Use Wildcard Characters in Google Sheets Query

Authored by

Mohammed looti

November 2, 2025

The QUERY function stands as one of the most robust and indispensable tools within Google Sheets for serious data analysts. This powerful function leverages a specialized dialect of SQL (Structured Query Language) to execute intricate data operations, including filtering, aggregation, and sorting. However, when working with textual data, analysts frequently need to search for patterns rather than exact matches. This is where the concept of pattern matching becomes vital. To facilitate flexible text searches, the percent sign (**%**) is implemented as a crucial wildcard character in Google Sheets queries, enabling powerful partial string matching. This guide will detail how to harness this feature to significantly enhance your data querying capabilities.

Quick Reference: Essential Wildcard Methods

The **%** symbol, when used in conjunction with the **LIKE** operator, is the foundation for pattern matching within the QUERY language. The following formulas serve as a quick reference, illustrating the three fundamental ways you can position the wildcard to achieve specific types of string matches, determined by the location of the desired substring relative to the anchor text. Understanding these positions is key to efficient data retrieval.

Method 1: Return Cells That Start with Certain Characters (Prefix Match)

=QUERY(A1:A10, "Select A where A like 'hello%'")

Method 2: Return Cells That End with Certain Characters (Suffix Match)

=QUERY(A1:A10, "Select A where A like '%hello'")

Method 3: Return Cells That Contain Certain Characters (Substring Match)

=QUERY(A1:A10, "Select A where A like '%hello%'")

These three patterns--prefix, suffix, and internal substring matching--cover the vast majority of textual filtering needs. The subsequent sections will delve into the underlying mechanism of the **LIKE** operator and provide detailed, practical examples demonstrating the implementation and output of each method using a common dataset to illustrate how string matching is executed.

Understanding the QUERY Function and the LIKE Operator

To fully appreciate the utility of the wildcard, we must first confirm the role of the QUERY function. It fundamentally structures data analysis, requiring a data range (the source) and a query string (the commands). The query string uses clauses like `SELECT`, `WHERE`, and `ORDER BY`, mirroring traditional database languages. While powerful, the standard equality operator (`=`) within the `WHERE`

clause is limited; it demands an exact, character-for-character match of the search criteria against the cell content.

When exactness is too restrictive, especially with user-generated text or descriptions, we introduce the **LIKE** operator. The **LIKE** operator is purpose-built for performing flexible text comparisons based on specified patterns. It is intrinsically tied to the usage of wildcard characters. Unlike the strict comparison of the equality operator, **LIKE** evaluates whether a string matches a given pattern containing these flexible markers. This ability to search based on pattern rather than absolute value is essential for robust data filtering, allowing analysts to capture variations and partial matches seamlessly.

The percent sign (**%**) acts as the primary wildcard character in the Google Sheets QUERY environment. Crucially, the **%** represents zero or more characters of any type. This is what allows for variable-length matching, meaning it can stand in for a single character, dozens of characters, or no characters at all. Its placement relative to the search term dictates the type of match: anchoring the term at the start (prefix), anchoring it at the end (suffix), or allowing it to float freely (substring containment). Mastering the **LIKE** operator and the **%** wildcard is crucial for moving beyond basic filtering in Google Sheets.

Wildcard Application 1: Matching String Prefixes (Starts With)

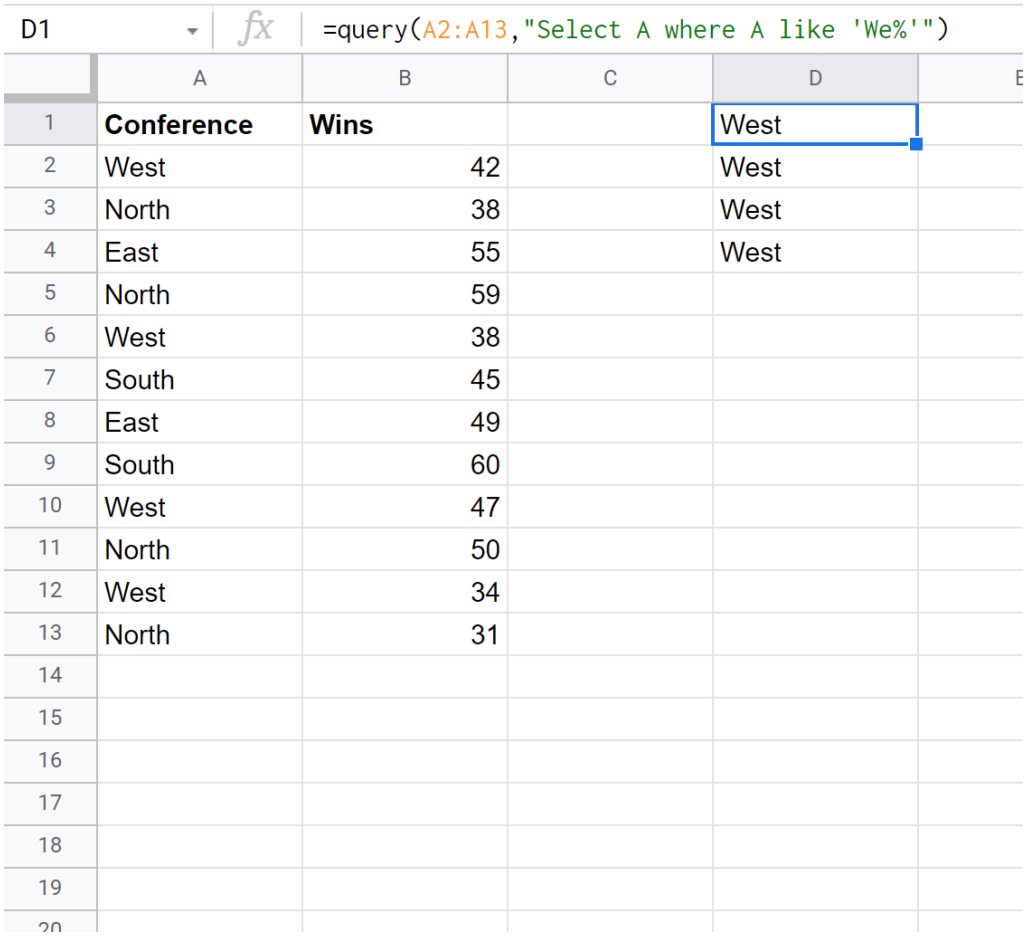
The first fundamental application of the **%** wildcard is **prefix matching**, which isolates records where the data begins with a specific sequence of characters. This technique is indispensable when categorizing data based on starting codes, standard departmental names, or product prefixes. To implement prefix matching, the search pattern must place the required starting text (the prefix) immediately followed by the percent sign. This anchors the search criteria firmly at the beginning of the string.

Consider a scenario where you are analyzing a list of text entries in column A and need to extract only those that start with 'We'. By using the pattern `'We%'`, we explicitly tell the **LIKE** operator that the string must begin with 'W' followed by 'e', and then the **%** allows for any subsequent characters (or no characters at all) to complete the string. This mechanism ensures that strings like "Wednesday," "Weasel," or even just "We" are included, while entries such as "Sweater" or "The Week" are accurately excluded from the results set.

The structure of the formula for this operation, targeting the range A1:A10, clearly illustrates this prefix criterion. The query selects entries from column A where the content of A matches the pattern anchored by 'We':

=QUERY(A1:A10, "Select A where A like 'We%'")

Notice how the string `'We%'` mandates that the cell content must start exactly with 'We', followed by any sequence of characters. The resulting output, visualized in the accompanying screenshot, confirms that only entries starting with the defined prefix 'We' are returned, demonstrating precise control over the initial characters of the data being filtered.



	A	B	C	D
1	**Conference**	**Wins**		West
2	West	42		West
3	North	38		West
4	East	55		West
5	North	59		
6	West	38		
7	South	45		
8	East	49		
9	South	60		
10	West	47		
11	North	50		
12	West	34		
13	North	31		
14				
15				
16				
17				
18				
19				

Wildcard Application 2: Matching String Suffixes (Ends With)

The opposite of prefix matching is **suffix matching**, a technique used to identify records that conclude with a specific character sequence. This is commonly applied when filtering file names based on extensions (e.g., '.pdf', '.docx') or when identifying products that share a particular concluding code. Achieving this type of filter requires placing the wildcard at the start of the pattern string, effectively reversing the anchoring logic used in the previous method.

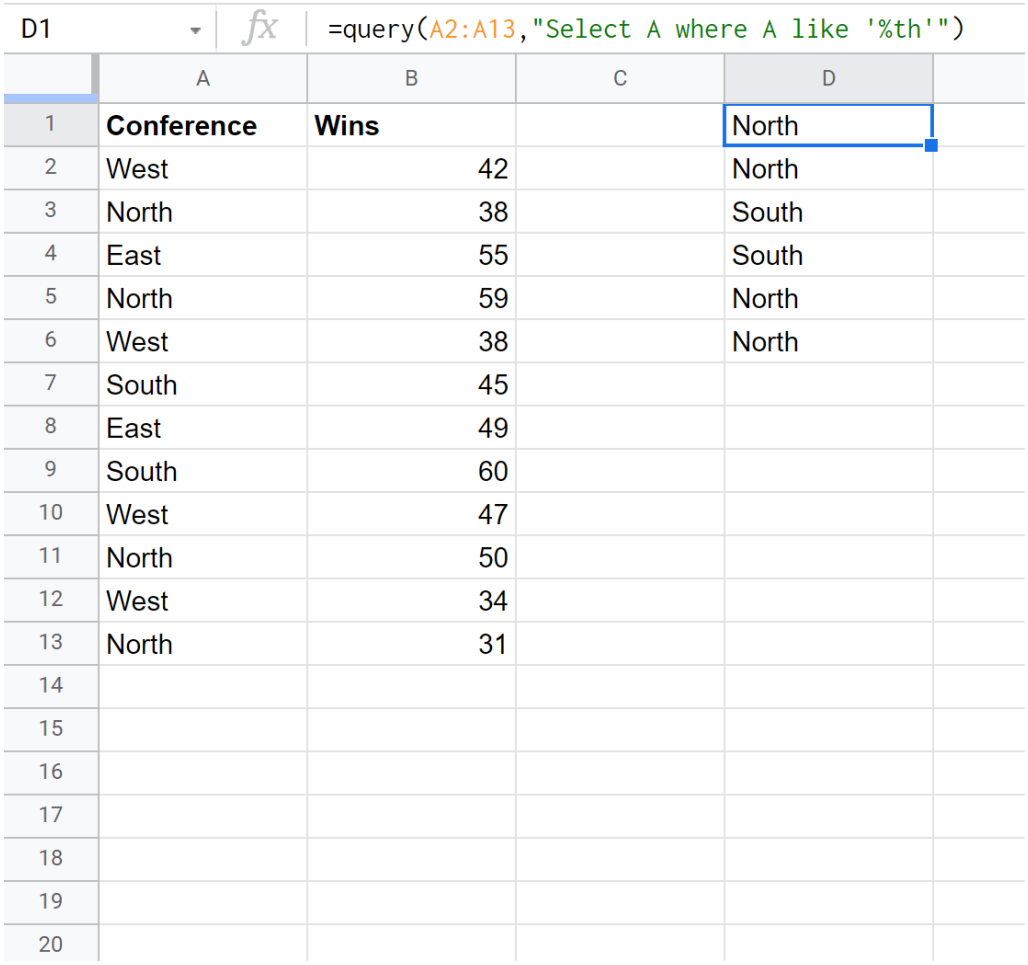
To implement suffix matching, the percent sign (**%**) precedes the target sequence. The **%** symbol acts as a placeholder, absorbing any characters that might appear at the beginning of the cell content. This ensures that the match criteria are only applied to the final characters of the string, regardless of how long the preceding text is. For example, if we want to find all entries in column A that end precisely with 'th', the pattern `'%th'` is used, ensuring the search is anchored at the

string's conclusion.

Using this method on the range A1:A10 to filter for the suffix 'th' provides the following formula structure. Note the critical placement of the wildcard at the beginning of the search term:

=QUERY(A1:A10, "Select A where A like '%th'")

In this setup, the pattern `'%th'` will successfully match strings such as "fourth," "smooth," and "depth" because the preceding characters are successfully matched by the wildcard. Critically, it will ignore strings where 'th' appears internally but not as the absolute ending sequence. The visual demonstration below confirms the effective application of this suffix-matching logic, highlighting its utility in targeted data retrieval based on string endings.



	A	B	C	D
1	**Conference**	**Wins**		North
2	West	42		North
3	North	38		South
4	East	55		South
5	North	59		North
6	West	38		North
7	South	45		
8	East	49		
9	South	60		
10	West	47		
11	North	50		
12	West	34		
13	North	31		
14				
15				
16				
17				
18				
19				
20				

Wildcard Application 3: Matching Internal Substrings (Contains)

Perhaps the most flexible and frequently used application of the percent wildcard character is **internal substring matching**, which allows you to search for a specific sequence of characters

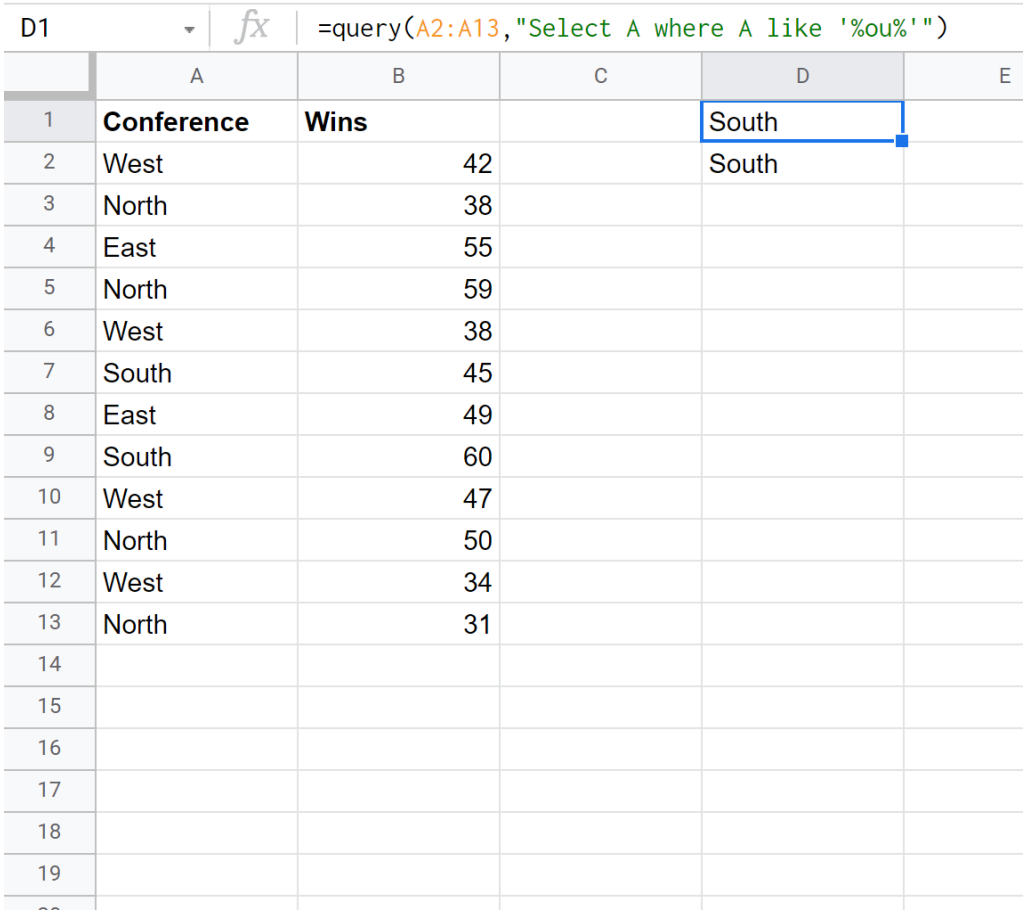
regardless of its position within the target cell. This is essential for full-text searching or when you are unsure of the exact placement of the required characters within a descriptive field, making the search pattern location-agnostic.

To achieve this "contains" logic, the target string must be sandwiched between two percent wildcards. The first **%** serves to match any characters preceding the desired sequence, and the second **%** accounts for any characters following it. This dual-wildcard approach renders the search highly permissive, ensuring the **LIKE** operator returns a match if the target substring exists anywhere within the cell content, including at the very start or end.

For example, to identify every cell in column A (A1:A10) that contains the sequence 'ou', we use the pattern `'%ou%'`. This construction ensures that strings like "cloud," "our," "mountain," and "four thousand" are all successfully captured, offering maximum flexibility in data filtering. The resulting formula is highly useful for generalized searching:

=QUERY(A1:A10, "Select A where A like '%ou%'")

This construction is highly useful for generalized searching, such as finding product descriptions that mention a specific material or locating names that contain a particular sequence. Review the accompanying image to see the effective filtering of cells based on this internal substring criterion, confirming that the search is entirely agnostic to the substring's position within the filtered cell.



	A	B	C	D	E
1	**Conference**	**Wins**		South	
2	West	42		South	
3	North	38			
4	East	55			
5	North	59			
6	West	38			
7	South	45			
8	East	49			
9	South	60			
10	West	47			
11	North	50			
12	West	34			
13	North	31			
14					
15					
16					
17					
18					
19					

Advanced Considerations for Wildcard Usage

While the percent sign provides excellent flexibility, advanced users must be aware of certain constraints and functionalities specific to the Google Visualization QUERY function when performing string comparisons. The most critical nuance is **case sensitivity**. By default, the **QUERY** language executes string comparisons, including those involving the **LIKE** operator, in a case-sensitive manner. This means that a search using `'we%'` will fail to match a string starting with "We" (capitalized).

If case-insensitivity is essential for your data analysis, a standard, highly effective workaround involves using the `lower()` function directly on the column reference within the query string. By converting both the data column and the search term to lowercase before the SQL **LIKE** comparison is executed, you ensure that "Wednesday" and "wednesday" are treated identically. For instance, the query might become: `"Select A where lower(A) like 'we%'"`. This technique guarantees reliable matching regardless of the source data capitalization, significantly broadening the scope of the search.

It is also vital to note the limitations regarding other traditional SQL wildcard features. Standard

SQL often utilizes the underscore (_) as a single-character wildcard (matching exactly one character). However, the Google Sheets QUERY language, based on the Google Visualization API, only officially supports the percent sign (**%**) for variable-length matching. Users requiring precise single-character matching, or highly complex pattern recognition beyond the scope of simple prefix, suffix, or containment searches, should consider using the dedicated Google Sheets function `REGEXMATCH` or integrating the `FILTER` function, which offers robust support for advanced regular expressions. This alternative path provides granular control over intricate patterns that the standard **LIKE** operator cannot handle.

Conclusion and Further Learning

The strategic application of the percent sign wildcard, in close partnership with the **LIKE** operator, elevates the QUERY function far beyond simple filtering capabilities. Mastering prefix matching, suffix matching, and internal substring containment provides the essential foundation for sophisticated textual analysis within Google Sheets. These three methods are powerful tools for dynamically manipulating data based on complex character patterns.

Consistent application and experimentation with these wildcard techniques will significantly enhance your ability to efficiently extract meaningful and precise insights from voluminous or unstructured datasets. For those ready to explore additional advanced filtering techniques and data aggregation methods, the following resources offer excellent pathways for continued learning and mastery of the Google Sheets environment.

Additional Resources

The following tutorials explain how to perform other common queries and advanced data manipulations in Google Sheets: