How to Use “NOT LIKE” in VBA (With Examples)

Authored by

stats writer

November 19, 2025

The use of pattern matching is fundamental when manipulating data within Visual Basic for Applications (VBA). While the standard LIKE operator checks for inclusion, the combination of the **Not** keyword with **Like** creates a powerful mechanism for exclusion, allowing developers to identify records or data points that specifically **do not match** a specified pattern. This construct operates as a Boolean operator, returning either true or false based on the outcome of the string comparison.

This technique is indispensable when performing complex data filtering, such as searching a large dataset for records that lack certain prefixes, suffixes, or embedded substrings. For instance, if you are managing inventory, you might need to quickly isolate product codes that do not contain 'DISCONTINUED' or identify email addresses that are missing the expected domain structure. Mastering the **Not Like** syntax provides the precision required for these advanced querying tasks, significantly enhancing the automation capabilities of your VBA macros.

Understanding the `NOT LIKE` Construct in VBA

In VBA, the standard `LIKE` operator is designed to perform flexible pattern matching against a text string. It returns `True` if the string conforms to the pattern, and `False` otherwise. The addition of the `Not` keyword flips this logical outcome entirely. When placed before the `LIKE` operation, the entire expression evaluates to `True` only when the string fails to match the specified pattern. This reversal is crucial for filtering data where non-conformance is the primary criterion for selection.

The syntax for this operation is straightforward: `If Not Like Then...`. This structure ensures that the conditional block of code executes only when the target data element actively deviates from the defined pattern. This approach is superior to simply checking for inequality (``) because it utilizes the power of wildcard characters and character lists inherent to the LIKE operator, allowing for sophisticated, partial, or flexible matching rules. Without the `Not Like` construct, achieving the same exclusion logic would require significantly more complex functions involving multiple `InStr` or regular expressions.

It is essential to recognize that the `Not` keyword acts upon the result of the `LIKE` evaluation. If `Range("A1") Like "*test*"` returns `False`, then `Not Range("A1") Like "*test*"` returns `True`. This relationship forms the core of exclusion-based logic in VBA programming, offering a clear and concise way to handle filtering requirements that prioritize negative matches.

Syntax and Mechanics of the `LIKE` Operator

To effectively use the `NOT LIKE` construct, a thorough understanding of the underlying LIKE operator and its associated wildcard characters is necessary. The LIKE operator utilizes special characters to represent unknown or variable parts of a string. These characters allow you to search

for patterns rather than exact matches, which is crucial for flexible data handling.

The primary wildcard characters used in wildcard characters matching include:

Asterisk (`*`): Represents any sequence of zero or more characters. For example, `A*B` matches 'AB', 'A1B', or 'APPLESAUCE BOWL'. This is perhaps the most commonly used wildcard for substring detection.
Question Mark (`?`): Represents any single character. For example, `C?T` matches 'CAT', 'COT', or 'CUT', but not 'CART'.
Pound Sign (`#`): Represents any single digit (0 through 9). This is vital when matching patterns containing numerical segments, such as serial numbers or version codes.
Brackets (``):** Used to match any single character within the specified set or range. For example, `` matches 'A', 'B', or 'C'. Using an exclamation point within the brackets (``) negates the set, matching any single character **not within the specified range. This negation feature is an advanced form of exclusion built directly into the pattern definition.

When constructing a pattern for a `NOT LIKE` comparison, the pattern must define the undesirable structure. If you wish to exclude any string that contains "ERROR" anywhere, the pattern must be `"*ERROR*"`. The `NOT LIKE` operator then ensures that only strings that successfully navigate this exclusion filter are processed. Understanding the nuances of these patterns is essential for writing efficient and accurate conditional logic in your VBA code.

Practical Implementation: Checking for Non-Matching Patterns

The most frequent use case for the `NOT LIKE` construct in VBA involves iterating through a collection of cells or records and filtering based on whether or not a specific substring exists. By combining the `Not` keyword with the pattern matching capabilities of `Like`, developers can create highly efficient data validation and cleaning routines.

Consider a scenario where you are processing a spreadsheet containing product descriptions, and you need to tag any item that does **not** contain the required safety certification identifier, represented here by the simple term "hot". We can loop through the relevant cells and apply the `Not Like` check. This approach bypasses the need for complex nested logic or auxiliary functions, keeping the code clean and focused on the core exclusion rule.

The following example demonstrates how to implement this pattern checking across a range of cells, specifically iterating through the range A2:A10 and outputting the determined status in the corresponding cells of column B. Notice how the use of the asterisk wildcard (`*`) is crucial for performing a substring search, meaning we are checking if the pattern exists anywhere within the target string.

```
Sub CheckNotLike()

Dim i As Integer

For i = 2 To 10
If Not Range("A" & i) Like "*hot*" Then
Range("B" & i) = "Does Not Contain hot"
Else
Range("B" & i) = "Contains hot"
End If
Next i

End Sub
```

This macro encapsulates the core logic of using the exclusion operator. By setting the condition to evaluate the negative match (`Not ... Like`), the code immediately jumps to the appropriate handling block, making the intention of the conditional statement explicit and easy to debug.

Step-by-Step Code Example Breakdown

Let us analyze the provided macro in detail, focusing on how the loop and the conditional logic work together to achieve the desired filtering outcome. This code serves as a robust template for applying pattern-based exclusion across tabular data in Excel.

Initialization: The code begins with `Sub CheckNotLike()` and declares an Integer variable `i`. This variable will serve as the row counter for our iteration.
Iteration Loop: The `For i = 2 To 10` statement initiates a loop that processes rows 2 through 10 of the active worksheet. This loop structure ensures that every relevant data point in column A is checked against the exclusion criteria.
Conditional Exclusion: The line `If Not Range("A" & i) Like "*hot*" Then` is the heart of the operation.

`Range("A" & i)` retrieves the value of the current cell in column A.
`Like "*hot*"` attempts to match any string containing the substring "hot".
The preceding `Not` keyword inverts the result. If the cell **does not** contain "hot", the `Like` comparison is `False`, and `Not False` becomes `True`, executing the subsequent code block.

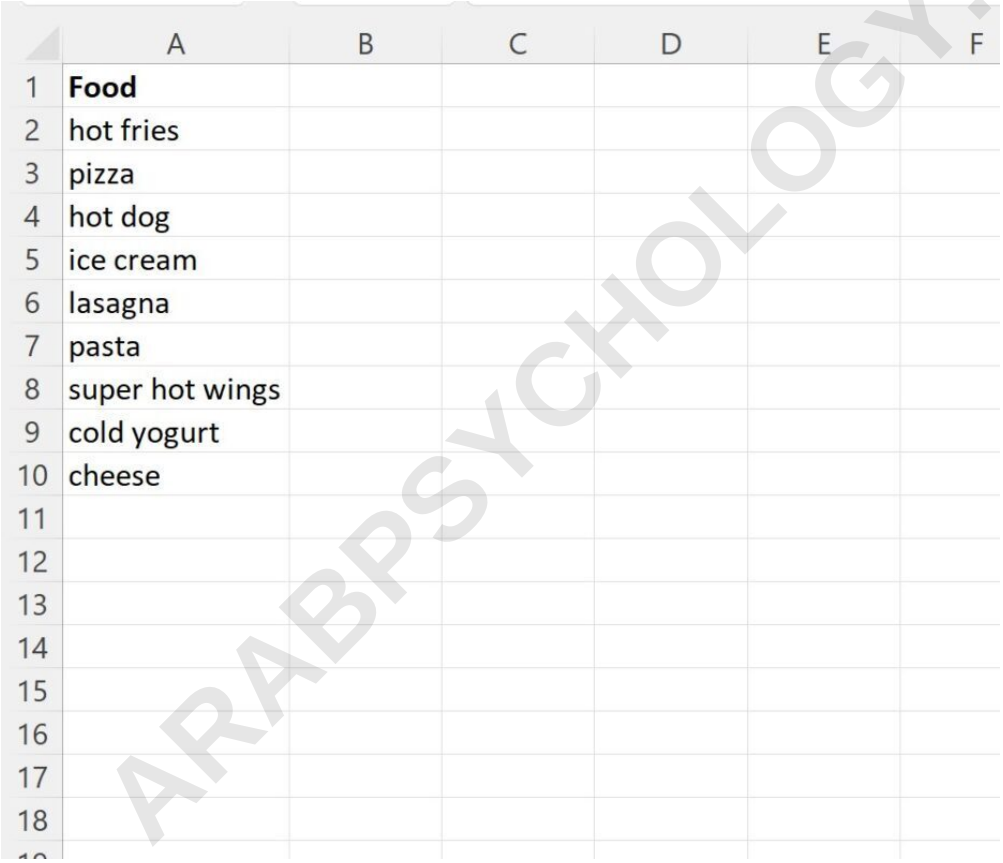
Result Handling: If the condition is `True` (meaning the cell does not match the pattern), the code executes `Range("B" & i) = "Does Not Contain hot"`. Conversely, the `Else` block executes if the cell **does** contain "hot", setting the output to "Contains hot".
Completion: The loop finishes with `Next i`, and the subroutine concludes with `End Sub`.

This structured approach ensures that the logic is clear: we are explicitly testing for the absence of the pattern defined by the LIKE operator, and assigning output accordingly. The use of the asterisk (`*`) is critical, as it confirms that the search for "hot" is non-positional, looking for the substring anywhere within the cell content.

