

第五部分 管理Oracle数据库

第21章 管理数据库存储

本章要点：

管理数据库对象

理解数据库碎片

管理回滚段

鉴别存储问题

管理增长的数据库

故障检查

方案：监控数据库增长

21.1 管理数据库对象

一个Oracle数据库有许多消耗磁盘和系统资源的组件。正如你所看到的，每个资源有唯一的存储特性并需要独特的处理和维持。在下面的部分中，你将看到数据库段及它们的唯一存储关系。我要讨论段本身，例如表、簇、索引、回滚段和临时段，以及 Oracle数据库存储它们时所使用的内部方法。

21.1.1 管理Oracle块

Oracle服务器可寻址的最小存储单元是 Oracle块。所有的数据库段都由 Oracle块组成，无论段是一个表、索引、簇或者其他对象，块结构是相同的。设计一个性能最优的数据库要从对Oracle块适当的配置和管理开始。

每个Oracle块都是由下面三部分组成的：

块头。

数据存储区。

自由空间区。

块头包含有关块的信息 什么类型的段数据存储在块中，什么段在块中有数据，块地址以及指向存储在其中的实际行的指针。块头由一个固定部分和一个可变部分组成，在块中块头通常使用85到100字节。

在Oracle块中，管理数据存储区和自由空间区域是彼此直接相关的。数据区域是块中实际存储行的地方。保留的自由空间区是一个区域，被定义为总的可用空间的百分数，被保留用于存储有关在块中的行将来更新的信息。管理数据和保留的块区域是 Oracle块管理所主要关注

的，这将在本章的后面讨论。

21.1.2 理解PCTFREE与PCTUSED

PCTFREE和PCTUSED是应用于段的两个存储参数，它们经常被误解，但是概念实际上非常简单。当Oracle向数据库中写信息时，它必须首先在一个段的分配区中找到一个或更多块来存储信息。Oracle保留了块的一个列表，这些块对每个段来说都是自由的（这也被称为自由表）。Oracle使用PCTFREE和PCTUSED参数的组合确定块何时何时没有足够的空间接受新信息。

提示 在许多情况中，Oracle对PCTFREE和PCTUSED参数的缺省值设置工作得非常好。因此，一直注意并牢记缺省设置的含义就非常重要，特别是要定期地注意行转移。

如果在块中自由空间的百分数比 PCTFREE参数大得多，它就要被添加到段的自由表中以便它可以用来存储新信息（例如，在表中的新行或者索引段的新索引节点）。当自由空间的百分数低于 PCTFREE时，块就被认为是“满的”，Oracle就把它从段的自由表中移出来。当块中使用空间的百分数低于 PCTUSED时，Oracle就把该块加到段的自由表中，这样它就可以再次被用来存储新信息。这种方法允许 Oracle保持足够的额外空间用于行增长，而不需要跨过超过一个块的空间。保持行被限制在一个单独的块中有助于使你的数据库以最高性能运行。

PCTFREE和PCTUSED的值永远都不能等于 100%。如果一个段被这样配置，块很可能将被连续地从自由表中取出，并且在每个数据子处理中又被放置到自由表中。由数据库引擎处理引起的系统开销可以通过在 PCTFREE和PCTUSED之间留至少 20% 的裕量而很容易地得到解决。Oracle 为PCTFREE预置了 10，为PCTUSED 预置了 40说明这个裕量。

PCTFREE和PCTUSED在数据库段的存储子句中被指定。你可以用 dba_tables、dba_clusters 或者 dba_indexes 数据字典视图查询它们目前的值。用 PCTFREE和PCTUSED，你可以微调单个段的存储特性以满足它们特殊的存储需要。例如，一个永远都不会被修改的表可以安全地把PCTFREE设为 0。这在每个块中都节约了空间 对于一个 4KB 的块以及设置为 10 的 PCTFREE来说，每个块中大约有 800 个字节可用于已被保存的数据。对于一个 500MB 的表，执行这个较小的改变，你会节约大约 50MB 的空间。另一方面，如果一个表包含了将来肯定要被改变的列，那么就可以用高一点的 PCTFREE 设置以避免行链接或者转移。行链接和转移在本章的 21.2 节中详细讨论。

图 21-1 显示了 PCTFREE 和 PCTUSED 如何在一起控制块内部的空间使用。在这个例子中假定 PCTFREE 被设为 10%，PCTUSED 被设为 40%。编号 1 显示了新数据可以被加到此块中直到剩余的自由空间不到 20%。这是因为在块总开销后空间的 20% 是留着为以后借助于 PCTFREE 参数进行修改用的。编号 2 显示当达到 PCTFREE 的限制时，剩余的自由空间只能用于对现存数据的修改 在这个块中不允许新的插入。编号 3 显示了由于删除或者修改而被重新放回自由表中的块，并且块中已用的空间下降到 40% 以下。编号 4 显示当已用空间低于 PCTUSED 时，在块中新的插入又是可能的了。当在块中进行插入时，已用空间开始增加，再一次接近上限，循环又开始了。

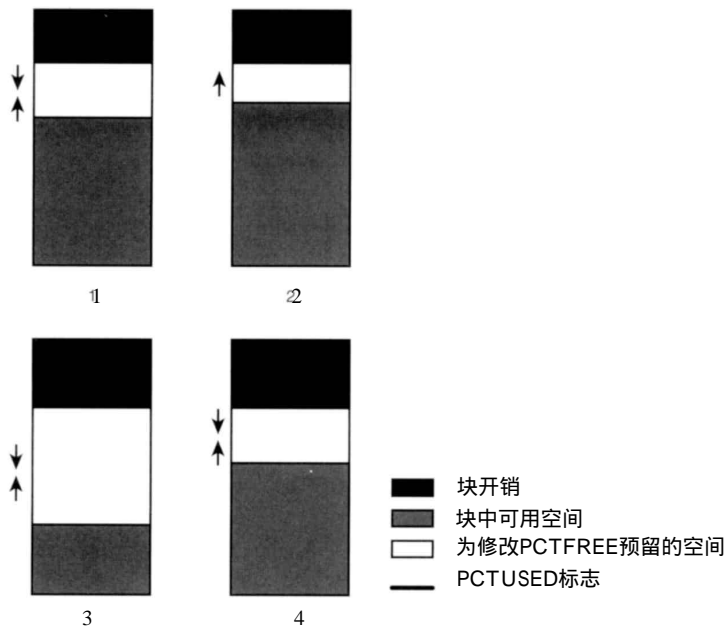


图21-1 PCTFREE和PCTUSED之间的关系

21.1.3 索引与PCTUSED/PCTFREE

索引是值得注意的，因为它们的结构性质，使用 PCTFREE和PCTUSED与表相比稍微有点不同。只有当一个索引最初建立时这些参数适用，在以后的修改期间没有更多的用处。因此，PCTFREE的设置应让索引结构最初在叶块中建立时就有足够的空间，这样就可以增加辅助关键字，而不必重复地拆分已存在的叶块提供给新关键字。由于在索引的建立期间不会删除行，因此索引就不能从任何 PCTUSED设置中获益。

提示 当为一个已存在的表建立新索引时，如果预料到表在近期将大量增长，就要考虑在建立索引时指定一个低的PCTFREE值。

21.1.4 管理表存储区

表容量（是行长 × 行数的函数）可能是确定存储需求的最好尺度。表的空间需求大致是平均行长与行数的乘积。当数据库管理员讨论他们所关心的数据库的近似容量时，行数经常被用做尺度。尽管它不会为计划长期存储需求提供许多信息，但它确实会使你了解某些有关数据库容量值的知识。

表通常是大多数存储管理的起始点；它们充当着一个排序标准，用来确定其他大多数数据库组件的存储需求。由于这个原因，理解包含在数据库表中的数据性质（数据类型、空列的数目等等）对数据库管理员是有利的。

表的容量由于新行的插入或者存在的行被修改为更大的值而增加。行的平均长度与行的数目是影响表容量的主要因素。

确定表的容量是一个概念上的任务，并且应该在表建立之前的设计阶段完成。问题，诸如预期的增长和存储参数值等问题应该在决定容量参数时处理。

考虑下面CREATE TABLE的例子：

```
create table employee (                                /* PART 1 */
  emp_no    char(4),
  emp_name  varchar2(30),
  age       number,
  dept_id   char(4)
)
  tablespace EMPLOYEE_TS                                /* PART 2 */
  storage ( initial 10M
            next    10M
            pctincrease 0
            minextents 1
            maxextents 50
            freelists 1)
  pctfree 10;
```

这个CREATE TABLE语句由两部分组成：第一部分是必须遵循的表定义，在这里指明关键字的属性，例如列、数据类型和约束条件。第二部分处理涉及到表的容量问题，这是你感兴趣的。让我们看一下对这些容量参数的说明：

TABLESPACE 表空间就是表要在其中创建的地方。如果省略了，表就会建立在拥有者的缺省表空间内。

INITIAL 给表分配的最初的区间容量。

NEXT 如果最初的区间已经完全被表行填满了，那么该参数就是给表分配的下一个区间容量。

PCTINCREASE 超尺寸的表的第三个（如果 minextents被设置为1）和随后的区间的百分数。如果最初建立时以一种方法“懒洋洋地”处理扩展的表，这个参数很快会给粗心的数据库管理员制造使用期限的困难。例如，一个 PCTINCREASE为50（缺省值）的表，INITIAL设置为10M，10M 的NEXT有第一个10M容量的区间，第二个10M的区间，第三个15M的区间（ $10M \times 150\%$ ），第四个22.5M的区间，等等。计算第11个区间的容量，你会看到值0或者1被推荐给这个参数。

MINEXTENTS 在建立时分配的区间数。对表来说，这几乎一直是 1 如后面在21.3节中讨论的那样，回滚段有不同的需求。

MAXEXTENS 表能分配的最大区间数。尽管版本 7.3和以后的Oracle RDBMS使你能够无限制地分配区间，但是如果只是对一个对象胡乱地扩展进行报警的话，给这个参数赋一个值还是有用的。

PCTFREE 在段的每个块中为现存的行将来发生改变预留的空间总数。

PCTUSED 到了这一点时，块将被放回到自由表中。PCTFREE和PCTUSED在前面部分中已经讨论了。

FREELISTS 指定要被存储到表、索引或者簇的自由表组中的自由表数。其缺省设置为1，并只在表中的一个块同时可能有许多修改时增加。

21.1.5 管理索引

索引可能是段的类型中最难管理的。许多应用程序员（以及数据库管理员）试图通过申请更多索引的办法解决数据库的性能问题。记住，更多的索引几乎总是损害修改性能，并且可能不会一直提供比较好的SELECT性能。在索引上保持关闭标志并删除任何不用的索引。

提示 用Oracle的EXPLAIN PLAN功能确定增加的索引是否对查询有所希望的影响。

许多不需要的索引会溜进你的数据库并引起性能问题。Oracle几乎总是宁愿要性能而不要节省磁盘空间。把每个没有主键的索引与索引所应用的应用程序以及建立这个索引要解决的特殊功能或者问题做一个记录。对几个为特殊应用程序建立的索引来说，没有什么与众不同的；如果没有记录，你会发现不再需要这个索引了。

21.1.6 监控回滚段

回滚段与诸如索引和表的其他段相比有点不同。因为回滚段实际上是动态的，数据库管理员通常应该集中精力保证每个回滚段及与之联系的表空间有足够的空间用于增长，而不需要使用太多的区间。

21.1.7 监控临时表空间和段

和回滚段一样，Oracle数据库中的临时表空间需要专门的监控。临时段仅在创建该段的事务处于活动状态时被保存。因此，高水位标志是你的度量尺度，而不是在某点有多少空间正被使用。目标是要确保系统要处理的最大事务有足够的空间。

21.2 数据库碎片

数据库管理员面对的一个最常见问题就是处理数据库对象的碎片。碎片浪费空间，导致性能问题，并给数据库对象的管理带来更大的困难。数据库碎片是许多问题，而不是一个问题。包括分裂成碎片的数据库对象，分裂成碎片的表空间、链接行和转移行。

提示 数据库管理员尤其应该注意数据库应用程序是如何执行任务的。开发者通常建立临时表或在程序结束之前便被删除的数据行。这实际上会导致碎片而不留下太多的痕迹。数据库管理员应至少了解一下数据库应用程序是如何设计的。

数据库碎片通常是行被插入、修改和删除以及对象被建立和删除的结果。因此，一个新建立的数据库就不会有碎片。只有当在数据库上运行了一个或更多的活动应用程序后，才开始看到数据库碎片的迹象。

在下面几节中，你会看到四种数据库碎片类型以及调整数据库并阻止碎片发生的简要方法。

21.2.1 分裂成了碎片的表空间

表空间变成碎片是由于错误以及表空间中的数据库对象的无计划撤消和重建造成的。当自由空间气泡在表空间的已用区域中被分离时产生表空间碎片。这在属于表空间的对象被删除时发生。这些被分离的空间气泡可以被重新使用，但是只有在一个对象被建立且能装入以前的对象留下的空间中时发生。

注意 表空间不会由于任何行级动作而变成碎片。这是常见的，但是完全错误的误解。

例如，假定在一个表空间中有四个表：customer、employee、stock和company。这些对象中的每一个都由一个10MB的区间组成。表空间的结构看起来如图21-2所示。注意表空间末尾10MB的自由区间，这些对象压缩得很紧。这就是目的，是你应该达到的。

当然，你不可能让这种状况持续太长时间。假定由于性能需要，你必须把 customer表移到另一个表空间中。表空间现在看起来就像图 21-3。注意表空间中10MB的空的部分。如果你需要建立一个长度小于或等于 10MB的另一个对象，没有问题。但是假定你需要在这个表空间建立一个policy表，其存储参数为15MB的初始区间、10MB的下一个区间。这个表空间总共有20MB的自由空间，但是任何自由区域的最大长度只有10MB；因此创建policy表的努力就失败了。

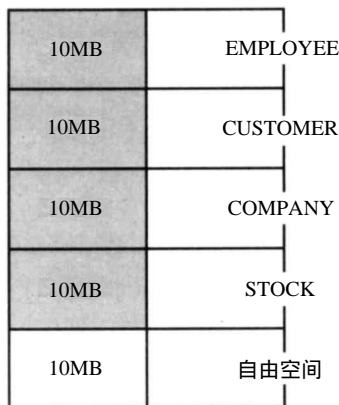


图21-2 分裂为碎片前的表空间结构

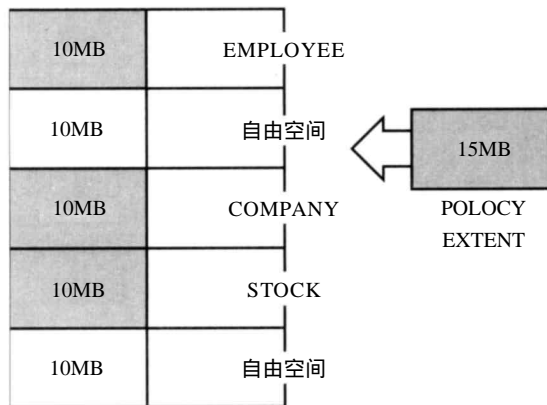


图21-3 在删除一个表之后表空间中的碎片

这个例子显示了一个小规模的问题，在表空间中只有一个对象被删除了。考虑一个繁忙的实际环境，在这里对象被删除以及不同大小的对象被建立几乎是连续不断的。这个动作导致组成表空间的自由空间的非连续气泡——与图21-4中描述的类似。表空间的总自由空间看起来是足够用的，但是在表空间中没有一个大的连续自由空间区域，这会严重地限制你使用自由空间的能力。

Oracle RDBMS尽力帮助数据库管理员进行空间分配和管理。考虑一下表空间中四个表的相同情况。现在假定你删除其中的两个表，这两个表都位于表空间的“中间”位置。这就给了你一个如图21-5所示的表空间结构。如果对一个15MB表的相同需求被执行了，SMON后台进程自动地把两个10MB的区间聚合为一个20MB区间，用它实现对象的建立。如果表空间的PCTINCREASE被设为非零值的话，SMON也可以自动地聚合空间。

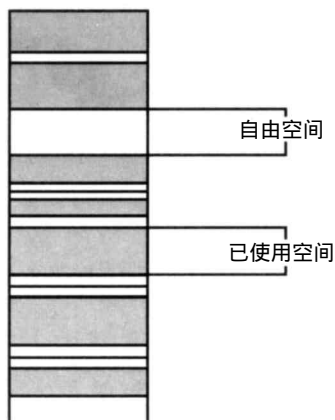


图21-4 一个完全分裂的表空间

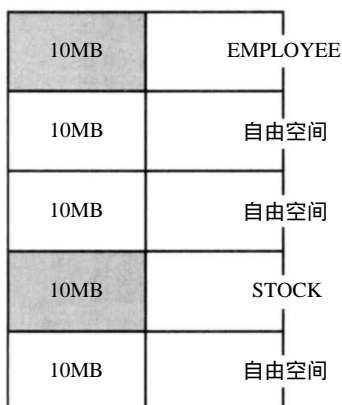


图21-5 在删除两个表后，表空间中的碎片

你也可以用下面的命令在表空间中人工聚合空间：

```
alter tablespace armslivedb01_ts coalesce;
```

下面的脚本建立了一个视图，你可以用它来识别表空间中有问题的段。用 `tablespace_name` 查询这个视图以获得一个空间问题会发生在哪儿的概念，也可以查看一下分裂成碎片的单个对象如何存在于表空间中。

```
CREATE OR REPLACE VIEW ts_blocks_v AS
SELECT tablespace_name, block_id, bytes, blocks, '== free space ==' segment_name
FROM dba_free_space
UNION ALL
SELECT tablespace_name, block_id, bytes, blocks, segment_name
FROM dba_extents;
```

```
Select * from ts_blocks_v;
```

TABLESPACE_NAME	BLOCK_ID	BYTES	BLOCKS	SEGMENT_NAME
PROG_PLAN_IDX_08	34562	221962240	27095	== free space ==
BMC_SMG_T	339	102088704	12462	== free space ==
DBA_TEST	42372	533700608	65149	== free space ==
BPW_DATA_01	111667	133808128	16334	== free space ==
BPW_IDX_01	155439	299515904	36562	== free space ==
AUDIT_01	84231	40960	5	SQLAB_COLLECTION
AUDIT_01	2	1064960	130	TB_225AUDIT
AUDIT_01	2692	532480	65	TB_237AUDIT
BPW_DATA_01	105737	41943040	5120	TB_G026CAPACITY
BPW_DATA_01	105702	286720	35	TB_G016RESOURCE
BPW_DATA_01	105667	286720	35	TB_G006NSU
BPW_DATA_01	105602	532480	65	TB_7056SUBST_AUDIT
BPW_DATA_01	105567	286720	35	TB_64S6MSG_TYPE_CODESET
BPW_DATA_01	101982	29368320	3585	TB_64R6CAPACITY_CONSTRAINT
BPW_DATA_01	1092	286720	35	TB_5317INV_RI

表空间碎片会导致下面的问题：

表空间中的空间被分离且不能有效地使用。

在需要重建分裂成碎片的对象时会导致管理问题。

注释 在某种程度上，在RDBMS和磁盘技术上的进展已经把分裂成碎片的表空间对整个系统性能的影响减到了最小。然而，仍然需要减小表空间碎片，以便最优化系统性能并使磁盘空间得到最有效使用。

提示 检查数据库碎片时，一定要检查行链接或者转移。这两个问题很容易被忽视，并导致可测的性能问题。

21.2.2 处理分裂成碎片的表空间

处理表空间碎片的最好方法是避免它。这可以做到，但是需要仔细地计划和管理的额外开销。如果你面临着整理分裂成碎片的表空间，最简单的方法是在表空间中导出损坏了的对象，删除该对象，再把它们导入回来。这不仅能够聚合你的自由空间，而且导出操作也可以把你的数据库对象聚合到一个区域中。

要避免碎片，有以下几种方法：

把对象用相似的空间和增长特性聚合在一起。

如果可能，把它们的区间设成相同的容量，这样所有的对象可以共享释放或者从删除对象中回收的区间。

使每个段的PCTINCREASE保持为0。

按照这些提示做会消除所有用不同大小建立的以及将来的数据库段不能使用的区间。

注意 从Oracle 7.3开始,在表空间上设置缺省的PCTINCREASE比0大会导致SMON自动聚合相邻的自由区域。除非你的数据库系统绝对不会分享CPU时间或者磁盘输入/输出,否则你应该把这个参数至少设为1。

最后,下面的脚本向你展示了在你的数据库的每个表空间中有多少自由空间是可用的和最大自由区间的容量以及每个表空间中自由区间的数目。在查找问题时这是一个好的起始点——如果你在表空间中看到了500MB的自由空间,但是只有一个最大15MB的自由区间,那么就明显有问题。

```
SELECT tablespace_name, sum(bytes) "Free Bytes", max(bytes) "Largest Extent",  
       count(block_id) "# Extents"  
FROM dba_free_space  
GROUP BY tablespace_name;
```

21.2.3 对象碎片

分裂成碎片的表空间的确会使任何数据库管理员对空间分配感到头疼。潜伏的和容易被忽视的碎片问题发生在对象级。在对象级碎片中,对象在表空间级可能看起来很好并很健康,有一个单独的区域,但是在那些区域内部的某些检查可能会揭示出问题。

一个常见的情形是数据库管理员在一个经常改变的表空间中发现空间分配问题,并重建了表空间和它的所有对象。随后,数据库性能提高了。这种在性能上的提高是消除表空间碎片的结果,但也可能是消除对象级碎片带来的性能的提高。

提示 虽然技术上没有碎片,但许多DELETE操作会产生很少被填充的块,这些块还不满足PCTUSED的被放回到自由表的条件。大部分打算处理对象碎片的数据库的操作也会处理稀疏的块问题。

对象碎片会导致下面的问题:

对数据库额外的读调用会导致响应时间的增加。

由于在表和索引块中的自由空间空洞导致空间的浪费。

读性能下降。因为数据不再被紧紧地排在一起,物理磁盘驱动器必需从一个比需要的大磁盘表面区域中搜寻和读数据。

对象碎片是三个特殊碎片类型的“保护伞”:行转移、行链接和过分扩展的段(对象有大量的区间或者几乎接近maxextents参数)。我在下面几节中讨论这些碎片类型。

21.2.4 行转移

行转移在对行的修改引起行的长度比块中可利用空间大时发生。当发生行转移时,行的一部分就转移到一个新的数据块中。这个新单元的地址存储在原始行单元中。当Oracle需要读这个行时,它首先读原始单元并找到一些数据和指向另一个块的指针,再在这个块中找到剩下的行数据,见图21-6。

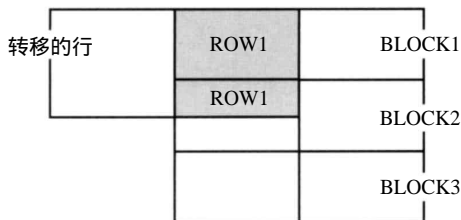


图21-6 块中行转移的例子

行转移存在两个问题：

Oracle每次必须执行至少一次额外的输入 / 输出读来获取转移的行。它必须读每一个包括行数据的一部分或者是指向另一个包含着行数据的块的指针的块。

Oracle也必须和行数据一起存储额外的指针来供给行转移机制。但这通常是不重要的，它浪费了一些磁盘空间。

注意 为什么Oracle执行行转移而不是把整个行移到一个新块中？移动一个行的单元是非常昂贵的，而作为行ROWID的锁定（ROWID与它的物理位置紧密相连）被存储在数据库的索引和其他区域中。对每个ROWID，Oracle必须对ROWID的所有引用有一个反向索引，这样它们就能被有效地修改。无疑这会导致相当大地全系统的额外开销，而这是不必要的。

21.2.5 行链接

行链接在一个行太长以致于不能放入任何一个数据块时发生。这导致该行被存储在一个或多个数据块的链中。行链接经常伴随着包含 LONG、LONG RAW 或者LOB数据类型的大行发生。参见图21-7了解有关一个表中链接行的更多信息。

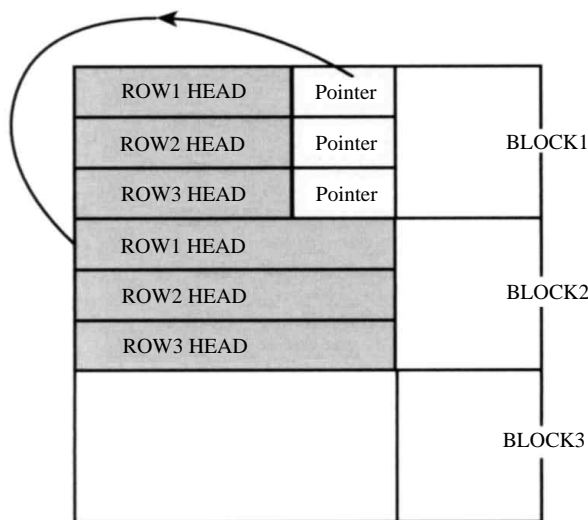


图21-7 行长度比块尺寸大导致行链接

21.2.6 消除链接和转移的行

链接和转移的行会对你的数据库的整体性能造成一系列的问题。你可以通过发出下面的SQL语句检查链接和转移的行：

```
analyze table customer list chained rows into chained_rows;
```

要建立chained_rows表，执行\$ORACLE_HOME/rdbms/admin目录下的utlchain.sql。当你运行了utlchain.sql，一个被称为chained_rows的表就建立了，它就是该SQL语句所要达到的目的。

这个命令把ROWID和数据库中所有链接和转移行的表存储到 chained_rows表中。直接查询chained_rows表，找出行和表的特定信息。

因为转移行仅当一个行被修改为一个新值时发生，你可以通过执行下面的步骤消除所有转移行：

1) 使用如下的SQL语句建立一个临时表来保存转移行：

```
CREATE TABLE temp_emp AS
SELECT * FROM emp WHERE rowid in
(SELECT rowid FROM chained_rows
WHERE table_name = 'EMP');
```

2) 从主表中删除前一语句存储的行：

```
DELETE FROM emp WHERE rowid in
(SELECT head_rowid FROM chained_rows
WHERE table_name = 'EMP');
```

3) 从临时表中插入行：

```
INSERT INTO emp SELECT * FROM temp_emp;
```

这个过程消除了所有转移行。通过删除在 chained_row中引用的行并重新运行 analyze语句，你一定能识别所有链接的行。

这个解决方案的一个问题（特别是对大表而言）就是你浪费了老的数据块中所有的空间，这些链接的行原来就保存在这些块中。插入的行不适合那里，这样所有删除的行都在表中留下了自由空间块，这些块也就是以前的行被放置的地方。这种行压缩类型有两个主要的缺点：

表中的许多自由空间可能永远都没用过，这取决于 PCTUSED的设置，因此空间被浪费了。

当这个表通过全表扫描被读取时，因为必须读非常多的块，所以可能会损失一些性能（尽管这通常不一定发生）。

可替代它的方法是把数据从表中存储到一个临时表中，截断表，把数据插入回去。你也可以使用导出，删除表，之后使用导入重建该表。后面的解决方法是最简单、最全面的，但也是最消耗时间的。

在尽力避免行转移时，确定 PCTFREE的值很重要。你可以用一个粗略的推算得到 PCTFREE，该推算假定表有一个平均的行长 x字节，y字节在最初插入后将被修改。有了这些必要条件，使用如下公式取得 PCTFREE：

$$PCTFREE=100*y/x$$

例如，如果你有一个行长为 100字节的表，21字节稍后将被修改，PCTFREE的值是 $100 \times 21/100$ ，即21%。

行链接是非常难调整的。唯一的解决办法是缩短表的行长或者增加数据库块的大小。选择前一种方法通常需要对表结构进行重新设计，但是后一种方法需要导出整个数据库，重新创建然后全部导入。这两个方法都不是很吸引人，但是如果一个表中的链接行引起了性能问题，那么就必须使用它们。

提示 在采取任何强有力的步骤消除行链接以前，看一下损坏表中的数据类型。Oracle中的一些较新的数据类型（例如 VARCHAR2）比它们以前的对应类型（CHAR）更有空间意识，并且几乎不需要改变应用程序。

21.3 管理回滚段

数据库回滚段的管理可能比数据库中其他段的管理要复杂。因此，在你为这些有些麻烦的对象考虑策略前理解回滚段操作的复杂性很重要。

回滚段被用来提供读一致性（在本章后面解释）、事务回滚和数据库恢复（在第24章中讨论）。它们存储一个事务中涉及的数据的前像（before image）。如果事务希望取消所做的改变，回滚段中的信息可以把数据库恢复到以前的状态。这被称为事务回滚。

像任何其他段一样，回滚段被定位在表空间中并且包含区间。每个数据库有一个SYSTEM回滚段，它位于系统表空间中并在数据库建立时建立。普通用户不允许使用它并且它不能被脱机使用或者删除。你应该为专用于回滚段的表空间中数据库的正常工作建立附加回滚段。回滚段的扩展和压缩是很普通的，它们不是你在表、索引或者系统表空间中所希望发生的行为。

必须调整回滚段以便处理并行事务以及不同大小的事务。通常，在有許多并行事务的系统上你需要有更多区间的更多回滚段。你的事务越大（例如，修改大量的行），你的回滚段也就需要越大。因为回滚段通常允许增长，然后收缩回存储参数OPTIMUM的值，一定要特别小心以确保你真正知道一个回滚段需要多少空间。

提示 Oracle的“快照太旧”错误信息通常是由太少和/或太小的回滚段引起的。

21.3.1 回滚段操作

事务或者是明确地申请或者是被隐式地指定到一个特殊的回滚段中。多个事务能够并行写入任何给定的回滚段或者该段中的区间。然而，块专用于某个事务。Oracle用循环的方式使用存在的区间和块分配给一个回滚段。根据需要（并且在可用的存储参数允许的范围内），新区间被分配给当前的事务。当事务结束时，它们的回滚信息不被删除（记住，回滚段是循环写的）。这是为了容纳其他已经存在的事务，这些事务可能仍然依赖于已经变化的用于数据库读一致性视图的数据的老版本。

注意图21-8对这个回滚段扩展性的描述。

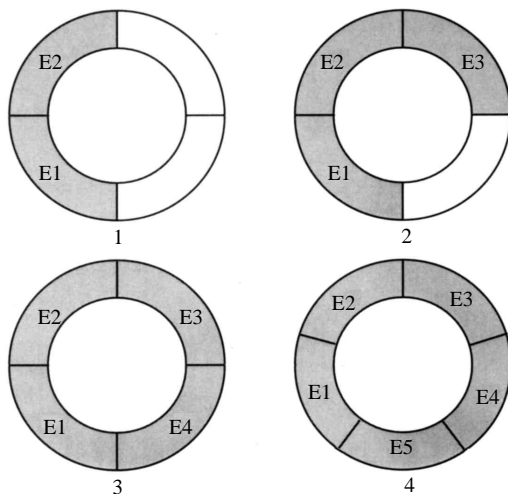


图21-8 回滚段中区域的分配

在图21-8中，图1到3显示Oracle使用已存在区间的顺序；图4显示Oracle增加一个区间到回滚段中。图1显示Oracle活跃地使用区间E1和E2，同时E3和E4被空闲。图2和3显示Oracle使用剩余的区间处理事务回滚信息。图4显示Oracle分配另一个区间E5来容纳活动事务。除非Oracle需要分配另一个区间给活动事务，否则下一个被用到的区间又将是E1。

注意 如果达到了一个回滚段的maxextents，或者表空间中没有更多的空间分配给回滚段的话，扩展就失败了。

回滚段的动态扩展是一个代价很高的操作，并且会导致性能干扰。如果回滚段没有小心地放置在整个数据库的结构中，它也会引起空间问题。应该给回滚段分配为回滚段保留的表空间。包含在任何给定的表空间中的所有回滚段应该被精确地分配成相同的大小——因为回滚段会频繁地扩展和释放区间；如果设置了不匹配的区间大小，那么前面讨论过的碎片问题确实是个问题。

提示 有特殊设计的大小的特殊回滚段是为特殊事务类（例如DSS和OLTP）创建的。因为这些是将被设置为不通用大小的回滚段，它们应该存储在它们自己的表空间中以使碎片减到最小。

这里有一些有关回滚段中区域的扩展和分配的一般概念：

回滚段中单独的区间可以被多个事务写入，但是区间中的每个块必须只包含一个事务的信息。

不管系统中存在多少回滚段，一个事务只能被分配给一个回滚段。

除非Oracle明确地申请某个回滚段，否则它使用循环方式来确定事务将使用的回滚段。尽管它试图使每个回滚段处理相同数目的事务，然而要猜想什么回滚段将要分配给事务变得很困难。

分配的区间在回滚段中按照块ID的升序顺序使用。例如，如果回滚段中第五个和第七个区间是自由的，那么第五个区间就总是比第七个区间早分配。

每个回滚段（和大多数段一样）必须最少有两个区间。

21.3.2 设定回滚段大小

当设定回滚段大小时，记住它们应该被设置得足够大，这样系统就不必马上重新使用刚被一个完成的事务释放的块。其他的并行事务可能仍然需要使用回滚段信息，这个回滚段信息是由修改/插入/删除事务产生的以完成数据库一致性读视图。

考虑一个长时间运行的报告，这个报告给出关于一个公司中所有雇员薪水的详细情况。这个报告遍历Employee表的每一个记录，并包含与之有关的其他表的信息。假定一名HR雇员在下午3:00发出报告。在下午4:00，薪水册中的另一个用户决定雇员薪水的年增量应该被处理（这是一个部门间很少联络的明显例子）并执行这个维护的工作。这个进程修改了公司中大部分雇员的记录，并马上提交了这个信息，参见图21-9。

提示 回滚段的使用受到事务运行的某种类型修改动作的影响。通常，DELETE消耗大部分的回滚空间，INSERT消耗的回滚空间次于UPDATE

其间，原来的HR报告仍然在变化着。它到了紧急时刻，也就是需要检索的雇员记录的薪水已经被薪水增加进程改变了。Oracle此时应该做什么呢？报告应该发出被提交的以及当前的

薪水详情，还是应该返回增加之前记录的薪水？

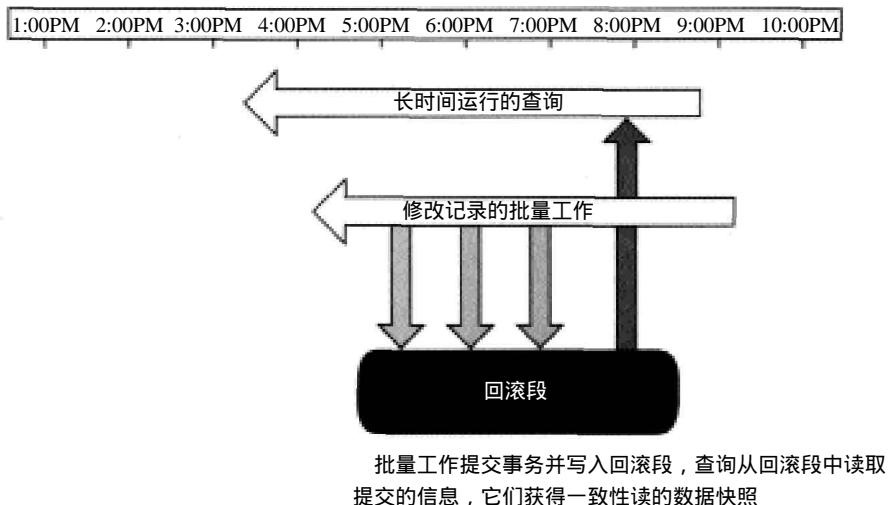


图21-9 长时间运行的查询与批量工作

如果报告得到当前的信息，那将意味着报告中的一些雇员拿到了增加前的薪水，一些拿到了增加后的薪水，这就会使报告前后矛盾并且没有用处。这就是回滚段的重要性显现的地方。通过Oracle的事务处理机制，以前修改的在增加薪水的处理程序中改变的所有行的映像被写进了回滚段。当报告程序到达一条已被修改的记录时，它会用块控制信息（SCN、或者系统改变号）确定需要注意回滚段以便找出报告开始运行时的信息。

注意，对这种情况所做的适当处理在事务有足够的回滚空间时是完全可信的，这样，写入回滚段的信息就不会被随后的事务重写。如果事务需要的信息被重写了，那么你会得到一条“all too common and hateful Snapshot Too Old”的错误。

前面的例子是在设置回滚段容量时要考虑的一个方面。从讨论中看到，很明显在回滚段中应该有足够的空间来存储这些非活动的或者被提交的数据事务，这些数据事务可能是其他长时间运行的查询所需要的。

你可以通过采取下述措施避免长时间运行的查询的问题：

- 分配足够大的回滚段空间以容纳最有可能的事务并行情况。

- 采用调度法，由此长时间运行的查询与小的 OLTP 事务会在不同时间里运行。

- 建立一个只被长时间运行的批量工作使用的大回滚段。

提示 使用SET TRANSACTION USE ROLLBACK语句在事务开始时明确地使用某个回滚段。

```
set transaction use rollback segment RB1;
```

21.3.3 使用OPTIMAL参数

当建立回滚段时，回滚段的目标容量可以用存储参数 OPTIMAL定义。OPTIMAL参数指定了回滚段要缩小到的尺寸。

让我们看一个create rollback segment 语句的例子：

```
create rollback segment r05
```

```
tablespace rbs
storage ( initial 10M
          next      10M
          optimal 20M
          minextents 2
          maxextents 100);
```

RDBMS尽力把回滚段的尺寸维持在指定的最佳值 20MB上。这个尺寸被解释为上对齐到区间的边界，意思是RDBMS尽力保持最少的区间数，这样总尺寸就大于或等于 OPTIMAL指定的尺寸。

无论何时，回滚段开始写入另一个区间时，Oracle检查段是否比OPTIMAL的尺寸大，如果大的话，那么就检查下一个区域是否完全空闲的。如果两个检查都是正确的，Oracle就释放该区间并继续进行下一个区间（这个区间如果满足相同的标准也会被释放）。当试图释放非活动区间时，最早的一个首先被释放。

OPTIMAL参数在某些方面被认为是存储参数PCTINCREASE的兄弟——懒惰的数据库管理员的管理方法。尽管它确实使回滚段的管理稍微简单一些，但是证明也会做很多有害的事情。

警告 如果发现你的回滚段由于OPTIMAL子句的缘故持续地收缩，那么很可能你的回滚段被不适当地设置了尺寸。如果这种区间的分配和释放很常见，那么它会影响到性能。

21.3.4 进行装载测试以获得回滚段估算

为了正确地设置回滚段的尺寸，你需要对将由较大事务产生的撤消信息的数量有个了解。通常，每个回滚段都应该足够大以适应最大事务回滚需要，如果不够大的话，它们就会无节制地浪费空间（此情况的唯一例外就是如果你有前面讨论过的专用回滚段）。为了对撤消产生的尺寸有个合理的估计，在样本事务中执行一个装载测试，使用清单 21-1，这个样本事务会产生大量的撤消信息。

提示 如果你不知道某个作业正在对数据库发出什么SQL语句（并且没有可用的源代码）的话，那么考虑使用Oracle的跟踪工具来捕获准确的要进行调整的SQL语句。

在你正在执行测试时，要确保没有其他活动事务产生撤消信息。下面的步骤产生一个装载测试：

- 1) 建立两个表保存你的统计信息：stat\$undo_begin和stat\$undo_end。
- 2) 假定装载测试在回滚段 R01中执行。在与数据库连接后，发出一系列事务处理命令，以确保所有属于此事务的撤消信息被写入回滚段 R01。
- 3) 使用v\$rollstat表把写入回滚段R01的原始数据捕获到表stat\$undo_begin中。
- 4) 发出测试事务的语句。
- 5) 在事务处理结束后，再把写入回滚段 R01的数据放到表stat\$undo_end中。
- 6) 既然你在表中已经有了开始和结束值，只需将这两个值相减便可得到写入回滚段 R01的数据。这些值可以用于确定你的回滚段的尺寸需求。

清单21-1是执行上述步骤的脚本。

清单21-1 执行装载测试以获得回滚段尺寸的估计值

```
REM
REM Create the set up tables
----
```

```
REM
create table stats$undo_begin (writes number);
create table stats$undo_end   (writes number);
REM
REM Capture the initial number of write for rollback seg R01.
REM Replace R01 with the name of your rollback segment.
REM
insert into stats$undo_begin
  select sum(writes) from v$rollstat
  where usn = ( select usn from v$rollname where
                name = 'R01');

REM
REM Alter the current session so that it uses the test rollback
REM segment only. Replace R01 with the name of your rollback segment.
REM
commit;
set transaction use rollback segment R01;
REM
REM Run the long running batch job which generates largest undo data.
REM Replace the name of your batch job script with 'large_batch_job'
REM
@large_batch_job;
REM
REM Capture the writes generated at the end of the batch job.
REM Replace R01 with the name of your rollback segment.
REM
insert into stats$undo_end
  select sum(writes) from v$rollstat
  where usn = ( select usn from v$rollname where
                name = 'R01');

REM
REM Get the amount of data that has been generated as redo for the
REM transaction under test
REM
select (e.writes - b.writes) undo_generated
  from stats$undo_begin b, stats$undo_end e;
UNDO_GENERATED
      898998
```

在清单 21-1 的输出中，由批处理工作产生的撤消信息有 898998 个字节。这个脚本可以用作适当地设定回滚段尺寸的起点。

21.4 鉴别存储问题

当数据库启动和运行时，有时你需要再次查看数据库的每个主要组件有多大。最关心的是要确保表空间足够大，能够存储它们所包含的段。表、索引与簇随着时间的推移都会增加，你需要确保 INSERT 和 UPDATE 在生产期间不会因为表空间超出空间之外而失败。

当我讨论为段分配的空间时，我指的是为段预留的空间。这并不表示段实际上正在使用所有的空间（尽管，它几乎总是使用空间的一些部分）。例如，假设表 ABC 在两个分别为 50MB 和 40MB 的区间中被分配了 90MB 的空间。表 ABC 实际上正在使用 50MB 和 90MB 之间的空间。当第一个区间被用尽了，表定义中的 NEXT 参数指示分配另一个 40MB；即使实际上只需要 10KB，也会这么做。ABC 不需要再去找更多的空间，直到剩余的 39.99MB 已被填满。尽管这种设计对一些不是从主机数据处理环境来的数据库管理员来说有些古怪，但它在 Oracle 的性能特性中是一个关键的要素。

使用Oracle企业管理器的数据库管理员会发现访问数据库的并行存储状态的工作比它们相对应的命令行方式简单。存储管理器是企业管理器集的一部分，它为大部分普通的有关存储的信息需求提供了GUI接口。图21-10显示了存储管理器的一个示例窗口。

你的数据库不必驻留在 Windows NT机器中来使用存储管理器。Oracle的Net 8网络组件使运行存储管理器的 Windows系统能够和远程机器上的数据库一起工作。远程机器可以驻留在本地或者广域网上，并且可以是任何 Oracle支持的平台。

有经验的数据库管理员会发现存储管理器非常像基于 Windows的Oracle GUI工具，它只是缺少几个用于有效管理存储的重要功能。存储管理器是一个非常有用的工具，它可以简化日常工作，但是它不能替代命令行界面。

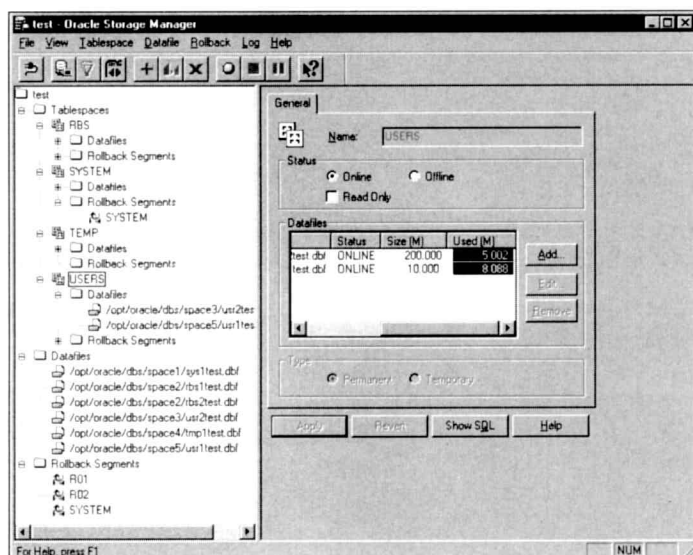


图21-10 Oracle存储管理器的例子

21.4.1 开发表空间

数据库表空间的开发通常包括解释表空间的总尺寸以及表空间使用什么样的数据文件。DBA_DATA_FILES表包含所有关于数据文件的信息。因为表空间的尺寸是它的数据文件的总和，所以你也可以从DBA_DATA_FILES表确定表空间的尺寸。

这里是一个例子：

```
COLUMN TABLESPACE_NAME FORMAT A10
COLUMN FILE_NAME FORMAT A35
SELECT TABLESPACE_NAME, FILE_NAME, BYTES, STATUS FROM DBA_DATA_FILES
ORDER BY TABLESPACE_NAME;
TABLESPACE FILE_NAME                                BYTES STATUS
-----
RBS          /u02/oradata/test/rbs1test.dbf  20971520 AVAILABLE
RBS          /u02/oradata/test/rbs2test.dbf  20971520 AVAILABLE
SYSTEM       /u01/oradata/test/sys1test.dbf  20971520 AVAILABLE
TEMP         /u04/oradata/test/tmp1test.dbf  10485760 AVAILABLE
USERS        /u05/oradata/test/usr1test.dbf  20971520 AVAILABLE
```

你看到数据库中每个数据文件占一行，该行显示了它所属的表空间、它的尺寸（用字节

表示)以及状态。状态域几乎总是 AVAILABLE。

应经常检查表空间以便确定在段需要增长的情况下有很多可用的自由空间。这个问题不像初看上去那样简单。为了回答这个问题,你需要知道下面的事情:

表空间有多大。

总计还有多少自由空间。

最大的自由区间是什么。

是否任何一个段都可以至少可以通过一个区间增长。

表空间如何分裂成碎片。

你需要从知道一个表空间有多大开始。把与数据文件相关的尺寸合计起来可以很容易地为你提供那些信息。例如:

提示 通常,可以很容易地编写一个自动搜集有关每个数据库存储信息的脚本,并把该脚本发送给适当的数据库管理人员。

```
SELECT TABLESPACE_NAME,SUM(BYTES) FROM DBA_DATA_FILES GROUP BY TABLESPACE_NAME
```

```
TABLESPACE SUM(BYTES)
```

```
-----  
RBS          41943040  
SYSTEM       20971520  
TEMP         10485760  
USERS        20971520
```

下一个问题是确定表空间中总计还有多少自由空间。Oracle在DBA_FREE_SPACE表中保存一张数据库中所有自由区间的清单。通过合计表空间中自由区间的尺寸,你就会看到还剩多少自由空间。下面这个例子显示了数据库中每个表空间的自由空间数。

```
SELECT TABLESPACE_NAME, SUM(BYTES) FROM DBA_FREE_SPACE  
GROUP BY TABLESPACE_NAME;
```

```
TABLESPACE SUM(BYTES)
```

```
-----  
RBS          29343744  
SYSTEM       11552768  
TEMP         10483712  
USERS        9439232
```

了解每个表空间中的最大自由区域有助于你确定是否任何一个段至少能够通过一个区间增长。尽管表空间USERS可以有9MB的自由空间,但它可以被分成三份,每份大约只有3MB。了解这一点是很重要的,因为这可能存在一张区间的NEXT值为5MB的表。如果是这样的话,Oracle不能分配额外的区间,因为这里没有至少5MB的可用自由区间。你需要再次求助于DBA_FREE_SPACE表,只是现在要寻找最大的自由区间,如在这个例子中显示的:

```
SELECT TABLESPACE_NAME, MAX(BYTES) FROM DBA_FREE_SPACE  
GROUP BY TABLESPACE_NAME;
```

```
TABLESPACE MAX(BYTES)
```

```
-----  
RBS          20969472  
SYSTEM       4096000  
TEMP         10483712  
USERS        9439232
```

观察最后两条SQL语句的输出,你可以看到USERS表空间实际上把所有的9MB放在一起。注意,尽管SYSTEM表空间的最大自由区域只有4MB,然而它总计有11MB的自由空间。

现在你需要看一下表空间中包含的所有段。回忆一下，存储参数 NEXT 一直指示段申请的下一个区域的尺寸。至少你应该弄明白任何一个段至少可以通过一个区域增长。这就把在空间之外运行的危险降到了最低。你会希望看一下 DBA_SEGMENTS 表，因为它考虑了数据库中的所有段，无论它们是什么类型。这个例子列出了数据库中不能增长至少一个区间的所有的段（与附加信息）：

```
COLUMN SEGMENT_NAME FORMAT A30
SELECT SEGMENT_NAME,SEGMENT_TYPE,OWNER,A.TABLESPACE_NAME "TABLESPACE_NAME",
INITIAL_EXTENT,NEXT_EXTENT,PCT_INCREASE, B.BYTES "TABLESPACE MAX FREE SPACE"
FROM DBA_SEGMENTS A,
(SELECT TABLESPACE_NAME,MAX(BYTES) "BYTES" FROM DBA_FREE_SPACE
GROUP BY TABLESPACE_NAME) B
WHERE
A.TABLESPACE_NAME = B.TABLESPACE_NAME AND NEXT_EXTENT > B.BYTES;

SEGMENT_NAME
-----
SEGMENT_TYPE      OWNER                TABLESPACE_NAME
-----
INITIAL_EXTENT NEXT_EXTENT PCT_INCREASE TABLESPACE MAX FREE SPACE
-----
TEST1
TABLE              TGASPER              USERS
512000      41943040              50              9439232
```

在这个例子中你可以看到 TGASPER 模式中的 TEST1 表不能分配另一个区间，因为 NEXT_EXTENT 参数被设置为 40MB，而 USERS 表空间的最大自由区间是 9MB。因为 40MB 也比表空间中的总自由空间大，你需要给表空间增加更多的空间以允许 TEST1 增长。既使已经有了合计起来足够的自由空间，你仍然要增加更多的空间以允许 TEST1 增长到 40MB，或者改变 TEST1 的 NEXT_EXTENT 参数以适合最大的自由区间尺寸。

提示 如果表空间达到了一个点，在这个点上最大的自由区间和总自由区间之间相差很远，这就可能是考虑重新组织表空间的时候了。

警告 在设定 PCTINCREASE 参数时会遇到警告。在前面的例子中，你看到 PCTINCREASE 值被设置为 50。这就意味着每个连续的区间比最后一个大 50%。你的段会非常快地有害地尽量分配大区间。合适的规划通常让你把 PCTINCREASE 设置成一个相当低的值。许多地方的策略要求把所有段的 PCTINCREASE 设置为 0。

不管如何设置段的 PCTINCREASE，几乎总是希望为表空间把 PCTINCREASE 缺省值设置为 1。在 Oracle 7.3 及以后的版本中，当这个参数不是 0 时，SMON 自动地合并相邻的自由段。

表空间碎片会变成一个严重的问题，尤其在开发或测试系统中，因为在这里不同的段被频繁地建立和删除。我以前说过，这种活动会由于造成空间气泡而导致表空间碎片，表空间中的空间气泡太小了不能被大多数段使用。

你应该定期评价你的表空间被分裂成碎片的程度。过多的碎片会阻止段增长，因为所有的自由空间被分裂成小得不能使用的碎片。作为一个临时的解决方法，你可以给表空间增加更多的空间。这就打开了一个大的单个区间，这个区间至少允许段增长并使你的数据库能够运行。建议定期消除碎片以优化性能及磁盘空间。

幸运的是，新的 Oracle 版本为数据库管理员提供了一个相当基本然而非常有用的存储管理工具，称为存储管理器。存储管理器是一个基于图形用户界面的工具，它使数据库管理员能

够快速地检查涉及对象和表空间的关键存储问题。

尽管存储管理器不能替代在这儿讨论的所有信息，但它确实提供了一个快速方法用于检查表空间中已分配的和总的自由空间。

启动存储管理器后，通过双击扩展表空间树的项目。你会在右边的窗口中看到所有的表空间以及每个表空间尺寸和自由空间总数。图 21-11 显示了此信息的一个例子。

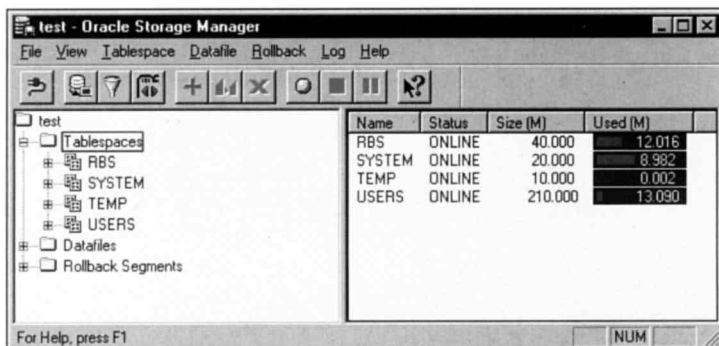


图21-11 存储管理器提供了快速的表空间存储信息

记住，显示的自由空间是表空间中所有自由区间的总数。如果存储管理器指示表空间 USERS 大约有 197MB 的自由空间，并不意味着所有 197MB 的空间是相连的。

21.4.2 表检查

数据库表检查包括估计总的已分配的容量以及实际使用的空间。你也会对表分配的区间数目感兴趣。一两个已用空间与分配空间的比值很低的区间可能暗示着 INITIAL 和 NEXT 区域值可能被设置得太高了。而很多区间可能表明 NEXT 区间参数被设置得太低了。

你可以通过合计所有相关的区域来确定表的总分配空间。你可以在 DBA_EXTENTS 表中找到区间信息。这个例子演示了你如何能够容易地计算出给定的表空间中每个表的分配空间（这个例子中的表空间是 USERS）：

```
SELECT TABLESPACE_NAME, SEGMENT_NAME, SUM(BYTES), COUNT(*) EXT_QUAN
FROM DBA_EXTENTS
```

```
WHERE TABLESPACE_NAME='USERS' AND SEGMENT_TYPE='TABLE'
```

```
GROUP BY TABLESPACE_NAME, SEGMENT_NAME;
```

TABLESPACE	SEGMENT_NAME	SUM(BYTES)	EXT_QUAN
USERS	EMP	102400	1
USERS	DEPT	10240	1
USERS	INOUT_LOG	4096000	100
USERS	PHONE_LOG	2048000	1
USERS	HOURS	1024000	2

你可以看到每个表（在 SEGMENT_NAME 列中）分配的空间总数（用字节表示）以及为表分配的区间数量（在 EXT_QUAN 列中）。注意 INOUT_LOG 表已经在 100 个区间中分配了 4MB。这表明存储参数 NEXT 被设置得太低了。

对表的最后评估集中在确定已用空间与分配空间的比值上。回忆一下，Oracle 是根据存储参数 INITIAL 和 NEXT 的值分配空间的。因此，你不能肯定表真正需要分配给它的所有空间。

你所计算的比值仅仅是个估计值，但是它对计算表所使用的实际空间仍然是有用的。

Oracle 通过 ANALYZE TABLE 语句提供了一个计算表统计量的方法。对数据库中的每个

表，你应该定期地计算（或者至少是估计）统计量。使用与分析 PHONE_LOG表的例子相似的命令：

```
ANALYZE TABLE PHONE_LOG COMPUTE STATISTICS;
```

当这条命令运行时，DBA_TABLES视图反映了搜集到的 PHONE_LOG表的统计量。在DBA_TABLES中，BLOCKS和EMPTY_BLOCKS列分别指示已经使用的和没被使用的块数目。要知道这些列假定当一个块已经使用了，它就一直被使用。如果一个表包含一百万行，其中700 000行后来被删除了，BLOCKS和EMPTY_BLOCKS值将不会改变。

下面的SQL语句用已分配空间的百分数显示 PHONE_LOG表使用的空间：

```
SELECT BLOCKS/(BLOCKS+EMPTY_BLOCKS)*100 USED_PCT FROM DBA_TABLES WHERE
       TABLE_NAME='PHONE_LOG';
USED_PCT
-----
15.397996
```

这里你可以看到尽管 PHONE_LOG表已经分到了2MB空间，但只有15%被实际用来存储表。除非你预计这个表在容量上将充分地增长，否则可以考虑删除并重新载入这些表，并只将大约700KB的空间分配给它。

提示 尽管对每个表来说运行一条ANALYZE TABLE命令是必要的，然而生成一个脚本计算整个数据库统计量的工作可以通过使用DBMS_UTILITY.ANALYZE_SCHEMA存储过程而变得更加容易。

21.4.3 优化簇存储

簇检查主要局限于识别过多的区间。为簇优化存储甚至比表更重要，因为它们的效率（或者不足）将不只在表中表现出来。你可以使用下面的SQL语句为每个簇计算区间的数目：

```
COLUMN SEGMENT_NAME FORMAT A20
SELECT OWNER, SEGMENT_NAME, COUNT(*)
       FROM DBA_EXTENTS WHERE SEGMENT_TYPE='CLUSTER'
GROUP BY SEGMENT_NAME, OWNER;
```

COUNT(*)列显示了每个簇的区间数目。理想情况下，你希望每个簇刚好有一个区间。如果超过一打就表明INITIAL和/或者NEXT存储参数值被设置得太低了。

不幸的是，Oracle没有为你提供一种简单的方法确定在一个簇中已使用的块与已分配的块的百分数。尽管这些问题的答案潜藏在x\$表中，仍然建议你不要依赖这些表。

你最好的选择是重新计算最佳的存储参数，就像打算重建一个簇，并把你最近计算的最佳值同该簇当前的设置值相比较。如果这些值不同的话，你最好的选择可能是用正确的存储参数重建簇。要这样做，遵循下面的步骤：

- 1) 执行一个逻辑备份。
 - 2) 删除该簇和相关的表。
 - 3) 用正确的存储参数重新创建已被删除的对象。
 - 4) 从第一步得到的逻辑备份中把行重新载入删除的表中。
- 当这个过程结束后，你就已经准备好允许用户回到数据库系统。

21.4.4 检查索引

同处理簇一样，大部分数据库管理员发现定期的存储完好性检查以验证区间的数目是合

理的。就像对待簇一样，用一条简单的 SQL 语句进行检查：

```
COLUMN SEGMENT_NAME FORMAT A20
SELECT OWNER, SEGMENT_NAME, COUNT(*)
FROM DBA_EXTENTS WHERE SEGMENT_TYPE='INDEX'
GROUP BY SEGMENT_NAME, OWNER;
```

索引可能会对过多的区间数很敏感。对大多数索引而言，应尽力使区间的数目下降到 6 个左右。理想情况当然是仅仅使用一个区域。

21.4.5 观察回滚段的增长

使用回滚段在前面的 21.3 节中已经详细讨论过了。从存储管理的观点出发，你想要了解回滚段增长到多大。回忆一下，通过指定一个存储参数 OPTIMAL 容量，回滚段在事务结束后尺寸会自动缩小。Oracle 保持一个高水位标志，这个标志指示每个回滚段已经增长了多大，自实例启动后它扩充了多少次以及收缩了多少次。使用如下 SQL 语句查看每个回滚段的高水位标志（以字节为单位）：

```
SELECT N.NAME, OPTSIZE, HWMSIZE
FROM V$ROLLNAME N, V$ROLLSTAT S WHERE N.USN=S.USN;
```

对每个回滚段而言，你可以分别在 OPTSIZE 和 HWMSIZE 列中看到优化尺寸和最大尺寸（到目前为止的）。如果 OPTSIZE 是空的，HWMSIZE 就是回滚段目前的容量（没有收缩发生）。

通常，你要为生产时间建立一组回滚段，同时为批量工作建立一组回滚段（可能为特殊工作需要建立一些额外的回滚段）。你会在适当的表空间中一直需要足够的空间以使用最佳的容量包含所有的回滚段。此外，表空间中必须有足够的自由空间，这样就可以适应每一组回滚段的 HWMSIZE，这些回滚段在任何给定的时间都应该是活动的。

例如，假设你有在生产时间使用的三个回滚段 RP01、RP02 和 RP03，RB01、RB02 在晚上用于批处理。前面 SQL 查询的输出结果是：

NAME	OPTSIZE	HWMSIZE
RP01	10240000	20480000
RP02	10240000	40960000
RP03	10240000	10240000
RB01	102400000	102400000
RB02	51200000	61440000

所有回滚段的总最佳容量大致是 180MB（意思是，这是回滚段的绝对最小尺寸）。假定批量和生产工作从来不同时运行，这样在任何给定的时间，无论是生产还是批量回滚段仍然保持它们的最佳容量。因此，在生产期间，你可以推断回滚段需要的最大空间量是 220MB（合计生产回滚段的 HWMSIZE 列和批量回滚段的 OPTSIZE 列）。在批量时间内，回滚段的最大空间需求是 190MB（从实例最后一次启动开始）。

提示 如果 V\$ROLLSTAT 显示了太多的收缩/扩展，你可能希望调度一个脚本以便提取 V\$ROLLSTAT 的内容来确定回滚段收缩或扩展是否有一个特殊时间。某个工作经常只需要分派给它自己的回滚段就可以被隔离出来。

不要直接采用这些计算。通常应提供大量的裕量空间，这些空间要超过你所计算的迄今为止的最大需求。记住，V\$视图中的值在实例重新启动后总是被复位。

21.4.6 管理临时表空间

监视临时表空间(排序发生的地方)可能是一个复杂的主题。临时表空间是专门满足 Oracle 内部需要的。段在大多数自动安装中按常规方式建立和删除。临时表空间的存储管理问题通常仅限于知道表空间尺寸以及任何时间使用的临时存储的最大值。

临时表空间的尺寸应相对不变,并且不会有太大的起伏。尽管涉及排序的大的查询、大表索引的建立,其他的主要事务会分配异常数量的临时段。

提示 记住, Oracle将多次执行排序作为其他功能的一项必要步骤,例如链接和 GROUP BY子句。不要因为没有明确的排序需求就认为Oracle不用临时表空间。

要监控临时段,可以使用下面的查询检查 DBA_SEGMENTS中类型为TEMPORARY的所有段:

```
SELECT owner, segment_name, tablespace_name, bytes, extents
FROM dba_segments
WHERE segment_type = 'TEMPORARY'
```

总是应寻找在改变日志文件中的错误或者返回到应用程序中的错误,这些错误表明临时表空间太小或者没给事务的完成提供足够的区间。在后面的情况中,解决这个问题通常需要将MAXEXTENTS设置得高一些(或者不受限制的)或者增加区间存储参数 NEXT。

21.5 管理增长的数据库

对数据库的仔细监控通常使你花费一些时间为以后的增长仔细地进行计划。大多数数据库需要性能平衡、容错和易维护。你需要为数据库的增长提供路径,同时维持可接受的性能水平,并确保正常运行时间以及数据库与它的数据的完整性。

数据库增长通常集中在段和表空间的增长。增长的表空间只是给每个需要更多存储的表空间增加一个或者多个数据文件,或者是重新设置已经分配给表空间的数据文件的尺寸的问题。尽管段是自动增长的,但是有一个实际的系统限制,这个限制规定你应该把所有的段数据合并到一个单独的区间中。对合并的需求很大程度上依赖于包含在段中的数据。查找很少增长和很少需要注意的表。相反,包含数据的表可能需要定期地合并,因为它们是定期增长的。

提示 Oracle也可以被配置成允许数据文件自动增长,但是这决不能替代标准的增长监控和管理。除非你确实了解数据库的增长方式,否则自动增长数据文件的使用就会受阻。

在本节中,我谈了围绕增长型数据库环境的维护问题,并且建议执行可能的操作以确保你提供的数据库服务能处理它们的需求。

21.5.1 校正过度的表增长

Oracle RDBMS通过给表段分配额外区间的方法处理常规的表增长。你会不时地发现表增长比预期的要快,这种表可以从完全重组中受益。校正过度扩展的表包括下面的步骤:

- 1) 计算表的总尺寸(把所有区间的尺寸加起来)。
- 2) 对表执行一个逻辑备份。
- 3) 删除该表。

4) 用至少是从步骤1中得出的尺寸的原始区间重建该表。应该使原始区间的尺寸足够大以包含当前的数据并加上额外数量的空间,以应付6~12个月的增长。

5) 导入来自第二步的逻辑备份中的表的数据。

这一般称为重组(改组)表。前面的步骤可以用两种方法之一执行——用导出/导入建立逻辑备份,或者用CREATE TABLE AS SELECT语句。对于较小的表,CREATE TABLE AS SELECT语句执行起来经常又快又容易。

提示 设计定期的数据库改组次数可能对你的组织有帮助。这使你可以重组大大突破设计的存储需求的段。

执行表重组的主要问题是处理依赖于表的外关键字约束条件。Oracle不允许删除那种允许外关键字对表进行存取的表。围绕这个问题有两种方法:

删除该表的外关键字,重组该表,重建外关键字。

禁止所有引用将重组的表的外关键字,为该表改变区间存储参数NEXT,截断该表,重新载入数据,重新启用已被禁止的外关键字。

尽管第一种方法工作量较小,但第二种方法更受欢迎,因为没有损失完整性定义。它更受欢迎还因为许可及其他的数据库结构没有像表被删除时那样在截断中受损失。即使你禁止了外关键字,Oracle也不允许删除表——这个限制不适用于截断表。确定下一个区间的尺寸,这样所有的数据库行都填充在这两个区间中并分配足够的空间以满足将来的增长。完成后,你就有一个含有两个区间的表——一个与它建立时的尺寸相同的原始区间以及另一个包含剩下的表数据和增长空间的区间。

21.5.2 合并簇

因为簇和表在它们的存储性质方面是相似的,把它们各自的数据合并到一个单独区域中的过程稍微有所不同。按照下面的步骤合并簇:

1) 计算簇本身以及相关的表的总分配容量。该簇的总容量应该大致和成员表的累计总容量相匹配。

2) 执行该簇的逻辑备份(包含所有相关的表)。

3) 删除该簇和相关的表。

4) 首先重建该表,之后以原来的区间尺寸重建该簇,这个区间要足够包含每个段的所有数据加上足够的能提供6~12个月增长的剩余空间。

5) 重新载入来自第二步的逻辑备份中的簇。

簇与表以及前面提到过的适用于簇的原则在性质上是类似的。

21.5.3 合并索引

索引非常容易合并到一个单独的区域中。因为没有实际数据一定要首先存储、之后重新载入,所以合并索引只有三步:

1) 计算索引需要的总数。

2) 删除该索引。

3) 用最初的区间重建该索引,这个区间要足够大,能包含所有索引的数据以及足够的额

外空间容纳6~12个月的增长。

换句话说，就是用 ALTER INDEX REBUILD 命令重建一个有新的存储参数的索引。这对非常大的索引，或者不能被轻易删除的主关键字索引特别有帮助。务必观察 UNRECOVERABLE 和 PARALLEL 子句以加速索引的重建时间。

警告 ALTER INDEX REBUILD 建立一个已存在索引的新拷贝，并且在新索引被完全建立以前不删除原来的索引。尽管它有索引构造时间方面的好处并且把用户冲突减到最小，但是它需要有足够的自由表空间供给原来的索引和新的索引。而且，过多地使用它会严重地分裂索引表空间。

21.5.4 管理表空间的增长

处理表空间的增长，过去常通过使用 ALTER TABLESPACE ADD DATAFILE 命令把一个数据文件增加到表空间中来解决。尽管相当常规，但这是自动或脚本化的方法都不容易实现的过程，因此需要数据库管理员全神贯注地执行。尽管在按计划进行存储升级时，这不是问题；但当生产系统用完临时空间而数据库管理员不在现场时，这就是个问题了。

很幸运，Oracle 在 7.3 版本中提出了这个问题并超出了数据库的范围。现在有了两个重要特性，用这两个特性可以帮助你管理表空间和数据文件——可重新调整的数据文件和自动数据文件扩展。尽管它们不是严密的空间监控和计划例程的代替品，但这些新特性在任何 Oracle 数据库管理员的“武器库”中都占有很重要的位置。

要重新调整数据文件，可发出下面的命令：

```
alter database datafile [datafile name] resize [new_size];
```

警告 许多操作系统和相关的工具（例如备份软件）在文件尺寸上有限制。在重新调整数据文件或开启 Oracle 的数据文件自动扩展特性前，必须熟悉你的操作系统的限制。

例如，下面的语句把 user_data 表空间中数据文件的尺寸重新调整到 20MB：

```
alter database datafile '/u01/oradata/PROD/user_PROD_01.dbf' resize 20M;
```

你可以把数据文件的尺寸重新调整到同存储数据文件的操作系统卷所能处理的一样大。当重新调整表空间的尺寸以收回失去的空间时，表空间只能被调整到大于表空间的高水位标志的尺寸。考虑下面的表空间情况：

Table1	5MB
自由空间	10MB
Table2	15MB
自由空间	125MB
Table3	15MB
自由空间	100MB

在这个例子中，在表空间中有三个表，加起来共 35MB。表空间本身的尺寸是 265MB，但是你能收回的唯一的空间是最后的 100MB。

使用自动增长的数据文件，在数据文件装满容量时，Oracle 自动地分配预定数量的空间给该数据文件。命令的语法如下所示：

```
ALTER DATABASE DATAFILE [file_spec]  
AUTOEXTEND ON NEXT [increment_size] MAXSIZE [max_size, UNLIMITED];
```

increment_size 定义了当数据文件装满时增加多少空间给它。MAXSIZE 设置数据文件的最

大尺寸。这条命令的例子如下所示：

```
ALTER DATABASE DATAFILE '/u02/oradata/DEVL/temp_DEVL_01.dbf'  
AUTOEXTEND ON NEXT 10M MAXSIZE 500M;
```

提示 当建立表空间时，也可以通过在数据文件说明后立即放置AUTOEXTEND参数和值的方法使能AUTOEXTEND。

AUTOEXTEND对临时段和回滚段表空间特别有用。因为没有更多的空间供临时段或回滚段扩展，因此它在避免事务失败方面是个强大的工具。但是，要提防 `UNLIMITED MAXSIZE` 因为一个在开始分配你所有的临时空间时就应该被删除的失控的查询，你可能发现你的临时表空间数据文件充满了整个操作系统卷。

OS文件系统碎片

Oracle数据库管理员习惯于处理数据库碎片，但是经常忘记操作系统碎片的含义。大多数Oracle站用数据文件建立表空间，而数据文件存储在属于操作系统的文件系统中。文件系统会快速地并很容易地使大部分自由空间严重地分裂成碎片。当Oracle试图建立一个有几兆字节的文件时，它通常依赖于操作系统实际分配物理设备中需要的空间。如果操作系统不能发现连续的自由空间，它将分配多个自由空间，直到凑出足够用于保存数据文件的集合空间。因此，在这个新数据文件中存取数据需要更多的读/写磁头运动。

从性能的观点出发，如果数据文件被操作系统分散在整个磁盘驱动器上，那么保持低的Oracle段碎片是绝对不好的。整理文件系统碎片的方法在不同的操作系统中是不一样的。与许多UNIX工具不一样，Windows NT可以在数据库和系统启动和运行时整理文件系统的碎片。几个正在出售的第三方产品将执行这个功能。在运行时整理碎片应该在最少的数据库活动期间运行。要有一个最近的备份以对灾难有所准备。

21.6 故障检查

21.6.1 区间限制

Oracle可能不能扩展段，这归因于段的表空间中连续空间不足或者是区间太多。记住，段中的MAXEXTENTS参数可以限制允许的区间数量，你可能已经达到了Oracle的内部区间限制（由数据库块容量确定）。如果问题是由于过多数量的区间引起的，你大概应该集中精力在减少区域数量而不是在段上增大区域限制。

21.6.2 临时表碎片

应用程序的开发者坚持认为，批处理程序建立和删除临时表以执行必要的功能是必要的。但这个行为会导致一些表空间会迅速分裂成严重的碎片。

尽量准确地确定哪个程序将建立临时段并改变它们，使它们在一个或更多的专用于临时段的表空间中建立段（临时段应该把所有缺省的PCTINCREASE设置为1或更高）。在批处理循环的结尾，所有的临时段应该已经被临时表空间清除了，并且SMON将把目前所有的自由区间合并到一个大的自由区间中。

21.7 方案：监控数据库的增长

还记得你的数据库第一次是什么时候建立的么？你恰好已建立每件事情；空间被最优地

分配了，没有明显的数据碎片的提示，并且你的查询几乎立刻就能运行。几年（也许几个月）以后，由数据文件组成的表空间放在任何你有空间的地方，自由磁盘存储得紧紧的，性能几乎不能接受；你必须整理这些混乱的情况。

如果剩下的未经检查，数据库很容易会失控增长。在这部分中，我将讨论用来处理数据库并监控它增长的技巧。有了这些技巧，你便能够计划磁盘的分配和数据库的增长。在用户打电话让你知道他们“在表空间 PAYROLL_DATA_01 中不能分配 64 字节”之前。我还会给出一些有用的调整信息，这些信息会帮助你保持数据库正常而且能够做出响应。

本书附带的 CD-ROM 包括几个不同的文件，这些文件组成了一个完整的数据库容量跟踪系统。通过定期（理想的情况是每天）运行 GETSTATS.SQL 程序（可以在 CD-ROM 中找到），你就会搜集到有关数据库中的每个段和表空间的详细历史报告以及趋势信息。优化软件系统以使存储这个历史信息的总开销最小。在存储统计数字时它只存储变化。例如，一个段的存储信息只有在它分配了另一个区间时才被记录。当然这是以运行的性能为代价的，几小时后可以通过调度查询跟踪系统，把对你的系统有影响的性能减到最小。

在运行 GETSTATS.SQL 搜集统计数字前，你需要建立知识库表并支持视图和存储过程。要做到这一点，应以一个有数据库管理员特权的用户身份在数据库上登录（这个计划将被用于保存知识库对象）并运行下面的 SQL 脚本：

1. CREATE_TABLES.SQL
2. CREATE_FUNCTIONS.SQL
3. CREATE_VIEWS.SQL

跟踪系统中包含的所有信息存储在 SPACEHIST 和 PROCDATES 表中。SPACEHIST 包含了你的数据库中段的历史尺寸，而 PROCDATES 每天都为存在统计数字进行存储。由于 SPACEHIST 只记录变化，你需要用一个视图搞清信息的含义。在你运行 GETSTATS.SQL 的时间里，能够使用视图 V_SPACEHIST 查看你的跟踪表中所包含的数据。

V_SPACEHIST 在 PROCDATES 中的每个日期上为每个段都包含一行。V_SPACEHIST 视图被定义成如下形式：

SAMPLE_DATE	DATE
OWNER	VARCHAR2(20)
SEGMENT_TYPE	VARCHAR2(20)
SEGMENT_NAME	VARCHAR2(20)
TABLESPACE	VARCHAR2(20)
BYTES	INTEGER
EXTENTS	INTEGER
NEXT_EXTENT	INTEGER

下面描述 V_SPACEHIST 的列：

SAMPLE_DATE 包含段的统计量生效的日期。

INSTANCE 是包含这个段的数据库的名字。

OWNER 是段的拥有者。

SEGMENT_TYPE 定义被检查的段的类型。

SEGMENT_NAME 指定段的名字。

TABLESPACE 指示包含这个段的表空间的名字。

BYTES 是分配给这个段的字节数。这个值是通过合计分配给该段的所有区间的字节计算出来的。

EXTENTS是用于段的区间数。

NEXT_EXTENT是NEXT_EXTENT参数的值。

如果你不用 WHERE子句查询 V_SPACEHIST, 就会为你跟踪统计数字的每个段返回行, 每天这些统计数字都被搜集。看一下这个视图中包含的所有信息, 它们可能不太有用。幸运的是, 你可以用SQL的强大查询功能搜集需要的信息。

提示 还有许多来自诸如BMC和Platinum的厂家的跟踪数据库增长信息的总控方案。

“一幅画相当于一千个词”精确地说明了数据的图示法比列表更有意义。例如, 如果你希望了解在最后6个月中你的数据库何时从2.7GB增长到6GB, 你可能用SQL查询检查几百行, 而查看一张显示相同信息的图, 立即就能知道最大增长发生的时刻。

ODBC连接和电子表格软件是用于建立简洁然而含有丰富信息的图的全部工具。如你在图21-12看到的, Microsoft Excel电子表格图总体上显示了数据库增长的一段时期。有了这个信息, 你可能希望使用V_SPACEHIST视图向下检查段级别数据。通过把数据用图表示, 你立即便能确定需要把调查集中在哪段时间上。

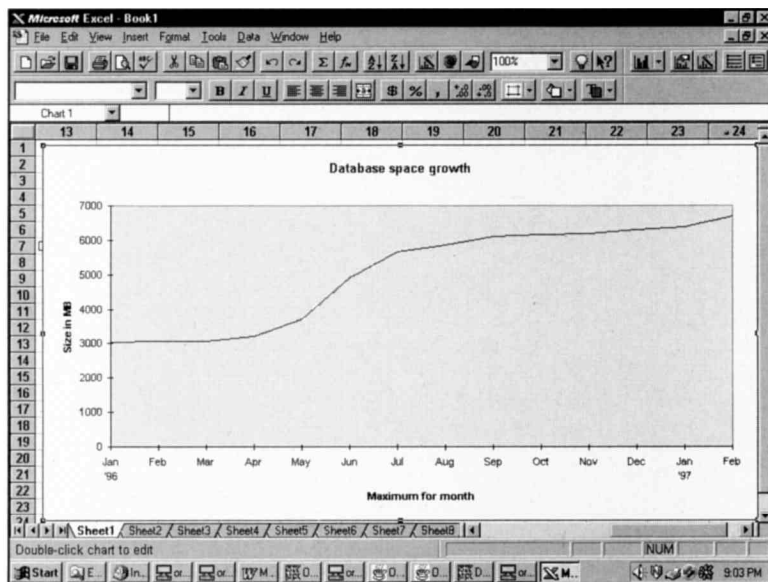


图21-12 用图表示来自V_SPACEHIST视图的信息可以快速确定峰值增长时间

数据库增长的突然改变经常是由于用户数量的增加或应用程序的改变, 这会导致大量数据被存储。当一个数据库被建立后, 数据库管理员经常会忘记查看信息, 你的数据库容量可能在你不知道的情况下迅速增长。通过定期地搜集存储统计量, 你可以了解什么时候发生无规律地增长, 并有机会在发生严重问题前采取预防措施。

假设, 你每天晚上运行 GETSTATS.SQL, 并且每星期你都看一次显示你的成品数据库在最后6个月中的增长的图。你要用如下的 SQL 语句开始:

```
SELECT sample_date, SUM(bytes)
  FROM v_spacehist WHERE sample_date > (SYSDATE) - 180
GROUP BY sample_date;
```

因为V_SPACEHIST视图包含了它所跟踪的每个段的 OWNER和TABLESPACE信息, 所以你可以缩小焦点来跟踪特定的用户或表空间的增长。这个例子提供了用户 TFASPER拥有的所

有段在90天内的增长信息。

```
SELECT sample_date, SUM(bytes) FROM v_spacehist
WHERE owner = 'TGASPER' AND sample_date > (SYSDATE - 90)
GROUP BY sample_date;
```

监控特定的表空间的增长一样容易实现，如同下面这个例子中所显示的那样：

```
SELECT sample_date, SUM(bytes) FROM v_spacehist
WHERE tablespace = 'USERS' AND sample_date > (SYSDATE - 180)
GROUP BY sample_date;
```

除非你处于一个独特的环境中，否则随着时间的推移你的数据库将会增长。当最初设计系统时，你要为将来预期的增长做出计划。对数据库执行定期的监控有助于确保你不会放松对不可预计的增长的警惕。