Optimizing and Simplifying Complex SQL with Advanced Grouping

Presented by: Jared Still

Pythian

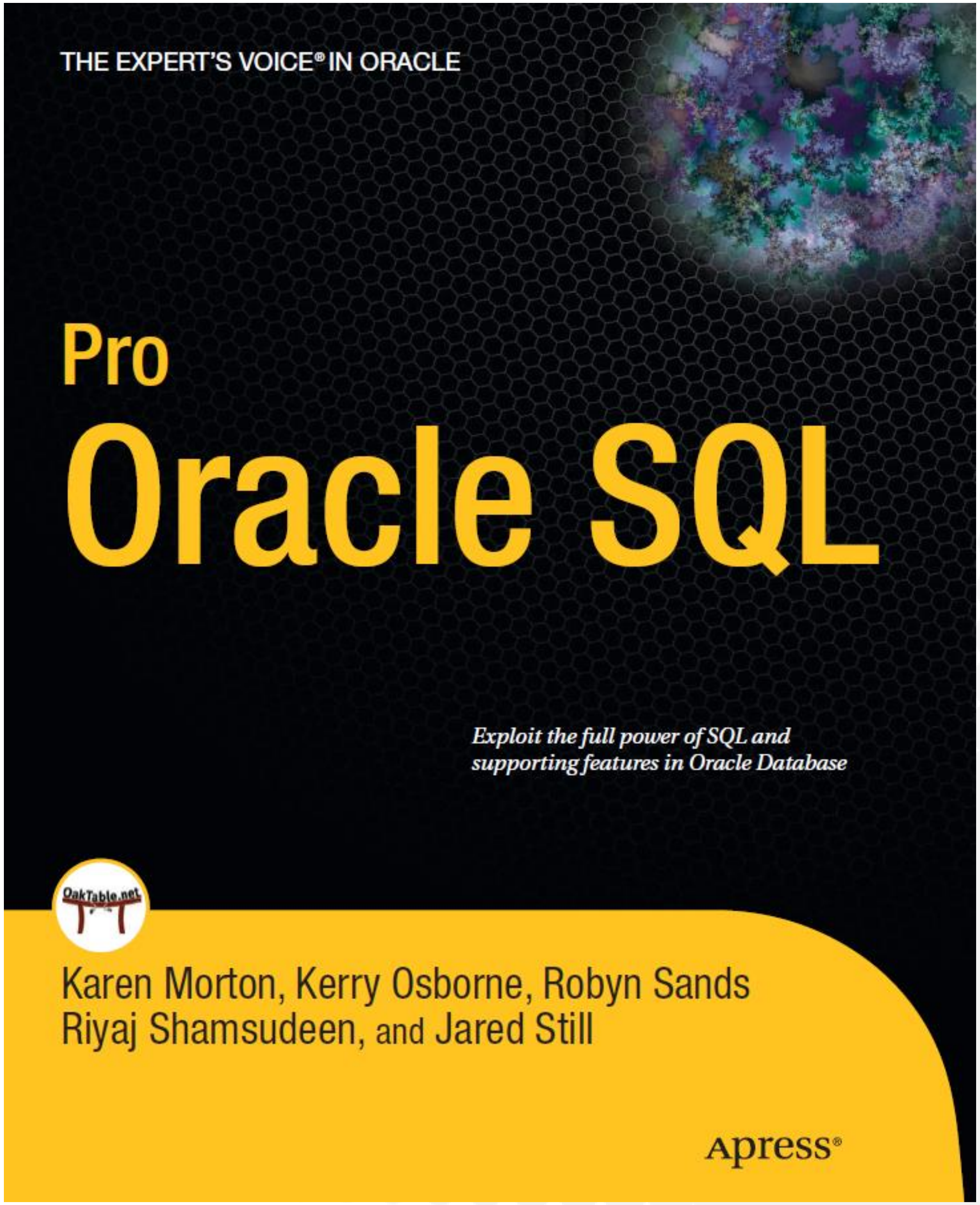
love your data

About Me

- Worked with Oracle since version 7.0
- Have an affinity for things Perlish, such as DBD::Oracle
- Working as a DBA at Pythian since Jan 2011
- Hobbies and extracurricular activities usually do not involve computers or databases.
- Contact: jkstill@gmail.com
- Oak Table
- Oracle ACE

• About this presentation

- We will explore advanced grouping functionality
- This presentation just skims the surface
- Truly understanding how to make use of advanced grouping you will need to invest some time experimenting with it and examining the results.



THE EXPERT'S VOICE® IN ORACLE

Pro Oracle SQL

Exploit the full power of SQL and supporting features in Oracle Database

OakTable.net

Karen Morton, Kerry Osborne, Robyn Sands
Riyaj Shamsudeen, and Jared Still

Apress®

Why talk about GROUP BY?

- Somewhat intimidating at first
- It seems to be underutilized
- The performance implications of GROUP BY are not often discussed

GROUP BY Basics

- GROUP BY does ***not*** guarantee a SORT
  @gb_1.sql

```
21:00:47 SQL> select  /*+ gather_plan_statistics */ deptno, count(*)
21:00:48   2  from scott.emp
21:00:48   3  group by deptno
21:00:48   4  /

    DEPTNO   COUNT(*)
---------- ----------
        30          6
        20          5
        10          3

3 rows selected.
```

- Notice the execution plan step is *HASH* GROUP BY

- Inline views and/or Subfactored Queries may change results – best not to rely on that behavior.

- GROUP BY can be HASH or SORT – neither guarantees sorted output

Introduction of GROUP BY functions

- 8i
  - CUBE()
    Generate rows for cross tab and summary reports
  - ROLLUP()
    Generate rows for summary reports – returns fewer null rows than CUBE()
  - GROUPING()
    Discern Superaggregate NULLs from Data NULLs
- 9i
  - GROUP_ID()
    Identify duplicate rows created by GROUP BY
  - GROUPING_ID()
    Returns a number corresponding to GROUPING bit vector for a row
  - GROUPING SETS
    Specify multiple groupings of data

GROUP BY Basics

- GROUP BY is a SQL optimization
- Following does 4 full table scans of EMP
  @gb_2.sql

```
select /*+ gather_plan_statistics */
distinct dname, decode(
       d.deptno,
       10, (select count(*) from scott.emp where deptno=
10),
       20, (select count(*) from scott.emp where deptno=
20),
       30, (select count(*) from scott.emp where deptno=
30),
       (select count(*) from scott.emp where deptno not in
(10,20,30))
) dept_count
from (select distinct deptno from scott.emp) d
join scott.dept d2 on d2.deptno = d.deptno;

DNAME          DEPT_COUNT
-------------- ----------
SALES                   6
ACCOUNTING              3
RESEARCH                5

3 rows selected.
```

GROUP BY Basics

- Use GROUP BY to reduce IO
- 1 full table scan of EMP
  @gb_3.sql

```
select /*+ gather_plan_statistics */
      d.dname
      , count(empno) empcount
from scott.emp e
join scott.dept d on d.deptno = e.deptno
group by d.dname
order by d.dname;

DNAME            EMPCOUNT
-------------- ----------
ACCOUNTING              3
RESEARCH                5
SALES                   6

3 rows selected.
```

GROUP BY Basics – HAVING

- Not used as much as it once was – here's why
- It is easily replaced by Subfactored Queries (ANSI CTE: Common Table Expressions)

```
select deptno,count(*)
from scott.emp
group by deptno
having count(*) > 5;
```

can be rewritten as:

```
with gcount as (
   select deptno,count(*) as dept_count
   from scott.emp
   group by deptno
)
select *
from gcount
where dept_count > 5;
```

Advanced GB – CUBE()

- Used to generate cross tab type reports
- Generates all combinations of columns in cube() @gb_4

```
with emps as (
    select /*+ gather_plan_statistics */
          ename
          , deptno
    from scott.emp
    group by cube(ename,deptno)
)
select rownum
    , ename
    , deptno
from emps
```

Advanced GB – CUBE()

- Notice the number of rows returned? 32
- Notice the #rows the raw query actually returned. 56 in GENERATE CUBE in execution plan.
- **Superaggregate** rows generated by Oracle with NULL for GROUP BY columns– these NULLS represent the set of all values (see GROUPING() docs).
- Re-examine output for rows with NULL.
- For each row, Oracle generates a row with NULL for all columns in CUBE()
- All but one of these rows is filtered from output with the SORT GROUP BY step.
- Number of rows is predictable - @gb_5.sql

Advanced GB – CUBE()

- Is CUBE() saving any work in the database?
- Without CUBE(), how would you do this?
- gb_6.sql – UNION ALL
- Notice the multiple TABLE ACCESS FULL steps
- CUBE() returned the same results with one TABLE scan

Advanced GB – CUBE()

- OK – so what good is it?
- Simple scripts for understanding
- sbase.sql
  - Sqlplus trick
  - s0.sql – show all test data
- s1.sql

Advanced GB – CUBE()

- Create a practical example
- Using the SALES example schema - Criteria:
  - all sales data for the year 2001.
  - sales summarized by product category,
  - aggregates based on 10-year customer age ranges, income levels,
  - summaries income level regardless of age group
  - summaries by age group regardless of income
- Here’s one way to do it.
- @gb_7.sql

Advanced GB – CUBE()

- Use CUBE() to generate the same output
- @gb_8.sql
- UNION ALL
  - 8 seconds
  - 9 table scans
- CUBE()
  - 4 seconds
  - 4 table scans
  - 2 index scans

Advanced GB–Discern SA NULL

- Look at output from previous SQL – See all those NULLS on CUST_INCOME_LEVEL and AGE_RANGE
- How should you handle them?
- Can you use NVL() ?
- How will you discern between NULL data and Superaggregate NULLs?
- @gb_9.sql
- Are all those NULL values generated as Superaggregate rows?

Advanced GB–GROUPING()

- Use GROUPING to discern Superaggregates
- @gb_10a.sql - 0 = data null, 1 = SA null
- Use with DECODE() or CASE to determine output
- @gb_10b.sql – examine the use of GROUPING()
- Now we can see which is NULL data and which is SA NULL, and assign appropriate text for SA NULL columns.
- @gb_11.sql - Put it to work in our Sales Report
- “ALL INCOME” and “ALL AGE” where sales are Aggregated on the income regardless of age, and age regardless of income.

Advanced GB–GROUPING_ID()

- GROUPING_ID() takes the idea behind GROUPING() up a notch
- GROUPING() returns 0 or 1
- GROUPING_ID() evaluates expressions and returns a bit vector – arguments correspond to bit position
- @gb_12a.sql
- GROUPING_ID() generates the GID values
- GROUPING() illustrates binary bit vector
- @gb_12b.sql
- OK – we made a truth table.
  What can we do with it?

Advanced GB–GROUPING_ID()

- Use GROUPING_ID() to customize sales report
- Useful for customizing report without any code change
  - Summaries only
  - Age Range only
  - Income level + summaries
  - etc…
- Options chosen by user are assigned values that correspond to bit vector used in GROUPING_ID()
- @gb_13.sql – examine PL/SQL block
- Experiment with different values and check output
- What do you think will happen when all options=0?
- How would you create this report without advanced grouping?
- No, I did not write an example – too much work. ☺

Advanced GB–ROLLUP()

- Similar to CUBE()
- for 1 argument ROLLUP() identical to CUBE()
- @gb_14a.sql
- for 1+N arguments ROLLUP produces fewer redundant rows
- @gb_14b.sql

Advanced GB–ROLLUP()

- ROLLUP() – running subtotals without UNION ALL
- Much like CUBE(), ROLLUP() reduces the database workload
- Sales Report:
  - All customers that begin with ‘Sul’
  - subtotal by year per customer
  - subtotal by product category per customer
  - grand total
- @gb_14c.sql

Advanced GB–GROUPING SETS

- Use with ROLLUP()
- GROUPING SETS allows aggregations not easily done with CUBE()
    - GROUP BY with columns or expressions
- s4.sql
    - Easily aggregate on country, region, and group of both columns
- s5.sql
    - Add rollup() to get grand total
    - May require ‘distinct’

Advanced GB–GROUPING SETS

- Use with ROLLUP()
- @gb_15a.sql
- This looks just like the CUBE() output from gb_14b.sql
- Add “Country” to generated data
- Total by Country and ROLLUP(Region, Group)
- @gb_15b.sql

Advanced GB–GROUPING SETS

- Combine what has been covered into the sales report
- @gb_16.sql
- Sometimes GROUPING SETS produces duplicate rows
- Last 2 lines of reports are duplicates
- In this case due to ROLLUP(PROD_CATEGORY)
- Use GROUP_ID() – its purpose is to distinguish duplicate rows caused by GROUP BY
- uncomment HAVING clause and rerun to see effect
- Performance Note:
  - GROUPING SETS is better at reducing workload
  - GROUPING_ID more flexible – no code changes

Advanced GROUP BY - Summary

- Greatly reduce database workload with Advance GROUP BY functionality
- Greatly reduce the amount of SQL to produce the same results
- There is a learning curve
- Start using it!

References

- URL: http://tinyurl.com/advanced-grouping
- Oracle 11g Documentation on advanced GROUP BY is quite good
- Pro Oracle SQL – Apress http://www.apress.com/9781430232285
- Pro Oracle SQL 2nd Edition – Apress http://www.apress.com/9781430262206
- Advanced SQL Functions in Oracle 10g http://www.amazon.com/Advanced-SQL-Functions-Oracle-10G/dp/818333184X

Grouping Glossary

CUBE()
GROUP_ID()
GROUPING()
GROUPING_ID()
GROUPING_SETS()
ROLLUP()

Glossary–SUPERAGGRETE ROW

GROUP BY extension will generate rows that have a NULL value in place of the value of the column being operated on.

The NULL represents the set of all values for that column.

The GROUPING() and GROUPING_ID() functions can be used to distinguish these.

Glossary – CUBE()

GROUP BY extension CUBE(expr1,expr2,…)

returns all possible combination of columns passed

Demo: gl_cube.sql

Glossary – GROUP_ID()

Function GROUP_ID()

Returns > 0 for duplicate rows

Demo: gl_group_id.sql

Glossary – ROLLUP()

GROUP BY extension ROLLUP(expr1, expr2,…)

Creates summaries of GROUP BY expressions

Demo: gl_rollup.sql

Glossary – GROUPING()

Function GROUPING(expr)

returns 1 for superaggregate rows

returns 0 for non-superaggregate rows

Demo: gl_rollup.sql

Used in demo to order the rows

Glossary – GROUPING_ID()

Function GROUPING_ID(expr)

returns a number representing the GROUP BY level of a row

Demo: gl_grouping_id.sql

Glossary – GROUPING SETS

GROUP BY Extension GROUPING SETS(expr1, expr2,…)

Used to create subtotals based on the expressions page

Demo: gl_grouping_sets.sql

GROUP BY Bug

- Malformed GROUP BY statements that worked < 11.2.0.2 may now get ORA-979 not a GROUP BY expression
- Due to bug #9477688 being fixed in 11.2.0.2
- Patch 10624168 can be used to re-institute previous behavior (must be patched offline – online mode patch is broken)
- @group_by_malformed.sql