How to Easily Find Multiple Values in Excel

Authored by

stats writer

November 28, 2025

One of the most common challenges faced by users of Excel is the need to perform lookups that return not just the first match, but all corresponding matches based on a specific criterion. Traditional lookup tools, such as the widely-used VLOOKUP function, are inherently designed to stop calculating once they find the first instance of a value, leaving subsequent matches untouched. This limitation necessitates the use of more complex, array-based solutions when dealing with dynamic data sets where duplicates are expected and must be fully extracted.

To overcome this hurdle and efficiently find multiple values in a sprawling data range, we must employ combinations of powerful native Excel formulas. Functions like INDEX, MATCH, and COUNTIF, when correctly nested, allow us to create a robust extraction mechanism. This guide delves into the advanced array formula technique--specifically using the INDEX, SMALL, IF, and ROW functions--to pull every associated record from a lookup column, providing a comprehensive solution for data analysis.

Understanding the Limitations of Standard Lookups

Most beginners in data management rely heavily on straightforward functions like VLOOKUP or its counterpart, XLOOKUP. While these tools are indispensable for simple, one-to-one mapping tasks (e.g., finding a single product price corresponding to a unique ID), they fail spectacularly when the data is structured as one-to-many. If a specific employee name appears four times in a column, VLOOKUP will only identify the entry associated with the first row it encounters, ignoring the three subsequent matches entirely. This behavioral constraint highlights the need for advanced methods whenever searching for multiple corresponding values.

In scenarios requiring the extraction of several related records, such as isolating all sales transactions for a single sales agent, combining functions becomes mandatory. Historically, users might attempt complex nested VLOOKUP structures or perhaps use helper columns paired with the COUNTIF function to assign unique identifiers to duplicate entries before performing a lookup. However, these methods often introduce complexity and rely on modifying the source dataset structure. The most elegant and efficient solution involves creating a powerful array formula that dynamically identifies the row numbers of all matching entries.

Introducing the Array Formula Approach

The standard methodology for retrieving multiple matching records involves utilizing a complex, yet highly effective, array formula built around the INDEX, SMALL, IF, and ROW functions. This combination allows Excel to perform a calculation across an entire range simultaneously, generating an array of row numbers that correspond to the matching criteria. This array is then iteratively processed to extract the desired output values, one match at a time, into separate cells.

The core philosophy of this technique centers on converting the lookup process from a linear,

single-match operation into a matrix evaluation. The INDEX function is critical here, as it fetches a value from a specific position within a range, while the nested SMALL(IF(...)) construct determines exactly which row number INDEX should pull from. By carefully controlling the arguments within this nested structure, we can ensure that subsequent calculations pull the second, third, and fourth matches, until all corresponding records are displayed. This method provides superior accuracy and robustness compared to manual lookups or complicated helper columns.

Deconstructing the Advanced Lookup Formula Structure

The specialized array formula used to extract multiple values in Excel combines four functions to achieve its powerful result. Below is the structure, followed by a detailed breakdown:

=INDEX($A$1:$B$12,SMALL(IF($A$1:$A$12=$F$1,ROW($A$1:$A$12)),ROW(1:1)),2)

This particular formula is configured to find all of the values in the result range (column **B1:B12**) where the corresponding value in the criteria range (column **A1:A12**) is equal to the search key residing in cell **F1**.

Let's analyze the components, which are crucial for understanding how the extraction works. The outermost function is INDEX, which takes three main arguments: the array to return values from (`$A$1:$B$12`), the row number to return (calculated by the SMALL function), and the column number to return (`2`, representing column B). The goal of the nested functions is simply to feed the correct row number into INDEX for sequential extraction.

The inner core is the `IF($A$1:$A$12=$F$1, ROW($A$1:$A$12))` section. Here, the IF statement checks every cell in the lookup range `$A$1:$A$12` against the criterion cell `$F$1`. If the condition is TRUE (i.e., a match is found), the IF function returns the actual absolute row number using the `ROW($A$1:$A$12)` function. If the condition is FALSE, IF returns `FALSE`. This operation creates an array containing only the row numbers of matching entries and `FALSE` values for non-matches.

Finally, the `SMALL(..., ROW(1:1))` wraps the IF statement. The SMALL function is used to retrieve the K-th smallest value from the array generated by IF. The key trick here is using `ROW(1:1)` as the K argument. When this formula is dragged down using the Autofill handle, `ROW(1:1)` increments to `ROW(2:2)`, `ROW(3:3)`, and so on. This dynamic change causes SMALL to sequentially pull the 1st, 2nd, 3rd, etc., smallest row number corresponding to a match, thereby ensuring every single match is retrieved in sequence.

Setting Up the Practical Example

To solidify this understanding, let us walk through a practical example involving an employee sales

dataset. Suppose we maintain records in Excel detailing which employees sold various products within a company, and we need to isolate all products sold by a single employee, regardless of how many times their name appears.

The dataset must be structured appropriately, with the employee names in one column (the criteria column) and the product names in an adjacent column (the result column). For this demonstration, our data spans cells A1 through B12, representing our overall array range for the INDEX function.

Suppose we have the following dataset in Excel that shows which employees sold various products at some company:

	A	B	C	D	E	F
1	**Employee**	**Product**				
2	Jan	Apples				
3	Mike	Oranges				
4	Evan	Oranges				
5	Jan	Bananas				
6	Mike	Kiwis				
7	Mike	Apples				
8	Tom	Berries				
9	Mike	Bananas				
10	Evan	Berries				
11	Evan	Apples				
12	Jan	Kiwis				
13						
14						
15						
16						
17						
18						
19						

In this scenario, Column A holds the employee names, and Column B holds the products they sold. Notice that the employee "Mike" appears multiple times, associated with different products. Our objective is to generate a clean, consolidated list of all products specifically sold by Mike, without manually filtering or sorting the source data.

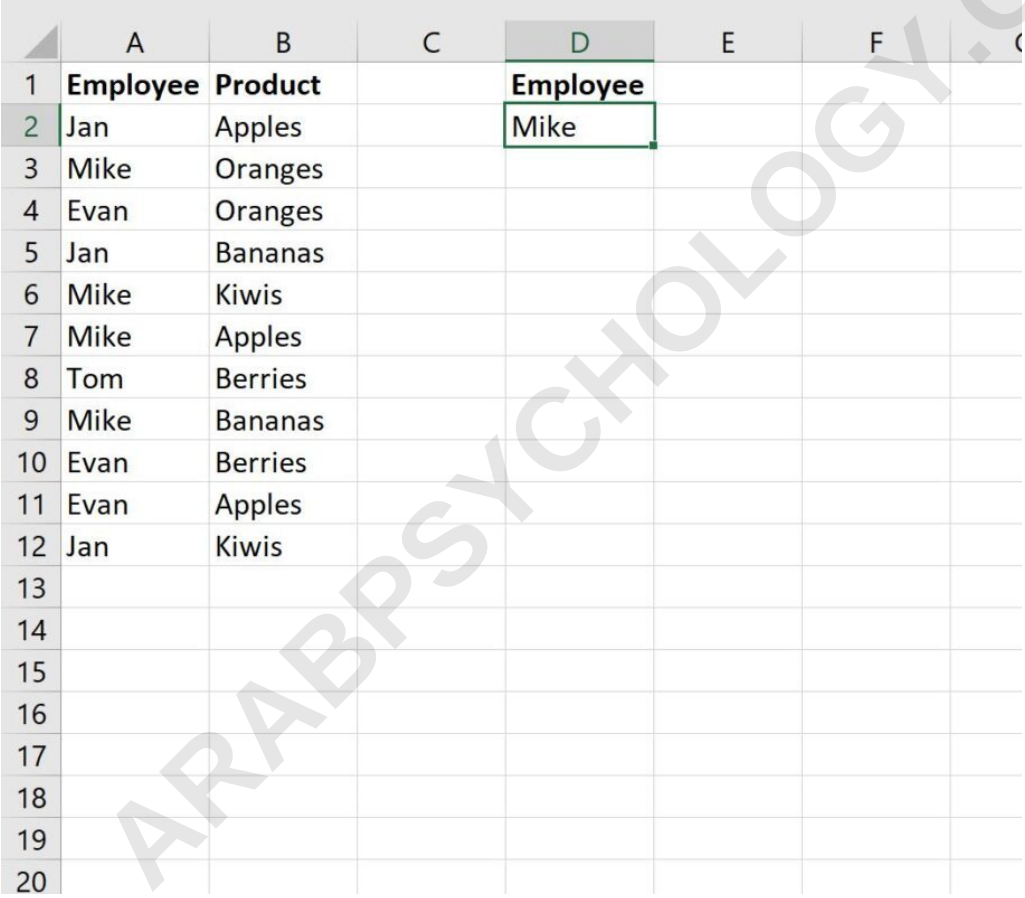
Defining the Search Criterion

Before applying the advanced formula, we must define the search term clearly. It is best practice to

dedicate a separate cell for the lookup value, as this allows the formula to reference the criterion dynamically. This is crucial for easy modification of the search query later, enabling the user to quickly swap out "Mike" for "Sarah" or "John" without having to edit the lengthy array formula itself.

In our example, we designate cell **D2** as the location for the employee name we wish to search for. By centralizing the criteria, we ensure that the entire extraction process remains flexible and scalable. This cell serves the same function as `$F$1` in the generalized formula structure provided earlier.

Now suppose we would like to find all of the products sold by Mike. To do so, we can type his name in cell **D2**:



	A	B	C	D	E	F
1	**Employee**	**Product**		**Employee**		
2	Jan	Apples		Mike		
3	Mike	Oranges				
4	Evan	Oranges				
5	Jan	Bananas				
6	Mike	Kiwis				
7	Mike	Apples				
8	Tom	Berries				
9	Mike	Bananas				
10	Evan	Berries				
11	Evan	Apples				
12	Jan	Kiwis				
13						
14						
15						
16						
17						
18						
19						
20						

Applying the Array Formula for the First Match

The next step involves inputting the calculated array formula into the cell where we want the first result to appear. Following our setup, this will be cell **E2**. The formula must be carefully entered, ensuring that all ranges are absolute references (using dollar signs, e.g., `$A$1:$B$12`) except for the dynamic `ROW(1:1)` component, which must be relative to allow for iteration.

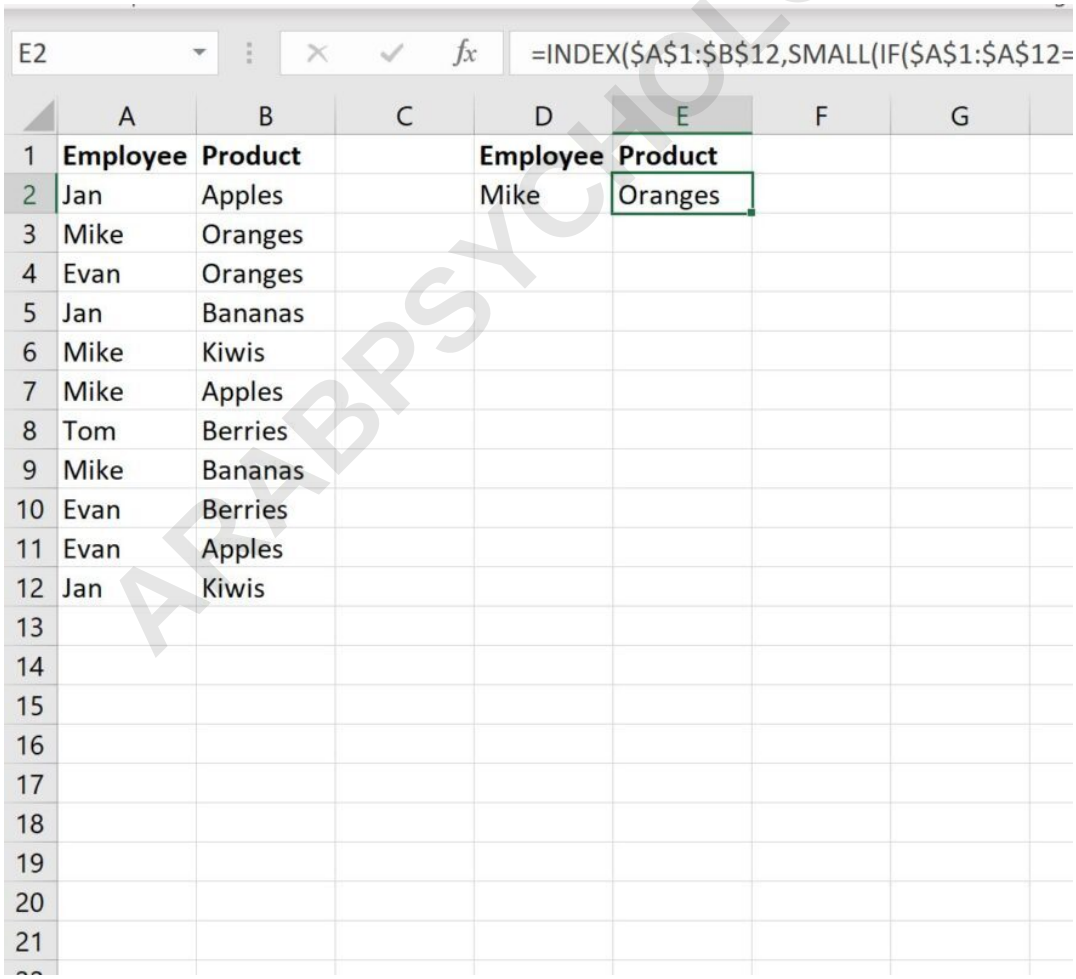
The formula structure must now be adapted to reference our specific ranges: the entire data array is `$A$1:$B$12`, the criteria column is `$A$1:$A$12`, and the search criterion cell is `$D$2`. The column index for the results (Products) is `2`.

Then we can type the following formula into cell **E2**:

=INDEX($A$1:$B$12,SMALL(IF($A$1:$A$12=$D$2,ROW($A$1:$A$12)),ROW(1:1)),2)

A crucial technical detail is the execution of array formulas in older versions of Excel (pre-Microsoft 365 or Excel 2021). If you are using an older version, you must press **CTRL + SHIFT + ENTER** instead of just ENTER after typing the formula. This key combination signals to Excel that the calculation must be performed as an array operation, encapsulating the formula in curly braces `{}` automatically. If you are using a modern version that supports Dynamic Arrays, pressing ENTER alone may suffice, but using CTRL + SHIFT + ENTER remains the safest standard practice.

This will return the first product sold by Mike:

