How to Extract the First N Characters from a String in MySQL

Authored by

mohammed looti

January 5, 2026

Efficient MySQL database management often requires precise manipulation of stored text data. Selecting a specific segment, or string, from a larger column value is a fundamental operation in SQL, particularly when standardizing codes, creating abbreviations, or generating summarized reports. Although several functions are available for this purpose, two primary and highly effective methods exist for extracting the first N characters of a text field: the versatile SUBSTRING() function and the specialized LEFT() function. Understanding the nuance of these commands is essential for database professionals seeking clarity and performance in their queries.

The core challenge addressed by these functions is the need to truncate data dynamically without losing context. Imagine a scenario where a database column holds full names of sports teams, but an accompanying report requires only a standardized four-letter abbreviation. Manual truncation is impractical for large datasets; thus, relying on built-in MySQL string handling capabilities becomes necessary. These methods ensure that the extraction process is both accurate and scalable, adhering to the principle that character positions are indexed starting from 1 in standard SQL environments, which is crucial when defining the starting point for character selection.

Methods for Extracting Leading Characters in MySQL

To summarize, there are two standard and widely accepted methods within the MySQL environment for selecting the first N characters of a given string. While both achieve the same result, they differ slightly in syntax and intended purpose, making one potentially more intuitive than the other depending on the context of the SQL query being constructed. Both rely on robust, native MySQL functions designed specifically for string manipulation tasks.

We will explore both syntaxes in detail, but here is a quick overview demonstrating how to extract the initial four characters from a column named `team` within a table named `athletes`:

Method 1: Utilize the LEFT() Function

The LEFT() function is arguably the cleanest approach when the goal is strictly to extract characters starting from the very beginning of the string. It simplifies the syntax by requiring only the target string and the desired length, implicitly assuming the start position is the first character (position 1).

SELECT LEFT(team, 4) FROM athletes;

Method 2: Utilize the SUBSTRING() Function

The SUBSTRING() function offers greater versatility, as it allows extraction from any point within the string. To select the first N characters, the starting position must be explicitly set to 1. This

function is beneficial when the querying requirements might later expand to selecting characters from the middle of a string.

SELECT SUBSTRING(team, 1, 4) FROM athletes;

Both of these examples successfully extract the first 4 characters of the values found in the **team** column of the table named **athletes**, providing identical results in this specific use case.

Deep Dive into the LEFT() Function Syntax

The LEFT() function is fundamentally designed for simplicity and readability when performing front-end truncation. Its syntax is streamlined, requiring only two arguments: the column or string expression you wish to manipulate, and the integer representing the number of characters to retain from the left side. This function is a highly performant and intuitive choice for routine abbreviation tasks.

The general syntax is `LEFT(str, len)`, where `str` is the input string and `len` is the positive integer count of characters to extract. If `len` is specified as zero, the result will be an empty string, and if `len` is negative, the result is null. If the length requested (N) exceeds the actual length of the string, the function simply returns the entire original string without error, maintaining robustness in the query execution.

While straightforward, it is important to remember that the LEFT() function is limited exclusively to extracting characters from the beginning. It cannot be used to select characters starting from position 5 or position 10, unlike its more flexible counterpart, SUBSTRING(). Therefore, `LEFT()` should be chosen when the specific requirement is always fixed at the initial character position.

Understanding the Versatility of SUBSTRING()

In contrast to the specialized nature of `LEFT()`, the SUBSTRING() function (often interchangeable with `MID()` or `SUBSTR()` in SQL environments) is built for comprehensive extraction from any part of a string. When using `SUBSTRING()` to achieve the same result as `LEFT()`--extracting the first N characters--we must explicitly define the starting position.

The core syntax for the three-parameter version is `SUBSTRING(str, pos, len)`. Here, `str` is the input string, `pos` is the starting position (which must be 1 for extracting the first N characters), and `len` is the number of characters to extract from that starting point. Specifying `pos=1` effectively instructs the function to begin counting immediately from the start of the string.

This function's inherent flexibility makes it a powerful tool for complex string analysis. For instance, if you later need to extract characters 5 through 8, you would simply change the `pos` argument to

5. When used to select the first N characters, however, the explicit inclusion of the '1' parameter makes the command slightly more verbose than `LEFT()`, though functionally identical for this specific task.

Practical Demonstration: Setting up the Sample Dataset

To illustrate the functionality of both methods, we will use a hypothetical dataset representing basketball player statistics. This example ensures that the application of the functions is clear and reproducible. The following code demonstrates the creation of the table named **athletes** and the insertion of sample rows, providing a clear foundation for our string extraction queries.

The dataset includes varying team names, some capitalized, some lowercase, and some mixed case, which is typical of real-world data and demonstrates how these string functions handle different text inputs uniformly.

```
-- create table
CREATE TABLE athletes (
id INT PRIMARY KEY,
team TEXT NOT NULL,
position TEXT NOT NULL,
points INT NOT NULL
);
```

```
-- insert rows into table
INSERT INTO athletes VALUES (0001, 'grizzlies', 'Guard', 15);
INSERT INTO athletes VALUES (0002, 'mavericks', 'Guard', 22);
INSERT INTO athletes VALUES (0003, 'CAVALIERS', 'Forward', 36);
INSERT INTO athletes VALUES (0004, 'Spurs', 'Guard', 18);
INSERT INTO athletes VALUES (0005, 'hawKs', 'Forward', 40);
INSERT INTO athletes VALUES (0006, 'nets', 'Forward', 25);

-- view all rows in table
SELECT * FROM athletes;
```

Output: The complete table structure and data, as confirmed by the final `SELECT *` statement, are shown below. Note the varying lengths of the team names.

```
+----+-----------+----------+--------+
| id | team | position | points |
+----+-----------+----------+--------+
| 1 | grizzlies | Guard | 15 |
```

| 2 | mavericks | Guard | 22 |
| 3 | CAVALIERS | Forward | 36 |
| 4 | Spurs | Guard | 18 |
| 5 | hawKs | Forward | 40 |
| 6 | nets | Forward | 25 |
+----+-----------+----------+--------+

Executing Extraction using the LEFT() Function

We will now apply the **LEFT** function to extract the first four characters from the **team** column. This process simulates generating a four-letter team abbreviation for reporting purposes. Since the function is designed to handle this specific task, the query is concise and highly readable, clearly communicating the intent to truncate the string from the left.

This method is highly favored by developers who prioritize short, explicit syntax for simple substring operations starting at position 1. Observe how the mixed-case nature of the original data (e.g., 'grizzlies', 'CAVALIERS', 'hawKs') is fully preserved in the resulting output, as string functions in MySQL typically operate based on byte position without altering case unless explicitly instructed by other functions like `UPPER()` or `LOWER()`.

SELECT LEFT(team, 4) FROM athletes;

Output: The result set confirms that only the leading four characters have been successfully extracted from each team name.

+---------------+
| LEFT(team, 4) |
+---------------+
| Griz |
| Mave |
| Cava |
| Spur |
| Hawk |
| Nets |
+---------------+

It is evident that the output contains only the first four letters of each string in the **team** column. Note that the output column name defaults to the function call itself (`LEFT(team, 4)`), which can often be cumbersome for use in application layers or complex joins, prompting the need for an

alias.

Executing Extraction using the SUBSTRING() Function

Next, we demonstrate the functionally equivalent query using the SUBSTRING() function. By setting the second parameter (the starting position) explicitly to 1 and the third parameter (the length) to 4, we achieve the identical output as the `LEFT()` function. This confirms that for extracting the first N characters, `LEFT(str, N)` is merely a syntactic sugar for `SUBSTRING(str, 1, N)`.

Understanding this equivalence is important for maintaining compatibility and consistency, especially when migrating SQL code between different database systems where the availability of `LEFT()` might vary, although `SUBSTRING()` is nearly universally supported.

SELECT SUBSTRING(team, 1, 4) FROM athletes;

Output: As expected, the results are identical to the previous query.

+----------------------+
| SUBSTRING(team, 1, 4) |
+----------------------+
| Griz |
| Mave |
| Cava |
| Spur |
| Hawk |
| Nets |
+----------------------+

Enhancing Readability with Column Aliases

As observed in the previous outputs, the default column names resulting from function calls are descriptive but often overly long and difficult to integrate into subsequent programming logic. To solve this, we can use the **AS** keyword to assign an alias to the derived column, dramatically improving readability and usability.

By defining an alias, we provide a meaningful, simplified name for the extracted column, such as `first_four`. This practice is standard in professional SQL query writing and applies equally well to both `LEFT()` and `SUBSTRING()` functions. Below is an example applying an alias using the `SUBSTRING()` function.

SELECT SUBSTRING(team, 1, 4) AS first_four FROM athletes;

Output: The column header is now clean and descriptive.

```
+------------+
| first_four |
+------------+
| Griz |
| Mave |
| Cava |
| Spur |
| Hawk |
| Nets |
+------------+
```

Notice that the column name in the output is now **first_four**, which is much easier to read and reference in application code or further complex queries. Adopting the use of aliases is a critical step in mastering efficient and maintainable database querying in MySQL.

Further Resources and String Manipulation Techniques

Mastery of string functions extends beyond simple extraction. SUBSTRING() and LEFT() are foundational, but MySQL offers a suite of related functions for more complex tasks, such as padding, concatenation, and pattern matching.

For those interested in advancing their string manipulation skills, consider exploring the following related topics and functions:

RIGHT() Function: Selects the last N characters of a string.

LOCATE() and INSTR(): Used to find the position of a substring within a larger string, often combined with `SUBSTRING()` to extract text based on delimiters (e.g., extracting a first name before a space).

CONCAT(): Joins multiple strings or column values together.

TRIM(): Removes leading or trailing spaces from a string, ensuring cleanliness before extraction or comparison.

The ability to efficiently handle and transform text data is indispensable for accurate reporting and data normalization within any relational database system.