How to Use NOT LIKE in VBA: A Demonstration

To provide a clear visualization of the utility of the `NOT LIKE` construct, let us apply this code to a real-world list of food items. Suppose we are tracking an inventory list in Excel where column A contains various food descriptions.

We start with the following list in column A:



	A	B	C	D	E	F
1	**Food**					
2	hot fries					
3	pizza					
4	hot dog					
5	ice cream					
6	lasagna					
7	pasta					
8	super hot wings					
9	cold yogurt					
10	cheese					
11						
12						
13						
14						
15						
16						
17						
18						

We want to use our previously defined macro, `CheckNotLike()`, to automatically determine which items do not contain the substring "hot". This is particularly useful if "hot" signifies a spicy or special preparation that needs separate classification.

We apply the exact same macro code:

Sub CheckNotLike()

```
Dim i As Integer

For i = 2 To 10
If Not Range("A" & i) Like "*hot*" Then
Range("B" & i) = "Does Not Contain hot"
Else
Range("B" & i) = "Contains hot"
End If
Next i

End Sub
```

Upon execution of the macro, the results are populated immediately in column B, providing a clear classification of the data based on the absence or presence of the target pattern.

	A	B	C	D	E
1	**Food**				
2	hot fries	Contains hot			
3	pizza	Does Not Contain hot			
4	hot dog	Contains hot			
5	ice cream	Does Not Contain hot			
6	lasagna	Does Not Contain hot			
7	pasta	Does Not Contain hot			
8	super hot wings	Contains hot			
9	cold yogurt	Does Not Contain hot			
10	cheese	Does Not Contain hot			
11					
12					
13					
14					
15					
16					
17					
18					

As illustrated, the output in Column B accurately reflects whether the corresponding cell in Column A contains the substring "hot". Items like "Chicken Strips" and "Pizza" successfully pass the `NOT LIKE "*hot*"` test, whereas items like "Hot Sauce" and "Hot Dog" fail the exclusion test (meaning they match the pattern, and thus `NOT LIKE` returns `False`), falling into the `Else` category.

Note: We utilized asterisks (*****) around the substring to indicate that any character sequence can come before or after the string "hot" in the cell. This makes the search non-positional and highly flexible for identifying embedded patterns.

Advanced Exclusion Techniques with `NOT LIKE`

While checking for simple substrings is the most common application, the power of `NOT LIKE` truly shines when combined with more complex wildcard characters. Advanced exclusion allows you to filter out strings based on specific positions, character types, or ranges.

Consider the need to filter out product codes that do not conform to a structure where the third character is a digit. You could use the pattern `??#*`. Applying the `NOT` keyword allows you to select only those strings that fail this structural validation: `If Not productCode Like "??#*" Then...` This would identify codes like 'ABX123' or 'A-B456' as valid matches for the pattern (and thus excluded by `NOT LIKE`), while codes like 'ABCDE' or '12345' would be selected because they do not have a digit in the third position.

Furthermore, using the bracket negation feature allows for highly specific exclusions. Suppose you want to ensure a column only contains entries that start with a capital letter, but not the letters A, B, or C. The pattern to exclude would be `*`. The conditional check would be: `If Not targetString Like "*" Then`. This line of code would return `True` for strings starting with 'D' or 'X', but `False` for strings starting with 'A', 'B', or 'C', thereby filtering out the unwanted entries. This level of precise character exclusion makes the `NOT LIKE` combination a powerful tool for data normalization and integrity checks.

Case Sensitivity and Best Practices

One critical aspect of using the LIKE operator, and consequently `NOT LIKE`, is the handling of case sensitivity. By default, VBA performs case-insensitive comparisons when evaluating string operations. This means that, by default, searching for `"*hot*"` will match "Hot", "hOT", or "cHoti".

If your application requires case-sensitive pattern matching--for example, distinguishing between a product code 'T-shirt' and a variable 'T-SHIRT'--you must explicitly set the comparison method. This is achieved by placing the statement `Option Compare Binary` at the very top of your module, before any procedures. If this option is set, the comparison becomes case-sensitive, and the `NOT LIKE` operator will adhere to this stricter requirement. If case-insensitivity is desired (which is usually the default and preferred for user-input pattern matching), ensure that `Option Compare Text` is used, or simply rely on the default settings.

Finally, when dealing with large datasets, it is a best practice to pre-filter data where possible before initiating a loop using `NOT LIKE`. While the operator is efficient, repeatedly calling

`Range()` objects within a loop can be slow. Loading the data into an array first and performing the Boolean operator comparisons on array elements will drastically improve performance, especially when processing thousands of rows. Always prioritize array manipulation over direct cell manipulation within extensive loops.

The following tutorials explain how to perform other common tasks using VBA: