How to Extract Substrings in Power BI: A Step-by-Step Guide

Authored by

stats writer

January 13, 2026

A substring in Power BI refers to a specific, contiguous portion of a larger text string. The capability to extract and isolate these segments is critical for data preparation, normalization, and dimensional modeling. This process can be executed using several powerful text manipulation functions available within DAX (Data Analysis Expressions).

While basic extraction may sometimes be handled in the Power Query Editor, using DAX calculated columns offers enhanced flexibility and integration within the data model. For instance, if you have a column containing combined data like "ProductID_LocationCode," the extraction functions allow you to reliably separate these two components into distinct, usable columns. Similarly, extracting the year from a text-formatted date field requires defining a precise starting position and character count, a task handled elegantly by DAX functions like MID, LEFT, and RIGHT.

Utilizing DAX for Text Manipulation

The primary method for extracting specific substrings in DAX involves using a combination of dedicated text functions. These functions allow the analyst to specify the exact boundaries for extraction, whether based on positional counting or the location of a specific delimiter character. Understanding these tools is foundational for anyone tasked with cleaning or transforming text-heavy datasets in Power BI.

The most common functions employed for fixed-length extraction are **LEFT**, **MID**, and **RIGHT**. For dynamic extraction, where the length of the desired substring varies, these functions are often nested with utility functions like **SEARCH** and **LEN** (Length) to calculate the required character count dynamically. Below is a detailed breakdown of the standard formulas used to achieve specific extraction goals.

DAX Formula 1: Extracting from the Start (LEFT Function)

The LEFT function is designed for scenarios where the required segment is located at the absolute beginning of the text string. It requires two inputs: the column containing the source text and the number of characters you wish to retrieve, starting from the first character.

This formula extracts the first three characters at the start of the string in the **Email** column of the **my_data** table. It is ideal for isolating consistent prefixes.

first_three = LEFT('my_data', 3)

DAX Formula 2: Extracting from the Middle (MID Function)

When the target segment is embedded within the string, the MID function is utilized. This function

is more complex as it requires three parameters: the source column, the starting position (using 1-based indexing), and the exact number of characters to extract from that starting point.

This formula extracts 4 characters, commencing the count from the 2nd position within the string in the **Email** column. This technique is highly effective for retrieving fixed-length codes or IDs located centrally within a longer text field.

mid = MID('my_data', 2, 4)

DAX Formula 3: Extracting from the End (RIGHT Function)

The RIGHT function is the counterpart to LEFT, designed for extracting a specified number of characters counting backward from the end of the string. Like the LEFT function, it requires only two arguments: the source column and the number of trailing characters needed.

This formula extracts the last three characters located at the end of the string in the **Email** column. This is commonly used to isolate file extensions, country codes, or other consistent suffixes.

last_three = RIGHT('my_data', 3)

DAX Formula 4: Extract Substring Before a Delimiter

For variable-length strings, extraction often depends on locating a delimiter (like "@", "-", or "_"). To extract the text before a delimiter, we nest the **SEARCH** function within the LEFT function. SEARCH finds the position of the delimiter, and by subtracting 1 from this position, we calculate the exact length of the preceding text.

This particular formula extracts all of the text in the **Email** column before the @ symbol, successfully isolating the username portion, irrespective of its length.

text_before = LEFT('my_data', SEARCH("@", 'my_data', ,LEN('my_data')+1)-1)

DAX Formula 5: Extract Substring After a Delimiter

Extracting text after a delimiter requires a more intricate formula combining RIGHT, **LEN**, and **SEARCH**. The logic involves first finding the delimiter's position (SEARCH) and then calculating the total number of characters in the string (LEN). Subtracting the delimiter's position from the total length yields the exact number of characters that follow the delimiter, which is then passed to the RIGHT function.

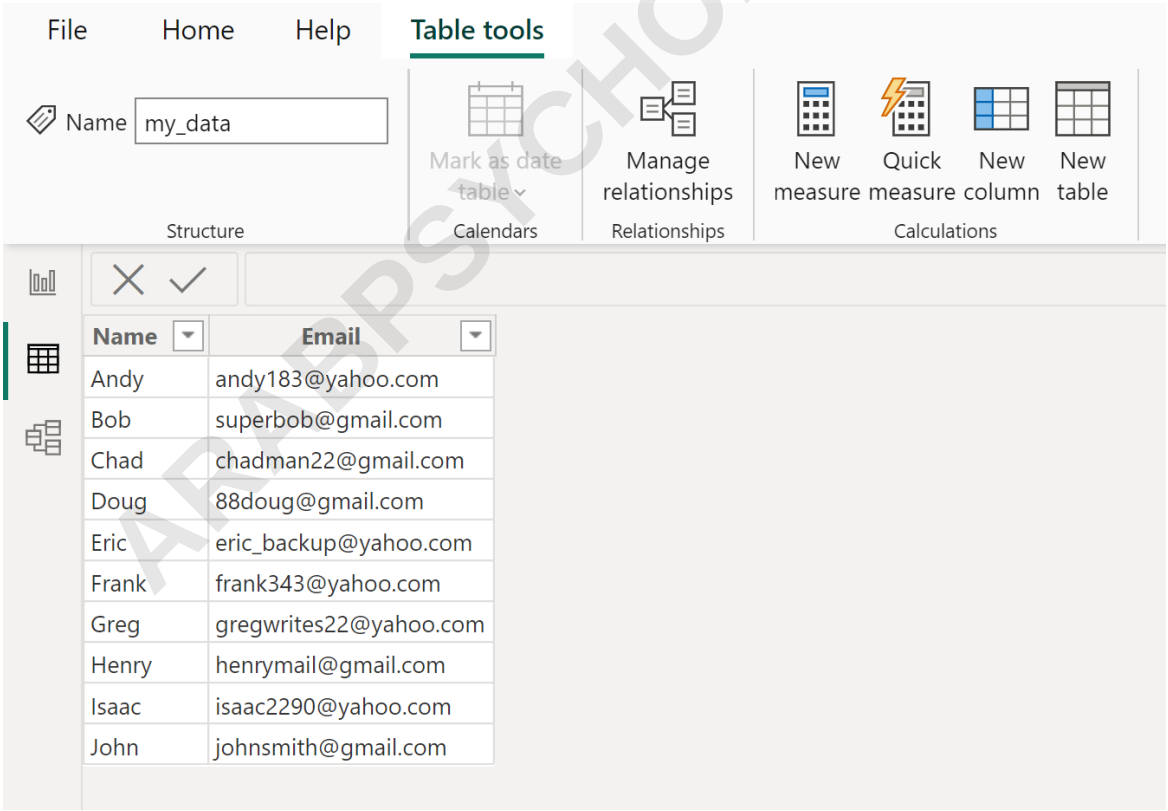
This particular formula extracts all of the text in the **Email** column after the @ symbol, effectively isolating the domain name. This dynamic calculation ensures accuracy across varied email address lengths.

text_after = RIGHT('my_data', LEN('my_data') - SEARCH("@",'my_data',,0))

Practical Implementation with Sample Data

The following practical examples demonstrate how to apply each of the DAX formulas detailed above using a sample table. We will utilize a table named **my_data** in Power BI, which contains an **Email** column serving as our source data. This step-by-step guidance illustrates the required actions within the Power BI interface to create the new calculated columns.

For each example, the process involves navigating to the Data View, selecting the **Table tools** tab, and clicking the **New column** icon to open the DAX formula bar. The results confirm the precise extraction capabilities of the respective functions, whether dealing with fixed positions or dynamic delimiters.



Name	Email
Andy	andy183@yahoo.com
Bob	superbob@gmail.com
Chad	chadman22@gmail.com
Doug	88doug@gmail.com
Eric	eric_backup@yahoo.com
Frank	frank343@yahoo.com
Greg	gregwrites22@yahoo.com
Henry	henrymail@gmail.com
Isaac	isaac2290@yahoo.com
John	johnsmith@gmail.com

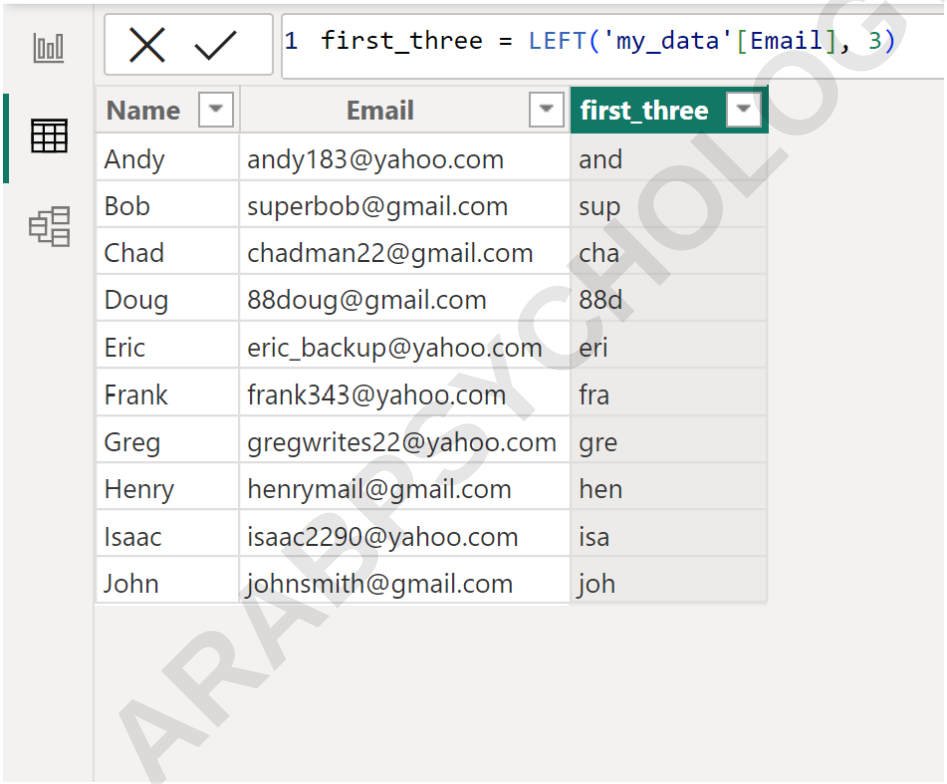
Example 1: Extract Substring from Start of String

To extract the first three characters from the **Email** column, we use the LEFT function, specifying the desired length as 3. This is the simplest form of substring extraction.

Begin by navigating to the **Table tools** tab, then click the **New column** icon, and finally, type in the following formula into the formula bar:

first_three = LEFT('my_data', 3)

This operation generates a new calculated column named **first_three**. This column successfully contains the first three characters derived from each row's entry in the source **Email** column, demonstrating fixed-length extraction from the left.



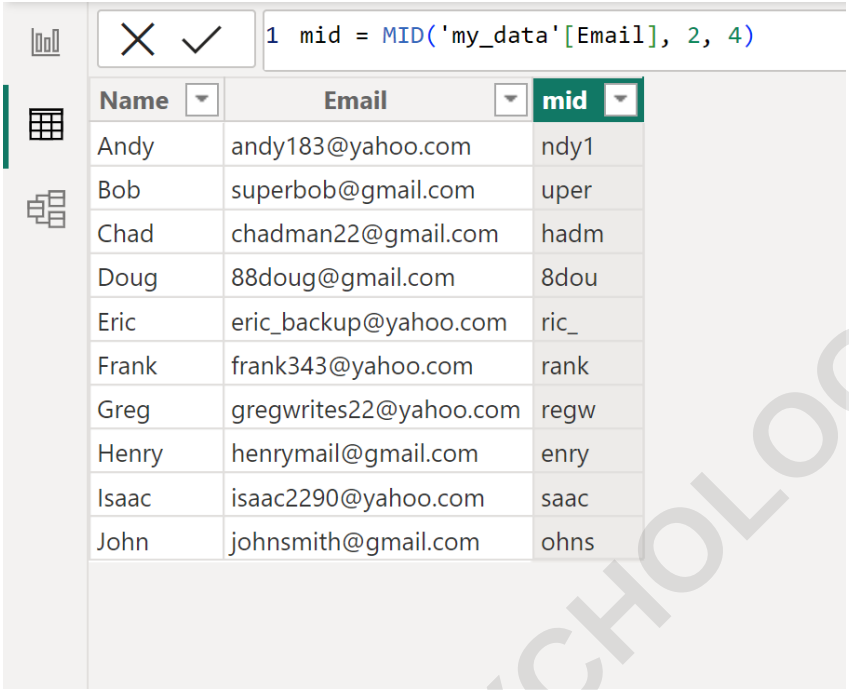
Example 2: Extract Substring from Middle of String

To demonstrate middle extraction, we will isolate 4 characters starting from the 2nd position of the string in the **Email** column. This process utilizes the positional control offered by the MID function.

Access the **Table tools** tab, click the **New column** icon, and input the following formula into the formula bar:

mid = MID('my_data', 2, 4)

The execution of this formula creates a new column labeled **mid**. This column accurately displays the middle 4 characters, commencing extraction precisely at position 2, as defined in the MID function arguments.



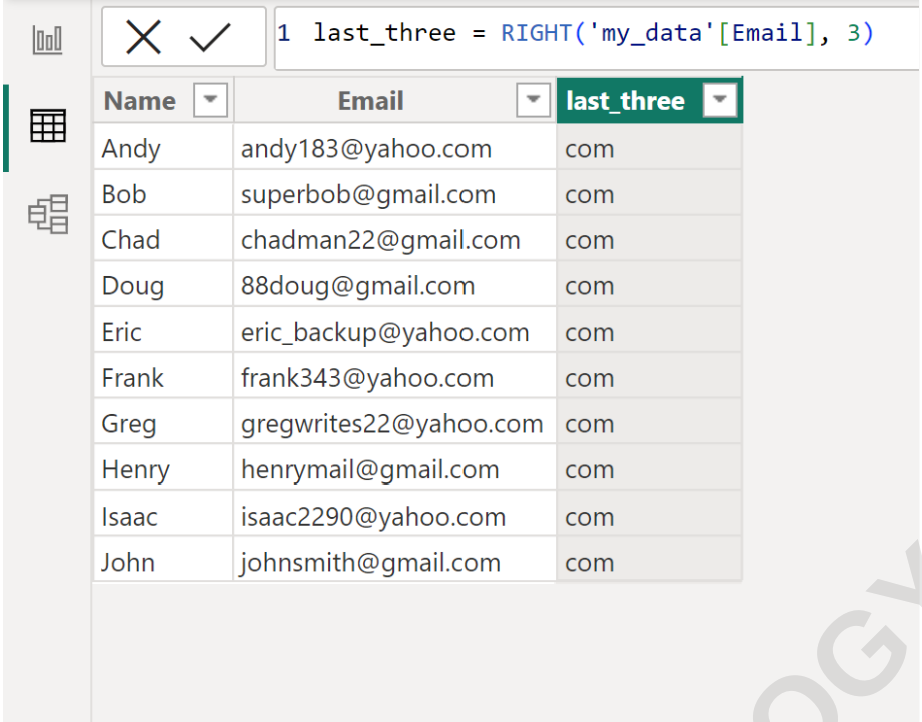
Example 3: Extract Substring from End of String

This example confirms the fixed-length extraction from the right side of the string. We will use the RIGHT function to retrieve the last three characters from the **Email** column.

