Objectives

Become aware of Oracle8i Release 2 (8.1.6) Analytic Functions at a high level

Learn about the Cube and Rollup enhancements to GROUP BY

Know how to use the Cube, Rollup to enhance systems

Oracle8i Version 2 (8.1.6) Analytic Functions

- Oracle 8.1.6 includes a new set of functions designed to provide expanded support for data mining operations -
  (this topic is too rich to fully cover in the context of this paper)
- The analytic functions are divided into four "families"
- Lag/Lead Compares values of rows to other rows in same table: LAG, LEAD
- Ranking Supports "top n" queries: CUME_DIST, DENSE_RANK, NTILE, PERCENT_RANK, RANK, ROW_NUMBER
- Reporting Aggregate Compares aggregates to non-aggregates (pct of total):
  RATIO_TO_REPORT
- Window Aggregate Moving average type queries: FIRST_VALUE, LAST_VALUE
- The analytic functions allow users to divide query result sets into ordered groups of rows called partitions
  (not the same as database partitions)

Oracle8i Version 2 (8.1.6) Analytic Function Clauses

- Along with the new functions came new clauses (again, too rich to cover completely here):

analytic_function ( ) OVER (analytic clause)

  - Analytic clause
    Query_partition_clause-Order_by clause-Windowing clause
  - Query partition clause
    PARTITION BY list,of,cols
  - Windowing clause
    RANGE … or **ROWS …**
  - Order by clause
    ORDER BY col,list

Analyzing across Multiple Dimensions

- One of the key concepts in decision support systems is "multi-dimensional analysis": examining the enterprise from all necessary combinations of dimensions.

- We use the term "dimension" to mean any category used in specifying questions. Among the most commonly specified dimensions are time, geography, product, department, and distribution channel, but the potential dimensions are as endless as the varieties of enterprise activity.

- The events or entities associated with a particular set of dimension values are usually referred to as "facts." The facts may be sales in units or local currency, profits, customer counts, production volumes, or anything else worth tracking.

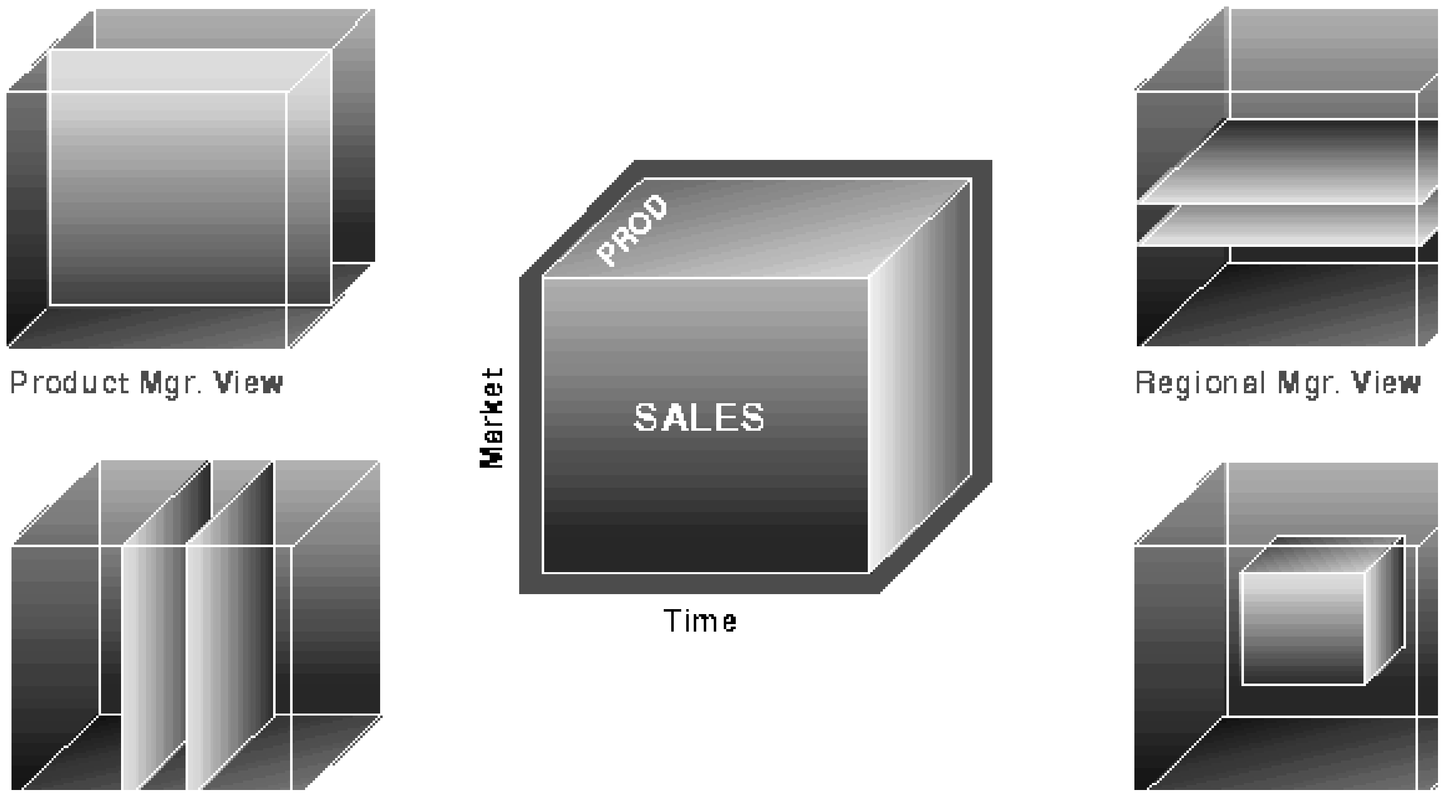
Here are some examples of multi-dimensional requests:

- Show total sales across all products at increasing aggregation levels: from state to country to region for 1996 and 1997.
- Create a cross-tabular analysis of our operations showing expenses by territory in South America for 1996 and 1997. Include all possible subtotals.
- List the top 10 sales representatives in Asia according to 1997 sales revenue in for automotive products and rank their commissions.

All the requests above constrain multiple dimensions. Many multi-dimensional questions require aggregated data and comparisons of data sets, often across time, geography or budgets.

Cube and Views by Different Users

- To visualize data that has many dimensions, analysts commonly use the analogy of a data "cube," that is, a space where facts are stored at the intersection of n dimensions.
- Figure 20-1 shows a data cube and how it could be used differently by various groups.
- The cube stores sales data organized by the dimensions of Product, Market, and Time.



Optimized Performance

Not only multi-dimensional issues, but all types of processing can benefit from enhanced aggregation facilities.

Transaction processing, financial and manufacturing systems--all of these generate large numbers of production reports needing substantial system resources.

Improved efficiency when creating these reports will reduce system load.

In fact, any computer process that aggregates data from details to higher levels needs optimized performance.

To leverage the power of the database server, powerful aggregation commands should be available inside the SQL engine.

New extensions in Oracle provide these features and bring many benefits, including:

- Simplified programming requiring less SQL code for many tasks
- Quicker and more efficient query processing
- Reduced client processing loads and network traffic because aggregation work is shifted to servers
- Opportunities for caching aggregations because similar queries can leverage existing work

Oracle8i provides all these benefits with the new **CUBE** and **ROLLUP** extensions to the **GROUP BY** clause.
These extensions adhere to the ANSI and ISO proposals for SQL3, a draft standard for enhancements to SQL.

A Scenario

To illustrate **CUBE**, **ROLLUP**, and **Top-N queries**, this chapter uses a hypothetical videotape sales and rental company.

All the examples given refer to data from this scenario

The hypothetical company has stores in several regions and tracks sales and profit information.

The data is categorized by three dimensions: **Time**, **Department**, and **Region**.

The time dimensions are 1996 and 1997, the departments are Video Sales and Video Rentals, and the regions are East, West, and Central.

Table 20-1 Simple Cross-Tabular Report, with Subtotals Shaded

A Scenario

Table 20-1 Simple Cross-Tabular Report, with Subtotals Shaded

		1997	
Region		Department	
	Video Rental Profit	Video Sales Profit	Total Profit
Central	82,000	85,000	167,000
East	101,000	137,000	238,000
West	96,000	97,000	193,000
Total	279,000	319,000	598,000

Consider that even a simple report like Table 20-1, with just twelve values in its grid, needs five subtotals and a grand total.

The subtotals are the shaded numbers, such as Video Rental Profits across regions, namely, 279,000, and Eastern region profits across department,
namely, 238,000.

Half of the values needed for this report would not be calculated with a query that used a standard SUM() AND GROUPBY()

Database commands that offer improved calculation of subtotals bring major benefits to querying, reporting and analytical operations.

ROLLUP

- 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.

Syntax

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

SELECT … GROUP BY ROLLUP(grouping_column_reference_list)

Details

- **ROLLUP's** action is straightforward: it creates subtotals which "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 will create subtotals at n+1 levels, where n is the number of grouping columns.
- For instance, if a query specifies ROLLUP on grouping columns of Time, Region, and Department (n=3), the result set will include rows at four aggregation levels.

snowflake

Example

This example of ROLLUP uses the data in the video store database.

```
SELECT Time, Region, Department, sum(Profit) AS Profit
FROM sales
GROUP BY ROLLUP(Time, Region, Dept)
```

As you can see in Table 20-2, this query returns the following sets of rows:

- Regular aggregation rows that would be produced by GROUP BY without using ROLLUP
- First-level subtotals aggregating across Department for each combination of Time and Region
- Second-level subtotals aggregating across Region and Department for each Time value
- A grand total row

Example

Time	Region	Department	Profit
1996	Central	VideoRental	75,000
1996	Central	VideoSales	74,000
1996	Central	[NULL]	149,000
1996	East	VideoRental	89,000
1996	East	VideoSales	115,000
1996	East	[NULL]	204,000
1996	West	VideoRental	87,000
1996	West	VideoSales	86,000
1996	West	[NULL]	173,000
1996	[NULL]	[NULL]	526,000
1997	Central	VideoRental	82,000
1997	Central	VideoSales	85,000
1997	Central	[NULL]	167,000
1997	East	VideoRental	101,000
1997	East	VideoSales	137,000
1997	East	[NULL]	238,000
1997	West	VideoRental	96,000
1997	West	VideoSales	97,000
1997	West	[NULL]	193,000
1997	[NULL]	[NULL]	598,000
[NULL]	[NULL]	[NULL]	1,124,000

Interpreting "[NULL]" Values in Results

The NULL values returned by ROLLUP and CUBE are not always the traditional NULL value meaning "value unknown."

Instead, a NULL may indicate that its row is a subtotal.

For instance, the first NULL value shown in Table 20-2 is in the Department column.
This NULL means that the row is a subtotal for "All Departments" for the Central region in 1996.

To avoid introducing another non-value in the database system, these subtotal values are not given a special tag.

Note: The NULLs shown in the figures of this paper are displayed only for clarity: in standard Oracle output these cells would be blank.

Calculating Subtotals without ROLLUP

The result set in Table 20-1 could be generated by the UNION of four SELECT statements, as shown below.

This is a subtotal across three dimensions.

Notice that a complete set of ROLLUP-style subtotals in n dimensions would require n+1 SELECT statements linked with UNION ALL.

```
SELECT Time, Region, Department, SUM(Profit)
 FROM Sales
 GROUP BY Time, Region, Department
UNION  ALL
 SELECT Time, Region, '' , SUM(Profit)
 FROM Sales
 GROUP BY Time, Region
UNION ALL
 SELECT Time, '', '', SUM(Profits)
 FROM Sales
 GROUP BY Time
UNION ALL
 SELECT '', '', '', SUM(Profits)
 FROM Sales;
```

Calculating Subtotals without ROLLUP

- The approach shown in the SQL above has two shortcomings compared to using the ROLLUP operator.
- First, the syntax is complex, requiring more effort to generate and understand.
- Second, and more importantly, query execution is inefficient because the optimizer receives no guidance about the user's overall goal.
- Each of the four SELECT statements above causes table access even though all the needed subtotals could be gathered with a single pass.
- The ROLLUP extension makes the desired result explicit and gathers its results with just one table access.
- The more columns used in a ROLLUP clause, the greater the savings versus the UNION approach.
- For instance, if a four-column ROLLUP replaces a UNION of 5 SELECT statements, the reduction in table access is four-fifths or 80%.
- Some data access tools calculate subtotals on the client side and thereby avoid the multiple SELECT statements described above.
- While this approach can work, it places significant loads on the computing environment.
- For large reports, the client must have substantial memory and processing power to handle the subtotaling tasks.
- Even if the client has the necessary resources, a heavy processing burden for subtotal calculations may slow down the client in its performance of other activities.

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 of year/month/day or country/state/city.

•It simplifies and speeds the population and maintenance of summary tables.

•Data warehouse administrators may want to make extensive use of it.

•Note that population of summary tables is even faster if the ROLLUP query executes in parallel.

CUBE

Note that the subtotals created by ROLLUP are only a fraction of possible subtotal combinations.

For instance, in the cross-tab shown in Table 20-1, the departmental totals across regions (279,000 and 319,000) would not be calculated by a ROLLUP(Time, Region, Department) clause.

To generate those numbers would require a **ROLLUP** clause with the grouping columns specified in a different order:

ROLLUP(Time, Department, Region). The easiest way to generate the full set of subtotals needed for cross-tabular reports such as those needed for Figure 20-1 is to use the CUBE extension.

snowflake

ANALYTICS WITH ANAND

CUBE

CUBE enables a SELECT statement to calculate subtotals for all possible combinations of a group of dimensions.

It also calculates a grand total.

This is the set of information typically needed for all cross-tabular reports, so CUBE can calculate a cross-tabular report with a single SELECT statement.

Like ROLLUP, CUBE is a simple extension to the GROUP BY clause, and its syntax is also easy to learn.

Syntax

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

```
SELECT ...  GROUP BY
  CUBE (grouping_column_reference_list)
```

Details

CUBE takes a specified set of grouping columns and creates subtotals for all possible combinations of them.
In terms of multi-dimensional 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 Table 20-1, 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 there are n columns specified for a CUBE, there will be 2n combinations of subtotals returned.

Table 20-3 gives an example of a three-dimension CUBE.

Example

This example of CUBE uses the data in the video store database.

```
SELECT Time, Region, Department, sum(Profit) AS Profit
FROM sales
GROUP BY CUBE (Time, Region, Dept)
```

Table 20-3 shows the results of this query.

Table 20-3 Cube Aggregation across Three Dimensions

Time	Region	Department	Profit
1996	Central	VideoRental	75,000
1996	Central	VideoSales	74,000
1996	Central	[NULL]	149,000
1996	East	VideoRental	89,000
1996	East	VideoSales	115,000
1996	East	[NULL]	204,000
1996	West	VideoRental	87,000
1996	West	VideoSales	86,000
1996	West	[NULL]	173,000

Example

Time	Region	Department	Profit
1996	[NULL]	VideoRental	251,000
1996	[NULL]	VideoSales	275,000
1996	[NULL]	[NULL]	526,000
1997	Central	VideoRental	82,000
1997	Central	VideoSales	85,000
1997	Central	[NULL]	167,000
1997	East	VideoRental	101,000
1997	East	VideoSales	137,000
1997	East	[NULL]	238,000
1997	West	VideoRental	96,000
1997	West	VideoSales	97,000
1997	West	[NULL]	193,000
1997	[NULL]	VideoRental	279,000
1997	[NULL]	VideoSales	319,000
1997	[NULL]	[NULL]	598,000
[NULL]	Central	VideoRental	157,000
[NULL]	Central	VideoSales	159,000
[NULL]	Central	[NULL]	316,000
[NULL]	East	VideoRental	190,000
[NULL]	East	VideoSales	252,000
[NULL]	East	[NULL]	442,000
[NULL]	West	VideoRental	183,000
[NULL]	West	VideoSales	183,000
[NULL]	West	[NULL]	366,000
[NULL]	[NULL]	VideoRental	530,000
[NULL]	[NULL]	VideoSales	594,000
[NULL]	[NULL]	[NULL]	1,124,000

snowflake

Calculating subtotals without CUBE

Just as for ROLLUP, multiple SELECT statements combined with UNION statements could provide the same information gathered through CUBE.

However, this may require many SELECT statements: for an n-dimensional cube, 2n SELECT statements are needed.

In our 3-dimension example, this would mean issuing 8 SELECTS linked with UNION ALL.

Consider the impact of adding just one more dimension when calculating all possible combinations: the number of SELECT statements would double to 16.

The more columns used in a CUBE clause, the greater the savings versus the UNION approach.

For instance, if a four-column CUBE replaces a UNION of 16 SELECT statements, the reduction in table access is fifteen-sixteenths or 93.75%.

When to Use CUBE

Use 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 especially valuable 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/product.

•These are three independent dimensions, and analysis of all possible subtotal combinations will be commonplace.

•In contrast, a cross-tabulation showing all possible combinations of year/month/day would have several values of limited interest, since 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.

Using Other Aggregate Functions with ROLLUP and CUBE

The examples in this chapter show ROLLUP and CUBE used with the SUM() operator.

While this is the most common type of aggregation, the extensions can also be used with all the other functions available to Group by clauses, for example, COUNT, AVG, MIN, MAX, STDDEV, and VARIANCE. COUNT, which is often needed in cross-tabular analyses, is likely be the second most helpful function.

Note:

The **DISTINCT** qualifier has ambiguous semantics when combined with ROLLUP and CUBE.

To minimize confusion and opportunities for error, **DISTINCT** is not permitted together with the extensions.

GROUPING Function

Two challenges arise with the use of ROLLUP and CUBE.

First, how can we programmatically determine which result set rows are subtotals, and how do we find the exact level of aggregation of a given subtotal?

We will often need to use subtotals in calculations such as percent-of-totals, so we need an easy way to determine which rows are the subtotals we seek.

Second, what happens if query results contain both stored NULL values and "NULL" values created by a ROLLUP or CUBE?

How does an application or developer differentiate between the two?

To handle these issues, Oracle 8i introduces a new function called GROUPING.

Using a single column as its argument, Grouping returns 1 when it encounters a NULL value created by a ROLLUP or 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, will return a 0.

GROUPING Function

Syntax

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

```
SELECT ...  [GROUPING(dimension_column)...]  ...
  GROUP BY ...    {CUBE | ROLLUP}
```

Examples

This example uses grouping to create a set of mask columns for the result set shown in Table 20-3. The mask columns are easy to analyze programmatically.

```
SELECT Time, Region, Department, SUM(Profit) AS Profit,
  GROUPING (Time) as T,
  GROUPING (Region) as R,
  GROUPING (Department) as D
  FROM Sales
GROUP BY ROLLUP (Time, Region, Department)
```

Table 20-4 Use of Grouping Function

Time	Region	Department	Profit	T	R	D
1996	Central	Video Rental	75,000	0	0	0
1996	Central	Video Sales	74,000	0	0	0
1996	Central	[NULL]	149,000	0	0	1
1996	East	Video Rental	89,000	0	0	0
1996	East	Video Sales	115,000	0	0	0
1996	East	[NULL]	204,000	0	0	1
1996	West	Video Rental	87,000	0	0	0
1996	West	Video Sales	86,000	0	0	0
1996	West	[NULL]	173,000	0	0	1
1996	[NULL]	[NULL]	526,000	0	1	1
1997	Central	Video Rental	82,000	0	0	0
1997	Central	Video Sales	85,000	0	0	0
1997	Central	[NULL]	167,000	0	0	1
1997	East	Video Rental	101,000	0	0	0
1997	East	Video Sales	137,000	0	0	0
1997	East	[NULL]	238,000	0	0	1
1997	West	VideoRental	96,000	0	0	0
1997	West	VideoSales	97,000	0	0	0
1997	West	[NULL]	193,000	0	0	1
1997	[NULL]	[NULL]	598,000	0	1	1
[NULL]	[NULL]	[NULL]	1,124,000	1	1	1

A program can easily identify the detail rows above by a mask of "0 0 0" on the T, R, and D columns.

The first level subtotal rows have a mask of "0 0 1", the second level subtotal rows have a mask of "0 1 1", and the overall total row have a mask of "1 1 1".

Table 20-5 shows an ambiguous result set created using the CUBE extension.

Table 20-5 Distinguishing Aggregate NULL from Stored NULL Value

Time	Region	Profit
1996	East	200,000
1996	[NULL]	200,000
[NULL]	East	200,000
[NULL]	[NULL]	190,000
[NULL]	[NULL]	190,000
[NULL]	[NULL]	190,000
[NULL]	[NULL]	390,000

In this case, four different rows show NULL for both Time and Region.

Some of those NULLs must represent aggregates due to the CUBE extension, and others must be NULLs stored in the database.

How can we tell which is which?

GROUPING functions, combined with the NVL and DECODE functions, resolve the ambiguity so that human readers can easily interpret the values.

AMBGUITY

We can resolve the ambiguity by using the GROUPING and other functions in the code below.

```
SELECT
  decode(grouping(Time), 1, 'All Times', Time) as Time,
  decode(grouping(region), 1, 'All Regions', 0, null)) as
  Region, sum(Profit) AS Profit from Sales
  group by CUBE(Time, Region)
```

To explain the SQL statement above, we will examine its first column specification, which handles the Time column.
Look at the first line of the in the SQL code above, namely,

decode(grouping(Time), 1, 'All Times', Time) as Time,

The Time value is determined with a DECODE function that contains a GROUPING function.
The GROUPING function returns a 1 if a row value is an aggregate created by ROLLUP or CUBE, otherwise it returns a 0.

The DECODE function then operates on the GROUPING function's results.
It returns the text "All Times" if it receives a 1 and the time value from the database if it receives a 0.

Values from the database will be either a real value such as 1996 or a stored NULL.
The second column specification, displaying Region, works the same way.

When to Use GROUPING

The GROUPING function is not only useful for identifying NULLs, it also enables sorting subtotal rows and filtering results.

In the example below (Table 20-7), we retrieve a subset of the subtotals created by a CUBE and none of the base-level aggregations.

The HAVING clause constrains columns which use GROUPING functions.

```
SELECT Time, Region, Department, SUM(Profit) AS Profit,
  GROUPING (Time) AS T,
  GROUPING (Region) AS R,
  GROUPING (Department) AS D
  FROM Sales
  GROUP BY CUBE (Time, Region, Department)
  HAVING (D=1 AND R=1 AND T=1)
  OR (R=1 AND D=1)
  OR (T=1 AND D=1)
```

When to Use GROUPING

Table 20-7 shows the results of this query.

Table 20-7 Example of GROUPING Function Used to Filter Results to Subtotals and Grand Total

Time	Region	Department	Profit
1996	[NULL]	[NULL]	526,000
1997	[NULL]	[NULL]	598,000
[NULL]	Central	[NULL]	316,000
[NULL]	East	[NULL]	442,000
[NULL]	West	[NULL]	366,000
[NULL]	[NULL]	[NULL]	1,124,000

Compare the result set of Table 20-7 with that in Table 20-3 to see how Table 20-7 is a precisely specified group: it contains only the yearly totals, regional totals aggregated over time and department, and the grand total.

Hierarchy Handling in ROLLUP and CUBE

The ROLLUP and CUBE extensions work independently of any hierarchy metadata in your system.

Their calculations are based entirely on the columns specified in the SELECT statement in which they appear.

This approach enables CUBE and ROLLUP to be used whether or not hierarchy metadata is available.

The simplest way to handle levels in hierarchical dimensions is by using the ROLLUP extension and indicating levels explicitly through separate columns.

The code below shows a simple example of this with months rolled up to quarters and quarters rolled up to years.

```
SELECT Year, Quarter, Month,
    SUM(Profit) AS Profit FROM sales
    GROUP BY ROLLUP(Year, Quarter, Month)
```

Hierarchy Handling in ROLLUP and CUBE

Table 20-8 Example of ROLLUP across Time Levels

Year	Quarter	Month	Profit
1997	Winter	Jan	55,000
1997	Winter	Feb	64,000
1997	Winter	March	71,000
1997	Winter	[NULL]	190,000
1997	Spring	April	75,000
1997	Spring	May	86,000
1997	Spring	June	88,000
1997	Spring	[NULL]	249,000
1997	Summer	July	91,000
1997	Summer	August	87,000
1997	Summer	September	101,000
1997	Summer	[NULL]	279,000
1997	Fall	October	109,000
1997	Fall	November	114,000
1997	Fall	December	133,000
1997	Fall	[NULL]	356,000
1997	[NULL]	[NULL]	1,074,000

Column Capacity in ROLLUP and CUBE

CUBE and ROLLUP do not restrict the GROUP BY clause column capacity.

The GROUP BY clause, with or without the extensions, can work with up to 255 columns.

However, the combinatorial explosion of CUBE makes it unwise to specify a large number of columns with the CUBE extension.

Consider that a 20-column list for CUBE would create 2^{20} combinations in the result set.

A very large CUBE list could strain system resources, so any such query needs to be tested carefully for performance and the load it places on the system.

HAVING Clause Used with ROLLUP and CUBE

The HAVING clause of SELECT statements is unaffected by the use of ROLLUP and CUBE. Note that the conditions specified in the HAVING clause apply to both the subtotal and non-subtotal rows of the result set.

In some cases a query may need to exclude the subtotal rows or the non-subtotal rows from the HAVING clause.

This can be achieved by using the GROUPING function together with the HAVING clause.

See Table 20-7 and its associated SQL for an example.

Overview of CUBE, ROLLUP, and Top-N Queries

The last decade has seen a tremendous increase in the use of query, reporting, and on-line analytical processing (OLAP) tools, often in conjunction with data warehouses and data marts.

Enterprises exploring new markets and facing greater competition expect these tools to provide the maximum possible decision-making value from their data resources.

Oracle expands its long-standing support for analytical applications in Oracle8i release 8.1.5 with the CUBE and ROLLUP extensions to SQL.

Oracle also provides optimized performance and simplified syntax for Top-N queries.

These enhancements make important calculations significantly easier and more efficient, enhancing database performance, scalability and simplicity.

Overview of CUBE, ROLLUP, and Top-N Queries

ROLLUP and **CUBE** are simple extensions to the SELECT statement's GROUP BY clause.

ROLLUP creates subtotals at any level of aggregation needed, from the most detailed up to a grand total.

CUBE is an extension similar to ROLLUP

CUBE can generate the information needed in cross-tab reports with a single query.

To enhance performance, both **CUBE** and **ROLLUP** are parallelized: multiple processes can simultaneously execute both types of statements.

Cube and Rollup

- CUBE and ROLLUP extend GROUP BY
- ROLLUP builds subtotal aggregates at any level, including grand total
- CUBE extends ROLLUP to calculate all possible combinations of subtotals for a GROUP BY
- Cross-tabulation reports are easy with CUBE
- Oracle8i Release 2 (Oracle version 8.1.6) began release in February 2000, it’s new “Analytic” functions include: ranking, moving aggregates, period comparisons, ratio of total, and cumulative aggregates

Normal GROUP BY Functionality

- Normally, GROUP BY allows aggregates (sub-totals) by specific column or set of columns
- Before Oracle8i SQL required JOIN or UNION to combine subtotal information and grand totals in a single SQL query
- ROLLUP creates subtotals and grand totals in the same query along with intermediate subtotals
- CUBE adds cross-tabulation information based upon the GROUP BY columns

GROUP BY (without CUBE or ROLLUP)

- Normally GROUP BY sorts on GROUP BY columns, then calculates aggregates

```
SQL> select deptno Department
  2           ,job
  3           ,sum(sal)  "Total SAL"
  4         from emp
  5         group by deptno,job
  6   /

DEPARTMENT JOB        Total SAL
---------- --------- ----------
        10 CLERK           1300
        10 MANAGER         2450
        10 PRESIDENT       5000
        20 ANALYST         6000
        20 CLERK           1900
        20 MANAGER         2975
        30 CLERK            950
        30 MANAGER         2850
        30 SALESMAN        5600
```

GROUP BY ROLLUP

- ROLLUP provides aggregates at each GROUP BY level

```
SQL> col Department format a20
SQL> break on Department
SQL> select nvl(to_char(deptno),'Whole Company') Department
  2              ,nvl(job,'All Employees') job
  3              ,sum(sal)  "Total SAL"
  4        from emp
  5      group by rollup (deptno,job)
  6   /
DEPARTMENT                   JOB                 Total SAL
---------------------------- ------------------- ------------
10                           CLERK                       1300
                             MANAGER                     2450
                             PRESIDENT                   5000
                             All Employees               8750
20                           ANALYST                     6000
                             CLERK                       1900
                             MANAGER                     2975
                             All Employees              10875
30                           CLERK                        950
                             MANAGER                     2850
                             SALESMAN                    5600
                             All Employees               9400
Whole Company                All Employees              29025
```

NULL Values in CUBE/ROLLUP Rows

- Subtotal and grand total lines generated by ROLLUP substitute NULL for column values not present in the manufactured output row
- The example uses the NVL function to replace NULLS
- Some columns might normally contain NULL values, thus, normally occurring NULLS would be grouped with rows manufactured by ROLLUP or CUBE

GROUPING Function

- To improve dealing with the NULL values present in the rows created by ROLLUP (and CUBE discussed later), Oracle provides the new GROUPING function
- GROUPING returns a value of 1 if a row is a subtotal created by ROLLUP or CUBE, and a 0 otherwise
- The following example shows the same query used previously, with DECODE used in conjunction with GROUPING to more-elegantly deal with the null values created by ROLLUP and CUBE

  (Note: sample data contains no null values, the results from this query and the previous query are the same).

GROUPING Example

```
SQL> col Department format a20
SQL> break on Department
SQL> select decode(grouping(deptno),1,'Whole Company'
  2             ,'Department ' || to_char(deptno)) Department
  3             ,decode(grouping(job),1,'All Employees',job) job
  4             ,sum(sal)  "Total SAL"
  5         from emp
  6         GROUP BY ROLLUP (deptno,job)
/
DEPARTMENT                 JOB              Total SAL

-------------------------- ---------------- ---------
Department 10              CLERK                 1300
                           MANAGER               2450
                           PRESIDENT             5000
                           All Employees         8750
Department 20              ANALYST               6000
                           CLERK                 1900
                           MANAGER               2975
                           All Employees        10875
Department 30              CLERK                  950
                           MANAGER               2850
                           SALESMAN              5600
                           All Employees         9400
Whole Company              All Employees        29025
```

GROUP BY CUBE

- CUBE automatically calculates all possible combinations of subtotals

```
SQL> select decode(grouping(deptno),1,'Whole Company','Department ' || to_char(deptno)) Department
  2            ,decode(grouping(job),1,'All Employees',job) job
  3            ,sum(sal)  "Total SAL"
  4        from emp GROUP BY CUBE(deptno,job)
```

DEPARTMENT	JOB	Total SAL
Department 10	CLERK	1300
	MANAGER	2450
	PRESIDENT	5000
	All Employees	8750
Department 20	ANALYST	6000
	CLERK	1900
	MANAGER	2975
	All Employees	10875
Department 30	CLERK	950
	MANAGER	2850
	All Employees	9400
Whole Company	ANALYST	6000
	CLERK	4150
	MANAGER	8275
	PRESIDENT	5000
	SALESMAN	5600
	All Employees	29025

GROUP BY/ROLLUP/CUBE

- CUBE add subtotals for all combinations of categories (total salary for each job type was added in the example)
- If there were three GROUP BY columns (i.e. country, customer_id, product):
  - GROUP BY would produce aggregates each unique combination of the three columns showing the aggregate for each product ordered by each customer within each country
  - ROLLUP would add aggregates showing the total products by country and customer_id, total products by country, and a grand total of all products sold
  - CUBE would add aggregates for each product regardless of country or customer id, aggregates for each customer_id regardless of country or products ordered, and aggregates of each product by country regardless of customer id

CUBE/ROLLUP & Analytic Functions (8.1.6)

- Combine Analytic Functions and Clauses with CUBE and ROLLUP

```
SQL> run
  1 select decode(grouping(deptno),1,'Whole  Company'
  2            ,'Department ' || to_char(deptno)) Department
  3            ,decode(grouping(job),1,'All  Employees',job)  job
  4            ,sum(sal)  "Total SAL"
  5            ,ROW_NUMBER() OVER (PARTITION BY deptno ORDER BY sum(sal)) rownbr
  6*        from emp where deptno in (10,20) group by rollup  (deptno,job)
```

DEPARTMENT	JOB	Total SAL	ROWNBR
Department 10	CLERK	1300	1
	MANAGER	2450	2
	PRESIDENT	5000	3
	All Employees	8750	4
Department 20	CLERK	1900	1
	MANAGER	2975	2
	ANALYST	6000	3
	All Employees	10875	4
Whole Company	All Employees	19625	1

Conclusion

- CUBE and ROLLUP reduce work necessary to code and create aggregates often requested by management to provide competitive or summary information
- CUBE and ROLLUP provide mechanisms for using a single SQL statement to provide data that would have required multiple SQL statements, programming, or manual summarization in the past


THANK YOU