Learning MySQL: Retrieving the Row with the Maximum Value in a Column

Authored by

Mohammed looti

November 12, 2025

Introduction to Finding Maximum Values in MySQL

A frequent requirement in database management is the need to retrieve an entire row of data based on the highest value present in a specific column. Whether you are building a leaderboard, identifying the highest-priced item, or finding the most recent transaction, efficiently selecting the record associated with the maximum value is a fundamental skill for working with MySQL. While finding the maximum value itself is trivial using aggregate functions, retrieving the complete, corresponding row requires a more sophisticated approach, often involving nested queries or specialized ranking techniques.

The most straightforward and universally compatible method for isolating the row containing the maximum value involves employing a subquery. This technique breaks down the problem into two distinct, manageable steps: first, determining the absolute maximum value within the desired column, and second, using that result to filter the entire database table. This ensures accuracy and allows the full context of the record--not just the maximum number--to be returned to the user.

The core syntax detailed below leverages this subquery pattern, making it a reliable solution across various MySQL versions.

```
SELECT id, team, points
FROM athletes
WHERE points=(SELECT MAX(points) FROM athletes);
```

This particular example demonstrates how to select the complete record associated with the maximum value found in the **points** column within the database table named **athletes**. We will now explore this methodology in detail using a practical, real-world dataset.

Method 1: Utilizing the Subquery Approach

The subquery approach is highly effective because it resolves the selection criterion before the main query attempts to retrieve data. A subquery, or inner query, is executed first and its result is then used by the outer query. When searching for the maximum value row, the inner query is solely responsible for calculating the absolute maximum numerical value present in the specified column using the MAX() function.

In the example above, the inner query, **(SELECT MAX(points) FROM athletes)**, returns a single scalar value--the highest number of points recorded. The outer SELECT statement then uses this single number within its **WHERE** clause to filter the entire table. The condition **WHERE points=(...)** ensures that only rows where the **points** column exactly matches the maximum score identified by the inner query are returned.

This two-tiered strategy is favored for its clarity and reliability, especially when dealing with moderate dataset sizes. It avoids ambiguity and guarantees that you are comparing every row's value against the true maximum, thus retrieving the correct corresponding record. The following example demonstrates how to apply this syntax in practice, beginning with the creation and population of our sample data.

Practical Demonstration: Setting Up the Data

To illustrate this concept, let us establish a sample database table named **athletes**. This table is designed to store basic performance metrics for various basketball players, including their unique identifier, team affiliation, and key statistics like points, assists, and rebounds.

The following SQL commands create the table structure and populate it with six distinct records. Notice that the MySQL syntax is clean and defines appropriate data types (e.g., INT, TEXT) for each column, ensuring data integrity before we proceed with the query execution.

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

```
-- insert rows into table
INSERT INTO athletes VALUES (0001, 'Mavs', 22, 4, 3);
INSERT INTO athletes VALUES (0002, 'Kings', 14, 5, 13);
INSERT INTO athletes VALUES (0003, 'Lakers', 37, 6, 10);
INSERT INTO athletes VALUES (0004, 'Nets', 19, 10, 3);
INSERT INTO athletes VALUES (0005, 'Knicks', 26, 12, 8);
INSERT INTO athletes VALUES (0006, 'Celtics', 15, 1, 2);

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

After executing the insertion and the final SELECT statement, the current state of our **athletes** table is displayed below. A quick review confirms that the highest value in the **points** column is 37, corresponding to the Lakers team (ID 3). Our goal is to use the subquery method to programmatically retrieve only this specific row.

Output of Initial Data:

```
+----+---------+--------+---------+----------+
| id | team | points | assists | rebounds |
+----+---------+--------+---------+----------+
| 1 | Mavs | 22 | 4 | 3 |
| 2 | Kings | 14 | 5 | 13 |
| 3 | Lakers | 37 | 6 | 10 |
| 4 | Nets | 19 | 10 | 3 |
| 5 | Knicks | 26 | 12 | 8 |
| 6 | Celtics | 15 | 1 | 2 |
+----+---------+--------+---------+----------+
```

Executing the SELECT statement and Analyzing Results

We now apply the primary query structure designed to isolate the row containing the maximum value in the **points** column. As previously explained, the inner query first determines that 37 is the maximum value, and the outer query then filters the entire table where **points = 37**. This ensures robust retrieval of the corresponding record.

The syntax is concise and highly efficient for this specific task:

```
SELECT id, team, points
FROM athletes
WHERE points=(SELECT MAX(points) FROM athletes);
```

Upon execution, the result clearly shows that only the row matching the highest score is returned. This confirms the successful application of the nested SELECT statement using the MAX() function.

Output of Max Value Query:

```
+----+--------+--------+
| id | team | points |
+----+--------+--------+
| 3 | Lakers | 37 |
+----+--------+--------+
```

Notice that only the row corresponding to the athlete with the max value in the **points** column is returned. A crucial advantage of this subquery method is its inherent ability to handle ties

gracefully. If there were multiple rows tied with the maximum value (e.g., two athletes scoring 37 points), the outer query's **WHERE** clause would evaluate to true for both, and consequently, both rows would be returned in the final result set. This capability makes the subquery method robust against data duplication in the maximum range.

Alternative Method: Using Window Functions and Ranking

While the subquery method is highly effective, modern MySQL (versions 8.0 and later) introduced Window Functions, which often provide more readable and sometimes more performant alternatives, particularly when complex ranking or grouping is required. Functions like **RANK()** or **DENSE_RANK()** can assign a rank to each row based on the column value (e.g., **points**) and then the outer query can simply filter for rank 1.

To achieve the same goal--selecting the row(s) with the maximum points--using a Window Function, we would first define a Common Table Expression (CTE) or derived table where the ranking is calculated. We use **DENSE_RANK() OVER (ORDER BY points DESC)** to assign the highest point earners a rank of 1.

The benefit of this approach is its flexibility. If you later needed the top 3 highest scorers, you would simply change the final **WHERE** clause from **rank = 1** to **rank <= 3**. For highly complex scenarios involving partitioning (e.g., finding the highest scorer *per team*), Window Functions are generally superior to the traditional subquery method. However, for the simple task of finding the global maximum row, both methods yield equivalent results.

Handling Ties and Edge Cases

It is critical for database operations to account for scenarios where multiple records share the maximum value. This situation, known as a tie, must be handled correctly to ensure all relevant data is retrieved.

As noted previously, both the subquery method and the **DENSE_RANK()** window function naturally handle ties by returning all rows that satisfy the maximum condition. If, for instance, athlete 5 also scored 37 points, the subquery would return both athlete 3 and athlete 5 because both satisfy the criterion **WHERE points = 37**.

However, if the requirement is strictly to return only one row, even in the case of a tie, two primary alternatives exist. The first is appending **LIMIT 1** to the query after sorting the data: **SELECT * FROM athletes ORDER BY points DESC LIMIT 1;** While this is syntactically simple, it offers no guarantee which tied row will be returned. A more controlled method is to introduce a secondary sorting criterion (e.g., sorting by **id** in ascending order) to break the tie deterministically.

Conclusion and Further Resources

Retrieving the row associated with the maximum value in a column is a common yet nuanced task in MySQL. The subquery pattern--where the inner query calculates the maximum using the MAX() function and the outer query filters based on that result--remains the most reliable and widely compatible method for achieving this goal. For those using modern MySQL versions, exploring Window Functions offers powerful, scalable alternatives, especially when dealing with complex ranking requirements.

Mastering the use of nested queries and aggregate functions ensures that you can always pinpoint and retrieve the most significant records from your database table efficiently and accurately, regardless of whether ties are present in your data.

Additional Resources

The following tutorials explain how to perform other common tasks in MySQL and enhance your understanding of advanced SQL concepts:

Tutorial on using the MAX() function in grouped queries.

Detailed guide to writing effective subquery structures.

Understanding the difference between **RANK()** and **DENSE_RANK()**.