Proceed to the **Table tools** tab, click the **New column** icon, and enter the subsequent formula into the formula bar:

last_three = RIGHT('my_data', 3)

This action results in a new column named **last_three**. This column contains the last three characters extracted from each string in the **Email** column, verifying the functionality for trailing segment extraction.



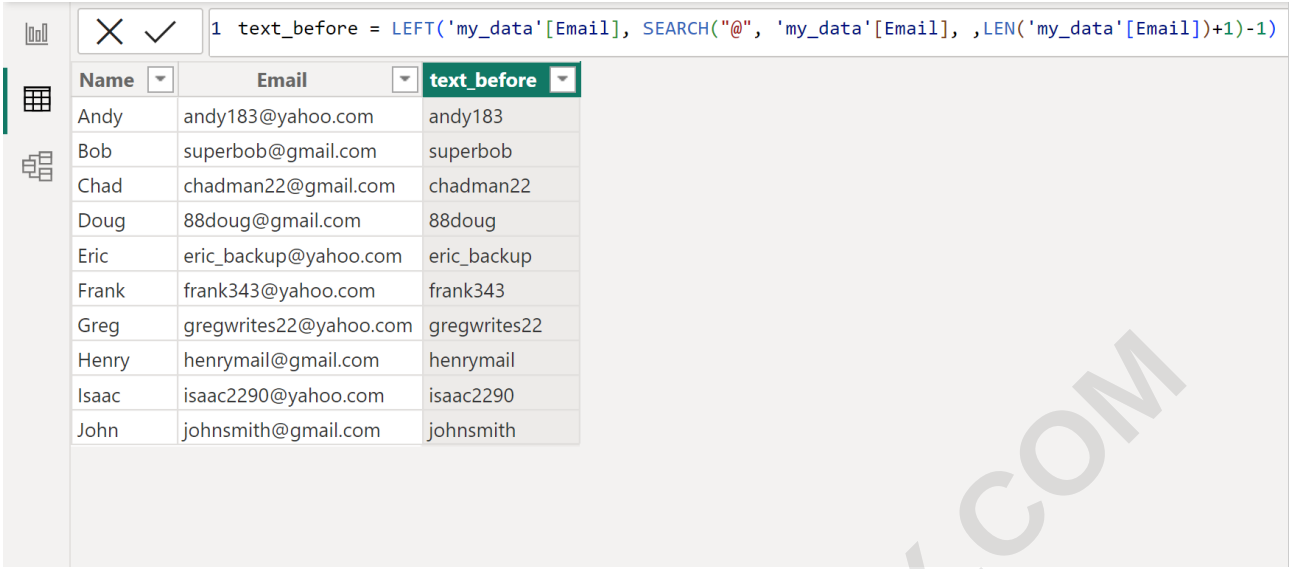
Example 4: Dynamic Extraction Before Delimiter

We now move to dynamic extraction: isolating all text preceding the @ symbol in the **Email** column. This technique is vital because the length of the username varies across rows.

Click the **Table tools** tab, then click the **New column** icon, and type in the following nested DAX formula into the formula bar:

text_before = LEFT('my_data', SEARCH("@", 'my_data', ,LEN('my_data')+1)-1)

This complex calculation creates a new column named **text_before**. This column contains all text that precedes the @ symbol, successfully demonstrating dynamic length handling based on delimiter position.



