第 8 章　回滚与撤销

为了保证数据库中多个用户间的读一致性和能够回退事务，Oracle 必须拥有一种机制，能够为变更的数据构造一种前镜像 (before image)数据(保存修改之前的旧值)， 以保证能够回滚或撤消对数据库所作的修改，同时为数据恢复以及一致性读服务。

这就是回滚（或撤消）。在上一章中我们提到 Redo，我们说 Redo 是用来保证在故障时事务可以被恢复；那么 Undo 则是用来保证事务可以被回退或者撤销。

本章将着重介绍回滚（rollback）与撤销（undo）方面的知识。

8.1　什么是回滚和撤消

首先来介绍一下什么是回滚和撤消。我们知道，从 Oracle6 版本到 Oracle 9i 版本，Oracle 用数据库中的回滚段（Rollback）来提供撤消数据（Undo Data）；而从 Oracle 9i 开始，Oracle 还提供了一种新的撤消数据（Undo Data）管理方式，就是使用 Oracle 自动管理的撤消（UNDO）表空间（Automatic Undo Management，通常可以被缩写为 AUM）。

事务使用回滚段来记录变化前的数据或者撤销信息，回忆一下上一章中讲过的一个例子，假定发出了一个更新语句：

```
UPDATE emp SET sal = 4000 Where empno= 7788;
```

下面看一下这个语句是怎样执行的（为了叙述方便，我们尽量简化了情况）：

1. 检查 empno=7788 记录在 Buffer Cache 中是否存在，如果不存在则读取到 Buffer Cache 中
2. 在回滚表空间的相应回滚段事务表上分配事务槽，这个操作需要记录 Redo 信息
3. 从回滚段读入或者在 Buffer Cache 中创建 sal=3000 的前镜像，这需要产生 Redo 信息并记入 Redo Log Buffer
4. 修改 Sal=4000，这是 update 的数据变更，需要记入 Redo Log Buffer。
5. 当用户提交时，会在 Redo Log Buffer 记录提交信息，并在回滚段标记该事务为非激活（inactive）。

在以上事务处理过程中，注意 REDO 和 UNDO 是交替出现的，这两者对于数据库来说都非常重要。在以上的步骤中，对于回滚段的操作存在多处，在事务开始时，首先需要在回滚表空间获得一个事务槽，分配空间，然后创建前镜像，此后事务的修改才能进行，Oracle 必须以此来保证事务是可以回退的。

如果用户提交（commit）了事务，Oracle 会在日志文件记录提交，并且写出日志，同时会在回滚段中把该事务标记为已提交，提交事务在回滚段事务表的状态变为 INACIVE，然后该事务所使用的回滚空间可以被重用，回滚段空间是循环使用的；如果用户回滚（rollback）事务，则 Oracle 需要从回滚段中把前镜像数据读取出来，修改数据缓冲区，完成回滚，这个过

程本身也要产生 Redo，所以回退这个操作是很昂贵的。

在 Oracle 的性能优化中，有一个性能指标称为平均事务回滚率（Rollback per transaction），用来衡量数据库的提交与回滚效率。在 Statspack 的报告中，可以从开头部分找到这个指标：

```
Load Profile
~~~~~~~~~~~~
% Blocks changed per Read:      0.37      Recursive Call %:      1.14
  Rollback per transaction %:   38.22     Rows per Sort:      11.83
```

该参数计算公式如下:

Round(User rollbacks / (user commits + user rollbacks) ,4)* 100%

其中 user commits 和 user rollbacks 数据来自系统的统计信息，可以从 V$SYSSTAT 视图中得到，在 Statspack 中也包含着部分数据的输出，本例选择的报告相关部分摘录如下：

user commits	31,910	12.9	0.6
user rollbacks	19,740	8.0	0.4

按照公式计算就可以得出平均事务回滚率:

Round(19740 / (31910 + 19740),4) = .3822

这个指标应该接近于 0，如果该指标过高，则说明数据库的回滚过多。回滚过多不仅说明数据库经历了太多的无效操作，而且这些操作会极大影响数据库性能。

8.2 回滚段存储的内容

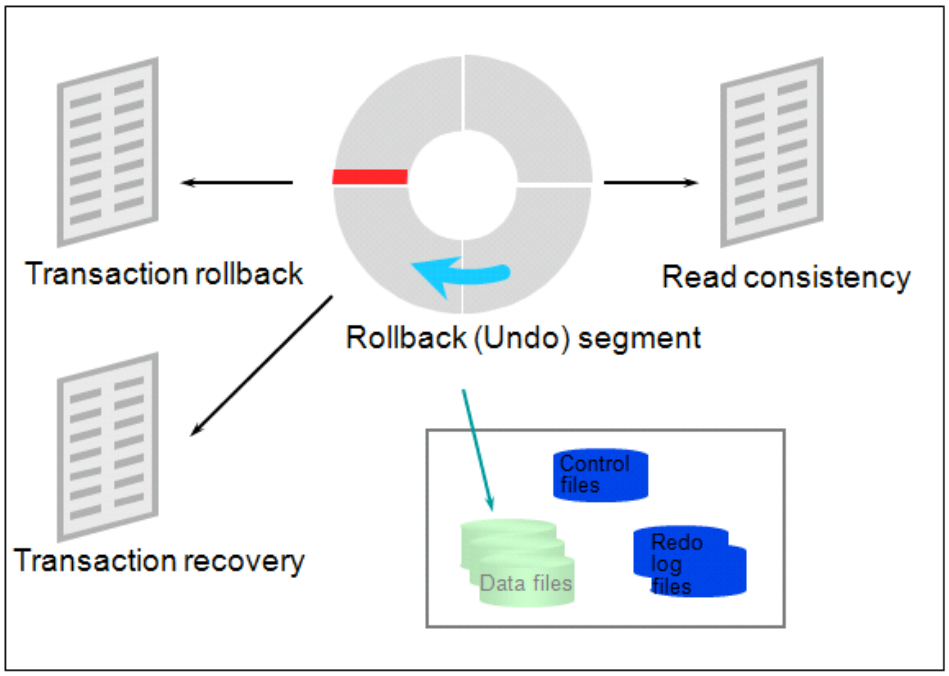
在上一章中讲过，REDO 中只会记录少量信息，这些信息足以重演事务；同样 UNDO 中也只记录精简信息，这些信息足以撤消事务。

对于 insert 操作，回滚段只需要记录插入记录的 rowid，如果回退，只需将该记录根据 rowid 删除即可；对于 update 操作，回滚段只需要记录被更新字段的旧值即可（前镜像）,回退时通过旧值覆盖新值即可完成回退；对于 delete 操作，Oracle 则必须记录整行的数据，在回退时，Oracle 通过一个反向操作恢复删除的数据。

通过以上介绍可以简单总结一下：对于相同数据量的数据操作, 通常 insert 产生最少的 UNDO, update 产生的 UNDO 居中，而 delete 操作产生的 UNDO 最多。这也就是我们经常看到的，当一个大的 Delete 操作失败或者回滚，总是需要很长的时间，并且会有大量的 redo 生成。所以通常在进行大规模数据删除操作时，推荐通过分批删除分次提交，以减少对于回滚段的占用和冲击。

回滚段在 UNDO 表空间中分配，其数据在 Buffer Cache 内存中的管理方式与用户数据一致，同样按照相同的规则写出到 Undo 表空间的数据文件上。UNDO 表空间中的存储空间同样按照 Segment 来分配和使用。下图显示的就是回滚段的机制与原理示意，回滚段的作用除了回退事务外，还要参与事务恢复，以及提供读一致性。

因为 UNDO 数据要参与事务恢复，所以在备份数据库时一定要包含 UNDO 表空间，而且一旦 UNDO 表空间损坏会丢失，那么数据库将会出现故障，需要进行介质恢复来恢复相关数据文件：



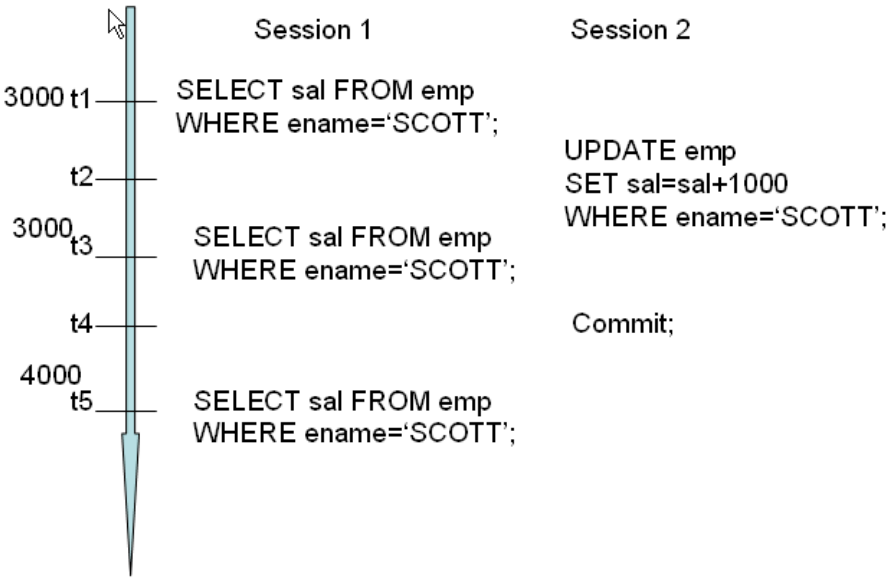
8.3　并发控制和一致性读

允许多用户并发访问是数据库必需满足的功能,那么怎样实现并发访问、控制和数据修改就成为了一个重要问题。

一方面 Oracle 通过锁定机制实现数据库的并发控制；一方面通过多版本（Multi-versioning Model）模型来进行并发数据访问。通过多版本架构，Oracle 实现了读取和写入的分离，使得写入不阻塞读取；读取不阻塞修改。这是 Oracle 数据库区别于其他数据库的一个重要特征。

多版本模型在 Oracle 数据库中是通过一致性读来实现的，一致性读也正是回滚表空间的主要作用之一。

Oracle 一方面不允许其他用户读取未提交数据，一方面要保证用户读取的数据要来自同一时间点。让我们通过下图来看一下什么是 Oracle 的一致性读取（Consistent Reads）：



假定员工 SCOTT 的薪水为 3000：

在 T1 时间我们在 Session1 查询可以得到这个结果；

在 T2 时间 Session2 进行更新，将 SCOTT 的薪水增加 1000，并未提交（此时数据在内存中已经修改，该 Buffer 状态变为 Dirty）；

在 T3 时间 Session1 再次查询，注意此时，Oracle 不会允许其他用户看到未提交数据，所以此时，Oracle 需要通过回滚段记录的前镜像进行一致性读，将 3000 恢复出来提供给用户，这是一致性读的作用；

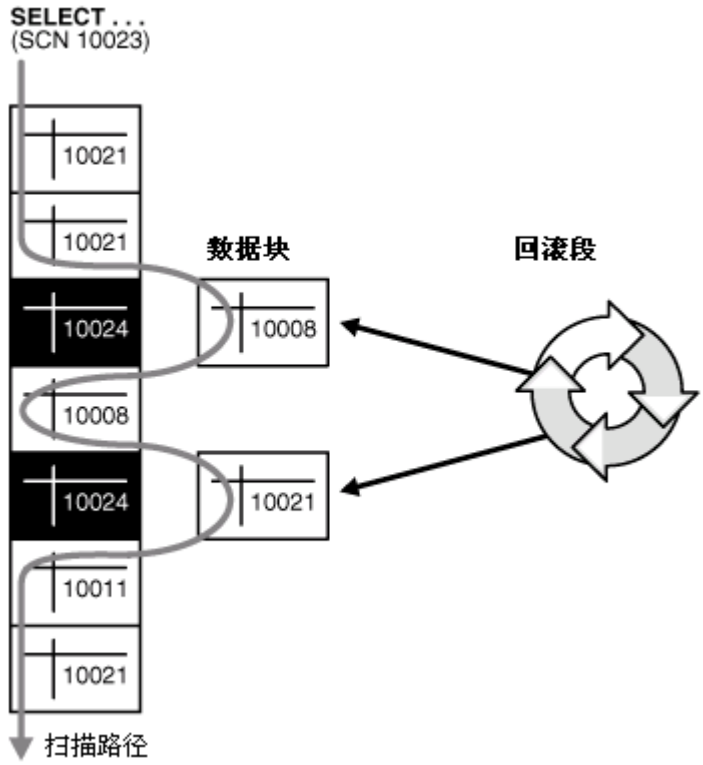
在 T4 时间，Session2 提交该更改，此时数据修改被永久化；

在 T5 时间，其他用户再次查询时，将会看到变化后的数据，也就是“4000”

在前文我们曾经提到，Oracle 内部使用 SCN 作为数据库时钟，这里查询结果集就是根据 SCN 来进行判断的，每个数据块头部都会记录一个提交 SCN，当数据更改提交后，提交 SCN 同时被修改，这个 SCN 在查询时可以用来进行一致性读判断。

在上图所示中，假定查询开始的时间为 T1，则在查询获取的数据块中，如果数据块的提交 SCN 小于 T1，则 Oracle 接受该数据，如果提交 SCN 大于 T1 或者数据被锁定修改尚未记录 COMMIT SCN，则 Oracle 需要通过回滚段构造前镜像来返回结果，这就是一致性读的本质含义。

下图进一步体现了回滚段在一致性读中的重要作用：



8.4　回滚段的前世今生

在 Oracle9i 之前，回滚表空间创建之后，Oracle 随后创建回滚段供数据库使用，也可以手工创建或者删除回滚段进行维护，比如在开始事务之前，可以通过如下命令指定使用特定的回滚段：

set transaction use rollback segment <rollback_segment_name>;

以下是从 Oracle8i 数据库创建日志摘录的部分信息，这就是 Oracle8i 的基本管理方式：

```
Sat Apr 24 16:27:23 2004
CREATE TABLESPACE RBS DATAFILE '/data1/oracle/oradata/8.1.7/rbs01.dbf' SIZE 256M REUSE
  AUTOEXTEND ON NEXT 5120K MINIMUM EXTENT 512K
DEFAULT STORAGE ( INITIAL 512K NEXT 512K MINEXTENTS 8 MAXEXTENTS
UNLIMITED )
Completed: CREATE TABLESPACE RBS DATAFILE '/data1/oracle/oradata/8.1.7/rbs01.dbf'
....
Sat Apr 24 16:27:36 2004
CREATE PUBLIC ROLLBACK SEGMENT RBS0 TABLESPACE RBS STORAGE ( OPTIMAL
4096K )
```

```
Completed: CREATE PUBLIC ROLLBACK SEGMENT RBS0 TABLESPACE RBS
Sat Apr 24 16:27:36 2004
CREATE PUBLIC ROLLBACK SEGMENT RBS1 TABLESPACE RBS STORAGE ( OPTIMAL 4096K )
Completed: CREATE PUBLIC ROLLBACK SEGMENT RBS1 TABLESPACE RBS
.....
Sat Apr 24 16:27:37 2004
CREATE PUBLIC ROLLBACK SEGMENT RBS11 TABLESPACE RBS STORAGE ( OPTIMAL 4096K )
Completed: CREATE PUBLIC ROLLBACK SEGMENT RBS11 TABLESPACE RB
Sat Apr 24 16:27:37 2004
ALTER ROLLBACK SEGMENT "RBS0" ONLINE
Completed: ALTER ROLLBACK SEGMENT "RBS0" ONLINE
Sat Apr 24 16:27:37 2004
ALTER ROLLBACK SEGMENT "RBS1" ONLINE
Completed: ALTER ROLLBACK SEGMENT "RBS1" ONLINE
.......
Sat Apr 24 16:27:37 2004
ALTER ROLLBACK SEGMENT "RBS10" ONLINE
Completed: ALTER ROLLBACK SEGMENT "RBS10" ONLINE
Sat Apr 24 16:27:37 2004
ALTER ROLLBACK SEGMENT "RBS11" ONLINE
Completed: ALTER ROLLBACK SEGMENT "RBS11" ONLINE
```

可以从数据库中查询这些回滚段的状态：

```
SQL> col segment_name for a10
SQL> select segment_name,tablespace_name,status from dba_rollback_segs;
SEGMENT_NA TABLESPACE_NAME                  STATUS
---------- ------------------------------ ----------------
SYSTEM     SYSTEM                           ONLINE
RBS0       RBS                              ONLINE
RBS1       RBS                              ONLINE
……..
RBS10      RBS                              ONLINE
RBS11      RBS                              ONLINE
13 rows selected.
```

从 Oracle9i 开始，Oracle 引入了自动管理的 UNDO 表空间，如果选择使用自动的 UNDO 表空间的管理， 那么用户不再能够创建或删除回滚段，也不再需要为事务指定回滚段，这一切将由 Oracle 自动进行。

```
SQL> select * from v$version where rownum <2;
BANNER
----------------------------------------------------------------
```

```
Oracle9i Enterprise Edition Release 9.2.0.4.0 - Production
SQL> show parameter undo
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
undo_management                      string      AUTO
undo_retention                       integer     10800
undo_suppress_errors                 boolean     FALSE
undo_tablespace                      string      UNDOTBS1
```

伴随自动的 UNDO 管理功能的引入，Oracle 随之引入了几个新的初始化参数：

1．undo_management 用来定义数据库使用的回滚段是否使用自动管理模式。该参数有两个可选项，AUTO 表示自动管理，MANUAL 表示手工管理。

2．undo_tablespace 用来定义在自动管理模式下，当前实例使用哪个 UNDO 表空间

3．undo_suppress_errors 表示当使用自动管理模式时，如果使用不再支持的操作时（比如为事务指定回滚段）是否返回出错信息。设置为 True 时不返回出错信息，操作无效但是可以继续，设置为 False 时，则操作不能继续，这实际上是一个向后兼容的参数。

```
SQL> set transaction use rollback segment rbs1;
set transaction use rollback segment rbs1
*
ERROR 位于第 1 行:
ORA-30019: 自动撤消模式中的回退段操作非法
SQL> show parameter undo_suppress_errors
NAME                                 TYPE        VALUE
------------------------------------ ----------- ----------------
undo_suppress_errors                 boolean     FALSE
SQL> set transaction use rollback segment rbs1;
set transaction use rollback segment rbs1
*
ERROR 位于第 1 行:
ORA-30019: 自动撤消模式中的回退段操作非法
SQL> alter session set undo_suppress_errors=true;
会话已更改。
SQL> set transaction use rollback segment rbs1;
事务处理集。
```

该参数在 Oracle10g 中已经被舍弃：

```
SQL> select * from v$version where rownum <2;
BANNER
----------------------------------------------------------------
Oracle Database 10g Enterprise Edition Release 10.1.0.2.0 - 64bi
SQL> show parameter undo
```

```
NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
undo_management                      string      AUTO
undo_retention                       integer     100000
undo_tablespace                      string      UNDOTBS1
```

4．undo_retention 表示在自动管理模式下，当回滚段变得非激活（INACTIVE）之后，回滚段中的数据在被覆盖前保留的时间，该参数单位是秒。在 Oracle9iR2 中，这个参数的缺省值为 10800 秒，也就是 3 个小时。通过该参数的调节作用，在繁忙的查询系统中，可以有效地避免 ORA-01555 错误，这是 Oracle9i AUM 的一大增强。

在自动管理的 UNDO 表空间下，回滚段的个数是 Oracle 根据数据库的繁忙程度自动分配或者回收的，缺省的，数据库创建时初始化 10 个回滚段：

```
Mon Apr 26 15:56:27 2004
CREATE UNDO TABLESPACE UNDOTBS1 DATAFILE '/opt/oracle9/oradata/testora9/undotbs01.dbf'
SIZE 200M REUSE AUTOEXTEND ON NEXT 5
120K MAXSIZE UNLIMITED
Mon Apr 26 15:56:33 2004
Mon Apr 26 15:56:33 2004
Created Undo Segment _SYSSMU1$
Created Undo Segment _SYSSMU2$
Created Undo Segment _SYSSMU3$
Created Undo Segment _SYSSMU4$
Created Undo Segment _SYSSMU5$
Created Undo Segment _SYSSMU6$
Created Undo Segment _SYSSMU7$
Created Undo Segment _SYSSMU8$
Created Undo Segment _SYSSMU9$
Created Undo Segment _SYSSMU10$
Undo Segment 1 Onlined
Undo Segment 2 Onlined
Undo Segment 3 Onlined
Undo Segment 4 Onlined
Undo Segment 5 Onlined
Undo Segment 6 Onlined
Undo Segment 7 Onlined
Undo Segment 8 Onlined
Undo Segment 9 Onlined
Undo Segment 10 Onlined
Successfully onlined Undo Tablespace 1.
```

从数据库内部 v$rollname 视图也可以查询得到这些自动创建的回滚段信息：

```
SQL> select * from v$rollname;
           USN NAME
---------- ------------------------------
             0 SYSTEM
             1 _SYSSMU1$
             2 _SYSSMU2$
             3 _SYSSMU3$
             4 _SYSSMU4$
             5 _SYSSMU5$
             6 _SYSSMU6$
             7 _SYSSMU7$
             8 _SYSSMU8$
             9 _SYSSMU9$
            10 _SYSSMU10$
11 rows selected.
```

在系统繁忙时，可以从数据库的告警日志文件中看到回滚段的动态创建和释放过程：

```
Tue Mar 21 00:06:51 2006
Created Undo Segment _SYSSMU11$
Tue Mar 21 00:06:51 2006
Undo Segment 11 Onlined
Tue Mar 21 00:06:52 2006
Created Undo Segment _SYSSMU12$
Undo Segment 12 Onlined
Tue Mar 21 00:06:53 2006
Created Undo Segment _SYSSMU13$
Tue Mar 21 00:06:54 2006
Created Undo Segment _SYSSMU14$
Tue Mar 21 00:06:54 2006
Undo Segment 13 Onlined
Tue Mar 21 00:06:54 2006
Undo Segment 14 Onlined
。。。。。。。
Created Undo Segment _SYSSMU34$
Undo Segment 34 Onlined
。。。。。。。。
Tue Mar 21 03:47:39 2006
SMON offlining US=11
SMON offlining US=12
SMON offlining US=13
```

```
SMON offlining US=14
……
SMON offlining US=33
SMON offlining US=34
```

动态创建和释放，这也正是自动管理的 UNDO 表空间的优势之一。

在 Oracle11g 中，UNDO 段的命名规则有了进一步变化，现在 Oracle 将回滚段创建的时间戳包含在 回滚段名称中，这样我们通过名称就能得知一个 UNDO Segment 的创建时间，请看以下查询：

```
SQL> select * from v$version where rownum <2;
BANNER
--------------------------------------------------------------------------------
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
SQL> select * from v$rollname;
       USN NAME
---------- ------------------------------
         0 SYSTEM
         1 _SYSSMU1_1215150730$
         2 _SYSSMU2_1215150730$
         3 _SYSSMU3_1215150730$
         4 _SYSSMU4_1215150730$
         5 _SYSSMU5_1215150730$
         6 _SYSSMU6_1215150730$
         7 _SYSSMU7_1215150730$
         8 _SYSSMU8_1215150730$
         9 _SYSSMU9_1215150730$
        10 _SYSSMU10_1215150730$
11 rows selected.
```

这里的时间戳是 Unix Time，需要经过转换才能变幻为标准时间。请看以下测试。首先创建一个新的 UNDO 表空间 undotbs2，然后设置数据库切换到这个新建的 UNDO 表空间：

```
SQL> create undo tablespace undotbs2 datafile size 50M;
Tablespace created.
SQL>    ALTER SYSTEM SET undo_tablespace='UNDOTBS2';
System altered.
```

原有 UNDO 表空间的回滚段会逐渐离线，新的表空间 UNDO 段顺序创建：

```
SQL> select * from v$rollname;
       USN NAME
---------- ------------------------------
         0 SYSTEM
         3 _SYSSMU3_1215150730$
```

```
        11 _SYSSMU11_1217486802$
        12 _SYSSMU12_1217486803$
        13 _SYSSMU13_1217486803$
        14 _SYSSMU14_1217486803$
        15 _SYSSMU15_1217486803$
        16 _SYSSMU16_1217486803$
        17 _SYSSMU17_1217486803$
        18 _SYSSMU18_1217486803$
        19 _SYSSMU19_1217486803$
        20 _SYSSMU20_1217486803$
12 rows selected.
```

在 Linux 上可以通过 Date 命令将这个 Unix Time 时间戳转换成标准时间（也可以在数据库中自己编写一个函数进行转换）：

```
[oracle@localhost trace]$ date -d '1970-01-01 UTC 1217486802 seconds' +"%Y-%m-%d %T %z"
2008-07-31 14:46:42 +0800
[oracle@localhost trace]$ date
Thu Jul 31 14:47:27 CST 2008
```

8.5　Oracle10g 的 UNDO_RETENTION 管理增强

在 AUM 模式下，我们知道 UNDO_RETENTION 参数用以控制事务提交以后 undo 信息保留的时间。该参数以秒为单位，9iR1 初始值为 900 秒，在 Oracle9iR2 增加为 10800 秒。但是这是一个 NO Guaranteed 的限制。也就是说，如果有其他事务需要回滚空间，而空间出现不足时，这些信息仍然会被覆盖。

从 Oracle10g 开始，如果你设置 UNDO_RETENTION 为 0，那么 Oracle 启用自动调整以满足最长运行查询的需要。当然如果空间不足，那么 Oracle 满足最大允许的长时间查询。而不再需要用户手工调整。当设置 undo_retention 为 0 后，在告警日志文件中可以看到如下信息：

Autotune of undo retention is turned on.

这个新特性的引入伴随着几个新的隐含初始化参数，主要的参数有 2 个：

```
SQL> select ksppinm,ksppdesc
  2  from x$ksppi where ksppinm like '%&var%';
Enter value for var: undo_autotune
old    2: from x$ksppi where ksppinm like '%&var%'
new    2: from x$ksppi where ksppinm like '%undo_autotune%'
KSPPINM                                KSPPDESC
------------------------------ ------------------------------------------------------------
_undo_autotune                 enable auto tuning of undo_retention
SQL> /
```

```
Enter value for var: collect_undo_stats
old   2: from x$ksppi where ksppinm like '%&var%'
new   2: from x$ksppi where ksppinm like '%collect_undo_stats%'
KSPPINM                                    KSPPDESC
------------------------------ ------------------------------------------------------------
_collect_undo_stats                Collect Statistics v$undostat
```

这两个参数缺省都是打开的。

很多时候我们希望前镜像数据能够尽量，而不是被覆盖，Oracle10g 对于 UNDO 增加了 Guarantee 控制，也就是说，你可以指定 UNDO 表空间必须满足 UNDO_RETENTION 的限制。通过如下命令可以修改 UNDO 表空间的新属性：

```
alter tablespace undotbs1 retention guarantee|noguarantee;
```

设置期望的保留时间，修改 UNDO 表空间属性，就可以使 UNDO 表空间运行在 Guarantee 模式：

```
SQL> alter system set undo_retention=900;
System altered.
SQL> alter tablespace undotbs1 retention guarantee;
Tablespace altered.
SQL> select tablespace_name,contents,retention from dba_tablespaces
  2   where tablespace_name like 'UNDO%';
TABLESPACE_NAME                CONTENTS    RETENTION
------------------------------ --------- -----------
UNDOTBS1                       UNDO        GUARANTEE
```

将 UNDO 表空间自动扩展属性取消进行如下测试：

```
SQL> select bytes/1024/1024 from v$datafile where name like '%undo%';
BYTES/1024/1024
---------------
             50
SQL> alter database datafile '/opt/oracle/oradata/eygle/undotbs01.dbf' autoextend off;
Database altered.
```

尝试循环小批量删除数据，在 GURANTEE 设置下，很快出现 ORA-30036 错误：

```
SQL> connect eygle/eygle
Connected.
SQL> select count(*) from t;
  COUNT(*)
----------
   1298100
SQL> begin
  2   for i in 1 .. 1000 loop
  3   delete from t where rownum <1001;
```

```
  4  commit;
  5  end loop;
  6  end;
  7  /
begin
*
ERROR at line 1:
ORA-30036: unable to extend segment by 8 in undo tablespace 'UNDOTBS1'
ORA-06512: at line 3
SQL> select count(*) from t;
  COUNT(*)
----------
   1294100
```

而在修改了 UNDO 表空间 retention 属性后，删除可以顺利完成：

```
SQL> alter tablespace undotbs1 retention noguarantee;
Tablespace altered.
SQL> begin
  2  for i in 1 .. 1000 loop
  3  delete from t where rownum <1001;
  4  commit;
  5  end loop;
  6  end;
  7  /
PL/SQL procedure successfully completed.
```

这就是 GUARANTEE 与 NOGUARANTEE 的不同。

8.6　UNDO_RETENTION 的内部实现

UNDO_RETENTION 机制从 Oracle9i 开始引入，为了实现这一机制，Oracle 在 Undo Segment Header 上创建了一个 Retention Table 用于记录相关 UNDO 存储的提交时间，从而实现其保留策略。

接下来让我们一起来看一下这个 Retention Table 的内容，以下测试来自于 Oracle11g 数据库环境。

```
SQL> select banner from x$version where indx=3;
BANNER
--------------------------------------------------------------------------------
TNS for Linux: Version 11.1.0.6.0 - Production
```

首先使用测试用户执行一个 DML 事务，删除测试表中的部分数据：

```
SQL> connect eygle/eygle
Connected.
SQL> delete from eygle where rownum <10;
9 rows deleted.
```

由于这是一个测试数据库，没有其他事务进行，所以可以通过以下查询找到当前事务使用的回滚段：

```
SQL> select a.usn,a.xacts,b.name from v$rollstat a,v$rollname b
  2    where a.usn=b.usn and a.xacts >0;
         USN          XACTS NAME
---------- ---------- ------------------------------
          15                  1 _SYSSMU15_1217486803$
```

使用如下命令将 UNDO HEADER 转储出来：

```
SQL> alter system dump undo header '_SYSSMU15_1217486803$';
System altered.
```

然后提交这个事务：

```
SQL> commit;
Commit complete.
```

接下来再启动一个新的会话连接，再次执行一次回滚段头的转储输出：

```
[oracle@localhost trace]$ sqlplus "/ as sysdba"
SQL> alter system dump undo header '_SYSSMU15_1217486803$';
System altered.
```

找到两次转储生成的跟踪文件：

```
[oracle@localhost trace]$ ls -sort|tail -4
   8 -rw-r-----    1 oracle          60 Jul 31 16:00 11gtest_ora_30455.trm
  16 -rw-r-----    1 oracle        8924 Jul 31 16:00 11gtest_ora_30455.trc
   8 -rw-r-----    1 oracle          70 Jul 31 16:01 11gtest_ora_30492.trm
  16 -rw-r-----    1 oracle        9033 Jul 31 16:01 11gtest_ora_30492.trc
```

先进行简单的 diff 差异比较，以下摘录出来的信息就是来自保留表中的数据：

```
[oracle@localhost trace]$ diff 11gtest_ora_30455.trc 11gtest_ora_30492.trc
45c46,47
<    Extent Number:1    Commit Time: 1217486940
---
>    Extent Number:1    Commit Time: 1217491248
>    Extent Number:2    Commit Time: 0
```

注意提交后，回滚段区间（Extent）的提交时间发生改变，后者的时间正是刚才的提交时间：

```
[oracle@localhost trace]$ date
Thu Jul 31 16:01:54 CST 2008
[oracle@localhost trace]$ date -d '1970-01-01 UTC 1217491248 seconds' +"%Y-%m-%d %T %z"
```

```
2008-07-31 16:00:48 +0800
```

最后让从转储文件中摘录一个更为完整的保留表信息供参考（提交后的信息输出）：

```
Retention Table
  ------------------------------------------------------------
 Extent Number:0    Commit Time: 1217486940
 Extent Number:1    Commit Time: 1217491248
 Extent Number:2    Commit Time: 0
```

8.7　Oracle10g In Memory Undo 新特性

通过以前的介绍我们知道 UNDO 的管理方式和常规的数据管理方式是相同的，当进行数据修改时，会在 Buffer 中创建前镜像，同时会记录相应的 REDO，然后这些 UNDO 数据同样会写出到 UNDO SEGMENT 上，当进行一致性读或回滚时，可能会产生大量的 Consistent Gets 和 physical reads。注意到这里，UNDO 会产生 REDO 信息，又会写 UNDO SEGMENT，进而又可能产生大量读取 I/O，这些都是资源密集型操作。如果能够缩减 UNDO 在这些环节中的 REDO 与 UNDO 写出，那么显然就可以极大的提升数据库性能，减少资源的消耗和使用。

从 Oracle10g 开始，Oracle 在数据库中引入了 In Memory UNDO（可以被缩写为 IMU）的新技术，使用这一技术，数据库会在共享内存中（Shared Pool）开辟独立的内存区域用于存储 UNDO 信息，这样就可以避免 UNDO 信息以前在 Buffer Cache 中的读写操作，从而可以进一步的减少 Redo 生成，同时可以大大减少以前的 UNDO Segment 的操作。IMU 中数据通过暂存、整理与收缩之后也可以写出到回滚段，这样的写出提供了有序、批量写的性能提升。

IMU 机制与上一章提到的 PVRS 紧密相关，由于每个 IMU Buffer 的大小在 64～128k 左右，所以仅有特定的小事务可以使用，每个事务会被绑定到一个独立空闲的 IMU Buffer，同时相关的 Redo 信息会写入 PVRS 中，同样每个 IMU Buffer 会由一个独立的 In Memory Undo Latch 保护，当 IMU Buffer 或 PRVS 写满之后，数据库需要写出 IMU 中的信息。

一个新引入的隐含参数可以控制该特性是否启用，这个参数是_in_memory_undo，在 Oracle10g 中这个参数的缺省值是 TRUE（不同版本和平台参数的初始设置可能不同）：

```
SQL> @GetHidPar
Enter value for par: in_memo
old     4:      AND x.ksppinm LIKE '%&par%'
new     4:       AND x.ksppinm LIKE '%in_memo%'
NAME                                          VALUE          PDESC
------------------------------ ---------- ----------------------------------------------
_in_memory_undo                           TRUE           Make in memory undo for top level transactions
```

IMU 的内存在 Shared Pool 中分配，回想一下 Redo Log Buffer 的内存使用与功能，实际上 IMU 技术在某种程度上也是参考了 Log Buffer 的机制，通过以下查询可以获得系统当前分配的 IMU 内存：

```
SQL> select * from v$sgastat where name ='KTI-UNDO';
```

```
POOL         NAME                                BYTES
------------ ------------------------------ ----------
shared pool  KTI-UNDO                          1235304
```

In Memory Undo 池缺省的会分配三个，用以提供更好的并发：

```
SQL> @GetHidPar
Enter value for par: imu_pool
old   4:       AND x.ksppinm LIKE '%&par%'
new   4:       AND x.ksppinm LIKE '%imu_pool%'
NAME                           VALUE      PDESC
------------------------------ ---------- --------------------------------
_imu_pools                     3          in memory undo pools
```

IMU 的使用信息，如提交次数可以通过 V$SYSSTAT 视图查询：

```
SQL> select name,value from v$sysstat where name like '%commits';
NAME                                VALUE
------------------------------ ----------------
user commits                        73980
IMU commits                         57548
```

新的内存 Buffer 通过 In Memory Undo Latch 来进行保护：

```
SQL> select name,gets,misses,immediate_gets,sleeps
  2  from v$latch_children where name like '%undo latch';
NAME                             GETS     MISSES IMMEDIATE_GETS     SLEEPS
-------------------------------- ---------- ---------- -------------- ----------
In memory undo latch              54128          0          20253          0
In memory undo latch             303069          3          56878          2
In memory undo latch             135677          1          39360          0
In memory undo latch             118217         16          35919          2
In memory undo latch             110777          0          39421          0
In memory undo latch             192209         31          20977          0
In memory undo latch             234610          1          87358          0
In memory undo latch              61835          0          34315          0
In memory undo latch                  3          0              0          0
In memory undo latch                  3          0              0          0
In memory undo latch                  3          0              0          0
In memory undo latch                  3          0              0          0
In memory undo latch                  3          0              0          0
In memory undo latch                  3          0              0          0
In memory undo latch                  3          0              0          0
In memory undo latch                  3          0              0          0
In memory undo latch                  3          0              0          0
```

In memory undo latch	3	0	0	0

18 rows selected.

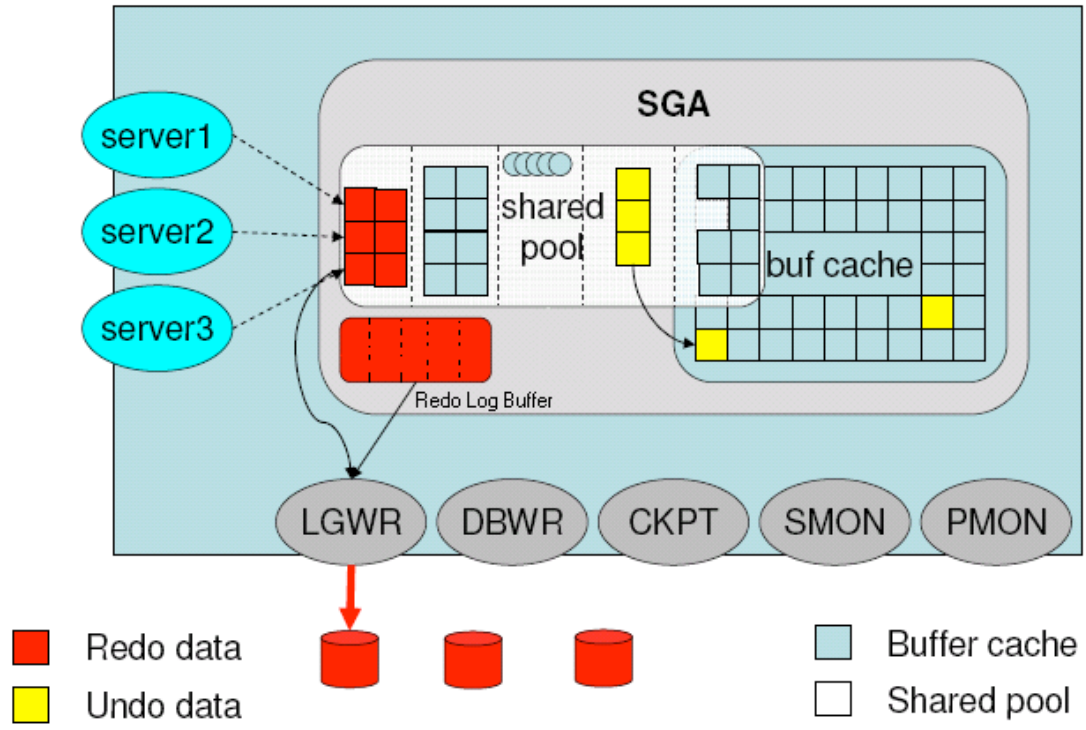
除了前面提到的，还有几个隐含参数与 IMU 有关：

- _recursive_imu_transactions 控制递归事务是否使用 IMU，该参数缺省值为 False
- _db_writer_flush_imu 控制是否允许 DBWR 将 IMU 事务的降级为常规事务，并执行 UNDO SEGMENT 的写出操作，缺省值为 TRUE

此外，在 RAC 环境中，IMU 不被支持。

经过不同版本 Oracle 技术的不断演进，Oracle 的内存管理已经和以前大为不同，现在 Buffer Cache、Shared Pool、Log Buffer 的内容正在不断交换渗透，Redo、UNDO 数据都可以部分的存储在共享池中，Oracle11g 的 Result Cache 也被记录在 Shared Pool 当中。

回忆一下前几章描述过的变化，现在 Oracle 的实例结构图如下显示则更为确切：



8.8　Oracle11g UNDO 表空间备份增强

前面提到，由于 UNDO 表空间在恢复时不可缺少，所以在进行备份时必须备份该表空间，但是我们知道一旦事务提交，修改被确认，则该事务的前镜像被标记为 INACTIVE，其中的信息在恢复时也就不会被用到，如果在备份时能够跳过这些数据，则备份 UNDO 表空间的效率就可以大大提高。

在 Oracle Database 11g 中，Oracle 引入了一个新的特性 RMAN UNDO 备份优化。在 RMAN

备份 UNDO 表空间时，提交事务的 UNDO 信息将不再备份，这个特性随 RMAN 强制启用。在测试中，一个数 G 的 Undo 表空间备份文件的大小仅为数百 K：

```
RMAN> list backup;
List of Backup Sets
===================
BS Key  Type LV Size       Device Type Elapsed Time Completion Time
------- ---- -- ---------- ----------- ------------ ---------------
1       Full    352.00K    DISK        00:00:10     14-JUL-08
        BP Key: 1   Status: AVAILABLE  Compressed: NO  Tag: TAG20080714T164554
        Piece Name: /backupset/2008_07_14/o1_mf_nnndf_TAG20080714T164554_47p4ldcm_.bkp
  List of Datafiles in backup set 1
  File LV Type Ckp SCN    Ckp Time  Name
  ---- -- ---- ---------- --------- ----
  3       Full 1045063    14-JUL-08 /data1/oradata/11gtest/11gtest/undotbs01.dbf
```

这一特性是许多 DBA 期待已久的，在一个繁忙的生产环境中，UNDO 表空间可能占用几十 G 的空间，全部备份显然并不合理，现在 Oracle11g 解决了这个问题。

8.9 回滚机制的深入研究

如果大家有兴趣深入了解一下回滚段的机制，那么请跟随我将前面的例子进一步深化。

1．从 DML 更新事务开始

我们重新来看这个更新语句：

```
SQL> connect scott/tiger
Connected.
SQL> select * from emp;
EMPNO ENAME      JOB       MGR HIREDATE       SAL       COMM     DEPTNO
----------  ----------  ---------  ----------  ---------  ----------  ----------
      7369 SMITH      CLERK      7902 17-DEC-80       800                   20
      7499 ALLEN      SALESMAN   7698 20-FEB-81      1600        300        30
      7521 WARD       SALESMAN   7698 22-FEB-81      1250        500        30
      7566 JONES      MANAGER    7839 02-APR-81      2975                   20
      7654 MARTIN     SALESMAN   7698 28-SEP-81      1250       1400        30
      7698 BLAKE      MANAGER    7839 01-MAY-81      2850                   30
      7782 CLARK      MANAGER    7839 09-JUN-81      2450                   10
      7788 SCOTT      ANALYST    7566 19-APR-87      3000                   20
。。。。。。
14 rows selected.
```

```
SQL> UPDATE emp SET sal = 4000 Where empno= 7788;
1 row updated.
SQL> select * from emp where empno=7788;
     EMPNO  ENAME      JOB              MGR HIREDATE         SAL       COMM
DEPTNO
---------- ---------- --------- ---------- --------- ----------  ---------- ----------
      7788 SCOTT      ANALYST         7566 19-APR-87         4000                    20
```

先不提交这个事务，在另外窗口新开 Session，使用 SYS 用户查询相关信息，进行进一步的分析研究。

2．获得事务信息

从事务表中我们可以获得关于这个事务的信息，该事务位于 6 号回滚段（XIDUSN），在 6 号回滚段上，该事务位于第 23 号事务槽（XIDSLOT）：

```
SQL> SELECT xidusn, xidslot, xidsqn, ubablk, ubafil, ubarec    FROM v$transaction;
    XIDUSN    XIDSLOT     XIDSQN     UBABLK     UBAFIL     UBAREC
---------- ---------- ---------- ---------- ---------- ----------
         6         23      14030         85          2         63
```

从 V$ROLLSTAT 视图中也可以获得事务信息，XACTS 字段代表的是活动事务的数量，同样我们看到该事务位于 6 号回滚段：

```
SQL> select usn,writes,rssize,xacts,hwmsize,shrinks,wraps from v$rollstat;
       USN     WRITES     RSSIZE      XACTS    HWMSIZE    SHRINKS      WRAPS
---------- ---------- ---------- ---------- ---------- ---------- ----------
         0       4680     385024          0     385024          0          0
         1      20674    1171456          0    1171456          0          0
         2      29240    1171456          0    1171456          0          0
         3      31652    1171456          0    1171456          0          0
         4      22968    1171456          0    1171456          0          0
         5      30406    1171456          0    1171456          0          0
         6      31282    1171456          1    1171456          0          0
         7      21510    1171456          0    1171456          0          0
         8      28472    1171456          0    1171456          0          1
         9      69830    1171456          0    1171456          0          0
        10      23328    1171456          0    1171456          0          0
11 rows selected.
```

3．获得回滚段名称并转储段头信息

查询 V$ROLLNAME 视图获得回滚段名称，并转储回滚段头信息：

```
SQL> select * from v$rollname a where a.usn=6;
       USN NAME
```

```
---------- ------------------------------
           6 _SYSSMU6$
SQL> alter system dump undo header '_SYSSMU6$';
System altered
```

生成的跟踪文件如下：

```
SQL> @gettrcname
TRACE_FILE_NAME
------------------------------------------------------------------------
/opt/oracle/admin/conner/udump/conner_ora_15309.trc
```

4．获得跟踪文件信息

注意这就是我们前边多次提到过的回滚段头的信息，其中包括事务表信息，从以下的跟踪文件中，可以清晰的看到这些内容：

```
*******************************************************************************
Undo Segment:  _SYSSMU6$ (6)
*******************************************************************************
  Extent Control Header
  -----------------------------------------------------------------
  Extent Header:: spare1: 0          spare2: 0          #extents: 2          #blocks: 15
                          last map   0x00000000   #maps: 0          offset: 4080
       Highwater::   0x00800055   ext#: 1          blk#: 4          ext size: 8
  #blocks in seg. hdr's freelists: 0
  #blocks below: 0
  mapblk   0x00000000   offset: 1
                          Unlocked
     Map Header:: next   0x00000000   #extents: 2      obj#: 0          flag: 0x40000000
  Extent Map
  -----------------------------------------------------------------
   0x0080004a   length: 7
   0x00800051   length: 8

 Retention Table
  ------------------------------------------------------------
 Extent Number:0   Commit Time: 1144779956
 Extent Number:1   Commit Time: 1143540568

  TRN CTL:: seq: 0x02de chd: 0x0025 ctl: 0x0023 inc: 0x00000000 nfb: 0x0000
            mgc: 0x8201 xts: 0x0068 flg: 0x0001 opt: 2147483646 (0x7ffffffe)
            uba: 0x00800055.02de.3d scn: 0x0819.003f5d3e
Version: 0x01
```

```
FREE BLOCK POOL::
   uba: 0x00000000.02de.3c ext: 0x1    spc: 0x30a
   uba: 0x00000000.02de.22 ext: 0x1    spc: 0x144e
   uba: 0x00000000.02d8.18 ext: 0x6    spc: 0x1206
   uba: 0x00000000.0000.00 ext: 0x0    spc: 0x0
   uba: 0x00000000.0000.00 ext: 0x0    spc: 0x0
TRN TBL::

index    state cflags    wrap#       uel             scn                 dba                 parent-xid
---------------------------------------------------------------------------------------------
<为了排版关系，省略了一些条目，并且截去了最后一个列的信息，完整内容可以从作者的网站得到>
   0x14      9      0x00    0x36ce    0x0016    0x0819.00402706    0x00800055    0x0000.000.00000000
   0x15      9      0x00    0x36ce    0x001b    0x0819.004036d1    0x00800055    0x0000.000.00000000
   0x16      9      0x00    0x36ce    0x0019    0x0819.00402af9    0x00800055    0x0000.000.00000000
   0x17     10      0x80    0x36ce    0x0001    0x0819.004066aa    0x00800055    0x0000.000.00000000
   0x18      9      0x00    0x36ce    0x0015    0x0819.004032df    0x00800055    0x0000.000.00000000
   0x19      9      0x00    0x36ce    0x0018    0x0819.00402eec    0x00800055    0x0000.000.00000000
   0x1a      9      0x00    0x36ce    0x001c    0x0819.00403eb9    0x00800055    0x0000.000.00000000
.............
```

回顾前面的事务信息，该事务正好占用的是第 23 号事务槽(0x17)，状态（State）为 10 代表的是活动事务。

5．转储前镜像信息

我们再来看 DBA（Data Block Address），这个 DBA 指向的就是包含这个事务的前镜像的数据块地址: **0x00800055** 。

我们看一下这个地址如何换算：

DBA 代表数据块的存储地址，由 10 位文件号 + 22 位数据块（Block）组成。

我们将 **0x00800055** 转换为 2 进制就是：

0000 0000 1000 0000 0000 0000 0101 0101

前 10 位代表文件号为 2，后 22 位代表 Block 号为 85。经过转换后，该前镜像信息位于 file 2 block 85。

这和我们从事务表中查询得到的数据完全一致：

```
SQL> SELECT xidusn, xidslot, xidsqn, ubablk, ubafil, ubarec    FROM v$transaction;
     XIDUSN       XIDSLOT        XIDSQN        UBABLK        UBAFIL        UBAREC
---------- ---------- ---------- ---------- ---------- ----------
          6           23        14030           85            2           63
```

提示：很多内容深入研究的内容在数据库内部都有完整的体现，不过通常我们很少注意，只有将两者结合起来学习、研究和理解，我们才能深刻的理解到 Oracle 的本质。希望大家在阅读

这部分内容的时候能够耐心、细致，能够有所收获。

为了同时说明一些其他内容，我们继续先前 Scott 用户的事务，再更新 2 条记录：

```
SQL> update emp set sal=4000 where empno=7788;
1 row updated.
SQL> update emp set sal=4000 where empno=7782;
1 row updated.
SQL> update emp set sal=4000 where empno=7698;
1 row updated.
```

然后将回滚段中的这个 Block 转储出来：

```
SQL> alter system dump datafile 2 block 85;
System altered
```

这是跟踪文件开始部分的信息：

```
Start dump data blocks tsn: 1 file#: 2 minblk 85 maxblk 85
buffer tsn: 1 rdba: 0x00800055 (2/85)
scn: 0x0819.004066c8 seq: 0x01 flg: 0x00 tail: 0x66c80201
frmt: 0x02 chkval: 0x0000 type: 0x02=KTU UNDO BLOCK
*******************************************************************************
UNDO BLK:
xid: 0x0006.018.000036ce    seq: 0x2de cnt: 0x3f    irb: 0x3f    icl: 0x0      flg: 0x0000
```

注意，这部分信息中有一个参数: **irb:0x3f**，irb 指的是回滚段中记录的最近的未提交变更开始之处，如果开始回滚，这是起始的搜索点。

接下来是回滚信息的偏移量，最后一个偏移地址正是 0x3f 的信息：

```
Rec Offset      Rec Offset      Rec Offset      Rec Offset      Rec Offset
----------------------------------------------------------------------------
0x01 0x1f98     0x02 0x1f54     0x03 0x1efc     0x04 0x1ea4     0x05 0x1e54
0x06 0x1e10     0x07 0x1dbc     0x08 0x1d64     0x09 0x1d14     0x0a 0x1cd0
0x0b 0x1c74     0x0c 0x1c1c     0x0d 0x1bcc     0x0e 0x1b88     0x0f 0x1b30
0x10 0x1ad8     0x11 0x1a88     0x12 0x1a44     0x13 0x19f0     0x14 0x1998
0x15 0x1948     0x16 0x1904     0x17 0x18ac     0x18 0x1854     0x19 0x1804
0x1a 0x17c0     0x1b 0x1768     0x1c 0x1710     0x1d 0x16c0     0x1e 0x167c
0x1f 0x1624     0x20 0x15cc     0x21 0x157c     0x22 0x14a4     0x23 0x13fc
0x24 0x1354     0x25 0x12ac     0x26 0x1204     0x27 0x115c     0x28 0x10b4
0x29 0x100c     0x2a 0x0f64     0x2b 0x0ebc     0x2c 0x0e14     0x2d 0x0d6c
0x2e 0x0cc4     0x2f 0x0c1c     0x30 0x0b74     0x31 0x0acc     0x32 0x0a24
0x33 0x097c     0x34 0x08d4     0x35 0x082c     0x36 0x0784     0x37 0x06dc
0x38 0x0634     0x39 0x058c     0x3a 0x04e4     0x3b 0x043c     0x3c 0x0394
0x3d 0x0310     0x3e 0x02b4     0x3f 0x0258
 *----------------------------
```

从接下来的信息中找到 0x3f 信息：

```
*----------------------------
* Rec #0x3f  slt: 0x17  objn: 7961(0x00001f19)  objd: 7961  tblspc: 0(0x00000000)
*          Layer:  11 (Row)    opc: 1    rci 0x3e
Undo type:  Regular undo    Last buffer split:  No
Temp Object:  No
Tablespace Undo:  No
rdba: 0x00000000
*----------------------------
KDO undo record:
KTB Redo
op: 0x02  ver: 0x01
op: C  uba: 0x00800055.02de.3e
KDO Op code: URP row dependencies Disabled
  xtype: XA  bdba: 0x00405c5a  hdba: 0x00405c59
itli: 2  ispac: 0  maxfr: 4863
tabn: 0 slot: 5(0x5) flag: 0x2c lock: 0 ckix: 0
ncol: 8 nnew: 1 size: 1
col  5: [ 3]  c2 1d 33
```

c2 1d 33 转换为 10 进制就是: 2850 （关于数字值的内部存储及转换方式请参考本章末相关部分）。

这是我们最后更新记录的前镜像，Oracle 就是这样通过回滚段保留前镜像信息的：

```
update emp set sal=4000 where empno=7698;
```

我们注意，在这条 UNDO 记录上，还记录一个数据:rci ,该参数代表的就是 undo chain（同一事务中的多次修改，根据 Chain 链接关联）的下一个偏移量，此处为 0x3e.

我们找到 0x3e 这条 UNDO 记录:

```
*----------------------------
* Rec #0x3e  slt: 0x17  objn: 7961(0x00001f19)  objd: 7961  tblspc: 0(0x00000000)
*          Layer:  11 (Row)    opc: 1    rci 0x3d
Undo type:  Regular undo    Last buffer split:  No
Temp Object:  No
Tablespace Undo:  No
rdba: 0x00000000
*----------------------------
KDO undo record:
KTB Redo
op: 0x02  ver: 0x01
op: C  uba: 0x00800055.02de.3d
KDO Op code: URP row dependencies Disabled
```

```
  xtype: XA  bdba: 0x00405c5a  hdba: 0x00405c59
itli: 2  ispac: 0  maxfr: 4863
tabn: 0 slot: 6(0x6) flag: 0x2c lock: 0 ckix: 0
ncol: 8 nnew: 1 size: 1
col  5: [ 3]  c2 19 33
```

这里记录的 c2 19 33 转换为 10 进制就是：2450，是我们第二条更新的数据:

```
update emp set sal=4000 where empno=7782;
```

这里的 rci 指向下一条记录 0x3d。找到 0x3d:

```
*-----------------------------
* Rec #0x3d  slt: 0x17  objn: 7961(0x00001f19)  objd: 7961  tblspc: 0(0x00000000)
*          Layer:  11 (Row)    opc: 1    rci 0x00
Undo type:  Regular undo      Begin trans      Last buffer split:  No
Temp Object:  No
Tablespace Undo:  No
rdba: 0x00000000
*-----------------------------
uba: 0x00800055.02de.3c ctl max scn: 0x0819.003f594b prv tx scn: 0x0819.003f5d3e
KDO undo record:
KTB Redo
op: 0x04  ver: 0x01
op: L  itl: xid:  0x0002.01d.000038ea uba: 0x008000c3.04b1.0c
                                  flg: C---      lkc:  0        scn: 0x0819.0036f14f
KDO Op code: URP row dependencies Disabled
  xtype: XA  bdba: 0x00405c5a  hdba: 0x00405c59
itli: 2  ispac: 0  maxfr: 4863
tabn: 0 slot: 7(0x7) flag: 0x2c lock: 0 ckix: 0
ncol: 8 nnew: 1 size: 0
col  5: [ 2]  c2 1f
```

这里 c2 1f 转换为 10 进制是：3000，正是我们第一条更新的记录：

```
update emp set sal=4000 where empno=7788;
```

这是这个事务中最后一条更新的数据，所以其 undo chain 的指针为 0x00，表示这是最后一条记录。

我们也可以从 x$bh 中找到这些数据块:

```
SQL> select b.segment_name,a.file#,a.dbarfil,a.dbablk,a.class,a.state
  2  from x$bh a,dba_extents b where b.RELATIVE_FNO = a.dbarfil
  3  and b.BLOCK_ID <= a.dbablk and b.block_id + b.blocks > a.dbablk
  4  and b.owner='SCOTT' and b.segment_name='EMP';
SEGMENT_NAME                    FILE#     DBARFIL    DBABLK      CLASS      STATE
--------------------      ----------     --------  ---------- ---------- ------
```

```
EMP                            1          1      23641          4          1
EMP                            1          1      23642          1          1
```

我们注意 class 为 4 的是段头，class 为 1、块号为 23642 的为数据块。

如果此时在其他进程查询 scott.emp 表，Oracle 需要构造一致性读，通过前镜像把变化前的数据展现给用户：

```
SQL> select * from scott.emp;
EMPNO ENAME      JOB              MGR HIREDATE            SAL       COMM DEPTNO
----- ---------- --------- ----- ----------- --------- --------- ------
 7369 SMITH      CLERK          7902 1980-12-17       800.00                20
 7499 ALLEN      SALESMAN       7698 1981-2-20       1600.00    300.00      30
 7521 WARD       SALESMAN       7698 1981-2-22       1250.00    500.00      30
 7566 JONES      MANAGER        7839 1981-4-2        2975.00                20
 7654 MARTIN     SALESMAN       7698 1981-9-28       1250.00   1400.00      30
 7698 BLAKE      MANAGER        7839 1981-5-1        2850.00                30
 7782 CLARK      MANAGER        7839 1981-6-9        2450.00                10
 7788 SCOTT      ANALYST        7566 1987-4-19       3000.00                20
 7839 KING       PRESIDENT           1981-11-17      5000.00                10
 7844 TURNER     SALESMAN       7698 1981-9-8        1500.00       0.00      30
 7876 ADAMS      CLERK          7788 1987-5-23       1100.00                20
 7900 JAMES      CLERK          7698 1981-12-3        950.00                30
 7902 FORD       ANALYST        7566 1981-12-3       3000.00                20
 7934 MILLER     CLERK          7782 1982-1-23       1300.00                10
14 rows selected
```

我们再来查询：

```
SQL> select b.segment_name,a.file#,a.dbarfil,a.dbablk,a.class,a.state,
  2  decode(bitand(flag,1), 0, 'N', 'Y') DIRTY
  3  from x$bh a,dba_extents b  where b.RELATIVE_FNO = a.dbarfil
  4  and b.BLOCK_ID <= a.dbablk and b.block_id + b.blocks > a.dbablk
  5  and b.owner='SCOTT' and b.segment_name='EMP';
SEGMENT_NAME                     FILE#    DBARFIL    DBABLK      CLASS      STATE
DIRTY
-------------------- ---------- ---------- ----------  ---------- ---------- -----
EMP                            1          1      23641          4          1 N
EMP                            1          1      23642          1          3 Y
EMP                            1          1      23642          1          1 N
```

注意到此时，Buffer Cache 中多出一个数据块，也就是 23642 存在 2 份，其中 state 为 3 的就是一致性读构造的前镜像。

6．转储数据块信息

在前镜像信息中，Oracle 还记录了前镜像对应的数据块的地址，我们可以从 bdba 记录中

获得这部分信息，以先前的一个数据为例：

bdba: 0x00405c5a 记录了更改的数据块的地址，0x00405c5a 经过转换为 2 进制就是：

0000 0000 0100 0000 0101 1100 0101 1010

也正是 file 1 block 23642。

我们再将数据表中的 Block 转储出来，看看其中记录了什么样的信息：

```
SQL> alter system dump datafile 1 block 23642;
System altered.
```

检查跟踪文件，获取数据块信息：

```
Start dump data blocks tsn: 0 file#: 1 minblk 23642 maxblk 23642
buffer tsn: 0 rdba: 0x00405c5a (1/23642)
scn: 0x0819.00406ddf seq: 0x01 flg: 0x04 tail: 0x6ddf0601
frmt: 0x02 chkval: 0x6b6a type: 0x06=trans data
Block header dump:    0x00405c5a
 Object id on Block? Y
 seg/obj: 0x1f19    csc: 0x819.406ddf    itc: 2    flg: O    typ: 1 - DATA
       fsl: 0    fnx: 0x0 ver: 0x01

 Itl            Xid                  Uba            Flag   Lck        Scn/Fsc
0x01    0x0009.02e.000036b0    0x00800081.0302.11    C---    0    scn 0x0819.0036f205
0x02    0x0006.018.000036ce    0x00800055.02de.3f    ----    3    fsc 0x0002.00000000
```

这里存在 ITL 事务槽信息，ITL 事务槽指 Interested Transaction List(ITL)，事务必须获得一个 ITL 事务槽才能够进行数据修改。

ITL 内容主要包括:

xid---Transaction ID

Uba---Undo Block Address

Lck---Lock Status

xid=Undo.Segment.Number+Transaction.Table.Slot.Number+Wrap

在以上输出中，我们看到 itl2 （0x02）上存在活动事务。

我们将 xid= **0x0006.018.000036ce** 分解一下：

该事务指向 6 号回滚段，Slot 号为 0x17（转换为 10 进制正好是 23）,Wrap#为 36ce,正是我们 dump 回滚段看到的那个事务。

```
index  state  cflags  wrap#     uel      scn              dba              parent-xid  nub
stmt_num
--------------------------------------------------------------------------------------------
......
0x17    10      0x80    0x36ce    0x0001    0x0819.004066aa    0x00800055    0x0000.000.00000000
0x00000001    0x00000000
```

我们看到，在数据块上同样存在指向回滚段的事务信息。

Uba 代表的是 Undo Block Address，指向具体的回滚段，我们看到该 ITL 上

uba=**0x00800055.02de.3f**

我们将这个 UBA 进行分解：

0x00800055　正是前镜像的地址，

seq: **02de**　　是顺序号

3f　　是 UNDO 记录的开始地址（irb 信息）

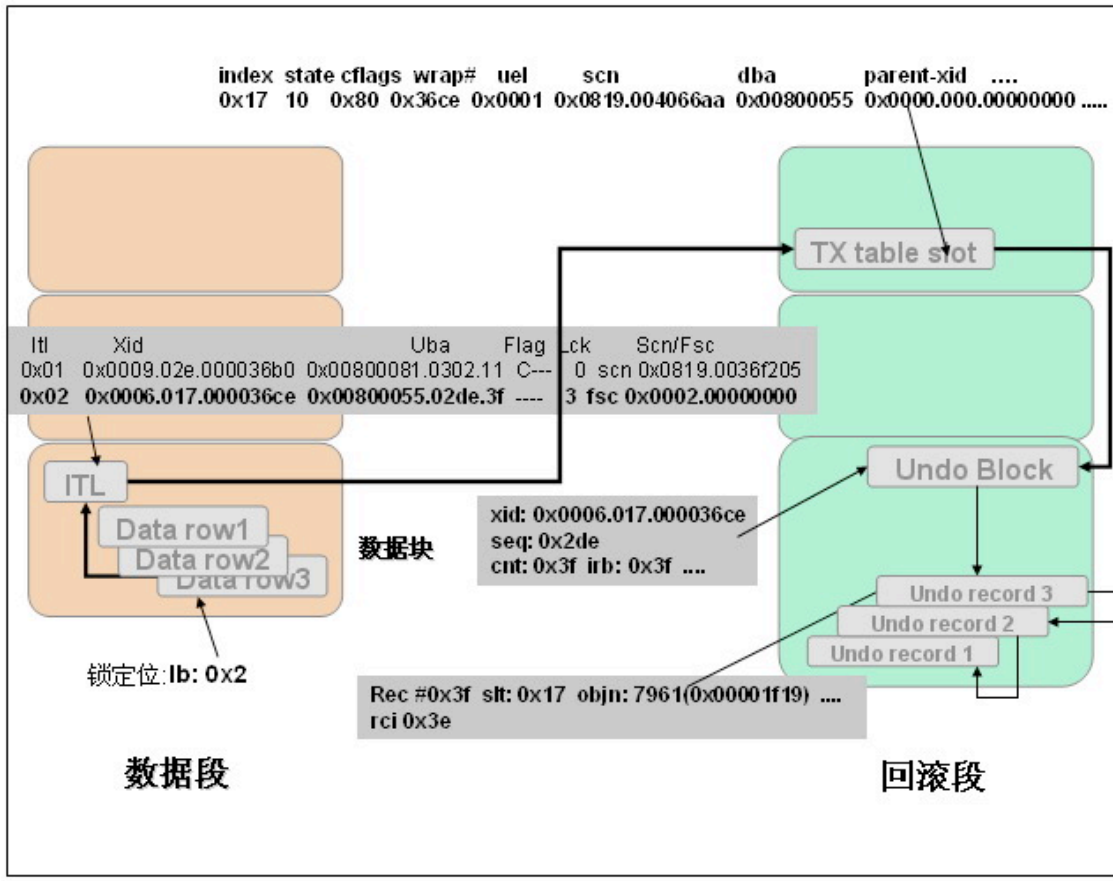
UBA 的内容和 UNDO 中的信息完全相符：

```
UNDO BLK:
xid: 0x0006.018.000036ce   seq: 0x2de cnt: 0x3f   irb: 0x3f   icl: 0x0     flg: 0x0000
```

继续向下我们可以找到这三条被修改的记录，锁定位信息 LB 指向 0x2 号 ITL 事务槽：

```
tab 0, row 5, @0x1d19
tl: 40 fb: --H-FL-- lb: 0x2   cc: 8
col   0: [ 3]   c2 4d 63
col   1: [ 5]   42 4c 41 4b 45
col   2: [ 7]   4d 41 4e 41 47 45 52
col   3: [ 3]   c2 4f 28
col   4: [ 7]   77 b5 05 01 01 01 01
col   5: [ 2]   c2 29
col   6: *NULL*
col   7: [ 2]   c1 1f
tab 0, row 6, @0x1d41
tl: 40 fb: --H-FL-- lb: 0x2   cc: 8
col   0: [ 3]   c2 4e 53
col   1: [ 5]   43 4c 41 52 4b
col   2: [ 7]   4d 41 4e 41 47 45 52
col   3: [ 3]   c2 4f 28
col   4: [ 7]   77 b5 06 09 01 01 01
col   5: [ 2]   c2 29
col   6: *NULL*
col   7: [ 2]   c1 0b
tab 0, row 7, @0x1e54
tl: 40 fb: --H-FL-- lb: 0x2   cc: 8
col   0: [ 3]   c2 4e 59
col   1: [ 5]   53 43 4f 54 54
col   2: [ 7]   41 4e 41 4c 59 53 54
col   3: [ 3]   c2 4c 43
col   4: [ 7]   77 bb 04 13 01 01 01
col   5: [ 2]   c2 29
col   6: *NULL*
col   7: [ 2]   c1 15
```

至此，整个事务过程被完全解析。最后让我们通过一个图表清晰的看一下这个事务的内部流程：



1. 首先当一个事务开始时，需要在回滚段事务表上分配一个事务槽
2. 在数据块头部获取一个 ITL 事务槽，该事务槽指向回滚段头的事务槽
3. 在修改数据之前，需要记录前镜像信息，这个信息以 UNDO RECORD 的形式存储在回滚段中，回滚段头事务槽指向该记录
4. 锁定修改行，修改行锁定位（lb-lock byte）指向 ITL 事务槽
5. 数据修改可以进行

这就是一个事务的基本流程。

7. 块清除（Block Cleanouts）

我们继续看，当我们发出提交（commit）之后，Oracle 怎样来处理。

通过上一章的内容我们知道，Oracle 需要写出 Redo 来保证故障时数据可以被恢复；我们也知道 Oracle 并不需要在提交时就写出变更的数据块。

那么在提交时，Oracle 需要对数据块进行哪些操作呢？

回忆一下上文，我们知道，在事务需要修改数据时，必须分配 ITL 事务槽，必须锁定行，

必须分配回滚段事务槽和回滚空间记录前镜像。当事务提交时，Oracle 需要将回滚段上的事务表信息标记为非活动，以便空间可以重用；那么还有 ITL 事务信息和锁定信息需要清除，以记录提交。

由于 Oracle 在数据块上存储了 ITL 和锁定等事务信息，所以 Oracle 必须在事务提交之后清除这些事务数据。这就是**块清除**。块清除主要要清除的数据有行级锁、ITL 信息（包括提交标志，SCN 等）。

如果提交时修改过的数据块仍然在 Buffer Cache 之中，那么 Oracle 可以清除 ITL 信息，这叫做快速块清除（Fast Block Cleanout），快速块清除还有一个限制，当修改的块数量超过 Buffer Cache 的约 10%，则对超出部分不再进行快速块清除。

如果提交事务的时候，修改过的数据块已经被写回到数据文件上（或大量修改超出 10% 的部分），再次读出该数据块进行修改，显然成本过于高昂，对于这种情况，Oracle 选择延迟块清除（Delayed Block Cleanout），等到下一次访问该 Block 时再来清除 ITL 锁定信息，这就是延迟块清除。Oracle 通过延迟块清除来提高数据库的性能，加快提交操作。

快速提交是最普遍的情况，我们来看一下延迟块清除的处理。继续前面的测试：

```
SQL> update emp set sal=4000 where empno=7788;
1 row updated.
SQL> update emp set sal=4000 where empno=7782;
1 row updated.
SQL> update emp set sal=4000 where empno=7698;
1 row updated.
```

更新完成之后，强制刷新 Buffer Cache，将 Buffer Cache 中的数据都写出到数据文件：

```
SQL> alter session set events = 'immediate trace name flush_cache';
Session altered.
```

此时再提交事务：

```
SQL> commit;
Commit complete.
```

由于此时更新过的数据已经写出到数据文件，Oracle 将执行延迟块清除，将此时的数据块和回滚段转储出来:

```
SQL> alter system dump datafile 1 block 23642;
System altered.
SQL> alter system dump undo header '_SYSSMU6$';
System altered.
SQL> alter system dump datafile 2 block 85;
System altered.
```

研究一下，我们看数据块上的信息,ITL 事务信息仍然存在:

```
Start dump data blocks tsn: 0 file#: 1 minblk 23642 maxblk 23642
buffer tsn: 0 rdba: 0x00405c5a (1/23642)
scn: 0x0819.00406ddf seq: 0x01 flg: 0x04 tail: 0x6ddf0601
frmt: 0x02 chkval: 0x6b6a type: 0x06=trans data
```

```
Block header dump:    0x00405c5a
 Object id on Block? Y
 seg/obj: 0x1f19    csc: 0x819.406ddf    itc: 2    flg: O    typ: 1 - DATA
        fsl: 0    fnx: 0x0 ver: 0x01

 Itl              Xid                          Uba              Flag    Lck            Scn/Fsc
0x01      0x0009.02e.000036b0    0x00800081.0302.11    C---       0    scn 0x0819.0036f205
0x02      0x0006.018.000036ce    0x00800055.02de.3f    ----       3    fsc 0x0002.00000000
```

数据块的锁定信息仍然存在:

```
tab 0, row 5, @0x1d19
tl: 40 fb: --H-FL-- lb: 0x2    cc: 8
col    0: [ 3]    c2 4d 63
col    1: [ 5]    42 4c 41 4b 45
col    2: [ 7]    4d 41 4e 41 47 45 52
col    3: [ 3]    c2 4f 28
col    4: [ 7]    77 b5 05 01 01 01 01
col    5: [ 2]    c2 29
col    6: *NULL*
col    7: [ 2]    c1 1f
tab 0, row 6, @0x1d41
tl: 40 fb: --H-FL-- lb: 0x2    cc: 8
col    0: [ 3]    c2 4e 53
col    1: [ 5]    43 4c 41 52 4b
col    2: [ 7]    4d 41 4e 41 47 45 52
col    3: [ 3]    c2 4f 28
col    4: [ 7]    77 b5 06 09 01 01 01
col    5: [ 2]    c2 29
col    6: *NULL*
col    7: [ 2]    c1 0b
tab 0, row 7, @0x1e54
tl: 40 fb: --H-FL-- lb: 0x2    cc: 8
col    0: [ 3]    c2 4e 59
col    1: [ 5]    53 43 4f 54 54
col    2: [ 7]    41 4e 41 4c 59 53 54
col    3: [ 3]    c2 4c 43
col    4: [ 7]    77 bb 04 13 01 01 01
col    5: [ 2]    c2 29
col    6: *NULL*
col    7: [ 2]    c1 15
```

再来看回滚段的信息:

```
   0x17      9      0x00    0x36ce    0xffff    0x0819.0040791b    0x00800055    0x0000.000.00000000
0x00000001    0x00000000
```

事务提交，事务表已经释放。

如果此时我们查询 SCOTT.EMP 表，数据库将产生延迟块清除：

```
SQL> set autotrace on
SQL> select * from scott.emp;
EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM       DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7369 SMITH      CLERK             7902 17-DEC-80        800                    20
      7499 ALLEN      SALESMAN          7698 20-FEB-81       1600        300         30
      7521 WARD       SALESMAN          7698 22-FEB-81       1250        500         30
      7566 JONES      MANAGER           7839 02-APR-81       2975                    20
      7654 MARTIN     SALESMAN          7698 28-SEP-81       1250       1400         30
      7698 BLAKE      MANAGER           7839 01-MAY-81       4000                    30
      7782 CLARK      MANAGER           7839 09-JUN-81       4000                    10
      7788 SCOTT      ANALYST           7566 19-APR-87       4000                    20
      7839 KING       PRESIDENT              17-NOV-81       5000                    10
  。。。。。。。
14 rows selected.
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   TABLE ACCESS (FULL) OF 'EMP'
Statistics
----------------------------------------------------------
          0  recursive calls
          0  db block gets
          5  consistent gets
          2  physical reads
         60  redo size
```

注意到查询在此时产生了物理读和 REDO，这个 REDO 就是因为延迟块清除导致的。再次查询，则不会继续生成 REDO 了：

```
SQL> /
     EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM
DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7369 SMITH      CLERK             7902 17-DEC-80        800                    20
      7499 ALLEN      SALESMAN          7698 20-FEB-81       1600        300         30
```

```
 。。。。。。
14 rows selected.

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   TABLE ACCESS (FULL) OF 'EMP'

Statistics
----------------------------------------------------------
            0  recursive calls
            0  db block gets
            4  consistent gets
            0  physical reads
            0  redo size
```

再次转储一下该 Block 来看看此时数据块上的信息：

```
SQL> alter system dump datafile 1 block 23642;

System altered.
```

我们看到此时 ITL 事务信息已经清除，但是注意，这里的 Xid 和 Uba 信息仍然存在：

```
Start dump data blocks tsn: 0 file#: 1 minblk 23642 maxblk 23642
buffer tsn: 0 rdba: 0x00405c5a (1/23642)
scn: 0x0819.00407961 seq: 0x01 flg: 0x00 tail: 0x79610601
frmt: 0x02 chkval: 0x0000 type: 0x06=trans data
Block header dump:    0x00405c5a
 Object id on Block? Y
 seg/obj: 0x1f19    csc: 0x819.407961    itc: 2    flg: O    typ: 1 - DATA
       fsl: 0    fnx: 0x0 ver: 0x01

 Itl              Xid                  Uba              Flag  Lck        Scn/Fsc
0x01    0x0009.02e.000036b0    0x00800081.0302.11    C---     0    scn 0x0819.0036f205
0x02    0x0006.018.000036ce    0x00800055.02de.3f    C---     0    scn 0x0819.0040791b
```

数据行的锁定位也已经清除：

```
tab 0, row 5, @0x1d19
tl: 40 fb: --H-FL-- lb: 0x0    cc: 8
col    0: [ 3]    c2 4d 63
col    1: [ 5]    42 4c 41 4b 45
col    2: [ 7]    4d 41 4e 41 47 45 52
```

```
col    3: [ 3]    c2 4f 28
col    4: [ 7]    77 b5 05 01 01 01 01
col    5: [ 2]    c2 29
col    6: *NULL*
col    7: [ 2]    c1 1f
tab 0, row 6, @0x1d41
tl: 40 fb: --H-FL-- lb: 0x0    cc: 8
col    0: [ 3]    c2 4e 53
col    1: [ 5]    43 4c 41 52 4b
col    2: [ 7]    4d 41 4e 41 47 45 52
col    3: [ 3]    c2 4f 28
col    4: [ 7]    77 b5 06 09 01 01 01
col    5: [ 2]    c2 29
col    6: *NULL*
col    7: [ 2]    c1 0b
tab 0, row 7, @0x1e54
tl: 40 fb: --H-FL-- lb: 0x0    cc: 8
col    0: [ 3]    c2 4e 59
col    1: [ 5]    53 43 4f 54 54
col    2: [ 7]    41 4e 41 4c 59 53 54
col    3: [ 3]    c2 4c 43
col    4: [ 7]    77 bb 04 13 01 01 01
col    5: [ 2]    c2 29
col    6: *NULL*
col    7: [ 2]    c1 15
```

8．提交之后的 UNDO 信息

当提交事务之后，回滚段事务表标记标记该事务为非活动，继续再来看一下回滚段数据块的信息：

我们看到这里 irb 指向了 0x40，此前的事务已经不可回滚。

```
Start dump data blocks tsn: 1 file#: 2 minblk 85 maxblk 85
buffer tsn: 1 rdba: 0x00800055 (2/85)
scn: 0x0819.00407baa seq: 0x01 flg: 0x04 tail: 0x7baa0201
frmt: 0x02 chkval: 0x053d type: 0x02=KTU UNDO BLOCK

*******************************************************************************
UNDO BLK:
xid: 0x0006.025.000036ce    seq: 0x2de cnt: 0x40    irb: 0x40    icl: 0x0      flg: 0x0000
```

偏移量列表也已经新增了一条信息 0x40 **0x01b0** ：

```
 Rec Offset      Rec Offset      Rec Offset      Rec Offset      Rec Offset
----------------------------------------------------------------------------

°°°°°°°°°°°°
0x33 0x097c     0x34 0x08d4     0x35 0x082c     0x36 0x0784     0x37 0x06dc
0x38 0x0634     0x39 0x058c     0x3a 0x04e4     0x3b 0x043c     0x3c 0x0394
0x3d 0x0310     0x3e 0x02b4     0x3f 0x0258     0x40 0x01b0
```

至于前镜像 0x3d 0x3e 0x3f 的信息，仍然存在：

```
*-----------------------------
* Rec #0x3d    slt: 0x17    objn: 7961(0x00001f19)    objd: 7961    tblspc: 0(0x00000000)
*            Layer:    11 (Row)      opc: 1      rci 0x00
Undo type:    Regular undo       Begin trans       Last buffer split:    No
Temp Object:    No
Tablespace Undo:    No
rdba: 0x00000000
*-----------------------------
uba: 0x00800055.02de.3c ctl max scn: 0x0819.003f594b prv tx scn: 0x0819.003f5d3e
KDO undo record:
KTB Redo
op: 0x04    ver: 0x01
op: L    itl: xid:    0x0002.01d.000038ea uba: 0x008000c3.04b1.0c
                                  flg: C---       lkc:    0         scn: 0x0819.0036f14f
KDO Op code: URP row dependencies Disabled
  xtype: XA    bdba: 0x00405c5a    hdba: 0x00405c59
itli: 2    ispac: 0    maxfr: 4863
tabn: 0 slot: 7(0x7) flag: 0x2c lock: 0 ckix: 0
ncol: 8 nnew: 1 size: 0
col    5: [ 2]    c2 1f


*-----------------------------
* Rec #0x3e    slt: 0x17    objn: 7961(0x00001f19)    objd: 7961    tblspc: 0(0x00000000)
*            Layer:    11 (Row)      opc: 1      rci 0x3d
Undo type:    Regular undo      Last buffer split:    No
Temp Object:    No
Tablespace Undo:    No
rdba: 0x00000000
*-----------------------------
KDO undo record:
KTB Redo
op: 0x02    ver: 0x01
```

```
op: C    uba: 0x00800055.02de.3d
KDO Op code: URP row dependencies Disabled
  xtype: XA    bdba: 0x00405c5a    hdba: 0x00405c59
itli: 2    ispac: 0    maxfr: 4863
tabn: 0 slot: 6(0x6) flag: 0x2c lock: 0 ckix: 0
ncol: 8 nnew: 1 size: 1
col    5: [ 3]    c2 19 33


*-----------------------------
```

我们大家可以猜想，虽然这个事务已经提交，不可以回滚了，但是在覆盖之前，这个前镜像信息仍然存在，通过某种手段，我们应该仍然可以获得这个信息。

这个猜想显然是成立的。

8.10 Oracle 9i 闪回查询的新特性

从 Oracle9i 开始，Oracle 开始提供闪回查询特性(flashback query)，允许将回滚段中的数据进行闪回。通过这个例子我们来看一下这个从 Oracle9i 开始提供的新特性。

首先我们注意到这里存在一个信息: ctl max scn: 0x0819.003f594b ，这个转换为 SCN 值就是：

```
SQL> select (to_number('819','xxxx')*power(2,32) + to_number('3f594b','xxxxxxxx')) scn
  2    from dual;
                    SCN
------------------
      8903471356235
```

可以查询一下当前数据库的 SCN：

```
SQL> select dbms_flashback.get_system_change_number scn from dual;
                    SCN
------------------
      8903471437610
```

通过特定的语法，我们可以将 SCN 为 8903471356235 的历史状态数据查询出来：

```
SQL> select * from emp as of scn 8903471356235 where empno in (7788,7782,7698);
  EMPNO  ENAME          JOB                       MGR  HIREDATE              SAL             COMM
DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
         7698 BLAKE          MANAGER              7839 01-MAY-81          2850                      30
         7782 CLARK          MANAGER              7839 09-JUN-81          2450                      10
         7788 SCOTT          ANALYST              7566 19-APR-87          3000                      20
```

在结果中，我们注意到 3 名员工的薪水恢复到了之前值。而在当前的查询中，这个数值

是变化后的 4000：

```
SQL> select * from emp where empno in (7788,7782,7698);
     EMPNO  ENAME          JOB                  MGR  HIREDATE              SAL          COMM
DEPTNO
---------- ---------- --------- ---------- --------- ---------- ---------- ----------
      7698 BLAKE         MANAGER              7839 01-MAY-81          4000                    30
      7782 CLARK         MANAGER              7839 09-JUN-81          4000                    10
      7788 SCOTT         ANALYST              7566 19-APR-87          4000                    20
```

由于这个查询需要从 UNDO 中获取前镜像信息，如果 UNDO 中的信息被覆盖，则以上查询将会失败。为了模拟不同情况，创建一个新的 UNDO 表空间，切换数据库使用新的 UNDO 表空间，再将原表空间 Offline：

```
SQL> create undo tablespace undotbs datafile '/opt/oracle/oradata/conner/undotbs.dbf' size 2M;
Tablespace created.
SQL> alter system set undo_tablespace=undotbs;
System altered.
SQL> alter tablespace UNDOTBS1 offline;
Tablespace altered.
SQL> alter session set events = 'immediate trace name flush_cache';
Session altered.
```

再来查询，此时出现错误，记录该文件已经不可读取：

```
SQL> select * from emp as of scn 8903471356235 where empno in (7788,7782,7698);
select * from emp as of scn 8903471356235 where empno in (7788,7782,7698)
                    *
ERROR at line 1:
ORA-00376: file 2 cannot be read at this time
ORA-01110: data file 2: '/opt/oracle/oradata/conner/undotbs1.dbf'
```

将 UNDOTBS1 重新启用，则此时前镜像信息再次可以查询：

```
SQL> alter tablespace UNDOTBS1 online;
Tablespace altered.
SQL> alter system set undo_tablespace=UNDOTBS1;
System altered.
```

在其他 Session 执行大量事务，使得前镜像信息被覆盖：

```
SQL> begin
  2  for i in 1 .. 2000 loop
  3  update emp set sal=4000;
  4  rollback;
  5  end loop;
  6  end;
  7  /
```

```
PL/SQL procedure successfully completed.
SQL> /
PL/SQL procedure successfully completed.
SQL> /
```

观察回滚段的使用：

```
SQL> select usn,xacts,RSSIZE,HWMSIZE from v$rollstat where usn=6;
       USN      XACTS      RSSIZE     HWMSIZE
---------- ---------- ---------- ----------
         6          1    7331840    7331840
```

那么再次查询就可能收到如下错误：

```
SQL> select * from emp as of scn 8903471356235 where empno in (7788,7782,7698);
select * from emp as of scn 8903471356235 where empno in (7788,7782,7698)
                    *
ERROR at line 1:
ORA-01555: snapshot too old: rollback segment number 6 with name "_SYSSMU6$" too small
```

ORA-01555 错误出现，说明要查询的前镜像信息已经失去。

8.11　使用 ERRORSTACK 进行错误跟踪

ERRORSTACK 是 Oracle 提供的接口，用于诊断 Oracle 的错误信息。

诊断事件可以在 Session 级设置，也可以在系统级设置，通常如果要诊断全局错误，最好在系统级设置。设置了 ERRORSTACK 事件之后，Oracle 会将出错时的信息记入跟踪文件之中，然后再出现错误时就可以通过跟踪文件进行错误诊断和排查了。

继续上边的测试，尝试通过 ERRORSTACK 事件来跟踪 ORA-01555 错误：

```
SQL> ALTER SYSTEM SET EVENTS '1555 TRACE NAME ERRORSTACK LEVEL 4';
System altered.
SQL> select * from emp as of scn 8903471356235 where empno in (7788,7782,7698);
select * from emp as of scn 8903471356235 where empno in (7788,7782,7698)
                    *
ERROR at line 1:
ORA-01555: snapshot too old: rollback segment number 6 with name "_SYSSMU6$" too small
```

检查告警日志文件，可以得到如下信息：

```
Wed Apr 19 16:46:51 2006
OS Pid: 1274 executed alter system set events '1555 TRACE NAME ERRORSTACK LEVEL 4'
Wed Apr 19 16:47:06 2006
ORA-01555 caused by SQL statement below (Query Duration=0 sec, SCN: 0x0819.003f594b):
Wed Apr 19 16:47:06 2006
select * from emp as of scn 8903471356235 where empno in (7788,7782,7698)
```

```
Wed Apr 19 16:47:06 2006
Errors in file /opt/oracle/admin/conner/udump/conner_ora_1274.trc:
ORA-01555: snapshot too old: rollback segment number 6 with name "???" too small
```

注意以上日志信息，触发 ORA-01555 错误的语句被记录，出现错误的 SCN 也被纪录，这个 SCN: 0x0819.003f594b 正是之前提到的 ctl max scn: 0x0819.003f594b，进一步的，找到 conner_ora_1274.trc 跟踪文件，从中可以获得关于这次错误的相信信息用于诊断。下面从跟踪文件中摘录一点点重要信息，简要说明：

1．错误信息

```
*** 2006-04-19 16:47:06.606
ksedmp: internal or fatal error
ORA-01555: snapshot too old: rollback segment number 6 with name "???" too small
Current SQL statement for this session:
select * from emp as of scn 8903471356235 where empno in (7788,7782,7698)
```

2．数据块信息

这里的块头就包含了 ITL 信息，根据这个 ITL 信息中的 UBA，Oracle 可以去定位回滚段，查询前镜像信息，如果不存在，就可能出现 ORA-01555 错误。

```
Block header dump:  0x00405c5a
 Object id on Block? Y
 seg/obj: 0x1f19  csc: 0x819.407961  itc: 2  flg: O  typ: 1 - DATA
      fsl: 0  fnx: 0x0 ver: 0x01

 Itl           Xid                  Uba              Flag  Lck        Scn/Fsc
0x01  0x0009.02e.000036b0  0x00800081.0302.11  C---    0  scn 0x0819.0036f205
0x02  0x0006.018.000036ce  0x00800055.02de.3f  C---    0  scn 0x0819.0040791b
```

根据这里的 Xid，可以进一步的获得回滚段号，事务槽等信息，主动查询或转储回滚段信息进行研究观察。

8.12 Oracle10g 闪回查询特性的增强

Oracle9i 提供的闪回特性增强，为数据恢复带来了极大的方便，但是 Oracle9i 的闪回查询只能提供某个时间点的数据视图，并不能告诉我们这样的数据经过了几个事务、怎样的修改（update,insert,delete 等），而这些信息在回滚段中是存在的，在 Oracle10g 中，Oracle 进一步加强了闪回查询的特性，提供了以下两种闪回查询：

闪回版本查询- Flashback Versions Query

闪回事务查询- Flashback Transaction Query

闪回版本查询允许使用一个新的 VERSIONS 子句查询两个时间点或者 SCN 之间的数据版本。这些版本可以按照事务进行区分，闪回版本查询只返回提交数据，未提交数据不被显示。

通过以下示例，来理解闪回版本查询的作用。首先创建一个测试表，执行一系列的 DML 操作：

```
SQL>create table t as select username,user_id from dba_users;
Table created.
SQL>select * from t;
USERNAME                                 USER_ID
------------------------------ ----------
SYSTEM                                        5
SYS                                           0
TEST                                         25
EYGLE                                        26
SCOTT                                        29
DIP                                          19
TRANS                                        27
TEST1                                        28
OPERATOR                                     31
WMSYS                                        23
DBSNMP                                       22
OUTLN                                        11

12 rows selected.
SQL>delete from t where username='OUTLN';
1 row deleted.
SQL>commit;
Commit complete.
SQL>delete from t where username='TEST1';
1 row deleted.
SQL>commit;
Commit complete.
```

再执行一系列 DML 操作并提交：

```
SQL>update t set user_id=1 where username='EYGLE';
1 row updated.
SQL>commit;
Commit complete.
SQL>delete from t where user_id >10;
7 rows deleted.
SQL>commit;
Commit complete.
SQL>select * from t;
```

```
USERNAME                                 USER_ID
------------------------------ ----------
SYSTEM                                    5
SYS                                       0
EYGLE                                     1
SQL>insert into t values('PENNY',2);
1 row created.
SQL>commit;
Commit complete.
```

至此数据库中已经交替执行了多个事务，进行了众多的数据修改，现在的测试表已经与最初完全不同了。如果使用 Oracle9i 的闪回查询，是很难区分这些不同事务的变更，找到合适的、正确的数据将变得极为困难。

再来看看 Oracle10g 的闪回版本查询，通过使用 versions 子句和对数据表引入了一系列的伪列（version_starttime 等），我们可以获得对数据表的所有事务操作，注意以下输出中 versions_operation 代表了不同类型的操作（D-Delete,I-Insert,U-Update）, VERSIONS_XID 是一个重要数据，代表了不同版本的事务 ID：

```
SQL>select versions_starttime, versions_endtime, versions_xid,
  2              versions_operation, username,user_id
  3              from t versions between timestamp minvalue and maxvalue;
VERSIONS_STARTTIME VERSIONS_ENDTIME     VERSIONS_XID       V USERNAME    USER_ID
---------------------- ---------------------- ---------------- - ------------------
30-MAR-05 09.34.49 AM                          000A000B000000F1 D DBSNMP            22
30-MAR-05 09.34.49 AM                          000A000B000000F1 D WMSYS             23
30-MAR-05 09.34.49 AM                          000A000B000000F1 D OPERATOR          31
30-MAR-05 09.34.49 AM                          000A000B000000F1 D TRANS             27
30-MAR-05 09.34.49 AM                          000A000B000000F1 D DIP               19
30-MAR-05 09.34.49 AM                          000A000B000000F1 D SCOTT             29
30-MAR-05 09.34.49 AM                          000A000B000000F1 D TEST              25
30-MAR-05 09.34.15 AM                          0001001900000F0F U EYGLE              1
30-MAR-05 09.33.51 AM                          00080016000000EF D TEST1             28
30-MAR-05 09.33.23 AM                          0004000A000005EF D OUTLN             11
                                                                 SYSTEM              5
                                                                 SYS                 0
                        30-MAR-05 09.34.49 AM                    TEST               25
                        30-MAR-05 09.34.15 AM                    EYGLE              26
                        30-MAR-05 09.34.49 AM                    SCOTT              29
                        30-MAR-05 09.34.49 AM                    DIP                19
                        30-MAR-05 09.34.49 AM                    TRANS              27
                        30-MAR-05 09.33.51 AM                    TEST1              28
```

```
                    30-MAR-05 09.34.49 AM                    OPERATOR        31
                    30-MAR-05 09.34.49 AM                    WMSYS           23
                    30-MAR-05 09.34.49 AM                    DBSNMP          22
                    30-MAR-05 09.33.23 AM                    OUTLN           11
30-MAR-05 09.49.24 AM                    00080006000000EF I PENNY           2
23 rows selected.
```

通过以上输出，我们根据 VERSIONS_XID 可以清晰地区分不同事务在不同时间对数据所作的更改。具备了 flashback version query 查询的基础，就可以进行基于 flashback version query 的事务级恢复，这就是 flashback transaction query。

flashback transaction query 可以从 FLASHBACK_TRANSACTION_QUERY 视图中获得指定事务的历史信息以及 Undo_SQL，通过这个 UNDO_SQL，我们就可以撤销特定的提交事务。

Flashback transaction query 需要用到 FLASHBACK_TRANSACTION_QUERY 视图，我们先看一下视图

```
SQL> desc FLASHBACK_TRANSACTION_QUERY;
Name                 Type                Nullable Default Comments
-------------------- ------------------- -------- ------- --------------------------
XID                  RAW(8)              Y        Transaction identifier
START_SCN            NUMBER              Y        Transaction start SCN
START_TIMESTAMP      DATE                Y        Transaction start timestamp
COMMIT_SCN           NUMBER              Y        Transaction commit SCN
COMMIT_TIMESTAMP     DATE                Y        Transaction commit timestamp
LOGON_USER           VARCHAR2(30)        Y        Logon user for transaction
UNDO_CHANGE#         NUMBER              Y        1-based undo change number
OPERATION            VARCHAR2(32)        Y        forward operation for this undo
TABLE_NAME           VARCHAR2(256)       Y        table name to which this undo applies
TABLE_OWNER          VARCHAR2(32)        Y        owner of table to which this undo applies
ROW_ID               VARCHAR2(19)        Y        rowid to which this undo applies
UNDO_SQL             VARCHAR2(4000)      Y        SQL corresponding to this undo
```

该视图的定义为：

```
select xid, start_scn, start_timestamp,
       decode(commit_scn, 0, commit_scn, 281474976710655, NULL, commit_scn)
       commit_scn, commit_timestamp,
       logon_user, undo_change#, operation, table_name, table_owner,
       row_id, undo_sql
from sys.x$ktuqqry
```

继续前面的测试，如果需要撤销 XID=000A000B000000F1 的事务，可以通过如下步骤进行（注意当查询 FLASHBACK_TRANSACTION_QUERY 视图时如果直接引用 XID 则查询会因为无法使用索引而极其耗时）：

```
SQL>set autotrace on
```

```
SQL>SELECT UNDO_SQL FROM FLASHBACK_TRANSACTION_QUERY
  2    WHERE XID = '000A000B000000F1';
UNDO_SQL
--------------------------------------------------------------------------------
insert into "EYGLE"."T"("USERNAME","USER_ID") values ('DBSNMP','22');
insert into "EYGLE"."T"("USERNAME","USER_ID") values ('WMSYS','23');
insert into "EYGLE"."T"("USERNAME","USER_ID") values ('OPERATOR','31');
insert into "EYGLE"."T"("USERNAME","USER_ID") values ('TRANS','27');
insert into "EYGLE"."T"("USERNAME","USER_ID") values ('DIP','19');
insert into "EYGLE"."T"("USERNAME","USER_ID") values ('SCOTT','29');
insert into "EYGLE"."T"("USERNAME","USER_ID") values ('TEST','25');
8 rows selected.
Elapsed: 00:05:55.30
Execution Plan
----------------------------------------------------------
    0          SELECT STATEMENT Optimizer=ALL_ROWS (Cost=25 Card=1 Bytes=2008)
    1      0     FIXED TABLE (FULL) OF 'X$KTUQQRY' (TABLE (FIXED)) (Cost=25 Card=1 Bytes=2008)


Statistics
----------------------------------------------------------
       393454  recursive calls
            0  db block gets
      1562425  consistent gets
         4644  physical reads
            0  redo size
         1069  bytes sent via SQL*Net to client
          664  bytes received via SQL*Net from client
            2  SQL*Net roundtrips to/from client
        23166  sorts (memory)
            0  sorts (disk)
            8  rows processed
```

此时可以通过 hextoraw 转换利用底层索引，提高查询速度：

```
SQL> set autotrace trace explain
SQL> SELECT UNDO_SQL FROM FLASHBACK_TRANSACTION_QUERY
  2    WHERE XID = hextoraw('09000D00A3080000');
Elapsed: 00:00:00.00
Execution Plan
----------------------------------------------------------
Plan hash value: 1747778896
```

```
-------------------------------------------------------------------------------------
| Id  | Operation                         | Name                    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                  |                         |     1 |  2008 |     0   (0)| 00:00:01 |
|*  1 |  FIXED TABLE FIXED INDEX| X$KTUQQRY (ind:1) |     1 |  2008 |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("XID"=HEXTORAW('09000D00A3080000') )
```

通过执行相应的 UNDO 语句我们可以撤销该事务，通过这些新特性，Oracle 为我们提供了一种“回滚”提交事务的手段，极大的方便了我们应对不同情况的数据库恢复。

8.13　ORA-01555 成因与解决

前面提到了 ORA-01555 错误，那么现在让我们看一下 ORA-01555 错误是怎样产生的。

我们知道，回滚段是循环使用的，当事务提交以后，该事务占用的回滚段事务表会被标记为非活动，回滚段空间可以被覆盖重用。

那么一个问题就出现了，如果一个查询需要使用被覆盖的回滚段构造前镜像实现一致性读，那么此时就会出现 Oracle 著名的 ORA-01555 错误。

ORA-01555 错误的另外一个原因是因为延迟块清除（Delayed Block Cleanout）。当一个查询触发延迟块清除时，Oracle 需要去查询回滚段获得该事务的提交 SCN，如果事务的前镜像信息已经被覆盖，并且查询 SCN 也小于回滚段中记录的最小提交 SCN，那么 Oracle 将无从判断查询 SCN 和事务提交 SCN 的大小，此时出现延迟块清除导致的 ORA-01555 错误。

另外一种导致 ORA-01555 错误的情况出现在使用 sqlldr 直接方式加载（direct=true）数据时。当通过 sqlldr direct=true 方式加载数据时，由于不产生重做和回滚信息，Oracle 直接指定 Cached Commit SCN 给加载数据，在访问这些数据时，有时会产生 ORA-01555 错误。

看一下下图的描述：假定在时间 T 用户 A 发出一条更新语句，更新 SCOTT 用户的 SAL；用户 B 在 Ty 时间发出查询语句，查询 SCOTT 用户的 SAL；用户 A 的更新在 Tx 时间提交，提交可能为快速提交清除，也可能是延迟块清除；用户 B 的查询在 Tz 时间输出。

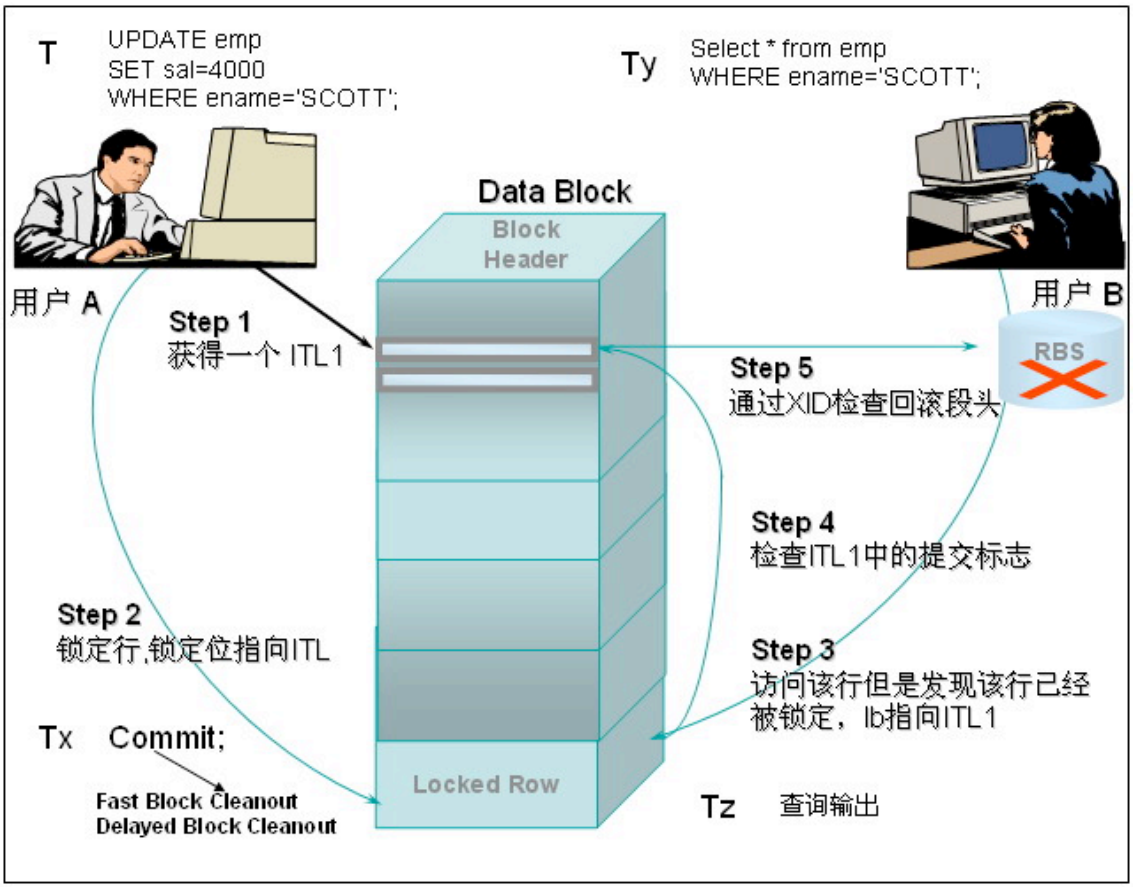
我们来看一下数据库在不同情况下的内部处理。

1. 如果 Ty < T< Tz <Tx,那么查询需要构造一致性读，由于事务尚未提交，可以通过回滚段构造前镜像，完成一致性读取。
2. 如果 Ty < T < Tx < Tz，由于 Ty 查询时间小于 T 事务更新时间，那么数据库需要构造一致性读取，而 Tz 查询完成时间大于 Tx 提交时间，那么前镜像就有可能被覆盖，不可获取。

   如果 Tx 的提交方式为 Fast Block Cleanout，那么回滚段信息不可用时就会出现一

致性读 ORA-01555 错误。

如果 Tx 的提交方式为 Delayed Block Cleanout，那么回滚段信息不可用时 Oracle 将无法判断 Ty 和 Tx 的时间先后关系。如果 Ty>Tx,那么 Oracle 可以正常进行块清除，并将块清除后的数据返回给用户 B；如果 Ty<T，那么 Oracle 需要继续构造一致性读返回给用户 B；Oracle 无法判断这两种情况，就会出现**延迟块清除 ORA-01555 错误。**



ORA-01555 的直观解释是“Snapshot too old”，也就是快照太旧，其根本含义就是查询需要的前镜像过于“久远”，已经无法找到了。

我们可以想象，如果一个历时数个小时或 10 数小时的查询，如果最后遭遇 ORA-01555 错误而失败，会是多么令人沮丧的一件事。一直以来，ORA-01555 都是 ORACLE 最为头痛的问题之一。

当数据出现 ORA-01555 错误时，Oracle 会将错误信息记录在警告日志中，从生产数据库中摘录 ORA-01555 错误信息举例如下。

这是一段 ORA-01555 错误信息，其中 Query Duration=14616 sec 指查询经历的时间，这个时间已经超过了 undo_retention 的缺省值 10800 的设置：

```
Mon Aug    1 15:03:39 2005
ORA-01555 caused by SQL statement below (Query Duration=14616 sec, SCN: 0x0000.1e5294a9):
Mon Aug    1 15:03:39 2005
select sp.vc2planguid
                          from cy_spinfo          pvd,
                               cy_serviceinfo svr,
                               cy_serviceplan sp,
                               cy_feetype         ft
                         where pvd.vc2spguid = svr.vc2spguid and
                               svr.vc2serviceguid = sp.vc2serviceguid and
                               sp.vc2ftguid = ft.vc2ftguid and

                               to_char(sysdate, 'yyyyMMddhh24miss') between
                               nvl(pvd.vc2startdate,
                                    to_char(sysdate, 'yyyyMMddhh24miss')) and
                               nvl(pvd.vc2enddate,
                                    to_char(sysdate, 'yyyyMMddhh24miss')) and

                               to_char(sysdate, 'yyyyMMddhh24miss') between
                               nvl(sp.vc2startdate,
                                    to_char(sysdate, 'yyyyMMddhh24miss')) and
                               nvl(svr.vc2enddate,
                                    to_char(sysdate, 'yyyyMMddhh24miss')) and
```

这个错误是由于一个 job 任务的执行导致的:

```
Mon Aug    1 15:03:39 2005
Errors in file /opt/oracle/admin/hsboss/bdump/hsboss_j000_1088.trc:
ORA-12012: error on auto execute of job 181
ORA-01555: snapshot too old: rollback segment number 8 with name "_SYSSMU8$" too small
ORA-06512: at "CYUSER.CYPKG_BILLING", line 385
ORA-06512: at line 1
```

在 Oracle9i 的文档中这样描述 ORA-01555 错误：

```
ORA-01555 snapshot too old: rollback segment number string with name "string" too small
Cause: Rollback records needed by a reader for consistent read are overwritten by other writers.

Action: If in Automatic Undo Management mode, increase the setting of UNDO_RETENTION. Otherwise, use
larger rollback segments.
```

我们看到，在 Oracle9i 自动管理的 UNDO 表空间模式下，UNDO_RETENTION 参数的引入正是为了减少 ORA-01555 错误的出现。

这个参数设置当事务提交之后（回滚段变得非激活），回滚段中的前镜像数据在被覆盖前

保留的时间，该参数以秒为单位，9iR1 初始值为 900 秒，在 Oracle9iR2 增加为 10800 秒。

显然该参数设置的越高就越能够减少 ORA-01555 错误的出现，但是保留时间和存储空间是紧密相关的，如果 UNDO 表空间的存储空间有限，那么 ORACLE 就会选择回收已提交事务占用的空间，置 UNDO_RETENTION 参数的设置于不顾。

在 Oracle9i 的 AUM 模式下，UNDO_RETENTION 实际上是一个非但保（NO Guaranteed）限制。也就是说，如果有其他事务需要回滚空间，而空间出现不足时，这些信息仍然会被覆盖；从 Oracle10g 开始，Oracle 对于 UNDO 增加了 Guarantee 控制，也就是说，你可以指定 UNDO 表空间必须满足 UNDO_RETENTION 的限制。当 UNDO 表空间设置为 Guarantee，那么提交事务的回滚空间必须被保留足够的时间，如果 UNDO 表空间的空间不足，那么新事务会因空间不足而失败，而不是选择之前的覆盖。

从各个不同版本回滚段的管理变迁，我们可以看出，Oracle 一直在进步。

Oracle 提供一个内部事件（10203 事件）可以用来跟踪数据库的块清除操作， 10203 事件可以通过以下命令设置，设置后需要重新启动数据库该参数方能生效:

alter system set event="10203 trace name context forever" scope=spfile;

设置了 10203 事件之后，可以通过实验来研究一下 ORA-01555 错误的成因。为了实验的方便，首先创建一个手工管理不可扩展的小 Undo 表空间：

```
SQL> create undo tablespace undotbs
  2    datafile '/opt/oracle/oradata/conner/undotbs.dbf' size 2m autoextend off;
Tablespace created.
SQL> alter system set undo_tablespace=undotbs;
System altered.
SQL> alter system set undo_management=manual scope=spfile;
System altered.
SQL> alter system set event="10203 trace name context forever" scope=spfile;
System altered.
SQL> shutdown immediate;
SQL> startup
SQL> select * from v$rollname;
            USN NAME
---------- ------------------------------
              0 SYSTEM
SQL> create rollback segment rbs01 tablespace undotbs;
Rollback segment created.
SQL> alter rollback segment rbs01 online;
Rollback segment altered.
SQL> select * from v$rollname;
            USN NAME
---------- ------------------------------
```

```
         0 SYSTEM
        21 RBS01
```

再来执行一个任务，首先打开一个游标，执行一个查询（在适当步骤加入了数据块转储及回滚段转储命令,并将跟踪信息同步进行讲解）：

```
SQL> var cemp refcursor
SQL> begin
  2      open :cemp for select * from emp where empno=7788;
  3   end;
  4   /
PL/SQL procedure successfully completed.
SQL> alter system dump datafile 1 block 23642;
System altered.
```

此时无 DML 事务进行，数据块 ITL 状态如下所示：

```
 Itl             Xid                  Uba         Flag  Lck        Scn/Fsc
0x01   0x0015.045.000003b7  0x010000b1.2b39.04  C---    0  scn 0x0819.0045cd56
0x02   0x0015.03f.000003b6  0x010000b6.2b2b.06  C---    0  scn 0x0819.0045cb5c
```

然后更新一条记录并且提交，同时转储数据块的内容：

```
SQL> update emp set sal=4000 where empno=7788;
1 row updated.
SQL> alter system dump datafile 1 block 23642;
System altered.
```

此时数据块上 ITL 及具体记录上都记录了锁定信息，本例里 ITL2（0x02）被使用：

```
Itl             Xid                  Uba         Flag  Lck        Scn/Fsc
0x01   0x0015.045.000003b7  0x010000b1.2b39.04  C---    0  scn 0x0819.0045cd56
0x02   0x0015.049.000003b8  0x010000ac.2b46.02  ----    1  fsc 0x0000.00000000
```

数据行 LB 指向了 ITL 2:

```
tab 0, row 7, @0x1e7f
tl: 40 fb: --H-FL-- lb: 0x2  cc: 8
col  0: [ 3]  c2 4e 59
col  1: [ 5]  53 43 4f 54 54
col  2: [ 7]  41 4e 41 4c 59 53 54
col  3: [ 3]  c2 4c 43
col  4: [ 7]  77 bb 04 13 01 01 01
col  5: [ 2]  c2 29
col  6: *NULL*
col  7: [ 2]  c1 15
```

继续 SQL*Plus 中的操作，转储一下回滚段头信息：

```
SQL> alter system dump undo header RBS01;
System altered.
```

回滚段第 0x49 号事务槽被使用：

```
index  state cflags  wrap#    uel         scn            dba         parent-xid    nub
stmt_num
0x49   10     0x80   0x03b8   0x0000   0x0819.0045cea3   0x010000ac   0x0000.000.00000000
0x00000001   0x00000000
```

继续，提交该事务，并再次转储回滚段头信息：

```
SQL> commit;
Commit complete.
SQL> alter system dump undo header RBS01;
System altered.
```

可以看到该事务已经被提交，回滚段事务状态变为非激活（state=9），提交的 SCN 为：**0x0819.0045cea4** 。

```
index  state cflags  wrap#    uel         scn            dba         parent-xid    nub
stmt_num
0x49   9      0x80   0x03b8   0xffff   0x0819.0045cea4   0x010000ac   0x0000.000.00000000
0x00000001   0x00000000
```

继续，我们来看此时的数据块：

```
SQL> alter system dump datafile 1 block 23642;
System altered.
```

ITL 已经记录了事务提交，SCN/FSC 表示 Commit SCN 或快速提交（Fast Commit Fsc）的 SCN，这个 SCN 和 UNDO 中的提交 SCN 一致，是准确的 SCN：

```
Itl       Xid                  Uba              Flag  Lck        Scn/Fsc
0x01   0x0015.045.000003b7  0x010000b1.2b39.04  C---    0  scn 0x0819.0045cd56
0x02   0x0015.049.000003b8  0x010000ac.2b46.02  --U-    1  fsc 0x0000.0045cea4
```

此时的数据行锁定位并不需要清除：

```
tab 0, row 7, @0x1e7f
tl: 40 fb: --H-FL-- lb: 0x2   cc: 8
col  0: [ 3]  c2 4e 59
col  1: [ 5]  53 43 4f 54 54
col  2: [ 7]  41 4e 41 4c 59 53 54
col  3: [ 3]  c2 4c 43
col  4: [ 7]  77 bb 04 13 01 01 01
col  5: [ 2]  c2 29
col  6: *NULL*
col  7: [ 2]  c1 15
```

此时 10203 事件的跟踪信息被记录，块清除记录了确切的 SCN 信息：

```
Begin cleaning out block ...
Found active transactions
Block cleanout record, scn:  0x0819.0045cea6 ver: 0x01 opt: 0x02, entries follow...
```

```
  itli: 2  flg: 2  scn: 0x0819.0045cea4
Block cleanout under the cache...
Block cleanout record, scn:  0x0819.0045cea6 ver: 0x01 opt: 0x02, entries follow...
  itli: 2  flg: 2  scn: 0x0819.0045cea4
... clean out dump complete.
Start dump data blocks tsn: 0 file#: 1 minblk 23642 maxblk 23642
```

继续我们在 SQL*Plus 中的操作，执行一个批处理更新，由于我们只有一个用户回滚段，先前的事务信息很快被覆盖：

```
SQL> begin
  2     for i in 1 .. 100 loop
  3     update emp set sal=4000;
  4     rollback;
  5     end loop;
  6  end;
  7  /
PL/SQL procedure successfully completed.
SQL> alter system dump datafile 1 block 23642;
```

我们再看此时的数据块信息，数据块被写出，锁定位被清除，ITL Flag 标志位置为提交状态（C-Commit）：

```
Itl           Xid                  Uba            Flag  Lck        Scn/Fsc
0x01  0x0015.045.000003b7  0x010000b1.2b39.04  C---    0  scn 0x0819.0045cd56
0x02  0x0015.049.000003b8  0x010000ac.2b46.02  C---    0  scn 0x0819.0045cea4
```

数据行锁定位锁定信息清除:

```
tab 0, row 7, @0x1e81
tl: 40 fb: --H-FL-- lb: 0x0  cc: 8
col  0: [ 3]  c2 4e 59
col  1: [ 5]  53 43 4f 54 54
col  2: [ 7]  41 4e 41 4c 59 53 54
col  3: [ 3]  c2 4c 43
col  4: [ 7]  77 bb 04 13 01 01 01
col  5: [ 2]  c2 29
col  6: *NULL*
col  7: [ 2]  c1 15
```

最后输出游标查询结果，由于游标打开时间在更新操作之前，Oracle 需要构造一致性读，获取前镜像信息，而前镜像信息已经被覆盖，所以出现 ORA-01555 错误，这是最常见的 1555 错误来源，时间过长的查询很容易因 ORA-01555 错误而失败：

```
SQL> print :cemp
ERROR:
ORA-01555: snapshot too old: rollback segment number 21 with name "RBS01" too small
```

```
no rows selected
```

再来看看第二种情况，同样构造一个游标打开进行查询：

```
SQL> var cemp refcursor
SQL> begin
  2      open :cemp for select * from emp where empno=7788;
  3   end;
  4   /
PL/SQL procedure successfully completed.
```

然后更新数据：

```
SQL> update emp set sal=4000 where empno=7788;
1 row updated.
SQL> alter system dump undo header RBS01;
System altered.
```

此时的 UNDO 事务表事务槽信息如下：

```
index  state cflags  wrap#     uel       scn          dba          parent-xid    nub
stmt_num
0x43     10     0x80    0x03bd    0x0001    0x0819.0045f09c    0x010000b7    0x0000.000.00000000
0x00000001    0x00000000
```

在提交之前，我们强制刷新 Buffer Cache，写出脏数据：

```
SQL> alter session set events = 'immediate trace name flush_cache';
Session altered.
```

然后提交，此时，Oracle 将执行延迟块清除：

```
SQL> commit;
Commit complete.
SQL> alter system dump datafile 1 block 23642;
System altered.
SQL> alter system dump undo header RBS01;
System altered.
```

由于事务已经提交，回滚段事务表已经被标记为非活动，此时的提交 SCN 为：**0x0819.0045f09e ：**

```
index  state cflags  wrap#     uel       scn          dba          parent-xid    nub
stmt_num
0x43     9      0x80    0x03bd    0xffff    0x0819.0045f09e    0x010000b7    0x0000.000.00000000
0x00000001    0x00000000
```

此时的数据块已经被写出到数据文件，Oracle 执行延迟块清除，ITL 事务槽及锁定信息仍然在 Block 上存在：

```
Itl       Xid                  Uba            Flag  Lck        Scn/Fsc
0x01   0x0015.043.000003bd  0x010000b8.2b7b.04  ----    1    fsc 0x0000.00000000
```

```
0x02    0x0015.041.000003bd   0x010000b8.2b7b.02   C---      0   scn 0x0819.0045f093
```

执行批处理更新，覆盖前镜像信息：

```
SQL> begin
  2      for i in 1 .. 100 loop
  3      update emp set sal=4000;
  4      rollback;
  5      end loop;
  6  end;
  7  /
PL/SQL procedure successfully completed.
```

此时的 ORA-01555 错误就是因为延迟块清除所导致的：

```
SQL> print :cemp
ERROR:
ORA-01555: snapshot too old: rollback segment number 21 with name "RBS01" too small


no rows selected
```

此时的延迟块清除被跟踪记录：

```
Begin cleaning out block ...
Found all committed transactions
Block cleanout record, scn:   0xffff.ffffffff ver: 0x01 opt: 0x01, entries follow...
  itli: 1   flg: 2   scn: 0x0819.0045f09e
Block cleanout under the cache...
Block cleanout record, scn:   0x0819.0045f09f ver: 0x01 opt: 0x01, entries follow...
  itli: 1   flg: 2   scn: 0x0819.0045f09e
... clean out dump complete.
```

如果此时 DUMP 日志文件，也可以看到块清除信息（我们看到此时，由于延迟块清除也产生了日志，也就是在查询时可能看到的日志生成）：

```
REDO RECORD - Thread:1 RBA: 0x000068.00000006.0010 LEN: 0x003c VLD: 0x01
SCN: 0x0819.0045f09f SUBSCN:   1 04/20/2006 21:50:33
CHANGE #1 TYP:0 CLS: 1 AFN:1 DBA:0x00405c5a SCN:0x0819.0045f09c SEQ:   1 OP:4.1
Block cleanout record, scn:   0x0819.0045f09f ver: 0x01 opt: 0x01, entries follow...
  itli: 1   flg: 2   scn: 0x0819.0045f09e
```

这里的 OP(Operation Code):4.1 就是指 Block Cleanout 。来看一下此时的数据块信息：

```
SQL> alter system dump datafile 1 block 23642;
System altered.
```

数据块的 ITL 和锁定信息都被清除：

```
Itl           Xid                  Uba            Flag  Lck        Scn/Fsc
0x01   0x0015.043.000003bd  0x010000b8.2b7b.04  C---    0   scn 0x0819.0045f09e
```

```
0x02    0x0015.041.000003bd    0x010000b8.2b7b.02    C---        0    scn 0x0819.0045f093
```

需要注意的是，可能存在另外一种情况，就是当执行延迟块清除时，回滚段或原回滚表空间已经被删除，此时 Oracle 仍然可以通过字典表 UNDO$来获得 SCN 信息，执行块清除。

关于 Oracle 的提交处理及块清除机制是一个极其复杂的过程，本文对这部分内容进行了适当简化说明，旨在让大家能够对 Oracle 的回滚机制，块清除机制有所了解。

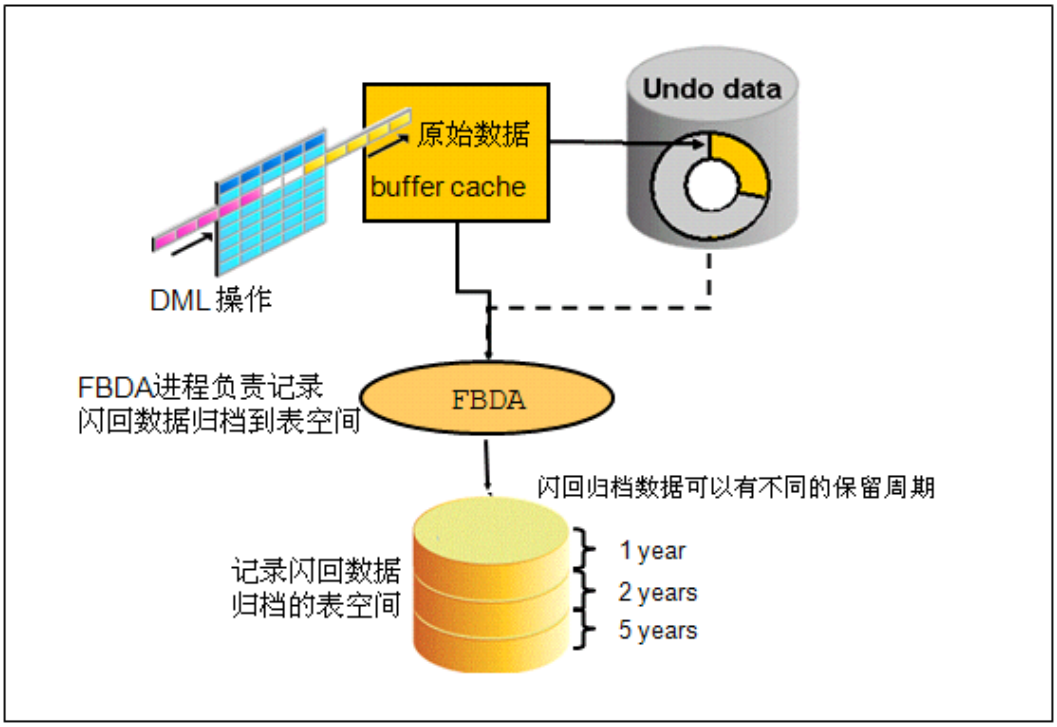
8.14 Oracle11g 闪回数据归档

虽然 ORA-01555 错误可以通过种种手段来避免和减少，但是随着时间的流逝，这些 UNDO 信息总会失去，那么能否将这些信息保存起来，使得数据库在一定的历史周期之内可以不断向后追溯，使得我们可以看到一个数据表在任意历史时间点上的切片呢？

从 Oracle Database 11g 开始，Oracle 提供了一个这样的功能：闪回数据归档（Flashback Data Archive）。通过这一功能 Oracle 数据库可以将 UNDO 数据进行归档，从而提供全面的历史数据查询，也因此 Oracle 引入一个新的概念 **Oracle Total Recall**，也即 Oracle 全面回忆功能。闪回数据归档可以和我们一直熟悉的日志归档类比，日志归档记录的是 Redo 的历史状态，用于保证恢复的连续性；而闪回归档记录的是 UNDO 的历史状态，可以用于对数据进行闪回追溯查询；后台进程 LGWR 用于将 Redo 信息写出到日志文件，ARCH 进程负责进行日志归档；在 Oracle11g 中，新增的后台进程 FBDA（Flashback data archiver process）则用于对闪回数据进行归档写出：

```
[oracle@localhost ~]$ ps -ef|grep fbda|grep -v grep
oracle      7627        1    0 Jul10 ?            00:00:04 ora_fbda_11gtest
```

下图对闪回数据归档的机制做出了简要描述：



闪回归档数据甚至可以以年为单位进行保存，Oracle 可以通过内部分区和压缩算法减少空间耗用，这一特性对于需要审计以及历史数据分析的环境尤其有用，但是注意，对于繁忙的数据库环境，闪回数据存储显然要好用更多的存储空间。当然，我们可以根据需要，对部分表进行闪回数据归档，从而满足特定的业务需求。

因为闪回数据归档需要独立的存储，所以在使用该特性之前需要创建独立的 ASSM 表空间：

```
SQL> create tablespace fbra datafile size 200M
  2    segment space management auto;
Tablespace created.
```

然后可以基于该表空间创建闪回数据归档区， FLASHBACK ARCHIVE ADMINISTER 系统权限是创建闪回数据存档所必需的，此处使用 SYS 用户进行：

```
SQL> create flashback archive dataarchive tablespace fbra retention 1 month;
Flashback archive created.
```

此后就可以使用该归档区来记录数据表的闪回数据量。为了测试方便，先将 UNDO 表空间更改为较小，以使得 UNDO 数据能够尽快老化：

```
SQL> alter database datafile '/data1/oradata/11gtest/11gtest/undotbs01.dbf' resize 20M;
Database altered.
SQL> alter database datafile '/data1/oradata/11gtest/11gtest/undotbs01.dbf' autoextend off;
Database altered.
```

接下来使用测试用户连接，对测试表执行闪回归档设置，FLASHBACK ARCHIVE 对象权限是启用历史数据跟踪所必需的：

```
SQL> connect eygle/eygle
Connected.
SQL> select * from tab;
TNAME                          TABTYPE  CLUSTERID
------------------------------ ------- ----------
EYGLE                          TABLE
SQL> alter table eygle flashback archive dataarchive;
Table altered.
```

注意：取消对于数据表的闪回归档可以使用如下命令：

alter table *table_name* no flashback archive;

接下来记录一下 SCN，从数据表中删除部分数据：

```
SQL> select dbms_flashback.get_system_change_number from dual;
GET_SYSTEM_CHANGE_NUMBER
------------------------
                 1107235
SQL> select count(*) from eygle;
  COUNT(*)
----------
      4000
SQL> delete from eygle where rownum <1001;
1000 rows deleted.
SQL> commit;
Commit complete.
```

现在执行闪回查询，则数据来自 UNDO 表空间：

```
SQL> select count(*) from eygle as of scn 1107235;
  COUNT(*)
----------
      4000
Execution Plan
----------------------------------------------------------
Plan hash value: 3602634261
--------------------------------------------------------------------------
| Id  | Operation                 | Name  | Rows  | Cost (%CPU)| Time     |
--------------------------------------------------------------------------
|   0 | SELECT STATEMENT          |       |     1 |    76   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE           |       |     1 |            |          |
|   2 |   TABLE ACCESS FULL| EYGLE |  6126 |    76   (0)| 00:00:01 |
--------------------------------------------------------------------------
```

接下来执行一小段批量循环代码，使 UNDO 数据老化覆盖：

```
SQL> begin
  2  for i in 1 .. 100 loop
  3  delete from eygle where rownum <31;
  4  commit;
  5  end loop;
  6  end;
  7  /
PL/SQL procedure successfully completed.
```

现在来看一下闪回数据归档发挥作用的闪回查询，通过执行计划能够看到和之前查询执行方式的不同：

```
SQL> select count(*) from eygle as of scn 1107235;

  COUNT(*)
---------
      4000

Execution Plan
----------------------------------------------------------
Plan hash value: 2549523072


-------------------------------------------------------------------------------------------------------------
| Id  | Operation                | Name                    | Rows  | Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |                         |     1 |       |   104   (2)| 00:00:02 |       |       |
|   1 |  SORT AGGREGATE          |                         |     1 |       |            |          |       |       |
|   2 |   VIEW                   |                         |  4030 |       |   104   (2)| 00:00:02 |       |       |
|   3 |    UNION-ALL             |                         |       |       |            |          |       |       |
|   4 |     PARTITION RANGE SINGLE|                        |  4000 |  101K|    18   (6)| 00:00:01 |     1 |     1 |
|*  5 |      TABLE ACCESS FULL   | SYS_FBA_HIST_16585      |  4000 |  101K|    18   (6)| 00:00:01 |     1 |     1 |
|*  6 |     FILTER               |                         |       |       |            |          |       |       |
|*  7 |      HASH JOIN OUTER     |                         |    30 | 60810 |    86   (2)| 00:00:02 |       |       |
|*  8 |       TABLE ACCESS FULL  | EYGLE                   |   306 |  3672 |    76   (0)| 00:00:01 |       |       |
|   9 |       VIEW               |                         |  4308 |  8477K|     9   (0)| 00:00:01 |       |       |
|* 10 |        TABLE ACCESS FULL | SYS_FBA_TCRV_16585      |  4308 |  8531K|     9   (0)| 00:00:01 |       |       |
-------------------------------------------------------------------------------------------------------------


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


   5 - filter("ENDSCN">1107235 AND "ENDSCN"<=1108486 AND ("STARTSCN" IS NULL OR "STARTSCN"<=1107235))
   6 - filter("F"."STARTSCN"<=1107235 OR "F"."STARTSCN" IS NULL)
```

```
   7 - access("T".ROWID=CHARTOROWID("F"."RID"(+)))
   8 - filter("T"."VERSIONS_STARTSCN" IS NULL)
  10 - filter(("ENDSCN" IS NULL OR "ENDSCN">1108486) AND ("STARTSCN" IS NULL OR "STARTSCN"<1108486))
Note
-----
   - dynamic sampling used for this statement
```

通过以上执行计划可以看到，查询闪回来自 SYS_FBA_TCRV_16585 系统表，该表隶属于闪回归档表空间，用于记录闪回数据：

```
SQL> desc    SYS_FBA_TCRV_16585
 Name                                                  Null?    Type
 ----------------------------------------- -------- ----------------------------
 RID                                                            VARCHAR2(4000)
 STARTSCN                                                         NUMBER
 ENDSCN                                                           NUMBER
 XID                                                            RAW(8)
 OP                                                             VARCHAR2(1)
SQL> select count(*) from    SYS_FBA_TCRV_16585 ;
  COUNT(*)
----------
         4308
```

闪回功能生成的字典对象有多个，通过查询 USER_TABLES/USER_OBJECTS 视图可以获得这些对象的详细信息：

```
SQL> select table_name,tablespace_name from user_tables where table_name like '%FBA%';
TABLE_NAME                     TABLESPACE_NAME
------------------------------ ------------------------------
SYS_FBA_DDL_COLMAP_16585          FBRA
SYS_FBA_TCRV_16585                FBRA
SYS_FBA_HIST_16585
SQL> select object_name,object_type from user_objects where object_name like '%FBA%';
OBJECT_NAME                    OBJECT_TYPE
------------------------------ -------------------
SYS_FBA_HIST_16585             TABLE PARTITION
SYS_FBA_DDL_COLMAP_16585          TABLE
SYS_FBA_HIST_16585             TABLE
SYS_FBA_TCRV_16585              TABLE
SYS_FBA_TCRV_IDX_16585           INDEX
```

可以通过数据字典视图来查看关于闪回归档表的记录：

```
SQL>    select * from user_flashback_archive_tables;
TABLE_NAME OWNER_NAME FLASHBACK_ARCHIVE_NAME          ARCHIVE_TABLE_NAME
```

```
---------- ---------- ------------------------------ --------------------
EYGLE      EYGLE      DATAARCHIVE                    SYS_FBA_HIST_16585
```

可以通过 dict 字典查询和闪回归档有关的数据字典表：

```
SQL> select table_name from dict where table_name like '%FLASHBACK_ARCHIVE%';
TABLE_NAME
----------------------------------------------------
DBA_FLASHBACK_ARCHIVE
DBA_FLASHBACK_ARCHIVE_TABLES
DBA_FLASHBACK_ARCHIVE_TS
USER_FLASHBACK_ARCHIVE
USER_FLASHBACK_ARCHIVE_TABLES
```

总之，闪回数据归档是 Oracle11g 提供的重要增强之一，通过合理使用这一增强，可以为数据库提供更为全面的数据生命周期管理，Oracle 关于 UNDO 技术的进化至此又迈进了重要的一步。

8.15　AUM 下如何重建 UNDO 表空间

曾经有朋友问到，在迁移(同平台)的时候由于 UNDO 表空间过大，不打算要现在的 UNDO 文件，想要重建一个，该如何做，是否需要通过一些隐含参数来做特殊处理？前提是他拥有一个有效的冷备份（或者 Clean Shutdown 的数据库）。拥有冷备份，那么这个操作是很简单的，并不需要使用隐含参数。

以下是一个简单的测试过程，重建 UNDO 表空间的步骤和此类似。

1．假定拥有一个 Clean shutdown 的数据库（以 shutdown immediate 方式关闭并执行备份）

```
C:\Documents and Settings\gqgai>sqlplus "/ as sysdba"
SQL*Plus: Release 9.2.0.6.0 - Production on Fri Mar 4 20:55:59 2005
Copyright (c) 1982, 2002, Oracle Corporation.    All rights reserved.

SQL> shutdown immediate;
```

2．同平台迁移时可以放弃 UNDO 表空间，这时候启动会抱错 ORA-01157：

```
SQL> startup
ORACLE instance started.
。。。
Database mounted.
ORA-01157: cannot identify/lock data file 2 - see DBWR trace file
ORA-01110: data file 2: 'D:\ORADATA\EYGLE\UNDOTBS01.DBF'
```

3．删除 UNDO 文件启动数据库

```
SQL> alter database datafile 'D:\ORADATA\EYGLE\UNDOTBS01.DBF' offline drop;
Database altered.
```

```
SQL> alter database open;
Database altered.
SQL> select name from v$datafile;
NAME
-----------------------------------------------------
D:\ORADATA\EYGLE\SYSTEM01.DBF
D:\ORADATA\EYGLE\UNDOTBS01.DBF
D:\ORADATA\EYGLE\EYGLE.DBF
```

4．重建 UNDO 表空间，并切换为当前 UNDO 表空间

```
SQL> create undo tablespace undotbs2
  2    datafile 'd:\oradata\eygle\undotbs2.dbf' size 10M;
Tablespace created.
SQL> ALTER SYSTEM SET undo_tablespace='UNDOTBS2';
System altered.
```

然后数据库即可恢复正常使用。

8.16 使用 Flashback Query 恢复误删除数据

这是一个实际生产环境中的恢复案例，当时接到研发工程师的电话，说误删除了部分重要数据，并且已经提交，需要恢复。

登陆到数据库上查看，由于是 Oracle9iR2，首先尝试使用 flashback query 闪回数据。

首先确认数据库的 SCN 变化:

```
SQL> col fscn for 9999999999999999999
SQL> col nscn for 9999999999999999999
SQL> select name,FIRST_CHANGE# fscn,NEXT_CHANGE# nscn,FIRST_TIME from v$archived_log;
NAME                                          FSCN                NSCN          FIRST_TIME
--------------------------------- ------------- -------------- -------------------
.............................
/mwarch/oracle/1_52414.dbf          12929942881       12929943706 2005-06-22 14:38:32
/mwarch/oracle/1_52415.dbf          12929943706       12929944623 2005-06-22 14:38:35
/mwarch/oracle/1_52416.dbf          12929944623       12929945392 2005-06-22 14:38:38
/mwarch/oracle/1_52418.dbf          12929945392       12929945888 2005-06-22 14:38:41
/mwarch/oracle/1_52418.dbf          12929945888       12929945965 2005-06-22 14:38:44
/mwarch/oracle/1_52419.dbf          12929945965       12929948945 2005-06-22 14:38:45
/mwarch/oracle/1_52420.dbf          12929948945       12929949904 2005-06-22 14:46:05
/mwarch/oracle/1_52421.dbf          12929949904       12929950854 2005-06-22 14:46:08
/mwarch/oracle/1_52422.dbf          12929950854       12929951751 2005-06-22 14:46:11
/mwarch/oracle/1_52423.dbf          12929951751       12929952587 2005-06-22 14:46:14
```

```
.....

/mwarch/oracle/1_52498.dbf        12930138975     12930139212 2005-06-22 15:55:57
/mwarch/oracle/1_52499.dbf        12930139212     12930139446 2005-06-22 15:55:59
/mwarch/oracle/1_52500.dbf        12930139446     12930139682 2005-06-22 15:56:00
/mwarch/oracle/1_52501.dbf        12930139682     12930139915 2005-06-22 15:56:02
/mwarch/oracle/1_52502.dbf        12930139915     12930140149 2005-06-22 15:56:03
/mwarch/oracle/1_52503.dbf        12930140149     12930140379 2005-06-22 15:56:05
/mwarch/oracle/1_52504.dbf        12930140379     12930140610 2005-06-22 15:56:05
/mwarch/oracle/1_52505.dbf        12930140610     12930140845 2005-06-22 15:56:07

14811 rows selected.
```

当前的 SCN 为:

```
SQL> select dbms_flashback.get_system_change_number fscn from dual;
                      FSCN
--------------------
             12930142214
```

使用应用用户连接数据库尝试闪回：

```
SQL> connect username/password
Connected.
```

现有数据:

```
SQL> select count(*) from hs_passport;
  COUNT(*)
----------
    851998
```

创建恢复表:

```
SQL> create table hs_passport_recov as select * from hs_passport where 1=0;
Table created.
```

选择合适的 SCN 向前恢复：

```
SQL> select count(*) from hs_passport as of scn 12929970422;
  COUNT(*)
----------
    861686
```

尝试多个 SCN，获取最佳值（如果能得知具体时间，那么可以获得准确的数据闪回）

```
SQL> select count(*) from hs_passport as of scn &scn;
Enter value for scn: 12929941968
old   1: select count(*) from hs_passport as of scn &scn
new   1: select count(*) from hs_passport as of scn 12929941968
  COUNT(*)
```

```
----------
      861684
SQL> /
Enter value for scn: 12929928784
old      1: select count(*) from hs_passport as of scn &scn
new      1: select count(*) from hs_passport as of scn 12929928784
  COUNT(*)
----------
      825110
SQL> /
Enter value for scn: 12928000000
old      1: select count(*) from hs_passport as of scn &scn
new      1: select count(*) from hs_passport as of scn 12928000000
select count(*) from hs_passport as of scn 12928000000
                                *
ERROR at line 1:
ORA-01466: unable to read data - table definition has changed
```

最后选择恢复到 SCN 为 12929941968 的时间点

```
SQL> insert into hs_passport_recov select * from hs_passport as of scn 12929941968;
861684 rows created.
SQL> commit;
Commit complete.
```

研发人员确认，已经可以满足需要，找回误删除部分数据，至此闪回恢复成功完成。

8.17 诊断案例之-释放过度扩展的 Undo 空间

从 Oracle9i 开始，当我们使用 AUM 管理时，通常会选择设置 UNDO 表空间自动扩展，这就带来了另外一个问题，经常会出现 UNDO 表空间过度扩展而不能回缩的问题。这类问题有的是因为 Bug 引起的，以下的案例来自 Oracle10g，但是对于 Oracle9i 同样适用。

环境:

```
OS:Red Hat Enterprise Linux AS release 4 (Nahant)
DB:Oracle Database 10g Enterprise Edition Release 10.2.0.1.0 - Production
```

一台 Oracle10gR2 数据库报出如下错误:

```
ORA-1653: unable to extend table SYSMAN.MGMT_SYSTEM_ERROR_LOG by 8 in    tablespace SYSAUX
ORA-1653: unable to extend table SYSMAN.MGMT_SYSTEM_ERROR_LOG by 8 in    tablespace SYSAUX
ORA-1653: unable to extend table SYSMAN.MGMT_SYSTEM_ERROR_LOG by 8 in    tablespace SYSAUX
ORA-1653: unable to extend table SYSMAN.MGMT_SYSTEM_ERROR_LOG by 8 in    tablespace SYSAUX
ORA-1653: unable to extend table SYSMAN.MGMT_SYSTEM_ERROR_LOG by 8 in    tablespace SYSAUX
```

登陆检查，发现是 SYSAUX 表空间空间用尽,不能扩展,尝试手工扩展 SYSAUX 表空间:

```
alter database datafile '+ORADG/danaly/datafile/sysaux.266.600173881' resize 800m
Tue Nov 29 23:31:38 2005
ORA-1237 signalled during: alter database datafile '+ORADG/danaly/datafile/sysaux.266.600173881' resize 800m...
```

出现 ORA-1237 错误，提示空间不足。这时候我才认识到是磁盘空间可能被用完了。是谁"偷偷的"用了那么多空间呢(本来有几十个 G 的 Free 磁盘空间的)?

检查数据库表空间占用空间情况:

```
SQL> select tablespace_name,sum(bytes)/1024/1024/1024 GB
  2  from dba_data_files group by tablespace_name
  3  union all
  4  select tablespace_name,sum(bytes)/1024/1024/1024 GB
  5  from dba_temp_files group by tablespace_name order by GB;
TABLESPACE_NAME                                GB
------------------------------ ----------
UNDOTBS2                                .09765625
SYSTEM                                 .478515625
SYSAUX                                 .634765625
。。。。。。。
IVRCN_TS_DATA                                   2
MMS_TS_DATA1                                    2
CM_TS_DEFAULT                                   5
TEMP                                   20.5498047
UNDOTBS1                               28.1582031

15 rows selected.
```

不幸的发现，UNDO 表空间已经扩展至 27G，而 TEMP 表空间也扩展至 20G，这 2 个表空间加起来占用了 47G 的磁盘空间,导致了空间不足。

显然曾经有大事务占用了大量的 UNDO 表空间和 Temp 表空间，Oracle 的 AUM(Auto Undo Management)从出生以来就经常出现只扩展,不收缩(shrink)的情况(通常可以设置足够的 UNDO 表空间大小,然后取消其自动扩展属性).

现在可以采用如下步骤回收 UNDO 空间:

1.确认文件

```
SQL> select file_name,bytes/1024/1024 from dba_data_files
  2  where tablespace_name like 'UNDOTBS1';
FILE_NAME                                                BYTES/1024/1024
-------------------------------------------------- ---------------
+ORADG/danaly/datafile/undotbs1.265.600173875                      27810
```

2.检查 UNDO Segment 状态

发现有的回滚段大小已经扩展到了约 3G 的大小：

```
SQL> select usn,xacts,rssize/1024/1024/1024,hwmsize/1024/1024/1024,shrinks
  2  from v$rollstat order by rssize;
       USN      XACTS RSSIZE/1024/1024/1024 HWMSIZE/1024/1024/1024    SHRINKS
---------- ---------- ---------------------- ---------------------- ----------
         0          0           .000358582             .000358582          0
         2          0           .071517944             .071517944          0
         3          0            .13722229              .13722229          0
         9          0           .236984253             .236984253          0
        10          0           .625144958             .625144958          0
         5          1           1.22946167             1.22946167          0
         8          0           1.27175903             1.27175903          0
         4          1           1.27895355             1.27895355          0
         7          0           1.56770325             1.56770325          0
         1          0           2.02474976             2.02474976          0
         6          0            2.9671936              2.9671936          0

11 rows selected.
```

3.创建新的 UNDO 表空间

```
SQL> create undo tablespace undotbs2;
Tablespace created.
```

4.切换 UNDO 表空间为新的 UNDO 表空间

```
SQL> alter system set undo_tablespace=undotbs2 scope=both;
System altered.
```

此处使用 spfile 需要注意，需要让修改同时变更到 spfile 文件。

5.等待原 UNDO 表空间所有 UNDO SEGMENT OFFLINE

```
SQL> select usn,xacts,status,rssize/1024/1024/1024,hwmsize/1024/1024/1024,shrinks
  2 from v$rollstat order by rssize;
USN        XACTS   STATUS                RSSIZE/1024/1024/1024   HWMSIZE/1024/1024/1024
SHRINKS
---------- ---------- --------------- ---------------------- ---------------------- ----
14          0 ONLINE                .000114441             .000114441          0
19          0 ONLINE                .000114441             .000114441          0
11          0 ONLINE                .000114441             .000114441          0
12          0 ONLINE                .000114441             .000114441          0
13          0 ONLINE                .000114441             .000114441          0
20          0 ONLINE                .000114441             .000114441          0
15          1 ONLINE                .000114441             .000114441          0
16          0 ONLINE                .000114441             .000114441          0
```

```
17              0 ONLINE                .000114441              .000114441              0
18              0 ONLINE                .000114441              .000114441              0
0               0 ONLINE                .000358582              .000358582              0
6               0 PENDING OFFLINE       2.9671936               2.9671936               0

12 rows selected.
```

确认原回滚表空间所有回滚段都正常 OFFLINE:

```
11:32:11 SQL> /
USN XACTS STATUS      RSSIZE/1024/1024/1024 HWMSIZE/1024/1024/1024       SHRINKS
--- ----- -------     ---------------------- ----------------------- ----------
 15       1 ONLINE                .000114441              .000114441              0
 11       0 ONLINE                .000114441              .000114441              0
 12       0 ONLINE                .000114441              .000114441              0
 13       0 ONLINE                .000114441              .000114441              0
 14       0 ONLINE                .000114441              .000114441              0
 20       0 ONLINE                .000114441              .000114441              0
 16       0 ONLINE                .000114441              .000114441              0
 17       0 ONLINE                .000114441              .000114441              0
 18       0 ONLINE                .000114441              .000114441              0
 19       0 ONLINE                .000114441              .000114441              0
  0       0 ONLINE                .000358582              .000358582              0

11 rows selected.
```

6.删除原 UNDO 表空间

```
11:34:00 SQL> drop tablespace undotbs1 including contents;
Tablespace dropped.
```

8.检查空间情况

由于我使用的 ASM 管理,可以使用 10gR2 提供的信工具 asmcmd 来察看空间占用情况.

```
[oracle@danaly ~]$ export ORACLE_SID=+ASM
[oracle@danaly ~]$ asmcmd
ASMCMD> du
Used_MB      Mirror_used_MB
  21625               21625
ASMCMD> exit
```

此时空间已经释放。本案例包含了常规 UNDO 表空间重建、切换等过程，这也是 UNDO 表空间常规维护操作之一，需要熟记。

8.18 特殊情况的恢复

在很多情况下，特别是在使用隐含参数强制打开数据库之后，可能会在出现 Ora-00600 4194 错误，在 alert 文件中，记录主要错误日志:

```
Sat Jan 21 13:55:21 2006
Errors in file /opt/oracle/admin/conner/bdump/conner_smon_17113.trc:
ORA-00600: internal error code, arguments: [4194], [43], [46], [], [], [], [], []
Sat Jan 21 13:55:21 2006
Errors in file /opt/oracle/admin/conner/udump/conner_ora_17121.trc:
ORA-00600: internal error code, arguments: [4194], [45], [44], [], [], [], [], []
```

4194 错误通常说明 UNDO 段出现问题,最好的办法是通过备份进行恢复,如果没有备份,那么可以通过特殊的初始化参数进行强制启动,本文就 Oracle 的隐含参数进行恢复说明(由于实际情况可能各不相同，进行测试前请先行备份)，仅供参考。

首先确定当前回滚段名称，这可以从 alert 文件中获得:

```
Sat Jan 21 13:55:21 2006
Undo Segment 11 Onlined
Undo Segment 12 Onlined
Undo Segment 13 Onlined
Successfully onlined Undo Tablespace 16.
```

对应的 AUM (auto undo management) 下的回滚段名称为:

```
'_SYSSMU11$','_SYSSMU12$','_SYSSMU13$'
```

修改 init<sid>.ora 参数文件，使用 Oracle 隐含参数_corrupted_rollback_segments 将回滚段标记为损坏，此时启动数据库，Oracle 会跳过对于这些回滚段的相关操作，强制启动数据库。

```
._corrupted_rollback_segments='_SYSSMU11$','_SYSSMU12$','_SYSSMU13$'
```

使用 init<sid>.ora 参数文件启动数据库:

```
[oracle@jumper dbs]$ sqlplus "/ as sysdba"
SQL*Plus: Release 9.2.0.4.0 - Production on Sat Jan 21 13:56:47 2006
Copyright (c) 1982, 2002, Oracle Corporation.    All rights reserved.
Connected to an idle instance.
SQL> startup pfile=initconner.ora
ORACLE instance started.
Total System Global Area     97588504 bytes
Fixed Size                     451864 bytes
Variable Size                33554432 bytes
Database Buffers             62914560 bytes
Redo Buffers                   667648 bytes
Database mounted.
Database opened.
```

此时数据库正常 Open。观察 alert 文件可以获得如下信息:

```
Sat Jan 21 13:57:03 2006
SMON: enabling tx recovery
SMON: about to recover undo segment 11
SMON: mark undo segment 11 as needs recovery
SMON: about to recover undo segment 12
SMON: mark undo segment 12 as needs recovery
SMON: about to recover undo segment 13
SMON: mark undo segment 13 as needs recovery
Sat Jan 21 13:57:03 2006
Database Characterset is ZHS16GBK
Sat Jan 21 13:57:03 2006
SMON: about to recover undo segment 11
SMON: mark undo segment 11 as needs recovery
SMON: about to recover undo segment 12
SMON: mark undo segment 12 as needs recovery
SMON: about to recover undo segment 13
SMON: mark undo segment 13 as needs recovery
Sat Jan 21 13:57:04 2006
Created Undo Segment _SYSSMU1$
Undo Segment 1 Onlined
Completed: ALTER DATABASE OPEN
Sat Jan 21 14:02:11 2006
SMON: about to recover undo segment 11
SMON: mark undo segment 11 as needs recovery
SMON: about to recover undo segment 12
SMON: mark undo segment 12 as needs recovery
SMON: about to recover undo segment 13
SMON: mark undo segment 13 as needs recovery
```

此时可以重新创建新的 UNDO 表空间，删除出现问题的表空间，修改参数文件，由参数文件生成新的 spfile，重新启动数据库：

```
SQL> create undo tablespace undotbs1
  2  datafile '/opt/oracle/oradata/conner/undotbs1.dbf' size 10M;
Tablespace created.
SQL> alter system set undo_tablespace=undotbs1;
System altered.

SQL> drop tablespace undotbs2;
Tablespace dropped.
```

此时的 alert 文件记录的:

```
Sat Jan 21 14:03:29 2006
create undo tablespace undotbs1
datafile '/opt/oracle/oradata/conner/undotbs1.dbf' size 10M
Sat Jan 21 14:03:29 2006
Created Undo Segment _SYSSMU2$
Created Undo Segment _SYSSMU3$
Created Undo Segment _SYSSMU4$
Created Undo Segment _SYSSMU5$
Created Undo Segment _SYSSMU6$
Created Undo Segment _SYSSMU7$
Created Undo Segment _SYSSMU8$
Created Undo Segment _SYSSMU9$
Created Undo Segment _SYSSMU10$
Created Undo Segment _SYSSMU14$
Starting control autobackup
Control autobackup written to DISK device
        handle '/opt/oracle/product/9.2.0/dbs/c-3152029224-20060121-00'
Completed: create undo tablespace undotbs1
datafile '/opt/ora
Sat Jan 21 14:03:43 2006
Undo Segment 2 Onlined
Undo Segment 3 Onlined
Undo Segment 4 Onlined
Undo Segment 5 Onlined
Undo Segment 6 Onlined
Undo Segment 7 Onlined
Undo Segment 8 Onlined
Undo Segment 9 Onlined
Undo Segment 10 Onlined
Undo Segment 14 Onlined
Successfully onlined Undo Tablespace 1.
Undo Segment 1 Offlined
Undo Tablespace 16 successfully switched out.
Sat Jan 21 14:03:43 2006
ALTER SYSTEM SET undo_tablespace='UNDOTBS1' SCOPE=MEMORY;
Sat Jan 21 14:07:18 2006
SMON: about to recover undo segment 11
SMON: mark undo segment 11 as needs recovery
SMON: about to recover undo segment 12
```

```
SMON: mark undo segment 12 as needs recovery
SMON: about to recover undo segment 13
SMON: mark undo segment 13 as needs recovery
Sat Jan 21 14:08:06 2006
drop tablespace undotbs2
Sat Jan 21 14:08:07 2006
Starting control autobackup
Control autobackup written to DISK device
        handle '/opt/oracle/product/9.2.0/dbs/c-3152029224-20060121-01'
Completed: drop tablespace undotbs2
```

修改参数文件，变更 undo 表空间，并取消_corrupted_rollback_segments 设置:

```
*.undo_tablespace='UNDOTBS1'
```

由参数文件创建 spfile 文件。

```
SQL> create spfile from pfile;
File created.
SQL> shutdown immediate;
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL> startup
ORACLE instance started.
Total System Global Area     97588504 bytes
Fixed Size                     451864 bytes
Variable Size                33554432 bytes
Database Buffers             62914560 bytes
Redo Buffers                   667648 bytes
Database mounted.
Database opened.
```

重起数据库，观察 alert 文件:

```
Sat Jan 21 14:08:36 2006
Undo Segment 2 Onlined
Undo Segment 3 Onlined
Undo Segment 4 Onlined
Undo Segment 5 Onlined
Undo Segment 6 Onlined
Undo Segment 7 Onlined
Undo Segment 8 Onlined
Undo Segment 9 Onlined
Undo Segment 10 Onlined
```

```
Undo Segment 14 Onlined
Successfully onlined Undo Tablespace 1.
```

此时数据库可以正常启动，通过以上方法恢复数据库，通常会导致数据库内部存在不一致的状况，通常建议立即进行全库 exp，然后重新建库，再通过 imp 恢复数据库。

8.19 诊断恢复案例-回滚段损坏的恢复

在这里收录曾经遇到的另一个案例，本案例发生在某大型电信运营商的生产系统。

收到技术援助请求后，登陆该数据库主机。客户报告的情况是，CPU 忙，IO 缓慢，业务请求无法响应，客户投诉大量增加。

首先查看告警日志文件，得到如下信息:

```
bash-2.03$ cd admin/wap/bdump/
bash-2.03$ tail -f alert_WAPTL.log
SMON: mark undo segment 29 as available
SMON: about to recover undo segment 29
SMON: mark undo segment 29 as available
SMON: about to recover undo segment 29
SMON: mark undo segment 29 as available
SMON: about to recover undo segment 29
SMON: mark undo segment 29 as available
SMON: about to recover undo segment 29
SMON: mark undo segment 29 as available
SMON: about to recover undo segment 29
SMON: mark undo segment 29 as available
SMON: about to recover undo segment 29
SMON: mark undo segment 29 as available
```

这个信息高速我们，目前 29 号回滚段出现问题，SMON 不断尝试恢复该回滚段，但是始终无法恢复正常。这个恢复尝试占用了大量的 CPU 资源。继续搜索这个损坏出现的原因，在 alert 文件的前面部分，找到如下线索：

```
Thu Aug 31 19:13:23 2006
Shutting down instance (abort)
```

在 abort 关闭数据库之后，重新启动数据库即出现以上错误。

询问用户当时的情况是，数据库运行极其缓慢，用户请求得不到响应，就强行关闭了数据库，这次 Abort 关闭最终导致了回滚段损坏。在此提醒大家，使用 abort 方式关闭数据库，是具有相当风险的，具体执行时应该相当谨慎。

进一步的，我们检查数据库的当前状态，查询 v$session_wait 视图，发现数据库当前存在大量等待：

```
SQL> select sid,event from v$session_wait where event not like 'SQL%';
```

```
       SID EVENT
---------- ----------------------------------------------------------------
。。。。。。。。。。。。。。
        35 enqueue
        92 enqueue
        95 enqueue
        97 enqueue
       207 enqueue
       212 enqueue
       223 enqueue
       224 enqueue
       220 enqueue
       210 enqueue
       203 enqueue
        99 enqueue
        89 enqueue
         3 log file parallel write
         5 log buffer space
       237 log buffer space
        22 log buffer space
       238 log file sync
        14 wait for a undo record
        34 wait for a undo record
       246 wait for a undo record
        28 row cache lock
       110 row cache lock
       243 row cache lock

33 rows selected.
```

其中存在大量队列竞争，我们进而检查 v$lock 视图:

```
SQL> select * from v$lock where type<>'MR';

ADDR                   SID TY            ID1     ID2    LMODE REQUEST     CTIME          BLOCK
---------------- ----- -- ---------- ----- ------- ------- ------- ----------
.....................
00000400C72A1CE0         5 TX             116     338         6          0          0              0
00000400C70CF990         5 TM              15       0         3          0          0              0
00000400C5672A50         5 TX         1900602     120         6          0        188              1
00000400C5672720         5 PS               1       4         4          0        188              0
```

```
00000400C5672698       5 PS            1       3       4       0       188             0
00000400C5672610       5 PS            1       2       4       0       188             0
00000400C5670328       5 TS            2       1       3       0      1444             0
00000400C56702A0       5 PS            1       1       4       0       188             0
00000400C5670190       5 PS            1       0       4       0       188             0
00000400C5670108       5 TS           18       1       3       0       365             0
.....................
00000400C74B31A0      35 TX      3342351      26       6       0       189             0
00000400C70CEF10      35 TM       111452       0       3       0       188             0
00000400C70CEE50      35 TM       111449       0       3       0       188             0
00000400C5672AF0      35 TX      1900602     120       0       4       188             0
00000400C70CF5D0      89 TM       112649       0       3       0        41             0
00000400C5673040      89 TX      1900602     120       0       6        41             0
00000400C70CF210      92 TM       112649       0       3       0        74             0
00000400C5672D98      92 TX      1900602     120       0       6        74             0
00000400C70CF750      95 TM       112649       0       3       0        29             0
00000400C5673150      95 TX      1900602     120       0       6        29             0
00000400C70CF8D0      97 TM       112649       0       3       0        20             0
00000400C5673260      97 TX      1900602     120       0       6        20             0
00000400C70CF390      99 TM       112649       0       3       0        52             0
00000400C5672EA8      99 TX      1900602     120       0       6        52             0
00000400C70CF690     203 TM       112649       0       3       0        30             0
00000400C56730C8     203 TX      1900602     120       0       6        30             0
00000400C70CF2D0     207 TM       112649       0       3       0        59             0
00000400C5672E20     207 TX      1900602     120       0       6        59             0
00000400C70CF450     210 TM       112649       0       3       0        50             0
00000400C5672F30     210 TX      1900602     120       0       6        50             0
00000400C70CF510     212 TM       112649       0       3       0        42             0
00000400C5672FB8     212 TX      1900602     120       0       6        42             0
00000400C70CF090     220 TM       112649       0       3       0       128             0
00000400C5672C00     220 TX      1900602     120       0       6       128             0
00000400C70CF810     223 TM       112649       0       3       0        20             0
00000400C56731D8     223 TX      1900602     120       0       6        20             0
00000400C70CEFD0     224 TM       112649       0       3       0       120             0
00000400C5672C88     224 TX      1900602     120       0       6       120             0

48 rows selected.
```

从以上输出中注意到，大量 Session 的请求都被阻塞，而阻塞这些 Session 的进程正是 SMON 进程（sid=5）。

那么问题已经基本清晰，由于 abort 关闭数据库，导致回滚段损坏，回滚段不断尝试修复回滚段，回滚事务，这一事务有导致其他事务的锁等待，从而整个数据库异常缓慢，无法响应。

这种情况下，最好的方式是通过备份进行恢复，使整个数据库恢复正常。但是通过备份进行恢复可能存在的问题有：

恢复时间可能很长，业务影响过大。

如果执行不完全恢复，则可能导致部分数据库丢失。

….

由于用户环境不允许进行停机长时间的恢复，所以只能作为特殊情况进行处理。在初始化参数文件中设置隐含参数：

```
_offline_rollback_segments=   _SYSSMU29$
_corrupted_rollback_segments= _SYSSMU29$
```

将 29 号回滚段标记为损坏并脱机，然后重新启动数据库，此时观察 alert 文件中的错误信息：

```
Fri Sep   1 07:56:18 2006
SMON: about to recover undo segment 29
SMON: mark undo segment 29 as needs recovery
Fri Sep   1 08:01:26 2006
SMON: about to recover undo segment 29
SMON: mark undo segment 29 as needs recovery
Fri Sep   1 08:06:33 2006
SMON: about to recover undo segment 29
SMON: mark undo segment 29 as needs recovery
```

此时数据库仍然尝试去恢复 29 号回滚段，由于已经强制将该回滚段 Offline，此时可以将损坏的回滚段删除：

```
SQL> drop rollback segment "_SYSSMU29$";
        Rollback segment dropped.
```

此时继续观察 alert 文件，之前的错误不再出现

```
Fri Sep   1 10:27:11 2006
drop rollback segment "_SYSSMU29$"
Fri Sep   1 10:27:11 2006
Completed: drop rollback segment "_SYSSMU29$"
```

至此数据库基本恢复了正常运行，但是需要注意的是，由于我们强制删除了一个回滚段，那么一定会损失部分事务，导致数据库不一致。

如果损失的事务是用户事务，那么通过检查和修正数据错误就可以解决；如果损失的事务是系统事务，那么可能会出现 ORA-600 错误，就需要 DBA 介入进行进一步的故障处理和问题解决。

参考文档：

Performance and Scalability Improvements in Oracle 10g and 11g　　By Tanel Poder