ANALYZE THIS! USING ORACLE8I ANALYTIC FUNCTIONS

John Jay King, King Training Resources

ABSTRACT

Oracle 8.1.6 introduced new Analytic functions allowing complex statistical calculations to be accomplished more easily. Analytic functions provide performance benefits over the code previously required to accomplish the same tasks. New analytic function families include: lag/lead to compare values of rows in the same table, ranking to support "top n" queries, reporting to compare aggregates to non-aggregates, windowing to allow moving average types of queries, and statistics to extend the current power of aggregation. Analytic functions allow division of results into ordered groups using the over clause and its subordinate query partition clause, windowing clause, and order by clause.

INTRODUCTION

Oracle8i Release 2 (8.1.6) introduced several new features intended to enrich Oracle's ability to support decision making and statistical analysis including CUBE and ROLLUP extensions to GROUP BY as well as Analytic functions (the subject of this paper).

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. CUBE and ROLLUP are not discussed further in this paper except where they impact use of Analytic functions.

Analytic functions lend statistical muscle to SQL that has in the past called for joins, unions, and complex programming. Performance is improved (sometimes significantly) because the functions are performing work that previously required self-joins and unions. Using Analytic functions requires far less SQL coding than previously required to accomplish the same task because one SQL statement takes the place of many.

Analytic functions are categorized into five groups: Ranking, Windowing, Reporting, Lag/Lead, and Statistics. The first four are sometimes referred to as "Analytic Families" in Oracle literature. Statistics provide sophisticated aggregation capabilities.

Analytic functions are not intended to replace OLAP environments, rather, they may be used by OLAP products like Oracle's Express to improve query speed.

Together, the performance and readability of Analytic functions compared to what has been used make a compelling argument to move to the new techniques.

IMPORTANT CONCEPTS

Using Analytic functions adds a new stage to the processing of a query: First all joining, WHERE clause, GROUP BY, and HAVING clause activity selects desired rows; Next, the Analytic functions and any partitioning they require take place; Finally, SELECT DISTINCT and ORDER BY processing occurs for the query.

Query result sets are divided into ordered groups called Partitions (unrelated to database table partitioning). Partitioning (like all analytic functions) takes place after GROUP BY. Result sets may be divided into as many partitions as makes sense for the values being derived. Partitioning may be performed using expressions or column values. Each result set may represent a single Partition, a few larger Partitions, or many small Partitions.

Each Partition may be represented by a sliding Window defining the range of rows used for calculations on the Current Row (defined in the next paragraph). Windows may be defined representing a number of physical rows or some logical interval (e.g. time). Each Window has a starting row and an ending row and may slide on either end or at both ends. For example a cumulative sum's Window would be the (unmoving) first and last records of the partition. Or, a moving average would slide at both ends so that the averaging made sense. Windows may represent 1 or more rows in a partition (or the entire partition).

Each analytic function is based upon a current row within a Window (defined by OVER (ORDER BY) clause). That is, each calculation returns values that involve the rows included in the current Window. Current Row is the reference point setting the start and end of a window. For example a moving average defines a window that begins some number of rows before the current row, the current row, and some number of rows after the current row.

The Current Row is inside a Window, a Window is inside a Partition, and a Partition is inside of the Result Set.

Overview of Analytic Families

The various sets of Analytic "Families" each provide functions that solve a particular group of problems. Ranking functions allow values that represent some internal ordering of data such as "top 5 products sold by country" or "find the top three salespersons in each city" requiring that all rows be processed before performing the function. Windowing allows moving and cumulative capability to answer questions like "show a moving average for the last 3 months of sales by department" or "show a cumulative sum of sales by country." Reporting functions allow the comparison of aggregates to non-aggregates such as "percent of total department salaries represented by each employee." Lag/Lead compares values in different rows of the same table without having to code self-joins. Statistics provide a new set of group-level or aggregate data. Unlike the original aggregate functions, Statistics functions generally require two parameters.

Ranking

Ranking functions include: RANK, DENSE_RANK, CUME_DIST, PERCENT_RANK , NTILE, and ROW_NUMBER

RANK produces a ranking within a given set of rows using the OVER clause ORDER BY to define the sort sequence of the group. In the event of two values being equal the ranking skips as appropriate (e.g. 10->12 below).

```
select empno
      ,ename
      ,hiredate
      ,rank() over (order by hiredate) rank
  from emp
  order by hiredate,ename
```

```
     EMPNO ENAME      HIREDATE        RANK
---------- ---------- --------- ----------
      7369 SMITH      17-DEC-80          1
      7499 ALLEN      20-FEB-81          2
      7521 WARD       22-FEB-81          3
      7566 JONES      02-APR-81          4
      7698 BLAKE      01-MAY-81          5
      7782 CLARK      09-JUN-81          6
      7844 TURNER     08-SEP-81          7
      7654 MARTIN     28-SEP-81          8
      7839 KING       17-NOV-81          9
      7902 FORD       03-DEC-81         10
      7900 JAMES      03-DEC-81         10
      7934 MILLER     23-JAN-82         12
      7788 SCOTT      09-DEC-82         13
      7876 ADAMS      12-JAN-83         14
```

Rank may also be used with GROUP aggregation:

```
  1  select dname,
  2         nvl(avg(sal),0) avg_sal,
  3         count(empno) nbr_emps,
  4         rank() over (order by nvl(avg(sal),0)) rank
  5     from emp,dept
  6     where dept.deptno = emp.deptno(+)
  7*    group by dname

DNAME             AVG_SAL   NBR_EMPS       RANK
-------------- ---------- ---------- ----------
OPERATIONS              0          0          1
SALES          1566.66667          6          2
RESEARCH             2175          5          3
ACCOUNTING     2916.66667          3          4
```

DENSE_RANK also produces a ranking within a given set of rows using the OVER clause ORDER BY to define the sort sequence of the group. However, in the event of two values being equal the ranking does not skip.

```
  1  select empno
  2        ,ename
  3        ,hiredate
  4        ,dense_rank() over (order by hiredate) rank
  5    from emp
  6*   order by hiredate,ename

     EMPNO ENAME      HIREDATE        RANK
---------- ---------- --------- ----------
      7369 SMITH      17-DEC-80          1
      7499 ALLEN      20-FEB-81          2
      7521 WARD       22-FEB-81          3
      7566 JONES      02-APR-81          4
      7698 BLAKE      01-MAY-81          5
      7782 CLARK      09-JUN-81          6
      7844 TURNER     08-SEP-81          7
      7654 MARTIN     28-SEP-81          8
      7839 KING       17-NOV-81          9
      7902 FORD       03-DEC-81         10
      7900 JAMES      03-DEC-81         10
      7934 MILLER     23-JAN-82         11
      7788 SCOTT      09-DEC-82         12
      7876 ADAMS      12-JAN-83         13
```

Partitioning defines where the rank is reset.

```
select empno
      ,ename
      ,hiredate
      ,deptno
      ,rank() over (partition by deptno order by hiredate) rank
  from emp
  order by hiredate,ename
```

```
     EMPNO ENAME      HIREDATE      DEPTNO       RANK
---------- ---------- --------- ---------- ----------
      7369 SMITH      17-DEC-80         20          1
      7499 ALLEN      20-FEB-81         30          1
      7521 WARD       22-FEB-81         30          2
      7566 JONES      02-APR-81         20          2
      7698 BLAKE      01-MAY-81         30          3
      7782 CLARK      09-JUN-81         10          1
      7844 TURNER     08-SEP-81         30          4
      7654 MARTIN     28-SEP-81         30          5
      7839 KING       17-NOV-81         10          2
      7902 FORD       03-DEC-81         20          3
      7900 JAMES      03-DEC-81         30          6
      7934 MILLER     23-JAN-82         10          3
      7788 SCOTT      09-DEC-82         20          4
      7876 ADAMS      12-JAN-83         20          5
```

Partitioning also works with aggregates.

```
select dname,
       job,
       nvl(avg(sal),0) avg_sal,
       count(empno) nbr_emps,
       rank() over (partition by dname order by nvl(avg(sal),0)) rank
   from emp,dept
   where dept.deptno = emp.deptno(+)
   group by dname, job
```

```
DNAME          JOB          AVG_SAL   NBR_EMPS       RANK
-------------- --------- ---------- ---------- ----------
ACCOUNTING     CLERK           1300          1          1
ACCOUNTING     MANAGER         2450          1          2
ACCOUNTING     PRESIDENT       5000          1          3
```

```
OPERATIONS                        0          0          1
RESEARCH       CLERK            950          2          1
RESEARCH       MANAGER         2975          1          2
RESEARCH       ANALYST         3000          2          3
SALES          CLERK            950          1          1
SALES          SALESMAN        1400          4          2
SALES          MANAGER         2850          1          3
```

Rank also might include rows created by CUBE or ROLLUP.

```
select deptno Department
      ,decode(grouping(job),1,'All Employee
      ,sum(sal)  "Total SAL"
      ,rank() over (order by sum(sal)) rank
      from emp
     group by rollup (deptno,job)
```

```
DEPARTMENT JOB            Total SAL       RANK
---------- ------------- ---------- ----------
       30 CLERK                950          1
       10 CLERK               1300          2
       20 CLERK               1900          3
       10 MANAGER             2450          4
       30 MANAGER             2850          5
       20 MANAGER             2975          6
       10 PRESIDENT           5000          7
       30 SALESMAN            5600          8
       20 ANALYST             6000          9
       10 All Employees       8750         10
       30 All Employees       9400         11
       20 All Employees      10875         12
          All Employees      29025         13
```

The GROUPING() function provided with ROLLUP and CUBE may also be used.

```
select deptno Department
      ,decode(grouping(job),1,'All Employees',job) job
      ,sum(sal)  "Total SAL"
      ,rank() over (partition by grouping(job) order by sum(sal)) rank
      from emp
     group by rollup (deptno,job)
```

```
DEPARTMENT JOB            Total SAL       RANK
---------- ------------- ---------- ----------
        30 CLERK                950          1
        10 CLERK               1300          2
        20 CLERK               1900          3
        10 MANAGER             2450          4
        30 MANAGER             2850          5
        20 MANAGER             2975          6
        10 PRESIDENT           5000          7
        30 SALESMAN            5600          8
        20 ANALYST             6000          9
        10 All Employees       8750          1
        30 All Employees       9400          2
        20 All Employees      10875          3
           All Employees      29025          4
```

"Top N" queries may be solved easily by using RANK or DENSE_RANK in dynamic view (query in FROM clause).

```
select dynemp.ename
      ,dynemp.job
      ,dynemp.sal
      ,dynemp.rank
   from (select ename
          ,sal
          ,job
          ,dense_rank() over (partition by job order by sal desc) rank
    from emp) dynemp
  where dynemp.rank < 3
  order by dynemp.job
          ,dynemp.rank
```

```
ENAME      JOB              SAL       RANK
---------- --------- ---------- ----------
SCOTT      ANALYST         3000          1
FORD       ANALYST         3000          1
MILLER     CLERK           1300          1
ADAMS      CLERK           1100          2
JONES      MANAGER         2975          1
BLAKE      MANAGER         2850          2
KING       PRESIDENT       5000          1
ALLEN      SALESMAN        1600          1
TURNER     SALESMAN        1500          2
```

NULLs are treated like normal values and for ranking are treated as equal to other NULLs. The ORDER BY clause may specify NULLS FIRST or NULLS LAST. If unspecified NULLS are treated as larger than any other value and appear depending upon the ASC or DESC part of the ORDER BY.

NTILE divides the result set into the specified number of groups and then includes each value according to its ranking.

```
  1  select empno
  2        ,ename
  3        ,hiredate
  4        ,rank() over (order by hiredate) rank
  5        ,ntile(3) over (order by hiredate) ntile3
  6*   from emp

     EMPNO ENAME      HIREDATE        RANK     NTILE3
---------- ---------- --------- ---------- ----------
      7369 SMITH      17-DEC-80          1          1
      7499 ALLEN      20-FEB-81          2          1
      7521 WARD       22-FEB-81          3          1
      7566 JONES      02-APR-81          4          1
      7698 BLAKE      01-MAY-81          5          1
      7782 CLARK      09-JUN-81          6          2
      7844 TURNER     08-SEP-81          7          2
      7654 MARTIN     28-SEP-81          8          2
      7839 KING       17-NOV-81          9          2
      7900 JAMES      03-DEC-81         10          2
      7902 FORD       03-DEC-81         10          3
      7934 MILLER     23-JAN-82         12          3
      7788 SCOTT      09-DEC-82         13          3
      7876 ADAMS      12-JAN-83         14          3
```

ROW_NUMBER assigns a unique value (starting with 1, incrementing by 1 in the ORDER BY sequence) to each row within the partition.

```
  1  select ename
  2        ,job
  3        ,hiredate
  4        ,rank() over (partition by job order by hiredate desc) hire_ra
  5        ,row_number() over(partition by job order by hiredate) row_nbr
  6    from emp
  7*   order by job,hiredate,ename

ENAME      JOB       HIREDATE   HIRE_RANK    ROW_NBR
```

```
---------- --------- --------- ---------- ----------
FORD       ANALYST   03-DEC-81          2          1
SCOTT      ANALYST   09-DEC-82          1          2
SMITH      CLERK     17-DEC-80          4          1
JAMES      CLERK     03-DEC-81          3          2
MILLER     CLERK     23-JAN-82          2          3
ADAMS      CLERK     12-JAN-83          1          4
JONES      MANAGER   02-APR-81          3          1
BLAKE      MANAGER   01-MAY-81          2          2
CLARK      MANAGER   09-JUN-81          1          3
KING       PRESIDENT 17-NOV-81          1          1
ALLEN      SALESMAN  20-FEB-81          4          1
WARD       SALESMAN  22-FEB-81          3          2
TURNER     SALESMAN  08-SEP-81          2          3
MARTIN     SALESMAN  28-SEP-81          1          4
```

CUME_DIST

CUME_DIST determines the position of a specific value relative to a set of values.

```
select deptno,job,sum(sal) sum_sal
   , cume_dist() over (order by job) cume
  from emp
  group by deptno,job
```

```
    DEPTNO JOB          SUM_SAL       CUME
---------- --------- ---------- ----------
        20 ANALYST         6000 .111111111
        10 CLERK           1300 .444444444
        20 CLERK           1900 .444444444
        30 CLERK            950 .444444444
        10 MANAGER         2450 .777777778
        20 MANAGER         2975 .777777778
        30 MANAGER         2850 .777777778
        10 PRESIDENT       5000 .888888889
        30 SALESMAN        5600          1
```

Partition adds some meaning to this

```
select deptno,job,sum(sal) sum_sal
   , cume_dist() over (order by job) cume
  from emp
  group by deptno,job
```

```
    DEPTNO JOB          SUM_SAL       CUME
---------- --------- ---------- ----------
        20 ANALYST         6000 .111111111
        10 CLERK           1300 .444444444
        20 CLERK           1900 .444444444
        30 CLERK            950 .444444444
        10 MANAGER         2450 .777777778
        20 MANAGER         2975 .777777778
        30 MANAGER         2850 .777777778
        10 PRESIDENT       5000 .888888889
        30 SALESMAN        5600          1
```

PERCENT_RANK calculates the percent rank of a value relative to the number of rows.

```
select deptno,job,sum(sal) sum_sal
   , percent_rank() over (order by deptno) pct_rank
  from emp
  group by deptno,job
  order by job,deptno
```

```
    DEPTNO JOB          SUM_SAL   PCT_RANK
---------- --------- ---------- ----------
        20 ANALYST         6000       .375
        10 CLERK           1300          0
        20 CLERK           1900       .375
        30 CLERK            950        .75
        10 MANAGER         2450          0
        20 MANAGER         2975       .375
        30 MANAGER         2850        .75
        10 PRESIDENT       5000          0
        30 SALESMAN        5600        .75
```

Again, Partitioning adds a little clarity.

```
select deptno,job,sum(sal) sum_sal
   , percent_rank() over (partition by job order by deptno) pct_rank
  from emp
  group by deptno,job
  order by job,deptno
```

```
    DEPTNO JOB          SUM_SAL   PCT_RANK
---------- --------- ---------- ----------
        20 ANALYST         6000          0
        10 CLERK           1300          0
        20 CLERK           1900         .5
        30 CLERK            950          1
        10 MANAGER         2450          0
        20 MANAGER         2975         .5
        30 MANAGER         2850          1
        10 PRESIDENT       5000          0
        30 SALESMAN        5600          0
```

WINDOWING

Windowing functions create moving, centered, and cumulative aggregates based upon the value of rows that depend upon rows in the other window. The Windowing functions that may be used are AVG, COUNT, MAX, MIN, STDDEV, SUM, VARIANCE, FIRST_VALUE, and LAST_VALUE. Bounds include CURRENT ROW, UNBOUNDED PRECEDING, and UNBOUNDED FOLLOWING.

```
  1  select empno
  2        ,deptno
  3        ,sal
  4        ,sum(sal) over (partition by deptno
  5                        order by empno
  6                        rows 2 preceding) as sumsal
  7     from emp
  8*    order by deptno,empno

     EMPNO     DEPTNO        SAL     SUMSAL
---------- ---------- ---------- ----------
      7782         10       2450       2450
      7839         10       5000       7450
      7934         10       1300       8750
      7369         20        800        800
      7566         20       2975       3775
      7788         20       3000       6775
      7876         20       1100       7075
      7902         20       3000       7100
      7499         30       1600       1600
      7521         30       1250       2850
      7654         30       1250       4100
      7698         30       2850       5350
      7844         30       1500       5600
      7900         30        950       5300
```

A moving average may be created using bounds. Bounds include a number of rows in addition to a range.

```
  1  select deptno
  2        ,empno
  3        ,hiredate
  4        ,sal
  5        ,avg(sal) over (partition by deptno
  6                        order by hiredate
  7                        range between interval '10' month preceding
  8                           and interval '10' month following) ten_day
  9     from emp
 10*    order by deptno,hiredate,empno
```

```
    DEPTNO      EMPNO HIREDATE         SAL TEN_DAY
---------- ---------- --------- ---------- ----------
        10       7782 09-JUN-81       2450 2916.66667
        10       7839 17-NOV-81       5000 2916.66667
        10       7934 23-JAN-82       1300 2916.66667
        20       7369 17-DEC-80        800     1887.5
        20       7566 02-APR-81       2975 2258.33333
        20       7902 03-DEC-81       3000     2987.5
        20       7788 09-DEC-82       3000       2050
        20       7876 12-JAN-83       1100       2050
        30       7499 20-FEB-81       1600 1566.66667
        30       7521 22-FEB-81       1250 1566.66667
        30       7698 01-MAY-81       2850 1566.66667
        30       7844 08-SEP-81       1500 1566.66667
        30       7654 28-SEP-81       1250 1566.66667
        30       7900 03-DEC-81        950 1566.66667
```

In addition to the aggregates that are familiar, two special functions are available: FIRST_VALUE returns the first value in the window, LAST_VALUE returns the last.

```
select deptno
      ,empno
      ,hiredate
      ,sal
      ,avg(sal) over (partition by deptno
                      order by hiredate
                      range between interval '3' month preceding
                           and interval '3' month following) three_mon
      ,first_value(sal) over (partition by deptno
                      order by hiredate
                      range between interval '3' month preceding
                           and interval '3' month following) first_val
      ,last_value(sal) over (partition by deptno
                      order by hiredate
                      range between interval '3' month preceding
                           and interval '3' month following) last_val
   from emp
   order by deptno,hiredate,empno
```

```
    DEPTNO      EMPNO HIREDATE         SAL  THREE_MON  FIRST_VAL   LAST_VAL
---------- ---------- --------- ---------- ---------- ---------- ----------
        10       7782 09-JUN-81       2450       2450       2450       2450
        10       7839 17-NOV-81       5000       3150       5000       1300
        10       7934 23-JAN-82       1300       3150       5000       1300
        20       7369 17-DEC-80        800        800        800        800
        20       7566 02-APR-81       2975       2975       2975       2975
        20       7902 03-DEC-81       3000       3000       3000       3000
        20       7788 09-DEC-82       3000       2050       3000       1100
        20       7876 12-JAN-83       1100       2050       3000       1100
        30       7499 20-FEB-81       1600       1900       1600       2850
        30       7521 22-FEB-81       1250       1900       1600       2850
        30       7698 01-MAY-81       2850       1900       1600       2850
        30       7844 08-SEP-81       1500 1233.33333       1500        950
        30       7654 28-SEP-81       1250 1233.33333       1500        950
        30       7900 03-DEC-81        950 1233.33333       1500        950
```

REPORTING

Reporting functions use the values that have been generated by other aggregates. The aggregates that may be used include AVG, COUNT, MAX, MIN, STDDEV, SUM, and VARIANCE. Reporting functions may only be used in the SELECT and ORDER BY clause.

```
select deptno
      ,job
      ,sal
      ,maxsal
from (select deptno
            ,job
            ,sal
            ,max(sal) over
             (partition by deptno) maxsal
       from emp )
where sal = maxsal

    DEPTNO JOB              SAL     MAXSAL
---------- --------- ---------- ----------
        10 PRESIDENT       5000       5000
        20 ANALYST         3000       3000
        20 ANALYST         3000       3000
        30 MANAGER         2850       2850
```

The ratio_to_report function computes the ration of the value to the aggregate value.

```
select deptno
      ,sum(sal) sumsal
      ,sum(sum(sal)) over () sumsumsal
      ,ratio_to_report(sum(sal)) over () ratio
from emp
group by deptno

    DEPTNO     SUMSAL  SUMSUMSAL      RATIO
---------- ---------- ---------- ----------
        10       8750      29025 .301464255
        20      10875      29025 .374677003
        30       9400      29025 .323858742
```

LAG/LEAD

LAG and LEAD obtain values from other rows in the same table. This is particularly useful when dealing with time periods but is not limited to time.

```
select empno
      ,ename
      ,lag(empno,1) over (order by empno) lag1_emp
      ,lead(empno,1) over (order by empno) lead1_emp
      ,lag(empno,3) over (order by empno) lag3_emp
      ,lead(empno,3) over (order by empno) lead3_emp
from emp
```

```
     EMPNO ENAME        LAG1_EMP  LEAD1_EMP   LAG3_EMP  LEAD3_EMP
---------- ---------- ---------- ---------- ---------- ----------
      7369 SMITH                       7499                  7566
      7499 ALLEN            7369       7521                  7654
      7521 WARD             7499       7566                  7698
      7566 JONES            7521       7654       7369       7782
      7654 MARTIN           7566       7698       7499       7788
      7698 BLAKE            7654       7782       7521       7839
      7782 CLARK            7698       7788       7566       7844
      7788 SCOTT            7782       7839       7654       7876
      7839 KING             7788       7844       7698       7900
      7844 TURNER           7839       7876       7782       7902
      7876 ADAMS            7844       7900       7788       7934
      7900 JAMES            7876       7902       7839
      7902 FORD             7900       7934       7844
      7934 MILLER           7902                  7876
```

Statistics

New statistical functions provide complex mathematics not present in Oracle previously incuding CORR, COVAR_POP, COVAR_SAMP, REGR_AVGX, REGR_AVGY, REGR_COUNT, REGR_INTERCEPT, REGR_R2, REGR_SLOPE, REGR_SXX, REGR_SYY, REGR_SXY, STDDEV_POP, STDDEV_SAMP, VAR_POP, and VAR_SAMP.

Conclusion

This paper has presented the new analytic functions supported by Oracle. Lag and Lead compare values of rows to other rows in the same table. Ranking support "top n" queries and other ranking issues, reporting aggregates compare aggregates to non-aggregates, windowing aggregates provide cumulative or moving aggregates, and statistics provide complex statistical features.

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 and the database 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:     1.303.798.5727  1.800.252.0652 (within the U.S.)
Fax:       1.303.730.8542
Email:     john@kingtraining.com
```

If you have any questions or comments, please contact me in the fashion most convenient to you. Copies of this paper are available for download from King Training Resources upon request (www.kingtraining.com).

Bibliography

Oracle8i SQL Reference, Oracle Corporation

Oracle8i Data Warehousing Guide, Oracle Corporation