Name	Email	text_before
Andy	andy183@yahoo.com	andy183
Bob	superbob@gmail.com	superbob
Chad	chadman22@gmail.com	chadman22
Doug	88doug@gmail.com	88doug
Eric	eric_backup@yahoo.com	eric_backup
Frank	frank343@yahoo.com	frank343
Greg	gregwrites22@yahoo.com	gregwrites22
Henry	henrymail@gmail.com	henrymail
Isaac	isaac2290@yahoo.com	isaac2290
John	johnsmith@gmail.com	johnsmith

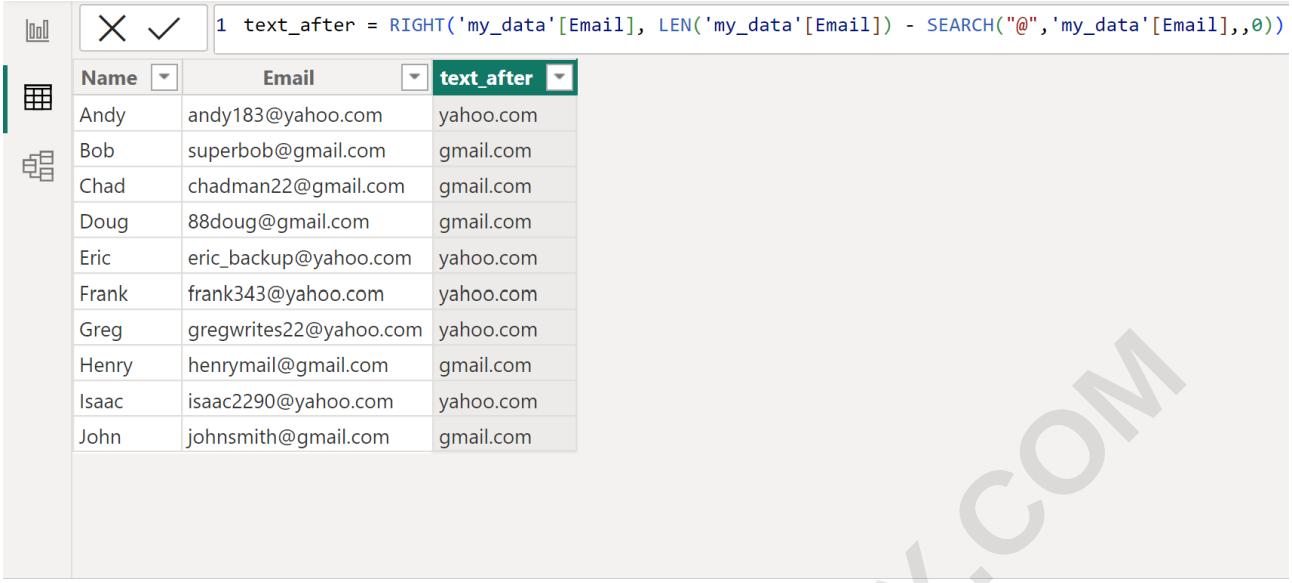
Example 5: Dynamic Extraction After Delimiter

Our final example isolates the domain name--all text following the @ symbol. This requires calculating the remaining length of the string after the delimiter has been located, utilizing the combination of RIGHT, LEN, and SEARCH.

Follow the standard steps: click the **Table tools** tab, select **New column**, and input the precise formula below:

text_after = RIGHT('my_data', LEN('my_data') - SEARCH("@",'my_data',,0))

The resulting column is named **text_after**. It contains the domain name (text after the @ symbol) for all entries in the **Email** column, confirming the success of complex, dynamic substring extraction using DAX.



Name	Email	text_after
Andy	andy183@yahoo.com	yahoo.com
Bob	superbob@gmail.com	gmail.com
Chad	chadman22@gmail.com	gmail.com
Doug	88doug@gmail.com	gmail.com
Eric	eric_backup@yahoo.com	yahoo.com
Frank	frank343@yahoo.com	yahoo.com
Greg	gregwrites22@yahoo.com	yahoo.com
Henry	henrymail@gmail.com	gmail.com
Isaac	isaac2290@yahoo.com	yahoo.com
John	johnsmith@gmail.com	gmail.com

By mastering the combination of positional functions (LEFT, MID, RIGHT) and utility functions (SEARCH, LEN), analysts can efficiently transform raw, unstructured text data into clean, segregated fields ready for deep analysis and visualization in Power BI.

The following tutorials explain how to perform other common tasks in Power BI: