

GROUP BY

GROUP BY

When working in SQL, results can be grouped according to a specific field or fields

- GROUP BY must be placed immediately after the WHERE conditions, if any, and just before the ORDER BY clause
- GROUP BY is one of the most powerful and useful tools in SQL

GROUP BY

GROUP BY



SQL

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

Lecturer notes: The syntax to comply with is the same: SELECT column names from a given table, where some condition or conditions have been satisfied. Group by column name or column names, and then finish with order by and the same or different column names. So, for the moment, we need to remember that the group by clause is located just above the order by clause.

An example:

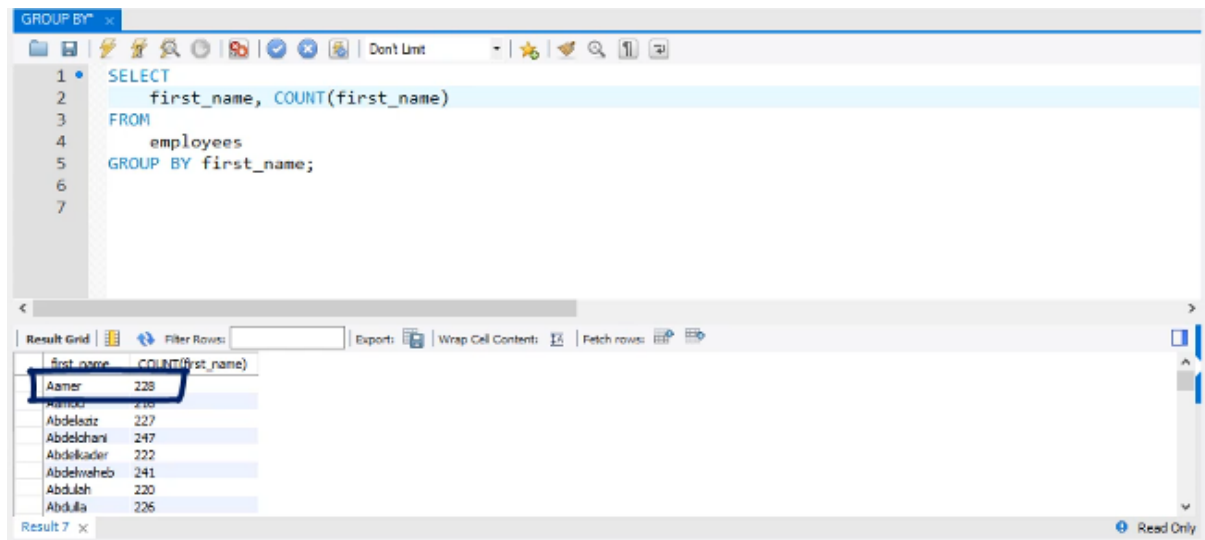
```
1 SELECT
2   COUNT(first_name)
3 FROM
4   employees
5 GROUP BY first_name;
```

COUNT(first_name)
228
216
227
247
222
241
220
226

Result 6 x Read Only

Lecturer notes: In most cases, when we need an aggregate function, we must add a group by clause in our query too. E.g. assuming we need a list composed of two fields; the first must contain a distinct first name of the employee, and the second field, the number of times we encounter this

name in our database. Looking for a single total value must ring a bell straight away. If we type select count first name from employees, we will get the total number of records in this table. Then, if we add group by first name, we'll split the result returned from the select statement into groups. In this column, we see the number of times each name is encountered, but we don't see the names these values refer to. A rule of thumb that professionals strictly comply with always include the field you have group to results by in the select statement. Let's do that:



The screenshot shows a SQL IDE window titled "GROUP BY". The query editor contains the following SQL code:

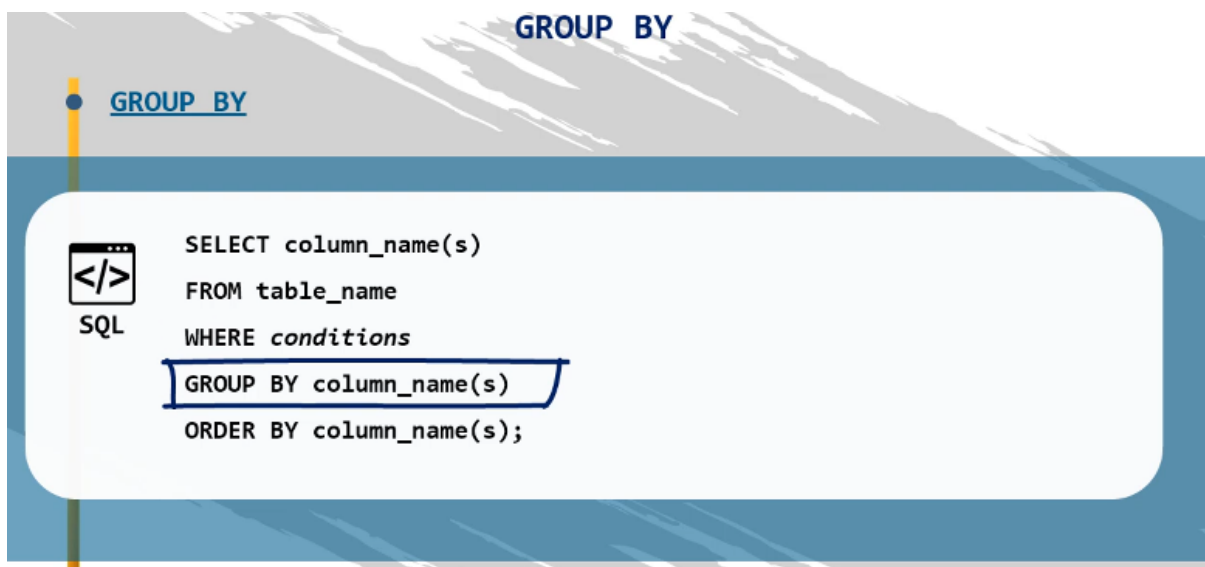
```
1 SELECT
2   first_name, COUNT(first_name)
3 FROM
4   employees
5 GROUP BY first_name;
```

The results pane shows a table with two columns: "first_name" and "COUNT(first_name)". The data is as follows:

first_name	COUNT(first_name)
Aamer	228
Ahmad	216
Abdelaziz	227
Abdelchani	247
Abdelkader	222
Abdelmoheb	241
Abdulah	220
Abdulla	226

Lecturer notes: Aamer can be seen 228 times Ahmad 216 and so on. This rule is crucial because although in workbench the query would run properly if we don't include the group by field and the select statement, but this will not be valid in some other databases. There it will be impossible to execute the query if written without the group by column in the select statement. So we should stick to this simple rule. It also improves the organisation and readability of our output.

This last piece of information was an important addition to the content of this lecture. Not all blocks of code are mandatory, but we must get used to the order in which we state these blocks in the query. Remember the following logical flow. Select something from a certain table where certain conditions are met. Group the results by a column and possibly order them in a certain direction:



The diagram shows a SQL query structure with a focus on the **GROUP BY** clause. The query is:

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

The **GROUP BY** clause is highlighted with a blue box. The title "GROUP BY" is displayed in large blue letters at the top of the diagram.

Using Aliases (AS) *

```

1 SELECT
2   first_name, COUNT(first_name) AS names_count
3 FROM
4   employees
5 GROUP BY first_name
6 ORDER BY first_name;
7

```

Result Grid

first_name	names_count
Aamer	228
Aamod	216
Abdelaziz	227
Abdelchani	247
Abdelkader	222
Abdelmehebo	241
Abdullah	220
Abdulla	226

HAVING

HAVING

refines the output from records that do not satisfy a certain condition

- frequently implemented with GROUP BY



SQL

```

SELECT column_name(s)
FROM table_name
WHERE conditions
GROUP BY column_name(s)
HAVING conditions
ORDER BY column_name(s);

```

Lecturer notes: 'Having' is a clause frequently implemented with group by because it refines the output from records that do not satisfy a certain condition. Why does the having clause exist? Internalising the corresponding syntax will help us explain the difference between the two keywords. Having needs to be inserted between the group by and order by clauses. The difference between WHERE and HAVING is; after HAVING, you can have a condition with an aggregate function, while WHERE cannot use aggregate functions within its conditions. An aggregate function is a type of function that performs a calculation on a set of values and returns a single value e.g. COUNT(), MIN(), MAX(), SUM() and AVG().

aggregate functions

they gather data from *many* rows of a table, then aggregate it into a *single* value

SQL Query Editor:

```

23 • SELECT
24     first_name, COUNT(first_name) as names_count
25 FROM
26     employees
27 WHERE
28     COUNT(first_name) > 250
29 GROUP BY first_name
30 ORDER BY first_name;
31
32

```

Result Grid:

emp_no	birth_date	first_name	last_name	gender	hire_date
47291	1960-09-09	Ulf	Flexer	M	2000-01-12
60134	1964-04-21	Seshu	Rathanvi	F	2000-01-02
72329	1953-02-09	Randi	Lut	F	2000-01-02
108201	1955-04-14	Mariacola	Boreale	M	2000-01-01
205048	1960-09-12	Ennio	Alblas	F	2000-01-06
222965	1959-08-07	Volkmar	Perko	F	2000-01-13
226633	1958-06-10	Xuelun	Benzmuller	F	2000-01-04
227544	1954-11-17	Shahab	Demeyer	M	2000-01-08

employees 2 x

Output:

#	Time	Action	Message
1	18:05:25	SELECT * FROM employees WHERE hire_date >= '2000-01-01'	13 row(s) returned
2	18:05:30	SELECT * FROM employees HAVING hire_date >= '2000-01-01'	13 row(s) returned
3	18:06:19	SELECT first_name, COUNT(first_name) as names_count FROM employees...	Error Code: 1111. Invalid use of group function

Lecturer notes: ^ Assume we want to extract a list with all first names that appear more than 250 times in the employee's table. If we try to set this condition in the where clause, workbench wouldn't indicate there's a mistake in our code because this is the correct syntax, but we'll be shown an error message when we try to execute the query, and it will be a very eloquent one. Invalid use of group function.

