Learning DAX: Creating New Tables from Existing Tables in Power BI

Authored by

Mohammed looti

November 12, 2025

RECOMMENDED CITATION

Mohammed looti (2025). *Learning DAX: Creating New Tables from Existing Tables in Power BI*. PSYCHOLOGICAL STATISTICS. Retrieved from https://statistics.arabpsychology.com/?p=17261

The Strategic Role of Calculated Tables in Power BI

The ability to dynamically generate new tables from existing data sources is not just a convenience; it is a fundamental requirement for **advanced data modeling** within the Power BI ecosystem. Analysts frequently encounter raw datasets that contain numerous columns, many of which are superfluous for specific analytical tasks or visualization needs. By creating a derived or calculated table, practitioners can drastically streamline the data structure, isolate only the relevant fields, and subsequently boost calculation efficiency across the entire model. This strategic approach leads to cleaner, more manageable models where measures and visualizations operate exclusively on the necessary subset of information.

This critical transformation process is governed by DAX, or **Data Analysis Expressions**, which provides a robust and complex set of functions necessary for advanced data manipulation. While many initial structural transformations--such as merging or simple filtering--are often best handled in Power Query (M language), calculated tables using DAX become essential when the new table must reference existing measures, derived calculated columns, or when the transformation logic is inherently analytical rather than purely structural. This distinction determines whether the transformation is performed at the ETL stage or within the semantic model itself.

Calculated tables offer a unique advantage over simple table views in that they are materialized in the model's memory during the data refresh process. This means they exist as distinct entities in the data model, capable of forming relationships with other tables, serving as dimension tables, or providing input for complex analytical measures. Understanding when to leverage this capability is a hallmark of sophisticated data modeling in Power BI.

Defining the DAX Syntax using SELECTCOLUMNS

To construct a new, calculated table in Power BI that projects specific columns from an existing source, we primarily rely on the highly versatile SELECTCOLUMNS function. This function is specifically engineered to iterate over the rows of an input table and return a resultant table containing only the specified columns. Crucially, these output columns can either be direct references to existing fields or entirely new calculated expressions derived during the iteration.

The core syntax for this operation is intuitive yet powerful. It necessitates designating the name of the new calculated table, followed by the source table reference, and then a sequential, paired list of the desired columns. Each pair must strictly adhere to the structure: the desired column name for the new table (enclosed in double quotes) followed immediately by the expression that defines its content. In the simplest case, this expression is a direct column reference from the source table, enclosed in square brackets.

The fundamental structure below demonstrates how DAX is utilized to create a filtered column

projection from an existing table within the Power BI environment:

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

This specific example defines a new table named **New_Data**. This table will contain only two columns, "Team" and "Points," which are directly sourced from the identically named columns within the existing table, **My_Data**. This formula represents the most efficient way to generate a simplified dimension or fact table when the goal is purely to narrow the column count while retaining the original row context.

Deep Dive into the SELECTCOLUMNS Function

The SELECTCOLUMNS function is indispensable for analysts focused on precise data modeling. It operates with an implicit row context, evaluating the expression provided for each output column for every row in the input table. A crucial behavioral characteristic is that, unlike aggregation functions such as `SUMMARIZE`, SELECTCOLUMNS does not inherently group or aggregate data. It meticulously maintains the original **cardinality** (the row count) of the source table, making it the perfect choice for creating filtered or narrowed versions of existing tables without altering the underlying granularity.

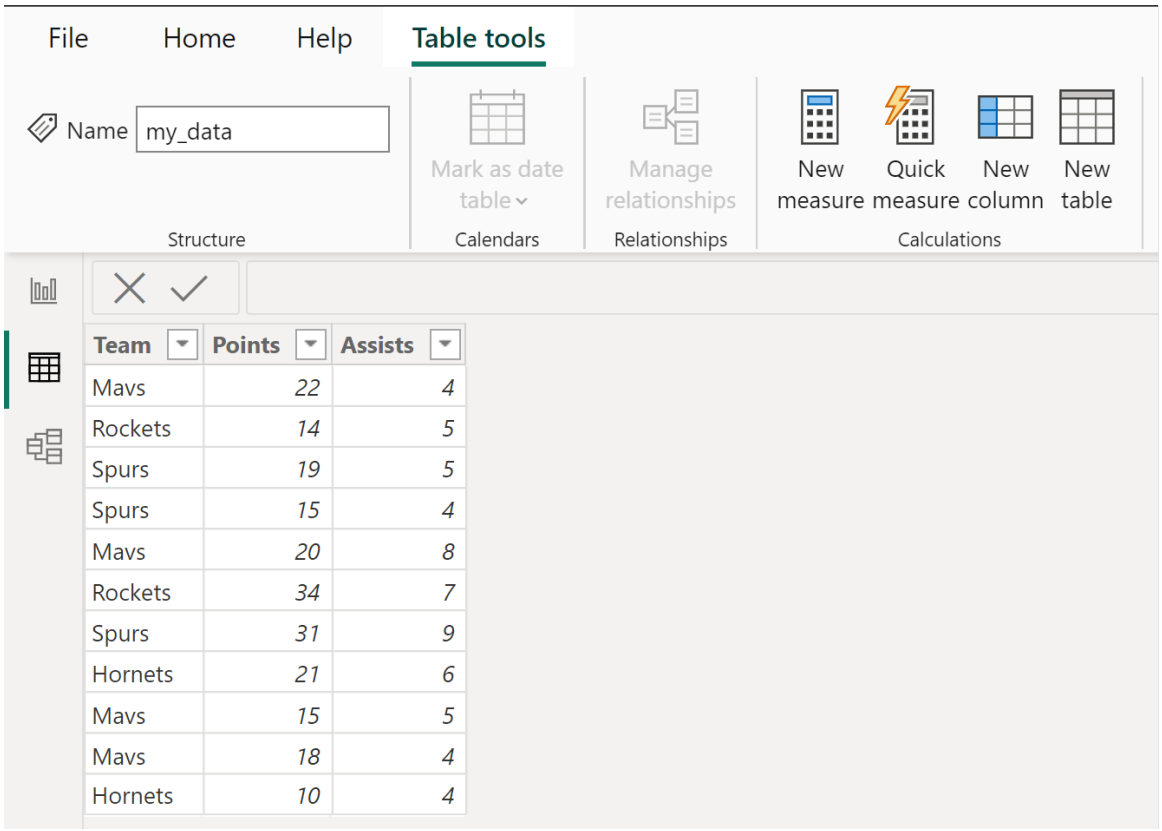
The primary advantage of employing SELECTCOLUMNS lies in its exceptional flexibility. While the most straightforward use case involves simply projecting existing columns, its true power emerges when incorporating complex calculated expressions. For instance, an analyst could define a new column within the resulting table that calculates the ratio of points to games played, or applies a conditional logic (like an `IF` statement), all defined seamlessly within the same DAX formula. This powerful combination of projection and calculation makes it an extremely valuable tool for data preparation directly within the Power BI semantic layer.

It is vital to distinguish this function from its frequently confused counterpart, `ADDCOLUMNS`. While `ADDCOLUMNS` is used to append new calculated columns onto an existing table, SELECTCOLUMNS defines the new table entirely from scratch. It includes **only** the columns explicitly specified in the argument list, effectively discarding all others. If the objective is to dramatically reduce the column count of a wide table, SELECTCOLUMNS is the preferred, most explicit, and cleanest method available.

Practical Implementation: Subsetting Data in Power BI

To fully grasp the mechanics of table creation, we will walk through a practical implementation using a sample dataset. Assume we have an existing table in Power BI named **My_Data**. This comprehensive table contains various metrics, but for a specific reporting requirement, we only need the team identifiers and their corresponding scores.

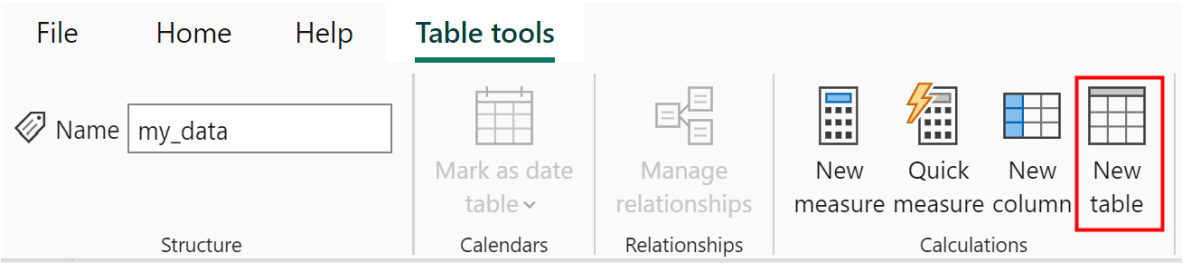
The visual representation below illustrates the original structure of the **My_Data** table, highlighting the presence of multiple columns that are not necessary for our current reporting objective:



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 goal is precise: create a new, static table containing only the **Team** and **Points** columns from the source table. This new table will be highly efficient for use in relationships or for visualizations that demand a narrowed focus, as it is calculated once during data refresh.

To begin the table creation process, navigate to the **Table Tools** ribbon in the Power BI interface. Locate and click the **New table** icon. This action immediately opens the formula bar, enabling the input of the desired DAX definition for your calculated table. The illustration below pinpoints the location of the **New table** button within the ribbon:

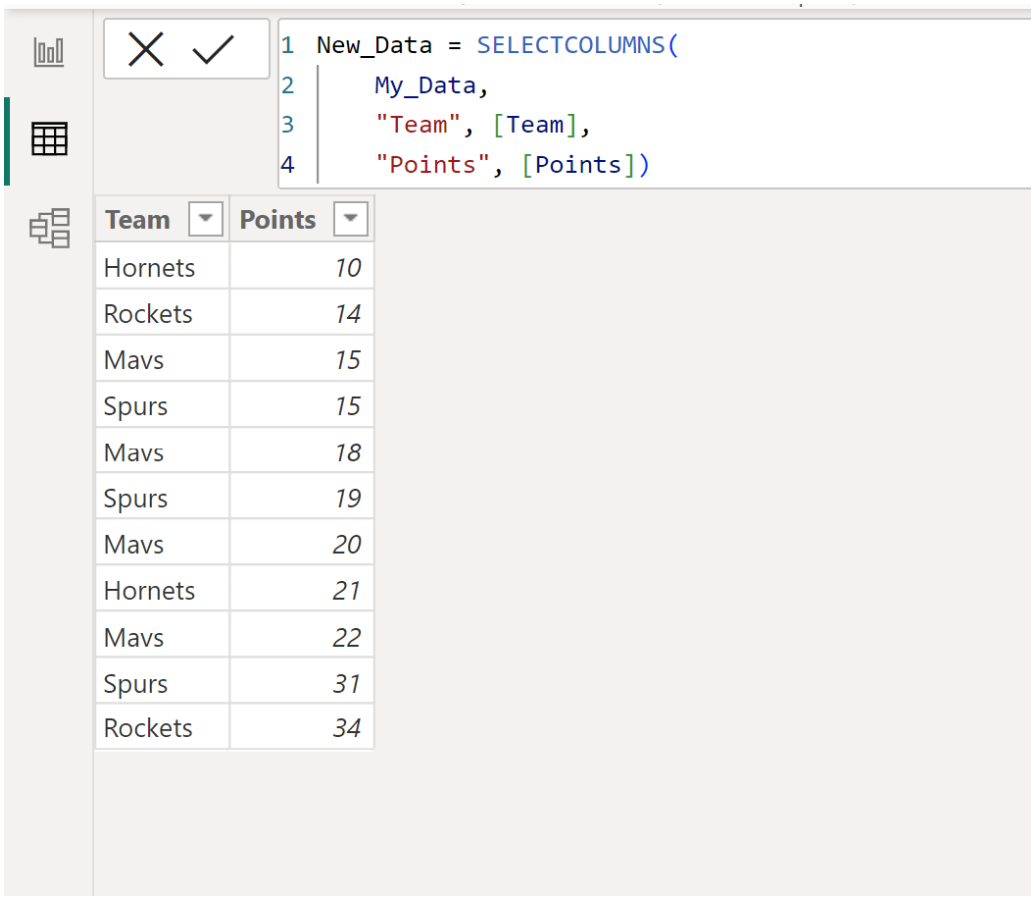


Next, input the defining DAX formula into the activated formula bar. Remember the syntax convention: the new table name must be declared first, followed by the assignment operator (=), and then the function call and its arguments:

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

Upon execution, Power BI processes the formula, and a new calculated table named **New_Data** is instantly generated. This table serves as a projected subset of **My_Data**, containing only the selected fields.

The resulting table, **New_Data**, successfully demonstrates the selection and projection of only the required fields, significantly simplifying subsequent data modeling efforts:



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

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

Customizing Output Names: Renaming Columns

One of the most practical and useful features of the `SELECTCOLUMNS` function is the capability to rename columns simultaneously as they are being created in the new table. This feature is indispensable for enforcing organizational naming conventions, enhancing report readability, or clearly differentiating fields when pulling similar columns from multiple source tables during intricate data modeling scenarios.

The argument structure of `SELECTCOLUMNS` naturally accommodates this renaming process. The key lies in the column pair structure: the first element is always the desired, user-defined name for the column in the new output table (the text string enclosed in quotes), and the second element is the actual expression or column reference from the source table (the input data). This explicit pairing ensures clarity regarding the transformation.

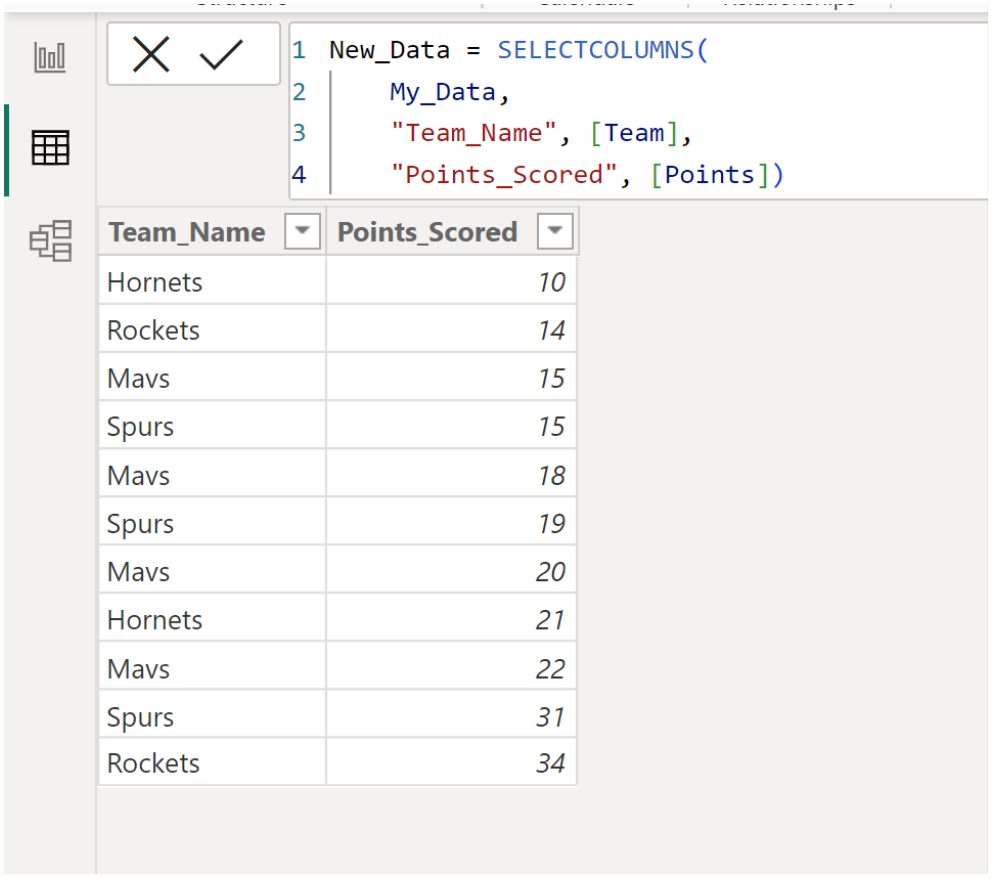
For example, instead of retaining the generic column names "Team" and "Points," which might lack context, we can define more descriptive labels such as **Team_Name** and **Points_Scored** in our new table. This clarification greatly assists end-users and collaborators in immediately understanding the context and content of the data.

We apply the following modified syntax to rename the columns while projecting them from the **My_Data** table:

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

By executing this adjusted formula, the resulting calculated table maintains the exact row count and data integrity of the source but presents the fields under the new, descriptive names, thus improving the overall usability of the derived dataset.

The final output confirms these successful structural and naming changes, resulting in a cleaner and more standardized table structure optimized for reporting:



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

Performance and Strategic Considerations

While calculated tables created using DAX provide immense flexibility and power, analysts must critically evaluate their implications for model performance and memory usage. It is essential to

remember that calculated tables are **static**; they are materialized and stored entirely in the model's memory (the VertiPaq engine) during every data refresh cycle. If the source table is extremely large, creating a calculated table--even a narrowed one--will consume significant storage space and invariably increase the overall refresh duration for the report.

The strategic decision to use a calculated table versus performing the transformation in Power Query (M language) should always hinge on the nature of the calculation. If the calculation requires access to the semantic model context, such as referencing existing measures or needing the resulting table to interact dynamically with complex DAX filter contexts, then a calculated table is necessary. Conversely, if the transformation is purely structural--a simple column projection, filtering, or merging--and does not rely on advanced analytical functions, the most performant approach is almost always to handle it during the Extract, Transform, Load (ETL) phase using Power Query.

However, calculated tables remain indispensable for specific advanced scenarios. These include the generation of disconnected slicer tables (tables that do not filter the main model), creating robust and complex date dimension tables, or defining highly specific subsets of data that must exist as independent dimensions within the model. They represent a powerful, complementary layer of transformation that extends far beyond the structural capabilities of the query editor, providing the analytical flexibility needed for complex business intelligence solutions.

Note: For detailed technical specifications, syntax variations, error handling, and in-depth performance considerations regarding the `SELECTCOLUMNS` function, always consult the complete and authoritative documentation provided by Microsoft.

Continuing Your Journey to DAX Mastery

Mastering the creation and manipulation of tables is merely one stepping stone on the path to proficiency in Power BI and advanced DAX development. Continued education focusing on topics such as filtering context, aggregation techniques, and relationship management is critical for building robust, scalable, and efficient business intelligence solutions.

To further enhance your skills and explore the extensive capabilities of the DAX language, consider studying the following advanced topics and functions:

Understanding the Context Transition concept in DAX.
Utilizing the `FILTER` function effectively to create dynamic subsets of data.
Differentiating between the appropriate use cases for calculated columns versus measures for aggregation tasks.
Advanced techniques for time intelligence using the built-in DAX functions (e.g., `DATESYTD`, `DATEADD`).