SQL GROUP BY

Summary: in this tutorial, you will learn how to use SQL GROUP BY clause to group rows based on one or more columns.

Introduction to SQL GROUP BY clause

Grouping is one of the most important tasks that you have to deal with while working with the databases. To group rows into groups, you use the GROUP BY clause.

The GROUP BY clause is an optional clause of the SELECT statement that combines rows into groups based on matching values in specified columns. One row is returned for each group.

You often use the GROUP BY in conjunction with an aggregate function such as MIN, MAX, AVG, SUM, or COUNT to calculate a measure that provides the information for each group.

The following illustrates the syntax of the GROUP BY clause.

```
SELECT
        column1,
        column2,
        AGGREGATE_FUNCTION (column3)
FROM
        table1
GROUP BY
        column1,
        column2;
```

It is not mandatory to include an aggregate function in the SELECT clause. However, if you use an aggregate function, it will calculate the summary value for each group.

If you want to filter the rows before grouping, you add a WHERE clause. However, to filter groups, you use the HAVING clause.

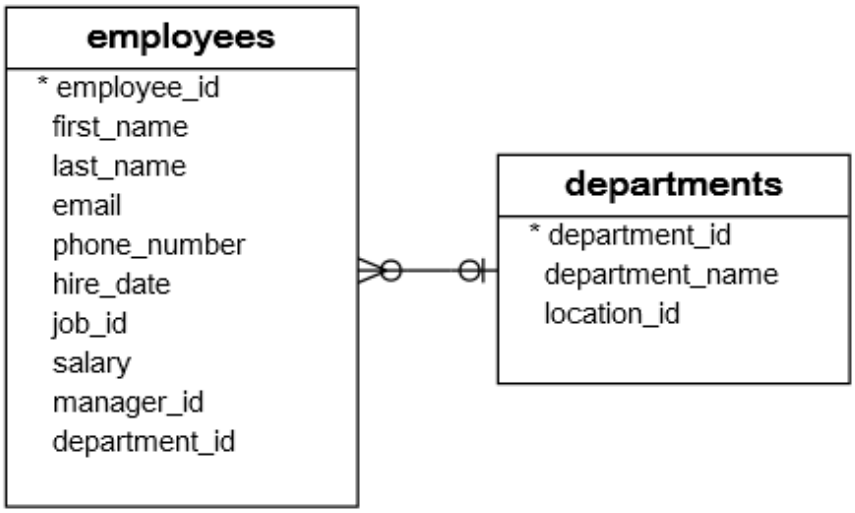
It is important to emphasize that the WHERE clause is applied before rows are grouped whereas the HAVING clause is applied after rows are grouped. In other words, the WHERE clause is applied to rows whereas the HAVING clause is applied to groups.

To sort the groups, you add the ORDER BY clause after the GROUP BY clause.

The columns that appear in the GROUP BY clause are called *grouping columns*. If a grouping column contains NULL values, all NULL values are summarized into a single group because the GROUP BY clause considers NULL values are equal.

SQL GROUP BY examples

We will use the `employees` and `departments` tables in the sample database to demonstrate how the GROUP BY clause works.



To find the headcount of each department, you group the employees by the `department_id` column, and apply the COUNT function to each group as the following query:

```
SELECT
        department_id,
        COUNT(employee_id) headcount
FROM
        employees
GROUP BY
        department_id;
```

See it in action >

department_id	headcount
1	1
2	2
3	6
4	1
5	7
6	5
7	1
8	6
9	3
10	6
11	2

SQL GROUP BY with INNER JOIN example

To get the department name, you join the `employees` table with the `departments` table as follows:

```
SELECT
        e.department_id,
        department_name,
        COUNT(employee_id) headcount
FROM
        employees e
INNER JOIN departments d ON d.department_id = e.department_id
GROUP BY
        e.department_id;
```

See it in action >

department_id	department_name	headcount
1	Administration	1
2	Marketing	2
3	Purchasing	6
4	Human Resources	1
5	Shipping	7
6	IT	5
7	Public Relations	1
8	Sales	6
9	Executive	3
10	Finance	6
11	Accounting	2

SQL GROUP BY with ORDER BY example

To sort the departments by headcount, you add an ORDER BY clause as the following statement:

```
SELECT
    e.department_id,
    department_name,
    COUNT(employee_id) headcount
FROM
    employees e
        INNER JOIN
    departments d ON d.department_id = e.department_id
GROUP BY e.department_id
ORDER BY headcount DESC;
```

See it in action >

department_id	department_name	headcount
5	Shipping	7
3	Purchasing	6
10	Finance	6
8	Sales	6
6	IT	5
9	Executive	3
11	Accounting	2
2	Marketing	2
7	Public Relations	1
1	Administration	1
4	Human Resources	1

Note that you can use either the `headcount` alias or the `COUNT(employee_id)` in the ORDER BY clause.

SQL GROUP BY with HAVING example

To find the department whose headcount is greater than 5, you use the HAVING clause as the following query:

```
SELECT
    e.department_id,
    department_name,
    COUNT(employee_id) headcount
FROM
```

```
    employees e
        INNER JOIN
    departments d ON d.department_id = e.department_id
GROUP BY e.department_id
HAVING headcount > 5
ORDER BY headcount DESC;
```

See it in action >

	department_id	department_name	headcount
▶	5	Shipping	7
	3	Purchasing	6
	10	Finance	6
	8	Sales	6

SQL GROUP BY with MIN, MAX, and AVG example

The following query returns the minimum, maximum, and average salary of employees in each department.

```
SELECT
    e.department_id,
    department_name,
    MIN(salary) min_salary,
    MAX(salary) max_salary,
    ROUND(AVG(salary), 2) average_salary
FROM
    employees e
        INNER JOIN
    departments d ON d.department_id = e.department_id
GROUP BY e.department_id;
```

See it in action >

department_id	department_name	min_salary	max_salary	average_salary
1	Administration	4400.00	4400.00	4400.00
2	Marketing	6000.00	13000.00	9500.00
3	Purchasing	2500.00	11000.00	4150.00
4	Human Resources	6500.00	6500.00	6500.00
5	Shipping	2700.00	8200.00	5885.71
6	IT	4200.00	9000.00	5760.00
7	Public Relations	10000.00	10000.00	10000.00
8	Sales	6200.00	14000.00	9616.67
9	Executive	17000.00	24000.00	19333.33
10	Finance	6900.00	12000.00	8600.00
11	Accounting	8300.00	12000.00	10150.00

SQL GROUP BY with SUM function example

To get the total salary per department, you apply the SUM function to the `salary` column and group employees by the `department_id` column as follows:

```
SELECT
    e.department_id,
    department_name,
    SUM(salary) total_salary
FROM
    employees e
        INNER JOIN
    departments d ON d.department_id = e.department_id
GROUP BY e.department_id;
```

See it in action >

department_id	department_name	total_salary
1	Administration	4400.00
2	Marketing	19000.00
3	Purchasing	24900.00
4	Human Resources	6500.00
5	Shipping	41200.00
6	IT	28800.00
7	Public Relations	10000.00
8	Sales	57700.00
9	Executive	58000.00
10	Finance	51600.00
11	Accounting	20300.00

SQL GROUP BY multiple columns

So far, you have seen that we have grouped all employees by one column. For example, the following clause

```
GROUP BY department_id
```

place all rows with the same values in the `department_id` column in one group.

How about grouping employees by values in both `department_id` and `job_id` columns?

```
GROUP BY department_id, job_id
```

This clause will group all employees with the same values in both `department_id` and `job_id` columns in one group.

The following statement groups rows with the same values in both `department_id` and `job_id` columns in the same group then returns the rows for each of these groups.

```
SELECT
    e.department_id,
    department_name,
    e.job_id,
    job_title,
    COUNT(employee_id)
FROM
    employees e
        INNER JOIN
    departments d ON d.department_id = e.department_id
        INNER JOIN
    jobs j ON j.job_id = e.job_id
GROUP BY e.department_id , e.job_id;
```

See it in action >

department_id	department_name	job_id	job_title	COUNT(employee_id)
1	Administration	3	Administration Assistant	1
2	Marketing	10	Marketing Manager	1
2	Marketing	11	Marketing Representative	1
3	Purchasing	13	Purchasing Clerk	5
3	Purchasing	14	Purchasing Manager	1
4	Human Resources	8	Human Resources Representative	1
5	Shipping	17	Shipping Clerk	2
5	Shipping	18	Stock Clerk	1
5	Shipping	19	Stock Manager	4
6	IT	9	Programmer	5
7	Public Relations	12	Public Relations Representative	1

The department 2, 3 and 5 appears more than one.

This is because these departments have employees who hold different jobs. For example, in the shipping department, there are 2 employees holding the shipping clerk job, 1 employee holding the stock clerk job, and 4 employees holding the stock manager job.

SQL GROUP BY and DISTINCT

If you use the `GROUP BY` clause without an aggregate function, the `GROUP BY` clause behaves like the DISTINCT operator.

The following gets the phone numbers of employees and also group rows by the phone numbers.

```
SELECT
        phone_number
FROM
        employees
GROUP BY
        phone_number;
```

See it in action >

Notice that the phone numbers are sorted.

The following statement also retrieves the phone numbers but instead of using the `GROUP BY` clause, it uses the `DISTINCT` operator.

```
SELECT DISTINCT
        phone_number
FROM
        employees;
```

See it in action >

The result set is the same except that the one returned by the `DISTINCT` operator is not sorted.

In this tutorial, we have shown you how to use the `GROUP BY` clause to summarize rows into groups and apply the aggregate function to each group.

Previous
SQL CROSS JOIN

Next
SQL HAVING

What Is SQL

SQL Sample Database

SQL Syntax

SQL TUTORIAL

SQL SELECT

SQL ORDER BY

SQL DISTINCT

SQL LIMIT

SQL FETCH

SQL WHERE

SQL Comparison Operators

SQL Logical Operators

SQL AND

SQL OR

SQL BETWEEN

SQL IN

SQL LIKE

SQL NOT

SQL IS NULL

SQL Alias

SQL INNER JOIN

SQL LEFT JOIN

SQL SELF JOIN

SQL FULL OUTER JOIN

SQL CROSS JOIN

SQL GROUP BY

SQL GROUPING SETS

SQL ROLLUP

SQL CUBE

SQL HAVING

SQL Subquery

SQL Correlated Subquery

SQL ALL

SQL ANY

SQL EXISTS

SQL UNION

SQL INTERSECT

SQL CASE

SQL MINUS

SQL INSERT

SQL UPDATE

SQL DELETE

SQL AGGREGATE FUNCTIONS

SQL AVG

SQL COUNT

SQL MAX

SQL MIN

SQL SUM

MANAGING DATABASE OBJECTS

SQL Data Types

SQL CREATE TABLE

SQL Identity

SQL Auto Increment

SQL ALTER TABLE

SQL ADD COLUMN

SQL DROP COLUMN

SQL DROP TABLE

SQL TRUNCATE TABLE

SQL CONSTRAINTS

SQL Primary Key

SQL Foreign Key

SQL UNIQUE Constraint

SQL CHECK Constraint

SQL NOT NULL Constraint

ABOUT SQL TUTORIAL

The SQLTutorial.org is created to help you master the SQL language fast by using simple but practical examples and easy-to-understand explanations.

Search this website

RECENT TUTORIALS

SQL Sample Database

SQL DROP COLUMN

SQL Identity

SQL Auto Increment

SQL ADD COLUMN

SITE LINKS

Home

Contact Us

Privacy Policy