Learn How to Combine Multiple Ranges in Excel VBA Using the Union Method

Authored by

Mohammed looti

November 9, 2025

Introduction to the VBA Union Method

When automating sophisticated processes in Excel using VBA (Visual Basic for Applications), developers often encounter situations requiring simultaneous manipulation of multiple, non-contiguous areas of a worksheet. While interacting with a single, continuous selection is simple, applying a single operation across several disconnected data sets demands a specialized technique. This is precisely where the **Union** method becomes an indispensable tool for efficient coding.

The **Union** method, accessed via the Application object, is meticulously designed to merge two or more discrete Range objects into one unified, cohesive object. This consolidation capability is crucial because it allows complex actions--such as applying formatting, assigning complex formulas, or manipulating data--to be executed with a single instruction. This approach significantly streamlines code, eliminating the need for tedious iterative loops or repetitive code blocks targeting each separate area individually.

Mastering the invocation and management of the resulting combined object from the **Union** method is fundamental for writing robust, scalable, and high-performing macros. It provides a clean solution for handling complex selections and is a staple technique when managing dynamic data environments where the necessary input areas may frequently shift or be physically separated on the spreadsheet.

Syntax and Implementation Fundamentals

The use of the Union method is executed through the main Excel **Application** object. Critically, it requires a minimum of two arguments, each of which must be a valid Range object reference. The method's output is a brand new **Range** object that collectively represents the total area covered by all specified input ranges. Since this method returns an object, it is mandatory in VBA to utilize the Set keyword when assigning the result to any declared variable.

The general syntax structure for calling this method is highly flexible and easy to remember:

```
Set CombinedRange = Application.Union(Range1, Range2, , ...)
```

This structure clearly shows that while two ranges are required to initiate the union, the method efficiently accepts numerous optional arguments. This powerful flexibility enables developers to seamlessly merge any number of disparate selections--from two to dozens--into a single, unified selection set. Once created, this combined range can be treated as a singular entity for all subsequent operations, whether that involves coloring the background or updating cell values with a complex formula.

To demonstrate this principle, consider a basic implementation that highlights the core technique of

creating and acting upon a combined range:

```
Sub UseUnion()

Set UnionRange = Application.Union(Range("A1:A10"), Range("C1:C10"))
UnionRange.Formula = "=RANDBETWEEN(1, 100)"

End Sub
```

In this simple macro, we effectively merge the cell block **A1:A10** with **C1:C10**. Immediately following this consolidation, the code assigns the Excel formula **=RANDBETWEEN(1, 100)** to every cell contained within the newly established **UnionRange** object. This process results in the rapid population of both distinct columns with random integer values, demonstrating the immediate advantage of range unification.

Practical Demonstration: Combining Two Disjoint Ranges

To fully appreciate the efficiency and clarity provided by the **Union** method, let's expand on the scenario where data needs to be populated across two non-adjacent columns. If we were to perform this task manually, we would need to select and process each column separately, or write redundant code blocks. The **Union** command resolves this by handling both ranges simultaneously within a single, powerful line of VBA.

The following procedure defines and utilizes the Union method to explicitly merge the areas **A1:A10** and **C1:C10**. This crucial action generates a virtual collection of cells that can be acted upon as a unified group, despite the substantial physical gap separating them on the active worksheet. The resulting code is both concise and highly readable:

