回滚段探究

● 前言

对于 oracle 的回滚段的分配与管理，实际上不那么复杂，当然如果我们从原理甚至从 oracle internal 的角度深究相关问题的话有不是一件容易的事。但对于我们普通开发人员和 DBA 来说，重要的是在概念上清晰，原理上了解，使用上熟悉。要做到这一点，其实并不是一件困难的事情。问题的根源在于，很少有中文文章就回滚段进行清晰的介绍和整理，而对于 oracle document，能沉下心来慢慢读的人实在太少，虽然我不得不承认如果您认真的读了 oracle concepts 中关于回滚段部分，这里的大部分内容对于您来说已经是很熟悉的东西，但是，我还是愿意抛开 oracle document 来谈我的理解，在现实中我们到底遇见怎样的问题。在琐碎的文档中，抓不住问题的关键部分是我们阅读的最大的障碍。又由于过于琐碎使得难以在宏观上进行把握，从而打击了我们坚持读下去的信心。所以也才让我有产生写这点文字的念头，其中一部分是重复了 oracle 文档中的内容，只不过可能来源于文档的不同地方。在这文章里我打算从回滚段的作用、原理、分配与管理、诊断、常见问题等几个方面来阐述相关内容。

● 什么是回滚段

如果想按照数学上的定义来定义回滚段，我想是乏味的、抽象的。比如可以说是用来保存数据变化前映像而提供一致读和保障事务完整性的一段磁盘存储区域。做应用的人，往往并不是关心概念或者定义而是关心具体的使用，就好比我们天天都在管理使用数据库但却根本记不住数据库的定义一样。事实上我们的工作也不要求我们记得数据库定义。重要的是，对于我们的数据库，我们该怎么具体去理解它、怎么去使用和管理它。

我们大概知道回滚段是在磁盘上的一段空间，当一个事务开始的时候，会首先把变化前的数据和变化后的数据先写入日志缓冲区，然后把变化前的数据写入回滚段，最后才在数据缓冲区中修改数据（日志缓冲区内容在满足一定的条件后可能被写入磁盘，但在事务提交的时候日志必须写入磁盘，而数据缓冲区中数据依赖于检查点的发生和 DBWR 进程的活动，在这里不继续讨论相关内容）。举个例子，当我们做如下操作

```
SQL>    update t set object_id = '0' where object_id = '12344';

1 row updated.

SQL> commit;

Commit complete.
```

这时数据库首先把该语句的整个操作包括数据’0’和’12344’写入日志缓冲区，然后把数据’12344’和一些相关信息写入回滚段，最后把’0’修改到数据缓冲区。当发出提交命令的时候，如果日志缓冲区内容还没有写入日志文件则必须写进日志文件，回滚段把该事务标记为已经提交，数据缓冲区中块的这些事务也标记为已经提交（在大事务的情况下如果数据缓冲区中块已经被写入磁盘或者该事务更改的块超过数据缓冲区总大小的 10%则不再对这些块

标志该事务为已经提交，这会影响到我们下次读该块)。当然如果我们回退这个事务，则数据库将把回滚段中数据’12344’读出来写回数据缓冲区（数据缓冲区已经被修改为’0’），这个回退的变化本身也被写入日志，这就是回退的过程。我们可以看出如果事务很大、数据缓冲区中事务数据如果已经被写入磁盘了，那这简直就是一个代价极其昂贵的操作。所以通常来说，我们认为一个系统中的事务的回退率应该比较低，否则应该检查系统是否正常或者程序设计思路是否存在问题。我们可以通过下面查询来衡量数据库启动以来的事务回退率，如果发现 transaction rollbacks/(transaction rollbacks + user commits)太高一定要引起重视。

```
SQL>    select name,value from v$sysstat where name in ('user commits','transaction rollbacks');

NAME                                                             VALUE
---------------------------------------------------------------- ----------
user commits                                                      12532
transaction rollbacks                                            21

SQL>
```

在这里我们要指出的关于回滚段存储的数据，假如是 delete 操作，则回滚段将会记录整个行的数据，假如是 update,则回滚段只记录被修改了的字段的变化前的数据（前映像），也就是没有被修改的字段是不会被记录的，假如是 insert，则回滚段只记录插入记录的 rowid。这样假如事务提交，那回滚段中简单标记该事务已经提交；假如是回退，则如果操作是是 delete,回退的时候把回滚段中数据重新写回数据块，操作如果是 update，则把变化前数据修改回去，操作如果是 insert，则根据记录的 rowid 把该记录删除。这个过程，可以看作保障事务的完整性，保障数据不丢失。

这里我们介绍一下一致性读取，英文是 consistent reads。对于 oracle 而言，查询的结果集是根据时间点来判定的。Oracle 内部通过系统改变号 SCN 作为相对时间点的标准，任何对数据库的改变都会产生 SCN,这个一个正整数，对数据块的数据改变的时候会把该改变所对应的 SCN 记录在块中。假设查询开始的时候的 SCN 为 T，则在查询所扫描的数据块中，如果数据的 COMMIT SCN 小于 T，则查询接受该数据，如果 COMMIT SCN 大于 T 或者说还没有产生 COMMIT SCN，则查询会尝试去回滚段中查找数据。这是为了保证数据的读取的时间点的一致性，所以叫一致性读。

在通过回滚段中获取数据的时候，本质上是把数据缓冲区中数据块做一份拷贝，然后将回滚段中记录的内容恢复到该块中，然后查询使用这个块来进行读取。这样的块的数量，如果大家有兴趣可以通过 SYS 用户登陆到数据库然后查询获得

```
SQL>    select count(*) from x$bh where state = 3;

  COUNT(*)
----------
        15

SQL>
```

我们很可能产生这样的困惑，回滚段的块是否存在于 SGA 中？如果有存在那为何不见有参数设置？真实情况是，回滚段的块跟其他数据文件的块一样，都在 SGA 中，也都是数据缓冲区中的一份子。同样地，对于 oracle8 以上的版本的数据库，我们可以通过查询 x$bh 来确定回滚段所占的块的多少。我们可以参考 v$rollstat 事视图和 x$bh 表，在 x$bh 中的 class 字段，如果是回滚段块，假设回滚段 USN 为 n,则回滚段头 class 为 11+2n，回滚段块为 12+2n。我们来看看实际的例子

```
SQL> select usn from v$rollstat;

       USN
----------
         0
         1
         2
         3
         4
         5
         6
         7
         9

9 rows selected.

SQL>
```

从回滚段编号中我们可以看到编号 8 的回滚段已经被删除了，总共 9 个回滚段。

```
SQL>    select class,count(*) from x$bh where class > 10 group by class;

     CLASS   COUNT(*)
---------- ----------
        11          1
        12          2
        13          1
        14          1
        15          1
        16          1
        17          1
        18        982
        19          1
        20          1
        21          1

     CLASS   COUNT(*)
---------- ----------
        22          1
```

```
        23          1
        24          1
        25          1
        26          1
        29          1
        30          1

18 rows selected.

SQL>
```

从这里我们可以看出，回滚段编号为 8 的 class 应该为 27、28 的块是没有。回滚段头始终都是一个块，class=18(usn=3)的回滚段使用的块比较多，是因为我的数据库刚启动，刚才执行了下面的语句。

```
SQL> delete from t;

25374 rows deleted.

SQL> commit;

Commit complete.

SQL>
```

● 回滚段的分配和使用

当事务产生的时候，数据库会给事务分配一个回滚段。当然我们可以指定事务使用某个回滚段。在我的测试用的数据库中，有如下回滚段

```
SQL> select SEGMENT_ID ,SEGMENT_NAME    from dba_rollback_segs;

SEGMENT_ID SEGMENT_NAME
---------- ------------------------------
         0 SYSTEM
         1 RBS0
         2 RBS1
         3 RBS2
         4 RBS3
         5 RBS4
         6 RBS5
         7 RBS6
         9 RBS12

9 rows selected.
```

SQL>

如果我们要指定事务使用某个回滚段，如下

```
SQL>    set transaction use rollback segment rbs6;

Transaction set.

SQL>    insert into t select * from all_objects;

25649 rows created.

SQL> commit;

Commit complete.

SQL>
```

如果我们不人为的指定使用哪个回滚段，则数据库会根据回滚段中事务来权衡，以使得所有回滚段中事务压力尽可能平均。我们考虑存在非系统回滚段的情况（这也是绝大多数系统的情况，除非因为 DBA 严重错误才会使得系统只存在系统回滚段），在这种情况下我们发出的事务不会去使用系统回滚段，这时系统回滚段只用于系统级使用，比如 create、drop、truncate 等发生的时候的系统级的数据字典的回滚记录。数据库为我们发出的事务选择非系统回滚段，同一个事务不能跨越回滚段，也就是说即使其他回滚段还很空闲，该大事务也只能使用被分配的回滚段即使该回滚段扩展。

接下来我们来考究单个回滚段内的使用、扩展、回缩的问题。一个回滚段至少包含 2 个 extent。每个回滚段有一个回滚段头，回滚段头是一个 block，里面主要记录了事务表信息。当产生一个事务的时候，就在回滚段头的事务表中记录一条信息，该信息中包含了事务标志、事务状态、使用的回滚段块数等等信息。我们假定新创建的一个回滚段存在 extent 1，2，3，4，5。当第一个事务分配到回滚段中的时候，除了事务表信息，该事务从 extent 1 的第二个 block 开始使用，该事务提交后陆续不断的事务到来并提交，假设我们已经使用到了 extent 5 的末尾。这时再来事务内容需要写入，则重新使用 extent 1 的第二个 block。这样循环使用回滚段空间。同理回滚段头的事务表也是如此循环使用。Extent 的使用从一个跨越到另一个的次数，可以通过如下查询获得

```
SQL>    select usn,WRAPS from v$rollstat;

       USN      WRAPS
---------- ----------
         0          0
         1         15
         2         15
         3         15
         4         15
         5         12
         6         15
```

```
         7         17
         9         12

9 rows selected.

SQL>
```

那么我们会有一个问题，那就是既然回滚段是循环使用的，那为什么会扩展呢？注意在上面的例子中，所有的事务都是迅速提交了的。那我们现在假定这样一种情况，存在一个事务，假设在 extent 3 中有一个事务一直没有提交，然后回滚段一直循环使用到了 extent 2。当 extent 2 使用完毕的时候发现 extent 3 中存在未提交事务，这是即使 extent 4,5,1 中的事务都已经提交，当前事务也不能越过 extent 3 而去使用后面的可使用的 extent，这时该回滚段就扩展新的 extent，假定为 extent 2-1。在数据库中实际上回滚段的 extent 之间是通过指针连起来的一个单向循环的链表结构。扩展的时候相当于在链表中插入一个节点 extent 2-1，但节点 2-1 的下一个 extent 依然是 extent 3，假如 2-1 使用完毕发现 extent 3 仍然存在未提交事务回滚段会继续扩展。

我们做个例子，在 **SQLPLUS 1** 中运行不提交(我们在 v$rollstat 和 dba_rollback_segs 中分别查询发现 usn =5 的回滚段对应了名字 rbs4)

```
SQL> select a.usn,b.segment_name from v$rollstat a,dba_rollback_segs b
  2    where a.usn = b.segment_id;

       USN SEGMENT_NAME
---------- ------------------------------
         0 SYSTEM
         1 RBS0
         2 RBS1
         3 RBS2
         4 RBS3
         5 RBS4
         6 RBS5
         7 RBS6
         9 RBS12

9 rows selected.

SQL>
SQL> select usn,rssize "rollback segment size" from v$rollstat where usn = 5;

USN         rollback segment size
---------- ----------------------
5               4186112

SQL>    set transaction use rollback segment rbs4;
```

Transaction set.

SQL> update t_small set object_id = 1;

100 rows updated.

打开 SQLPLUS 2 运行

```
begin
for i in 1..1000 loop
set transaction use rollback segment rbs4;
update t set object_id = i where rownum < 101;
commit;
end loop;
end;
SQL>  select usn,rssize "rollback segment size" from v$rollstat where usn = 5;

USN          rollback segment size
----------       --------------------
5                55042048
```

现在我们来看在一个独立的 session 中运行下面的查询并对比查询前后的回滚段变化。

```
SQL> select   usn,rssize "rollback segment size" from v$rollstat where usn= 4;

USN         rollback segment size
---------- ---------------------
4           4186112

SQL> begin
  2  for i in 1..1000 loop
  3  set transaction use rollback segment rbs3;
  4  update t set object_id = i where rownum < 101;
  5  commit;
  6  end loop;
  7  end;
  8  /

PL/SQL procedure successfully completed.

SQL>  select   usn,rssize "rollback segment size" from v$rollstat where usn= 4;

USN         rollback segment size
---------- ---------------------
```

4 **4186112**

SQL>

在这两个例子中我们很明显地看到了回滚段是否扩展的对比。那么，做完第二个例子后，我再回过头来查询原来扩展的比较大的回滚段,发现又变成 4M 了

SQL> select usn,rssize "rollback segment size" from v$rollstat where usn= 5;

USN rollback segment size
---------- ----------
5 **4186112**

SQL>

回滚段在扩展后，是要回缩的。假设回滚段当前 extent n,使用完毕将准备使用 extent n+1 的时候(extent n+1 无活动事务)，如果设置了 optimal 并且回滚段大于 optimal 设置的大小，检查 extent n+2 中是否有未提交事务，如果没有，则回收 extent n+2，本质上，就是在回滚段的链表上摘去 extent n+2 ，然后继续检查 extent n+3 决定是否回收,由此重复该动作。Optimal 设置决定了回滚段最终回收后的大小，回滚段回缩后大小尽可能的接近 optimal 设置，如上面例子是 4M，可通过下面查询(shrinks 表示回滚段回缩的次数，实际上 v$rollstat 提供了很多的回滚段信息，大家可以参考 oracle document)

SQL> select USN,OPTSIZE,SHRINKS from v$rollstat;

USN	OPTSIZE	SHRINKS
0		0
1	4194304	0
2	4194304	0
3	4194304	0
4	4194304	0
5	4194304	10
6	4194304	0
7	4194304	0
9		0

9 rows selected.

● 系统回滚段与延迟回滚段

SYSTEM 回滚段是创建在系统表空间中，主要是用于系统级的事务和分配普通事务于其他回滚段上。当手工创建数据库后需要创建普通回滚段之前必须首先创建系统回滚段。按照 oracle 文档说明，当普通事务异常多的事情可能会出现使用系统回滚段的情况。但正常情况下，系统回滚段主要用于两个方面。一是系统事务，比如针对数据字典的操作的 truncate table 和 drop table 。如果 truncate table or drop table 的过程中没有成功，则系统会根据系

统回滚段中的数据字典操作信息对该 DDL 操作进行回退。另一个方面，就是延迟回滚段(Deferred Rollback Segment)。延迟回滚段表示的是，当我们使一个表空间 OFFLINE(exeample: alter tablespace users offline)之后，由于表空间不可用（不能进行读写），这个时候若有事务数据位于该表空间并且执行了回滚命令，回滚完成将显示给 client，对于 client 看起来该事务已经回滚，但是对于数据库来说该回滚并没有真正完成，这个时候数据库将该回滚信息写入系统回滚段（这就是延迟回滚段），等表空间重新 ONLINE 的时候，数据库从系统回滚段中将回滚信息写入表空间。

● 回滚段的设置和管理

事实上，对于作为个人意见来说，回滚段的管理本不应该是作为 DBA 的复杂的任务来对待的，因为回滚段的管理本来就可以使其很简单。几乎所有的系统出现回滚段问题，不外乎都是回滚段大小不足、回滚段个数太少。9i 以前的版本因为对于我们来说，无非也就是这两个问题。

关于回滚段表空间大小、回滚段数据文件的扩展、回滚段的扩展等创建时指定的参数问题，我想参考创建语法就足够了，没有必要在这上去纠缠 max extents 是 100 还是 200，你会发现去考虑这些参数没有实质上的意义了。所有的一切设置，我只需要问几个问题就足够了：

1：系统并发事务数有多少？

2：系统是否存在大查询或者大是事务？频繁么？

3：能提供给系统的回滚段表空间的磁盘空间是多少？

在初始化参数文件中存在参数 transactions_per_rollback_segment 和 transactions，共同决定了实例启动的时候将尝试联机的最大回滚段个数，transactions 决定了同时存在的最大事务数。在这里顺便提及的 2 个初始化参数

max_rollback_segments　系统允许的最大回滚段个数

rollback_segments　该参数是一个参数列表，假如创建回滚段的时候是 PUBLIC 类型，则跟该参数无关，假如是 PRIVATE 类型的，则必须使得回滚段出现在该参数列表里面，否则数据库启动之后这些回滚段不会自动联机(可手动)。在 OPS/RAC 中，PUBLIC 类型的回滚段表示所有的 INSTANCE 都可以联机这些回滚段（但同一时刻只能有一个实例联机），相当于是一个公共回滚段池。

在实例启动的时候，实例尝试联机 rollback_segments 中设置的回滚段，直到达到个数为 min(CEIL(transactions/transactions_per_rollback_segment), max_rollback_segments)。若没有达到这个值，则实例尝试在 PUBLIC 类型的回滚段池中尝试联机回滚段，直到达到该值或者不再有回滚段可联机。

当一个实例启动后其联机回滚段的个数是否足够，跟并发事务数有关，若每个回滚段上活动事务过多可能导致严重的回滚段争用。

这里说了问题一相关内容，我们再看后面两个问题。由于回滚段的扩展和回收是昂贵代价的操作，通常我们是要避免的。如果存在大的查询，就算不会去写回滚段，但是由于一致读，我们也可以参照前面内容，知道如果这期间事务繁忙回滚段被循环使用覆盖过，可能出现著名的 ORA-01555 错误。又由于事务产生的时候除非人为指定使用哪个回滚段，否则事务使用哪个回滚段对于我们应用来说是透明的，同时我们能指定事务使用哪个回滚段但并不能阻止别的事务不使用某个回滚段，这样我们就必须认识到，回滚段设置成大小不一致是不合适的，几乎是没有意义的，因为瓶颈总是决定于最小的一个回滚段(这类似于木桶原理，

决定装水量的多少是由最短的片所决定的)。所以我们应该统一回滚段的大小。那通常对于一个系统来说，几百 M 的磁盘空间甚至几 G 的磁盘空间根本不是问题，所以我们没有理由在这里研究回滚段到底是使用 4M 大小还是 10M 大小，我们根据能提供的磁盘空间的估计，完全可以设置回滚段为 50M/100M 甚至更大的大小，这主要决定于在大查询运行期间每个回滚段上可能的事务生成量，以及单个事务可能产生的回滚数据的大小。假如系统偶尔存在批量作业的时候可能使得某个回滚段扩展到 1G，但平常我们的回滚段大小在 50M 就不会出现回缩现象。那这个特定的时候如果数据库不繁忙只有大作业我们可以创建几个很大的回滚段，然后是其他回滚段 offline,等批作业完成然后再 online 其他回滚段，使大回滚段 offline。当然可能的话也可以指定批作业使用大的回滚段。或者，我们可以为所有回滚段设置 optimal 为 50M，任其特定时刻扩展然后回缩（注意所有回滚段的 optimal 必须设置一样大小）。

对于回滚段除了按照我们对系统状况估计进行创建、删除外，还有使回滚段联机和脱机，我们要注意的是如果回滚段处于联机并且里面有活动事务的时候，若想使回滚段脱机(offline)，则这时回滚段处于一种悬置的状态，也就是新的事务将不能使用该回滚段，而原有的事务继续存在，等待回滚段中所有事务完毕后，回滚段成为脱机状态。

● 9i 的 UNDO TABLESPACE

从 oracle9i 开始，推荐使用 UNDO TABLESPACE，让系统自动管理回滚段。

```
SQL> show parameters undo

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------
undo_management                      string      AUTO
undo_retention                       integer     10800
undo_suppress_errors                 Boolean     FALSE
undo_tablespace                      string      UNDOTBS1
SQL>
```

初始化参数 undo_management 决定数据库使用的回滚段是否使用自动管理模式。AUTO 表示自动管理，MANUAL 表示手工管理，也就是跟 8i 中一样。undo_tablespace 指定在自动管理模式下 INSTANCE 使用哪个表空间，undo_retention 表示在自动管理模式下，回滚段中的数据在被覆盖前保留多长的时间，单位是秒。这个参数应该决定于系统所中一些大查询运行的时间长度，以避免 ORA-01555 错误。当然，实际上由于 9i 提供的 flashback 功能，可根据需要决定该参数设置的时间长度。设置 undo_retention 的同时要估计这样长的时间内系统所产生的回滚数据的大小，结合硬件所提供的磁盘空间来综合考虑。undo_suppress_errors 参数如果设置为 true 表示在自动管理模式下，如果我们尝试在创建回滚段，则返回错误信息，设置为 false 表示尝试创建回滚段不会返回错误信息，但实际上是无效的。

回滚段成为了自动管理，一方面验证了前面所说的回滚段的管理的问题本不是一个复杂的问题，另一方面，自动管理让我们觉得无所适从，几乎人为的难以控制它了。比如 UNDO 表空间变的很大，我们却不能缩小。这个时候我们可以考虑创建新的 UNDO 表空间，然后换到新的表空间，这时即使 UNDO 表空间中有事务也可以切换，只不过不能立即删除该表

空间，切换之后等到原来的表空间中的所有事务处理完毕并且达到 undo_retention 所限定时间之后，就可以 drop 原来的 UNDO 表空间。这样可以解决 UNDO 表空间变的太大而无法缩小的问题。命令如下：

```
SQL> alter system set undo_tablespace = undotbs1;

System altered.

SQL>
```

这里要注意若切换了 UNDO 表空间后应该修改 pfile 或者 spfile 使得下次启动应用新的 UNDO 表空间。

在自动管理模式下的 UNDO 表空间中的回滚段的个数是变化的，里面回滚段依然存在着联机和脱机的状况，只不过是系统自己管理。若在脱机的时候回滚段中还存在着活动事务，这时也出现前面所讲的一种悬置的状态。这个时候系统可能会为此而生成提示信息的 trace 文件，这是没有关系的，对系统没有影响。

在 9i 下创建非自动管理的回滚段而不使用 UNDO 表空间，则设置 undo_management 为 MANUAL，然后在系统表空间中创建一个回滚段(注意这是必须的)，创建自己的回滚段表空间，这时可以在回滚段表空间中创建回滚段，创建完毕删除系统表空间中的回滚段。

顺便提及几点，UNDO 表空间在做含有 LOB 类型数据的 EXP 的时候如果数据量过大（也许是超过 5G）可能出现 BUG，ORACLE9.2.中如果 UNDO 是 ASSM，若设置 undo_retention 不当等因素导致扩展过大，也会出现 BUG，导致系统崩溃只能进行介质恢复。当然，建议管理生产数据库的 DBA 们有空多浏览 metalink，了解各种版本的特性和 BUG。

● 回滚段著名的 ORA-01555 问题

关于一致性读前面已经有介绍，但是这里我们不得不产生一个疑问，一致读获取的时候发现回滚段已经被覆盖而出现找不着变化前映像，也就是当 COMMIT SCN 大于 T 查询尝试去回滚段中找回数据却发现回滚段已经被覆盖了（因为回滚段是循环使用的），则会出现著名的 ORA-01555 错误。

另外一种情况是关于块清除(block cleanout)，这涉及到 oracle 的一个块清除(block cleanout)的概念，也就是说已经提交的数据，需要标志为已经提交，从而使得后面的会话访问该数据的时候不再产生一致读而直接读该块。而如果事务提交的时候块已经被写入磁盘，则当时不会对块进行清除，需要延迟清除(delay block cleanout)。当被提交但没有来得及标志为提交的块在下次被会话读取的时候会话会检查该块上最新的事务状态是否是活动的，如果已经不是活动的则修改事务标志。这样做一次后就不再产生不必要的一致读。但这种情况仅仅出现在：事务很大可能产生这种情况，事务早已提交，回滚段已经被覆盖，块中没有被标记是否提交，而当前查询 SCN T 也比目前回滚段中记录的最小的 SCN 小（查询已经运行较长时间回滚段都已经被覆盖）。这个时候数据库不能判定当前查询的 SCN T 与该块的 COMMIT SCN 之间的大小关系，于是返回了错误。事实上后面这种情况是罕见的，而理解起来也比较困难一些。通常我们可以不考虑这种情况。

当然上面是从数据库角度来描述，从应用角度来确诊这个问题，有几种可能：

1：查询执行时间太长。那么在这个时候我们首先要做的工作就是优化查询，然后是考虑能否把查询放在数据库不繁忙的时候运行，最后才是考虑加大回滚段。

2：过度频繁的提交。假如可以成批提交的事务，我们可能是单条提交了，应该考虑对整个处理一起提交，或者说分段提交而不是单条提交。

3：exp 的时候使用了 consistent = y。这个参数主要是为了保证在 exp 的时候使得所有导出来的表在时间点上具有一致性，可避免存在主外键关系的表由于不同表时间点的不一致而破坏了数据的完整性。建议该操作在系统空闲的时候进行。

4：由于回滚段回缩导致回滚段还没有循环使用的情况下就出现了在回滚段中找不着数据的情况，那这只能是加大回滚段增大 optimal 设置。

其他诸如回滚段不能扩展等等一目了然的原因，不再累述。

● 回滚段的监控和一些有用 script

对于回滚段的监控，其实最通常的是查看 v$rollstat 动态视图

```
SQL>    desc v$rollstat
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------------
 USN                                                NUMBER
 EXTENTS                                            NUMBER
 RSSIZE                                             NUMBER
 WRITES                                             NUMBER
 XACTS                                              NUMBER
 GETS                                               NUMBER
 WAITS                                              NUMBER
 OPTSIZE                                            NUMBER
 HWMSIZE                                            NUMBER
 SHRINKS                                            NUMBER
 WRAPS                                              NUMBER
 EXTENDS                                            NUMBER
 AVESHRINK                                          NUMBER
 AVEACTIVE                                          NUMBER
 STATUS                                             VARCHAR2(15)
 CUREXT                                             NUMBER
 CURBLK                                             NUMBER

SQL>
```

在这个view中如果发现SHRINKS很大可以认为回滚段设置太小或者optimal设置太小，如果是waits太大可以认为是回滚段的个数显得少。我认为经常根据关键字搜索 http://tahiti.oracle.com是一个很好的习惯，这里有几乎所有的基本概念的问题，大量的oracle documents，这些view的信息都可以在这里查询，所有关于oracle的基本问题在这里几乎都能找着答案。

另外一个帮助我们诊断问题的 view

```
SQL> select * from v$waitstat;
```

```
CLASS                          COUNT          TIME
------------------             ----------     ----------
data block                      341            0
sort block                       0             0
save undo block                  0             0
segment header                   0             0
save undo header                 0             0
free list                        0             0
extent map                       0             0
bitmap block                     0             0
bitmap index block               0             0
unused                           0             0
system undo header               0             0
system undo block                0             0
undo header                      4             0
undo block                      81             0

14 rows selected.

SQL>
```

从这个 view 中可以看出回滚段头和回滚段块的争用，如果严重也可以考虑增加回滚段个数和增大回滚段大小

现在给出一些有用的 script

下面这个 script 是对于系统中存在着锁等待的时候查询到底是什么 session 阻塞了别人。**(注意在 oracle9.2.0 版本中提供了 dba_blockers 表显示锁住其他 session 的 session 的信息，而 dba_waiters 则显示了被锁住的 session 的信息)**

```
SQL>  select username,
  2   v$lock.sid,
  3   trunc(id1/power(2,16)) rbs,
  4   bitand(id1,to_number('ffff','xxxx'))+0 slot,
  5   id2 seq,
  6   lmode,
  7   request
  8   from v$lock,v$session
  9   where v$lock.type='TX'
 10   and v$lock.sid = v$session.sid
 11   and v$session.username = user;

USERNAME                                SID        RBS       SLOT
SEQ       LMODE     REQUEST
------------------------------ ---------- ---------- ---------- ---------- ---------- ----------
RAINY                                    8          7         45        300
0          6
```

```
RAINY                                  13          7          45          300
6           0
```

从这里我们可以看到是 LMODE=6 的阻塞了 LMODE=0 的 session（LMODE=6 表示 session 拥有锁，锁的类型是 6，表示独占）。

在$ORACLE_HOME\RDBMS\ADMIN 目录下有一个名为 utllockt.sql 的 script，提供了详细的说明，该 script 输出一个很直观的数据格式。

另外再介绍一个怎样查询数据库当前某个 session 的事务所使用的回滚段大小

```
SQL> select b.sid,
  2  a.XIDUSN ,
  3  a.USED_UBLK
  4  from v$transaction a,v$session b
  5  where a.addr = b.TADDR;

       SID        XIDUSN    USED_UBLK
----------          ----------     ----------
         8              1              3

SQL>
```

ORACLE9.2.0 版本提供的表内容如下, HOLDING_SESSION 与 WAITING_SESSION 均表示 SID，可根据 v$session 结合查询信息

```
SQL>   desc dba_blockers
 Name                           Null?      Type
 ------------------------ -------- ----------------
 HOLDING_SESSION                        NUMBER

SQL>   desc dba_waiters
 Name                           Null?      Type
 ------------------------ -------- ----------------
 WAITING_SESSION                        NUMBER
 HOLDING_SESSION                        NUMBER
 LOCK_TYPE                              VARCHAR2(26)
 MODE_HELD                              VARCHAR2(40)
 MODE_REQUESTED                         VARCHAR2(40)
 LOCK_ID1                               NUMBER
 LOCK_ID2                               NUMBER

SQL>
```

当然，事实上，还可以结合 v$session 中 SQL_ADDRESS、SQL_HASH_VALUE 联合 v$sqlarea，从而知道 session 正在运行什么 sql，通过 v$lock、dba_objects、v$locked_object 查询有什么锁，锁住了什么对象。只要你了解了 oracle 这些 view 的内容，所谓这些查询就变成跟普通应用中查询的需求一样，解决问题不过是信手拈来的事情。

- **回滚段表空间中的一个数据文件丢失或者损坏的恢复方法的总结**

回滚段表空间中的一个数据文件丢失或者损坏导致数据库无法识别它，在启动数据库的时候会出现 ORA-1157，ORA-1110 的错误，或者操作系统级别的错误，例如 ORA-7360。在关闭数据库的时候(normal 或者 immediate)会出现 ORA-1116，ORA-1110 的错误，或者操作系统级别的错误，例如 ORA-7368。

当然，这种时候也会有很多种不同的情况，下面就一一讲述一下在各种情况下一般的恢复方法。

一。数据库此时处于关闭的状态

这种时候又可以分成两种情况：

A。数据库是正常的关闭的(normal 或者 immediate)

这种情况下最简单的方法就是 offline drop 掉这个坏了的或者丢失的数据文件，然后以 restricted 模式打开数据库然后删除并且重建包含损坏文件的回滚段表空间。

具体步骤如下：

1. 确定数据库是正常的关闭的。方法是可以去查看 alert 文件，到最后看是否有如下信息：
   "alter database dismount
   Completed: alter database dismount"
   如果有的话，就证明数据库是正常关闭的，如果你最后关闭数据库是用的 abort 模式或者是数据库自己 crash 了，那就不能使用这个方法进行恢复了。
2. 修改 init 参数文件，移去 ROLLBACK_SEGMENTS 中包含的损坏数据文件的回滚段表空间的回滚段，如果你不能确定哪些回滚段是坏的，简单的方法是你可以注释掉整个 ROLLBACK_SEGMENTS。
3. 以 restricted 模式去 mount 数据库。
   STARTUP RESTRICT MOUNT
4. offline drop 掉那个坏的数据文件
   ALTER DATABASE DATAFILE '<full_path_file_name>' OFFLINE DROP；
5. 打开数据库
   ALTER DATABASE OPEN ；
   如果你看到如下信息"Statement processed"，则跳到第 7 步，如果你看到 ORA-604，ORA-376，and ORA-1110 的错误信息，继续第 6 步。
6. 正常的关闭数据库，然后在 init 文件中注释掉 ROLLBACK_SEGMENTS，并加入隐含参数：
   _corrupted_rollback_segments = (<rollback1>,...., <rollbackN>)
   注：这个参数是 Oracle 的隐含参数，一定要慎用，最好在 Oracle Support 的指导下做，因为如果未经 Oracle 公司同意使用这个参数后 Oracle 公司对这个数据库将不再支持了。
   然后以 restricted 模式打开数据库
   STARTUP RESTRICT
7. 删除掉那个包含损坏文件的回滚段表空间。
   DROP TABLESPACE <tablespace_name> INCLUDING CONTENTS；

8．重建回滚段表空间，记得创建后要把回滚段都 online。
9．重新使数据库对所有用户可用。
ALTER SYSTEM DISABLE RESTRICTED SESSION；
10．然后正常关闭数据库，修改 init 文件，如果开始只是注释掉了 ROLLBACK_SEGMENTS 的，就去掉注释即可，如果加了隐含参数的，注释掉它，并在 ROLLBACK_SEGMENTS 加入所有的回滚段。
11．重建数据库。

B．数据库不是正常关闭的(abort 或者突然掉电)

这种情况下数据库最后是 shutdown abort 或者 crash 了，一般这种情况下回滚段中都包含有激活的事务，因此损坏的数据文件是不能被 offline drop 掉的，此时就需要从一个有效的备份中恢复并执行介质恢复。如果数据库是非归档方式下的话，那只能在 redolog 没有被覆盖的情况下才能成功恢复。
具体步骤如下：

1．从一个有效的备份中恢复损坏的数据文件。
2．mount 数据库。
3．执行以下查询：
SELECT FILE#, NAME, STATUS FROM V$DATAFILE；
如果发现要恢复的文件是 offline 状态的话，要先 online 它：
ALTER DATABASE DATAFILE '<full_path_file_name>' ONLINE；
4．执行以下查询：
SELECT V1.GROUP#, MEMBER, SEQUENCE#, FIRST_CHANGE#
FROM V$LOG V1, V$LOGFILE V2
WHERE V1.GROUP# = V2.GROUP# ；
这个将列出 redolog 文件所代表的 sequence 和 first change numbers。
5．如果数据库是非归档情况下，执行以下查询：
SELECT FILE#, CHANGE# FROM V$RECOVER_FILE；
如果 CHANGE#大于最小的 redolog 文件的 FIRST_CHANGE#，则数据文件可以被恢复，记得在应用日志的时候要把所有 redolog 文件全部应用一遍。
如果 CHANGE#小于最小的 redolog 文件的 FIRST_CHANGE#，则数据文件就不可以被恢复了，这时候你要从一个有效的全备份中去恢复数据库了，如果没有全备份的话，那你就只能把数据库强制打开到一个不一致的状态去 exp 出数据，然后重新建库导入数据，因为这种方式的恢复 oracle 是不推荐用户自己做的，所以这里我就不详细说明了。
6．恢复数据文件：
RECOVER DATAFILE '<full_path_file_name>'；
7．确信应用了所有的 redolog 文件，直至出现提示信息"Media recovery complete"，提示介质恢复完成后就可以了。
8．打开数据库。
ALTER DATABASE OPEN；

二。数据库此时处于打开状态。

如果你发现有回滚段的数据文件丢失或者损坏了，而此时的数据库是处于打开的状态下并且在运行，就千万不要关闭数据库了，因为在大多数的情况下打开的时候比关闭的时候好解决问题一些。

一般也是存在有两种情况：

A. 是 offline 丢失或损坏的数据文件，然后从一个备份中恢复，执行介质恢复以保持一致性。但是这种情况要求数据库是归档方式下才可以采用的。

具体步骤如下：

1. offline 丢失或损坏的数据文件。

   ALTER DATABASE DATAFILE '<full_path_file_name>' OFFLINE；

2. 从一个有效的备份中恢复。

3. 执行以下查询：

   SELECT V1.GROUP#, MEMBER, SEQUENCE#
   FROM V$LOG V1, V$LOGFILE V2
   WHERE V1.GROUP# = V2.GROUP# ；

   这个将列出你的所有 redolog 文件以及它们所代表的 sequence numbers。

4. 恢复数据文件：

   RECOVER DATAFILE '<full_path_file_name>'；

5. 确信你应用了所有的 redolog 文件，直至出现提示信息"Media recovery complete"，提示介质恢复完成后就可以了。

6. 把开始 offline 的那个数据文件 Online。

   ALTER DATABASE DATAFILE '<full_path_file_name>' ONLINE；

B。是 offline 那个存在丢失或损坏的数据文件所在的整个回滚段表空间，然后删除整个回滚段表空间并重建，但是你必须要杀掉那些在回滚段中已经激活的用户进程才可以 offline 的。

通常情况 A 就比较简单实现，但是更多的用户事务将会出错并且回滚。

具体步骤如下：

1. offline 存在丢失或损坏的数据文件的回滚段表空间中的所有回滚段：

   ALTER ROLLBACK SEGMENT <rollback_segment> OFFLINE；

2. 检测当然回滚段的状态：

   SELECT SEGMENT_NAME, STATUS FROM DBA_ROLLBACK_SEGS
   WHERE TABLESPACE_NAME = '<TABLESPACE_NAME>'；

3. 删除所有 offline 的回滚段：

   DROP ROLLBACK SEGMENT <rollback_segment>；

4. 处理那些 online 状态的回滚段。

   重新执行第二步的查询：

   SELECT SEGMENT_NAME, STATUS FROM DBA_ROLLBACK_SEGS
   WHERE TABLESPACE_NAME = '<TABLESPACE_NAME>'；

   如果你已经执行过 offline 操作的回滚段状态仍然是 online，则说明这个回滚段内有活动的事务。你要接着查询：

   SELECT SEGMENT_NAME, XACTS ACTIVE_TX, V.STATUS
   FROM V$ROLLSTAT V, DBA_ROLLBACK_SEGS
   WHERE TABLESPACE_NAME = '<TABLESPACE_NAME>' AND SEGMENT_ID = USN；

   如果没有返回结果，则证明存在丢失或损坏的数据文件的回滚段表空间中

的所有回滚段都已经被 offline 了，然后重新执行第二步，第三步。如果查询有结果返回，则状态应该是"PENDING OFFLINE"。接着查看 ACTIVE_TX 列，如果值为 0，则表明此回滚段中已经没有未处理的事务了，很快就会被 offline 的，然后等它 offline 后重新执行 2，3 步后跳至第六步。如果值大于 0，则继续到第五步。

5. 那些包含活动事务的回滚段 offline，活动的事务应该被提交或者回滚，执行下面的查询看看哪些用户占用了回滚段：

```
SELECT S.SID, S.SERIAL#, S.USERNAME, R.NAME "ROLLBACK"
FROM V$SESSION S, V$TRANSACTION T, V$ROLLNAME R
WHERE R.NAME IN ('<PENDING_ROLLBACK_1>', '<PENDING_ROLLBACK_N>')
AND S.TADDR = T.ADDR AND T.XIDUSN = R.USN;
```

最好能直接通知到那些 user 让他们自己去回滚或者提交事务，如果不能做到的话，那就只能强制性的杀掉进程了。

```
ALTER SYSTEM KILL SESSION '<SID>, <SERIAL#>';
```

杀掉进程后再过一段时间后回滚段会自动清除那些事务，然后就可以回到第二步继续查询了。

6. 删除回滚段：

```
DROP TABLESPACE <tablespace_name> INCLUDING CONTENTS;
```

7. 重建回滚段表空间。
8. 重建回滚段并 online 这些回滚段。

• 作者介绍

冯春培，毕业于北京信息工程学院。曾做电信计费后台程序开发，从事过开发DBA工作，目前公司主要做数据库优化产品开发。热爱ORACLE，在www.itpub.net任数据库管理版块版主(biti_rainy)，个人兴趣主要在oracle internal、performance tuning。对数据库管理、备份与恢复、数据库应用开发、SQL优化均有广泛理解。希望大家一起探讨oracle及相关技术。