Analytic Functions 101

Kim Berg Hansen
KiBeHa
Middelfart, Denmark

Keywords:

SQL, analytics

Introduction

The company I have worked at since 2000 as developer/architect is a major Danish retail company called T. Hansen Gruppen A/S selling primarily spare parts for cars via about 80 stores plus a webshop. (Readers from USA may think of us as similar to Pepboys, german readers as ATU.)

We have an old legacy ERP system running on top of an Oracle database in which we continuously keep on developing new functionality - partly in the legacy client language, partly in SQL and a bit of PL/SQL.

We have by now more than 1200 analytic statements throughout our source code - and it keeps growing. We do a lot of code in SQL statements and quite a lot of it we either could not easily do or at the least couldn't do as quickly and efficiently if we didn't have analytics.

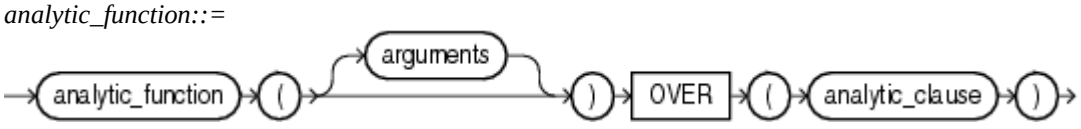
When you need your detail rows in the result set, but at the same time need to reference data from other rows - then is the case for analytics: Comparing one row with the total, running totals, distance from previous row, etc. etc.

Once you understand how analytics can retrieve data across rows in your result set and you start thinking in these terms, you will continually find places where you can solve your tasks easier with analytics than without.

Syntax

From the documentation we see the general syntax of an analytic function call:

analytic_function::=



analytic_clause::=

Let’s look at salaries in good old SCOTT.EMP table:

```
SQL> select deptno
  2       , ename
  3       , sal
  4    from scott.emp
  5  order by deptno
  6         , sal
  7  /

 DEPTNO ENAME    SAL
------- ------ -----
     10 MILLER  1300
     10 CLARK   2450
     10 KING    5000
     20 SMITH    800
     20 ADAMS   1100
     20 JONES   2975
     20 SCOTT   3000
     20 FORD    3000
     30 JAMES    950
     30 MARTIN  1250
     30 WARD    1250
     30 TURNER  1500
     30 ALLEN   1600
     30 BLAKE   2850
```

Simplest use of SUM analytic function would then be:

```
SQL> select deptno
  2       , ename
  3       , sal
  4       , sum(sal) over () sum_sal
  5    from scott.emp
  6  order by deptno
  7         , sal
  8  /

 DEPTNO ENAME    SAL   SUM_SAL
------- ------ ----- ---------
     10 MILLER  1300     29025
     10 CLARK   2450     29025
     10 KING    5000     29025
     20 SMITH    800     29025
     20 ADAMS   1100     29025
     20 JONES   2975     29025
     20 SCOTT   3000     29025
     20 FORD    3000     29025
     30 JAMES    950     29025
     30 MARTIN  1250     29025
     30 WARD    1250     29025
```

```
     30 TURNER  1500     29025
     30 ALLEN   1600     29025
     30 BLAKE   2850     29025
```

PARTITION, ORDER and WINDOW

Query partition clause is similar to group by in normal aggregate functions. We can get the sum by department:

```
SQL> select deptno
  2       , ename
  3       , sal
  4       , sum(sal) over (
  5            partition by deptno
  6         ) sum_sal
  7    from scott.emp
  8  order by deptno
  9         , sal
 10  /

 DEPTNO ENAME    SAL   SUM_SAL
------- ------ ----- ---------
     10 MILLER  1300      8750
     10 CLARK   2450      8750
     10 KING    5000      8750
     20 SMITH    800     10875
     20 ADAMS   1100     10875
     20 JONES   2975     10875
     20 SCOTT   3000     10875
     20 FORD    3000     10875
     30 JAMES    950      9400
     30 MARTIN  1250      9400
     30 WARD    1250      9400
     30 TURNER  1500      9400
     30 ALLEN   1600      9400
     30 BLAKE   2850      9400
```

Order by clause and windowing clause can be used to get for example incremental sums:

```
SQL> select deptno
  2       , ename
  3       , sal
  4       , sum(sal) over (
  5            order by sal
  6            rows between unbounded preceding and current row
  7         ) sum_sal
  8    from scott.emp
  9  order by sal
 10  /
```

```
 DEPTNO ENAME    SAL   SUM_SAL
------- ------ ----- ---------
     20 SMITH    800       800
     30 JAMES    950      1750
     20 ADAMS   1100      2850
     30 WARD    1250      4100
     30 MARTIN  1250      5350
     10 MILLER  1300      6650
     30 TURNER  1500      8150
     30 ALLEN   1600      9750
     10 CLARK   2450     12200
     30 BLAKE   2850     15050
     20 JONES   2975     18025
     20 SCOTT   3000     21025
     20 FORD    3000     24025
     10 KING    5000     29025
```

We ordered the result by the same as the analytic order by clause. We do not need to do that:

```
SQL> select deptno
  2       , ename
  3       , sal
  4       , sum(sal) over (
  5            order by sal
  6            rows between unbounded preceding and current row
  7         ) sum_sal
  8    from scott.emp
  9  order by deptno
 10         , sal
 11  /

 DEPTNO ENAME    SAL   SUM_SAL
------- ------ ----- ---------
     10 MILLER  1300      6650
     10 CLARK   2450     12200
     10 KING    5000     29025
     20 SMITH    800       800
     20 ADAMS   1100      2850
     20 JONES   2975     18025
     20 FORD    3000     24025
     20 SCOTT   3000     21025
     30 JAMES    950      1750
     30 WARD    1250      4100
     30 MARTIN  1250      5350
     30 TURNER  1500      8150
     30 ALLEN   1600      9750
     30 BLAKE   2850     15050
```

We can combine query partition clause with order by clause and windowing clause:

```
SQL> select deptno
  2       , ename
  3       , sal
  4       , sum(sal) over (
  5            partition by deptno
  6            order by sal
  7            rows between unbounded preceding and current row
  8         ) sum_sal
  9    from scott.emp
 10  order by deptno
 11         , sal
 12  /

 DEPTNO ENAME      SAL   SUM_SAL
------- ------ ----- ---------
     10 MILLER  1300      1300
     10 CLARK   2450      3750
     10 KING    5000      8750
     20 SMITH    800       800
     20 ADAMS   1100      1900
     20 JONES   2975      4875
     20 SCOTT   3000      7875
     20 FORD    3000     10875
     30 JAMES    950       950
     30 MARTIN  1250      2200
     30 WARD    1250      3450
     30 TURNER  1500      4950
     30 ALLEN   1600      6550
     30 BLAKE   2850      9400
```

The windowing clause also allows us to specify “windows” of data in other ways:

```
SQL> select deptno
  2       , ename
  3       , sal
  4       , sum(sal) over (
  5            partition by deptno
  6            order by sal
  7            rows between current row and unbounded following
  8         ) sum_sal
  9    from scott.emp
 10  order by deptno
 11         , sal
 12  /
```

```
DEPTNO ENAME    SAL   SUM_SAL
------- ------ ----- ---------
     10 MILLER  1300      8750
     10 CLARK   2450      7450
     10 KING    5000      5000
     20 SMITH    800     10875
     20 ADAMS   1100     10075
     20 JONES   2975      8975
     20 SCOTT   3000      6000
     20 FORD    3000      3000
     30 JAMES    950      9400
     30 MARTIN  1250      8450
     30 WARD    1250      7200
     30 TURNER  1500      5950
     30 ALLEN   1600      4450
     30 BLAKE   2850      2850
```

You can if you wish specify a completely "unbounded" window:

```
SQL> select deptno
  2       , ename
  3       , sal
  4       , sum(sal) over (
  5            partition by deptno
  6            order by sal
  7            rows between unbounded preceding and unbounded
following
  8         ) sum_sal
  9    from scott.emp
 10  order by deptno
 11         , sal
 12  /

 DEPTNO ENAME    SAL   SUM_SAL
------- ------ ----- ---------
     10 MILLER  1300      8750
     10 CLARK   2450      8750
     10 KING    5000      8750
     20 SMITH    800     10875
     20 ADAMS   1100     10875
     20 JONES   2975     10875
     20 SCOTT   3000     10875
     20 FORD    3000     10875
     30 JAMES    950      9400
     30 MARTIN  1250      9400
     30 WARD    1250      9400
     30 TURNER  1500      9400
     30 ALLEN   1600      9400
     30 BLAKE   2850      9400
```

Which is same as just using the query partition clause without any order by and window clause.

You can specify a specific number of rows preceding and/or following:

```
SQL> select deptno
  2       , ename
  3       , sal
  4       , sum(sal) over (
  5            partition by deptno
  6            order by sal
  7            rows between 1 preceding and 1 following
  8         ) sum_sal
  9    from scott.emp
 10  order by deptno
 11         , sal
 12  /

 DEPTNO ENAME      SAL    SUM_SAL
------- ------ ------- ---------
     10 MILLER   1300       3750
     10 CLARK    2450       8750
     10 KING     5000       7450
     20 SMITH     800       1900
     20 ADAMS    1100       4875
     20 JONES    2975       7075
     20 SCOTT    3000       8975
     20 FORD     3000       6000
     30 JAMES     950       2200
     30 MARTIN   1250       3450
     30 WARD     1250       4000
     30 TURNER   1500       4350
     30 ALLEN    1600       5950
     30 BLAKE    2850       4450
```

RANGE versus ROWS

Alternative to using ROWS BETWEEN is RANGE BETWEEN.
For example sum of those salaries that are “the same plus/minus 500”:

```
SQL> select deptno
  2       , ename
  3       , sal
  4       , sum(sal) over (
  5            partition by deptno
  6            order by sal
  7            range between 500 preceding and 500 following
  8         ) sum_sal
  9    from scott.emp
 10  order by deptno
 11         , sal
 12  /
```

```
 DEPTNO ENAME    SAL   SUM_SAL
------- ------ ----- ---------
     10 MILLER  1300      1300
     10 CLARK   2450      2450
     10 KING    5000      5000
     20 SMITH    800      1900
     20 ADAMS   1100      1900
     20 JONES   2975      8975
     20 SCOTT   3000      8975
     20 FORD    3000      8975
     30 JAMES    950      3450
     30 MARTIN  1250      6550
     30 WARD    1250      6550
     30 TURNER  1500      5600
     30 ALLEN   1600      5600
     30 BLAKE   2850      2850
```

UNBOUNDED can also be used with RANGE - like a sum of salaries smaller than or equal to:

```
SQL> select deptno
  2       , ename
  3       , sal
  4       , sum(sal) over (
  5            partition by deptno
  6            order by sal
  7            range between unbounded preceding and current row
  8         ) sum_sal
  9    from scott.emp
 10  order by deptno
 11         , sal
 12  /

 DEPTNO ENAME    SAL   SUM_SAL
------- ------ ----- ---------
     10 MILLER  1300      1300
     10 CLARK   2450      3750
     10 KING    5000      8750
     20 SMITH    800       800
     20 ADAMS   1100      1900
     20 JONES   2975      4875
     20 SCOTT   3000     10875
     20 FORD    3000     10875
     30 JAMES    950       950
     30 MARTIN  1250      3450
     30 WARD    1250      3450
     30 TURNER  1500      4950
     30 ALLEN   1600      6550
     30 BLAKE   2850      9400
```

Notice the result for SCOTT and MARTIN - see how this is different from the ROWS BETWEEN earlier:

```
SQL> select deptno
  2       , ename
  3       , sal
  4       , sum(sal) over (
  5            partition by deptno
  6            order by sal
  7            rows between unbounded preceding and current row
  8         ) sum_sal
  9    from scott.emp
 10  order by deptno
 11         , sal
 12  /

 DEPTNO ENAME    SAL   SUM_SAL
------- ------ ----- ---------
     10 MILLER  1300      1300
     10 CLARK   2450      3750
     10 KING    5000      8750
     20 SMITH    800       800
     20 ADAMS   1100      1900
     20 JONES   2975      4875
     20 SCOTT   3000      7875
     20 FORD    3000     10875
     30 JAMES    950       950
     30 MARTIN  1250      2200
     30 WARD    1250      3450
     30 TURNER  1500      4950
     30 ALLEN   1600      6550
     30 BLAKE   2850      9400
```

If you omit the windowing clause, but you have an order by clause, then the default is a RANGE window:

```
SQL> select deptno
  2       , ename
  3       , sal
  4       , sum(sal) over (
  5            partition by deptno
  6            order by sal
  7         ) sum_sal
  8    from scott.emp
  9  order by deptno
 10         , sal
 11  /

 DEPTNO ENAME    SAL   SUM_SAL
------- ------ ----- ---------
     10 MILLER  1300      1300
     10 CLARK   2450      3750
     10 KING    5000      8750
```

```
    20 SMITH    800        800
    20 ADAMS   1100       1900
    20 JONES   2975       4875
    20 SCOTT   3000      10875
    20 FORD    3000      10875
    30 JAMES    950        950
    30 MARTIN  1250       3450
    30 WARD    1250       3450
    30 TURNER  1500       4950
    30 ALLEN   1600       6550
    30 BLAKE   2850       9400
```

This is like the example with RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW.

Very often we use a ROWS BETWEEN window and if we are not careful we might end up involuntarily using RANGE BETWEEN simply by relying on the defaults.

My rule of thumb is, that if I have no ORDER BY, then fine - it will default to BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING. But if I have an ORDER BY clause in my analytic function call, I make it a habit to always explicitly write the windowing clause as well - even if it is the default. That way I am sure always to do "the right thing" :-)

ORDER BY uniqueness

Another thing to keep in mind is that when you do use a windowing clause then it is often a good idea to make sure the order by clause is "unique" - otherwise the result could be indeterminate.

Let's look at the incremental sum again:

```
SQL> select deptno
  2       , empno
  3       , ename
  4       , sal
  5       , sum(sal) over (
  6            partition by deptno
  7            order by sal
  8            rows between unbounded preceding and current row
  9         ) sum_sal
 10    from scott.emp
 11  order by deptno
 12         , sal
 13  /

 DEPTNO  EMPNO ENAME      SAL     SUM_SAL
------- ------ ------ ------ -----------
     10   7934 MILLER   1300        1300
     10   7782 CLARK    2450        3750
     10   7839 KING     5000        8750
     20   7369 SMITH     800         800
     20   7876 ADAMS    1100        1900
```

```
      20   7566 JONES    2975         4875
      20   7788 SCOTT    3000         7875
      20   7902 FORD     3000        10875
      30   7900 JAMES     950          950
      30   7654 MARTIN   1250         2200
      30   7521 WARD     1250         3450
      30   7844 TURNER   1500         4950
      30   7499 ALLEN    1600         6550
      30   7698 BLAKE    2850         9400
```

That Scott comes before Ford and Martin comes before Ward is "chance" - you cannot rely on it.

Oracle tries to do as little work as possible (just as lazy as us developers :-) so when you do not specify how Oracle should "break ties" it will return the data in whatever order it happens to retrieve it without any extra sorting. That will depend on the access plan used - if an index scan is used you may get lucky and get it in the order you wish, if a hash join is used you probably won't.

So usually I always make sure the order by is "unique" to make the output deterministic. Even if the end user has specified "it doesn't matter", he is likely to be concerned if the output changes between runs of the report :-)

```
SQL> select deptno
  2       , empno
  3       , ename
  4       , sal
  5       , sum(sal) over (
  6            partition by deptno
  7            order by sal, empno
  8            rows between unbounded preceding and current row
  9         ) sum_sal
 10    from scott.emp
 11  order by deptno
 12         , sal
 13  /

 DEPTNO  EMPNO ENAME      SAL     SUM_SAL
------- ------ ------ ------ -----------
     10   7934 MILLER   1300        1300
     10   7782 CLARK    2450        3750
     10   7839 KING     5000        8750
     20   7369 SMITH     800         800
     20   7876 ADAMS    1100        1900
     20   7566 JONES    2975        4875
     20   7788 SCOTT    3000        7875
     20   7902 FORD     3000       10875
     30   7900 JAMES     950         950
     30   7521 WARD     1250        2200
     30   7654 MARTIN   1250        3450
     30   7844 TURNER   1500        4950
     30   7499 ALLEN    1600        6550
     30   7698 BLAKE    2850        9400
```

Notice how this time Ward comes before Martin because I explicitly stated that employees with same salary should be ordered by empno. (And actually I ought to order the result by deptno, sal, empno as well - here I am lucky that Oracle is lazy and since it already did order the data by empno for the analytic clause, it just keeps that order :-)

Case/Demo: Top selling items

A classic task given to a programmer is to make a TOP-N report of some data. Often a TOP within each group of some defined grouping (department, country, product type, etc.) And many times the report should also include the percentage of the total - even though you only display the TOP records.

Many ways can be devised for this and many ways has been used and blogged about and given lectures on. This is nothing new, but it is a technique we often use in many practical cases. One typical case is a top list of best selling items within product groups.

We create a couple of tables first. A table of items with a column grp to identify the product group the item belongs to, and a table of sales history. For this demo the sales table stores sales history by the month, but the technique would be identical if it were weekly or daily data or even if it were individual order lines.

```
SQL> create table items(
  2   item  varchar2(10) primary key,
  3   grp   varchar2(10),
  4   name  varchar2(20)
  5  )
  6  /

SQL> create table sales (
  2   item  varchar2(10) references items (item),
  3   mth   date,
  4   qty   number
  5  )
  6  /
```

We populate the tables with 10 typical items for our shop - 5 spare parts for cars, 5 accessories for mobile phones.

```
SQL> begin
  2    insert into items values ('101010','AUTO','Brake disc');
  3    insert into items values ('102020','AUTO','Snow chain');
  4    insert into items values ('103030','AUTO','Sparc plug');
  5    insert into items values ('104040','AUTO','Oil filter');
  6    insert into items values ('105050','AUTO','Light bulb');
  7
  8    insert into items values ('201010','MOBILE','Handsfree');
  9    insert into items values ('202020','MOBILE','Charger');
 10    insert into items values ('203030','MOBILE','iGloves');
```

```
 11    insert into items values ('204040','MOBILE','Headset');
 12    insert into items values ('205050','MOBILE','Cover');
 13
 14    insert into sales values ('101010',date '2011-04-01',10);
 15    insert into sales values ('101010',date '2011-05-01',11);
 16    insert into sales values ('101010',date '2011-06-01',12);
 17    insert into sales values ('102020',date '2011-03-01', 7);
 18    insert into sales values ('102020',date '2011-07-01', 8);
 19    insert into sales values ('103030',date '2011-01-01', 6);
 20    insert into sales values ('103030',date '2011-02-01', 9);
 21    insert into sales values ('103030',date '2011-11-01', 4);
 22    insert into sales values ('103030',date '2011-12-01',14);
 23    insert into sales values ('104040',date '2011-08-01',22);
 24    insert into sales values ('105050',date '2011-09-01',13);
 25    insert into sales values ('105050',date '2011-10-01',15);
 26
 27    insert into sales values ('201010',date '2011-04-01', 5);
 28    insert into sales values ('201010',date '2011-05-01', 6);
 29    insert into sales values ('201010',date '2011-06-01', 7);
 30    insert into sales values ('202020',date '2011-03-01',21);
 31    insert into sales values ('202020',date '2011-07-01',23);
 32    insert into sales values ('203030',date '2011-01-01', 7);
 33    insert into sales values ('203030',date '2011-02-01', 7);
 34    insert into sales values ('203030',date '2011-11-01', 6);
 35    insert into sales values ('203030',date '2011-12-01', 8);
 36    insert into sales values ('204040',date '2011-08-01',35);
 37    insert into sales values ('205050',date '2011-09-01',13);
 38    insert into sales values ('205050',date '2011-10-01',15);
 39
 40    commit;
 41  end;
 42  /
```

Now let's see the result of an ordinary aggregate group by for sales data for the year 2011:

```
SQL> select i.grp
  2     , i.item
  3     , max(i.name) name
  4     , sum(s.qty) qty
  5    from items i
  6    join sales s
  7     on s.item = i.item
  8    where s.mth between date '2011-01-01' and date '2011-12-01'
  9    group by i.grp, i.item
 10    order by i.grp, sum(s.qty) desc, i.item
 11  /
```

```
GRP        ITEM       NAME                          QTY
---------- ---------- -------------------- ----------
AUTO       101010     Brake disc                    33
AUTO       103030     Sparc plug                    33
AUTO       105050     Light bulb                    28
AUTO       104040     Oil filter                    22
AUTO       102020     Snow chain                    15
MOBILE     202020     Charger                       44
MOBILE     204040     Headset                       35
MOBILE     203030     iGloves                       28
MOBILE     205050     Cover                         28
MOBILE     201010     Handsfree                     18
```

Ordering by the sum shows us which items are the top three selling items in each group. Let us see if we cannot do the same with analytic functions to get a TOP-3 report.

RANK versus DENSE_RANK versus ROW_NUMBER

First let's try using the aggregate query as source for three different analytic functions - DENSE_RANK(), RANK() and ROW_NUMBER(). Each of them we do PARTITION BY in order to get a ranking within each produkt group and we use ORDER BY to specify the ranking order:

```
SQL> select g.grp
  2     , g.item
  3     , g.name
  4     , g.qty
  5     , dense_rank() over
                (partition by g.grp order by g.qty desc) drnk
  6     , rank() over (partition by g.grp order by g.qty desc) rnk
  7     , row_number() over
                (partition by g.grp order by g.qty desc, g.item) rnum
  8    from (
  9   select i.grp
 10        , i.item
 11        , max(i.name) name
 12        , sum(s.qty) qty
 13     from items i
 14     join sales s
 15       on s.item = i.item
 16    where s.mth between date '2011-01-01' and date '2011-12-01'
 17    group by i.grp, i.item
 18  ) g
 19   order by g.grp, g.qty desc, g.item
 20  /

GRP     ITEM    NAME                      QTY    DRNK     RNK    RNUM
------- ------- -------------------- ------- ------- ------- -------
AUTO    101010  Brake disc                33       1       1       1
AUTO    103030  Sparc plug                33       1       1       2
AUTO    105050  Light bulb                28       2       3       3
```

```
AUTO    104040  Oil filter                 22       3       4       4
AUTO    102020  Snow chain                 15       4       5       5
MOBILE  202020  Charger                    44       1       1       1
MOBILE  204040  Headset                    35       2       2       2
MOBILE  203030  iGloves                    28       3       3       3
MOBILE  205050  Cover                      28       3       3       4
MOBILE  201010  Handsfree                  18       4       5       5
```

Notice the differences in the three ranking functions:

- DENSE_RANK gives the same rank to items having the same value - brake disc and sparc plug both get 1, light bulb gets 2.
- RANK does the same, but it works like olympic medals: if there are two gold medals then there is no silver medal, the next one gets bronze - here light bulb gets 3.
- ROW_NUMBER never gives the same rank, it will always give 1, 2, 3, 4 … If two items have the same value it will be "random" which one gets to be first - here brake disc gets 1 and sparc plug gets 2, but that is because we gave the "ORDER BY g.qty desc, g.item", this tells Oracle that if there is a tie for g.qty, then use item id for ordering and breaking the tie. For ROW_NUMBER it is good practice to always use an ORDER BY clause that ensures consistent ordering - otherwise you might get different results in different runs of the query.

You can get the same result as above without using an inline view. It is possible simply to use the analytic functions directly on the aggregations:

```
SQL> select i.grp
  2     , i.item
  3     , max(i.name) name
  4     , sum(s.qty) qty
  5     , dense_rank() over
           (partition by i.grp order by sum(s.qty) desc) drnk
  6     , rank() over
           (partition by i.grp order by sum(s.qty) desc) rnk
  7     , row_number() over
           (partition by i.grp order by sum(s.qty) desc, i.item) rnum
  8    from items i
  9    join sales s
 10    on s.item = i.item
 11   where s.mth between date '2011-01-01' and date '2011-12-01'
 12   group by i.grp, i.item
 13   order by i.grp, sum(s.qty) desc, i.item
 14  /

GRP     ITEM    NAME                     QTY     DRNK      RNK     RNUM
------- ------- -------------------- ------- ------- ------- -------
AUTO    101010  Brake disc                33       1       1       1
AUTO    103030  Sparc plug                33       1       1       2
AUTO    105050  Light bulb                28       2       3       3
AUTO    104040  Oil filter                22       3       4       4
AUTO    102020  Snow chain                15       4       5       5
MOBILE  202020  Charger                   44       1       1       1
```

```
MOBILE  204040  Headset                   35       2        2        2
MOBILE  203030  iGloves                   28       3        3        3
MOBILE  205050  Cover                     28       3        3        4
MOBILE  201010  Handsfree                 18       4        5        5
```

The effect is the same - the aggregation is done first and then the analytic functions are applied. Sometimes it can be nice to do the inline view anyway for readability and to make it clear what happens, but for simpler cases like this I tend to prefer the compact method. One of the reasons is that often you need to put the entire thing in another inline view anyway in order to filter the result, and then you can get confused by too many inline views inside inline views :-)

RANK for TOP-N report

The analytic functions cannot be used directly in a where clause, so when we wish to filter on the ranking function in order to get our TOP-3 report, we need an inline view:

```
SQL> select g.grp
  2     , g.item
  3     , g.name
  4     , g.qty
  5     , g.rnk
  6    from (
  7   select i.grp
  8        , i.item
  9        , max(i.name) name
 10        , sum(s.qty) qty
 11        , rank() over
                (partition by i.grp order by sum(s.qty) desc) rnk
 12     from items i
 13     join sales s
 14       on s.item = i.item
 15    where s.mth between date '2011-01-01' and date '2011-12-01'
 16    group by i.grp, i.item
 17  ) g
 18   where g.rnk <= 3
 19   order by g.grp, g.rnk, g.item
 20  /

GRP        ITEM       NAME                      QTY        RNK
---------- ---------- -------------------- ---------- ----------
AUTO       101010     Brake disc                33          1
AUTO       103030     Sparc plug                33          1
AUTO       105050     Light bulb                28          3
MOBILE     202020     Charger                   44          1
MOBILE     204040     Headset                   35          2
MOBILE     203030     iGloves                   28          3
MOBILE     205050     Cover                     28          3
```

Here we used RANK and thus we got three AUTO items and four MOBILE items.

Let us try DENSE_RANK:

```
SQL> select g.grp
  2     , g.item
  3     , g.name
  4     , g.qty
  5     , g.rnk
  6    from (
  7   select i.grp
  8        , i.item
  9        , max(i.name) name
 10        , sum(s.qty) qty
 11        , dense_rank() over
                (partition by i.grp order by sum(s.qty) desc) rnk
 12     from items i
 13     join sales s
 14       on s.item = i.item
 15    where s.mth between date '2011-01-01' and date '2011-12-01'
 16    group by i.grp, i.item
 17  ) g
 18   where g.rnk <= 3
 19   order by g.grp, g.rnk, g.item
 20  /

GRP        ITEM       NAME                      QTY        RNK
---------- ---------- -------------------- ---------- ----------
AUTO       101010     Brake disc                33          1
AUTO       103030     Sparc plug                33          1
AUTO       105050     Light bulb                28          2
AUTO       104040     Oil filter                22          3
MOBILE     202020     Charger                   44          1
MOBILE     204040     Headset                   35          2
MOBILE     203030     iGloves                   28          3
MOBILE     205050     Cover                     28          3
```

That gave us four items in each group.

So how about ROW_NUMBER:

```
SQL> select g.grp
  2     , g.item
  3     , g.name
  4     , g.qty
  5     , g.rnk
  6    from (
  7   select i.grp
  8        , i.item
  9        , max(i.name) name
 10        , sum(s.qty) qty
```

```
 11         , row_number() over (partition by
                    i.grp order by sum(s.qty) desc, i.item) rnk
 12      from items i
 13      join sales s
 14        on s.item = i.item
 15     where s.mth between date '2011-01-01' and date '2011-12-01'
 16     group by i.grp, i.item
 17  ) g
 18   where g.rnk <= 3
 19   order by g.grp, g.rnk, g.item
 20  /

GRP        ITEM       NAME                      QTY        RNK
---------- ---------- -------------------- ---------- ----------
AUTO       101010     Brake disc                 33          1
AUTO       103030     Sparc plug                 33          2
AUTO       105050     Light bulb                 28          3
MOBILE     202020     Charger                    44          1
MOBILE     204040     Headset                    35          2
MOBILE     203030     iGloves                    28          3
```

That gave us exactly three items in each group. Notice here that even though both iGloves and Cover sold 28 pieces and ties for third place in group MOBILE, we only get iGloves here. That is because we told Oracle to resolve ties by ordering by item id and 203030 is less than 205050. If we had omitted the ", i.item" part of the ORDER BY for ROW_NUMBER(), then it would have been random whether we got iGloves or Cover.

RATIO_TO_REPORT

Along with the quantity sold and the rank, we would also like to know how big a percentage of the total sales each item had. And we would like that information both within the product groups, but at the same time also how big percentage of the grand total sold.

For that we use RATIO_TO_REPORT which returns the ratio of the particular row to the total (then we multiply the ratio by 100 to get percent.)

If we use the PARTITION BY clause, then we can get a percentage within the group. If we omit the PARTITION BY, then we get a percentage of the grand total:

```
SQL> select g.grp
  2     , g.item
  3     , g.name
  4     , g.qty
  5     , g.rnk
  6     , round(g.g_pct,1) g_pct
  7     , round(g.t_pct,1) t_pct
  8    from (
  9   select i.grp
```

```
 10         , i.item
 11         , max(i.name) name
 12         , sum(s.qty) qty
 13         , rank() over
                  (partition by i.grp order by sum(s.qty) desc) rnk
 14         , 100 * ratio_to_report(sum(s.qty)) over
                  (partition by i.grp) g_pct
 15         , 100 * ratio_to_report(sum(s.qty)) over () t_pct
 16      from items i
 17      join sales s
 18        on s.item = i.item
 19     where s.mth between date '2011-01-01' and date '2011-12-01'
 20     group by i.grp, i.item
 21   ) g
 22    where g.rnk <= 3
 23    order by g.grp, g.rnk, g.item
 24   /
```

GRP	ITEM	NAME	QTY	RNK	G_PCT	T_PCT
AUTO	101010	Brake disc	33	1	25.2	11.6
AUTO	103030	Sparc plug	33	1	25.2	11.6
AUTO	105050	Light bulb	28	3	21.4	9.9
MOBILE	202020	Charger	44	1	28.8	15.5
MOBILE	204040	Headset	35	2	22.9	12.3
MOBILE	203030	iGloves	28	3	18.3	9.9
MOBILE	205050	Cover	28	3	18.3	9.9

Here we see that light bulb was 21.4% of the total sales in the AUTO group and 9.9% of the grand total sales.

This is technique we use often in many of our reports. Sometimes (as this case) we combine aggregation with analytics, where the aggregation creates the set of data that the analytic functions then work on. Sometimes the data is ready for use by the analytic functions and no aggregation is needed. That depends on the actual case whether you need aggregation combined with analytics.

Conclusion

Just start using analytic functions!

When you get a task, have the analytic way of thinking in your mind as part of your tool set - you will not always have a case for it, but over time your own case list of analytic use will grow as you get used to it. The more you use it, the more you also will find your cases for it (as with any tool.)

When you start to think you need to process your data procedurally – think again! There is a good chance you can use the power of SQL to let the database do the hard work processing data. That's what the database does best, and you're paying for it so why not use it :-)

Good luck with analytics.

Author

Kim Berg Hansen

KIBEHA
Middelfart, Denmark

Email: kibeha@kibeha.dk
Internet: http://www.kibeha.dk