Learning to Calculate Days in a Month with Power BI DAX

Authored by

Mohammed looti

November 12, 2025

The Essential DAX Formula for Counting Monthly Days

In advanced time intelligence analysis within Power BI, accurately determining the number of days contained within a specific month is frequently required. This metric is essential for calculating averages, normalizing monthly totals, or ensuring accurate comparisons across time periods, especially when dealing with uneven calendars (e.g., February vs. July). The most robust and direct method utilizes a combination of DAX (Data Analysis Expressions) functions, providing reliable results regardless of leap years or dataset size.

The specific syntax detailed below allows developers to quickly generate a column that contains the total count of days for the month corresponding to each row's date entry. This approach leverages the inherent calendar intelligence built into the DAX engine by isolating the last day of the month and extracting its date number.

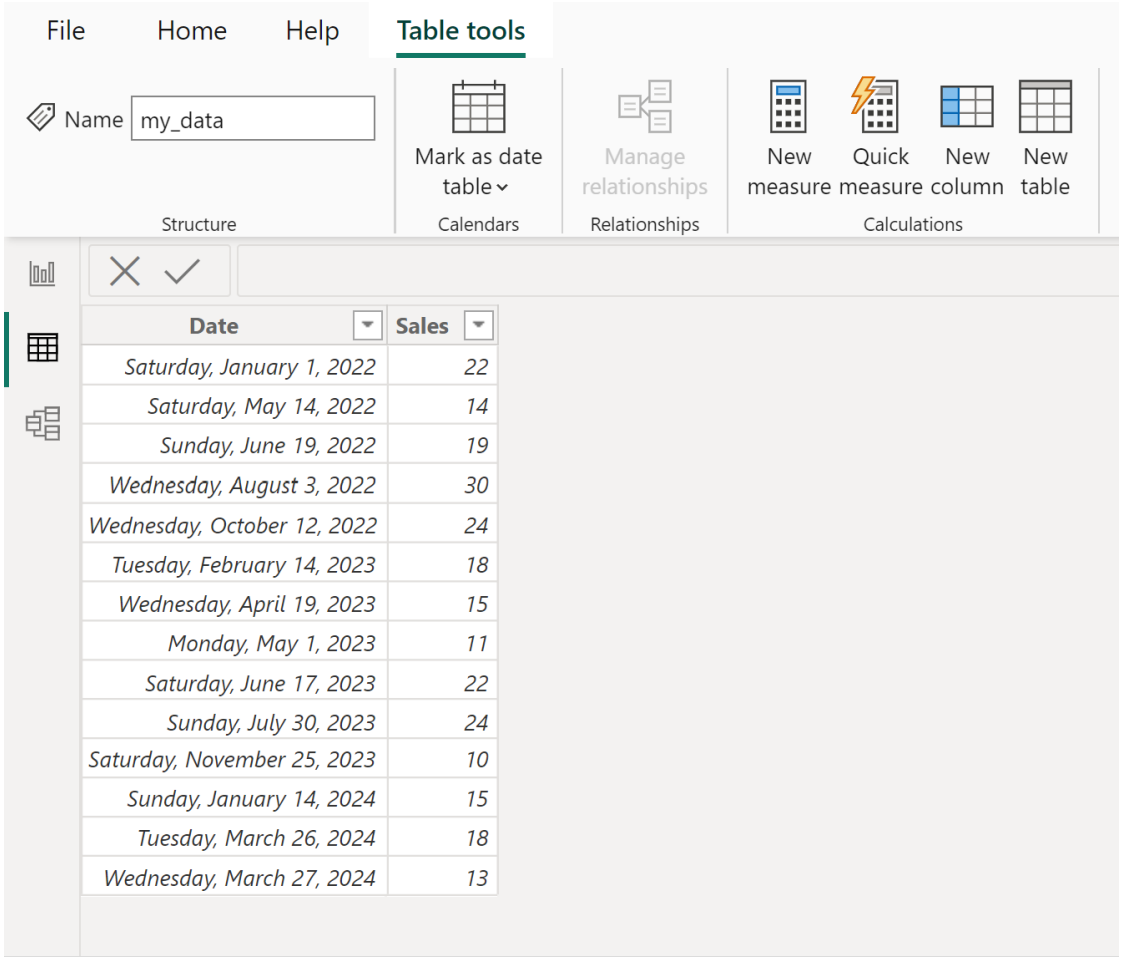
Days in Month = DAY(EOMONTH('my_data',0))

This particular formula creates a new calculated column named **Days in Month**. This column contains the precise number of days in the month for the corresponding date found in the **Date** column of the specified table (here, 'my_data'). Understanding how to deploy this technique is fundamental for effective data modeling and subsequent analysis in Power BI environments.

Practical Implementation: Creating the Calculated Column

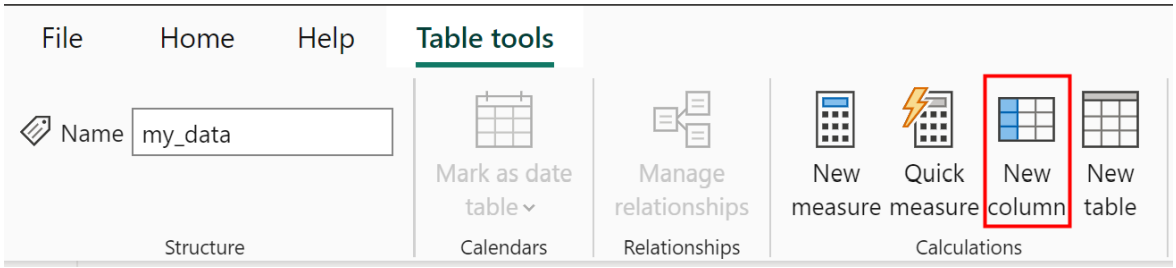
To illustrate the power and simplicity of this DAX solution, consider a scenario involving a sample dataset. Suppose we are analyzing a sales ledger stored in a table named **my_data**, which records transaction amounts against various dates. Our objective is to enrich this table by adding a column that quantifies the total number of days in the month associated with each sale date. This preparation is often necessary before calculating metrics like "Average Daily Sales per Month" or performing other time-based normalization.

We begin with our raw data table, which clearly shows the date and sales amounts. This foundation is necessary before we introduce any time intelligence calculations or augmentations.



Date	Sales
Saturday, January 1, 2022	22
Saturday, May 14, 2022	14
Sunday, June 19, 2022	19
Wednesday, August 3, 2022	30
Wednesday, October 12, 2022	24
Tuesday, February 14, 2023	18
Wednesday, April 19, 2023	15
Monday, May 1, 2023	11
Saturday, June 17, 2023	22
Sunday, July 30, 2023	24
Saturday, November 25, 2023	10
Sunday, January 14, 2024	15
Tuesday, March 26, 2024	18
Wednesday, March 27, 2024	13

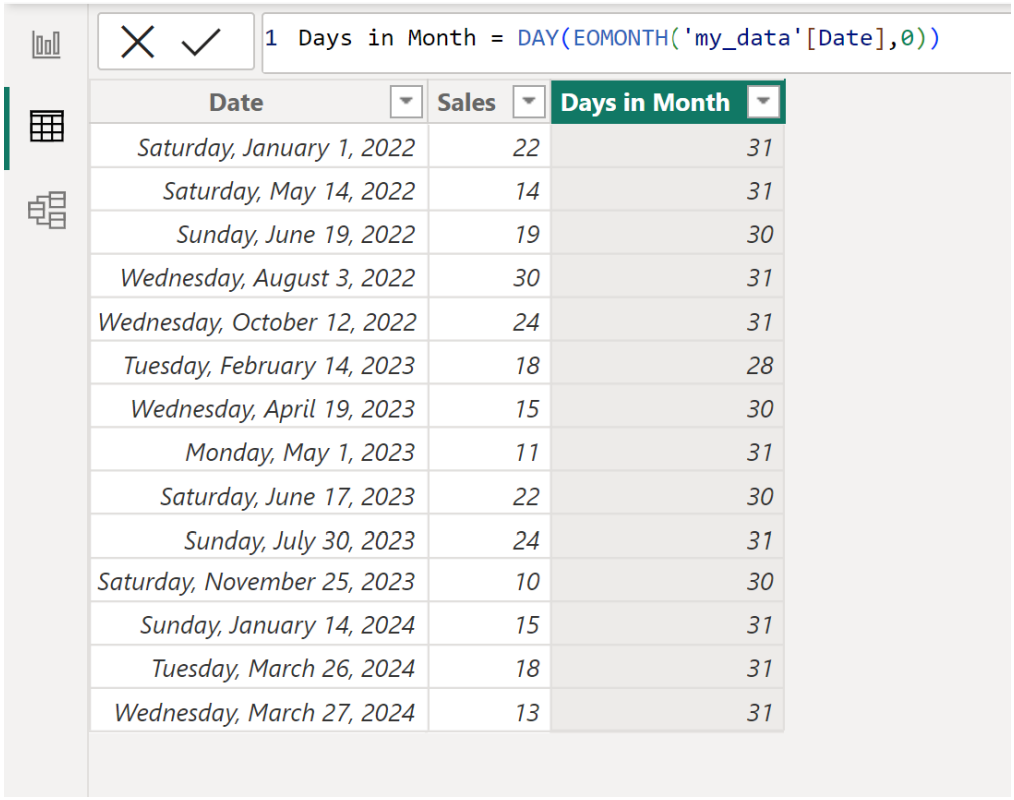
The first step in Power BI is navigating to the modeling environment. Click the **Table tools** tab in the ribbon interface, and then select the **New column** icon. This action opens the formula bar, allowing us to define our new calculated column using DAX syntax.



Once the formula bar is active, input the precise DAX syntax provided earlier. This calculation is instantaneous and highly efficient, relying on Power BI's optimized date engine.

Days in Month = DAY(EOMONTH('my_data',0))

Executing this formula yields a new column, **Days in Month**, which correctly assigns the total number of days for the corresponding date's month. This result automatically handles variations between 28, 29, 30, and 31 days, including complexities introduced by leap years.



Date	Sales	Days in Month
Saturday, January 1, 2022	22	31
Saturday, May 14, 2022	14	31
Sunday, June 19, 2022	19	30
Wednesday, August 3, 2022	30	31
Wednesday, October 12, 2022	24	31
Tuesday, February 14, 2023	18	28
Wednesday, April 19, 2023	15	30
Monday, May 1, 2023	11	31
Saturday, June 17, 2023	22	30
Sunday, July 30, 2023	24	31
Saturday, November 25, 2023	10	30
Sunday, January 14, 2024	15	31
Tuesday, March 26, 2024	18	31
Wednesday, March 27, 2024	13	31

Reviewing the results confirms the accurate calculation for various months throughout the year:

A date like January 1, 2022, correctly falls within January, resulting in **31** days in the month.
Similarly, May 14, 2022, belongs to May, which consistently has **31** days.
Conversely, June 19, 2022, is recognized as being in June, which only contains **30** days.

This validation ensures that the calculated column is reliable for any subsequent analytical or reporting tasks.

Deconstructing the Logic: How EOMONTH and DAY Functions Interact

To truly leverage the capabilities of DAX, it is essential to understand the functional mechanism behind this calculation. We rely on the nested structure of two powerful date functions to arrive at our desired integer output: the total number of days in the month.

Days in Month = DAY(EOMONTH('my_data',0))

The entire calculation hinges on the internal **EOMONTH** function. The purpose of EOMONTH (End of Month) is to return the date that represents the last day of a month, offset by a specified number of months. In our formula, we pass the date column ('my_data') and a crucial offset argument of **0**. Using zero ensures that the function returns the last calendar date of the month corresponding to the input date itself. For instance, if the input date is January 1, 2022, the **EOMONTH** function successfully isolates and returns the date value of **January 31, 2022**.

The result of the **EOMONTH** calculation--which is a full date value--is then fed as the argument into the outer **DAY** function. The DAY function is designed specifically to extract the day number (an integer from 1 to 31) from any given date value. Therefore, when **DAY** receives January 31, 2022, it simply extracts the numerical value, yielding **31**.

This chained process is repeated for every row in the source table. This mechanism guarantees that whether the date falls in a 30-day month (like November) or a 29-day month (like February during a leap year), the final output accurately reflects the total duration of that specific calendar month.

Alternative Approaches for Date Dimension Tables

While the calculated column method using **EOMONTH** and **DAY** is highly effective for ad-hoc analysis or when modifying a fact table, professional Power BI data models often rely on a dedicated Date Dimension table (commonly known as a Calendar table). In such dimensional structures, it is generally more efficient to pre-calculate the number of days in the month and store it directly within the dimension table itself, rather than adding redundant columns to potentially vast fact tables.

If you are working with a dimensional model, you would define the column within your central Calendar table. Assuming your Calendar table has a column named , the formula remains identical: `Days in Month = DAY(EOMONTH('Calendar',0))`. Once this attribute is established in the Calendar table, you can leverage table relationships to retrieve this value in your fact tables using functions like **RELATED**, which is typically more memory-efficient and scalable than creating the same calculated column in the larger transactional table.

A less common but valid alternative, particularly useful in measure creation, involves using time intelligence functions such as **LASTDATE** or **ENDOFMONTH** to establish context before applying the **DAY** function. However, for a simple row-by-row context calculation--which is required to assign a static attribute to every date--the nested **DAY(EOMONTH(...))** structure remains the cleanest and most performance-friendly option for defining this specific calculated column.

Performance Considerations and Best Practices

When implementing any DAX calculation, especially in large datasets, analysts must consider the performance implications. The formula we utilized creates a calculated column, meaning the results are computed during data refresh and stored in the model's memory (RAM). For this specific calculation, the formula is highly optimized because it operates purely within the row context; it does not require complex context transition or reference filters outside the current row, making it exceptionally fast to compute.

While the **EOMONTH** and **DAY** combination is efficient, a general best practice is to avoid creating excessive calculated columns if the required output can be achieved via a measure. However, in this scenario--where we need a static attribute (the number of days in that row's month) associated with every single row--a calculated column is the appropriate design choice. If the calculation were only used for aggregate reporting (e.g., total days in the currently filtered month), a measure using **MAX** or **LASTDATE** in conjunction with the calculated column might be preferable.

A crucial best practice when dealing with date calculations in Power BI is ensuring data type integrity. The output of the nested **DAY(EOMONTH(...))** formula will inherently be an integer (a whole number), which is the precise format required for subsequent numerical operations such as division (e.g., calculating daily averages). Always verify that Power BI has correctly assigned the resulting column as a Whole Number to maximize storage efficiency and calculation speed across the data model.

Summary and Additional Resources

The ability to accurately derive calendar metadata, such as the total number of days in a month, is foundational for advanced time intelligence modeling in Power BI. By skillfully combining the EOMONTH and DAY functions, developers can create robust and efficient data models that automatically account for all calendar variances, including leap years.

For those seeking deeper mastery of date and time handling, the complete documentation for the core functions used in this tutorial, especially the powerful **EOMONTH** function in DAX, should be reviewed for optional arguments and related date handling capabilities within the Microsoft ecosystem.

The following tutorials explain how to perform other common time intelligence tasks in Power BI: