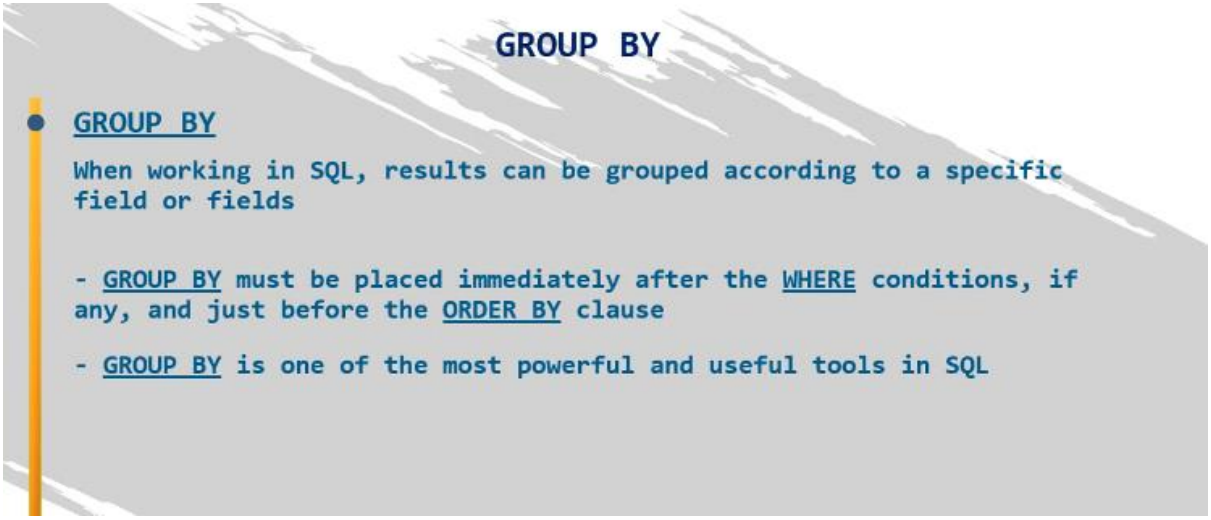
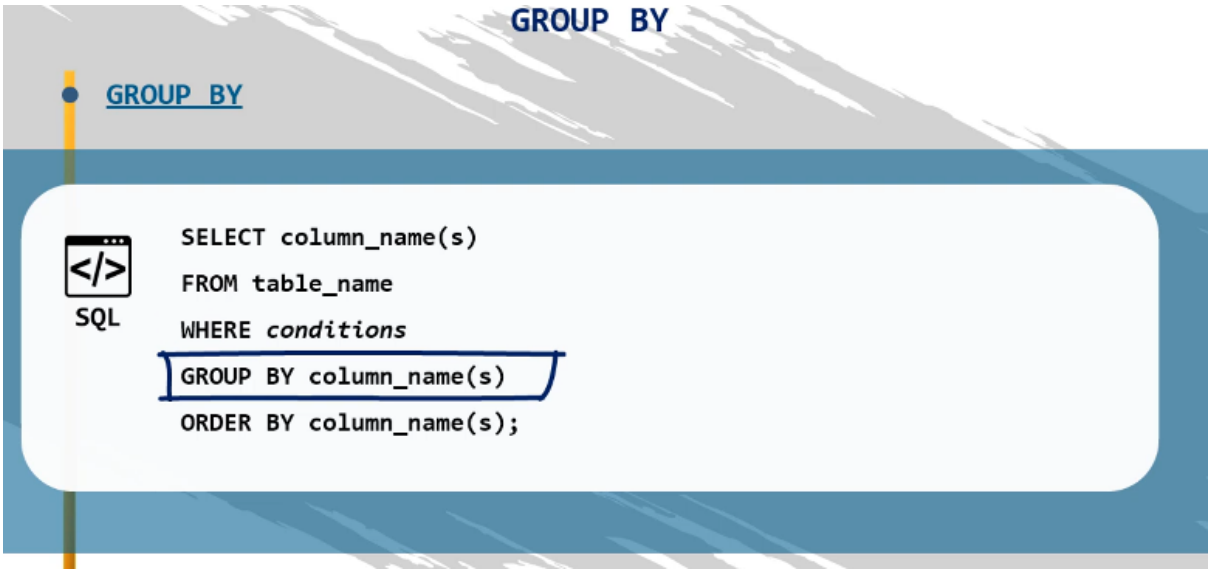
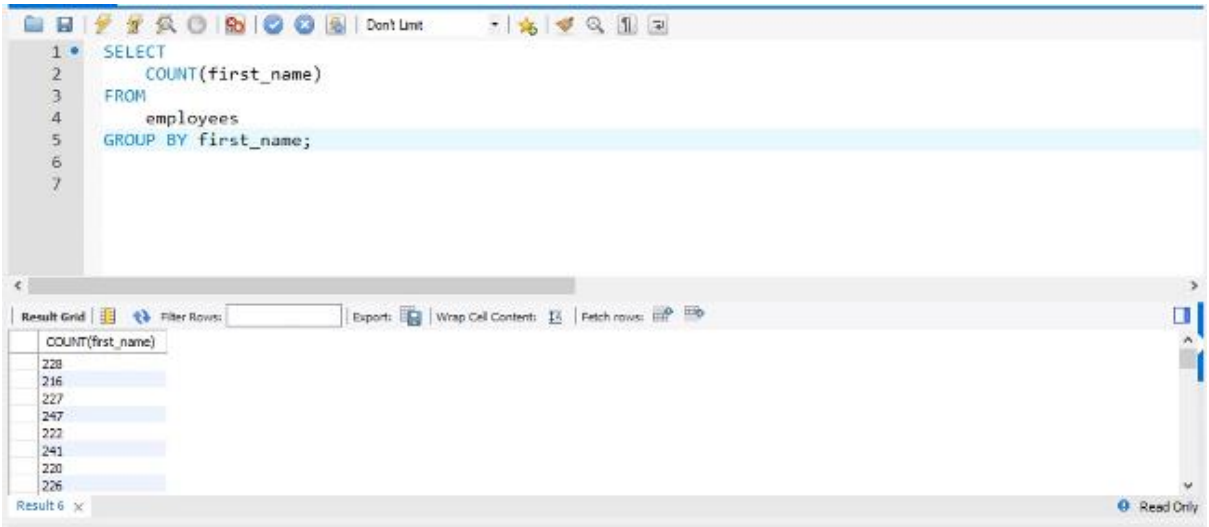




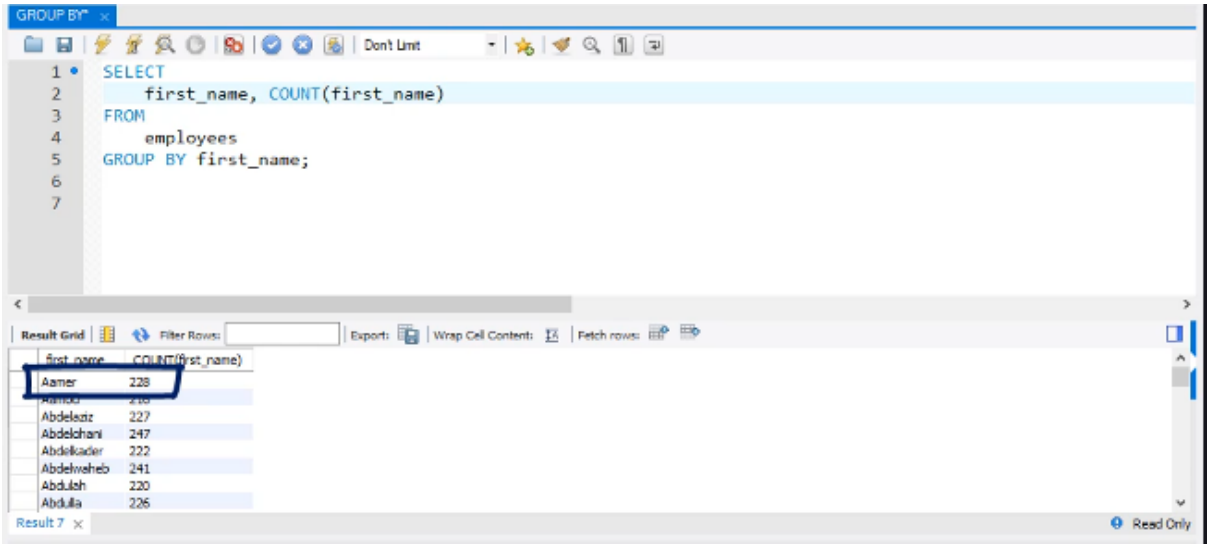
Lecturer notes: The syntax to comply with is the same: SELECT column names from a given table, where some condition or conditions have been satisfied. Group by column name or column names, and then finish with order by and the same or different column names. So, for the moment, we need to remember that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

An example:



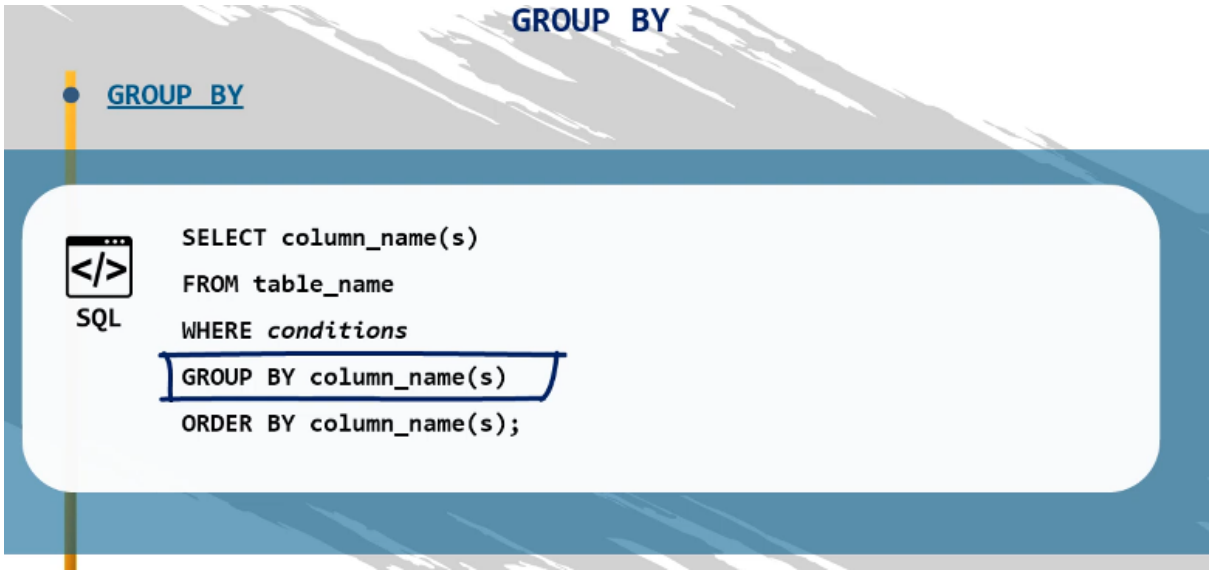
Lecturer notes: In most cases, when we need an aggregate function, we must add a group by clause in our query too. E.g. assuming we need a list composed of two fields; the first must contain a distinct first name of the employee, and the second field, the number of times we encounter this

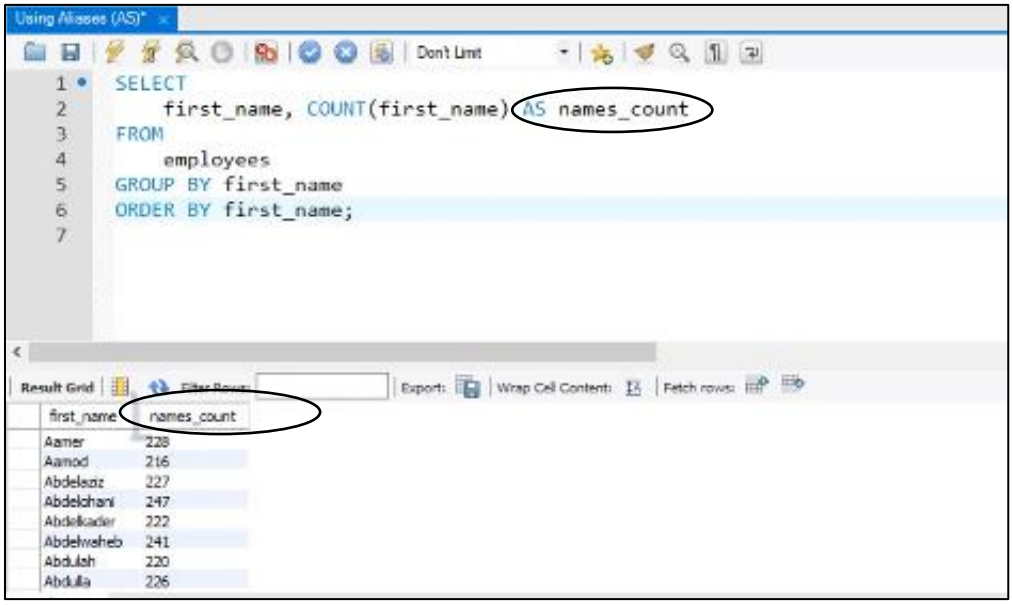
name in our database. Looking for a single total value must ring a bell straight away. If we type select count first name from employees, we will get the total number of records in this table. Then, if we add group by first name, we'll split the result returned from the select statement into groups. 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:

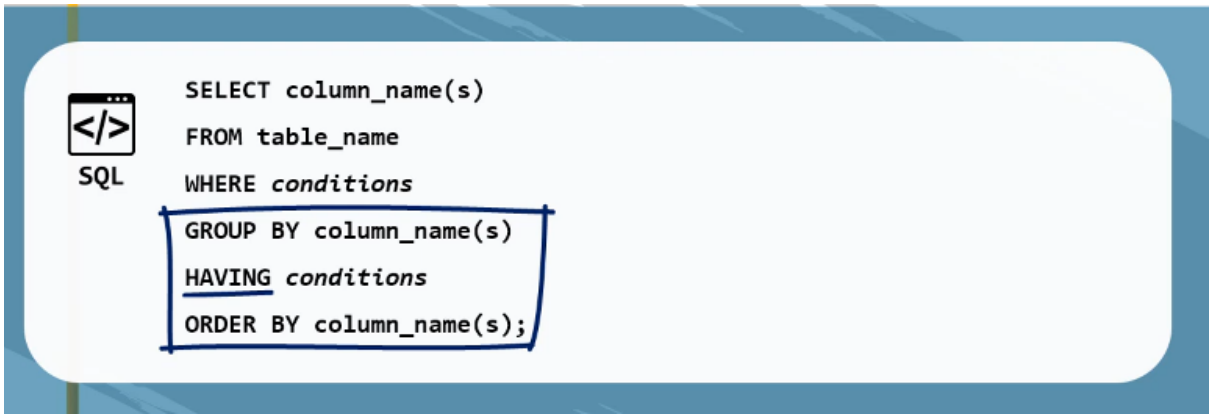


Lecturer notes: Almer can be seen 228 times Ahmad 216 and so on. This rule is crucial because although in workbench the query would run properly if we don't include the group by field and the select statement, but this will not be valid in some other databases. There it will be impossible to execute 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:



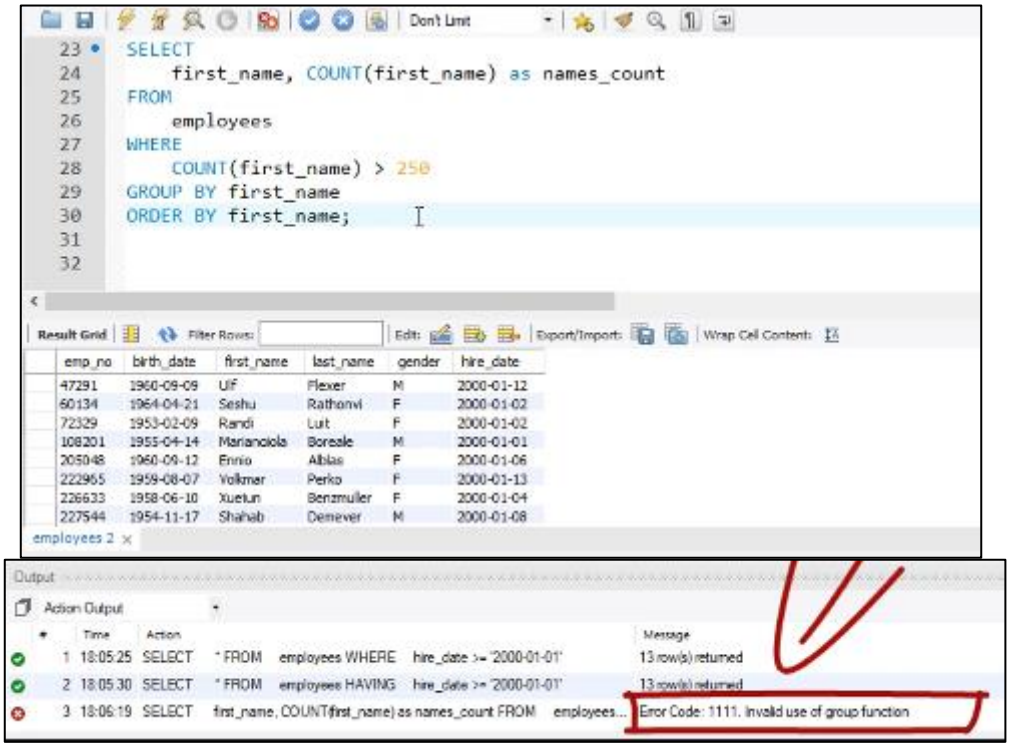




Lecturer notes: 'Having' is a clause frequently implemented with group by because it refines the output from records that do not satisfy a certain condition. Why does the having clause exist? Internalising the corresponding syntax will help us explain the difference between the two keywords. Having needs 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().

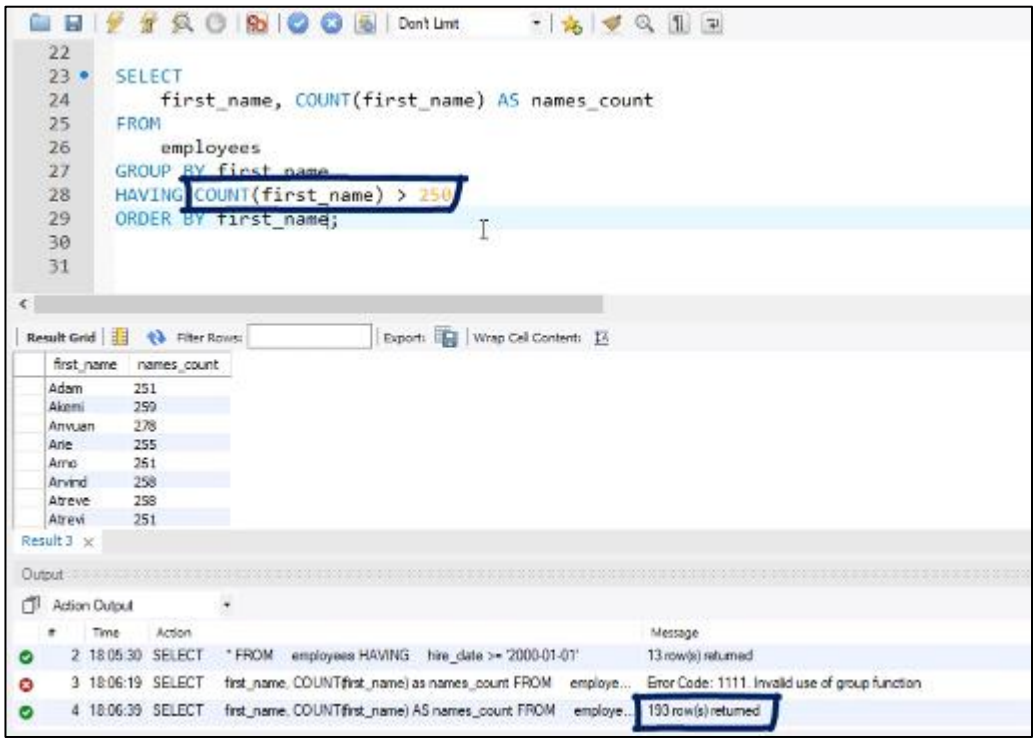
aggregate functions

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



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:



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
```

esult 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: