How to Concatenate MySQL Rows with Commas

Authored by

mohammed looti

January 5, 2026

One of the most common data manipulation tasks in database management involves aggregating information from multiple records into a single, summary field. In MySQL, achieving this cross-row concatenation, especially when separating the results with commas, requires a specialized aggregate function. While simple string joining across columns utilizes the standard CONCAT() function, grouping and concatenating values vertically across rows demands the use of GROUP_CONCAT().

This powerful function is specifically designed to combine non-NULL values from a group into a single string result. This capability is exceptionally useful for generating reports, creating quick summary lists, or preparing data for output formats that require a comma-separated list (CSV-like structure) within a single field. Understanding how to properly implement this function, especially in conjunction with the GROUP BY clause, is essential for advanced data retrieval.

Understanding Multi-Row Concatenation in MySQL

When working with relational databases, data is distributed across many rows based on specific entities or attributes. Often, we need to transform this vertical data structure into a horizontal, summarized view. For instance, if you have a list of products and their associated categories spread across multiple rows, you might want a single row for the product, listing all its categories separated by commas. This process is known as string aggregation or multi-row concatenation.

The standard SQL language does not provide a universal, cross-database function for this task, which is why various database systems implement their own solutions. MySQL's implementation, GROUP_CONCAT(), is highly flexible and provides built-in mechanisms for defining separators, ordering the concatenated elements, and managing duplicates.

In the context of comma separation, this method eliminates the need for complex subqueries or application-side processing to achieve the desired output format. Instead, the aggregation is handled efficiently at the database level, returning clean, summarized data ready for display or further processing. This technique greatly simplifies reporting queries that require one-to-many relationships to be collapsed into a single field.

The Power of GROUP_CONCAT()

The core mechanism for concatenating rows in MySQL is the `GROUP_CONCAT()` function. Unlike the basic CONCAT() function, which joins strings or column values horizontally within a single row, `GROUP_CONCAT()` operates vertically across groups of rows defined by the GROUP BY clause. It is classified as an aggregate function because it summarizes data from many input rows into a single output row.

The function takes one or more column expressions as arguments, processes all values within the

specified group, and returns them combined as one string. By default, GROUP_CONCAT() uses a comma and a space (`, `) as the separator between the concatenated values. However, for strict comma-separated output, we can override this default behavior using the `SEPARATOR` keyword.

This functionality is crucial when transforming datasets. For example, if you are tracking athlete scores across multiple games, and you want a concise list of all scores achieved by a single team, `GROUP_CONCAT()` provides the immediate solution. The process involves identifying the grouping column (e.g., `team`) and the column whose values are to be aggregated (e.g., `points`).

Syntax Breakdown for Grouped Concatenation

The most concise and common way to execute multi-row concatenation with a comma separator in MySQL involves combining the SELECT statement, the `GROUP_CONCAT()` function, and the `GROUP BY` clause. The inclusion of the `SEPARATOR` keyword ensures precise control over the output formatting. The general syntax for this operation is shown below:

You can use the following syntax in MySQL to concatenate the values from multiple rows with a comma:

```
SELECT team, GROUP_CONCAT(points SEPARATOR ', ')
AS all_points
FROM athletes
GROUP BY team;
```

This particular example selects the **team** column, which serves as the grouping key. It then creates a new column named **all_points** that concatenates all values from the **points** column, explicitly using a comma followed by a space (`, `) as the separator between each value. The GROUP BY team statement dictates how the aggregation should occur.

The following example provides a detailed walkthrough of how to use this powerful syntax in a practical database scenario.

Practical Demonstration: Setting Up the Athletes Table

To illustrate the functionality of `GROUP_CONCAT()`, consider a scenario where we are tracking basketball player scores within a database. We want to retrieve a list of all scores associated with each unique team, aggregated into a single field per team. This requires setting up a sample table and populating it with relevant data points.

We will create a table named `athletes` containing basic columns: `id` (primary key), `team` (the grouping criterion), and `points` (the value to be concatenated). Notice how multiple records are

associated with the same team, such as 'Mavs' or 'Rockets'. It is these related records that we aim to collapse using aggregation.

The SQL commands below demonstrate the creation and population of this table, followed by a standard SELECT statement to view the initial data structure:

Example: How to Concatenate Rows with Comma in MySQL

Suppose we have the following table named **athletes** that contains information about various basketball players:

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

-- insert rows into table
INSERT INTO athletes VALUES (0001, 'Mavs', 22);
INSERT INTO athletes VALUES (0002, 'Mavs', 14);
INSERT INTO athletes VALUES (0003, 'Spurs', 37);
INSERT INTO athletes VALUES (0004, 'Mavs', 19);
INSERT INTO athletes VALUES (0005, 'Rockets', 26);
INSERT INTO athletes VALUES (0006, 'Rockets', 35);
INSERT INTO athletes VALUES (0007, 'Rockets', 14);

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

Output:

```
+----+---------+--------+
| id | team | points |
+----+---------+--------+
| 1 | Mavs | 22 |
| 2 | Mavs | 14 |
| 3 | Spurs | 37 |
| 4 | Mavs | 19 |
| 5 | Rockets | 26 |
| 6 | Rockets | 35 |
```

| 7 | Rockets | 14 |
+----+---------+--------+

Executing the GROUP_CONCAT() Query

Our objective is to aggregate the values from the **points** column for each unique **team**, resulting in three distinct rows: one for Mavs, one for Rockets, and one for Spurs. Each resulting row must contain a comma-separated list of all associated point values. This is achieved by using the GROUP BY clause on the `team` column, telling MySQL to treat all records belonging to the same team as a single unit for aggregation.

The GROUP_CONCAT() function then operates within these defined groups, collecting all `points` values and joining them together using the separator specified. The alias `AS all_points` is used to give the new aggregated column a meaningful name in the result set.

We can use the following syntax to concatenate the rows as required:

```
SELECT team, GROUP_CONCAT(points SEPARATOR ', ')
AS all_points
FROM athletes
GROUP BY team;
```

Output:

```
+---------+------------+
| team | all_points |
+---------+------------+
| Mavs | 22, 14, 19 |
| Rockets | 26, 35, 14 |
| Spurs | 37 |
+---------+------------+
```

This query successfully concatenates all of the **points** values for each unique **team**, utilizing the specified comma and space as the separator.

Customizing Separators and Ordering

The flexibility of GROUP_CONCAT() extends beyond simple comma separation. If you need a different delimiter, such as a pipe symbol (`|`) or a semicolon (`;`), you only need to adjust the string provided after the `SEPARATOR` keyword. This allows for adaptability when integrating with external

systems that require specific delimited formats.

Furthermore, one critical aspect of data aggregation is ensuring that the concatenated string elements are ordered logically. By default, the order in which elements are concatenated is non-deterministic (it depends on the physical storage order). To guarantee the output order, the `ORDER BY` clause can be used *inside* the `GROUP_CONCAT()` function. For instance, to ensure scores are listed in descending order within the aggregated string, you would use `GROUP_CONCAT(points ORDER BY points DESC SEPARATOR ', ')`.

Using `ORDER BY` internally ensures that the output list is consistent and predictable, a necessity for reliable reporting and data validation. For example, if we wanted the Mavs points listed from highest score to lowest, the output would change from `22, 14, 19` to `22, 19, 14` if we applied the internal `ORDER BY` clause.

Comparison: CONCAT() vs. GROUP_CONCAT()

It is important to clearly distinguish between the CONCAT() function and the `GROUP_CONCAT()` function, as they serve fundamentally different purposes within SQL query construction. The standard CONCAT() function is a scalar function; it takes multiple arguments (typically column values or literal strings) and joins them horizontally into a single string *within the context of a single row*. It does not perform aggregation across multiple records.

For example, if a table has columns for `first_name` and `last_name`, using `CONCAT(first_name, ' ', last_name)` combines these two fields to produce a full name for that specific row. Conversely, `GROUP_CONCAT()` is an aggregation function. It requires the context of grouping (usually provided by the GROUP BY clause) to operate across multiple rows and condense their values into one field in the resulting summary row.

To summarize, if you are joining data horizontally (columns in the same record), use CONCAT(). If you are joining data vertically (rows based on a common group), use `GROUP_CONCAT()`. The requirement of concatenating rows with a comma separator definitively points toward the use of the aggregate function.

Key Considerations for Data Aggregation

When implementing `GROUP_CONCAT()`, two technical limits require attention. First, the maximum length of the resulting concatenated string is governed by the system variable `group_concat_max_len`. By default, this limit is often set to 1024 characters. If your groups contain many values, the concatenated string may be truncated. If truncation is observed, this variable must be increased using the `SET` command (e.g., `SET group_concat_max_len = 10000;`) before executing the query.

Second, the GROUP_CONCAT() function also supports the `DISTINCT` keyword. If the source data contains duplicate values within a group that you do not wish to include in the final concatenated string, you can specify `GROUP_CONCAT(DISTINCT points SEPARATOR ', ')`. This ensures that each unique point value appears only once in the resulting comma-separated list for that group.

Note #1: We used the **AS** statement to name the new aggregated column **all_points**, which is a standard SQL practice for assigning meaningful aliases to calculated fields.

Note #2: If you would like to use a different separator when concatenating values together, simply specify that separator after the **SEPARATOR** keyword in the **GROUP_CONCAT()** function.

Further MySQL Tutorials

The following resources explain how to perform other common tasks in MySQL:

Tutorial on using the GROUP BY clause effectively.

Advanced techniques for using the SELECT statement.

How to utilize the CONCAT() function for single-row data manipulation.