How to Easily Filter Data in Google Sheets Using the “Not Equal” Query

Authored by

stats writer

November 30, 2025

When working with large datasets in Google Sheets Query function, the ability to selectively filter data is paramount for effective analysis. The "not equal" operator provides a powerful mechanism for data exclusion, allowing users to quickly discard rows that match a specific criterion. This capability is fundamental in data cleaning and targeted reporting where exceptions must be ignored.

The standard syntax for expressing "not equal" within the query string typically involves the mathematical relational operator, represented by either the `<>` operator or the `!=` operator. In the context of the Google Visualization API Query Language, which powers the Sheets function, both symbols are generally accepted, although `!=` is highly common and easily readable. Following this operator, you specify the literal value or reference you wish to exclude. For instance, the command `"Select * from Sheet1 where Column1 <> 'Value'"` is designed to retrieve all records from Sheet1 where the content of Column1 is definitively not the literal text 'Value'.

Understanding how to implement this exclusion logic is vital for anyone manipulating data within the Google Sheets ecosystem. Whether you are dealing with numerical data, dates, or strings, mastering the "not equal" constraint ensures that your resulting dataset is precisely tailored to your reporting requirements, focusing only on the observations that truly matter for your analysis. This guide details the essential methods for applying these powerful exclusion criteria.

Understanding the Syntax Fundamentals of Exclusion

The primary goal of the "not equal" operator is to define exceptions within your data selection criteria. When writing a Google Sheets query, you are essentially writing a simplified version of an SQL SELECT statement. The `WHERE` clause is where all filtering happens, and it requires precise logical operators to function correctly. The two symbols for "not equal," `!=` and `<>`, serve the same exclusionary purpose, providing flexibility in syntax choice.

Choosing which operator to use often comes down to personal readability preference. Both operators perform an exact comparison: they check if the value in the specified column is completely different from the target value provided. It is important to remember that when querying strings (text), the target value must be enclosed in single quotes (e.g., `'Target'`). Failure to properly format the target value based on its data type will result in a `#VALUE!` error or inaccurate filtering results.

We will focus on two distinct scenarios for applying this filtering logic: the simple exclusion of a single specific value across one column, and the complex exclusion involving multiple values across one or more columns using logical connectors like `AND`. These techniques form the backbone of advanced data manipulation within Google Sheets, allowing for highly targeted data subsets.

You can use the following methods to use a "not equal" operator in a Google Sheets query:

Method 1: Not Equal to One Specific Value

This method involves applying the "not equal" operator directly to a single column to exclude a single, defined value. This is the simplest and most common application, used when you know exactly one undesirable item you want to filter out.

=query(A1:C11, "select * where A != 'Value1'")

Method 2: Not Equal to One of Several Values

When your exclusion criteria span multiple conditions--either excluding different values within the same column (using `AND`) or excluding specific values across multiple columns--you must combine the "not equal" operators using logical connectors. This approach allows for sophisticated, multi-layered data exclusion.

=query(A1:C11, "select * where A != 'Value1' and B != 'Value2'")

Note that **!=** is the primary "not equal" operator recognized and used consistently in Google Sheets Query function.

Setting the Context: The Sample Dataset

To illustrate these methods clearly and practically, we will apply the formulas to a sample dataset, typical of data encountered in administrative or analytical tasks. This dataset, contained within the range `A1:C11`, comprises records detailing various attributes, including Position, Team, and corresponding numerical data. Analyzing this structure helps visualize exactly which rows are retained and which are successfully excluded by the query criteria.

Understanding the structure of your data range is the critical first step before constructing any powerful query. Column A represents the Player Position, Column B represents the Team affiliation, and Column C holds numerical scores. Our examples will rely heavily on filtering based on the categorical text values found in Columns A and B, demonstrating robust text exclusion.

The following examples show how to use each formula in practice with the following dataset in Google Sheets:

	A	B	C	D
1	**Position**	**Team**	**Points**	
2	Guard	Lakers	13.4	
3	Guard	Mavericks	7.8	
4	Forward	Spurs	13.7	
5	Forward	Warriors	22.3	
6	Guard	Mavericks	27.8	
7	Center	Mavericks	20.8	
8	Forward	Spurs	12.7	
9	Center	Lakers	8.2	
10	Guard	Warriors	12.5	
11	Center	Warriors	30.2	
12				
13				
14				
15				
16				
17				
18				
19				

Practical Example 1: Excluding a Single Category Value

This first example demonstrates Method 1, focusing on filtering out rows that match a single, specific text value in a designated column. This scenario is useful when conducting analyses that must exclude a common category or an outlier that skews overall statistics, thereby isolating a particular subset of interest.

Suppose our objective is to analyze all players, but we specifically need to exclude all entries where the player's **Position** is 'Guard'. This requires applying the "not equal" constraint to Column A (Position) against the literal string 'Guard' using the `!=` operator within the `WHERE` clause.

We can use the following formula to select all rows where the **Position** column is not equal to 'Guard':

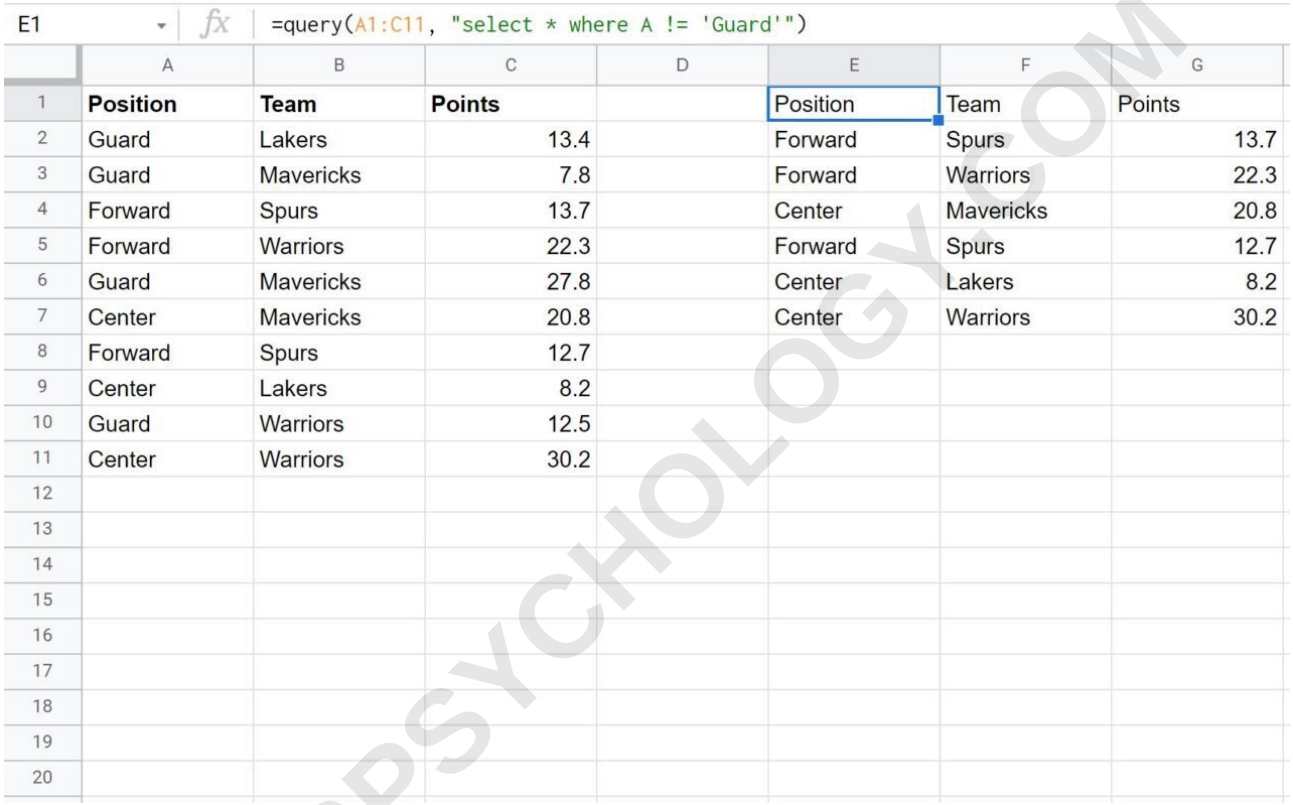
=query(A1:C11, "select * where A != 'Guard'")

The resulting output clearly demonstrates the effectiveness of the exclusion criterion. Every row where Column A contained the value 'Guard' has been successfully removed from the result set, leaving only records for other positions such as 'Forward' and 'Center'. This ensures that any

subsequent analysis or summarization performed on this filtered data accurately reflects the subset of players we are interested in.

It is crucial to verify that the value being excluded ('Guard') matches the capitalization and spelling present in the source data, as the query function is inherently case-sensitive.

The following screenshot shows how to use this formula in practice:



	A	B	C	D	E	F	G
1	**Position**	**Team**	**Points**		Position	Team	Points
2	Guard	Lakers	13.4		Forward	Spurs	13.7
3	Guard	Mavericks	7.8		Forward	Warriors	22.3
4	Forward	Spurs	13.7		Center	Mavericks	20.8
5	Forward	Warriors	22.3		Forward	Spurs	12.7
6	Guard	Mavericks	27.8		Center	Lakers	8.2
7	Center	Mavericks	20.8		Center	Warriors	30.2
8	Forward	Spurs	12.7				
9	Center	Lakers	8.2				
10	Guard	Warriors	12.5				
11	Center	Warriors	30.2				
12							
13							
14							
15							
16							
17							
18							
19							
20							

Notice that only the rows where the Position is not equal to 'Guard' are returned. The query successfully applied the `WHERE` clause using the not equal operator (`!=`), yielding a filtered table that is ready for further processing or visualization.

Practical Example 2: Excluding Multiple Combined Criteria

This example utilizes Method 2, demonstrating how to apply multiple exclusion criteria across different columns using the logical operator `AND`. This technique is indispensable when you need to narrow down your results by simultaneously discarding specific combinations of attributes or eliminating data based on distinct, unrelated filters.

Imagine we need to analyze all players except those who are Guards, and simultaneously, we must exclude any player associated with the 'Warriors' team, regardless of their position. This

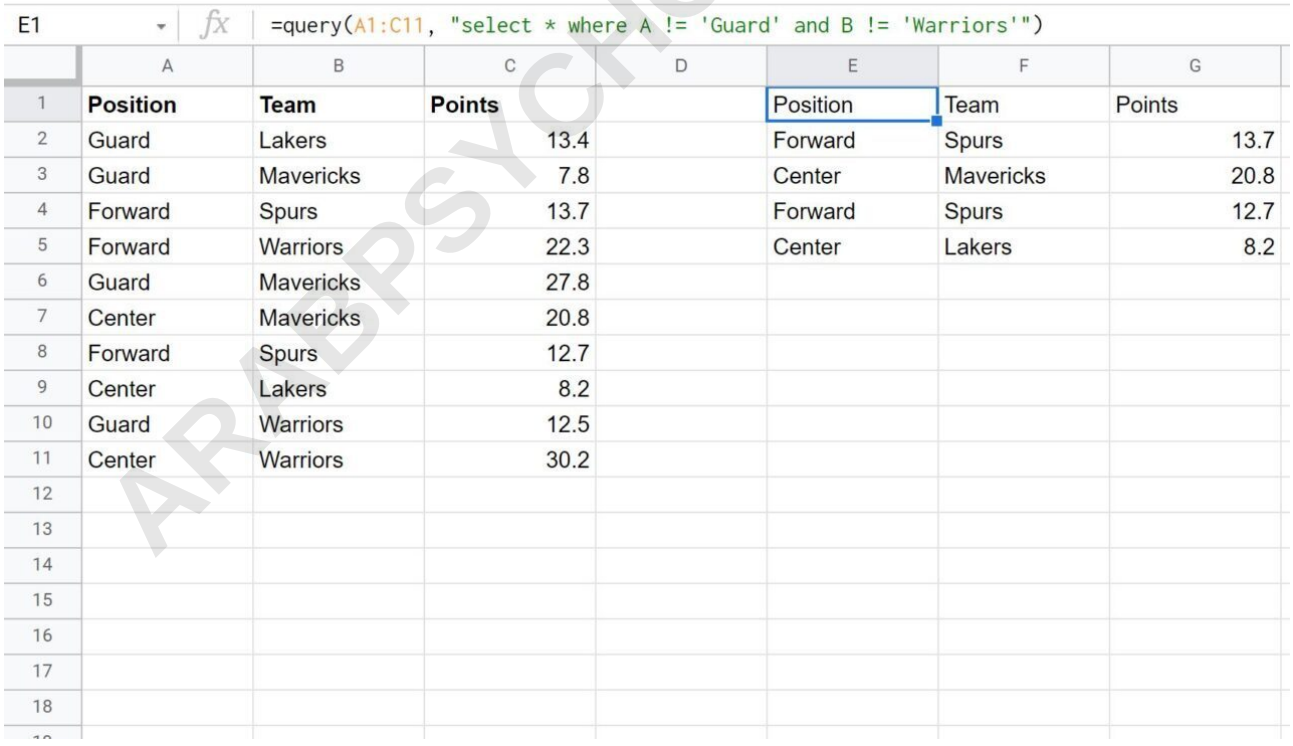
requires combining two distinct exclusion criteria using `AND`, ensuring both conditions are checked for every row.

We can use the following formula to select all rows where the **Position** column is not equal to 'Guard' and the **Team** column is not equal to 'Warriors':

=query(A1:C11, "select * where A != 'Guard' and B != 'Warriors'")

By using the `AND` operator between the two exclusion statements, we instruct the Google Sheets Query function to keep a row only if the value in Column A is not 'Guard' AND the value in Column B is not 'Warriors'. If a row happens to be 'Forward' (passes the first check) but belongs to 'Warriors' (fails the second check, as it is excluded), the entire row is discarded because the second exclusion criterion was met.

Reviewing the output confirms that the exclusion was successful. Rows that had 'Guard' in Column A are gone, and rows that had 'Warriors' in Column B are also gone. This compound filtering mechanism is highly effective for removing specific organizational units or roles from a consolidated report, providing a clean subset focused on remaining entities.



	A	B	C	D	E	F	G
1	**Position**	**Team**	**Points**		Position	Team	Points
2	Guard	Lakers	13.4		Forward	Spurs	13.7
3	Guard	Mavericks	7.8		Center	Mavericks	20.8
4	Forward	Spurs	13.7		Forward	Spurs	12.7
5	Forward	Warriors	22.3		Center	Lakers	8.2
6	Guard	Mavericks	27.8				
7	Center	Mavericks	20.8				
8	Forward	Spurs	12.7				
9	Center	Lakers	8.2				
10	Guard	Warriors	12.5				
11	Center	Warriors	30.2				
12							
13							
14							
15							
16							
17							
18							

Notice that only the rows where the **Position** is not equal to 'Guard' and the **Team** is not equal to 'Warriors' are returned. This result highlights the power of combining operators derived from SQL logic within the familiar environment of Google Sheets.

Crucial Considerations: Case Sensitivity and Data Types

The Google Sheets query environment, particularly when dealing with textual data, is fundamentally case-sensitive. This attribute is a frequent source of error for new users attempting to filter data. Case sensitivity means that the value you provide in quotes must precisely match the capitalization of the value in the source data for the comparison to work correctly.

For example, if the source data contains 'Guard' with a capital 'G', and your query specifies `A != 'guard'`, the query will fail to exclude the desired rows. Instead, it will treat 'Guard' and 'guard' as two entirely different, non-matching values. Consequently, the row containing 'Guard' will be returned because it technically does not equal 'guard'.

To circumvent issues related to case sensitivity, especially when dealing with user-entered or messy data, best practice suggests using built-in functions like `LOWER()` or `UPPER()` if you process the data before querying, or using the more complex `matches` operator in conjunction with regular expressions (regex) within the query string itself to enforce case-insensitive matching if necessary. However, for simple exclusions using `!=`, exact case matching is mandatory.

Beyond case, remember that data types matter significantly. When excluding numerical values or dates, they require specific formatting that differs from string literal values. Numerical values should not be wrapped in single quotes (e.g., `where C != 95`). Dates, however, must be formatted using the `date 'YYYY-MM-DD'` structure to ensure the query language interprets them correctly, such as `where D != date '2023-11-01'`. Mismatching the data type format is a common cause of query failure.

Note: The **query()** function is case-sensitive. This means that if you type **A != 'guard'** then the query will still return rows where the Position is Guard because the two values don't have the same case.

Troubleshooting and Further Learning

Constructing complex exclusion queries requires careful attention to detail to avoid common pitfalls. One frequent issue arises when comparing values that might contain leading or trailing spaces. Even a single space character can cause a row to be mistakenly included because the source data `'Value '` is not equal to your query string `'Value'`. Always ensure your source data is clean, perhaps by using the `TRIM()` function on the source range before executing the query.

For scenarios where you need to exclude multiple values from a single column, the most idiomatic and readable approach, derived from SQL, is to use the `NOT IN` clause. For example, to exclude both 'Guard' and 'Center' positions, you would write `where A not in ('Guard', 'Center')`. This is cleaner and less error-prone than chaining multiple `!=` operators with `AND`.

The following tutorials explain how to perform other common tasks with Google Sheets queries:

Google Sheets Query: How to Use Group By