How to Add a Column from Another Table in Power BI Easily

Authored by

mohammed looti

January 10, 2026

Integrating data across multiple tables is a fundamental requirement for building robust analytical models in Power BI. Analysts often encounter scenarios where a crucial piece of information--such as a category, description, or grouping metric--resides in a secondary dimension table but is needed directly within the primary fact table. Addressing the question, "How can I add a column from another table in Power BI?" involves understanding two primary, powerful approaches: the data transformation method via **Merge Queries** in Power Query, and the data modeling method via DAX calculated columns using functions like **LOOKUPVALUE**.

The choice between these methods depends heavily on where the data manipulation should occur in the data flow pipeline. Should the transformation happen during the Extraction, Transformation, and Loading (ETL) phase before the data model is built (using Power Query), or should it be handled dynamically within the data model using DAX calculations? Both techniques are highly effective for consolidating columns, but they have distinct performance implications and usability characteristics that experts must consider.

While relational modeling with standard table relationships is the preferred method for linking tables in Power BI, there are specific, valid use cases where physically adding a column is necessary. This often occurs when preparing source data that must be highly denormalized for specific visualizations, or when the relationship between tables is complex and requires custom logic that is easier to manage within a single column context. We will first explore the robust ETL approach using Merge Queries before diving into the flexibility offered by DAX.

Method 1: Utilizing the Power Query Editor with Merge Queries (The ETL Approach)

The **Merge Queries** function, accessible within the Power Query Editor, is the standard, data-shaping approach to combining data from two different sources based on a common key. This process is analogous to performing a `JOIN` operation in SQL. When you merge queries, you are physically transforming the data before it loads into the Power BI Data Model, ensuring that the merged column is available for direct use without requiring subsequent DAX calculations.

This method is generally preferred for creating static, non-calculated relationship links, especially when dealing with large datasets, as the heavy lifting is performed during the data refresh phase. By integrating the column during the ETL process, you optimize the data model for faster report execution and simpler measure creation. The merged column becomes a native part of the destination table, streamlining subsequent data visualization and analysis steps.

To initiate this process, you must enter the **Power Query Editor**, which serves as the powerful transformation layer of Power BI Desktop. It is crucial to identify which table is the primary table (the one receiving the new column) and which is the secondary table (the source of the column). A successful merge relies on having at least one common field--the join key--that uniquely identifies

the rows that should be matched across both tables.

Step-by-Step Guide to Merging Queries

The process of merging involves several steps, ensuring a clean and effective transformation. This is the official and most scalable way to perform cross-table column integration in the ETL phase.

Access the Power Query Editor: Navigate to the "Home" tab in Power BI Desktop and click "Transform data" (or "Edit Queries" in older versions) to open the Power Query Editor window.

Select the Primary Table: In the list of queries (tables) on the left-hand side, select the query (Table A) that you wish to add the new column to. This table will be the recipient of the merged data.

Initiate the Merge Operation: On the "Home" tab of the Power Query Editor ribbon, locate the "Combine" group and click on the **Merge Queries** option. You have two choices: "Merge Queries" (which modifies the current Table A) or "Merge Queries as New" (which creates a new table combining both). Typically, for adding a column, you select the former.

Configure the Merge Dialog: In the Merge dialog box, the primary table (Table A) is already selected at the top. Use the drop-down menu in the bottom half to select the secondary table (Table B) that contains the desired column. Next, click on the common column in Table A and then click on the corresponding common column in Table B to establish the join key. Ensure you select the appropriate **Join Kind** (e.g., Left Outer is common for adding dimension attributes).

Execute and Expand the Column: After clicking "OK," a new column will be added to Table A. This new column contains a set of nested table values, representing the matched rows from Table B. To extract the specific attribute you need (the column you want to add), click on the small **expand icon** (a double-headed arrow) located in the header of this new merged column. A dialog box will appear allowing you to select only the required column(s) to include in your original table. After selection, uncheck the box labeled "Use original column name as prefix" if you prefer a clean column name.

Upon expanding the column and applying the changes, the selected column from the source table is successfully integrated into your primary table. Remember to click "Close & Apply" back in the Power BI Desktop environment to load the transformed data model.

Method 2: Creating a Calculated Column using the DAX LOOKUPVALUE Function

While **Merge Queries** handles data integration during the ETL phase, the DAX approach allows

you to achieve the same result dynamically within the data model using a **Calculated Column**. The easiest and most common way to add a column based on a matching key using DAX is by employing the **LOOKUPVALUE** function. This function searches a value in one column of a table and returns the corresponding value from another column in the same table, provided certain criteria are met.

