# 8.分析函数

2012年1月25日　　20:07

高晅

通过使用分析函数，可以不使用任何自联接就能够在一行中取出聚合的和未经聚合的值。考虑获取一个员工的薪水，同时将部门平均工资和所在地平均工资放在一行中，将会需要对employees表进行多次自联接。可以使用分析函数来写这个查询而不需要任何自联接。

分析函数有时候被称为窗口函数。分析函数以一定的方法在一个与当前行相关的结果集子集中进行计算。这个子集可以被称为窗口。

```
drop table sales_fact;
创建一个事实表，模拟销售记录：国家、地区、产品、年、周、售价、收入。包含1998到2001年的销售数据。
```

```
CREATE table sales_fact AS
SELECT country_name country, country_subRegion region, prod_name product, calendar_year year,
       calendar_week_number week, SUM(amount_sold) sale,
       sum(amount_sold * (case
              when mod(rownum, 10) = 0 then 1.4
              when mod(rownum, 5) = 0 then  0.6
              when mod(rownum, 2) = 0 then  0.9
              when mod(rownum, 2) = 1 then  1.2
              else 1
           end)) receipts
  FROM sh.sales, sh.times,sh.customers, sh.countries, sh.products
 WHERE sales.time_id = times.time_id
   AND sales.prod_id = products.prod_id
   AND sales.cust_id = customers.cust_id
   AND customers.country_id = countries.country_id
 GROUP BY country_name, country_subRegion, prod_name, calendar_year, calendar_week_number;
```

## 分析函数剖析

分析函数有三个基本组成部分：分区子句，排序子句及开窗子句。

function1 (argument1, argument2, ...argumentN)  --分析函数

  over([partition-by-clause] [order-by-clause] [windowing-clause] )  --三个子句

**function1是所调用的接受0个或多个参数的分析函数。分区子句按照分区列的值对数据进行分组。所有分区列的值相同的数据行被组合成为一个分区。**

1. 从操作上来说，数据行按照分区列进行排序并被分为数据分区。例如，SQL子句partition by product,country。使用Product列和Country列的值进行分区。数据行按照Product和Country两列的值进行排序并按照产品和国别的组合来进行分区。
2. 排序子句通过一列或者一个表达式的值来对数据分区中的行进行排序。在一个分析型的SQL语句中，一个数据行在数据分区中的位置是很重要的，并区是由排序子句来控制的。在一个数据分区内的数据行按照排序列的值进行排序。因为在分区子句中按照分区的值来进行排序，实际上最终得到的是按照分区子句和排序子句中指定的列来进行排序后的结果。
   排序可以是升序或降序。使用NULLS FIRST或NULLS LAST子句可以将空值放到数据分区的最上面和最下面。
3. **开窗子句指定了分析函数进行运算的数据子集**。这个窗口可以是动态的并且被很恰当地成为滑动窗口。你可以使用窗口说明子句来指定滑动窗口的上下边界条件。窗口说明子句的语法如下：
   [rows | range] between <start expr> and <end expr>
   whereas
   <start expr> is [unbounded preceding | current row| n preceding | n following]
   <end exprt>is [unbounded following | current row| n preceding | n following]
   关键字preceding指定了窗口的上边界条件。following或current row子句指定了窗口的下边界条件。滑动窗口提供了简便的复杂矩阵计算能力。例如，你可以使用子句rows between unbounded preceding and current row来对sale列动态求和。在这个例子中，窗口的上面的一行是当前分区中的第一行而窗口最下面一行是当前数据行。(unbounded preceding最早的到 current row当前行 )

   并不是所有的分析函数都支持开窗子句。

分析函数不能进行嵌套。但可以通过将所包含的SQL语句放在内嵌视图中，然后在视图外使用分析函数来实现嵌套效果。分析函数可以被用在多层嵌套内嵌视图中。

## 函数列表

| | 函数 | 描述 | |
|---|---|---|---|
| | lag | 访问一个分区或结果集中之前的一行。 | |
| | lead | 访问一个分区或结果集中之后的一行。 | |
| | first_value | 访问一个分区或结果集中第一行。 | |
| | last_value | 访问一个分区或结果集中最后一行。 | |
| | **nth_value** | 访问一个分区或结果集中的任意一行。 | |
| | rank | 将数据行值按照排序后的顺序进行排名，在有并列的情况下排名值将被跳过。 | 不可开窗，跳过排名 |
| | **dense_rank** | 将数据行值按照排序后的顺序进行排名，在有并列的情况下也不跳过排名值。 | |
| | **row_number** | 对进行排序并每一行增加一个唯一编号。这是一个非确定性函数。 | |
| | **ratio_to_report** | 计算报告中值的比例 | |
| | percent_rank | 将计算得到的排名值标准化为0到1之间的值 | |
| | percentile_cont | 取出与指定的排名百分比相匹配的值。是percent_rank的反函数 | |
| | percentile_dist | 取出与指定的排名百分比相匹配的值。采用谨慎分布模型 | |
| | **ntile** | 将数据行分组为单元 | |
| | listagg | 将来自不同行的列转化为列表格式 | |

## 聚合函数

计算年初到当前周这个范围(rows between，即第二周的sum为第一周+第二周，以此类推)：
rows between unbounded preceding and current row
求sum(sale)，根据product, country, region, year进行分区（合并这些值），根据week进行排序。
这里由于指定了country/product，并且country与region基本相同，因此只有year/week/sale数据有区别。

```
create index sf_country_product on sales_fact(COUNTRY,PRODUCT)
```

```
SELECT YEAR, WEEK, SALE,
       SUM(SALE) OVER(PARTITION BY PRODUCT, COUNTRY, REGION, YEAR ORDER BY WEEK ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
RUNNING_SUM_YTD
  FROM SALES_FACT
 WHERE COUNTRY IN ('Australia')
   AND PRODUCT = '256MB Memory Card'
 ORDER BY PRODUCT, COUNTRY, YEAR, WEEK;
```

| YEAR | WEEK | SALE | RUNNING_SUM_YTD |
|---|---|---|---|
| 1999 | 45 | 176.98 | 8675.52 |
| 1999 | 46 | 178.86 | 8854.38 |
| 1999 | 47 | 355.24 | 9209.62 |
| 2000 | 11 | 310.44 | 310.44 |
| 2000 | 12 | 155.22 | 465.66 |
| 2000 | 13 | 157.4 | 623.06 |

而如果是PARTITION BY PRODUCT, COUNTRY, REGION ORDER BY YEAR, WEEK，则累计历史年份的数据。

```
SELECT YEAR, WEEK, SALE,
       SUM(SALE) OVER(PARTITION BY PRODUCT, COUNTRY, REGION ORDER BY YEAR, WEEK ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
RUNNING_SUM_YTD
  FROM SALES_FACT
 WHERE COUNTRY IN ('Australia')
   AND PRODUCT = '256MB Memory Card'
 ORDER BY PRODUCT, COUNTRY, YEAR, WEEK;
```

| | | | |
|---|---|---|---|
| 1999 | 45 | 176.98 | 8675.52 |
| 1999 | 46 | 178.86 | 8854.38 |
| 1999 | 47 | 355.24 | 9209.62 |
| 2000 | 11 | 310.44 | 9520.06 |
| 2000 | 12 | 155.22 | 9675.28 |
| 2000 | 13 | 157.4 | 9832.68 |

```
create table hd.test_part (rankid int,percent numeric(12,2);

insert into hd.test_part(rankid,percent) values(1,10);
insert into hd.test_part(rankid,percent) values(2,5);
insert into hd.test_part(rankid,percent) values(3,15);


select rankid,percent,row_number() over (
        -- partition by rankid
        order by percent desc
    ) as sort_number,
sum(percent) over (
        -- partition by rankid
        order by percent desc
        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
    ) as aspercent
from hd.test_part;
```

## 跨越整个分区的聚合函数

跨分区的计算：rows between unbounded preceding and unbounded following
从整个分区partition by product, country, region, year，求商品、国家、地区、年份分区中的最大值，包括未来数据unbounded following。

```
select year, week, sale,
       max(sale) over(partition by product, country, region, year order by week rows between unbounded preceding and
unbounded following) max_sale_ytd
  from sales_fact
 where country in ('Australia')
   and product = 'Xtend Memory'
 order by product, country, year, week;
```

| YEAR | WEEK | SALE | MAX_SALE_YTD |
|---|---|---|---|
| 1998 | 46 | 57.52 | 172.56 |
| 1998 | 47 | 57.72 | 172.56 |
| 1998 | 48 | 172.56 | 172.56 |
| 1998 | 50 | 28.76 | 172.56 |
| 1998 | 51 | 58.32 | 172.56 |

## 细粒度窗口声明

**计算本周（当前行）前2周到本周后2周的滑动时间窗口。求当前行的上下两行的最大值。**
rows between 2 preceding and 2 following

```
select year, week, sale,
       max(sale) over(partition by product, country, region, year order by year, week rows between 2 preceding and 2
following) max_weeks_5
```

```
  from sales_fact
 where country in ('Australia')
   and product = 'Xtend Memory'
 order by product, country, year, week;
```

| YEAR | WEEK | SALE | MAX_WEEKS_5 |
|---|---|---|---|
| 1998 | 1 | 58.15 | 58.15 |
| 1998 | 2 | 29.39 | 58.15 |
| 1998 | 3 | 29.49 | **58.15 --上两行和后两行最大值58.15** |
| 1998 | 4 | 29.49 | **58.78 --上两行和后两行最大值58.78** |
| 1998 | 5 | 29.8 | 58.78 |
| 1998 | 6 | 58.78 | 117.76 |
| 1998 | 9 | 58.78 | 117.76 |
| 1998 | 10 | 117.76 | 117.76 |
| 1998 | 12 | 59.6 | 117.76 |
| 1998 | 14 | 58.78 | 117.76 |
| 1998 | 15 | 58.78 | 117.56 |

### 默认窗口声明

| | |
|---|---|
| 默认的是 | `rows between unbounded preceding and current row`<br>从开始到当前行 |

### lead和lag

lag和lead函数提供了跨行引用的能力。lag提供了访问结果集中前面行的能力，lead允许访问结果集中的后面的行。
在零售业中，同店销售额是一个计算得到的矩阵，用来衡量一个门店的业绩，通常用销售数据与去年统计度情况进行比较。在标准化数据模型中，这个矩阵的计算需要访问另一行，因为当年和前一年的销售数据在Sale列中是存储在不同的数据行中的。使用lead和lag函数强大的跨行引用能力，可以轻松计算出这个矩阵。

另一个例子是需要访问前一行或后一行数据的百分比增减计算。这个计算也可以使用lead和lag函数来最优化地写出。

### 语法和排序

在这样的计算中顺序是很重要的，可以使用order来控制排序。
lag (expression, offset, default ) over (partition-clause order-by-clause)

### 例1：从前一行中返回一个值

根据week列，获取上一周的sale列的值。注意是根据year,week来进行排序的。
lag(sale, 1, sale)在结果集中从sale列取出前一行的值。第三个值sale是默认值，如果引用了不存在的值，会返回空值，使用这个参数来修改，也可以直接使用某个数字来替代。

```
select year, week, sale,
       lag(sale, 1, sale) over(partition by product, country, region order by year, week) prior_wk_sales
  from sales_fact
 where country in ('Australia')
   and product = 'Xtend Memory'
 order by product, country, year, week;
```

| YEAR | WEEK | SALE | PRIOR_WK_SALES |
|---|---|---|---|
| 1998 | 1 | 58.15 | 58.15 **-- 默认值，前一行不存在，使用本行的sale值来替代。** |
| 1998 | 2 | 29.39 | 58.15 |
| 1998 | 3 | 29.49 | 29.39 |
| 1998 | 4 | 29.49 | 29.49 |
| 1998 | 5 | 29.8 | 29.49 |
| 1998 | 6 | 58.78 | 29.8 |
| 1998 | 9 | 58.78 | 58.78 |

### 理解数据行位移

lag(sale, 10, sale)向前偏移10行。

```
select year, week, sale,
       lag(sale, 10, sale) over(partition by product, country, region order by year, week) prior_wk_sales_10
  from sales_fact
 where country in ('Australia')
   and product = 'Xtend Memory'
 order by product, country, year, week;
```

| YEAR | WEEK | SALE | PRIOR_WK_SALES |
|---|---|---|---|
| 1998 | 1 | 58.15 | 58.15 |
| 1998 | 2 | 29.39 | 29.39 |
| 1998 | 3 | 29.49 | 29.49 |
| 1998 | 4 | 29.49 | 29.49 |
| 1998 | 5 | 29.8 | 29.8 |
| 1998 | 6 | 58.78 | 58.78 |
| 1998 | 9 | 58.78 | 58.78 |
| 1998 | 10 | 117.76 | 117.76 |
| 1998 | 12 | 59.6 | 59.6 |
| 1998 | 14 | 58.78 | 58.78 --包含此行在内的前面的10行都没有前第10的值，使用默认的sale来替代。 |
| 1998 | 15 | 58.78 | 58.15 --取前第10行的值，也就是第一行的sale值。 |
| 1998 | 17 | 58.78 | 29.39 |

如果默认值是0，即`lag(sale, 10, 0)`，结果看起来更明显：

| YEAR | WEEK | SALE | PRIOR_WK_SALES_10 |
|---|---|---|---|
| 1998 | 1 | 58.15 | 0 |
| 1998 | 2 | 29.39 | 0 |
| 1998 | 3 | 29.49 | 0 |
| 1998 | 4 | 29.49 | 0 |
| 1998 | 5 | 29.8 | 0 |
| 1998 | 6 | 58.78 | 0 |
| 1998 | 9 | 58.78 | 0 |
| 1998 | 10 | 117.76 | 0 |
| 1998 | 12 | 59.6 | 0 |
| 1998 | 14 | 58.78 | 0 |
| 1998 | 15 | 58.78 | 58.15 |
| 1998 | 17 | 58.78 | 29.39 |

### 例2：从下一行中返回一个值

lead(sale, 1, sale)访问后一行的值，这里默认值使用了自身，即最后一行的值为自身。相应地，它的位置很重要，注意order by year,week

```
select year, week, sale,
       lead(sale, 1, sale) over(partition by product, country, region order by year, week) prior_wk_sales
  from sales_fact
 where country in ('Australia')
   and product = 'Xtend Memory'
 order by product, country, year, week;
```

```
YEAR  WEEK  SALE   PRIOR_WK_SALES
1998  1     58.15  29.39 --后一行的值
1998  2     29.39  29.49
1998  3     29.49  29.49
1998  4     29.49  29.8
1998  5     29.8   58.78
```

### First_value和Last_value

First_value和Last_value，获取某个排序后的第一行或最后一行。
即根据窗口计算某个列的最大值和最小值。生成某个产品在一定市场领域的销售额最高商店的报表是这些分析函数的最经典应用。通常，sale列值最大的商店详情和销售额会在报表中放到一起显示。
任何计算最大值和最小值的报表都可以使用first_value和last_value函数。

### 例子：使用First_value来计算最大值

注意first_value(sale)和first_value(week)都是order by sale desc，获得这个排序的第一行的值（最大值）

```
select year, week, sale,
       first_value(sale) over(partition by product, country, region, year order by sale desc rows between unbounded
preceding and unbounded following) top_sale_value,
       first_value(week) over(partition by product, country, region, year order by sale desc rows between unbounded
preceding and unbounded following) top_sale_week
  from sales_fact
 where country in ('Australia')
   and product = 'Xtend Memory'
 order by product, country, year, week;
```

```
WITH T3 AS
 (WITH T2 AS
      (WITH T1 AS
          (SELECT d.regioncode,TO_CHAR(START_DT, 'yyyymm') AS REGMON, COUNT(1) AS REGNUM
            FROM DW.S_LOY_MEMBER c
            left join dw.s_org_ext b on c.X_STORE_ID = b.PAR_ROW_ID
            left join dw.shop d on d.sapshopid=b.INTEGRATION_ID
           GROUP BY d.regioncode,TO_CHAR(START_DT, 'yyyymm')
           ORDER BY d.regioncode,TO_CHAR(START_DT, 'yyyymm'))
         SELECT regioncode,SUBSTR(REGMON, 1, 4) AS REGYEAR, SUBSTR(REGMON, 5, 2) AS REGMON, REGNUM
           FROM T1
          ORDER BY regioncode,REGYEAR, REGMON)
     SELECT regioncode,REGYEAR, REGMON, REGNUM,
            LAG(REGNUM, 12, REGNUM) OVER(ORDER BY regioncode,REGYEAR, REGMON) PRIOR_YEAR_NUM,
            LAG(REGNUM, 1, REGNUM) OVER(ORDER BY regioncode,REGYEAR, REGMON) PRIOR_MON_NUM
       FROM T2)
SELECT *, TRUNC((REGNUM - PRIOR_YEAR_NUM) / PRIOR_YEAR_NUM ::NUMERIC * 100, 2) || '%' 同比,
       TRUNC((REGNUM - PRIOR_MON_NUM) / PRIOR_MON_NUM ::NUMERIC * 100, 2) || '%' 环比
  FROM T3
 WHERE REGYEAR >= 2013
   AND REGYEAR <= EXTRACT(YEAR FROM NOW())
   AND REGmon < lpad(EXTRACT(MONTH FROM NOW()),2,'0');
```

```
YEAR  WEEK  SALE   TOP_SALE_VALUE  TOP_SALE_WEEK
1998  1     58.15  172.56  48 --第一名是172.56，出现在第48周
1998  2     29.39  172.56  48
1998  3     29.49  172.56  48
1998  4     29.49  172.56  48
1998  5     29.8   172.56  48
1998  6     58.78  172.56  48
1998  9     58.78  172.56  48
1998  10    117.76 172.56  48
```

### 例子：使用Last_value来计算最小值

同样是order by sale desc，获得排序后的最后一行（最小值）

```
select year, week, sale,
       last_value(sale) over(partition by product, country, region, year order by sale desc rows between unbounded
preceding and unbounded following) low_sale
  from sales_fact
 where country in ('Australia')
   and product = 'Xtend Memory'
 order by product, country, year, week;
```

```
YEAR  WEEK  SALE   LOW_SALE
1998  1     58.15  28.76 --该分区的sale最后一名是28.76
1998  2     29.39  28.76
1998  3     29.49  28.76
1998  4     29.49  28.76
1998  5     29.8   28.76
1998  6     58.78  28.76
1998  9     58.78  28.76
1998  10    117.76 28.76
1998  12    59.6   28.76
1998  14    58.78  28.76
```

## 其他分析函数

### Nth_value(11gR2)

first_value和last_value提供了获取排过序的结果集中第一行或最后一行数据的能力，但如果需要获取其他行比如第二行是复杂的任务。使用nth_value可以获取结果集中的任意一行，而不仅是第一行或最后一行。first_value函数可以被写为nth_value(column_name,1)。
在统计分析中，在结果集的头部或尾部可能会出现异常值。在某些情况下，忽略first_value或经过排序的结果集中的first_value而从第二行开始取值是很重要的。结果集中的第2个值可以通过使用nth_value函数并将位移值设为2来取得。

nth_value语法如下：
nth_value(measure, n) [from first | from last] [respect nulls |ignore nulls]
 over (partitioning-clause order-by-clause windowing-clause)

nth_value函数的第一个参数是列名，第二个参数是窗口位移量。
对于nth_value函数，from first和respect nulls子句是默认值。如果声明了from first子句，则nth_value函数从窗口的第1行开始寻找位移后的数据行。respect nulls子句表示如果在位移行中包含空值则将会返回空值。

具有了声明开窗子句的能力，nth_value函数就变得非常强大，可以访问结果集或分区中的任意行。

nth_value(sale,2)子句访问窗口中的第2行。

```
select year, week, sale,
       nth_value(sale, 2) over(partition by product, country, region, year order by sale desc rows between unbounded
preceding and unbounded following) sale_2nd_top
  from sales_fact
 where country in ('Australia')
   and product = 'Xtend Memory'
 order by product, country, year, week;
```

```
YEAR  WEEK  SALE   SALE_2ND_TOP
1998  1     58.15  117.76
1998  2     29.39  117.76
1998  3     29.49  117.76
1998  4     29.49  117.76
1998  5     29.8   117.76
1998  6     58.78  117.76
1998  9     58.78  117.76
1998  10    117.76 117.76 --经过分区后排名第一的应该是上面的172.56，117.76是第二名
1998  12    59.6   117.76
```

产品在年度范围内根据销售额进行排序

```
SELECT YEAR, WEEK, SALE,
       NTH_VALUE(SALE, 1) OVER(PARTITION BY PRODUCT, COUNTRY, REGION, YEAR ORDER BY SALE DESC ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
FOLLOWING) SALE_1ND_TOP,
       NTH_VALUE(SALE, 2) OVER(PARTITION BY PRODUCT, COUNTRY, REGION, YEAR ORDER BY SALE DESC ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
FOLLOWING) SALE_2ND_TOP,
       NTH_VALUE(SALE, 3) OVER(PARTITION BY PRODUCT, COUNTRY, REGION, YEAR ORDER BY SALE DESC ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED
FOLLOWING) SALE_3ND_TOP
  FROM SALES_FACT
 WHERE COUNTRY IN ('Australia')
   AND PRODUCT = '256MB Memory Card'
 ORDER BY PRODUCT, COUNTRY, YEAR, WEEK;
```

| YEAR | WEEK | SALE | SALE_1ND_TOP | SALE_2ND_TOP | SALE_3ND_TOP |
|---|---|---|---|---|---|
| 1999 | 47 | 355.24 | 2765.84 | 1382.92 | 358.18 |
| 2000 | 11 | 310.44 | 312.62 | 310.44 | 310.44 |

### Rank

**rank函数以数值形式返回一个数据行在排序后的结果集中的位置。**如果数据行是按某一列进行排序的，则这一行在窗口中的位置就反映了该值在窗口内数据行中的排名。在排名并列的情况下，具有同样值的行将具有同样的排名而接下来的排名将会被跳过，从而在排名值上留下空隙。这意味着某两行可能具有同一排名，排名也不一定是连续的。

**rank函数对于计算最上面或最下面N行是非常有用的。例如，查找销售量在前10位的周就是零售业数据仓库的典型查询。这样一个查询将会大大受益于rank函数的使用。关键在于Order by的排序字段。**
rank函数对于找出中间的N行数据也是有用的。例如，取出按销售额排序的21到40位的数据，那么就可以使用between 21 and 40的子查询中使用rank函数来筛选出中间的20行数据。

rank函数语法如下：
rank() over (partition-clause order-by-clause)

开窗子句在rank函数中不适用，rank函数是应用在数据分区中所有行上的。

rank() 根据sale desc列出排名。并将排名结果进行排序。rank()遇到并列在之后的排名中会跳过并列的数量

```
select *
  from (select year, week, sale,
               rank() over(partition by product, country, region, year order by sale desc) sales_rank
          from sales_fact
         where country in ('Australia')
           and product = 'Xtend Memory'
         order by product, country, year, week)
 where sales_rank <= 10
 order by 1, 4;
```

```
YEAR  WEEK  SALE   SALES_RANK
1998  48    172.56  1
1998  10    117.76  2
1998  18    117.56  3
1998  23    117.56  3
1998  26    117.56  3
1998  38    115.84  6 --跳过了4、5两个排名
1998  42    115.84  6
1998  39    115.84  6
1998  34    115.44  9--跳过了7、8两个排名
1998  52    86.38   10
1999  17    148.12  1
1999  47    147.78  2
1999  15    135.1   3
1999  44    130.72  4
1999  42    120.59  5
1999  25    107.44  6
1999  22    107.44  6
1999  34    105.8   8
1999  37    105.8   8
1999  8     103.11  10
```

### Dense_rank

与rank区别是在相同值时不跳过排名.
dense_rank函数中的空值排序位置可以通过nulls first或nulls last子句来控制。对于升序排列来说，nulls last是默认值，而对于降序排列来说，nulls first是默认值。默认是nulls first起作用。

同上。

```
dense_rank()
over(partition by product, country, region, year order by sale desc nulls first)
```

```
select *
  from (select year, week, sale,
               dense_rank() over(partition by product, country, region, year order by sale desc) sales_rank
          from sales_fact
         where country in ('Australia')
           and product = 'Xtend Memory'
         order by product, country, year, week)
 where sales_rank <= 10
 order by 1, 4
```

```
YEAR  WEEK  SALE   SALES_RANK
1998  48    172.56  1
1998  10    117.76  2
1998  18    117.56  3
1998  23    117.56  3
1998  26    117.56  3
1998  38    115.84  4 --不跳过排名，使用并列排名方式。
1998  39    115.84  4
1998  42    115.84  4
1998  34    115.44  5
1998  52    86.38   6
1998  21    59.6    7
1998  12    59.6    7
1998  19    58.98   8 --不跳过排名，使用并列排名方式。
1998  17    58.78   9
1998  15    58.78   9
1998  14    58.78   9
1998  6     58.78   9
1998  9     58.78   9
1998  51    58.32   10
1998  41    58.32   10
```

### Row_number

row_number函数为有序结果集中的每一行分配一个唯一编号。如果指定分区，则为每个分区中的位置分配一个编号。如果没有指定分区，则为结果集中的每行指定一个编号。

与rank、dense_rank不同，row_number不支持开窗子句。
row_number是一个非确定函数，如果一个数据分区中两行具有相同的值，每次生成结果可能不同，；而rank、dense_rank是确定函数，每次结果一致。

这里为每个year分区指定一个唯一编号。

```
SELECT YEAR, WEEK, SALE,
       ROW_NUMBER() OVER(PARTITION BY PRODUCT, COUNTRY, REGION, YEAR ORDER BY SALE DESC) SALES_RN,
       RANK() OVER(PARTITION BY PRODUCT, COUNTRY, REGION, YEAR ORDER BY SALE DESC) SALES_RANK
  FROM SALES_FACT
 WHERE COUNTRY IN ('Australia')
   AND PRODUCT = 'Xtend Memory'
 ORDER BY PRODUCT, COUNTRY, YEAR, SALES_RN;
```

```
YEAR  WEEK  SALE   SALES_RN     SALES_RANK
1998  48    172.56 1            1
1998  10    117.76 2            2
1998  26    117.56 4            3 --row_number是4，rank是3，sale值相同row_number可能不同。
1998  18    117.56 3            3
1998  23    117.56 5            3
1998  39    115.84 6            6
1998  38    115.84 7            6
1998  42    115.84 8            6
```

```
WITH T1 AS
 (SELECT SHOP_ID, AGE_GROUP,
         SUM(M) OVER(PARTITION BY SHOP_ID) AS ALL_M,
         SUM(M) OVER(PARTITION BY SHOP_ID, AGE_GROUP) AS AGE_GROUP_M
    FROM TEST.CUSTA_BASICLABEL_BASE_RE_9354)
SELECT DISTINCT SHOP_ID, AGE_GROUP, ALL_M, AGE_GROUP_M,
                ROUND(AGE_GROUP_M / ALL_M * 100, 2) AS PERCENT
  FROM T1;
```

按年份、周进行排序，给出序号

```
SELECT PRODUCT, COUNTRY, REGION, YEAR, WEEK, SALE,
       ROW_NUMBER() OVER(PARTITION BY PRODUCT, COUNTRY, REGION, YEAR ORDER BY YEAR, WEEK) SALES_RN
  FROM SALES_FACT
 WHERE COUNTRY IN ('Australia')
   AND PRODUCT = 'Xtend Memory'
 ORDER BY PRODUCT, COUNTRY, REGION, YEAR, SALES_RN;
```

| PRODUCT | COUNTRY | REGION | YEAR | WEEK | SALE | SALES_RN |
|---|---|---|---|---|---|---|
| Xtend Memory | Australia | Australia | 1998 | 1 | 58.15 | 1 |
| Xtend Memory | Australia | Australia | 1998 | 2 | 29.39 | 2 |
| Xtend Memory | Australia | Australia | 1998 | 3 | 29.49 | 3 |
| Xtend Memory | Australia | Australia | 1998 | 4 | 29.49 | 4 |
| Xtend Memory | Australia | Australia | 1998 | 5 | 29.8 | 5 |
| Xtend Memory | Australia | Australia | 1998 | 6 | 58.78 | 6 |

### Ratio_to_report

**ratio_to_report计算分区中某个值和值的比率。**如果没有声明分区子句，这个函数将会计算一个值与整个结果集中和值的比率。这个分析函数在不同层级上计算比率是非常有用的，不用进行自联接。

ratio_to_report在计算报表中某个值占总值的百分比的时候是有用的。例如，**考虑某个零售连锁店中某种产品的销售报表。每家门店都对该产品的销售总额做出了贡献，并且知道每家门店的销售额占总销售额的百分比对于市场趋势是非常有用的。ratio_to_report允许很方便地计算百分比。并且，这个比率可以在不同的层级例如district、region和country上计算。**本质上，数据可以被以多种不同的方式切块或切片来进行市场趋势分析。

sales_yr是在product, country, region, year层级上计算比率；
sales_prod是在product, country, region上计算比率，是所有年份。
用trunc函数保留两位。

```
select year, week, sale,
       trunc(100 * ratio_to_report(sale) over(partition by product, country, region, year), 2) sales_yr,
       trunc(100 * ratio_to_report(sale) over(partition by product, country, region), 2) sales_prod
  from sales_fact
 where country in ('Australia')
   and product = 'Xtend Memory'
 order by product, country, year, week;
```

```
YEAR  WEEK  SALE   SALES_YR     SALES_PROD
1998  1     58.15  2.26         0.43 --sales_yr：这一周销售额占这一年的百分多少；占有史以来的销售额多少。
1998  2     29.39  1.14         0.21
1998  3     29.49  1.15         0.22
1998  4     29.49  1.15         0.22
1998  5     29.8   1.16         0.22
1998  6     58.78  2.29         0.43
1998  9     58.78  2.29         0.43
1998  10    117.76 4.59         0.88
1998  12    59.6   2.32         0.44
1998  14    58.78  2.29         0.43
1998  15    58.78  2.29         0.43
1998  17    58.78  2.29         0.43
1998  18    117.56 4.58         0.87
1998  19    58.98  2.3          0.44
1998  21    59.6   2.32         0.44
1998  23    117.56 4.58         0.87
1998  26    117.56 4.58         0.87
1998  27    57.52  2.24         0.43
1998  28    57.72  2.25         0.43
1998  29    57.72  2.25         0.43
1998  34    115.44 4.5          0.86
1998  35    57.52  2.24         0.43
1998  38    115.84 4.52         0.86
1998  39    115.84 4.52         0.86
1998  40    57.52  2.24         0.43
1998  41    58.32  2.27         0.43
1998  42    115.84 4.52         0.86
1998  43    57.52  2.24         0.43
1998  44    57.52  2.24         0.43
1998  45    57.52  2.24         0.43
1998  46    57.52  2.24         0.43
1998  47    57.72  2.25         0.43
1998  48    172.56 6.73         1.29
1998  50    28.76  1.12         0.21
1998  51    58.32  2.27         0.43
1998  52    86.38  3.37         0.64
```

### Percent_rank

percent_rank函数以0到1之间的分数形式返回某个值在数据分区中的排名。percent_rank的计算公式为(rank-1)/(n-1)，其中如果声明了分区子句N就是分区中的数据行数，如果没有声明分区子句N就是结果集中所有的数据函数。**percent_rank函数对于计算某个值在结果集中按百分比所处的位置是很有用的。**
例如，计算一家零售连锁门店的销售额在某个地区或区域中所处的名次可以有助于找出表现最好或最差的门店。

根据product, country, region, year分区，计算sale desc。值为0-1之间，为得到百分比乘以100。

```
SELECT *
  FROM (SELECT YEAR, WEEK, SALE,
               TRUNC(100 * PERCENT_RANK()
                     OVER(PARTITION BY PRODUCT, COUNTRY, REGION, YEAR ORDER BY SALE DESC), 2) PR
          FROM SALES_FACT
         WHERE COUNTRY IN ('Australia')
           AND PRODUCT = 'Xtend Memory')
 WHERE PR < 50
 ORDER BY YEAR, SALE DESC;
```

| YEAR | WEEK | SALE | PR |
|---|---|---|---|
| 1998 | 48 | 172.56 | 0 |
| 1998 | 10 | 117.76 | 2.85 |
| 1998 | 26 | 117.56 | 5.71 |
| 1998 | 18 | 117.56 | 5.71 |
| 1998 | 23 | 117.56 | 5.71 |
| 1998 | 39 | 115.84 | 14.28 |
| 1998 | 38 | 115.84 | 14.28 |
| 1998 | 42 | 115.84 | 14.28 |
| 1998 | 34 | 115.44 | 22.85 |

### Percentile_cont

percentile_cont对于计算内插值（例如每个地区或城市中等收入家庭的收入）是非常有用的。percentile_cont函数接受一个0到1之间的几率值并返回与声明了排序的percentile_rank函数计算值相等的百分比。事实上，percentile_cont是percent_rank的反函数，与percent_rank函数的输出结合起来看更可以容易理解percentile_cont函数。

**percentile_cont函数取与参数的percent_rank相匹配（或内插）的列值。**例如，percentile_cont(0.25)获取percent_rank为0.25的值，假设这两个函数的排序顺序相匹配。另一个例子是计算一个城市或地区中等收入家庭的收入值。由于定义的是中等家庭，中位值的percent_rank为0.5。percentile_cont(0.5)将返回中位值，**median函数是percentile_cont函数的一个默认值为0.5的特例**。

当没有值与指定的percentile_rank精确匹配的时候，percentile_cont(0.5)会计算两个离得很近的值的平均值。

percentile_cont函数的语法是：
percentile_cont (expr) within group (sort-clause)
  over (partition-clause order-by-clause)

percentile_cont(0.5) within group(order by sale desc) over(partition by product, country, region, year) pc
percentile_cont(0.5) within group(order by sale) over(partition by product, country, region, year) pc
两个值是相同的，如果没有0.5，则取两个相近值的平均值。

```
select year, week, sale,
       percentile_cont(0.5) within group(order by sale desc) over(partition by product, country, region, year) pc,
       percent_rank() over(partition by product, country, region, year order by sale desc) pr
  from sales_fact
 where country in ('Australia')
   and product = 'Xtend Memory';
```

| YEAR | WEEK | SALE | PC | PR |
|---|---|---|---|---|
| 1998 | 48 | 172.56 | 58.55 | 0 **--percentile_cont(0.5)表示中位数。两个最值为0和1，接近0.5的是中位数。** |
| 1998 | 10 | 117.76 | 58.55 | 0.0285714285714286 |
| 1998 | 18 | 117.56 | 58.55 | 0.0571428571428571 |
| 1998 | 23 | 117.56 | 58.55 | 0.0571428571428571 |
| 1998 | 26 | 117.56 | 58.55 | 0.0571428571428571 |
| 1998 | 39 | 115.84 | 58.55 | 0.142857142857143 |
| 1998 | 42 | 115.84 | 58.55 | 0.142857142857143 |
| 1998 | 38 | 115.84 | 58.55 | 0.142857142857143 |
| 1998 | 34 | 115.44 | 58.55 | 0.228571428571429 |
| 1998 | 52 | 86.38 | 58.55 | 0.257142857142857 |
| 1998 | 21 | 59.6 | 58.55 | 0.285714285714286 |
| 1998 | 12 | 59.6 | 58.55 | 0.285714285714286 |
| 1998 | 19 | 58.98 | 58.55 | 0.342857142857143 |
| 1998 | 17 | 58.78 | 58.55 | 0.371428571428571 |
| 1998 | 6 | 58.78 | 58.55 | 0.371428571428571 |
| 1998 | 15 | 58.78 | 58.55 | 0.371428571428571 |
| 1998 | 14 | 58.78 | 58.55 | 0.371428571428571 |
| 1998 | 9 | 58.78 | 58.55 | 0.371428571428571 |
| 1998 | 51 | 58.32 | 58.55 | 0.514285714285714 **--中位数。** |
| 1998 | 41 | 58.32 | 58.55 | 0.514285714285714 |
| 1998 | 1 | 58.15 | 58.55 | 0.571428571428571 |
| 1998 | 28 | 57.72 | 58.55 | 0.6 |
| 1998 | 29 | 57.72 | 58.55 | 0.6 |
| 1998 | 47 | 57.72 | 58.55 | 0.6 |
| 1998 | 43 | 57.52 | 58.55 | 0.685714285714286 |
| 1998 | 40 | 57.52 | 58.55 | 0.685714285714286 |
| 1998 | 27 | 57.52 | 58.55 | 0.685714285714286 |
| 1998 | 35 | 57.52 | 58.55 | 0.685714285714286 |
| 1998 | 44 | 57.52 | 58.55 | 0.685714285714286 |
| 1998 | 46 | 57.52 | 58.55 | 0.685714285714286 |
| 1998 | 45 | 57.52 | 58.55 | 0.685714285714286 |
| 1998 | 5 | 29.8 | 58.55 | 0.885714285714286 |
| 1998 | 3 | 29.49 | 58.55 | 0.914285714285714 |
| 1998 | 4 | 29.49 | 58.55 | 0.914285714285714 |
| 1998 | 2 | 29.39 | 58.55 | 0.971428571428571 |
| 1998 | 50 | 28.76 | 58.55 | 1 **--另一端的最值。** |

### Percentile_disc

percentile_disc类似于percentile_cont函数，只是percentile_cont函数使用列连续分布模型，percentile_disc使用了离散分布模型。当没有值与指定的percentile_rank精确匹配的时候，percentile_cont(0.5)会计算两个离得很近的值的平均值。相反，在升序情况下，percentile_disc函数只取比所传递的参数percentile_rank值更大的值。在降序排列的时候，percentile_disc函数只取比所传递percentile_rank值更小的值。

percentile_disc(0.5) within group(order by sale desc) over(partition by product, country, region, year) pd_desc,
percentile_disc(0.5) within group(order by sale) over(partition by product, country, region, year) pd_asc,

正反order by sale的取值是不同的，反向取58.78，正向取58.32。

```
select year, week, sale,
       percentile_disc(0.5) within group(order by sale desc) over(partition by product, country, region, year) pd_desc,
       percentile_disc(0.5) within group(order by sale) over(partition by product, country, region, year) pd_asc,
       percent_rank() over(partition by product, country, region, year order by sale desc) pr
  from sales_fact
 where country in ('Australia')
   and product = 'Xtend Memory';
```

| YEAR | WEEK | SALE | PD_DESC | PD_ASC | PR |
|---|---|---|---|---|---|
| 1998 | 48 | 172.56 | 58.78 | 58.32 | 0 --PD_DESC PD_ASC的值不同。 |
| 1998 | 10 | 117.76 | 58.78 | 58.32 | 0.0285714285714286 |
| 1998 | 18 | 117.56 | 58.78 | 58.32 | 0.0571428571428571 |
| 1998 | 23 | 117.56 | 58.78 | 58.32 | 0.0571428571428571 |
| 1998 | 26 | 117.56 | 58.78 | 58.32 | 0.0571428571428571 |
| 1998 | 39 | 115.84 | 58.78 | 58.32 | 0.142857142857143 |
| 1998 | 42 | 115.84 | 58.78 | 58.32 | 0.142857142857143 |
| 1998 | 38 | 115.84 | 58.78 | 58.32 | 0.142857142857143 |
| 1998 | 34 | 115.44 | 58.78 | 58.32 | 0.228571428571429 |
| 1998 | 52 | 86.38 | 58.78 | 58.32 | 0.257142857142857 |
| 1998 | 21 | 59.6 | 58.78 | 58.32 | 0.285714285714286 |
| 1998 | 12 | 59.6 | 58.78 | 58.32 | 0.285714285714286 |
| 1998 | 19 | 58.98 | 58.78 | 58.32 | 0.342857142857143 |
| 1998 | 17 | 58.78 | 58.78 | 58.32 | 0.371428571428571 |
| 1998 | 6 | 58.78 | 58.78 | 58.32 | 0.371428571428571 |
| 1998 | 15 | 58.78 | 58.78 | 58.32 | 0.371428571428571 |
| 1998 | 14 | 58.78 | 58.78 | 58.32 | 0.371428571428571 |
| 1998 | 9 | 58.78 | 58.78 | 58.32 | 0.371428571428571 |
| 1998 | 51 | 58.32 | 58.78 | 58.32 | 0.514285714285714 |
| 1998 | 41 | 58.32 | 58.78 | 58.32 | 0.514285714285714 |
| 1998 | 1 | 58.15 | 58.78 | 58.32 | 0.571428571428571 |
| 1998 | 28 | 57.72 | 58.78 | 58.32 | 0.6 |
| 1998 | 29 | 57.72 | 58.78 | 58.32 | 0.6 |
| 1998 | 47 | 57.72 | 58.78 | 58.32 | 0.6 |
| 1998 | 43 | 57.52 | 58.78 | 58.32 | 0.685714285714286 |
| 1998 | 40 | 57.52 | 58.78 | 58.32 | 0.685714285714286 |
| 1998 | 27 | 57.52 | 58.78 | 58.32 | 0.685714285714286 |
| 1998 | 35 | 57.52 | 58.78 | 58.32 | 0.685714285714286 |
| 1998 | 44 | 57.52 | 58.78 | 58.32 | 0.685714285714286 |
| 1998 | 46 | 57.52 | 58.78 | 58.32 | 0.685714285714286 |
| 1998 | 45 | 57.52 | 58.78 | 58.32 | 0.685714285714286 |
| 1998 | 5 | 29.8 | 58.78 | 58.32 | 0.885714285714286 |
| 1998 | 3 | 29.49 | 58.78 | 58.32 | 0.914285714285714 |
| 1998 | 4 | 29.49 | 58.78 | 58.32 | 0.914285714285714 |
| 1998 | 2 | 29.39 | 58.78 | 58.32 | 0.971428571428571 |
| 1998 | 50 | 28.76 | 58.78 | 58.32 | 1 |

### Ntile

ntile函数对一个数据分区中的有序结果集进行划分，将其分组为几个桶，并为每个小组分配一个唯一的组编号。这个函数在统计分析时候是很有用的。例如，如果想移除异常值，可以将它们分组到顶部或底部的桶中，然后在统计分析的时候将这些值排除。Oracle数据库的统计分析也使用ntile函数来计算直方图信息边界。在统计学术语中，ntile函数创建等宽直方图信息。
ntile不支持开窗子句。

ntile函数在诸如将工作量在N个并行的进程中进行划分的实际应用是很有用的，假设有10个并行进程，可以将总工作量划分为10个桶并将每个桶放到一个进程中。

使用ntile将数据分为10个桶。这里的依据是sale的值进行分桶。

```
select year, week, sale, ntile(10) over(partition by product, country, region, year order by sale desc) group#
  from sales_fact
 where country in ('Australia')
   and product = 'Xtend Memory';
```

| YEAR | WEEK | SALE | GROUP# |
|---|---|---|---|
| 1998 | 48 | 172.56 | 1 |
| 1998 | 10 | 117.76 | 1 |
| 1998 | 18 | 117.56 | 1 |
| 1998 | 26 | 117.56 | 1 |
| 1998 | 23 | 117.56 | 2 |
| 1998 | 39 | 115.84 | 2 |
| 1998 | 38 | 115.84 | 2 |
| 1998 | 42 | 115.84 | 2 |
| 1998 | 34 | 115.44 | 3 |
| 1998 | 52 | 86.38 | 3 |
| 1998 | 12 | 59.6 | 3 |
| 1998 | 21 | 59.6 | 3 |

```
--将表dsn_account根据gaid主键分成16个部分，并取各个部分的最值。
with t as
 (SELECT a.gaid, NTILE(16) OVER(ORDER BY gaid) AS ordered FROM dsn_account a)
select MIN(gaid), MAX(gaid) from t group by ordered;
```

| MIN(GAID) | MAX(GAID) |
|---|---|

```
1       61125
305626  366750
61126   122250
798793  859916
244501  305625
859917  925535
613875  674998
737669  798792
183376  244500
427876  489000
122251  183375
366751  427875
489001  550125
550126  613874
674999  737668
925536  987424
```

拼凑出分割表的语句（在SQL Server 2008中）

```
with t as (
SELECT a.gaid
   ,NTILE(100) OVER(ORDER BY gaid ) AS 'ordered'
FROM dsn_account a)
select 'insert into dsn_account(
GAID,ACCOUNT_ID,ACCOUNT_NAME,ACCOUNT_CODE,VIP,CREATEDATE,VERCODE)
select "gaid","account_id","account_name","account_code","vip","createdate","vercode"
from "dsn_account"@replt where "gaid">='+ cast(MIN(gaid) as varchar) +' and "gaid"<='+cast(max(gaid) as varchar)+';commit;'
 from t group by ordered;
```

```
insert into dsn_account
   (GAID, ACCOUNT_ID, ACCOUNT_NAME, ACCOUNT_CODE, VIP, CREATEDATE, VERCODE)
   select "gaid", "account_id", "account_name", "account_code", "vip", "createdate", "vercode"
     from "dsn_account"@replt
    where "gaid" >= 1
      and "gaid" <= 9787;
commit;
insert into dsn_account
   (GAID, ACCOUNT_ID, ACCOUNT_NAME, ACCOUNT_CODE, VIP, CREATEDATE, VERCODE)
   select "gaid", "account_id", "account_name", "account_code", "vip", "createdate", "vercode"
     from "dsn_account"@replt
    where "gaid" >= 9788
      and "gaid" <= 19574;
commit;
```

### Stddev

stddev函数可以用来在一个数据分区中的某些数据行上，或者如果没有声明分区的子句在整个结果集上计算**标准差**。
其他统计函数还有：

| | |
|---|---|
| stddev_samp | 计算累积采样标准偏差 |
| stddev_pop | 计算总体标准偏差 |

rows between unbounded preceding and unbounded following为所有行计算标准差。

```
select year, week, sale,
       stddev(sale) over(partition by product, country, region, year order by Sale desc rows between unbounded preceding
and unbounded following) stddv
  from sales_fact
 where country in ('Australia')
   and product = 'Xtend Memory'
 order by year, week;
```

```
YEAR  WEEK  SALE   STDDV
.....
2000  50    21.19  49.8657423121942
2000  52    67.45  49.8657423121942
2001  1     92.26  59.1063592290468
2001  2     118.38 59.1063592290468
2001  3     47.24  59.1063592290468
.....
```

### Listagg

在11gR2中引入，进行字符串处理。它提供了将来自多个行中的列值转化为列表格式的能力。
listagg (string, separator) within group (order-by-clause)
  over (partition-by-clause)

listagg函数使用within group (order-by-clause) 子句来声明排序顺序，类似于其他分析函数的order-by子句类似。
第一个参数是进行连接的字符串或列名，第二个参数是分隔符。

根据country的降序连接子查询结果的country列。

```
select listagg(country, ',') within group(order by country desc)
  from (select distinct country from sales_fact order by country);
```

```
United States of America,United Kingdom,Turkey,Spain,Singapore,Saudi Arabia,Poland,New
Zealand,Japan,Italy,Germany,France,Denmark,China,Canada,Brazil,Australia,Argentina
```

```
SELECT LISTAGG(COLUMN_NAME, ',') WITHIN GROUP(ORDER BY COLUMN_ID)
  FROM USER_TAB_COLS UT
 WHERE UT.TABLE_NAME = UPPER('pos_journal');
```

## 性能调优

可以用来改进不使用分析函数的查询语句的性能。
注意：基于成本的优化器不会分析函数分配或计算成本（11gR2开始）。计算出的SQL语句成本并未考虑分析函数成本。

### 执行计划
执行计划中出现window sort表示SQL使用了一个分析函数。

### 谓语
谓语应该尽可能早地应用于表上来减小结果集以获得更好的性能。
**一般来说，分区列上的谓语可以安全地前推到视图中。但分析函数中的order-by子句的列不能被前推，因为跨行引用需要访问同一分区中的其他行，即使这些数据行并不在最终的结果中返回。**

这个视图中先执行分析了函数而没有尽量使用谓语进行过滤。

```
create or replace view max_5_weeks_vw as
   select  country , product, region, year, week,sale,
    max (sale) over(
          partition by product, country, region ,year
          order by year, week
    rows between 2 preceding and 2 following
           ) max_weeks_5
  from sales_fact;
```

这里因为order by year,week，因此不会谓语前推。这里选择了全表扫描（如果不存在(country, product)的复合索引）。

```
select year, week, sale, max_weeks_5
  from max_5_weeks_vw
 where country in ('Australia')
   and product = 'Xtend Memory'
   and region = 'Australia'
   and year = 2000
   and week < 14
 order by year, week;
```

### 索引

```
create index sales_fact_i1 on sales_fact(country, product);
```

**添加了索引，这时候加入week是没有用的，因为它不会被前推！**

```
select year, week, sale, max_weeks_5
  from max_5_weeks_vw
 where country in ('Australia')
   and product = 'Xtend Memory'
   and region = 'Australia'
   and year = 2000
   and week < 14
 order by year, week;
```

## 高级话题

### 动态SQL
分析SQL的分区或排序列上不能使用绑定变量。
如果目的是动态调整分区列，那么考虑创建一个存储过程包来获取存储过程中的逻辑。

存储过程analytic_dynamic_prc接收分区列的字符串，以及筛选条件。

```
create or replace procedure analytic_dynamic_prc
(
   part_col_string varchar2, v_country varchar2, v_product varchar2
) is
   type numtab is table of number(18, 2) index by binary_integer;
   l_year       numtab;
   l_week       numtab;
   l_sale       numtab;
   l_rank       numtab;
   l_sql_string varchar2(512);
begin
   l_sql_String := 'select * from (
   select year, week,sale,
    rank() over(
          partition by ' || part_col_string || '
          order by sale desc
           ) sales_rank
   from sales_fact
   where country in (' || chr(39) || v_country || chr(39) || ' )  and
         product =' || chr(39) || v_product || chr(39) || ' order by product, country,year, week
    ) where sales_rank<=10
    order by 1,4';
   execute immediate l_sql_string bulk collect
      into l_year, l_week, l_sale, l_rank;
   for i in 1 .. l_year.count
   loop
      dbms_output.put_line(l_year(i) || ' |' || l_week(i) || '|' || l_sale(i) || '|' || l_rank(i));
   end loop;
end;
```

```
exec analytic_dynamic_prc ( 'product, country, region','Australia','Xtend Memory');
```

```
1998 |48|172.56|9
2000 |46|246.74|3
2000 |21|187.48|5
2000 |43|179.12|7
2000 |34|178.52|8
2001 |16|278.44|1
2001 |4|256.7|2
2001 |21|233.7|4
2001 |48|182.96|6
2001 |30|162.91|10
2001 |14|162.91|10
```

```
exec analytic_dynamic_prc ( 'product, country,region, year','Australia','Xtend Memory');
```

```
1998 |48|172.56|1
1998 |10|117.76|2
1998 |18|117.56|3
1998 |23|117.56|3
1998 |26|117.56|3
1998 |38|115.84|6
1998 |42|115.84|6
1998 |39|115.84|6
1998 |34|115.44|9
1998 |52|86.38|10
1999 |17|148.12|1
.....
```

### 嵌套分析函数

分析函数不能进行嵌套，但可以通过子查询来实现嵌套的效果。例如lag(first_value (column,1),1)子句就是错误的。
假设要在同一行中列出今年和去年sale列的最大值，可以在子查询中使用lag和first_value分析函数来写出SQL语句。

原书例子中使用lag(访问前一行)，但并不符合使用习惯，应该改为lead（访问后一行）。

```
select year, week, top_sale_year, lag(top_sale_year) over(order by year desc) prev_top_sale_yer
  from (select distinct first_value(year) over(partition by product, country, region, year order by sale desc rows
between unbounded preceding and unbounded following) year,
                        first_value(week) over(partition by product, country, region, year order by sale desc rows
between unbounded preceding and unbounded following) week,
                        first_value(sale) over(partition by product, country, region, year order by sale desc rows
between unbounded preceding and unbounded following) top_sale_year
          from sales_fact
         where country in ('Australia')
           and product = 'Xtend Memory')
 order by year, week;
```

```
YEAR  WEEK  TOP_SALE_YEAR      PREV_TOP_SALE_YER
1998    48       172.56  148.12
1999    17       148.12  246.74
2000    46       246.74  278.44
2001    16       278.44
```

```
select year, week, top_sale_year, lead(top_sale_year) over(order by year desc) prev_top_sale_yer
  from (select distinct first_value(year) over(partition by product, country, region, year order by sale desc rows
between unbounded preceding and unbounded following) year,
                        first_value(week) over(partition by product, country, region, year order by sale desc rows
between unbounded preceding and unbounded following) week,
                        first_value(sale) over(partition by product, country, region, year order by sale desc rows
between unbounded preceding and unbounded following) top_sale_year
          from sales_fact
         where country in ('Australia')
           and product = 'Xtend Memory')
 order by year, week;
```

```
YEAR  WEEK  TOP_SALE_YEAR      PREV_TOP_SALE_YER
1998    48       172.56
1999    17       148.12  172.56
2000    46       246.74  148.12
2001    16       278.44  246.74
```

### 并行

通过在SQL语句中声明parallel提示活在对象级设置并行度，分析函数也可以是并行的。

```
explain plan for select /*+parallel(4)*/ year, week, top_sale_year, lead(top_sale_year) over(order by year desc)
prev_top_sale_yer
  from (select distinct first_value(year) over(partition by product, country, region, year order by sale desc rows
between unbounded preceding and unbounded following) year,
                        first_value(week) over(partition by product, country, region, year order by sale desc rows
between unbounded preceding and unbounded following) week,
                        first_value(sale) over(partition by product, country, region, year order by sale desc rows
between unbounded preceding and unbounded following) top_sale_year
          from sales_fact
         where country in ('Australia')
           and product = 'Xtend Memory')
 order by year, week;
```

```
select * from table(dbms_xplan.display);
```

```
Plan hash value: 2880616722
-------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                  | Name                  | Rows  | Bytes | Cost (%CPU)| Time     |  TQ  |IN-OUT| PQ Distrib |
-------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |                       |   161 |  6279 |    90   (5)| 00:00:02 |      |      |            |
|   1 |  RESULT CACHE              | 62bpwxub3119bfqcf2fasfc8tx |      |       |            |          |      |      |            |
|   2 |   SORT ORDER BY            |                       |   161 |  6279 |    90   (5)| 00:00:02 |      |      |            |
```

```
|  3 |   WINDOW BUFFER               |                     |  161 |  6279 |    90   (5)| 00:00:02 |       |      |            |
|  4 |    PX COORDINATOR             |                     |      |       |          |          |       |      |          |         |
|  5 |     PX SEND QC (ORDER)        | :TQ10003            |  161 |  6279 |    90   (5)| 00:00:02 |  Q1,03 | P ->S | QC (ORDER) |
|  6 |      SORT ORDER BY            |                     |  161 |  6279 |    90   (5)| 00:00:02 |  Q1,03 | PCW P |            |
|  7 |       PX RECEIVE              |                     |  161 |  6279 |    88   (3)| 00:00:02 |  Q1,03 | PCW P |            |
|  8 |        PX SEND RANGE          | :TQ10002            |  161 |  6279 |    88   (3)| 00:00:02 |  Q1,02 | P ->P | RANGE      |
|  9 |         VIEW                  |                     |  161 |  6279 |    88   (3)| 00:00:02 |  Q1,02 | PCW P |            |
| 10 |          HASH UNIQUE          |                     |  161 | 10304 |    88   (3)| 00:00:02 |  Q1,02 | PCW P |            |
| 11 |           PX RECEIVE          |                     |  161 | 10304 |    88   (3)| 00:00:02 |  Q1,02 | PCW P |            |
| 12 |            PX SEND HASH       | :TQ10001            |  161 | 10304 |    88   (3)| 00:00:02 |  Q1,01 | P ->P | HASH       |
| 13 |             WINDOW SORT       |                     |  161 | 10304 |    88   (3)| 00:00:02 |  Q1,01 | PCW P |            |
| 14 |              PX RECEIVE       |                     |  161 | 10304 |    86   (0)| 00:00:02 |  Q1,01 | PCW P |            |
| 15 |               PX SEND HASH    | :TQ10000            |  161 | 10304 |    86   (0)| 00:00:02 |  Q1,00 | P ->P | HASH       |
| 16 |                PX BLOCK ITERATOR |                  |  161 | 10304 |    86   (0)| 00:00:02 |  Q1,00 | PCW C |            |
|* 17 |                 TABLE ACCESS FULL| SALES_FACT       |  161 | 10304 |    86   (0)| 00:00:02 |  Q1,00 | PCW P |            |
-------------------------------------------------------------------------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
  17 - filter("PRODUCT"='Xtend Memory' AND "COUNTRY"='Australia')
Result Cache Information (identified by operation id):
------------------------------------------------------
"   1 - column-count=4; dependencies=(GAOXUAN.SALES_FACT); parameters=(nls); name=""select /*+parallel(4)*/ year, week, top_sale_year, lead( top_sale_year) over(order
by year desc) prev_top_sale_yer
  from (select"""
Note
-----
   - Degree of Parallelism is 4 because of hint
```

### PGA大小

**大部分与分析函数相关的运算都是在Program Global Area，PGA中进行的。因此，为了得到最优的性能，有一个足够大的内存区域能够不必使用硬盘来执行分析函数是很重要的。**

### 小结

```
select * from table(dbms_xplan.display_cursor('', '', 'ALLSTATS LAST'));
```