SQL Programming

Groping Our Way through Grouping

Page A-1: Intro

In the module on SET FUNCTIONS we learned that set functions operate on a group of rows and return a single row for that group.

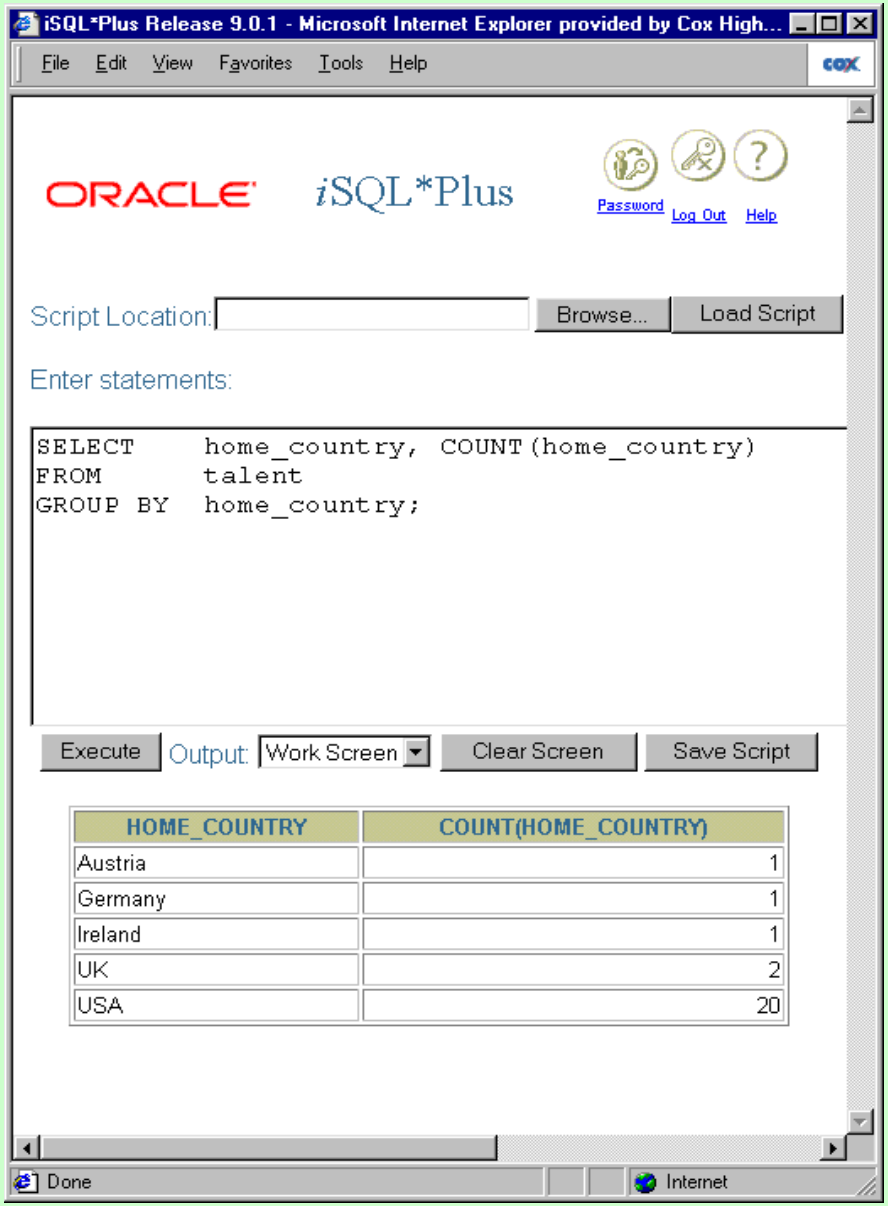
But, if you ask me, the grouping structure was somewhat weak. There were only two groups, or grouping mechanisms that we could use:

1. The whole table, or
2. Those rows of the table that were included as part of a WHERE predicate expression.

And, as you recall, with either of these options, the result table included a single row.

In this module we'll learn how to deal with multiple groups at the same time.

Page B-1 GROUP BY - Example



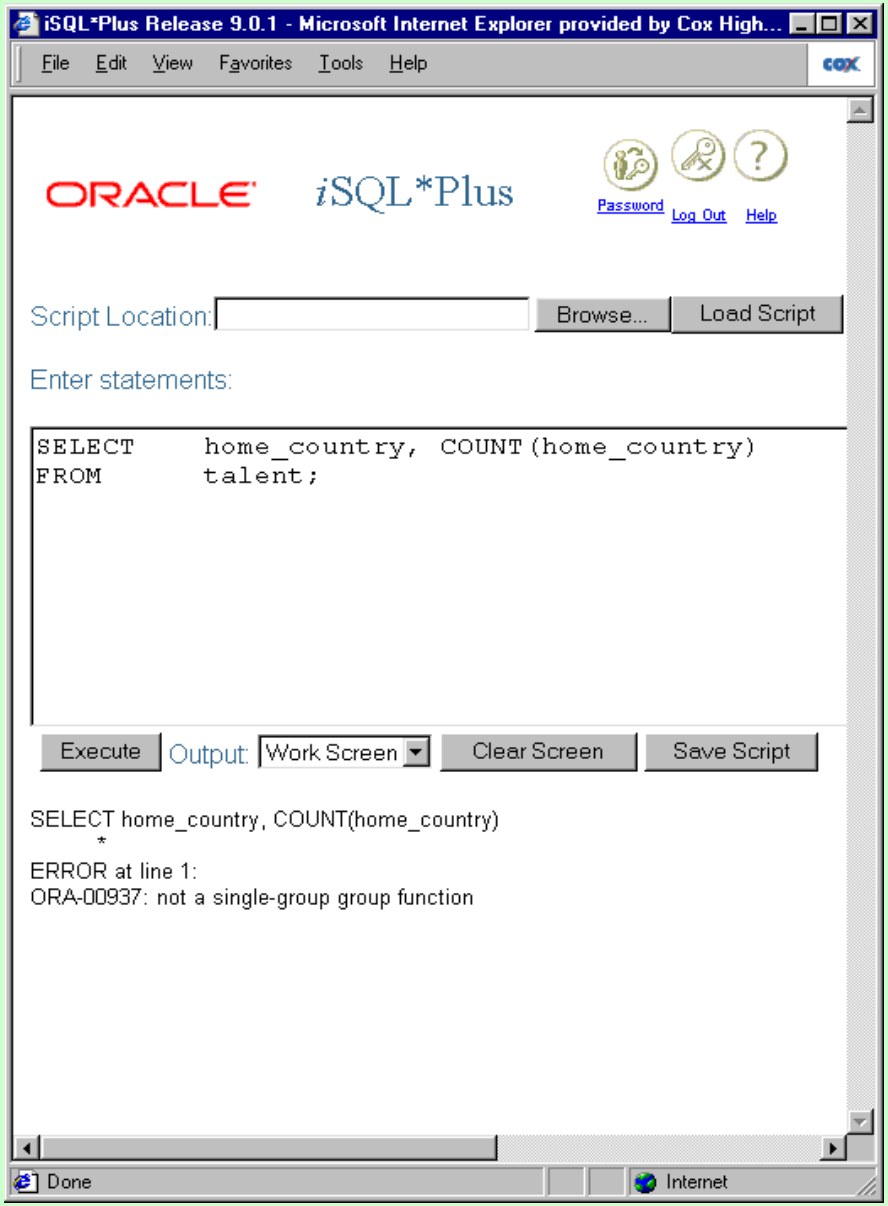
Take a moment to examine this SQL program.

The first thing you probably noticed in this program was the new clause: GROUP BY.

The GROUP BY clause specifies the column(s) that will be used to group the data.

In this example, the grouping column is the home_country column, and SQL will group the data based on each of the distinct values that occurs in the home_country column.

In the talent table, the home_country column contains 5 distinct values, and as we can see from the result table, summary information (COUNT) has been provided for each of these groups.



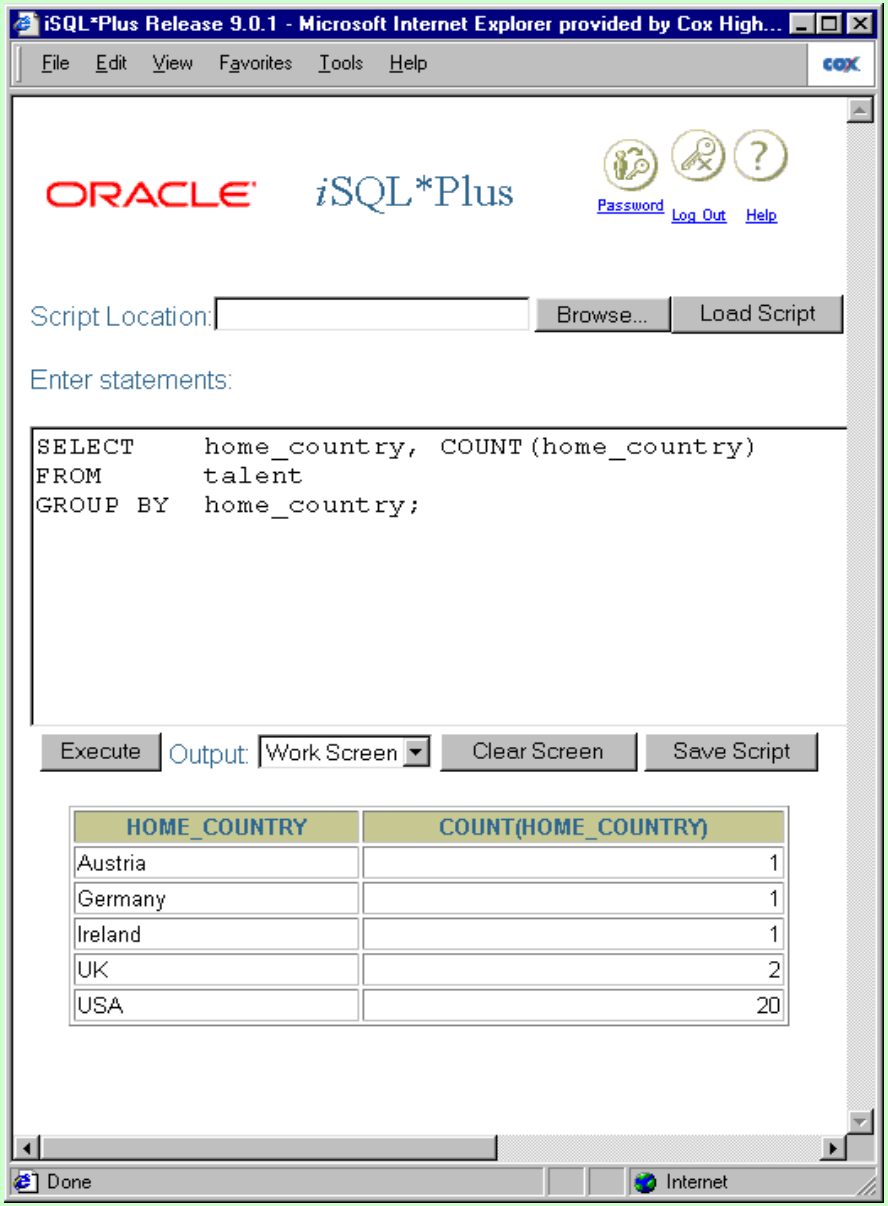
Grouping isn't an overly complicated concept, but the syntax is rather precise, so as you work on your SQL programs, expect that you'll be generating a LOT more syntax errors with these problem sets.

The SQL syntax requires that each column expression that is listed in the SELECT clause must also be named in the GROUP BY clause, or that column must be a group function.

The error message on this slide cryptically warns you that you're mixing detail with summary information. For obviously home_country is not a single-group group function ☺

If we eliminate the double-speak in the Oracle error message, the warning becomes intelligible (almost)

not a group function!

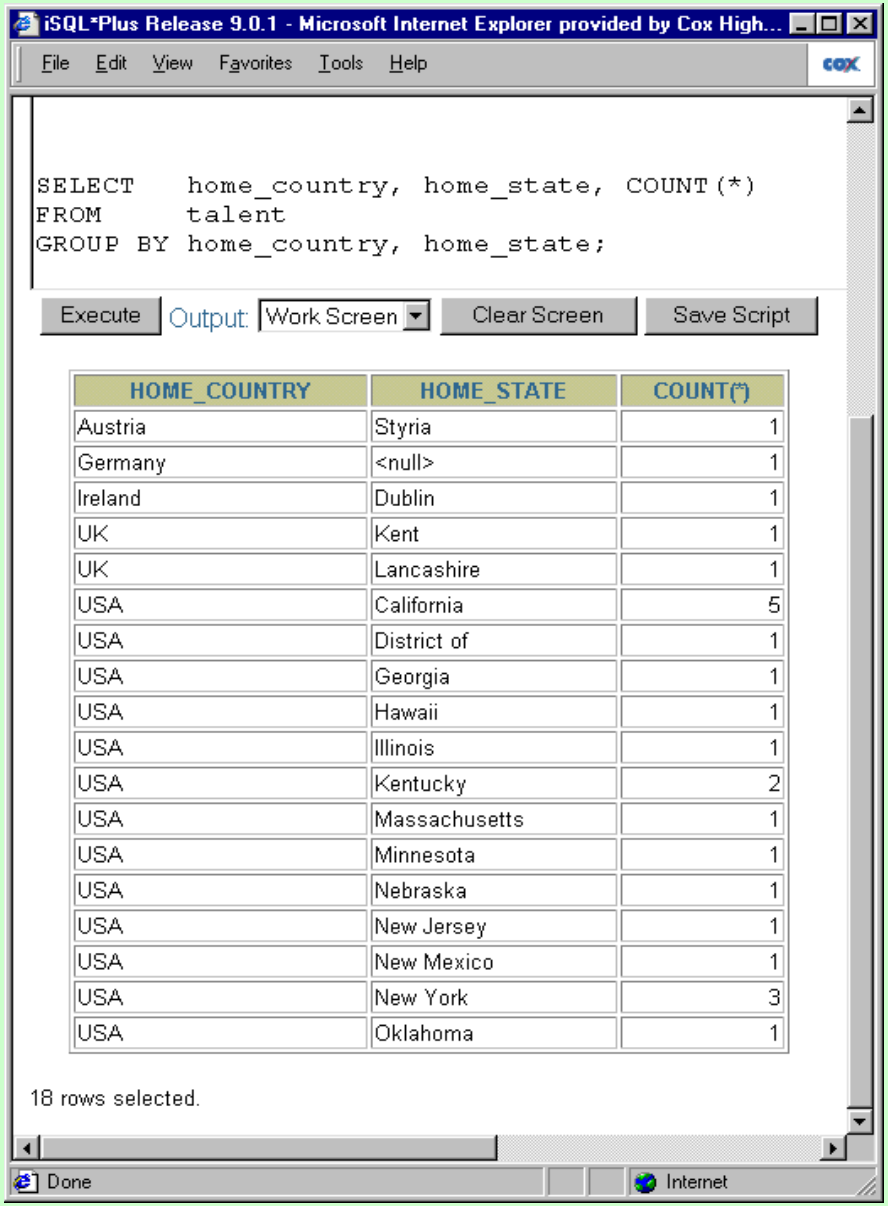


Conceptually you can think of SQL as creating a number of intermediate tables, one for each data value in the group.

These tables will be processed individually in order to calculate the statistics you've requested, and then the single result row from each of these tables showing that statistic, will be gathered together and presented in a single result table.

In this example, consider the FROM clause as building the 1st working table with all the rows from the base table (talent), If there were a WHERE clause in this program, it would trigger immediately after the FROM clause and create yet another intermediate working table with just the 'keeper' rows.

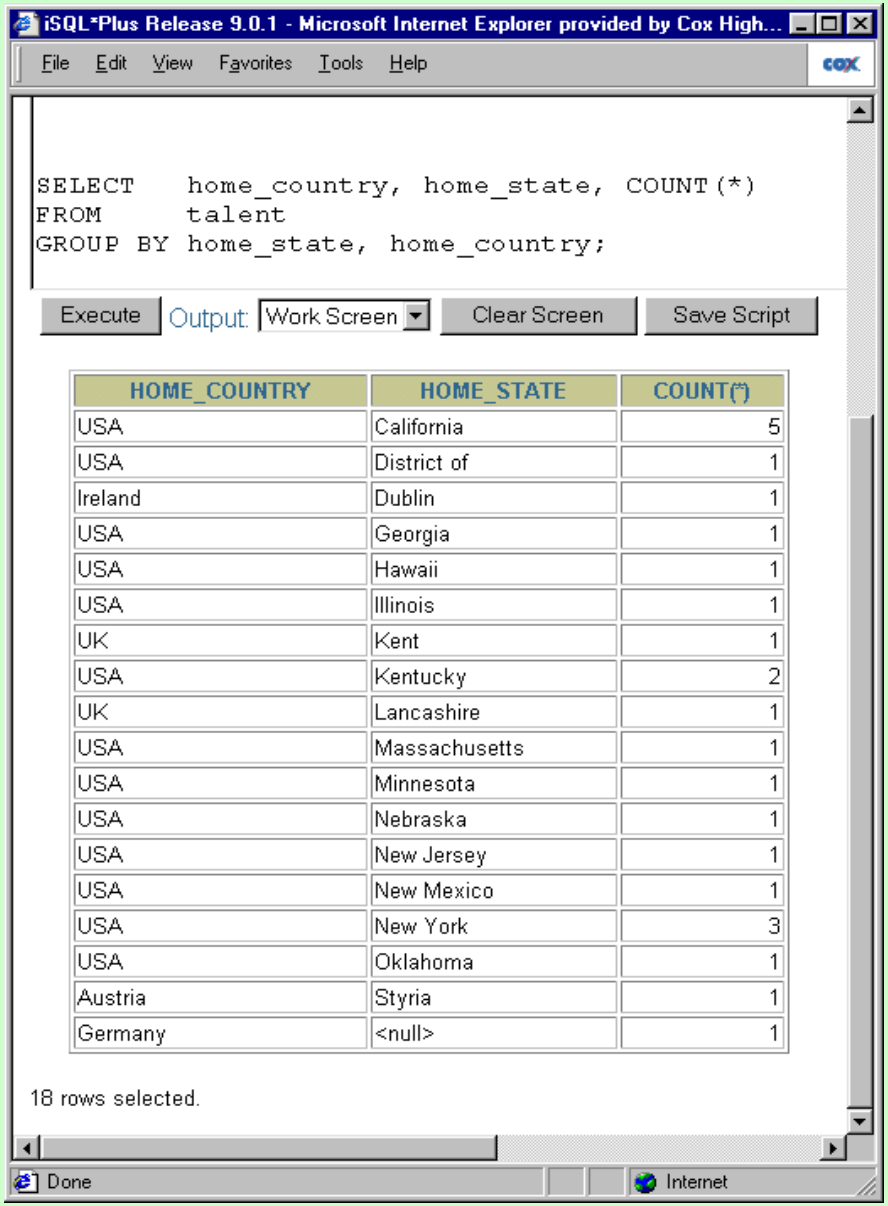
From this intermediate working table, think of SQL as building 5 more working tables, one for each of the values in the home_country column: Austria, Germany, Ireland, UK, and USA. Each of these tables will be processed, the single row statistic will be generated, and then these single row results will be combined in the final result table that is presented on screen.



GROUP BY generates groups in pretty much the same fashion as DISTINCT. And as we saw with DISTINCT, many columns can be selected.

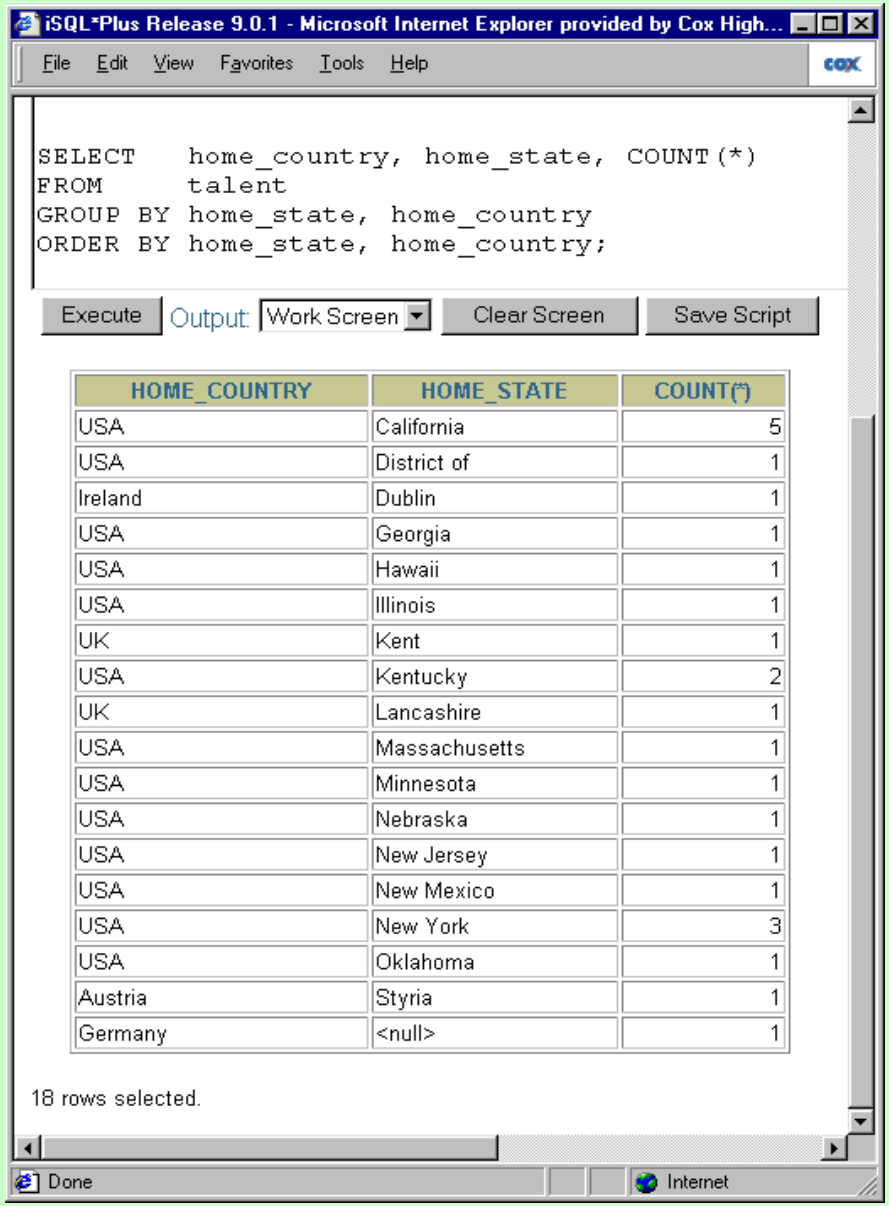
In this example we want a breakdown of talent, by home_state, within home_country.

Page B-5 GROUP BY Sequence



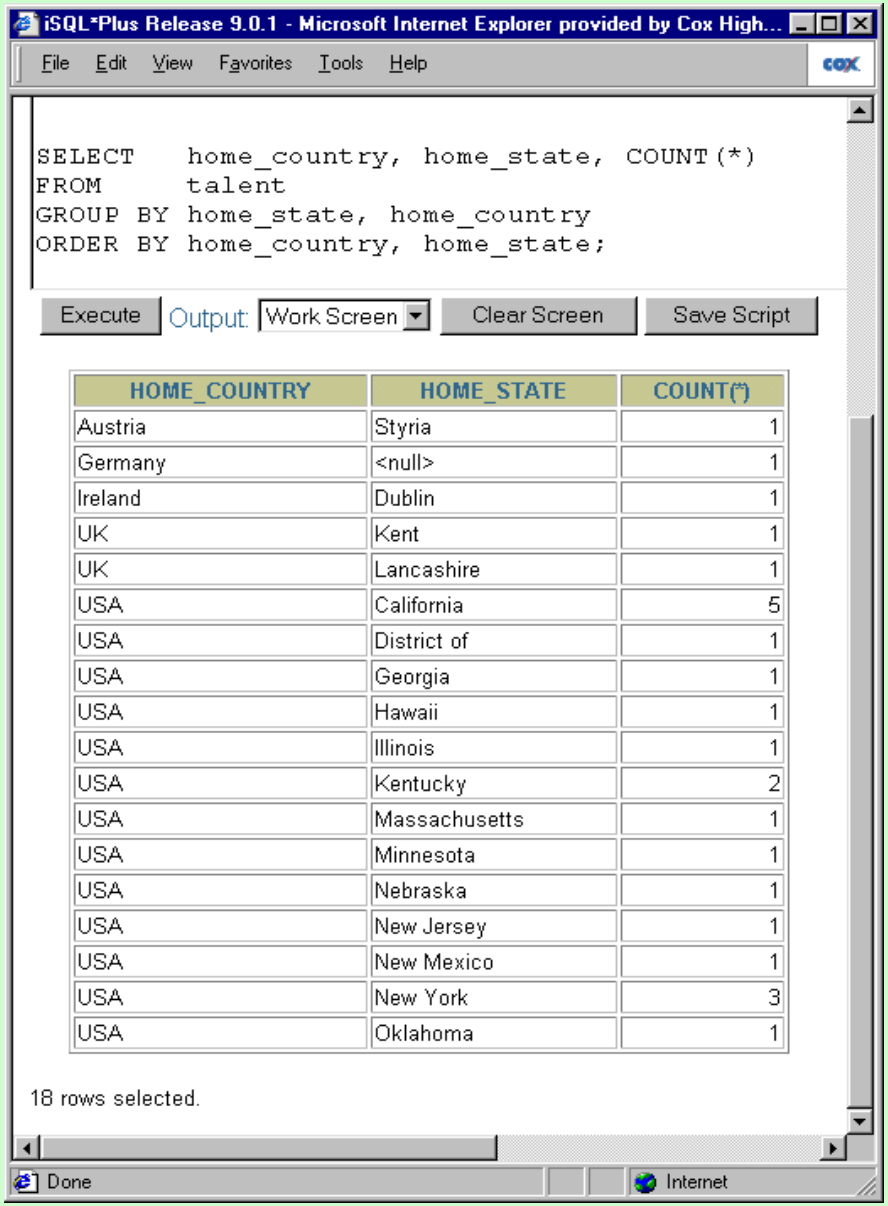
Most vendor implementations of SQL will display grouped results in sorted (ascending) order based on the list of column names in the GROUP BY clause.

But do note, that this is not a feature of the standard, and once again, if this is something that is critical to your application, you should include an ORDER BY clause.



This example demonstrates the use of the ORDER BY clause in conjunction with GROUPing.

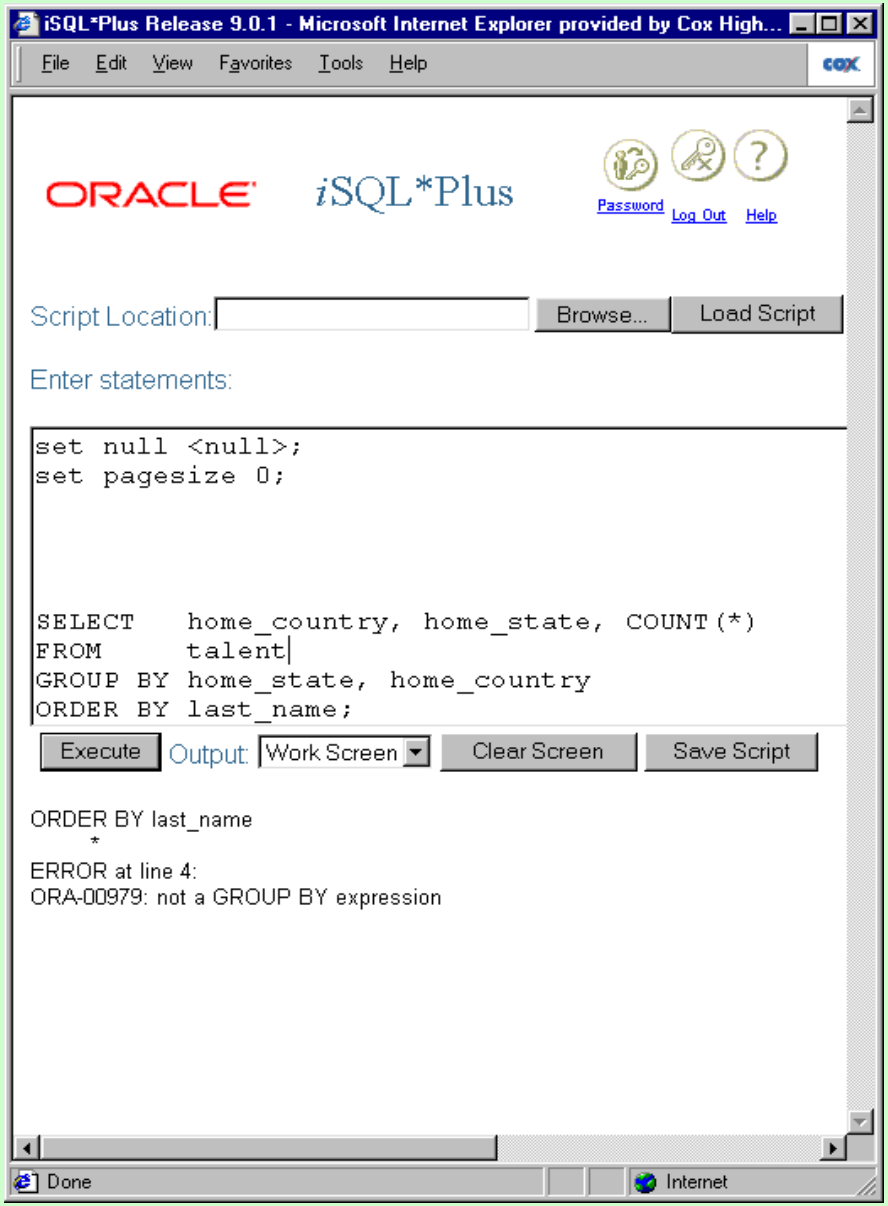
Page B-7 ORDER BY (cont)



As you might expect, the sort order does not have to jibe with the GROUP BY specification.

The previous slide demonstrated this SQL code:

```
SELECT   home_country, home_state, COUNT(*)
FROM     talent
GROUP BY home_state, home_country
ORDER BY home_state, home_country;
```

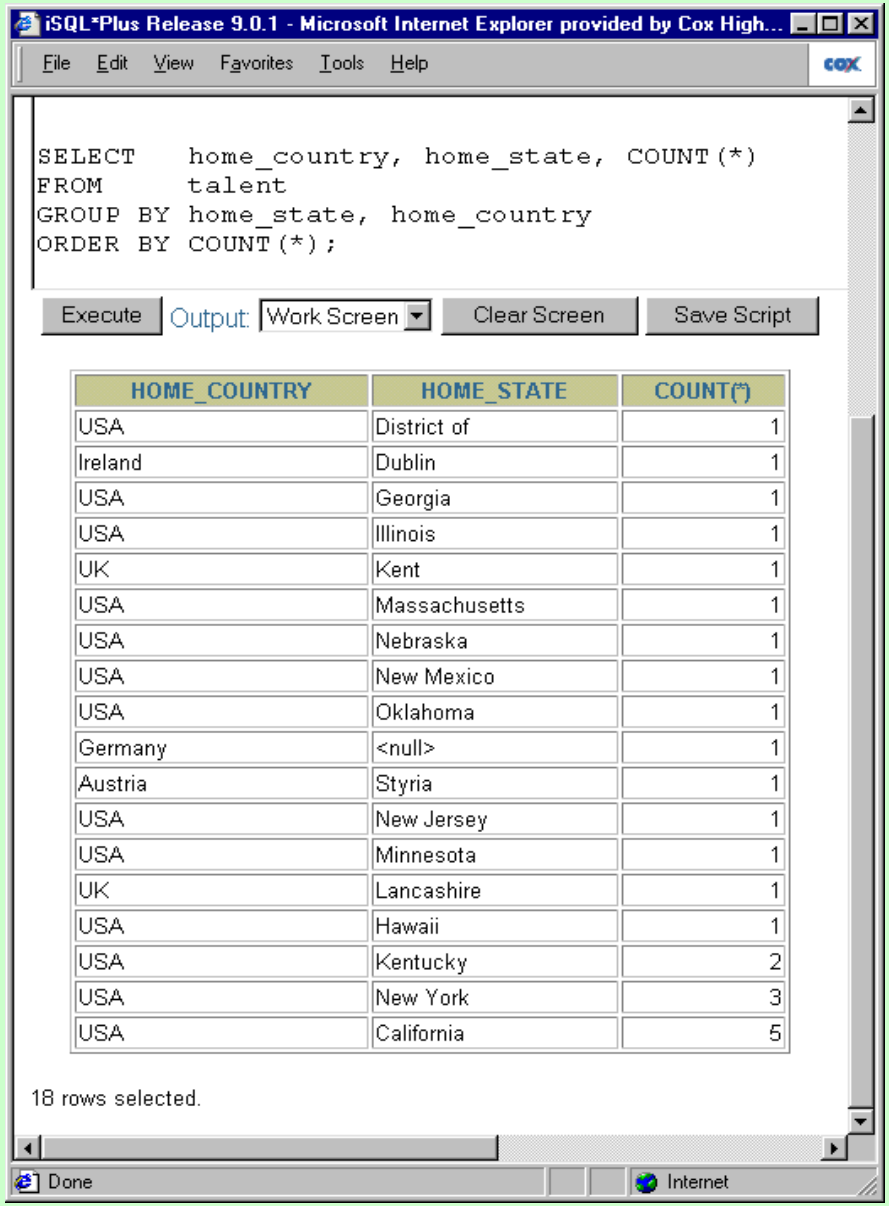


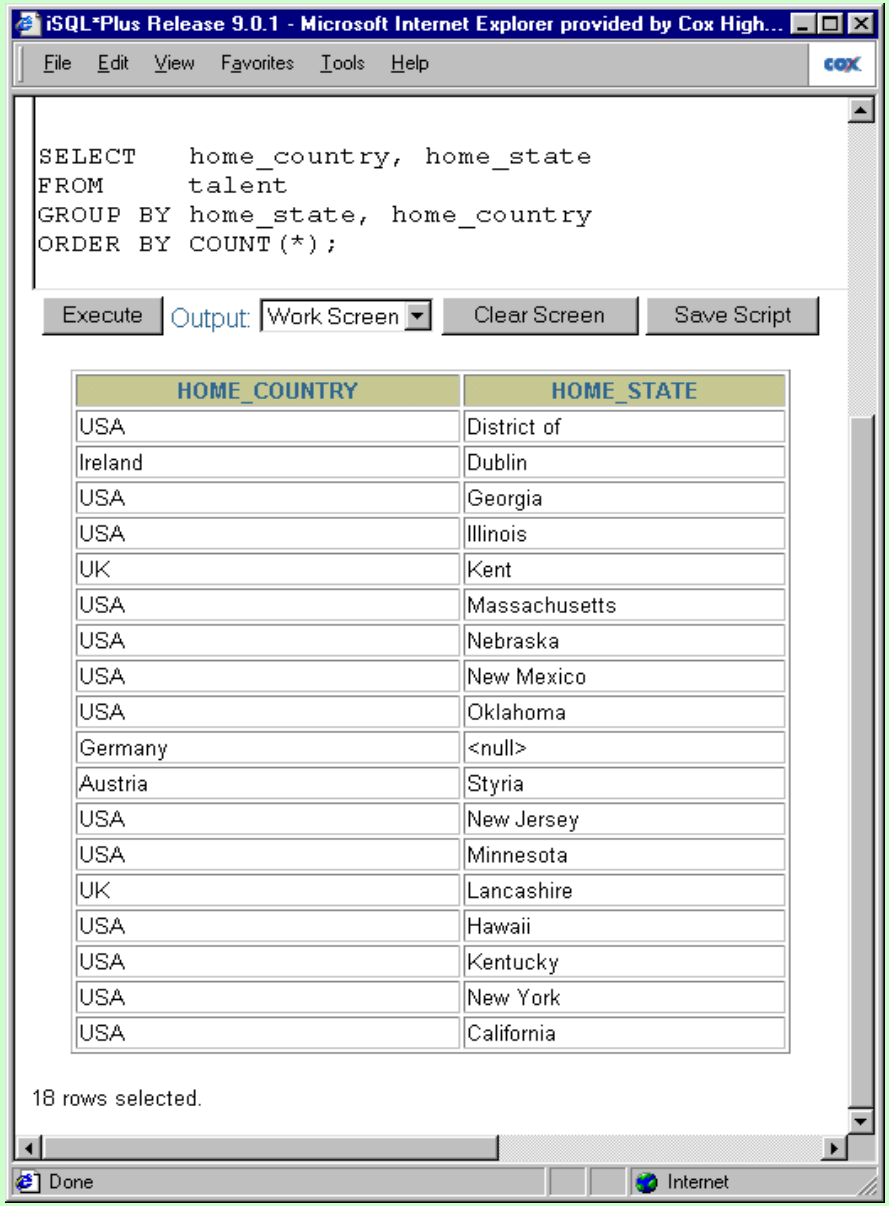
But notice what's happening in this example.

We observed earlier that all of the columns of the base table were available tot the ORDER BY clause, but that doesn't seem to be the case any more.

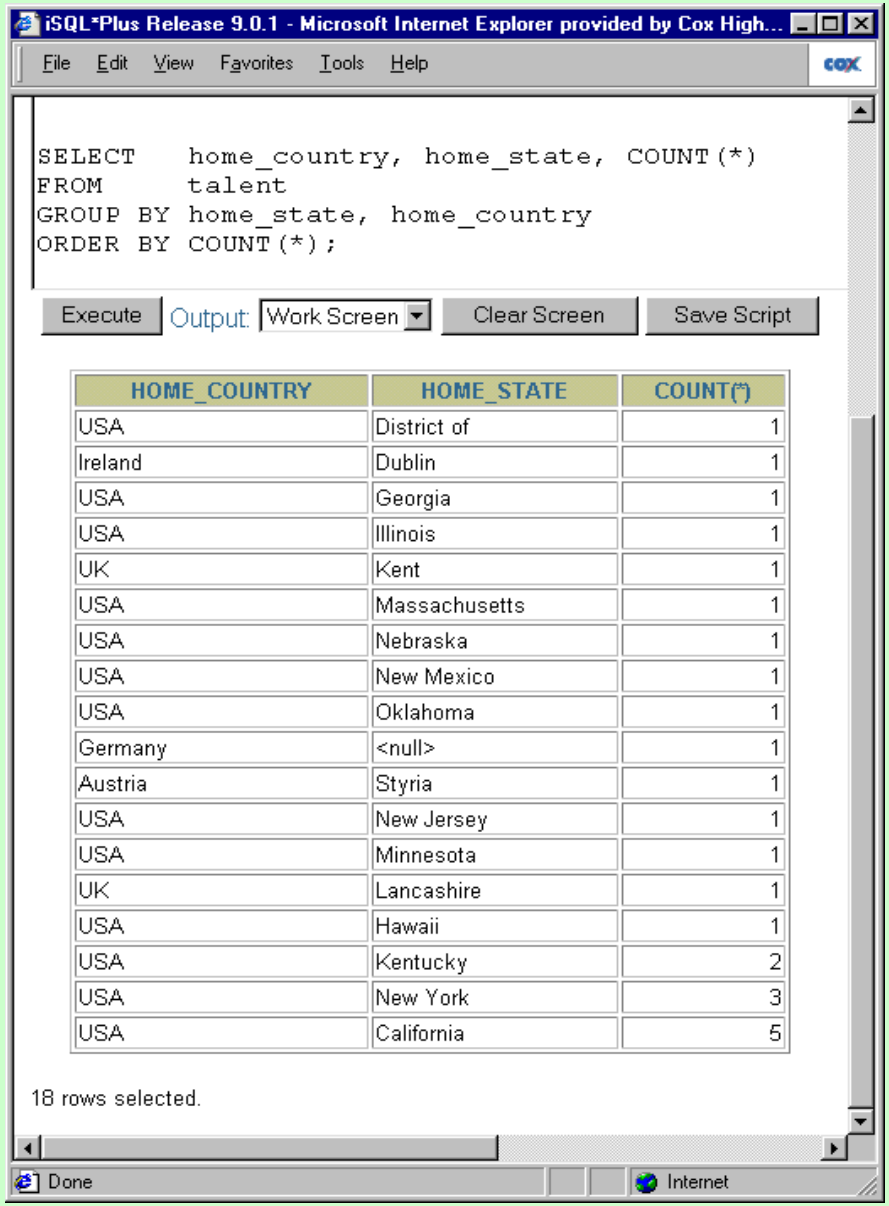
And indeed, it is not. Once SQL makes the break to working tables based on group values, the detail columns disappear. Only group level items cascade from this point forward.

But as the next pair of slides demonstrate, all of the group level items remain available to the ORDER BY clause.





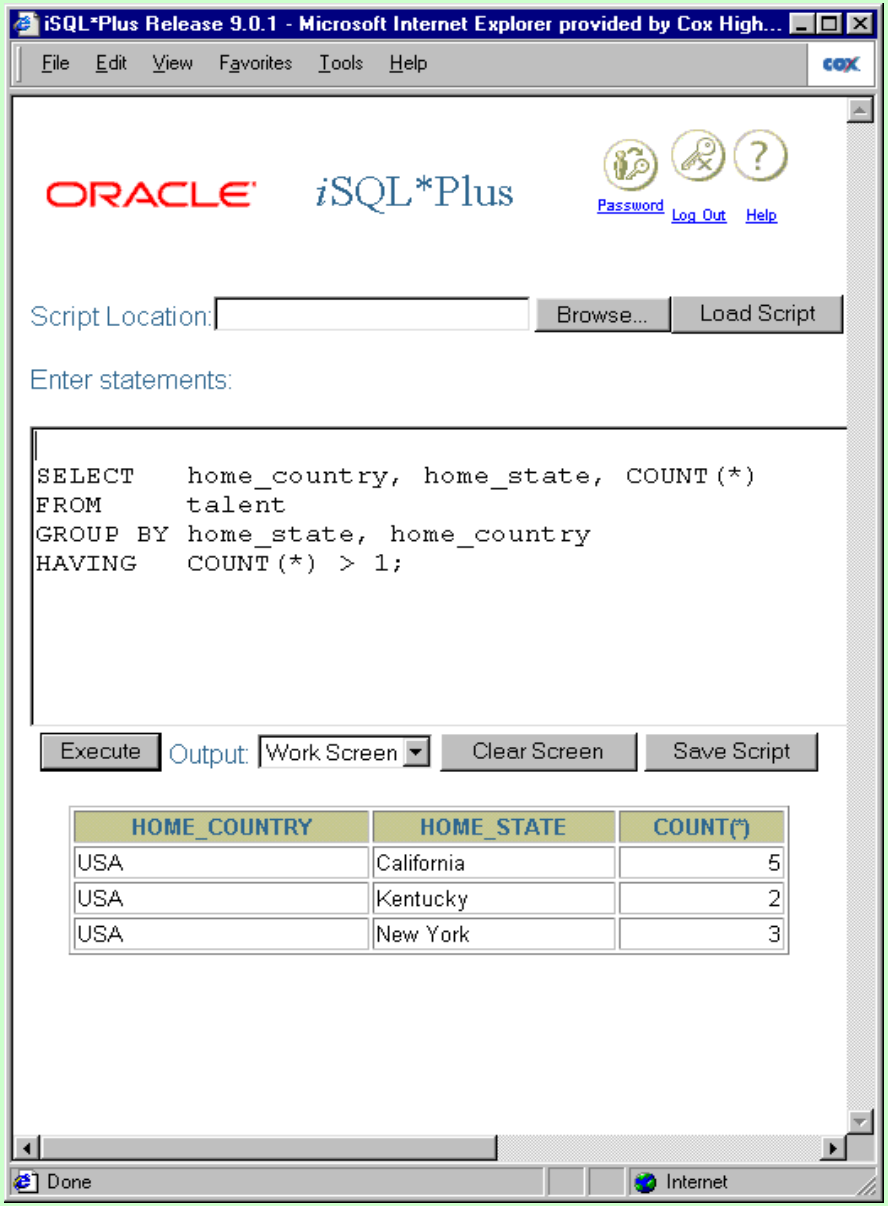
Page C-1 Restricting rows?



We've seen that the WHERE clause identifies 'keeper' rows, that is, rows that are carried forward from the base table for additional processing by SQL.

There ought to be a similar convention for carrying rows forward from the GROUPed intermediate working tables.

For example, we might be interested in seeing only those rows where the COUNT is greater than 1.

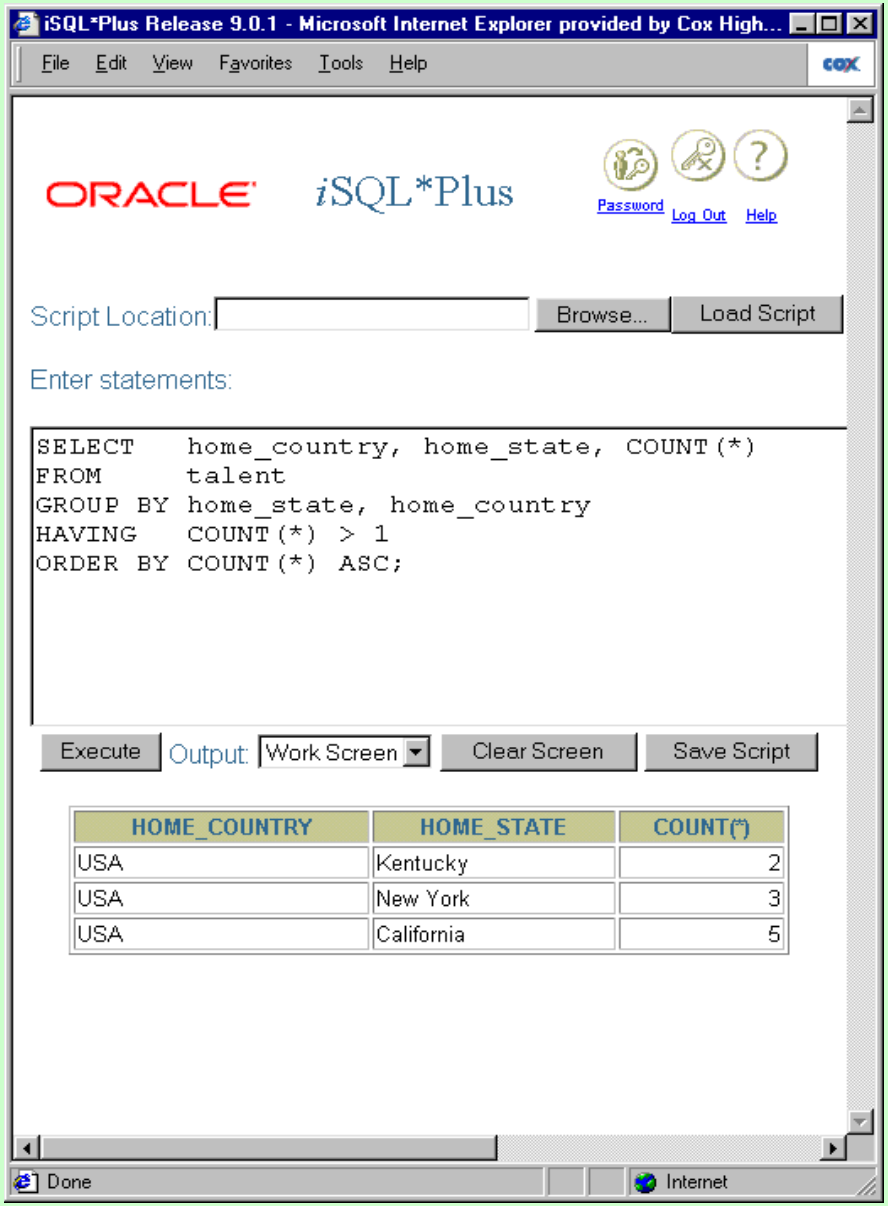


The HAVING clause operates similarly to the WHERE clause.

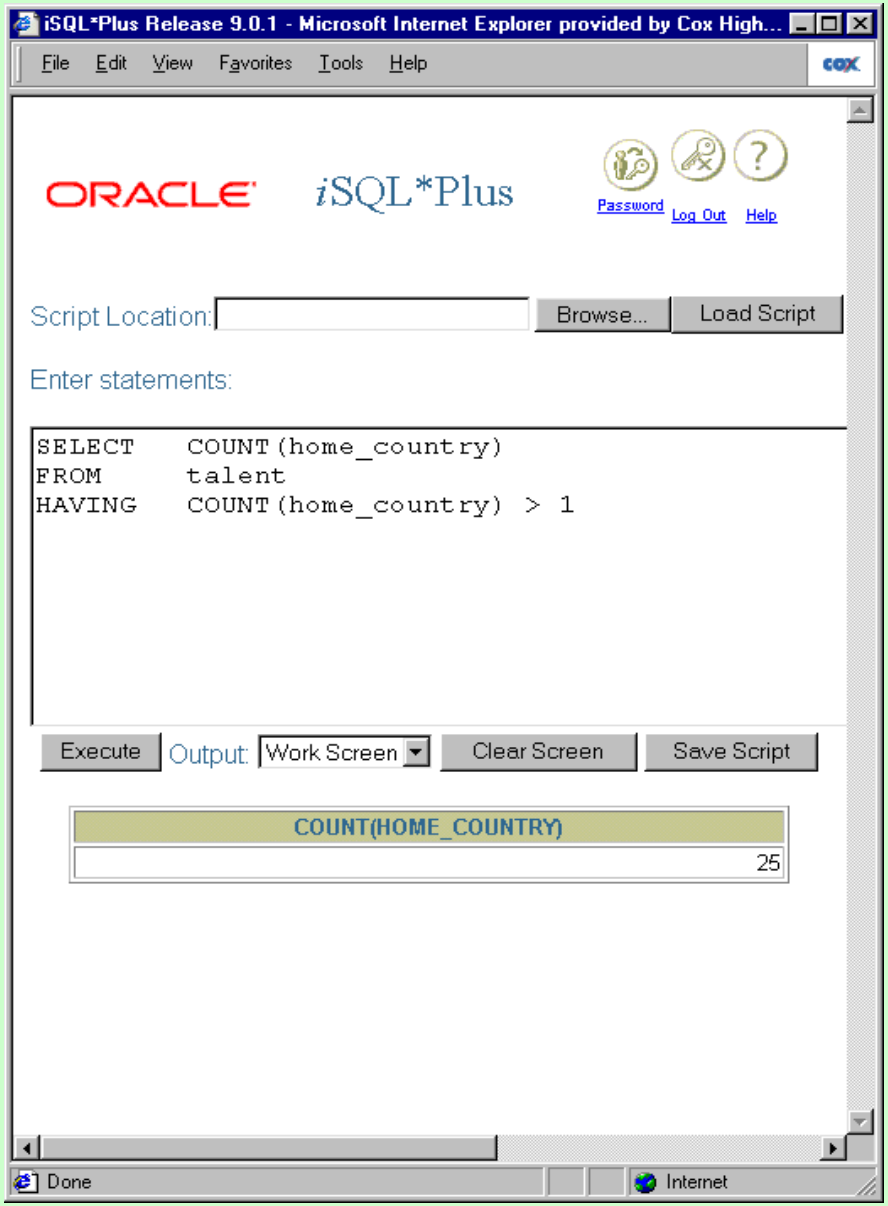
The WHERE clause identifies which rows carry forward from the base table thru the intermediate working tables.

The HAVING clause identifies which result rows, from GROUP-intermediate working tables are carried forward to the final result table.

Page C-3 ORDER BY (again)



The ORDER BY clause allows us to arrange the rows in the result table in whatever fashion suits us best.



Here's another question for the discussion forum.

← What's going on with this program?

MySQL provides a very novel, and very useful, group function that can be used to create a 'list'.

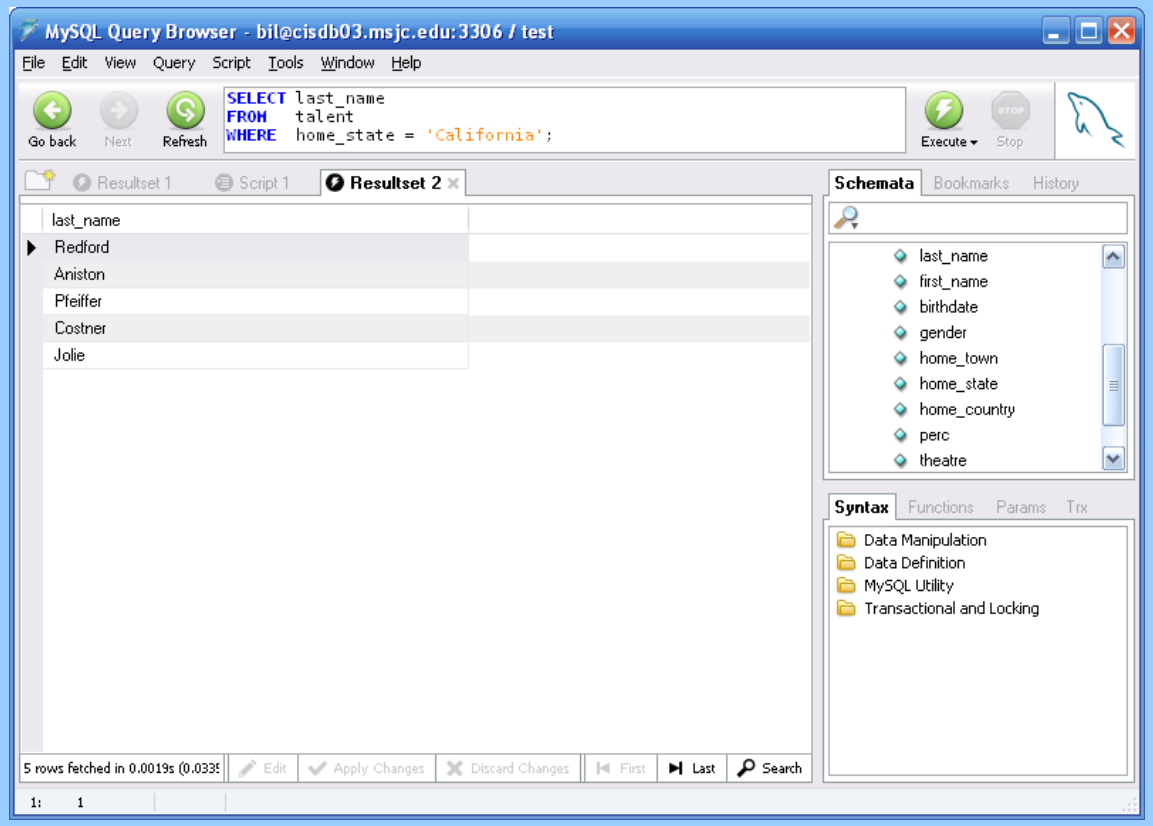
Note: This is a group function, and it could have been presented in the previous lecture but I didn't want to show you this until after you'd had a chance to learn about GROUPing.

This function might appear to be at odds with the intuitions you've been developing – but please bear with me.

	Oracle	MySQL
List		GROUP_CONCAT

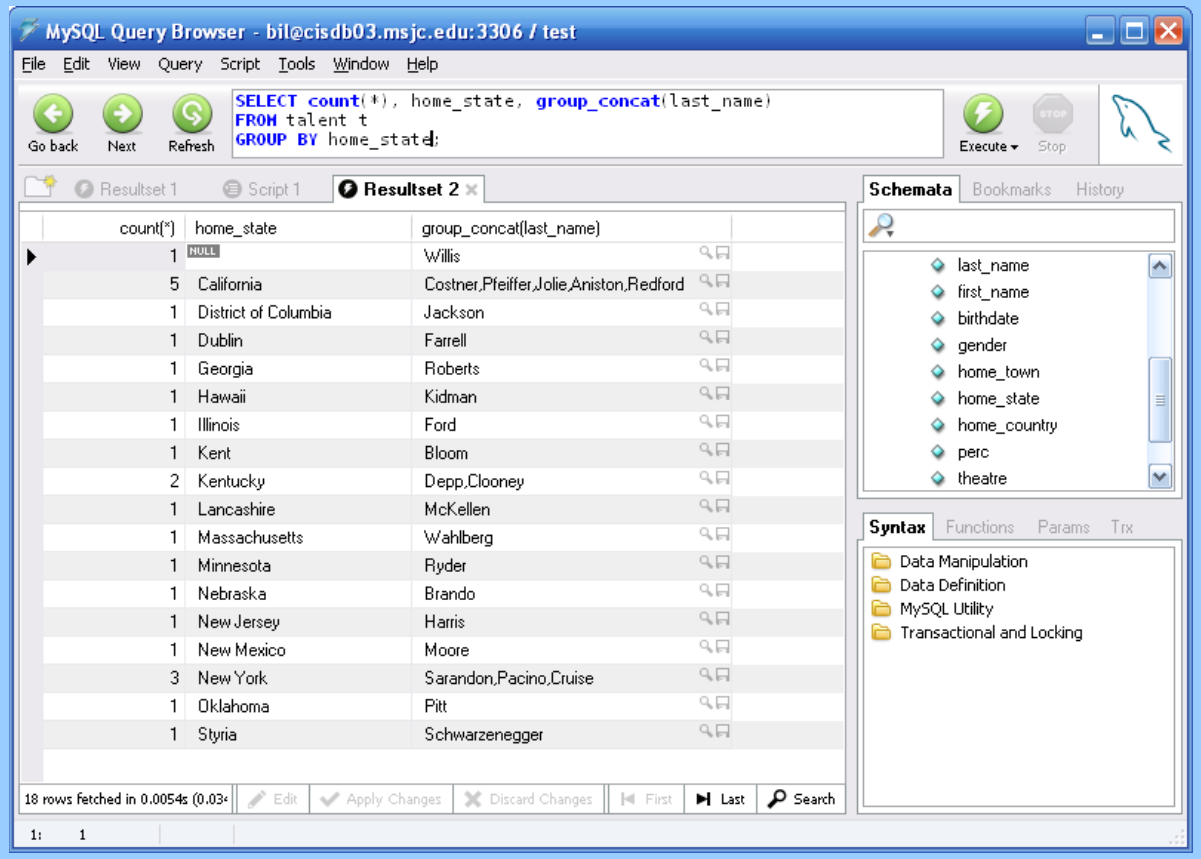
MySQL allows the programmer to create a list of items based on the values in a column that is being grouped.

Let's say I wanted to know the names of all of the actors in the Talens agency who live in California.



That's pretty simple.

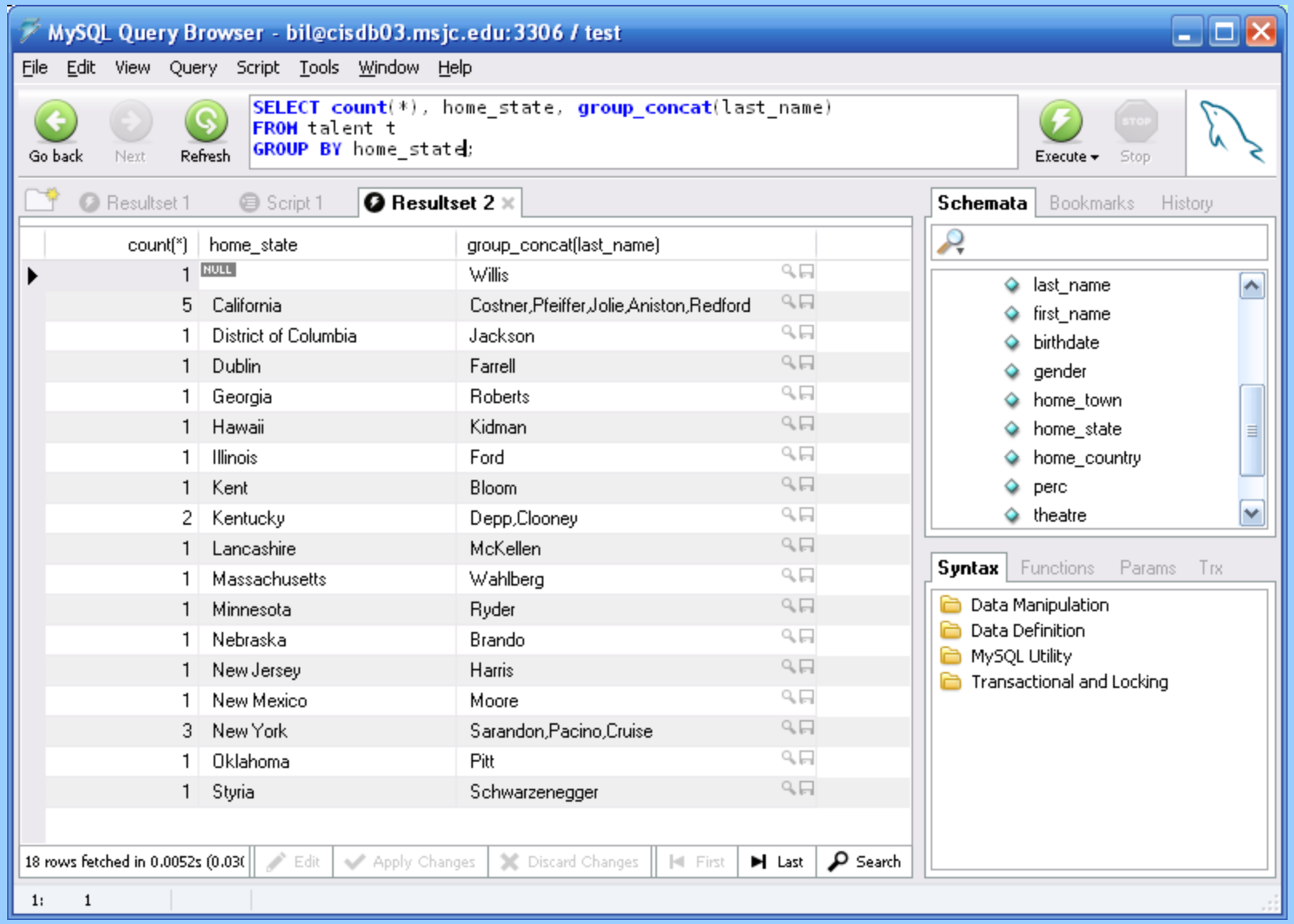
But now let's say I've been asked to prepare a report that requires the use of GROUPing – say something like, what is the breakdown by state, AND show us who's living there…



I'm thinking of a report like this

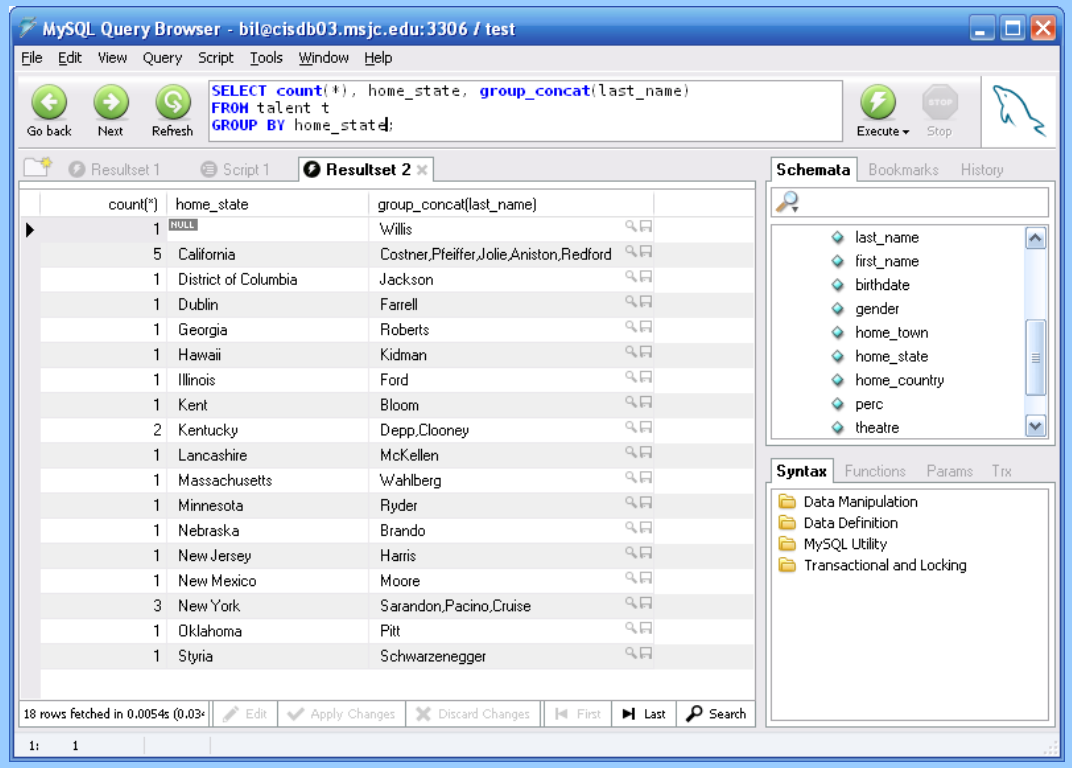
Just focus on the report, the result table

Here's a bigger image…



We created this report using the GROUP_CONCAT function.

The function creates a list. The list itself is a single item (it's displayed in a single column), and this list is composed as the concatenation of each of the named elements in that group.

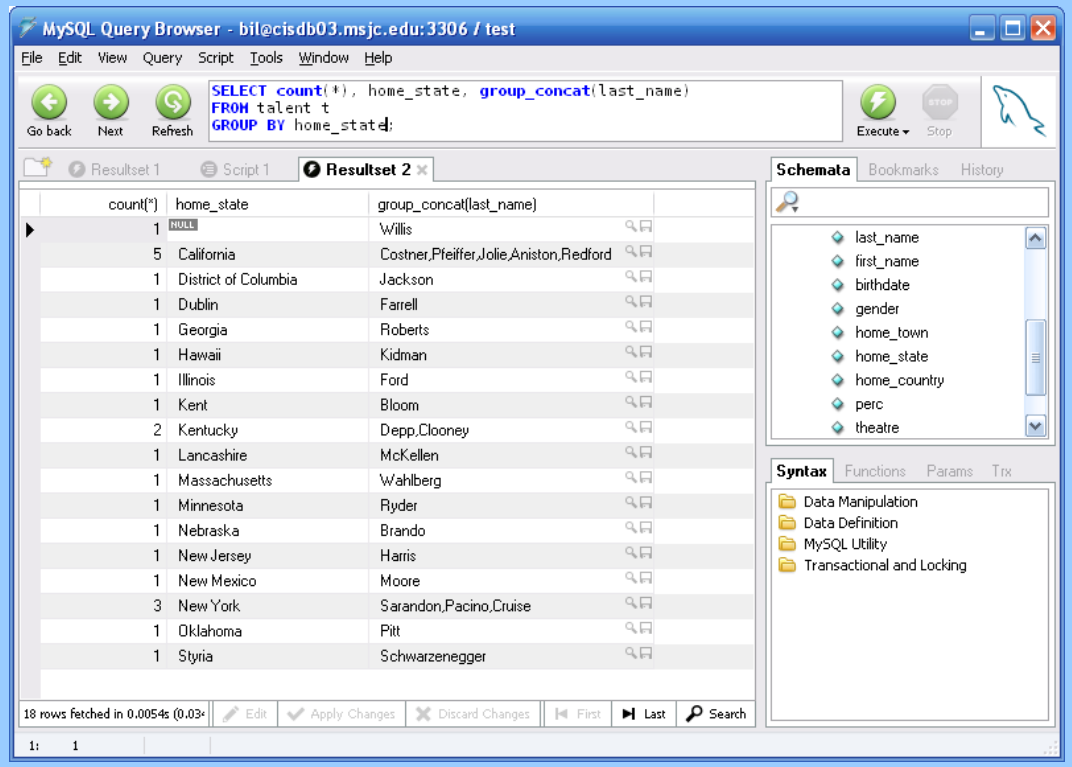


This collection of items might appear to be at odds with what we learned earlier. That is, GROUPing doesn't let you look at detail values.

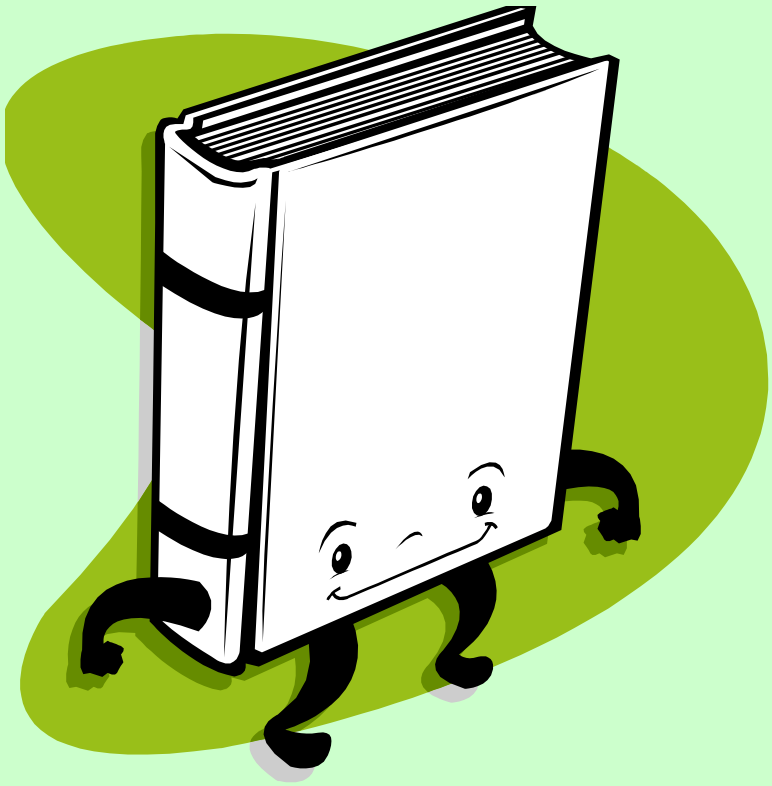
That's still the case. When GROUPing, the detail columns essentially disappear and information is no longer accessible at the single row level. The only data values we can access are:

1)the ones that are named in the GROUP BY clause, and

2)The aggregate functions



Page T-1: Terminology



GROUP BY
HAVING

List
GROUP_CONCAT

Page Z-1: End Notes

Please drop me an email if you noticed any errors in this module. I'd also appreciate reading your comments, criticisms, and or suggestions as to how this module could be improved.

Thanks,

bil

That’s All