第7章 回滚段管理

回滚段收集开始事务处理之前存在的数据的“前”映像。其他用户可以查询数据变化之前的情况。

无论数据库的其他部分工作如何出色，回滚段都需要一些特殊关注。由于回滚段控制着数据库处理事务的能力，所以直接决定着数据库的成功与否。

这一章描述DBA需要为回滚段执行的主要管理任务。这些主要任务是：

- 回滚段的基本功能。
- 回滚段使用可用空间的独特方式。
- 监控回滚段的使用。
- 正确选择数据库回滚段的大小和数量。

7.1 回滚段概述

SQL的rollback命令使用户能撤消对数据库所做的事务。这种功能对任何 update、insert或者delete事务都有效；但对数据库对象的变动则无能为力 (如alter table命令)。当选择其他用户正在改变的数据时，Oracle使用回滚段来展示变动前的数据。

7.1.1 数据库对回滚段的使用

回滚段包含数据库中发生的每一次事务处理，使数据库在多次变动中仍能保持读取的一致性。可用回滚段的数量和大小由 DBA在数据库创建时设定。

由于回滚段在数据库中创建，所以也必须在一个表空间中创建它们。第一个回滚段被称为SYSTEM，它存储在SYSTEM表空间中。接下来的回滚段通常也在另外至少一个表空间中创建。由于回滚段在表空间中创建，所以它们的最大尺寸也就要受数据文件空间容量的限制。因此，恰当地设置回滚段的大小是一项非常重要的工作。图 7-1描述回滚段在表空间中的存储情况。

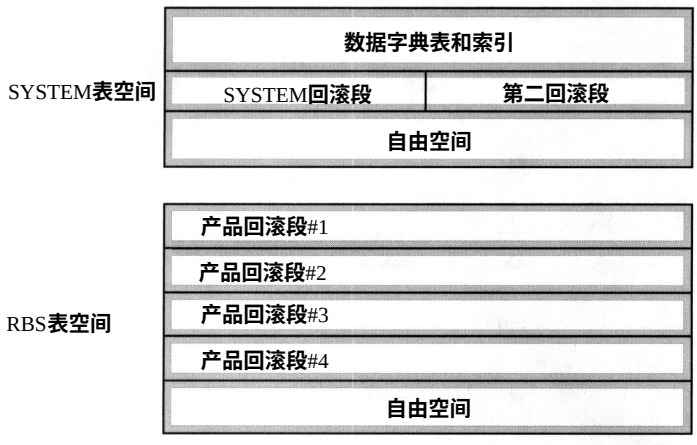


图7-1 表空间中回滚段的存储情况

回滚段条目(rollback segment entry)是“前”映像数据块的集合，含有被一个事务修改的数据行。每个回滚段条目必须完整地包含在回滚段内。一个回滚段可以支持多个回滚段条目。这就使得可用回滚段的数目直接关系到数据库的性能。图 7-2说明了回滚段与回滚段条目之间的关系。

数据库以循环方式将事务分配给回滚段。这种分配方式使得每一个回滚段中被分配的事务数目比较平均。尽管可以给一个事务指定回滚段 (见本章后面的7.1.3节“为事务指定回滚段”)，但大多数事务使用缺省方式。因为使用循环分配方法，所以拥有不同大小的回滚段没有任何益处。

可以创建指定为私有 (private)或公用(public)的回滚段。这些指定表示该回滚段是对一个实例可用还是对访问那个数据库的多个实例可用。当一个实例打开一个数据库时，就显式获得私有回滚段(见本章7.1.2节“激活回滚段”)。如果另一个实例访问同一数据库，这个实例就不能使用已由第一个实例获得的同一个私有回滚段，而只能使用它自己的私有回滚段或是从公用回滚段的共享池中获得。

图7-2 回滚段中的回滚段条目的存储情况

1. SYSTEM回滚段

SYSTEM回滚段在数据库创建时自动创建。在 create database命令中，没有指定SYSTEM回滚段的名称及存储参数，它将自动在 SYSTEM表空间中创建。

SYSTEM回滚段的用途因配置不同而不同，若数据库有多个表空间 (几乎所有数据库都是这样)，就需要自己创建第二回滚段来支持多个表空间。如果有其他回滚段， SYSTEM回滚段就只用于管理数据库级的事务(如修改记录用户权限的数据字典表)。

2. 第二回滚段

如果数据库中有多个表空间，就需要创建第二回滚段。除非至少有两个可用的回滚段，否则就不能将任何对象写入非 SYSTEM表空间中。数据库创建时必须在 SYSTEM表空间中创建第二回滚段。不过，不应使第二回滚段对成品事务有效，否则进行大型事务处理时，将会直接威胁到SYSTEM表空间的自由空间。

因此，第二回滚段只应在数据库创建时被使用。只要回滚段的表空间已经创建，就需要在该表空间中创建回滚段，或者撤消 SYSTEM表空间中的第二回滚段。

在考虑回滚段的情况下，数据库的创建过程应如下进行：

1) 创建一个数据库时，会自动在 SYSTEM表空间中创建SYSTEM回滚段。

2) 在SYSTEM表空间中，创建名为r0的第二回滚段。

3) 使新的回滚段有效。然后可创建其他表空间。

4) 创建RBS表空间，为以后的回滚段做准备。

5) 在RBS表空间中，创建另一个回滚段。

6) 停用SYSTEM表空间中的第二回滚段，并激活RBS中的新回滚段。

尽管在数据库创建后就不再需要第二回滚段，但依然可以使其存在（但处于非激活状态）。为此，停用该回滚段（见本章后面7.1.2节“激活回滚段”），但不要撤消它。这样，在出现影响RBS表空间的紧急情况下，就可以迅速激活这个回滚段。

3. 产品回滚段

非SYSTEM产品回滚段(Non-SYSTEM production rollback segment)，支持使用数据库创建的回滚段条目。它们支持用rollback命令来恢复修改前的数据映像。当回滚段出现问题或用户取消事务时，它们回滚事务。查询时，回滚段在执行查询前为那些被修改——但没有提交——的数据构造数据映像。

数据库按照循环(round-robin)方式，向产品回滚段分配回滚段条目。这种方法用于给回滚段分配事务(如图7-2所示)。由于一个回滚段可以支持多个事务，所以能创建一个大型回滚段来处理数据库中的所有事务。然而，这样的设计会因为回滚段的冲突而引起性能问题。

因此，可能要创建许多小型回滚段，以便保证每个事务拥有自己的回滚段。但是如果创建的回滚段太小，就必须对回滚段进行动态扩展以适应它们的事务处理，从而使运行发生困难。所以，数据库回滚段的设计涉及如何恰当折衷回滚段大小和数量这两个问题。本章中7.5节“选择回滚段的数量和大小”将讨论此问题。

7.1.2 激活回滚段

激活回滚段就是使它对数据库用户可用。一个回滚段可以在不关闭的情况下，将其设置为非激活状态。这样既可以保留已分配给它的空间，也可以在将来需要时重新激活它，下面的例子提供了控制回滚段可用性的全部命令：

通过alter rollback segment命令使一个回滚段从激活状态转变为非激活状态。

```
alter rollback segment SEGMENT_NAME offline;
```

若要撤消回滚段，请使用drop rollback segment命令。

```
drop rollback segment SEGMENT_NAME;
```

若要创建回滚段，请使用如下所示的create rollback segment命令：

```
create rollback segment SEGMENT_NAME
tablespace RBS;
```

要注意的是，这个create rollback segment命令创建一个私有回滚段(因为没有使用public关键字)，并将它创建在称作RBS的非SYSTEM表空间中。由于未指定存储参数，回滚段将使用这个表空间的缺省存储参数(有关回滚段的空间管理情况，将在本章后面描述)。

尽管回滚段已经创建，但还没有被数据库使用。若要激活新建的回滚段，请使用下面的命令使它联机：

```
alter rollback segment SEGMENT_NAME online;
```

一旦回滚段被创建，就应当将其列入数据库的init.ora文件中。这个文件在数据库运行期间为只读文件。下面示出了回滚段的init.ora条目样例：

```
rollback_segments     = (r0,r1,r2)
```

注意 SYSTEM回滚段不会在init.ora文件中列出，并且 SYSTEM 回滚段不能被撤消；它总是同实例能获得的其他回滚段一起被获取。

对于这个数据库， r0、r1和r2这三个回滚段为联机状态。若要取消其中一个，只需从init.ora文件中移去其条目。当使一个回滚段为联机状态时，在其当前有效事务完成之前它将一直保持联机状态。如本章后面所述，可以通过 V$ROLLSTAT视图来查看回滚段的PENDING OFFLINE状态。

7.1.3 为事务处理指定回滚段

可以用set transaction命令指定一个事务处理应使用的回滚段。但应在实施大的事务处理前使用这个命令，以确保这个事务处理使用为它们特别创建的回滚段。

通过set transaction命令指定的这种设置只适用于当前事务。下面的例子示出一系列的事务。第一个事务被指定使用 ROLL_BATCH回滚段，第二个事务 (在第二个commit之后)将被随机地分配给一个产品回滚段。

```
commit;

set transaction use rollback segment ROLL_BATCH;
insert into TABLE_NAME
select * from DATA_LOAD_TABLE;

commit;

REM*  The commit command clears the rollback segment assignment.
REM*  Implicit commits, like those caused by DDL commands, will
REM*  also clear the rollback segment designation.

insert into TABLE_NAME select * from SOME_OTHER_TABLE;
```

7.2 回滚段的空间使用

当一个事务开始时，Oracle就开始向回滚段写入一个回滚段条目。该条目不能够扩展到其他任何回滚段，也不能动态地转向使用另一个回滚段。条目开始在回滚段的一个盘区中连续写入。这个盘区中的每一块必须只包含一个事务的信息。不同事务的块可以被存储在同一个盘区中(见图7-3)。

图7-3示出了回滚段中一个盘区的前五个块。两个不同事务正把回滚信息存储在这个回滚盘区中。

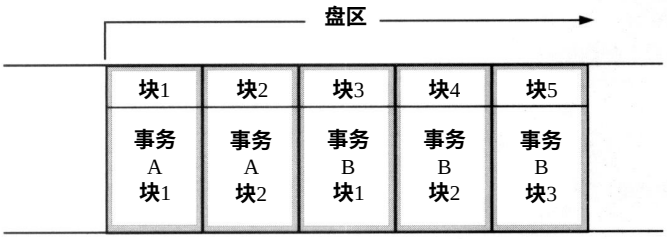


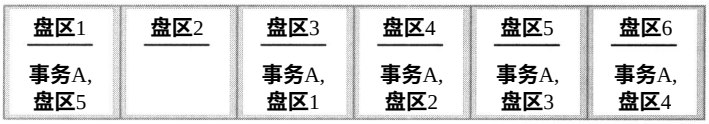
图7-3 一个回滚段盘区中存储两个事务

在理想的数据库中，一个盘区只有一个事务。然而，这种情况并不常见。这是因为，当一个事务处理无法在一个盘区中再获得空间时，回滚段会寻找另一个盘区来继续写入回滚段条目。

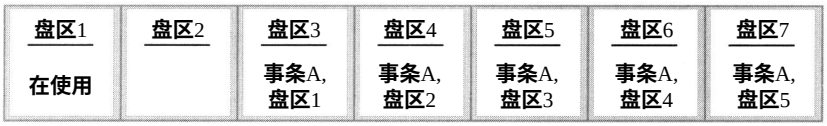
数据库首先试图把条目扩展到回滚段的下一个盘区。如果当前盘区是回滚段的最后一个盘区，数据库则试图把条目扩展到它的第一个盘区。

然而，这第一个盘区可能已被使用。如果是这样的话，回滚段就不得不获得一个新盘区。图7-4说明了这种扩展选择的过程。

a) 事务A从盘区3开始填充到盘区6



b) 如果盘区1能用，事务A就使用它



c) 如果盘区1正使用着，回滚段就进行扩展

图7-4 回滚段的选择性扩展

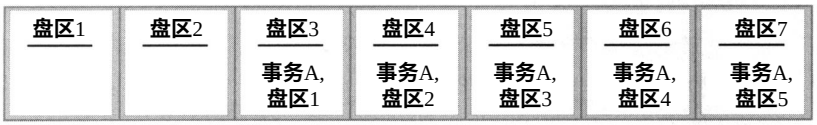
如图7-4a所示，一个事务(这里是事务A)已占用回滚段r1的四个盘区。由于事务A从回滚段r1的盘区3开始，所以此事务开始时，这个回滚段的前两个盘区必须已不可用。这时，这个事务已拥有四个盘区，并且在寻找第五个盘区。

由于它已经占据了该回滚段的最后一个盘区(6#盘区)，所以数据库将检查r1回滚段中的第一个盘区是否含有激活的事务数据。如果没有，则这个盘区将被用作事务A条目的第五个盘区(见图7-4b)。不过，如果这个盘区正被一个事务使用，则回滚段会动态进行自身扩展，从而7#盘区将用作事务A的第五个盘区(见图7-4c)。

一旦一个事务完成，它的数据不会从回滚段中删除。旧的回滚数据保留在回滚段中，以便对提交前开始执行的事务和查询提供服务。对于冗长的查询，这样可能会引发问题，产生如下的错误消息：

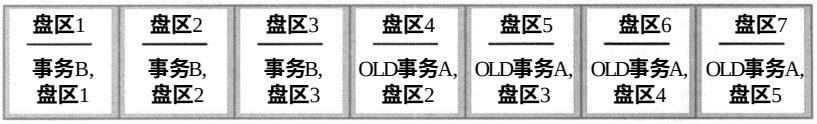
```
ORA-1555：snapshot too old (rollback segment too small)
```

这种错误是由对“active(激活的)”数据的定义引起的。考虑到图7-4a中事务A那样的大型事务，当一个冗长查询访问事务A所处理的表时，需要使用由事务A的回滚段条目存储的数据块。然而，一旦事务A完成，那些数据块将被标记为inactive(非激活)。尽管此时对这些数据块的冗长查询还没有结束，但这些数据块可能被其他事务所重写。这将导致对这些数据块的查询失败。具体情况如图7-5所示。



a) 事务A正在进行，其数据由一个大型查询使用

b) 事务A完成，其条目数据保留并被长期运行的查询使用，事务 B开始



c) 事务B重写大型查询使用的块，查询失败

图7-5 “快照太旧”引起的查询失败

如图7-5所示，一个事务可以跨多个盘区。在图 7-5a中，事务 A使用回滚段的5个盘区并且已经完成。即使在事务完成后，其他用户也仍可使用这些数据。例如，如果其他用户在事务 A完成前正在查询数据，就需要使用回滚段条目来重新构造所要查询的表数据。虽然事务 A的回滚条目数据处于非激活状态，但它正在使用中。当事务 B开始时(见图7-5b)，它从回滚段的第一个可用盘区开始。当它扩展到第二个盘区后，事务 A的回滚段条目数据将会被重写 (因为数据为非激活状态)，从而导致任何使用这些回滚段条目数据的行为失败 (因为它们正在被使用着)。

这其中有两个导致查询失败的重要因素。首先，在进行一个冗长查询的同时，数据也正在被处理；换句话说，在这个时候，数据库中正同时进行着批处理和数据的联机事务处理。在前面对回滚段功能的讨论中，这种问题的解决方案已经很清楚：一个冗长查询操作不仅必须访问需要完成的所有表及索引，而且还必须访问存储在回滚段中的数据，以保证所读数据的一致性。

查询必须连续访问回滚段，直到查询完成为止。因此，冗长查询要求正在使用的回滚段条目没有被重写。但是这些条目一旦完成，就无法保证这种情况。错误消息 (“回滚段太小”) 中给出的建议是通过解决另一个相关问题来解决这个问题：通过增加回滚段空间来延长第一个盘区被重写的时间间隔，从而推迟回滚段中已存在的条目被重写。

给回滚段增加更多的空间并不是一个真正的解决方案。这只是一个延时策略，因为回滚段最终会重写所有数据块。比较合适的方案是在联机事务处理最少时安排冗长查询。

最佳storage子句

如前面的图7-4所示，回滚段可以动态扩展来处理大型事务条目的装载。一旦这种扩展完成，回滚段就保留在扩展期间获得的空间。这可能就是回滚段空间管理问题的症结所在。因为一个大型事务处理可能消耗掉回滚段表空间的所有空间，这将阻止该表空间中其他回滚段的扩展。

可以通过两种改变回滚段所用 storage子句的方法来解决这个问题。首先，不再需要回滚段的 pctincrease参数。因为这会强制回滚段均匀增长，而不是几何增长 (见第4章4.6.1节“storage子句的意义”对这个问题的说明)。

第二种改变回滚段 storage子句的方法是增加 optimal参数。该参数允许数据库管理为回滚段指定一个最佳长度(以字节为单位)。当回滚段扩展超过这个长度时，它便会通过删除最旧的盘区来实现自身的动态收缩(shrink)。

初看起来，这似乎是一个灾难性的选择。它阻止一个回滚段使用表空间的全部自由空间。不过请注意，这个数据库：

1) 动态扩展回滚段，造成性能问题。

2) 动态选择并删除最旧盘区，也同样造成性能问题。

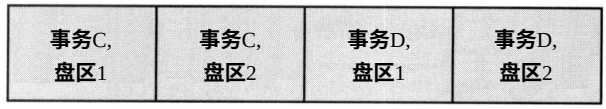
3) 在非激活数据被覆盖前，删除它们。

最后一点会使具有 optimal尺寸的数据库设置得太小，从而引发比图 7-5所示的“快照太旧”情况更大的问题。这是因为旧的事务数据现在可以用两种方式消除：重写或者在收缩当中被删除。

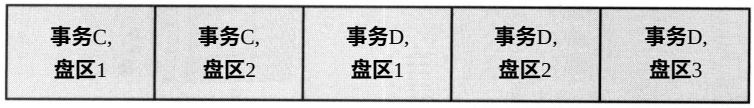
图7-6展示了通过optimal参数扩展和收缩回滚段的进程。在事务大小不详且回滚段表空间的可用自由空间又有限时，这种方法很有用。然而即使如此，它也不是非常合适的回滚段尺寸解决方案。这个问题将在本章的 7.5节“选择数量和大小”中讨论。

如图 7-6所示，当一个事务完成时，回滚段要检查自己的 optimal值。如果超过了这个optimal值，回滚段会删除它最旧的盘区 (本例中的盘区1)。这样便可以在查询服务中使回滚段保持optimal值。收缩回滚段的副作用是减少了非激活回滚数据的数量，而这些非激活回滚数据对当前事务依然可用。

当回滚段超过其 optimal值时，如果回滚段中没有非激活的盘区，将会出现什么情况呢？回滚段将必须继续扩展以支持该事务。如果该回滚段被强制 (例如通过一个大型事务)超过其optimal值，则它需要的空间暂时是回滚段的一部分。结束时，下一个超过 optimal设置值的事务在它结束时将促使数据库恢复使其超过 optimal值的任何空间。



a) 事务C占据回滚段的盘区 1和盘区2，事务D占据盘区3和盘区4，OPTIMAL设置被设置成等于3个盘区



b) 事务D扩展到第三个盘区，强制回滚段扩展

c) 事务C完成，然后事务 D完成。最旧的盘区被从回滚段中删除，回滚段收缩为其最佳尺寸

图7-6 回滚段动态收缩为最佳尺寸

7.3 监控回滚段使用

对回滚段的监控要求与数据段类似。有关对段的空间和内存使用的监控方法，请参见第 6章。由于回滚段是事务处理期间被访问的动态对象，所以它们的其他特性也需要被监视。

7.3.1 监控当前空间分配

若要查看数据库回滚段的当前空间分配情况，可查询 DBA_SEGMENTS字典视图，这里的Segment_Type等于"ROLLBACK"：

```
select * from DBA_SEGMENTS
 where Segment_Type = 'ROLLBACK';
```

表7-1列出了查询后返回的有关列。

表7-1 DBA_SEGMENTS中的回滚段相关列

列 名	描 述
Segment_Name	回滚段的名称
Tablespace_Name	存储回滚段的表空间
Header_File	存储回滚段第一个盘区的文件
Bytes	实际分配的回滚段大小，以字节为单位
Blocks	实际分配的回滚段大小，以Oracle块为单位
Extents	回滚段中的盘区数
Initial_Extent	以Oracle块为单位初始盘区的大小
Next_Extent	以Oracle块为单位下一个盘区的大小
Min_Extents	回滚段的最小盘区数
Max_Extents	回滚段的最大盘区数

optimal参数的值没有存储在 DBA_SEGMENTS中。它存储在 V$ROLLSTAT动态性能表的OptSize列中。若要检索这个值，必须对 V$ROLLSTAT和V$ROLLNAME一起查询，以同时获取该回滚段的名称和 optimal参数值。

```
select N.Name,          /* rollback segment name */
       S.OptSize        /* rollback segment OPTIMAL size */
from V$ROLLNAME N, V$ROLLSTAT S
where N.USN=S.USN;
```

如果没为回滚段指定 optimal值，查询返回的OptSize值将是空值(NULL)。

由于回滚段是数据库中的物理存储段，所以第 6章给出的空间监控脚本文件中也包含了有关回滚段的内容。表空间的空间监控程序会报告 RBS或SYSTEM表空间中可用自由空间的任何变动。盘区的监控脚本文件存储了所有回滚段的有关记录。这样做，回滚段中空间分配的所有改变便会立即被发现。

第6章中Command Center空间监控脚本文件中的查询通过查询 DBA_SEGMENTS视图来完成，表7-1列出了这个视图中主要的列。

7.3.2 收缩回滚段

可以对回滚段进行收缩。可以用 alter rollback segment命令中的shrink子句，将回滚段收缩到想要的大小。若没有指定收缩尺寸，回滚段将收缩到其 optimal值。但不能把回滚段收缩到小于两个盘区。

在下面的例子中，对r1回滚段做了两次修改。第一个命令将r1回滚段收缩到15MB，第二个命令将它收缩到其optimal值：

```
alter rollback segment R1 shrink to 15M;

alter rollback segment R1 shrink;
```

7.3.3 监控当前状态

要得到有关回滚段状态的信息，可以查询DBA_ROLLBACK_SEGS视图。这个视图不仅含有DBA_SEGMENT中提供的存储参数(包括Tablespace_Name、Initial_Extent、Next_Extent、Min_Extents、Max_Extents和Relative_FNo)，还包括了表7-2列出的另两个列：Status和Instance_Num。

表7-2 DBA_ROLLBACK_SEGS中的附加列

列 名	说 明
Status	回滚段的状态
Instance_Num	回滚段所属的实例。对于单实例系统，这个值是NULL(空值)

回滚段的状态将是表7-3列出的一个值。如果当前状态是OFFLINE或PARTLY AVAILABLE，就可以使一个回滚段联机。如果回滚段含有由未解决的跨越多个数据库的事务使用的数据，它的状态值为PARTLY AVAILABLE(有关分布事务的信息，请参见第三部分)。

表7-3 DBA_ROLLBACK_SEGS中的回滚段状态值

状 态	说 明
IN USE	回滚段处于联机状态
AVAILABLE	回滚段已被创建，但不处于联机状态
OFFLINE	回滚段处于脱机状态
PENDING OFFLINE	回滚段处于转到脱机状态的过程中
INVALID	回滚段已被删除。被删除的回滚段在数据字典中以这种状态列出
NEEDS RECOVERY	回滚段含有不能被回滚或已被破坏的数据
PARTLY AVAILABLE	回滚段含有分布式数据库中未解决事务的数据

7.3.4 监控动态扩展

回滚段可以扩展，也可以收缩。另外，回滚段条目也会在盘区不够时，在回滚段内一个接一个地覆盖盘区。所有这三种情况，都需要数据库执行另外的操作来处理这些事务。但这些附加操作会影响数据库的性能。

要考虑到，当一个条目覆盖和使回滚段扩展到另一个盘区时，那些大型事务处理常常会使回滚段扩展到超过optimal值。处理这种事件的过程很像下面所述：

1) 事务处理开始。
2) 在回滚段头中，为新事务条目增加一个条目。
3) 事务条目在回滚段的一个盘区中获得数据块。
4) 条目试图覆盖到第二个盘区。若没有可用段，则必须扩展回滚段。
5) 扩展回滚段。

6) 更新用于空间管理的数据字典表。

7) 事务完成。

8) 回滚段检查是否超过它的盘区optimal值，若已超过，就执行下面步骤。

9) 回滚段选择其最旧的非激活盘区。

10)删除这个最旧的非激活盘区。

如果回滚段大小已定，条目适合在一个盘区中完成，则过程如下：

1) 事务处理开始。

2) 在回滚段头部为新事务条目增加一个条目。

3) 事务条目在回滚段的一个盘区中获得数据块。

4) 事务完成。

空间管理所需要的系统开销量明显地节省了。

收缩、覆盖、扩展等一系列操作，均可使用V $ ROLLSTAT动态性能表来进行监控。如下面两小节所述，可以从这个视图以特定的时间间隔检索记录或通过特定查询来检索记录。

1. 在一个时间间隔内的动态扩展

若要确定在一个特定的时间间隔中V$ROLLSTAT中各项值的变动，可以使用系统统计脚本文件。这两个脚本文件是UTLBSTAT.SQL和UTLESTAT.SQL。它们位于Oracle软件主目录下的/rdbms/admin子目录中。可以按第6章所述的方式来运行它们。

UTLBSTAT创建一个存储V$ROLLSTAT表中的当前值的表。当UTLBSTAT再次运行时，将V$ROLLSTAT的值与那些存储的值进行比较并报告它们之间的差异。重要的是，在运行UTLBSTAT和UTLESTAT脚本文件时不能关闭数据库。因为数据库在系统启动时，重新在V$ROLLSTAT表中设置统计值，由UTLBSTAT生成的基线值在数据库重新启动后将不起作用。

在UTLBSTAT脚本文件中，用下面的命令在时间间隔的开始创建一些表。前两个 create table命令创建STATS$BEGIN_ROLL和STATS$END_ROLL表，它们都不含任何记录。 insert命令将V$ROLLSTAT表中的当前值存储到STATS$BEGIN_ROLL表中：

```
DROP TABLE stats$begin_roll;
CREATE TABLE stats$begin_roll
AS SELECT * FROM v$rollstat WHERE 0 = 1;

DROP TABLE stats$end_roll;
CREATE TABLE stats$end_roll
AS SELECT * FROM stats$begin_roll;

INSERT INTO stats$begin_roll SELECT * FROM v$rollstat;
```

当UTLESTAT文件运行时，它通过查询 V$ROLLSTAT表来填充STATS$END_ROLL表，然后创建一个STATS$ROLL表。STATS$ROLL表的唯一用途是存储确定 STATS$END_ROLL和STATS$BEGIN_ROLL记录之间差别的查询结果。

```
INSERT INTO stats$end_roll SELECT * FROM v$rollstat;

CREATE TABLE stats$roll
AS SELECT  e.usn undo_segment,
          e.gets-b.gets trans_tbl_gets,
    e.waits-b.waits trans_tbl_waits,
    e.writes-b.writes undo_bytes_written,
```

```
e.rssize segment_size_bytes,
     e.xacts-b.xacts xacts,
e.shrinks-b.shrinks shrinks,
     e.wraps-b.wraps wraps
FROM stats$begin_roll b, stats$end_roll e
     WHERE e.usn = b.usn;
```

STATS$ROLL表中的数据列出了每一个回滚段和在运行 UTLBSTAT和UTLESTAT的时间段内积累的统计数字。

对STATS$ROLL表的查询将返回表7-4中列出的列。

表7-4 STATS$ROLL中的列

列 名	说 明
Trans_Tbl_Gets	需要回滚段头的请求数量
Trans_Tbl_Waits	等待所需要的回滚段头的请求数量
Undo_Bytes_Written	写入回滚段中的字节数量
Segment_Size_Bytes	回滚段的字节数。注意，这个列只考虑最后的值 (不是差值)
Xacts	激活的事务的数量
Shrinks	为保持optimal尺寸而进行的收缩次数
Wraps	回滚段条目从一个盘区覆盖到另一个盘区的次数

必须对UTLESTAT.SQL的回滚段部分进行两处修改。第一，应改变 UTLESTAT查询中的Xacts引用。Xacts列反映的是激活事务的当前数量，而不是事务的累积数量。这样， Xacts的开始值与结束值之间的差别就没有什么意义。如果关心这个值，可以将下面的语句

```
e.xacts-b.xacts xacts,
```

改为

```
e.xacts        xacts,
```

否则，就将它完全删除。

第二，可以在报告中包含 Extends(扩展)列(这是累积列)。Extends列反映回滚段扩展的次数。对于那些还没有达到optimal值的回滚段，监控Extends列数值最合适。

这样看来，监控回滚段动态扩展、覆盖和收缩都只是一些简单的事。可以使用UTLBSTAT/UTLESTAT报表的回滚段部分确定回滚段中所有动态扩展的特性。

如果(Trans_Tbl_Waits/Trans_Table_Gets)的值大于0.01(在要获取回滚段头的请求中有 1%以上要等待)，数据库中的回滚段就会出现冲突。

最后需要注意的一点是：这些脚本文件可以将记录插入表中。换句话说，它们也执行事务处理，也生成回滚段条目，因此可能会影响最终的结果。通过这些查询所生成的字节数小于100个字节。可以通过从一个远程数据库进行查询来避免这种影响 (见第6章的相关部分)。

2. 特别查询

可以用特定的方式来查询 V$ROLLSTAT表。表7-5中列出了V$ROLLSTAT中的列。当查询V$ROLLSTAT时，也可以查询V$ROLLNAME。V$ROLLNAME表将回滚段号映射到其名字上(例如，‘SYSTEM’、‘R0’)。

表7-5 V$ROLLSTAT中的列

列 名	说 明
Usn	回滚段号
Extents	回滚段中的盘区数
RsSize	回滚段的字节数
Writes	写入回滚段中的条目的字节数
Xacts	激活的事务数
Gets	需要回滚段头的请求数量
Waits	等待所需回滚段头的请求数量
Optsize	回滚段的optimal参数值
Hwmsize	RsSize在使用中所达到的最大值(高水位标志)，以字节为单位
Shrinks	为达到 optimal数值而进行的收缩次数
Wraps	回滚段从一个盘区覆盖到另一个盘区的次数
Extends	回滚段获取新盘区所需要的时间
Aveshrink	收缩中释放的平均字节数
Aveactive	激活的盘区的平均尺寸
Status	回滚段的状态；与前面列出的状态值类似。这两个状态值是ONLINE(与 ‘ IN USE ’ 相同)和 PENDING OFFLINE(与 ‘ PARTLY AVAILABLE ’ 相同)
Curext	当前盘区
Curblk	当前数据块

V$ROLLNAME和V$ROLLSTAT之间存在着一对一关系，并且都具有一个主关键字USN(Undo Segment Number)。当以特别方式查询它们时，可以用这个关键字来将它们连结，如下所示：

```
select
   N.Name,                      /* rollback segment name */
   S.RsSize                     /* rollback segment size */
from V$ROLLNAME N, V$ROLLSTAT S
where N.USN=S.USN;
```

7.3.5 每个回滚段中的事务

确定每一个回滚段中拥有激活条目的用户，可以有效地解决两个问题：当前回滚段是如何分配的以及谁在工作。

理解这种查询需要知道事务在回滚段头内获取锁。因此可以将V$LOCK表与V$ROLLNAME表连结。由于锁由进程拥有，所以可以把V$LOCK与V$PROCESS连结，以便查看V$PROCESS中的用户进程到V$ROLLNAME中的回滚段名的映射。

```
REM  Users in rollback segments
REM
column rr heading 'RB Segment' format a18
column us heading 'Username' format a15
column os heading 'OS User' format a10
column te heading 'Terminal' format a10
select R.Name rr,
       nvl(S.Username,'no transaction') us,
       S.Osuser os,
       S.Terminal te
```

```
  from V$LOCK L, V$SESSION S, V$ROLLNAME R
 where L.Sid = S.Sid(+)
   and trunc(L.Id1/65536) = R.USN
   and L.Type = 'TX'
   and L.Lmode = 6
order by R.Name
/
```

下面是上述查询的输出样例：

```
RB Segment          Username         OS User     Terminal
------------------- ---------------- ---------- ----------
R01                 APPL1_BAT        georgehj    ttypc
R02                 APPL1_BAT        detmerst    ttypb
```

这个输出例子表明，当前只有两个用户写入回滚段 (APPL1_BAT Oracle用户的两个不同会话由两个不同的操作系统用户来完成)。每个用户写入一个没有其他人使用的回滚段。 R01、R02回滚段是由现有事务使用的仅有的两个回滚段。如果有多个用户使用一个回滚段，对于这个回滚段输出中将会有多个记录。

7.3.6 回滚段中的数据量

可以查询V$ROLLSTAT来确定写入回滚段的字节数。 V$ROLLSTAT含有Writes列，它记录了自从数据库上次启动以来写入每一个回滚段的字节数。

若要确定一个指定的时间间隔内一个回滚段中的活动量，可以在测试期开始时选择 Writes列。当测试结束时，可以查询当前值，两者之间的差别便是在这个时间间隔中写入回滚段的字节数。由于关闭数据库将会复位 V$ROLLSTAT表中的统计值，所以在测试期间必须保持数据库为打开状态。

可以用下面的查询从V$ROLLSTAT表中选择Writes值：

```
select
   N.Name,                      /* rollback segment name */
   S.Writes                     /* bytes written to date */
from V$ROLLNAME N, V$ROLLSTAT S
where N.USN=S.USN;
```

要检测由一个事务创建的回滚段的大小，需要把这些查询与本章前面所给的一个命令结合起来。首先，通过执行它来分离该事务，它是数据库中的唯一进程。通过下面的命令把事务指定到特定的回滚段：

```
set transaction use rollback segment SEGMENT_NAME
```

然后为此回滚段查询 V$ROLLSTAT表中的 Writes列。当事务处理完时，重新查询V$ROLLSTAT。这两个Writes值之间的差值，便是事务回滚段条目的准确大小。

7.4 使用Oracle Enterprise Manager管理回滚段

可以使用Oracle Enterprise Manager(OEM，Oracle企业管理器)来执行大量的回滚段管理功能。可以使用OEM创建回滚段，通过将它们置于联机或脱机状态使其有效，收缩或删除它们。也可以创建一个与现有回滚段具有相同特征的新回滚段；这个选项称为 create like。本节描述通过OEM管理回滚段所需要的步骤。

7.4.1 从OEM创建回滚段

若要从OEM中创建回滚段，请遵循下述步骤

1) 从OEM屏幕左边选择Rollback Segment选项，但不选择已有的回滚段。

2) 下拉菜单或右击鼠标按钮选择Create(创建)选项。

3) 将合适的信息填充到Create Rollback Segment(创建回滚段)屏幕中。OEM将确认新回滚段的创建。

图7-7是使用下拉菜单选择创建回滚段的选项。

图7-8是使用右击鼠标按钮选择创建回滚段的选项。

图7-7 使用下拉菜单选择创建回滚段的选项

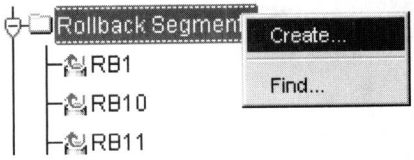


图7-8 使用右击鼠标按钮选择创建回滚段的选项

一旦做出选择，OEM将显示Create Rollback Segment窗口。在这个窗口中输入下述有关新回滚段的信息：

- 新回滚段的名称。
- 将它置于联机状态还是脱机状态。
- 它是公用的还是私有的。
- 它所在的表空间。

图7-9展示填充有合适信息的Create Rollback Segment窗口。为新回滚段选择的缺省表空间是表空间列表中的第一个表空间。其他缺省值是公用的和脱机。必须认真为回滚段选择正确的表空间。

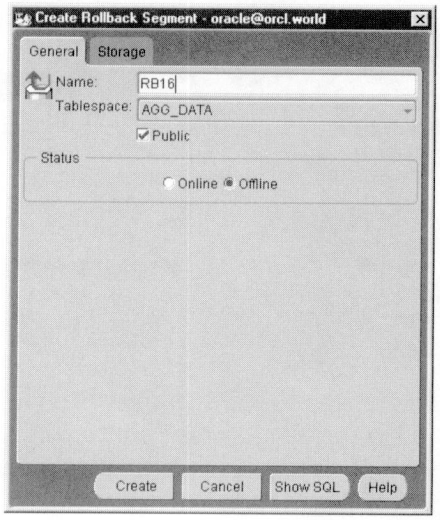


图 7-9填充有信息的Create Rollback Segment窗口

一旦新的回滚段被创建，将出现一个信息屏幕来向你表明已创建了新回滚段。单击 Okay 以确认新回滚段的创建。

7.4.2 创建与现有回滚段一样的回滚段

OEM使你能创建一个其结构与当前回滚段完全相同的回滚段。若要根据现有回滚段的结构创建新回滚段，请执行下述几步操作：

1) 选择要复制的回滚段。

2) 下拉菜单或右击鼠标按钮选择Create Like选项。

3) 将合适的信息填写到Create Rollback Segment窗口中。OEM将对创建新回滚段进行确认。

图7-10示出了OEM Oracle Storage Manager屏幕和下拉菜单；屏幕上有一个为复制选择的回滚段RB15，下拉菜单有一个突出显示的Create Like选项。

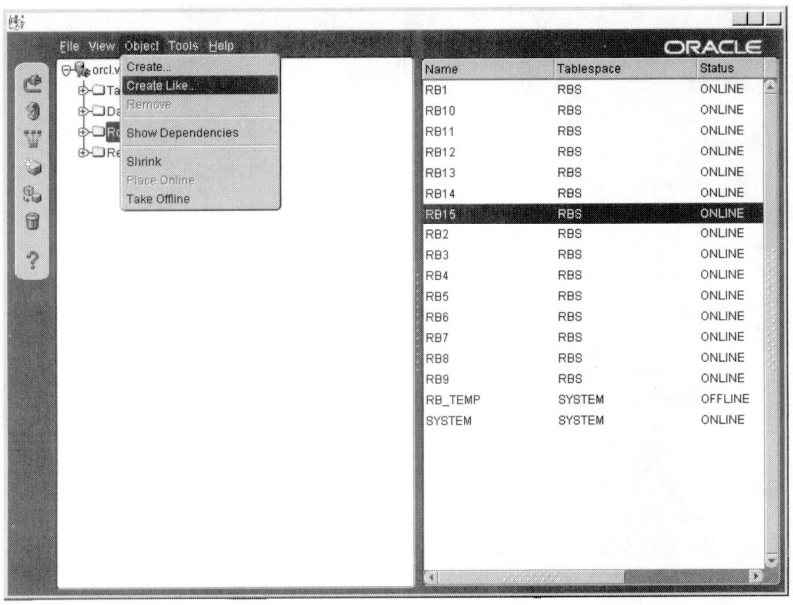


图7-10 为复制选择一个回滚段

图7-11显示用右击鼠标按钮选择选项。

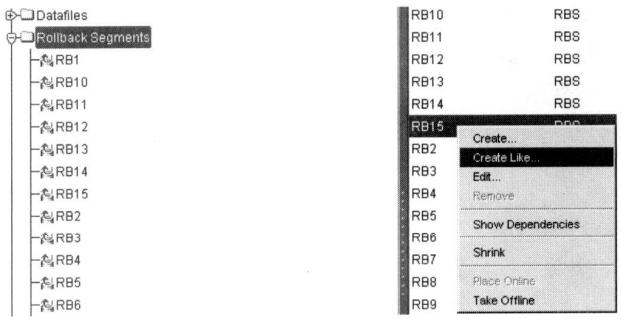


图7-11 创建复制的回滚段

一旦选中要复制的回滚段， OEM将显示Create Rollback Segment窗口。这时，请输入有关新回滚段的下述信息：

- 新回滚段名。
- 将其置于联机状态还是脱机状态。
- 它是公用的还是私有的。
- 它所在的表空间。

图7-12展示Create Rollback Segment窗口。回滚段的缺省值是为复制选择的回滚段的当前值。

一旦创建了新的回滚段，就显示一个表示回滚段已创建的信息屏幕。通过单击“Okay”来确认新回滚段的创建，新回滚段将可用。

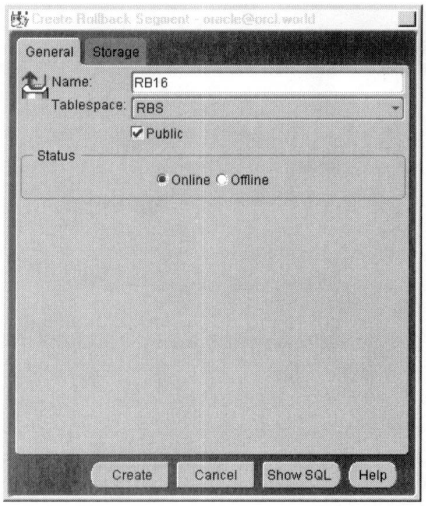


图7-12 Create Rollback Segment窗口

7.4.3 设置回滚段为联机状态

若要将一个回滚段设置为联机状态并使其有效，请遵循下述步骤：

1) 选择当前处于脱机状态的回滚段。
2) 下拉菜单或右击鼠标按钮，选中Place Online(设置联机状态)选项。
3) 确认选择。

图7-13示出了OEM Oracle Storage Manager屏幕及选择的回滚段RB15。

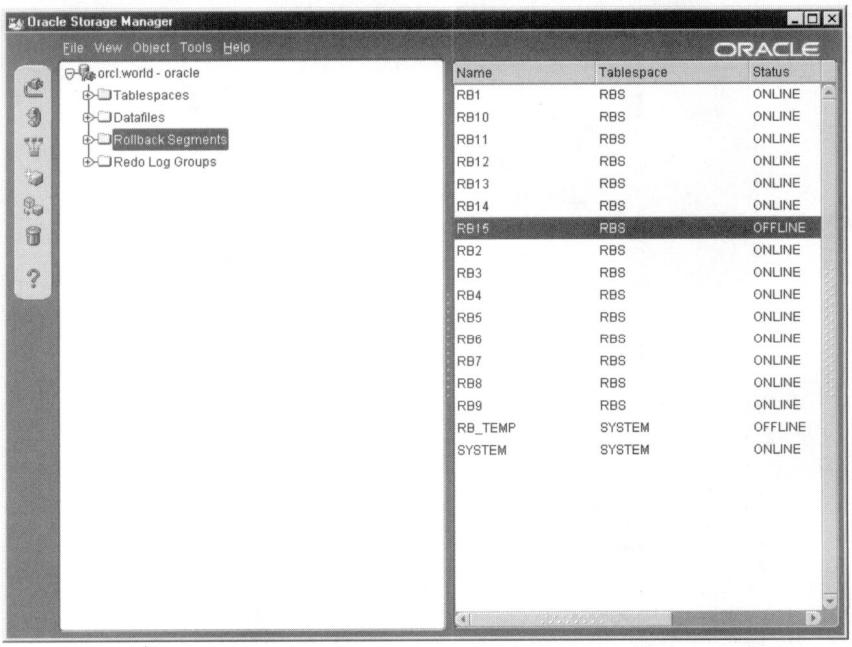


图7-13 选择一个脱机的回滚段

图7-14展示使用下拉菜单选择选项将RB15置于联机状态。

图7-15展示使用右击鼠标按钮选择选项将RB15置于联机状态。

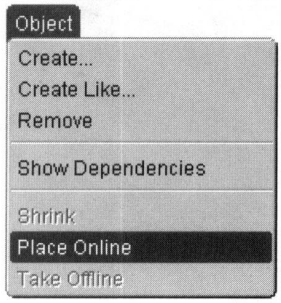


图7-14 使用下拉菜单将RB15置于联机状态

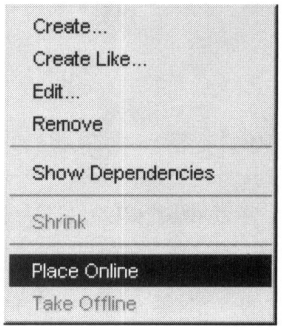


图7-15 使用右击鼠标将RB15置于联机状态

一旦做出选择，就显示操作确认窗口，以确认所选择的操作是正确操作。

OEM将提示“Are you sure want to bring the Rollback Segment Online?”，请选择“Yes”将回滚段置于联机状态。图7-16示出了联机状态的回滚段。

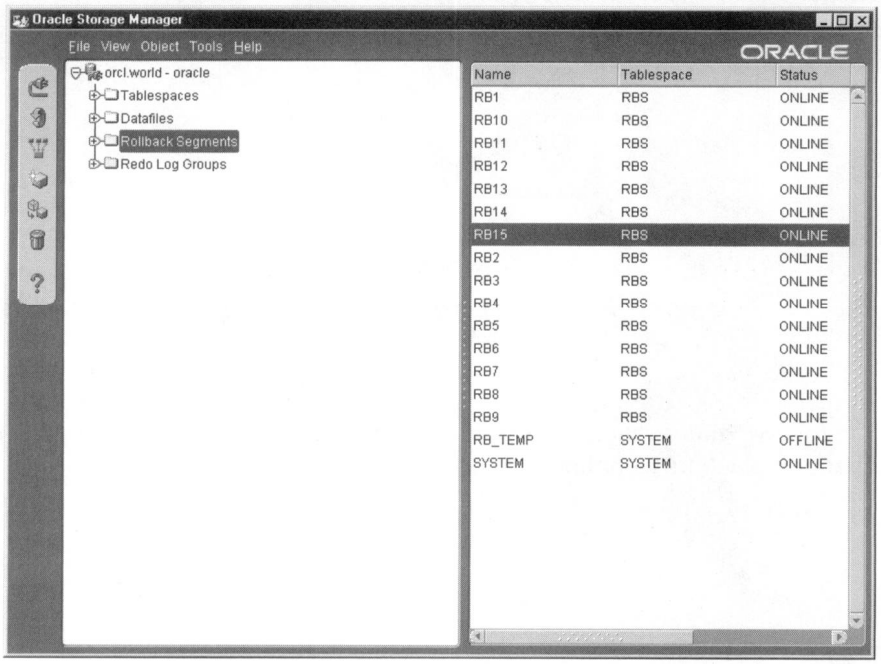


图7-16 联机状态的回滚段

7.4.4 设置回滚段为脱机状态

将一个回滚段设置为脱机状态且使其不可用所采用的步骤如下：

1) 选择当前为联机状态的回滚段。

2) 下拉菜单或右击鼠标按钮，选择Place Offline(设置为脱机状态)选项。

3)确认选择。

图17-16示出了OEM Oracle Storage Manager屏幕及选择的回滚段RB15。要注意的是，RB15的状态当前是联机状态。通过下拉菜单(见图7-17)或右击鼠标按钮(见图7-18)，为RB15选择Take Offline选项。

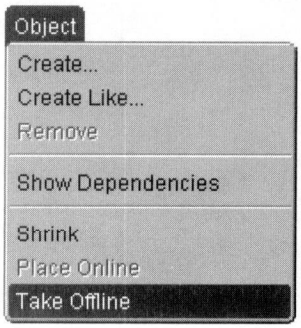


图7-17 使用下拉菜单选择Take Off line选项

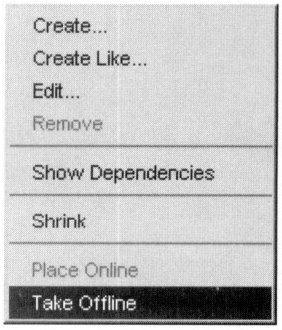


图7-18 使用鼠标选择Take Off line选项

一旦选择完毕，就显示操作确认窗口，以便确认所选择的操作是正确的操作。图 7-19显示的操作确认窗口的提示是：“Are you sure you want to take this Rollback segment offline?”。响应选项是Yes和No。

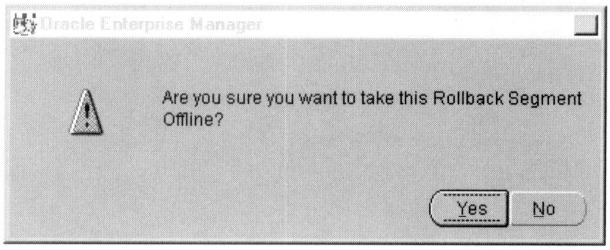


图7-19 操作确认窗口的提示

图7-20示出了操作结果，RB15现在处于脱机状态。

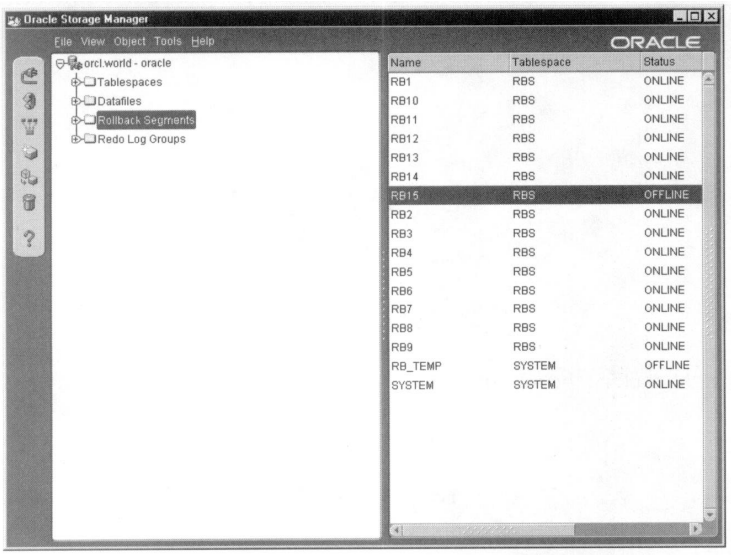


图7-20 脱机回滚段

7.4.5 删除回滚段

若要删除一个回滚段，请遵循下述步骤：

1) 选择要删除的回滚段。

2) 下拉菜单或右击鼠标按钮，选择 Remove(删除)选项。

3) 确认选择。

图7-21示出了OEM Oracle Storage Manager屏幕及选择的回滚段RB15。

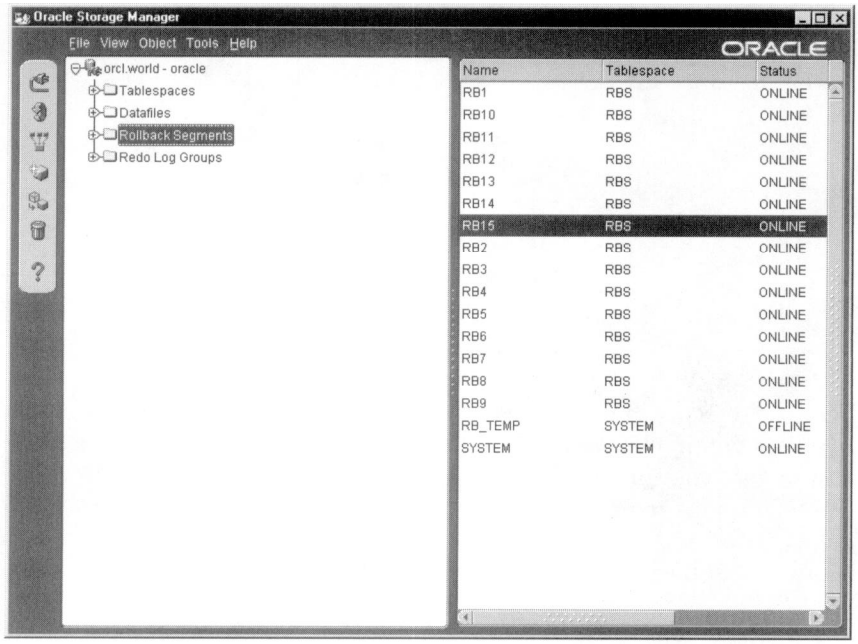


图7-21 选择删除的回滚段

图7-22展示用下拉菜单选择删除RB15的选项。

图7-23展示用右击鼠标按钮选择删除RB15的选项。

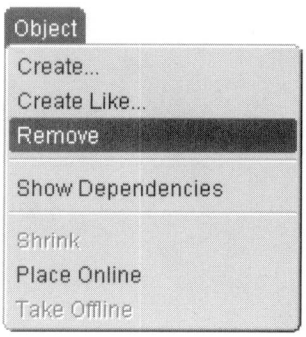


图7-22 使用下拉菜单删除RB15的选项

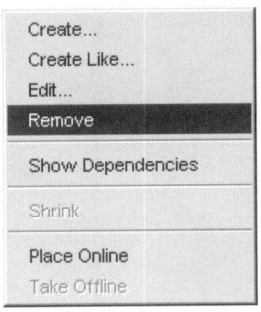


图7-23 使用鼠标删除RB15的选项

一旦做出选择，就显示操作确认窗口，以确认选择的操作是正确的操作。当 OEM显示提示：“ Are you sure you want to remove rollback segment RB15? ”。选择“ Yes ”确认删除回滚段；或选择“ No ”取消操作。

图7-24示出了操作结果，RB15被 删除。

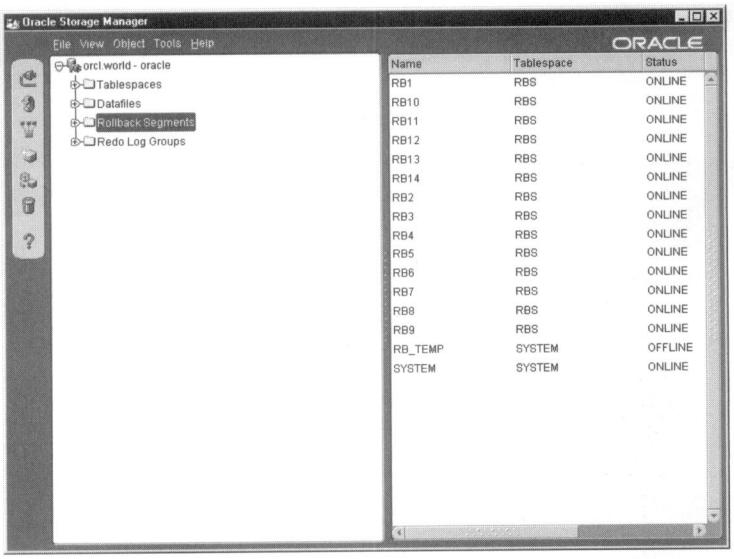


图7-24 删除RB15之后的回滚段列表

7.5 选择回滚段的数量和大小

可以用本章对回滚段条目的描述，为自己的数据库做类似的回滚段设计。这里要注意的是，除非数据库有着一致的事务特性，否则每个数据库的最终设计都不一样。

设计过程包括确定事务量并估计事务的数量及类型。下面几节，将会在一个应用实例中示范这个过程。可以从这个事务数据中产生回滚段的合适数量、尺寸及结构。

7.5.1 事务条目数量

设计过程的第一步是确定回滚段条目数据的总量，这是任何实例都要使用的。注意，这里有两种类型的条目要予以考虑。

- 激活的条目 —— 还没有被提交或回滚。
- 非激活、但正在被使用(IIU)的条目 —— 已经被提交或回滚，但其数据正被另一个进程使用(例如一个冗长查询)。

那些非激活且没有被其他进程使用的回滚段条目在这些计算中并不重要。

高效管理回滚段的关键因素是使非激活、但正在被使用 (IIU)的条目数据最少。作为一个数据库管理人员，没有办法发现 IIU条目所使用的回滚段空间，而只有在用户报告出现了ORA-1555 “ snapshot too old(快照太旧) ”，错误消息时，才能证明它们的存在 (见图7-5)。

要最小化回滚段中的IIU数据，需要知道何时执行那些冗长查询。如果它们恰好同时出现并带有多个事务，便会出现 IIU回滚段条目的不断积累。这时，无论回滚段有多大，这种糟糕的事务分配最终会导致查询失败。

要解决这个问题，需要将全部大型查询隔离开，以便在事务活动很少时分别运行它们。这种隔离既可以最小化 IIU回滚段条目数据，同时也有助于避免查询与事务之间可能出现的

I/O并发冲突。

可以用前面7.3.6节“回滚段中的数据量”给出的查询，来确定正写入回滚段的回滚段条目数据。每一个大型事务可以使用这里讲到的方法来确定其尺寸(在测试环境中)。确定事务尺寸应当是应用程序开发中确定数据库尺寸过程的标准部分。

还应注意的是，对事务中共享的IIU回滚数据量也要实施最小化。在一个事务访问一个数据表时，另一个事务正在对这个数据表进行处理，这样就会造成大量的IIU数据。如果将其最小化，就可以分配并测量这个系统的回滚段数据需求。

每一个事务都有一个与其相关联的开销。不过，这个头信息已在本章前面的统计查询中计算了。因此，这些查询对所需的回滚段空间量给出了非常准确的报告。

7.5.2 事务数量

一旦知道回滚段条目数据的总量，就必须考虑事务的数量。根据这些事务间的相对量，将它们分成几种类型。对每一组，都要确定最大及平均事务尺寸。对这部分回滚段大小的处理，只考虑系统正常运行时出现的事务，而不考虑所有数据装载。

可创建一个表7-6所示的电子数据表结构。所示数据仅作参考。表中的“事务类型”一列反映的是一个动物园巡回展出的应用程序数据条目。“数量”列反映的是每一类并发事务的数量。“全部条目尺寸”列以字节形式表示这种类型的所有并发条目的总尺寸。

这里要注意的是，“数量”列指的是不同事务的数量。如果在一个事务中更新多条记录，则这个列还是按一个事务计算。

表7-6 事务分布样例

事务类型	数 量	条目总尺寸	平均条目尺寸	最大条目
FEEDING_LOG	3	210KB	70KB	70KB
NEW_BIRTH	1	500KB	500KB	500KB
VISITOR_LOG	20	800KB	40KB	40KB
VISITOR_EATEN	1	700KB	700KB	700KB
TIME_SHEETS	10	500KB	50KB	50KB
合计	35	2710KB		

表7-6中列出的事务表明，用户在每10~100个记录后提交一次。例如，VISITOR_LOG事务的平均大小为40KB，这可能是10个4KB的记录或100个400字节的记录，具体情况依记录长度和提交频率而定。

根据表7-6中的数据，数据库中平均有35个并发事务，每一次它们平均要创建2710KB数据。

该数据库中最大的一个事务为700KB，把它作为确定回滚段大小的起点。这个事务必须适合单个回滚段。这个回滚段还将包含回滚段头部空间，并且也可能有非激活数据。若要计算能支持单个这种事务的回滚段的最小尺寸，可以用下面的计算公式。这个公式采用以下假设：

- 保留20%的回滚段空间作为自由空间。
- 将15%的回滚段空间用于IIU数据。
- 5%的回滚段空间由回滚段头区域使用。

```
最小可行尺寸(MPS) =最大事务尺寸×100 /
                   (100-(自由空间百分比 +IIU数据空间百分比+头空间百分比))
                   =最大事务尺寸×100 /
                   (100-(20+15+5))
                   =最大事务尺寸×100/60
                   =最大事务尺寸×1.67
                   =700KB×1.67
                   =1170KB
```

由于数据库以循环方式选择回滚段，所以处理这种事务的每一个回滚段都要不小于这个规模。

任何时候需要的回滚段总空间都可以计算。这同样将用到上面的假设。

```
最小总尺寸(MTS) =所有条目数据总尺寸×100/
                (100-(自由空间百分比 +IIU数据空间百分比+头空间百分比))
                =所有条目数据总尺寸×100 /
                (100-(20 + 15 + 5))
                =所有条目数据总尺寸×100/60
                =所有条目数据总尺寸×1.67
                =2710 KB×1.67
                =4525KB
```

因此，对于这个例子来说，所有回滚段的最低空间总量，在任何时候至少为 4525KB。

但应当分给这个空间多少个回滚段呢？由于回滚段的尺寸全部相同，所以最小值也就容易计算。正如你所看到的那样，每个回滚段的最小值为 1170KB。所有回滚段尺寸的最小总和为4525KB。下面的式子用这两个值进行了计算，以确定所需的回滚段最小数量：

```
最小回滚段数量 (MNRS) =最小总尺寸/
                       单个回滚段最小可行尺寸
                       = 4525 KB / 1170KB
                       = 3.87 向上舍入到4)
```

这将是所需回滚段数量的一个起点。注意，这个值只考虑了空间要求。

要进一步细化这个计算，则要考虑并发事务的数量。回滚段中的事务处理越少，数据库管理其空间需求的工作就越少。回滚段的最大数量是并发事务的数量 (假设每个回滚段一个事务)。对于表7-6中所示的样本数据，这个数值为 35。

```
回滚段的最大数量 = 并发事务数量=35
```

这样，产品数据库大约需要 4~35个回滚段。若要进一步缩小范围，就必须确定每个回滚段的事务数量。每个事务现在都将适合其自己的盘区而不是回滚段本身。而要做到这一点，就必须评估事务大小的分配。

如表 7-7所示，将那个动物园巡回展出中的事务分为两组。第一组的平均尺寸为 40~70KB；第二组的平均值为 500~700KB。由于这两组平均尺寸值之间存在巨大差别，所以不浪费空间或实施覆盖就不可能解决它们的空间要求。

表7-7 按平均尺寸分组的事务分配样例

事务类型	数 量	条目总尺寸	条目平均尺寸	最大条目
VISITOR_LOG	20	800KB	40KB	40KB
TIME_SHEETS	10	500KB	50KB	50KB
FEEDING_LOG	3	210KB	70KB	70KB

(续)

事务类型	数 量	条目总尺寸	条目平均尺寸	最大条目
小计	33	1510KB		
NEW_BIRTH	1	500KB	500KB	500KB
VISION_EATEN	1	700KB	700KB	700KB
小计	2	1200KB		
合计	35	2710KB		

条目尺寸较大的组数量非常少，在全部35个并发事务中仅占2个。因此可以设计处理小尺寸的条目组，而预留出空间来处理大型的意外并发条目组。因此，确定了小型条目组的尺寸也就形成了设计要求的最低范围。

对于一个小条目组，条目的总和为1510KB。这样，这个组所需的回滚段最低总规模为：

```
MTS = 1510KB×1.67 =  2522 KB
```

该组中的最小盘区尺寸是最大事务的尺寸：

```
最小盘区尺寸 (MES) = 70KB
```

一个70KB的盘区将处理每个小型条目事务。对于那些大型事务，它至少将需要 7个覆盖段(对500KB的条目，需要8个70KB的盘区)。若要最小化大型条目事务所覆盖的盘区数量，就要考虑大一些的盘区尺寸。对于大小为 125KB的盘区，被覆盖的盘区数量就很少，但对存储空间的要求也增加了。若再将盘区增大到 250KB，会进一步减少被覆盖的盘区数量。不过，如表7-8所示，这将需要增加大量物理空间。 .

在表7-8中，对大小不同的这三种盘区进行了比较。第一种盘区为 70KB，它基于计算出的最小平均尺寸。第二种盘区为 125KB，意图是减少支持大型事务所需要覆盖的盘区数量；第三种盘区为250KB，旨在进一步减少支持大型事务所需要覆盖的盘区数量。

表7-8中的“ 空间需求 ”一栏表示，第一种盘区是每个盘区一个事务，共需要 3570KB回滚段空间，但要产生 16个被覆盖的盘区。对于第二种盘区，尺寸增加到 125KB，所需的回滚段空间也增加到 5350KB，同时被覆盖的盘区数量减少到 7个。第三种盘区的尺寸再增加一倍，达到250KB，所须的回滚段空间增加到 9500KB，然而被覆盖的盘区数量也减少到了 3个。以上的空间需求计算全部假设一个事务对应一个盘区。这种假设也许会过高地估计了实际要求，但通常这是一个准确的起点。

从表7-8中可以看出，在这个例子中，只需将盘区增加 2/3，便会将被覆盖的盘区数量减少一半。但是如果使盘区尺寸超过 125KB，就不能如想象中那样相应地减少被覆盖的盘区数量。要注意的是，这些计算都是假设一个事务占用一个盘区，即使大型事务也是如此。

必须能够估计你的事务量，才能做到这一点。由于这个计算假设只有 15%的回滚段空间用以支持IIU条目，而这些条目也必须通过安排来使其最小化。为了做出这样的判断，必须能估计系统事务的类型、数量和特性。

表7-8 盘区尺寸调整折衷方案

盘区尺寸	事务条目类型	数 量	空间要求	覆盖盘区数
70KB	小(<70KB)	33	2310KB	0
	大(500KB)	1	560KB	7
	大(700KB)	1	700KB	9
		合计	3570KB	16

(续)

盘区尺寸	事务条目类型	数 量	空间要求	覆盖盘区数
125KB	小(<70KB)	33	4125KB	0
	大(500KB)	1	500KB	3
	大(700KB)	1	750KB	5
	合计		5350KB	8
250KB	小(<70KB)	33	8250KB	0
	大(500KB)	1	500KB	1
	大(700KB)	1	750KB	2
	合计		9500KB	3

由表7-8中的选项可以看出，小尺寸的盘区几乎一直是最合适的选择，因为它一般是数量最大的事务类型，而决定因素就在于事务的数量分布。在这个例子的 35个并发事务中，有33个是小型事务条目，所以表中最大尺寸的盘区可以不予考虑，而只需在较小的盘区(70~125KB)之间做出选择。由于只需将盘区尺寸增加一些，便可迅速降低被覆盖的盘区数量，所以选择125KB。要注意的是，我们总是在最小盘区尺寸(MES)和消除全部覆盖所需要的最小盘区尺寸之间做出最佳折衷选择。而在折衷选择时我们有两点假设： 1)需要增加的磁盘空间是可以得到的。2)因盘区被覆盖而受到的性能影响，可以接受。

在前面的计算中，数据库全部回滚段的最小值为 4525KB。一个回滚段的最小可能尺寸为1170KB，是4个回滚段的最小尺寸。

在这样的配置中， 4个回滚段中的每一个都可以一次支持 9个事务(35个事务除以4个回滚段，舍入)。Oracle建议每个回滚段含 4个事务。这样，根据情况差异，从每个回滚段含 6个事务开始，创建6个回滚段(35/6，舍入)。这样就可以确定实际的空间需求。

这六个回滚段，每个都将包含一个用于回滚段头的盘区。表 7-9列出了其盘区分配情况。

这个最终的回滚段含18个大小相等的盘区，合计2250KB。它可以处理绝大多数小型输入项事务，也可以支持型大事务。并且包含有自由空间，以便在事务量或事务负载超过预测值时提供支持。

有关满足这种设计要求的监控细节，请参见第 6章。下面描述设计的后期计算。

表7-9 回滚段的盘区分配

盘区号	说 明	盘区号	说 明
1	回滚段头	10	大型事务使用的扩展空间
2	事务1	11	大型事务使用的扩展空间
3	事务2	12	大型事务使用的扩展空间
4	事务3	13	大型事务使用的扩展空间
5	事务4	14	大型事务使用的扩展空间
6	事务5	15	大型事务使用的扩展空间
7	事务6	16	自由空间
8	不激活、但正在使用的数据	17	自由空间
9	不激活、但正在使用的数据	18	自由空间

7.5.3 确定最佳值

回滚段的optimal值必须适合事务量及管理事务所需的系统开销。这种设计也应当能在一

个盘区中处理大多数事务。

因此，回滚段中的事务数量应按盘区估计。每一个回滚段所需的盘区数量为：

每个回滚段的盘区数 = 每盘区中的小事务数+((长事务的覆盖数 +1) ×平均长事务数)

将它应用到我们的样本数据中便会是：

```
每个回滚段的盘区数 = (33/6) +
                (5+ 1)×1
                =5.6+6
                =11.6
                =12(向上舍入)
```

最长的事务为700KB，将需要6个125KB的盘区。这就要求有5个被覆盖的盘区。由于一次只会有一个这样的事务，所以一个回滚段需要为每一个事务准备一个盘区，并且每一个覆盖上再加一个附加盘区。

使用这么多盘区可以使回滚段既可以处理分配给它们的小型输入项事务负载，也可以提供空间以支持大型事务的平均负载。表 7-9示出了盘区的分配情况。在表 7-9中，从盘区2到7支持6个不同事务的第一个盘区。盘区10~15处理大型事务所需的扩展。

这样，事务数据需要12个盘区。此时需要估计回滚段的系统开销要求。系统开销由三个部分组成。

回滚段开销 = 回滚段头空间 + 未激活但正在使用的(IIU) +自由空间

总是估计回滚段的头占据一个盘区(表7-9中的盘区1)。

IIU空间由为应用程序安排的事务来确定。如果冗长查询与使用相同数据的联机事务同时进行，这个IIU值就应设置得高些。可能出现IIU数据量超过当前使用的事务量。

如果事务分布得合理，就不会发生冗长查询与数据事务同时进行的现象。即使这样，也可能有事务间的重叠现象。这种重叠导致产生 IIU空间，通常至少要占据 10%的回滚段事务量。

对于那个动物园巡回展出的例子，事务间的重叠估计为 15%。

```
IIU空间          = IIU空间百分比×数据盘区数
                = 0.15×12
                = 1.80
                =2(向上舍入)
```

这2个盘区在表7-9中被示为盘区8和盘区９。

最后一个系统开销因素是自由空间。这个自由空间必须能适应最糟糕的事务分配情况。在这种情况中，两个大型事务被分配到同一个回滚段中。

一个大型事务所需要的空间已经在最小盘区量的段计算中予以考虑。因此，只需加上第二个事务出现时的覆盖数量。表7-7中列出的大小为500KB的事务，在盘区大小为125KB时(见表7-8)，将需要4个盘区(3个覆盖)。

```
自由空间盘区 = 需要的附加盘最大数量
          = 3
```

这3个盘区是表7-9中所示的盘区16、17和18。

回滚段的optimal值及optimal存储参数值为：

```
optimal = 每回滚段最小数据盘区数 + 回滚段头盘区 + 未激活但正使用 + 自由空间盘区)×盘区尺寸
        =(12+1+2+3)×125KB
```

```
=18×125KB
=2250KB
```

若将回滚段的动态扩展减至最小程度，就要把 minextents设置为18。这样将预先分配这个回滚段的optimal值。

7.5.4 创建回滚段

现在就可以创建回滚段。不过应当以同样的存储参数来创建这些回滚段。表 7-10列出了动物园巡回展出应用程序的存储参数。

表7-10 回滚段的存储参数

参 数	值
initlal	125KB
next	125KB
minextents	18B
maxextents	249B
optimal	2250KB

所有这些回滚段都在 RBS表空间中创建。因此，可以设置该表空间的缺省存储参数以使回滚段采用理想存储值。用下面的命令来设置这些参数：

```
alter tablespace RBS
default storage
(initial 125K next 125K minextents 18 maxextents 249)
```

如下面的命令所示，在这个表空间中创建回滚段时，只需指定表空间及 optimal值。

```
create rollback segment R4 tablespace RBS
   storage (optimal 2250K);
alter rollback segment R4 online;
```

RBS表空间至少要有可存储6个2250KB回滚段(共13500KB)的空间。当规划它的空间需求时，考虑一下表空间的安排会很有帮助。图 7-25与图7-1、图7-2一起表现了RBS表空间的设计。

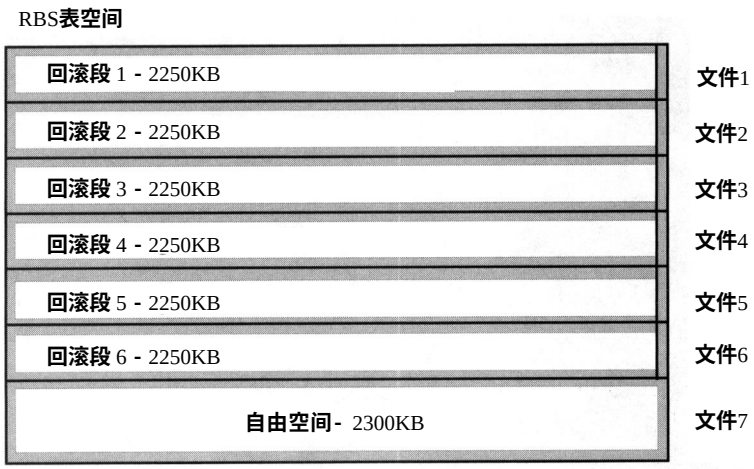


图7-25 RBS表空间的设计

在图7-25的设计中，有6个大小相等的回滚段，还有一个相等大小的附加自由段空间被添加在底部。这个自由空间可用于添加第7个回滚段(如果回滚段头存在冲突问题)或前6个回滚段的临时扩充。

图7-25还表明，这些回滚段都分别归属于各自的文件。在这些文件中，只为系统开销保留少量空间。在图中有7个2300KB的文件，也可以在一个数据文件中存储所有的回滚段。在数据库调整时，使用多个文件就可以有更多改进方法，因为这些文件可以放置在不同的磁盘上来分配事务的I/O负载。

7.5.5 产品回滚段与数据装载回滚段

这里进行的所有计算都是假设应用程序没有办法分配事务到指定的回滚段。回滚段因此必须同时对大、小条目提供支持。虽然这在大多数使用情况中是可以接受的，但在处理数据装载事务时不可接受。

数据装载事务用于管理应用程序中的大量数据。这些事务也许包含有初始化数据装载或从明细表中创建的大型摘要表。无论哪种情况，它们包含的事务量远大于这里的设计。

数据装载不仅要处理大量数据，而且数据装载中的事务更大。例如，当使用 Oracle的Import实用程序时，它的缺省功能是为每个表的数据执行一次提交。要支持这一点，就需要有与该数据表同尺寸的一个回滚段。有关Import的缺省事务尺寸的选择方案，请参见第10章。

数据装载事务必须指定给回滚段。可以使用

```
set transaction use rollback segment SEGMENTS_NAME;
```

命令，或者除了只保留一个回滚段为激活状态外，使所有其他回滚段都变成非激活状态(在非峰值期)。使用本章前面提到的V$ROLLSTAT查询来测量数据装载事务的大小。

一旦数据装载事务被分离到指定的回滚段，这些回滚段就应在 RBS_2表空间中隔离开。如第3章所述，这个表空间只用于那些有临时空间要求的回滚段。把这些回滚段放置在 RBS_2表空间中，使它们能使用RBS_2表空间的自由空间，而无需压缩产品回滚段(在RBS表空间中)的可用自由空间。有关支持大型批事务的讨论，请参见第12章。

其结果便是创建大小正确且已预先分配的产品回滚段。这些回滚段的盘区设计得足够大，以便处理整个事务。而对于那些不常见的特殊事务，用特别分配的空间处理。解决了最糟糕的情况，又得到了最好的情况。

7.6 解决方案

上面几节介绍了为一个样本应用配置回滚段的处理过程。本节描述两种应用的通用回滚段配置：OLTP应用和数据仓库。

7.6.1 OLTP应用

OLTP(联机事务处理)应用支持许多用户执行小型事务。若要支持许多小型事务，应有许多回滚段，每个回滚段要有许多盘区。若要为一个OLTP数据库配置回滚段，必须首先确定数据库中的并发用户数量。如果有一个数据库，就可以查询V$LICENSE视图的Sessions_Highwater列，查看最后一次启动数据库以来达到的最大并发用户数量。

```
select Sessions_Highwater from V$LICENSE;
```

Sessions_Highwater 值是数据库所需回滚段的最大数量。如果回滚段的数量与Sessions_Highwater值相等，每个事务都将有其自己的回滚段。由于数据库并非总是处于最大使用状态(并且OLTP应用中的用户不是总是启用新事务)，所以Sessions_Highwater值除以4便可确定回滚段数量的起始值。如果 Sessions_Highwater值是100，则可创建25个回滚段来支持你的用户。

每一个回滚段最多将支持 4个用户，因此应至少为这些事务创建 4个盘区。另外，需要分配大型事务的空间和 IIU空间。以为每个回滚段分配 20个盘区来开始；盘区的大小应根据应用中的平均事务大小而定。如果平均事务大小为 50KB，就可以用 20个50KB的盘区创建每一个回滚段。需要监控回滚段创建后的情况，以确定是否有覆盖、扩展和收缩发生。

7.6.2 数据仓库/批处理应用

数据仓库支持两种不同类型的事务：由用户执行的小型事务和把数据装入数据库时执行的巨型事务。若支持小型事务，则通过查询 V$LICENSE来确定并发用户的最大数量：

```
select Sessions_Highwater from V$LICENSE;
```

数据仓库一般执行少量事务，所以 Sessions_Highwater值除以10就可确定回滚段数量的起始值。如果有100个并发用户，就要创建 10个回滚段来支持他们。可创建 10个回滚段，每个回滚段有20个盘区，并确定这些盘区的大小以支持用户正在执行的事务尺寸。

对于批处理操作，需要创建一系列独立的回滚段。支持批处理的回滚段一般存储在它们自己的表空间(例如RBS_2，参见第3章和第4章)中。不应使用终端用户的回滚段来支持批数据的装载处理。

在许多数据仓库中，数据装载处理都是串行进行的 —— 一次只出现一个表装载或聚集。如果数据仓库的数据装载处理是串行的，则只需要有一个回滚段来支持数据装载。如果在数据装载处理中有多个大型事务同时激活，就要有多个大型回滚段。数据装载回滚段中应有少量的大型盘区。

尽管可以把一个回滚段表空间的数据文件设置为 autoextend(自动扩展)，但这样就不能控制数据库中的空间使用。如果对回滚段中的最大盘区数设置了限制且表空间的数据文件不能扩展，则任何批事务的最大尺寸都要受到限制。任何试图获得比确定的最大容量更多的回滚段空间的事务，都将遭到失败并促使你重新估算空间。如果将 autoextend选项用于数据文件且不限制回滚段的最大尺寸，一个大型事务就能使用磁盘上的全部可用空间。如果可能的话，可在批数据装载期间对事务的尺寸进行限制。

在一个数据仓库中，通常不在联机仓库用户正在访问数据库时进行批装载处理。因此，在批装载处理时，除了批装载回滚段外全部回滚段都可以设置成脱机状态。如果用户的小型回滚段处于联机状态，则批装载事务偶尔会使用它们，使你在管理数据库其余部分的回滚段空间分配方面遇到麻烦。如果把批数据装载回滚段与用户的回滚段分开，就能高效地管理不同类型的事务的空间需求。