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

MySQL has supported window functions since version 8.0. The window functions allow you to solve query problems in new, easier ways and with better performance.

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

```
CREATE TABLE sales(
    sales_employee VARCHAR(50) NOT NULL,
    fiscal_year INT NOT NULL,
    sale DECIMAL(14,2) NOT NULL,
    PRIMARY KEY(sales_employee,fiscal_year)
);

INSERT INTO sales(sales_employee,fiscal_year,sale)
VALUES('Bob',2016,100),
      ('Bob',2017,150),
      ('Bob',2018,200),
      ('Alice',2016,150),
      ('Alice',2017,100),
      ('Alice',2018,200),
       ('John',2016,200),
      ('John',2017,150),
      ('John',2018,250);

SELECT * FROM sales;
```

| | sales_employee | fiscal_year | sale |
|---|---|---|---|
| ▶ | Alice | 2016 | 150.00 |
| | Alice | 2017 | 100.00 |
| | Alice | 2018 | 200.00 |
| | Bob | 2016 | 100.00 |
| | Bob | 2017 | 150.00 |
| | Bob | 2018 | 200.00 |
| | John | 2016 | 200.00 |
| | John | 2017 | 150.00 |
| | John | 2018 | 250.00 |

It's probably easier to understand window functions is to start with aggregate functions.

Aggregate functions summarize data from multiple rows into a single result row. For example, the following `SUM()` function returns the total sales of all employees in the recorded years:

```
SELECT
    SUM(sale)
FROM
    sales;
```

| | SUM(sale) |
|---|---|
| ▶ | 1500.00 |

The 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:

```
SELECT
    fiscal_year,
    SUM(sale)
FROM
    sales
GROUP BY
    fiscal_year;
```

| fiscal_year | SUM(sale) |
|---|---|
| 2016 | 450.00 |
| 2017 | 400.00 |
| 2018 | 650.00 |

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

Like the aggregate functions with the 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 total sales of the employees by fiscal year:

```
SELECT
    fiscal_year,
    sales_employee,
    sale,
    SUM(sale) OVER (PARTITION BY fiscal_year) total_sales
FROM
    sales;
```

| fiscal_year | sales_employee | sale | total_sales |
|---|---|---|---|
| 2016 | Alice | 150.00 | 450.00 |
| 2016 | Bob | 100.00 | 450.00 |
| 2016 | John | 200.00 | 450.00 |
| 2017 | Alice | 100.00 | 400.00 |
| 2017 | Bob | 150.00 | 400.00 |
| 2017 | John | 150.00 | 400.00 |
| 2018 | Alice | 200.00 | 650.00 |
| 2018 | Bob | 200.00 | 650.00 |
| 2018 | John | 250.00 | 650.00 |

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`, `WHERE`, `GROUP BY`, and `HAVING` clauses and before the `ORDER BY`, `LIMIT` and `SELECT DISTINCT`.

# Window function syntax

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

```
window_function_name(expression) OVER (
   [partition_defintion]
   [order_definition]
   [frame_definition]
)
```

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 after the `OVER` clause are mandatory, even with no expression, for example:

```
window_function_name(expression) OVER()
```

## 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:

```
PARTITION BY <expression>[{,<expression>...}]
```

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:

```
ORDER BY <expression> [ASC|DESC], [{,<expression>...}]
```

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:

```
frame_unit {<frame_start>|<frame_between>}
```

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:

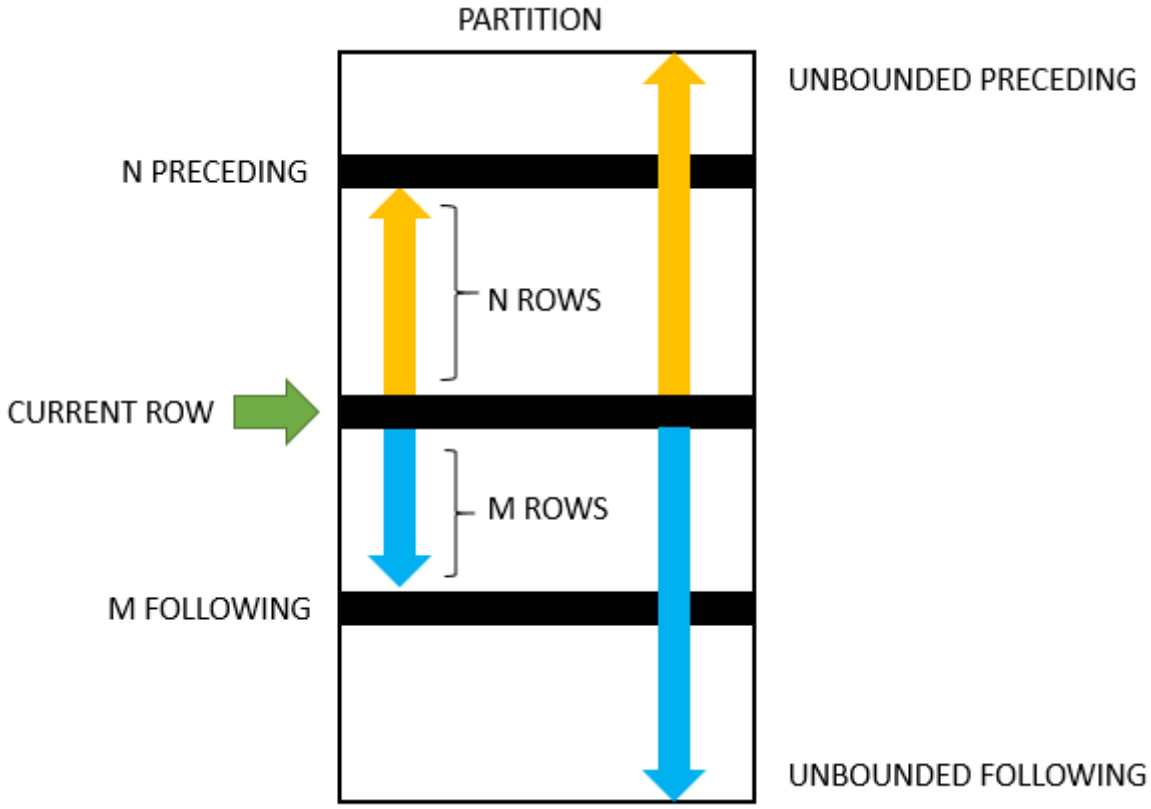
```
BETWEEN frame_boundary_1 AND frame_boundary_2
```

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 ROW
```




## MySQL Window Function list

The following table shows the window functions in MySQL:

Type a function name to search...

| Name | Description |
|---|---|
| CUME_DIST | Calculates the cumulative distribution of a value in a set of values. |
| DENSE_RANK | 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 | Returns the value of the specified expression with respect to the first row in the window frame. |
| LAG | Returns the value of the Nth row before the current row in a partition. It returns NULL if no preceding row exists. |
| LAST_VALUE | Returns the value of the specified expression with respect to the last row in the window frame. |
| LEAD | Returns the value of the Nth row after the current row in a partition. It returns NULL if no subsequent row exists. |

| Name | Description |
| --- | --- |
| NTH_VALUE | Returns value of argument from Nth row of the window frame |
| NTILE | Distributes the rows for each window partition into a specified number of ranked groups. |
| PERCENT_RANK | Calculates the percentile rank of a row in a partition or result set |
| RANK | 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 | 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.