How to Easily Select Multiple Columns with Google Sheets Query

Authored by

stats writer

December 5, 2025

The Google Sheets Query function is arguably the most powerful tool available for data manipulation within the spreadsheet environment. It allows users to perform complex data retrieval and analysis, mirroring the capabilities found in traditional database systems using a structure similar to SQL (Structured Query Language). Understanding how to utilize this function effectively is crucial for anyone handling large datasets in Google Sheets. At its core, the function is designed to query data based on specified criteria and, critically, to precisely retrieve only the necessary columns of data from a source spreadsheet. This efficiency ensures cleaner output and faster data processing, particularly when working with extensive sheets that contain dozens or even hundreds of columns.

When structuring a query, one of the most fundamental operations is defining which columns you intend to extract. This is achieved using the central keyword, "SELECT", which tells the function exactly what data fields to include in the resulting output. While simple selection might involve just one column, most real-world applications require selecting multiple columns simultaneously. This process involves listing out the specific column references--such as A, B, C, etc.--separated by a comma within the query string. This method provides explicit control over the output structure. Alternatively, for scenarios where every single piece of information is required, the function offers a shortcut: using the asterisk (*****) wildcard. The results generated by the query are then dynamically displayed in the designated output range, creating a clean, filtered subset of the original data.

Mastering column selection is the first step toward advanced data querying. Whether you are aggregating sales data, filtering user lists, or generating custom reports, the ability to specify multiple columns streamlines your workflow significantly. The structured approach of the query function ensures that the output remains dynamic, updating automatically whenever the source data changes, which is a major advantage over static copy-pasting methods. Furthermore, learning the nuances of the **SELECT** command prepares you for integrating more complex clauses like **WHERE**, **GROUP BY**, and **ORDER BY**, enabling sophisticated data analysis directly within your spreadsheet interface.

Understanding the Core QUERY Syntax

To execute any data extraction operation, it is essential to grasp the precise syntax required by the Query function in Google Sheets. The function adheres to a strict structure consisting of three primary arguments. The first argument defines the source **data range**, which specifies the cells or named range containing the data you wish to analyze. The second argument is the actual **query string**, which must be enclosed in quotation marks, containing the SQL-like commands (beginning with SELECT). Finally, the third optional argument dictates the number of header rows present in the source data, which helps the function correctly interpret and categorize the output columns.

The flexibility of the query string allows for precise control over the data filtering and presentation.

Although the syntax mimics SQL, there are minor differences, particularly in how column identifiers are handled (using letters A, B, C, or internal references Col1, Col2, Col3, depending on context). The query string is the engine of the function, and meticulous attention to its formatting--including proper spacing, comma separation, and correct quotation usage--is necessary to avoid common parsing errors. If the syntax is incorrect, the function will return an error, halting the data transformation process. Therefore, understanding the position of the **SELECT** clause within this string is paramount.

The following example illustrates the mandatory structure for selecting data using the function. Note how the arguments are positioned: the source range is first, followed by the command string, and then the header count. This structure ensures that the function correctly identifies the data source, processes the instruction, and formats the output with appropriate column headers. This fundamental structure is the foundation upon which all subsequent complex queries are built, regardless of how many columns are eventually selected or how complex the filtering criteria become.

You can use the following syntax to select multiple columns using the Google Sheets query function:

=query(Range, "select A, B, C", 1)

This particular query selects columns A, B, and C in a dataset, and the **1** specifies that there is 1 header row at the top of the dataset. This parameter is critical for accurate sorting and grouping operations later in the query structure.

Implementing Explicit Multiple Column Selection

When you need to retrieve a non-contiguous set of data columns from your source range, the explicit listing method is utilized. This involves calling the **SELECT** keyword followed by the column identifiers, separated by commas. For instance, if your source data covers columns A through G, but you only require the Player Name (A), Team (C), and Points Scored (G), your query string would look like: `"SELECT A, C, G"`. This highly specific method ensures that only the relevant attributes are returned, significantly decluttering the resulting sheet and focusing the data on immediate reporting needs.

It is important to remember that the order in which you list the columns within the **SELECT** clause determines the order in which they appear in the final output. If the source data has columns A, B, C, but you specify `"SELECT C, A, B"`, the output will display Column C data first, followed by A, and then B. This feature provides powerful flexibility, allowing users to reorganize their data presentation without manually rearranging the source sheet itself. Furthermore, the column references must correspond exactly to the columns within the data range defined in the first

argument of the function.

Consider a scenario where a master sheet contains demographic data, but a report requires only names and contact information. By using comma separation, you can easily pull columns D (Name), E (Email), and H (Phone Number), ignoring potentially sensitive or irrelevant information like salary or internal IDs. This targeted approach not only enhances data security by limiting the visibility of certain fields but also dramatically improves the readability of generated reports. The precision offered by explicitly selecting multiple columns makes the Query function indispensable for focused data extraction.

The following examples illustrate how to use this function in practice with the dataset shown below:

	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					

Example 1: Selecting Specific Multiple Columns

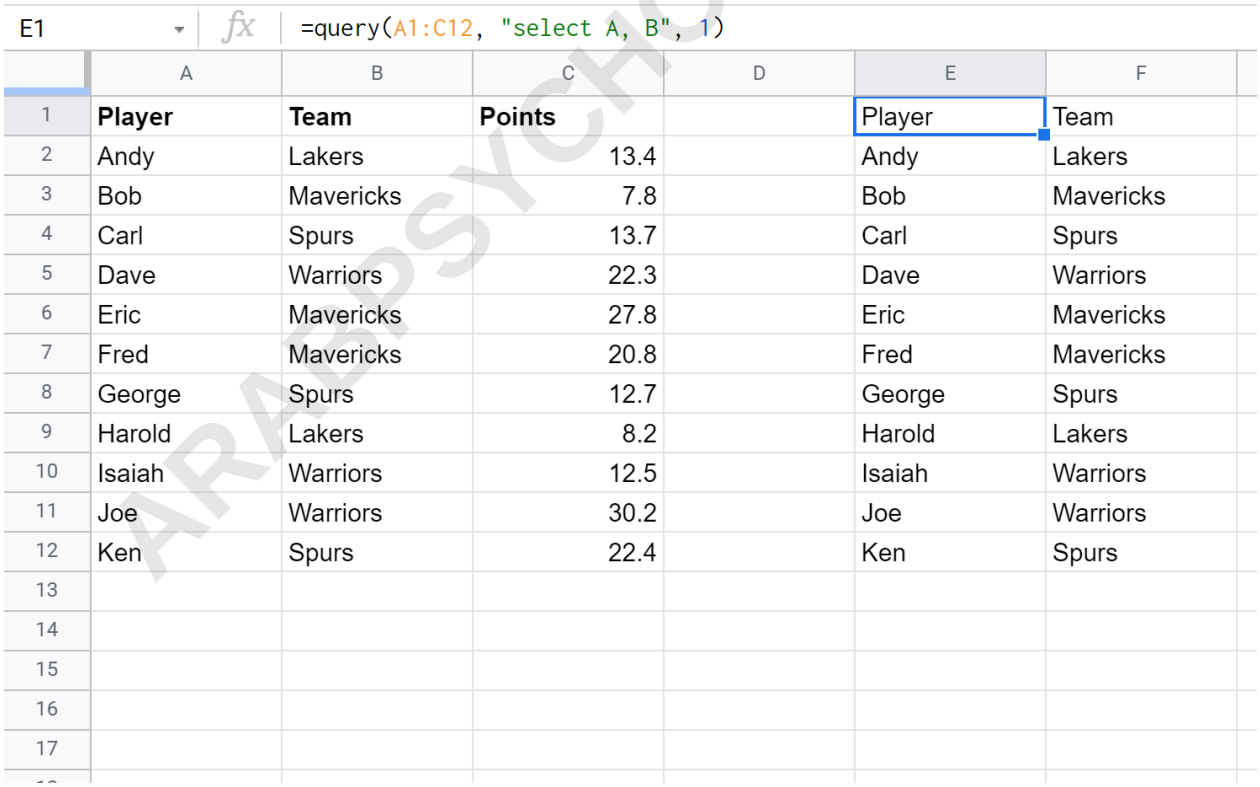
In this initial practical example, we demonstrate the simplest form of multiple column selection using the comma-separated list. Utilizing the sample dataset provided above, let us assume we are interested only in identifying the **Player Name** and the **Team** they represent, while excluding the 'Points' and 'Rebounds' data for the moment. The Player Name is located in Column A, and the

Team is in Column B of the source data range. Therefore, the query must specifically instruct the function to retrieve data from these two identifiers.

To execute this operation, we formulate the query string to begin with the **SELECT** clause, immediately followed by the column letters A and B, separated by a comma. The overall formula must ensure the source data range is correctly defined (e.g., A1:D100) and that the header count is set appropriately (in this case, 1). The resultant output will be a two-column table containing every row from the source data, but limited only to the content of columns A and B, maintaining the integrity of the data relationships.

The resulting syntax is exceptionally clean and efficient: `=QUERY(A1:D10, "SELECT A, B", 1)`. This formula instructs Google Sheets to select all rows, but only display the values corresponding to column A (Player) and column B (Team). This exemplifies how the power of the SELECT keyword allows for targeted retrieval, ensuring that only the essential variables are presented to the end-user, thereby maximizing report clarity and minimizing information overload. This foundational technique is reusable across any Google Sheet dataset structure.

We can use the following syntax to select all rows from the Player and Team columns:



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

Example 2: Selecting Multiple Columns Based on Condition

One of the most valuable aspects of the Google Sheets Query function is its ability to combine

column selection with data filtering. This is accomplished by integrating the **WHERE** clause directly into the query string, immediately following the **SELECT** clause. The **WHERE** clause allows users to specify criteria that each row must meet to be included in the final output. When selecting multiple columns, the **WHERE** clause acts as a filter on the entire source data before the selected columns are presented.

For instance, if we still want to view the Player and Team columns (A and B), but we are only interested in players belonging to a specific organization, such as the "Mavericks," we must add a conditional statement. This conditional statement checks the value of the Team column (B) against the specified criteria. The syntax requires that text values (strings) used in the condition must be enclosed in single quotes, ensuring the query language correctly interprets the filtering value.

The resulting query string for this conditional selection would be: `"SELECT A, B WHERE B = 'Mavericks'"`. This tells the function to first filter the dataset to include only rows where Column B equals 'Mavericks', and then, from those filtered rows, display only the data found in Column A and Column B. This demonstrates an extremely powerful synthesis of selection and filtering capabilities, yielding highly precise and actionable data subsets. The ability to select multiple columns while simultaneously applying robust filtering criteria is what elevates the Query function above simpler filtering tools available in Google Sheets.

For example, we can use the following syntax to select the Player and Team columns **where** the Team is equal to Mavericks.

E1 =query(A1:C12, "select A, B where B='Mavericks'", 1)

	A	B	C	D	E	F
1	**Player**	**Team**	**Points**		Player	Team
2	Andy	Lakers	13.4		Bob	Mavericks
3	Bob	Mavericks	7.8		Eric	Mavericks
4	Carl	Spurs	13.7		Fred	Mavericks
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						

The Power of the Asterisk (*): Selecting All Columns

While explicit column selection (e.g., SELECT A, B, C) provides precise control and is generally preferred for production reports, there are situations where retrieving all available data fields is necessary. The query language accommodates this need by recognizing the asterisk (*****) as a wildcard character. When placed immediately after the **SELECT** keyword, the asterisk instructs the function to include every column present within the defined data range, regardless of how many columns that range encompasses.

Using the asterisk simplifies the query formula immensely, transforming `"SELECT A, B, C, D, E, F, G, H..."` into the concise command `"SELECT *"`. This is particularly useful during the exploratory data analysis phase or when the source data structure is frequently changing. If new columns are added to the source range, using `"SELECT *"` ensures that these new data fields are automatically included in the query output without the need for manual formula updates, enhancing the maintainability of the spreadsheet solution.

It is crucial, however, to pair the `"SELECT *"` command with appropriate filtering, especially when

dealing with very wide datasets. A query that simply says `"SELECT *"` will return the entirety of the source data, which can sometimes defeat the purpose of using the Query function for targeted extraction. Therefore, the asterisk is most effectively utilized in conjunction with other clauses, such as **WHERE**, to filter rows, or **ORDER BY**, to sort the results, ensuring that while all columns are returned, only the relevant subset of rows is presented.

Example 3: Selecting All Columns Using the Wildcard

To illustrate the efficiency of the wildcard selector, we apply it to our sample dataset. Instead of listing A, B, C, and D individually, we use the asterisk to retrieve all columns: Player, Team, Points, and Rebounds. This example assumes our data range is A1:D10. The resulting formula is simply: `=QUERY(A1:D10, "SELECT *", 1)`. This instructs the function to return the full width of the data range specified, offering a complete view of all attributes for all players listed in the source data.

This technique is foundational for creating 'mirror' tabs or backup copies of data, where the goal is to duplicate the source structure entirely but perhaps apply a simple filtering rule (e.g., showing only active rows). Even when using `"SELECT *"`, you still retain the ability to apply a **WHERE** clause. For instance, `"SELECT * WHERE D > 5"` would return all columns, but only for players whose Rebounds (Column D) statistic exceeds 5. This combination maximizes both breadth (all columns) and depth (filtered rows) of data retrieval.

Understanding when to use explicit selection versus the wildcard is a key decision point in query optimization. Explicit selection is robust against changes outside the selected columns and is faster for very wide sheets where most columns are irrelevant. The wildcard, while convenient for completeness and maintenance, is best suited when almost all column data is required or when the structure of the data range is expected to expand frequently. In both cases, the SELECT clause remains the defining command.

We can use the following syntax to select *all* of the columns in the dataset:

E1 fx =query(A1:C12, "select *", 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		Bob	Mavericks	7.8
4	Carl	Spurs	13.7		Carl	Spurs	13.7
5	Dave	Warriors	22.3		Dave	Warriors	22.3
6	Eric	Mavericks	27.8		Eric	Mavericks	27.8
7	Fred	Mavericks	20.8		Fred	Mavericks	20.8
8	George	Spurs	12.7		George	Spurs	12.7
9	Harold	Lakers	8.2		Harold	Lakers	8.2
10	Isaiah	Warriors	12.5		Isaiah	Warriors	12.5
11	Joe	Warriors	30.2		Joe	Warriors	30.2
12	Ken	Spurs	22.4		Ken	Spurs	22.4
13							
14							
15							
16							
17							
18							
19							
20							

Best Practices for Column Referencing

When constructing QUERY statements, one critical element is how you reference the columns. In standard Google Sheets usage, columns are typically referenced using their letter identifiers (A, B, C). However, confusion can arise if the source data range provided in the first argument does not start at column A. For example, if your data range is defined as `B2:E100`, the column labeled B in the spreadsheet is actually the first column (Col1) in the query's defined dataset, column C is the second (Col2), and so on.

To maintain clarity and avoid potential errors, especially when dealing with data ranges that are subsets of the entire sheet, many advanced users opt to use the functional reference notation: **Col1, Col2, Col3**, etc. This method ensures that the column reference is relative to the starting point of the data range defined in the first argument. If your range is `C1:F50`, then C is Col1, D is Col2, E is Col3, and F is Col4. Using ColX notation is highly recommended when the query function itself is placed on a different sheet or when the data range is complex, promoting formula stability and easier debugging.

Another important best practice relates to handling column headers, which is controlled by the third argument (the header count). If this argument is set incorrectly (e.g., set to 0 when headers exist), the query might treat the header row as data, leading to calculation errors or improper data types in the output. Conversely, setting the header count too high can result in valuable data being

misinterpreted as headers and omitted from the results. Always verify the structure of your source data and ensure the header count accurately reflects the number of descriptive rows at the top of your dataset.

Summary and Advanced Selection Techniques

The ability to use the **SELECT** clause to retrieve multiple columns, either explicitly by listing identifiers (A, B, C) or implicitly using the wildcard (*****), forms the foundation of all effective data manipulation using the Query function. This capability allows users to transform raw, wide datasets into highly specialized, narrow reports, filtering out noise and focusing on key performance indicators or specific analytical needs.

Beyond simple selection, the **SELECT** clause is also capable of performing on-the-fly calculations and aggregations. For example, you can select column A (Player Name) and then select the average of column C (Points) using the formula `"SELECT A, AVG(C) GROUP BY A"`. This demonstrates that the columns selected are not merely static representations of the source data but can be dynamic results of computational functions, significantly extending the analytical utility of the spreadsheet environment.

In conclusion, mastering multiple column selection, combined with conditional filtering using **WHERE**, provides spreadsheet users with database-level capabilities. By adhering to the precise syntax and understanding the relationship between the data range, the query string, and the header count, complex data extraction tasks become straightforward and fully automated, maintaining dynamic updates whenever the source information is modified. Continuous practice with explicit column selection and conditional clauses will unlock the full potential of the Google Sheets Query function for advanced data management.