
# 高效SQL-Phase III

研发中心 曾庆典

GUANGZHOU KETENG INFORMATION TECHNOLOGY CO.,LTD.
PINGYUN RD,WEST HUANGPU AVE,GUANGZHOU

# 目录

（无小节要点提示）

# 第一章 分析函数

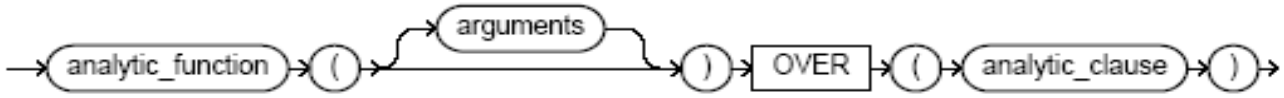
# Analytic functions:

在一個QUERY中, 它是除final ORDER BY clause外的最後執行的操作。即所有的 join / WHERE / GROUP BY / HAVING clauses都會先於它執行

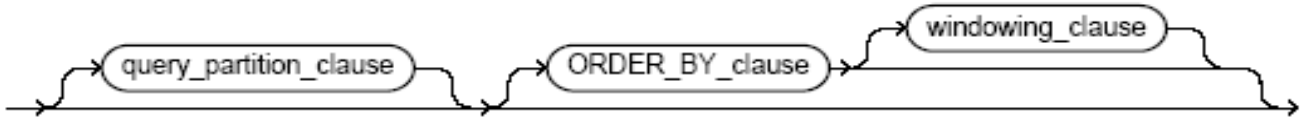
所以, analytic functions可以出現在select list 或 ORDER BY clause.

# 語法圖:

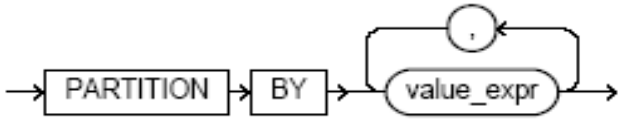
**analytic_function::=**




**analytic_clause::=**

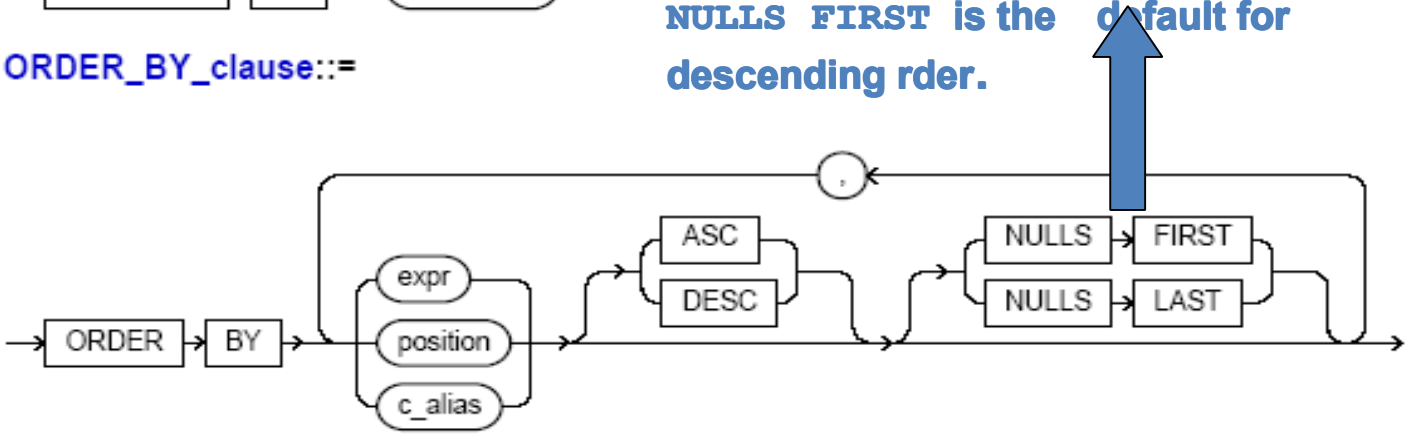



**query_partition_clause::=**




**ORDER_BY_clause::=**

**NULLS LAST is the default for ascending order,**
**NULLS FIRST is the default for descending rder.**



# Order By 的限制:

參與 **order by** 的**column**不允許在**order by clause**中使用**column alias / column position**的形式

而在一般**STATEMENT**的**ORDER BY**中，可以使用形如如下的形式：

**order by 1; / order by col_alias;**

# 常用分析函數的實例

## Enviroment:

---

DB WHL2.CAN1_FS
OWER WHSD
TABLE CNREC1014_TMP1

例1: SELECT 出 PAY_AMOUNT 由高到低, 数值在第5到第10的FN_OFFICE, BOOK_NO, PAY_AMOUNT的数據

形如：

| FN_OFFICE | BOOK_NO | PAY_AMOUNT | |
|---|---|---|---|
| HKHKG01 | 0243011484 | 3700 | =>第5 |
| CNSZP01 | 0243011484 | 3700 | =>第6 |
| CNCAN01 | 0243011705 | 1800 | =>第7 |
| CNSZP01 | 0243011705 | 1800 | =>第8 |
| HKHKG01 | 0243011705 | 1800 | =>第9 |
| CNCAN01 | 0243011517 | 1750 | =>第10 |

**Statement 1:**

```
SQL>   select fn_office,book_no,pay_amount,rn from
  2     (
  3      select fn_office,book_no,pay_amount, rownum rn
  4      from
  5      (select fn_office,book_no,pay_amount
  6       from cnrec1014_tmp1
  7       order by pay_amount desc
  8      )
  9      where rownum < 11
 10     )
 11    where rn > 4 ;

FN_OFFICE        BOOK_NO          PAY_AMOUNT            RN
---------------- ---------------- ---------- ----------
HKHKG01          0243010754             3700          5
CNSZP01                  0243011484           3700                6
CNCAN01          0243011705             1800          7
CNSZP01                  0243011705           1800                8
HKHKG01          0243011705             1800          9
CNCAN01          0243011517             1750         10

6 rows selected.
```

**Statement 1:**

Execution Plan

```
----------------------------------------------------------
0   SELECT STATEMENT Optimizer=CHOOSE (Cost=4 Card=219 Bytes=8760)
1 0   VIEW (Cost=4 Card=219 Bytes=8760)
2 1     COUNT (STOPKEY)
3 2       VIEW (Cost=4 Card=219 Bytes=5913)
4 3         SORT (ORDER BY STOPKEY) (Cost=4 Card=219 Bytes=4380)
5 4           TABLE ACCESS (FULL) OF 'CNREC1014_TMP1' (Cost=1 Ca
    rd=219 Bytes=4380)
```

Statistics

```
----------------------------------------------------------
        241  recursive calls
          4  db block gets
         30  consistent gets
          2  physical reads
          0  redo size
        440  bytes sent via SQL*Net to client
        251  bytes received via SQL*Net from client
          4  SQL*Net roundtrips to/from client
          6  sorts (memory)
          0  sorts (disk)
          6  rows processed
```

## Statement 2:

```
SQL> select fn_office,book_no,pay_amount
  2      from
  3      (select fn_office,book_no,pay_amount,
  4         row_number() over(order by pay_amount desc)
  5         rn from cnrec1014_tmp1
  6         )
  7      where rn between 5 and 10
  8     order by rn ;

FN_OFFICE          BOOK_NO            PAY_AMOUNT
------------------ ------------------ ----------
HKHKG01            0243011484              3700
CNSZP01            0243011484              3700
CNCAN01            0243011705              1800
CNSZP01            0243011705              1800
HKHKG01            0243011705              1800
CNCAN01            0243011517              1750

6 rows selected.
```

**Statement 2:**

Execution Plan

```
----------------------------------------------------------
0   SELECT STATEMENT Optimizer=CHOOSE (Cost=7 Card=219 Bytes=8760)
1 0   SORT (ORDER BY) (Cost=7 Card=219 Bytes=8760)
2 1     VIEW (Cost=4 Card=219 Bytes=8760)
3 2       WINDOW (SORT)
4 3         TABLE ACCESS (FULL) OF 'CNREC1014_TMP1' (Cost=1 Card
          =219 Bytes=4380)
```

Statistics

```
----------------------------------------------------------
        133  recursive calls
          4  db block gets
         15  consistent gets
          0  physical reads
          0  redo size
        404  bytes sent via SQL*Net to client
        251  bytes received via SQL*Net from client
          4  SQL*Net roundtrips to/from client
          6  sorts (memory)
          0  sorts (disk)
          6  rows processed
```

**Statement 1**需用**30**個**logical IO unit;** 而**statement 2**則需**15**個就可以了。

從上述例子看出，使用**analysis function**來實現某些統計數據，一般會好於使用**subquery statement**的**performance**

例**2:**

每一行的WHOLE的數值都是該行之前的PAY_AMOUNT的數值之和;
每一行的OFFICE的數值都是以FN_OFFICE分組後, 同一組中該行之前的PAY_AMOUNT的數值之和;
同時, 還要求每組中的每行都分別用SEQ標記其排行順序

形如:

| FN_OFFICE | BOOK_NO | PAY_AMOUNT | WHOLE | OFFICE | SEQ |
|---|---|---|---|---|---|
| CNCAN01 | 0243010271 | 700 | 700 | 700 | 1 |
| | 0243010372 | 23 | 723 | 723 | 2 |
| | 0243010686 | 780 | 1503 | 1503 | 3 |
| CNSZP01 | 0243010271 | 700 | 2203 | 700 | 1 |
| | 0243010372 | 23 | 2226 | 723 | 2 |
| | 0243010686 | 780 | 3006 | 1503 | 3 |
| HKHKG01 | 0243010271 | 700 | 3706 | 700 | 1 |
| | 0243010372 | 23 | 3729 | 723 | 2 |
| | 0243010686 | 780 | 4509 | 1503 | 3 |

# Statement 1 使用Subquery statement:

```
select fn_office,book_no,pay_amount,
      (select sum(a2.pay_amount) from cnrec1014_tmp1 a2
      where a2.fn_office < a1.fn_office
      or (a2.fn_office=a1.fn_office and a2.book_no<=a1.book_no)) whole_total,
     (select sum(a3.pay_amount) from cnrec1014_tmp1 a3
     where a3.fn_office=a1.fn_office and a3.book_no<=a1.book_no) office_total,
     (select count(a4.book_no) from cnrec1014_tmp1 a4   where a4.fn_office=a1.fn_office
 and a4.book_no<=a1.book_no) seq  from  cnrec1014_tmp1 a1

order by  fn_office,book_no  ;
Statistics
----------------------------------------------------------
          0  recursive calls
       2632  db block gets
       1316  consistent gets
          0  physical reads
          0  redo size
      11533  bytes sent via SQL*Net to client
        754  bytes received via SQL*Net from client
         32  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
        219  rows processed
```

## Statement 2 使用analysis function:

```
select fn_office,book_no,pay_amount,
  sum(pay_amount) over(order by fn_office,book_no) whole_total,
  sum(pay_amount) over(partition by fn_office order by book_no) office_total,
  row_number() over(partition by fn_office order by book_no) seq
  from cnrec1014_tmp1 order by fn_office,book_no ;

219 rows selected.

Statistics
----------------------------------------------------------
          0  recursive calls
          4  db block gets
          2  consistent gets
          0  physical reads
          0  redo size
      11549  bytes sent via SQL*Net to client
        754  bytes received via SQL*Net from client
         32  SQL*Net roundtrips to/from client
          1  sorts (memory)
          0  sorts (disk)
        219  rows processed
```

# 從例2看，
# 數據量較大時，有明顯的區別

常用的**analysis function**的例子**:**

**1.** sum(pay_amount) over(partition by fn_office order by book_no)

2. row_number() over(partition by fn_office order by book_no)

3.LEAD()  & LAG()

參考文檔:
H:\Hardware\CAI\ORACLE\ora8i_doc\DOC\index.pdf
=> **Oracle8*i* Server and Data Warehousing,**
**=>** SQL Reference
**=> Page 137 ( Character 4-7 ) : analysis function**

## 練習:

```
SQL> select year,month,exp
  2  from facts;

      YEAR      MONTH        EXP
---------- ---------- ----------
      2000          1    1000000
      2000          2    2000000
      2000          3    3000000
      2000          4    4000000
      2000          5    5000000
      2001          1    1110000
      2001          2    2220000
      2001          3    3330000
      2001          4    4440000
      2001          5    5550000
```

要求列出2000年和2001年的每月EXP　数據進行比照：

| LAST_YEAR | MONTH | LAST_YEAR_EXP | CURR_YEAR_EXP |
|---|---|---|---|
| 2000 | 1 | 1000000 | 1110000 |
| 2000 | 2 | 2000000 | 2220000 |
| 2000 | 3 | 3000000 | 3330000 |
| 2000 | 4 | 4000000 | 4440000 |
| 2000 | 5 | 5000000 | 5550000 |

# 第二章 Insert Append

# Insert /*+ append*/

- Insert append 用法为强制ORACLE使用直接路径加载（direct path load），直接在table HWM之上进行插入数据。比直接insert into省去了检查segment Freelist和省去写Cache的时间。 在大数据量插入的时候适用。

# Insert /*+ append*/

优点：

1.在大数据量插入时比insert into快(快主要由上述的两个原因)

2.减少REDO log的产生

# Insert /*+ append*/

缺点：

- 1.由于Insert Append时会加入一个排他锁，会阻塞表的所有DML操作。故只能有一个transaction可以使用，其他适用此table必须等待。在多线程操作的环境下不建议时候。
- 2.由于是在HWM之上进行插入，故会导致空间浪费的问题。也会导致碎片的产生。

# Insert /*+ append*/

可能会适应的场景：

- 存储过程中的一些临时TABLE：
  1.假如临时Table只有当前存储过程会使用，没有其他transaction会使用，故可忽略不能并发访问的缺点。
  2.临时TABLE在使用完后会立即被TRUNCATE,由于TRUNCATE时会清除碎片和HWM,故空间浪费的问题也可忽略。
  3.也可考虑再加上NOLOGGING参数更加减少REDO LOG。

# Insert /*+ append*/

缺点：

1. 由于Insert Append时会加入一个排他锁，会阻塞表的所有DML操作。故只能有一个transaction可以使用，其他适用此table必须等待。在多线程操作的环境下不建议时候。

- 2. 由于是在HWM之上进行插入，故会导致空间浪费的问题。也会导致碎片的产生。

# Insert /*+ append*/

e.g.

```
insert  /*+ append */ into
  P_FDM_MM_EXEC_SCHE_TRACE_TEMP a
  (project_id,…
  select MR.PROJECT_ID,...
    from
  LCAM.MM_REQUIREMENT_ITEM@LCAM
  MRI, LCAM.MM_REQUIREMENT@LCAM MR
   where MR.REQUIREMENT_ID =
  MRI.REQUIREMENT_ID
     AND MR.STATUS <> 'new';
```

# Insert append 前的statistics

| | |
|---|---|
| Unformatted Blocks ..... | 0 |
| FS1 Blocks (0-25) ..... | 0 |
| FS2 Blocks (25-50) ..... | 0 |
| FS3 Blocks (50-75) ..... | 0 |
| FS4 Blocks (75-100)..... | 44 |
| Full Blocks ..... | 48,554 |
| Total Blocks..... | 98,432 |
| Total Bytes..... | 806,354,944 |
| Total MBytes..... | 769 |
| Unused Blocks..... | 49,536 |
| Unused Bytes..... | 405,798,912 |
| Last Used Ext FileId..... | 53 |
| Last Used Ext BlockId..... | 214,025 |
| Last Used Block..... | 1,024 |

# Insert append 后的statistics

- Truncate 后的statistics，空间和碎片已清除

Unformatted Blocks ..... 0
FS1 Blocks (0-25) ..... 0
FS2 Blocks (25-50) ..... 0
FS3 Blocks (50-75) ..... 0
FS4 Blocks (75-100)..... 0
Full Blocks ..... 0
Total Blocks..... 8
Total Bytes..... 65,536
Total MBytes..... 0
Unused Blocks..... 5
Unused Bytes..... 40,960
Last Used Ext FileId..... 60
Last Used Ext BlockId..... 922,513
Last Used Block..... 3

# 谢谢聆听

## Thanks


智能电网·信息化整体解决专家

广州科腾信息技术有限公司