Analyze This!

Using Oracle8i Analytic Functions

John Jay King

King Training Resources

6341 South Williams Street

Littleton, CO 80121-2627 USA

www.kingtraining.com

800.252.0652 or 303.798.5727

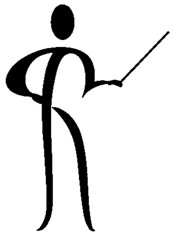
Objectives

- Learn about the Oracle8i analytic functions
- Understand the different “families” available:
  - Ranking
  - Windowing
  - Reporting
  - Lag/Lead
  - Statistics
- Be familiar with the options used to make analytic functions useful
- Be ready to use the analytic functions to create useful business statistics

Oracle8i Analytic Functions

- Oracle 8.1.6 Analytic functions allow complex statistical calculations to be accomplished more easily.
- Analytic functions lend statistical muscle 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 allow division of results into ordered groups using the over clause and its subordinate query partition clause, windowing clause, and order by clause.

Analytic Function Families



- 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
- Statistics to extend the current power of aggregation

Introduction to Analytic "Families"

- Ranking, Windowing, Reporting, and Lag/Lead 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.

Analytic Processing

- 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 occur

Analytic Function Query Partitions

- 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.

Query Partition Result Sets

- 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.

Analytic Function Windows

- 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 (or both) ends.
- A cumulative sum's Window might 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).

Current Row

- Each analytic function is based upon a current row within a Window (defined by OVER (ORDER BY) clause). 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 in a range or rows surrounding 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.

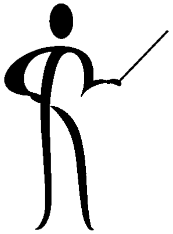
Ranking Functions

- Ranking functions allow values that represent some internal ordering of data
- Ranking supports such queries 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 Functions

- Windowing functions allow 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



- Reporting functions allow the comparison of aggregates to non-aggregates such as "percent of total department salaries represented by each employee."

Lag/Lead Functions

- Lag/Lead compares values in different rows of the same table without having to code self-joins.

Statistics

- 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 in following example).

Ranking Syntax

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

Ranking Output

```
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 with GROUP Aggregates

- 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
```

RANK with GROUP Output

```
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

- DENSE_RANK produces a ranking within a given set of rows using the OVER clause ORDER BY to define the sort sequence of
- If two values are equal the ranking does not skip.

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

DENSE_RANK Output

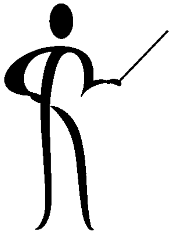
```
EMPNO  ENAME         HIREDATE         RANK
_____  _____         ----_____-       _____
7369   SMITH         17-DEC-80           1
7499   ALLEN         20-FEB-81           2
7521   WARD          22-FEB-81           3
7566   JONES         02-APR-81           4
7698   BLAKE         01-MAY-81           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 with Rank

- Partitioning defines where the rank is reset

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

Partitioning with Rank Results



```
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 and Aggregates

- 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
```

Partitioning and Aggregates Output

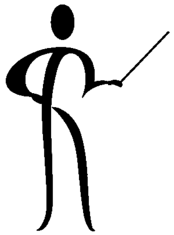
```
DNAME         JOB          AVG_SAL    NBR_EMPS    RANK
———————       ————-        --——————     —————     —————
ACCOUNTING    CLERK          1300          1        1
ACCOUNTING    MANAGER        2450          1        2
ACCOUNTING    PRESIDENT      5000          1        3
OPERATIONS                      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 with CUBE or ROLLUP

- Rank also might include rows created by CUBE or ROLLUP.

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

Rank CUBE/ROLLUP Output



```
DEPARTMENT    JOB                 Total SAL      RANK
_____         ______-                _____      _____
30            CLERK                    950         1
10            CLERK                   1300         2
20            CLERK                   1900         3
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
```

Rank with GROUPING Function

- 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)
```

Rank with GROUPING Output

```
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 using RANK/DENSE_RANK

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

“Top N” Syntax

```
1  select dynemp.ename
 2       ,dynemp.job
 3       ,dynemp.sal
 4       ,dynemp.rank
 5    from (select ename
 6          ,sal
 7          ,job
 8          ,dense_rank() over (partition by job order by sal desc) rank
 9     from emp) dynemp
10    where dynemp.rank < 3
11    order by dynemp.job
12*          ,dynemp.rank
```

“Top N” Output

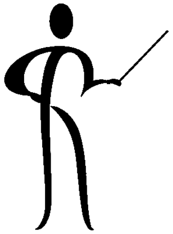
```
ENAME        JOB           SAL         RANK
_____        ____-        _____        _____
SCOTT        ANALYST       3000          1
FORD         ANALYST       3000          1
MILLER       CLERK         1300          1
ADAMS        CLERK         1100          2
JONES        MANAGER       2975    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

Top 2 Sales of Tennis Rackets

- Using the Oracle Customer and Sales tables:

```
select custid,prodname,avg_sales,rank
 from (select state,city,customer.custid,prodname
           ,rank() over (partition by prodname
                order by nvl(avg(amount),0) desc) rank
           ,avg(amount) avg_sales
      from customer,sales
      where customer.custid = sales.custid
      group by state,city,customer.custid,prodname)
  where prodname like 'ACE TENNIS RACKET%'
   and rank < 3
  order by prodname,rank
```

Top 2 Sales: Output



```
CUSTID  PRODNAME                   AVG_SALES    RANK
------  -----------------------   ---------   ------
102     ACE TENNIS RACKET I         16569          1
104     ACE TENNIS RACKET I          3000          2
106     ACE TENNIS RACKET II         4584          1
105     ACE TENNIS RACKET II         4500          2
```

NTILE

- 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
```

NTILE Output

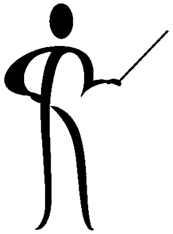
```
EMPNO  ENAME         HIREDATE      RANK    NTILE3
_____  _____         ____-----    _____    _____
7369   SMITH         17-DEC-80        1        1
7499   ALLEN         20-FEB-81        2        1
7521   WARD          22-FEB-81        3        1
7566   JONES         02-APR-81        4        1
7698   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

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

```
select ename
   ,job
  ,hiredate
  ,rank() over (partition by job order by hiredate desc) hire_rank
  ,row_number() over(partition by job order by hiredate) row_nbr
   from emp
   order by job,hiredate,ename
```

ROW_NUMBER Output



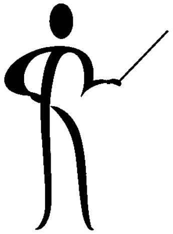
```
ENAME        JOB          HIREDATE   HIRE_RANK  ROW_NBR
-----        ---------    ---------    -----      -----
FORD         ANALYST      03-DEC-81    2            1
SCOTT        ANALYST      09-DEC-82    1            2
SMITH        CLERK        17-DEC-80    4            1
JAMES        CLERK        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.

```
1  select deptno,job,sum(sal) sum_sal
2      , cume_dist() over (order by job) cume
3    from emp
4*   group by deptno,job
```

CUME_DIST Output



```
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
```

CUME_DIST with Partition

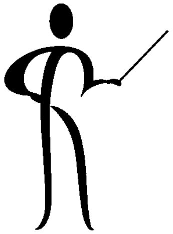
- Partition adds some meaning to the previous example:

```
1  select deptno,job,sum(sal) sum_sal
2      , cume_dist() over (partition by job order by deptno) cume
3    from emp
4    group by deptno,job
5*   order by job,deptno
```

CUME_DIST with Partition Output

```
DEPTNO        JOB        SUM_SAL          CUME
--------      --------- ----------      ----------
20            ANALYST      6000          1
10            CLERK        1300           .333333333
20            CLERK        1900           .666666667
30            CLERK         950          1
10            MANAGER      2450           .333333333
20            MANAGER      2975           .666666667
30            MANAGER      2850          1
10            PRESIDENT    5000          1
30            SALESMAN     5600          1
```

PERCENT_RANK



- 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
```

PERCENT_RANK Output

```
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
```

PERCENT_RANK with Partition

- Again, Partitioning adds a little clarity.

```
 1  select deptno,job,sum(sal) sum_sal
2     , percent_rank() over (partition by job order by deptno) pct_rank
3    from emp
4    group by deptno,job
5*   order by job,deptno
```

PERCENT_RANK with Partition Output

```
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.

Windowing Syntax

```
  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
```

Windowing Output

```
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
```

Moving Average with Bounds

- Moving averages 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) twenty_mo
 9    from emp
10*   order by deptno,hiredate,empno
```

Moving Average with Bounds Output

```
DEPTNO    EMPNO       HIREDATE     SAL        TWENTY_MO
______    ______      ----_____-   _____      ----______
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
```

FIRST_VALUE and LAST_VALUE

- 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 value in a window

FIRST_VALUE & LAST_VALUE Syntax

```
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
```

FIRST_VALUE & LAST_VALUE Output

```
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
```

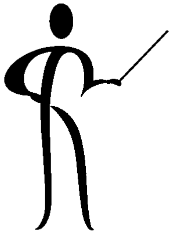
Reporting

- 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.

Reporting Syntax

```
 1  select deptno
 2        ,job
 3        ,sal
 4        ,maxsal
 5  from (select deptno
 6              ,job
 7              ,sal
 8              ,max(sal) over
 9               (partition by deptno) maxsal
10        from emp )
11* where sal = maxsal
```

Reporting Output



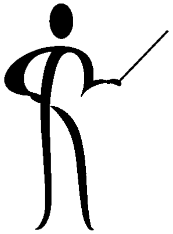
```
DEPTNO    JOB                 SAL     MAXSAL
-----     -----              -----    -----
10       PRESIDENT           5000      5000
20       ANALYST             3000      3000
20       ANALYST             3000      3000
30       MANAGER             2850      2850
```

RATIO_TO_REPORT

- 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
```

RATIO_TO_REPORT Output



```
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.
- Lag and lead are particularly useful when dealing with time periods but are not limited to time.

```
1  select empno
2        ,ename
3        ,lag(empno,1) over (order by empno) lag1_emp
4        ,lead(empno,1) over (order by empno) lead1_emp
5        ,lag(empno,3) over (order by empno) lag3_emp
6        ,lead(empno,3) over (order by empno) lead3_emp
7* from emp
```

Lag/Lead Output

```
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
```

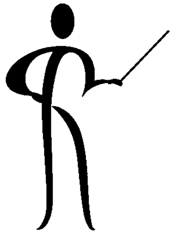
Statistics Functions

- 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
- VAR_SAMP

Oracle 8.1.7 CASE Expression

```
select custid,prodname,
     case when rank = 1 then ’1 - Platinum'
          when rank = 2 then ’2 - Gold'
          when rank = 3 then ’3 - Silver'
          else 'N/A'
          end as rank
  from (select state,city,customer.custid,prodname
              ,rank() over (partition by prodname
                 order by nvl(avg(amount),0) desc)
                        as rank
            from customer,sales
            where customer.custid = sales.custid
            group by state,city,customer.custid,prodname)
   where prodname like 'ACE TENNIS RACKET%'
     and rank < 4
   order by prodname,rank
```

Conclusion



- This paper has presented the new analytic functions supported by Oracle including:
  - 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

King

Training Resources

To contact the author:

John King
King Training Resources
6341 South Williams Street
Littleton, CO 80121-2627 USA
1.800.252.0652 - 1.303.798.5727
Email: john@kingtraining.com