Paper PO17

Oracle's RANK() Smells Good

Using RANK() in Pass-through Queries

Richard A. DeVenezia, Independent Consultant

Abstract

Proc SQL in SAS® software is a powerful tool. For many years SAS users have enjoyed an automatic self join that occurs when an groupwise aggregate function is used in a select clause. In plainer terms, you can group by one set of variables and select rows from the group using a different set of variables. The ORACLE function RANK() (PARTION BY ... ORDER BY ...) operates in a similar way. You can use this function in your pass-through queries to speed up processing and simplify the codebase that has to be maintained. A four way join will be shown demonstrating the utility of RANK().

Tables

Each table has a primary key named "ID", and may have columns named "{Table}_ID" that are foreign keyed to table {Table}. For example column B_ID would be foreign keyed to column ID of table B. The four tables are as follows:

Table A

Table A maintains a list of unique names and their corresponding ids.

```
create table a
( ID number not null
, NAME char(2)
, constraint a_pk primary key (id)
, constraint name_unique unique (name)
);
insert into a values(1,'A1');
insert into a values(2,'A2');
```

ID	NAME
1	A1
2	A2

Table B

Table B acts as an organizer; it lets the rows of C determine which A they belong to

```
create table b
( ID number not null
, A_ID number
, constraint b_pk primary key (id)
, constraint fk_ba foreign key (a_id) references a
);
insert into b values(11, 1);
insert into b values(12, 2);
insert into b values(13, 1);
insert into b values(14, 1);
```

ID	A_ID
11	1
12	2
13	1
14	1
15	1

```
insert into b values(15, 1);
```

Table D

Table D is a simple look up table.

```
create table d
( ID number not null
, NAME char(2)
, constraint d_pk primary key (id)
);
insert into d values (1, 'D1');
insert into d values (2, 'D2');
insert into d values (3, 'D3');
```

ID	NAME
1	D1
2	D2
3	D3

Table C

Table C records a transaction. The date is stored in column DATEX and some satellite information in columns V1 ... V35. For simplicity, the sample table has DATEX a number and the satellite columns are excluded. A date type would be used in a real word application.

```
create table c
( ID number not null
, B_ID number
, D_ID number
, DATEX number
, constraint c_pk primary key (id)
, constraint fk_cb foreign key (b_id) references b
, constraint fk_cd foreign key (d_id) references d
);
insert into c values (1, 11, 1, 41);
insert into c values (2, 11, 1, 40);
insert into c values (3, 11, 1, 43);
insert into c values (4, 11, 1, 42);
insert into c values (5, 12, 1, 55);
insert into c values (6, 12, 1, 38);
insert into c values (7, 12, 1, 65);
insert into c values (8, 13, 2, 43);
insert into c values (9, 13, 2, 42);
insert into c values (10, 13, 2, 41);
insert into c values (11, 14, 3, 16);
insert into c values (12, 14, 3, 18);
insert into c values (13, 15, 3, 15);
insert into c values (14, 15, 3, 19);
```

ID	B_ID	D_ID	DATEX
1	11	1	41
2	11	1	40
3	11	1	43
4	11	1	42
5	12	1	55
6	12	1	38
7	12	1	65
8	13	2	43
9	13	2	42
10	13	2	41
11	14	3	16
12	14	3	18
13	15	3	15
14	15	3	19

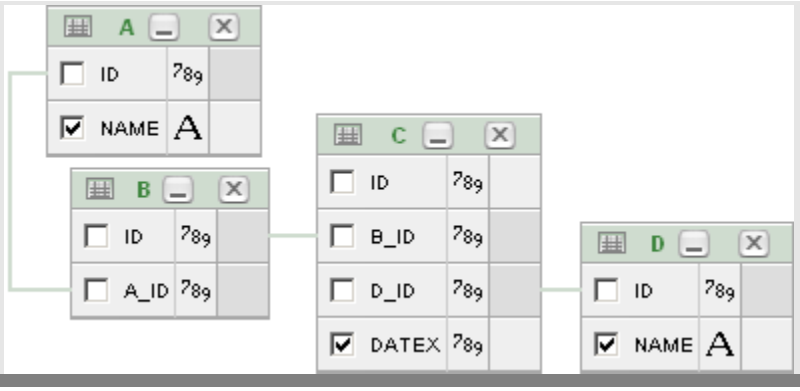
The Join

The objective is to pick one row from C (having the earliest DATEX) for each name of D that corresponds to a name of A.

Try 1

A first attempt gets close enough to see what is wanted.

```
select A.NAME A_NAME
     , C.*
     , D.NAME D_NAME
  from
       A,B,C,D
  where
       A.NAME = 'A1'
   and B.A_ID = A.ID
   and C.B_ID = B.ID
   and C.D_ID = D.ID
 order by D.ID, C.DATEX
```



The bold rows are the ones that meet the objective.

```
A_    ID  B_ID  D_ID DATEX D_
-- ---- ----- ----- ----- --
A1     2    11     1    40 D1
A1     1    11     1    41 D1
A1     4    11     1    42 D1
A1     3    11     1    43 D1

A1    10    13     2    41 D2
A1     9    13     2    42 D2
A1     8    13     2    43 D2

A1    13    15     3    15 D3
A1    11    14     3    16 D3
A1    12    14     3    18 D3
A1    14    15     3    19 D3
```

What query would select only these rows?

```
A1,  2, 11, 1, 40, D1
A1, 10, 13, 2, 41, D2
A1, 13, 15, 3, 15, D3
```

Try 2

A SAS software Proc SQL style query with auto-remerge (having C.DATEX = MIN(C.DATEX)) is tried. The query is not accepted by the Oracle parser and thus not acceptable for pass-through.

```
select A.NAME A_NAME
     , C.*
     , D.NAME D_NAME
 from
       A,B,C,D
 where
```

```
        A.NAME = 'A1'
    and B.A_ID = A.ID
    and C.B_ID = B.ID
    and C.D_ID = D.ID
  group by
        D.ID
  having
        C.DATEX = MIN(C.DATEX) ;
```

Oracle error message.

```
       C.DATEX = min(C.DATEX)
       *
ERROR at line 14:
ORA-00979: not a GROUP BY expression
```

Try 3

Update the Try 1 query with a new column based on the Oracle RANK function. DATEX values within each combination of a.name and d.name are ranked.

```
SELECT a.name a_name, c.*, d.name d_name
     , RANK () OVER
         ( PARTITION BY a.name, d.name
           ORDER BY datex
         ) AS rank
  FROM a, b, c, d
 WHERE a.name = 'A1'
   AND b.a_id = a.id
   AND c.b_id = b.id
   AND c.d_id = d.id;
```

The rows with the lowest DATEX (within group A_NAME, D_NAME) can now be easily identified by RANK=1.

```
A_  ID B_ID D_ID DATEX D_ RANK
-- -- ---- ---- ----- -- ----
A1  2   11    1    40 D1    1
A1  1   11    1    41 D1    2
A1  4   11    1    42 D1    3
A1  3   11    1    43 D1    4
A1 10   13    2    41 D2    1
A1  9   13    2    42 D2    2
A1  8   13    2    43 D2    3
A1 13   15    3    15 D3    1
A1 11   14    3    16 D3    2
A1 12   14    3    18 D3    3
A1 14   15    3    19 D3    4
```

Final query

The Try 3 query is used as a sub-query, and only the pertinent rows are selected.

```
SELECT *
FROM ( SELECT a.name a_name, c.*, d.name d_name
     , RANK () OVER
         ( PARTITION BY a.name, d.name
           ORDER BY datex
         ) AS rank
       FROM a, b, c, d
       WHERE b.a_id = a.id
         AND c.b_id = b.id
         AND c.d_id = d.id)
WHERE rank=1;
```

A_NAME	ID	B_ID	D_ID	DATEX	D_NAME	RANK
A1	2	11	1	40	D1	1
A1	10	13	2	41	D2	1
A1	13	15	3	15	D3	1
A2	6	12	1	38	D1	1

RANK is only one of many Analytic Functions introduced into Oracle at release 8.1.6. You can learn more about them at http://www.akadia.com/services/ora_analytic_functions.html and http://www.orafaq.com/node/55

SAS software can access remote DBMS tables using a SAS/ACCESS LIBNAME engine. Some automatic optimization is performed by the ACCESS engine, however, not all capabilities of the remote system are necessarily utilized. The author does not know if the ORACLE engine utilizes analytics functions in its optimizations.

The join demonstrated in this paper was used as a sub-query within a much larger query in a real world application. The names have been changed to protect the innocent.

Conclusion

Proc SQL offers the SAS software user the ability to submit a query to a remote systems using SQL dialects and features specific to that system. The ORACLE RDBMS has many features that can be taken advantage of when old programmers learn new tricks.

About the Author

Richard A. DeVenezia has previously presented papers at SUGI, SESUG and NESUG, and is an active contributor on SAS-L. He is an independent consultant with fifteen years of SAS experience. He has worked with an extensive mix of SAS products in a variety of industries, including manufacturing, retail and pharmaceutical companies.

This paper and others can be found at the author's website. Visit http://www.devenezia.com and follow the link to Papers.