Chapter 8

GROUPING DATA

- GROUP BY clause
- Group functions
- HAVING clause
- ROLLUP and CUBE

In this chapter, we will see how to group data to get summary information. We have so far seen WHERE, ORDER BY clauses of SELECT command. In this chapter, we will see GROUP BY and HAVING clauses. We will also see how to use group functions and two new function introduced in Oracle8i - ROLLUP and CUBE.

GROUP BY Clause

Some times, we have to group the rows of the table by a particular column and perform certain operations on each group. For example, you may want to display the number of students in each batch. For this we have to group rows of STUDENTS table based on the column BCODE. This will create groups – one for each unique batch code. Then using group function COUNT we can count the number of rows (equivalent to number of students) in each group.

Now let us see how the above-mentioned task can be accomplished. The following SELECT command groups rows of STUDENTS table on BCODE using GROUP BY clause. Then it counts number of students we have in each batch using group function COUNT.

```
select  bcode, count(rollno)
from  students
group by bcode;

BCODE COUNT(ROLLNO)
----- -------------
b1                2
b2                3
b3                2
b4                1
b5                3
```

We will see one more example. In this we will display the total amount of fee paid by each student. For this we use PAYMENTS table and group function SUM. Here is the required query.

```
select  rollno, sum(amount) from payments group by rollno;

   ROLLNO SUM(AMOUNT)
--------- -----------
        1        4500
        2        4500
        3        5000
        4        5000
        5        5000
        6        3500
        7        3500
        8        4000
        9        3000
       10        6500
       11        4500
```

While you are grouping the data the following points are to be taken in to account.

- Find out the table from where the data is to be taken.
- Find out the column or columns on which the data is to be grouped.
- Find out the group function that is to be used to get required aggregate.

GROUP BY clause is used to specify the column(s) on which the rows of the table are to be grouped. It means it divides the rows into different groups based on the column(s) that are given in GROUP BY clause. For example, in the above example, the rows of STUDENTS table are to be grouped based on the value of column ROLLNO. So all rows that contain the same roll number will be taken as one group. Like this the number of groups will be equivalent to the number of unique roll numbers in the table.

Points to remember

It is important to remember that only the following can be selected in SELECT while you are grouping the data.

- The columns that are given in GROUP BY clause can be selected.
- Group function

Group Functions

Group functions are the functions that are used to perform operations on groups. Table 1 lists all the available group functions. The general syntax of group functions is given below.

```
group-function (DISTINCT  column)
```

If DISTINCT option is used then only distinct (unique) values are taken into account. The following query is used to count the number of students who have paid some amount.

```
select  count(rollno) from payments;

COUNT(ROLLNO)
-------------
           17
```

But the count includes the duplicates. That means, if the same student has made the payment twice then his number will be counted for twice. But what if I want to count how many students have made some payment without taking duplicates?. The answer is DISTINCT clause in the group function as shown in the next query.

```
select  count(DISTINCT rollno) from payments;

COUNT(DISTINCTROLLNO)
---------------------
                   11
```

The DISTINCT clause is making COUNT function count only distinct value and not all values.

***Note**: COUNT(*) counts the number of rows in each group. Asterisk (*) refers to the entire row. You can use any column name instead to count the number of not null values in that column.*

Group Function	Description
SUM	Returns the sum of the given column.
AVG	Returns the average of the given column.
MIN	Returns the minimum value in the given column.
MAX	Returns the maximum value in the given column.
STDDEV	Returns the standard deviation of the values in the given column.
VAR	Returns the variance of the values in the given column..
COUNT	Returns the number of not null values in the given column. Returns number of rows, if * is given instead of column.

Table 1: GROUP functions.

The following query displays the subject and date on which the most recent batch of that subject has started.

```
select  ccode,max(stdate)
from batches
group by ccode;

CCODE MAX(STDAT
----- ---------
asp   15-JAN-01
c     20-JAN-01
java  05-APR-01
ora   15-AUG-01
vbnet 12-JUL-01
xml   02-MAR-01
```

It is also possible to use group functions without using GROUP BY clause. In this case the entire selected rows will be taken as a single group and group function performs the operation on the entire set of selected rows.

The following query will display the highest course fee.

```
select max(fee) from courses;

 MAX(FEE)
---------
      5500
```

GROUP BY and WHERE clause

If WHERE clause is used along with GROUP BY then WHERE clause is executed first. The rows that are selected by WHERE clause will be grouped by GROUP BY clause. The following query shows the number of batches that started for each subject in the last 3 months.

```
select  ccode, count(*) count
from  batches
where   months_between(sysdate,stdate) <= 3
group by ccode;

CCODE     COUNT
----- ---------
ora           1
vbnet         1
```

In the above query, first rows that satisfy the given condition are retrieved. Then these rows are passed to GROUP BY clause for grouping.

***Note**: GROUP BY clause should follow WHERE clause, if both are used together.*

WHERE clause is used to select the rows that are to be used for grouping.

Grouping on more than one column

It is possible to group rows on more than one column. In this case, the first columns is used to group the data, and then within that group records are again grouped based on second column, and so on.

The following query displays the date on which the last batch was taken by each faculty for each subject.

```
select  fcode, ccode, max(stdate)
from  batches
group by  fcode,ccode;

FCODE CCODE MAX(STDAT
----- ----- ---------
da    asp   15-JAN-01
da    vbnet 12-JUL-01
dh    xml   02-MAR-01
gk    ora   12-JAN-01
hs    c     20-JAN-01
hs    java  05-APR-01
jc    ora   15-AUG-01
```

HAVING Clause

What if you have to select only a few groups based on the condition that is using the result of one of the group functions?. For example, what if you have to display the batches that have more than 10 students. HAVING clause is used for this purpose. It is used to filter groups based on the given condition. The following example shows the courses that have got more than 1 batch.

```
select  ccode
from batches
group by  ccode
having count(*) > 1;
```

```
CCODE
-----
ora
```

In the above example, first rows of BATCHES table are grouped on CCODE. Then the query selects the groups that contain more than 1 row. The later is achieved using HAVING clause, which is specially meant to select groups based on the given condition.

Here is a list of important points that are to be noted:

- WHERE clause can be used to check for conditions based on values of columns and expressions related to individual rows. It cannot be used with conditions related to groups.
- HAVING clause is specially designed to evaluate the conditions that are based on group functions such as SUM, and COUNT.
- HAVING clause cannot be used for conditions that are not related to groups.

The following few example will illustrate the above-mentioned important points.

```
SQL> select  ccode
  2  from batches
  3  where count(*) > 1
  4  group by ccode;
where count(*) > 1
      *
ERROR at line 3:
ORA-00934: group function is not allowed here
```

The above example returns error because WHERE clause cannot be used with group functions.

```
SQL> select  ccode, count(*)
  2  from batches
  3  group by ccode
  4  having  enddate is null;
having  enddate is null
        *
ERROR at line 4:
ORA-00979: not a GROUP BY expression
```

In the above example as we tried to use normal condition using HAVING clause Oracle returned an error saying that the given expression is not acceptable.

The following query is valid as HAVING clause is used with a condition that is related to group by expression – CCODE.

```
SQL> select ccode, count(*)
  2  from batches
  3  group by ccode
  4  having  length(ccode) > 3;

CCODE  COUNT(*)
----- ---------
java           1
vbnet          1
```

Using WHERE and HAVING together

It is possible to use both WHERE and HAVING clauses together. When these two clauses are used together, first WHERE clause is to be given then HAVING clause.

Oracle first selects rows based on WHERE clause. After that it groups the selected data. Then groups are selected based on HAVING clause.

The following query displays the courses that we started for more than once in the last six months.

```
select ccode, count(*)
from batches
where  months_between(sysdate,stdate) <= 6
group by ccode
having  count(*) > 1;

CCODE  COUNT(*)
----- ---------
ora        2
```

ORDER BY with GROUP BY

ORDER BY clause can be used along with GROUP BY clause to order the results. When used ORDER BY clause must be the last clause to be used in the query.

```
select fcode, count(*)
from course_faculty
group by fcode
order by  count(*);

FCODE  COUNT(*)
----- ---------
dh            1
gk            1
kl            1
jc            1
sw            1
da            2
hs            2
jj            3
```

***Note**: When ORDER BY is used with GROUP BY clause, ORDER BY should have either the group by column or group function.*

Order of execution

Here is the order Oracle uses to execute different clauses given in SELECT command.

- Selects rows based on WHERE clause.
- Groups rows based on GROUP BY clause.
- Calculates results for each group.
- Eliminates groups based on HAVING clause.
- Then uses ORDER BY to order the results.

ROLLUP and CUBE

Oracle8i enhanced grouping by adding ROLLUP and CUBE. These two provide totals at multiple dimensions.

The following is the syntax of ROLLUP.

```
SELECT ...
GROUP BY ROLLUP (columns);
```

ROLLUP creates sub totals from most detailed to grand total. It moves from right to left in the list of columns given in ROLLUP.

Before we use ROLLUP, let us see the result of simple group by clause using BATCHES table.

```
select ccode,fcode, count(*)
from  batches
group by ccode,fcode;
```

```
CCODE FCODE  COUNT(*)
----- ----- ---------
asp   da            1
c     hs            1
c     kl            1
java  hs            1
ora   gk            2
ora   kl            1
vbnet da            2
xml   dh            1
```

In case of normal GROUP BY clause we get count of each course and faculty. But what if you want to get total number of batches for each course and also the total number of batches.

The following ROLLUP creates the required aggregates.

```
select ccode,fcode, count(*)
from  batches
group by rollup(ccode,fcode);
```

```
CCODE FCODE  COUNT(*)
----- ----- ---------
asp   da            1
asp                 1
c     hs            1
c     kl            1
c                   2
java  hs            1
java                1
ora   gk            2
ora   kl            1
ora                 3
vbnet da            2
vbnet               2
xml   dh            1
xml                 1
                   10
```

Apart from rows that come from GROUP BY, ROLLUP generates new rows that display the number of batches for each course and also total number of batches.

In case of total number of batches of a single course the CCODE is contains course code and FCODE contains null. In case of total number of batches both CCODE and FCODE contain null value.

The following query will display a meaningful value for these columns.

```
select nvl(ccode,'ALL courses'), nvl(fcode,'All faculty'), count(*)
from  batches
group by rollup(ccode,fcode);
```

```
NVL(CCODE,' NVL(FCODE,'  COUNT(*)
----------- ----------- ---------
asp         da                  1
asp         All faculty         1
c           hs                  1
c           kl                  1
c           All faculty         2
java        hs                  1
java        All faculty         1
ora         gk                  2
ora         kl                  1
ora         All faculty         3
vbnet       da                  2
vbnet       All faculty         2
xml         dh                  1
xml         All faculty         1
ALL courses All faculty        10
```

GROUPING function

This will return a value of 1 if the column's value is generated by ROLLUP. So we can use DECODE and GROUPING functions to get the same result as the above.

```
select decode(grouping(ccode),1,'ALL courses',ccode)  ccode,
       decode(grouping(fcode),1,'All faculty',fcode)  fcode,
       count(*) count
from  batches
group by rollup(ccode,fcode);
```

```
CCODE       FCODE           COUNT
----------- ----------- ---------
asp         da                  1
asp         All faculty         1
c           hs                  1
c           kl                  1
c           All faculty         2
java        hs                  1
java        All faculty         1
ora         gk                  2
ora         kl                  1
```

```
ora         All faculty         3
vbnet       da                  2
vbnet       All faculty         2
xml         dh                  1
xml         All faculty         1
ALL courses All faculty        10
```

CUBE

This generates the same subtotals as ROLLUP and plus a few more. This provides all possible subtotals. For example in the previous output of ROLLUP we got the number of batches taken by each faculty for each course, number of batches for each course and total number of bathes.

CUBE apart from generating all these can also generate subtotals for each faculty. That means it provides number of batches taken by each faculty also.

```
select decode(grouping(ccode),1,'ALL courses',ccode)  ccode,
       decode(grouping(fcode),1,'All faculty',fcode)  fcode,
       count(*) count
from  batches
group by cube(ccode,fcode);

CCODE       FCODE           COUNT
----------- ----------- ---------
asp         da                  1
asp         All faculty         1
c           hs                  1
c           kl                  1
c           All faculty         2
java        hs                  1
java        All faculty         1
ora         gk                  2
ora         kl                  1
ora         All faculty         3
vbnet       da                  2
vbnet       All faculty         2
xml         dh                  1
xml         All faculty         1
ALL courses da                  3
ALL courses dh                  1
ALL courses gk                  2
ALL courses hs                  2
ALL courses kl                  2
ALL courses All faculty        10
```

CUBE adds five more rows to the output of ROLLUP. Each of these new rows is to display the total number of batches taken by each faculty for all courses.

Summary

GROUP BY clause is used to group the rows of the table based on the given columns. Group functions can be used to calculate aggregates like average of each group. HAVING clause is used to filter groups based on the result of group function. Oracle executes first WHERE then GROUP BY then HAVING and finally ORDER BY.

Exercises

1. _____ clause is used to select groups based on condition.
2. Select count(*) from students; Is it a valid query.
3. What is the order of WHERE, GROUP BY and ORDER BY.
4. Display ROLLNO of students who have paid for more than twice.
5. Display average time taken for subject ORA.
6. Display faculty who can take more than 2 courses.
7. Display least course fee.
8. Display the number of months between first and last batches of course Java.
9. Display Year, course and number of batches of that course.
10. Display faculty who has got A grade for more than 1 subject.
11. Display the number of students joined in each month.
12. Display the number of students joined in each month of the current year.