The syntax of **LOOKUPVALUE** is specifically designed to perform this lookup operation efficiently, requiring three primary arguments: the result column you want to retrieve, the search column in the source table, and the value to match against. This method bypasses the need to modify the underlying data source or utilize Power Query, making it ideal for analysts who prefer to work directly within the model view.

The primary advantage of using a calculated column with **LOOKUPVALUE** is its immediacy and relative simplicity compared to navigating the complex environment of the Power Query Editor. However, it is crucial to remember that calculated columns consume memory (RAM) within the model, as the result of the calculation is stored for every row. Therefore, while convenient for smaller lookups or proof-of-concept models, heavy reliance on calculated columns for complex operations can impact model performance.

The easiest way to add a column from another table in Power BI is to use the **LOOKUPVALUE** function as follows:

Conf = LOOKUPVALUE(data2, data2, data1)

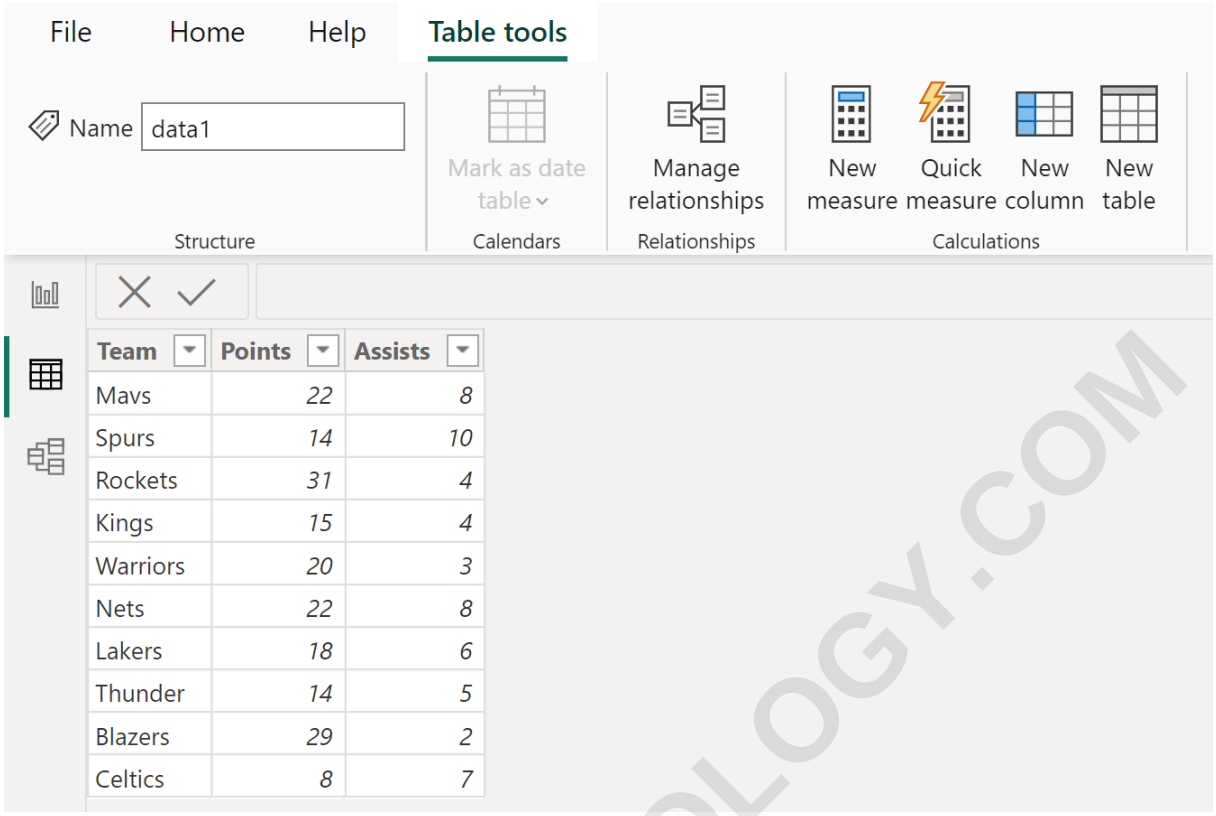
This particular example adds the column named **Conference** from the table named **data2** to the table named **data1** based on matching the values from the **Team** columns between the two tables. Specifically, the function iterates through the rows of **data1**, finds the corresponding value in `data2`, and returns the related value from `data2`.

The following example shows how to implement this powerful DAX technique in practice.

Practical Example: Implementing LOOKUPVALUE in Power BI

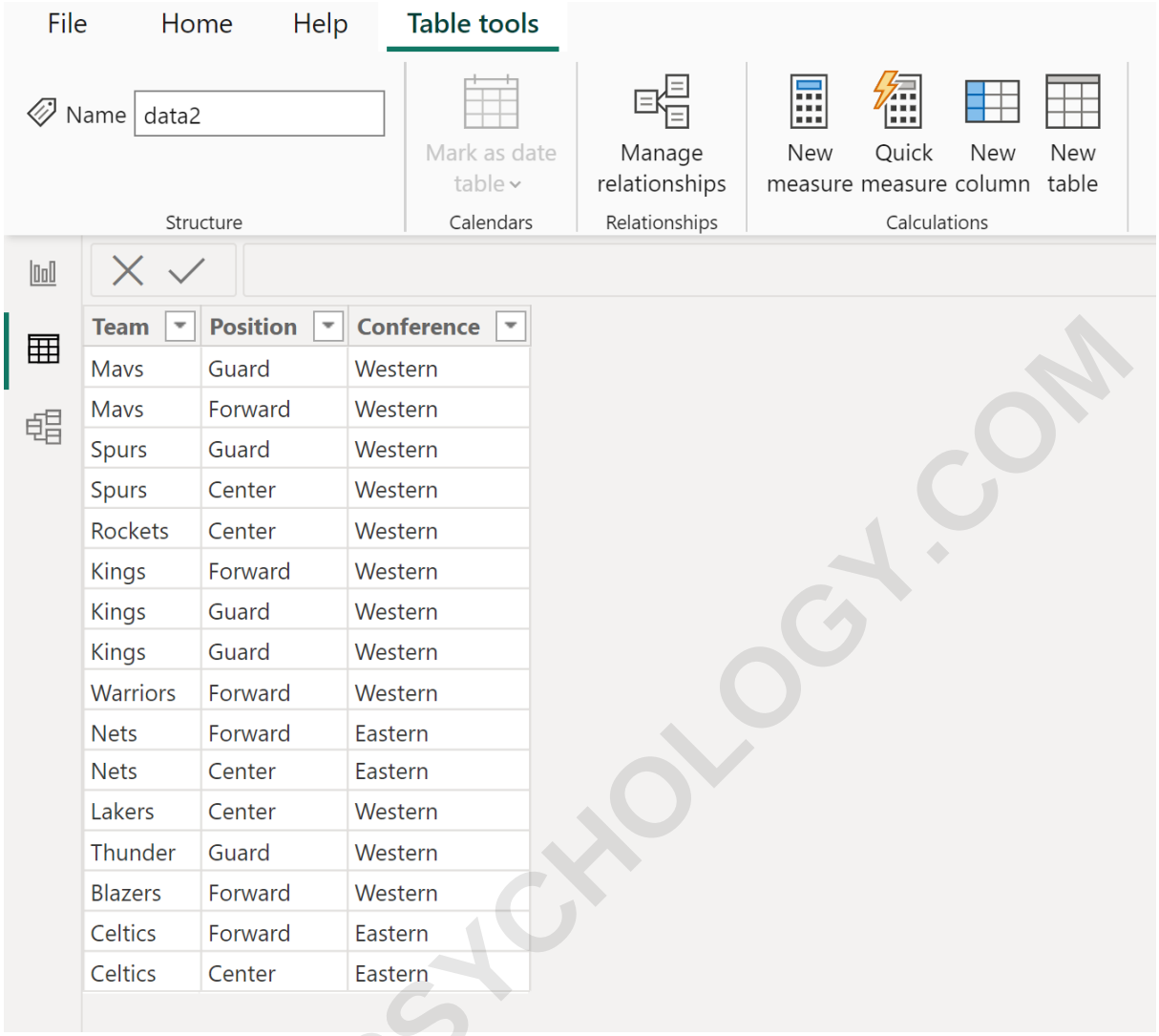
Consider a scenario where we have two distinct datasets loaded into our Power BI model. We need to enrich the primary transactional data with categorical information from a secondary dimension table.

Suppose we have the following primary table in Power BI named **data1**:



Team	Points	Assists
Mavs	22	8
Spurs	14	10
Rockets	31	4
Kings	15	4
Warriors	20	3
Nets	22	8
Lakers	18	6
Thunder	14	5
Blazers	29	2
Celtics	8	7

And suppose we have another dimension table named **data2**, which contains the conference information:

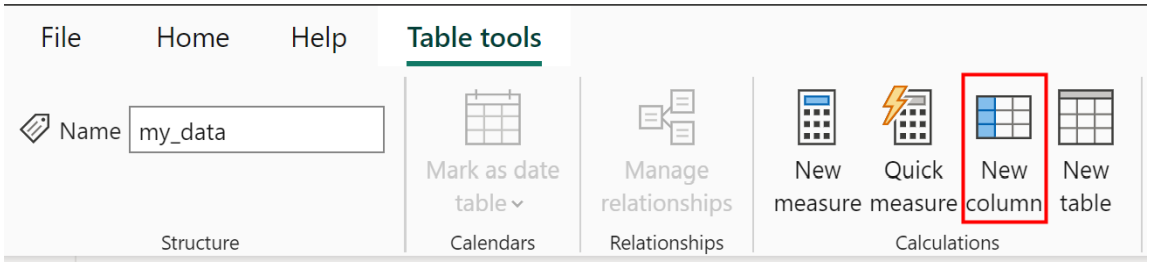


Team	Position	Conference
Mavs	Guard	Western
Mavs	Forward	Western
Spurs	Guard	Western
Spurs	Center	Western
Rockets	Center	Western
Kings	Forward	Western
Kings	Guard	Western
Kings	Guard	Western
Warriors	Forward	Western
Nets	Forward	Eastern
Nets	Center	Eastern
Lakers	Center	Western
Thunder	Guard	Western
Blazers	Forward	Western
Celtics	Forward	Eastern
Celtics	Center	Eastern

The objective is clear: we would like to add the column named **Conference** from the table named **data2** to the table named **data1**. This linkage must be established by accurately matching the values found in the shared **Team** columns between the two tables. This ensures that each team in **data1** is correctly associated with its respective conference listed in **data2**.

To execute this calculated column addition, we must first ensure that the **data1** table is active within the Data View or Report View. Then, we initiate the creation of a new column. This action signals to Power BI that we intend to define a persistent, row-context calculation using DAX.

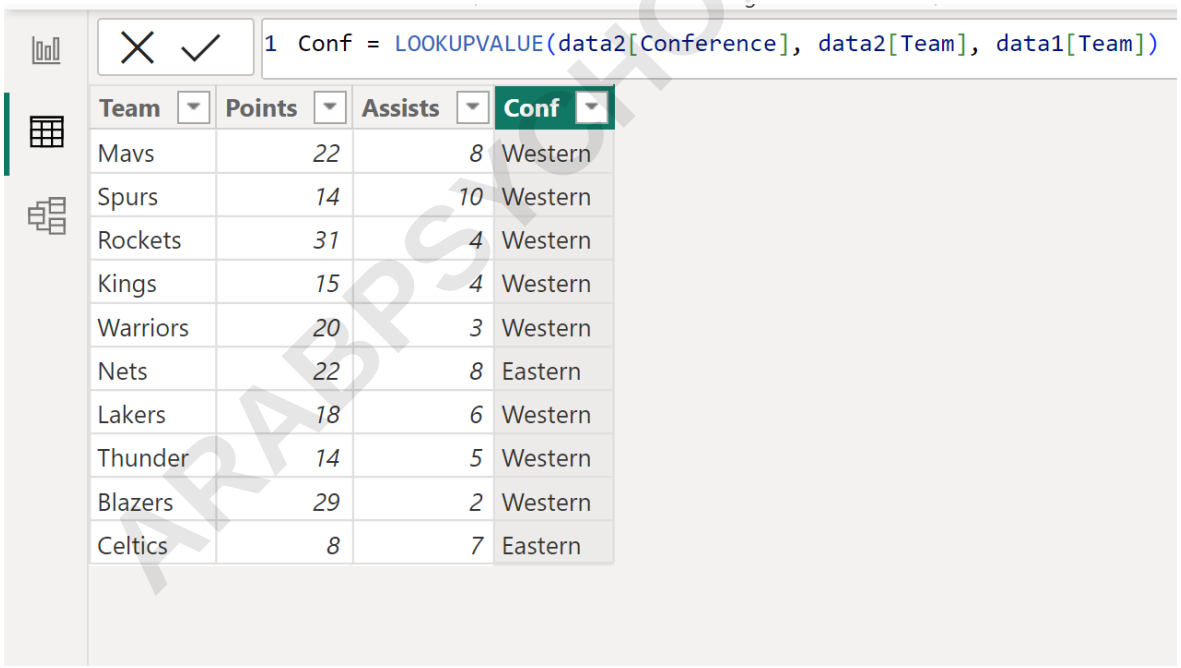
The operational steps are straightforward. First, select the **data1** table, then click the **Table tools** tab in the ribbon, and finally click the **New column** icon:



This action will open the DAX formula bar. Into this bar, we type the precise formula using the **LOOKUPVALUE** function, specifying the columns for the result, the lookup key, and the value to match against:

Conf = LOOKUPVALUE(data2, data2, data1)

Once the formula is confirmed, Power BI processes the calculation row by row, resulting in a new column named **Conf**. This column successfully contains the conference values retrieved from **data2**, now appended directly to the corresponding rows in **data1**:



Team	Points	Assists	Conf
Mavs	22	8	Western
Spurs	14	10	Western
Rockets	31	4	Western
Kings	15	4	Western
Warriors	20	3	Western
Nets	22	8	Eastern
Lakers	18	6	Western
Thunder	14	5	Western
Blazers	29	2	Western
Celtics	8	7	Eastern

This method provides instantaneous data enrichment directly within the model environment, offering flexibility for immediate analysis.

Important Considerations for LOOKUPVALUE

While **LOOKUPVALUE** is highly effective, users must be aware of its limitations, particularly

regarding data uniqueness. The function is specifically designed to handle one-to-one or many-to-one relationships (i.e., retrieving a unique value). If the lookup criteria results in more than one matching row in the secondary table (a many-to-many scenario), the function will return an error, as it cannot determine which value to select.

For scenarios involving complex many-to-many relationships or when multiple potential matches exist, alternative DAX functions such as `CALCULATE` combined with `RELATED` or aggregation functions like `MAXX` or `MINX` might be required. Furthermore, relying on **LOOKUPVALUE** creates a dependency on the underlying data model relationships, even if those relationships aren't formally defined, which is important for maintenance.

Note: You can find the complete documentation for the **LOOKUPVALUE** function in DAX official documentation. Understanding the precise behavior, especially regarding non-matching values (which return BLANK), is essential for accurate modeling.

When to Choose DAX vs. Power Query (Merge vs. Lookup)

Deciding whether to use **Merge Queries** in Power Query or a DAX **Calculated Column** is a crucial decision that impacts data refresh performance, model size, and flexibility. Generally, experts recommend utilizing Power Query for static data consolidation (Merge) and reserving DAX for dynamic calculations (Measures).

The **Power Query (Merge Queries)** approach is highly efficient for data transformation because it happens during the data load process. The resulting table that lands in the Power BI model is already clean and denormalized. This approach minimizes the subsequent workload on the DAX engine and reduces the memory footprint of the model, making it the best practice for large, stable datasets where the source column is essentially an attribute of the target table.

Conversely, the **DAX (LOOKUPVALUE)** approach is most appropriate when the logic for adding the column is complex, dynamic, or depends on filtering conditions that are better handled in the data model. If the lookup column is only required temporarily or needs to be calculated based on other existing calculated fields, DAX offers greater flexibility. However, every calculated column increases the storage required by the model and recalculates on every full data refresh, potentially slowing down the process if many complex lookups are implemented.

Summary of Best Practices

To ensure optimal Power BI model design, follow these guidelines when deciding how to integrate columns:

Use **Merge Queries** for simple, non-dynamic lookups that are part of the core data structure (e.g.,

adding a product category ID to a sales fact table). This keeps the model lean and fast.

Avoid using calculated columns for simple lookups if the data can be merged in Power Query, especially in very large models.

If you need dynamic results that respond to filter contexts in a report, create a **Measure** using DAX functions (like `CALCULATE`) instead of a calculated column. Measures are calculated on the fly and do not increase model size.

Only use **LOOKUPVALUE** when the relationship is guaranteed to be one-to-one or many-to-one, and when the column is strictly necessary within the row context of the target table.

Conclusion and Further Reading

Adding a column from another table in Power BI is a common requirement in data preparation, handled effectively by both the robust ETL capabilities of **Merge Queries** and the dynamic modeling power of DAX **LOOKUPVALUE**. By understanding the operational differences--ETL vs. Data Model--analysts can select the technique that best optimizes model performance and scalability.

For complex modeling tasks, always prioritize creating explicit relationships between tables in the Model View, as this is the foundational principle of star schemas and efficient data modeling in Power BI. Physical column addition should be reserved for specific scenarios where denormalization is unavoidable or where a quick, static lookup is required.

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

Placeholder: How to combine tables using `UNION` in DAX.
Placeholder: Advanced data cleaning techniques in Power Query.
Placeholder: Creating dynamic measures using `CALCULATE`.