HOWEVER, IF WE CHANGE THE KEYWORD TO 'HAVING' and add the line of code in the right place. Just after the group by statement. Now rerun the query:

SQL Query Editor:

```

22
23 • SELECT
24     first_name, COUNT(first_name) AS names_count
25 FROM
26     employees
27 GROUP BY first_name
28 HAVING COUNT(first_name) > 250
29 ORDER BY first_name;
30
31

```

Result Grid:

first_name	names_count
Adam	251
Akemi	259
Anvuan	278
Arie	255
Arno	251
Arvind	258
Atreve	258
Atrevi	251

Result 3 x

Output:

#	Time	Action	Message
2	18:05:30	SELECT * FROM employees HAVING hire_date >= '2000-01-01'	13 row(s) returned
3	18:06:19	SELECT first_name, COUNT(first_name) as names_count FROM employee...	Error Code: 1111. Invalid use of group function
4	18:06:39	SELECT first_name, COUNT(first_name) AS names_count FROM employee...	193 row(s) returned

Lecturer notes: Anytime an aggregate function is required for the solution of our task, we use HAVING. In the problem we just solved, "extract all first names that appear more than 250 times in the employee's table" We must first spot the phrase 250 times. It leads to counting. COUNT() is an aggregate function.

QUESTION: Select all employees whose average salary is higher than \$120,000 per annum.

Hint: We should obtain 101 records.

```
101 • SELECT emp_no, salary FROM salaries
102     GROUP BY emp_no
103     HAVING AVG(salary) > 120000
104     ORDER BY salary;
```

^This didn't work at all. But this did:

```
93 • SELECT emp_no, AVG(salary) AS avg_salary
94     FROM salaries
95     GROUP BY emp_no
96     HAVING AVG(salary) > 120000
97     ORDER BY avg_salary ASC;
98
99
```

Result Grid	Filter Rows:	Export:	Wrap Cell Content:	Result Grid
emp_no	avg_salary			
17238	120084.0000			
29224	120089.6667			
64633	120112.8889			
51022	120150.9000			
75138	120250.0000			

My original query didn't work because I grouped by the salary column instead of the employee number, which meant SQL was calculating averages per salary value rather than per employee. Even when I previously tried grouping by emp_no, it still failed because my SELECT list contained salary, a non-aggregated column that wasn't included in the GROUP BY, making the query invalid SQL. To fix this, we needed to select the aggregated value - AVG(salary) - and group only by emp_no, because that's the level at which we want to calculate the average. The reason it feels like we're "averaging twice" is simply that SQL requires the aggregate to appear in both the SELECT clause (to show it) and in the HAVING clause (to filter on it); it's not actually performing the calculation twice. Once these issues were corrected, the query returned the employees whose average salary exceeds \$120,000 as intended. ESSENTIALLY IN ORDER FOR HAVING() TO WORK THE COLUMN NEEDS TO BE AGGREGATED PRIOR TO PUTTING IT IN THE HAVING() FUNCTION.

When to use WHERE and HAVING:

The screenshot shows the MySQL Workbench interface with a SQL query editor. The query is as follows:

```
1 select first_name, count(first_name) as names_count
2 from employees
3 where hire_date > '1999-01-01'
4 group by first_name
5 having count(first_name) < 200
```

Annotations and arrows:

- A blue box labeled "HAVING" points to the `having count(first_name) < 200` clause.
- A blue box labeled "COUNT ()" points to the `count(first_name)` function in the `having` clause.
- A large blue box contains the text: "Extract a list of all names that are encountered less than 200 times. Let the data refer to people hired after the 1st of January 1999."
- A blue box labeled "WHERE" points to the `where hire_date > '1999-01-01'` clause.
- A blue box contains the text: "refers to all individual rows in the 'employees' table", with an arrow pointing to the `employees` table in the `from` clause.

The interface also shows a "SCHEMAS" panel on the left with a tree view containing `employees`, `information_schema`, `mysql`, `performance_schema`, and `sys`. The bottom right corner features a "365 Careers" logo.

WHERE vs HAVING

Aggregate functions - GROUP BY and HAVING

General conditions - WHERE