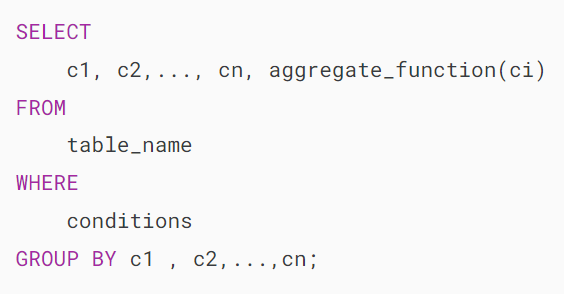
# **MySQL GROUP BY**

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

## **Introduction to MySQL GROUP BY clause**

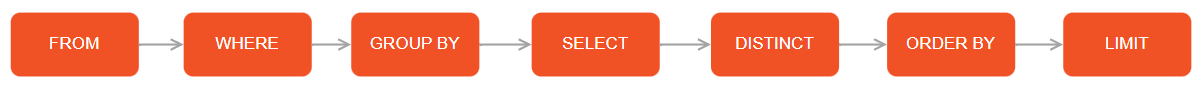
The GROUP BY clause groups rows into summary rows based on column values or expressions. It returns one row for each group and reduces the number of rows in the result set.

The GROUP BY clause is an optional part of the [SELECT](https://www.mysqltutorial.org/mysql-basics/mysql-select-from/) statement. The following illustrates the syntax of the GROUP BY clause:



In this syntax, you place the GROUP BY clause after the FROM and WHERE clauses. Following the the GROUP BY keywords, you list the columns or expressions you want to group, separated by commas.

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



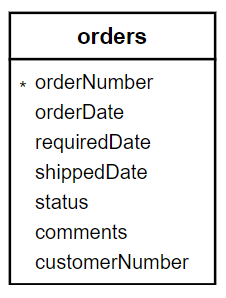
In practice, you often use the GROUP BY clause with [aggregate functions](https://www.mysqltutorial.org/mysql-aggregate-functions/) such as [SUM](https://www.mysqltutorial.org/mysql-aggregate-functions/mysql-sum/), [AVG](https://www.mysqltutorial.org/mysql-aggregate-functions/mysql-avg/), [MAX](https://www.mysqltutorial.org/mysql-aggregate-functions/mysql-max-fu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

## **MySQL GROUP BY examples**

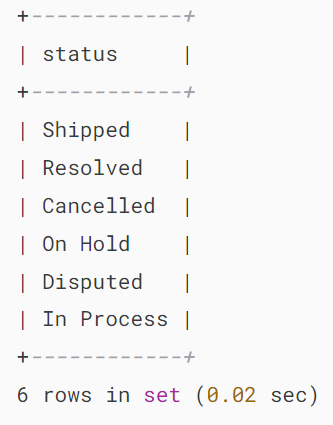
Let’s look at some examples of using the GROUP BY clause.

### **1) Basic MySQL GROUP BY example**

We’ll use the orders table in the [sample database](https://www.mysqltutorial.org/getting-started-with-mysql/mysql-sample-database/):



If you want to group the order statuses, you can use the GROUP BY clause with the status column in the following query:



The output shows that the GROUP BY clause returns unique occurrences of the values in the status columns.

It works like the [DISTINCT](https://www.mysqltutorial.org/mysql-basics/mysql-distinct/) operator, as demonstrated in the following query:

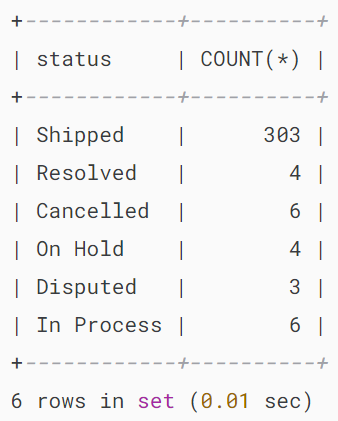
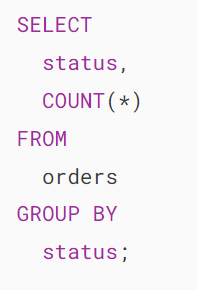
### 

### **2) Using MySQL GROUP BY with aggregate functions**

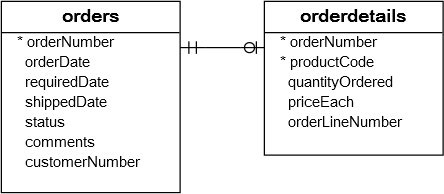
In practice, you often use the GROUP BY clause with an aggregate function to group rows into sets and return a single value for each group.

An [aggregate function](https://www.mysqltutorial.org/mysql-aggregate-functions/) calculates a set of rows and returns a single value.

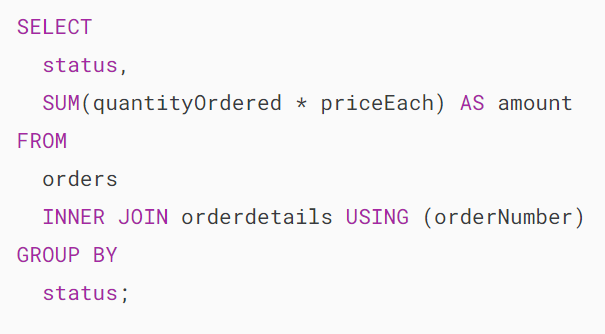
For example, to obtain the number of orders in each status, you can use the COUNT function with the GROUP BY clause as follows:

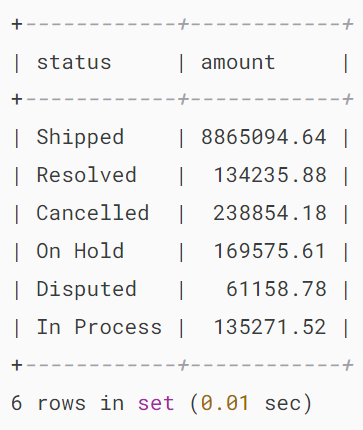


See the following orders and  orderdetails table from the sample database:

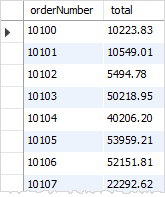
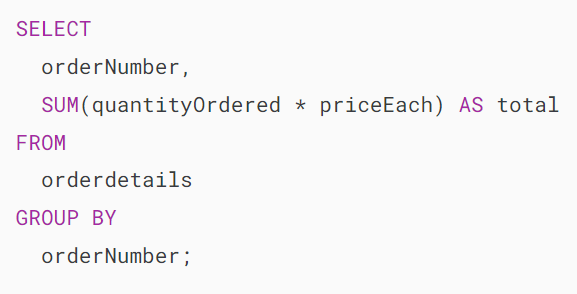


To get the total amount of all orders by status, you [join](https://www.mysqltutorial.org/mysql-basics/mysql-inner-join/)the orders table with the orderdetails table and use the SUM function to calculate the total amount:



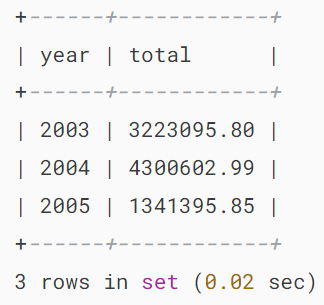
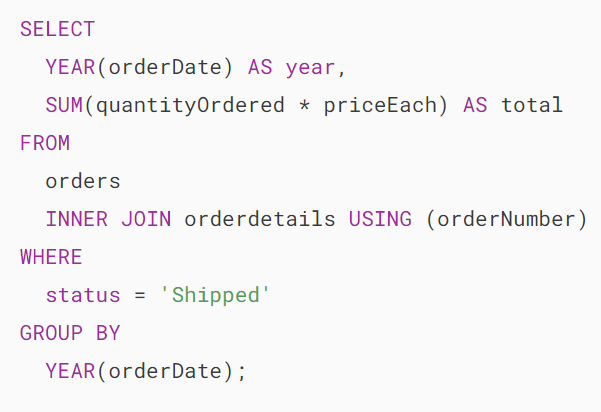


Similarly, the following query returns the order numbers and the total amount of each order.



