CUBE, ROLLUP, AND MATERIALIZED VIEWS: MINING ORACLE GOLD

John Jay King, King Training Resources

Abstract:

Oracle8i provides new features that reduce the costs of summary queries and provide easier summarization. The CUBE and ROLLUP extensions to GROUP BY allow more-complete summaries to be created. CUBE and ROLLUP provide information that would otherwise require additional queries or coding. Taking advantage of Materialized Views allows organizations to pre-build tables of summary data used frequently. The results of common summaries in Materialized Views may be indexed helping speed typical queries. Materialized Views may be refreshed in the same manner used for Snapshots allowing summarization to occur with a frequency that meets the needs of the user community. By including CUBE and ROLLUP in a Materialized View along with other GROUP BY results, more complete information is available for reduced-cost queries. Those present will learn how to create and use Materialized Views and how to use CUBE and ROLLUP to create more complete summaries.

All sample code was tested using Oracle 8.1.5. Oracle8i Release 2 (Oracle version 8.1.6) is due for release within days of this paper's writing, without having an opportunity to the new release, new features are only mentioned here, unfortunately no examples are possible at this time. Some of the more significant features included in the new release are "Analytic" functions allowing ranking, moving aggregates, period comparisons, ratio of total, cumulative aggregates, addition of ORDER BY to materialized view definitions, and more.

CUBE and ROLLUP Extensions to GROUP BY

The business community frequently calls for statistical information from databases that is useful in decision making. The increased emphasis on data-mining activities includes a need for super-aggregate capabilities. Oracle8i provides CUBE and ROLLUP to extend the ability of GROUP BY to include some of the following features:

- ROLLUP builds subtotal aggregates at every level requested, including grand total.
- CUBE extends ROLLUP to calculate all possible combinations of subtotals for a specific GROUP BY.
- Data for cross-tabulation reports is created easily using CUBE.

Normally, the SQL-standard GROUP BY function allows aggregation (sub-totals) by some specific column or set of columns. Before Oracle8i SQL statements required JOIN or UNION to combine subtotal information (1 output row per group of detail rows) and grand totals (1 output row for the entire set of data) in a single SQL query. ROLLUP allows the creation of subtotal and grand total information in the same query along with intermediate-level subtotals at each level of aggregation. CUBE adds cross-tabulation information based upon the GROUP BY columns.

All examples in this article were created with the Oracle-provided SCOTT/TIGER sample tables. Execute the DEMOBLD procedure under your own userid to create personal copies of the tables. All of the examples should work as illustrated.

Some of the examples below use SQL*Plus COL and BREAK commands to make the output more like that typically supplied to management. Each example is shown as it might be coded into SQL*Plus. If you are using a tool other than SQL*Plus to access Oracle the formatting of COL and BREAK may be redundant.

GROUP BY (without CUBE or ROLLUP)

The following query shows the normal functionality of GROUP BY. SQL sorts the detail data using the GROUP BY columns, then calculates the subtotals and outputs a row for each group.

```
 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
```

This query provides the sum of salaries for groups of employees sorted by department and job. For instance, the sum of salaries for CLERKs in Department 10 is calculated separately from the sum for CLERKS in Department 20. Note that sum for all jobs in Department 10 is not shown, nor is the grand total of all salaries.

GROUP BY ROLLUP

ROLLUP provides aggregation at each level indicated by the GROUP BY columns. This is a level of information not possible previously without adding code or manual processes. The following statement shows the impact of ROLLUP:

```
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
```

Note that ROLLUP creates subtotals for each level of subtotal (Department subtotal), and a grand total.

For example: suppose an international sporting goods firm calculated sales totals using three GROUP BY columns (i.e. Country, Customer_ID, Product). GROUP BY would normally produce aggregates (subtotals) for each unique combination of the three columns showing the aggregate for each product ordered by each customer within each country.

```
Country     Customer_ID     Product           Sales
France      FR1234          Tennis Balls     34,000
```

ROLLUP adds aggregates showing the total products by Country and Customer_ID, total products by Country, and a grand total of all products sold.

```
Country     Customer_ID     Product           Sales
France      FR1234          Tennis Balls     34,000      <- produced by GROUP BY
France      FR1234                          100,000      <- Rollup produces total by country+customer
France                                      340,000      <- Rollup produces total by country
Total                                     1,000,345      <- Rollup produces Grand total (all records)
```

ROLLUP provides useful high-level summary information of the type often requested by management. Before ROLLUP, additional processing or manual work was required to provide this level of cumulative data.

The subtotal and grand total lines generated by ROLLUP substitute NULL for column values not present in the manufactured output row. The NVL function was used in the example above to replace NULL values. The problem with this technique is that it is possible for some columns to 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 example below shows the same query used previously, with the DECODE function used in conjunction with the GROUPING function 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).

```
SQL> col Department format a20
SQL> break on Department
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
  5      group by rollup (deptno,job)
  6  Input truncated to 1 characters
/
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
```

In the example above, the Department subtotals and grand total now have more-specific titles and values. Since ROLLUP generates a row summarizing salaries for all of the employees (regardless of job) in a department, the example uses “All Employees” to show that subtotal instead of a job title. When ROLLUP generates a summary of salaries for all of the departments (grand total), the department column shows “Whole Company” instead of a specific department number.

GROUP BY CUBE

In addition to the group subtotals and grand total created by ROLLUP, CUBE automatically calculates all possible combinations from the available subtotals. This provides a recap of summary information for each category of information listed. The example below shows the same query used previously, but, the CUBE function is used to provide additional summary information:

```
SQL> col Department format a20
SQL> break on Department
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
  5      group by cube (deptno,job)
  6  /

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        ANALYST            6000
                     CLERK              4150
                     MANAGER            8275
                     PRESIDENT          5000
                     SALESMAN           5600
                     All Employees     29025
```

In the CUBE example above, all of the information provided by ROLLUP (previous example) is joined by subtotals for each combination of categories (GROUP BY columns) in the output. Total salary for each type of job is added to the previously available information. The summary is listed in the "Whole Company" category since the summary is for everyone with a specific job regardless of department assignment.

Like ROLLUP, CUBE creates subtotals for each level of subtotal and a grand total. In addition, summaries are created for each combination of categories listed in the GROUP BY columns. If there were three GROUP BY columns (i.e. country, customer_id, product) GROUP BY would normally produce aggregates (subtotals) for 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.

```
Country     Customer_ID     Product           Sales
France      FR1234          Tennis Balls     34,000      <- produced by GROUP BY
France      FR1234                          100,000      <- Rollup produces total by country & cust.
France                                      340,000      <- Rollup produces total by country
            Customer ID                     135,000      <- Cube produces total for customer
            Customer ID     Tennis Balls     99,000      <- Cube produces total products by cust. ID
                            Tennis Balls    123,000      <- Cube produces total products sold
France                      Tennis Balls     78,000      <- Cube produces total products by country
Total                                     1,000,345      <- Rollup produces grand total (all records)
```

The information produced by CUBE is useful in cross-tabulation summarization reports often requested by management.

Materialized Views (Oracle8i)

Oracle's SNAPSHOT is a query result table that is created periodically to facilitate distribution or replication of data. the materialized view feature of Oracle8i uses similar technology to allow a view's results to be stored as materialized in the database for use by subsequent SQL statements. The view materialization is refreshed periodically based upon time criteria specified when the view is created or upon demand. View data is "old" until the view is refreshed. In addition, indexes may be defined for Materialized Views. This is an ideal mechanism for improving the performance of frequent requests for aggregate data.

```
create materialized view dept_summary
      refresh start with sysdate next sysdate + 1
    as
      select dept.deptno,
                 dname,
                 count(*) nbr_emps,
                 sum(nvl(sal,0)) tot_sal
            from scott.emp emp
                ,scott.dept dept
            where emp.deptno(+) = dept.deptno
            group by dept.deptno,dname;
```

Since this is an extension of SNAPSHOT technology first created in Oracle 7, the refresh mechanism and indexing schemes are reliable and efficient. A high-level statement breakdown follows, for more complete information see Oracle documentation:

- ORACLE recommends that materialized view names not exceed 19 characters, this makes the complete Oracle-generated name 30 characters or less.
- (not shown) Physical attributes (PCTFREE, PCTUSED, INITRANS, MAXTRANS, etc…), TABLESPACE, LOB, CACHE, LOGGING, CLUSTER, and partitioning are similar to CREATE SNAPSHOT and CREATE TABLE.
- BUILD IMMEDIATE (not shown) is the default, BUILD DEFERRED places data into view only when refreshed.
- ON PREBUILT TABLE (not shown) allows use of Materialized Views for existing tables (very handy!). If used, the Materialized View name and the Table name must be identical.
- REFRESH controls the rate of reloading, START WITH specifies the time of the first automatic refresh and NEXT specifies the time of the next refresh. Other REFRESH options include: FAST, COMPLETE, FORCE, ON COMMIT, ON DEMAND, START WITH, NEXT, WITH PRIMARY KEY, WITH ROWID, USING ROLLBACK SEGMENT. FAST uses a LOG defined ahead of time and requires that the materialized view's query conforms to guidelines in the Oracle8i Replication manual.
- AS describes the query used to create the materialized view, just about any query may be used with a few restrictions including: the table(s) referenced may not belong to the user "SYS", LONG columns are not allowed, views that include joins and GROUP BY may not select from an IOT (Index-Organized Table). Note that in the example the table name is fully qualified with the schema of the table owner, this is suggested by Oracle but is not required. The query may include GROUP BY, CUBE, ROLLUP, joins, nested queries (dynamic views), and just about any other construct.
- Oracle8i Release 2 (8.1.6) allows a query to contain the ORDER BY clause and for INSERT…SELECT into a materialized view to stipulate ORDER BY as well.

Using Pre-built Tables

Basing a materialized view upon an existing table (ON PREBUILT TABLE) allows the use of existing tables and indexes.

```
drop table dept_summary_tab;
drop snapshot dept_summary_tab;

create table dept_summary_tab
    as
      select dept.deptno
            ,dname
            ,count(*) nbr_emps
            ,sum(nvl(sal,0)) tot_sal
            from scott.emp emp
                ,scott.dept dept
            where emp.deptno(+) = dept.deptno
            group by dept.deptno,dname;
```

```
create materialized view dept_summary_tab
          on prebuilt table with reduced precision
       refresh start with sysdate next sysdate + 1
    as
      select dept.deptno
            ,dname
            ,count(*) nbr_emps
            ,sum(nvl(sal,0)) tot_sal
            from scott.emp emp
                ,scott.dept dept
            where emp.deptno(+) = dept.deptno
            group by dept.deptno,dname;
```

The use of ON PREBUILT TABLE requires that the underlying table and the materialized view share the same name and schema. WITH REDUCED PRECISION allows a refresh to work properly even if some columns generate different precision than originally defined.

To refresh an existing table, use the Oracle-provided PL/SQL package DBMS_MVIEW as shown below:

```
begin
  dbms_mview.refresh('dept_summary_tab');
end;
/
```

Be Careful! This packaged procedure COMMITs changes in the active transaction as part of its execution.

Here is an example use of the materialized view joined to the base table:

```
select dname,ename,sal,sal/tot_sal pct_dept
   from emp,dept_summary_tab
   where emp.deptno = dept_summary_tab.deptno
   order by dname
/
```

Conclusion

This paper explores Oracle8i's CUBE and ROLLUP extensions to the GROUP BY function and Materialized Views. Use of CUBE and ROLLUP reduces the work necessary to code and create aggregates of the sort often requested by management to provide competitive or summary information. Recaps of summary data are needed frequently to support management decisions. CUBE and ROLLUP provide a mechanism for using a single SQL statement to provide data that would have required multiple SQL statements, programming, or manual summarization in the past. Materialized Views reduce the impact of frequently executed aggregate queries by storing results and refreshing them on a periodic basis. Together, these tools may be used to "mine" Oracle databases for the "golden" information frequently in demand today.

About the Author

John King is a Partner in King Training Resources, a firm providing instructor-led training since 1988 across the United States and Internationally. John has worked with Oracle products since Version 4 and has been providing training to application developers since Oracle Version 5. He has presented papers at various industry events including: IOUG-A Live!, UKOUG Conference, EOUG Conference, ECO/SEOUC, RMOUG Training Days, and the ODTUG conference.

John Jay King
King Training Resources
6341 South Williams Street
Littleton, CO 80121-2627 U.S.A.

Phone: **1.300.798.5727** **1.800.252.0652**
Email: **john@kingtraining.com** **Website: http://www.kingtraining.com**

A copy of this paper and the accompanying slides may be found at our company website: http://www.kingtraining.com under Conference Downloads. If you have any questions about our training services, this paper, or comments, please contact us.

Bibliography

Oracle8i SQL Reference, Oracle Corporation

Oracle8i Concepts, Oracle Corporation

Oracle8i SQL Reference, Oracle Corporation

Oracle8i PL/SQL User's Guide and Reference, Oracle Corporation

Oracle8i Application Developer's Guide (Fundamentals, Advanced Queuing, Large Objects), Oracle Corporation

Oracle8i Administrator's Guide, Oracle Corporation

Oracle8i Supplied Packages Reference, Oracle Corporation

Oracle8i Replication, Oracle Corporation

Oracle8i Replication API Reference, Oracle Corporation

Oracle8i Tuning, Oracle Corporation