GROUP BY and HAVING Clauses, ROLLUP and CUBE Operations, and GROUPING SETS

What Will I Learn?

In this lesson, you will learn to:

- Construct and execute a SQL query using GROUP BY
- Construct and execute a SQL query using GROUP BY … HAVING
- Construct and execute a GROUP BY on more than one column
- Nest group functions
- Use ROLLUP to produce subtotal values
- Use CUBE to produce cross-tabulation values
- Use GROUPING SETS to produce a single result set

Why Learn It?

What if you wanted to know the average height of all students? You could write a query that looks like this:

```
SELECT AVG(height) FROM students;
```

But what if you wanted to know the average height of the students based on their year in school? Right now, you would have to write a number of different SQL statements to accomplish this:

```
SELECT AVG(height) FROM students WHERE year_in_school = 10;
SELECT AVG(height) FROM students WHERE year_in_school = 11;
SELECT AVG(height) FROM students WHERE year_in_school = 12;
```

And so on! To simplify problems like this with just one statement you use the GROUP BY and HAVING clauses.

Why Learn It?

What if, once you have selected your groups and computed your aggregates across these groups, you also wanted subtotals per group and a grand total of all the rows selected.

You can obviously import the result into a spreadsheet application, you can get out your calculator or you can do the mental arithmetic. Or, if any of them seem like hard work, you can use some of the extensions to the GROUP BY clause: ROLLUP, CUBE and GROUPING SETS.

Using these extensions makes life a lot easier for you and they are all highly efficient to use, from the point of view of the database.

Tell Me / Show Me

GROUP BY

You use the GROUP BY clause to divide the rows in a table into smaller groups. You can then use the group functions to return summary information for each group.

In the SELECT statement shown, the rows are being grouped by department_id. The AVG function is then automatically applied to each group

```
SELECT department_id, AVG(salary)
FROM employees
GROUP BY department_id;
```

DEPARTMENT_ID	AVG (SALARY)
10	4400
20	9500
50	3500
60	6400
…	…

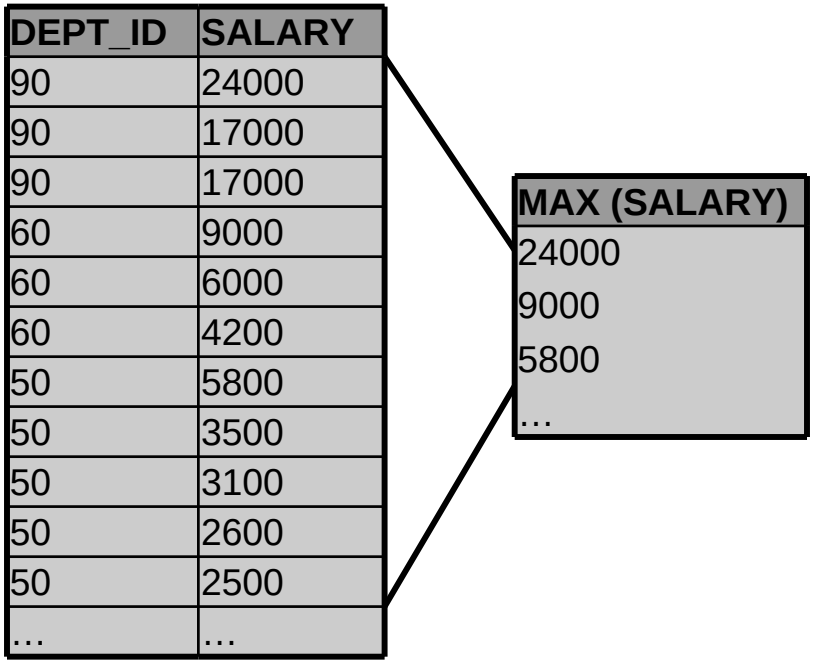
Tell Me / Show Me

GROUP BY

What if we wanted to find the maximum salary of employees in each department? We use a GROUP BY clause stating which column to use to group the rows.

```
SELECT MAX(salary)
FROM employees
GROUP BY department_id;
```

But how can we tell which maximum salary belongs to which department?

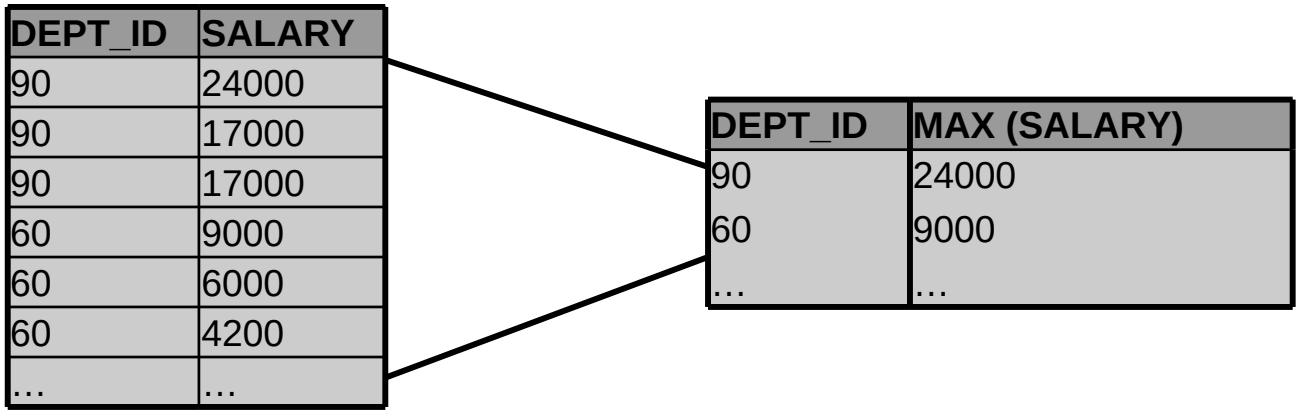


Tell Me / Show Me

GROUP BY

Usually we want to include the GROUP BY column in the SELECT list.

```
SELECT department_id, MAX(salary)
FROM employees
GROUP BY department_id;
```



DEPT_ID	SALARY
90	24000
90	17000
90	17000
60	9000
60	6000
60	4200
…	…

DEPT_ID	MAX (SALARY)
90	24000
60	9000
…	…

Tell Me / Show Me

GROUP BY

Group functions require that any column listed in the SELECT clause that is not part of a group function must be listed in a GROUP BY clause.

What is wrong with this example?

```
SELECT job_id, last_name, AVG(salary)
FROM employees
GROUP BY job_id;
```

ORA-00979: not a GROUP BY expression

Tell Me / Show Me

COUNT

```
SELECT count(first_name), shirt_color
FROM students
GROUP BY shirt_color
```

This example shows how many students wear shirts of each color.

Remember that group functions ignore null values, so if any student does not have a first name, he or she will not be included in the COUNT. Of course this is unlikely, but when constructing SQL statements we have to think about all the possibilities.

It would be better to start with:

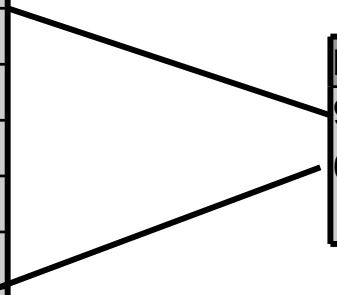
SELECT COUNT(*), shirt_color

Tell Me / Show Me

We can also use a WHERE clause to exclude rows before the remaining rows are formed into groups.

```
SELECT department_id, MAX(salary)
FROM employees
WHERE last_name <> 'King'
GROUP BY department_id;
```

LAST_NAME	DEPT_ID	SALARY
King	90	24000
Kochhar	90	17000
De Haan	90	17000
Hunold	60	9000
Ernst	60	6000
Lorentz	60	4200
…	…	…



DEPT_ID	MAX (SALARY)
90	17000
60	9000
…	…

Tell Me / Show Me

More GROUP BY Examples:

1. Show the average graduation rate of the schools in several cities; include only those students who have graduated in the last few years

```
SELECT AVG(graduation_rate), city
FROM students
WHERE graduation_date >= '01-JUN-07'
GROUP BY city;
```

2. Count the number of students in the school, grouped by grade; include all students

```
SELECT COUNT(first_name), grade
FROM students
GROUP BY grade;
```

Tell Me / Show Me

GROUP BY Guidelines

Important guidelines to remember when using a GROUP BY clause are:

- If you include a group function (AVG, SUM, COUNT, MAX, MIN, STDDEV, VARIANCE) in a SELECT clause and any other individual columns, each individual column must also appear in the GROUP BY clause.
- You cannot use a column alias in the GROUP BY clause.
- The WHERE clause excludes rows before they are divided into groups.

Tell Me / Show Me

GROUPS WITHIN GROUPS

Sometimes you need to divide groups into smaller groups. For example, you may want to group all employees by department; then, within each department, group them by job.

This example shows how many employees are doing each job within each department.

```
SELECT department_id, job_id, count(*)
FROM employees
WHERE department_id > 40
GROUP BY department_id, job_id;
```

DEPT_ID	JOB_ID	COUNT(*)
50	ST_MAN	1
50	ST_CLERK	4
60	IT_PROG	3
80	SA_MAN	1
80	SA_REP	2
…	…	…

Tell Me / Show Me

NESTING GROUP FUNCTIONS

Group functions can be nested to a depth of two when GROUP BY is used.

```
SELECT max(avg(salary))
FROM employees
GROUP by department_id;
```

How many values will be returned by this query? The answer is one – the query will find the average salary for each department, and then from that list, select the single largest value.

Tell Me / Show Me

HAVING

Suppose we want to find the maximum salary in each department, but only for those departments which have more than one employee? What is wrong with this example?

```
SELECT department_id, MAX(salary)
FROM employees
WHERE COUNT(*) > 1
GROUP BY department_id;
```

ORA-00934: group function is not allowed here

The next slide solves this problem.

Tell Me / Show Me

HAVING

In the same way you used the WHERE clause to restrict the rows that you selected, you can use the HAVING clause to restrict groups.

In a query using a GROUP BY and HAVING clause, the rows are first grouped, group functions are applied, and then only those groups matching the HAVING clause are displayed.

The WHERE clause is used to restrict rows; the HAVING clause is used to restrict groups returned from a GROUP BY clause.

```
SELECT department_id, MAX(salary)
FROM employees
GROUP BY department_id
HAVING COUNT(*) > 1;
```

DEPARTMENT_ID	MAX(SALARY)
20	13000
50	5800
60	9000
…	…

Tell Me / Show Me

HAVING

Although the HAVING clause can precede the GROUP BY clause in a SELECT statement, it is recommended that you place each clause in the order shown. The ORDER BY clause (if used) is always last!

SELECT column, group_function
FROM table
WHERE
GROUP BY
HAVING
ORDER BY

Tell Me / Show Me

ROLLUP

In GROUP BY queries you are quite often required to produce subtotals and totals, and the ROLLUP operation can do that for you.

The action of ROLLUP is straightforward: it creates subtotals that roll up from the most detailed level to a grand total, following a grouping list specified in the ROLLUP clause. ROLLUP takes as its argument an ordered list of grouping columns. First, it calculates the standard aggregate values specified in the GROUP BY clause. Then, it creates progressively higher-level subtotals, moving from right to left through the list of grouping columns. Finally, it creates a grand total.

Tell Me / Show Me

ROLLUP

In the following result table the rows in red are generated by the ROLLUP operation:

```
SELECT department_id, job_id, SUM(salary)
FROM   employees
WHERE department_id < 50
GROUP BY ROLLUP (department_id, job_id)
```

DEPARTMENT_ID	JOB_ID	SUM (SALARY)	
10	AD_ASST	4400	
10	-	4400	← Subtotal for dept 10
20	MK_MAN	13000	
20	MK_REP	6000	
20	-	19000	← Subtotal for dept 20
-	-	23400	← Grand Total for report

Tell Me / Show Me

CUBE

CUBE is an extension to the GROUP BY clause like ROLLUP. It produces cross-tabulation reports.

It can be applied to all aggregate functions including AVG, SUM, MIN, MAX and COUNT.

Columns listed in the GROUP BY clause are cross-referenced to create a superset of groups. The aggregate functions specified in the SELECT list are applied to this group to create summary values for the additional super-aggregate rows. Every possible combination of rows is aggregated by CUBE. If you have *n* columns in the GROUP BY clause, there will be 2^n possible super-aggregate combinations. Mathematically these combinations form an *n*-dimensional cube, which is how the operator got its name.

Tell Me / Show Me

CUBE

CUBE is typically most suitable in queries that use columns from multiple tables rather than columns representing different rows of a single table.

Imagine for example a user querying the Sales table for a company like AMAZON.COM. For instance, a commonly requested cross-tabulation might need subtotals for all the combinations of Month, Region and Product.

These are three independent tables, and analysis of all possible subtotal combinations is commonplace. In contrast, a cross-tabulation showing all possible combinations of year, month and day would have several values of limited interest, because there is a natural hierarchy in the time table. Subtotals such as profit by day of month summed across year would be unnecessary in most analyses. Relatively few users need to ask "What were the total sales for the 16th of each month across the year?"

Tell Me / Show Me

CUBE

In the following statement the rows in red are generated by the CUBE operation:

```
SELECT department_id, job_id, SUM(salary)
FROM   employees
WHERE department_id < 50
GROUP BY CUBE (department_id, job_id)
```

DEPARTMENT_ID	JOB_ID	SUM(SALARY)	
-	-	23400	← Total for report
-	MK_MAN	13000	← Subtotal for MK_MAN
-	MK_REP	6000	← Subtotal for MK_REP
-	AD_ASST	4400	← Subtotal for AD_ASST
10	-	4400	← Subtotal for dept 10
10	AD_ASST	4400	
20	-	19000	← Subtotal for dept 20
20	MK_MAN	13000	
20	MK_REP	6000	

Tell Me / Show Me

GROUPING SETS

GROUPING SETS is another extension to the GROUP BY clause, like ROLLUP and CUBE. It is used to specify multiple groupings of data. It is like giving you the possibility to have multiple GROUP BY clauses in the same SELECT statement, which is not allowed in the syntax.

The point of GROUPING SETS is that if you want to see data from the EMPLOYEES table grouped by (department_id, job_id , manager_id), but also by (department_id, manager_id) and also by (job_id, manager_id) then you would normally have to write 3 different select statements with the only difference being the GROUP BY clauses. For the database this means retrieving the same data in this case 3 times, and that can be quite a big overhead. Imagine if your company had 3,000,000 employees. Then you are asking the database to retrieve 9 million rows instead of just 3 million rows – quite a big difference.

So GROUPING SETS are much more efficient when writing complex reports.

Tell Me / Show Me

GROUPING SETS

In the following statement the rows highlighted in color are generated by the GROUPING SETS operation:

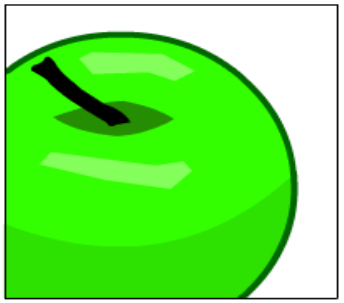
```
SELECT department_id, job_id, manager_id, SUM(salary)
FROM employees
WHERE department_id < 50
GROUP BY GROUPING SETS
((job_id, manager_id), (department_id, job_id), (department_id, manager_id))
```

DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)
-	MK_MAN	100	13000
-	MK_MAN	201	6000
-	AD_ASST	101	4400
10	AD_ASST	-	4400
20	MK_MAN	-	13000
20	MK_REP	-	6000
10	-	101	19000
20	-	100	13000
20	-	201	6000

Tell Me / Show Me

Terminology

Key terms used in this lesson include:



GROUP BY

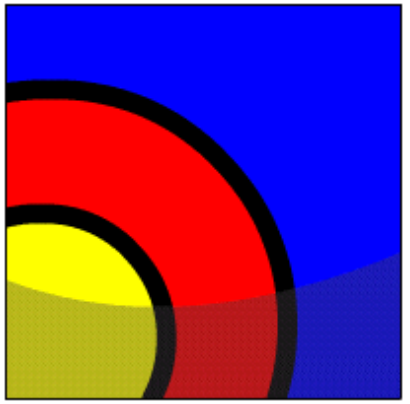
HAVING

ROLLUP

CUBE

GROUPING SETS

Summary



In this lesson you have learned to:

- Construct and execute a SQL query using GROUP BY
- Construct and execute a SQL query using GROUP BY … HAVING
- Construct and execute a GROUP BY on more than one column
- Nest group functions
- Use ROLLUP to produce subtotal values
- Use CUBE to produce cross-tabulation values
- Use GROUPING SETS to produce a single result set

Summary

Practice Guide

The link for the lesson practice guide can be found in the course resources in Section 0.