SQL for Citizen Data Scientists

SQL GROUP BY Statement - Learn By Example

SETScholars & WACAMLDS

People are less interested in viewing raw data; Instead, people engaging in data analysis often seek to manipulate raw data to fit their needs. Examples of common data manipulations include:

1. Generating totals, such as total sales for each geographic region
2. Finding outliers, such as the top performing employee in each department
3. Determining frequencies, such as the number of products sold by each store

To answer these types of queries, you will need to divide the data into logical sets by grouping them and then perform aggregate calculations on each group.

Such groups can be created using the **GROUP BY** clause in your SELECT statement. The GROUP BY clause instructs the DBMS to group rows together by one or more columns or expressions.

Syntax

Here is a syntax of the GROUP BY clause.

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s);
```

Keep in mind that the GROUP BY clause must come after any WHERE clause and before any ORDER BY clause.

Sample Table

To help you better understand the examples, and enable you to follow along with the tutorial, we are going to use the following sample table.

This table is part of an 'Employee Management System' that contains basic information about employees.

ID	Name	City	Job	Salary	HireDate
1	Bob	New York	Manager	60000	2012-03-07
2	Kim	Chicago	Manager	55000	2014-04-25
3	Eve	New York	Developer	32000	2015-03-11
4	Max	New York	Janitor	9000	2015-01-15
5	Joe	Chicago	Developer	30000	2016-10-05
6	Amy	New York	Developer	31000	2013-05-13
7	Ray	New York	Developer	31500	2016-01-15
8	Sam	Chicago	Janitor	10000	2012-02-10
9	Liv	Chicago	Developer	30000	2014-08-10
10	Ian	New York	Janitor	8500	2018-01-12

Single-Column Grouping

Single-column groups are the simplest and most frequently used type of grouping. For example, if you want to find the number of employees hired for a specific job title, you need to group the data by a 'Job' column, such as:

```
SELECT Job, COUNT(*) AS emp_count
FROM Employees
GROUP BY Job;
```

Job	emp_count
Developer	5
Janitor	3
Manager	2

The above query specifies two columns, `job` which has different job titles and `emp_count` which is a calculated field created using the `COUNT(*)` function. The GROUP BY clause instructs the DBMS to group the data by job and then count the number of employees for each group.

Multicolumn Grouping

The GROUP BY clause can contain more than one column. This enables you to nest groups, giving you more control over how data is grouped.

For example, let's say you want to find the employee count not just for each job title, but for both job titles and cities. The following example shows how you can accomplish this:

```
SELECT City, Job, COUNT(*) AS emp_count
FROM Employees
GROUP BY City, Job;
```

City	Job	emp_count
Chicago	Developer	2
Chicago	Janitor	1
Chicago	Manager	1
New York	Developer	3
New York	Janitor	2
New York	Manager	1

This version of the query generates 6 rows, one for each combination of 'City' and 'Job' found in the 'Employees' table.

Note that along with the 'Job' column, the 'City' column is added to the GROUP BY clause. This is because in SQL, every column in your SELECT statement except the column created via the aggregate function must be present in the GROUP BY clause.

Generating Rollups

Sometimes it is beneficial to obtain summary totals within groups. SQL provides such functionality using the ROLLUP expression. It is used to get subtotals, or what is commonly referred to as super-aggregate rows plus a grand total row.

Extending the previous example, let's say that along with the employee count for each city-job combination, you also want the total count for each distinct city.

In this case you could run an additional query and merge the results or you could use the WITH ROLLUP option to have the DBMS do the work for you. Here's the extension of the previous query using WITH ROLLUP in the GROUP BY clause:

```
SELECT City, Job, COUNT(*) AS emp_count
FROM Employees
GROUP BY City, Job WITH ROLLUP;
```

City	Job	emp_count
Chicago	Developer	2
Chicago	Janitor	1
Chicago	Manager	1
Chicago	*NULL*	4
New York	Developer	3
New York	Janitor	2
New York	Manager	1
New York	*NULL*	6
NULL	*NULL*	10

There are now three additional rows (highlighted) in the result set, one for each of the two distinct cities and one for the grand total. For the two city rollups, a NULL value is provided for the 'Job' column, since the rollup is being performed across all job titles. Whereas for the grand total row, a NULL value is provided for both the 'City' and 'Job' columns.

Looking at the fourth and eighth line of output, for example, you will find that a total of 4 employees work from Chicago and 6 from New York. Whereas the last line of the output shows that a total of 10 employees work for the company.

Grouping via Expressions

In addition to using columns, you can also use expressions to group data. It is important to note that, if an expression is used in the SELECT, that same expression must be specified in GROUP BY clause.

For example, consider the following query, which groups employees by the year they began working for the company:

```
SELECT YEAR(HireDate) AS year, COUNT(*) AS emp_count
FROM Employees
GROUP BY YEAR(HireDate);
```

year	emp_count
2012	2
2013	1
2014	2
2015	2
2016	2
2018	1

This query uses the `YEAR()` function to construct a fairly simple expression, which returns only the year part of the date, to group the rows in an employee table.

Group Filter Conditions

After the data is grouped, you may want to apply a filter condition to include or exclude certain groups. Although your first instinct may be to use the WHERE statement, it won't really work because WHERE filters the records, but not the groups. SQL provides another clause, where you can place these types of filter conditions – **the HAVING clause**.

In the very first example of this tutorial, we find out the number of employees hired for a specific job title. Let's extend that query and get the list of jobs for which less than five employees are hired.

```
SELECT Job, COUNT(*) AS emp_count
FROM Employees
GROUP BY Job
HAVING COUNT(*) < 5;
```

Job	emp_count
Janitor	3
Manager	2

The first three lines of this query are similar to the first example. The final line adds a HAVING clause that filters on those groups with a `COUNT(*) < 5`.

Read in detail about the HAVING clause here.

How Nulls Are Handled

If the grouping column contains a row with a NULL value, NULL will be returned as a group.

For example, suppose some employees in the 'Employees' table are not assigned any job (assigned to NULL):

ID	Name	Job
1	Bob	Manager
2	Kim	Developer
3	Ray	*NULL*
4	Max	Janitor
5	Joe	Developer
6	Amy	*NULL*
7	Eve	Developer
8	Sam	Janitor

Now let's group the table by 'Job' column:

```
SELECT Job, COUNT(*) AS emp_count
FROM Employees
GROUP BY Job;
```

Job	emp_count
NULL	2
Developer	3
Janitor	2
Manager	1

As you can see there are several rows with NULL values in the 'Job' column, they are all grouped together and included at the top of the result set.