


Using group by on multiple columns

Asked 10 years, 11 months ago

Active 14 days ago

Viewed 1.3m times

 I understand the point of `GROUP BY x`.

1159 But how does `GROUP BY x, y` work, and what does it mean?



sql group-by multiple-columns



335

 Share Improve this question

Follow

 [janw](#)

4,359 5 19 41

edited Jul 22 '20 at 14:47

asked Mar 10 '10 at 23:11

 [Alex Gordon](#)

48.9k 273 598 965

2

You won't find it described as this question poses it. The GROUP BY clause can take one or more fields. GROUP BY customer; GROUP BY lastname, firstname; GROUP BY year, store, sku etc. – [Bill](#) Mar 10 '10 at 23:32 

4 Answers

Active

Oldest

Votes

 Group By X means **put all those with the same value for X in the one group.**

2224 Group By X, Y means **put all those with the same values for both X and Y in the one group.**

 To illustrate using an example, let's say we have the following table, to do with who is attending what subject at a university:



Table: Subject_Selection

Subject	Semester	Attendee
ITB001	1	John
ITB001	1	Bob
ITB001	1	Mickey
ITB001	2	Jenny
ITB001	2	James
MKB114	1	John
MKB114	1	Erica

When you use a `group by` on the subject column only; say:

```
select Subject, Count(*)
from Subject_Selection
group by Subject
```

https://stackoverflow.com/questions/2421388/using-group-by-on-multiple-columns

1/9

You will get something like:

```
+-----+-----+
| Subject | Count |
+-----+-----+
| ITB001  |      5 |
| MKB114  |      2 |
+-----+-----+
```

...because there are 5 entries for ITB001, and 2 for MKB114

If we were to group by two columns:

```
select Subject, Semester, Count(*)
from Subject_Selection
group by Subject, Semester
```

we would get this:

```
+-----+-----+-----+
| Subject | Semester | Count |
+-----+-----+-----+
| ITB001  |      1  |      3 |
| ITB001  |      2  |      2 |
| MKB114  |      1  |      2 |
+-----+-----+-----+
```

This is because, when we group by two columns, it is saying **"Group them so that all of those with the same Subject and Semester are in the same group, and then calculate all the aggregate functions (Count, Sum, Average, etc.) for each of those groups"**. In this example, this is demonstrated by the fact that, when we count them, there are **three** people doing ITB001 in semester 1, and **two** doing it in semester 2. Both of the people doing MKB114 are in semester 1, so there is no row for semester 2 (no data fits into the group "MKB114, Semester 2")

Hopefully that makes sense.

Share Improve this answer

Follow

edited Aug 2 '20 at 0:49



Bitswazsky

2,284 2 19 35

answered Mar 10 '10 at 23:24



Smashery

48.5k 30 90 123

14 @Smashery: So does this also mean that GROUP BY A,B is same as GROUP BY B,A ? – tumchaaditya Sep 26 '14 at 18:08

32 Yes, it does. I can't say for certain whether they are as efficient as each other, but they will give the same result, yes. – Smashery Sep 29 '14 at 0:10

3 May here should be added that there is a difference between GROUP BY a, b and GROUP BY a AND b since the second one only lists grouped items with exactly the same content and no "undergroups". In this case the output would be same like the first one. – Dwza Mar 4 '15 at 15:06

5 I would like to add that the order in which you group by the columns does not matter. In the above example group by Semester, Subject would have given the same result – user2441441 Sep 29 '15 at 21:07

- 3 well, group by a, b and group by b, a do NOT return the same result - the rows are displayed in a different order – [fanny](#) Oct 3 '18 at 9:14
-



43



The `GROUP BY` clause is used in conjunction with the aggregate functions to group the result-set by one or more columns. e.g.:

```
-- GROUP BY with one parameter:
SELECT column_name, AGGREGATE_FUNCTION(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name;

-- GROUP BY with two parameters:
SELECT
    column_name1,
    column_name2,
    AGGREGATE_FUNCTION(column_name3)
FROM
    table_name
GROUP BY
    column_name1,
    column_name2;
```

Remember this order:

1. `SELECT` (is used to select data from a database)
2. `FROM` (clause is used to list the tables)
3. `WHERE` (clause is used to filter records)
4. `GROUP BY` (clause can be used in a `SELECT` statement to collect data across multiple records and group the results by one or more columns)
5. `HAVING` (clause is used in combination with the `GROUP BY` clause to restrict the groups of returned rows to only those whose the condition is `TRUE`)
6. `ORDER BY` (keyword is used to sort the result-set)

You can use all of these if you are using aggregate functions, and this is the order that they must be set, otherwise you can get an error.

Aggregate Functions are:

`MIN()` returns the smallest value in a given column

`MAX()` returns the maximum value in a given column.

`SUM()` returns the sum of the numeric values in a given column

`AVG()` returns the average value of a given column

`COUNT()` returns the total number of values in a given column

`COUNT(*)` returns the number of rows in a table

SQL script examples about using aggregate functions:

Let's say we need to find the sale orders whose total sale is greater than \$950. We combine the `HAVING` clause and the `GROUP BY` clause to accomplish this:

```
SELECT
    orderId, SUM(unitPrice * qty) Total
FROM
    OrderDetails
GROUP BY orderId
HAVING Total > 950;
```

Counting all orders and grouping them customerID and sorting the result ascendant. We combine the `COUNT` function and the `GROUP BY`, `ORDER BY` clauses and `ASC`:

```
SELECT
    customerId, COUNT(*)
FROM
    Orders
GROUP BY customerId
ORDER BY COUNT(*) ASC;
```

Retrieve the category that has an average Unit Price greater than \$10, using `AVG` function combine with `GROUP BY` and `HAVING` clauses:

```
SELECT
    categoryName, AVG(unitPrice)
FROM
    Products p
INNER JOIN
    Categories c ON c.categoryId = p.categoryId
GROUP BY categoryName
HAVING AVG(unitPrice) > 10;
```

Getting the less expensive product by each category, using the `MIN` function in a subquery:

```
SELECT categoryId,
    productId,
    productName,
    unitPrice
FROM Products p1
WHERE unitPrice = (
    SELECT MIN(unitPrice)
    FROM Products p2
    WHERE p2.categoryId = p1.categoryId)
```

The following statement groups rows with the same values in both ***categoryId*** and ***productId*** columns:

```
SELECT
    categoryId, categoryName, productId, SUM(unitPrice)
FROM
    Products p
```

INNER JOINCategories c **ON** c.categoryId = p.categoryId**GROUP BY** categoryId, productId

edited Dec 28 '20 at 15:20

answered Dec 15 '16 at 22:14

Share Improve this answer

Follow

**S. Mayol****1,756** 2 25 25

-
- 3 but where do we put the 2 columns, how to aggregate based on 2/more columns is the question – [Chaitanya Bapat](#) Nov 17 '17 at 22:13
-
- 1 This doesn't even remotely answer the question... The question here is how to achieve "chained grouping" of "subject" and "semester" at the same time, as explained in the given example... – [MahNas92](#) May 20 '19 at 10:22
-
- 1 The last example shows you how to put 2 columns using aggregate function. @ChaitanyaBapat – [S. Mayol](#) Jan 7 at 20:47
-



As you understand the point of **GROUP BY** x. Here x is table Column_Name. This is Call **GROUP By** Table Single Column_Name. In **GROUP By** x,y. This means that the table **GROUP By** Two Column_Names x and y. This is a called "**multiple grouping columns**". I can explain it in more detail below.



In **SQL** the **GROUP BY** Statement is used to arrange identical data into groups with the help of some functions. Example:- If a particular column has the same values in different rows then it will arrange these rows in a group. This **GROUP BY** clause follows the **WHERE** clause in a **SELECT** statement and precedes the **ORDER BY** clause. These are the important points:-

1. **GROUP BY** clause is used with the **SELECT** statement.
2. In the query, **GROUP BY** clause is placed after the **WHERE** clause.
3. In the query, **GROUP BY** clause is placed before **ORDER BY** clause if used any.

The basic syntax of a **GROUP BY** clause is shown in the following code block. The **GROUP BY** clause must follow the conditions in the **WHERE** clause and must precede the **ORDER BY** clause if one is used.

```
SELECT column1, column2
FROM table_name
WHERE [ conditions ]
GROUP BY column1, column2
ORDER BY column1, column2
```

1. **function_name**:- Name of the function used for example, MIN(), MAX(), SUM(), AVG(), COUNT().
2. **table_name**:- Name of the table.
3. **condition**:- Condition used.

A **GROUP BY** clause can contain two or more columns—or, in other words, a grouping can consist of two or more columns. We illustrate this with two examples.

In the **First Example** We GROUP BY single column, to place all the rows with same value of only that particular column in one group. This is the Employee Table Details:-

SI NO	NAME	SALARY	AGE
1	Harsh	2000	19
2	Dhanraj	3000	20
3	Ashish	1500	19
4	Harsh	3500	19
5	Ashish	1500	19

Consider the query as shown below for **GROUP BY single column** in Employee Table:-

```
SELECT NAME, SUM(SALARY)
FROM Employee
GROUP BY NAME;
```

The Output of the GROUP BY Single Column is:-

NAME	SALARY
Ashish	3000
Dhanraj	3000
Harsh	5500

As We can see in the above output, the rows with duplicate NAMES are grouped under same NAME and their corresponding SALARY is the sum of the SALARY of duplicate rows. The SUM() function of SQL is used here to calculate the sum.

In the **Second Example** we GROUP BY multiple columns. This means to place all the rows with same values of both the columns column1 and column2 in one group. See the Student Table Below:-

SUBJECT	YEAR	NAME
English	1	Harsh
English	1	Pratik
English	1	Ramesh
English	2	Ashish
English	2	Suresh
Mathematics	1	Deepak
Mathematics	1	Sayan

Consider the below query for **GROUP BY multiple columns** in Student Table:-

```
SELECT SUBJECT, YEAR, Count(*)
FROM Student
GROUP BY SUBJECT, YEAR;
```

The Output of the GROUP BY Multiple Column is:-

SUBJECT	YEAR	Count
English	1	3
English	2	2
Mathematics	1	2

As We can see in the above output the students with both same SUBJECT and YEAR are placed in same group. And those whose only SUBJECT is same but not YEAR belongs to different groups. So here we have grouped the table according to two columns or more than one column.

As with **ORDER BY**, We can substitute numbers for column names in the **GROUP BY** clause. It's generally recommended to do this only when you're grouping many columns, or if something else is causing the text in the **GROUP BY** clause to be excessively long. These are some basic reason to use **GROUP BY** in SQL.

1. The **GROUP BY** Clause **SQL** is used to group rows with same values.
2. The **GROUP BY** Clause is used together with the **SQL SELECT** statement.
3. The **SELECT** statement used in the **GROUP BY** clause can only be used contain column names, aggregate functions, constants and expressions.
4. **SQL** Having Clause is used to restrict the results returned by the **GROUP BY** clause.
5. **MYSQL GROUP BY** Clause is used to collect data from multiple records and returned record set by one or more columns.

This is the [GROUP BY](#) SQL Official Documentation to make more understanding about the GROUP BY.

