

How to Rename Columns in Google Sheets Queries with the SQL LABEL Clause

Authored by
stats writer

December 3, 2025

RECOMMENDED CITATION

stats writer (2025). *How to Rename Columns in Google Sheets Queries with the SQL LABEL Clause*. PSYCHOLOGICAL SCALES. Retrieved from <https://scales.arabpsychology.com/?p=104074>

The [Google Sheets Query](#) function utilizes a powerful language based on [SQL](#) syntax, enabling users to filter, sort, and aggregate data directly within a spreadsheet environment. Among its many capabilities, the **Label clause** stands out as a critical tool for enhancing the readability and professional presentation of query results. This clause allows users to assign a descriptive name, often referred to as an [alias](#), to any column derived from the data range.

Effective data presentation is paramount, especially when sharing analysis with stakeholders who may not be familiar with the original dataset structure. Raw column headers, such as `COL1`, `C`, or abbreviated field names like `Pts_Scrd`, are often ambiguous. By employing the [Label clause](#), you can transform these cryptic headers into meaningful titles like 'Employee Name' or 'Total Points Scored,' ensuring immediate clarity for the end-user reviewing the queried output.

Furthermore, the utility of the **Label clause** extends beyond mere aesthetics. In complex queries involving statistical aggregations--such as counting unique entries, calculating averages, or summing values--the resulting columns often receive default, non-descriptive names (e.g., `sum(B)` or `count(A)`). The inability to quickly identify what these aggregated columns represent can severely hinder data interpretation. The intentional use of the [Label clause](#) resolves this issue by providing immediate, context-specific column titles, thereby streamlining the overall process of data analysis and reporting.

You can use the **label** clause in a Google Sheets query to create specific labels for the results of certain columns.

Understanding the Role of the Label Clause in Data Presentation

The primary function of the [Label clause](#) within the [Google Sheets Query](#) function is to enable column aliasing for improved output clarity. Unlike the `SELECT` clause, which dictates which columns are returned, or the `WHERE` clause, which handles filtering, the `LABEL` clause specifically targets the presentation layer. It must be placed after the selection criteria but before optional clauses like `LIMIT` or `OFFSET`.

When working with large datasets, maintaining consistency and clarity is a major challenge. Imagine a scenario where a master sheet uses single-letter column headers (A, B, C) for simplicity, but the generated report needs professional titles. Instead of manually editing the output headers every time the query runs, the **Label clause** automates this process. This not only saves time but drastically reduces the potential for human error, ensuring that every generated report is standardized and easily understood by its intended audience, regardless of the underlying complexity of the [SQL](#) statement used.

A frequent challenge arises when combining data using aggregation functions like `SUM`, `AVG`, or `COUNT`. By default, these functions generate complex header names that reflect the operation itself,

such as `AVG(Co13)`. While accurate from a database perspective, these headers are cumbersome for reporting. The use of an alias via the **Label clause** allows the transformation of `AVG(Co13)` into a simple, descriptive header like 'Average Price,' thereby optimizing the results for business intelligence and reporting purposes.

Syntax Fundamentals: Implementing the Label Clause

The structure for incorporating the **Label clause** into the Google Sheets Query function is straightforward, yet precise. It requires specifying the column identifier, followed by the keyword `LABEL`, and then the desired new column header enclosed in single quotes. This clause is typically the last required element of the query string before optional formatting or limiting clauses.

When implementing the query, it is essential to remember that the entire query statement resides within double quotes inside the `QUERY` function. The syntax for applying a label to a single column is structured as follows. Note that the column identifier (e.g., `A`) refers to the column in the source data range (`A1:C13`):

```
=QUERY(A1:C13, "select * label A 'Column A'")
```

In this fundamental example, we utilize `SELECT *` to return all columns from the defined range (`A1:C13`). Crucially, the `LABEL` instruction immediately follows, targeting column `A` and assigning it the new header 'Column A' in the resulting output table. This method demonstrates the most basic application of renaming a standard, non-aggregated column.

To label multiple columns within a single query, you simply chain the column identifier, the `LABEL` keyword, and the new header using commas to separate the definitions. This provides a clean and compact way to manage aliasing for all relevant columns simultaneously:

```
=QUERY(A1:C13, "select * label A 'A Column', B 'B Column'")
```

Note that when labeling multiple columns, the `LABEL` keyword is only used once at the beginning of the clause, followed by the comma-separated list of column-label pairs. This structure maintains efficiency and readability, ensuring that the intent of the query--to rename columns A and B--is perfectly clear.

Practical Application: Labeling a Single Column

To fully grasp the practical utility of the **Label clause**, let us examine a specific scenario involving sports data. Suppose we have a dataset containing columns for Team, Points, and Rebounds. While 'Team' is descriptive, for a formal report, we prefer the header 'Team Name'. This renaming

process is achieved seamlessly using the Label clause, ensuring that only the targeted column header is modified in the final output.

The goal of this operation is to select all three columns (Team, Points, Rebounds) but apply an alias specifically to the 'Team' column. Assuming 'Team' corresponds to Column A in the source range, the query formula should look like this: `=QUERY(A1:C, "SELECT A, B, C LABEL A 'Team Name' ")`. This demonstrates how the `SELECT` statement identifies the columns to return (A, B, C), and the `LABEL` statement modifies the presentation of column A.

Upon execution of the query, the results clearly illustrate the intended outcome. The data associated with Columns B (Points) and C (Rebounds) retain their original headers because they were not explicitly targeted by the **Label clause**. Only Column A's header is transformed into 'Team Name,' proving the selective precision of the aliasing function. This level of control is essential for maintaining accuracy while simultaneously improving presentation quality.

	A	B	C	D	E	F	G
E1	<code>=QUERY(A1:C13, "select * label A 'Team Name'")</code>						
1	Team	Points	Rebounds		Team Name	Points	Rebounds
2	Mavs	96	30		Mavs	96	30
3	Nets	93	22		Nets	93	22
4	Hawks	94	28		Hawks	94	28
5	Heat	94	25		Heat	94	25
6	Magic	99	25		Magic	99	25
7	Spurs	105	26		Spurs	105	26
8	Rockets	103	28		Rockets	103	28
9	Hornets	95	33		Hornets	95	33
10	Suns	93	31		Suns	93	31
11	Bucks	90	30		Bucks	90	30
12	Warriors	88	36		Warriors	88	36
13	Lakers	91	24		Lakers	91	24
14							
15							
16							
17							
18							
19							

Advanced Usage: Labeling Multiple Columns Simultaneously

When restructuring a report or optimizing data for consumption, it is often necessary to rename several columns at once. The **Label clause** is efficiently designed to handle multiple aliasing operations within a single, continuous statement. This prevents the need for separate query functions or complex array formulas, maintaining the integrity and conciseness of the SQL-like query.

Continuing with the sports data example, suppose we wish to rename both the 'Team' column to 'Team Name' and the 'Points' column to 'Points Scored'. If 'Team' is Column A and 'Points' is Column B, the command structure involves listing both aliasing definitions separated only by a comma, following the initial `LABEL` keyword. The resulting formula is concise and powerful: `=QUERY(A1:C, "SELECT A, B, C LABEL A 'Team Name', B 'Points Scored'")`. This simultaneous renaming capability is crucial for generating comprehensive and fully customized reports.

This implementation showcases the flexibility of the syntax. We have successfully renamed two specific columns (A and B) while leaving the remaining selected column (C, Rebounds) untouched. This illustrates that the **Label clause** can selectively target columns regardless of their position within the `SELECT` statement. The key takeaway is that the syntax remains identical whether you are renaming two columns or ten: the definitions are simply appended to the list, separated by commas.

	A	B	C	D	E	F	G
E1	<code>=QUERY(A1:C13, "select * label A 'Team Name', B 'Points Scored'")</code>						
1	Team	Points	Rebounds		Team Name	Points Scored	Rebounds
2	Mavs	96	30		Mavs	96	30
3	Nets	93	22		Nets	93	22
4	Hawks	94	28		Hawks	94	28
5	Heat	94	25		Heat	94	25
6	Magic	99	25		Magic	99	25
7	Spurs	105	26		Spurs	105	26
8	Rockets	103	28		Rockets	103	28
9	Hornets	95	33		Hornets	95	33
10	Suns	93	31		Suns	93	31
11	Bucks	90	30		Bucks	90	30
12	Warriors	88	36		Warriors	88	36
13	Lakers	91	24		Lakers	91	24
14							
15							
16							
17							
18							

Integrating Label with Other Query Clauses (Selection and Filtering)

The true power of the **Label clause** emerges when it is combined with other core components of the Google Sheets Query function, such as `WHERE`, `GROUP BY`, and aggregation functions. It is crucial to understand the order of operations in the query language. The `LABEL` clause is applied only after the data has been filtered (`WHERE`) and grouped/aggregated (`GROUP BY`), but before the final output formatting (`FORMAT`).

One of the most valuable uses is applying labels to aggregated data. Consider a scenario where you want to find the total points scored per team. The query would use `GROUP BY` and `SUM(B)`. By default, the resulting column header for the summation would be 'sum Points' or 'sum(B)'. To make this report-ready, we must apply a label directly to the aggregated column identifier. The correct syntax would be: `SELECT A, SUM(B) GROUP BY A LABEL SUM(B) 'Total Points Accumulated'`. Notice that the label targets the function itself, `SUM(B)`, not the original column B.

When filtering data using the `WHERE` clause, the column identification within the `WHERE` clause must always refer to the original column header (or letter index), not the new alias created by the **Label clause**. This is because filtering occurs logically before the final presentation layer where the labeling takes effect. Maintaining this distinction is vital for writing functionally correct and logically sound queries in Google Sheets.

For instance, to filter for teams with more than 50 points and then label the column: `SELECT A, B WHERE B > 50 LABEL B 'High Scorer Points'`. If you attempt to use the new label ('High Scorer Points') in the `WHERE` clause, the query will fail because that label does not exist in the source data during the filtering stage.

Best Practices for Effective Column Labeling

To maximize the benefits of using the **Label clause**, adherence to specific best practices is recommended. These practices ensure that your formulas are maintainable, readable, and perform reliably, especially as the complexity of your data models increases.

Use Descriptive Names: Avoid generic names like 'New Column 1'. Labels should clearly describe the data they contain, facilitating rapid understanding. If the original column was `Date_of_Entry`, a good label might be 'Submission Date (YYYY-MM-DD)'.

Consistency in Quotes: Always enclose the label text in **single quotes** ('New Label'). Using double quotes or no quotes around the label text will inevitably lead to syntax errors within the query string.

Label Aggregated Columns: Always apply the **Label clause** to columns resulting from aggregation (e.g., `AVG(C)`, `COUNT(A)`). Failure to do so results in default, often overly verbose, headers that undermine the clarity of the report.

Place Label Clause Last: For standard queries, the `LABEL` clause should typically be the last command before the closing double quote of the query string, preceding only optional clauses like `LIMIT`, `OFFSET`, or `FORMAT`. The required order of operations for SQL-like languages dictates that labeling occurs near the end of the execution sequence.

Handle Special Characters: If your desired label contains special characters, such as commas, parentheses, or multiple spaces, the use of single quotes is mandatory to enclose the entire string and treat it as a single unit.

Adopting these guidelines ensures that your reports are not just functional, but also provide a professional and polished representation of the analyzed data, improving overall efficiency in data exchange and reporting across teams.

Troubleshooting Common Label Clause Errors

While the **Label clause** is simple to implement, users sometimes encounter common errors related to syntax or placement within the overall query structure. Identifying and addressing these issues quickly is key to successful implementation of the [Google Sheets Query function](#).

One frequent error involves incorrect quotation marks. Since the entire query string is enclosed in double quotes ("..."), the actual column label itself must be enclosed in single quotes ('...'). Using double quotes for the label (e.g., "LABEL A "Team Name"") will prematurely terminate the primary query string, leading to a parsing error. Similarly, omitting quotes entirely will cause the query processor to interpret the label text as an unintended function or reserved keyword.

Another prevalent issue is misidentifying the column to be labeled, especially when dealing with aggregated data. If you write `SELECT A, SUM(B) ... LABEL B 'Total'`, the label will not apply correctly because column B no longer exists as a standalone column after the `SUM` aggregation has occurred. You must target the aggregated identifier: `LABEL SUM(B) 'Total'`. Always ensure that the column identifier you are labeling exactly matches the identifier used in the `SELECT` statement, whether it is a simple column letter or a complex aggregation function.

Finally, improper clause ordering is a common pitfall. Placing the **Label clause** before necessary clauses like `WHERE` or `GROUP BY` will result in a syntax error because the query engine expects the clauses in a specific sequence (e.g., `SELECT, WHERE, GROUP BY, ORDER BY, LABEL`). Always verify that the `LABEL` clause is positioned correctly towards the end of your query string to ensure proper execution.

Further Exploration of Google Sheets Query Capabilities

The ability to rename columns using the **Label clause** is just one facet of the extensive capabilities offered by the [Google Sheets Query function](#). Mastering the structure and application of this function opens up advanced possibilities for dynamic data manipulation and reporting, far exceeding the standard filtering and sorting tools available directly in the spreadsheet interface.

By effectively controlling the output format and column headers, users can generate highly sophisticated dashboards and automated reports that are ready for immediate consumption without post-processing. This mastery is particularly important for professionals dealing with large volumes of data who require precise control over both the content and the presentation layer.

We highly recommend exploring related tutorials to enhance your command of this powerful tool. The following operations represent common advanced techniques used in conjunction with column labeling to produce polished data summaries:

[Google Sheets Query: How to Use Group By](#)

ARABPSYCHOLOGY.COM