How to Join Two Tables in Google Sheets Query?

Authored by

stats writer

November 23, 2025

The ability to efficiently combine disparate datasets is fundamental to modern data analysis. While the QUERY function in **Google Sheets** offers powerful SQL-like capabilities for filtering and aggregating data, users frequently encounter a significant challenge when attempting to perform standard relational database operations: the lack of a direct, built-in `JOIN` command.

Many users familiar with SQL databases expect to use a straightforward `JOIN` syntax to merge information from two separate tables based on a shared key. However, the data manipulation language supported by the QUERY function within Google Sheets is fundamentally limited in this regard. This limitation necessitates the use of complex array formulas, combining functions like VLOOKUP and ArrayFormula, to effectively simulate the process of joining two tables. This method, while complex in appearance, provides a robust workaround for integrating datasets dynamically across multiple ranges or tabs.

Understanding this workaround is essential for advanced data management in Google Sheets. By mastering the combination of these specific functions, practitioners can achieve flexible and dynamic data integrations, mimicking the results of a relational join operation, such as the crucial left join. This sophisticated approach ensures that data integrity is maintained while compiling unified reports or analytical views from segmented source data.

The Necessity of Workarounds in Google Sheets Query

Often, data analysts require the functionality of SQL's `JOIN` clause to integrate information from multiple sources, such as linking employee IDs in one sheet to salary data in another. This requirement is often addressed by trying to embed the join logic directly within the powerful QUERY() function. Unfortunately, Google Sheets does not support this native integration syntax. The internal query language is designed primarily for selecting, filtering, grouping, and pivoting data from a single input range, not for cross-referencing multiple ranges simultaneously within the query statement itself.

The absence of a direct `JOIN()` function within the context of the QUERY() function means that users must pivot to traditional spreadsheet functions designed for data lookup and array manipulation. The most effective technique involves combining two key capabilities: using ArrayFormula to process multiple rows simultaneously and using the VLOOKUP function to search for matching records across the secondary dataset. This approach effectively simulates a lookup-based left join, where every row in the primary table is checked against the corresponding key in the secondary table, and matching values are returned.

For those aiming to merge datasets dynamically, the following comprehensive formula structure provides the necessary logic. It is an array-based solution that bypasses the limitations of the Sheets QUERY syntax by performing the data merging operation outside of the query itself, thereby enabling the integration of two distinct tables into one cohesive array output:

```
=ArrayFormula(
{
A2:B6,
vlookup(A2:A6,D2:E6,COLUMN(Indirect("R1C2:R1C"&COLUMNS(D2:E6),0)),0)
}
)
```

This particular formula configuration performs a highly effective simulation of a left join operation. It seamlessly combines the primary data range, specified here as **A2:B6** (the "left" table), with the relevant supplemental data retrieved from the secondary lookup range, **D2:E6** (the "right" table). The power of this methodology lies in its ability to handle multi-column lookups and return multiple columns of data simultaneously, a significant advancement over simple cell-by-cell VLOOKUP applications.

Prerequisites for Successful Data Joining

Before implementing the advanced array formula, it is critical to ensure that your source data tables are structured correctly. Effective data joining requires a common identifier, or primary key, present in both datasets. This key acts as the link that allows the system to match corresponding rows across the two tables. If the datasets lack a unique, shared identifier column, the lookup will fail or produce inaccurate results.

Specifically, the left table (the primary dataset) should contain all the records you wish to preserve in the final output. The key column in the left table (e.g., column A in the example) will be used as the search key within the VLOOKUP function. Conversely, the right table (the secondary dataset) must be organized such that its matching key column is the very first column in its range (e.g., column D in the example, if the range is D2:E6). This is a strict requirement of the standard VLOOKUP function.

Furthermore, careful attention must be paid to the specific ranges used in the formula. The ranges should be appropriately sized to encompass all headers and data rows intended for the join. Although the example uses fixed ranges like **A2:B6** and **D2:E6**, in production environments, it is often beneficial to use open ranges (e.g., A2:B) to ensure the formula automatically adjusts to new data additions. However, when using open ranges, conditional formatting or wrapping with functions like `FILTER` or `IFERROR` may be necessary to handle blank rows gracefully.

Step-by-Step Breakdown of the Array Join Formula

To fully utilize this powerful array formula, it is essential to dissect its components and understand how they interact to achieve the desired join operation. This formula is highly optimized for

returning multiple lookup columns, which is necessary for a complete table join.

The entire structure is enclosed by the ArrayFormula() function, which instructs Google Sheets to apply the enclosed operation--in this case, the combination of two arrays within curly braces `{}`--across a range of cells, rather than just a single cell. The curly braces themselves are used to concatenate arrays horizontally, effectively placing the results of the VLOOKUP directly alongside the source data.

Primary Array (Left Table): `A2:B6` is the first component inside the curly braces. This range is the complete left table (Team Name and Points in our example) that forms the basis of the resulting joined table. This part ensures that all primary records are retained, characteristic of a left join.

VLOOKUP Function: The heart of the linking mechanism is `vlookup(A2:A6, D2:E6, ..., 0)`. The lookup range is **A2:A6**, representing the common key (Team Name) applied across all rows simultaneously due to the ArrayFormula wrap. The lookup table is **D2:E6** (the right table).

Dynamic Column Index Generation: The most complex part is the index argument: `COLUMN(Indirect("R1C2:R1C"&COLUMNS(D2:E6),0))`. This intricate section dynamically generates the necessary column indices for the VLOOKUP to return multiple columns.

`COLUMNS(D2:E6)` calculates the total number of columns in the lookup range (in this case, 2).
This value is used to construct a range reference using the Indirect function, specifically defining a row of column numbers starting from the second column (since the first column is the lookup key). The resulting array of column numbers (e.g., {2}) tells the VLOOKUP which columns to extract from the right table. If the right table had three columns (D, E, F), this would dynamically generate the indices {2, 3}.

Exact Match Requirement: The final argument, `0` (or `FALSE`), ensures that the VLOOKUP performs an exact match, which is critical for accurate table joining.

Practical Example: Joining Basketball Team Data

To illustrate the efficiency of this method, consider a common scenario involving two separate tables containing data about basketball teams. The first table (Table 1) tracks points scored, and the second table (Table 2) tracks assists, with both tables sharing the unique team name as the key identifier.

Suppose we have the following two tables in Google Sheets that contain information about various basketball teams:

	A	B	C	D	E
1	**Team**	**Points**		**Team**	**Assists**
2	Mavs	99		Spurs	22
3	Spurs	94		Kings	26
4	Hawks	105		Rockets	25
5	Kings	100		Mavs	31
6	Rockets	92		Hawks	34
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					
19					

Table 1 occupies the range **A1:B6**, listing Team Name and Points. Table 2 occupies **D1:E6**, listing Team Name and Assists. Note that the lookup key (Team Name) is the first column in both tables, satisfying the VLOOKUP requirement for the secondary table.

Our goal is to execute a left join, ensuring that all teams listed in Table 1 appear in the final output, supplemented by their corresponding assist counts from Table 2. We can achieve this by placing the complete array formula in a new cell, typically C2 or G2, outside the source data ranges. This consolidation ensures that we generate one comprehensive table containing the team name, points, and assists for every team initially present in the left table.

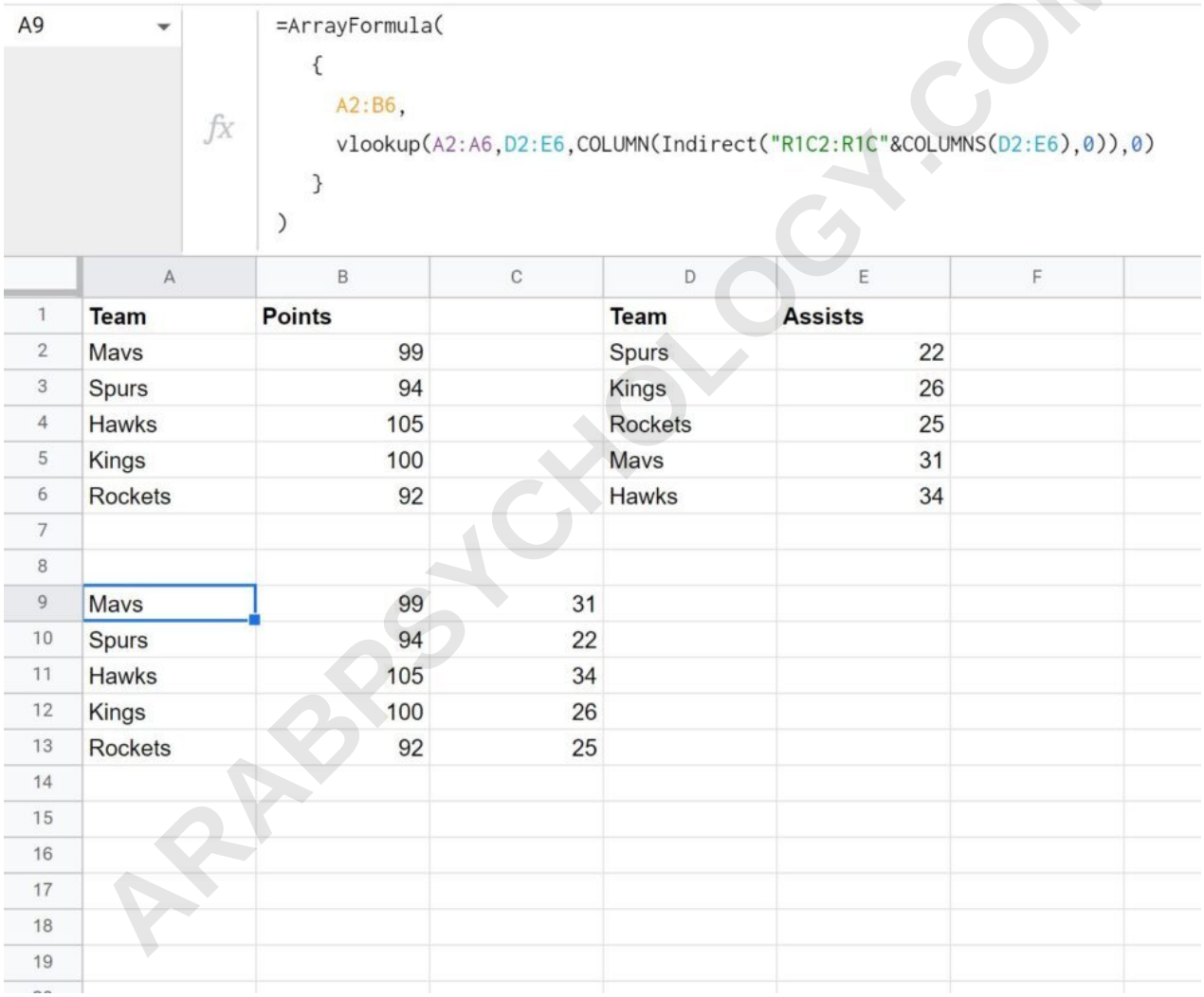
We use the following formula, structured precisely to handle the required ranges:

```
=ArrayFormula(
{
A2:B6,
vlookup(A2:A6,D2:E6,COLUMN(Indirect("R1C2:R1C"&COLUMNS(D2:E6),0)),0)
}
)
```

Upon execution, the ArrayFormula calculates the required lookup value for each row in A2:A6 and returns the corresponding data from the second column of the D2:E6 range (which holds the Assists data) alongside the original data from A2:B6.

Interpreting the Results and Handling #N/A Values

The following screenshot demonstrates the successful application of the array join formula, resulting in a single, unified dataset:



	A	B	C	D	E	F
1	**Team**	**Points**		**Team**	**Assists**	
2	Mavs	99		Spurs	22	
3	Spurs	94		Kings	26	
4	Hawks	105		Rockets	25	
5	Kings	100		Mavs	31	
6	Rockets	92		Hawks	34	
7						
8						
9	Mavs	99	31			
10	Spurs	94	22			
11	Hawks	105	34			
12	Kings	100	26			
13	Rockets	92	25			
14						
15						
16						
17						
18						
19						

Crucially, notice that the final result is one single table efficiently merged through the power of the array structure. The output retains the structure of the left table (Team Name and Points) and appends the relevant column (Assists) from the right table. This consolidation streamlines analysis and reporting processes by placing all necessary metrics side-by-side.

A key characteristic of any left join simulation using VLOOKUP is how it handles unmatched

records. If a specific team listed in the left table (e.g., "Falcons") does not have a corresponding entry in the right table (the assists data range), the VLOOKUP function cannot find a match. When an exact match is requested but not found, the function returns the standard Google Sheets error value: **#N/A**.

Note: If a team in the left table does not exist in the right table, a value of **#N/A** will be returned in the Assists column of the resulting table. While this error explicitly indicates missing data, it often needs to be cleaned up for presentation or further calculation. To handle the **#N/A** error gracefully, the entire VLOOKUP statement can be wrapped in an `IFERROR()` function. For example, wrapping the VLOOKUP part in `IFERROR(VLOOKUP(...), "No Match")` allows you to replace the **#N/A** value with a zero, a blank cell, or a descriptive text string, significantly improving the readability and usability of the final joined table.

Generalizing the Array Join Methodology

The methodology described using ArrayFormula and VLOOKUP is highly flexible and can be adapted beyond the simple two-column join demonstrated. For instance, if the right table (D2:Z) contained twenty columns of data, the dynamic column index generator using the Indirect function and `COLUMNS` would automatically adjust to return all twenty columns in the joined output, provided they all follow the initial lookup key.

This approach primarily simulates an **Outer Join** (specifically, a Left Outer Join) because it retains all rows from the primary range (A2:B6) regardless of a match in the secondary range. If an analyst required an **Inner Join** (where only records present in both tables are returned), the solution would need to be filtered. This filtering can be achieved by wrapping the entire ArrayFormula structure within a `FILTER()` function, filtering out any rows where the VLOOKUP result column contains the **#N/A** error.

Alternatively, if the goal is to perform a join based on multiple criteria--a composite key--the VLOOKUP structure becomes unsuitable, necessitating a switch to more advanced functions like `INDEX` combined with `MATCH`, often utilizing an intermediary helper column to concatenate the multiple key fields into a single unique string for lookup purposes. However, for standard single-key lookups, the demonstrated ArrayFormula and VLOOKUP method remains the most streamlined and efficient workaround for simulating SQL join operations within Google Sheets.

Further Resources for Advanced Google Sheets Operations

The process of manipulating and integrating data in Google Sheets extends far beyond simple joining. Advanced users frequently utilize the capabilities of the QUERY function for tasks such as data aggregation and summarization. Understanding how to structure complex query clauses is essential for deriving meaningful insights from large datasets.

The following tutorials explain how to perform other common, yet complex, tasks in Google Sheets, often complementing the array joining techniques discussed above:

Google Sheets Query: How to Use Group By