Learning Google Sheets QUERY: Selecting Multiple Columns for Data Analysis

Authored by

Mohammed looti

November 4, 2025

The Google Sheets environment offers robust tools for data analysis, but none are quite as transformative as the **`QUERY` function**. This function empowers users to perform sophisticated data retrieval and manipulation tasks by leveraging a command structure closely modeled after standard SQL (Structured Query Language). Understanding how to harness this power is crucial for anyone managing large volumes of information.

In data management, the ability to isolate and analyze specific components of a larger dataset is paramount. This detailed guide focuses specifically on the essential skill of selecting multiple, non-contiguous columns efficiently. We will break down the precise syntax required to utilize the **`QUERY` function** for targeted column extraction, enabling you to create clean, focused reports quickly and accurately.

Deconstructing the QUERY Function Syntax

The structure of the **`QUERY` function** is defined by three fundamental components, each playing a critical role in data retrieval: the source range, the query string, and the header specification. Effective data extraction hinges entirely upon mastering the construction of the central element--the query string, which houses the SQL-like commands.

When the objective is to select multiple columns, the process involves listing the specific column identifiers (e.g., A, B, C) directly following the mandatory **`SELECT`** clause. These identifiers must be separated by commas within the query string. This method allows for precise targeting of the desired fields, regardless of their position within the source data range.

The standard syntax pattern used for selecting multiple columns is illustrated below. The `Range` defines the boundaries of the source data, while the `Query_String` dictates the instructions for selection and filtering:

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

Crucially, the final argument, the numerical value **`1`**, informs the function about the presence of a header row within the defined range. Specifying `1` ensures that the top row is correctly recognized and used as labels for the output columns, rather than being treated as raw data. If your dataset lacks labels, this argument should be set to `0`.

Structuring Your Data for Optimal Query Performance

Prior to running any data manipulation command, successful execution of the **`QUERY` function** relies heavily on the quality and consistency of the source data. It is imperative that your data is clean and uniformly structured, particularly regarding column data types. For instance, a column designated for statistics must exclusively contain numerical values, while a name column should

contain only text strings. Mixed data types within a single column often lead to unpredictable or erroneous query results.

Throughout the subsequent examples, we will reference a sample sports dataset. This dataset includes essential player metrics, such as names, team affiliations, and performance statistics like Points, Assists, and Rebounds. The data spans the range **A1:E7** in our worksheet, providing a clear reference point for all column identifiers.

By visually inspecting the source data, as shown below, you can establish a definitive mapping between the data fields (Player, Team, Stats) and their corresponding column identifiers (A, B, C, D, E), which are indispensable for writing accurate query strings.

	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: Retrieving Specific Columns Without Filtering

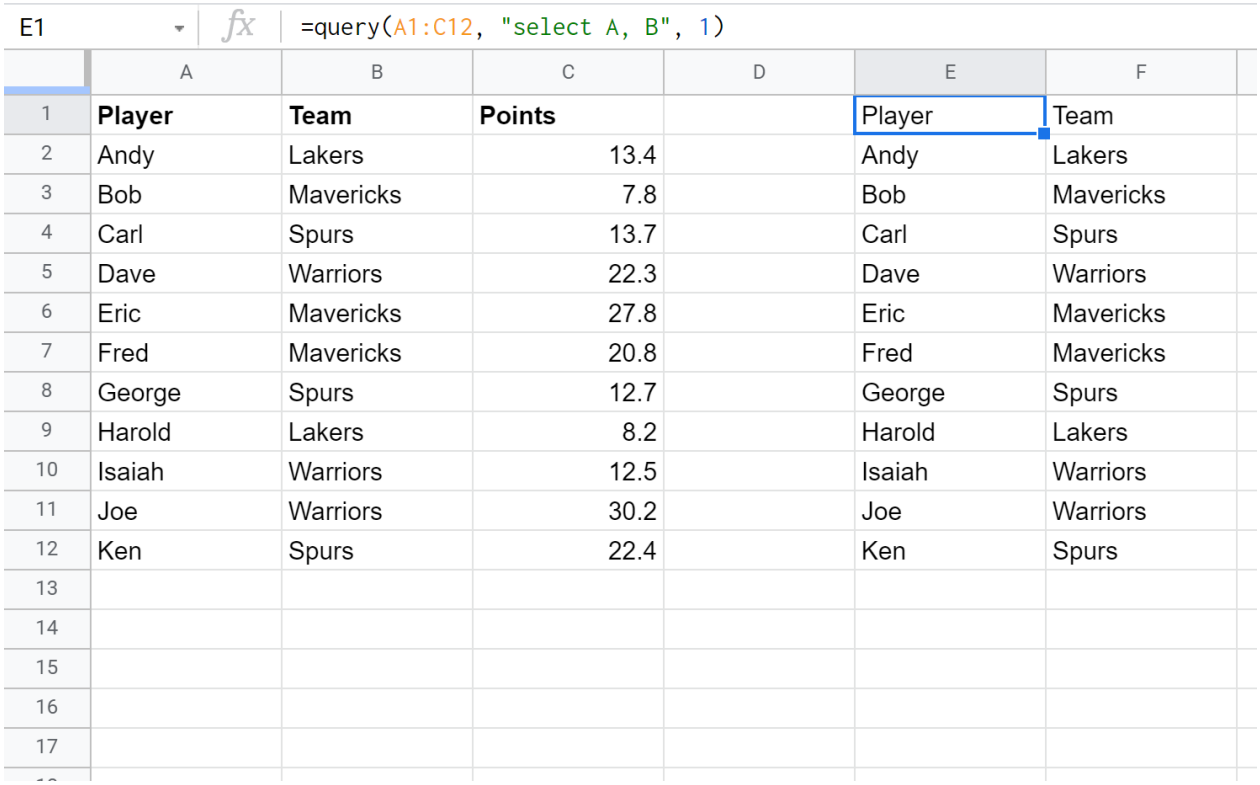
The primary application of the **`QUERY` function** for column selection is its simplest form: retrieving a specific subset of columns without applying any constraints, grouping, or aggregation. This technique is indispensable for quickly generating focused reports or preparing data for visualization, where the full breadth of the source data is unnecessary.

Consider the requirement to extract only the **Player** names (Column A) and their respective **Team** affiliations (Column B). The goal is to return every row available in the range A1:E7, but only for these two specified fields. This requires explicitly listing the column identifiers in the `SELECT` statement.

The necessary query syntax is concise, utilizing the `SELECT` clause followed immediately by the required column identifiers, separated by a comma:

=QUERY(A1:E7, "SELECT A, B", 1)

Executing this command directs the spreadsheet to analyze the data from A1 to E7, extract and display only the data contained within column A and column B, and correctly identify the first row as the header row for the results table.



	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: Combining Selection with Filtering (The WHERE Clause)

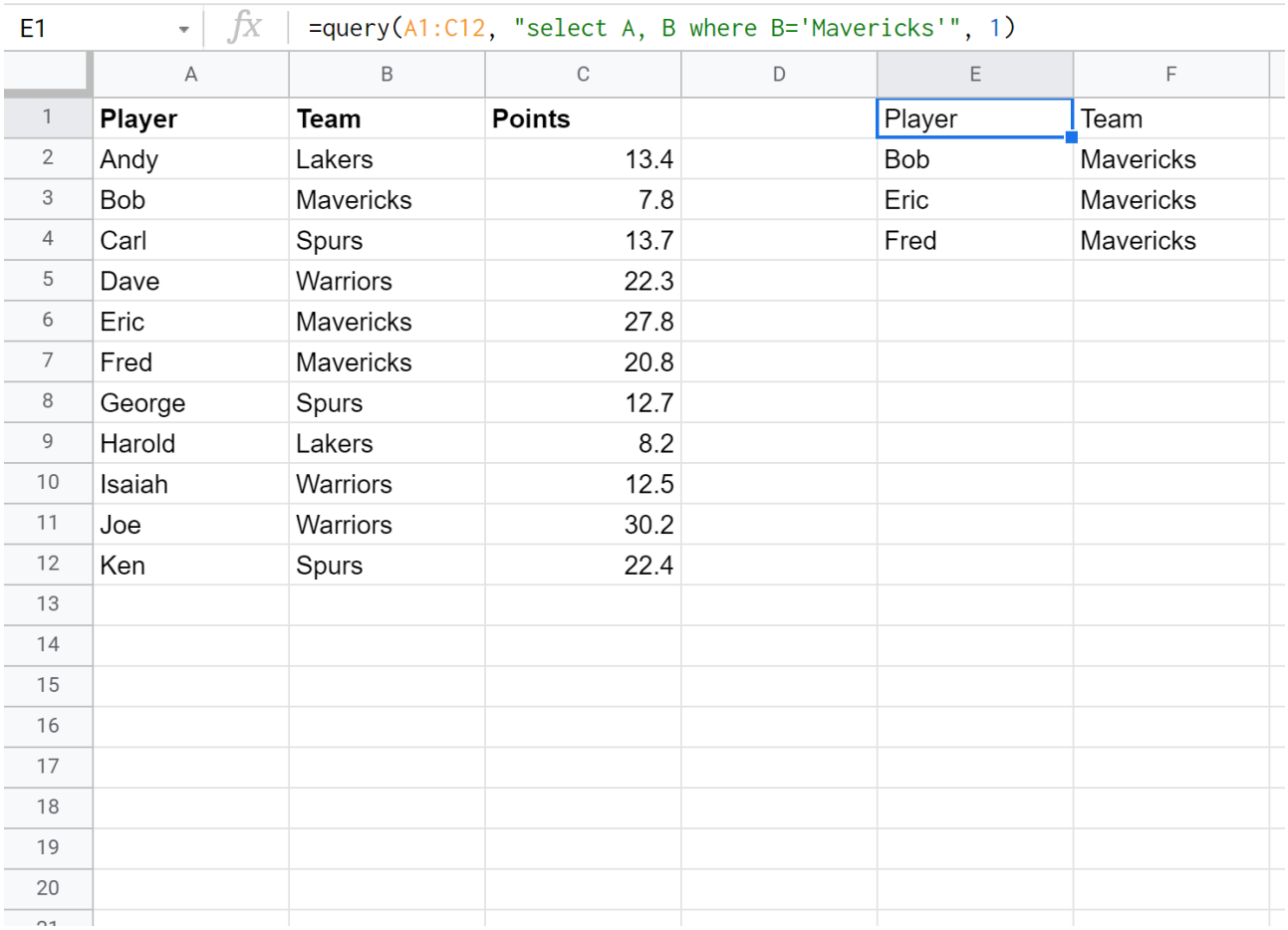
To move beyond simple data subsetting, the **`QUERY` function** allows for dynamic filtering through the use of the **`WHERE` clause**, an essential component borrowed from SQL. This clause enables users to specify conditions that must be met for a row to be included in the final output, dramatically enhancing the function's analytical capabilities.

Let's refine our previous selection. We still require the **Player** (A) and **Team** (B) columns, but now we must filter the results to display only those players associated with the "Mavericks" team. This requires appending the WHERE clause, targeting Column B to check for the exact value.

A crucial rule to remember when implementing the **`QUERY` function** is the handling of text strings: the entire query string must be enclosed in double quotes, while any literal text value used for comparison (like 'Mavericks') must be enclosed in single quotes. Failing to follow this nesting convention will result in a formula error. The resulting precise syntax is:

=QUERY(A1:E7, "SELECT A, B WHERE B = 'Mavericks'", 1)

This powerful command not only limits the output fields to columns A and B but also executes a simultaneous filter, ensuring that only records where the value in column B is precisely "Mavericks" are returned, thereby providing a highly focused, conditional report.



	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						

Example 3: Utilizing the Wildcard Operator for Full Selection

Although the primary focus of efficient querying is often narrowing down fields, sometimes the

requirement is to retrieve every column present in the source range. Listing dozens of column identifiers manually is tedious and prone to error. Fortunately, the **`QUERY` function** supports the use of the wildcard operator, symbolized by the asterisk (`*`), to simplify this task.

The command **`SELECT *`** serves as a powerful shorthand, instructing the function to return all data points within the specified range (A1:E7). This approach is frequently employed during initial data exploration, debugging complex filters, or when the objective is to duplicate the entire dataset into a new tab for archival or manipulation purposes.

The resulting syntax for selecting all columns is exceptionally clean and intuitive:

=QUERY(A1:E7, "SELECT *", 1)

Upon execution, this function extracts all five columns--Player, Team, Points, Assists, and Rebounds--along with every row of corresponding data. It correctly processes the range A1:E7 and acknowledges the first row as the descriptive header row, ensuring the output is perfectly structured.

E1 =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 and Expanding Your Query Capabilities

Successful and efficient utilization of the **`QUERY` function** hinges on meticulous attention to syntax

and a solid understanding of data structure. Establishing consistent best practices ensures that your queries execute reliably, regardless of the complexity of the underlying data.

To avoid common errors and maximize the accuracy of your multi-column selections, keep the following essential guidelines in mind:

Referencing Conventions: Always use the column letters (A, B, C, etc.) that correspond to the columns within the defined **source range** (the first argument, e.g., A1:E7). Never confuse these references with the actual column letters of the sheet where the resulting formula is placed.
Precise Separation: Within the `SELECT` clause, column identifiers must be separated strictly by single commas, with no additional operators or punctuation (e.g., `"SELECT A, C, E"` is correct).
Header Row Integrity: The third argument, which specifies the number of header rows (usually `1`), is non-negotiable for correct output formatting. If your range starts at the first row and includes labels, use `1`. If you fail to account for a header, the first row of valuable data will be mistakenly promoted and displayed as a column label. Conversely, specifying `1` when no header exists will incorrectly categorize the first data entry.

The mastery of selecting and filtering columns transforms Google Sheets from a basic tool into a high-caliber data manipulation engine. We strongly recommend continuing to explore advanced SQL clauses supported by the **`QUERY` function**, such as the `ORDER BY` clause for dynamic sorting, or the powerful `GROUP BY` clause for complex data aggregation, to fully realize the potential of spreadsheet data analytics.

For more comprehensive Google Sheets tutorials covering advanced data analysis and manipulation techniques, please refer to the resources linked throughout this library.