# 第 2 章　Yangtingkun 的 DBA 工作手记

编者题记：老杨在网络上的博客数年来一直维持着频繁更新的记录，积累了近千篇的技术文章，在本书的组稿过程中，老杨将自己多年来的积累整理、精炼、讲解出来，与读者们分享他技术上的结晶。这一章是为老杨的 DBA 手记。DBA 的工作可以苦中作乐，也可以乐在其中，让我们首先来看看老杨如何用 SQL 来为我们解释一个魔术-----

## 1.1利用 SQL 解释一个魔术

一个偶然的机会在电视上看到一个有关扑克牌的魔术，觉得很有意思。这个魔术明显不是靠手快或者作假来实现的，奥妙在于魔术中包含了数学原理。

首先描述一下这个魔术，有兴趣的话，可以按照这个方法试一试。

从一副扑克牌中随意抽取 21 张牌。让观众从这些牌中随意选择一张，这张牌就是最后通过魔术需要找到的目标牌。让观众牢记后将其放回到其余 20 张牌中，然后任意洗牌。

下面开始进行发牌的工作，发牌和普通扑克的发牌规则一样。将牌发成 3 叠，每叠 7 张。将每叠牌依次展示给观众，要求观众确认目标牌在 3 叠的哪一叠中即可。

之后将 3 叠牌合在一起，将包含目标牌的一叠放在其他两叠牌中间。注意此时不要打乱每叠牌的顺序。

然后再次发牌，和刚才完全一样，还是将牌发成 3 叠。让对方确认目标牌所在的一叠，将这叠牌放到另外两叠牌的中间。

最后，再次重复上面的发牌、确认的过程，仍然将包含目标牌的那叠牌，放到另外两叠牌的中间。

下面神奇的时刻到来了：从这叠扑克牌的上面每次拿起一张，每拿起一张牌的同时要说一个字：**“你要相信魔术你的牌是”**。说完这句话，下一张牌就是目标牌了。

看上去这个魔术很神奇，而且最神奇的是，这个魔术任何人都可以来表演。这就说明无论这张牌最初在哪个位置，只要按照这个规则最后都一定会来到这个指定的位置。

看了这个魔术，不禁有点手痒，既然是 DBA 出身，就用 SQL 来演示一下这个魔术的过程吧：

```
SQL> WITH A AS
  2  (SELECT ROWNUM P FROM DUAL CONNECT BY LEVEL <= 21)
  3  SELECT
  4   7 + CEIL(
  5    (7 + CEIL(
  6     (7 + CEIL(P/3))
  7    /3))
  8   /3)
```

```
  9  FROM A;
7+CEIL((7+CEIL((7+CEIL(P/3))/3))/3)
-----------------------------------
                                 11
                                 11
                                 11
                                 11
                                 11
                                 11
                                 11
                                 11
                                 11
                                 11
                                 11
                                 11
                                 11
                                 11
                                 11
                                 11
                                 11
                                 11
                                 11
                                 11
已选择 21 行。
```

简单的解释一下这个 SQL：WITH 查询中的 P 表示这张牌的初始位置，这个位置的取值范围是从 1 到 21。而后将牌按照发牌的顺序分成 3 份，于是这张牌的位置变为 CEIL(P/3)。由于在魔术表演的过程中，目标牌所在的那一叠牌会放在其他两叠牌的中间，也就是说目标牌的前面增加了 7 张牌，因此目标牌的位置要增加 7。

魔术中上面的步骤重复了 3 次，因此在 SQL 中这个过程也重复 3 次，最终 SQL 返回的结果就是目标牌 21 种不同初始位置所对应的魔术结束时刻目标牌的最终位置。

根据计算结果可以看到，无论这张牌在哪里，最终都会达到第十一张的位置。这也就是这个魔术的奥秘之所在。

不过 SQL 只是演示了结果，并没有给出为什么会出现这种结果，下面通过数学手段简单分析一下：

由于第一次平均分的时候这张牌的位置是任意的，所以这次平均分的意义不大。这次平均分的目的只是将目标牌的那一份放到中间的位置。所以可以认为这张牌在中间位置第 1 到 7 的任何一个位置上。，因此这张牌的位置就是 7 + p。下面将牌分成三份，然后将目标牌堆放到中间，这时这张牌的位置变为 7 + (7+p)/3。最后再重复一次这个动作，最终结果变为：7 +

(7 + (7+p)/3)/3。

对上面的表达式进行通分计算后，结果变成(7*9 + 7*3 + 7 + p)/9，进一步简化变成(91 + p)/9，最后变成了 10 + (1+p)/9，而 p 的位置是 1 到 7，也就是说无论取何值，(1+p)/9 都不会大于 1，所以最终的结果是 11。

最后，应该修改一下魔术中咒语：**“你要相信数学你的牌是”**。

## 1.2ORA-600(17069)错误的解决过程

在一个报表数据库后台发现了这个错误,详细的错误信息为：

```
Fri Feb 20 08:16:44 2009
Errors in file /u1/oracle/admin/repdb01/bdump/repdb01_j015_5099.trc:
ORA-00600: internal error code, arguments: [17069], [0x6A5DEE1E0], [], [], [], [], [], []
Fri Feb 20 08:16:47 2009
Errors in file /u1/oracle/admin/repdb01/bdump/repdb01_j015_5099.trc:
ORA-00600: internal error code, arguments: [17069], [0x6A5DEE1E0], [], [], [], [], [], []
```

进一步检查对应的 trace 文件：

```
bash-2.03$ more /u1/oracle/admin/repdb01/bdump/repdb01_j015_5099.trc
/u1/oracle/admin/repdb01/bdump/repdb01_j015_5099.trc
Oracle9i Enterprise Edition Release 9.2.0.4.0 - 64bit Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.4.0 - Production
ORACLE_HOME = /data/oracle/product/920
System name:    SunOS
Node name:      newreport
Release:        5.8
Version:        Generic_117350-26
Machine:        sun4u
Instance name: repdb01
Redo thread mounted by this instance: 1
Oracle process number: 35
Unix process pid: 5099, image: oracle@newreport (J015)
*** SESSION ID:(12.28191) 2009-02-20 08:16:44.060
*** 2009-02-20 08:16:44.060
ksedmp: internal or fatal error
ORA-00600: internal error code, arguments: [17069], [0x6A5DEE1E0], [], [], [], [], [], []
Current SQL statement for this session:
DECLARE job BINARY_INTEGER := :job; next_date DATE := :mydate;  broken BOOLEAN := FALSE; BEGIN
P_GENERATE_REPDATA('FR20T000002000000
```

```
0000032'); :mydate := next_date; IF broken THEN :b := 1; ELSE :b := 0; END IF; END;
----- Call Stack Trace -----
calling              call     entry               argument values in hex
location             type     point               (? means dubious value)
-------------------- -------- ------------------- ----------------------------
ksedmp()+328         CALL     ksedst()+0          FFFFFFFF7FFF6430 ?
                                                  000000000 ? 000000000 ?
                                                  00000003E ?
                                                  FFFFFFFF7FFF6CC8 ?
                                                  1031D56C8 ?
kgeriv()+208         PTR_CALL 0000000000000000    000000000 ? 000103400 ?
                                                  0001035D9 ? 000102C00 ?
                                                  1035D9000 ? 1035D9C28 ?
```

从 trace 文件包含的进程名称 j015 来看，导致问题是一个 JOB。从 trace 文件中包含的错误语句则更加证实了这一点。由于这个 JOB 在数据库中已经运行了很长时间，一直没有出现过错误。现在运行报错，肯定是由于其他的外部原因导致 JOB 运行的异常。

大部分的ORA-600错误在metalink上都有详细的描述，于是查询了metalink：文档Doc ID: 39616.1 中汇总了 ORA-600（17069）错误的已知 bug，不过这些 bug 的描述都与当前问题的现象并不太相符。不过文档中还是包含了一些有价值的信息。文档中描述 ORA-600(17069）错误的第二个参数代表 Library Cache Object Handle，这里的值为 0x6A5DEE1E0。

看起来问题可能和 LATCH 有关，但是根据错误信息所显示的地址，在 V$LATCH 和 V$LATCH_CHILDREN 视图中都没有找到有价值的信息。

从错误信息这个方向入手找不到什么有价值的信息了，现在只能回过头来检查发生错误的 JOB。根据 JOB 的特性，在运行失败后这个 JOB 会自动再次执行，检查 JOB 运行时的 V$LOCK 信息：

```
SQL> SELECT ADDR, TYPE, ID1, ID2, LMODE, REQUEST, BLOCK
  2  FROM V$LOCK  WHERE SID = 75;
ADDR             TY         ID1         ID2      LMODE    REQUEST      BLOCK
---------------- -- ---------- ---------- ---------- ---------- ----------
0000000690342780 CU -1.703E+09          6          6          0          0
00000006903426F8 JQ          0         63          6          0          0
```

在 V$LOCK 中没有什么特别的信息，接着检查 V$SESSION_WAIT，看看这个 JOB 在等待什么：

```
SQL> SELECT EVENT, P1TEXT, P1RAW, P2TEXT, P2RAW, STATE
  2  FROM V$SESSION_WAIT  WHERE SID = 75;
EVENT             P1TEXT          P1RAW            P2TEXT       P2RAW            STATE
----------------- --------------- ---------------- ------------ ---------------- -------
library cache pin handle address  00000006A5DEE1E0 pin address  00000006B1A971A8 WAITING
```

很明显，查询结果中 P1RAW 的值就是 ORA-600(17069)错误的第二个参数，配合等待事件信息基本上可以确定问题就是出现在 LIBRARY CACHE PIN 的过程中。再次查看 METALINK 信息，

Oracle 指出这个错误的原因多半是：**运行时间很长的 PROCEDURE 在执行过程中，所依赖的对象被编译或者删除了**。

检查出错 JOB 所调用过程的状态：

```
SQL> SELECT OWNER, OBJECT_NAME, OBJECT_TYPE, STATUS
  2  FROM DBA_OBJECTS
  3  WHERE OWNER = 'FUJIANREP'
  4  AND OBJECT_NAME = 'P_GENERATE_REPDATA';
OWNER                          OBJECT_NAME                    OBJECT_TYPE         STATUS
------------------------------ ------------------------------ ------------------- -------
FUJIANREP                      P_GENERATE_REPDATA             PROCEDURE           INVALID
```

果然，出错过程的状态是不正常的。在修正错误前，首先将 JOB 至于 BROKEN 状态以避免 JOB 再次运行：

```
SQL> EXEC DBMS_JOB.BROKEN(63, TRUE)
PL/SQL procedure successfully completed.
SQL> COMMIT;
Commit complete.
```

在操作系统级杀掉 JOB 对应的 PROCESS：

```
SQL> SELECT SPID FROM V$PROCESS WHERE ADDR IN (SELECT PADDR FROM V$SESSION WHERE SID = 75);
SPID
------------
14927
SQL> HOST kill -9 14927
```

现在 JOB 调用已经被终止，可以手工重新编译过程了：

```
SQL> ALTER PROCEDURE P_GENERATE_REPDATA COMPILE;
ALTER PROCEDURE P_GENERATE_REPDATA COMPILE
*
ERROR at line 1:
ORA-04021: timeout occurred while waiting to lock object FUJIANREP.P_GENERATE_REPDATA
```

编译报错，错误信息指出没有获得编译对象所需的锁，而导致超时错误的发生。

由于从 V$LOCK 和 V$LATCH 视图中都无法获得有意义的信息，只能检查是否有其他人当前在访问 P_GENERATE_REPDATA 所依赖的对象：

```
SQL> SELECT * FROM V$ACCESS
  2  WHERE (OWNER, OBJECT) IN
  3  (SELECT REFERENCED_OWNER, REFERENCED_NAME FROM DBA_DEPENDENCIES
  4  WHERE OWNER = 'FUJIANREP' AND NAME = 'P_GENERATE_REPDATA');
       SID OWNER                          OBJECT                         TYPE
---------- ------------------------------ ------------------------------ -------------
        54 FUJIANREP                      CAT_BUYER                      SYNONYM
        54 FUJIANREP                      CAT_CATEGORY                   SYNONYM
```

| | | |
|---|---|---|
| 54 | FUJIANREP | CAT_DOSEAGE_FORM | SYNONYM |
| 54 | FUJIANREP | CAT_DRUG | SYNONYM |
| 54 | FUJIANREP | CAT_ENTERPRISE | SYNONYM |
| 54 | FUJIANREP | CAT_METRIC | SYNONYM |
| 54 | FUJIANREP | CAT_ORG | SYNONYM |
| 54 | FUJIANREP | CAT_PRODUCT | SYNONYM |
| 54 | FUJIANREP | CAT_QUALITY_DEFINE | SYNONYM |
| 54 | FUJIANREP | GOV_CAT_BUYER | TABLE |
| 54 | FUJIANREP | GOV_CAT_ENTERPRISE | TABLE |
| 54 | FUJIANREP | GOV_S_MO_BU | TABLE |
| 54 | FUJIANREP | GOV_S_MO_BU_EN | TABLE |
| 54 | FUJIANREP | GOV_S_MO_BU_PR | TABLE |
| 54 | FUJIANREP | GOV_S_MO_EN | TABLE |
| 54 | FUJIANREP | GOV_S_MO_ME | TABLE |
| 54 | FUJIANREP | GOV_S_MO_ME_CA | TABLE |
| 54 | FUJIANREP | GOV_S_MO_ME_PR | TABLE |
| 54 | FUJIANREP | GOV_S_MO_ORDER | TABLE |
| 54 | FUJIANREP | GOV_S_YE_ORDER | TABLE |
| 54 | FUJIANREP | GRP_HOSPITAL | TABLE |
| 54 | FUJIANREP | GRP_LEVEL | TABLE |
| 54 | FUJIANREP | ORD_ORDER | TABLE |
| 54 | FUJIANREP | ORD_ORDER_ITEM | TABLE |
| 54 | FUJIANREP | ORD_ORDER_ITEM_REP | CURSOR |
| 54 | FUJIANREP | ORD_ORDER_RECEIVE | TABLE |
| 54 | FUJIANREP | ORD_ORDER_RECEIVE_REP | SYNONYM |
| 54 | FUJIANREP | ORD_ORDER_REP | CURSOR |
| 54 | FUJIANREP | ORD_ORDER_RETURN | TABLE |
| 54 | FUJIANREP | ORD_ORDER_RETURN_REP | CURSOR |
| 54 | FUJIANREP | PLT_PLAT | CURSOR |
| 54 | FUJIANREP | USER_TAB_PARTITIONS | CURSOR |
| 54 | NDMAIN | CAT_BUYER | TABLE |
| 54 | NDMAIN | CAT_CATEGORY | TABLE |
| 54 | NDMAIN | CAT_DOSEAGE_FORM | TABLE |
| 54 | NDMAIN | CAT_DRUG | TABLE |
| 54 | NDMAIN | CAT_ENTERPRISE | TABLE |
| 54 | NDMAIN | CAT_METRIC | TABLE |
| 54 | NDMAIN | CAT_ORG | TABLE |
| 54 | NDMAIN | CAT_PRODUCT | TABLE |
| 54 | NDMAIN | CAT_QUALITY_DEFINE | TABLE |

```
        54 NDMAIN                          ORD_ORDER                        VIEW
        54 NDMAIN                          ORD_ORDER_ITEM                   VIEW
        54 NDMAIN                          ORD_ORDER_RECEIVE                VIEW
        54 NDMAIN                          ORD_ORDER_RETURN                 VIEW
        54 NDMAIN                          PLT_PLAT                         TABLE
        54 PUBLIC                          USER_TAB_PARTITIONS              SYNONYM
        54 SYS                             STANDARD                         PACKAGE
       145 SYS                             STANDARD                         PACKAGE
        54 SYS                             SYS_STUB_FOR_PURITY_ANALYSIS     PACKAGE
        54 SYS                             USER_TAB_PARTITIONS              VIEW
51 rows selected.
```

过程依赖的对象果然被其他人所访问，检查这个会话的信息：

```
SQL> SELECT SID, SERIAL#, USERNAME, PROGRAM, TERMINAL
  2  FROM V$SESSION WHERE SID = 54;
       SID    SERIAL# USERNAME                       PROGRAM      TERMINAL
---------- ---------- ------------------------------ ------------ ----------
        54      26216 FUJIANREP                      PlSqlDev.exe LIBY
```

没想到是同事通过 pldevelop 连接的会话，看看这个会话在做什么：

```
SQL> SELECT SQL_TEXT FROM V$SQL
  2  WHERE ADDRESS IN (SELECT SQL_ADDRESS FROM V$SESSION WHERE SID = 54);
SQL_TEXT
----------------------------------------------------------------
ALTER TABLE GOV_S_MO_EN TRUNCATE PARTITION P200901
```

居然是 TRUNCATE 分区的操作，难怪会导致过程处于 INVALID 状态，不过 TRUNCATE 操作应该不会持续很长时间，而导致问题产生的语句理论上应该已经运行很久了：

```
SQL> SELECT EVENT, P1TEXT, P1, P2TEXT, P2, P3TEXT, P3, SECONDS_IN_WAIT
  2  FROM V$SESSION_WAIT WHERE SID = 54;
EVENT                     P1TEXT    P1 P2TEXT         P2 P3TEXT     P3 SECONDS_IN_WAIT
------------------------- -------- ---- --------- --------- --------- ---- ---------------
db file sequential read   file#      1 block#     170158 blocks      1         3995643
```

这个 TRUNCATE 的等待时间已经超过十天了，很显然这是一个僵死的会话。从后台 kill 掉对应的进程：

```
SQL> SELECT SPID FROM V$PROCESS WHERE ADDR IN (SELECT PADDR FROM V$SESSION WHERE SID = 54);
SPID
------------
12974
SQL> HOST kill -9 12974
```

切换为 FUJIANREP 用户，再次编译过程：

```
SQL> ALTER PROCEDURE P_GENERATE_REPDATA COMPILE;
```

```
Procedure altered.
```

至此问题解决。将 JOB 重新设置 BROKEN 即可。

```
SQL> EXEC DBMS_JOB.BROKEN(63, FALSE)
PL/SQL procedure successfully completed.
SQL> COMMIT;
Commit complete.
```

问题解决后再次检查过程，发现 TRUNCATE 语句居然就是这个过程的一部分。这个过程会先执行 TRUNCATE，然后执行插入等 DML 语句。

那么所有问题都搞清楚了：问题出在手工执行出错的过程中，可能由于网络的原因导致客户端与数据库端失去联系，数据库中的会话变成僵死状态停在了 TRUNCATE TABLE 语句处。而执行者只是中止了客户端的请求，并没有意识到后台进程的问题。

等到 JOB 的运行时间一到，JOB 尝试再次运行相同的过程，但是发现过程处于运行状态，且对一些表持有 LOCK 和 LATCH，于是引发了上面的 ORA-600(17079)错误。这也是随后手工编译 PROCEDURE 报错 ORA-4021 的原因。

## 1.3V$SQL 视图显示结果异常的诊断

碰到一个很奇怪的问题，在检查会话所执行的 SQL 时，发现 V$SQL 视图中 SQL_TEXT 列中的数据是不正常的。

由于 V$SQL 是动态性能视图，里面保存的是当前共享池中加载的 SQL 语句，所以如果这个 SQL 不是执行很频繁的话，那么它很可能会被替换出共享池。或者数据库意外重启，也会导致这个 SQL 彻底的丢失。那么首要任务就是保留现场，一旦错误不可再现，那么所有的问题就都无从查起了。

将显示异常的 V$SQL 记录备份到了 BAK_V$SQL 表中，首先看一下异常的 SQL 语句：

```
SQL> SELECT SQL_TEXT FROM BAK_V$SQL;
SQL_TEXT
--------------------------------------------------------------------------------
        info.CONTRACT_ITEM_ID,info.BUYER_ORG_ID a
          el未承诺'DISCOUNT_STEP is null or INC.NUM_STEP is null then
           to_char(INC.DISCOUNT_STEP * 100,'0.0') || '%'劭勐?' ||
           '未 CASH.CASH is null and CASH.CASH_THIRTY is null then
           '现款:' || to_char(CASH.CASH * 100,'0.0'
           '30日结款:'SH_THIRTY is not null then
 case whend end as CASH_DISCOUNT,0,'0.0') || '%;'
en info.modify_date > inc.modify_date and info.modify_date > cash.modify_date then info.mo
```

显然这个 SQL 语句是不正常的，语句中甚至连 SELECT、INSERT、UPDATE、DELETE 命令都不包括。但是这个 SQL 又不完全是乱码，从显示的部分看，大部分是有一定逻辑在里面的。

观察这个 SQL，第一感觉像是 V$SQL 视图中显示了部分 SQL，而没有从 SQL 语句的开头部

分开始显示。当然即使是部分 SQL，也不是连续显示的，因为连续的两行并不连贯。

首先想到的就是 Oracle 的 bug，因为除了这一点，暂时没有想到其他的原因来解释这个现象。那么如果确实是由于 bug，导致 V$SQL 显示不完整，那么完整的 SQL 又是什么呢，是否完整的 SQL 也会存在问题呢。刚才一直在查询 V$SQL 视图，下面不妨查询一下 V$SQLTEXT_WITH_NEWLINES 视图，看看这个视图中的结果是否也是不正常的。并且可以对比两个视图的结果，看看从中能否找到一点线索。

```
SQL> SELECT SQL_TEXT FROM V$SQLTEXT_WITH_NEWLINES
  2  WHERE HASH_VALUE IN (SELECT HASH_VALUE FROM BAK_V$SQL) ORDER BY PIECE;
SQL_TEXT
----------------------------------------------------------------

 SELECT * FROM (
SELECT ROWNUM as numrow, yy.* from ( select


info.record_id,
       info.CONTRACT_ITEM_ID,info.BUYER_ORG_ID
..............
WHERE numrow >= 1
32 rows selected.
```

出乎意料的是，从 V$SQLTEXT_WITH_NEWLINES 中查询，发现 SQL 的结果是正常的，而且 V$SQL 中显示的内容在 V$SQLTEXT_WITH_NEWLINES 中都可以找到，只不过不是连续的。从这一点上看，似乎更可以肯定是 bug 了。不过还存在几个疑点：

首先在 Metalink 上没有发现类似的描述，难道是一个还没有发现的 bug。

其次对比两个视图的结果，二者的差异完全没法解释。V$SQL 视图中的 SQL_TEXT 截取了全部 SQL 语句的前 1000 个字节,而 V_SQLTEXT_WITH_NEWLINES 视图则包含全部的 SQL 内容,但是查询结果是分行显示的。没有道理完全显示结果是正常的，而显示前 1000 个字节就出现错误。而且错误出现的那么离谱，很多信息都是跳着显示的。

关键的是，找不到引发这个 bug 的原因。如果确实是显示问题，那么应该所有的 SQL 都会有问题。如果仅仅是这个 SQL 有问题，那么多半问题出在这个 SQL 本身。

V$SQL 中的 SQL_TEXT 字段长度为 1000，对于长度大于 1000 的 SQL，会显示前面 1000 个字符。从 V$SQLTEXT_WITH_NEWLINES 视图的结果看，SQL 的长度肯定超过了 1000，但从 V$SQL 中的查询结果来看，长度应该远远小于 1000。

查询一下 V$SQL 中 SQL_TEXT 的具体长度：

```
SQL> SELECT LENGTH(SQL_TEXT) FROM BAK_V$SQL;
LENGTH(SQL_TEXT)
----------------
             980
```

长度为 980，这个长度到是对的，可是查询出来的内容却很少，再次查询，将长度和内容一起显示：

```
SQL> SELECT LENGTH(SQL_TEXT), SQL_TEXT FROM BAK_V$SQL;
```

```
LENGTH(SQL_TEXT)
----------------
SQL_TEXT
--------------------------------------------------------------------------------

             980
        info.CONTRACT_ITEM_ID,info.BUYER_ORG_ID a
          el 未承诺'DISCOUNT_STEP is null or INC.NUM_STEP is null then
           to_char(INC.DISCOUNT_STEP * 100,'0.0') || '%'劭勐?' ||
           '未 CASH.CASH is null and CASH.CASH_THIRTY is null then
           '现款:' || to_char(CASH.CASH * 100,'0.0'
           '30 日结款:'SH_THIRTY is not null then
 case whend end as CASH_DISCOUNT,0,'0.0') || '%;'
en info.modify_date > inc.modify_date and info.modify_date > cash.modify_date then info.mo
```

这个格式不是很美观，设置 COL 调整一下输出的格式：

```
SQL> COL SQL_TEXT FORMAT A70 WRAP
SQL> SELECT LENGTH(SQL_TEXT), SQL_TEXT FROM BAK_V$SQL;
LENGTH(SQL_TEXT) SQL_TEXT
---------------- ----------------------------------------------------------------------
 info.T ROWNUM as numrow, yy.* from ( select
        info.CONTRACT_ITEM_ID,info.BUYER_ORG_ID as BUYER_OR
          when INC.DISCOUNT_STEP is null or INC.NUM_
           '起付诺'EP is null then
           to_char(额:' || to_char(INC.NUM_STEP) || '万,折扣率:' ||
        end as NUM_DISCOUNT,T_STEP * 100,'0.0') || '%'
          when CASH.CASH is null and CASH.CASH_THIR
           case 信? is null then
           '现款:' || to_char(CSH.CASH is not null then
        end''e   ASH.CASH * 100,'0.0') || '%;'
           'en CASH.CASH_THIRTY is not null then
                30 日结款:' || to_char(CASH.CASH_THIRTY * 100,'0.0') || '%;'
 case whend end as CASH_DISCOUNT,
                en info.modify_date > inc.modify_date and info.modify_date > cash.modi
                fy_date then info.mo
```

奇怪的事情出现了，不仅 SQL_TEXT 的长度内容被覆盖掉了，而且 SQL_TEXT 的内容并没有从 SQL_TEXT 的栏位开始，而是从一行的开始位置开始的。更关键的是，查询的内容已经发生了变化。

到这里已经可以确定问题的原因了，为了更加精确的定位问题，将 SQL_TEXT 中的内容进行 DUMP：

```
SQL> SELECT DUMP(SUBSTR(SQL_TEXT, 1, 100), 16) FROM BAK_V$SQL;
```

```
DUMP(SUBSTR(SQL_TEXT,1,100),16)
--------------------------------------------------------------------------------
Typ=1                                                                Len=100:
20,53,45,4c,45,43,54,20,2a,20,46,52,4f,4d,20,28,20,d,20,53,45,4c,45,43,54,20,52,4f,57,4e,
55,4d,20,61,73,20,6e,75,6d
,72,6f,77,2c,20,79,79,2e,2a,20,66,72,6f,6d,20,28,20,73,65,6c,65,63,74,20,d,20,69,6e,66,6f
,2e,72,65,63,6f,72,64,5f,69,64,2c,d,20,20
,20,20,20,20,20,20,69,6e,66,6f,2e,43,4f,4e,54,52,41
SQL> SELECT SUBSTR(SQL_TEXT, 1, 100) FROM BAK_V$SQL;
SUBSTR(SQL_TEXT,1,100)
--------------------------------------------------------------------------------
        info.CONTRAumrow, yy.* from ( select
```

从 DUMP 文件中已经可以清晰的看到问题的原因了，SQL 语句中的仅使用了 ASCII(0XD)回车符，而没有使用 ASCII(0XA)换行。这会导致在 UNIX 和 LINUX 环境下，第二行的数据仍然从第一行的第一列位置开始输出，这样就会覆盖第一行本来的内容。

一个简单的例子：

```
SQL> SELECT 'AB' || CHR(13) || 'C' FROM DUAL;
'AB'
----
CB
1 row selected.
SQL> SELECT 'AB' || CHR(10) || CHR(13) || 'C' FROM DUAL;
'AB'|
-----
AB
C


1 row selected.
```

正是这个原因造成了 V$SQL 中显示不正常，而 V$SQLTEXT_WITH_NEWLINES 中由于每行只有 64 个字符，因此还没有被覆盖就切换到下一条记录了。

了解了问题的原因，剩下的就简单了：

```
SQL> SELECT REPLACE(SQL_TEXT, CHR(13), CHR(10) || CHR(13)) FROM BAK_V$SQL;
REPLACE(SQL_TEXT,CHR(13),CHR(10)||CHR(13))
--------------------------------------------------------------------------------
 SELECT * FROM (
 SELECT ROWNUM as numrow, yy.* from ( select
 info.record_id,
        info.CONTRACT_ITEM_ID,info.BUYER_ORG_ID as BUYER_ORGID,
        case
```

```
.
.
.
        end end as CASH_DISCOUNT,
 case when info.modify_date > inc.modify_date and info.modify_date > cash.modify_date then
info.mo
```

手工添加换行信息，就可以解决上面的问题。看似是 bug 的问题，其实并不是 bug 引起的，而是不同操作系统的差异导致了这个问题。在 Windows 环境中，一个回车符就足够了，但是对于 UNIX 和 LINUX 环境，还需要一个换行符。

## 1.4存储过程 ORA-4068 之错误解析

在运行一个过程时报了一个 ORA-4068 错误。虽然问题很简单而且解决过程也不复杂，但是要真正理解错误产生的原因，还需要对概念理解的比较清晰。

下面做一个简单的例子来重现错误：

```
SQL> CREATE TABLE T AS SELECT * FROM TAB;
表已创建。
SQL> CREATE OR REPLACE PROCEDURE P_RECREATE AS
  2  BEGIN
  3   EXECUTE IMMEDIATE 'DROP TABLE T';
  4   EXECUTE IMMEDIATE 'CREATE TABLE T AS SELECT * FROM TAB';
  5  END;
  6  /
过程已创建。
SQL> CREATE OR REPLACE PROCEDURE P_INSERT_T AS
  2  BEGIN
  3   INSERT INTO T SELECT * FROM T;
  4  END;
  5  /
过程已创建。
SQL> BEGIN
  2   P_RECREATE;
  3   P_INSERT_T;
  4  END;
  5  /
BEGIN
*
第 1 行出现错误:
```

```
ORA-04068: 已丢弃程序包　的当前状态
ORA-04065: 未执行, 已更改或删除 stored procedure "YANGTK.P_INSERT_T"
ORA-06508: PL/SQL: 无法找到正在调用 : "YANGTK.P_INSERT_T" 的程序单元
ORA-06512: 在 line 3
```

上面建立了两个过程，在第一个过程 P_RECREATE 中，通过动态语句 DROP TABLE T，然后通过动态语句 CREATE TABLE AS SELECT 来重建 T 表。第二个过程 P_INSERT_T 则更简单，就是根据 T 表的内容实现自复制。

将两个存储过程放在同一个匿名块中调用，就会出现上面的 ORA-4068 错误。

如果单独执行两个过程，则不会报错：

```
SQL> EXEC P_RECREATE
PL/SQL 过程已成功完成。
SQL> EXEC P_INSERT_T
PL/SQL 过程已成功完成。
```

看到这个 ORA-4068 错误，首先想到的是 P_RECREATE 过程对表进行删除并重建的操作，这会导致所有和这个 T 表相关的存储过程变为 INVALID 状态。

由于 T 表结构在重建前面没有发生变化，因此通过简单的编译，就可以使得 P_INSERT_T 过程的状态变为 VALID。

因此，尝试在调用过程之前重编译 P_INSERT_T 过程：

```
SQL> BEGIN
  2   P_RECREATE;
  3   EXECUTE IMMEDIATE 'ALTER PROCEDURE P_INSERT_T COMPILE';
  4   P_INSERT_T;
  5  END;
  6  /
BEGIN
*
第 1 行出现错误:
ORA-04068: 已丢弃程序包　的当前状态
ORA-04065: 未执行, 已更改或删除 stored procedure "YANGTK.P_INSERT_T"
ORA-06508: PL/SQL: 无法找到正在调用 : "YANGTK.P_INSERT_T" 的程序单元
ORA-06512: 在 line 4
```

错误依旧。而且有趣的是，由于错误发生在调用 P_INSERT_T 过程中，所以 ALTER PROCEDURE 命令显然已经执行完成，而且已经成功了。而这个命令的执行成功说明 P_INSERT_T 过程的状态已经恢复为 VALID 状态了，但是错误仍然发生了。

```
SQL> BEGIN
  2   P_RECREATE;
  3   EXECUTE IMMEDIATE 'BEGIN P_INSERT_T; END;';
  4  END;
  5  /
```

```
PL/SQL 过程已成功完成。
```

如果使用动态 SQL 的方式调用 P_INSERT_T 过程，则不会报错。

问题已经清楚了，但是要想说明白，还需要从头说起。

**存储过程在编译时，自动检查语法错误、权限以及所有对象的依赖性。而等到执行的时候，Oracle 则不会再进行类似的检查，而是直接运行过程，这也是存储过程拥有较高效率的原因之一。**

**当存储过程所依赖的对象发生了变化，Oracle 会自动将存储过程的状态置为 INVALID，而存储过程的状态如果为 INVALID，则会在下次执行时尝试重新编译，如果编译通过则继续执行；如果编译失败则报错。**

这就是上面例子中两个存储过程单独执行并不会报错的原因。虽然 P_RECREATE 过程会重建 T 表，并导致 P_INSERT_T 过程失效，但是 P_INSERT_T 在调用时会尝试重新编译，而由于 T 的结构保持不变，所以编译不会报错，因此 P_INSERT_T 的调用就不会报错。

那么为什么两个过程放到一起执行就会报错且尝试重新编译也不起作用呢。这是由于导致 P_INSERT_T 失效的过程就在调用 P_INSERT_T 过程的匿名块中。在将匿名块提交给 Oracle 时，Oracle 对里面每个过程的状态进行了检查，由于导致 P_INSERT_T 失效的 P_RECREATE 过程还没有执行，因此这时所有过程的状态都是正常的。于是 Oracle 记录了过程的状态信息并开始调用过程。当调用 P_RECREATE 过程后，由于 T 表被删除重建，P_INSERT_T 的状态发生变化，但此时 Oracle 对过程 P_INSERT_T 的检查已经完成，因此在尝试直接运行 P_INSERT_T 的代码时发现 P_INSERT_T 的状态已经发生变化，所以这里报错 ORA-4068。

同样的道理，即使对 P_INSERT_T 进行了重新编译，Oracle 在执行时发现检查时的代码已经发生了变化，当前的 P_INSERT_T 和调用时的 P_INSERT_T 已经发生了变化，因此仍然会报错，即使这时存储过程的状态已经是正常的。

而采用动态 SQL 不会报错的原因就更容易理解了。采用动态 SQL 语句，Oracle 将编译时进行的操作推迟到运行时进行。也就是说 Oracle 会在调用 P_RECREATE 之后，在调用 P_INSERT_T 过程之前对 P_INSERT_T 进行检查并重新编译，所以采用动态 SQL 不会报错。

## 1.5 一次网络连接错误的诊断

本节描述的是一次帮别人解决网络连接问题的经过。

**无论是帮别人解决问题，还是处理自己遇到的问题，首先要明确最重要的两点：一是出现错误的环境；二是具体的错误信息。**这两个要点对于后续问题的解决是至关重要的。如果自己无法解决这个问题，那么无论是向别人请教，还是在网络上搜索结果，这两点同样也是关键信息。

如果环境不是很复杂，或是解决问题后发现和环境、版本等关系不密切，可以在描述问题时一笔带过。比如当前这个错误简单的描述就是：在通过 9i 的客户端连接 10g 的 rac 环境时出现了 ORA-12054 错误。

```
E:\>sqlplus test@testrac
SQL*Plus: Release 10.2.0.1.0 - Production on 星期二 5月 8 18:40:18 2007
```

```
Copyright (c) 1982, 2005, Oracle.  All rights reserved.
输入口令:
ERROR:
ORA-12504: TNS: 监听程序在 CONNECT_DATA 中未获得 SID
```

有了错误信息后就可以根据线索进行进一步的检查了;也可以通过进一步的检查来缩小错误可能发生的范围。比如随后进行的 tnsping 检查，就发现当前 TNS 配置正常：

```
E:\>tnsping testrac
TNS Ping Utility for 32-bit Windows: Version 10.2.0.1.0 - Production on 08-5 月 -2007 18:42:00
Copyright (c) 1997, 2005, Oracle.  All rights reserved.
已使用的参数文件:
E:\oracle\10.2\network\admin\sqlnet.ora

已使用 TNSNAMES 适配器来解析别名
Attempting to contact (DESCRIPTION = (ADDRESS = (PROTOCOL = TCP)(HOST = 172.25.1
98.224)(PORT = 1521)) (CONNECT_DATA = (SERVER = DEDICATED) (SERVER_NAME = testrac)))
OK (10 毫秒)
```

TNSPING 正常，说明检查主机名和端口是正常的，这就缩小了错误的范围。

继续进行检查,尝试在 TESTRAC 服务名的配置中添加(SID=TESTRAC1)信息,这时 sqlplus 可以正常登陆了。

问题可能发生的范围被进一步被缩小了。现在多半由于 SERVICE_NAME 的配置出现了错误。不过当前的错误和普通 SERVICE_NAME 出错时的错误又有所不同。

再次仔细检查，最终发现在手工编辑 TNSNAMES.ORA 时出现了拼写错误，将 SERVICE_NAME 拼写为 SERVER_NAME，由此导致了上面错误的产生。

将其改正后，错误消失。

```
TESTRAC =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = 172.25.198.224)(PORT = 1521))
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = testrac)
    )
  )
E:\>sqlplus test@testrac
SQL*Plus: Release 10.2.0.1.0 - Production on 星期二 5 月 8 18:52:52 2007
Copyright (c) 1982, 2005, Oracle.  All rights reserved.
输入口令:
连接到:
Oracle Database 10g Enterprise Edition Release 10.2.0.3.0 - 64bit Production
With the Partitioning, Real Application Clusters, OLAP and Data Mining options
```

```
SQL>
```

由于这个拼写的错误导致 TESTRAC 的配置中，即没有设置 SID 也没有设置 SERVICE_NAME，从而导致了这个 ORA-12504 错误的产生。

同时，这个错误也说明了一个问题。Oracle 并不对 tnsnames.ora 文件中错误的参数名称进行检测，而仅仅是忽略这个错误。

## 1.6浅谈 job 和 database link 的一个特点

昨天遇到一个比较“奇怪”的错误，job 自动运行报错，但是手工执行不会出错，而且手工执行 job 运行的存储过程也不会出错。

仔细研究后发现问题是 job 和 database link 二者特性的共同作用造成的。

通过下面的例子来模仿错误并加以说明。

```
SQL> conn yangtk/yangtk@test
已连接。
SQL> select * from user_db_links;
未选定行
SQL> select * from dba_db_links;
OWNER     DB_LINK                  USERNAME   HOST       CREATED
--------- ------------------------ ---------- ---------- ---------
PUBLIC    REP.YANGTINGKUN                     rep        21-10 月-04
```

在当前用户下没有建立数据库链，但是建有一个到 rep.yangtingkun 的公用数据库链。

在当前用户下执行 select * from tab@rep.yangtingkun 会找到并使用这个公用数据库链。

注意一点，这个数据库链的 USERNAME 为空，也就是说，在建立数据库链的时候没有指定 CONNECT TO 语句，建立这个数据库链的语句如下：

```
CREATE PUBLIC DATABASE LINK REP.YANGTINGKUN USING 'REP';
```

如果没有给出 CONNECT TO 语句，在调用数据库链的时候，会用当前 session 的用户名、密码去访问远端数据库。使用这种数据库链要求本地数据库和远端数据库具有相同的用户名和密码，否则将无法访问。

```
SQL> create or replace procedure p_test as
  2  begin
  3  for c in (select * from tab@rep.yangtingkun) loop
  4  null;
  5  end loop;
  6  end;
  7  /
过程已创建。
SQL> exec p_test;
PL/SQL 过程已成功完成。
```

接着建立存储过程，通过数据库链访问远端数据。

存储过程在本地执行成功。

```
SQL> declare
  2  v_job number;
  3  begin
  4  dbms_job.submit(v_job, 'p_test;', sysdate, 'sysdate + 1/1440');
  5  commit;
  6  end;
  7  /
PL/SQL 过程已成功完成。
SQL> select job, what, next_date, interval, failures, broken from user_jobs;
       JOB WHAT          NEXT_DATE  INTERVAL            FAILURES B
---------- ------------- ---------- ----------------- ---------- -
         6 p_test;       01-12 月-04 sysdate + 1/1440           0 N
```

将存储过程加入 job 中。

```
SQL> select job, failures, broken from user_jobs;
       JOB   FAILURES B
---------- ---------- -
         6          8 N
```

经过一段时间检查 job，发现已经出现了 8 次错误。

```
SQL> exec dbms_job.run(6);
PL/SQL 过程已成功完成。
SQL> select job, failures, broken from user_jobs;
       JOB   FAILURES B
---------- ---------- -
         6          0 N
```

手工执行 job，没有报错。

等待一段时间后，再次查看。发现 job 已经停止。

```
SQL>  select job, failures, broken from user_jobs;
       JOB   FAILURES B
---------- ---------- -
         6         16 Y
```

这时检查 alert_test.log 可以发现下面的错误：

```
Wed Dec 01 19:16:51 2004
Errors in file e:\oracle\admin\test\udump\test_j000_7236.trc:
ORA-12012: 自动执行作业 6 出错
ORA-01005: null password given; logon denied
ORA-06512: 在"YANGTK.P_TEST", line 3
ORA-06512: 在 line 1
```

根据上面这些信息可以判断：job 在自动执行的时候，虽然是以当前用户执行，但是并不包含当前用户的密码信息。而手工执行 job 时，由于运行在当前用户的 session 中，带有当前用户的用户名和密码信息，所以不会报错。

如果比较清晰的了解 ORACLE 数据库链和 JOB 的运行特点就会发现，这个“奇怪”的错误并不奇怪。

## 1.7 一次 ORA-01041 错误诊断

同样这是一次协助别人解决问题，其实这个错误并不是很复杂，不过让我感触比较多。我最开始收到的信息就是在访问数据库时出现了 ORA-01041 错误。

由于只有一个错误信息，诊断问题很困难。在我的进一步要求下，收到了错误的详细信息：

```
2006-7-3 12:13:30:
ORA-01041: 内部错误，hostdef 扩展名不存在
 SELECT * FROM ( SELECT ROWNUM as numrow, zz.* from ( SELECT ID, URL, MEMBER_TYPE ,
LAST_UPDATE_DATE FROM USR_MODULE where LAST_UPDATE_DATE-to_date('2006-06-21 09:19:52',
'yyyy-mm-dd HH24:mi:ss')>0 ) zz WHERE ROWNUM < = 2000 ) WHERE numrow >= 1
```

这个错误倒是不陌生。在一个 sqlplus 客户端连接到 Oracle 的过程中，Oracle 数据库执行了关闭和启动的操作，且在关闭数据库时这个 sqlplus 的连接并没有退出。那么这个 sqlplus 客户端有很大的可能性会碰到这个 ORA-1041 错误。一旦出现了这个错误，无论使用什么用户进行重新连接都没有作用，必须退出 sqlplus，再重新启动 sqlplus 才行。这说明 sqlplus 工具的状态已经不正确了。

仅仅有了这个错误信息还是不够的，判断问题的原因还需要很多相关的信息，比如数据库的版本、平台，以及在何种情况下出现了上面的错误。

因此我提出需要数据库的版本和操作系统的信息，最好将数据库的 alert 文件同时发送过来。

在等待回复的过程中，我在 metalink 上查询了一些有关 ORA-1041 错误的信息。看看是否有其他未知的 bug 会导致这个错误的产生。不过缺少数据库版本信息和平台信息，无法确定 Oracle 描述的 bug 与当前的问题是否相符，因为绝大部分 bug 都是在特定的版本、特定的平台上出现的。

直到我得到完整的错误日志文件才发现做了很多的无用功：根本没有必要去查询 metalink。

首先，这个错误信息不是记录在 Oracle 的 alert 文件中，而是应用程序打印的日志信息。其次，提供给我的错误只是整个错误文件中次数最多的错误，而并不是所有的错误信息。不但漏掉了其中的几个关键错误信息，而且导致问题产生的第一个错误也没有提及。

整理错误信息后，发现没有发给我的错误信息还包括：

ORA-12571: TNS: 包写入程序失败

ORA-03114: 未连接到 ORALCE

ORA-03113: 通信通道的文件结束

只需要通过上面的这三个错误信息，马上就可以定位到问题。由于会话连接到 Oracle 的时间超过了交换器上空闲时间的设置，导致这个连接直接被交换器切断，使得应用程序处于了一个不正常的状态，最终导致了上面的那个 ORA-01041 错误。

如果一开始就把完整的错误日志发送过来，问题可以很快的被解决。很多时候能否准确的定位问题完全取决于信息提供的是否准确和完整。

**其实提出问题的最好方式是通过自己的精简，能用一个简单的例子重现问题。如果这点无法做到或者很难做到，那么最起码应该提供充足的信息。这是问题定位、分析和解决的基础。**如果错误日志中报了 30 多个错误，而你对其中一个进行分析，显然得到的结果也是片面的、不准确的。

在 metalink 上能发现很多没有解决的问题，这些问题中有不少的状态都是“需要进一步的信息”。Oracle 的技术支持在缺少关键信息时都无法确定问题，何况其他人了。

## 1.8 一个 ORA-604 错误的分析

同事遇到一个 ORA-604 错误，分析一下感觉还比较有趣。

出错的 SQL 大致如下：

```
SQL> CREATE TABLE T_604 AS
  2  SELECT * FROM
  3  (SELECT OBJECT_TYPE, TO_CHAR(AVG(OBJECT_ID), '999999.999') FROM DBA_OBJECTS
  4  GROUP BY OBJECT_TYPE
  5  ORDER BY 2 DESC)
  6  WHERE ROWNUM < 10;
(SELECT OBJECT_TYPE, TO_CHAR(AVG(OBJECT_ID), '999999.999') FROM DBA_OBJECTS
                                                            *
第 3 行出现错误:
ORA-00604: 递归 SQL 层 1 出现错误
ORA-01401: 插入的值对于列过大
```

由于同事并不是 DBA，因此对这个错误比较困惑，不清楚为什么 SELECT 语句直接执行没有任何的问题，而根据 SELECT 的查询结果去创建表却出现了错误，因此同事认为可能是空间分配上出现了问题。

```
SQL> SELECT * FROM
  2  (SELECT OBJECT_TYPE, TO_CHAR(AVG(OBJECT_ID), '999999.999') FROM DBA_OBJECTS
  3  GROUP BY OBJECT_TYPE
  4  ORDER BY 2 DESC)
  5  WHERE ROWNUM < 10;
OBJECT_TYPE         TO_CHAR(AVG
------------------- -----------
DATABASE LINK
```

```
MATERIALIZED VIEW    31618.000
RULE SET             31581.400
DIMENSION            31414.000
DIRECTORY            31207.667
EVALUATION CONTEXT   29200.091
XML SCHEMA           28358.700
TRIGGER              27552.375
INDEXTYPE            27381.750
已选择 9 行。
```

一般来说，ORA-604 错误很少直接出现在用户调用的 SQL 中。对于这种情况，后面的那个错误信息才是真正引发错误的原因。也就是说真正的错误是后面的那个 ORA-1401 错误。这个 ORA-1401 错误很好理解：插入的值比表中列的定义要大。

不过 CREATE TABLE AS SELECT 语句没有为创建表的列指定数据类型和长度限制，数据类型和长度都由 SELECT 语句的查询结果来确定。按道理讲不应该出现这种错误。

其实问题很简单，导致错误的真正原因是列名长度太长了，因此只需将上面的 CREATE TABLE 语句改变一下写法就可以顺利执行了：

```
SQL> CREATE TABLE T_604 (OBJECT_TYPE, AVG_OBJECT_ID) AS
  2  SELECT * FROM
  3  (SELECT OBJECT_TYPE, TO_CHAR(AVG(OBJECT_ID), '999999.999') FROM DBA_OBJECTS
  4  GROUP BY OBJECT_TYPE
  5  ORDER BY 2 DESC)
  6  WHERE ROWNUM < 10;
表已创建。
SQL> DROP TABLE T_604;
表已删除。
SQL> CREATE TABLE T_604 AS
  2  SELECT * FROM
  3  (SELECT OBJECT_TYPE, TO_CHAR(AVG(OBJECT_ID), '999999.999') AVG_OBJECT_ID
  4  FROM DBA_OBJECTS
  5  GROUP BY OBJECT_TYPE
  6  ORDER BY 2 DESC)
  7  WHERE ROWNUM < 10;
表已创建。
```

当用户执行 DDL 操作时，Oracle 通过大量的递归调用来维护数据字典。比如这个 CREATE TABLE 语句，Oracle 就会插入或更新 TAB$、COL$等表。这些操作都是递归调用操作，而在递归调用过程中出现的错误就会报错 ORA-604。

出错的 SQL 语句由于没有指定别名，Oracle 会将查询语句中的字段名称作为 CREATE TABLE 语句的列名。也就是说，Oracle 试图将 TO_CHAR(AVG(OBJECT_ID), '999999.999')作为列名，而这个列名的长度显然超过了列长度 30 的限制，因此 Oracle 在插入数据字典表时出现

ORA-1401 错误。

而上面可以成功执行的两个 SQL，一个是在 CREATE TABLE 时就指定了列名；而另一个是在 SELECT 时为超长的列指定了别名。因此这两个 SQL 都不会出现列名超长的问题。

当然这个错误的产生还有一定的条件：如果是 TO_CHAR(AVG(OBJECT_ID),'999999.999') 直接出现在 SELECT 的外层，在 CREATE TABLE 的时候，Oracle 会明确要求用户提供别名。

```
SQL> CREATE TABLE T_604 AS
  2  SELECT OBJECT_TYPE, TO_CHAR(AVG(OBJECT_ID), '999999.999')
  3  FROM DBA_OBJECTS
  4  GROUP BY OBJECT_TYPE
  5  ORDER BY 2 DESC;
SELECT OBJECT_TYPE, TO_CHAR(AVG(OBJECT_ID), '999999.999')
                    *
第 2 行出现错误:
ORA-00998: 必须使用列别名命名此表达式
```

而这个错误就要比前面的 ORA-604 和 ORA-1401 错误直观多了。

## 1.9 ORA-7445(kdodpm)错误

在进行 LOGMINER 操作时碰到了这个错误。下面描述一下错误的起因及解决的过程。

由于需要查看被程序删除的一些记录，因此对重做日志进行了 LOGMINER 的操作：

```
SQL> alter session set nls_date_format = 'yyyy-mm-dd hh24:mi:ss';
会话已更改。
SQL> select * from v$log;
GROUP# THREAD# SEQUENCE#      BYTES MEMBERS ARC STATUS    FIRST_CHANGE# FIRST_TIME
------ ------- --------- ---------- ------- --- --------- ------------- -------------------
     1       1      3285 1073741824       2 YES INACTIVE     1.1124E+12 2009-07-28 00:00:08
     2       1      3286 1073741824       2 YES INACTIVE     1.1124E+12 2009-07-28 00:02:20
     3       1      3287 1073741824       2 NO  CURRENT      1.1124E+12 2009-07-28 01:00:13
     4       1      3284 1073741824       2 YES INACTIVE     1.1124E+12 2009-07-27 01:00:30
     5       2      1457 1073741824       2 NO  CURRENT      1.1124E+12 2009-07-27 23:53:27
     6       2      1454 1073741824       2 YES INACTIVE     1.1124E+12 2009-07-26 00:25:45
     7       2      1455 1073741824       2 YES INACTIVE     1.1124E+12 2009-07-26 00:26:39
     8       2      1456 1073741824       2 YES INACTIVE     1.1124E+12 2009-07-26 23:53:34
已选择 8 行。
```

检查日志信息后发现需要处理的是当前的日志文件：

```
SQL> select member from v$logfile where group# in (3, 5);
MEMBER
------------------------------------------------
```

```
/dev/vx/rdsk/datadg/tradedb_redo1_3_1_1g
/dev/vx/rdsk/datadg/tradedb_redo1_3_2_1g
/dev/vx/rdsk/datadg/tradedb_redo2_1_1_1g
/dev/vx/rdsk/datadg/tradedb_redo2_1_2_1g
SQL>       exec       dbms_logmnr.add_logfile('/dev/vx/rdsk/datadg/tradedb_redo1_3_1_1g',
dbms_logmnr.new)
PL/SQL 过程已成功完成。
SQL>       exec       dbms_logmnr.add_logfile('/dev/vx/rdsk/datadg/tradedb_redo2_1_1_1g',
dbms_logmnr.addfile)
PL/SQL 过程已成功完成。
SQL> exec dbms_logmnr.start_logmnr(options=> dbms_logmnr.dict_from_online_catalog)
PL/SQL 过程已成功完成。
SQL> create table t_logminer tablespace sysaux
  2  as select * from v$logmnr_contents where seg_owner = 'ZHEJIANG' and seg_name =
'LP_PRICE_PROCESS';
create table t_logminer tablespace sysaux
           *
第 1 行出现错误:
ORA-03113: 通信通道的文件结束


SQL> exec dbms_logmnr.end_logmnr
ERROR:
ORA-03114: 未连接到 ORALCE
```

本来一个很简单的操作，没想到遇到了错误。一般前台出现 ORA-3113 错误，后台多半出现 ORA-7445 或 ORA-600 错误，检查 alert 文件：

```
Tue Jul 28 16:04:24 2009
Errors in file /opt/oracle/admin/tradedb/udump/tradedb1_ora_10798.trc:
ORA-07445: 出现异常错误: 核心转储 [kdodpm()+2068] [SIGSEGV] [Invalid permissions for mapped
object] [0xFFFFFFFF79270002] [] []
```

在 alert 文件中果然出现了 ORA-7445 错误，错误参数为 kdodpm。检查这个错误对应的 trace 文件：

```
*** 2009-07-28 15:59:15.046
*** SERVICE NAME:(SYS$USERS) 2009-07-28 15:59:15.027
*** SESSION ID:(282.47827) 2009-07-28 15:59:15.027
* kjdrpkey2hv: called with pkey 207480, options x8
*** 2009-07-28 16:04:24.179
Exception signal: 11 (SIGSEGV), code: 2 (Invalid permissions for mapped object), addr:
0xffffffff79270002, PC: [0x10285aa34, kdodpm(
)+2068]
```

```
*** 2009-07-28 16:04:24.186
ksedmp: internal or fatal error
ORA-07445: 出现异常错误: 核心转储 [kdodpm()+2068] [SIGSEGV] [Invalid permissions for mapped
object] [0xFFFFFFFF79270002] [] []
Current SQL statement for this session:
create table t_logminer tablespace sysaux
as select * from v$logmnr_contents where seg_owner = 'ZHEJIANG' and seg_name =
'LP_PRICE_PROCESS'
----- Call Stack Trace -----
calling              call     entry                argument values in hex
location             type     point                (? means dubious value)
-------------------- -------- -------------------- ----------------------------
ksedmp()+744         CALL     ksedst()             000000300 ? 1066DE17C ?
                                                   000000000 ? 1066DAC70 ?
                                                   1066D99D8 ? 1066DA3D8 ?
```

显然问题是 LOGMINER 操作造成的。Oracle 从 9i 开始就提供了 LOGMINER 功能，到 10g 已经比较稳定了，再加上平时使用 LOGMINER 很少出现错误，那么一定是由于一些特殊的操作或配置导致了错误的产生。如果说当前这个 LOGMINER 操作与其他普通的操作有什么不同的话，那么主要是两点：一个是当前为 RAC 环境，LOGMINER 需要同时对两个实例的日志进行分析；另一个就是当前分析的日志是 ONLINE 日志。

由于 RAC 环境的日志对于任何一个实例都是可以访问的，理论上不大可能出现上面的问题，多半是当前日志造成的问题。切换当前日志，然后再次执行同样的操作：

```
SQL> conn / as sysdba
已连接。
SQL> alter session set nls_date_format = 'yyyy-mm-dd hh24:mi:ss';
会话已更改。
SQL> select group#, status from v$log where status = 'CURRENT';
    GROUP# STATUS
---------- ----------------
         3 CURRENT
         5 CURRENT
已选择 2 行。
SQL> select member from v$logfile where group# in (3, 5);
MEMBER
--------------------------------------------------
/dev/vx/rdsk/datadg/tradedb_redo1_3_1_1g
/dev/vx/rdsk/datadg/tradedb_redo1_3_2_1g
/dev/vx/rdsk/datadg/tradedb_redo2_1_1_1g
/dev/vx/rdsk/datadg/tradedb_redo2_1_2_1g
```

```
已选择 4 行。
SQL> alter system archive log current;
系统已更改。
SQL> select group#, status from v$log where status = 'CURRENT';
    GROUP# STATUS
---------- ----------------
         4 CURRENT
         6 CURRENT
已选择 2 行。
SQL>        exec        dbms_logmnr.add_logfile('/dev/vx/rdsk/datadg/tradedb_redo1_3_1_1g',
dbms_logmnr.new)
PL/SQL 过程已成功完成。
SQL>        exec        dbms_logmnr.add_logfile('/dev/vx/rdsk/datadg/tradedb_redo2_1_1_1g',
dbms_logmnr.addfile)
PL/SQL 过程已成功完成。
SQL> exec dbms_logmnr.start_logmnr(options=> dbms_logmnr.dict_from_online_catalog)
PL/SQL 过程已成功完成。
SQL> create table t_logminer tablespace sysaux
  2  as select * from v$logmnr_contents
  3  where seg_owner = 'ZHEJIANG'
  4  and seg_name = 'LP_PRICE_PROCESS';
表已创建。
```

切换日志后 LOGMINER 分析的日志不再是 ONLINE 日志文件，问题不再出现。

在出现错误时不要着急,冷静地思考当前的操作有何特殊之处往往可以迅速的定位并解决问题。

## 1.10 函数索引产生隐藏列

发现这个问题其实比较偶然，在一次执行 LOGMNR 操作时发现了空的列名。由于这个现象很奇怪且以前从未遇到，于是顺着这个问题研究下去，最终找到了问题的原因。

有一次，由于需要撤销一些已经提交的数据，且提交时间超过了可闪回的时间范围，因此只能采用 LOGMNR 进行处理。而处理过程中，发现得到的 SQL 语句包含了空列的情况。情况类似下面的例子：

```
SQL> CONN SYS@YTK102 AS SYSDBA
输入口令:
已连接。
SQL> ALTER SESSION SET NLS_DATE_FORMAT = 'YYYY-MM-DD HH24:MI:SS';
会话已更改。
```

```
SQL> SELECT GROUP#, FIRST_TIME FROM V$LOG;
    GROUP# FIRST_TIME
---------- -------------------
         1 2008-01-18 22:35:57
         2 2008-01-20 18:58:37
         3 2008-01-16 23:56:50
SQL> SELECT MEMBER FROM V$LOGFILE WHERE GROUP# = 2;
MEMBER
------------------------------------------------------------
E:\ORACLE\ORADATA\YTK102\REDO02.LOG
SQL> ALTER SYSTEM SWITCH LOGFILE;
系统已更改。
SQL> EXEC DBMS_LOGMNR.ADD_LOGFILE('E:\ORACLE\ORADATA\YTK102\REDO02.LOG', DBMS_LOGMNR.NEW)
PL/SQL 过程已成功完成。
SQL> EXEC DBMS_LOGMNR.START_LOGMNR(OPTIONS => DBMS_LOGMNR.DICT_FROM_ONLINE_CATALOG)
PL/SQL 过程已成功完成。
SQL> SELECT SQL_UNDO FROM V$LOGMNR_CONTENTS WHERE SEG_NAME = 'T_LOGMNR';
SQL_UNDO
--------------------------------------------------------------------------------
insert into "YANGTK"."T_LOGMNR"("ID","NAME","") values ('1','A',NULL);
insert into "YANGTK"."T_LOGMNR"("ID","NAME","") values ('2','B',NULL);
insert into "YANGTK"."T_LOGMNR"("ID","NAME","") values ('3','C',NULL);
```

可以看到 T_LOGMNR 表的最后一列是一个空列，而这个列所对应的值也是 NULL。

问题肯定和这个表的结果有关：

```
SQL> DESC YANGTK.T_LOGMNR
 名称                                      是否为空? 类型
 ----------------------------------------- -------- ----------------
 ID                                        NOT NULL NUMBER
 NAME                                               VARCHAR2(30)
SQL> SELECT TABLE_NAME, COLUMN_NAME FROM DBA_TAB_COLUMNS
  2  WHERE TABLE_NAME = 'T_LOGMNR' AND OWNER = 'YANGTK';
TABLE_NAME                     COLUMN_NAME
------------------------------ ------------------------------
T_LOGMNR                       ID
T_LOGMNR                       NAME
```

从数据字典中找不到这个空列的信息。根据经验有些用户所不需要关心的辅助列在 Oracle 的数据字典中是不显示的，想要查询隐藏列只能直接查询 COL$表：

```
SQL> SELECT OBJECT_ID FROM DBA_OBJECTS
  2  WHERE OBJECT_NAME = 'T_LOGMNR' AND OWNER = 'YANGTK';
```

```
 OBJECT_ID
----------
     53636
SQL> SELECT NAME FROM COL$ WHERE OBJ# = 53636;
NAME
------------------------------
ID
NAME
SYS_NC00003$
```

通过查询 DBA_OBJECTS 视图得到 OBJECT_ID 的值。在 COL$表中可以根据这个值找到表 T_LOGMNR 所对应的列信息。

现在从 COL$中可以看到这个隐藏列了。下面进一步查询 COL$视图，检查这个隐藏列到底是什么：

```
SQL> SELECT NAME, DEFAULT$ FROM COL$ WHERE OBJ# = 53636;
NAME                           DEFAULT$
------------------------------ ------------------------------------------
ID
NAME
SYS_NC00003$                   DECODE("ID",1,"NAME"||'1',"NAME"||'2')
```

由于是自动生成的列，Oracle 也会自动来维护，因此这个列拥有 DEFAULT$值。根据这个 DEFAULT 值可以看出这个列是一个基于常规列的函数表达式。这种表达式经常会用在函数索引中，检查这个表是否存在函数索引：

```
SQL> SELECT INDEX_NAME, INDEX_TYPE, UNIQUENESS FROM DBA_INDEXES
  2  WHERE TABLE_NAME = 'T_LOGMNR'
  3  AND TABLE_OWNER = 'YANGTK';
INDEX_NAME                     INDEX_TYPE                  UNIQUENES
------------------------------ --------------------------- ---------
SYS_C005983                    NORMAL                      UNIQUE
IND_T_LOGMNR_NAME              FUNCTION-BASED NORMAL       NONUNIQUE
SQL> SELECT COLUMN_EXPRESSION FROM DBA_IND_EXPRESSIONS
  2  WHERE TABLE_NAME = 'T_LOGMNR'
  3  AND TABLE_OWNER = 'YANGTK'
  4  AND INDEX_NAME = 'IND_T_LOGMNR_NAME';
COLUMN_EXPRESSION
--------------------------------------------------------------------------------
DECODE("ID",1,"NAME"||'1',"NAME"||'2')
```

现在可以确定问题的原因了，这个隐藏列就是函数索引导致的。

## 1.11 用 SQL 解决一道有趣的题

Oracle 的 SQL 语句功能是很强大的，它可以实现一些你意想不到的功能。比如下面这个数学问题，如果使用常规的数学方法进行手工分析将会十分的麻烦，而使用 SQL 来求解则要简单得多。

首先来看一下这个问题：

Gauss 和 Poincare 在天堂相遇了，上帝说：你们都是人间最伟大的数学家，那我来出道题考考你们谁更聪明。我在左手写一个大于 1 小于 100 的数，在右手同样写一个大于 1 小于 100 的数，然后把他们的和写在 Gauss 手上，把积写在 Poincare 手上，看看你们能不能猜出这两个数字是几。

Gauss 看了手上的数字，说：“我不知道这两个数字是几，可我保证 Poincare 也不知道。”

Poincare 看了手上的数字，说：“我原来的确不知道那两个数字是几，可我现在知道了。”

Gauss 说：“那我也知道了。”

问题：那两个数字是几?

这道题给出的已知条件十分的隐蔽，首先来分析一下已知条件。根据题意，最终所求的是两个数字，而这两个数字的范围是在 1 和 100 之间。题目中明确说明的条件仅此而已，剩下的条件就要依靠分析才能得出了。

对于 Gauss 来说他知道两个数之和，而不知道两个数的积，但是 Gauss 却肯定的说，他保证 Poincare 也不知道这两个数是什么。这句话就很有深意。举个例子，如果两个数分别是 3 和 7，那么这两个数之积就是 21。由于这两个数都是大于 1 的，因此两个数乘积为 21 的只有 3 和 7 这一种可能。如果 Poincare 手中的值是 21，那么 Poincare 肯定可以确定这两个数是什么，因此对于 Gauss 而言，手中的值肯定不可能是 10（3 和 7 这两个数的和）。如果这个值是 10，那么这两个数就有可能是 3 和 7，而 3 和 7 的乘积又是唯一的，所以 Gauss 就无法确定 Poincare 也不知道结果。

当然这只是举了一个例子，如果归纳一下就是说，对于 Gauss 而言，所有满足相加与手中数值相等的两个数，它们的积都不是唯一的。只有满足这个条件，他才能确认 Poincare 不知道这两个数是什么。

对于 Poincare 来说，他开始并不知道两个数是什么，但是 Gauss 说出了他的推断之后，Poincare 居然知道了这两个数是什么。这说明由于 Gauss 刚才的推断所排除的一些数值组合后，使得 Poincare 手中的积可以唯一确定这两个数值了。

随后 Gauss 说他也知道了，意味着根据 Poincare 能够确认这两个数的信息，Gauss 也可以唯一的确定这两个数了。

题目已经分析完了，那么如何用 SQL 来求解呢，为了便于描述，这里先给出最终的结果，然后描述一下这个 SQL 的思路：

```
SQL> WITH T_NUM AS
  2  (SELECT ROWNUM + 1 NUM FROM DUAL CONNECT BY LEVEL < 99)
  3  SELECT A, B
  4  FROM
  5  (
  6   SELECT
```

```
  A,
  B,
  TOTAL,
  MUL,
  MUL_P,
  COUNT(DECODE(MUL_P, 1, 1)) OVER(PARTITION BY TOTAL) VALUE
 FROM
 (
  SELECT
   A,
   B,
   TOTAL,
   MUL,
   COUNT(*) OVER (PARTITION BY TOTAL) TOTAL_P,
   COUNT(*) OVER (PARTITION BY MUL) MUL_P
  FROM
  (
   SELECT
    A,
    B,
    TOTAL,
    MUL,
    MIN(MUL_P) OVER (PARTITION BY TOTAL) MUL_M
   FROM
   (
    SELECT
     A.NUM A,
     B.NUM B,
     A.NUM + B.NUM TOTAL,
     A.NUM * B.NUM MUL,
     COUNT(*) OVER (PARTITION BY A.NUM + B.NUM) TOTAL_P,
     COUNT(*) OVER (PARTITION BY A.NUM * B.NUM) MUL_P
    FROM T_NUM A, T_NUM B
    WHERE A.NUM < B.NUM
   )
  )
  WHERE MUL_M != 1
 )
)
```

```
 46  WHERE MUL_P = 1
 47  AND VALUE = 1;
         A          B
---------- ----------
         4         13
```

SQL 有点长，下面简单分析一下。首先看一下 WITH 语句，这个语句其实就是构造大于 1 小于 100 的所有的数。

```
 31     (
 32      SELECT
 33       A.NUM A,
 34       B.NUM B,
 35       A.NUM + B.NUM TOTAL,
 36       A.NUM * B.NUM MUL,
 37       COUNT(*) OVER (PARTITION BY A.NUM + B.NUM) TOTAL_P,
 38       COUNT(*) OVER (PARTITION BY A.NUM * B.NUM) MUL_P
 39      FROM T_NUM A, T_NUM B
 40      WHERE A.NUM < B.NUM
 41     )
```

首先从 SQL 的最内层开始分析，这一层很简单，构造符合大于 1 小于 100 的两个数的笛卡儿积，得到所有的可能性。根据题目的描述，第一个数是 2，第二个数是 3 的情况，与第一个数是 3，第二个数是 2 没有区别，所以这层 SQL 在连接时加上了限制条件 A > B，这样可以去掉重复的结果。在 SELECT 列表中分别列出 A、B 两个数值，以及两个数值之和（A+B）、两个数值之积(A*B)，还通过分析函数计算所有可能性中两个数之和与当前两个数之和相等的组合的个数，以及所有可能性中两个数之积与当前两个数之积相等的组合的个数。

```
 23    (
 24     SELECT
 25      A,
 26      B,
 27      TOTAL,
 28      MUL,
 29      MIN(MUL_P) OVER (PARTITION BY TOTAL) MUL_M
 30     FROM
 31     (
 32      SELECT
 33       A.NUM A,
 34       B.NUM B,
 35       A.NUM + B.NUM TOTAL,
 36       A.NUM * B.NUM MUL,
 37       COUNT(*) OVER (PARTITION BY A.NUM + B.NUM) TOTAL_P,
```

```
38        COUNT(*) OVER (PARTITION BY A.NUM * B.NUM) MUL_P
39       FROM T_NUM A, T_NUM B
40       WHERE A.NUM < B.NUM
41      )
42    )
```

接着看第二层 SQL，除了列出 A 和 B 两个数外，还列出了 A 和 B 之和、A 和 B 之积以及一个很重要的值：根据两个数之和进行分组，找出这两个数之积的组合的最小个数。这样描述确实很抽象，不过没有关系，马上要分析的第三层，会对这个值的含义做进一步的说明。

```
14   (
15    SELECT
16     A,
17     B,
18     TOTAL,
19     MUL,
20     COUNT(*) OVER (PARTITION BY TOTAL) TOTAL_P,
21     COUNT(*) OVER (PARTITION BY MUL) MUL_P
22    FROM
23    (
24     SELECT
25      A,
26      B,
27      TOTAL,
28      MUL,
29      MIN(MUL_P) OVER (PARTITION BY TOTAL) MUL_M
30     FROM
31     (
32      SELECT
33       A.NUM A,
34       B.NUM B,
35       A.NUM + B.NUM TOTAL,
36       A.NUM * B.NUM MUL,
37       COUNT(*) OVER (PARTITION BY A.NUM + B.NUM) TOTAL_P,
38       COUNT(*) OVER (PARTITION BY A.NUM * B.NUM) MUL_P
39      FROM T_NUM A, T_NUM B
40      WHERE A.NUM < B.NUM
41     )
42    )
43    WHERE MUL_M != 1
44   )
```

第三次列出的仍然包括 A 和 B 两个数，以及两个数之和、两个数之积。除此之外，还列出了与当前记录中两个数之和相同的组合数；与当前记录中两个数之积相同的组合数，更关键的是这里进行了过滤，在第二层得到的 MUL_M 不等于 1。

目前得到的结果就是 Gauss 虽然自己不知道两个数是什么，但是可以确认 Poincare 也不知道。前面已经举过 3 和 7 的例子来说明这个问题了，这里就不再重复了。如果归纳起来，就是 Gauss 手中的两数的和，分解为任何一种可能所得到的两个数的积，都不是唯一的。在 SQL 中表示的结果就是 MIN(MUL_P) OVER (PARTITION BY TOTAL) != 1。

随后要解决的问题就是 Poincare 说他原来并不知道两个数分别是什么，而当 Gauss 说完那句话后他已经知道这两个数的值了。也就是说到目前为止，两个数的积已经可以唯一确定这两个数是什么了，数学描述就是两个数乘积分组后值相同的个数是 1。在 SQL 中的表示也就是最外层 SQL 的限制条件 MUL_P = 1。

随后还有最后一个条件，就是 Gauss 这时也知道了两个数是什么，说明 Gauss 根据 Poincare 确定两个数的数值这个事实，进一步推断出了最终的结果。这个 SQL 的实现是通过 COUNT(DECODE(MUL_P, 1, 1)) OVER(PARTITION BY TOTAL) = 1 来实现的。前面提到了，只有 MUL_P 为 1 的情况，Poincare 才能确定唯一确定两个数的值，而 Gauss 根据这个结果也推断出两个数的值，说明在当前两个数之和的分组中，只有一种情况满足 MUL_P 的值为 1。

因此整个 SQL 组合起来，就是这道题的解。

这道题本身解决问题的思路只有穷举法，如果手工计算会非常麻烦且容易出错，而利用穷举法解决问题正是 SQL 所擅长的。将数学语句通过 SQL 来进行表述，就可以方便快速的得到最终的结果了。