Important Dax formulas Every Analyst Must Know..!

krishna kumar

@Krishan kumar

01

=CALCULATE

- The **CALCULATE** function evaluates an expression in a **modified filter** context.

- It's essential for applying complex filters to your calculations. It can change the current row context by applying one or more filters to the data model.

Syntax with example

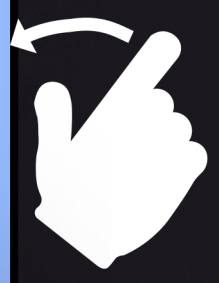
Syntax:

```
CALCULATE(<expression>, <filter1>, <filter2>, ...)
```

Example:

➡ **Suppose you have a table Sales with columns Amount and Region.**

To calculate the total sales amount for the 'North' region:



```
TotalSalesNorth = CALCULATE(
    SUM(Sales[Amount]),
    Sales[Region] = "North")
```

➡ **This calculates the sum of the Amount column where the Region is 'North'.**

02

=YEAR

- The **YEAR** function extracts the **year** as a four-digit number from a given date.
- It is useful for time-based calculations and analysis.

Syntax with example

Syntax:

```
YEAR(<date>)
```

Example:

➡ To get the year from the date '2023-05-23':

```
YearValue = YEAR("2023-05-23")
```

➡ Result: 2023

03

=MONTH

- The **MONTH** function extracts the month as **a number** from a given date.
- It's useful for grouping or filtering data by month.

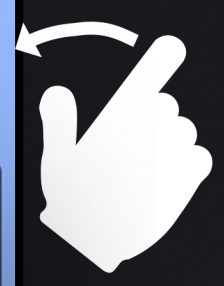
Syntax with example

Syntax:

```
MONTH(<date>)
```

Example:

➡ To get the month from the date '2023-05-23':



```
MonthValue = MONTH("2023-05-23")
```

➡ Result: 5

04

=DAY

- The **DAY** function extracts the day of the month as **a number** from a given date
- It helps in detailed date-based analysis.

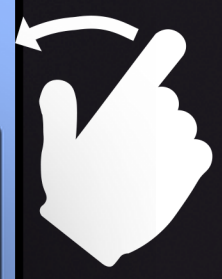
Syntax with example

Syntax:

```
DAY(<date>)
```

Example:

➡️ To get the day from the date '2023-05-23':



```
DayValue = DAY("2023-05-23")
```

➡️ Result: 23

05

=DAY

- The **TODAY** function returns the **current date**.
- It's useful for dynamic calculations based on the current date.

Syntax with example

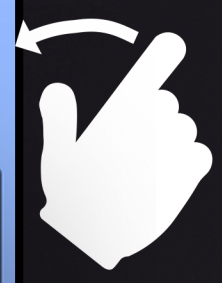
Syntax:

```
TODAY()
```

Example:

➡ To get today's date:

```
CurrentDate = TODAY()
```



➡ Result: 2024-05-23 (example)

06

=TOTALYTD

- The **TOTALYTD** function calculates the **year-to-date** total for a specific value, up to the current date or a specified date.
- It's widely used in financial and sales analysis.

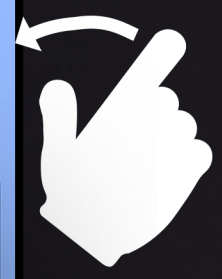
Syntax with example

Syntax:

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

Example:

➡️ To calculate year-to-date sales from a Sales table:



```
SalesYTD = TOTALYTD(
    SUM(Sales[Amount]),
    Sales[Date])
```

➡️ This calculates the total sales amount from the beginning of the year to the current date.

07

=FILTER

- The **FILTER** function returns a table that represents **a subset of another table** or **expression**, filtered by a specific condition.
- It's powerful for creating calculated tables and columns.

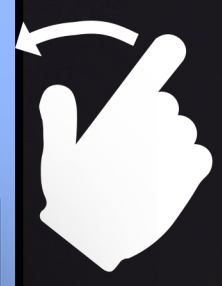
Syntax with example

Syntax:

```
FILTER(<table>, <expression>)
```

Example:

➡️ **To filter the Sales table for amounts greater than 1000:**



```
HighSales = FILTER(
    Sales,
    Sales[Amount] > 1000)
```

➡️ **This creates a table with rows where the Amount column is greater than 1000.**

08

=SWITCH

- The SWITCH function evaluates an expression against a list of values and returns one of multiple possible result expressions.

- It's similar to a series of nested IF statements but more readable and efficient.

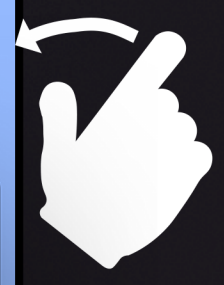
Syntax with example

Syntax:

```
SWITCH(<expression>, <value1>, <result1>, <value2>,
<result2>, ..., <else_result>)
```

Example:

➡️ **To classify sales regions as 'High', 'Medium', or 'Low':**



```
RegionCategory = SWITCH(Sales[Region],
                        "North", "High",
                        "South", "Medium",
                        "West", "Low",
                        "Unknown")
```

➡️ **This assigns a category based on the value of the Region column.**

09

=CONCATENATEX

- The **CONCATENATEX** function concatenates the result of an expression evaluated for **each row of a table**, optionally using a delimiter.

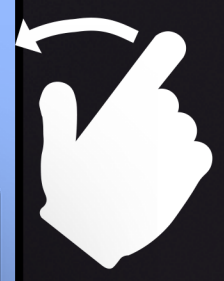
- It's useful for creating summary text strings.

Syntax:

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

Example:

➡ **To concatenate product names separated by a comma:**



```
ProductList = CONCATENATEX(
    Products,
    Products[ProductName], ", ")
```

➡ **This creates a single string with all product names separated by commas.**

10

=RANKX

- The **RANKX** function returns the rank of a value in a list of values in a specified order. .

- It's useful for creating rank metrics, such as ranking salespeople or products.

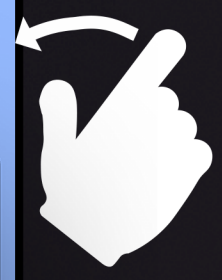
Syntax with example

Syntax:

```
RANKX(<table>, <expression>, [<value>, <order>, <ties>])
```

Example:

➡ **To rank products based on their sales amount:**



```
ProductRank = RANKX(
    ALL(Products),
    SUM(Products[SalesAmount]))
```

➡ **This assigns a rank to each product based on the total sales amount.**

Found this Useful ?

Would you mind showing your support by giving it a like?