MySQLTUTORIAL

MySQL GROUP BY

Summary: in this tutorial, you will learn how to use **MySQL** `GROUP BY` to group rows into subgroups based on values of columns or expressions.

Introduction to MySQL GROUP BY clause

The `GROUP BY` clause groups a set of rows into a set of summary rows by values of columns or expressions. The `GROUP BY` clause returns one row for each group. In other words, it reduces the number of rows in the result set.

The `GROUP BY` clause is an optional clause of the `SELECT (https://www.mysqltutorial.org/mysql-select-statement-query-data.aspx)` statement. The following illustrates the `GROUP BY` clause syntax:

```
SELECT
    c1, c2,..., cn, aggregate_function(ci)
FROM
    table
WHERE
    where_conditions
GROUP BY c1 , c2,...,cn;
```

In this syntax, you place the `GROUP BY` clause after the `FROM` and `WHERE` clauses. After the GROUP BY keywords, you place is a list of comma-separated columns or expressions to group rows.

MySQL evaluates the `GROUP BY (https://www.mysqltutorial.org/mysql-group-by.aspx)` clause after the `FROM` and `WHERE (https://www.mysqltutorial.org/mysql-where/)` clauses and before the `HAVING (https://www.mysqltutorial.org/mysql-having.aspx)` , `SELECT` , `DISTINCT (https://www.mysqltutorial.org/mysql-distinct.aspx)` , `ORDER BY (https://www.mysqltutorial.org/mysql-order-by/)` and `LIMIT (https://www.mysqltutorial.org/mysql-limit.aspx)` clauses:

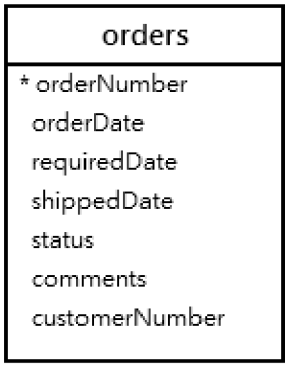
In practice, you often use the `GROUP BY` clause with aggregate functions (https://www.mysqltutorial.org/mysql-aggregate-functions.aspx) such as `SUM` (https://www.mysqltutorial.org/mysql-sum/), `AVG` (https://www.mysqltutorial.org/mysql-avg/), `MAX` (https://www.mysqltutorial.org/mysql-max-function/), `MIN` (https://www.mysqltutorial.org/mysql-min/), and `COUNT` (https://www.mysqltutorial.org/mysql-count/). The aggregate function that appears in the `SELECT` clause provides the information of each group.

MySQL GROUP BY examples

Let's take some examples of using the `GROUP BY` clause.

A) Simple MySQL GROUP BY example

Let's take a look at the `orders` table in the sample database (https://www.mysqltutorial.org/mysql-sample-database.aspx).



Suppose you want to group values of the order's status into subgroups, you use the `GROUP BY` clause with the `status` column as the following query:

```
SELECT
    status
FROM
    orders
GROUP BY status;
```

Try It Out ›

	status
▶	Cancelled
	Disputed
	In Process
	On Hold
	Resolved
	Shipped

As you can see clearly from the output, the `GROUP BY` clause returns unique occurrences of `status` values. It works like the `DISTINCT` (https://www.mysqltutorial.org/mysql-distinct.aspx) operator as shown in the following query:

```
SELECT DISTINCT
    status
FROM
    orders;
```

Try It Out ›

B) Using MySQL GROUP BY with aggregate functions

The aggregate functions (https://www.mysqltutorial.org/mysql-aggregate-functions.aspx) allow you to perform the calculation of a set of rows and return a single value. The `GROUP BY` clause is often used with an aggregate function to perform calculations and return a single value for each subgroup.

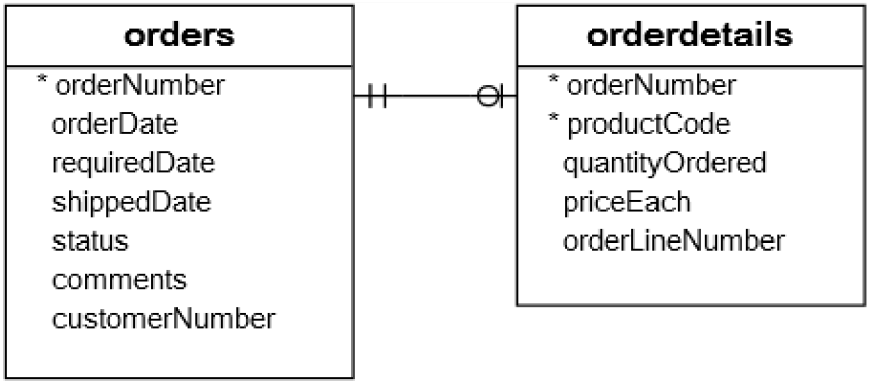
For example, if you want to know the number of orders in each status, you can use the `COUNT` function with the `GROUP BY` clause as follows:

```
SELECT
    status, COUNT(*)
FROM
    orders
GROUP BY status;
```

Try It Out ›

	status	COUNT(*)
▶	Cancelled	6
	Disputed	3
	In Process	6
	On Hold	4
	Resolved	4
	Shipped	303

See the following `orders` and `orderdetails` table.



To get the total amount of all orders by status, you join (https://www.mysqltutorial.org/mysql-inner-join.aspx) the `orders` table with the `orderdetails` table and use the `SUM` function to calculate the total amount. See the following query:

```
SELECT
    status,
    SUM(quantityOrdered * priceEach) AS amount
FROM
    orders
INNER JOIN orderdetails
    USING (orderNumber)
GROUP BY
    status;
```

Try It Out ›

	status	amount
▶	Cancelled	238854.18
	Disputed	61158.78
	In Process	135271.52
	On Hold	169575.61
	Resolved	134235.88
	Shipped	8865094.64

Similarly, the following query returns the order numbers and the total amount of each order.

```
SELECT
    orderNumber,
    SUM(quantityOrdered * priceEach) AS total
FROM
    orderdetails
GROUP BY
    orderNumber;
```

Try It Out ›

	orderNumber	total
▶	10100	10223.83
	10101	10549.01
	10102	5494.78
	10103	50218.95
	10104	40206.20
	10105	53959.21
	10106	52151.81
	10107	22292.62

C) MySQL GROUP BY with expression example

In addition to columns, you can group rows by expressions. The following query gets the total sales for each year.

```
SELECT
    YEAR(orderDate) AS year,
    SUM(quantityOrdered * priceEach) AS total
FROM
    orders
```

```
INNER JOIN orderdetails
    USING (orderNumber)
WHERE
    status = 'Shipped'
GROUP BY
    YEAR(orderDate);
```

Try It Out

	year	total
▶	2003	3223095.80
	2004	4300602.99
	2005	1341395.85

In this example, we used the `YEAR (https://www.mysqltutorial.org/mysql-year/)` function to extract year data from order date (`orderDate`). We included only orders with `shipped` status in the total sales. Note that the expression which appears in the `SELECT` clause must be the same as the one in the `GROUP BY` clause.

D) Using MySQL GROUP BY with HAVING clause example

To filter the groups returned by `GROUP BY` clause, you use a `HAVING` (https://www.mysqltutorial.org/mysql-having.aspx) clause. The following query uses the `HAVING` clause to select the total sales of the years after 2003.

```
SELECT
    YEAR(orderDate) AS year,
    SUM(quantityOrdered * priceEach) AS total
FROM
    orders
INNER JOIN orderdetails
    USING (orderNumber)
WHERE
    status = 'Shipped'
GROUP BY
    year
HAVING
    year > 2003;
```

Try It Out ›

	year	total
▶	2004	4300602.99
	2005	1341395.85

The GROUP BY clause: MySQL vs. SQL standard

The SQL standard does not allow you to use an alias in the `GROUP BY` clause whereas MySQL supports this.

For example, the following query extracts the year from the order date. It first uses the `year` as an alias of the expression `YEAR(orderDate)` and then uses the `year` alias in the `GROUP BY` clause.

The following query is not valid in SQL standard:

```
SELECT
    YEAR(orderDate) AS year,
    COUNT(orderNumber)
FROM
    orders
GROUP BY
    year;
```

Try It Out ›

	year	COUNT(orderNumber)
▶	2003	111
	2004	151
	2005	64

Also, MySQL allows you to sort the groups in ascending or descending orders. The default sorting order is ascending. For example, if you want to get the number of orders by status and sort the status in descending order, you can use the `GROUP BY` clause with `DESC` as the following query:

```
SELECT
    status,
    COUNT(*)
FROM
```

```
    orders
GROUP BY
    status DESC;
```

Try It Out ›

	status	COUNT(*)
▶	Shipped	303
	Resolved	4
	On Hold	4
	In Process	6
	Disputed	3
	Cancelled	6

Notice the `DESC` in the `GROUP BY` clause sorts the `status` in descending order. And you can also use the `ASC` explicitly in the `GROUP BY` clause to sort the groups by status in ascending order.

The GROUP BY clause vs. DISTINCT clause

If you use the `GROUP BY` clause in the `SELECT` statement without using aggregate functions (https://www.mysqltutorial.org/mysql-aggregate-functions.aspx) , the `GROUP BY` clause behaves like the `DISTINCT` (https://www.mysqltutorial.org/mysql-distinct.aspx) clause.

The following statement uses the `GROUP BY` clause to select the unique states of customers from the `customers` table.

```
SELECT
    state
FROM
    customers
GROUP BY state;
```

Try It Out ›

state
NULL
BC
CA
Co. Cork
CT
Isle of Wight
MA

You can achieve a similar result by using the `DISTINCT` clause:

```
SELECT DISTINCT
    state
FROM
    customers;
```

Try It Out ›

state
NULL
NV
Victoria
CA
NY
PA

Generally speaking, the `DISTINCT` clause is a special case of the `GROUP BY` clause. The difference between `DISTINCT` clause and `GROUP BY` clause is that the `GROUP BY` clause sorts the result set (https://www.mysqltutorial.org/mysql-order-by/) , whereas the `DISTINCT` clause does not.

> Notice that MySQL 8.0 removed the implicit sorting for the `GROUP BY` clause. Therefore, if you use MySQL 8.0+, you will find that the result set of the above query with the `GROUP BY` clause is not sorted.

If you add the `ORDER BY (https://www.mysqltutorial.org/mysql-order-by/)` clause to the statement that uses the `DISTINCT` clause, the result set is sorted, and it is the same as the one returned by the statement that uses `GROUP BY` clause.

```
SELECT DISTINCT
    state
FROM
    customers
ORDER BY
    state;
```

Try It Out ›

Summary

- Use the `GROUP BY` clause to group rows into subgroups.