How to Retrieve Multiple Results with INDEX MATCH in Google Sheets

Authored by

mohammed looti

January 8, 2026

The combination of the INDEX function and the MATCH function is widely recognized as an extremely potent lookup method within spreadsheet environments, notably in Google Sheets. While the standard implementation of INDEX MATCH is designed primarily to return a single, corresponding value based on a single criterion--acting as a more flexible and less restrictive alternative to VLOOKUP--advanced data analysis often requires the retrieval of multiple entries associated with the same lookup key. For instance, if you are tracking sales data and need to pull every transaction linked to a specific customer ID, the conventional approach falls short. This limitation necessitates the integration of additional functions to transform the standard lookup into a robust, multi-result retrieval system, enabling users to efficiently handle complex datasets where criteria matching is not unique.

The core challenge lies in instructing the formula to continue searching the data range after the first match has been found. Standard lookup functions halt processing immediately upon identifying the initial relevant entry. To overcome this, we utilize array processing capabilities inherent to Google Sheets, combining functions like INDEX function, SMALL function, and ROW function into a single, comprehensive array formula. This sophisticated construct calculates the exact row indices for every matching record and then sequentially retrieves the associated values. Mastering this technique significantly elevates one's data management proficiency, providing an invaluable tool for complex data extraction and organizational tasks within large relational datasets.

Understanding the Need for Advanced Lookups

While simpler functions like MATCH function or VLOOKUP are essential for basic data retrieval, they are inherently limited when dealing with one-to-many relationships. When a database contains multiple records that satisfy a single search criterion, we need a mechanism that iterates through the entire dataset and collects the location of every valid entry. This complexity demands a shift from standard cell processing to an array formula structure, which allows calculations to be performed on entire ranges simultaneously rather than just individual cells. This capability is paramount for generating clean, dynamic reports where filtered subsets of data must be displayed adjacent to the main dataset without manually sorting or filtering the source range.

The standard methodology for achieving this multiple-result lookup in Google Sheets combines several key utility functions. The resulting formula structure, which we will analyze in depth, is specifically designed to overcome the single-result limitation by performing a conditional check across a range, calculating the relative position of every true match, and then using that position dynamically to pull the corresponding result. Understanding this specific syntax is crucial for anyone engaging in high-level data manipulation, as it offers granular control over the lookup process that simpler dedicated functions often cannot provide, especially when dealing with older spreadsheet versions or environments lacking modern functions like FILTER or QUERY.

This powerful methodology allows content creators, analysts, and data managers to bypass manual data restructuring and instead rely on an automated system to extract specific subsets of information. By treating the lookup range as an array, the formula effectively filters the data in memory before retrieving the results, ensuring that the process is both efficient and reliable across varying dataset sizes. This technique forms a crucial pillar of advanced spreadsheet modeling, enabling complex reporting requirements to be met directly within the sheet environment.

Deconstructing the Multiple Result INDEX MATCH Formula

To successfully retrieve all corresponding values based on a single criteria match, we must employ a highly specific, nested array formula structure. This complex formula requires careful construction, utilizing functions that identify matches and subsequently order the results for sequential retrieval. The general syntax provided below achieves this by generating an array of row numbers corresponding to the matches, ordering those numbers, and then feeding them one by one into the INDEX function as the formula is dragged down across multiple output cells.

The foundational syntax required to execute a multi-result lookup using the INDEX MATCH paradigm (enhanced with array processing) in Google Sheets is as follows. Note that this structure must be entered as a standard formula but operates internally as an array based on the functions contained within:

=IFERROR(INDEX($B$2:$B$11,SMALL(IF($D$2=$A$2:$A$11,ROW($A$2:$A$11)-ROW($A$2)+1),ROW(1:1))),"")

In this powerful expression, the formula is configured to examine the lookup range, defined as the column spanning **A2:A11**, and compare every cell in this range against the specific criteria provided in cell **D2**. Upon identifying matches, the formula calculates and stores the relative row position of these matches. The final output is drawn from the return range, **B2:B11**, ensuring that all values corresponding to the criteria in **D2** are sequentially displayed. The inclusion of the IFERROR function acts as a vital safety net, preventing unsightly error messages once all available matches have been retrieved, thereby cleaning up the final display when the formula is extended beyond the necessary number of output rows.

This method leverages conditional logic within an array formula context, allowing the formula to generate an intermediate data structure--an array of row numbers--that dictates the order of retrieval. By separating the matching process (using IF) from the retrieval process (using INDEX and SMALL), we achieve a dynamic filtering mechanism that automatically adjusts to the volume of matching records in the source data.

A Deep Dive into Formula Components

To truly master this technique, it is necessary to understand the role of each nested function, as they collaborate to perform the filtering and ordering process. The formula relies on three primary internal mechanisms that work together: conditional matching, position calculation, and sequential ranking.

Firstly, the inner core, `IF($D$2=$A$2:$A$11, ROW($A$2:$A$11)-ROW($A$2)+1)`, is responsible for the conditional matching. This section creates an array. Where the condition (Value in **D2** equals cell in range **A2:A11**) is **TRUE**, it returns the relative row number of the match within the range **A2:A11**. Where the condition is **FALSE**, it returns the value **FALSE**. The subtraction logic (`ROW($A$2:$A$11)-ROW($A$2)+1`) is essential; it converts the absolute row number (e.g., 2, 3, 4...) into a relative row position within the array (e.g., 1, 2, 3...). For instance, if the first match is in row 2, the calculation is 2 - 2 + 1 = 1 (the first relative position). This ensures the row index is always correct relative to the start of the return array **B2:B11**, regardless of where the data set begins.

Secondly, the SMALL function then acts upon this resultant array of row positions and **FALSE** values. The SMALL function is tasked with pulling the K-th smallest value from a set. The non-numeric **FALSE** values are ignored by the SMALL function, meaning it focuses exclusively on the valid relative row numbers generated by the IF statement. Crucially, the argument for K (which match to retrieve) is dynamically supplied by `ROW(1:1)`. When the formula is first entered, `ROW(1:1)` returns 1 (the 1st smallest relative row number). When the formula is dragged down to the next cell, `ROW(1:1)` automatically increments to `ROW(2:2)`, returning 2 (the 2nd smallest relative row number), and so on. This ingenious use of a relative ROW function is what allows the single formula structure to retrieve multiple results sequentially.

Finally, the outer INDEX function utilizes the relative row number calculated by the nested SMALL function. The INDEX function syntax is `INDEX(range, row_number)`. The range is the output column (**$B$2:$B$11**), and the row number is the result of the SMALL function calculation. If SMALL function returns the relative position 3, INDEX function retrieves the value in the third row of the range **B2:B11**. The entire expression is wrapped in IFERROR function so that when the SMALL function attempts to find a 4th or 5th smallest value but only 3 matches exist, it generates an error, which IFERROR function intercepts and converts into a clean, empty string (`""`).

Setting Up the Dataset (Practical Example)

To demonstrate the effectiveness and efficiency of this specialized formula, we will work with a simple, yet representative, dataset consisting of basketball team affiliations and corresponding player names. This scenario perfectly illustrates a one-to-many relationship: multiple players belong to a single team, and we need to extract all associated player names when querying for a

specific team name.

Assume the following structure is established in your Google Sheet, starting at cell **A1**:

Column A: Team Name (The Lookup Range)
Column B: Player Name (The Return Range)

The dataset includes ten records, showing the affiliations of various players. The goal is to dynamically extract all player names based on a team specified in a separate criteria cell. Below is a representation of the data:

	A	B	C
1	**Team**	**Player**	
2	Mavs	Dan	
3	Mavs	Mike	
4	Warriors	Steven	
5	Heat	Carl	
6	Heat	Jake	
7	Mavs	Brad	
8	Warriors	Tyler	
9	Kings	AJ	
10	Mavs	Spencer	
11	Heat	Dawkins	
12			
13			
14			
15			

In this setup, we can observe that the "Mavs" team appears multiple times, confirming the need for a multi-result lookup approach. Our objective is specifically to retrieve the names of all players associated with the "Mavs" team using a single, draggable formula, displaying the results in a dedicated output column. We will designate cell **D2** as our search criterion input and column E as our output column for the retrieved player names.

Implementing the Formula Step-by-Step

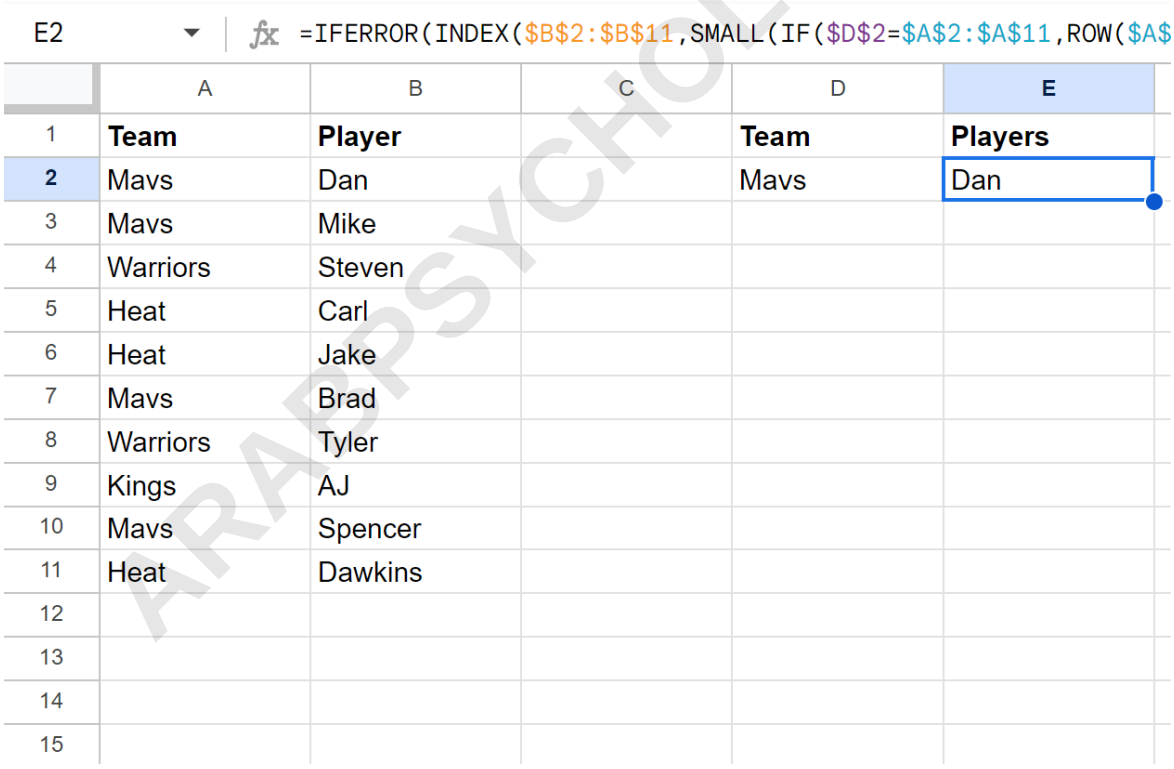
Our practical application involves identifying all players belonging to the "Mavs" team. We designate a cell, **D2**, as the input criteria cell, where we will type the lookup value, "Mavs". The results will then be populated starting in cell **E2**, extending downwards.

First, input the criteria: Type **Mavs** into cell **D2**. This cell now serves as the anchor for the conditional filtering within the array formula.

Next, enter the complete array formula into cell **E2**. Based on our dataset ranges (A2:A11 for the lookup criteria and B2:B11 for the return values), the formula remains exactly as analyzed previously. Ensure that all absolute and relative references are correctly positioned before pressing Enter:

=IFERROR(INDEX($B$2:$B$11,SMALL(IF($D$2=$A$2:$A$11,ROW($A$2:$A$11)-ROW($A$2)+1),ROW(1:1))),"")

Upon pressing **Enter** in cell **E2**, the formula immediately executes the array calculations. It identifies the smallest relative row position corresponding to "Mavs" in column A (which is row 1 relative to the range A2:A11), and the INDEX function retrieves the player name from that position in column B. The result in cell **E2** will be the name of the first player on the "Mavs" team, confirming the successful retrieval of the initial match:



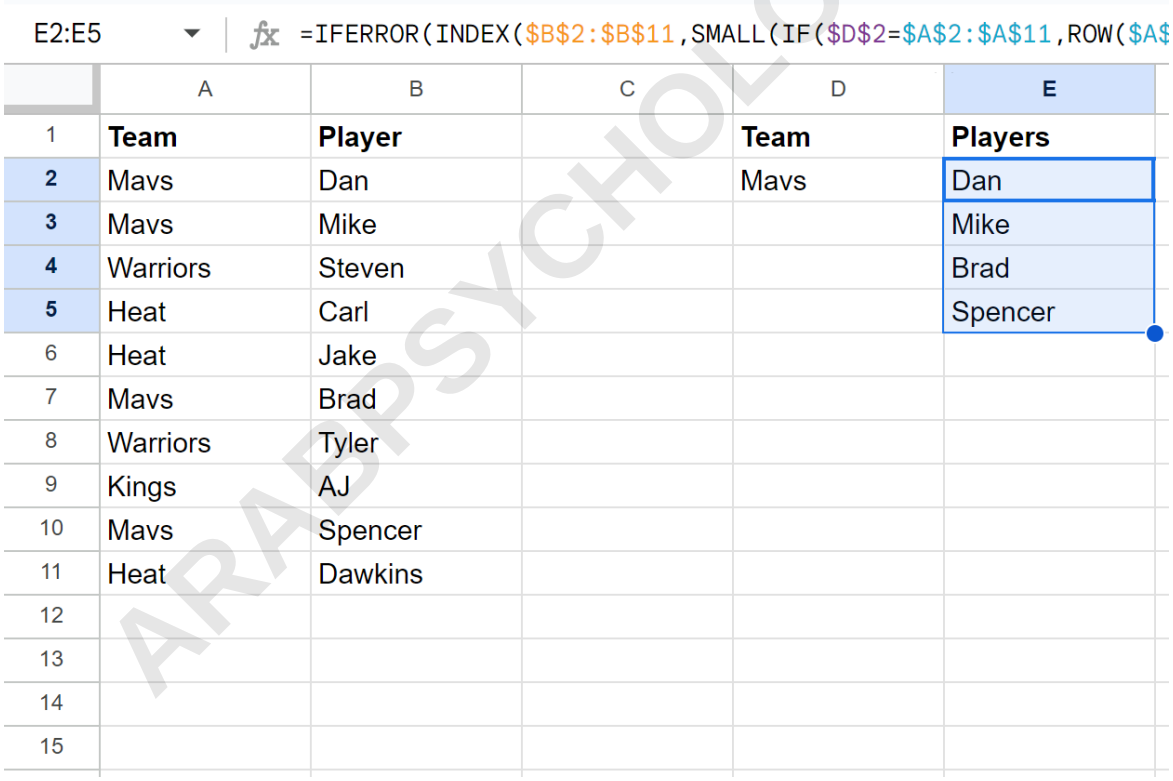
Analyzing the Results and Dynamic Behavior

The true power of this array formula is unlocked when it is copied down through subsequent cells. Since the `ROW(1:1)` component uses a relative reference while all other range references are

absolute (using dollar signs, **$**), the formula changes dynamically as it is dragged down the output column E. This dynamic adjustment is what facilitates sequential, multi-result extraction.

When the formula is dragged from **E2** to **E3**, the core logic remains constant, but the retrieval index shifts. The relative row reference inside the SMALL function automatically updates from `ROW(1:1)` to `ROW(2:2)`. This instructs the SMALL function to find the second smallest relative row number identified in the array. This corresponds to the second player on the "Mavs" team. By continuing this dragging and filling action down column E, we sequentially retrieve all matching records until the list is exhausted.

The subsequent illustration confirms the successful retrieval of each of the four players associated with the "Mavs" team. Once the formula is extended beyond the fourth match, the SMALL function fails to find a fifth relative row number, as only four were generated by the IF condition. This triggers an internal error, which is gracefully handled by the outer IFERROR function, resulting in clean, empty cells below the last valid result:



A significant advantage of this formula is its inherent dynamic capability. If the user decides to change the criteria in cell **D2**--for example, updating "Mavs" to "Rockets"--the entire output column E recalculates automatically. This instantaneous update reflects the new set of matching players without any need to modify or re-enter the formula, demonstrating the efficiency and robust automation this complex lookup methodology provides for reporting and interactive data

dashboards:

	A	B	C	D	E
1	**Team**	**Player**		**Team**	**Players**
2	Mavs	Dan		Heat	Carl
3	Mavs	Mike			Jake
4	Warriors	Steven			Dawkins
5	Heat	Carl			
6	Heat	Jake			
7	Mavs	Brad			
8	Warriors	Tyler			
9	Kings	AJ			
10	Mavs	Spencer			
11	Heat	Dawkins			
12					
13					
14					
15					

Key Considerations: Absolute vs. Relative References

The successful deployment of this multi-result lookup formula hinges entirely on the meticulous use of absolute and relative cell references. Failure to apply these references correctly will break the formula when it is copied down the column, leading to erroneous or inconsistent results. This strict adherence to referencing rules is mandatory because the formula must simultaneously reference fixed source data while dynamically altering its retrieval index.

Specific components that must utilize absolute references (prefixed with the **$** symbol, e.g., **$A$2:$A$11**) include the following ranges: the return range (where results are pulled from, **B2:B11**), the lookup criteria range (where matches are checked, **A2:A11**), and the single criteria cell (**D2**). These ranges must remain locked regardless of where the formula is copied. Similarly, the reference used to calculate the starting row position (`ROW($A$2)`) must be absolute to ensure the relative row counting always begins at 1.

In contrast, the only element that must utilize a relative reference is `ROW(1:1)`. As discussed, this specific relative range reference is the mechanism that increments the 'K' value for the SMALL function (from 1 to 2, 3, 4, and so forth) as the formula is dragged down the output column. Maintaining this distinction is crucial; absolute references anchor the source data, while the relative reference drives the sequential retrieval of multiple matching results, transforming a static lookup

into a dynamic, cascading result generator.

Alternatives to INDEX MATCH for Multiple Results

While the advanced INDEX MATCH array formula is highly effective and demonstrates a deep mastery of spreadsheet logic, modern versions of Google Sheets offer alternative functions that can accomplish the same multi-result lookup with simpler, more straightforward syntax. Two primary alternatives, **FILTER** and **QUERY**, are often preferred for their readability and ease of implementation, especially when dealing with large datasets or when simplicity is prioritized over backward compatibility.

The **FILTER** function is arguably the most direct replacement. It allows the user to specify a data range and a corresponding array of Boolean (TRUE/FALSE) conditions, returning only the rows where the condition is TRUE. The formula structure is significantly simpler: `=FILTER(B2:B11, A2:A11=D2)`. This formula is entered only once into the top output cell (e.g., E2) and automatically spills all results down the column, eliminating the need to drag the formula and manage complex row indexing logic. A simple wrapper using IFERROR function can be added if the query might return no results, ensuring a clean output.

The **QUERY** function provides even greater flexibility, allowing users to write complex data manipulation statements using the Google Visualization API Query Language, which is similar to SQL. To achieve the same result as our complex INDEX MATCH, the QUERY syntax would be: `=QUERY(A:B, "SELECT B WHERE A = '"&D2&"'", 0)`. This function is powerful, concise, and highly efficient for managing large volumes of data, enabling grouping, sorting, and conditional selection all within a single function call, making it the preferred method for advanced data warehousing tasks within Google Sheets. While the INDEX MATCH array formula remains a valuable legacy technique, especially for compatibility or specific architectural requirements, newcomers are often advised to leverage FILTER or QUERY for multi-criteria retrieval due to their superior readability and simplified deployment.

Further Exploration in Google Sheets

Mastering complex array formula structures is only one aspect of advanced data management in Google Sheets. The techniques discussed here lay the foundation for solving various other complex lookup and filtering challenges. Users interested in expanding their knowledge should explore related tutorials that cover other common data manipulation tasks. These include performing lookups with multiple criteria, managing dynamic ranges, and integrating other array functions for sophisticated reporting:

The following tutorials explain how to perform other common tasks in Google Sheets: