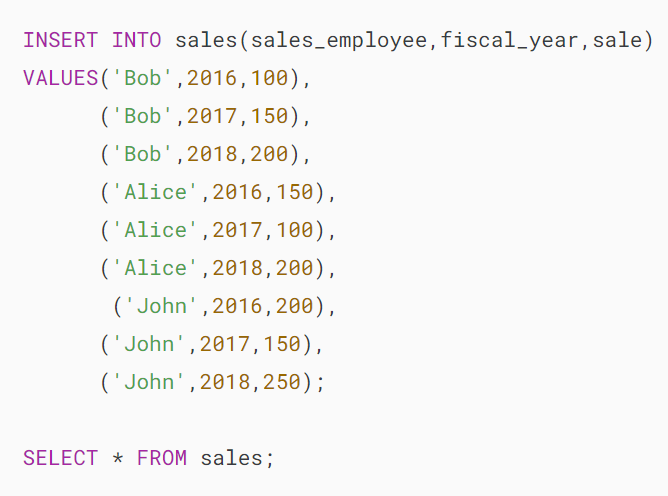
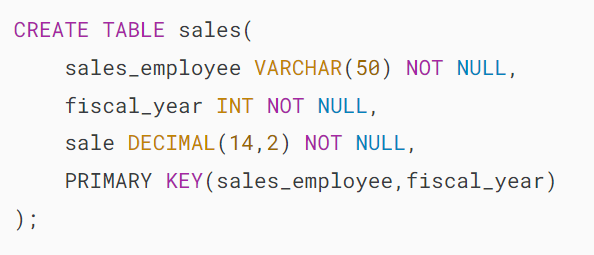
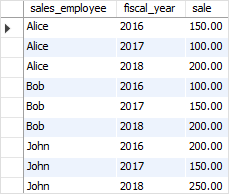
# **MySQL Window Functions**

**Summary**: in this tutorial, you will learn about MySQL window functions and their practical applications for solving analytical query challenges.

MySQL has supported window functions since version 8.0, allowing you to solve query problems more easily and with better performance.

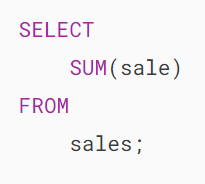
Suppose that we have the sales table which stores the sales by employees and fiscal years:



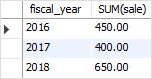
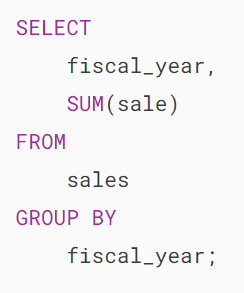


It’s probably easier to understand window functions is to start with [aggregate functions](https://www.mysqltutorial.org/mysql-aggregate-functions/).

Aggregate functions summarize data from multiple rows into a single result row. For example, the following [SUM()](https://www.mysqltutorial.org/mysql-aggregate-functions/mysql-sum/)function returns the total sales of all employees in the recorded years:

MySQL Window Function - vs Aggregate SUM

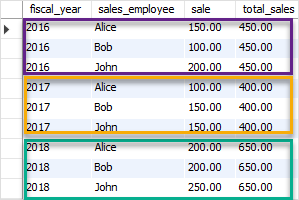
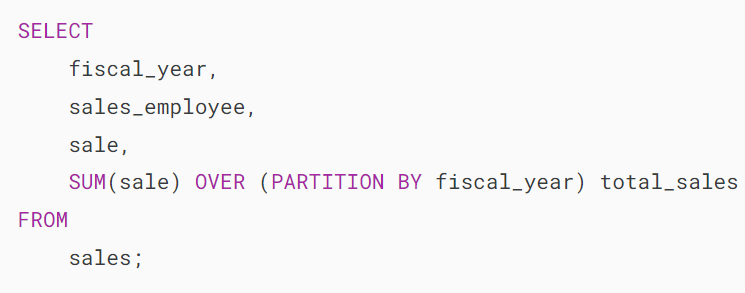
The [GROUP BY](https://www.mysqltutorial.org/mysql-basics/mysql-group-by/) clause allows you to apply aggregate functions to a subset of rows. For example, you may want to calculate the total sales by fiscal years:



In both examples, the aggregate functions reduce the number of rows returned by the query.

Like the aggregate functions with the [GROUP BY](https://www.mysqltutorial.org/mysql-basics/mysql-group-by/) clause, window functions also operate on a subset of rows but they do not reduce the number of rows returned by the query.

For example, the following query returns the sales for each employee, along with the total sales of the employees by fiscal year:



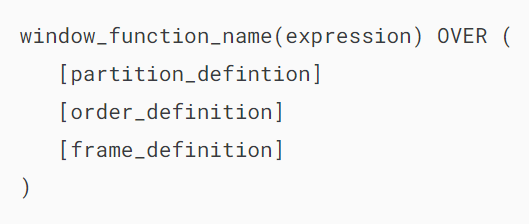
In this example, the SUM() function works as a window function that operates on a set of rows defined by the contents of the OVER clause. A set of rows to which the SUM() function applies is referred to as a window.

The SUM() window function reports not only the total sales by fiscal year as it does in the query with the GROUP BY clause, but also the result in each row, rather than the total number of rows returned.

Note that window functions are performed on the result set after all [JOIN](https://www.mysqltutorial.org/mysql-basics/mysql-join/), [WHERE](https://www.mysqltutorial.org/mysql-basics/mysql-where/), [GROUP BY](https://www.mysqltutorial.org/mysql-basics/mysql-group-by/), and [HAVING](https://www.mysqltutorial.org/mysql-basics/mysql-having/) clauses and before the [ORDER BY](https://www.mysqltutorial.org/mysql-basics/mysql-order-by/), [LIMIT](https://www.mysqltutorial.org/mysql-basics/mysql-limit/) and [SELECT DISTINCT](https://www.mysqltutorial.org/mysql-basics/mysql-distinct/).

## **Window function syntax**

The general syntax of calling a window function is as follows:



In this syntax:

* First, specify the window function name followed by an expression.
* Second, specify the OVER clause which has three possible elements: partition definition, order definition, and frame definition.

The opening and closing parentheses, which appear after the OVER clause, is mandatory, even with no expression, for example:

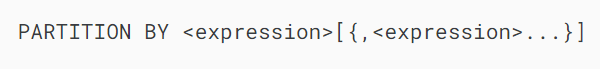
### 

### **partition\_clause syntax**

The partition\_clause breaks up the rows into chunks or partitions. Two partitions are separated by a partition boundary.

The window function is performed within partitions and re-initialized when crossing the partition boundary.

The partition\_clause syntax looks like the following:



You can specify one or more expressions in the PARTITION BY clause. Multiple expressions are separated by commas.

### **order\_by\_clause syntax**

The order\_by\_clause has the following syntax:



The ORDER BY clause specifies how the rows are ordered within a partition. It is possible to order data within a partition on multiple keys, each key is specified by an expression. Multiple expressions are also separated by commas.

Similar to the PARTITION BY clause, the ORDER BY clause is also supported by all the window functions. However, it only makes sense to use the ORDER BY clause for order-sensitive window functions.

### **frame\_clause syntax**

A frame is a subset of the current partition. To define the subset, you use the frame clause as follows:



A frame is defined with respect to the current row, which allows a frame to move within a partition depending on the position of the current row within its partition.

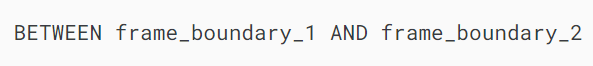
The frame unit specifies the type of relationship between the current row and frame rows. It can be ROWS or RANGE. The offsets of the current row and frame rows are the row numbers if the frame unit is ROWS and row values the frame unit is RANGE.

The frame\_start and frame\_between define the frame boundary.

The frame\_start contains one of the following:

* UNBOUNDED PRECEDING: frame starts at the first row of the partition.
* N PRECEDING: a physical N of rows before the first current row. N can be a literal number or an expression that evaluates to a number.
* CURRENT ROW: the row of the current calculation

The frame\_between is as follows:

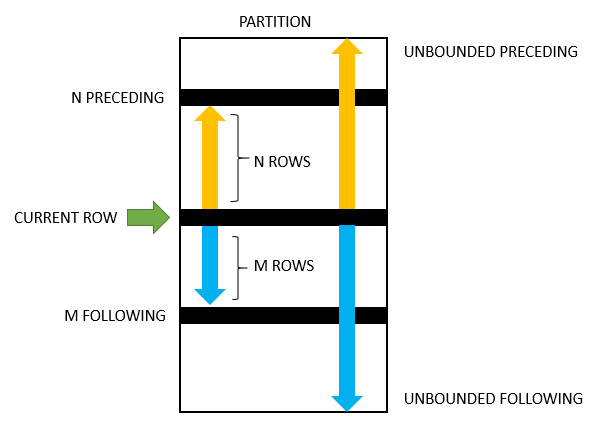


The frame\_boundary\_1 and frame\_boundary\_2 can each contain one of the following:

* frame\_start: as mentioned previously.
* UNBOUNDED FOLLOWING: the frame ends at the final row in the partition.
* N FOLLOWING: a physical N of rows after the current row.

If you don’t specify the frame\_definition in the OVER clause, then MySQL uses the following frame by default:

RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROWCode language: SQL (Structured Query Language) (sql)



## **MySQL Window Function list**

The following table shows the window functions in MySQL:

| **Name** | **Description** |
| --- | --- |
| [CUME\_DIST](https://www.mysqltutorial.org/mysql-window-functions/mysql-cume_dist-function/) | Calculates the cumulative distribution of a value in a set of values. |
| [DENSE\_RANK](https://www.mysqltutorial.org/mysql-window-functions/mysql-dense_rank-function/) | Assigns a rank to every row within its partition based on the ORDER BY clause. It assigns the same rank to the rows with equal values. If two or more rows have the same rank, then there will be no gaps in the sequence of ranked values. |
| [FIRST\_VALUE](https://www.mysqltutorial.org/mysql-window-functions/mysql-first_value-function/) | Returns the value of the specified expression with respect to the first row in the window frame. |
| [LAG](https://www.mysqltutorial.org/mysql-window-functions/mysql-lag-function/) | Returns the value of the Nth row before the current row in a partition. It returns NULL if no preceding row exists. |
| [LAST\_VALUE](https://www.mysqltutorial.org/mysql-window-functions/mysql-last_value-function/) | Returns the value of the specified expression with respect to the last row in the window frame. |
| [LEAD](https://www.mysqltutorial.org/mysql-window-functions/mysql-lead-function/) | Returns the value of the Nth row after the current row in a partition. It returns NULL if no subsequent row exists. |
| [NTH\_VALUE](https://www.mysqltutorial.org/mysql-window-functions/mysql-nth_value-function/) | Returns value of argument from Nth row of the window frame |
| [NTILE](https://www.mysqltutorial.org/mysql-window-functions/mysql-ntile-function/) | Distributes the rows for each window partition into a specified number of ranked groups. |
| [PERCENT\_RANK](https://www.mysqltutorial.org/mysql-window-functions/mysql-percent_rank-function/) | Calculates the percentile rank of a row in a partition or result set |
| [RANK](https://www.mysqltutorial.org/mysql-window-functions/mysql-rank-function/) | Similar to the DENSE\_RANK() function except that there are gaps in the sequence of ranked values when two or more rows have the same rank. |
| [ROW\_NUMBER](https://www.mysqltutorial.org/mysql-window-functions/mysql-row_number-function/) | Assigns a sequential integer to every row within its partition |

In this tutorial, you have learned about the MySQL window functions and their syntax. In the next tutorials, you will learn more about each window function and its applications in more detail.

# **MySQL CUME\_DIST Function**

**Summary**: in this tutorial, you will learn how to use the MySQL CUME\_DIST() function to calculate cumulative distribution value.

## **Overview of MySQL CUME\_DIST() Function**

The CUME\_DIST() is a [window function](https://www.mysqltutorial.org/mysql-window-functions/) that returns the cumulative distribution of a value within a set of values. It represents the number of rows with values less than or equal to that row’s value divided by the total number of rows.

The returned value of the CUME\_DIST() function is greater than zero and less than or equal to one (0 < CUME\_DIST() <= 1). The repeated column values receive the same CUME\_DIST() value.

The following shows the syntax of the CUME\_DIST() function:

CUME\_DIST()

OVER (

PARTITION BY expr

ORDER BY expr [ASC | DESC]

)Code language: SQL (Structured Query Language) (sql)

In this syntax:

### **PARTITION BY**

The  PARTITION BY clause divides the result set into partitions to which the CUME\_DIST() function is applied independently. If you omit the PARTITION BY clause, the function is applied to the whole result set.

### **ORDER BY**

The ORDER BY clause specifies the order of the rows in each partition or the whole result set in case the PARTITION BY is omitted.

The CUME\_DIST() function calculates the cumulative distribution value of each row based on its order in the partition.

The approximate formula of the CUME\_DIST() function is as follows:

ROW\_NUMBER() / total\_rowsCode language: SQL (Structured Query Language) (sql)

## **MySQL CUME\_DIST() function example**

Let’s [create a table](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/) called scores and populate some data for the demonstration:

CREATE TABLE scores (

name VARCHAR(20) PRIMARY KEY,

score INT NOT NULL

);

INSERT INTO

scores(name, score)

VALUES

('Smith',81),

('Jones',55),

('Williams',55),

('Taylor',62),

('Brown',62),

('Davies',84),

('Evans',87),

('Wilson',72),

('Thomas',72),

('Johnson',100);Code language: SQL (Structured Query Language) (sql)

The following statement uses the CUME\_DIST() function to find the cumulative distribution of the score in the result set:

SELECT

name,

score,

ROW\_NUMBER() OVER (

ORDER BY

score

) row\_num,

CUME\_DIST() OVER (

ORDER BY

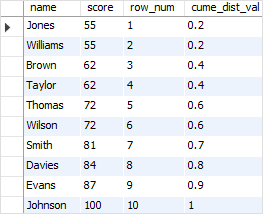
score

) cume\_dist\_val

FROM

scores;Code language: SQL (Structured Query Language) (sql)

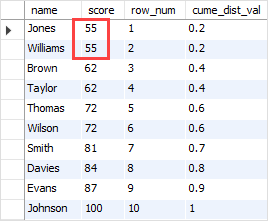
Here is the output:



In this example, the score is sorted in ascending order from 55 to 100. Note that the [ROW\_NUMBER()](https://www.mysqltutorial.org/mysql-window-functions/mysql-row_number-function/) function was added for reference.

So how does the CUME\_DIST() function perform calculation?

In the initial step, the function identifies the number of rows in the result set where the values are less than or equal to 55. This count is found to be 2 for the first row. Next, the CUME\_DIST() function computes the cumulative distribution by dividing this count (2) by the total number of rows in the set, which is 10: 2/10. The result is 0.2 or 20%. The same procedure is then repeated for the second row.



In the case of the third row, the CUME\_DIST() function identifies four rows within the result set where the values are less than or equal to 62. then, the CUME\_DIST() function computes the cumulative distribution by dividing this count (4) by the total number of rows in the set, which is 10: 4/10. The result is 0.4 or 40%.

The same calculation logic is applied to the remaining rows.

## **Summary**

* Use the MySQL CUME\_DIST() function to calculate the cumulative distribution of a value in a set of values.