How to Sort Google Sheets Query Results with Order By

Authored by

stats writer

December 5, 2025

The Google Sheets Query function is arguably the most powerful tool available within the platform, allowing users to manipulate datasets using commands derived from Structured Query Language (SQL). This function provides dynamic filtering, aggregation, and, most critically, sophisticated data sorting capabilities. While simple sorts are possible using standard spreadsheet menus, the `QUERY` function enables complex, multi-layered data sorting that refreshes automatically as the source data changes.

Central to effective data organization within the `QUERY` function is the **Order By** clause. This critical component dictates the sequence in which the resulting records are displayed. Understanding how to correctly implement **Order By** is essential for transforming raw data into highly organized, actionable information. Whether you need to arrange financial records by date, prioritize tasks by status, or display leaderboard results by score, the `QUERY` function's sorting power ensures clarity and precision in your spreadsheet analysis.

The ability to sort results in either ascending order (A-Z, lowest-highest) or descending order (Z-A, highest-lowest) across single or multiple columns gives the user complete control over the presentation of their data. This guide will delve deep into the mechanics of the **Order By** clause, providing syntax rules, practical examples, and expert tips to help you master advanced sorting techniques within Google Sheets.

Understanding the Anatomy of the QUERY Function

Before implementing the **Order By** clause, it is vital to grasp the foundational structure of the `QUERY` function itself. The function requires three distinct arguments: the **Data Range**, the **Query String**, and the optional **Header Count**. The Data Range defines the source table that the query will operate on, typically formatted as `A1:Z100`. This range must be accurately defined to ensure all relevant information is included in the processing.

The second argument, the **Query String**, is where the magic happens. This string, always enclosed in double quotes, contains the SQL-like commands--such as `SELECT`, `WHERE`, and `ORDER BY`--that define how the data should be filtered, processed, and structured. Because the query string uses column letters from the defined range (e.g., A, B, C) rather than cell addresses, it makes the logic easy to read and manage, even for very large datasets spanning hundreds of rows and columns. It is within this string that we place the `ORDER BY` statement.

The final argument, the **Header Count**, specifies how many header rows are present in the defined Data Range. Setting this to **1** (as is common) tells the function to treat the first row as column labels, ensuring it is excluded from the data processing and sorting operations. This prevents the header row from being mistakenly sorted alongside the actual data, which is a common error for beginners when using the `QUERY` function for the first time.

Implementing the ORDER BY Clause: Basic Syntax

The basic implementation of the **Order By** clause is straightforward. You must specify which column you wish to sort by, followed by the sorting direction: `ASC` for ascending order or `DESC` for descending order. If no direction is specified, the function defaults to `ASC`. The sorting is applied after the `SELECT` and `WHERE` clauses have defined the subset of data to be returned, ensuring only the finalized results are ordered.

To use the **Order By** clause effectively, it must be placed at the end of the SQL query string, typically after the `SELECT` clause (and any optional `WHERE` or `GROUP BY` clauses). The structure is simple: `ORDER BY `. For instance, to sort a list of customer names (in Column B) alphabetically from A to Z, you would specify `ORDER BY B ASC` within your query string. This clear, declarative syntax is what makes the Google Sheets Query function so efficient for data manipulation.

You can use the following standard syntax to order the results of a Google Sheets Query by a single column. Note how the direction keyword (`asc`) determines the final presentation of the data, arranging it from the smallest to the largest value based on the selected column. If the data is text, it will be sorted alphabetically; if it is numeric, it will be sorted numerically.

You can use the following syntax to order the results of a Google Sheets Query by a certain column:

=query(A1:C12, "select A, B order by B asc", 1)

In this example, we select columns **A** and **B** and order the results by column B ascending. We also specify a **1** to indicate that there is 1 header row at the top of the dataset.

Mastering Descending Sort (DESC) and Data Types

While ascending sort (`ASC`) is the default and is useful for chronological or alphabetical ordering, the descending sort (`DESC`) is crucial for creating leaderboards, ranking highest values, or showing the most recent entries first. When using `DESC` with numeric data, the results will display the largest numbers first, progressively moving down to the smallest. For text data, `DESC` will sort alphabetically from Z to A.

