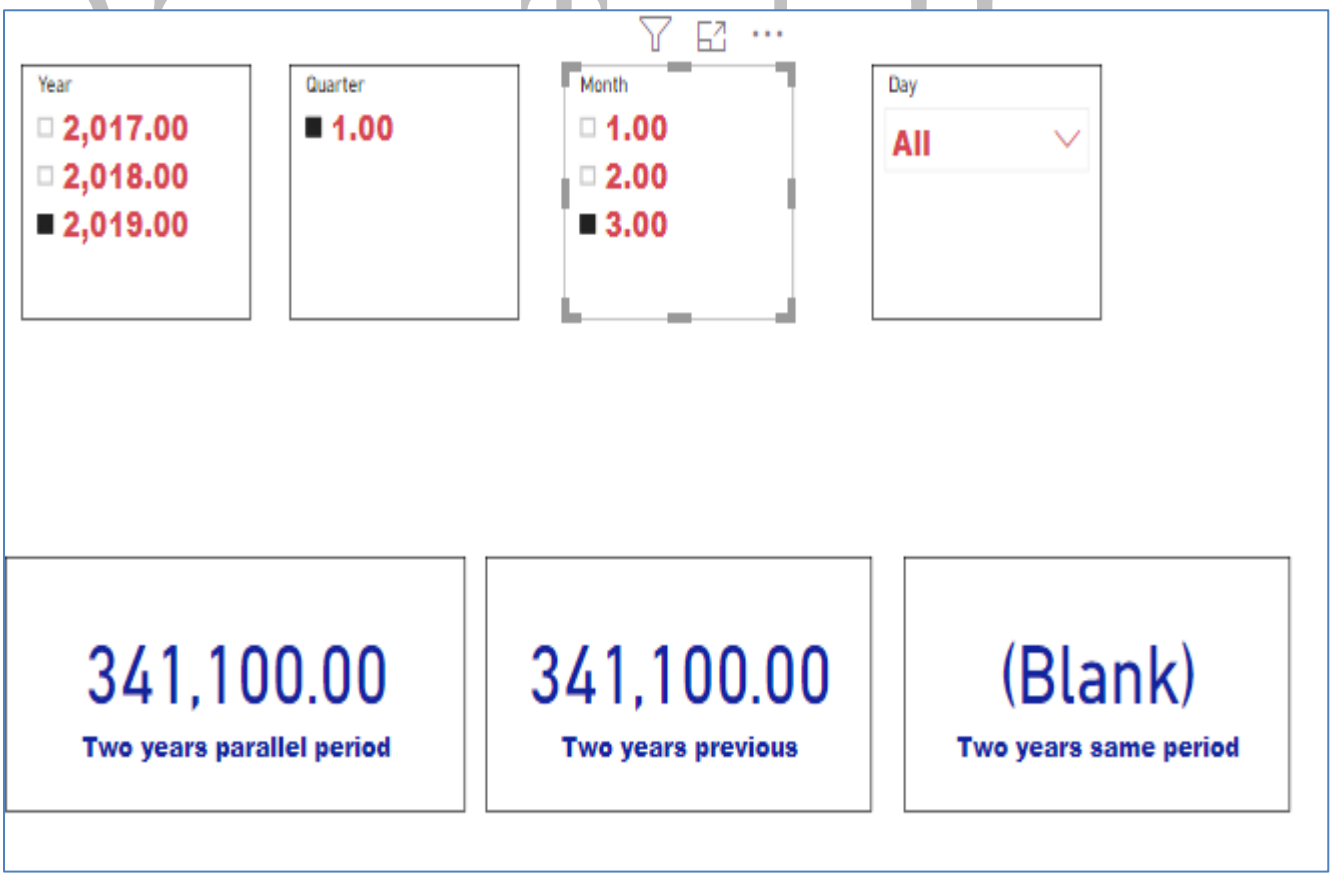
Two years previous data = CALCULATE(SUM(FactPayments[Discount_Fee]),

PREVIOUSYEAR(PREVIOUSYEAR(DimDate[Date])))



Their values are different if the context is changed

As Sameperiodlastyear is not getting respective data, it is showing blank.



BEST ADVANTAGE WITH SAMEPERIOD LAST YEAR:

Last year or two years same period data [Year / Quarter / Month / Day filters applicable] it is

suitable

BEST ADVANTAGE WITH PREVIOUS YEAR:

Last year or two years back data [YEAR filter applicable] it is suitable

Note:In the above two cases less coding suffifient, for more years back or forward, Parallel Period is suitable.

BEST ADVANTAGE WITH PARALLEL PERIOD:

Current year or two years same **any interval Previous or Next data [year / quarter / month/ day]** it is suitable

Example:

5 years back data

PARALLELPERIOD(DimDate[Date],-5,year))

5 years Next data

PARALLELPERIOD(DimDate[Date],5,year))

20 months back data

PARALLELPERIOD(DimDate[Date],-20,Month))

20 months forward data

PARALLELPERIOD(DimDate[Date], 20,Month))

3 quarters back data

PARALLELPERIOD(DimDate[Date],-3,quarter))

CASE STUDY 9 NEXTDAY

[Create New Measure with the below expression, Place it on a card see the data]

today sales=sum(SUM(FactPayments[Discount_Fee])

next day=CALCULATE(SUM(FactPayments[Discount_Fee]), **nextday(dimdate[date]))**

Note: Parallel Period is not suitable for this operation

CASE STUDY 10 NEXTMONTH

SQL Query

```
SELECT SUM(F.DISCOUNT_FEE) FROM FactPayments F

INNER JOIN DIMDATE D ON F.DATE=D.DATE

WHERE D.YEAR=2019 AND D.MONTH=3
```

DAX:

1st Way:

```
Nextmonth_parallel =

CALCULATE(sum(FactPayments[Discount_Fee]),NEXTMONTH(DimDate[Date]))
```

2nd Way:

```
Nextmonth_parallel =

CALCULATE(sum(FactPayments[Discount_Fee]),

PARALLELPERIOD(DimDate[Date],1,MONTH))
```

CASE STUDY 10 NEXTQUARTER

SQL Query

```
SELECT SUM(F.DISCOUNT_FEE) FROM FactPayments F

INNER JOIN DIMDATE D ON F.DATE=D.DATE

WHERE D.YEAR=2019 AND D.QUARTER=1
```

DAX:

1st Way:

```
Nextmonth_parallel =

CALCULATE(sum(FactPayments[Discount_Fee]),NEXTQUARTER(DimDate[Date]))
```

2nd Way:

```
Nextmonth_parallel =

CALCULATE(sum(FactPayments[Discount_Fee]),

PARALLELPERIOD(DimDate[Date],1,QUARTER))
```

CASE STUDY 11 NEXTYEAR

SQL Query

```
SELECT SUM(F.DISCOUNT_FEE) FROM FactPayments F

INNER JOIN DIMDATE D ON F.DATE=D.DATE

WHERE D.YEAR=2018
```

DAX:

1st Way:

```
Nextmonth_parallel =

CALCULATE(sum(FactPayments[Discount_Fee]),NEXTYEAR(DimDate[Date]))
```

2nd Way:


```
Nextmonth_parallel =

CALCULATE(sum(FactPayments[Discount_Fee]),

PARALLELPERIOD(DimDate[Date],1,YEAR))
```

CASE STUDY 12 NEXT TWO YEARS

SQL Query

```
SELECT SUM(F.DISCOUNT_FEE) FROM FactPayments F

INNER JOIN DIMDATE D ON F.DATE=D.DATE

WHERE D.YEAR=2017
```

DAX:

1st Way:

```
Nextmonth_parallel =

CALCULATE(sum(FactPayments[Discount_Fee]),NEXTYEAR(NEXTYEAR(DimDate[Date])))
```

2nd Way:

```
Nextmonth_parallel = CALCULATE(sum(FactPayments[Discount_Fee]),

PARALLELPERIOD(DimDate[Date],2,YEAR))
```

CASE STUDY 13 NEXT TWO QUARTERS

SQL Query

```
SELECT SUM(F.DISCOUNT_FEE) FROM FactPayments F
INNER JOIN DIMDATE D ON F.DATE=D.DATE
WHERE D.YEAR=2017
```

DAX:

1st Way:

```
Nextmonth_parallel = CALCULATE(sum(FactPayments[Discount_Fee]),
NEXTQUARTER(NEXTQUARTER(DimDate[Date])))
```

2nd Way:

```
Nextmonth_parallel = CALCULATE(sum(FactPayments[Discount_Fee]),
PARALLELPERIOD(DimDate[Date],2,QUARTER))
```

CASE STUDY 14 NEXT TWO MONTHS

SQL Query

```
SELECT SUM(F.DISCOUNT_FEE) FROM FactPayments F
INNER JOIN DIMDATE D ON F.DATE=D.DATE
WHERE D.YEAR=2017
```

DAX:

1st Way:

```
Nextmonth_parallel =
CALCULATE(sum(FactPayments[Discount_Fee]),
NEXTMONTH(NEXTMONTH(DimDate[Date])))
```


2nd Way:

```
Nextmonth_parallel =
CALCULATE(sum(FactPayments[Discount_Fee]),
PARALLELPERIOD(DimDate[Date],2,MONTH))
```

Scenario: Display Last Non Blank DiscountFee Date

Create a measure and place on card, slice the years to see different years last non blank dates

LastNonBlankSales =

LASTNONBLANK(DimDate[Date],[TotalDiscountFee])

Scenario: Display Last Non Blank DiscountFee value

Create a measure and place on card, slice the years to see different years last non blank dates DicountFee

LastNonBlankSales =

CALCULATE([TotalDiscountFee],LASTNONBLANK(DimDate[Date],[TotalDiscountFee]))

Data Validation with Data Feed [Source Data]

```
SELECT F.DATE,SUM(F.DISCOUNT_FEE) FROM FactPayments F

GROUP BY F.DATE

ORDER BY F.DATE DESC
```


Scenario: Display First Non Blank DiscountFee Date

Create a measure and place on card, slice the years to see different years last non blank dates

LastNonBlankSales =

FIRSTNONBLANK(DimDate[Date],[TotalDiscountFee])

Scenario: Display FIRST Non Blank DiscountFee value

Create a measure and place on card, slice the years to see different years last non blank dates

DicountFee

LastNonBlankSales =

CALCULATE([TotalDiscountFee],FIRSTNONBLANK(DimDate[Date],[TotalDiscountFee]))

Data Validation with Data Feed [Source Data]

```
SELECT F.DATE,SUM(F.DISCOUNT_FEE) FROM FactPayments F

GROUP BY F.DATE

ORDER BY F.DATE DESC
```

Scenario: Display Last Date of the business in the context

Create a measure and place on card, slice the years to see different years last non blank dates

```
LastNonBlankSales =

LASTDATE(DimDate[Date],[TotalDiscountFee])
```

Scenario: Display Last date DiscountFee value

Create a measure and place on card, slice the years to see different years last non blank dates

DicountFee


```
LastNonBlankSales =

CALCULATE([TotalDiscountFee],LASTDATE(DimDate[Date],[TotalDiscountFee]))
```

Scenario: Display First date

Create a measure and place on card, slice the years to see different years last non blank dates

```
LastNonBlankSales =

FIRSTDATE(DimDate[Date],[TotalDiscountFee])
```

Scenario: Display First date DiscountFee value

Create a measure and place on card, slice the years to see different years last non blank dates

DicountFee

```
LastNonBlankSales =

CALCULATE([TotalDiscountFee],FIRSTDATE(DimDate[Date],[TotalDiscountFee]))
```

Scenario: Closing balance of Online courses in a month [month slicing]

```
ClosingBalanceOfAYEAR = CLOSINGBALANCEYEAR(
sum(FactPayments[Discount_Fee]),DimDate[Date],
FILTER(DimCourseMode,DimCourseMode[ModeID]="Online"))
```

Scenario: Opening balance of Online courses in a month [month slicing]

```
OpeningBalance=OPENINGBALANCEMONTH([TotalDiscountFee],DimDate[Date],
FILTER(DimCourseMode,DimCourseMode[ModeID]="Online"))
```

```
Slicerselect = CONCATENATEX(Values(Sheet1[System]),Sheet1[System],",")
```

Very important comparison document

RUNNING TOTALS	ALTERNATE-1	ALTERNATE-2	ALTERNATE-3	ALTERNATE-4	ALTERNATE-5
DATESMTD	TOTALMTD	DATESBETWEEN	DATESINPERIOD	SAME PERIOD LAST YEAR	PARALLELPERIOD
DATESQTD	TOTALQTD	DATESBETWEEN	DATESINPERIOD	SAME PERIOD LAST YEAR	PARALLELPERIOD
DATESYTD	TOTALYTD	DATESBETWEEN	DATESINPERIOD	SAME PERIOD LAST YEAR	PARALLELPERIOD
NEXTDAY	DATESBETWEEN	DATESINPERIOD		SAME PERIOD LAST YEAR [slicing]	
PREVIOUSDAY	DATESBETWEEN	DATESINPERIOD		SAME PERIOD LAST YEAR [slicing]	
NEXTYEAR/NEXTMONTH/ NETXT QUARTER	DATESBETWEEN	DATESINPERIOD		SAME PERIOD LAST YEAR [slicing]	PARALLELPERIOD [positive]
PREVIOUS YEAR/MONTH/QUARTER	DATESBETWEEN	DATESINPERIOD		SAME PERIOD LAST YEAR [slicing]	PARALLELPERIOD [negative]

FILTER FUNCTIONS

The below functions help to filter the data explicitly or implicitly

ADDMISSINGITEMS	The ADDMISSINGITEMS function will return BLANK values for the IsSubtotal columns of blank rows it adds.
FILTER	Filters based on condition
FILTERS	Find the number of filters applied on field or query
Distinct	Non duplicate values display
COUNTROWS	Count rows for non duplicate values
ALL	Skips the filters and use all rows
ALLEXCEPT	Skip all filters except the one we specified
Earlier	Previous value
Earliest	Mostly first value
HASONEFILTER	Checks the availability of at least One filter
RELATED	Works like lookup / join, to bring column value {single value} from another query
RELATEDTABLE	Get set of rows as a table {multiple rows} based on data selected (joined data)
KEEPFILTERS	Keeping filters on queries when drill through or some operations performed.
CROSSFILTER	Specifies the cross-filtering direction to be used in a calculation for a relationship that exists between two columns.
ISCROSSFILTERED	
CALCULATE	Perform calculations based on the expression
CALCULATE TABLE	Provides table of values based on condition
AllSelected	The ALLSELECTED function gets the context that represents all rows and columns in the query, while keeping explicit filters and contexts other than row and column filters. This function can be used to obtain visual totals in queries.
AllNoBlankRow	From the parent table of a relationship, returns all rows but the blank row, or all distinct values of a column but the blank row, and disregards any context filters that might exist.
USERELATIONSHIP	To use the inactive relationship columns for calculations this is helpful
VALUES	Gets columns or column values

Run the below query in the DAX Studio

```
/*
Q: Differences between Calculate and SUMX?

The below describes Calcualte and SUMX differences, and also explain how Filter is used

Calculate can use columns from different table

Sumx use columns from same table

Note:

a) Both can take condition / expression
b)Calculate can perform any aggregation (avg, count, max, min etc...)

Q: DIfference between Relatedtable and Calcualtetable.

Relatedtable will use model level join and bring set of records.

Calcualtetable take explicit condition and bring set of records.

*/

EVALUATE
(
Row
(
"2019 Total using Calculate", CALCULATE(sum(FactPayments[Discount_Fee]),
FILTER(DimDate,DimDate[Year]=2019)),
                --filtering on different table and then sum

"Hyd Average using Calculate",CALCULATE(AVERAGE(FactPayments[Discount_Fee]),
FILTER(DimLocation,DimLocation[LocID]="HYD")),

                --filtering on different table and then sum
"2019 data using sumx and calculate
table",sumx(CALCULATETABLE(FactPayments,DimDate[Year]=2019),FactPayments[Discount_Fee]),
                --filtering on same table [factpayments] and then sum

"Number of rows",COUNTROWS(DISTINCT(FactPayments[Discount_Fee]))
)
)

/* Displaying 2017 table data. The below functions uses dataset model condition. */

evaluate
(
CALCULATETABLE(FactPayments,DimDate[Year]=2017)
)
```

```
/* Display courname for each course. Here Related used because of 1-1 / many-1 relationship */

Evaluate
(
ADDCOLUMNS(FactPayments,"Course Name",related(DimCourse[Coursename]))
)

/* Display for each course the total discount fee from fact table. Here related table used for 1-many
relationship */

Evaluate
(
ADDCOLUMNS(DimCourse,"Each Couse
fee",SUMX(RELATEDTABLE(FactPayments),FactPayments[Discount_Fee]))
)

/* Using Values */

evaluate
(
VALUES(DimCourse[CourseID])
)

/*
Q: How do we establish relationship between between tables at runtime using DAX without going to
Model?
Ans:

Userelatioship function does this, It establishes the relationship at runtime.


Can we establish relationship between tables using DAX function without going to the Model?

Ans: Yes, By using USERELATIONSHIP

*/
```

```
EVALUATE
(

row(
"Join date total",
CALCULATE(sum(FactPayments[Discount_Fee]),USERELATIONSHIP(FactPayments[Join_Date],DimDate[
Date])),

   --model level I took Course End date and factpayments date as active relationship

"Enquiry Date total",
CALCULATE(sum(FactPayments[Discount_Fee]),USERELATIONSHIP(FactPayments[Enquiry_Date],DimDa
te[Date]))
)
)

/* Usage of cross filter
Q: How do we establish direction (crossfilter) between two queries at runtime using DAX function?
A: CROSSFILTER function

*/

evaluate
(
row
(
"Distinct courses in 2019",
CALCULATE(DISTINCTCOUNT(DimCourse[CourseID]),FILTER(DimDate,DimDate[Date]=2019)),
"Distinct courses in 2018",
CALCULATE(DISTINCTCOUNT(DimCourse[CourseID]),FILTER(DimDate,DimDate[Date]=2018))
--"Distinct courses in 2019",
CALCULATE(DISTINCTCOUNT(DimCourse[CourseID]),FILTER(DimDate,DimDate[Date]=2019)),

"Distinct courses in 2018",
CALCULATE(DISTINCTCOUNT(DimCourse[CourseID]),CROSSFILTER(FactPayments[CourseID],DimCourse[
CourseID],BOTH))
--"Distinct courses in 2019",
CALCULATE(DISTINCTCOUNT(DimCourse[CourseID]),FILTER(DimDate,DimDate[Date]=2019)),


)
)
```

Calculate wide range of usage

a)It does calculation based on expression.

b) It can perform evaluation between multiple tables [not possible in SUMX, AVGX, COUNTX etc. functions]

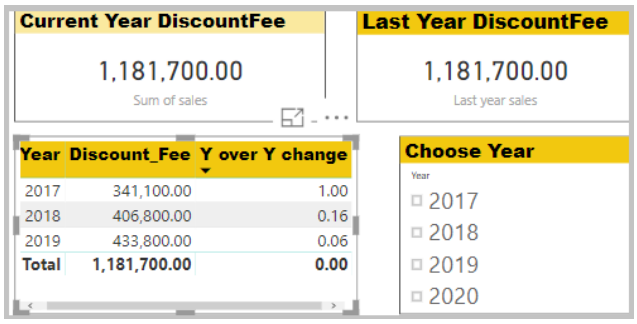
In Realtime we use like below for different types of filtering

Calculate (Sum / Avg /Count etc. <Calculation Argument1>,	Expression Argument 2
	Filter (FactPayments, Fatpayments[LocID]="HYD"]
	RelatedTable(FactPayments)
	CrossFilter(FactPayments[CouseID],DimCourse[CourseID],both)
	USERELATIONSHIP(FactPayments[JoinDate],DImDate[Date])
	Datesbetween (…)
	DatesInPeriod(…)
	Sameperiodlastyear(…)
	Parallelperiod ()
	Previousyear/previousmonth/previourquarter()
	Nextyear/Nextmonth/Nextquarter()
	FirstDate / FirstNonBlankDate /Last Date/ LastNOnblankDate ()…
	DatesYTD /DatesMTD /DatesQTD

Note:

We use independently SUM/SUMX/AVG/AVX/COUNT/COUNTX etc. TOTALYTD/TOTALMTD/TOTALQTD etc.

We don't place those inside Calcualte command



CALCULATING YEAR OVER YEAR PERCENTAGE

Legacy Method:Require three measures

Measure 1: Fulltotal

Sum of fee = sum(FACT_FEE[Amount])

Measure 2: Lastyear total

Sumof Previous Year = CALCULATE([Sum of fee],SAMEPERIODLASTYEAR(DimDate[Date]))

Measure3: Year over year growth

Sum of Amount YoY% = IF([Sum of fee] ,DIVIDE(([Sum of fee]-[Sumof Previous Year]), [Sum of Previous Year]))

Advanced approach: Using Variables

YoY% = var TotalFee = sum(FACT_FEE[Amount])

var SumofPreviousYear=CALCULATE(TotalFee, SAMEPERIODLASTYEAR(DimDate[Date]))

return if(TotalFee, DIVIDE(TotalFee-SumofPreviousYear, TotalFee))

YoY% = var TotalFee = sum(FACT_FEE[Amount])

var Lastyearfee=CALCULATE(TotalFee, SAMEPERIODLASTYEAR(DimDate[Date]))

return if(TotalFee, DIVIDE(TotalFee-Lastyearfee, TotalFee))

By using a variable, you can get the same outcome, but in a more readable way. In addition, the result of the

expression is stored in the variable upon declaration. It doesn't have to be recalculated each time it is used, as

it would without using a variable. This can improve the measure's performance.

Working on specific year total fee:

Specific Year Data = CALCULATE(sum(FACT_FEE[Amount]),DimDate[Year]=2018)

Specific month and year data

Specific Month Data =

CALCULATE(sum(FACT_FEE[Amount]),filter(filter(FACT_FEE,RELATED(DimDate[Year])=2018),related(DimDate[Mo

nthnumber])="02"))

The output of one filter is input to another [year data output is input to month]

Add Dateonly column to the Date table

Dateonly = (FORMAT ([Date], "YYYY-MM-DD"))

Specific Date Data

Specific Date Data =

CALCULATE(sum(FACT_FEE[Amount]),filter(filter(FACT_FEE,RELATED(DimDate[Year])=2017),related(DimDate[Dateonly])="2017-01-23"))

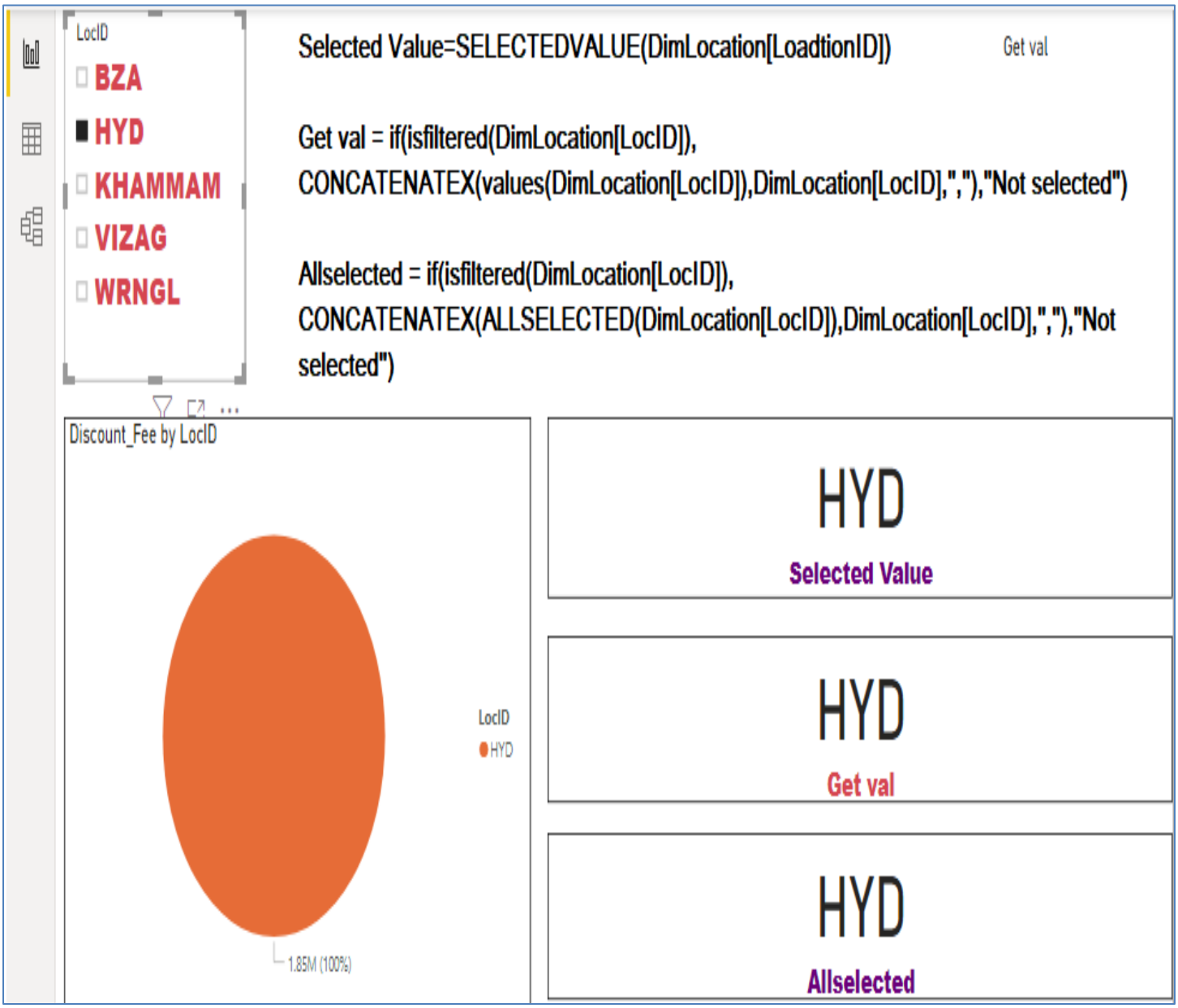
SELECTED VALUE, ALLSELECTED, VALUES, and CONCATEX

SELECTEDVALUE: To get the value selected value

ALLSELECTED: To get all selected values

VALUES: To get all values selected

CONTACTEX: To get the values and then concatenate



USERELATIONSHIP

Specifies the relationship to be used in a specific calculation as the one that exists between columnName1 and columnName2.

Syntax

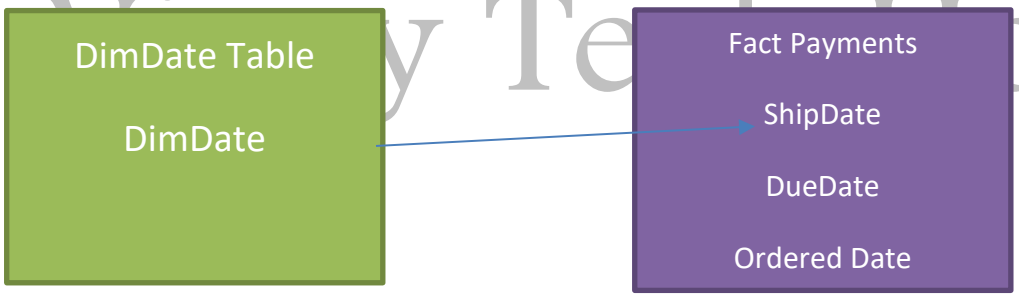
USERELATIONSHIP(<columnName1>, <columnName2>)

- If CALCULATE expressions are nested, and more than one CALCULATE expression contains a USERELATIONSHIP function, then the innermost USERELATIONSHIP is the one that prevails in case of a conflict or ambiguity.

Up to 10 USERELATIONSHIP functions can be nested;

however, your expression might have a deeper level of nesting, ie.

Classroom Example



Here Date table is Role Playing dimension table. Now it will show you the values based on Shipdate.

If you want the values based on OrderDate or DueDate, then use this function to establish relationship and get rows.

Create a role playing Date dimension like above with active and inactive relationships

Step1:

Go to FactPayments→ Power Query→ Right Click Date column→ Duplicate column [do it two times]

Rename first duplicate column to Date_Ship

Rename second duplicate column to Date_Due

Step2:

Connect Date dimension date column to Date_Ship and Date_Due with 1-Many relationship.

Note:

a)Now the Date dimension has three keys, one is active and remaining inactive, hence we call this as "ROLE PLAYING DIMENSION".

b)You can make active any relationship at anytime. [simply double click on inactive link and tick mark active]

Example: Display sum of discount fee based on the Date_Due between FactInternet Sales and Date Table

[Default the Date column has active relationship and Date_Due has inactive]

Create a new measure and write the below expression, see the result

```
Discount_Fee_Relationship
= CALCULATE(sum(FactPayments[Discount_Fee]),
USERELATIONSHIP(FactPayments[Date_Due],DimDate[Date]))
```

Here the relationship between Fact table and Date_Due table system uses to perform calculation

Other Examples:

The following sample expression is nested 3 levels deep but only 2 for USEREALTIONSHIP:

=CALCULATE(CALCULATE(CALCULATE(<anyExpression>, USERELATIONSHIP(t1[colA], t2[colB])), t99[colZ]=999), USERELATIONSHIP(t1[colA], t2[colA]))

Example

The following sample shows how to override the default, active, relationship between InternetSales and DateTime tables. The default relationship exists between the OrderDate column, in the InternetSales table, and the Date column, in the DateTime table.

To calculate the sum of internet sales and allow slicing by ShippingDate instead of the traditional OrderDate you need to create a measure, [InternetSales by ShippingDate] using the following expression:

=CALCULATE(SUM(InternetSales[SalesAmount]),

USERELATIONSHIP(InternetSales[ShippingDate], DateTime[Date]))

CrossFilter

Specifies the cross-filtering direction to be used in a calculation for a relationship that exists between two columns.

Syntax

CROSSFILTER(<columnName1>, <columnName2>, <direction>)

Remarks

- In the case of a 1:1 relationship, there is no difference between the one and both direction.
- CROSSFILTER can only be used in functions that take a filter as an argument, for example: CALCULATE, CALCULATETABLE, CLOSINGBALANCEMONTH, CLOSINGBALANCEQUARTER, CLOSINGBALANCEYEAR, OPENINGBALANCEMONTH, OPENINGBALANCEQUARTER OPENINGBALANCEYEAR, TOTALMTD, TOTALQTD and TOTALYTD functions.
- CROSSFILTER uses existing relationships in the model, identifying relationships by their ending point columns.

BiDi:= CALCULATE([Distinct Count of ProductKey], CROSSFILTER(FactInternetSales[ProductKey], DimProduct[ProductKey] , Both))

Practical: [Finding Distinct Courses Count]

Directly you will not get distinct coursesID count in an year due to 1-Many between DimCourse and FactPayments

Create a measure like this and place on the card.

Distinct count of CourseID =DistinctCount(DimCourse[COURSEID])

Take Year Slicer, change year and see the result. **It won't change.**

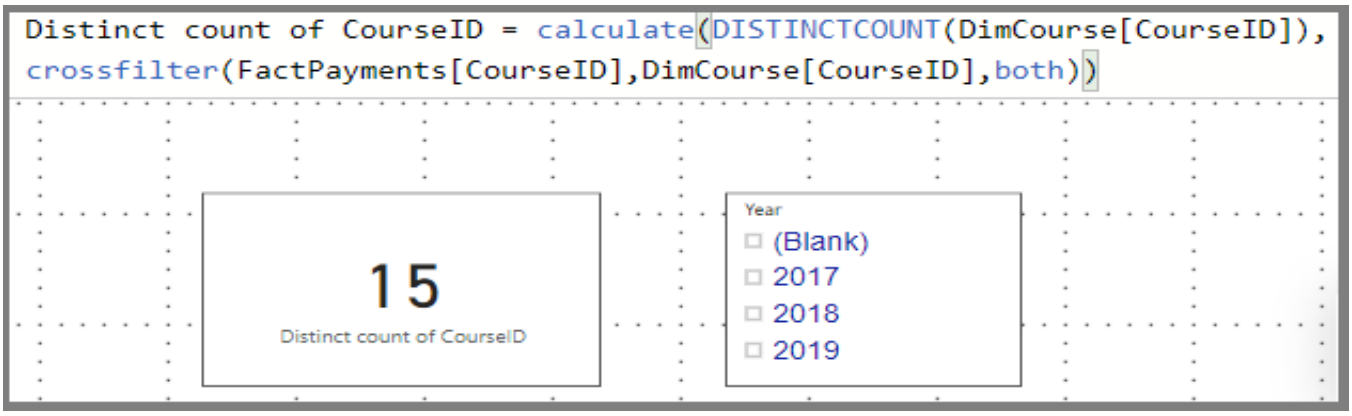
Two ways to resolve

1st way:

Create a measure with below the expression and see:

Distinct count of CourseID

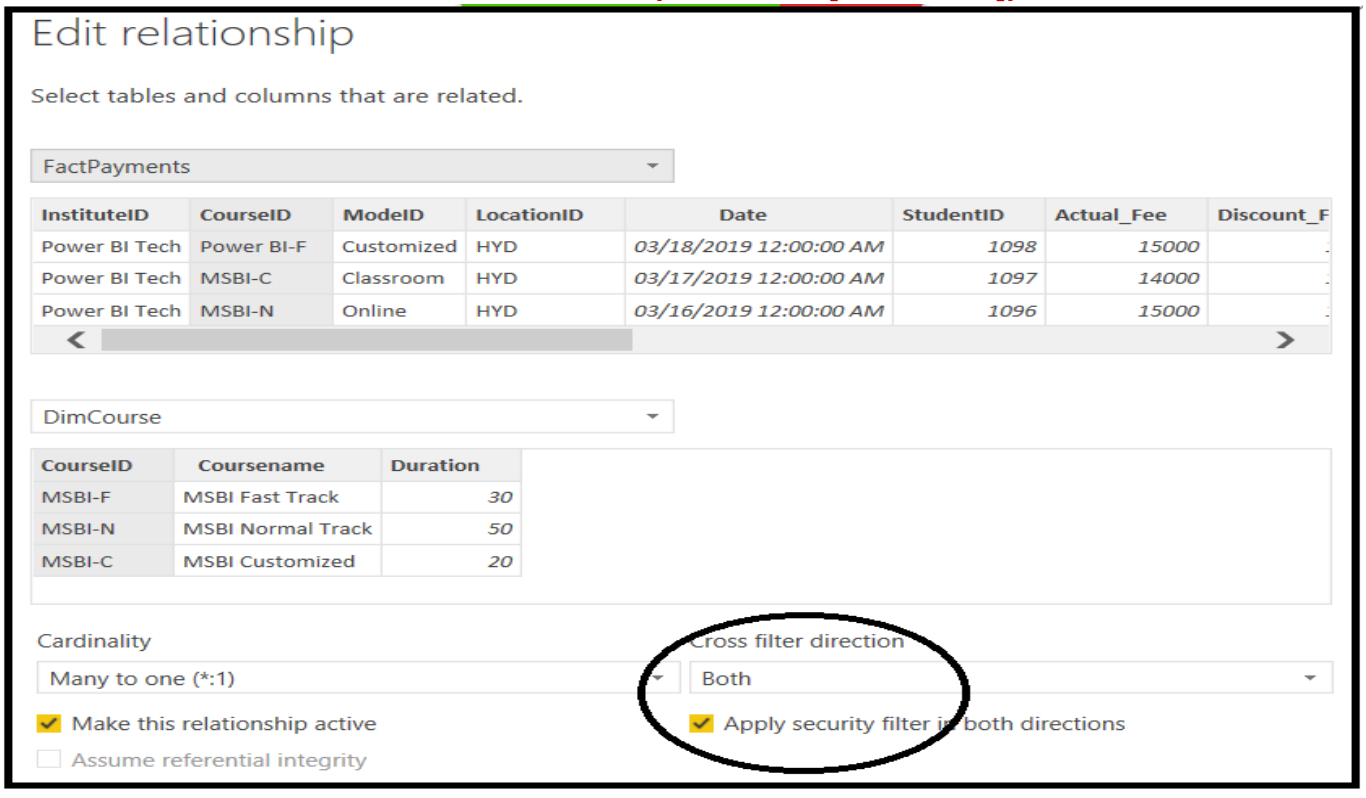
=calculate(DISTINCTCOUNT(DimCourse[CourseID]),crossfilter(FactPayments[CourseID],DimCourse[CourseID],both))



Second way:

Go to the model and provide the below, and this measure will give you same accurate result like first way.

Distinct count of CourseID =DistinctCount(DimCourse[COURSEID])



FILTER:

Scenario: Display total discount fee of 2019 year.

SQL Query:

```
SELECT SUM(F.DISCOUNT_FEE) FROM FactPayments F

INNER JOIN DIMDATE D ON F.DATE=D.DATE

WHERE D.YEAR=2019
```

Create a measure like below and use on a card

```
Total Discount Fee 2019 = CALCULATE(sum(FactPayments[Discount_Fee]),

FILTER(DimDate,DimDate[Year]=2019))
```

ALL:

Scenario:

Take an year slicer and below two measures on two cards and verify the result. When year changes Discount total 1 changes (Sumx_ALL), where as Discount total 2 (Sumx_NO_ALL) does not change.

Card 1 Measure:

```
Sumx_ALL = sumx(all(FactPayments),FactPayments[Discount_Fee])
```

Card 2 Measure:

```
Sumx_NO_ALL = SUMX(FactPayments,FactPayments[Discount_Fee])
```

ALLEXCEPT

Scenario:

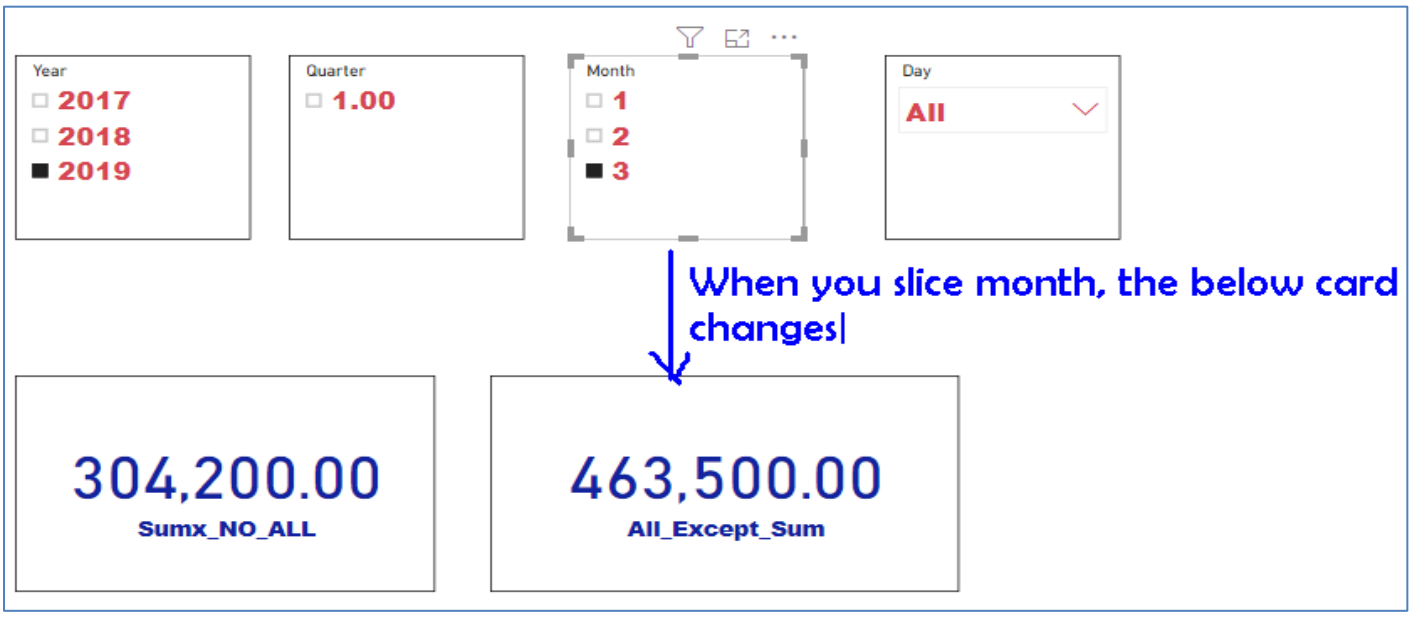
Allowing change of values when monthname only changes.

Take year slicer,change year, it does not change the values.

Take monthname slicer, if you change it will affect.

Create a measure like below and place on card to see the result.

```
Discount total 2 = calculate(sum(FactPayments[Discount_Fee]),

allexcept(DimDate,DimDate[Monthname]))
```



CALCULATE: [Return single value of result]

```
calculate(sum(FactPayments[Discount_Fee]),
allexcept(DimDate,DimDate[Monthname]))
```

CALCULATETABLE [Return a table of rows]

Scenario: Display 2017 year data

```
SELECT F.* FROM FactPayments F
INNER JOIN DIMDATE D ON F.DATE=D.DATE
WHERE D.YEAR=2017
```

Create a new table and write the below expression, preview the table

Modeling menu→new Table

```
Caltable 2017 rows= CALCULATETABLE(FactPayments,DimDate[Year]=2017)
```

Explanation: 2017 year data comes from FactPayments and create a table

Scenario: Total Discount fee in 2017 using calculate table

```
SELECT
SUM(F.DISCOUNT_FEE)
FROM  (SELECT F.* FROM FactPayments F
INNER JOIN DIMDATE D ON F.DATE=D.DATE
WHERE D.YEAR=2017) F
```

Create a new table and write the below expression, preview the table

Modeling menu→new measure

Calcuatetable_2017_total =

sumx(CALCULATETABLE(FactPayments,DimDate[Year]=2017),FactPayments[Discount_Fee])

Note:

The above 2017 data using Calculate and Filter

=CALCULATE(Sum(FactPayments],FILTER(DimDate[Year]=2017))

COUNTROWS AND DISTINCT

I need the count of distinct discountfee issued in the business fact table.

Create a measure with the below expression and take on a card

Number of rows=COUNTROWS(DISTINCT(FactPayments[Discount_Fee]))

Here distinct will et distinct values, later system finds the count of values.

FILTER

SQL Query:

```
SELECT SUM(F.DISCOUNT_FEE) FROM FactPayments F
INNER JOIN DIMDATE D ON F.DATE=D.DATE
WHERE F.MODEID='Online'
```

Display total discount fee for online Using SUMX

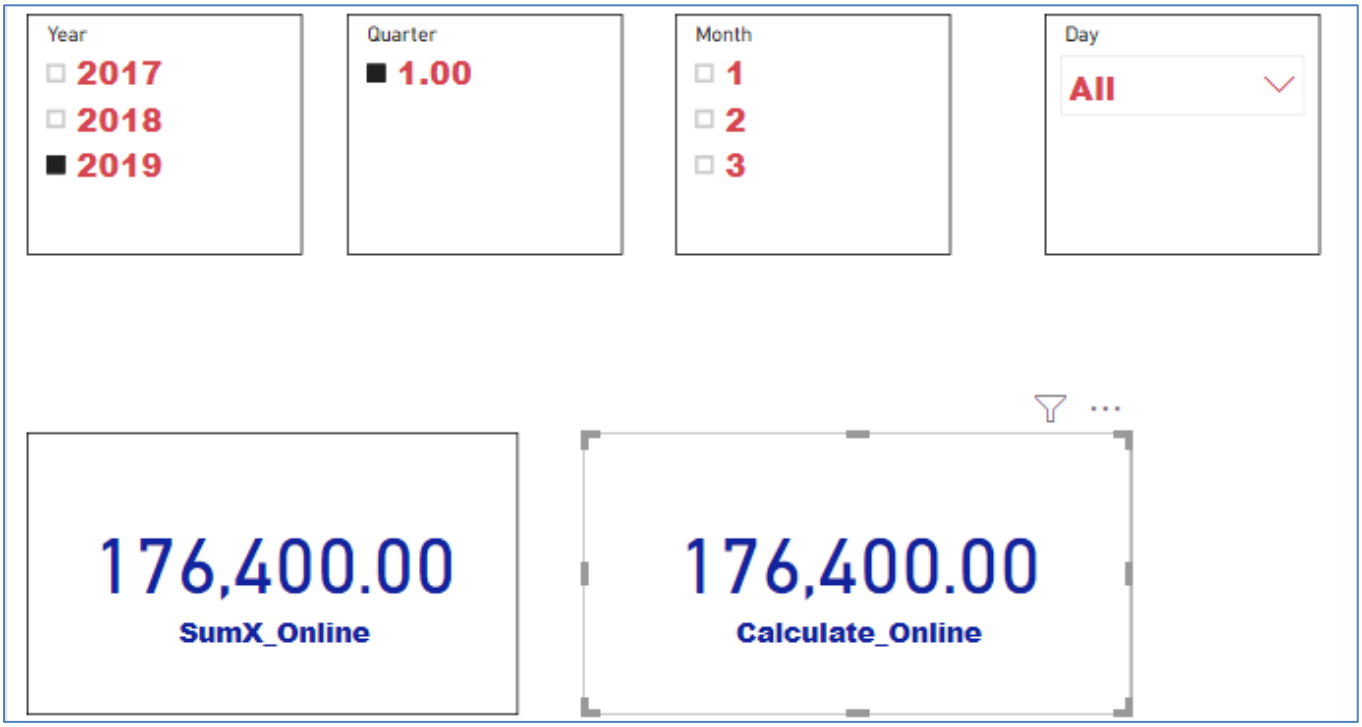
Create a measure like below and place on a card

```
Total fee for Online=
sumx(filter(FactPayments,FactPayments[ModeID]="Online"),
FactPayments[Discount_Fee])
```

Using CALCULATE

Create a measure like below and place on a card

```
Total fee for online=Calcualte(Sum(FactPayments[Discount_Fee]),
filter(FactPayments,FactPayments[ModeID]="Online"))
```



SUM X **always take same table [single] fields** in the first and second argument.

Calculate **can take multiple table arguments.** It uses internal model join condition to perform calculation.

Performance Tip: Single table filtering and calculations, then **SUMx gives better performance**.

FILTERS

Create a measure with the below expression and take on a card

```
Numberoffilters = COUNTROWS(filters(FactPayments[CourseID]))
```

HASONEFILTER

Scenario: If courseID has filter, then it will show you count. Otherwise returns BLANK.

Create a measure with the below expression and take on a card

```
DirectFilter =
IF(HASONEFILTER(FactPayments[CourseID]),
FILTERS(FactPayments[CourseID]),BLANK())
```

RELATED:

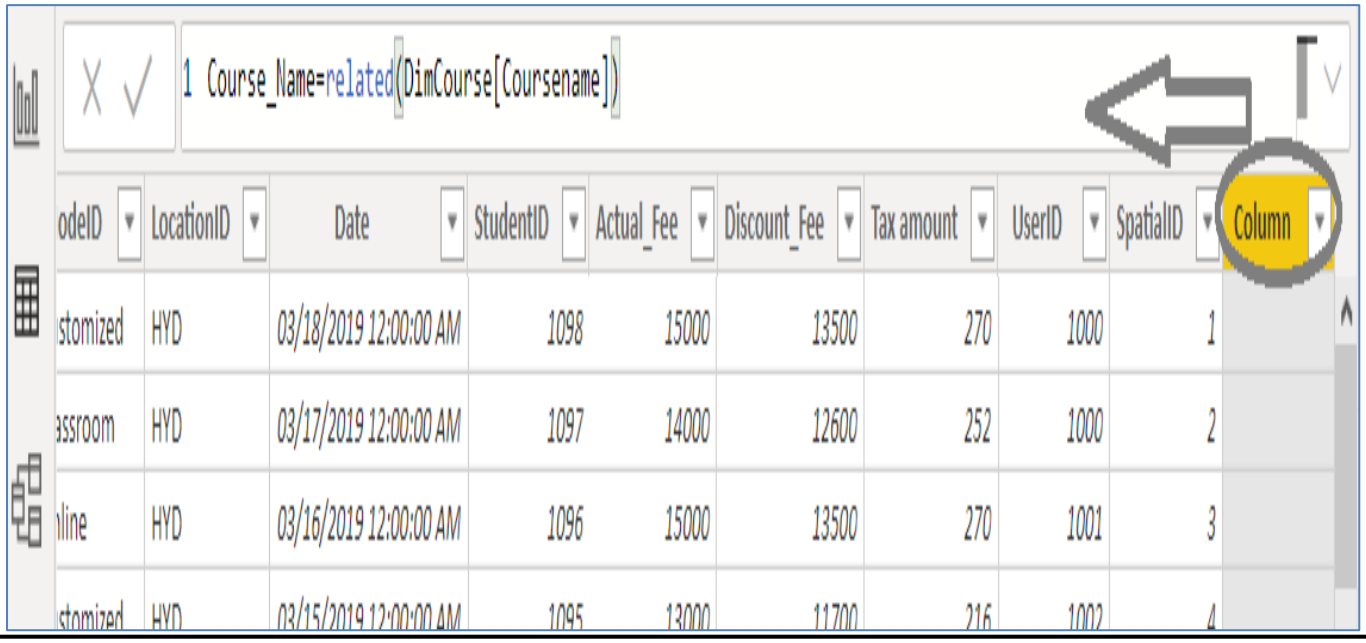
Scenario: Display CourseName in the fact table against to each CourseID value

SQL


```
SELECT F.*, M.Description  FROM FactPayments F
INNER JOIN DIMCOURSEMODE M ON F.ModeID=M.MODEID
```

Add new column in the FactPayment and write the below expression, and see the data view

Course_name=Related(DimCourse[CourseName])



Process:

Each courseID value from FactPayments passed to the model, join with DimCourses table, and then

Matched row CourseName it gets from DimCourse table

Scenario: MSBI Training Institute DiscountFee required from FactPayments

Create a measure like below and place on card

```
DiscountFee based on Institutename =

sumx(
      filter(FactPayments,related(DimInstitute[Institutename])="MSBI Training
         Institute"),

      FactPayments[Discount_Fee]
)
```

Function	Meaning
Related	One value pass and get one value (1:1) –Join and return a value
Related table	One value pass and many values retrieval (1:many)–Join and return table data
Lookupvalue	Performs a explicit inner join between tables and return column

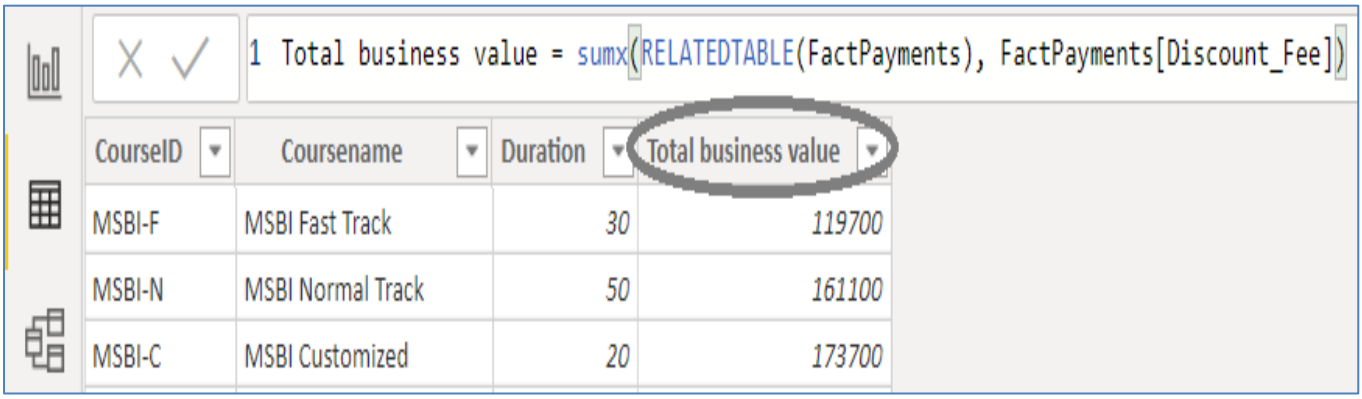
RELATEDTABLE:

Scenario: Findout in course table for each courseid total discount fee generated.

1.Go to course table→add new column→ write the below

```
SELECT C.COURSEID, SUM(F.DISCOUNT_FEE)    FROM FactPayments F

INNER JOIN DIMCOURSE C ON F.COURSEID=M.COURSEID

GROUP BY C.COURSEID
```

```
Total business value = sumx(RELATEDTABLE(FactPayments),

FactPayments[Discount_Fee])
```



```
1 Total business value = sumx(RELATEDTABLE(FactPayments), FactPayments[Discount_Fee])
```

CourseID	Coursename	Duration	Total business value
MSBI-F	MSBI Fast Track	30	119700
MSBI-N	MSBI Normal Track	50	161100
MSBI-C	MSBI Customized	20	173700

Process:

Each CouseID from DimCourse passed to FactPayments, compare and get table of rows, and then sum of all retrieved rows total created.

VALUES:

SQL QUERY:

```
SELECT COURSENAME FROM DimCourse
```


Scenario: Display only coursenames

Create new table or column and write the below expression, see the data view

```
Values_table = VALUES(DimCourse[Coursename])
```

Scenario: Display the count of coursenames

```
SELECT COUNT(COURSENAME) FROM DimCourse
```

DAX

Create new measure, write the below expression and display on the card

```
=COUNTROWS(VALUES(DimCourse[Coursename]))
```

EARLIER

Returns the current value of the specified column in an outer evaluation pass of the mentioned column.

EARLIER is useful for nested calculations where you want to use a certain value as an input and produce calculations based on that input. In Microsoft Excel, you can do such calculations only within the context of the current row; however, in DAX you can store the value of the input and then make calculation using data from the entire table.

EARLIER is mostly used in the context of calculated columns.

Syntax

EARLIER(<column>, <number>)

Exceptions

Description of errors

Remarks

EARLIER succeeds if there is a row context prior to the beginning of the table scan. Otherwise it returns an error.

Example: Display for each row how many UPPER level values available

Go to fact table and add new column and see the result

```
Count_Rows =
Countrows
(
filter(FactPayments, EARLIER(FactPayments[Discount_Fee])<FactPayments[Discount_Fee])
)+1
```

```
1 Count_Rows = countrows(filter(FACT_1,EARLIER(FACT_1[Tax amount])<FACT_1[Tax amount]))+1
```

Date	StudentID	Actual_Fee	Discount_Fee	Tax amount	DiscountValue	GRanking	Count_Rows
Monday, March 18, 2019	1098	15000	13500	270	1500	8	3
Sunday, March 17, 2019	1097	14000	12600	252	1400	10	4
Friday, March 15, 2019	1095	12000	10800	216	1200	14	6
Wednesday, March 13, 2019	1093	13000	11700	234	1300	12	5
Monday, March 11, 2019	1091	10000	9000	180	1000	17	7
Wednesday, March 7, 2018	1035	16000	14400	288	1600	6	2
Wednesday, January 3, 2018	1002	18000	16200	324	1800	3	1
							8

Earlier will work like outer condition value, gets first row in the table and passes to the complete table of rows to apply filter condition, and then found rows +1 displayed. Repeat this process for each row in the table.

Scenario 2:

A)Take a duplicate query for the Factpayments and then keep only Taxamount

b) Implement RANKX, RANK.EQ, and EARLIER on the column


```
Count_Rows =

Countrows
(
filter('FactPayments (2)', EARLIER('FactPayments (2)'[Tax amount]) >
'FactPayments (2)'[Tax amount])
)
```

```
1 Count_Rows =
2 Countrows
3 (
4 filter('FactPayments (2)', EARLIER('FactPayments (2)'[Tax amount]) > 'FactPayments (2)'[Tax amount])
5 )
6
7
```

Each value compared to Outer Value — Comparing — The list of values in the table Inner Values

Tax amount	Rank	Global Rank	Count_Rows
342	1		8
324	2	3	7
306	3		6
288	4	6	5
270	5	8	4
252	6	10	3
234	7	12	2
216	8	14	1
180	9	17	

NOTE

In practice, the xVelocity in-memory analytics engine (VertiPaq) performs optimizations to reduce the actual number of calculations, but you should be cautious when creating formulas that involve recursion.

A new calculated column, **SubCategorySalesRanking**, is created by using the following formula.

```
=COUNTROWS(FILTER(ProductSubcategory,
EARLIER(ProductSubcategory[TotalSubcategorySales])
<ProductSubcategory[TotalSubcategorySales]))+1
```

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

1. The **EARLIER** function gets the value of *TotalSubcategorySales* for the current row in the table. In this case, because the process is starting, it is the first row in the table
2. **EARLIER**([*TotalSubcategorySales*]) evaluates to $156,167.88, the current row in the outer loop.
3. The **FILTER** function now returns a table where all rows have a value of *TotalSubcategorySales* larger than $156,167.88 (which is the current value for **EARLIER**).
4. The **COUNTROWS** function counts the rows of the filtered table and assigns that value to the new calculated column in the current row plus 1. Adding 1 is needed to prevent the top ranked value from become a Blank.
5. The calculated column formula moves to the next row and repeats steps 1 to 4. These steps are repeated until the end of the table is reached.

The **EARLIER** function will always get the value of the column prior to the current table operation. If you need to get a value from the loop before that, set the second argument to 2.

Earliest

Returns the current value of the specified column in an outer evaluation pass of the specified column.

Syntax

EARLIEST(<column>)

Return value

A column with filters removed.

Remarks

The EARLIEST function is similar to EARLIER, but lets you specify one additional level of recursion.

Example

The current sample data does not support this scenario.

=EARLIEST(<column>)


```
Count_Rows = countrows(filter(FACT_1,EARLIEST(FACT_1[Tax amount])<FACT_1[Tax amount]))+1
```

Date	StudentID	Actual_Fee	Discount_Fee	Tax amount	DiscountValue	GRanking	Count_Rows
Monday, March 18, 2019	1098	15000	13500	270	1500	8	3
Sunday, March 17, 2019	1097	14000	12600	252	1400	10	4
Friday, March 15, 2019	1095	12000	10800	216	1200	14	6
Wednesday, March 13, 2019	1093	13000	11700	234	1300	12	5
Monday, March 11, 2019	1091	10000	9000	180	1000	17	7
Wednesday, March 7, 2018	1035	16000	14400	288	1600	6	2
Wednesday, January 3, 2018	1002	18000	16200	324	1800	3	1
							8

A)Take a duplicate query for the Factpayments and then keep only Taxamount

b) Implement RANKX, RANK.EQ, and EARLIER on the column

```
Count_Rows =
Countrows
(
filter('FactPayments (2)', EARLIEST('FactPayments (2)'[Tax amount]) > 'FactPayments (2)'[Tax amount])
)
```

```
Count_Rows =
Countrows
(
filter('FactPayments (2)', EARLIER('FactPayments (2)'[Tax amount]) > 'FactPayments (2)'[Tax amount])
)
```

Each value compared to Outer Value > Comparing The list of values in the table Inner Values

Tax amount	Rank	Global Rank	Count_Rows
342	1		8
324	2	3	7
306	3		6
288	4	6	5
270	5	8	4
252	6	10	3
234	7	12	2
216	8	14	1
180	9	17	

INFORMATION FUNCTIONS

Most of the functions deal with true / false returning based on the function. Few functions provide information.

INFORMATION FUNCTIONS	Most of the functions work on boolean operation [True / False operation]	
CONTAINS ROW	Works like IN Clause, if multiple row values exists return TRUE, otherwise false.	Measure/ Col
CONTAINS	Works like EXISTS clause, if the value exists return TRUE	Measure/ Col
ISEVEN	Returns true if it is even	Measure/ Col
ISODD	Returns true if it is ODD	Measure/ Col
ISTEXT	Returns true if it is text value	Measure/ Col
USERNAME	Displays Domain name and user name at Desktop, EmailID as Service. **Useful for RLS at Power BI Desktop level**	Measure/ Col
USERPRICIPALNAME	Displays Domain name and user name at Desktop, EmailID as Service. **Useful for Row Level Security at Power BI Service Level**	Measure/ Col
LOOKUPVALUE	Returns required column values, based on search column against a value	Measure/ Col
ISERROR	Returns true if it is error	Measure/ Col
ISNONTEXT	Returns true if it is not a textual value	Measure/ Col
ISNUMBER	Returns true if it is a number	Measure/ Col
ISLOGICAL	Returns true if it is logical value	Measure/ Col
ISINSCOPE	Returns true when the specified column is the level in a hierarchy of levels.	Measure/ Col
ISONORAFTER	This function takes a variable number of triples, the first two values in a triple are the expressions to be compared, and the third parameter indicates the sort order. The sort order can be ascending (default) or descending.	Measure/ Col

Run the below queries in the DAX Studio

```
EVALUATE
(
Row
(
"Is even 4",iseven(4),--true
"Is Odd 3", isodd(3), --true
"Is Number 34", ISNUMBER(34), --true
"Is nontext 35",ISNONTEXT(35), ---true
"is nontext vinaytech",ISNONTEXT("vinaytech"), --false
"Is Text vinaytech", Istext("vinaytech"), --true
"Is logical true", Islogical(false), --true
"Is logical false", ISLOGICAL("ramana"),--false
"User Name",username(),--domain name \ username at desktop,  email id at service level
"User Principal Name", USERPRINCIPALNAME()--domain name \ username at desktop,  email id at
service level
)
)
```

```
EVALUATE
(
ROW
(

"Contains MSBI-M and Online",CONTAINS(FactPayments,FactPayments[CourseID],"MSBI-
M",FactPayments[ModeID],"Online"),
"Lookup MSBI-N", lookupvalue(DimCourse[Coursename],DimCourse[CourseID],"MSBI-N"),
"Is error 25/0", iserror(25/0),
"Is error 25/5", iserror(25/5)
)

)
```

```
/* Contains Row Example : Getting ModeIDs if Online and Classroom exists

select modeid from tablename
where modeid in ('Online','Classroom')

*/

EVALUATE
(
FILTER(ALL(DimCourseMode[ModeID]), CONTAINSROW({ "Online", "Classroom" }, [ModeID]))
)

/* Contains row example: Getting location and mode if they match Hyd and Online*/


EVALUATE
(
FILTER(SUMMARIZE(FactPayments, FactPayments[LocationID],FactPayments[ModeID]),
CONTAINSROW({ ("HYD", "Online") },

[LocationID], [ModeID]))

)
```

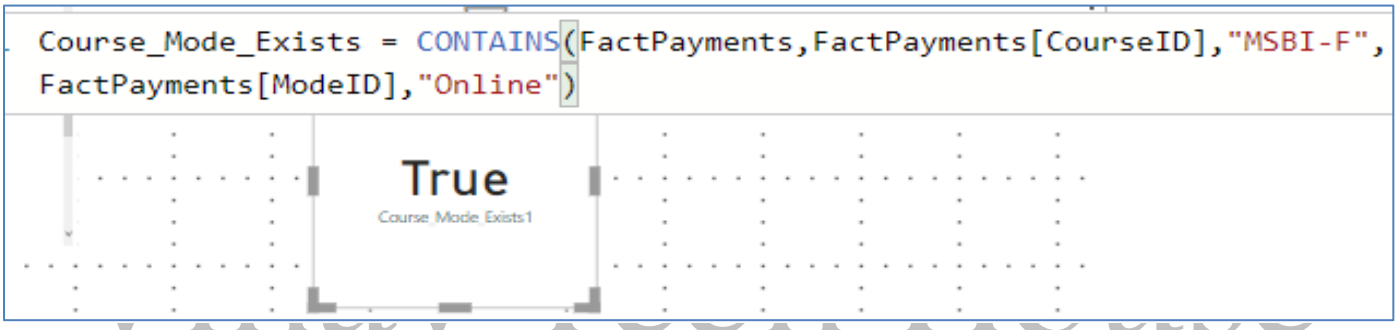
CONTAINS

Scenario: Return true or false, if courseID=”MSBI-F” and ModeID=”Online” available in fact table

Create a measure and place on a card

```
Course_Mode_Exists = CONTAINS(FactPayments,FactPayments[CourseID],"MSBI-F",
FactPayments[ModeID],"Online")
```

The condition is like Logical AND operations [all are matched, then only returns true]



CONTAINSROW Example:

SQL Query

```
SELECT ModeID FROM DimCourseMode
WHERE MODEID IN ('Online','Classroom')
```

Inside PBI Desktop

a) Take a table and write the below expression, view the data in data view

```
Mode_table=FILTER(ALL(DimCourseMode[ModeID]),

CONTAINSROW({ "Online", "Classroom" }, [ModeID]))
```

Or

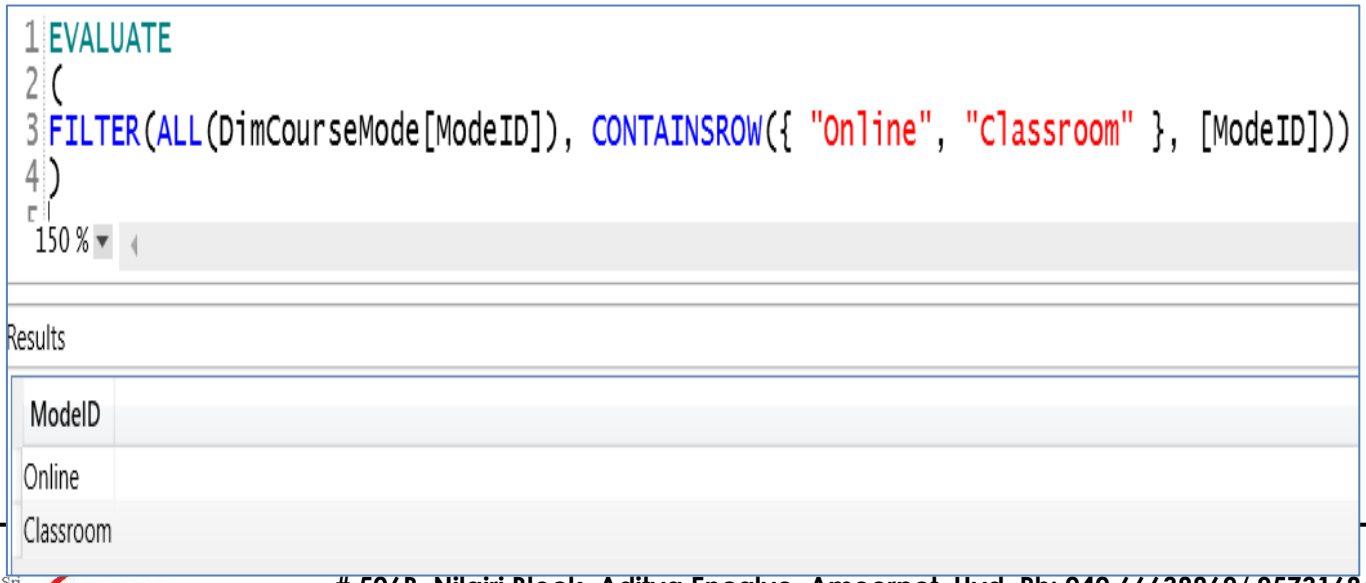
```
Online_Classroom_Table=FILTER(ALL(Factpayments),

CONTAINSROW({ "Online", "Classroom" }, Factpayments[ModeID]))
```

Example 1: Run the below queries in DAX studio

```
EVALUATE
(
FILTER(ALL(DimCourseMode[ModeID]), CONTAINSROW({ "Online", "Classroom" }, [ModeID]))
)
-- This query executed like below
```


a) Bring all ModeIDs irrespective of filter

b) With in the list, FILTER function applies condition {containsrow}



Example 2:

SQL Query:

```
SELECT  LocationID, MODEID  FROM FactPayments

WHERE ModeID='Online' and LocationID='HYD'  --filter in DAX

Group by ModeID,LocationID   --summarize in DAX
```

DAX Studio

```
EVALUATE
(

FILTER(SUMMARIZE(FactPayments, [LocID], [ModeID]), CONTAINSROW({ ("HYD", "Online") },

[LocID], [ModeID]))

)
```

Execution:

a)Summarized (grouped) data based on LocID and ModeID comes from FactPayments

b)Filter applies the condition on the grouped data, so that HYD and Online group information only displayed

```
1 EVALUATE
2 (
3 FILTER(SUMMARIZE(FactPayments, [LocID], [ModeID]), CONTAINSROW({ ("HYD", "Online") },
4 [LocID], [ModeID]))
5 )
6
```

150 %

Results

LocID	ModeID
HYD	Online

Scenario: Please find the total fee of online and hyd in the previous situation

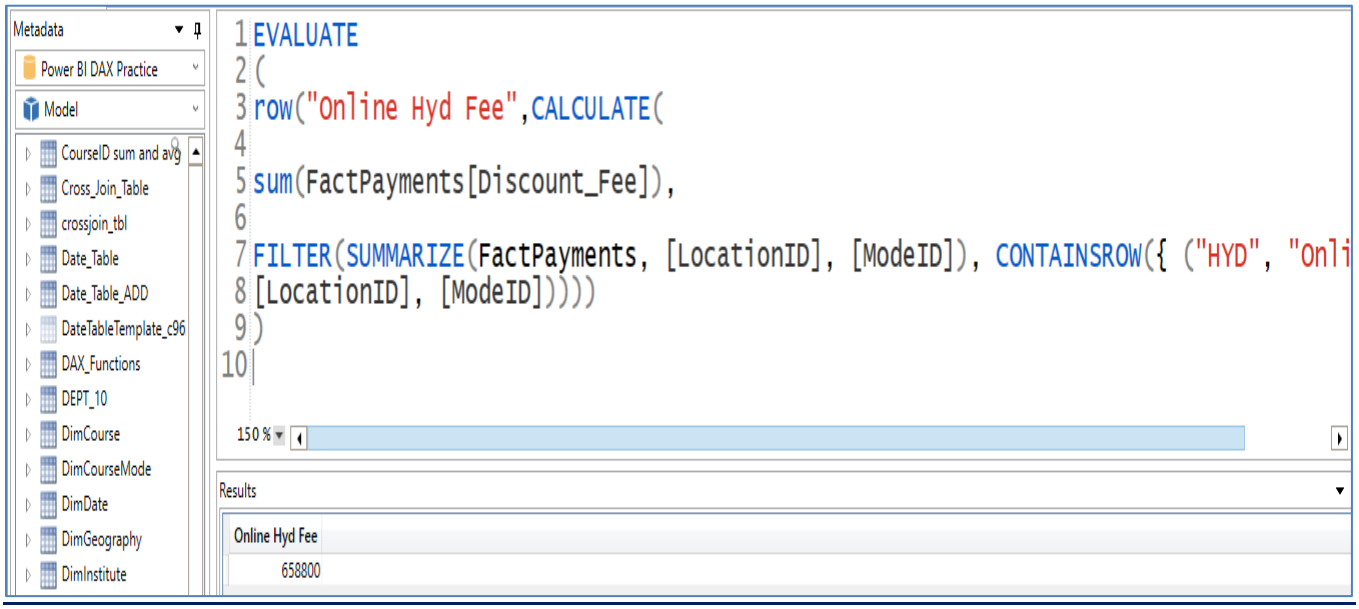
1st Way in the Desktop:

Create a measure, use on a card

```
CALCULATE(
sum(FactPayments[Discount_Fee]),
FILTER(SUMMARIZE(FactPayments, [LocationID], [ModeID]), CONTAINSROW({ ("HYD", "Online") },
[LocationID], [ModeID])))
```

2nd Way (DAX Studio): Write the below formula and see the result

```
EVALUATE
(
row("Online Hyd Fee",CALCULATE(
sum(FactPayments[Discount_Fee]),
FILTER(SUMMARIZE(FactPayments, [LocationID], [ModeID]), CONTAINSROW({ ("HYD", "Online") },
[LocationID], [ModeID])))
)
```

ISBLANK

Create a new column in the factpayments and see the result

=ISBLANK(FactPayments[Discount_Fee])

ISERROR

Create a new measure and place on a card

=iserror(25/0)

Result: True

Create a new measure and place on a card

=iserror(25/5)

Result: False

Create a new colum in the factpayments and see the result

Scenario: If there is an error substistuing with 99999

```
if(iserror(FactPayments[DiscountFee]/0),99999)
```

ISEVEN

Create a new measure and place on a card

=iseven(24)

Result: True

Create a new measure and place on a card

=iseven(25)

Result: False

Create a new column in the factpayments and see the result

=ISEVEN(FactPayments[Discount_Fee])

ISODD

Create a new measure and place on a card

=ISODD(25)

Result: True

Create a new measure and place on a card

=ISODD(24)

Result: False

Create a new colum in the factpayments and see the result

=ISODD(FactPayments[Discount_Fee])

ISLOGICAL

Create a new measure and place on a card

=islogical(TRUE)

Result: True

Create a new measure and place on a card

```
=IF(ISLOGICAL(25), "Is Boolean type or Logical", "Is different type")
```

ISNONTEXT

Create a new measure and place on a card

```
=IF(ISNONTEXT(1), "Is Non-Text", "Is Text")
```

ISNUMBER

Create a new measure and place on a card

```
=IF(ISNUMBER("123"), "Is
number",                    "Is Not number")
```

ISTEXT

Create a new measure and place on a card

```
=IF(ISTEXT("text"), "Is Text", "Is Non-Text")
```

ISONORAFTER

Scenario: Create a table with CourseID MSBI-F sort oder (Asc) and Coursename MSBI Fast Track sort order (ASC)

Modeling Menu-> New Table

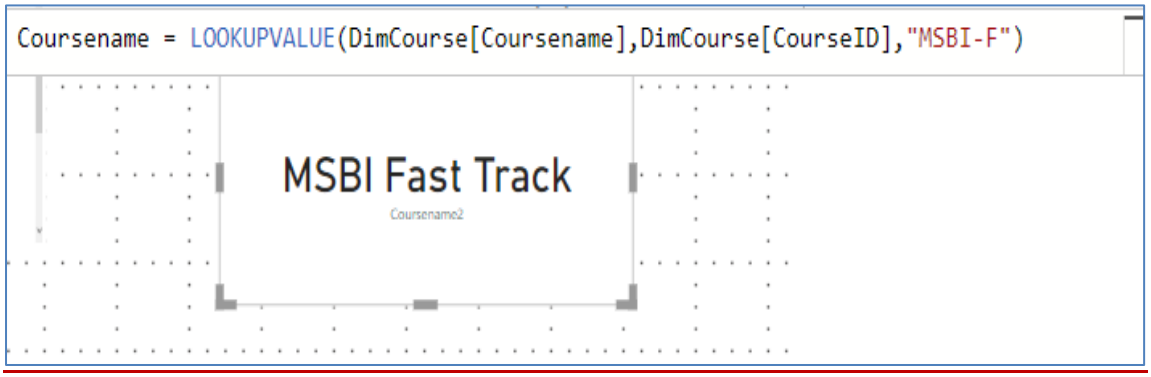
```
TBLNEW = FILTER(DimCourse, ISONORAFTER(DimCourse[CourseID], "MSBI-F", ASC,
DimCourse[Coursename], "MSBI Fast Track", ASC))
```

LOOKUP VALUE

Scenario: Getting Coursename if courseid is "MSBI-F"

Create a new measure and place on a card

```
Coursename=LOOKUPVALUE(DimCourse[Coursename],DimCourse[CourseID],"MSBI-F")
```



Comparion between Related and Lookupvalue

Go to FactPayments table, new column and write the below expression

```
Mode_Dec=
LOOKUPVALUE(DimCourseMode[Description],DimCourseMode[ModeID],
FactPayments[ModeID])
```

Go to FactPayments table, new column and write the below expression

```
Mode_Dec= Related(DimCourseMode[Description])
```

Observe result.

Both should show same result

Practice the below

Go to Model and remove the relationship between FactPayments and DimCourseMode, see the above columns result.

Related does not work as it required relationship, where as lookup dees not require relationship and joins on the fly, it will be able to show.

USERNAME

Create a new measure and place on a card

=USERNAME

Create a new measure and place on a card

=USERPRINCIPALNAME()

Scenario: Specify ALLOWED if the current user is in the list

Create a new measure and place on a card

1ST WAY:

```
Usravailable =
if(contains(DimUsers,DimUsers[Login],USERPRINCIPALNAME()),"Allowed",BLANK())
```

Execution:

a) If starts execution and goes to condition

b) Contains verifies the current user in the login list

c) If available, then ALLOWD will displayed, otherwise BLANK

2ND WAY:

```
Usravailable =

if(contains(DimUsers,DimUsers[Login],USERNAME()),"Allowed",BLANK())
```

OTHER FUNCTIONS

OTHER FUNCTIONS		
DATA TABLE	Helps to construct a dynamic table with rows --Close to CREATE TABLE	New Table
EXCEPT	Set thoery except [exclusive rows from first object]	New Table
INTERSECT	Set thoery intersect [common rows from both objects]	New Table
UNION	SET thoery except [APPEND rows from both objects]	New Table
GROUP	Group columns and uses a GROUP keyword as an argument	Cal/New Table
SUMMARIZECOLUMN	Grouping columns and perform aggregations	Cal/New Tab
GENERATE SERIES	Generate sequence of values –Identity topic in SQL Server	New Table/Col
NATURAL JOIN	Inner join [explicit join]	Cal/New Table
TREATAS	Treating the same rows from another object for operations on first and second object	Cal/New Table
ISEMPTY	If it is empty value returns true	Measure/Col
VAR	Variable [intermediate value holder, wherever used substitutes with a value]	

Run the below queries in the DAX Studio:

```
/* Empty Check validation, If empty returns TRUE, Otherwise False */

EVALUATE
(
ADDCOLUMNS(
DimCourse,
"IsEmpty Result", ISEMPTY(DimCourse)
)

)


/* Set theory practice
Create two tables with data
EMP1 --EID, ENM, ELOC --(1,'x','hyd')
EMP2 --EID, ENM, ELOC --(2,'y','mum')
*/

/* Perform Union operation (merging resultsets with duplicates)*/


EVALUATE
(

UNION(EMP1, EMP2)

)

/* Perform Interset operation (Common records between resultsets)*/


EVALUATE
(
INTERSECT(EMP1, EMP2)
)

/* Perform Except operation (Get Exclusive Records)*/


EVALUATE
(
EXCEPT(EMP1, EMP2)
)


/* Sequence of values starting from 1 and ending at 10 */

Evaluate
(
GENERATESERIES(1,10)
)
```

```
/* Sequence of values starting from 10 and ending at 50 by incrementing 5 */

Evaluate
(

GENERATESERIES(10,50,5)


)

/* Intitute and mode wise total discount fee */

evaluate
(

GROUPBY(FactPayments,FactPayments[InstituteID],FactPayments[ModeID],

"Total Fee",SUMx(currentgroup(),FactPayments[Discount_Fee]))

)

/* Finding the maximum value in each group created in the above formula

Execution of the below:

a) The Variable holds the result of GroupBY query
b) The Evaluate uses the variable and finds Max value in each group

*/

DEFINE
VAR DiscountFeeLocationMode =
GROUPBY (
FactPayments,
DimLocation[Locationname],
DimCourseMode[ModeID],
"Total Discount Fee", SUMX( CURRENTGROUP(), FactPayments[Discount_Fee])
)

Evaluate GROUPBY (
DiscountFeeLocationMode,
DimLocation[Locationname],
"Max Sales", MAXX( CURRENTGROUP(), [Total Discount Fee])
)

/* Summarize columns example */
Evaluate
(
SUMMARIZECOLUMNS(FactPayments[InstituteID],FactPayments[ModeID],FactPayments,"Total",sum(F
actPayments[Discount_Fee]))
)
```

```
/*
  Evaluating Rollupaddissubtotal to show locationid and grandtotalrow
  To show data in Reporting Format
*/

evaluate

(

SUMMARIZECOLUMNS

 (
        ROLLUPADDISSUBTOTAL('FactPayments'[LocationID], "IsGrandTotalRowTotal"),

       "SumDiscount_Fee", CALCULATE(SUM('FactPayments'[Discount_Fee]))
    )
)


EVALUATE
 TOPN
        (
    502,
    SUMMARIZECOLUMNS

  (
        ROLLUPADDISSUBTOTAL('FactPayments'[LocationID], "IsGrandTotalRowTotal"),

        "SumDiscount_Fee", CALCULATE(SUM('FactPayments'[Discount_Fee]))
    ),

    [IsGrandTotalRowTotal],

    0,

    'FactPayments'[LocationID],
    1
 )
ORDER BY

        [IsGrandTotalRowTotal] DESC, 'FactPayments'[LocationID]
```

DATA TABLE

Provides a mechanism for declaring an inline set of data values.

Syntax

DATATABLE (ColumnName1, DataType1, ColumnName2, DataType2..., {{Value1, Value2...}, {ValueN, ValueN+1...}...})

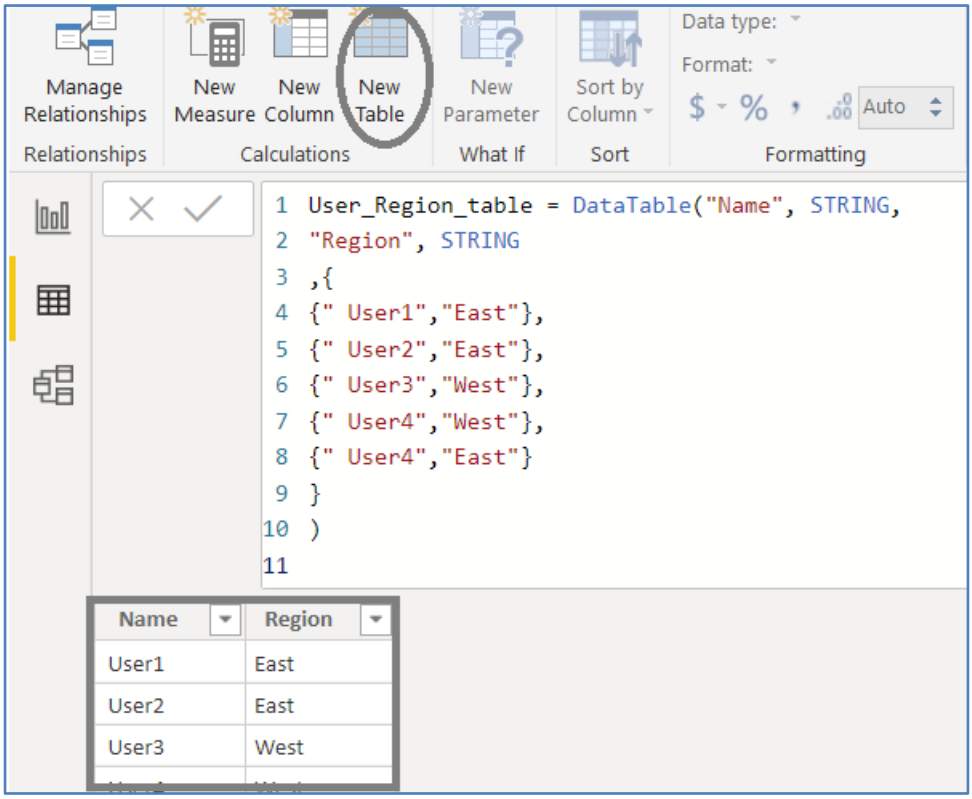
Example

Create New table (Modeling Menu→New Table) and write the below expression

```
DimLocation=DataTable("Name", STRING,
        "Region", STRING
        ,{
                {" User1","East"},
                {" User2","East"},
                {" User3","West"},
                {" User4","West"},
                {" User4","East"}
        }
)
```


SQL Query for the above:

```
CREATE TABLE DimLocation (Name Varchar(30), Region Varchar(30));
INSERT INTO DimLocation VALUES("User1","East"),("User2","East"),("User3","West"),
("User4","West"), ("User4","East")
```



ERROR

Raises an error with an error message. Suitable for error handling situation with manual error message.

Example 1

Write the below query and execute at DAX Studio

```
DEFINE
MEASURE DimProduct[Measure] =
  IF(
   SELECTEDVALUE(DimProduct[Color]) = "Red",
   ERROR("red color encountered"),
   SELECTEDVALUE(DimProduct[Color])
  )
EVALUATE SUMMARIZECOLUMNS(DimProduct[Color], "Measure", [Measure])
ORDER BY [Color]
```

Fails and raises and error message containing "red color encountered".

Example 2

The following DAX query:

```
DEFINE
MEASURE DimProduct[Measure] =
  IF(
   SELECTEDVALUE(DimProduct[Color]) = "Magenta",
   ERROR("magenta color encountered"),
   SELECTEDVALUE(DimProduct[Color])
  )
EVALUATE SUMMARIZECOLUMNS(DimProduct[Color], "Measure", [Measure])
ORDER BY [Color]
```

Returns the following table:

DIMPRODUCT[COLOR]	[MEASURE]
Black	Black
Blue	Blue
Grey	Grey
Multi	Multi
NA	NA
Red	Red
Silver	Silver
Silver\Black	Silver\Black
White	White
Yellow	Yellow

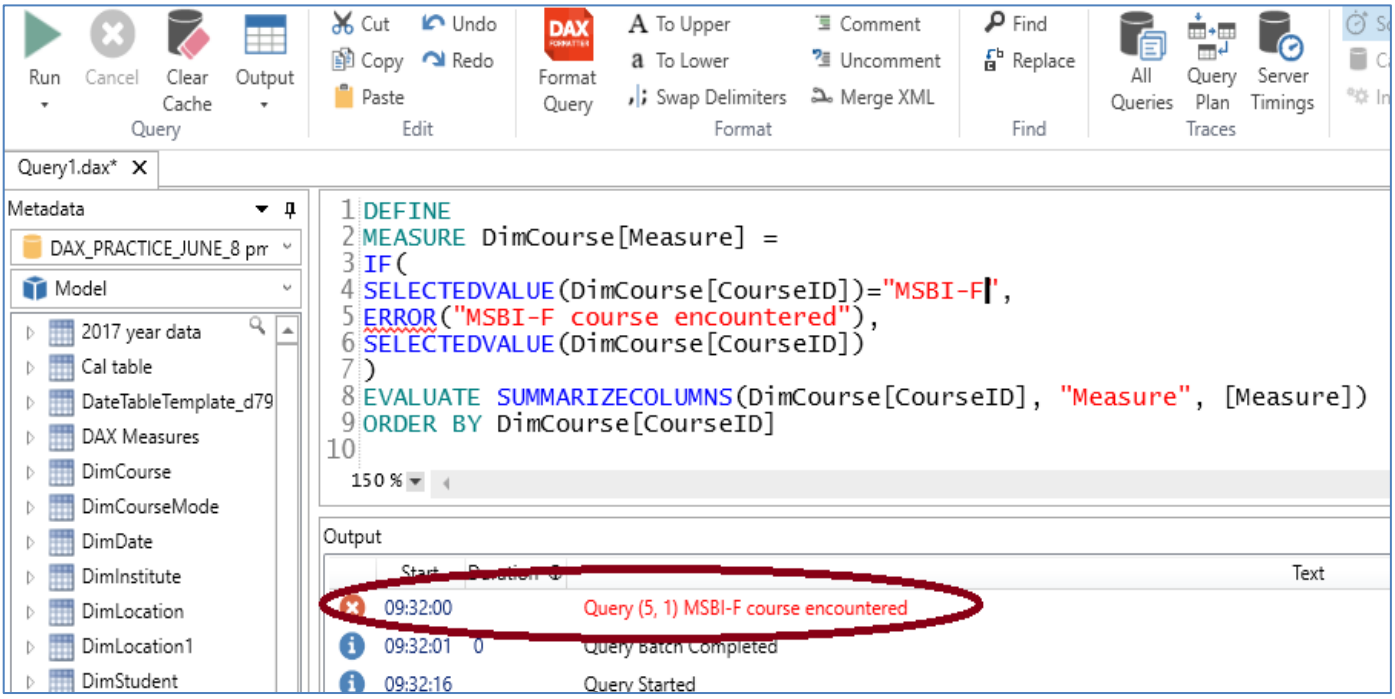
Because Magenta is not one of the product colors, the ERROR function is not executed.

Scenario:

Example: If the CourseID is MSBI_F in the list while displaying, then consider as ERROR and show manual ERROR message (user defined error message). This is basically suitable for Error Handling situations

Run the below query in DAX studio and see the result

```
DEFINE

MEASURE DimCourse[Measure] =

IF(

SELECTEDVALUE(DimCourse[CourseID])="MSBI-F",

ERROR("MSBI-F course encountered"),

SELECTEDVALUE(DimCourse[CourseID])

)
EVALUATE SUMMARIZECOLUMNS(DimCourse[CourseID], "Measure", [Measure])

ORDER BY DimCourse[CourseID]
```



Note: ERROR function is to consider user defined data as ERROR

Practice at Power BI Desktop

a) **Create a new measure like below and place on a card**

```
Selected Course= IF(
SELECTEDVALUE(DimCourse[CourseID])="MSBI-w",
ERROR("MSBI-w course encountered"),
SELECTEDVALUE(DimCourse[CourseID])
)
```

b) Take a slicer and use FactPayments coursed on the slicer

c) Browse slicer for the courses, for all courses, it will show you the selected value on the card. When we select MSBI-F, it will throw an error.

SET THEORY PRACTICE USING DAX FUNCTIONS

There are three set operation functions in DAX

a)Union b) Intersect c) Except

Note: There is no Union ALL

Create the below three tables using DATA TABLE FUNCTION.

New Table and write the below expression

```
EMP_1 = DataTable("EID", INTEGER,
"ENM", STRING,
"ELOC",STRING
,
{{1,"VINAY","HYD"},

{2,"MADHU","HYD"}
}

)
```

New Table and write the below expression

```
EMP_2 = DataTable("EID", INTEGER,
"ENM", STRING,
"ELOC",STRING
,
{{1,"VINAY","HYD"},
{3,"KISHORE","MUM"}

}

)
```

EXCEPT

Returns the rows of one table which do not appear in another table.

Syntax

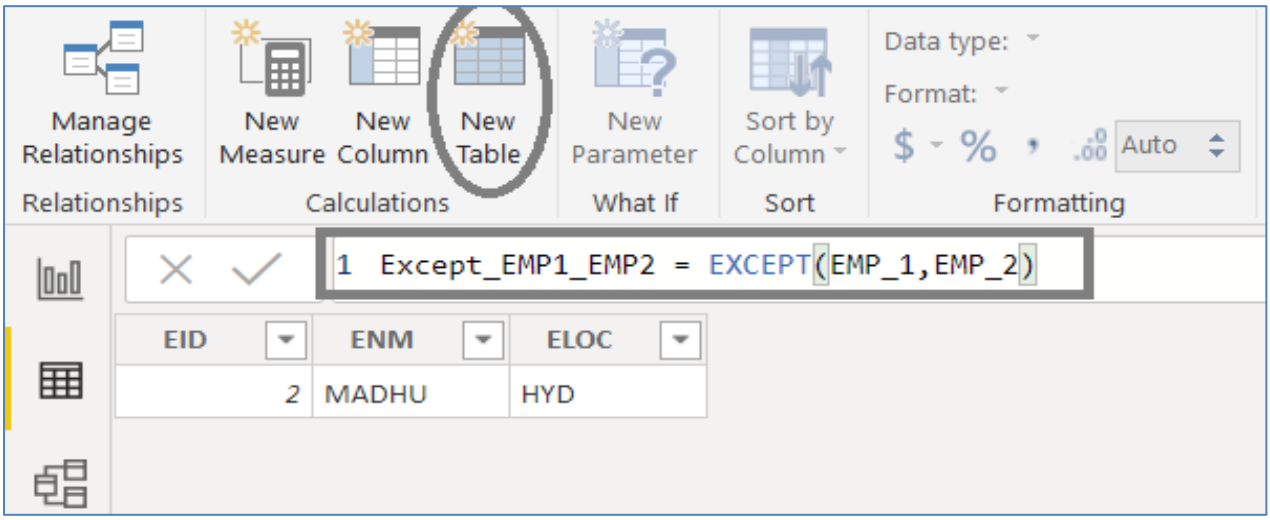
EXCEPT(<table_expression1>, <table_expression2>)

Example

Gives the courses which are new and not available in the regular courses table.

Table_Exclusive = except(DimCourses_new,DimCourse)

Create new table like below and see the result



GENERATESERIES

Returns a single column table containing the values of an arithmetic series, that is, a sequence of values in which each differs from the preceding by a constant quantity. The name of the column returned is Value.

Syntax

GENERATESERIES(<startValue>, <endValue>[, <incrementValue>])

Example 1

In SQL Server Identity mechanism or sequence mechanism does this functionality

Ex:

1st way:

IDENTITY (1,5)

2nd way:

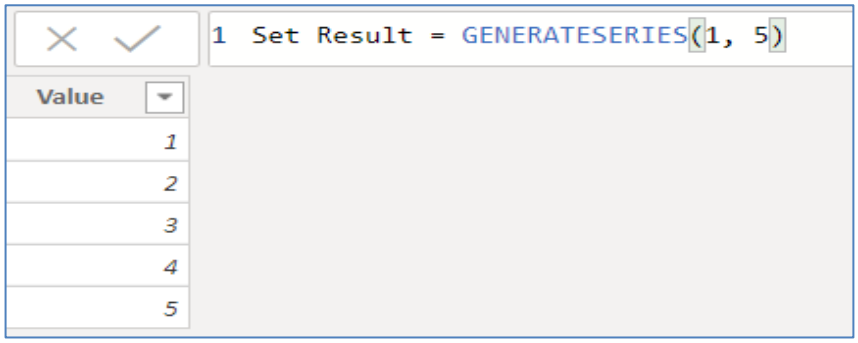
Create sequence seq start with 1 increment by 1 maxvalue 5;

A) Run the below query in DAX Studio

```
EVALUATE GENERATESERIES(1, 5)
```

B)Create New table and write like below to see the result

```
Sequence_Table=GENERATESERIES(1, 5)
```



Example 2

The following DAX query:

Create New table and write like below to see the result

```
EVALUATE GENERATESERIES(1.2, 2.4, 0.4)
```

Returns the following table with a single column:

VALUE	VALUE
1.2	**1.2**
1.6	**1.6**
2	**2**
2.4	**2.4**

Example 3

The following DAX query:

```
EVALUATE GENERATESERIES(CURRENCY(10), CURRENCY(12.4), CURRENCY(0.5))
```

Returns the following table with a single column:

VALUE
10
10.5
11
11.5
12

GROUP BY

The GROUPBY function is similar to the SUMMARIZE function. However, GROUPBY does not do an implicit CALCULATE for any extension columns that it adds. GROUPBY permits a new function, CURRENTGROUP(), to be used inside aggregation functions in the extension columns that it adds. GROUPBY attempts to reuse the data that has been grouped making it highly performant.

Syntax

```
GROUPBY (<table>, [<groupBy_columnName1>], [<name>, <expression>]... )
```

Example

Assume a data model has four tables: Sales, Customer, Product, Geography where Sales is on the "many" side of a relationship to each of the other three tables.

```
GROUPBY (
Sales,

Geography[Country],

Product[Category],

"Total Sales", SUMX( CURRENTGROUP(), Sales[Price] * Sales[Qty])
)
```

This will start with the Sales table, extended with all the columns from all the related

tables.Then it will build a result with three columns.

- The first column is each of the countries for which there is a sale.
- The second column is each product category for which there is a sale in that country.
- The third column is the sum of sales amount (as calculated from price*qty) for the selected country and product category.

Suppose we've built the previous result. We can use GROUPBY again, to find the maximum category sales figure within each country as shown here.

```
DEFINE
VAR SalesByCountryAndCategory =
GROUPBY (
Sales,
Geography[Country],
Product[Category],
"Total Sales", SUMX( CURRENTGROUP(), Sales[Price] * Sales[Qty])
)
Evaluate GROUPBY (
SalesByCountryAndCategory,
Geography[Country],
 "Max Sales", MAXX( CURRENTGROUP(), [Total Sales])
)
```

Scenario: Institute and Mode wise total discount fee

Create a new table with the below expression and see the result

SQL Code:

```
SELECT InstituteID,modeid, SUM(discount_fee) from FactPayments
GROUP BY InstituteID,ModeID
```

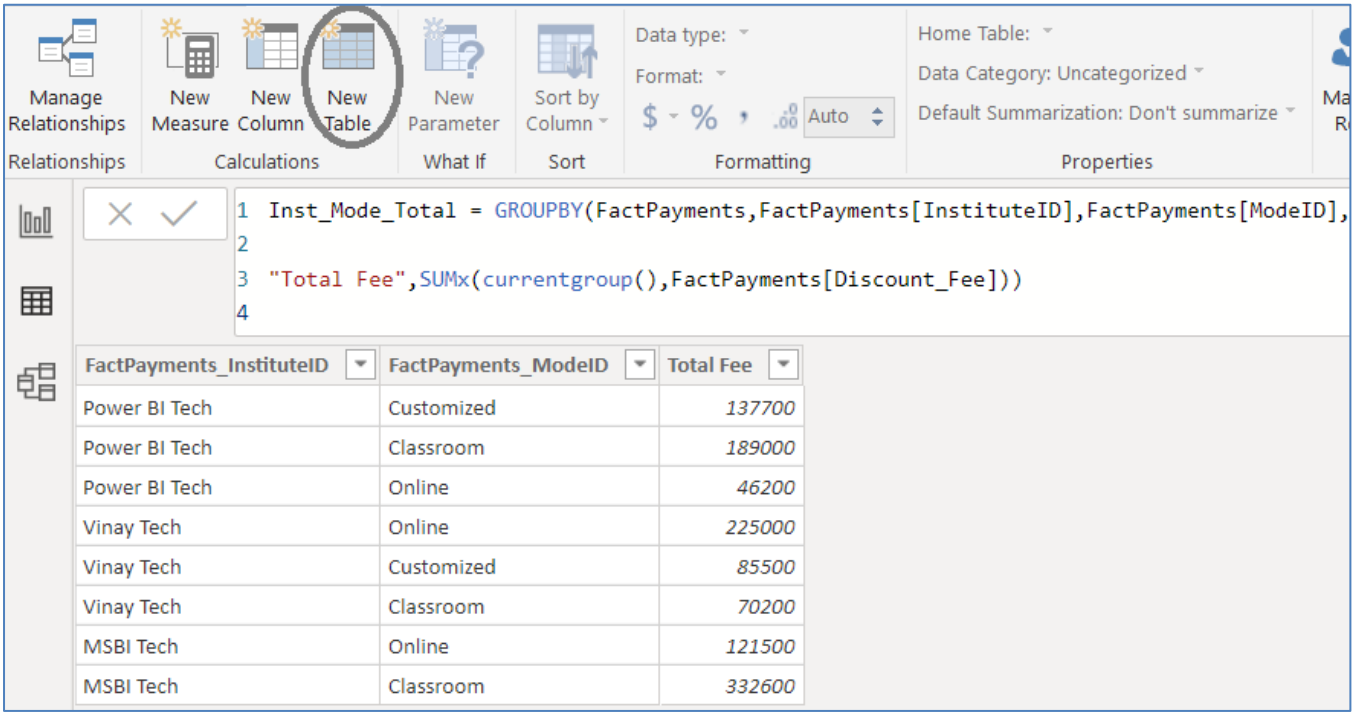
150 %

Results | Messages

	InstituteID	modeid	(No column name)
1	MSBI Tech	Classroom	422100
2	Power BI Tech	Classroom	321300
3	Vinay Tech	Classroom	57600
4	Power BI Tech	Customized	136800
5	Vinay Tech	Customized	84600
6	MSBI Tech	Online	360900
7	Power BI Tech	Online	171000
8	Vinay Tech	Online	297000

Inst_Mode_Total = GROUPBY(FactPayments,FactPayments[InstituteID],FactPayments[ModeID],

"Total Fee",SUMx(**currentgroup()**,FactPayments[Discount_Fee]))



FactPayments_InstituteID	FactPayments_ModeID	Total Fee
Power BI Tech	Customized	137700
Power BI Tech	Classroom	189000
Power BI Tech	Online	46200
Vinay Tech	Online	225000
Vinay Tech	Customized	85500
Vinay Tech	Classroom	70200
MSBI Tech	Online	121500
MSBI Tech	Classroom	332600

```
evaluate
(
GROUPBY(FactPayments,FactPayments[InstituteID],FactPayments[ModeID],

"Total Fee",SUMx(currentgroup(),FactPayments[Discount_Fee])))
```

Example on VAR and GroupBY

Keep location and course wise total value , from there display maximum value for each locationname.

The below is a table variable which contain multiple columns and values.

a)First Variable created

b) The variable result as input to Evaluate statement in the group by

General usage and syntax pattern of VAR

PBI Desktop Level

VAR <variablename>=expression

RETURN <use variable> with expression

DAX STUDIO Level

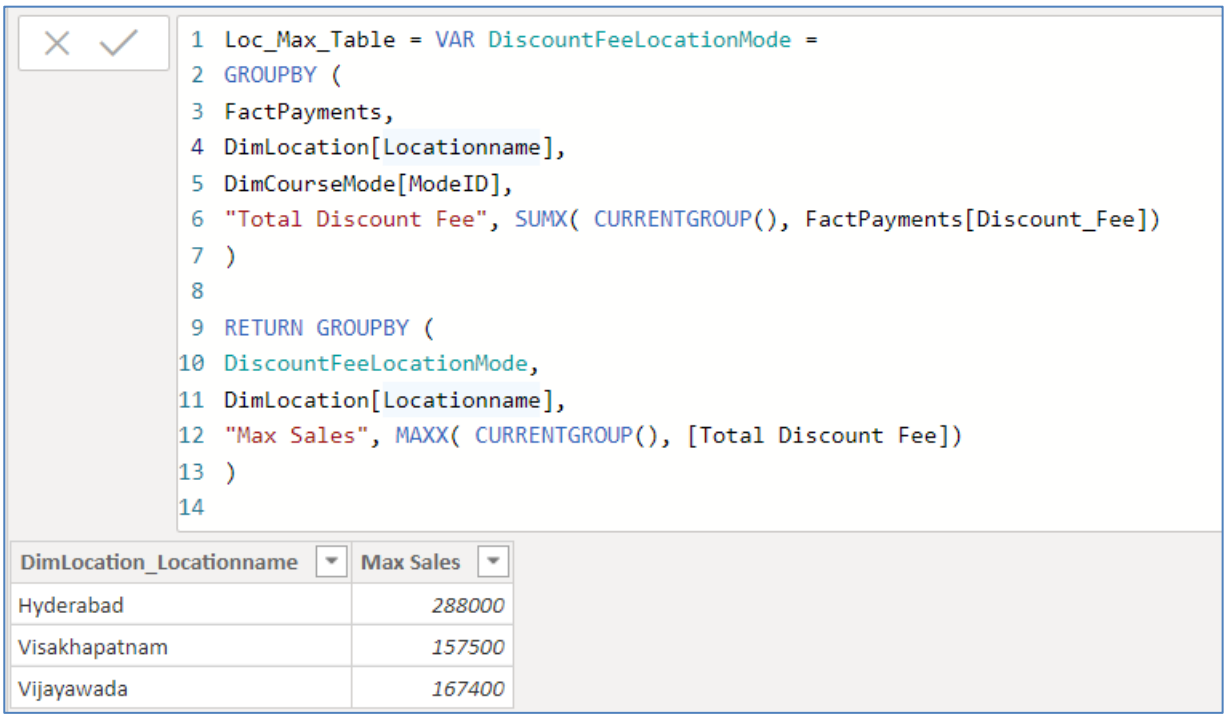
DEFINE <variablename>=expression

EVALUATE <use variable> with expression

PBI Desktop Level

Create new table

```
Loc_Max_Table = VAR DiscountFeeLocationMode =


GROUPBY(

FactPayments,

DimLocation[Locationname],

DimCourseMode[ModeID],

"Total Discount Fee", SUMX(CURRENTGROUP(), FactPayments[Discount_Fee])

)

RETURN GROUPBY(

DiscountFeeLocationMode,

DimLocation[Locationname],

"Max Sales", MAXX(CURRENTGROUP(), [Total Discount Fee])

)
```

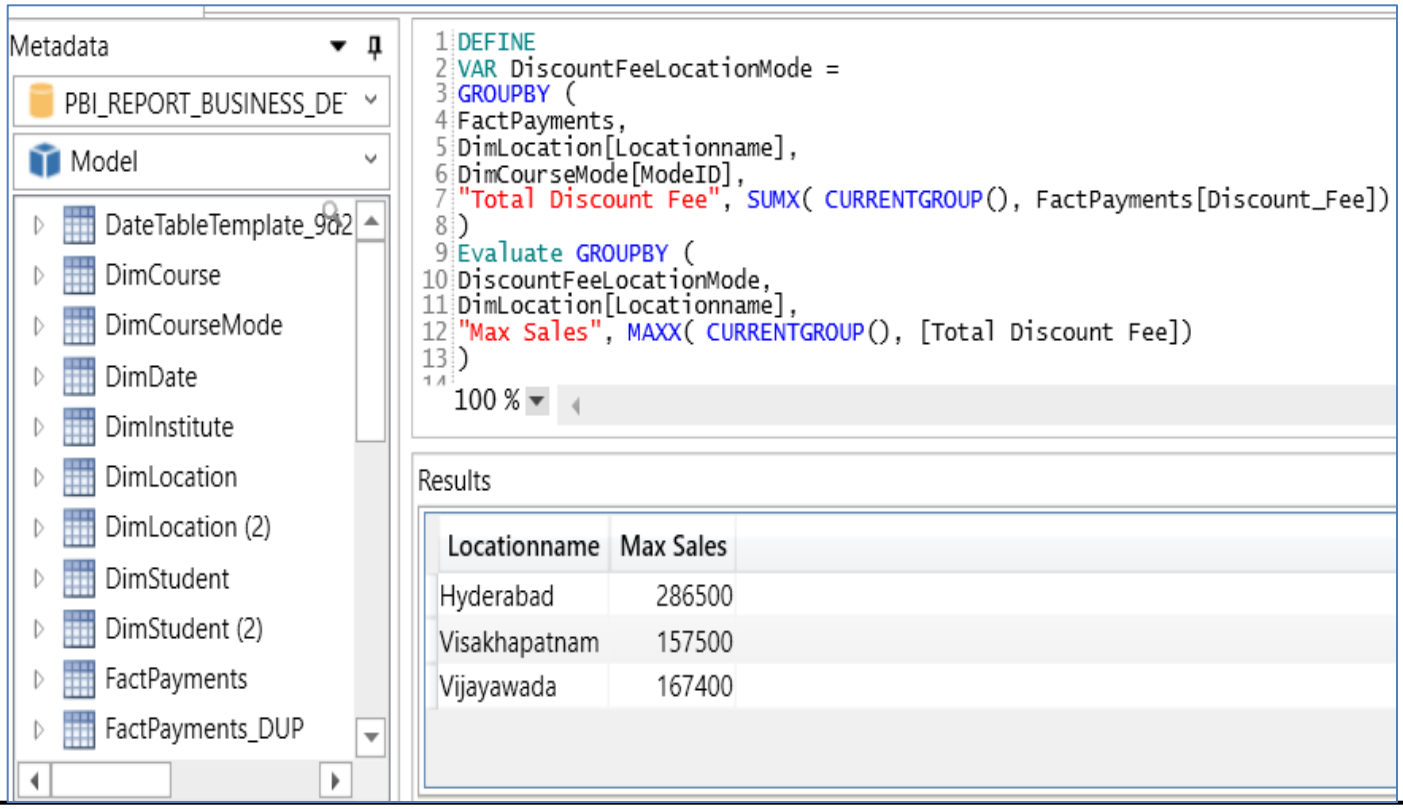


```
1  Loc_Max_Table = VAR DiscountFeeLocationMode =
2  GROUPBY (
3  FactPayments,
4  DimLocation[Locationname],
5  DimCourseMode[ModeID],
6  "Total Discount Fee", SUMX( CURRENTGROUP(), FactPayments[Discount_Fee])
7  )
8
9  RETURN GROUPBY (
10 DiscountFeeLocationMode,
11 DimLocation[Locationname],
12 "Max Sales", MAXX( CURRENTGROUP(), [Total Discount Fee])
13 )
14
```

DimLocation_Locationname	Max Sales
Hyderabad	288000
Visakhapatnam	157500
Vijayawada	167400

Run the below query at DAX Studo Level

```
DEFINE

VAR DiscountFeeLocationMode =

GROUPBY (

FactPayments,

DimLocation[Locationname],

DimCourseMode[ModeID],

"Total Discount Fee", SUMX( CURRENTGROUP(), FactPayments[Discount_Fee])

)

Evaluate GROUPBY (

DiscountFeeLocationMode,

DimLocation[Locationname],

"Max Sales", MAXX( CURRENTGROUP(), [Total Discount Fee])

)
```

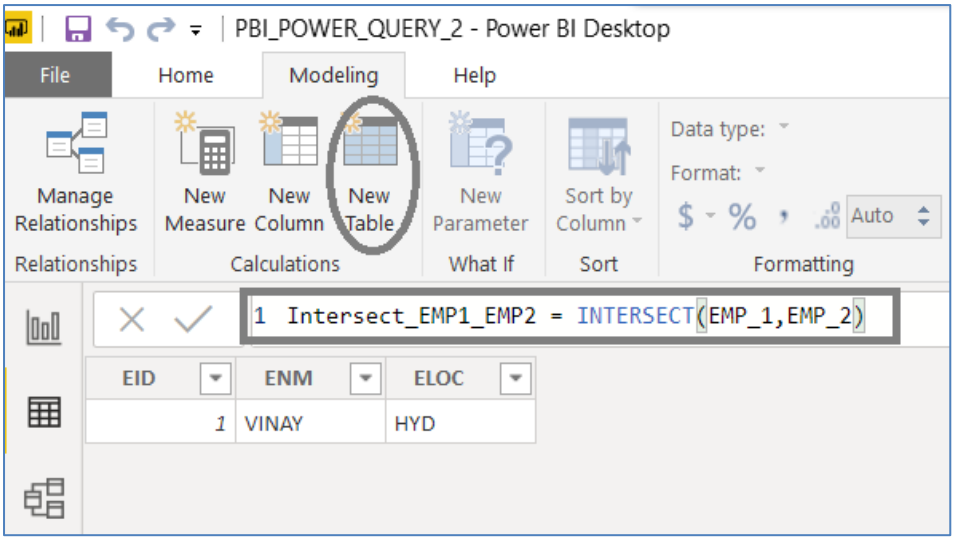
INTERSECT

Returns the row intersection of two tables, retaining duplicates.

Syntax : INTERSECT(<table_expression1>, <table_expression2>)

Create a new table (Modeling Menu->New Table) and place like below

Table_Common = Intersect(DimCourses_new,DimCourse)



ISEMPTY

Checks if a table is empty.

Syntax

ISEMPTY(<table_expression>)

Example

For the below table named 'Info':

COUNTRY	STATE	COUNTY	TOTAL
IND	JK	20	800
IND	MH	25	1000
IND	WB	10	900
USA	CA	5	500
USA	WA	10	900

```
EVALUATE

ROW("Any countries with count > 25?",
NOT(ISEMPTY(FILTER(Info, [Count]>25)))
```

Return value: **FALSE**

NATURALINNERJOIN

Performs an inner join of a table with another table. The tables are joined on common columns (by name) in the two tables. If the two tables have no common column names, an error is returned.

Syntax

NATURALINNERJOIN(<leftJoinTable>, <rightJoinTable>)

For example,

Products[ProductID], WebSales[ProductdID], StoreSales[ProductdID] with many-to-one relationships between WebSales and StoreSales and the Products table based on the ProductID column, WebSales and StoreSales tables are joined on [ProductID].

Strict comparison semantics are used during join. There is no type coercion; for example, 1 does not equal 1.0.

NATURALLEFTOUTERJOIN

Performs an inner join of a table with another table. The tables are joined on common columns (by name) in the two tables. If the two tables have no common column names, an error is returned.

Syntax

NATURALLEFTOUTERJOIN(<leftJoinTable>, <rightJoinTable>)

For example,

Products[ProductID], WebSales[ProductdID], StoreSales[ProductdID] with many-to-one relationships between WebSales and StoreSales and the Products table based on the ProductID column, WebSales and StoreSales tables are joined on [ProductID].

SUMMARIZECOLUMNS

Strict comparison semantics are used during join. There is no type coercion; for example, 1 does not equal 1.0.

Returns a summary table over a set of groups.

Syntax

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

Remarks

SUMMARIZECOLUMNS does not guarantee any sort order for the results.
A column cannot be specified more than once in the groupBy_columnName parameter. For example, the following formulas are invalid.

SUMMARIZECOLUMNS(Sales[StoreId], Sales[StoreId])

Filter context

Consider the following query:

SUMMARIZECOLUMNS ('Sales Territory'[Category], FILTER('Customer', 'Customer'

[First Name] = "Alicia"))

In this query, without a measure the groupBy columns do not contain any columns from the Filter expression (i.e.from Customer table). The filter is not applied to the groupBy columns. The Sales Territory and the Customer table may be indirectly related through the Reseller sales fact table. Since they're not directly related, the filter expression is a no-op and the groupBy columns are not impacted.

However, with this query:

SUMMARIZECOLUMNS ('Sales Territory'[Category], 'Customer' [Education],

FILTER('Customer', 'Customer'[First Name] = "Alicia"))

The groupBy columns contain a column which is impacted by the filter and that filter is

applied to the groupBy results.

Scenario: InstituteID and ModeID wise total discount fee

Create new table (Modeling Menu→ New Table), write the below expression, Preview the table data

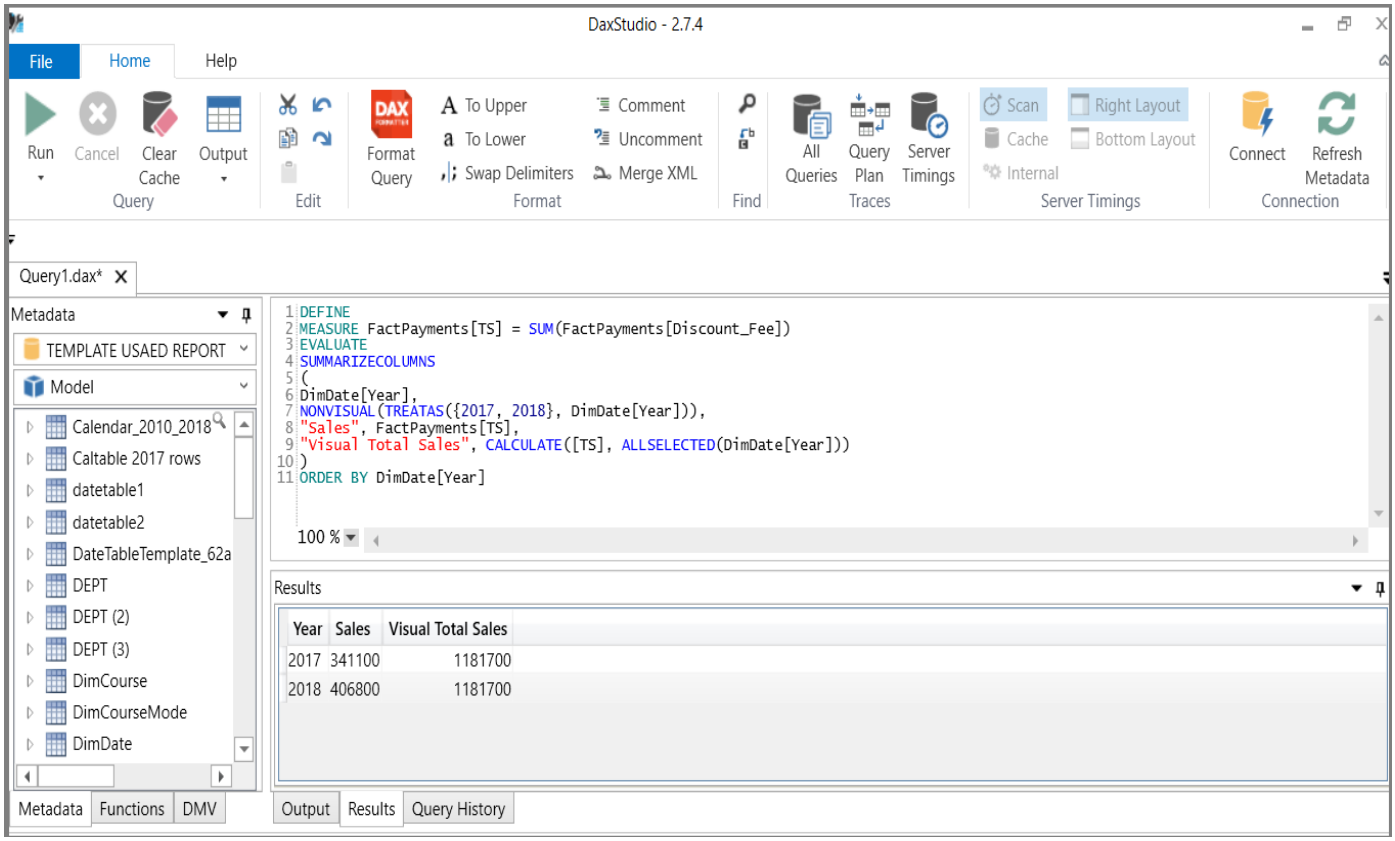
```
Inst_Mode_Total =
GROUPBY(FactPayments,FactPayments[InstituteID],FactPayments[ModeID],"Total Fee",SUMX(currentgroup(),FactPayments[Discount_Fee]))
```

Create new table (Modeling Menu→ New Table), write the below expression, Preview the table data

```
Inst_Mode_Total1 =
SUMMARIZECOLUMNS(FactPayments[InstituteID],FactPayments[ModeID],FactPayments,"Total",sum(FactPayments[Discount_Fee]))
```

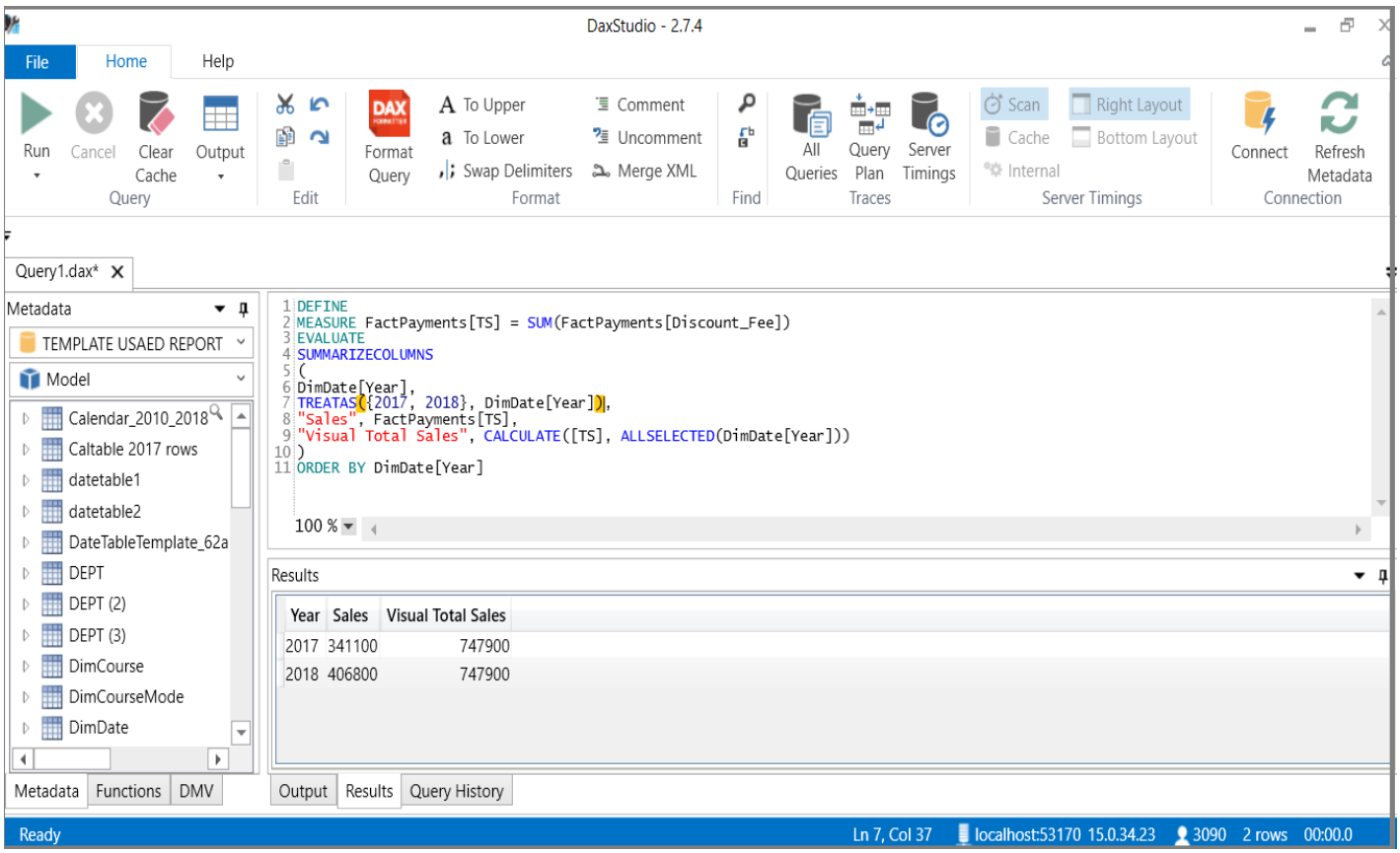
Run the below DAX Query in DAX Studio

```
DEFINE
MEASURE FactPayments[TS] = SUM(FactPayments[Discount_Fee])
EVALUATE
SUMMARIZECOLUMNS
(
DimDate[Year],
NONVISUAL(TREATAS({2017, 2018}, DimDate[Year])),
"Sales", FactPayments[TS],
"Visual Total Sales", CALCULATE([TS], ALLSELECTED(DimDate[Year]))
)
ORDER BY DimDate[Year]
```



Run the below DAX query at DAX Studio

```
DEFINE
MEASURE FactPayments[TS] = SUM(FactPayments[Discount_Fee])
EVALUATE
SUMMARIZECOLUMNS
(
DimDate[Year],
TREATAS({2017, 2018}, DimDate[Year]),
"Sales", FactPayments[TS],
"Visual Total Sales", CALCULATE([TS], ALLSELECTED(DimDate[Year]))
)
ORDER BY DimDate[Year]
```



IGNORE

The IGNORE() syntax can be used to modify the behavior of the

SUMMARIZECOLUMNS function by omitting specific expressions from the

BLANK/NULL evaluation. Rows for which all expressions not using IGNORE return BLANK/NULL will be excluded independent of whether the expressions which do use IGNORE evaluate to BLANK/NULL or not.

Syntax

IGNORE(<expression>)

With SUMMARIZECOLUMNS

SUMMARIZECOLUMNS(<groupBy_columnName>[, < groupBy_columnName >]…, [<filterTable>]…[, <name>, IGNORE(…)]…)

Example

SUMMARIZECOLUMNS(Sales[CustomerId], "Total Qty", IGNORE(SUM(Sales[Qty])), "BlankIfTotalQtyIsNot3", IF(SUM(Sales[Qty])=3, 3))

This rolls up the Sales[CustomerId] column, creating a subtotal for all customers in the given grouping. Without IGNORE, the result is:

CUSTOMERID	TOTALQTY	BLANKIFTOTALQTYISNOT3
A	5	
B	3	3

CUSTOMERID	TOTALQTY	BLANKIFTOTALQTYISNOT3
C	3	3

With IGNORE

CUSTOMERID	TOTALQTY	BLANKIFTOTALQTYISNOT3
B	3	3
C	3	3

All expression ignored

SUMMARIZECOLUMNS(Sales[CustomerId], "Blank", IGNORE(Blank()),
"BlankIfTotalQtyIsNot5", IGNORE(IF(SUM(Sales[Qty])=5, 5)))

Even though both expressions return blank for some rows, they're included since there are no non-ignored expressions which return blank.

CUSTOMERID	TOTALQTY	BLANKIFTOTALQTYISNOT3
A		5
B		
C		

Create new table (Modeling Menu→ New Table), write the below expression, Preview the table data

```
Student_Table =
SUMMARIZECOLUMNS(FactPayments[StudentID],"blank",ignore(BLANK()),"nonblank",
if(IGNORE([Total DF1]=0),[Total DF1]))
```

NONVISUAL

Marks a value filter in SUMMARIZECOLUMNS function as not affecting measure values, but only applying to group-by columns.

Syntax

NONVISUAL(<expression>)

Return value

A table of values.

Example

```
DEFINE
MEASURE FactInternetSales[Sales] = SUM(FactInternetSales[Sales Amount])
EVALUATE
SUMMARIZECOLUMNS
(
 DimDate[CalendarYear],
 NONVISUAL(TREATAS({2007, 2008}, DimDate[CalendarYear])),
 "Sales", [Sales],
 "Visual Total Sales", CALCULATE([Sales], ALLSELECTED(DimDate[CalendarYear]))
)
ORDER BY [CalendarYear]
```


Result

Returns the result where [Visual Total Sales] is the total across all years:

DIMDATE[CALENDARYEAR]	[SALES]	[VISUAL TOTAL SALES]
2007	9,791,060.30	29,358,677.22
2008	9,770,899.74	29,358,677.22

In contrast, the same query without the NONVISUAL function:

Run the below DAX query at DAX Studio

```
DEFINE
MEASURE FactInternetSales[Sales] = SUM(FactInternetSales[Sales Amount])
EVALUATE
SUMMARIZECOLUMNS
(
 DimDate[CalendarYear],
 TREATAS({2007, 2008}, DimDate[CalendarYear]),
 "Sales", [Sales],
 "Visual Total Sales", CALCULATE([Sales], ALLSELECTED(DimDate[CalendarYear]))
)
ORDER BY [CalendarYear]
```

Result

Returns the result where [Visual Total Sales] is the total across the two selected years:

DIMDATE[CALENDARYEAR]	[SALES]	[VISUAL TOTAL SALES]
2007	9,791,060.30	19,561,960.04
2008	9,770,899.74	19,561,960.04

ROLLUPADDISSUBTOTAL

The addition of the ROLLUPADDISSUBTOTAL() syntax modifies the behavior of the SUMMARIZECOUMNS function by adding roll-up/subtotal rows to the result based on the columns.

Syntax

```
ROLLUPADDISSUBTOTAL ( [<filter> ..., ] <groupBy_columnName>,
<isSubtotal_columnName>[, <filter> ...][, <groupBy_columnName >,
<isSubtotal_columnName>[, <filter> ...]...] )
```

Example

Single subtotal

```
DEFINE
VAR vCategoryFilter =
  TREATAS({"Accessories", "Clothing"}, Product[Category])
VAR vSubcategoryFilter =
  TREATAS({"Bike Racks", "Mountain Bikes"}, Product[Subcategory])
EVALUATE
  SUMMARIZECOLUMNS
  (
    ROLLUPADDISSUBTOTAL
    (
      Product[Category], "IsCategorySubtotal",
      vCategoryFilter, Product[Subcategory],
      "IsSubcategorySubtotal", vSubcategoryFilter
    ),
    "Total Qty", SUM(Sales[Qty])
  )
  ORDER BY
  [IsCategorySubtotal] DESC, [Category],
  [IsSubcategorySubtotal] DESC, [Subcategory]
```


ROLLUPGROUP

Like with the SUMMARIZE function, ROLLUPGROUP can be used together with ROLLUPADDISSUBTOTAL to specify which summary groups/granularities (subtotals) to include (reducing the number of subtotal rows returned).

Syntax

ROLLUPGROUP(<groupBy_columnName>, <groupBy_columnName>)

With ROLLUPADDISSUBTOTAL

ROLLUPADDISSUBTOTAL(ROLLUPGROUP(...), isSubtotal_columnName[, <groupBy_columnName>...])

Example

Multiple subtotals

```
SUMMARIZECOLUMNS( ROLLUPADDISSUBTOTAL( Sales[CustomerId],
"IsCustomerSubtotal" ),
ROLLUPADDISSUBTOTAL(ROLLUPGROUP(Regions[City], Regions[State]),
“IsCityStateSubtotal”),"Total Qty", SUM( Sales[Qty] ) )
```

Still grouped by City and State, but rolled together when reporting a subtotal.

EVALUATE

Returns a table of one or more columns.

Syntax

{ <scalarExpr1>, <scalarExpr2>, ... }

{ (<scalarExpr1>, <scalarExpr2>, ...), (<scalarExpr1>, <scalarExpr2>, ...), ... }

Example 1

The following DAX queries:

```
EVALUATE { 1, 2, 3 }
```

and

```
EVALUATE { (1), (2), (3) }
```

Return the following table of a single column:

Result:

Value

1

2

3

Example 2

The following DAX query:

```
EVALUATE
 {
  (1.5, DATE(2017, 1, 1), CURRENCY(199.99), "A"),

  (2.5, DATE(2017, 1, 2), CURRENCY(249.99), "B"),

  (3.5, DATE(2017, 1, 3), CURRENCY(299.99), "C")
 }
```

[VALUE1]	[VALUE2]	[VALUE3]	[VALUE4]
1.5	1/1/2017	199.99	A
Row2	1/2/2017	249.99	B
Row3	1/3/2017	299.99	C

Example 3

The following DAX query

```
EVALUATE { 1, DATE(2017, 1, 1), TRUE, "A" }
```

Returns the following table of a single column of String data type:

[VALUE]
1
1/1/2017
TRUE
A

TREATAS

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

Syntax

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

Examples

In the following example, the model contains two unrelated product tables. If a user applies a filter to DimProduct1[ProductCategory] selecting Bikes, Seats, Tires, the same filter, Bikes, Seats, Tires is applied to DimProduct2[ProductCategory].

```
CALCULATE(
SUM(Sales[Amount]),
TREATAS(VALUES(DimProduct1[ProductCategory]), DimProduct2[ProductCategory])
)
```


Scenario: Finding the total from DimCourse and DimCourses_new based on the selection from DimCourse query [based on the selection from one table getting result from both]

1.Create another table [DimCourses_New] with few similar CourseID values .

2.Connect this table to the Fact Table if possible.

3.Take slicer and use CourseID

4. Create a measure like below and use in a card

```
Multitblsum = calculate(sum(FactPayments[Discount_Fee]),
treatas(values(DimCourse[CourseID]),DimCourses_new[CourseID]))
```

5.Slice the values in the slicer, then the result is from both the tables.

Ex: If you select MSBI-F from the slicer, then the total from DimCourse and DimCourse_new for MSBI-F will be displayed.

UNION

Creates a union (join) table from a pair of tables.

Syntax

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

Example

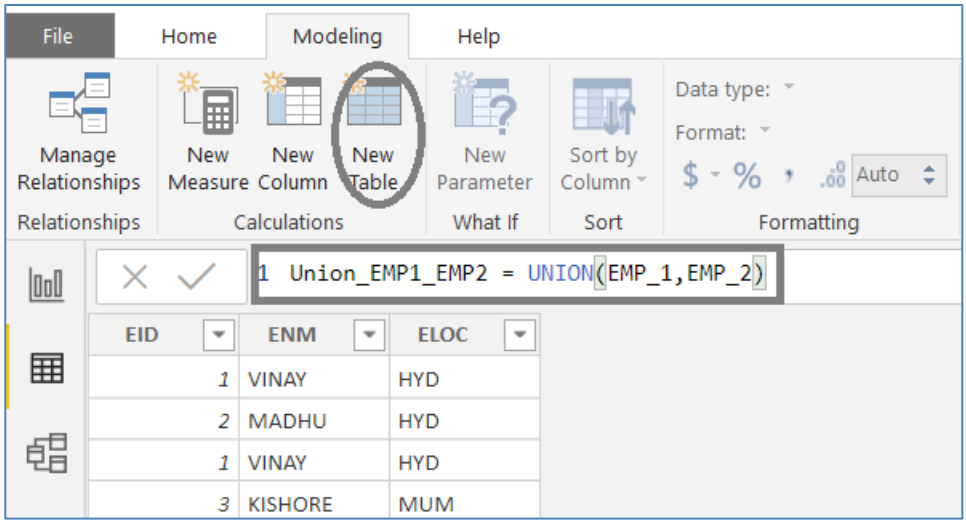
The following expression creates a union by combining the USAInventory table and the INDInventory table into a single table:

UNION(UsaInventory, IndInventory)

Create new table (Modeling Menu→ New Table), write the below expression,

Preview the table data

Table_Merge = union(DimCourses_new,DimCourse)



VAR

Stores the result of an expression as a named variable, 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.

Syntax

VAR <name> = <expression>

Example: Refer to the next example

Note: VAR is nothing but DEFINE in DAX Query

Class room practice [year over year growth using variables in DAX]:

General implementation:

a) **Create current year value**
b) **Create last year value**
c) **Growth percentage=current-last/ last * 100**

The above three steps in single step in the below example.
Create a measure with the below expression and place on a card
I am creating two variables and returning value by using the variables.

```
YoY Growth Percent Measure =

var Currentyeardiscountfee = sum(FactPayments[Discount_Fee])

var Lastyeardiscountfee=CALCULATE(SUM(FactPayments[Discount_Fee]),

SAMEPERIODLASTYEAR(DimDate[Date]))

return if(Currentyeardiscountfee,

DIVIDE(Currentyeardiscountfee - lastyeardiscountfee, lastyeardiscountfee) * 100)
```

SUM

To calculate a percentage of year-over-year growth without using a variable, you could create three separate measures.

This first measure calculates Sum of Sales Amount:

Sum of SalesAmount = SUM(SalesTable[SalesAmount])

A second measure calculates the sales amount for the previous year

SalesAmount PreviousYear=CALCULATE([Sum of SalesAmount],

SAMEPERIODLASTYEAR(Calendar[Date]))

You can then create a third measure that combines the other two measures to calculate a growth percentage. Notice the Sum of SalesAmount measure is used in two places; first to determine if there is a sale, then again to calculate a percentage.

Sum of SalesAmount YoY%:=IF([Sum of SalesAmount] ,
DIVIDE(([Sum of SalesAmount] – [SalesAmount PreviousYear]), [Sum of PreviousYear]))

By using a variable, you can create a single measure that calculates the same result:

Create a measure and place on card

YoY% = var Sales = SUM(SalesTable[SalesAmount])
var SalesLastYear=CALCULATE(Sales, SAMEPERIODLASTYEAR('Calendar'[Date]))
return if(Sales, DIVIDE(Sales – SalesLastYear, Sales))

By using a variable, you can get the same outcome, but in a more readable way. In addition, the result of the expression is stored in the variable upon declaration. It doesn't have to be recalculated each time it is used, as it would without using a variable. This can improve the measure's performance.

Class room practice [year over year growth using variables in DAX]:

Create a measure and place on a card

YoY varmech = var tod = sum(FactPayments[Discount_Fee])

var tod

=CALCULATE(SUM(FactPayments[Discount_Fee]),SAMEPERIODLASTYEAR(DimDate[Date]))

return if(tod,divide(tod-lod,lod))

Note: tod (total discount fee), lod(last year discount fee)

STATISTICAL FUNCTIONS

These work on multiple rows and provide a result. Useful for business statistical Analysis [you want to find out tops, bottoms, avgs, count, max, min, ranks, samples etc..]

STATISTICAL FUNCTION	
ADDCOLUMNS	Adding columns to the existing columns
AVERAGE	Returns the average of all numbers in a column
AVERAGEX	Calculates the average (arithmetic mean) of a set of expressions evaluated over a table.
AVERAGEA	Returns the average (arithmetic mean) of the values in a column. Handles text and non-numeric values.
COUNT	The COUNT function counts the number of cells in a column that contain numbers.
COUNTA	The COUNTA function counts the number of cells in a column that are not empty. It counts not just rows that contain numeric values, but also rows that contain nonblank values, including text, dates, and logical values.
COUNTAX	The COUNTAX function counts nonblank results when evaluating the result of an expression over a table. That is, it works just like the COUNTA function, but is used to iterate through the rows in a table and count rows where the specified expressions results in a nonblank result.
COUNTBLANK	the number of blank cells Counts in a column.
COUNTROWS	The COUNTROWS function counts the number of rows in the specified table, or in a table defined by an expression.
COUNTX	Counts the number of rows that contain a number or an expression that evaluates to a number, when evaluating an expression over a table.
CROSSJOIN	Returns a table that contains the Cartesian product of all rows from all tables in the arguments. The columns in the new table are all the columns in all the argument tables.
DISTINCTCOUNT	The DISTINCTCOUNT function counts the number of distinct values in a column.
GENERATE	Returns a table with the Cartesian product between each row in table1 and the table that results from evaluating table2 in the context of the current row from table1.
GENERATEALL	Returns a table with the Cartesian product between each row in table1 and the table that results from evaluating table2 in the context of the current row from table1.
GEOMEAN	To return the geometric mean of an expression evaluated for each row in a table
MAX	Returns the largest numeric value in a column, or between two scalar expressions.
MAXA	Returns the largest value in a column. Logical values and blanks are counted.
MAXX	Evaluates an expression for each row of a table and returns the largest numeric value.

MEDIAN	Returns the median of numbers in a column.
MEDIANX	Returns the median number of an expression evaluated for each row in a table
MIN	Returns the smallest numeric value in a column, or between two scalar expressions. Ignores logical values and text.
MINA	Returns the smallest value in a column, including any logical values and numbers represented as text.
MINX	Returns the smallest numeric value that results from evaluating an expression for each row of a table
PERMUT	Returns the number of permutations for a given number of objects that can be selected from number objects. A permutation is any set or subset of objects or events where internal order is significant. Permutations are different from combinations, for which the internal order is not significant. Use this function for lottery-style probability calculations.
RANK.EQ	Returns the ranking of a number in a list of numbers.
RANKX	Returns the ranking of a number in a list of numbers for each row in the *table* argument.
ROW	Returns a table with a single row containing values that result from the expressions given to each column.
SAMPLE	Returns a sample of N rows from the specified table
SELECTCOLUMNS	Adds calculated columns to the given table or table expression.
SUMMARIZE	Returns a summary table for the requested totals over a set of groups.
TOPN	Returns the top N rows of the specified table.

Statistical / Aggregate Function : Complete Column Wise Operation

Statistical / Aggregate Function + X: Expression based statistical / aggregate operation

Statistical / Aggregate Function + A:

Even column having Nulls, Logical Values, Text, Date etc… few supporting

Run the below queries in DAX Studio:

```
EVALUATE
(

ROW(

"DF_Sum", sum(FactPayments[Discount_Fee]),

"DF_Avg", AVERAGE(FactPayments[Discount_Fee]),

"DF_Avg_A", AVERAGEA(FactPayments[Discount_Fee]),

"DF_Count",COUNT(FactPayments[Discount_Fee]),

"DF_Count_A",COUNTA(FactPayments[Discount_Fee]),

"DF_Count_BLANK",COUNTBLANK(FactPayments[Discount_Fee]),

"DF_DistinctCount", DISTINCTCOUNT(FactPayments[Discount_Fee]),

"DF_MaxVal", max(FactPayments[Discount_Fee]),

"DF_MaxVal_A", maxA(FactPayments[Discount_Fee]),

"DF_Minval",min(FactPayments[Discount_Fee]),

"DF_Minval_A",minA(FactPayments[Discount_Fee]),

"DF_Median", MEDIAN(FactPayments[Discount_Fee])
)
)
```



```
/*

Create a table called test(eid integer, salary numeric, salary_text varchar(30)),
keep salary as numeric column, salary_text as
string column with NULLS, BLANKS and Numeric VALUES


*/

evaluate
(
row
(
   "count_sal",count(TEST[SALARY]),

   "count_sal_blank",COUNTBLANK(TEST[SALARY]),

   "count_sal_Text", count(TEST[SALARY_TEXT]),

   "count_sal_Text_1", counta(TEST[SALARY_TEXT]),

   "Max value", max(TEST[SALARY_TEXT]),

   "Max value_1", maxa(TEST[SALARY_TEXT]),

   "Min value", min(TEST[SALARY_TEXT]),

   "Min value_1", mina(TEST[SALARY_TEXT])
)
)
```

```
/*
sample 5 records by keeping factpayments discount fee in ascending order

1--indicate asc, 0--indicate desc

*/

EVALUATE
(
sample(5,FactPayments,FactPayments[Discount_Fee],0)
)

/*
Creating a table with courseid wise sum, avg of incomes
SQL Statement:
Select sum(income), avg(income) from tablename
group by courseid
*/

EVALUATE
(
SUMMARIZE
(
FactPayments,FactPayments[CourseID],"total",sum(FactPayments[Discount_Fee]),
"avg", AVERAGE(FactPayments[Discount_Fee])
)
)
```

```
/* Creating a table with top 2 courseids based on total income ascending order*/

EVALUATE
(
TOPN(
2,
SUMMARIZE
(
FactPayments,FactPayments[CourseID],"total",sum(FactPayments[Discount_Fee]),
"avg", AVERAGE(FactPayments[Discount_Fee])
),

[total],DESC
)

)

/* Creating a table with top 2 courseids based on total income descending order*/
```



```
EVALUATE
(
TOPN(
2,
SUMMARIZE
(
FactPayments,FactPayments[CourseID],"total",sum(FactPayments[Discount_Fee]),
"avg", AVERAGE(FactPayments[Discount_Fee])
),[total],desc)
)
--Create two tables with data emp1 (eid,enm,did), dept1 (did,dnm)
```

```
/* CROSSJOIN of two tables. Here emp table each record mapped to DEPT table and gets data */

evaluate
(emp)

EVALUATE

(
CROSSJOIN(EMP,DEPT)
)


/* For each location all modes total income */

EVALUATE
(
CROSSJOIN
(
SUMMARIZE(DimLocation,DimLocation[LocID]),

SUMMARIZE(DimCourseMode,DimCourseMode[ModeID],"Total",sumx(RELATEDTABLE(FactPayments),
FactPayments[Discount_Fee])

)
)
)
```

```
/* Explanation:

1) system create two results first

    a)List of locations one result
    b) Mode wise total another result

2) Perform simple cross join between both

Note: Because of this reason we got non business locations also values



*/


/* For each location mode wise total income */
EVALUATE
(
GENERATE --works like correlated sub query
(
SUMMARIZE(DimLocation,DimLocation[LocID]), --outer query

SUMMARIZE(DimCourseMode,DimCourseMode[ModeID],"Total",sumx(RELATEDTABLE(FactPayments),
FactPayments[Discount_Fee])

)
)
)
/* Explanation:
system pass each location to second resultset and then gives result
Note: Because of this reason we got business locations only values
Simply Cross join precaulates the result sets and perform cross join, whereas each row passes
from first result set to second result set and then perform cross join in generate
*/
```

```
--Syntax: Create a table 'EMP'
Evaluate
(
DATATABLE("EID",INTEGER,"ENM",STRING,{{1,"X"},{2,"Y"}})
)

--Syntax: Create a table 'DEPT'
Evaluate
(
DATATABLE("EID",INTEGER,"ENM",STRING,{{1,"X"},{2,"Y"}})
)

Evaluate
(
crossjoin(emp1,dept1)
)
```



```
/* Finding rank of each duration value keeping in descending order
Rank:
      Each value it gives a rank starting with 1
      Same value it gives same rank
      There are gaps between ranks
 */

evaluate
(
addcolumns(DimCourse,"rank", RANKX(DimCourse,DimCourse[Duration]))
)
order by [rank] asc
```

```
/*

Rank.EQ Practice

a)Add Gtax file to the dataset [the file is in the lab back up folder / dax folder]

b)Create a Fact_Test table simply by taking Tax amount column in the fact table

--Refer to class room PDF [single pdf]s

*/

EVALUATE
(
ADDCOLUMNS(

Fact_Test,
"Local Rank 1", rankx(Fact_Test,Fact_Test[Tax amount]),
"Global Rank 1", rank.eq(Fact_Test[Tax amount],GTAX[ Gtax],desc)

)
)
```

```
--Sum, Count, Avg, Min, Max etc... calculations on complete column wise.
--All rows participated

Evaluate
(

row(
"Sum of Discount Fee",sum(FactPayments[Discount_Fee]),
"Average of Discount Fee", Average(FactPayments[Discount_Fee]),
"Count of Students", count(FactPayments[StudentID]),
"Max value", max(FactPayments[Discount_Fee]),
"Min value", min(FactPayments[Discount_Fee])
)
)
----Sumx, Countx, Avgx, Minx, Maxx etc... calculations on specified column values
--Expression [X-indicate expression] matched column values participated

Evaluate
(

row(
"Sum of Discount Fee for Online mode",
sumx(Filter(FactPayments,FactPayments[ModeID]="Online"),FactPayments[Discount_Fee]),
"Average of Discount Fee for Online mode",
Averagex(Filter(FactPayments,FactPayments[ModeID]="Online"),FactPayments[Discount_Fee]),
"Count of Students for online",
countx(Filter(FactPayments,FactPayments[ModeID]="Online"),FactPayments[Discount_Fee]),
"Max value in Online mode",
maxx(Filter(FactPayments,FactPayments[ModeID]="Online"),FactPayments[Discount_Fee]),
"Min value in Online mode",
minx(Filter(FactPayments,FactPayments[ModeID]="Online"),FactPayments[Discount_Fee])
)
)
```

```
--SUMX Typical Situations Usage
--I need for each course the total discount fee generated

Evaluate
(
ADDCOLUMNS(DimCourse,
"Each course Total",sum(FactPayments[Discount_Fee]),--this sum is total discount fee
"Each course total 1", SUMX(RELATEDTABLE(FactPayments),FactPayments[Discount_Fee])
--Related table with join course table with facttable and get all matched rows
--as a table
)
)

/*
Related and Related table uses existing data model, join the tables when
we them.No need to mention join condition and columns
*/
```

ADDCOLUMNS

Adds calculated columns to the given table or table expression.

Syntax

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

Example 1:

Class room practice: Refer to manual date table creation in the Date Time Functions

1.Create a new table using [Modeling Menu→ New Table]

Date_table_2019 =Calendar(date(2019,01,01),date(2019,12,31))

2.Create another table using Addcolumns function [Modeling Menu→New table]

Date_Full-table_2019=**ADDCOLUMNS**(Date_table_2019, "YEAR", YEAR(Date_table_2019 [Date]),"MONTH",MONTH(Date_table_2019 [Date]),"DAY",DAY(Date_table_2019 [Date]))

3. Preview result

Example 2:

Take New Table [Modeling Menu→New Table] and write the below expression

Coursesmode_new = ADDCOLUMNS(DimCourseMode,"count_co", count('Caltable 2017 rows'[CourseID]))

AVERAGE

Returns the average (arithmetic mean) of all the **numbers in a column.**

Syntax

AVERAGE(<column>)

Example

Create a new measure and place on a card

Avg Discount Fee=AVERAGE(FactPayments[Discount_Fee])

Related functions

The AVERAGEX function can take as its argument an expression that is evaluated for each row in a table. This enables you to perform calculations and then take the average of the calculated values.

The AVERAGEA function takes a column as its argument, but otherwise is like the Excel function of the same name. By using the AVERAGEA function, you can calculate a mean on a column that contains empty values.

AVERAGEA

Returns the average (arithmetic mean) of the values in a column. Handles text and non-numeric values.

Syntax

AVERAGEA(<column>)

=AVERAGEA([Amount])

Create a new measure and place on a card

Avg Discount Fee=AVERAGEA(FactPayments[Discount_Fee])

AVERAGEX

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

Syntax

AVERAGEX(<table>,<expression>)

Example

Add a new column in the factpayments and see the result

=AVERAGEX(FactPayments, FactPayments[Discount_Fee]+ FactPayments[TaxAmt])

COUNT

The COUNT function counts the **number of cells in a column that contain numbers.**

Syntax

COUNT(<column>)

Example

Create a new measure and place on a card

Count of students=COUNT(FactPayments[StudentID])

To count logical values or text, use the COUNTA or COUNTAX functions.

COUNTA

The COUNTA function counts the number of cells in a column that are not empty. It counts not just rows that contain numeric values, but also rows that contain nonblank values, including text, dates, and logical values.

Syntax

COUNTA(<column>)

Example

Create a new measure and place on a card

Count of Phone Numbers=COUNTA(DimStudent[Phone])

COUNTAX

The COUNTAX function counts nonblank results when evaluating the result of an expression over a table. That is, it works just like the COUNTA function, but is used to iterate through the rows in a table and count rows where the specified expressions results in a nonblank result.

Syntax

COUNTAX(<table>,<expression>)

Example

Create a new measure and place on a card

Count of hyd location students
=COUNTAX(FILTER('FactPayments',[LocID]="HYD"),[StudentID])

COUNTBLANK

Counts the number of blank cells in a column.

Syntax

COUNTBLANK(<column>)

Example

Create a new measure and place on a card

Count Blank Values
=COUNTBLANK(DimStudent[Fname])

To count logical values or text, use the COUNTA or COUNTAX functions.

COUNTROWS

The COUNTROWS function counts the number of rows in the specified table, or in a table defined by an expression.

Syntax

COUNTROWS(<table>)

Example

Create a new measure and place on a card

Count of rows
=COUNTROWS('FactPayments')

Other Example

The following example demonstrates how to use COUNTROWS with a row context. In this scenario, there are two sets of data that are related by order number. The table Reseller contains one row for each reseller; the table ResellerSales contains multiple rows for each order, each row containing one order for a particular reseller. The tables are connected by a relationship on the column, ResellerKey.

The formula gets the value of ResellerKey and then counts the number of rows in the related table that have the same reseller ID. The result is output in the column,

CalculatedColumn1.

```
=COUNTROWS(RELATEDTABLE(ResellerSales))
```

The following table shows a portion of the expected results:

RESELLERKEY	CALCULATEDCOLUMN1
1	73
2	70
3	394

COUNTX

Counts the number of rows that contain a number or an expression that evaluates to a number, when evaluating an expression over a table.

Syntax

```
COUNTX(<table>,<expression>)
```

Example

Create a new measure and place on a card

```
Count of students having names=COUNTX(FactPayments,DimStudent[Fname])
```

Example

The following formula illustrates how to pass a filtered table to COUNTX for the first argument. The formula uses a filter expression to get only the rows in the Product table that meet the condition, ProductSubCategory = "Caps", and then counts the rows in the resulting table that have a list price. The FILTER expression applies to the table Products but uses a value that you look up in the related table, ProductSubCategory.

=COUNTX(FILTER(Product,RELATED(ProductSubcategory[EnglishProductSubcategoryName])="Caps", Product[ListPrice])

CROSSJOIN

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

Syntax

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

Example

Create two tables like below

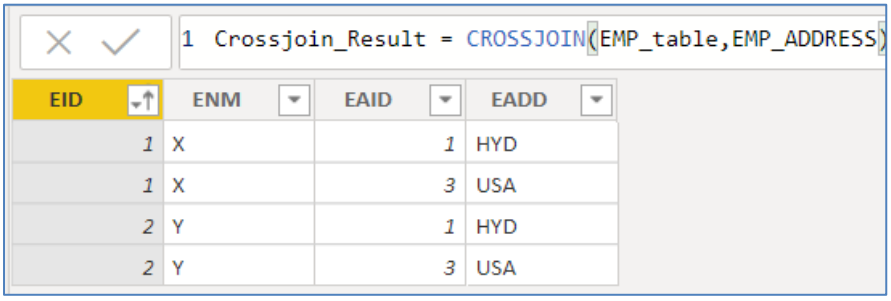
EMP_table =DATATABLE("EID",INTEGER,"ENM",STRING,{{1,"X"},{2,"Y"}})

EMP_ADDRESS=DATATABLE("EAID",INTEGER,"EADD",STRING,{{1,"HYD"},{3,"USA"}})

Modeling menu→New Table, Preview the resule

Crossjoin_Result=CROSSJOIN(EMP_table,EMP_ADDRESS)

Note: Same column should not exist



DISTINCTCOUNT

The DISTINCTCOUNT function counts the number of distinct values in a column.

Syntax

DISTINCTCOUNT(<column>)

Example

Create a new measure and place on a card

Distinct number of students=**DISTINCTCOUNT**(FactPayments[StudentID])

Difference between GENERATE and GENERATE ALL?

Based on row section in table1, it does Cartesian join with table2. Whereas ALL ignore all filters and perform

GENERATE

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

Syntax

GENERATE(<table1>, <table2>)

Difference between GENERATE and GENERATESERIES?

Scenario: Display every location and mode with their total discountfee

Create a new table (Modeling menu->New table) with the below expression, review data

Each location and mode discountfee=

```
GENERATE
(
SUMMARIZE(DimLocation,DimLocation[LocID]),

SUMMARIZE(DimCourseMode,DimCourseMode[ModeID],"Total",sumx(RELATEDTABLE(FactPay

ments),FactPayments[Discount_Fee])

)
```

```
1 Each location and mode discountfee = GENERATE(SUMMARIZE(DimLocation,DimLocation[LocID]),SUMMARIZE
  (DimCourseMode,DimCourseMode[ModeID],"Total",sumx(RELATEDTABLE(FactPayments),FactPayments[Discount_Fee])))
```

LocID	ModeID	Total
HYD	Online	350100
VIZAG	Online	135900
BZA	Online	71100
HYD	Classroom	354600
VIZAG	Classroom	197100
BZA	Classroom	196200
HYD	Customized	133200
VIZAG	Customized	58500
BZA	Customized	29700

Differences between Cross Join and Generate

Generate [passes each record from first dataset, evalauate at second dataset and get data]

```
1 Gen_Table = GENERATE
2 (
3 SUMMARIZE(DimLocation,DimLocation[LocID]),
4
5 SUMMARIZE(DimCourseMode,DimCourseMode[ModeID],"Total",sumx(RELATEDTABLE(FactPayments),FactPayments[Discount_Fee])
6
7 )
8 )
```

LocID	ModeID	Total
HYD	Online	508500
VIZAG	Online	153000
BZA	Online	107100
WRNGL	Online	
KHAMMAM	Online	
HYD	Classroom	357300
VIZAG	Classroom	202500
BZA	Classroom	196200
WRNGL	Classroom	
KHAMMAM	Classroom	
HYD	Customized	133200
VIZAG	Customized	58500
BZA	Customized	29700
WRNGL	Customized	
KHAMMAM	Customized	

Observation:

Locations not having busines no discount fee

Each location based on the mode appropriate value

Note: Generate follows context of data, so the above displayed

Cross Join [perform cross join between two dataset row]

```
1 Cross_tab = CROSSJOIN
2 (
3 SUMMARIZE(DimLocation,DimLocation[LocID]),
4
5 SUMMARIZE(DimCourseMode,DimCourseMode[ModeID],"Total",sumx(RELATEDTABLE(FactPayments),FactPayments[Discount_Fee])
6
7 )
8 )
```

LocID	ModeID	Total
HYD	Online	828900
VIZAG	Online	828900
BZA	Online	828900
WRNGL	Online	828900
KHAMMAM	Online	828900
HYD	Classroom	801000
VIZAG	Classroom	801000
BZA	Classroom	801000
WRNGL	Classroom	801000
KHAMMAM	Classroom	801000
HYD	Customized	221400
VIZAG	Customized	221400
BZA	Customized	221400
WRNGL	Customized	221400
KHAMMAM	Customized	221400

Observation

Non business locations having discount fee of the mode

All locations having same mode value

Note: No data context at the time of cross join

Other Example

In the following example the user wants a summary table of the sales by Region and Product Category for the Resellers channel, like the following table:

SalesTerritory[SalesTerritoryGroup]	ProductCategory[ProductCategoryName]	[Reseller Sales]
Europe	Accessories	$ 142,227.27
Europe	Bikes	$ 9,970,200.44
Europe	Clothing	$ 365,847.63
Europe	Components	$ 2,214,440.19
North America	Accessories	$ 379,305.15
North America	Bikes	$ 52,403,796.85
North America	Clothing	$ 1,281,193.26
North America	Components	$ 8,882,848.05
Pacific	Accessories	$ 12,769.57
Pacific	Bikes	$ 710,677.75
Pacific	Clothing	$ 22,902.38
Pacific	Components	$ 108,549.71

The following code produces the above table:

```
GENERATE(
SUMMARIZE(SalesTerritory, SalesTerritory[SalesTerritoryGroup])
,SUMMARIZE(ProductCategory
, [ProductCategoryName]
, "Reseller Sales", SUMX(RELATEDTABLE(ResellerSales_USD),

, ResellerSales_USD[SalesAmount_USD])
)

)
```

1. The first SUMMARIZE statement, SUMMARIZE(SalesTerritory,

2. SalesTerritory[SalesTerritoryGroup]) ,

produces a table of territory groups, where each row is a territory group, as shown below:

SALESTERRITORY[SALESTERRITORYGROUP]
North America
Europe
Pacific
NA

2. The second SUMMARIZE statement,

SUMMARIZE(ProductCategory, [ProductCategoryName], "Reseller Sales",
SUMX(RELATEDTABLE(ResellerSales_USD), ResellerSales_USD[SalesAmount_USD]))

, produces a table of Product Category groups with the Reseller sales for each group, as shown below:

PRODUCTCATEGORY[PRODUCTCATEGORYNAME]	[RESELLER SALES]
Bikes	$ 63,084,675.04
Components	$ 11,205,837.96
Clothing	$ 1,669,943.27
Accessories	$ 534,301.99

3. However, when you take the above table and evaluate it under the context of each row from the territory

groups table, you obtain different results for each territory.

GENERATEALL

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

Syntax

```
GENERATEALL(<table1>, <table2>)
```

Parameters

TERM	DEFINITION
table1	Any DAX expression that returns a table.
table2	Any DAX expression that returns a table.

Return value

A table with the Cartesian product between each row in *table1* and the table that results from evaluating *table2* in the context of the current row from *table1*

Remarks

- If the evaluation of *table2* for the current row in *table1* returns an empty table, then the current row from *table1* will be included in the results and columns corresponding to *table2* will have null values for that row.This is different than GENERATE() where the current row from *table1* will **not** be included in the results.

All column names from *table1* and *table2* must be different or an error is returned.

Example

In the following example, the user wants a summary table of the sales by Region and Product Category for the
Resellers channel, like the following table:

SALESTERRITORY[SALESTERRITORYGROUP]	PRODUCTCATEGORY[PRODUCTCATEGORY NAME]	[RESELLER SALES]
Europe	Accessories	$ 142,227.27

Europe	Bikes	$ 9,970,200.44
Europe	Clothing	$ 365,847.63
Europe	Components	$ 2,214,440.19
NA	Accessories	

SALESTERRITORY[SALESTERRITORYGROUP]	PRODUCTCATEGORY[PRODUCTCATEGORY NAME]	[RESELLER SALES]
NA	Bikes	
NA	Clothing	
NA	Components	
North America	Accessories	$ 379,305.15
North America	Bikes	$ 52,403,796.85
North America	Clothing	$ 1,281,193.26
North America	Components	$ 8,882,848.05
Pacific	Accessories	$ 12,769.57
Pacific	Bikes	$ 710,677.75
Pacific	Clothing	$ 22,902.38
Pacific	Components	$ 108,549.71

The following code produces the above table:

```
GENERATEALL(
SUMMARIZE(SalesTerritory, SalesTerritory[SalesTerritoryGroup])
,SUMMARIZE(ProductCategory
, [ProductCategoryName]
, "Reseller Sales", SUMX(RELATEDTABLE(ResellerSales_USD),

, ResellerSales_USD[SalesAmount_USD])
)
)
```

1. The first SUMMARIZE produces a table of territory groups, where each row is a
2.
3. territory group, like those listed below:

SALESTERRITORY[SALESTERRITORYGROUP]
North America
Europe
Pacific
NA

2. The second SUMMARIZE produces a table of Product Category groups with the
3.
4.
5. Reseller sales for each group, as shown below:
6.

PRODUCTCATEGORY[PRODUCTCATEGORYNAME]	[RESELLER SALES]
Bikes	$ 63,084,675.04

PRODUCTCATEGORY[PRODUCTCATEGORYNAME]	[RESELLER SALES]
Components	$ 11,205,837.96
Clothing	$ 1,669,943.27
Accessories	$ 534,301.99

3. However, when you take the above table and evaluate the table under the context of
4.
5. each row from the territory groups table, you obtain different results for each
6.
7. territory.Returns the geometric mean of the numbers in a column.

GEOMEAN

To return the geometric mean of an expression evaluated for each row in a table, use GEOMEANX function (DAX).

Syntax

```
GEOMEAN(<column>)
```

Parameters

TERM	DEFINITION
	geometric mean is to be computed.

Return value

A decimal number.

Remarks

Only the numbers in the column are counted. Blanks, logical values, and text are ignored.

GEOMEAN(Table[Column]) is equivalent to GEOMEANX(Table, Table[Column])

Example

The following computes the geometric mean of the Return column in the Investment table:

```
=GEOMEAN( Investment[Return] )
```

GEOMEANX

Returns the geometric mean of an expression evaluated for each row in a table.

To return the geometric mean of the numbers in a column, use GEOMEAN function (DAX).

Syntax

```
GEOMEANX(<table>, <expression>)
```


Parameters

TERM	DEFINITION
table	The table containing the rows for which the expression will be evaluated.
expression	The expression to be evaluated for each row of the table.

Return value

A decimal number.

Remarks

The GEOMEANX function takes as its first argument a table, or an expression that returns a table. The second argument is a column that contains the numbers for which you want to compute the geometric mean, or an expression that evaluates to a column.

Only the numbers in the column are counted. Blanks, logical values, and text are ignored.

Example

The following computes the geometric mean of the ReturnPct column in the Investments table:

```
=GEOMEANX( Investments, Investments[ReturnPct] + 1 )
```

MAX

Returns the largest **numeric value** in a column, or between two scalar expressions.

Syntax

```
MAX(<column>)
MAX(<expression1>, <expression2>)
```

Parameters

TERM	DEFINITION
column	The column in which you want to find the largest numeric value.
expression	Any DAX expression which returns a single numeric value.

Return value

A decimal number.

Remarks

When evaluating a single column that contains numeric values, if the column contains no numbers, MAX returns a blank. If you want to evaluate values that are not numbers, use the MAXA function.

When comparing two expressions, blank is treated as 0 when comparing. That is, Max(1, Blank()) returns 1, and Max(-1, Blank()) returns 0. If both arguments are blank, MAX returns a blank. If either expression returns a value which is not allowed, MAX returns an error.

Example

Create a measure and place on a card

Max value in the discount fee=MAX(FactPayments[Discount_Fee])

Example

The following example returns the largest value between the result of two expressions.

Create a measure and place on a card

=Max(FactPayments[Discount_Fee]), FactPayments[Actual_Fee]))

MAXA

Returns the largest value in a column. **Logical values and blanks** are counted.

Syntax

Difference between MIN, MINA, and MINX?

MAXA(<column>)

Parameters

TERM	DEFINITION
column	The column in which you want to find the largest value.

Return value

A decimal number.

Remarks

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

- Numbers
- Dates

  Logical values, such as TRUE and FALSE. Rows that evaluate to TRUE count as 1; rows that evaluate to FALSE count as 0 (zero).

Empty cells are ignored. If the column contains no values that can be used, MAXA returns 0 (zero).
If you do not want to include logical values and blanks as part of the calculation, use the MAX function.

Example

Create a measure and place on a card

=MAXA(Factpayments[DiscountFee])

Example

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

Create a measure and place on a card

=MAXA([Factpayments[End_Date]])

MAXX

Evaluates an expression for each row of a table and returns the largest numeric value.

Syntax

MAXX(<table>,<expression>)

Parameters

TERM	DEFINITION
table	The table containing the rows for which the expression will be evaluated.
expression	The expression to be evaluated for each row of the table.

Return value

A decimal number.

Remarks

The **table** argument to the MAXX function can be a table name, or an expression that evaluates to a table. The second argument indicates the expression to be evaluated for each row of the table.

Of the values to evaluate, only the following are counted:

- Numbers. If the expression does not evaluate to a number, MAXX returns 0 (zero).
- Dates.

Empty cells, logical values, and text values are ignored. If you want to include non-numeric values in the formula, use the MAXA function.
If a blank cell is included in the column or expression, MAXX returns an empty column.

Example

Add a new column to the factpayments and see the result

```
=MAXX(Factpayments, Factpayments[TaxAmt]+ Factpayments[Discount_Fee])
```


Other Example

The following formula first filters the table InternetSales, by using a FILTER expression, to return a subset of orders for a specific sales region, defined as [SalesTerritory] = 5. The MAXX function then evaluates the expression used as the second argument for each row of the filtered table, and returns the highest amount for taxes and

shipping for just those orders. The expected result is 250.3724.

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

MEDIAN

Returns the median of numbers in a column.

To return the median of an expression evaluated for each row in a table, use MEDIANX function (DAX).

Syntax

```
MEDIAN(<column>)
```

Parameters

TERM	DEFINITION
column	The column that contains the numbers for which the median is to be computed.

Return value

A decimal number.

Remarks

Only the numbers in the column are counted. Blanks, logical values, and text are ignored.

MEDIAN(Table[Column]) is equivalent to MEDIANX(Table, Table[Column]).

Example

The following computes the median of a column named Age in a table named Customers:

```
=MEDIAN( Customers[Age] )
```


MEDIANX

Returns the median number of an expression evaluated for each row in a table.

To return the median of numbers in a column, use MEDIAN function (DAX).

Syntax

```
MEDIANX(<table>, <expression>)
```

Parameters

TERM	DEFINITION
table	The table containing the rows for which the expression will be evaluated.
expression	The expression to be evaluated for each row of the table.

Return value

A decimal number.

Remarks

The MEDIANX function takes as its first argument a table, or an expression that returns a table. The second argument is a column that contains the numbers for which you want to compute the median, or an expression that evaluates to a column.

Only the numbers in the column are counted.
Logical values and text are ignored.
MEDIANX does not ignore blanks; however, MEDIAN does ignore blanks

Example

The following computes the median age of customers who live in the USA.

```
=MEDIANX( FILTER(Customers, RELATED( Geography[Country]="USA" ) ),
Customers[Age] )
```


MIN

Returns the smallest numeric value in a column, or between two scalar expressions. Ignores logical values and text.

Syntax

MIN(<column>)

MIN(<expression1>, <expression2>)

Parameters

TERM	DEFINITION
column	The column in which you want to find the smallest numeric value.
expression	Any DAX expression which returns a single numeric value.

Return value

A decimal number.

Remarks

The MIN function takes a column or two expressions as an argument, and returns the smallest numeric value. The following types of values in the columns are counted:

- Numbers
- Dates
- Blanks
- If the column contains no numerical data, MIN returns blanks.

When evaluating a column, empty cells, logical values, and text are ignored. If you want to include logical values and text representations of numbers in a reference as part of the calculation, use the MINA function.

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 either expression returns a value which is not allowed, MIN returns an error.

Example

Create a measure and place on a card

```
=MIN(Factpayments[Discount_Fee])
```

Example

Create a measure and place on a card

```
=MIN(Factpayments[Course_End_DT)
```

Example

The following example returns the smallest value from the result of two scalar expressions.

```
=Min(Factpayments[Discount_Fee], Factpayments[Actual_Fee])
```

MINA

Returns the smallest value in a column, including any **logical values and numbers represented as text.**

Syntax

```
MINA(<column>)
```

Parameters

TERM	DEFINITION
column	The column for which you want to find the minimum value.

Return value

A decimal number.

Remarks

The MINA function takes as argument a column that contains numbers, and determines the smallest value as follows:

- If the column contains no numeric values, MINA returns 0 (zero).
- Rows in the column that evaluates to logical values, such as TRUE and FALSE are treated as 1 if TRUE and 0 (zero) if FALSE.
- Empty cells are ignored.

If you do not want to include logical values and text as part of the calculation, use the MIN function instead.

Example

Create a measure and place on a card

```
=MINA(Factpayments[Discount_Fee])
```

Example

Create a measure and place on a card

```
=MINA([DimStudent[PostalCode])
```

MINX

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

Syntax

MINX(<table>, < expression>)

Parameters

TERM	DEFINITION
table	The table containing the rows for which the expression will be evaluated.
expression	The expression to be evaluated for each row of the table.

Return value

A decimal number.

Remarks

The MINX function takes as its first argument a table, or an expression that returns a table. The second argument contains the expression that is evaluated for each row of the table.

The MINX function evaluates the results of the expression in the second argument according to the following rules:

- Only numbers are counted. If the expression does not result in a number, MINX returns 0 (zero).
- Empty cells, logical values, and text values are ignored. Numbers represented as text are treated as text.

If you want to include logical values and text representations of numbers in a reference as part of the calculation, use the MINA function.

Create a measure and place on a card

=MINX(FILTER(Factpayments, [LocID] = "HYD"),[Discount_Fee])

Other Example

The following example uses the same filtered table as in the previous example, but instead of merely looking up values in the column for each row of the filtered table, the function calculates the sum of two columns, Freight and TaxAmt, and returns the smallest value resulting from that calculation.

```
=MINX( FILTER(InternetSales, InternetSales[SalesTerritoryKey] = 5),
InternetSales[Freight] + InternetSales[TaxAmt])
```

NORM.DIST

Returns the normal distribution for the specified mean and standard deviation.

Syntax

```
NORM.DIST(X, Mean, Standard_dev, Cumulative)
```

Example


```
EVALUATE { NORM.DIST(42, 40, 1.5, TRUE) }
```

Returns

[VALUE]
0.908788780274132

NORM.INV

The inverse of the normal cumulative distribution for the specified mean and standard deviation.

Syntax

```
NORM.INV(Probability, Mean, Standard_dev)
```

Example

```
EVALUATE { NORM.INV(0.908789, 40, 1.5) }
```

Returns

[VALUE]
42.00000200956628780274132

NORM.S.DIST

Returns the standard normal distribution (has a mean of zero and a standard deviation of one).

Syntax

NORM.S.DIST(Z, Cumulative)

Example

EVALUATE { NORM.S.DIST(1.333333, TRUE) }

Returns

[VALUE]
0.908788725604095

NORM.S.INV

Returns the inverse of the standard normal cumulative distribution. The distribution has a mean of zero and a standard deviation of one.

Syntax

NORM.S.INV(Probability)

Example

EVALUATE { NORM.S.INV(0.908789) }

Returns

[VALUE]
1.33333467304411

PERCENTILEX.EXC

Returns the k-th percentile of values in a range, where k is in the range 0..1, exclusive.

To return the percentile number of an expression evaluated for each row in a table, use PERCENTILEX.EXCfunction (DAX).

Syntax

PERCENTILE.EXC(<column>, <k>)

Return value

The k-th percentile of values in a range, where k is in the range 0..1, exclusive.

PERCENTILEX.INC

Returns the k-th percentile of values in a range, where k is in the range 0..1, inclusive.

To return the percentile number of an expression evaluated for each row in a table, use PERCENTILEX.INCfunction (DAX).

Syntax

PERCENTILE.INC(<column>, <k>)

Return value

The k-th percentile of values in a range, where k is in the range 0..1, inclusive.

PERCENTILEX.EXC

Returns the percentile number of an expression evaluated for each row in a table.
To return the percentile of numbers in a column, use PERCENTILE.EXC function (DAX).

Syntax

PERCENTILEX.EXC(<table>, <expression>, k)

Return value

The percentile number of an expression evaluated for each row in a table.

PERCENTILEX.INC

Returns the percentile number of an expression evaluated for each row in a table.

To return the percentile of numbers in a column, use PERCENTILE.INC function (DAX).

Syntax

PERCENTILEX.INC(<table>, <expression>;, k)

Return value

The percentile number of an expression evaluated for each row in a table.

PERMUT

Returns the number of permutations for a given number of objects that can be selected from number objects. A permutation is any set or subset of objects or events where internal order is significant. Permutations are different from combinations, for which the internal order is not significant. Use this function for lottery-style probability calculations.

Syntax

PERMUT(number, number_chosen)

Return value

Returns the number of permutations for a given number of objects that can be selected from number objects

Both arguments are truncated to integers.
If number or number_chosen is nonnumeric, PERMUT returns the #VALUE! error value.
If number ≤ 0 or if number_chosen < 0, PERMUT returns the #NUM! error value.
If number < number_chosen, PERMUT returns the #NUM! error value.

The equation for the number of permutations is:

$$P_{k,n} = \frac{n!}{(n-k)!}$$

Example

FORMULA	DESCRIPTION	RESULT
=PERMUT(3,2)	Permutations possible for a group of 3 objects where 2 are chosen.	6

POISSON.DIST

Returns the Poisson distribution. A common application of the Poisson distribution is predicting the number of events over a specific time, such as the number of cars arriving at a toll plaza in 1 minute.

Syntax

POISSON.DIST(x,mean,cumulative)

Return value

Returns the Poisson distribution.

Remarks

If x is not an integer, it is rounded.
If x or mean is nonnumeric, POISSON.DIST returns the #VALUE! error value.
If x < 0, POISSON.DIST returns the #NUM! error value.
If mean < 0, POISSON.DIST returns the #NUM! error value.
POISSON.DIST is calculated as follows.
For cumulative = FALSE:

$$POISSON = \frac{e^{-\lambda}\lambda^{x}}{x!}$$

For cumulative = TRUE:

$$CUMPOISSON = \sum_{k=0}^{x} \frac{e^{-\lambda}\lambda^{k}}{k!}$$

RANK.EQ

Returns the ranking of a number in a list of numbers.

Syntax

RANK.EQ(<value>, <columnName>[, <order>])

Return value

A number indicating the rank of *value* among the numbers in *columnName*.

Example1: Display ranking for each tax amount value

Create new Column

Ranking = RANK.EQ(FactPayments[Tax Amount], FactPayments[Tax Amount], DESC)

```
1 Ranking = RANK.EQ(FACT1[Tax amount], FACT1[Tax amount], DESC)
```

LocationID	Date	StudentID	Actual_Fee	Discount_Fee	Tax amount	DiscountValue	Ranking
1000 HYD	Monday, March 18, 2019	1098	15000	13500	270	1500	3
1001 HYD	Sunday, March 17, 2019	1097	14000	12600	252	1400	4
1003 HYD	Friday, March 15, 2019	1095	12000	10800	216	1200	6
1000 HYD	Wednesday, March 13, 2019	1093	13000	11700	234	1300	5
1002 HYD	Monday, March 11, 2019	1091	10000	9000	180	1000	7
1003 HYD	Wednesday, March 7, 2018	1035	16000	14400	288	1600	2
1002 BZA	Wednesday, January 3, 2018	1002	18000	16200	324	1800	1

Example 2: Findout the ranking of Taxamount in the GTAX table for the local tax

1.Add the below table or file data

```
Tax_no, Gtax
1,480
2,440
3,324
4,300
5,290
6,288
7,280,
8,270,
9,260
10,252
11,240
12,234
13,230
14,216
15,200
16,190
17,180
18,170
19,160
20,150
```

2.Goto Factpayments and write like below to see the Global tax ranking for the local tax amunt

GRanking = RANK.EQ(FACT_1[Tax amount],GTAX[Gtax],DESC)

```
GRanking = RANK.EQ(FACT_1[Tax amount],GTAX[ Gtax],DESC)
```

LocationID	Date	StudentID	Actual_Fee	Discount_Fee	Tax amount	DiscountValue	GRanking
HYD	Monday, March 18, 2019	1098	15000	13500	270	1500	8
HYD	Sunday, March 17, 2019	1097	14000	12600	252	1400	10
HYD	Friday, March 15, 2019	1095	12000	10800	216	1200	14
HYD	Wednesday, March 13, 2019	1093	13000	11700	234	1300	12
HYD	Monday, March 11, 2019	1091	10000	9000	180	1000	17
HYD	Wednesday, March 7, 2018	1035	16000	14400	288	1600	6
BZA	Wednesday, January 3, 2018	1002	18000	16200	324	1800	3

```
1 GRanking = RANK.EQ(FACT1[Tax amount],GTAX[ Gtax],DESC)
```

Date	StudentID	Actual_Fee	Discount_Fee	Tax amount	DiscountValue	Ranking	GRanking
Monday, March 18, 2019	1098	15000	13500	270	1500	3	8
Sunday, March 17, 2019	1097	14000	12600	252	1400	4	10
Friday, March 15, 2019	1095	12000	10800	216	1200	6	14
Wednesday, March 13, 2019	1093	13000	11700	234	1300	5	12
Monday, March 11, 2019	1091	10000	9000	180	1000	7	17
Wednesday, March 7, 2018	1035	16000	14400	288	1600	2	6
Wednesday, January 3, 2018	1002	18000	16200	324	1800	1	3

Other Example

The following example creates a calculated column that ranks the values in SalesAmount_USD, from the *InternetSales_USD* table, against all numbers in the same column.

```
=RANK.EQ(InternetSales_USD[SalesAmount_USD],
InternetSales_USD[SalesAmount_USD])
```

Example

The following example ranks a subset of values against a given sample. Assume that you have a table of local students with their performance in a specific national test and, also, you have the entire set of scores in that national test. The following calculated column will give you the national ranking for each of the local students.

```
=RANK.EQ(Students[Test_Score], NationalScores[Test_Score])
```

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>]]])

Return value

The rank number of *value* among all possible values of *expression* evaluated for all rows of *table* numbers.

Remarks

- If *expression* or *value* evaluates to BLANK it is treated as a 0 (zero) for all expressions that result in a number, or as an empty text for all text expressions.
- If *value* is not among all possible values of *expression* then RANKX temporarily adds *value* to the values from *expression* and re-evaluates RANKX to determine the proper rank of *value*.
- Optional arguments might be skipped by placing an empty comma (,) in the argument list, i.e.

Scenario: Based on each course discount fee value providing rank

Ranking = RANKX(ALL(DimCourse), SUMX(RELATEDTABLE(FactPayments), FactPayments[Discount_Fee]))

RankX is row wise operation, each course joined with Factpayments and gets discountfee values, finding discount fee total and then generate ranking based on that.

NOTE: CREATE A NEW COLUMN WITH RANK EXPRESSION

```
1 Ranking = RANKX(ALL(DimCourse), SUMX(RELATEDTABLE(FactPayments), FactPayments[Discount_Fee]))
```

CourseID	Coursename	Duration	Ranking
MSBI-F	MSBI Fast Track	30	5
MSBI-N	MSBI Normal Track	50	2
MSBI-C	MSBI Customized	20	1
Teradata-F	Teradata Fast Track	30	9
Teradata-N	Teradata Normal Track	40	8
Teradata-C	Teradata Customized	20	7
Power BI-F	POWER BI Fast Track	20	3
Power BI-N	POWER BI Fast Track	30	4
Power BI-C	Power BI Customized	20	6
Informatica-F	Informatica Fast Track	30	12
Informatica-N	Informatica Normal Track	40	14
Informatica-C	Informatica Customized	20	11
Azure-F	Azure Fast Track	20	10
Azure-N	Azure Normal Track	30	14
Azure-C	Azure Customized	20	12

ROW

Returns a table with a single row containing values that result from the expressions given to each column.

Syntax

ROW(<name>, <expression>[[,<name>, <expression>]...])

Parameters

TERM	DEFINITION
name	The name given to the column, enclosed in double quotes.
expression	Any DAX expression that returns a single scalar value to populate. *name*.

Return value

A single row table

Remarks

Arguments must always come in pairs of *name* and *expression*.

Example:

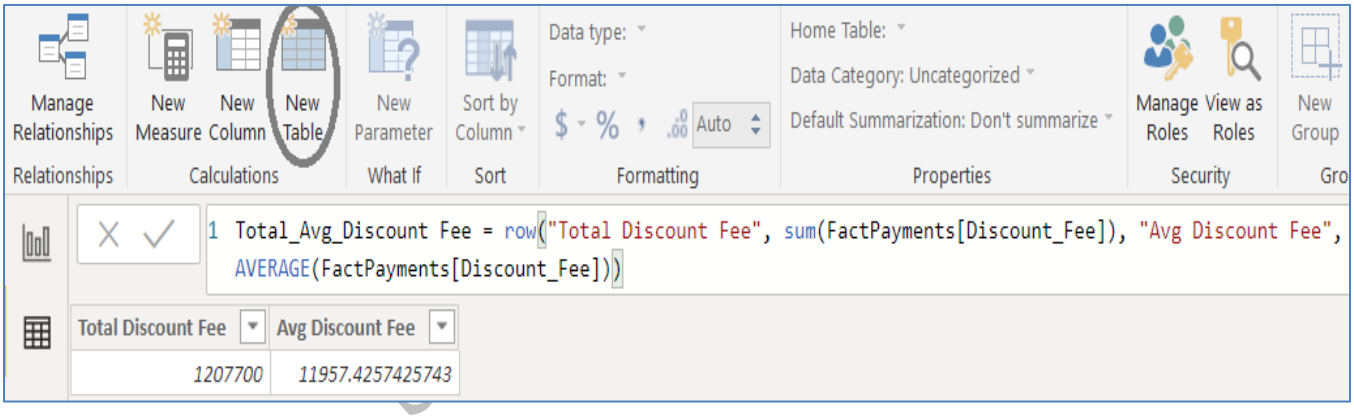
Construct a row with total and average discount fee values

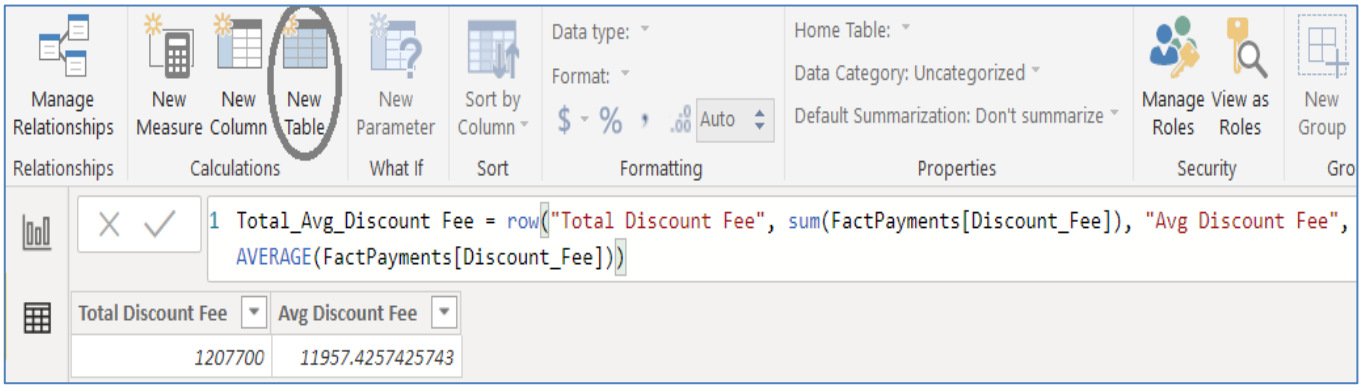
Total_Avg_Discount Fee =**row**("Total Discount

Fee",sum(FactPayments[Discont_Fee]),"Avg Discount Fee",

AVERAGE(FactPayments[Discount_Fee]))

The code is split in two lines for readability purposes





SAMPLE

Returns a sample of N rows from the specified table.

Syntax

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

Display sample 3 records

Sample 3 records = sample(3,'Sales',Sales[CountryID])

Sample 3 courses=sample(3,DimCourse, DimCourse[CourseID])

SELECTCOLUMNS

Adds calculated columns to the given table or table expression.

Syntax

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

Example:

SELECTCOLUMNS(Factpayments, "LocID", [LocID]&", "&[ModeID])

SIN

Returns the sine of the given angle.

Syntax

SIN(NUMBER)

Return value

Returns the sine of the given angle.

Parameters

TERM	DEFINITION
number	Required. The angle in radians for which you want the sine.

Remarks

If your argument is in degrees, multiply it by PI()/180 or use the RADIANS function to convert it to radians.

Example

FORMULA	DESCRIPTION	RESULT
=SIN(PI())	Sine of pi radians (0, approximately).	0.0
=SIN(PI()/2)	Sine of pi/2 radians.	1.0
=SIN(30*PI()/180)	Sine of 30 degrees.	0.5
=SIN(RADIANS(30))	Sine of 30 degrees.	0.5

SINH

Returns the hyperbolic sine of a number

Syntax

SINH(NUMBER)

Parameters

TERM	DEFINITION
number	Required. Any real number.

Return value

Returns the hyperbolic sine of a number.

Remarks

The formula for the hyperbolic sine is:

$$SINH(z) = \frac{e^{z} - e^{-z}}{2}$$

Example

FORMULA	DESCRIPTION	RESULT
=2.868*SINH(0.0342*1.03)	Probability of obtaining a result of less than 1.03 seconds.	0.1010491

STDEV.S

Returns the standard deviation of a sample population.

Syntax

STDEV.S(<ColumnName>)

Example

The following example shows the formula for a measure that calculates the standard deviation of the column, SalesAmount_USD, when the table InternetSales_USD is the sample population.

=STDEV.S(InternetSales_USD[SalesAmount_USD])

STDEV.P

Returns the standard deviation of the entire population.

Syntax

STDEV.P(<ColumnName>)

Example

The following example shows the formula for a measure that calculates the standard deviation of the column, SalesAmount_USD, when the table InternetSales_USD is the entire population.

=STDEV.P(InternetSales_USD[SalesAmount_USD])

STDEVX.S

Returns the standard deviation of a sample population.

Syntax

STDEVX.S(<table>, <expression>)

Example

The following example shows the formula for a calculated column that estimates the standard deviation of the unit price per product for a sample population, when the formula is used in the Product table.

```
=STDEVX.S(RELATEDTABLE(InternetSales_USD),
InternetSales_USD[UnitPrice_USD] –
(InternetSales_USD[DiscountAmount_USD]/InternetSales_USD[
OrderQuantity]))
```

STDEVX.P

Returns the standard deviation of the entire population.

Syntax

STDEVX.P(<table>, <expression>)

Example

The following example shows the formula for a calculated column that calculates the standard deviation of the unit price per product, when the formula is used in the *Product* table.

=STDEVX.P(RELATEDTABLE(InternetSales_USD), InternetSales_USD[UnitPrice_USD] – (InternetSales_USD[DiscountAmount_USD]/InternetSales_USD[OrderQuantity]))

SQRTPI

Returns the square root of (number * pi)

Syntax

SQRTPI(NUMBER)

Remarks

FORMULA	DESCRIPTION	RESULT
=SQRTPI(1)	Square root of pi.	1.772454
=SQRTPI(2)	Square root of 2 * pi.	2.506628

SUMMARIZE

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

Syntax

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

Parameters

TERM	DEFINITION
table	Any DAX expression that returns a table of data.
groupBy_columnName	(Optional) The qualified name of an existing column to be 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 Asummarized columns designed by the name arguments.

Remarks

1. Each column for which you define a name must have a corresponding expression; otherwise, an error is returned. The first argument, name, defines the name of the column in the results. The second argument, expression, defines the calculation performed to obtain the value for each row in that column.
2. groupBy_columnName must be either in *table* or in a related table to *table*.
3. Each name must be enclosed in double quotation marks.
4. The function groups a selected set of rows into a set of summary rows by the values of one or more groupBy_columnName columns. One row is returned for each group.

Scenario1: Based on courseID display sum and average of discount fee

Course wise total and average =
summarize(FactPayments,FactPayments[CourseID],"Total",sum(FactPayments[Discount_Fee]),"Average",AVERAGE(FactPayments[Discount_Fee]))

Scenario2: Based on country and date display sum and average of discount fee

SQL QUERY:

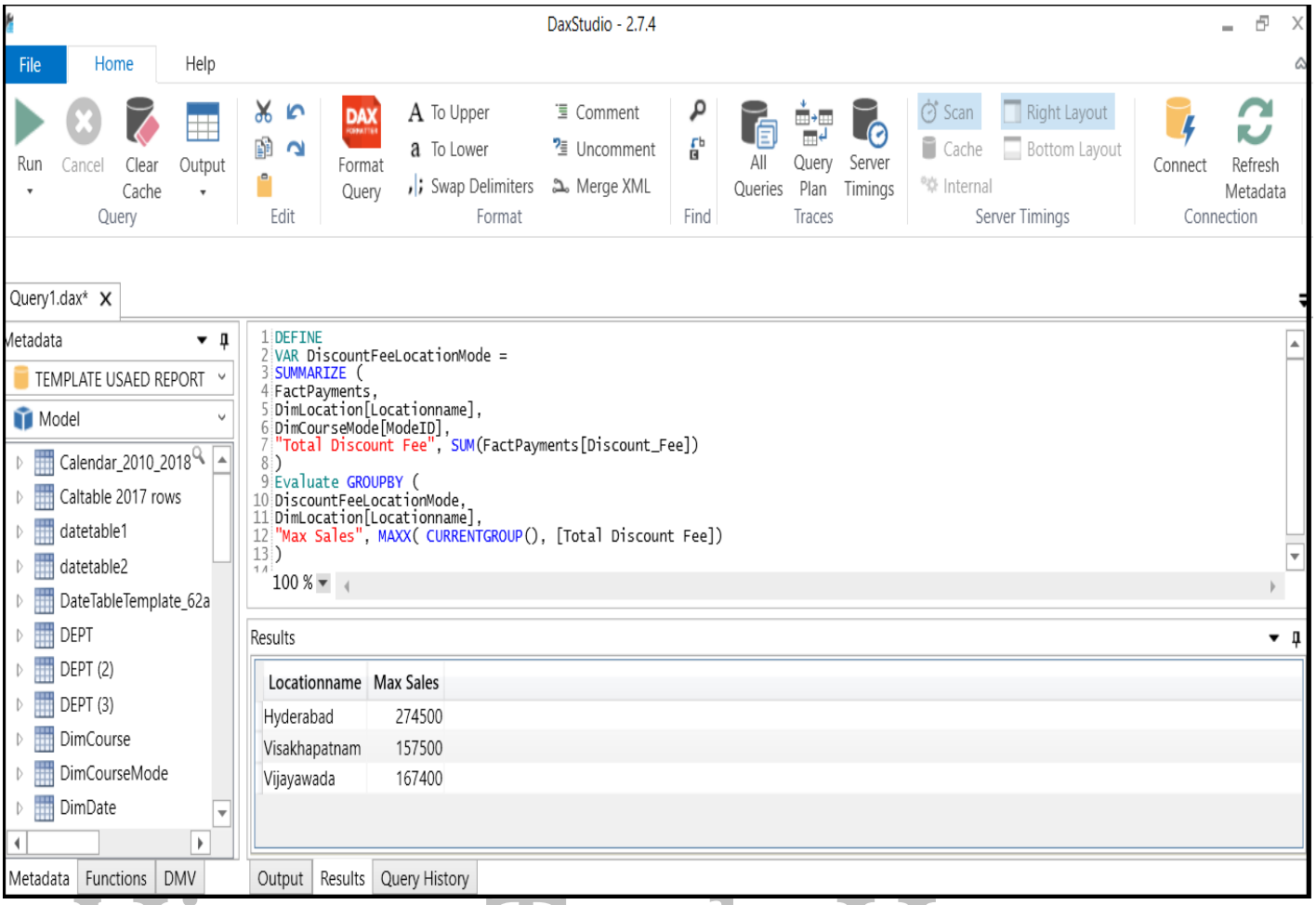
SELECT CouseID,ModeID, Sum(Discount_fee) "Total", Avg(Discount_fee) "Avg"
From FactPayments
Group by CourseID, ModeID

```
SUMMARIZE(FactPayments,FactPayments[CourseID],FactPayments[ModeID],"Total",
sum(FactPayments[Discount_Fee]),"avg",AVERAGE(FactPayments[Discount_Fee]))
```

Sales by country and date =
GENERATE(SUMMARIZE(SalesGeography,SalesGeography[CountryID]),SUMMARIZE(Sales,Dates[DateID],"Sales by country and date", SUM(Sales[Sales])))

USING GROUP, SUMMARIZE AND VAR TO FINDOUT BASED ON DISCOUNT FEE

```
DEFINE
VAR DiscountFeeLocationMode =
SUMMARIZE (
FactPayments,
DimLocation[Locationname],
DimCourseMode[ModeID],
"Total Discount Fee", SUM(FactPayments[Discount_Fee])
)
Evaluate GROUPBY (
DiscountFeeLocationMode,
DimLocation[Locationname],
"Max Sales", MAXX( CURRENTGROUP(), [Total Discount Fee])
)
```



Other Examples:

The following example returns a summary of the reseller sales grouped around the calendar year and the product category name, this result table allows you to do analysis over the reseller sales by year and product category.

```
SUMMARIZE(ResellerSales_USD
DateTime[CalendarYear]
, ProductCategory[ProductCategoryName]
,
, "Sales Amount (USD)", SUM(ResellerSales_USD[SalesAmount_USD])
, "Discount Amount (USD)", SUM(ResellerSales_USD[DiscountAmount])
)
```

The following table shows a preview of the data as it would be received by any function expecting to receive a table:

DATETIME[CALENDARYEAR]	PRODUCTCATEGORY[PRODUCTCATEGORYNAME]	[SALES AMOUNT (USD)]	[DISCOUNT AMOUNT (USD)]
2008	Bikes	12968255.42	36167.6592
2005	Bikes	6958251.043	4231.1621
2006	Bikes	18901351.08	178175.8399
2007	Bikes	24256817.5	276065.992
2005	Components	574256.9865	0
2006	Components	3428213.05	948.7674
2007	Components	5195315.216	4226.0444
2008	Clothing	366507.844	4151.1235
2005	Clothing	31851.1628	90.9593
2006	Clothing	455730.9729	4233.039
2007	Clothing	815853.2868	12489.3835
2005	Accessories	18594.4782	4.293
2006	Accessories	86612.7463	1061.4872
2007	Accessories	275794.8403	4756.6546

Advanced SUMMARIZE options

SUMMARIZE with ROLLUP

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

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

ROLLUP parameters

groupBy_columnName

The qualified name of an existing column to be used to create summary groups based on the values found in it.

This parameter cannot be an expression.

Note: All other SUMMARIZE parameters are explained before and not repeated here for brevity.

Remarks

- The columns mentioned in the ROLLUP expression cannot be referenced as part of a *groupBy_columnName* columns.

Example

The following example adds roll-up rows to the Group-By columns of the SUMMARIZE function call.

```
SUMMARIZE(ResellerSales_USD
    , ROLLUP( DateTime[CalendarYear], ProductCategory[ProductCategoryName])
    , "Sales Amount (USD)", SUM(ResellerSales_USD[SalesAmount_USD])
    , "Discount Amount (USD)", SUM(ResellerSales_USD[DiscountAmount])
)
```

The following table shows a preview of the data as it would be received by any function expecting to receive a table:

DATETIME[CALENDARYEAR]	PRODUCTCATEGORY[PRODUCTCATEGORYNAME]	[SALES AMOUNT (USD)]	[DISCOUNT AMOUNT (USD)]
2008	Bikes	12968255.42	36167.6592
2005	Bikes	6958251.043	4231.1621
2006	Bikes	18901351.08	178175.8399
2007	Bikes	24256817.5	276065.992
2008	Components	2008052.706	39.9266
2005	Components	574256.9865	0
2006	Components	3428213.05	948.7674
2007	Components	5195315.216	4226.0444
2008	Clothing	366507.844	4151.1235
2005	Clothing	31851.1628	90.9593
2006	Clothing	455730.9729	4233.039
2007	Clothing	815853.2868	12489.3835
2008	Accessories	153299.924	865.5945
	PRODUCTCATEGORY[PRODUC		

DATETIME[CALENDARYEAR]	TCATEGORYNAME]	[SALES AMOUNT (USD)]	[DISCOUNT AMOUNT (USD)]
2005	Accessories	18594.4782	4.293
2006	Accessories	86612.7463	1061.4872
2007	Accessories	275794.8403	4756.6546
2008		15496115.89	41224.3038
2005		7582953.67	4326.4144
2006		22871907.85	184419.1335
2007		30543780.84	297538.0745
		76494758.25	527507.9262

ROLLUPGROUP

ROLLUPGROUP() can be used to calculate groups of subtotals. If used in-place of ROLLUP, ROLLUPGROUP will yield the same result by adding roll-up rows to the result on the groupBy_columnName columns. However, the addition of ROLLUPGROUP() inside a ROLLUP syntax can be used to prevent partial subtotals in roll-up rows.

The following example shows only the grand total of all years and categories without the subtotal of each year with all categories:

```
SUMMARIZE(ResellerSales_USD
        , ROLLUP(ROLLUPGROUP( DateTime[CalendarYear],
        , ProductCategory[ProductCategoryName]))
        , "Sales Amount (USD)", SUM(ResellerSales_USD[SalesAmount_USD])
        , "Discount Amount (USD)", SUM(ResellerSales_USD[DiscountAmount])
)
```

The following table shows a preview of the data as it would be received by any function expecting to receive a table:

DATETIME[CALENDARYEAR]	PRODUCTCATEGORY[PRODUCTCATEGORYNAME]	[SALES AMOUNT (USD)]	[DISCOUNT AMOUNT (USD)]
2008	Bikes	12968255.42	36167.6592
2005	Bikes	6958251.043	4231.1621
2006	Bikes	18901351.08	178175.8399
2007	Bikes	24256817.5	276065.992

2008	Components	2008052.706	39.9266
2005	Components	574256.9865	0
2006	Components	3428213.05	948.7674

DATETIME[CALENDARYEAR]	PRODUCTCATEGORY[PRODUCTCATEGORYNAME]	[SALES AMOUNT (USD)]	[DISCOUNT AMOUNT (USD)]
2007	Components	5195315.216	4226.0444
2008	Clothing	366507.844	4151.1235
2005	Clothing	31851.1628	90.9593
2006	Clothing	455730.9729	4233.039
2007	Clothing	815853.2868	12489.3835
2008	Accessories	153299.924	865.5945
2005	Accessories	18594.4782	4.293
2006	Accessories	86612.7463	1061.4872
2007	Accessories	275794.8403	4756.6546
		76494758.25	527507.9262

SUMMARIZE with ISSUBTOTAL

Enables the user to create another column, in the Summarize function, that returns True if the row contains sub-total values for the column given as argument to ISSUBTOTAL, otherwise returns False.

SUMMARIZE(<table>, <groupBy_columnName>[, <groupBy_columnName>]...[, ROLLUP(<groupBy_columnName>[,< groupBy_columnName>...])][, <name>, {<expression>|ISSUBTOTAL(<columnName>)}]...)

ISSUBTOTAL parameters

columnName

The name of any column in table of the SUMMARIZE function or any column in a related table to table.

Return value

A **True** value if the row contains a sub-total value for the column given as argument, otherwise returns **False**

Remarks

- ISSUBTOTAL can only be used in the expression part of a SUMMARIZE function.
- ISSUBTOTAL must be preceded by a matching *name* column.

Example

The following sample generates an ISSUBTOTAL() column for each of the ROLLUP() columns in the given SUMMARIZE() function call.

```
SUMMARIZE(ResellerSales_USD
    , ROLLUP( DateTime[CalendarYear], ProductCategory[ProductCategoryName])
    , "Sales Amount (USD)", SUM(ResellerSales_USD[SalesAmount_USD])
    , "Discount Amount (USD)", SUM(ResellerSales_USD[DiscountAmount])
    , "Is Sub Total for DateTimeCalendarYear", ISSUBTOTAL(DateTime[CalendarYear])
    , "Is Sub Total for ProductCategoryName",
      ISSUBTOTAL(ProductCategory[ProductCategoryName])
)
```

The following table shows a preview of the data as it would be received by any function expecting to receive a table:

[IS SUB TOTAL FOR DATETIMECALENDARYEAR]	[IS SUB TOTAL FOR PRODUCTCATEGORYNAME]	DATETIME[CALENDARYEAR]	PRODUCTCATEGO RY[PRODUCTCATEGORYNAME]	[SALES AMOUNT (USD)]	[DISCOUNT AMOUNT (USD)]
FALSE	FALSE				
FALSE	FALSE	2008	Bikes	12968255.42	36167.6592
FALSE	FALSE	2005	Bikes	6958251.043	4231.1621
FALSE	FALSE	2006	Bikes	18901351.08	178175.8399
FALSE	FALSE	2007	Bikes	24256817.5	276065.992
FALSE	FALSE	2008	Components	2008052.706	39.9266
FALSE	FALSE	2005	Components	574256.9865	0
FALSE	FALSE	2006	Components	3428213.05	948.7674

FALSE	FALSE	2007	Components	5195315.216	4226.0444
FALSE	FALSE	2008	Clothing	366507.844	4151.1235
FALSE	FALSE	2005	Clothing	31851.1628	90.9593
FALSE	FALSE	2006	Clothing	455730.9729	4233.039
FALSE	FALSE	2007	Clothing	815853.2868	12489.3835
FALSE	FALSE	2008	Accessories	153299.924	865.5945
FALSE	FALSE	2005	Accessories	18594.4782	4.293
FALSE	FALSE	2006	Accessories	86612.7463	1061.4872
FALSE	FALSE	2007	Accessories	275794.8403	4756.6546
FALSE	TRUE				
FALSE	TRUE	2008		15496115.89	41224.3038
FALSE	TRUE	2005		7582953.67	4326.4144
FALSE	TRUE	2006		22871907.85	184419.1335
FALSE	TRUE	2007		30543780.84	297538.0745
TRUE	TRUE			76494758.25	527507.9262

T.DIST

Returns the Student's left-tailed t-distribution.

Syntax: **T.DIST(X,Deg_freedom,Cumulative)**

Return value

The Student's left-tailed t-distribution.

Example

```
EVALUATE { T.DIST(60, 1, TRUE) }
```

[VALUE]

0.994695326367377

T.DIST.2T

Returns the **two-tailed** Student's t-distribution.

Syntax: T.DIST.2T(X,Deg_freedom)

Return value

The two-tailed Student's t-distribution.

Example

```
EVALUATE { T.DIST.2T(1.959999998, 60) }
```

[VALUE]

0.054644929975921

T.DIST.RT

Returns the **right-tailed** Student's t-distribution.

Syntax: T.DIST.RT(X,Deg_freedom)

Return Value:

The right-tailed Student's t-distribution.

Example : EVALUATE { T.DIST.RT(1.959999998, 60) }

Returns:

0.0273224649879605

T.INV

Returns the **left-tailed inverse** of the Student's t-distribution.

Syntax

T.INV(Probability,Deg_freedom)

Parameters

TERM	DEFINITION
Probability	The probability associated with the Student's t-distribution.
Deg_freedom	The number of degrees of freedom with which to characterize the distribution.

Return value

The left-tailed inverse of the Student's t-distribution.

Example

EVALUATE { T.INV(0.75, 2) }

Returns

[VALUE]
0.816496580927726

T.INV.2T

Returns the two-tailed inverse of the Student's t-distribution.

Syntax

```
T.INV.2T(Probability,Deg_freedom)
```

Parameters

TERM	DEFINITION
Probability	The probability associated with the Student's t-distribution.
Deg_freedom	The number of degrees of freedom with which to characterize the distribution.

Return value

The two-tailed inverse of the Student's t-distribution.

Example

```
EVALUATE { T.INV.2T(0.546449, 60) }
```

Returns

[VALUE]
0.606533075825759

TAN

Returns the tangent of the given angle.

Syntax

```
TAN(NUMBER)
```

Return value

Parameters

TERM	DEFINITION
number	Required. The angle in radians for which you want the tangent.

Remarks

If your argument is in degrees, multiply it by PI()/180 or use the RADIANS function to convert it to radians.

Example

FORMULA	DESCRIPTION	RESULT
=TAN(0.785)	Tangent of 0.785 radians (0.99920)	0.99920
=TAN(45*PI()/180)	Tangent of 45 degrees (1)	1
=TAN(RADIANS(45))	Tangent of 45 degrees (1)	1

TANH

Returns the hyperbolic tangent of a number.

Syntax

TANH(NUMBER)

Parameters

TERM	DEFINITION
number	Required. Any real number.

Return value

Returns the hyperbolic tangent of a number.

Remarks

The formula for the hyperbolic tangent is:

$$\text{TANH}(z) = \frac{\text{SINH}(z)}{\text{COSH}(z)}$$

Example

=TANH(-2)

=TANH(0)

=TANH(0.5)

TOPN

Returns the top N rows of the specified table.

Syntax

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

Example

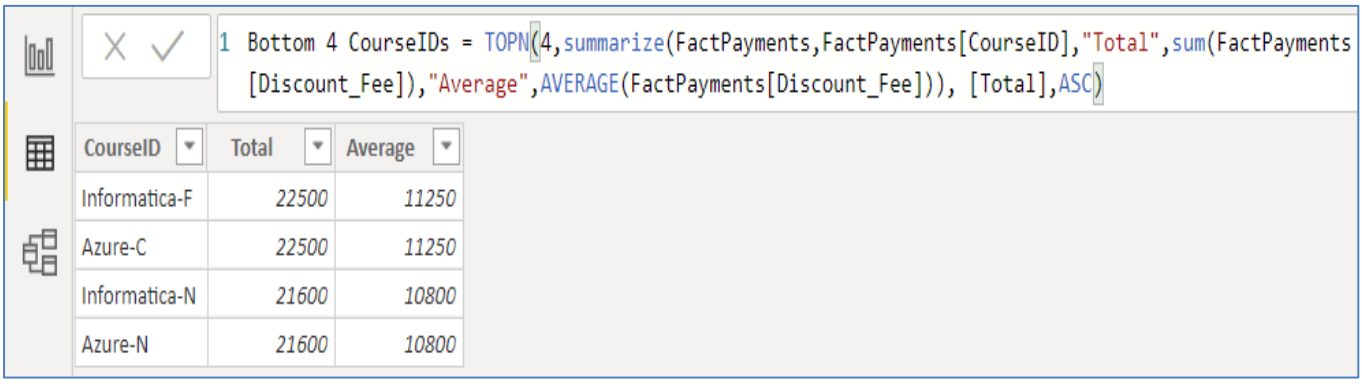
Display Top 4 CourseIDs based on discountfee

First findout courseids and discount fee, later findout top 4 values

```
Top 4 CourseIDs =

TOPN(4,

summarize(FactPayments,FactPayments[CourseID],"Total",

sum(FactPayments[Discount_Fee]),"Average",AVERAGE(FactPayments[Discount_Fee])),

[Total],DESC)
```

Display bottom 4 CourseIDs based on discountfee

```
Top 4 CourseIDs =

TOPN(4,summarize(FactPayments,FactPayments[CourseID],"Total",sum(FactPayments[Di

scount_Fee]),"Average",AVERAGE(FactPayments[Discount_Fee])), [Total],ASC)
```



CourseID	Total	Average
Informatica-F	22500	11250
Azure-C	22500	11250
Informatica-N	21600	10800
Azure-N	21600	10800

Finout out top 4 discountfee rows by summarizing CourseID and ModeID

```
1 Tbl_new = TOPN(3,SUMMARIZE(FactPayments,FactPayments[CourseID],FactPayments[ModeID],"Total", sum(FactPayments[Discount_Fee]),"avg",AVERAGE(FactPayments
[Discount_Fee])),[Total], DESC)
```

CourseID	Total	avg	ModeID
Power BI-N	122400	12240	Classroom
MSBI-N	250200	25020	Online
MSBI-N	173700	13361.5384615385	Classroom

VAR.S

Returns the variance of a sample population

Syntax

VAR.S(<COLUMNNAME>)

Example

The following example shows the formula for a measure that calculates the variance of the SalesAmount_USD column from the InternetSales_USD for a sample population.

=VAR.S(InternetSales_USD[SalesAmount_USD])

VAR.P

Returns the variance of the entire population.

Syntax

VAR.P(<columnName>)

Example

The following example shows the formula for a measure that estimates the variance of the SalesAmount_USD column from the InternetSales_USD table, for the entire population.

=VAR.P(InternetSales_USD[SalesAmount_USD])

VARX.S

Returns the variance of a sample population.

Syntax

VARX.S(<table>, <expression>)

Example

The following example shows the formula for a calculated column that estimates the variance of the unit price per product for a sample population, when the formula is used in the Product table.

=VARX.S(InternetSales_USD,
InternetSales_USD[UnitPrice_USD] –
(InternetSales_USD[DiscountAmount_USD]/InternetSales_USD[
OrderQuantity]))

VARX.P

Returns the variance of the entire population.

Syntax

VARX.P(<table>, <expression>)

Example

The following example shows the formula for a calculated column that calculates the variance of the unit price per product, when the formula is used in the Product table

=VARX.P(InternetSales_USD, InternetSales_USD[UnitPrice_USD] –
(InternetSales_USD[DiscountAmount_USD]/InternetSales_USD[OrderQuantity]))

XIRR

Returns the internal rate of return for a schedule of cash flows that is not necessarily periodic.

Syntax

XIRR(<table>, <values>, <dates>, [guess])

Example

The following calculates the internal rate of return of the CashFlows table:

Rate of return := XIRR(CashFlows, [Payment], [Date])

DATE	PAYMENT
1/1/2014	-10000
3/1/2014	2750
10/30/2014	4250
2/15/2015	3250
4/1/2015	2750

Rate of return = 37.49%

XNPV

Returns the present value for a schedule of cash flows that is not necessarily periodic.

Syntax

XNPV(<table>, <values>, <dates>, <rate>)

Example

The following calculates the present value of the CashFlows table:

Present value := XNPV(CashFlows, [Payment], [Date], 0.09)

DATE	PAYMENT
1/1/2014	-10000
3/1/2014	2750
10/30/2014	4250
2/15/2015	3250
4/1/2015	2750

Present value = 2086.65