Learn How to Extract Multiple Matching Values in Google Sheets

Authored by

Mohammed looti

November 12, 2025

The Lookup Limitation: Why Standard Functions Fall Short

Standard lookup utilities available in Google Sheets, such as **VLOOKUP** or **XLOOKUP**, are fundamentally designed to handle single-match scenarios. Their primary directive is to scan a dataset based on a specific criterion and return the very first corresponding value they encounter. While this is highly efficient for unique identifiers, this inherent limitation renders them inadequate when data extraction requires retrieving an entire collection of related values from a single lookup. This challenge frequently arises when analysts need to filter large tables to isolate all instances of a particular category--for example, listing every date a specific financial transaction occurred or, as we will demonstrate, compiling a complete list of championship years associated with a single team name.

To successfully navigate this technical hurdle, users must implement a sophisticated construction known as an Array formula. This solution moves beyond simple direct lookups by dynamically creating a temporary array of data points. Instead of relying on a single cell reference, this technique leverages the combined functionality of several core functions to identify every row where the criterion is met, collect those row indices, and then sequentially pull the required data points. This methodology is crucial for establishing robust, non-destructive filtering mechanisms that operate directly within the spreadsheet, thereby eliminating the need for cumbersome helper columns or complex scripting languages.

The advanced method detailed in this guide offers a highly reliable framework for data extraction, ensuring that every matching entry is successfully located and displayed in a clean, sequential list. Achieving mastery in this area requires a deep understanding of the intricate interplay between the individual components--specifically, how they cooperate to transform a conditional test into a list of specific data coordinates. This powerful formula architecture is indispensable for building highly flexible, automated, and dynamic reports within the Google Sheets environment.

Introduction to the Array Formula Solution

The core of our strategy for extracting multiple corresponding values in Google Sheets rests on a single, powerful array formula. This formula is designed to be entered once in the top cell of the designated results column (for instance, cell **E2** in our scenario) and then copied down to automatically populate all subsequent matching entries. It avoids the typical single-cell computation model by processing the entire data range simultaneously.

The formula configuration below represents the most efficient way to execute this complex, conditional lookup:

=INDEX($A$1:$A$14, SMALL(IF(D$2=$B$1:$B$14, MATCH(ROW($B$1:$B$14), ROW($B$1:$B$14)), ""), ROWS($A$1:A1)))

This sophisticated expression is engineered to retrieve all values contained within the specified return range, **$A$1:$A$14**, only when the corresponding entry in the criterion range, **$B$1:$B$14**, precisely matches the lookup value provided in cell **D2**. The formula's brilliance lies not in finding the first match, but in its ability to iterate through the entire criteria range, collecting the row coordinates for every match found, regardless of its position in the dataset. This iterative collection of row numbers is what distinguishes it from standard, single-value lookup functions.

Understanding the flow of control within this nested structure is paramount. The conditional test is executed by the IF function. Upon determining a match, the formula uses the combined power of the MATCH function and the ROW function to capture the exact row index of the successful match. These row indices are aggregated and then fed into the SMALL function. The SMALL function systematically extracts the 1st, 2nd, 3rd, and subsequent row numbers as the formula is dragged down the column. Finally, the INDEX function utilizes the extracted row index to retrieve and display the actual desired data value from Column A.

Deconstructing the Formula's Logic: INDEX, SMALL, and IF

To fully grasp the mechanism of this array formula, we must analyze the specific role each function plays within the nested structure. The **outermost function** is the INDEX function. Its responsibility is to return a value from a designated range (the return data range, `$A$1:$A$14`) based solely on a provided row number, which is dynamically calculated by the inner functions. The use of **absolute references** (indicated by the dollar signs, e.g., `$A$1:$A$14`) is absolutely critical here, as it prevents the source data ranges from shifting when the formula is copied vertically across multiple cells.

The core logic resides within the combined operation of the SMALL function and the IF function. The IF function executes the necessary array comparison: `D$2=$B$1:$B$14`. This comparison evaluates every cell in the criteria range against the single cell criterion, producing a Boolean array of TRUE/FALSE values. Where the condition is TRUE (a match occurs), the formula utilizes a specialized construction--`MATCH(ROW($B$1:$B$14), ROW($B$1:$B$14))`--to return the exact row index number of that match within the array. All non-matching rows are represented by blank strings (""). This results in an array containing only the numerical index positions of the matched data points, interspersed with empty values.

This resulting array of row numbers and blank strings is subsequently passed to the SMALL function. The role of SMALL is to extract the k-th smallest valid numerical value from this collection. The key to the sequential extraction is the final element of the formula: `ROWS($A$1:A1)`. When the formula is initially placed in cell **E2**, the ROWS function calculates the number of rows in the reference range `$A$1:A1`, which returns 1. This instructs the SMALL function to retrieve the 1st smallest matched row index. As the formula is dragged down to cell **E3**, the relative reference

updates to `$A$1:A2`, causing ROWS to return 2. This ingenious use of a dynamically expanding reference creates an automatic counter, enabling the formula to sequentially retrieve the 2nd, 3rd, and subsequent matched row indices generated by the IF function.

Practical Example: Setting Up the Championship Dataset

To demonstrate the effectiveness of this advanced extraction technique, we will apply it to a common real-world dataset scenario. Imagine a table within Google Sheets that logs the winners of the NBA Finals over several seasons. In this dataset, Column A holds the year (which represents the values we wish to return), and Column B contains the winning team name (which serves as our criteria range for the lookup).

Our sample dataset spans the range A1:B14, illustrating a clear separation between the data identifier (Year) and the criterion data (Team). This fundamental structure is essential for the formula's successful execution:

	A	B	C	D
1	**Year**	**Winner**		
2	2010	Lakers		
3	2011	Mavs		
4	2012	Heat		
5	2013	Heat		
6	2014	Spurs		
7	2015	Warriors		
8	2016	Cavs		
9	2017	Warriors		
10	2018	Warriors		
11	2019	Raptors		
12	2020	Lakers		
13	2021	Bucks		
14	2022	Warriors		
15				
16				
17				

The objective is to establish a dynamic filtering mechanism. We need to specify a team name in a designated cell, and have a separate results column instantly list all the years that team secured the championship title. For this initial demonstration, we will search for the years the "Warriors" won, which requires setting up our criterion cell.

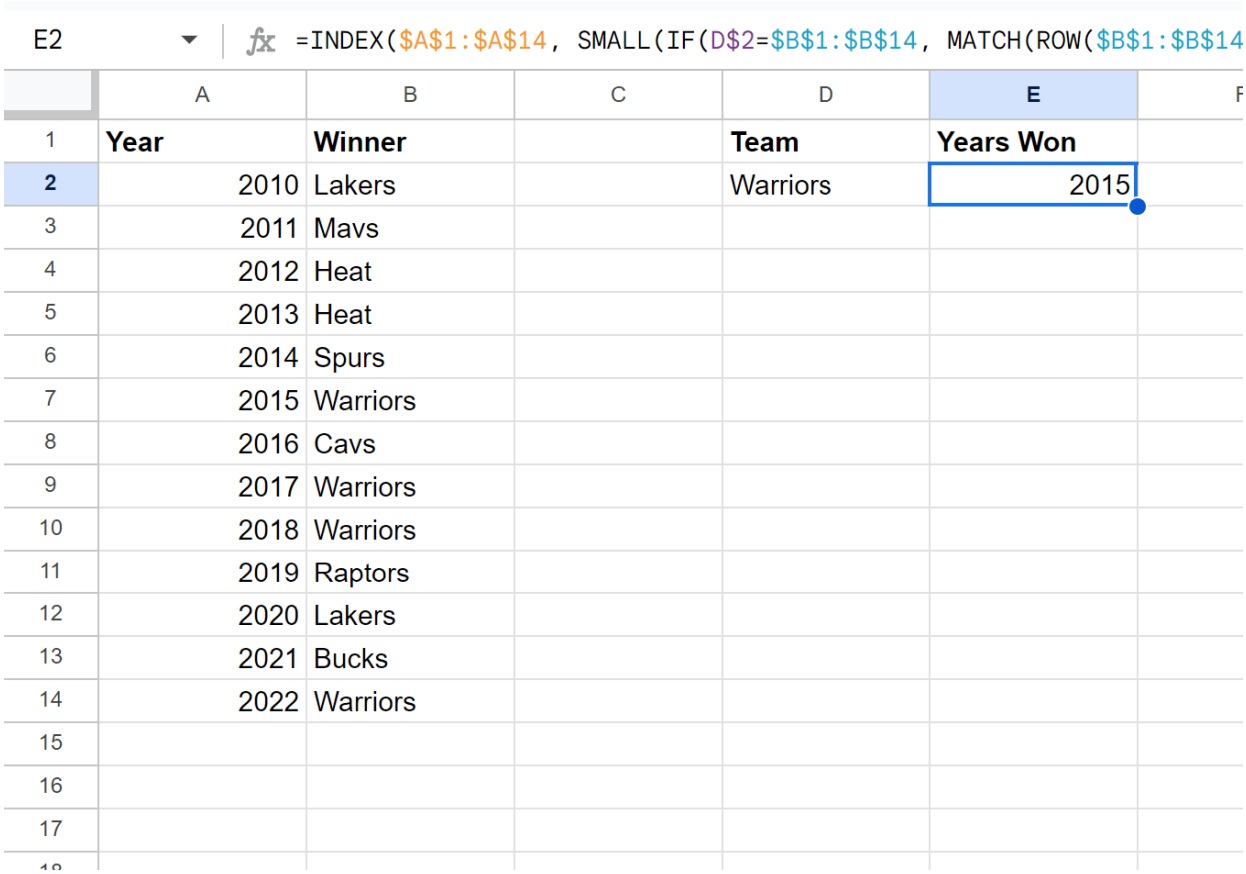
To initiate the lookup process, we first define the single criterion. Type the specific team name you wish to search for--in this case, **"Warriors"**--into cell **D2**. This cell now functions as the dynamic input for the formula's conditional test. Subsequently, we will insert the complex array formula into the designated results column, starting at cell **E2**.

Executing the Formula and Interpreting the Results

With the search criterion correctly established in cell **D2**, input the complete array formula into cell **E2**. It is imperative that this formula is entered precisely as written, paying close attention to the absolute references that anchor the source ranges:

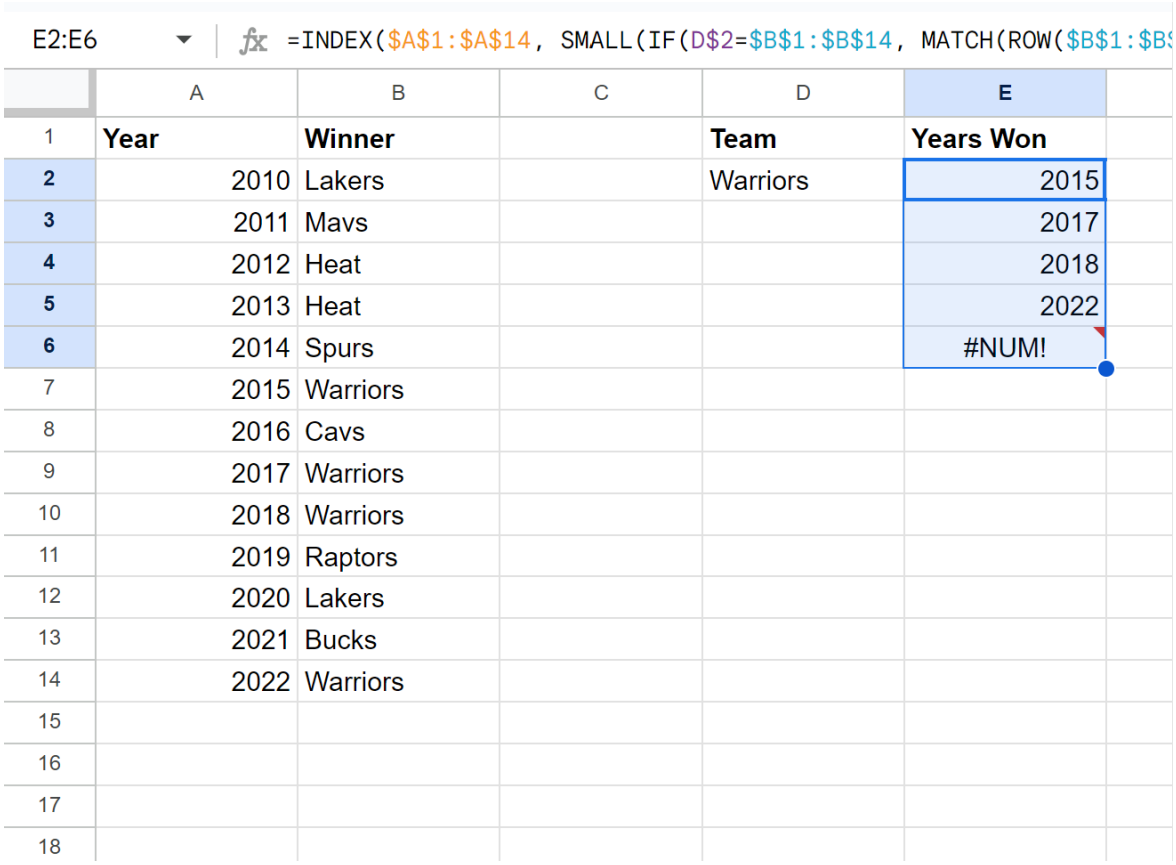
=INDEX($A$1:$A$14, SMALL(IF(D$2=$B$1:$B$14, MATCH(ROW($B$1:$B$14), ROW($B$1:$B$14)), ""), ROWS($A$1:A1)))

Once **Enter** is pressed, the formula immediately executes the array comparison. Since the counter `ROWS($A$1:A1)` evaluates to 1, the SMALL function retrieves the smallest (first) row number corresponding to a "Warriors" championship win. Consequently, cell **E2** instantly displays the first year the Warriors secured the title.



	A	B	C	D	E
1	**Year**	**Winner**		**Team**	**Years Won**
2	2010	Lakers		Warriors	2015
3	2011	Mavs			
4	2012	Heat			
5	2013	Heat			
6	2014	Spurs			
7	2015	Warriors			
8	2016	Cavs			
9	2017	Warriors			
10	2018	Warriors			
11	2019	Raptors			
12	2020	Lakers			
13	2021	Bucks			
14	2022	Warriors			
15					
16					
17					

To extract the remaining matches, we leverage the fill handle feature. Drag the formula down column E to cover the anticipated number of matches. As the formula propagates downward, the relative reference within the ROWS function increments sequentially (2, 3, 4, etc.). This automatically instructs the SMALL function to retrieve the second, third, and fourth smallest matched row numbers, respectively. This iterative process continues until all possible matches have been successfully extracted from the dataset.



Once the formula has successfully returned all matching values, the internal counter will attempt to locate a non-existent subsequent match (e.g., a fifth or sixth entry). When no further valid numerical index is found in the array of row numbers, the SMALL function fails, resulting in the display of the **#NUM!** error value. This error serves as a clear indicator that the search is complete and provides a definitive end boundary for the extracted list. Based on our sample dataset, the Warriors' winning years are:

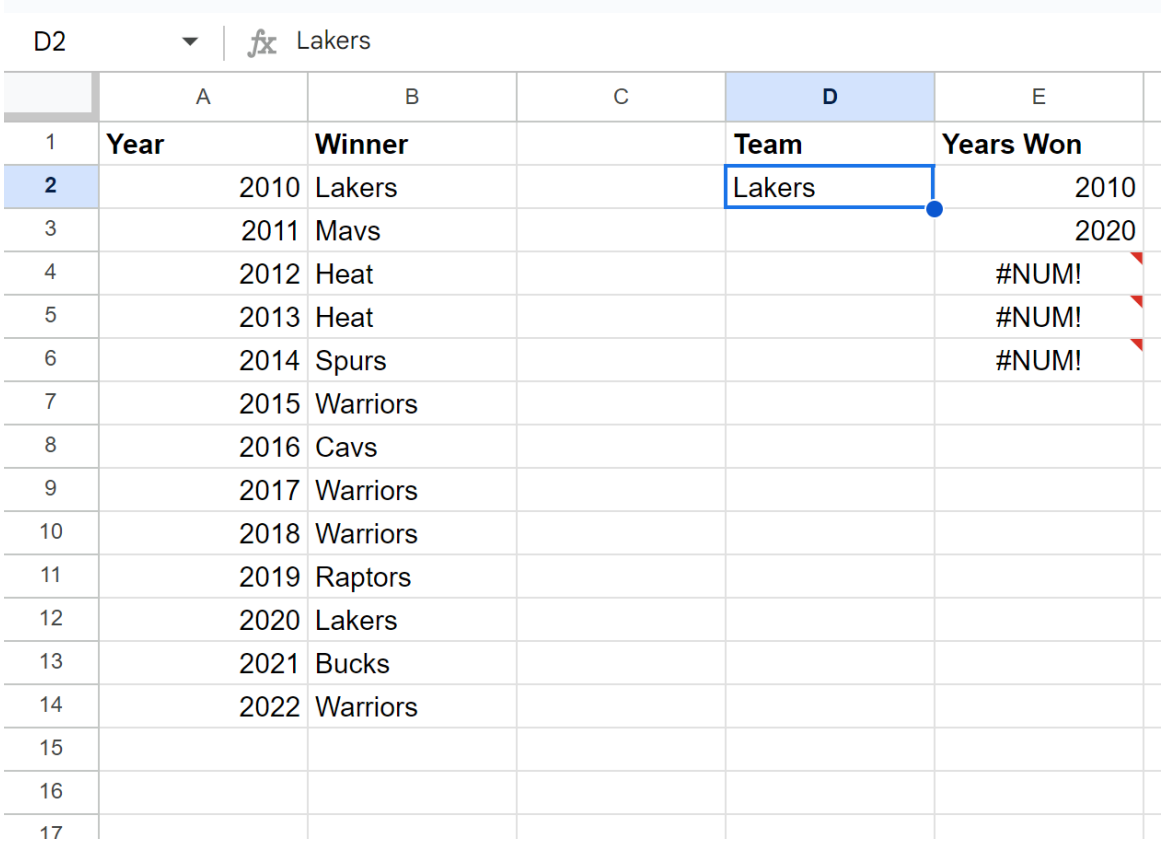
2015
2017
2018
2022

Enhancing Flexibility with Dynamic Criteria Selection

A significant advantage of employing this array-based approach is the high degree of dynamism it introduces to data reporting. Since the criterion cell (**D2**) is referenced using a semi-absolute reference (`D$2`), the entire list of results in column E automatically and instantly recalculates whenever the input team name in **D2** is changed. This transforms the results column into a powerful, dynamic filter that provides immediate feedback based on user input, without requiring any manual modification of the complex formula itself.

For instance, if we modify the team name in cell **D2** from "Warriors" to "Lakers," the list of years displayed in column E instantly updates. The formula discards the previous results and executes a new lookup, displaying only the years corresponding to the Lakers' championship victories. This exceptional flexibility makes this formula ideal for creating interactive dashboards and reporting tools where users must rapidly switch between different criteria views for analysis.

Observe the instantaneous shift in the output when the criterion is updated:



	A	B	C	D	E
1	**Year**	**Winner**		**Team**	**Years Won**
2	2010	Lakers		Lakers	2010
3	2011	Mavs			2020
4	2012	Heat			#NUM!
5	2013	Heat			#NUM!
6	2014	Spurs			#NUM!
7	2015	Warriors			
8	2016	Cavs			
9	2017	Warriors			
10	2018	Warriors			
11	2019	Raptors			
12	2020	Lakers			
13	2021	Bucks			
14	2022	Warriors			
15					
16					
17					

Based on the newly filtered data, we can confirm the Lakers' championship years within this dataset are:

2010
2020

We strongly encourage users to experiment further by entering any team name present in the source column (B1:B14) into cell **D2**. The resultant list in column E will automatically adjust, powerfully demonstrating the utility and efficiency of a well-constructed array filtering formula in a modern spreadsheet environment.

Further Exploration and Related Resources

Mastering the conditional combination of the INDEX function and SMALL function, supported by the logical testing capabilities of the IF function, is a fundamental skill set for advanced data manipulation in spreadsheet software. This technique enables complex data extraction scenarios that are simply unattainable using traditional, single-value lookup functions.

For users looking to refine the user experience of these filtered lists, the next logical step often involves exploring methods to suppress the visually distracting **#NUM!** error. The IFERROR function is typically employed for this purpose, resulting in a cleaner output. Furthermore, expanding this array logic to handle lookups based on **multiple criteria** simultaneously represents the next level of proficiency, unlocking even greater possibilities for deep data analysis and advanced reporting structures.

The following authoritative resources offer detailed explanations of the component functions utilized in this powerful technique, providing essential documentation for a deeper understanding of their individual capabilities:

Documentation on the INDEX function
Documentation on the SMALL function
Documentation on the IF function
Documentation on the MATCH function