**PRACTICAL NO. 3**

**Aim:** Write and Execute SQL aggregation queries for data warehouse.

**Details:** To run queries for CUBE, PARTIAL CUBE, ROLLUP, PARTIAL ROLLUP, GROUPING, GROUPING SETS

**Theory:**

Aggregation is a fundamental part of data warehousing. To improve aggregation performance in your warehouse, Oracle Database provides the following functionality:

* **CUBE** and **ROLLUP** extensions to the **GROUP** **BY** clause
* Three **GROUPING** functions
* **GROUPING** **SETS** expression
* Pivoting operations

The **CUBE**, **ROLLUP**, and **GROUPING** **SETS** extensions to SQL make querying and reporting easier and faster. **CUBE**, **ROLLUP**, and grouping sets produce a single result set that is equivalent to a **UNION** **ALL** of differently grouped rows. **ROLLUP** calculates aggregations such as **SUM**, **COUNT**, **MAX**, **MIN**, and **AVG** at increasing levels of aggregation, from the most detailed up to a grand total. **CUBE** is an extension similar to **ROLLUP**, enabling a single statement to calculate all possible combinations of aggregations. The **CUBE**, **ROLLUP**, and the **GROUPING** **SETS** extension lets you specify just the groupings needed in the **GROUP** **BY** clause. This allows efficient analysis across multiple dimensions without performing a **CUBE** operation. Computing a **CUBE** creates a heavy processing load, so replacing cubes with grouping sets can significantly increase performance.

To enhance performance, **CUBE**, **ROLLUP**, and **GROUPING** **SETS** can be parallelized: multiple processes can simultaneously execute all of these statements. These capabilities make aggregate calculations more efficient, thereby enhancing database performance, and scalability.

The three **GROUPING** functions help you identify the group each row belongs to and enable sorting subtotal rows and filtering results.

**ROLLUP Extension to GROUP BY**

ROLLUP enables a SELECT statement to calculate multiple levels of subtotals across a specified group of dimensions. It also calculates a grand total. ROLLUP is a simple extension to the GROUP BY clause, so its syntax is extremely easy to use. The ROLLUP extension is highly efficient, adding minimal overhead to a query.

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.

ROLLUP creates subtotals at n+1 levels, where n is the number of grouping columns. For instance, if a query specifies ROLLUP on grouping columns oftime, region, and department (n=3), the result set will include rows at four aggregation levels.

You might want to compress your data when using ROLLUP. This is particularly useful when there are few updates to older partitions.

When to Use ROLLUP

Use the **ROLLUP extension in tasks involving subtotals.**

* It is very helpful for subtotaling along a hierarchical dimension such as time or geography. For instance, a query could specify a ROLLUP(y, m, day)or ROLLUP(country, state, city).
* For data warehouse administrators using summary tables, ROLLUP can simplify and speed up the maintenance of summary tables.

ROLLUP Syntax

ROLLUP appears in the GROUP BY clause in a SELECT statement. Its form is:

SELECT … GROUP BY ROLLUP (grouping\_column\_reference\_list)

### PARTIAL ROLLUP

You can also roll up so that only some of the sub-totals will be included. This partial rollup uses the following syntax:

GROUP BY expr1, ROLLUP (expr2, expr3);

In this case, the GROUP BY clause creates subtotals at (2+1=3) aggregation levels. That is, at level (expr1, expr2, expr3), (expr1, expr2), and (expr1).

## CUBE Extension to GROUP BY

CUBE takes a specified set of grouping columns and creates subtotals for all of their possible combinations. In terms of multidimensional analysis, CUBE generates all the subtotals that could be calculated for a data cube with the specified dimensions. If you have specified CUBE (time, region,department), the result set will include all the values that would be included in an equivalent ROLLUP statement plus additional combinations. For instance, in [Figure 21-1](http://docs.oracle.com/cd/E11882_01/server.112/e25554/aggreg.htm#i1012453), the departmental totals across regions (279,000 and 319,000) would not be calculated by a ROLLUP(time, region,department) clause, but they would be calculated by a CUBE(time, region, department) clause. If n columns are specified for a CUBE, there will be 2 to the n combinations of subtotals returned. [Example 21-4](http://docs.oracle.com/cd/E11882_01/server.112/e25554/aggreg.htm#i1011609) gives an example of a three-dimension cube.

### When to Use CUBE

Consider Using CUBE in any situation requiring cross-tabular reports. The data needed for cross-tabular reports can be generated with a single SELECT using CUBE. Like ROLLUP, CUBE can be helpful in generating summary tables. Note that population of summary tables is even faster if the CUBE query executes in parallel.

CUBE is typically most suitable in queries that use columns from multiple dimensions rather than columns representing different levels of a single dimension. For instance, a commonly requested cross-tabulation might need subtotals for all the combinations of month, state, and product. These are three independent dimensions, 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 dimension. 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?" See ["Hierarchy Handling in ROLLUP and CUBE"](http://docs.oracle.com/cd/E11882_01/server.112/e25554/aggreg.htm#i1007192) for an example of handling rollup calculations efficiently.

### CUBE Syntax

CUBE appears in the GROUP BY clause in a SELECT statement. Its form is:

SELECT … GROUP BY CUBE (grouping\_column\_reference\_list)

### Partial CUBE

Partial CUBE resembles partial ROLLUP in that you can limit it to certain dimensions and precede it with columns outside the CUBE operator. In this case, subtotals of all possible combinations are limited to the dimensions within the cube list (in parentheses), and they are combined with the preceding items in the GROUP BY list.

The syntax for partial CUBE is as follows:

GROUP BY expr1, CUBE(expr2, expr3)

This syntax example calculates 2\*2, or 4, subtotals. That is:

* (expr1, expr2, expr3)
* (expr1, expr2)
* (expr1, expr3)
* (expr1)

## GROUPING Functions

Two challenges arise with the use of ROLLUP and CUBE. First, how can you programmatically determine which result set rows are subtotals, and how do you find the exact level of aggregation for a given subtotal? You often need to use subtotals in calculations such as percent-of-totals, so you need an easy way to determine which rows are the subtotals. Second, what happens if query results contain both stored NULL values and "NULL" values created by aROLLUP or CUBE? How can you differentiate between the two? This section discusses some of these situations.

### GROUPING Function

GROUPING handles these problems. Using a single column as its argument, GROUPING returns 1 when it encounters a NULL value created by a ROLLUPor CUBE operation. That is, if the NULL indicates the row is a subtotal, GROUPING returns a 1. Any other type of value, including a stored NULL, returns a 0.

GROUPING appears in the selection list portion of a SELECT statement. Its form is:

SELECT … [GROUPING(dimension\_column)…] …

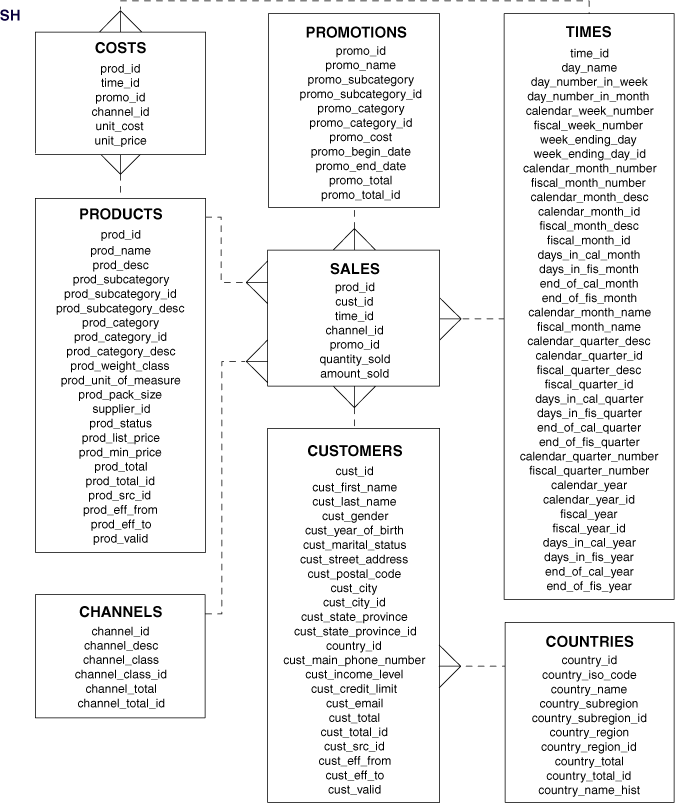
GROUP BY … {CUBE | ROLLUP| GROUPING SETS} (dimension\_column)

## GROUPING SETS Expression

You can selectively specify the set of groups that you want to create using a GROUPING SETS expression within a GROUP BY clause. This allows precise specification across multiple dimensions without computing the whole CUBE.

**DATA WAREHOUSE SCHEMA USED: SH**

The hypothetical company has sales across the world and tracks sales by both dollars and quantities information. SH is sales history.



Write the queries for the following:

Q1. Find the total sales by **country\_id** and **channel\_desc** for the US and GB through the Internet and direct sales in September 2000 and October 2000 using **ROLL-UP Extension**. The query should return the following:

* The aggregation rows that would be produced by GROUP BY ,
* The First-level subtotals aggregating across country\_iso\_code for each combination of channel\_desc and calendar\_month.
* Second-level subtotals aggregating across calendar\_month\_desc and country\_iso\_code for each channel\_desc value.
* A grand total row.

**Query:**

SELECT channels.channel\_desc, calendar\_month\_desc,

countries.country\_iso\_code,

TO\_CHAR(SUM(amount\_sold), '9,999,999,999') SALES$

FROM sales, customers, times, channels, countries

WHERE sales.time\_id=times.time\_id

AND sales.cust\_id=customers.cust\_id

AND customers.country\_id = countries.country\_id

AND sales.channel\_id = channels.channel\_id

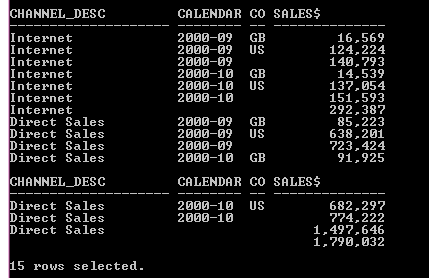
AND channels.channel\_desc IN ('Direct Sales', 'Internet')

AND times.calendar\_month\_desc IN ('2000-09', '2000-10')

AND countries.country\_iso\_code IN ('GB', 'US')

GROUP BY ROLLUP(channels.channel\_desc, calendar\_month\_desc, countries.country\_iso\_code);

Result:



Q2. Find the total sales by **country\_iso\_code** and **channel\_desc** for the US and GB through the Internet and direct sales in September 2000 and October 2009 using **CUBE** aggregation across three dimensions- channel\_desc, calendar\_month\_desc, countries. country\_iso\_code.

Q3. Find the total sales by **country\_iso** and **channel\_desc** for the US and France through the Internet and direct sales in September 2000

Q4. Find the total sales by **country\_iso** and **channel\_desc** for the US and GB through the Internet and direct sales in September 2000 and October 2009 using **PARTIAL ROLL-UP**. The query should return the following:

* Regular aggregation rows that would be produced by GROUP BY without using ROLLUP.
* First-level subtotals aggregating across country\_iso for each combination of channel\_desc and calendar\_month\_desc.
* Second-level subtotals aggregating across calendar\_month\_desc and country\_iso for each channel\_desc value.
* It does not produce a grand total row.

Q5. Find the total sales by **country\_id** and **channel\_desc** for the US and GB through the Internet and direct sales in September 2000 and October 2009 using **PARTIAL CUBE aggregation** on month and country code and GROUP BY on channel\_desc.

Q6. Use GROUPING to create a set of mask columns for the result set of Q1.

* Create grouping on channel\_desc and name it as CH
* Create grouping calendar\_month\_desc and name it as MO
* Create grouping on country\_iso\_code and name it as CO

**Query:**

SELECT channel\_desc, calendar\_month\_desc, country\_iso\_code,

TO\_CHAR(SUM(amount\_sold), '9,999,999,999') SALES$, GROUPING(channel\_desc) AS Ch,

GROUPING(calendar\_month\_desc) AS Mo, GROUPING(country\_iso\_code) AS Co

FROM sales, customers, times, channels, countries

WHERE sales.time\_id=times.time\_id

AND sales.cust\_id=customers.cust\_id

AND customers.country\_id = countries.country\_id

AND sales.channel\_id= channels.channel\_id

AND channels.channel\_desc IN ('Direct Sales', 'Internet')

AND times.calendar\_month\_desc IN ('2000-09', '2000-10')

AND countries.country\_iso\_code IN ('GB', 'US')

GROUP BY ROLLUP(channel\_desc, calendar\_month\_desc, countries.country\_iso\_code);

Q7.  Find the total sales by **country\_id** and **channel\_desc** for the US and GB through the Internet and direct sales in September 2000 and October 2009 using GROUPING SETS.

Calculate aggregates over three groupings:

* (channel\_desc, calendar\_month\_desc, country\_iso\_code)
* (channel\_desc, country\_iso\_code)
* (calendar\_month\_desc, country\_iso\_code)

Q8. Consider the following Query and make conclusion from the result obtained.

Query: (scott Schema)

SELECT deptno, job, SUM(sal)  
FROM emp  
GROUP BY CUBE(deptno, job)

Q9. Calculate the salary for each department present in different cities of hr schema using rollup.

Q10. Calculate the salary for each department present in different cities of hr schema using cube.