### **3) MySQL GROUP BY with expression example**

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



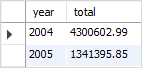
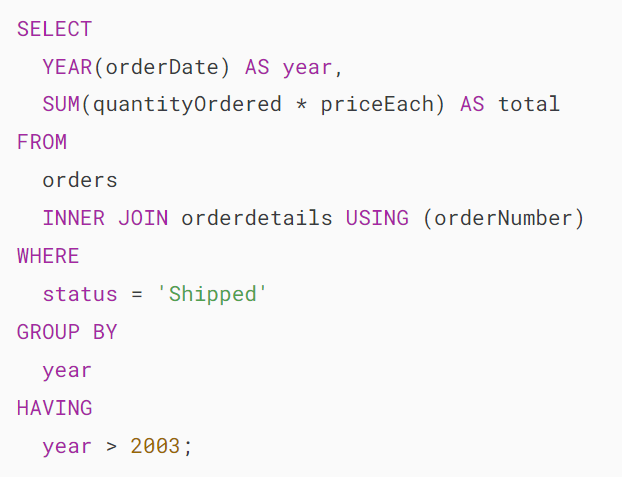
In this example, we used the [YEAR](https://www.mysqltutorial.org/mysql-year/) function to extract year data from order date ( orderDate) and included only orders with shipped status in the total sales.

Note that the expression in the SELECT clause must match the one in the GROUP BY clause.

### **4) 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-basics/mysql-having/) clause.

The following query uses the HAVING clause to select the total sales of the years after 2003.



### **5) Grouping by multiple columns**

The following query returns the year, order status, and the total order for each combination of year and order status by grouping rows into groups:

## 

## **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.

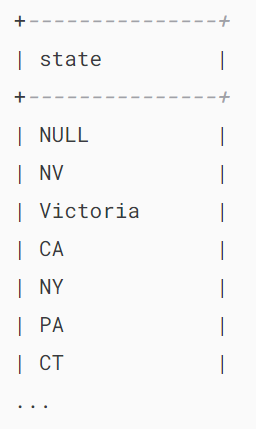
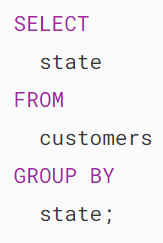
The following query is not valid in SQL standard:

## 

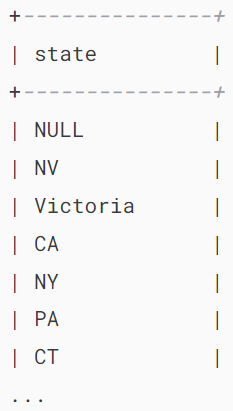
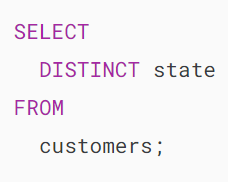
## **The GROUP BY clause vs. DISTINCT clause**

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

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



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



Notice that MySQL 8.0 or later removed the implicit sorting for the GROUP BY clause. Therefore, if you are using earlier versions, you will find that the result set with the GROUP BY clause is sorted.

## **Summary**

* Use the GROUP BY clause to group rows into subgroups.

# **MySQL HAVING**

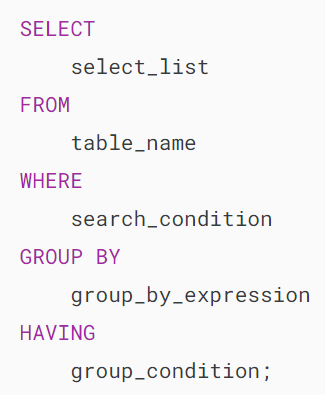
**Summary**: in this tutorial, you will learn how to use MySQL HAVING clause to specify a filter condition for groups of rows or aggregates.

## **Introduction to MySQL HAVING clause**

The HAVING clause is used with the [GROUP BY](https://www.mysqltutorial.org/mysql-basics/mysql-group-by/) clause to filter the groups based on a specified condition.

The HAVING clause allows you to apply a condition to the groups returned by the GROUP BY clause and only include groups that meet the specified condition.

Here’s the syntax of the HAVING clause:



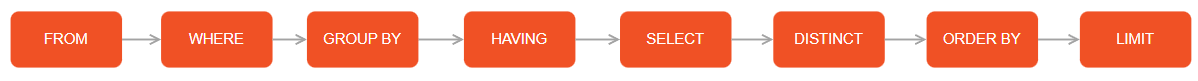
In this syntax, you specify a condition in the HAVING clause.

The HAVING clause evaluates each group returned by the GROUP BY clause. If the result is true (1), it includes the group in the result set.

The HAVING clause applies the condition to groups of rows, while the WHERE clause applies the condition to individual rows

If you omit the GROUP BY clause, the HAVING clause behaves like the [WHERE](https://www.mysqltutorial.org/mysql-basics/mysql-where/) clause.

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

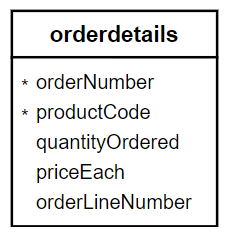


Note that the SQL standard specifies that the HAVING is evaluated before SELECT clause and after GROUP BY clause.

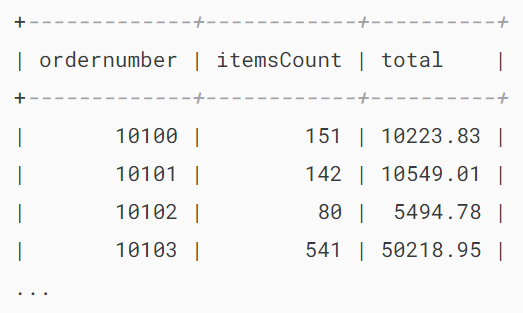
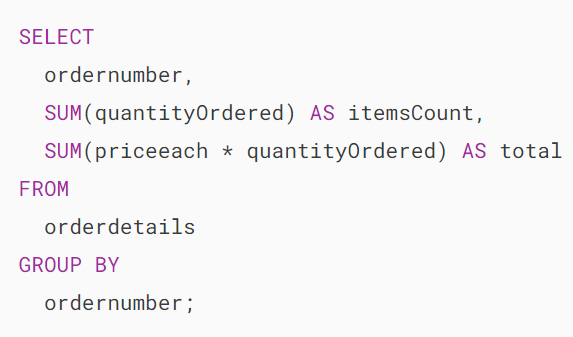
## **MySQL HAVING clause examples**

Let’s take some examples of using the HAVING clause to understand how it works.

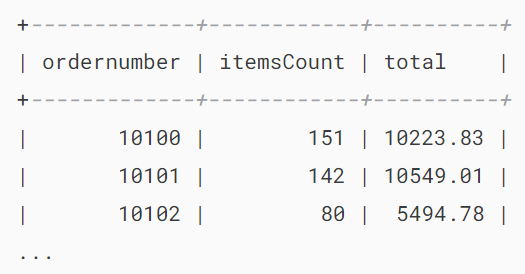
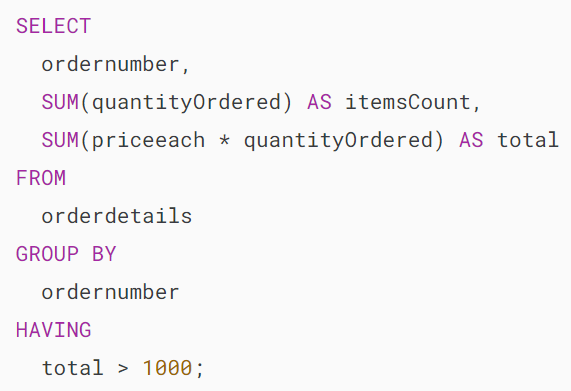
We’ll use the orderdetails table in the [sample database](https://www.mysqltutorial.org/getting-started-with-mysql/mysql-sample-database/) for the demonstration.



