Google Sheets: Sort and Ignore Blanks

Authored by

Mohammed looti

October 29, 2025

Mastering Efficient Data Sorting in Google Sheets

Effective data organization is paramount for any meaningful analysis conducted within Google Sheets. While standard data sorting methods are straightforward, a significant complication frequently arises when the target column for sorting contains blank cells. These empty entries, often referred to as null values, can disrupt the desired sequence, leading to disorganized or visually misleading output when dealing with a large dataset. Addressing this common hurdle requires a more sophisticated approach than simple menu-based sorting.

Fortunately, Google Sheets provides robust functionalities designed to handle these data complexities. This expert guide focuses specifically on how to harness the power of the versatile QUERY function. We will demonstrate a precise method for executing advanced data sorting operations that not only orders your information but also intelligently filters out rows containing blank cells in the critical sort column, ensuring only complete and relevant data is presented.

By the conclusion of this tutorial, you will possess the foundational knowledge required to construct a powerful QUERY function statement. This technique is invaluable for maintaining exceptionally clean and accurate data representations, allowing you to prioritize the analysis of non-empty entries while completely excluding rows where the sorting criterion is absent. This level of control is essential for professional spreadsheet management.

Leveraging the QUERY Function for Dynamic Data Manipulation

The QUERY function stands as one of the most powerful and flexible features within Google Sheets. It operates similarly to a streamlined version of the SQL database language, allowing users to perform complex selections, filtering, aggregations, and sorting directly on spreadsheet data ranges. Understanding its basic syntax is the first step toward mastering dynamic data manipulation: `=QUERY(data, query, )`. Here, `data` specifies the range being analyzed, `query` is the string containing the SQL-like commands, and `headers` is an optional argument denoting the number of header rows.

To achieve our objective of sorting data while simultaneously ignoring blanks, we must strategically employ three core SQL clauses within the query string. These clauses work in tandem to define the scope, criteria, and final presentation of the data extraction:

SELECT clause: Defines which columns are returned in the final result set. Using `SELECT *` is the most common approach when the goal is to retrieve all columns from the source range.

WHERE clause: Crucially filters the rows based on specific conditions. For our purpose, the WHERE clause will implement the logic necessary to exclude rows that contain null values in the designated sort column.

ORDER BY clause: Specifies the column or columns used for sorting the filtered results, along with the desired direction (either ascending or descending).

By combining these foundational elements, the QUERY function provides precise and non-destructive control over how your source data is processed, ensuring that the final output is both highly relevant and accurately structured according to your requirements.

The Definitive Formula for Filtering and Sorting Null Values

The core formula required to execute a sort operation in Google Sheets that simultaneously excludes rows where the sort column contains blank values is elegantly straightforward. This powerful QUERY function utilizes a specific syntax tailored for this filtering requirement:

=QUERY(A1:B11,"select * where B is not null order by B")

To fully appreciate the functionality, we must analyze the structure and intent of the query string embedded within the formula:

`A1:B11`: This first argument clearly defines the **source data range**. The QUERY function will perform all subsequent operations exclusively on the data contained within these specified cells.

`"select * where B is not null order by B"`: This command string is the heart of the operation, written in the SQL-like language supported by the function.

`select *`: This instruction specifies the retrieval of **all columns** from the designated range, ensuring that the entire record remains intact.

`where B is not null`: This is the critical **filtering condition**. It mandates that only rows where the value in **column B is populated** (i.e., not empty or null) are passed through. This single clause achieves the goal of excluding all blank entries.

`order by B`: This final instruction dictates that the resulting filtered data set must be sorted based on the values present in **column B**. If no direction is specified, the default ascending order is applied.

This robust, single-formula solution ensures methodical data handling. It guarantees that the resulting dataset is not only perfectly organized but also rigorously filtered to remove the visual and analytical interference caused by empty records in the sort column.

Case Study: Applying the Formula to a Real-World Dataset

To demonstrate the practical application and efficacy of this filtering and sorting technique, let us

examine a common scenario. Consider a spreadsheet used to manage sports statistics, tracking various basketball teams along with their scores. Inevitably, data collection delays or errors result in missing entries in the "Points" column (Column B), creating a fragmented dataset.

The following image displays our initial source data. Note the presence of rows where the "Points" value is clearly missing--these are the entries we aim to exclude from our final sorted report.

	A	B	C	D
1	**Team**	**Points**		
2	Mavs	99		
3	Nets	104		
4	Hawks			
5	Warriors	86		
6	Celtics	97		
7	Heat	109		
8	Magic	114		
9	Thunder			
10	Spurs	100		
11	Rockets	94		
12				
13				
14				
15				
16				

Our mandate is clear: sort this information based on the "Points" column (B) while completely omitting any team that lacks a recorded score. We will implement the QUERY function by placing it in an empty cell, such as **D1**, which will serve as the starting point for the new, organized output table.

=QUERY(A1:B11,"select * where B is not null order by B")

Once the formula is entered, Google Sheets executes the query, first filtering the data via the `WHERE` clause and then applying the sort via the `ORDER BY` clause. The subsequent screenshot vividly illustrates the results, confirming that all rows corresponding to blank values in the "Points" column have been successfully excluded, and the remaining teams are sorted correctly in ascending order.

D1 =QUERY(A1:B11,"select * where B is not null order by B")

	A	B	C	D	E
1	**Team**	**Points**		Team	Points
2	Mavs	99		Warriors	86
3	Nets	104		Rockets	94
4	Hawks			Celtics	97
5	Warriors	86		Mavs	99
6	Celtics	97		Spurs	100
7	Heat	109		Nets	104
8	Magic	114		Heat	109
9	Thunder			Magic	114
10	Spurs	100			
11	Rockets	94			
12					
13					
14					
15					
16					
17					
18					
19					

The output confirms the precision of the method: the results are dynamically generated, sorted exclusively by the "Points" column, and entirely free of clutter from incomplete records.

Implementing Custom Sort Orders: Ascending and Descending

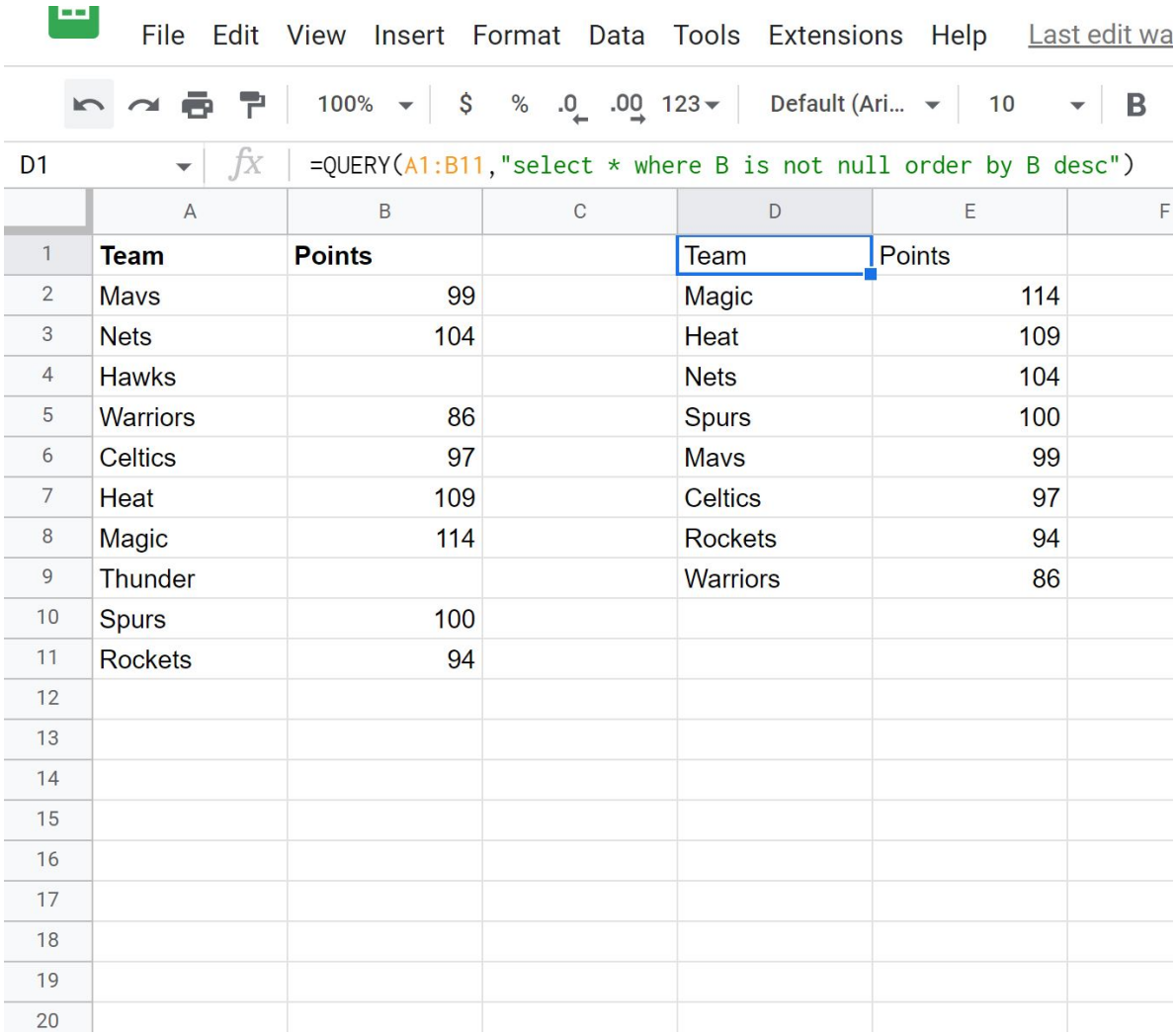
When employing the ORDER BY clause within the QUERY function without any explicit direction, the system defaults to ascending order (ASC). This standard arrangement sorts numerical data from the smallest value to the largest, and text data alphabetically (A to Z). This is ideal for viewing minimum values or early entries.

However, in many analytical contexts, the inverse arrangement is necessary. For example, when analyzing sales figures, performance metrics, or high scores, prioritizing the largest values is critical. To reverse the sort order and achieve a descending order, one must simply append the keyword `DESC` after the column identifier in the ORDER BY clause.

Returning to our basketball dataset, suppose we wish to rank the teams from the highest score to the lowest, while still maintaining the integrity of our data by excluding all blank "Points" entries. The modified QUERY function formula incorporates the new directional modifier:

=QUERY(A1:B11,"select * where B is not null order by B desc")

The addition of `desc` dynamically changes the sort behavior without altering the filtering criteria. The subsequent image demonstrates the outcome of this modification: the data is now organized with the highest scores appearing at the top of the output, maintaining a clear separation from the source data and excluding any blank rows.



	A	B	C	D	E	F
1	**Team**	**Points**		Team	Points	
2	Mavs	99		Magic	114	
3	Nets	104		Heat	109	
4	Hawks			Nets	104	
5	Warriors	86		Spurs	100	
6	Celtics	97		Mavs	99	
7	Heat	109		Celtics	97	
8	Magic	114		Rockets	94	
9	Thunder			Warriors	86	
10	Spurs	100				
11	Rockets	94				
12						
13						
14						
15						
16						
17						
18						
19						
20						

This ability to precisely define both the filter logic and the sort direction solidifies the QUERY function as an indispensable utility for advanced data presentation and analytical reporting in Google Sheets.

Strategic Advantages and Best Practices for QUERY

The decision to utilize the QUERY function for sorting and filtering nulls offers substantial operational benefits over traditional, manual sorting methods. Firstly, it establishes a **dynamic link**

to the source data. Any modification, addition, or deletion in the source range automatically triggers an update in the QUERY result, providing a live, accurate, and filtered view--a feature critical for dashboards and frequently updated datasets.

Secondly, the function provides unparalleled **flexibility and scalability**. Unlike simple filters, QUERY allows for the seamless integration of other powerful SQL clauses, enabling complex operations such as summing data (`GROUP BY`), performing calculations, and creating summary tables, all contained within a single formula string. This modularity makes it a centerpiece for advanced data analysis workflows.

To ensure optimal performance and maintainability, adhere to the following best practices: Always **verify the accuracy of your data range**, particularly when dealing with expanding data sets (e.g., using open ranges like `A:B`). When structuring the query string, always reference columns using their alphabetical identifiers (A, B, C, etc.) relative to the defined range, regardless of the column headers. While the QUERY function is generally highly efficient, complex nested queries on massive datasets should be tested for potential performance impacts, though for typical corporate uses, its speed is excellent.

Expanding Data Proficiency Beyond Basic Sorting

Mastering dynamic data manipulation techniques, such as those demonstrated with the QUERY function, is paramount for elevating your proficiency within Google Sheets. This function represents a gateway to transforming raw data into highly organized and actionable insights.

We encourage users to continue expanding their knowledge by exploring the full spectrum of advanced data management tools available. Key areas for further study include utilizing array formulas, developing sophisticated conditional formatting rules, and learning how to integrate the QUERY function with other powerful functions like `IMPORTRANGE` for cross-sheet data analysis.

For continued learning and detailed walkthroughs covering various aspects of sorting, filtering, and comprehensive data analysis techniques in Google Sheets, consult authoritative documentation and dedicated expert resources. This ongoing educational effort will significantly enhance your analytical capabilities and overall spreadsheet expertise.