# Oracle 中 ROWNUM 的使用技巧

作者：fuyuncat
来源：WWW.HelloDBA.COM

**作者简介**

黄玮，男，99 年开始从事 DBA 工作，有多年的水利、军工、电信及航运行业大型数据库 Oracle 开发、设计和维护经验。

曾供职于南方某著名电信设备制造商——H 公司。期间，作为 DB 组长，负责设计、开发和维护彩铃业务的数据库系统。目前，H 公司的彩铃系统是世界上终端用户最多的彩铃系统。最终用户数过亿。

目前供职于某世界著名物流公司，负责公司的电子物流系统的数据库开发、维护工作。

msn: fuyuncat@hotmail.com
Email：fuyuncat@gmail.com

ROWNUM 是一种伪列，它会根据返回记录生成一个序列化的数字。利用 ROWNUM，我们可以生产一些原先难以实现的结果输出，但因为它是伪列的这个特殊性，我们在使用时也需要注意一些事项，不要掉入“陷阱”。下面就介绍一下它的使用技巧及注意事项。

# 1 特殊结果输出

利用 ROWNUM，我们可以做到一些特殊方式的输出。

## *1.1 Top N 结果输出*

我们如果希望取输出结果的前面几条数据，通过 ROWNUM 可以轻松实现：

```
SQL> select * from t_test4
  2  where rownum <= 5;

USERNAME                          USER_ID CREATED
------------------------------ ---------- ---------
WOW                                    71 26-APR-07
CS2                                    70 15-JAN-07
3                                      69 01-NOV-06
DMP                                    68 12-OCT-06
PROFILER                               67 05-SEP-06
```

但是，如果你希望对一个排序结果取 Top N 数据的话，使用 ROWNUM 存在一些“陷阱”，我们后面部分会介绍这些“陷阱”并且说明如何避免。

## 1.2 分页查询

利用 ROWNUM 对结果进行分页，下面返回结果中的第 6 到第 10 条记录：

```
SQL> select * from
  2  (
  3  select a.*, rownum as rn from css_bl_view a
  4  where capture_phone_num = '(1) 925-4604800'
  5  ) b
  6  where b.rn between 6 and 10;

6 rows selected.


Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=2770 Card=2183 Bytes =7166789)
   1    0   VIEW (Cost=2770 Card=2183 Bytes=7166789)
   2    1     COUNT
   3    2       TABLE ACCESS (FULL) OF 'CSS_BL_VIEW' (Cost=2770 Card=2183
Bytes=1305434)


Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
      29346  consistent gets
      29190  physical reads
          0  redo size
       7328  bytes sent via SQL*Net to client
        234  bytes received via SQL*Net from client
          4  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          5  rows processed
```

另外一种实现方式：

```
SQL> select * from css_bl_view a
  2  where capture_phone_num = '(1) 925-4604800'
  3  and rownum <= 10
  4  minus
  5  select * from css_bl_view a
  6  where capture_phone_num = '(1) 925-4604800'
  7  and rownum <= 5
  8  ;


Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=5920 Card=10 Bytes=8970)
   1    0   MINUS
   2    1     SORT (UNIQUE) (Cost=2960 Card=10 Bytes=5980)
   3    2       COUNT (STOPKEY)
   4    3         TABLE ACCESS (FULL) OF 'CSS_BL_VIEW' (Cost=2770 Card=2183
Bytes=1305434)
   5    1     SORT (UNIQUE) (Cost=2960 Card=5 Bytes=2990)
   6    5       COUNT (STOPKEY)
   7    6         TABLE ACCESS (FULL) OF 'CSS_BL_VIEW' (Cost=2770 Card=2183
Bytes=1305434)
```

```
Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
         62  consistent gets
         50  physical reads
          0  redo size
       7232  bytes sent via SQL*Net to client
        234  bytes received via SQL*Net from client
          4  SQL*Net roundtrips to/from client
          2  sorts (memory)
          0  sorts (disk)
          5  rows processed
```

第三种实现方式：

```
SQL> select * from
  2  (
  3  select a.*, rownum as rn from css_bl_view a
  4  where capture_phone_num = '(1) 925-4604800'
  5  and rownum <= 10
  6  ) b
  7  where b.rn > 5;


Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=2770 Card=10 Bytes=32830)
   1    0   VIEW (Cost=2770 Card=10 Bytes=32830)
   2    1     COUNT (STOPKEY)
   3    2       TABLE ACCESS (FULL) OF 'CSS_BL_VIEW' (Cost=2770 Card=2183
Bytes=1305434)


Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
         35  consistent gets
         30  physical reads
          0  redo size
       7271  bytes sent via SQL*Net to client
        234  bytes received via SQL*Net from client
          4  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          5  rows processed
```

这里特地将三种实现方式的查询计划及统计数据打印出来，大家可以比较一下 3 中方式的性能。

## *1.3 利用 ROWNUM 做分组子排序*

对于以下表 T_TEST4 的内容：

```
OWNER                                     NAME
-------------------------------------------------------
STRMADMIN                                 STREAMS_QUEUE
APARKMAN                                  JOB_QUEUE
```

```
SYS                                       AQ$_AQ_SRVNTFN_TABLE_E
SYS                                       AQ$_KUPC$DATAPUMP_QUETAB_E
APARKMAN                                  AQ$_JMS_TEXT_E
STRMADMIN                                 AQ$_STREAMS_QUEUE_TABLE_E
SYS                                       AQ$_SCHEDULER$_EVENT_QTAB_E
…
```

如果我们希望结果按照 OWNER 进行分组后，再对每组中成员进行编号，结果类似如下：

```
OWNER                                     NO NAME
-----------------------------------------------------
APARKMAN                                   1 JOB_QUEUE
                                           2 AQ$_JMS_TEXT_E
STRMADMIN                                  1 STREAMS_QUEUE
                                           2 AQ$_STREAMS_QUEUE_TABLE_E
SYS                                        1 AQ$_AQ_SRVNTFN_TABLE_E
                                           2 AQ$_KUPC$DATAPUMP_QUETAB_E
                                           3 AQ$_SCHEDULER$_EVENT_QTAB_E
…
```

在没有 ROWNUM 时要实现这样的功能会很复杂，但通过 ROWNUM 我们可以轻松实现：

```
SQL> SELECT DECODE(ROWNUM-min sno,0,a.owner,NULL) owner,DECODE(ROWNUM-
min_sno,0,1,rownum+1-min_sno) sno, a.name
  2  FROM (SELECT *
  3        FROM t_test8
  4        ORDER BY owner, name ) a,
  5       (SELECT owner, MIN(rownum) min_sno
  6        FROM( SELECT *
  7              FROM t_test8
  8              ORDER BY owner, name)
  9        GROUP BY owner) b
 10  WHERE a.owner=b.owner;

OWNER                                 SNO NAME
------------------------------ ---------- ------------------------------
APARKMAN                                1 JOB_QUEUE
                                        2 AQ$_JMS_TEXT_E
STRMADMIN                               1 STREAMS_QUEUE
                                        2 AQ$_STREAMS_QUEUE_TABLE_E
SYS                                     1 AQ$_AQ_SRVNTFN_TABLE_E
                                        2 AQ$_KUPC$DATAPUMP_QUETAB_E
                                        3 AQ$_SCHEDULER$_EVENT_QTAB_E
                                        4 AQ$_SCHEDULER$_JOBQTAB_E
                                        5 AQ$_STREAMS_QUEUE_TABLE_E
                                        6 AQ$_SYS$SERVICE_METRICS_TAB_E
                                        7 AQ$_AQ_EVENT_TABLE_E
                                        8 AQ$_AQ$_MEM_MC_E
                                        9 AQ$_ALERT_QT_E
                                       10 ALERT_QUE
                                       11 AQ_EVENT_TABLE_Q
                                       12 SYS$SERVICE_METRICS
                                       13 STREAMS_QUEUE
                                       14 SRVQUEUE
                                       15 SCHEDULER$_JOBQ
                                       16 SCHEDULER$_EVENT_QUEUE
                                       17 AQ_SRVNTFN_TABLE_Q
SYSMAN                                  1 AQ$_MGMT_NOTIFY_QTABLE_E
                                        2 MGMT_NOTIFY_Q
```

```
SYSTEM                                          1 DEF$_AQERROR
                                                2 DEF$_AQCALL
                                                3 AQ$_DEF$_AQERROR_E
                                                4 AQ$_DEF$_AQCALL_E
WMSYS                                           1 AQ$_WM$EVENT_QUEUE_TABLE_E
                                                2 WM$EVENT_QUEUE

29 rows selected.
```

# 2 性能

我们很多程序员在确认某个表中是否有相应数据时，喜欢加上 ROWNUM=1，其思路就是只要存在一条数据就说明有相应数据，查询就可以直接返回了，这样就能提高性能了。但是在 10G 之前，使用 ROWNUM=1 是不能达到预期的性能效果的，而是需要通过<2 或<=1 作为过滤条件才能达到预期效果，看以下查询计划：

```
SQL> select * from t_test1
  2  where object_id <100
  3  and rownum = 1;


Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=37 Card=1 Bytes=86)
   1    0   COUNT (STOPKEY)
   2    1     TABLE ACCESS (BY INDEX ROWID) OF 'T_TEST1' (Cost=37 Card=89
Bytes=7654)
   3    2       INDEX (RANGE SCAN) OF 'T_TEST1_PK' (UNIQUE) (Cost=2 Card=89)


Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
         62  consistent gets
          0  physical reads
          0  redo size
        654  bytes sent via SQL*Net to client
        234  bytes received via SQL*Net from client
          4  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          1  rows processed

SQL> select * from t_test1
  2  where object_id <100
  3  and rownum <= 1;


Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=37 Card=1 Bytes=86)
   1    0   COUNT (STOPKEY)
```

```
   2    1     TABLE ACCESS (BY INDEX ROWID) OF 'T_TEST1' (Cost=37 Card=89
Bytes=7654)
   3    2       INDEX (RANGE SCAN) OF 'T_TEST1_PK' (UNIQUE) (Cost=2 Card=89)


Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
          3  consistent gets
          0  physical reads
          0  redo size
        654  bytes sent via SQL*Net to client
        234  bytes received via SQL*Net from client
          4  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          1  rows processed

SQL> /


Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=37 Card=1 Bytes=86)
   1    0   COUNT (STOPKEY)
   2    1     TABLE ACCESS (BY INDEX ROWID) OF 'T_TEST1' (Cost=37 Card=89
Bytes=7654)
   3    2       INDEX (RANGE SCAN) OF 'T_TEST1_PK' (UNIQUE) (Cost=2 Card=89)


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

10G 以后，这个问题就被修正了：

```
SQL> select * from t_test1
  2  where rownum = 1;


Execution Plan
----------------------------------------------------------
Plan hash value: 536364188

---------------------------------------------------------------------------
| Id  | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |     1 |    86 |     2   (0)| 00:00:01 |
|*  1 |  COUNT STOPKEY     |         |       |       |            |          |
|   2 |   TABLE ACCESS FULL| T_TEST1 |     1 |    86 |     2   (0)| 00:00:01 |
---------------------------------------------------------------------------
```

```
Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter(ROWNUM=1)


Statistics
----------------------------------------------------------
          1  recursive calls
          0  db block gets
          4  consistent gets
          1  physical reads
          0  redo size
       1201  bytes sent via SQL*Net to client
        385  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          1  rows processed

SQL> select * from t_test1
  2  where rownum <= 1;


Execution Plan
----------------------------------------------------------
Plan hash value: 536364188

---------------------------------------------------------------------------
| Id  | Operation          | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
---------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |         |     1 |    86 |     2   (0)| 00:00:01 |
|*  1 |  COUNT STOPKEY     |         |       |       |            |          |
|   2 |   TABLE ACCESS FULL| T_TEST1 |     1 |    86 |     2   (0)| 00:00:01 |
---------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter(ROWNUM<=1)


Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
          4  consistent gets
          0  physical reads
          0  redo size
       1201  bytes sent via SQL*Net to client
        385  bytes received via SQL*Net from client
          2  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          1  rows processed
```

# 3 ROWNUM 的使用“陷阱”

由于 ROWNUM 是一个伪列，只有有结果记录时，ROWNUM 才有相应数据，因此对它的使用不能向普通列那样使用，否则就会陷入一些“陷阱”当中。

## 3.1 对 ROWNUM 进行>、>=、=操作

不能对 ROWNUM 使用>（大于 1 的数值）、>=（大于或等于 1 的数值）、=（大于或等于 1 的数值），否则无结果

```
SQL> select count(*) from css_bl_view a where rownum>0;

  COUNT(*)
----------
361928


SQL> select count(*) from css_bl_view a
  2  where rownum > 1;

  COUNT(*)
----------
         0
```

这是因为：

1、ROWNUM 是伪列，必须要要有返回结果后，每条返回记录就会对应产生一个 ROWNUM 数值；

2、返回结果记录的 ROWNUM 是从 1 开始排序的，因此第一条始终是 1；

这样，当查询到第一条记录时，该记录的 ROWNUM 为 1，但条件要求 ROWNUM>1，因此不符合，继续查询下一条；因为前面没有符合要求的记录，因此下一条记录过来后，其 ROWNUM 还是为 1，如此循环，就不会产生结果。上述查询可以通过子查询来替代：

```
SQL> select count(*)
  2  from
  3  (select BL_REF_CDE, rownum rn from css_bl_view)
  4  where rn > 1;

  COUNT(*)
----------
    361927
```

我们可以通过以下方式来实现对 ROWNUM 的>、=的查询：

查询 ROWNUM=5 的数据：

```
SQL> select object_id,object_name
  2  from (select object_id,object_name, rownum as rn from t_test1)
  3  where rn = 5;
```

```
 OBJECT_ID OBJECT_NAME
---------- ------------------------------
        29 C_COBJ#
```

查询 ROWNUM > 25 的数据：

```
SQL> select * from t_test4
  2  minus
  3  select * from t_test4
  4  where rownum <= 25;

USERNAME                          USER_ID CREATED
------------------------------ ---------- ---------
DIP                                    19 21-NOV-05
OUTLN                                  11 21-NOV-05
PUBLIC                              99999 18-JUL-07
SYS                                     0 21-NOV-05
SYSMAN                                 32 21-NOV-05
SYSTEM                                  5 21-NOV-05

6 rows selected.
```

## 3.2 ROWNUM 和 Order BY

要注意的是：在使用 ROWNUM 时，只有当 Order By 的字段是主键时，查询结果才会先排序再计算 ROWNUM，下面 OBJECT_ID 是表 T_TEST1 的主键字段：

```
SQL> select object_id,object_name from t_test1
  2  where rownum <= 5
  3  order by object_id;

 OBJECT_ID OBJECT_NAME
---------- ------------------------------
         2 C_OBJ#
         3 I_OBJ#
         4 TAB$
         5 CLU$
         6 C_TS#
```

但是，对非主键字段 OBJECT_NAME 进行排序时，结果就混乱了：

```
SQL> select object_id,object_name from t_test1
  2  where rownum <= 5
  3  order by object_name;

 OBJECT_ID OBJECT_NAME
---------- ------------------------------
        28 CON$
        29 C_COBJ#
        20 ICOL$
        44 I_USER1
        15 UNDO$

SQL> select count(*) from t_test1
  2  where object_name < 'CON$';

  COUNT(*)
----------
     21645
```

出现这种混乱的原因是：Oracle 先按物理存储位置（rowid）顺序取出满足 rownum 条件的记录，即物理位置上的前 5 条数据，然后在对这些数据按照 Order By 的字段进行排序，而不是我们所期望的先排序、再取特定记录数。

如果需要对非主键字段排序再去前 n 条数据，我们可以以以下方式实现：

```
SQL> select object_id,object_name
  2  from (select object_id,object_name from t_test1
  3        order by object_name)
  4  where rownum <= 5;

 OBJECT_ID OBJECT_NAME
---------- ------------------------------
     35489 /1000e8d1_LinkedHashMapValueIt
     35490 /1000e8d1_LinkedHashMapValueIt
     21801 /1005bd30_LnkdConstant
     21802 /1005bd30_LnkdConstant
     17205 /10076b23_OraCustomDatumClosur
```

## 3.3 排序分页

当对存在重复值的字段排序后再分页输出，我们很容易会陷入到另外一个“陷阱”。

请看以下例子，我们希望对 T_TEST1 的 OWNER 字段排序后，以每页输出 10 个结果的方式分页输出：

```
SQL> select owner, object_name from
  2  (select a.*, rownum as rn from
  3  (select owner, object_name from t_test1 order by owner) a
  4  where rownum <= 10)
  5  where rn >= 1;

OWNER                          OBJECT_NAME
------------------------------ ------------------------------
AFWOWNER                       AFWADAPTER
AFWOWNER                       AFWADAPTERCONFIGURATION
AFWOWNER                       AFWADAPTERCONFIGURATION_IDX1
AFWOWNER                       AFWADAPTERFQN_PK
AFWOWNER                       AFWADAPTERCONFIGURATION_PK
AFWOWNER                       AFWADAPTERCONFIGURATION_IDX2
AFWOWNER                       AFWSERVERCODE_PK
AFWOWNER                       AFWSERVER
AFWOWNER                       AFWADAPTERLOOKUP_IDX1
AFWOWNER                       AFWADAPTERLOOKUP

10 rows selected.

SQL> select owner, object_name from
  2  (select a.*, rownum as rn from
  3  (select owner, object_name from t_test1 order by owner) a
  4  where rownum <= 20)
  5  where rn >= 11;

OWNER                          OBJECT_NAME
------------------------------ ------------------------------
AFWOWNER                       AFWTOKENSTATUSCODE_PK
```

```
AFWOWNER                        AFWTOKENSTATUS
AFWOWNER                        AFWTOKENADMIN_IDX1
AFWOWNER                        AFWTOKENADMINCODE_PK
AFWOWNER                        AFWTOKENADMIN
AFWOWNER                        AFWTOKEN
AFWOWNER                        AFWSERVERCONFIGURATION_PK
AFWOWNER                        AFWSERVERCONFIGURATION
AFWOWNER                        AFWSERVER
AFWOWNER                        AFWADAPTERLOOKUP

10 rows selected.
```

仔细比较结果，你会发现“AFWSERVER”、“AFWADAPTERLOOKUP”在两次分页结果中都出现了。但是 OBJECT_NAME 在每个 OWNER 中的值是唯一的，说明这个输出结果是错误的，我们又陷入了一个“陷阱”。这是怎么回事呢，请先看下上述语句的查询计划：

```
SQL> select owner, object_name from
  2  (select a.*, rownum as rn from
  3  (select owner, object_name from t_test1 order by owner) a
  4  where rownum <= 20)
  5  where rn >= 11;

10 rows selected.


Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=205 Card=20 Bytes=94
          0)

   1    0   VIEW (Cost=205 Card=20 Bytes=940)
   2    1     COUNT (STOPKEY)
   3    2       VIEW (Cost=205 Card=30670 Bytes=1042780)
   4    3         SORT (ORDER BY STOPKEY) (Cost=205 Card=30670 Bytes=858760)
   5    4           TABLE ACCESS (FULL) OF 'T_TEST1' (Cost=42 Card=30670
Bytes=858760)
```

看下这个“陷阱”是怎么形成的。从查询计划上，我们可以注意到，对于子查询

```
select a.*, rownum as rn from
     (select owner, object_name from t_test1 order by owner) a
where rownum <= 20
```

优化器采用了“SORT (ORDER BY STOPKEY)”。

“SORT (ORDER BY STOPKEY)”不需要对所有数据进行排序，而是只要找出结果集中的按特定顺序的最前 N 条记录，一旦找出了这 N 条记录，就无需再对剩下的数据进行排序，而直接返回结果。这种算法我们可以视为是“快速排序”算法的变种。快速排序算法的基本思想是：先将数据分 2 组集合，保证第一集合中的每个数据都大于第二个集合中每个数据，然后再按这个原则对每个集合进行递归分组，直到集合的单位最小。在进行“SORT (ORDER BY STOPKEY)”时，首先找出 N 条数据（这些数据并没有做排序）放在第一组，保证第一组的数据都大于第二组的数据，然后只对第一组数据进行递归。

可以看到，基于这样的算法基础上，如果 N 的数值不同，数据的分组也不同（如 N=20 时，第一次分组比例为 12:8，然后继续递归；当 N=10 时，第一次分组比例为 3:7 … …），这样，在数据的排序字段值都相等时，输出结果的顺序就会因为 N 值不同而不同。

知道原因后，我们可以通过以下几种方法来避免这个“陷阱”。

1、让查询计划避免“SORT (ORDER BY STOPKEY)”，采用“SORT (ORDER BY)”，使数据排序不受 ROWNUM 的影响。但这样会使所有数据都做排序：

```
SQL> select owner, object_name from
  2  (select a.*, rownum as rn from
  3  (select owner, object_name, rowid from t_test1 order by owner) a)
  4  where rn <= 10
  5  and rn >= 1;

OWNER                          OBJECT_NAME
------------------------------ ------------------------------
AFWOWNER                       AFWADAPTER
AFWOWNER                       AFWADAPTERCONFIGURATION
AFWOWNER                       AFWADAPTERCONFIGURATION_IDX2
AFWOWNER                       AFWADAPTERCONFIGURATION_PK
AFWOWNER                       AFWADAPTERCONFIGURATION_IDX1
AFWOWNER                       AFWADAPTERFQN_PK
AFWOWNER                       AFWADAPTERLOOKUP_IDX1
AFWOWNER                       AFWSERVERCODE_PK
AFWOWNER                       AFWSERVERCONFIGURATION_IDX1
AFWOWNER                       AFWTOKENTYPECODE_PK

10 rows selected.

SQL> select owner, object_name from
  2  (select a.*, rownum as rn from
  3  (select owner, object_name, rowid from t_test1 order by owner) a)
  4  where rn <= 20
  5  and rn >= 11;

OWNER                          OBJECT_NAME
------------------------------ ------------------------------
AFWOWNER                       AFWTOKENTYPE
AFWOWNER                       AFWTOKENSTATUSCODE_PK
AFWOWNER                       AFWTOKENSTATUS
AFWOWNER                       AFWTOKENADMIN_IDX1
AFWOWNER                       AFWTOKENADMINCODE_PK
AFWOWNER                       AFWTOKENADMIN
AFWOWNER                       AFWTOKEN
AFWOWNER                       AFWSERVERCONFIGURATION_PK
AFWOWNER                       AFWTOKEN_PK
AFWOWNER                       AFWTOKEN_IDX6

10 rows selected.

SQL> set autot trace
SQL> select owner, object_name from
  2  (select a.*, rownum as rn from
  3  (select owner, object_name, rowid from t_test1 order by owner) a)
  4  where rn <= 20
  5  and rn >= 11;

10 rows selected.


Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=237 Card=30670 Bytes=1441490)
   1    0   VIEW (Cost=237 Card=30670 Bytes=1441490)
   2    1     COUNT
```

```
   3    2       VIEW (Cost=237 Card=30670 Bytes=1042780)
   4    3         SORT (ORDER BY) (Cost=237 Card=30670 Bytes=1073450)
   5    4           TABLE ACCESS (FULL) OF 'T_TEST1' (Cost=42 Card=30670
Bytes=1073450)
```

2、在排序时，加上一个或多个字段（如主键字段、ROWID），使排序结果具有唯一性：

```
SQL> select owner, object_name from
  2  (select a.*, rownum as rn from
  3  (select owner, object_name, rowid from t_test1 order by owner, object_id) a
  4  where rownum <= 10)
  5  where rn >= 1;

OWNER                          OBJECT_NAME
------------------------------ ------------------------------
AFWOWNER                       AFWADAPTER
AFWOWNER                       AFWADAPTERFQN_PK
AFWOWNER                       AFWADAPTERCONFIGURATION
AFWOWNER                       AFWADAPTERCONFIGURATION_PK
AFWOWNER                       AFWADAPTERCONFIGURATION_IDX1
AFWOWNER                       AFWADAPTERCONFIGURATION_IDX2
AFWOWNER                       AFWADAPTERLOOKUP
AFWOWNER                       AFWADAPTERLOOKUP_IDX1
AFWOWNER                       AFWSERVER
AFWOWNER                       AFWSERVERCODE_PK

10 rows selected.

SQL> select owner, object_name from
  2  (select a.*, rownum as rn from
  3  (select owner, object_name, rowid from t_test1 order by owner, object_id) a
  4  where rownum <= 20)
  5  where rn >= 11;

OWNER                          OBJECT_NAME
------------------------------ ------------------------------
AFWOWNER                       AFWSERVERCONFIGURATION
AFWOWNER                       AFWSERVERCONFIGURATION_PK
AFWOWNER                       AFWSERVERCONFIGURATION_IDX1
AFWOWNER                       AFWTOKEN
AFWOWNER                       AFWTOKEN_PK
AFWOWNER                       AFWTOKEN_IDX1
AFWOWNER                       AFWTOKEN_IDX2
AFWOWNER                       AFWTOKEN_IDX3
AFWOWNER                       AFWTOKEN_IDX4
AFWOWNER                       AFWTOKEN_IDX5

10 rows selected.

SQL> set autot trace
SQL> select owner, object_name from
  2  (select a.*, rownum as rn from
  3  (select owner, object_name, rowid from t_test1 order by owner, object_id) a
  4  where rownum <= 20)
  5  where rn >= 11;

10 rows selected.


Execution Plan
```

```
-----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=253 Card=20 Bytes=940)
   1    0   VIEW (Cost=253 Card=20 Bytes=940)
   2    1     COUNT (STOPKEY)
   3    2       VIEW (Cost=253 Card=30670 Bytes=1042780)
   4    3         SORT (ORDER BY STOPKEY) (Cost=253 Card=30670 Bytes=1196130)
   5    4           TABLE ACCESS (FULL) OF 'T_TEST1' (Cost=42 Card=306
          70 Bytes=1196130)
```

3、对排序字段建立索引，并强制使用索引。这样就能利用索引已经建立好的排序结果：

```
SQL> create index t_test1_idx1 on t_test1(owner);

Index created.

SQL> select owner, object_name from
  2  (select a.*, rownum as rn from
  3  (select /*+index(t T_TEST1_IDX1)*/owner, object_name from t_test1 t order by
owner) a
  4  where rownum <= 10)
  5  where rn >= 1
  6  ;

OWNER                          OBJECT_NAME
------------------------------ ------------------------------
AFWOWNER                       AFWADAPTER
AFWOWNER                       AFWADAPTERCONFIGURATION
AFWOWNER                       AFWADAPTERCONFIGURATION_IDX1
AFWOWNER                       AFWADAPTERCONFIGURATION_IDX2
AFWOWNER                       AFWADAPTERCONFIGURATION_PK
AFWOWNER                       AFWADAPTERFQN_PK
AFWOWNER                       AFWADAPTERLOOKUP
AFWOWNER                       AFWADAPTERLOOKUP_IDX1
AFWOWNER                       AFWSERVER
AFWOWNER                       AFWSERVERCODE_PK

10 rows selected.

SQL> select owner, object_name from
  2  (select a.*, rownum as rn from
  3  (select /*+index(t T_TEST1_IDX1)*/owner, object_name from t_test1 t order by
owner) a
  4  where rownum <= 20)
  5  where rn >= 11;

OWNER                          OBJECT_NAME
------------------------------ ------------------------------
AFWOWNER                       AFWSERVERCONFIGURATION
AFWOWNER                       AFWSERVERCONFIGURATION_IDX1
AFWOWNER                       AFWSERVERCONFIGURATION_PK
AFWOWNER                       AFWTOKEN
AFWOWNER                       AFWTOKENADMIN
AFWOWNER                       AFWTOKENADMINCODE_PK
AFWOWNER                       AFWTOKENADMIN_IDX1
AFWOWNER                       AFWTOKENSTATUS
AFWOWNER                       AFWTOKENSTATUSCODE_PK
AFWOWNER                       AFWTOKENTYPE

10 rows selected.

SQL> set autot trace
SQL> select owner, object_name from
```

```
  2  (select a.*, rownum as rn from
  3  (select /*+index(t T_TEST1_IDX1)*/owner, object_name from t_test1 t order by
owner) a
  4  where rownum <= 20)
  5  where rn >= 11;

10 rows selected.


Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=414 Card=20 Bytes=940)
   1    0   VIEW (Cost=414 Card=20 Bytes=940)
   2    1     COUNT (STOPKEY)
   3    2       VIEW (Cost=414 Card=30670 Bytes=1042780)
   4    3         TABLE ACCESS (BY INDEX ROWID) OF 'T_TEST1' (Cost=414Card=30670
Bytes=858760)
   5    4           INDEX (FULL SCAN) OF 'T_TEST1_IDX1' (NON-UNIQUE) (
          Cost=26 Card=30670)
```

以上就是 ROWNUM 的使用技巧及其注意事项，希望编程成员正确使用 ROWNUM，也希望 DBA 遇到相关问题能迅速定位。

## 3.4 性能陷阱

在之前，我们已经介绍过 ROWNUM 的使用及一些要注意的问题。这里，再介绍一种由 ROWNUM 导致的性能陷阱。事实上，如果能清楚了解 ROWNUM 的基本机制，那么对这个陷阱就很容易理解。

我们有定义如下视图，

```
CREATE OR REPLACE VIEW MY_VIEW AS
SELECT ROWNUM AS ID,
       T0.COMP_ID AS COMPANY_ID,
       T0.SH_ID AS B_NUM,
       '' AS L_NUM,
       '' AS C_NUM,
       T0.C_COMP_ID AS SP_COMP_ID,
       '' AS C_TYPE,
       T0.REC_UPD_DT AS LAST_MOD_DT
FROM T_COM T0
WHERE T0.DELETED_IND = 0
AND T0.VIEW_TYPE = 'BK'
```

在引用表 T_COM 的(COMP_ID, VIEW_TYPE, DELETED_IND, SH_ID)已经建立索引，且 COMP_ID 的选择性很强，VIEW_TYPE 和 DELETED_IND 选择性则很弱。

再看下面的查询

```
select id,
       company_id,
       c_num,
       b_num,
       l_num,
       last_mod_dt,
```

```
       c_type,
       sp_comp_id
from MY_VIEW
where company_id = '1111111111'
```

嗯，优化器应该先进行查询重写（Predicate Pushing），将主查询中的 company_id = ’1111111111’与视图中的 T0.DELETED_IND = 0 AND T0.VIEW_TYPE = ’BK’共同作用到表 T_COM 上，命中到索引。但是，是这样吗，看看它的查询计划：

```
-------------------------------------------------------------------------------
---------
| Id  | Operation            | Name              | Rows  | Bytes | Cost (%CPU)|
Time     |
-------------------------------------------------------------------------------
---------
|   0 | SELECT STATEMENT     |                   |   397K|    26M| 49582   (2)|
00:09:55 |
|*  1
|  FILTER              |                   |       |       |            |
  |
|*  2 |   VIEW               | MY_VIEW           |   397K|    26M| 49582   (2)|
00:09:55 |
|   3
|    COUNT             |                   |       |       |            |
  |
|*  4 |     TABLE ACCESS FULL| T_COM             |   397K|    23M| 49582   (2)|
00:09:55 |
-------------------------------------------------------------------------------
---------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("COMPANY_ID"='111111111')
   3 - filter("T0"."DELETED_IND"=0 AND "T0"."VIEW_TYPE"='BK')
```

并未命中索引。仔细分析视图的定义语句，其中包含了对 ROWNUM 的查询。ROWNUM 是一个虚字段，只有产生结果集时才会有值。因此，为保证逻辑结果，优化器并没有将视图查询条件与外部主查询条件合并后再执行查询操作，而是先执行子查询部分（条件不足以命中索引），得到结果集（计划中的第一步 Full Table Scan）和对应的 ROWNUM 值（计划中的第二步 COUNT）以后再根据外部条件对结果集过滤（计划中的第三步 FILTER）。当我们将 ROWNUM 从视图中拿掉，执行计划就能像我们之前所预想的命中索引了。

```
CREATE OR REPLACE VIEW MY_VIEW AS
SELECT 1 AS ID,
       T0.COMP_ID AS COMPANY_ID,
       T0.SH_ID AS B_NUM,
       '' AS L_NUM,
       '' AS C_NUM,
       T0.C_COMP_ID AS SP_COMP_ID,
       '' AS C_TYPE,
       T0.REC_UPD_DT AS LAST_MOD_DT
FROM T_COM T0
WHERE T0.DELETED_IND = 0
AND T0.VIEW_TYPE = 'BK'
```

```
-----------------------------------------------------------------------------------
---------
| Id  | Operation                   | Name       | Rows  | Bytes | Cost (%CPU)|
Time     |
-----------------------------------------------------------------------------------
--------
|   0 | SELECT STATEMENT            |            |   299 | 20631 |   340   (1)|
00:00:05 |
|   1 |  TABLE ACCESS BY INDEX ROWID| T_COM      |   299 | 20631 |   340   (1)|
00:00:05 |
|*  2 |   INDEX RANGE SCAN          | T_COM_INDX4|   299 |       |    77   (0)|
00:00:01 |
-----------------------------------------------------------------------------------
---------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("T0"."COMP_ID"='111111111' AND "T0"."DELETED_IND"=0 AND
              "T0"."VIEW_TYPE"='BK')
       filter("T0"."DELETED_IND"=0 AND "T0"."VIEW_TYPE"='BK')
```

如果开发人员不深入理解 ROWNUM 的机制，而简单的将其理解为一个子增长的字段的话，就很容易掉如这样的陷阱了。

补充：看看假如在子查询或视图中存在 ROWNUM 时，查询重写和不重写的逻辑结果的差异吧。

```
SQL> select id, owner, object_name
  2  from (select rownum as id, owner, object_name from t_test1 where object_name
like 'TEM%') v
  3  where owner='DEMO';

        ID OWNER                                              OBJECT_NAME
---------- -------------------------------------------------- -------------------
-----
         5 DEMO                                               TEMP_BOOK_CMP
SQL>
SQL> select id, owner, object_name
  2  from (select rownum as id, owner, object_name from t_test1 where object_name
like 'TEM%' and owner='DEMO') v;

        ID OWNER                                              OBJECT_NAME
---------- -------------------------------------------------- -------------------
-----
         1 DEMO                                               TEMP_BOOK_CMP
```

真实字段的内容没有不同，但是由 ROWNUM 返回的数据就不同了。

--- Fuyuncat ---