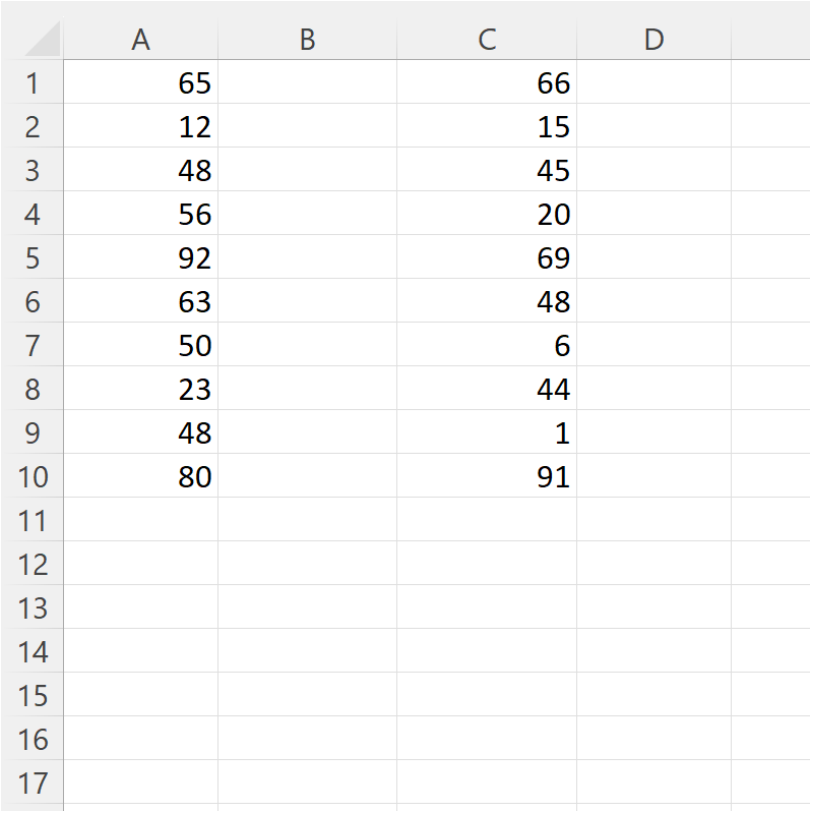
```
Sub UseUnion()

Set UnionRange = Application.Union(Range("A1:A10"), Range("C1:C10"))
UnionRange.Formula = "=RANDBETWEEN(1, 100)"

End Sub
```

The core operational line is `Set UnionRange = Application.Union(Range("A1:A10"), Range("C1:C10"))`. Once this Range object variable is set, the subsequent instruction, `UnionRange.Formula = "=RANDBETWEEN(1, 100)"`, efficiently applies the formula across all 20 cells (10 in column A and 10 in column C) using just one command. This demonstrates a significant gain in code clarity and performance compared to techniques that require iterating through or repeatedly selecting each range individually.

Visual Confirmation of the Combined Range Operation

Upon successful execution of the `UseUnion` procedure detailed above, the immediate effect is clearly evident on the Excel worksheet. The output confirms that the specified operation (formula assignment) was applied exclusively to the combined ranges, leaving the intervening column B completely untouched. This visual separation confirms the precision of the **Union** method.

When this macro runs, the result visually illustrates the dual-column population:



	A	B	C	D
1	65		66	
2	12		15	
3	48		45	
4	56		20	
5	92		69	
6	63		48	
7	50		6	
8	23		44	
9	48		1	
10	80		91	
11				
12				
13				
14				
15				
16				
17				

As depicted in the visualization, every cell within the designated ranges--specifically **A1:A10** and **C1:C10**--now contains the formula **=RANDBETWEEN(1, 100)**. The RANDBETWEEN function dynamically returns a random integer between the provided lower and upper bounds (1 and 100). This result confirms that the **Union** method successfully created a composite Range object that behaves as a single unit when performing complex actions like formula assignments.

Extending Functionality: Merging Three or More Ranges

A key benefit of the Union method is its exceptional scalability. Unlike certain other methods that may be limited to two arguments, `Application.Union` is engineered to fluidly manage multiple Range objects concurrently. This capability becomes critical when dealing with large or complex worksheets where necessary data is organized across several scattered areas that all require

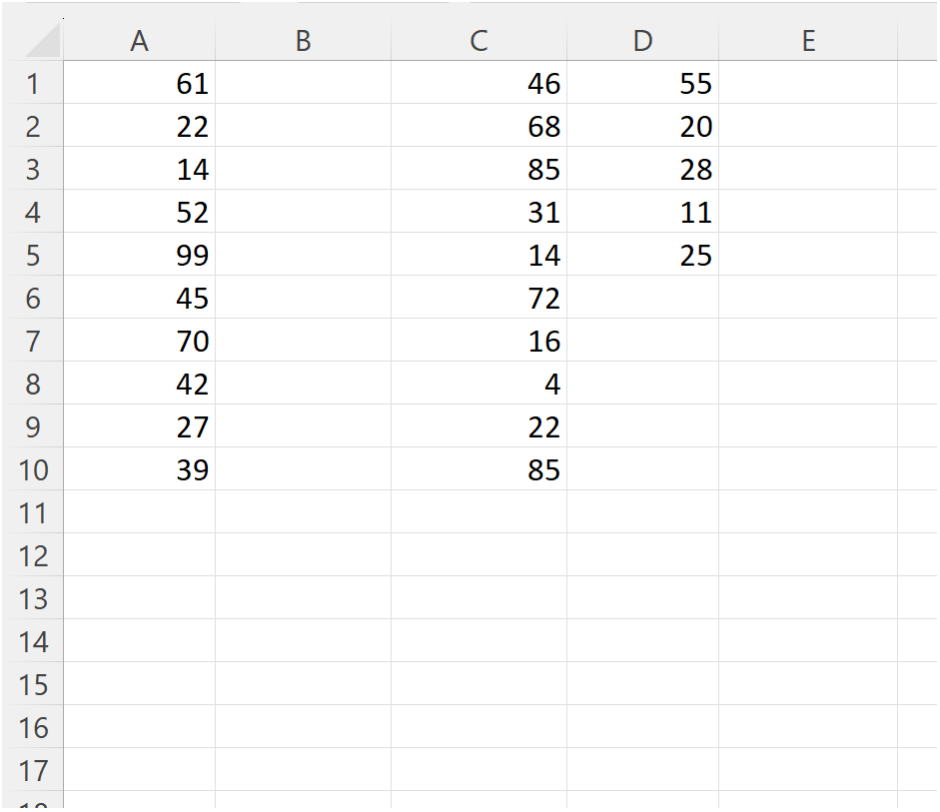
identical processing or updates.

For example, if the requirement is to apply the same randomization function across columns A, C, and a shorter, distinct section in column D, the developer simply needs to add the third range argument to the method call. The resulting combined range seamlessly encompasses all three defined areas, regardless of their location or size discrepancy.

The following extended VBA example illustrates how the **Union** method handles this complexity, merging **A1:A10**, **C1:C10**, and **D1:D5**. Following the merge, the code efficiently inserts the RANDBETWEEN formula into the cells of all three combined areas simultaneously:

```
Sub UseUnion()

Set UnionRange = Application.Union(Range("A1:A10"), Range("C1:C10"), Range("D1:D5"))
UnionRange.Formula = "=RANDBETWEEN(1, 100)"

End Sub
```

When this modified macro is executed, the resulting output clearly demonstrates the successful merging and collective manipulation of the three distinct zones:



	A	B	C	D	E
1	61		46	55	
2	22		68	20	
3	14		85	28	
4	52		31	11	
5	99		14	25	
6	45		72		
7	70		16		
8	42		4		
9	27		22		
10	39		85		
11					
12					
13					
14					
15					
16					
17					

This visual confirmation verifies that all three specified ranges--irrespective of their unique dimensions or locations--now contain the **RANDBETWEEN** formula. This strongly emphasizes the substantial efficiency gains provided by the **Union** method when performing collective operations across multiple, scattered data sets.

Advanced Considerations and Best Practices

While the **Union** method is undeniably a powerful utility, developers must adhere to several best practices and advanced considerations to ensure their VBA code remains stable, error-free, and performs optimally under various conditions. One crucial area relates to robust error handling. If any argument passed to `Application.Union` fails to resolve into a valid **Range** object (for instance, if a variable intended to hold a range reference has not been properly initialized), the code will inevitably trigger a run-time error. It is therefore paramount to implement comprehensive error checking, especially when dynamically constructing range strings based on user input or data calculations.

Furthermore, although **Union** is generally very fast for typical Excel tasks, performance optimization may become a concern when attempting to combine hundreds or thousands of extremely small, highly non-contiguous ranges. In such extreme, fragmented scenarios, developers might consider alternative approaches. For example, collecting all necessary range addresses into a single, comma-delimited string and then using the Range object's direct constructor (e.g., `Range("A1, C1, E1, G1")`) can sometimes offer marginal speed improvements. However, this string-based approach often sacrifices the programmatic flexibility and object-oriented cleanliness offered by `Application.Union`.

Finally, it is essential to clearly distinguish the behavior of the **Union** method from its counterpart, the Intersect method. While **Union** serves to combine multiple ranges into a larger whole, **Intersect** performs the opposite function: it returns a **Range** object representing only the area where two or more specified ranges physically overlap. Both methods are vital components of the **Application** object model, and mastering both allows for precise, sophisticated control over complex worksheet selections and data manipulation within professional Excel automation tasks.

Summary and Additional Resources

The **Union** method stands as an indispensable function within VBA development, enabling the efficient management and manipulation of multiple, non-contiguous areas of an Excel worksheet through a single, streamlined operation. By consolidating several Range objects into one unified object, developers are empowered to write cleaner, more efficient, and demonstrably better-performing code, whether the goal is applying complex formulas, standardizing formatting, or transferring data between disparate locations.

To further complement your expertise in Excel automation, the following resources explain how to perform other critical and common tasks in VBA, building upon the foundational knowledge gained regarding the powerful **Union** method: