How to Create a New Table from an Existing Table Using DAX in Power BI

Authored by

stats writer

January 13, 2026

Creating new tables from existing data sources is a fundamental requirement in advanced Power BI development. While the Power Query Editor handles transformations, utilizing DAX (Data Analysis Expressions) allows for powerful, dynamic table generation based on calculated contexts and relationships. To construct a table from another table using **DAX**, one can employ various functions, primarily focusing on iteration and context manipulation. Functions such as SUMMARIZE, CALCULATETABLE, and FILTER are essential tools in this process, enabling users to group, aggregate, or subset data with precision.

The flexibility of creating calculated tables in **Power BI** offers significant advantages, particularly when dealing with complex data models or requiring intermediate tables for specific visualization needs. Unlike calculated columns, which add data vertically to an existing table, calculated tables define a completely new structure horizontally. This method is particularly useful for creating dimension tables based on distinct values in a fact table, or for performing predefined aggregations that remain static until the model refreshes. Mastering these DAX table functions is crucial for optimizing data analysis workflows and ensuring efficiency within the data model.

Leveraging SELECTCOLUMNS for Data Projection

One of the most straightforward and powerful methods for creating a new table that is merely a projection of columns from an existing table is by using the **SELECTCOLUMNS** function. This function allows developers to iterate over an existing table row by row, selecting specific columns and optionally transforming them or renaming them in the process. It is highly efficient when the goal is simply to subset the data horizontally--that is, selecting only the necessary columns without performing complex aggregations or filters that require a more robust function like **CALCULATETABLE** or **SUMMARIZE**.

The syntax for **SELECTCOLUMNS** is intuitive and requires specifying the source table followed by pairs of new column names and the corresponding expressions (usually the original column reference). This structure ensures clarity and readability, which is vital in maintaining complex **DAX** logic. The resulting table, designated by the variable name chosen by the user, exists as a stand-alone calculated table in the **Power BI** data model. This approach is superior to relying solely on Power Query for column selection when the table definition needs to be dynamically calculated based on other parameters or measures within the **DAX** environment.

The general syntax required in DAX to create a new table, often referred to as a derived table, from an existing table using column selection is demonstrated below. This specific pattern is ideal for quickly generating a simplified view of a larger, more complex table for targeted reporting or relationship building:

New_Data = SELECTCOLUMNS(
My_Data,

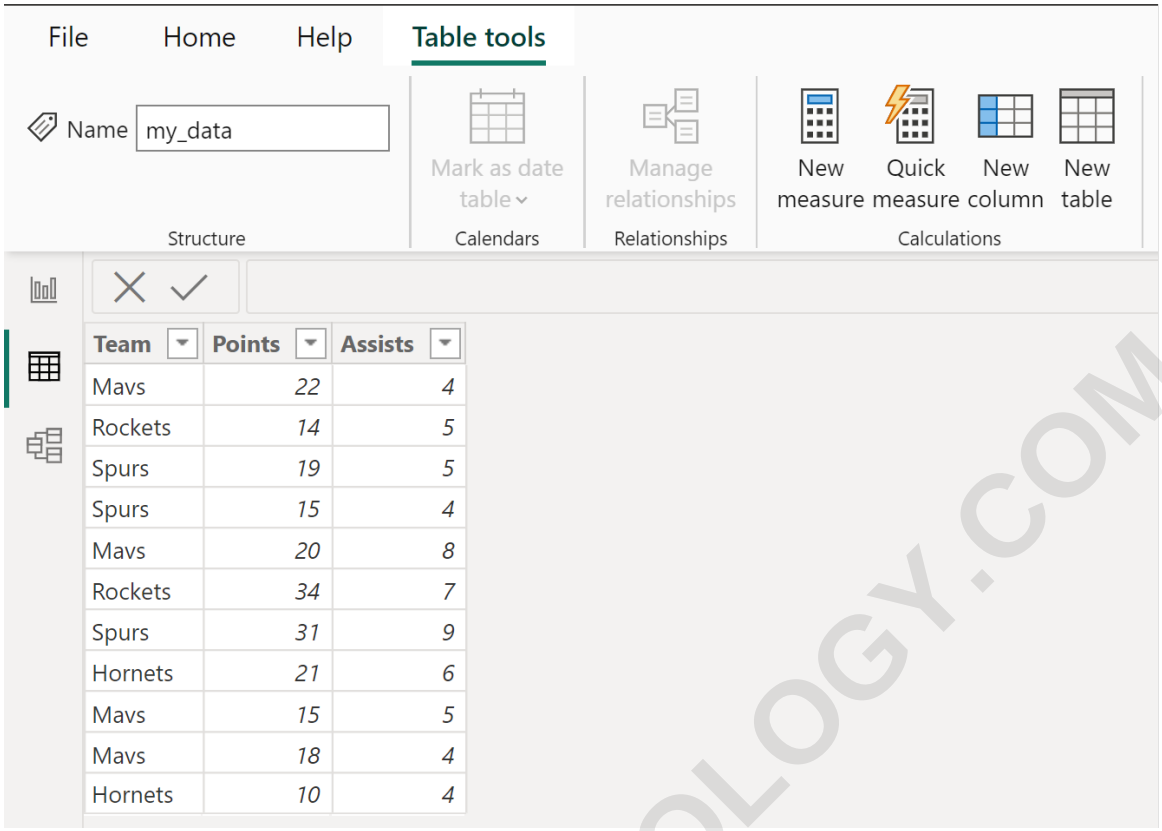
"Team", ,
"Points",)

In this example, we define a new calculated table called **New_Data**. This table is populated by iterating through the original table, **My_Data**, and extracting the values from the columns **Team** and **Points**. Crucially, the string literals ("Team" and "Points") specify the names of the columns in the newly created table, demonstrating the function's ability to rename columns immediately upon projection. This particular technique creates a new table named **New_Data** that contains only the columns named "Team" and "Points" from the existing table named **My_Data**, thereby achieving simple data subsetting.

Practical Walkthrough: Implementing SELECTCOLUMNS

To solidify the understanding of calculated table creation, let us walk through a practical scenario within the **Power BI** environment. Suppose we are analyzing sports data, and we have a primary data source containing detailed match statistics. We need a simpler table containing only the team names and their current point totals for a dashboard visualization. We begin with an existing table named **my_data**, which serves as our source of truth for the comprehensive data set.

Below is the representation of the original data table, **my_data**, illustrating the raw information available before we apply our DAX transformation:



Team	Points	Assists
Mavs	22	4
Rockets	14	5
Spurs	19	5
Spurs	15	4
Mavs	20	8
Rockets	34	7
Spurs	31	9
Hornets	21	6
Mavs	15	5
Mavs	18	4
Hornets	10	4

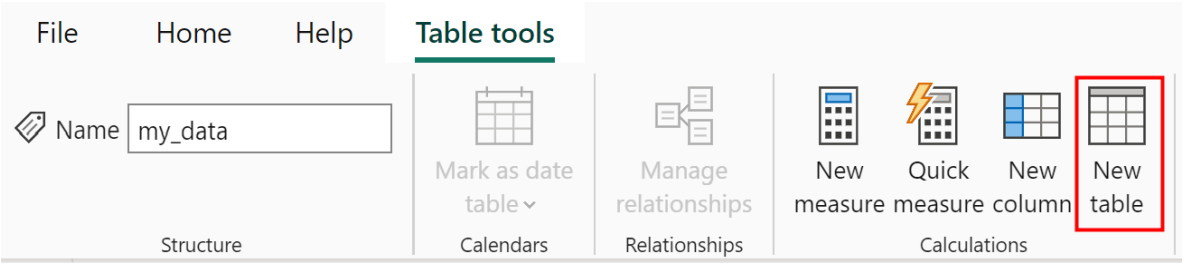
Our objective is clearly defined: to create a new, streamlined table that contains just the **Team** and **Points** columns from the **My_Data** table. This derived table will be independent of the original table structure but will inherit the data context at the time of calculation or refresh. This is a common requirement in reporting where efficiency demands that we load only essential columns into memory for specific visuals.

Steps for Creating a New Calculated Table

The creation process in **Power BI** Desktop is initiated within the modeling view. The first step involves navigating to the appropriate menu option designed for defining calculated tables. This ensures that the **DAX** expression is correctly evaluated in the table context rather than the column or measure context.

Navigate to the **Table Tools** tab within the **Power BI** ribbon interface.
Click the **New table** icon, typically found in the Calculations group.

Executing this action opens the formula bar, which is the dedicated space for entering the defining **DAX** expression for the new calculated table. This interface is crucial for ensuring the syntax is correct and the resulting table schema is properly defined based on the output of the **DAX** function used.



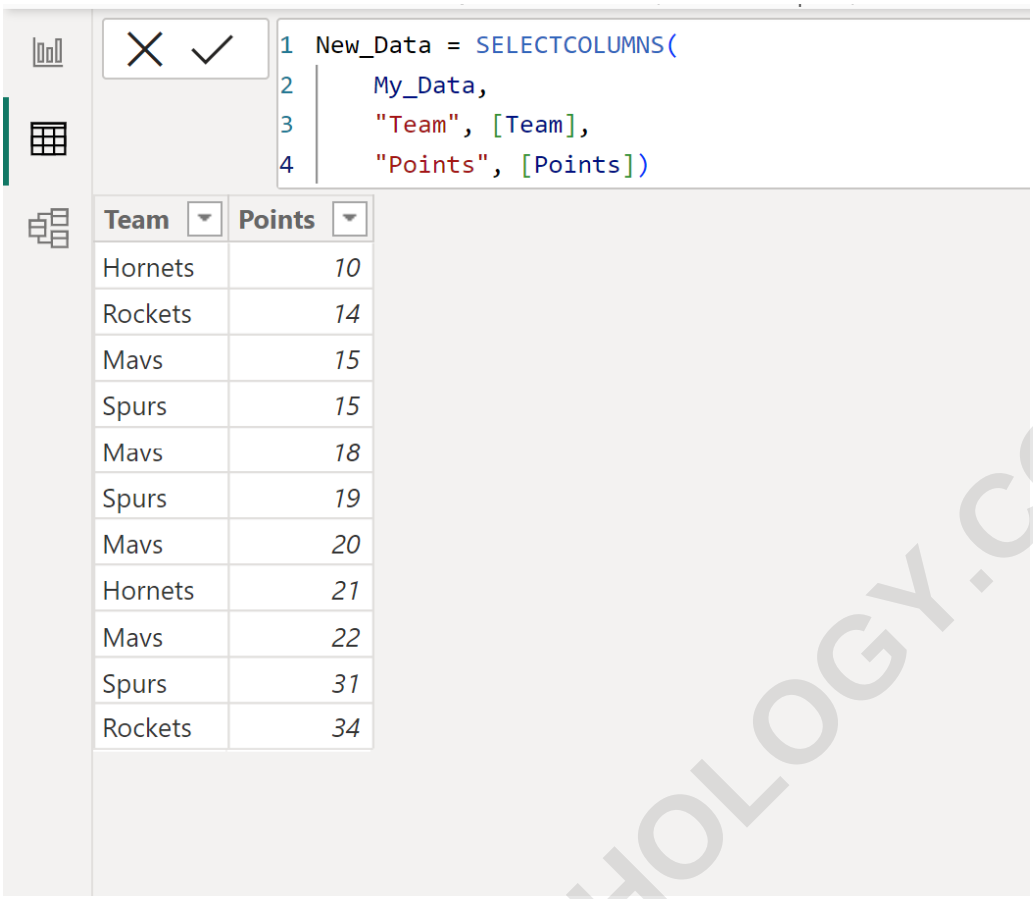
Next, we type the following formula into the formula bar. This formula utilizes the **SELECTCOLUMNS** function to project the desired columns from the source table, **My_Data**, into our new table, **New_Data**. This code explicitly dictates the input table and the exact columns to be included in the output.

```
New_Data = SELECTCOLUMNS(
My_Data,
"Team", ,
"Points", )
```

Result Verification and Column Renaming

Once the formula is entered and confirmed by pressing **Enter**, **Power BI** evaluates the DAX expression and immediately creates the new calculated table. This table, named **New_Data**, now resides within the data model, containing only the projected columns and their corresponding data rows from the original **My_Data** table. The resulting structure confirms that the **SELECTCOLUMNS** function successfully filtered the table horizontally, retaining only the **Team** and **Points** columns.

The resulting table structure, visible in the Data View of **Power BI**, confirms the successful execution of the **DAX** command. This streamlined table is now ready for use in reports or as an intermediate table for further calculations.



Team	Points
Hornets	10
Rockets	14
Mavs	15
Spurs	15
Mavs	18
Spurs	19
Mavs	20
Hornets	21
Mavs	22
Spurs	31
Rockets	34

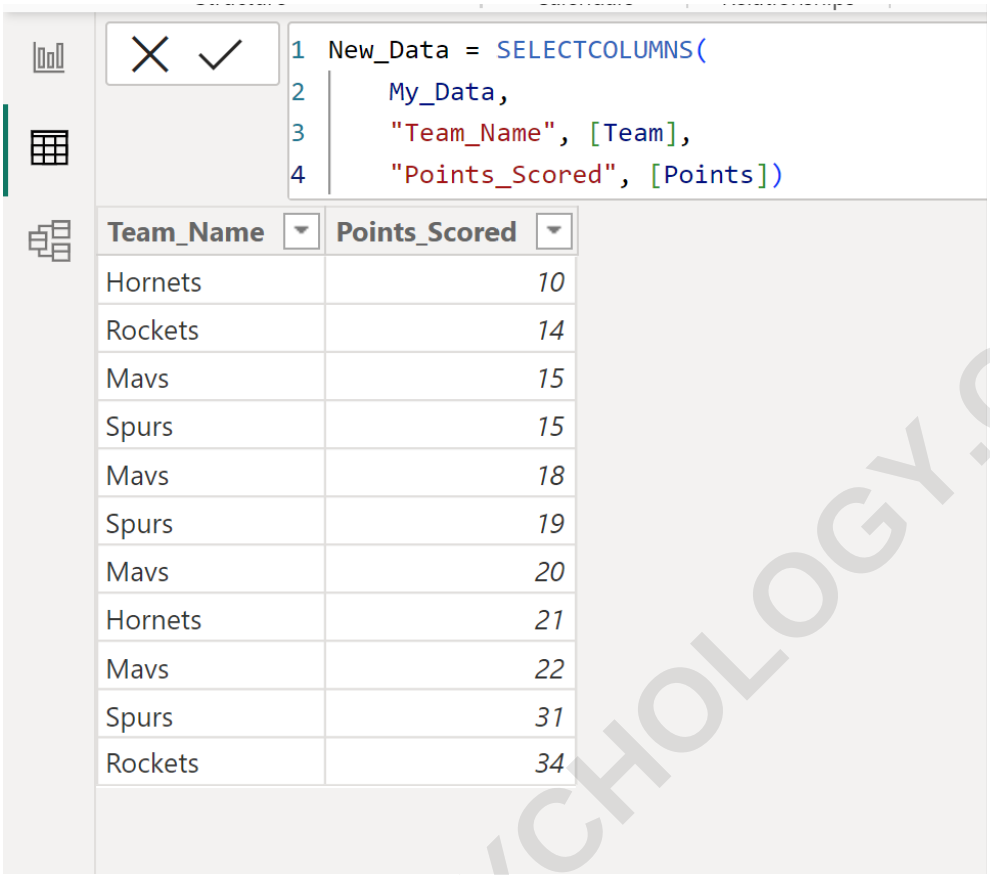
A significant advantage of the **SELECTCOLUMNS** function is the inherent ability to rename columns during the table creation process. Instead of accepting the original column names, users can specify new, more descriptive names as the first argument in each column pair within the function call. This avoids the necessity of adding an extra step later to rename columns, streamlining the data preparation phase.

For instance, if we wanted to make the column names more explicit for reporting purposes, such as renaming **Team** to **Team_Name** and **Points** to **Points_Scored**, we would modify the syntax slightly. This practice enhances model clarity and ensures that column headers are easily understandable by end-users accessing the reports. The following syntax demonstrates how to incorporate column renaming directly into the table creation formula:

New_Data = SELECTCOLUMNS(
My_Data,
"Team_Name", ,
"Points_Scored",)

The result of using this revised DAX formula is a new calculated table where the column headers

are updated according to the specified string literals, reflecting the enhanced clarity of the data fields.



```
New_Data = SELECTCOLUMNS(
    My_Data,
    "Team_Name", [Team],
    "Points_Scored", [Points])
```

Team_Name	Points_Scored
Hornets	10
Rockets	14
Mavs	15
Spurs	15
Mavs	18
Spurs	19
Mavs	20
Hornets	21
Mavs	22
Spurs	31
Rockets	34

Note: It is highly recommended to consult the complete documentation for the SELECTCOLUMNS function in **DAX** to understand all its capabilities, especially when dealing with complex expressions instead of simple column references.

Advanced Table Creation Techniques: Summarization and Aggregation

While **SELECTCOLUMNS** is excellent for simple projection, creating dimension tables or performing advanced data analysis often requires summarizing data based on specific criteria. The SUMMARIZE function is the cornerstone for achieving aggregation within DAX calculated tables. This function allows developers to define grouping columns and then apply aggregation functions (like SUM, AVERAGE, or COUNT) to other columns within those groups.

The typical use case for **SUMMARIZE** involves creating summarized tables that reduce the granularity of the original data. For example, instead of listing every transaction, one could summarize sales data by Region and Product Category, calculating the total revenue for each combination. This significantly simplifies the data structure for high-level reporting. However,

developers must be mindful of the context transition that occurs within the aggregate expressions used inside **SUMMARIZE**, ensuring that the results accurately reflect the desired calculations.

Alternatively, the **SUMMARIZECOLUMNS** function (often preferred in modern **DAX** development) provides a more robust and flexible way to achieve summarization, especially when integrating filters. Although the older **SUMMARIZE** function is still valid, understanding its mechanisms is vital. Both functions empower the user to define a new table structure that includes only the grouping columns and the calculated measure results, efficiently handling aggregation logic directly within the data model definition.

Using CALCULATETABLE for Context Manipulation and Filtering

When the requirement shifts from simple column projection or aggregation to applying complex filters or context manipulation before table output, the CALCULATETABLE function becomes indispensable. Analogous to the **CALCULATE** function used for measures, **CALCULATETABLE** allows users to evaluate a table expression within a modified filter context. This enables highly specific data subsetting based on complex logical conditions.

CALCULATETABLE takes an existing table (or another table expression) as its first argument and is followed by one or more filter arguments. These filters can leverage any valid **DAX** expression, including functions like FILTER, to define the exact rows that should be included in the new calculated table. For instance, one could create a table containing only sales data from the previous fiscal year, or data pertaining only to customers in a specific geographical region.

The power of **CALCULATETABLE** lies in its ability to override existing filter contexts or introduce new ones dynamically. This is crucial for advanced modeling where data lineage and context preservation are key. While **SELECTCOLUMNS** excels at horizontal subsetting (column selection), **CALCULATETABLE** excels at vertical subsetting (row selection based on complex criteria), making it a cornerstone function for generating targeted fact or dimension tables.

Integrating FILTER with Table Functions

The FILTER function is frequently used in conjunction with **CALCULATETABLE** or other table-returning functions to precisely define the subset of rows required. The FILTER function takes a table and a boolean expression, returning a new table that contains only the rows for which the expression evaluates to true. This allows for conditional row selection that is necessary for generating derived tables based on specific business rules.

For example, to create a table containing only high-scoring teams from our **My_Data** table (Points > 100), the structure would involve using the **FILTER** function. While this can often be simplified using filter arguments in **CALCULATETABLE**, understanding the explicit use of **FILTER** is vital as

it forms the basis for complex filtering logic within **DAX** iterators and table construction. The use of **FILTER** ensures that the resulting calculated table is a clean, filtered subset of the original data, perfectly tailored to specific analytical needs.

Summary of DAX Table Creation Methods

In summary, generating a table from an existing one in **Power BI** using DAX involves choosing the right function based on the intended outcome:

For **Column Projection and Renaming**: Use **SELECTCOLUMNS**. This is ideal for simplifying a wide table by extracting only necessary fields.
For **Aggregation and Grouping**: Use **SUMMARIZE** (or **SUMMARIZECOLUMNS**). This is necessary for creating dimension tables or high-level summary reports.
For **Row Filtering and Context Modification**: Use **CALCULATETABLE**, often coupled with the **FILTER** function, to apply complex row-level logic and dynamically change the evaluation context.

By judiciously applying these powerful **DAX** functions, developers can easily create new tables with aggregated or filtered data from an existing table, significantly saving time and effort in advanced data modeling and analysis within the **Power BI** environment. These derived tables serve as stable, calculated components crucial for maintaining efficiency and accuracy in complex data environments.