It is important to remember that the Google Sheets Query function is highly sensitive to data type consistency. If a column intended for numerical sorting contains mixed data--some cells formatted as numbers and others as text--the sorting results may be unpredictable. For instance, text values are often treated as zero or grouped separately. Therefore, always ensure the column you are using in the **Order By** clause is uniformly formatted (e.g., all numbers, all dates, or all consistent

text strings) for reliable data sorting.

Furthermore, when sorting dates, the `QUERY` function treats dates as numeric values representing the number of days since a certain epoch. Therefore, sorting dates using `ASC` will place the oldest dates first, and using `DESC` will place the newest (most recent) dates first. This functionality makes the `QUERY` function superior to simple manual sorting, especially when dealing with complex time series data or large chronological records where precise ordering is paramount for analysis.

Advanced Sorting: Using ORDER BY with Multiple Columns

One of the most powerful features of the **Order By** clause is the ability to sort by multiple columns sequentially. This is essential for creating detailed, hierarchical views of your data. For example, you might want to sort a list of sales employees first by their region (alphabetically) and then, within each region, by their total sales figures (highest to lowest). This two-tiered sorting provides a contextually rich view of performance.

When sorting by multiple columns, the columns are listed in the query string separated by commas, and the sorting is performed in the order they appear. The first column specified acts as the primary sort key. If two or more rows have the same value in the primary sort column, the function then applies the secondary sort key to those tied rows. This process continues down the list of specified columns, resolving ties until all records are uniquely ordered or all specified columns have been exhausted.

The flexibility of multi-column sorting also extends to mixing directions. You can apply ascending order to the primary column and descending order to the secondary column, or vice versa, tailoring the output exactly to the analytical requirement. This granular control is what sets the SQL-like `QUERY` function apart from simpler filtering tools. The following demonstrates the syntax for sorting by two columns with different directions:

You can also use the following syntax to order by multiple columns:

=query(A1:C12, "select A, B order by B asc, A desc", 1)

In this example, we select columns **A** and **B** and order the results by column B ascending, then by column A descending.

The following examples show how to use these formulas in practice, based on a sample dataset of player statistics.

Practical Application: Ordering Data by a Single Column (Ascending Example)

When working with datasets that require simple alphabetical or numerical ordering, sorting by a single column in ascending order is the most common approach. Consider a scenario involving a sports league table where you want to organize all players by the name of their team. This ensures that all members of the same team are grouped together, making team-level analysis much easier.

In this specific example, we utilize a dataset that contains Player, Team, and Points columns. We aim to display only the Player and Team information, but we require the results to be neatly organized based on the Team column alphabetically. This requires specifying `ORDER BY ASC`. Assuming the Team column is B, the clause becomes `ORDER BY B ASC`. The `SELECT` clause restricts the output to only the necessary columns, resulting in a clean, focused, and sorted output table.

The resulting output clearly demonstrates how the system processes the `QUERY`. First, it extracts the data defined by the `SELECT` statement, and then the **Order By** clause rearranges these results. Notice how the Team names are presented from A to Z, guaranteeing logical grouping. This method is highly effective for quickly generating organized subsets of data from complex master tables, vastly improving data consumption and interpretability for reports or dashboards.

Example 1: Order By One Column Ascending

We can use the following formula to select the Player and Team columns, then order the results by Team in ascending order:

E1 =query(A1:C12, "select A, B order by B asc", 1)

	A	B	C	D	E	F
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		Bob	Mavericks
5	Dave	Warriors	22.3		Eric	Mavericks
6	Eric	Mavericks	27.8		Fred	Mavericks
7	Fred	Mavericks	20.8		Carl	Spurs
8	George	Spurs	12.7		George	Spurs
9	Harold	Lakers	8.2		Ken	Spurs
10	Isaiah	Warriors	12.5		Dave	Warriors
11	Joe	Warriors	30.2		Isaiah	Warriors
12	Ken	Spurs	22.4		Joe	Warriors
13						
14						
15						
16						
17						
18						
19						
20						
21						
22						

Practical Application: Ordering Data by a Single Column (Descending Example)

Conversely, the need to identify top performers or recent activity necessitates the use of the descending sort direction. A frequent use case is the generation of a leaderboard, where the highest score or largest quantity must appear at the top of the list. If our dataset includes player performance metrics, sorting by the Points column in descending order is the optimal way to rank the players.

In this example, we decide to select all available columns (`select *`) to provide a complete view of the data, but we mandate that the results must be ordered by the Points column using the `DESC` keyword. If Points is in Column C, the clause is `ORDER BY C DESC`. This simple modification instantly flips the sorting logic, prioritizing the highest numerical values. This is an indispensable technique for quick ranking and performance assessment.

The resulting image shows the players ranked from the highest points scored down to the lowest. This transformation of the raw data into a ranked list showcases the power and efficiency of the **Order By** clause in filtering and prioritizing information based on magnitude. Whether analyzing

sales performance, inventory levels, or website traffic, using `DESC` ensures that the most impactful data points are immediately visible, facilitating rapid decision-making based on peak values.

Example 2: Order By One Column Descending

We can use the following formula to select all of the columns and order the results by Points in descending order:

E1 fx =query(A1:C12, "select * order by C desc", 1)

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

Practical Application: Combining Sort Orders (Multiple Columns Example)

The true mastery of the **Order By** clause comes when combining multiple columns and directions to achieve highly specific data hierarchy. Continuing with the sports example, a manager might need to see players grouped by their Team, but within each team, they want to rank the players by their points scored. This requires a two-level sort: primary sort by Team (`ASC`) and secondary sort by Points (`DESC`).

To implement this, the query string must specify both sort criteria separated by a comma. We order first by Team ascending (`ORDER BY B ASC`) to group all teams together alphabetically. Then,

we use the secondary sort key, Points descending (`C DESC`), which is applied only to the subset of records sharing the same team name. This ensures that the highest-scoring players within 'Team Alpha' appear at the top of the 'Team Alpha' section, followed by the highest-scoring players in 'Team Beta,' and so on.

This multi-column sorting technique provides a level of organization that is crucial for complex reporting. It moves beyond simple list generation to structure the data into meaningful analytical blocks. The visual evidence in the corresponding image confirms this: teams are in alphabetical order, and within each team, the points are correctly ranked from highest to lowest. This is the cornerstone of sophisticated data sorting and visualization using the Google Sheets Query function.

Example 3: Order by Multiple Columns

We can use the following formula to select all columns and order the results first by Team ascending, then by Points descending:

E1 fx =query(A1:C12, "select * order by B asc, C desc", 1)

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

Tips and Best Practices for Using ORDER BY

To ensure your `QUERY` functions remain robust and efficient, especially as your spreadsheet data grows, adherence to several best practices is recommended. First, always explicitly specify the sort direction (`ASC` or `DESC`), even if you intend to use the default ascending order. This practice improves readability and prevents potential confusion for future users who might inherit your formula. Ambiguity in SQL-like syntax can often lead to difficult-to-debug errors.

Secondly, pay close attention to the column index used in the `ORDER BY` clause. These must correspond to the column letters of the original data range specified in the first argument of the `QUERY` function, not the column letters of the resulting output table. A common mistake is adjusting the column references within the `ORDER BY` clause when the `SELECT` clause changes, which is incorrect. The query logic always refers back to the initial data source defined in the formula's range.

Finally, understand that the **Order By** clause dictates the final presentation, but it is executed after all filtering and grouping (from `WHERE` and `GROUP BY` clauses) have occurred. If you apply a `LIMIT` clause to restrict the number of rows returned, always place the `ORDER BY` clause immediately before the `LIMIT` clause. This guarantees that the function sorts the entire dataset first and then selects the top N results based on that defined order, ensuring that the results presented are the true top or bottom records according to your criteria.

Google Sheets Query: How to Use Group By