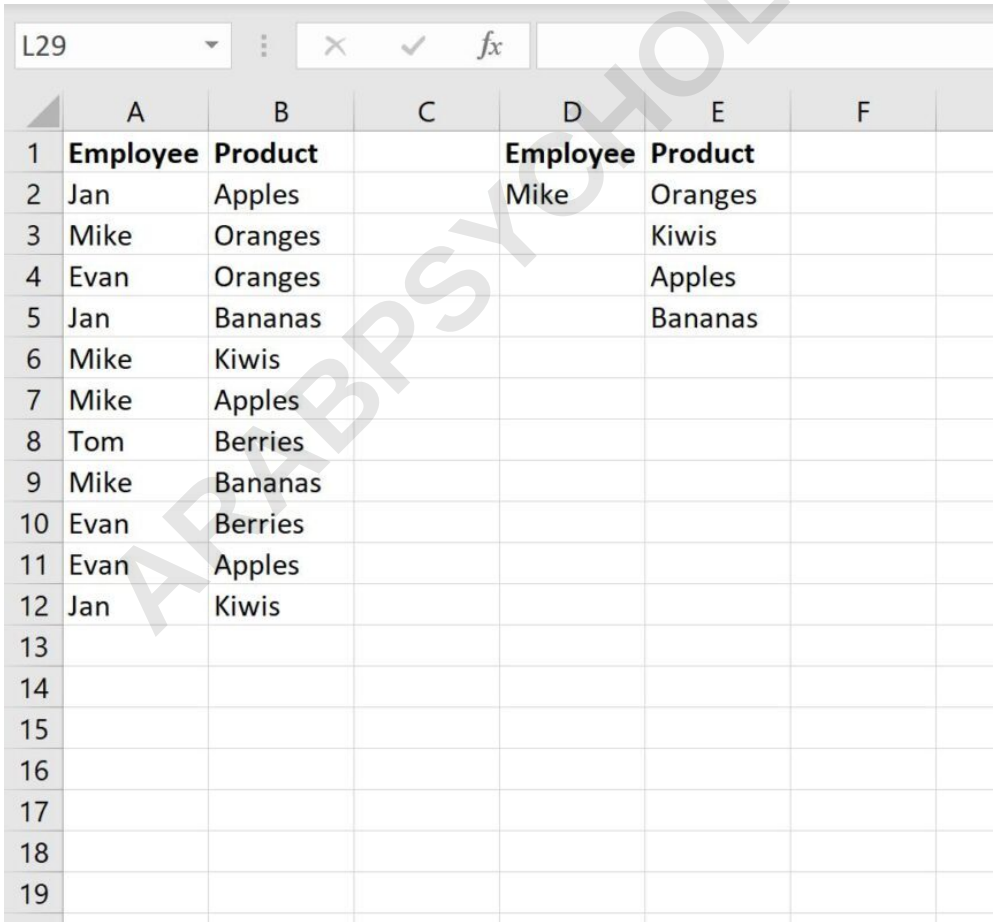


Iterating Results Using Autofill

Once the array formula is correctly entered in the first result cell (E2) and confirmed (potentially with CTRL + SHIFT + ENTER), the remaining matching records can be retrieved effortlessly using the Autofill feature. The inherent design of the `ROW(1:1)` function ensures that as the formula is copied down, the value it returns increments sequentially (2, 3, 4, etc.), instructing the SMALL function to look for the next available matching row number.

To execute this, click on cell E2, locate the small square box (the Autofill handle) in the bottom right corner, and drag it downwards across the cells in Column E where results are expected. It is crucial to drag down far enough to capture all potential matches. If the formula is dragged past the point where matches exist, it will return an error, typically `#NUM!`, indicating that the SMALL function could not find the specified K-th smallest row number.

We can then Autofill this formula down to the remaining cells in column E to find all products sold by Mike:



Reviewing and Utilizing the Results

The output in Column E now provides a comprehensive list of all products associated with the search criterion, "Mike." Any cells returning the `#NUM!` error indicate that no further matches were found within the dataset, confirming the list is complete. For presentation purposes, these error cells can be hidden or managed by wrapping the primary formula in an IFERROR or IFNA function, allowing them to display a blank cell (`""`) instead of an error message.

In our demonstrated case, the extraction successfully pulled four unique records associated with the employee Mike. This extraction confirms the power of array formulas in tackling complex many-to-one lookup scenarios where standard functions would fail. The retrieved list is ready for further analysis, reporting, or integration into other parts of the spreadsheet.

We can now see all four products sold by Mike:

Oranges
Kiwis
Apples
Bananas

We can look at the original data in columns A and B to confirm that Mike indeed sold all four of these products. This highly adaptable method serves as a foundation for advanced data management tasks in Excel.