INSIGHTSMASTERY ACADEMY

DAX MASTERY

STUDENT HANDBOOK

From Absolute Beginner to Advanced

A Complete Guided Learning Path for Power BI DAX

COUNT • COUNTA • COUNTX • COUNTROWS

SUM • SUMX • AVERAGE • AVERAGEX

CALCULATE • FILTER CONTEXT • ROW CONTEXT

ALL • ALLEXCEPT • ALLSELECTED • REMOVEFILTERS • KEEPFILTERS

FILTER • VALUES • DISTINCT • CONTEXT TRANSITION

Professional Training Edition

Designed for Classroom & Self-Learning

www.insightsmastry.in

1. How to Use This Handbook

This handbook is written as if an experienced teacher is sitting beside you and explaining every concept slowly. You do not need any programming background. We start from absolute zero.

Recommended Learning Path

1. Read each chapter in order. Do not skip the fundamentals.
2. Type every DAX formula yourself in Power BI Desktop.
3. After each example, change the numbers and observe what happens.
4. Complete the practice exercises at the end of each major section.
5. Build the Mini Project only after finishing the core chapters.
6. Use the Cheat Sheet for quick revision.

Teaching Pattern Used Throughout

For every important function we follow this exact pattern:

> *WHAT → WHY → SYNTAX → EVERY WORD → SIMPLE EXAMPLE → RESULT → WHY RESULT → REAL-WORLD EXAMPLE → COMMON MISTAKE → PRACTICE*

2. DAX Roadmap

Here is the complete journey you will take. Each stage builds on the previous one.

Stage	Focus	Key Concepts
1. Foundations	What is DAX?	Tables, Columns, Rows, Measures, Calculated Columns
2. Context Basics	How DAX sees data	Filter Context, Row Context
3. Aggregation	Simple totals	COUNT, COUNTA, COUNTROWS, SUM, AVERAGE
4. Iterators	Row-by-row work	COUNTX, SUMX, AVERAGEX
5. CALCULATE	Changing context	CALCULATE + simple filters
6. Advanced Filters	Precise control	FILTER, ALL, REMOVEFILTERS, ALLEXCEPT, ALLSELECTED, KEEPFILTERS
7. Distinct Values	Unique lists	VALUES, DISTINCT
8. Deep Concepts	How engines work	Context Transition
9. Mastery	Real projects	Comparisons, Debugging, Dashboard Project

Important Promise

By the end of this handbook you will not only know which formula to write — you will understand WHY you are writing it, what filter context is active, whether you are in row context or filter context, and how Power BI evaluates the expression.

3. DAX Fundamentals

3.1 What is DAX?

DAX stands for **Data Analysis Expressions**.

It is a formula language used to create custom calculations in Power BI, Power Pivot (Excel), and Analysis Services.

Where is DAX used?

- Power BI Desktop and Power BI Service
- Excel Power Pivot
- SQL Server Analysis Services (SSAS) Tabular models
- Azure Analysis Services

What DAX is NOT

DAX is not a programming language like Python or C#. You do not write loops or create applications with it. DAX is a calculation language designed specifically for business intelligence and analytical reporting.

> **Simple Analogy**
>
> Think of Excel formulas. In Excel you write =SUM(A1:A10). In Power BI you write similar ideas, but the language is more powerful because it understands tables, relationships, and filters automatically.

3.2 Tables, Rows and Columns

Before we write any DAX, we must clearly understand the building blocks.

What is a Table?

A table is a collection of related data arranged in rows and columns. In our handbook we will mostly use a table called **Sales**.

What is a Column?

A column stores one type of information for every row. Examples: Customer, Product, SalesAmount, Region.

What is a Row?

A row is one complete record. In the Sales table, each row is one sales transaction.

> **Classroom Analogy**

Imagine a classroom with 10 students.
• Each student is one **row**.
• "Age" is a **column**.
• "Name" is another column.
• The entire list of students is the **table**.

If we want to know how many students have an age recorded, we can use COUNT.

3.3 Our Standard Sample Dataset

Throughout this handbook we will use one consistent Sales table. Memorize its structure. All examples refer to this data.

SalesID	Date	Customer	Product	Category	Region	Qty	SalesAmt	Cost	Profit	Salesperson
1	01-Jan-26	Amit	Laptop	Electronics	North	2	100000	80000	20000	Ravi
2	02-Jan-26	Neha	Mouse	Electronics	South	5	5000	3000	2000	Priya
3	03-Jan-26	Rahul	Keyboard	Electronics	North	3	6000	3500	2500	Ravi
4	04-Jan-26	Amit	Monitor	Electronics	West	2	30000	22000	8000	Amit
5	05-Jan-26	Priya	Chair	Furniture	South	4	20000	14000	6000	Priya
6	06-Jan-26	Rahul	Desk	Furniture	North	1	15000	10000	5000	Ravi
7	07-Jan-26	Neha	Laptop	Electronics	South	1	50000	40000	10000	Priya
8	08-Jan-26	Amit	Chair	Furniture	West	2	10000	7000	3000	Amit
9	09-Jan-26	Rahul	Mouse	Electronics	North	10	10000	6000	4000	Ravi
10	10-Jan-26	Priya	Desk	Furniture	South	2	30000	20000	10000	Priya

Note: We may introduce blank values later when comparing COUNT vs COUNTA.

Quick Facts from the Sample Data

• Total rows = 10
• Regions: North, South, West
• Categories: Electronics, Furniture
• Salespersons: Ravi, Priya, Amit
• Total SalesAmount = 100000+5000+6000+30000+20000+15000+50000+10000+10000+30000 = 276000

4. Measures vs Calculated Columns

This is one of the most important distinctions in Power BI. Many beginners confuse the two.

4.1 Calculated Column

A calculated column is stored in the table. It is calculated once for every row when the data is refreshed.

Example:

```
Profit =

Sales[SalesAmount] - Sales[Cost]
```

Read it like English:

«Create a new column called Profit. For each row, take SalesAmount from this row and subtract Cost from this row.»

Line-by-line explanation:

Profit — This is the name of the new column we are creating.

= — Means we are defining an expression.

Sales[SalesAmount] — Means the SalesAmount column from the Sales table (current row).

- — Subtract.

Sales[Cost] — Means the Cost column from the Sales table (current row).

Because this runs row by row, we say it has **row context**. Power BI knows which row it is currently looking at.

4.2 Measure

A measure is a calculation that is evaluated at query time based on the filters currently applied in the report (the filter context). Measures are not stored as columns; they are calculated on demand.

Example:

```
Total Sales =

SUM(Sales[SalesAmount])
```

Read it like English:

«Create a measure called Total Sales. Add up all the values in the SalesAmount column (considering current filters).»

Key Differences

Aspect	Calculated Column	Measure
When calculated	At data refresh, for every row	At query time, based on filters
Context	Row context	Filter context
Storage	Stored in the model (uses memory)	Not stored; calculated on demand
Typical use	New attributes, flags, ratios per row	Totals, averages, percentages, KPIs

Aspect	Calculated Column	Measure
Example	Profit = SalesAmount – Cost	Total Sales = SUM(SalesAmount)

Teacher Tip

If the calculation needs to change when the user clicks a slicer or filters a visual → use a **Measure**.
If the calculation is a fixed attribute of each row (like Profit per transaction) → a **Calculated Column** is often fine.

5. Filter Context

Filter context is the most important concept in DAX. If you master only one idea, master this.

5.1 Simple Meaning

Filter context means: **"The filters currently affecting your calculation."**

> **Window Analogy**
>
> Think of filter context like looking through a window. You only see the rows that the current filters allow. When you change a slicer, the window changes — different rows become visible, and measures recalculate automatically.

5.2 How Filter Context is Created

Filter context comes from many places:

- Slicers on the report page
- Page-level filters
- Visual-level filters
- Rows and columns of a matrix or table visual
- Cross-filtering from other visuals
- Relationship filters from related tables
- Filters applied inside CALCULATE

5.3 Demonstration

Base measure:

```
Total Sales =

SUM(Sales[SalesAmount])
```

If no filters are applied, this returns the sum of all 10 rows = 276,000.

If a slicer or matrix row has **Region = North**, the filter context now contains only the North rows (SalesID 1, 3, 6, 9). The same measure returns only the North total.

Visual flow:

```
All Sales (10 rows)
      ↓
Region = North filter applied
      ↓
Only North rows remain (4 rows)
      ↓
```

```
SUM(Sales[SalesAmount]) on those rows
      ↓
Result = North Sales
```

5.4 In a Matrix Visual

When you put Region on rows and Total Sales on values, Power BI automatically creates a different filter context for each row:

- For the North row → filter Region = "North" → calculate Total Sales
- For the South row → filter Region = "South" → calculate Total Sales
- For the West row → filter Region = "West" → calculate Total Sales

You do not write three different measures. One measure + the visual's filter context does the work.

6. Row Context

Row context means Power BI knows which single row it is currently looking at.

6.1 Where Row Context Appears

- Inside calculated columns (always)
- Inside iterator functions (SUMX, AVERAGEX, COUNTX, FILTER, etc.)

6.2 Classic Example — Calculated Column

```
Profit =

Sales[SalesAmount] - Sales[Cost]
```

When this formula runs, Power BI goes to row 1, takes SalesAmount of row 1, subtracts Cost of row 1, stores the result. Then moves to row 2 and repeats. That "current row" awareness is row context.

Visual walkthrough:

Row	SalesAmount	Cost	Calculation	Profit
1	100000	80000	100000 – 80000	20000
2	5000	3000	5000 – 3000	2000
3	6000	3500	6000 – 3500	2500
...	...	...	...	...

Key Distinction

Row context = "I am looking at one specific row right now."
Filter context = "These are the rows currently visible because of filters."

They are different. Later we will learn how CALCULATE can turn row context into filter context (context transition).

7. COUNT Family

7.1 COUNT

1. What is it?

COUNT counts the number of numeric (non-blank) values in a column.

2. Why do we need it?

We often need to know how many numbers exist in a column — for example, how many quantities were recorded.

3. Syntax

```
COUNT()
```

4. Syntax breakdown

COUNT — the function name
(...) — parentheses hold the argument
<column> — a single column that contains numbers

5. Simplest example

```
Total Quantities Recorded =

COUNT(Sales[Quantity])
```

6. Line-by-line

Total Quantities Recorded = — name of the measure
COUNT(— start the COUNT function
Sales[Quantity] — look at the Quantity column
) — end the function

7. Result

In our sample data every row has a Quantity, so the result is 10.

8. Why?

COUNT only looks at numeric values and ignores blanks. All 10 rows have a number, so it returns 10.

Important notes about COUNT

- Works only on numeric columns (or dates).
- Ignores blank values.
- Does not count text values.

Another common example:

```
Order Count =

COUNT(Sales[SalesID])
```

Because SalesID is numeric and present on every row, this also returns 10.

7.2 COUNTA

1. What is it?

COUNTA counts the number of non-blank values in a column (numbers or text).

2. Why do we need it?

When the column contains text (Customer names, Product names, Regions) we cannot use COUNT. We use COUNTA.

3. Syntax

```
COUNTA()
```

4. Example

```
Customer Entries =

COUNTA(Sales[Customer])
```

Result = 10 (all rows have a customer name).

COUNT vs COUNTA

Function	Counts	Ignores Blank?	Typical Use
COUNT	Numeric values only	Yes	Numbers, quantities, IDs
COUNTA	Any non-blank value (text or number)	Yes	Text columns + mixed
COUNTX	Non-blank results of an expression	Yes	Calculated values row-by-row
COUNTROWS	Number of rows in a table	N/A	Counting rows

7.3 COUNTX

Functions ending with **X** usually iterate through a table row by row.

1. What is it?

COUNTX goes through each row of a table, evaluates an expression, and counts how many times the expression returns a non-blank value.

2. Syntax

```
COUNTX(

    ,



)
```

3. Simple example (same as COUNT)

```
Order Count =

COUNTX(

    Sales,

    Sales[SalesID]

)
```

Read it like English:

«Go through every row of the Sales table. For each row evaluate Sales[SalesID]. Count how many non-blank results you get.»

4. More useful example — conditional counting

```
Valid Orders =

COUNTX(

    Sales,

    IF(

        Sales[SalesAmount] > 0,

        Sales[SalesID]

    )

)
```

Teacher explanation (row by row):

1. Go to row 1. Check SalesAmount. Is it greater than 0? Yes → return SalesID.
2. COUNTX counts this returned value.
3. Move to row 2. Repeat the same check.
4. Continue until the last row.
5. Return the total count of non-blank results.

7.4 COUNTROWS

1. What is it?

COUNTROWS simply counts the number of rows in a table.

2. Syntax

```
COUNTROWS()
```

3. Simplest example

```
Total Orders =

COUNTROWS(Sales)
```

Result = 10

4. With a filter (preview of FILTER)

```
North Orders =

COUNTROWS(

    FILTER(

        Sales,

        Sales[Region] = "North"

    )

)
```

FILTER returns only the rows where Region is North. COUNTROWS then counts those rows. We will study FILTER in detail later.

8. SUM

1. What is it?

SUM adds up all the numeric values in a column (respecting the current filter context).

2. Why do we need it?

Almost every report needs totals: Total Sales, Total Cost, Total Quantity, etc.

3. Syntax

```
SUM()
```

4. Examples

```
Total Sales =

SUM(Sales[SalesAmount])
```

```
Total Cost =

SUM(Sales[Cost])
```

```
Total Profit =

SUM(Sales[Profit])
```

With no filters, Total Sales = 276,000.

Important points

- The column must be numeric.
- Blank values are treated as zero (they do not affect the sum).
- The result changes automatically when filter context changes (slicers, matrix rows, etc.).

9. SUMX — The Major Iterator

This is one of the most important functions in DAX. Take your time here.

Core idea

SUM adds an existing column.
SUMX calculates something for every row and then adds the results.

1. What is it?

SUMX iterates over a table, evaluates an expression for each row, and returns the sum of those results.

2. Syntax

```
SUMX(

    ,



)
```

3. Classic example

```
Total Revenue =

SUMX(

    Sales,

    Sales[Quantity] * Sales[SalesAmount]

)
```

Read it like English:

«Go through every row of Sales. For each row multiply Quantity by SalesAmount. Then add all those products together.»

4. Mini calculation table (simplified)

Quantity	Price	Calculation
2	100	2 × 100 = 200
3	50	3 × 50 = 150
4	25	4 × 25 = 100
	Total	200 + 150 + 100 = 450

Why can't we just write SUM(Sales[Quantity] * Sales[SalesAmount])? Because SUM only accepts a column, not an expression. SUMX exists exactly for this reason.

5. Real example with our data

In our sample, Quantity × SalesAmount for each row is not the same as simply summing SalesAmount. If you need line-level multiplication, you must use SUMX.

When to choose SUM vs SUMX

• Use **SUM** when you only need to add an existing numeric column.
• Use **SUMX** when you need to calculate an expression (multiply, subtract, IF logic, etc.) for each row and then sum.

10. AVERAGE

AVERAGE returns the arithmetic mean of all non-blank numeric values in a column.

```
Average Sales =

AVERAGE(Sales[SalesAmount])
```

Conceptually it is similar to SUM / COUNT (of non-blank values).

Blanks are ignored. Filters change the set of values that participate in the average.

11. AVERAGEX

AVERAGEX iterates a table, evaluates an expression per row, and returns the average of the non-blank results.

```
Average Order Value =

AVERAGEX(

    Sales,

    Sales[SalesAmount]

)
```

More meaningful example:

```
Average Line Profit =

AVERAGEX(

    Sales,

    Sales[SalesAmount] - Sales[Cost]

)
```

This calculates profit for every row, then averages those profits.

12. CALCULATE — The Heart of DAX

If you master CALCULATE, you master DAX. This is the largest and most important chapter.

1. What is it?

CALCULATE changes the filter context in which an expression is evaluated.

2. Syntax

```
CALCULATE(

    ,

    ,

    ,

    ...

)
```

3. Simple starting point

```
Total Sales =

SUM(Sales[SalesAmount])
```

Now force the Region to North:

```
North Sales =

CALCULATE(

    [Total Sales],

    Sales[Region] = "North"

)
```

Read it like English:

«Create a measure called North Sales. Calculate Total Sales, but only for the rows where Region is North.»

4. What happens step by step

1. CALCULATE receives the expression [Total Sales].
2. It applies the filter Sales[Region] = "North".
3. The original filter context is modified (Region is now forced to North).
4. [Total Sales] is evaluated under this new filter context.
5. The result is returned.

5. More simple filter examples

```
Electronics Sales =
```

```
CALCULATE(
    [Total Sales],
    Sales[Category] = "Electronics"
)
```

```
Ravi Sales =
CALCULATE(
    [Total Sales],
    Sales[Salesperson] = "Ravi"
)
```

Multiple filters (they work together as AND):

```
North Electronics Sales =
CALCULATE(
    [Total Sales],
    Sales[Region] = "North",
    Sales[Category] = "Electronics"
)
```

Both conditions must be true at the same time.

13. CALCULATE + Aggregation Functions

13.1 CALCULATE + COUNT / COUNTA / COUNTROWS

```
North Order Count =
CALCULATE(
    COUNT(Sales[SalesID]),
    Sales[Region] = "North"
)
```

```
North Customer Count =
CALCULATE(
    COUNTA(Sales[Customer]),
    Sales[Region] = "North"
)
```

```
North Rows =
CALCULATE(
    COUNTROWS(Sales),
    Sales[Region] = "North"
)
```

13.2 CALCULATE + SUM

```
North Sales =
CALCULATE(
    [Total Sales],
    Sales[Region] = "North"
)
```

```
South Sales =
CALCULATE(
    [Total Sales],
    Sales[Region] = "South"
)
```

```
Electronics Sales =
CALCULATE(
```

```
    [Total Sales],
    Sales[Category] = "Electronics"
)
```

```
Furniture Sales =
CALCULATE(
    [Total Sales],
    Sales[Category] = "Furniture"
)
```

13.3 CALCULATE + AVERAGE

```
North Average Sales =
CALCULATE(
    AVERAGE(Sales[SalesAmount]),
    Sales[Region] = "North"
)
```

```
Electronics Average Sales =
CALCULATE(
    AVERAGE(Sales[SalesAmount]),
    Sales[Category] = "Electronics"
)
```

13.4 CALCULATE + X Functions

```
North Revenue =
CALCULATE(
    SUMX(
        Sales,
        Sales[Quantity] * Sales[SalesAmount]
    ),
    Sales[Region] = "North"
)
```

Conceptual order:

```
CALCULATE
```

```
   ↓
changes filter context to North
   ↓
only North rows remain
   ↓
SUMX iterates those North rows
   ↓
calculates Quantity × SalesAmount per row
   ↓
sums the results
```

```
North Average Profit =
CALCULATE(
    AVERAGEX(
        Sales,
        Sales[SalesAmount] - Sales[Cost]
    ),
    Sales[Region] = "North"
)
```

14. FILTER Function

FILTER creates a table that contains only the rows that satisfy a condition.

Syntax

```
FILTER(

    ,


)
```

Example inside CALCULATE

```
High Value Sales =

CALCULATE(

    [Total Sales],

    FILTER(

        Sales,

        Sales[SalesAmount] > 30000

    )

)
```

Read it like English:

«Calculate Total Sales, but only keep the rows where SalesAmount is greater than 30,000.»

Row-by-row thinking:

- Row 1: 100000 > 30000? Yes → keep
- Row 2: 5000 > 30000? No → discard
- Row 3: 6000 > 30000? No → discard
- Row 4: 30000 > 30000? No → discard
- Row 7: 50000 > 30000? Yes → keep
- ... and so on

Multiple conditions with && (AND) and || (OR)

```
North High Value Sales =

CALCULATE(

    [Total Sales],

    FILTER(

        Sales,

        Sales[Region] = "North"
```

```
            && Sales[SalesAmount] > 30000
    )
)
```

&& means AND. Both conditions must be true.
|| means OR. At least one condition must be true.

15. ALL Family — Removing Filters

15.1 ALL

ALL removes filters from a table or from specific columns.

```
All Region Sales =

CALCULATE(

    [Total Sales],

    ALL(Sales[Region])

)
```

«Ignore whatever Region filter is currently active. Calculate Total Sales as if no Region filter exists.»

Classic use — Percentage of Total:

```
Sales % of Total =

DIVIDE(

    [Total Sales],

    CALCULATE(

        [Total Sales],

        ALL(Sales[Region])

    )

)
```

Numerator respects the current Region filter. Denominator removes the Region filter so we get the grand total. DIVIDE safely handles division by zero.

15.2 REMOVEFILTERS

REMOVEFILTERS is a clearer way to express the same idea as ALL when you only want to remove filters.

```
All Region Sales =

CALCULATE(

    [Total Sales],

    REMOVEFILTERS(Sales[Region])

)
```

In modern DAX, many authors prefer REMOVEFILTERS for readability when the goal is simply to clear filters.

15.3 ALLEXCEPT

ALLEXCEPT removes all filters from a table except the ones you want to keep.

```
Sales by Category Only =

CALCULATE(

    [Total Sales],

    ALLEXCEPT(

        Sales,

        Sales[Category]

    )

)
```

«Remove every filter on the Sales table except the Category filter.»

15.4 ALLSELECTED

ALLSELECTED removes filters that come from inside the visual, but keeps filters that come from outside (slicers, page filters, etc.). It is extremely useful for "percentage of selected total".

```
Sales % of Selected =

DIVIDE(

    [Total Sales],

    CALCULATE(

        [Total Sales],

        ALLSELECTED(Sales[Region])

    )

)
```

15.5 KEEPFILTERS

Normally, a filter argument inside CALCULATE replaces any existing filter on the same column. KEEPFILTERS changes that behavior so the new filter is intersected with (kept together with) the existing filter.

```
North Sales Keep =

CALCULATE(

    [Total Sales],

    KEEPFILTERS(

        Sales[Region] = "North"

    )

)
```

This is an advanced topic. Use it when you need the new filter to work together with, rather than replace, existing filters.

16. VALUES and DISTINCT

16.1 VALUES

VALUES returns the distinct values of a column that are visible under the current filter context. It returns a single-column table.

```
Selected Region Count =

COUNTROWS(

    VALUES(Sales[Region])

)
```

If the user has selected two regions in a slicer, this measure returns 2.

16.2 DISTINCT

DISTINCT also returns unique values. The main practical difference for beginners is subtle; VALUES is more commonly used with filter context and has special behavior with blank values in some scenarios.

```
Unique Customers =

COUNTROWS(

    DISTINCT(Sales[Customer])

)
```

17. Context Transition

Context transition is one of the most confusing ideas for beginners. We will go slowly.

When you use CALCULATE (or a measure, which internally uses CALCULATE) inside a row context, CALCULATE converts the current row context into an equivalent filter context. This conversion is called context transition.

Simple diagram:

```
ROW CONTEXT

     ↓

CALCULATE (or measure call)

     ↓

CONTEXT TRANSITION

     ↓

FILTER CONTEXT

     ↓

MEASURE EVALUATION
```

You do not need to master every detail on the first reading. Just remember: placing a measure inside an iterator (or using CALCULATE inside a calculated column) triggers context transition.

18. Function Comparison Tables

SUM vs SUMX

Aspect	SUM	SUMX
What it does	Adds a column	Iterates a table and sums an expression
Accepts	A single column	A table + an expression
Row-level calc	No	Yes
When to use	Simple totals	When you need Quantity × Price, IF logic, etc.
Complexity	Simple	More flexible, slightly heavier

AVERAGE vs AVERAGEX

Aspect	AVERAGE	AVERAGEX
What it does	Averages a column	Iterates and averages an expression
Accepts	A single column	A table + an expression
Row-level calc	No	Yes
Typical use	Average of SalesAmount	Average of (SalesAmount – Cost)

COUNT Family

Function	Counts	Ignores Blank?	Typical Use
COUNT	Numeric values	Yes	Numbers / IDs
COUNTA	Non-blank values	Yes	Text + numbers
COUNTX	Non-blank expression results	Yes	Conditional / calculated counts
COUNTROWS	Rows in a table	N/A	Number of rows / orders

19. Common Beginner Mistakes

Mistake 1 — Using SUM when row-level calculation is required

Writing SUM(Sales[Quantity] * Sales[SalesAmount]) fails because SUM does not accept an expression. Use SUMX.

Mistake 2 — Using SUMX when SUM is enough

SUMX(Sales, Sales[SalesAmount]) works but is unnecessary. Prefer the simpler SUM(Sales[SalesAmount]).

Mistake 3 — Not understanding filter context

Expecting a measure to always return the same number regardless of slicers or matrix rows.

Mistake 4 — Confusing calculated columns and measures

Creating a calculated column for a total that should respond to filters. Use a measure instead.

Mistake 5 — Using FILTER unnecessarily

Writing CALCULATE([Total Sales], FILTER(Sales, Sales[Region] = "North")) when the simple filter Sales[Region] = "North" is sufficient and clearer.

Mistake 6 — Not understanding ALL

Forgetting that ALL removes filters, so percentage-of-total measures need ALL (or REMOVEFILTERS) in the denominator.

Mistake 7 — Not understanding CALCULATE

Thinking CALCULATE is only for "complex" formulas. Almost every interesting measure uses CALCULATE.

Mistake 8 — Ignoring relationships

Writing DAX that assumes tables are related when no relationship exists (or the relationship is inactive).

Mistake 9 — Confusing COUNT and COUNTA

Using COUNT on a text column. COUNT only works on numeric/date columns.

Mistake 10 — Thinking X functions are automatically "advanced"

X functions are simply iterators. Use them when you need row-by-row evaluation; otherwise prefer the simpler versions.

20. DAX Debugging Process

When a measure returns unexpected results, follow this systematic process:

1. Check the table — Does the table exist and contain the expected rows?
2. Check the column — Is the column name spelled correctly? Is the data type correct?
3. Check data type — Numbers stored as text will cause problems.
4. Check filter context — What slicers, page filters, and visual filters are active?
5. Check row context — Are you inside an iterator or calculated column?
6. Test the base measure — Does the simplest version (e.g. SUM) work correctly?
7. Test FILTER separately — Create a temporary measure that only returns COUNTROWS of the FILTER result.
8. Test CALCULATE step by step — Add one filter at a time.
9. Break complicated formulas into smaller measures — Name each intermediate step.

Pro Tip

Create temporary measures with simple names like Test1, Test2 while debugging. Delete them once the final measure works.

21. Practice Exercises

Level 1 — Beginner

1. Count the number of Sales IDs.
2. Count the number of customer entries (use COUNTA).
3. Calculate total sales.
4. Calculate total cost.
5. Calculate average sales amount.
6. Count the total number of rows in Sales.

Level 2 — Beginner+

1. North Sales
2. South Sales
3. Electronics Sales
4. Furniture Sales
5. North Order Count
6. Average Electronics Sales

Level 3 — Intermediate

1. Sales above ■30,000
2. North high-value sales (Region = North AND SalesAmount > 30000)
3. Sales percentage of total (by Region)
4. Category percentage of total
5. Selected-region percentage (using ALLSELECTED)

Level 4 — Advanced

1. CALCULATE + FILTER + SUMX combination
2. CALCULATE + AVERAGEX
3. ALL + CALCULATE for % of total
4. ALLEXCEPT example
5. ALLSELECTED example
6. KEEPFILTERS example
7. VALUES + COUNTROWS
8. Nested CALCULATE

22. Answer Key (Selected)

Level 1 Answers

1. Count Sales IDs

```
Order Count =

COUNT(Sales[SalesID])
```

Expected result: 10

3. Total Sales

```
Total Sales =

SUM(Sales[SalesAmount])
```

Expected result: 276000

6. Total Rows

```
Total Orders =

COUNTROWS(Sales)
```

Expected result: 10

Level 2 Answers

1. North Sales

```
North Sales =

CALCULATE(

    [Total Sales],

    Sales[Region] = "North"

)
```

3. Electronics Sales

```
Electronics Sales =

CALCULATE(

    [Total Sales],

    Sales[Category] = "Electronics"

)
```

Level 3 Answers

1. Sales above 30000

```
Sales Above 30000 =

CALCULATE(

    [Total Sales],

    FILTER(

        Sales,

        Sales[SalesAmount] > 30000

    )

)
```

3. Sales % of Total (by Region)

```
Sales % of Total =

DIVIDE(

    [Total Sales],

    CALCULATE(

        [Total Sales],

        ALL(Sales[Region])

    )

)
```

23. Mini Project — Sales Performance Dashboard

Build a realistic Sales Performance Dashboard using the measures below. Create each measure step by step and place them in cards, tables, and charts.

Required Measures

Total Sales

```
Total Sales =

SUM(Sales[SalesAmount])
```

Total Cost

```
Total Cost =

SUM(Sales[Cost])
```

Total Profit

```
Total Profit =

SUM(Sales[Profit])
```

Total Orders

```
Total Orders =

COUNTROWS(Sales)
```

Average Order Value

```
Average Order Value =

AVERAGEX(Sales, Sales[SalesAmount])
```

North Sales

```
North Sales =

CALCULATE([Total Sales], Sales[Region] = "North")
```

South Sales

```
South Sales =

CALCULATE([Total Sales], Sales[Region] = "South")
```

Electronics Sales

```
Electronics Sales =

CALCULATE([Total Sales], Sales[Category] = "Electronics")
```

Furniture Sales

```
Furniture Sales =

CALCULATE([Total Sales], Sales[Category] = "Furniture")
```

Sales % of Total

```
Sales % of Total =

DIVIDE([Total Sales], CALCULATE([Total Sales], ALL(Sales[Region])))
```

Category % of Total

```
Category % of Total =

DIVIDE([Total Sales], CALCULATE([Total Sales], ALL(Sales[Category])))
```

High Value Sales

```
High Value Sales =

CALCULATE([Total Sales], FILTER(Sales, Sales[SalesAmount] > 30000))
```

Selected Region Sales

```
Selected Region Sales =

CALCULATE([Total Sales], ALLSELECTED(Sales[Region]))
```

Dashboard Tips

- Put Total Sales, Total Profit, Total Orders, Average Order Value in Cards at the top.
- Create a matrix with Region on rows and the regional sales measures.
- Create a bar chart of Sales by Category.
- Add a slicer for Region and another for Category.
- Add the % of Total measures so users can see contribution.

24. DAX Cheat Sheet

Function	Purpose	Simple Example
COUNT	Count numeric values	COUNT(Sales[Quantity])
COUNTA	Count non-blank values	COUNTA(Sales[Customer])
COUNTX	Count expression results	COUNTX(Sales, Sales[SalesID])
COUNTROWS	Count rows in a table	COUNTROWS(Sales)
SUM	Sum a column	SUM(Sales[SalesAmount])
SUMX	Sum an expression row-by-row	SUMX(Sales, Qty * Price)
AVERAGE	Average a column	AVERAGE(Sales[SalesAmount])
AVERAGEX	Average an expression	AVERAGEX(Sales, Amt - Cost)
CALCULATE	Change filter context	CALCULATE([Total], Region="North")
FILTER	Return filtered table	FILTER(Sales, Amt > 30000)
ALL	Remove filters	ALL(Sales[Region])
REMOVEFILTERS	Clear filters (clearer)	REMOVEFILTERS(Sales[Region])
ALLEXCEPT	Remove all except...	ALLEXCEPT(Sales, Sales[Category])
ALLSELECTED	Respect outer filters	ALLSELECTED(Sales[Region])
KEEPFILTERS	Intersect filters	KEEPFILTERS(Region="North")
VALUES	Distinct visible values	VALUES(Sales[Region])
DISTINCT	Unique values	DISTINCT(Sales[Customer])

25. Final DAX Learning Roadmap

You have now completed a full journey from absolute beginner to advanced DAX concepts.

What you should now be able to do:

- Explain the difference between a measure and a calculated column
- Describe filter context and row context in simple words
- Choose correctly between COUNT / COUNTA / COUNTX / COUNTROWS
- Decide when to use SUM vs SUMX and AVERAGE vs AVERAGEX
- Write CALCULATE measures with simple and multiple filters
- Use FILTER for complex conditions
- Build percentage-of-total measures with ALL / REMOVEFILTERS / ALLSELECTED
- Understand the purpose of ALLEXCEPT and KEEPFILTERS
- Debug a measure systematically
- Build a complete Sales Performance Dashboard

Continue Your Journey

Next topics to explore after this handbook:
- Time Intelligence (TOTALYTD, SAMEPERIODLASTYEAR, DATEADD, etc.)
- Ranking (RANKX)
- Virtual relationships (TREATAS)
- Advanced context transition patterns
- Performance tuning and variables (VAR)

Keep practicing with real datasets. The more you write and test, the deeper your understanding becomes.

Congratulations on completing the DAX Mastery Student Handbook.

You now understand not only what formula to write, but why it works.