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

Authored by

**stats writer**

January 29, 2026

Welcome to this detailed guide on text manipulation within Power BI. When working with complex textual data, the ability to isolate specific portions--known as a substring--is essential for accurate reporting and modeling. Although many programming languages feature a generic SUBSTRING function, in Power BI, we primarily rely on a suite of specialized DAX functions, specifically **LEFT**, **RIGHT**, and **MID**, to achieve precise text extraction based on position and length parameters. Mastering these techniques is fundamental for any serious data analyst utilizing the platform for intricate **data analysis** and transformation.

The process of extracting a substring involves carefully defining the starting point and the required length of the character sequence you wish to isolate. This capability is critical in numerous real-world scenarios, such as standardizing messy datasets, cleaning up identifiers, or parsing delimited text fields. For instance, imagine a scenario where a single column contains concatenated customer identifiers, combining region codes, product types, and sequential numbers. By using DAX string functions, you can decompose this single text field into multiple, meaningful fields, thereby enhancing the analytical depth of your data model and improving the efficacy of your reports.

Consider a practical application: if you possess a column containing complete user emails (e.g., john.doe@company.com), and your goal is to extract only the username component, you would need a mechanism to identify a delimiter (like the "@" symbol) and extract all characters preceding it. The generic concept of a SUBSTRING operation, implemented using a combination of **LEFT** and **SEARCH** functions in DAX, allows for this dynamic extraction, regardless of the username's length. This flexibility and precision are what make Power BI's text functions powerful tools for data shaping.



## Advanced Substring Extraction Techniques in Power BI (DAX Implementation)

To perform advanced text manipulation and extraction within the Power BI environment, familiarity with the core string functions in DAX (Data Analysis Expressions) is essential. Unlike some SQL environments that rely solely on a single SUBSTRING command, DAX provides distinct, optimized functions tailored for different extraction needs: extracting from the start, middle, or end of a string, or using dynamic delimiters. Below, we detail the five most common formula patterns used to extract specific substrings from text columns.

### Formula 1: Extracting Substrings from the Start of a String using LEFT

The LEFT function is the simplest and most direct method for extracting a fixed number of characters starting from the very beginning of a text string. This is invaluable when dealing with standardized codes or prefixes where the desired information is always located at the leading edge

of the value. For instance, if every entry in a column begins with a two-character regional code, the **LEFT** function provides the cleanest way to isolate that code into a separate column for reporting or filtering purposes.

The syntax requires two parameters: the reference to the text column and the number of characters you wish to retrieve. It is crucial to ensure that the specified number of characters does not exceed the total length of the string, although Power BI handles this gracefully by returning the full string if the requested length is greater than the actual length. Always remember that **DAX** string indexing is 1-based, meaning counting starts at the first character (position 1).

This formula specifically demonstrates how to extract the first three characters from the values stored in the **Email** column of the table named **'my_data'**. This results in a new calculated column named **first_three**, ideal for quickly segmenting data based on initial identifiers.

**first_three = LEFT('my_data', 3)**

## Formula 2: Extracting Substrings from the Middle of a String using MID

When the required substring is located somewhere in the middle of a larger text block, the **MID** function becomes the tool of choice. The MID function requires three distinct arguments: the text column reference, the character position where the extraction should begin (the starting number), and the total number of characters to extract from that starting point (the length). This function facilitates sophisticated parsing of text fields that follow defined structural patterns.

Using **MID** requires precise knowledge of the starting position, as calculated from the left-most character (position 1). If the starting position is beyond the length of the text string, **MID** will return an empty string. Furthermore, if the requested length extends beyond the end of the string, **MID** will gracefully return all remaining characters from the specified starting point to the end of the string, preventing truncation errors.

In the example below, we are instructing Power BI to start counting from the second character (position 2) in the **Email** column and then extract the next four characters sequentially. This technique is often employed to isolate unique identifiers nested within a longer code sequence.

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

## Formula 3: Extracting Substrings from the End of a String using RIGHT

The **RIGHT** function serves as the mirror image of the **LEFT** function, specifically designed for extracting a fixed number of characters counting backwards from the end of the string. This is

particularly useful for separating suffix codes, file extensions, or version numbers that consistently appear at the terminus of a text field. Like **LEFT**, **RIGHT** only requires two parameters: the text column and the count of characters to retrieve.

Employing the **RIGHT** function significantly streamlines the process compared to calculating the full length of the string and then using **MID** or **LEFT** in conjunction with the length function. If the requested number of characters is larger than the entire string, the function returns the complete string, ensuring data integrity. This focus on efficiency is a cornerstone of effective DAX usage in Power BI.

The provided formula creates a new column called **last_three**, containing the final three characters for every entry in the **Email** column. This is a common requirement when processing standardized file names where the extension (e.g., '.pdf', '.csv') needs to be isolated for categorization or filtering in **data analysis**.

**last_three = RIGHT('my_data', 3)**

## Formula 4: Dynamic Extraction Before a Specific Delimiter using SEARCH and LEFT

Often, the required substring length is not fixed but depends on the location of a specific character, known as a delimiter. To extract text located before a delimiter, we must combine the **LEFT** function with the **SEARCH** function. The **SEARCH** function dynamically finds the exact position of the delimiter within the string.

The key mechanism here involves two steps: first, **SEARCH** determines the position of the target delimiter (in this case, the "@" symbol). Second, we subtract 1 from this position to ensure the delimiter itself is excluded from the final result. The resulting number is then passed to the **LEFT** function as the character count to extract. The use of **SEARCH** makes this formula robust, as it works correctly regardless of the length of the text preceding the delimiter.

Notice the use of **LEN('my_data')+1** within the **SEARCH** function's optional parameters. This handles cases where the "@" symbol might be missing, preventing the entire expression from returning an error and ensuring a successful return of the entire string if the delimiter is not found. This dynamic and error-resistant approach is essential for cleaning real-world datasets in Power BI.

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

## Formula 5: Dynamic Extraction After a Specific Delimiter using SEARCH, LEN, and RIGHT

Extracting text that follows a specific delimiter requires a more complex calculation involving the **RIGHT** function, paired with **LEN** and **SEARCH**. Since the **RIGHT** function requires the number of characters to extract from the end, we must first calculate how many characters exist *after* the delimiter.

This calculation is achieved by taking the total length of the string (using **LEN**) and subtracting the position of the delimiter (found using **SEARCH**). The result is the exact length of the trailing substring, excluding the delimiter itself. This resulting length is then fed into the **RIGHT** function, which retrieves the correct portion. This technique is frequently used to isolate domain names from email addresses or file paths from URLs during extensive **data analysis** projects.

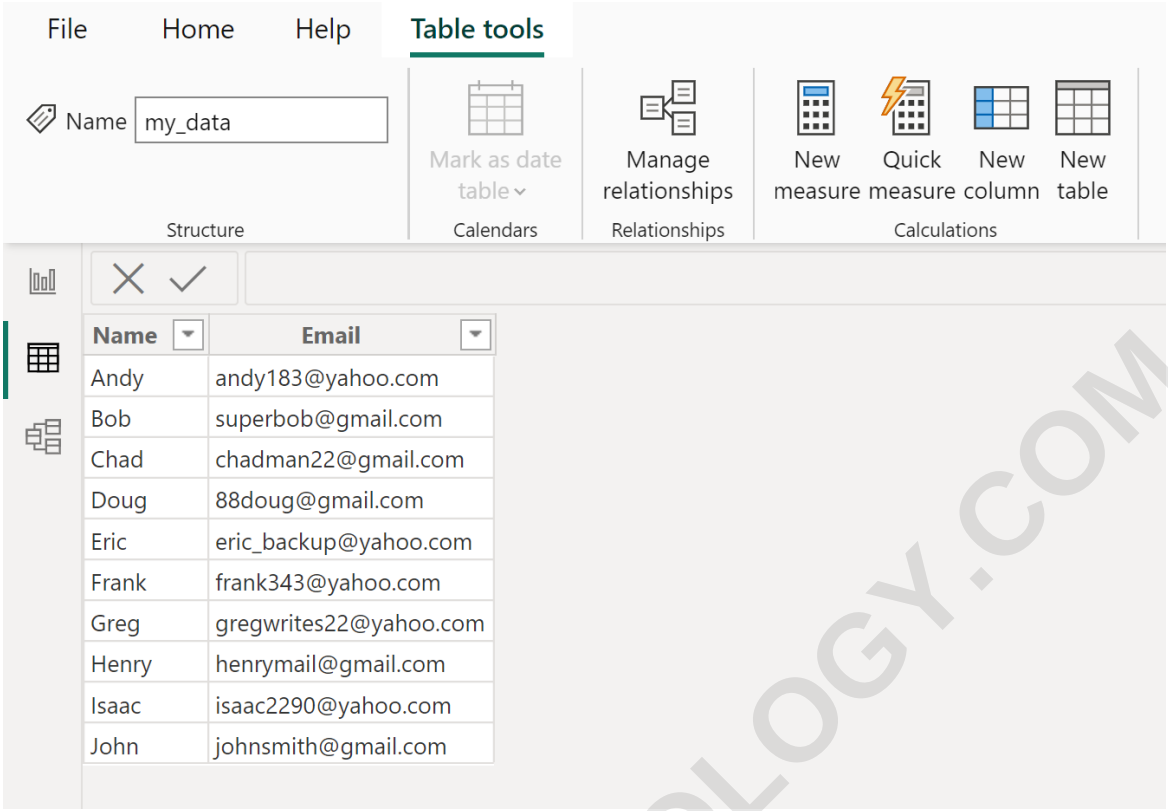
The formula below extracts everything found after the "@" symbol in the **Email** column. The parameter 0 in the **SEARCH** function ensures that if the delimiter is not found, the search returns 0, leading the calculation to return the full length of the string, preventing critical errors in the calculated column.

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

## Practical Implementation Examples in Power BI

To demonstrate these powerful DAX functions, we will apply each formula sequentially to a sample dataset named **my_data**. This table contains various email addresses, providing a realistic context for text parsing challenges. These examples illustrate the step-by-step process of creating calculated columns, which is the primary method for persistent data transformation within Power BI Desktop.

The foundational dataset we will be working with throughout these examples is displayed below. Observe the different lengths and structures of the email addresses, which validate the need for flexible, dynamic DAX solutions rather than simple fixed-length extractions.




## Example 1: Utilizing the LEFT Function for Fixed-Length Extraction

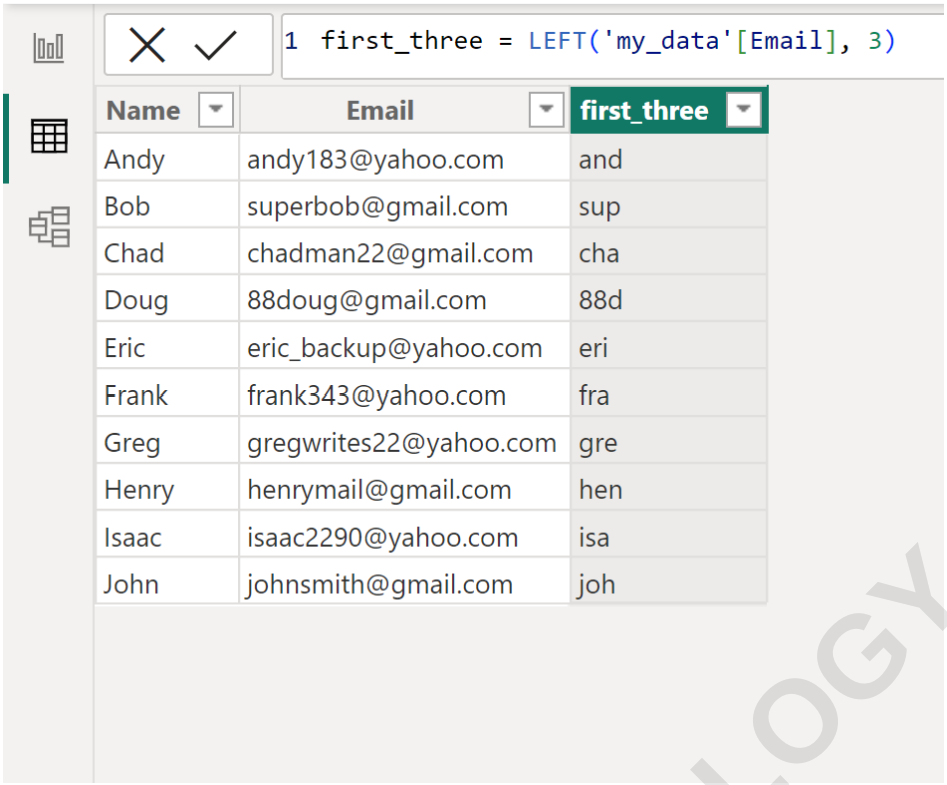
This first example focuses on extracting a predefined number of characters from the start of the email string. This might represent an initial, simplified user ID or a standardized prefix used for internal classification. To execute this, you navigate to the **Table tools** tab in Power BI Desktop, select the **New column** icon, and then input the required DAX expression into the formula bar.

The **LEFT** function efficiently processes each row, pulling only the first three characters. This is a fast and resource-light operation, making it suitable for large datasets where initial segmentation is needed. Remember, the core concept of the **LEFT** function is positional--it guarantees retrieval starting from character one, irrespective of any delimiters later in the string.

Applying the following concise formula generates the desired output:

**first_three = LEFT('my_data', 3)**

As visible in the resulting table, a new calculated column named **first_three** is appended to the data model. This column contains the initial three characters retrieved from every corresponding entry in the source **Email** column, providing immediate value for preliminary filtering or grouping based on these prefixes.




| Name | Email | first_three |
|---|---|---|
| Andy | andy183@yahoo.com | and |
| Bob | superbob@gmail.com | sup |
| Chad | chadman22@gmail.com | cha |
| Doug | 88doug@gmail.com | 88d |
| Eric | eric_backup@yahoo.com | eri |
| Frank | frank343@yahoo.com | fra |
| Greg | gregwrites22@yahoo.com | gre |
| Henry | henrymail@gmail.com | hen |
| Isaac | isaac2290@yahoo.com | isa |
| John | johnsmith@gmail.com | joh |



## Example 2: Utilizing the MID Function for Mid-String Segmentation

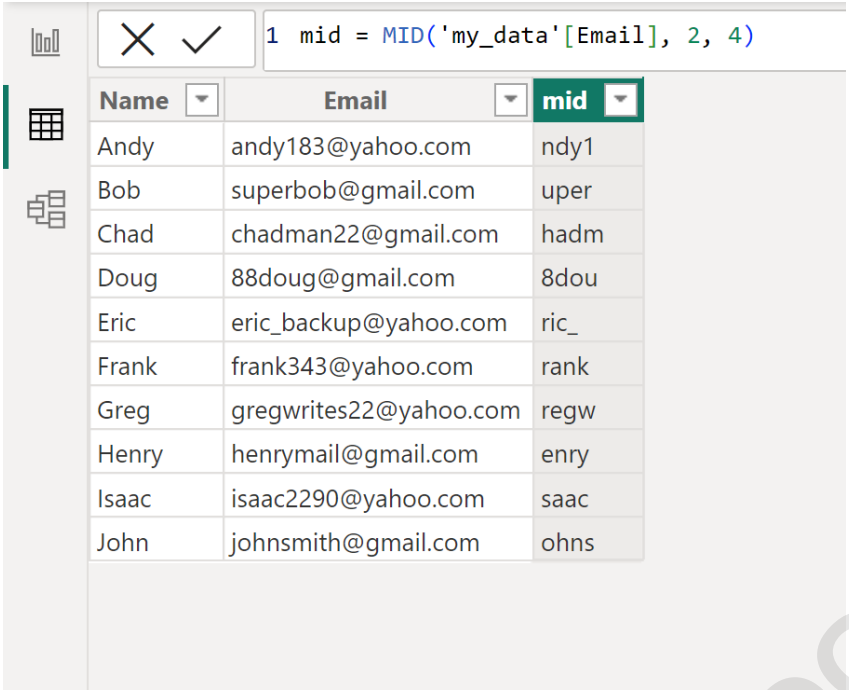
When focusing on internal codes or unique identifiers that are structurally embedded within the string, the MID function is indispensable. This example demonstrates how to precisely extract a segment that begins at an intermediate position and spans a fixed length. Follow the standard procedure: access the **Table tools** tab, click **New column**, and paste the formula into the bar.

By specifying a starting position of 2 and a length of 4, we are skipping the first character and extracting the subsequent four characters. This presupposes a consistent structure where the relevant information is always located in this specific window. If the structure of the data varies widely, consider using a combination of **MID** and **SEARCH** (similar to Formulas 4 and 5) for a more robust extraction.

The necessary MID function syntax is:

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

Upon execution, a new column named **mid** is generated. This column clearly displays the four-character segment starting from the second position, showcasing how critical details can be isolated even when they are not at the boundaries of the string. This type of extraction is critical for normalizing data fields for subsequent joins or lookups.




## Example 3: Utilizing the RIGHT Function for Fixed-Length Suffixes

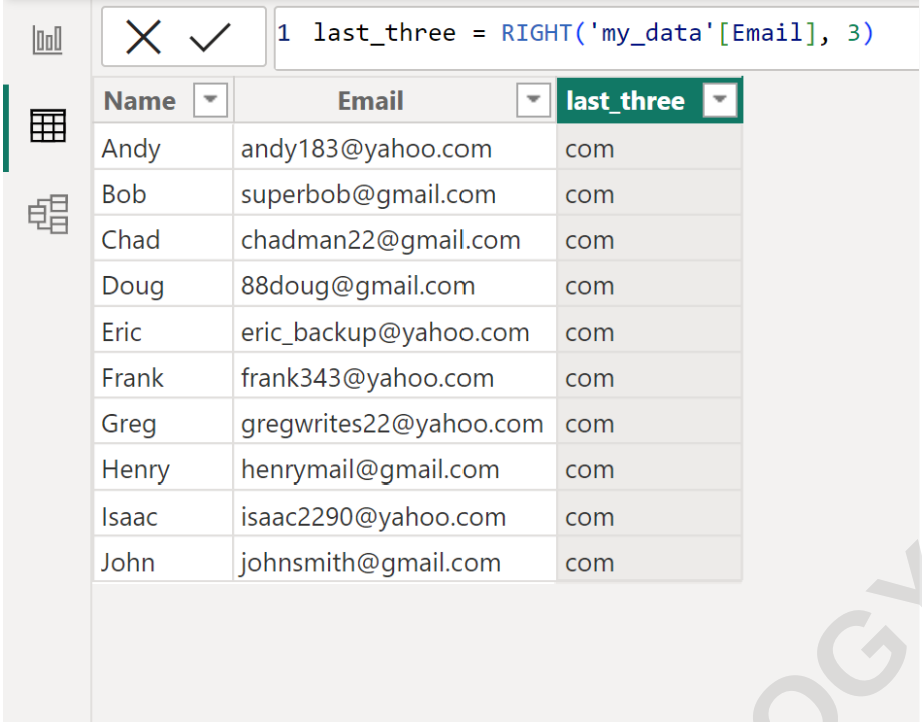
The **RIGHT** function is optimized for extracting trailing information, such as domain suffixes or file type extensions. Assuming a requirement to extract the last three characters from the **Email** column, we again initiate the creation of a new column via the **Table tools** tab in Power BI Desktop.

This approach is highly efficient for data cleansing where suffixes are important for categorization. Unlike the **MID** function, which requires determining an offset from the start, **RIGHT** inherently calculates the offset from the end, simplifying the required DAX formula significantly.

The concise formula implementation is as follows:

**last_three = RIGHT('my_data', 3)**

The resulting column, **last_three**, holds the last three characters of each email address. This is a straightforward method for obtaining standardized trailing data, enabling quick comparisons or aggregations based on these ending segments, forming a vital part of effective **data analysis**.




## Example 4: Dynamic Extraction of Username Before the '@' Delimiter

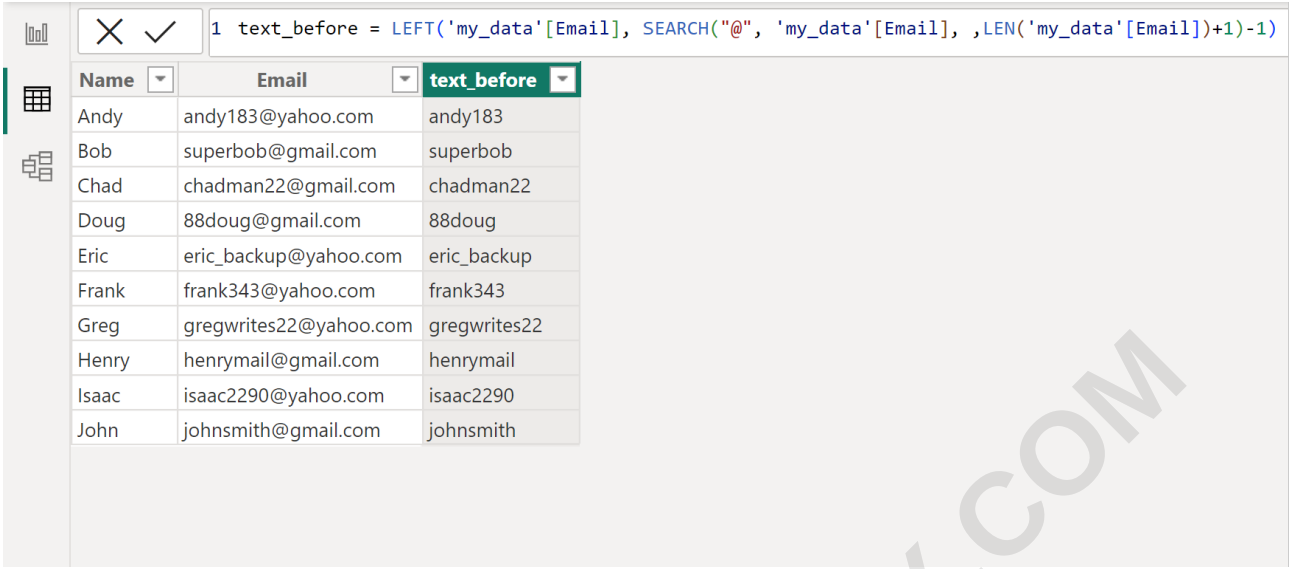
Handling strings where the length varies necessitates a dynamic approach using delimiters. This example demonstrates how to isolate the username portion of an email address--the text preceding the "@" symbol--using the powerful combination of **LEFT** and **SEARCH**. Begin by creating a new column as previously instructed.

The formula calculates the position of the "@" symbol and then extracts all characters to the left of that position, minus one character (to exclude the delimiter itself). This ensures that usernames of any length are extracted correctly, providing robust text parsing capabilities in Power BI. This is a standard procedure for separating identifiers from their context.

The full formula, optimized for error handling, is:

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

The newly generated column, **text_before**, contains only the username for each email entry. This separation of identifiers is foundational for creating relationships between data tables or performing user-specific aggregations in advanced **data analysis** models.




| 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 of Domain After the '@' Delimiter

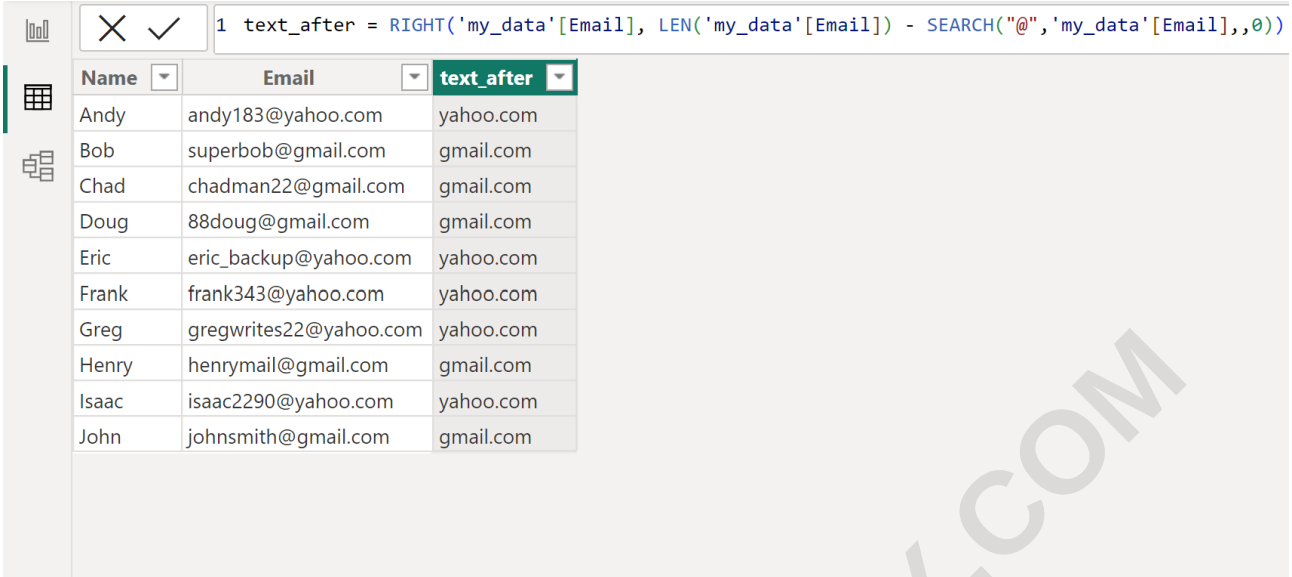
The final crucial technique is isolating the portion of the string that follows a variable delimiter, which, in the case of email addresses, is the domain name. We rely on the **RIGHT**, **LEN**, and **SEARCH** functions working in concert to calculate the required length dynamically. As always, create a new calculated column in the **Table tools** tab.

By determining the total length of the string and subtracting the character position of the "@" symbol, we accurately deduce the length of the domain name. This calculated length is then passed to the **RIGHT** function, ensuring that the domain and its suffix are perfectly isolated, regardless of how long the domain name happens to be.

Implement the following comprehensive DAX expression:

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

The output is the column **text_after**, which contains the extracted domain names. This ability to dynamically parse text using DAX formulas is essential for tasks like grouping customers by email domain or identifying corporate versus personal email addresses, offering high levels of refinement in your segmentation strategies.




## Conclusion and Further Power BI Resources

Mastering the art of substring extraction in Power BI, primarily through the use of **LEFT**, **RIGHT**, **MID**, and **SEARCH** functions in DAX, is non-negotiable for effective data preparation and **data analysis**. These techniques move beyond simple data viewing, allowing analysts to structurally reshape textual data into highly organized and measurable components. Whether you require fixed-length extractions or dynamic parsing based on delimiters, the DAX language provides all the tools necessary for sophisticated text handling.

The examples provided demonstrate how to transition from conceptual understanding to practical implementation, resulting in clean, manageable calculated columns that significantly enhance the utility of your raw data. Utilizing these extracted substrings allows for more precise filtering, grouping, and visualization, ultimately leading to richer business insights. Continue exploring the extensive capabilities of Power BI to unlock its full potential in your daily reporting workflows.

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