Oracle Database 11g DBA手册

第Ⅰ部分　数据库体系结构

第III部分 高可 用 性

前言

无论是有经验的 DBA、DBA 新手或者是应用程序开发人员，都需要了解 Oracle 11g 的新特性，以最好地满足顾客的需求。本书将介绍这些最新的特性以及如何将这些特性结合到 Oracle 数据库管理中。贯穿本书的重点是以实际而有效的方式管理数据库的功能，从而交付高质量的产品。最终的结果将是产生一个可靠的、健壮的、安全的和可扩展的数据库。

有些元素对于实现这个目标至关重要。本书第Ⅰ部分中，在介绍了 Oracle 体系结构、Oracle 11g 升级问题以及表空间计划后，将深入探讨这些元素。设计良好的逻辑和物理数据库体系结构将通过适当地分布数据库对象来改进性能并简化管理。本书的第Ⅱ部分将介绍针对单机和网络数据库的适当监控、安全性和调整策略。本书也介绍了用于帮助确保数据库可恢复性的备份和恢复策略。每一章节都关注相应的特性以及每个领域的专门的计划和管理技术。

本书的第Ⅲ部分将深入介绍所有相关方面的高可用性：实时应用集群(RAC)、恢复管理器(RMAN)以及 Oracle Data Guard，并且将这些方面作为第三部分中各章节的标题。

本书也详尽地介绍了网络化问题以及分布式数据库和客户端/服务器数据库的管理。本书的第Ⅳ部分详细讨论了 Oracle Net、网络化配置、物化视图、位置透明性以及其他方面的内容，我们需要这些内容来成功实现分布式数据库或客户端/服务器数据库。这一部分也介绍了一些针对每个主要配置的实际示例。

除了执行 DBA 活动所需要的命令外，本书还介绍了 Oracle Enterprise Manager，据此可以执行类似的功能。根据本书中介绍的技术，可以很好地设计并实现自己的系统，从而最小化调整工作。数据库管理工作也将变得更为简单，同时用户还可以获得更好的产品，数据库也可以运作得更良好。

DBA 配置数据库中表空间布局的方式会直接影响到数据库的性能和可管理性。本章将回顾不同类型的表空间，以及如何利用 Oracle 10g 中新增的临时表空间组特性，使用临时表空间来推动数据库中表空间的大小和数量。

本章还将介绍 Oracle 的优化灵活体系结构(Optimal Flexible Architecture，OFA)如何使 Oracle 可执行文件和数据库文件自身的目录结构标准化。Oracle 从版本 7 开始支持 OFA。Oracle Database 11g 进一步增强了 OFA 的功能，补充了 OFA 最初的提高性能的作用，增强了安全性并简化了复制和升级任务。

Oracle 的默认安装为 DBA 提供了良好的起点，不仅创建了遵从 OFA 的目录结构，而且根据各个段的功能将它们分离到大量表空间中。本章将回顾每个表空间的空间需求，并且介绍一些关于如何微调这些表空间特征的技巧。

本章的末尾将提供一些指导原则，帮助您基于类型、大小和访问频率来将段放入不同的表空间中，同时还提供了一些方法来标识一个或多个表空间中的热点。

3.1 表空间的体系结构

在数据库中完全设置表空间的先决条件是理解不同类型的表空间，以及如何将它们用于 Oracle 数据库中。本节将介绍不同类型的表空间，并且给出一些管理它们的示例。

此外，本节将概述 Oracle 的优化灵活体系结构，并且介绍它如何提供框架来存储表空间数据文件以及 Oracle 可执行文件和其他 Oracle 组件，例如重做日志文件、控制文件等。本节将分类回顾不同类型的表空间(SYSTEM 表空间、SYSAUX 表空间、临时表空间、撤销表空间和大文件表空间)并且描述它们的功能。

3.1.1 表空间类型

Oracle 数据库中主要的表空间类型有：永久表空间、撤销表空间和临时表空间。永久表空间包含一些段，这些段在超出会话或事务的持续时间后持续存在。

虽然撤销表空间可能有一些段在超出会话或事务末尾后仍然保留，但它为访问被修改表的 select 语句提供读一致性，同时为数据库的大量闪回特性提供撤销数据。然而，撤销段主要用来存储一些列在更新或删除前的值，或者用于提供指示，表明不存在用于插入的行。如果用户的会话在用户发出 commit 或 rollback 前失败，则取消更新、插入和删除。用户的会话永远不可以直接访问撤销段，并且撤销表空间可能只有撤销段。

顾名思义，临时表空间包含暂时的数据，这些数据只存在于会话的持续时间，例如完成分类操作的空间不适合来自于内存。

大文件表空间可用于这 3 类表空间的任何一种，大文件表空间将维护点从数据文件移动到表空间，从而简化了表空间的管理。大文件表空间只包含一个数据文件。大文件表空间也有一些缺点，本章后文中会介绍这些缺点。

1. 永久表空间

SYSTEM 表空间和 SYSAUX 表空间是永久表空间的两个示例。此外，任何在超出会话或事务边界后需要由用户或应用程序保留的段都应该存储在永久表空间中。

SYSTEM 表空间　用户段绝对不应该驻留在 SYSTEM 表空间中。从 Oracle 10g 开始，除了保留 Oracle 9i 中指定默认临时表空间的能力外，还可以指定默认的永久表空间。

如果使用 Oracle 通用安装程序(Oracle Universal Installer，OUI)来创建数据库，则会为永久段和临时段创建不同于 SYSTEM 的单独表空间。如果手动创建数据库，则要确保指定默认永久表空间和默认临时表空间，如同下面的样例 create database 命令。

```
CREATE DATABASE rjbdb
   USER SYS IDENTIFIED BY kshelt25
   USER SYSTEM IDENTIFIED BY mgrab45
   LOGFILE GROUP 1 ('/u02/oracle11g/oradata/rjbdb/redo01.log') SIZE 100M,
           GROUP 2 ('/u04/oracle11g/oradata/rjbdb/redo02.log') SIZE 100M,
           GROUP 3 ('/u06/oracle11g/oradata/rjbdb/redo03.log') SIZE 100M
   MAXLOGFILES 5
   MAXLOGMEMBERS 5
   MAXLOGHISTORY 1
   MAXDATAFILES 100
   MAXINSTANCES 1
   CHARACTER SET US7ASCII
   NATIONAL CHARACTER SET AL16UTF16
   DATAFILE '/u01/oracle11g/oradata/rjbdb/system01.dbf' SIZE 325M REUSE
   EXTENT MANAGEMENT LOCAL
   SYSAUX DATAFILE '/u01/oracle11g/oradata/rjbdb/sysaux01.dbf'
      SIZE 325M REUSE
   DEFAULT TABLESPACE USERS
      DATAFILE '/u03/oracle11g/oradata/rjbdb/users01.dbf'
      SIZE 50M REUSE
```

```
DEFAULT TEMPORARY TABLESPACE tempts1
   TEMPFILE '/u01/oracle11g/oradata/rjbdb/temp01.dbf'
   SIZE 20M REUSE
UNDO TABLESPACE undotbs
   DATAFILE '/u02/oracle11g/oradata/rjbdb/undotbs01.dbf'
   SIZE 200M REUSE AUTOEXTEND ON MAXSIZE UNLIMITED;
```

从 Oracle 10g 开始，SYSTEM 表空间默认为本地管理。换句话说，所有的表空间使用由位图段管理，位图段在表空间的第一个数据文件的第一个部分中。在本地管理 SYSTEM 表空间的数据库中，数据库中的其他表空间也必须被本地管理，或者必须是只读的。使用本地管理的表空间可免除一些 SYSTEM 表空间的争用，因为表空间的空间分配和释放操作不需要使用数据字典表。关于本地管理表空间的更多细节将在第 6 章中介绍。

SYSAUX 表空间　类似于 SYSTEM 表空间，SYSAUX 表空间不应该有任何用户段。SYSAUX 表空间的内容根据应用程序划分，可以使用 EM 数据库控制台(Database Control)进行查看。通过单击 Server 选项卡下面的 Tablespaces 链接，并单击 SYSAUX 链接，可以编辑 SYSAUX 表空间。图 3-1 为 SYSAUX 中空间使用的图形表示。

图 3-1　EM 数据库控制台中的 SYSAUX 表空间内容

如果驻留在 SYSAUX 表空间中的特定应用程序的空间使用率过高，或者由于与其他使用 SYSAUX 表空间的应用程序严重争用表空间而造成了 I/O 瓶颈，那么可以将这些应用程序中的一个或多个移动到不同的表空间。将如图 3-1 所示的屏幕向下滚动，可以单击一个 SYSAUX 占用者的 Change Tablespace 链接，将此 SYSAUX 占用者移动到另一个表空间，如图 3-2 所示。第 6 章将介绍使用命令行界面将 SYSAUX 占用者移动到不同表空间的示例。

图 3-2　使用 EM 数据库控制台移动 SYSAUX 占用者

可以像监控其他表空间一样监控 SYSAUX 表空间。本章后面将介绍如何使用 EM 数据库控制台来标识表空间中的热点。

2. 撤销表空间

多个撤销表空间可以存在于一个数据库中，但在任何给定的时间内只有一个撤销表空间可以是活动的。撤销表空间用于回滚事务，以及提供与 DML 语句同时运行在相同的表或表集上的 select 语句的读一致性，并支持大量 Oracle 闪回特性，例如闪回查询(Flashback Query)。

撤销表空间需要正确地确定大小，从而防止“Snapshot too old”错误，并且提供足够的空间来支持初始参数，例如 UNDO_RETENTION。关于如何监控、确定大小和创建撤销表空间的更多信息将在第 7 章介绍。

3. 临时表空间

数据库中可以有多个临时表空间联机并处于活动状态，但在 Oracle 10g 之前，同一个用户的多个会话只可以使用同一个临时表空间，因为只有一个默认的临时表空间可以被赋予用户。为了解决这个潜在的性能瓶颈，Oracle 现在支持临时表空间组。临时表空间组即为一系列临时表空间。

临时表空间组必须至少包含一个临时表空间，它不可以为空。一旦临时表空间组没有任何成员，它将不再存在。

使用临时表空间组的一个最大优点是，向具有多个会话的单个用户提供如下功能：对每个会话使用不同的实际临时表空间。在如图 3-3 所示的图表中，用户 OE 具有两个活动会话，这些会话需要临时的表空间来执行分类操作。

图 3-3　临时表空间组 TEMPGRP

并不是将单个临时表空间赋给用户，而是赋予临时表空间组。在这个示例中，将临时表空间组 TEMPGRP 赋给 OE。因为 TEMPGRP 临时表空间组中有 3 个实际的临时表空间，所以第一个 OE 会话可以使用临时表空间 TEMP1，第二个 OE 会话执行的 select 语句可以并行使用其他两个临时表空间 TEMP2 和 TEMP3。在 Oracle 10g 以前，两个会话都使用相同的临时表空间，从而潜在地造成性能问题。

创建临时表空间组非常简单。创建单独的表空间 TEMP1、TEMP2 和 TEMP3 后，可以创建名为 TEMPGRP 的临时表空间组，具体如下：

```
SQL> alter tablespace temp1 tablespace group tempgrp;
Tablespace altered.
SQL> alter tablespace temp2 tablespace group tempgrp;
Tablespace altered.
SQL> alter tablespace temp3 tablespace group tempgrp;
Tablespace altered.
```

使用将实际临时表空间改为默认临时表空间的相同命令，可以将数据库的默认临时表空间改为 TEMPGRP。临时表空间组逻辑上可视为与一个临时表空间相同。

```
SQL> alter database default temporary tablespace tempgrp;
Database altered.
```

为了删除表空间组，必须先删除它的所有成员。对组中的临时表空间赋予空字符串(取消组中的表空间)，即可删除表空间组的成员：

```
SQL> alter tablespace temp3 tablespace group '';
Tablespace altered.
```

将临时表空间组赋给用户等同于将一个临时表空间赋给用户，这种分配可以发生在创建用户时或者将来的某个时刻。下面的示例表示将新用户 JENWEB 赋给临时表空间 TEMPGRP：

```
SQL> create user jenweb identified by pi4001
  2       default tablespace users
  3       temporary tablespace tempgrp;
User created.
```

注意，如果在创建用户期间没有分配表空间，将仍然向用户 JENWEB 赋予 TEMPGRP 作为临时表空间，因为根据前面的 create database 示例，这是数据库默认的临时表空间。

在 Oracle Database 10g 和 Oracle Database 11g 中，对数据字典视图进行了一些改动以支持临时表空间组。与 Oracle 以前的版本一样，数据字典视图 DBA_USERS 仍然具有列 TEMPORARY_TABLESPACE，但该列现在可以包含赋给用户的临时表空间的名称，或者是临时表空间组的名称。

```
SQL> select username, default_tablespace, temporary_tablespace
  2       from dba_users where username = 'JENWEB';

USERNAME             DEFAULT_TABLESPACE TEMPORARY_TABLESPACE
-------------------- ------------------ --------------------
JENWEB               USERS              TEMPGRP

1 row selected.
```

新的数据字典视图 DBA_TABLESPACE_GROUPS 显示了每个临时表空间组的成员：

```
SQL> select group_name, tablespace_name from dba_tablespace_groups;
GROUP_NAME                     TABLESPACE_NAME
------------------------------ ------------------------------
TEMPGRP                        TEMP1
TEMPGRP                        TEMP2
TEMPGRP                        TEMP3
3 rows selected.
```

和其他大多数可以使用命令行实现的 Oracle 特性一样，可以使用 EM 数据库控制台将成员赋给临时表空间组，也可以从临时表空间组中取消成员。在图 3-4 中，可以从临时表空间组中添加或取消成员。

4. 大文件表空间

大文件表空间减轻了数据库管理，因为它只包含一个数据文件。如果表空间块大小是 32KB，则该数据文件的大小最多可以为 128TB。前面许多只可用于维护数据文件的命令现在都可以用于表空间，只要表空间是大文件表空间即可。第 6 章将介绍如何创建和维护大文件表空间。

虽然大文件表空间的维护很方便，但大文件表空间也存在一些潜在的缺点。因为大文件表空间是单一的数据文件，所以完全备份单一的一个大型数据文件所花的时间比完全备份多个小型数据文件(这些小型数据文件的总大小与单一数据文件表空间相等)要长得多，这是因为在 Oracle 中，每个数据文件只使用一个从进程，因此不能使用并行进程备份大文件表空间的不同部分。如果大文件表空间是只读的，或者只定期备份已改变的块，则备份问题在这种环境中也许并不突出。

图 3-4　使用 EM 数据库控制台编辑临时表空间组

3.1.2　优化灵活体系结构

Oracle 的优化灵活体系结构(OFA)提供了减轻 Oracle 软件和数据库文件维护工作的指导原则，并且改进了数据库的性能，即适当地放置数据库文件，从而最小化 I/O 瓶颈。

安装或维护 Oracle 环境时，虽然并不严格要求使用 OFA，但使用 OFA 可以使客户更容易理解如何在磁盘上组织数据库，从而可以防止客户在您度假时午夜打电话给您。

根据所使用的存储器选项类型，OFA 具有细微的区别—— 或者是自动存储管理(Automatic Storage Management，ASM)环境，或者是标准的操作系统文件系统，该系统使用(或者不使用)第三方逻辑卷管理器或支持 RAID 的磁盘子系统。

1. 非 ASM 环境

在 UNIX 服务器的非 ASM 环境中，单独的物理设备上至少要求有 3 个文件系统才能实现 OFA 的推荐标准。从顶层开始，安装点的推荐格式是/<*string const*><*numeric key*>，其中<*string const*>可以是一个或多个字母，<*numeric key*>是两个或三个数字。例如，在某个系统上，可以有安装点/u01、/u02、/u03 和/u04，利用空间可以扩展到额外的 96 个安装点，且不需要改变文件命名约定。图 3-5 显示了典型的 UNIX 文件系统布局，它具有遵从 OFA 的 Oracle 目录结构。

该服务器上有两个实例：管理磁盘组的 ASM 实例和标准的 RDBMS 实例(dw)。

图 3-5　遵从 OFA 的 Oracle 目录结构

软件可执行文件　每个单独产品名的软件可执行文件驻留在目录/<*string const*><*numeric key*>/<*directory type*>/<*product owner*>中，其中<*string const*>和<*numeric key*>在前面已经定义过，<*directory type*>包含安装在这个目录上的文件类型，<*product owner*>则是拥有并在这个目录上安装文件的用户名。例如，/u01/app/oracle 可以包含由用户 Oracle 在服务器上安装的应用程序相关文件(可执行文件)。目录/u01/app/apache 可以包含从前一个 Oracle 版本中安装的中间件 Web 服务器的可执行文件。

从 Oracle 10g 开始,OFA 标准使 DBA 更容易在相同的高级目录中安装多个版本的数据库和客户端软件。遵从 OFA 的 Oracle 主路径对应于环境变量 ORACLE_HOME，该路径包含一个后缀，对应于安装类型和具体化。例如，Oracle 11g 的一个安装、Oracle 10g 的两个不同安装和 Oracle 9i 的一个安装可以驻留在下面 3 个目录中：

/u01/app/oracle/product/9.2.0.1
/u01/app/oracle/product/10.1.0/db_1
/u01/app/oracle/product/10.1.0/db_2
/u01/app/oracle/product/11.1.0/db_1

同时，Oracle 客户端可执行文件和配置可以存储在与数据库可执行文件相同的父目录中：

/u01/app/oracle/product/10.1.0/client_1

一些安装目录将永远不会有一个给定产品的多个实例。例如，Oracle 的集群就绪服务(Cluster Ready Services，CRS)将安装在下面的目录中(给定前面的安装)：

/u01/app/oracle/product/11.1.0/crs

因为 CRS 一次只可以安装在一个系统上，所以它没有自增的数字后缀。

数据库文件　任何非 ASM 的 Oracle 数据文件都驻留在/<*mount point*>/oradata/<*database name*>中，其中<*mount point*>是前面讨论的一种安装点，<*database name*>是初始参数 DB_NAME 的值。例如，/u02/oradata/rac0 和/u03/oradata/rac0 可以包含实例 rac0 的非 ASM 控制文件、重做日志文件和数据文件，而/u05/oradata/dev1 可以包含相同服务器上实例 dev1 的相同文件。表 3-1 详述了 oradata 目录下不同文件类型的命名约定。

表 3-1　遵从 OFA 的控制文件、重做日志文件和数据文件命名约定

文 件 类 型	文件名格式	变　量
控制文件	control.ctl	没有
重做日志文件	redo<*n*>.log	*n* 是两位数的数字
数据文件	<*tn*>.dbf	*t* 是 Oracle 表空间名，*n* 是两位数的数字

虽然 Oracle 表空间名可以长达 30 个字符，但是建议在 UNIX 环境中保持表空间名为 8 个字符或更少。因为可移植的 UNIX 文件名限制为 14 个字符，并且 OFA 数据文件名的后缀为<*n*>.dbf，其中 *n* 是两个数字，即文件系统中总共需要 6 个字符用于后缀。这就为表空间名自身留下了 8 个字符。

只有与数据库<*database name*>关联的控制文件、重做日志文件和数据文件应该存储在目录/<*mount point*>/oradata/<*database name*>中。对于没有使用 ASM 管理的数据库 ord，数据文件名如下：

```
SQL> select file#, name from v$datafile;
     FILE# NAME
---------- ----------------------------------
         1 /u05/oradata/ord/system01.dbf
         2 /u05/oradata/ord/undotbs01.dbf
         3 /u05/oradata/ord/sysaux01.dbf
         4 /u05/oradata/ord/users01.dbf
         5 /u09/oradata/ord/example01.dbf
         6 /u09/oradata/ord/oe_trans01.dbf
         7 /u05/oradata/ord/users02.dbf
         8 /u06/oradata/ord/logmnr_rep01.dbf
         9 /u09/oradata/ord/big_users.dbf
        10 /u08/oradata/ord/idx01.dbf
        11 /u08/oradata/ord/idx02.dbf
        12 /u08/oradata/ord/idx03.dbf
        13 /u08/oradata/ord/idx04.dbf
        14 /u08/oradata/ord/idx05.dbf
        15 /u08/oradata/ord/idx06.dbf
        16 /u08/oradata/ord/idx07.dbf
        17 /u08/oradata/ord/idx08.dbf
17 rows selected.
```

除了编号为 8 和 9 的文件之外，ord 数据库中的所有数据文件都遵从 OFA，并且被展开到 4 个不同的安装点。编号为 8 的文件中的表空间名太长，而编号为 9 的文件没有两位数的数字计数器来表示相同表空间的新数据文件。

2. ASM 环境

在 ASM 环境中，可执行文件存储在前面表示的目录结构中。然而，如果浏览图 3-5 中的目录/u02/oradata，可以看到其中没有任何文件。实例 dw 的所有控制文件、重做日志文件和数据文件都由该服务器上的 ASM 实例+ASM 管理。

大多数管理功能并不需要实际的数据文件名，因为 ASM 文件都是 Oracle 管理文件(Oracle Managed Files，OMF)。这减轻了数据库所需的全部管理工作。在 ASM 存储结构中，类似于 OFA 的语法用于进一步细分文件类型：

```
SQL> select file#, name from v$datafile;
     FILE# NAME
---------- ----------------------------------------
         1 +DATA/dw/datafile/system.256.622426913
         2 +DATA/dw/datafile/sysaux.257.622426915
         3 +DATA/dw/datafile/undotbs1.258.622426919
         4 +DATA/dw/datafile/users.259.622426921
         5 +DATA/dw/datafile/example.265.622427181
5 rows selected.
SQL> select name from v$controlfile;
NAME
----------------------------------------
+DATA/dw/controlfile/current.260.622427059
+RECOV/dw/controlfile/current.256.622427123
2 rows selected.
SQL> select member from v$logfile;
MEMBER
----------------------------------------
+DATA/dw/onlinelog/group_3.263.622427143
+RECOV/dw/onlinelog/group_3.259.622427145
+DATA/dw/onlinelog/group_2.262.622427135
+RECOV/dw/onlinelog/group_2.258.622427137
+DATA/dw/onlinelog/group_1.261.622427127
+RECOV/dw/onlinelog/group_1.257.622427131
6 rows selected.
```

在磁盘组+DATA 和+RECOV 中，可以看到每个数据库文件类型，例如数据文件、控制文件和联机日志文件，都有自己的目录。完全限定的 ASM 文件名具有如下格式：

+<group>/<dbname>/<file type>/<tag>.<file>.<incarnation>

其中<group>是磁盘组名，<dbname>是文件所属的数据库，<file type>是 Oracle 文件类型，<tag>是特定于文件类型的信息，<file>.<incarnation>对用来确保文件在磁盘组中的唯一性。

第 6 章将介绍自动存储管理。

3.2 Oracle 安装表空间

表 3-2 列出了使用标准 Oracle 安装创建的表空间，其中使用了 Oracle 通用安装程序(OUI)。EXAMPLE 表空间是可选的，如果在安装对话期间指定想要创建示例模式，则安装该表空间。

表 3-2 标准 Oracle 安装表空间

表 空 间	类 型	段空间管理	初始分配的近似大小(MB)
SYSTEM	永久	手动	680
SYSAUX	永久	自动	585
TEMP	临时	手动	20
UNDOTBS1	永久	手动	115
USERS	永久	自动	16
EXAMPLE	永久	自动	100

3.2.1 SYSTEM

本章前面提及，没有任何用户段应该存储在 SYSTEM 表空间中。通过自动将一个永久表空间赋予还没有被显式赋予永久表空间的所有用户，create database 命令中的新子句 default tablespace 可帮助防止这种情况的发生。使用 Oracle 通用安装程序(OUI)执行的 Oracle 安装将自动分配 USERS 表空间为默认的永久表空间。

如果过多地使用过程化的对象，例如函数、过程和触发器等，SYSTEM 表空间将快速增长，因为这些对象必须驻留在数据字典中。对于抽象数据类型和 Oracle 的其他面向对象特性，情况也是如此。

3.2.2 SYSAUX

和 SYSTEM 表空间一样，用户段永远不应该存储在 SYSAUX 表空间中。如果 SYSAUX 表空间的特定占用者占据了过多的可用空间，或者严重影响了其他使用 SYSAUX 表空间的应用程序的性能，则应该考虑将该占用者移动到另一个表空间。

3.2.3 TEMP

不推荐使用一个非常大的临时表空间，而应该考虑使用一些较小的临时表空间，并且创建一个临时表空间组来保存它们。如同您在本章前面所看到的，这可以缩短某些应用程序的响应时间。这些受影响的应用程序创建了许多具有相同用户名的会话。

3.2.4 UNDOTBS1

即使数据库可能有多个撤销表空间，在任意给定时间内也只能有一个活动的撤销表空间。如果撤销表

空间需要使用更多的空间，且 AUTOEXTEND 不可用，则可以添加另一个数据文件。一个撤销表空间必须可用于实时应用集群(Real Application Clusters，RAC)环境中的每个节点，因为每个实例都管理自己的撤销。

3.2.5 USERS

USERS 表空间计划用于由每个数据库用户创建的各种段，它不适合于任何产品应用程序。应该为每个应用程序和段类型创建单独的表空间。稍后将介绍一些额外的标准，您可以使用这些标准来决定何时将段分离到它们自己的表空间中。

3.2.6 EXAMPLE

在产品环境中，EXAMPLE 表空间应该被删除。它占用 100MB 磁盘空间，并且具有所有 Oracle 段类型和数据结构类型的示例。如果需要练习，应该创建单独的数据库，使其包含这些示例模式。对于已有的练习数据库，可以使用$ORACLE_HOME/demo/schema 中的脚本将这些示例模式安装到所选的表空间中。

3.3 段分离

一般可根据类型、大小和访问频率将段划分到不同的表空间中。此外，每个表空间将从自己的磁盘组或磁盘设备上获益。然而在实际情况中，大多数计算站并没有能力将每个表空间存储到自己的设备上。下面的要点标识了一些条件，您可以使用这些条件来确定如何将段分离到表空间中。这些条件之间不存在优先级，因为优先级取决于特定的环境。使用自动存储管理(ASM)可以消除这里所列出的许多争用问题，从而不需要 DBA 进行额外的工作。第 4 章将详细讨论 ASM。

- 大段和小段应该在单独的表中。
- 表段和它们所对应的索引段应该在单独的表中。
- 单独的表空间应该用于每个应用程序。
- 较少使用的段和较多使用的段应该在不同的表空间中。
- 静态段应该和高 DML 段分离。
- 只读表应该在其自己的表空间中。
- 数据仓库的分段表应该在其自己的表空间中。
- 根据是否逐行访问段以及是否通过完整表扫描访问段，使用适当的块大小来创建表空间。
- 物化视图应该在与基表不同的单独表空间中。
- 对于分区的表和索引，每个分区应该在其自己的表空间中。

使用 EM 数据库控制台，可以通过标识热点(在文件级或对象级中)来标识任意表空间上的总体争用情况。第 8 章将讨论性能调整，包括解决 I/O 争用问题。

第 3 章讨论了数据库的逻辑组成部分、表空间，以及如何创建正确数量和类型的表空间，并根据使用模式和功能将表和索引段放在适当的表空间中。本章将重点关注数据库和数据文件的物理方面，以及如何存储它们才能最大化 I/O 吞吐量和数据库的整体性能。

本章内容基于以下假设：正在使用本地化管理的表空间，并且具有自动段空间管理功能。除了使用存储在表空间自身中的位图而不是存储在表或索引头块中的空闲列表来减少 SYSTEM 表空间上的加载之外，自动段空间管理(自动分配的或统一的)还可帮助更有效地使用表空间中的空间。从 Oracle 10g 开始，SYSTEM 表空间被创建为本地管理的。因此，所有的读写表空间也必须是本地管理的。

本章的第一部分将回顾使用传统磁盘空间管理时一些常见的问题和解决方案，传统的磁盘空间管理使用数据库服务器上的文件系统。本章的第二部分将概述自动存储管理(Automatic Storage Management，ASM)，这个内置的逻辑卷管理程序可以简化管理、增强性能和改进可用性。

4.1 传统磁盘空间存储

在使用第三方逻辑卷或者 Oracle 自动存储管理(本章后面将讨论)的时候，必须能够管理数据库中的物理数据文件，从而确保高级别的性能、可用性和可恢复性。一般来说，这意味着将数据文件分散到不同的物理磁盘。通过在不同磁盘上保存重做日志文件和控制文件的镜像副本，除了可以确保可用性外，当用户访问驻留在多个物理磁盘上(而不是一个物理磁盘上)的表空间中的表时，还可以有效地改进 I/O 性能。标识特定磁盘卷上的 I/O 瓶颈或存储缺陷只是完成了一半工作，一旦标识了瓶颈，就需要使用各种工具和知识将数据文件移动到不同的磁盘。如果数据文件具有过多的空间或空间不够，则调整已有数据文件的大小是一项常见的任务。

本章将讨论调整表空间大小的各种不同方法，无论这些表空间是小文件表空间还是大文件表空间。此外，本章还将介绍将数据文件、联机重做日志文件和控制文件移动到不同磁盘的最常见方法。

4.1.1 调整表空间和数据文件的大小

在理想的数据库中，应按照最优的大小创建所有的表空间和其中的对象。主动调整表空间的大小或建立自动扩展的表空间可以潜在地避免对性能的影响，这些性能影响发生在表空间扩展或由于表空间中的数据文件无法扩展而造成应用程序失败的情况下。第 6 章将介绍监控空间利用率的更多细节。

用于调整表空间大小的过程和方法存在细微的区别，这取决于表空间是小文件表空间还是大文件表空间。小文件表空间是 Oracle 10g 之前唯一可用的表空间类型，可以由多个数据文件组成。与之相反，大文件表空间可以只由一个数据文件组成，但该数据文件可以远远大于小文件表空间中的数据文件：具有 64KB 块的大文件表空间可以有最大为 128TB 的数据文件。此外，大文件表空间必须是本地管理的。

1. 使用 ALTER DATABASE 调整小文件表空间的大小

在下面的示例中，尝试调整 USERS 表空间的大小，该表空间包含一个数据文件，并且开始时的大小为 5MB。首先，将其调整为 15MB，然后意识到该表空间过大，将其缩减到 10MB。接下来，尝试更多地缩减该表空间的大小。最后，尝试增加该表空间的大小。

```
SQL> alter database
  2     datafile '/u01/app/oracle/oradata/rmanrep/users01.dbf' resize 15m;
   Database altered.
SQL> alter database
  2     datafile '/u01/app/oracle/oradata/rmanrep/users01.dbf' resize 10m;
   Database altered.
SQL> alter database
  2     datafile '/u01/app/oracle/oradata/rmanrep/users01.dbf' resize 1m;
alter database
*
ERROR at line 1:
ORA-03297: file contains used data beyond requested RESIZE value
SQL> alter database
  2     datafile '/u01/app/oracle/oradata/rmanrep/users01.dbf' resize 100t;
alter database
*
ERROR at line 1:
ORA-00740: datafile size of (13421772800) blocks exceeds maximum file size
SQL> alter database
  2     datafile '/u01/app/oracle/oradata/rmanrep/users01.dbf' resize 50g;
alter database
*
ERROR at line 1:
ORA-01144: File size (6553600 blocks) exceeds maximum of 4194303 blocks
```

如果可用的空闲空间不支持重新调整大小的请求，或者数据超出请求减少的大小，或者超出 Oracle 文件大小的限制，则系统都会返回错误。

为了避免被动地手动调整表空间的大小，可以在修改或创建数据文件时使用 autoextend、next 和 maxsize 子句将其改为主动式调整。表 4-1 列出了 alter datafile 和 alter tablespace 命令中用于修改或创建数据文件的空间相关子句。

表 4-1　数据文件扩展子句

子　　句	说　　明
autoextend	将该子句设置为 ON 时，允许扩展数据文件。将其设置为 OFF 时，则不允许扩展，并且将其他子句设置为 0
next<*size*>	在需要扩展时，分配给数据文件的下一个磁盘空间量的大小(以字节为单位)；<*size*>值可以限定为 K、M、G、T，分别用于指定以千字节、兆字节、千兆字节、百万兆字节为单位的大小
maxsize<*size*>	将该子句设置为 unlimited 时，Oracle 中数据文件的大小是无限的，对于大文件表空间，数据文件最大为 128TB；对于具有 32K 块的小文件表空间，数据文件最大为 128GB（另外受到包含数据文件的文件系统的限制）。否则，将 maxsize 设置为数据文件中最大的字节量，使用与 next 子句中相同的限定符：K、M、G、T

在下面的示例中，针对 datafile/u01/app/oracle/oradata/rmanrep/users01.dbf，设置 autoextend 为 ON，指定数据文件的每次扩展为 20MB，并且指定数据文件的大小总计不能超出 1GB：

```
SQL> alter database
  2      datafile '/u01/app/oracle/oradata/rmanrep/users01.dbf'
  3      autoextend on
  4      next 20m
  5      maxsize 1g;
Database altered.
```

如果包含数据文件的磁盘卷没有可用于数据文件扩展的磁盘空间，就必须将数据文件移动到另一个磁盘卷，或者创建位于另一个磁盘卷上的表空间的第二个数据文件。在该示例中，将第二个数据文件添加到不同磁盘卷上的 USERS 表空间，该表空间的初始大小为 50MB，允许自动扩展数据文件，且每次扩展 10MB，最大数据文件大小为 200MB：

```
SQL> alter tablespace users
  2 add datafile '/u03/oradata/users02.dbf'
  3 size 50m
  4 autoextend on
  5 next 10m
  6 maxsize 200m;
Tablespace altered.
```

注意，修改表空间中的已有数据文件时，使用 alter database 命令，而在添加数据文件到表空间时，使用 alter tablespace 命令。如同您将在后面所看到的那样，使用大文件表空间可简化这些类型的操作。

2. 使用 EM Database Control 调整小文件表空间的大小

使用 EM Database Control，可以使用前面小节中所描述的两种方法中的任何一种：增加大小并针对表空间的单个数据文件打开自动扩展，或者添加第二个数据文件。

调整小文件表空间中的数据文件　为了在 EM Database Control 中调整数据文件的大小，从数据库

实例主页中选择 Server 选项卡，然后单击 Storage 标题下的 Tablespaces。在图 4-1 中，已经选择了 XPORT 表空间，它所分配空间的利用率已超过 85%，因此决定使用第二个数据文件来扩展它的大小。此表空间最初是使用如下命令创建的：

```
create tablespace xport datafile '/u02/oradata/xport.dbf' size 150m;
```

接下来将不设置自动扩展的表空间数据文件，而是将其当前大小从 150MB 改为 200MB。

图 4-1　使用 EM Database Control 编辑表空间

单击 Edit 按钮，可以看到 XPORT 表空间的特征，如图 4-2 所示。这是本地管理的、永久性的、非大文件表空间(即小文件表空间)。该页面的底部是 XPORT 表空间的一个数据文件，/u02/oradata/xport.dbf。

图 4-2　表空间特征

选择 XPORT 表空间中唯一的数据文件，单击 Edit 按钮或数据文件名自身，可以看到 Edit Tablespace: Edit Datafile 页面，如图 4-3 所示，可以在其中改变数据文件的大小。在该页面上，将文件大小从 150MB 改为 200MB，然后单击 Continue 按钮。

图 4-3　编辑表空间的数据文件

在图 4-4 中，返回到 Edit Tablespace 页面。此时，可以通过单击 Apply 按钮改变数据文件，通过单击 Revert 按钮取消改变，或者通过单击 Show SQL 按钮显示执行的 SQL。

图 4-4　确认数据文件改变

在提交改动之前，最好先通过单击 Show SQL 按钮显示将要执行的 SQL 命令：这是复习 SQL 命令语法的好方法。下面是单击 Apply 按钮时将要执行的 SQL 命令：

```
ALTER DATABASE DATAFILE '/u02/oradata/xport.dbf' RESIZE 200M
```

单击 Apply 按钮时，对数据文件的大小进行改动。Edit Tablespace: XPORT 页面反映了成功的操作和数据文件的新大小，如图 4-5 所示。

添加数据文件到小文件表空间　添加数据文件到小文件表空间和使用 EM Database Control 调整数据文件大小一样容易。在前面的示例中，将 XPORT 表空间的数据文件扩展为 200MB。因为包含 XPORT 表空间数据文件的文件系统(/u02)容量已满，所以必须关闭已有数据文件上的 AUTOEXTEND，然后在不同的文件系统上创建新的数据文件。在图 4-6 中，通过取消选中 Storage 部分中的复选框，关闭了已有数据文件的 AUTOEXTEND。下面是单击 Continue 按钮然后单击 Apply 按钮时针对该操作执行的 SQL 命令：

```
ALTER DATABASE
    DATAFILE '/u02/oradata/xport.dbf'
    AUTOEXTEND OFF;
```

图 4-5　数据文件调整大小的结果

图 4-6　编辑表空间的数据文件特征

在图 4-1 中的 Tablespaces 页面上，选择 XPORT 表空间旁边的单选按钮，并单击 Edit 按钮，会看到如图 4-7 所示的页面。

图 4-7　编辑 XPORT 表空间

在图 4-7 中，单击 Add 按钮，会看到如图 4-8 所示的页面。

图 4-8　向 XPORT 表空间添加数据文件

在图 4-8 所示的页面上，指定新数据文件的文件名和目录位置。因为已经知道/u04 文件系统具有至少 100MB 的空闲空间，所以指定/u04/oradata 作为目录，并且指定 xport2.dbf 作为文件名，虽然该文件名自身并不需要包含表空间名称。此外，设置文件大小为 100MB，并且不选中 AUTOEXTEND 的复选框。

单击 Continue 按钮，然后单击 Apply 按钮，可以看到 Update Message 和 XPORT 表空间数据文件的新大小，如图 4-9 所示。

图 4-9　添加数据文件后查看 XPORT 表空间

3. 从表空间中删除数据文件

在以前的 Oracle 版本中，删除表空间中的数据文件存在一定的问题。那就是无法提交单个的命令来删除数据文件，除非删除整个表空间。此时，只有 3 种选择：

- 容忍该数据文件。
- 缩减该数据文件并关闭 AUTOEXTEND。
- 创建新的表空间，将所有的对象移动到新的表空间，并且删除原来的表空间。

从维护和元数据的观点来看，虽然创建新的表空间是最理想的选择，但执行有关步骤很容易产生错误，并且需要一定的停机时间，从而影响其可用性。

使用 EM Database Control，可以删除数据文件并最小化停机时间，并且由 EM Database Control 生成所需的脚本。前面的示例通过添加数据文件来扩展 XPORT 表空间，而这里我们通过重新组织表空间来

删除数据文件。在 Tablespace 页面上，选择需要重新组织的表空间(在此例中是 XPORT)，在 Actions 下拉列表框中选择 Reorganize，然后单击 Go 按钮，如图 4-10 所示。

在图 4-11 所示的 Reorganize Objects 页面上，确认需要重新组织的 XPORT 表空间，然后单击 Next 按钮。

图 4-10 Tablespace: Reorganize

图 4-11 Reorganize Objects: Objects

如图 4-12 所示，在下一个页面中，可以为重新组织设置一些参数。例如，对于当前的重新组织，重

新组织的速度和表空间的可用性哪一个更为重要。此外，可以利用表空间重命名特性，而不是使用临时表空间作为工作区域，从而节省磁盘空间以及重新组织的时间。该页面上的其他参数包括指定并行执行、没有记录的索引重建，以及重新组织完成后所需的统计收集级别。

图 4-12 Reorganize Objects: Options

图 4-13 显示了脚本创建的状态。生成脚本所花费的时间基本上与表空间中对象的数量成比例。

图 4-13 Processing: Generating Reorganization Script

小结界面中显示了在脚本生成期间遇到的任何警告或错误，如图 4-14 中的 Impact Report 所示。

图 4-14 Reorganize Objects: Impact Report

单击 Next 按钮后，可以看到 Schedule 页面，如图 4-15 所示。在当前情况中，继续前进并指定服务器的主机凭证，但在该向导的末尾不会提交该工作，因为还需要对脚本进行编辑。

图 4-15 Reorganize Objects: Schedule

单击 Next 按钮，可显示如图 4-16 所示的 Review 页面。文本框中显示了所生成的脚本的摘录。不要立刻提交工作，而应单击 Save Full Script，以在运行脚本前对其进行一些小的改动。

图 4-16 Reorganize Objects: Review

在图 4-17 中，指定希望保存脚本的位置。

编辑完整的脚本时，应将 execute immediate 命令定位在创建表空间的位置：

```
EXECUTE IMMEDIATE 'CREATE SMALLFILE TABLESPACE "XPORT_REORG0"
  DATAFILE '/u02/oradata/xport_reorg0.dbf'    SIZE 200M REUSE,
    ''/u04/oradata/xport2_reorg0.dbf'' SIZE 100M REUSE
    LOGGING EXTENT MANAGEMENT LOCAL
    SEGMENT SPACE MANAGEMENT AUTO';
```

图 4-17 Review: Save Full Script

因为希望删除数据文件，所以需要删除脚本中高亮显示的数据文件子句，然后改变第二个数据文件的位置，或者重新创建第一个数据文件，使其具有较大的空间。在该示例中，通过修改 create tablespace 命令，不仅创建较大的新表空间，而且将其放在不同的磁盘卷上：

```
EXECUTE IMMEDIATE 'CREATE SMALLFILE TABLESPACE "XPORT_REORG0"
  DATAFILE ''/u04/oradata/xport.dbf''
    SIZE 300M REUSE
    LOGGING EXTENT MANAGEMENT LOCAL
    SEGMENT SPACE MANAGEMENT AUTO';
```

一旦已经编辑完脚本，就应使用具有 DBA 权限的账户在 SQL*Plus 中运行脚本。脚本的输出看起来类似于下面这样：

```
SQL> @reorg1.sql
--   Target database:   dw.world
--   Script generated at:       08-JUL-2007   23:38
Starting reorganization
Executing as user: RJB
CREATE  SMALLFILE  TABLESPACE "XPORT_REORG0" DATAFILE
'/u04/oradata/xport_reorg0.dbf' SIZE 300M REUSE  LOGGING  EXTENT MANAGEMENT
LOCAL SEGMENT SPACE MANAGEMENT  AUTO
ALTER TABLE "SYS"."OBJ_FILL" MOVE TABLESPACE "XPORT_REORG0"
DROP TABLESPACE "XPORT" INCLUDING CONTENTS AND DATAFILES CASCADE CONSTRAINTS
ALTER TABLESPACE "XPORT_REORG0" RENAME TO "XPORT"
Completed Reorganization.  Starting cleanup phase.
Starting cleanup of recovery tables
Completed cleanup of recovery tables
Starting cleanup of generated procedures
Completed cleanup of generated procedures
Script execution complete
SQL>
```

如果使用大文件表空间，则可以在很多情况下避免使用重新组织脚本，因为大文件表空间只由一个数据文件组成。下一节将讨论大文件表空间的重新组织。

4. 使用 ALTER TABLESPACE 调整大文件表空间的大小

大文件表空间只由一个数据文件组成。第 6 章将介绍关于大文件表空间的更多内容，这里仅介绍一些重新调整大文件表空间大小的细节。用于改变表空间数据文件特征(例如最大尺寸、是否可以完全扩展，以及扩展盘区的大小)的大多数参数现在都可以在表空间级别上进行修改。首先创建一个大文件表空间，如下所示：

```
create bigfile tablespace dmarts
   datafile '/u05/oradata/dmarts.dbf' size 750m
   autoextend on next 100m maxsize unlimited
   extent management local
   segment space management auto;
```

只在具有小文件表空间的数据文件级别中有效的操作可以在表空间级别上用于大文件表　空间：

```
SQL> alter tablespace dmarts resize 1g;
Tablespace altered.
```

虽然将 alter database 命令和 DMARTS 表空间的数据文件规范一起使用也可以起作用，但使用 alter tablespace 语法的优点是显而易见的：不需要知道数据文件存储在何处。在具有小文件表空间的表空间级别上尝试改变数据文件参数是不允许的：

```
SQL> alter tablespace users resize 500m;
alter tablespace users resize 500m
*
ERROR at line 1:
ORA-32773: operation not supported for smallfile tablespace USERS
```

如果大文件表空间因为其单个数据文件无法在磁盘上扩展而用完空间，则需要重新分配数据文件到另一个卷，4.1.2 小节中将讨论这一点。使用本章后面介绍的自动存储管理(ASM)，完全可能做到不需要手动移动数据文件：不需要移动数据文件，只要添加另一个磁盘卷到 ASM 存储组即可。

4.1.2 移动数据文件

为了更好地管理数据文件的大小或改进数据库的整个 I/O 性能，可能需要将表空间中的一个或多个数据文件移动到不同的位置。重新定位数据文件有3种方法：使用alter database命令、使用alter tablespace命令，以及通过 EM Database Control(尽管 EM Database Control 没有提供重新定位数据文件所需的所有命令)。

除了 SYSTEM、SYSAUX、联机撤销表空间以及临时表空间以外，alter tablespace 方法可作用于其他所有表空间中的数据文件。alter database 可作用于所有表空间中的数据文件，因为在进行移动操作时实例将被关闭。

1. 使用 ALTER DATABASE 移动数据文件

使用 alter database 移动一个或多个数据文件的步骤如下：

(1) 作为 SYSDBA 连接到数据库，并且关闭实例。

(2) 使用操作系统命令移动数据文件。

(3) 以 MOUNT 模式打开数据库。

(4) 使用 alter database 改变对数据库中数据文件的引用。

(5) 以 OPEN 模式打开数据库。

(6) 对包括控制文件的数据库执行增量备份或完整备份。

在下面的示例中，将介绍如何将 XPORT 表空间的数据文件从文件系统/u04 移动到文件系统/u06。首先，用如下命令在 SYSDBA 权限下连接到数据库：

```
sqlplus / as sysdba
```

接下来，使用针对动态性能视图 V$DATAFILE 和 V$TABLESPACE 的查询，确认 XPORT 表空间中数据文件的名称。

```
SQL> select d.name from
  2     v$datafile d join v$tablespace t using(ts#)
  3     where t.name = 'XPORT';
NAME
-------------------------------------------------------------
/u04/oradata/xport.dbf
1 row selected.
SQL>
```

为了完成步骤 1，需要关闭数据库：

```
SQL> shutdown immediate;
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL>
```

对于步骤 2，在 SQL*Plus 中，使用“!”转义字符，执行操作系统命令以移动数据文件：

```
SQL> ! mv /u04/oradata/xport.dbf /u06/oradata
```

在步骤 3 中，以 MOUNT 模式启动数据库，从而在不需要打开数据文件的情况下就可以使用控制文件：

```
SQL> startup mount
ORACLE instance started.
Total System Global Area  422670336 bytes
Fixed Size                  1299112 bytes
Variable Size             230690136 bytes
Database Buffers          184549376 bytes
Redo Buffers                6131712 bytes
Database mounted.
```

对于步骤 4，改变控制文件中的路径名引用，以将其指向数据文件的新位置：

```
SQL> alter database rename file
  2    '/u04/oradata/xport.dbf' to
  3    '/u06/oradata/xport.dbf';
Database altered.
```

在步骤 5 中，打开数据库，使用户可使用该数据库：

```
SQL> alter database open;
Database altered.
```

最后，在步骤 6 中，建立更新过的控制文件的备份副本：

```
SQL> alter database backup controlfile to trace;
```

```
Database altered.
SQL>
```

也可以选择使用 RMAN 执行增量备份，其中包括了控制文件的备份。

2. 使用 ALTER TABLESPACE 移动数据文件

如果希望移动的数据文件是某个表空间的一部分，而该表空间不是 SYSTEM、SYSAUX、活动的撤销表空间或临时表空间，则使用 alter tablespace 方法移动表空间会更好一些，其主要原因在于：除了其数据文件将被移动的表空间外，所有用户在整个操作期间都可以使用数据库的剩余部分。

使用 alter tablespace 移动一个或多个数据文件的步骤如下：

(1) 使用具有 ALTER TABLESPACE 权限的账户，对表空间进行脱机处理。

(2) 使用操作系统命令移动数据文件。

(3) 使用 alter tablespace 改变对数据库中数据文件的引用。

(4) 将表空间返回到联机状态。

在 alter database 示例中，假设将 XPORT 表空间的数据文件移动到了错误的文件系统。在本示例中，将数据文件从/u06/oradata 移动到了/u05/oradata：

```
SQL> alter tablespace xport offline;
Tablespace altered.
SQL> ! mv /u06/oradata/xport.dbf /u05/oradata/xport.dbf
SQL> alter tablespace xport rename datafile
  2     '/u06/oradata/xport.dbf' to '/u05/oradata/xport.dbf';
Tablespace altered.
SQL> alter tablespace xport online;
Tablespace altered.
```

需要注意的是，该方法比 alter database 方法更为直观并具有更少的中断。XPORT 表空间唯一的停机时间是将数据文件从一个磁盘卷移动到另一个磁盘卷所花费的时间量。

3. 使用 EM Database Control 移动数据文件

在 Oracle Database 11g 的版本 1 中，EM Database Control 没有专门用于移动数据文件的功能，并且无法执行本章前面所演示的表空间的重新组织。因此，它无法将数据文件移动到另一个卷。

4.1.3 移动联机重做日志文件

虽然通过删除整个重做日志组并在不同的位置重新添加这些组，可以间接移动联机重做日志文件，但是，如果只有两个重做日志文件组，则这种解决方案将不起作用，因为数据库不会在只有一个重做日志文件组的情况下打开。如果数据库必须保持打开状态，可以选择临时添加第三个组并删除第一个或第二个组。或者，关闭数据库，并使用下面的方法移动重做日志文件。

在下面的示例中，有三个重做日志文件组，每个组有两个成员。每个组都有一个成员位于与 Oracle 软件相同的卷上，因此应将其移动到不同卷以消除写入日志文件与访问 Oracle 组件之间的争用问题。这里使用的方法非常类似于使用 alter database 移动数据文件时所采取的方法。

```
SQL> select group#, member from v$logfile
  2     order by group#, member;
    GROUP# MEMBER
---------- ---------------------------------------------
         1 /u01/app/oracle/oradata/redo01.log
         1 /u05/oradata/redo01.log
         2 /u01/app/oracle/oradata/redo02.log
         2 /u05/oradata/redo02.log
         3 /u01/app/oracle/oradata/redo03.log
         3 /u05/oradata/redo03.log
6 rows selected.
```

```
SQL> shutdown immediate;
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL> ! mv /u01/app/oracle/oradata/redo0[1-3].log /u04/oradata
SQL> startup mount
ORACLE instance started.
Total System Global Area    422670336 bytes
Fixed Size                    1299112 bytes
Variable Size                230690136 bytes
Database Buffers             184549376 bytes
Redo Buffers                   6131712 bytes
Database mounted.
SQL> alter database rename file '/u01/app/oracle/oradata/redo01.log'
  2     to '/u04/oradata/redo01.log';
Database altered.
SQL> alter database rename file '/u01/app/oracle/oradata/redo02.log'
  2     to '/u04/oradata/redo02.log';
Database altered.
SQL> alter database rename file '/u01/app/oracle/oradata/redo03.log'
  2     to '/u04/oradata/redo03.log';
Database altered.
SQL> alter database open;
Database altered.
SQL> select group#, member from v$logfile
  2     order by group#, member;
    GROUP# MEMBER
---------- ----------------------------------------
         1 /u04/oradata/redo01.log
         1 /u05/oradata/redo01.log
         2 /u04/oradata/redo02.log
         2 /u05/oradata/redo02.log
         3 /u04/oradata/redo03.log
         3 /u05/oradata/redo03.log
```

6 rows selected.

SQL>

重做日志文件不再和 Oracle 软件竞争 I/O。另外，上面的示例在两个不同的安装点/u04 和/u05 之间多元复用了重做日志文件。

4.1.4 移动控制文件

在使用初始参数文件时，移动控制文件的步骤类似于前面移动数据文件和重做日志文件的过程。过程如下：关闭实例，使用操作系统命令移动文件，然后重新启动实例。

然而，在使用服务器参数文件(SPFILE)时，该过程稍有不同。当实例正在运行，或者实例已经关闭，但以 NOMOUNT 模式打开时，应使用 alter system…scope=spfile 改变初始文件参数 CONTROL_FILES。由于 CONTROL_FILES 参数不是动态参数，因此无论何种情况都必须先关闭实例，然后重新启动它。

在本示例中，数据库中有控制文件的 3 个副本，但没有在不同的磁盘上实现多元复用。编辑 SPFILE 以使用新的位置，关闭实例，移动控制文件到不同的磁盘，然后再重新启动实例。

```
SQL> select name, value from v$spparameter
  2     where name = 'control_files';
NAME                 VALUE
---------------  --------------------------------------------------
control_files    /u01/app/oracle/oradata/control01.ctl
control_files    /u01/app/oracle/oradata/control02.ctl
control_files    /u01/app/oracle/oradata/control03.ctl
SQL> show parameter control_files
NAME                 TYPE         VALUE
---------------  ---------  ------------------------------
control_files     string       /u01/app/oracle/oradata/contro
                                  l01.ctl, /u01/app/orac le/orad
                                  ata/control02.ctl, /u01/app/or
                                  acle/oradata/control03.ctl

SQL> alter system set control_files =
  2    '/u02/oradata/control01.ctl',
  3    '/u03/oradata/control02.ctl',
  4    '/u04/oradata/control03.ctl'
  5  scope = spfile;
System altered.
SQL> shutdown immediate
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL> ! mv /u01/app/oracle/oradata/control01.ctl /u02/oradata
SQL> ! mv /u01/app/oracle/oradata/control02.ctl /u03/oradata
```

```
SQL> ! mv /u01/app/oracle/oradata/control03.ctl /u04/oradata
SQL> startup
ORACLE instance started.

Total System Global Area    422670336 bytes
Fixed Size                    1299112 bytes
Variable Size                230690136 bytes
Database Buffers             184549376 bytes
Redo Buffers                   6131712 bytes
Database mounted.
Database opened.
```

```
SQL> select name, value from v$spparameter
  2  where name = 'control_files';
NAME            VALUE
--------------- ------------------------------------------------
control_files   /u02/oradata/control01.ctl
control_files   /u03/oradata/control02.ctl
control_files   /u04/oradata/control03.ctl
SQL> show parameter control_files
NAME                TYPE        VALUE
--------------   --------   ------------------------------
control_files    string     /u02/oradata/control01.ctl, /u
                                 03/oradata/control02.ctl, /u04
                                 /oradata/control03.ctl
SQL>
```

至此

已经将 3 个控制文件移动到单独的文件系统，不再位于具有 Oracle 软件的卷上，且具有较高的可用性配置(如果包含一个控制文件的卷失败，那么其他两个卷包含更新的控制文件)。

注意：
在对表空间存储和闪回恢复区使用 ASM 磁盘的 Oracle Database 11g 默认安装中，控制文件的一个副本在默认表空间 ASM 磁盘中创建，另一个副本在闪回恢复区中创建。

复制控制文件的一个或多个副本到 ASM 卷很容易：使用 RMAN 实用工具(第 12 章将会详细介绍)，将控制文件备份恢复到 ASM 磁盘位置即可，如下面这个示例所示。

```
RMAN> restore controlfile to
        '+DATA/dw/controlfile/control_bak.ctl';
```

下一步骤与前面所介绍的添加基于文件系统的控制文件的步骤是相同的：改变 CONTROL_FILES 参数，除已有控制文件位置外，添加位置+DATA/dw/controlfile/control_ bak.ctl，然后关闭数据库，并重新启动数据库。

```
SQL> show parameter control_files
NAME                 TYPE          VALUE
---------------   --------    ------------------------------
control_files   string      /u02/oradata/control01.ctl, /u
                                    03/oradata/control02.ctl, /u04
                                    /oradata/control03.ctl, +DATA/
                                    dw/controlfile/control_bak.ctl
SQL>
```

类似地，可以使用 Linux 实用工具 asmcmd，将控制文件从一个磁盘组复制到另一个磁盘组，并改变 CONTROL_FILES 参数，以反映控制文件的新位置。本章后面的内容将概括介绍 asmcmd 命令。

4.2 自动存储管理

第3章介绍了一些用于ASM对象的文件命名约定。本节将深入介绍如何在具有一个或多个磁盘组的ASM环境中创建表空间，以及如何创建表空间中的数据文件。

在创建新的表空间或其他数据库结构(例如控制文件或重做日志文件)时，可以指定磁盘组，而不是一个操作系统文件，作为数据库结构的存储区域。ASM 简化了 Oracle 管理文件(OMF)的使用，并将 OMF 与镜像和条带化特性结合起来，从而提供了健壮的文件系统和逻辑卷管理程序，这种管理程序甚至支持 Oracle 实时应用集群(RAC)中的多个节点。ASM 不再需要购买第三方逻辑卷管理程序。

ASM 不仅能够自动将数据库对象扩展到多个设备以增强性能，而且允许在不关闭数据库的情况下将新的磁盘设备添加到数据库，从而增加可用性。ASM 自动重新平衡文件的分布，使其具有最低限度的干涉。

下面将讨论 ASM 体系结构。此外，我们还将展示如何创建特殊类型的 Oracle 实例以支持 ASM，并介绍如何启动和关闭 ASM 实例。接着，介绍与 ASM 相关的新初始参数以及具有新值以支持 ASM 实例的已有初始参数。另外还要介绍 asmcmd 命令行实用工具，它是 Oracle 10g 版本 2 的新增特性，提供了浏览和维护 ASM 磁盘组中对象的另一种方法。最后，使用 Linux 服务器上的一些裸磁盘设备来演示如何创建并维护磁盘组。

4.2.1 ASM 体系结构

ASM 将数据文件和其他数据库结构划分为多个盘区，并且将盘区划分到磁盘组的所有磁盘中，从而增强性能和可靠性。ASM 并没有镜像整个磁盘卷，而是镜像数据库对象，从而提供根据类型有区别地镜像或条带化数据库对象的灵活性。如果底层的磁盘硬件已经启用了 RAID 作为存储区域网络(Storage Area Network，SAN)的一部分，或者网络附加存储(Network Attached Storage，NAS)设备的一部分，则对象不可以条带化。

自动重新平衡是 ASM 的另一个关键特性。需要增加磁盘空间时，可以将额外的磁盘设备添加到磁盘组，ASM 会将一定比例的文件从一个或多个已有的磁盘移动到新的磁盘，从而维持所有磁盘之间整体的 I/O 平衡。当包含在磁盘文件中的数据库对象保持联机并且用户可以使用这些对象时，这种自动重新平衡在后台发生。如果在重新平衡操作期间 I/O 子系统会受到非常大的影响，则可以使用初始参数降低重新平衡发生的速度。

ASM 需要特殊类型的 Oracle 实例来提供传统 Oracle 实例和文件系统之间的接口。ASM 软件的组件和 Oracle 数据库软件一起传输。在创建数据库并为 SYSTEM、SYSAUX 和其他表空间选择存储类型时，总是可以将这些组件作为一种选择。

然而，ASM 并不能支持组合使用 ASM 磁盘组与手动 Oracle 数据文件管理技术(例如第 3 章和本章前面介绍的一些技术)。然而，ASM 易于使用并且具有很高的性能，因此使用 ASM 磁盘组可以很好地满足所有存

储需求。

Oracle Database 10g 中新增加了两个 Oracle 后台进程以支持 ASM 实例：RBAL 和 ORB*n*。RBAL 协调磁盘组的磁盘活动，而 ORB*n*(其中 *n* 可以是 0~9 之间的数字)执行磁盘组中磁盘之间的实际盘区移动。

对于使用 ASM 磁盘的数据库，Oracle Database 10g 也有两个新的后台进程：OSMB 和 RBAL。OSMB 执行数据库和 ASM 实例之间的通信，而 RBAL 执行代表数据库的磁盘组中磁盘的打开和关闭。

4.2.2 创建 ASM 实例

ASM 需要专用的 Oracle 实例来管理磁盘组。ASM 实例一般只需要较少的内存：60MB~120MB。当安装 Oracle 软件时，在指定 ASM 为数据库的文件存储选项，而 ASM 实例不存在时，Oracle 软件将自动配置 ASM 实例。这一点可以在图 4-18 所示的 Oracle Universal Installer 界面上看到。

图 4-18　指定 ASM 作为数据库文件存储方法

作为创建 ASM 磁盘组的磁盘设备的示例，假定 Linux 服务器具有大量裸磁盘设备，表 4-2 列出了它们的容量。

表 4-2　ASM 磁盘组的裸设备

设　备　名	容　　量
/dev/raw/raw1	12GB
/dev/raw/raw2	12GB
/dev/raw/raw3	12GB
/dev/raw/raw4	12GB
/dev/raw/raw5	4GB

/dev/raw/raw6	4GB
/dev/raw/raw7	4GB
/dev/raw/raw8	4GB

在 Oracle 通用安装程序(Oracle Universal Installer，OUI)中配置第一个磁盘组，如图 4-19 所示。

图 4-19　使用 OUI 配置初始 ASM 磁盘组

第一个磁盘组的名称是 DATA，并且使用/dev/raw/raw1 和/dev/raw/raw2 创建普通的冗余磁盘组。如果为所需的冗余级别选择的裸磁盘数量不够，OUI 就生成一个错误消息。创建数据库后，启动普通的实例和 ASM 实例。

ASM 实例有其他一些独特的特征。虽然它确实有初始参数文件和密码文件，但没有数据字典，因此所有的 ASM 实例连接都通过 SYS 或 SYSTEM 并且只使用操作系统验证。可以使用 connect / as sysdba 命令只连接到 ASM 实例。connect 命令中的任何用户名/密码都被忽略。磁盘组命令，例如 create diskgroup、alter diskgroup 和 drop diskgroup，只在 ASM 实例中有效。最后，ASM 实例只能处于 NOMOUNT 或 MOUNT 状态，而绝对不会处于 OPEN 状态。

4.2.3　ASM 实例组成部分

使用可用于传统数据库的各种方法无法访问 ASM 实例。本节将讨论使用 SYSDBA 和 SYSOPER 权限连接时用户所具有的权限。通过新增的和扩展的只可用于 ASM 实例的初始参数(在 Oracle Database 10g 中引入并在 Oracle Database 11g 中得到增强)，可以区分 ASM 实例。本节的末尾将介绍启动和停止 ASM 实例的过程，以及 ASM 实例与其服务的数据库实例之间的相关性。

1. 访问 ASM 实例

本章前面曾提及，ASM 实例没有数据字典，因此只有可以使用操作系统验证的用户才可以访问实例，即由 dba 组中的操作系统用户以 SYSDBA 或 SYSOPER 权限连接。

作为 SYSDBA 连接到 ASM 实例的用户可以执行所有的 ASM 操作，例如创建和删除磁盘组，以及向磁盘组中添加磁盘和从磁盘组中删除磁盘。

SYSOPER 用户在使用可用于 ASM 实例中的命令时有更多限制。一般来说，SYSOPER 用户可用的命令只提供了足够的权限来执行已配置的和稳定的 ASM 实例的例程操作。下面的列表包含了 SYSOPER 可用的操作：

- 启动和关闭 ASM 实例。
- 安装和卸载磁盘组。
- 将磁盘组的磁盘状态从 ONLINE 改为 OFFLINE，或者从 OFFLINE 改为 ONLINE。
- 重新平衡磁盘组。
- 执行磁盘组的完整性检查。
- 访问 V$ASM_*动态性能视图。

2. ASM 初始参数

ASM 实例有许多特有的初始参数，并且某些初始参数在 ASM 实例中有新的值。对于 ASM 实例，强烈推荐使用 SPFILE，而不是初始参数文件。例如，在添加或删除磁盘组时，自动维护 ASM_DISKGROUPS 等参数，从而不需要手动改变该值。

下面将介绍相关的 ASM 初始参数。

INSTANCE_TYPE　对于 ASM 实例，INSTANCE_TYPE 参数具有 ASM 的值。对于传统的 Oracle 实例，默认值为 RDBMS。

DB_UNIQUE_NAME DB_UNIQUE_NAME 参数的默认值是+ASM，它是集群中或单个节点上 ASM 实例组的唯一名称。

ASM_POWER_LIMIT 为了确保重新平衡操作不会干扰到正在进行的用户 I/O，可用 ASM_POWER_LIMIT 参数控制重新平衡操作发生的速度。其值的范围是 1~11，11 是最大可能的值；默认值是 1(较低的 I/O 开销)。因为这是动态参数，所以可以在白天将其设为较低的值，而在夜间必须进行磁盘重新平衡操作时，将其设为较高的值。

ASM_DISKSTRING　ASM_DISKSTRING 参数指定一个或多个串，这些串与操作系统相关，用于限制可用于创建磁盘组的磁盘设备。如果该值为 NULL，则 ASM 实例可见的所有磁盘将潜在地成为创建磁盘组的候选项。对于本章中作为测试服务器的示例，ASM_DISKSTRING 参数的值为/dev/raw/*：

```
SQL> select name, type, value from v$parameter
  2      where name = 'asm_diskstring';
NAME                 TYPE VALUE
---------------- ---------- -------------------------
asm_diskstring            2 /dev/raw/*
```

ASM_DISKGROUPS　ASM_DISKGROUPS 参数指定一个包含磁盘组名称的列表，可以在启动时由 ASM 实例自动安装这些磁盘组，或者通过 alter diskgroup all mount 命令安装。即使在实例启动时该列表为空，也可以手动安装任何已有的磁盘组。

LARGE_POOL_SIZE LARGE_POOL_SIZE 参数可用于普通实例和 ASM 实例，然而对于 ASM 实例而言，使用这种池有一些区别。所有的内部 ASM 程序包都从该池中执行，因此，对于单一的实例，应该将该参数至少设置为 12MB；对于 RAC 实例，应该将该参数至少设置为 16MB。

ASM_PREFERRED_READ_FAILURE_GROUPS　ASM_PREFERRED_READ_FAILURE_GROUPS 参数是 Oracle Database 11g 中新增的参数，它是一个故障组列表，包含使用集群化的 ASM 实例时给定数据库实例的首选故障组。不同的实例可以有不同的参数值：每个实例可以指定距离实例的结点最近的故障组(例如，服务器的本地磁盘上的故障组)，从而提高性能。

3. ASM 实例的启动和关闭

ASM 实例的启动非常类似于数据库实例，除了 startup 命令默认为 startup mount。由于没有安装任何控制文件、数据库或数据字典，因此系统将安装 ASM 磁盘组而不是数据库。命令 startup nomount 启动实例，但是不安装任何 ASM 磁盘。此外，可以指定 startup restrict，临时防止数据库实例连接到 ASM 实例以安装磁盘组。

在 ASM 实例上执行 shutdown 命令，相当于在使用 ASM 实例的任何数据库实例上执行相同的 shutdown 命令。在 ASM 实例完成关闭前，它等待所有相关的数据库关闭。这种情况的唯一例外是，如果在 ASM 实例上使用 shutdown abort 命令，则将最终迫使所有相关的数据库执行 shutdown abort。

对于共享磁盘组的多个 ASM 实例，例如在实时应用集群(RAC)环境中，一个 ASM 实例故障不会造成数据库实例失败。此时，另一个 ASM 实例可以执行失败实例的恢复操作。

4.2.4　ASM 动态性能视图

一些新的动态性能视图与 ASM 实例关联。表 4-3 包含了常见的与 ASM 相关的动态性能视图。本章后面将对其中一些视图进行更深入的解释。

表 4-3　与 ASM 相关的动态性能视图

视 图 名 称	是否用于标准数据库	说　明
V$ASM_DISK	是	一行对应于由 ASM 实例发现的某个磁盘，磁盘组使用(或不使用)该磁盘。对于数据库实例，

		一行对应于由实例使用的某个磁盘组
V$ASM_DISKGROUP	是	对于 ASM 实例，一行对应于包含磁盘组一般性特征的某个磁盘组。对于数据库实例，一行对应于使用的某个磁盘组，无论是否安装该磁盘组
V$ASM_FILE	否	一行对应于某个已安装磁盘组中的每个文件
V$ASM_OPERATION	否	一行对应于 ASM 实例中某个正在执行的、长期运行的操作
V$ASM_TEMPLATE	是	一行对应于ASM实例中某个已安装磁盘组中的某个模板。对于数据库实例，一行对应于某个已安装磁盘组的某个模板
V$ASM_CLIENT	是	一行对应于使用由ASM实例管理的磁盘组的某个数据库。对于数据库实例，如果打开任何 ASM 文件，则一行对应于 ASM 实例
V$ASM_ALIAS	否	一行对应于某个已安装磁盘组中的某个别名

4.2.5 ASM 文件名格式

所有 ASM 文件都是 Oracle 管理文件(OMF)，因此大多数管理功能并不需要磁盘组中实际文件名的细节。删除 ASM 磁盘组中的对象时，将自动删除对应的文件。某些命令将提供实际的文件名，例如 alter database backup controlfile to trace；一些数据字典和动态性能视图也可以提供实际的文件名，例如，动态性能视图 V$DATAFILE 显示每个磁盘组中的实际文件名。下面是一个示例：

```
SQL> select file#, name, blocks from v$datafile;
     FILE# NAME                                               BLOCKS
---------- ----------------------------------------    ----------
         1 +DATA/dw/datafile/system.256.627432971          89600
         2 +DATA/dw/datafile/sysaux.257.627432973         77640
         3 +DATA/dw/datafile/undotbs1.258.627432975       12800
         4 +DATA/dw/datafile/users.259.627432977            640
         5 +DATA/dw/datafile/example.265.627433157        12800
         6 /u05/oradata/dmarts.dbf                          32000
         8 /u05/oradata/xport.dbf                           38400
7 rows selected.
```

ASM 文件名可以是 6 种不同格式中的一种。下面分别概述不同的格式以及使用它们的环境：或者作为对已有文件的引用，或者用于单个文件和多个文件的创建。

1. 完全限定的名称

完全限定的 ASM 文件名只在引用已有文件时使用。完全限定的 ASM 文件名具有如下　格式：

+*group*/*dbname*/*file type*/*tag.file.incarnation*

其中，*group* 是磁盘组名，*dbname* 是文件所属的数据库，*file type* 是 Oracle 文件类型，*tag* 是文件类型特有的信息，*file.incarnation* 对确保唯一性。下面是 USERS 表空间的 ASM 文件示例：

+DATA/dw/datafile/users.259.627432977

磁盘组名是+DATA，数据库名是 dw，它是 USERS 表空间的数据文件，如果决定创建 USERS 表空间的另一个 ASM 数据文件，则文件编号/具体名称对 259.627432977 可确保唯一性。

2. 数字名称

数字名称只在引用已有的 ASM 文件时使用。这允许仅通过磁盘组名和文件编号/具体名称对来引用已有的 ASM 文件。前面小节中 ASM 文件的数字名称是：

+DATA.259.627432977

3. 别名

在引用已有的对象或创建单个 ASM 文件时，可以使用别名。使用 alter diskgroup add alias 命令，可以为已有的或新的 ASM 文件创建更易读懂的名称，并且很容易与普通的 ASM 文件名区分，因为别名的结尾没有包含点的数字对(文件编号/具体名称对)，如下所示：

```
SQL> alter diskgroup data
  2     add directory '+data/purch';
```

```
Diskgroup altered.
SQL> alter diskgroup data
  2     add alias '+data/purch/users.dbf'
  3     for '+data/dw/datafile/users.259.627432977';
Diskgroup altered.
SQL>
```

4. 具有模板名称的别名

具有模板的别名只可以在创建新的 ASM 文件时使用。创建新 ASM 文件时，模板提供了指定文件类型和标志的简略方式。下面是使用+DATA 磁盘组中新表空间的模板的一个别名示例：

```
SQL> create tablespace users2 datafile '+data(datafile)';
Tablespace created.
```

模板 datafile 指定条带化为 COARSE(粗糙)，普通冗余组为 MIRROR，高度冗余组为 HIGH，这是数据文件的默认设置。因为未完全限定名称，所以此磁盘组的 ASM 名称如下所示：

```
+DATA/dw/datafile/users2.267.627782171
```

稍后在 4.2.6 小节中将更多地讨论 ASM 模板。

5. 不完整的名称

不完整的文件名格式可用于单文件创建或多文件创建操作。可以只指定磁盘组名，并且根据文件类型使用默认的模板，如下所示：

```
SQL> create tablespace users5 datafile '+data1';
```

Tablespace created.

6. 具有模板的不完整名称

和不完整的 ASM 文件名一样，具有模板的不完整文件名可用于单文件创建或多文件创建操作。无论实际的文件类型是什么，模板名都可确定文件的特征。

下面的示例创建一个表空间，但对这个新的表空间使用联机日志文件的条带化和镜像特征(细密条带化)代替了数据文件的属性(粗糙条带化)：

```
SQL> create tablespace users6 datafile '+data1(onlinelog)';
Tablespace created.
```

4.2.6 ASM 文件类型和模板

除了操作系统可执行文件之外，ASM 支持数据库使用的所有文件类型。表 4-4 包含了 ASM 文件类型的完整列表，“ASM 文件类型”和“标志”列是针对前面的 ASM 文件命名约定所介绍的内容。

表 4-4 ASM 文件类型

Oracle 文件类型	ASM 文件类型	标 志	默 认 模 板
控制文件	controlfile	cf(控制文件)或 bcf(备份控制文件)	CONTROLFILE
数据文件	datafile	tablespace name.file#	DATAFILE
联机日志	online_log	log_thread#	ONLINELOG
归档日志	archive_log	parameter	ARCHIVELOG
临时文件	temp	tablespace name.file#	TEMPFILE
RMAN 数据文件的备份部分	backupset	客户端指定	BACKUPSET
RMAN 增量备份部分	backupset	客户端指定	BACKUPSET
RMAN 归档日志的备份部分	backupset	客户端指定	BACKUPSET
RMAN 数据文件副本	datafile	tablespace name.file#	DATAFILE
初始参数	init	spfile	PARAMETERFILE
代理程序配置	drc	drc	DATAGUARDCONFIG

(续表)

Oracle 文件类型	ASM 文件类型	标 志	默 认 模 板
闪回日志	rlog	thread#_log#	FLASHBACK
改变跟踪位图	ctb	bitmap	CHANGETRACKING
自动备份	autobackup	客户端指定	AUTOBACKUP
数据泵的转储集	dumpset	dump	DUMPSET
跨平台的数据文件			XTRANSPORT

表 4-5 中介绍了表 4-4 的最后一列中所引用的默认 ASM 文件模板。

表 4-5　ASM 文件模板的默认值

系 统 模 板	外 部 冗 余	普 通 冗 余	高 度 冗 余	条　带　化
CONTROLFILE	不受保护的	双向镜像	三向镜像	细密
DATAFILE	不受保护的	双向镜像	三向镜像	粗糙
ONLINELOG	不受保护的	双向镜像	三向镜像	细密
ARCHIVELOG	不受保护的	双向镜像	三向镜像	粗糙
TEMPFILE	不受保护的	双向镜像	三向镜像	粗糙
BACKUPSET	不受保护的	双向镜像	三向镜像	粗糙
XTRANSPORT	不受保护的	双向镜像	三向镜像	粗糙
PARAMETERFILE	不受保护的	双向镜像	三向镜像	粗糙
DATAGUARDCONFIG	不受保护的	双向镜像	三向镜像	粗糙
FLASHBACK	不受保护的	双向镜像	三向镜像	细密
CHANGETRACKING	不受保护的	双向镜像	三向镜像	粗糙
AUTOBACKUP	不受保护的	双向镜像	三向镜像	粗糙
DUMPSET	不受保护的	双向镜像	三向镜像	粗糙

创建新的磁盘组时，从表 4-5 中默认模板复制而来的一组 ASM 文件模板与磁盘组一起保存。因此，可以改变单个的模板特征，并且只应用于它们驻留的磁盘组。换句话说，磁盘组+DATA1 中的 DATAFILE 系统模板可能有默认的粗糙条带化，而磁盘组+DATA2 中的 DATAFILE 模板可能有细密条带化。用户可以根据需要在每个磁盘组中创建自己的模板。

使用 DATAFILE 模板创建 ASM 数据文件时，默认情况下数据文件为 100MB，它可以自动扩展，并且最大尺寸是 32 767MB (32GB)。

4.2.7　管理 ASM 磁盘组

使用 ASM 磁盘组有很多方面的优点：改进 I/O 性能、增加可用性、简化添加磁盘到磁盘组或添加全新的磁盘组，从而允许在相同的时间内管理更多的数据库。理解磁盘组的组成部分并正确配置磁盘组，这是成功 DBA 的重要目标。

本节将深入研究磁盘组结构的细节；同时，将回顾与磁盘组相关的不同管理任务类型，并且显示如何将磁盘赋予故障组，如何镜像磁盘组，以及如何创建、删除和改变磁盘组；此外也将简要回顾 ASM 的 EM Database Control 接口；另外还要介绍 asmcmd 这一命令行实用程序，使用此命令可以浏览、复制和管理 ASM 对象。

1. 磁盘组的体系结构

本章前面定义过，磁盘组是作为一个单位管理的物理磁盘的集合。作为磁盘组一部分的每个 ASM 磁盘都有一个 ASM 磁盘名，可以由 DBA 赋予该磁盘名，也可以在将该磁盘赋予磁盘组时自动分配磁盘名。

使用粗糙条带化或细密条带化，在磁盘上对磁盘组中的文件条带化。粗糙条带化以每个 1MB 为单位将文件扩展到所有的磁盘。粗糙条带化适合于具有高度并发的小 I/O 请求的系统，例如 OLTP 环境。作为选择，细密条带化以 128KB 为单位扩展文件，它适合于传统的数据仓库环境或具有较低并发性的 OLTP 系统，可以

最大化单个 I/O 请求的响应时间。

2. 磁盘组镜像和故障组

在定义磁盘组中的镜像类型前，必须将磁盘分组到故障组中。故障组是磁盘组中的一个或多个磁盘，这些磁盘共享常见的资源，例如磁盘控制器，它的故障将造成磁盘组无法使用整个磁盘集。在大多数情况下，ASM 实例不知道给定磁盘的硬件和软件相关性。因此，除非专门将一个磁盘赋给故障组，否则磁盘组中的每个磁盘都赋给它自己的故障组。

一旦已经定义故障组，就可以定义磁盘组的镜像。可用于磁盘组中的故障组的数量可以限制可用于磁盘组的镜像类型。有 3 种可用的镜像类型：外部冗余、普通冗余和高度冗余。

外部冗余　外部冗余只需要一个磁盘位置，并且假设磁盘对于正在进行的数据库操作不是至关重要的，或者使用高可用性的硬件(例如 RAID 控制器)在外部管理磁盘。

普通冗余　普通冗余提供双向镜像，并且需要磁盘组中至少有两个故障组。故障组中的一个磁盘产生故障不会造成磁盘组的任何停机时间或数据丢失，除了对磁盘组中对象的查询有一些性能上的影响。当故障组的所有磁盘都处于联机状态时，读性能一般会得到提高，因为请求的数据在多个磁盘上可用。

高度冗余　高度冗余提供三向镜像，并且需要磁盘组中的至少 3 个故障组。对于数据库用户来说，任意两个故障组中的磁盘产生故障基本上不会有明显的表现，如同在普通冗余镜像中　那样。

镜像管理的级别非常低。被镜像的是盘区，而不是磁盘。此外，每个磁盘将具有每个磁盘上主要的和镜像的(次要的和第三位的)盘区。虽然在盘区级别中管理镜像会带来少量的系统开销，但它具有如下优点：将负载从失败的磁盘扩展到所有其他的磁盘，而不是一个磁盘。

3. 磁盘组的动态重新平衡

改变磁盘组的配置时，无论添加或删除故障组或故障组中的磁盘，都将自动进行动态的重新平衡，按比例将数据从磁盘组的其他成员重新分配到磁盘组的新成员。当数据库联机并且用户可使用该数据库时，这种重新平衡就可以发生。通过将初始参数 ASM_POWER_LIMIT 的值调整为较低的值，可以控制对正在进行的数据库 I/O 的任何影响。

动态重新平衡不仅可以免除标识磁盘组中热点这种麻烦的、通常很容易产生错误的任务，也提供了将整个数据库从一组较慢的磁盘迁移到一组较快磁盘的自动方法，同时整个数据库保持联机。较快的磁盘作为已有磁盘组中新的故障组和较慢的磁盘一起添加，并且进行自动的重新平衡。重新平衡操作完成后，删除包含较慢磁盘的故障组，保留只有较快磁盘的磁盘组。为了使这一操作更为快速，可以在相同的 alter diskgroup 命令中启动 add 和 drop 操作。

作为示例，假定希望创建具有高度冗余的新磁盘组，保存用于新信用卡授权的表空间。使用视图 V$ASM_DISK，可以查看使用初始参数 ASM_DISKSTRING 发现的所有磁盘，以及这些磁盘的状态(换句话说，是否将其赋给已有的磁盘组)。下面是相关的命令：

```
SQL> select group_number, disk_number, name,
  2      failgroup, create_date, path from v$asm_disk;
GROUP_NUMBER DISK_NUMBER NAME       FAILGROUP            CREATE_DA PATH
------------ ----------- ---------- ---------- --------- ---------------
           0           0                                            /dev/raw/raw8
           0           1                                            /dev/raw/raw7
           0           2                                            /dev/raw/raw6
           0           3                                            /dev/raw/raw5
           2           1 RECOV_0001 RECOV_0001 08-JUL-07 /dev/raw/raw4
           2           0 RECOV_0000 RECOV_0000 08-JUL-07 /dev/raw/raw3
           1           1 DATA_0001  DATA_0001  08-JUL-07   /dev/raw/raw2
           1           0 DATA_0000  DATA_0000  08-JUL-07   /dev/raw/raw1
8 rows selected.
SQL>
```

在 8 个可用于 ASM 的磁盘中，只有 4 个磁盘被赋予 2 个磁盘组 DATA 和 RECOV，每个都在自己的故障组中。可以从视图 V$ASM_DISKGROUP 中获得磁盘组名。

```
SQL> select group_number, name, type, total_mb, free_mb
  2     from v$asm_diskgroup;
GROUP_NUMBER NAME          TYPE      TOTAL_MB    FREE_MB
------------ ---------    ------    ---------- -------
           1    DATA          NORMAL       24568       20798
           2    RECOV         NORMAL       24568       24090
SQL>
```

注意,如果有大量ASM磁盘和磁盘组,可以在GROUP_NUMBER列上连接两个视图,并且通过GROUP_NUMBER过滤查询结果。同时，从V$ASM_DISKGROUP中看到，两个磁盘组都是由两个磁盘组成的NORMAL REDUNDANCY组。

第一步是创建磁盘组：

```
SQL> create diskgroup data2 high redundancy
  2       failgroup fg1 disk '/dev/raw/raw5' name d2a
  3       failgroup fg2 disk '/dev/raw/raw6' name d2b
  4       failgroup fg3 disk '/dev/raw/raw7' name d2c
  5       failgroup fg4 disk '/dev/raw/raw8' name d2d;
Diskgroup created.
SQL>
```

查看动态性能视图，可以看到新的磁盘组出现在 V$ASM_DISKGROUP 中，故障组出现在 V$ASM_DISK 中：

```
SQL> select group_number, name, type, total_mb, free_mb
  2    from v$asm_diskgroup;
GROUP_NUMBER NAME       TYPE     TOTAL_MB    FREE_MB
------------ -------- ------ ---------- ----------
           1   DATA       NORMAL       24568       20798
           2   RECOV      NORMAL       24568       24090
           3   DATA2      HIGH         16376       16221
SQL> select group_number, disk_number, name,
  2      failgroup, create_date, path from v$asm_disk;
GROUP_NUMBER DISK_NUMBER NAME         FAILGROUP  CREATE_DA PATH
------------ ----------- ---------- ---------- -------- -------------
           3           3     D2D           FG4           13-JUL-07 /dev/raw/raw8
           3           2    D2C           FG3           13-JUL-07 /dev/raw/raw7
           3           1     D2B           FG2           13-JUL-07 /dev/raw/raw6
           3           0     D2A           FG1           13-JUL-07 /dev/raw/raw5
           2           1     RECOV_0001 RECOV_0001 08-JUL-07 /dev/raw/raw4
           2           0     RECOV_0000 RECOV_0000 08-JUL-07 /dev/raw/raw3
           1           1     DATA_0001  DATA_0001  08-JUL-07 /dev/raw/raw2
           1           0     DATA_0000  DATA_0000  08-JUL-07 /dev/raw/raw1
8 rows selected.
SQL>
```

然而，如果磁盘空间非常紧密，则不需要 4 个成员。对于高度冗余的磁盘组，只需要 3 个故障组，因此接下来删除磁盘组，并且重新创建只有 3 个成员的磁盘组：

```
SQL> drop diskgroup data2;
Diskgroup dropped.
```

如果磁盘组有任何不同于磁盘组元数据的数据库对象，则必须在 drop diskgroup 命令中指定 including contents。这是额外的安全措施，可确保不会无意中删除具有数据库对象的磁盘组。下面是相关的命令：

```
SQL> create diskgroup data2 high redundancy
  2    failgroup fg1 disk '/dev/raw/raw5' name d2a
  3    failgroup fg2 disk '/dev/raw/raw6' name d2b
  4    failgroup fg3 disk '/dev/raw/raw7' name d2c;
Diskgroup created.
SQL> select group_number, disk_number, name,
```

```
  2       failgroup, create_date, path from v$asm_disk;
GROUP_NUMBER DISK_NUMBER NAME          FAILGROUP  CREATE_DA PATH
------------ ----------- ----------   ---------- --------- ---------------
           0           3                                  13-JUL-07 /dev/raw/raw8
           3           2     D2C            FG3            13-JUL-07 /dev/raw/raw7
           3           1     D2B            FG2            13-JUL-07 /dev/raw/raw6
           3           0     D2A            FG1            13-JUL-07 /dev/raw/raw5
           2           1     RECOV_0001 RECOV_0001 08-JUL-07 /dev/raw/raw4
           2           0     RECOV_0000  RECOV_0000 08-JUL-07 /dev/raw/raw3
           1           1     DATA_0001  DATA_0001  08-JUL-07 /dev/raw/raw2
           1           0     DATA_0000  DATA_0000  08-JUL-07 /dev/raw/raw1
8 rows selected.
SQL>
```

在完成新磁盘组的配置之后，可以通过数据库实例创建新磁盘组中的表空间：

```
SQL> create tablespace users3 datafile '+DATA2';
Tablespace created.
```

因为 ASM 文件是 Oracle 管理文件(OMF)，所以在创建表空间时不需要指定其他的数据文件特征。

4. 磁盘组快速镜像重新同步

镜像磁盘组中的文件可以提高性能和可用性。但是，当修理磁盘组中的故障磁盘并将它重新联机时，重新镜像整个新磁盘很耗费时间。在有些情况下，由于磁盘控制器故障，需要将磁盘组中的某个磁盘进行脱机处理，此时并不需要对整个磁盘重新镜像，只有在故障磁盘停机期间发生改变的数据需要重新同步。因此，可以使用 Oracle Database 11g 中引入的 ASM 快速镜像重新同步特性。

要实现快速镜像重新同步，需要设置时间窗口，在此时间窗口内，当短暂的计划内或计划外故障发生时，ASM 并不自动删除磁盘组中的磁盘。在此短暂故障期间，ASM 跟踪所有发生改变的数据块，当不可用的磁盘重新联机时，只需要重新镜像改变的数据块，而不需要重新镜像整个磁盘。

要为 DATA 磁盘组设置时间窗口，必须先将 RDBMS 实例和 ASM 实例的磁盘组的兼容性级别设置为 11.1 或更高(只需要对磁盘组设置一次)：

```
SQL> alter diskgroup data set attribute
  2     'compatible.asm' = '11.1.0.0.0';
Diskgroup altered.
SQL> alter diskgroup data set attribute
  2     'compatible.rdbms' = '11.1.0.0.0';
Diskgroup altered.
SQL>
```

对 RDBMS 实例和 ASM 实例使用较高兼容性级别的唯一缺点是，只有版本号为 11.1.0.0.0 或更高的其他实例才能访问此磁盘组。接下来，设置磁盘组属性 disk_repair_time，如下例所示：

```
SQL> alter diskgroup data set attribute
  2     'disk_repair_time' = '2.5h';
Diskgroup altered.
SQL>
```

默认的磁盘修理时间是 3.6 小时，这对于大多数计划内和计划外的短暂停机来说应该绰绰有余。一旦磁盘重新联机，则运行此命令，通知 ASM 实例：磁盘 DATA_0001 重新联机：

```
SQL> alter diskgroup data online disk data_0001;
Diskgroup altered.
SQL>
```

此命令启动后台进程，将磁盘组中剩余磁盘上所有改变的盘区复制到现在重新联机的磁盘 DATA_0001。

5. 改变磁盘组

可以向磁盘组中添加或从磁盘组中删除磁盘，也可以改变磁盘组的大多数特征，而不需要重新创建磁盘组或影响磁盘组中对象上的用户事务。

在将磁盘添加到磁盘组时，需要把新的磁盘格式化以用于磁盘组中，并在后台执行重新平衡操作。本章前面提及，通过初始参数 ASM_POWER_LIMIT 可以控制重新平衡的速度。

继续前一节中的示例，假定您决定添加最近可用的裸磁盘到磁盘组，以便改进磁盘组 DATA 的 I/O 特征，具体如下：

```
SQL> alter diskgroup data
  2     add failgroup d1fg3 disk '/dev/raw/raw8' name d1c;
Diskgroup altered.
```

该命令立刻返回，并且在后台进行格式化和重新平衡。然后，通过检查 V$ASM_OPERATION 视图来检查重新平衡操作的状态：

```
SQL> select group_number, operation, state, power, actual,
  2     sofar, est_work, est_rate, est_minutes from v$asm_operation;
GROUP_NUMBER OPERA STAT POWER ACTUA SOFAR EST_WORK EST_RATE EST_MINUTES
------------ ----- ---- ----- ----- ----- -------- -------- -----------
           1   REBAL RUN      1     1     3      964         60          16
```

因为完成重新平衡操作估计要用 16 分钟，所以可以决定分配更多的资源给重新平衡操作，并且改变当前重新平衡操作的功率极限：

```
SQL> alter diskgroup data rebalance power 8;
Diskgroup altered.
```

检查重新平衡操作的状态可确认估计的完成时间已经减少到 4 分钟，而不是原来的 16 分钟：

```
SQL> select group_number, operation, state, power, actual,
  2     sofar, est_work, est_rate, est_minutes from v$asm_operation;
GROUP_NUMBER OPERA STAT POWER ACTUA SOFAR   EST_WORK EST_RATE EST_MINUTES
------------ ----- ---- ----- ----- -----   -------- -------- -----------
           1    REBAL RUN      8     8    16      605      118           4
```

大约 4 分钟后，再次检查重新平衡操作的状态：

```
SQL> /
no rows selected
```

最后，可以通过 V$ASM_DISK 和 V$ASM_DISKGROUP 视图确认新磁盘的配置：

```
SQL> select group_number, disk_number, name,
  2   failgroup, create_date, path from v$asm_disk;
GROUP_NUMBER DISK_NUMBER NAME       FAILGROUP  CREATE_DA PATH
------------ ----------- ---------- ---------- --------- ---------------
           1           2     D1C          D1FG3        13-JUL-07 /dev/raw/raw8
           3           2     D2C          FG3           13-JUL-07 /dev/raw/raw7
           3           1     D2B          FG2           13-JUL-07 /dev/raw/raw6
           3           0     D2A          FG1           13-JUL-07 /dev/raw/raw5
           2           1     RECOV_0001 RECOV_0001  08-JUL-07 /dev/raw/raw4
           2           0     RECOV_0000 RECOV_0000 08-JUL-07 /dev/raw/raw3
           1           1     DATA_0001  DATA_0001    08-JUL-07 /dev/raw/raw2
           1           0     DATA_0000  DATA_0000    08-JUL-07 /dev/raw/raw1
8 rows selected.
SQL> select group_number, name, type, total_mb, free_mb
  2     from v$asm_diskgroup;
GROUP_NUMBER NAME       TYPE     TOTAL_MB    FREE_MB
------------ ---------- ------ ---------- ----------
           1 DATA       NORMAL      28662      24814
           2 RECOV      NORMAL      24568      24090
           3 DATA2      HIGH        12282      11820
SQL>
```

注意，磁盘组 DATA 仍然是普通冗余，即使它有 3 个故障组。然而，由于磁盘组中有额外的盘区副本，因此可以改进针对 DATA 磁盘组中对象的 select 语句的 I/O 性能。

表 4-6 中列出了其他的磁盘组 alter 命令。

表 4-6　磁盘组的 ALTER 命令

alter diskgroup 命令	说　　明
alter diskgroup ... drop disk	删除磁盘组中来自于故障组的磁盘，并且执行自动的重新平衡
alter diskgroup ... drop ... add	删除来自于故障组的磁盘，并且添加另一个磁盘，所有这些操作都在相同的命令中完成
alter diskgroup ... mount	使磁盘组可用于所有的实例
alter diskgroup ... dismount	使磁盘组不可用于所有的实例
alter diskgroup ... check all	验证磁盘组的内部一致性

6. EM Database Control 和 ASM 磁盘组

EM Database Control 也可以用于管理磁盘组。对于使用 ASM 磁盘组的数据库，单击 Administration 选项卡下面的 Disk Groups 链接可打开 ASM 实例的登录页面，如图 4-20 所示。记住，ASM 实例的验证只使用操作系统验证。图 4-21 展示了 ASM 实例的主页。

图 4-20　EM Database Control ASM 实例登录页面

验证 ASM 实例后，可以执行本章前面在命令行中执行的相同操作：安装和卸载磁盘组、添加磁盘组、添加或删除磁盘组成员。图 4-22 显示了 ASM 管理页面，而图 4-23 显示了磁盘组 DATA 的统计信息和选项。

图 4-21　EM Database Control ASM 实例主页

图 4-22　EM Database Control ASM 磁盘组管理页面

图 4-23　EM Database Control ASM 磁盘组统计信息

在图 4-23 的页面中，可以看到磁盘组中的新磁盘比其他磁盘小得多，这可能会影响性能，并浪费磁盘组中的磁盘空间。要使用 EM Database Control 删除故障组，则选中该成员磁盘的复选框，并单击 Remove 按钮。

其他 EM Database Control ASM 相关页面显示了磁盘组的 I/O 响应时间、为磁盘组定义的模板、对当前 ASM 实例有效的初始参数等。

7. 使用 asmcmd 命令

asmcmd 实用程序是 Oracle 10g 版本 2 的新增特性，它是一个命令行实用工具，提供了一种简单的方法，可以使用类似于 Linux 外壳命令(例如 ls 和 mkdir)的命令集，浏览和维护 ASM 磁盘组中的对象。ASM 实例所维护对象的分层性质适合于采用类似 Linux 文件系统中浏览和维护文件所使用的命令集。

在使用 asmcmd 之前，必须确保将环境变量 ORACLE_BASE、ORACLE_HOME 和 ORACLE_SID 设置为指向 ASM 实例。对于本章所使用的 ASM 实例，这些环境变量的设置如下所示：

```
ORACLE_BASE=/u01/app/oracle
ORACLE_HOME=/u01/app/oracle/product/11.1.0/db_1
ORACLE_SID=+ASM
```

此外，必须以 dba 组中的用户登录到操作系统，因为 asmcmd 实用程序使用 SYSDBA 权限连接到数据库。操作系统用户通常是 oracle，但也可以是 dba 组中的任何其他用户。

可以采用 asmcmd command 格式，一次使用一条 asmcmd 命令，或者也可以在 Linux 外壳提示符下输入 asmcmd 来交互式地启动 asmcmd。为获得可用命令列表，可以在 ASMCMD>提示符下输入 help，这会得到更多的详细信息。表 4-7 列出了 asmcmd 命令及其简单说明，注意有些 asmcmd 命令只在 Oracle Database 11g 中可用。

表 4-7　asmcmd 命令汇总

asmcmd 命令	是否只在 11g 中可用	说　　明
cd		改变目录到指定目录
cp	是	在 ASM 磁盘组之间复制文件，既可以在相同实例中复制，也可以在远程实例中复制
du		循环显示当前目录和所有子目录的总体磁盘利用率
exit		终止 asmcmd，并返回到操作系统外壳提示符
find		从指定目录开始查找名称的所有匹配(也可以使用通配符)
help		列出 asmcmd 命令
ls		列出当前目录的内容
lsct		列出当前 ASM 客户数据库的有关信息
lsdg		列出所有磁盘组及其属性
lsdsk	是	列出此 ASM 实例可见的所有磁盘
md_backup	是	为指定磁盘组创建元数据备份脚本
md_restore	是	从备份恢复磁盘组
mkalias		为系统生成的 ASM 文件名创建一个别名
mkdir		创建一个 ASM 目录
pwd		显示当前的 ASM 目录
remap	是	修理磁盘上遭到破坏或损坏的一系列物理块
rm		删除 ASM 文件或目录
rmalias		删除一个 ASM 别名，但不删除此别名的目标

当启动 asmcmd 命令时，从 ASM 实例的文件系统的根节点开始。与 Linux 文件系统不同，根节点不是由前导正斜杠(/)来指明，而是由加号(+)来指明，但以下各级目录则使用正斜杠。在此示例中，启动 asmcmd 命令，并查询现有的磁盘组以及所有磁盘组中使用的总磁盘空间：

```
[oracle@dw ~]$ asmcmd
ASMCMD> ls -l
State    Type    Rebal  Unbal  Name
MOUNTED  NORMAL  N      N      DATA/
MOUNTED  HIGH    N      N      DATA2/
MOUNTED  NORMAL  N      N      RECOV/
ASMCMD> du
Used_MB      Mirror_used_MB
   2143                4399
ASMCMD> pwd
+
ASMCMD>
```

与 Linux 外壳的 ls 命令一样，如果想要得到此命令所检索出来的对象的详细信息，可以在 ls 命令后追加-l 参数。ls 命令显示了本章所用的 ASM 实例中的 3 个磁盘组+DATA、+DATA2 和+RECOV。

另外需要注意的是，du 命令只显示已用的磁盘空间以及跨镜像磁盘组所用的总磁盘空间。想要获得每个磁盘组中的空闲空间量，则需要使用 lsdg 命令。

此示例查找文件名中包含字符串 user 的所有文件：

```
ASMCMD> pwd
+
ASMCMD> find . user*
+DATA/DW/DATAFILE/USERS.259.627432977
+DATA/DW/DATAFILE/USERS2.267.627782171
+DATA/purch/users.dbf
+DATA2/DW/DATAFILE/USERS3.256.627786775
ASMCMD> ls -l +DATA/purch/users.dbf
Type      Redund  Striped  Time     Sys  Name
                                    N    users.dbf =>
                                       +DATA/DW/DATAFILE/USERS.259.627432977
ASMCMD>
```

注意包含+DATA/purch/users.dbf 的这一行：find 命令查找所有 ASM 对象。在此例中，它查找一个别名以及与此模式相匹配的数据文件。

最后可以对外部文件系统甚至其他 ASM 实例进行文件备份。在此示例中，使用 cp 命令将数据库的 SPFILE 备份到主机文件系统的/tmp 目录中：

```
ASMCMD> pwd
+data/DW
ASMCMD> ls
CONTROLFILE/
DATAFILE/
ONLINELOG/
PARAMETERFILE/
TEMPFILE/
spfiledw.ora
ASMCMD> cp spfiledw.ora /tmp/BACKUPspfiledw.ora
source +data/DW/spfiledw.ora
target /tmp/BACKUPspfiledw.ora
copying file(s)...
file, /tmp/BACKUPspfiledw.ora, copy committed.
ASMCMD> exit
[oracle@dw ~]$ ls -l /tmp/BACKUP*
-rw-r-----  1 oracle oinstall 2560 Jul 13 09:47 /tmp/BACKUPspfiledw.ora
[oracle@dw ~]$
```

此示例也展示了数据库 dw 的所有数据库文件是如何存储在 ASM 文件系统中的。看起来这些数据库文件好像存储在传统的主机文件系统中，但实际上是由 ASM 进行管理的，其所提供的内置性能和冗余特性(为用于 Oracle Database 11g 已进行了优化)使 DBA 可以更轻松地管理数据文件。

从调整的角度来看，每个系统都存在性能瓶颈，且在不同的日子甚至不同的星期中，瓶颈可能从一个组件移动到另一个组件。性能设计的目标是确保应用程序以及相关硬件的物理限制—— I/O 吞吐率、内存大小、查询性能等—— 不会影响到业务性能。如果应用程序的性能限制了它所支持的业务过程，则必须调整应用程序。在设计过程中，必须估计应用程序环境的限制，包括硬件以及应用程序与数据库交互的设计。没有任何环境可以提供无限的计算功能，因此每个环境都会在某个性能点处失败。在设计应用程序的过程中，应该努力让环境的性能功能充分地为性能需求服务。

性能调整是每个数据库应用程序生命周期的一部分，越早发现性能问题(最好是在投入生产之前)，就越有可能成功解决它。本章前面曾提及，大多数性能问题并不是孤立的症状，而是系统设计的结果。因此，调整工作的重点应该是识别和修正严重影响性能的底层缺陷。

调整是四步过程中的最后一步：计划、实现和监控必须在此之前进行。如果只是为了调整而进行调整，则无法进行完整的活动循环，从而很可能永远无法解决造成性能问题的底层缺陷。

本书的其他部分将讨论大多数可以调整的数据库对象：例如，第 7 章详细介绍了撤销段。本章只讨论与这些对象的调整相关的活动，而在介绍这些对象的章节中介绍计划和监控活动。

从 Oracle Database 10g 开始，可以使用新增的调整工具和功能，包括自动工作负荷存储库(AWR)，在 Oracle Database 11g 中，这些调整工具及其性能得到很大的增强。为使用方便，同时为了利用很多自动监控和诊断工具，Oracle 建议日常使用 OEM Database Control 工具。但是，在介绍 OEM 工具之前，先来了解一些预备知识和指导原则，主动积极的有效调整方法需要这些预备知识和指导原则。

在下面的各小节中，将介绍下列领域的调整活动：

- 应用程序设计
- SQL
- 内存使用率
- 数据存储
- 数据操作

- 物理存储
- 逻辑存储
- 网络流量

8.1 调整应用程序设计

为什么DBA调整指南应该包括关于应用程序设计的章节呢？为什么要先介绍它呢？这是因为DBA所做的工作中对系统性能带来最大影响的就是应用程序设计。第5章中讨论了DBA真正参与到应用程序开发工作中的必要性。在设计应用程序中，可以采取一些步骤以便有效且适当地使用一些技术，下面各小节将分别进行描述。

8.1.1 有效的表设计

不论数据库设计得有多好，拙劣的表设计都将导致较差的性能。不仅如此，过度严格坚持关系表设计也会导致较差的性能。这是由于虽然在逻辑上希望完全的关系表设计(判断标准是满足第三范式甚至第四范式)，但在物理上则除OLTP环境外并不希望这样。

使用这种设计的问题在于，虽然它们精确地反映了应用程序的数据与其他数据关联的方式，但是它们没有反映用户访问该数据将使用的的一般访问路径。一旦评估用户的访问需求，完全的关系表设计将成为许多大型查询的不可实现的部分。一般来说，最先出现问题的地方是返回大量列的查询。这些列通常分散在一些表中，从而迫使在查询期间连接这些表。如果其中一个连接表较大，则整个查询的性能就会受到影响。

在为应用程序设计表时，开发人员首先应该开发满足第三范式的模型，然后考虑反规范化数据以满足特殊的需求：例如，根据大型静态表创建小型的总结表(或物化视图)。是否可以根据需求从大型静态表中动态派生数据？当然可以。但如果用户频繁地请求它，并且数据在很大程度上是不变的，则周期性地以用户请求的格式存储数据就非常有意义。

例如，一些应用程序在相同的表中存储历史数据和当前的数据。每个记录可能有一个时间戳列，从而记录集中的当前记录是具有最近时间戳的记录。每次用户查询表中的当前记录时，都要执行一个子查询，例如：

```
where timestamp_col =
  (select max(timestamp_col)
     from table
    where emp_no=196811)
```

如果连接两个这样的表，就有两个子查询。在小型数据库中，这可能不会引起性能问题，但随着表和行的数量的增加，性能问题将随之而来。将历史数据和当前数据区分开来，或者在单独的表中存储历史数据，都将涉及 DBA 和开发人员的更多工作，但这样做可以改善应用程序的长期性能。

以用户为中心进行表设计，而不是以理论为中心进行表设计，这将产生能够更好满足用户需求的系统。这并不是说不应该使用 3NF 和 4NF 方法来设计数据库：恰恰相反，这么设计是一个很好的起点，可以揭示业务需求和物理数据库设计的先决条件。物理数据库设计选项包括将一个表分为多个表，以及相反的操作：将多个表组合为一个表。其重点是以最直接的路径为用户提供所需格式的数据。

8.1.2 CPU 需求的分布

当进行了有效的设计并拥有适当的硬件时，Oracle 数据库应用程序应该能够处理 I/O 请求，而不需要过多的等待；使用内存区域，而不需要交换和分页磁盘的内存；使用 CPU，而不会生成高负载平均值。由一个进程读入到内存中的数据将存储在内存中，并且在将该数据从内存中移除之前由许多进程重用。通过共享的 SQL 区域重用 SQL 命令，可以进一步减少系统上的负荷。

如果系统的 I/O 负荷减少，CPU 负荷就可能增加。可以通过以下这些方法来管理 CPU 资源：

- 调度 CPU 负载。应该把长期运行的批处理查询或更新程序安排在非高峰时刻运行。不是在联机用户正在执行事务时以较低的操作系统优先级运行它们，而是在适当的时间以普通的操作系统优先级运行它们。维持它们的普通优先级，同时适当地调度作业，这将最小化潜在的锁定、撤销和 CPU 冲突。
- 利用各种机会，在物理上将 CPU 需求从一个服务器转移到另一个服务器。任何情况下，尽可能隔离数据库服务器和应用程序的 CPU 需求。利用本书关于连网章节中描述的数据分布技术，可以将数据存储在最适合的位置，并且可以将应用程序的 CPU 需求与数据库的 I/O 需求分离开来。
- 考虑使用 Oracle 的实时应用集群(RAC)技术，将单个数据库的数据库访问需求扩展到多个实例。关于 RAC 特性的深入介绍以及 RAC 数据库的创建步骤，请参见第 10 章。
- 使用数据库资源管理功能。可以使用数据库资源管理器来建立资源分配计划和资源消费者组。可以使用 Oracle 的功能来改变可用于消费者组的资源分配。请查看第 5 章，了解通过数据库资源管理器来创建和实现资源消费者组以及资源计划。
- 使用并行查询将 SQL 语句的处理需求分布到多个 CPU。几乎每个 SQL 命令都可以使用并行性，包括 select、create table as select、create index、recover，以及 SQL*Loader 直接路径加载选项。

事务的并行化程度取决于为事务定义的并行性程度。每个表都有一个已定义的并行性程度，并且一个查询可以通过使用 PARALLEL 提示来重写默认的并行性程度。Oracle 评估可用于服务器上的 CPU 数量以及存储表数据的磁盘数量，从而确定默认的并行性程度。

在实例级别上设置最大的可用并行性。PARALLEL_MAX_SERVERS 初始参数用于设置最大数量的并行查询服务器进程，数据库中的所有进程可以在任意一个时间内使用这些服务器进程。例如，对于您的实例，如果设置 PARALLEL_MAX_SERVERS 为 32，并且运行一个查询，将 30 个并行查询服务器进程用于它的查询和排序操作，则数据库中剩余的所有用户只有 2 个并行查询服务器进程可用。因此，需要认真管理允许查询和批处理操作具有的并行性。当设置 PARALLEL_ADAPTIVE_MULTI_USER 参数为 TRUE 时，应在使用并行执行的多用户环境中启用为改进性能而设计的自适应算法。该算法自动减少根据查询启动时的系统负载而请求的并行性程度。有效的并行性程度或者是基于默认的并行性程度，或者是通过用归约因子除表或提示而得出的程度。

对于每个表，可以通过 create table 或 alter table 命令的 parallel 子句来设置默认的并行性程度。并行性程度可告诉 Oracle 为操作的每个部分尝试使用多少并行查询服务器进程。例如，如果执行表扫描和数据排序操作的查询的并行性程度为 5，则可以使用 10 个并行查询服务器进程：5 个用于扫描，剩下的 5 个用于排序。也可以在创建索引时通过 create index 命令的 parallel 子句为其指定并行性程度。

最初的最小并行查询服务器进程数量可以通过 PARALLEL_MIN_SERVERS 初始参数进行设置。一般来说，应该将该参数设置为非常小的数量(小于 5)，除非在一天的所有时间中都活跃地使用系统。设置该参数为较低的值，这将迫使 Oracle 重复启动新的查询服务器进程，但这将极大地减少低使用周期期间由空闲并行查询服务器进程保留的内存量。如果为 PARALLEL_MIN_SERVERS 设置较大的值，则服务器上可能会频繁地有空闲的并行查询服务器进程，这些空闲进程占据着前面获得的内存，但不执行任何功能。

并行化操作将处理需求分布到多个 CPU 上，然而应该谨慎地使用这些特性。如果为大型查询使用值为 5 的并行性程度，将会有 5 个访问数据的单独进程。如果有许多访问数据的进程，则可能造成对存储数据的磁盘的争用，从而损害性能。使用并行查询时，应该有选择地将其应用于其中的数据很好地分布在许多物理设备上的表。同时，应该避免将其用于所有的表。前面说过，单个查询可能使用所有可用的并行查询服务器进程，从而消除数据库中所有剩余事务的并行性。

8.1.3 有效的应用程序设计

除了应用程序设计外，本章后面描述的主题是 Oracle 应用程序的几个一般性指导原则。

首先，它们应该最小化请求数据库中数据的次数。解决方法包括使用序列、PL/SQL 块以及表的反规范化。可以使用分布的数据库对象(如物化视图)来帮助减少查询数据库的次数。

注意：

如果执行太频繁,即使是效率稍低的 SQL 也会影响数据库的性能。生成较少或没有物理 I/O 读取的 SQL 仍然会消耗 CPU 资源。

其次，相同应用程序的不同用户应该以非常类似的方式查询数据库。一致的访问路径可增加通过 SGA 中的可用信息解决请求的可能性。数据的共享不仅包括检索的表和行，还有使用的查询。如果查询相同，则查询的解析版本就可能已经存在于共享的 SQL 池中，从而减少处理查询所需的时间量。优化器中的游标共享增强可增加共享池中语句重用的可能性，但需要在设计应用程序时注意语句重用。

第三点，应该限制使用动态 SQL。按照定义，动态 SQL 直到运行时才被定义。应用程序的动态 SQL 第一次可以选择几行，第二次对有序表执行若干完整的表扫描，第三次以不注意的方式执行迪卡尔连接(或在 select 语句中使用 cross join 关键字有意识地执行迪卡尔连接)。另外，没有办法可以保证动态生成的 SQL 语句在语法上是正确的，直到运行时。动态生成的 SQL 是一把双刃剑：既具有根据用户输入动态创建 SQL 的灵活性，又可能对内部应用程序和外部网站应用程序产生 SQL 攻击。

第四点，应该最小化打开和关闭数据库中会话的次数。如果应用程序重复打开会话，执行少量的命令，然后关闭会话，则 SQL 的性能就可能是整体性能中的次要因素。会话管理可能比应用程序中的其他任何步骤都花费更多的时间。

使用存储过程时，相同的代码可能通过利用共享池多次执行。也可以手动编译过程、函数和程序包，从而避免运行时编译。创建过程时，Oracle 自动编译它。如果该过程以后变得无效，则数据库必须在执行它之前重新编译。为了避免在运行时导致这种编译开销，使用如下所示的 alter procedure 命令：

```
alter procedure MY_RAISE compile;
```

可以通过 DBA_SOURCE 视图中的 Text 列查看数据库中所有过程的 SQL 文本。USER_SOURCE 视图将显示执行查询的用户拥有的过程。程序包、函数和程序包主体的文本也可以通过 DBA_SOURCE 和 USER_SOURCE 视图访问，这些视图都引用名为 SYS.SOURCE$的表。

前面讨论的前两种设计指导原则，即限制用户访问数量以及协调他们的请求，都需要应用程序开发人员尽可能多地了解如何使用数据以及涉及的访问路径。出于这一原因，用户应该参与应用程序设计，如同参与表设计一样，这一点至关重要。如果用户花费较长的时间和数据建模者绘制表的结构，而花费较小的时间和应用程序开发人员讨论访问路径，则应用程序将很可能无法满足用户的需求。访问路径应该作为数据建模练习的一部分进行讨论。

8.2 调整 SQL

和应用程序设计一样，SQL 语句的调整看起来完全不是 DBA 的职责。然而，DBA 应该参与评审作为应用程序的一部分而编写的 SQL。设计良好的应用程序可能仍然会经历性能问题—— 如果它所使用的 SQL 没有经过很好调整的话。在设计合理的数据库中，应用程序设计和 SQL 问题造成了大多数的性能问题。

调整 SQL 的关键是最小化数据库查找数据所使用的搜索路径。在大多数 Oracle 表中，每一行都有一个与之关联的 RowID。RowID 包含有关行的物理位置的信息：它的文件、文件中的块以及数据库块中的行。

执行没有 where 子句的查询时，数据库通常执行完整的表扫描，读取表中的每一个块。在完整的表扫描期间，数据库定位表的第一个块，然后按顺序读取表中其他所有的块。对于大型表，完整的表扫描可能非常消耗时间。

查询特殊的行时，数据库可能使用索引来帮助加速所需行的检索。索引将表中的逻辑值映射到它们的 RowID：RowID 又将它们映射到特殊的物理位置。索引可能是唯一的—— 在这种情况下，每个值不会出现两次—— 也可以是非唯一的。索引只存储索引列中 NOT NULL 值的 RowID。

可以同时索引若干列，这称为“联结索引”或“组合索引”，如果查询的 where 子句中使用了它的第一列，则使用这种查询。优化器也可以使用“跳跃扫描”方法。在这种方法中将使用联结索引，即使查询的 where 子句中没有使用它的第一列。

索引必须裁剪为需要的访问路径。考虑具有 3 列的联结索引情况，如同下面的程序清单所示，在 EMPLOYEE 表的 City、State 和 Zip 列上创建该索引：

```
create index CITY_ST_ZIP_NDX
on EMPLOYEE(City, State, Zip)
tablespace INDEXES;
```

如果执行如下形式的查询：

```
select * from EMPLOYEE
       where State='NJ';
```

则索引的第一列(City)就不在 where 子句中。Oracle 可以使用两种类型的基于索引的访问来检索行：索引的跳跃扫描和索引的完整扫描。优化器将根据索引的统计信息来选择执行路径：索引的大小、表的大小以及索引的选择性。如果用户频繁运行这种类型的查询，则可能需要重新排序索引的列，将 State 列放在第一位，从而反映实际的使用模式。

索引范围扫描是另一种基于索引的优化方法,Oracle 使用这种方法可以高效率地检索选择的数据。当 where 子句中的变量等于、小于或大于指定常量，而且如果索引由多个部分组成，此变量是第一列时，则 Oracle 使用索引范围扫描。如果想要按索引顺序返回行，则不要求 order by 子句，如下面示例所示，在此例中，查找在 2007 年 8 月 1 日之前雇佣的雇员：

```
select * from EMPLOYEE where hire_date < '1-AUG-2007';
```

最重要的是表的数据应该尽可能有序。如果用户正在频繁执行范围查询—— 选择在特定范围内的值—— 则使数据有序就可能在解决查询时只需要读取较少的数据块，从而改进性能。索引中有序的条目将指向表中一组邻近的块，而不是分散在整个数据文件中的块。

例如，考虑下列类型的范围查询：

```
select *
        from EMPLOYEE
      where Empno between 1 and 100;
```

如果 EMPLOYEE 表中的物理记录按照 EMPNO 列排序，则该范围查询就只需要读取较少的数据块。为了保证行在表中适当地排序，将记录提取到一个平面文件(或另一个表)，在该处排序这些行，然后删除旧的行，并从有序的数据集中重新加载它们。另外，应该使用联机段收缩来收回 DML 活动频繁的表处于高水位之下的空闲空间碎片，这可以提高缓存利用率，且在全表扫描中只需要扫描较少的块。使用 alter table . . . shrink space 命令来压缩表中的空闲空间。

8.2.1 顺序对加载速率的影响

索引影响查询和数据加载的性能。在 insert 操作期间，行的顺序对加载性能有重大影响。即使是在具有大量索引的环境中，在 insert 操作之前对行进行适当的排序都可以将加载性能改进 50%。

在索引增长时，Oracle 分配新的块。如果在最近一个条目之后添加新的索引条目，则将新的条目添加到索引中的最后一个块。如果新的条目造成 Oracle 超出块中可用的空间，则将该条目移动到新的块。这种块分配对性能具有非常小的影响。

如果插入的行不是有序的，新的索引条目将写入到已有的索引节点块。如果块中没有更多的空间来添加新值，且该块不是索引中的最后一个块，则块的条目将一分为二。一半的索引条目将保留在原始的块中，而另一半将移动到新的块中。结果，性能在加载期间(因为产生了额外的空间管理活动)以及查询期间(因为索引包含更多未使用的空间，需要为相同数量的条目读取而读取更多的块)受到损害。

注意：

当索引增加其内部层次数量时，加载性能就会严重地降低。为了查看层次数量，分析索引，然后从DBA_INDEXES 选择它的 B 层次列值。

由于 Oracle 采用内部管理索引的方法，所以每次添加新的索引时，加载速率都会受到影响(因为对于多个列，插入的行不可能正确排序)。从加载速率的角度来看，赞成使用较少的多列索引，而不是使用多个单列索引。

8.2.2 其他的索引选项

如果数据不是非常有选择性，可以考虑使用位图索引。如同第 16 章中所描述，对于针对具有非常少不同值的大型静态数据集的查询，位图索引最为有效。可以在同一个表上创建位图索引和普通的(B-树)索引，Oracle 将在查询处理期间动态执行任何必需的索引转换。查看第 16 章以了解使用位图索引的详细信息。

注意：

避免在由联机事务修改的表上创建位图索引，但数据仓库表非常适合采用位图索引。

如果频繁地同时查询两个表，则使用集群就是改进性能的有效方法。根据它们的逻辑值(集群键)，集群在相同的物理数据块中存储来自多个表中的行。

将列值与确切的值(而不是某个范围内的值)进行比较的查询称为“等价查询”。根据某一行在集群键列中的值，散列集群在特定的位置存储该行。每次插入一行时，使用它的集群键值来确定应该将其存储在哪个块中，可以在查询期间使用这种相同的逻辑来快速查找检索所需的数据块。设计散列集群是为了改进等价查询的性能，它们对改进前面讨论的范围查询的性能没有任何帮助。对于范围查询、强制全表扫描的查询、或者频繁更新的散列集群，性能将糟得多。

反向索引为等价查询提供了另一种调整解决方案。在反向索引中，以倒序存储索引的字节。在传统的索引中，两个连续的值彼此相邻地存储在一起。在反向索引中，连续的值不会彼此相邻地存储在一起。例如，在反向索引中，值 2004 和 2005 分别存储为 4002 和 5002。虽然反向索引不适合于范围扫描，但如果执行许多等价查询，反向索引就可以减少索引块的争用。反向键索引可能常常需要重新构建才能很好地执行。反向键索引也应该包含一个大的 PCTFREE 值，以便允许插入。

注意：

不可以反向位图索引。

可以在涉及列的表达式上创建基于函数的索引，这种查询不可以在 Name 列上使用 B-树索引：

```
select * from EMPLOYEE
 where UPPER(Name) = 'JONES';
```

然而，下列查询：

```
select * from EMPLOYEE
 where Name = 'JONES';
```

可以使用，因为第二个查询没有在 Name 列上执行函数。可以不在 Name 列上创建索引，取而代之的是，在列表达式 UPPER(Name)上创建索引，如同下面示例所示：

```
create index EMP_UPPER_NAME on
EMPLOYEE(UPPER(Name));
```

虽然基于函数的索引非常有用，但在创建它们时应考虑下面的问题：

- 是否可以限制将用于列上的函数？如果可以，是否可以限制在列上执行所有函数？
- 对于额外的索引，是否有足够的存储空间？
- 删除表时，将比以前删除更多的索引(因此删除更多的盘区)。这对删除表所需的时间有何影响？(如果正在使用本地管理的表空间，则不必太多地考虑这一点。如果正在运行 Oracle Database 10g 或更高版本，则应该正在使用本地管理的表空间。)

基于函数的索引非常有用，但应该有节制地实现它们。在表上创建的索引越多，所有的 insert、update 和 delete 操作所花费的时间就越长。当然，这适用于在表上创建任何额外的索引，而与索引类型无关。

文本索引使用 Oracle 的文本选项(Oracle Text)来创建并管理单词列表及其出现次数，这与书籍的索引方式是相似的。文本索引最常用于支持使用通配符搜索单词某一部分的应用程序。

分区表可以有横跨所有分区的索引(全局索引)或按照表分区进行分区的索引(局部索引)。从查询调整的角度来看，可能更倾向于使用局部索引，因为它们比全局索引包含较少的条目。

8.2.3 生成解释计划

如何确定数据库将使用哪些访问路径来执行查询？可以通过 explain plan 命令查看该信息。该命令将计算查询的执行路径，并且将它的输出放置在数据库的表(名为 PLAN_TABLE)中。下面的程序清单显示了一个样例 explain plan 命令：

```
explain plan
 for
select *
  from BOOKSHELF
 where Title like 'M%';
```

该命令的第一行告诉数据库，它将为查询解释它的执行计划，而不是实际地执行查询。可以有选择地包括 set Statement_ID 子句，用于标记 PLAN_TABLE 中的解释计划。在关键字 for 后面，列出了将进行分析的查询。

运行该命令的账户必须在它的模式中具有计划表。Oracle 提供了创建该表所需的 create table 命令。文件 utlxplan.sql 位于$ORACLE_HOME/rdbms/admin 目录下。用户可以运行该脚本，在他们的模式中创建

表。

注意：

在每次 Oracle 升级后，应该删除并重新创建计划表，因为升级脚本可能添加了新的列。

使用 DBMS_XPLAN 过程查询计划表：

```
select * from table(DBMS_XPLAN.DISPLAY);
```

也可以使用 Oracle 在$ORACLE_HOME/rdbms/admin/utlxpls.sql 中提供的脚本查询串行执行的计划表，或使用$ORACLE_HOME/rdbms/admin/utlxplp.sql 中提供的脚本查询并行执行的计划表。

该查询将报告数据库解决查询必须执行的操作类型。输出将以层次结构的方式显示查询执行的步骤，同时图示各步骤之间的关系。例如，可能看到一个基于索引的步骤，该步骤具有一个 TABLE ACCESS BY INDEX ROWID 步骤作为它的父步骤，表明首先处理索引步骤，并且从该索引返回的 RowID 用于检索表中特殊的行。

可以使用 SQL*Plus 中的 set autotrace on 命令，自动生成 explain plan 输出并跟踪运行的每个查询的信息。直到查询完成后，自动跟踪生成的输出才会显示。然而，不需要运行该命令就可以生成 explain plan 输出。为了启用自动跟踪生成的输出，要么必须在某个模式中创建计划表，在该模式中将使用自动跟踪实用程序；要么必须在 SYSTEM 模式中创建计划表，该模式授权访问将使用自动跟踪实用程序的模式。脚本 plustrace.sql 位于$ORACLE_HOME/sqlplus/ admin 目录中，必须在执行 set autotrace on 命令之前作为 SYS 运行该脚本。在执行 set autotrace on 之前，用户还必须启用 PLUSTRACE 角色。对于 Oracle Database 10g 或更高版本的安装或升级，此脚本自动运行。

注意：

为了在不运行查询的情况下显示解释计划的输出，可以使用 set autotrace traceonly explain 命令。

如果使用并行查询选项或查询远程数据库，set autotrace on 输出的额外部分将显示由并行查询服务器进程执行的查询的文本，或者是在远程数据库中执行的查询的文本。

为了禁止自动跟踪特性，可以使用 set autotrace off 命令。

下面的程序清单显示了如何打开自动跟踪和生成解释计划：

```
set autotrace traceonly explain
select *
  from BOOKSHELF
 where Title like 'M%';
Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=ALL_ROWS (Cost=3 Card=2 Bytes=80)
   1    0   TABLE ACCESS (BY INDEX ROWID) OF 'BOOKSHELF' (TABLE) (Cost
          =3 Card=2 Bytes=80)
   2    1     INDEX (RANGE SCAN) OF 'SYS_C004834' (INDEX (UNIQUE)) (Co
```

```
st=1 Card=2)
```

为了理解解释计划，由里到外读取层次结构中的操作序列，直到到达具有相同缩进级的一组操作，然后从上向下读取。在本示例中，没有具有相同缩进级的操作，因此由里到外读取操作序列。第一个操作是索引范围扫描，接下来是表访问，SELECT STATEMENT 操作向用户显示输出。每个操作具有一个 ID 值(第一列)和父 ID 值(第二个数字，在最顶端的操作中，该数字为空)。在更为复杂的解释计划中，可能需要使用父 ID 值来确定操作的顺序。

该计划显示了通过 TABLE ACCESS BY INDEX ROWID 操作获得返回给用户的数据。通过唯一索引的索引范围扫描提供 RowID。

每个步骤都被赋予一个“成本”。成本是累积的，反映了该步骤的成本加上其所有子步骤的成本。可以使用成本值来标识在查询的整体成本中占用最大数量的步骤，然后将它们作为主要的调整目标。

在评估 explain plan 命令的输出时，应该确保查询使用最有选择性的索引(即最接近唯一性的索引)。如果使用非选择性的索引，可能会迫使数据库执行不必要的读取以解决查询。SQL 调整的完整讨论超出了本书的范围，但应该将调整工作的重点放在确保资源最密集的 SQL 语句尽可能使用最有选择性的索引上。

一般来说，面向事务的应用程序(例如，用于数据录入的多用户系统)根据返回查询第一行所花费的时间来判断性能。对于面向事务的应用程序，应该将调整工作的重点放在使用索引以减少数据库对查询的响应时间上。

如果应用程序是面向批处理的(具有大型事务和报表)，关注的重点应该是改善完成整个事务所花费的时间，而不是改善从事务中返回第一行所花费的时间。改善事务的整体吞吐量可能需要使用完整的表扫描来代替索引访问，并且可能改进应用程序的整体性能。

如果应用程序分布在多个数据库上，则应该减少在查询中使用的数据库链接的次数。如果在查询期间频繁访问远程数据库，则每次访问远程数据时就都会花费访问远程数据库的成本。即使访问远程数据的成本非常低，但数千次访问远程数据最终会对应用程序的性能带来影响。请查看本章第 8.7 节，了解关于分布式数据库的额外调整建议。

8.3 调整内存使用率

从 Oracle 10g 开始，可以使用自动工作负荷存储库(AWR)工具箱来收集和管理统计数据(本章后面将描述这一点)。从 Oracle 11g 开始，可以使用新的初始参数(如 MEMORY_TARGET)进一步使 Oracle 使用的总体内存自动化—— 当您没有时间读 AWR 报告时可以帮助自动调整数据库！

Oracle 通过“最近最少使用”(LRU)算法来管理数据块缓冲区缓存和共享池。留出预先设置的区域以保存值，填满该区域时，从内存中移除最近最少使用的数据并写回到磁盘。大小调整合适的内存区域可在内存中保留最频繁访问的数据，访问最少使用的数据则需要物理读取。

可以通过 V$SQL 视图查看在数据库中执行逻辑读和物理读的查询。V$SQL 视图报告为当前在共享池中的每个查询执行的逻辑读和物理读的累积数量，以及执行每个查询的次数。下面的脚本显示了共享池中的每个查询的 SQL 文本，最先列出的是 I/O 最密集的查询。该查询也显示了每次执行时的逻辑读数量(缓冲区获取)：

```
select Buffer_Gets,
       Disk_Reads,
       Executions,
       Buffer_Gets/Executions B_E,
       SQL_Text
  from V$SQL where executions != 0
 order by Disk_Reads desc;
```

如果已经刷新共享池，则不再可以通过 V$SQL 访问刷新之前执行的查询。然而假设用户仍然处于登录状态，则这些查询的影响仍然可以看到。V$SESS_IO 视图记录了为每个用户的会话执行的逻辑读和物理读

的累积数量。可以查询 V$SESS_IO 以获得每个会话的命中率，如同下面的程序清单所示：

```
select SESS.Username,
       SESS_IO.Block_Gets,
       SESS_IO.Consistent_Gets,
       SESS_IO.Physical_Reads,
       round(100*(SESS_IO.Consistent_Gets
          +SESS_IO.Block_Gets-SESS_IO.Physical_Reads)/
         (decode(SESS_IO.Consistent_Gets,0,1,
             SESS_IO.Consistent_Gets+SESS_IO.Block_Gets)),2)
               session_hit_ratio
  from V$SESS_IO sess_io, V$SESSION sess
 where SESS.Sid = SESS_IO.Sid
   and SESS.Username is not null
 order by Username;
```

为了查看当前位于数据块缓冲区缓存中的块的对象，查询 SYS 的模式中的 X$BH 表，如同下面的查询所示(注意，输出中没有包括 SYS 和 SYSTEM 对象，因此 DBA 可以重点关注出现在 SGA 中的应用程序表和索引)：

```
select Object_Name,
       Object_Type ,
```

```
       count(*) Num_Buff
  from X$BH a, SYS.DBA_OBJECTS b
 where A.Obj = B.Object_Id
   and Owner not in ('SYS','SYSTEM')
 group by Object_Name, Object_Type;
```

注意：

如果未以 SYS 用户连接到数据库，则查询 V$CACHE 中的 Name 和 Kind 列可以看到类似的数据。

数据块缓冲区缓存中有多个缓存区域：

- DEFAULT 缓存：对于使用数据库的默认数据库块大小的对象，这是标准的缓存。
- KEEP 缓存：该缓存区域专门用于希望任何时候都保留在内存中的对象。一般来说，该区域用于具有非常少的事务的小型表。此缓存非常适合于从表中查找如州代码、邮政编码和销售人员数据等信息。
- RECYCLE 缓存：该缓存区域专门用于希望从内存中快速刷新的对象。类似于 KEEP 缓存，RECYCLE 缓存隔离内存中的对象，从而它们不会干扰到 DEFAULT 缓存的普通机能。
- 块大小特定的缓存：Oracle 支持一个数据库中的多个数据库块大小。必须为每个非默认的数据库块大小创建一个缓存。

使用 SGA 的所有区域—— 数据块缓冲区、字典缓存以及共享池—— 时，重点应该在多个用户之间的共享数据上。每个区域应该足够大，从而可以保存数据库中最经常请求的数据。在共享池的情况中，它应该足够大，可以保存最常用查询的解析版本。适当调整大小时，SGA 中的内存区域可以极大地改善单个查询的性能和数据库的整体性能。

KEEP 和 RECYCLE 缓冲区池的大小不会减少数据块缓冲区缓存中的可用空间。为了使表可以使用一个新的缓冲区池，通过表的 storage 子句中的 buffer_pool 参数指定缓冲区池的名称。例如，如果希望从内存中快速删除表，可以将其赋予 RECYCLE 池。默认的池命名为 DEFAULT，因此可以在以后使用 alter table 命令将表重定向到 DEFAULT 池。下面是将表赋予 KEEP 缓冲区池的一个示例：

```
create table state_cd_lookup
 (state_cd   char(2),
  state_nm   varchar2(50)
 )
storage (buffer_pool keep);
```

可以使用 LARGE_POOL_SIZE 初始参数来以字节为单位指定大型池分配堆的大小。在共享的服务器系统中，大型池分配堆用于会话内存，通过并行执行用于消息缓冲区，以及通过备份进程用于 I/O 缓冲区。默认情况下，不创建大型池。

从 Oracle Database 10g 开始，可以使用自动共享内存管理(Automatic Shared Memory Management，ASMM)。为了激活 ASMM，应为 SGA_TARGET 数据库初始参数设置非零值。设置 SGA_TARGET 为所需的 SGA 大小(即所有的缓存加在一起)后，然后可以设置其他与缓存相关的参数(DB_CACHE_SIZE、SHARED_POOL_SIZE、JAVA_POOL_SIZE 以及 LARGE_POOL_SIZE)都为 0；如果为这些参数提供值，则这些值将作为自动调整算法的下限。关闭并重启数据库，使这些改动生效；数据库然后开始有效地管理不同缓存的大小。通过 V$SGASTAT 动态性能视图，可以在任何时刻监控缓存的大小。Oracle Database 11g 将自动化又向前推进了一步：可以将 MEMORY_TARGET 设置为 Oracle 可用的总内存数量。MEMORY_TARGET 中指定的内存数量自动在 SGA 和 PGA 之间进行分配。当设置了 MEMORY_TARGET 时，SGA_TARGET 和 PGA_AGGREGATE_TARGET 被设置为 0，不必理会它们。

当数据库中的工作量改变时，数据库将改变缓存大小以反映应用程序的需求。例如，如果在夜间有大量批处理工作量，并且在白天有更为密集的联机事务工作量，数据库就可以在工作量改动时改变缓存大小。这些改动是自动发生的，不需要 DBA 的介入。如果在初始参数文件中为池指定一个值，Oracle 将使用该值作为池的最小值。

注意：

DBA 可以在缓冲区缓存中创建 KEEP 和 RECYCLE 池。KEEP 和 RECYCLE 池不会受到动态缓存大小调整的影响，它们也不是 DEFAULT 缓冲区池的一部分。

在 OEM 中，可以通过单击 Memory Parameters 选项查看动态内存管理是否已启用。Automatic Shared Memory Management 按钮可以设置为“Enabled”或“Disabled”。

可能希望有选择地“固定”程序包在共享池中。启动数据库后立刻固定程序包在内存中，这样将增加内存中具有足够大连续空闲空间的可能性。如同下面的程序清单所示，DBMS_SHARED_POOL 程序包中的 KEEP 过程指定固定在共享池中的程序包：

```
execute DBMS_SHARED_POOL.KEEP('APPOWNER.ADD_CLIENT','P');
```

相比于应用程序调整，固定程序包与应用程序管理更紧密地相关，但它对性能有一定的影响。如果可以避免动态管理成碎片的内存区域，则可以最小化 Oracle 在管理共享池时必须做的　工作。

8.3.1　指定 SGA 的大小

为了启用缓存的自动管理，可以设置 SGA_TARGET 初始参数为 SGA 的大小。

如果选择手动管理缓存，可以设置 SGA_MAX_SIZE 参数为 SGA 的大小。然后可以指定单个缓存的大小，可以在数据库运行时通过 alter system 命令动态改变这些大小。

也可以将 SGA_TARGET 设置为比 SGA_MAX_SIZE 小的大小。Oracle 使用 SGA_TARGET 设置单个缓存的初始值，且随着时间的推移其大小可以增长，以便占用更多的内存，最多可到 SGA_MAX_SIZE。这是一种很好的方法，可以确定在产品环境中部署数据库之前的总内存需求。

参　　数	说　　明
SGA_MAX_SIZE	SGA 可以增长到的最大大小
SHARED_POOL_SIZE	共享池的大小
DB_BLOCK_SIZE	数据库的默认数据库块大小
DB_CACHE_SIZE	以字节为单位指定的缓存大小
DB_*n*K_CACHE_SIZE	如果在一个数据库中使用多个数据库块大小，必须指定 DB_CACHE_SIZE 参数值以及至少一个 DB_*n*K_CACHE_SIZE 参数值。例如，如果标准数据库块大小为 4KB，则也可以通过 DB_8K_CACHE_SIZE 参数为 8KB 块大小的表空间指定缓存

例如，指定如下参数：

```
SGA_MAX_SIZE=1024M
SHARED_POOL_SIZE=220M
DB_BLOCK_SIZE=8192
DB_CACHE_SIZE=320M
DB_4K_BLOCK_SIZE=4M
```

使用这些参数时，4MB 将可用于从某些对象查询而来的数据，这些对象位于具有 4KB 块大小的表空间中。使用标准 8KB 块大小的对象将使用 160MB 缓存。当数据库打开时，可以通过 alter system 命令改变 SHARED_POOL_SIZE 和 DB_CACHE_SIZE 参数值。

SGA_TARGET 是动态参数，可以通过 Database Control 或使用 alter system 命令改变它。

SGA_TARGET 最大可以增加到 SGA_MAX_SIZE 的值。可以减少 SGA_TARGET 参数的值，直到任何一个自动调整的组件到达它的最小尺寸：用户指定的最小值或内部确定的最小值。这两个参数都可以用于调整 SGA。

8.3.2　使用基于成本的优化器

Oracle 的每个软件版本都对优化器添加了新的特性，并且对已有的特性进行了增强。有效使用基于成本的优化器需要定期分析应用程序中的表和索引。分析对象的频率取决于对象中的改动率。对于批处理事务应用程序，应该在每次大量处理事务后重新分析对象。对于 OLTP 应用程序，应该在基于时间的进度表上重新分析对象(例如，通过每周或每晚的进程)。

注意：
从 Oracle Database 10g 版本 1 开始，不支持基于规则的优化器。

可以通过执行 DBMS_STATS 程序包的过程收集关于对象的统计信息。如果分析表，也会自动分析它的

相关索引。可以分析模式(通过 GATHER_SCHEMA_STATS 过程)或特殊的表(通过 GATHER_TABLE_STATS)。也可以只分析索引列，从而加速分析过程。一般来说，每次分析表时，也应该分析表的索引。下面的程序分析了 PRACTICE 模式：

```
execute DBMS_STATS.GATHER_SCHEMA_STATS('PRACTICE', 'COMPUTE');
```

可以通过 DBA_TABLES、DBA_TAB_COL_STATISTICS 和 DBA_INDEXES 查看关于表和索引的统计信息。DBA_TAB_COLUMNS 中也提供了一些列级别的统计信息，但在此处提供这些统计信息只是为了严格地遵循向下兼容性。DBA_PART_COL_STATISTICS 中可以看到分区表的列的统计信息。

注意：
从 Oracle Database 10g 开始，在默认安装中，使用自动维护任务基础结构(AutoTask)自动在维护期间收集统计信息。

执行前面程序清单中的命令时，使用 compute statistics 选项分析属于 PRACTICE 模式的所有对象。也可以选择根据指定百分比的表行评估统计信息。

8.3.3 COMPUTE STATISTICS 选项的含义

在前一节的示例中，compute statistics 选项用于收集有关对象的统计信息。Oracle 也提供了 estimate statistics 选项，该选项将对象的统计信息基于部分数据的回顾。如果选择使用 estimate statistics，则分析尽可能多的表。可以指定进行分析的行的百分比：一般分析 20%就足够了。

提示：
在未来的 Oracle 版本中,可能无法使用 DBMS_STATS 程序包外部的 analyze table . . . compute statistics 或 analyze table . . . estimate statistics 命令，但对于与统计无关的任务，例如 validate structure 或 list chained rows 或者其他任务，则可以使用 analyze 命令来收集空闲列表块的有关信息。

分析数据可能需要大量排序空间。因为分析可能也包括完整的表扫描，所以应该在开始分析之前立刻改变会话设置。当分析完成时，应结束会话或将设置改回到数据库的值。要改变会话设置可以通过 SORT_AREA_SIZE 和 DB_FILE_MULTIBLOCK_READ_COUNT。从 Oracle Database 10g 开始，Oracle 强烈建议使用 PGA_AGGREGATE_TARGET 自动管理 SORT_AREA_SIZE 的值。排序区域的大小越大，需要将临时表空间用于排序段的可能性就越小。多块读计数越大，在一次物理读期间能够读取的块就越多(受到操作系统的限制)。使用 alter session 命令，为会话增加这些值。

8.4 调整数据访问

即使适当地配置并索引表，但如果存在由文件访问造成的等待事件，则依然会影响性能。在下面的几小节中，将介绍文件和表空间配置的一些相关建议。

一般来说，应该避免将 Oracle 文件放置在分布式奇偶校验的 RAID 系统(如 RAID 5)上。在写入到这种文件系统期间所产生的系统开销会随着系统使用的增加而导致性能瓶颈，特别是对于按顺序写入的文件，例如联机重做日志文件。赞成使用 RAID 0+1，它可以支持数据的镜像和条带化，而不会引入这些性能瓶颈。

8.4.1 本地管理的表空间

可以使用本地管理的表空间来处理表空间中的盘区管理。通过维护空闲块和已使用块的每个数据文件中的位图或者数据文件中的多组块，本地管理的表空间可以管理自己的空间。每次分配盘区或释放盘区以重用时，数据库更新位图以显示新的状态。

注意：
从 Oracle Database 10g 开始，默认安装的所有表空间都是本地管理，大文件表空间必须本地管理。使用字典管理的表空间只是为了与以前的 Oracle 版本兼容。

使用本地管理的表空间时，没有更新数据字典，并且在盘区创建期间没有生成回滚活动。本地管理的表空间自动跟踪邻近的空闲空间，从而不需要合并盘区。在本地管理的表空间中，所有的盘区可以有相同的大小，或者系统可以自动确定盘区的大小。

为了使用本地空间管理，可以为 create tablespace 命令中的 extent management 子句指定 local 选项。下面显示了声明本地管理表空间的 create tablespace 命令示例：

```
create tablespace CODES_TABLES
datafile '/u01/oracle/VLDB/codes_tables.dbf'
size 500M
extent management local uniform size 256K;
```

假设在其中创建表空间的数据库的块大小为 8KB，在该示例中，创建表空间，将盘区管理声明为 local，并且使用统一的大小 256KB。位图中的每一位描述了 32 个块(256/8)。如果省略 uniform size 子句，默认是 autoallocate。默认的 uniform size 大小是 1MB。

注意：
如果在 create tablespace 命令中指定 local，则不可以指定 default storage 子句、minextents 或 temporary。如果使用 create temporary tablespace 命令来创建表空间，则可以指定 extent_management local。

从 Oracle9i 开始，表空间默认创建为本地管理，因此创建新的表空间时，extent management local 子句是可选的。

注意：
如果将 SYSTEM 表空间设置为本地管理，则只可以在数据库中创建本地管理的表空间。使用可移植表空间特性导入的任何字典管理表空间只能以只读方式打开。

8.4.2 标识链行

创建数据段时，可以指定 pctfree 值。pctfree 参数可告诉数据库每个数据块中应该保留多少空闲的空间。通过 update 操作扩展存储在数据块中的行的长度时，就需要使用空闲空间。

如果对行的 update 操作造成行不再可以完全容纳在一个数据块中，该行就可能移动到另一个数据块，或者该行可能链接到另一个数据块。如果所存储行的长度大于 Oracle 的块大小，则自动具有链接。

链接会影响性能，因为它需要 Oracle 查看多个物理位置，查找来自于相同逻辑行的数据。通过消除不必要的链接，可以减少从数据文件中返回数据所需的物理读数量。

通过在数据段创建期间设置适当的 pctfree 值，可以避免链接。如果应用程序频繁将 NULL 值更新为非 NULL 值，或者频繁更新长文本值，则需要增加默认值 10。

可以使用 analyze 命令收集有关数据库对象的统计信息。基于成本的优化器可以使用这些统计信息来确定使用的最佳执行路径。analyze 命令具有检测并记录表中链行的选项。它的语法如下：

```
analyze table TABLE_NAME list chained rows into CHAINED_ROWS
```

analyze 命令将这个操作中的输出放置在本地模式中称为 CHAINED_ROWS 的表中。创建 CHAINED_ROWS 表的 SQL 位于名为 utlchain.sql 的文件中，该文件在$ORACLE_ HOME/rdbms/admin 目录中。下面的查询将从 CHAINED_ROWS 表中选择最重要的列：

```
select
       Owner_Name,       /*Owner of the data segment*/
       Table_Name,       /*Name of the table with the chained rows*/
       Cluster_Name,     /*Name of the cluster, if it is clustered*/
       Head_RowID        /*Rowid of the first part of the row*/
from CHAINED_ROWS;
```

输出将显示所有链行的 RowID，从而允许快速查看链接了表中的多少行。如果链接在表中非常普遍，则应该使用较大的 pctfree 值重新构建该表。

通过查看 V$SYSSTAT，可以看到行链接带来的影响。每次 Oracle 从链行中选择数据时，“table fetch continued row”统计信息的 V$SYSSTAT 条目将增加。当 Oracle 从跨越行中选择数据时，该统计信息也会增加，“跨越行”指的是由于行长度大于一个块的长度而被链接的行。具有 LONG、BLOB、CLOB 和 NCLOB 数据类型的表很可能具有跨越行。“table fetch continued row”统计信息也可用于 AWR 报告中，或者在 Oracle Database 10g 及较早版本中可用于 STATSPACK 报告中。

除了链接行以外，Oracle 有时也会移动行。如果行超出它的块的可用空间，该行就可能插入到一个不同的块中。将行从一个块移动到另一个块的过程称为“行迁移”，被移动的行称为“迁移行”。在行迁移

期间，Oracle 必须动态管理多个块中的空间，并且访问空闲列表(可用于 insert 操作的块列表)。迁移行不会表现为链行，但它会影响事务的性能。使用 DBMS_ADVISOR 查找并重新组织具有链行的表的示例，请参见第 6 章。

提示：
访问迁移行会增加“table fetch continued row”统计信息的数量。

8.4.3 增加 Oracle 块大小

增加数据库块大小的效果非常显著。将数据库块大小增加 1 倍可以将查询密集操作的性能改进多达 50%。

这种性能优点只需很少的成本。因为如果每个数据库块具有更多的行，在数据操作命令期间就更可能产生块级别的争用。为了解决争用问题，在表级别和索引级别上增加 freelists 和 initrans 的设置。一般来说，设置 freelists 大于 4 不会产生更多额外的优点。initrans 设置应该反映块中期望的并发事务的数量。

对于 DML 活动较多的 OLTP 应用程序，INITRANS 为 4 是合适的。对数据仓库应用程序增加 INITRANS 的值并不会提高性能。另外需要注意的是，空闲列表只用于非 ASSM 表空间中的对象。

注意：
在任何数据块中，Oracle 现在自动允许最多达 255 个并发更新事务，具体取决于块中可用的空间。

创建表空间时，可以为表空间指定数据库块大小。默认情况下，表空间将使用通过 DB_BLOCK_SIZE 初始参数指定的数据库块大小。如果为表空间使用非默认的数据库块大小，则需要为该块大小创建缓存。例如，如果数据库块大小是 8KB，并且希望创建 4KB 数据库块大小的表空间，则必须首先为 DB_4K_CACHE_SIZE 设置一个值。

为了增加整个数据库的数据库块大小，必须重新构建整个数据库，并且删除所有旧的数据库文件。可以在和旧文件相同的位置中创建新文件，并且具有相同大小，但数据库可以更有效地管理新的文件。挽救的性能来自于 Oracle 管理块标题信息的方法。数据使用更多的空间，从而改进多个用户访问内存中相同数据块的能力。将 Oracle 块的大小增加 1 倍对块标题没有什么影响，因此使用较少比例的空间来存储块标题信息。

为了设置块大小，可以在创建新数据库之前修改 DB_BLOCK_SIZE 初始参数。

8.4.4 使用索引组织表

索引组织表(Index-Organized Table，IOT)是在其中存储整个行的索引，而不仅仅是存储行的键值。行的主键作为行的逻辑标识符而不是存储行的 RowID。IOT 中的行没有 RowID。

在 IOT 中，通过它们的主键值按顺序存储行。因此，任何基于主键的范围查询都会受益，因为行是按顺序存储的(查看本章第 8.2 节，了解排序普通表中数据的相关步骤)。此外，任何基于主键的相等查询也会受益，因为表的数据全部存储在索引中。在传统的表/索引组合中，基于索引的访问需要在索引访问后面跟上表访问。在 IOT 中，只需要访问 IOT，没有任何伴随的索引。

然而，通过使用单个索引访问替代普通的索引/表组合访问获得的性能改进可能非常小，因为任何基于索引的访问都是非常快速的。为了帮助进一步改进性能，索引组织表提供了额外的特性：

- 溢出区域　通过在创建 IOT 时设置 pctthreshold 参数，可以分开存储主键数据和行数据。如果行数据超出块中可用空间的限度，它将动态移动到溢出区域。可以指定溢出区域位于单独的表空间中，从而改进分布与表相关的 I/O 的能力。
- 二级索引　可以在 IOT 上创建二级索引。Oracle 将使用主键值作为行的逻辑 RowID。
- 减少的存储需求　在传统的表/索引组合中，相同的键值存储在两个位置中。在 IOT 中，它们只存储一次，从而减少存储需求。

提示：
当指定溢出区域时，可以使用 including column 子句来指定将存储在溢出区域中的列(以及表定义中的所有后继列)：

```
create table ord_iot
   (order_id number,
    order_date date,
    order_notes varchar2(1000), primary key(order_id,order_date))
     organization index including order_date
     overflow tablespace over_ord_tab
     PARTITION BY RANGE (order_date)
      (PARTITION p1 VALUES LESS THAN ('01-JAN-2005')
       TABLESPACE data01,
       PARTITION p2 VALUES LESS THAN (MAXVALUE)
      TABLESPACE data02);
```

order_date 和 order_notes 都将存储在溢出区域中。

为了创建 IOT，可以使用 create table 命令的 organization index 子句。在创建 IOT 时必须指定主键。在 IOT 中，可以删除列，或者通过 alter table 命令的 set unused 子句将列标记为不　　活动。

8.4.5　索引组织表的调整问题

类似于索引，在插入、更新和删除值时，IOT 可能会逐渐在内部产生存储碎片。为了重新构建 IOT，可以使用 alter table 命令的 move 子句。在下面的示例中，重新构建了 EMPLOYEE_IOT 表及其溢出区域：

```
alter table EMPLOYEE_IOT
 move tablespace DATA
overflow tablespace DATA_OVERFLOW;
```

应该避免在 IOT 中存储长行的数据。一般来说，如果数据长度超过数据库块大小的 75%，则应该避免使用 IOT。如果数据库块大小是 4KB，并且行的长度超出 3KB，则应该考虑使用普通表和索引，而不是使用 IOT。行越长，针对 IOT 执行的事务越多，就越需要频繁地重新构建 IOT。

注意：

在 IOT 中不可以使用 LONG 数据类型，但可以使用 LOB。

本章前面提及，索引影响数据加载速率。为了获得最佳的结果，索引组织表的主键索引应该和连续的值一起加载，从而最小化索引管理的成本。

8.5 调整数据操作

一些数据操作任务——通常是涉及大量数据的操作，可能需要 DBA 的参与。在加载和删除大量数据时，可以有一些选择，下面各小节将描述这些选择。

8.5.1 大量插入：使用 SQL*Loader Direct Path 选项

用于 Conventional Path 模式中时，SQL*Loader 从文件中读取记录，生成 insert 命令，并且将它们传递到 Oracle 内核。然后，Oracle 在表中的空闲块中为这些记录查找存放空间，并且更新任何相关的索引。

在 Direct Path 模式中，SQL*Loader 创建格式化的数据块，并且直接写入到数据文件。这需要偶尔检查数据库以获得数据块的新位置，但不需要其他使用数据库内核的 I/O。结果是数据加载过程远快于 Conventional Path 模式。

如果表上建立了索引，则索引将在加载期间处于 DIRECT PATH 状态。加载完成后，排序新的键(索引列值)，并且将其与索引中已有的键合并在一起。为了维护临时的一组键，加载过程将创建一个临时的索引段,该段至少与表上最大的索引一样大。通过预先排序数据并在 SQL*Loader 控制文件中使用 SORTED INDEXES 子句，可以最小化这种情况的空间需求。

为了最小化在加载期间必需的动态空间分配数量，正在加载进来的数据段应该是已经创建的数据段，并且已经分配了所需要的全部空间。也应该预先排序表中最大索引的列上的数据。相比于在加载之前删除索引，然后在加载完成后重新创建这些索引，在 Direct Path 加载期间排序数据并保留表上的索引通常会产生更好的性能。

为了利用 Direct Path 选项，不可以群集表，也不可以有针对它的其他活动事务。在加载期间，只实施 NOT NULL、UNIQUE 和 PRIMARY KEY 约束；在加载完成后，可以自动重新启用 CHECK 和 FOREIGN KEY 约束。为了强制进行这种重新启用过程，在 SQL*Loader 控制文件中使用如下子句：

```
REENABLE DISABLED_CONSTRAINTS
```

这种重新启用过程的唯一例外是表插入触发器，在重新启用时，不会为表中的每个新行执行表插入触发器。无论这种类型的触发器执行了什么命令，单独的过程必须手动执行这些命令。

在将数据加载到 Oracle 表中时，SQL*Loader Direct Path 加载选项比 SQL*Loader Conventional Path 加载器提供了更多的性能改进，其方法是绕开 SQL 处理、缓冲区缓存管理以及不必要的数据块读取。SQL*Loader 的 Parallel Data Loading 选项允许使用多个进程将数据加载到相同的表中，利用系统上多余的资源，从而减少加载过程消耗的总时间。假设有足够的 CPU 和 I/O 资源，这样做就可以极大地减少总的加载时间。

为了使用 Parallel Data Loading，使用 parallel 关键字启动多个 SQL*Loader 会话(否则，SQL*Loader 会在表上放置一个独占的锁)。每个会话都是一个独立的会话，需要有自己的控制文件。下面的程序清单显示了 3 个单独的 Direct Path 加载，它们都在命令行中使用了 PARALLEL=TRUE 参数：

```
sqlload USERID=ME/PASS CONTROL=PART1.CTL DIRECT=TRUE PARALLEL=TRUE
sqlload USERID=ME/PASS CONTROL=PART2.CTL DIRECT=TRUE PARALLEL=TRUE
sqlload USERID=ME/PASS CONTROL=PART3.CTL DIRECT=TRUE PARALLEL=TRUE
```

每个会话默认情况下创建自己的日志文件、错误文件和丢弃文件(part1.log、part2.log、part3.log、part1.bad、part2.bad 等)。因为具有多个将数据加载到相同表中的会话，所以对于 Parallel Data Loading，只允许 APPEND 选项。而对于 Parallel Data Loading，不允许 SQL*Loader REPLACE、TRUNCATE 和 INSERT 选项。如果在启动加载前需要删除表中的数据，则必须手动删除数据(通过 delete 或 truncate 命令)。如果正在使用 Parallel Data Loading，则不可以使用 SQL*Loader 自动删除记录。

注意：

如果使用 Parallel Data Loading，SQL*Loader 会话不会维护索引。在启动加载进程之前，必须删除表上的所有索引，并且禁用它的所有 PRIMARY KEY 和 UNIQUE 约束。在加载完成后，可以重新创建表的索引。

在串行的 Direct Path Loading(PARALLEL=FALSE)中，SQL*Loader 将数据加载到表中的盘区中。如果加载过程在加载完成之前失败，一些数据可能会在过程失败之前提交给表。在 Parallel Data Loading 中，每个加载过程为加载数据创建临时段。临时段在以后与表合并在一起。如果 Parallel Data Loading 过程在加载完成之前失败，临时段就不会与表合并。如果临时段没有与加载的表合并，该加载过程中的任何数据都不会提交到表中。

可以使用 SQL*Loader FILE 参数将每个数据加载会话定向到不同的数据文件。通过将每个数据加载会话定向到它自己的数据文件，可以平衡加载过程的 I/O 负载。数据加载是 I/O 操作非常密集的一个过程，必须分布到多个磁盘上实施并行加载。与串行加载相比，并行加载的性能得到了很大的改善。

Parallel Data Load 后，每个会话可能尝试重新启用表的约束。只要至少有一个加载会话仍然在进行中，则尝试重新启用约束就会失败。最后一个完成的加载会话应该尝试重新启用约束，并且应该会成功。应该在加载完成后检查约束的状态。如果加载的表具有 PRIMARY KEY 和 UNIQUE 约束，可以在启用约束前并行创建相关的索引。

8.5.2 大量数据移动：使用外部表

通过称为外部表的对象，可以查询数据库外部的文件中的数据。通过 create table 命令的 organization external 子句定义外部表的结构，其语法与 SQL*Loader 控制文件的语法非常相似。

不可以操作外部表中的行，并且不可以索引它：每次对该表的访问都会产生完整的表扫描(即在操作

系统级上完整扫描文件)。结果，相比于对存储在数据库中的表的查询性能，对外部表的查询往往具有较差的性能。然而，外部表为加载大型数据集的系统提供了一些潜在的优点：

- 因为数据不是存储在数据库中，并且数据只存储一次(在数据库外部，而不是数据库外部和内部都存储)，从而节省了空间。
- 因为数据绝对不会加载到数据库中，从而消除了数据加载时间。

由于不可以索引外部表，所以它们对由批处理程序访问大量数据的操作最为有用。例如，许多数据仓储环境具有分步操作，其中，在将行插入到用户查询的表中之前，将数据加载到临时表中。也可以不将数据加载到这些临时表，而是直接通过外部表访问操作系统文件，从而节省时间和空间。

从体系结构的角度来看，外部表允许将数据库内容的重点放在用户最常使用的对象上：小型代码表、聚集表以及事务表，同时在数据库外部保持非常大的数据集。可以在任何时刻替换由外部表访问的文件，而不会导致数据库中的任何事务开销。

8.5.3 大量插入：常见的陷阱和成功的技巧

如果没有从平面文件中插入数据，SQL*Loader 将不会是有用的解决方案。例如，如果需要将大型的数据集从一个表移动到另一个表，则很可能希望避免必须将数据写入到平面文件，然后再将其读回到数据库中。在数据库中移动数据的最快方法是将其从一个表移动到另一个表，而不需要进入到操作系统。

将数据从一个表移动到另一个表时，有一些常见的方法可用于改进数据迁移的性能：

- 调整结构(删除索引和触发器)。
- 在数据迁移期间禁用约束。
- 使用提示和选项来改进事务性能。

第一个技巧是调整结构，它涉及禁用要加载数据的表上的任何触发器或索引。例如，如果在目标表上有行级别的触发器，则为插入到表中的每一行执行该触发器。如果可能的话，在数据加载之前禁用该触发器。如果应该为每个插入的行执行该触发器，则可以在插入行以后，再执行批量操作，而不是每次插入期间执行重复的操作。如果进行适当的调整，批量操作将比重复执行触发器更快地完成。需要确保为触发器还没有处理的所有行执行这些批量操作。

除了禁用触发器，应该在启动数据加载之前禁用目标表上的索引。如果索引留在表上，Oracle 将在插入每一行时动态管理索引。不是持续地管理索引，而是在启动加载之前删除它，并且在加载完成时重新创建它。

注意：

禁用索引和触发器可以解决大多数和大型表与表之间数据迁移工作关联的性能问题。

除了禁用索引，还应该考虑禁用表上的约束。如果源数据已经在数据库的表中，在将其加载到目标表中之前，可以检查该数据，了解它的相关约束(如外键约束或 CHECK 约束)。一旦已经加载数据，就可以重

新启用这些约束。

如果这些选项都无法提供适当的性能，就应该调查 Oracle 为数据迁移调整而引入的选项。这些选项包括如下：

- 插入命令的 APPEND 提示　类似于 Direct Path Loader，APPEND 提示将数据块加载到表中，从表的高水位线开始。使用 APPEND 提示可以增加空间利用率。
- nologging 选项　如果正在执行 create table as select 命令，使用 nologging 选项可避免在操作期间写入到重做日志。
- 并行选项　并行查询使用多个进程来完成一个任务。对于 create table as select 命令，可以并行化 create table 部分和查询部分。如果使用并行选项，则也应该使用 nologging 选项；否则，并行操作将不得不由于串行化写入到联机重做日志文件而等待。

在使用这些高级选项之前，首先应该调查目标表的结构，确保已经避免了本章前面列举的一些常见陷阱。

也可以使用编程逻辑强制以阵列的方式处理插入，而不是作为整体的集合。例如，COBOL 和 C 支持阵列的插入，从而减少处理大型数据集所需的事务大小。

8.5.4　大量删除：truncate 命令

有时，用户会尝试一次性删除表中的所有记录。当他们在这个过程期间遇到错误时，就会抱怨回滚段太小，而实际上是事务太大。

一旦已经删除所有记录，第二个问题就会发生。即使段中不再有任何记录，它仍然会维持分配给它的所有空间。因此，删除所有这些记录不会节省任何已分配的空间。

truncate 命令可解决这两个问题。这是一个 DDL 命令，而不是 DML 命令，因此不可以回滚该命令。一旦已经在表上使用 truncate 命令，则删除它的记录，并且在该过程中不会执行任何 delete 触发器。然而，表保留它的所有从属对象：例如授权、索引和约束。

truncate 命令是删除大量数据的最快方法。因为它将删除表中的所有记录，这可能会迫使您改变应用程序设计，从而不会有任何受保护的记录存储在和删除的记录相同的表中。如果使用分区，可以截取表的一个分区，而不会影响表的剩余分区(参见第 16 章)。

表的示例 truncate 命令如下所示：

```
truncate table EMPLOYEE drop storage;
```

前面的示例是删除 EMPLOYEE 表中的记录，它显示了 truncate 强大的功能。drop storage 子句用于释放表中非 initial 的空间(这是默认选项)。因此，可以删除表的所有行，并且回收所有空间，除了初始盘区的分配空间，而不会删除表。

truncate 命令也可以用于集群。在下面的示例中，reuse storage 选项用于在获得它的段中保留所有已分配的空间为空：

```
truncate cluster EMP_DEPT reuse storage;
```

执行该示例命令时，EMP_DEPT 集群中的所有记录将立刻删除。

为了截取分区，需要知道它的名称。在下面的示例中，通过 alter table 命令截取 EMPLOYEE 表中名为 PART3 的分区：

```
alter table EMPLOYEE
truncate partition PART3
drop storage;
```

EMPLOYEE 表中剩余的分区将不会受到截取 PART3 分区的影响。查看第 16 章以了解创建和管理分区的详细信息。

作为选择，可以创建 PL/SQL 程序，使用动态 SQL 将大型的 delete 操作划分为多个较小的事务。

8.5.5 使用分区

可以使用分区在物理上隔离数据。例如，可以在 ORDERS 表的单独分区中存储每个月的事务。如果在表上执行大量数据加载或删除，可以自定义分区以调整数据操纵的操作。例如：

- 可以截取分区和它的索引，而不会影响表的剩余部分。
- 可以通过 alter table 命令的 drop partition 子句删除分区。
- 可以删除分区的本地索引。
- 可以设置分区为 nologging，减少大型事务的影响。

从性能的角度来看，分区的主要优点在于可以和表的其他部分分开管理的能力。例如，截取分区的能力允许从表中删除大量数据(但不是表的所有数据)，而不会生成任何重做信息。就眼前来说，这种性能改进的受益者是 DBA；从长远的观点来看，整个企业都会从这种改进的数据可用性中受益。查看第 16 章以了解实现分区和子分区的详细信息。

可以使用 exchange partition 选项来极大地减小数据加载过程对系统可用性带来的影响。首先创建具有和分区表相同列结构的空表。将数据加载到新的表中，然后分析新的表。在新的表上创建和分区表相同的索引，这些索引必须是局部索引，而不能是全局索引。这些步骤完成时，使用 exchange partition 子句改变分区表，交换空的分区和填充的新表。现在通过分区表可以访问所有加载的数据。在这个步骤期间对系统的可用性几乎没有任何影响，因为它是一个 DDL 操作。

8.6 调整物理存储

数据库 I/O 应该均匀地分布在尽可能多的设备上。标准解决方案称为 SAME(是 stripe and mirror everything 的首字母缩写，代表条带化和镜像所有内容)。磁盘的 I/O 吞吐量限制是需要克服的关键限制，因此将 I/O 需求分布在许多磁盘上就允许利用许多设备的结合吞吐量。条带化可以增强吞吐量，从而可以改善性能。镜像提供在磁盘故障情况下的支持。

除了这种级别的物理存储调整外，也应该考虑其他一些因素。下面的小节将讨论数据库外部的一些因素，这些因素可能对快速访问数据能力有深远的影响。

8.6.1 使用裸设备

裸设备可以和大多数 Unix 操作系统一起使用。使用裸设备时，Oracle 绕开 Unix 缓冲区缓存，并且消除文件系统的系统开销。对于 I/O 密集型应用程序，使用裸设备可能比传统的文件系统改进大约 20%的性能(比自动存储管理的性能稍有改进)。最近的文件系统增强已经在很大程度上克服了这种性能区别。

不可以使用与文件系统相同的命令管理裸设备。例如，tar 命令不可以用于备份单个的文件；相反，必须使用 dd 命令。使用该命令具有较小的灵活性，并且限制了恢复功能。

注意：

Oracle 文件不应该和非 Oracle 文件驻留在相同的物理设备上,特别是在使用裸设备时。将活动的 Unix 文件系统和活动的 Oracle 裸设备混合在一起将造成 I/O 性能问题。

大多数支持裸设备的操作系统也提供了逻辑卷管理层，允许管理员执行裸设备的文件系统命令。这种方法允许将文件系统管理的优点和裸设备的性能优点结合在一起。如果计划使用裸设备，应该使用逻辑卷管理工具来简化系统管理。

8.6.2 使用自动存储管理

从 Oracle 10g 开始，可以使用自动存储管理(ASM)来管理数据库存储区域。第 4 章详细分析了 ASM 如何能够提供裸设备的大多数性能好处，同时可以轻松使用传统操作系统的文件系统，另外还给出了很多示例。

在 ASM 环境中，当创建新的表空间或其他数据库结构(如控制文件或重做日志文件)时，可以指定磁盘组，而不是指定一个操作系统文件，作为数据库结构的存储区域。ASM 简化了 Oracle 管理文件(OMF)的使用，并将 OMF 与镜像和条带化特性结合起来，从而提供了健壮的文件系统和逻辑卷管理程序，这种管理程序甚至支持 Oracle 实时应用集群(RAC)中的多个节点。ASM 不再需要购买第三方逻辑卷管理程序。

ASM 不仅能够自动扩展数据库对象到多个设备以增强性能，而且允许在不关闭数据库的情况下将新的磁盘设备添加到数据库，从而增加可用性。ASM 自动重新平衡文件的分布，使其具有最低限度的干涉。

8.7 减少网络流量

当数据库和使用它们的应用程序变成更为分布的情况时，支持服务器的网络可能成为交付数据给用户过程中的瓶颈。因为 DBA 一般无法对网络管理进行更多的控制，所以使用数据库的功能来减少要交付的数据所需的网络包的数量就显得很重要。减少网络流量将减少对网络的依赖性，从而消除潜在造成的性能问题。

8.7.1 使用物化视图复制数据

可以操作和查询远程数据库中的数据。然而，不希望经常将大量数据从一个数据库发送到另一个数据

库。为了减少在网络上发送的数据量，应该考虑不同的数据复制选项。

在完全分布式的环境中，每个数据元素存储于一个数据库中。需要数据时，通过数据库链接从远程数据库中访问该数据。这种纯粹的方法类似于严格按照第三范式实现应用程序：一种无法轻松支持任何主要产品应用程序的方法。修改应用程序的表以改进数据检索性能会涉及数据的反规格化。反规格化过程特意存储多余的数据，从而缩短用户对数据的访问路径。

在分布式环境中，复制数据可实现这一目标。不是迫使查询通过网络来解决用户请求，而是将远程服务器中选择的数据复制到本地服务器中。可以通过许多方法实现这一点，下面的小节将描述这些方法。

复制的数据在创建后就会立刻过时。因此，在源数据不是频繁改变或业务过程可以支持旧数据的使用时，针对性能目标而复制数据最为有效。

Oracle 的分布式功能提供了管理数据库中数据复制的方法。物化视图将数据从主要的来源复制到多个目标。Oracle 提供了一些工具，用于以指定的时间间隔刷新数据并更新目标。

物化视图可以是只读的，也可以是可更新的。第 17 章中将介绍物化视图的管理问题，本节将介绍它们的性能调整。

在为复制而创建物化视图之前，应该首先创建到源数据库的数据库链接。下面的示例使用 LOC 服务名创建了名为 HR_LINK 的私有数据库链接：

```
create database link HR_LINK
connect to HR identified by ESNOTHR1968
using 'loc';
```

该示例中所示的 create database link 命令具有如下一些参数：

- 链接的名称(当前情况下是 HR_LINK)。
- 连接到的账户。
- 远程数据库的服务名(在服务器的 tnsnames.ora 文件中可以找到该名称)。在当前情况下，服务名是 LOC。

物化视图自动完成数据复制和刷新过程。创建物化视图时，建立刷新间隔时间以调度被复制数据的刷新。物化视图可以防止本地的更新，但可以使用基于事务的刷新。基于事务的刷新只从主数据库中发送针对物化视图改变的行，可用于许多类型的物化视图。本章后面将描述该功能，它可以极大地改进刷新的性能。

下面的示例展示了用于在本地服务器上创建物化视图的语法，其中为物化视图提供了名称(LOCAL_EMP)，并且指定了它的存储参数，给出了它的基本查询和刷新间隔时间。在此例中，告知物化视图

立刻检索主要的数据，然后在 7 天之后再次执行刷新操作(SYSDATE+7)：

```
create materialized view LOCAL_EMP
pctfree 5
tablespace data_2
storage (initial 100K next 100K pctincrease 0)
refresh fast
      start with SysDate
      next SysDate+7
as select * from EMPLOYEE@HR_LINK;
```

refresh fast 子句告诉数据库使用物化视图日志来刷新本地的物化视图。如果物化视图的基本查询足够简单，Oracle 可以利用它来确定当源表中的一行改变时物化视图中的哪一行会改变，这时才可以使用在刷新期间使用物化视图日志的能力。

使用物化视图日志时，只有对主表的改动才会发送到目标。如果使用复杂的物化视图，必须使用 refresh complete 子句来代替 refresh fast 子句。在完整的刷新中，刷新完全替换物化视图的底层表中的已有数据。

必须通过 create materialized view log 命令在主数据库中创建物化视图日志。create materialized view log 命令的示例如下所示：

```
create materialized view log on EMPLOYEE
tablespace DATA
storage (initial 500k next 100k pctincrease 0);
```

物化视图日志总是创建在与主表相同的模式中。

可以使用具有物化视图日志的简单物化视图来减少维护被复制数据所涉及的网络流量数。因为只会通过物化视图日志发送对数据的改动，所以简单物化视图的维护应该比复杂物化视图使用较少的网络资源，特别是在主表是大型的、相当静态的表时。如果主表不是静态的，则通过物化视图日志发送的事务量可能不会少于执行完整刷新所发送的事务量。查看第 17 章以了解关于物化视图的刷新功能的详细信息。

不论选择什么刷新选项，都应该索引物化视图的基表，优化针对物化视图的查询。从性能的角度来看，最终目标是尽可能快速地为用户提供他们所需的数据，并且使用用户所需的格式。通过创建关于远程数据的物化视图，可以避免在查询期间遍历数据库链接。通过创建关于本地数据的物化视图，可以防止用户重复聚集大量数据，代之以向用户提供预先聚集好的数据，这些数据可以回答大多数常见的查询。

8.7.2 使用远程过程调用

在分布式数据库环境中使用过程时，可以使用两种选项中的一种：创建引用远程表的本地过程，或者创建由本地应用程序调用的远程过程。

过程的合适位置取决于数据的分布情况以及数据的使用方式。其重点应该放在最小化为了解决数据请求而必须通过网络发送的数据量上。驻留该过程的数据库应包含此过程操作期间使用的大多数数据。

例如，考虑如下过程：

```
create procedure MY_RAISE (My_Emp_No IN NUMBER, Raise IN NUMBER)
as begin
      update EMPLOYEE@HR_LINK
      set Salary = Salary+Raise
      where Empno = My_Emp_No;
end;
```

在这种情况下，过程只访问远程节点(由数据库链接 HR_LINK 表明)上的一个表(EMPLOYEE)。为了减少通过网络发送的数据量，将该过程移动到由数据库链接 HR_LINK 标识的远程数据库，并且从该过程的 from 子句中删除对数据库链接的引用。然后，通过使用数据库链接，从本地数据库中调用该过程，如下所示：

```
execute MY_RAISE@HR_LINK(1234,2000);
```

在这种情况下，两个参数传递给过程：My_Emp_No 设置为 1234，Raise 设置为 2000。使用数据库链接调用该过程，告诉数据库在何处查找过程。

执行远程过程调用的调整优点是，在驻留数据的数据库中执行过程的所有处理。远程过程调用可最小化完成过程处理所必需的网络流量。

为了维持位置的透明性，可以创建一个指向远程过程的本地同义词。在同义词中指定数据库链接名，从而用户请求将自动使用远程数据库：

```
create synonym MY_RAISE for MY_RAISE@HR_LINK;
```

然后，用户可以输入如下命令：

```
execute MY_RAISE(1234,2000);
```

这将执行由同义词 MY_RAISE 定义的远程过程。

8.8 使用自动工作负荷存储库(AWR)

在 Oracle Database 10g 及较早的版本中，STATSPACK 收集并报告数据库统计信息，虽然这些统计信息严格按照基于文本的格式！从 Oracle 10g 开始，自动工作负荷存储库(Automatic Workload Repository，AWR)对 STATSPACK 概念进行了增强，除生成 STATSPACK 收集的所有统计信息外，AWR 还生成更多的信息。此外，AWR 与 OEM 高度集成，从而很容易分析和修正性能问题。

类似于 STATSPACK，AWR 收集和维护性能统计信息，以便发现问题并进行自调整。您可以生成关于 AWR 数据的报表，并且可以通过视图或通过 OEM 来访问它。您可以报告最近的会话活动以及整体的系统统计信息和 SQL 的使用。

AWR 每小时捕获一次系统统计信息(生成数据库的“快照”)，并且将数据存储在它的存储库表中。和 STATSPACK 一样，当历史保留周期增加或快照之间的间隔时间减少时，AWR 存储库的空间需求就会增加。默认情况下，在存储库中维持 7 天的数据量。可以通过 DBA_HIST_SNAPSHOT 视图查看存储在 AWR 存储库中的快照。

为了启用 AWR,可以将 STATISTICS_LEVEL 初始参数设置为 TYPICAL 或 ALL。如果设置 STATISTICS_LEVEL 为 BASIC，则可以生成 AWR 数据的手动快照，但这些快照不像由 AWR 自动执行的快照那样全面。将 STATISTICS_LEVEL 设置为 ALL 可以将定时的 OS 统计信息和计划执行统计信息添加到用 TYPICAL 设置收集的那些信息中。

8.8.1 管理快照

为了生成手动的快照，可以使用 DBMS_WORKLOAD_REPOSITORY 程序包的 CREATE_ SNAPSHOT 过程：

```
execute DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT ();
```

为了改变快照设置，可以使用 MODIFY_SNAPSHOT_SETTINGS 过程。可以修改快照的保留时间(以分钟为

单位)和间隔时间(以分钟为单位)。下面的示例将当前数据库的快照间隔时间改为 30 分钟：

```
execute  DBMS_WORKLOAD_REPOSITORY.MODIFY_SNAPSHOT_SETTINGS
( interval => 30);
```

为了删除一定范围的快照，可以使用 DROP_SNAPSHOT_RANGE 过程，同时指定要删除的开始快照 ID 和结束快照 ID：

```
execute DBMS_WORKLOAD_REPOSITORY.DROP_SNAPSHOT_RANGE
     (low_snap_id => 1, high_snap_id => 10);
```

8.8.2 管理基线

可以指定一组快照作为系统性能的基线。这些基线数据将被保留，便于以后与快照进行比较。使用 CREATE_BASELINE 过程来指定基线的开始快照和结束快照：

```
execute DBMS_WORKLOAD_REPOSITORY.CREATE_BASELINE
(start_snap_id => 1, end_snap_id => 10,
baseline_name => 'Monday baseline');
```

创建基线时，Oracle 将一个 ID 赋予基线，可以通过 DBA_HIST_BASELINE 视图查看过去的基线。作为基线开始和结束的快照将一直保留，直到删除基线。为了删除基线，可以使用 DROP_BASELINE 过程：

```
execute DBMS_WORKLOAD_REPOSITORY.DROP_BASELINE
(baseline_name => 'Monday baseline', cascade => FALSE);
```

如果设置 DROP_BASELINE 过程的 CASCADE 参数为 TRUE，则在删除基线时将删除相关的快照。

可以通过 OEM 或通过本节前面提到的 DBA_HIST_SNAPSHOT 数据字典视图来查看 AWR 数据。支持 AWR 的额外视图包括 V$ACTIVE_SESSION_HISTORY(每秒采样一次)、DBA_HIST_SQL_PLAN(执行计划)以及 DBA_HIST_WR_CONTROL(用于 AWR 设置)。

8.8.3 生成 AWR 报表

可以通过 OEM 或通过提供的报告脚本来从 AWR 中生成报表。awrrpt.sql 脚本根据开始快照和结束快照

之间统计方面的差别来生成报表。第二个报告脚本 awrrpti.sql 根据指定数据库和实例的开始快照和结束快照来显示报表。

awrrpt.sql 和 awrrpti.sql 都位于$ORACLE_HOME/rdbms/admin 目录中。执行报表(通过任何 DBA 账户)时，会提示输入报表的类型(HTML 或文本)、列出的快照的天数、开始和结束快照 ID 以及输出文件的名称。

8.8.4 运行 Automatic Database Diagnostic Monitor 报表

除了依赖于针对 AWR 表(在以前的 Oracle 版本中与 STATSPACK 非常相似)的手动报告，还可以使用 Automatic Database Diagnostic Monitor (ADDM)。因为是基于 AWR 数据，所以 ADDM 需要设置 STATISTICS_LEVEL 参数(根据前文的建议，该参数设置为 TYPICAL 或 ALL)。可以通过 OEM 的 Performance Analysis 部分访问 ADDM，或者手动运行 ADDM 报表。

为了对一组快照运行 ADDM，可以使用位于$ORACLE_HOME/rdbms/admin 目录中的 addmrpt.sql 脚本。

注意：

必须具有 ADVISOR 系统权限才能执行 ADDM 报表。

在 SQL*Plus 中，执行 addmrpt.sql 脚本。这时会提示输入进行分析的开始和结束快照 ID，以及输出文件的名称。

为了查看 ADDM 数据，可以使用 OEM 或顾问数据字典视图。顾问视图包括 DBA_ ADVISOR_TASKS(已有的任务)、DBA_ADVISOR_LOG(任务的状态和进展)、DBA_ADVISOR_ RECOMMENDATIONS(完成的诊断任务和推荐)以及 DBA_ADVISOR_FINDINGS。可以使用推荐的方法来解决通过 ADDM 发现的问题。图 8-1 展示了从默认基线生成的一个典型 AWR 报告。在此例中，快照开始于 14-Sep-2007，结束于 22-Sep-2007。加载此数据库似乎需要大量的 CPU 和内存资源，但是，栓锁争用(latch contention)不存在，所以有足够的内存执行所有排序，而不必使用磁盘。

图 8-1　通过 OEM 访问 AWR 报告示例

8.8.5　使用自动 SQL 调整顾问

Oracle Database 11g 的新增特性自动 SQL 调整顾问(Automatic SQL Tuning Advisor)运行于默认维护窗口期间(使用 AutoTask),并以 AWR 中收集的负载最高的 SQL 语句为目标。一旦在维护窗口期间开始自动 SQL 调整，自动 SQL 调整顾问就执行下列步骤：

(1) 从 AWR 统计信息识别重复的高负载 SQL。忽略最近调整的 SQL 和循环 SQL。

(2) 调用 SQL 调整顾问，调整高负载 SQL。

(3) 为高负载 SQL 创建 SQL 配置文件，并分别测试有配置文件和无配置文件的性能。

(4) 如果性能至少提高 1/3，则自动保留配置文件，否则，注意在调整报告中加以改进。

图 8-2 展示了 Advisor Central(顾问中心)的 Advisor tasks(顾问任务)的小结。在此例中，可以看到 Automatic Database Diagnostic Monitor (ADDM)、Segment Advisor 和 SQL Tuning Advisor 的结果小结。

图 8-2　OEM 顾问中心(Advisor Central)小结

单击 SQL Tuning Advisor 结果链接，可以看到 SQL 调整结果小结，如图 8-3 所示。在这个使用率较低的数据库中，SQL Tuning Advisor 找到 14 个重复的 SQL 语句属于高负载类别，但并未找到改进这些 SQL 语句性能的方法。

图 8-3　自动 SQL 调整顾问(Automatic SQL Tuning Advisor)结果

8.9　调整解决方案

本章没有介绍每个可能的调整解决方案。然而，本章介绍了各种技术和工具的底层方法。在花费时间和资源来实现新特性之前，应该首先稳定环境和体系结构：服务器、数据库和应用程序。如果环境是稳定的，则应该能够快速实现两个目标：

(1) 成功重现性能问题。

(2) 成功发现问题的起因。

为了完成这些目标，可能需要具有可用于性能测试的测试环境。一旦成功发现问题，则可以将本章中概述的步骤应用于问题。一般来说，调整方法应该遵循本章中各个小节的顺序：

(1) 评估应用程序设计。

(2) 调整 SQL。

(3) 调整内存利用率。

(4) 调整数据存储。

(5) 调整数据操作。

(6) 调整物理和逻辑存储。

(7) 调整网络流量。

根据应用程序的性质，既可以按不同的顺序执行这些步骤，也可以合并这些步骤。

如果不可以改变应用程序设计和 SQL，则可以调整应用程序使用的内存和磁盘区域。改变内存和磁盘区域设置时，必须再次访问应用程序设计和 SQL 实现，从而确保改变不会反过来影响应用程序。如果选择使用数据复制方法，则再次访问应用程序设计过程的需求就非常重要，因为被复制数据的时间线可能会对应用程序的业务处理造成影响。

为了保护公司中最至关重要的财产，即它的数据，DBA 必须深入了解 Oracle 如何保护公司数据以及他们可以使用的不同工具。Oracle 提供的工具和机制分为 3 类：验证、授权和审计。

验证包括用于标识谁正在访问数据库的方法，确认用户身份，而不考虑正在请求数据库的什么资源。即使仅仅尝试访问自助餐厅每日的午餐菜单，向数据库正确标识自己也非常重要。例如，如果基于 Web 的数据库应用程序能够根据用户账户给出定制内容，就应确保某个用户获得位于德克萨斯州休斯顿市的分部的午餐菜单，而不是位于纽约州布法罗市的总部的午餐菜单！

只要数据库对用户进行了验证，授权就可以为用户提供对数据库中各种对象的访问。可以授权一些用户运行针对每日销售表的报表；有些用户可能是开发人员，因此需要创建表和报表，而有些用户可能只被允许查看每日的午餐菜单。一些用户可能从来不会登录，但他们的模式可能拥有特定应用程序的大量表，例如工资单和账户应收款项。还应为数据库管理员提供额外的授权方法，这是由于数据库管理员具有极大的权限。由于 DBA 可以关闭和启动数据库，因此也应为其提供额外的授权级别。

授权并不只是对表或报表的简单访问，它也包括使用数据库中系统资源的权利以及在数据库中执行某些操作的权限。某个数据库用户的每个会话可能只允许使用 15 秒的 CPU 时间，或者在与数据库断开连接之前只可以空闲 5 分钟。另一个数据库用户可能被授予在任何其他用户的模式中创建或删除表的权限，但不能够创建同义词或查看数据字典表。细粒度的访问控制为 DBA 提供了对如何访问数据库对象的更多控制。例如，标准对象权限将为用户提供对表中一个整行的访问，或者完全不可以访问；使用细粒度的访问控制，DBA 可以创建由存储过程实现的策略，该策略基于以下三种情况或者其中的某一种情况来限制访问：一天当中的不同时间、请求源自哪里、访问表的哪些列。

在关于数据库授权的结尾部分中，将介绍关于 Virtual Private Database(VPD)的简短示例，VPD 用于提供定义、设置和访问应用程序属性的方法。同时也介绍了一些谓词(通常是 WHERE 子句)，用于控制哪些数据是应用程序用户可访问的，或者是可以返回给应用程序用户的。

Oracle 数据库中的审计包含了数据库中大量不同级别的监控。在较高的级别中，审计可以记录成功的和不成功的登录尝试、访问对象或者执行操作。从 Oralce9i 开始，细粒度审计(FGA)不仅可以记录访问了什么对象，还可以记录在对列中的数据执行插入、更新或删除时访问表的哪些列。细粒度审计是审计用于标准授权的细粒度访问控制：有关访问对象和执行操作的更为精确的控制和信息。

DBA 必须经过深思熟虑再使用审计，以免无法理解审计记录，或者由于实现连续的审计而导致过多的系统开销。但是，通过监控谁正在使用什么资源、什么时候使用、多长时间使用一次，以及这次访问是否成功，审计可以帮助保护公司财产。因此，审计是 DBA 应该持续使用的另一种工具，用于监控数据库的安全状况。

9.1 非数据库的安全性

如果对操作系统的访问是不安全的，或者物理硬件处于不安全的位置，则本章后面介绍的所有方法学就没有任何作用。本节中将讨论数据库自身外部的一些元素，在认为数据库是安全的之前，需要确保这些元素是安全的。

下面的列表是一些需要在数据库外部考虑的内容：

- 操作系统安全性　除非 Oracle 数据库运行在自己的专用硬件上，并且只启用了 root 和 oracle 用户账户，否则就必须检查和实现操作系统安全性。确保使用 oracle 账户而不是 root 账户安装软件。也可以考虑使用另一个账户而不是 oracle 账户作为软件和数据库文件的拥有者，从而防止黑客对这个容易的目标进行攻击。确保只有 oracle 账户和 oracle 所属的组才可以读取软件和数据文件。除了需要 SUID 的 Oracle 可执行文件以外，在不需要 SUID 的文件上关闭 SUID(设置 UID，或者使用根权限运行)位。不要通过以纯文本编写的电子邮件发送密码(操作系统密码或 Oracle 密码)给用户。最后，删除支持数据库的服务器上不需要的任何系统服务，例如 telnet 和 ftp。
- 保护备份介质的安全　确保数据库备份介质，无论是磁带、磁盘或 CD/DVD-ROM，都只可以由有限数量的人访问。如果黑客可以获得数据库的备份副本并在另一个服务器上加载它们，则安全的操作系统和数据库上健壮的、加密的密码就没有任何价值。这同样也适用于任何包含从数据库中复制的数据的服务器。
- 后台安全性检查　审查处理敏感数据库数据的雇员是必须的工作，无论雇员是 DBA、审计者或操作系统管理员。
- 安全性教育　确保所有数据库用户都了解 IT 基础结构的安全性和使用策略。需要用户理解和遵循安全性策略，要着重强调数据对公司的关键特性和价值，包括数据库中的信息。受过良好教育的用户更有可能抵抗黑客通过社会工程(social-engineering)技巧进行的系统访问尝试。
- 控制对硬件的访问　所有驻留数据库的计算机硬件都应该位于安全环境中，只可以使用证件或安全访问代码进行访问。

9.2 数据库验证方法

在数据库允许某人或应用程序对数据库中的对象或权限进行访问之前，必须验证此人或应用程序。换句话说，需要验证尝试访问数据库的人的身份。

本节将概述用于允许访问数据库的最基本方法：用户账户，也称为“数据库验证”。此外，本章也将介绍如何减少用户需要记住的密码数量，其方法是允许操作系统验证用户，并且自动将用户连接到数据库。使用通过应用程序服务器的 3 层验证、网络验证或 Oracle 的身份管理(Identity Management)，这样可以更进一步减少密码数量。最后，本节讨论在数据库停机或无法提供验证服务时，使用密码文件来验证 DBA。

9.2.1 数据库验证

在通过使用防火墙来隔离外部环境和网络，并且客户和数据库服务器之间的网络通信量使用某种加密方法的环境中，数据库验证是在数据库中验证用户的最常见和最容易的方法。验证用户需要的所有信息存储在 SYSTEM 表空间中的某个表中。

非常特殊的数据库操作，例如启动和关闭数据库，需要不同的和更安全的验证形式，即通过使用操作系统验证或使用密码文件。

网络验证依赖于第三方的验证服务，例如分布式计算环境(Distributed Computing Environment，DCE)、Kerberos、公钥基础结构(Public Key Infrastructure，PKI)以及远程验证拨号用户服务(Remote Authentication Dial-In User Service，RADIUS)。3 层验证虽然乍看起来是一种网络验证方法，但它与网络验证方法的区别在于，中间层——例如 Oracle 应用程序服务器(Oracle Application Server)—— 在验证用户的同时在服务器上维护客户的身份。此外，中间层为客户提供了连接入池服务，并且实现了业务逻辑。

本章第 9.2.8 节中将介绍 DBA 在数据库中建立账户进行验证可以使用的所有方法。

9.2.2 数据库管理员验证

数据库并不总是可用于验证数据库管理员，例如在由于意外的断电或由于进行脱机数据库备份而关闭数据库时。为了解决这种情况，Oracle 使用密码文件来维护一个数据库用户列表，允许这些用户执行一些功能，例如启动和关闭数据库、初始化备份等。

另外，数据库管理员可以使用操作系统验证，下面的小节将讨论该方法。图 9-1 中显示的流程图标识了数据库管理员在决定哪种方法最适合应用于他们环境中时的选择。

图 9-1　验证方法流程图

对于本地服务器连接，主要的考虑方面是以下两种情况之间的对比：将相同账户用于操作系统和 Oracle 服务器的方便性与维护密码文件的方便性。对于远程管理员，连接的安全性是选择验证方法时的驱动因素。如果没有安全的连接，黑客就可以很容易地扮演具有和服务器自身上的管理员相同账户的用户，并且可以获得对具有 OS 验证的数据库的完全访问。

注意：
使用密码文件进行验证时，确保密码文件自身在某个目录位置中，只有操作系统管理员和拥有 Oracle 软件安装权限的用户或组可以访问该目录位置。

本章后面将更详细地讨论系统权限。然而，现在只需要知道存在三种特殊的系统权限，它们可以为管理员提供数据库中的特殊验证：SYSDBA、SYSOPER 和 SYSASM。具有 SYSOPER 权限的管理员可以启动和关闭数据库，执行联机和脱机备份，归档当前的重做日志文件，并且在数据库处于 RESTRICTED SESSION 模式时连接到该数据库。SYSDBA 权限包含 SYSOPER 的所有权利，另外还能够创建数据库，并且可授权 SYSDBA 或 SYSOPER 权限给其他数据库用户。SYSASM 权限是 Oracle Database 11g 的新增特性，它是 ASM 实例所特有的，用来管理数据库存储。

为了从 SQL*Plus 会话连接到数据库，可以将 AS SYSDBA 或 AS SYSOPER 附加到 connect 命令。下面是

一个示例：

```
[oracle@dw ~]$ sqlplus /nolog
SQL*Plus: Release 11.1.0.6.0 - Production on Fri Aug 10 20:57:30 2007
Copyright (c) 1982, 2007, Oracle.  All rights reserved.
SQL> connect rjb/rjb as sysdba
Connected.
SQL> show user
USER is "SYS"
SQL>
```

对于作为 SYSDBA 或 SYSOPER 连接的用户，除了具有不同的额外权限，在他们连接到数据库时，默认的模式也不相同。使用 SYSDBA 或 SYSASM 权限连接的用户将作为 SYS 用户连接，SYSOPER 权限设置用户为 PUBLIC：

```
SQL> show user
USER is "SYS"
```

和任何数据库连接请求一样，可以在 sqlplus 命令的相同行上指定用户名和密码，同时指定 SYSDBA 或 SYSOPER 关键字：

```
[oracle@dw ~]$ sqlplus rjb/rjb as sysdba
```

虽然使用 Oracle Universal Installer(具有种子数据库)或使用 Database Creation Assistant 的 Oracle Database 默认安装将自动创建密码文件，但在无意中删除或损坏密码文件的情况下，可能需要重新创建该文件。orapwd 命令可创建密码文件，其中只有一个用于 SYS 用户的条目，而在运行没有任何选项的 orapwd 命令时，其他选项将显示出来：

```
[oracle@dw ~]$ orapwd
Usage: orapwd file=<fname> password=<password>
     entries=<users> force=<y/n> ignorecase=<y/n> nosysdba=<y/n>
  where
    file - name of password file (required),
    password - password for SYS (optional),
    entries - maximum number of distinct DBA (required),
    force - whether to overwrite existing file (optional),
    ignorecase - passwords are case-insensitive (optional),
```

```
    nosysdba - whether to shut out the SYSDBA logon
      (optional Database Vault only).
  There must be no spaces around the equal-to (=) character.
[oracle@dw ~]$
```

一旦重新创建密码文件，则必须将 SYSDBA 和 SYSOPER 权限授权给前面已经具有这些权限的数据库用户。此外，如果在 orapwd 命令中提供的密码不是 SYS 账户在数据库中具有的相同密码，也没有什么问题：当使用 connect / as sysdba 连接数据库时，使用的是操作系统验证；如果使用 connect sys/syspassword as sysdba 连接数据库，则密码 syspassword 是 SYS 在数据库中的密码。这里要重申的只是，如果数据库处于停机状态或处于 MOUNT 模式，则必须使用操作系统验证或密码文件验证。另外值得注意的是，操作系统验证优先于密码文件验证，因此，只要满足了操作系统验证需求，即使存在密码文件，也不使用密码文件进行验证。

警告：
从 Oracle Database 11g 开始，数据库密码是大小写敏感的。要禁用大小写敏感，可以将 SEC_CASE_SENSITIVE_LOGON 初始参数设置为 FALSE。

系统初始参数 REMOTE_LOGIN_PASSWORDFILE 可以控制密码文件如何用于数据库实例。它有 3 个可能的值：NONE、SHARED 以及 EXCLUSIVE。

如果 REMOTE_LOGIN_PASSWORDFILE 参数的值是 NONE，则 Oracle 忽略任何已有的密码文件。必须通过其他方式验证任何具有权限的用户，例如通过操作系统验证，9.2.3 小节将讨论操作系统验证。

REMOTE_LOGIN_PASSWORDFILE 参数的值是 SHARED 时，多个数据库可以共享相同的密码文件，但只有 SYS 用户使用密码文件来验证，并且不可以改变 SYS 的密码。因此，这种方法不是最安全的，但它确实允许 DBA 使用一个 SYS 账户维护多个数据库。

提示：

如果必须使用共享的密码文件，确保 SYS 的密码至少为 8 个字符长，并且包括如下的组合：大小写字母、数字以及可以防护强力猜测攻击的特殊字符。

REMOTE_LOGIN_PASSWORDFILE 参数的值是 EXCLUSIVE 则表示将密码文件只绑定到一个数据库，并且其他数据库用户账户可以存在于密码文件中。一旦创建密码文件，使用这个值可最大化 SYSDBA 和 SYSOPER 连接的安全性。

动态性能视图 V$PWFILE_USERS 列出了所有具有 SYSDBA 或 SYSOPER 权限的数据库用户，如下所示：

```
SQL> select * from v$pwfile_users;
USERNAME                       SYSDB SYSOP SYSAS
------------------------------ ----- ----- -----
SYS                            TRUE  TRUE  FALSE
RJB                            TRUE  FALSE FALSE
```

9.2.3 操作系统验证

如果 DBA 选择实现操作系统验证，则在数据库用户使用下面的 SQL*Plus 语法时，他将自动连接到数据库：

```
SQL> sqlplus /
```

该方法类似于管理员连接到数据库的方法，但是没有 as sysdba 或 as sysoper 子句。主要的区别在于，使用了操作系统账户授权方法，而不是使用 Oracle 生成和维护的密码文件。

实际上，管理员也可以使用操作系统验证，通过 as sysdba 或 as sysoper 进行连接。如果管理员的操作系统登录账户位于 Unix 组 dba(或 Windows 组 ORA_DBA)中，管理员就可以使用 as sysdba 连接到数据库。类似地，如果操作系统登录账户位于 Unix 组 oper(或 Windows 组 ORA_OPER)中，管理员就可以使用 as sysoper 连接到数据库，而不需要使用 Oracle 密码文件。

Oracle Server 建立如下假设：如果通过操作系统账户验证用户，则该用户也通过了数据库验证。使用操作系统验证时，Oracle 不需要维护数据库中的密码，但仍然维护用户名。用户名仍然需要设置默认模式和表空间，此外还需要提供信息进行审计。

在默认的 Oracle 11g 安装中，以及在以前的 Oracle 版本中，如果使用 identified externally 子句

创建数据库用户，则为用户账户启用操作系统验证。数据库用户名的前缀必须匹配初始参数OS_AUTHENT_PREFIX的值，默认值是OPS$。下面是示例：

```
SQL> create user ops$corie identified externally;
```

当用户使用账户CORIE登录到操作系统中时，将在Oracle数据库中自动验证该账户，如同使用数据库验证创建OPS$CORIE账户一样。

设置OS_AUTHENT_PREFIX的值为空字符串，这就允许数据库管理员和操作系统账户管理员在使用外部验证时使用相等的用户名。

使用identified globally类似于使用identified externally，这时在数据库外部完成验证。然而使用全局标识的用户时，则通过企业目录服务执行验证，例如Oracle Internet Directory(OID)。OID可简化数据库管理员的账户维护工作，并且方便需要访问多个数据库或服务的数据库用户的单点登录(Single Sign-On，SSO)。

9.2.4 网络验证

通过网络服务进行验证是DBA验证数据库用户的另一个可用选项。虽然完整的处理方法超出了本书的范围，但此处将给出每种方法及其组成部分的简要概述。这些组成部分包括安全套接字层(Secure Sockets Layer，SSL)、分布式计算环境(Distributed Computing Environment，DCE)、Kerberos、PKI、RADIUS以及基于目录的服务。

1. 安全套接字层协议

安全套接字层(SSL)最初是由Netscape Development Corporation开发的用于Web浏览器中的协议。因为它是公共标准，并且是开放源代码的，因此需要接受编程界的连续审查，从而确保没有漏洞或“后门”可以损害到其健壮性。

最低限度，需要使用服务器端的证书进行验证。客户验证也可使用SSL，验证客户的有效性，但建立证书很可能成为一项庞大的管理工作。

在TCP/IP上面使用SSL只需要稍微改变监听器配置，具体做法是在listener.ora文件中的不同端口号中添加另一个协议(TCPS)。在下面的摘要中，使用Oracle Net Configuration Assistant(netca)进行配置，服务器dw10g上名为LISTENER的监听器将接受端口1521上的TCP流量以及端口2484上的SSL TCP流量：

```
# listener.ora Network Configuration File:
    /u01/app/oracle/product/10.1.0/network/admin/listener.ora
# Generated by Oracle configuration tools.
SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = PLSExtProc)
```

```
      (ORACLE_HOME = /u01/app/oracle/product/10.1.0)
      (PROGRAM = extproc)
    )
    (SID_DESC =
      (GLOBAL_DBNAME = dw.world)
      (ORACLE_HOME = /u01/app/oracle/product/10.1.0)
      (SID_NAME = dw)
    )
  )
LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = TCP)(HOST = dw10g)(PORT = 1521))
      )
      (ADDRESS_LIST =
        (ADDRESS = (PROTOCOL = TCPS)(HOST = dw10g)(PORT = 2484))
      )
    )
  )
```

2. 分布式计算环境

分布式计算环境(Distributed Computing Environment，DCE)提供了大量服务，例如远程过程调用、分布式文件服务以及分布式时间服务，此外还有安全服务。DCE 支持所有主要软件和硬件平台上不同环境类型中的分布式应用程序。

DCE 是支持单点登录(SSO)的一种协议。一旦用户使用 DCE 进行验证，他们就可以安全地访问使用 DCE 配置的任何 Oracle 数据库，而不需要指定用户名或密码。

3. Kerberos

Kerberos 是另一个可信的第三方验证系统，类似于 DCE，它也提供了 SSO 功能。Oracle 完全支持 Kerberos 版本 5，该版本具有 Oracle Database 10g 和 11g 企业版本下的 Oracle Advanced Security。

和其他中间件验证解决方案一样，基本前提是永远不要通过网络发送密码，所有的验证都由 Kerberos 服务器进行代理。在 Kerberos 术语中，密码是“共享的秘密”。

4. 公钥基础结构

公钥基础结构(Public Key Infrastructure，PKI)由大量组件组成。使用 SSL 协议实现公钥基础结构，并且该基础结构是基于秘密的私钥和相关的公钥，从而有助于客户和服务器之间的安全通信。

为了提供标识和验证服务，PKI 使用证书和认证授权(Certificate Authority，CA)。简单地说，证书是实体的公钥，由可信的第三方(认证授权中心)验证该公钥，并且证书包含证书用户的名称、到期日期、公钥等相关信息。

5. RADIUS

远程验证拨号用户服务(Remote Authentication Dial-In User Service，RADIUS)是一个轻量级的协议，用于验证以及授权和账户管理服务。在 Oracle 环境中，从 Oracle 客户发送授权请求时，Oracle Server 扮演 RADIUS 服务器的客户。

支持 RADIUS 标准的任何验证方法，无论是标记卡、智能卡或 SecurID ACE，都可以作为新的验证方法简单地添加到 RADIUS 服务器，而不需要在客户或服务器配置文件(例如 sqlnet.ora)上进行任何改动。

9.2.5 3 层验证

在 3 层或多层环境中，应用程序服务器可以为客户提供验证服务，并且提供数据库服务器的常见接口，即使客户使用多种不同的浏览器或“胖”客户应用程序。然后，使用数据库验证应用程序服务器，并且表

明允许客户连接到数据库，从而在所有层中保存客户的身份。

在多层环境中，给予用户和中间层尽可能少的权限，即只向它们提供完成工作所需的权限。使用如下的命令授权中间层代表用户执行操作：

```
alter user kmourgos
     grant connect through oes_as
     with role all except ordmgmt;
```

在该示例中，授权应用程序服务器服务 OES_AS 代表数据库用户 KMOURGOS 执行操作。已经赋予了用户 KMOURGOS 很多角色,并且可以通过应用程序服务器启用所有这些角色,除了 ORDMGMT 角色。因此,当 KMOURGOS 通过应用程序服务器连接时，允许他通过 Web 访问通过角色授予他的所有表和权限，除了订单管理功能。因为他所在公司具有适当的业务规则,所有对订单管理应用程序的访问必须通过对数据库的直接连接完成。本章第 9.3.4 节中将详细讨论角色。

9.2.6 客户端验证

客户端验证是在多层环境中验证用户的一种方法，但 Oracle 强烈建议不使用这种方法，除非所有的客户都位于安全的网络中，在防火墙内，并且不允许来自于防火墙外部的任何数据库连接。此外，用户不应该具有任何可以连接到数据库的任何工作站上的管理权利。

如果使用 IDENTIFIED EXTERNALLY 属性创建 Oracle 用户，并且将初始参数 REMOTE_ OS_AUTHENT 设置为 TRUE，则攻击者可以简单地使用匹配 Oracle 用户账户的本地用户账户来在工作站上验证自己，并且最终获得对数据库的访问。

因此，强烈推荐将 REMOTE_OS_AUTHENT 参数设置为 FALSE。必须停止并重新启动数据库，从而使这一改动生效。

注意：
在 Oracle Database 11g 中不赞成使用参数 REMOTE_OS_AUTHENT。有其他几种更安全的方法允许远程访问数据库。

9.2.7 Oracle 身份管理

Oracle 身份管理(Identity Management，IM)是 Oracle Application Server 10g 和 11g 的一个组成部分，它为集中管理用户账户提供了完整的端到端的框架，从账户创建到资源授权再到账户删除。它集中了如下方面的管理：账户、设备、应用程序、Web 服务或任何其他使用验证和授权的网络实体。

IM 节省了金钱和时间。因为用户账户和相关的资源是集中的，所以不论如何维护应用程序，管理都是相同的。

此外，IM 增强了企业的安全性。因为用户只使用一个用户名和密码来访问所有的企业资源，所以他们基本上不会倾向于写下或忘记自己的密码。当用户离开公司时，可以快速而容易地在一个位置删除所有对应用程序和服务的访问。

虽然 Oracle 身份管理(IM)的完整讨论超出了本书的范围，但是 DBA 需要重点理解 IM 的组成部分如何影响 Oracle 数据库的性能和安全性。用户账户信息和其他元数据需要存储在 Oracle 数据库中的某个地方，并且冗余存储。此外，必须在合理的时间量内处理对验证和授权服务的请求，一般是在 Service Level Agreement(SLA)中定义这个时间量，使其对一个或多个应用程序生效。

例如，Oracle Internet Directory(OID)是 Oracle 身份管理的一个主要组成部分，它需要在某些位置像调整 OLTP 系统一样进行数据库调整，来自于大量用户的许多较短事务在一天的不同时间其负载的变化范围很大。但是，相似点仅限于此。表 9-1 中是设置数据库的各种系统初始参数的总体指导原则，该数据库将维护轻量级目录访问协议(Lightweight Directory Access Protocol，LDAP)目录信息。

表 9-1　OID 的数据库初始参数大小调整

数据库参数	500 个并发用户	2000 个并发用户
OPEN_CURSORS	200	200
SESSIONS	225	1200
DB_BLOCK_SIZE	8K	8K
DB_CACHE_SIZE	250MB	250MB
SHARED_POOL_SIZE	40MB	40MB
PROCESSES	400	1500
SORT_AREA_SIZE	256KB	256KB
LOG_BUFFER	512KB	512KB

假设这个数据库唯一的工作是维护 OID 目录信息。除了调整基本的数据库参数以外，整体的吞吐量还将取决于如下因素：服务器和用户团体之间的可用网络带宽、共享磁盘资源的位置、磁盘吞吐量等。具有 500 000 个目录条目的典型 IM 部署将需要大约 3GB 的磁盘空间，将条目写入磁盘和从磁盘读取条目的速度将很容易成为吞吐量的瓶颈。

9.2.8　用户账户

为了获得对数据库的访问，用户必须提供用户名，这样才能访问与账户关联的资源。每个用户名必须有一个密码，并且只和数据库中的一个模式关联。有些账户可能在模式中没有任何对象，但相反会有授权给该账户的权限，用于访问其他模式中的对象。

本节将解释相关语法，并且给出创建、改变和删除用户的示例。此外，本节将介绍如何成为另一个用户，而不需要显式地知道该用户的密码。

1. 创建用户

create user 命令非常简单。它具有大量参数，表 9-2 列出了这些参数，并且简要描述了每个参数。

表 9-2　CREATE USER 命令的选项

参　　数	用　　法
username	模式的名称，因此是将要创建的用户名。用户名最多可以为 30 个字符长，并且不可以是保留字，除非加上引号(不推荐这样做)
IDENTIFIED { BY *password*	EXTERNALLY	GLOBALLY AS '*extname*' }	指定如何验证用户：通过数据库使用密码验证，通过操作系统(本地的或远程的)，或者通过服务(例如 Oracle Internet Directory)
DEFAULT TABLESPACE *tablespace*	在其中创建永久对象的表空间，除非在创建期间显式指定了一个表空间
TEMPORARY TABLESPACE *tablespace*	在排序操作、创建索引等期间，在其中创建临时段的表空间
QUOTA { *size*	UNLIMITED } ON *tablespace*	为指定表空间上创建的对象预留的空间量。其大小以千字节(K)或兆字节(M)为单位
PROFILE *profile*	赋予这个用户的配置文件。本章后面将讨论配置文件。如果没有指定配置文件，则使用 DEFAULT 配置文件
PASSWORD EXPIRE	在第一次登录时，用户必须改变他们的密码
ACCOUNT {LOCK	UNLOCK}	指定是否锁定账户或解除账户锁定。默认情况下，账户是解除锁定的

在下面的示例中，创建了一个用户(SKING)，对应于用户 Steven King，该用户在 HR.EMPLOYEES 表中的雇员号为 100，该表来自于和数据库一起安装的示例模式：

```
SQL> create user sking identified by sking901
  2     account unlock
  3     default tablespace users
  4     temporary tablespace temp;
User created.
```

通过数据库验证用户 SKING，该用户具有初始密码 SKING901。第二行不是必需的，所有账户在创建时都是默认解除锁定的。在数据库级别中定义默认的永久表空间和默认的临时表空间，从而该命令的最后两行不是必需的，除非需要为用户提供不同的默认永久表空间或默认临时表空间。

即使已经显式地或隐式地为用户 SKING 分配了默认的永久表空间，他也不可以在数据库中创建任何对象，直到提供磁盘限额以及在他们自己的模式中创建对象的权利。

限额是针对给定用户的、按照表空间的简单空间限制。除非显式分配限额或授予用户 UNLIMITED TABLESPACE 权限(本章后面将讨论权限)，否则用户不可以在自己的模式中创建对象。在下面的示例中，为

SKING 账户提供 USERS 表空间中 250MB 的限额：

```
SQL> alter user sking quota 250M on users;
User altered.
```

注意，可能已经在创建账户时随同 create user 命令中的几乎其他每个选项一起授予了该限额。然而，只可以在创建账户后分配默认的角色(本章后面将讨论角色管理)。

除非将一些基本的权限授予新的账户，否则账户甚至不可以登录。因此，至少需要授予 CREATE SESSION 权限或 CONNECT 角色(本章后面将详细讨论角色)。对于 Oracle Database 10g 版本 1 及更早的版本，CONNECT 角色包含 CREATE SESSION 权限以及其他基本权限，例如 CREATE TABLE 和 ALTER SESSION。从 Oracle Database 10g 版本 2 开始，CONNECT 角色只有 CREATE SESSION 权限，因此不赞称使用 CONNECT 角色。在下面的示例中，将 CREATE SESSION 和 CREATE TABLE 权限授予 SKING：

```
SQL> grant create session, create table to sking;
Grant succeeded.
```

现在，用户 SKING 具有 USERS 表空间上的限额，同时具有在该表空间中创建对象的权限。

可以从基于 Web 的 Oracle Enterprise Manager 界面上获得 create user 的所有这些选项，如图 9-2 所示。

和任何 Enterprise Manager 操作一样，Show SQL 按钮显示了实际的 SQL 命令，例如 create 和 grant，创建用户时将运行这些命令。这是利用 Web 界面易用性的极好方法，同时也可以复习 SQL 命令的语法。

在图 9-3 中可以看到，也可以非常容易地选择一个已有的用户，并且创建具有相同特征(除了密码)的新用户。

图 9-2　使用 Enterprise Manager 创建用户

图 9-3　使用 Enterprise Manager 复制用户

Enterprise Manager 界面中的其他可用选项包括使用户账户到期，生成用于创建账户的 DDL，以及锁定账户或解除账户锁定。

2. 改变用户

可以通过使用 alter user 命令改变用户的特征。alter user 命令的语法基本等同于 create user 命

令的语法，但 alter user 命令允许分配角色以及授权给中间层应用程序，该应用程序代表用户执行功能。

在该示例中，将用户 SKING 改为使用不同的默认永久表空间：

```
SQL> alter user sking
  2       default tablespace users2
  3       quota 500M on users2;
User altered.
```

注意，用户 SKING 仍然可以在 USERS 表空间中创建对象，但他必须在任何 create table 和 create index 命令中显式指定 USERS。

3. 删除用户

删除用户非常简单，可以使用 drop user 命令来完成。此命令唯一的参数是需要删除的用户名以及 cascade 选项。如果没有使用 cascade 选项，则必须显式删除该用户拥有的任何对象，或将这些对象移动到另一个模式。在下面的示例中，删除用户 QUEENB，如果 QUEENB 拥有任何对象，则也自动删除这些对象：

```
SQL> drop user queenb cascade;
User dropped.
```

如果任何其他模式对象，例如视图或程序包，依赖于删除用户时删除的对象，则这些模式对象标记为 INVALID，并且必须重新编码以使用其他对象，然后重新编译。此外，如果删除第一个用户，则由第一个用户通过 with grant option 子句授予第二个用户的任何对象权限都会自动取消。

4. 成为另一个用户

为了调试应用程序，DBA 有时需要作为另一个用户来连接数据库，从而模仿存在的问题。不需要知道该用户实际的纯文本密码，DBA 可以从数据库中检索加密的密码，改变该用户的密码，使用改过的密码连接到数据库，然后使用 alter user 命令中没有归档的子句改回密码。上面的操作假设 DBA 可以访问 DBA_USERS 表，同时具有 ALTER USER 权限。如果 DBA 具有 DBA 角色，则可以满足这两种情况。

第一步是检索用户的加密密码，该密码存储在表 DBA_USERS 中：

```
SQL> select password from dba_users
  2      where username = 'SKING';
PASSWORD
------------------------------
83C7CBD27A941428
```

```
1 row selected.
```

在 GUI 环境中使用复制和粘贴保存该密码，或者将其保存在文本文件中，便于以后进行检索。下一步是临时改变用户的密码，然后使用临时密码进行登录：

```
SQL> alter user sking identified by temp_pass;
User altered.
SQL> connect sking/temp_pass@dw;
Connected.
```

此时,可以从SKING的观点来调试应用程序。一旦完成调试,使用alter user命令中未归档的by values子句改回密码：

```
SQL> alter user sking identified by values '83C7CBD27A941428';
User altered.
```

5. 与用户相关的数据字典视图

大量数据字典视图都包含与用户和用户特征相关的信息。表 9-3 列出了最常见的视图和表。

表 9-3　与用户相关的数据字典视图和表

数据字典视图	说　　明
DBA_USERS	包含用户名、加密的密码、账户状态以及默认的表空间
DBA_TS_QUOTAS	按照用户和表空间的磁盘空间利用率以及限制，针对其限额不是 UNLIMITED 的用户

DBA_PROFILES	可以赋予用户的配置文件，这些用户具有赋予配置文件的资源限制
USER_HISTORY$	具有用户名、加密密码和时间戳的密码历史记录。如果将初始参数 RESOURCE_LIMIT 设置为 TRUE，则用于实施密码重用规则，使用 alter profile 参数 password_reuse_*可以限制密码重用

9.3 数据库授权方法

一旦使用数据库验证了用户，下一步就是确定用户有权访问或使用的对象类型、权限和资源。本节中将介绍配置文件控制管理密码的方式，还将介绍配置文件在各种类型的系统资源上添加限制的方式。

此外，本节将讨论 Oracle 数据库中两种类型的权限：系统权限和对象权限。这两种权限都可以直接赋予用户，或者通过角色间接赋予用户，这是另一种在将权限赋予用户时可以简化 DBA 工作的机制。

本节的末尾将概述 Oracle 的虚拟专用数据库(VPD)特性，以及如何使用此特性更精确地控制：用户根据赋予自己的一组 DBA 定义的证书可以查看表的哪些部分。为了帮助使这一概念更加清晰，我们将完整地实现一个 VPD。

9.3.1 配置文件的管理

看起来似乎总是不会有足够的 CPU 功率或磁盘空间或 I/O 带宽来运行用户的查询。因为所有这些资源本来就是有限的，Oracle 提供了一种机制来控制用户可以使用多少资源，Oracle 配置文件就是提供这种机制的指定资源限制集。

此外，配置文件可用作授权机制来控制如何创建、重用和验证用户密码。例如，可能希望实施最小的密码长度，同时需要密码中至少出现一个大写字母和一个小写字母。本节中将讨论配置文件如何管理密码和资源。

1. CREATE PROFILE 命令

create profile 命令有双重用途。可以创建配置文件，将用户的连接时间限制为 120 分钟。

```
create profile lim_connect limit
        connect_time 120;
```

类似地，可以限制在锁定账户之前登录可以连续失败的次数：

```
create profile lim_fail_login limit
        failed_login_attempts 8;
```

或者，可以将这两种类型的限制合并在一个配置文件中：

```
create profile lim_connectime_faillog limit
        connect_time 120
        failed_login_attempts 8;
```

Oracle 如何响应超出的一种资源限制取决于限制的类型。当到达一个连接时间限制或空闲时间限制(如 CPU_PER_SESSION)时，回滚进行中的事务，并且取消会话连接。对于大多数其他的资源限制(如 PRIVATE_SGA)，回滚当前的事务，将一个错误返回给用户，并且用户可以选择提交或回滚事务。如果操作超出某个调用的限制(如 LOGICAL_READS_PER_ CALL)，则中断该操作，回滚当前的语句，并且将一个错误返回给用户。事务的剩余部分保持不变，然后，用户可以回滚、提交或尝试在不超出语句限制的情况下完成事务。

Oracle 提供了 DEFAULT 配置文件，如果没有指定其他的配置文件，则将该配置文件应用于任何新用户。下面针对数据字典视图 DBA_PROFILES 的查询显示了 DEFAULT 配置文件的 限制：

```
SQL> select * from dba_profiles
  2      where profile = 'DEFAULT';
PROFILE          RESOURCE_NAME                RESOURCE LIMIT
---------- ------------------------     -------- ----------
DEFAULT          COMPOSITE_LIMIT               KERNEL   UNLIMITED
DEFAULT          SESSIONS_PER_USER           KERNEL   UNLIMITED
DEFAULT          CPU_PER_SESSION               KERNEL   UNLIMITED
DEFAULT          CPU_PER_CALL                KERNEL   UNLIMITED
DEFAULT          LOGICAL_READS_PER_SESSION KERNEL   UNLIMITED
DEFAULT          LOGICAL_READS_PER_CALL      KERNEL   UNLIMITED
DEFAULT          IDLE_TIME                     KERNEL   UNLIMITED
DEFAULT          CONNECT_TIME                KERNEL   UNLIMITED
DEFAULT          PRIVATE_SGA                   KERNEL   UNLIMITED
DEFAULT          FAILED_LOGIN_ATTEMPTS       PASSWORD 10
DEFAULT          PASSWORD_LIFE_TIME          PASSWORD 180
DEFAULT          PASSWORD_REUSE_TIME       PASSWORD UNLIMITED
DEFAULT          PASSWORD_REUSE_MAX          PASSWORD UNLIMITED
DEFAULT          PASSWORD_VERIFY_FUNCTION  PASSWORD NULL
DEFAULT          PASSWORD_LOCK_TIME          PASSWORD 1
DEFAULT          PASSWORD_GRACE_TIME       PASSWORD 7
16 rows selected.
```

DEFAULT 配置文件中唯一真正的约束将锁定账户前连续不成功的登录尝试数量(FAILED_LOGIN_ATTEMPTS)限制为 10，将必须改变密码前此密码可以使用的天数 (PASSWORD_LIFE_TIME)设置为 180。此外，没有启用任何密码验证功能。

2. 配置文件和密码控制

表 9-4 中是密码相关的配置文件参数。按照天数指定所有时间单位(例如，为了以分钟为单位指定这些参数，可以将其除以 1440)：

```
SQL> create profile lim_lock limit password_lock_time 5/1440;
Profile created.
```

在该示例中，在登录失败指定的次数后，账户将只锁定 5 分钟。

表 9-4　密码相关的配置文件参数

密 码 参 数	说　　明
FAILED_LOGIN_ATTEMPTS	锁定账户前失败的登录尝试次数
PASSWORD_LIFE_TIME	在必须改变密码前可以使用该密码的天数。如果没有在 PASSWORD_GRACE_TIME 中进行改动，则必须在允许登录前改变该密码
PASSWORD_REUSE_TIME	用户在重新使用密码前必须等待的天数；该参数和 PASSWORD_REUSE_MAX 结合起来使用
PASSWORD_REUSE_MAX	在可以重用密码前必须进行的密码改动次数；该参数和 PASSWORD_REUSE_TIME 结合起来使用
PASSWORD_LOCK_TIME	在 FAILED_LOGIN_ATTEMPTS 尝试后锁定账户的天数。在这个时间周期后，账户自动解除锁定
PASSWORD_GRACE_TIME	在多少天之后到期密码必须改变。如果没有在这个时间周期内进行改动，则账户到期，并且必须在用户可以成功登录之前改变该密码
PASSWORD_VERIFY_FUNCTION	PL/SQL 脚本,用于提供高级密码验证例程。如果指定为 NULL(默认值)，则不执行任何密码验证

参数值 unlimited 表示没有限制可以使用的给定资源数量。default 表示该参数从 DEFAULT 配置文件

中获得它的值。

参数 password_reuse_time 和 password_reuse_max 必须同时使用。设置其中一个参数，而不设置另一个参数，则不会有任何作用。在下面的示例中，创建一个配置文件，将 password_reuse_time 设置为 20 天，而将 password_reuse_max 设置为 5：

```
create profile lim_reuse_pass limit
        password_reuse_time 20
        password_reuse_max 5;
```

对于具有该配置文件的用户，如果他们的密码至少已经改变了 5 次，则这些密码可以在 20 天后重新使用。如果为其中一个参数指定一个值，而为另一个参数指定 UNLIMITED，则用户可以永远不重用密码。

和大多数其他的操作一样，使用 Oracle Enterprise Manager 可以很容易地管理配置文件。图 9-4 显示了一个示例：将 DEFAULT 配置文件改为在 15 分钟不活动后取消用户连接。

图 9-4　使用 Oracle Enterprise Manager 改变密码限制

如果希望对如何创建和重用密码提供更严格的控制，例如在每个密码中混合使用大写字母和小写字母，则需要在每个应用程序配置文件中启用 PASSWORD_VERIFY_FUNCTION 限制。Oracle 提供了一个模板来实施组织的密码策略。该模板位于$ORACLE_HOME/rdbms/admin/ utlpwdmg.sql。该脚本的一些关键部分如下：

```
CREATE OR REPLACE FUNCTION verify_function_11G
(username varchar2,
  password varchar2,
  old_password varchar2)
```

```
  RETURN boolean IS
   n boolean;
   m integer;
   differ integer;
   isdigit boolean;
   ischar  boolean;
   ispunct boolean;
   db_name varchar2(40);
   digitarray varchar2(20);
   punctarray varchar2(25);
   chararray varchar2(52);
   i_char varchar2(10);
   simple_password varchar2(10);
   reverse_user varchar2(32);
BEGIN
   digitarray:= '0123456789';
   chararray:= 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
. . .
   -- Check if the password is same as the username reversed
   FOR i in REVERSE 1..length(username) LOOP
     reverse_user := reverse_user || substr(username, i, 1);
   END LOOP;
   IF NLS_LOWER(password) = NLS_LOWER(reverse_user) THEN
     raise_application_error(-20003, 'Password same as username reversed');
   END IF;
. . .
   -- Everything is fine; return TRUE ;
   RETURN(TRUE);
END;
/
-- This script alters the default parameters for Password Management
-- This means that all the users on the system have Password Management
-- enabled and set to the following values unless another profile is
-- created with parameter values set to different value or UNLIMITED
-- is created and assigned to the user.
ALTER PROFILE DEFAULT LIMIT
PASSWORD_LIFE_TIME 180
PASSWORD_GRACE_TIME 7
PASSWORD_REUSE_TIME UNLIMITED
PASSWORD_REUSE_MAX UNLIMITED
FAILED_LOGIN_ATTEMPTS 10
PASSWORD_LOCK_TIME 1 PASSWORD_VERIFY_FUNCTION verify_function_11G;
```

该脚本为密码复杂性提供了如下功能：

- 确保密码与用户名不同。
- 确保密码至少具有 4 个字符长。
- 进行检查，确保密码不是简单的、显而易见的单词，例如 ORACLE 或 DATABASE。
- 需要密码包含一个字母、一个数字以及一个标点符号。
- 确保密码与前面的密码至少有 3 个字符不同。

为了使用这一策略，首先应对该脚本进行自定义改动。例如，可能希望具有几个不同的验证函数，每一个函数针对一个国家或一个业务部门，用于将数据库密码复杂性需求匹配特定国家或业务部门中使用的操作系统的需求。例如，可以将这种函数重命名为 VERIFY_FUNCTION_US_MIDWEST。此外，可能希望将简单单词的列表改为包括公司的部门名称或建筑大楼名称。

一旦成功编译该函数，可以通过 alter profile 命令，改变已有的配置文件以使用该函数，或者可以创建使用该函数的新的配置文件。在下面的示例中，改变 DEFAULT 配置文件以使用函数 VERIFY_FUNCTION_US_MIDWEST：

```
SQL> alter profile default limit
  2      password_verify_function verify_function_us_midwest;
Profile altered.
```

对于所有使用 DEFAULT 配置文件的已有用户，或者是使用 DEFAULT 配置文件的新用户，通过 VERIFY_FUNCTION_US_MIDWEST 函数检查他们的密码。如果该函数返回不同于 TRUE 的值，则不允许该密码，用户必须指定不同的密码。如果用户的当前密码不符合该函数中的规则，该密码仍然有效，直到改变密码，此时该函数必须验证新的密码。

3. 配置文件和资源控制

表 9-5 中列出了使用 CREATE PROFILE *profilename* LIMIT 后可以出现的资源控制配置文件选项列表。每个参数都可以是整数、UNLIMITED 或 DEFAULT。

表 9-5　与资源相关的配置文件参数

资 源 参 数	说　　明
SESSIONS_PER_USER	用户可以同时具有的最大会话数量
CPU_PER_SESSION	每个会话允许的最大 CPU 时间，以 1%秒为单位
CPU_PER_CALL	语句解析、执行或读取操作的最大 CPU 时间，以 1%秒为单位
CONNECT_TIME	最大总计消耗时间，以分钟为单位
IDLE_TIME	当查询或其他操作停止执行时，会话中的最大连续不活动时间，以分钟为单位
LOGICAL_READS_PER_SESSION	每个会话从内存或磁盘中读取的数据块总量
LOGICAL_READS_PER_CALL	语句解析、执行或读取操作的最大数据块读取量
COMPOSITE_LIMIT	以服务单位划分的总计资源成本，作为 CPU_PER_SESSION、

	CONNECT_TIME、LOGICAL_READS_PER_SESSION 和 PRIVATE_SGA 的组合加权和
PRIVATE_SGA	会话可以在共享池中分配的最大内存量，以字节、千字节或兆字节为单位

和密码相关的参数一样，UNLIMITED 表示没有限制可以使用的资源数量。DEFAULT 表示该参数从 DEFAULT 配置文件中获得它的值。

COMPOSITE_LIMIT 参数允许在使用的资源类型剧烈变化时控制一组资源限制。它允许用户在一个会话期间使用大量 CPU 时间和较少的磁盘 I/O，而在另一个会话期间采用相反的情况，但不需要通过策略来取消连接。

默认情况下，所有的资源成本为 0：

```
SQL> select * from resource_cost;
RESOURCE_NAME                    UNIT_COST
-------------------------------- ----------
CPU_PER_SESSION                           0
LOGICAL_READS_PER_SESSION                 0
CONNECT_TIME                              0
PRIVATE_SGA                               0
4 rows selected.
```

为了调整资源成本的权值，可以使用 ALTER RESOURCE COST 命令。在下面的示例中，改变加权，从而

CPU_PER_SESSION 更关注 CPU 利用率而不是连接时间，其比例系数为 25：1。换句话说，用户将更可能由于 CPU 利用率(而不是连接时间)而取消连接：

```
SQL> alter resource cost
  2      cpu_per_session 50
  3      connect_time 2;
Resource cost altered.
SQL> select * from resource_cost;
RESOURCE_NAME                    UNIT_COST
-------------------------------- ----------
CPU_PER_SESSION                          50
LOGICAL_READS_PER_SESSION                 0
CONNECT_TIME                              2
PRIVATE_SGA                               0
4 rows selected.
```

下一步是创建新的配置文件或修改已有的配置文件，从而可以使用组合的限制：

```
SQL> create profile lim_comp_cpu_conn limit
  2      composite_limit 250;
Profile created.
```

因此，赋予配置文件 LIM_COMP_CPU_CONN 的用户将使用下面的公式计算成本，从而限制他们的会话资

源：

```
composite_cost = (50 * CPU_PER_SESSION) + (2 * CONNECT_TIME);
```

在表 9-6 中，提供了资源利用的一些示例，用于查看是否超出了组合限制 250。

表 9-6　资源利用情况

CPU(秒)	连接(秒)	组合的成本	是 否 超 出
0.05	100	(50*5) + (2*100) = 450	是
0.02	30	(50*2) + (2*30) = 160	否
0.01	150	(50*1) + (2*150) = 350	是
0.02	5	(50*2) + (2*5) = 110	否

在这个特定的示例中没有使用参数 PRIVATE_SGA 和 LOGICAL_READS_PER_SESSION，除非在配置文件定义中的其他位置指定它们，否则它们默认为在 DEFAULT 配置文件中的值。使用组合限制的目的在于用户可以运行更多类型的查询或 DML。在某些天中，他们可能运行许多查询，这些查询执行大量计算，但是没有访问过多的表行。在其他一些天中，他们可能执行许多完整的表扫描，但是没有保持长时间的连接。在这些情况中，不希望通过单一的一个参数来限制用户，而是通过总计的资源利用率来限制用户，该资源利用率按照服务器上每个资源的可用性的加权来获得。

9.3.2　系统权限

系统权限是在数据库中任何对象上执行操作的权利，以及其他一些权限，这些权限完全不涉及对象，而是涉及运行批处理作业、改变系统参数、创建角色、甚至是连接到数据库自身等方面。Oracle 11g 的版本 1 中有 206 个系统权限。可以在数据字典表 SYSTEM_PRIVILEGE_MAP 中看到所有这些权限：

```
SQL> select * from system_privilege_map;
PRIVILEGE   NAME                                    PROPERTY
---------- ----------------------------------------  ----------
        -3   ALTER SYSTEM                                    0
        -4   AUDIT SYSTEM                                    0
        -5   CREATE SESSION                                  0
        -6   ALTER SESSION                                   0
        -7   RESTRICTED SESSION                             0
       -10    CREATE TABLESPACE                             0
       -11    ALTER TABLESPACE                                0
       -12    MANAGE TABLESPACE                             0
       -13    DROP TABLESPACE                                 0
       -15    UNLIMITED TABLESPACE                          0
       -20    CREATE USER                                     0
       -21    BECOME USER                                     0
       -22    ALTER USER                                      0
       -23    DROP USER                                       0
. . .
      -318   INSERT ANY MEASURE FOLDER                       0
```

```
      -319    CREATE CUBE BUILD PROCESS                    0
      -320    CREATE ANY CUBE BUILD PROCESS                0
      -321    DROP ANY CUBE BUILD PROCESS                  0
      -322    UPDATE ANY CUBE BUILD PROCESS                0
      -326    UPDATE ANY CUBE DIMENSION                    0
      -327    ADMINISTER SQL MANAGEMENT OBJECT             0
      -350    FLASHBACK ARCHIVE ADMINISTER                 0
206 rows selected.
```

表 9-7 列出了一些更为常见的系统权限，并且简要描述了这些权限。

表 9-7 常见的系统权限

系 统 权 限	功　　能
ALTER DATABASE	对数据库进行改动，例如将数据库状态从 MOUNT 改为 OPEN，或者是恢复数据库
ALTER SYSTEM	发布 ALTER SYSTEM 语句：切换到下一个重做日志组，改变 SPFILE 中的系统初始参数
AUDIT SYSTEM	发布 AUDIT 语句
CREATE DATABASE LINK	创建到远程数据库的数据库链接
CREATE ANY INDEX	在任意模式中创建索引；针对用户的模式，随同 CREATE TABLE 一起授权 CREATE INDEX

(续表)

系 统 权 限	功　　能
CREATE PROFILE	创建资源/密码配置文件
CREATE PROCEDURE	在自己的模式中创建函数、过程或程序包
CREATE ANY PROCEDURE	在任意的模式中创建函数、过程或程序包
CREATE SESSION	连接到数据库
CREATE SYNONYM	在自己的模式中创建私有同义词
CREATE ANY SYNONYM	在任意模式中创建私有同义词
CREATE PUBLIC SYNONYM	创建公有同义词
DROP ANY SYNONYM	在任意模式中删除私有同义词
DROP PUBLIC SYNONYM	删除公有同义词
CREATE TABLE	在自己的模式中创建表
CREATE ANY TABLE	在任意模式中创建表
CREATE TABLESPACE	在数据库中创建新的表空间
CREATE USER	创建用户账户/模式
ALTER USER	改动用户账户/模式
CREATE VIEW	在自己的模式中创建视图
SYSDBA	如果启用了外部密码文件，则在外部密码文件中创建一个条目；同时，执行启动/关闭数据库，改变数据库，创建数据库，恢复数据库，创建 SPFILE，以及当数据库处于 RESTRICTED SESSION 模式时连接数据库
SYSOPER	如果启用了外部密码文件，则在外部密码文件中创建一个条目；同

	时，执行启动/关闭数据库，改变数据库，恢复数据库，创建 SPFILE，以及当数据库处于 RESTRICTED SESSION 模式时连接数据库

1. 授予系统权限

使用 grant 命令将权限授予用户、角色或 PUBLIC，使用 revoke 命令取消权限。PUBLIC 是一个特殊的组，包含所有的数据库用户，通过它可以方便快捷地将权限授予数据库中的每　个人。

为了授予用户 SCOTT 创建存储过程和同义词的能力，可以使用类似于如下的命令：

```
SQL> grant create procedure, create synonym to scott;
Grant succeeded.
```

取消权限也非常容易：

```
SQL> revoke create synonym from scott;
Revoke succeeded.
```

如果希望允许被授权者有权将相同的权限授予其他某个人，可以在授予权限时包括 with admin option 选项。在前面的示例中，希望用户 SCOTT 能够将 CREATE PROCEDURE 权限授予其他用户。为了实现这一点，需要重新授予 CREATE PROCEDURE 权限：

```
SQL> grant create procedure to scott with admin option;
Grant succeeded.
```

现在用户 SCOTT 可以发布 grant create procedure 命令。注意，如果取消 SCOTT 将该权限授予其他人的许可，他已经授予权限的用户将保留该权限。

2. 系统权限数据字典视图

表 9-8 包含了与系统权限相关的数据字典视图。

表 9-8　系统权限数据字典视图

数据字典视图	说　　明
DBA_SYS_PRIVS	赋予角色和用户的系统权限
SESSION_PRIVS	对该会话的这个用户有效的所有系统权限，直接授权或通过角色
ROLE_SYS_PRIVS	通过角色授权给用户的当前会话权限

9.3.3　对象权限

与系统权限相比，对象权限是在特定对象(如表或序列)上执行特定类型操作的权利，该对象不在用户自己的模式中。和系统权限一样，使用 grant 和 revoke 命令来授予和取消对象上的权限。

和系统权限一样，可以授予对象权限给 PUBLIC 或特定用户，并且具有对象权限的用户可以将其传递给其他用户，其方法是使用 with grant option 子句授予对象权限。

警告：
当所有当前的和未来的数据库用户确切地需要权限时，只将对象权限或系统权限授予 PUBLIC。

一些模式对象，例如集群和索引，依赖于系统权限来控制访问。在这些情况中，如果用户拥有这些对象或具有 ALTER ANY CLUSTER 或 ALTER ANY INDEX 系统权限，则可以改变这些对象。

在自己的模式中拥有对象的用户自动具有这些对象上的所有对象权限，并且可以将这些对象上的任何对象权限授予任意用户或另一个角色，使用或不使用 grant option 子句。

表 9-9 中是可用于不同类型对象的对象权限，一些权限只适用于某些类型的对象。例如，INSERT 权限只对表、视图和物化视图有意义。另一方面，EXECUTE 权限适用于函数、过程和程序包，但不适用于表。

表 9-9　对象权限

对 象 权 限	功　　能
ALTER	可以改变表或序列的定义
DELETE	可以从表、视图或物化视图中删除行
EXECUTE	可以执行函数或过程，使用或不使用程序包
DEBUG	允许查看在表上定义的触发器中的 PL/SQL 代码，或者查看引用表的 SQL 语句。对于对象类型，该权限允许访问在对象类型上定义的所有共有和私有变量、方法和类型
FLASHBACK	允许使用保留的撤销信息在表、视图和物化视图中进行闪回查询
INDEX	可以在表上创建索引
INSERT	可以向表、视图或物化视图中插入行
ON COMMIT REFRESH	可以根据表创建提交后刷新的物化视图
QUERY REWRITE	可以根据表创建用于查询重写的物化视图
READ	可以使用 Oracle DIRECTORY 定义读取操作系统目录的内容
REFERENCES	可以创建引用另一个表的主键或唯一键的外键约束
SELECT	可以从表、视图或物化视图中读取行，此外还可以从序列中读取当前值或下面的值
UNDER	可以根据已有的视图创建视图
UPDATE	可以更新表、视图或物化视图中的行
WRITE	可以使用 Oracle DIRECTORY 定义将信息写入到操作系统目录

值得注意的是，不可以将 DELETE、UPDATE 和 INSERT 权限授予物化视图，除非这些视图是可更新的。一些对象权限和系统权限重复；例如，如果没有表上的 FLASHBACK 对象权限，但只要有 FLASHBACK ANY TABLE 系统权限，就仍然可以执行闪回查询。

在下面的示例中，DBA 授权 SCOTT 对表 HR.EMPLOYEES 的完全访问，但只允许 SCOTT 将 SELECT 对象权限传递给其他用户：

```
SQL> grant insert, update, delete on hr.employees to scott;
Grant succeeded.
SQL> grant select on hr.employees to scott with grant option;
Grant succeeded.
```

注意,如果取消了 SCOTT 在表 HR.EMPLOYEES 上的 SELECT 权限,则也取消他授予该权限的用户的 SELECT 权限。

1. 表权限

可以在表上授予的权限类型主要分为两类:DML 操作和 DDL 操作。DML 操作包括 delete、insert、select 和 update，而 DDL 操作包括添加、删除和改变表中的列，以及在表上创建索引。

授权表上的 DML 操作时，可以将这些操作限制为只针对某些列。例如，可能希望允许 SCOTT 查看和更新 HR.EMPLOYEES 中所有的行和列，除了 SALARY 列。为了做到这一点，首先需要取消表上已有的 SELECT 权限：

```
SQL> revoke update on hr.employees from scott;
Revoke succeeded.
```

接下来，让 SCOTT 更新除了 SALARY 列之外的所有列：

```
SQL> grant update (employee_id, first_name, last_name, email,
  2               phone_number, hire_date, job_id, commission_pct,
  3               manager_id, department_id)
  4  on hr.employees to scott;
Grant succeeded.
```

SCOTT 将能够更新 HR.EMPLOYEES 表中除了 SALARY 列之外的所有列：

```
SQL> update hr.employees set first_name = 'Stephen' where employee_id = 100;
1 row updated.
SQL> update hr.employees set salary = 50000 where employee_id = 203;
update hr.employees set salary = 50000 where employee_id = 203
          *
ERROR at line 1:
ORA-01031: insufficient privileges
```

使用基于 Web 的 OEM 工具也很容易执行该操作，如图 9-5 所示。

图 9-5　在 Oracle Enterprise Manager 中授予列权限

2. 视图权限

视图上的权限类似于在表上授予的权限。假设视图是可更新的，则可以选择、更新、删除或插入视图中的行。为了创建视图，首先需要 CREATE VIEW 系统权限(用于在自己的模式中创建视图)或 CREATE ANY VIEW 系统权限(用于在任意模式中创建视图)。即使是创建视图，也必须至少具有视图的底层表上的 SELECT 对象权限以及 INSERT、UPDATE 和 DELETE 等对象权限(如果希望在视图上执行这些操作，并且视图是可更新的)。作为选择，如果底层的对象不在自己的模式中，则可以有 SELECT ANY TABLE、INSERT ANY TABLE、UPDATE ANY TABLE 或 DELETE ANY TABLE 权限。

为了允许其他人使用您的视图，必须使用 GRANT OPTION 具有视图的基表上的许可，或者必须使用 ADMIN OPTION 具有系统权限。例如，如果创建针对 HR.EMPLOYEES 表的视图，则必须通过 WITH GRANT OPTION 子句授予 HR.EMPLOYEES 表上的 SELECT 对象权限，或者通过 WITH ADMIN OPTION 子句具有 SELECT ANY TABLE 系统权限。

3. 过程权限

对于过程、函数以及包含过程和函数的程序包，EXECUTE 权限是唯一可以应用的对象权限。从 Oracle 8i 开始，可以从定义者、过程或函数的创建者、调用者、运行函数或过程的用户等的角度来运行过程和函数。

使用定义者的权利运行过程时，如同定义者自身运行该过程一样，定义者所有的权限都对过程中引用的对象有效。这是在私有数据库对象上实施约束的好方法：授予其他用户在过程上的 EXECUTE 许可，而没有授予引用对象上的任何许可。结果，定义者可以控制其他用户如何访问对象。

相反，使用调用者权利的过程需要调用者具有针对该过程中引用的任何对象的直接权利，例如 SELECT 和 UPDATE。该过程可能引用了无限定的表 ORDERS，并且如果数据库的所有用户都具有 ORDERS 表，则自己有 ORDERS 表的任何用户都可以使用相同的过程。使用调用者权利的过程的另一个优点是在过程中启用该角色。本章后面将深入讨论角色。

默认情况下，使用定义者的权利创建过程。为了指定过程使用调用者的权利，必须在过程定义中包括关键字 authid current_user，如同下面的示例所示：

```
create or replace procedure process_orders (order_batch_date   date)
authid current_user as
begin
     -- process user's ORDERS table here using invoker's rights,
     -- all roles are in effect
end;
```

为了创建过程，用户必须具有 CREATE PROCEDURE 或 CREATE ANY PROCEDURE 系统权限。对于正确编译的过程，用户必须具有针对过程中引用的所有对象的直接权限，即使在运行时，在使用调用者权利的过程中启用了角色以获得这些相同的权限。为了允许其他用户访问过程，可以授予过程或程序包上的 EXECUTE 权限。

4. 对象权限数据字典视图

大量数据字典视图包含了赋予用户的对象权限的相关信息。表 9-10 列出了包含对象权限信息的最重要的视图。

表 9-10　对象权限数据字典视图

数据字典视图	说　　明
DBA_TAB_PRIVS	授予角色和用户的表权限。包括将权限授予角色或用户的用户，使用或不使用 GRANT OPTION
DBA_COL_PRIVS	授予角色或用户的列权限。包含列名和列上的权限类型
SESSION_PRIVS	对会话的该用户有效的所有系统权限，直接授予或通过角色
ROLE_TAB_PRIVS	对于当前的会话，通过角色授予的表上的权限

9.3.4　创建、分配和维护角色

角色是一组指定的权限，这些权限是系统权限、对象权限或者两者的结合，用于帮助简化权限的管理。不同于单独将系统权限或对象权限授予每个用户，可以将一组系统权限或对象权限授予一个角色，然后将该角色授予用户。这将大量减少维护用户的权限所需的管理开销。图 9-6 显示了角色如何减少在将角色用于分组权限时需要执行的 grant 命令(最终是 revoke 命令)的数量。

图 9-6　使用角色管理权限

如果需要改变由角色授权给一组人的权限，则只需要改变该角色的权限，并且该角色的用户有能力自动使用改动后的新权限。用户可以有选择地启用角色，有些角色可以在登录时自动启用。此外，可以使用密码保护角色，添加对数据库中该功能的另一种验证级别。

在表 9-11 中是数据库自动提供的最常见的角色，其中也简要描述了每个角色中的权限。

表 9-11　预定义的 Oracle 角色

角　色　名	权　　限
CONNECT	Oracle Database 10g 版本 2 之前的版本:ALTER SESSION、CREATE CLUSTER、CREATE DATABASE LINK、CREATE SEQUENCE、CREATE SESSION、CREATE SYNONYM、CREATE TABLE、CREATE VIEW。这些权限一般是提供给数据库普通用户的权限，允许他们连接和创建表、索引以及视图。Oracle Database 10g 版本 2 及之后的版本：只有 CREATE SESSION

(续表)

角　色　名	权　　限
RESOURCE	CREATE CLUSTER、CREATE INDEXTYPE、CREATE OPERATOR、CREATE PROCEDURE、CREATE SEQUENCE、CREATE TABLE、CREATE TRIGGER、CREATE TYPE。这些权限一般用于可能正在编写 PL/SQL 过程和函数的应用程序开发人员
DBA	所有具有 WITH ADMIN OPTION 的系统权限。允许具有 DBA 角色的人将系统权限授予其他人
DELETE_CATALOG_ROLE	没有任何系统权限，而只有 SYS.AUD$和 FGA_LOG$上的对象权限(DELETE)。换句话说，该角色允许用户从用于常规或细粒度审计的审计跟踪中删除审计记录
EXECUTE_CATALOG_ROLE	各种系统程序包、过程和函数上的执行权限，例如 DBMS_FGA 和 DBMS_RLS
SELECT_CATALOG_ROLE	1 638 个数据字典表上的 SELECT 对象权限

EXP_FULL_DATABASE	EXECUTE_CATALOG_ROLE、SELECT_CATALOG_ROLE、以及诸如 BACKUP ANY TABLE 和 RESUMABLE 等系统权限。允许具有该角色的用户导出数据库中的所有对象
IMP_FULL_DATABASE	类似于 EXP_FULL_DATABASE，但是具有多很多的系统权限，例如 CREATE ANY TABLE，用于允许导入前面导出的完整数据库
AQ_USER_ROLE	Advanced Queuing 所需例程的执行访问，例如 DBMS_AQ
AQ_ADMINISTRATOR_ROLE	Advanced Queuing 查询的管理程序
SNMPAGENT	由 Enterprise Manager Intelligent Agent 使用
RECOVERY_CATALOG_OWNER	用于创建一个用户，该用户拥有用于 RMAN 备份和恢复的恢复目录
HS_ADMIN_ROLE	提供对表 HS_*和程序包 DBMS_HS 的访问，用于管理 Oracle Heterogeneous Services
SCHEDULER_ADMIN	提供对程序包 DBMS_SCHEDULER 的访问，以及用于创建批处理作业的权限

提供角色 CONNECT、RESOURCE 和 DBA 主要是为了兼容以前的 Oracle 版本，而在将来的 Oracle 版本中可能不会有这些角色。数据库管理员应该使用授权给这些角色的权限作为起点来创建自定义的角色。

1. 创建或删除角色

为了创建角色，可以使用 create role 命令，并且必须具有 CREATE ROLE 系统权限。一般来说，该系统权限只授权给数据库管理员或应用程序管理员。下面是示例：

```
SQL> create role hr_admin not identified;
Role created.
```

默认情况下，启用或使用已分配的角色不需要任何密码或验证。因此，not identified 子句是可选项。

和创建用户一样，可以通过密码(使用 identified by password 的数据库授权)、通过操作系统(identified externally)或者通过网络或目录服务(identified globally)授权使用角色。

除了这些熟悉的方法，还可以通过使用程序包授权角色：这称为使用“安全应用程序角色”。这种类型的角色使用程序包中的过程来启用角色。一般来说，只在某些条件下启用这种角色：用户正在通过 Web 接口或某个 IP 地址连接，或者是一天的某个时间。下面是使用过程启用的角色：

```
SQL> create role hr_clerk identified using hr.clerk_verif;
Role created.
```

创建角色时，过程 HR.CLERK_VERIF 不需要存在。然而，当授予该角色的用户需要启用它时，它必须经过编译并且有效。一般来说，使用安全应用程序角色时，默认情况下不针对用户启用该角色。为了指定在默认情况下启用除了安全应用程序角色之外的所有角色，可以使用如下的命令：

```
SQL> alter user kshelton default role all except hr_clerk;
User altered.
```

通过这种方式，当 HR 应用程序启动时，它可以启用角色，其方法是执行 set role hr_clerk 命令，从而调用过程 HR.CLERK_VERIF。用户不需要知道角色或启用角色的过程，因此，对象的访问和角色提供的权限都不可用于应用程序外部的用户。

删除角色和创建角色一样简单：

```
SQL> drop role keypunch_operator;
Role dropped.
```

下一次连接到数据库时，赋予该角色的任何用户将丢失赋予该角色的权限。如果他们当前已经登录，他们将保留这些权限，直到断开与数据库的连接。

2. 将权限授予角色

将权限赋予角色非常简单，可以使用 grant 命令将权限赋予角色，如同将权限赋予用户一样：

```
SQL> grant select on hr.employees to hr_clerk;
Grant succeeded.
SQL> grant create table to hr_clerk;
Grant succeeded.
```

在该示例中，将对象权限和系统权限赋予 HR_CLERK 角色。在图 9-7 中，可以使用基于 Web 的 OEM 来将更多的对象权限或系统权限添加给该角色。

图 9-7　使用 OEM 将权限赋予角色

3. 分配或取消角色

一旦已经将所需的对象权限和系统权限赋予角色，就可以使用如下熟悉的语法将角色赋予用户：

```
SQL> grant hr_clerk to smavris;
Grant succeeded.
```

SMAVRIS 可以自动使用未来授予 HR_CLERK 角色的其他任何权限，因为 SMAVRIS 已经被授予该角色。

角色可以授予其他角色，这就允许 DBA 设计多层次的角色，从而使角色管理更为容易。例如，可能已经具有名为 DEPT30、DEPT50 和 DEPT100 的角色，每个角色具有一些对象权限，分别对应各个部门的表。部门 30 中的雇员将分配 DEPT30 角色，依此类推。公司的董事长希望看到所有部门中的表，不必将单个的对象权限赋予角色 ALL_DEPTS，而是可以将单个的部门角色赋予 ALL_DEPTS：

```
SQL> create role all_depts;
Role created.
SQL> grant dept30, dept50, dept100 to all_depts;
Grant succeeded.
SQL> grant all_depts to sking;
Grant succeeded.
```

角色 ALL_DEPTS 可能也包含单个对象权限和系统权限，这些权限不适用于单个部门，例如订单条目表或账户应收款项表上的对象权限。

从用户处取消角色非常类似于从用户处取消权限：

```
SQL> revoke all_depts from sking;
Revoke succeeded.
```

下次用户连接到数据库时，这些取消的权限将不再可用于这些用户。然而，值得注意的是，如果另一个角色包含与删除角色相同对象上的权限，或者直接授予对象上的权限，则用户将保留对象上的这些权限，直到显式地取消这些授权和所有其他授权。

4. 默认的角色

默认情况下，当用户连接到数据库时启用授予该用户的所有角色。如果角色将只用于应用程序的上下文中，则在用户登录时可以先禁用该角色，然后在应用程序中启用和禁用该角色。如果用户 SCOTT 具有 CONNECT、RESOURCE、HR_CLERK 和 DEPT30 角色，希望指定 HR_CLERK 和 DEPT30 默认情况下不启用，则可以使用类似于如下的代码：

```
SQL> alter user scott default role all
  2>     except hr_clerk, dept30;
User altered.
```

当 SCOTT 连接到数据库时，他自动具有除 HR_CLERK 和 DEPT30 外的所有角色授予的所有权限。通过使用 set role，SCOTT 可以在他的会话中显式地启用一个角色：

```
SQL> set role dept30;
Role set.
```

当完成对部门 30 的表的访问时，可以在会话中禁用该角色：

```
SQL> set role all except dept30;
Role set.
```

注意：

在 Oracle 10g 中不赞成使用初始参数 MAX_ENABLED_ROLES。保留该参数只是为了和以前的版本兼容。

5. 启用密码的角色

为了增强数据库中的安全性，DBA 可以为角色赋予密码。在创建角色时为其赋予密码：

```
SQL> create role dept99 identified by d99secretpw;
Role created.
SQL> grant dept99 to scott;
Grant succeeded.
SQL> alter user scott default role all except hr_clerk, dept30, dept99;
User altered.
```

当用户 SCOTT 连接到数据库时，他正在使用的应用程序将提供密码或提示用户输入密码，或者他可以在启用角色时输入密码：

```
SQL> set role dept99 identified by d99secretpw;
Role set.
```

6. 角色数据字典视图

表 9-12 列出了与角色相关的数据字典视图。

表 9-12　与角色相关的数据字典视图

数据字典视图	说　　明
DBA_ROLES	所有的角色以及它们是否需要密码
DBA_ROLE_PRIVS	授予用户或其他角色的角色
ROLE_ROLE_PRIVS	授予其他角色的角色
ROLE_SYS_PRIVS	已经授予角色的系统权限
ROLE_TAB_PRIVS	已经授予角色的表权限和表列权限
SESSION_ROLES	当前对该会话有效的角色。可用于每个用户会话

视图 DBA_ROLE_PRIVS 可以很好地用于：找出哪些角色被授予了用户，这些用户是否可以将该角色传递给另一个用户(ADMIN_OPTION)，以及该角色是否在默认情况下启用(DEFAULT_ROLE)：

```
SQL> select * from dba_role_privs
  2        where grantee = 'SCOTT';
GRANTEE      GRANTED_ROLE         ADMIN_OPTION DEFAULT_ROLE
```

```
---------- -------------------- ------------ ------------
SCOTT      DEPT30                    NO           NO
SCOTT      DEPT50                    NO           YES
SCOTT      DEPT99                    NO           YES
SCOTT      CONNECT                   NO           YES
SCOTT      HR_CLERK                  NO           NO
SCOTT      RESOURCE                  NO           YES
SCOTT      ALL_DEPTS                 NO           YES
SCOTT      DELETE_CATALOG_ROLE  NO              YES
8 rows selected.
```

类似地，可以找出将哪些角色赋予 ALL_DEPTS 角色：

```
SQL> select * from dba_role_privs
  2>       where grantee = 'ALL_DEPTS';
GRANTEE        GRANTED_ROLE        ADMIN_OPTION DEFAULT_ROLE
------------ ------------------ ------------ ------------
ALL_DEPTS      DEPT30              NO           YES
ALL_DEPTS      DEPT50              NO           YES
ALL_DEPTS      DEPT100             NO           YES
```

```
3 rows selected.
```

数据字典视图 ROLE_ROLE_PRIVS 也可以用于获得这些信息。它只包含有关赋予角色的角色信息，没有 DEFAULT_ROLE 信息。

为了找出表或表列上授予用户的权限，可以编写 2 个查询：一个查询用于检索直接授予的权限，另一个查询用于检索通过角色间接授予的权限。检索直接授予的权限非常简单：

```
SQL> select dtp.grantee, dtp.owner, dtp.table_name,
  2        dtp.grantor, dtp.privilege, dtp.grantable
  3  from dba_tab_privs dtp
  4  where dtp.grantee = 'SCOTT';
GRANTEE      OWNER      TABLE_NAME     GRANTOR     PRIVILEGE  GRANTABLE
-----------  ---------  -------------  ----------  ----------  ----------
SCOTT        HR         EMPLOYEES      HR            SELECT       YES
SCOTT        HR         EMPLOYEES      HR           DELETE       NO
SCOTT        HR         EMPLOYEES      HR           INSERT       NO
4 rows selected.
```

为了检索通过角色授予的表权限，需要连接 DBA_ROLE_PRIVS 和 ROLE_TAB_PRIVS。DBA_ROLE_PRIVS 具有赋予用户的角色，而 ROLE_TAB_PRIVS 具有赋予角色的权限：

```
SQL> select drp.grantee, rtp.owner, rtp.table_name,
  2       rtp.privilege, rtp.grantable, rtp.role
  3  from role_tab_privs rtp
  4       join dba_role_privs drp on rtp.role = drp.granted_role
  5  where drp.grantee = 'SCOTT';
GRANTEE    OWNER    TABLE_NAME      PRIVILEGE    GRANTABLE  ROLE
---------- -------- --------------- ------------ ---------- ---------------
SCOTT      HR       EMPLOYEES       SELECT       NO         HR_CLERK
SCOTT      HR       JOBS            SELECT       NO         JOB_MAINT
SCOTT      HR       JOBS            UPDATE       NO         JOB_MAINT
SCOTT      SYS      AUD$            DELETE       NO         DELETE_CATA
                                                            LOG_ROLE
SCOTT      SYS      FGA_LOG$        DELETE       NO         DELETE_CATA
                                                            LOG_ROLE
5 rows selected.
```

在 SCOTT 的权限中，注意他具有 HR.EMPLOYEES 表上的 SELECT 权限，该权限不仅直接通过 grant 命令授予，而且还通过角色授予。取消其中一个权限不会影响 SCOTT 对 HR.EMPLOYEES 表的访问，除非同时删除这两个权限。

9.3.5 使用 VPD 实现应用程序安全策略

虚拟专用数据库(Virtual Private Database，VPD)将服务器实施的细粒度访问控制和安全应用程序上下文结合起来。支持上下文的函数返回一个谓词，即 where 子句，该子句自动附加到所有的 select 语句或其他 DML 语句。换句话说，由 VPD 控制的表、视图、同义词上的 select 语句将根据 where 子句返回行的子集，该子句由通过应用程序上下文生效的安全策略函数自动生成。VPD 的主要组成部分是行级别的安全性(RLS)，也称为“细粒度的访问控制”(FGAC)。

因为 VPD 在语句解析期间透明地生成谓词，因此无论用户是否正在运行特别的查询、检索应用程序中的数据或者查看 Oracle Forms 中的数据，都可以一致地实施安全策略。因为 Oracle Server 在解析时将谓词应用于语句，所以应用程序不需要使用特殊的表、视图等来实现该策略。因此，Oracle 可以使用索引、物化视图和并行操作来优化查询，而以其他的方式则不能够进行优化。因此，相比于使用应用程序或其他方式过滤结果的查询，使用 VPD 可能会产生较少的系统开销。

从维护的角度来看，安全策略可以在安全函数中定义，使用角色和权限很难创建这种安全函数。类似地，应用程序服务器提供商(Application Server Provider，ASP)可能只需要建立一个数据库来为相同应用程序的多个客户服务，使用 VPD 策略来确保一个顾客的雇员只可以查看他们自己的数据。DBA 可以使用少量的 VPD 策略维护一个较大的数据库，而不是针对每个客户都使用一个单独的数据库。

Oracle 10g 中的新增内容是列级别的 VPD 操作。使用列级别的 VPD，DBA 可以约束对表中特定列的访问。查询返回相同数量的行，但如果用户的上下文不允许访问列，则在约束的列中返回 NULL 值。

VPD 策略可以是静态的、上下文相关的或动态的。静态的或上下文相关的 VPD 策略是 Oracle Database 10g 的新增内容，它们可以极大地改进性能，因为它们不需要在每次运行查询时调用策略函数，这是由于在会话中将其缓存以方便以后使用。在 Oracle Database 10g 之前，所有的策略都是动态的。换句话说，每次解析包含目标 VPD 表的 SQL 语句时都运行策略函数。每次登录，静态策略都要评估一次，并且在整个会话期间保持缓存，而不考虑应用程序上下文。使用上下文相关的策略时，如果应用程序上下文改变，则在语句解析时调用策略函数：例如，实施“雇员只可以看到他们自己的薪水历史记录，但经理可以看到他们雇员的所有薪水情况”这种业务规则的策略。如果执行语句的雇员没有改变，就不需要再次调用策略函数，从而减少由于 VPD 策略实施而产生的系统开销量。

可以使用 create context 命令创建应用程序上下文，并且使用程序包 DBMS_RLS 管理 VPD 策略。可以像其他任何函数一样创建用于返回谓词以实施策略的函数，但这种函数具有两个必需的参数，并且返回一个 VARCHAR2。本章后面将详细介绍这些函数，并且使用在 Oracle 数据库安装期间提供的示例模式来创建一个 VPD 示例。

1. 应用程序上下文

使用 create context 命令，可以创建应用程序定义的属性的名称，这些属性用于实施安全策略。此外，还可以定义函数和过程的程序包名称，这些函数和过程用于设置用户会话的安全上下文。下面是示例：

```
create context hr_security using vpd.emp_access;
create or replace package emp_access as
    procedure set_security_parameters;
end;
```

在该示例中，上下文名称是 HR_SECURITY，用于在会话期间为用户建立特征或属性的程序包称为 EMP_ACCESS。在登录触发器中调用过程 SET_SECURITY_PARAMETERS。因为上下文 HR_SECURITY 只绑定到 EMP_ACCESS，因此没有其他的过程可以改变会话属性。这可以确保在连接到数据库后用户或任何其他进程都不可以改变安全的应用程序上下文。

在用于实现应用程序上下文的典型程序包中，使用内置的上下文 USERENV 来检索有关用户会话自身的信息。在表 9-13 中是 USERENV 上下文中一些更为常见的参数。

表 9-13　常见的 USERENV 上下文参数

参　数	返　回　值
CURRENT_SCHEMA	会话的默认模式
DB_NAME	在初始参数 DB_NAME 中指定的数据库名称
HOST	用户连接的主机名称
IP_ADDRESS	用户连接的 IP 地址
OS_USER	初始化数据库会话的操作系统账户
SESSION_USER	经过验证的数据库用户名

例如，下面对 SYS_CONTEXT 的调用将检索数据库会话的用户名和 IP_ADDRESS：

```
declare
     username        varchar2(30);
     ip_addr         varchar2(30);
```

```
begin
     username := SYS_CONTEXT('USERENV','SESSION_USER');
     ip_addr := SYS_CONTEXT('USERENV','IP_ADDRESS');
     -- other processing here
end;
```

类似地，可以在 SQL select 语句中使用 SYS_CONTEXT 函数：

```
SQL> select SYS_CONTEXT('USERENV','SESSION_USER') username from dual;
USERNAME
-------------------------
KSHELTON
```

使用 USERENV 上下文和数据库中授权信息的一些组合，可以使用 DBMS_SESSION.SET_ CONTEXT，将值赋予所创建的应用程序上下文中的参数：

```
dbms_session.set_context('HR_SECURITY','SEC_LEVEL','HIGH');
```

在该示例中，应用程序上下文变量 SEC_LEVEL 在 HR_SECURITY 上下文中设置为 HIGH。可以根据大量条件来分配该值，包括根据用户 ID 来分配安全级别的映射表。

为了确保针对每个会话设置上下文变量，可以使用登录触发器来调用与该上下文关联的过程。前面提及，在分配的程序包中只可以设置或改变上下文中的变量。下面是一个示例登录触发器，该触发器调用过程以建立上下文：

```
create or replace trigger vpd.set_security_parameters
   after logon on database
begin
   vpd.emp_access.set_security_parameters;
end;
```

在该示例中，过程 SET_SECURITY_PARAMETERS 将需要调用 DBMS_SESSION.SET_ CONTEXT。

在 Oracle Enterprise Manager 中，可以使用 Policy Manager 来建立上下文和策略组，如图 9-8 所示。

图 9-8 Oracle Policy Manager

2. 安全策略实现

基础结构到位之后，就可以建立安全环境，下一步就是定义用于生成谓词的函数，这些谓词将附加到受保护表的每个 select 语句或 DML 命令。用于实现谓词生成的函数有 2 个参数：受保护对象的拥有者、拥有者模式中对象的名称。一个函数只可以处理一种操作类型的谓词生成，例如 select，或者可以适用于所有的 DML 命令，这取决于该函数如何关联受保护的表。下面的示例显示了包含两个函数的程序包主体：一个函数将用于控制 select 语句中的访问，另一个函数将用于任何其他的 DML 语句：

```
create or replace package body get_predicates is
   function emp_select_restrict(owner varchar2, object_name varchar2)
         return varchar2 is
```

```
      ret_predicate    varchar2(1000);  -- part of WHERE clause
   begin
      -- only allow certain employees to see rows in the table
      -- . . . check context variables and build predicate
      return ret_predicate;
   end emp_select_restrict;
   function emp_dml_restrict(owner varchar2, object_name varchar2)
         return varchar2 is
      ret_predicate    varchar2(1000);  -- part of WHERE clause
   begin
      -- only allow certain employees to make changes to the table
      -- . . . check context variables and build predicate
      return ret_predicate;
   end emp_dml_restrict;
end; -- package body
```

每个函数返回一个包含表达式的字符串，该表达式被添加到 select 语句或 DML 命令的 where 子句。用户或应用程序永远不会看到这个 WHERE 子句的值，它在解析时自动添加到该命令。

开发人员必须确保这些函数总是返回有效的表达式。否则，任何对受保护表的访问总会失败，如同下面的示例所示：

```
SQL> select * from hr.employees;
select * from hr.employees
                *
ERROR at line 1:
ORA-28113: policy predicate has error
```

错误消息不会表明谓词是什么，并且所有的用户都无法访问表，直到修正谓词函数。本章后面将介绍关于如何调试谓词函数的技巧。

3. 使用 DBMS_RLS

内置的程序包 DBMS_RLS 包含大量子程序，DBA 使用这些子程序维护与表、视图和同义词关联的安全策略。表 9-14 列出了程序包 DBMS_RLS 中的子程序。任何需要创建和管理策略的用户都必须被授予程序包 SYS.DBMS_RLS 上的 EXECUTE 权限。

表 9-14　DBMS_RLS 程序包的子程序

子　程　序	说　　明
ADD_POLICY	将细粒度的访问控制策略添加到对象
DROP_POLICY	删除对象中的 FGAC 策略
REFRESH_POLICY	重新解析与策略关联的、缓存的所有语句
ENABLE_POLICY	启用或禁用 FGAC 策略
CREATE_POLICY_GROUP	创建策略组
ADD_GROUPED_POLICY	将策略添加到策略组
ADD_POLICY_CONTEXT	添加当前应用程序的上下文
DELETE_POLICY_GROUP	删除策略组
DROP_GROUPED_POLICY	从策略组中删除一个策略

DROP_POLICY_CONTEXT	删除活动应用程序的上下文
ENABLE_GROUPED_POLICY	启用或禁用组策略
DISABLE_GROUPED_POLICY	禁用组策略
REFRESH_GROUPED_POLICY	重新解析与策略组关联的、缓存的所有语句

本章将介绍最常用的子程序 ADD_POLICY 和 DROP_POLICY。ADD_POLICY 的语法如下：

```
DBMS_RLS.ADD_POLICY
(
     object_schema          IN varchar2 null,
     object_name            IN varchar2,
     policy_name            IN varchar2,
     function_schema        IN varchar2 null,
     policy_function        IN varchar2,
     statement_types        IN varchar2 null,
     update_check           IN boolean false,
     enable                   IN boolean true,
     static_policy          IN boolean false,
     policy_type            IN binary_integer null,
     long_predicate         IN in Boolean false,
     sec_relevant_cols     IN varchar2,
     sec_relevant_cols_opt  IN binary_integer null
);
```

注意，其中一些参数具有 BOOLEAN 默认值，并且较少使用的参数都在参数列表的末端。对于绝大多数情况来说，这就使对 DBMS_RLS.ADD_POLICY 的特定调用的语法更易于编写和理解。表 9-15 提供了每个参数的说明和用法。

表 9-15　DBMS_RLS.ADD_POLICY 的参数

参　　数	说　　明
object_schema	包含由策略保护的表、视图或同义词的模式。如果该值是 NULL，则使用调用过程的用户的模式
object_name	由策略保护的表、视图或同义词的名称
policy_name	添加到该对象的策略的名称。对于受保护的每个对象，该策略名必须唯一
function_schema	拥有策略函数的模式；如果该值为 NULL，则使用调用过程的用户的模式
policy_function	函数名称，该函数为针对 object_name 的策略生成谓词。如果函数是程序包的一部分，则在此处必须也指定程序包名，用于限定策略函数名
statement_types	应用策略的语句类型。允许的值(以逗号分隔)可以是 SELECT、INSERT、UPDATE、DELETE 和 INDEX 的任意组合。默认情况下，除了 INDEX 之外的所有类型都适用
update_check	对于 INSERT 或 UPDATE 类型，该参数是可选项，它默认为 FALSE。如果该参数为 TRUE，则在检查 SELECT 或 DELETE 操作时，则对 INSERT 或 UPDATE 语句也要检查该策略
enable	该参数默认为 TRUE，表明添加该策略时是否启用它
static_policy	如果该参数为 TRUE，该策略为任何访问该对象的人产生相同的谓词字符串，除了 SYS 用户或具有 EXEMPT ACCESS POLICY 权限的任何用户。该参数的默认值为 FALSE
policy_type	如果该值不是 NULL，则覆盖 static_policy。可允许的值是 STATIC、SHARED_STATIC、CONTEXT_SENSITIVE、SHARED_CONTEXT_SENSITIVE 和 DYNAMIC
long_predicate	该参数默认为 FALSE。如果它为 TRUE，谓词字符串最多可为 32K 字节长。否则，限制为 4000 字节
sec_relevant_cols	实施列级别的 VPD，这是 Oracle 10g 的新增内容。只应用于表和视图。在列表中指定受保护的列，使用逗号或空格作为分隔符。该策略只应用于指定的敏感列位于查询或 DML 语句中时。默认情况下，所有的列都是受保护的
sec_relevant_cols_opt	允许在列级别 VPD 过滤查询中的行仍然出现在结果集中，敏感列返回 NULL 值。该参数的默认值为 NULL；如果不是默认值，则必须指定 DBMS_RLS.ALL_ROWS，用于显示敏感列为 NULL 的所有列

如果不介意用户是否会看到行的部分内容，其实只是看不到包含机密信息的列，例如 Social Security Number(社会保障号)或薪水情况，则使用参数 sec_relevant_cols 非常便利。在本章后面的示例中，将根据定义的第一个安全策略对公司大多数雇员过滤敏感数据。

在下面的示例中，将名为 EMP_SELECT_RESTRICT 的策略应用于表 HR.EMPLOYEES。模式 VPD 拥有策略函数 get_predicates.emp_select_restrict。该策略显式地应用于表上的 SELECT 语句。然而，将 UPDATE_CHECK 设置为 TRUE 时，在更新行或将行插入到表中时，也会检查 update 或 delete 命令：

```
dbms_rls.add_policy (
         object_schema =>     'HR',
         object_name =>       'EMPLOYEES',
         policy_name =>       'EMP_SELECT_RESTRICT',
         function_schema => 'VPD',
         policy_function => 'get_predicates.emp_select_restrict',
         statement_types => 'SELECT',
         update_check =>      TRUE,
         enable =>             TRUE
);
```

因为没有设置 static_policy，它默认为 FALSE，这意味着该策略是动态的，并且在每次解析 select 语句时检查该策略。这是在 Oracle Database 10g 之前唯一可用的行为。

使用子程序 ENABLE_POLICY 是临时禁用策略的一种简单方法，并且不需要在以后将策略重新绑定到表：

```
dbms_rls.enable_policy(
         object_schema =>   'HR',
         object_name =>     'EMPLOYEES',
         policy_name =>     'EMP_SELECT_RESTRICT',
         enable =>          FALSE
);
```

如果为相同的对象指定多个策略，则在每个谓词之间添加 AND 条件。如果需要在多个策略的谓词之间使用 OR 条件，则很可能需要修订策略。每个策略的逻辑需要整合到一个策略中，该策略在谓词的每个部分之间具有 OR 条件。

4. 创建 VPD

本节将从头到尾遍历 VPD 的完整实现，该示例依赖于和 Oracle Database 10g 和 11g 一起安装的示例模式。具体来说，将实现 HR.EMPLOYEES 表上的 FGAC 策略，用于根据经理地位和雇员的部门编号来限制访问。如果是雇员，可以在 HR.EMPLOYEES 中看到自己的行；如果是经理，则可以看到他直接管理的所有雇员的行。

提示：

如果没有在数据库中安装示例模式，可以使用$ORACLE_HOME/demo/schema 中的脚本创建它们。

示例模式到位之后，需要在数据库中创建一些用户，他们想要查看表 HR.EMPLOYEES 中的行：

```
create user smavris identified by smavris702;
grant connect, resource to smavris;
create user dgrant identified by dgrant507;
grant connect, resource to dgrant;
create user kmourgos identified by kmourgos622;
grant connect, resource to kmourgos;
```

用户 KMOURGOS 是所有存货管理员的经理，而 DGRANT 是 KMOURGOS 的一个雇员。用户 SMAVRIS 是公司的 HR_REP。

在下面的 3 个步骤中，将 HR.EMPLOYEES 表上的 SELECT 权限授予数据库中的每个人，并且将创建一个

查找表，将雇员 ID 号映射到他们的数据库账户。设置用户会话上下文变量的过程将使用该表把雇员 ID 号赋予上下文变量，在策略函数中将使用该上下文变量来生成谓词。

```
grant select on hr.employees to public;
create table hr.emp_login_map (employee_id, login_acct)
   as select employee_id, email from hr.employees;
grant select on hr.emp_login_map to public;
```

接下来，创建称为 VPD 的用户账户，该账户具有创建上下文和维护策略函数的权限：

```
create user vpd identified by vpd439;
grant connect, resource, create any context, create public synonym to vpd;
```

连接到 VPD 模式，创建称为 HR_SECURITY 的上下文，并且定义用于设置应用程序上下文的程序包和过程：

```
connect vpd/vpd439@dw;
create context hr_security using vpd.emp_access;
create or replace package vpd.emp_access as
    procedure set_security_parameters;
end;
```

记住，程序包 VPD.EMP_ACCESS 中的过程是可以设置上下文变量的唯一过程。VPD.EMP_ACCESS 的程序包主体如下：

```
create or replace package body vpd.emp_access is
--
-- At user login, run set_security_parameters to
-- retrieve the user login name, which corresponds to the EMAIL
-- column in the table HR.EMPLOYEES.
--
-- context USERENV is pre-defined for user characteristics such
-- as username, IP address from which the connection is made,
-- and so forth.
--
-- for this procedure, we are only using SESSION_USER
-- from the USERENV context.
--
   procedure set_security_parameters is
      emp_id_num      number;
      emp_login       varchar2(50);
   begin
      -- database username corresponds to email address in HR.EMPLOYEES
      emp_login := sys_context('USERENV','SESSION_USER');
      dbms_session.set_context('HR_SECURITY','USERNAME',emp_login);
      -- get employee id number, so manager rights can be established
      -- but don't bomb out other DB users who are not in the
      -- EMPLOYEES table
      begin
         select employee_id into emp_id_num
            from hr.emp_login_map where login_acct = emp_login;
         dbms_session.set_context('HR_SECURITY','EMP_ID',emp_id_num);
      exception
         when no_data_found then
            dbms_session.set_context('HR_SECURITY','EMP_ID',0);
      end;
      -- Future queries will restrict rows based on emp_id
   end; -- procedure
end; -- package body
```

有关该过程还需要注意一些事情。我们通过查询 USERENV 上下文来检索用户的模式，默认情况下为所有用户自动启用该上下文。然后，将该模式赋给新创建的上下文 HR_SECURITY 中的变量 USERNAME。通过在映射表 HR.EMP_LOGIN_MAP 中进行查找来确定另一个 HR_SECURITY 上下文变量 EMP_ID。不希望该过程在已登录的用户不在映射表中时中断，并且显示一个错误。相反，分配 EMP_ID 的值为 0，结果就是在策略函数中生成谓词时没有对 HR.EMPLOYEES 表的任何访问。

在下面的步骤中，授予数据库中每个人程序包上的 EXECUTE 权限，并且为其创建一个同义词，从而在每次需要调用它时节省一些击键次数：

```
grant execute on vpd.emp_access to PUBLIC;
```

```
create public synonym emp_access for vpd.emp_access;
```

为了确保在每个用户登录时为其定义上下文，我们将以 SYSTEM 身份连接到数据库，并且创建一个登录触发器，用于在上下文中设置变量：

```
connect system/nolongermanager@dw as sysdba;
create or replace trigger vpd.set_security_parameters
   after logon on database
begin
   vpd.emp_access.set_security_parameters;
end;
```

因为为每个连接到数据库的用户激活该触发器，所以如果不是为每个用户测试代码的话，则为每一类用户测试代码就极其重要！如果触发器失败并且显示一个错误，那么常规用户将无法登录。

到目前为止，已经有了定义的上下文、用来设置上下文变量的过程以及自动调用该过程的触发器。作为前面定义的 3 个用户中的一个进行登录，可以查询上下文的内容：

```
SQL> connect smavris/smavris702@dw
Connected.
SQL> select * from session_context;
NAMESPACE                ATTRIBUTE                VALUE
------------------------ ------------------------ --------------------
HR_SECURITY              USERNAME                 SMAVRIS
HR_SECURITY              EMP_ID                   203
2 rows selected.
```

注意当 SMAVRIS 尝试冒充另一个雇员时发生的情况：

```
SQL> begin
  2    dbms_session.set_context('HR_SECURITY','EMP_ID',100);
  3  end;
begin
*
ERROR at line 1:
ORA-01031: insufficient privileges
ORA-06512: at "SYS.DBMS_SESSION", line 94
ORA-06512: at line 2
```

只允许程序包 VPD.EMP_ACCESS 设置或改变上下文中的变量。

最后的步骤包括定义将生成谓词的过程，以及将其中的一个或多个过程赋给 HR.EMPLOYEES 表。作为用户 VPD，该用户已经拥有了上下文过程，接下来建立定义谓词的程序包：

```
connect vpd/vpd439@dw;
create or replace package vpd.get_predicates as
   -- note -- security function ALWAYS has two parameters,
   -- table owner name and table name
   function emp_select_restrict
       (owner varchar2, object_name varchar2) return varchar2;
   -- other functions can be written here for INSERT, DELETE, and so forth.
end get_predicates;
create or replace package body vpd.get_predicates is
   function emp_select_restrict
      (owner varchar2, object_name varchar2) return varchar2 is
      ret_predicate    varchar2(1000);  -- part of WHERE clause
   begin
      -- only allow employee to see their row or immediate subordinates
      ret_predicate := 'EMPLOYEE_ID = ' ||
                          sys_context('HR_SECURITY','EMP_ID') ||
                          ' OR MANAGER_ID = ' ||
                          sys_context('HR_SECURITY','EMP_ID');
      return ret_predicate;
   end emp_select_restrict;
end; -- package body
```

一旦使用 DBMS_RLS 将该函数附加到表，将生成一个文本字符串，在每次访问表时将该文本字符串用于 WHERE 子句中。该字符串总是类似于：

```
EMPLOYEE_ID = 124 OR MANAGER_ID = 124
```

和建立上下文环境的程序包一样，需要允许用户访问该程序包：

```
grant execute on vpd.get_predicates to PUBLIC;
create public synonym get_predicates for vpd.get_predicates;
```

最后(但并不是最不重要的方面)，使用 DBMS_RLS.ADD_POLICY 过程将策略函数附加 到表：

```
dbms_rls.add_policy (
         object_schema =>   'HR',
         object_name =>     'EMPLOYEES',
         policy_name =>     'EMP_SELECT_RESTRICT',
         function_schema => 'VPD',
         policy_function => 'get_predicates.emp_select_restrict',
         statement_types => 'SELECT',
         update_check =>    TRUE,
         enable =>          TRUE
);
```

雇员可以像前面一样访问HR.EMPLOYEES表,但他们将只能看到自己的行,以及为其工作的雇员的行(如果存在这种雇员的话)。作为KMOURGOS登录，尝试检索HR.EMPLOYEES表的所有行，但只可以看到KMOURGOS的行以及他直接管理的雇员的行：

```
SQL> connect kmourgos/kmourgos622@dw;
Connected.
SQL> select employee_id, first_name, last_name,
  2         email, job_id, salary, manager_id from hr.employees;
EMPLOYEE_ID FIRST_NAME LAST_NAME   EMAIL      JOB_ID      SALARY MANAGER_ID
----------- ---------- ----------   ---------- ---------- ------- ----------
        124   Kevin      Mourgos     KMOURGOS   ST_MAN       5800        100
        141   Trenna     Rajs          TRAJS      ST_CLERK     3500        124
        142   Curtis     Davies      CDAVIES    ST_CLERK     3100        124
        143   Randall    Matos       RMATOS     ST_CLERK     2600        124
        144   Peter      Vargas      PVARGAS    ST_CLERK     2500        124
        196   Alana      Walsh       AWALSH     SH_CLERK     3100        124
        197   Kevin      Feeney      KFEENEY    SH_CLERK     3000        124
        198   Donald     OConnell    DOCONNEL   SH_CLERK     2600        124
        199   Douglas    Grant       DGRANT     SH_CLERK     2600        124
9 rows selected.
```

对于用户DGRANT，情况则完全不同：

```
SQL> connect dgrant/dgrant507@dw;
Connected.
SQL> select employee_id, first_name, last_name,
  2         email, job_id, salary, manager_id from hr.employees;
EMPLOYEE_ID FIRST_NAME LAST_NAME EMAIL     JOB_ID     SALARY   MANAGER_ID
----------- ---------- --------- -------- --------- ------- ----------
        199   Douglas    Grant      DGRANT    SH_CLERK    2600        124
1 row selected.
```

DGRANT 开始只能看到自己的行，因为他不管理公司内的其他任何人。

在 SMAVRIS 的情况中，可以通过查询看到类似的结果：

```
SQL> connect smavris/smavris702@dw;
Connected.
SQL> select employee_id, first_name, last_name,
  2         email, job_id, salary, manager_id from hr.employees;
EMPLOYEE_ID FIRST_NAME LAST_NAME    EMAIL      JOB_ID      SALARY MANAGER_ID
----------- ---------- ------------ ---------- ---------- ------- ----------
        203 Susan      Mavris       SMAVRIS    HR_REP        6500        101
1 row selected.
```

但是需要注意，SMAVRIS 属于人力资源(HR)部门，所以应该能够看到表中的所有行。此外，SMAVRIS 应该是可以查看所有雇员薪水信息的唯一的人。因此，需要改变策略函数，为 SMAVRIS 和 HR 部门中的其他雇员提供对 HR.EMPLOYEES 表的完全访问。此外，可以使用策略赋值中列级别的约束来返回相同数量的行，但其中的敏感数据作为 NULL 值返回。

为了方便 HR 部门的雇员对 HR.EMPLOYEES 表的访问，首先需要改变映射表，使其包括 JOB_ID 列。如果 JOB_ID 列的值是 HR_REP，该雇员就属于 HR 部门。首先禁止该策略生效，并且创建新的映射表：

```
SQL> begin
  2    dbms_rls.enable_policy(
  3          object_schema =>   'HR',
  4          object_name =>     'EMPLOYEES',
  5          policy_name =>     'EMP_SELECT_RESTRICT',
  6          enable =>          FALSE
  7       );
  8  end;
PL/SQL procedure successfully completed.
SQL> drop table hr.emp_login_map;
Table dropped.
SQL> create table hr.emp_login_map (employee_id, login_acct, job_id)
  2    as select employee_id, email, job_id from hr.employees;
Table created.
SQL> grant select on hr.emp_login_map to public;
Grant succeeded.
```

用于设置上下文变量的过程 VPD.EMP_ACCESS 需要添加另一个上下文变量，该上下文变量表明访问表的用户的安全级别。改变 SELECT 语句，并且对 DBMS_SESSION.SET_CONTEXT 进行另一次调用，如下所示：

```
. . .
      emp_job_id      varchar2(50);
. . .
         select employee_id, job_id into emp_id_num, emp_job_id
            from hr.emp_login_map where login_acct = emp_login;
         dbms_session.set_context('HR_SECURITY','SEC_LEVEL',
             case emp_job_id when 'HR_REP' then 'HIGH' else 'NORMAL' end );
. . .
```

当雇员具有 HR_REP 的职位时，将上下文变量 SEC_LEVEL 设置为 HIGH 而不是 NORMAL。在策略函数中，需要检查这个新的条件，如下所示：

```
create or replace package body vpd.get_predicates is
   function emp_select_restrict
      (owner varchar2, object_name varchar2) return varchar2 is
      ret_predicate    varchar2(1000);  -- part of WHERE clause
   begin
      -- only allow employee to see their row or immediate subordinates,
      -- unless they have high security clearance
      if sys_context('HR_SECURITY','SEC_LEVEL') = 'HIGH' then
         ret_predicate := '';  -- no restrictions in WHERE clause
      else
         ret_predicate := 'EMPLOYEE_ID = ' ||
                          sys_context('HR_SECURITY','EMP_ID') ||
                          ' OR MANAGER_ID = ' ||
                          sys_context('HR_SECURITY','EMP_ID');
      end if;
      return ret_predicate;
   end emp_select_restrict;
end; -- package body
```

因为策略是动态的，因此每次执行 SELECT 语句时都生成谓词，从而不需要进行策略刷新。当用户 SMAVRIS，即 HR 部门的代表，现在运行查询时，可以看到 HR.EMPLOYEES 表中的所有行：

```
SQL> connect smavris/smavris702@dw;
Connected.
SQL> select employee_id, first_name, last_name,
  2         email, job_id, salary, manager_id from hr.employees;
EMPLOYEE_ID FIRST_NAME  LAST_NAME   EMAIL      JOB_ID       SALARY MANAGER_ID
----------- ----------- ---------   --------- ----------  ------- ----------
        100 Steven      King          SKING      AD_PRES      24000
        101 Neena       Kochhar     NKOCHHAR   AD_VP      17000         100
. . .
        204 Hermann     Baer          HBAER      PR_REP       10000         101
        205 Shelley     Higgins     SHIGGINS   AC_MGR      12000         101
        206 William     Gietz       WGIETZ     AC_ACCOUNT  8300         205
107 rows selected.
```

SMAVRIS 在 HR_SECURITY 上下文中的安全级别是 HIGH：

```
SQL> connect smavris/smavris702
Connected.
SQL> select sys_context('HR_SECURITY','SEC_LEVEL') from dual;
SYS_CONTEXT('HR_SECURITY','SEC_LEVEL')
----------------------------------------------------------------
HIGH
```

```
SQL>
```

然而，DGRANT 仍然只可以看到表中自己的行，因为他在 HR_SECURITY 上下文中的安全级别是 NORMAL：

```
SQL> connect dgrant/dgrant507@dw;
Connected.
SQL> select employee_id, first_name, last_name,
  2         email, job_id, salary, manager_id from hr.employees;
EMPLOYEE_ID FIRST_NAME LAST_NAME   EMAIL      JOB_ID      SALARY MANAGER_ID
----------- ---------- ---------- --------- ---------- ------- ----------
        199 Douglas      Grant       DGRANT     SH_CLERK     2600      124
1 row selected.
SQL> select sys_context('HR_SECURITY','SEC_LEVEL') from dual;
SYS_CONTEXT('HR_SECURITY','SEC_LEVEL')
--------------------------------------------------------------
NORMAL
```

为了实施只有 HR 雇员可以看到薪水信息的需求，需要稍微修改策略函数，启用具有列级别约束的策略：

```
dbms_rls.add_policy (
         object_schema =>   'HR',
         object_name =>     'EMPLOYEES',
         policy_name =>     'EMP_SELECT_RESTRICT',
         function_schema => 'VPD',
         policy_function => 'get_predicates.emp_select_restrict',
         statement_types => 'SELECT',
         update_check =>    TRUE,
```

```
        enable =>          TRUE,
        sec_relevant_cols => 'SALARY',
        sec_relevant_cols_opt => dbms_rls.all_rows
);
```

最后一个参数 SEC_RELEVANT_COLS_OPT 指定程序包常量 DBMS_RLS.ALL_ROWS，用于表明仍然希望看到查询结果中的所有行，但是具有返回 NULL 值的相关列(在当前情况中是 SALARY)。否则，将不会看到包含 SALARY 列的查询中的任何行。

5. 调试 VPD 策略

即使没有获得“ORA-28113: policy predicate has error”或“ORA-00936: missing expression”，查看语句解析时生成的实际谓词也会非常有用。有两种方法可调试谓词，这两种方法各有其优点和缺点。

第一种方法使用动态性能视图 V$SQLAREA 和 V$VPD_POLICY。顾名思义，V$SQLAREA 包含当前位于共享池中的 SQL 语句，以及当前的执行统计。视图 V$VPD_POLICY 列出当前在数据库中实施的所有策略，以及谓词。如同下面的示例所示，连接 2 个表，可以提供一些信息，我们需要通过这些信息来帮助调试在查询结果中遇到的任何问题：

```
SQL> select s.sql_text, v.object_name, v.policy, v.predicate
  2     from v$sqlarea s, v$vpd_policy v
  3     where s.hash_value = v.sql_hash;
SQL_TEXT                  OBJECT_NAM    POLICY                  PREDICATE
```

```
---------------------- ----------    ------------------- -----------------
select employee_id, first EMPLOYEES  EMP_SELECT_RESTRICT EMPLOYEE_ID = 199
_name, last_name, email,                                       OR MANAGER_ID = 199
job_id, salary, manager_i
d from hr.employees
select employee_id, first EMPLOYEES  EMP_SELECT_RESTRICT
_name, last_name, email,
job_id, salary, manager_i
d from hr.employees
SQL>
```

如果在此查询中添加一个到 V$SESSION 的连接，则可以识别哪个用户正在运行 SQL。这在第二个 SQL 语句中尤其重要，此 SQL 语句没有应用谓词，因此，我们能够推断的只是 HR 雇员之一运行此查询。该方法的不足之处在于：如果数据库非常忙，则在有机会运行该查询之前，可能由于其他的 SQL 命令而在共享池中刷新了当前 SQL 命令。

另一个方法使用 alter session 命令来生成纯文本的跟踪文件，该文件包含前面查询的许多信息。下面是建立跟踪的命令：

```
SQL> begin
  2     dbms_rls.refresh_policy;
  3  end;
PL/SQL procedure successfully completed.
SQL> alter session set events '10730 trace name context forever, level 12';
Session altered.
```

为跟踪 RLS 策略谓词定义事件 10730。其他可以跟踪的常见事件是用于会话登录/退出的 10029 和 10030、用于跟踪位图索引访问的 10710、用于模仿重做日志的写入错误的 10253，以及其他事件。一旦改变会话，用户 DGRANT 运行其查询：

```
SQL> select employee_id, first_name, last_name,
  2      email, job_id, salary, manager_id from hr.employees;
EMPLOYEE_ID FIRST_NAME  LAST_NAME   EMAIL      JOB_ID      SALARY MANAGER_ID
----------- ----------- --------- ---------- ---------- ------- ----------
        199   Douglas      Grant       DGRANT     SH_CLERK      2600        124

1 row selected.
```

下面查看跟踪文件底部的内容，该跟踪文件位于由初始参数 USER_DUMP_DEST 指定的目录中(在 Oracle Database 11g 中由 DIAGNOSTIC_DEST 参数指定)：

```
Trace file
/u01/app/oracle/diag/rdbms/dw/dw/trace/dw_ora_31128.trc
Oracle Database 11g Enterprise Edition
                    Release 11.1.0.6.0 ¨C Productio
With the Partitioning, OLAP, Data Mining and
                    Real Application Testing options
ORACLE_HOME = /u01/app/oracle/product/11.1.0/db_1
System name:    Linux
Node name:      dw
Release:        2.6.9-55.0.2.0.1.EL
Version:        #1 Mon Jun 25 14:24:38 PDT 2007
Machine:        i686
Instance name: dw
Redo thread mounted by this instance: 1
Oracle process number: 40
Unix process pid: 31128, image: oracle@dw (TNS V1-V3)
*** 2007-08-12 12:48:37.852
*** SESSION ID:(120.9389) 2007-08-12 12:48:37.852
*** CLIENT ID:() 2007-08-12 12:48:37.852
*** SERVICE NAME:(SYS$USERS) 2007-08-12 12:48:37.852
*** MODULE NAME:(SQL*Plus) 2007-08-12 12:48:37.852
*** ACTION NAME:() 2007-08-12 12:48:37.852
------------------------------------------------------------
Logon user     : DGRANT
Table/View     : HR.EMPLOYEES
Policy name    : EMP_SELECT_RESTRICT
Policy function: VPD.GET_PREDICATES.EMP_SELECT_RESTRICT
RLS view :
SELECT  "EMPLOYEE_ID","FIRST_NAME","LAST_NAME",
"EMAIL","PHONE_NUMBER",
"HIRE_DATE","JOB_ID","SALARY","COMMISSION_PCT","MANAGER_ID",
"DEPARTMENT_ID" FROM "HR"."EMPLOYEES"
```

```
"EMPLOYEES" WHERE (EMPLOYEE_ID = 199 OR MANAGER_ID = 199)
-------------------------------------------------------------
```

用户的初始 SQL 语句以及附加的谓词都清楚地显示在跟踪文件中。使用这种方法的不利方面在于，虽然用户也许能够访问动态性能视图，但开发人员通常可能没有对服务器自身的用户转储目录的访问权。因此，在尝试调整谓词问题时可能需要 DBA 的参与。

确保在完成调试时关闭跟踪，这样可以减少与跟踪操作关联的系统开销和磁盘空间(否则只能退出系统)：

```
SQL> alter session set events '10730 trace name context off';
Session altered.
```

9.4 审计

Oracle 使用大量不同的审计方法来监控使用何种权限，以及访问哪些对象。审计不会防止使用这些权

限，但可以提供有用的信息，用于揭示权限的滥用和误用。

表 9-16 中总结了 Oracle 数据库中不同类型的审计。

表 9-16 审 计 类 型

审 计 类 型	说 明
语句审计	按照语句类型审计 SQL 语句，而不论访问何种特定的模式对象。也可以在数据库中指定一个或多个用户，针对特定的语句审计这些用户
权限审计	审计系统权限，例如 CREATE TABLE 或 ALTER INDEX。和语句审计一样，权限审计可以指定一个或多个特定的用户作为审计的目标
模式对象审计	审计特定模式对象上运行的特定语句(例如，DEPARTMENTS 表上的 UPDATE 语句)。模式对象审计总是应用于数据库中的所有用户
细粒度的审计	根据访问对象的内容来审计表访问和权限。使用程序包 DBMS_FGA 来建立特定表上的策略

下面几节介绍 DBA 如何管理系统和对象权限使用的审计。当需要一定的粒度时，DBA 可以使用细粒度的审计来监控对表中某些行或列的访问，而不仅仅是是否访问表。

9.4.1 审计位置

审计记录可以发送到 SYS.AUD$数据库表或操作系统文件。为了启用审计并指定记录审计记录的位置，将初始参数 AUDIT_TRAIL 设置为如下几个值之一：

参 数 值	动 作
NONE, FALSE	禁用审计
OS	启用审计，将审计记录发送到操作系统文件
DB, TRUE	启用审计，将审计记录发送到 SYS.AUD$表
DB_EXTENDED	启用审计，将审计记录发送到 SYS.AUD$表，并在 CLOB 列 SQLBIND 和 SQLTEXT 中记录额外的信息
XML	启用审计，以 XML 格式写所有审计记录
EXTENDED	启用审计，在审计跟踪中记录所有列，包括 SqlText 和 SqlBind 的值

参数 AUDIT_TRAIL 不是动态的，为了使 AUDIT_TRAIL 参数中的改动生效，必须关闭数据库并重新启动。在对 SYS.AUD$表进行审计时，应该注意监控该表的大小，以避免影响 SYS 表空间中其他对象的空间需求。推荐周期性归档 SYS.AUD$中的行，并且截取该表。Oracle 提供了角色 DELETE_CATALOG_ROLE，和批处理作业中的特殊账户一起使用，用于归档和截取审计表。

9.4.2 语句审计

所有类型的审计都使用 audit 命令来打开审计，使用 noaudit 命令来关闭审计。对于语句审计，audit 命令的格式看起来如下所示：

```
AUDIT sql_statement_clause BY {SESSION | ACCESS}
    WHENEVER [NOT] SUCCESSFUL;
```

sql_statement_clause 包含很多条不同的信息，例如希望审计的 SQL 语句类型以及审计什 么人。

此外，希望在每次动作发生时都对其进行审计(by access)或者只审计一次(by session)。默认是 by session。

有时希望审计成功的动作：没有生成错误消息的语句。对于这些语句，添加 whenever successful。而有时只关心使用审计语句的命令是否失败，失败原因是权限违犯、用完表空间中的空间还是语法错误。对于这些情况，使用 whenever not successful。

对于大多数类别的审计方法，如果确实希望审计所有类型的表访问或某个用户的任何权限，则可以指定 all 而不是单个的语句类型或对象。

表 9-17 列出了可以审计的语句类型，并且在每个类别中包含了相关语句的简要描述。如果指定 all，则审计该列表中的任何语句。然而，表 9-18 中的语句类型在启用审计时不属于 all 类别；必须在 audit 命令中显式地指定它们。

表 9-17 包括在 ALL 类别中的可审计语句

语 句 选 项	SQL 操作
ALTER SYSTEM	所有 ALTER SYSTEM 选项，例如，动态改变实例参数，切换到下一个日志文件组，以及终止用户会话
CLUSTER	CREATE、ALTER、DROP 或 TRUNCATE 集群
CONTEXT	CREATE CONTEXT 或 DROP CONTEXT
DATABASE LINK	CREATE 或 DROP 数据库链接
DIMENSION	CREATE、ALTER 或 DROP 维数
DIRECTORY	CREATE 或 DROP 目录
INDEX	CREATE、ALTER 或 DROP 索引
MATERIALIZED VIEW	CREATE、ALTER 或 DROP 物化视图
NOT EXISTS	由于不存在的引用对象而造成的 SQL 语句的失败
PROCEDURE	CREATE 或 DROP FUNCTION、LIBRARY、PACKAGE、PACKAGE BODY 或 PROCEDURE
PROFILE	CREATE、ALTER 或 DROP 配置文件
PUBLIC DATABASE LINK	CREATE 或 DROP 公有数据库链接
PUBLIC SYNONYM	CREATE 或 DROP 公有同义词
ROLE	CREATE、ALTER、DROP 或 SET 角色
ROLLBACK SEGMENT	CREATE、ALTER 或 DROP 回滚段
SEQUENCE	CREATE 或 DROP 序列
SESSION	登录和退出
SYNONYM	CREATE 或 DROP 同义词
SYSTEM AUDIT	系统权限的 AUDIT 或 NOAUDIT
SYSTEM GRANT	GRANT 或 REVOKE 系统权限和角色
TABLE	CREATE、DROP 或 TRUNCATE 表
TABLESPACE	CREATE、ALTER 或 DROP 表空间
TRIGGER	CREATE、ALTER(启用/禁用)、DROP 触发器；具有 ENABLE ALL TRIGGERS 或 DISABLE ALL TRIGGERS 的 ALTER TABLE
TYPE	CREATE、ALTER 和 DROP 类型以及类型主体

USER	CREATE、ALTER 或 DROP 用户
VIEW	CREATE 或 DROP 视图

表 9-18　显式指定的语句类型

语 句 选 项	SQL 操 作
ALTER SEQUENCE	任何 ALTER SEQUENCE 命令
ALTER TABLE	任何 ALTER TABLE 命令
COMMENT TABLE	添加注释到表、视图、物化视图或它们中的任何列
DELETE TABLE	删除表或视图中的行
EXECUTE PROCEDURE	执行程序包中的过程、函数或任何变量或游标
GRANT DIRECTORY	GRANT 或 REVOKE DIRECTORY 对象上的权限
GRANT PROCEDURE	GRANT 或 REVOKE 过程、函数或程序包上的权限
GRANT SEQUENCE	GRANT 或 REVOKE 序列上的权限
GRANT TABLE	GRANT 或 REVOKE 表、视图或物化视图上的权限
GRANT TYPE	GRANT 或 REVOKE TYPE 上的权限
INSERT TABLE	INSERT INTO 表或视图
LOCK TABLE	表或视图上的 LOCK TABLE 命令
SELECT SEQUENCE	引用序列的 CURRVAL 或 NEXTVAL 的任何命令
SELECT TABLE	SELECT FROM 表、视图或物化视图
UPDATE TABLE	在表或视图上执行 UPDATE

一些示例可以帮助读者更清楚地了解所有这些选项。在示例数据库中，用户 KSHELTON 具有 HR 模式和其他模式中所有表上的权限。允许 KSHELTON 创建其中一些表上的索引，但如果有一些与执行计划改动相关的性能问题，则需要知道何时创建这些索引。可以使用如下命令审计 KSHELTON 创建的索引：

```
SQL> audit index by kshelton;
Audit succeeded.
```

后面的某一天，KSHELTON 在 HR.JOBS 表上创建了一个索引：

```
SQL> create index job_title_idx on hr.jobs(job_title);
Index created.
```

检查数据字典视图 DBA_AUDIT_TRAIL 中的审计跟踪，可以看到 KSHELTON 实际上在 8 月 12 日的 5:15 P.M. 创建了索引：

```
SQL> select username, to_char(timestamp,'MM/DD/YY HH24:MI') Timestamp,
  2      obj_name, action_name, sql_text from dba_audit_trail
  3  where username = 'KSHELTON';
USERNAME   TIMESTAMP        OBJ_NAME        ACTION_NAME     SQL_TEXT
---------- -------------- --------------- --------------- -----------------
KSHELTON   08/12/07 17:15 JOB_TITLE_IDX  CREATE INDEX   create index hr.
                                                                job_title_idx on
                                                                hr.jobs(job_title)
1 row selected.
```

注意：

从 Oracle Database 11g 开始，只有在初始参数 AUDIT_TRAIL 被设置为 DB_EXTENDED 时，才填充 DBA_AUDIT_TRAIL 中的列 SQL_TEXT 和 SQL_BIND。默认情况下，AUDIT_TRAIL 的值是 DB。

为了关闭 HR.JOBS 表上 KSHELTON 的审计，可以使用 noaudit 命令，如下所示：

```
SQL> noaudit index by kshelton;
Noaudit succeeded.
```

也可能希望按常规方式审计成功的和不成功的登录，这需要两个 audit 命令：

```
SQL> audit session whenever successful;
Audit succeeded.
SQL> audit session whenever not successful;
Audit succeeded.
```

回顾审计跟踪，可以看到用户 RJB 在 8 月 10 日的失败的登录尝试：

```
SQL> select username, to_char(timestamp,'MM/DD/YY HH24:MI') Timestamp,
  2       obj_name, returncode, action_name, sql_text from dba_audit_trail
  3  where action_name in ('LOGON','LOGOFF')
  4        and username in ('SCOTT','RJB','KSHELTON')
  5  order by timestamp desc;
USERNAME   TIMESTAMP        OBJ_NAME  RETURNCODE ACTION_NAME      SQL_TEXT
---------- -------------- --------- ---------- -------------- ----------
KSHELTON   08/12/07 17:04                    0 LOGON
SCOTT      08/12/07 16:10                    0 LOGOFF
RJB          08/12/07 11:35                  0 LOGON
RJB          08/12/07 11:35                  0 LOGON
RJB          08/11/07 22:51                  0 LOGON
RJB          08/11/07 22:51                  0 LOGOFF
RJB          08/11/07 21:55                  0 LOGOFF
RJB          08/11/07 21:40                  0 LOGOFF
RJB          08/10/07 22:52                  0 LOGOFF
RJB          08/10/07 22:52                  0 LOGOFF
RJB          08/10/07 22:52               1017 LOGON
RJB          08/10/07 12:23                  0 LOGOFF
SCOTT      08/03/07 04:18                    0 LOGOFF
13 rows selected.
```

RETURNCODE 代表 ORA 错误消息。ORA-1017 消息表明输入了不正确的密码。注意，如果仅对登录和退出感兴趣，可以改为使用 DBA_AUDIT_SESSION 视图。

语句审计也包括启动和关闭操作。虽然可以审计 SYS.AUD$表中的命令 shutdown immediate，但不可以审计 SYS.AUD$中的 startup 命令，因为必须在可以将行添加到这个表中之前启动数据库。对于这些情况，可以在初始参数 AUDIT_FILE_DEST 中指定的目录中查找，查看由系统管理员执行的启动操作的记录(默认情况下，此参数包含$ORACLE_HOME/admin/dw/adump)。下面是使用 startup 命令启动数据库时创建的文本文件：

```
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 ¨C Productio
With the Partitioning, OLAP, Data Mining
                        and Real Application Testing options
ORACLE_HOME = /u01/app/oracle/product/11.1.0/db_1
System name:    Linux
Node name:      dw
Release:        2.6.9-55.0.2.0.1.EL
Version:        #1 Mon Jun 25 14:24:38 PDT 2007
Machine:        i686
Instance name: dw
Redo thread mounted by this instance: 1
Oracle process number: 44
Unix process pid: 28962, image: oracle@dw (TNS V1-V3)
Sun Aug 12 11:57:36 2007
ACTION : 'CONNECT'
DATABASE USER: '/'
PRIVILEGE : SYSDBA
CLIENT USER: oracle
```

```
CLIENT TERMINAL: pts/2
STATUS: 0
```

在该示例中，由主机系统上作为 oracle 角色连接的用户启动数据库，并且该用户使用操作系统验证连接到实例。下一节将介绍额外的系统管理员审计问题。

9.4.3 权限审计

审计系统权限具有与语句审计相同的基本语法，但审计系统权限是在 *sql_statement_clause* 中，而不是在语句中，指定系统权限。

例如，可能希望将 ALTER TABLESPACE 权限授予所有的 DBA，但希望在发生这种情况时生成审计记录。启用对这种权限的审计的命令看起来类似于语句审计：

```
SQL> audit alter tablespace by access whenever successful;
Audit succeeded.
```

每次成功使用 ALTER TABLESPACE 权限时，都会将一行内容添加到 SYS.AUD$。

使用 SYSDBA 和 SYSOPER 权限或者以 SYS 用户连接到数据库的系统管理员可以利用特殊的审计。为了启用这种额外的审计级别，可以设置初始参数 AUDIT_SYS_OPERATIONS 为 TRUE。这种审计记录发送到与操作系统审计记录相同的位置。因此，这个位置是和操作系统相关的。当使用其中一种权限时执行的所有 SQL 语句，以及作为用户 SYS 执行的任何 SQL 语句，都会发送到操作系统审计位置。

9.4.4 模式对象审计

审计对各种模式对象的访问看起来类似于语句审计和权限审计：

```
AUDIT schema_object_clause BY {SESSION | ACCESS}
     WHENEVER [NOT] SUCCESSFUL;
```

schema_object_clause 指定对象访问的类型以及访问的对象。可以审计特定对象上 14 种不同的操作类型，表 9-19 中列出了这些操作。

表 9-19 对象审计选项

对 象 选 项	说　　明
ALTER	改变表、序列或物化视图
AUDIT	审计任何对象上的命令
COMMENT	添加注释到表、视图或物化视图
DELETE	从表、视图或物化视图中删除行
EXECUTE	执行过程、函数或程序包
FLASHBACK	执行表或视图上的闪回操作
GRANT	授予任何类型对象上的权限
INDEX	创建表或物化视图上的索引
INSERT	将行插入表、视图或物化视图中
LOCK	锁定表、视图或物化视图
READ	对 DIRECTORY 对象的内容执行读操作
RENAME	重命名表、视图或过程
SELECT	从表、视图、序列或物化视图中选择行
UPDATE	更新表、视图或物化视图

如果希望审计 HR.JOBS 表上的所有 insert 和 update 命令，而不管谁正在进行更新，则每次该动作发生时，都可以使用如下所示的 audit 命令：

```
SQL> audit insert, update on hr.jobs by access whenever successful;
Audit successful.
```

用户 KSHELTON 决定向 HR.JOBS 表添加两个新行：

```
SQL> insert into hr.jobs (job_id, job_title, min_salary, max_salary)
  2  values ('IN_CFO','Internet Chief Fun Officer', 7500, 50000);
1 row created.
SQL> insert into hr.jobs (job_id, job_title, min_salary, max_salary)
  2  values ('OE_VLD','Order Entry CC Validation', 5500, 20000);
1 row created.
```

查看 DBA_AUDIT_TRAIL 视图，可以看到 KSHELTON 会话中的两个 insert 命令：

```
USERNAME   TIMESTAMP      OWNER    OBJ_NAME   ACTION_NAME
SQL_TEXT
---------- -------------- -------- ---------- ---------------
----------------------------------------------------------------
KSHELTON   08/12/07 22:54 HR       JOBS       INSERT
insert into hr.jobs (job_id, job_title, min_salary, max_salary)
 values ('IN_CFO','Internet Chief Fun Officer', 7500, 50000);
KSHELTON   08/12/07 22:53 HR       JOBS       INSERT
insert into hr.jobs (job_id, job_title, min_salary, max_salary)
 values ('OE_VLD','Order Entry CC Validation', 5500, 20000);
KSHELTON   08/12/07 22:51                     LOGON
3 rows selected.
```

9.4.5 细粒度的审计

从 Oracle9i 开始，通过引入细粒度的对象审计，或称为 FGA，审计变得更为关注某个方面，并且更为精确。由称为 DBMS_FGA 的 PL/SQL 程序包实现 FGA。

使用标准的审计，可以轻松发现访问了哪些对象以及由谁访问，但无法知道访问了哪些行或列。细粒度的审计可解决这个问题，它不仅为需要访问的行指定谓词(或 where 子句)，还指定了表中访问的列。通过只在访问某些行和列时审计对表的访问，可以极大地减少审计表条目的数量。

程序包 DBMS_FGA 具有 4 个过程：

ADD_POLICY	添加使用谓词和审计列的审计策略
DROP_POLICY	删除审计策略
DISABLE_POLICY	禁用审计策略，但保留与表或视图关联的策略
ENABLE_POLICY	启用策略

用户 TAMARA 通常每天访问 HR.EMPLOYEES 表，查找雇员的电子邮件地址。系统管理员怀疑 TAMARA 正在查看经理们的薪水信息，因此他们建立一个 FGA 策略，用于审计任何经理对 SALARY 列的任何访问：

```
begin
    dbms_fga.add_policy(
         object_schema =>   'HR',
         object_name =>     'EMPLOYEES',
         policy_name =>     'SAL_SELECT_AUDIT',
         audit_condition => 'instr(job_id,''_MAN'') > 0',
         audit_column =>    'SALARY'
```

```
      );
end;
```

可以使用数据字典视图 DBA_FGA_AUDIT_TRAIL 访问细粒度审计的审计记录。如果一般需要查看标准的审计行和细粒度的审计行，则数据字典视图 DBA_COMMON_AUDIT_TRAIL 结合了这两种审计类型中的行。

继续看示例，用户 TAMARA 运行了如下两个 SQL 查询：

```
SQL> select employee_id, first_name, last_name, email from hr.employees
  2     where employee_id = 114;
EMPLOYEE_ID FIRST_NAME           LAST_NAME                 EMAIL
----------- ------------------ ---------------------   --------------
        114   Den                   Raphaely                  DRAPHEAL
1 row selected.
SQL> select employee_id, first_name, last_name, salary from hr.employees
  2     where employee_id = 114;
EMPLOYEE_ID FIRST_NAME           LAST_NAME                      SALARY
----------- ------------------ ------------------------   ----------
        114   Den                   Raphaely                       11000

1 row selected.
```

第一个查询访问经理信息，但没有访问 SALARY 列。第二个查询与第一个查询相同，但是访问了 SALARY 列，因此触发了 FGA 策略，从而在审计跟踪中生成了一行：

```
SQL> select to_char(timestamp,'mm/dd/yy hh24:mi') timestamp,
  2      object_schema, object_name, policy_name, statement_type
  3  from dba_fga_audit_trail
  4  where db_user = 'TAMARA';
TIMESTAMP        OBJECT_SCHEMA OBJECT_NAME    POLICY_NAME      STATEMENT_TYPE
-------------- ------------- ------------- ---------------- --------------
08/12/07 18:07 HR               EMPLOYEES      SAL_SELECT_AUDIT SELECT
1 row selected.
```

因为在本章前面的 VPD 示例中建立了细粒度的访问控制来阻止对 SALARY 列的未授权访问，因此需要加倍检查策略函数，确保仍然正确限制了 SALARY 信息。细粒度的审计以及标准审计是确保首先正确建立授

权策略的好方法。

9.4.6 与审计相关的数据字典视图

表 9-20 包含了与审计相关的数据字典视图。

表 9-20 与审计相关的数据字典视图

数据字典视图	说　　明
AUDIT_ACTIONS	包含审计跟踪动作类型代码的描述，例如 INSERT、DROP VIEW、DELETE、LOGON 和 LOCK
DBA_AUDIT_OBJECT	与数据库中对象相关的审计跟踪记录
DBA_AUDIT_POLICIES	数据库中的细粒度审计策略
DBA_AUDIT_SESSION	与 CONNECT 和 DISCONNECT 相关的所有审计跟踪记录
DBA_AUDIT_STATEMENT	与 GRANT、REVOKE、AUDIT、NOAUDIT 和 ALTER SYSTEM 命令相关的审计跟踪条目
DBA_AUDIT_TRAIL	包含标准审计跟踪条目。USER_AUDIT_TRAIL 只包含已连接用户的审计行
DBA_FGA_AUDIT_TRAIL	细粒度审计策略的审计跟踪条目

(续表)

数据字典视图	说　　明
DBA_COMMON_AUDIT_TRAIL	将标准的审计行和细粒度的审计行结合在一个视图中
DBA_OBJ_AUDIT_OPTS	对数据库对象生效的审计选项
DBA_PRIV_AUDIT_OPTS	对系统权限生效的审计选项
DBA_STMT_AUDIT_OPTS	对语句生效的审计选项

9.4.7 保护审计跟踪

审计跟踪自身需要受到保护，特别是在非系统用户必须访问表 SYS.AUD$时。内置的角色 DELETE_ANY_CATALOG 是非 SYS 用户可以访问审计跟踪的一种方法(例如，归档和截取审计跟踪，以确保它不会影响到 SYS 表空间中其他对象的空间需求)。

为了建立对审计跟踪自身的审计，以 SYSDBA 身份连接到数据库，并运行下面的命令：

```
SQL> audit all on sys.aud$ by access;
Audit succeeded.
```

现在，所有针对表 SYS.AUD$的动作，包括 select、insert、update 和 delete，都记录在 SYS.AUD$自身中。但是，您可能会问，如果某个人删除了标识对表 SYS.AUD$访问的审计记录，这时会发生什么？此时将删除表中的行，但接着插入另一行，记录行的删除。因此，总是存在一些针对 SYS.AUD$表的(有意的或偶然的)活动的证据。此外，如果将 AUDIT_SYS _OPERATIONS 设置为 True，使用 as sysdba、as sysoper 或以 SYS 自身连接的任何会话将记录到操作系统审计位置中，甚至 Oracle DBA 可能都无法访问该位置。因此，有许多合适的安全措施，用于确保记录数据库中所有权限的活动，以及隐藏该活动的任何尝试。

9.4.8 启用增强的审计

从 Oracle Database 11g 开始，数据库配置助手(Database Configuration Assistant，DBCA)很容易启用默认的(增强的)审计。虽然记录审计信息有一些系统开销，但兼容性需求(例如，Sarbanes-Oxley 法案中规定的兼容性需求)要求严格监控所有业务操作，包括数据库中与安全相关的操作。

可以在创建数据库时或在数据库已经创建之后使用 DBCA 配置默认审计。如果已经改变了很多审计设置，并想要将审计选项重置为基线值，则在数据库已创建之后使用 DBCA 配置默认审计就非常有用。

除将初始参数 AUDIT_TRAIL 的值设置为 DB 外，默认审计设置还审计 audit role 命令本身。另外，在 Audited Privileges 选项卡的 Oracle Enterprise Manager Audit Settings 页面中，可以查看默认的审计权限。图 9-9 展示了默认的审计权限以及本章前面创建的其他两个审计权限。

图 9-9　使用 OEM 显示审计权限

9.5　数据加密技术

数据加密可以增强数据库内部和外部的安全性。用户可能具有访问表中大多数列的合法需求，但如果对其中一列进行加密，并且用户不知道加密密钥，则无法使用相关的信息。同样的问题也适用于需要通过网络安全发送的信息。本章已介绍的技术，包括验证、授权和审计，可以确保数据库用户合法访问数据，但不能阻止操作系统用户访问数据，因为这些用户可能具有操作系统文件的访问权，而数据库本身就是由这些操作系统文件组成的。

用户可以采用如下两种方法进行数据加密：一种方法是使用程序包 DBMS_CRYPTO(在 Oracle Database 10g 中，这一程序包替换了 Oracle9i 中的 DBMS_OBFUSCATION_TOOLKIT 程序包)；另一种方法是透明数据加密，这种方法以全局方式存储加密密钥，并包含加密整个表空间的方法。

9.5.1 DBMS_CRYPTO 程序包

作为 Oracle 10g 的新增内容，程序包 DBMS_CRYPTO 代替了 DBMS_OBFUSCATION_ TOOLKIT，并且包括 Advanced Encryption Standard(AES)加密算法，(AES 算法代替了 Data Encryption Standard(DES)算法)。

DBMS_CRYPTO 中的过程可以生成私有密钥,也可以自己指定并存储密钥。与只可以加密 RAW 或 VARCHAR2 数据类型的 DBMS_OBFUSCATION_TOOLKIT 不同的是，DBMS_ CRYPTO 可以加密 BLOB 和 CLOB 类型。

9.5.2 透明数据加密

透明数据加密是一种基于密钥的访问控制系统，它依赖于外部模块实施授权。包含加密列的每个表都有自己的加密密钥，加密密钥又由为数据库创建的主密钥来加密，加密密钥以加密方式存储在数据库中，但主密钥并不存储在数据库中。重点要强调的是“透明”这一术语——当访问表中或加密表空间中的加密列时，授权用户不必指定密码或密钥。

- 虽然 Oracle Database 11g 大大地增强了透明数据加密特性，但它的使用仍然受到一些限制，例如，不能使用外键约束对列进行加密，因为每个表都有一个唯一的列加密密钥。一般来说，这不应该是什么问题，因为外键约束中使用的密钥应该是系统生成的、唯一的和非智能的。表的业务键和其他业务属性可能更需要加密，且它们通常并不参与与其他表的外键关系。其他数据库特性和类型也不适合进行透明数据加密：
- 除 B-树索引外的其他索引类型
- 索引的范围扫描搜索
- BFILE(外部对象)
- 物化视图日志
- 同步的更改数据捕获(Synchronous Change Data Capture)
- 可移植的表空间
- 原来的导入/导出实用工具(Oracle 9i 及更早版本)

另外还可以选择使用 DBMS_CRYPTO 以手动方式加密这些类型和特性。

注意：
在 Oracle Database 11g 中，现在可以加密内部大对象，例如 BLOB 和 CLOB 类型。

1. 创建 Oracle 钱夹

使用 Oracle 企业管理器可以为透明数据加密创建一个钱夹。选择 Server 选项卡，然后单击 Security Heading 下面的 Transparent Data Encryption 链接，将会看到如图 9-10 所示的页面。在此例中，还没有创建钱夹。文件 sqlnet.ora 用 ENCRYPTION_WALLET_LOCATION 变量存储钱夹的位置。如果 sqlnet.ora 文件中不存在此变量，则将钱夹创建在$ORACLE_HOME/admin/ database_name/wallet 中，在此例中是/u01/app/oracle/admin/dw/wallet。

图 9-10　透明数据加密：创建一个钱夹

要创建加密密钥，并将它放在钱夹中，需要创建一个钱夹密码，此密码至少有 10 个字符，要包含大写字母、小写字母、数字和标点符号。单击 OK，创建钱夹，可以看到如图 9-11 所示的页面。

图 9-11　透明数据加密：钱夹处于打开状态

如果主密钥被泄密，则可以使用图 9-10 中的页面重新创建主密钥。也可以关闭钱夹——禁用透明数据加密(Transparent Data Encryption)——并阻止访问任何加密的表列或表空间。

创建、打开和关闭钱夹的等价 SQL 命令非常简单，且输入这些命令所花费的时间可能比使用 Oracle 企业管

理器要少！要创建新的密钥，并在钱夹还不存在时创建钱夹，则使用 alter system 命令，如下所示：

```
SQL> alter system set encryption key identified by "Uni123#Lng";
System altered.
SQL>
```

注意将钱夹密钥放在双引号中，这一点非常重要。如果不将钱夹密钥用双引号引起来，则密码将映射所有小写字母，钱夹将不处于打开状态。数据库实例被停机并重新启动之后，如果此任务未以其他方式自动化，则需要使用 alter system 命令打开钱夹：

```
SQL> alter system set encryption wallet open identified by "Uni123#Lng";
System altered.
SQL>
```

最后，通过关闭钱夹，随时可以毫不费力地禁用对数据库中所有加密列的访问：

```
SQL> alter system set encryption wallet close;
System altered.
SQL>
```

要频繁地备份钱夹，且不要忘记钱夹密钥(或者是安全管理员不应该忘记钱夹密钥，安全管理员可以是与 DBA 不同的角色)，因为丢失钱夹或钱夹密码将无法对任何加密的列或表空间进行解密。

2. 加密表

可以加密一个或多个表的一列或多列，具体做法是，在 create table 命令中的列的数据类型后面添加 encrypt 关键字；或者是在已存在列的列名后面添加 encrypt 关键字。例如，要加密 EMPLOYEES 表的 SALARY 列，则使用如下命令：

```
SQL> alter table employees modify (salary encrypt);
Table altered.
SQL>
```

以前有权访问 SALARY 列的任何用户仍具有对此列的相同访问权，对用户来说，这完全是透明的。唯一的区别在于，访问包含 EMPLOYEES 表的操作系统文件的任何人都无法破译 SALARY 列。

3. 加密表空间

要加密整个数据库，则必须将 COMPATIBLE 初始化参数设置为 11.1.0.0.0，这是 Oracle Database 11g 的默认设置。如果数据库已从以前的版本升级，且将 COMPATIBLE 参数改变为 11.1.0.0.0，则这种改变是不可逆的。

已存在的表空间不能加密，要加密已有表空间的内容，则必须用 ENCRYPTION 选项创建一个新的表空间，并将已有的对象复制或移动到新的表空间。Oracle 企业管理器可以很容易地创建一个新的加密表空间。在图

9-12 中，创建一个新的名为 USERS_CRYPT 的表空间，其大小是 500MB，位于 ASM 磁盘组中。

图 9-12　创建加密的表空间

单击 Encryption Options 按钮，可以看到以前创建的钱夹的状态(钱夹必须处于打开状态才能创建加密的表空间)，可以为表空间选择想要使用的加密算法。单击 Continue 按钮之后，如图 9-13 所示，返回到 Create Tablespace 页面。

单击 Show SQL 按钮，可以看到 Oracle 企业管理器创建表空间将使用的 SQL 命令：

```
CREATE SMALLFILE TABLESPACE "USERS_CRYPT"
DATAFILE'+DATA' SIZE 500M LOGGING EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO NOCOMPRESS ENCRYPTION
USING 'AES256' DEFAULT STORAGE(ENCRYPT)
```

单击 Return 按钮，然后单击 OK 按钮，则 Oracle 企业管理器创建表空间。

图 9-13　指定加密的表空间选项

Oracle Data Guard(Oracle 数据卫士)提供了一种解决方案来实现高可用性、增强的性能和自动的故障转移。我们可以使用 Oracle Data Guard 为主数据库创建和维护多个备用数据库。可以按照只读模式启动备用数据库来支持报表用户，然后返回到备用模式。主数据库的改变能够自动从主数据库传递到备用数据库，并保证在此过程中没有数据丢失。备用数据库服务器在物理上可以与主服务器相分离。

在本章中，我们将会介绍如何管理 Oracle Data Guard 环境，并说明一个用于 Data Guard 环境的配置文件示例。

13.1　Data Guard 体系结构

在 Data Guard 的实现中，将一个运行在 ARCHIVELOG 模式下的数据库指定为服务于一个应用程序的主数据库。通过 Oracle Net(Oracle 网络)可访问的一个或多个备用数据库提供故障转移功能。Data Guard 自动将重做信息传送到应用此信息的备用数据库。因此，备用数据库在事务处理上可以保持一致。根据重做应用程序过程的配置情况，备用数据库可能与主数据库同步，也可能滞后于主数据库。

图 13-1 给出了一个标准的 Data Guard 的实现。

日志传输服务(Log Transport Services)将重做日志数据传递到备用数据库(Standby Databases)，可以通过初始化参数设置来定义日志传输服务。日志应用服务(Log Apply Services)将重做信息应用到备用数据库。第三组服务，即角色管理服务(Role Management Services)，可以简化使备用数据库充当主数据库的过程。

图 13-1　简单的 Data Guard 配置

注意：

主数据库可以是一个单实例或多实例的 RAC 实现。

13.1.1　物理备用数据库与逻辑备用数据库

有两种类型的备用数据库：物理备用数据库和逻辑备用数据库。物理备用数据库具有和主数据库相同的结构。逻辑备用数据库具有不同的内部结构(如用于报表的额外索引)。通过将重做数据转换为依据备用数据库执行的 SQL 语句，可以同步逻辑备用数据库和主数据库。

物理备用数据库和逻辑备用数据库服务于不同的目的。物理备用数据库是一种对主数据库的逐块的复制，因此它可以用作替代主数据库的数据库备份。在灾难恢复过程中，物理备用数据库看起来就像是它替代的主数据库。

由于逻辑备用数据库支持额外的数据库结构，因此可以更为容易地支持特定的报表需求，否则这种需求会加重主数据库的负担。另外，当使用逻辑备用数据库时，能够以最小的停用时间执行主数据库和备用数据库的滚动更新。使用的备用类型依赖于需要，许多环境最开始将物理备用数据库用于灾难恢复，然后添加额外的逻辑备用数据库来支持特定的报表和业务需求。

注意：

从 Oracle Database 11g 开始，主位置和备用位置上的操作系统和平台体系结构不需要相同。主数据库和备用数据库的目录结构可以有所不同，但是应最小化这种区别来简化管理和故障转移过程。如果备用数据库和主数据库位于同一个服务器上，则这两个数据库必须使用不同的目录结构，并且它们不能共享一个归档日志目录。另外，Oracle Data Guard 只能用于 Oracle 企业版中。

13.1.2 数据保护模式

当配置主数据库和备用数据库时，将需要确定业务可以接受的数据丢失的程度。在主数据库中，需要定义它的归档日志目的区，并且至少有一个目的区将会引用备用数据库使用的远程站点。备用数据库的 LOG_ARCHIVE_DEST_*n* 参数设置的 ASYNC、SYNC、ARCH、LGWR、NOAFFIRM 和 AFFIRM 属性(参见表 13-1)将会指导 Oracle Data Guard 在多种操作模式中做出选择：

- 在最大保护(或“无数据丢失”)模式下，在将一个事务处理提交到主数据库中之前，必须至少写入到一个备用位置。如果备用数据库的日志存放位置不可用，主数据库会关闭。
- 在最大可用性模式下，在将一个事务处理提交到主数据库中之前，必须至少写入到一个备用位置。如果备用位置不可用，主数据库不会关闭。当纠正了错误后，从错误出现起已经生成的重做数据会传送并应用到备用数据库上。
- 在最大性能模式(默认模式)下，可以在将它们的重做信息传送到备用位置之前提交事务处理。一旦完成写入到本地联机重做日志，就可以在主数据库中进行提交。默认情况下，由 ARCH 进程负责写入到备用位置。

一旦已经为配置确定了备用类型和数据保护模式，就可以创建备用数据库。

13.2 LOG_ARCHIVE_DEST_*n* 参数属性

正如以下的小节中说明的那样，Oracle Data Guard 配置依赖于 LOG_ARCHIVE_DEST_*n* 内的许多属性。表 13-1 总结了可用于该参数的属性。在几乎所有情况下，属性都是成对的。在一些情况下，属性对中的第二项只不过是用来取消设置。

表 13-1 LOG_ARCHIVE_DEST_*n* 参数属性

属　　性	说　　明
AFFIRM 和 NOAFFIRM	AFFIRM 保证在日志写入进程(LGWR)能够继续写入之前，同步执行并成功完成到备用目的地的归档重做日志文件或备用重做日志文件的所有磁盘 I/O 操作，因此，LGWR 在写入到主数

	据库上的本地联机重做日志文件之前一直等待
	NOAFFIRM 指示将要同步地执行到归档重做日志文件和备用重做日志文件的所有磁盘 I/O 操作；在备用目的地上的磁盘 I/O 操作完成之前，可以重用主数据库上的联机重做日志文件
ALTERNATE 和 NOALTERNATE	当原始的归档目的地失效时，ALTERNATE 指定一个可替换使用的 LOG_ARCHIVE_DEST_*n* 目的地
ARCH 和 LGWR	ARCH 默认情况下指定 ARCH 进程负责将重做数据传送到归档目的地。LGWR 指定 LGWR 进程执行日志传输操作

(续表)

属　　性	说　　明
DB_UNIQUE_NAME 和 NODB_UNIQUE _NAME	DB_UNIQUE_NAME 为目的地指定唯一的数据库名字
DELAY 和 NODELAY	DELAY 指定在备用站点上归档重做数据和将归档重做日志文件应用到备用数据库之间的时间间隔；DELAY 可以用来保护备用数据库免受损坏或错误的主数据的影响。如果没有指定 DELAY 和 NODELAY，默认采用 NODELAY
DEPENDENCY 和 NODEPENDENCY	DEPENDENCY 允许向一个目的地传输重做数据，然后在多个备用数据库之间共享它的归档重做日志文件。当创建 DEPENDENCY 时，必须使用 REGISTER 和 SERVICE 属性
LOCATION 和 SERVICE	每个目的地必须指定 LOCATION 或 SERVICE 属性来标识一个本地磁盘目录(通过 LOCATION)或一个远程数据库目的地(通过 SERVICE)，Log Transport Service 可以向此数据库传送重做数据
MANDATORY 和 OPTIONAL	如果目的地是 OPTIONAL，到此目的地的归档操作可能失败，然而仍可以重用联机重做日志文件并最终可以重写它
	如果一个 MANDATORY 目的地的归档操作失败,则不能重写联机重做日志文件
MAX_FAILURE 和 NOMAX_FAILURE	MAX_FAILURE 指定在主数据库永久放弃备用数据库之前执行的重新打开尝试的最大次数
NET_TIMEOUT 和 NONET_TIMEOUT	NET_TIMEOUT 指定在终结网络连接之前主系统上的日志写入器(log writer)进程等待来自网络服务器进程的状态所允许的秒数。默认值是 180 秒
QUOTA_SIZE 和 NOQUOTA_SIZE	QUOTA_SIZE 指示本地目的地能够使用的一个磁盘设备上的 512 字节物理存储块的最大数量
QUOTA_USED 和 NOQUOTA_USED	QUOTA_USED 标识在一个特定的目的地上归档的 512 字节数据块的数量
REGISTER 和 NOREGISTER	REGISTER 指示归档重做日志文件的位置将记录在对应的目的地
REOPEN 和 NOREOPEN	REOPEN 指定在归档器进程(ARCn)或日志写入器进程(LGWR)尝试再次访问一个以前失效的目的地之前允许的最小秒数(默认值是 300 秒)
SYNC 和 ASYNC	在使用日志写入器进程(LGWR)时，SYNC 和 ASYNC 指定网络 I/O

	操作是同步执行还是异步执行。默认情况下，SYNC=PARALLEL，用于存在多个使用 SYNC 属性的目的地的情况下。所有的目的地应该使用相同的值

（续表）

属　　性	说　　明
TEMPLATE 和 NOTEMPLATE	TEMPLATE 为备用目的地上的归档重做日志文件或备用重做日志文件的名字定义了一个目录规范和格式模版。可以在主或备用初始化参数文件中指定这些属性，但是该属性只适用于正在归档的数据库角色
VALID_FOR	VALID_FOR 根据以下的因素来标识 Log Transport Service 什么时候可以向目的地传送重做数据：(1)数据库当前运行在主角色还是备用角色下，(2)当前是否正在该目的地的数据库上归档联机重做日志文件、备用重做日志文件或者这两类文件。该属性的默认值是 VALID_FOR= (ALL_LOGFILES, ALL_ROLES)。其他的取值包括 PRIMARY_ROLE、STANDBY_ROLE、ONLINE_LOGFILES 和 STANDBY_LOGFILE
VERIFY 和 NOVERIFY	VERIFY 指示 archiver 进程应该检验完成的归档重做日志文件内容的正确性。默认值是 NOVERIFY

13.3　创建备用数据库配置

可以使用 SQL*Plus、Oracle Enterprise Manager(OEM)或 Data Guard 特有的工具来配置和管理 Data Guard 配置。设定的参数将依赖于所选择的配置。

如果主数据库和备用数据库在同一个服务器上，需要为 DB_UNIQUE_NAME 参数设定一个值。由于这两个数据库的目录结构将会是不同的，因此必须手动重新命名文件或者为备用数据库中的 DB_FILE_NAME_CONVERT 和 LOG_FILE_NAME_CONVERT 参数定义值。必须通过 SERVICE_NAMES 初始化参数为主和备用数据库设置唯一的服务名。

如果主数据库和备用数据库位于分离的服务器上，可以将相同的目录结构用于每个数据库，从而不需要文件名转换参数。如果将一个不同的目录结构用于数据库文件，需要为备用数据库中的 DB_FILE_NAME_CONVERT 和 LOG_FILE_NAME_CONVERT 参数定义值。

在物理备用数据库中，所有的重做(数据)来自于主数据库。当物理备用数据库在只读模式下打开时，不会产生重做(数据)。然而，Oracle Data Guard 的确要使用归档的重做日志文件来支持数据和 SQL 命令的复制，以便用于更新备用数据库。

注意：

对每个备用数据库而言，应该创建一个备用重做日志文件来存储从主数据库上接收到的重做数据。

13.3.1　准备主数据库

在主数据库上，确保已经为以下会影响重做日志数据传递的参数设定了值。下面列出的前 5 个参数对于大多数数据库而言是标准的参数。将 REMOTE_LOGIN_PASSWORDFILE 设置为 EXCLUSIVE 来支持 SYSDBA 权限用户的远程访问：

DB_NAME	数据库名字。所有的备用数据库和主数据库都使用相同的名字
DB_UNIQUE_NAME	数据库的唯一名字。每个备用数据库的这个参数值必须不同，且它们与主数据库的该参数值也不同
SERVICE_NAMES	数据库的服务名称；为主和备用数据库设置不同的服务名
CONTROL_FILES	控制文件的位置
REMOTE_LOGIN_PASSWORDFILE	设置为 EXCLUSIVE 或 SHARED。为主和备用数据库上的 SYS 设置相同的口令

下面列出的与 LOG_ARCHIVE 相关的参数将用于配置 Log Transport Services 的工作方式：

LOG_ARCHIVE_CONFIG	在 DB_CONFIG 参数内，列出主数据库和备用数据库
LOG_ARCHIVE_DEST_1	主数据库的归档重做日志文件的位置
LOG_ARCHIVE_DEST_2	用于存放备用重做日志文件的远程位置
LOG_ARCHIVE_DEST_STATE_1	设为 ENABLE
LOG_ARCHIVE_DEST_STATE_2	设为 ENABLE 来启用日志传输
LOG_ARCHIVE_FORMAT	为归档日志文件的名字指定格式

对于该示例，假定主数据库的 DB_UNIQUE_NAME 值为 headqtr 且物理备用数据库的 DB_UNIQUE_NAME 值为 salesofc。SERVICE_NAMES 的值可以与 DB_UNIQUE_NAME 的值相同，但并不是必须相同。事实上，SERVICE_NAMES 的值对于 RAC 实例中的单一节点也许是唯一的。

LOG_ARCHIVE_CONFIG 参数设置可能类似于下面：

```
LOG_ARCHIVE_CONFIG='DG_CONFIG=(headqtr,salesofc)'
```

有两个 LOG_ARCHIVE_DEST_*n* 表项——一个用于归档重做日志文件的本地副本，另一个用于将传输到物理备用数据库上的远程副本。

```
LOG_ARCHIVE_DEST_1=
 'LOCATION=/arch/headqtr/
  VALID_FOR=(ALL_LOGFILES,ALL_ROLES)
  DB_UNIQUE_NAME=headqtr'
LOG_ARCHIVE_DEST_2=
 'SERVICE=salesofc
  VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)
  DB_UNIQUE_NAME=salesofc'
```

LOG_ARCHIVE_DEST_1 参数为主数据库指定归档重做日志文件的位置(通过 DB_UNIQUE_NAME 参数来指定)。LOG_ARCHIVE_DEST_2 参数赋予物理备用数据库的服务名作为它的位置。对于每一个目的地，对应的 LOG_ARCHIVE_DEST_STATE_*n* 参数应该有一个 ENABLE 值。

与备用角色相关的参数包括 FAL(Fetch Archive Log)参数，在 Oracle Database 10g 之前，这些参数用来解决复制到备用数据库的一系列归档日志中的间隙：

FAL_SERVER	指定 FAL 服务器(通常是主数据库)的服务名
FAL_CLIENT	指定 FAL 客户机(取用日志的备用数据库)的服务名
DB_FILE_NAME_CONVERT	如果主数据库和备用数据库使用不同的目录结构，指定主数据库数据文件的路径名和文件名位置，后面跟着备用位置
LOG_FILE_NAME_CONVERT	如果主数据库和备用数据库使用不同的目录结构，指定主数据库日志文件的路径名和文件名位置，后面跟着备用位置
STANDBY_FILE_MANAGEMENT	设为 AUTO

提示：
在每个节点上都应该定义 FAL_SERVER 和 FAL_CLIENT，这样在角色切换之后它们就做好准备切换回原来的角色。

下面的程序清单中给出了这些参数的设置示例：

```
FAL_SERVER=headqtr
FAL_CLIENT=salesofc
LOG_FILE_NAME_CONVERT=
'/arch/headqtr/','/arch/salesofc/','/arch1/headqtr/','/arch1/salesofc/'
STANDBY_FILE_MANAGEMENT=AUTO
```

如果主数据库还没有处于 ARCHIVELOG 模式下，那么在安装数据库但没有打开它的时候，通过发出 alter database archivelog 命令来启用归档。另外，使用 alter database force logging 命令启用主数据库的强制日志记录，以确保未记录日志而直接写入的所有操作都传播到备用数据库。

一旦已经设置了与日志相关的参数，就可以开始创建备用数据库的过程。

1. 步骤 1：备份主数据库的数据文件

首先，对主数据库执行物理备份。Oracle 建议使用 RMAN 实用程序来备份数据库，可以在 RMAN 内使用 duplicate 命令来自动化操作创建备用数据库的过程。

2. 步骤 2：为备用数据库创建控制文件

在主数据库中，发出如下的命令来产生将用于备用数据库的控制文件：

```
alter database create standby controlfile as '/tmp/salesofc.ctl';
```

需要注意的是，必须为要创建的控制文件指定目录和文件名。而且，不要使用与主数据库相同的目录和控制文件名。

3. 步骤 3：为备用数据库创建初始化文件

在主数据库中，从服务器参数文件中创建参数文件:

```
create pfile='/tmp/initsalesofc.ora' from spfile;
```

编辑该初始化文件来为备用数据库设定正确的值。为备用数据库设定 DB_UNIQUE_NAME、SERVICE_NAMES、CONTROL_FILES、DB_FILE_NAME_CONVERT、LOG_FILE_NAME_CONVERT、LOG_ARCHIVE_DEST_*n*、INSTANCE_NAME、FAL_SERVER 和 FAL_CLIENT 参数值。文件名转换应该与主数库中的情况相同——当应用重做信息时，希望将文件名从主数据库转换到备用数据库格式：

```
LOG_ARCHIVE_DEST_1=
'LOCATION=/arch/salesofc/
VALID_FOR=(ALL_LOGFILES,ALL_ROLES)
DB_UNIQUE_NAME=salesofc'
LOG_ARCHIVE_DEST_2=
'SERVICE=headqtr
VALID_FOR=(ONLINE_LOGFILES,PRIMARY_ROLE)
```

在备用环境中，LOG_ARCHIVE_DEST_1 参数指向它的本地归档日志目的地，并且 LOG_ARCHIVE_DEST_2 指向主数据库的服务名。如果交换这两个数据库的角色，原始的主数据库将可以充当备用数据库。当备用数据库正运行在备用模式下时，将会忽略 LOG_ARCHIVE_DEST_2 参数值。

注意：

为主数据库和备用数据库将 COMPATIBLE 参数设置为相同的值。为了利用 Oracle 11g 中的新特性，将 COMPATIBLE 设置为 11.0.0 或更高的值。一旦将 COMPATIBLE 设置为 11.0.0，则不能把它重置为一个更低的值。

4. 步骤 4：将数据库文件复制到备用数据库位置

将步骤 1 中的数据文件、步骤 2 中的控制文件以及步骤 3 中的备用初始化文件复制到备用数据库位置。将这些文件放置在正确的目录中(按照 CONTROL_FILES、DB_FILE_NAME_ CONVERT 和 LOG_FILE_NAME_CONVERT 参数定义的那样)。作为一种替换方法，使用主数据库的 RMAN 备份来创建备用数据库文件。

5. 步骤 5：配置备用数据库环境

此时，文件已经处于适当的位置。需要创建正确的环境变量和服务来允许实例访问这些文件。例如，在 Windows 环境中，应该使用 ORADIM 实用程序来创建新的服务，如下面这个示例所示：

```
ORADIM -NEW -SID salesofc -INTPWD oracle -STARTMODE MANUAL
```

接下来，通过 ORAPWD 实用程序为备用数据库创建一个口令文件(创建新的口令文件的详细内容请参见第 2 章)。

然后，创建访问备用数据库所需要的 Oracle Net 参数和服务。在备用环境中，为备用数据库创建一个 Oracle Net 侦听服务。在备用服务器的 sqlnet.ora 文件中，将 SQLNET.EXPIRE_TIME 参数设为 1，以便在 1 分钟后激活对断开连接的检测。 参见第 15 章进一步了解有关 Oracle Net 连接的细节。

接下来，在 tnsnames.ora 文件中为备用数据库创建一个服务名表项，然后将更新分发到主数据库和备用数据库服务器。

如果主数据库有一个加密钱夹，则将钱夹复制到备用数据库系统，并配置备用数据库来使用此钱夹。每当主加密密钥更新时，必须将钱夹从主数据库重新复制到所有备用数据库。

最后，通过 create spfile from pfile 命令创建一个服务器参数文件，将备用参数文件的名字和位置作为输入传递到该命令。

6. 步骤 6：启动备用数据库

从 SQL*Plus 中，以 mount 模式启动备用数据库，如下面的例子中所示：

```
startup mount;
```

注意：

可以向备用数据库中的临时表空间添加新的临时文件。添加临时文件将会支持报告备用数据库内的活动所需的分类操作。

虽然是可选的，但 Oracle 建议在每个备用数据库上创建一个联机重做日志，以便减少将备用数据库转换为主角色所花费的时间。

通过如下的 alter database 命令在备用数据库内启动重做应用程序过程：

```
alter database recover managed standby database
   using current logfile disconnect from session;
```

7. 步骤 7：检验配置

为了检验配置，使用主数据库并通过 alter system 命令强制进行日志转换，如下所示：

```
alter system switch logfile;
```

随后，主数据库的重做日志数据将复制到备用位置。

在备用数据库上，可以通过查询 V$ARCHIVED_LOG 视图或使用 archive log list 命令来查看已经将哪些归档日志应用到数据库上。当从主数据库上接收新的日志并将它们应用到备用数据库上时，新的行将会添加到 V$ARCHIVED_LOG 中的列表上。

13.3.2 创建逻辑备用数据库

创建逻辑备用数据库与创建物理备用数据库有很多步骤是相同的。因为逻辑备用数据库依赖于重新执行 SQL 命令，所以在使用上逻辑备用数据库将有更大的限制。如果主数据库中的任何一个表使用如下的数据类型，在重做应用程序过程中将会忽略它们：

- BFILE
- ROWID
- UROWID
- 用户定义的数据类型
- 对象类型
- REF
- 可变的数组
- 嵌套的表
- XML 类型

此外，在重做应用过程中会忽略采用表压缩的表和使用 Oracle 软件安装的模式。DBA_LOGSTDBY_UNSUPPORTED 视图列出了不支持用于逻辑备用数据库的对象。DBA_LOGSTDBY_SKIP 视图列出了将会忽略的模式。

逻辑备用数据库不同于主数据库。在逻辑备用数据库中执行的每个事务处理必定是主数据库中已执行的事务处理的逻辑等价体。因此，应该确保您的表对它们施加了适当的限制—— 主键、唯一约束、检查约束和外键—— 因此能够选取正确的行用于逻辑备用数据库中的更新。可以查询 DBA_LOGSTDBY_NOT_UNIQUE 来列出主数据库中那些缺少主键或唯一约束的表。

为了创建逻辑备用数据库，遵循以下步骤：

1. 步骤 1：创建物理备用数据库

遵照本章上一节中的步骤，创建一个物理备用数据库。创建并启动物理备用数据库之后，停止物理备用数据库上的“重做应用(Redo Apply)”，以避免将变更应用到包含补充日志信息的重做上：

```
alter database recover managed standby database cancel;
```

2. 步骤 2：启用补充日志

主数据库上的补充日志会在重做日志中产生额外的信息。然后，在备用数据库上的重做应用程序过程中会使用这些信息，以便确保生成的 SQL 可以作用于正确的行。为了向重做数据添加主键和唯一的索引信息，在主数据库中发出如下的命令：

```
execute dbms_logstdby.build;
```

此过程等待已存在的所有事务完成。如果主数据库上有长期运行的事务，则这一过程直到那些事务提交或回滚才完成。

3. 步骤 3：将物理备用转换为逻辑备用

重做日志文件包括将物理数据库转换为逻辑数据库所必需的信息。运行这一命令继续对物理备用数据库运行重做日志数据应用，直到准备将物理备用数据库转换为逻辑备用数据库：

```
alter database recover to logical standby new_db_name;
```

Oracle 自动将新逻辑备用数据库的名称 new_db_name 存储在 SPFILE 中。否则此命令会生成一条消息，提醒您在关闭数据库之后改变初始化参数文件中的 DB_NAME parameter 参数。

物理备用数据库以只读模式操作；逻辑备用数据库是打开的，以便可以写入，并且产生其自己的重做数据。在用于逻辑备用数据库的初始化文件中，为逻辑备用数据库的重做数据(LOG_ARCHIVE_DEST_1)和从主数据库中输入的重做数据(在这个例子中，将使用 LOG_ARCHIVE_DEST_3 来避免与前面的 LOG_ARCHIVE_DEST_2 设置相冲突)指定目的地。我们不希望逻辑备用数据库启用 LOG_ARCHIVE_DEST_2 目的地，并往回指向主数据库。

关闭并启动数据库，改变这些参数：

```
shutdown;
startup mount;
```

4. 步骤 4：启动逻辑备用数据库

使用逻辑备用数据库的新初始化参数文件或 SPFILE 打开逻辑备用数据库，如下所示：

```
alter database open resetlogs;
```

因为这是数据库被转换为备用数据库之后第一次打开，所以数据库的全局名被调整，以匹配新的 DB_NAME 初始化参数。

5. 步骤 5：启动重做应用过程

目前在逻辑备用数据库内，可以启动重做应用程序过程：

```
alter database start logical standby apply immediate;
```

为了查看已经收到并应用到逻辑备用数据库的日志，可以查询 DBA_LOGSTDBY_LOG 视图。可以查询 V$LOGSTDBY 视图来了解逻辑备用重做应用过程的活动日志。现在，可以使用逻辑备用数据库。

13.4 使用实时应用

默认情况下，直到已经归档了备用重做日志文件后，才将重做数据应用到备用数据库。当采用实时应用特性时，在接收重做数据的同时将它应用到备用数据库，从而减少了数据库之间的时间延迟并潜在地缩短了故障转移到备用数据库所需的时间。

为了在物理备用数据库中启用实时应用，在备用数据库内执行如下的命令：

```
alter database recover managed standby database
using current logfile;
```

对于逻辑备用数据库，使用的命令是：

```
alter database start logical standby apply immediate;
```

如果已经启用了实时应用，V$ARCHIVE_DEST_STATUS 视图的 Recovery_Mode 列将有一个“MANAGED REAL-TIME APPLY”的值。

正如本章前面所述，可以通过如下的命令在物理备用数据库上启用重做应用 程序：

```
alter database recover managed standby database disconnect;
```

在与 Oracle 会话失去连接后，disconnect 关键字允许该命令在后台运行。当启动了一个前台会话并

发出相同的不带 disconnect 关键字的命令时，控制不会返回到命令提示符，直到通过另一个会话取消了恢复。为了在物理备用数据库中停止重做应用程序——无论是在后台会话还是在前台会话中—— 使用如下命令：

```
alter database recover managed standby database cancel;
```

对于逻辑备用数据库，停止 Log Apply Service 的命令是：

```
alter database stop logical standby apply;
```

13.5 管理归档日志序列中的间隙

如果备用数据库还没有收到由主数据库产生的一个或多个归档日志，它不会拥有主数据库中事务处理的完全记录。Oracle Data Guard 会自动检测归档日志序列中的间隙，它通过将遗失的日志文件序列复制到备用目的地来解决这一问题。在 Oracle 10g 之前，使用 FAL(Fetch Archive Log)客户机和服务器来解决源自主数据库的间隙。

为了确定物理备用数据库中是否存在间隙，可以查询 V$ARCHIVE_GAP 视图。对于每个间隙，该视图将会报告备用数据库中遗失的一组日志中的最低和最高的日志序列号。如果由于某种原因 Oracle Data Guard 未能复制这些日志，可以手动将这些文件复制到物理备用数据库环境中，并使用 alter database register logfile filename 命令来注册它们，然后可以启动重做应用过程。在已经应用了日志后，再次检查 V$ARCHIVE_GAP 视图来查看是否存在另一个要解决的间隙。

13.6 管理角色—— 切换和故障转移

参与 Data Guard 配置的每个数据库都有一个角色—— 或者是主数据库，或者是备用数据库。在某些时候，这些角色可能需要改变。例如，如果主数据库的服务器上有一个硬件故障，可以在故障发生时转移到备用数据库。根据配置选择，在故障转移过程中可能有一些数据丢失。

第二类角色改变称为“切换”(switchover)，这种情况出现在当主数据库和备用数据库切换角色并且备用数据库变成新的主数据库的时候。在切换过程中，应该不存在数据丢失。切换和故障转移都需要数据库管理员的手动干预。

13.6.1 切换

切换是有计划的角色改变，通常考虑在主数据库服务器上执行维护活动。当选择一个备用数据库充当新的主数据库时，切换会发生，应用程序此刻把它们的数据写入到新的主数据库中。在以后的某个时间点，可以把数据库切换回它们原始的角色。

注意：

可以使用一个逻辑备用数据库或一个物理备用数据库来执行切换，物理备用数据库是首选的选项。

如果已经定义了多个备用数据库，那么该怎么办呢？当其中一个物理备用数据库变为新的主数据库时，其他的备用数据库必须能够从新的主数据库中接收它们的重做日志数据。在这种配置中，必须定义 LOG_ARCHIVE_DEST_*n* 参数，以便允许这些备用站点从新的主数据库那里接收数据。

注意：

要验证将变成新的主数据库的数据库是否正运行在 ARCHIVELOG 模式下。

在以下的小节中，将会学习执行切换到备用数据库所需的步骤。在切换之前，备用数据库应该主动地应用重做日志数据，因为这会最小化完成切换所需的时间。

13.6.2 切换到物理备用数据库

在主数据库上发起切换并在备用数据库上完成切换。在本节中，将会了解切换到物理备用数据库所需要的步骤。在切换过程中不会丢失数据。

从检验主数据是否能够执行切换开始，查询 V$DATABASE 来了解 SWITCHOVER_STATUS 列的值：

```
select switchover_status from v$database;
```

如果 SWITCHOVER_STATUS 列的值不是 TO STANDBY，是不可能执行切换的(通常由于配置或硬件问题)。如果该列的值是 SESSIONS ACTIVE，应该终止活动的用户会话。Switchover_Status 列的有效值如表 13-2 所示。

表 13-2 SWITCHOVER_STATUS 的值

Switchover_Status 值	说　明
NOT ALLOWED	当前的数据库不是带有备用数据库的主数据库
PREPARING DICTIONARY	该逻辑备用数据库正在向一个主数据库和其他备用数据库发送它的重做数据，以便为切换做准备
PREPARING SWITCHOVER	接受用于切换的重做数据时，逻辑备用配置会使用它
RECOVERY NEEDED	备用数据库还没有接收到切换请求
SESSIONS ACTIVE	在主数据库中存在活动的 SQL 会话；在继续执行之前必须断开这些会话
SWITCHOVER PENDING	适用于那些已收到主数据库切换请求但是还没有处理该请求的备用数据库
SWITCHOVER LATENT	切换没有完成并返回到主数据库
TO LOGICAL STANDBY	主数据库已经收到了来自逻辑备用数据库的完整的字典
TO PRIMARY	该备用数据库可以转换为主数据库
TO STANDBY	该主数据库可以转换为备用数据库

在主数据库中，可以使用如下命令把它转换到物理备用数据库角色：

```
alter database commit to switchover to physical standby;
```

在执行该命令的同时，Oracle 会将当前主数据库的控制文件备份到一个跟踪文件上。此时，应该关闭主数据库并安装它：

```
shutdown immediate;
startup mount;
```

主数据库为切换做好了准备，现在应该回到将充当新的主数据库的物理备用数据库。

在物理备用数据库中,检查V$DATABASE视图中的切换状态,它的状态应该为TO PRIMARY(参见表13-2)。现在可以通过如下命令将物理备用数据库切换为主数据库：

```
alter database commit to switchover to primary;
```

如果增加with session shutdown wait子句，则在切换完成之前该语句将不会返回到SQL>提示符。使用open关键字启动数据库：

```
alter database open;
```

数据库已经完成了到主数据库角色的转换。如果重做应用服务没有在后台运行的话，接下来在备用数据库上启动它们：

```
alter database recover managed standby database using current logfile
   disconnect from session;
```

13.6.3 切换到逻辑备用数据库

在主数据库上发起切换并在备用数据库上完成切换。在本节中，将会了解执行到逻辑备用数据库的切换所需的步骤。

从检验主数据库是否能够执行切换开始，查询V$DATABASE来了解Switchover_Status列 的值：

```
select switchover_status from v$database;
```

为了完成切换，该状态必须是TO STANDBY、TO LOGICAL STANDBY或SESSIONS ACTIVE。

在主数据库中，发出如下命令以便使主数据库准备用于切换：

```
alter database prepare to switchover to logical standby;
```

在逻辑备用数据库中，发出如下命令：

```
alter database prepare to switchover to primary;
```

在这个时候，逻辑备用数据库将开始向当前的主数据库和配置中的其他备用数据库传送它的重做数据。此时，传送逻辑备用数据库中的重做数据，但是没有应用这些数据。

在主数据库中，现在必须检验从逻辑备用数据库上接收了字典数据。在能够继续执行下一步骤之前，V$DATABASE中的SWITCHOVER_STATUS列的值在主数据库中必须读取为TO LOGICAL STANDBY。当该状态值显示在主数据库中时，将主数据库切换到逻辑备用角色：

```
alter database commit to switchover to logical standby;
```

不需要关闭和重启旧的主数据库。现在应该回到最初的逻辑备用数据库并检验它在 V$DATABASE 中的 SWITCHOVER_STATUS 值(应该是 TO PRIMARY)。然后可以完成切换，在原有的逻辑备用数据库中，发出如下命令：

```
alter database commit to switchover to primary;
```

现在最初的逻辑备用数据库成为主数据库。

在新的逻辑备用数据库(旧的主数据库)中，启动重做应用过程：

```
alter database start logical standby apply immediate;
```

到此完成了切换。

13.6.4 到物理备用数据库的故障转移

当主数据库不再是主数据库配置的一部分时会出现故障转移。在本节中，将会了解在 Data Guard 配置中执行物理备用数据库到主数据库角色的故障转移所需要的步骤。

在备用数据库中，首先要设法标识和解决归档重做日志文件中的任何间隙(参见第 13.5 节)。可能需要手动复制和注册日志文件，以便用于备用数据库。

然后在备用数据库中，必须完成恢复过程。如果已经配置了备用数据库拥有备用重做日志文件，要执行的命令是：

```
alter database recover managed standby database finish;
```

如果没有备用重做日志文件，执行如下命令：

```
alter database recover managed standby database finish
        skip standby logfile;
```

一旦已完成了备用恢复操作，可以使用如下命令执行切换：

```
alter database commit to switchover to primary;
```

关闭并重启新的主数据库来完成此转换。旧的主数据库不再是 Data Guard 配置的一部分。如果希望重新创建旧的主数据库并将它用作备用数据库，必须采用本章前面提供的步骤将它创建为一个备用数据库。

13.6.5 到逻辑备用数据库的故障转移

当主数据库不再是主数据库配置的一部分时会出现故障转移。在本节中，将会了解在 Data Guard 配

置中执行逻辑备用数据库到主数据库角色的故障转移所需要的步骤。

在备用数据库中，首先要设法标识和解决归档重做日志文件中的任何间隙(参看第 13.5 节)。可能需要手动复制和注册日志文件，以便用于备用数据库。查询 DBA_LOGSTDBY_LOG 视图来了解将要应用的其余日志的详细信息。如果在逻辑备用数据库中没有激活重做应用过程，使用如下命令来启动它：

```
alter database start logical standby apply nodelay finish;
```

接下来，为逻辑备用数据库产生的重做日志文件启用远程存储位置。可能需要更新逻辑备用数据库的 LOG_ARCHIVE_DEST_STATE_*n* 参数设置，以便配置中的其他备用数据库将可以接收到由原始的逻辑备用数据库产生的重做数据。然后，可以通过如下命令将原始的逻辑备用数据库激活为新的主数据库：

```
alter database activate logical standby database finish apply;
```

如果存在属于 Data Guard 配置的其他逻辑备用数据库，可能需要重新创建它们或使用数据库链接将它们添加到新的配置中。首先，在将要充当新的主数据库的逻辑备用数据库的每个数据库中创建一个链接。alter session disable guard 命令允许绕过会话中的 Data Guard 过程。数据库链接使用的数据库账户必须具有 SELECT_CATALOG_ROLE 角色：

```
alter session disable guard;
create database link salesofc
   connect to username identified by password using 'salesofc';
alter session enable guard;
```

应该通过查询远程数据库(新的主数据库)中的 DBA_LOGSTDBY_PARAMETERS 视图来检验该链接。

在每个逻辑备用数据库中，现在可以基于新的主数据库来启动重做应用程序过程：

```
alter database start logical standby apply new primary salesofc;
```

13.7 管理数据库

在以下部分中，我们会介绍在属于 Data Guard 配置的数据库上执行标准的维护行动需要采取的步骤，包括启动和关闭操作。

13.7.1 启动和关闭物理备用数据库

当启动物理备用数据库时，应该启动重做应用过程。首先，安装数据库：

```
startup mount;
```

接下来，启动重做应用过程：

```
alter database recover managed standby database
       disconnect from session;
```

使用 using current logfile 子句替代 disconnect from session 子句来启动实时应用。

为了关闭备用数据库，应该首先停止 Log Apply Service。查询 V$MANAGED_STANDBY 视图，如果在视图中列出了 Log Apply Service，使用如下命令来取消它们：

```
alter database recover managed standby database cancel;
```

然后，可以关闭数据库。

13.7.2 以只读模式打开物理备用数据库

为了打开物理备用数据库用于读操作，首先应该取消数据库中的任何日志应用操作：

```
alter database recover managed standby database cancel;
```

接下来，打开数据库：

```
alter database open;
```

13.7.3 在 Data Guard 环境下管理数据文件

在本章的前面指出过，应该将 STANDBY_FILE_MANAGEMENT 初始化参数设置为 AUTO。设置该参数可以简化 Data Guard 环境的管理，因为添加到主数据库环境中的文件能够自动传播到物理备用数据库。当该参数设置为 AUTO 时，在主数据库中创建的任何新的数据文件会自动地在备用数据库中创建；当该参数设置为 MANUAL 时，必须在备用数据库中手动地创建新的数据文件。

当 STANDBY_FILE_MANAGEMENT 设置为 MANUAL 时，采用如下步骤将数据文件添加到表空间中：

(1) 在主数据库中添加新的数据文件。

(2) 改变该数据文件的表空间，使它处于脱机状态。

(3) 将数据文件复制到备用位置。

(4) 改变数据文件的表空间，使它再次联机。

为了使用手动文件管理来添加一个新的表空间，采用相同的步骤—— 创建表空间，使表空间脱机，

将它的数据文件复制到备用位置，然后修改表空间使它联机。如果正使用自动文件管理，只需在主数据库中为它创建要传播到备用数据库的新的表空间。

为了删除表空间，只需在主数据库中删除它并通过 alter system switch logfile 命令强制执行日志切换。然后，可以在主和备用环境中的操作系统级删除文件。

不传播数据文件名的改动，即使正在使用自动文件管理。为了在 Data Guard 配置中重命名一个数据文件，将表空间置为脱机，并在主服务器上的操作系统级对该数据文件重命名。在主数据库上使用 alter tablespace rename datafile 命令来指向数据文件的新位置。使用 alter tablespace tablespace_name online 命令使表空间返回联机状态，在备用数据库上，查询 V$ARCHIVED_LOG 视图来检验是否已应用了所有的日志，然后关闭重做应用服务：

```
alter database recover managed standby database cancel;
```

关闭备用数据库，并在备用服务器上重命名该文件。接下来，使用 startup mount 命令来安装备用数据库。在数据库已安装但未打开的情况下，使用 alter database rename file 命令来指向备用服务器上新的文件位置。最后，重新启动重做应用过程：

```
alter database recover managed standby database
      disconnect from session;
```

13.7.4 在逻辑备用数据库上执行 DDL

在本章的前面说明过，可以在逻辑备用数据库中临时禁用 Data Guard。当需要执行 DLL 操作(例如创建新的索引来改善查询性能)时，遵循相同的基本步骤：

(1) 在逻辑备用数据库上停止重做应用程序。

(2) 禁用 Data Guard。

(3) 执行 DDL 命令。

(4) 启用 Data Guard。

(5) 重启重做应用过程。

例如，为了创建一个新的索引，首先要关闭 Data Guard 特性：

```
alter database stop logical standby apply;
alter session disable guard;
```

此时，可以执行 DDL 操作。当操作完成后，重新启用 Data Guard 特性：

```
alter session enable guard;
alter database start logical standby apply;
```

然后，逻辑备用数据库将会重启它的重做应用过程，同时该索引可供其查询用户使用。

在本章中，将会学习能够显著增强数据库应用程序可用性的各种功能特性的实现细节。其中一些特性，如 LogMiner 选项，是对以前 Oracle 版本中可用的特性的增强。其他特性，如回收站和 flashback database 命令，是在 Oracle Database 10g 新引入并在 Oracle Database 11g 中得到增强。第 7 章全面涵盖了只依赖于撤销表空间的其他闪回选项，例如闪回表(Flashback Table)和闪回查询(Flashback Query)。在本章中，将会学习如何使用如下特性来增强数据库的可用性：

- Flashback Drop(闪回删除)
- Flashback Database(闪回数据库)
- LogMiner
- 联机对象重组织选项

闪回删除(Flashback Drop)依赖于 Oracle Database 10g 中引入的一种结构——回收站，回收站的行为非常类似于基于 Windows 计算机中的回收站：如果表空间中有足够的空间，则被删除的对象会恢复到它们最初的模式，所有索引和约束原封不动。闪回数据库(Flashback Database)依赖于闪回恢复区中存储的数据，闪回恢复区也是 Oracle Database 10g 中新引入的一种存储区域。从 Oracle9i 开始可用的 LogMiner 依赖归档重做日志文件来持续查看对表、索引和其他数据库结构(DDL 操作)所做的变更。

14.1 使用闪回删除来恢复被删除的表

当删除一个表(及其相关的索引、约束和嵌套表)时，Oracle 并不会立即释放该表的磁盘空间供表空间中的其他对象使用。相反，对象仍维护在回收站(recycle bin)中，直到对象被其所有者清除，或者有新的对象需要已删除对象所占用的空间。

在此例中，考虑 AUTHOR 表，它定义如下：

```
SQL> describe AUTHOR

Name                 Null?      Type
-------------------- --------- ----------------------------
AUTHORNAME           NOT NULL VARCHAR2(50)
COMMENTS                      VARCHAR2(100)
```

现在，假设意外地删除了该表。当一个用户对存在于多个环境中的一个表拥有权限，他打算在开发环境中删除一个表，但在命令执行时却实际指向了产品数据库的时候，就会出现这种情况。

```
SQL> drop table AUTHOR cascade constraints;

Table dropped.
```

如何才能恢复该表呢？自从 Oracle Database 10g 以来，删除的表并没有完全消失。它的块仍旧保持在其表空间中，并且仍旧占用空间限额。可以通过查询 RECYCLEBIN 数据字典视图来查看删除的对象。需要注意的是，在不同的版本之间 OBJECT_NAME 列的格式可能有所不同：

```
SQL> select object_name, original_name, operation, type, user,
  2  can_undrop, space from recyclebin;
```

```
OBJECT_NAME                          ORIGINAL_NAME           OPERATION
---------------------------- -------------------- ---------
TYPE                          USER            CAN_UNDROP      SPACE
------------------------ --------------- ---------- --------

BIN$OyXS+NT+J47gQKjAXwJcSA==$0 AUTH_NAME_IDX         DROP
INDEX                         HR              NO                  384

BIN$OyXS+NT/J47gQKjAXwJcSA==$0 AUTHORS               DROP
TABLE                         HR              YES                1152

SQL>
```

RECYCLEBIN 是用于 USER_RECYCLEBIN 数据字典视图的公共同义词，为当前的用户显示了回收站表项。数据库管理员可以通过 DBA_RECYCLEBIN 数据字典视图来查看所有删除的对象。

从上面的清单中可以看到，一个用户已经删除了 AUTHOR 表及其相关的主键索引。尽管删除了它们，它们仍可用于闪回。索引不能独自恢复(它的 CAN_UNDROP 列的值是'NO',同时 AUTHOR 表的 CAN_UNDROP 值为'YES')。

可以使用 flashback table to before drop 命令从回收站中恢复该表：

```
SQL> flashback table AUTHOR to before drop;

Flashback complete.
```

此时，已经恢复了该表以及它的行、索引和统计信息。

如果删除 AUTHOR 表，重新创建该表，然后再次删除它，那么会出现什么情况呢？回收站将会包含这两个表。回收站中的每个表项将会通过它的 SCN 和删除时间戳来标识。

注意：

flashback table to before drop 命令不会恢复引用的约束。

为了从回收站中清除旧的记录项，可以使用 purge 命令。可以清除所有删除的对象、数据库中所有已删除的对象(如果您是数据库管理员)、特定的表空间中的所有对象或者特定的表空间中针对某个特定用户的所有对象。

当闪回表时，可以使用 flashback table 命令的 rename to 子句对该表重命名。

在 Oracle Database 10g 和 11g 中，默认情况下，回收站是启用的。可以使用初始化参数 RECYCLEBIN 打开和关闭回收站，也可以在会话级别打开和关闭回收站，如下例所示：

```
alter session set recyclebin = off;
```

临时禁用回收站功能并不影响回收站中的当前对象。即使在回收站被禁用时，仍然可以恢复回收站中的当前对象。只有回收站被禁用时删除的对象不能恢复。

14.2 flashback database 命令

flashback database 命令将数据库返回到一个过去的时间或 SCN，提供了一种执行不完整的数据库恢复的快速替换方法。采用 flashback database 操作时，为了具有到闪回的数据库的写访问权，必须使用 alter database open resetlogs 命令再次打开它。必须拥有 SYSDBA 系统权限，以便使用 flashback database 命令。

注意：

必须已经使用 alter database flashback on 命令将数据库置于 FLASHBACK 模式。当执行该命令时，必须以独占的模式安装数据库但不打开它。

flashback database 命令的语法如下：

```
flashback [standby] database [database]
{ to {scn | timestamp} expr
| to before {scn | timestamp } expr
}
```

可以使用 to scn 或 to timestamp 子句来设置应将整个数据库闪回到的时间点。可以闪回到一个临界点(例如一个对多个表产生了未预料的结果的事务处理)之前。使用 ORA_ROWSCN 伪列来查看最近作用于行上的事务处理的 SCN。

如果还没有这样做的话，需要关闭数据库，并在启动过程中使用如下命令启用闪回：

```
startup mount exclusive;
alter database archivelog;
alter database flashback on;
alter database open;
```

注意：

在执行 alter database flashback on 命令之前，必须通过 alter database archivelog 命令启用介质恢复。

有两个初始化参数用来控制保留在数据库中的闪回数据的数量。DB_FLASHBACK_ RETENTION_TARGET 初始化参数为闪回数据库的时间程度设置上限(单位是分钟)。DB_RECOVERY_FILE_DEST 初始化参数设定闪回恢复区的大小(有关闪回恢复区设置的更多信息请参见第 12 章)。需要注意的是，flashback table 命令使用已经存储在撤销表空间中的数据(它没有创建额外的记录项)，而 flashback database 命令依赖于存储在闪回恢复区中的闪回日志。

可以通过查询 V$FLASHBACK_DATABASE_LOG 视图来确定可以闪回数据库的程度。保留在数据库中的闪回数据量由初始化参数和闪回恢复区的大小来控制。下面的清单显示了 V$FLASHBACK_DATABASE_LOG 中可用的列和内容样本：

```
SQL> describe V$FLASHBACK_DATABASE_LOG

Name                                      Null?    Type
----------------------------------------  -------- -------
OLDEST_FLASHBACK_SCN                               NUMBER
OLDEST_FLASHBACK_TIME                              DATE
RETENTION_TARGET                                   NUMBER
FLASHBACK_SIZE                                     NUMBER
ESTIMATED_FLASHBACK_SIZE                           NUMBER

SQL> select * from V$FLASHBACK_DATABASE_LOG;

OLDEST_FLASHBACK_SCN OLDEST_FL RETENTION_TARGET FLASHBACK_SIZE
-------------------- --------- ----------------- --------------
ESTIMATED_FLASHBACK_SIZE
-------------------------
             5903482 27-SEP-07              1440         8192000
                           0
```

可以通过查询 V$DATABASE 来检验数据库的闪回状态。如果已经为数据库启用了闪回，FLASHBACK_ON 列将会有一个'YES'值：

```
select current_scn, flashback_on from V$DATABASE;

CURRENT_SCN FLA
----------- ---
    5910734 YES
```

保持数据库打开超过一个小时，检验闪回数据是可用的，然后对它执行闪回—— 这会丢失在此期间发生的所有事务处理：

```
shutdown;
startup mount exclusive;
flashback database to timestamp sysdate-1/24;
```

需要注意的是，flashback database 命令要求以独占模式安装数据库，这将会影响它在任何 RAC 集群(参见第 10 章)中的分区。

当执行 flashback database 命令时，Oracle 要检查确保所有需要的归档重做日志文件和联机重做日志文件是可用的。如果日志是可用的，则将联机数据文件还原到指定的时间或 SCN。

如果在归档日志和闪回区中没有足够的联机数据，将需要使用传统的数据库恢复方法来恢复数据。例如，可能需要使用文件系统恢复方法，接着向前滚动数据。

一旦完成了闪回，必须使用 resetlogs 选项打开数据库，以便拥有到数据库的写访问权：

```
alter database open resetlogs;
```

为了关闭闪回数据库选项，当安装了数据库但未打开它时执行 alter database flashback off 命令：

```
startup mount exclusive;
alter database flashback off;
alter database open;
```

可以使用闪回选项来执行一系列操作—— 恢复旧的数据、将表还原为它早期的数据、维护各个行变化的历史记录，以及快速恢复整个数据库。如果已经配置数据库支持自动撤销管理(AUM)，那么可以极大地简化所有这些操作。另外，flashback database 命令要求修改数据库的状态。尽管这些要求给数据库管理员增加了额外的负担，但在需要的恢复数量以及完成这些恢复的速度方面可以得到显著的好处。

14.3 使用 LogMiner

Oracle 使用联机重做日志文件来跟踪对用户数据和数据字典所做的每一处改动。在恢复过程中，使用存储在重做日志文件中的信息来重新创建部分或完整的数据库。为了支持将数据库恢复到创建了数据库备份之后的一个时间点，可以维护重做日志文件的归档副本。LogMiner 实用程序提供了一种重要的视图来了解数据库中已经发生的改动。

当使用 LogMiner 时，可以看到已经做出的改动(SQL_redo)和可以用于还原这些改变的 SQL(SQL_undo)。因此，可以查看数据库的历史记录而实际上不会应用任何重做日志，并可以获得用于还原有问题的事务处理的代码。使用 LogMiner，可以指出首次出现损坏的事务处理，以便确定将合适的时间点或 SCN 用作数据库恢复的端点。

如果有少量的需要回滚的事务处理，在使用 LogMiner 之前，必须将表恢复到一个早期的状态，并且应用归档日志文件来将表前置到恰好在损坏出现之前的状态。当恢复表并应用归档日志文件时，将会有丢失随后想要保留的事务处理的风险。现在，可以使用 LogMiner 来仅仅回滚那些有问题的事务处理，而不会随后丢失有效的事务处理。

原始形式的 LogMiner 在使用上有一些限制。使用原始的方法，一次只能查看一个日志文件，并且该工具的界面使用起来很不方便。在 Oracle 9i 中，已经对该界面进行了重大改变，并大大增强了功能，包括一个和 OEM(Oracle 企业管理器)一起使用的 LogMiner Viewer(LogMiner 阅读器)。在本节中会对手动使用 LogMiner 的方法和 OEM LogMiner Viewer 予以介绍。

14.3.1 LogMiner 的工作方式

为了运行 LogMiner 实用程序，必须拥有对 DMBS_LOGMNR 程序包的 EXECUTE 权限或者 EXECUTE_CATALOG_ROLE 角色。LogMiner 需要数据字典来完全地翻译重做日志文件内容，并将内部对象标识符和数据类型转换为对象名和外部数据格式。如果不能使用数据字典，LogMiner 将会返回以十六进制格式标识的数据和以内部对象 ID 表示的对象信息。

有三种选择来获得一个供 LogMiner 使用的数据字典：

- 将数据字典信息提取到一个平面文件中。
- 将数据字典提取到重做日志文件中。
- 从当前的数据库中使用联机数据字典。

LogMiner 分析通常要求使用的数据字典源自于产生重做日志文件的同一个数据库。但是，如果正在使用平面文件格式或者正在使用源自重做日志文件的数据字典，则可以从 LogMiner 正在其上运行的数据库或从另一个数据库来分析重做日志文件。但是，如果正从当前的数据库中使用联机目录，只能从当前的数据库来分析重做日志文件。

由于可以从一个数据库中依据另一个数据库中的重做日志文件来运行 LogMiner，这两个数据库上使用的字符集必须匹配。硬件平台也必须和生成重做日志文件时采用的平台相匹配。

14.3.2 提取数据字典

将数据字典提取到平面文件中的一个潜在问题是，当正在提取数据字典的时候，其他人可能正在发送 DDL 语句。因此，提取出的数据字典可能和数据库不同步。相比于使用重做日志文件，使用平面文件来存储数据字典时需要更少的系统资源。

当提取数据字典到重做日志文件时，在提取数据字典的过程中不能处理 DDL 语句。因此，字典将会和数据库同步。提取过程更加耗费资源，但是更为迅速。

为了将数据字典提取到平面文件或重做日志文件中，可以使用 DBMS_LOGMNR_D.BUILD 程序。数据字典文件放置在一个目录中。因此，必须拥有放置该文件的目录的写权限。为了定义目录的位置，使用初始化参数 UTL_FILE_DIR。例如，为了指定位置 D:\Oracle\Ora10\database 作为 LogMiner 的输出位置，将以下的记录项放在参数文件中：

```
UTL_FILE_DIR= D:\Oracle\Ora10\database
```

注意：

不能使用 alter system 命令动态地改变 UTL_FILE_DIR 参数。必须修改初始化文件，然后停止并重启数据库。

为了执行 DBMS_LOGMNR_D.BUILD 程序，必须为目录指定文件名，为文件指定目录路径名，并指定希望将目录写入到平面文件中还是重做日志文件中。为了将数据字典提取到位于 G:\Oracle\Ora10\database 目录中的文件名为 mydb_dictionary 的平面文件中，可以发出如下命令：

```
execute DBMS_LOGMNR_D.BUILD
('mydb_dictionary.ora',
'G:\Oracle\Ora10\database',
options=>DBMS_LOGMNR_D.STORE_IN_FLAT_FILE);
```

可以使用 DBMS_LOGMNR_D.STORE_IN_FLAT_FILE 作为其他的可选项。

一旦将字典存储在平面文件中，可以将它复制到另一个平台来运行 LogMiner。可能需要运行其他数据库上的 dbmslmd.sql 来建立正确的环境。可以在 Unix 系统上的$ORACLE_HOME\rdbms\admin 目录中找到 dbmslmd.sql 文件。

14.3.3 分析一个或多个重做日志文件

为了使用 LogMiner 分析重做日志文件，遵照以下步骤：

(1) 使用 V$LOGMNR_LOGS 获得一个可用的重做日志文件的清单。

(2) 使用 DBMS_LOGMNR.START_LOGMNR 程序启动 LogMiner 实用程序。参见本节后面的表 14-2 来了解 START_LOGMNR 参数。

(3) 查询 V$LOGMNR_CONTENTS 来查看结果。

(4) 一旦已经完成对重做日志的查看，发出如下命令来结束会话：

```
execute DBMS_LOGMNR.END_LOGMNR;
```

表 14-1 中介绍了 DBMS_LOGMNR 程序包可用的子程序。

表 14-2 显示了 START_LOGMNR 程序的参数。

为了创建一个可用于分析的重做日志文件的清单，如下所示运行带有 NEW 选项的 DBMS_LOGMNR.ADD_LOGFILE 程序，此例使用 Linux 文件系统：

```
execute DBMS_LOGMNR.ADD_LOGFILE(
LogFileName=> '/oracle/ora10/redo01.ora',
Options=> DBMS_LOGMNR.NEW);
execute DBMS_LOGMNR.ADD_LOGFILE(
LogFileName=> '/oracle/ora10/redo02.ora',
Options=> DBMS_LOGMNR.NEW);
```

表 14-1　DBMS_LOGMNR 子程序

子　程　序	说　　明
ADD_LOGFILE	向要处理的归档文件清单添加一个文件
START_LOGMNR	初始化 LogMiner 实用程序
END_LOGMNR	完成并结束一个 LogMiner 会话
MINE_VALUE (函数)	对于任何从 V$LOGMNR_CONTENT 返回的行，返回由 COLUMN_NAME 参数指定的列名的撤销或重做列的值
COLUMN_PRESENT (函数)	对于任何从 V$LOGMNR_CONTENT 返回的行，决定是否存在由 COLUMN_NAME 参数指定的列名的撤销或重做列的值
REMOVE_LOGFILE	从 LogMiner 将要处理的日志文件清单中删除一个日志文件

表 14-2　START_LOGMNR 选项的值

选　　项	说　　明
COMMITTED_DATA_ONLY	如果设定了这个选项，只返回对应于提交的事务处理的 DML
SKIP_CORRUPTION	在从 V$LOGMNR_CONTENTS 选择的过程中，跳过重做日志文件中遇到的任何损坏的块。只有在实际的重做日志文件中存在一个损坏的块时，该选项才有用，如果数据头块损坏，则该选项不起作用
DDL_DICT_TRACKING	如果发生一个 DDL 事件，启用 LogMiner 来更新内部数据字典，以确保 SQL_REDO 和 SQL_UNDO 信息被维护并保持正确
DICT_FROM_ONLINE_CATALOG	指示 LogMiner 使用联机数据字典来代替平面文件或重做日志文件存储的字典
DICT_FROM_REDO_LOGS	指示 LogMiner 使用存储在一个或多个重做日志文件中的数

	据字典
NO_SQL_DELIMITER	指示 LogMiner 在重构的 SQL 语句末尾不插入 SQL 定界符(;)
NO_ROWID_IN_STMT	指示 LogMiner 在重构的 SQL 语句中不包含 ROWID 语句
PRINT_PRETTY_SQL	指示 LogMiner 格式化重构的 SQL 语句以便于阅读
CONTINUOUS_MINE	指示 LogMiner 自动添加重做日志文件来找到感兴趣的数据。
	指定起始 SCN、日期或要挖掘的第一个日志。LogMiner 必须连接到正在生成重做日志文件的同一个数据库实例

可以指定数据字典文件的位置，如下所示：

```
execute DBMS_LOGMNR.ADD_LOGFILE(
DictFileName=> '/oracle/ora10/dictionary.ora',
```

在已经告知 LogMiner 数据字典的位置并添加了重做日志文件后，可以使用 DBMS_LOGMNR.START_LOGMNR 程序包开始分析重做日志文件。例如，以下的命令在一段时间内分析日志文件：

```
execute DBMS_LOGMNR.START_LOGMNR(
DictFileName => '/oracle/dictionary.ora',
StartTime => TO_DATE('01-SEP-2007 12:45:00', DD-MON-YYYY HH:MI:SS')
EndTime => TO_DATE('01-OCT-2007 00:00:00', DD-MON-YYYY HH:MI:SS'));
```

注意：

使用时间戳将不能保证重做记录的次序，必须使用 SCN 编号来确保记录的次序。

可以使用 SCN 值来筛选数据，如下所示：

```
execute DBMS_LOGMNR.START_LOGMNR(
DictFileName => '/oracle/dictionary.ora',
StartScn => 125,
EndScr => 300);
```

如果没有输入起始和结束时间或者 SCN 编号范围，对于发出的每条 select 语句，将读取整个文件。

为了查看重做代码和取消代码，可以选择 SQL_REDO 和 SQL_UNDO 列，如下所示：

```
select sql_redo, sql_undo
 from V$LOGMNR_CONTENTS;
```

可以使用 OEM 服务器管理控制台(OEM Server Manager Console)启动基于 Java 的 LogMiner Viewer 来查看重做日志和归档的重做日志。为了在 Windows 平台上启动 LogMiner Viewer，使用 Start | Programs | Oracle_Home | Oracle Enterprise Manager Console 选项。一旦已经连接到基于 Java 的 OEM Server Console(Oracle Database 10g 及更早版本)，选择希望在它上面运行 LogMiner Viewer 的数据库。要确保已经启动了该数据库。

为了启动 LogMiner Viewer，突出显示数据库并右击。将光标移动到 Related Tools 选项，然后移到 LogMiner Viewer 选项。当出现 LogMiner Viewer Console 屏幕时，通过单击图标面板中顶端的图标或从 Object 下拉菜单中选择 Create Query 来创建一个对象查询。LogMiner Viewer 自动查找用来创建查询的可用的归档重做日志文件。如果没有可用的归档重做日志文件，会接收到错误消息。可以创建筛选选项(通过创建查询标准)，查看每个可用的重做日志文件的起始和结束 SCN，并选择要显示的列。OEM LogMiner Viewer 可以简化筛选日志文件内容的过程。此外，可以使用 Grid Control 屏幕来访问和查看 LogMiner 的输出。

14.3.4 Oracle Database 10g 中引入的 LogMiner 特性

如果使用过 Oracle Database 10g 之前版本的 LogMiner，下面是一些现在可用的增强特性：

如前面的表 14-1 所示，现在 DBMS_LOGMNR 有一个 REMOVE_LOGFILE 程序，它可以从要分析的列表中删除文件。不应该再使用 ADD_LOGFILE 程序的 REMOVEFILE 选项。

可以使用 START_LOGMNR 的 NO_ROWID_IN_STMT 选项(参见表 14-2)从重构的 SQL 命令中筛选出 rowid 子句。

可以通过 alter database 命令来扩充补充日志，使补充日志包含外键或行的所有变化。使用这些设置将会增加写入到重做日志文件中的数据量。

可以在表级别扩充补充日志来跟踪主键、外键、唯一的索引和所有的变化。还可以使用 no log 选项

来防止记录用户定义的日志组中的列。

参见 Oracle Utilities(Oracle 实用程序)指南来进一步了解 LogMiner 及其程序的使用细节。

14.3.5 Oracle Database 11g 中引入的 LogMiner 特性

在 Oracle Database 11g 之前，数据库管理员不得不使用基于 Java 的 LogMiner 控制台，它很难安装，且不能与 Oracle Enterprise Manager Database Control 完全集成。通过集成基于任务的日志挖掘操作与 Flashback Transaction，LogMiner 与 OEM DB Control 的集成进一步增强了 LogMiner 的易使用性。图 14-1 展示了 LogMiner 的 OEM 界面。

图 14-1 OEM LogMiner 和 Flashback Transaction 界面

14.4 联机对象重组织

可以联机重组织许多数据库对象，可用的选项如下：

- 联机创建索引
- 联机重建索引
- 联机合并索引
- 联机重建按索引组织的表
- 联机使用 DBMS_REDEFINITION 程序包来重新定义表

在以下的各小节中，可以看到以上每种操作的示例。

14.4.1 联机创建索引

在终端用户可以访问基本表时可以创建和重建索引。在创建索引的同时不允许 DDL 操作。为了联机建立索引，使用 create index 命令的 online 子句，如下面的示例所示：

```
create index AUTH$NAME on AUTHOR (AuthorName) online;
```

14.4.2 联机重建索引

当使用 alter index 命令的 rebuild 子句时，Oracle 使用现有的索引作为新索引的数据源。因此，在进行操作时必须有足够的空间来存储索引的两个副本。可以使用 alter index rebuild 命令来改变一个索引的存储特征和表空间分配。

为了联机重建索引，使用 alter table 命令的 rebuild online 子句，如下面的示例所示：

```
alter index AUTH$NAME rebuild online;
```

14.4.3 联机合并索引

可以合并索引来收回索引内的空间。当合并索引时，不能将它移动到另一个表空间(对于重建索引是允许的)。合并不需要用于存储索引多个副本的空间，因此当试图在空间受限的环境下重组织索引时，这种方法可能是有用的。

为了合并索引，使用 alter index 命令的 coalesce 子句。所有的索引合并都是联机操作。下面是一个合并例子：

```
alter index AUTH$NAME coalesce;
```

14.4.4 联机重建以索引组织的表

可以使用 alter table … move online 命令来联机重建一个以索引组织的表。在存在溢出的数据段的情况下，如果指定 overflow 关键字，会重建该数据段。例如，如果 BOOKSHELF 是一个索引组织的表，可以通过如下的命令联机重建它：

```
alter table BOOKSHELF move online;
```

当使用该命令时，不能执行并行的 DML。另外，移动联机选项只适用于非分割的索引组织的表。

14.4.5 联机重新定义表

当应用程序用户可以访问表的时候，可以改变它的定义。例如，当正在使用一个表时，可以分割以前未分割的表——这是一项对高可用 OLTP 应用程序很有用的功能。

从 Oracle Database 11g 开始，对于不能联机重新定义的表的类型只有很少的限制。下面列出了关键限制：

- 重新定义具有物化视图日志的表之后，必须完全刷新依赖的物化视图。
- 索引组织表(IOT)的溢出表必须与基本 IOT 同时重新定义。

- 具有细粒度访问控制的表不能联机重新定义。
- 包含 BFILE 列的表不能联机重新定义。
- 包含 LONG 和 LONG RAW 列的表可以重新定义，但必须将 LONG 和 LONG RAW 列转换为 CLOB 和 BLOB。
- SYS 和 SYSTEM 模式中的表不能联机重新定义。
- 临时表不能联机重新定义。

下面的示例说明了联机重新定义一个表需要的步骤。首先，要检验能够重新定义该表。对于这个示例，将会在 SCOTT 模式下创建 CUSTOMER 表，然后对其重新定义：

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

接下来，通过执行 DBMS_REDEFINITION 程序包的 CAN_REDEF_TABLE 程序来检验能够重新定义该表。它的输入参数是用户名和表名：

```
execute DBMS_REDEFINITION.CAN_REDEF_TABLE('SCOTT','CUSTOMER');
```

如果程序返回如下消息，该表成为联机重新定义的候选对象：

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

如果它返回一个错误，不能联机重新定义该表，并且错误消息会给出原因。

接下来，在相同的模式下创建一个临时表，它具有重新定义的表的期望属性。例如，可以分割 CUSTOMER 表(为了简化该示例，不显示用于分割的 tablespace 和 storage 子句)：

```
create table CUSTOMER_INTERIM
(Name     VARCHAR2(25) primary key,

Street   VARCHAR2(50),
 City     VARCHAR2(25),
 State    CHAR(2),
 Zip      NUMBER)
partition by range (Name)
 (partition PART1  values less than ('L'),
  partition PART2  values less than (MAXVALUE))
;
```

现在可以执行 DBMS_REDEFINITION 程序包的 START_REDEF_TABLE 程序来开始重新定义过程。它的输入变量是模式所有者、将要重新定义的表、临时表名称和列映射(类似于一个选择查询中的列名称列表)。如果没有提供列映射，那么原始表和临时表中的所有列名和定义必须相同。

```
execute DBMS_REDEFINITION.START_REDEF_TABLE -
 ('SCOTT','CUSTOMER','CUSTOMER_INTERIM');
```

接下来,在临时表上创建要求的任何触发器、索引、授权或约束。在该示例中,已经在 CUSTOMER_INTERIM 上定义了主键，此时在重新定义过程中可以添加外键、二级索引和授权。不允许创建外键，直到完成了重新定义过程。

注意：

为了避免手动操作步骤，可以使用 COPY_TABLE_DEPENDENTS 程序在临时表上创建所有相关的对象。通过这种方法支持的相关对象包括触发器、索引、授权和约束。

当重新定义过程完成时，临时表上的索引、触发器、约束和授权将会替代原始表上的对应对象。此时会启用临时表上禁用的引用约束。

为了完成重新定义，执行 DBMS_REDEFINITION 程序包的 FINISH_REDEF_TABLE 程序。它的输入参数是模式名、原始的表名称和临时表名称：

```
execute DBMS_REDEFINITION.FINISH_REDEF_TABLE -
 ('SCOTT','CUSTOMER','CUSTOMER_INTERIM');
```

可以通过查询数据字典来检验重新定义：

```
select table_name, high_value
 from DBA_TAB_PARTITIONS
 where owner = 'SCOTT';
TABLE_NAME              HIGH_VALUE
----------------------- ------------------------
CUSTOMER                MAXVALUE
CUSTOMER                'L'
```

在执行了 START_REDEF_TABLE 程序后，为了中断此过程，执行 ABORT_REDEF_TABLE 程序(输入参数是模式、原始的表名称和临时表名称)。

[U1]此图已变，请置换新图

[U2]此图已变，请置换新图

[U3]此图中还有些内容需要译出来，见图下面

[U4]此图已变，请置换新图

[U5]此图已变，请置换新图

[U6]此图已变，请置换新图

[U7]此图已变，请置换新图

[U8]此图已变，请置换新图

[U9]此图已变，请置换新图

[U10]请置换新图

[U11]此图已变，请置换新图

[U12]此图已变，请置换新图

[U13]请置换新图

[U14]请置换新图

[U15]请置换新图

[U16]请插入图
[U17]此图已变，请置换新图
[U18]此图已变，请置换新图
[U19]此图已变，请置换新图
[U20]此图已变，请置换新图
[U21]此图已变，请置换新图
[U22]此图已变，请置换新图
[U23]此图已变，请置换新图
[U24]此图已变，请置换新图
[U25]此图已变，请置换新图
[U26]请插入图
[U27]此图已变，请置换新图
[U28]此图已变，请置换新图