Generating Reports by Grouping Related Data

Objectives

After completing this appendix, you should be able to use the:

- ROLLUP operation to produce subtotal values
- CUBE operation to produce cross-tabulation values
- GROUPING function to identify the row values created by ROLLUP or CUBE
- 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:

Review of the HAVING Clause

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

```
SELECT [column,] group_function(column)...

FROM table
[WHERE condition]
[GROUP BY group_by_expression]
[HAVING having_expression]
[ORDER BY column];
```

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 result set containing the regular grouped rows and the subtotal values.
- CUBE grouping produces a result set containing the rows from ROLLUP and cross-tabulation rows.

ROLLUP Operator

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

```
SELECT [column,] group_function(column)...

FROM table
[WHERE condition]
[GROUP BY [ROLLUP] group_by_expression]
[HAVING having_expression];
[ORDER BY column];
```

ROLLUP Operator: Example

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

| | 2 | DEPARTMENT | T_ID | ₿ JOB_ID | £ | SUM(SALARY) | |
|----|---|------------|-------|----------|---|-------------|-------|
| 1 | | | 10 | AD_ASST | | 4400 | - 1 |
| 2 | | | 10 | (null) | | 4400 | , |
| 3 | | | 20 | MK_MAN | | 13000 | |
| 4 | | | 20 | MK_REP | | 6000 | |
| 5 | | | 20 | (null) | | 19000 | |
| 6 | | | 30 | PU_MAN | | 11000 | |
| 7 | | | 30 | PU_CLERK | | 13900 | |
| 8 | | | 30 | (null) | | 24900 | |
| 9 | | | 40 | HR_REP | | 6500 | |
| 10 | | | 40 | (null) | | 6500 | |
| 11 | | | 50 | ST_MAN | | 36400 | |
| 12 | | | 50 | SH_CLERK | | 64300 | |
| 13 | | | 50 | ST_CLERK | | 55700 | |
| 14 | | | 50 | (null) | | 156400 | . 1 |
| 15 | | (| null) | (null) | | 211200 | ' |







CUBE Operator

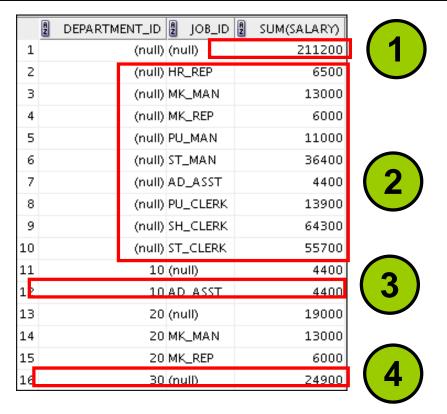
- CUBE is an extension to the GROUP BY clause.
- You can use the CUBE operator to produce crosstabulation values with a single SELECT statement.

```
SELECT [column,] group_function(column)...

FROM table
[WHERE condition]
[GROUP BY [CUBE] group_by_expression]
[HAVING having_expression]
[ORDER BY column];
```

CUBE Operator: Example

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



GROUPING Function

The GROUPING function:

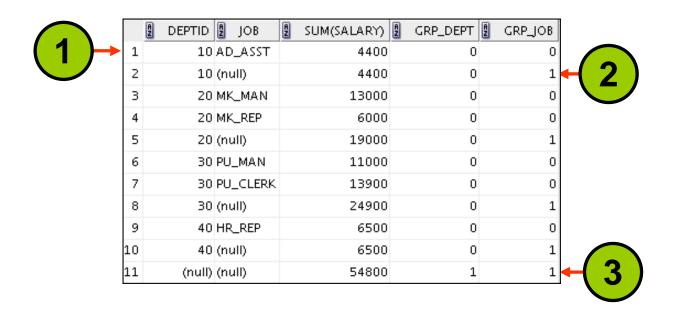
- Is used with either the CUBE or ROLLUP operator
- Is used to find the groups forming the subtotal in a row
- Is used to differentiate stored NULL values from NULL values created by ROLLUP or CUBE
- Returns 0 or 1

```
SELECT [column,] group_function(column) .. ,

GROUPING(expr)

FROM table
[WHERE condition]
[GROUP BY [ROLLUP] [CUBE] group_by_expression]
[HAVING having_expression]
[ORDER BY column];
```

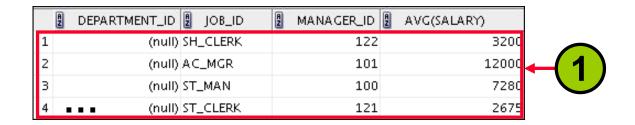
GROUPING Function: Example

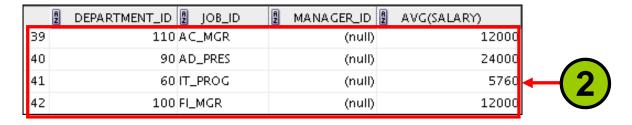


GROUPING SETS

- The GROUPING SETS syntax is used to define multiple groupings in the same query.
- All groupings specified in the GROUPING SETS clause are computed and the results of individual groupings are combined 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 GROUPING SETS has, the greater is the performance benefit.

GROUPING SETS: Example





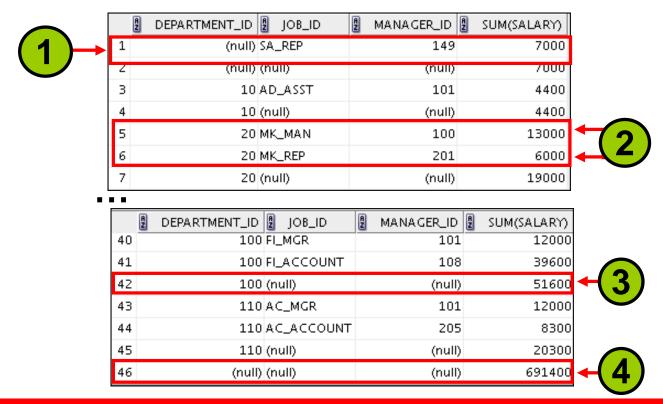
- - -

Composite Columns

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

- Use parentheses within the GROUP BY clause to group columns, so that they are treated as a unit while computing ROLLUP or CUBE operations.
- When used with ROLLUP or CUBE, composite columns would require skipping aggregation across certain levels.

Composite Columns: Example

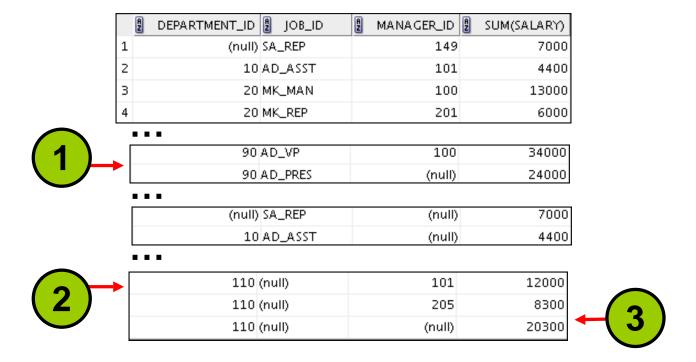


Concatenated Groupings

- 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



Summary

In this appendix, you should have learned how to use the:

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