Learning to Filter Data in Google Sheets with the QUERY Function

Authored by

Mohammed looti

November 4, 2025

The ability to efficiently search and filter large datasets is fundamental to modern data analysis. In Google Sheets, the powerful QUERY function provides unparalleled flexibility, acting much like a lightweight version of SQL directly within your spreadsheet environment. This function is essential when you need to extract specific rows based on complex criteria, such as determining whether a cell contains a particular sequence of characters or a specific string.

Unlike simple, static filtering tools, the **QUERY** function allows for dynamic, formula-driven data extraction. This capability is especially vital for creating summary tables or interactive dashboards that automatically update whenever the source data changes. Learning how to select rows that contain a specific text fragment is arguably one of the most common and powerful operations performed using this feature, enabling users to quickly isolate relevant information from vast sheets.

The core mechanism for performing this operation relies on utilizing the **WHERE** clause in conjunction with the `CONTAINS` keyword. This powerful combination instructs Google Sheets to scan a specified column and return only those rows where the target column includes the specified text fragment. Mastering this specific syntax is the gateway to sophisticated and precise data manipulation within the Google Sheets platform, allowing for filtering that moves beyond exact cell matches.

Understanding the Core Syntax of String Containment

The QUERY function requires three main arguments: the **data range**, the **query string** (which is written in the Google Visualization Query Language), and an optional argument specifying the number of header rows. To effectively filter based on partial text content, we must focus heavily on constructing an accurate and precise query string that utilizes the appropriate operators.

The typical structure of the query begins with the `SELECT` clause, which defines the columns you wish to output, followed immediately by the WHERE clause, which sets the conditions for inclusion. When dealing with textual data, the `CONTAINS` operator is used exclusively within the WHERE clause to perform a partial match search. This is crucial because it means the text being sought does not have to match the entire cell content; it only needs to exist as a portion of the cell's value.

The general syntax for selecting rows where a column contains a specific string is demonstrated below. It is important to pay close attention to the nesting of quotation marks: the entire query string is enclosed in double quotes, while the specific text fragment being searched for (e.g., `'this'`) must be enclosed in single quotes.

=query(A1:C9, "select A, B where B contains 'this'", 1)

In this fundamental example, the query instructs the system to extract columns A and B from the

specified data range **A1:C9**. Crucially, it only includes rows where column B successfully `CONTAINS` the literal string 'this'. The trailing **1** argument is essential as it informs the QUERY function that the input range includes one header row, which should be treated separately during processing and excluded from the filtering logic.

It is paramount to remember a key operational detail: the Google Sheets `CONTAINS` operator is generally **case-sensitive** by default. This means that searching for 'apple' will return different results than searching for 'Apple'. This distinction requires careful planning, and we will discuss effective workarounds for achieving case-insensitive searching in a later section.

Dataset Overview for Practical Application

To provide a clear, practical illustration of the `CONTAINS` keyword in action, we will utilize a sample dataset formatted as a standard table. This data includes information typically found in sports analytics, featuring columns for Player Name, Team, and Points Scored. The ability to filter this data dynamically based on partial team names or player characteristics effectively showcases the immense utility of the **QUERY** function.

This representative dataset, spanning the range **A1:C9**, will serve as the reference point for all subsequent examples. By tracking how changes in the query syntax impact the extracted results, we can clearly demonstrate the filtering mechanisms and precision of the `CONTAINS` operator.

	A	B	C	D	E
1	**Player**	**Team**	**Points**		
2	Andy	Lakers	13.4		
3	Bob	Mavericks	7.8		
4	Carl	Spurs	13.7		
5	Dave	Warriors	22.3		
6	Eric	Mavericks	27.8		
7	Fred	Mavericks	20.8		
8	George	Spurs	12.7		
9	Harold	Lakers	8.2		
10	Isaiah	Warriors	12.5		
11	Joe	Warriors	30.2		
12	Ken	Spurs	22.4		
13					
14					
15					
16					
17					
18					
19					
20					
21					
22					

As you can observe, the data range includes several different teams. Our primary goal in the upcoming examples will be to isolate specific records based on partial text matches contained within the 'Team' column (Column B).

Example 1: Selecting Rows that Contain a Specific String

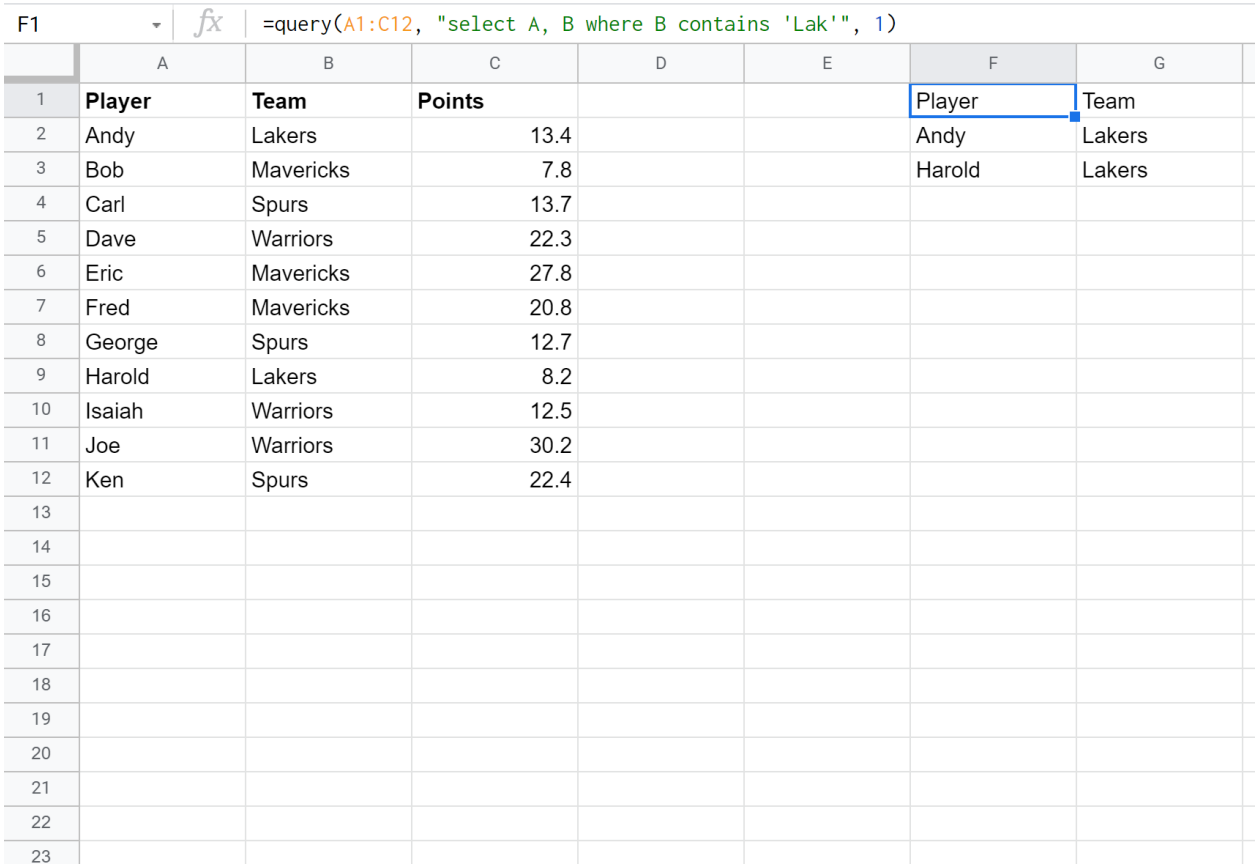
Our first practical demonstration focuses on isolating all records associated with a specific team using only a partial text match. Suppose the objective is to retrieve all rows connected to the Los Angeles Lakers. Instead of filtering for the full, exact name, we can search for a unique and defining substring, such as 'Lak'.

This method is highly efficient when dealing with inconsistent naming conventions, or when the user only knows a fragment of the required entry. By targeting the string '**Lak**', we ensure that both the full team name 'Lakers' and any potential misspellings or variations containing that specific sequence are captured, provided they exist within the source data.

The required query syntax is structured to target the 'Team' column, which corresponds to Column B within the designated data range **A1:C9**:

=QUERY(A1:C9, "select A, B, C where B contains 'Lak'", 1)

Executing this query against our sample data successfully applies the containment filter. The output returns only those records where the text in Column B (Team) includes the substring 'Lak', demonstrating a powerful and selective filtering operation based on partial string matching.



	A	B	C	D	E	F	G
1	**Player**	**Team**	**Points**			Player	Team
2	Andy	Lakers	13.4			Andy	Lakers
3	Bob	Mavericks	7.8			Harold	Lakers
4	Carl	Spurs	13.7				
5	Dave	Warriors	22.3				
6	Eric	Mavericks	27.8				
7	Fred	Mavericks	20.8				
8	George	Spurs	12.7				
9	Harold	Lakers	8.2				
10	Isaiah	Warriors	12.5				
11	Joe	Warriors	30.2				
12	Ken	Spurs	22.4				
13							
14							
15							
16							
17							
18							
19							
20							
21							
22							
23							

As illustrated by the output, this query returns precisely the two rows corresponding to the Lakers team, effectively filtering out all other entries based solely on the presence of the substring 'Lak' in the designated column. This confirms the functionality and precision of the `CONTAINS` operator for inclusion filtering.

Example 2: Selecting Rows that Do Not Contain a Specific String

Filtering by exclusion is just as valuable and often necessary as filtering by inclusion. There are many analytical scenarios where the objective is to analyze data pertaining to every entry **except** those containing a certain string. This is easily achieved by introducing the logical operator `NOT` directly before the `CONTAINS` keyword within the WHERE clause.

To select all rows where the Team column *does not* contain the string '**Lak**', we simply prepend

`NOT` to the condition established in the previous example. This logical negation reverses the matching logic, ensuring that any row where the specified substring is found is immediately discarded from the results set, thereby isolating the remainder of the data.

The syntax for exclusion filtering is as follows, clearly utilizing the `NOT CONTAINS` pair to define the negative condition:

=QUERY(A1:C9, "select A, B, C where B not contains 'Lak'", 1)

This type of query is a fundamental tool for isolating residual data or focusing analysis on non-conforming entries. For instance, if you were analyzing performance metrics across all teams except one dominant outlier (like the Lakers in this case), using `NOT CONTAINS` allows you to quickly segment the data pool for focused study.

F1 | =query(A1:C12, "select A, B where not B contains'Lak'", 1)

	A	B	C	D	E	F	G
1	**Player**	**Team**	**Points**			Player	Team
2	Andy	Lakers	13.4			Bob	Mavericks
3	Bob	Mavericks	7.8			Carl	Spurs
4	Carl	Spurs	13.7			Dave	Warriors
5	Dave	Warriors	22.3			Eric	Mavericks
6	Eric	Mavericks	27.8			Fred	Mavericks
7	Fred	Mavericks	20.8			George	Spurs
8	George	Spurs	12.7			Isaiah	Warriors
9	Harold	Lakers	8.2			Joe	Warriors
10	Isaiah	Warriors	12.5			Ken	Spurs
11	Joe	Warriors	30.2				
12	Ken	Spurs	22.4				
13							
14							
15							
16							
17							
18							
19							
20							
21							
22							

The resulting table demonstrates that every row where the Team column did not contain 'Lak' was successfully returned. This confirms that `NOT CONTAINS` provides a robust and reliable method for exclusion filtering, returning all records except those associated with the Lakers.

Example 3: Selecting Rows that Contain One of Several Strings (Using OR)

Frequently, filtering requirements involve satisfying multiple potential criteria simultaneously. If we need to select rows that contain either string A **or** string B, we must combine multiple `CONTAINS` clauses using the logical operator `OR`. This technique allows for complex pattern matching where the returned rows must satisfy at least one of the defined conditions to be included.

For example, if we wanted to view data for the Lakers and the Mavericks simultaneously, we would need the query to check if the Team column contains 'Lak' **OR** if it contains 'Mav'. The explicit use of `OR` ensures that a row is included in the final result if either condition evaluates to true, thus consolidating data from disparate categories.

The syntax for combining multiple containment searches using `OR` is structured as follows, requiring the full column identifier and the `CONTAINS` operator to be restated for each individual condition:

=QUERY(A1:C9, "select A, B, C where B contains 'Lak' or B contains 'Mav'", 1)

This method is highly scalable and incredibly flexible. You could chain together several `OR` conditions to include data from numerous specific teams or categories within a single query result set, making it an indispensable tool for building comprehensive, consolidated reports based on heterogeneous data points.

F1 =query(A1:C12, "select A, B where B contains 'Lak' or B contains 'Mav'", 1)

	A	B	C	D	E	F	G
1	**Player**	**Team**	**Points**			Player	Team
2	Andy	Lakers	13.4			Andy	Lakers
3	Bob	Mavericks	7.8			Bob	Mavericks
4	Carl	Spurs	13.7			Eric	Mavericks
5	Dave	Warriors	22.3			Fred	Mavericks
6	Eric	Mavericks	27.8			Harold	Lakers
7	Fred	Mavericks	20.8				
8	George	Spurs	12.7				
9	Harold	Lakers	8.2				
10	Isaiah	Warriors	12.5				
11	Joe	Warriors	30.2				
12	Ken	Spurs	22.4				
13							
14							
15							
16							
17							
18							
19							
20							
21							
22							

The resulting output successfully aggregates the data, returning only the rows where the team name equals Lakers or Mavericks. This clearly illustrates the effective use of the `OR` operator to satisfy multiple partial string matching criteria within a single powerful query.

Advanced Considerations and Best Practices

While the `CONTAINS` operator is robust and handles most basic needs, advanced users should be aware of specific operational details, particularly regarding inherent case sensitivity and achieving more complex pattern matching requirements.

Case Sensitivity Workarounds

As established earlier, the `CONTAINS` operator in the Google Sheets QUERY function is case-sensitive by default. If you search for 'lak' (lowercase), you will fail to match entries stored as 'Lakers' (capitalized), potentially leading to incomplete results. To effectively overcome this limitation and perform a case-insensitive search, you must convert both the column data and the search string to a consistent case (typically lowercase) before the comparison occurs.

This is achieved by applying the `LOWER()` function directly to the column identifier within the

WHERE clause of the query. For example, to search column B for 'lak' regardless of capitalization, the required syntax changes significantly:

=QUERY(A1:C9, "select A, B, C where lower(B) contains 'lak'", 1)

By applying `lower(B)`, the query temporarily converts all entries in column B to lowercase solely for the purpose of comparison, ensuring that the search for 'lak' captures 'LAKERS', 'lakers', and 'Lakers' equally, providing comprehensive results.

Alternative Matching Operators: MATCHES

While `CONTAINS` is perfectly suited for basic substring matching, the Google Visualization Query Language also offers the `MATCHES` operator. This alternative operator utilizes regular expressions (regex) for extremely flexible and complex pattern matching. If your filtering needs extend beyond simple containment--for instance, requiring a string to start with a certain letter, end with a specific character, or conform to a complex format (like a validated email address or phone number)--`MATCHES` is the superior tool for precision filtering.

For example, to select rows where the team column starts with 'L' and ends with 's', you would use `MATCHES` with the appropriate regex pattern (e.g., `'L.*s'`). While the syntax is more complex to write, `MATCHES` provides the ultimate level of control over intricate text filtering logic.

Performance Considerations

For small datasets, the performance difference between various filtering methods (such as the built-in Filter Views versus the **QUERY** function) is negligible. However, when dealing with thousands or even tens of thousands of rows, the efficiency of your query becomes critical for spreadsheet responsiveness. Using the QUERY function, particularly with well-defined `CONTAINS` criteria, is generally highly efficient because it leverages Google's robust backend processing power to execute the filtering logic quickly before returning the results to the sheet.

It is best practice to avoid highly complex, deeply chained `OR` statements if possible. In extremely large datasets, using temporary helper columns to pre-process complex text data (e.g., creating a standardized lowercase version of a column) can sometimes improve the overall responsiveness of your spreadsheet, although for standard text filtering, the native `CONTAINS` operator remains the most straightforward and effective method.

Conclusion and Additional Resources

The **Google Sheets QUERY function**, combined with the powerful `CONTAINS` operator, provides users with essential capabilities for dynamic data extraction based on partial text matches.

Whether you are isolating specific categories, excluding outliers using `NOT CONTAINS`, or consolidating multiple criteria using `OR`, mastering this fundamental syntax significantly elevates your data management skills within the spreadsheet environment.

By understanding the precise structure of the query string, including the importance of correct quotation usage and how to handle case sensitivity through functions like `LOWER()`, you can transform static, cumbersome data into actionable, filtered views that update seamlessly as your source data changes.

For those looking to deepen their expertise, exploring the official documentation on the Google Visualization Query Language is highly recommended. Pay particular attention to sections detailing the various comparison operators, aggregation functions, and advanced options like `MATCHES` that can be combined with the `CONTAINS` keyword for even more powerful data insights.