Learn How to Use INDEX and MATCH to Return Multiple Values Vertically in Excel

Authored by

Mohammed looti

October 28, 2025

The Challenge of Multi-Value Lookups

Traditional lookup methods in Excel, such as the widely used VLOOKUP function or the standard combination of INDEX and MATCH function, are expertly designed to retrieve a single corresponding value based on a specific criterion. While these tools are essential for basic data retrieval, real-world data analysis frequently demands the extraction of **multiple values** associated with a single lookup key. This requirement presents a significant limitation for standard functions, which typically stop searching after the first match is found.

When dealing with datasets where criteria are repeated--for example, listing all employees belonging to a single department or all products sold by a specific vendor--a more sophisticated, iterative approach is necessary. The goal is not just to identify the existence of a match, but to systematically locate the row number of every single occurrence and return the corresponding data points sequentially in a vertical list. Successfully implementing this requires moving beyond basic functions and embracing the power of combination formulas, providing a robust solution for complex data extraction needs.

The Advanced Array Formula Solution

To overcome the limitations of single-value lookups, we construct a powerful combined formula, often referred to as an array formula, that leverages several key Excel functions. This method integrates the retrieval capability of the INDEX function with the positional logic of SMALL function, the conditional testing provided by the IF function, and robust error handling via the IFERROR function. This complex structure allows the formula to iterate through the data, identify all matches, and return their corresponding values vertically, creating a dynamic filtered list.

The following formula represents the fundamental approach used to achieve this sequential, multi-value extraction in Excel. It is important to understand that this formula must be entered and treated as an array formula in traditional Excel versions, though modern versions handle it differently.

=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 essence, this sophisticated structure is engineered to dynamically extract all corresponding values from the designated results range (**B2:B11**), specifically where the entries in the lookup range (**A2:A11**) precisely match the specified criterion located in cell **D2**. The result is a clean, automated list of all related records, making it an indispensable technique for advanced data manipulation and reporting.

Detailed Dissection of the Formula Components

Understanding the interplay between these nested functions is key to troubleshooting and adapting this technique for different datasets. The formula operates by first identifying the relative row numbers of all matches, then sequentially selecting those row numbers, and finally using the INDEX function to retrieve the corresponding data.

`IFERROR(..., "")`: The Error Handler

This is the outermost layer, functioning as essential error handling. When the formula is dragged down past the last successful match, the inner components will inevitably return an error value, such as `#NUM!`, indicating no more matches exist. The IFERROR function intercepts this error and gracefully replaces it with an empty string (`""`). This results in a professional and clean output list that terminates after the last relevant entry.

`INDEX($B$2:$B$11, ...)`: The Retriever

The INDEX function is the core mechanism that returns the actual value we seek. It requires two main arguments: the array to pull data from (**$B$2:$B$11**, locked with absolute references) and the row number within that array. The complexity of the rest of the formula is solely dedicated to calculating and supplying the correct relative row number for each sequential match.

`SMALL(..., ROW(1:1))`: The Sequencer

The SMALL function is pivotal for iteration. It finds the k-th smallest value in a dataset. The array provided to `SMALL` consists of the relative row numbers of all matching items. Crucially, the 'k' argument is supplied by `ROW(1:1)`. When this formula is copied down one cell, `ROW(1:1)` automatically increments to `ROW(2:2)`, then `ROW(3:3)`, and so on. This dynamic increment forces `SMALL` to sequentially return the 1st, 2nd, 3rd, etc., smallest relative row number of the matches, ensuring we retrieve every matching record one after the other.

`IF($D$2=$A$2:$A$11, ROW($A$2:$A$11)-ROW($A$2)+1)`: The Match Identifier and Positional Calculator

This is the engine of the array formula, responsible for generating the list of relative positions for all successful lookups.

`$D$2=$A$2:$A$11`: This section performs a logical comparison across the entire lookup range (**A2:A11**) against the lookup value in **D2**. Because it is an array operation, it returns an array of `TRUE` (match) and `FALSE` (no match) values.
`ROW($A$2:$A$11)-ROW($A$2)+1`: This calculation determines the relative position. `ROW($A$2:$A$11)` yields the absolute row numbers (e.g., {2; 3; ...; 11}). By subtracting the starting

row number of the range (`ROW($A$2)`, which is 2) and adding 1, we convert these absolute positions into relative row indexes starting at 1 (e.g., {1; 2; ...; 10}).

When combined, the `IF` function substitutes the relative row number for every `TRUE` condition (match) and returns `FALSE` for every non-match. The result is an array passed to the `SMALL` function, containing only the relative row indices of the successful matches, interspersed with `FALSE` values. For example, if matches occur on the 3rd and 7th rows of the lookup range, the array might look like `{FALSE; FALSE; 3; FALSE; FALSE; FALSE; 7; FALSE;...}`.

Practical Scenario: Extracting Specific Data

To ground this complex formula in a tangible use case, let us examine a typical data scenario involving a list of basketball players and their respective teams. Data analysts frequently face the need to filter and display subsets of data rapidly--a perfect application for this multi-value lookup technique. Imagine a source dataset where player names are listed alongside the team they represent.

The goal is straightforward: given a specific team name as the input criterion, we must generate a complete, vertical list of every player associated with that team. This task goes far beyond what a simple VLOOKUP function could achieve.

Here is an illustration of the foundational dataset we will be using for this demonstration:

	A	B	C	D	E	F
1	**Team**	**Player**				
2	Mavs	Don				
3	Mavs	Mike				
4	Heat	Kurt				
5	Nets	Andy				
6	Mavs	Brad				
7	Celtics	Chad				
8	Nets	Derrick				
9	Celtics	Mark				
10	Mavs	Luke				
11	Nets	Dean				
12						
13						
14						
15						
16						
17						
18						
19						
20						

Our specific objective in this example is to filter the player list to include only those players who are members of the "Mavs" team. We require a solution that is both accurate and scalable, capable of handling any number of matching entries without manual adjustment.

Executing the Formula and Interpreting Results

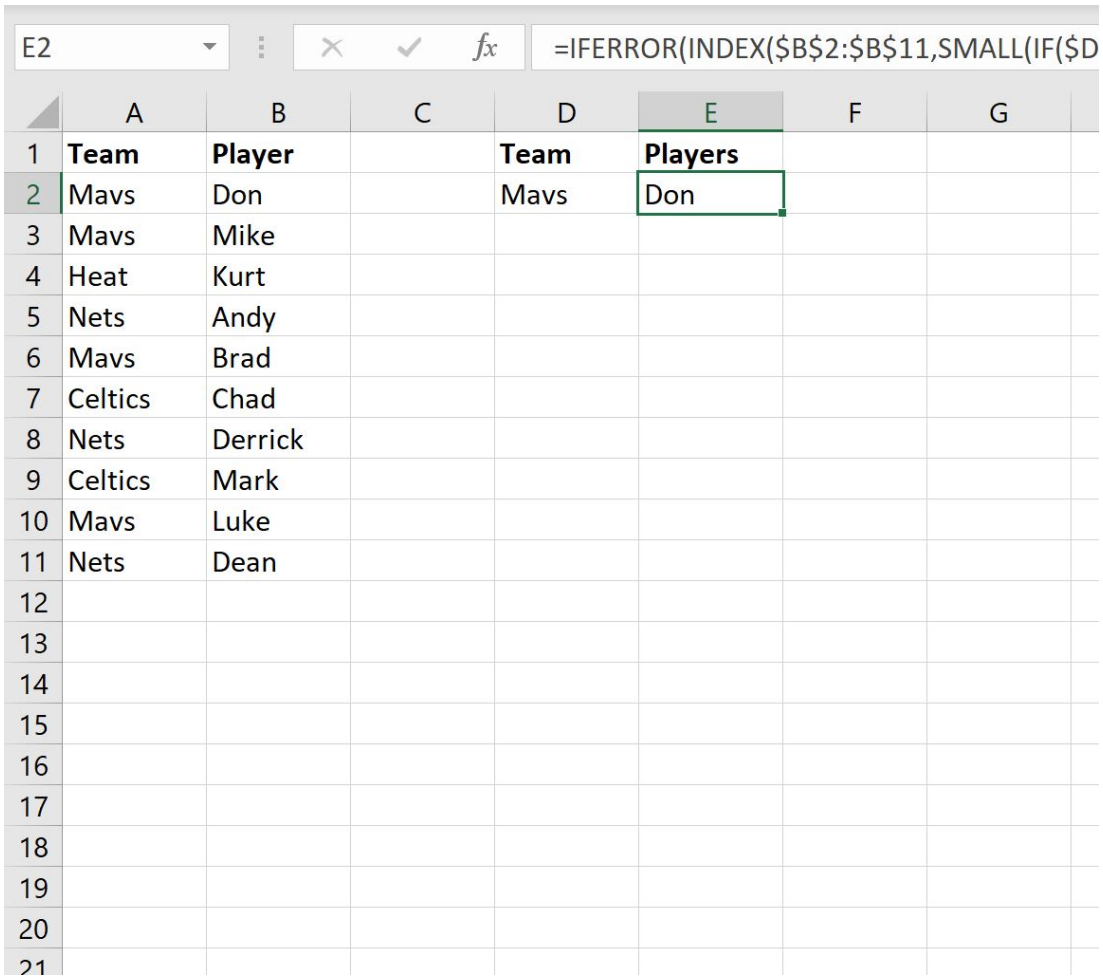
The implementation process begins by designating the lookup criterion. We will place the target team name, "Mavs," into cell **D2**. The array formula itself must then be entered into the first cell of the desired results column, which is **E2** in this scenario. The formula input is as follows:

=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))),"")

A critical step, particularly for users of older Excel versions (pre-Microsoft 365), is confirming the array formula by pressing **Ctrl + Shift + Enter** simultaneously. This action wraps the formula in curly braces `{ }`, signaling to Excel that it must process the data as an array. Users utilizing modern Excel versions with Dynamic Array functions can simply press **Enter**, and the results will

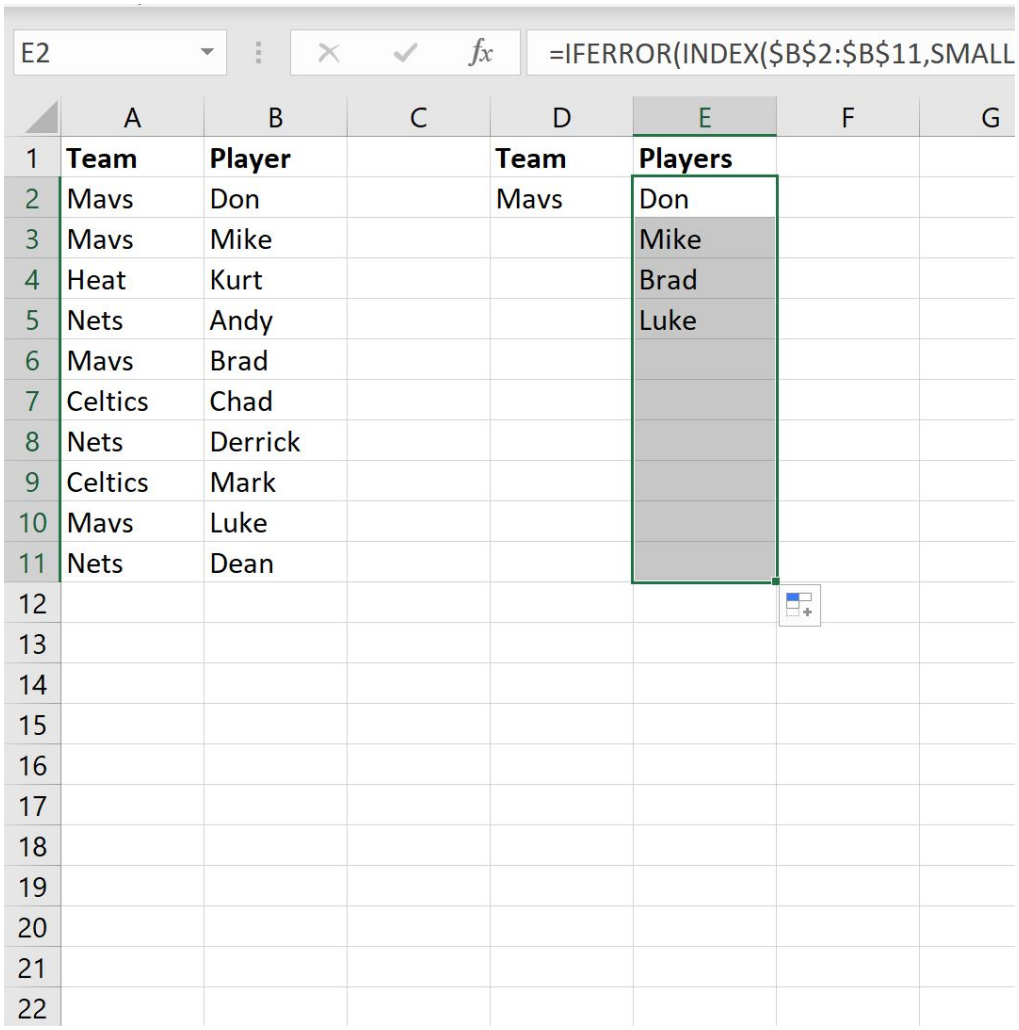
automatically "spill" down the column.

Once successfully entered, cell **E2** will immediately display the name of the first player found matching the criterion:



	A	B	C	D	E	F	G
1	**Team**	**Player**		**Team**	**Players**		
2	Mavs	Don		Mavs	Don		
3	Mavs	Mike					
4	Heat	Kurt					
5	Nets	Andy					
6	Mavs	Brad					
7	Celtics	Chad					
8	Nets	Derrick					
9	Celtics	Mark					
10	Mavs	Luke					
11	Nets	Dean					
12							
13							
14							
15							
16							
17							
18							
19							
20							
21							

The next step involves populating the rest of the results column. By dragging the formula down from cell **E2** to subsequent cells in column E, the `ROW(1:1)` component iterates, and the SMALL function sequentially retrieves the row index for every remaining match. This yields the complete, filtered list:



As demonstrated, all four players associated with the "Mavs" team are correctly identified and listed. The proactive inclusion of the IFERROR function is critical here, ensuring that the cells below the fourth match remain visually blank, preventing the appearance of disruptive `#NUM!` errors and maintaining professional report quality.

Leveraging Dynamic Adaptability

One of the most compelling features of this combined array formula approach is its inherent adaptability and dynamism. Since all ranges are secured with absolute references (e.g., `$A$2:$A$11`) and the lookup criterion is linked directly to a single cell (**D2**), the results in column E are intrinsically tied to the content of that criterion cell.

This means the formula does not need to be re-entered or modified if the filtration requirements change. If the user decides to search for players belonging to a different team--for example, changing the value in cell **D2** from "Mavs" to "Rockets"--the entire list in column E will instantaneously recalculate and update to display the corresponding players from the newly

specified team.

	A	B	C	D	E	F	G
1	**Team**	**Player**		**Team**	**Players**		
2	Mavs	Don		Nets	Andy		
3	Mavs	Mike			Derrick		
4	Heat	Kurt			Dean		
5	Nets	Andy					
6	Mavs	Brad					
7	Celtics	Chad					
8	Nets	Derrick					
9	Celtics	Mark					
10	Mavs	Luke					
11	Nets	Dean					
12							
13							
14							
15							
16							
17							
18							
19							
20							
21							

This reactive capability transforms a static data analysis task into a highly interactive process. The ability to perform quick, dynamic filtering based on varying criteria makes this formula solution ideally suited for developing advanced Excel dashboards, interactive reports, and data exploration tools that empower users without requiring them to manipulate the underlying formulas.

Conclusion and Modern Alternatives

The classic combination of the INDEX function, IF logic, SMALL function, and IFERROR function provides a robust, reliable, and powerful solution for the complex task of extracting multiple matching values vertically in any version of Excel. Mastering this technique significantly expands your data manipulation toolkit, enabling you to perform complex filtering and reporting tasks that surpass the capabilities of simpler lookup functions.

While this method remains essential for users on older Excel platforms, it is worth noting the evolution of the software. Users subscribed to Microsoft 365 now benefit from modern Dynamic Array functions, most notably the `FILTER` function. `FILTER` offers a far more streamlined and

intuitive syntax for achieving the exact same multi-value extraction results without the complexity of nested array logic and the need for Ctrl + Shift + Enter confirmation. Regardless of the version you use, understanding the mechanisms detailed here provides a foundational appreciation for advanced data handling.

Additional Resources for Enhanced Excel Proficiency

To further develop your mastery of advanced Excel techniques and data management strategies, explore these related topics and tutorials:

Techniques for using **INDEX and MATCH** to perform efficient horizontal lookups across rows rather than columns.

A comprehensive guide to understanding the intricacies and performance considerations of **array formulas** in Excel, including best practices.

Advanced methods for implementing robust **data validation** rules and utilizing conditional formatting to highlight specific data subsets.