SOUG

Suncoast Oracle Users' Group

Technology Day 2006

Oracle Analytics

Lewis Cunningham

Shepherd Systems

September 26, 2005

SQL Analytics

Lewis R Cunningham
Database Architect
Sheperd Systems

An Expert's Guide to Oracle
http://blogs.ittoolbox.com/oracle/guide

An expert is a person who has made all the mistakes that can be made in a very narrow field. - Niels Bohr (1885 - 1962)

Introduction to Oracle Analytic Functions

David Wong

Introduction

- Analytic functions were introduced in Release 2 of 8i and simplify greatly the means by which pivot reports and OLAP queries can be computed in straight, non-procedural SQL.
- Prior to the introduction of analytic functions, complex reports could be produced in SQL by complex self-joins, sub-queries and inline-views but these were resource-intensive and very inefficient.

Introduction

- Furthermore, if a question to be answered was too complex, it could be written in PL/SQL, which by its very nature is usually less efficient than a single SQL statement

Addresses These Problems

- Calculate a running total
- Top-N Queries
- Compute a moving average
- Rankings and percentiles
- Lag/lead analysis
- First/last analysis
- Linear regression statistics
- And more…

How Analytic Functions Work

- Analytic functions compute an aggregate value based on a group of rows. They differ from aggregate functions in that they return multiple rows for each group.
- Analytic functions are the last set of operations performed in a query except for the final ORDER BY clause. Therefore, analytic functions can appear only in the select list or ORDER BY clause.

The Syntax

- Analytic-Function(<Argument>,<Argument>,...)
  OVER (
    <Query-Partition-Clause>
    <Order-By-Clause>
    <Windowing-Clause>
)
- Analytic-Function – *AVG, CORR, COVAR_POP, COVAR_SAMP, COUNT, CUME_DIST, DENSE_RANK, FIRST, FIRST_VALUE, LAG, LAST, LAST_VALUE, LEAD, MAX, MIN, NTILE, PERCENT_RANK, PERCENTILE_CONT, PERCENTILE_DISC, RANK, RATIO_TO_REPORT, STDDEV, STDDEV_POP, STDDEV_SAMP, SUM, VAR_POP, and more.*

The Syntax

- **Query-Partition-Clause -**Logically breaks a single result set into N groups, according to the criteria set by the partition expressions. The words "partition" and "group" are used synonymously here. The analytic functions are applied to each group independently, they are reset for each group
- **Order-By-Clause -** Specifies how the data is sorted within each group (partition). This will definitely affect the outcome of any analytic function.

The Syntax

- **Windowing-Clause -** The windowing clause gives us a way to define a sliding or anchored window of data, on which the analytic function will operate, within a group. This clause can be used to have the analytic function compute its value based on any arbitrary sliding or anchored window within a group.

Running Total Example

- **Calculate a cumulative salary within a department row by row**

LAST_NAME	DEPARTMENT_ID	SALARY
Whalen	10	4400
Fay	20	6000
Hartstein	20	13000
Baida	30	2900
Colmenares	30	2500
Himuro	30	2600
Khoo	30	3100
Raphaely	30	11000
Tobias	30	2800

Running Total Example

```
SELECT
      last_name,
      department_id,
      salary,
      SUM(salary) OVER
              (PARTITION BY department_id
              ORDER BY last_name) AS running_total
      ROW_NUMBER() OVER
              (PARTITION BY department_id
              ORDER BY last_name) AS  emp_sequence
FROM
      employees
ORDER BY
      department_id,
      last_name;
```

Running Total Example

LAST_NAME	DEPARTMENT_ID	SALARY	RUNNING_TOTAL	EMP_SEQUENCE
Whalen	10	4400	4400	1
Fay	20	6000	6000	1
Hartstein	20	13000	19000	2
Baida	30	2900	2900	1
Colmenares	30	2500	5400	2
Himuro	30	2600	8000	3
Khoo	30	3100	11100	4
Raphaely	30	11000	22100	5
Tobias	30	2800	24900	6

ROW_NUMBER function

- `ROW_NUMBER` is an analytic function. It assigns a unique number to each row to which it is applied (either each row in the partition or each row returned by the query), in the ordered sequence of rows specified in the `order_by_clause`, beginning with 1.

Top-N Query Example

- **Find the top four paid sales rep by department**

LAST_NAME	DEPARTMENT_ID	SALARY
Ozer	80	11500
Errazuriz	80	12000
Partners	80	13500
Russell	80	14000
Cambrault	80	11000
Hunold	60	9000
Ernst	60	6000
Austin	60	4800
Pataballa	60	4800
Lorentz	60	4200

Top-N Query Example

ROW_NUMBER SOLUTION

```
SELECT
    *
FROM
(
    SELECT
          department_id,
          last_name,
          salary,
          ROW_NUMBER() OVER
                   (PARTITION BY department_id
                   ORDER BY salary DESC) AS top4
    FROM
          employees
)
WHERE
    top4 <= 4
```

Top-N Query Example

ROW_NUMBER SOLUTION

LAST_NAME	DEPARTMENT_ID	SALARY	TOP4
Hunold	60	9000	1
Ernst	60	6000	2
Austin	60	4800	3
Pataballa	60	4800	4
Russell	80	14000	1
Partners	80	13500	2
Errazuriz	80	12000	3
Ozer	80	11500	4

DENSE_RANK function

- DENSE_RANK computes the rank of a row in an ordered group of rows. The ranks are consecutive integers beginning with 1. The largest rank value is the number of unique values returned by the query. Rank values are not skipped in the event of ties. Rows with equal values for the ranking criteria receive the same rank.

Top-N Query Example

DENSE_RANK SOLUTION

```
SELECT
    *
FROM
(
    SELECT
          department_id,
          last_name,
          salary,
          DENSE_RANK() OVER
                (PARTITION BY department_id
                ORDER BY salary DESC) AS top4
    FROM
          employees
)
WHERE
    top4 <= 4
```

Top-N Query Example

DENSE_RANK SOLUTION

LAST_NAME	DEPARTMENT_ID	SALARY	TOP4
Hunold	60	9000	1
Ernst	60	6000	2
Austin	60	4800	3
Pataballa	60	4800	3
Lorentz	60	4200	4
Russell	80	14000	1
Partners	80	13500	2
Errazuriz	80	12000	3
Ozer	80	11500	4

RANK function

- RANK calculates the rank of a value in a group of values. Rows with equal values for the ranking criteria receive the same rank. Oracle then adds the number of tied rows to the tied rank to calculate the next rank. Therefore, the ranks may not be consecutive numbers.

Top-N Query Example

RANK SOLUTION

```
SELECT
    *
FROM
(
    SELECT
          department_id,
          last_name,
          salary,
          RANK() OVER
                   (PARTITION BY department_id
                   ORDER BY salary DESC) AS top4
    FROM
          employees
)
WHERE
    top4 <= 4
```

Top-N Query Example

RANK SOLUTION

LAST_NAME	DEPARTMENT_ID	SALARY	TOP4
Hunold	60	9000	1
Ernst	60	6000	2
Austin	60	4800	3
Pataballa	60	4800	3
Russell	80	14000	1
Partners	80	13500	2
Errazuriz	80	12000	3
Ozer	80	11500	4

First and Last Rows

- The FIRST_VALUE and LAST_VALUE functions allow you to select the first and last rows from a group. These rows are especially valuable because they are often used as the baselines in calculations.

First Row Example

- **Find the employee with the lowest salary in each department**

LAST_NAME	DEPARTMENT_ID	SALARY
Hunold	60	9000
Ernst	60	6000
Austin	60	4800
Russell	80	14000
Partners	80	13500
Errazuriz	80	12000
Ozer	80	11500

First Row Example

```
SELECT
   department_id,
   last_name,
   salary,
   FIRST_VALUE(last_name) OVER
        (PARTITION BY department_id
        ORDER BY salary ASC) AS min_sal
FROM
   employees
```

First Row Example

LAST_NAME	DEPARTMENT_ID	SALARY	MIN_SAL
Hunold	60	9000	Austin
Ernst	60	6000	Austin
Austin	60	4800	Austin
Russell	80	14000	Ozer
Partners	80	13500	Ozer
Errazuriz	80	12000	Ozer
Ozer	80	11500	Ozer

Best Use for Me

- I can use the result of a grouping (aggregate) function within each record of a group – much more flexible, much less pain.
- I can perform relative ranking within a group – it used to be tortuous with "straight SQL"
- I can perform calculations in the SELECT clause based on neighboring row values.

Summary

- Analytic functions provide an easy mechanism to compute resultsets that, before 8i, were inefficient, impractical and, in some cases, impossible in "straight SQL".
- In addition to their flexibility and power, they are also extremely efficient.

Conclusion

- This new set of functionality holds some exciting possibilities. It opens up a whole new way of looking at the data. It will remove a lot of procedural code and complex or inefficient queries that would have taken a long time to develop.
- Add analytic functions to your SQL arsenal and actively seek opportunities to use them.

Where to Get More Information

- *Oracle 9i Data Warehousing Guide* – Oracle documentation, technet.oracle.com, March 2002
- *Oracle SQL Reference* – Oracle documentation, technet.oracle.com, October 2002

SQL Analytics

Lewis R Cunningham
Database Architect
Sheperd Systems

An Expert's Guide to Oracle
http://blogs.ittoolbox.com/oracle/guide

An expert is a person who has made all the mistakes that can be made in a very narrow field. - Niels Bohr (1885 - 1962)

SUNCOAST ORACLE® USERS' GROUP

Monthly
4th Thursday
6pm – 8pm

IBM Center
Rocky Point

WWW.SOUG.NET