Learning DAX: Calculating Cumulative Sums in Power BI

Authored by

Mohammed looti

November 12, 2025

Calculating a Cumulative Sum, frequently referred to as a running total, is an indispensable analytical technique within the realm of business intelligence. This powerful calculation allows analysts to dynamically track the total accumulation of a specific metric--such as sales figures, revenue streams, or inventory levels--over a chronological dimension, typically time. In the robust environment of Power BI, achieving this complex calculation, which requires manipulating the evaluation context, necessitates leveraging the sophisticated capabilities of Data Analysis Expressions (DAX). Mastering the precise DAX syntax is fundamental for generating accurate and responsive running totals that correctly interact with the dynamic filter contexts established in your reports and visuals.

The core challenge in implementing a running total in DAX stems from the need to evaluate a summation based not just on the current row, but on all previous rows up to that point. The standard and highly effective approach for deriving the cumulative sum of values within a designated column relies on a specific combination of functions that explicitly manipulate the filter context during row-by-row iteration. This method, often implemented as a calculated column, ensures that the historical data required for accumulation is always included in the calculation.

Deconstructing the Core DAX Formula for Cumulative Sums

The following syntax represents the foundational DAX pattern used to create a calculated column that produces a running total. This formula is designed to perform a complex context transition, which is essential for iterating through the table and summing values based on chronological order:

```
Cumulative Sum =
CALCULATE (
SUM ( 'my_data' ),
ALL ( 'my_data' ),
'my_data' <= EARLIER ( 'my_data' )
)
```

This expression is designed to instantiate a new calculated column titled **Cumulative Sum**. This column systematically calculates the running total of entries found in the **Sales** column, relative to the chronological order defined by the **Date** column within the table designated **my_data**. Understanding the crucial interplay between the primary functions--specifically the CALCULATE function and the EARLIER function--is paramount for grasping how filter context is managed row-by-row during this highly iterative operation. While simple in appearance, this formula encapsulates sophisticated logic for manipulating the data model's evaluation context.

Analyzing the Key Functions: CALCULATE, ALL, and EARLIER

To successfully implement a running total using this calculated column approach, it is essential to analyze the specific role and behavior of the three primary DAX functions used in this common pattern. The execution hinges on context modification, where the standard row context must be transformed into a filter context that correctly encompasses all preceding rows in time.

The CALCULATE function serves as the engine of context transition in DAX. It allows developers to override or modify the standard filter context under which an aggregation, defined here as **SUM('my_data')**, is evaluated. In the context of a running total, we instruct DAX to evaluate the sum of sales not just based on the filters of the current row, but across a modified subset of rows defined by stringent date criteria. This flexibility is what enables complex calculations that break free from simple row-level arithmetic.

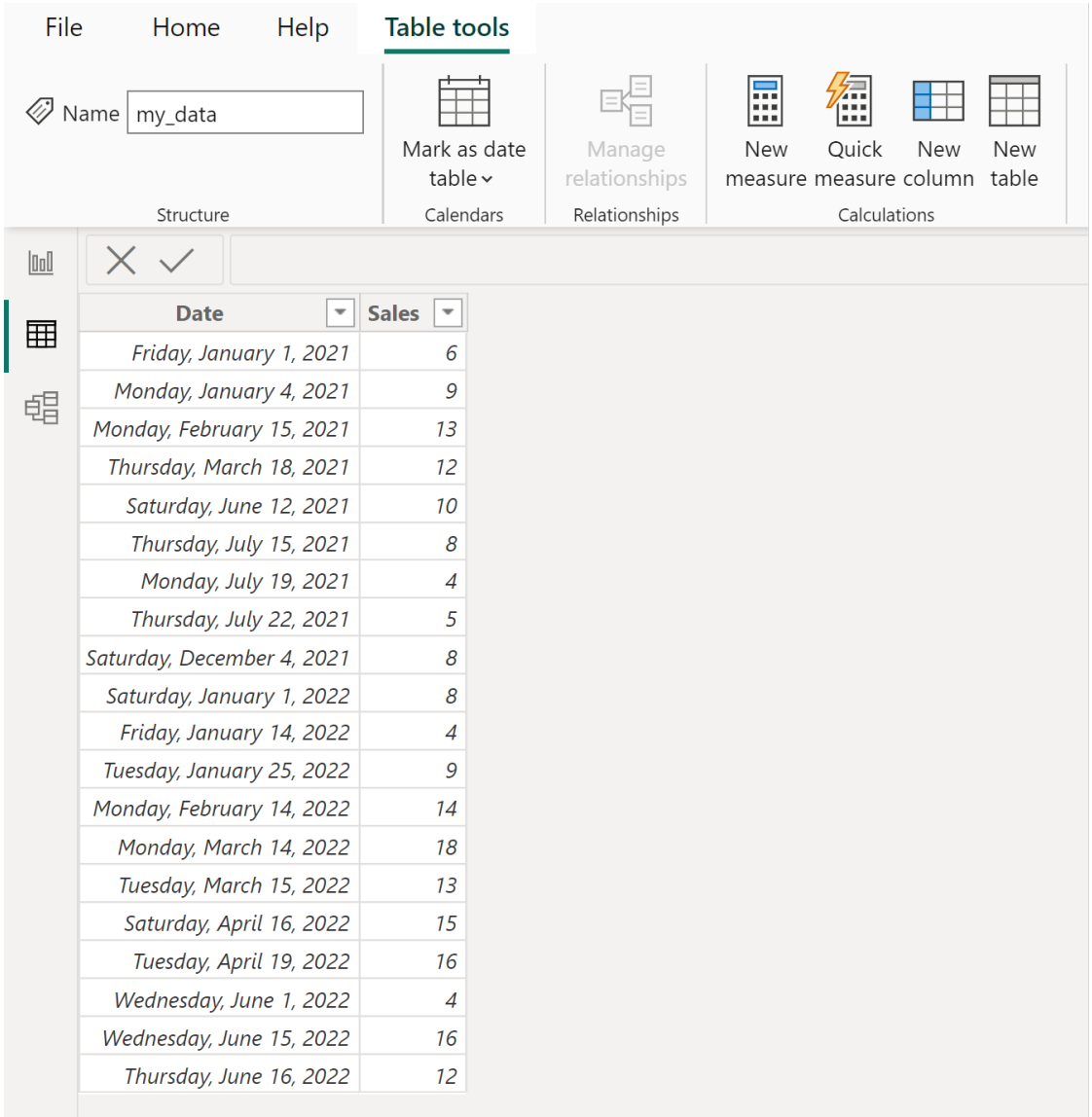
The second critical component is the filter argument **ALL('my_data')**. The role of the ALL function here is to remove any pre-existing filters that might be applied to the 'my_data' table, effectively restoring access to the entire dataset. This action is crucial; it ensures that the calculation begins with access to all historical sales figures necessary for accumulation. If ALL were omitted, the calculation would be limited only to the sales figures associated with the current date being evaluated, thereby failing to produce the desired cumulative effect by inadvertently respecting the implicit row filter. The CALCULATE function then applies the subsequent filter argument over this entire, unrestricted table.

The final and most defining filter argument is the comparison clause: **'my_data' <= EARLIER('my_data')**. The EARLIER function is specifically designed to work within calculated columns, retrieving the value of the specified column from the outer row context--that is, the row currently being processed by the iteration engine. As DAX iterates through the table row by row, EARLIER holds the date of the current iteration. The filter then restricts the aggregated SUM to only include those rows where the date is less than or equal to that current date held by EARLIER. This meticulous filtering mechanism successfully generates the chronological running total by building the necessary filter context for the CALCULATE function.

Practical Scenario: Implementing the Calculation in Power BI

To fully appreciate the utility of this DAX technique, let us consider a typical business intelligence scenario within Power BI. We are tasked with analyzing the performance of a company using a table named **my_data**, which logs daily sales transactions. This dataset minimally contains a Date column and a corresponding Sales amount column. While daily sales figures are informative for immediate monitoring, analyzing them cumulatively provides far superior insight into overarching performance trends, growth velocity, and the progression toward established business goals over longer time frames.

Our initial dataset, illustrating the distinct daily sales figures recorded across several dates, requires transformation. This static view of daily activity must be converted into a dynamic running total that allows for continuous monitoring of cumulative performance and growth trajectory. The input data typically looks like a simple ledger:

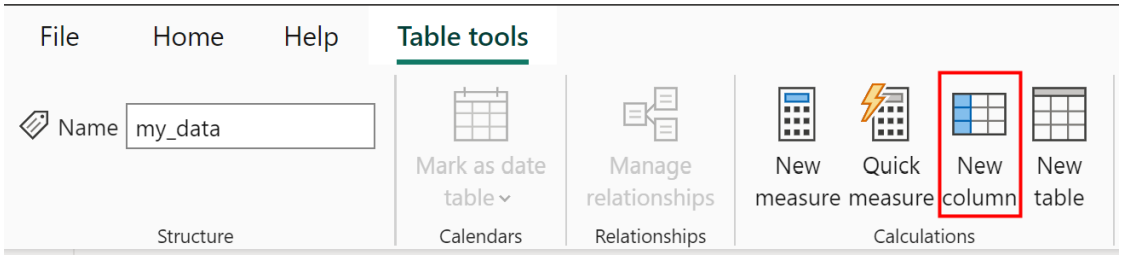


Date	Sales
Friday, January 1, 2021	6
Monday, January 4, 2021	9
Monday, February 15, 2021	13
Thursday, March 18, 2021	12
Saturday, June 12, 2021	10
Thursday, July 15, 2021	8
Monday, July 19, 2021	4
Thursday, July 22, 2021	5
Saturday, December 4, 2021	8
Saturday, January 1, 2022	8
Friday, January 14, 2022	4
Tuesday, January 25, 2022	9
Monday, February 14, 2022	14
Monday, March 14, 2022	18
Tuesday, March 15, 2022	13
Saturday, April 16, 2022	15
Tuesday, April 19, 2022	16
Wednesday, June 1, 2022	4
Wednesday, June 15, 2022	16
Thursday, June 16, 2022	12

The requirement is clearly defined: we must augment the data model by creating a new column that calculates the Cumulative Sum of the values in the **Sales** column, adhering strictly to the chronological ordering provided by the **Date** column. Since this calculation fundamentally depends on iterating through the row context of the table to determine the necessary filter range, the solution must be implemented as a calculated column within Power BI's Data view, utilizing the specific row context provided by the EARLIER function.

Step-by-Step Guide to Creating the Calculated Column

The process of creating a new calculated column in Power BI begins in the modeling environment. The first procedural step requires navigating to the **Table tools** tab, which is accessible when you are viewing your data model in the Data view. Within this ribbon interface, locate and select the **New column** icon. This action immediately initiates the column creation process and activates the formula bar, providing the necessary input field for our DAX expression.



With the formula bar active, the crucial step is to accurately input the DAX formula previously analyzed. It is imperative to verify that the table and column references (e.g., 'my_data' and 'Sales') exactly match the naming conventions within your specific data model. Inputting the following expression defines the comprehensive logic for the cumulative calculation, ensuring that each row correctly aggregates all prior sales up to and including its current date:

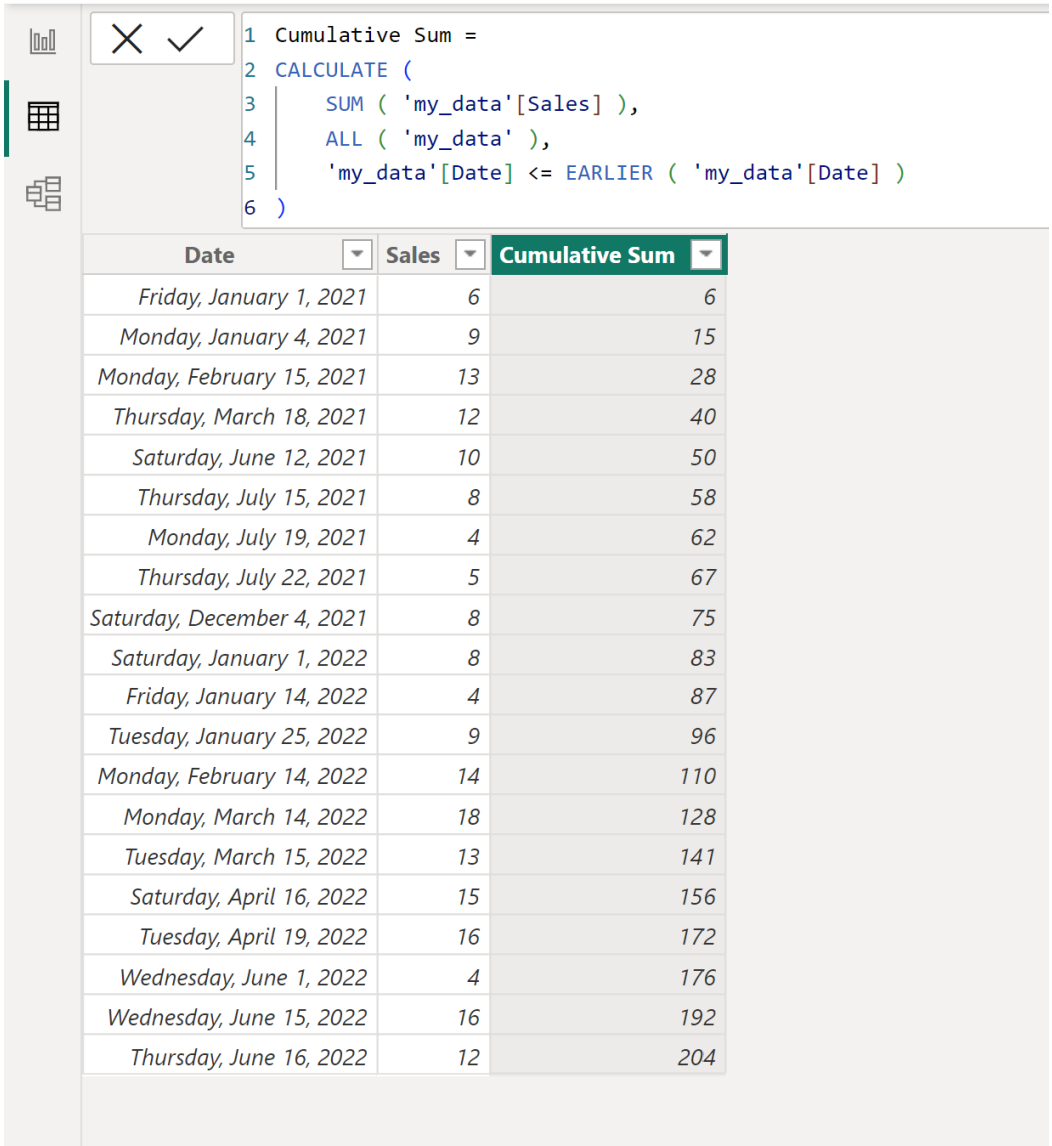
```
Cumulative Sum =
CALCULATE (
SUM ( 'my_data' ),
ALL ( 'my_data' ),
'my_data' <= EARLIER ( 'my_data' )
)
```

Upon entering the formula and confirming the input, Power BI initiates the calculation engine. It processes the instruction by iterating sequentially through every row of the **my_data** table. The culmination of this process is the automatic generation of a new column named **Cumulative Sum**. This column now dynamically holds the calculated running totals, successfully transforming the raw transaction data into a highly useful, chronologically sensitive analytical metric ready for advanced reporting and visualization.

Validating the Accuracy of the Cumulative Results

The successful execution of the DAX formula immediately updates the table structure, integrating the newly calculated column. Before proceeding to visualization, it is critical to verify the derived

values to confirm that the context transition and filtering operations performed exactly as intended. Visual confirmation demonstrates how the daily sales figures are progressively summed, offering a transparent view of the accumulation over time. This verification step confirms the accuracy of the highly specialized row-context manipulation orchestrated by the EARLIER function.



```
Cumulative Sum =
CALCULATE (
    SUM ( 'my_data'[Sales] ),
    ALL ( 'my_data' ),
    'my_data'[Date] <= EARLIER ( 'my_data'[Date] )
)
```

Date	Sales	Cumulative Sum
Friday, January 1, 2021	6	6
Monday, January 4, 2021	9	15
Monday, February 15, 2021	13	28
Thursday, March 18, 2021	12	40
Saturday, June 12, 2021	10	50
Thursday, July 15, 2021	8	58
Monday, July 19, 2021	4	62
Thursday, July 22, 2021	5	67
Saturday, December 4, 2021	8	75
Saturday, January 1, 2022	8	83
Friday, January 14, 2022	4	87
Tuesday, January 25, 2022	9	96
Monday, February 14, 2022	14	110
Monday, March 14, 2022	18	128
Tuesday, March 15, 2022	13	141
Saturday, April 16, 2022	15	156
Tuesday, April 19, 2022	16	172
Wednesday, June 1, 2022	4	176
Wednesday, June 15, 2022	16	192
Thursday, June 16, 2022	12	204

We can systematically inspect the derived values to confirm the accurate calculation of the Cumulative Sum. This row-by-row verification process is essential for establishing confidence in complex data transformations:

The cumulative sum of sales for the first recorded row (Date 1) is simply equal to the daily sales amount: **6**.

The cumulative sum for the second row (Date 2) correctly combines the sales from Date 1 and Date 2: 6 + 9 = **15**.

The cumulative sum for the third row (Date 3) incorporates all previous sales up to that point: 6 + 9 + 13 = **28**.

The cumulative sum for the fourth row (Date 4) continues the chronological aggregation: 6 + 9 + 13 + 12 = **40**.

This accumulating pattern successfully validates the formula's logic for all subsequent rows in the table. The result confirms that the combination of CALCULATE, ALL, and EARLIER effectively executed the necessary row-level filtering required for generating an accurate chronological running total.

Alternative DAX Strategies and Modern Best Practices

While the calculated column approach utilizing EARLIER remains a highly effective method, particularly useful for specific row-level dependencies or when learning foundational DAX context transition, it is important to note its inherent limitations regarding performance. Calculated columns increase the memory footprint (RAM) of the data model and are generally slower to process than measure-based solutions. Consequently, for modern Power BI report development, measure-based running totals are significantly preferred due to their dynamic nature and superior efficiency within report visuals.

Modern DAX offers several powerful and efficient alternatives for generating running totals dynamically, often relying on time intelligence functionality. One common and robust approach involves time intelligence functions like **DATESBETWEEN** or **DATESINPERIOD**, which accurately define the date range for accumulation based on the current filter context of a visual. Additionally, advanced techniques leverage the **FILTER** function in conjunction with context modification functions such as **ALLEXCEPT** to achieve highly flexible running sums that respect specific dimensional filters.

Furthermore, Microsoft has recently introduced the highly optimized **WINDOW** function. This function provides a structured, explicit, and significantly performant way to define calculations over a specified window of rows, making it highly beneficial for complex sorting requirements, non-contiguous data sets, or complex running calculations beyond simple summation. Adopting these measure-based solutions ensures maximum responsiveness in interactive reports.

As a key best practice, the EARLIER method demonstrated here should generally be reserved for scenarios where the cumulative result must exist statically at the row level--for instance, if the running total is required as an input for a subsequent calculated column. For standard reporting and visualization needs, a measure-based solution that reacts dynamically to slicers and filters offers far greater flexibility, ensures a smaller memory footprint, and maintains a more performant data model overall.

Next Steps for DAX Mastery

The ability to calculate critical analytical measures like the Cumulative Sum is a fundamental skill for advanced data analysis and visualization in any robust BI platform. Successfully implementing running totals requires a deep understanding of DAX context concepts--specifically the distinction between row context and filter context--and how functions like CALCULATE bridge that gap. Continuous exploration and a deep dive into the broader DAX function library are essential steps toward unlocking the full analytical potential of your data models.

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