Share Improve this answer Follow

answered Jan 21 at 5:37



Raksha Saini

1,923 16 30

In simple English from **GROUP BY** with two parameters what we are doing is **looking for similar value pairs and get the count to a 3rd column.**

Look at the following example for reference. Here I'm using [International football results from 1872 to 2020](#)

```
+-----+-----+-----+-----+-----+-----+-----+
---+
|      _c0|      _c1|      _c2|_c3|_c4|      _c5|      _c6|      _c7|
|_c8|
```

```

+-----+-----+-----+-----+-----+-----+-----+-----+
---+
|1872-11-30|      Scotland| England| 0| 0|Friendly| Glasgow|
Scotland|FALSE|
|1873-03-08|      England|Scotland| 4| 2|Friendly| London|
England|FALSE|
|1874-03-07|      Scotland| England| 2| 1|Friendly| Glasgow|
Scotland|FALSE|
|1875-03-06|      England|Scotland| 2| 2|Friendly| London|
England|FALSE|
|1876-03-04|      Scotland| England| 3| 0|Friendly| Glasgow|
Scotland|FALSE|
|1876-03-25|      Scotland|  Wales| 4| 0|Friendly| Glasgow|
Scotland|FALSE|
|1877-03-03|      England|Scotland| 1| 3|Friendly| London|
England|FALSE|
|1877-03-05|      Wales|Scotland| 0| 2|Friendly| Wrexham|
Wales|FALSE|
|1878-03-02|      Scotland| England| 7| 2|Friendly| Glasgow|
Scotland|FALSE|
|1878-03-23|      Scotland|  Wales| 9| 0|Friendly| Glasgow|
Scotland|FALSE|
|1879-01-18|      England|  Wales| 2| 1|Friendly| London|
England|FALSE|
|1879-04-05|      England|Scotland| 5| 4|Friendly| London|
England|FALSE|
|1879-04-07|      Wales|Scotland| 0| 3|Friendly| Wrexham|
Wales|FALSE|
|1880-03-13|      Scotland| England| 5| 4|Friendly| Glasgow|
Scotland|FALSE|
|1880-03-15|      Wales| England| 2| 3|Friendly| Wrexham|
Wales|FALSE|
|1880-03-27|      Scotland|  Wales| 5| 1|Friendly| Glasgow|
Scotland|FALSE|
|1881-02-26|      England|  Wales| 0| 1|Friendly|Blackburn|
England|FALSE|
|1881-03-12|      England|Scotland| 1| 6|Friendly| London|
England|FALSE|
|1881-03-14|      Wales|Scotland| 1| 5|Friendly| Wrexham|
Wales|FALSE|
|1882-02-18|Northern Ireland| England| 0| 13|Friendly| Belfast|Republic of
Ireland|FALSE|
+-----+-----+-----+-----+-----+-----+-----+-----+
---+

```

And now I'm going to group by similar country(column `_c7`) and tournament(`_c5`) value pairs by `GROUP BY` operation,

```
SELECT `_c5`,`_c7`,count(*) FROM res GROUP BY `_c5`,`_c7`
```

```

+-----+-----+-----+
|          _c5|          _c7|count(1)|
+-----+-----+-----+
|      Friendly| Southern Rhodesia|      11|
|      Friendly|      Ecuador|      68|
|African Cup of Na...|      Ethiopia|      41|
|Gold Cup qualific...|Trinidad and Tobago|       9|
|AFC Asian Cup qua...|      Bhutan|       7|
|African Nations C...|      Gabon|       2|
|      Friendly|      China PR|     170|
|FIFA World Cup qu...|      Israel|      59|
|FIFA World Cup qu...|      Japan|      61|
|UEFA Euro qualifi...|      Romania|      62|
|AFC Asian Cup qua...|      Macau|       9|
|      Friendly|      South Sudan|       1|
|CONCACAF Nations ...|      Suriname|       3|

```


Copa Newton	Argentina	12
Friendly	Philippines	38
FIFA World Cup qu...	Chile	68
African Cup of Na...	Madagascar	29
FIFA World Cup qu...	Burkina Faso	30
UEFA Nations League	Denmark	4
Atlantic Cup	Paraguay	2

Explanation: The meaning of the first row is there were 11 Friendly tournaments held on Southern Rhodesia in total.

Note: Here it's mandatory to use a counter column in this case.

Share Improve this answer Follow

answered Dec 26 '20 at 14:33



[Indrajith Ekanayake](#)

2,190 10 21 45



Highly active question. Earn 10 reputation in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.