



Belgareg mohamed amine  
business intelligence developer



Power BI

## The Ultimate Guide to DAX Formulas

# Top 10 DAX Formulas You Absolutely Need to Know

## What it contains

1. Introduction	1
2. Part 1: SUM() – The Essentials of Addition	2
3. Part 2: CALCULATE() – The Magic of Conditional Calculations	3
4. Part 3: RELATED() – Linking Tables	4
5. Part 4: DISTINCTCOUNT() – Unique Counting	5
6. Part 5: AVERAGE() – The Key Measure of Central Tendency	5
7. Part 6: FILTER() – The Power of Selection	6
8. Part 7: ALL() – Removing Filters for Global Analysis	7
9. Part 8: MAX() and MIN() – Identifying Extremes	8
10. Part 9: CONCATENATEX() – Merging Texts	9
11. Part 10: DATEADD() – Manipulating Dates	10
12. Conclusion	11

## Introduction



Even if you're proficient with Excel, navigating the world of DAX in Power BI might leave you feeling a bit lost. The approach seems familiar, but you find yourself treading water.

DAX, short for Data Analysis Expressions, is the backbone of Power BI, transforming raw data into valuable insights. But with so many formulas at your disposal, where do you begin?

Fear not, we're here to guide you. This book introduces you to the 10 essential DAX formulas that every data enthusiast should know. They are simple, powerful, and will revolutionize how you work with data.

**TOP 10**

## The Ultimate Guide to **DAX** Formulas



### Part 1

## *SUM(): The Foundation of Aggregation*

The **SUM()** function is one of the most fundamental and widely used DAX formulas. It allows you to quickly add up the values in a column, providing an instant overview of the total sum.

For example, if you have a sales table, using **SUM(Sales[Amount])** will give you the total sales amount.

### Practical Applications:

- Calculating total sales by month, quarter, or year.
- Adding up the hours worked by employees for project management.
- Aggregating expenses or costs across different departments for budgeting.

### Part 2

# CALCULATE(): The Power of Conditional Calculation

**CALCULATE()** is DAX's magic wand, offering unparalleled flexibility in conditional calculations. It allows you to apply filters to any measure or calculation, changing the context of your analysis to meet specific criteria.

For instance, **CALCULATE(SUM(Sales[Amount]), Sales[Category] = "Electronics")** sums sales only for the "Electronics" category.

The diagram shows a table of sales data with columns Date, Amount, and Category. The table has six rows. The first row (Date: 2024-01-01, Amount: 100, Category: Electronics) is highlighted with a red background. The last row (Date: 2024-01-06, Amount: 160, Category: Furniture) is also highlighted with a red background. A yellow box at the bottom right contains the formula  $100 + 200 + 180 = 380$ .

**CALCULATE(SUM(Sales[Amount]), Sales[Category] = "Electronics")**

**SUM(Sales[Amount]):** This part of the formula calculates the total sum of the Amount column in the Sales table.

**Sales[Category] = "Electronics":** This filter restricts the data to rows where the Category column equals "Electronics".

@BELGAREG.MOHAMED.AMINE

## Practical Applications:

- Calculating sales in a specific region while excluding certain product categories.
- Analyzing sales trends by applying filters for specific time periods.
- Comparing team performances by filtering different departments.

## Part 3

# RELATED(): Bridging the Gap Between Tables

In relational data models, **RELATED()** is essential for leveraging the power of relationships between tables. This function allows you to access values from another related table, enriching your analysis with complementary data.

For example, if you have a sales table and a products table linked by a product ID, **RELATED(Products[Name])** allows you to display the product name alongside each sale.

**RELATED(Products[Name])**

The diagram illustrates the relationship between three tables: Sales, Products, and a resulting table. A brace on the right side groups the Sales and Products tables, indicating they are being joined. The Sales table has columns SaleID, ProductID, Amount, and Date. The Products table has columns ProductID, Name, and Category. The resulting table, labeled 'Table: New Sales' on the right, combines these into five rows, each with a SaleID, ProductID, Amount, Date, and a corresponding ProductName from the Products table. The Sales table has rows for SaleID 1-5 with ProductID 101, 102, 101, 103, and 101 respectively. The Products table has rows for ProductID 101 (Laptop), 102 (Desk), 101 (Laptop), 103 (Chair), and 101 (Laptop). The resulting table shows the original sales data with the added ProductName column.

Table: Sales		Table: New Sales		
SaleID	ProductID	Amount	Date	ProductName
1	101	100	2024-01-01	Laptop
2	102	150	2024-01-02	Desk
3	101	200	2024-01-03	Laptop
4	103	120	2024-01-04	Chair
5	101	180	2024-01-05	Laptop

  

Table: Products		
ProductID	Name	Category
101	Laptop	Electronics
102	Desk	Furniture
103	Chair	Furniture

**RELATED(Products[Name]):** This function retrieves the value from the Name column in the Products table based on the relationship defined between the Sales and Products tables. It requires that there is a relationship between these tables based on a common field, such as ProductID.

@BELGAREG.MOHAMED.AMINE

## Practical Applications:

- Combining product information, such as name or category, with sales data for detailed reports.
- Linking employee information to project data to analyze performance by employee.
- Cross-referencing supplier cost data with order information for expense analysis.

## Part 4

# **DISTINCTCOUNT()** : Counting Unique Values

**DISTINCTCOUNT()** is crucial for counting the number of unique occurrences in a column. For example, using **DISTINCTCOUNT(***Customers[CustomerID]***)** will give you the total number of unique customers in your database, which is especially useful for analyzing data diversity or uniqueness.

## **DISTINCTCOUNT(***Customers[CustomerID]***)**

**Objective :** You want to count the number of unique customers in the Customers table.

**Table: Customers**

CustomerID	Name	City
C001	John Smith	New York
C002	Jane Doe	Los Angeles
C003	Mary Johnson	Chicago
C004	James Brown	Houston
C001	John Smith	New York

**Result**

- The result of this formula would be 4 because there are four unique CustomerIDs: C001, C002, C003, and C004
- Even though C001 (John Smith) appears twice in the table, DISTINCTCOUNT counts it only once because it is the same CustomerID.

@BELGAREG.MOHAMED.AMINE

## Practical Applications:

- Measuring the number of unique customers who made purchases.
- Counting the number of unique products sold during a given period.
- Determining the number of unique days on which transactions were made.

## Part 5

# **AVERAGE()** : Calculating the Mean

The **AVERAGE()** function calculates the mean of values in a column, offering valuable insights into average performance or central tendencies. Using **AVERAGE(***Sales[Amount]***)** provides the

average sales amount, a critical tool for evaluating overall performance and identifying deviations from the norm.

# AVERAGE(Sales[Amount])

## Objective

You want to calculate the average sales amount across all transactions.

Table: Sales

SaleID	ProductID	CustomerID	Amount	Date
1	101	C001	100	2024-01-01
2	102	C002	150	2024-01-02
3	101	C001	200	2024-01-03
4	103	C003	250	2024-01-04
5	102	C004	150	2024-01-05

## Explanation

The AVERAGE function calculates the mean value of all the numbers in the Amount column of the Sales table.

## Calculation Steps:

Sum of Amounts:  $100 + 150 + 200 + 250 + 150 = 850$

Number of entries: 5

Average:  $850 / 5 = 170$

## Result

The result of the **AVERAGE(Sales[Amount])** formula is **170**.

@BELGAREG.MOHAMED.AMINE

## Practical Applications:

- Assessing the average spending per customer.
- Calculating the average salary of employees in a company.
- Determining the average response time to customer inquiries.

## Part 6

# FILTER(): The Power of Selection

**FILTER()** is a powerful function that creates subsets of data based on specific criteria. For instance, **FILTER(AllCustomers, Customers[Age] > 18)** generates a table containing only adult customers. This function is indispensable for refining analyses and reports according to precise conditions.

## FILTER(**AllCustomers**, Customers[Age] > 18)

### Objective

You want to create a filtered table that includes only customers who are older than 18.

Table: Customers

CustomerID	Name	Age	Gender
C001	John Doe	25	Male
C002	Jane Smith	17	Female
C003	Bob Johnson	34	Male
C004	Alice White	16	Female
C005	Tom Brown	40	Male

### Filtered Result

CustomerID	Name	Age	Gender
C001	John Doe	25	Male
C003	Bob Johnson	34	Male
C005	Tom Brown	40	Male

### Explanation

The FILTER function checks each row in the Customers table to see if the value in the Age column is greater than 18. It returns a table containing only the rows where this condition is TRUE.

@BELGAREG.MOHAMED.AMINE

## Practical Applications:

- Selecting sales made in certain regions for regional analysis.
- Filtering project data to include only those exceeding a certain budget.
- Excluding outlier data or errors for more accurate analysis.

## Part 7

# ALL(): Removing Filters for Global Analysis

**ALL()** is a powerful function that removes all filters applied in the calculation context, allowing calculations on the entire dataset, regardless of the filters currently applied. For example, to calculate total sales regardless of user-applied filters, you can use **CALCULATE(SUM(Sales[Amount]), ALL(Sales))**.

## CALCULATE(SUM(Sales[Amount])), ALL(Sales))

### Objective

You want to calculate the total sales amount without considering any filters that might be applied to the table.

Table: Sales

SalesID	Category	Amount	Region
1	Electronics	1000	East
2	Furniture	1500	West
3	Electronics	2000	East
4	Furniture	1200	West
5	Clothing	800	East

### Explanation

- If you were using a filter on the Category column (e.g., filtering only "Electronics"), normally the SUM(Sales[Amount]) would only sum the amounts for "Electronics".
- However, by using ALL(Sales), the CALCULATE function ignores any filters and sums the Amount column across the entire Sales table.

### Result

The formula would return the sum of all amounts:

$$1000 + 1500 + 2000 + 1200 + 800 = 6500.$$

@BELGAREG.MOHAMED.AMINE

## Practical Applications:

- Comparing a specific product's sales to the total sales.
- Calculating a product's market share within total sales.
- Evaluating performance for a period against the entire year.

## Part 8

# MAX() and MIN(): Identifying Extremes

**MAX()** and **MIN()** are essential functions for identifying the maximum and minimum values in a column. For instance, **MAX(Sales[SaleDate])** and **MIN(Sales[SaleDate])** allow you to find the most recent and oldest transaction dates. These functions are crucial for analyzing data ranges and extremes.

### Example Scenario

Let's say you're creating a report to analyze the sales performance over time, and you want to show the date range of the sales data.

## MAX(Sales[SaleDate])

Table: Sales

SaleID	SaleDate	Amount
1	2024-01-15	1000
2	2024-03-20	1500
3	2024-05-10	2000
4	2024-07-05	1200
5	2024-02-25	800

This function returns the most recent date in the SaleDate column of the Sales table.

This formula returns **2024-07-05** because it is the latest date in the SaleDate column.

## MIN(Sales[SaleDate])

This function returns the earliest date in the SaleDate column of the Sales table.

This formula returns **2024-01-15** because it is the earliest date in the SaleDate column.

@BELGAREG.MOHAMED.AMINE

## Practical Applications:

- Determining the highest and lowest prices among sold products.
- Identifying the youngest and oldest ages among event participants.
- Finding the shortest and longest call durations in a call center.

## Part 9

# CONCATENATEX(): Merging Text

**CONCATENATEX()** allows you to combine text values from multiple rows into a single string, separated by a delimiter of your choice. For example, **CONCATENATEX(Customers, Customers[Name], ", ")** assembles all your customer names into one text string, separated by commas. This is ideal for creating lists or readable summaries.

### Objective:

Create a summary of all customer names in a single text field for easy reporting and analysis.

## CONCATENATEX(**Customers**, **Customers[Name]**, ", ")

Table: Customers

CustomerID	Name
1	John Smith
2	Jane Doe
3	Alice Brown
4	Bob White

### Result:

The output will be a single string that lists all customer names, such as:

"John Smith, Jane Doe, Alice Brown, Bob White"

@BELGAREG.MOHAMED.AMINE

### Practical Applications:

- Generating a list of products sold for each order.
- Creating custom labels for charts by combining multiple attributes.
- Summarizing customer activities or preferences in a single field for quick analysis.

## Part 10

## DATEADD(): Manipulating Dates

**DATEADD()** is an essential function for working with temporal data, allowing you to add or subtract a specific number of time units (**days**, **months**, **years**) from a date. For instance, **DATEADD(Sales[SaleDate], -1, YEAR)** retrieves the date that corresponds to one year before each sale. This function is crucial for performing comparative time analysis, such as year-over-year sales growth.

**Objective :** Calculate the total sales for March 2024 and compare it to the total sales for March 2023.

Table: Sales

SaleID	SaleDate	Amount
1	2024-03-01	200
2	2024-03-15	300
3	2024-04-01	250
4	2023-03-01	180
5	2023-03-15	320
6	2023-04-01	220

Create a measure to calculate total sales for March 2024:

```
TotalSalesMarch2024 =  
CALCULATE(  
    SUM(Sales[Amount]),  
    MONTH(Sales[SaleDate]) = 3,  
    YEAR(Sales[SaleDate]) = 2024  
)
```

Total Sales for  
March 2024: 200  
+ 300 = 500

Create a measure to calculate total sales for March 2023:

```
TotalSalesMarch2023 =  
CALCULATE(  
    SUM(Sales[Amount]),  
    MONTH(Sales[SaleDate]) = 3,  
    YEAR(Sales[SaleDate]) = 2023  
)
```

Total Sales for  
March 2023: 180  
+ 320 = 500

Create a measure to compare sales between March 2024 and March 2023:

SalesComparison = [TotalSalesMarch2024] - [TotalSalesMarch2023] = 500 - 500 = 0

=> In this case, the sales for March 2024 are the same as for March 2023, resulting in a comparison value of 0.

@BELGAREG.MOHAMED.AMINE

## Practical Applications:

- Calculating sales growth or other key indicators over different periods.
- Identifying seasonal trends by comparing data from the same month over several years.
- Estimating deadlines or important anniversaries for customers or projects.

## Conclusion

By mastering these 10 essential DAX formulas, you now have the tools to unlock the full potential of your data in Power BI. Each of these functions plays a crucial role in transforming raw data into actionable insights, allowing you to navigate through complex analyses with ease and precision.

From simple data aggregation with SUM() to advanced date manipulation with DATEADD(), these formulas form the foundation of powerful and insightful data analysis. Remember, as with any tool, practice is key to refining your mastery of DAX.

Don't hesitate to experiment with these functions, combine them creatively, and explore how they can meet your specific analytical needs. Every dataset is unique, and with these formulas at your disposal, you're well-equipped to tackle almost any analytical challenge your data may present.

We hope this guide has not only familiarized you with the essential DAX formulas but also inspired you to dive deeper into data analysis with Power BI. The possibilities are vast, and with a little practice and exploration, you'll soon uncover even deeper and more meaningful insights in your own data, truly becoming a skilled data analyst!

## Contact Us

For questions, clarifications, or feedback, reach out to us:

belgaregmoahamedamine@outlook.fr

+216 56 490 416

Portfolio