The following uses the GROUP BY clause to get order numbers, the number of items sold per order, and total sales for each from the orderdetails table:

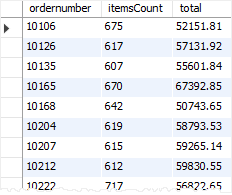
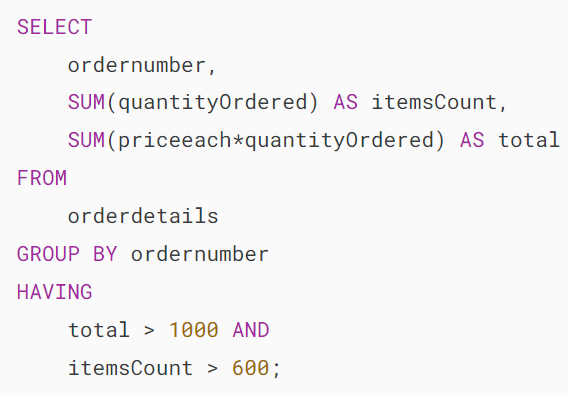


Now, you can find which order has total sales greater than 1000 by using the HAVING clause as follows:

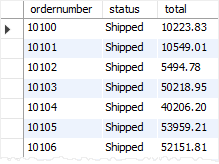
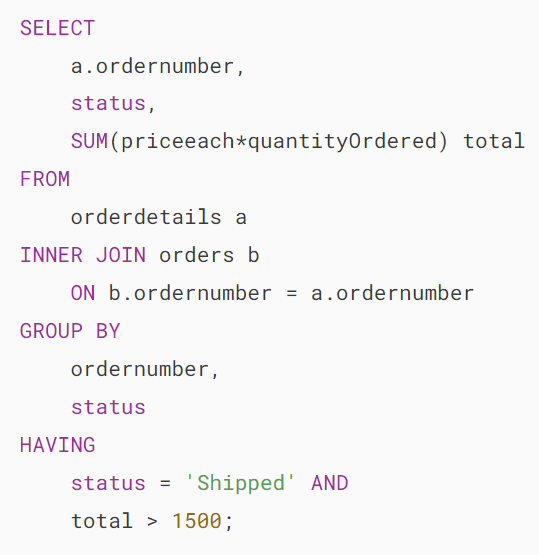


It’s possible to form a complex condition in the HAVING clause using logical operators such as [OR](https://www.mysqltutorial.org/mysql-basics/mysql-or/) and [AND](https://www.mysqltutorial.org/mysql-basics/mysql-and/).

The following example uses the HAVING clause to find orders that have total amounts greater than 1000 and contain more than 600 items:



Suppose that you want to find all orders that already shipped and have a total amount greater than 1500, you can [join](https://www.mysqltutorial.org/mysql-basics/mysql-join/) the orderdetails table with the orders table using the [INNER JOIN](https://www.mysqltutorial.org/mysql-basics/mysql-inner-join/) clause and apply a condition on status column and total aggregate as shown in the following query:



The HAVING clause is only useful when you use it with the GROUP BY clause to generate the output of the high-level reports.

For example, you can use the HAVING clause to answer questions like finding the number of orders this month, this quarter, or this year that have a total amount greater than 10K.

## **Summary**

* Use the MySQL HAVING clause with the GROUP BY clause to specify a filter condition for groups of rows or aggregates.

# **MySQL HAVING COUNT**

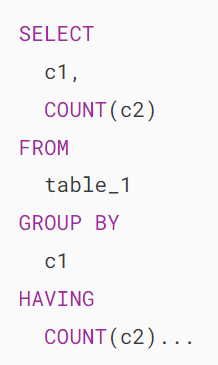
**Summary**: in this tutorial, you will learn how to use MySQL HAVING COUNT to filter groups based on the number of items in each group.

## **Introduction to MySQL HAVING COUNT**

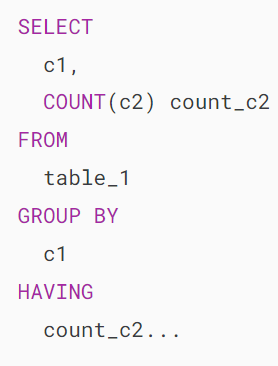
In MySQL, the [GROUP BY](https://www.mysqltutorial.org/mysql-basics/mysql-group-by/) clause organizes rows into groups. When you combine the GROUP BY clause with the [COUNT](https://www.mysqltutorial.org/mysql-aggregate-functions/mysql-count/) function, you will get both the groups and the number of items in each group.

To filter the groups based on the number of items in each group, you use the [HAVING](https://www.mysqltutorial.org/mysql-basics/mysql-having/) clause and the COUNT function.

The following illustrates the basic syntax for using the HAVING clause with the COUNT function to filter groups:



Note that you cannot assign an alias to the COUNT(column\_2) in the SELECT clause and use the column alias in the HAVING clause like this:



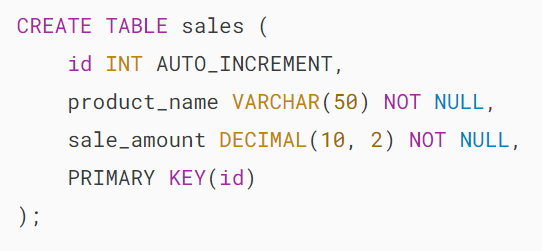
The reason is that MySQL evaluates the HAVING clause before the SELECT clause. Therefore, at the time MySQL evaluated the HAVING clause, it doesn’t know the column alias count\_c2 because it has not evaluated the SELECT clause yet.

## **MySQL HAVING COUNT example**

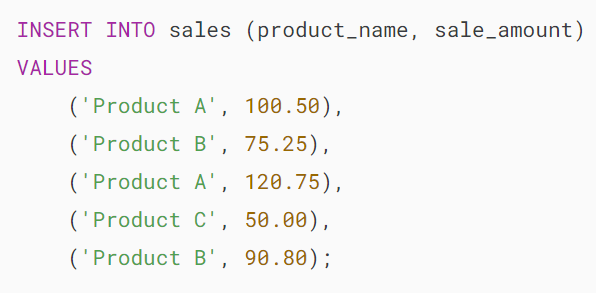
We’ll take some examples of using the HAVING COUNT clause

### **1) Simple HAVING COUNT example**

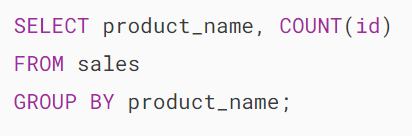
First, create a new table of sales that has three columns sale\_id, product\_name, and sale\_amount:



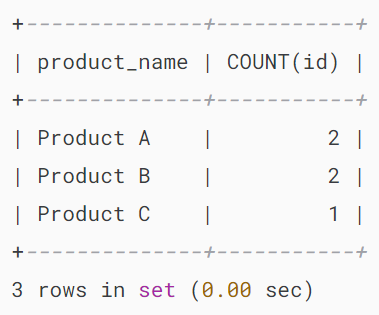
Second, insert some rows into the sales table:

.

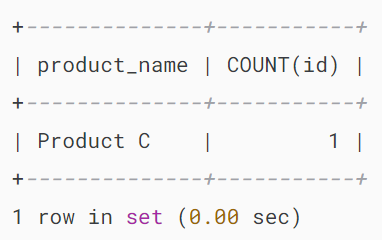
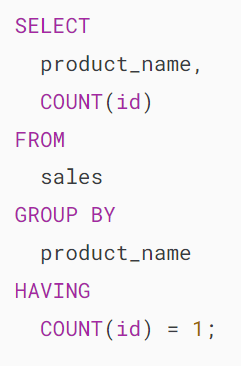
Third, count the number of sales per product:



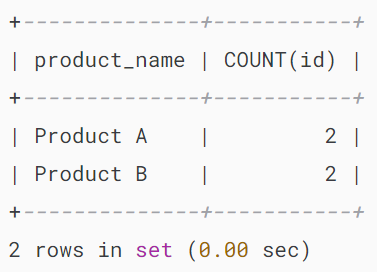
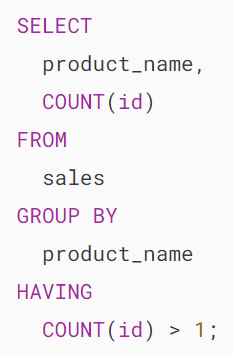
The query shows the product names and the number of sales for each product:



To find which product had one sale, you use the HAVING clause with the COUNT function as follows:

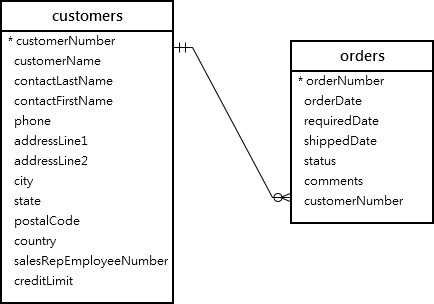


Similarly, you can find the products that had more than one sale by using the operator > in the HAVING clause:



### **2) Practical HAVING with COUNT example**

We’ll use the table customers and orders tables from the [sample database](https://www.mysqltutorial.org/getting-started-with-mysql/mysql-sample-database/):



The following query uses the HAVING clause with the COUNT function to get the customers who placed more than four orders:

## 

## **Summary**

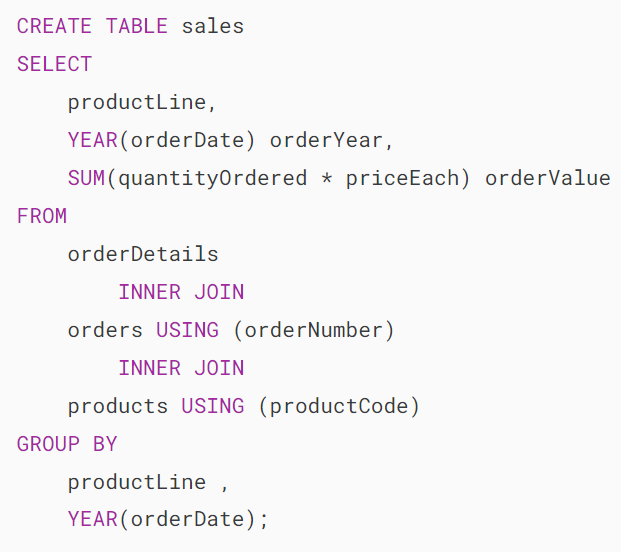
* Use the HAVING COUNT clause to filter groups by the number of items in each group.

# **MySQL ROLLUP**

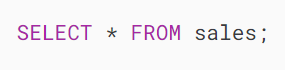
**Summary**: in this tutorial, you will learn how to use the MySQL ROLLUP clause to generate subtotals and grand totals.

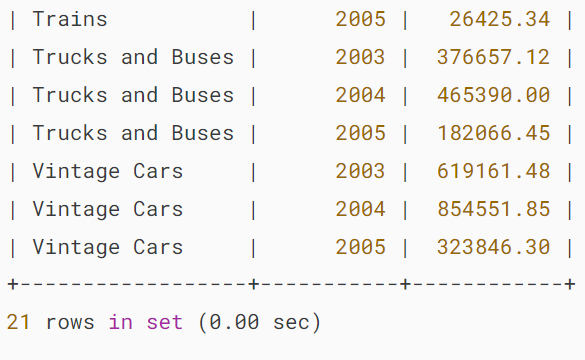
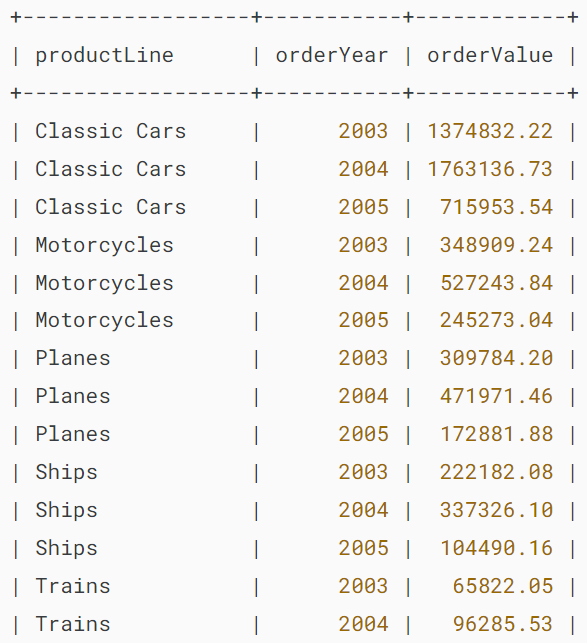
## **Setting up a sample table**

The following statement [creates a new table](https://www.mysqltutorial.org/mysql-basics/mysql-create-table/) named sales that stores the order values summarized by product lines and years. The data comes from the products, orders, and orderDetails tables in the [sample database](https://www.mysqltutorial.org/getting-started-with-mysql/mysql-sample-database/).



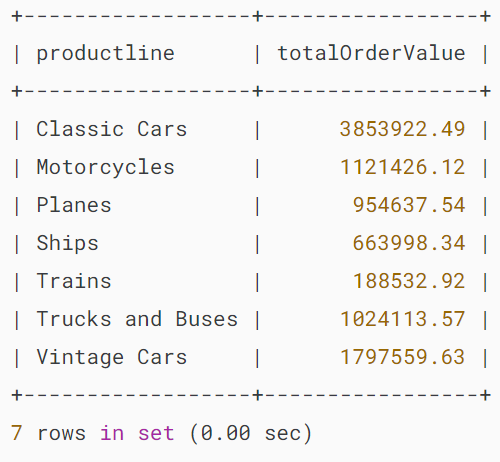
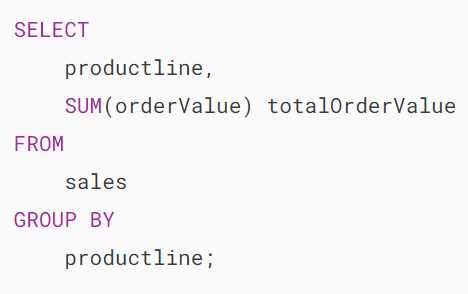
The following query returns all rows from the sales table:



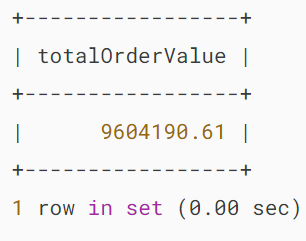
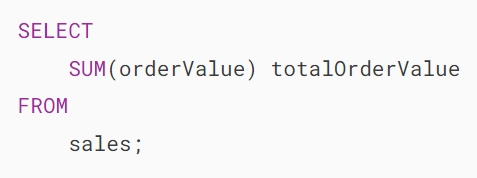


**MySQL ROLLUP Overview**

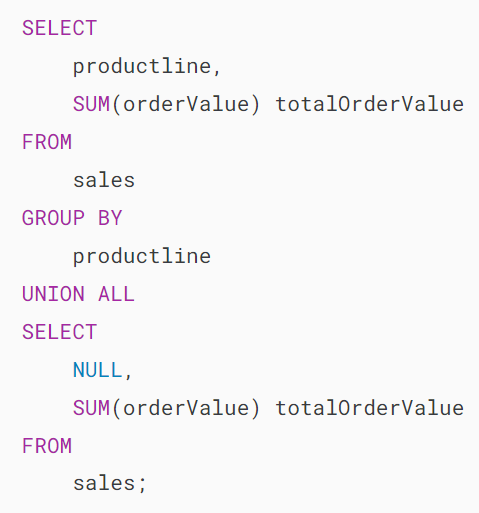
A grouping set is a set of columns to which you want to group. For example, the following query creates a grouping set denoted by (productline)



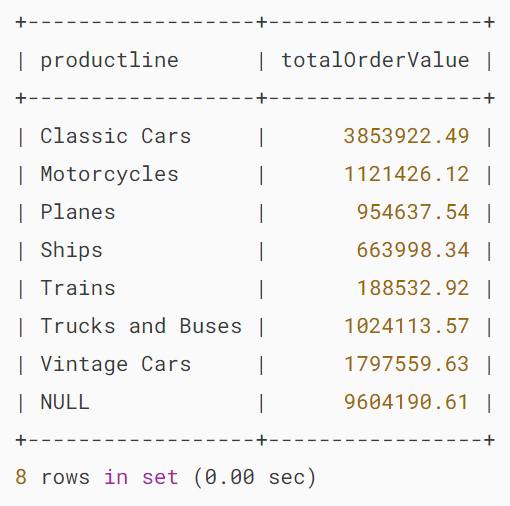
The following query creates an empty grouping set denoted by the ():



If you want to generate two or more grouping sets together in one query, you may use the [UNION ALL](https://www.mysqltutorial.org/mysql-basics/mysql-union/) operator as follows:



Here’s the query output:



Because the UNION ALL requires all queries to have the same number of columns, we added NULL in the select list of the second query to fulfill this requirement.

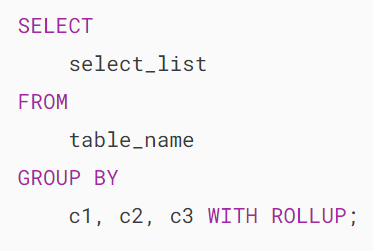
The NULL in the productLine column identifies the total super-aggregate line.

This query can generate the total order values by product lines and also the grand total row. However, it has two problems:

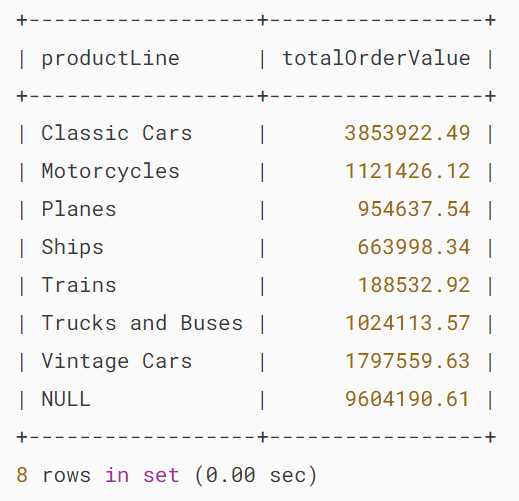
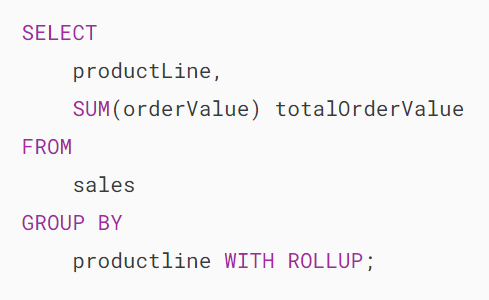
1. The query is quite lengthy.
2. The performance of the query may not be good since the database engine has to internally execute two separate queries and combine the result sets into one.

To fix these issues, you can use the ROLLUP clause.

The ROLLUP clause is an extension of the [GROUP BY](https://www.mysqltutorial.org/mysql-basics/mysql-group-by/) clause with the following syntax:



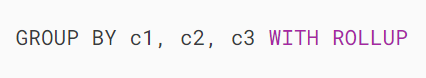
The ROLLUP generates multiple grouping sets based on the columns or expressions specified in the GROUP BY clause. For example:



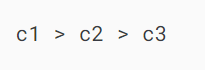
As clearly shown in the output, the ROLLUP clause generates not only the subtotals but also the grand total of the order values.

If you have more than one column specified in the GROUP BY clause, the ROLLUP clause assumes a hierarchy among the input columns.

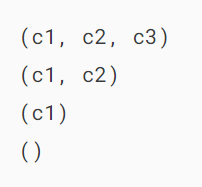
For example:



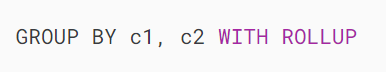
The ROLLUP assumes that there is the following hierarchy:



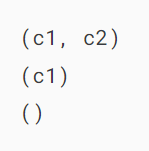
It generates the following grouping sets:



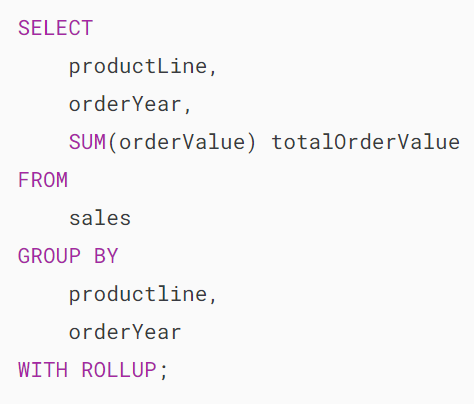
If you have two columns specified in the GROUP BY clause:

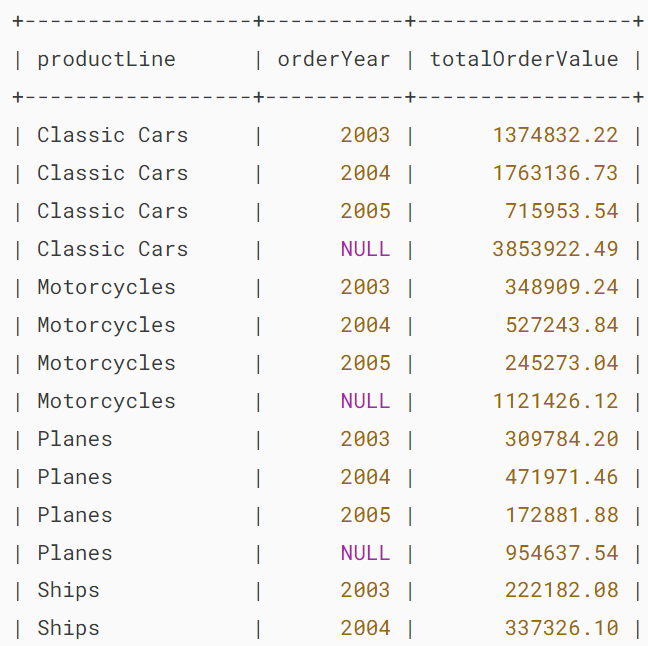


then the ROLLUP generates the following grouping sets:

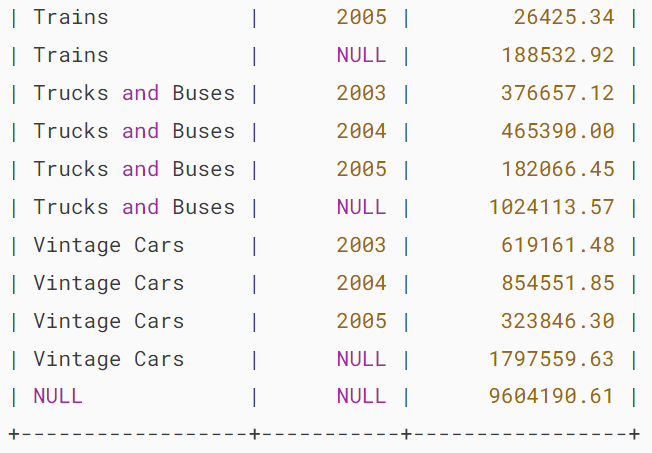


See the following query example:





..

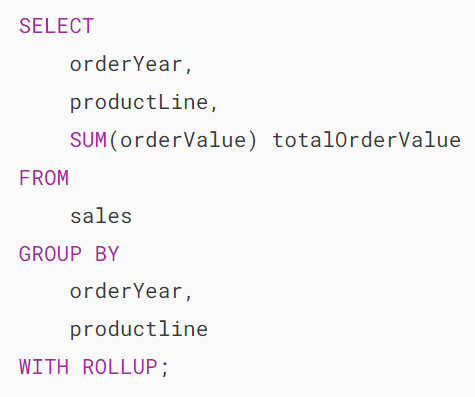


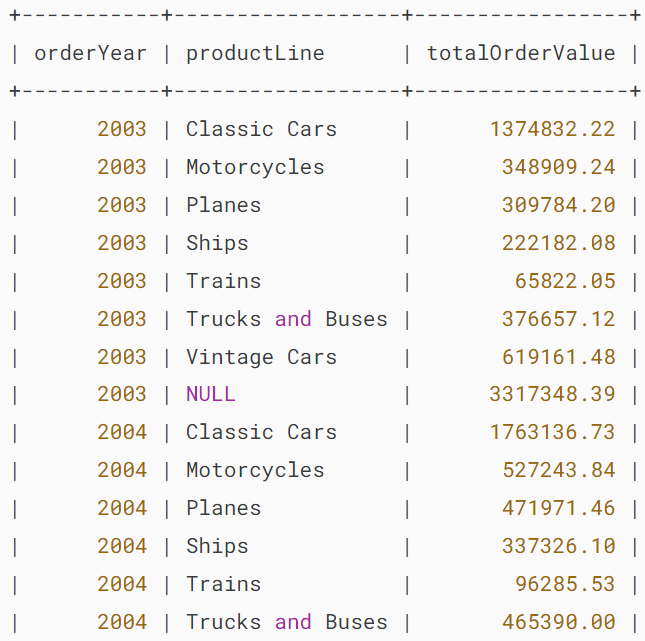
The ROLLUP generates the subtotal row every time the product line changes and the grand total at the end of the result.

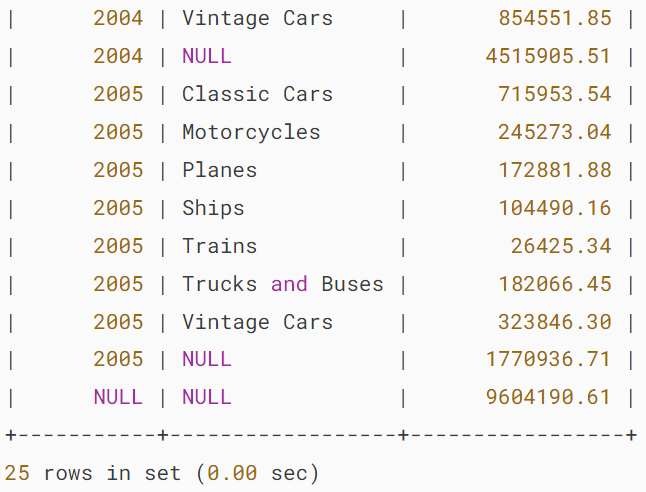
The hierarchy in this case is:



If you reverse the hierarchy, for example:







The ROLLUP generates the subtotal every time the year changes and the grand total at the end of the result set.

The hierarchy in this example is:

## 

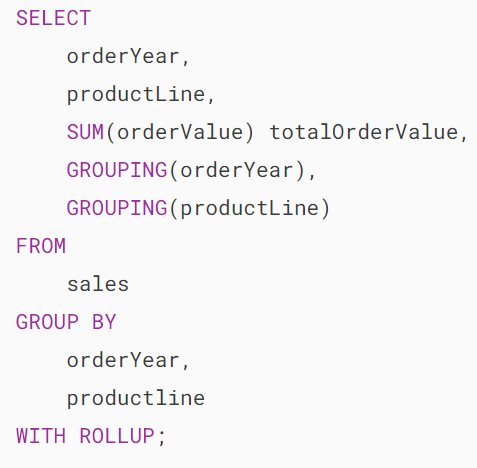
## **The GROUPING() function**

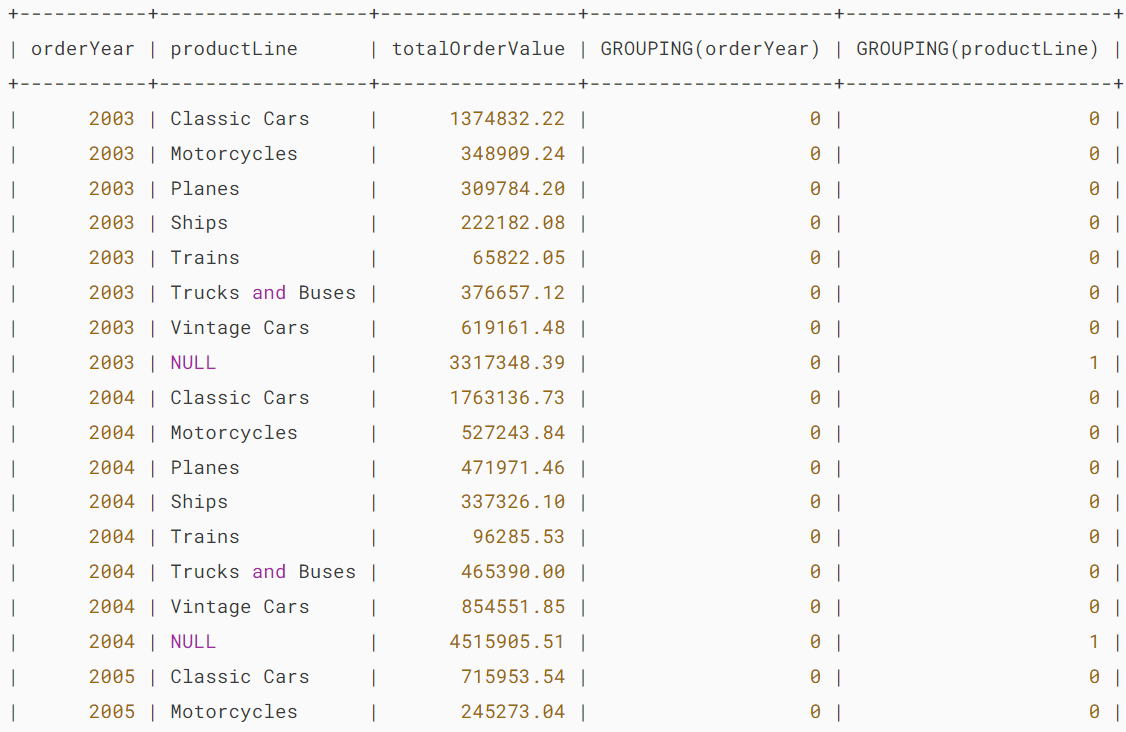
To check whether NULL in the result set represents the subtotals or grand totals, you use the GROUPING() function.

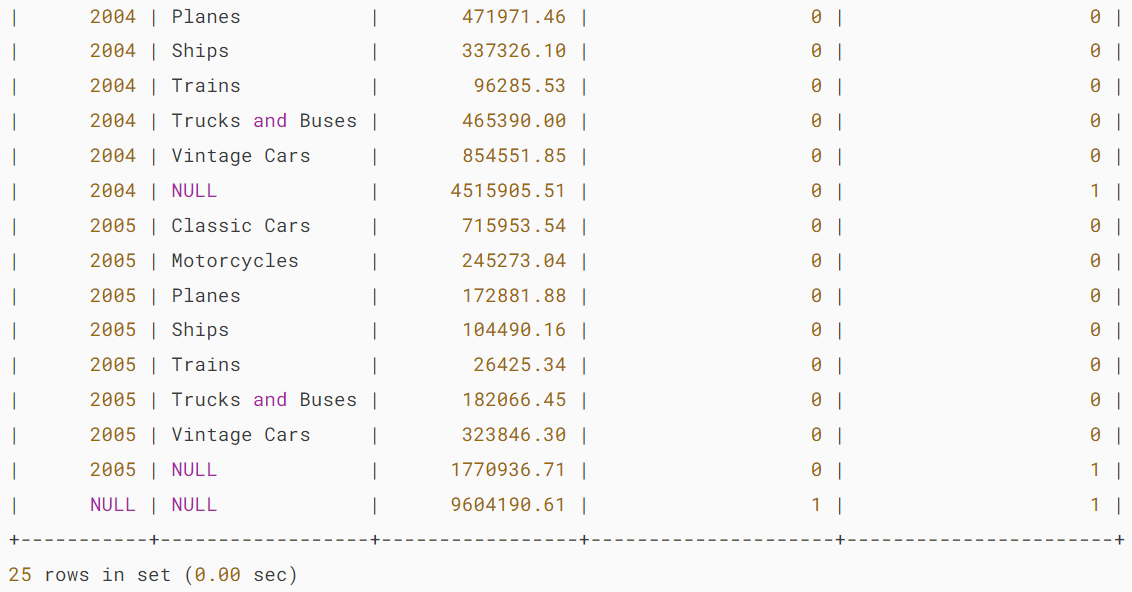
The GROUPING() function returns 1 when NULL occurs in a supper-aggregate row, otherwise, it returns 0.

The GROUPING() function can be used in the select list, HAVING clause, and (as of MySQL 8.0.12 ) ORDER BY clause.

Consider the following query:





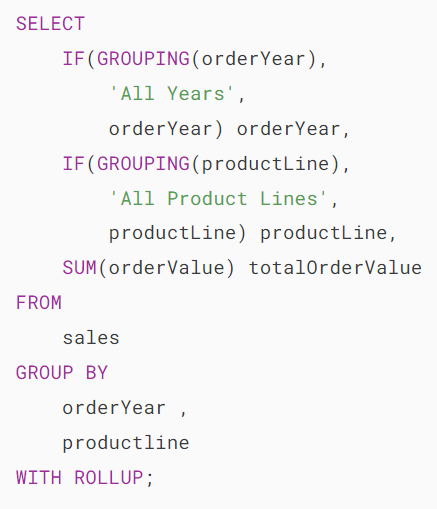


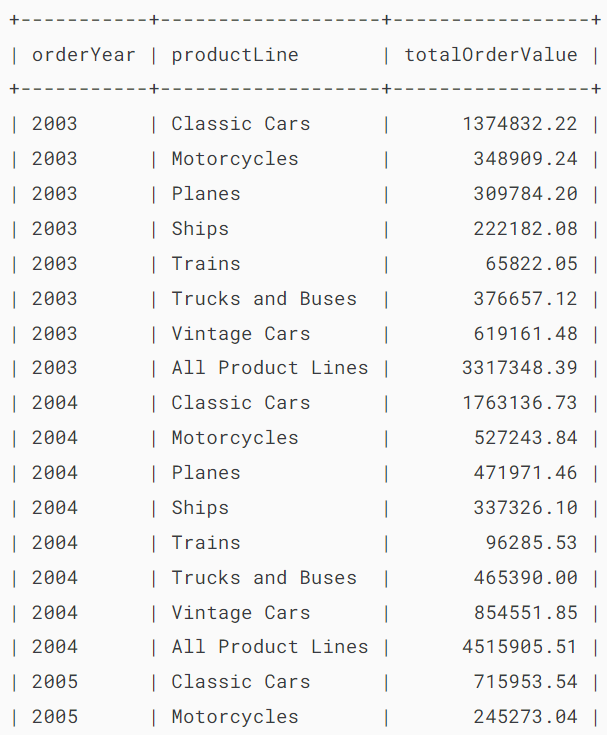
The GROUPING(orderYear) returns 1 when NULL in the orderYear column occurs in a super-aggregate row, 0 otherwise.

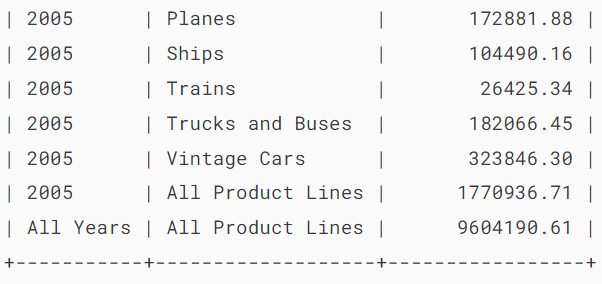
Similarly, the GROUPING(productLine) returns 1 when NULL in the productLine column occurs in a super-aggregate row, 0 otherwise.

We often use GROUPING() function to substitute meaningful labels for super-aggregate NULL values instead of displaying it directly.

The following example shows how to combine the [IF()](https://www.mysqltutorial.org/mysql-control-flow-functions/mysql-if-function/) function with the GROUPING() function to substitute labels for the super-aggregate NULL values in orderYear and productLine columns:







# **MySQL Subquery**

**Summary**: in this tutorial, you will learn how to use the MySQL subquery to write complex queries and understand the correlated subquery concept.

## **Introduction to the MySQL Subquery**

A MySQL subquery is a query nested within another query such as [SELECT](https://www.mysqltutorial.org/mysql-basics/mysql-select-from/), [INSERT](https://www.mysqltutorial.org/mysql-basics/mysql-insert/), [UPDATE](https://www.mysqltutorial.org/mysql-basics/mysql-update/)or [DELETE](https://www.mysqltutorial.org/mysql-basics/mysql-delete/). Also, a subquery can be nested within another subquery.

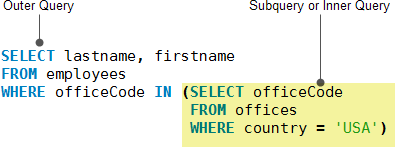
A MySQL subquery is called an inner query whereas the query that contains the subquery is called an outer query. A subquery can be used anywhere that expression is used and must be closed in parentheses.

For example, the following query uses a subquery to return the employees who work in the offices located in the USA.



In this example:

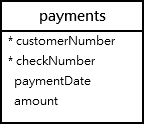
* The subquery returns all office codes of the offices located in the USA.
* The outer query selects the last name and first name of employees who work in the offices whose office codes are in the result set returned by the subquery.



When executing the query, MySQL evaluates the subquery first and uses the result of the subquery for the outer query.

## **Using a MySQL subquery in the WHERE clause**

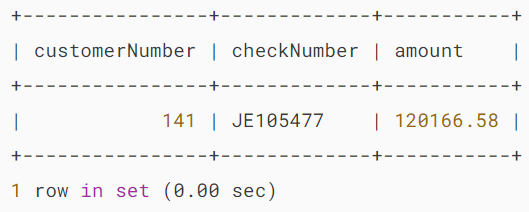
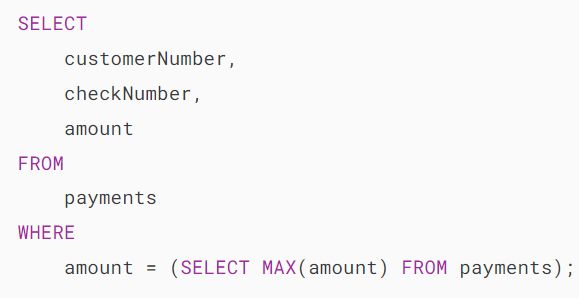
We will use the table payments in the [sample database](https://www.mysqltutorial.org/getting-started-with-mysql/mysql-sample-database/) for the demonstration.



### **MySQL subquery with comparison operators**

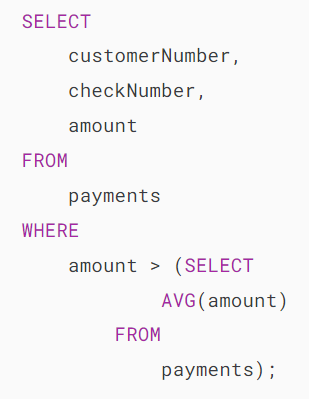
You can use comparison operators e.g., =, >, < to compare a single value returned by the subquery with the expression in the [WHERE](https://www.mysqltutorial.org/mysql-basics/mysql-where/) clause.

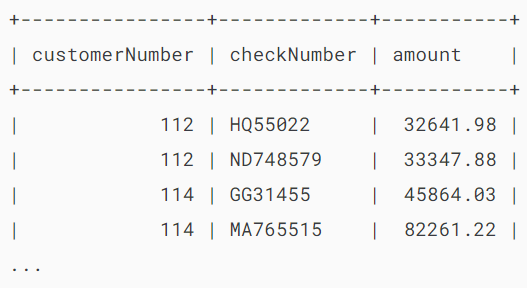
For example, the following query returns the customer who has the highest payment.



Besides the = operator, you can use other comparison operators such as greater than (>), greater than or equal to (>=) less than(<), and less than or equal to (<=).

For example, you can find customers whose payments are greater than the average payment using a subquery:





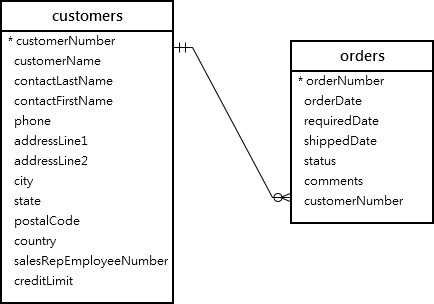
In this example:

* First, get the average payment by using a subquery.
* Then, select the payments that are greater than the average payment returned by the subquery in the outer query.

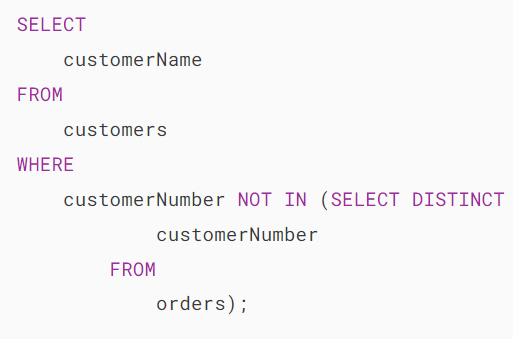
### **MySQL subquery with IN and NOT IN operators**

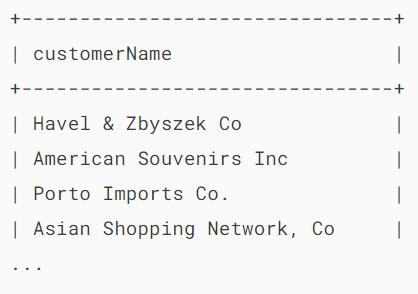
If a subquery returns more than one value, you can use other operators such as [IN](https://www.mysqltutorial.org/mysql-basics/mysql-in/) or [NOT IN](https://www.mysqltutorial.org/mysql-basics/mysql-in/) operator in the WHERE clause.

See the following customers and orders tables:



For example, you can use a subquery with NOT IN operator to find the customers who have not placed any orders as follows:

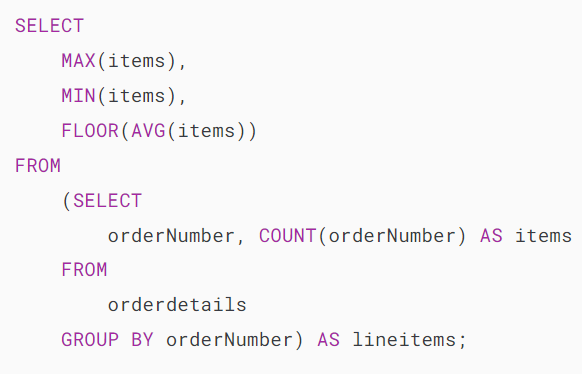


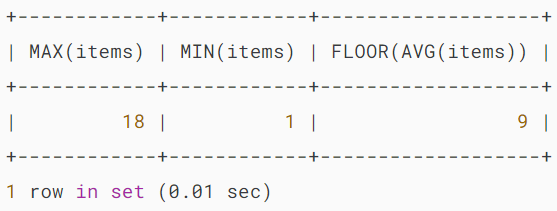


## **MySQL subquery in the FROM clause**

When you use a subquery in the FROM clause, the result set returned from a subquery is used as a [temporary table.](https://www.mysqltutorial.org/mysql-basics/mysql-temporary-table/) This table is referred to as a [derived table](https://www.mysqltutorial.org/mysql-basics/mysql-derived-table/) or materialized subquery.

The following subquery finds the [maximum](https://www.mysqltutorial.org/mysql-aggregate-functions/mysql-max-function/), [minimum,](https://www.mysqltutorial.org/mysql-aggregate-functions/mysql-min/)and [average](https://www.mysqltutorial.org/mysql-aggregate-functions/mysql-avg/)number of items in sale orders:

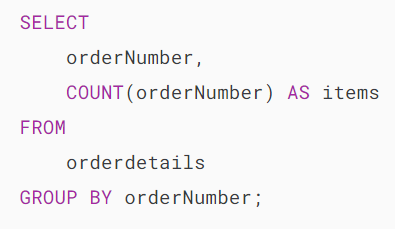




Note that the [FLOOR()](https://www.mysqltutorial.org/mysql-math-functions/mysql-floor/) is used to remove decimal places from the average values of items.

## **MySQL correlated subquery**

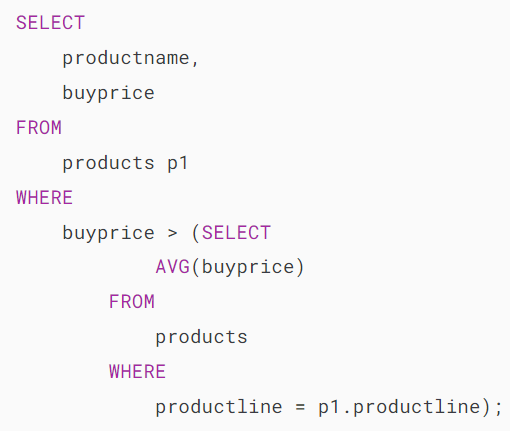
In the previous examples, you notice that a subquery is independent. It means that you can execute the subquery as a standalone query, for example:



Unlike a standalone subquery, a correlated subquery is a subquery that uses the data from the outer query. In other words, a correlated subquery depends on the outer query. A correlated subquery is evaluated once for each row in the outer query.

See the following products table from the [sample database](https://www.mysqltutorial.org/getting-started-with-mysql/mysql-sample-database/):

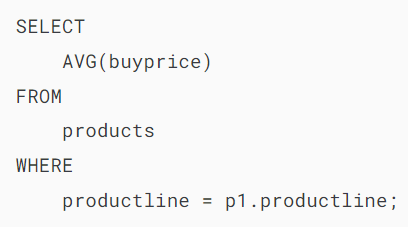
The following example uses a correlated subquery to select products whose buy prices are greater than the average buy price of all products in each product line.





In this example, both the outer query and correlated subquery reference the same products table. Therefore, we need to use a table alias p1 for the products table in the outer query.

Unlike a regular subquery, you cannot execute a correlated subquery independently like this. If you do so, MySQL doesn’t know the p1 table and will issue an error.

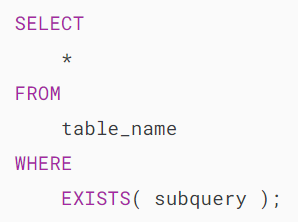


For each row in the products (or p1) table, the correlated subquery needs to execute once to get the average buy price of all products in the productline of that row.

If the buy price of the current row is greater than the average buy price returned by the correlated subquery, the query includes the row in the result set.

### **MySQL subquery with EXISTS and NOT EXISTS**

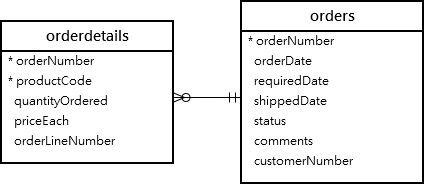
When a subquery is used with the [EXISTS](https://www.mysqltutorial.org/mysql-basicshttps:/www.mysqltutorial.org/mysql-basics/mysql-exists/) or [NOT EXISTS](https://www.mysqltutorial.org/mysql-basicshttps:/www.mysqltutorial.org/mysql-basics/mysql-exists/) operator, a subquery returns a Boolean value of TRUE or FALSE.  The following query illustrates a subquery used with the EXISTS operator:



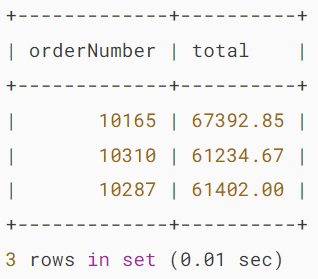
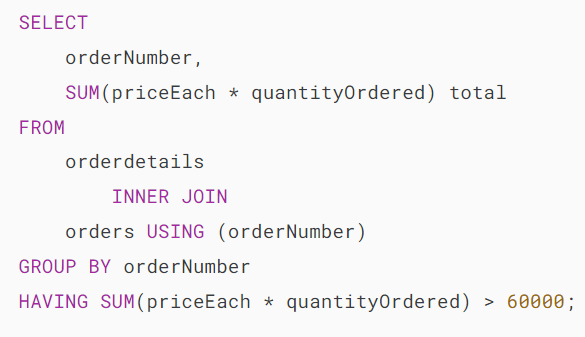
In the query above, if the subquery returns any rows, EXISTS subquery returns TRUE, otherwise, it returns FALSE.

The EXISTS and NOT EXISTS are often used in the correlated subqueries.

Let’s take a look at the orders and orderdetails tables from the [sample database](https://www.mysqltutorial.org/getting-started-with-mysql/mysql-sample-database/):

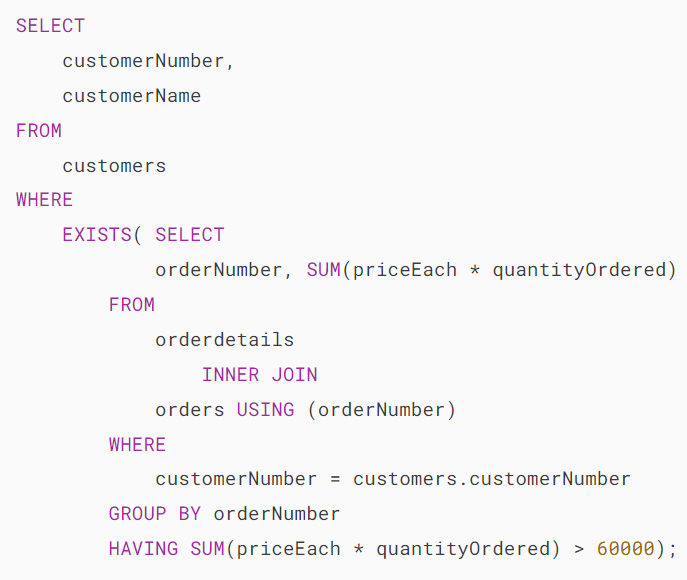


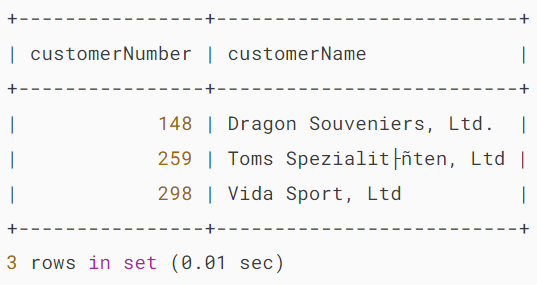
The following query finds sales orders whose total values are greater than 60K.



It returns 3 rows, meaning that there are three sales orders whose total values are greater than 60K.

You can use the query above as a correlated subquery to find customers who placed at least one sales order with a total value greater than 60K by using the EXISTS operator:





## **Summary**

* A subquery is a query nested within another query (or outer query).
* A correlated subquery depends on the outer query.