17

Enhancements to the GROUP BY Clause

Objectives

After completing this lesson, you should be able to do the following:

- Use the `ROLLUP` operation to produce subtotal values
- Use the `CUBE` operation to produce cross-tabulation values
- Use the `GROUPING` function to identify the row values created by `ROLLUP` or `CUBE`
- Use `GROUPING SETS` to produce a single result set

Review of Group Functions

Group functions operate on sets of rows to give one result per group.

```
SELECT        [column,] group_function(column). . .
FROM          table
[WHERE        condition]
[GROUP BY     group_by_expression]
[ORDER BY     column];
```

Example:

```
SELECT AVG(salary), STDDEV(salary),
       COUNT(commission_pct),MAX(hire_date)
FROM   employees
WHERE  job_id LIKE 'SA%';
```

Review of the GROUP BY Clause

Syntax:

```
SELECT        [column,] group_function(column). . .
FROM          table
[WHERE        condition]
[GROUP BY     group_by_expression]
[ORDER BY     column];
```

Example:

```
SELECT   department_id, job_id, SUM(salary),
         COUNT(employee_id)
FROM     employees
GROUP BY department_id, job_id ;
```

Review of the HAVING Clause

```
SELECT      [column,] group_function(column)...
FROM        table
[WHERE      condition]
[GROUP BY   group_by_expression]
[HAVING     having_expression]
[ORDER BY   column];
```

- Use the HAVING clause to specify which groups are to be displayed.
- You further restrict the groups on the basis of a limiting condition.

GROUP BY with ROLLUP and CUBE Operators

- Use ROLLUP or CUBE with GROUP BY to produce superaggregate rows by cross-referencing columns.
- ROLLUP grouping produces a results set containing the regular grouped rows and the subtotal values.
- CUBE grouping produces a results set containing the rows from ROLLUP and cross-tabulation rows.

ROLLUP Operator

```
SELECT      [column,] group_function(column). . .
FROM        table
[WHERE      condition]
[GROUP BY   [ROLLUP] group_by_expression]
[HAVING     having_expression];
[ORDER BY   column];
```

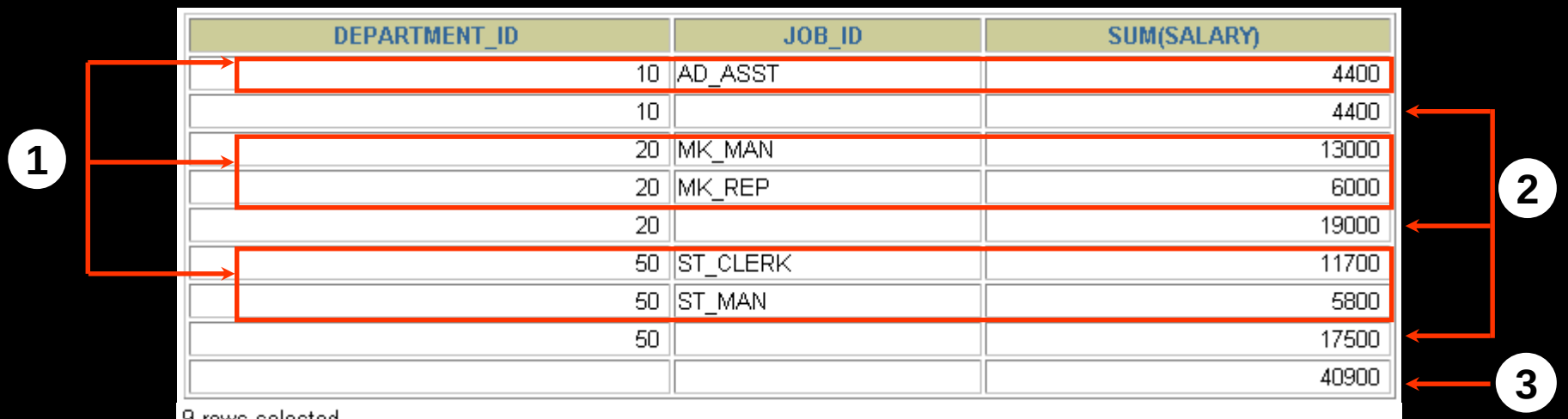
- ROLLUP is an extension to the GROUP BY clause.
- Use the ROLLUP operation to produce cumulative aggregates, such as subtotals.

ROLLUP Operator Example

```
SELECT    department_id, job_id, SUM(salary)
FROM      employees
WHERE     department_id < 60
GROUP BY ROLLUP(department_id, job_id);
```

DEPARTMENT_ID	JOB_ID	SUM(SALARY)
10	AD_ASST	4400
10		4400
20	MK_MAN	13000
20	MK_REP	6000
20		19000
50	ST_CLERK	11700
50	ST_MAN	5800
50		17500
		40900

9 rows selected.



CUBE Operator

```
SELECT      [column,] group_function(column)...
FROM        table
[WHERE      condition]
[GROUP BY   [CUBE] group_by_expression]
[HAVING     having_expression]
[ORDER BY   column];
```

- CUBE is an extension to the GROUP BY clause.
- You can use the CUBE operator to produce cross-tabulation values with a single SELECT statement.

CUBE Operator: Example

```
SELECT    department_id, job_id, SUM(salary)
FROM      employees
WHERE     department_id < 60
GROUP BY CUBE (department_id, job_id) ;
```

DEPARTMENT_ID	JOB_ID	SUM(SALARY)
10	AD_ASST	4400
10		4400
20	MK_MAN	13000
20	MK_REP	6000
20		19000
50	ST_CLERK	11700
50	ST_MAN	5800
50		17500
	AD_ASST	4400
	MK_MAN	13000
	MK_REP	6000
	ST_CLERK	11700
	ST_MAN	5800
		40900

14 rows selected.

1

2

3

4

GROUPING Function

```
SELECT     [column,] group_function(column) . ,
           GROUPING(expr)
FROM       table
[WHERE     condition]
[GROUP BY [ROLLUP][CUBE] group_by_expression]
[HAVING    having_expression]
[ORDER BY column];
```

- The GROUPING function can be used with either the CUBE or ROLLUP operator.
- Using the GROUPING function, you can find the groups forming the subtotal in a row.
- Using the GROUPING function, you can differentiate stored NULL values from NULL values created by ROLLUP or CUBE.
- The GROUPING function returns 0 or 1.

GROUPING Function: Example

```
SELECT    department_id DEPTID, job_id JOB,
          SUM(salary),
          GROUPING(department_id) GRP_DEPT,
          GROUPING(job_id) GRP_JOB
FROM      employees
WHERE     department_id < 50
GROUP BY ROLLUP(department_id, job_id);
```

DEPTID	JOB	SUM(SALARY)	GRP_DEPT	GRP_JOB
10	AD_ASST	4400	0	0
10		4400	0	1
20	MK_MAN	13000	0	0
20	MK_REP	6000	0	0
20		19000	0	1
		23400	1	1

6 rows selected.

1

2

3

GROUPING SETS

- GROUPING SETS are a further extension of the GROUP BY clause.
- You can use GROUPING SETS to define multiple groupings in the same query.
- The Oracle Server computes all groupings specified in the GROUPING SETS clause and combines the results of individual groupings with a UNION ALL operation.
- Grouping set efficiency:
  - Only one pass over the base table is required.
  - There is no need to write complex UNION statements.
  - The more elements the GROUPING SETS have, the greater the performance benefit.

GROUPING SETS: Example

```
SELECT    department_id, job_id,
          manager_id,avg(salary)
FROM      employees
GROUP BY GROUPING SETS
((department_id,job_id), (job_id,manager_id));
```

DEPARTMENT_ID	JOB_ID	MANAGER_ID	AVG(SALARY)
10	AD_ASST		4400
20	MK_MAN		13000
20	MK_REP		6000
50	ST_CLERK		2925
...			
	SA_MAN	100	10500
	SA_REP	149	8866.66667
	ST_CLERK	124	2925
	ST_MAN	100	5800

26 rows selected.

1

2

Composite Columns

- A composite column is a collection of columns that are treated as a unit.

```
ROLLUP (a, (b,c), d)
```

- To specify composite columns, use the `GROUP BY` clause to group columns within parentheses so that the Oracle server treats them as a unit while computing `ROLLUP` or `CUBE` operations.
- When used with `ROLLUP` or `CUBE`, composite columns would mean skipping aggregation across certain levels.

Composite Columns: Example

```
SELECT    department_id, job_id, manager_id,
          SUM(salary)
FROM      employees
GROUP BY ROLLUP( department_id,(job_id, manager_id));
```

DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)
10	AD_ASST	101	4400
10			4400
20	MK_MAN	100	13000
20	MK_REP	201	6000
20			19000
50	ST_CLERK	124	11700
...			
			175500

23 rows selected.

1

2

3

Concatenated Groupings

- Concatenated groupings offer a concise way to generate useful combinations of groupings.
- To specify concatenated grouping sets, you separate multiple grouping sets, `ROLLUP`, and `CUBE` operations with commas so that the Oracle Server combines them into a single `GROUP BY` clause.
- The result is a cross-product of groupings from each grouping set.

```
GROUP BY GROUPING SETS(a, b), GROUPING SETS(c, d)
```

Concatenated Groupings Example

```
SELECT    department_id, job_id, manager_id,
          SUM(salary)
FROM      employees
GROUP BY department_id,
          ROLLUP(job_id),
          CUBE(manager_id);
```

1
2
3
4

DEPARTMENT_ID	JOB_ID	MANAGER_ID	SUM(SALARY)
10	AD_ASST	101	4400
20	MK_MAN	100	13000
...			
10		101	4400
20		100	13000
...			
10	AD_ASST		4400
10			4400
...			
	SA_REP		7000
			7000

49 rows selected.

Summary

In this lesson, you should have learned how to:

- Use the `ROLLUP` operation to produce subtotal values
- Use the `CUBE` operation to produce cross-tabulation values
- Use the `GROUPING` function to identify the row values created by `ROLLUP` or `CUBE`
- Use the `GROUPING SETS` syntax to define multiple groupings in the same query
- Use the `GROUP BY` clause, to combine expressions in various ways:
  - Composite columns
  - Concatenated grouping sets

Practice 17 Overview

This practice covers the following topics:

- Using the `ROLLUP` operator
- Using the `CUBE` operator
- Using the `GROUPING` function
- Using `GROUPING SETS`