CHEATSHEET SILO

PRINTABLE QUICK REFERENCE

# Power BI DAX Cheat Sheet

Functions, formulas, measures and practical examples

## Inside this guide

- Core syntax, context and CALCULATE
- Aggregation, iterator, filter and table functions
- Date, time intelligence and ranking patterns
- Business measures, troubleshooting and performance

SECTION 01

# Quick Reference

Use this page to orient yourself before jumping into the function reference and reusable measures.

### 02. Measures, columns and context

Compact reference section with syntax and ready-to-adapt examples.

### 03. CALCULATE, variables and operators

Compact reference section with syntax and ready-to-adapt examples.

### 04. Aggregation and iterator functions

Compact reference section with syntax and ready-to-adapt examples.

### 05. Filter, table and relationship functions

Compact reference section with syntax and ready-to-adapt examples.

### 06. Logical, information, text and date functions

Compact reference section with syntax and ready-to-adapt examples.

### 07. Time intelligence and ranking

Compact reference section with syntax and ready-to-adapt examples.

### 08. Common measure patterns

Compact reference section with syntax and ready-to-adapt examples.

### 09. Practical business examples

Compact reference section with syntax and ready-to-adapt examples.

### 10. Errors, troubleshooting and performance

Compact reference section with syntax and ready-to-adapt examples.

### 11. DAX vs M vs Excel vs SQL

Compact reference section with syntax and ready-to-adapt examples.

## Essential syntax

MEASURE SYNTAX

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

Filtered Sales =
CALCULATE(
    [Total Sales],
    Product[Color] = "Red"
)
```

| Reference | Example | Meaning |
|---|---|---|
| **Measure** | [Total Sales] | A reusable calculation evaluated in filter context. |
| **Column** | Sales[Quantity] | A column reference qualified by its table. |
| **Table** | 'Sales Data' | Single quotes are used when a table name contains spaces. |
| **Variable** | VAR SalesPY = ... | Stores a scalar or table result inside an expression. |
| **Blank** | BLANK() | Represents an absent value; often preferred to zero in visuals. |

SECTION 02

# Measures, Calculated Columns and Context

Understanding where and how an expression is evaluated is the foundation of reliable DAX.

| Feature | Measure | Calculated column |
|---|---|---|
| **Evaluation** | At query time in the current filter context. | For each row during model refresh. |
| **Storage** | Stores the definition, not one value per row. | Stores a value for every row. |
| **Best use** | KPIs, ratios, totals and dynamic analysis. | Grouping, sorting, relationships or static row attributes. |
| **Responds to slicers** | Yes. | The stored row value does not recalculate from slicers. |
| **Typical choice** | Prefer for report calculations. | Use only when a persistent row-level value is required. |

### Row context

The current row. It commonly exists in calculated columns and iterator functions such as SUMX.

### Filter context

The values allowed by slicers, visual axes, filters, relationships and filter arguments.

### Context transition

CALCULATE can transform an existing row context into filter context.

### Relationship flow

Filters normally propagate from dimension tables to related fact tables.

**COLUMN VERSUS MEASURE**

```
Line Amount =
Sales[Quantity] * Sales[Unit Price]

Total Sales =
SUMX(
    Sales,
    Sales[Quantity] * Sales[Unit Price]
)
```

### Why totals can look different

A total evaluates the measure again in the total row's filter context. It does not always add the visible rows. Use an iterator when the intended business logic must be evaluated row by row.

SECTION 03

# CALCULATE, Variables and Operators

CALCULATE modifies filter context; variables make complex expressions clearer and easier to reuse.

### CALCULATE PATTERNS

```
Blue Sales =
CALCULATE(
    [Total Sales],
    Product[Color] = "Blue"
)

All Product Sales =
CALCULATE(
    [Total Sales],
    REMOVEFILTERS(Product)
)
```

### VAR AND RETURN

```
Sales YoY Growth % =
VAR CurrentSales = [Total Sales]
VAR PreviousSales =
    CALCULATE(
        [Total Sales],
        SAMEPERIODLASTYEAR('Date'[Date])
    )
RETURN
    DIVIDE(CurrentSales - PreviousSales, PreviousSales)
```

| Operator type | Operators | Example |
|---|---|---|
| **Arithmetic** | + - * / ^ | [Revenue] - [Cost] |
| **Comparison** | = == <> > < >= <= | Sales[Quantity] >= 10 |
| **Text** | & | Customer[First] & " " & Customer[Last] |
| **Logical AND** | && | [Sales] > 0 && [Margin] > 0 |
| **Logical OR** | \|\| | Product[Color] = "Red" \|\| Product[Color] = "Blue" |
| **Membership** | IN | Product[Color] IN {"Red", "Blue"} |

**Variable rule**

A variable is evaluated once in the context where it is defined. Use clear names and return only the final expression.

SECTION 04

# Aggregation and Iterator Functions

Use simple aggregators for a column; use X-functions when an expression must be evaluated for each row.

| Function | Purpose | Syntax / example |
|---|---|---|
| **SUM** | Adds numeric values in a column. | SUM(Sales[Sales Amount]) |
| **AVERAGE** | Returns the arithmetic mean of a column. | AVERAGE(Sales[Unit Price]) |
| **MIN / MAX** | Returns the smallest or largest value. | MAX(Sales[Order Date]) |
| **COUNT** | Counts numeric nonblank values. | COUNT(Sales[Quantity]) |
| **COUNTA** | Counts nonblank values. | COUNTA(Customer[Email]) |
| **COUNTROWS** | Counts rows in a table expression. | COUNTROWS(Sales) |
| **DISTINCTCOUNT** | Counts unique nonblank values. | DISTINCTCOUNT(Sales[Customer ID]) |
| **SUMX** | Evaluates an expression per row, then sums. | SUMX(Sales, Sales[Qty] * Sales[Price]) |
| **AVERAGEX** | Evaluates an expression per row, then averages. | AVERAGEX(Product, [Product Sales]) |
| **MINX / MAXX** | Returns the minimum or maximum evaluated result. | MAXX(Product, [Product Sales]) |
| **COUNTX** | Counts nonblank expression results. | COUNTX(Sales, Sales[Invoice]) |
| **RANKX** | Ranks evaluated values across a table. | RANKX(ALL(Product[Name]), [Total Sales]) |

### COMMON AGGREGATIONS

```
Average Order Value =
DIVIDE(
    [Total Sales],
    DISTINCTCOUNT(Sales[Order ID])
)

Weighted Revenue =
SUMX(
    Sales,
    Sales[Quantity] * Sales[Unit Price]
)
```

### Choose the simpler function

Use SUM when you only need to aggregate one stored column. Use SUMX when each row requires an expression before aggregation.

SECTION 05

# Filter, Table and Relationship Functions

These functions shape virtual tables, modify context and work with relationships inside the model.

| Function | Purpose | Example |
|---|---|---|
| **FILTER** | Returns rows satisfying an expression. | FILTER(Sales, Sales[Amount] > 1000) |
| **ALL** | Removes filters from a table or column. | ALL(Product) |
| **REMOVEFILTERS** | Clears filters without returning a table for reuse. | REMOVEFILTERS(Product[Category]) |
| **ALLEXCEPT** | Removes filters except specified columns. | ALLEXCEPT('Date', 'Date'[Year]) |
| **ALLSELECTED** | Keeps filters originating outside the current query shape. | ALLSELECTED(Product[Category]) |
| **KEEPFILTERS** | Intersects a filter with existing filters. | KEEPFILTERS(Product[Color] = "Red") |
| **VALUES** | Returns distinct values plus a possible blank member. | VALUES(Product[Category]) |
| **DISTINCT** | Returns distinct values or rows. | DISTINCT(Sales[Customer ID]) |
| **SELECTCOLUMNS** | Projects named columns from a table expression. | SELECTCOLUMNS(Product, "Name", Product[Name]) |
| **ADDCOLUMNS** | Adds calculated columns to a table expression. | ADDCOLUMNS(Product, "Sales", [Total Sales]) |
| **SUMMARIZECOLUMNS** | Groups columns and returns summary results. | SUMMARIZECOLUMNS(Product[Category], "Sales", [Total Sales]) |
| **TOPN** | Returns the top N rows by an expression. | TOPN(5, Product, [Total Sales], DESC) |
| **RELATED** | Returns a value from a related one-side table. | RELATED(Product[Category]) |
| **RELATEDTABLE** | Returns related rows from another table. | COUNTROWS(RELATEDTABLE(Sales)) |
| **USERELATIONSHIP** | Activates an existing inactive relationship for a calculation. | USERELATIONSHIP(Sales[Ship Date], 'Date'[Date]) |
| **TREATAS** | Applies table values as filters to other columns. | TREATAS(VALUES(Input[Year]), 'Date'[Year]) |

### FILTER CONTEXT PATTERN

```
Product % of Total =
DIVIDE(
    [Total Sales],
    CALCULATE([Total Sales], REMOVEFILTERS(Product))
)
```

SECTION 06

# Logical, Information, Text and Date Functions

Small utility functions often make measures safer, more readable and more useful in report labels.

| Category | Functions | Typical use |
|---|---|---|
| **Logical** | IF, SWITCH, AND, OR, NOT, COALESCE | Branching, classification and fallback values. |
| **Information** | ISBLANK, ISERROR, ISNUMBER, ISTEXT, HASONEVALUE | Test values or the current selection state. |
| **Selection** | SELECTEDVALUE | Return one selected value or an alternate result. |
| **Text** | CONCATENATE, CONCATENATEX, FORMAT, LEFT, RIGHT, MID | Labels, lists, display formatting and text extraction. |
| **Search** | CONTAINSSTRING, SEARCH, FIND | Test or locate text inside another string. |
| **Date creation** | DATE, TIME, DATEVALUE, TIMEVALUE | Create or convert date and time values. |
| **Date parts** | YEAR, MONTH, DAY, WEEKDAY, WEEKNUM | Extract calendar attributes. |
| **Date arithmetic** | DATEDIFF, EDATE, EOMONTH | Compare dates or move across months. |
| **Current date** | TODAY, NOW | Return the current date or date and time. |

### LOGIC AND SELECTION

```
Performance Band =
SWITCH(
    TRUE(),
    [Margin %] >= 0.30, "High",
    [Margin %] >= 0.15, "Medium",
    "Low"
)

Selected Category =
SELECTEDVALUE(Product[Category], "Multiple categories")
```

### TEXT AND DATES

```
Customer List =
CONCATENATEX(
    VALUES(Customer[Customer Name]),
    Customer[Customer Name],
    ", ",
    Customer[Customer Name], ASC
)

Month End = EOMONTH(MAX('Date'[Date]), 0)
```

SECTION 07

# Time Intelligence, Ranking and Statistics

Reliable time intelligence requires a suitable date table and a correct relationship to the fact table.

| Function | Purpose | Pattern |
|---|---|---|
| **TOTALYTD** | Year-to-date aggregation. | TOTALYTD([Total Sales], 'Date'[Date]) |
| **TOTALQTD** | Quarter-to-date aggregation. | TOTALQTD([Total Sales], 'Date'[Date]) |
| **TOTALMTD** | Month-to-date aggregation. | TOTALMTD([Total Sales], 'Date'[Date]) |
| **DATEADD** | Shifts the current date context. | DATEADD('Date'[Date], -1, YEAR) |
| **SAMEPERIODLASTYEAR** | Returns the corresponding prior-year dates. | SAMEPERIODLASTYEAR('Date'[Date]) |
| **DATESINPERIOD** | Returns dates over a relative interval. | DATESINPERIOD('Date'[Date], MAX('Date'[Date]), -12, MONTH) |
| **DATESBETWEEN** | Returns dates between explicit boundaries. | DATESBETWEEN('Date'[Date], StartDate, EndDate) |
| **RANKX** | Ranks an expression over a table. | RANKX(ALL(Product[Name]), [Total Sales],, DESC) |
| **MEDIANX** | Returns the median evaluated result. | MEDIANX(Customer, [Customer Sales]) |
| **PERCENTILEX.INC** | Returns an inclusive percentile. | PERCENTILEX.INC(Product, [Total Sales], 0.9) |
| **STDEVX.P** | Population standard deviation over an expression. | STDEVX.P(Product, [Total Sales]) |

**TIME AND RANKING MEASURES**

```
Sales YTD =
TOTALYTD([Total Sales], 'Date'[Date])

Sales Previous Year =
CALCULATE(
    [Total Sales],
    SAMEPERIODLASTYEAR('Date'[Date])
)

Sales Rank =
RANKX(
    ALL(Product[Product Name]),
    [Total Sales],,
    DESC,
    DENSE
)
```

SECTION 08

# Common DAX Measure Patterns

Create a small set of trusted base measures, then reuse them in ratios, comparisons and cumulative calculations.

| Pattern | Formula |
| --- | --- |
| **Total Sales** | Total Sales = SUM(Sales[Sales Amount]) |
| **Gross Profit** | Gross Profit = [Total Sales] - [Total Cost] |
| **Gross Margin %** | Gross Margin % = DIVIDE([Gross Profit], [Total Sales]) |
| **Distinct Customers** | Customers = DISTINCTCOUNT(Sales[Customer ID]) |
| **Average Order Value** | Average Order Value = DIVIDE([Total Sales], [Orders]) |
| **Percent of Total** | Sales % Total = DIVIDE([Total Sales], CALCULATE([Total Sales], REMOVEFILTERS(Product))) |

### CUMULATIVE AND ROLLING CALCULATIONS

```
Running Total Sales =
VAR LastVisibleDate = MAX('Date'[Date])
RETURN
    CALCULATE(
        [Total Sales],
        FILTER(
            ALL('Date'[Date]),
            'Date'[Date] <= LastVisibleDate
        )
    )

Rolling 12M Sales =
CALCULATE(
    [Total Sales],
    DATESINPERIOD(
        'Date'[Date],
        MAX('Date'[Date]),
        -12,
        MONTH
    )
)
```

### DYNAMIC METRIC SELECTOR

```
Selected Metric =
SWITCH(
    SELECTEDVALUE('Metric'[Metric]),
    "Sales", [Total Sales],
    "Profit", [Gross Profit],
    "Margin", [Gross Margin %],
    BLANK()
)
```

SECTION 09

# Practical Business Examples

Adapt the table and column names to your model while preserving the intent of each calculation.

### CUSTOMER AND PRODUCT ACTIVITY

```
Active Customers =
CALCULATE(
    DISTINCTCOUNT(Sales[Customer ID]),
    Sales[Sales Amount] > 0
)

Products With Sales =
COUNTROWS(
    FILTER(
        VALUES(Product[Product ID]),
        [Total Sales] > 0
    )
)
```

### RELATIONSHIPS AND CONDITIONAL SALES

```
Sales by Ship Date =
CALCULATE(
    [Total Sales],
    USERELATIONSHIP(Sales[Ship Date], 'Date'[Date])
)

High Value Sales =
CALCULATE(
    [Total Sales],
    FILTER(Sales, Sales[Sales Amount] > 1000)
)
```

### CUSTOMER ACQUISITION PATTERN

```
New Customers =
VAR VisibleCustomers = VALUES(Sales[Customer ID])
VAR FirstPurchaseCustomers =
    FILTER(
        VisibleCustomers,
        CALCULATE(MIN(Sales[Order Date]), ALL('Date'))
            IN VALUES('Date'[Date])
    )
RETURN
    COUNTROWS(FirstPurchaseCustomers)
```

**Model dependency**

Business measures are only as reliable as the model beneath them. Confirm table grain, keys, relationships and date logic before debugging the final expression.

SECTION 10

# Errors, Troubleshooting and Performance

Debug from the model outward: relationships and filter context often explain results that initially look like formula errors.

| Symptom | Likely cause | Check |
|---|---|---|
| **Same result on every row** | Filter context is missing or not reaching the fact table. | Relationships, field source and filter direction. |
| **Unexpected total** | The measure is reevaluated at total level. | Test the intended row-level expression with an iterator. |
| **Blank result** | No matching rows, empty selection or zero denominator. | Intermediate variables, relationships and filters. |
| **Circular dependency** | Calculated objects depend on each other. | Move logic upstream or redesign dependencies. |
| **Time intelligence fails** | Date table or relationship is unsuitable. | Continuous dates, data type and active relationship. |
| **Slow visual** | Large scans, complex iterators or too much detail. | Performance Analyzer, model size and visual fields. |

## Performance checklist

### Model first

Use a clear star schema, consistent fact grain and appropriate relationships.

### Reduce volume

Remove unused rows and columns and avoid high-cardinality text where possible.

### Use variables

Store repeated expressions with VAR to improve readability and avoid repeated work.

### Filter efficiently

Prefer Boolean filter arguments for simple conditions; reserve FILTER for complex logic.

### Prefer DIVIDE

Use DIVIDE when a denominator may be zero or blank.

### Test visuals

Use Performance Analyzer to identify expensive visuals and DAX queries.

**DEBUG WITH VARIABLES**

```
Debug Measure =
VAR Step1 = [Total Sales]
VAR Step2 = CALCULATE([Total Sales], REMOVEFILTERS(Product))
RETURN
    Step2  // Return one variable at a time while testing
```

SECTION 11

# DAX vs M vs Excel vs SQL

Choose the language based on when the logic should run and where the result belongs.

| Language | Primary role | Best fit |
|---|---|---|
| **DAX** | Semantic-model calculations. | Measures that respond to report filters and slicers. |
| **Power Query M** | Refresh-time data preparation. | Cleaning, merging, pivoting and reshaping imported data. |
| **Excel formulas** | Worksheet calculations. | Cell, range, table and dynamic-array analysis. |
| **SQL** | Database querying and transformation. | Source filtering, joins and large-scale aggregation. |

### Practical placement rule

Use SQL for source-side operations, Power Query for repeatable preparation, DAX for context-sensitive model calculations and Excel formulas for worksheet-based calculations. A strong solution may use all four.

## Final DAX checklist

| Step | Check |
|---|---|
| **1** | Start with a clear business question and a trusted base measure. |
| **2** | Identify the current row context and filter context. |
| **3** | Use CALCULATE only when the filter context must change. |
| **4** | Use variables to name and test intermediate results. |
| **5** | Prefer measures over stored calculated columns for report aggregations. |
| **6** | Validate totals separately from individual visual rows. |
| **7** | Optimize the model and visual before micro-optimizing syntax. |

## Official references

Microsoft Learn: DAX overview

Microsoft Learn: DAX function reference

Microsoft Learn: Star schema guidance

Microsoft Learn: Use variables to improve DAX formulas

Microsoft Learn: Performance Analyzer

### Keep learning

Visit cheatsheetsilo.com for the full searchable DAX guide and more free cheat sheets for data, development and productivity.