第二部分　数据库管理

第5章　开发过程管理

管理应用程序的开发并非一个容易的过程。从 DBA的观点看，管理开发过程的最好方法是成为开发小组的一个成员。在本章中，你将看到有关应用程序向数据库移植涉及的活动，以及执行操作的相关技巧。这些细节包括系统角色的描述和数据库对象的定制规则。

本章的重点在于，在数据库的不同阶段对生成对象的活动进行控制。这些活动与第 3章、第4章所描述的数据库规划活动相一致。第 6章和第8章将分别讨论数据库生成后的监控与调整活动。

仅仅通过运行一系列的create table命令来执行数据库中的一个应用程序，将使这一个生成进程无法与其他主要方面 (如规划、监控、协调)相统一。DBA必须参与应用程序的开发过程，以保证正确设计可以支持最终产品的数据库结构。本章中所讨论的这些方法也同时为进行数据库监控和调整工作提供了重要信息。

5.1　成功三要素

一个数据库的生命周期由前面已经提到的四种活动组成：规划(p1an)、生成(create)、监控(monitor)、调整(tune)。这一周期成功实现的因素取决于三个重要方面：培植过程(cultural process)、管理过程(management process)和技巧(technology)。

对数据库开发人员的实现效果的管理，要求执行下面三个方面的活动：

1) 培植: 合作精神和开发小组应当全力满足 DBA在这一活动中的需要。

2) 管理: 开发人员对数据库生命周期方法所持的观点必须是可实施的。

3) 技巧: 开发人员和DBA必须定义相应的机制，以确保对一些细节保持适当的关注。

注意　若没有一个投入的团体或没有可跟踪交付使用的产品的技术，试图实施生命周期方法将不会有长期的收效。

5.2　培植过程

为了打破传统意义上的DBA与开发人员的界线， 应当重新确立他们之间的关系。双方也应当认可这种修正过的关系。只有组合在一起的小组感到新的小组结构对开发过程的价值，这一开发过程才能得以完成。这个开发团体有责任消除掉原先的势力范围之争，否则这种势力范围上的争议会在开发过程一开始就毁掉整个计划。

一个联合的DBA/开发人员小组可以通过以下几个方面来提高效率：

- 创建易于管理的应用程序。
- 创建正确确定大小和组织的应用程序，从而在重新组织时无需停机。
- 为尽可能提高性能生成相应的索引。

- 标识出应用程序最常使用的表和索引。
- 标识并改正构造拙劣的SQL以避免对性能的冲击。
- 标识应用程序中的只静态查询的表。
- 为每个应用程序创建一个易理解的外部应用程序接口。
- 在开发过程中及早发现技术问题。
- 标识联机用户和长时间运行的批处理之间的资源调度冲突。
- 在开发过程中，应当允许支持应用程序开发的DBA对应用程序拥有部分所有权。

当DBA了解他所服务的应用和业务需求时，他对上述这些工作所发挥的作用会大大增强。如果你了解业务需求，就能更好地了解开发人员在应用程序开发过程中的目标。另一个好处是，了解应用程序和业务将会大大改善你与应用程序开发人员和用户的有效联系能力。最后一个好处是，从一开始就和应用开发小组一起工作可提高你正确调整应用程序所使用的数据库的能力。

维护的艰难、碎片、缓慢以及分散的数据库应用程序将会导致停机、协调困难甚至用户的反对，这些都会增加所花的代价。然而，只要建立了DBA和开发人员之间的良好关系，这些代价都是可以避免的。开发方法必须明确DBA和开发人员关系中各自的作用与责任，并且必须被开发小组中不同岗位的人员所接受。这样就会大大降低开发方法的第三方执行难度。

5.3 管理过程

为了正确管理开发工作，开发方法不仅要表明不同功能阶段之间的关系，还必须明确定义应用程序开发过程中每一个阶段所要达到的意图。例如，何时将应用程序从开发阶段移交并进入测试阶段，何时将应用程序从测试阶段转向最终产品阶段？由谁来决定？

答案是由这个开发方法来决定。如果开发阶段的意图已经实现并已进行过检查和验证，则可以将应用程序转入测试环境，开发人员也应在这样的限制条件下进行工作。

5.3.1 定义环境

大多数应用程序的开发环境被分成2至5个阶段。它们是开发（development）、系统测试（system test）、承受力测试(stress test)、验收测试（acceptance test）、产品（production）。为便于讨论，我们将三个测试阶段放在一起。实际采用的阶段数由开发方法确定。

> **注意** 虽然在这里没深入讨论，承受力测试是开发过程的一个重要部分。实际上，在验收测试之前进行承受力测试是非常有用的，与使用负载有关的所有问题都应在验收测试之前解决。

对于它们当中的每一阶段，开发方法都需要指定该阶段的最终结果，并且需要指定前一阶段的结果对后一阶段的意义。一旦完成了这一工作，数据库对不同阶段的需求就会减少。

例如，在开发阶段，用户有权力对表进行改动，测试新的思路和创建新的对象。这就需要一个像Oracle Designer这样的集成CASE（计算机辅助系统工程）工具来维持一个同步的逻辑模型。CASE库用于在每一个环境中生成第一组数据库对象，然后开发人员负责维护CASE字典。

一旦系统进入测试阶段，它的最终配置也应该是明确的。此时，应当已确定表卷、用户帐号及所需的性能。同时也应该允许DBA初次尝试创建对应用程序合适的数据库，并且监控

可接受范围之外的性能问题。

在产品阶段，开发人员往往被拒之门外。从数据库的观点看，开发人员属于另一类用户。在产品数据库中对所有数据库对象的必要改变，都必须首先经确定的更改控制过程而通过测试环境。任何系统所需要的改变都应当在测试阶段明确定义。

为了维护用户的系统级权限的正确级别，开发人员的帐号在不同阶段必须进行不同的配置。下一节将要针对前面描述的开发、测试、产品等不同阶段，讨论给予开发人员的相应系统角色。

5.3.2 角色的定义

Oracle所提供的系统级角色中，有三个角色（CONNECT、RESOURCE和DBA）可用于开发环境（其他的角色与数据库管理有关，将在第9章讨论）。可以创建自己的系统级角色，从而定义CONNECT、RESOURCE和DBA之外的系统权限，但对自己所定义的系统权限，可能会在使用和维护上比系统提供的角色困难些。可以根据用户和开发人员在各自环境中所需的系统权限，将系统角色授予他们。

1. CONNECT

CONNECT角色不只给予用户能够在数据库中创建会话的权限。除了CREATE SESSI0N系统权限外，CONNECT角色还给予用户以下权限:ALTER SESSION、CREATE CLUSTER、CREATE DATABASE LINK、CREATE SEQUENCE、CREATE SYNONYM、CREATE TABLE和CREATE VIEW。然而，用户不具有创建表和簇的能力(这些对象都会占用数据库空间)，除非授予用户相应的表空间定额，或被授予RESOURCE角色（在下面讨论）。有关授予空间定额的详细情况，请参见第9章。

通常CONNECT角色已能够满足所有环境中的最终用户。对于某些开发人员，它也可能满足需求。例如，如果开发人员并不需要创建诸如进程、包、触发器及各种抽象数据类型等数据库对象，那么CONNECT角色就可以满足他的需要。

如果希望限制应用程序用户的系统权限，可以创建自己的角色——APPLICATION_USER——它只具有CREATE SESSI0N权限:

```
create role APPLICATION_USER;
grant CREATE SESSION to APPLICATION_USER;
grant APPLICATION_USER to username;
```

2.RESOURCE

RESOURCE角色具有以下系统权限:CREATE CLUSTER、CREATE INDEXTYPE、CREATE OPERATOR、CREATE PROCEDURE、CREATE SEQUENCE、CREATE TABLE、CREATE TRIGGER和CREATE TYPE。具有RESOURCE角色的用户也被授予UNLIMITED TABLESPACE权限，因此这些用户可超越为他们定义的空间定额。应该把RESOURCE角色授予那些需要创建进程和触发器等PL/SQL对象的开发人员。如果开发人员使用了Objects Option（对象选项），RESOURCE角色将给予他们CREATE TYPE权限，该权限允许他们创建和执行类型和方法。

如果想要限制开发人员的权限，可以创建自己的角色并授予它某些系统级权限。例如，可以限制开发人员创建表和簇的能力，而允许他们创建索引和过程对象。在这种情况下，便

可以创建系统级角色并授予它RESOURCE角中除被拒绝的权限之外的所有系统级权限。通常，只须在开发阶段授予开发人员RESOURCE角色，而在测试和产品阶段，CONNECT角色就足够了。如果给予开发人员对产品数据库的RESOURCE访问权，你就将失去对数据库的控制并对你实施改动控制过程的能力有重大影响。

3. DBA

DBA角色拥有带with admin option的所有系统权限，with admin option意味着DBA可以授予其他用户系统权限。在任何开发、测试、产品数据库中，不应授予用户或开发人员DBA角色。如果在开发中授予了开发人员DBA角色，那么在应用程序交付给产品环境时，开发人员将假设他们仍拥有相同的系统权限而编制他们的应用程序。如果不能严格限定DBA权限帐号的访问，那么也就不能保证数据库中数据的安全，而这是DBA人员的一项重要任务。

4. 在测试和产品环境中

测试环境中，角色的合适配置取决于如何使用该环境。如果该环境被用作一个模拟产品数据库的真正的验收测试区域，那么它被授予的角色便应当反映产品角色。然而，如果开发人员被允许修改测试数据库，那么他们将要求访问一个帐户，该帐户拥有他们在开发环境中所拥有的相同权限。

应用程序所使用的表及其他数据库对象，在测试和产品阶段中，通常只被一个帐户所拥有。如果必须在拥有应用程序模式的帐户中执行改动，则DBA可以暂时用该帐户登录并执行变动。通常，开发人员在测试环境中，不应当拥有RESOURCE角色。如果要对系统进行改动，则该改动应首先在开发阶段中实现并经文档化的改动控制过程而移植到测试阶段。

5.3.3 交付使用

如何知道该开发方法已被遵守?为此必须生成一个称为deliverable（交付使用）的项目清单，该清单中的项目必须在应用程序开发中完成。该方法必须明确定义相应的格式及详尽程度，以及数据库生命周期中每一个阶段所需的交付项目。它们应该包括下面所列各项的说明：

- 实体关系图表。
- 物理数据库图表。
- 空间需求。
- 调整目标。
- 安全需求。
- 数据需求。
- 执行计划。
- 验收测试过程。

下面将对上面每个项目进行讨论。

1. 实体关系图表

实体关系图表(entity relationship diagram E-R图表)描述了组成应用程序的实体间所定义的关系。E-R图表对系统目标的理解起着重要作用。同时，它也有利于明确与其他应用程序的接口点，并且确保企业间定义的一致性。有关E-R图表的建模约定请参见第1章。

2. 物理数据库图表

物理数据库图表(physical database diagram)展示了从实体生成的物理表及从逻辑模型中定

义的属性生成的列。物理数据库图表工具通常也能够生成应用程序对象所需的 DDL。有关物理数据库图表的建模约定请参见第 1章。

也可以利用物理数据库图表来识别事务最可能涉及的表，同时也能够识别那些在数据输入和查询操作期间通常一起使用的表。可以使用这些信息来高效地规划在物理设备中这些表（及其索引）的分布以减少所遇到的 I/O冲突（参见第3章和第 4 章）。

3. 空间需求

空间需求(space requirement)应当显示出每一个数据库表及索引的初始空间需求。有关表、簇和索引的合适尺寸的推荐值在本章后面的 5.3.7节“ 确定数据库对象的大小 ”中描述。

4. 调整目标

改变应用程序的设计可能对应用程序的性能产生重大的影响。应用程序设计的选择也可能直接影响到调整应用程序的能力。由于应用程序的设计对 DBA调整程序性能的能力具有重大影响，因此DBA必须参予该设计过程。

在一个系统进入产品环境之前，必须标识该系统的性能目标。不能光靠感觉来确定期望值。如果用户期望该系统至少应与现有系统一样快，那么缺少任何东西都是不能接受的。必须定义和验证应用程序的每个最常用部份的估计响应时间。

在此过程中，很有必要建立两套目标：合理目标和扩展目标。扩展目标描述了全力超越限制系统性能的硬、软件制约而达到的最终结果。保持两套性能目标有利于将主要精力集中在真正与任务有关的目标上， 而不是那些已超越核心系统交付范围的目标。

5. 安全需求

开发小组应当指定应用程序将使用的帐户结构。这应当包括应用程序中所有对象的拥有权和授权方式。所有的角色和权限必须明确定义。本节中所述的交付方案可以用于生成应用产品中的帐户及权限结构(参见第9章中关于Oracle安全能力的综述)。

根据应用程序，可能有必要将批处理帐户的帐户使用从联机帐户中分离开。例如在联机帐户人工登录的同时，批处理帐户可利用数据库的自动登录性能。应用程序的安全计划必须支持这两类用户。

与空间需求的交付方案类似，安全规划是 DBA的一个工作领域。 DBA必须能够设计既符合企业数据库安全要求同时又满足应用程序需求的工具。

6. 数据需求

必须明确定义数据输入和检索的方法，在测试阶段，必须同时测试和验证数据输入方法，也应该记录应用程序的特殊数据归档需求。

同时，还必须对应用程序的备份及恢复要求进行表述。这些要求可以与现场数据库中的备份计划进行比较(参见第10章)。超越现场标准的任何数据库恢复需求，都将要求改变现场备份标准，或增加一个模块以适应应用程序的需要。

7. 执行计划

执行计划(Execution plan)是执行查询请求时数据库将要经历的步骤。它们可以用第 8章讨论的explain plan或set autotrace命令生成。记录数据库的重要查询的执行计划，在规划应用程序的索引使用及调整目标时会起到辅助作用。在产品完成之前生成它们将有利于简化调整工作，也能够在应用发布之前发现潜在的性能问题。生成最重要查询的说明规划将简化执行应用程序代码检查的过程。

8. 验收测试过程

在将应用程序移到产品环境之前，开发人员和用户必须明确定义要达到的功能及性能目标。这些目标组成测试过程的基础部分，而该测试过程将在测试环境中针对应用程序来执行。

这些过程也应当表述如何处理未达到的目标，它们必须非常清楚地列出系统转到下一阶段之前必须达到的功能目标。同时也应当提供那些并不是非常重要的功能目标的清单。这种根据功能不同所做的区分将有利于解决冲突，也有利于构造正确的测试。

5.3.4 Oracle8i中引入的开发环境特性

在Oracle8i中，开发者和DBA可使用两种新的特性来管理开发过程。可以使用存储概要（stored outline）来移植实例之间的执行路径，使用 Database Resource Manager（数据库资源管理器）来控制数据库用户中的系统资源分配。存储概要和资源管理是管理开发环境工作中的重要组成部分。与只使用操作系统控制相比， Database Resource Manager可给DBA更多的对于系统资源分配的控制能力。

1. 实施Database Resource Manager

可使用Database Resource Manager来分配用户类和作业类系统资源的百分比。例如，可给联机用户分配 75%的可用CPU资源，剩下的 25%分配给批用户。要使用 Database Resource Manager，必须创建资源规划（resource plan）、资源用户组（resource consumer group）和资源规划指令（resource plan directive）。在缺省的情况下，可从catproc.sql中调用catrm.sql脚本文件来创建Database Resource Manager需要的结构。

在使用Database Resource Manager命令之前，必须为你的工作创建一个“未决区域”(pending area)。要创建一个未决区域，可使用 DBMS_RESOURCE_MANAGE软件包的CREATE_PENDING_AREA过程。当完成了这些更改后，可使用 VALIDATE_PENDING_AREA过程来检查这些新的规划、子规划和命令的有效性。然后或者提交这些更改（通过 SUBMIT_PENDING_AREA），或者清除这些更改（通过 CLEAR_PENDING_AREA）。管理未决区域的过程不具有任何输入变量，因此创建未决区域样例使用了下列语法：

```
execute DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA( );
```

如果没创建未决区域，就会收到下列消息：

```
ERROR at line 1:
ORA-29371: pending area is not active
ORA-06512: at "SYS.DBMS_RMIN", line 249
ORA-06512: at "SYS.DBMS_RESOURCE_MANAGER", line 26
ORA-06512: at line 1
```

如果收到这样一组错误消息，就如上述所示执行 CREATE_PENDING_AREA过程。在这一节的末尾将介绍验证和提交未决区域的命令。

可使用DBMS_RESOURCE_MANAGER软件包的CREATE_PLAN过程来创建资源规划。由于资源管理是 Oracle8i中的新特性，所以该语法依赖于执行 PL/SQL过程而不是SQL命令。

CREATE_PLAN过程的语法如下所示：

```
CREATE_PLAN
    (plan                          IN VARCHAR2,
     comment                       IN VARCHAR2,
     cpu_mth                       IN VARCHAR2 DEFAULT 'EMPHASIS',
     max_active_sess_target_mth IN VARCHAR2 DEFAULT
           'MAX_ACTIVE_SESS_ABSOLUTE',
     parallel_degree_limit_mth  IN VARCHAR2 DEFAULT
           'PARALLEL_DEGREE_LIMIT_ABSOLUTE')
```

创建规划之后，应给予该规划一个名称（通过 Plan变量）和一个注释。在缺省情况下，CPU分配方法将使用“增强”方法按比例分配CPU。下列例子创建一个称为DEVELOPERS的规划：

```
execute DBMS_RESOURCE_MANAGER.CREATE_PLAN -
 (Plan => 'DEVELOPERS', -
  Comment => 'Developers, in Development database');
```

注意 连字符（-）是SQL*Plus中的一个连续符号，允许一个命令跨越多行。

为了创建和管理资源规划和资源用户组，必须具有对会话的 ADMINISTER_RESOURCE_MANAGER系统权限。DBA具有带with admin option的这一权限。要把这种权限赋予非DBA，则必须执行DBMS_RESOURCE_MANAGER_PRIVS软件包的GRANT_SYSTEM_PRIVILEGE 过程。下列样例赋予用户 MARTHA管理Database Resource Manager的能力：

```
execute DBMS_RESOURCE_MANAGER_PRIVS -
 (grantee_name => 'Martha',  -
  admin_option => TRUE);
```

通过执行DBMS_RESOURCE_MANAGER_PRIVS软件包的 REVOKE_SYSTEM_PRIVILEGE过程可取消MARTHA的权限。

利用ADMINISTER_RESOURCE_MANAGER权限，可以使用 DBMS_RESOURCE_MANAGER中的CREATE_CONSUMER_GROUP过程来创建一个资源用户组。CREATE_CONSUMER_GROUP过程的语法如下所示：

```
CREATE_CONSUMER_GROUP
    (consumer_group IN VARCHAR2,
     comment        IN VARCHAR2,
     cpu_mth        IN VARCHAR2 DEFAULT 'ROUND-ROBIN')
```

可以把用户分配给资源用户组，因此可根据用户的逻辑划分给这些组起名。下列样例创建两个组，分别针对联机开发者和批开发者：

```
execute DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP -
 (Consumer_Group => 'Online_developers', -
  Comment => 'Online developers');

execute DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP -
 (Consumer_Group => 'Batch_developers', -
  Comment => 'Batch developers');
```

一旦完成了规划和资源用户组，就必须创建资源规划指令并把用户分配给资源用户组。要把指令赋予规划，可使用 DBMS_RESOURCE_MANAGER软件包中的 CREATE_PLAN_DIRECTIVE过程。CREATE_PLAN_DIRECTIVE过程的语法如下所示：

```
CREATE_PLAN_DIRECTIVE
    (plan                        IN VARCHAR2,
     group_or_subplan            IN VARCHAR2,
     comment                     IN VARCHAR2,
     cpu_p1                      IN NUMBER    DEFAULT NULL,
     cpu_p2                      IN NUMBER    DEFAULT NULL,
     cpu_p3                      IN NUMBER    DEFAULT NULL,
     cpu_p4                      IN NUMBER    DEFAULT NULL,
     cpu_p5                      IN NUMBER    DEFAULT NULL,
     cpu_p6                      IN NUMBER    DEFAULT NULL,
     cpu_p7                      IN NUMBER    DEFAULT NULL,
     cpu_p8                      IN NUMBER    DEFAULT NULL,
     max_active_sess_target_p1   IN NUMBER    DEFAULT NULL,
     parallel_degree_limit_p1    IN NUMBER    DEFAULT NULL)
```

CREATE_PLAN_DIRECTIVE过程中的多个CPU变量支持创建多级的CPU分配。例如，可以把全部CPU资源（级1）的75%分配给联机用户，从剩余的CPU资源（级2）中，把50%分配给第二个组，再把第二级中的另外50%CPU资源在第三级中拆分成多个组。CREATE_PLAN_DIRECTIVE过程最多可支持8级CPU分配。

下列样例为DEVELOPERS资源规划中的Online_developers（联机开发者）和Batch_developers（批开发者）资源用户组创建了规划指令：

```
execute DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE -
 (Plan => 'DEVELOPERS', -
  Group_or_subplan => 'Online_developers', -
  Comment => 'online developers', -
  Cpu_p1 => 75, -
  Cpu_p2=> 0, -
  Parallel_degree_limit_p1 => 12);

execute DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE -
 (Plan => 'DEVELOPERS', -
  Group_or_subplan => 'Batch_developers', -
  Comment => 'Batch developers', -
  Cpu_p1 => 25, -
  Cpu_p2 => 0, -
  Parallel_degree_limit_p1 => 6);
```

除分配CPU资源外，这些规划指令还限制了由资源用户组的成员执行的并行操作。在上述例子中，批开发者的并行度被限制为6，这样就降低了使用系统资源的能力。联机开发者的并行度被限制为12。

要把一个用户赋予资源用户组，可使用DBMS_RESOURCE_MANAGER软件包中的SET_INITIAL_CONSUMER_GROUP过程。SET_INITIAL_CONSUMER_GROUP过程的语法如下所示：

```
SET_INITIAL_CONSUMER_GROUP
    (user           IN VARCHAR2,
     consumer_group IN VARCHAR2)
```

如果一个用户不具有由SET_INITIAL_CONSUMER_GROUP过程设置的初始用户组，那么该用户就自动被编入称为DEFAULT_CONSUMER_GROUP的资源用户组。

要启动数据库中的Resource Manager，可把init.ora参数RESOURCE_MANAGER_PLAN设置为该实例的资源规划的名字。资源规划可具有子规划，因此可在该实例中创建多个资源分

配级。如果没有设置RESOURCE_MANAGER_PLAN参数的值，就不能在该实例中执行资源管理。

通过alter system命令的set initial_consumer_group子句可以动态地改变该实例以使用另一个资源分配规划。例如，可为日间用户（DAYTIME_USERS）创建一个资源规划，而为批用户（BATCH_USERS）创建第二个资源规划。可以创建一个每天上午6时执行以下命令的作业：

```
alter system set initial_consumer_group='DAYTIME_USERS';
```

在傍晚的设置时间，可改变用户组以有利于批用户：

```
alter system set initial_consumer_group='BATCH_USERS'
```

因此无需停止并重新启动实例就可更改对该实例的资源分配规划。

当以这种方式使用多个资源分配规划时，必须确保不能偶然在错误的时间使用错误的规划。例如，如果在预定的规划改变期间数据库停机，那么改变规划分配的工作可能就不执行。这对用户会有什么样的影响？如果使用多个资源分配规划，就必须考虑在错误的时间使用错误的规划的影响。要避免这样的问题，应试着减少使用中的资源分配规划的数量。

除了本节所示的例子和命令外，也可更改现有的资源规划（通过UPDATE_PLAN过程）、删除资源规划（通过DELETE_PLAN）和删除资源规划加上其所有的子规划及相关的资源用户组（DELETE_PLAN_CASCADE）。可分别通过UPDATE_CONSUMER_GROUP、DELETE_CONSUMER_GROUP过程来更改和删除资源用户组。资源规划指令可通过UPDATE_PLAN_DIRECTIVE来更改和通过DELETE_PLAN_DIRECTIVE来删除。当修改资源规划、资源用户组和资源规划指令时，必须在执行它们之前测试其变化。要测试这种改变，可为操作建立一个未决区域。可使用DBMS_RESOURCE_MANAGER包中的CREATE_PENDING_AREA过程来创建未决区域。当完成改变时，使用VALIDATE_PENDING_AREA过程来检查新的规划、子规划和指令的有效性。然后或者提交这些改变（通过SUBMIT_PENDING_AREA），或者清除这些改变（通过CLEAR_PENDING_AREA）。管理未决区域的过程不具有任何输入变量，因此验证和提交未决区域应使用下列语法：

```
execute DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA( );
execute DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA( );
```

2. 实施存储概要

当从一个数据库移植到另一个数据库时，查询的执行路径可能会有所变化。执行路径可能由于下列几种原因而发生变化：

1) 可能在不同的数据库中使用了不同的优化程序（一个基于成本，而另一个基于规则）。
2) 可能在不同的数据库中启动了不同的优化程序特性。
3) 对被查询表的统计值可能在各个数据库之间不一致。
4) 收集统计值的频率可能在各个数据库之间不一致。
5) 各数据库可能运行不同版本的Oracle内核。

执行路径中的这些差别的影响可能是惊人的。当移植或升级应用程序时，它们可能对查询性能具有极大的负面影响。为了减少这些差别对查询性能的影响，Oracle公司在Oracle8i中引入了一种称为存储概要（stored outline）的特性。

存储概要存储了一组用于查询的提示，每次执行查询时，就使用这些提示。使用存储的提示将提高每次查询都使用相同的执行路径的可能性。提示不能管理执行路径（它们是提示，不是命令），但可降低数据库转移对查询性能的影响。

若要开始为所有的查询建立提示，可把init.ora参数CREATE_STORED_OUTLINES设置为TRUE。如果把CREATE_STORED_OUTLINES设置为TRUE，那么所有的概要就将保存在DEFAULT类别之下。也可以创建概要的自定义类别并使用类别名作为 init.ora文件中CREATE_STORED_OUTLINESA的值，如下所示：

```
CREATE_STORED_OUTLINES = development
```

在这个例子中，存储概要被存储在DEVELOPMENT类别中。

必须拥有CREATE ANY OUTLINE系统权限才能创建概要。如下所示，可使用 create outline命令来创建查询的概要：

```
create outline YTD_SALES
   for category DEVELOPMENT
    on
select Year_to_Date_Sales
  from SALES
 where region = 'SOUTH'
   and period = 1;
```

注意 如果没为概要指定名字，那么该概要就被赋予系统生成的名字。

如果已经在init.ora文件中把CREATE_STORED_OUTLINES设置为TRUE，那么Oracle就为查询创建存储概要。使用create outline命令可给你提供更多对于创建概要的控制。

注意 可以为DML命令和create table as select命令创建概要。

一旦创建了概要，就可以修改它。例如，可能需要改变概要以反映数据卷和分布的重大变化。可以使用alter outline命令的rebuild子句来重新生成执行查询期间所使用的提示，如下所示：

```
alter outline YTD_SALES rebuild;
```

也可以通过alter outline命令的rename子句来重新命名概要，如下所示：

```
alter outline YTD_SALES rename to YTD_SALES_REGION;
```

如下列例子所示，可以通过change category子句来更改概要的类别：

```
alter outline YTD_SALES_REGION change category to DEFAULT;
```

为使优化程序能利用存储概要，应把init.ora参数USE_STORED_OUTLINES设置为TRUE或设置为类别名（如前面例子中的 DEVELOPMENT）。如果允许使用存储概要，那么任何具有存储概要的查询都将使用该概要创建时所生成的提示。也可以通过 alter session命令来启用会话级的USE_STORED_OUTLINES.

使用OUTLN_PKG软件包来管理存储概要。 OUTLN_PKG软件包提供三种能力：

- 撤消从没使用过的概要。
- 撤消特定类别中的概要。
- 把概要从一个类别移到另一类别。

这三种能力中的每一种在 OUTLN_PKG中都有相应的过程。要撤消从没用过的概要，可执行DROP_UNUSED过程，如下所示：

```
execute OUTLN_PKG.DROP_UNUSED;
```

要撤消一个类别中的所有概要，可执行 DROP_BY_CAT过程。DROP_BY_CAT过程以类别名作为它的唯一输入参数。下列例子撤消 DEVELOPMENT类别中的所有概要：

```
execute OUTLN_PKG.DROP_BY_CAT -
  (category_name => 'DEVELOPMENT');
```

要把概要从一个旧的类别重新指定到一个新的类别，可使用 UPDATE_BY_CAT过程，如下列例子所示：

```
execute OUTLN_PKG.UPDATE_BY_CAT -
 (old_category_name => 'DEVELOPMENT', -
  new_category_name => 'TEST');
```

5.3.5 确定数据库对象的大小

为数据库对象选择合适的空间分配是至关重要的。开发人员应在第一次创建数据库对象之前就开始估算所需的空间，而后再根据实际的使用统计对空间需要进行协调。在下面的各小节中，将介绍表、索引和簇的空间估算方法，同时还将介绍用于帮助正确设置 pctfree和pctused值的方法。

1. 为什么要确定对象大小

确定数据库对象大小的理由有以下四条：

1) 预分配数据库中的空间，以把管理对象的空间需要所需的工作量减为最小。

2) 减少由于过量分配空间而消耗的空间量。

3) 消除可能发生的与I/O有关的性能问题。

4) 提高已撤消的自由盘区可被另一个段重新使用的可能性。

通过下面所介绍的设置方法可实现所有这些目标。该方法基于 Oracle对数据库对象空间的内部分配法。这种方法不是依赖于精确的计算，而是依靠近似法，它明显地简化了设置大小的过程，同时也简化了数据库的长期维护。

2. Oracle为何忽略大多数空间计算

如果没考虑用Oracle的内部空间分配法来计算空间要求， Oracle就很可能忽略详细的空间要求。当分配空间时， Oracle遵循下列内部规则：

1) Oracle只分配整个块，而不分配块的部分。

2) Oracle分配块组，通常是5块的倍数。

3) 根据表空间中的可用自由空间， Oracle可分配更大或更小的块组。

可以将这些规则应用于你的空间分配。对于这个例子，假定数据库的块大小为 4KB，创建下列各表：

表 名	initial	next	pctincrease
SmallTab	7K	7K	0
MediumTab	103K	103K	20

在创建这些表之前， DBA需使用Oracle文档中的空间计算方法估算空间需要。创建表之后会发生什么情况呢？

对于SmallTab表，DBA指定初始盘区的大小为 7KB。然而，数据库的块大小是 4KB，为了避免分割块， Oracle将把初始盘区大小舍入(即“向上取整”)为 8KB。但是，8KB表示只有2个块。在大多数情况下， Oracle将把初始盘区大小舍入为5个块——20KB。当分配下一盘

区时，Oracle将使用指定的next值——7KB，并执行类似的计算。

对于MediumTab表，initial值是103KB，该值必须舍入为整数块的倍数，因此 Oracle将把该值舍入为104KB，即26块。此时，Oracle将分析表空间中的可用空间。理想情况下， Oracle将把块分配舍入为5块的下一倍数——30块。如果在表空间中有一个自由盘区，其大小正好在26块和30块之间，那么Oracle可能使用该自由盘区作为MediumTab的第一个盘区。

在为MediumTab分配它的第二个盘区时， Oracle进行同样的空间分配分析并分配第二个盘区30块（120KB）。

当为MediumTab分配第三个盘区时， Oracle回到next的指定值，而不是实际分配的值。MediumTab有一个为20的pctincrease值，因此第三个盘区应比第二个盘区大 20%。如果Oracle在其计算中依赖于实际分配的空间，那么它就会为第三盘区启动分配进程，试图分配 36块（30块×1.2）并搜索一个40块的自由盘区。然而，当 Oracle执行这个计算时，它依靠指定的next值。因此，它取next值（103KB）再增加20%而给出值为123.6KB。对于4KB的块大小，123.6KB是30.9块，舍入为31块，而它又被舍入为5块的下个倍数35块。

SmallTab和MediumTab的指定空间分配和实际空间分配如下：

表	initial	next	pctincrease	第一盘区	第二盘区	第三盘区
SmallTab	7K	7K	0	20KB	20KB	20KB
MediumTab	103K	103K	20	120KB	120KB	140KB

这些结果对于执行空间分配的DBA来说可能是令人沮丧的。SmallTab表只需14KB用于前两个盘区，而它分配了 40KB。MediumTab表应需要206KB用于前两个盘区，而它分配了240KB。并且，MediumTab表的pctincrease设置要求第三盘区应比第二盘区大20%，而它只大16.6%。

正如这一练习所显示，如果不首先考虑Oracle如何分配空间，那么进行设置大小的计算练习毫无意义。如果在空间计算方法中考虑了Oracle的空间分配方法这一因素，就能够有效地分配空间，减少由Oracle进行的改动。然而，在选择盘区大小之前，应首先考虑盘区的尺寸对性能有什么影响。

3. 盘区尺寸对性能的影响

减少表中的盘区数量并不能获得直接的性能好处。在某些情况下（例如在并行查询环境中），一个表中有多个盘区可明显减少 I/O冲突并提高性能。必须正确设置盘区的大小，而不管表中的盘区数量。

Oracle以两种方法从表中读数据：通过RowID（通常直接跟在一个索引访问后）和通过全表扫描。如果通过RowID来读数据，那么表中的盘区数就不是读性能的一个因素。 Oracle将从其物理位置（按RowID）读取每一行并检索数据。

如果通过全表扫描读取数据，那么盘区的大小可能会影响性能。当通过全表扫描读取数据时，Oracle将每次读出多个块。每次读取的块数通过 init.ora参数DB_FILE_MULTIBLOCK_READ_COUNT来设置并受操作系统的I/O缓冲区大小的限制。例如，如果数据库的块大小是4KB，操作系统的I/O缓冲区大小是64KB，那么在全表扫描时每次最多可读取 16块。在这种情况下，把DB_FILE_MULTIBLOCK_READ_COUNT的值设置为大于16不会改变全表扫描的性能。

盘区大小应利用Oracle的能力以便在全表扫描时执行多块读取，因此，如果操作系统的

I/O缓冲区是64KB，那么盘区的大小应是64KB的倍数。

我们来考察一个具有10个盘区、每个盘区的大小为64KB的表。对于这个例子，操作系统的I/O缓冲区大小为64KB。要执行全表扫描，Oracle必须执行10次读操作（因为操作系统的缓冲区大小是64KB）。如果把数据压缩为一个640KB的盘区，Oracle仍然需要执行10次读操作以扫描该表。压缩盘区并不能提高性能。

如果表的盘区大小不是I/O缓冲区大小的倍数，那么所需要的读操作次数可能增加。对于同样的640KB的表，可创建８个盘区，每个盘区80KB。要读第一个盘区，Oracle将执行两次读：一次用于盘区的前64KB，另一次用于盘区的最后16KB（读操作不能跨盘区）。因此，要读整个表，Oracle必须对每个盘区执行两次读，即16次读操作。盘区数从10个减少为8个却增加了60%的读操作数！

因此，要避免为盘区大小付出性能代价，必须选择如下两种策略之一：

1) 创建明显大于I/O容量的盘区。如果盘区非常大，即使盘区的大小不是I/O缓冲区大小的倍数，也只需要很少的附加读操作。

2) 创建其大小是操作系统的I/O缓冲区大小的倍数的盘区。

如果操作系统的I/O缓冲区的大小是64KB，那么盘区大小可选择为64KB、128KB、192KB、256KB等等。下一节将介绍如何进一步筛选这些选择。

4. 尽量再利用撤消的盘区

当撤消一个段时，其盘区就被加回可用自由盘区的池中。必要时其他段可分配撤消的盘区。如果使用一致的盘区大小，Oracle就更有可能重新使用撤消的盘区，因而更有效地利用表空间中的空间。

如果对于所有的盘区使用自定义的大小，那么将要花费更多的时间来管理自由空间（诸如整理表空间的碎片）。例如，如果创建一个初始盘区大小为100KB的表，那么Oracle就将为该表分配一个100KB的盘区。当撤消该表时，这100KB就成为自由盘区。正如第４章所述，如果该表空间的缺省pctincrease设置为非０，那么Oracle将自动合并相邻的自由盘区。举例来说，假设没有相邻的自由盘区——这100KB自由盘区在各端被数据盘区所封闭。当Oracle试图为另一段分配空间时，它将考虑使用这100KB自由盘区。例如，假定一个新段需要各为60KB的两个盘区，Oracle可能选择使用这100KB自由盘区的60KB用于新段，而留下40KB自由盘区，结果是浪费了40%的自由空间。

要避免浪费自由空间，必须使用一组满足下列准则的盘区大小：每个盘区大小必须是较小盘区大小的整数倍。

考察一下先前生成的一组盘区大小：

64KB、128KB、192KB、256KB、320KB、384KB

估算：

- 64KB是基本数值。
- 128KB是64KB盘区的整数倍，因此该值是可接受的。
- 192KB不是128KB盘区的整数倍，因此不该接受该值。
- 如果放弃192KB，那么256KB是可接受的。
- 320KB和384KB不是256KB的整数倍，因此它们是不能接受的。

方案很快就出来了，每个盘区大小必须是其前面盘区大小的两倍。因此，盘区的可接受数值如下：

64KB、128KB、256KB、512KB、1MB、2MB、4MB、8MB、16MB、32MB等等

使用这样的盘区大小值将可消除任何潜在的I/O问题并提高重新利用撤消盘区的可能性。

5. 最后的障碍

正如前面所述，Oracle通常把分配给盘区的块数舍入为5的倍数。对于一个4KB的块大小，可接受的盘区大小（以块为单位）的列表如下：

16、32、64、128、256、512、1024、2048、4096、8192

所有这些值都不能被5整除，因此Oracle把它们舍入为

20、35、65、130、260、515、1025、2050、4100、8195

但是这又破坏了使用正确盘区大小的效果！然而，Oracle可能不舍入这些值。在其空间分配过程中，Oracle搜索要使用的可用自由表空间。如果要求一个32块的盘区并且Oracle找到一个32块的可用自由盘区，就可以分配32块的盘区而不是35块的盘区。此外，盘区尺寸越大，舍入操作对多块读性能的冲击就越小。

6. 估计非簇表的空间需要

只需知道下列4个值就可估计一个表所需的空间：

- 数据库的块大小。
- 该表的pctfree值。
- 行的平均长度。
- 该表中的期望行数。

要计算表的精确空间需求，则需要一些附加信息（例如列数）。要进行快速估算，所有这4种信息都需要。

数据库的块大小可通过数据库的init.ora文件中的DB_BLOCK_SIZE参数来设置。一旦创建了数据库的块大小，就不能改变它。要增大数据库的块大小，必须完全重新创建它（通过create database命令）并导入从旧数据库导出的所有数据。

每个数据库块都有一个用于该块内开销的区域。表的块开销估计为90字节。因此，一个块中的有效空间如下：

数据库块大小（字节）	有效空间（字节）
2048	1958
4096	4006
8192	8102

必须保留该有效空间的一部份以用于更新先前插入块中的行。表的pctfree值设置在insert期间未使用的自由空间的大小。有效空间乘以pctfree值确定多少空间未使用，从块的有效空间扣除该值确定多少空间可用于行。

例如，对于4KB的块大小和pctfree值为10的表，有效空间是：

```
4006-(0.1×4006) = 3605.字节，向下舍入到3605字节
```

在该例的每个数据库块中，有3605字节可用于新记录。

接下来估算平均行长度。DATE值的长度估计为8字节，NUMBER值的长度估计为4字节。对于VARCHAR2列，估算存储在该列中的数值的实际长度。

注意 这些估算含有附加的列开销。实际上，一个DATE值存有7字节，而一个NUMBER值是3字节。

例如，一个表具有10列，估计的平均行长度为600字节。由于每块有3605字节的有效空间（根据前面的估计），所以每块的行数是：

```
3605/ 600= 6
```

现在，必须估算表中的期望行数，如果该样例表拥有25000行，那么所需块数为：

```
25000/6= 4166
```

该表大约需要4166块。然而，它与上述几节所指定的盘区大小不相符。可有下列两种选择：

1) 创建一个16MB(4096块)的initial盘区和一个512KB(128块)的next盘区。

2) 如果空间够用并且预先考虑了表的进一步增长，可创建一个32MB的initial盘区。

如果使用第一个选项，则空间分配（4224块）将超过估算值（4166块）1%。通过分配这附加的1%，就可创建一个表，其盘区大小可正确调整以符合性能和再利用自由盘区的要求。

在下一节中，将介绍如何估算索引的空间以及表、索引和簇的详细空间计算。通常，不需要详细计算，使用有效的估算和预确定允许的盘区大小将极大地简化空间管理过程。

7. 估算索引的空间需要

索引的估算过程与表的估算过程类似。本节中所介绍的估算过程并不生成准确的空间分配要求，而是让你快速估计空间要求并使它们与标准的盘区大小相匹配。要估计索引所要求的空间，只需知道下列4个值：

- 数据库的块大小。
- 索引的pctfree值。
- 索引条目的平均长度。
- 索引中的期望条目。

可通过数据库的init.ora文件中的DB_BLOCK_SIZE参数来设置数据库的块大小。每个数据库块都有一个用于该块中开销的区域。一个索引的块开销估计为161字节。因此，一个块中的有效空间如下：

数据库的块大小（字节）	有效空间（字节）
2048	1887
4096	3935
8192	8031

根据索引的pctfree设置值，该有效空间的一部分将保留不用。然而，索引值不应频繁更改。对于索引来说，pctfree值通常被设置为小于5。有效空间乘以pctfree值就可确定有多少空间可用于索引条目。

例如，对于4KB大小的块和pctfree值为2的索引，有效空间为

```
3935-(0.02×3935) = 3856 字节，向下舍入到3856字节
```

在本例中的每个数据库块，有3856字节可用于新的索引条目。

接下来，估计索引条目的平均行长度。如果该索引是一个连结的索引，则估计每个列的值的长度并把它们加在一起以求出总条目长度。估计DATE值的长度为8字节，NUMBER值的长度为4字节。对于VARCHAR2列，估计存储在该列中的数值的实际长度。

注意 这些估计包括了附加的列开销。实际上，DATE值含有7字节，而NUMBER值一般是3字节。

例如，一个索引具有３个列，估计的平均行长度为 17字节，由于每个块有3856字节可供使用（根据上述估计），所以每个块的条目数为

```
3856/17= 226
```

现在，估计索引中所期望的条目数。如果索引有25000个条目，那么需要的块数为

```
25000/226= 111
```

索引大约需要111块空间，但是，该大小（444KB）与前几节所规定的任何盘区大小不相符。可从下列方案中选择：

1) 创建一个256KB的initial盘区和一个64KB的next盘区，指定minextents为４和pctincrease为０（112块）。

2) 如果空间可用，并且准备进一步增大表，则创建一个512KB(128块)的initial盘区。

如果采用第一个选项，那么空间分配（112块）将超过估算（111块）一个块。如果采用第二个选项，则比估算多13%。遵循上述任一选项就可创建一个表，其盘区大小可正确设置以符合性能和再利用自由空间的要求。

在下列各节中，你将看到对于表、索引和簇的空间要求的详细计算。可以使用这些更详细的计算来验证估算。在使用详细计算方法之前，先通过估算过程。不管所使用的空间估计方法是什么，必须保证盘区大小与为数据库建立的标准盘区大小系列相一致。

8. 一个有关近似法的注释

在Oracle所提供的公式中，有一些是在确定数据表和索引尺寸时必须要用到的精确计算公式。这些计算中所需的细节是不必要的，因为这些计算结果最终要加上 10%～20%。

为什么在事先就已经估计到可能至少有10%的错误，还要去做详尽的计算呢？

空间预测的错误往往是由Oracle对已创建对象的空间管理方式导致的。当对数据库进行insert、update、delete行操作时，Oracle管理数据库数据块中的空间。由于Oracle对空间的管理方式，所使用的空间总是达不到最优。这会导致要求增加空间，在某些情况下甚至要加大一倍。

假定Oracle中的空间管理具有动态性质，只有将静态数据装载到静态数据表时， Oracle提供的精确空间计算才是正确的。由于这只是实际情况表的很小部分，所以本章中的计算只是针对空间使用的估算，而不是去执行非常详尽的计算。在任何情况下，估算与精确计算之间的差异应少于5%，这也是Oracle所设置的一个界限。

9. 计算非簇表的尺寸需要

除了要显示出表的初始空间要求外，空间需求交付方案也应当显示出每一个表中记录量的年增长率的估计值。如果可行，还应当定义每个表的记录量的最大值。

一旦知道了数据表的列定义及数据量，便可计算它的存储需求。这是一个需要猜测能力的过程，因为在创建数据表之前，并不能知道数据量及行长度的准确值。在计算过程中，有效的样本数据是十分重要的，这样可以使计算结果尽可能准确。

首先， 估计块头部所需空间量；这个块头部中含有 Oracle用来管理块中数据的空间。该块头部的尺寸大约为90字节。如果使用一个2KB大小的数据库块，那么此块将有1958字节大小的自由空间；如果数据库的块大小为4KB，则可留下4006字节的自由空间。

其次，用表中的pctfree设置值乘以自由空间来确定用于行更新的所需空间。

如果pctfree值为10，那么有效自由空间乘以0.1，如下所示：

```
4006×(pctfree/100) = 4006×0.1 = 401字节(向上舍入)
```

所以在可用的自由空间中，仅有401字节用于行扩展。可用于存储行的自由空间是原块中自由空间减去为pctfree所保留的空间：

```
4006 - 401 = 3605字节
```

所以在4096字节的数据块中，只有3605字节用于存储行(见图5-1)。

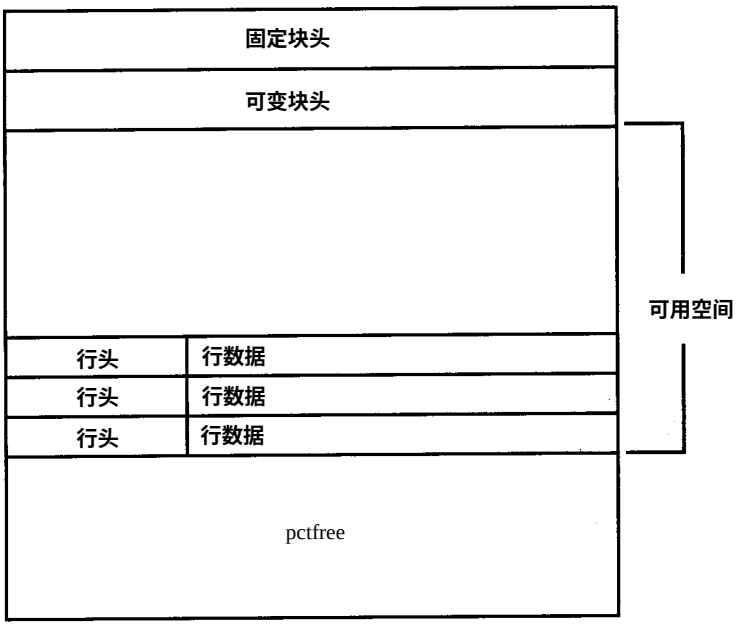

图5-1 数据块中的空间分配

下一步便是计算每一行所用的空间。要估计每行所需的空间值，需要先估计平均行长度。这个平均行长度是行中每一个数值的平均长度之和。如果没有可用数据，那么就估计列中数值的实际长度。但不要把一个列的全长作为它的实际长度，除非你希望这一列中总是充满数据。

例如，假设一个表包含三列，并且均为VARCHAR2(10)。平均行长度不可能超过30；它的实际长度取决于所存储的数据。

如果有样本数据，那么可使用VSIZE函数来确定数据使用的实际空间。还假设一个拥有三个列的表，可以执行下面的查询来确定它的平均行长度：

```
select AVG(NVL(VSIZE(Column1),0))+
       AVG(NVL(VSIZE(Column2),0))+
       AVG(NVL(VSIZE(Co1umn3),0))  Avg_Row_Length
From TABLENAME;
```

在这个例子中，可以确定每一列的平均长度，相加便可得到平均行长度。

如果表较大，上述查询可能要执行较长时间。作为一种代替方案，可用如下语句分析该表：

```
analyze table TABLENAME compute statistics;
```

对于较大的表，可使用estimate statistics子句，如下所示：

```
analyze table TABLENAME estimate statistics;
```

一旦分析了该表，就可从数据字典中选择统计值，如下所示：

```
select Avg_Row_Len
  from USER_TABLES
 where Table_Name = 'TABLENAME';
```

Avg_Row_len值含有用于行开销（如列头部）的附加字节。

对于样本3列表，假设平均行长度是24字节，给表中的每一列加1个字节，则每行长度为27字节。如果表中存在含有超过250个字符的列，则需给这样的列再增加一个字节。最后，为行头部加上3个字节。

每行的空间 = Arg-Row - Length+3 + 列数 + 长列数

在这个例子中，每行的空间为：

每行的空间 = 24 + 3 + 3 + 0 = 30字节

假定一个块的有效字节为3605并且每行为30字节，则可以在该块中拥有120个行:

每块中的行数 = 3605/30

= 120行(向下舍入)

由于可以在每一个块中放入120行，所以可以像估计所期望的行数一样估计所需的块数。正如前一节所提到的，这也只是表存储要求的近似值。由于表中的记录是可操作的，所以它们的空间要求也就会增加。所出现的delete和update次数越多，空间要求量也就越大。

10. 计算适当的pctfree值

必须为每一个表确定适当的pctfree值。这个值代表了为自由空间保留的每个数据块的百分比。当数据块中所存储的行进行长度扩展时，可使用这个保留的自由空间，而行长度的增加是由前面的NULL字段更新或已存在值更新为大值所造成的。

不可能有一个适用于所有数据库中所有表的pctfree值。但是由于pctfree值是针对于应用程序中所出现的更新方式，所以该值的确定也是一个很直接的过程。 pctfree值控制了一个表的块中存储的记录数。要想知道pctfree值是否设置得合适，首先要确定块中行的数量。正如上一节所示，这可以用analyze命令来度量现有表中的行分配。在下面的示例中， analyze命令的compute statistics子句用于生成表的统计结果，从而可以从数据字典视图中选择这些统计值。

```
analyze table TABLENAME compute statistics;
```

注意 如果使用基于规则的优化程序，应在计算pctfree值之后删除统计结果。

一旦完成了对表的分析，便可查询USER_TABLES视图中的记录并记录Num_Rows值和Blocks值。用Blocks值去除Num_Rows就可得到每一个块中所存储的行数量，如下列查询所示：

```
select Num_Rows,                /*number of rows*/
       Blocks,                  /*number of blocks used*/
       Num_Rows/Blocks          /*number of rows per block*/
  from USER_TABLES
 where Table_Name='TABLENAME';
```

一旦知道了每一个块中的行数量，就可模仿最终产品的使用方式来update表中的记录。一旦完成update，就可以通过对表的分析及再次运行查询来检查每一个块中的行数量。如果pctfree值设置得不够大，则有可能使一些行移到新的数据块中以适应它们的新长度。如果这

个行数量没有发生变化，则说明pctfree值是合适的。

注意 如果由于pctfree区中的不合适空间而移动行，这种移动就称为行迁移。行迁移将影响事务处理的性能。

尽管pctfree值可以很大，但会导致不必要的空间浪费。前面提到的 analyze命令也可以生成USER_TABLES视图的Avg_Space列的值。该列显示了每个数据块中的自由字节数的平均值。如果这个值在update测试后较大，则说明pctfree值还可以减小些。

11. 确定正确的pctused值

pctused值决定了何时可以把一个已使用的数据块再增加到能插入行的块清单中。例如，假设一个表的pctfree值为20，pctused值为50，当向表中插入行时， Oracle将为每一个数据块保留20%的自由空间(以备将来更新插入的记录)。如果这时开始delete块中记录， Oracle将不会自动重新使用这些块中的自由空间，直到块中已用空间低于其 pctused值(50%)时，才会插入新的行到记录中。

pctused参数的缺省值为 40。如果你的应用程序要经常进行删除操作，并且你使用这个缺省的pctused值，那将会使数据表中只有 40%的空间被使用。

要得到最好的结果，可设置 pctused值使pctused加上pctfree等于85。如果pctfree值为20%，则可把pctused值设置为65%；这样，至少有65%的块空间总可以被利用，而只保留 20%用于更新和行扩展。

12. 计算索引大小

确定索引大小的过程与上述确定表大小的过程非常相似。只在对象大小的确定上有一些不同，这是由于表和索引有着不同的结构。这里对表空间的估计，尺寸的计算都只是近似的。

一旦知道了索引的列定义及数据量，便可确定它的空间需求。由于在创建表之前不能知道数据量及行长度的准确值， 故这一计算也是一个需要猜测能力的过程。在这里也同样希望有样本数据，以使结果尽可能准确。

首先，估计块头的空间需求， Oracle将用此块头来管理块中数据。这个块头的大小大约为161字节。如果使用一个4KB大小的数据库块，将留下1887字节的自由空间；对于一个4KB大小的数据库块，将留下3935字节的自由空间。

```
4096-161=3935字节
```

接下来，用表中的pctfree值乘以自由空间来确定将有多少自由空间可用于行更新。

如果使用的pctfree值为５，则自由空间乘以0.05，如下所示:

```
3935×(pctfree/100) = 3935×0.05 = 19字节（向上舍入）
```

所以在自由空间中，将为输入扩展保留 197字节。这时自由空间为块自由空间减去为pctfree保留的空间:

```
3935 - 197 = 3738字节
```

可见，对于4096字节的数据块，将有3738字节的有效空间来存储索引条目。

下一步，要计算每一条目所需的空间。要估计出此值，需要首先估计索引中列的平均行长度。对于数据表，需要计算所有列的平均行长度；对于索引，只需关注索引列。平均行长度为索引列中每一个值的平均长度的总和。如果没有样本数据，则估计这些值的实际长度。不要将列的全长用作它的实际长度，除非此列始终充满数据。

例如，假设一个索引含有两个列，且均为 VARCHAR2(10)。平均索引行长度不可能超过

20，它的实际长度取决于所存储的数据。

如果有样本数据，则可用 VSIZE函数来确定数据所用的实际空间。同样假设一个索引拥有两个列，要确定它的平均行长度，可执行如下查询：

```
select AVG(NVL(VSIZE(Column1),0))+
       AVG(NVL(VSIZE(Column2),0))    Avg_Row_Length
from TABLENAME;
```

在这个例子中，每列的平均长度可以被确定，则它们的和可以确定平均行长度。

例如，在这个两列的索引样本中，假设索引列的平均行长度为 16字节。对于总和值，索引中每一个列加1个字节，从而变成每行18字节。如果索引中的列含有127个以上字符的数据，则给每个这样的列再加1个字节。最后，为索引项头部加8个字节。

```
每行空间 = Arg - Row-Lengt列数＋长列数＋8
```

对于这个索引，每行的平均空间为

```
每行空间 = 16 + 2 + 0 + 8 = 字节
```

如果该索引是唯一索引，则再加1个字节：

```
26 + 1 = 2字节
```

对于每块3738字节可用自由空间和每行长度为27字节的索引，则可以在每块中包含138个索引条目。

```
每块条目数＝3738/27 = 138向下舍入)
```

由于每个数据块中可以包含138个索引条目，所以可以在估计所期望的行数量的同时，估计所需的块数量。正如前面所提到的，这也只是表的存储需要的一个近似值。由于表中的记录是可操作的，所以它们的空间需要也要相应增加。 delete和update出现的次数越多，所需的空间也就越大。

在索引中，很少重复使用删除的空间，这样即使数据表不增加，索引也可能增加。例如，如果从表中delete了100行并且随即insert100个新行，则数据表可能会把被删除记录中的自由空间用于插入的记录，并且它的使用空间保持不变。然而这个表中的索引，则很有可能不重新使用已删除记录所释放的自由空间，这样，这个表的使用空间还是会增加。

13. 计算簇表的大小

簇用于在相同物理数据块中存储不同数据表的数据。如果这些表中的记录被一起频繁查询，则适合使用簇。通过在相同的数据块中存储它们，会相应减少读取数据块来完成此类查询的次数，从而改进系统的性能。不过它们也对数据管理事务和仅引用簇中一个表的查询有一定负作用。

由于它们独特的结构，簇表有着与非簇表不同的存储要求。每一个簇需要存储表数据，并维持用于数据排序的簇索引。

簇索引中的列称为簇键(cluster key)，它是该簇中所有表所共用的一组列。由于簇键决定了簇中行的物理位置，所以它们不能被频繁更新。簇键通常是簇中一个表的外键，它引用簇中另一个表的主键。

创建簇之后，在簇键列中创建簇索引。在创建簇键索引之后，则可把数据输入簇中存储的数据表。当插入行时，数据库将在每一个簇块中存储簇键及相应的行。

注意 由于它们相对复杂的结构，即使使用此简单方法，簇大小的确定往往要比索引或表复杂得多。

确定簇的大小也包含了诸如确定表大小和索引大小等内容。例如在前几节所使用的表中含有三个列，每个列中都是 VARCHAR2(10)。如果这个表经常与其他表合并，那么这两个表应当聚簇在一起。为了说明这一点，我们假设第二个表含有两个列 :一个VARCHAR2(10)列和一个VARCHAR2(5)列，前者用于将这两个表合在一起。

由于这两个表使用它们所共有的 VARCHAR2(10)列进行合并，所以该列就是簇键。

首先，估计块头使用的空间量，这是 Oracle用来管理块数据的空间量，簇的块头的大小大约为110字节。如果使用一个 2KB大小的数据库块，就会留下 1938字节的自由空间；对于 4KB大小的数据块，就会留下 3986字节的自由空间。

```
2048 - 110 = 1938字节
```

接下来，用该簇的pctfree值去乘这个自由空间，从而确定为更新行保留的空间。

如果簇中的pctfree值为10，则用有效自由空间乘 0.1:

```
1938×(pctfree/100) = 1938×0.1 = 194字节(向上舍入)
```

所以，对于这个自由空间，将保留 194字节用于行扩展。这时可用的自由空间等于块中自由空间减去为pctfree保留的空间：

```
1938 - 194 = 1744字节
```

然后，再减去用于表头条目的头空间。缓冲区为表数量的 4倍，再加上4字节。对于一个两表的簇，可用的自由空间为 :

```
可用空间 = 1744 - 4-×表数
         = 1744 - 4 -8
         = 1732字节
```

对于一个2048字节的块，将有1732字节用于存储簇输入项。

接下来，计算一下每一个表中单个行所需的空间，这不包括簇键所在的列的长度。

如果有样本数据，便可使用 VSIZE函数来确定数据使用的实际空间。同样，假设一个含三个列的表和一个含两个列的表。要确定平均行长度，可执行下面的查询 :

```
select AVG(NVL(VSIZE(Column1),0))+
       AVG(NVL(VSIZE(Column2),0))    Avg_Row_Length_1
from TABLE1;

select AVG(NVL(VSIZE(Column1),0))    Avg_Row_Length_2
from TABLE2;
```

在这个例子中，可以确定不是簇键的每个列的平均长度，并以这些平均长度之和确定平均行长度。这个例子假设簇键是 TABLE1中的Column3和TABLE2中的Column2。

对于这个例子，假设 TABLE1中非簇键列的平均行长度为 20，TABLE2中非簇键列的平均行长度为3，那么平均行长为23字节。

簇中的每一行中存有行头信息。行的空间要求总和包括行头的空间要求在内，可以通过下面的公式来计算。在这个公式中，长列是指其中的数据长度超过 250个字符的列。

```
行头空间= 4 + 列数 + 长列数
```

合并平均行长度和行头空间得到 :

```
每行空间 = 平均行长 + 行头空间
         = 23 + 4 +列数 + 长列数
         = 23 + 4 + 3 + 0
         = 30字节
```

因此，每一个簇条目将要求30个字节。这个空间并不包含簇索引的空间要求。

确定簇大小过程中的下一步便是确定 size参数值，这也是簇特有的参数。 size参数用于估计簇键及其相关行所需的字节数。

size参数取决于数据的分布，也就是说，表中有多少行用于簇键中的不同数值？要确定这些值，可以对这个簇表进行查询，并且用不同簇键值的数量去除表中的记录量。

```
select
   COUNT(DISTINCT(column name))/     /* Num of records in table*/
   COUNT(*)  rows_per_key            /* Num of cluster key values*/
from tablename;
```

对于这个例子，假设在表TABLE1中，每一个簇键值对应30行，表TABLE2中每一个簇键值只对应一个数据行。

size参数也需要知道簇键值的平均长度。可以用前面所示的 VSIZE查询来对簇表进行查询。由于在两个表中， size参数应是同样的值（通过引用完整性），所以只需对一个表进行查询:

```
select
   AVG(NVL(VSIZE(cluster key column),0)) Avg_Key_Length
 from TABLE1;
```

对于这个例子，假设平均键列值为5字节。这样size参数的值也可通过下面的计算得到：

```
size = (TABLE1中每个簇键值对应的行数×TABLE1的平均行空间)+
       (TABLE2中每个簇键值对应的行数×TABLE2的平均行空间) +
       簇键头+
       簇键列长+
       簇键平均长+
       2×(TABLE1中每个簇键值对应的行数+TABLE2中每个簇键对应的行数)
```

簇关键字头部为19字节长。因此，对于样例数据：

```
size = (30×20) + (1×3) + 19 + 10 + 5 + (2×21)
     = 699字节
```

因此，每一个簇键将要求699字节，取整为700字节。这将放置在块中的可用空间内。先前计算的可用空间为1732字节，这样，每个块的簇键数为:

```
每块的簇键数 = 自由空间/(size + 42) = 1732/(700 + 42) 向下舍入)
```

每一个数据库块可以存储两个簇键的值。

由于每个数据块可以存储两个簇键的数值，故该簇所需的块数就是簇值数量的一半。也就是说，如果有40个不同的簇键值，则需要为该簇分配20个块。

14. 确定函数索引的大小

在Oracle8i中, 可以创建函数索引（function index），例如，可以在UPPER(Name)上创建一个索引。函数索引可明显提高查询调整能力。确定函数索引大小的方法与确定标准索引大小所采用的方法相同，如本章前几节中所描述。

15. 逆向关键字索引

在Oracle8中，可以创建一个索引并指定 reverse参数来反顺序存储索引块中的字节。在Oracle8i中，则可以创建一个逆向键索引(reverse key index)。在逆向键索引中，数值是逆向存储的。例如，值2201被存为1022。如果使用标准索引，那么连续的值就一个靠一个地存储。在逆向键索引中，连续的值不是紧邻着存储的。如果你的查询不执行范围扫描，并且你只关

心索引中的I/O冲突，则可把逆向键索引作为可选的调整方法。确定逆向键索引大小的方法与确定标准索引所用方法相同。

16. 确定位映射索引的大小

如果创建一个位映射索引，则Oracle将动态压缩所生成的位映射。压缩位映射可极大地节省存储空间。要估计位映射索引的大小，可使用本章前面各节所提供的公式来估计相同列中的标准（B*-tree）索引的大小。在计算了B*-tree索引的空间需求之后，将该大小除以10来确定这些列中的位映射索引的大小。通常，位映射索引的大小是相应的B*-tree索引的5～10%。

17. 确定索引组织表的大小

索引组织表是按其主键排序存储的。索引组织表的空间要求等于一个在该表所有列上建立的索引的空间要求。空间估计中的差别在于每行所用空间的计算，因为索引组织表没有RowID。对于索引组织表，头字节从8字节减少为2字节。

```
每行空间 = Arg - Row - Length列数 + 长列数+ 2
```

18. 确定包含大对象（LOB）的表的大小

LOB数据（属BLOB或CLOB数据类型）通常是在主表外存储的。可使用create table命令中的lob子句来指定LOB数据的存储。在主表中，Oracle则存储指出LOB数据的lob定位值。lob定位值的长度估计为24字节。

Oracle并不总是在主表外存储LOB数据。一般情况下，如果LOB数据不超过4KB长，就不在主表外存储LOB数据。因此，如果要存储短的LOB值，就必须考虑对主表存储空间的影响。如果LOB值小于4000字符，就能够使用VARCHAR2数据类型而不是LOB数据类型来进行数据存储。

19. 确定分区的大小

在Oracle8中，可创建表的多个分区（partition）。在分区表中，多个独立的物理分区构成一个表。例如，一个SALES表可以具有4个分区：SALES_NORTH、SALES_SOUTH、SALES_EAST和SALES_WEST。可以使用本章前面介绍的确定表大小的方法来确定每个分区的大小。也可以使用本章前面介绍的确定索引大小的方法来确定分区索引的大小。有关分区管理的详细信息，请参见第12章。

20. 确定有抽象数据类型的表的大小

Oracle把对象相关结构引入Oracle8中的数据库。在这些新结构中，两个最重要的特性是抽象数据类型（abstract datatype）和构造方法（constructor method）。抽象数据类型定义数据的结构，例如，一个ADDRESS_TY数据类型可以包含地址数据的属性以及操作该数据的方法。当创建ADDRESS_TY数据类型时，Oracle将自动创建一个称为ADDRESS_TY的构造方法。该方法含有与数据类型的属性相符的参数，因而简化了新值插入的操作。下面将介绍如何创建使用抽象数据类型的表以及与这种实现有关的确定大小问题和安全问题等信息。

对于Oracle8，可以用抽象数据类型作为创建表时的列定义。例如，可以创建一个抽象数据类型的地址：

```
create type ADDRESS_TY as object
(Street    VARCHAR2(50),
City       VARCHAR2(25),
State      CHAR(2),
Zip        NUMBER);
/
```

一旦建立了ADDRESS_TY数据类型，便可以在生成数据表时使用它：

```
create table CUSTOMER
(Name      VARCHAR2(25),
Address    ADDRESS_TY);
```

当创建一个抽象数据类型时，Oracle 在insert操作期间生成一个可供使用的构造方法。该构造方法有着与抽象数据类型相同的名字，并且它的参数是此抽象数据类型的属性。当在CUSTOMER表中插入记录时，就需要使用这个 ADDRESS_TY 抽象数据类型的构造方法来插入地址值。

```
insert into CUSTOMER values
('Joe',ADDRESS_TY('My Street', 'Some City', 'ST', 10001));
```

在这个例子中，insert命令调用ADDRESS_TY构造方法，以便把值插入到 ADDRESS_TY 数据类型的属性中。

抽象数据类型的使用会使数据表的空间要求增加，每使用一次，就增加 8个字节。如果一个数据类型含有另一个数据类型，那么就应给每个数据类型增加 8个字节。

21. 使用对象视图

抽象数据类型的使用可能会增加开发环境的复杂性。当查询一个抽象数据类型的属性时，必须使用对于不包含抽象数据类型的表不使用的语法。如果不是在所有的表中采用抽象数据类型，那么就必须对一些表使用一种语法，而对其他表使用另一种语法。还应事先知道哪些查询使用抽象数据类型。

例如，CUSTOMER表使用上面定义的ADDRESS_TY数据类型。

```
create table CUSTOMER
(Name      VARCHAR2(25),
Address    ADDRESS_TY);
```

而ADDRESS_TY数据类型本身又有４个属性：Street（街道）、City（城市）、State（州）Zip（邮政代码）。如果想从CUSTOMER表的Address列中选择Street属性，你可能写下列查询：

```
select Address.Street from CUSTOMER;
```

然而，这个查询将不工作。当查询抽象数据类型的属性时，必须使用该表名的相关变量，就是说，关于所选对象存在二义性。要查询 Street属性，可使用 CUSTOMER表的相关变量（在这种情况中是“ C ”），如下列所示：

```
select C.Address.Street from CUSTOMER C;
```

正如本例所示，必须为抽象数据类型属性的查询使用相关变量，即使该查询只访问一个表。因此对于抽象数据类型属性，有两个非标准的查询特性：用于访问属性的符号和相关变量要求。为了始终如一地实施抽象数据类型，可能需要改变 SQL标准以支持百分之百地使用相关变量。即使始终如一地使用相关变量，访问属性值所需的符号也可能会发生问题，因为不能在没使用抽象数据类型的表中使用类似符号。

对象视图可为解决这种矛盾提供有效的折衷方案。在前面的例子中创建的 CUSTOMER表时，假设已经存在ADDRESS_TY数据类型。但是如果表已存在那该怎么办呢？如果先前已创建了关系型数据库应用程序并试图在应用程序中实施对象关系原则而不重新生成或创建整个应用程序，那又该怎么办呢？所需要的就是能够重叠面向对象（ OO）的结构，例如现有关系型表上的抽象数据类型。Oracle提供对象视图作为定义现有关系表所使用对象的一种方法。

如果CUSTOMER表已存在，就应创建 ADDRESS_TY数据类型并使用对象视图来把它与

CUSTOMER表关联起来。在下面的例子中，只使用通常提供的数据类型来把 CUSTOMER表创建为关系型表：

```
create table CUSTOMER
(Name         VARCHAR2(25) primary key,
 Street       VARCHAR2(50),
 City         VARCHAR2(25),
 State        CHAR(2),
 Zip          NUMBER);
```

如果想创建另一个用来存储有关人员和地址的表或应用程序，则可选择创建ADDRESS_TY数据类型。然而，为了统一起见，该数据类型也应应用于 CUSTOMER表。下列例子将使用上节中创建的ADDRESS_TY数据类型。

可以使用任何已定义的数据类型来创建一个基于 CUSTOMER表的对象视图。要创建一个对象视图，可使用create view命令。在create view命令中，指定将构成视图基础的查询。创建CUSTOMER_OV对象视图的代码如下所示：

```
create view CUSTOMER_OV (Name, Address) as
select Name,
       ADDRESS_TY(Street, City, State, Zip)
  from CUSTOMER;
```

CUSTOMER_OV对象视图将有两个列：Name（名字）列和Address（地址）列（后者由ADDRESS_TY数据类型定义）。注意不能指定object作为create view命令中的一个选项。

在这个例子中出现了几个重要的语法问题。当根据现有的抽象数据类型建立一个表时，可通过引用列名（例如Name）而不是构造方法从该表中选择列值。然而，当创建对象视图时，就应该引用构造方法的名字（例如 ADDRESS_TY）。还可使用构成对象视图基础的查询中的where子句，因此能够限制可通过对象视图访问的行数。

如果使用对象视图，那么作为 DBA将可用以前所用的相同方法来管理表。仍然需要管理数据类型的权限（参见本章下一节关于抽象数据类型的安全管理的信息），但是表和索引的结构将和创建该抽象数据类型之前的结构相同。使用旧的结构有利于简化管理任务，同时也允许开发人员通过表的对象视图访问对象。

也可以使用对象视图来模拟由行对象所使用的引用。行对象是对象表中的一些行。要创建支持行对象的对象视图，必须首先创建一个其结构与表的结构相同的数据类型。

```
create or replace type CUSTOMER_TY as object
(Name         VARCHAR2(25),
 Street       VARCHAR2(50),
 City         VARCHAR2(25),
 State        CHAR(2),
 Zip          NUMBER);
/
```

接下来，创建一个基于 CUSTOMER_TY类型的对象视图，同时把 OID（对象标识符）值赋予CUSTOMER中的记录。

```
create view CUSTOMER_OV of CUSTOMER_TY
  with object OID (Name) as
select Name, Street, City, State, Zip
  from CUSTOMER;
```

该create view命令的第一部分给予视图一个名字 (CUSTOMER_OV)并通知ORACLE，该视图的结构基于CUSTOMER_TY数据类型。 OID是行对象的对象标识符。在这个对象视图中，Name列将被用作OID。

如果具有第二个通过外键／主键关系而引用CUSTOMER的表，那么就可以设置一个包含对CUSTOMER_OV引用的对象视图。例如，CUSTOMER_CALL表含有一个到CUSTOMER表的外键，如下所示：

```
create table CUSTOMER_CALL
(Name          VARCHAR2(25),
 Call_Number   NUMBER,
 Call_Date     DATE,
 constraint CUSTOMER_CALL_PK
     primary key (Name, Call_Number),
 constraint CUSTOMER_CALL_FK foreign key (Name)
    references CUSTOMER(Name));
```

CUSTOMER_CALL的Name列引用CUSTOMER表中的相同列。由于已经模拟了基于CUSTOMER的主键的OID（称为pkOID），所以需要创建对这些OID的引用。Oracle提供了一个创建引用（称为pkREF）的操作符MAKE_REF。在下面的程序段中，MAKE_REF操作符被用来创建从CUSTOMER_CALL的对象视图到CUSTOMER的对象视图的引用：

```
create view CUSTOMER_CALL_OV as
select MAKE_REF(CUSTOMER_OV, Name) Name,
       Call_Number,
       Call_Date
  from CUSTOMER_CALL;
```

在CUSTOMER_CALL_OV视图中，必须告诉Oracle要引用的视图的名字和构成pkREF的列。现在可以通过在Customer_ID上使用DEREF操作符来从CUSTOMER_CALL_OV中查询CUSTOMER_OV数据。

```
select DEREF(CCOV.Name)
  from CUSTOMER_CALL_OV CCOV
 where Call_Date = TRUNC(SysDate);
```

因此可以不用直接查询CUSTOMER表而从查询中返回CUSTOMER数据。在这个例子中，Call_Date列被用作对查询返回行的限制条件。

不管是使用行对象还是列对象，都可以使用对象视图来保护对象关系中的表。不必修改表，可以采用一直采用的方法来管理这些表。所不同的是，现在用户可以访问CUSTOMER的行，就像它们是行对象似的。

22. 确定对象表和REF的大小

当创建一个对象表时，Oracle就为该表中的每个行生成一个OID值。OID(对象ID)值把16个字节添加到平均行长度。当创建一个引用对象表的表时，该表将含有一个带REF数据类型的列。当估计表的空间要求时，则把REF数据类型列的长度估计为16字节。

23. 抽象数据类型的安全

在上面几节的例子中，都假设同一用户拥有ADDRESS_TY数据类型和CUSTOMER表。如果数据类型的拥有者不是表的拥有者又该怎么办呢？如果另一用户想根据你已创建的数据类型来创建一个数据类型，那又该怎么办呢？在开发环境中，必须建立所有权和使用抽象数据类型的准则。

例如，如果帐户DORA拥有ADDRESS_TY数据类型，该帐户的用户GEORGE试图创建一个PERSON_TY数据类型，该怎么办？GEORGE可执行下列命令：

```
create type PERSON_TY as object
(Name      VARCHAR2(25),
 Address   ADDRESS_TY);
/
```

如果GEORGE不拥有ADDRESS_TY抽象数据类型，Oracle将用下列消息来响应create type命令：

```
Warning: Type created with compilation errors.
```

当创建数据类型时，生成构造方法的问题引发了编译错误。由于GEORGE不拥有具有该名字的数据类型，所以Oracle不能解决对ADDRESS_TY数据类型的引用。他可再次发出create type命令（使用or replace子句）以明确引用DORA的ADDRESS_TY数据类型。

```
create or replace type PERSON_TY as object
(Name      VARCHAR2(25),
 Address   Dora.ADDRESS_TY);
/

Warning: Type created with compilation errors.
```

要查看与数据类型的创建有关的错误，可使用show errors命令，如下所示：

```
show errors
Errors for TYPE PERSON_TY:
LINE/COL ERROR
-------- -----------------------------------------------------------
0/0      PL/SQL: Compilation unit analysis terminated
3/11     PLS-00201: identifier 'DORA.ADDRESS_TY' must be declared
```

GEORGE将不能创建PERSON_TY数据类型（它包含ADDRESS_TY数据类型），除非DORA首先在她的类型中授予他EXECUTE权限。下列代码示出了这种授权：

```
grant EXECUTE on ADDRESS_TY to George;
```

现在已经正确授权，GEORGE可以创建一个基于DORA的ADDRESS_TY数据类型的数据类型。

```
create or replace type PERSON_TY as object
(Name      VARCHAR2(25),
 Address   Dora.ADDRESS_TY);
/
```

现在可以成功创建GEORGE的PERSON_TY数据类型。然而，使用基于另一用户的数据类型的数据类型并不是一件简单的事。例如，在insert操作期间，必须全部指定每一类型的拥有者的名字。GEORGE能够创建一个基于PERSON_TY数据类型（它包含DORA的ADDRESS_TY数据类型）的表，如下所示：

```
create table GEORGE_CUSTOMERS
(Customer_ID  NUMBER,
 Person       PERSON_TY);
```

如果GEORGE拥有PERSON_TY和ADDRESS_TY数据类型，那么插入到CUSTOMER可使用如下格式：

```
insert into GEORGE_CUSTOMERS values
(1,PERSON_TY('SomeName',
   ADDRESS_TY('StreetValue','CityValue','ST',11111)));
```

由于GEORGE不拥有ADDRESS_TY数据类型，该命令将失败。在insert期间，ADDRESS_TY构造方法被使用，DORA拥有它。因此，必须修改insert命令以指定DORA作为ADDRESS_TY的拥有者。下列例子示出了修改后的insert语句，黑体字所示是对DORA的引用：

```
insert into GEORGE_CUSTOMERS values
(1,PERSON_TY('SomeName',
   Dora.ADDRESS_TY('StreetValue','CityValue','ST',11111)));
```

GEORGE能使用DORA的数据类型的同义词吗？不能。 GEORGE可以创建一个名为ADDRESS_TY的同义词：

```
create synonym ADDRESS_TY for Dora.ADDRESS_TY;
```

但是这一同义词不能使用：

```
create type PERSON2_TY as object
(Name      VARCHAR2(25),
 Address   ADDRESS_TY);
/

create type PERSON2_TY as object
*
ERROR at line 1:
ORA-22863: synonym for datatype DORA.ADDRESS_TY not allowed
```

正如错误消息所示，不能使用另一用户的数据类型的同义词。因此，在每次 insert命令时，必须引用数据类型的拥有者。

注意 当创建同义词时，Oracle并不检查正为其创建同义词的对象的有效性。如果创建synonym x for y, Oracle将不检查该“ y ”是否是有效对象名或有效对象类型。只有通过同义词访问该对象时，才检查该对象的通过同义词访问的有效性。

在Oracle的关系实现中，必须授予过程对象上的 EXECUTE权限，例如过程和包。在Oracle的对象关系实现中，扩展了 EXECUTE权限以包含抽象数据类型。由于抽象数据类型可能包含在该数据类型中运算的 PL/SQL函数和过程，所以要使用 EXECUTE权限。如果你授予了某用户使用你的数据类型的权限，那么你也授予了该用户执行你在该数据类型中定义的方法的权限。因此，要授予的合适权限是 EXECUTE。虽然DORA还没有定义 ADDRESS_TY上的任何方法，但 Oracle将自动创建称为构造方法的特定过程，该方法用来访问数据。使用ADDRESS_TY数据类型的任何对象（例如 PERSON_TY）都使用与ADDRESS_TY有关的构造方法。因此，尽管没有为抽象数据类型创建任何方法，仍然有与该数据类型有关的过程。

不能创建公共的数据类型，也不能创建数据类型的公共同义词。因此，需要引用数据类型的拥有者，或者在每个能在数据库中创建表的帐户之下创建这个数据类型。这两者都不是数据类型管理问题的简单解决办法。

24. 检索抽象数据类型的属性

在上述例子中， GEORGE_CUSTOMERS表是基于 PERSON_TY数据类型和ADDRESS_TY数据类型创建的。正如下列程序段所示， GEORGE_CUSTOMERS表含有一个常规列 —— Customer_ID和一个由PERSON_TY抽象数据类型定义的Person列：

```
create table GEORGE_CUSTOMERS
(Customer_ID    NUMBER,
 Person         PERSON_TY);
```

从本章上一节的数据类型定义中可以看出， PERSON_TY有一个Name列，它后面跟着一个由ADDRESS_TY数据类型定义的Address列。

当在查询、更新和删除操作期间引用抽象数据类型中的列时，必须指定到数据类型属性的全路径。下列查询返回 Customer_ID列和Name列。 Name列是定义Person列的数据类型的属性，因此把该属性称为 Person.Name。

```
select C.Customer_ID, C.Person.Name
  from GEORGE_CUSTOMERS C;
```

通过指定通向相关列的全路径可以引用 ADDRESS_TY数据类型中的属性。例如， Street列被引用为Person.Address.Street，这完全描述了它在表结构中的位置。在下列例子中， City列被引用两次——一次在要选择的列的列表中，另一次在 where子句中。

```
select C.Person.Name,
       C.Person.Address.City
  from GEORGE_CUSTOMERS C
 where C.Person.Address.City like 'C%';
```

由于City列与where子句中的范围搜索一起使用，因此当分析查询时， Oracle优化程序可以使用索引。如果在 City列中有索引，那么 Oracle就能快速找出具有特定 City值的所有行，这些City值按查询的要求以字母“ f ”打头。

要在是抽象数据类型一部分的列上创建一个索引，必须指定到该列的全路径作为 create index命令的一部分。要创建 City列（它是 Address列的一部分）上的一个索引，可执行下列命令：

```
create index I_GEORGE_CUSTOMERS$CITY
on GEORGE_CUSTOMERS(Person.Address.City);
```

该命令将在 Person.Address.City列上创建一个名为 I_GEORGE_CUSTOMERS$CITY的索引。每当访问 City列时， Oracle优化程序就估计用于访问数据的 SQL并确定新索引是否可改善访问性能。

当创建基于抽象数据类型的表时，应当考虑如何访问抽象数据类型中的列。像上例中的 City列，如果某些列被用作查询中的限制条件的一部分，那么这些列应被索引。在这方面 ,在一个抽象数据类型中表示多个列可能会妨碍应用程序的性能，因为它可能闹不清楚是否需要索引数据类型中的特定列。

当使用抽象数据类型时，你会习惯于把一组列作为一个实体来处理，例如 Address列或 Person列。重要的是要记住，当估计查询的访问路径时，优化程序将逐一考虑各个列。因此当使用抽象数据类型时，必须提出这些列的索引要求。此外，索引一个使用 ADDRESS_TY数据类型的表中的 City列并不对另二个使用 ADDRESS_TY数据类型的表中的 City列产生什么影响。例如，如果存在着另二个使用 ADDRESS_TY数据类型的表 BRANCH，那么它的City列将不被索引，除非你创建了一个对它的索引。事实是，对 CUSTOMER表中的City列的索引不会对BRANCH表中的City列产生什么影响。

5.3.6 迭代开发

迭代开发法（iterative development methodology）通常由一系列快速开发原型组成。这些原型用于在系统开发时定义系统的要求。由于它们能够在开发过程中给用户展示一些实实在在的性能，所以有着较大的吸引力。然而，在迭代开发过程中，也会出现一些损害性能的潜在隐患。

首先，不总是使用高效的版本。我们知道，生成一个应用程序的多个版本允许在其他功能发生变化时“ 冻结 ”某些特征。它也允许应用程序的不同部分处在不同的阶段，有的在开发中，而其他部分则在测试中。而在迭代开发中，会经常出现应用程序的一个版本被用于每一种性能的每一次重复，这将导致最终产品缺乏足够的灵活性来处理那些（迭代开发提出的）变化要求。

第二，不能抛弃原型。原型的开发给用户一个概念，即原型的表现便是最终产品的表现。

实际上，这些原型不应当作为最终产品的基础。若将它们作为基础，将不能产生最强壮而灵活的系统。当进行迭代开发时，可将原型作为一个临时的将要放弃的系统。不应希望创建出一个基于要放弃的系统的优质新系统。

第三，开发/测试/产品的划分是模糊的，迭代开发方法必须非常明确地定义一个应用程序版本转移到下一个阶段之前必须满足的条件。所以，将原型的开发与全部应用程序的开发完全隔离开来，可能是最好的。

最后，要经常设置虚构的时间线。适用于构造方法的交付方案也同样适用于迭代开发方法。事实是，快速开发的应用程序并不意味着可以很快交付使用。

5.3.7 迭代列定义

在开发过程中，可能要频繁改变列定义。在 Oracle8i中，可从现有表中撤消列，这种功能在以前是不能使用的。可以直接撤消列，也可以把它标记为“不使用”而在以后撤消它。如果直接撤消列，则这种操作可能会影响性能。如果把该列标记为“不使用”，就对性能没有什么影响。实际上可以在以后数据库活动不频繁时撤消列。

要撤消一个列，可使用 alter table命令中的set unused子句或drop子句。不能撤消伪列、嵌套表中的列或分区键列。有关 alter table命令的完整语法和限制，请参见附录 A。

在下列的例子中，Col2列被从TABLE1表中撤消：

```
alter table TABLE1 drop column Col2;
```

可以把一个列标记为“不使用”：

```
alter table TABLE1 set unused column Col3;
```

把一个列标记为“不使用”并不释放先前由该列占用的空间。可以撤消不使用的列：

```
alter table TABLE1 drop unused columns;
```

可以查询 USER_UNUSED_COL_TABS、DBA_UNUSED_COL和ALL_UNUSED_COL_TABS来查看具有标记为不使用的列的所有表。

注意 一旦把一个列标记为“不使用”，就不能访问该列。

可以撤消一个命令中的多个列，如下所示：

```
alter table TABLE1 drop (Col4, Col5);
```

注意 当撤消多个列时，不应使用 alter table命令的column关键字，它会引起语法错误。如上所示，多个列的名字应括在括号中。

如果撤消的列是主键的一部分或唯一约束，那么还必须使用 cascade constraints子句作为 alter table命令的一部分。如果撤消一个属于主键的列，Oracle就将同时撤消该列和该主键索引。

5.4 管理技术

在开发进行的过程中，必须要建立有效的中间交付方案。由于大多数开发小组包含了多个开发人员（至少有一个 DBA），所以也就必须要建立一些通信方式。这些通信方式有助于保持计划和实现的一致性。

要使这一方法能有效工作，需要 4个技术方案。目前还没有有效的集成产品开发包可供使用，所以不得不需要 4个不同的方案。它们分别是 CASE工具、共享目录、项目管理数据库和

讨论数据库。

5.4.1 CASE工具

CASE工具可以用来生成实体关系图表和物理数据库图表。 Oracle Designer是一个可以生成实体关系图表的多用户CASE工具，并拥有一个集成的数据字典。它允许跨应用程序间的实体共享，并能存储表中行与列的尺寸信息。这有利于解决若干在本章中定义的交付问题。Designer的多用户功能有助于确保开发人员间的一致性。它也允许对不同版本的数据模型进行维护或冻结。

创建应用程序的数据库对象的SQL命令应直接从CASE工具中生成。也可使用CASE生成基于定义的数据库对象的应用程序的普通版本。

5.4.2 共享目录

一些交付要求(如备份要求等)没有专门的工具来创建它们。这些交付要求必须用现场中的任何最合适、有效的工具来建立。结果文件必须存储在共享的项目目录中，以便开发小组中的所有人员都可以访问它们。这些文件的格式和命名协议必须在开发过程前期指定。

5.4.3 项目管理数据库

为了就应用程序的当前状态和它的交付要求与开发小组以外的人员进行通信，需要维护一个项目管理数据库。这个数据库必须能够向外部人员提供项目视图和当前历程。这将使得系统管理员等非项目开发人员也可提出进一步的产品要求。项目管理数据库也允许对针对关键历程所做的改变或延迟造成的影响进行分析。这种分析可能导致修改项目任务所分配的资源级。

5.4.4 讨论数据库

在前面的三个共享区域（CASE工具、共享目录和项目管理数据库）中，大多数信息代表了一种默契的观点。例如，一些开发小组成员可能有一些关于备份策略的想法，系统管理员与DBA也加进来讨论。要进行这种交流，就需要创建一系列讨论数据库（这通常通过在一个局域网中使用组件产品来实现）。这样草案就可以在最终方案被放置在共享交付目录中之前，被传递到这些讨论区域供大家讨论。

5.5 管理包开发

假设一个开发环境具有下列特性：

- 没有一个标准被执行。
- 在SYS或SYSTEM帐户下创建对象。
- 对表和索引的大小计算及正确分配考虑甚少。
- 每一个应用程序的设计都好像它是数据库中唯一运行的应用程序。

如果是这样，请进行包管理。

正确管理包的实现包含了许多与前几节中对应用程序开发过程所描述的问题相同的问题。本节就如何处理包使其能最好地适合开发环境进行一个大概的讨论。

5.5.1 生成图表

大多数CASE工具能够将工程包转变成一个物理数据库图表。这包含了对表结构的分析并生成与表结构相一致的物理数据库图表，通常是通过分析列名称和索引来确定键列而实现的。然而，通常并不存在物理数据库图表与实体关系图表之间的一一对应。包的实体图表一般可以从经销商得到，它们有助于包数据库接口的规划。

5.5.2 空间需求

大多数基于Oracle的包在产品使用期间提供对数据库资源使用情况的准确估计。然而，在数据装载和软件更新时，它们却对资源使用要求的计算无能为力。正是因为这个原因，创建一个特殊回滚段表空间(RBS_2)用于处理大量数据装载是十分明智的。在更新操作期间，当数据包生成它的所有表的拷贝时，有可能需要一个备用数据表空间。有关包管理准则的更多信息，请参见第11章。

5.5.3 调整目标

正如定制的应用程序需要调整目标一样，包也同样需要调整目标。设置和跟踪这些控制值有利于识别需要调整的包区域（见第8章）。

5.5.4 安全需求

不幸的是，大多数使用Oracle数据库的包都属于如下两个类别之一： 1)这些包从另一数据库系统移植到Oracle ；2)它们假设对它们的对象拥有者帐户具有全部DBA权限。

如果首先在另一个数据库系统中创建包，则它们的 Oracle 端口很有可能无法充分利用Oracle 的功能特性。这些功能包括行级的锁定和队列、触发器以及方法的使用。对这样一个包进行调整以满足要求，可能需要对源代码进行修改。

如果假设包具有全部的 DBA权限，那么它将不能与其他重要的数据库应用程序存储在同一个数据库中。大多数要求拥有 DBA权限的包这样做，是为了向数据库中添加新用户。必须明确确定什么样的系统权限才是包管理员帐户所实际需要的 (通常仅仅是 CREATE SESSION和CREATE USER)。可以创建一个专门的系统级角色以便为包管理员提供一些有限的系统权限。

那些首先在非Oracle 数据库中开发的包有可能要求与另一个 Oracle 包使用同一个帐户。例如，称为SYSADM 的数据库帐户的拥有权可能同时被多个应用程序所要求。解决此冲突的唯一方法便是在不同的数据库中建立这两个包。

5.5.5 数据需求

必须明确定义包所具有的所有处理要求，特别是在数据输入方面。这通常是在包文档中进行记录。

5.5.6 版本要求

你所支持的应用程序可能依赖于Oracle的特定版本和特性。例如，封装的应用程序在版本

8.0.5.1和8.1.6中可以得到认证，但在版本 8.1.5却不能认证。如果使用封装的应用程序，则应根据不同Oracle版本的供应商支持来制定核心版本升级计划。此外，供应商可能会更换它所支持的优化程序 —— 例如，在版本 8.0.5.1中基于规则，但在 8.1.6中基于成本。你的数据库环境必须尽可能灵活以便支持这些变化。

由于这些限制不受你控制，应当设法把应用程序与其自身的实例隔离开来。如果要频繁地在应用程序之间查询数据，应用程序与其自身实例的隔离会增加你对数据库链接的信任。需要对照估计支持多个实例的维护成本和支持一个实例中多个应用程序的维护成本。

5.5.7 执行规划

生成执行规划需要访问那些以数据库为运行背景的 SQL语句。SGA中的SQL共享区域(见第1章和第6章)保留着要对数据库执行的 SQL语句。使这些SQL语句同应用程序的特定部分匹配是一个耗时的过程。所以最好识别出其功能和性能对应用程序的成功至关重要的特定区域。并且与包支持小组一起解决性能上的问题。

5.5.8 验收测试过程

通常在应用程序已经安装后，对包进行验收测试。然而，包也应当满足定制的应用程序必须满足的功能要求。因此验收测试过程应当在选择包以前进行开发，这样它们可以从包的选择准则中生成。通过这种方式的测试，可以对实际所需的功能而不是包的开发者认为所需的功能进行测试。

在包由于功能上或性能上的原因而验收测试失败时，要确信自己的选择。不要因为是买来的应用程序而忽略了应用程序成功与否的重要因素。

5.5.9 测试环境

当建立测试环境时，请遵循下列准则：

1) 必须大于产品环境，必须能够预测未来的性能。

2) 必须包含已知数据集、说明计划、性能结果和数据结果集。

3) 必须用于数据库和工具的每个版本以及新的特性。

4) 必须支持多测试条件的生成以便估计特性的业务成本。不能只依赖于对结果的点分析。理想情况下，应能确定当数据库尺寸增大时特性的成本／效益曲线。

5) 必须足够灵活，以便估计不同的认证成本选项。

6) 必须主动地用作技术实现方法的一部分。

当实现和执行测试环境时，并不是简单地跟踪执行路径和数据库中每次查询的执行情况。可以使用两种传统的统计技术 —— 分组和抽样 —— 来执行测试。

要有效地测试执行结果必须对一系列查询和操作进行分组并把组作为一个整体来测试。如果该组的性能不符合期望值，则可测试该组的各个部分。例如，可以把涉及数据仓库的数据装载的所有操作分为一个组。如果该组操作的性能符合测试准则，那么就不必估计该组中每个操作的执行情况。如果该组的性能不符合你的期望，那么可把该组细分为更小的组并进一步隔离问题。通过分组操作，可以大大简化识别问题区域及其原因的过程。

可以使用抽样来确定改变那些不易分组的操作造成的影响。例如，可以从数据库中随机

采集样本查询（手工采集或通过 V$SQLAREA视图的SQL_Text列）并运行它们来确定其预期结果和说明计划。然后在你的环境中执行这些查询并把这些查询的结果与预期结果进行比较。当调查大量的人口（如全国的人口）时，传统的抽样比例可选定为 1/300000。对于一个具有成千上万个查询的数据库，必须在抽样集中用不到 100个SQL语句来代表最常见的查询。

抽样集不应完全随机，它应代表下列每一个组：

- 执行连结的查询，至少包含两个相互代表的合并连结、嵌套循环、外部连结和散列连结。
- 使用数据库链接的查询。
- 使用数据库链接的DML。
- 至少两个DML语句类型（insert、update和delete）。
- 至少一个主要的DDL语句类型，包括表创建、索引重建、分析操作和授权。
- 至少一个使用 Parallel Query Option（并行查询选项）的查询 (如果在你的环境中使用了该选项)。

不应伪造采样集，它应代表你的操作。因此，生成采样集应包括检查主要的操作组和由用户执行的OLTP操作。其结果并不反映数据库中的每一个操作，但可让你了解实际情况，因此可减少风险并做出关于实现新选项的更好决定。

5.6 管理环境

实现这三个重要因素 —— 培植过程、管理过程、开发方法 —— 将会产生一个适宜进行质量控制的开发环境。这也允许在开发过程中进行改进。这样，产品应用程序将会性能更佳、维护更简化、与其他企业应用程序的集成更好。