



**ORACLE** Oracle 数据库技术丛书(1)

# ORACLE 8.1.6

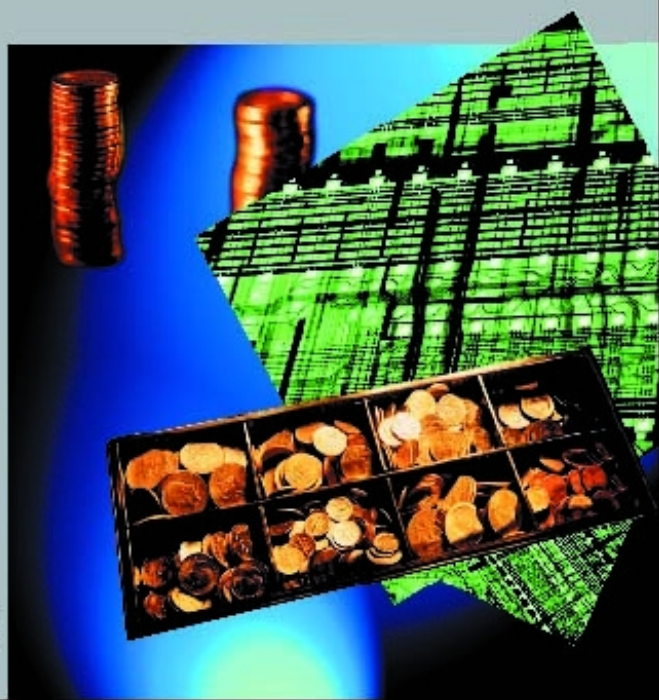
Administrator's Guide

## 管理员指南

本丛书编委会 主编



北京希望电子出版社  
Beijing Hope Electronic Press  
[www.bhep.com.cn](http://www.bhep.com.cn)



Oracle 数据库技术丛书 (1)

# Oracle 8.1.6 管理员指南

本丛书编委会 编写



**北京希望电子出版社**

Beijing Hope Electronic Press

**[www.bhp.com.cn](http://www.bhp.com.cn)**

## 内 容 简 介

本书是“Oracle 数据库技术丛书”之一，全套书共 5 册。本书全面讲述了基本的 Oracle 数据库管理、配置以及存储，数据库的安全和资源管理等知识。

全书共分 6 个部分。第一部分：基本的数据库管理，其中包括 Oracle 8i 数据库管理员的典型任务，Oracle 8i 数据库的启动、创建和关闭；第二部分：数据库的配置，其中讲述了过程、控制文件、重复日志以及作业序列的管理；第三部分是数据库存储；第四部分：规划对象，内容包括表、索引、簇以及模式对象的管理；第五部分：数据库的安全，讲述了安全策略的建立、用户与资源的管理以及审计数据库的使用；第六部分：数据库资源管理，描述了如何使用数据库资源管理器分配资源。

本书内容全面、系统，由浅入深，既可以作为数据库管理员、应用程序管理员的参考书，也可以作为高等院校相关专业教学、自学用教材以及社会各种 Oracle 系统培训用教材。

本书光盘内容包括与本书配套的电子书。

- 系 列 名: Oracle 数据库技术丛书 (1)  
书 名: Oracle 8.1.6 管理员指南  
文 本 著 者: 本丛书编委会 编写  
C D 制 作 者: 希望多媒体创作中心  
C D 测 试 者: 希望多媒体测试部  
责 任 编 辑: 柴文强 郑明红 周 艳 郭淑珍  
出版、发行者: 北京希望电子出版社  
地 址: 北京海淀路 82 号 100080  
网址: [www.bhp.com.cn](http://www.bhp.com.cn) E-mail: [lwm@hope.com.cn](mailto:lwm@hope.com.cn)  
电话: 010-62562329,62541992,62637101,62637102,62633308,62633309  
(发行和技术支持)  
010-62613322-215 (门市) 010-62531267 (编辑部)  
经 销: 各地新华书店、软件连锁店  
排 版: 希望图书输出中心  
C D 生 产 者: 北京中新联光盘有限责任公司  
文 本 印 刷 者: 北京双青印刷厂  
规 格 / 开 本: 787×1092 1/16 开本 23 印张 527 千字  
版 次 / 印 次: 2000 年 10 月第 1 版 2000 年 10 月第 1 次印刷  
印 数: 0001~5000 册  
本 版 号: ISBN 7-900044-93-0/TP·93  
定 价: 50.00 元(1CD, 含配套书)

说明: 凡我社图书及其配套光盘若有缺页、倒页、脱页、自然破损, 本社负责调换。

# Oracle 数据库技术丛书

## 编 委 会 名 单

主 编：奥克拉·马尔

副主编：夏尔·杰克 本杰明·卡里基 沈 鸿

编 委：（按姓氏笔划排序）

纪 红 刘大伟 刘晓融 海尔默·劳尔

基尔斯腾·扬 陆卫民 张长富 陈河南

卡里亚诺夫·波尔加 柯里奇·金巴

徐建华 袁勤勇 帕金斯·赫尔墨斯

本书执笔人：袁勤勇 邓静 李学群等

## 序

Oracle 系统以其先进的数据库、服务器、企业商务应用技术和丰富的应用开发与决策支持工具，以及针对设计、开发、客户三方的电子商务解决方案和 Internet 应用，在国内企事业单位、科研院校得到广泛应用。为满足 Oracle 技术的深入应用所带来的对技术资料的要求，我们与国内外有关专家共同组织编写了《Oracle 数据库技术丛书》。全书共 5 册，全面反映了 Oracle 最新技术和应用。它们分别如下：

(1)《Oracle 8.1.6 管理员指南》：全面讲述了 Oracle 数据库基本管理、配置、存储、数据库的安全和资源管理等知识。

本书共分 6 个部分。第一部分“基本的数据库管理”，其中包括 Oracle 8i 数据库管理员的典型任务，Oracle 8i 数据库的启动、创建和关闭；第二部分“数据库的配置”，其中讲述了过程、控制文件、重复日志以及作业序列的管理；第三部分介绍数据库存储；第四部分“规划对象”，内容包括表、索引、簇以及模式对象的管理；第五部分“数据库安全”，讲述了安全策略的建立、用户与资源的管理以及审计数据库的使用；第六部分“数据库资源管理”，描述了如何使用数据库资源管理器分配资源。

(2)《Oracle 8.1.6 系统安全与网络管理指南》：主要讲述了 Oracle 的高级网络安全特性。

本书共分两篇。第一篇“Oracle 高级安全管理指南”，分为四个部分，全面详细介绍了关于 Oracle 8i 数据库平台高级安全特性的相关机制。内容主要涵盖 Oracle 高级安全特性、Oracle 分布式计算环境（DCE）集成和 Oracle 安全/目录集成等主题共计 20 章，以及 6 个附录。第一部分介绍 Oracle 高级安全特性，第二部分介绍 Oracle DCE 集成，第三部分介绍 Oracle 8i 安全/目录集成，第四部分是附录及一个词汇表。本书第二篇“Oracle Net8 管理指南”，讲述了 Net8 的使用以及如何利用 Net8 来配置 Oracle 的高级安全特性。该篇分四个部分，共计 12 章和一个附录。主要内容包括：Net8 简介，Net8 概念，Net8 体系结构，Net8 产品和工具，规划网络、配置命名方法，配置监听者，配置多线程服务器，打开 Net8 程序员增强特性，建立连接和测试网络、问题解决和控制实用程序。通过本书的学习，读者可以熟练掌握 Oracle 高级安全特性的配置和管理。

(3)《Oracle 8.1.6 核心管理技术大全》：全面系统地介绍了 Oracle 8i 数据仓库技术的基本知识和使用方法，Oracle WebDB 的使用环境以及它在建立应用程序和创建 Oracle Web 站点两个方面的应用，并对如何管理和使用 Oracle 分布式数据库系统进行了详细描述。

本书由三篇构成。第一篇“Oracle 8i 数据仓库技术”，共分 5 个部分。第一部分介绍了数据仓库的基本概念；第二部分讨论了有关数据仓库的逻辑设计问题；第三部分讲述了有关数据仓库的物理设计问题；第四部分描述了处理管理数据仓库的任务，包括 ETT（提取、传输、转换）以及装载和刷新；第五部分讨论了有关提高数据仓库性能的方法，包括模式、SQL 分析、调整并行执行以及查询重写等，并在本篇最后给出了一个中英文对照的术语表。第二篇“Oracle 8i 分布式数据库系统”，由 3 部分组成。第一部分介绍了分布式数据库的基本概念，同时说明了 Oracle 分布式数据库系统的特点和用法，以及如何管理、开发分布式数据库系统应用程序的方法和途径；第二部分阐述了分布式事务、两阶段提交的实现和分布式事务的管理；第三部分描述了异构分布式事务的概念和管理方法。第三篇“Oracle Web 数据库应用”由 4 部分构成，第一部分详细介绍了建立应用程序所涉及的各个环节，包括如何创建新的用户和角色，如何在数据库中建立各种数据库对象以及如何创建组件等；第二部分分别叙述了 WebDB 站点的有关创建、设计和管理方面的问

题，包括如何利用各种向导和导航条进行操作，如何控制用户对站点中各种文件夹的访问；第三部分主要讨论了如何控制用户对报表的访问，包括对报表服务器和打印机的访问；第四部分附录说明了完成本篇练习的前提。

(4)《Oracle 8.1.6 开发指南》：主要介绍了 Oracle 8i 数据库性能设计与优化，以及数据库管理的备份和恢复技术。

本书由两篇构成。第一篇“Oracle 8i 性能设计与优化”分 4 部分共 24 章。第一部分“性能优化知识”，包括理解 Oracle 性能优化，性能优化方法两章；第二部分“设计人员和开发人员的应用程序设计优化技术”，包括应用程序和系统性特性，优化性能，EXPLAIN PLAN 使用说明，SQL Trace 和 TKPROF 使用说明，优化程序提示使用说明，收集统计信息，优化 SQL 语句和 Plain Stability 使用说明等七章；第三部分“设计人员和 DBA 应用程序工具”，包括诊断工具概述，数据访问方法，管理共享 SQL 和 PL/SQL 区域，Oracle Trace 使用说明，动态性能视图，诊断系统性能故障和事务模式等六章；第四部分“优化实例性能”，包括优化 CPU 资源，优化内存分配，优化 I/O，优化资源争用，优化网络，优化操作系统和优化实例恢复性能等七章。本篇内容丰富，涵盖了大型数据库性能设计与优化中的大部分内容，具有很强的启发性，可谓既有广度，又有相当的深度。第二篇“Oracle 8i 的备份与恢复”分两部分共 7 章。第一部分“开发备份和恢复战略”，包括备份和恢复概述，管理数据结构和开发备份和恢复战略等三章；第二部分“实施操作系统备份和恢复”，包括实施操作系统备份，实施介质管理，介质管理实例，实施操作系统表空间的时间点恢复共四章。本篇详细讲述了 Oracle 8i 数据库的备份和恢复的方方面面，内容全面、新颖，结构清晰，示例丰富翔实，实用性很强，是广大用户掌握大型数据库系统理论和实践的首选用书。

(5)《Oracle 8.1.6 应用系统使用指南》：由“Oracle 销售分析系统管理指南”和“Oracle 现金管理系统应用”两篇组成，介绍了 Oracle 的两个应用系统。

本书第一篇阐述了 Oracle 销售分析器的基本概念、安装方法及其性能。该篇由 14 章和两个附录组成，内容包括：Oracle 销售分析器介绍，管理员的职责，分析器的安装，构造数据库及片断，使用 Express 数据库和分类，建立客户访问、Web 访问、数据库安全机制，为客户访问建立通信，限定对数据库的访问范围，定制数据库、客户端界面，以及控制数据显示等，附录给出了分类描述和初始化文件描述。第二篇讨论了一个 Oracle 财务管理软件的应用，共分 6 章，旨在指导读者如何更有效地使用“Oracle 现金管理系统”软件。内容包括：现金管理系统过程的概述，现金管理系统设置，加载银行报告书，调节银行报告书，现金预测，查询和报表等；附录 A、B、C 列出了关于导航路径、简述选项和功能安全性等信息。

本丛书内容全面新颖，结构清晰，叙述详细，事例翔实，技术内涵高，指导性强，既可以作为 Oracle 数据库系统管理员、网络安全管理员、应用与开发人员、维护和技术支持人员必备的技术指导书，也可以作为高等院校相关专业教学、自学用教材和社会各种 Oracle 系统培训用教材。

我们特别感谢美国伯克利大学计算机系奥克拉·马尔教授，Oracle 加拿大研究中心高级研究员夏尔·杰克教授和美国卡内基梅隆大学计算机系本杰明·卡里基教授，本丛书就是在他们的大力帮助和协调下才得以完成。感谢 Oracle 美国研究中心高级研究员海尔默·劳尔教授、卡内基梅隆大学计算机系基尔斯腾·扬教授、Oracle 公司数据仓库专家卡里亚诺夫·波尔加博士、美国麻省理工大学计算机系帕金斯·赫尔墨斯教授和柯里奇·金巴教授，由于他们的技术指导和全力参与，本丛书才得以及时完稿。真诚感谢参与本丛书编写的全体专家和技术人员，是他们的积极配合和努力，才使本丛书如期付梓出版。最后，向本丛书的读者表示诚挚的谢意，感谢你们的相知。

Oracle 数据库技术丛书编委会

2000 年 8 月 15 日

# 总目录

## 第一部分 基本数据库管理

### 1 Oracle 数据库管理员

- 1.1 Oracle 用户的类型
- 1.2 数据库管理安全性和权限
- 1.3 数据库管理员认证
- 1.4 密码文件管理
- 1.5 数据库管理员实用程序
- 1.6 数据库管理员的优先权
- 1.7 区别 Oracle 数据库软件版本

### 2 创建 Oracle 数据库

- 2.1 创建数据库之前应考虑的事项
- 2.2 Oracle 数据库配置助理(DBCA)
- 2.3 手工创建 Oracle 数据库
- 2.4 安装参数
- 2.5 数据库创建之后应考虑的事项
- 2.6 初始化调整指导

### 3 启动与关闭

- 3.1 启动数据库
- 3.2 更改数据库的可用性
- 3.3 关闭数据库
- 3.4 挂起和继续执行数据库
- 3.5 使用初始化参数文件

## 第二部分 Oracle 服务器配置

### 4 管理 Oracle 进程

- 4.1 服务器进程
- 4.2 配置多线程服务器的 Oracle

- 4.3 跟踪 Oracle 后台进程
- 4.4 管理并行查询选项的进程
- 4.5 管理外部应用程序的进程
- 4.6 结束会话

## 5 管理控制文件

- 5.1 什么是控制文件
- 5.2 控制文件指南
- 5.3 创建控制文件
- 5.4 创建控制文件之后的故障排除
- 5.5 删除控制文件

## 6 管理联机重做日志

- 6.1 什么是联机重做日志
- 6.2 规划联机重做日志
- 6.3 创建联机重做日志组和成员
- 6.4 联机重做日志的重命名和重定位
- 6.5 删除联机重做日志组和成员
- 6.6 强制日志切换
- 6.7 校验重做日志文件中的块
- 6.8 清除联机重做日志文件
- 6.9 列出关于联机重做日志的信息

## 7 管理存档重做日志

- 7.1 什么是存档重做日志
- 7.2 在存档日志和非存档日志之间选择
- 7.3 控制存档模式
- 7.4 确定存档目标
- 7.5 指定日志传输模式
- 7.6 管理存档目标失败
- 7.7 调整存档性能
- 7.8 显示存档重做日志信息
- 7.9 控制由存档日志程序产生的踪迹输出
- 7.10 使用日志采集器来分析联机重做日志和存档重做日志

## 8 管理作业队列

- 8.1 SNP 后台进程
- 8.2 管理作业队列
- 8.3 查看作业队列信息



## 第三部分 数据库存储

### 9 表空间管理

- 9.1 表空间管理的指南
- 9.2 创建表空间
- 9.3 管理表空间的地址分配
- 9.4 改变表空间的工作效率
- 9.5 只读表空间
- 9.6 删除表空间
- 9.7 使用 `DBMS_SPACE_ADMIN` 程序包
- 9.8 在数据库之间移动表空间
- 9.9 查看表空间信息

### 10 管理数据文件

- 10.1 管理数据文件的方针
- 10.2 向一个表空间创建和增加数据文件
- 10.3 变更数据文件的尺寸
- 10.4 改变数据文件的可用性
- 10.5 重命名和重新部署数据文件
- 10.6 校验数据文件中的数据块
- 10.7 查看数据文件的相关信息

### 11 管理回退段

- 11.1 管理回退段的方针
- 11.2 创建回退段
- 11.3 改变回退段
- 11.4 明确地给回退段分派事务处理
- 11.5 删除回退段
- 11.6 监视回退段信息

## 第四部分 模式对象

### 12 模式对象管理指南

- 12.1 管理数据块的空间
- 12.2 事务项设置 (`INITRANS` 和 `MAXTRANS` 参数)
- 12.3 设置储存参数

- 12.4 释放空间
- 12.5 了解数据类型的空间使用

## 13 管理数据表

- 13.1 管理数据表的方针
- 13.2 创建表
- 13.3 改变表
- 13.4 删除表
- 13.5 索引管理表

## 14 管理索引

- 14.1 索引管理指南
- 14.2 创建索引
- 14.3 改变索引
- 14.4 监视索引的空间使用
- 14.5 删除索引

## 15 管理分区数据表和索引

- 15.1 什么是分区的表和索引
- 15.2 分区方法
- 15.3 创建分区
- 15.4 维护分区
- 15.5 分区表和索引实例

## 16 管理簇

- 16.1 管理簇指南 7
- 16.2 创建簇
- 16.3 改变簇
- 16.4 删除簇

## 17 管理散列簇

- 17.1 是否应该使用散列簇
- 17.2 创建散列簇
- 17.3 改变散列簇
- 17.4 删除散列簇

## 18 管理视图、序列和别名单元

- 18.1 管理视图
- 18.2 管理序列

### 18.3 管理别名单元

## 19 模式对象的常规管理

### 19.1 在单一操作中创建多表和多视图

### 19.2 模式对象重命名

### 19.3 分析表、视图和簇

### 19.4 截断表和簇

### 19.5 启用和禁用触发器

### 19.6 管理完整性约束

### 19.7 管理对象相关性

### 19.8 管理对象名称解析

### 19.9 为数据字典改变存储参数

### 19.10 显示有关模式对象的信息

## 20 解决数据块讹误的问题

### 20.1 修正数据块讹误的选项

### 20.2 关于 DBMS\_REPAIR 程序包

### 20.3 使用 DBMS\_REPAIR 程序包

### 20.4 DBMS\_REPAIR 示例

## 第五部分 数据库安全

## 21 建立安全策略

### 21.1 系统安全策略

### 21.2 数据安全策略 (Data SecurityPolicy)

### 21.3 用户安全策略

### 21.4 口令管理策略 (Password Management Policy)

### 21.5 审计策略

## 22 用户和资源管理

### 22.1 会话和用户许可

### 22.2 用户身份验证

### 22.3 Oracle 用户

### 22.4 使用概要文件管理资源

### 22.5 列举数据库用户和 PROFILE 的相关信息

### 22.6 示例

## 23 用户权限管理和角色管理

- 23.1 识别用户权限
- 23.2 管理用户角色
- 23.3 授权用户权限和角色
- 23.4 取消用户权限和角色
- 23.5 授予权限和取消权限何时生效
- 23.6 运用操作系统或运用网络授权
- 23.7 权限和角色信息列表

## 24 数据库使用审计

- 24.1 审计准则
- 24.2 创建和删除数据库审计踪迹试图
- 24.3 管理审计踪迹信息
- 24.4 查看数据库审计踪迹信息
- 24.5 通过数据库触发器进行审计

## 第六部分 数据库资源管理

## 25 数据库资源管理器

- 25.1 什么是数据库资源管理器
- 25.2 管理数据库资源管理器
- 25.3 创建和管理资源规划
- 25.4 管理资源用户组
- 25.5 启用数据库资源管理器
- 25.6 举例
- 25.7 数据库资源管理器视图 ....

# 第一部分 基本数据库管理

---

第一部分介绍数据库管理职责的概况，并描述了数据库的创建，以及怎样启动和关闭数据库事例。它包含下列三章：

- 第 1 章 “Oracle 数据库管理员”
- 第 2 章 “创建 Oracle 数据库”
- 第 3 章 “启动和关闭”

## 1 Oracle 数据库管理员

本章描述管理 Oracle 服务器的人——数据库管理员的职责。  
包括下列专题：

- Oracle 用户类型
- 数据库管理员的安全性和权限
- 数据库管理员授权
- 密码文件管理
- 数据库管理员实用程序
- 数据库管理员的优先级
- 标识 Oracle 数据库软件版本

### 1.1 Oracle 用户的类型

在用户站点，用户类型及其职责可能不同。例如，在一个大型站点上，数据库管理员的职责可以分给几个人。

本节包括下列专题：

- 数据库管理员
- 安全官员
- 应用程序开发人员
- 应用程序管理员
- 数据库用户
- 网络管理员

#### 1.1.1 数据库管理员

因为 Oracle 数据库系统可以非常大，且拥有很多用户，所以必须有人或有一组人管理

系统。数据库管理员（DBA）就是这样的管理人员。每一个数据库至少需要一个人来承担管理职责。

数据库管理员的职责包括：

- 安装并更新 Oracle 服务器和应用程序工具。
- 分配系统存储，为数据库系统规划将来的存储需求。
- 在应用程序开发人员设计了应用程序以后，创建主数据库存储结构（表空间）。
- 一旦应用程序开发人员完成应用程序的设计以后，创建主对象（表、视图、索引）。
- 必要时，根据应用程序开发人员所给的信息，修改数据库结构。
- 控制并监视用户对数据库的访问。
- 监视并优化数据库性能。
- 计划数据库信息的备份和性质。
- 在磁带上维护归档数据。
- 备份并恢复数据库。
- 与 Oracle 公司联系，寻求技术支持。

#### 1.1.2 安全官员

在有些情况下，数据可能有一个或多个安全官员。安全官员主要从事下列工作：登记用户，控制并监视用户对数据的访问，维护系统安全性。如果你的站点有单独的安全官，你可以不负责这些职责。

#### 1.1.3 应用程序开发人员

应用程序开发人员设计、实施数据库应用程序。应用程序开发人员的职责包括：

- 设计并开发数据库应用程序
- 为应用程序设计数据库结构
- 估计应用程序的存储需求
- 为应用程序规定数据库结构的修改
- 向数据库管理员传达上述信息
- 在开发过程中调整应用程序
- 在开发过程中建立应用程序的安全评价

#### 1.1.4 应用程序管理员

一个 Oracle 系统也可能拥有一个或多个应用程序管理员。应用程序管理员负责特定应用程序的管理需求。

#### 1.1.5 数据库用户

数据库用户通过应用程序或实用程序与数据库进行交互。用户职责通常包括：

- 在允许的地方，输入、修改、删除数据
- 生成数据报表

### 1.1.6 网络管理员

在一些系统中，也许有一个或多个网络管理员。网络管理员负责管理 Oracle 联网产品，如 Net8。

参见 欲了解在一个分布式环境中网络管理的有关信息，请参见“*Oracle8i Distributed Database Systems*”。

## 1.2 数据库管理安全性和权限

要在 Oracle 中完成管理任务，用户需要数据库及其运行所在的服务器操作系统的额外权限。对数据库管理员的帐号的访问必须严格访问。

本节包括下列专题：

- 数据库管理员的操作系统帐号
- 数据库管理员用户名
- DBA 角色

### 1.2.1 数据库管理员的操作系统帐号

要完成数据库管理任务，必须能够执行操作系统命令。取决于执行 Oracle 的操作系统，数据库管理员可能需要操作系统帐号或 ID 来获取对操作系统的访问。这样，数据库管理员的操作系统帐号可能需要比数据库用户更多的操作系统特权或访问权制。虽然数据库管理员不需要把 Oracle 文件存储在自己的帐号中，但必须能访问这些文件。

参见 区分数据库管理员帐号的方法是操作系统专用的。详见有关操作系统专用的 Oracle 文档。

### 1.2.2 数据库管理员用户名

数据库自动创建两个用户帐号，并为他们授予 DBA 角色。这两个用户帐号是：

- SYS （初始密码：CHANGE\_ON\_INSTALL）
- SYSTEM（初始密码：MANAGER）

下面几节中将介绍这两个用户名。

**注意** 要防止对数据字典表的不恰当的访问，必须在创建 Oracle 数据后立即改变 SYS 和 SYSTEM 用户点的密码。

你也许至少要创建一个管理员用户，用于完成日常的管理任务。

#### SYS

数据库新建时，用户 SYS 自动创建，并被授予 DBA 角色，该用户由密码 CHANGE\_ON\_INSTALL 来标识。

数据库数据字典的所有基表和视图都存储在 SYS 模式中。这些基表和视图对于 Oracle 操作是至关重要的。要维护数据字典的一致性，SYS 模式中的表都只能由 Oracle 处理；它们绝对不能由任何用户或数据库管理员进行修改，任何人都不要在用户 SYS 的模式中创建任何表。但是，如果有必要，用户可以改变数据字典设置的存储参数。

大多数数据库绝对不应该用 **SYS** 帐号进行连接。用户可以使用 **SYS** 帐号连接到数据库，但只能在 **Oracle** 技术人员或文档指导下才能进行。

## SYSTEM

数据库创建时，用户 **SYSTEM** 也自动创建，并授予该数据库的所有系统特权，用户 **SYSTEM** 由密码 **MANAGER** 标识。

**SYSTEM** 用户创建显示管理信息的表和视图，以及 **Oracle** 工具使用的表和视图。绝对不要在 **SYSTEM** 模式中创建与单个用户有关的表。

### 1.2.3 DBA 角色

每个 **Oracle** 数据库都会自动创建命名为 **DBA** 的预定义角色。该角色包含所有数据库系统特权。因此，该角色功能强大，应该只授予全功能的数据库管理员。

## 1.3 数据库管理员认证

数据库管理员必须经常进行一些特殊操作，如关闭、启动数据库。因为这些操作不能由普通数据库用户实施，数据库管理员用户名需要更安全的认证机制。

本节包括下列专题：

- 选择认证方式
- 使用操作系统认证
- **OSOPER** 和 **OSDBA**
- 使用认证密码文件

### 1.3.1 选择认证方式

下列认证数据库管理员的方法取代 **Oracle** 老版本提供的 **CONNECT INTERNAL** 语法。为了支持向后兼容，**CONNECT INTERNAL** 继续使用。

- 操作系统认证
- 密码文件

根据用户是希望管理在数据库驻留的同一台机器的数据库，还是管理来自单个远程客户的多个不同数据库，用户可以在操作系统认证或密码文件之间进行选择，以认证数据库管理员。图 1-1 演示了数据库管理员认证模式。

在大多数操作系统上，数据库管理员的 **OS** 认证包括在特殊组中放置数据库管理员的 **OS** 用户名（在 **UNIX** 中是 **DBA** 组），或者给予 **OS** 用户名特殊的进程权力。

数据库使用密码文件来记录已授予管理员权限的数据库用户名。

参见 在 “*Oracle8i Concepts*” 中有关于用户认证的更多信息。



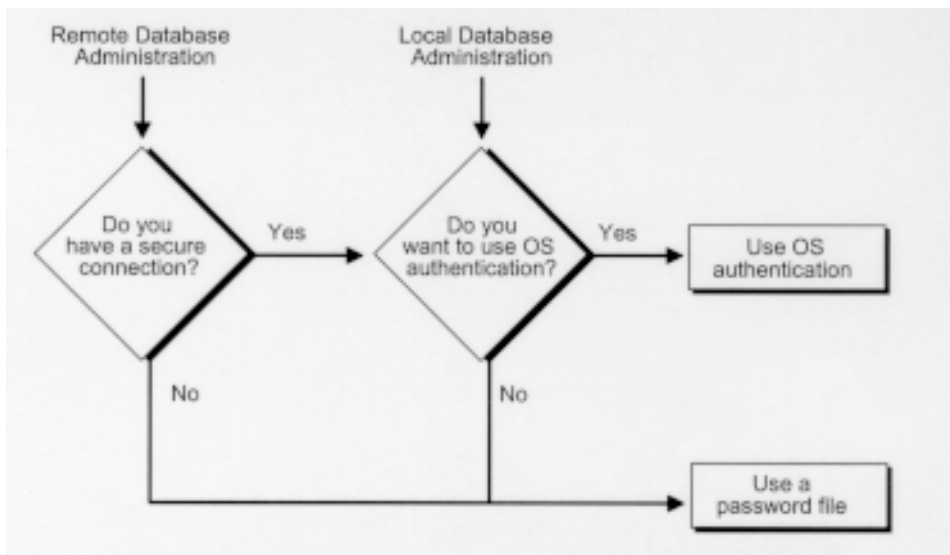


图 1-1 数据库管理员认证方法

### 1.3.2 使用操作系统认证

如果你选择操作系统认证，那么你可以使操作系统认证用户实施数据库管理操作。

1. 建立由操作系统认证的用户。
2. 确保初始参数 `REMOTE-LOGIN-PASSIVORDFILE` 设置为 `NONE`，`NONE` 是该参数的缺省值。
3. 通过键入下列 `SQL* plus` 命令中的一个，已认证用户应该能够连接到本地数据库，或通过安全连接，连接到一个远程数据库；

```
CONNECT /AS SYSOPER
CONNECT /AS SYSDBA
```

如果用户使用 Oracle 的老版本，以 `INTERNAL` 方式成功连接，那么用户应该能够使用在步骤 3 中的新语法继续连接。

**注意** 要使用 OS 认证，连接为 `SYSOPER` 或 `SYSDBA`，用户不必需要 `SYSOPER` 或 `SYSDBA` 系统特权。相反，服务器在操作系统级验证用户已经授予合适的 `OSDBA` 或 `OSOPER` 角色。

`CONNECT` 是 `SQL*Plus` 命令，欲了解有关其用法和句法的信息，可参见“*SQL\*Plus User's Guide and Reference*”。

### 1.3.3 OSOPER 和 OSDBA

两个特殊的操作系统角色：`OSOPER` 和 `OSDBA` 控制着在使用操作系统认证时的数据库管理员的注册。

**OSOPER** 允许用户进行下列操作：`STARTUP`、`SHUTDOWN`、`ALTER DATABASE OPEN/MOUNT`、`ALTER DATABASE BACKUP`、`ARCHIVE LOG` 和

RECOVER，并包括 RESTRICTED SESSION 权限。

OSDBA 包含 ADMIN OPTION 的所有系统权限、OSOPER 角色；允许进行 CREATE DATABASE 和基于时间的恢复。

对于你所使用的操作系统，OSOPER 和 OSDBA 可以拥有不同的名字和功能。

OSOPER 和 OSDBA 角色只能通过操作系统授予用户。它们不能通过 GRANT 语句来授予，也不能取消或丢弃。当用户以管理员权限进行登录且 REMOTE-LOGIN-PASSWORDFILE 设置为 NONE 时，Oracle 与操作系统进行通信，试图首先启用 OSDBA，如果不成功，然后启用 OSOPER。如果这两个操作都失败，则连接失败。怎样通过操作系统授予这些权限是与操作系统相关的。

如果正在进行远程数据库管理，那么应该阅读 Net8 文档，来判断是否在使用安全连接。最普通的连接协议，如 TCP/IP 和 DECnet 是不安全的，不管你正在使用哪个 Net8 版本。

参见 欲了解有关数据库管理员的 OS 认证信息，请参见专门针对操作系统的 Oracle 文档。

### 1.3.4 使用密码文件认证

如果已经决定使用密码文件来认证进行数据库管理的用户，那么必须完成下面介绍的步骤。每一个步骤都将在本章中的后续几节中详细介绍。

1. 使用 ORAPWP 实用程序创建密码文件。

```
ORAPWD FILE=filename PASSWORD=Password ENTRIES=max-users
```

2. 把 REMOTE-LOGIN-PASSWORDFILE 初始化参数设置为 EXCLUSIVE。

3. 通过使用 SQL，为每个需要进行数据库管理的用户授予合适的权限，在密码文件中加入用户。下面是一个实例：

```
GRANT SYSDBA TO SCOTT;  
GRANT SYSOPER TO SCOTT;
```

SYSDBA 权限允许用户实施与 OSDBA 相同的操作。同样，SYSOPER 权限允许用户实施与 OSOPER 相同的操作。

4. 已授权的原用户现在应该能够通过使用类似于下面的命令连接到数据库。

```
CONNECT scott/tiger a acct.hq.com As SYSDBA
```

## 1.4 密码文件管理

可以使用密码文件创建实用程序 ORAPWD 来创建密码文件；或者，对于选定的操作系统，可以创建该文件，作为标准安装的一部分。

本节包括下列专题：

- 使用 ORAPWD
- 设置 REMOTE-LOGIN-PASSWORDFILE
- 在密码文件中加入用户
- 以管理员权限连接
- 维护密码文件

参见 参见专门针对操作系统的 Oracle 文档，以了解使用权用安装实用程序来安装密码文件的信息。

#### 1.4.1 使用 ORAPWD

如果调用密码文件创建实用程序，而不提供任何参数，那么系统将显示一则消息，表示该命令的正确用法如下面的样本输出。

```
orapwd
Usage: orapwd file=<fname> password=<password> entries=<users>
where
file - name of password file (mand) ,
password - password for SYS and INTERNAL (mand),
entries - maximum number of distinct DBAs and OPERs (opt) ,
There are no spaces around the equal-to (=) character.
```

例如，下面的命令创建名为 ACCT.PWD 的密码文件，该文件允许有 30 个不同密码的授权用户。该文件最初由以 SYSOPER 或 SYSDBA 连接的用户以密码 SECRET 来创建。

```
ORAPWD FILE=acct.pwd PASSWORD=secret ENTRIES=30
```

下面介绍 PRAPWD 实用程序的参数。

##### FILE (文件)

该参数设置欲创建的密码文件的名定。必须规定该文件的全路径名。该文件的内容是加密的，也就是用户不可读的。该参数是强制的。

密码文件所允许的文件名类型是与操作系统有关的。有些平台要求密码文件必须是特定格式，且位于特定目录中。其它平台允许使用环境变量来指定密码文件的名称和位置。欲了解在各种平台上允许的文件名和位置，请参见与操作系统相关的 Oracle 文档。

如果使用 Oracle Parallel Server，运行多个 Oracle 事例，那么每个事例的环境变量应该指向相同的密码文件。

**警告** 保护密码文件和标识密码文件位置的环境变量，这对于系统的安全性是非常重要的。拥有访问这些文件和变量的能力的用户，都潜在地威胁系统的安全性。

##### PASSWORD (密码)

该参数设置 SYSOPER 和 SYSDBA 的密码。如果在连接到数据库以后，你发出 ALTER USER 语句来改变密码，那么，存储在数据字典中的密码和存储在密码文件中的密码都被更新。仅仅是为了向下兼容，系统才支持 INTERNAL 用户。该参数是强制的。

##### ENTRIES (条目)

该参数指定密码文件接受的条目数。它对应于允许以 SYSDBA 或 SYSOPER 连接到数据库的不同用户的数量。可以输入的实际条目数可以稍高，因为 ORAPWD 实用程序将继续指定密码项，直到操作系统块填满。例如，如果操作系统块大小为 512 字节，它将容纳 4 个密码项，所分配的密码项数目将始终是 4 的倍数。

因为用户不断地从密码文件中加入并删除，所以文件中条目可以重新使用。如果你希望

指定 `REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE`，并允许给用户授予 `SYSOPER` 和 `SYSDBA` 权限，那么该参数是必需的。

**警告** 当用户超出这个权制时，用户必须创建一个新的密码文件。要避免这种情况的发生，应该选择一个比自己想像的更大的数字。

#### 1.4.2 设置 `REMOTE_LOGIN_PASSWORDFILE`

除了创建密码文件以外，还必须把初始参数 `REMOTE_LOGIN_PASSWORDFILE` 设置为合适的值。可识别的值在下面介绍。

**注意** 要启动一个数据事例，你必须指定数据库名字和参数文件，来初始化事例设置。用户可以使用 `Nets` 指定一个全限定的远程数据库名字。但是，初始化参数文件和任何相关的文件，例如配置文件都必须存在于客户机上。也就是，参数文件必须位于用户启动事例所在的机器。

##### `NONE`

把参数设置为 `NONE`，将导致 `Oracle` 的行为好象密码文件不存在。即，通过不安全的连接，不允许有授权的连接。`NONE` 是该参数的缺省值。

##### `EXCLUSIVE`

`EXCLUSIVE` 密码文件只能用于一个数据库。只有 `EXCLUSIVE` 文件能包含不同于 `SYSOPER` 和 `SYSDBA` 的用户名，使用 `EXCLUSIVE` 密码文件，允许你为单个用户授予 `SYSDBA` 和 `SYSOPER` 系统特权，并让用户以他们自身进行连接。

##### `SHARED`

`SHARED` 密码文件可以用于多个数据库。但是，`SHARED` 密码文件只能识别 `SYSDBA` 和 `SYSOPER` 用户；你不能在 `SHARED` 密码文件中加入用户。所有需要 `SYSDBA` 和 `SYSOPER` 权限的用户必须使用相同的名字 `SYS` 和密码进行连接。如果要让单个 `DBA` 管理多个数据库，那么该选项非常有用。

**建议** 要达到最高的安全级，你应该在创建密码文件以后立即把 `REMOTE_LOGIN_PASSWORDFILE` 文件初始化参数设置为 `EXCLUSIVE`。

#### 1.4.3 在密码文件中加入用户

当你为用户授予 `SYSDBA` 或 `SYSOPER` 特权时，该用户的名字和权限信息加入到密码文件。如果没有 `EXCLUSIVE` 密码文件，即如果初始参数 `REMOTE_LOGIN_PASSWORDFILE` 为 `NONE` 或 `SHARED`，那么，如果你试图授予这些权限，会接收到错误消息。

用户至少拥有这两个权限之一时，用户名保留在密码文件中。当取消用户的最新权限时，该用户从密码文件中删除。

创建密码文件，并在其中加入新用户

1. 按照创建密码文件的指令操作。
2. 把 `REMOTE_LOGIN_PASSWORDFILE` 初始参数设置为 `EXCLUSIVE`。
3. 以 `SYSDBA` 权限进行连接，下面是一个例子：

```
CONNECT SYS/change_on_install AS SYSDBA
```

4. 启动事例，创建数据库，装载并打开现有的数据库。
5. 根据需要建立用户。为自己和合适的其他用户，授予 **SYSOPER** 或 **SYSDBA** 权限。
6. 这些用户现在加入到密码文件中，可以用用户名和密码（代替 **SYS**）以 **SYSOPER** 或 **SYSDBA** 连接到数据库。密码文件的使用不会防止 OS 认证用户连接到数据库。

#### 授予并取消 **SYSOPER** 和 **SYSDBA** 权限

如果服务器使用 **EXCLUSIVE** 密码文件，就使用 **GRANT** 语句为用户授予 **SYSDBA** 或 **SYSOPER** 系统特权，下面是一个例子：

```
GRANT SYSDBA TO scott;
```

使用 **REVOKE** 语句取消用户的 **SYSDBA** 或 **SYSOPER** 系统权限，下面是一个示例：

```
REVOKE SYSDBA FROM scott;
```

因为 **SYSDBA** 和 **SYSOPER** 是最强大的数据库权限，所以不用使用 **ADMIN OPTION**。只有当前以 **SYSDBA**（或 **INTERNAL**）连接的用户能够为另一用户授予 **SYSDBA** 或 **SYSOPER** 系统权限。对于取消权限也是如此。这些权限不授予角色，因为角色只能在数据库启动才可用。不要把 **SYSDBA** 和 **SYSOPER** 数据库权限与操作系统角色混淆，它们是完全无关的特征。

系统权限的更详细信息参见第 23 章“管理用户权限和角色”。

#### 列举密码文件成员

使用 **V\$PFILE\_USERS** 视图来判断哪些用户已经授予了数据库的 **SYSDBA** 和 **SYSOPER** 系统权限。该视图显示的列如下所示：

##### **USERNAME**

密码文件识别的用户的名字。

##### **SYSDBA**

如果该列的值为 **TRUE**，那么用户可以以 **SYSDBA** 系统权限进行登录。

##### **SYSOPER**

如果该列的值为 **TRUE**，那么用户可以以 **SYSOPER** 系统权限进行登录。

#### 1.4.4 以管理权限连接

当你用用户名和密码以 **SYSOPER** 或 **SYSDBA** 权限连接，你以缺省模式，不是以与自己的用户名相关的模式连接。对于 **SYSDBA**，该模式为 **SYS**；对于 **SYSOPER**，该模式为 **PUBLIC**。

##### 以管理员权限连接：实例

例如，假设用户 **SCOTT** 发出下列语句：

```
CONNECT scott/tiger
```

```
CREATE TABLE scott_test (name VARCHAR2 (20));
```

然后，在 SCOTT 发出以下语句时：

```
CONNECT SCOTT/tiger AS SYSDBA
```

```
SELECT * FROM scott_test;
```

他收到一则错误消息，SCOTT\_TEST 不存在。这是因为 SCOTT 现在缺省地引用 SYS 模式，而表在 SCOTT 模式中创建。

#### 非安全远程连接

要通过一个非安全连接作为一个授权用户连接到 Oracle，必须满足以下条件：

- 所连接到的服务器必须拥有密码文件。
- 必须授予 SYSOPER 或 SYSDBA 系统权限。
- 必须使用用户名和密码文件进行连接。

#### 本地和安全的远程连接

要通过本地的或安全的远程连接作为授权用户连接到 Oracle，必须满足下列条件集的一个：

- 假设你满足在以前的公告列表中的非安全连接的准则，可以使用密码文件进行连接。
- 如果服务没有使用密码文件，或者没有被授予 SYSOPER 或 SYSDBA 权限，因而不在于密码文件中，那么你的操作系统名必须为授权连接由操作系统进行认证。这种形式的认证为操作系统专用。

有关密码文件管理的更详细信息在 1.4 节“密码文件管理”中介绍。

参见 阅读操作系统专用的 Oracle 文档。以了解操作系统认证的细节。

### 1.4.5 维护密码文件

本节描述怎样在密码文件充满以后扩充密码文件中用户数，怎样删除密码文件，以及怎样避免改变密码文件的状态。

#### 扩充密码文件用户的数量

如果在试图为一个用户授予 SYSDBA 或 SYSOPER 系统权限限时，收到文件满的错误 (ORA-1996)，那么必须创建一个更大的密码文件，并重新为用户授权。

#### 取代密码文件

1. 通过查询 V\$PWFIL\_USERS 视图来注释哪些用户拥有 SYSDBA 或 SYSOPER 权限。
2. 关闭数据库。
3. 删除现有的密码文件。
4. 按照 1.4.1 节“使用 ORAPWD”只介绍的使用 ORAPWD 创建新密码文件的步骤，进行逐步操作。确保把 ENTRIES 参数设置为足够大的数。
5. 按照 1.4.3 节“在密码文件中加入用户”中介绍的步骤进行操作。

## 删除密码文件

如果决定不再需要使用密码文件来认证用户，则可以删除密码文件，并把 `REMOTE_LOGIN_PASSWORDFILE` 初始参数设置为 `NONE`。在删除密码文件以后，只有由操作系统认证的用户才可以实施数据库管理操作。

**警告** 如果拥有使用 `REMOTE_LOGIN_PASSWORD FILE=EXCLUSIVE`（或 `SHARED`）装载的数据库或事例，就不要删除或修改密码文件。如果删除或修改该文件，就不能使用密码文件进行远程重新连接。即使你取代了密码文件，也不能使用新密码文件，因为时间标记和校验和是错误的。

## 改变密码文件状态

密码文件状态保存在密码文件中。第一次创建密码文件时，其缺省状态是 `SHARED`。可以通过设置参数 `REMOTE_LOGIN_PASSWORDFILE` 参数来改变密码文件的状态。在用户启动一个事例时，Oracle 从存储在用户客户机上的初始参数文件中提取该参数的值。当用户装载数据库时，Oracle 把该参数的值与保存在密码文件的值进行比较。如果这两个值不匹配，那么就改写保存在文件中的值。

**警告** 应该使用警告，以保证 `EXCLUSIVE` 密码文件没有偶然改为 `SHARED`。如果你计划允许在多个客户上启动事例，那么每个客户必须拥有初始参数文件。而且在所有初始参数文件中的 `REMOTE_LOGIN_PASSWORDFILE` 参数的值必须相同。否则，密码文件的状态可能根据事例启动的地方而改变。

## 1.5 数据库管理员实用程序

有一些实用程序可以帮助你维护和控制 Oracle 服务器。下面几节包括下列专题：

■ `SQL * Loader`

■ `Export` 和 `Import`

参见 Oracle 提供的实用程序的有关信息在 “*Oracle8i Utilities*” 中。

### 1.5.1 `SQL * Loader`

数据库管理员和 Oracle 用户都使用 `SQL * Loader`。它把标准操作系统文件（文本文件或 `C` 数据格式文件）中的数据加载到 Oracle 数据库表中

### 1.5.2 `Export` 和 `Import`

`Export` 和 `Import` 实用程序让用户把 Oracle 格式的现有数据移进或移出 Oracle 数据库。例如，输出文件可以归档数据库数据，或者在运行于相同或不同操作系统之间的不同 Oracle 数据库之间移动数据。

## 1.6 数据库管理员的优先权

通常，必须实施一系列的步骤，以使数据库系统启动、运行，然后维护它。这些步骤是：

- 步骤 1: 安装 Oracle 软件。
- 步骤 2: 评估数据库服务器硬件。
- 步骤 3: 规划数据库。
- 步骤 4: 创建并打开数据库。
- 步骤 5: 实现数据库设计。
- 步骤 6: 备份数据库。
- 步骤 7: 加入系统用户。
- 步骤 8: 调整数据库性能。

下面几节详细介绍以上的每个步骤。

**注意** 如果要迁移到一个新版本，就要在安装前备份现有的产品数据库。欲了解怎样保留现有产品数据库的详细信息，请参见 “*Oracle8i Migration*”。

### 1.6.1 步骤 1: 安装 Oracle 软件

作为数据库管理员，你必须安装 Oracle 服务器软件以及访问数据库的任何前端工具和数据库应用程序。在一些分布式处理安装中，数据库由中央计算机控制，数据库工具和应用程序在远程机器上运行；在这种情况下，必须安装 Oracle Net8 驱动程序，它是把远程机器连接到执行 Oracle 的计算机所必需的。

欲详细了解安装哪些软件的有关信息，请参见 1.7 节 “识别 Oracle 数据库软件的版本。”

**参见** 要了解安装的特殊需求和指导，请参阅操作系统专用的 Oracle 文档，以及前端工具和 Net8 驱动程序的安装指南。

### 1.6.2 步骤 2: 评估数据库服务器硬件

在安装以后，要评估 Oracle 及其应用程序怎样能够最充分地使用可用的计算机资源。该评估应该揭示下列信息：

- 有多少个磁盘驱动器能够用于 Oracle 及其数据库？
- 有多少个专用磁带驱动器（如果有的话）能够用于 Oracle 及其数据库？
- 有多少内存能够用于将运行的 Oracle 的事例（参见系统配置文档）

### 1.6.3 步骤 3: 规划数据库

作为数据库管理员，你必须规划：

- 数据库的逻辑存储结构
- 数据库总体设计
- 数据库的备份战略

规划数据库的逻辑存储结构怎样影响系统性能和各种数据库管理操作，这是非常重要的。例如，在创建数据库和表空间之前，你应该知道多少个数据文件构成表空间，数据文件物理上存储在什么位置（在哪个磁盘驱动器上），在每个表空间中保存哪类信息。在规划数据库的总体逻辑存储结构时，应该考虑在数据库实际创建和运行时结构的效用。此类的考虑包括数据库的逻辑存储结构将如何影响下列项：

- 执行 Oracle 的计算机的性能
- 在数据访问操作期间的数据库性能



## ■ 数据库备份和恢复策略的效率

规划数据库对象的关系设计，以及每一个对象的存储特点。通过在创建对象之前规划对象之间的关系，以及每个对象的物理存储，可以直接从整体上影响数据库的性能。一定要为数据库的扩展作好规划。

在分布式数据库环境中，规划阶段尤其重要。频繁访问的数据的物理位置能显著地影响应用程序的性能。

在上述的规划阶段，也要规划数据库的备份策略。在开发了备份策略以后，你可能发现要改变已规划的数据库逻辑存储结构或数据设计以改善备份效率。

分布式数据库和关系型数据库的讨论超出了本书的范围；如果你不熟悉此类设计问题，请参阅解释这些问题的工业标准书籍。

第二部分“数据库存储”和第四部分“模式对象”提供有关逻辑存储结构、对象的创建以及数据库的一致性约束的有关信息。

### 1.6.4 步骤 4: 创建和打开数据库

一旦完成了数据库设计，你可以创建数据库，打开该数据库。使用 Oracle 的 Database Configuration Assistant 可以在安装时创建数据库，或者你可以提供自己的创建数据库的脚本。请参阅第 2 章“创建 Oracle 数据库”以了解数据库创建的有关信息，参阅第 3 章“启动和关闭”，以了解数据库启动的指南。

### 1.6.5 步骤 5: 实现数据库设计

一旦创建并启动了数据库，便可以通过创建所有必需的回退段和表空间，来实现数据库的已规划逻辑结构。一旦建立逻辑结构以后，你可以创建数据库的对象。第三部分“数据库存储”和第四部分“模式对象”包含帮助用户创建数据库的逻辑存储结构和对象的信息。

### 1.6.6 步骤 6: 备份数据库

在你创建了数据库结构以后，要通过创建额外的重复日志文件，取出第一个完整数据库备份（联机或脱机），以周期性的间隔安排将来的数据库备份，来实现规划的备份策略。

参见 参见“*Oracle8i Backup and Recovery Guide*”或“*Oracle8i Recovery Manager User's Guide and Reference*”以了解定制备份操作和实现恢复步骤的指导。

### 1.6.7 步骤 7: 加入系统用户

一旦你备份了数据库结构，你可以开始按照你的 Oracle 授权协议加入数据库用户，为用户创建角色，为他们授予合适的角色。

下列章将为你提供帮助：

- 第 21 章“建立安全策略”
- 第 22 章“管理用户和资源”
- 第 23 章“管理用户权限和角色”

### 1.6.8 步骤 8: 调整数据库性能

优化数据库系统的性能，是数据库管理员的日常职责。此外，Oracle 提供了数据库资源管理特征，它允许你控制怎样为多个用户组分配资源。数据库资源管理器将在第 25 章“数据库资源管理器”中介绍。

参见 “*Oracle8i Designing and Tuning for Performance*” 包含怎样调整数据库和应用程序的信息。

## 1.7 区别 Oracle 数据库软件版本

因为 Oracle 产品继续升级，为了解决问题和增强功能，需要进行维护，所以产生了数据库服务器的新版本。在任何时间点，提供多个版本是正常的。要完全识别一个版本，需要多达五个号码。这些号码的意义在下面讨论。

### 1.7.1 版本号格式

Oracle 数据库服务器发布磁带可能标上“版本 8.1.5.1.1”。下面将帮助读者理解 Oracle 使用的版本级别命名方法。

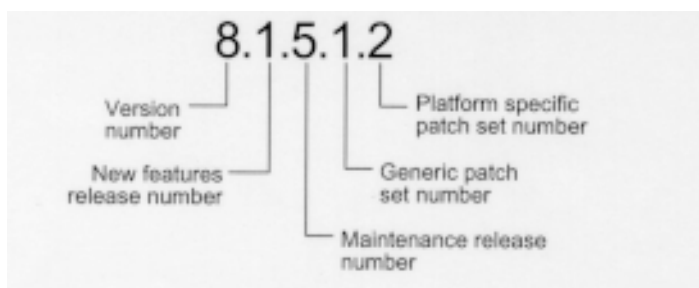


图 1-2 Oracle 版本号的例子

#### Version (版本号)

这是最主要的标识符，它表示软件的主要版本，包含显著的新功能。  
例如：版本 8（也标识为版本 8.0）。

#### 新特征版本号 (New Features Release Number)

这个号表示新特征版本级。例如：版本 8.1。

#### 维护版本号 (Maintenance Release Number)

该号表示一个维护版本级。也可能包含一些新特征。例如：版本 8.0.4，版本 8.1.6。

#### 通用补丁集号 (Generic Patch Set Number)

该号表示一个通用补丁集。补丁集可以跨越所有操作系统和硬件平台使用。例如：补丁集版本 8.1.5.2。

#### 专用平台补丁集 (Platform Specific Patch Set Number)

该是表示一个只能应用到特定操作系统和硬件平台的补丁集。例如：补丁集 8.0.4.1.1。

### 1. 7. 2 检测当前版本号

要识别当前安装的 Oracle 数据库服务器的版本，并弄清正在使用的其它 Oracle 组件的版本级，请查询数据字典视图 **PRODUCT\_COMPONENT\_VERSION**。下面显示了一个样板查询。注意，其它产品的版本可以与数据库服务器无关地增长。

```
SELECT * FROM product_component_version ;
```

| PRODUCT                     | VERSION   | STATUS     |
|-----------------------------|-----------|------------|
| CORE                        | 8.1.5.0.0 | Production |
| NLSRTL                      | 3.4.0.0.0 | Production |
| Oracle8i Enterprise Edition | 8.1.5.0.0 | Production |
| PL/SQL                      | 8.1.5.0.0 | Production |
| TNS for 32-bit Windows ;    | 8.1.5.0.0 | Production |

5 rows selected.

该查询显示的信息，对于报告软件问题是非常重要的。

此外，你也可以查询 **V\$VERSION** 视图，以弄清组件的级别信息。

# 2 创建 Oracle 数据库

本章讨论 Oracle 数据库的创建过程，其中包括如下主题：

- 创建数据库之前应考虑的事项。
- Oracle 数据库配置助理（DBCA）。
- 手动创建 Oracle 数据库。
- 安装参数。
- 数据库创建之后应考虑的事项。
- 初始化调整指导。

参见 本章讨论单一数据库实例的创建。虽然其中有许多材料相关，但用户欲了解 Oracle 并行服务器环境的特定信息时，应参阅“*Oracle8i Parallel Server Setup and Configuration Guide*”。

## 2.1 创建数据库之前应考虑的事项

数据库的创建准备了一些操作系统文件，所以它们可以作为 Oracle 数据库来一起工作。不管有多少个数据库文件，也不管已经访问了多少个事例，用户需要的仅仅是创建一次数据库。创建一个与已存在数据库名称相同的数据库，用相同的物理结构，也可以清除已存在数据库的信息。

下面的主题可以帮助用户准备数据库的创建：

- 规划数据库的创建。
- 创建的先决条件。
- 确定怎样创建一个 Oracle 数据库。

### 2.1.1 规划数据库的创建

在用户规划数据库的创建时，应考虑如下的操作：

- 规划数据库的表和索引，估计它们将要求多少空间。本书中其它的章节包含不同数据库模式对象的创建，可以帮助用户进行这样的估计。也可以参考“*Oracle8i Concepts*”。
- 逐步熟悉那些构成用户初始化参数文件的不同安装参数。一些参数的信息可以参见 2.4 节“安装参数”，以及本书的其它一些章节，但完整的初始化参数在“*Oracle8i Reference*”中。
- 规划怎样防止用户的新数据库失败，包括那些联机 and 归档的重复日志的配置（以及它们需要多大的空间），以及备份策略。联机和归档的重复操作日志在第 6 章“管理联机重复日志”和第 7 章“管理归档重复日志”中讨论。规划备份策略的其它必要信息，在“*Oracle8i Backup and Recovery Guide*”中介绍。

- 选择数据库的字符集。在创建数据库时，用户必须确定数据库的字符集。所有的字符数据，包括数据字典中的数据，都存储在数据字符集中。如果用户用不同的字符集来访问数据库，数据库字符集应该与所使用的字符集相同或是它的超集。本土语言的支持是“*Oracle8i National Language Support Guide*”中的一个题目。
- 选择数据库块的大小。这确定为初始化参数，在不进行数据库的重新创建时，它是不可更改的。
- 选择用户的全局数据库名。这个名字在 `CREATE DATABASE` 语句中指定，而且它也可以由安装参数指定。

另外，要熟悉一个事例的启动和关闭、数据库的安装和打开的原则和选项。这些在第 3 章中讨论。其它的方法会在用户的 Oracle 指定操作系统的文档中讨论。

### 2.1.2 创造必要的先决条件

要创建一个新的数据库，必须满足下面必要的先决条件：

- 安装期望的 Oracle 软件。包括设置各种环境变量，这些变量对于用户的操作系统是唯一的；建立软件和数据库文件的结构目录。
- 用户拥有操作系统与整个数据库管理操作相关的权限。用户必须由操作系统或通过一个密码文件来进行特别授权，使得用户可以在数据库创建或打开之前能够启动和关闭某个事例。该授权在 1.3 节“数据库管理员认证”中进行讨论。
- 充足的内存用于启动 Oracle 事例。
- 在运行 Oracle 数据库的计算机上有规划数据库所需之充足的磁盘存储空间。

所有这些在给特定用户操作系统的 Oracle 安装指南中均有论述。另外，Oracle Universal Installer 将指导用户的安装，并在设置环境变量、目录结构及认证时提供帮助。

### 2.1.3 决定怎样创建一个 Oracle 数据库

创建一个数据库应包括下面的操作：

- 创建信息结构，包括数据目录，这是 Oracle 用来访问和使用数据库用的。
- 创建和安装数据库的控制文件和重做日志文件。
- 创建新的数据库文件或删除以前数据库文件中的数据。

用户使用 `CREATE DATABASE` 语句来执行这些操作，但是在用户拥有一个可操作的数据库之前其它操作是需要的。这些操作中的少数几个是创建用户和临时表空间、建造数据字典表的视图、以及安装 Oracle 内置的包。这就是为什么数据库创建过程中包括运行一个已经准备好的脚本的原因。但是，用户不需要自己来准备这个脚本。

用户在创建新的 Oracle 数据库时，有如下一些选项：

- 使用 Oracle 数据库配置助理（DBCA）

DBCA 是由 Oracle 通用安装器来装入的，可以自动地创建用户的数据库开始器。用户拥有使用或不使用 DBCA 的选项，也有创建自定义数据库的选项。另外，用户可以在任何要建立新数据库时作为独立的工具来装入 DBCA。DBCA 提供创建数据库最简单的方法。参阅 2.2 节“Oracle 数据库配置助理（DBCA）”。

- 通过脚本手工创建数据库

在已经拥有一个现存的数据库时，或者有在使用 DBCA 中所能遇到的要求以外的新要求时，用户可能选择手工地创建数据库。Oracle 在所发行的数据库软件文件中提供了一个示例数据库的创建脚本和一个示例安装参数文件。这两个文件都可以为适合用户需要而进行编辑。参阅 2.3 节“手工创建 Oracle 数据库”。

■ 移植或升级一个现存的数据库

如果用户在使用以前版本的 Oracle，那么只有在用户希望要一个全新的数据库时才要求创建数据库。其实，用户可以移植在 Oracle 以前版本管理下的数据库，用新版本的 Oracle 软件来使用。本书中对数据库的移植没有论及。“Oracle8i 移植手册”中包含关于移植现有数据库的信息。

2.2 Oracle 数据库配置助理 (DBCA)

DBCA 是一个图形用户界面(GUI)的工具，可以通过 Oracle 通用安装器来进行交互，或者独立使用，以简化数据库的创建。它在下面几节中进行描述：

- 使用 DBCA 的优点。
- 数据库创建的 DBCA 模式。

2.2.1 使用 DBCA 的优点

下面是使用 DBCA 的一些优点：

- 它使用最佳柔性结构（OFA,Optimal Flexible Architecture），而数据库文件和管理文件，包括安装文件，按照标准的命名和布局实践。
- 速度快。一个已创建的数据库可以被拷贝到位，而不是进行一个冗长的创建过程。
- 已经为用户作了选择。用户不需要花费时间来决定怎样设置参数。
- 可以自定义用户的数据库。甚至无需选择拷贝数据库的开始器，用户依然可以使用 DBCA 来生成创建用户数据库的 OLTP 的脚本、数据仓库、多目标环境。这仅需要回答几个由 DBCA 提出的问题。如果你拥有超过指定数量的用户，它会自动地包括在 MTS 之中。

由 DBCA 创建的数据库类型（OLTP、数据仓库、多目标环境）的描述，在 2.2.3 节“识别用户的数据库环境”中。

用户可以使用 DBCA 来创建、删除、或修改数据库。修改选项允许用户启动未被启用的选项。只有创建数据库选项在这一节中讨论。

2.2.2 数据库创建的 DBCA 模式

当用户在安装时从 Oracle 通用安装器里运行 DBCA 时，用户为 Oracle 通用安装器所选择的安装类型影响用户能创建数据库（OLTP、数据仓库或多目标）的类型。下面是 Oracle 通用安装器提供给用户的安装类型：

| 安装类型 | 创建数据库所要求的用户输入 |    |
|------|---------------|----|
|      | 最小            | 扩展 |
| 典型   | X             |    |
| 最小   | X             |    |
| 定制   | X             | X  |

2.2.4 节“选择数据库的创建方法”中，略述了基于用户安装选择的可创建的数据库的类型。

2.2.3 识别用户的数据库环境

Oracle 通用安装器使用户创建在下述环境之一进行操作的 Oracle8i 数据库。Oracle8i 数据库的合适环境的识别：

| 环境           | 描述   |
|--------------|--|
| 在线事务处理（OLTP） | 许多并发用户进行无数的要求快速访问的数据事务。有效性、速度、并发、可恢复性是关键点<br>事务的组成：在数据库表中读取（SELECT 语句）、写入（INSERT 和 UPDATE 语句），和删除（DELETE 语句）数据       |
| 数据仓库（或 DSS）  | 用户执行无数的需要处理大量数据的复杂查询。响应时间、准确性、有效性是关键点<br>这些查询（典型的是只读查询）简单地从找出几个记录，复杂到在不同的表中进行数以千计的记录排序都会有<br>数据仓库环境也被视为决策支持系统(DSS)环境 |
| 多目标          | 两种应用程序都可以访问这个数据库   |

2.2.4 选择数据库的创建方法

Oracle 数据库的类型（OLTP、数据仓库、多目标）是通过典型、最小和自定义安装方式来创建的，所需的用户输入量在下面描述。复习这些选择，确定最适合用户要求的数据库和数据库创建等专业知识：

| 如果用户执行的步骤...  | 那么...  |
|---|--|
| 1. 选择典型安装（Typical installation type）  | Oracle 数据库配置助理在安装的最后会自动启动，并创建一个经过配置的、备用的多目标数据库启动器，有如下特点：<br>■ 缺省安装参数<br>■ Oracle 选项和 interMedia 元件 1 的自动安装和配置<br>■ 先进的复制能力<br>■ 多线程服务器模式的数据库配置 2<br>■ NOARCHIVELOG 的归档模式<br>无需用户进行输入 |
| 1. 选择最小安装类型（Minimal installation type）.   | Oracle 数据库配置助理在安装的最后自动启动，并创建与典型安装相同的 Oracle8i 数据库，有如下的例外情况：  |
| 2. 在提示创建数据库启动器时选择 Yes.  | ■ 不提供 Oracle 选项和 interMedia 元件的安装和配置<br>■ 专用服务器模式的数据库配置  |
| 注意 如果用户选择 No，除数据库以外的所有元件将被安装。用户以后能够手工地运行 Oracle 数据库配置助理，或用 SQL 脚本来创建数据库。参阅“Oracle8i Administrator’s Guide for Windows NT”以获取指导。 |  |

| 如果用户执行的步骤...   | 那么...   |
|--|---|
| <p><b>注意</b> 多目标数据库通过 Oracle Internet Directory (互连网目录) 安装类型也是可安装的。该数据库只用于保存 Oracle Internet Directory (互连网目录) 信息。</p>   |   |
| <ol style="list-style-type: none"> <li>1. 选择自定义安装类型 (Custom installation type)</li> <li>2. 选择 Oracle 服务器和 Availabel Products 窗口中的附加产品。</li> <li>3. 在提示创建数据库启动器时选择 Yes。</li> <li>4. Oracle 数据库配置助理提示用户选择如下两种数据库创建方法之一： <ul style="list-style-type: none"> <li>■ 自定义 (Custom)</li> <li>■ 典型 (Typical)</li> </ul> </li> </ol> | <p>如果用户选择自定义数据库创建方法：</p> <p>Oracle 数据库配置助理指导用户在数据库创建过程中自定义地按用户选择来匹配环境 (OLTP、数据仓库、多目标) 和数据库配置模式 (专用服务器或多线程服务器)。选项和 interMedia 元件 (如果安装了) 和高级复制 (如果选择了) 也会自动配置。只有用户对高级的数据库创建程序熟练时，才选择这个选项。例如自定义成：</p> <ul style="list-style-type: none"> <li>■ 数据、控制、和重做日志文件的设置</li> <li>■ 表空间和扩展空间的大小</li> <li>■ 数据库内存的参数</li> <li>■ 存档模式、格式、和目标文件</li> <li>■ 目标的踪迹文件</li> <li>■ 字符集值</li> </ul>   |
|  | <p>如果用户选择典型数据库创建方法：</p> <p>用户有两个选择。数据库创建过程中的 Oracle 数据库配置助理的作用取决于用户的选择：</p> <p>如果选择：</p> <ul style="list-style-type: none"> <li>■ 从 CD 上拷贝现存文件</li> </ul> <p>Oracle 数据库配置助理创建在前页描述的典型模式相同的 Oracle8i 数据库。选项和 interMedia 元件 (如果已经安装) 也会自动配置而不需要用户输入 3。</p> <ul style="list-style-type: none"> <li>■ 创建新的数据库文件</li> </ul> <p>Oracle 数据库配置助理提示用户回答几个问题，包括选择数据库环境 (OLTP, 数据仓库、多目标) 和确定当前连接的数量。Oracle 数据库配置助理会动态创建数据库，选项和 interMedia 元件 (如果已经安装) 和高级复制 (如果已经选择) 也会自动配置 3, 4</p> |

1. Oracle 数据库配置助理只配置通过 Oracle 通用安装器安装的选项。

2. 要获得关于专用和多线程的服务器模式 (也叫做共享服务器模式) 的描述，请参阅 "Oracle8i Administrator's Guide for Windows NT"。

3. 如果用户在安装时选择 Oracle Jserver，数据库将用多线程服务器模式 IIOP 客户来创建。

4. 如果用户在用户数据库环境中选择 OLTP，并键入 20 或更多的并发数据库连接的数量，用户的数据库将以多线程服务器模式来创建，否则，服务器模式是专用的。



## 2.3 手工创建 Oracle 数据库

手工创建数据库的过程最好通过检查示例数据库创建脚本来说明。但是用户也应该注意到在创建数据库时应遵循的步骤，以及如果问题发生或改变主意时应该怎么解决。

本节讨论：

- 创建 Oracle 数据库的步骤
- 验证数据库创建脚本
- 数据库创建中的故障诊断
- 删除数据库

### 2.3.1 创建 Oracle 数据库的步骤

下面描述创建 Oracle 数据库的步骤，应该按所提出的顺序来进行。用户应该预先已经创建了创建 Oracle 数据库的环境，包括作为 Oracle 软件安装过程中一部分的操作系统有关的环境变量。

创建一个新的数据库,并使之对系统的使用有效

1. 决定用户的事例标志 (DB\_NAME 和 SID)。

Oracle 事例标志应该与数据库名字 (DB\_NAME 的值) 匹配。该事例标志用来避免与其他 Oracle 事例混淆，而这里所指的其他 Oracle 事例你可在系统上创建和运行。

要获得更多的信息，参阅用户的指定操作系统的 Oracle 文档。

2. 创建参数文件的安装

对于任意一个 Oracle 数据库的事例（系统全局区域和后台进程）使用安装参数文件来启动。要为数据库创建参数文件，用户将要使用用户的操作系统来制作一个 Oracle 在其发行介质上的安装参数文件的一份拷贝。给这个拷贝一个新的文件名。然后用户可以编辑并定制新数据库的这个新文件。参阅 2.4 节“安装参数”以获得有关用户可能希望编辑的参数的启示。也可参阅 3.5 节“使用初始化参数文件”。

用户系统上的每一个数据库应该至少有一个定制的安装参数文件仅与该数据库对应。不要在几个数据库上使用相同的文件。

**注意** 在分布式处理环境中，Enterprise Manager (企业管理器) 经常在网络的客户端机器上运行。如果客户端机器用来运行 Enterprise Manager，并创建一个新的数据库，用户需要拷贝新的安装参数文件（当前位于正在执行 Oracle 的计算机上）到用户的客户端工作站。这个程序是与操作系统相关的。关于在用户网络上拷贝文件的更多信息，请参阅用户特定操作系统的 Oracle 文档。

本书不讨论 Enterprise Manager。它主要在 3.1.2 节中的“使用 Oracle Enterprise Manager”小节中讨论。

3. 启动 SQL\*Plus，并作为 SYSDBA 连接到用户 Oracle 事例。

这个例子假设用户有适当的权限。

```
$ SQLPLUS /nolog
CONNECT username/password AS sysdba
```

#### 4. 启动一个事例。

用户不安装数据库就可以启动一个事例。典型的，用户只在数据库创建期间这样做。使用带 NOMOUNT 选项的 STARTUP 语句，如果不指定 PFILE，安装参数文件从操作系统特定的缺省安装中读取。

STARTUP NOMOUNT;

STARTUP 语句将在第 3 章“启动和关闭”中讨论。

这里，没有数据库。只有 SGA 和后台进程在为创建新数据库的准备中启动。

#### 5. 创建数据库。

要创建一个新的数据库，应使用 SQL CREATE DATABASE 语句，在该语句中可选择地使用一些设置参数以命名数据库、建立文件的最多数值、命名文件并设置大小等等。

要使数据库实用，用户将需要创建附加的文件和表空间。这通常通过运行数据库创建脚本来实现。参阅 2.3.2 节“检查数据库创建脚本”。

#### 6. 运行创建视图、替代名所需要的脚本。

用户必须运行的主要脚本为：

■ CATALOG.SQL——创建数据字典表的视图和动态性能视图

■ CATPROC.SQL——建立 PL/SQL 功能的应用，并创建 Oracle 提供的 PL/SQL 包。

参阅用户特定操作系统的 Oracle 安装指南，以获知这些脚本的位置。

用户运行的脚本是由用户选择使用的特点或选项来决定的。供用户运行的许多脚本都在“Oracle8i Reference”中提供。

#### 7. 备份数据库

用户应该制作数据库的全部备份以保证用户在介质发生错误、进行恢复时得到一个完整的文件系列。关于备份数据库的信息可参阅“Oracle8i Backup and Recovery Guide”。

参见 这些步骤提供了关于在各种操作系统上创建数据库的一般信息。参阅用户特定操作系统的 Oracle 文档，以获取在用户平台创建数据库的信息。

### 2.3.2 检查数据库创建脚本

本节检查和解释数据库的创建脚本，与同操作系统一起发行的例子脚本类似。

#### 数据库创建脚本

这里是一个创建数据库 RBDB1 的数据库创建示例。对于解释脚本的注释，参阅下一小节“注释脚本”。

```
-- Create database
CREATE DATABASE rbdb1
CONTROLFILE REUSE
LOGFILE '/u01/oracle/rbdb1/redo01.log' SIZE 1M REUSE,
        '/u01/oracle/rbdb1/redo02.log' SIZE 1M REUSE,
        '/u01/oracle/rbdb1/redo03.log' SIZE 1M REUSE,
        '/u01/oracle/rbdb1/redo04.log' SIZE 1M REUSE,
DATAFILE '/u01/oracle/rbdb1/system01.dbf' SIZE 10M REUSE,
        AUTOEXTEND ON
        NEXT 10M MAXSIZE 200M
```

```

CHARACTER SET WE8ISO8859PL;

--Create another (temporary) system tablespace
CREATE ROLLBACK SEGMENT rb_temp STORAGE (INITIAL 100 k NEXT 250 k);

--Alter temporary system tablespace online before proceeding
ALTER ROLLBACK SEGMENT rb_temp ONLINE;

--Create additional tablespaces ...
--RBS: For rollback segments
--USERS: Create user sets this as the default tablespace
--TEMP: Create user sets this as the temporary tablespace
CREATE TABLESPACE rbs
    DATAFILE 'u01/oracle/rbdb1/rbs01.dbf' SIZE 5M REUSE AUTOEXTEND ON
    NEXT 5M MAXSIZE 150M;
CREATE TABLESPACE users
    DATAFILE 'u01/oracle/rbdb1/users01.dbf' SIZE 3M REUSE AUTOEXTEND ON
    NEXT 5M MAXSIZE 150M;
CREATE TABLESPACE temp
    DATAFILE 'u01/oracle/rbdb1/temp01.dbf' SIZE 2M REUSE AUTOEXTEND ON
    NEXT 5M MAXSIZE 150M;

--Create rollback segments.
CREATE ROLLBACK SEGMENT rb1 STORAGE (INITIAL 50K NEXT 250K)
    TABLESPACE rbs;
CREATE ROLLBACK SEGMENT rb2 STORAGE (INITIAL 50K NEXT 250K)
    TABLESPACE rbs;
CREATE ROLLBACK SEGMENT rb3 STORAGE (INITIAL 50K NEXT 250K)
    TABLESPACE rbs;
CREATE ROLLBACK SEGMENT rb4 STORAGE (INITIAL 50K NEXT 250K)
    TABLESPACE rbs;

--Bring new rollback segments online and drop the temporary system one
ALTER ROLLBACK SEGMENT rb1 ONLINE;
ALTER ROLLBACK SEGMENT rb2 ONLINE;
ALTER ROLLBACK SEGMENT rb3 ONLINE;
ALTER ROLLBACK SEGMENT rb4 ONLINE;

ALTER ROLLBACK SEGMENT rb_temp OFFLINE;
DROP ROLLBACK SEGMENT rb_temp;

```

### 解释脚本

现在解释上述数据库的脚本。

CREATE DATABASE 语句

```

CREATE DATABASE rbdb1
  CONTROLFILE REUSE
  LOGFILE '/u01/oracle/rbdb1/redo01.log' SIZE 1M REUSE,
         '/u01/oracle/rbdb1/redo02.log' SIZE 1M REUSE,
         '/u01/oracle/rbdb1/redo03.log' SIZE 1M REUSE,
         '/u01/oracle/rbdb1/redo04.log' SIZE 1M REUSE,
  DATAFILE '/u01/oracle/rbdb1/system01.dbf' SIZE 10M REUSE,
  AUTOEXTEND ON
  NEXT 10M MAXSIZE 200M
  CHARACTER SET WE8ISO8859P1;

```

当用户运行 **CREATE DATABASE** 语句时，Oracle 会执行下面的操作：

- 创建数据库的数据文件。
- 创建数据库的控制文件，参阅第 5 章“管理控制文件”。
- 创建数据库的重做日志文件，参阅第 6 章“管理联机重做日志”。
- 创建 **SYSTEM** 表空间和 **SYSTEM** 回退段。
- 创建数据目录。
- 创建用户 **SYS** 和 **SYSTEM**，参阅 1.2.2 节“数据库管理员用户名”。
- 指定数据库中存储数据的字符集。
- 装入、打开数据库。

假设例子中的选项 **MAXLOGFILES**、**MAXLOGMEMBERS**、**MAXDATAFILES**、**MAXLOGHISTORY** 和 **MAXINSTANCES** 的值都采用缺省值，它们依赖于操作系统。数据库按 **NOARCHIVELOG** 和 **EXCLUSIVE** 的缺省模式装入，然后打开。

上面讲述的例子中的项目和信息导致创建的数据库有下面的特点：

- 新数据库命名为 **RBDB1**。
- 新数据库的 **SYSTEM** 表空间由 10MB 的数据文件组成：/u01/oracle/rbdb1/system01.dbf。
- 新数据库有 4 个 1MB 的重做日志文件。
- 新数据库不覆盖在参数文件中指定的现存文件。
- 使用 **WE8ISO8859P1** 字符集。

**注意** 用户在数据库创建期间可以设置几个限制。其中的几个限制也是导致取代操作系统的极限，并且互相影响。例如，如果用户设置 **MAXDATAFILES**，Oracle 在控制文件中分配充足的空间来存储 **MAXDATAFILES** 文件名，即使数据库最初只有一个数据文件。因为最大的控制文件大小是有限的，且是与操作系统相关的，用户可能不能将全部的 **CREATE DATABASE** 参数设置到理论上的最大值。

要获得更多关于数据库创建期间设置极限的信息，请参阅“Oracle8i SQL Reference”和用户特定操作系统的 Oracle 文档。

**参见** 关于 **CREATE DATABASE** 语句、字符集和数据库创建的信息，参阅“Oracle8i SQL Reference”。

### 创建另一个系统回滚段

```
CREATE ROLLBACK SEGMENT rb_temp STORAGE (INITIAL 100 k NEXT 250 k);  
ALTER ROLLBACK SEGMENT rb_temp ONLINE;
```

这些语句在其它的数据库表空间正在创建时创建一个临时的系统回滚段。关于回滚段的讨论，参阅第 11 章“管理回滚段”。

### 为回滚段创建表空间

```
CREATE TABLESPACE rbs  
  DATAFILE 'u01/oracle/rbdb1/rbs01.dbf' SIZE 5M REUSE AUTOEXTEND ON  
  NEXT 5M MAXSIZE 150M;
```

此语句创建表空间来保存回滚段。参阅第 9 章“管理表空间”和第 11 章“管理回滚段”。

### 创建用户表空间

```
CREATE TABLESPACE users  
  DATAFILE 'u01/oracle/rbdb1/users01.dbf' SIZE 3M REUSE AUTOEXTEND ON  
  NEXT 5M MAXSIZE 150M;
```

此语句创建一个在用户概要中可以作为缺省表空间来分配的表空间。参阅第 9 章“管理表空间”和第 22 章的“分配缺省表空间”。

### 创建临时表空间

```
CREATE TABLESPACE temp  
  DATAFILE 'u01/oracle/rbdb1/temp01.dbf' SIZE 2M REUSE AUTOEXTEND ON  
  NEXT 5M MAXSIZE 150M;
```

临时表空间对排序操作有特殊的用途。可以在用户概要中分配给用户临时表空间。参阅第 9 章“管理表空间”和第 22 章的“分配缺省表空间”。

### 创建回滚段

```
CREATE ROLLBACK SEGMENT rb1 STORAGE (INITIAL 50K NEXT 250K)  
  Tablespace rbs;  
CREATE ROLLBACK SEGMENT rb2 STORAGE (INITIAL 50K NEXT 250K)  
  Tablespace rbs;  
CREATE ROLLBACK SEGMENT rb3 STORAGE (INITIAL 50K NEXT 250K)  
  Tablespace rbs;  
CREATE ROLLBACK SEGMENT rb4 STORAGE (INITIAL 50K NEXT 250K)  
  Tablespace rbs;
```

--Bring new rollback segments online and drop the temporary system one

```
ALTER ROLLBACK SEGMENT rb1 ONLINE;  
ALTER ROLLBACK SEGMENT rb2 ONLINE;  
ALTER ROLLBACK SEGMENT rb3 ONLINE;  
ALTER ROLLBACK SEGMENT rb4 ONLINE;
```

```
ALTER ROLLBACK SEGMENT rb_temp OFFLINE;  
DROP ROLLBACK SEGMENT rb_temp;
```

这一系列的语句可创建为用户事务使用的回滚段。在最初创建时，它们是 **OFFLINE**。它们必须明确的变成联机。而且，临时的系统回滚段现在也是脱机的，然后撤消。

更多的信息参阅第 11 章“管理回滚段”。

### 2.3.3 发现并修复数据库创建的故障

如果由于某种原因数据库创建失败，那么在试图再次创建时，关闭这个事例并删除由 **CREATE DATABASE** 所创建的文件。

在纠正了造成数据库创建失败的错误之后，再试着重新运行脚本。

### 2.3.4 丢弃数据库

要丢弃一个数据库，应删除它的数据文件、重做日志文件以及所有其它的相关文件（控制文件、参数文件、归档日志文件）。

要浏览数据库的数据文件和重做日志文件的名称，可以询问数据字典的 **V\$DATAFILE** 和 **V\$LOGFILE** 视图。

参见 关于这些视图的更多信息，参阅“Oracle8i Reference”。

## 2.4 安装参数

在创建数据库的步骤开始时，用户会希望编辑 Oracle 提供的安装参数文件。Oracle 的意图是在初始参数文件启动器中提供合适的值；我们建议用户改变最小的参数。当用户对于数据库和环境变得更熟悉时，可以通过 **ALTER SYSTEM** 语句来动态地调整这些参数。用户想要永久改变这些参数中的任意参数时，都应在安装参数文件中更新。

本节讨论下面的题目：

■ 一个安装文件的例子

■ 编辑安装参数文件

参见 关于安装参数和所有这些参数的描述的更多信息，请参阅“Oracle8i Reference”。

### 2.4.1 一个安装文件的例子

列在这里的是 Oracle 提供的安装参数文件的例子，这个文件已经编辑过，可以用 **RBDB1** 数据库来使用。用户要注意，在脚本中，Oracle 已经提供了设置这些安装参数的指南。

```
#####  
# Example INIT.ORA file  
#  
# This file is provided by Oracle Corporation to help you customize  
# your RDBMS installation for your site. Important system parameters  
# are discussed, and example settings given.  
#  
# Some parameter settings are generic to any size installation.
```

```

# For parameters that require different values in different size
# installations, three scenarios have been provided: SMALL, MEDIUM
# and LARGE. Any parameter that needs to be tuned according to
# installation size will have three settings, each one commented
# according to installation size.
#
# Use the following table to approximate the SGA size needed for the
# three scenarios provided in this file:
#
#          -----Installation/Database Size-----
#          SMALL          MEDIUM          LARGE
# Block      2k      4500k      6800k      17000k
# Size      4k      5500k      8800k      21000k
#
# To set up a database that multiple instances will be using, place
# all instance-specific parameters in one file, and then have all
# of these files point to a master file using the IFILE command.
# This way, when you change a public
# parameter, it will automatically change on all instances. This is
# necessary, since all instances must run with the same value for many
# parameters. For example, if you choose to use private rollback segments,
# these must be specified in different files, but since all gc_*
# parameters must be the same on all instances, they should be in one file.
#
# INSTRUCTIONS: Edit this file and the other INIT files it calls for
# your site, either by using the values provided here or by providing
# your own. Then place an IFILE= line into each instance-specific
# INIT file that points at this file.
#
# NOTE: Parameter values suggested in this file are based on conservative
# estimates for computer memory availability. You should adjust values upward
# for modern machines.
#
#####

db_name = RBDB1

db_files = 1024                                # INITIAL
# db_files = 80                                # SMALL
# db_files = 400                               # MEDIUM
# db_files = 1500                              # LARGE

```

```

control_files = ("/u01/oracle/rbdb1/control01.ctl",
                "/u01/oracle/rbdb1/control02.ctl")

db_file_multiblock_read_count = 8                # INITIAL
# db_file_multiblock_read_count = 8              # SMALL
# db_file_multiblock_read_count = 16             # MEDIUM
# db_file_multiblock_read_count = 32             # LARGE

db_block_buffers = 8192                          # INITIAL
# db_block_buffers = 8192                        # SMALL
# db_block_buffers = 8192                        # MEDIUM
# db_block_buffers = 8192                        # LARG

shared_pool_size = 15728640                       # INITIAL
# shared_pool_size = 3500000                     # SMALL
# shared_pool_size = 5000000                     # MEDIUM
# shared_pool_size = 9000000                     # LARGE

log_checkpoint_interval = 10000
log_checkpoint_timeoit = 1800

processes = 59                                    # INITIAL
# processes = 50                                  # SMALL
# processes = 100                                 # MEDIUM
# processes = 200                                 # LARGE

parallel_max_servers = 5                          # SMALL
# parallel_max_servers = 4 x (number of CPUs)     # MEDIUM
# parallel_max_servers = 4 x (number of CPUs)     # LARGE

log_buffer = 32768                                # INITIAL
# log_buffer = 32768                             # SMALL
# log_buffer = 32768                             # MEDIUM
# log_buffer = 163840                             # LARGE

#audit_trail = true # if you want auditing
#timed_statistics = if you want timed statistics
max_dump_file_size = 10240 # limit trace file size to 5M each

# Uncommenting the line below will cause automatic archiving if archiving has
# been enabled using ALTER DATABASE ARCHIVELOG.
# log_archive_start = true
# log_archive_dest_1 = "location=/u01/oracle/rbdb1/archive"
# log_archive_format = "%%RDBD1%%T%TS%S.ARC"

```



```

# If using private rollback segments, place lines of the following
# form in each of your instance-specific init.ora files:
rollback_segments = (rb1, rb2, rb3, rb4)

# If using public rollback segments, define how many
# rollback segments each instance will pick up, using the formula
# # of rollback segments = transactions / transactions_per_rollback_segment
# In this example each instance will grab 40/5 = 8
# transactions = 40
# transactions_per_rollback_segment = 5

# Global Naming -- enforce that a dblink has same name as the db it connects to
global_names = true

# Edit and uncomment the following line to provide the suffix that will be
# appended to the db_name parameter (separated with a dot) and stored as the
# global database name when a database is created. If your site uses
# Internet Domain names for e-mail, then the part of your e-mail address after
db_domain = us.acme.com
#global database name is db_name.db_domain
compatible = 8.1.0

```

## 2.4.2 编辑安装参数文件

要创建新的数据库，有一些用户希望编辑的安装参数。根据用户的配置和选项，以及用户希望怎样调整数据库，可以有其它的安装参数来让用户编辑或添加。整个这本书都讨论了一些其它的安装参数。

用户也应该添加合适的许可安装参数。

下面的章节描述了下列参数：

- DB\_NAME 和 DB\_DOMAIN
- CONTROL\_FILES
- DB\_BLOCK\_BUFFERS
- PROCESSES
- ROLLBACK\_SEGMENTS
- 许可参数

### DB\_NAME 和 DB\_DOMAIN

一个数据库的全局数据库名（global database name）（网络结构中的名字和地址）是在数据库创建之前通过设置 DB\_NAME 和 DB\_DOMAIN 参数来创建的。创建之后，数据库的名字不能很容易地改变，用户必须重新创建控制文件。DB\_NAME 参数决定了数据库名的本地名部分，而 DB\_DOMAIN 参数指出网络结构中的主域（逻辑地址）。这两个参数的设

置组合应该形成网络中唯一的数据库名。例如，要创建一个全局数据库名为 TEST.US.ACME.COM 的数据库，应通过如下方式编辑新参数文件的参数：

```
DB_NAME=TEST
DB_DOMAIN=US.ACME.COM
```

DB\_NAME 必须设置一个不超过 8 个字符的文本串。在数据库创建期间，由 DB\_NAME 提供的名字记录在数据库的数据文件、重做日志文件和控制文件中。如果在数据库事例启动期间，DB\_NAME 参数（参数文件中的）的值和控制文件中的数据库名不一样，那么数据库不能启动。

DB\_DOMAIN 是指定创建数据库所在网络主域的一个文本串，它通常是拥有数据库的组织的名称。如果用户准备创建的数据库要成为发行的数据库系统的一部分，就要特别注意在数据库创建之前的安装参数。

参见 关于分布式数据库的信息，请参阅“Oracle8i Distributed Database Systems”。

### CONTROL\_FILES

CONTROL\_FILES 参数包括在用户的新参数文件中，并被设置到一系列新数据库使用的控制文件。如果用户希望在创建数据库控制文件时，Oracle 创建新的操作系统文件，就要保证在 CONTROL\_FILES 参数中所列的文件名不与当前用户系统中所存在的任何文件名相匹配。如果在创建控制文件时用户希望 Oracle 重复利用或者覆盖现存的文件，就要保证在 CONTROL\_FILES 参数中所列的文件名与当前用户系统中所存在的文件名相匹配。

**警告** 设置这个选项时要特别的小心。当用户不经意地指定一个用户不希望的文件时运行 CREATE DATABASE 语句，那个文件以前的内容就会被覆盖。

如果没有列出 CONTROL\_FILES 参数的文件名，Oracle 会使用一个缺省的名字。

Oracle 公司强烈地提醒用户对于每一个数据库，至少要使用存储在不同物理磁盘上的两个控制文件。因此，当指定新参数文件中的 CONTROL\_FILES 参数时，要遵循下面的指导：

- 列出至少两个 CONTROL\_FILES 参数的文件名。
- 通过每个文件名的指向不同的磁盘驱动器的完全确定的文件名，将每一个控制文件放在独立的物理磁盘驱动器上。

**注意** 确定控制文件的文件与操作系统相关。不管用户的操作系统是什么，总是要完整地确定控制文件的文件名。

当用户运行 CREATE DATABASE 语句(在步骤 7 中)时，在参数文件的 CONTROL\_FILES 参数中所列的控制文件将被创建。

参见 CONTROL\_FILES 参数的缺省文件名是与操作系统相关的。参阅用户的特定操作系统的 Oracle 文档，以获详解。

### DB\_BLOCK\_SIZE

每个 Oracle 服务器缺省的数据块大小随操作系统而定。Oracle 数据块的典型大小为 2K 或 4K。一般来说，缺省的数据块大小是唯一的。然而，在某些情况下，较大的数据块大小提供较高的磁盘和内存 I/O（访问和存储数据）的效率。这些情况有：

- 在拥有大量内存和高速磁盘驱动器的大型计算机系统上的 Oracle。例如，有带有大

量硬件资源的大型计算机控制的数据库典型使用 4K 或更大的数据块大小。

- 运行 Oracle 的操作系统使用较小的操作系统数据块。例如，如果操作系统数据块是 1K，并且数据块的大小与之匹配，Oracle 可能在正常操作时执行过量的磁盘 I/O。这种情况下，为了获得最好的性能，数据块应该包含多个操作系统块。

每一个数据块的大小可以在数据库创建期间由安装参数 DB\_BLOCK\_SIZE 来设定。块的大小不能在数据库创建后改变，除非重新创建数据库。如果数据库的块大小与操作系统块的大小不同，就要将数据库块的大小设置为操作系统块大小的整倍数。

例如，如果用户的操作系统的块大小是 2K（2048 字节），下面的 DB\_BLOCK\_SIZE 安装参数的设置是有效的：

```
DB_BLOCK_SIZE=4096
```

DB\_BLOCK\_SIZE 也决定了在系统全局区域（SGA）高速缓冲存储器中的数据库缓冲器的大小。

参见 关于用户缺省块的大小的细节，可参阅用户的特定操作系统的 Oracle 文档。

#### DB\_BLOCK\_BUFFERS

这个参数决定了系统全局区域（SGA）高速缓冲存储器中的数据库缓冲器的个数。缓冲器的个数影响了高速缓冲存储器的性能。较大的高速缓冲存储器减少了磁盘修改数据的写次数。然而，较大的高速缓冲存储器会占据太多的内存，减少内存分页或交换。

估计用户最常访问的应用程序的数据块的数量，包括表、索引和回滚段。这个估计是一个高速缓冲存储器应有缓冲器的大概的近似值。典型的，1000 到 2000 是缓冲器数量的实际的最小值。

参见 关于调整高速缓冲存储器的缓冲器更多的信息，参阅 “Oracle8i Designing and Tuning for Performance.”。

#### PROCESSES

这个参数决定了可以并发连接到 Oracle 上的操作系统进程的最大数量。这个参数的值必须包括后台进程 5 和每个用户进程 1。例如，如果你计划有 50 个并发用户，则将这个参数至少设置为 55。

#### ROLLBACK\_SEGMENTS

这个参数是在数据库启动时要求的 Oracle 事例回滚段的列表。作为这个参数的值，列出了用户的回滚段。

注意 在安装之后，除了在用户可以创建任何模式对象之前的 SYSTEM 回滚段以外，用户必须在 SYSTEM 表空间中创建至少一个回滚段。

#### 许可参数

Oracle 帮助你保证你所在的地方遵守 Oracle 许可协议。如果你所在的地方用并发应用来授权，用户可以跟踪并限制并发连接到事例的会话数量。如果用户是由指名用户授权的，可以限制在数据库中创建的命名用户的数量。要使用这个工具，用户需要知道用户拥有哪种

许可协议，以及最大值的会话或指名用户是什么。用户的地点应该使用某一种许可（会话许可或者指名用户许可），而不能两个都用。

关于管理许可的更多信息，参阅第 22 章的“会话和用户的许可”。

`LICENSE_MAX_SESSIONS` 和 `LICENSE_SESSIONS_WARNING`

用户可以对可以连接到特定计算机的数据库上的并发会话设置一个限制。要为一个事例的并发会话设置最大的数量，应该设置启动事例的参数文件中的参数 `LICENSE_MAX_SESSIONS`，如下面的例子所示：

```
LICENSE_MAX_SESSIONS = 80
```

另外，设置会话数量的最大值，用户可以在并发会话的数量上设置一个警告限制，一旦达到这个限制，附加的用户可以继续连接（直到限制的最大值），但是，Oracle 给所连接的用户发出一个警告。要设置事例的警告界限，就要设置参数 `LICENSE_MAX_WARNING`。要将警告限制设置为小于 `LICENSE_MAX_SESSIONS` 的值。

对于并行服务器上运行的事例，每一个事例都可以由自己的并发应用限制和警告限制。然而，事例的限制总和必须不超过用户的会话许可。

参见 关于在使用并行服务器上设置许可限制的更多信息，请参阅“Oracle8i Parallel Server Administration, Deployment, and Performance”和“Oracle8i Parallel Server Setup and Configuration Guide”。

`LICENSE_MAX_USERS` 用户可以设置数据库中创建用户数量的限制。一旦达到了这个极限，你将不能再创建更多的用户。

**注意** 这个机制假定每一个正在访问数据库的人有唯一的用户名称，并且没有人共享用户名称。因此，指名用户许可可以帮助用户保证用户遵守 Oracle 许可协议，不允许多个用户用相同的用户名来注册。

要限制在数据库中创建的用户数量，可以在数据库的参数文件中设置 `LICENSE_MAX_USERS` 参数。如下面例子所示：

```
LICENSE_MAX_USERS = 200
```

对于在并行服务器上运行的事例，所有连接到相同数据库上的事例应该有相同名称的用户限制。

## 2.5 数据库创建之后应考虑的事项

在用户完成数据库的创建之后，接着就是事例的运行了，并且数据库是打开，可用于一般的数据库用途。如果在用户的数据库系统中有一个以上的数据库存在，用一个后来的数据库启动来确定要使用的参数文件。

如果用户打算安装其它的 Oracle 产品来使这个数据库工作，请参阅这些产品的安装指导。一些产品要求用户创建附加的数据目录表。参阅用户的特定操作系统附加产品的 Oracle 文档。通常，提供命令文件来创建表，并将这些表装入数据库的数据目录。

Oracle 服务器的发行介质可以包括各种各样的 SQL 文件以便用户可以用系统试验、学习 SQL，或创建附加的表、视图、或替代名。

一个新创建的数据库仅有两个用户：**SYS** 和 **SYSTEM**。这两个用户名的密码应该在数据库创建之后迅速更改。关于用户 **SYS** 和 **SYSTEM** 的更多信息，参阅 1.2.2 节“数据库管理员用户名”。

关于更改用户密码的更多的信息，参阅第 22 章的“更改用户”。

## 2.6 初始化调整指导

用户可以立即在安装之后对 **Oracle** 作一些显著的调整更改。通过遵循这些指导，用户可以在 **Oracle** 运行时缩减对调整 **Oracle** 的需求。本节通过下面的安装事项提供一些建议：

- 分配回滚段
- 选择 **DB\_BLOCK\_LRU\_LATCHES** 的数量
- 分配 I/O

参见 关于调整这些安装参数的更多信息，请参阅“*Oracle8i Designing and Tuning for Performance*”。

### 2.6.1 分配回滚段

适当的回滚段分配有助于获得最佳的数据库性能。最佳性能所要求的回滚段的大小和数量取决于用户的应用程序。“*Oracle8i Designing and Tuning for Performance*”包含一些关于在基于用户 **Oracle** 服务器的并发事务的数量选择应该分配多少个回滚段的一般性指导。这些指导对大多数的应用程序混合是合适的。

要创建回滚段，可使用 **CREATE ROLLBACK SEGMENT** 语句。用户回滚段的大小也可能影响性能。回滚段的大小由 **CREATE ROLLBACK SEGMENT** 语句中的存储参数来确定。用户的回滚段必须足够大，以便可以保存用户事务的全部回滚项。

### 2.6.2 选择 **DB\_BLOCK\_LRU\_LATCHES** 的数量

**LRU**(最近最少使用算法，least recently used)锁存器的争用，可以阻碍拥有大量 **CPU** 的对称多处理器（**SMP**）机器的性能。**LRU** 锁存器控制在高速缓冲存储器中的缓冲器的替换。对于 **SMP** 系统，**Oracle** 自动将 **LRU** 锁存器的数量设置成系统中 **CPU** 数量的一半。对于非 **SMP** 系统，一个 **LRU** 锁存器就足够了。

用户可以指定用户系统的 **LRU** 锁存器的数量，这要通过安装参数 **DB\_BLOCK\_LRU\_LATCHES**。这个参数设置所要求的 **LRU** 锁存器的最大值。每一个 **LRU** 锁存器将控制一系列的缓冲器，**Oracle** 平衡替代缓冲器的分配。

### 2.6.3 分配 I/O

适当地分配 **I/O** 可以戏剧性地提高数据库的性能。**I/O** 可以在 **Oracle** 安装时分布。在安装时分布 **I/O** 可以缩减后来在 **Oracle** 运行时分布 **I/O** 的需求。

在安装 **Oracle** 时可以用如下几种方法来分布 **I/O**：

- 重做日志文件安排
- 数据文件安排
- 表和索引的分离
- 数据密度（每个数据块有多少行）

# 3 启动与关闭

本章描述启动和停止 Oracle 数据库的过程，包括下面的专题：

- 启动数据库
- 更改数据库的可用性
- 关闭数据库
- 挂起和继续执行数据库
- 使用初始化参数文件

## 3.1 启动数据库

在用户启动数据库时，可创建这个数据库的事例，也可选择数据库启动的状态。一般来说，用户应该通过安装（mounting）和打开（opening）数据库来启动事例，这样，会产生与之连接的有效用户的可用性，并执行典型的数据访问操作。然而，还有其它的选项，也在本节中进行讨论。

本节包括下面与启动一个数据库事例相关的专题：

- 准备启动事例
- 启动数据库的选项
- 启动事例：方案

### 3.1.1 准备启动事例

用户需要在打算启动数据库事例之前执行一些预备步骤：

1. 在不连接数据库的情况下通过键入来启动 SQL\*Plus，

SQLPLUS

2. 以 SYSDBA 方式连接 Oracle：

CONNECT username/password AS sysdba

注意用户不能通过多线程服务器来连接。

现在用户被连接到 Oracle，并准备启动用户数据库的事例。

参见 CONNECT、STARTUP、和 SHUTDOWN 是 SQL \*Plus 的命令。它们在“SQL \*Plus User's Guide and Reference”中描述，语法也在其中给出。

### 3.1.2 启动数据库的选项

用户启动（和管理）数据库事例所用方法有多种。虽然提及有三种方法，但本书中只使用 SQL\*Plus 这一种方法。

## 使用 SQL\*Plus

要启动数据库，可使用 SQL\*Plus 来以系统管理员权限（如前所述）连接到数据库，然后发出 STARTUP 命令。当用户输入 STARTUP 命令时，用户可以确定数据库的名字以及初始化参数文件的全路径：

```
STARTUP database_name PFILE=myinit.ora
```

如果用户不确定 database\_name，则 Oracle 使用由指定的 PFILE 中的 DB\_NAME 初始化参数所指定的值。Oracle 使用缺省的初始化参数文件位置，就象在针对用户操作系统的 Oracle 安装指南中所描述的那样。

用户可以有不同的方法来启动一个事例和数据库：

- 不装入数据库而启动事例。这样不允许访问数据库，通常是在数据库创建或控制文件的创建时这样做。
- 启动事例并装入数据库，但是不打开。这种状态在特定的 DBA 激活时才可使用，但不允许对数据库进行一般的访问操作。
- 启动事例装入并打开数据库。这样可以在非严格模式下进行，允许访问所有的用户，或在严格模式下，只允许系统管理员来访问。

**注意** 用户如果通过多线程服务器程序来连接数据库，则不能启动数据库事例。

另外，用户可以强制事例启动，或启动事例并立即恢复全部的介质。如果用户操作系统支持 Oracle 并行服务器(OPS)，用户可以在独占或共享方式下启动事例和装入数据库。

## 使用恢复管理器

用户也可以使用恢复管理器（RMAN）来执行 STARTUP(和 SHUTDOWN)命令。如果用户在 RMAN 环境并不希望调出来 SQL\*Plus，则用户可以倾向于这样做。本书不讨论 RMAN，但是它在“Oracle8I Recovery Manager User's Guide and Reference”中讨论。

## 使用 Oracle 企业管理器

用户可以选择使用 Oracle 企业管理器来管理用户的数据库，包括启动和关闭。Oracle 企业管理器是与 Oracle 产品分立的，包括了图形控制台、助理、一般服务、提供集成和理解系统管理平台来管理 Oracle 产品的工具。它允许用户执行在本书中讨论的功能（通过使用 GUI 界面，而不是用命令行）。

参阅下面几本书，可以熟悉 Oracle 企业管理器：

- “Oracle Enterprise Manager Concepts Guide”
- “Oracle Enterprise Manager Administrator's Guide”

### 3.1.3 启动事例：方案

下面的方案描述并图示用户可以启动事例的若干方法。关于在合并 STARTUP 命令选项时所施加的限制的更多信息，参阅“SQL\*Plus User's Guide and Reference”。

**注意** 如果控制文件、数据库文件、或重做日志文件没有提供时，用户会在启动事例时遇到问题。当用户装入数据库时，如果一个或多个在 CONTROL-FILES 参数中指定的文件不存在或不能打开，Oracle 会返回一个警告信息但不装入数据库。如果



在试图打开数据库时，一个或多个数据文件或重做日志文件不可用或不能打开，Oracle 返回一个警告信息，且不打开数据库。

### 不装入数据库而启动事例

用户可以不装入数据库而启动事例。典型情况下，用户只是在数据库创建时才可以这样做。使用 NOMOUNT 选项的 STARTUP 命令：

```
STARTUP NOMOUNT;
```

### 启动事例并装入数据库

用户可以启动事例并装入数据库但不打开数据库，允许用户执行特定的维护操作。例如，可以在下面的任务中装入数据库但不打开：

- 重命名数据文件，如第 10 章“管理数据文件”中所述。
- 添加、撤消、或重命名重做日志文件，如第 6 章“管理联机重做日志”中所述。
- 启用和禁止重做日志存档选项，如第 7 章“管理存档重做日志”中所述。
- 执行全部的数据库恢复。数据库恢复是“Oracle8I Backup and Recovery Guide”和“Oracle8I Recovery Manager User's Guide and Reference”的专题。

启动事例并装入数据库但不打开，可用带有 MOUNT 选项的 STARTUP 命令：

```
STARTUP MOUNT;
```

### 启动事例装入并打开数据库

一般数据库操作(Normal database operation)指的是一个事例被启动，数据库装入并打开；这个模式允许任何有效用户连接到数据库并执行典型的数据访问操作。

启动事例然后装入并打开数据库就通过使用 STARTUP 命令本身（这个例子使用数据库的名字由标准的 PFILE 中的 DB\_NAME 初始化参数指定）：

```
STARTUP;
```

### 限制在启动时对数据库的访问

用户可以在严格模式下启动事例并装入打开数据库，这样数据库仅对管理者个人有效（而不是一般的数据库用户）。当用户需要完成下面任务之一时使用这个数据库启动模式：

- 执行结构维护，如重建索引
- 执行数据库文件的导入或导出
- 执行数据装载（用 SQL\*Loader）
- 临时阻止典型用户使用数据

典型情况下，所有 CREATE SESSION 系统权限的用户都可以访问一个打开的数据库。在严格模式下打开数据库仅允许同时拥有 CREATE SESSION 和 RESTRICTED SESSION 系统权限的用户来进行数据库访问；只有数据库系统管理员有 RESTRICTED SESSION 系统权限。

在严格模式下启动事例（并且，随意地装入和打开数据库），使用带有 RESTRICT 选项的 STARTUP 命令：

STARTUP RESTRICT;

然后，使用 **ALTER SYSTEM** 语句来禁止 **RESTRICTED SESSION** 特性。如果用户用非严格模式打开数据库，之后又发现需要严格访问，可以使用 **ALTER SYSTEM** 来实现，如 3.2.4 节“限制访问一个打开的数据库”中所述。

参见 关于 **ALTER SYSTEM** 语句的更多的信息，参阅 “Oracle8I SQL Reference”。

### 强制事例启动

在特殊的环境下，用户可能在试图启动数据库事例时遇到问题。用户应该强制数据库启动，除非遇到下面的情况：

- 用户不能用 **SHUTDOWN NORMAL**、**SHUTDOWN IMMEDIATE**、或 **SHUTDOWN TRANSACTIONAL** 命令来关闭当前的事例。

- 在启动事例时遇到问题。

如果上述问题之一发生，用户一般可以通过启动一个新的事例（并随意地装入并打开数据库）来解决问题，使用带有 **FORCE** 选项的 **STARTUP** 命令：

**STARTUP FORCE;**

如果一个事例正在运行，**STARTUP FORCE** 则在重新启动之前用 **ABORT** 模式将之关闭。要理解终止当前事例的影响，参阅 3.3.3 节“带有终止选项的关闭”。

### 启动一个事例，装入数据库，并启动全部的介质恢复

如果用户知道所要求的介质恢复，可以启动一个事例，装入指向事例的数据库，并自动地启动恢复程序。可使用带有 **RECOVER** 选项的 **STARTUP** 命令。

**STARTUP OPEN RECOVER;**

如果用户在没有要求恢复的情况下希望执行恢复，Oracle 会发出一个错误信息。

### 启动独占或并行模式

如果用户的 Oracle 服务器允许多个事例来并发的访问一个数据库（Oracle 并行服务器选项），选择是独占的或是并行的装入数据库。例如，要用并行模式打开，则发出：

**STARTUP OPEN sales PFILE=init sale.ora PARALLEL;**

现在，多个事例可以访问数据库了。

如果用户指定独占模式（缺省），那么数据库只能由当前的事例装入并打开。下面是一个用独占模式启动的事例，装入并打开一个名为 **sales** 的数据库的例子，并且限制管理者个人来访问：

**STARTUP OPEN sales PFILE=init sale.ora EXCLUSIVE RESTRICT;**

参见 关于用独占或并行模式启动的更多信息，参阅手册 “Oracle8I Parallel Server Administration, Deployment, and Performance”。

### 在操作系统启动时的自动的数据库启动

很多地点使用允许一个或多个 Oracle 事例自动启动的进程，数据库可以随着系统立即启动。执行这一任务的进程随着操作系统不同而不同。关于自动启动进程专题的更多的信息，

参阅用户特定操作系统的 Oracle 文档。

### 启动远程事例

如果用户本地的 Oracle 服务器是分布式数据库的一部分，可能需要启动一个远程的事例和数据库。启动和关闭远程事例的程序基本上取决于通讯协议和操作系统。

## 3.2 更改数据库的可用性

用户可以更改一个数据库的可用性。用户可能希望为了限制维护原因而访问或使数据库只读而这样做。下面的几节解释了怎样更改数据库的可用性：

- 将数据库装入事例
- 打开一个关闭的数据库
- 在只读模式下打开数据库
- 限制对打开数据库的访问

### 3.2.1 将数据库装入事例

当用户需要执行特定的管理操作时，数据库必须启动，并装入事例，但是处于关闭状态。用户可以通过启动事例并装入数据库来获得这一方案。

在装入数据库时，指出是将数据库独占地装入这个事例，还是并发地装入其它事例。要将数据库装入先前启动的事例，可使用带有 MOUNT 选项的 SQL 语句 ALTER DATABASE。在用户需要以独占模式装入数据库时，可使用如下的语句：

```
ALTER DATABASE MOUNT;
```

对于要求数据库装入并关闭（一步启动事例并装入数据库的程序）的操作列表，参阅 3.1.3 节中的“启动事例并装入数据库”。

### 3.2.2 打开一个关闭的数据库

用户通过操作数据库可以使一个已装入但关闭的数据库用于一般的使用。要打开一个装入的数据库，可使用带有 OPEN 选项的 ALTER DATABASE 语句：

```
ALTER DATABASE OPEN;
```

在运行了这个语句之后，任何拥有 CREATE SESSION 系统权限的有效的 Oracle 用户都可以连接到这个数据库了。

### 3.2.3 在只读模式下打开数据库

在只读状态下打开数据库，使得用户可以询问一个打开的数据库，同时排除了任何潜在的联机数据内容改变。在只读状态下打开数据库，可保证数据文件和重做日志文件不可写。它不限制数据的恢复或“声明”不生成重做的修改。例如，用户可以使数据文件脱机，或在这些操作不影响数据内容的情况下将它们联机。

理想情况下，用户在更改一个只读和恢复模式之间的备用数据库时用只读模式打开数据库。注意只读和恢复是互相排斥的模式。

下面的语句用只读模式打开数据库:

```
ALTER DATABASE OPEN READ ONLY;
```

用户也可以这样用只读模式打开数据库:

```
ALTER DATABASE OPEN READ WRITE;
```

**注意** 用户不能使用带有 READ ONLY 子句的 RESETLOGS 子句。

参见 关于 ALTER DATABASE 语句的更多的信息, 参阅 “Oracle 8I SQL Reference”。

关于只读模式下打开数据库的更多概念, 参阅 “Oracle8I Concepts”。

### 3.2.4 限制对打开数据库的访问

要将事例置成限制模式, 可使用带有 ENABLE RESTRICTED SESSION 子句的 SQL 语句 ALTER SYSTEM。在将一个事例置成限制模式后, 用户可能希望在执行和管理任务之前, 删除所有的用户会话。要将一个事例从限制模式中释放出来, 使用带有 DISABLE RESTRICTED SESSION 选项的 ALTER SYSTEM。

关于用户要将事例置成限制模式的原因, 参阅 3.1.3 节中的 “限制访问启动的数据库”。

## 3.3 关闭数据库

下面几节描述关闭程序的过程:

- 用 NORMAL 选项关闭
- 用 IMMEDIATE 选项关闭
- 用 TRANSACTIONAL 选项关闭
- 用 ABORT 选项关闭

要开始数据库的关闭, 可用 SQL\*Plus 的 SHUTDOWN 命令, 直到关闭结束。否则, 初始化数据库的会话不能恢复到控制状态。希望连接的用户在关闭过程中会收到如下的信息:

```
ORA-01090: shutdown in progress - connection is not permitted
```

**注意** 如果用户已通过多线程服务器程序连接到数据库, 则不能关闭数据库。

要关闭数据库和事例, 先用 SYSOPER 或 SYSDBA 来连接。图 3-1 表示了一个银行帐号到另一个帐号的基金的传送过程中不同的 SHUTDOWN 命令输入时事件的顺序。

### 3.3.1 用 NORMAL 选项关闭

一般数据库的关闭在下面的情况下进行:

- 在语句发出后不允许新的连接。
- 在数据库关闭之前, Oracle 等待所有当前与数据库连接的用户断开连接。
- 数据库的下一次启动将不需要任何事例恢复程序。

要在一般状态下关闭数据库, 使用带有 NORMAL 选项的 SHUTDOWN 命令:

```
SHUTDOWN NORMAL;
```

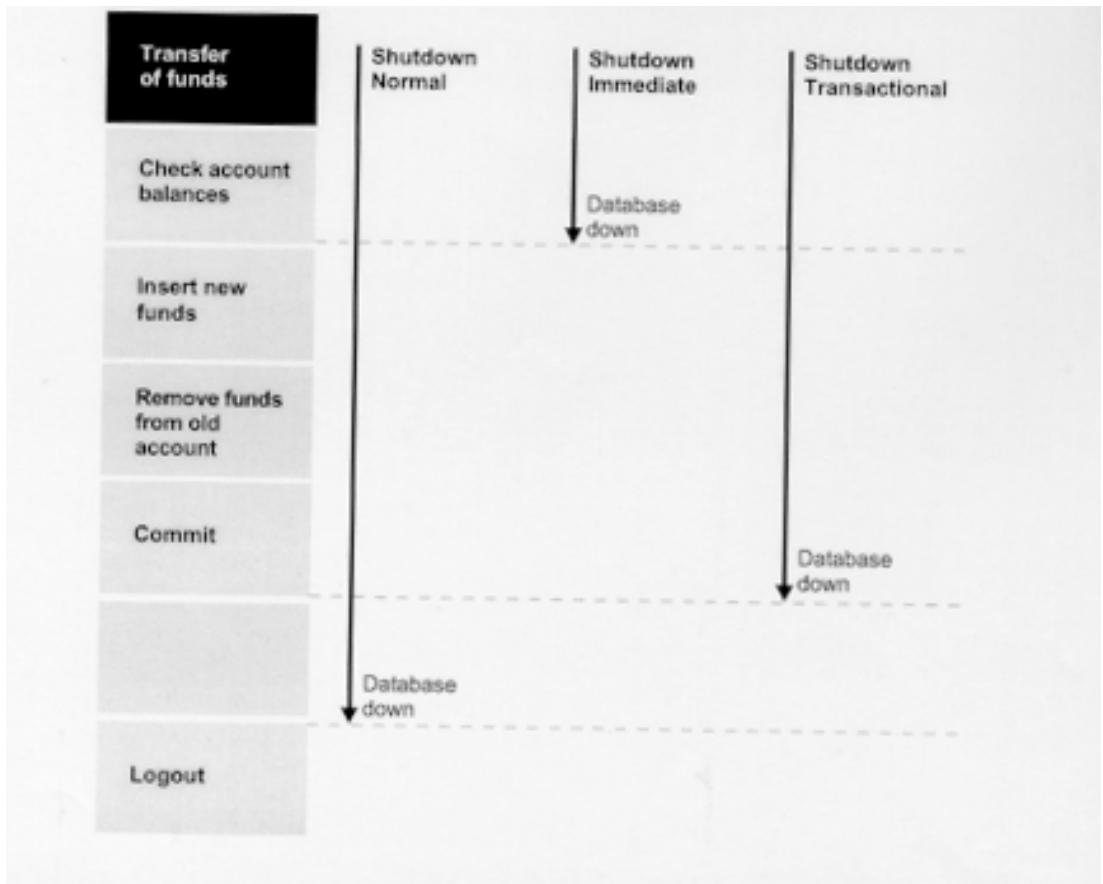


图 3-1 以不同形式关闭事件的顺序

### 3.3.2 用 IMMEDIATE 选项关闭

仅在下列情况下使用立即数据库关闭：

- 当很快就要关电源时。
  - 数据库或其应用程序功能不正常时。
- 立即数据库关闭通过下面的情况进行：

- 未授权的事务退回（如果长期有未授权的事务存在，这种关闭方法可能不会完成得很快，虽然名字叫做“立即”）。
- Oracle 不等待用户当前的数据库连接断开；Oracle 不问原因地退回激活事务，并断开所有的连接用户。
- 数据库的下次启动将不要求任何事例恢复程序。

要立即关闭数据库，用 IMMEDIATE 选项的 SHUTDOWN 命令：

SHUTDOWN IMMEDIATE;

**注意** SHUTDOWN IMMEDIATE 语句会断开所有现存的空闲连接，并关闭数据库。然而，如果用户已经递交了要等待结果的处理（例如：插入，选择或和升级），SHUTDOWN

TRANSACTIONAL 语句允许进程在断开连接之前完成处理。

### 3.3.3 带有 TRANSACTIONAL 选项的关闭

当用户希望在允许活动事务首先完成时进行一个计划中的事例关闭时，应使用带有 TRANSACTIONAL 选项的关闭：

```
SHUTDOWN TRANSACTIONAL;
```

在递交了这个语句之后，客户端就不能启动这个事例的新事务了。如果客户端试图启动一个新的事务，它们将断开连接。在所有的事务完成之后，仍然连接的客户端都将断开连接。在这一点上，事例的关闭就象当一个 SHUTDOWN IMMEDIATE 命令递交了一样。数据库的下一个启动将不要求任何的事例恢复程序。

一个事务的关闭可防止客户端丢失工作，并且同时不要求所有的用户注销。

### 3.3.4 带有 ABORT 选项的关闭

用户可以通过退出数据库的事例来马上关闭数据库。如果可能，只在下面的情况下执行这种形式的关闭。

- 数据库或它的应用程序功能不正常，并且其它形式的关闭不起作用时。
- 用户需要立即关闭数据库时（例如，如果用户知道一分钟之内就要停电时）。
- 用户在启动数据库事例时遭遇问题时。

退出事例关闭数据库会产生如下的后果：

- Oracle 处理的当前客户端 SQL 语句立即中断。
- 未递交的事务将不会回退。
- Oracle 将不等待当前与数据库的连接断开；Oracle 立即断开所有的连接用户。
- 数据库的下一次启动将要求事例恢复程序。

如果正常与立即关闭选项都不能工作，要立即退出当前的数据库事例，用带有 ABORT 选项的 SHUTDOWN 命令：

```
SHUTDOWN ABORT;
```

## 3.4 挂起和继续执行数据库

ALTER SYSTEM SUSPEND 语句通过停止所有到数据文件（文件头和文件数据）和控制文件的 I/O 来挂起数据库。这样，允许数据库在没有 I/O 干预的情况下进行备份。当数据库变成挂起状态时，预先存在的 I/O 操作将完成，新的数据库访问将变成排队状态。

挂起命令挂起数据库。且并不指定一个事例。因此，在 OPS 环境，如果挂起命令输入到系统中，则内部锁定机制将在事例中传播停止请求，因此，在给出的簇中停顿所有激活的事例。然而，在用户挂起另一个事例时，不要启动一个新的事例，因为新的事例不被挂起。

使用 ALTER SYSTEM RESUME 语句来继续运行一般的数据库操作。注意，用户可以在不同的事例中指定 SUSPEND 和 RESUME。例如，假如事例 1、2 和 3 在运行，用户在事例 1 中发出 ALTER SYSTEM SUSPEND 语句，然后用户可以在事例 1、2 或 3 中发出 RESUME，效果相同。

挂起/继续特点征允许用户映射磁盘或文件，然后再分割映射的系统中是有用的。提供一个备选的恢复和存储解决方案。如果用户使用在写操作时不能分割映射盘和现存数据库的系统时，用户可以使用挂起/继续特点来实现分割。关于使用挂起/继续特征来备份数据库的细节，参阅“Oracle8I Backup and Recovery Guide”。

挂起/继续特征不是一般的关闭操作的便利替代物，因为挂起的数据库拷贝可以包含未授权的升级。

**警告** 不要使用 ALTER SYSTEM 语句作为在热备份下放置的替代。用 ALTER TABLESPACE BEGIN BACKUP 语句进行数据库的挂起操作。

下面的语句演示 ALTER SYSTEM SUSPEND/RESUME 的使用。查询 V\$INSTANCE 视图以确定数据库状态。

```
SQL> ALTER SYSTEM SUSPEND;
System altered
SQL> SELECT database_status FROM v$instance;
DATABASE_STATUS
-----
SUSPENDED

SQL> ALTER SYSTEM RESUME;
System altered
SQL> SELECT database_status FROM v$instance;
DATABASE_STATUS
-----
ACTIVE
```

### 3.5 使用初始化参数文件

下面的几节包括怎样使用初始化参数文件的信息：

- 初始化参数文件的例子
- 初始化参数文件的数量
- 在分布式环境的初始化参数文件的位置

要启动一个事例，Oracle 必须读初始化参数文件（initialization parameter file）。它是一个包含有配置参数列表的文本文件。在很多情况下，虽然不总是这样，这个文件被命名为 INIT.ORA 或 INITsid.ORA，其中 sid 是操作系统指定的。

**注意** 如果用户正在使用 Oracle 企业管理器，参考“Oracle Enterprise Manager Administrator’s Guide”以获得关于使用作为初始化参数文件的可供选择方案的存储配置的信息。

用户可以用基本的文本编辑器来编辑参数文件的参数值；然而，编辑方法随操作系统而变。Oracle 在文件中处理国家语言支持（NLS）参数定义的串文字，就象它们在数据库的字符集中。

**参见** 关于指定用户安装的初始化参数文件的更多信息，参阅用户的指定操作系统的

Oracle 文档。

每一个初始化参数的描述在 “Oracle8I Reference” 中。

### 3.5.1 初始化参数文件的例子

参数文件的例子 (INIT.ORA 或 INITsid.ORA) 包括在 Oracle 发行集中。这个例子文件的参数对于 Oracle 数据库的初始化安装是足够的。在用户的系统运转后, 用户对 Oracle 有了一些经验, 用户将会希望改变其中的某些参数。

参见 关于用初始化参数文件来优化数据库性能的更多信息, 参阅 “Oracle8I Designing and Tuning for Performance”。

### 3.5.2 初始化参数文件的数量

每一个 Oracle 数据库有至少一个初始化参数文件来对应该数据库。这种方法, 特定数据库参数, 例如给定文件的 DB\_NAME 和 CONTROL\_FILES 总是属于特定的数据库。一个数据库也可能有多个初始化参数文件。例如, 用户对于一个数据库可以有几个不同参数文件, 这样可以在不同状况下优化数据库的性能。

### 3.5.3 在分布式环境的初始化参数文件的位置

用户用来访问数据库的客户端必须可以读取数据库的初始化参数文件来启动数据库事例。因此, 总要在运行客户端的计算机上存储数据库的参数文件。

在非分布式处理的安装中, 相同的计算机运行 Oracle 和它的客户端。这台计算机已经在它的某一个磁盘驱动器上安装了参数文件。在分布式处理的安装中, 与之相反, 本地客户端工作站可以管理存储于远程计算机的数据库。在这种配置中, 本地的客户端计算机必须分别存储相应数据库的参数文件的一份拷贝。

参见 关于在分布式环境中使用 Oracle 管理的更多信息, 参阅 “Oracle8i Distributed Database Systems”。



## 第二部分 Oracle 服务器配置

---

第二部分描述组成 Oracle 数据库服务器结构，并支持其运转的 Oracle 服务器进程和控制文件。它包括如下几章：

- 第 4 章：管理 Oracle 的进程
- 第 5 章：管理控制文件
- 第 6 章：管理联机重做日志
- 第 7 章：管理归档重做日志
- 第 8 章：管理作业队列

# 4

## 管理 Oracle 进程

本章描述怎样管理 Oracle 事件的进程，包括如下专题：

- 服务器进程
- 配置多线程服务器的 Oracle
- 跟踪 Oracle 后台进程
- 管理并行查询选项的进程
- 管理外部过程的进程
- 终止会话

### 4.1 服务器进程

Oracle 创建服务器进程来处理连接到事例的用户过程的请求。一个服务器进程可以是一个独占的服务器进程(dedicated server process)，其中，一个服务器进程只可以服务一个用户过程；也可以是共享的服务器进程，它的服务器进程可以服务多个用户过程。共享的服务器进程(Shared server processes)是 Oracle 多线程服务器(multi-threaded server(MTS))结构的一部分。

参见 关于服务器进程的概念信息，参阅“Oracle8i Concepts”。

#### 4.1.1 独占的服务器进程

图 4-1 显示了独占的服务器进程是怎样工作的。

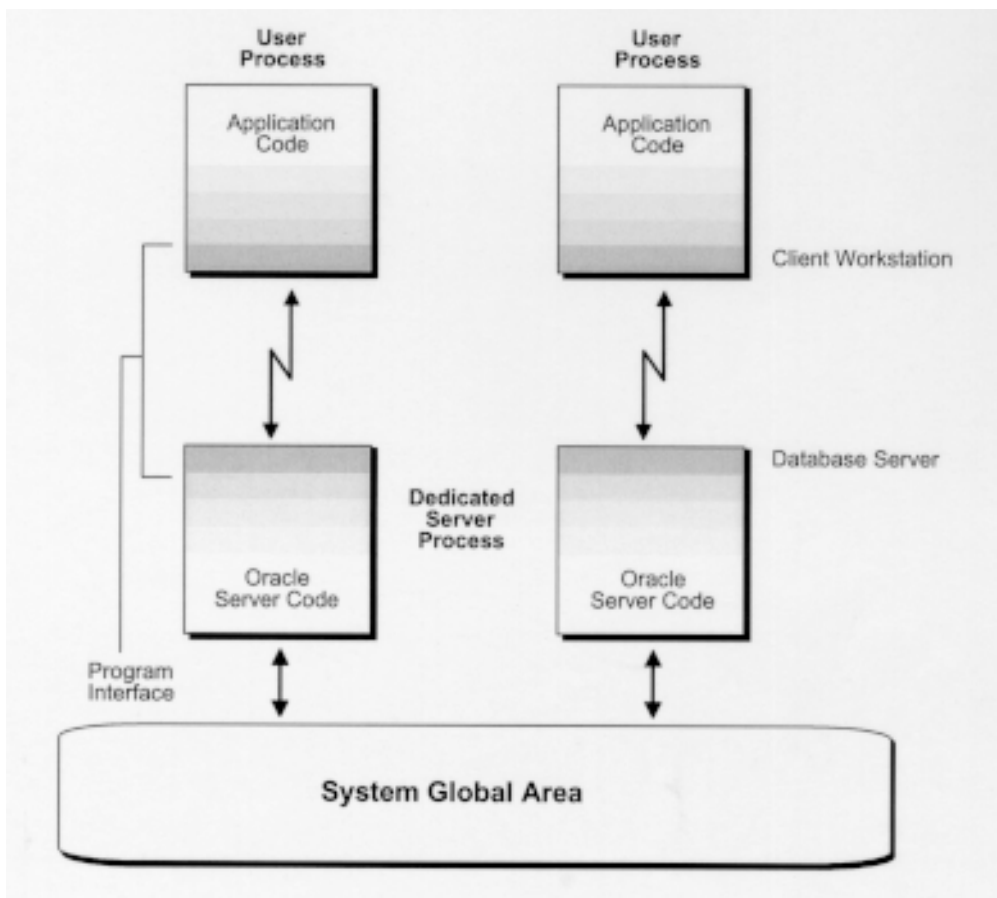


图 4-1 Oracle 独占的服务器进程

一般来说，最好通过使用一个共享服务器进程的调度器（dispatcher）来连接，这样做可以更高效，因为它使得运行事例所要求的进程数量少一些。然而，在下面的情况下，使用者和系统管理员应该明确地用独占的服务器进程来连接事例：

- 要递交批处理作业（例如，当一个作业允许少量或没有空闲时间用于服务器进程时）
- 要使用企业管理器在一个数据库上来启动、关闭、或执行介质恢复时
- 要使用恢复管理器来备份、恢复或重新恢复数据库时

在服务器配置成 MTS 时要请求独占的服务器连接，使用者必须用独占服务的 NET8 网络服务名（net service name）来连接，即用独占服务器来配置。特别是，网络服务名的值应该在连接描述符中包括 **SERVER=DEDICATED** 语句。

参见 关于 Net8 网络服务名的完整的描述，参阅 “Net8 Administrator’s Guide” 和用户特定操作系统的 Oracle 文档。

#### 4.1.2 多线程服务器进程

现在考虑一个订单输入系统怎样进入独占服务器进程。以职员身份下单的顾客将订单输入数据库。对于大多数事务，职员在电话中向顾客讲话，被职员用户进程独占的服务器进

程变得空闲。在大多数的会话期间，不需要服务器进程。由于空闲服务器进程占用系统资源，系统在其它职员输入订单时变得较缓慢。

多线程服务器（multi-threaded server）结构排除了每个连接对独占服务器进程的需求。（见图 4-2）。

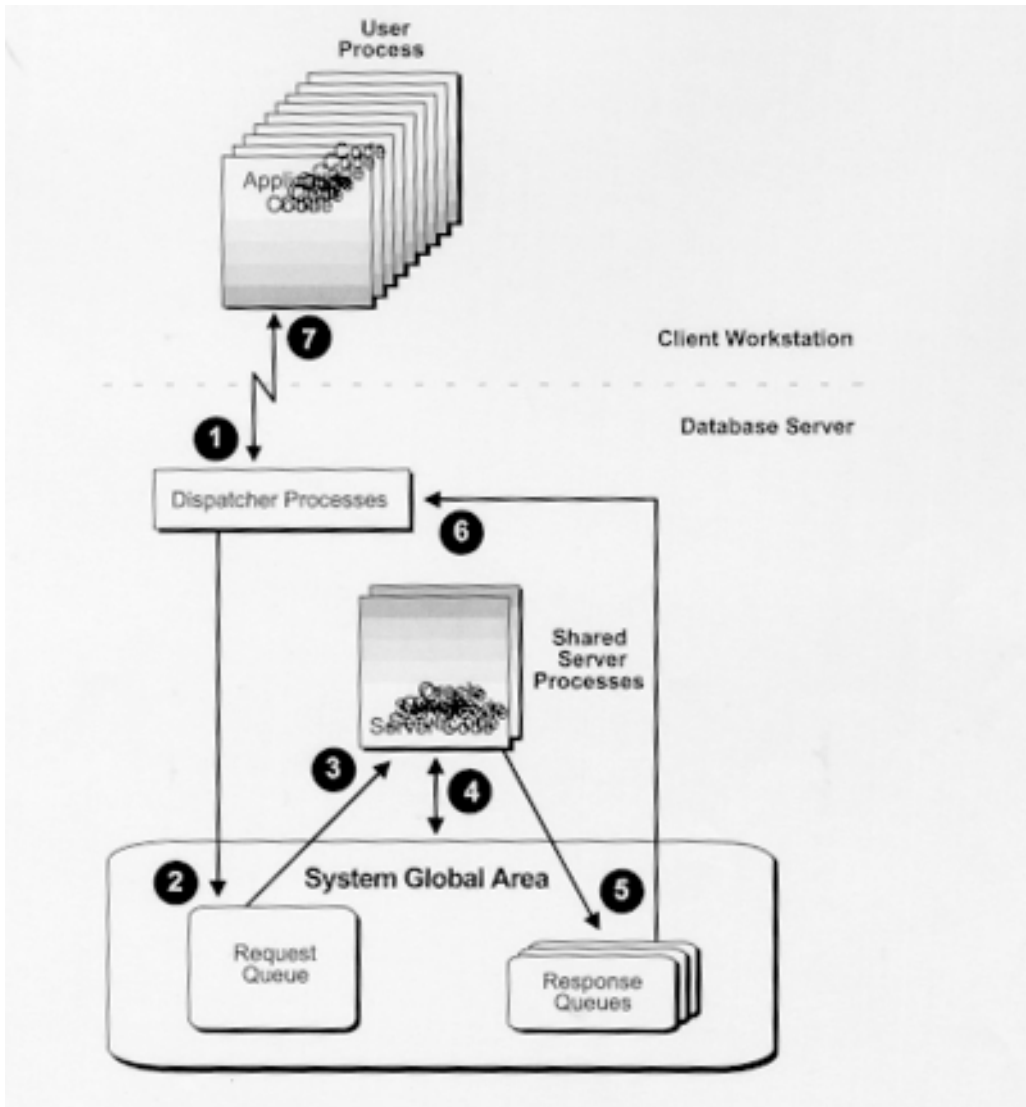


图 4-2 Oracle 多线程服务器进程

在多线程服务器（MTS）配置中，客户端用户进程连接调度器。调度器可以并发地支持多个客户端连接。每个客户端连接绑定在一个虚拟线路上。一个虚拟线路是一片共享的内存，被调度器用来进行客户端数据库连接的请求和应答。在请求到达时，调度器将虚拟线路放在一个公共的队列上。

一个空闲的共享服务器在公共队列中拾取那个虚拟线路，对请求进行服务，并试图在

公共队列中检索另一个虚拟线路之前释放虚拟线路。这个处理使得一个小的服务器进程池对大量的客户端进行服务。MTS 结构超过独占服务器模型的一个显著的优点是减少了系统的资源，使得其支持用户的数量增长。

多线程服务器结构要求 Net8。指向多线程服务器的用户进程必须通过 Net8 来连接，即使它们作为 Oracle 事例同在一台计算机上。

要配置用户的 MTS 系统，有很多的事情需要完成。这将在下一节进行讨论。

参见 要学习 MTS 的更多的内容，包括附加的特点如连接统筹，参阅 “Net8 Administrator’s Guide”。

## 4.2 配置多线程服务器的 Oracle

通过设置数据库的初始化参数来激活 MTS，同时要求激活 Net8 接收器。本节讨论初始化参数的设置和更改。欲了解与 Net8 相关的细节，请参阅 “Net8 Administrator’s Guide”。

### 4.2.1 MTS 的初始化参数

控制 MTS 的初始化参数是：

| 参数                                | 描述  |
|-----------------------------------|---|
| 必需的：                              |   |
| MTS_DISPATCHERS                   | 在多线程服务器结构中配置调度器进程   |
| 可选的：如果用户不指定这些参数，Oracle 会使用合适的缺省值。 |   |
| MTS_MAX_DISPATCHERS               | 确定允许同时运行的调度器数量的最大值  |
| MTS_SERVERS                       | 确定在一个事例启动时用户希望创建的服务器进程的数量   |
| MTS_MAX_SERVERS                   | 确定允许同时运行的共享服务器进程数量的最大值  |
| MTS_CIRCUITS                      | 确定网络会话可提供给入站和出站的虚拟线路的总数量  |
| MTS_SESSIONS                      | 确定允许 MTS 用户会话的总数量。设置这个参数使用户可以为独占服务器保留用户会话                                       |
| 可能要求调整的其它的 MTS 影响的初始化参数           |   |
| LARGE_POOL_SIZE                   | 确定大池分配堆的字节大小。MTS 可能强制缺省值设置得太高，带来性能问题或数据库不能启动。参考 “Oracle8i Reference”，以获得更进一步的细节 |
| SESSIONS                          | 确定系统可创建的会话的最大数量。可能需要为 MTS 而调整。参阅 “Oracle8i Reference”，以获得更进一步的细节                |

参见 关于这些参数的设置和缺省的详细描述，参阅 “Net8 Administrator’s Guide” 和 “Oracle8i Reference”。

#### 4.2.2 MTS\_DISPATCHERS: 设置最初的调度器数量

在事例启动时启动的调度器进程的数量是由初始化参数 `MTS_DISPATCHERS` 来控制的。至少有一个调度器是为参数中确定的每一个通讯协议来创建的。用户可以多次确定在初始化文件中的 `MTS_DISPATCHERS` 参数，但是它们必须连在一起。在内部，Oracle 会分配一个 `INDEX` 值给每一个 `MTS_DISPATCHERS` 参数。这样用户就可以在后来专门的 `ALTER SYSTEM` 语句中访问 `MTS_DISPATCHERS` 参数了。

每个事例的调度器的最佳数量取决于数据库所希望的性能。主操作系统限制连接到每个进程（与操作系统相关）的数量和每个网络协议所要求的连接的数量。事例必须可以提供和数据库系统上的并发用户一样多的连接。在事例启动后，如果需要，用户可以启动更多的调度器进程。这在第 4.2.4 节“增加和删除调度器进程”中进行讨论。

对于典型系统，在每 1000 个连接有 1 个调度器的比例下工作较好。但是，这个比例只入不舍。例如，如果用户期望在最高峰时有 1500 个连接，那么用户需要配置 2 个调度器。过多的估计是不利的，因为配置太多的调度器会降低性能。以这个比例为指导，根据用户的具体情况进行调整。

下面是几个设置初始化参数 `MTS_DISPATCHERS` 的几个例子：

##### 例 4-1

要强制调度器使用的 IP 地址，输入：

```
MTS_DISPATCHERS="(ADDRESS=(PROTOCOL=TCP)\
(HOST=144.25.16.201))(DISPATCHERS=2)"
```

这会启动两个调度器来收听那个 IP 地址。这个 IP 地址必须是一个对事例所在的主机上有效的 IP 地址，必须是一块可访问到调度器的卡。

##### 例 4-2

要强制调度器的准确地址，象如下这样添加 `PORT`：

```
MTS_DISPATCHERS="(ADDRESS=(PROTOCOL=TCP)(HOST=144.25.16.201)(PORT=5000))"
MTS_DISPATCHERS="(ADDRESS=(PROTOCOL=TCP)(HOST=144.25.16.201)(PORT=5001))"
```

#### 4.2.3 MTS\_SERVERS: 设置共享服务器的初始化数量

`MTS_SERVERS` 参数确定在事例启动时用户希望创建的服务器进程的数量。Oracle 基于请求队列长度动态地调整共享服务器进程的数量。可创建的共享服务器进程的数量值在初始化参数 `MTS_SERVERS` 和 `MTS_MAX_SERVERS` 之间变化。

典型的说，看上去稳定的系统在十个连接用一个共享的服务器这个比例上。对于 `OLP` 应用程序，到服务器的连接率可以更高一些，这是因为请求的速度可以降低，请求的服务器使用率可以降低。在请求速度高的应用程序中，或在服务器请求使用率高的应用程序中，到服务器的连接会变得更低一些。

基于用户的应用程序，可将 `MTS_MAX_SERVERS` 设置为一个合理的值。对于典型的配置，Oracle 为 `MTS_SERVERS` 和 `MTS_MAX_SERVERS` 提供了一个好的缺省值，但是这些设置的优化的值可以根据应用程序而不同。

**注意** 在 Windows NT 上，设置 `MTS_MAX_SERVERS` 为一个高值时要小心。每一个服务器

在公共进程中是一个线程。

MTS\_MAX\_SERVERS 是一个静态的初始化参数，所以用户不能在数据库没有关闭的情况下改变它。然而，MTS\_SERVERS 是一个动态的初始化参数，可以用 ALTER SYSTEM 语句来更改。

#### 4.2.4 修改调度器进程和服务器进程

用户可以在事例运行时动态地更换 MTS\_DISPATCHERS 和 MTS\_SERVERS。如果用户有 ALTER SYSTEM 的权限，就可以使用 ALTER SYSTEM 语句来进行更换。

参见 关于 ALTER SYSTEM 语句的信息，参阅 “Oracle8i Reference”。

##### 改变共享服务器进程的最小值

在启动事例之后，用户可以使用 SQL 语句 ALTER SYSTEM 来改变共享服务器进程的最小值。在有比用户确定的最小极限更多的共享服务器时，Oracle 最后将结束空的服务器。

如果用户将 MTS\_SERVERS 设置为 0，Oracle 将终结所有的当前空闲的服务器，并且直到用户增加了 MTS\_SERVERS 值之后，才可以打开新的服务器。这样，将 MTS\_SERVERS 设置为 0 可以用来有效地禁止多线程服务器。

下面的语句将动态地将共享服务器进程的个数设置为两个：

```
ALTER SYSTEM SET MTS_SERVERS = 2
```

##### 添加和删除调度器进程

用户可以控制在事例中的调度器进程的个数。如果 V\$QUEUE、V\$DISPATCHER 和 V\$DISPATCHER\_RATE 视图显示调度器进程的负载偏高，启动附加的调度器进程来处理用户请求，可能会提高性能。相比之下，如果调度器的负荷偏低，缩减调度器的数量会提高性能。

要更换调度器进程的数量，可使用 SQL 语句 ALTER SYSTEM。

用户可以为现存的 MTS\_DISPATCHERS 值启动新的调度器进程，或是添加新的 MTS\_DISPATCHERS 值。用户可以添加调度器到 MTS\_MAX\_DISPATCHERS 所确定的极限。

下面的语句动态地将 TCP/IP 协议的调度器进程数量更改为 5，并添加 SPX 协议的调度器进程。由于没有 SPX 协议的 MTS\_DISPATCHERS 初始化参数（MTS 调度器参数只是对于 TCP 协议的），所以这个语句实际上添加了一个。

```
ALTER SYSTEM  
SET MTS_DISPATCHERS=  
'(PROTOCOL=TCP)(DISPATCHERS=5) (INDEX=0)',  
'(PROTOCOL=SPX)(DISPATCHERS=2) (INDEX=1)';
```

如果当前少于 5 个 TCP 的调度器进程，Oracle 就创建一个新的进程。如果当前多于 5 个，Oracle 会决定在连接的用户断开连接后取消其中的一些进程。

**注意** 关键词 INDEX 可以用来识别哪一个 MTS\_DISPATCHERS 参数被更改。INDEX 的值由 0 到 n。其中 n 是一个小于 MTS\_DISPATCHER 参数所定义数的值。如果用户的 ALTER SYSTEM 语句确定 INDEX 值等于 n+1，一个新的 MTS\_DISPATCHER 参数就被添加。

要识别分配给 MTS\_DISPATCHERS 参数的索引号，可查询 V\$DISPATCHER 视图中的 CONF\_INDX 值。

关闭指定的调度器进程

关闭指定的调度器进程是可能的。要识别将要关闭的指定调度器进程的名字，可使用 V\$DISPATCHER 动态性能视图。

```
SELECT name, network FROM v$dispatcher;
NAME      NETWORK
-----
D000      (ADDRESS=(PROTOCOL=tcp)(HOST=rbaylis-hpc.us.oracle.com)(PORT=3499))
D001      (ADDRESS=(PROTOCOL=tcp)(HOST=rbaylis-hpc.us.oracle.com)(PORT=3531))
D002      (ADDRESS=(PROTOCOL=tcp)(HOST=rbaylis-hpc.us.oracle.com)(PORT=3532))
```

每一个调度器唯一地由表单名称 Dnnn 确定。

要关闭调度器 D002，应发出下面的语句：

```
ALTER SYSTEM SHUTDOWN IMMEDIATE 'D002';
```

关键词 IMMEDIATE 使调度器停止接收新的连接，Oracle 立即中断现有的与调度器的所有连接。在所有的会话清除后，调度器进程结束。如果没有指定 IMMEDIATE，调度器将等待，直到所有的用户断开，并且所有的数据库连接在关闭前结束时为止。

4.2.5 监视 MTS

下面是一些对于获得 MTS 配置信息和监视性能有用的一些视图：

| 视图                      | 描述                                     |
|-------------------------|--|
| V\$DISPATCHER           | 提供调度器进程的信息，包括名称、网络地址、状态、各种应用的统计和索引序列号。 |
| V\$DISPATCHER_RATE      | 提供调度器进程的速度统计。                          |
| V\$QUEUE                | 包含多线程消息队列的信息。                          |
| V\$SHARED_SERVER        | 包含共享服务器进程的信息。                          |
| V\$CIRCUIT              | 包含关于虚拟线路（用户用来通过调度器和服务器来连接数据库）的信息。      |
| V\$SGA                  | 包含关于各种系统全局地区（SGA）组的大小信息。可能在调整 MTS 时有用。 |
| V\$SGASTAT              | 关于 SGA 的详细的统计信息，对调整有用。                 |
| V\$SHARED_POOL_RESERVED | 帮助在共享池中调整保留的池和空间的统计列表。                 |

参见 所有这些视图在 “Oracle8i Reference” 中有详细的描述。

关于监视和调整多线程服务器的详细信息，参阅 “Oracle8i Designing and Tuning for Performance”。

4.3 跟踪 Oracle 后台进程

Oracle 事例可以有多个后台进程。本节提出监视和跟踪这些后台进程的一般方法，包括如下专题：

- 什么是 Oracle 后台进程
- 监视 Oracle 事例的进程
- 踪迹文件、报警日志、和后台进程

参见 关于后台进程的更多的细节描述, 参阅 “Oracle8i Concepts”。

#### 4.3.1 什么是 Oracle 后台进程

简单来说, 如下就是 Oracle 后台进程:

##### ■ 数据库书写器 (Database writer,缩写 DBWn)

数据库书写器将修改过的块从数据库高速缓冲存储器中写入数据文件。虽然一个数据库书写器进程 (DBW0) 对于大多数系统已经足够了, 但是用户可以配置附加的进程 (DBW1 到 DBW9) 来提高有大量数据修改的系统写入性能。初始化参数 DB\_WRITER\_PROCESSES 确定 DBWn 进程的个数。

##### ■ 日志书写器 (Log Writer,缩写 LGWR)

日志书写器进程将重做日志记录写入磁盘。重做日志记录是在系统全局区 (SGA) 的重做日志缓冲器中产生的。并且 LGWR 将重做日志记录按顺序写入联机重做日志文件中。如果数据库有多种重做日志, LGWR 将重做日志记录写入联机重做日志文件组中。参阅第 6 章 “管理联机重做日志”, 以获得日志书写器进程的信息。

##### ■ 检查点 (Checkpoint,缩写 CKPT)

在特定的时间, 所有修改的系统全局区数据库缓冲器由 DBWn 将它们写入数据文件; 这种情况叫作检查点。检查点进程在检查点响应 DBWn 信号, 更新所有的数据库的数据文件和控制文件到大多数最近的检查点。

##### ■ 系统监控器 (System Monitor,缩写 SMON)

系统监控器在一个失败的事例再次启动时执行崩溃恢复。在多事例系统 (使用 Oracle 并行服务器的系统) 中, 事例的 SMON 进程可以为其它已经失败的事例执行事例恢复。SMON 页清除不在使用的临时段, 恢复在崩溃期间忽略的事务和由于文件读取或掉线错误引起的事例恢复。这些事务最终由 SMON 在表空间或文件重新联机时恢复。

SMON 也在数据库字典管理表空间中合并自由区域, 产生连续的自由空间, 使之容易分配。(参阅第 9 章的 9.3.3 节 “合并字典管理表空间中的自由空间”)。

##### ■ 进程监视器 (Process Monitor,缩写 PMON)

进程监视器在用户进程失败时执行进程恢复。PMON 响应清除高速缓冲存储器和释放进程正在使用的资源。PMON 也检查调度器 (见下面) 和服务进程, 如果它们已经失败则重新启动。关于 PMON 的信息, 参阅 “Oracle8i Concepts”。

##### ■ 存档器 (Archiver,缩写 ARCn)

一个或多个存档器进程在联机重做日志满了或日志切换发生时, 将它们拷贝到归档储存在中。存档器进程是第 5 章 “管理存档日志文件” 的专题。

##### ■ 恢复器 (Recoverer,缩写 RECO)

恢复器用来解决在分布数据库中由于网络或系统失败而挂起的分布事务。在一定的时间间隔里, 本地的 RECO 视图连接远程的数据库, 并自动地完成提交或任何挂起的分布事务的本地端口的回滚。关于这个进程以及怎样启动这个进程的信息, 参阅 “Oracle8i Distributed



Database Systems”。

■ 调度器 (Dispatcher,缩写 Dnnn)

调度器是后台进程的选项，只有在多线程服务器（MTS）配置使用时才给出。MTS 在前面 4.2 节“配置多线程服务器的 Oracle”中讨论。

■ 锁 (Lock,缩写 LCK0)

在 Oracle 并行服务器中，一个锁进程提供内部事务的锁定。关于这个后台进程的信息参阅“Oracle8i Parallel Server Setup and Configuration Guide”、“Oracle8i Parallel Server Administration,Deployment,and Performance”、和“Oracle8i Parallel Server Concepts”。

■ 作业队列 (Job Queue,缩写 SNPn)

在分布的数据库配置中，多至 36 个作业队列进程可以自动地刷新表的快照。它们周期性地唤醒，按计划刷新快照。关于创建和刷新快照的信息，可参阅“Oracle8i Replication”、“Oracle8i Replication Management API Reference”、和“Getting Started with Replication Manager”。

这些进程的其它功能是将这些排队的信息传播到其它数据库的队列中。参阅“Oracle8i Application Developer's Guide-Advanced Queuing”，以获得关于传播排队消息的信息。

这些进程也运行由 DBMS\_JOBS 包创建的作业请求。这是第 8 章“管理作业队列”的专题。

与大多数后台进程不同，如果 SNP 进程失败，并不产生事例失败。

■ 队列监视器 (Queue Monitor,缩写 QMNn)

队列监视器进程是针对 Oracle 高级队列的可选的后台进程。用户可以配置多至 10 个的监视器进程。像 SNPn 进程一样，如果这些进程失败，它们并不引起事例的失败。AQ\_TM\_PROCESSES 初始化参数确定在事例启动时的队列监视器的创建。关于高级队列的信息，参阅“Oracle8i Application Developer's Guide-Advanced Queuing”。

4.3.2 监视 Oracle 事例的进程

本节列出了一些用户可以监视 Oracle 事例的一些视图。这些视图在它们的范围内是常规的。也有一些其它的更特殊的视图，在本书中描述进程的章节中讨论。也给出了监视锁定状态的视图和脚本。

参见 所有这些视图在“Oracle8i Reference”中有详细描述。

“Oracle8i Designing and Tuning for Performance”中提供了可能通过监视这些视图可能暴露的性能问题和冲突问题的解决方法。

进程和会话的视图

这些视图提供了特定信息的进程和会话。

| 视图                 | 描述                             |
|--------------------|--------------------------------|
| V\$PROCESS         | 包含关于当前激活进程的信息                  |
| V\$SESSION         | 列出每个当前会话的会话信息                  |
| V\$SESS_IO         | 包含每个用户会话的 I/O 统计               |
| V\$SESSION_LONGOPS | 这个视图显示运行超过 6 秒（绝对时间）的各种操作的状态。这 |

|                   |   |
|-------------------|---|
|                   | 些操作当前包括很多备份和恢复功能、统计汇总、询问执行和每个 Oracle 版本添加的更多的操作             |
| V\$SYSSTAT        | 包含会话统计  |
| V\$RESOURCE_LIMIT | 提供关于一些系统资源应用的当前和最大的全局资源信息                                   |
| V\$SQLAREA        | 包含关于共享的 SQL 地区的统计和包含每个 SQL 串的一行。也提供关于内存中、分列的、准备运行的 SQL 语句统计 |
| V\$LATCH          | 包含没有父母的锁存器的统计和有父母的锁存器的概要统计                                  |

### 监视锁定

UTLLOCKT.SQL 脚本以树状结构方式显示一个简单的字符锁等待图。使用一个特别的询问工具，例如象 SQL\*Plus，该脚本打印系统中等待锁定和响应块锁定的会话。脚本文件的位址取决于操作系统。参阅用户特定操作系统的 Oracle 文档。第二个脚本，CATBLOCK.SQL，创建 UTLLOCKT.SQL 需要的锁视图，所有用户必须在运行 UTLLOCKT.SQL 之前运行它。

下面的视图可以用来监视锁定。

| 视图      | 描述                                 |
|---------|------------------------------------|
| V\$LOCK | 列出当前由 Oracle 服务器控制的锁，以及锁或锁存器未完成的请求 |

### 4.3.3 踪迹文件、报警日志和后台进程

每一个服务器和后台进程可以写到一个相关的踪迹文件。当进程检测到一个内部错误时，将关于错误的信息转储到它的踪迹文件。一些写入踪迹文件的信息是给系统管理员的，而其它的信息是给 Oracle 全球支持系统的。踪迹文件信息也用来调整应用程序和事例。

Alter log 是一个特殊的踪迹文件。数据库的报警日志是消息和错误的按年月排列的日志，它包含下列信息：

- 所有发生的内部错误（ORA-600）、块讹误信息（ORA-1578）、和死锁信息（ORA-60）
- 管理操作，如 CREATE/ALTER/DROP/DROP DATABASE/TABLESPACE/ROLLBACK SEGMENTS SQL 语句和 STARTUP、SHUTDOWN、和 ARCHIVE LOG
- 关于共享服务器和调度器进程功能相关的几个消息和错误
- 在自动刷新快照期间发生的错误
- 数据库和事例启动时的初始化参数的值。

Oracle 使用报警日志来保存这些特殊操作的日志，作为显示这些操作者控制台信息的备选（虽然很多系统在控制台显示信息）。如果操作成功，与时间戳一起，在报警日志中写入“completed” (完成)信息。

### 使用踪迹文件

用户可以周期性地检查事例的报警日志和它的踪迹文件来查看后台进程是否发生了错误。例如，当日志书写器进程（LGWR）不能向组中的成员书写时，表示问题本质的错误消息已经被写入 LGWR 跟踪进程和数据库报警日志中。如果用户看到这样的消息，则介质问题或 I/O 问题已经发生了，应该立即进行纠正。

除了其它一些重要的统计以外，Oracle 也向报警日志中写入初始化参数的值。例如，当用户以正常方式或立即方式（不能是退出方式）关闭事例时，Oracle 向报警日志中写入连接到事例的并发会话的最高数量，即使事例是启动状态。用户可以使用这个数量来查看用户是否需要更新用户的 Oracle 会话许可。

#### 确定踪迹文件位址

所有的后台进程和报警日志的踪迹文件所写入的目的地由初始化参数 `BACKGROUND_DUMP_DEST` 来确定。所有服务器进程的踪迹文件所写入的目的地由初始化参数 `USER_DUMP_DEST` 来确定。踪迹文件的名字由操作系统指定，但通常包括写文件的进程的名字（例如 `LGWR` 和 `RECO`）

#### 控制踪迹文件的大小

用户可以使用初始化参数 `MAX_DUMP_FILE_SIZE`，来控制所有踪迹文件（包括报警日志）大小的最大值。这个极限作为操作系统块的数量来设置。要控制报警日志的大小，用户必须手工的删除用户不再使用的文件；否则，Oracle 继续向这个文件中添加。用户在事例运行时可以安全地删除报警日志，虽然用户可能希望首先制作一个存档的备份。

#### 在 Oracle 写踪迹文件时进行控制

后台进程总是在合适时写踪迹文件。在 `LGWR` 后台进程的情况下，可能通过初始化参数，来控制产生的跟踪信息的数量和类型。这在 7.9 节“控制由存档进程产生的跟踪输出”中描述。其它的后台进程没有这种灵活性。

只有当初始化参数 `SQL_TRACE` 被设置为 `TRUE` 时，踪迹文件才是以服务器进程的名义来书写的（除了在错误期间被写入以外）。通过使用带有 `SET SQL_TRACE` 参数的 SQL 语句 `ALTER SESSION`，不管当前的 `SQL_TRACE` 的值，每一个会话可以启用或禁止代表相关服务器进程的日志跟踪。

```
ALTER SESSION SET SQL_TRACE TRUE;
```

对于多线程的服务器，使用调度器的每一个会话指向一个共享的服务器进程，只有在会话已经启用跟踪时（或错误发生时），跟踪信息才写入服务器踪迹文件。因此，要跟踪连接到正在使用调度器的特定会话的跟踪，用户可能必须浏览一些共享的服务器踪迹文件。因为服务器的 SQL 跟踪设备可以产生显著的系统开销，所以只在收集统计时启用这个特征。

参见 关于踪迹文件的名称信息，参阅用户的特定操作系统的 Oracle 文档。

关于控制书写踪迹文件的初始化参数的信息，参阅“Oracle8i Reference”。

## 4.4. 管理并行查询选项的进程

本节描述借助并行查询选项，Oracle 怎样执行并行处理。在这个配置下，Oracle 可以在多查询服务器进程中把处理工作划分为一定的 SQL 语句类型。本节包括下面的专题：

- 管理查询服务器
- 查询服务器进程的数量变化

参见 关于并行查询选项的更多信息，参阅 “Oracle8i Designing and Tuning for Performance” 和 “Oracle 8I Concepts”。

#### 4.4.1 管理查询服务器

当用户启动事例时，Oracle 数据库服务器创建一个查询服务器进程池以用于查询协调。通过初始化参数 `PARALLEL_MIN_SERVERS`，确定在事例启动时 Oracle 创建的查询服务器进程的数量。

查询服务器进程在整个运行状态保持与语句的联系。在语句完全处理完时，它的查询服务器进程可用于处理其它语句。查询协调器进程把结果数据返回给发出语句的用户进程。

#### 4.4.2 查询服务器进程的数量变化

如果由用户事例并行处理的 SQL 语句的容量发生激烈变化，Oracle 数据库服务器自动改变在池中的查询服务器进程的数量来适应这个容量。

如果这个容量增加，那么 Oracle 自动地创建附加的查询服务器进程来处理输入语句。用户事例的查询服务器进程的最大数量由初始化参数 `PARALLEL_MAX_SERVERS` 来确定。

如果这个容量随后又减少了，如果查询服务器进程在由初始化参数 `PARALLEL_SERVER_IDLE_TIME` 确定的时间段里空闲下来，Oracle 就结束它。不管查询服务器进程空闲了多长时间，Oracle 不能将池的大小减少到 `PARALLEL_MIN_SERVERS` 的值以下。

如果池中的所有查询服务器进程被占用，并且查询服务器的最大数量已经启动，查询协调器将随后处理语句。

参见 关于监视查询服务器的事例池和确定初始化参数的合适值的更多信息，参阅 “Oracle8i Designing and Tuning for Performance”。

### 4.5 管理外部应用程序的进程

用户可能有共享的 C 函数库，希望能在 Oracle 数据库中调用。本节描述怎样建立调用这些外部进程的环境。

**注意** 虽然没有要求，但提醒用户在安装期间执行这些任务。

数据库管理员将合适库的运行权限给予应用程序开发者。他们依次创建外部进程，将特定外部进程的运行权限授予其他用户。

#### 4.5.1 为调用外部例程设置环境

按下面步骤为调用外部例程设置环境：

1. 编辑 `tnsnames.ora` 文件，添加一个使用户可以连接到收听进程的入口（即 `EXTPROC` 进程）。
2. 编辑 `listener.ora` 文件，添加“外部过程收听器”入口。
3. 启动独立的收听器进程专门地控制外部进程。
4. 由收听器大量产生的 `EXTPROC` 进程继承收听器的操作系统权限，所以 Oracle 强烈

地提醒用户限制各自的收听器权限。进程应该不允许读取或写入数据库文件，或写向 Oracle 服务器。同样，独立的收听器进程的所有者也不应该是 Oracle（Oracle 是缺省的服务器运行和数据库文件的所有者）。

5. 如果还没有安装，将可运行的 EXTPROC 放在 \$ORACLE\_HOME/bin 中。注意外部库（DLL 文件）必须静态地连接。换句话说，它不能从其它的外部库（DLL 文件）中引用任何外部符号。这些符号不可分辨，且可能造成用户外部过程失败。

#### 4.5.2 Tnsnames.ora 中的入口示例

下面是在 tnsnames.ora 中外部过程收听器的入口例子：

```
extproc_connection_data = (DISCRIPTION=
                           (ADDRESS = (PROTOCOL=IPC)
                               (KEY=extproc_key)
                           )
                           (CONNECT_DATA = (SID = extproc_agent)
                           )
                           )
```

在这个例子中，所有外部过程的 callout、入口名字 EXTPROC\_CONNECTION\_DATA 都不可更改。必须严格按这里呈现的样子输入。用户确定的键码，在这里是 EXTPROC\_KEY，必须与用户在 listener.ora 中确定的键码 KEY 相匹配。另外，用户确定的 SID 名，在这里是 EXTPROC\_AGENT，必须与 listener.ora 文件中的 SID\_NAME 相匹配。

#### 4.5.3 Listener.ora 中的入口示例

下面是在 listener.ora 中的外部过程入口示例：

```
EXTERNAL_PROCEDURE_LISTENER =

(ADDRESS_LIST =
  (ADDRESS = (PROTOCOL=ipc)
              (KEY=extproc_key)
  )
)
...
SID_LIST_EXTERNAL_PROCEDURE_LISTENER =
(SID_LIST =
  (SID_DESC = (SID_NAME=extproc_agent)
              (ORACLE_HOME=/oracle)
              (PROGRAM=extproc)
  )
)
```

在这个例子中，PROGRAM 必须是 EXTPROC，并且不能改变。必须象例子中的样子进行输入。SID\_NAME 必须与 tnsnames.ora 文件中的 SID 名字相匹配。ORACLE\_HOME 必须设置到用户 Oracle 软件安装的目录。EXTPROC 可运行必须放在 \$ORACLE\_HOME/bin 中。

参见 关于外部进程的更多的信息，参阅“PL/SQL User's Guide and Reference”。

关于 tnsnames.ora 和 listener.ora 文件的更多信息，参阅 “Net8 Administrator's Guide”。

## 4.6 结束会话

在某些场合，用户可能希望中断当前的用户会话。例如，用户可能希望运行一个管理操作，需要中断所有的非管理会话。

本节描述中断会话的各个方面，包括下面的专题：

- 识别要中断的是哪个会话
- 中断一个激活的会话
- 中断一个非激活的会话

当会话中断时，会话的事务回滚，由会话控制的资源（例如锁定和内存区）立即释放，可以被其它会话使用。

要中断当前的会话，可使用 SQL 语句 ALTER SYSTEM KILL SESSION。

下面的语句中断一个 SID 为 7，序列号为 15 的一个会话：

```
ALTER SYSTEM KILL SESSION '7,15';
```

### 4.6.1 识别要中断的是哪个会话

要识别中断哪个会话，应确定会话的索引号和序列号。要识别会话的索引（SID）和序列号，查询 V\$SESSION 动态性能视图。

下面的查询识别用户 JWARD 的所有会话：

```
SELECT sid, Serial#  
FROM v$session  
WHERE username = 'JWARD';
```

| SID | SERIAL# | STATUS   |
|-----|---------|----------|
| 7   | 15      | ACTIVE   |
| 12  | 63      | INACTIVE |

在会话向 Oracle 进行 SQL 调用时，它是 ACTIVE（激活）的。在不向 Oracle 进行 SQL 调用时，是 INACTIVE(非激活)的。

参见 关于会话的状态值的详细描述，参阅 “Oracle8i Reference”。

### 4.6.2 中断一个激活的会话

如果在用户的会话中断时向 Oracle 进行 SQL 调用（激活状态），事务回滚，用户会立即收到如下消息：

```
ORA-0028: your session has been killed
```

如果在收到 ORA-0028 消息之后，一个用户在重新连接到数据库之前发出附加语句，Oracle 返回如下的消息：

```
ORA-010012: not logged on
```

如果一个激活的会话不能被打断（正在执行网络 I/O 或回滚事务），会话就直到操作完成后才能中断。在这种情况下，会话在中断之前一直控制资源。另外，发出 ALTER SYSTEM 语句来中断会话的会话将等待 60 秒来等待会话的中断；如果不能被打断的操作持续了一分钟，ALTER SYSTEM 语句的发出者将收到一个指示会话已经“标志”为中断的一条消息。标志为中断的会话在 V\$SESSION 和除了 PSEUDO 以外的服务器中表示，其状态为 KILLED。

#### 4.6.3 中断一个非激活的会话

如果在中断时会话没有进行 Oracle 的 SQL 调用（非激活），则不会立即收到 ORA-0028 消息。直到用户后来视图使用中断的会话时，才收到这条消息。

当一个非激活的会话中断时，V\$SESSION 视图中的 STATUS 是 KILLED。中断会话的行在用户视图再次使用会话时被删除，并收到 ORA\_0028 消息。

在下面的例子里，非激活的会话被中断。首先，V\$SESSION 被查询来识别会话的 SID 和 SERIAL#，然后，会话中断。

```
SELECT sid,serial#,status, server
      FROM v$session
      WHERE username = 'JWARD';
```

| SID | SERIAL# | STATUS   | SERVER    |
|-----|---------|----------|-----------|
| 7   | 15      | INACTIVE | DEDICATED |
| 12  | 63      | INACTIVE | DEDICATED |

2 rows selected.

```
ALTER SYSTEM KILL SESSION '7,15';
Statement processed.
```

```
SELECT sid, serial#, status, server
      FROM v$session
      WHERE username = 'JWARD';
```

| SID | SERIAL# | STATUS   | SERVER    |
|-----|---------|----------|-----------|
| 7   | 15      | KILLED   | PSEUDO    |
| 12  | 63      | INACTIVE | DEDICATED |

2 rows selected.

# 5 管理控制文件

本章解释怎样创建和维护用户数据库的控制文件，其中包括如下几个专题：

- 什么是控制文件
- 控制文件指南
- 创建控制文件
- 创建控制文件之后的故障排除
- 删除控制文件

## 5.1 什么是控制文件

每一个 Oracle 数据库都有一个控制文件（control file）。控制文件记录数据库的物理结构，其中包括：

- 数据库名称
- 相关的数据库和联机重做日志文件的名称和位置
- 数据库创建的时间戳
- 当前日志序列号
- 检查点信息

Oracle 数据库的控制文件和数据库同时创建。在缺省情况下，至少有一个控制文件的拷贝必须在数据库创建期间创建。在一些操作系统上，Oracle 创建多个拷贝。用户应该在数据库创建期间创建两个或更多的控制文件的拷贝。如果用户丢失控制文件或希望改变控制文件中的特殊设置，也可能需要在后来创建控制文件。

## 5.2 控制文件指南

本节描述用户用以管理数据库控制文件的指南，包括如下专题：

- 命名控制文件
- 在不同的磁盘上多工控制文件
- 适当地放置控制文件
- 管理控制文件大小

### 5.2.1 命名控制文件

借助数据库初始化参数文件中的初始化参数 `CONTROL_FILES` 来分配控制文件名称。`CONTROL_FILES` 指定一个或多个控制文件的名称，中间用逗号隔开。通过事例启动过程来识别和打开所有列出的文件。事例在数据库操作期间写并维护所有列出的控制文件。

参见 关于初始化参数 `CONTROL_FILES` 的描述，参阅 “Oracle8i Reference”。



### 5.2.2 在不同磁盘上多工控制文件

每一个 Oracle 数据库应该至少有两个控制文件，并存储于不同的磁盘上。如果由于磁盘失败而导致控制文件损坏，相关的事例必须关闭。一旦磁盘驱动器修复，损坏的磁盘文件可以使用控制文件的原样拷贝来恢复，事例可以重新启动。不要求介质恢复。

下面描述多工控制文件的性能：

- 数据库初始化参数文件中的初始化控制参数 `CONTROL_FILES` 列出的两个或更多的文件名。Oracle 同时写这两个文件。
- 在 `CONTROL_FILES` 参数中列出的第一个文件是数据库操作期间 Oracle 服务器唯一读取的文件。
- 如果在数据库操作期间有任何一个控制文件变得不可用，事例将变得不能执行，并应该退出。

有多个控制文件的唯一缺点是更新控制文件的所有操作（例如添加数据库的数据文件或检查点）耗时稍长一些。然而，这个区别一般不明显（特别是对于支持多重、并行写的操作系统）。而且只使用一个控制文件是不合适的。

**特别注意** Oracle 强烈地提醒用户，数据库至少有在不同磁盘上的两个控制文件。

### 5.2.3 合理放置控制文件

每个控制文件的拷贝应该存到不同的磁盘驱动器上。更进一步来说，如果联机重做日志是多工的，一个控制文件的拷贝应该存到每个存储联机重做日志组的成员的磁盘驱动器上。通过在这些地方存储控制文件，用户把风险降低到最小——在一个磁盘失败中丢失所有的控制文件和联机重做日志组。

### 5.2.4 管理控制文件的大小

控制文件大小的主要决定因数是创建相关数据库的 `CREATE DATABASE` 语句中的 `MAXDATAFILES`、`MAXLOGFILES`、`MAXLOGMEMBERS`、`MAXLOGHISTORY` 和 `MAXINSTANCES` 参数设置的值。增加这些参数的值，就要增加相关数据库的控制文件的大小。

**参见** 控制文件大小的最大值与操作系统相关。参阅用户特定操作系统的 Oracle 文档，以获得更多的信息。

关于 `CREATE DATABASE` 语句的语法，参阅“Oracle8i SQL Reference”。

## 5.3 创建控制文件

本节描述创建控制文件的方法，包括下面的专题：

- 创建初始控制文件
- 创建附加的控制文件拷贝，对控制文件重命名、重定位
- 新的控制文件
- 创建新的控制文件

### 5.3.1 创建初始控制文件

用户通过在数据库创建期间使用的初始化参数文件中的初始化参数 `CONTROL_FILES` 中指定一个或多个控制文件名来创建 Oracle 数据库的初始控制文件。由 `CONTROL_FILES` 指定的文件名应该全部指定。文件名的指定是与操作系统相关的。

在数据库创建时，如果指定名称的文件当前存在，用户必须指定在 `CREATE DATABASE` 语句中的 `CONTROLFILE REUSE` 参数，否则就发生错误。同样，如果旧的控制文件的大小与新的不同，用户不能使用 `REUSE` 选项。控制文件的大小在一些 Oracle 版本之间是不同的。在控制文件中指定的文件数目也不同。配置参数，如 `MAXLOGFILES`、`MAXLOGMEMBERS`、`MAXLOGHISTORY`、`MAXDATAFILES` 和 `MAXINSTANCES` 影响控制文件的大小。

如果用户在数据库创建之前不指定 `CONTROL_FILES` 的文件，Oracle 会创建一个控制文件，并使用缺省的文件名。这缺省的文件名也是与系统相关的。

用户可以随后改变初始化参数 `CONTROL_FILES` 的值，来添加更多的控制文件或改变现有控制文件的名称或地址。

参见 关于确定控制文件的更多信息，参阅用户特定操作系统的 Oracle 文档。

### 5.3.2 创建附加的控制文件拷贝，对控制文件重命名、重定位

用户创建新的控制文件的方法是将现有文件拷贝到新的地址，将文件名添加到控制文件列表。

类似地，用户通过拷贝文件为新的名字或地址来重命名现有文件，并且改变控制文件列表中的文件名。

在下面两种情况，要保证控制文件在程序中不被更改，应在拷贝控制文件之前关闭事例。

要将附加的当前控制文件的拷贝多工或移动，应该：

1. 关闭数据库。
2. 使用操作系统的命令将现存的控制文件拷贝到不同的地址。
3. 编辑数据库初始化文件中的 `CONTROL_FILES` 参数，添加新的控制文件名，或更改现存的控制文件。
4. 重新启动数据库。

### 5.3.3 新的控制文件

用户可以使用 `CREATE CONTROLFILE` 语句来创建数据库的新的控制文件。在下面情况下这是需要的：

- 所有数据库的控制文件已经永久性地破坏了，并且用户没有备份的控制文件。
- 用户希望改变其中一个永久的数据库设置，而设置为 `CREATE DATABASE` 语句中初始确定的。`MAXLOGFILES`、`MAXLOGMEMBERS`、`MAXLOGHISTORY`、`MAXDATAFILES`、和 `MAXINSTANCES` 的。

例如，用户可能需要改变在分布式环境中的其它与数据库名称冲突的数据库名。或者，另一个例子，用户可能需要改变前面提到的起初设置太低的一个参数。

下面的语句创建一个新的 **PROD** 数据库（一个起初使用不同数据库名称的数据库）的控制文件：

```
CREATE CONTROLFILE
  SET DATABASE prod
  LOGFILE GROUP 1 ('logfile1A', 'logfile1B') SIZE 50K,
  GROUP 2 ('logfile2A', 'logfile2B') SIZE 50K
  NORESETLOGS
  DATAFILE 'datafile1' SIZE 3M, 'datafile2' SIZE 5M
MAXLOGFILES 50
MAXLOGMEMBERS 3
MAXDATAFILES 200
MAXINSTANCES 6
ARCHIVELOG;
```

**警告** `CREATE CONTROLFILE` 语句可以潜在地破坏指定的数据文件和联机重做日志文件；忽略文件名可以造成文件中数据的丢失，或者对整个数据库的访问丢失。在使用这些语句时注意警告，并保证遵循下节中介绍的步骤。

**参见** 关于更换全局数据库名和 `DB_DOMAIN` 初始化参数的信息，请参阅“Oracle8i Distributed Database Systems”。

### 5.3.4 创建新的控制文件

本节提供创建新的控制文件的分步骤的指导：

要创建新的控制文件：

1. 列出数据库所有的数据文件和联机重做日志文件。

如果用户遵循数据库备份的建议，用户应该已经拥有了反映数据库当前结构的数据文件和联机重做日志文件的列表。

如果用户没有这个列表，并且用户的控制文件已经损坏，以至于数据库不能打开，那就试一下定位所有构成数据库的数据文件和联机重做日志文件。一旦控制文件创建好，在第五步没有确定的文件不能恢复。此外，如果用户忽略任何一个组成 **SYSTEM** 表空间的文件，用户可能不能恢复数据库。

2. 关闭数据库

如果数据库是打开的，如果可能的话，用通常的优先权来关闭数据库。实在不行时，再用 **IMMEDIATE** 或 **ABORT** 选项。

3. 备份所有数据库的数据文件和联机重做日志文件。

4. 启动一个新的事例，但不要装入和打开数据库。

5. 用 `CREATE CONTROLFILE` 语句来创建新的数据库控制文件。

在创建新的控制文件时，如果用户已经丢失任何一个附加到控制文件的联机重做日志组时，选择 **RESETLOGS** 选项。在这种情况下，用户将需要从重做日志的损失中恢复。（步骤8）。如果用户已经重命名了数据库，用户也必须确定 **RESETLOGS** 选项。否则，选择 **NORESETLOGS** 选项。

6. 在离线的存储设备上存储新控制文件的备份。

#### 7. 编辑数据库的初始化参数文件。

编辑数据库的初始化参数文件，指出 **CONTROL\_FILES** 参数在步骤 5 和步骤 6 中创建的所有控制文件（不包括备份控制文件）。如果用户正在重命名数据库，编辑 **DB\_NAME** 参数来确定新名称。

#### 8. 如果需要，恢复数据库。如果用户没有恢复数据库，跳到步骤 9。

如果用户作为恢复来创建控制文件则恢复数据库。如果新的控制文件用 **NORESETLOGS** 选项（步骤 5）来创建，用户可以完整地恢复数据库，关闭数据库恢复。

如果新的数据库用 **RESETLOGS** 选项来创建，用户必须指定 **USING BACKUP CONTROL FILE**。如果用户已经丢失了联机或存档的重做日志或数据文件，使用恢复这些文件的程序。

#### 9. 打开数据库

使用下面的方法之一来打开数据库：

- 如果用户没有执行恢复，正常地打开数据库。
- 如果用户完全执行，按步骤 8 关闭了数据库恢复，启动数据库。
- 如果用户在创建控制文件时指定 **RESETLOGS**，使用 **ALTER DATABASE** 语句，显示 **RESETLOGS**。

数据库现在是打开，并可供使用了。

参见 参阅 “Oracle8i Backup and Recovery Guide” 以获得关于如下内容的更多信息

- 列出数据库文件
- 备份数据库所有的数据文件和联机重做日志文件
- 恢复联机和存档的重做日志文件
- 关闭数据库恢复

## 5.4 创建控制文件之后的故障排除

在发出 **CREATE CONTROLFILE** 语句之后，用户可能遇到一些普遍的错误。本节描述最一般的控制文件使用错误，包括如下专题：

- 检查丢失或多余的文件
- 在 **CREATE CONTROLFILE** 期间处理错误

### 5.4.1 检查丢失或多余的文件

在创建新的控制文件，并用它打开数据库之后，检查报警日志，看一下 **Oracle** 是否已经检查到在数据字典和控制文件之间的矛盾。例如数据字典包括，但控制文件没有列出的数据文件。

如果数据文件在数据字典中存在，但在新的控制文件中不存在，则 **Oracle** 在控制文件中，在名字 **MISSINGnnnn**（其中 **nnnn** 是十进制文件编号）下，创建占位符项。**MISSINGnnnn** 是在控制文件中作为脱机和请求介质恢复的标志。

只有在下面的两种情况中，对响应于 **MISSINGnnnn** 的实际文件可以通过指向它的 **MISSINGnnnn** 的重命名来变得可访问：

**情况 1** 新的控制文件使用带有 NORESETLOGS 选项的 CREATE CONTROLFILE 语句来创建，这样允许数据库在没有使用 RESETLOGS 选项时被打开。这仅在所有联机重做日志可用时才有可能。

**情况 2** 有必要使用 CREATE CONTROLFILE 语句的 RESETLOGS 选项，这样使用 RESETLOGS 选项强制数据库打开，但是响应 MISSINGnnnn 的实际的数据库是只读或正常脱机的。

如果，另一方面，使用 RESETLOGS 选项来打开数据库是必要的，并且 MISSINGnnnn 对应于非只读或非正常脱机的数据文件，那么重命名操作不能用来使数据文件可访问（因为数据文件请求介质恢复，而介质恢复被 RESETLOGS 的结果排除在外）。在这种情况下，包含数据文件的表空间必须删除。

相反，如果包含在控制文件中的数据文件没有在数据字典中给出，Oracle 从新的控制文件中删除对它的引用。在这两种情况下，Oracle 在报警日志中包含一个解释消息，来让用户了解找到了什么。

#### 5.4.2 在 CREATE CONTROLFILE 期间处理错误

如果在用户创建新控制文件之后试图装入并打开数据库时，Oracle 发给用户一个错误（通常是 ORA-01173、ORA-01176、ORA-01177、ORA-01215 或 ORA-01216），最可能的原因是用户在 CREATE CONTROLFILE 语句中忽略了一个文件或包含了一个应该没有列出的文件。在这种情况下，用户应该恢复在步骤 3 中的备份，并使用正确的文件名，从步骤 4 开始重复操作过程。

### 5.5 删除控制文件

用户可以从数据库中删除控制文件。例如，如果控制文件的位置不合适时，用户可能希望这样做。记住，在任何时候，数据库必须至少有两个控制文件。

1. 关闭数据库。
2. 编辑数据库初始化参数文件中的 CONTROL\_FILES 参数，删除旧的控制文件名。
3. 重新启动数据库。

**警告** 这个操作不能物理上从磁盘上删除不希望的控制文件。用户从数据库中删除控制文件之后，使用操作系统的命令来删除不需要的文件。

# 6 管理联机重做日志

本章解释怎样管理联机重做日志，包括如下专题：

- 什么是联机重做日志
- 规划联机重做日志
- 创建联机重做日志组和成员
- 联机重做日志的重命名和重定位
- 删除联机重做日志组和成员
- 强制日志切换
- 校验重做日志文件中的块
- 清除联机重做日志文件
- 列出关于联机重做日志的信息

参见 关于在使用 Oracle 并行服务器时，管理事例的联机重做日志的更多信息，参阅“Oracle8i Parallel Server Administration, Deployment, and Performance”。

要学习检查点和重做日志怎样影响事例恢复，请参阅“Oracle8i Designing and Tuning for Performance”。

## 6.1 什么是联机重做日志

操作恢复的最至关重要结构是联机重做日志，它有两个或多个预先分配的文件在更改发生时存储对数据库进行更改。万一在事例失败时，每一个 Oracle 数据库的事例都有一个与之相关的联机重做日志来保护数据库。

**注意** Oracle 不提示用户备份联机重做日志。

### 6.1.1 重做线程

每个数据库事例有其自己的联机重做日志组 (redo log groups)。这些联机重做日志组，不管是多工的或不是多工的，都叫做事例的联机重做线程 (thread)。在典型的配置中，只有一个数据库事例访问 Oracle 数据库，所以只给出一个线程。然而，当运行 Oracle 并行服务器时，两个或多个事例并发地访问单一的数据库。每个事例有它自己的线程。

本章描述在没有使用 Oracle 并行服务器时怎样配置和管理联机重做日志。因此，在所有的讨论和例子语句中，线程数可以假定为 1。

### 6.1.2 联机重做日志内容

联机重做日志中充满了重做记录 (redo records)。重做记录，也称作重做项 (redo entry)，由一组更改向量 (change vectors) 组成，每一个都描述数据库中一个块的改变。例如，如果

用户改变雇员表的工资值，用户生成一个包含描述表的数据段块、回滚段数据块、回滚段事务表更改的更改向量重做记录。

用户可以用重做项记录来重建所有数据库所作改变的数据，包括回滚段。因此，联机重做日志也保护回滚数据。当用户恢复使用重做数据的数据库时，Oracle 读取在重做记录中的更改向量，对相关块实施更改。

重做记录在 SGA 的重做日志缓冲器中用循环方式放入缓冲器，通过 Oracle 后台进程日志书写器(LGWR)来写入联机重做日志文件之中的一个文件。不论什么时候提交事务，LGWR 将从 SGA 的重做日志缓冲器中将事务的重做记录写到联机重做日志文件中，并且，分配系统更改号(system change number,SCN)来识别每一个递交事务的重做记录。只有当所有的与给定的事务相关的重做记录安全地位于在磁盘上的联机日志中，系统才会通知用户进程：事务已经递交。

重做日志也能在相应的事务递交之前写到联机重做文件中。如果重做日志缓冲器已经满了，或有其它的事务提交，即使一些重做记录可能不被提交，LGWR 也将所有的重做日志缓冲器中的重做日志项刷新到重做日志文件。如有必要，Oracle 可以回退这些改变。

### 6.1.3 Oracle 怎样写联机重做日志

数据库的联机重做日志由两个或更多的联机重做日志文件组成。Oracle 要求最少有两个文件来保证，其中一个可用于写操作，而另一个用于存档（如果在存档模式中）。

在循环方式中 LGWR 写联机重做日志。当当前联机重做日志满了时，LGWR 开始写到下一个可用的联机重做日志文件。当最后一个可用的联机重做日志文件满了时，LGWR 回到第一个联机重做日志，并向之写入，并再次启动循环。图 6-1 图示了联机重做日志文件的循环写入方式。每条线后面的数字号表示 LGWR 写向每一个联机重做日志文件的顺序。

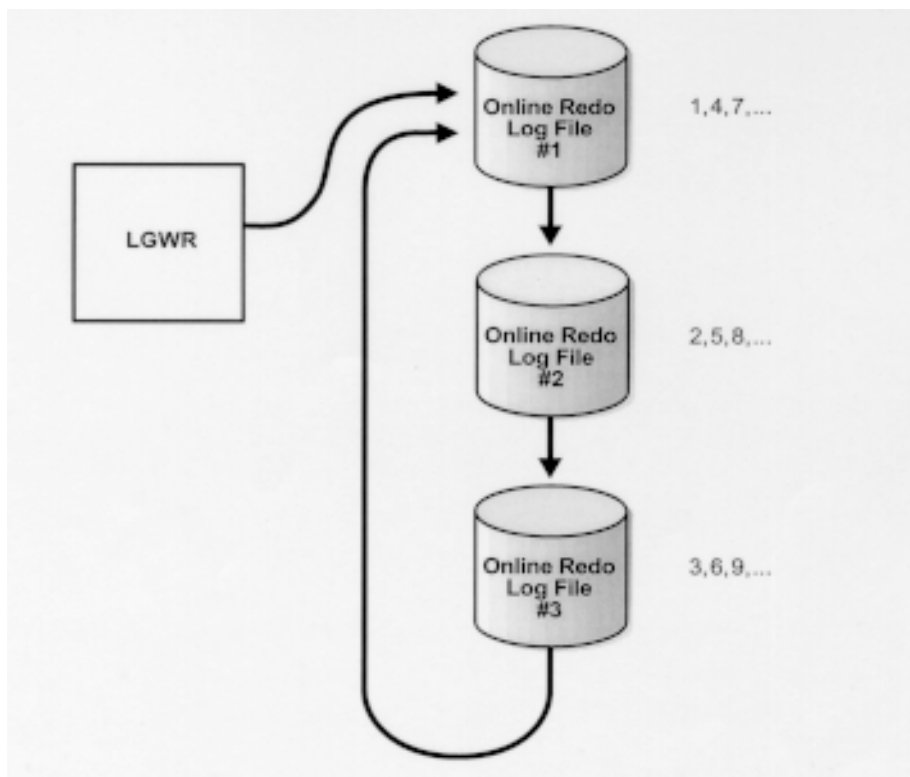


图 6-1 通过 LGWR 的联机重做日志的循环使用

充满的联机重做日志文件对于 LGWR 在重新使用时是否可用，取决于是否启用了存档：

- 如果存档没有启用（NOARCHIVELOG 模式），一个充满的联机重做日志文件在内部更改写入数据文件后即可用。
- 如果存档被启用，（ARCHIVELOG 模式），一个充满的联机重做日志文件在内部更改已经写入数据文件，且数据文件已经存档后即对 LGWR 可用。

#### 激活的（当前）和非激活的联机重做日志文件

在任何给定的时间，Oracle 只使用联机重做日志文件中的一个来存储由重做日志缓冲器写入的重做记录。LGWR 正在写的联机重做日志文件叫做当前（current）联机重做日志文件。

由事例恢复要求的联机重做日志文件叫做激活的（active）联机重做日志文件。不是由事例恢复要求的联机重做日志文件叫做非激活的（inactive）联机重做日志文件。

如果用户启用存档，Oracle 不能重新使用或覆盖一个激活的联机日志文件，直到 ARCH 已经存档了它的内容为止。如果存档没有启用，在最后一个联机重做日志充满了以后，通过覆盖第一个可用的激活文件，写入继续进行。

#### 日志切换和日志序列号

日志切换（log switch）是一个点，在这个点，Oracle 结束向一个联机重做日志文件的写入，开始向另一个写入。日志切换总是在当前联机重做日志文件完全充满了时发生，并且必须继续向下一个联机重做日志写入。用户也可以手工地强制进行日志切换。



在每次日志切换发生时，Oracle 给每一个联机重做日志文件分配一个新的日志序列号（log sequence number），并且 LGWR 开始向它写入。如果 Oracle 存档联机重做日志文件，那么，存档的日志保留它的日志序列号。循环使用的联机重做日志文件得到下一个可用的日志序列号。

每一个联机或存档的重做日志文件由它的日志序列号唯一地识别。在崩溃、事例、或介质恢复期间，通过使用必要的存档和联机重做日志文件的日志序列号，Oracle 按升序适当地应用重做日志文件。

## 6.2 规划联机重做日志

本节描述在配置数据库事例的联机重做日志时，用户应该考虑的准则。其中包括下面的专题：

- 多工联机重做日志文件
- 在不同的磁盘上放置联机重做日志成员
- 设置联机重做日志成员的大小
- 选择联机重做日志文件的数量

### 6.2.1 多工联机重做日志文件

Oracle 提供多工（multiplex）事例的联机重做日志文件的能力，以防止联机重做日志文件损坏。在多工联机重做日志文件时，LGWR 并发地将相同的重做日志信息写入多个同样的联机重做日志文件中，因此消除造成重做日志失败的一个原因。

**注意** Oracle 提醒用户多工用户的重做日志文件。如果要求恢复，日志文件数据的损失可能是灾难性的。

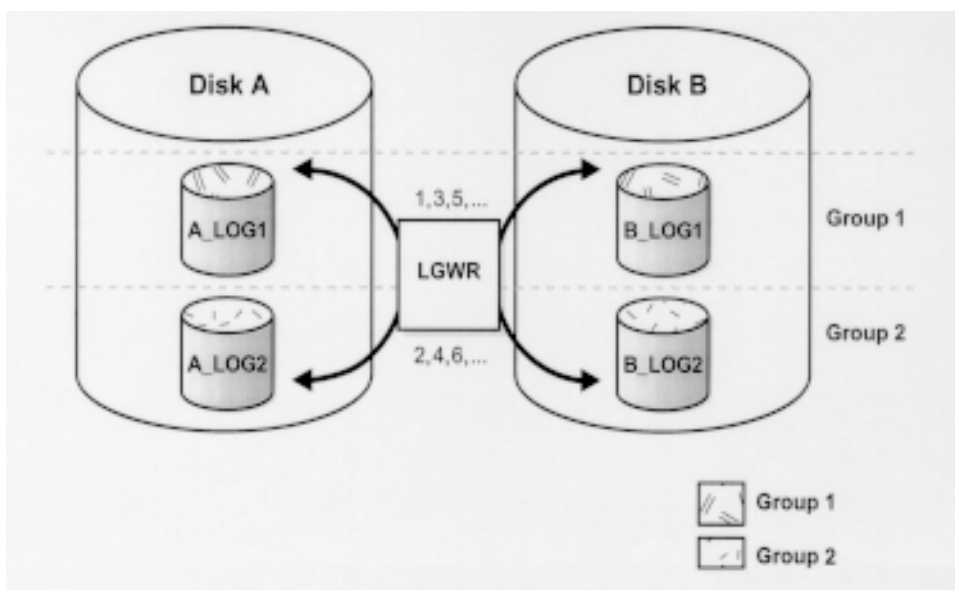


图 6-2 多工的联机重做日志文件

响应联机重做日志的一组文件称做组（groups）。每一个组中的联机重做日志文件称做成员（member）。在图 6-2 中，A\_LOG1 和 A\_LOG2 都是组 2 中的成员，依次类推。每个组中的成员必须有恰好相同的大小。注意每个组中的成员都是并发激活的。或者，并发地被 LGWR 写入。在图 6-2 中，第一个 LGWR 写到与 B\_LOG1 连接的 A\_LOG1 中，然后是与 B\_LOG2 连接的 A\_LOG2 以此类推。LGWR 从来不会并发地写到不同组的成员中（例如，到 A\_LOG1 和 B\_LOG2）。

**响应联机重做日志失败**

不论何时 LGWR 不能写到组的成员中时，Oracle 将那个成员标记为失败，在 LGWR 的踪迹文件和数据库的报警文件写一个错误消息，表示不可访问文件的问题。在某一个联机重做日志成员不可用时，根据不可用的原因，LGWR 有不同的反应。

| 如果                                   | 那么   |
|--------------------------------------|--|
| LGWR 可以成功地写向组中至少一个成员                 | 写操作进行正常；LGWR 简单地写向可用的组的成员，忽略不可用的成员   |
| LGWR 因为组需要存档，在日志切换时不能访问下一个组          | 数据库操作临时停止，直到组变得可用为止，或者到组被存档为止  |
| 在日志切换时，由于介质失败，下一个组的所有成员对 LGWR 都不可访问。 | Oracle 返回一个错误，数据库事例关闭。在这种情况下，用户可能需要执行数据库的从联机重做日志文件的损失处的介质恢复<br>如果数据库检查点已经移出丢失的重做日志（这个例子中不是当前日志），在 Oracle 已经存储数据记录于数据文件的重做日志中后，介质恢复就不重要了。简单地删除不可访问的重做日志组。如果 Oracle 不存档坏日志，使用 ALTER DATABASE CLEAR UNARCHIVED LOG 在日志可以被删除之前禁止存档 |
| 如果所有的组成员在向它们写入时突然对于 LGWR 变得不可访问      | Oracle 返回一个错误，数据库事例立即关闭。在这个情况下，用户可能需要执行介质恢复。如果包含日志的介质实际没有丢失，例如，如果日志的驱动器不小心关掉了，介质恢复可能就不需要了。在这种情况下，用户只需要将驱动器重新打开，让 Oracle 来执行事例恢复  |

## 合法与非法的配置

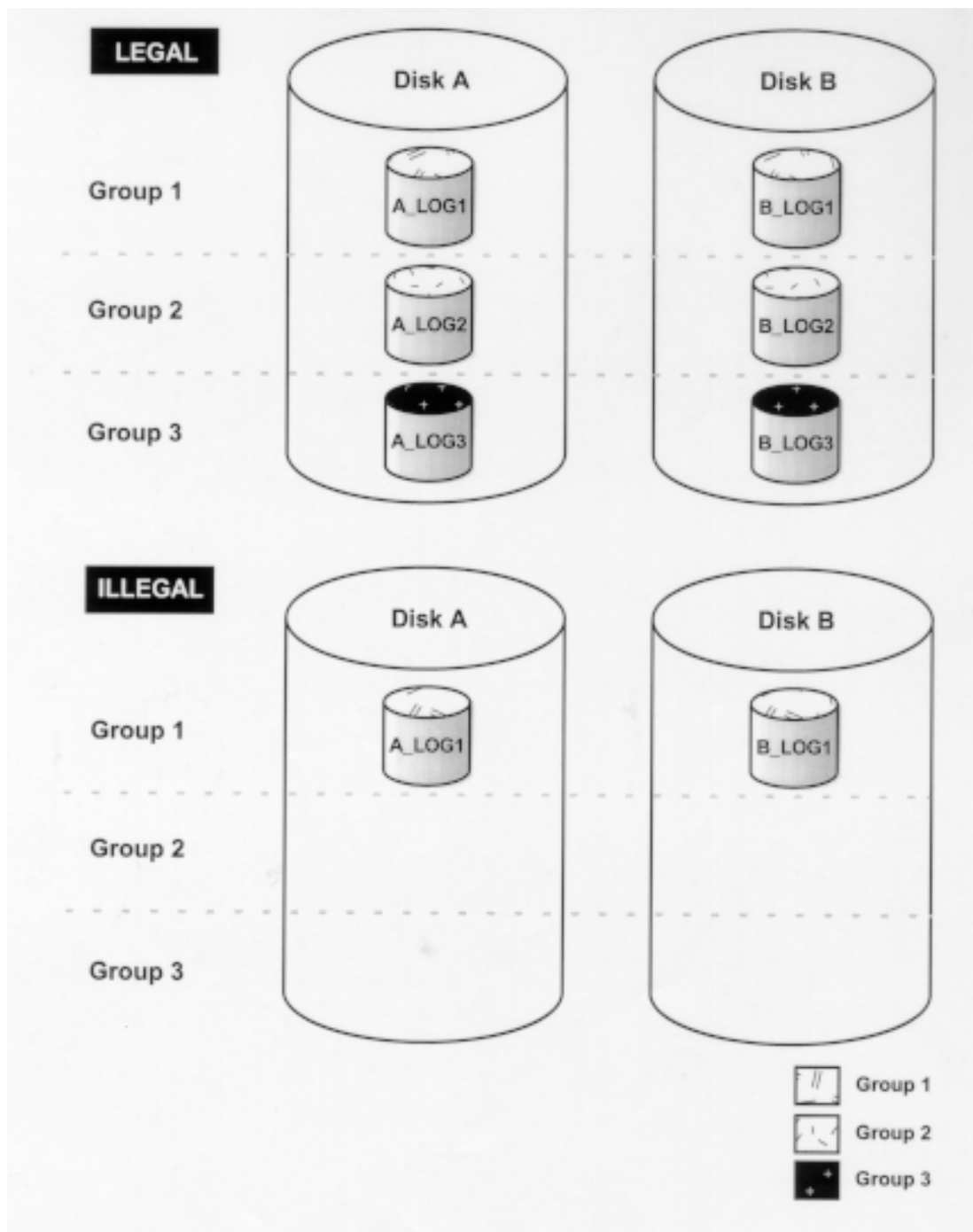


图 6-3 合法与非法的多工联机重做日志配置

要防止一个联机重做日志的单一失败，应使多工的联机重做日志是完全对称的：所有联机重做日志组有相同的成员号。不过，Oracle 不要求多工联机重做日志是对称的。例

如，一个组可以只有一个成员，而其它组有两个成员。这个配置防止磁盘失败，临时地影响一些联机的重做日志成员，但其它的不受影响。

事例的联机重做日志的唯一要求是有至少两个以上的组。图 6-3 表示合法与非法的多工联机重做日志配置。第二个配置是非法的，因为它只有一个组。

### 6.2.2 在不同的盘上放置联机重做日志成员

在设置多工联机重做日志时，应在不同的盘上放置组的成员。如果只是一个磁盘失败，那么组中只有一个成员对 LGWR 不可用，其他的成员对于 LGWR 仍然可以访问。这样事例仍然可以继续它的功能。

如果用户对联机重做日志进行存档，将在磁盘之间传播联机重做日志来清除 LGWR 和 ARCn 后台进程的连接。例如，如果用户有两个组的双工联机重做日志成员，则将每个成员放在不同的盘上，并设置存档目标为第五个盘。因此，不会再有 LGWR(写入成员)和 ARCn(读取成员)之间的竞争。

数据文件和联机重做日志文件也应该放在不同的盘上，来缩减在写数据块和写重做记录之间的竞争。

参见 关于联机重做日志怎样影响备份和恢复的更多信息，参阅“Oracle8i Backup and Recovery Guide”。

### 6.2.3 设置联机重做日志成员的大小

在设置联机重做日志文件大小时，考虑是否要存档重做日志。联机重做日志文件应该确定大小，以便充满的组可以被存档到脱机存储介质（如磁盘或磁带）的单一单元中，并使介质中未用空间量最小。例如，假设只有一个充满的联机重做日志组可以适用于磁带，并且磁带有 49% 的存储空间保留未用，在这种情况下，最好稍微减少一下联机重做日志文件的大小，这样使每个磁带上可以存档两个日志组。

通过多工联机重做日志的组，所有相同组的成员必须有相同的大小。不同组的成员可以有不同的大小，然而，在组之间变化大小没有任何优势可言。如果检查点在日志切换之间没有设置成发生，应该将所有的组设置成相同的大小，以保证检查点在有规律的间隔下发生。

参见 缺省的联机重做日志文件的大小取决于操作系统。要获取更多的细节，请参阅特定操作系统的 Oracle 文档。

### 6.2.4 选择联机重做日志文件数量

确定数据库事例的联机重做日志文件的合适数量的最好方法是检测不同的配置。优化的配置有最少的组，不妨碍 LGWR 书写重做日志信息。

在一些情况下，数据库事例可能只要求两个组。在其他的情况下，数据库事例可能要求附加的组来保证循环的组总对 LGWR 可用。在测试过程中，确定当前联机重做日志配置是否满意的最简单方法是检查 LGWR 踪迹文件和数据库报警日志的内容。如果消息指出 LGWR 必须频繁地等待组，原因为检查点没有完成或组还没有存档，就要添加组了。

在设置或更改事例的联机重做日志配置之前，应考虑可以限制联机重做日志文件数量的参数。下面的参数限制可由用户添加进数据库的联机重做日志文件的数量。

■ 参数 MAXLOGFILES 用于 CREATE DATABASE 语句中，确定每个数据库的联机

重做日志文件组的最大数量。组值范围为 1 到 MAXLOGFILES。替换这个上限的唯一方法是重新创建数据库或它的控制文件。这样，在创建数据库之前考虑这个参数是非常重要的。如果 MAXLOGFILES 对于 CREATE DATABASE 语句没有指定，则 Oracle 使用操作系统的缺省值。

■ 初始化参数 LOG\_FILES（在初始化参数文件中）可以临时地减少当前事例期间的联机重做日志文件组的最大数量。然而，LOG\_FILES 不能替换 MAXLOGFILES 来增加这个极限。如果 LOG\_FILES 在数据库参数文件中没有设置，Oracle 则使用特定操作系统的缺省值。

■ 参数 MAXLOGMEMBERS 用于 CREATE DATABASE 语句，确定每个组的最大成员数量。和使用 MAXLOGFILES 一样，替换这个上限的唯一方法是重新创建数据库或控制文件。这样，在数据库创建之前考虑这个参数是非常重要的。如果在 CREATE DATABASE 语句中没有指定 MAXLOGMEMBERS 参数，Oracle 使用操作系统的缺省值。

参见 关于 MAXLOGFILES 和 MAXLOGMEMBERS 参数和初始化参数 LOG\_FILES 的缺省与合法值，参阅用户特定操作系统的 Oracle 文档。

## 6.3 创建联机重做日志组和成员

在数据库创建期间规划数据库的联机重做日志，要创建所有要求的联机重做日志文件的组和成员。然而，有的用户可能在数据库创建之后希望创建附加的联机重做日志文件的组和成员。例如，在联机重做日志上添加组可以纠正重做日志组的可用性问题。

要创建新的联机重做日志组和成员，用户必须有 ALTER DATABASE 系统权限。数据库可以有多个 MAXLOGFILES 个组。

### 6.3.1 创建联机重做日志组

要创建新的联机重做日志文件组，使用带有 ADD LOGFILE 子句的 SQL 语句 ALTER DATABASE。

下面的语句给数据库添加一个新的重做日志组：

```
ALTER DATABASE ADD LOGFILE ('/oracle/dbs/log1c.rdo', '/oracle/dbs/log2c.rdo') SIZE 500K;
```

**注意** 使用新日志成员的全指定名称来表示操作系统文件应该创建的位置。否则，根据用户的操作系统，文件将被创建在数据库服务器的缺省或当前的目录下。要重新使用现存的操作系统的文件，用户不必指出文件大小。

用户也可以确定识别组的号码，使用 GROUP 选项：

```
ALTER DATABASE ADD LOGFILE GROUP 10 ('/oracle/dbs/log1c.rdo', '/oracle/dbs/log2c.rdo')  
SIZE 500K;
```

使用组号可以使得管理重做日志组更容易一些。然而，组号必须在 1 和 MAXLOGFILES 之间。不要跳过重做日志文件的组号，（即，不要使用组号如 10，20，30，等等），否则用户将消耗数据库控制文件的空间。

### 6.3.2 创建联机重做日志成员

在某些情况下，可能不必要创建一个完整的联机重做日志文件的组。组可能已经存在，但由于组中一个或多个成员被删除而不完整（例如，由于磁盘失败）。在这种情况下，用户可以向现存的组中添加新成员。

要为现存的组添加新的联机重做日志成员，使用带有 **ADD LOG MEMBER** 参数的 **SQL** 语句 **ALTER DATABASE**。下面的语句将向 2 号重做日志组中添加新的重做日志成员：

```
ALTER DATABASE ADD LOGFILE MEMBER '/oracle/dbs/log2b.rdo' TO GROUP 2;
```

**注意** 文件名必须被指定。但大小不需要指定。新成员的大小由组的现存成员来决定。

当使用 **ALTER DATABASE** 语句时，用户可能作为一种选择，通过指定所有 **TO** 参数中组的其它成员来识别目标组。如下面例子所示：

```
ALTER DATABASE ADD LOGFILE MEMBER '/oracle/dbs/log2c.rdo' TO  
('/oracle/dbs/log2a.rdo', '/oracle/dbs/log2b.rdo');
```

**注意** 要完整地指定新的日志成员的文件名，指出操作系统将文件创建于何处。否则，文件将根据用户的操作系统，存储在数据库服务器的缺省或当前的目录之下。用户可能也要注意新日志成员的状态显示为 **INVALID**。这是正常的，它将在第一次使用时改为激活（空白）。

### 6.4 联机重做日志的重命名和重定位

用户可以通过重新命名联机重做日志文件来改变它们的位置。这个过程是必要的。例如，如果当前有一些联机重做日志文件使用的磁盘设备被删除，或者在数据文件和大量的联机重做日志文件存储于一个磁盘上，且应该分开来减少争用时。

要重新命名联机重做日志成员，用户必须有 **ALTER DATABASE** 系统权限。另外，用户可能也需要操作系统权限来将文件拷贝到希望的地方，应有打开和备份数据库的权限。

在重新命名联机重做日志成员之前，保证新的联机重做日志文件已经存在。

**警告** 下面的步骤仅仅在数据库控制文件中修改内部文件指针。它们不是物理的重命名或创建任何操作系统文件。使用用户计算机的操作系统来拷贝现存联机重做日志文件到新的位置。

要重新命名联机重做日志成员：

#### 1. 备份数据库。

在对数据库进行任何结构性更改之前，例如对联机重做日志文件进行重命名或重定位，要完整地备份数据库（包括控制文件），以防万一用户在执行这些操作时遇到问题。

#### 2. 拷贝联机重做日志文件到新的地址。

操作系统文件，例如联机重做日志成员，必须用合适的操作系统命令来拷贝。参阅用户特定操作系统的文档，以获得关于拷贝文件的更多信息。

**注意** 用户可以在没有现存的 **SQL\*Plus** 的情况下使用 **HOST** 命令来运行操作系统命令来拷贝一个文件。

#### 3. 重命名联机重做日志成员。

使用带有 RENAME FILE 子句的 ALTER DATABASE 语句来重命名数据库的联机重做日志文件。

#### 4. 打开数据库进行一般操作。

联机重做日志改变在下次数据库打开时发生作用。打开数据库可能要求关闭当前的事例（如果数据库先前被当前的事例打开），或者仅打开使用当前事例的数据库。

#### 5. 备份控制文件

作为预防，在对一系列的联机重做日志文件重命名或重定位之后，立刻备份数据库的控制文件。

下面的例子重命名联机重做日志文件。然而，首先假设：

- 数据库当前由事例装入，但随之关闭。
- 日志文件位于两个盘上：diska 和 diskb。
- 联机重做日志是双工的：一个组中包含两个成员 /diska/logs/log1a.rdo 和 /diskb/logs/log2a.rdo，并且第二个组中包括成员 /diska/logs/log2a.rdo 和 /diskb/logs/log2b.rdo。
- 位于 diska 的联机重做日志必须重新定位到 diskc。新文件名将反映新的地址：/diskc/logs/log1c.rdo 和 /diskc/logs/log2c.rdo。
- 在 diska 上的文件 /diska/logs/log1a.rdo 和 /diska/logs/log2a.rdo 必须拷贝到 diskc 上的新文件 /diskc/logs/log1c.rdo 和 /diskc/logs/log2c.rdo。

ALTER DATABASE RENAME FILE

```
    '/diska/logs/log1a.rdo', '/diska/logs/log2a.rdo'  
    TO '/diskc/logs/log1c.rdo', '/diskc/logs/log2c.rdo';
```

## 6.5 删除联机重做日志组和成员

在某些情况下，用户可能希望删除整个联机重做日志成员的组。例如，用户希望减少事例的联机重做日志中的组的个数。在另一种情况，用户可能希望删除一个或多个指定联机重做日志成员。例如，如果磁盘失败发生，用户可能需要删除失败盘上的所有的联机重做日志，这样 Oracle 将不再试图写那些不可访问的文件。在其它的情况下，特殊的联机重做日志文件变得不需要。例如，文件可能存储于一个不合适的地方。

### 6.5.1 删除日志组

要删除一个联机重做日志组，用户必须有 ALTER DATABASE 系统权限。在删除联机重做日志组之前，考虑下面的限制和预防措施：

- 不管组中有多少个成员（一个组中有一个或多个成员），事例要求至少两个联机重做日志文件组。
- 只有非激活的联机重做日志组，用户才可以删除。如果用户需要删除激活的组，首先强制日志切换发生。参阅 6.6 节“强制日志切换”。
- 保证联机重做日志组在删除之前是存档的（如果存档可能）。要看存档是否已经发生，则使用带有 LIST 参数的 SQL\*Plus ARCHIVE LOG 语句。

用带有 **DROP LOGFILE** 子句的 **SQL** 语句 **ALTER DATABASE** 来删除联机重做日志组。

下面的语句删除 3 号重做日志组：

```
ALTER DATABASE DROP LOGFILE GROUP 3;
```

在一个联机重做日志组从数据库中删除时，操作系统文件不从磁盘中删除。相反，与数据库相关的控制文件更新，从数据库结构中删除组的成员。在删除了一个联机重做日志组之后，保证成功地完成了删除。然后使用合适的操作系统命令来删除已经删除的联机重做日志文件。

### 6.5.2 删除联机重做日志成员

要删除联机重做日志成员，用户必须拥有 **ALTER SYSTEM** 系统权限。

在删除单个的联机重做日志成员之前，考虑下面的限制和预防措施：

- 删除联机重做日志文件是容许的，这样多工的联机重做日志变成暂时不对称的。例如，如果用户使用双工的联机重做日志文件组，用户可以删除一个组中的一个成员，即使其它组每个都有两个成员。然而，用户应该立即调整这个情况，使得所有的组至少有两个成员。从而消除联机重做日志可能失败的一个点。
- 不管组中成员的个数，一个事例总是要求至少两个有效的联机重做日志文件。（一个组有一个或多个成员。）如果用户希望删除的成员是上一个有效的组的成员，则直到其它的成员变为有效时，用户才能删除该成员。要查看重做日志文件的状态，可使用 **V\$LOGFILE** 视图。如果 **Oracle** 不能访问一个重做日志文件，该文件就变成 **INVALID**。如果 **Oracle** 怀疑日志文件不完整或不正确，则该文件变成 **STALE**（失时效）。一个失时效的日志文件在下一它的组成为激活组的时候再次变为有效。
- 只有它不是激活组的一部分时，用户可以删除联机重做日志成员。如果用户希望删除一个激活组的成员，首先强制日志切换发生。
- 保证在删除联机重做日志成员之前，属于这个成员的组是存档的（如果存档被启用）。要看是否这已经发生，使用带有 **LIST** 参数的 **SQL\*PLUS ARCHIVE LOG** 语句。

要删除特定非激活的联机重做日志成员，应使用带有 **DROP LOGFILE MEMBER** 子句的 **SQL ALTER DATABASE** 语句。

下面的语句删除重做日志/oracle/dbs/log3c.rdo：

```
ALTER DATABASE DROP LOGFILE MEMBER '/oracle/dbs/log3c.rdo';
```

当联机重做日志成员从数据库删除时，操作系统文件没有从磁盘上删除。相反，与数据库相关的控制文件从数据库结构中删除成员来更新。在删除联机重做日志文件之后，保证成功地、完整地删除，然后使用合适的操作系统命令，来删除已删除的联机重做日志文件。关于删除激活组成员的信息，参阅下面的“强制日志切换”一节。

参阅 关于 **SQL\*Plus** 命令语法的更多信息，参阅 “**SQL\*Plus User's Guide and Reference**”。



## 6.6 强制日志切换

在 LGWR 停止向一个联机重做日志组中写入，并开始写向另一个联机重做日志组时，发生日志切换。缺省情况下，日志切换在当前联机重做日志文件组充满了时自动发生。

用户可以强制日志切换来使当前的激活组成为非激活状态，使得可以进行联机重做日志维护操作。例如，用户希望删除当前激活的组，但直到组为非激活时，才能那么做。在组成员完全满了之前的特定时间，如果当前激活组需要存档，用户可能也希望强制日志切换。这个选项在需要长时间才能充满的大联机重做日志文件的配置中是有用的。

要强制日志切换，用户必须有 ALTER SYSTEM 权限。要强制日志切换，使用任何一个带 SWITCH LOGFILE 选项的 SQL 语句 ALTER SYSTEM。

下面的语句强制日志切换：

```
ALTER SYSTEM SWITCH LOGFILE;
```

参见 在 Oracle 并行服务器上强制日志切换的信息，参阅 “Oracle8i Parallel Server Administration, Deployment, and Performance”。

## 6.7 校验重做日志文件中的块

用户可以使用校验和校验重做日志文件来配置 Oracle。设置初始化参数 LOG\_BLOCK\_CHECKSUM 为 TRUE，来启用重做日志块的检查。LOG\_BLOCK\_CHECKSUM 的缺省值是 FALSE。

如果用户启用重做日志块的检查，Oracle 为每一个写到当前日志的重做日志块计算校验和。Oracle 在块的首部写校验和。

Oracle 使用校验和来检测重做日志块中的讹误。Oracle 在将重做日志块写到一个存档日志文件时，以及在恢复期间重做日志块从一个存档日志中读取时，试图校验重做日志块。

如果 Oracle 在试图存档时在重做日志块中检测到一个讹误，系统试图从组中的另一个成员里来读取块。如果块在重做日志组中的每一个成员里都有问题，则存档就不能进行。

## 6.8 清除联机重做日志文件

如果用户已经启用重做日志块检查，Oracle 在存档前对每个块进行校验。如果个别的重做日志块在组中所有的成员里都有问题，存档停止。最后，所有的重做日志变满，存档停止，直到存档可以再次进行时为止。

在这种情况下，使用 SQL 语句 ALTER DATABASE...CLEAR LOGFILE 来清除有错的重做日志，避免将之存档。清除的重做日志即使它们没有存档也可以使用。

下面的语句清除 3 号重做日志组中的日志文件：

```
ALTER DATABASE CLEAR UNARCHIVED LOGFILE GROUP 3;
```

### 6.8.1 限制

不论是一个重做日志文件是否是存档的，用户都可以清除它。然而，当它不是存档的

时，用户必须在用户的 `ALTER DATABASE CLEAR LOGFILE` 语句中包括关键词 `UNARCHIVED`。如果用户清除了一个在备份恢复时需要的日志文件，那么，用户不再可以从该备份中恢复了。Oracle 在报警日志中写入一个消息，描述用户不能恢复的那个备份。

**注意** 如果用户清除了一个非存档的重做日志文件，则用户应该制作数据库的另一个备份。

如果用户希望清除一个非存档的重做日志，这个日志需要用来使脱机的表空间联机，则使用 `ALTER DATABASE CLEAR LOGFILE` 语句中的 `UNRECOVERABLE DATAFILE` 子句。

如果用户清除将脱机的表空间变成联机所需的重做日志，用户将不能使表空间再次联机了。用户将必须删除表空间或执行不完全的恢复。注意，正常脱机的表空间不要求恢复。

参见 关于 `ALTER DATABASE` 语句的完整的描述，参阅 “Oracle8i SQL Reference”。

## 6.9 列出关于联机重做日志的信息

使用 `V$LOG`、`V$LOGFILE` 和 `V$THREAD` 视图来查看关于数据库的联机重做日志的信息。`V$THREAD` 视图对于并行服务器系统管理员有特殊的兴趣。

下面的查询返回关于所有不在并行服务器上使用的数据库的联机重做日志信息：

```
SELECT group#, bytes, members FROM sys, v$log;
```

| GROUP# | BYTES | MEMBERS |
|--------|-------|---------|
| 1      | 81920 | 2       |
| 2      | 81920 | 2       |

要看组中所有成员的名字，使用类似如下的查询：

```
SELECT * FROM sys.v$logfile
WHERE group# = 2;
```

| GROUP# | STATUS | MEMBER |
|--------|--------|--------|
| 2      |        | LOG2A  |
| 2      | STALE  | LOG2B  |
| 2      |        | LOG2C  |

如果 `STATUS` 对于一个成员是空的，那么这个文件正被使用。

# 7 管理存档重做日志

本章描述怎样存档重做数据，包括如下专题：

- 什么是存档重做日志？
- 在存档日志和非存档日志之间进行选择
- 控制存档模式
- 确定存档目标
- 确定日志传输模式
- 管理存档目标失败
- 显示存档重做日志信息
- 控制由存档日志程序产生的踪迹输出
- 使用日志采集器来分析联机重做日志和存档重做日志

参见 如果用户使用并行服务器的 Oracle，参阅“Oracle Parallel Server Administration, Deployment, and Performance”以获得关于 OPS 环境中存档的附加信息。

## 7.1 什么是存档重做日志

Oracle 允许用户将充满的称为存档重做日志（archived redo logs）的联机重做日志文件组，保存到一个或多个脱机的目标中。存档（Archiving）是将联机重做日志转变为存档的重做日志。后台进程 ARCn 使存档操作自动进行。用户可以使用存档的日志来进行下列操作：

- 恢复数据库
- 更新一个备用的数据库
- 借助于日志采集器来获得关于数据库历史的信息

一个存档的重做日志文件是一联机重做日志组的同一个充满的成员的一份拷贝。它包括在组中同一成员给出的重做入口，并保存组的唯一的日志序列号。例如，如果用户多工用户的联机重做日志，并且如果组 1 中包含成员文件 A\_LOG1 和 B\_LOG1，那么 ARCn 将存档这些同一成员中的一个。假如 A\_LOG1 被破坏，那么 ARCn 仍然可以存档同样的 B\_LOG2。

如果用户启用存档，直到已经完成存档时，LGWR 才能允许重新使用，并继而重写联机重做日志组。因此，存档的重做日志由于用户启用了存档而包含所创建的组的一个拷贝。

图 7-1 表示 ARCn 怎样存档重做日志。

**警告** Oracle 提醒用户不要拷贝一个当前的联机日志。如果用户这么做，并接着恢复了那份拷贝，那份拷贝将出现在重做线程的末尾。因为附加重做可能已经在线程中产生了，当用户通过提供重做日志拷贝来试图运行恢复时，恢复将错误地

检测到重做线程的结尾，并过早地结束，这可能破坏数据库。备份当前联机日志的最好方法是总是存档它，然后备份到存档日志中。

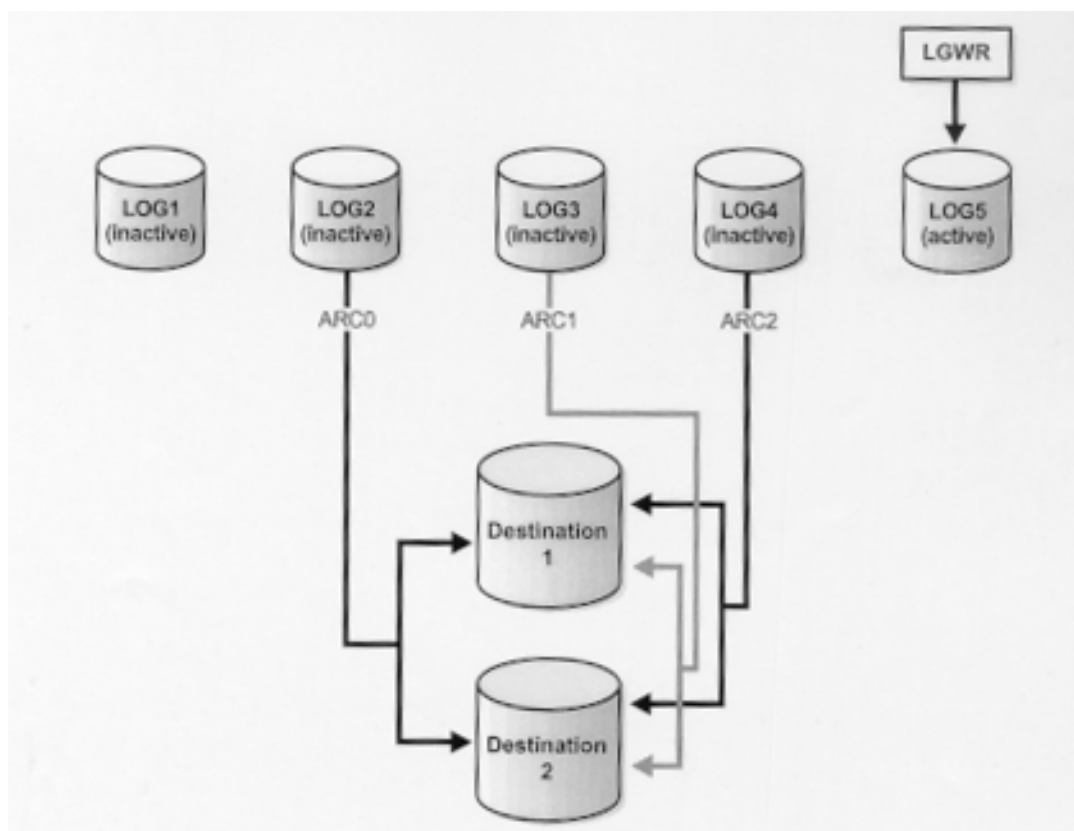


图 7-1 联机重做日志的存档

## 7.2 在存档日志和非存档日志之间选择

本节描述在选择 NOARCHIVELOG 或 ARCHIVELOG 模式来运行用户数据库时，用户必须考虑的问题。包括如下几个专题：

- 在 NOARCHIVELOG 模式下运行数据库
- 在 ARCHIVELOG 模式下运行数据库

### 7.2.1 在 NOARCHIVELOG 模式下运行数据库

当用户在 NOARCHIVELOG 模式下运行数据库时，用户禁用了联机重做日志的存档。数据库的控制文件显示充满的组不要求存档。因此，当一个充满的组在日志切换后变成非激活时，组对于 LGWR 的重新使用来说是可用的了。

是否启用充满的联机重做日志文件存档的选择取决于在数据库上运行的应用程序要求的可用性和可靠性。如果用户不能承受在磁盘失败发生时数据库的任何数据丢失，就应该使用 ARCHIVELOG 模式。注意，充满的联机重做日志文件的存档要求用户执行额外的管理

操作。

NOARCHIVELOG 模式只在事例失败时保护数据库，在介质失败时不保护。只有对数据库最近的更改，即存在联机重做日志组中的更改，对于事例恢复是可用的。换句话说，如果用户使用 NOARCHIVELOG 模式，用户只能修复(restore)（而不是恢复(recovery)）数据库到最近的完整数据库备份时那一点上。用户不能恢复后来的事务。

在 NOARCHIVELOG 模式，用户也不能执行联机表空间的备份。更进一步说，用户不能使用先前在数据库操作于 ARCHIVELOG 模式时完成的联机表空间备份。在 NOARCHIVELOG 模式，用户只能使用在数据库关闭时完成的整个数据库备份来修复数据库操作。因此，如果用户决定在 NOARCHIVELOG 模式操作数据库，要经常每隔一定时间将整个的数据库做备份。

### 7.2.2 在 ARCHIVELOG 模式下运行数据库

当用户在 ARCHIVELOG 模式下运行数据库时，用户启用了联机重做日志的存档。数据库控制文件显示直到组被存档后，充满的联机重做日志文件组才可以被 LGWR 使用。在重做日志切换发生后，一个充满的组的存档立即可用。

充满了的组的存档有下列优点：

- 数据库的备份和联机与存档的重做日志文件一起，保证了用户可以恢复在操作系统和磁盘失败中递交的事务。
- 如果用户保存一个存档的日志，用户可以使用在数据库打开和一般系统使用中完成的备份。
- 通过连续地把原始存档重做日志应用到备用数据库，用户可以保存一个与它的原始数据库同步的备用数据库。

决定用户怎样计划存档充满的联机重做日志组。用户可以配置一个事例来自动存档充满的联机重做日志文件，或者用户可以手工地存档。为了方便和高效，自动存档一般是最好的。图 7-2 图示了存档充满的组（在这个图中是 ARC0）的进程是怎样生成数据库的联机重做日志的。

如果在分布式数据库中的所有数据库都在 ARCHIVELOG 模式下操作，那么用户可以执行协同的分布式数据库恢复。然而，如果任何一个在分布式数据库中的数据库使用 NOARCHIVELOG 模式，那么全局分布式数据库（使所有的数据库一致）的恢复被最后的操作于 NOARCHIVELOG 模式的数据库的备份所限制。

用户也可以配置 Oracle，在日志块存档时校验重做日志块。这在 6.7 节“校验重做日志文件中的块”中讨论。

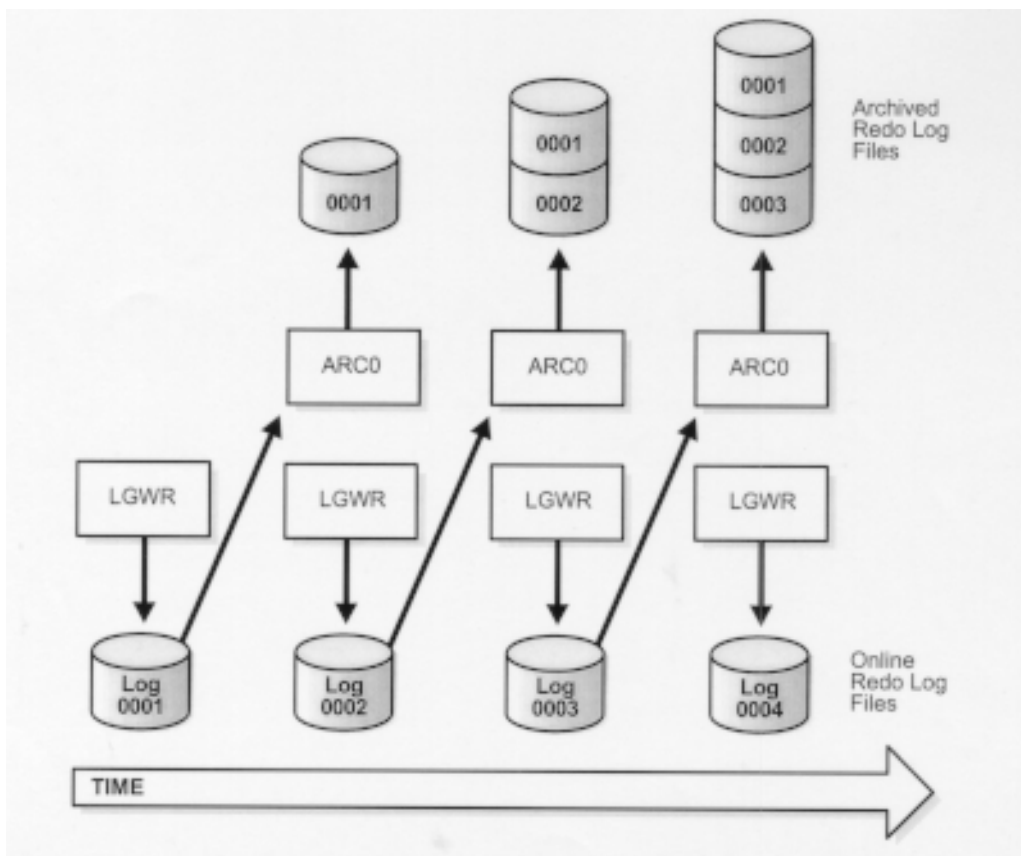


图 7-2 使用于 ARCHIVELOG 模式下的联机重做日志文件

## 7.3 控制存档模式

本节描述控制存档执行模式的方法。包括如下几个专题：

- 设置初始化数据库存档模式
- 改变初始化数据库存档模式
- 启用自动存档
- 禁用自动存档
- 执行手工存档

**参见** 如果数据库是在 Oracle 安装时自动创建的，数据库的初始化存档模式是与特定的操作系统相关的。参考特定用户操作系统的 Oracle 安装文档，以获得控制存档模式的附加信息。

### 7.3.1 设置初始化数据库存档模式

作为 CREATE DATABASE 语句中数据库创建的一部分，用户可以设置数据库的初始化存档模式。通常，用户可以在数据库创建时使用缺省的 NOARCHIVELOG 模式，这是因为那时生成的重做信息不需要存档。在数据库创建之后，决定是否更改初始化存档模式。

### 7.3.2 改变数据库存档模式

要在 NOARCHIVELOG 和 ARCHIVELOG 模式之间切换数据库的存档模式，使用带有 NOARCHIVELOG 或 ARCHIVELOG 选项的 SQL 语句 ALTER DATABASE。下面的语句切换数据库的存档模式，从 NOARCHIVELOG 切换到 ARCHIVELOG：

```
ALTER DATABASE ARCHIVELOG;
```

在切换数据库存档模式之前，执行下面的操作：

1. 关闭数据库事例。

一个打开的数据库必须关闭并卸出（dismount），并且任何与之相关的数据库都要在用户可以切换数据库存档模式之前关闭。如果有数据文件需要介质恢复时，用户不能禁用存档。

2. 备份数据库。

在对数据库做任何较大的改动之前，总要备份数据库以保证不出问题。

3. 启动一个新的事例，并装入数据库，但不打开数据库。

要启用或禁用存档，数据库必须装入但不打开。

**注意** 如果用户使用 Oracle 并行服务器，用户必须专门装入数据库，使用一个事例来切换数据库的存档模式。参阅“Oracle8i Parallel Server Administration, Deployment, and Performance”来获得更多关于在使用 Oracle 并行服务器时切换存档模式的信息。

4. 切换数据库的存档模式。

在使用 ALTER DATABASE 语句来切换数据库的存档模式之后，可用标准操作来打开数据库。如果用户切换到 ARCHIVELOG 模式，用户也必须设置附加的存档选项，指定是否启用 Oracle 的自动存档联机重做日志文件组来作为它们装入的方式。

### 7.3.3 启用自动存档

如果用户的操作系统允许，用户可以启用联机重做日志的自动存档。在这个选项下，在它装入之后，不要求任何动作来拷贝组。Oracle 自动地存档。仅仅是为了方便，自动存档是应该选择的存档方法。然而，如果自动存档被启用，用户仍然可以运行手工的存档，如 7.3.5 节“执行手工存档”中所描述的那样。

用户可以在事例启动之前或之后来启用自动存档。要在事例启动之后启用自动存档，用户必须用系统管理员权限来连接 Oracle。

当启用自动存档时，总是指定存档重做日志目标和文件名称格式。如 7.4 节“确定存档目标”中所描述。

**警告** 除非数据库也在 ARCHIVELOG 模式下，否则 Oracle 不能自动存档日志文件。

#### 在事例启动时启用自动存档

要在每次事例启动时，启用充满的组的自动存档，应该在数据库的初始化参数文件中包括初始化参数 LOG\_ARCHIVE\_START，并将它设置为 TRUE：

```
LOG_ARCHIVE_START=TRUE
```

新值在下一次用户启动数据库时起作用。

### 在事例启动后启用自动存档

要在不关闭当前事例的情况下启用充满的联机重做日志组的自动存档，使用带有 **ARCHIVE LOG START** 参数的 SQL 语句 **ALTER SYSTEM**。用户可以有选择地包括存档目标。

```
ALTER SYSTEM ARCHIVE LOG START;
```

如果用户使用 **ALTER SYSTEM** 方法，用户不需要关闭事例来启用自动存档。然而如果事例被关闭，并在自动存档启用之后重新启动，则事例使用初始化参数文件的设置重新初始化，这可能会启用自动存档，也可能不启用自动存档。

### 7.3.4 禁用自动存档

在任何时候，用户都可以禁用自动的联机重做日志组的存档。然而，一旦用户禁用自动存档，用户必须以定期的方式手工存档联机重做日志文件组。如果用户在 **ARCHIVELOG** 模式下运行数据库，并禁用自动存档，并且如果所有的联机重做日志文件组都充满了，但没有存档，那么 **LGWR** 不能重新使用任何非激活的联机重做日志组的组，来继续写重做日志项。因此，数据库操作被暂时挂起，直到用户执行必要的存档之后才停止挂起。

在事例启动时或启动之后，用户都可以禁用自动存档。要在事例启动之后禁用自动存档，用户必须以系统管理员权限来连接，并拥有 **ALTER SYSTEM** 权限。

### 在事例启动时禁用自动存档

要在每次数据库事例启动时禁用对充满的联机重做日志组的自动存档，可设置数据库初始化参数文件中的初始化参数 **LOG\_ARCHIVE\_START** 为 **FALSE**：

```
LOG_ARCHIVE_START = FALSE
```

新的值在数据库下一次启动时起作用。

### 在事例启动后禁用自动存档

要在不关闭当前事例的情况下禁用充满的联机重做日志组的自动存档，应使用带有 **ARCHIVE LOG STOP** 选项的 SQL 语句 **ALTER SYSTEM**。下面的语句用来停止存档：

```
ALTER SYSTEM ARCHIVE LOG STOP;
```

在用户试图禁用自动存档时，如果 **ARCn** 是存档的重做日志组，**ARCn** 完成当前组的存档，但不开始下一个充满的联机重做日志组的存档。

要禁用自动存档，不要求必须关闭事例。然而，如果一个事例被关闭，并在自动存档被关闭后重新启动，事例将使用初始化参数文件的设置重新初始化，这可能会启用自动存档，也可能不启用自动存档。

### 7.3.5 执行手工存档

如果用户在 **ARCHIVELOG** 模式下操作用户的数据库，那么用户必须存档充满的非激活的联机重做日志文件组。用户可以手工的存档联机重做日志组，不论是否启用自动存档：

- 如果自动存档没有启用，那么用户必须以定时的方式手工存档充满的联机重做日志文件组。如果所有的联机重做日志组都充满了，但没有存档，**LGWR** 不能重新



使用任何非激活的联机重做日志成员的组来继续写重做日志项。因此，数据库操作暂时被挂起，直到执行了必要的存档之后才停止挂起。

- 如果启用自动存档，但用户希望重新存档一个非激活的充满的联机重做日志成员的组到另一个位置，那么用户可以使用手工存档。注意，在用户完成手工存档之前，事例可以决定重新使用重做日志组，因而重写文件。如果这种情况发生，Oracle 将在 **ALERT** 文件中放置一个错误消息。

要手工地存档一个充满的联机重做日志组，则要以系统管理员权限来连接。使用带有 **ARCHIVE LOG** 子句的 **SQL** 语句 **ALTER SYSTEM** 来手工存档充满的联机重做日志文件。下面的语句存档所有的非存档日志文件：

```
ALTER SYSTEM ARCHIVE LOG ALL;
```

参见 在用户使用 Oracle 并行服务器时，可通过手工和自动两种存档，用户仅仅确定一个线程。参阅 “Oracle8i Parallel Server Administration, Deployment, and Performance” 来获得更多的信息。

7.4 确定存档目标

在存档重做日志时，决定用户将要存档的目标，并使自己精通各种各样的目标状态。请用户使用固定视图，来访问存档信息。固定视图列于 7.8 节 “显示存档重做日志信息” 中。本节讨论下面的专题：

- 确定存档目标
- 理解存档目标状态

7.4.1 确定存档目标

用户必须决定是创建日志的单一目标，还是创建日志的多工目标，也就是，将日志存档到多于一个的目标。用户通过根据下面的方法设置初始化参数来确定其选择。

| 方法 | 初始化参数  | 主机    | 例子   |
|----|--|-------|--|
| 1  | LOG_ARCHIVE_DEST_n<br>(其中 n 是一个从 1 到 5 的整数)    | 本地或远程 | LOG_ARCHIVE_DEST_1='LOCATION =<br>/disk1/arc'LOG_ARCHIVE_DEST_2='SER<br>VICE=standby1' |
| 2  | LOG_ARCHIVE_DEST 和 LOG_<br>ARCHIVE_DUPLEX_DEST | 只能是本地 | LOG_ARCHIVE_DEST =<br>'/disk1/arc'LOG_ARCHIVE_DUPLEX_DE<br>ST='/disk2/arc'             |

方法 1: 使用 LOG\_ARCHIVE\_DEST\_n 参数

第一个方法是使用 LOG\_ARCHIVE\_DEST\_n 参数（其中 n 是一个从 1 到 5 的整数）从用户存档的 1~5 个不同的目标中指定。每一个数字下标的参数唯一的识别一个专门的目标。用户可使用如下下标的 LOG\_ARCHIVE\_DEST\_n 来指定地址：

| 关键词      | 指示                | 例子                                       |
|----------|-------------------|--|
| LOCATION | 一个本地的文件系统位置。      | LOG_ARCHIVE_DEST_1='LOCATION'=disk1/arc' |
| SERVICE  | 借助于 NET8 服务名的远程存档 | LOG_ARCHIVE_DEST_2='SERVICE=standby1'    |

如果用户使用 **LOCATION** 关键词，则要指定用户操作系统的一个有效的路径名。如果用户指定 **SERVICE** 关键词，Oracle 将通过 `tnsnames.ora` 文件将网络服务名解释为一个连接的描述符。描述符中包含连接远程数据库所需的信息。注意，服务名必须有一个相关的数据库 **SID**，这样 Oracle 才能正确地更新备用数据库中控制文件的日志历史。

执行下面的步骤来设置存档日志的目标，可用如下的方法：

1. 使用 **SQL\*PLUS** 来关闭这个数据库：

```
SHUTDOWN IMMEDIATE;
```

2. 编辑 `LOG_ARCHIVE_DEST_n` 参数来从 1~5 个存档地址中指定。**LOCATION** 关键词指定一个特定 O/S 的路径名。例如，输入：

```
LOG_ARCHIVE_DEST_1='LOCATION = /disk1/archive'
```

```
LOG_ARCHIVE_DEST_2='LOCATION = /disk2/archive'
```

```
LOG_ARCHIVE_DEST_3='LOCATION = /disk3/archive'
```

如果用户正在存档一个备用的数据库，那么使用 **SERVICE** 关键词来从 `tnsnames.ora` 中指定一个有效的网络服务名。例如输入：

```
LOG_ARCHIVE_DEST_4= 'SERVICE = standby1'
```

3. 编辑初始化参数 `LOG_ARCHIVE_FORMAT`，使用 `%s` 来包括作为文件名的一部分的日志序列号，使用 `%t` 来包括线程号。使用大写字母 (`%S` 和 `%T`) 来将零填充到文件名的左边。例如，输入：

```
LOG_ARCHIVE_FORMAT = arch%s.arc
```

这些设置将生成如下序列号为 100、101、和 102 的日志的存档日志：

```
/disk1/archive/arch100.arc, /disk1/archive/arch101.arc, /disk1/archive/arch102.arc
```

```
/disk2/archive/arch100.arc, /disk2/archive/arch101.arc, /disk2/archive/arch102.arc
```

```
/disk3/archive/arch100.arc, /disk3/archive/arch101.arc, /disk3/archive/arch102.arc
```

#### 方法 2：使用 `LOG_ARCHIVE_DEST` 和 `LOG_ARCHIVE_DUPLEX_DEST`

第二个方法，允许用户指定两个位置的最大值，它使用 `LOG_ARCHIVE_DEST` 参数来指定主要存档目标，用 `LOG_ARCHIVE_DUPLEX_DEST` 参数来决定可选的次要地址。无论什么时候 Oracle 存档重做日志，都会存档到由参数设置指定的每一个目标。

使用方法 2 来执行下面的步骤：

1. 使用 **SQL\*PLUS** 来关闭这个数据库：

```
SHUTDOWN IMMEDIATE;
```

2. 确定 `LOG_ARCHIVE_DEST` 和 `LOG_ARCHIVE_DUPLEX_DEST` 参数的目标（用户也可以使用 `ALTER SYSTEM` 语句来动态地指定 `LOG_ARCHIVE_DUPLEX_DEST`）。例如，输入：

```
LOG_ARCHIVE_DEST= '/disk1/archive'
```

```
LOG_ARCHIVE_DUPLEX_DEST= '/disk2/archive'
```

3. 编辑 LOG\_ARCHIVE\_FORMAT 参数, 使用%s 作为文件名的一部分来包括日志序列号, 并用%t 来包括线程号。使用大写字母 (%S 和%T) 来将零填充文件名的左边。例如, 输入:

```
LOG_ARCHIVE_FORMAT = arch_%t_%s.arc
```

例如, 上述的设置将生成线程 1 中如下序列号为 100、101 的日志的存档日志:

```
/disk1/archive/arch_1_100.arc, /disk1/archive/arch_1_101.arc  
/disk2/archive/arch_1_100.arc, /disk2/archive/arch_1_101.arc
```

参见 关于存档到备用数据库的信息, 参阅 “Oracle8i Backup and Recovery Guide” 和 “Oracle8i Standby Database Concepts and Administration”。

7.4.2 理解存档目标状态

初始化参数 LOG\_ARCHIVE\_DEST\_n(其中 n 是一个从 1 到 5 的整数)识别指定目标的状态。目标参数可以有两个值: ENABLE 和 DEFER。ENABLE 表示 Oracle 可以使用目标, 反之, DEFER 表示不能用。

每一个存档目标有 3 个不同的特性:

- Valid/Invalid(有效/无效), 它表示是否指定磁盘地址或服务名称信息。
- Enabled/Disabled(启用/禁用), 它表示 Oracle 是否应该使用目标信息。
- Active/Inactive (激活/非激活), 它表示访问目标是否有问题。

一些目标状态是可能的, 并在目标的状况上反映出来。要获得事例的每一个目标的当前状态和其他信息, 请查询 V\$ARCHIVE\_DEST 视图。用户将访问最近输入的参数定义——这不需要包含完整的存档目标数据。

在视图中显示的決定位置状态的特性在表 7-1 中给出。注意, 对于使用的目标, 它的特性必须是有效的、启用的、激活的。

表 7-1 目标状态

| 状态        | 特性    |       |       |  | 含义  |
|-----------|-------|-------|-------|--|---|
|           | 有效    | 启用    | 激活    |  |   |
| VALID     | TRUE  | TRUE  | TRUE  |  | 用户已经正确初始化了目标, 对于存档可用                              |
| INACTIVE  | FALSE | N/A   | N/A   |  | 用户还没有提供或已经删除了目标信息                                 |
| ERROR     | TRUE  | TRUE  | FALSE |  | 创建或写向目标文件时发生错误; 涉及错误数据                            |
| DEFERRED  | TRUE  | FALSE | TRUE  |  | 用户临时手工禁用目标  |
| DISABLED  | TRUE  | FALSE | FALSE |  | 在错误之后用户手工临时禁用目标, 这涉及错误数据                          |
| BAD PARAM | N/A   | N/A   | N/A   |  | 参数错误发生。涉及错误数据。通常这个状态只是在 LOG_ARCHIVE_START 没有设置时发生 |

参见 关于 V\$ARCHIVE\_DEST 和存档目标参数的详细信息, 参阅 “Oracle8i Reference”。

7.5 指定日志传输模式

将存档日志传输到它们的目标有两种模式: 标准存档传输与备用传输模式。标准存档

传输包含传输文件到本地磁盘。备用传输包含借助网络将文件传输到本地或远程备用数据库。

### 7.5.1 标准传输模式

在标准传输模式中，存档目标是数据库服务器的另一个磁盘驱动器。因为在这种配置中，存档不与其他事例要求的文件相争夺，并且迅速完成，这样组可以变得对于 LGWR 可用。可用 LOG\_ARCHIVE\_DEST\_n 或 LOG\_ARCHIVE\_DEST 参数之一来确定目标。

理想地，用户应该永久地移动存档重做日志文件，并响应数据库从本地磁盘到廉价的脱机存储介质，如磁带备份。因为存档日志的主要价值是数据库恢复，所以用户希望在灾难袭击用户的主要数据库时，要保证这些日志是安全的。

### 7.5.2 备用传输模式

在备用传输模式中，存档目标是本地或远程的备用数据库。

**警告** 用户可以在本地磁盘上保持一个备用的数据库。但是 Oracle 强烈地鼓励用户保持用户在远程地点的备用数据库。

如果用户以管理的恢复模式操作用户的备用数据库，通过自动的应用传输存档日志，用户可以和源数据库同步地保存用户的备用数据库。

要成功地传输文件到一个备用数据库，ARCn 或服务器程序必须进行如下操作：

- 识别远程位置。
- 通过远程文件服务器（remote file server (RFS)）程序来传输存档日志。

每个 ARCn 进程为每个备用目标创建一个 RFS 响应。例如，如果三个 ARCn 进程正在向两个备用数据库存档，然后 oracle 建立六个 RTS 连接。

通过使用 Net8，用户可以通过网络来将存档日志传输到远程位置。通过指定一个 Net8 服务器名来作为目标的属性，来指示远程存档。Oracle 然后翻译用户用 SERVICE\_NAME 参数设置的服务器名，通过 tnsnames.ora 文件来连接描述符。描述符包含连接远程数据库的必要信息。注意服务器名必须有一个相关的数据库 SID。这样 Oracle 正确地更新备用数据库控制文件的日志历史。

在目标节点上运行的 RFS 程序，充当到 ARCn 客户的网络服务器。基本上，ARCn 将信息推给 RFS，由 RFS 将之传输给备用数据库。

在存档远程目标时要求的 RFS 程序，响应下面的任务：

- 消耗 ARCn 进程的网络 I/O。
- 通过使用 STANDBY\_ARCHIVE\_DEST 参数来创建备用数据库上的文件名。
- 在远程地点填充日志文件。
- 更新备用数据库的控制文件（恢复管理器可以使用它来进行恢复）。
- 存档的重做日志对于维护备用数据库来说是必需的。它是数据库的复制品。用户可以在备用存档模式中操作用户的数据库，用来自原始的数据库的存档重做日志自动更新备用数据库。

**参见** 关于备用数据库详细的描述，参阅“Oracle8i Standby Database Concepts and Administration”中相关的章节。

关于 Net8 的信息，参阅 “Net8 Administrator's Guide”。

## 7.6 管理存档目标失败

当用户在自动存档模式下操作时，有时，存档目标会失败。要减少与目标失败相关的问题，Oracle8i 提供给用户一些如本节下面所描述的选项：

- 确定成功目标的最小数量
- 重新存档到一个失败的目标

### 7.6.1 确定成功目标的最小数量

可选的初始化参数 `LOG_ARCHIVE_MIN_SUCCEED_DEST=n`（其中 `n` 时从 1 到 5 的整数，如果用户选择使用多工，则为 1 或 2）决定了 Oracle 在重新使用联机日志文件之前，必须成功存档重做日志组目标的最小数量（缺省值是 1）。

#### 指定强制和可选目标

使用 `LOG_ARCHIVE_DEST_n` 参数，用户可以指定目标是可选的（OPTIONAL）（缺省），还是强制（MANDATORY）。`LOG_ARCHIVE_MIN_SUCCEED_DEST=n` 参数使用所有的 MANDATORY 目标加上一些 OPTIONAL 非备用目标来决定 LGWR 是否可以重写联机日志。

当用户决定怎样设置他们的参数时，要注意：

- 不确定目标的 MANDATORY 与确定 OPTIONAL 相同。
- 用户必须至少有一个可以将之声明为 OPTIONAL 或 MANDATORY 的本地目标。
- 在使用 `LOG_ARCHIVE_MIN_SUCCEED_DEST=n` 时，至少一个本地目标会在运行上被处理为 MANDATORY，因为 `LOG_ARCHIVE_MIN_SUCCEED_DEST` 的最小值是 1。
- 任何 MANDATORY 目标的失败，包括 MANDATORY 备用目标，都与 `LOG_ARCHIVE_MIN_SUCCEED_DEST` 参数不相关。
- `LOG_ARCHIVE_MIN_SUCCEED_DEST` 参数不能大于目标的数量，也不能大于 MANDATORY 目标数量加上 OPTIONAL 本地目标数量的和。
- 如果用户将 MANDATORY 目标设置为 DEFER，并且 Oracle 在没有将存档日志传输到备用地点的情况下重写联机日志，那么用户必须手工将日志传输到备用地点。

用户使用 `LOG_ARCHIVE` 和 `LOG_ARCHIVE_DUPLEX_DEST` 参数也可以建立强制或可选的目标。注意下面的规则：

- 任何借助于 `LOG_ARCHIVE_DEST` 声明的目标是强制的。
- 如果 `LOG_ARCHIVE_MIN_SUCCEED_DEST=1`，那么，任何借助于 `LOG_ARCHIVE_DUPLEX_DEST` 声明的目标是可选的。如果 `LOG_ARCHIVE_MIN_SUCCEED_DEST=2`，那么，任何借助于 `LOG_ARCHIVE_DUPLEX_DEST` 声明的目标是强制的。

## 示例方案

用户通过示例方案可以最容易的看到 LOG\_ARCHIVE\_DEST\_n 和 LOG\_ARCHIVE\_MIN\_SUCCEED\_DEST 参数之间的关系。

**方案 1:** 在这个方案中，用户存档三个本地目标，每一个用户都声明为 OPTIONAL。表 7-2 图示了这种情况下，LOG\_ARCHIVE\_MIN\_SUCCEED\_DEST=n 的可能的值。

表 7-2 方案 1 的 LOG\_ARCHIVE\_MIN\_SUCCEED\_DEST 值

| 值 | 含义   |
|---|--|
| 1 | 只有在至少有 1 个 OPTIONAL 目标成功时，Oracle 才可以再次使用日志文件 |
| 2 | 只有在至少有 2 个 OPTIONAL 目标成功时，Oracle 才可以再次使用日志文件 |
| 3 | 只有在所有的 OPTIONAL 目标成功时，Oracle 才可以再次使用日志文件     |
| 4 | 出错：值大于目标的数量                                  |
| 5 | 出错：值大于目标的数量                                  |

这个方案表示，在 LOG\_ARCHIVE\_MIN\_SUCCEED\_DEST 被设置为 1、2、3 时，即使用户不使用 LOG\_ARCHIVE\_DEST\_n 来明确地设置任何目标为 MANDATORY，Oracle 必须成功地存档到这些地址。

**方案 2:** 在这个方案中，可考虑下面的情况：

- 用户确定两个 MANDATORY 目标。
- 用户确定两个 OPTIONAL 目标。
- 目标不是备用数据库。

表 7-3 表示 LOG\_ARCHIVE\_MIN\_SUCCEED\_DEST=n 的可能的值。

表 7-3 示例 2 的 LOG\_ARCHIVE\_MIN\_SUCCEED\_DEST 的值。

| 值 | 含义                                       |
|---|--|
| 1 | Oracle 忽略值，使用 MANDATORY 目标的数量（在这个示例 2 中） |
| 2 | 即使 OPTIONAL 目标都不成功，Oracle 也可以再次使用日志文件    |
| 3 | 只有在至少有一个 OPTIONAL 目标成功时，Oracle 才可以再次使用日志 |
| 4 | 只有在两个 OPTIONAL 目标都成功时，Oracle 才可以再次使用日志   |
| 5 | 出错：值大于目标的数量                              |

这个案例表示不管用户是否设置 LOG\_ARCHIVE\_MIN\_SUCCEED\_DEST 为一个较小的数，Oracle 都必须存档到用户用 MANDATORY 指定的目标。

参见 关于 LOG\_ARCHIVE\_MIN\_SUCCEED\_DEST=n 或其他与存档相关的初始化参数的附加信息，参阅 “Oracle8i Reference”。

### 7.6.2 失败目标的重新存档

使用 LOG\_ARCHIVE\_DEST 参数的 REOPEN 属性来决定在错误之后，ARCn 是否以及何时试图重新存档到一个失败的目标。REOPEN 应用于所有的错误，而不仅仅是 OPEN 错误。

REOPEN=n 设置在 ARCn 应该试图重新打开失败目标之前的最小的秒数。n 的缺省值为 300 秒。值为 0 时相当于关闭 REOPEN 选项，换句话说，ARCn 在失败之后将不再试图

存档。如果用户不指定 REOPEN 关键词，ARCn 将在错误之后永远也不重新打开目标。

用户不能使用 REOPEN 来指定试图重新连接，并传输存档日志次数的极限。在 REOPEN 信息重新设置之后，REOPEN 的尝试可能成功，也可能失败。

如果用户指定一个 OPTIONAL 目标的 REOPEN，Oracle 可以在有错误时重写联机日志。如果用户指定一个 MANDATORY 目标的 REOPEN，Oracle 在它不能成功存档时停止数据库。这个方案要求用户：

- 对失败的目标手工存档。
- 通过推迟目标，确定目标为可选的或更改服务的方法来改变目标。
- 删除目标。

当使用 REOPEN 关键词时，注意：

- ARCn 只有在从日志文件的开头启动一个存档操作时重新打开目标，在当前操作期间不打开目标。ARCn 总是在开始时重新试图进行日志拷贝。
- 如果 REOPEN 时间是指定或缺省的，ARCn 检查记录错误的时间和 REOPEN 间隔时间之和是否小于当前时间。如果是，ARCn 再试日志拷贝。
- REOPEN 子句成功地影响 ACTIVE=TRUE 目标状态；但 VALID 或 ENABLED 状态不改变。

## 7.7 调整存档性能

对于大多数数据库，ARCn 对整体的系统性能没有影响。然而，在一些大的数据库，存档可能对系统性能有影响。一方面，在 ARCn 运行时，CPU 周期在存档中被消耗，如果 ARCn 工作得很快，整体系统性能可能降低。另一方面，如果 ARCn 运行得特别慢，它对系统性能就有一点有害的影响，但如果所有的重做日志组由于等待存档而都不可用，这需要较长的时间存档重做日志文件，会产生一个瓶颈。

可使用下面的方法来调整存档：

- 指定多个 ARCn 进程
- 调整存档缓冲器参数

参见 关于调整数据库的更多信息，参阅“Oracle8i Designing and Tuning for Performance”。

### 7.7.1 指定多个 ARCn 进程

每个数据库事例最多可以指定十个 ARCn 进程。要在启动或运行时启用多处理特征，这要通过设置初始化参数 LOG\_ARCHIVE\_MAX\_PROCESS=n(其中 n 是从 1 到 10 的任意整数)来完成。缺省情况下，参数设置为 1。

因为 LGWR 在 ARCn 进程数量不足以处理当前工作量自动地增加 ARCn 进程的数量，所以参数打算允许用户指定 ARCn 进程的初始数量，或增加减少当前的数量。假设 ARCn 初始的数量是 4，下面的语句将程序的数量减少到 2：

```
ALTER SYSTEM SET LOG_ARCHIVE_MAX_PROCESS=2
```

在减少 ARCn 进程数量的时候，没有明确地决定哪个进程被停止。用户也不允许更改参数为 0，这样至少有一个 ARCn 进程是激活的。查询 V\$ARCHIVE\_PROCESSES 视图查看关于每一个存档程序的状态。已经停止的进程将在 IDLE 状态中显示出来。

当用户在下列情况下，创建多进程显得特别有用：

- 使用多于两个联机重做日志时
- 对多于一个的目标进行存档时

多个联机重做日志在写非激活日志到多个目标时，快于单个 ARCn 进程，多 ARCn 进程防止了在 LGWR 切换时发生的瓶颈。注意每个 ARCn 进程每次只能对一个非激活日志工作。但每个 ARCn 进程必须存档到每一个指定目标。

例如，如果用户维护五个联机重做日志文件，那么用户可能决定使用三个 ARCn 进程来启动事例。由于 LGWR 当前写日志文件中的一个文件，ARCn 进程可以同时存档多至三个非激活日志文件到不同的目标。如图 7-3 所示，每个 ARCn 事例假定响应一个单一的日志，并将之存档到所有定义的目标中。

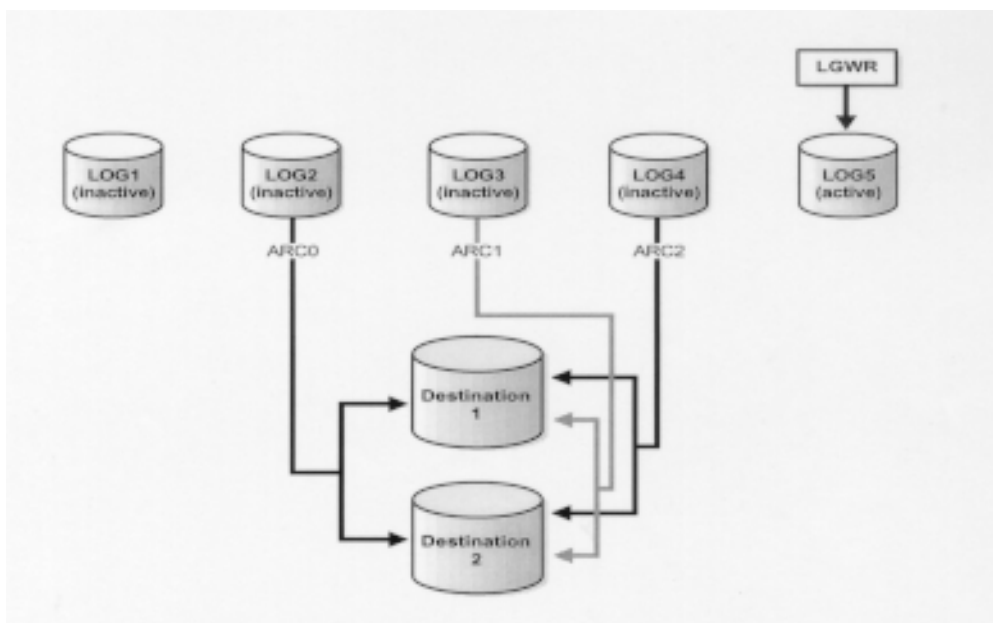


图 7-3 使用多 ARCn 进程

### 7.7.2 调整存档缓冲器参数

本节描述的是使用存档缓冲器初始化参数来进行调整。

用户可以调整存档，即可以调整得运行非常慢，但没有瓶颈，又可以调整得非常快，但不以减少系统性能为代价。要做到这一点，需要调整初始化参数。

**注意** 当用户改变 LOG\_ARCHIVE\_BUFFERS 和 LOG\_ARCHIVE\_BUFFER\_SIZE 的值时，新的值将在下次用户启动事例时起作用。



减小对系统性能的影响

要在不强制系统等待重做日志的情况下使 ARCn 工作得尽可能得慢，开始先将存档缓冲器（LOG\_ARCHIVE\_BUFFERS）的数量设置为 1，并将每个缓冲器的大小（LOG\_ARCHIVE\_BUFFERS\_SIZE）设置为最大。

如果在 ARCn 工作时系统性能显著下降，将 LOG\_ARCHIVE\_BUFFERS\_SIZE 减小，直到系统性能在 ARCn 工作时不再减少为止。

**注意** 如果用户希望将存档设置得非常慢，但发现 Oracle 经常必须等待重做日志文件在重新使用之前存档，则用户可以创建附加的重做日志文件组。添加组可以保证组对数据库使用总是可用的。

提高存档速度

要提高存档的性能，应使用多个存档缓冲器来强制 ARCn 进程来读取存档日志，同时，写输出日志。用户可以设置 LOG\_ARCHIVE\_BUFFERS 为 2，但对于一个快速的磁带驱动器，用户可能希望设置为 3 或更高。然后，将存档缓冲器的大小调整到一个中等的数并提高它，直到存档速度和用户所需的一样，而不会降低系统的性能。

参见 关于 LOG\_ARCHIVE 参数的更多信息，参阅 “Oracle8i Reference”。

7.8 显示存档重做日志信息

用户可以显示有关存档重做日志的信息，使用如下方法：

- 固定视图
- ARCHIVE LOG LIST SQL 语句

7.8.1 固定视图

有几个包含有关于存档重做日志有用信息的固定视图：

| 固定视图                 | 描述  |
|----------------------|---|
| V\$DATABASE          | 识别数据库是在 ARCHIVELOG 模式还是 NO ARCHIVELOG 模式                |
| V\$ARCHIVED          | 显示来自控制文件的历史的存档日志信息。如果用户使用恢复目录，则 RC_ARCHIVED_LOG 包含类似的信息 |
| V\$ARCHIVE_DEST      | 描述当前的事例、所有的存档目标、和这些目标当前的值、模式状态                          |
| V\$ARCHIVE_PROCESSES | 显示事例关于不同的存档进程的状态  |
| V\$BACKUP_REDOLOG    | 包含关于存档日志的备份信息。如果用户使用恢复目录，则 RC_ARCHIVED_LOG 包含类似的信息      |
| V\$LOG               | 显示数据库所有的联机重做日志组，并指出哪一个需要存档                              |
| V\$LOG_HISTORY       | 包含日志历史信息，如那个日志已经存档。还包含每个存档日志的 SCN 范围                    |

例如：下面的查询显示那个联机重做日志组要求存档：

```
SELECT group#, archived
FROM sys.v$log;
```

```
GROUP      ARC
-----
1          YES
2          NO
```

要看当前的存档模式，可查询 V\$DATABASE 视图：

```
SELECT log_mode FROM sys.v$database;
LOG_MODE
```

```
-----
NOARCHIVELOG
```

参见 关于数据字典视图的更多信息，参阅 “Oracle8i Reference”。

### 7.8.2 SQL 语句 ARCHIVE LOG LIST

SQL 语句 ARCHIVE LOG LIST 也显示连接事例的存档信息：

```
ARCHIVE LOG LIST;
Database log mode                ARCHIVELOG
Automatic archival                ENABLED
Archive destination               destination
Oldest online log sequence       30
Next log sequence to archive     32
Current log sequence number      33
```

这个显示告诉用户所有关于当前事例存档重做日志设置的所需信息：

- 数据库当前操作于 ARCHIVELOG 模式下。
- 启用自动存档。
- 存档重做日志的目标（操作系统指定）。
- 最老的充满的联机重做日志组的序列号是 30。
- 下一个要存档的充满的联机重做日志组的序列号是 32。
- 当前联机重做日志文件的序列号是 33。

用户必须用一个等于或大于 Next log sequence to archive，但小于 Current log sequence number 的序列号来存档所有的重做日志组。例如，上面显示指出，序列号为 32 的联机重做日志组需要存档。

参见 关于 ARCHIVE-LOG-LIST 语句的更多信息，参阅 “Oracle8i SQL Reference”。

## 7.9 控制由存档日志程序产生的踪迹输出

如 4.3.3 节“踪迹文件、报警日志和后台进程”中所述，后台进程在合适时总是写一个踪迹文件。在存档日志程序的情况下，有可能控制生成的输出。

初始化参数 LOG\_ARCHIVE\_TRACE 可以设置来指定一个跟踪级（trace level）。下面的值可以被指定：

| 跟踪级 | 含义              |
|-----|-----------------|
| 0   | 禁用存档日志跟踪——缺省设置  |
| 1   | 跟踪 REDO 日志文件的存档 |
| 2   | 跟踪每个存档日志目标的存档状态 |
| 4   | 跟踪存档操作状态        |
| 8   | 跟踪存档日志目标活动      |
| 16  | 跟踪详细的存档日志目标活动   |
| 32  | 跟踪存档日志目标参数的改变   |
| 64  | 跟踪 ARCn 进程状态活动  |

用户可以通过指定一个等于用户想要跟踪的单独级之和的值来组合跟踪级。例如，`LOG_ARCHIVE_TRACE=12`，将生成跟踪级 8 和 4 输出。用户可以将主数据库和备用数据库设置成不同的值。

`LOG_ARCHIVE_TRACE` 参数的缺省值是 0，在这个级，错误情况将仍然会生成合适的报警和跟踪项。

用户可以使用 `ALTER SYSTEM` 语句，动态地改变这个参数的值。例如：

```
ALTER SYSTEM SET ARCHIVE_LOG_TRACE=12
```

用这种方式改变初始值将在下次存档操作启动时产生影响。

参见 `LOG_ARCHIVE_TRACE` 初始化参数在“Oracle8i Reference”中讨论。

关于用备用数据库时使用这个参数的信息，参阅“Oracle8i Standby Database Concepts and Administration”。

## 7.10 使用日志采集器来分析联机重做日志和存档重做日志

Oracle 实用工具日志采集器（LogMiner）允许用户基于选择标准读取联机和存档重做日志中包含的信息。日志采集器的全关系的 SQL 接口提供了到全部数据库历史视图的直接访问，不需要强制用户修复存档重做日志文件。

本节包括下面的专题：

- 怎样使用日志采集器
- 限制
- 创建字典文件
- 确定要分析的重做日志
- 使用日志采集器
- 使用日志采集器：方案

参见 关于本节涉及的初始化参数和日志采集器视图的详细信息，参阅“Oracle8i Reference”。

关于日志采集器 PL/SQL 包的更多信息，参阅“Oracle8i Supplied PL/SQL Packages Reference”。

### 7.10.1 怎样使用日志采集器

日志采集器在识别和删除逻辑讹误时特别有用。日志采集器处理重做日志文件，将它们的内容翻译为代表对数据库执行的逻辑操作的 SQL 语句。V\$LOGMNR\_CONTENTS 视图然后列出重新构建的代表原始操作 (SQL\_REDO 列) 的 SQL 语句，并响应 SQL 语句来删除操作 (SQL\_UNDO 列)。应用 SQL\_UNDO 语句来回退对数据库的原始改变。

更进一步说，用户可以使用 V\$LOGMNR\_CONTENTS 视图来：

- 决定数据库的逻辑讹误何时开始，查明用户需要来执行不完全恢复的时间或 SCN。
- 跟踪特定表的改变。
- 跟踪由特定用户造成的改变。
- 映射数据访问方式。
- 使用存档数据进行调整和容量规划。

### 7.10.2 限制

日志采集器有下列的使用和兼容性要求。日志采集器只能：

- 在 Oracle8.1 版或之后的版本上运行。
- 分析 8.0 版本或之后版本数据库的重做日志文件。这些数据库和分析事例使用相同的数据库字符组，运行在相同的硬件平台。

**注意** 分析事例的块大小 (DB\_BLOCK\_SIZE) 必须与生成事例的日志具有相同的块大小。

如果不是这样，用户将收到一个指出存档日志被破坏的错误（而实际没有破坏）。

- 在由 PL/SQL 包创建的字典的帮助下完整地分析重做日志文件的内容。字典允许日志采集器将内部对象标志符和数据类型翻译成对象名和外部数据格式。
- 获得关于惯例表的 DML 操作的信息。它在如下情况不支持操作：
  - 索引组织的表
  - 簇表/索引
  - 非标量(non\_scalar)数据类型
  - 链接起来的行

日志采集器也不能控制直接地址插入操作，即使这样的操作记入日志。它不支持传统的通过 SQL\*Loader 进行的路径插入。

### 7.10.3 创建字典文件

在数据库装入或不装入的情况下，日志采集器都可以运行在 Oracle 事例之中。日志采集器使用一个字典文件 (dictionary file)，它是一个特殊的文件，指出创建它的数据库，也指出创建的时间。字典文件不是要求的，但它是推荐的。

没有字典文件，等价的 SQL 语句将使用 Oracle 对象名的内部对象 IDs，并以十六进制给出列的值。例如，取代 SQL 语句：

```
INSERT INTO emp(name, salary) VALUES ('John Doe', 50000);
```

日志采集器将显示：

```
insert into Object#2581(col#1, col#2) values (hextoraw(4a6f686e20446f65'),
hextoraw('c306');
```

通过装入数据库来创建字典文件，然后将字典信息收取到外部文件中。用户必须从与生成用户希望分析的日志文件的数据库相同的数据库中创建字典文件。一旦创建，用户可以使用字典文件来分析重做日志。

在创建字典时，要指定：

■ **DICTIONARY\_FILENAME** 来命名字典文件。

■ **DICTIONARY\_LOCATION** 来确定文件的位置。

**要在 Oracle8i 上创建字典文件：**

1. 通过设置初始化参数 **UTL\_FILE\_DIR**，保证确定了一个目录供 **PL/SQL** 程序使用。

例如，如下设置是为了使用 **/oracle/logs**：

```
UTL_FILE_DIR = /oracle/logs
```

如果用户不提及这个参数，程序将失败。

2. 使用 **SQL\*Plus** 来装入数据库，然后打开数据库。这个数据库的文件是用户希望分析的。例如输入：

```
STARTUP
```

3. 运行 **PL/SQL** 程序 **DBMS\_LOGMNR\_D.BUILD**。指定目录的文件名和文件的路径名。这个程序创建用户将要使用来分析日志文件的目录文件。例如，输入下面的语句来在 **/oracle/logs** 中创建 **dictionary.ora**：

```
EXECUTE DBMS_LOGMNR_D.BUILD( -  
  DICTIONARY_FILENAME => 'dictionary.ora', -  
  DICTIONARY_LOCATION => '/oracle/logs');
```

**要在 Oracle8 数据库上创建字典文件：**

虽然日志采集器只能在 8.1 版本或更高的数据库上运行，但是用户可以用它来分析 8.0 版本数据库的重做日志。

1. 使用一个 **O/S** 命令来将 **dbmslmd.sql** 脚本从 Oracle8i 数据库上的 **\$ORACLE\_HOME/rdbms/admin** 目录中拷贝到 Oracle8 数据库中相同的目录里。例如，输入：

```
% cp /8.1/oracle/rdbms/admin/dbmslmd.sql /8.0/oracle/rdbms/admin/dbmslmd.sql
```

2. 使用 **SQL\*Plus** 来装入并打开用户希望分析的文件所在的数据库。例如，输入：

```
STARTUP
```

3. 运行在 8.0 数据库上的所拷贝的 **dbmslmd.sql** 脚本，来创建 **DBMS\_LOGMNR\_D** 包。例如，输入：

```
@dbmslmd.sql
```

4. 通过设置初始化参数 **UTL\_FILE\_DIR**，确定 **PL/SQL** 包使用的目录。例如，输入：

```
UTL_FILE_DIR=/8.0/oracle/logs
```

5. 运行 **PL/SQL** 程序 **DBMS\_LOGMNR\_D.BUILD**。确定目录的文件名和文件的目录名。这个程序创建用户要用来分析日志文件的字典文件。例如，输入下面的语句在 **/8.0/oracle/logs** 中创建文件 **dictionary.ora**。

```
EXECUTE DBMS_LOGMNR_D.BUILD(  
  DICTIONARY_FILENAME => 'dicionary.ora',
```

DICTIONARY\_LOCATION=>'/8.0/oracle/logs');

#### 7. 10. 4 确定要分析的重做日志

一旦用户创建好了字典文件，用户就可以开始分析重做日志了。用户的第一步是确定用户希望用来分析的日志文件，使用 ADD\_LOGFILE 程序。使用下面的常量：

- NEW 来创建新的清单。
- ADDFILE 来向清单中添加重做日志。
- REMOVEFILE 来从清单中删除重做日志。

要使用日志采集器：

1. 使用 SQL\*Plus 来启动 Oracle 事例，数据库可以被装入，也可以不装入。例如输入：  
STARTUP
2. 通过在运行 DBMS\_LOGMNR.ADD\_LOGFILE 程序时确定 NEW 选项，创建日志清单。例如输入下面的语句来确定/oracle/logs/log1.f

```
EXECUTE DBMS_LOGMNR.ADD_LOGFILE(-  
LOGFILENAME => '/oracle/logs/log1.f', -  
OPTION => dbms_logmnr.NEW);
```

3. 如果想要的话，可通过指定 ADDFILE 选项来添加更多的日志。例如，输入下面的语句来添加/oracle/logs/log2.f:

```
EXECUTE DBMS_LOGMNR.ADD_LOGFILE(-  
LOGFILENAME => '/oracle/logs/log2.f', -  
OPTION => dbms_logmnr.ADDFILE);
```

4. 如果想要的话，可通过指定 REMOVEFILE 选项来删除日志。输入下面的语句来添加/oracle/logs/log2.f:

```
EXECUTE DBMS_LOGMNR.ADD_LOGFILE(-  
LOGFILENAME => '/oracle/logs/log2.f', -  
OPTION => dbms_logmnr.REMOVEFILE);
```

#### 7. 10. 5 使用日志采集器

一旦用户已经创建了字典文件，并指定了要分析的文件，就可以启动日志采集器并开始分析了。要缩小用户搜索的范围，在启动时可使用如下的选项：

| 这个选项         | 确定        |
|--------------|-----------|
| STARTSCN     | SCN 范围的开头 |
| ENDSCN       | SCN 范围的末尾 |
| STARTTIME    | 时间间隔的开头   |
| ENDTIME      | 时间间隔的末尾   |
| DICTFILENAME | 字典文件的名称   |

一旦用户启动了日志采集器，用户可以应用如下的字典视图进行分析：

| 这个视图                 | 显示的信息是关于    |
|----------------------|-------------|
| V\$LOGMNR_DICTIONARY | 正在使用的字典文件   |
| V\$LOGMNR_PARAMETERS | 日志采集器的当前的设置 |

|                    |               |
|--------------------|---------------|
| V\$LOGMNR_LOGS     | 将要分析哪个重做日志文件  |
| V\$LOGMNR_CONTENTS | 被分析的重做日志文件的内容 |

**要使用日志采集器:**

1. 发出 DBMS\_LOGMNR.START\_LOGMNR 程序来启动日志采集器应用工具。例如，要使用/oracle/dictionary.ora 来启动日志采集器，可发出：

```
EXECUTE DBMS_LOGMNR.START_LOGMNR( -
  DICTFILENAME =>'/oracle/dictionary.ora');
```

设置 STARTTIME 和 ENDTIME 参数根据时间来过滤数据。注意程序期望时间值：使用 TO\_DATE 函数来确定日期和时间，如下面例子中所示：

```
EXECUTE DBMS_LOGMNR.START_LOGMNR( -
  DICTFILENAME =>'/oracle/dictionary.ora', -
  STARTTIME => to_date('01-Jan-1998 08:30:00', 'DD-MON-YYYY HH:MI:SS') -
  ENDTIME => to_date('01-Jan-1998 08:45:00', 'DD-MON-YYYY HH:MI:SS'));
```

使用 STARTSCN 和 ENDSCN 参数来通过 SCN 过滤数据。如下面的例子中所示：

```
EXECUTE DBMS_LOGMNR.START_LOGMNR( -
  DICTFILENAME =>'/oracle/dictionary.ora', -
  STARTSCN =>100, -
  ENDSCN =>150);
```

2. 借助 V\$LOGMNR\_CONTENTS 表来观看输出。日志采集器返回以 SCN 顺序返回所有行，该顺序与在介质恢复中应用的顺序相同。例如，下面的查询列出关于操作的信息：

```
SELECT operation, sql_redo FROM v$logmnr_contents;
OPERATION  SQL_REDO
-----
INTERAL
INTERAL
START      set transaction read write;
UPDATE     update SYS.UNDO$ set NAME = 'RS0', USER# = 1, FILE# = 1, BLOCK# =2450, SCNBAS =
COMMIT     commit;
START      set transaction read write;
UPDATE     update SYS.UNDO$ set NAME = 'RS0', USER# = 1, FILE# = 1, BLOCK# =2450, SCNBAS =
COMMIT     commit;
START      set transaction read write;
UPDATE     update SYS.UNDO$ set NAME = 'RS0', USER# = 1, FILE# = 1, BLOCK# =2450, SCNBAS =
COMMIT     commit;
11 rows selected.
```

**分析其他数据库的存档重做日志文件** 用户可以在一个数据库的事例上运行日志采集器，但是却分析另一个不同的数据库的重做日志文件。要分析其他数据库的存档重做日志文件，日志采集器必须：

- 访问的字典文件有下面的要求：字典文件由与创建日志文件相同的数据库来创建，并由相同的数据库字符组来创建。

- 运行在与生成日志文件的平台相同的硬件平台，尽管不要求相同的系统。
- 使用的重做日志文件可以用来从 Oracle 8.0 或更高的版本进行恢复。

### 7.10.6 使用日志采集器：方案

本节包括下面的日志采集器方案：

- 跟踪用户
- 计算表访问统计

#### 跟踪用户

在这个例子中，用户对看到用户的一个使用者：JOEDEVO 在特定时间范围里所做的数据库的所有改变感兴趣。用户按下面的步骤来执行操作：

- 步骤 1：创建字典
- 步骤 2：添加日志并限制搜索范围
- 步骤 3：启动日志采集器并分析数据

**步骤 1：创建字典** 要使用日志采集器来分析 JOEDEVO 的数据，用户必须在启动日志采集器之前创建一个字典文件。

用户决定做下面的事情：

- 调用字典文件 orcldict.ora。
- 将字典放入目录/user/local/dbs。
- 将初始化参数 UTL\_FILE\_DIR 设置为/user/local/dbs。

```
# Set the initialization parameter UTL_FILE_DIR in the initialization parameter file
UTL_FILE_DIR = /user/local/dbs
```

```
# Start SQL*Plus and then connect to the database
CONNECT system/manager
```

```
# Open the database to create the dictionary file
STARTUP
```

```
# Create the dictionary file
EXECUTE DBMS_LOGMNR_D.BUILD( -
  DICTIONARY_FILENAME => 'orcldict.ora', -
  DICTIONARY_LOCATION => 'usr/local/dbs');
```

```
# The dictionary has been created and can be used later
SHUTDOWN;
```

**步骤 2：添加日志并限制搜索范围** 现在，创建数据字典。用户决定查看在特定时间中发生的改变。用户做如下的事情：

- 创建日志文件清单备用，并确定日志 loglorc1.ora。
- 将日志 log2orc1.ora 添加进清单。
- 启动日志采集器，在 1998 年 1 月 1 日的 8: 30a.m.到 8: 45a.m.范围内搜索。

```
# Start SQL*Plus, connect as SYSTEM, then start the instance
```



```
CONNECT system/manager
STARTUP NOMOUNT
```

```
# Supply the list of logfiles to the reader. The Options flag is set to indicate
# this is a new list
EXECUTE DBMS_LOGMNR.ADD_LOGFILE(OPTIONS => dbms_logmnr.NEW, -
LOGFILENAME => 'log1orc1.ora');
```

```
# ADD a file to the existing list. The Options flag is clear to indicate that
# you are adding a file to the existing list
```

```
EXECUTE DBMS_LOGMNR.ADD_LOGFILE(OPTIONS => dbms_logmnr.ADDFILE, -
LOGFILENAME => 'log2orc1.ora');
```

**步骤 3：启动日志采集器并分析数据** 在这一点上，V\$LOGMNR\_CONTENTS 表可用于查询。用户决定查找由使用者 JOEDEVO 对工资表所作的所有改变。如用户所发现，JOEDEVO 请求了两个操作，他删除了旧的工资，然后插入了一个新的工资——更高额的工资。用户现在有了需要删除这个操作的数据（并且多半证明该开除 JOEDEVO！）。

```
# Start the LogMiner . Limit the search to the specified time range.
EXECUTE DBMS_LOGMNR.START_LOGMNR( -
DICTFILENAME => 'orcdict.ora', -
STARTTIME => to_date('01-Jan-1998 08:30:00', 'DD-MON-YYYY HH:MI:SS') -
ENDTIME => to_date('01-Jan-1998 08:45:00', 'DD-MON-YYYY HH:MI:SS'));
```

```
SELECT sql_redo, sql_undo FROM v$logmnr_contents
WHERE username = 'JOEDEVO' AND tablename = 'SALARY';
```

```
# The following data is displayed (properly formatted)
```

SQL\_REDO

-----

```
delete * from SALARY
where EMPNO = 12345
and ROWID = 'AAABOOAABAAEPCABA';
```

```
insert into SALARY(NAME, EMPNO, SAL)
values('JOEDEVO', 12345, 2500)
```

SQL\_UNDO

-----

```
insert into SALARY(NAME, EMPNO, SAL)
values ('JOEDEVO', 12345, 500)
```

```
delete * from SALARY
where EMPNO = 12345
and ROWID = 'AAABOOAABAAEPCABA';
```

```
2 rows selected
```

### 计算表访问统计

由 Oracle RDBMS 生成的重做日志中包含数据库所做的所有更改的历史。采集重做日志可以生成大量信息，可以用来调整数据库。在这个例子中，用户管理一个直接销售数据库，并希望求出在 8 月份的两个星期之中，客户合同在生成收入方面有多少效益。

首先，用户启动日志采集器，并指定时间范围：

```
EXECUTE DBMS_LOGMNR.START_LOGMNR( -  
  STARTTIME => '07-Aug-98', -  
  ENDTIME => '15-Aug-98', -  
  DICTFILENAME => '/usr/local/dict.ora');
```

然后用户查询 V\$LOGMNR\_CONTENTS 来决定在用户指定的时间段中哪个表是已经更改的。

```
SELECT seg_owner, seg_name, count(*) AS Hits FROM  
v$logmnr_contents WHERE seg_name NOT LIKE '%$' GROUP BY  
seg_owner, seg_name;
```

| SEG_OWNER | SEG_NAME  | Hits  |
|-----------|-----------|-------|
| -----     | -----     | ----- |
| CUST      | ACCOUNT   | 384   |
| SCOTT     | EMP       | 12    |
| SYS       | DONOR     | 12    |
| UNIV      | DONOR     | 234   |
| UNIV      | EXECDONOR | 325   |
| UNIV      | MEGADONOR | 32    |

# 8 管理作业队列

本章描述怎样使用作业队列来调度 PL/SQL 代码的周期性运行，包括下面的专题：

- SNP 后台进程
- 管理作业队列
- 查看作业队列信息

## 8.1 SNP 后台进程

为了提高性能，多进程 Oracle 系统使用一些附加的叫做后台进程 (background processes) 的附加进程。后台进程加强了一些功能，否则这些功能将由运行每个用户程序的多重 Oracle 程序来处理。后台进程异步地执行 I/O，并监视其它 Oracle 进程来提供较高的并行性（性能更好）和可靠性。

SNP 后台进程运行作业队列。用户可以调度例程，例程是指一些 PL/SQL 代码，它们用作业队列来周期性地执行。要调度一个作业，用户将它提交到作业队列中，并确定将要运行的作业的频率。用户也可以更改、禁止、或删除用户提交的作业。

用户至少必须有一个在后台运行的 SNP 进程来执行排队作业。SNP 进程周期性地唤醒，并运行运行队列中应该运行的作业。SNP 后台进程不同于其他 Oracle 后台进程，因为 SNP 进程的失败不会造成事例的失败。如果 SNP 进程失败，Oracle 会重新启动它。

如果系统已经在限制模式下启动，SNP 后台进程不运行作业。然而，用户可以使用 ALTER SYSTEM 语句来将这个操作打开或关闭，如下所示：

```
ALTER SYSTEM ENABLE RESTRICTED SESSION;  
ALTER SYSTEM DISABLE RESTRICTED SESSION;
```

当用户启用 (ENABLE) 一个限制的会话，SNP 后台进程不运行作业；当用户 DISABLE 一个限制的会话时，SNP 后台进程运行作业。

参见 关于 SNP 后台进程的更多信息，参阅 “Oracle8i Concept”。

### 8.1.1 多个 SNP 进程

一个事例可以有多个 SNP 进程，命名为 SNP0 到 SNP9，和 SNAP 到 SNPZ。如果事例有多个 SNP 进程，运行队列作业的任务可以在这些进程之间共享以提高性能。然而，要注意，在任何时候，每个作业只能由一个进程运行。一个单一的作业不能由多个 SNP 进程同时共享。

### 8.1.2 启动 SNP 进程

作业队列初始化参数让用户可以控制 SNP 后台进程的操作。当用户在事例的初始化参

数文件中设置这些参数时，它们在用户下一次启动事例时发生作用。

**JOB\_QUEUE\_PROCESSES** 参数确定每个事例的作业队列进程的数量。不同的事例可以有不同的值。这个初始化参数可以用 **ALTER SYSTEM** 语句来动态地更改。如下面例子所示：

```
ALTER SYSTEM SET JOB_QUEUE_PROCESSES = 10;
```

**JOB\_QUEUE\_INTERVAL** 参数确定 **SNP** 进程每次唤醒之间的间隔。不同的事例有不同的值。

参见 关于这些初始化参数的描述，参阅 “Oracle8i Reference”。

## 8.2 管理作业队列

本节描述管理作业队列的各个方面，包括如下专题：

- **DBMS\_JOB** 包
- 提交作业到作业队列
- 作业怎样运行
- 从作业队列中删除作业
- 更改作业
- 损坏作业
- 强制作业运行
- 终止作业

### 8.2.1 DBMS\_JOB 包

要调度和管理作业队列中的作业，应使用 **DBMS\_JOB** 包中的程序。使用作业队列与数据库权限无关。

下面是本节涉及的 **DBMS\_JOB** 包的程序和程序注解：

| 程序        | 描述  |
|-----------|---|
| SUBMIT    | 提交一个作业到作业队列。参阅 8.2.2 节 “提交作业到作业队列”。                                       |
| REMOVE    | 从作业队列中删除一个特定的作业。参阅 8.2.4 节 “从作业队列中删除作业”                                   |
| CHANGE    | 更改一个已经提交到作业队列中的一个特定作业。用户可以更改作业描述、作业将要运行的时间、或者在作业运行之间的间隔。参阅 8.2.5 节 “更改作业” |
| WHAT      | 更改特定作业的作业描述。参阅 8.2.5 节 “更改作业”   |
| NEXT_DATE | 更改特定作业的下一次运行时间。参阅 8.2.5 节 “更改作业”  |
| INTERVAL  | 更改特定作业运行之间的间隔。参阅 8.2.5 节 “更改作业”   |
| BROKEN    | 禁止作业运行。如果作业被标志为损坏，Oracle 将不会再试图运行它。参阅 8.2.6 节的 “损坏作业”                     |
| RUN       | 强制特定的作业运行。参阅 8.2.7 节 “强制作业运行”   |

参见 关于 **DBMS\_LOG** 包的语法信息可以在 “Oracle8i Supplied PL/SQL Package

Reference” 中找到。

关于在 Oracle 并行服务器环境中的 DBMS\_JOB 包的使用，参阅“Oracle8i Parallel Server Administration, Deployment, and Performance”，书中还提供其他选项。

### 8.2.2 提交作业到作业队列

要将一个新的作业提交到作业队列，应使用 DBMS\_JOB 包中的 SUBMIT 程序。用户可使用 SUBMIT 参数确定下面的参数：

| 参数        | 描述   |
|-----------|--|
| JOB       | 一个输出参数，这是分配到用户所创建作业的一个标志符。无论什么时候用户希望更改或删除作业，用户必须使用这个作业号。参阅 8.2.2 节中的“作业号”  |
| WHAT      | 这是用户希望拥有来运行的 PL/SQL 代码。参阅 8.2.2 节中的“作业定义”  |
| NEXT_DATE | 这是运行作业的下一个日期。缺省值是 SYSDATE  |
| INTERVAL  | 这是一个日期函数，计算下一次运行作业的时间。缺省值是 NULL。INTERVAL 必须及时估计未来的点，或者为空。参阅 8.2.2 节中的“作业运行间隔”  |
| NO_PARSE  | 这是一个标记。如果 NO_PARSE 设置为 FALSE（缺省），Oracle 分析与作业相关的过程。如果 NO_PARSE 设置为 TRUE，Oracle 分析与第一次作业运行时的作业。例如，如果用户希望在用户创建与作业相关的表之前提交作业，则将 NO_PARSE 设置为 TRUE |

作为一个例子，让我们提交一个新的作业到作业队列。作业调用过程 DBMS\_DDL.ANALYZE\_OBJECT 来产生表 DQUON.ACCOUNTS 的优化器统计。这个统计建立在 ACCOUNTS 表的一半行之上。作业每 24 小时运行一次。

```
VARIABLE jobno number;
BEGIN
  DBMS_JOB.SUBMIT(:jobno,
    'dbms_ddl.analyze_object(''TABLE'',
    ''DQUON'', ''ACCOUNTS'',
    ''ESTIMATE'', NULL, 50);',
    SYSDATE, 'SYSDATE + 1');
  COMMIT;
END;
/
Statement processed.
PRINT jobno
JOBNO
-----
14144
```

#### 作业环境

当用户将作业提交到作业队列，或更改作业的定义时，Oracle 记录下面的环境特征：

- 当前的用户
- 提交作业或更改作业的用户
- 当前的模式（如果已经发出了 `ALTER SESSION SET CURRENT_SCHEMA` 语句，可能与提交用户中的当前用户的不同）
- MAC 权限（如果适当）

Oracle 也记录下面的 NLS 参数：

- NLS\_LANGUAGE
- NLS\_TERRITORY
- NLS\_CURRENCY
- NLS\_ISO\_CURRENCY
- NLS\_NUMERIC\_CHARACTERS
- NLS\_DATE\_FORMAT
- NLS\_DATE\_LANGUAGE
- NLS\_SORT

Oracle 在每次作业运行的时间恢复这些环境特征。在没有指定 NLS 参数时，NLS\_LANGUAGE 和 NLS\_TERRITORY 参数是缺省的。

用户可以改变作业的环境，使用 DBMS\_SQL 包和 ALTER SESSION 语句。

参见 关于 DBMS-SQL 更多的信息，参阅 “Oracle8i Supplied PL/SQL Packages Reference”。

关于改变作业环境的 ALTER SESSION 语句的使用，参阅 “Oracle8i SQL Reference”。

### 作业和导入/导出

作业可以导入和导出。这样，如果用户在一个数据库中定义一个作业，用户可以将它传输到另一个数据库中。当导出和导入作业时，作业的号、环境和定义保持不变。

**注意** 如果用户希望导出作业的作业号与数据库中已经存在的作业的号相匹配，用户将不允许导出那个作业。作为数据库中的新作业来提交作业。

### 作业持有者

当用户提交作业到作业队列中时，Oracle 识别用户为作业的持有者。只有作业的持有者可以更改作业、强制作业运行、或从作业队列中删除作业。

### 作业号

队列的作业由它的队列号来识别。当用户提交一个作业时，它的作业号自动的从序列 SYS.JOBSEQ 中生成。

一旦作业赋予了作业号，这个号将不再改变。即使作业被导出或导入，它的作业号保持不变。

### 作业定义

作业定义是在 SUBMIT 过程中的 WHAT 参数里指定的 PL/SQL 代码。

正常情况下，作业定义是对一个过程的简单调用。过程调用可以有任意参数号。

**注意** 在作业定义中，使用两个单引号来前后包容字符串。总是将分号包括在作业定义的结尾。

Oracle 在作业定义中识别特殊的参数值，下面给予列出：

| 参数        | 模式     | 描述                       |
|-----------|--------|--------------------------|
| JOB       | IN     | 当前作业的号                   |
| NEXT_DATE | IN/OUT | 作业下次运行的日期。缺省值是 SYSDATE   |
| BROKEN    | IN/OUT | 作业的状态，损坏或未损坏。IN 值是 FALSE |

下面是一个有效的作业定义的例子：

```
'myproc('10-JAN-99', next_date, broken);'  
'scott.emppackage.give_raise('JFEE',3000.00);'  
'dbms_job.remove(job);'
```

**作业运行间隔**

INTERVAL 日期函数在作业运行之前立即求解。如果作业成功地完成，根据 INTERVAL 中计算的日期变成新的 NEXT\_DATE。如果 INTERVAL 日期函数求解为 NULL，且作业成功地完成，作业从队列中删除。

如果作业应该以一组间隔来周期性地运行，在 INTERVAL 参数中使用类似于 'SYSDATE+7' 的日期表达式。例如，如果用户设置运行间隔为星期一的 'SYSDATE+7'，但是由于某种原因（例如网络失败），作业直到星期三才运行，那么 'SYSDATE+7' 计算出每个星期三再运行，而不是星期一。

如果用户希望在特定的时间总是自动地运行某个作业，而不管上次运行的时间（例如，每个星期一），INTERVAL 和 NEXT\_DATE 参数应该指定日期表达式为类似 'NEXT\_DATE DAY(TRUNC(SYSDATE), 'MONDAY')'。

下面表示在作业运行间隔使用的一些通用的日期表达式：

| 日期表达式   | 估计           |
|---|--------------|
| 'SYSDATE + 7'   | 从上次运行恰好 7 天  |
| 'SYSDATE + 1/48'  | 每半个小时        |
| 'NEXT_DAY(TRUNC(SYSDATE), 'MONDAY') + 15/24'                | 每个星期一的下午 3 点 |
| 'NEXT_DAY(ADD_MONTHS(TRUNC(SYSDATE), 'Q'), 3), 'THURSDAY')' | 每个季度的第一个星期四  |

**注意** 当确定 NEXT\_DATE 或 INTERVAL 时，记住，日期和字符串必须包在单引号中。INTERVAL 的值必须也括在单引号中。

**数据库连接和作业**

如果用户提交一个使用数据库连接的作业，则连接必须包括用户名和密码。匿名的数据库连接是不会成功的。

**8.2.3 作业怎样运行**

SNP 后台进程运行作业。要运行一个作业，过程要创建一个运行作业的会话。

当 SNP 进程运行作业时，作业是在与作业提交相同的环境和持有者缺省的权限下运行的。

当用户强制作业用程序 DBMS\_JOB.RUN 来运行时，作业是由用户的使用者程序来运行的。当用户的使用者程序运行作业时，仅使用用户的缺省权限。通过角色来授予用户的权限是不可用的。

### 作业队列锁

Oracle 使用作业队列锁来保证每次作业由一个会话来运行。当作业运行时，它的会话获得那个队列的一个作业队列（JQ）锁。用户可以使用数据字典中的锁定视图来检查关于被会话控制的当前锁的信息。

下面的查询列出持有 JQ 锁的所有会话的会话标志符、锁类型、和锁的标志符：

```
SELECT sid, type, id1, id2
FROM v$lock
WHERE type = 'JQ';
```

| SID | TY | ID1 | ID2   |
|-----|----|-----|-------|
| 12  | JQ | 0   | 14144 |

1 row selected.

在上述的查询中，持有锁的会话的标志符为 12。对于 JQ 锁，ID1 锁标志符总是为 0。ID2 锁标志符是正在运行会话的作业的作业号。这个视图可以用 DBA\_JOBS\_RUNNING 视图来连接，以获得关于作业的更多的信息。参阅 8.3 节“观看作业队列信息”以获得更多关于视图的信息。

### 作业运行错误

当作业失败时，关于失败的信息记录在一个踪迹文件和报警日志中。Oracle 写信息号 ORA-12012，包括失败作业的作业号。

下面可以防止成功的排队作业运行：

- 没有任何 SNP 后台进程来运行任务。
- 网络或事例失败。
- 在运行作业时的异常事件。

在 Oracle 试图运行作业时，如果作业返回一个错误，Oracle 将再次试图来运行它。第一次尝试是在一分钟之后，第二次尝试是在两分钟之后，第三次尝试是在四分钟之后，依此类推，通过加倍间隔每次尝试之间的间隔来进行。当重试间隔超过了运行间隔时，Oracle 以正常的运行间隔来继续重试。然而，如果作业失败了 16 次，Oracle 自动将作业标记为损坏，不再试图运行它了。

**注意** 如果有 SNP 进程，除非运行所有准备好的作业的时间超过了作业 J 的作业间隔，作业 J 将在每个 JOB\_QUEUE\_INTERVAL 最多运行一次。用户不会看到上面所述的指数补偿。

例如，如果 JOB\_QUEUE\_INTERVAL = 600 (10 分钟)，但是运行任务的间隔指定



为 3 分钟。没有观察到指数的补偿，因为直到 SNP 进程达到 10 分钟的间隔，否则它不会再次“唤醒”来重试任务。实际上，作业将每十分钟运行一次，直到第 16 次尝试为止。

由于指数补偿被作业 J 限制，如果作业 J 的间隔小于或等于一分钟时，用户也将不会看到任何的指数补偿。

这样，如果用户可以纠正防止作业在作业失败 16 次之前运行这一问题，Oracle 将最后再次运行那个作业。

参见 关于锁定视图的更多信息，参阅“Oracle8i Reference”。

关于锁定的更多信息，参阅“Oracle8i Concepts”。

#### 8.2.4 从作业队列中删除作业

要从作业队列中删除一个作业，使用 DBMS\_JOB 包中的 REMOVE 程序。

下面的语句从作业队列中删除作业号 14144：

```
DBMS_JOB.REMOVE(14144);
```

**限制：**

- 用户可以从作业队列中删除当前运行的作业。然而，作业不会被中断，当前的运行会完成。
- 用户只可以删除属于自己的作业。如果用户试图删除不属于自己的作业，则会收到一个声明作业不在作业队列中的一条消息。

#### 8.2.5 更改作业

要更改一个用户已经提交到作业队列中的作业，使用 DBMS\_JOB 包中的程序 CHANGE、WHAT、NEXT\_DATE、或 INTERVAL。

**限制：**

- 用户只可以更改属于用户自己的作业。如果用户视图更改不属于自己的作业，则会收到一个声明作业不在作业队列中的一条消息。

##### CHANGE

用户可以更改任何一个与作业相关的可由用户定义的参数，这通过调用 DBMS\_JOB.CHANGE 程序来完成。

在这个例子中，标志为 14144 的作业现在每三天运行一次：

```
DBMS_JOB.CHANGE(14144, null, null, 'SYSDATE + 3');
```

如果用户在调用程序 CHANGE 时，指定 WHAT、NEXT\_DATE、或 INTERVAL 为 NULL，则当前的值保持不变。

**注意** 用户可以使用程序 CHANGE 中的 WHAT 参数来改变作业的定义，Oracle 记录用户当前的环境。这成为作业的新环境。

##### WHAT

用户通过调用 DBMS\_JOB.WHAT 程序可以更改作业的定义。下面的例子更改标志为

14144 作业的定义:

```
DBMS_JOB.WHAT(14144, 'scott.emppackage.give_raise(''RBAYLIS'',6000.00);'
```

**注意** 当用户执行程序 WHAT 时, Oracle 记录当前的作业环境。这成为作业的新环境。

#### NEXT\_DATE

用户可以通过调用 DBMS\_JOB.NEXT\_DATE 过程来改变 Oracle 执行作业的下一次时间, 下面是一个例子:

```
DBMS_JOB.NEXT_DATE(14144, 'SYSDATE+1');
```

#### INTERVAL

用户通过调用 DBMS\_JOB.NEXT\_DATE 程序可以更改 Oracle 运行作业的下次日期。如下面的例子所示:

```
DBMS_JOB.NEXT_DATE(14144, 'NULL');
```

在这个情况, 作业在成功运行之后将不会再次运行。

### 8.2.6 损坏的作业

作业可以标志为损坏或未损坏, Oracle 不会试图运行损坏的作业。然而, 用户可以强制一个损坏的作业运行。这要通过调用 DBMS\_JOB.RUN 程序来进行。

#### 作业是怎样变得损坏的

当用户提交作业时, 作业被认为未损坏。

有两种途径使作业损坏:

- Oracle 在经过 16 次尝试未能成功的运行那个任务。
- 用户已经使用程序 DBMS\_JOB.BROKEN 将作业标志为损坏:

```
DBMS_JOB.BROKEN(14144, TRUE)
```

一旦作业被标志为损坏, Oracle 将不会再试图运行这个作业, 一直到用户将作业标志为未损坏, 或者通过调用程序 DBMS\_JOB.RUN 来强制作业运行时为止。

下面的例子标志作业 14144 为未损坏, 并设置下次运行日志为即将到来的星期一:

```
DBMS_JOB.BROKEN(14144, FALSE, NEXT_DAY(SYSDATE, 'MONDAY'));
```

#### 限制:

- 用户仅可以将属于自己的作业标志为损坏。如果用户试图标志一个不属于自己的作业, 则会收到一个声明作业不在作业队列中的一条消息。

#### 运行损坏任务

如果问题已经使一个作业失败了 16 次, Oracle 将作业标志为损坏。一旦用户解决了这个问题, 用户可以通过下面的方法之一来运行作业。

- 通过调用 DBMS\_JOB.RUN 来强制作业运行
  - 通过调用 DBMS\_JOB.BROKEN 来标志作业为未损坏, 并等待 Oracle 来运行作业
- 如果用户通过调用过程 DBMS\_JOB.RUN 来强制作业运行, Oracle 会立即运行这个作

业。如果作业成功，那么 Oracle 将作业标记为未损坏，并重新设置作业失败运行的数量的计数。

一旦用户重新设置作业的损坏标志（通过调用 RUN 或 BROKEN），根据为作业设置的运行间隔，作业运行重新开始。

### 8.2.7 强制作业运行

用户想要手工运行任务的时候可能会有几次。例如，如果用户已经修好了一个损坏的作业，用户可能希望立刻强制它运行来测试这个作业。要强制一个作业立即运行，使用 DBMS\_JOB 包中的程序 RUN。

当用户使用 DBMS\_JOB.RUN 运行一个作业时，Oracle 重新计算下次运行日期。例如，如果用户在星期一那天，用 NEXT\_DATE 为“SYSDATE”值和 INTERVAL 为“SYSDATE+7”值创建的作业，则作业将在每隔 7 天，在星期一启动。然而，如果用户在星期三来运行 RUN，则下次运行的时间将是下个星期三。

下面的语句运行用户会话中的作业 14144，并重新计算下次运行日期：

```
DBMS_JOB.RUN(14144);
```

**注意** 当用户强制作业运行时，作业在用户当前会话中运行。作业的运行重新初始化用户会话的包。

**限制：**

- 用户只可以运行属于自己的作业。如果用户试图运行一个不属于自己的作业，则会收到一个声明作业不在作业队列中的一条消息。
- 程序 RUN 包含一个隐含的提交。一旦用户使用 RUN 运行一个作业，用户不能回退。

### 8.2.8 结束一个作业

用户可以通过标志作业为损坏来结束一个运行的作业，识别运行作业的会话，并断开那个会话。用户应该标志作业为损坏，这样，Oracle 将不会再试图运行这个作业。

在用户识别了运行作业的会话之后（借助 V\$SESSION），用户可以使用 SQL 语句 ALTER SYSTEM 来断开会话。

关于作业和会话信息的查看的例子，参阅下一节“查看作业队列信息”。

参见 关于 V\$SESSION 的更多信息，参阅“Oracle8i Reference”。

## 8.3 查看作业队列信息

用户可以查看队列中作业的信息，借助于如下所列的数据字典视图：

| 视图               | 描述   |
|------------------|--|
| DBA_JOBS         | 列出数据库中所有的作业                                  |
| USER_JOBS        | 列出用户拥有的所有作业                                  |
| DBA_JOBS_RUNNING | 列出当前运行数据库中的所有的作业。这个视图可以加入 V\$LOCK 来识别已经锁定的作业 |

例如，用户可以列出关于作业状态和失败运行的信息。下面的示例查询为用户已经提交的每一个作业创建一个作业号、下次运行时间、失败和损坏状态的列表：

```
SELECT job, next_date, next_sec, failure, broken
```

```
FROM user_jobs;
```

| JOB   | NEXT_DATE | NEXT_SRC | FAILURES | B    |
|-------|-----------|----------|----------|------|
| 9125  | 01-NOV-98 |          | 00:00:00 | 4 N  |
| 14144 | 24-OCT-99 |          | 16:35:35 | 0 N  |
| 41762 | 01-JAN-00 |          | 00:00:00 | 16 Y |

3 rows selected.

用户也可以列出关于当前运行作业的信息。下面的示例查询列出了会话标志符、作业号、提交作业的用户，和当前运行作业的启动时间：

```
SELECT sid, r.job, log_user, r.this_date, r.this_sec
```

```
FROM dba_jobs_running r, dba_jobs j
```

```
WHERE r.job = j.job;
```

| SID | JOB   | LOG_USER | THIS_DATE | THIS_SEC |
|-----|-------|----------|-----------|----------|
| 12  | 14144 | JFEE     | 24-OCT-94 | 17:21:24 |
| 25  | 8536  | SCOTT    | 24-OCT-94 | 16:45:12 |

2 rows selected.

参见 关于数据字典视图的更多信息，参阅 “Oracle8i Reference”。

## 第三部分 数据库存储

---

第三部分讲述基本的数据库构造，它支持数据库对象的创建，并保持事务完整。包括下列 3 章：

- 第 9 章，表空间管理。
- 第 10 章，数据文件管理。
- 第 11 章，回退数据段管理。

# 9

## 表空间管理

本章讲述表空间管理的各个方面，包括下列主题：

- 表空间管理的指南
- 创建表空间
- 管理表空间的地址分配
- 改变表空间的利用率
- 只读表空间
- 删除表空间
- 使用 DBMS\_SPACE\_ADMIN 程序包
- 在数据库之间移动表空间
- 查看表空间信息

### 9.1 表空间管理的指南

在使用 Oracle 数据库的表空间之前，用户应熟悉下列各节中的指南：

- 使用多重表空间
- 指定表空间存储参数
- 为用户分配表空间定额

参见 “Oracle8i Concepts” 以获取有关数据库结构、空间管理、表空间、数据文件的完整论述。

#### 9.1.1 使用多重表空间

使用多重表空间，可使用户在执行数据库运算的过程中更加灵活。例如，用户可以执行下列操作：

- 把用户数据从数据字典的数据中分开，以减少字典对象和模式对象对于相同数据

文件的争用。

- 把某一应用程序的数据和其它应用程序的数据分隔开；这样，如果必须使表空间脱机时，可以防止多个应用程序受到影响。
- 在不同的磁盘存储表空间的数据文件，以减少 I/O 争用。
- 把回退数据从用户数据中分开，防止单个磁盘的故障导致数据的永久性丢失。
- 在其他表空间保持联机的时候，可使个别表空间脱机。这样可以提供更好的综合工作效率。
- 为特定类型数据库的用法保存表空间，比如频繁的更新操作、只读操作或临时数据段的存储，这样用户可以优化表空间的使用。
- 备份个别表空间。

一些操作系统限定了可以同时打开的文件数量，这些限定影响了可以同时联机的表空间的数量。为了避免超出用户操作系统的限定，用户需要高效地设计表空间：仅创建满足用户要求的表空间，并且用尽可能少的文件创建这些表空间。如果用户需要增加表空间的容量，应加入一两个大型数据文件，或是在 `autoextend` 选项设置为 `on` 的情况下创建数据文件而不是许多小数据文件。

用户应根据这些因素分析数据，并判断数据库需要多少表空间。

### 9.1.2 指定表空间存储参数

当用户创建新的表空间时，用户可以为表空间中创建的数据对象指定缺省的存储参数。如果数据对象创建时指定了存储参数，该参数将取代包含数据对象的表空间的缺省存储参数。如果在创建新的数据对象时，用户没有指定存储参数，那么该对象的数据段将自动使用表空间的缺省存储参数。

设置表空间的缺省存储参数，以便估算其中典型数据对象的尺寸(需要用户估算这个尺寸)。当用户创建特殊的数据对象时，可以为其指定不同的存储参数。用户还可以在以后改变缺省存储参数。

**注意** 如果用户没有为新的表空间指定存储参数，用户操作系统中 Oracle 的缺省存储参数将成为该表空间的缺省存储参数。

在 9.3 节“管理表空间的地址分配”中有关于存储参数的详细论述。

### 9.1.3 为用户分配表空间定额

对创建数据表、簇、快照、索引和其它数据对象的用户授权，使他们可以创建数据对象，以及保存该数据对象数据段的表空间定额(空间容限或者范围)。安全管理员负责授予与数据库用户创建数据对象的权限，并在必要的时候为数据库用户分配表空间定额。

了解更多有关数据库用户分配表空间定额的内容，参见第 22 章“分配表空间定额”中的内容。

## 9.2 创建表空间

在创建一个表空间之前，用户必须创建一个数据库以包含它。任何数据库中的第一个表空间总是系统表空间，并且在数据库创建期间，任何数据库的第一个数据文件都将自动定位

在系统表空间里。创建数据库的内容在第 2 章有所论述。

创建表空间的步骤与操作系统相关。就一般情况而论，用户应该是通过操作系统创建一个目录结构以放置用户的数据文件。在大部分操作系统中，当用户创建一个新的表空间，或者因增加数据文件而改变一个表空间时，要指明其大小，并给出完整文件名。在各种情况下，Oracle 都将按照给定的条件自动地定位并格式化数据文件。然而，在一些操作系统中，用户必须在安装之前创建数据文件。

**注意** 在当前请求至少获得两个回退数据段(包括 SYSTEM 回退数据段)之前，没有数据能够插入任何表空间。回退数据段在第 11 章“管理回退段”中有所论述。

为了创建新的表空间，使用 SQL 语句 CREATE TABLESPACE 或 CREATE TEMPORARY TABLESPACE。用户创建表空间必须具有 CREATE TABLESPACE 的系统权限。以后，用户改变表空间可以使用 ALTER TABLESPACE 或 ALTER DATABASE 语句。用户必须具有 ALTER TABLESPACE 或 ALTER DATABASE 的系统权限。

在 Oracle8i 以前的版本中，全部表空间都是以字典管理类型(dictionary-managed)创建的。字典管理表空间依靠 SQL 字典表跟踪磁盘空间的利用。而从 Oracle8i 开始，用户可以创建本地管理(locally managed)表空间，它使用位图(而不是 SQL 字典列表)来跟踪已使用的和自由的空间。为与较早的版本兼容，字典管理类型被作为表空间的缺省类型保留，但是 Oracle 建议用户现在使用本地管理表空间。

用户还可以创建临时的表空间，它可以是字典管理类型或本地管理类型。各类型的表空间分别在下列章节中讨论：

- 字典管理表空间
- 本地管理表空间
- 临时表空间

**参见** 供用户操作系统使用的 Oracle 安装文件，以获取有关安装时创建的表空间的信息。

有关 CREATE TABLESPACE、CREATE TEMPORARY TABLESPACE、ALTER TABLESPACE 语法和用法的更多信息，参见“Oracle8i SQL Reference”。

### 9.2.1 字典管理表空间

由于向后兼容性的关系，字典管理保留为缺省的表空间空间管理方式。无论什么时候某一区域被分配，或为了再使用而释放，Oracle 都会更新数据字典中适当的条目。

#### 创建一个字典管理表空间

作为一个例子，让我们创建一个带有下面特性的表空间--TBSA：

- 新的表空间的数据包含在单个数据文件中，其容量为 50M。
- 明确地设置在了此表空间中创建的数据段的缺省存储参数。

下面的语句用于创建表空间 TBSA：

```
CREATE TABLESPACE tbsa
  DATAFILE 'u02/oracle/data/tbsa01.dbf' SIZE 50M
  DEFAULT STORAGE (
```

```
INITIAL 50K
NEXT 50K
MINEXTENTS 2
MAXEXTENTS 50
PCTINCREASE 0);
```

**注意** 如果用户没有完全指定数据文件的文件名，Oracle 将根据用户的操作系统，在缺省数据库目录或当前目录中创建数据文件。Oracle 建议用户一定要指定完整定义的名称。

### 改变字典管理表空间

使用 ALTER TABLESPACE 语句的理由包括以下各方面，但不仅仅是这些方面：

- 改变缺省存储参数。参见 9.3.2 节中“改变表空间存储设置”部分。
- 合并表空间中的自由空间。参见 9.3.3 节“合并字典管理表空间中的自由空间”。
- 改变表空间的工作效率(ONLINE/OFFLINE)。参见 9.4 节“改变表空间工作效率”。
- 使表空间只读或可读写。参见 9.5 节“只读表空间”。
- 增加或重新命名数据文件，或允许/禁止表空间中数据文件容量的自动扩展。参见第 10 章“管理数据文件”。

### 9.2.2 本地管理表空间

本地管理表空间使用位图跟踪表空间自己全部区域的信息，这将产生下列好处：

- 改进了并行操作和空间操作速度，因为空间分配和重新分配主要改变本地管理资源(存储在头部文件中的位图)，而不需要诸如队列等集中管理资源。
- 改进了操作性能，因为消除了递归操作，而在字典管理的空间分配期间往往要求这一操作。
- 允许可读的备用数据库，因为本地管理的临时表空间(用于排序，等等)是在本地管理的，不生成任何回退或重演。
- 简化空间分配。当指定 AUTOALLOCATE 子句时，将自动选择适当的区域容量。
- 减少用户对数据字典的依赖，因为必要的信息被储存在文件头部和位图区段中。

另外，DBMS\_SPACE\_ADMIN 程序包——9.7 节“使用 DBMS\_SPACE\_ADMIN 程序包”中有所论述——提供了对于本地管理表空间的维护过程。

### 创建本地管理表空间

为了创建本地管理表空间，用户需要指定 CREATE TABLESPACE 语句中区域管理子句的 LOCAL 项。那么用户将有两个选项。用户可以用 AUTOALLOCATE 选项让 Oracle 自动管理区域，或者规定用指定容量的统一区域管理表空间(UNIFORM SIZE)。

如果表空间预定包含容量各不相同的对象，这些对象需要许多区域，并且区域容量各不相同，那么 AUTOALLOCATE 是最好的选择。如果拥有针对空间分配和重分配的控制权对用户来说并不重要，则 AUTOALLOCATE 为用户提供了一种管理表空间的简化方法。Oracle 管理用户空间可能会有一些空间浪费，但是它所带来的效益很可能比这些浪费的空间更有价值。



从另一方面来说,如果用户需要严格控制不用的空间,并且能够准确地预知分配给数据对象的空间以及区域的数量和容量,那么 **UNIFORM** 是一个好的选择。这样可以保证在用户表空间中不会有无用的空间。

下面的语句创建一个名为 **LMTBSB** 的本地管理表空间,语句中 **AUTOALLOCATE** 使 Oracle 自动地管理区域容量。

```
CREATE TABLESPACE lmtbsb DATAFILE 'u02/oracle/data/lmtbsb01.dbf' SIZE 50M
EXTENT MANAGMENT LOCAL AUTOALLOCATE ;
```

另外,也可以指定 **UNIFORM** 子句创建这个表空间。例句中,指定了 128K 的区域容量。各 128K 区域(相当于 64 个 Oracle 存储块)被描述为文件位图中的位。

```
CREATE TABLESPACE lmtbsb DATAFILE 'u02/oracle/data/lmtbsb01.dbf' SIZE 50M
EXTENT MANAGE MENT LOCAL UNIFORM SIZE 128K;
```

用户不能创建一个本地管理的系统表空间。

**注意** 当用户为一个本地管理表空间分配一个数据文件时,用户应该考虑到用于空间管理的元数据空间(区域位图或者空间标题段),它是用户空间的一部分。例如:如果指定 **UNIFORM** 的时候,用户在区域管理子句中没有指定 **SIZE** 参数,缺省区域容量是 1MB。那么,为数据文件指定的尺寸必定大于 1MB(至少一个存储块加位图容量)。

#### 改变本地管理表空间

用户可以因为许多与字典管理表空间一样的理由而改变一个本地管理表空间。然而,存储参数的改变不是随意的,并且对于本地管理表空间不必要合并自由区域。用户也不能将一个本地管理表空间改变为本地管理的临时表空间。

### 9.2.3 临时表空间

如果用户希望改进多重排序操作的并行性,减少它们的总开销,或者完全避免 Oracle 空间管理操作,用户可以创建临时表空间。

在临时的表空间内,对于一个给定的事例和表空间,所有的排序操作共用一个单一的排序数据段。排序数据段是因给定表空间内执行分类操作的每个事例而存在的。在使用临时表空间排序的第一个语句启动后,排序数据段即可创建,而只有在停止工作的情况下才会释放。一个区域不能被多个事务共享。

通过 **V\$SORT SEGMENT**,用户可以查看临时表空间排序数据段的空间地址分配和重分配,而 **V\$SORT USAGE** 则标识在那些数据段中当前的排序使用者。

用户不能在临时表空间里显式地创建对象。为用户分配临时表空间将在第 22 章“管理用户和资源”中进行论述。

参见 关于 **V\$SORT SEGMENT** 和 **V\$SORT USAGE** 的更多信息,请参见“Oracle8i Reference”。

#### 创建字典管理表空间

要在表空间创建期间标识其为临时的表空间,可在 **CREATE TABLESPACE** 语句中指定

TEMPORARY 关键字。下面的语句创建了临时字典管理表空间。

```
CREATE TABLESPACE sort
  DATAFILE 'u02/oracle/data/sort01.dbf' SIZE 50M
  DEFAULT STORAGE (
    INITIAL 2M
    NEXT 2M
    MINEXTENTS 1
    PCTINCREASE 0)
  TEMPORARY;
```

为了将一个现存的永久性字典管理表空间更改为一个临时的表空间，使用 ALTER TABLESPACE 语句。例如：

```
ALTERR TABLESPACE tbsa TEMPORARY;
```

用户针对字典管理的临时表空间发出 ALTER TABLESPACE 指令，可以使用若干关键字和子句，它们与永久性字典管理表空间中使用的关键字和子句是一样的。任何有关的限制均在“Oracle8i SQL Reference”中论述。

**注意** 用户可以使字典管理临时表空间脱机。将它们恢复联机后不会影响它们的临时性状态。

#### 创建本地管理临时表空间

因为在本地管理表空间中，空间管理非常简单，并且更有效，所以更适合于临时表空间。本地管理临时表空间使用临时文件，它不会修改临时表空间以外的数据，或是为临时表空间数据生成重演数据。因此，它们可用于备用的或者只读数据库。

用户查看临时文件信息所使用的视图与查看数据文件的视图也不同。V\$TEMPFILE 和 DBA\_TEMP\_FILES 类似于 V\$DATAFILE 和 DBA\_DATA\_FILES。参见 9.9 节“查看表空间信息”以获取关于表空间视图的摘要信息。

为了创建本地管理临时表空间，用户需要使用 CREATE TEMPORARY TABLESPACE 语句，它要求用户具有 CREATE TABLESPACE 的系统权限。

下面的语句创建了一个临时表空间，它的每个区域是 16M。每个 16M 区域(相当于 8000 个 Oracle 存储块)被描述为文件位图中的位。

```
CREATE TEMPORARY TABLESPACE ltemp TEMPFILE 'u02/oracle/data/ltemp01.dbf'
  SIZE 20M REUSE
  EXTENI MANAGEMENT LOCAL UNIFORM SIZE 16M;
```

除增加临时文件之外——如同下列例子中说明的——用户不能对于一个本地管理临时表空间使用 ALTER TABLESPACE 语句。

```
ALTER TABLESPACE ltemp
  ADD TEMPFILE 'u02/oracle/data/ltemp02.dbf' SIZE 2M REUSE;
```

**注意** 用户不能使用带有 TEMPORARY 关键字的 ALTER TABLESPACE 语句将本地管理的永久性表空间改变为本地管理的临时表空间。用户必须使用 CREATE TEMPORARY TABLESPACE 语句来创建一个本地管理的临时表空间。

然而，ALTER DATABASE 语句可用于改变临时文件。下面的语句使临时文件脱机和联

机:

```
ALTER DATABASE TEMPFILE '%u02/oracle/data/lmtemp02.dbf' OFFLINE;  
ALTER DATABASE TEMPFILE '%u02/oracle/data/lmtemp02.abf' ONLINE;
```

下面的语句调整临时文件的大小:

```
u02/oracle/data/lmtemp02.dbf to 4M:  
ALTER DATABASE TEMPFILE '%u02/oracle/data/lmtemp02.dbf' RESIZE 4M;
```

下面的语句删除一个临时文件:

```
ALTER DATAEASE TEMPFILE '%u02/oracle/data/lmtemp02.abf' DROP;
```

也可以对于一个临时文件执行 `AUTOEXTEND` 和重命名(`RENAME FILE`)操作。但在这里不做演示。

## 9.3 管理表空间的地址分配

当用户创建一个表空间时,用户确定由哪些实际的数据文件构成表空间,以及对于字典管理表空间,其缺省存储特性是什么。这两种表空间的属性都可以在以后进行改变。表空间的缺省存储特性在这里进行论述;数据文件的管理是第 10 章“管理数据文件”的主题。

将来,字典管理表空间中的自由空间可能成为碎片,使配置新的区域变得更困难。消除磁盘碎块的方法在下面进行论述。

其中的章节如下:

- 本地管理表空间的存储参数
- 字典管理表空间的存储参数
- 合并字典管理表空间中的自由空间

### 9.3.1 本地管理表空间的存储参数

当用户配置一个本地管理表空间时,用户不能指定缺省存储参数或者最小区域容量。如果指定了 `AUTOALLOCATE`,表空间是用最小的区域容量 `64K` 进行系统管理的。如果指定 `UNIFORM SIZE`,那么表空间是用指定的统一区域容量管理的且缺省大小是 `1M`。

当用户在一个本地管理表空间中分配数据段时,存储子句将不同于在字典管理表空间中的说明。当在一个本地地管理表空间中创建一个对象时,Oracle 使用它们的 `INITIAL`, `NEXT`, 以及 `MINEXTENTS` 参数来计算此对象数据段最初的容量。

### 9.3.2 典管理表空间的存储参数

存储参数影响两个方面:访问存储在数据库中的数据需要的时间,数据库中空间的使用效率。

参见 关于这些参数作用的论述,请参见“Oracle8i Designing and Tuning for Performance”。

关于存储参数的完整叙述,请参见“Oracle8i SQL Reference”。

### 改变表空间的存储参数

下面的参数影响数据段在一个表空间里的存储分配。

它们作为存储参数引用，并且包含在 **CREATE TABLESPACE** 语句的存储子句中。

|                    |   |
|--------------------|---|
| <b>INITIAL</b>     | 以字节(K 或者 M)定义数据段里第一个区域的尺寸。                  |
| <b>NEXT</b>        | 定义第二区域字节的大小 (K 或 M) 。                       |
| <b>PCTINCREASE</b> | 第二区域后面的各个区域的增长率。                            |
| <b>MINEXTENTS</b>  | 表空间中数据段最初创建时分配的区域数目。                        |
| <b>MAXEXTENTS</b>  | 确定数据段可以占用的区域数量的最大值。可以指定为 <b>UNLIMITED</b> 。 |

**CREATE TABLESPACE** 语句的其它参数，如 **MIMIMUM EXTENT**，也影响数据段地址分配。如果已指定，它将保证所有自由的和已分配的表空间区域至少与指定的字节数量(K 或者 M)一样大或是其倍数。它提供了控制表空间中自由空间存储碎片的方法。

### 改变表空间存储设置

用户可以改变表空间的缺省存储参数，以便改变表空间中将要创建的对象缺省规格。为了改变表空间中随后创建对象的缺省存储参数，可使用 **SQL** 语句 **ALTER TABLESPACE**。

```
ALTER TABLESPACE users
  DEFAULT STORAGE (
    NEXT 100K
    MAXEXTENTS 20
    PCTINCREASE 0);
```

不能在 **ALTER** 语句中指定 **INITIAL** 以及 **MINEXTENTS** 关键字。表空间缺省存储参数的新数值只影响将来表空间中数据段区域的分配。

#### 9.3.3 合并字典管理表空间中的自由空间

表空间自由区域由一批邻接的自由存储块组成。当为表空间数据段分配新的区域时，可使用在容量上最接近要求的自由区域。有时候，数据段被删除，它们的区域将被重新分配，并被标记为自由的，但是任何相邻的自由区域不会立即重新结合成为大的自由区域。结果是产生存储碎片，它使较大区域的地址分配更为困难。

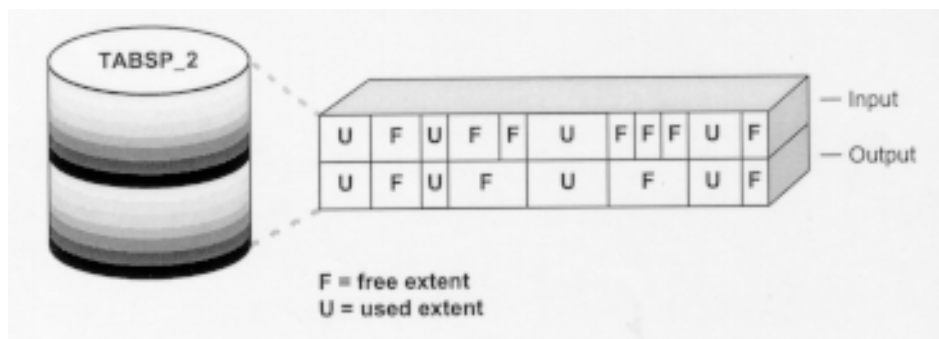
访问存储碎片有下列方法：

1. 当试图为数据段分配一个新区域的时候，**Oracle** 将首先设法为此新区域找到一个足够大的自由区域。如果没有找到足够大的自由区域，**Oracle** 就会合并相邻的表空间自由区域，并再一次查找。无论什么时候，如果 **Oracle** 不能为安装新区域找到一个适合的自由区域，就会执行合并操作。
2. 当表空间的 **PCTINCREASE** 值不为零时，**SMON** 后台进程会定期合并邻近的自由区域。如果用户设置 **PCTINCREASE=0**，将不会定期合并自由区域。如果用户担心正在进行的 **SMON** 的总开销，可以设置 **PCTINCREASE=0**，并如(4)中的论述，定期合并自由空间。
3. 当一个数据段丢失或者被截断时，如果数据段的 **PCTINCREASE** 值非零。就会执行限定形式的合并操作。即使包含此数据段的表空间的 **PCTINCREASE** 为零，这个操作也会

进行。

4. 用户可以使用 `ALTER TABLESPACE...COALESCE` 语句手动地合并任何相邻的自由区域。

合并自由空间的过程如下列图表所示。



对于本地管理表空间没有必要合并自由空间。

为获取区域分配和自由空间合并的详细信息，请参见“Oracle8i Concepts”。

如果用户发现表空间中的存储碎片较多(用户磁盘上相连存储空间出现非邻接的情况)，用户可以使用 `ALTER TABLESPACE ...COALESCE` 语句合并任何自由空间。用户必须有 `ALTER TABLESPACE` 的系统权限来合并表空间。

如果 `PCTINCREASE` 为零，用户可能要用到此语句，或者使用它来补充 `SMON` 和区域分配的合并。注意，如果表空间中所有的区域同样大，则没有必要合并的，也就是下面这种情况：如果表空间的 `PCTINCREASE` 缺省值是零，所有表空间的数据段使用缺省存储参数，并且 `INITIAL=NEXT=MINIMUM EXTENT`。

下面的语句合并表空间 `TABSP_4` 中的自由空间。

```
ALTER TABLESPACE tabsp_4 COALESCE;
```

与 `ALTER TABLESPACE` 语句的其他选项相似，`COALESCE` 选项是排它的：当进行指定时，它必须是唯一的选项。

这个语句不会合并被数据区域分开的自由区域。如果用户注意到有许多自由区域位于数据区域之间，用户必须重新组织此表空间(例如：导出导入数据)以创建可用的自由空间区域。

用户可以使用 `DBA_FREE_SPACE` 和 `DBA_FREE_SPACE_COALESCED` 监视自由空间，并且为合并操作显示统计信息。下面的语句显示表空间 `TABSP_4` 中的自由空间。

```
SELECT block_id, bytes, blocks
FROM dba_free_space
WHERE tablespace_name = 'TABSP_4'
ORDER BY block_id;
```

| BLOCK_ID | BYTES | BLOCKS |
|----------|-------|--------|
| -----    | ----- | -----  |

|    |        |    |
|----|--------|----|
| 2  | 16384  | 2  |
| 4  | 16384  | 2  |
| 6  | 81920  | 10 |
| 16 | 16384  | 2  |
| 27 | 16384  | 2  |
| 29 | 16384  | 2  |
| 31 | 16384  | 2  |
| 33 | 16384  | 2  |
| 35 | 16384  | 2  |
| 37 | 16384  | 2  |
| 39 | 8192   | 1  |
| 40 | 8192   | 1  |
| 41 | 196608 | 24 |

13 rows selected.

这个视图表明在 **TABSP\_4** 中有相邻的自由空间(例如，从存储块 IDs 2, 4, 6, 16 开始的存储块)没有被合并。在使用前面演示中提到的 **ALTER TABLE** 语句合并此表空间以后，查询的结果将是：

| BLOCK_ID | BYTES  | BLOCKS |
|----------|--------|--------|
| 2        | 131072 | 16     |
| 27       | 311296 | 38     |

2 rows selected.

要显示关于合并操作的统计信息，可以使用 **DBA\_FREE\_SPACE\_COALESCED** 视图。它也有助于用户合并空间。关于这两个视图的更多信息，请参见“Oracle8i Reference”。

## 9.4 改变表空间的工作效率

用户可以使一个联机表空间脱机，以便使这个数据库的部分内容对于一般的应用暂时不可用，而其余部分仍是开启并且可用的。相反，用户可以把一个脱机表空间联机，使此表空间内的主要对象可供数据库用户使用且此数据库必须开启。

为了改变表空间的工作效率，可使用 SQL 语句 **ALTER TABLESPACE**。用户必须具有 **ALTER TABLESPACE** 或者 **MANAGE TABLESPACE** 的系统权限来执行相应的操作。

### 9.4.1 使表空间脱机

用户可能由于下列原因使一个表空间脱机：

- 在数据库的其它部分允许正常访问的情况下，使数据库的某个部分不可用。
- 想要执行一个脱机表空间的备份(即使此表空间可以在联机和使用状态进行备份)。
- 在更新或者维护应用的时候使某一应用和相应的数据表组群暂时不可用。

当某个表空间脱机时，**Oracle** 会使所有的相关文件脱机。但不能使 **SYSTEM** 表空间脱机。

当用户使某个表空间脱机时，可以指定下列任一选项：

**NORMAL**     如果表空间的数据文件不存在错误，那么它可以正常地脱机。在表空间

中，不会有数据文件存在写入错误而能够正常脱机。Oracle 在使表空间所有数据文件脱机的时候，会利用正常脱机次序为它们生成检验点。

**TEMPORARY** 一个表空间可以暂时脱机，即使表空间的某一个或者多个文件存在错误状态。Oracle 在使没有脱机的数据文件脱机时，会利用临时脱机次序为它们建立检验点。

如果文件没有脱机，而用户使用了临时选项，那么使表空间回复联机并不要求介质恢复操作。然而，如果某一或者多个表空间文件因为写入错误而脱机，并且用户使表空间暂时脱机，则表空间在回复联机之前需要进行恢复操作。

**IMMEDIATE** 表空间可以立即脱机，而在其数据文件中没有 Oracle 生成的任何检验点。在表空间可以联机之前，需要利用立即脱机次序对表空间进行介质恢复。如果数据库运行在 NOARCHIVELOG 模式下，则用户不能使表空间立即脱机。

**FOR RECOVER** 为了执行表空间的时间点恢复而使恢复集中的生产数据库表空间脱机。详细信息请参见“Oracle8i Backup and Recovery Guide”。

**警告** 如果用户必须使一个表空间脱机，如有可能应使用 NORMAL 选项(缺省)；这样，即使用户在将表空间回复联机之前将重演日志序列复位(在不完全介质恢复后使用 ALTER DATABASE OPEN RESETLOGS 语句)，也可以确保表空间恢复联机而不需要进行恢复操作。

只有在用户不能使表空间正常脱机的情况下才指定 **TEMPORARY**；假若这样，在表空间可以联机之前，只有因为错误而脱机的文件需要恢复。在尝试 normal 和 temporary 选项后再指定使用 **IMMEDIATE** 选项。

下面的例子使 **USERS** 表空间正常脱机：

```
ALTER TABLESPACE users OFFLINE NORMAL;
```

在使一个联机表空间脱机之前，用户需要考虑如下问题：

- 验证此表空间中不包含有效的回退数据段。这样的表空间不能脱机。更多的信息，请参见第 11 章“使回退段脱机”的内容。
- 因为表空间在脱机的时候将不能存取对象，如果它们被作为缺省的临时表空间或者表空间中的排序区域分配给用户，那么用户可能要改变表空间分配。

#### 9.4.2 使表空间联机

只要数据库开启，用户就可以将其中的任何表空间联机。表空间正常联机，数据库使用者就可利用它内部的数据。

**注意** 如果将要联机的表空间不曾“干净的”脱机(也就是说，未使用 ALTER TABLESPACE OFFLINE 语句的 NORMAL 选项)，那么用户在将其联机之前必须首先执行针对这个表空间的介质恢复。否则，Oracle 将返回错误，而且此表空间保持脱机状态。

下面的语句使 **USERS** 表空间联机：

```
ALTER TABLESPACE users ONLINE;
```

## 9.5 只读表空间

将表空间设置为只读可以防止对表空间中数据文件的写入操作。只读表空间的主要目的是消除对数据库大型静态部分的备份和恢复操作，但同时也是对历史数据完全保护提供了方法，这样就没有人可以改变数据。将表空间设置为只读可以防止对表空间中全部表的更新，而与用户的更新权限级别无关。

只读表空间还可以移动到其它的数据库。参见 9.8 节“在数据库之间移动表空间”以获取有关该功能的更多信息。

**注意** 将表空间设置为只读，其本身不能满足存档或者数据发表的要求，因为表空间只可在创建它的数据库中联机。然而，用户使用可移动表空间功能可以满足这样的要求。

用户可以从只读表空间中删除一些项目，比如目录或者索引，但是用户不能在表空间中创建或者改变对象。用户可以执行一些指令以更新数据字典中的文件描述，比如 `ALTER TABLE ADD` 或者 `ALTER TABLE...MODIFY`，但是用户在表空间设置为可读写以前不能应用新的描述。

因为只读表空间不会被修改，它们就可以存于 `CD-ROM` 或者 `WORM`(一次写-多次读)装置中。参见 9.5.3 节“在 `WORM` 设备上创建只读表空间”。

本节论述下面的主题：

- 将表空间设置为只读
- 使只读表空间可写
- 在 `WORM` 设备上创建只读表空间
- 在只读表空间中延迟打开数据文件

参见 关于只读表空间的更多信息，请参见“Oracle8i Concepts”。

### 9.5.1 将表空间设置为只读

所有的表空间最初被创建时都是可读写的。在 `ALTER TABLESPACE` 指令中使用 `READ ONLY` 关键字，可以把一个表空间改为只读。用户必须具有 `ALTER TABLESPACE` 或 `TABLESPACE` 的系统权限。在将表空间设置为只读之前，必须满足下列条件。

- 表空间必须被联机。  
确保没有要应用于表空间的复原信息。
- 此表空间决不能包含任何有效的回退数据段(这是正常的情况，因为一个数据表空间不应该包含回退数据段)。

由于这个原因，`SYSTEM` 表空间将不会设置为只读，因为其包含 `SYSTEM` 重新运行数据段。另外，因为只读表空间的任何回退数据段是不可访问的，用户在将表空间设置为只读之前必须删除回退数据段。

- 此表空间决不能包含在联机备份中，因为在备份工作的结尾会修改表空间中所有数据文件的主文件。

为了在访问只读表空间的数据时有更好的操作性能，在将其设置为只读前，用户可以发出一个查询，以访问表空间中数据表的所有存储块。一个针对各个表执行的简单查询，比



如 `SELECT COUNT(*)`，将确保随后对此表空间中数据块的访问非常高效。这样可以消除 Oracle 对于检验事务处理状态的需要，事务处理会造成对存储块的修改。

下面的指令使 `FLIGHTS` 表空间只读：

```
ALTER TABLESPACE flights READ ONLY;
```

在发出 `ALTER TABLESPACE...READ ONLY` 指令之前，用户不必等待事务处理完成。当指令发出时，目标表空间会进入过渡性的只读方式，此时允许提交或者回退现存的修改表空间的事件，但不允许更进一步的 `DML` 指令。一旦这种情况出现，就有效的事务处理而言，表空间是静止的。

**注意** 只有初始参数 `COMPATIBLE` 的值是 8.1.0 或者更大的情况下，这种过渡的只读状态才会出现。因为如果参数值不到 8.1.0 而又存在一些有效的事务处理时，`ALTER TABLESPACE...READ ONLY` 指令会失效。

如果用户发现对于此表空间来说，静止花费了较长的时间，有可能表明妨碍只读状态的事务处理在起作用。这些事务处理的使用者可以得到提示，并且在必要时可以决定终止那些事务处理。下面的例子说明用户如何确定此类阻塞事务。

■ 为 `ALTER TABLESPACE...READ ONLY` 指定确定事务处理的条目。

```
SELECT sql_text, saddr
FROM V$sqlarea, V$session
WHERE V$sqlarea.address = V$session.sql_address
AND sql_text like 'alter tablespace%';
```

| SQL_TEXT                        | SADDR    |
|---------------------------------|----------|
| alter tablespace tbs1 read only | 80034AFO |

■ 各个有效事务处理的 `SCN` 启动储存在 `V$TRANSACTION` 视图中。升序排列的启动 `SCN` 显示视图列举了执行序列中的事务处理。了解只读指令的事务处理条目，它可以在 `V$TRANSACTION` 视图中找到。所有等于或者小于启动 `SCN` 的事务处理，都有可能阻滞表空间的静止状态以及其后的只读状态。

```
SELECT ses_addr, start_scnb
FROM v$transaction
ORDER BY start_scnb;
```

| SES_ADDR | START_SCNB |  |
|----------|------------|--|
| 800352A0 | 3621       | --> waiting on this txn                    |
| 80035A50 | 3623       | --> waiting on this txn                    |
| 80034AFO | 3628       | --> this is the ALTER TABLESPACE statement |
| 80037910 | 3629       | --> don't care about this txn              |

在使表空间只读后，立即对它进行备份是明智的。只要此表空间保持只读，没有必要进行更进一步的表空间备份，因为它不会再产生变化。

参见 为了获取关于利用只读数据文件恢复数据库的信息，请参见“Oracle8i Backup and

Recovery Guide”。

### 9.5.2 使只读表空间可写

在 SQL 语句 ALTER TABLESPACE 中使用 READ WRITE 关键字，可以将表空间改变为可写。用户必须具有 ALTER TABLESPACE 或 TABLESPACE 的系统权限。

使表空间可读写所必须具备的条件是表空间中所有数据文件以及表空间本身应联机。使用 ALTER DATABASE 指令的 DATAFILE...ONLINE 子句可将数据文件联机。V\$DATAFILE 列举了数据文件的当前状态。

下列指令可使 FLIGHTS 表空间可写：

```
ALTER TABLESPACE flights READ WRITE;
```

使只读表空间可写的操作会修改数据文件的控制文件条目，因此用户可以用数据文件的只读版本作为恢复操作的起点。

### 9.5.3 在 WORM 设备上创建只读表空间

执行下列步骤以便在 CD-ROM 或者 WORM 设备上创建只读表空间。

1. 在其它设备上创建可写表空间。创建属于此表空间的数据对象并放入用户的数据。
2. 将表空间改变为只读。
3. 将表空间的数据文件复制到 WORM 设备上。用操作系统命令复制文件。
4. 使表空间脱机。
5. 重命名数据文件以符合用户复制到 WORM 设备上的数据文件的名称。使用带有 RENAME DATAFILE 子句的 ALTER TABLESPACE 语句。数据文件的重命名会改变它们在控制文件中的名称。
6. 将表空间恢复联机。

### 9.5.4 在只读表空间中延迟打开数据文件

若一个大型数据库的重要部份存储在只读表空间中，而该表空间位于慢速存取设备或者分级存储装置中，用户应该考虑设置初始参数 READ\_ONLY\_OPEN\_DELAYED 为 TRUE。这将加快一些操作的执行，主要是数据库的打开操作，这是因为对于只读表空间中的数据文件，只在试图读出其中数据时才第一次被访问。

设置 READ\_ONLY\_OPEN\_DELAYED=TRUE 有下列副作用：

- 在打开时不会发现只读文件的丢失或者损坏。只有在试图存取时才会发现。
- ALTER DATABASE CHECK DATAFILES 不会检验只读文件。
- ALTER TABLESPACE <name> ONLINE 和 ALTER DATABASE DATAFILE <name> ONLINE 将不检查只读文件。它们只在首次存取时检查。
- V\$RECOVER\_FILE, V\$BACKUP 和 V\$DATAFILE\_HEADER 将不存取只读文件；只读文件将在带有“DELAYED OPEN”错误显示的结果列表中，与其它数据列值的零点一起标明。
- V\$DATAFILE 将不存取只读文件；只读文件列出的容量为 0。
- V\$RECOVER\_LOG 不存取只读文件；恢复操作可能需要的日志不会加入列表。

- **ALTER DATABASE NOARCHIVELOG** 不会存取只读文件；即使有需要恢复的只读文件，该操作也将继续进行。

注意：

- **RECOVER DATABASE** 和 **ALTER DATABASE OPEN RESETLOGS** 仍旧存取所有的只读数据文件，而与参数值无关。如果用户要避免这些操作存取只读文件，应该使文件脱机。
- 如果使用备份的控制文件，一些文件的只读状态可能是不正确的。这可能导致一些操作返回意外的结果，应该注意这个情况。

## 9.6 删除表空间

如果不再需要表空间及其内容(包含在表空间中的数据段)，用户可以从数据库中将其删除。除 **SYSTEM** 表空间之外，**Oracle** 数据库中的任何表空间都可以删除。用户必须具有 **DROP TABLESPACE** 的系统权限才可以删除表空间。

**警告** 一旦表空间被删除，此表空间的数据将是不可恢复的。因此，要确定将来不再需要将被删除的表空间中的所有数据。并且，在从数据库中删除表空间前后，应立即完整备份此数据库。强烈建议进行这样的操作，因为一旦用户错误地删除了一个表空间，或者在删除了表空间后，数据库在将来的使用中将遇到问题。但用户可以恢复数据库。

如果用户删除一个表空间，只有数据库控制文件中的相关文件位置指针丢失。构成此表空间的数据文件仍旧存在。如果要释放以前使用的磁盘空间，就要在完成操作之后，用操作系统的适当命令删除已删除的表空间的数据文件。

用户不能删除一个包含任何现用段的表空间。例如：如果一个表空间的数据表正在使用中，或者此表空间包含一个活动的回退数据段，则用户不能删除该表空间。为了简单起见，在删除表空间之前，应使其脱机。

在一个表空间被删除以后，该表空间的条目依然保留在数据字典中(参见 **DBA\_TABLESPACES** 视图)，但是该表空间的状态被改变为 **INVALID**。

要删除一个表空间，可以使用 **SQL** 语句 **DROP TABLESPACE**。下面的指令删除了 **USERS** 表空间，包括该表空间中的数据段：

```
DROP TABLESPACE users INCLUDING CONTENTS;
```

如果该表空间是空的(不包含任何表、视图或者其它的结构)，用户不需要指定 **INCLUDING CONTENTS** 选项。使用 **CASCADE CONSTRAINTS** 选项可以删除所有引用完整性约束，它们来源于表空间外部的表，该表引用了表空间内部表主要的和独特的键值。

## 9.7 使用 DBMS\_SPACE\_ADMIN 程序包

**DBMS\_SPACE\_ADMIN** 程序包为管理员提供了针对本地管理表空间的故障诊断和修理功能。**DBMS\_SPACE\_ADMIN** 程序包包含下列过程：

| 程序                            | 描述                            |
|-------------------------------|-------------------------------|
| SEGMENT_VERIFY                | 验证数据段区域位图的一致性                 |
| SEGMENT_CORRUPT               | 标记该数据段为损坏的或者有效的，以便可以执行适当的错误校正 |
| SEGMENT_DROP_CORRUPT          | 删除当前标记为坏的数据段(不回收设备空间)         |
| SEGMENT_DUMP                  | 清除数据段头部信息和给定数据段的区域位图          |
| TABLESPACE_VERIFY             | 验证表空间数据段的位图和区域位图是同步的          |
| TABLESPACE_REBUILD_BITMAPS    | 重建正确的位图                       |
| TABLESPACE_FIX_BITMAPS        | 将正确的 DBA 范围(区域)标记为自由的或是用于位图的  |
| TABLESPACE_REBUILD_QUOTAS     | 为给定表空间重建定额                    |
| TABLESPACE_MIGRATE_FROM_LOCAL | 将本地管理表空间转为字典管理表空间             |
| TABLESPACE_MIGRATE_TO_LOCAL   | 将表空间从字典管理格式转为本地管理格式           |
| TABLESPACE_RELOCATE_BITMAPS   | 重定位指定目的地的位图                   |
| TABLESPACE_FIX_SEGMENT_STATES | 修复移动失败的表空间中的数据段               |

下面设想的情况描述了用户可以使用 DBMS\_SPACE\_ADMIN 程序包来诊断和解决问题的典型情况。

**注意** 如果不能恰当地使用，一些程序可能产生丢失和不可恢复的数据。如果用户对这些程序有疑问，可以利用遍及全世界的 Oracle 技术支持进行工作。

**参见** 有关 DBMS\_SPACE\_ADMIN 程序包和程序的细节，参见“Oracle8i Supplied PL/SQL Packages Reference”。

### 9.7.1 情况 1

TABLESPACE\_VERIFY 程序可以发现位图中已经分配的但被标记为“free”的存储块，反过来也是一样。它还会发现在数据段之间的交叉。

在此情况下，执行下面的任务：

- 调用 SEGMENT\_DUMP 程序清除管理员分配给数据段的范围。
- 对于每个范围，利用 TABLESPACE\_EXTENT\_MAKE\_USED 或者 TABLESPACE\_EXTENT\_MAKE\_FREE 选项调用 TABLESPACE\_FIX\_BITMAPS 程序，将空间标记为已使用或是自由的。

### 9.7.2 情况 2

因为位图中已经将数据段存储块标记为“free”，用户不能删除数据段。

在此情况下，执行下面的任务：

- 利用 SEGMENT\_VERIFY\_EXTENTS\_GLOBAL 选项调用 SEGMENT\_VERIFY 程序。如果没有产生交叉报告，可执行下面的操作：
  - 利用 SEGMENT\_DUMP\_EXTENT\_MAP 选项调用 SEGMENT\_DUMP 程序，清除管理员分配给该数据段的范围。
  - 对于每个范围，利用 TABLESPACE\_EXTENT\_MAKE\_FREE 选项调用 TABLESPACE\_FIX\_BITMAPS 程序，标记该空间为“free”。

- 利用 `SEGMENT_MARK_CORRUPT` 选项调用 `SEGMENT_CORRUPT` 程序，标记该数据段为坏的。
- 调用 `SEGMENT_DROP_CORRUPT` 程序删除 `SEG$` 条目。

### 9.7.3 情况 3

`TABLESPACE_VERIFY` 程序已经报告存在一些交叉。由于先前的内部错误，一些有效的数据必须删除。

在选择好要删除的对象以后，假定是表 `T1`，可执行下面的任务：

- 生成与 `T1` 交叉的所有对象的列表。
- 删除表 `T1`。必要时，调用 `SEGMENT_DROP_CORRUPT` 程序继续操作。
- 对于与 `T1` 交叉的所有数据对象调用 `SEGMENT_VERIFY` 程序。必要时，调用 `TABLESPACE_FIX_BITMAPS` 程序将正确的位图标记为已用的。
- 重新运行 `TABLESPACE_VERIFY` 程序验证问题是否已被解决。

### 9.7.4 情况 4

一组位图存储块已经发生介质损坏。在此情况下，执行下面的任务：

- 调用 `TABLESPACE_REBUILD_MAPS` 程序处理所有的位图存储块，或是处理单个被损坏的存储块。
- 调用 `TABLESPACE_VERIFY` 程序验证位图的一致性。

### 9.7.5 情况 5

用户将一个字典管理的表空间转为本地管理的表空间，可使用 `TABLESPACE_MIGRATE_TO_LOCAL` 程序。

假定该数据库模块大小是 2K，并且表空间 `TBS_1` 中现存的区域容量是 10、50 和 10,000 个存储块(已用，已用，自由的)。`MINIMUM EXTENT` 值是 20K(10 个存储块)。在此情况下，用户允许由系统选择位图地址分配单位，选择结果是 10 个存储块，因为它是最大的共同尺度，并且不会超过 `MINIMUM EXTENT`。

将 `TBS_1` 转换为一个本地管理表空间的指令如下：

```
EXEC DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_TO_LOCAL ('tbs_1');
```

如果用户选择指定地址分配单位的大小，那它必须是由系统计算出的单位大小的倍数，否则会返回错误信息。

## 9.8 在数据库之间移动表空间

本节描述如何在数据库之间移动表空间，其中包括下列主题：

- 可移动表空间的介绍
- 限制
- 在数据库之间移动表空间的程序
- 对象行为

## ■ 使用可移动表空间

### 9.8.1 可移动表空间的介绍

**注意** 用户必须有 Oracle8i Enterprise Edition 以生成一个可移动表空间集。然而，用户可以使用任何版本的 Oracle8i 将一个可移动表空间集插入 Oracle 数据库。

用户可以使用移动表空间功能移动 Oracle 数据库的一个子集，并且在其他的 Oracle 数据库中“插入”它，实质上在数据库之间移动表空间。移动表空间对于下列操作特别有用：

- 从 OLTP 系统向数据仓库逐级分离系统移动数据
- 用逐级分离系统更新数据仓库和数据商业中心
- 用中央数据仓库装载数据商业中心
- OLTP 和数据仓库系统的有效存档
- 向内部和外部客户发布数据

通过可移动表空间移动数据可以比导入/导出或装载/卸载相同的数据快许多，因为移动一个表空间只需要复制数据文件并整合该表空间的结构信息。用户还可以使用可移动表空间来移动索引数据，由此可避免该索引重建，它是用户在导入或者装载列表数据时必须执行的操作。

**参见** 关于可移动表空间及其在数据商业中心和数据仓库中使用的更多细节，请参见“Oracle8i Concepts”。

关于使用可移动表空间执行介质恢复的信息，请参见“Oracle8i Backup and Recovery Guide”。

关于可移动表空间兼容性问题的信息(在不同 Oracle 版本间的兼容性问题)，请参见“Oracle8i Migration”。

### 9.8.2 限制

在用户打算使用可移动表空间时，要注意下列限制：

- 源数据库以及目标数据库必须在相同的硬件平台上。例如：用户可以在 Sun Solaris Oracle 数据库之间移动表空间，或者可以在 NT Oracle 数据库之间移动表空间。然而，用户不能从 SUN Solaris Oracle 数据库向 NT Oracle 数据库移动表空间。
- 源数据库和目标数据库必须有相同大小的数据库模块。
- 源数据库和目标数据库必须使用相同的字符集和国家标准符号集。
- 对于一个表空间，如果在目标数据库中已经存在与其同名的表空间，则用户不能将其移动到目标数据库中。
- 可移动表空间不支持下列操作：
  - 快照/复本
  - 基于功能的索引
  - 已划定作用域的 REFs
  - 域索引(由可扩展索引提供的新索引类型)
  - 有多个收件人与 8.0 兼容的高级队列

### 9.8.3 在数据库之间移动表空间的程序

要移动或复制一组表空间，用户必须执行下面的步骤：

1. 选择独立表空间。
2. 生成一个可移动的表空间集。

一个可移动的表空间集包括将要移动的表空间集的数据文件，以及一个包含表空间集结构信息的文件。

3. 移动该表空间集。

将数据文件以及导出文件复制到目标数据库，用户可以使用任何手段复制单调信息(例如：0/S 复制工具，远程传输，或者在 CD 上发布)。

4. 插入表空间。

启用导入操作以将表空间集插入目标数据库。

以下是详细步骤。

#### 步骤 1: 选择独立的表空间集

用户只可移动独立的表空间。在上下文中“独立的”意思是表空间集的内部没有指向表空间外部的参考。例如：如果在表空间集中有一个关于表空间集外部的数据表的索引，那么该表空间集不是独立的。

用户想要复制的表空间集中必须包含已分区数据表的所有分区，或是不包含已分区数据表中的任何分区。如果用户想要移动已分区数据表的某个子集，用户必须将该分区转换为表。

当移动一组表空间时，用户可以选择包括引用完整性约束。不论表空间集是否是独立的，这么做都可以起作用。如果用户决定不移动约束，那么该约束将不会当作线索使用。一些违反独立表空间规定的例子如下：

- 表空间集内部的索引是关于表空间集外部的某个数据表的。
- 某个已分区数据表被部分地包含在该表空间集中。
- 表空间集内部的数据表包含指向表空间集外部 LOB 的 LOB 数据列。

为了确定一组表空间是否是独立的，用户可以启用 Oracle 提供的 DBMS\_TTS 程序包中的 TRANSPORT\_SET\_CHECK 程序。用户必须授予了 EXECUTE\_CATALOG\_ROLE 角色(起初标记为 SYS)才可执行此程序。用户列举表空间名称列表并表明是否想要移动引用完整性约束。

例如：假定用户要确定表空间 SALES\_1 和 SALES\_2 是否是独立的，并考虑完整性约束(由 TRUE 表明)，可以执行下面的指令：

```
EXECUTE dbms_tts.transport_set_check('sales_1,sales_2', TRUE);
```

在启用 PL/SQL 程序以后，用户可以看到从 TRANSPORT\_SET\_VIOLATIONS 视图选出的所有违例情况。如果该表空间集是独立的，此视图将是空的。下面的查询显示有两个违例的情况：外部关键字约束，DEPT\_FK，它不在表空间集的范围内，而且已分区数据表，JIM.SALES 被部分地包含在该表空间集中。

```
SELECT * FROM transport_set_violations;  
VIOLATIONS
```

-----  
Constraint DEPT\_FK between table JIM.ENIP in tablespace SALES\_1 and table  
JIM.DEPT in tablespace OTHER

Partitioned table JIM.SALES is partially contained in the transportable set

对象引用(比如 REFs)超出表空间集不会认为是违例。TRANSPORT\_SET\_CHECK 程序不检查 REFs。如果包含挂起 REFs 的表空间插入数据库, 按照挂起 REF 的查询会指示用户发生错误。

参见 关于 REFs 的更多信息, 请参见 “Oracle8i Application Developer ’s Guide-Fundamentals”。

关于 DBMS\_TTS 程序包的更多信息, 请参见 “Oracle8i Supplied PL/SQL Packages Reference”。

### 步骤 2: 生成可移动表空间集

在确定用户要移动的独立表空间集后, 下列操作可生成可移动表空间:

1. 使用户正在复制的集合中所有的表空间只读。

```
ALTER TABLESPACE sales_1 READ ONLY;
```

2. 启用导出工具并列举哪些表空间在可移动集合中的指令如下:

```
EXP TRANSPORT_TABLESPACE=y TABLESPACES=(sales 1,sales 2)  
TRIGGERS=y/n CONSTRAINTS=y/n GRANTS=y/n FILE=expdat.dmp
```

注意 虽然使用了导出工具, 只有数据字典结构信息被导出。因此, 此操作对于大型表空间执行的也很快。

当有提示时, 按照 “sys AS sysdba” 进行连接。

用户必须指定 TABLESPACES。参数 FILE 说明要创建的结构信息导出文件的文件名称。

如果用户设置 TRIGGERS=n, 触发器不被导出。如果用户设置 TRIGGERS=y, 触发器不经有效性检查而导出。无效的触发器在其后的导入期间将导致编码错误。

如果用户设置 GRANTS=y, 所有在导出表上的粒度 (grants) 也被导出; 否则, 所有的 GRANTS 都被忽略。

如果用户设置 CONSTRAINTS=y, 引用完整性约束被导出; 否则, 引用完整性约束被忽略。

所有选项的缺省设置都是 “Y”。

3. 将数据文件复制到某个单独的存储空间或是目标数据库中。

4. 必要时, 将已复制集合中的表空间返回为可读写模式, 指令如下:

```
ALTER TABLESPACE sales_1 READ WRITE;
```

如果被移动的表空间不是独立的, 导出操作会失败并指示可移动集合不独立。用户必须返回步骤 1 以解决所有的违例情况。

参见 有关使用导出工具的信息, 可参考 “Oracle8i Utilities”。

### 步骤 3: 移动表空间集

将数据文件和导出文件都移动到某个位置, 该位置对于目标数据库是可访问的。用户可



以使用任何手段复制单调信息(例如: 0/S 复制工具, 远程传输, 或是在 CD 上发表)。

#### 步骤 4: 插入表空间集

可执行下面的任务插入表空间集:

1. 将已复制的表空间集数据文件放置在某个位置, 在此位置上目标数据库可以存取它们。

2. 插入表空间并使用下面的导入指令整合结构信息:

```
IMP TRANSPORT_TABLESPACE=y DATAFILES= ('/db/sales_jan', '/db/sales_feb', ...)
```

```
TABLESPACES=(sales_1,sales_2) TTS_OWNERS=(dcranney,jfee)
```

```
FROMUSER= (dcranney, jfee) TOUSER= (smith, williams) FILE=expdat.dmp
```

当有提示时, 可按照 “sys AS sysdba” 进行连接。

以下是两个更多的例子:

```
IMP TRANSPORT TABLESPACE=y DATAFILES= ('/db/staging1.f', '/db/staging2.f')
```

```
IMP TRANSPORT_TABLESPACE=y DATAFILES='/db/staging.f' TABLESPACES=jan OWNERS=smith
```

用户必须指定 DATAFILES。

TABLESPACES, TTS\_OWNERS, FROMUSER 和 TOUSER 是可选的。FILE 参数说明结构信息导出文件的名称。当用户指定了 TABLESPACES 时, 提供的表空间名称将与导出文件中名称进行比较。如果有任何不匹配, 导入操作将返回一个错误。否则, 表空间名称从导出文件中提取。TTS\_OWNERS 列出在表空间集中拥有数据的所有用户。当用户指定 TTS\_OWNERS 时, 用户名将与在导出文件中的名称进行比较。如果有任何不匹配, 导入操作将返回一个错误。否则, 文件编写人名从导出文件中提取。

如果用户不指定 FROMUSER 和 TOUSER, 所有的数据库对象(比如表和索引)将根据源数据库的用户创建。那些用户必须已经存在于目标数据库之中。不然的话, 导入操作将返回错误信息, 表明一些必需的用户不在目标数据库之中。

用户可以使用 FROMUSER 和 TOUSER 改变数据对象的拥有者。例如: 用户指定 FROMUSER=(dcranney,jfee) TOUSER=(smith,williams()), 则源数据库中属于 DCRANNEY 的表空间集中的数据对象, 在表空间集插入目标数据库后, 在目标数据库中将属于 SMITH。同样, 源数据库中属于 JFEE 的数据对象, 在目标数据库中将属于 WILLIAMS。假若这样, 目标数据库中不必有用户 DCRANNEY 和 JFEE, 但是必须要有用户 SMITH 和 WILLIAMS。

在成功执行指令后, 集合中所有被复制的表空间将保持只读方式。用户应该检查导入日志, 确保没有错误出现。此时, 用户可以发出 ALTER TABLESPACE...READ WRITE 指令, 将新的表空间置于可读写模式。

当处理大量的数据文件时, 在命令行中详细说明数据文件名称列表将是一个繁重的处理过程; 它甚至可能超出命令行的限定。在此情况下, 用户可以使用导入参数文件。例如: 下列指令相当于此步骤中执行的指令:

```
IMP PARFILE='par.f'
```

文件 par.f 包含下面的内容:

```
TRANSPORT_TABLESPACE=y
```

```
DATAFILES=db/staging.f  
TABLESPACES=jan  
TT_OWNERS=smith
```

为了在数据库之间移动表空间，源数据库和目标数据库必须都运行 Oracle8i，初始化文件兼容性参数设置为 8.1。

参见 有关使用导入工具的信息，可参考 “Oracle8i Utilities”。

#### 9.8.4 对象行为

大部分数据对象，不论是在表空间里的数据，还是与表空间相关的结构信息，在被移动到不同的数据库后，都可以正常地工作。然而，下列对象除外：

- ROWIDs
- REFs
- 授权
- 已分区数据表
- 对象
- 高级队列
- 索引
- 触发器
- 快照/复本

##### ROWIDs

当数据库包含插入的表空间(从其它的数据库)时，数据库的 ROWIDs 不再唯一。在数据表内，ROWID 应被保证唯一。

##### REFs

当 Oracle 决定一组表空间是否独立时，不检查 REFs。结果，一个加载表空间可能包含悬挂 REFs。任何遵循悬挂 REFs 的查询都将返回用户错误。

##### 授权

如果用户在导出期间指定 GRANTS=y，则授权被移动。在导入期间，一些授权可能失效。例如：授权的用户可能不存在，或是授权的角色不存在。

##### 已分区数据表

如果表空间集合中只包含已分区数据表的子集，则用户不能通过可移动表空间移动一个已分区数据表。用户必须确保表的所有分区都在表空间集中，或是在复制表空间集合之前，把分区转换为表。然而，用户应该注意到，分区转换为表的操作将使已分区数据表的全局索引无效。

在目标数据库中，如果有一个已分区数据表可以准确地匹配数据列，用户可以将表转换回分区。如果表的所有分区均出自相同的外部数据库，则可以确保转换操作成功。如果它们不是——很少出现的情况——转换操作可能返回错误，表明存在数据对象的数量冲突。如果用户把表转换回分区时，收到数据对象数量冲突的报告，用户可以使用 ALTER TABLE

**MOVE PARTITION** 指令移动损坏了的分区。然后，重试转换操作。

如果用户指定使用转换语句的 **WITHOUT VALIDATION** 选项，指令会立即返回，因为它只对结构信息进行操作。然而，因为可能要复制分区中的数据，移动分区的操作可能较慢。参见 9.8.5 节“移动并为数据仓库附加分区”中使用已分区数据表的例子。

#### 对象

一个可移动的表空间集可能包含：

- 表
- 索引
- 位图索引
- 索引组织表-LOBs
- 嵌套表
- 可变数组
- 带有自定义类型数据列的表

如果表空间集包含指向 **BFILE** 的指针，用户必须移动 **BFILE** 并在目标数据库中正确地设置索引簿。

#### 高级队列

用户可以使用移动表空间移动或复制 **Oracle** 高级队列，只要这些队列不是带有多个收件人的 8.0 兼容队列。在队列被移动到目标数据库以后，队列起初是被禁用的。在目标数据库中，移动表空间可读写后，用户可以通过内建的 PL/SQL 程序 **DBMS\_AQADM.START\_QUEUE** 启动队列使它起作用。

#### 索引

用户可以移动普通索引和位图索引。当可移动集合完全包含一个已分区数据表时，用户还可以移动已分区数据表的全局索引。

不支持基于功能的索引和域索引。如果它们存在于表空间之中，在移动该表空间前，用户必须删除它们。

#### 触发器

触发器可以不经有效性检查而导出。换句话说，触发器只参考可移动集合内的对象，**Oracle** 不会对其进行验证。在其后的导入期间，无效的触发器将导致编码错误。

#### 快照/复本

不支持移动快照或是重复的结构信息。如果用户要移动的表空间中某个表是重复的，在移动该表空间前，用户必须删除重复的结构信息，并把该表转化成一个正常的表。

### 9.8.5 使用可移动表空间

下面是可移动表空间的一些可能的应用。

## 移动并为数据仓库附加分区

典型的企业数据仓库包含一个或更多的大型事件表。这些事件表可能按日期分区，使该企业数据仓库成为历史数据库。用户可以建造索引以加快星型查询。事实上，Oracle 建议用户对于这样的已分区数据表建造局部索引，以避免每次从该历史数据库中删除最老的分区后要重建全局索引。

假定每个月份用户想要将一个月的数据装载进数据仓库。在该数据仓库中有一个大型事件表称为 SALES，它下面的数据列：

```
CREATE TABLE sales (invoice_no NUMBER,  
    Sale_year INT NOT NULL,  
    Sale_month INT NOT NULL,  
    Sale_day INT NOT NULL)  
PARTITION BY RANGE (sale_year, sale_month, sale_day)  
    (partition jan98 VALUES LESS THAN (1998, 2, 1),  
    partition feb98 VALUES LESS THAN (1998, 3, 1),  
    partition mar98 VALUES LESS THAN (1998, 4, 1),  
    partition apr98 VALUES LESS THAN (1998, 5, 1),  
    partition may98 VALUES LESS THAN (1998, 6, 1),  
    partition jun98 VALUES LESS THAN (1998, 7, 1),
```

用户可创建一个局部的无前缀索引：

```
CREATE INDEX sales_index ON sales ( invoice_no) LOCAL;
```

起初，所有的分区都是空的，并且在同一个缺省表空间中。每个月，用户希望创建一个分区并将它附在已分区的 SALES 表上。假定是 1998 年 7 月，用户想要将 7 月的销售数据装载进已分区数据表。在一个逐级分离数据库里，用户创建了一个新的表空间，TS\_JUL。用户还在此表空间中创建了一个表，JUL\_SALES，带有与 SALES 表完全一样的数据列类型。用户可以用 CREATE TABLE...AS SELECT 语句创建表 JUL\_SALES。在创建并填充 JUL\_SALES 以后，用户还可以为该表创建一个索引，JUL\_SALE\_INDEX，索引的数据列与 SALES 表中的局部索引相同。在建立索引以后，把表空间 TS\_JUL 移动到数据仓库中。

在该数据仓库中，把一个分区加到 SALES 表中以供 7 月销售数据使用。此操作还为局部无前缀索引创建了另一个分区：

```
ALTER TABLE sales ADD PARTITION ju198 VALUES LESS THAN (1998, 8, 1);
```

通过把移动表 JUL\_SALES 转换为 SALES 表的新分区，可以把它附在 SALES 表上，：

```
ALTER TABLE sales EXCHANGE PARTITION ju198 WITH TABLE jul_sales  
INCLUDING INDEXES  
WITHOUT VALIDATION;
```

此语句把 7 月销售数据放入新分区 JUL98，把新建数据附在已分区数据表上。此语句还把索引 JUL\_SALE\_INDEX 改变为 SALES 表的一个局部索引。此语句应该立即返回，因为它只对结构信息进行操作；它简单地切换数据库指针。如果用户已知新分区的数据不会与先前分区中的数据发生交叉链接，建议用户指定 WITHOUT VALIDATION 选项；否则，在试图使分区范围生效的操作中，该语句将仔细检查所有新分区中的新建数据。

如果所有 **SALES** 表的分区出自相同的逐级分离数据库(该逐级分离数据库从未被破坏), 则转换指令总可以成功。然而, 一般而言, 如果已分区数据表的数据出自不同的数据库, 转换操作就有可能失败。例如: 如果 **SALES** 的 **JUL98** 分区不出自相同的逐级分离数据库, 以上所述的转换操作可能失败, 并返回下面的错误:

```
ORA-19728: data object number conflict between table JUL_SALES and partition  
JAN98 in table SALES
```

为了解决冲突, 可发出下列指令移动损坏了的分区:

```
ALTER TABLE sales MOVE PARTITION jan98;
```

然后重试转换操作。

在转换成功后, 用户可以安全地删除 **JUL\_SALES** 和 **JUL\_SALE\_INDEX**(两者现在是空的)。这样用户成功地把 7 月销售数据装载进用户的数据仓库。

### 在 CD 上发布结构化数据

可移动表空间提供了一种在 CD 上发布结构化数据的方法。数据提供者可以把要发布数据的表空间装入, 生成可移动的集合, 并且把可移动集合复制到 CD 上。然后分发此 CD。当客户收到此 CD 时, 不必把数据文件从 CD 复制到磁盘存储器中, 就可以把它插入现有数据库。例如: 假定 NT 计算机的 D: 驱动器是 CD 驱动器。用户可以插入一个带有数据文件 **catalog.f** 和导出文件 **expdat.dmp** 的可移动集合, 指令如下:

```
IMP TRANSPORT_TABLESPACE=y DATAFILES=D:\Catalog.f FILE='D:\expdat.dmp'
```

数据库仍处于开启状态的情况下, 用户可以移去 CD。其后对表空间的查询将返回一个错误, 表明 Oracle 不能开启在 CD 上的数据文件。然而, 对数据库其它部分的操作不受影响。将 CD 放回驱动器就可以使该表空间再一次可读。

移去 CD 与删除只读表空间的数据文件相同。如果用户关闭并重新启动数据库, Oracle 就会指出它不能发现已经删除的数据文件, 而且将不能开启数据库(除非用户把初始参数 **READ\_ONLY\_OPEN\_DELAYED** 设置为 **TRUE**)。当 **READ\_ONLY\_OPEN\_DELAYED** 是设置为 **TRUE** 时, Oracle 只在有人查询该加载表空间时才读取文件。因此, 如果在 CD 中加入表空间, 用户就应该把 **READ\_ONLY\_OPEN\_DELAYED** 初始参数设置为 **TRUE**, 除非该 CD 永久地附在数据库上。

### 在多重数据库中以只读方式安装相同的表空间

用户可以使用可移动表空间在多重数据库上以只读方式安装一个表空间。这样, 相互独立的数据库可以共用磁盘上的相同数据而不是使用各自磁盘上的数据副本。表空间数据文件必须可被所有的数据库访问。为了避免数据库损坏, 在安装表空间的所有数据库中, 该表空间必须保持只读方式。

用户在多重数据库中以只读方式安装相同的表空间, 可以使用下列的任何一种方法:

- 把表空间插入每个用户想要安装该表空间的数据库中。在单独的数据库中生成一个可移动的集合。将可移动集合中的数据文件放置于所有数据库都可访问的磁盘上。把结构信息导入到每个数据库。

- 在一个数据库中生成可移动集合并把它插入其它的数据库中。 如果用户使用这个方法，就要假定该数据文件已经在共用盘上，并且它们属于一个数据库中现有的表空间。 用户可以使表空间只读，生成可移动的集合，然后，当数据文件在共用盘上保持相同位置的时候，将表空间插入到其它数据库中。

用户可以通过若干方法使这张磁盘可被多个计算机访问。用户可以使用簇文件系统或未处理的磁盘，因为 Oracle 并联服务程序要求如此。由于 Oracle 只读取共用盘上此类型的数据文件，用户还可以使用 NFS。 然而要知道，如果在 NFS 发生故障时用户查询共享的表空间，数据库可能挂起，直到 NFS 操作超时。

以后，用户可以把只读表空间从一些数据库中删除。这么做将不修改表空间的数据文件；因此删除操作将不会损坏表空间。不要使表空间可读写，除非只有一个数据库安装此表空间。

通过可移动表空间存档历史数据

因为可移动表空间集是独立的文件集合，它可以插入任何 Oracle 数据库，用户可以通过本章中叙述的可移动表空间的程序在企业数据仓库中存储旧的/历史的数据。

参见 更多的细节，请参见 “Oracle8i Backup and Recovery Guide”。

用可移动表空间执行 TSPITR

用户可以使用可移动表空间执行表空间的时间点恢复(TSPITR)。

参见 关于如何使用可移动表空间执行 TSPITR 的信息，请参见 “Oracle8i Backup and Recovery Guide”。

9.9 查看表空间信息

下面的数据字典视图提供了关于数据库表空间的有用信息。

| 视图                                   | 描述                              |
|--------------------------------------|---------------------------------|
| V\$TABLESPACE                        | 来自控制文件的所有表空间的名称和数量。             |
| DBA_TABLESPACES,<br>USER_TABLESPACES | 所有表空间(或是用户可访问的)的描述。             |
| DBA_SEGMENTS, USER_SEGMENTS          | 所有表空间(或是用户可访问的)中数据段的信息。         |
| DBA_EXTENTS, USER_EXTENTS            | 所有表空间(或是用户可访问的)中数据区域的信息。        |
| DBA_FREE_SPACE,<br>USER_FREE_SPACE   | 所有表空间(或是用户可访问的)中自由区域的信息。        |
| V\$DATAFILE                          | 关于所有数据文件的信息，包括特有表空间的数量。         |
| V\$TEMPFILE                          | 关于所有临时文件的信息，包括特有表空间的数量。         |
| DBA_DATA_FILES                       | 显示属于表空间的文件(数据文件)。               |
| DBA_TEMP_FILES                       | 显示属于临时表空间的文件(临时文件)。             |
| V\$TEMP_EXTENT_MAP                   | 所有本地管理临时表空间所有区域的信息。             |
| V\$TEMP_EXTENT_POOL                  | 对于本地管理临时表空间：每个请求所保留和使用的临时空间的状态。 |
| V\$TEMP_SPACE_HEADER                 | 显示每个临时文件的已用/自由空间。               |
| DBA_USERS                            | 所有用户的缺省表空间和临时表空间。               |

|                 |  |
|-----------------|--|
| DBA_TS_QUOTAS   | 所有用户的表空间分配列表。                                |
| V\$SORT_SEGMENT | 每个给定请求中排序数据段的信息。仅当表空间是 TEMPORARY 类型时，视图才被更新。 |
| V\$SORT_USER    | 就用户和临时/永久表空间来说的临时排序空间的使用率。                   |

下列是使用上述视图的一些例子。

参见 这些视图的完整描述包含在 “Oracle8i Reference” 中。

#### 列举表空间和缺省存储装置参数: 实例

为了列举数据库中所有表空间的名称和缺省存储装置参数，可使用下列对 DBA\_TABLESPACES 的查询:

```
SELECT tablespace_name "TABLESPACE",
       initial_extent "INITIAL_EXT",
       next_extent "NEXT_EXT",
       min_extents "MIN_EXT",
       max_extents "MAX_EXT",
       pct_increase
FROM dba_tablespaces;
```

| TABLESPACE | INITIAL_EXT | NEXT_EXT | MIN_EXT | MAX_EXT | PCT_INCREASE |
|------------|-------------|----------|---------|---------|--------------|
| -----      | -----       | -----    | -----   | -----   | -----        |
| RBS        | 1048576     | 1048576  | 2       | 40      | 0            |
| SYSTEM     | 106496      | 106496   | 1       | 99      | 1            |
| TEMP       | 106496      | 106496   | 1       | 99      | 0            |
| TESTTBS    | 57344       | 16384    | 2       | 10      | 1            |
| USERS      | 57344       | 57344    | 1       | 99      | 1            |

#### 列举数据库的数据文件和相关表空间: 实例

为了列举数据库的名称，容量，和相关表空间，键入下列对 DBA\_DATA\_FILES 的查询:

```
SELECT file_name, blocks, tablespace_name
FROM dba_data_files;
```

| FILE_NAME                       | BLOCKS | TABLESPACE_NAME |
|---------------------------------|--------|-----------------|
| -----                           | -----  | -----           |
| /U02/ORACLE/IDDB3/RBS01.DBF     | 1536   | RBS             |
| /U02/ORACLE/IDDB3/SYSTEM01.DBF  | 6586   | SYSTEM          |
| /U02/ORACLE/IDDB3/TEMP01.DBF    | 6400   | TEMP            |
| /U02/ORACLE/IDDB3/TESTTBS01.DBF | 6400   | TESTTBS         |
| /U02/ORACLE/IDDB3/USERS01.DBF   | 384    | USERS           |

#### 关于各表空间中自由空间(区域)的统计信息: 实例

为了生产数据库中各表空间的自由区域和合并活动的统计信息，可键入下列查询:

```

SELECT tablespace_name "TABLESPACE", file_id,
       COUNT(*)        "PIECES",
       MAX(blocks)      "MAXIMUM",
       MIN(blocks)      "MINIMUM",
       AVG(blocks)      "AVERAGE",
       SUM(blocks)      "TOTAL"
FROM sys.dba_free_space
WHERE tablespace_name = 'SYSTEM'
GROUP BY tablespace_name, file_id;

```

| TABLESPACE | FILE_ID | PIECES | MAXIMUM | MINIMUM | AVERAGE | TOTAL |
|------------|---------|--------|---------|---------|---------|-------|
| RBS        | 2       | 1      | 955     | 955     | 955     | 955   |
| SYSTEM     | 1       | 1      | 119     | 119     | 119     | 119   |
| TEMP       | 4       | 1      | 6399    | 6399    | 6399    | 6399  |
| TESTTBS    | 5       | 5      | 6364    | 3       | 1278    | 6390  |
| USERS      | 3       | 1      | 363     | 363     | 363     | 363   |

**PIECES** 显示表空间文件中自由空间区域的数量，**MAXIMUM** 和 **MINIMUM** 显示数据库存储块中最大的和最小的邻接区域，**AVERAGE** 显示自由空间区域存储块中的平均容量，**TOTAL** 以存储块方式显示各表空间文件中自由空间的数量。 如果用户要创建新对象，或是用户知道一个数据段即将扩展，并且要查明在表空间中是否有足够空间，那么这些查询是非常有用的。



# 10 管理数据文件

本章讲述数据文件管理的各个方面，其中包括下列章节：

- 管理数据文件的方针
- 向一个表空间创建和增加数据文件
- 变更数据文件的尺寸
- 改变数据文件的可用性
- 重命名和重新部署数据文件
- 校验数据文件中的数据块
- 查看数据文件的相关信息

参见 当从失败的介质中恢复数据库信息时，也能创建数据文件。更多的信息参见“Oracle8i Backup and Recovery Guide”。

## 10.1 管理数据文件的方针

本节讲述管理数据文件的各个方面，包括下列标题：

- 确定数据文件的数量
- 设置数据文件尺寸
- 适当放置数据文件
- 存储数据文件与重做日志文件分开

每个数据文件都有两个相关的文件编号：一个是绝对文件号，一个是相对文件号。

绝对文件号是在数据库中的数据文件的唯一标识。在早期的 Oracle 版本中，绝对文件号被简单地称作“文件号”。

相对文件号是在一个表空间中的数据文件的唯一标识。对于中、小型数据库，相对文件号通常与绝对文件号相同。然而，当数据库中的数据文件超过某个极限时（一般为 1023），相对文件号将不同于绝对文件号。你能在许多数据字典视图中定位相对文件号。

### 10.1.1 确定数据文件数量

数据库中的 **SYSTEM** 表空间至少需要一个数据文件；一个小型系统也许只有一个单独的数据文件。一般来说，保持较少的大数据文件比拥有许多的小数据文件更好一些，因为你只能保持同时打开较少的文件。

向表空间增加数据文件时，要遵从操作系统对数据文件的下列限制规定：

- 操作系统限制

每个操作系统都设定了每个过程打开文件数量的最大极限。不管所有其它的限制，当操作系统打开的文件数达到极限值时，就不能创建较多的数据文件。

- Oracle 系统限制

Oracle 规定了由任何事例打开的任何 Oracle 数据库的数据文件的最大极限，这个极限是端口指定的。

#### ■ 控制文件的上限

当你发布 CREATE DATABASE 或 CREATE CONTROLFILE 语句时，MAXDATAFILES 参数指定控制文件的数据文件部分一个初始值。稍后，如果你要增加一个文件，它的文件号超过了 MAXDATAFILE 值但小于或等于 DB\_FILES 初始参数的设定值，控制文件自动增长数据文件部分的参数值以适应更多的文件。

#### ■ 事例或 SGA 的上限

当启动一个 Oracle8i 事例，数据库的初始化参数文件显示 SGA 空间的数量，以为数据文件信息预定空间。数据文件的最大数由 DB\_FILES 初始参数控制。这个限制仅适用于这个事例的生命周期中。

**注意** DB\_FILES 的缺省值是操作系统指定的。

对于 Oracle Parallel Server（并行服务器），所有事例必须将事例数据文件上限设定为相同的值。

确定 DB\_FILES 的值时，要考虑下列问题：

- 如果 DB\_FILES 的值太低，当超出 DB\_FILES 值时你将不能增加数据文件，除非首先关闭数据库。

- 如果 DB\_FILES 的值太高，内存会被不必要地消耗掉。

理论上讲，Oracle 数据库的数据文件数量是可以不受限制的。

然而，当确定数据文件的数量时，应当考虑下列两点：

- 少量的数据文件要比大量的小的数据文件在性能上更好一些。大量的文件也增加了可恢复单元的尺寸。

- 操作系统经常强制规定一个程序能同时打开的文件的数量。Oracle 的 DBWn 程序能打开所有联机的数据文件。Oracle 也能把打开的文件描述符视为一个高速缓存，当打开的文件描述符达到操作系统定义的极限时，自动关闭文件。

Oracle 允许数据库中的数据文件比操作系统定义的极限值多；这能对性能产生消极影响。可能的话，可调整操作系统对打开文件描述符的极限值的设定，使它比数据库中的联机数据文件的数值大。

操作系统指定的一个表空间里允许的数据文件的最大数量是 1023 个文件。

**参见** 关于操作系统限制的更多信息，可参见 Oracle 文档的操作系统说明。

关于并行服务器操作系统限制的更多信息，见“Oracle8i Parallel Server Administration, Deployment, and Performance”

关于 CREATE DATABASE 或 CREATE CONTROLFILE 语句的 MAXDATAFILES 参数的更多信息，见“Oracle8i SQL Reference”。

### 10.1.2 设置数据文件尺寸

第一个数据文件（在原始的 SYSTEM 表空间里）至少必须 7M，包括初始的数据辞典和回退段。如果你安装了其它的 Oracle 产品，它们也许在 SYSTEM 表空间里还需要更多的空间（例如：在线帮助）；参见这些产品的安装说明。

### 10.1.3 适当放置数据文件

表空间的位置是由组成这个表空间的数据文件的物理位置决定的。应适当地使用计算机的硬件资源。

例如，如果有数个磁盘驱动器都可以存储数据库，那么把一个表空间的表数据存放在一个磁盘驱动器上，而把表空间的索引数据存放在另一个磁盘驱动器上是很有帮助的。以这种方式工作时，当用户查询表信息时，两个磁盘驱动器能够同时工作，同时提取表数据和索引数据。

### 10.1.4 存储数据文件与重做日志文件分开

数据文件不应当与数据库的重做日志文件存储在同一个磁盘驱动器上。如果数据文件和重做日志文件存储在同一个磁盘驱动器上，当这个磁盘驱动器发生故障时，这些文件就不能被用于你的数据库恢复过程了。

如果你有多个重做日志文件，那么丢失重做日志文件的可能性就较低了，因此在有多个重做日志文件时，可以把数据文件存储在相同的驱动器上。

## 10.2 向一个表空间创建和增加数据文件

当创建一个表空间时，你最好能够估计出数据库对象可能达到的尺寸，并且增加足够的文件或设备，以保证数据均匀地分布在所有设备上。如果需要，你能够向表空间中创建和增加数据文件，以增加分配给表空间以及数据库的磁盘总量。

为了向表空间增加数据文件，要使用 `ALTER TABLESPACE...ADD DATAFILE` 语句。你必须要有向一个表空间增加数据文件的 `ALTER TABLESPACE` 系统许可权。

下列语句是向 `RB_SEGS` 表空间创建一个新数据文件：

```
ALTER TABLESPACE rb_segs
ADD DATAFILE '/u02/oracle/rbdb1/rb_segs03.dbf' SIZE 1M;
```

如果你向一个表空间增加新数据文件而没有完全地指定文件名，Oracle 是在默认的数据库目录下或当前的目录下创建数据文件，这取决于你的操作系统。Oracle 推荐你始终要完全给定数据文件的名称。要确信新的文件名不与其它文件名冲突，除非你想使用已存在的文件，而旧的文件内容将被覆盖。

## 10.3 变更数据文件的尺寸

本节讲述修改数据文件的各种方法，其中包括下列标题：

- 允许和禁止数据文件的自动扩展
- 手工调整数据文件的尺寸

### 10.3.1 允许和禁止数据文件的自动扩展

用户可创建或修改已存在的数据文件，使它们在数据库需要更多空间时自动增加文件尺寸。这些文件以指定的增长量增长到指定的最大值。

设置你的数据文件自动扩展将导致下列结果：

- 减少了数据表空间用完磁盘空间时要立即人工干涉的需要
- 保证不会因为分配区域的失败而中断应用

要查明数据文件是否能自动扩展，可查询 `DBA_DATA_FILES` 视图，并且测试 `AUTOEXTENSIBLE` 项。

当用下列 `SQL` 语句创建数据文件时，可以指定一个 `AUTOEXTEND ON` 子句来指定数据文件的自动扩展：

- `CREATE DATABASE`
- `CREATE TABLESPACE`
- `ALTER TABLESPACE`

用户可允许或禁止已存在文件的自动扩展特性，或用 `SQL` 语句 `ALTER DATABASE` 手工调整数据文件的尺寸。

下面的例子允许增加到 `USERS` 表空间的数据文件自动扩展尺寸：

```
ALTER TABLESPACE users
ADD DATAFILE '/u02/oracle/rbdb1/users03.dbf' SIZE 10M
AUTOEXTEND ON
NEXT 512K
MAXSIZE 250M;
```

`NEXT` 值是文件扩展时加到文件的最小增长量。`MAXSIZE` 值是文件能够自动扩展到的最大尺寸。

下面的例子可禁止数据文件的自动扩展：

```
ALTER DATABASE DATAFILE '/u02/oracle/rbdb1/users03.dbf'
AUTOEXTEND OFF;
```

参见 关于创建或修改数据文件的 `SQL` 语句的更多信息，见“Oracle8i SQL Reference”。

### 10.3.2 手工调整数据文件的尺寸

用 `ALTER DATABASE` 语句，可手工增加或减少数据文件的尺寸。

由于能够变更数据文件的尺寸，你可以不增加数据文件而使数据库增加更多的空间。如果你担心数据文件要达到数据库所允许的最大数量时，这个功能是很有益处的。

手动减小数据文件的尺寸使你能够收回数据库中不使用的空间，这有助于纠正估计空间需求时出现的错误。

在下面的例子中，假定数据文件 `/u02/oracle/rbdb1/stuff01.dbf` 已经扩展到 250M。然而，由于它的表空间现在存储的对象较少，数据文件的尺寸能够减小。

下列语句可减小数据文件的尺寸：

```
/u02/oracle/rbdb1/stuff01.dbf:
ALTER DATABASE DATAFILE '/u02/oracle/rbdb1/stuff01.dbf'
RESIZE 100M;
```

**注意** 不是总可以把一个文件的尺寸减小到一个指定值。

参见 关于实现调整文件尺寸的更多信息，见“Oracle8i Migration”。

## 10.4 改变数据文件的可用性

本节讲述改变数据文件可用性的方法，其中包括下列标题：

- 使数据文件联机或以 ARCHIVELOG 模式脱机
- 使数据文件以 NOARCHIVELOG 模式脱机

在极个别情况下，也许需要使某些数据文件联机（使它们有效）或使某些文件脱机（使它们无效）。例如：当 Oracle 写一个数据文件出现问题时，它能使这个文件自动脱机。也许需要使一个损坏的数据文件脱机然后手工使它联机。

**注意** 除了 SYSTEM 表空间的文件，还可以通过使一个表空间脱机的方法使这个表空间的所有数据文件都暂时无效。必须使这些文件留在表空间中以使表空间再次联机。

关于使一个表空间脱机的更多信息，参见第 9 章“使表空间脱机”。

脱机的数据文件不能被访问。使一个数据文件联机到一个只读表空间，则这个文件是可读的，但不能写，除非它关连的表空间转到可读写状态。只读表空间的文件能用 ALTER DATABASE 语句的 DATAFILE 选项单独地联机或脱机。

要使一个数据文件联机或脱机，必须有 ALTER DATABASE 的系统许可权，只有当数据库是在高级模式下打开时，才能完成这些操作。

### 10.4.1 使数据文件联机或以 ARCHIVELOG 模式脱机

使一个单独的数据文件联机，可用 ALTER DATABASE 语句和 DATAFILE 子句。下面的语句可使指定的数据文件联机：

```
ALTER DATABASE DATAFILE '/u02/oracle/rbdb1/stuff01.dbf' ONLINE;
```

使这个文件脱机，用下面的语句：

```
ALTER DATABASE DATAFILE '/u02/oracle/rbdb1/stuff01.dbf' OFFLINE;
```

**注意** 用 ALTER DATABASE 的这个选项，数据库必须是在 ARCHIVELOG 模式。这个要求可防止偶然地丢失数据文件，因为以 NOARCHIVELOG 模式脱机的数据文件很可能导致丢失文件。

**参见** 关于在恢复介质期间使数据文件联机的更多信息，参见“Oracle8i Backup and Recovery Guide”。

### 10.4.2 使数据文件以 NOARCHIVELOG 模式脱机

当数据库在 NOARCHIVELOG 模式下使数据文件脱机时，可用 ALTER DATABASE 语句及 DATAFILE 和 OFFLINE DROP 子句。它允许你使数据文件脱机并立即丢弃它。这个功能是有用的，例如，如果数据文件只含有临时段的数据，而且还没有被备份，并且数据库是在 NOARCHIVELOG 模式。

下面的语句使指定的数据文件脱机：

```
ALTER DATABASE DATAFILE '/u02/oracle/rbdb1/users03.dbf' OFFLINE DROP;
```

## 10.5 重命名和重新部署数据文件

重新命名数据文件，或变更它们的名称或重新部署它们。某些选项和过程将在下面的段落中讲述：

### ■ 为一个单独的表空间重命名和重新部署数据文件

例如，重命名表空间 1 的文件名 1 和文件名 2，当数据库的其余部分打开时。

### ■ 为多个表空间重命名和重新部署数据文件

例如，重命名表空间 1 的文件名 1 和表空间 2 的文件名 2，当数据库被安装但没打开时。

**注意** 当重命名或重新部署 SYSTEM 表空间的数据文件时，必须使用第二个选项，因为你不能使 SYSTEM 表空间脱机。

当在重命名和重新部署数据文件的这些过程时，仅仅是改变了记录在数据库的控制文件里的数据文件指针，并没有重新命名任何操作系统文件，也没有做操作系统级的文件拷贝。因此，重命名和重新部署数据文件包括几个步骤，在完成这些过程之前要读懂这些步骤和例子。

### 10.5.1 为一个单独的表空间重命名和重新部署数据文件

为一个单独的表空间重命名和重新部署数据文件需要一些过程。要为一个单独的表空间重命名数据文件必须有 ALTER TABLESPACE 系统许可权。

#### 在一个单独的表空间里重命名数据文件

在一个单独的表空间重命名数据文件要遵从这个过程。

1. 使包含要改名的数据文件的非 SYSTEM 表空间脱机。
  2. 用操作系统语句重命名数据文件。
  3. 确认新的、完全指定的文件名不同于老文件名。
  4. 用 ALTER TABLESPACE 语句及 RENAME DATAFILE 选项改变数据库中的文件名。
- 例如：下面的语句把数据文件文件名 1 和文件名 2 重命名为相应的文件名 3 和文件名 4：

```
ALTER TABLESPACE users
  RENAME DATAFILE '/u02/oracle/rbdb1/users1.dbf',
                  '/u02/oracle/rbdb1/users2.dbf',
  TO '/u02/oracle/rbdb1/users01.dbf',
    '/u02/oracle/rbdb1/users02.dbf';
```

新的文件必须已经存在；这个语句不创建文件。另外，必须始终提供完整的数据文件名（包括它们的路径）以正确识别新旧数据文件，特别是当旧数据文件出现在数据词典的 DBA\_DATA\_FILES 的视图时。

#### 在一个单独的表空间里重新部署和重命名数据文件

这里有一个例子说明重新部署一个数据文件包含的步骤。

假定有下列条件：

- 一个打开的数据库有一个名为 USERS 的表空间，它是由位于一台计算机的同一个磁盘里的多个数据文件组成的。

■ **USERS** 表空间的数据文件将被重新部署到不同的磁盘驱动器上。

■ 目前连接了打开的数据库的系统管理员权限。

这些步骤是：

1. 识别所关心的数据文件名。

下面查询 **USERS** 表空间的数据字典视图 **DBA\_DATA\_FILES** 列出的数据文件名称和相应的尺寸（单位：字节）：

```
SELECT file_name, bytes FROM sys.dba_data_files
WHERE tablespace_name = 'USERS';
```

| FILE_NAME                     | BYTES     |
|-------------------------------|-----------|
| /U02/ORACLE/RBDB1/USERS01.DBF | 102400000 |
| /U02/ORACLE/RBDB1/USERS02.DBF | 102400000 |

2. 备份数据库。

在使数据库结构发生任何变化前，例如在一个或多个表空间中重命名和重新部署数据文件，总是要完全备份数据库。

3. 使包含数据文件的表空间脱机，或停止运行数据库并重新启动和安装它，使它保持关闭状态，或者选择关闭表空间的数据文件。

4. 把数据文件拷贝到它们的新位置，并用操作系统命令重命名它们。

**注意** 可通过使用 **HOST** 命令来执行操作系统命令以拷贝一个文件。

5. 在 **Oracle** 内重命名数据文件。

组成 **USERS** 表空间的数据文件的指针是记录在相关数据库的控制文件里，现在必须把它从旧的文件名改变到新的文件名。

如果表空间是脱机的但数据库是打开的，用 **ALTER TABLESPACE...RENAME DATAFILE** 语句。如果数据库被安装但是关闭的，用 **ALTER DATABASE...RENAME FILE** 语句。

```
ALTER TABLESPACE users
  RENAME DATAFILE '/u02/oracle/rbdb1/users01.dbf',
                  '/u02/oracle/rbdb1/users02.dbf',
  TO '/u03/oracle/rbdb1/users01.dbf',
    '/u04/oracle/rbdb1/users02.dbf';
```

6. 使表空间联机，或停止运行数据库后再重新启动。

如果 **USERS** 表空间是脱机的并且数据库是打开的，使表空间联机。如果数据库被安装并在关闭状态，可打开数据库。

7. 备份数据库。在使数据库发生任何结构变化后，总是要完成一个立即的、全面的备份。

## 10.5.2 为多个表空间重命名和重新部署数据文件

能够使用 **ALTER DATABASE** 语句及 **RENAME FILE** 选项为一个或多个表空间重命名和重新部署数据文件。这个选项只是给你一个选择机会，如果你想在一次操作中为多个表空间重命名或重新部署数据文件，或重命名或重新部署 **SYSTEM** 表空间的数据文件。如果数据库

必须保持打开状态，可考虑用前面章节讲述的相关内容代替这个过程。

为了在一次操作中为多个表空间重命名数据文件，或重命名 **SYSTEM** 表空间的数据文件，必须有使用 **ALTER DATABASE** 的系统许可权。

在多个表空间里重命名数据文件有下列步骤。

1. 确保数据库被安装并且是关闭的。
2. 用操作系统命令把重命名的数据文件以新的名称拷贝到新的位置。
3. 确信新拷贝的数据文件有完全指定的文件名且不同于当前正在使用的数据文件名。
4. 用 **ALTER DATABASE** 重命名数据库的控制文件里的文件指针。

例如：下面的语句把数据文件文件名 1 和文件名 2 重命名为相应的文件名 3 和文件名 4：

```
ALTER DATABASE
  RENAME FILE '/u02/oracle/rbdb1/sort01.dbf',
             '/u02/oracle/rbdb1/user3.dbf',
  TO '/u02/oracle/rbdb1/temp01.dbf',
     '/u02/oracle/rbdb1/users03.dbf';
```

新的文件必须已经存在；这个语句不创建文件。另外，必须始终提供完整的数据文件名（包括它们的路径）以正确识别新、旧数据文件，特别是当旧数据文件出现在数据辞典的 **DBA\_DATA\_FILES** 的视图时。

## 10.6 校验数据文件中的数据块

如果想用校验和校验数据块来配置 **Oracle**，那么把初始参数 **DB\_BLOCK\_CHECKSUM** 设置为 **TRUE**。这个参数的值能够动态改变或在初始参数文件里设置。**DB\_BLOCK\_CHECKSUM** 的缺省值是 **FALSE**。不管这个参数的设置，系统表空间将始终用校验和校验数据块。

当你允许块检查时，**Oracle** 为每个写入磁盘的块计算校验和。每个数据块都计算校验和，包括临时块。

**DBWn** 程序为每个块计算校验和并把它们存储在块头。直接装入程序也计算块的校验和。

**Oracle** 下一次读数据块时，用校验和检测块是否损坏。如果检测到有损坏，**Oracle** 返回信息 **ORA-01578**，并把这个损坏信息写入一个踪迹文件。

**警告** 设置 **DB\_BLOCK\_CHECKSUM** 为 **TRUE** 时可能引起附加功能。仅当 **Oracle** 支持个人诊断数据问题的前提下才能把这个参数设置为 **TRUE**。

## 10.7 查看数据文件的相关信息

下列数据辞典视图提供了关于数据库中的数据文件的有用信息：



| 视图                                 | 描述   |
|------------------------------------|--|
| DBA_DATA_FILES                     | 提供了关于数据文件的描述信息，包括它属于哪一个表空间和文件的 ID 号。文件的 ID 号能用于连接提供详细信息的其它视图 |
| USER_EXTENTS,<br>DBA_EXTENTS       | 列出数据库中组成的所有段的区域，包括数据文件的文件 ID 号包含的区域                          |
| USER_FREE_SPACE,<br>DBA_FREE_SPACE | 列出所有表空间的空闲区域，包括数据文件的文件 ID 号包含的区域                             |
| V\$DATAFILE                        | 包含控制文件中的数据文件信息   |
| V\$DATAFILE_HEADER                 | 包含数据文件头的信息   |

下面的例子说明了这些视图之一的 V\$DATAFILE 视图的用途。

假定你正在使用包含两个表空间 **SYSTEM** 和 **USERS** 的数据库。**USERS** 由两个文件组成，文件 1（100MB）和文件 2（200MB）；表空间已经被正常脱机。现在你查询 V\$DATAFILE 要查看数据库中数据文件的状态信息：

```
SELECT name,
       file#,
       status,
       checkpoint_change# "CHECKPOINT"
FROM   V$DATAFILE;
```

| NAME      | FILE# | STATUS  | CHECKPOINT |
|-----------|-------|---------|------------|
| -----     | ----- | -----   | -----      |
| filename1 | 1     | SYSTEM  | 3839       |
| filename2 | 2     | OFFLINE | 3782       |
| filename3 | 3     | OFFLINE | 3782       |

**FILE#**栏列出了每个数据文件的文件编号；在 **SYSTEM** 表空间中的第一个数据文件是与数据库一起创建的，文件号总是 1。**STATUS** 栏列出了数据文件的其它信息。如果数据文件属于 **SYSTEM** 表空间，它的状态就是 **SYSTEM**（除非它要求恢复）。如果在非 **SYSTEM** 表空间的数据文件是联机的，它的状态就是 **ONLINE**。如果在非 **SYSTEM** 表空间的数据文件是脱机的，它的状态或者是 **OFFLINE** 或者是 **RECOVER**。**CHECKPOINT** 栏列出了记录数据文件最新检测点的最终的 SCN 值。

参见 关于这些视图的完整论述，见 “Oracle8i Reference”。

# 11 管理回退段

本章讲述了怎样管理回退段，包括下列章节：

- 管理回退段的方针
- 创建回退段
- 改变回退段
- 明确地给回退段分配事务处理
- 删除回退段
- 监视回退段信息

参见 如果你正在用并行服务器方式使用 Oracle，关于在这种环境下创建回退段的信息见 “Oracle8i Parallel Server Administration, Deployment, and Performance”。

## 11.1 管理回退段的方针

本节讲述在创建或管理你的数据库的回退段前的思考方针，包括以下专题：

- 使用多个回退段
- 选择公用回退段和私用回退段
- 自动获得指定的回退段
- 近似的回退段尺寸
- 创建具有许多相等尺寸区域的回退段
- 为每个回退段设置一个最佳的区域数
- 把回退段放置在一个单独的表空间

每个数据库都包含一个或多个回退段，它们是数据库的组成部分，负责记录事务处理的行为，使每个事务处理可以被回退。使用回退段可以提供读取数据的一致性、回退事务处理结果以及恢复数据库。

参见 关于回退段的更多信息，见 “Oracle8i Concepts”。

### 11.1.1 使用多个回退段

使用多个回退段是在许多的区段上分配争用回退段，并改进系统性能。下列情况下需要多个回退段：

- 当创建数据库时，在 **SYSTEM** 表空间中创建一个名为 **SYSTEM** 的单个回退段。你能在非 **SYSTEM** 表空间中创建任何对象，但不能向对象里写，除非你在非 **SYSTEM** 表空间中创建了至少一个附加的回退段并使它联机（为非 **SYSTEM** 对象）。
- 当许多事务处理并发进行时，在同一时刻将产生较多的回退信息。用初始参数 **TRANSACTIONS**，可以指示在这个事例中你期望的同时进行的事务处理的数量；用初始化参数 **TRANSACTIONS\_PER\_ROLLBACK\_SEGMENT**，可以规定你期望的

每个回退段所必须处理的事务数量。那么，当一个事例打开一个数据库时，它试图至少获得 `TRANSACTIONS_PER_ROLLBACK_SEGMENT` 回退段已完成最大的事务处理量。因此，在设置参数后，要创建 `TRANSACTIONS_PER_ROLLBACK_SEGMENT` 回退段。

一个事例除了它需要的其它回退段外，总是要获得 `SYSTEM` 回退段。然而，如果有多个回退段，Oracle 试图只用 `SYSTEM` 回退段处理特定的系统事务，而分配其它的回退段处理用户的事务；如果非 `SYSTEM` 回退段有太多的事务处理，Oracle 也使用 `SYSTEM` 回退段。

参见 在 Oracle 并联服务器环境中启动事例，除了 `SYSTEM` 回退段外，必须使每个事例能够访问它自己的回退段，附加的详细信息见 “Oracle8i Parallel Server Administration, Deployment, and Performance”。

关于初始化参数 `TRANSACTIONS` 和 `TRANSACTIONS_PER_ROLLBACK_SEGMENT` 的更多信息，见 “Oracle8i Reference”。

### 11.1.2 选择公用回退段和私用回退段

私用回退段可以被一个事例明确获得，只要当事例打开数据库时，该段在初始参数文件里的 `ROLLBACK_SEGMENTS` 参数下被命名。私用回退段也能够通过手工键入命令语句指定它联机来获得。

公用回退段形成一个回退段池，任何事例都能够用它请求一个回退段。

用并行服务器的数据库可以只有一个公用回退段，当段的数量足够多的时候——每个事例打开数据库，除了它的 `SYSTEM` 回退段外，还能够获得至少一个回退段。用 Oracle 并联服务器时，也可以使用私用回退段。

如果数据库没有并行服务器选项，则公用回退段和私用回退段是一样的。

### 11.1.3 自动获得指定的回退段

当一个事例启动时，它获得默认的 `TRANSACTIONS/TRANSACTIONS_PER_ROLLBACK_SEGMENT` 回退段。如果你想保证事例获得的是有特殊尺寸或特殊表空间的回退段，可在事例的参数文件里用 `ROLLBACK_SEGMENTS` 参数命名指定的回退段。

事例获得的所有回退段都列在这个参数下，甚至超过了 `TRANSACTIONS/TRANSACTIONS_PER_ROLLBACK_SEGMENT` 指定的段。

### 11.1.4 近似的回退段尺寸

所有的回退段尺寸应该是基于数据库所发布的最普遍的事务处理量。一般情况下，当数据库有许多较小的回退段时，短的事务处理历程性能较好；当运行较长的事务处理时，像批处理工作，用较长的回退段完成较好。回退段一般能很容易地处理任何尺寸的事务，然而，在极端的情况下——事务处理过程或者很短或者很长，用户也许想使用与之近似的回退段尺寸。

当系统只运行短的事务处理时，回退段应当是小的，以使它们能始终隐藏在主存储器中。如果回退段足够的小，根据 LRU 计算法则它们很可能被隐藏在 SGA。由于只占用很少的 I/O 口，数据库的性能将被改善。小的回退段的主要缺点是增加了 “Snap too old” 错误的可能性，

当运行一个包含被其它事务处理频繁更新记录的长查询时，这个错误的出现是因为当其它更新条目缠绕着回退段时，回退条目所需要的读取一致性被重写。考虑到这一点，设计一个应用事务时，应使它们的工作单元短小，以避免这个问题。

与之相对照，长的事务处理工作最好用较大的回退段，因为长的事务处理的回退条目能适合大回退段的预先定位范围。

当数据库系统应用同时发布混合了很短和很长的事务处理时，如果事务处理被明确指定的回退段是基于事务处理/回退段尺寸的，那么系统性能将被优化。你能够把回退段的动态分配范围减到最小或者切断。对于大部分系统没有这个必要，这只是为极大或极小的事务处理考虑的。

当发布混合了极小和极大的事务处理时，为了优化性能，要使大量的回退段与各种事务处理尺寸近似（例如小型的，中型的和大型的）。大部分回退段应当与典型的事务处理相对应，较少的回退段用于非典型的事务处理。然后，把每个这样的回退段设置为 **OPTIMAL**，以使回退段能返回到它们想要的尺寸，如果它们不得不增长的话。

应当告诉用户关于不同的事务处理应相应设置不同的回退段。明确指定一个事务处理到特殊的回退段常常没有什么益处；但是你可以为一个非典型的事务处理指定一个为这样的事务处理创建的适当的回退段。例如，能够把一个包含大的批处理工作的事务处理指定到一个大的回退段。

当混合型的事务处理不普遍时，每个回退段应当是数据库的最大表尺寸的 10%，因为大部分的 **SQL** 语句只影响一个表的 10%或更少；因此，回退段的这个尺寸对于用大部分 **SQL** 语句完成的存储行为是足够的。

一般的讲，应该为回退段设置较高的 **MAXEXTENTS** 值；它允许回退段在需要时分配后续空间。

#### 11.1.5 创建具有许多相等尺寸区域的回退段

每个回退段的总的分配空间应当被分为许多个相等的尺寸区域。如果一个事例的每个回退段都有 10 到 20 个相等的尺寸区域，基本上能够得到理想的 I/O 回退性能。

在确定了所希望的回退段的总初始尺寸和这个段的初始区域数后，可用下面的公式计算回退段每个区域的尺寸：

$$T/n = s$$

这里：T = 回退段的总初始尺寸，以字节为单位

n = 初始分配的区域数

s = 初始分配每个区域的计算尺寸，以字节为单位

在 s 计算出来后，创建回退段并指定存储参数 **INITIAL** 和 **NEST** 为 s，**MINEXTENTS** 为 n。**PCTINCREASE** 不能为回退段指定，因此默认为 0。如果一个区域的尺寸 s 不是数据块尺寸的准确倍数，那么它自动增加到下一个整倍数。

#### 11.1.6 为每个回退段设置一个最佳的区域数

当为每个回退段设置了 **OPTIMAL** 参数时，应当小心系统运行的事务处理的种类。对于一个经常执行长的事务处理的系统，**OPTIMAL** 应当是大的，使 **Oracle** 没有必要频繁的收缩

或分配区域。同样，对于在激活的数据上执行较长查询的系统，OPTIMAL 也应当是大的，以避免“Snapshot too old”的错误。对于主要执行短的事务处理和查询的系统，OPTIMAL 应当是较小的，使回退段保持足够的小以能被隐藏在存储器中，从而改进系统性能。

通过监视 V\$ROLLNAME 和 V\$ROLLSTAT 动态性能视图，可收集有用的状态信息，以决定对 OPTIMAL 的适当的设置。见 11.6.2 节“回退段统计”。

#### 11.1.7 把回退段放置在一个单独的表空间

可能的话，创建一个表空间专门保存所有的回退段。这种方式使所有的回退段数据与其他类型的数据分开存储。创建“回退段”表空间可提供下列益处：

- 保存回退段的表空间能始终保持联机，因此，回退段在所有时间都具有最大的组合存储能力。注意，如果某些回退段无效了，全部的数据库操作都将受影响。
- 因为有激活回退段的表空间不能被脱机，所以指定一个表空间存放所有的回退段，可以保证使存储其它数据的表空间脱机而不必关心数据库的回退段问题。
- 如果表空间包含经常分配和收回区域的回退段时，表空间的空闲区域可能会非常零碎。

## 11.2 创建回退段

要创建回退段，必须有 CREATE ROLLBACK SEGMENT 的系统许可权，使用 CREATE ROLLBACK SEGMENT 语句。包含新回退段的表空间必须是联机的。回退段通常是用创建程序或过程作为数据库的一部分创建起来的，但你可以在以后增加更多的回退段。

本节包括下列与创建回退段相关的专题：

- CREATE ROLLBACK SEGMENT 语句
- 使新回退段联机
- 在创建一个回退段时设置存储参数

#### 11.2.1 CREATE ROLLBACK SEGMENT 语句

下面的语句是在 RBSSPACE 表空间创建一个名为 RBS\_02 的回退段，使用的是这个表空间默认的存储参数。由于这不是一个并行服务器环境，没有必要指定 PRIVATE 或 PUBLIC。默认值是 PRIVATE。

```
CREATE ROLLBACK SEGMENT rbs_02 TABLESPACE rbsspace;
```

参见 用于管理回退段的 SQL 语句的准确句法、限制和授权要求，见“Oracle8i SQL Reference”。

#### 11.2.2 使新回退段联机

新回退段初始是脱机的。为使一个事例的事务处理能使用它，必须发布 ALTER ROLLBACK SEGMENT 语句使它联机并使它有效。更多的信息见 11.3.3 “改变回退段的联机/脱机状态”。

如果你创建了一个私用回退段，应当把这个新回退段的名称加入到数据库的初始化参数

文件的 `ROLLBACK_SEGMENTS` 参数下，使事例启动时能够捕捉到这个私用回退段。例如，如果创建了两个新的私用回退段，命名为 `RBS_01` 和 `RBS_02`，参数文件的 `ROLLBACK_SEGMENTS` 参数应与下面的类似：

```
ROLLBACK_SEGMENTS = (RBS_01, RBS_02)
```

### 11.2.3 在创建一个回退段时设置存储参数

假定你想要用存储参数和优化尺寸设置创建一个名为 `RBS_01` 的回退段，可按下面的步骤进行：

- 分配回退段一个 100K 的初始区域。
- 分配回退段第二个 100K 的初始区域。
- 回退段的优化尺寸是 4M。
- 最小区域数和段创建时初始分配的区域数是 20。
- 回退段能够分配的最大区域数，包括初始区域数，是 100。

下面的语句用这些特性创建了一个回退段：

```
CREATE PUBLIC ROLLBACK SEGMENT rbs_01
    TABLESPACE rbsspace
    STORAGE (
        INITIAL 100K
        NEXT 100K
        OPTIMAL 4M
        MINEXTENTS 20
        MAXEXTENTS 100 );
```

你不能为存储参数 `PCTINCREASE` 设值，对于回退段它始终是 0。`OPTIMAL` 是回退段的唯一存储参数。关于存储参数的讨论见第 12 章。

Oracle 特别给出下列建议：

- 将 `INITIAL` 和 `NEST` 设置成相同的值，以保证所有的区域尺寸相同。
- 创建一个较大的初始区域数，使动态分配区域的可能性降到最小。`MAXEXTENTS = 20` 比较好。
- 避免设置 `MAXEXTENTS = UNLIMITED`，因为这可能由于一个程序错误引起回退段或数据文件的不必要的扩展。如果你指定了 `UNLIMITED`，要知道段的这些区域必须要有至少 4 个数据块。如果你想转变一个回退段，把它的 `MAXEXTENTS` 参数从给定值变为 `UNLIMITED`，如果回退段的任何区域中有小于 4 个数据块的，它都不能被转换。如果你想要从给定值变为 `UNLIMITED`，并且在一个区域中有小于 4 个的数据块，你只能选择删除这个回退段并重新创建它。

## 11.3 改变回退段

本节讨论了你能对回退段采取的各种维护行为。所有这些维护行为用 `ALTER ROLLBACK SEGMENT` 语句。使用这个语句，你必须有 `ALTER ROLLBACK SEGMENT` 的系统许可权。

本节介绍以下专题：

- 改变回退段存储参数
- 手动收缩回退段
- 改变回退段的联机/脱机状态

### 11.3.1 改变回退段存储参数

在创建回退段后你能改变它的某些存储参数。你也许想变更 **OPTIMAL** 或 **MAXEXTENTS** 的参数值。下面的语句修改了 **RBS\_01** 回退段所能分配区域的最大数。

```
ALTER ROLLBACK SEGMENT rbs_01  
STORAGE (MAXEXTENTS 120);
```

你能改变 **SYSTEM** 回退段的设置，包括 **OPTIMAL** 参数，就像你能改变任何其它回退段的这些参数一样。

### 11.3.2 手动收缩回退段

你能用 **ALTER ROLLBACK SEGMENT** 语句手动减小一个回退段的尺寸，你要收缩的这个回退段必须是联机的。

下面的语句将回退段 **RBS1** 收缩到 100K：

```
ALTER ROLLBACK SEGMENT rbs1 SHRINK TO 100K;
```

这个语句试图把回退段的尺寸减少到一个特定值，但是，如果一个区域是激活的将不能被重新分配，收缩将停止。

### 11.3.3 改变回退段的联机/脱机状态

本部分讲述了使回退段联机和脱机的各方面，包括下列专题：

- 使回退段联机
- 使回退段脱机

每个回退段都有两种状态，或者是联机状态，对事务处理有效；或者是脱机状态，对事务处理无效。正常情况下，回退段是联机的，可被事务处理使用。

在下列情况下你也许希望使联机的回退段脱机：

- 当你想要使一个包含回退段的表空间脱机时，如果所包含的回退段正在被事务处理使用，你不能使这个表空间脱机。为了避免相关的回退段被使用，你可以使表空间脱机前先使它们脱机。
- 当你想要删除一个回退段，但由于事务处理正在使用而不能进行时。为了避免回退段被使用，在删除它以前你可以使它先脱机。

**注意** 你不能使 **SYSTEM** 回退段脱机。

你也许后来还想要使一个脱机的回退段重新联机，以使事务处理能够使用它。当一个回退段创建时，它的初始状态是脱机的，因此在事例的事务处理能使用它之前，你必须明确地使新创建的回退段联机。你可以通过任何事例访问包含回退段的数据库，而使一个脱机的回退段联机。

## 使回退段联机

你仅可以使当前状态为 **OFFLINE** 或 **PARTLY AVAILABLE**（就像在数据辞典 **DBA\_ROLLBACK\_SEGS** 视图中看到的一样）的回退段联机。使一个脱机的回退段联机，用 SQL 语句 **ALTER ROLLBACK SEGMENT** 及 **ONLINE** 选项。

**使一个 PARTLY AVAILABLE 的回退段联机** 一个在 **PARTLY AVAILABLE** 状态的回退段包含有疑问的或被恢复的分布式事务处理数据，并且这些数据还将要被恢复。能在数据辞典 **DBA\_ROLLBACK\_SEGS** 视图中查看它的状态为 **PARTLY AVAILABLE**。回退段通常是保持在这种状态，除非这个事务处理由 **REO** 自动解决，或由一个 **DBA** 手动解决。但是，有时你也许会发现所有的回退段都在 **PARTLY AVAILABLE** 的状态。在这种情况下，你可以使 **PARTLY AVAILABLE** 的回退段联机，就像上面所讲述的一样。

回退段的某些用于有疑问的事务处理的资源不能被访问，直到这个事务处理被解决。这导致回退段也许不得不增大，如果指定给它的其它事务处理需要附加的空间。

由于使一个 **PARTLY AVAILABLE** 的回退段联机要进行选择，你也许发现创建一个新的临时回退段会更容易些，除非有疑问的事务处理被解决。

**使回退段自动联机** 如果你希望无论什么时候启动数据库时，回退段都能自动联机，就在数据库的参数文件中的 **ROLLBACK\_SEGMENTS** 参数下加入这个回退段的名称。

使回退段联机：下面的语句使回退段 **USER\_RS\_2** 联机：

```
ALTER ROLLBACK SEGMENT user_rs_2 ONLINE;
```

当你使一个回退段联机后，它在数据辞典视图 **DBA\_ROLLBACK\_SEGS** 中的状态就是 **ONLINE**。要查询一个回退段的状态，见 11.6.1 节。

## 使回退段脱机

要使一个联机的回退段脱机，使用 **ALTER ROLLBACK SEGMENT** 语句及 **OFFLINE** 选项。这个回退段在数据辞典视图 **DBA\_ROLLBACK\_SEGS** 中的状态必须是 **ONLINE**，并且这个回退段必须是当前事例所获得的。

下面的语句使回退段 **USER\_RS\_2** 脱机：

```
ALTER ROLLBACK SEGMENT user_rs_2 OFFLINE;
```

如果你要使一个不包含激活的回退条目的回退段脱机，**Oracle** 立即使这个段脱机并将它的状态改变为“**OFFLINE**”。

相反，如果你要使一个包含激活的事务处理（本地，远程或分布式的）的回退数据的回退段脱机时，**Oracle** 先要使回退段对进一步事务处理无效，然后在所有的事务处理对回退段的使用完成后才使它脱机。除了使它脱机的事例外，其它任何事例都不能使这个回退段联机，直到这些事务处理完成。在这期间，该回退段在视图 **DBA\_ROLLBACK\_SEGS** 中的状态保持为 **PENDING OFFLINE**；而且，在视图 **V\$ROLLSTAT** 中的状态也是 **PENDING OFFLINE**。查看回退段状态的更多信息，见 11.6.1 节。

使一个回退段脱机并引起它的状态改变为 **PENDING OFFLINE** 的事例，在任何时刻都可以使这个回退段返回到联机状态；如果回退段返回到联机状态，它将一切功能正常。

在你使一个公用的或私用的回退段脱机后，它将保持脱机状态，直到你明确指定它返回



联机状态或重新启动事例。

## 11.4 明确地给回退段分派事务处理

用 `SET TRANSACTION` 语句及 `USE ROLLBACK SEGMENT` 子句，能明确地把一个事务处理分派给一个指定的回退段。

下列原因事务处理要被明确地分配到回退段：

- 预期事务处理所产生的回退信息的数量与指定的回退段当前的区域数相匹配。
- 回退段的附加区域没有必要动态分配（并且随后再被缩短），这样将降低系统的全部性能。

要明确地给回退段分配事务处理，回退段必须与当前事例联机，并且 `SET TRANSACTION USE ROLLBACK SEGMENT` 语句必须是事务处理的第一个语句。如果指定的回退段没有联机，或 `SET TRANSACTION USE ROLLBACK SEGMENT` 语句不是事务处理的第一个语句，则返回一个错误码。

例如，如果你要开始一个包含大量工作（超过了大部分事务处理的工作量）的事务处理，你可以把这个事务处理像下面这样分配给一个较大的回退段：

```
SET TRANSACTION USE ROLLBACK SEGMENT large_rs1;
```

当事务处理被提交后，Oracle 将自动分配下一个事务处理给任何有效的回退段，除非用户明确地给一个特定的回退段指定新的事务处理。

## 11.5 删除回退段

当回退段的区域在磁盘上变得太零碎，或者回退段需要被重新部署在不同的表空间时，你可以删除它。删除回退段前，要保证它是 `OFFLINE` 状态。如果你想删除的回退段在任何别的状态，你就不能删除它。如果段的状态是 `INVALID`，说明它已经被删除。

要删除一个回退段，用 `DROP ROLLBACK SEGMENT` 语句，并且你要有 `DROP ROLLBACK SEGMENT` 的系统许可权。下面的语句删除了 `RBS1` 回退段：

```
DROP ROLLBACK SEGMENT rbs1;
```

**注意** 如果一个指定在 `ROLLBACK_SEGMENTS` 参数下的回退段被删除时，要编辑数据库的参数文件，保证从 `ROLLBACK_SEGMENTS` 参数下除去被删除的回退段的名称。如果在下次事例启动前没有完成这个步骤，启动将因不能获得删除的回退段而失败。

在一个回退段被删除后，它的状态变为 `INVALID`。再创建回退段时，它将占据删除的回退段所腾出的行，如果新回退段是有效的，删除的回退段的行将不再出现在 `DBA_ROLLBACK_SEGS` 视图中。

11.6 监视回退段信息

本节介绍了用于得到和监视回退段信息的视图，提供使用它们的相关信息和例子。  
本节介绍以下专题：

- 显示回退段信息
- 回退段的统计
- 显示全部回退段
- 显示是否有回退段已经脱机

11.6.1 显示回退段信息

数据辞典视图 `DBA_ROLLBACK_SEGS` 储存数据库的回退段的相关信息。例如：下面的查询列出了数据库中每个回退段的名称、所属表空间和状态：

```
SELECT segment_name, tablespace_name, status
      FROM sys.dba_rollback_segs;
```

| SEGMENT_NAME | TABLESPACE_NAME | STATUS |
|--------------|-----------------|--------|
| SYSTEM       | SYSTEM          | ONLINE |
| PUBLIC_RS    | SYSTEM          | ONLINE |
| USERS_RS     | USERS           | ONLINE |

此外，下列辞典视图包含了数据库的段的信息，其中包括回退段：

- `USEERS_SEGMENTS`
- `DBA_SEGMENTS`

11.6.2 回退段的统计

动态性能视图 `V$ROLLSTAT` 能被用于查询，以监视回退段的统计信息。要将回退段的段号映射到它的名称，必须与 `V$ROLLSTAT` 视图相连接。

所包括的值和关注的特殊栏目：

| 名称        | 描述  |
|-----------|---|
| USN       | 回退段的段号。如果这个视图与 <code>V\$ROLLSTAT</code> 视图相连接，回退段的名称能够被确定 |
| WRITES    | 写到回退段的条目的字节数  |
| XACTS     | 激活的事务处理数  |
| GETS      | 需要的回退段头数  |
| WAITS     | 需要等待的回退段头数  |
| OPTSIZE   | 回退段的优化参数值   |
| HWMSIZE   | 回退段在使用中达到的最大字节数   |
| SHRINKS   | 回退段为了保持最佳尺寸已经完成的收缩次数                                      |
| WRAPS     | 回退段条目被从一个区域缠绕到另一个区域的次数                                    |
| EXTENDS   | 回退段不得不获得新区域的次数  |
| AVESHRINK | 在一次收缩中重新释放出的平均字节数   |
| AVEACTIVE | 随时间测量的回退段激活区域的平均字节数                                       |

这些启动信息在系统启动时被重新复位。

对这个视图的特别查询有助于确定对 **OPTIMAL** 参数的最佳设置。假定一个事例具有尺寸区域相同的尺寸相等的回退段，回退段的 **OPTIMAL** 参数值应比 **AVEACTIVE** 值稍微高一些。下面提供了怎样解释这个视图中所给统计值的附加信息。

| SHRINKS | AVESHRINK | Analysis and Recommendation   |
|---------|-----------|---|
| 低       | 低         | 如果 AVEACTIVE 值与 OPTSIZE 值接近，那么 OPTIMAL 参数设置是正确的。反之，OPTIMAL 值就设置太大了（许多收缩不能被完成） |
| 低       | 高         | 极好：OPTIMAL 值设置得很好   |
| 高       | 低         | OPTIMAL 值设置得太小：要完成太多的收缩   |
| 高       | 高         | 周期性的长事务处理大概会引起这样的统计值。将 OPTIMAL 参数值设置得较高一些，除非 SHRINK 值低                        |

11.6.3 显示全部回退段

下面的查询语句返回每个回退段的名称、所属表空间和它们的尺寸：

```
SELECT segment_name, tablespace_name, bytes, blocks, extents
FROM sys.dba_segments
WHERE segment_type = 'ROLLBACK';
```

| SEGMENT_NAME | TABLESPACE_NAME | BYTES   | BLOCKS | EXTENTS |
|--------------|-----------------|---------|--------|---------|
| SYSTEM       | SYSTEM          | 409600  | 200    | 8       |
| RB_TEMP      | SYSTEM          | 1126400 | 550    | 11      |
| RB1          | RBS             | 614400  | 300    | 3       |
| RB2          | RBS             | 614400  | 300    | 3       |
| RB3          | RBS             | 614400  | 300    | 3       |
| RB4          | RBS             | 614400  | 300    | 3       |
| RB5          | RBS             | 614400  | 300    | 3       |
| RB6          | RBS             | 614400  | 300    | 3       |
| RB7          | RBS             | 614400  | 300    | 3       |
| RB8          | RBS             | 614400  | 300    | 3       |

10 rows selected.

11.6.4 显示是否有回退段已经脱机

当你使一个回退段脱机时，实际上它并不立即脱机，直到所有使用它的激活的事务处理都已经完成为止。在你试图使它脱机到它真正脱机期间，它在 **DBA\_ROLLBACK\_SEG** 的状态是 **PENDING OFFLINE**，且不能用于新的事务处理。为了测定一个事例是否有回退段在这种状态，使用下面的查询：

```
SELECT name, xacts "ACTIVE TRANSACTIONS"
FROM v$rollname, v$rollstat
WHERE status = 'PENDING OFFLINE'
AND v$rollname.usn = v$rollstat.usn
;
NAME ACTIVE TRANSACTIONS
```

如果你的事例是并联服务器配置的环境，这个查询仅显示当前事例回退段的信息，而不显示其它事例回退段信息。

## 第四部分 模式对象

---

第四部分讲述 Oracle 数据库中模式对象的创建和维护。包括下列各章：

- 第 12 章，“模式对象管理指南”
- 第 13 章，“管理数据表”
- 第 14 章，“管理索引”
- 第 15 章，“管理分区数据表和索引”
- 第 16 章，“管理簇”
- 第 17 章，“管理散列簇”
- 第 18 章，“管理视图、序列和别名单元”
- 第 19 章，“模式对象的常规管理”
- 第 20 章，“解决数据块讹误的问题”

# 12

## 模式对象管理指南

本章讲述管理模式对象的指南，包括下列主题：

- 管理数据块的内部空间
- 事务项设置（INITRANS 和 MAXTRANS）
- 设置存储参数
- 重分配空间
- 了解数据类型的空间使用

用户在试图管理指定的模式对象之前（如第 13——18 章所述），应熟知本章中的概念。

### 12.1 管理数据块的空间

本节讲述数据块空间管理的各个方面。论述 PCTFREE 和 PCTUSED 参数，它们的作用如下：

- 提高写入和检索数据的操作性能
- 减少未用数据块空间的数量
- 减少数据块之间行链接的数量

包括下列主题：

- PCTFREE 参数
- PCTUSED 参数
- 选择相关的 PCTUSED 和 PCTFREE 值

### 12.1.1 PCTFREE 参数

PCTFREE 参数用于设置存储块中备用空间的比率。存储块中数据行的更新需要保留一定的备用空间。例如：假定用户在 CREATE TABLE 语句内指定了下列参数：

PCTFREE 20

此例标明表数据段中各数据块的 20% 将保持自由和可用，它将用于存储块中已有数据行的更新，图 12-1 说明了 PCTFREE 参数的作用。

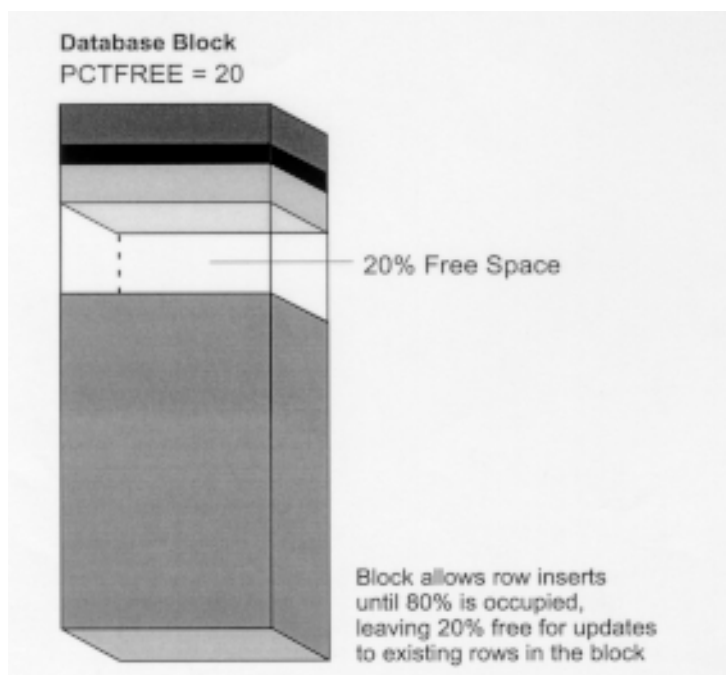


图 12-1 PCTFREE

要注意的是在存储块达到 PCTFREE 值之前，数据块的自由空间用于加入新的数据行以及数据块头部信息增长的内容。

#### 指定 PCTFREE 参数

PCTFREE 缺省值是 10%。用户可以使用任何在 0 和 99 之间的整数，只要 PCTFREE 和 PCTUSED 的总数不超过 100。

较小的 PCTFREE 值有下列影响：

- 仅保留较少的空间用于现有数据表中行（列）的扩展更新
- 使插入数据可以更充分地占满存储块
- 可以节省空间，因为表或索引总的可储存在较少的存储块里（每个存储块可以存放更多的数据行或条目）

较小的 PCTFREE 值适用于数据段很少改变的情况。

较大的 PCTFREE 值有下列影响：

- 保留了更多的空间用于现有数据表中行（列）的更新

- 可能需要更多的存储块用于存放相同数量的插入数据（每个存储块只能插入较少的行）

- 可以提高更新操作的性能，因为 Oracle 不需要频繁地链接行列片段

较大的 **PCTFREE** 值适合于数据段频繁更新的情况。在设置 **PCTFREE** 之前，用户应确实明白该数据表或者索引数据的性质。更新操作可能使数据行增大。新数值也许和被替换的值不一样大。如果有许多更新操作使数据值增大，**PCTFREE** 值应相应增加。如果更新操作不影响总的行列宽度，**PCTFREE** 可以是较低的值。用户的目的是在高密度压缩数据和优良的更新操作性能之间寻找一个令人满意的平衡点。

**非簇数据表的 PCTFREE 参数** 如果非簇数据表中的数据可能随着时间而增大，就需要为更新操作保留空间。否则，更新的行列可能被链接到多个存储块中。

**簇数据表的 PCTFREE 参数** 关于非簇数据表的讨论也适用于簇数据表。无论如何，如果达到了 **PCTFREE** 的设定值，新增行列----来自任何包含在相同簇集群主键中的数据表----就会存入链接于当前簇主键的新数据存储块中。

**索引的 PCTFREE 参数** 只有在最初创建索引时用户才可以指定 **PCTFREE** 参数。

### 12.1.2 PCTUSED 参数

按照 **PCTFREE** 的设定，数据块被认为是满的以后，Oracle 不会考虑在其中插入新的行列，直到存储块的使用率下降到 **PCTUSED** 参数值以下。在达到这个值之前，Oracle 只将数据块的自由空间用于数据块中已经包含的数据行（列）的更新。例如，假定用户在 **CREATE TABLE** 语句中指定下列参数：

**PCTUSED 40**

假若这样，在存储块已用空间的使用率降低到 39%或是更低的数值以前，存放表中数据段的存储块不会插入任何新的行列。（假定存储块已用空间已经达到 **PCTFREE** 的设定值）。

图 12-2 是图解说明。

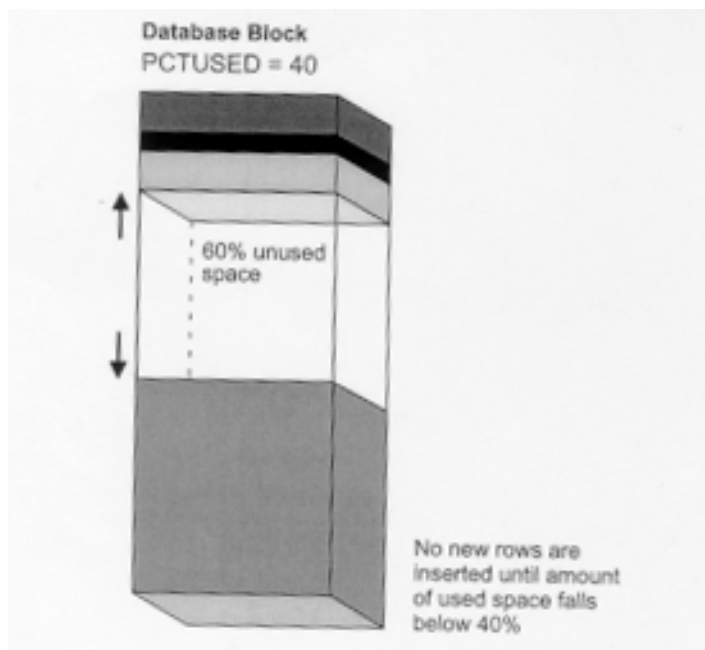


图 12-2 PCTUSED

#### 指定 PCTUSED 参数

PCTUSED 的缺省值是 40%。在数据存储块中自由空间达到 PCTFREE 的设定值以后，不会有新的行列插入数据块，直到该空间的使用率降到 PCTUSED 设定值之下。这里讲的百分率值是存储块总空间中减去通常开支后的可用空间。

用户可以为 PCTUSED 指定任何 0 到 99（含）之间的整数，只要 PCTUSED 和 PCTFREE 的总和不超过 100。

较小的 PCTUSED 有下列影响：

- 减少数据处理的耗费。当使用率低于给定值时，将存储块移到自由列表中的 UPDATE 和 DELETE 语句会产生这种耗费。
- 增大数据库闲置空间

较大的 PCTUSED 有下列影响：

- 提高空间效率
- 在 INSERT 和 UPDATE 操作期间增大数据处理的耗费

#### 12.1.3 选择相关的 PCTUSED 和 PCTFREE 值

如果用户决定不使用 PCTFREE 或者 PCTUSED 的缺省值，要记住下面的准则：

- PCTFREE 和 PCTUSED 总和必须等于或者小于 100。
- 如果总和等于 100，那么 Oracle 将试图保持不超过 PCTFREE 的自由空间，而数据处理耗费是最高的。
- PCTUSED 或者 PCTFREE 的计算不包括数据块的通常开支。
- 在一定的操作耗费条件下，PCTFREE 和 PCTUSED 的总和与 100 的差值越小（例如 PCTUSED 75，PCTFREE 20），空间使用越有效。



选择 PCTFREE 和 PCTUSED 值的例子

下列例子示范如何为数据表指定 PCTFREE 和 PCTUSED 的具体数值及其原因。

例 1      概要:    普通作业包含使行列容量增大的 UPDATE 语句。

          设置:    PCTFREE = 20

                  PCTUSED = 40

例 2      概要:    大多数作业包含 INSERT 和 DELETE 语句, 而 UPDATE 语句不使受影响的行列容量增大。

          设置:    PCTFREE = 5

                  PCTUSED = 60

          解释:    PCTFREE 设置为 5, 是因为大多数 UPDATE 语句不增大行列容量。PCTUSED 设置为 60, 以便被 DELETE 语句释放的空间可以很快得到使用, 而且数据处理量还极小。

例 3      概要:    数据表很大; 因此, 存储量是首要关心的。大多数作业包含只读处理。

          设置:    PCTFREE = 5

                  PCTUSED = 40

          解释:    PCTFREE 设置为 5, 是因为使用一个大数据表而用户要充分利用各个存储块。

## 12.2 事务项设置 (INITRANS 和 MAXTRANS 参数)

作为分配给表, 簇或者索引的数据存储块, 其 INITRANS 和 MAXTRANS 事务项设置应基于下列指标为各个对象逐一设置:

■ 用户为事务项保留的空间大小, 它与为数据库数据保留的空间相匹配

■ 在任一给定时间的条件下, 可能涉及相同数据块的并行事务的数量

例如: 如果一个数据表非常大并且只有少数的用户同时访问数据表, 则要求访问相同数据块的复杂并行事务的几率就比较低。因此, INITRANS 可以设置低些, 特别是在数据库中空间非常宝贵的情况。

另一方面, 假定一个数据表通常有许多用户同时访问, 用户可以考虑使用高 INITRANS 值 (以消除分配事务项空间的通常开支, 它在对象使用中是必需的) 预先分配事务项空间, 并允许设置较高的 MAXTRANS 值以便用户存取必要的数据块时不需要等待。

### 12.2.1 INITRANS

指定 DML 事务项的数量, 以在最初为其在数据块头部保留空间。保留空间位于关联数据或者索引数据段中所有数据块的头部。数据表的缺省值是 1, 簇和索引的缺省值是 2。

## 12.2.2 MAXTRANS

在多重事务同时访问相同数据块的行列时，在存储块中将为每个 DML 事务项分配空间。一旦由 INITRANS 保留的空间被耗尽，如果可以，将在存储块自由空间之外为额外事务项分配空间。一经分配，这个空间将永久性地成为存储块头部内容的一部分。MAXTRANS 参数限制了可以同时使用数据块中数据的事务项的数量。因此，用户可以使用 MAXTRANS 限定数据块中分配给事务项的自由空间的总数。

缺省值是操作系统特定功能确定的字组大小，不超过 255。

## 12.3 设置储存参数

本节讲述用户为各种的数据结构设置的储存参数。这些储存参数用于下列结构类型和模式对象：

- 表空间（用作所有数据段的储存参数预设值）
- 表、分区、簇、快照和快照日志（数据段）
- 索引（索引数据段）
- 回退数据段

将讨论下列主题：

- 确定储存参数
- 设置表空间数据段的缺省储存参数
- 设置数据段储存参数
- 设置索引数据段储存参数
- 设置 LOB，可变数组（Varray）和嵌套表的储存参数
- 改变储存参数值
- 了解储存参数中的优先权

### 12.3.1 确定储存参数

每个数据库都有储存参数的缺省值。但是，用户可以为表空间指定新的预设值，它仅在本表空间中取代系统预设值成为表空间中创建对象的预设值。缺省储存值是通过 CREATE 或是 ALTER TABLESPACE 指令的 DEFAULT STORAGE 子句指定的。

而且，用户可以为每一个模式对象指定储存设置，它将取代任何缺省储存设置。要做到这一点，用户需要对单一对象使用 CREATE 或是 ALTER 语句的 STORAGE 子句。

用户在创建模式对象时指定储存参数，并可以在以后改变它。不是所有的储存参数都可以用于每个数据库对象类型，并且也不是所有的储存参数既可以在 CREATE 语句中指定又可以在 ALTER 语句中指定。要设置或修改存储参数值，用户必须具有使用 CREATE 或 ALTER 语句的必要的授权。

下列章节标识了用户可以指定的储存参数。

一些存储参数的设置值是具体的操作系统相关的。参考用户具体操作系统提供的文件以获取那些参数值的信息。

## INITIAL

当创建数据段时分配的第一个区域的容量，以字节为单位。这个参数不能在 ALTER 语句中指定。

|      |                              |
|------|------------------------------|
| 缺省值  | 5 data blocks                |
| 最小值: | 2 个数据块（无位图数据段），3 个数据块（位图数据段） |
| 最大值: | 操作系统指定                       |

## NEXT

数据段中增加的下一个区域的容量，以字节为单位。第二区域的大小等于 NEXT 的初始设置。从这以后，NEXT 的值设置为前一容量乘以  $(1 + \text{PCTINCREASE}/100)$ 。

|      |        |
|------|--------|
| 缺省值: | 5 个数据块 |
| 最小值: | 1 个数据块 |
| 最大值: | 操作系统指定 |

## PCTINCREASE

相对于最后增加的区域，数据段每个新增区域大小的增大比率。如果 PCTINCREASE 为 0，那么所有增加的区域容量相同。如果 PCTINCREASE 大于零，那么每次计算 NEXT 值，都按 PCTINCREASE 增大相应的值。PCTINCREASE 不能为负。

新的 NEXT 值等于  $(1 + \text{PCTINCREASE}/100)$  乘以最后增加的区域容量（旧的 NEXT 值），并四舍五入到字组大小的倍数。

|      |        |
|------|--------|
| 缺省值: | 50 (%) |
| 最小值: | 0 (%)  |
| 最大值: | 操作系统指定 |

## MINEXTENTS

当创建数据段时分配区域的总数。这个参数是为创建时大的空间分配而设，即使连续的存储空间不可用。

|      |                           |
|------|---------------------------|
| 缺省值: | 1 ( 区域); 2 ( 区域), 用于回退数据段 |
| 最小值: | 1 ( 区域); 2 ( 区域), 用于回退数据段 |
| 最大值: | 操作系统指定                    |

## MAXEXTENTS

可用于数据段分配的区域总数，包括第一个区域。

|      |                           |
|------|---------------------------|
| 缺省值: | 取决于数据块容量和操作系统             |
| 最小值: | 1 ( 区域); 2 ( 区域), 用于回退数据段 |
| 最大值: | 无限制                       |

## FREELIST GROUPS

用户创建的数据库对象的自由列表组的数量。Oracle 使用 Oracle 并行服务事例的事例

编号将每个事例映射到自由表组。

|      |                       |
|------|-----------------------|
| 缺省值: | 1                     |
| 最小值: | 1                     |
| 最大值: | 取决于 Oracle 并行服务事例的数量。 |

**FREELISTS**

模式对象中每一自由表组的自由表数量。对于表空间无效。

|      |          |
|------|----------|
| 缺省值: | 1        |
| 最小值: | 1        |
| 最大值: | 取决于数据块容量 |

**OPTIMAL**

仅与回退数据段有关。参见第 11 章“管理回退数据段”中关于此参数使用的信息。

**BUFFER POOL**

给模式对象定义缺省缓冲池（高速缓存）。对于表空间或者回退数据段无效。

**12.3.2 设置表空间数据段的缺省储存参数**

用户可以为数据库的每个表空间设置缺省存储参数。对于任何存储参数，如果用户在创建表空间或者改变表空间的数据段时没有明确地设置，那么它们将自动地设置为数据段驻留的表空间中对应的缺省存储参数。

在表空间一级指定 **MINEXTENTS** 时，任何在该表空间中分配的区域都将四舍五入到最小区域值的某一倍数。基本上，区域数是存储块数量的倍数。

**12.3.3 设置数据段储存参数**

比较起来，用户使用带有 **STORAGE** 子句的 **CREATE CLUSTER** 或者 **ALTER CLUSTER** 语句设置簇中数据段存储参数，而不是单独使用 **CREATE** 或者 **ALTER** 语句将表和快照放入簇中。创建或者改变簇表或快照时指定的存储参数被忽略。对于簇设置的存储参数将覆盖表的存储参数。

对于分区表，用户可以在表一级设置缺省存储参数。创建表的新分区时，继承表一级的缺省存储参数（除非用户为个别分区单独指定这些参数）。如果在表一级没有指定存储参数，那么将继承表空间的。

**12.3.4 设置索引数据段储存参数**

可以使用 **CREATE INDEX** 或者 **ALTER INDEX** 语句的 **STORAGE** 子句设置表索引中索引数据段的存储参数。对于执行主关键字或者单一关键字约束的索引，可以在 **CREATE TABLE** 或者 **ALTER TABLE** 语句的 **ENABLE** 子句中设置其索引数据段的存储参数，也可以在 **ALTER INDEX** 语句的 **STORAGE** 子句中设置。

### 12.3.5 设置 LOB、Varray 和嵌套表的储存参数

表或者快照可能包含 LOB、varray 或者嵌套表数据列类型。这些实体可以储存在自己的数据段中。LOB 和 varray 储存于 LOB 数据段，而嵌套表储存于存储表中。用户可以为这些数据段指定 STORAGE 子句，覆盖在表一级指定的存储参数。

### 12.3.6 改变储存参数值

用户可以改变表空间的缺省存储参数，并且可以为个别数据段指定存储参数。可以重置表空间的缺省存储参数。无论如何，参数改变只影响表空间中创建的新对象，或是数据段中分配的新区域。

不能改变现有表、簇、索引或回退数据段的 INITIAL 和 MINEXTENTS 存储参数。如果只改变了数据段的 NEXT 值，下一个增加区域的容量是新 NEXT 的值，其后的区域容量仍按 PCTINCREASE 增大。

如果数据段的 NEXT 和 PCTINCREASE 都改变了，下一个新区域的容量按 NEXT 的新数值给定，并且从这一点起，NEXT 的值按新的 PCTINCREASE 值进行计算。

### 12.3.7 了解储存参数中的优先权

存储参数在某一时间起作用是由下列 SQL 语句决定的，依照优先权排列如下：

1. ALTER TABLE/CLUSTER/SNAPSHOT/SNAPSHOT LOG/INDEX/ROLLBACK 语句
2. CREATE TABLE/CLUSTER/SNAPSHOT/SNAPSHOT LOG/CREATE INDEX/ ROLLBACK SEGMENT 语句
3. ALTER TABLESPACE 语句
4. CREATE TABLESPACE 语句
5. Oracle 缺省值

任何存储参数在对象级的指定都将覆盖对应的表空间级的设置。如果在对象级没有明确地设置存储参数，存储参数缺省使用表空间级的设置。如果在表空间级仍没有设置存储参数，那么将使用 Oracle 系统缺省值。如果存储参数改变，新的选择只应用于将要分配的区域。

**注意** 临时数据段的存储参数总是使用相关表空间的缺省存储参数设置。

#### 存储参数举例

假定执行了下列语句：

```
CREATE TABLE test_storage
( ... )
STORAGE (INITIAL 100K NEXT 100K
          MINEXTENTS 2 MAXEXTENTS 5
          PCTINCREASE 50) ;
```

并且初始化参数 DB\_BLOCK\_SIZE 假定设置为 2K。下表显示了 TEST\_STORAGE 表中区域是如何分配的。同时还在 USER\_SEGMENTS 或者 DBA\_SEGMENTS 数据字典视图的 NEXT 栏显示了增加区域的值：

表 12-1 区域分配

| 区域# | 区域容量                 | NEXT 值               |
|-----|----------------------|----------------------|
| 1   | 50 个存储块或者 102400 字节  | 50 个存储块或者 102400 字节  |
| 2   | 50 个存储块或者 102400 字节  | 75 个存储块或者 153600 字节  |
| 3   | 75 个存储块或者 153600 字节  | 113 个存储块或者 231424 字节 |
| 4   | 115 个存储块或者 235520 字节 | 170 个存储块或者 348160 字节 |
| 5   | 170 个存储块或者 348160 字节 | MAXEXTENTS=5, 没有下一个值 |

如果用户利用 ALTER 语句（比如 ALTER TABLE）改变 NEXT 或 PCTINCREASE 存储参数，新的值将替换储存在数据字典中的当前值。例如，在为 TEST\_STORAGE 表分配第三个区域之前，用下列语句改变表的存储参数：

```
ALTER TABLE test storage STORAGE (NEXT 500K) ;
```

结果是第三个区域分配了 500K，第四个区域是  $(500K * 1.5) = 750K$ ，等等。

## 12.4 释放空间

为数据段分配空间是很平常的，只是要在以后弄清楚分配的空间是否没有使用。例如，用户可以设置较高的 PCTINCREASE 值，创建一个大的区域，而该区域只部分地使用。或者用户可以明确地用 ALTER TABLE...ALLOCATE EXTENT 语句过分配空间。如果发现未用的或者过量分配的空间，用户可以将其释放，以便其它的数据段可以使用它们。

本节叙述未用空间重分配方面的内容。

### 12.4.1 查看高水位标记

在重新分配前，用户可以使用 DBMS\_SPACE 程序包，其中包括一个程序 (UNUSED\_SPACE)，可以返回关于高水位标记的位置和数据段中未用空间的数量。

在数据段内高水位标记指示已用空间的数量，或者是已经被格式化以接受数据的空间。用户无法释放高水位标记以下的空间（即使用户想要重新分配的空间中沒有数据）。但是，如果数据段被完全清空，用户可以使用 TRUNCATE...DROP STORAGE 语句释放空间。

### 12.4.2 发布空间重新分配指令

下列指令重新分配数据段（表、索引或者簇）中未用的空间。KEEP 子句是可选择使用的部分。

```
ALTER TABLE table DEALLOCATE UNUSED KEEP integer;
ALTER INDEX index DEALLOCATE UNUSED KEEP integer;
ALTER CLUSTER cluster DEALLOCATE UNUSED KEEP integer;
```

如果用户明确地标识 KEEP 子句中未用空间的数量，那么当其余未用空间重新分配时，指定数量的未用空间将予以保留。如果区域的余留数量小于 MINEXTENTS，MINEXTENTS 值将改变以表现新的数量。如果初始区域变小，INITIAL 值将改变以表现初始区域的新容量。

如果用户没有指定 KEEP 子句，只要可以保持初始区域的容量和 MINEXTENTS 的值，所有未用空间（任何高于高位标记的部分）均重新分配。因此，即使高位标记出现在

MINEXTENTS 边界内，MINEXTENTS 也保持不变，并且初始区域容量不会减少。

用户可以查看 DBA\_FREE\_SPACE 视图，验证重新分配的空间被释放。

#### 重分配空间举例

本节提供了一些空间重新分配的例子。

##### 例 1:

一个表包含三个区域。第一个区域是 10K，第二个是 20K，第三个是 30K。高位标记在第二个区域的中部，并且有 40K 的未用空间。图 12-3 图解说明下列指令的作用：

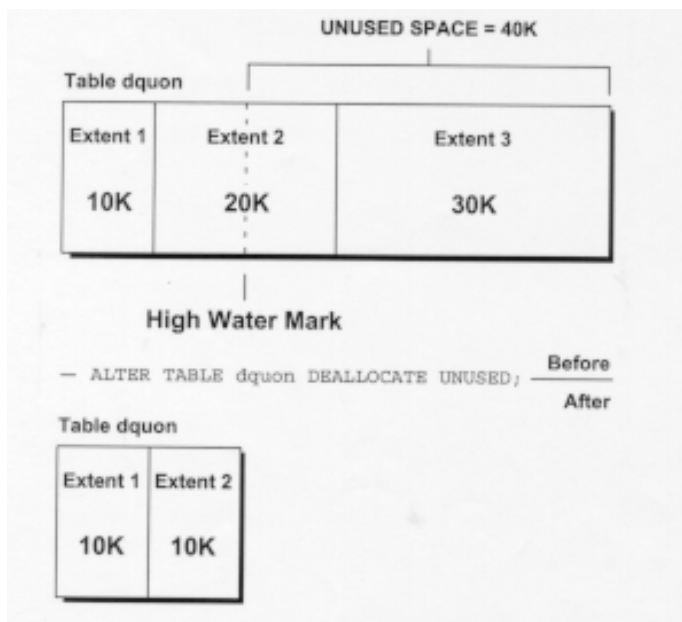


图 12-3 重分配所有未用空间

ALTER TABLE dquon DEALLOCATE UNUSED

重新分配了所有的未用空间，DQUON 表剩下两个区域。第三个区域消失，并且第二个区域容量是 10K。

但是，如果用户发出下列指令指定了 KEEP 关键字，那么高于高位标记的 10K 区域将被保留，DQUON 中其余的未用空间将重新分配。

ALTER TABLE dquon DEALLOCATE UNUSED KEEP 10K;

实际上，第三个区域进行了重新分配，而第二个区域保持原样。

图 12-4 说明了这个情况。

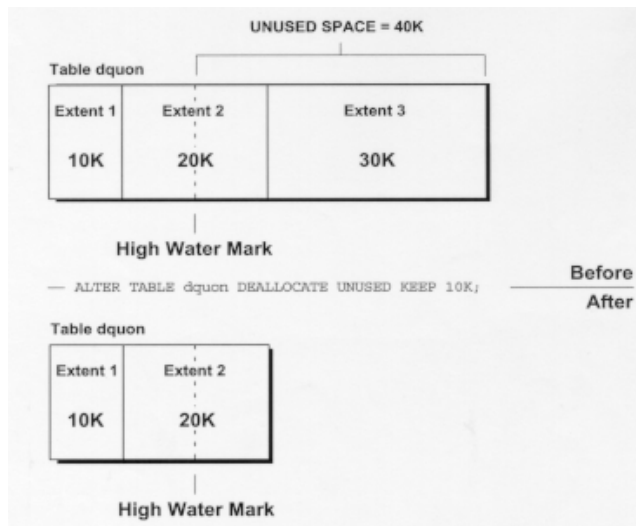


图 12-4 重分配未用空间，KEEP 10K

更进一步，如果用户用下列指令重新分配 DQUON 中所有的未用空间，并保留 20K 未用空间，则第三个区域被削减到 10K，而第二个区域保持原来的容量大小。

```
ALTER TABLE dquon DEALLOCATE UNUSED KEEP 20K;
```

#### 例 2:

设想图 12-3 说明的情况。区域 3 被完全重新分配，第二个区域保留了 10K。更进一步，下一个将分配的区域，其容量缺省值是最后一个完全重新分配区域的大小，在本例中是 30K。如果这不是用户想要的，用户可以明确地设置下一个区域的容量：使用 ALTER TABLE 指令，在 STORAGE 子句中为 NEXT 指定一个新数值。

下列指令设置 DQUON 表下一个区域的容量为 20K。

```
ALTER TABLE dquon STORAGE (NEXT 20K)
```

#### 例 3:

为了保持区域的 MINEXTENTS 值，DEALLOCATE 可以保持最初分配给数据段的区域。这一性能受到 KEEP 参数的影响，这一点前面已经说明。

如果 DQUON 表的 MINEXTENTS 值为 2，图 12-3 和图 12-4 中说明的指令产生如图显示的结果，保持了 MINEXTENTS 的初始值。

可是，如果 MINEXTENTS 值是 3，图 12-4 中说明的指令产生如同显示的相同的结果（第三个区域被删除），但 MINEXTENTS 的值改变为 2。图 12-3 中说明的指令将不能生产相同的结果。在本例中，该指令不起作用。

## 12.5 了解数据类型的空间使用

在创建表和其它数据结构的时候，用户需要知道它们要求多少空间。每种数据类型有不同的空间要求。



# 13 管理数据表

本章叙述了管理数据表的各个方面，包括下列内容：

- 管理数据表的方针
- 创建表
- 修改表
- 删除表
- 组织索引表

## 13.1 管理数据表的方针

本节讲述了管理数据表所应遵循的方针，包括下列内容：

- 创建表之前要先进行设计
- 规定如何使用数据块空间
- 规定事务条目的参数
- 规定每个表的位置
- 并行地创建表
- 考虑创建 UNRECOVERABLE 表
- 估算表尺寸和设置存储参数
- 规划大型表
- 表的约束

运用这些方针使表的管理尽可能地容易。

### 13.1.1 建表之前要先进行设计

通常，应用程序开发者负责规划应用程序的要素，包括表。数据库管理员根据程序开发者提供的如何使应用程序工作以及预期的数据类型的信息，设置存储参数，并为表定义簇。

与你的应用程序开发者一起工作，仔细地设计每个表以满足下列要求：

- 表是规范化的。
- 每个列都具有适当的数据类型。
- 允许空的列最后定义，以节省存储空间。
- 无论何时适当地簇化表以节省存储空间，并优化 SQL 语句的性能。

### 13.1.2 规定如何使用数据块空间

在创建每个表时通过指定 PCTFREE 和 PCTUSED 参数，你能够影响空间的使用效率，并为更新表段数据块的当前数据预留磁盘空间。

### 13.1.3 规定事务条目的参数

在创建每个表时通过指定 `INITRANS` 和 `MAXTRANS` 参数，你能够影响初始化的磁盘空间数量，甚至能影响表段数据块的事务条目的分配。

### 13.1.4 规定每个表的位置

如果你有适当的许可权和表空间配额，你可以在当前任何联机的表空间中创建新表。在 `CREATE TABLE` 语句指定 `TABLESPACE` 子句以确定存储新表的表空间，这种做法是明智的。如果你不在 `CREATE TABLE` 语句中指定一个表空间，将在你默认的表空间中创建表。

当指定表空间包含一个新表时，确信你了解你所做的选择的含义。在创建每个表时适当地指定一个表空间，你能：

- 增加数据库系统的性能
- 减少管理数据库所需要的时间

下列情形表明了为模式对象指定了不正确的存储位置能够对数据库产生怎样的影响：

- 如果用户的对象创建在 `SYSTEM` 表空间，Oracle 的性能将降低，因为数据词典对象和用户对象不得不争用相同的数据文件。
- 如果与一个应用相关的表被武断地存放在多个表空间中，那么，将增加为应用数据进行管理操作（备份和恢复）所需要的时间。

第 22 章提供了关于为用户指定默认表空间和表空间配额的信息。

### 13.1.5 并行地创建表

通过 `CREATE TABLE` 语句的子查询功能，可以并行地创建表。因为创建表是多个过程一起工作，因此能够改进系统创建表的性能。

### 13.1.6 考虑创建 `UNRECOVERABLE` 表

当创建一个不可恢复表时，该表就不能从档案日志中被恢复（因为不可恢复表不产生所需要的重做日志的记录）。因此，如果不能承受表的丢失，在表创建后应做一个备份。在某些情况下，例如为临时使用而创建的表，这种预防措施也许是不必要的。

当你用 `CREATE TABLE...AS SELECT` 语句及子查询项创建表时，通过指定 `UNRECOVERABLE` 能够创建一个不可恢复表。然而，后来加入的行记录是可恢复的。事实上，该语句完成后，所有后续的语句都是完全可恢复的。

创建一个不可恢复的表有以下优点：

- 重做日志文件的空间被节省了
- 创建表的时间减少了
- 改善了并行创建大型表的性能

一般来说，对于大型的不可恢复表性能的相对改善比小表更明显一些。创建小型的不可恢复表对于创建时间没有什么影响。而对于大型表，性能的改进是很有意义的，特别是也要并行地创建表时。

### 13.1.7 估算表尺寸和设置存储参数

由于下列原因，创建表时估计它们的尺寸是很有用的：

- 能够在估算索引、回退段和重做日志文件时一起估算表的尺寸，以决定你管理的数据库所需要的磁盘空间的容量。从这些估算中，可以做出正确的购买硬件以及其它事情的决定。
- 能够通过估算一个单独的表的尺寸来更好的管理该表所使用的磁盘空间。在创建表时，你可以设置适当的存储参数改善应用使用表的 I / O 性能。例如：假定在创建一个表前估算了它的最大值，如果在创建表时设置了存储参数，将只分配给表的数据段很少的区域，并且所有的表的数据将被存放在磁盘空间的相毗邻的区域。这样就减少了包含这个表的磁盘的 I / O 操作所需要的时间。

无论在创建表之前是否估算表的尺寸，在创建每个非簇表时都要明确地设置存储参数。（簇化表自动地使用簇的存储参数。）当创建或随后修改表时，任何没有明确设置的存储参数都自动使用表所在的表空间默认的存储参数。

如果要为表的数据段的区域明确地设置存储参数，应尽量使表的数据存放在少量的大区域中，而不是大量的小区域中。

### 13.1.8 为大表做规划

表和区域的物理尺寸没有限制。可以为 MAXEXTENTS 指定关键字 UNLIMITED，这样就简化了对大的对象的规划，减少了空间的浪费和碎片，改善了空间的再使用性能。然而，要注意当 Oracle 允许区域的数量没有限制时，一个表的区域数能够增到很大，在你对表进行任何需要的操作时，也许会发现它对系统性能的影响。

**注意** 不能修改数据字典表，使 MAXEXTENTS 大于所允许的块的最大值。

如果你的数据库中有这样的表，考虑下列建议：

- 将表和它的索引分开

把索引放在与其它对象分离的表空间中，尽可能放在分离的磁盘上。如果你在一个很大的表上需要废弃并重建一个索引（例如禁止或允许一项约束，或重建表时）。把索引隔离在单独的表空间常常比把它放在包含其它对象的表空间中更容易发现连续的空间。

- 分配足够的临时空间

如果在一个很大的表中存储数据的应用要完成大的分类，必须保证大的临时段有足够的空间（临时段始终使用默认的 STORAGE 参数设置它们的表空间）。

### 13.1.9 表的约束

在创建表之前，要了解下列约束：

- 表中包含新的对象类型的表不能输入到 Oracle8 的预备数据库。
- 当数据库中仍然存在原始数据时，不能把表的类型和区域移动到不同的模式。
- 预存在的表中有不同模式的相同名称，不能把它合并为一个输出表。
- Oracle 对一个表（或一个对象类型的属性）所能拥有的列的总数有一个限制。

此外，当创建一个包含用户定义的数据类型的表时，Oracle 映射用户定义类型的列到相关的存储用户定义类型的数据的列。这些“隐藏”的相关列在 DESCRIBE 表语句中是不可见的，并且也不能通过 SELECT\*语句返回信息。因此，当创建一个对象表、或一个包含 REF 列或可变数组、嵌套表、或对象类型的表时，Oracle 自动为表创建的列的总数也许要多于你

指定的列数，因为 Oracle 要为存储用户定义类型的数据创建隐藏列。  
下列公式决定了为含有用户定义类型数据的表创建的列的总数。

#### 在一个对象表的列数

$$\begin{aligned} \text{num\_columns}(\text{object\_table}) = & \\ & \text{num\_columns}(\text{object\_identifier}) \\ & + \text{num\_columns}(\text{row\_type}) \\ & + \text{number of top-level object columns in the object type of table} \\ & + \text{num\_columns}(\text{object\_type}) \end{aligned}$$

#### 在一个关系表的列数

$$\begin{aligned} \text{num\_columns}(\text{relational\_table}) = & \\ & \text{number of scalar columns in the table} \\ & + \text{number of object columns in the table} \\ & + \text{SUM} [\text{num\_columns}(\text{object\_type}(i))] \quad i = 1 \rightarrow n \\ & + \text{SUM} [\text{num\_columns}(\text{nested\_table}(j))] \quad j = 1 \rightarrow m \\ & + \text{SUM} [\text{num\_columns}(\text{varray}(k))] \quad k = 1 \rightarrow p \\ & + \text{SUM} [\text{num\_columns}(\text{REF}(l))] \quad l = 1 \rightarrow q \end{aligned}$$

#### 其中:

$\text{object\_type}(i)$  是第  $i$  个对象类型列并且  $n$  是这些对象类型列的总数，  
 $\text{nested\_table}(j)$  是第  $j$  个嵌套表列并且  $m$  是这些嵌套表列的总数，  
 $\text{varray}(k)$  是第  $k$  个可变数组列并且  $p$  是这些可变数组列的总数，  
 $\text{REF}(l)$  是第  $l$  个 REF 列并且  $q$  是这些 REF 列的总数。

#### 定义

$$\begin{aligned} \text{num\_columns}(\text{object\_identifier}) &= 1 \\ \text{num\_columns}(\text{row\_type}) &= 1 \\ \text{num\_columns}(\text{REF}) &= 1, \text{ 如果 REF 无作用域} \\ &= 1, \text{ 如果 REF 有作用域并且由系统创建对象标示符并且 REF 没有参考约束} \\ &= 2, \text{ 如果 REF 有作用域并且由系统创建对象标示符并且 REF 有一个参考约束} \\ &= 1 + \text{在主键的列数, 如果对象标示符是基于主键的} \\ \text{num\_columns}(\text{nested\_table}) &= 2 \\ \text{num\_columns}(\text{varray}) &= 1 \\ \text{num\_columns}(\text{object\_type}) &= \text{对象类型的分级属性的数量} \\ &+ \text{SUM} [\text{num\_columns}(\text{object\_typ}(i))] \quad i = 1 \rightarrow n \\ &+ \text{SUM} [\text{num\_columns}(\text{nested\_table}(j))] \quad j = 1 \rightarrow m \\ &+ \text{SUM} [\text{num\_columns}(\text{varray}(k))] \quad k = 1 \rightarrow p \\ &+ \text{SUM} [\text{num\_columns}(\text{REF}(l))] \quad l = 1 \rightarrow q \end{aligned}$$

#### 其中:

$\text{object\_type}(i)$  是一个内置对象类型属性并且  $n$  是这些对象类型属性的总数，  
 $\text{nested\_table}(j)$  是一个内置嵌套表属性并且  $m$  是这些嵌套表属性的总数，  
 $\text{varray}(k)$  是一个内置可变数组属性并且  $p$  是这些可变数组属性的总数，  
 $\text{REF}(l)$  是一个内置 REF 属性并且  $q$  是这些 REF 属性的总数。

下面是一些关于对象表、具有 REF 列、可变数组、嵌套表或对象类型的关系表的计算列数的例子。

#### 例子 1

```
CREATE TYPE physical_address_type AS OBJECT
    (no CHAR(4), street CHAR(31), city CHAR(5), state CHAR(3));
CREATE TYPE phone_type AS VARRAY(5) OF CHAR(15);
CREATE TYPE electronic_address_type AS OBJECT
    (phones phone_type, fax CHAR(12), email CHAR(31));
CREATE TYPE contact_info_type AS OBJECT
    (physical_address physical_address_type,
     electronic_address electronic_address_type);
CREATE TYPE employee_type AS OBJECT
    (eno NUMBER, ename CHAR(60),
     contact_info contact_info_type);
```

```
CREATE TABLE employee_object_table OF employee_type;
```

为了计算雇员对象表的列数，我们需要首先计算雇员类型所需要的列数：

```
num_columns(physical_address_type) =
    number of scalar attributes = 4
num_columns(phone_type) =
    num_columns(varray) = 1
num_columns(electronic_address_type) =
    number of scalar attributes
    + num_columns(phone_type)
    = 2 + 1 = 3
num_columns(contact_info_type) =
    num_columns(physical_address_type)
    + num_columns(electronic_address_type)
    = 3 + 4 = 7
num_columns(employee_type) =
    number of scalar attributes
    + num_columns(contact_info_type)
    = 2 + 7 = 9
```

现在利用对象表的公式：

```
num_columns(employee_object_table) =
    num_columns(object_identifier)
    + num_columns(row_type)
    + number of top level object columns in employee_type
    + num_columns(employee_type)
    = 1 + 1 + 1 + 9 = 12
```

#### 例子 2

```
CREATE TABLE employee_relational_table (einfo employee_type);
```

```

num_columns(employee_relational_table) =
    number of object columns in table
    + num_columns(employee_type)
    = 1 + 9 = 10

```

### 例子 3

```

CREATE TYPE project_type AS OBJECT
(pno NUMBER, pname CHAR(30), budget NUMBER);

```

```

CREATE TYPE project_set_type AS TABLE OF project_type;

```

```

CREATE TABLE department
(dno NUMBER, dname CHAR(30),
mgr REF employee_type REFERENCES employee_object_table,
project_set project_set_type)
NESTED TABLE project_set STORE AS project_set_nt;

```

```

num_columns(department) =
    number of scalar columns
    + num_columns(mgr)
    + num_columns(project_set)
    = 2 + 2 + 2 = 6

```

## 13.2 创建表

要以自己的模式创建新表，必须有 **CREATE TABLE** 的系统许可权。要以另一个用户的模式创建表，必须有 **CREATE ANY TABLE** 的系统许可权。此外，表的拥有者必须有该表所在表空间的配额，或有 **UNLIMITED TABLESPACE** 的系统许可权。

用 **SQL** 语句 **CREATE TABLE** 创建表。当用户 **SCOTT** 发布下列语句时，他就以他的模式创建了一个名为 **EMP** 的非簇表，并把它存储在 **USER** 表空间：

```

CREATE TABLE    emp(
    Empno    NUMBER(5) PRIIMARY KEY,
    Ename    VARCHAR2(15) NOT NULL,
    Job      VARCHAR2(10),
    Mgr      NUMBER(5),
    Hirdeate  DATE DEFAULT (sysdate),
    Sal      NUMBER (7,2),
    Comm.    NUMBER(7,2),
    Deptno   NUMBER(3) NOT NULL
                CONSTRAINT dept_fkey REFERENCES dept)
PCTFREE 10
PCTUSED 40
TABLESPACE users
STORAGE ( INITIAL 50K

```

```
NEXT 50K
MAXEXTENTS 10
PCTINCREASE 25);
```

在这个例子中，表的几个列都定义了完整的约束。完整的约束将在第 19 章中讨论。表的几个段的属性也被明确地指定。这些已在第 12 章中说明。

创建一个临时表也是可能的。临时表的定义对所有会话都是可见的，但临时表中的数据只有当会话向表中插入数据时才是可见的。使用 **CREATE GLOBAL TEMPORARY TABLE** 语句创建一个临时表。关键字 **ON COMMIT** 表明表中的数据是 **transaction-specific**(默认)还是 **session-specific**;

- **ON COMMIT DELETE ROWS** 说明临时表是事务指定，每次提交后 Oracle 将截短表（删除全部行）。
- **ON COMMIT PRESERVE ROWS** 说明临时表是会话指定，当中断会话时 Oracle 将截短表。

这个例子创建的临时表是事务指定：

```
CREATE GLOBAL TEMPORARY TABLE work_area
    (startdate DATE,
     enddate DATE,
     class CHAR(20))
ON COMMIT DELETE ROWS;
```

临时表不能创建索引。因为它们也是临时的，索引中的数据与表中的数据有相同的会话和事务范围。

参见 对于 **CREATE TABLE** 和其它本章讨论的语句，其正确的语法、授权或限制方面的信息，参见 “Oracle8i SQL Reference” 一书。

## 13.3 改变表

要改变的表必须包含在用户模式中，或是用户具有针对表的 **ALTER** 对象的权限，或是具有 **ALTER ANY TABLE** 的系统授权。

一个表可能因为下列原因而需要改变：

- 为了加入或删除数据列，或是要修改现有数据列的定义（数据类型、长度、缺省值以及 **NOT NULL** 完整性约束）
- 修改数据块空间使用参数（**PCTFREE**，**PCTUSED**）
- 修改事务条目设置（**INTRANS**，**MAXTRANS**）
- 修改存储参数
- 将表移动到新的数据段或表空间
- 明确地分配区域，或是重新分配未使用的空间
- 修改表的登录属性
- 修改 **CACHE/NOCACHE** 属性
- 增加、修改或删除与表相关的完整性约束

- 启用或禁用与表相关的完整性约束或
- 重命名表
- 增加或修改索引管理表的特性
- 增加或修改 LOB 数据列

用户可以增加现有数据列的长度。但是，用户不能将其减小，除非表中没有数据行（列）。另外，如果用户要修改 **CHAR** 类型数据列的长度，就要意识到这是一个耗费时间的操作，并且需要大量的辅助存储空间，特别是表中包含许多数据行（列）时。因为每一数据行（列）中的 **CHAR** 值都要为符合新的数据列长度而填补空格。

当改变表的数据块使用参数（**PCTFREE** 和 **PCTUSED**）时，要注意新的设置将应用于表中所有的数据块，包括已分配的和其后将分配给表的。但是当参数改变后，已分配给表的数据块并不立即重组，只是在需要的时候才进行重组。第 12 章中讲述了数据块的存储参数。

当改变表的事务条目设置（**INITRANS**, **MAXTRANS**）时，要注意 **INITRANS** 的新设置只应用于其后将分配给表的数据块，而此时 **MAXTRANS** 的新设置将应用于表中所有的数据块（已分配给表的和将分配给表的）。要更好地理解事务条目设置参数，参见第 12 章的“事务项设置（**INITRANS** 和 **MAXTRANS** 参数）”一节。

存储参数 **INITIAL** 和 **MINEXTENTS** 是不能改变的。其它存储参数（如 **NEXT**、**PCTINCREAS**）的新设置只能影响其后将分配给表的区域。下一个分配的区域大小由 **NEXT** 和 **PCTINCREASE** 的当前值确定，与这些参数的前一个值无关。在第 12 章的“设置存储参数”小节中讲述了这些参数。

用户使用 **ALTER TABLE** 语句改变表。下列语句改变表 **EMP**，修改了它的数据块存储参数，并增加了名为 **BONUS** 的数据列。

```
ALTER TABLE emp
  ADD (bonus NUMBER (7,2))
  PCTFREE 30
  PCTUSED 6;
```

**ALTER TABLE** 语句的其它用法在下面章节中介绍。

**警告** 在改变表前，用户应熟知这样做的结果。

如果表中加入新的数据列，其初始为空。只有表中没有任何数据行（列）时，才可以加入对表有 **NOT NULL** 约束的数据列。

如果有视图或 PL/SQL 程序依赖于一个基表，那么基表的改变将影响依赖于其上的对象。参见第 19 章的“管理对象相关性”一节以了解 Oracle 如何管理从属关系。

### 13.3.1 将表移动到新的数据段或表空间中

**ALTER TABLE...MOVE** 语句允许用户将非分区表的数据重新定位于新的数据段中，而且可以是其它任意一个用户拥有其分配额的表空间。此语句还允许用户修改表的任何一个存储属性，包括那些用 **ALTER TABLE** 不能修改的参数。

使用下列语句将表 **EMP** 移动到新的数据段中，并指定新的存储参数。

```
ALTER TABLE emp MOVE
```



```
STORAGE (INITIAL 20K
          NEXT 40K
          MINEXTENTS 2
          MAXEXTENTS 20
          PCTINCREASE 0);
```

如果表中包含 LOB 数据列，使用此语句移动表的同时，可以连同 LOB 数据以及用户明确指定的 LOB 索引数据段（与移动的表相关的）一起移动。

### 13.3.2 手工分配表的存储空间

当有需要时，Oracle 为表的数据段动态的分配增加的区域。但用户可能要为表明确地分配一个附加的区域。例如在使用 Oracle 并行服务程序时，表的区域可以明确地分配给某一指定的事例。

使用带有 ALLOCATE EXTENT 选项的 ALTER TABLE 语句，可以为表分配一个新的区域。第 12 章的“释放空间”一节讲述了这一内容。

### 13.3.3 删除数据列

Oracle 允许用户删除表中不再需要的数据列，包括索引管理表中的数据列。这提供了一种方便的释放空间的方法，而且可以避免导入导出数据然后再重建索引和约束。要使用下列任何删除数据列的指令，用户必须具有在目标表中的 ALTER 权限，或是拥有 ALTER ANY TABLE 的系统授权。

用户不能删除表中所有的数据列，也不能删除属于 SYS 表的数据列。任何这样的尝试都将产生错误。

#### 从表中删除数据列

当用户使用 ALTER TABLE...DROP COLUMN 语句时，数据列描述符以及表中每一个与目标数据列相关的数据都将被删除。可以在一个语句中删除多个数据列。下列是删除表 EMP 中数据列的例子。

仅删除 SAL 数据列：

```
ALTER TABLE emp DROP COLUMN sal;
```

删除 SAL 和 COMM 数据列：

```
ALTER TABLE emp DROP (sal,comm);
```

#### 标记未用数据列

如果用户担心从大型表中删除数据列的数据会花费太多的时间，可以使用 ALTER TABLE...SET UNUSED 语句。此语句可以标记一个或多个数据列为未用的，但并不删除目标数据列中的数据，或是释放被其占用的磁盘空间。但是，被标记为未用的数据列在查询或数据字典视图中不会显示，而且该数据列的名称将被删除，新的数据列可以重新使用这一名称。基于此数据列的所有约束、索引以及统计信息也都被删除。

执行下列指令将 SAL 和 COMM 数据列标记为未用：

```
ALTER TABLE emp SET UNUSED (sal,comm);
```

用户可以在以后使用 `ALTER TABLE...DROP UNUSED COLUMNS` 语句删除被标记为未用的数据列。无论何时用户明确地删除特定的数据列，被标记为未用的数据列也将从目标表中删除。

数据字典视图 `USER_UNUSED_COL_TABS`、`ALL_UNUSED_COL_TABS` 或 `DBA_UNUSED_COL_TABS` 可以用来列出包含未用数据列的所有的表。`COUNT` 域显示表中未用数据列的数量。

```
SELECT * FROM dba_unused_col_tabs;
OWNER          TABLE_NAME          COUNT
-----
SCOTT          EMP                      1
1 row select
```

#### 删除未用数据列

`ALTER TABLE...DROP UNUSED COLUMNS` 语句是唯一可用于未用数据列的操作的。它在物理上删除了表中未用数据列，并重新声明磁盘空间。

例句中指定了可选关键字 `CHECKPOINT`。这一选项在处理指定数量的数据行后，本例中是 250，将产生一个检验点。使用检验点可以在删除数据列的操作过程中削减复原日志的容量累积，以避免回退数据段空间的潜在消耗。

```
ALTER TABLE emp DROP UNUSED COLUMNS CHECKPOINT 250;
```

## 13.4 删除表

要删除表，该表必须包含在用户模式中，或者用户具有 `DROP ANY TABLE` 的系统授权。

使用 `DROP TABLE` 语句可以删除不再需要的表。下列语句删除了表 `EMP`。

```
DROP TABLE emp;
```

如果要删除的表中包含的主关键字或唯一关键字涉及到其它表的外部关键字，而且用户打算删除子表的 `FOREIGN KEY` 约束，那么就要使用如下所示包含 `CASCADE` 选项的 `DROP TABLE` 语句：

```
DROP TABLE emp CASCADE CONSTRAINTS;
```

**警告** 在删除表之前，用户应该熟知这样做的结果：

- 删除表将从数据字典中删除该表的定义。表的所有数据将不再可存取。
- 所有与表有关的索引和触发器都将删除。
- 所有依赖该表的视图和 PL/SQL 程序单元仍然保留，但都成为无效的（不可用）。
- 保留表的所有别名，但当使用时会返回错误。
- 分配给非簇表的区域在表被删除时作为自由空间返回给表空间，供其它数据对象申请区域时分配，或分配给新的数据对象。与簇表相对应的所有数据行将从簇的存储块中删除。

用户可能是要截断表而不是删除它。TRUNCATE 语句提供了一种快速有效的方法删除表中所有的数据，而不影响与表相关的任何结构关系（数据列的定义、约束、触发器，等等）或是授权。TRUNCATE 语句将在第 19 章的“截断表和簇”部分讲述。

## 13.5 索引管理表

本节讲述管理索引管理表方面的内容，包括下列主题：

- 什么是索引管理表
- 创建索引管理表
- 维护索引管理表
- 分析索引管理表
- 针对索引管理表使用 ORDER BY 子句
- 转换索引管理表为规则表

### 13.5.1 什么是索引管理表

索引管理表是数据行按照主关键字组织的数据表。簇是通过使用 B\*-树状结构索引实现的。B\*-树状结构索引是索引树的一种特殊类型，与规则表的 B-树状结构索引不同，规则表的 B-树状结构索引既保存了主关键字，也保存了其它非关键字数据列。索引管理表的属性按照索引完全存储在数据物理结构中。

#### 为什么要使用索引管理表

索引管理表为包含精确匹配的查询和范围搜索提供了对表数据基于关键字的快速存取。表数据的改变（诸如增加新的数据行，更新数据或删除数据等）仅导致索引结构的更新（因为没有分离的表存储区）。

而且，存储容量的需求也减小了，因为关键字数据列没有在表和索引中出现重复。其余的非关键字数据列存储在索引结构中。

索引管理表在用户的应用程序必须基于主关键字恢复数据时特别有用。索引管理表也适用于模仿指定应用程序的索引结构。例如，包含文本、图像和声音数据的基于内容的信息恢复应用程序要求反向索引，而这一反向索引可以用索引管理表非常有效地进行模仿。

#### 索引管理表与规则表之间的区别

索引管理表与规则表在其数据列的主关键字索引上是相似的。但索引管理表中只保留单一的 B\*树状结构索引，包含表的主关键字和其它数据列的值，而不是像规则表中保留有两个相对独立的存储空间——表和 B\*树状结构索引。

下图说明了规则表与索引管理表之间的结构差异。

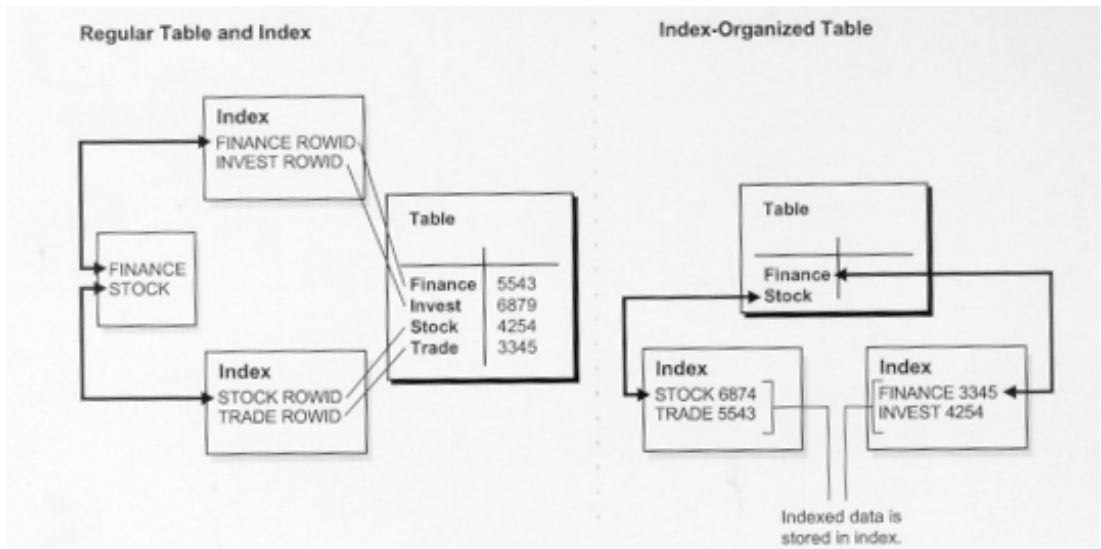


图 13-1 规则表与索引管理表结构比较

索引管理表适用于用主关键字或其它以主关键字为有效前缀的关键字存取数据。因为没有创建包含关键字值和 ROWID 的相对独立的索引，所以没有重复的关键字值，存储量的需求也减小了。

### 13.5.2 创建索引管理表

用户使用 CREATE TABLE 语句创建索引管理表，但用户需要提供下列附加信息：

- ORGANIZATION INDEX 限定词，声明这是一个索引管理表。
- 主关键字，通过数据列约束子句（针对单数据列主关键字）或表约束子句（针对多数据列主关键字）指定。必须为索引管理表指定主关键字。
- 可选的数据溢出规定子句（OVERFLOW），通过将超出指定阈值的数据行存入相对分离的溢出数据段中，可以保持索引管理表紧凑的簇化结构。INCLUDING 子句可以用来指定哪些数据列（非关键字的）被存入溢出数据段中。
- PCTTHRESHOLD 值，定义索引数据块中为索引管理表保留的空间比例。任何数据行超出阈值的部分都将存入溢出数据段中。也就是说，数据行在数据列边界处被断为两部分，头部和尾部。头部适合指定的阈值，与关键字一起存储在索引页数据块中。尾部与其它数据行的部分一样，存储在溢出数据段中。因此，索引条目包括关键字值、非关键字数据行适合指定阈值的部分，以及指向数据行其它部分的指针。

下例创建了一个索引管理表：

```
CREATE TABLE docindex(
    token char(20),
    doc_id NUMBER,
    token_frequency NUMBER,
    token_offsets VARCHAR2(512),
    CONSTRAINT INDEX TABLESPACE ind_tbs
```

```
PCTTHRESHOLD 20
OVERFLOW TABLESPACE ovf_tbs ;
```

上例显示：**ORGANIZATION INDEX** 限定词指明了索引管理表，表中关键字数据列和非关键字数据列驻存在索引中，索引是基于为表指定的主关键字数据列（**TOKEN**，**DOC\_ID**）定义的。

索引管理表可以存储对象类型。例如，可以创建一个索引管理表，表中包含 **MYTYPE** 对象类型的数据列，例句如下：

```
CREATE TABLE iot (c1 NUMBER primary key,c2 mytype)
ORGANIZATION INDEX;
```

但是，用户不能创建对象类型的索引管理表。例如，下列语句是无效的：

```
CREATE TABLE iot OF mytype ORGANIZATION INDEX;
```

#### 使用 AS 子查询

用户可以用 **AS** 子查询创建一个索引管理表。使用这种方法创建的索引管理表通过 **PARALLEL** 选项可以并行调用。

下列语句从常规表 **RT** 中选择一些数据行创建了一个索引管理表（并行的）：

```
CREATE TABLE iot (i PRIMARY KEY,j) ORGANIZATION INDEX PARALLEL
(DEGREE 2) AS SELECT * FROM rt;
```

#### 使用溢出子句

前面指定溢出子句的例子说明数据行的任何非关键字数据列，如果其超过数据块尺寸的 20%，将会放在存储于 **OVF\_TBS** 表空间的数据段中。关键字数据列应适合指定的阈值。

如果非关键字数据列的更新使数据行尺寸减小，**Oracle** 将判别更新的数据行片断（头部或尾部）并重写该片断。

如果非关键字数据列的更新使数据行尺寸增加，**Oracle** 将判别更新的数据行片断（头部或尾部）并重写该片断。如果更新的目标是头部片断，要注意这一片断将可能再次被打断为两部分，以保持行的尺寸小于阈值。

适合于索引页数据块的非关键字数据列将存储为行的头部片断，其中包含 **ROWID** 字段将其连到存储于溢出数据段中行的其它部分。只有不适合的数据列存储在溢出区域中。

**选择和监视阈值** 应该选择适应主键列和最前面的几个非主键列（如果它们被频繁访问）的阈值。

在选择阈值以后，可以监视表来验证所指定的值是否合适。可以使用 **ANALYZE TABLE...LIST CHAINED ROWS** 语句来决定超出阈值的行的号和标示。

**使用 INCLUDE** 子句除了指定 **PCTTHRESHOLD** 值以外，使用 **INCLUDE <column\_name>** 子句还可以控制那些非关键列能与关键列一起存储。**Oracle** 把直到在 **INCLUDE** 子句中指定的列为止的所有非关键列都适应索引叶块，其前提是它们没有超出指定的阈值。所有超出在 **INCLUDE** 子句中指定的列的非关键列存储在溢出区域。

**注意** 为了提供有效的基于主键的访问，**Oracle** 把一个索引管理表中的所有主键列都移动到表的开始（以它们的键顺序）。如下例所示：

```
CREATE TABLE io(a INT, b INT, c INT, d INT,
                Primary key(c,b))
  ORGANIZATION INDEX ;
```

存储行的顺序是：c、b、a 和 d（而不是 a、b、c 和 d）。根据存储行的顺序，最后的主键列是 b。INCLUDE 子句中所用的行可以是最后的主键列（本例中是 b），或者是任何非键列（即存储行的顺序中在 b 后面的列）。

前面提供的例子可以进行修改，以创建索引管理表，其中 **TOKEN\_OFFSETS** 列的值总是存储在溢出区域：

```
CREATE TABLE docindex(
  token CHAR(20),
  doc_id NUMBER,
  token_frequency NUMBER,
  token_offsets VARCHAR2(512),
  CONSTRAINT pk_docindex PRIMARY KEY (token, doc_id))
  ORGANIZATION INDEX TABLESPACE ind_tbs
  PCTTHRESHOLD 20
  INCLUDING token_frequency
  OVERFLOW TABLESPACE ovf_tbs ;
```

这里，只有到 **TOKEN\_FREQUENCY** 的非键列（本例中只有单个列）与键列值一起存储在索引叶块中。

### 使用键压缩

使用键压缩来创建一个索引管理表可以消除键列前缀值的重复。

键压缩把一个索引键分成前缀和后缀项。通过在一个索引块中的所有后缀项之间共享前缀项来达到压缩的目的。这种共享能节省大量的空间，从而在改善性能的同时，在每个索引块中存储更多的键。

可以在进行下列操作时使用 **COMPRESS** 子句来启用键压缩：

- 创建索引管理表
- 移动索引管理表

也可以指定前缀的长度（如同键列的数目一样），前缀长度标识键列怎样分成前缀和后缀项。

```
CREATE TABLE iot(i INT, j INT, k INT, l INT, PRIMARY KEY(i,j,k))
  ORGANIZATION INDEX COMPRESS ;
```

上面的语句等价于下列语句：

```
CREATE TABLE iot(i INT, j INT, k INT, l INT, PRIMARY KEY(i,j,k))
  ORGANIZATION INDEX COMPRESS 2 ;
```

对于 (1,2,3)、(1,2,4)、(1,2,7)、(1,3,5)、(1,3,4)、(1,4,4) 这组值，重复出现的 (1,2) 和 (1,3) 被压缩掉。

你也可以覆盖用于压缩的缺省前缀长度，如下所示：

```
CREATE TABLE iot(i INT, j INT, k INT, l INT, PRIMARY KEY(i,j,k))
  ORGANIZATION INDEX COMPRESS 1 ;
```

对于 (1,2,3)、(1,2,4)、(1,2,7)、(1,3,5)、(1,3,4)、(1,4,4)这组值，重复出现的 1 被压缩掉。可以按如下方式禁用压缩：

```
ALTER TABLE A MOVE NOCOMPRESS ;
```

### 13.5.3 维护索引管理表

索引管理表不同于规则表的唯一之处是物理组织；逻辑上，它们是以相同的方式处理的。在 INSERT、SELECT 和 UPDATE 语句中可以用索引管理表代替规则表。

#### 改变索引管理表

可以使用 ALTER TABLE 语句来修改主键索引和溢出数据段的物理和存储属性。在 OVERFLOW 关键字后面的指定的所用属性都可以应用到溢出数据段。例如，你可以把主键索引段的 INITRANS 设置为 4，把数据段的 OVERFLOW INITRANS 设置为 6，如下所示：

```
ALTER TABLE docindex INITRANS 4 OVERFLOW INITRANS 6 ;
```

也可以改变 PCTTHRESHOLD 和 INCLUDING 列的值。在后续的操作中，使用一个新设置来把行分成头和溢出尾片。例如，改变 DOCINDEX 表的 PCTTHRESHOLD 和 INCLUDING 列的值，如下所示：

```
ALTER TABLE docindex PCTTHRESHOLD 15 INCLUDING doc_id ;
```

通过把 INCLUDING 列设置为 DOC\_ID, 在 TOKEN\_FREQUENCY 和 TOKEN\_OFFSETS 后面的所有列都存储在溢出数据段中。

对于创建成没有溢出数据段的索引管理表，可以使用 ADD OVERFLOW 子句来加入一个溢出数据段。例如，如果 DOCINDEX 表没有溢出数据段，那么可以加入一个溢出段，如下所示：

```
ALTER TABLE docindex ADD OVERFLOW TABLESPACE ovf_tbs ;
```

#### 移动（重建）索引管理表

因为索引管理表主要以 B 型树索引方式进行存储，所以可能遇到由于不断更新所造成的破碎问题。但是，可以使用 ALTER TABLE ...MOVE 语句来重建索引，减少碎片。

下列语句在把 INITRANS 设置为 10 以后重建索引管理表：

```
ALTER TABLE docindex MOVE INITRANS 10 ;
```

使用 ONLINE 选项，可以在没有溢出数据段联机时移动索引管理段。例如，如果 DOCINDEX 表没有溢出数据段，那么可以完成联机移动，如下所示：

```
ALTER TABLE docindex MOVE ONLINE INITRANS 10 ;
```

下列语句重建索引管理表 DOCINDEX 及其溢出数据段：

```
ALTER TABLE docindex MOVE TABLESPACE ix_tbs OVERFLOW TABLESPACE ov_tbs ;
```

在下面的语句中，索引管理表 IOT 被移动，同时重建 LOB 索引和数据段 C2:

```
ALTER TABLE iot MOVE LOB (C2) STORE AS (TABLESPACE lot_ts) ;
```

### 更新键列实例

键列更新逻辑上相当于删除带旧键值的行，在合适的位置插入带新键值的行，以维护主键顺序。

在下面的例子中，dept\_id=20 和 e\_id=10 的雇员行逻辑上被删除，插入 dept\_id=23 和 e\_id=10 的雇员行：

```
UPDATE employees
SET dept_id = 23
WHERE dept_id = 20 and e_id = 10 ;
```

### 13.5.4 分析索引管理表

正如常规表一样，索引管理表的分析也使用 ANALYZE 语句：

```
ANALYZE TABLE docindex COMPUTE STATISTICS ;
```

ANALYZE 语句分析主键索引段和溢出(overflow)数据段，计算表的逻辑和物理统计。

- 使用 USER\_TABLE、ALL\_TABLE 或 DBA\_TABLE 可以查询逻辑统计。
- 使用 USER\_INDEXES、ALL\_INDEXES 或 DBA\_INDEXES（并使用主键索引名）可以查询物理统计。例如，可以获得表 docindex 的主键索引段的物理统计，如下所示：

```
SELECT * FROM DBA_INDEXES WHERE INDEX_NAME = 'PK_DOCINDEX';
```

- 使用 USER\_TABLE、ALL\_TABLE 或 DBA\_TABLE 可以溢出数据段的物理统计。可以通过 IOT\_TYPE = 'IOT\_OVERFLOW' 查询来标识溢出项。例如，你可以获得与表 docindex 相关的溢出数据段的物理属性，如下所示：

```
SELECT * FROM DBA_TABLES WHERE IOT_TYPE='IOT_OVERFLOW'
and IOT_NAME = 'DOCINDEX';
```

### 13.5.5 与索引管理的表一起使用 ORDER BY 子句

如果 ORDER BY 子句只引用主键列或其前缀，那么优化器避免由于返回的行在主键列上排序而带来的排序额外开销。

例如，你创建了下列表：

```
CREATE TABLE employees (dept_id INTEGER, e_id INTEGER, e_name
VARCHAR2, PRIMARY KEY (dept_id, e_id)) ORGANIZATION INDEX ;
```

下列查询避免了排序额外开销，因为数据已经在主键列上进行了排序：

```
SELECT * FROM employees ORDER BY (dept_id, e_id);
SELECT * FROM employees ORDER BY (dept_id);
```

但是，如果拥有针对主键列的后缀或非主键列的 ORDER BY 子句，则需要额外的排序（假设没有定义其它二级索引）。

```
SELECT * FROM employees ORDER BY (e_id);
SELECT * FROM employees ORDER BY (e_name);
```



### 13.5.6 把索引管理的表转化为规则表

使用 Oracle `IMPORT/EXPORT` 实用工具或 `CREATE TABLE...AS SELECT` 语句可以把索引管理的表转化为规则表。

要把索引管理的表转化为规则表，必须进行下列操作：

- 使用方便的路径输出索引管理的表
- 用相同的定义创建规则表定义
- 输入索引管理的表的数据，保证 `IGNORE=y`(确保存在错误的对象被忽略)

**注意** 在把索引管理的表转化为规则表之前，应该意识到不能使用 Oracle8 以前版本的 `Export` 实用工具来输出索引管理的表。

# 14 管理索引

本章叙述索引管理的各个方面，包括下列主题：

- 索引管理指南
- 创建索引
- 改变索引
- 监视索引的空间使用
- 删除索引

## 14.1 索引管理指南

本节叙述管理索引时要遵循的指南，包括下列主题：

- 插入列表数据后创建索引
- 限定每一表中索引的数量
- 指定事务项参数
- 规定索引存储块空间的使用
- 计算索引容量并设置存储参数
- 为每个索引指定表空间
- 并行创建索引
- 研究使用 **NOLOGGING** 参数创建索引
- 估量合并或者重建索引的耗费和利益
- 在禁用或者删除约束之前考虑耗费

索引是与表和簇相关的非必需的数据结构，用户可以明确地创建它以加快关于一个表的 SQL 语句的执行。正如这本手册的索引，与没有索引相比，可以帮助用户更快的找到信息，Oracle 索引提供了加快访问列表数据的途径。

Oracle 提供若干索引模式，这些索引模式提供了补充的操作功能：**B\*-树索引**(目前最常用的)、**B\*-树簇索引**、**散列簇索引**、**反向关键字索引**和**位图索引**。Oracle 还支持基于函数的索引和专门用于应用程序或字体盒的域索引。

索引的存在与否不要求任何 SQL 语句语法的改变。索引只是提供了对数据的快速访问路径；它只影响执行的速度。对于给定的编入索引的数据值，索引直接指向包括那个值的数据行(列)位置。

在相关表内，索引在逻辑上和物理上独立于数据。用户可以随时创建或者删除一个索引而不影响基表或是其它的索引。如果用户删除一个索引，所有的应用程序可以继续工作；只是，对于以前编入索引的数据的存取可能会减慢。作为独立的数据结构，索引需要存储空间。

在索引创建以后，Oracle 会自动地维护和使用它们。Oracle 自动地在所有有关的索引中反映数据变更，比如增加新的数据行，更新数据行，或者删除数据行。

#### 14.1.1 插入列表数据后创建索引

在向表中插入或者加载数据后，用户应该为表创建一个索引。在没有索引的表中插入数据行，然后为其后的存取创建索引是非常有用的。如果用户在加载列表数据之前创建索引，那么每次向表中插入行列时，每个索引都必须更新。用户还必须在向簇中插入数据之前为其创建索引。

当创建的索引是关于一个已经有数据的表时，Oracle 就必须使用排序空间。Oracle 为索引的创建者使用在内存中分配的排序空间(每个使用者的用量由初始化参数 `SORT_AREA_SIZE` 决定)，但是还必须与暂时数据段交换排序信息，暂时数据段是由于创建索引而分配的。

如果索引非常大，用户可能需要执行下列工作：

1. 创建一个新的临时数据段表空间。
2. 改变索引创建者的临时数据段表空间。
3. 创建索引。
4. 如果需要，删除临时数据段表空间并重新指定创建者的临时数据段表空间。

**注意** 在某种情况下，加载数据可以利用 SQL\*Loader 的直接路径加载，而索引可以在数据加载的时候创建。

#### 14.1.2 限制每一表中索引的数量

一个表可以有許多索引。但是，索引越多在修改表时产生的经常开支也越多。具体地说，当数据行插入或者删去时，关于表的所有的索引也同样必须更新。并且，当数据列更新时，所有的包括该数据列的索引也必须更新。

因此需要权衡表数据的检索速度和表的更新速度。例如，如果表主要是只读的，较多的索引可能很有效；但是如果表频繁地更新，较少的索引可能更好。

#### 14.1.3 指定事务项参数

通过在每个索引的创建过程中指定 `INITRANS` 和 `MAXTRANS` 参数，用户可以影响索引数据段的数据块中有多少空间最初可以分配给事务项。用户还应该为更新保留空间，并随后为这些设置确定长期的值(例如，索引的使用期限)。

#### 14.1.4 规定索引存储块空间的使用

当创建表的索引后，索引数据块被表的现有数值填充，直到达到 `PCTFREE` 值。通过 `PCTFREE` 保留的索引存储块空间只在表中插入新的行列时使用，此时对应的索引项必须放置在正确的索引存储块中(也就是说，在前后索引项之间)。如果在有关的索引存储块中没有更多的可用空间，编入索引的值将放置在它所属的地方(基于字顺顺序)。因此如果用户打算在编入索引的表中插入多个数据行，`PCTFREE` 值应该较高以容纳新的索引值。如果表相对稳定，没有太多的数据插入，相关索引的 `PCTFREE` 值可以较低，保存索引数据所需要的存储块也就较少。

**参见** 不能为索引指定 `PCTUSED` 参数。关于 `PCTFREE` 参数的信息参见第 12 章的“管理数据块的内部空间”一节。

#### 14.1.5 计算索引容量并设置存储参数

在创建索引之前估算其容量是很有益的，原因如下：

- 用户可以利用综合估算的索引容量，以及表、回退数据段、重演日志文件的估算值，确定保存一个数据库需要的磁盘空间的总数。根据这些估算，用户可以购买恰当的硬件并做出其它的决定。
- 用户可以利用个别索引的估算容量较好的管理该索引将使用的磁盘空间。在创建索引时，用户可以设置有关的存储参数，并改进使用该索引的应用程序的 I/O 操作性能。

例如，假定用户在创建索引之前估算出它的最大尺寸，然后在创建索引时设置了存储参数，就可以为表的数据段分配较少的区域，并且该索引的所有数据将储存在相对邻接的磁盘空间单元中。这将减少涉及此索引的磁盘 I/O 操作所必需的时间。

单个索引项的最大尺寸近似地为数据块容量的一半。如同对于表的操作，用户可以在创建索引时明确地设置存储参数。

#### 14.1.6 为每个索引指定表空间

索引可以创建在任何表空间中。索引可以在与表相同的或者不同的表空间中创建。

如果用户对表和它的索引使用相同的表空间，那么数据库的维护可能更方便(比如表空间或者文件的备份，以及应用程序的获取或者更新)，并且所有关联数据将总是同时联机。

如果表和索引使用不同的表空间(在不同的磁盘上)，与表和索引存储在相同的表空间相比，由于减少了磁盘冲突，就可以产生较好的操作性能。

如果用户对表和索引使用不同的表空间，而其中一个表空间是脱机的(或是包含数据的，或是包含索引的)，那么就不能保证涉及表的指令起作用。

#### 14.1.7 并行创建索引

用户可以并行创建索引。因为是多个处理过程合作创建索引，与用单个服务程序顺序地创建索引相比，Oracle 可以更快的创建索引。

在并行创建索引时，每个查询服务程序单独使用存储参数。因此，INITIAL 值为 5M 而并行度为 12 的索引创建过程至少消耗 60M 的存储量。

#### 14.1.8 考虑使用 NOLOGGING 参数创建索引

通过在 CREATE INDEX 指令中指定 NOLOGGING 参数，用户可以创建索引，并且生成最小的重演日志记录。

**注意** 因为使用 LOGGING 创建的索引是不存档的，用户在创建索引后应该进行备份。

使用 NOLOGGING 参数创建索引有下列好处：

- 节省重演日志文件中的空间。
- 减少创建索引的时间花费。
- 改进大型索引并行创建的操作性能。

总之，与创建较小的索引相比，不用 LOGGING 创建较大的索引其相对性能改善较大。创建较小的索引时不使用 LOGGING 参数，其在时间花费上的影响很少。但是，对于较大

的索引其性能改善是显著的，特别是用户并行创建索引的时候。

#### 14.1.9 估量合并或者重建索引的耗费和利益

当用户遇到索引存储碎片时(由于不恰当的容量调整或者增加)，用户可以重建或者合并索引。在用户进行任何一项工作之前，应该权衡每个选项的耗费和利益，并选择对于用户的情况作用最好的选项。表 14-1 是重建和合并索引时其耗费与利益的比较。

表 14-1 重建和合并方法的比较

| 重建                            | 合并              |
|-------------------------------|-----------------|
| 快速移动索引到另一个表空间                 | 无法移动索引到另一个表空间   |
| 较高的耗费。需要更多的磁盘空间               | 较低的耗费。不需要更多磁盘空间 |
| 创建新的树型结构，如果合适将缩小高度            | 把叶块合并到树的相同分枝内   |
| 使用户能够快速变更存储量和表空间的参数，而不必删除初始索引 | 快速释放索引页块以便使用    |

如果用户具有可以释放以便重新使用的 B\*-树索引页存储块，在此情况下，用户可以使用下列指令合并页存储块：

```
ALTER INDEX vmoore COALESCE;
```

图 14-1 说明了在索引 VMOORE 上 ALTER INDEX COALESCE 语句的作用。在执行操作前，开头两个页存储块是 50% 满，这表明用户有机会减少存储碎片，并完全装满第一个存储块，同时释放第二个存储块(此例中假定 PCTFREE=0)。

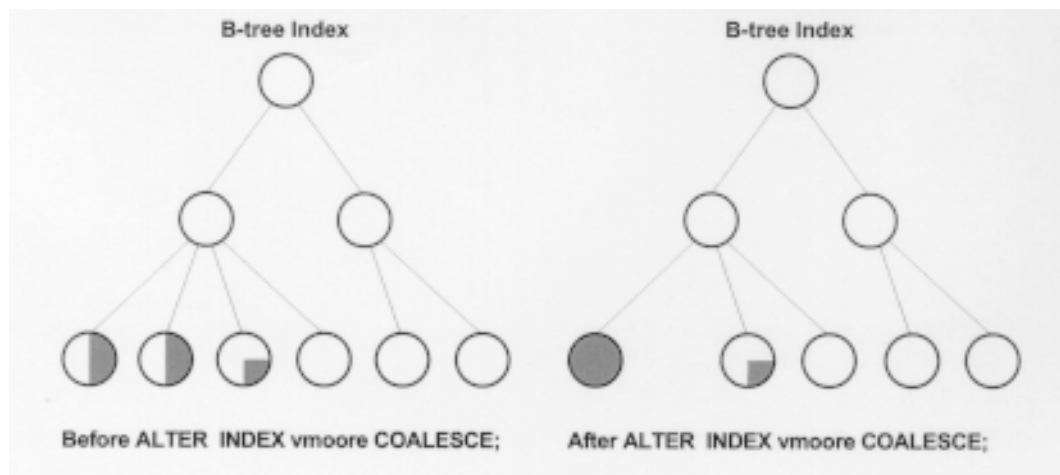


图 14-1 合并索引

#### 14.1.10 在禁用或删除约束之前考虑耗费

因为单值关键字和主关键字具有相关的索引，用户应该在考虑是否禁用或者删除 UNIQUE 或者 PRIMARY KEY 约束时，从删除和创建索引的耗费中提公因子。如果 UNIQUE 关键字或者 PRIMARY KEY 约束的相关索引非常大，用户可以通过保留约束而不是删除和重新创建大型索引的方法来节省时间。

## 14.2 创建索引

本节叙述如何创建索引，包括下列主题：

- 明确地创建索引
- 创建与约束相关的索引
- 联机创建索引
- 创建基于函数的索引
- 重建现有索引
- 创建压缩关键字的索引

要以用户模式创建索引，下列条件之一必须为真：

- 编入索引的表或者簇必须是在用户自己的模式中。
- 用户必须对要编入索引的表有 INDEX 授权。
- 用户必须有 CREATE ANY INDEX 的系统授权。

要以别的模式创建索引，用户必须有 CREATE ANY INDEX 的系统授权。而且，对于要包含索引的模式，其文件编写者必须具有包含索引或者索引分区的表空间的空间分配额，或是具有 UNLIMITED TABLESPACE 的系统授权。

### 14.2.1 明确地创建索引

用户可以使用 SQL 语句 CREATE INDEX 明确地创建索引(完整性约束除外)。下列指令创建了表 EMP 中数据列 ENAME 的名为 EMP\_ENAME 的索引：

```
CREATE INDEX emp_ename ON emp ( ename)
      TABLESPACE users
      STORAGE (INITIAL 20K NEXT 20k
      PCTINCREASE 75)
      PCTFREE 0 ;
```

注意对该索引明确地指定了若干存储设置。如果用户没有为索引指定存储选项(比如 INITIAL 和 NEXT)，将自动使用宿主表空间的缺省存储选项。

LOBS、LONG 和 LONG RAW 数据列无法编入索引。

### 14.2.2 创建与约束相关的索引

Oracle 通过对单值关键字或者主关键字创建唯一索引的方法强化 UNIQUE KEY 或者 PRIMARY KEY 完整性约束。当约束有效时 Oracle 自动地创建此索引；CREATE TABLE 或者 ALTER TABLE 指令的发布者不需要执行操作创建该索引。这包括两种情况：定义并启用约束、启用已定义但被禁用的约束。

要使 UNIQUE 关键字或者 PRIMARY KEY 起作用(它们将创建相关的索引)，表的所有者需要有包含该索引的表空间的分配额，或是 UNLIMITED TABLESPACE 的系统授权。

对于与 UNIQUE 关键字和 PRIMARY KEY 约束相关的索引，用户可以使用带有 USING INDEX 选项的 ENABLE 子句为其设置存储选项。下列指令定义了 PRIMARY KEY 约束，

并指定了相关索引的存储选项：

```
CREATE TABLE emp (  
    empno NUMBER ( 5 ) PRIMARY KEY, age INTEGER )  
    ENABLE PRIMARY KEY USING INDEX  
    TABLESPACE users PCIFREE 0 ;
```

Oracle 建议用户不要对表明确地定义 UNIQUE 索引(CREATE UNIQUE INDEX)。总之，创建约束以加强唯一性比使用 CREATE UNIQUE INDEX 要好。约束的相关索引总是采用该约束的名字，用户不能为其指定具体的名字。

### 14.2.3 联机创建索引

以前，当创建表的索引时，总存在 DML S- lock，意思是用户在构造期间不能对基表执行 DML 操作。

现在，随着表的容量日益增加以及对于连续作业的需要，用户可以联机创建和重建索引，也即用户可以在建造或者重建表的索引的同时更新基表。注意，仍然存在 DML SS -locks，它意味着用户在联机索引建立期间不能执行其它的 DDL 操作。

下列指令完成联机索引建立的操作：

```
ALTER INDEX emp name REBUILD ONLINE;  
CREATE INDEX emp_name ON emp (mgr, empl, emp2, emp3) ONLINE;
```

**注意** 虽然用户在联机索引建立期间可以执行 DML 操作，但是 Oracle 建议用户在此过程中不要执行较大的 DML 操作。例如，如果用户要加载数据行，其总数相当于表容量的 30%，那么用户应该在联机索引建立之前完成加载操作。

### 14.2.4 创建基于函数的索引

因为限定了函数或者表达式返回值，所以基于函数的索引简化了查询。函数或者表达式的值经过预先计算并储存在索引中。基于函数的索引的具体特点包括：

- 用户可以在检索关键字为表达式的情况下创建索引。
- 为评估涉及函数的逻辑谓词提供了有效的手法。
- 支持基于语言学排序关键字(校对)的语言排序。语言排序索引可用于 SQL 语句中高效的语言校对。
- 用户可以执行不区分大小写的排序。
- 用户可以创建精确降序索引。它们实际是作为基于函数的索引的一个特殊情况。

要说明基于函数的索引，可以研究一下下列指令，它们在函数 AREA( GEO)上定义了一个基于函数的索引 (AREA\_INDEX)：

```
CREATE INDEX area index ON rivers (Area(geo));
```

在下列 SQL 语句中，当在 WHERE 子句中引用了 AREA( GEO)时，较优化的设想是使用索引 AREA INDEX。

```
SELECT      id, geo, Area(geo), desc  
FROM        rivers r  
WHERE       Area (geo) >5000;
```

表的所有者应该有在基于函数的索引中使用函数的 EXECUTE 授权。而且，因为基于函数的索引依据任何它所使用的函数，当函数改变时，它可能会失去作用。如果函数有效，对于被禁用的基于函数的索引，用户可以使用 ALTER INDEX...ENABLE 指令恢复其正常操作。ALTER INDEX...DISABLE 语句允许用户禁用一个基于函数的索引。如果用户在维护函数体，则可能要执行此语句。

为了在用户自己的模式中创建基于函数的索引，用户必须有 CREATE INDEX 和 QUERY REWRITE 的系统授权。在其它模式中创建索引，或是创建其它模式表的索引，用户必须有 CREATE ANY INDEX 和 GLOBAL QUERY REWRITE 的授权。

用户必须定义下列初始化参数以创建基于函数的索引：

- QUERY REWRITE INTEGRITY 必须设置为 TRUSTED
- QUERY REWRITE ENABLED 必须设置为 TRUE
- COMPATIBLE 必须设置为 8.1.0.0.0 或者更大的值

另外，要使用基于函数的索引还要满足下列要求：

- 在创建索引后必须对表进行分析。
- 必须保证查询中不需要任何来自编入索引的表达式的 null 值，因为 null 值不储存在索引中。

**注意** CREATE INDEX 会存储基于函数的索引使用的最新函数的时间标签。当索引生效时，这个时间信息被更新。当对于表空间中基于函数的索引执行时间点恢复时，如果索引使用的大部分最新函数的时间信息比储存在索引中的时间信息新，那么该索引将被标记为无效的。用户必须使用 ANALYZE VALIDATE INDEX 语句使这个索引生效。

#### 例 1

下列语句在表 EMP 上基于对 ENAME 数据列的大写转换创建了基于函数的索引 IDX：

```
CREATE INDEX idx ON emp (UPPER(emp_name));
```

现在 SELECT 语句使用关于 UPPER( EMP\_NAME)的基于函数的索引检索所有名字以 JOH 开头的雇员：

```
SELECT * FROM emp WHERE UPPER (emp_name) LIKE 'JOH%';
```

本例还说明了不区分大小写的检索。

#### 例 2

下列语句创建一个关于表达式的基于函数的索引：

```
CREATE INDEX idx ON t (a + b * (c - 1), a, b);
```

SELECT 语句既可以使用索引范围搜索(在下列 SELECT 语句中，表达式是索引的前缀)，还可以使用索引整体搜索(当索引指定了高并行度时较适宜)。

```
SELECT a FROM t WHERE a + b * (c - 1) < 100;
```

#### 例 3

用户还可以使用基于函数的索引支持 NLS 排序检索。NLSSORT 是一个函数，返回给



定字符串的排序关键字。因此，如果用户要使用 NLSSORT 构造关于 NAME 的索引，可以使用下列语句：

```
CREATE INDEX nls_index ON T_TABLE (NLSSORT (name, 'NLS_SORT = German' ));
```

该语句以校对序列 German 创建了关于表 T\_TABLE 的索引 NLS\_INDEX。

现在，要用 NLS\_SORT 索引在 T\_TABLE 中进行选择：

```
SELECT * FROM T_TABLE ORDER BY name;
```

数据行将用校对顺序 German 进行排序。

#### 例 4

本例综合了不区分大小写的排序和语言排序。

```
CREATE INDEX empi ON emp  
UPPER ( (ename), NLSSORT (ename) );
```

这里，在 NLSSORT 参数中没有出现对 NLS\_SORT 的规定，因为 NLSSORT 着眼于语言排序关键字的语言设置。例 3 中说明了指定 NLS\_SORT 的情况。

### 14.2.5 重建现有索引

在重建现有索引之前，应按照表 14-1 中的叙述将重建索引的耗费和利益与合并索引的耗费和利益进行比较。

用户可以利用现有的索引作为数据源创建索引。如此创建的索引，用户可以改变其存储特性，或者将其移到新的表空间中。基于现有数据源重建索引还可以删除存储块内部的存储碎片。实际上，与删除索引并使用 CREATE INDEX 语句相比，重新创建现有索引可以提供比较好操作性能。

使用下列语句重建现有的索引：

```
ALTER INDEX index_name REBUILD,
```

REBUILD 子句必须紧跟索引名，并先于任何其他选项。而且，REBUILD 子句不能连同 DEALLOCATE UNUSED 子句使用。

### 14.2.6 创建压缩关键字的索引

使用关键字压缩创建索引使用户能够消除关键列标头值的重复出现。

关键字压缩使索引关键字断开成为前缀和后缀条目。压缩是在索引存储块中通过所有的后缀条目共享前缀条目而实现的。这一共享可以极大的节省空间，使用户可以在每个索引存储块中存储更多的关键字，同时改进了操作性能。

关键字压缩在下列情况下很有用：

- 用户有一个非唯一索引，要附加 ROWID 以使关键字唯一。如果用户使用关键字压缩，重复的关键字将作为前缀条目储存在索引存储块中，不带 ROWID。剩下的数据行将是只包含 ROWID 的后缀条目。
- 用户有一个唯一的多数数据列索引。

用户可以使用 COMPRESS 子句使关键字压缩起作用。用户还可以指定前缀长度(按照关键字数据列的数值)，以确定关键字数据列如何断成前缀和后缀。例如，下列语句将压缩

索引页存储块中关键字的重复出现。

```
CREATE INDEX emp_ename ( ename)
    TABLESPACE users
    COMPRESS 1
```

在重建期间同样可以指定 **COMPRESS** 子句。例如，在重建期间用户可以禁用压缩，如下所示：

```
ALTER INDEX emp_ename REBUILD NOCOMPRESS;
```

### 14.3 改变索引

要改变索引，用户的模式中必须包含该索引，或者用户必须有 **ALTER ANY INDEX** 的系统授权。用户可以重建或者合并索引，改变它的实际存储特性和缺省存储特性以及其它的物理属性，但是用户不能改变它的数据列结构。

使用 **SQL** 语句 **ALTER INDEX**，可以改变任何索引的存储参数，包括那些通过 **Oracle** 创建的、用以加强原始的以及唯一的關鍵字完整性约束的参数。例如，用下列语句改变索引 **EMP\_ENAME**：

```
ALTER INDEX emp_ename
    INITRANS 5
    MAXTRANS 10
    STORAGE ( PCTINCREASE 50 );
```

当用户改变索引的事务项设置时，**INITRANS** 的新设置只应用于其后分配的数据块，而 **MAXTRANS** 的新设置适用于索引的全部存储块(当前的以及其后分配的存储块)。

存储参数 **INITIAL** 和 **MINEXTENTS** 不能被改变。所有其它存储参数的新设置只影响其后为索引分配的区域。

对于执行完整性约束的索引，用户还可以通过使用 **ALTER TABLE** 语句调整其存储参数，语句中包括带有 **USING INDEX** 选项的 **ENABLE** 子句。例如，使用下列语句改变前一节中定义的索引的存储选项：

```
ALTER TABLE emp
    ENABLE PRINIARY KEY US ING INDEX
    PCTFREE 5;
```

### 14.4 监视索引的空间使用

如果索引中的关键字值频繁地插入、更新和删除，随着时间的过去，索引可能或者不能有效地使用它获得的空间。可以每隔一定时间，通过分析索引的结构然后查询 **INDEXSTATS** 视图的方法，监视索引的空间使用效率：

```
SELECT pct_used FROM sys.index_stats WHERE name = 'indexname'
```

索引的空间使用比率将根据索引关键字插入、更新或删除的频繁程度变化。按照下列操作顺序执行几次操作，可以研究出索引的平均空间使用效率的规律：

■ 分析统计资料

- 使索引生效
- 检验 PCT\_USED 参数
- 删除并重新创建(或者合并)该索引

当发现索引的空间使用率低于它的平均值时，用户可以通过删除索引并重建或者合并它的方法，压缩索引的空间。关于分析索引结构的信息，参见第 19 章的“分析表、索引和簇”部分。

## 14.5 删除索引

要删除一个索引，该索引必须包含在用户模式中，或者用户必须有 **DROP ANY INDEX** 的系统授权。

用户可能删除索引的原因如下：

- 索引不再需要。
- 对于针对相关表使用的查询，索引不能提供预期的性能改善。（例如，表可能很小，或者表中有许多数据行但是索引项极少。）
- 应用程序没有使用索引查询数据。
- 索引变为无效的，在重建之前必须被删除。
- 索引变得过于破碎，在重建之前必须被删除。

当用户删除索引时，该索引所有的数据段区域都被归还给包含它们的表空间，并且可为表空间中其它的对象所使用。用户如何删除一个索引取决于用户是利用 **CREATE INDEX** 语句明确地创建索引，还是通过在表上定义一个关键字约束隐含地创建索引。如果用户利用 **CREATE INDEX** 语句明确地创建索引，那么用户可以利用 **DROP INDEX** 语句删除该索引。

使用下列语句删除索引 EMP\_ENAME：

```
DROP INDEX emp_ename;
```

用户不能单独删除与有效的 **UNIQUE** 关键字或者 **PRIMARY KEY** 约束相关的索引。要删除与约束相关的索引，用户必须禁用或者删除约束本身。关于删除与约束相关索引的更多信息，参见第 19 章的“管理完整性约束”一节。

**注意** 如果表被删除，所有相关的索引都被自动地删除。

# 15 管理分区数据表和索引

本章从各方面描述如何管理分区数据表和索引，包含以下内容：

- 什么是分区表和索引
- 分区方法
- 创建分区
- 维护分区
- 分区表和索引的实例

## 15.1 什么是分区的表和索引

现代企业运行的数据库经常会包含超过几千亿字节的数据，很多时候数据量会达到几兆兆字节。这些企业面临着支持和维护这种大型数据库（VLDB）的强烈需求，因而必然会想方设法来满足这些需求。

满足 VLDB 需求的办法之一是创建和使用分区表和索引（partitioned tables and indexes）。分区表允许将用户的数据分解为一些小的更容易管理的数据块，这些块就叫分区（partition）或子分区（subpartition）。也可以采用相似的方式来分解索引。每个分区可以被单独管理，而且可以独立于其它分区进行操作，从而提供了可以更好地协调效率和性能的结构。

如果在使用并行运行，则分区可以提供另一种并行方式。通过将不同的并行服务器分配给表或索引的不同分区，可以并行地执行对分区表和索引的操作。

表或索引的分区和子分区都共享相同的逻辑属性。例如，同一个表的所有分区（或子分区）共享相同的列和约束定义；索引的索引分区（或子分区）共享相同的索引选项。但是，他们可以有不同的物理属性（如 TABLESPACE）。

虽然不是必须保证每个表或索引分区（或子分区）位于不同的表空间（tablespace），但是这样做是有好处的。将分区保存在不同的表空间能够：

- 减少在多个分区中破坏数据的可能性
- 独立地备份和恢复各个分区
- 控制分区到磁盘驱动器的映射（对平衡输入/输出加载很重要）
- 提高易管理性、实用性和使用性能。

对于已存在的应用程序和基于分区表的标准 DML 语句，分区操作是透明的。可以在 DML 中使用分区扩展表或索引来利用分区操作对应用程序进行编程。

## 15.2 分区方法

Oracle 提供了三种分区的方法：

- 范围分区

- 散列(hash)分区

- 复合分区

和表一样，索引也可以被分区。全局索引只能按范围分区，但它可以被定义在任何类型的分区或非分区表上。与局部索引相比，它通常有更多的维护要求。

为了反应基表的结构而构造了局部索引。它被基表均匀分区，即意味着在基表的相同列上对索引分区、创建相同数量的分区或子分区，并给它们与基表的分区相对应的相同分区边界。对于局部索引，在分区被维护活动影响时自动维护索引分区。这保证了索引一直被基表均分。

在下面将对 Oracle 的分区方法作一介绍：

- 使用范围分区法

- 使用散列(hash)分区法

- 使用复合分区法

### 15.2.1 使用范围分区法

使用范围分区法可以将行映射到基于列值范围的分区。在处理具有在逻辑范围内可以被分布的数据时这种分区类型非常有用。例如，一年中的月份。当数据在范围内均匀分布时，这种分区方法的性能最好。如果因为不均匀的分布而导致按范围分区时分区大小发生巨大变化，那么应该考虑使用其它的分区方法。

当创建范围分区时，必须指定：

- 分区方法：范围

- 分区列

- 分区说明中指定分区边界

下面的例子中创建了一个具有四个分区的表，每个分区是一个季度的销售额。列 SALE\_YEAR、SALE\_MONTH 和 SALE\_DAY 为分区列，它们的值建立了特定的行分区键（partitioning key）。VALUES LESS THAN 子句确定分区边界：分区键值小于该子句所指定值的行将被保存在这个分区中。各个分区被命名为（SALES\_Q1、SALES\_Q2...），而且每个分区位于不同的表空间（TSA、TSB...）中。

```
CREATE TABLE sales
( invoice_no NUMBER,
  sale_year INT NOT NULL,
  sale_month INT NOT NULL,
  sale_day INT NOT NULL )
PARTITION BY RANGE (sale_year, sale_month, sale_day)
( PARTITION sales_q1 VALUES LESS THAN (1999, 04, 01)
  TABLESPACE tsa,
  PARTITION sales_q2 VALUES LESS THAN (1999, 07, 01)
  TABLESPACE tsb,
  PARTITION sales_q3 VALUES LESS THAN (1999, 10, 01)
  TABLESPACE tsc,
  PARTITION sales_q4 VALUES LESS THAN (2000, 01, 01)
```

TABLESPACE tsd);

SALE\_YEAR=1999、SALE\_MONTH=8 和 SALE\_DAY=1 的行的分区键为 (1999,8,1) 将被保存在分区 SALES\_Q3 中。

范围分区表的每个分区被保存在不同的段中。

创建了范围分区表以后，可以使用"ALTER TABLE"语句来增加另外的分区、合并分区、加载数据（交换分区）或执行其它的维护操作。在表 15-1 中列出了这些维护操作。15.4 节“维护分区”中说明和展示了这些操作的例子。

可以在范围分区表上创建非分区全局索引、范围分区全局索引和局部索引，并像表 15-2 中说明的那样对它们执行维护操作。

**注意** 如果企业有或将要有使用不同字符集的数据库，那么就在对字符列分区时使用警告。因为在所有字符集中字符的排列次序并不相等。

### 15.2.2 使用散列（hash）分区法

如果数据不容易进行范围分区，而出于性能原因又要进行分区时建议使用散列分区。散列分区提供了一种通过指定分区编号来均匀地分布数据的方法。基于分区键的散列值将行映射到分区上。创建和使用散列分区给用户提供了高度灵活的放置数据的方法，因为通过在 I/O 设备上散布这些平均大小的分区，用户可以影响数据库的可行性和性能。

为了创建散列分区，需要指定以下信息：

- 分区方法：散列
- 分区列
- 分区编号或各个分区的说明

下面的例子创建了一个散列分区的表。分区列为 ID，创建了四个分区并分配了由系统生成的分区名，它们被放置在四个被命名的表空间（GEAR1,GEAR2...）中。

```
CREATE TABLE scubagear
(id NUMBER,
 name VARCHAR2 (60))
PARTITION BY HASH (id)
PARTITIONS 4
STORE IN (gear1, gear2, gear3, gear4);
```

散列分区表的每个分区被保存在不同的节中。

有关创建散列分区表的更详细信息包含在 15.3.2 节“创建散列分区”中。“ALTER TABLE”语句可以用来执行对散列分区表的维护操作。大多数范围分区的维护操作适用于散列分区表，只有以下例外：

- 拆分分区
- 撤销分区
- 合并分区

另外，有两个特别用于由散列分区方法创建的维护操作：

- 合并分区：按照既定的合并函数将分区内容重新分布到一个或多个保留分区中来撤销一个散列分区。
- 增加分区：与增加范围分区的子句语法不同。

在表 15-1 中列出了散列分区表的维护操作，并在 15.4 节的“维护分区”中进行了讨论。可以在散列分区表上创建非分区全局索引、范围分区全局索引和局部索引。

### 15.2.3 使用复合分区法

复合分区的分区数据使用范围分区法，而在每个分区或子分区内使用散列分区法。复合分区对历史数据和带数据都很理想，并提供范围分区和数据放置的改进的管理性能和散列分区的并行化优点。

创建复合分区时，需要指定以下数据：

- 分区方法：范围
- 分区列
- 指定分区边界的分区说明
- 子分区方法：散列
- 子分区列
- 每个分区的子分区数量或子分区说明

下面的语句创建了一个复合分区表。在这个实例中，创建了三个范围分区，每个分区又包含八个子分区。子分区没有名字所以赋予它们系统生成的名字，“STORE IN”子句将它们分布在四个指定的表空间中（TS1、...TS4）。

```
CREATE TABLE scubagear (equipno NUMBER, equipname VARCHAR(32), price NUMBER)
PARTITION BY RANGE (equipno) SUBPARTITION BY HASH(equipname)
SUBPARTITIONS 8 STORE IN (ts1, ts3, ts5, ts7)
(PARTITION p1 VALUES LESS THAN (1000),
PARTITION p2 VALUES LESS THAN (2000),
PARTITION p3 VALUES LESS THAN (MAXVALUE));
```

复合分区表的每个子分区被保存在它自己的段中。和它们的数据保存在子分区段中一样，复合分区表的分区是逻辑结构。跟分区一样，这些子分区共享相同的逻辑属性。与范围分区表中的范围分区不同，虽然不要求子分区和主分区在同一个表空间，但子分区不能具有与它们的主分区不同的物理属性。

“ALTER TABLE”语句可以用来执行对复合分区表的维护操作。用户可以对复合分区执行所有范围分区的维护操作，可以对散列分区表的散列分区和子分区执行相同的维护操作。在表 15-1 表中列出了复合分区表的维护操作，并在 15.4 节“维护分区”中进行了讨论。

可以在复合分区表上创建非分区的全局索引、范围分区的全局索引和局部索引。

## 15.3 创建分区

本节介绍为不同类型的分区表和索引创建分区的细节和实例。创建分区表或索引与创建规则表或索引非常相似，但要包含一个分区子句。包含的分区子句和分子句依赖于所要实现的分区的类型。

当创建（或改变）分区表时，可以指定一个行移动子句 **ENABLE ROW MOVEMENT** 或 **DISABLE ROW MOVEMENT**。这个子句在行的关键字被更新时允许或停止行移动到新

的分区。缺省状态为不允许移动。

可以分区包含带 LOB 的表、objects(对象)、varrays 和 nested tables（嵌套表）的表列。

### 15.3.1 创建范围分区

“CREATE TABLE”语句的“PARTITION BY RANGE”子句标识将被按范围分区的表。PARTITION 子句指定各个分区的范围，可选分子句 PARTITION 可以规定特定分区段的物理和其它属性。如果不超越分区层，分区将继承它们的基表的属性。

在这个例子中，将早先介绍过的范围分区表的例子进一步复杂化了。在表一级指定了存储参数和 LOGGING 属性。这些将替换表本身从表空间级继承来的缺省属性，同时这些属性将被范围分区继承。因为第一季度有一些生意，SALES\_Q1 分区的存储属性变小了。在关键字值发生更新即将行定位于不同的分区时，“ENABLE ROW MOVEMENT”子句允许将行移动到一个新的分区。

```
CREATE TABLE sales
( invoice_no NUMBER,
  sale_year INT NOT NULL,
  sale_month INT NOT NULL,
  sale_day INT NOT NULL )
STORAGE (INITIAL 100K NEXT 50K) LOGGING
PARTITION BY RANGE (sale_year, sale_month, sale_day)
( PARTITION sales_q1 VALUES LESS THAN (1999, 04, 01)
  TABLESPACE tsa STORAGE (INITIAL 20K, NEXT 10K) ,
  PARTITION sales_q2 VALUES LESS THAN (1999, 07, 01)
  TABLESPACE tsb ,
  PARTITION sales_q3 VALUES LESS THAN (1999, 10, 01)
  TABLESPACE tsc ,
  PARTITION sales_q4 VALUES LESS THAN (2000, 01, 01)
  TABLESPACE tsd )
ENABLE ROW MOVEMENT ;
```

创建范围分区全局索引的规则与创建范围分区表相似。下面是在上表中通过 SALES\_MONTH 创建范围分区全局索引的实例。每个索引分区被命名并保存在索引的缺省表空间中。

```
CREATE INDEX month_ix ON sales(sales_month)
GLOBAL PARTITION BY RANGE(sales_month)
( PARTITION pm1_ix VALUES LESS THAN(2)
  PARTITION pm2_ix VALUES LESS THAN(3)
  PARTITION pm3_ix VALUES LESS THAN(4)
  PARTITION pm4_ix VALUES LESS THAN(5)
  PARTITION pm5_ix VALUES LESS THAN(6)
  PARTITION pm6_ix VALUES LESS THAN(7)
  PARTITION pm7_ix VALUES LESS THAN(8)
  PARTITION pm8_ix VALUES LESS THAN(9)
  PARTITION pm9_ix VALUES LESS THAN(10)
  PARTITION pm10_ix VALUES LESS THAN(11)
```



```
(PARTITION pm11_ix VALUES LESS THAN(12)
(PARTITION pm12_ix VALUES LESS THAN(MAXVALUE));
```

可以对由索引组成的表和它们的二级索引进行分区，但只能使用范围分区法。下例创建了范围分区的索引表 **SALES**。**INCLUDING** 子句规定在 **WEEK\_NO** 之后的所有列保存在溢出段中；每个分区有一个溢出段，都保存在同一个表空间（**OVERFLOW\_HERE**）中。利用选项，可以将 **OVERFLOW TABLESPACE** 指定到不同的分区层次，这样部分或全部的溢出段将具有不同的 **TABLESPACE** 属性。

```
CREATE TABLE sales ( acct_no NUMBER(5),
                    acct_name CHAR(30),
                    amount_of_sales NUMBER(6),
                    week_no INTEGER,
                    sale_details VARCHAR2(1000),
                    PRIMARY KEY (acct_no, acct_name, week_no))
ORGANIZATION INDEX INCLUDING week_no
OVERFLOW TABLESPACE overflow_here
PARTITION BY RANGE (week_no)
(PARTITION VALUES LESS THAN (5) TABLESPACE ts1,
PARTITION VALUES LESS THAN (9) TABLESPACE ts2,
...
PARTITION VALUES LESS THAN (MAXVALUE) TABLESPACE ts13 );
```

### 15.3.2 创建散列分区

**CREATE TABLE** 语句的 **PARTITION BY HASH** 子句指定该表将被散列分区。**PARTITION** 子句可以用来指定将要创建的分区数量，同时选择性地指定保存这些分区的表空间。或者，可以使用 **PARTITION** 子句来命名各个分区和它们的表空间。

唯一可以指定的散列分区的属性是 **TABLESPACE**。表的所有散列分区共享从表一级继承的相同的段属性（**TABLESPACE** 例外）。

下面的实例介绍了两种创建名为 **DEPT** 的散列分区表的方法。在第一个例子中，指定了分区的个数，分配给它们系统生成的名字而且它们被保存在表的缺省表空间中。

```
CREATE TABLE dept (deptno NUMBER, deptname VARCHAR(32))
PARTITION BY HASH(deptno) PARTITIONS 16;
```

在第二个实例中，指定了各个分区的名字和它们所处的表空间。在表一级显式地定义了每个散列分区（段）的初始大小，所有分区就继承该属性。

```
CREATE TABLE dept (deptno NUMBER, deptname VARCHAR(32))
STORAGE (INITIAL 10k)
PARTITION BY HASH(deptno)
(PARTITION p1 TABLESPACE ts1, PARTITION p2 TABLESPACE ts2,
PARTITION p3 TABLESPACE ts1, PARTITION p4 TABLESPACE ts3);
```

如果创建上表的本地索引，Oracle 将构造该索引从而该索引被基表均匀分区，在对基表执行维护操作时保证该索引被自动维护。下面是基于表 **DEPT** 创建本地索引的实例。

```
CREATE INDEX locd_dept_ix ON dept(deptno) LOCAL
```

用户可以选择性地命名散列分区和将保存本地索引分区的表空间，但是如果用户没有使用该选项，Oracle 将使用相应的基分区作为索引分区名并将索引分区与表分区保存在相同的表空间中。

15.3.3 创建复合分区和子分区

要创建复合分区表，用户需使用 CREATE TABLE 语句的 PARTITION BY RANGE 子句开始。然后，指定与 PARTITION BY HASH 语句遵循相似语法和规则的 SUBPARTITION BY HASH 子句。后面跟各种 PARTITION 和 SUBPARTITION 或 SUBPARTITIONS 子句。

为（范围）分区指定的属性应用于该分区的所有子分区。用户可以为每个（范围）分区指定不同的属性，而且如果分区的子分区所跨的表空间列表不同于其它的分区的表空间列表时，可以在分区一级指定 STORE IN 子句。下例对所有这些内容进行了说明。

```
CREATE TABLE emp (deptno NUMBER, empname VARCHAR(32), grade NUMBER)
PARTITION BY RANGE(deptno) SUBPARTITION BY HASH(empname)
SUBPARTITIONS 8 STORE IN (ts1, ts3, ts5, ts7)
(PARTITION p1 VALUES LESS THAN (1000) PCUFREE 40,
PARTITION p2 VALUES LESS THAN (2000)
STORE IN (ts2, ts4, ts6, ts8),
PARTITION p3 VALUES LESS THAN (MAXVALUE)
(SUBPARTITION p3_s1 TABLESPACE ts4,
SUBPARTITION p3_s2 TABLESPACE ts5));
```

下面的语句在索引段将跨越表空间 TS7、TS8 和 TS9 的表 EMP 上创建了一个局部索引。

```
CREATE INDEX emp_ix ON emp (deptno)
LOCAL STORE IN (ts7, ts8, ts9);
```

这个局部索引将按照基表被均匀分区如下：

- 它将和基表一样由多个分区组成。
- 与基表的分区相对应，每个索引分区将由多个子分区组成。
- 所给基表子分区中的行的索引条目将保存在相应的索引的子分区中。

15.4 维护分区

本节说明如何对表和索引执行分区和子分区维护操作。

表 15-1 列出了可以用于表分区（或子分区）的维护操作，并针对每种分区类型，列出了执行该维护操作的特定 ALTER TABLE 语句。

表 15-1 针对表分区的 ALTER TABLE 维护操作

| 维护操作 | 范围            | 散列        | 复合                                  |
|------|---------------|-----------|-------------------------------------|
| 加入分区 | ADD PARTITION | ADD       | ADD PARTITION                       |
|      |               | PARTITION | MODIFY PARTITION...ADD SUBPARTITION |
| 合并分区 | n/a           | COALESCE  | MODIFY PARTITION...COALESCE         |
|      |               | PARTITION | SUBPARTITION                        |

(续表)

| 维护操作             | 范围                              | 散列                              | 复合  |
|------------------|---------------------------------|---------------------------------|---|
| 删除分区             | DROP<br>PARTITION               | n/a                             | DROP PARTITION  |
| 互换分区             | EXCHANGE<br>PARTITION           | EXCHANGE<br>PARTITION           | EXCHANGE PARTITION<br>EXCHANGE SUBPARTITION                             |
| 合并 (merge)<br>分区 | MERGE<br>PARTITIONS             | n/a                             | MERGE PARTITIONS  |
| 修改分区的缺<br>省属性    | MODIFY<br>DEFAULT<br>ATTRIBUTES | MODIFY<br>DEFAULT<br>ATTRIBUTES | MODIFY DEFAULT ATTRIBUTES<br>MODIFY<br>DEFAULT ATTRIBUTES FOR PARTITION |
| 修改分区的实<br>际属性    | MODIFY<br>PARTITION             | MODIFY<br>PARTITION             | MODIFY PARTITION<br>MODIFY<br>SUBPARTITION                              |
| 移动分区             | MOVE<br>PARTITION               | MOVE<br>PARTITION               | MOVE SUBPARTITION   |
| 重命名分区            | RENAME<br>PARTITION             | RENAME<br>PARTITION             | RENAME PARTITION<br>RENAME<br>SUBPARTITION                              |
| 拆分分区             | SPLIT<br>PARTITION              | n/a                             | SPLIT PARTITION   |
| 截短分区             | TRUNCATE<br>PARTITION           | TRUNCATE<br>PARTITION           | TRUNCATE PARTITION<br>TRUNCATE SUBPARTITION                             |

表 15-2 列出了可以针对索引分区执行的维护操作，并指出了可以使用该维护操作的索引（全局或局部）类型。全局索引不反应基表的结构，而且只能使用范围分区法进行分区。按范围分区的索引可以共享一些可以在范围分区表上执行的分区维护操作，但不能共享全部这些操作。

因为维护活动影响表分区和子分区时，局部索引分区被自动维护，因此对局部索引的分区维护不是很必要，也几乎没有选项。

表 15-2 针对索引分区的 ALTER INDEX 维护操作

| 维护操作            | 索引类型 | 索引分区类型                       |                                 |  |
|-----------------|------|------------------------------|---------------------------------|--|
|                 |      | 范围                           | 散列                              | 复合   |
| 删除索引分区          | 全局   | DROP PARTITION               |                                 |  |
|                 | 本地   | n/a                          | n/a                             | n/a  |
| 修改索引分区的<br>缺省属性 | 全局   | MODIFY DEFAULT<br>ATTRIBUTES |                                 |  |
|                 | 本地   | MODIFY DEFAULT<br>ATTRIBUTES | MODIFY<br>DEFAULT<br>ATTRIBUTES | MODIFY DEFAULT<br>ATTRIBUTES<br>MODIFY DEFAULT<br>ATTRIBUTES FOR |

(续表)

| 维护操作            | 索引类型 | 索引分区类型               |                      |                         |
|-----------------|------|----------------------|----------------------|-------------------------|
|                 |      | 范围                   | 散列                   | 复合                      |
|                 |      |                      |                      | PARTITION               |
| 修改索引分区的<br>实际属性 | 全局   | MODIFY<br>PARTITION  |                      |                         |
|                 | 本地   | MODIFY<br>PARTITION  | MODIFY<br>PARTITION  | MODIFY<br>SUBPARTITION  |
| 重建索引分区          | 全局   | REBUILD<br>PARTITION |                      |                         |
|                 | 本地   |                      | REBUILD<br>PARTITION | REBUILD<br>SUBPARTITION |
| 重命名索引分区         | 全局   | RENAME<br>PARTITION  |                      |                         |
|                 | 本地   | RENAME<br>PARTITION  | RENAME<br>PARTITION  | RENAME<br>SUBPARTITION  |
| 拆分索引分区          | 全局   | SPLIT PARTITION      |                      |                         |
| 本地              | n/a  | n/a                  | n/a                  |                         |

另外，可以使用 SQL\*Loader, IMPORT 和 EXPORT 工具来加载或卸载保存在分区表中的数据。这些工具都是分区和子分区工具包。

15.4.1 增加分区

本节说明如何在分区表中增加新的分区，并解释为什么不能把分区加到全局分区或局部索引。

给范围分区表增加分区

用户可以使用 ALTER TABLE ... ADD PARTITION 语句将新分区加到表的“高”端（已存在的最后一个分区的后面）。如果要分区加到表的开始或中间，则使用 SPLIT PARTITION 子句。

例如，DBA 有一个表 SALES，该表包含当月的数据和前 12 个月的数据。1999 年一月一日，DBA 增加了一月的分区，该分区保存在表空间 TSX 中。

```
ALTER TABLE sales
  ADD PARTITION jan96 VALUES LESS THAN ( '01-FEB-1999')
  TABLESPACE tsx;
```

给散列分区表增加分区

当用户给散列分区表增加分区时，Oracle 根据行号来定位到新的分区，该行号是按照散列函数从表的其它分区重新散列而来的。

下面的语句说明了将散列分区加到表 SCUBAGEAR 中的两种方法。使用第一个语句将增加由系统生成分区名的新的散列分区，而且它将被定位在表的缺省表空间中。第二个

语句也增加新的散列分区，但该分区被显式地命名为 **P\_NAMED**，该分区被创建在表空间 **GEAR5**。

```
ALTER TABLE scubagear ADD PARTITION;
```

```
ALTER TABLE scubagear  
ADD PARTITION p_named TABLESPACE gear5;
```

如果该表有局部索引，则将所有局部索引分区标记为 **UNUSABLE** 并必须重新建立。任何全局索引或全局索引的所有分区也将被标记为 **UNUSABLE**。

#### 给复合分区表增加分区

分区有可能同时加到范围分区级和散列子分区级。

**增加分区** 可以如前面“给范围分区表增加分区”中所说明的那样，用与其相似的方式增加新的范围分区，但是，可以指定允许用户增加一定数量的子分区 **SUBPARTITIONS** 子句或为特定子分区命名的 **SUBPARTITION** 子句。如果没有指定 **SUBPARTITIONS** 或 **SUBPARTITION** 子句，分区将继承表一级的有关子分区的缺省值。

下面的例子为表 **SALES** 增加了范围分区 **Q1\_2000**，该分区保存 2000 年第一季度的数据。在表空间 **TBS5** 中保存了 8 个子分区。

```
ALTER TABLE sales ADD PARTITION q1_2000  
VALUES LESS THAN (2000, 04, 01)  
SUBPARTITIONS 8 STORE IN tbs5;
```

**增加子分区** 可以使用 **ALTER TABLE** 语句的 **MODIFY PARTITION...ADD SUBPARTITION** 子句在复合分区表中增加散列子分区。新增加的子分区的数据是按照散列函数从相同分区中的其它子分区散列而来的。同时，必须重新建立与新增子分区和重新散列的子分区相对应的全局和局部索引分区。

在下面的实例中，保存在表空间 **US1** 中的新的散列子分区 **US\_LOC5** 被增加到表 **DIVING** 的范围分区 **LOCATIONS\_US** 中。

```
ALTER TABLE diving MODIFY PARTITION locations_us  
ADD SUBPARTITION us_locs5 TABLESPACE us1;
```

#### 增加索引分区

不可以显式地给局部索引增加分区。只有当用户给基表增加分区时，新的分区会加到局部索引。特别地，如果表带有局部索引，在使用 **ALTER TABLE** 语句增加分区时，也会为局部索引增加匹配的分区的。因为 **Oracle** 为新的索引分区分配名字和缺省的物理存储属性，用户可能在 **ADD** 操作完成后要进行重命名或修改它们。

用户不能给该全局索引增加分区，因为最高分区通常有值为 **MAXVALUE** 的分区边界。如果要增加新的最高分区，就使用 **ALTER INDEX...SPLIT PARTITION** 语句。

### 15.4.2 合并分区

可以通过合并分区来减少散列分区表中分区的个数，或减少复合分区表中子分区的个数。合并散列分区时，按照散列函数将它的内容分布到一个或多个保留分区中。特定的需要

合并的分区是由 RDBMS 选择的，而且在它的内容被分布之后再将它删去。

与所选择的分区相对应的局部索引分区也将被删除。与一个或多个吸收分区相应的局部索引分区被标记为 UNUSABLE，并被重新建立。所有全局索引被标记为 UNUSABLE。

#### 在散列分区表中合并分区

ALTER TABLE...COALESCE PARTITION 语句用于合并散列分区表中的分区。下面的语句通过合并分区将分区的个数减少了一个。

```
ALTER TABLE ouu1
    COALESCE PARTITION;
```

#### 在复合分区表中合并子分区

下面的语句将分区 US\_LOCATIONS 中的子分区的内容重新分布到一个或多个保留的相同分区的子分区中（由散列函数决定）。基本上，该操作是原先讨论过的 MODIFY PARTITION...ADD SUBPARTITION 子句的反向操作。

```
ALTER TABLE diving MODIFY PARTITION us_locations
    COALESCE SUBPARTITION;
```

### 15.4.3 删除分区

可以从范围或复合分区表中删除分区。针对散列分区表或复合分区表的散列子分区，用户必须执行合并操作来代替删除。

#### 删除表分区

用户可以使用 ALTER TABLE...DROP PARTITION 语句来删除范围或复合分区表中的表分区，如果要保留该分区中的数据，用户可以将数据合并到相邻的分区中。

如果表定义了局部索引，该语句也删除匹配的局部索引的分区或子分区。表的全局非分区索引将被标记为 UNUSABLE，全局分区索引的所有分区将被标记为 UNUSABLE，除非被删除的分区或它的子分区是空的。

**注意** 不能删除表的唯一分区。只能删除整个表。

下面的几节列出了删除表分区的几种情况。

**从包含数据和全局索引的表中删除分区** 如果分区包含数据，而且在表上定义了一个或多个全局索引，那么使用下面的方法之一可以删除表分区。

1. 在执行 ALTER TABLE...DROP PARTITION 语句时，保留全局索引。然后，用户必须重建全局索引（不管是否分区），因为索引（或索引分区）已被标记为不可使用。下面的语句提供了从表 SALES 中删除分区 DEC98 的实例，然后又重建了它的全局非分区索引。

```
ALTER TABLE sales DROP PARTITION dec98;
ALTER INDEX sales_area_ix REBUILD;
```

如果索引 SALES\_AREA\_IX 是范围分区全局索引，那么就需要重建索引的所有分区。另外，不可能在一个语句中重建索引的所有分区；用户必须为索引的每个分区写单独的 REBUILD 语句。下面的语句重建了索引分区 JAN99\_IX、FEB99\_IX、...、DEC99\_IX。

```
ALTER INDEX sales_area_ix REBUILD PARTITION jan99_ix;
```

```
ALTER INDEX sales_area_ix REBUILD PARTITION feb99_ix;
ALTER INDEX sales_area_ix REBUILD PARTITION mar99_ix;
...
ALTER INDEX sales_area_ix REBUILD PARTITION nov99_ix;
```

这种方法特别适合于包含有大量数据的分区将被删除的大表。

2. 执行 **ALTER TABLE ...DROP PARTITION** 语句之前执行 **DELETE** 语句删除分区的所有行。**DELETE** 语句更新全局索引，也激发触发器并生成重做和撤销日志。

例如，如果要删除分区界限为 10000 的第一个分区，用户可以执行下面的语句：

```
DELETE FROM sales WHERE TRANSID <10000;
ALTER TABLE sales DROP PARTITION dec98;
```

这种方法适用于小表，或将要删除的分区包含少量数据的大表。

**删除包含数据和参考统一性约束的分区** 如果分区包含数据而且表拥有参考统一性约束，可以选择下面的方法之一来删除表分区。该表只有局部索引，所以不必重建任何索引。

1. 停用统一性约束，执行 **ALTER TABLE ...DROP PARTITION** 语句，然后激活统一性约束：

```
ALTER TABLE sales
    DISABLE CONSTRAINT dname_sales1;
ALTER TABLE sales DROP PARTITION dec98;
ALTER TABLE sales
    ENABLE CONSTRAINT dname_sales1;
```

这种方法适合于将要删除的分区包含表的大部分数据的大型数据表。

2. 在执行 **ALTER TABLE...DROP PARTITION** 语句之前先执行 **DELETE** 语句删除分区中的索引行。**DELETE** 语句强行参考统一性约束，也激发触发器并生成重做和撤销日志。

```
DELETE FROM sales WHERE TRANSID < 10000;
ALTER TABLE sales DROP PARTITION dec94;
```

这种方法较适合于小表或将要删除的分区包含表的少量数据的大表。

### 删除索引分区

用户不能显式地删除局部索引的分区。只有当删除基表的分区时，局部索引的分区才会被删除。如果全局索引分区是空的，用户可以通过执行 **ALTER INDEX...DROP PARTITION** 语句来显式地删除该分区。但是，如果全局索引分区包含有数据，删除分区将引起下一个高分区被标记为 **UNUSABLE**。例如，用户要删除索引分区 **P1**，而 **P2** 是下一个最高分区，这时用户必须执行下面的语句：

```
ALTER INDEX npr DROP PARTITION p1;
ALTER INDEX npr REBUILD PARTITION p2;
```

注意：用户不能删除全局索引的最高分区。

### 15.4.4 交换分区

通过交换数据段，用户可以将分区（或子分区）转换为非分区表，也可以将非分区表转

换为分区表的分区（或子分区）。用户也可以将散列分区表转换为复合分区表的分区或将复合分区表的分区转换为散列分区表。

当用户要将非分区表转换到分区表的一个分区时，交换表分区就特别有用。例如，用户可能有要转换为分区表的分区视图。

交换分区也方便了在使用可传输表空间时的高速数据加载。

在交换分区时，日志属性被保留，但用户也可以选择性地指定局部索引是否也被交换和针对适当的映射行是否有效。

#### 交换散列或范围分区

为了与一个非分区表交换范围或散列分区表的分区，反之亦然，用户可以使用 **ALTER TABLE ...EXCHANGE PARTITION** 语句。下面是将分区转换为非分区表的实例，在本例中，表 **STOCKS** 可以是范围或散列分区表。

```
ALTER TABLE stocks
    EXCHANGE PARTITION p3 WITH stock_table_3;
```

#### 与复合分区交换散列分区表

用户可以再次使用 **ALTER TABLE ...EXCHANGE PARTITION** 语句来执行该操作，但是，这次是将整个散列分区表和它的所有分区以及复合分区表的范围分区和它的所有散列子分区进行交换。说明如下：

首先，创建散列分区表：

```
CREATE TABLE t1 (i NUMBER, j NUMBER)
    PARTITION BY HASH(i)
        (PARTITION p1, PARTITION p2);
```

填充该表，然后创建复合分区表如下：

```
CREATE TABLE t2 (i NUMBER, j NUMBER)
    PARTITION BY RANGE(j)
        SUBPARTITION BY HASH(i)
            (PARTITION p1 VALUES LESS THAN (10)
                SUBPARTITION t2_p1s1
                SUBPARTITION t2_p1s2,
            PARTITION p2 VALUES LESS THAN (20)
                SUBPARTITION t2_p2s1
                SUBPARTITION t2_p2s2));
```

表 T1 的分区键与表 T2 的子分区键相同，这一点特别重要。

为了将 T1 中的数据移到 T2，并使这些行有效，需使用下面的语句：

```
ALTER TABLE t1 EXCHANGE PARTITION p1 WITH TABLE t2
    WITH VALIDATION;
```

#### 交换复合分区表的子分区

可以使用 **ALTER TABLE...EXCHANGE SUBPARTITION** 语句将复合分区表的散列子分区转换到一个非分区表，或进行反向转换。下面的实例将表 **SALES** 的子分区 **Q3\_1999\_S1**



转换为非分区表 Q3\_1999。局部索引分区被转换为 Q3\_1999 表的相应索引。

```
ALTER TABLE sales EXCHANGE SUBPARTITIONS q3_1999_s1
WITH TABLE q3_1999 INCLUDING INDEXES;
```

#### 15.4.5 合并分区

用户可以使用 **ALTER TABLE ...MERGE PARTITIONS** 语句将两个相邻的范围分区合并为一个分区。新产生的分区继承了两个被合并分区中的较高的上边界。原先的两个分区与相应的局部索引一起被删除。如果被合并的分区不为空，则表的非分区全局索引被标记为 **UNUSABLE**，全局分区索引的所有分区被标记为 **UNUSABLE**。同时，除非包含的分区或子分区为空，Oracle 将新分区的所有局部索引分区或子分区标记为 **UNUSABLE**。

用户不能针对散列分区表或复合分区表的散列子分区执行该语句。

用户可能需要通过合并分区来保证在线的历史数据保留在较大的分区中。例如，用户可以将日分区连通老的分区数据一起累积到周分区，周分区又可以累积到月分区等等。

#### 合并范围分区

下面的脚本创建了合并范围分区的实例。

首先，创建分区表和局部索引。

```
-- Create a Table with four partitions each on its own tablespace
--Partitioned by range on the data column.
--
CREATE TABLE four_seasons
(
    one DATE,
    two VARCHAR2(60),
    three NUMBER
)
PARTITION BY RANGE(one)
(
    PARTITION quarter_one
        VALUES LESS THAN (TO_DATE('01-aug-1998','dd-mon-yyyy'))
        TABLESPACE quarter_one,
    PARTITION quarter_two
        VALUES LESS THAN (TO_DATE('01-sep-1998','dd-mon-yyyy'))
        TABLESPACE quarter_two,
    PARTITION quarter_three
        VALUES LESS THAN (TO_DATE('01-oct-1998','dd-mon-yyyy'))
        TABLESPACE quarter_three,
    PARTITION quarter_four
        VALUES LESS THAN (TO_DATE('01-nov-1998','dd-mon-yyyy'))
        TABLESPACE quarter_four
)
/
--
```

```
-- Create local PREFIXED index on Four_Seasons
-- Prefixed because the leftmost column of the index match the
-- Partition key
--
CREATE INDEX i_four_seasons_1 ON four_seasons(one,two)
LOCAL (
PARTITION i_quarter_one TABLESPACE i_quarter_one ,
PARTITION i_quarter_two TABLESPACE i_quarter_two ,
PARTITION i_quarter_three TABLESPACE i_quarter_three ,
PARTITION i_quarter_four TABLESPACE i_quarter_four
)
/
```

第二步，合并分区

```
--
-- Merge the first two partitions
--
ALTER TABLE four_seasons
MERGE PARTITIONS quarter_one, quarter_two INTO PARTITION quarter_two
/
```

然后，重建所影响分区的局部索引。

```
-- Rebuild index for quarter_two, which has been marked unusable
-- because it has not had all of the data from Q1 added to it.
-- Rebuilding the index will correct this.
--
ALTER TABLE four_seasons MODIFY PARTITION
quarter_two REBUILD UNUSABLE LOCAL INDEXES
/
```

### 合并范围复合分区

当用户合并范围复合分区时，子分区将被重新散列到由 **SUBPARTITIONS** 或 **SUBPARTITION** 子句指定的某些子分区中，或者如果不包含该子句，将使用表级缺省值。

```
ALTER TABLE all_seasons
MERGE PARTITIONS quarter_1, quarter_2 INTO PARTITION quarter_2 SUBPARTITIONS 8;
```

### 15.4.6 修改分区缺省属性

用户可以修改表或索引的分区或子分区的缺省属性。当修改缺省属性时，新的属性将只影响将来创建的分区。在创建新表时仍然可以指定属性来替换缺省值。

#### 修改分区的缺省属性

使用 **ALTER TABLE** 的 **MODIFY DEFAULT ATTRIBUTES** 子句可以修改将被范围或散列分区继承的缺省属性。下面的实例修改用于创建新分区的表 **EMP** 的 **PCTFREE** 缺省值。

```
ALTER TABLE emp
MODIFY DEFAULT ATTRIBUTES PCTFREE 25;
```

对于散列分区表，只能修改属性 TABLESPACE。

#### 修改子分区的缺省属性

要修改在复合分区表中创建范围分区的子分区的缺省属性，可以使用 **ALTER TABLE...MODIFY DEFAULT ATTRIBUTES FOR PARTITION**。该语句修改了在复合分区表 EMP 中的分区 P1 上将来所创建的子分区所在的 TABLESPACE。

```
ALTER TABLE emp
    MODIFY DEFAULT ATTRIBUTES FOR PARTITION p1 TABLESPACE ts1;
```

因为除了 TABLESPACE，所有的子分区必须共享相同的属性，所以只有 TABLESPACE 属性可以被修改。

#### 修改索引分区的缺省属性

与表分区相似，用户可以修改将被范围分区全局索引的分区或范围、散列和复合分区表的局部索引分区继承的缺省属性。为此，用户可以使用 **ALTER INDEX...MODIFY DEFAULT ATTRIBUTES** 语句。如果要改变将被复合分区表的子分区继承的缺省属性，可以使用 **ALTER INDEX...MODIFY DEFAULT ATTRIBUTES FOR PARTITION** 语句。

### 15.4.7 修改分区的存在属性

可以修改表或索引的已存在的属性。

用户不能修改 TABLESPACE 属性，可以使用 **ALTER TABLESPACE...MOVE PARTITION/SUBPARTITION** 将分区或子分区移到新的表空间。

#### 修改范围分区的存在属性

可以使用 **ALTER TABLE...MODIFY PARTITION** 语句来修改范围分区的已存在属性。修改段属性（TABLESPACE 除外）或分配和释放扩展区、标记局部索引分区为 UNUSABLE 或重建被标记为 UNUSABLE 的局部索引。

如果是复合分区表的范围分区，请注意以下两点：

- 如果分配或释放扩展区，该操作将对指定分区的每个子分区有效。
- 同样地，对分区的其它属性的改变将导致该分区的所有子分区的属性作相应的改变。分区级缺省属性也将会改变。为了避免改变已存在子分区的属性，请使用 **MODIFY DEFAULT ATTRIBUTE** 语句的 **FOR PARTITION** 子句。

下面是一些修改分区真实属性的实例。

该实例修改表 SALES 的范围分区 SALES\_Q1 的存储属性 MAXEXTENTS：

```
ALTER TABLE sales MODIFY PARTITION sales_Q1
    STORAGE (MAXEXTENTS 10);
```

下面的实例将复合分区表 SCUBAGEAR 的分区 TS1 的局部索引子分区标记为 UNUSABLE：

```
ALTER TABLE scubagear MODIFY PARTITION ts1 UNUSABLE LOCAL INDEXES;
```

### 修改散列分区存在属性

也可以使用 `ALTER TABLE...MODIFY PARTITION` 语句来修改散列分区的属性。然而因为各个散列分区的物理属性必须相同（`TABSPACE` 除外），因此有如下限制：

- 分配新扩展区
- 释放没有使用的扩展区
- 标记局部索引子分区为 `UNUSABLE`
- 重新建立被标记为 `UNUSABLE` 的局部索引子分区

下面的实例重新建立了与表 `DEPT` 的散列分区 `P1` 相关的所有不能使用的局部索引分区。

### 修改子分区存在属性

使用 `ALTER TABLE` 语句的 `MODIFY SUBPARTITION` 子句可以执行与前面列出的针对散列分区操作完全相同的功能，但只能在特定复合分区表的子分区级进行。如：

```
ALTER TABLE emp
  MODIFY SUBPARTITION p3_s1
  REBUILD UNUSABLE LOCAL INDEXES
```

### 修改索引分区存在属性

`ALTER INDEX` 的 `MODIFY PARTITION` 子句允许用户修改索引分区或它的子分区的已存在属性。其规则与表分区非常相似，但与 `ALTER TABLE` 的 `MODIFY PARTITION` 子句不同，这里没有重新建立不可使用索引分区的分句，但有合并索引分区或它的子分区的分句。这种情况下，合并意味着融合索引块使得它们可以被重新使用。

使用 `MODIFY SUBPARTITION` 子句也可以分配或释放局部索引的子分区的存储区，或将它们标记为 `UNUSABLE`。

## 15.4.8 移动分区

可以使用 `ALTER TABLE` 的 `MOVE PARTITION` 子句来：

- 重新聚合数据和减少碎片
- 将分区移到其它表空间
- 修改创建时间属性

通常，使用 `ALTER TABLE/INDEX...MODIFY PARTITION` 语句在一步内可以改变分区的物理存储属性。然而，有一些物理属性如 `TABSPACE`，不能通过 `MODIFY PARTITION` 来修改。这时，可以使用 `MOVE PARTITION` 子句。

当要移动的分包含有数据时，`MOVE PARTITION` 标记每个局部索引中的匹配分区并把所有全局索引分区标记为不可使用。执行 `MOVE PARTITION` 之后，必须重新建立这些索引分区。全局索引也必须重建。

### 移动表分区

可以使用 `MOVE PARTITION` 移动表的分区。例如，`DBA` 要将当前最新的分区移到位于它自己硬盘的表空间上（为了平衡 `I/O`），并希望 `LOG` 该操作。这时 `DBA` 可以执行以下

语句:

```
ALTER TABLE parts MOVE PARTITION depot2  
TABLESPACE ts094 NOLOGGING;
```

即使不指定新的表空间, 该语句也删除分区的旧段并创建新的数据段。

#### 移动子分区

下面的语句说明如何移动表的子分区中的数据。该例中也指定了 **PARALLEL** 子句。

```
ALTER TABLE scuba_gear MOVE SUBPARTITION bcd_types  
TABLESPACE tbs23 PARALLEL (DEGREE 2);
```

#### 移动索引分区

一些子句, 如 **MOVE PARTITION** 和 **DROP PARTITION** 将所有全局索引的分区标记为不可使用。用户可以通过使用 **ALTER INDEX...REBUILD PARTITION** 重建每个分区来重建所有索引。可以并行执行重建操作。

用户也可以删除该索引然后再重建它。

### 15.4.9 重建索引分区

用户可能出于下面的任何一种原因而重建索引分区:

- 恢复空间和提高性能
- 修复由于介质原因而被破坏的索引分区
- 在通过 **IMPORT** 或 **SQL\*Loader** 加载基表分区后重建局部索引分区
- 重建被标记为 **UNUSABLE** 的索引分区

在下面将讨论重建索引分区和子分区的选项。

#### 重建全局索引分区

可以采用以下的两种方法重建全局索引分区:

1. 通过执行 **ALTER INDEX...REBUILD PARTITION** 语句重建每个分区。
2. 删除索引并重建它。

**注意** 第二种方法更有效, 因为只对表扫描一次。

#### 重建局部索引分区

可以使用 **ALTER INDEX** 或 **ALTER TABLE** 重建局部索引:

- 语句 **ALTER INDEX...REBUILD PARTITION/SUBPARTITION** 无条件地重建索引分区或子分区。
- 语句 **ALTER TABLE...MODIFY PARTITION/SUBPARTITION...REBUILD UNUSABLE LOCAL INDEX** 找出所给表分区或子分区的所有不可使用索引并重建它们。该语句只重建被标记为 **UNUSABLE** 的索引分区。

**通过改变索引重建分区** **ALTER INDEX...REBUILD PARTITION** 语句重建索引的一个分区。它不能用于复合分区表。在重建索引的同时, 用户可以将分区移到新的表空间, 也可以修改属性。

针对复合分区表，用户可以使用 `ALTER INDEX...REBUILD SUBPARTITION` 语句重建索引的子分区。可以将子分区移到其它的表空间或指定并行子句。下面的语句重建了表的局部索引的子分区并将索引子分区移到其它表空间。

```
ALTER INDEX scuba
  REBUILD SUBPARTITION bcd_types
  TABLESPACE tbs23 PARALLEL (DEGREE 2);
```

**通过改变表重建索引分区** `ALTER TABLE...MODIFY PARTITION` 语句的 `REBUILD UNUSABLE LOCAL INDEXS` 子句不允许用户为重建的索引分区指定新的属性。下面的实例找出并重建表 `SCUBAGEAR` 的一个不可使用的局部索引分区 `P1`。

```
ALTER TABLE scubagear
  MODIFY PARTITION p1 REBUILD UNUSABLE LOCAL INDEXS;
```

相应的 `ALTER TABLE...MODIFY SUBPARTITION` 子句可以重建不可使用的局部索引子分区。

#### 15.4.10 重命名分区

可以对表和索引的分区和子分区进行重命名。通过重命名可以将系统分配的缺省名修改为有意义的分区名。

##### 重命名表分区

使用 `ALTER TABLE...RENAME PARTITION` 语句重命名范围或散列分区。实例如下：

```
ALTER TABLE scubagear RENAME PARTITION sys_p636 TO tanks
```

##### 重命名表的子分区

同样，可以为表的子分区分配新的名字。这时需使用 `ALTER TABLE...RENAME SUBPARTITION` 语法。

##### 重命名索引分区

索引的分区和子分区可以按相似的方式进行重命名，但需使用 `ALTER INDEX` 语法。

**重命名索引分区** 使用 `ALTER INDEX...RENAME PARTITION` 语句可以重命名索引分区。

**重命名索引子分区** 下面的语句简单地说明了如何对子分区进行重命名，该子分区的名字是在对基表增加分区时由系统自动分配的。

```
ALTER INDEX scuba RENAME SUBPARTITION sys_subp3254 TO bcd_types;
```

#### 15.4.11 拆分分区

`ALTER TABLE` 或 `ALTER INDEX` 语句的 `SPLIT PARTITION` 子句用于将一个分区中的内容重新分布到两个新的分区中。当分区太大而导致备份、恢复或维护操作需要很长时间时，用户可能会对分区进行拆分。也可以利用 `SPLIT PARTITION` 子句重新分布 I/O 加载。

如果要拆分的分区包含数据，`ALTER TABLE...SPLIT PARTITION` 语句将每个局部索引的两个新分区和所有全局索引分区以及全局非分区索引标记为 `UNUSABLE`。必须对这些被

影响的索引或索引分区进行重建。

该子句不能用于散列分区或子分区。

### 拆分范围表分区

通过执行 **ALTER TABLE...SPLIT PARTITION** 语句可以拆分表分区。用户可以选择性地为拆分后新产生两个分区指定新的属性。如果表定义了局部索引，该语句对每个局部索引上的匹配分区也进行拆分。

在下面的实例中，**FEE\_KATY** 是表 **VET\_CATS** 的分区，该表有局部索引 **JAF1**，也有全局索引 **VET**。**VET** 包含两个分区，它们是 **VET\_PARA** 和 **VET\_PARTB**。

为了拆分分区 **FEE\_KATY** 并重建索引分区，**DBA** 执行下面的语句：

```
ALTER TABLE vet_cats SPLIT PARTITION
    fee_katy at (100) INTO (PARTITION
        fee_katy1...,partition fee_katy2...);
ALTER INDEX JAF1 REBUILD PARTITION fee_katy1;
ALTER INDEX JAF1 REBUILD PARTITION fee_katy2;
ALTER INDEX VET REBUILD PARTITION vet_parta;
ALTER INDEX VET REBUILD PARTITION vet_partb;
```

**注意** 如果没有指定新的分区名，Oracle 将按 **SYS\_Pn** 形式分配名字。用户可以检查数据字典对分配给新局部索引分区的名字进行定位。用户可以对它们重命名。所有没有指定的属性都从原分区中继承。

### 拆分范围复合分区

该操作与合并范围复合分区相反。当用户拆分范围复合分区时，新的子分区将被重新散列到由 **SUBPARTITIONS** 或 **SUBPARTITION** 子句规定的某个子分区中，或者如果没有该子句时，将使用表级缺省值。

```
ALTER TABLE all_seasons SPLIT PARTITION quarter_1
    AT (TO_DATE('16-dec-1997','dd-mon-yyyy'))
    INTO (PARTITION q1_1997_1 SUBPARTITIONS 4 STORE IN (ts1,ts3),
        PARTITION q1_1997_2);
```

### 拆分索引分区

用户不能显式地拆分局部索引中的分区。只在拆分基表的分区时，可以对局部索引分区进行拆分。

下面的语句拆分全局索引分区 **QUON1**：

```
ALTER INDEX quon1 SPLIT
    PARTITION canada AT VALUES LESS THAN (100) INTO
    PARTITION canada1 ..., PARTITION canada2 ...);
ALTER INDEX quon1 REBUILD PARTITION canada1;
ALTER INDEX quon1 REBUILD PARTITION canada2;
```

被拆分的索引可以包含索引数据，只有当该分区先前被标记为 **UNUSABLE** 时，需要对其进行重建。

#### 15.4.12 截短分区

当要从表分区中删除所有的行时，使用 `ALTER TABLE...TRUNCATE PARTITION` 语句。截短分区与删除分区相似，除非清空分区中的数据，否则不会进行物理删除。

用户不能截短索引分区；`ALTER TABLE TRUNCATE PARTITION` 语句截短每个局部索引中的匹配分区。如果表上有全局索引（分区或非分区），则它被标记为 `UNUSABLE` 并必须重建。

##### 截短表分区

可以使用 `ALTER TABLE...TRUNCATE PARTITION` 语句通过回收或不回收空间两种方式删除表分区的所有行。如果该表定义了局部索引，`ALTER TABLE ...TRUNCATE PARTITION` 也可以从每个局部索引中截短匹配的分區。

**截短包含数据和全局索引的表分区** 如果分区包含数据和全局索引，可以使用下面的任何一种方法截短表分区：

1. 在执行 `ALTER TABLE TRUNCATE PARTITION` 语句时保持全局索引不变。本例中，表 `SALES` 具有需要重建的全局索引 `SALES_AREA_IX`。

```
ALTER TABLE sales TRUNCATE PARTITION dec94;  
ALTER INDEX sales_area_ix REBUILD;
```

该方法适用于将要截短的分區包含表的大部分数据的大型表。

2. 在执行 `ALTER TABLE...TRUNCATE PARTITION` 语句之前执行 `DELETE` 语句。`DELETE` 语句更新全局索引，同时也激发触发器并生成重做和恢复日志。

这种方法适合于小型表，或所要截短的分區包含小部分表数据的大型表。

**截短包含数据和相关一致性约束的分區** 如果分區包含数据并有相关一致性约束，则选择下面的方法之一截短表分区：

1. 取消一致性约束，执行 `ALTER TABLE...TRUNCATE PARTITION` 语句，然后激活一致性约束：

```
ALTER TABLE sales  
    DISABLE CONSTRAINT dname_sales1;  
ALTER TABLE sales TRUNCATE PARTITION dec94;  
ALTER TABLE sales  
    ENABLE CONSTRAINT dname_sales1;
```

这种方法适合于将要截短的分區包含有表中大部分数据的大表。

2. 在执行 `ALTER TABLE...TRUNCATE PARTITION` 语句之前，先执行 `DELETE` 语句来删除分區中的所有行。`DELETE` 语句实施相关一致性约束，并激活触发器和生成重做和恢复日志。

**注意** 用户可以通过设置 `NOLOGGING` 属性在删除所有行时减少分區的日志数量（使用 `ALTER TABLE...MODIFY PARTITION...NOLOGGING`）。

```
DELETE FROM sales WHERE TRANSID < 10000;  
ALTER TABLE sales TRUNCATE PARTITION dec94;
```

这种方法适合于小表或将要截短的分區包含表的少部分数据的大型表。



## 截短子分区

可以使用 `ALTER TABLE...TRUNCATE SUBPARTITION` 语句删除复合分区表的子分区中的所有行。相应的局部索引子分区也将被截短。

下面的语句说明如何截短表的子分区中的数据。本实例中，删除的行所占用的空间可以被表空间中的其它模式对象再使用：

```
ALTER TABLE diving
TRUNCATE SUBPARTITION us_locations
DROP STORAGE;
```

## 15.5 分区表和索引实例

本节介绍对分区表和索引进行处理的实例。

### 15.5.1 在历史表中移动时间窗口

历史表描述了企业在一个时间间隔内的商业事务。历史表可以有基表，其基表包含一些基础信息，例如，销售额、支票和订单。历史表也可以是 `rollup` 表，包含通过操作 `GROUP BY`、`AVERAGE` 或 `COUNT` 从基表信息导出的摘要信息。

历史表中的时间间隔通常是一个滚动窗口，DBA 定时删除描述旧事务的记录行集，并依次为描述近期事务的行分配空间。例如，在 1995 年 4 月 30 日商务结束时，DBA 删除了从 1994 年以来的事务的所有行，并为 1995 年 4 月的事务分配空间。

现在，考虑一个特定的实例。

用户有一个表 `ORDER`，该表包含 13 个月的事务：一年的历史数据外加当前月的订单。每个月有一个分区，分区与它们所在的表空间一样命名为 `ORDER_yymm`。

表 `ORDER` 包含两个局部索引，`ORDER_IX_ONUM` 是在顺序编号上的局部、前缀和唯一性索引，`ORDER_IX_SUPP` 是供应商编号上的局部、非前缀索引。局部索引分区用于基表相匹配的后缀命名。`ORDER_IX_CUST` 为针对客户名的全局唯一索引。`ORDER_IX_CUST` 包含三个分区，每个包含三分之一字母。因此，在 1994 年 10 月 31 日，按如下方法改变 `ORDER` 表的时间窗口：

1. 备份最早时间段的数据。

```
ALTER TABLESPACE order_9310 BEGIN BACKUP;
...
ALTER TABLESPACE order_9310 END BACKUP;
```

2. 删除最早时间段的分区。

```
ALTER TABLE order DROP PARTITION order_9310;
```

3. 在最近的时间段增加分区。

```
ALTER TABLE order ADD PARTITION order_9411;
```

4. 重新创建全局索引分区。

```
ALTER INDEX order_ix_cust REBUILD PARTITION order_ix_cust_AH;
```

```
ALTER INDEX order_ix_cust REBUILD PARTITION order_ix_cust_IP;
ALTER INDEX order_ix_cust REBUILD PARTITION order_ix_cust_QZ;
```

一般来说, Oracle 可以获得足够的“锁”来保证没有操作区影响各个 DDL 语句(如 ALTER TABLE...DROP PARTITION)的执行。然而, 如果分区维护操作需要几步来完成, 则 DBA 有责任来保证应用程序(或其它维护操作)不影响正在进行中的多步操作。

下面是实现该功能的两种方法:

- 在运行已定义的批窗口操作时, 停止所有用户级的应用程序。
- 通过回收用于所有应用程序的存取权限的角色来保证没有人能访问表 ORDER。

### 15.5.2 将分区视图转换到分区表

本节说明如何将分区视图(也称作“人工分区”)转换为分区表。分区视图按如下方法定义:

```
CREATE VIEW accounts
SELECT * FROM accounts_jan98
UNION ALL
SELECT * FROM accounts_feb98
UNION ALL
...
SELECT * FROM accounts_dec98;
```

按如下步骤, 可以递增地将分区视图转换到分区表:

1. 首先, 通过创建分区表将最新的两个分区 ACCOUNTS\_NOV98 和 ACCOUNTS\_DEC98 从视图转换到表。每个分区得到一个由两个块(相当于占位符)组成的段。

```
CREATE TABLE accounts_new(...)
TABLESPACE ts_temp STORAGE (INITIAL 2)
PARTITION BY RANGE (opening_date)
(PARTITION jan98 VALUES LESS THAN ('01-FEB-1998'),
...
PARTITION dec98 VALUES LESS THAN ('01-FEB-1998'));
```

2. 使用 EXCHANGE PARTITION 语句将表转换到相应的分区。

```
ALTER TABLE accounts_new
EXCHANGE PARTITION nov98 WITH TABLE
accounts_nov98 WITH VALIDATION;

ALTER TABLE accounts_new
EXCHANGE PARTITION dec98 WITH TABLE
accounts_dec98 WITH VALIDATION;
```

这样, 与 NOV98 和 DEC98 分区相关的占位符数据段已经和与 ACCOUNTS\_NOV98 和 ACCOUNTS\_DEC98 表相关的数据段进行了交换。

3. 重新定义 ACCOUNTS 视图

```
CRAETE OR REPLACE VIEW accounts
SELECT * FROM accounts_jan98
UNION ALL
```

```
SELECT * FROM accounts_feb_98
UNION ALL
...
UNION ALL
SELECT * FROM accounts_new PARTITION (nov98)
UNION ALL
SELECT * FROM accounts_new PARTITION (dec98);
```

4. 删除 ACCOUNTS\_NOV98 和 ACCOUNTS\_DEC98 表，这两个表拥有原先附着于 NOV98 和 DEC98 分区的占位符段。

5. UNION ALL 视图中的所有表转换到分区后，删除视图，并将分区重命名为删除了的视图的名字。

```
DROP VIEW accounts;
RENAME accounts_new TO accounts;
```

# 16 管理簇

本章阐述了簇（cluster）管理的知识，包括与索引簇、簇表和簇索引管理有关的几个方面：

- 管理簇指南
- 创建簇
- 改变簇
- 删除簇

簇的另一种类型——散列簇将在第 17 章“管理散列簇”中阐述。

## 16.1 管理簇指南 7

簇提供了一种可选择的存储表数据的方法。簇是由一组共享相同数据块的表组成，之所以将表成组是因为这些表共享相同的列，并经常一起使用。例如，EMP 表和 DEPT 表共享 DEPTNO 列，当你将 EMP 和 DEPT 组成一簇时（参见图 16-1），Oracle 为来自 EMP 表和 DEPT 表中相同数据块的每个单元存储所有的行。用户不必对那些经常单独存取的表使用簇。

由于簇将不同表的相关行存储在同一个数据块，因而正确应用簇将带来两个主要的利益：

- 磁盘输入/输出减少了，存取簇表的时间得到改善
- 簇关键字是簇表共有的列或成组的列，用户在创建簇时指明作为簇关键字的列，以后在创建加入到簇中的表是也应指定相同的列。每个簇关键字的值只在簇和簇索引中存放一个，不管不同的表的多少行都包含该值。

因此，存储簇中相关表和索引数据要求的存储空间可能比非簇表格式要求的存储空间更少。例如，注意对于 EMP 表和 DEPT 表中包含相同值的许多行，每个簇关键字（每个 DEPTNO）是如何只存储一次的。

创建簇后用户可创建簇中的表。可是，必须在插入行到簇表中之前创建簇索引。使用簇并不影响簇表的附加索引的创建，它们可以和以往一样创建和删除。

下面将阐述管理簇的指南，包括：

- 选择适当的表组成簇
- 选择适当列作为簇关键字
- 指定数据块空间使用
- 指定簇关键字及其关联行通常所要求的空间
- 指定每个簇和簇索引行的定位
- 估计簇的大小和设置存储参数

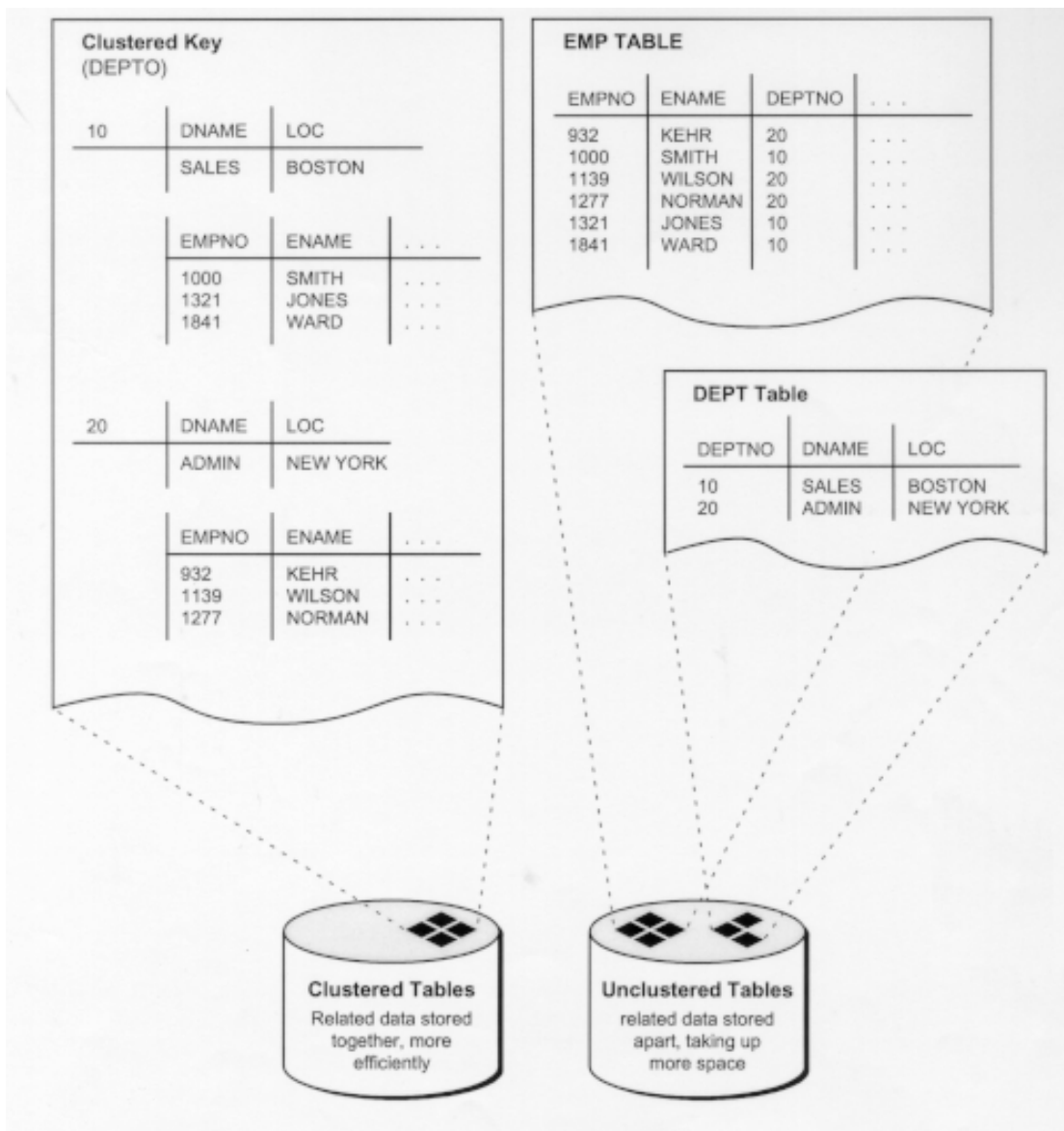


图 16-1 簇表数据

### 16.1.1 选择适当的表组成簇

使用簇来存储主要被查询（而不是插入或更新）的一个或多个表，这些查询通常是连接簇中多个表的数据或恢复单个表的相关数据。

### 16.1.2 选择适当的列作为簇关键字

必须非常小心地选择簇关键字列。如果连接表的查询中使用多个列，那么将簇关键字设置为一复合关键字。一般来说，一个好的簇索引的特性应与其它索引的特性一样。

一个好的簇关键字有非常多的唯一值，使得与每个关键字值相关的一组行能准确地填

充一数据块。含有太少行的簇关键字值会浪费空间，导致性能微不足道的增益。指定的簇关键字如果只有少数行共享一个共同的值，那么会引起块中存在废弃的空间，除非在簇创建时指定了一个较小的 **SIZE**(规模)值。

含有太多行的簇关键字值会导致需要更多的搜索。簇关键字值太普通（比如 **MALE** 和 **FEMALE**）会导致过度的搜索，使得性能比不创建簇时有所降低。

簇索引不可以是唯一的或包括一个定义为 **LONG** 的列。

### 16.1.3 指定数据块空间使用

在创建簇过程中指定 **PCTFREE** 和 **PCTUSED** 参数，可影响空间利用率以及为更新簇数据段的数据块中的当前行而预留空间的数量。记住为簇中创建的表设置的 **PCTFREE** 和 **PCTUSED** 参数是可以忽略的。

### 16.1.4 指定簇关键字及其关联行通常所要求的空间

**CREATE CLUSTER** 语句有一可选参数 “**SIZE**”，该参数用来估计一般簇关键字和与其关联行所要求的字节数。Oracle 执行下列任务时使用 **SIZE** 参数：

- 估计簇数据块中簇关键字及其相关行的数量。
- 限制放置在簇数据块中的簇关键字的数量，这将极大地提高簇中关键字的存储效率。

**SIZE** 不限制一给定簇关键字的可使用空间，例如，如果 **SIZE** 设置为两个簇关键字可安装在一个数据块中，那么任何可用的数据块空间都可被这两个关键字的任何一个使用。

缺省情况下，Oracle 只在簇数据段的每个数据块中存储一个簇关键字和它所关联的行。尽管块的大小因操作系统不同而不同，但是当簇表导入到其它机器上的其它数据库时，一个块只有一个关键字的原则仍保持不变。

如果一给定簇关键字的所有行无法安装在一个块中，将把多个块束缚在一起，从而提高存取给定关键字所有值的速度。簇索引指向联结块的开始端，每个联结块包含需关键字值及其相关联的行。如果 **SIZE** 设置为多个关键字安装在一个块中，块可以属于多个链。

### 16.1.5 指定每个簇和簇索引行的定位

如果用户拥有适当的权限和表空间配额，那么用户可在当前联机的任何表空间中创建一个新的簇和相关联的簇索引，永远在 **CREATE CLUSTER/INDEX** 语句中指定 **TABLESPACE** 选项，以便指明该表空间用来存放新的簇或索引。

可在不同的表空间中创建簇及其索引，事实上，在存储于不同存储设备上的表空间中创建簇及其索引，能以最小的磁盘争用同时提取表数据和索引数据。

### 16.1.6 估计簇的大小和设置存储参数

以下是创建簇前估计该簇的大小可产生的好处：

- 用户可使用对簇的估计同对索引、回退段和重做（**REDO**）日志文件的估计一起来确定容纳一个预期数据库所要求的磁盘空间的容量。从这些估计中，用户可制定正确的硬件购置和其他决策。
- 用户可使用对一个独立的簇的估计来更好地管理簇将使用的磁盘空间。创建簇时，

用户可设置适当的存储参数，改进使用簇的应用程序的 I/O 性能。

无论创建表前是否估计了表的大小，都可在创建每个非簇表时显式地设置存储参数。如果用户创建或以后更新表时没有显式地设置存储参数，那么它将自动使用为该表所驻留的表空间设置的相应的缺省存储参数。簇表也自动使用簇的存储参数。

## 16.2 创建簇

在自己的模式中创建簇，必须拥有 **CREATE CLUSTER** 的系统权限和欲包含该簇的表空间的配额，或 **UNLIMITED TABLESPACE** 的系统权限。

在其他用户的模式中创建簇，必须拥有 **CREATE ANY CLUSTER** 的系统权限，所有者必须有包含该簇的表空间的配额或 **UNLIMITED TABLESPACE** 的系统权限。有关系统权限的更多信息见第 23 章“管理用户权限和规则”；有关表空间配额的更多信息见第 22 章“管理用户和资源”。

使用 **CREATE CLUSTER** 语句创建簇，下面语句创建了一个名为 **EMP\_DEPT** 的簇，该簇存储了 **EMP** 表和 **DEPT** 表，通过 **DEPTNO** 列集成簇：

```
CREATE CLUSTER emp_dept (deptno NUMBER(3))
    PCTUSED 80
    PCTFREE 5
    SIZE 600
    TABLESPACE users
    STORAGE (INITIAL 200K
        NEXT 300K
        MINEXTENTS 2
        MAXEXTENTS 20
        PCTINCREASE 33);
```

如果没有指定 **INDEX** 关键字，正如本例，那么将缺省创建索引簇 (**index cluster**)。用户也可指定散列参数 (**HASHKEYS**、**HASH IS** 或 **SINGLE TABLE HASHKEYS**) 来创建一个散列簇。散列簇将在第 17 章“管理散列簇”中阐述。

### 16.2.1 创建簇表

在簇中创建一个表，用户必须有 **CREATE TABLE** 或 **CREATE ANY TABLE** 的系统权限。在簇中创建表不需要表空间配额或 **UNLIMITED TABLESPACE** 的系统权限。

使用带有 **CLUSTER** 选项的 **CREATE TABLE** 语句在簇中创建一个表，可使用下列语句在 **EMP\_DEPT** 簇中创建 **EMP** 表和 **DEPT** 表：

```
CREATE TABLE dept (
    deptno NUMBER(3) PRIMARY KEY, ... )
    CLUSTER emp_dept (deptno);

CREATE TABLE emp (
    empno NUMBER(5) PRIMARY KEY,
    ename VARCHAR2(15) NOT NULL,
```

```
...
deptno NUMBER (3) REFERENCES dept)
CLUSTER emp_dept (deptno);
```

**注意** 用户可在 `CRATE TABLE` 语句中为簇表指定模式。簇表可在不同的模式中，该模式包含该簇。同样，不要求列名相匹配，但它们的结构必须匹配。

### 16.2.2 创建簇索引

下列条件之一为真时方可创建簇索引：

- 用户的模式包含该簇并且用户具有 `CREATE INDEX` 的系统权限
- 用户具有 `CREATE ANY INDEX` 的系统权限

无论哪种情况，用户都必须有包含该簇索引的表空间的配额，或 `INLIMITED TABLESPACE` 系统权限。

簇索引必须在插入任何行到簇表之前创建，下面语句为 `EMP_DEPT` 簇创建了一簇索引：

```
CREATE INDEX emp_dept_index
ON CLUSTER emp_dept
INITRANS 2
MAXTRANS 5
TABLESPACE users
STORAGE (INITIAL 50K
NEXT 50K
MINEXTENTS 2
NAXEXTENTS 10
PCTINCREASE 33)
PCTFREE 5;
```

簇索引子句（`ON CLUSTER`）指出为 `EMP_DEPT` 簇创建簇索引，该语句亦显式地指出了几个对簇和簇索引的存储参数的设置。

## 16.3 改变簇

用户模式中必须包含该簇或用户具有 `ALTER ANY CLUSTER` 的系统权限，用户可通过改变以下设置改变一存在的簇：

- 物理属性（`PCTFREE`、`PCTUSED`、`INITRANS`、`MAXTRANS` 和存储特征）
- 为簇关键字值存储所有行所需要的一般的空间容量
- 缺省平行度

另外，用户可显式地为簇分配一新的盘区或释放簇的末端任何不使用的盘区，**Oracle** 动态地为簇的数据分配所需的额外盘区。然而，在有些情况下，用户可能想显式地为一簇分配额外的盘区。例如，用户使用 **Oracle** 并行服务器时，可显式地为一指定的实例分配簇的盘区。已分配给表的块必要时重新组织（不是立即），用户可使用带有 `ALLOCATE EXTENT` 子句的 `ALTER CLUSTER` 语句为簇分配一新的盘区。

当用户改变数据块空间使用参数（`PCTFREE` 和 `PCTUSED`）或簇的规模参数（`SIZE`）时，新的设置有效于所有簇所使用的数据块，包括已分配的块和即将分配给簇的块，同时对



MAXTRANS 的设置有效于簇的所有块（已分配的和即将分配的块）。

存储参数 INITIAL 和 MINEXTENTS 不可改变，对其他存储参数的新的设置只影响即将分配给簇的盘区。

使用 ALTER CLUSTER 语句改变一个簇。下面语句改变 EMP\_DEPT 簇：

```
ALTER CLUSTER emp_dept  
  PCTFREE 30  
  PCTUSED 60;
```

### 16.3.1 改变簇表和簇索引

可使用 ALTER TABLE 语句改变簇表，但所有用户在 ALTER TABLE 语句中为簇表设置的数据块空间参数、事务处理参数或存储参数将产生错误信息（ORA-01771，“illegal option for a xclustered table”），因此，用户使用 ALTER TABLE 语句只是用来增加或修改列、删除非簇关键字的列或增加、删除、启用或禁止簇表的完整性约束或触发器。

用户就像改变其它索引一样改变簇索引。

**注意** 估计簇索引的大小时要记住索引位于每个簇关键字之上而不是实际的行，因此，每个关键字只在索引中出现一次。

## 16.4 删除簇

如果簇中的所有表不再有用时可将该簇删除。当一个簇被删除时，所有属于簇数据段和簇索引段的盘区都被返回给所包含的表空间，并可用于表空间中的其它段。

使用 DROP CLUSTER 语句删除不含任何表的簇和簇索引。例如，下面语句删除名为 EMP\_DEPT 的空簇：

```
DROP CLUSTER emp_dept;
```

如果簇包含一个或多个簇表并且用户想同时删除表，则加上 DROP CLUSTER 语句的 INCLUDING TABLES 选项，正如：

```
DROP CLUSTER emp_dept INCLUDING TABLES;
```

如果不包含 INCLUDING TABLES 选项并且簇里有表，则返回一条错误信息。

如果簇中的一个或多个表包含主键或外部键，这些键由簇外部的表的 FOREIGN KEY 约束引用，该簇不可删除，除非从属 FOREIGN KEY 约束也被删除。使用带有 CASCADE CONSTRAINTS 选项的 DROP CLUSTER 语句这样做很容易，正如下例所示：

```
DROP CLUSTER emp_dept INCLUDING TABLES CASCADE CONSTRAINTS;
```

如果用户不使用 CASCADE CONSTRAINTS 选项并约束存在，Oracle 将返回一条错误信息。

### 16.4.1 删除簇表

删除一个簇，用户的模式中必须包含该簇或用户必须有 DROP ANY CLUSTER 系统权限，用户不再需要其它权限来删除一个含有表的簇，即使该簇表不被簇的所有者所有。

簇表可被单个删除而并不影响表的簇、其它簇表或者簇索引。删除一个簇表正如删除一个非簇表一样使用 **DROP TABLE** 语句。

**注意** 当用户从簇中删除单个表时，Oracle 单独删除表中的每一行。删除整个簇的最有效的方法是使用带有 **INCLUDING TABLES** 选项的 **DROP CLUSTER** 语句删除包含所有表的簇。只有当用户仍想保留簇中的其它表时才会从簇中删除单独的表（使用 **DROP TABLE** 语句）。

#### 16.4.2 删除簇索引

一个簇索引可被删除而不影响该簇或它的簇表，但是，如果没有簇索引簇表则不可用；用户必须重建簇索引使得能够访问其它簇。有时簇索引只是作为重建一个碎片簇索引过程的一部分而被删除。

# 17

## 管理散列簇

本章阐述了如何管理散列簇（hash cluster），包括以下几个方面：

- 用户是否应该使用簇
- 创建散列簇
- 改变散列簇
- 删除散列簇

### 17.1 是否应该使用散列簇

在散列簇中存储表是提高数据检索性能的一种可选方法，散列簇提供了一种选择，使得非簇表具有索引或索引簇。借助于索引表或索引簇，Oracle 使用存储在一单独的索引中的关键字值定位表中的行。用户可创建一散列簇并将表加载到其中。Oracle 从物理上存储散列簇中某表的所有行，按照散列函数（hash function）的结果检索它们。

Oracle 使用散列函数来产生数字化数值的分布，称为散列值（hash values），这些数值基于某指定的簇关键字的值。一个散列簇的关键字就像一索引簇的关键字一样可以是一单个的列或组合关键字（复合列关键字）。Oracle 将散列函数应用于行的簇关键字值来查找或存储散列簇中的一行，散列函数的结果值应与簇中的数据块相吻合，而且 Oracle 以后可在发出的语句中读或写这些结果值。

在一索引表或簇中查找或存储一行，至少必须执行下面两个 I/O 操作（通常更多）：

- 一个或多个 I/O 操作用来查找或存储索引中的关键字数值
- 另一个 I/O 用来读或写表或簇中的行

相反，Oracle 使用散列函数来定位散列簇中的行，不需要 I/O 操作。结果至少必须有一个 I/O 操作用来读或写散列簇中的行。

本节列举了使用散列簇的优点和缺点，帮助用户根据自己的情况确定是否采用散列簇。

#### 17.1.1 散列簇优点

如果用户选择使用索引簇而不是散列簇的话，应考虑是单独存储一表还是作为簇的一部分。

当用户的条件如下时，使用散列簇将发挥极大的优势：

- 大多数查询都集中在簇关键字上：

```
SELECT ... WHERE cluster_key = ...;
```

在这样的情况下，同等条件的簇关键字是散列化的，相应的散列关键字是由单个的读找到的。相比而言，索引表的关键字数值必须首先在索引中查找（通常是几个读），然后从表中读行（另一个读）。

- 散列簇中的表的大小主要都是静态的，所以用户可以确定簇中表的行数以及所要求的空间。如果散列簇中的表所需要的空间比初始分配给簇的多，结果由于需要过多的块造成性能下降。

### 17.1.2 散列簇的缺点

在下列情况下，散列化不具有优势：

- 大多数对表的查询要检索簇关键字值之外的行。例如，像下面那样全表扫描或查询，则不可用散列函数来明确指定散列关键字的定位，取而代之的是，必须执行与全表扫描的操作来取得所查询的行：

```
SELECT ... WHERE cluster_key < ...;
```

由于索引中关键字值是按序排列的，所以满足 **WHERE** 子句的簇关键字值用相对少的 I/O 操作便可查找到。

- 表不是静态的，而是在不断增长。如果表无限制地增长，那么表所要求的空间则无法预计。
- 应用程序经常执行全表扫描且表排列稀松，这种情况下在簇中进行全表扫描需要更长时间。
- 用户无法预分配散列簇需要的空间。

参见 即使用户决定使用散列簇，表仍可在每个列上设置单独的索引，包括簇索引。

## 17.2 创建散列簇

使用 **CREATE CLUSTER** 语句创建散列簇，但可以指明 **HASHKEYS** 子句。下面例子包含了创建名为 **TRIAL\_CLUSTER** 簇的语句，该簇存储 **TRIAL** 表，由 **TRIALNO** 列（簇关键字）构成簇，另一条语句是创建簇中的表。

```
CREATE CLUSTER trial_cluster (trialno NUMBER(5,0))
    PCTUSED 80
    PCTFREE 5
    TABLESPACE users
    STORAGE (INITIAL 250K    NEXT 50K
             MINEXTENTS 1    MAXEXTENTS 3
             PCTINCREASE 0)
    HASH IS trialno HASHKEYS 150;
```

```
CREATE TABLE trial (
    Trialno NUMBER (5,0) PRIMARY KEY,
    ...)
    CLUSTER trial_cluster (trialno);
```

和索引簇一样，散列簇的关键字可以是一单个的列或复合关键字（复合列关键字）。在该例中，它是一单个列。

**HASHKEYS** 的值 150 指明和限定了唯一散列值的数量，这些散列值可以由簇使用的散列函数产生。Oracle 舍入指定给最临近的质数。

如果没指定 **HASH IS** 子句, Oracle 使用一内部散列函数。如果簇关键字已经是一唯一标识, 用户可绕过内部散列函数, 指定簇关键字为散列值, 正如上例一样。用户也可使用 **HASH IS** 子句来指定一用户定义的散列函数。

用户不可在散列簇中创建散列索引, 且不必在散列簇关键字上创建索引。

关于在簇中创建表的其它信息、如何设置为索引和散列簇所共有的 **CREATE CLUSTER** 语句的参数以及创建簇所需的权限见第 16 章“管理簇”。本节阐明了 **CREATE CLUSTER** 语句中针对散列簇的参数设置:

- 创建单表散列簇
- 控制散列簇中空间的使用
- 如何估计散列簇所需的空间和设置存储参数

### 17.2.1 创建单表散列簇

用户可创建单表散列簇 (**single-table hash cluster**), 以便能快速访问表中的行, 但是, 该表必须是散列簇中的唯一的表。实质上, 散列关键字和数据行之间存在一对一的映射。下面语句创建了一名为 **PEANUT** 的单表散列簇, 散列关键字是 **VARIETY**:

```
CREATE CLUSTER peanut(variety NUMBER)
SIZE 512 SINGLE TABLE HASHKEYS 500;
```

Oracle 将 **HASHKEY** 的值舍入到最临近的质数, 所以该簇最多有 503 个散列关键字值, 每个大小为 512 字节。

**注释** **SINGLE TABLE** 选项只对散列簇有效, 必须指定 **HASHKEY**。

### 17.2.2 控制散列簇的使用空间

创建一个散列簇最重要的是正确选择簇关键字以及设置 **HASH IS**、**SIZE** 和 **HASHKEYS** 参数使得性能和空间的利用达到最优。下列准则描述了如何设置这些参数。

#### 挑选关键字

挑选正确的簇关键字取决于簇表遇到的最常见的查询语句的类型。例如, 考虑一散列簇的 **EMP** 表, 如果查询语句经常通过雇员号来选择行, 则 **EMPNO** 列应该是簇关键字。如果查询语句经常通过部门号选择行, 则 **DEPTNO** 列应该是簇关键字。对于只包含一个表的散列簇, 簇关键字通常是所包含表的全部的主键。

与索引簇的关键字一样, 散列簇的关键字可以是单个的列或复合关键字 (复合列关键字)。具有复合关键字的散列簇必须使用 Oracle 的内部散列函数。

#### 设置 **HASH IS**

只有当簇关键字是一数据类型为 **NUMBER** 的单个列且一律包含分布的整数时才可指定 **HASH IS** 参数。如果提供上述条件, 用户可分配簇中的行, 这样每个唯一的簇关键字数值散列为一唯一的散列数值, 不存在任何冲突。如果没提供这些条件, 则忽略这个选项, 这样用户使用内部散列函数。

设置 SIZE

SIZE 应设置为对某一给定散列关键字容纳所有行所需空间的平均数量，因此，为了正确设置 SIZE，用户必须注意用户数据的特性：

- 如果散列簇只用来包含一个表且该表中所有行的散列关键字数值都是唯一的，那么 SIZE 可设置为簇中行的大小的平均值；
- 如果散列簇将包含多个表，那么 SIZE 可以设置为容纳所有与一有代表性的散列数值相关联的行所需的空间的平均值；
- 如果散列簇不使用内部散列函数（如果用户指定 HASH IS）并且用户预料只有极少的冲突或根本没有冲突，那么用户可将 SIZE 设置为估计值，不会发生任何冲突且空间得到尽可能高的使用率；
- 如果用户预料插入时经常会发生冲突，则用来存储行的数据块产生溢出的可能性就高。为了减少块溢出的可能性，冲突频繁时提高系统性能，用户应该增加 SIZE 值，如下表所显示的一样。

| 每个块的有效空间/推算的 SIZE | SIZE 的设置     |
|-------------------|--------------|
| 1                 | 推算的 SIZE     |
| 2                 | 推算的 SIZE+15% |
| 3                 | 推算的 SIZE+12% |
| 4                 | 推算的 SIZE+8%  |
| >4                | 推算的 SIZE     |

**过高估算** SIZE 的值将增加簇中未使用的空间的数量。如果空间效率比数据检索性能重要，不用考虑上面的调整，使用估计的 SIZE 的值。

设置 HASHKEYS

为最大限度地散列簇中分配行，Oracle 将 HASHKEYS 的值舍入到最邻近的质数。

控制散列簇的空间：举例

下面例子显示了如何正确选择簇关键字以及设置 HASH IS、SIZE 和 HASHKEYS 参数。对所有例子，假设数据块的大小是 2K，平均每个块的 1950 字节是可用数据空间（块的大小减去系统开销）。

**例 1：**用户决定将表 EMP 加载到散列簇中，大多数查询语句通过雇员号检索雇员记录，用户预计任何时刻 EMP 表中最大行数为 10000，平均每行的大小是 55 字节。

本例中，EMPNO 应当是簇关键字。由于该列包含的整数是唯一的，所以可以回避内部散列函数。可将 SIZE 设置为平均行值——55 字节，注意 34 个簇关键字被分配到每个数据块，HASHKEYS 可设置为表中的行数，10000，舍入到最近的质数——10007。

```
CREATE CLUSTER emp_cluster( empno
NUMBER)
...
SIZE 55
HASH IS empno HASHKEYS 10007;
```

**例 2:** 条件与上一例相似, 但在此例中, 通过部门号检索行, 至多有 1000 个部门, 每个部门平均有 10 个雇员, 注意部门号以 10 增长 (0, 10, 20, 30, ...)。

本例中, DEPTNO 是簇关键字, 由于该列包含统一分配的整数, 所以可回避内部散列函数。SIZE 的预计值 (要求容纳每个部门所有行的平均空间) 是 55\*10 字节, 即 550 字节。将 SIZE 设置为该值, 每个数据块只可分配 3 个散列关键字。如果用户估计会有冲突并且希望最大限度地发挥数据检索的性能, 则稍稍改变 SIZE 的预计值以免发生块溢出冲突。调整 SIZE, 增加 12%, 即调整到 620 字节, 每个数据块分配 3 个散列关键字, 给行留有更多的空间处理预计的冲突。

HASHKEYS 可设置为唯一的部门号的值, 1000, 舍入到最近的质数--1009。

```
CREATE CLUSTER emp_cluster (deptno NUMBER)
```

```
SIZE 620
```

```
HASH IS deptno HASHKEYS 1009;
```

### 17.2.3 如何估计散列簇要求的空间 and 如何设置存储参数

正如和索引簇一样, 估计散列簇中数据所要求的存储空间是非常重要的。

Oracle 担保按照 SIZE 和 HASHKEYS 的设置空间的初始分配足够存储散列表。如果存储参数 INITIAL、NEXT 和 MINEXTENTS 的设置不能说明散列表的大小, 则需要分配额外的盘区直至达到 SIZE\*HASHKEYS。例如, 假设数据块大小是 2K, 每个块的可用数据空间接近 1900 字节 (数据块大小减去系统开销), CREATE CLUSTER 语句中的 STORAGE 和 HASH 参数设置如下:

```
STORAGE (INITIAL 100K
```

```
NEXT 150K
```

```
MINEXTENTS 1
```

```
PCTINCREASE 0)
```

```
SIZE 1500
```

```
HASHKEYS 100
```

本例中, 每个数据块只分配一个散列关键字, 因此散列簇要求的初始空间至少是 100\*2K 即 200K, 存储参数的设置并不能说明该需求, 因此, 100K 的初始盘区和 150K 的第二个盘区将分配给散列簇。

换句话说, 假设 HASH 参数设置如下:

```
SIZE 500 HASHKEYS 100
```

这种情况下, 每个数据块分配三个散列关键字, 因此, 散列簇所要求的初始空间至少是 34\*2K 或 68K, 对存储参数的初始化设置足够满足该需求 (一个 100K 的初始盘区被分配给散列簇)。

## 17.3 改变散列簇

用户可使用 ALTER CLUSTER 语句改变一散列簇:

```
ALTER CLUSTER emp_dept ...;
```

改变散列簇的含义与改变索引簇的含义是一样的，但是，注意不可在 **ALTER CLUSTER** 语句中指定 **SIZE**、**HASHKEYS** 和 **HASH IS** 参数。用户必须重创建簇来改变这些参数，然后从原始簇中拷贝所有数据。

## 17.4 删除散列簇

用户可使用 **DROP CLUSTER** 语句删除一散列簇：

```
DROP CLUSTER emp_dept;
```

使用 **DROP TABLE** 语句删除散列簇中的表，删除散列簇和散列簇中的表的含义和删除索引簇的一样。



# 18 管理视图、序列和别名单元

本章阐述了有关视图管理方面的内容，主要包括下列主题：

- 视图管理
- 序列管理
- 别名单元管理

在尝试本章中描述的任务之前，请熟练掌握第 12 章“管理模式对象指南”中出现的概念。

## 18.1 管理视图

视图 (view) 是包含在一个或多个表 (table) (或其它视图) 中数据的简化描述，执行一个查询结果，并将它看成一个表。用户可将视图看作“存储查询 (stored query)” 或一个“虚拟表 (virtual table)”，任何使用表的地方都可使用视图。

本节阐述了视图的管理，包括下列主题：

- 创建视图 (Creating Views)
- 更新连接视图 (Updating a Join View)
- 改变视图 (Altering Views)
- 删除视图 (Dropping Views)
- 替换视图 (Replacing Views)

### 18.1.1 创建视图

创建一个视图，必须满足以下要求：

- 在用户自己的模式中创建视图，该用户必须具有“CREATE VIEW”的权限；在其他用户的模式中创建视图，该用户必须具有“CREATE ANY VIEW”的系统权限。用户可明确地或通过某个角色获得这些权限。
- 视图的所有者（无论是自己还是其他用户）都必须明确授权可访问视图定义中涉及到的对象，所有者 (owner) 不可通过其他角色得到这些权限，同样，视图的功能性依赖于视图所有者的权限。例如，如果视图的所有者对 Scott 的 EMP 表只有“INSERT”权限，那么视图只可用来往 EMP 表中插入新的行，不可从 EMP 表中选择 (SELECT)、修改 (UPDATE) 或删除 (DELETE) 行。
- 如果视图的所有者想向其他用户授予访问该视图的权限，他必须具有对基本对象的“GRANT OPTION”的对象权限或“ADMIN OPTION”的系统权限。

用户使用“CREATE VIEW”语句创建视图，视图建立在相关表、快照 (snapshot) 或其它视图的查询之上，定义在视图中的查询语句不可含有“FOR UPDATE”子句。

下列语句创建了 EMP 表中一组数据的视图：

```
CREATE VIEW sales_staff AS
  SELECT empno, ename, deptno
  FROM emp
  WHERE deptno = 10
  WITH CHECK OPTION CONSTRAINT sales_staff_cnst;
```

SALES\_STAFF 视图是查询 10 系中的相关行，因此，CHECK OPTION 创建具有限制（称为 SALES\_STAFF\_CNST）的视图，作用于视图的 INSERT 语句和 UPDATE 语句不可作用于视图无法选出的行。例如，下面 INSERT 语句可通过 SALES\_STAFF 视图将一行记录成功地写入 EMP 表中：

```
INSERT INTO sales_staff VALUES (7584, 'OSTER', 10);
```

但是，下面的 INSERT 语句将返回一错误信息，因为它企图插入一条 department 为 30 的记录，而这样的记录是 SALES\_STAFF 视图无法得到的：

```
INSERT INTO sales_staff VALUES (7591, 'WILLIAMS', 30);
```

可以指定“WITH READ ONLY”子句优化创建视图，阻止任何通过视图更新、插入、删除基表的行为。如果没指明“WITH”语句，具有某些限制的视图很容易被更新。

**注意** 创建视图的相关限制包含在“Oracle8i SQL Reference”中“CREATE VIEW”语句的描述中。

**参见** 关于创建视图、替换视图、更新视图、改变视图和删除视图的详细的语法、限制和授权信息请参阅“Oracle8i SQL Reference”。

### 连接视图 (Join Views)

用户可通过 FROM 子句建立基于多于一个表或视图的视图，这些视图称为连接视图(join views)。下列语句创建了一个连接 EMP 表和 DEPT 表有关数据的视图：

```
CREATE VIEW division1_staff AS
  SELECT ename, empno, job, dname
  FROM emp, dept
  WHERE emp.deptno IN (10, 30)
  AND emp.deptno = dept.deptno;
```

一个可更新的视图允许对其进行 UPDATE、INSERT 和 DELETE 等操作，详情请参见第 18.1.2 节“更新连接视图”。

### 视图创建时查询定义 (defining query) 的扩充

为符合 ANSI/ISO 标准，Oracle 在创建视图时可使用通配符，并将查询结果存放到数据字典，可保持其他查询的完整性。扩展列表中的列名用双引号封装，这些列名将被提交给服务器，并检查语法正确性。

例如，假设创建一个 DEPT 视图：

```
CREATE VIEW dept AS SELECT * FROM scott.dept;
```

也就是说 Oracle 将 DEPT 视图的查询定义存储为：

```
SELECT "DEPTNO", "DNAME", "LOC" FROM scott.dept
```

创建时有语法错误的视图没有扩展通配符，但是，如果编译视图时没检查出错误，则可在查询定义中加入通配符。

### 错误创建视图

如果“CREATE VIEW”语句没有语法错误，即使其定义的查询语句无法执行，Oracle 依然会创建该视图，该视图被认为是“错误创建的视图”。例如，如果某视图创建时涉及到一不存在的表或无效的列名，或者视图的所有者不具备所要求的权限，该视图仍将创建并送入到数据字典，但该视图是不可用的。

创建一个存在错误的视图，必须在“CREATE VIEW”语句中加上“FORCE”选项。

CREATE FORCE VIEW AS ....;

缺省情况下，有错误的视图不可能有效地创建，即其状态不可能是 VALID。如果用户试图创建这样的视图，Oracle 将返回一条信息指明该视图创建时含有错误，且该视图的状态是 INVALID。如果以后情况发生变化，则该视图定义的查询可正确执行，该视图可重新编译，其状态变为“VALID”。有关条件改变的信息以及它们对视图的影响请参见第 19 章的 19.7 节“管理对象从属关系”。

### 18.1.2 更新连接视图

一个可更新的连接视图在 SELECT 语句的 FROM 子句中包含不止一个表，且不受“WITH READ ONLY”子句的限制。

**注意** 有很多限制和条件可影响一个连接视图是否可更新，在“Oracle8i SQL Reference”中 CREATE VIEW 语句的描述中详细列举了有关限制和条件，请参阅该书。

另外，如果一个视图是其它一些嵌套视图的联合，则这些嵌套的视图必须合并到视图的顶端。关于合并视图和非合并视图的讨论以及如何优化相关视图语句请参见“Oracle8i Concepts”和“Oracle8i Designing and Tuning for Performance”。

数据字典视图可表明连接视图中的列是否可更新，参见表 18-1。

更新连接视图的规则如下：

| 规则        | 描述  |
|-----------|---|
| 一般规则      | 连接视图的任何 INSERT、UPDATE 或 DELETE 操作在同一时刻只能修改其下属的一个基表  |
| UPDATE 规则 | 连接视图的所有可更新的列必须影射到一个保留关键字表（key-preserved table）的列，如果该视图定义时带有“WITH CHECK OPTION”子句，那么所有联合列和重复表的所有列都不可更改   |
| DELETE 规则 | 只要明确存在一个保留关键字表（key-preserved table），那么连接视图中的行都可删除。如果视图定义中含有“WITH CHECK OPTION”子句而且保留关键字表被重复，那么不可从视图中删除行 |
| INSERT 规则 | INSERT 语句不可显式或隐式地涉及保留非关键字表（non-key preserved table）的列，如果连接视图定义中含有“WITH CHECK OPTION”子句，                 |

有关这些规则的举例说明和保留关键字表的讨论请看下一节。

下面给出的例子只有当用户在表中显式定义主关键字和外部关键字或唯一索引时才能正常执行，下面是有约束条件的 EMP 表和 DEPT 表的正确定义：

```
CREATE TABLE dept (  
    deptno      NUMBER(4) PRIMARY KEY,  
    dname       VARCHAR2(14),  
    loc         VARCHAR2(13);  
  
CREATE TABLE emp (  
    empno       NUMBER(4) PRIMARY KEY,  
    ename       VARCHAR2(10),  
    job        VARCHAR2(9),  
    mgr         NUMBER(4),  
    sal        NUMBER(7,2),  
    comm       NUMBER(7,2),  
    deptno     NUMBER(2),  
    FOREIGN KEY (DEPTNO) REFERENCES DEPT (DEPTNO));
```

用户可忽略上面列出的主关键字和外部关键字的约束，在 DEPT 表中创建一 UNIQUE INDEX（唯一索引），使得下面的例子能正确执行。

下列语句创建了一个与这些例子有关的 EMP\_DEPT 连接视图：

```
CREATE VIEW emp_dept AS  
    SELECT emp.empno, emp.ename, emp.deptno, emp.sal, dept.dname, dept.loc  
    FROM emp, dept  
    WHERE emp.deptno = dept.deptno  
    AND dept.loc IN ('DALLAS', 'NEW YORK', 'BOSTON');
```

#### 保留关键字表（key-preserved table）

保留关键字表的概念是理解有关修改连接视图的限制的基础，如果一个表的每个关键字都是连接视图的关键字，那么这个表就是保留关键字表。所以，一个保留关键字表通过连接视图保留关键字。

**注意** 一个表选出的关键字并非一定要是保留关键字，如果关键字被选出，那么这些关键字可以是连接视图结果的关键字，这就足够了。

一个表的保留关键字的属性并不依赖于该表中的实际数据，而是其模式属性。例如，如果 EMP 表中每个部门有多个雇员，那么 EMP 表和 DEPT 表的联合结果中 DEPTNO 将是唯一的，但是 DEPT 表仍不是一保留关键字表。

如果从 EMP\_DEPT 中选择所有行，那么结果如下：

| EMPNO | ENAME  | DEPTNO | DNAME      | LOC      |
|-------|--------|--------|------------|----------|
| 7782  | CLARK  | 10     | ACCOUNTING | NEW YORK |
| 7839  | KING   | 10     | ACCOUNTING | NEW YORK |
| 7934  | MILLER | 10     | ACCOUNTING | NEW YORK |

|      |       |    |          |        |
|------|-------|----|----------|--------|
| 7369 | SMITH | 20 | RESEARCH | DALLAS |
| 7876 | ADAMS | 20 | RESEARCH | DALLAS |
| 7902 | FORD  | 20 | RESEARCH | DALLAS |
| 7788 | SCOTT | 20 | RESEARCH | DALLAS |
| 7566 | JONES | 20 | RESEARCH | DALLAS |

row selected.

在该视图中，EMP 是一保留关键字表，因为 EMPNO 是 EMP 表的关键字，也是连接视图结果的关键字。DEPT 表不是一保留关键字表，因为尽管 DEPTNO 是 DEPT 表的关键字，但不是连接视图结果的关键字。

### DML 语句和连接视图

一般规则即是连接视图的任何 UPDATE、INSERT 或 DELETE 语句只可修改一个下属基表，下面的例子证明了 UPDATE、DELETE 和 INSERT 语句指定的规则。

**UPDATE 语句** 下例表明成功修改 EMP\_DEPT 视图的 UPDATE 语句：

```
UPDATE emp_dept
SET sal = sal * 1.10
WHERE deptno = 10;
```

下面的 UPDATE 语句不允许作用于 EMP\_DEPT 视图：

```
UPDATE emp_dept
SET loc = 'BOSTON'
WHERE ename = 'SMITH';
```

这条语句将产生 ORA-01779 错误（“cannot modify a column which maps to a non key-preserved table”），因为它试图改变 DEPT 表，但 DEPT 表并非 EMP\_DEPT 视图的保留关键字表。

一般来说，一个连接视图的所有可更新的列必须映射到保留关键字表的所有列。如果视图定义时带有 “WITH CHECK OPTION” 子句，那么所有联合列和重复表的所有列都不可修改。

所以，例如，如果 EMP\_DEPT 定义时使用 “WITH CHECK OPTION”，那么下面的 UPDATE 语句将出错：

```
UPDATE emp_dept
SET deptno = 10
WHERE ename = 'SMITH';
```

该语句出错的原因是它试图更新一连接视图的列。

**DELETE 语句** 用户可删除有一个且只有一个保留关键字表的连接视图。

下面 DELETE 语句用来删除 EMP\_DEPT 视图：

```
DELETE FROM emp_dept
WHERE ename = 'SMITH';
```

EMP\_DEPT 视图上的 DELETE 语句是合法的，因为它可转换为 EMP 表上的 DELETE 操作，且 EMP 表是连接视图中唯一的保留关键字表。

如果用户创建下面的视图，那么不可在该视图上执行 **DELETE** 操作，因为 **E1** 和 **E2** 都是保留关键字表：

```
CREATE VIEW emp_emp AS
  SELECT e1.ename, e2.empno, deptno
  FROM emp e1, emp e2
  WHERE e1.empno = e2.empno
```

如果视图定义中使用 “**WITH CHECK OPTION**” 子句，并且保留关键字表被重复，那么不可从该视图中删除行。

```
CREATE VIEW emp_mgr AS
  SELECT e1.ename, e2.ename mname
  FROM emp e1, emp e2
  WHERE e1.mgr = e2.empno
  WITH CHECK OPTION;
```

不可在该视图上执行 **DELETE** 操作，因为视图涉及一自连接表，且该表是保留关键字表。

**INSERT 语句** 下面作用于 **EMP\_DEPT** 视图上的 **INSERT** 语句是合法的：

```
INSERT INTO emp_dept (ename, empno, deptno)
  VALUES ('KURODA', 9010, 40);
```

该语句能正确执行是因为只有一个保留关键字表（即 **EMP** 表）被修改，且 **40** 是 **DEPT** 表中有效的 **DEPTNO**（因此满足 **EMP** 表 **FOREIGN KEY** 是整型的约束）。

下面 **INSERT** 语句将无效，同样原因，**EMP** 表上 **UPDATE** 语句也同样失效，**EMP** 表的 **FOREIGN KEY**（外部关键字）受整型约束是不合理的。

```
INSERT INTO emp_dept(ename, empno, deptno)
  VALUES('KURODA', 9010, 77);
```

下面 **INSERT** 语句将出错，错误信息为 **ORA-01776** 错误（“cannot modify more than one base table through a view”）。

```
INSERT INTO emp_dept (empno, ename, loc)
  VALUES(9010, 'KURODA', 'BOSTON');
```

**INSERT** 语句不可隐式或显式地涉及非保留关键字表（**non-key-preserved table**）列。如果连接视图定义时使用 “**WITH CHECK OPTION**” 子句，那么不可执行 **INSERT** 语句。

使用 **UPDATABLE\_COLUMNS**（可更新列）视图

表 18-1 中提供的信息为用户修改连接视图提供帮助。

表 18-1 **UPDATABLE\_COLUMNS**（可更新列）视图

| 视图名称                                      | 描述  |
|---|---|
| <b>USER_UPDATABLE_COLUMNS</b> （用户可更新列）    | 显示可修改用户模式中的所有表和所有视图的所有列                   |
| <b>DBA_UPDATABLE_COLUMNS</b> （数据库管理员可更新列） | 显示可修改 <b>DBA</b> （数据库管理员）模式中的所有表和所有视图的所有列 |
| <b>ALL_UPDATABLE_VIEWS</b> （所有可更新视图）      | 显示所有可修改的表和视图中的所有列                         |

EMP\_DEPT 视图中所有可更新的列显示如下:

```
SELECT column_name, updatable
FROM user_updatable_columns
WHERE table_name = 'EMP_DEPT';
```

| COLUMN_NAME | UPD |
|-------------|-----|
| EMPNO       | YES |
| ENAME       | YES |
| DEPTNO      | YES |
| SAL         | YES |
| DNAME       | NO  |
| LOC         | NO  |

6 rows selected.

### 18.1.3 改变视图

ALTER VIEW 语句只用来明确地重编译一个无效的视图。如果用户需改变视图的定义,参见 18.1.5 节“替换视图”。

ALTER VIEW 语句允许运行前定位重编译错误,用户改变视图的某个基表后可明确地重编译视图以确保这种改变不会影响依赖于它的视图或其它对象。

使用 ALTER VIEW 语句时,视图必须在自己的模式中,否则用户必须具有“ALTER ANY TABLE”的系统权限。

### 18.1.4 删除视图

用户可删除自己模式中的任何视图,删除其他用户模式中的视图必须具有“DROP ANY VIEW”的系统权限,使用“DROP VIEW”语句删除视图。例如,下列语句用来删除 EMP\_DEPT 视图:

```
DROP VIEW emp_dept;
```

### 18.1.5 替换视图

用户必须具备删除和创建一个视图的所要求的权限才可替换一个视图,如果视图的定义必须改变,则必须替换视图,不可改变视图的定义。用户可采用下列方法替换视图:

#### ■ 删除并重建视图。

**警告** 当视图被删除时,其被授予的相应的对象权限也从角色和用户中取消,所以重建视图时必须重新授予相应的权限。

#### ■ 使用包含“OR REPLACE”选项的 CREATE VIEW 语句重新定义视图,“OR REPLACE”选项替换视图的当前定义,保留当前安全属性。例如,假设已创建 SALE\_STAFF 视图,并且已给角色和其他用户授予了几个对象权限,现在需重新定义 SALES\_STAFF 视图以改变 WHERE 子句中指定的系号,用户可用下列语句替换 SALES\_STAFF 视图的当前版本:

```
CREATE OR REPLACE VIEW sales_staff AS
  SELECT empno, ename, deptno
  FROM emp
  WHERE deptno=30
  WITH CHECK OPTION CONSTRAINT sales_staff_cnst;
```

在替换视图之前，请考虑下列结果：

- 替换视图即替换数据字典中视图的定义，与视图有关的所有下属对象不受影响。
- 如果视图以前的定义在“CHECK OPTION”中存在约束，但新定义中不包含该约束，那么该约束即被取消。
- 所有依赖于该替换视图的视图和 PL/SQL 程序单元不可用，详情请见第 19 章的 19.7 节“管理对象从属关系”以获取 Oracle 如何管理这种从属关系的信息。

## 18.2 管理序列

序列是多用户产生的唯一整数的数据库对象，用户可用序列自动生成主关键字的值。本节介绍管理序列的不同方面，包括以下主题：

- 创建序列
- 改变序列
- 删除序列

参见 有关序列的详细信息请参阅“Oracle8i Concepts”，有关语句的语法请参阅“Oracle8i SQL Reference”。

### 18.2.1 创建序列

在自己模式中创建序列需具备“CREATE SEQUENCE”系统权限，在其他用户模式中创建序列需具备“CREATE ANY SEQUENCE”权限。

使用 CREATE SEQUENCE 语句创建序列。例如，下列语句将创建一个生成 EMP 表的 EMPNO 列的雇员号：

```
CREATE SEQUENCE emp_sequence
  INCREMENT BY 1
  START WITH 1
  NOMAXVALUE
  NOCYCLE
  CACHE 10;
```

CACHE 选项预分配一组序列号并保存在内存中，这样可以较快地访问序列号。当高速缓存器中最后一个序列号已被使用时，Oracle 将另一组数值读入高速缓存器。

如果用户有选择地高速缓存一组序列数值，那么 Oracle 可能跳过某些序列数值。例如，当某一实例非正常退出（比如当实例失败或遇到 SHUTDOWN ABORT 语句时），已经缓存起来但还没使用的序列数值会丢失，同样，已被使用但未保存的序列数值也会丢失，导入和导出之后 Oracle 可能也会跳过已缓存的序列数值，详情请见“Oracle8i Utilities”。

参见 关于 Oracle Parallel Server (Oracle 并行服务器) 如何影响高速缓存的序列数值的信息请参见“Oracle8i Parallel Server Administration, Deployment ,



and Performance”；有关高速缓存的性能信息请参见“Oracle8i Designing and Tuning for Performance”。

### 18.2.2 改变序列

如果要改变一个序列，那么用户模式中必须有该序列，否则用户必须具备“ALTER ANY SEQUENCE”系统权限。用户可改变某一序列来改变任何定义如何产生序列数值的参数，产生序列初始数值的参数除外。若要改变序列的初始数值，需删除该序列，然后再重建它。当用户在序列数值上执行 DDL 时，用户将丢失高速缓存器中的数值。

使用 ALTER SEQUENCE 语句改变序列。例如，下列语句改变 EMP\_SEQUENCE 序列：

```
ALTER SEQUENCE emp_sequence
  INCREMENT BY 10
  MAXVALUE 10000
  CYCLE
  CACHE 20;
```

### 18.2.3 删除序列

用户可在自己模式中删除任何序列，删除另一用户模式中的序列用户必须具备“DROP ANY SEQUENCE”系统权限。如果序列不再有用，可使用 DROP SEQUENCE 语句删除它。例如，下列语句删除 ORDER\_SEQ 序列：

```
DROP SEQUENCE order_seq;
```

当序列被删除时，它的定义也从数据字典中删除，该序列的任何别名单元仍保留，但当被调用时会返回一条错误信息。

## 18.3 管理别名单元

别名单元 (synonym) 是模式对象的别名，别名单元通过隐藏对象的名称和所有者以及为分布式数据库的远程对象提供定位透明度来提供一安全层。同样，别名单元使用方便，减少了数据库用户使用 SQL 语句的复杂性。

别名单元允许潜在对象重命名或移动，此时只需别名单元重定义，基于别名单元的应用程序继续执行，无需做任何改动。

用户可创建公共别名单元和私有别名单元，公共别名单元由名为 PUBLIC 的指定用户组拥有，数据库中的任何用户都可访问；私有别名单元包含在特定的用户模式中，只有该用户和其他授权用户可访问。

本节包括下列别名单元管理信息：

■ 创建别名单元

■ 删除别名单元

参见 有关别名单元的详细信息请参见“Oracle8i Concepts”，语法定义请参见“Oracle8i SQL Reference”。

### 18.3.1 创建别名单元

在自己的模式中创建别名单元，用户必须具备 **CREATE SYNONYM** 权限；在其他用户模式中创建别名单元，用户必须具备 **CREATE ANY SYNONYM** 权限；创建一公共别名单元，用户必须具备 **CREATE PUBLIC SYNONYM** 系统权限。

使用 **CREATE SYNONYM** 语句创建别名单元，潜在模式对象不必存在。下列语句创建了一个包含在 **JWARD** 模式中的 **EMP** 表上的名为 **PUBLIC\_EMP** 的别名单元：

```
CREATE PUBLIC SYNONYM public_emp FOR jward.emp;
```

### 18.3.2 删除别名单元

用户可删除自己模式中的任何别名单元。删除其他用户模式中的别名单元，必须具备 **DROP ANY SYNONYM** 系统权限；删除一公共别名单元，必须具备 **DROP PUBLIC SYNONYM** 系统权限。

使用 **DROP SYNONYM** 语句删除一个不再需要的别名单元。删除一私有别名单元，省略 **PUBLIC** 关键字；删除一公共别名单元，包含 **PUBLIC** 关键字。

例如，下列语句删除一个名为 **EMP** 的私有别名单元：

```
DROP SYNONYM emp;
```

下列语句删除一名为 **PUBLIC\_EMP** 的公共别名单元：

```
DROP PUBLIC SYNONYM public_emp;
```

当用户删除一别名单元时，它的定义也从数据字典中删除，所有与一已删除的别名单元相关的对象仍保留，但这些对象是无效的。有关删除别名单元是如何影响其它模式对象的详细信息请参阅第 19 章的 19.7 节“管理对象从属关系”。

# 19 模式对象的常规管理

本章描述了多种类型模式对象经常遇到的模式对象管理问题，包括以下几个方面：

- 在单一操作（Single Operation）中创建多表和多视图
- 模式对象重命名
- 分析表、索引和簇
- 截取表和簇
- 启用和禁用触发器
- 管理完整性约束
- 管理对象相关性
- 管理对象名字解析
- 改变数据字典的存储参数
- 列举模式对象相关信息

参见 有关本章中讨论的 SQL 语句的语法、授权和限制等详细信息请参阅“Oracle8i SQL Reference”

## 19.1 在单一操作中创建多表和多视图

用户可使用 **CREATE SCHEMA** 语句在单一操作中创建几个表和视图并给它们授权。如果用户想在单一操作中确保创建几个表和视图并授权，那么 **CREATE SCHEMA** 语句是非常有用的。如果某个单独的表、视图或授权失败，则整个语句将返回，所有对象既没创建也没授权。

值得一提的是，**CREATE SCHEMA** 语句可包含 **CREATE TABLE**、**CREATE VIEW** 和 **GRANT** 语句，用户必须拥有发布所含命令的权限。

下列语句创建了两个表和一个连接两个表数据的视图：

```
CREATE SCHEMA AUTHORIZATION scott
  CREATE TABLE dept (
    deptno NUMBER(3,0) PRIMARY KEY,
    dname VARCHAR2(15),
    loc VARCHAR2(25)
  )
  CREATE TABLE emp (
    empno NUMBER(5,0) PRIMARY KEY,
    ename VARCHAR2(15) NOT NULL,
    job VARCHAR2(10),
    mgr NUMBER(5,0),
    hiredate DATE DEFAULT (sysdate),
    sal NUMBER(7,2),
```

```

comm NUMBER(7,2),
deptno NUMBER(3,0) NOT NULL
CONSTRAINT dept_fkey REFERENCES dept)
CREATE VIEW sales_staff AS
SELECT empno, ename, sal, comm
FROM emp
WHERE deptno = 30
WITH CHECK OPTION CONSTRAINT sales_staff_cnst
GRANT SELECT ON sales_staff TO human_resources;

```

CREATE SCHEMA 语句不支持 Oracle 扩展 ANSI CREATE TABLE 和 CREATE VIEW 语句；CREATE SCHEMA 语句包含 STORAGE 子句。

## 19.2 模式对象重命名

用户必须拥有一个对象才能对它进行重命名，用户可对下列情况下的模式对象重命名：

- 删除和重建对象
- 使用 RENAME 语句重命名对象

如果用户删除和重建一个对象，该对象原先授予的权限将全部丢失，重建一个对象必须重新授予权限。

作为选择，可使用 RENAME 语句重命名一个表、视图或序列以及它们的私有别名单元。使用 RENAME 语句时，该对象的完整性约束、索引和授权将转给新的对象。例如，下列语句将 SALES\_STAFF 视图重命名：

```
RENAME sales_staff TO dept_30;
```

**注意** 用户不可重命名已存储的 PL/SQL 程序单元、公共别名单元、索引或簇。重命名这类对象，用户必须先删除它，再重建它。

在重命名模式对象之前，请考虑下列结果：

- 所有依赖于重命名对象的视图和 PL/SQL 程序单元将失效，下次使用之前必须重编译。
- 重命名对象的别名单元使用时将返回一条错误。

有关 Oracle 如何管理对象相关性的详细信息，请参阅 19.7 节“管理对象相关性”。

## 19.3 分析表、视图和簇

用户可分析一个表、视图或簇来收集有关数据或验证其存储格式的有效性。

亦可通过分析模式对象来采集或修改特定对象的统计数值。当发布一条 DML 语句时，相关对象的统计数值用来确定该语句的最大有效执行计划，这种优化称为“基于开销的优化 (cost-based optimization)”，统计数值存储在数据字典中。

可通过分析一个表、索引或簇来证实对象的结构。例如，在一些极少出现的情况下，比如硬件或其它系统出现故障，索引可能被中断并不能正确执行。证实该索引时，用户可确信索引中的每个实体都指向相关表的恰当的行。如果某个模式对象不可靠，用户可删除它并重

建它。

可通过分析一个表或簇来收集有关该表或簇的连锁式的行的信息，这些信息对确定用户是否有足够的更新行的空间是非常有用的。例如，该信息可显示 PCTFREE 是否正确为该表或簇赋值。

用户必须拥有某个表、视图、簇或索引才可对其进行分析，或者具有“ANALYZE ANY”系统权限。

本节将讨论以下几个方面：

- 对表、索引和簇使用统计数值
- 证实表、索引和簇
- 列举表和簇的被链住行

有关有组织的索引表（index-organized table）的信息，请参见第 13 章 13.5.4 节“分析按索引组织的表”。

参见 有关分析表、索引和簇的性能和优化的详细信息，请参阅“Oracle8i Designing and Tuning for Performance”。

### 19.3.1 对表、索引和簇使用统计数值

使用 ANALYZE 语句可获得和存储在数据字典中的表、索引或簇的物理存储特征的统计值。当 Oracle 使用基于成本的优化（cost-based optimization）为 SQL 语句存取被分析对象选择最有效的执行计划时，Oracle 可以使用这些统计值。用户也可以使用这条语句生成的统计值来编写高效率的 SQL 语句，存取被分析的对象。

用户可选择下列 ANALYZE 语句的任一子句来获得统计值：

#### ■ COMPUTE STATISTICS（计算统计值）

Oracle 计算统计值时将扫描整个对象来获得该对象的数据，该数据被 Oracle 用来计算该对象的精确的统计值。这些统计值说明该对象的细微的差别。由于扫描整个对象来获得统计信息，所以对象越大，越需要做更多的工作来获得必要的信息。

#### ■ ESTIMATE STATISTICS（评估统计值）

Oracle 评估统计值时从对象的各个部分收集有代表性的信息，这些信息的子集为该对象提供了合理的、可评估的统计值，评估统计值的精确性依赖于 Oracle 所采用的实例的代表性。由于只是扫描对象的一部分来获取评估统计值信息，所以可快速分析对象。评估时用户可选择地指定 Oracle 将使用的行数或百分数。

**注意** 为表或簇计算统计值时，执行计算操作所需的临时空间的大小与指定的行数有关。对 COMPUTE STATISTICS（计算统计值）来说，需要足够的临时空间保留和分类整个表以及每一行所需的少部分系统开销；对 ESTIMATE STATISTICS（评估统计值）来说，需要足够的临时空间保留和分类所有行的实例以及每一行所需的少量的系统开销；对索引来说，分析时不需要临时空间。

#### 使用 ANALYZE 语句计算统计值

下面语句用来计算 EMP 表的统计值：

```
ANALYZE TABLE emp COMPUTE STATISTICS;
```

下面查询用来评估 EMP 表的统计值，使用 1064 行缺省统计实例：

```
ANALYZE TABLE emp ESTIMATE STATISTICS;
```

包含 SAMPLE 选项和 ESTIMATE STATISTICS 选项指定 Oracle 将使用的统计样本。用户可指定一个整数来指明行数或索引值，或者表中所有行或索引值的百分比。下面语句显示了每个选项的例子：

```
ANALYZE TABLE emp
  ESTIMATE STATISTICS
  SAMPLE 2000 ROWS;
```

```
ANALYZE TABLE emp
  ESTIMATE STATISTICS
  SAMPLE 33 PERCENT;
```

无论是哪种情况，如果用户指定的百分数大于 50，或者指定的行数和索引值大于对象中总行数或索引值的 50%，那么 Oracle 将精确计算统计值，而不是评估统计值。

如果使用 ANALYZE 语句时数据字典中已包含指定对象的统计值，那么新的将替代数据字典中旧的统计值。

#### 何种统计值被聚集

本节列举了为表、索引和簇聚集的统计值。

**注意** 符号 “\*” 表明计算统计值时，数值总是一个精确值。

表统计值 (Table Statistics)：

- 行数
- 已使用的块数\*
- 未使用的块数
- 平均有效空闲空间
- 被链锁住的行数
- 行的平均长度
- 列的不同值的个数
- 列的最低值
- 列的最高值

**注意** 与某表相关联的所有索引的统计值在分析该表时自动聚集。

索引统计值 (Index Statistics)

- 索引级 (index level) \*
- 页块 (leaf blocks) 数
- 显式关键字 (distinct keys) 数
- 页块/关键字平均数
- 数据块/关键字平均数
- 簇化因子 (clustering factor)

**注意** 如果用户对已标识不可用的索引使用 ANALYZE 语句，将得到一条错误信息。当用

户分析某个表时，Oracle 也为该表的每个索引收集统计值，所以任何一个索引已标识为不可用，那么用户将收到一条错误信息。为了使 ANALYZE 语句正确执行，用户必须删除已标识为不可用的索引，再重建它，或者指定一条“FOR”语句使得分析时跳过该索引统计值的收集。

簇统计值（Cluster Statistics）

簇可聚集的唯一统计值是平均簇关键字链长度，簇中表及与簇表相关联的索引（包括簇关键字索引）的统计值在分析簇统计值时自动聚集。

查看对象统计值

无论对象的统计值是计算的或评估的，其值存储在数据字典中。可使用下列数据字典视图查询统计值：

| 视图               | 描述   |
|------------------|--|
| USER_INDEXES     | 这个视图包含数据库中的索引的描述，包括由 ANALYZE 聚集的索引统计值。视图的类型（USER、ALL、DBA）决定了哪些索引实体将被显示 |
| ALL_INDEXES      |  |
| DBA_INDEXES      |  |
| USER_TABLES      | 这个视图包含数据库中关系表的描述，包括由 ANALYZE 聚集的表统计值                                   |
| ALL_TABLES       |  |
| DBA_TABLES       |  |
| USER_TAB_COLUMNS | 这个视图包含数据库中表、索引和簇的列的描述，包括由 ANALYZE 积聚的统计值                               |
| ALL_TAB_COLUMNS  |  |
| DBA_TAB_COLUMNS  |  |

-- 注意 这些视图中的行只包含用户已聚集统计值的表、索引和簇的统计列中的实体。  
每当用户分析某一对象时，这些实体都会为该对象而更新。

参见 有关包含统计的数据字典视图的详细信息，请参阅“Oracle8i Reference”。

删除模式对象的统计信息

用户可使用带有 DELETE STATISTICS 选项的 ANALYZE 语句从数据字典中删除表、索引或簇的统计信息。例如，如果用户不想为某个对象采用基于成本的优化，用户可能想删除该对象的统计信息。下面语句从数据字典中删除 EMP 表的统计信息：

```
ANALYZE TABLE emp DELETE STATISTICS;
```

共享 SQL 和分析统计信息

分析一个表、簇或索引可能影响当前共享 SQL 语句，这些语句当前存放在共享库中。任何时候当分析某个对象以更新或删除统计信息时，所有与被分析对象有关的共享语句都从内存中清洗，因此该语句下次执行时可以利用新的统计信息。

一些计算统计信息的可选平均数

有一些 PL/SQL 程序包允许用户有效地执行 ANALYZE 语句，这里将主要讨论这些程序包。

**DBMS\_STATS** 这是一个功能强大的程序包，它允许统计信息的聚集（包括利用并行操作）和统计信息的外部交换。统计值可存储在数据字典外的表中，可被交换而不影响优化程序。统计值可在数据库间拷贝或备份。

有关使用 **DBMS\_STATS** 程序包的信息请参阅“Oracle8i Designing and Tuning for Performance”；有关过程、语法和异常的描述请参阅“Oracle8i Supplied PL/SQL Packages Reference”。

**DBMS\_UTILITY** 这个程序包包含 **ANALYZE\_SCHEMA** 过程，该过程采用两个参数：模式名和分析方法（‘COMPUTE’，‘ESTIMATE’，或者‘DELETE’），它聚集模式中所有对象的条件信息。

有关 **DBMS\_UTILITY** 程序包的信息请参阅“Oracle8i Supplied PL/SQL Packages Reference”。

**DBMS\_DDL** 这个程序包包含 **ANALYZE\_OBJECT** 过程，该过程采用四个参数：对象类型（‘CLUSTER’，‘TABLE’，或者‘INDEX’）、对象的模式、对象的名称和分析方法（‘COMPUTE’，‘ESTIMATE’，或者‘DELETE’），它聚集该对象统计信息。

有关 **DBMS\_DDL** 程序包的信息请参阅“Oracle8i Supplied PL/SQL Packages Reference”。

### 19.3.2 证实表、索引和簇

使用带有 **VALIDATE STRUCTURE** 选项的 **ANALYZE** 语句验证表、索引、簇或快照的结构完整性。如果该结构是有效的，则不返回错误信息。相反，如果该结构具有破坏性，那么用户将收到一条错误信息。如果一个表、索引或簇具有破坏性，那么用户应该删除它再重建。如果快照具有破坏性，执行完全刷新操作，确保用户可纠正该问题；如果不能，删除快照并重建。

下面语句分析 **EMP** 表：

```
ANALYZE TABLE emp VALIDATE STRUCTURE;
```

用户可通过包括 **CASCADE** 选项验证一个对象和与其相关的所有对象，下面语句验证了 **EMP** 表和所有相关的索引：

```
ANALYZE TABLE emp VALIDATE STRUCTURE CASCADE;
```

### 19.3.3 列举表和簇的被链住的行（chained row）

用户可使用带有 **LIST CHAINED ROWS** 选项的 **ANALYZE** 语句考虑表或簇被链住的和迁移的行（**chained and migrated row**）。这条语句的结果存储在一个显式创建的特定的表中，用来接收由 **LIST CHAINED ROWS** 选项返回的信息。

使用 Oracle 提供的 **UTLCHAIN.SQL** 文本创建一合适的表来接收由 **ANALYZE...LIST CHAINED ROWS** 语句返回的数据，**UTLCHAIN.SQL** 文本在用户递交文本模式中创建一名为 **CHAINED\_ROWS** 的表。

**CHAINED\_ROWS** 表创建后，用户可在使用 **ANALYZE** 语句时指定它。例如，下面语句将含有 **EMP\_DEPT** 簇中链行的信息的行插入到 **CHAINED\_ROWS** 表：

```
ANALYZE CLUSTER emp_dept LIST CHAINED ROWS INTO chained_rows;
```

参见 **UTLCHAIN.SQL** 文本的名称和定位是依赖于操作系统的，参见指定操作系统的



Oracle 文档。

有关减少表或簇中被链住的行和移动行（chained and migrated rows）的数目的详细信息，请参见“Oracle8i Designing and Tuning for Performance”。

## 19.4 截断表和簇

用户可删除表中的所有行和一组簇表中的所有行，所以尽管该表（簇）仍存在，但完全是空的。例如，用户有一个包含月数据的表，在每个月的末尾，用户将数据存档后需清空它。

删除表中的所有行有以下三种选择：

### 1. 使用 DELETE 语句

用户可使用 DELETE 语句删除表中的所有行。例如，下面语句删除 EMP 表中的所有行：

```
DELETE FROM emp;
```

### 2. 使用 DROP 语句和 CREATE 语句

用户可删掉一个表，再重建它。例如，下面语句先删掉 EMP 表，再重建它：

```
DROP TABLE emp;  
CREATE TABLE emp (...);
```

### 3. 使用 TRUNCATE

用户可使用 SQL 语句 TRUNCATE 删除表中所有行。例如，下面语句截取 EMP 表：

```
TRUNCATE TABLE emp;
```

#### 19.4.1 使用 DELETE

如果使用 DELETE 语句时一个表或簇目前有许多行，那么当这些行被删除时将消耗有效的系统资源，比如 CPU 时间、重做日志空间、来自表的回退段空间（rollback segment space）以及任何相关索引所需的资源。同样，每一行被删除时，触发器（trigger）被激活。以前分配给表或簇的空间仍保留给该对象。用户可使用 DELETE 语句有选择地删除行，但 TRUNCATE 和 DROP 删掉整个对象。

#### 19.4.2 使用 DROP 和 CREATE

当删除并重建一个表或簇时，所有相连的索引、完整性约束和触发器也都被删除，所有依赖于被删除表或簇的对象变为无效，同样，该表或簇的所有授权也被删除。

#### 19.4.3 使用 TRUNCATE

使用 TRUNCATE 语句为删除表或簇的所有行提供了快速、有效的方法。TRUNCATE 语句并不产生任何返回信息，它将立即提交，TRUNCATE 语句是一 DDL 语句，不可能返回。TRUNCATE 语句并不影响任何与被删除表相关联的结构（限制和触发器）或权限，TRUNCATE 语句也指明删除该表后已分配给该表的空间是否返回给表空间。

用户可删除其关联模式中的任何表或簇。同样，任何具有 DROP ANY TABLE 系统权限的用户可删除任何模式中的表或簇。

在删除一个包含父关键字的表或簇之前，所有其它表中的相关的外部键必须禁止，自审

查限制无需禁止。

由于 TRUNCATE 语句删除表中的行，与该表关联的触发器不再被激活。同样，如果允许审计，TRUNCATE 语句并不产生与 DELETE 语句相对应的审计信息。相反，将为即将发生的 TRUNCATE 语句产生一条简单的审计信息，有关审计的详细信息请参阅第 24 章“审计数据库使用”。

不可删除散列簇（hash cluster），同样，散列簇或索引簇（index cluster）中的表也不可单个删除。如果必须删除单个簇表中的所有行，使用 DELETE 语句删除并重建该表。

TRUNCATE 语句的 REUSE STORAGE 选项或 DROP STORAGE 选项控制删除一个表或簇之后其已分配的空间是否归还给包含的表空间，缺省选项“DROP STORAGE”将目标表已分配的盘区减少到初始设置值，然后将空闲的盘区返回给系统，供其它对象使用。

另外，REUSE STORAGE 选项指明表或簇当前已分配的空间仍保留给它。例如，下面语句删除 EMP\_DEPT 簇，使得以前已分配给簇的盘区仍对以后的插入和删除操作有效：

```
TRUNCATE CLUSTER emp_dept REUSE STORAGE;
```

REUSE 或 DROP STORAGE 选项亦可应用于索引。当一个表或簇被删除时，所有关联的索引也被删除，同时也要注意一个被删除的表、簇或关联的索引的存储参数并不因其被删除而改变。

## 19.5 启用和禁用触发器

数据库的触发器（trigger）是存储在数据库中的过程，当指定条件出现时被激活，比如增加一行到表。用户可使用触发器增补 Oracle 的标准能力，提供高度用户化的数据库管理系统。例如，用户可创建一触发器来限制作用于表的 DML 操作，只允许常规事物处理时遇到的语句。

数据库触发器可与表、模式或数据库相关联，当下列情况发生时触发器被隐式激活：

- 对关联表执行 DML 语句（INSERT、UPDATE、DELETE）；
- 包含对数据库或模式中执行的对象 DDL 语句（例如，ALTER、CREATE、DROP）；
- 某特定数据库事件发生（例如，STARTUP、SHUTDOWN、SERVERERROR）。

关于可引发触发器的语句和数据库事件列表，请参见“Oracle8i SQL Reference”。

使用 CREATE TRIGGER 语句创建触发器，可将触发器定义为触发事件之前或之后（BEFORE 或 AFTER）触发，替代（INSTEAD OF）它时触发。下面语句在 SCOTT.EMP 表上创建了一个名为 SCOTT.EMP\_PERMIT\_CHANGES 的触发器，在任何特定语句执行前，该触发器将被触发。

```
CREATE TRIGGER scott.emp_permit_changes
  BEFORE
  DELETE OR INSERT OR UPDATE
  ON scott.emp
.
pl/sql block
```

用户可使用 DROP TRIGGER 语句将一触发器从数据字典中删除。

创建和使用触发器的信息包含在“Oracle8i Application Developer's Guide - Fundamentals”中。

触发器有两种确定的状态：

**启用** 如果遇到触发语句和触发限制是 **TRUE**，那么一个启用的触发器将执行它的触发体。缺省情况下，触发器第一次创建时是可启用的。

**禁用** 一个禁用的触发器不执行它的触发体，即使遇到可触发语句并且触发限制（如果有）是 **TRUE**。

使用 **ALTER TABLE** 语句启用或禁用一个触发器时，用户必须拥有该表，对该表有 **ALTER** 对象权限，或者有 **ALTER ANY TABLE** 的系统权限。使用 **ALTER TRIGGER** 语句启用或禁用一个独立的触发器时，用户必须拥有该触发器或有 **ALTER ANY TRIGGER** 系统权限。

**参见** 关于触发器的描述参见“Oracle8i Concepts”；关于用来创建和管理触发器的 SQL 语句的语法、限制、特定授权需求参见“Oracle8i SQL Reference”。

### 19.5.1 启用触发器

使用带有 **ENABLE** 选项的 **ALTER TRIGGER** 语句允许一个禁用的触发器。允许 **INVENTORY** 表上的名为 **REORDER** 的已禁用的触发器，键入下列语句：

```
ALTER TRIGGER reorder ENABLE;
```

允许为某一特定表定义所有触发器，使用带有 **ENABLE ALL TRIGGERS** 选项的 **ALTER TABLE** 语句。允许为 **INVENTORY** 表定义的所有触发器，键入下列语句：

```
ALTER TABLE inventory
  ENABLE ALL TRIGGERS;
```

### 19.5.2 禁用触发器

如果下列任一条件为真，用户可能想临时禁用一触发器：

- 触发器涉及的对象不可用；
- 用户不得不加载大量数据，希望加快处理而不激活触发器；
- 用户将数据加载到某表，该触发器应用于该表。

使用带有 **DISABLE** 选项的 **ALTER TRIGGER** 语句禁用触发器。禁用 **INVENTORY** 表上的 **REORDER** 触发器，键入下列语句：

```
ALTER TRIGGER reorder DISABLE;
```

使用带有 **DISABLE ALL TRIGGERS** 选项的 **ALTER TABLE** 语句可同时禁用与某表相关的所有触发器。例如，禁用所有为 **INVENTORY** 表定义的触发器，输入下列语句：

```
ALTER TABLE inventory
  DISABLE ALL TRIGGERS;
```

## 19.6 管理完整性约束

完整性约束是规则，限制表中一列或多列的取值。约束子句既可出现在 **CREATE TABLE**

语句中，也可出现在 **ALTER TABLE** 语句中，表明受约束影响的列和约束条件。

本节包括：

- 完整性约束状况
- 定义中设置完整性约束
- 修改已存在的完整性约束
- 延迟约束检查
- 管理有关联索引的约束
- 删除完整性约束
- 报告异常约束

参见 本书主要讨论约束的基本概念，识别用来定义和管理完整性约束的 SQL 语句。

有关完整性约束全面的讨论，参见 “Oracle8i Concepts”；有关应用程序中使用完整性的详细信息和实例，参见 “Oracle8i Developer’s Guide Fundamentals”。

### 19.6.1 完整性约束状态

用户可指明约束是可启用的 (**ENABLE**) 或禁用的 (**DISABLE**)。如果约束是可启用的，当数据库数据输入或修改时将检查该数据，不符合约束规则的数据被阻止输入数据库。如果约束是禁用的，那么不符合规则的数据也允许输入数据库。

另外，用户可以指定表中已存在的数据必须符合约束 (**VALIDATE**)。相反地，如果用户指定 **NOVALIDATE**，那么无法确保已存在数据符合约束。

表中定义的完整性约束可以是下列几种状态之一：

- **ENABLE, VALIDATE**
- **ENABLE, NOVALIDATE**
- **DISABLE, VALIDATE**
- **DISABLE, NOVALIDATE**

有关这些状态的意义和执行的结果，参见 “Oracle8i SQL Reference”。本节将讨论部分结果。

#### 禁用约束

为实施完整性约束定义规则，该约束应总是启用的。然而用户可能会因下列原因临时禁用完整性约束：

- 加载大量数据到一表时；
- 执行是一表发生巨大变化的批处理操作时（例如，改变每个雇员的编号，在现有值上增加 1000）；
- 导入或导出一表。

针对这三种情况，临时禁用完整性约束有助于提高操作性能，特别是在数据仓库配置中。

有这种可能性，当约束是禁用时可能键入违反约束的数据。因此，用户每次完成上述三种操作时都应启用约束。

## 启用约束

当约束被启用时，任何违反约束的行都不可能插入表中。但是，当约束被禁用时，这样的行便可插入表中；这样的行称为约束的例外。如果约束是在 `ENABLE NOVALIDATE` 状态，当约束被禁用时违法输入的数据仍保留。为了使约束处于 `VALIDATE` 状态，违反约束的行必须修改或删除。

当用户企图启用约束时，可表示一指定完整性约束的例外，参见 19.6.7 节“报告约束例外”。所有违反约束的行将被放到 `EXCEPTIONS` 表中，用户可检验它。

## 启用 NOVALIDATE 约束状态

当约束的状态是 `ENABLE NOVALIDATE` 时，所有后来的语句必须接受与约束一致的检查，但是，表中已存在的数据不接受检查。带有 `ENABLE NOVALIDATE` 约束的表可包含无效数据（invalid data），但不可能再增加新的无效数据（invalid data）给它。启用 `NOVALIDATE` 状态的约束在数据仓库配置中非常有用，这些配置可加载有效的 `OLTP` 数据。

启用约束无需经过确认。不经确认启用一约束比启用并确认一约束快。同样，确认一已启用的约束在确认过程中不需任何 `DML` 锁（lock）（不同于确认一以前已禁用的约束）。可以保证在确认过程不会有任何违法现象发生，因此，无需确认的启用可使用户明显地减少与启用约束相关联的下载时间。

## 完整性约束状态：过程和利益

按下面顺序使用完整性约束可确保最佳利益：

1. 禁用状态（disable state）；
2. 执行操作（加载、导出、导入）；
3. 启用非法状态（enable novalidate state）；
4. 启用状态（enable state）。

按这种顺序使用约束带来的利益有：

- 不保留任何锁；
- 所有约束当前都可成为“启用”状态；
- 并行启用约束；
- 允许目前表上的活动。

## 19.6.2 在定义中设置完整性约束

当一完整性约束在 `CREATE TABLE` 语句或 `ALTER TABLE` 语句中定义时，它可被启用、禁用、验证或不验证，这取决于用户是否指明 `ENABLE/DISABLE` 子句。如果不在约束定义中指明 `ENABLE/DISABLE` 子句，那么 Oracle 将自动启用和验证约束。

## 在定义中禁用约束

下面 `CREATE TABLE` 和 `ALTER TABLE` 语句定义和禁用一完整性约束：

```
CREATE TABLE emp (  
    empno NUMBER(5) PRIMARY KEY DISABLE, ...;
```

```
ALTER TABLE emp
  ADD PRIMARY KEY (empno) DISABLE;
```

定义和禁用完整性约束的 `ALTER TABLE` 语句从不会出故障，因为表中的行检验完整性约束。由于约束的规则没有实施，允许定义约束。

有关约束例外的信息，参见 19.6.7 节“报告约束例外”。

#### 在定义中启用约束

下面 `CREATE TABLE` 和 `ALTER TABLE` 定义和启用了一完整性约束：

```
CREATE TABLE emp (
  empno NUMBER(5) CONSTRAINT emp.pk PRIMARY KEY, ...;
ALTER TABLE emp
  ADD CONSTRAINT emp.pk PRIMARY KEY(empno);
```

由于表中的行可能检验完整性约束，定义和企图启用一完整性约束的 `ALTER TABLE` 可能会失败。在这种情况下，该语句将返回，约束定义不被保存，也不能启用。

### 19.6.3 修改已有完整性约束

使用 `ALTER TABLE` 语句启用、禁用和修改约束。

#### 禁用已启用的约束

下列语句禁用完整性约束：

```
ALTER TABLE dept
  DISABLE CONSTRAINT dname_ukey;
```

```
ALTER TABLE dept
  DISABLE PRIMARY KEY,
  DISABLE UNIQUE (dname, loc);
```

下列语句启用无需证实的已禁用的完整性约束：

```
ALTER TABLE dept
  ENABLE NOVALIDATE CONSTRAINT dname_ukey;
ALTER TABLE dept
  ENABLE NOVALIDATE PRIMARY KEY,
  ENABLE NOVALIDATE UNIQUE (dname, loc);
```

下列语句启用或证实已禁用的约束：

```
ALTER TABLE dept
  MODIFY CONSTRAINT dname_key VALIDATE;
ALTER TABLE dept
  MODIFY PRIMARY KEY ENABLE NOVALIDATE;
```

下列语句启用已禁用的完整性约束：

```
ALTER TABLE dept
  ENABLE CONSTRAINT dname_ukey;
ALTER TABLE dept
```

```
ENABLE PRIMARY KEY,  
ENABLE UNIQUE (dname, loc);
```

使用 **DISABLE** 或 **DROP** 子句的 **CASCADE** 选项禁用或删除 **UNIQUE KEY**（唯一关键字）或 **PRIMARY KEY**（主键）约束以及某单一步骤中所有相关的 **FOREIGN KEY**（外部键）约束。例如，下面语句禁用一 **PRIMARY KEY** 约束和任何依赖于它的 **FOREIGN KEY** 约束。

```
ALTER TABLE dept  
DISABLE PRIMARY KEY CASCADE;
```

#### 19.6.4 延迟约束检验

Oracle 检验约束时，如果约束不满意将显示一条错误。用户可延迟约束检验直到事物处理结束。

用户处理 **SET CONSTRAINTS** 语句时，**SET CONSTRAINTS** 方式将持续整个事务处理过程或另一个 **SET CONSTRAINTS** 语句重新设置模式。

**注意** 用户不可在触发器中处理 **SET CONSTRAINT** 语句。

##### 将所有约束延迟

在用来操纵数据的应用程序中，用户在实际开始处理任何数据之前必须将所有约束延迟。使用下面 **DML** 语句将所有可延迟的约束延迟：

```
SET CONSTRAINTS ALL DEFERRED;
```

**注意** **SET CONSTRAINTS** 语句只应用于当前事务处理。只要约束存在，创建约束时指定的缺省值一直保持。**ALTER SESSION SET CONSTRAINTS** 语句只应用于当前会话。

##### 检查提交（可选）

用户可通过仅在发布 **COMMIT** 前发布 **SET CONSTRAINTS ALL IMMEDIATE** 语句使得在提交之前检验约束违法行为。如果约束存在问题，那么该语句将失败，引起错误的约束被标识。如果约束违法时用户进行提交，那么事务处理将返回，用户将收到一条错误信息。

#### 19.6.5 管理与索引关联的约束

当用户创建一 **UNIQUE** 或 **PRIMARY** 关键字时，Oracle 将检验是否有已存在的索引可用来执行约束的唯一性。如果没有这些索引，Oracle 将创建一个。

当 Oracle 使用一唯一索引执行约束时，与唯一索引关联的约束将被删除或禁用，索引被删除。

当一已启用的外部键引用一 **PRIMARY** 关键字或 **UNIQUE** 关键字时，用户不可禁用或删除该 **PRIMARY** 或 **UNIQUE** 关键字或该索引。

**注意** 可延迟 **UNIQUE** 和 **PRIMARY** 关键字必须使用非唯一索引。

#### 19.6.6 删除完整性约束

如果一完整性约束执行的规则不再为真或者约束不再需要时，用户可删除它。使用带有 **DROP** 子句的 **ALTER TABLE** 语句删除约束，下面两条语句删除完整性约束：

```
ALTER TABLE dept
  DROP UNIQUE (dname, loc);
```

```
ALTER TABLE emp
  DROP PRIMARY KEY,
  DROP CONSTRAINT dept_fkey;
```

删除 **UNIQUE** 关键字和 **PRIMARY** 关键字时也就删除了相关联的唯一索引，同样，如果 **FOREIGN KEY**（外部键）引用一 **UNIQUE** 或 **PRIMARY** 关键字，用户必须在 **DROP** 语句中包含 **CASCADE CONSTRAINTS** 子句，否则用户不可能删除该约束。

## 19.6.7 报告约束例外

如果约束有效时存在例外，那么将返回一条错误，完整性约束保持无效。当一条语句由于存在完整性约束例外不能成功执行时，该语句将返回。如果例外存在，直到所有例外被修改或删除用户方可证实约束。

不可使用 **CREATE TABLE** 语句来确定哪些行是违例的。使用 **ENABLE** 子句中带有 **EXCEPTIONS** 选项的 **ALTER TABLE** 语句来确定哪些行违反完整性约束，**EXCEPTIONS** 选项将所有例外行的 **ROWID**、表的所有者、表名和约束名放如一指定表中。

**注意** 用户在启用约束之前必须创建一适当的例外报告表来接收来自 **ENABLE** 子句的 **EXCEPTIONS** 选项的信息，用户可通过提交 **UTLEXCPT.SQL** 文本创建一例外表，该文本创建了一名为 **EXCEPTIONS** 的表，用户也可通过修改和重提交文本创建另一个例外表。

**UTLEXCPT.SQL** 文本的准确名称和定位是由操作系统指定的。有关更多的信息，参见特定操作系统的 Oracle 文档。

下列语句试图证实 **DEPT** 表的 **PRIMARY KEY**（主键），如果出现例外，有关信息将插入名为 **EXCEPTIONS** 的表：

```
ALTER TABLE dept ENABLE PRIMARY KEY EXCEPTIONS INTO exceptions;
```

如果 **DEPT** 表中存在重复主键并且 **PRIMARY KEY**（主键）约束的名字是 **SYS\_C00610**，那么下列行可能会由上面的语句放入 **EXCEPTIONS** 表中：

```
SELECT * FROM exceptions;
ROWID                                OWNER      TABLE_NAME  CONSTRAINT
-----                                -
AAAAZ9AABAAABvqAAB      SCOTT      DEPT         SYS_C00610
AAAAZ9AABAAABvqAAGSCOTT      DEPT         SYS_C00610
```

一条可提供信息的查询将联合一例外报告表中的行和主表（**master table**）来显示实际违反指定约束的行，正如下面例子中所显示的：

```
SELECT deptno, dname, loc FROM dept, exceptions
  WHERE exceptions.constraint = 'SYS_C00610'
 AND dept.rowid = exceptions.row_id;
```

```
DEPTNO      DNAME      LOC
-----
```



|    |            |          |
|----|------------|----------|
| 10 | ACCOUNTING | NEW YORK |
| 10 | RESEARCH   | DALLAS   |

所有违反约束的行必须修改或从包含该约束的表中删除。修改例外时，必须将违反约束的值改为与约束一致的值或为空（NULL）。修改或删除主表（master table）中的行之后，例外报告中相应的行也应该删除，以免与以后的例外引起混乱。修改主表和例外报告表的语句应在同一事务处理中，以确保事务的一致性。

用户可发布以下事务处理来纠正前面例子中的例外：

```
UPDATE dept SET deptno = 20 WHERE dname = 'RESEARCH';
DELETE FROM exceptions WHERE constraint = 'SYS_C00610';
COMMIT;
```

管理例外时，总的目标是删除例外报告表中的所有例外。

**注意** 当用户纠正具有禁用约束表中的例外时，其他用户可能发布创建新的例外的语句。用户可通过在开始删除例外之前启用约束来避免这种情况的发生。

**参见** 有关 EXCEPTIONS 表的详情，参见 “Oracle8i Reference”。

## 19.7 管理对象相关性

本节描述不同的对象相关性，包括下面几个方面：

- 手动重编译视图；
- 手动重编译过程和函数；
- 手动重编译程序包。

**首先** 阅读表 19-1，该表显示了对象是如何受其依赖的其它对象的影响的。

表 19-1 影响对象状态的操作

| 操作   | 对象结果状态从属                         | 对象的结果状态                               |
|--|----------------------------------|---------------------------------------|
| CREATE 表、序列、别名单元                           | VALID（如果没错误）                     | 没变化 <sup>1</sup>                      |
| ALTER 表（ADD 列、MODIFY 列）RENAME 表、序列、别名单元、视图 | VALID（如果没错误）                     | INVALID                               |
| DROP 表、序列、别名单元、视图、过程、函数、程序包                | 没有，对象被删除                         | INVALID                               |
| CREATE 视图、过程 <sup>2</sup>                  | VALID（如果没错）；INVALID（如果有语法或授权错误）； | 没变化                                   |
| CREATE 或 REPLACE 视图或过程                     | VALID（如果没错）；INVALID（如果有语法或授权错误）  | INVALID                               |
| REVOKE 对象权限 <sup>3</sup> ON 对象 TO/FROM 用户  | 没变化                              | 从属于该对象的用户的所有对象都是 INVALID <sup>3</sup> |

(续表)

| 操作   | 对象结果状态从属 | 对象的结果状态                                       |
|--|----------|---|
| REVOKE 对象权限 <sup>3</sup> ON 对象<br>TO/FROM PUBLIC | 没变化      | 数据库中所有从属于<br>该对象的对象都是<br>INVALID <sup>3</sup> |
| REVOKE 系统权限 <sup>4</sup> TO/FROM 用<br>户          | 没变化      | 用户的所有对象都是<br>INVALID <sup>4</sup>             |
| REVOKE 系统权限 <sup>4</sup> TO/FROM<br>PUBLIC       | 没变化      | 数据库中的所有对象<br>都是 INVALID <sup>4</sup>          |

- 1 如果开始时对象不存在,可能会导致从属对象 INVALID;
- 2 单机过程和函数、程序包以及触发器;
- 3 只是 DML 对象权限, 包括 SELECT、INSERT、UPDATE、DELETE 和 EXECUTE, 证实操作无需重编译;
- 4 只是 DML 对象权限, 包括 SELECT、INSERT、UPDATE、DELETE ANY TABLE 和 EXECUTE ANY TABLE, 证实操作无需重编译。

当一个无效的视图或 PL/SQL 程序单元下次被使用时, Oracle 会自动地重编译它。另外, 用户可使用带有 COMPILE 子句的适当的 SQL 语句强迫 Oracle 重编译一视图或程序单元。当一从属视图或程序单元无效时, 经常使用强制编译来测试错误, 但当前不用。这种情况下, 只有执行视图或程序单元时才会自动重编译。为了标识无效从属对象, 应查询视图 USER\_/ALL\_/DBA\_OBJECTS。

### 19. 7. 1 手动重编译视图

用户必须具有 ALTER ANY TABLE 的系统权限或视图包含在自己的模式中方可对视图进行手动重编译。使用带有 COMPILE 子句的 ALTER VIEW 语句重编译视图。下面语句在用户自己的模式中重编译 EMP\_DEPT 视图:

```
ALTER VIEW emp_dept COMPILE;
```

### 19. 7. 2 手动重编译过程和函数

用户必须具有 ALTER ANY PROCEDURE 的系统权限, 或过程包含在自己的模式中, 方可手动重编译单机过程, 使用有 COMPILE 子句的 ALTER PROCEDURE/FUNCTION 语句重编译单机过程或函数。下面语句重编译包含在用户自己模式中的过程 UPDATE\_SALARY:

```
ALTER PROCEDURE update_salary COMPILE;
```

### 19. 7. 3 手动重编译程序包

用户必须具备 ALTER ANY PROCEDURE 的系统权限, 或程序包包含在用户自己的模式中, 方可手动重编译程序包。使用带有 COMPILE 子句的 ALTER PACKAGE 语句重编译程序包主体或程序包说明和主体。下面语句分别重编译 ACCT\_MGMT 的主体、主体以及说明:

```
ALTER PACKAGE acct_mgmt COMPILE BODY;
```

ALTER PACKAGE acct\_mgmt COMPILE PACKAGE;

## 19.8 管理对象名称解析

本节描述 Oracle 如何解析对象的名称。

1. 首先, Oracle 视图证明相关 SQL 语句中名称的第一部分具有资格。比如, 在 SCOTT.EMP 中, SCOTT 是第一部分。如果只有一片段, 那么该片段被认为是第一部分。

a. Oracle 在当前模式中搜索对象名称与第一部分相符合的对象, 如果没找到这样的对象, 继续到步骤 b;

b. 如果在当前模式中没发现模式对象, Oracle 搜索与名称第一部分匹配的公共别名单元。如果仍没找到, 继续到步骤 c;

c. 如果没找到公共别名单元, Oracle 寻找与对象名称第一部分匹配的模式。如果找到一个, 返回到步骤 b, 现在使用名称的第二部分在合格的模式中寻找相应的对象。如果在合格模式中找不到与第二部分匹配的对象或根本不存在第二部分, Oracle 返回一条错误信息。

如果在步骤 c 中没找到任何模式, 该对象不具有资格, Oracle 将返回一条错误信息。

2. 模式对象已具有资格, 名称的其他部分必须匹配已找到对象的有效部分。例如, 如果名称是 SCOTT.EMP.DEPTNO, SCOTT 被认为是模式, EMP 被认为是表, 则 DEPTNO 必须与某列一致 (因为 EMP 是表); 如果 EMP 被认为是程序包, 则 DEPTNO 必须与程序包的公共常量、变量、过程或函数一致。

分布式数据库使用全局对象名称时 (无论是显式地还是隐含在别名单元中), 局部 Oracle 局部地解析相关的对象名称。例如, 它解析一别名单元到远程表的全局对象名称。局部解析语句被发送到远程数据库, 远程数据库如上面描述那样完成对象名称解析。

## 19.9 为数据字典改变存储参数

本节描述了改变数据字典存储参数, 包括以下几个方面:

■ 数据字典中的结构;

■ 要求改变数据字典存储的错误信息

如果数据库很大或者包含大量的对象, 那么表中的列、约束定义、用户或其它定义、构成数据字典的表可能在某一时刻无法得到额外的盘区。例如, 某数据字典表可能需要一额外的盘区, 但在 SYSTEM 表空间里没有足够的连续的空间, 如果发生这种情况, 用户无法创建新的对象, 尽管计划容纳这些对象的表空间似乎有足够的空间。为纠正这种状况, 用户可从属数据字典表中存储参数, 允许它们分配到更多的盘区, 同样地用户可改变为用户创建段 (segment) 设置的存储参数。例如, 可调整数据字典表的 NEXT 或 PCTINCREASE 的值。

**警告** 改变数据字典对象设置的存储参数需要特别小心, 如果用户选择不适当的设置, 可能会损害数据字典的结构, 被迫重建整个数据库。例如, 如果将数据字典表 USER\$ 的 PCTINCREASE 设置为 0、NEXT 设置为 2K, 该表将快速达到一个段的盘区的最大值, 如果不导出、重建和导入整个数据库, 则无法创建任何更多的用户或规则。

### 19.9.1 数据字典中的结构

下面表和簇包含了数据库中所有用户创建的对象定义:

|          |  |
|----------|--|
| SEG\$    | 数据库中定义的段 (包括临时段)                                 |
| OBJ\$    | 数据库中用户定义的对象 (包括簇表); 索引为 I_OBJ1 和 I_OBJ2          |
| UNDO\$   | 数据库中定义的回退段; 索引为 I_UNDO1                          |
| FET\$    | 未分配给任何段的有效空闲盘区                                   |
| UET\$    | 已分配给段的盘区   |
| TS\$     | 数据库中定义的表空间                                       |
| FILE\$   | 构成数据库的文件; 索引为 I_FILE1                            |
| FILEXT\$ | 带 AUTOEXTEND 选项设置的数据文件                           |
| TAB\$    | 数据库中定义的表 (包括簇表); 索引为 I_TAB1                      |
| CLU\$    | 数据库中定义的簇   |
| IND\$    | 数据库中定义的索引; 索引为 I_IND1                            |
| ICOL\$   | 已定义索引的列 (包括组合索引中每一列的专用实体); 索引为 I_ICOL1           |
| COL\$    | 数据库表中定义的列; 索引为 I_COL1 和 I_COL2                   |
| CON\$    | 数据库中定义的约束 (包括约束所有者的信息); 索引为 I_CON1 和 I_CON2      |
| CDEF\$   | CON\$中约束的定义; 索引为 I_CDEF1、I_CDEF2 和 I_CDEF3       |
| CCOL\$   | 有约束的列 (包括组合键中每一列的专用实体); 索引为 I_CCOL1              |
| USER\$   | 数据库中定义的用户和规则; 索引为 I_USER1                        |
| TSQ\$    | 用户表空间定额 (包含给每个用户定义的每个表空间的一个实体)                   |
| C_OBJ#   | 包含 TAB\$、CLU\$、ICOL\$、IND\$和 COL\$的簇; 索引为 I_OBJ# |
| C_TS#    | 包含 FET\$、TS\$和 FILE\$; 索引为 I_TS#                 |
| C_USER#  | 包含 USER 和 TSQ\$\$的簇; 索引为 I_USER#                 |
| C_COBJ#  | 包含 CDEF\$和 CCOL\$的簇, 索引为 I_COBJ#                 |

对所有数据字典段, 下面是一些最有可能要求改变的参数:

|                |                |
|----------------|----------------|
| C_TS#          | 如果数据库空闲空间非常碎   |
| C_OBJ#         | 如果表中有很多索引或列    |
| CON\$, C_COBJ# | 如果过多地使用完整性约束   |
| C_USER#        | 如果数据库中定义了大量的用户 |

对于簇表必须改变为簇设置的而不是为表设置的存储参数。

### 19.9.2 要求改变数据存储参数的错误信息

如果用户试图创建一新的对象, 该对象需要 Oracle 分配一额外的盘区给数据字典而此时已不可能分配额外的盘区, 那么 Oracle 将返回一条错误信息。错误信息 ORA-1653 “failed to allocate extent of size num in tablespace ‘name’” (不能在 ‘name’ 表空间分配大小为 ‘size’ 的盘区) 指出了问题所在。

如果用户收到这条错误信息, 并且用户试图改变的段 (比如表或回退段) 没有达到其定义中指定的界限, 那么检查一下对象定义中设置的存储参数。

例如，如果用户试图在一表上定义一新的 **PRIMARY KEY**（主键）约束时收到 **ORA-1547** 信息，并且有足够的空间给 **Oracle** 为该主键必须创建的索引，那么检查一下 **CON\$**或 **C\_COBJ#** 是否可分配给别的盘区；这样做时，查询 **DBA\_SEGMENTS** 并且考虑改变 **CON\$**或 **C\_COBJ#** 存储参数。

详情参见 19.10.7 节 “例 7：显示不能分配附加盘区的段”。

## 19.10 显示有关模式对象的信息

数据字典提供了许多本书中描述的关于模式对象的视图。下面列出与模式对象相关联的视图：

- **ALL\_OBJECTS**, **USER\_OBJECTS**, **DBA\_OBJECTS**
- **ALL\_CATALOG**, **USER\_CATALOG**, **DBA\_CATALOG**
- **ALL\_TABLES**, **USER\_TABLES**, **DBA\_TABLES**
- **ALL\_TAB\_COLUMNS**, **USER\_TAB\_COLUMNS**, **DBA\_TAB\_COLUMNS**
- **ALL\_TAB\_COMMENTS**, **USER\_TAB\_COMMENTS**
- **ALL\_COL\_COMMENTS**, **USER\_COL\_COMMENTS**, **DBA\_COL\_COMMENTS**
- **ALL\_VIEWS**, **USER\_VIEWS**, **DBA\_VIEWS**
- **ALL\_INDEXES**, **USER\_INDEXES**, **DBA\_INDEXES**
- **ALL\_IND\_COLUMNS**, **USER\_IND\_COLUMNS**, **DBA\_IND\_COLUMNS**
- **USER\_CLUSTERS**, **DBA\_CLUSTERS**
- **USER\_CLU\_COLUMNS**, **DBA\_CLU\_COLUMNS**
- **ALL\_SEQUENCES**, **USER\_SEQUENCES**, **DBA\_SEQUENCES**
- **ALL\_SYNONYMS**, **USER\_SYNONYMS**, **DBA\_SYNONYMS**
- **ALL\_DEPENDENCIES**, **USER\_DEPENDENCIES**, **DBA\_DEPENDENCIES**

下面数据字典视图包含关于数据库的段的信息：

- **USER\_SEGMENTS**
- **DBA\_SEGMENTS**

下面数据字典视图包含关于数据库盘区的信息：

- **USER\_EXTENTS**
- **DBA\_EXTENTS**
- **USER\_FREE\_SPACE**
- **DBA\_FREE\_SPACE**

另外，下面 **Oracle** 的 **PL/SQL** 程序包提供了关于对象空间使用和空闲块的信息：

| 程序包和过程                         | 描述                  |
|--------------------------------|---------------------|
| <b>DBMS_SPACE.UNUSED_SPACE</b> | 返回某对象（表、索引或簇）未使用的空间 |
| <b>DBMS_SPACE.FREE_BLOCKS</b>  | 返回某对象（表、索引或簇）中的空闲块  |

下面例子证明了显示不同模式对象的方法。

参见 有关数据字典视图的完整描述，参见 “Oracle8i Reference”；有关 PL/SQL 程序包的描述，参见 “Oracle8i Supplied PL/SQL Packages Reference”。

19. 10. 1 例 1：显示模式对象类型

下面查询列出了查询用户所有的对象：

```
SELECT object_name, object_type
FROM user_objects;
```

| OBJECT_NAME    | OBJECT_TYPE |
|----------------|-------------|
| -----          | -----       |
| EMP_DEPT       | CLUSTER     |
| EMP            | TABLE       |
| DEPT           | TABLE       |
| EMP_DEPT_INDEX | INDEX       |
| PUBLIC_EMP     | SYNONYM     |
| EMP_MGR        | VIEW        |

19. 10. 2 例 2：显示列信息

列的有关信息，比如名称、数据类型、长度、精度、规模和缺省值可通过使用带有 \_COLUMNS 后缀的视图列出。例如，下面查询语句列出了 EMP 表和 DEPT 表的所有列的缺省值：

```
SELECT table_name, column_name, data_default
FROM user_tab_columns
WHERE table_name = 'DEPT' OR table_name = 'EMP';
```

| TABLE_NAME | COLUMN_NAME | DATA_DEFAULT |
|------------|-------------|--------------|
| -----      | -----       | -----        |
| DEPT       | DEPTNO      |              |
| DEPT       | DNAME       |              |
| DEPT       | LOC         | 'NEW YORK'   |
| EMP        | EMPNO       |              |
| EMP        | ENAME       |              |
| EMP        | JOB         |              |
| EMP        | MGB         |              |
| EMP        | HIREDATE    | SYSDATE      |
| EMP        | SAL         |              |
| EMP        | COMM        |              |
| EMP        | DEPTNO      |              |

切记不是所有的列都有用户指定的缺省值，这些列自动将 NULL 作为缺省值。

19. 10. 3 例 3：显示视图和别名单元的相关性

用户创建一视图或别名单元，该视图或别名单元必定基于某基本对象。可使用 ALL/USER/DBA\_DEPENDENCIES 数据字典视图来揭示视图间的相关性，使用 ALL/USER/DBA\_SYNONYMS 数据字典视图来列出一别名单元的基本对象。例如，下面查

询语句列出了由用户 JWARD 创建的别名单元的基本对象:

```
SELECT table_owner, table_name, synonym_name
      FROM sys.dba_synonyms
      WHERE owner = 'JWARD';
```

| TABLE_OWNER | TABLE_NAME | SYNONYM_NAME |
|-------------|------------|--------------|
| SCOTT       | DEPT       | DEPT         |
| SCOTT       | EMP        | EMP          |

#### 19. 10. 4 例 4: 显示段的一般信息

下面查询语句返回了每个回退 (rollback) 段的名称、包含每个回退段的表空间和每个回退段的大小:

```
SELECT segment_name, tablespace_name, bytes, blocks, extents
      FROM sys.dba_segments
      WHERE segment_type = 'ROLLBACK';
```

| SEGMENT_NAME | TABLESPACE_NAME | BYTES  | BLOCKS | EXTENTS |
|--------------|-----------------|--------|--------|---------|
| RS1          | SYSTEM          | 20480  | 10     | 2       |
| RS2          | TS1             | 40960  | 20     | 3       |
| SYSTEM       | SYSTEM          | 184320 | 90     | 3       |

#### 19. 10. 5 例 5: 显示盘区一般信息

数据库中当前已分配的盘区的一般信息存储在数据字典视图 DBA\_EXTENTS 中。例如, 下面查询语句指出了已分配给回退段的盘区以及每个盘区的大小:

```
SELECT segment_name, bytes, blocks
      FROM sys.dba_extents
      WHERE segment_type = 'ROLLBACK';
```

| SEGMENT_NAME | BYTES | BLOCKS |
|--------------|-------|--------|
| RS1          | 10240 | 5      |
| RS1          | 10240 | 5      |
| SYSTEM       | 51200 | 25     |
| SYSTEM       | 51200 | 25     |
| SYSTEM       | 51200 | 25     |

切记 RS1 回退段由两个盘区组成, 每个 10K; SYSTEM 回退段由三个大小均为 50K 的盘区组成。

#### 19. 10. 6 例 6: 显示数据库的空闲空间 (盘区)

数据库中空闲盘区 (未分配给任何段的盘区) 的信息存储在数据字典视图 DBA\_FREE\_SPACE 中。例如, 下面查询语句通过每个表空间的空闲盘区揭示了可用空闲盘

区的数量:

```
SELECT tablespace_name, file_id, bytes, blocks
FROM sys.dba_free_space;
```

| TABLESPACE_NAME | FILE_ID | BYTES    | BLOCKS |
|-----------------|---------|----------|--------|
| SYSTEM          | 1       | 8120320  | 3965   |
| SYSTEM          | 1       | 10240    | 5      |
| TS1             | 2       | 10432512 | 5094   |

#### 19. 10. 7 例 7: 显示不能分配附加盘区的段

用户可使用 DBA\_FREE\_SPACE、DBA\_SEGMENTS、DBA\_TABLES、DBA\_CLUSTERS、DBA\_ROLLBACK\_SEGS 共同确定是否有其它的段不能为数据字典对象分配额外的盘区。

一个段不能分配给一盘区有以下几个原因:

- 包含该段的表空间没有足够的空间给下一个盘区;
- 段设置了最大盘区数, 记录在数据字典中 (在 SEG.MAX\_EXTENTS 中);
- 段有由操作系统指定的数据块大小允许的盘区的最大数。

**注意** STORAGE 子句中 MAXEXTENTS 的值可以是 UNLIMITED, 数据字典表的 MAXEXTENTS 的值不可超过可允许的块的最大值, 因此, 数据字典表不可转变为无限制的格式 (unlimited format)。

下面查询语句返回了所有符合上述标准的段的名称、所有者和表空间:

```
SELECT seg.owner, seg.segment_name,
       seg.segment_type, seg.tablespace_name,
       DECODE(seg.segment_type,
              'TABLE', t.next_extent,
              'CLUSTER', c.next_extent,
              'INDEX', i.next_extent,
              'ROLLBACK', r.next_extent)
FROM sys.dba_segments seg,
     sys.dba_tables t,
     sys.dba_clusters c,
     sys.dba_indexes i,
     sys.dba_rollback_segs r
WHERE ((seg.segment_type = 'TABLE'
       AND seg.segment_name = t.table_name
       AND seg.owner = t.owner
       AND NOT EXISTS (SELECT tablespace_name
                        FROM dba_free_space free
                        WHERE free.tablespace_name = t.tablespace_name
                        AND free.bytes >= t.next_extent))
OR (seg.segment_type = 'CLUSTER'
   AND seg.segment_name = c.cluster_name
   AND seg.owner = c.owner
   AND NOT EXISTS (SELECT tablespace_name
```



```

        FROM dba_free_space free
        WHERE free.tablespace_name = c.tablespace_name
        AND free.bytes >= c.next_extent))
OR (seg.segment_type = 'INDEX'
    AND seg.segment_name = i.index_name
    AND seg.owner = i.owner
    AND NOT EXISTS (SELECT tablespace_name
        FROM dba_free_space free
        WHERE free.tablespace_name = i.tablespace_name
        AND free.bytes >= i.next_extent))
OR (seg.segment_type = 'ROLLBACK'
    AND seg.segment_name = r.segment_name
    AND seg.owner = r.owner
    AND NOT EXISTS (SELECT tablespace_name
        FROM dba_free_space free
        WHERE free.tablespace_name = r.tablespace_name
        AND free.bytes >= r.next_extent))
OR seg.extents = seg.max_extents
OR seg.extents = data_block_size;

```

**注意** 用户使用该查询语句时可将 data\_block\_size 替换为自己系统数据块的大小。

一旦用户已标识某段不可分配额外盘区，用户可根据其原因采用下面任何一种方法解决这个问题：

- 如果表空间已满，增加数据文件到表空间；
- 如果该段有太多的盘区，而且用户不可再为段增加 MAXEXTENTS 的值，执行下列步骤：首先，导出段中的数据；其次，删除和重建段，设置一个更大的 INITIAL 值，使得段无需分配如此多的盘区；最后，将导出的数据重新导入到段中。

# 20 解决数据块讹误的问题

本章阐明了使用 DBMS\_REPAIR PL/SQL 程序包修正数据库模式对象中的数据块讹误 (data block corruption)，包括以下几个方面：

- 修正数据块讹误的选项；
- 关于 DBMS\_REPAIR 程序包；
- 使用 DBMS\_REPAIR 程序包；
- DBMS\_REPAIR 示例。

**注意** 如果用户不熟悉 DBMS\_REPAIR 程序包，建议用户在执行该程序包包含的任何修复程序时，与 Oracle 全球支持分析师 (Oracle Worldwide Support Analyst) 一起工作。

## 20.1 修正数据块讹误的选项

Oracle 提供了不同的方法用来检测和改正数据块讹误，其中一种改正方法是检测完该讹误之后删除和重建该对象，但这往往不尽人意。如果数据块讹误只限于行的子集，则另一个选项是选择所有数据（不可靠的行除外）重建该表。

另一个管理数据块讹误的方法是使用 DBMS\_REPAIR 程序包，用户可使用 DBMS\_REPAIR 程序包检测和修正表和索引中不可靠的数据块。应用这种方法，用户可访问任何可能出现的讹误，同样可继续使用这些对象直到用户想重建或修正它们。

**注意** 任何涉及数据丢失的讹误都要求分析和了解这些数据是如何适合整个数据库系统的，因此 DBMS\_REPAIR 不是一根魔杖，用户还必须明确该程序包提供的修正方法是否是最合适的工具，可以解决任何指定的讹误问题。根据修正的性质，用户可能丢失数据，引起逻辑不一致；因此用户需权衡使用 DBMS\_REPAIR 带来的利弊。

## 20.2 关于 DBMS\_REPAIR 程序包

本节描述包含在 DBMS\_REPAIR 程序包中的过程，指出这些过程使用过程中的一些限制和约束。

**参见** 有关 DBMS\_REPAIR 程序包和它的过程的完整描述，见 “Oracle8i Supplied PL/SQL Packages Reference”。

### 20.2.1 DBMS\_REPAIR 的过程

下面是构成 DBMS\_REPAIR 程序包的过程：

| 过程名                 | 描述  |
|---------------------|---|
| CHECK_OBJECT        | 检测和报告表或索引中的讹误   |
| FIX_CORRUPT_BLOCKS  | 标识不可靠的块（以前是通过 CHECK_OBJECT 过程标识）                                  |
| DUMP_ORPHAN_KEYS    | 报告指向不可靠行的索引实体   |
| REBUILD_FREELISTS   | 重建对象的空闲列表   |
| SKIP_CORRUPT_BLOCKS | 使用时，忽略表和索引扫描过程中已标识为不可靠的块。如果不使用，当用户遇到已标识为不可靠的块时将收到错误信息<br>ORA-1578 |
| ADMIN_TABLES        | 为 DBMS_REPAIR 修正和孤立关键字表提供管理函数（创建、删除、清除）                           |

注意 这些通常在 SYS 模式中创建。  
这些过程将在第 20.4 节的“DBMS\_REPAIR 示例”中进一步阐述和举例说明其用途。

20.2.2 限制和约束

- DBMS\_REPAIR 过程具有以下约束：
- 支持 LOBs 表、嵌套表和 varrays，但忽略列的输出；
  - SKIP\_CORRUPT\_BLOCKS 和 REBUILD\_FREELISTS 过程支持簇，但 CHECK\_OBJECT 过程不支持；
  - 不支持按索引组织的表和 LOB 索引；
  - DUMP\_ORPHAN\_KEYS 过程不可在位图索引或基于函数的索引上操作；
  - DUMP\_ORPHAN\_KEYS 过程只能处理长度至多为 3,950 字节的关键字。

20.3 使用 DBMS\_REPAIR 程序包

在考虑使用 DBMS\_REPAIR 程序包访问数据块讹误时建议按下列步骤操作：

步骤 1：检测和报告讹误；

步骤 2：评估使用 DBMS\_REPAIR 的成本和利润；

步骤 3：使得对象可用；

步骤 4：修正讹误，重建丢失的数据。

下面分步讨论每个步骤。

20.3.1 步骤 1：检测和报告讹误

在使用 DBMS\_REPAIR 之前，用户首先应该是检测和报告讹误。报告不但应指出块出现的问题，还应指出修正块的相关指令。除了 DBMS\_REPAIR，用户还有多种选择检测讹误。表 20-1 描述了几种不同的检测方法：

表 20-1 讹误检测方法比较

| 检测方法              | 描述   |
|-------------------|--|
| DBMS_REPAIR       | 为指定的表、分区或索引检测块，将结果填充一修正表   |
| DB_VERIFY         | 检测离线数据库的外部命令行实用程序  |
| ANALYZE           | 使用时带有 VALIDATE STRUCTURE 选项，检验索引、表或簇的结构完整性，检查或检验表和索引是否同步         |
| DB_BLOCK_CHECKING | 当初始化参数 DB_BLOCK_CHECKING=TRUE 时执行，在不可靠块实际标识之前指出它们来。当块有所改变时执行检查操作 |

#### DBMS\_REPAIR: 使用 CHECK\_OBJECT 和 ADMIN\_TABLES 过程

CHECK\_OBJECT 过程为指定对象检查和报告块讹误，类似于 ANALYZE..VALIDATE STRAUCTURE 语句，分别对索引和数据块执行块检查操作。

CHECK\_OBJECT 不但报告讹误，而且标出运行 FIX\_CORRUPT\_BLOCKS 可能遇到的定位。这些信息有助于填充一修复表，该表必须由 ADMIN\_TABLES 过程首次创建。

用户运行 CHECK\_OBJECT 过程后，修复表上的简单查询语句显示了讹误和对对象的修复指令。借助这些信息，用户评估如何最好地访问报告中的问题。

#### DB\_VERIFY: 执行脱机数据库检查

当用户遇到数据讹误问题时，通常使用 DB\_VERIFY 作为脱机诊断实用程序。

参见 有关 DB\_VERIFY 详细信息，见 “Oracle8i Utilities”。

#### ANALYZE: 讹误报告

ANALYZE TABLE...VALIDATE STRUCTURE 语句验证被分析对象的结构。如果 Oracle 成功地证实了对象的结构，将返回一条证实信息给用户。如果 Oracle 遇到对象结构中的讹误，将返回一条错误信息，这种情况下，用户应该删除和重建对象。

参见 有关 ANALYZE 更多的信息，见 “Oracle8i SQL Reference”。

#### DB\_BLOCK\_CHECKING (块检查初始化参数)

用户可设置块接受检查，例如通过 DB\_BLOCK\_CHECKING 初始化参数（缺省值是 FALSE）；当数据或索引块改变时检查它们。DB\_BLOCK\_CHECKING 是动态参数，由 ALTER SYSTEM SET 语句修改。块检查对系统表空间总处于启用状态。

参见 有关 DB\_BLOCK\_CHECKING 初始化参数的更多信息，参见 “Oracle8i Reference”。

### 20.3.2 步骤 2: 评估使用 DBMS\_REPAIR 的成本和利润

使用 DBMS\_REPAIR 之前，用户必须权衡使用过程中涉及的利益。用户还应该审查其它访问讹误对象的选项。

首先回答下列问题：

1. 讹误的程度如何？

为了确定是否存在讹误并采取修正操作，执行 CHECK\_OBJECT 过程，查询修复表。

## 2. 还有其他什么方法适合于访问块讹误？

假设数据对另一源程序有效，删除、重建和重填充该对象。另一个选择是从另一个讹误表中发出 `CREATE TABLE...AS SELECT` 语句创建一个新的对象。

用户可通过排除 `SELECT` 语句选出的讹误的行来忽略讹误。

用户可执行媒体恢复操作。

## 3. 当你使用 `DBMS_REPAIR` 过程使得一对象可用时会导致什么逻辑讹误或副作用？这些讹误可访问吗？这样做会产生什么效果？

用户不可访问标识为讹误的块中的行，但是，尽管一个块仍存在用户可有效地访问的行，它仍可被标识为讹误的块。

当块标识为讹误时相关的完整性约束被破坏。如果发生这种情况，先禁止该约束，再重新启用它；任何不一致性都将被报告。处理完所有问题之后，用户应该能重新启用该约束。

当表中定义了触发器时可能会遇到逻辑讹误问题，例如，如果重新插入行，触发器会被重新激活还是不激活？如果用户在安装时已了解触发器及其用途，问题便可迎刃而解。

空闲列表块可能无法访问，如果一讹误的块在空闲列表块的头或尾，空间管理器重新初始化空闲列表块，然后将发现有些块应该在空闲序列上，但实际情况并不是这样，用户可运行 `REBUILD_FREELISTS` 过程存取它。

索引和表可能配合不一致，可首先执行 `DUMP_ORPHAN_KEYS` 过程（从对重建讹误数据有用的关键字中获得信息），然后发出 `ALTER INDEX REBUILD ONLIE` 语句时表和索引重新一致。

## 4. 如果修复引起数据丢失，该数据能否重新找回？

当数据块被标识为讹误时，用户可从索引检索数据。`DUMP_ORPHAN_KEYS` 过程可帮助用户检索该信息，当然，这种方式下收回的数据依赖于索引和表之间的冗余量。

### 20.3.3 步骤 3：使对象可用

在这一步中，`DBMS_REPAIR` 使得对象可用，而忽略扫描表和索引过程中出现的讹误。

**讹误修正：使用 `FIX_CORRUPT_BLOCKS` 和 `SKIP_CORRUPT_BLOCKS` 过程**

用户可通过建立一个环境来使得一讹误对象可用，这个环境跳过 `DBMS_REPAIR` 过程可修复的之外的讹误。

如果讹误引起数据丢失，比如数据块中的坏行，所有这样的块用 `FIX_CORRUPT_BLOCKS` 过程标识为讹误块，然后用户执行 `SKIP_CORRUPT_BLOCKS` 过程跳过已标识为讹误的块。当设置跳跃时，表和索引扫描跳过所有讹误块。这种方法同时应用于媒体和软件讹误块。

**跳过讹误块的蕴涵**

如果表和索引配合不一致，则 `SET TRANSACTION READ ONLY` 事务处理在下面情况下可能出现不一致：即一条查询语句只探索索引，另一条查询语句探索索引和表。如果表块被标识为讹误的，则这两条查询语句将得到不同的结果，因此将打破只读事务处理的规则。一个可行的方法是使用 `SET TRANSACTION READ ONLY` 事务处理时不跳过讹误。

SELECT 被链锁住的行会遇到类似的问题，从本质上说，同一行的查询语句可能会或不会存取讹误，并依次给出了不同的结果。

#### 20.3.4 步骤 4: 修正讹误和重建丢失数据

使得一对象可用之后，执行下列修复操作。

使用 DUMP\_ORPHAN\_KEYS 过程恢复数据

DUMP\_ORPHAN\_KEYS 过程报告指向讹误数据块中的行的索引实体，所有这样的索引实体都被插入到一孤立的关键字表，该表存储了讹误的关键字和行标识。

检索索引实体信息之后，使用 ALTER INDEX REBUILD ONLINE 语句重建索引。

使用 REBUILD\_FREELISTS 过程修复空闲列表

当空闲列表的头或尾的某一块发现被标识为“讹误”，空闲列表被重新初始化，并返回一错误信息。尽管将该块从空闲列表中取出，仍会使得用户失去访问空闲列表中列于讹误块后的所有数据块。

用户可使用 REBUILD\_FREELISTS 过程重新初始化空闲列表。该对象被扫描，如果其适合于空闲列表中的块，将其添加到主空闲列表。空闲列表组以公平的方式处理，某一时刻只处理一个块。对象中任何标识为“讹误”的块在重建时都被忽略。

### 20.4 DBMS\_REPAIR 示例

本节将举例说明 DBMS\_REPAIR 过程的用途：

- 使用 ADMIN\_TABLES 建立修复表或孤立关键字表；
- 使用 CHECK\_OBJECT 过程检测讹误；
- 使用 FIX\_CORRUPT\_BLOCKS 过程修复讹误块；
- 查找指向讹误数据块的索引实体: DUMP\_ORPHAN\_KEYS；
- 使用 REBUILD\_FREELISTS 过程重建空闲列表；
- 启用或禁用跳过讹误块: SKIP\_CORRUPT\_BLOCKS。

参见 有关 DBMS\_REPAIR 过程的语法、约束和例外的更多的信息，见“Oracle8i Supplied PL/SQL Packages Reference”。

#### 20.4.1 使用 ADMIN\_TABLES 建立修复表或孤立关键字表

修复表为用户提供了接口，使得用户可知道通过 CHECK\_OBJECT 过程会发现什么讹误，如果运行 FIX\_CORRUPT\_BLOCKS 过程如何访问这些讹误块。另外，它可用来驱动 FIX\_CORRUPT\_BLOCKS 过程的执行。

运行 DUMP\_ORPHAN\_KEYS 过程时将使用孤立关键字表 (orphan key table)，该表揭示指向讹误行的索引实体。DUMP\_ORPHAN\_KEYS 过程通过在一可用方式下记录日志和提供索引信息来填充孤立关键字表。

ADMIN\_TABLE 过程用来创建、清理或删除一修复表或一孤立关键字表。

## 创建修复表

下面例子用来创建一修复表:

```
BEGIN
DBMS_REPAIR.ADMIN_TABLES (
    TABLE_NAME => 'REPAIR_TABLE'
    TABLE_TYPE => dbms_repair.repair_table,
    ACTION      => dbms_repair.create_action,
    TABLESPACE => 'USERS');
END;
/
```

对于每个修复表或孤立关键字表, 可创建一视图删除任何属于不再存在对象的行, 视图的名称必须与修复表或孤立关键字表的名称一致, 但可加上 DBA\_前缀 (例如 DBA\_REPAIR\_TABLE 或 DBA\_ORPHAN\_KEY\_TABLE)。

下面查询语句描述了由前面例子创建的修复表:

SQL> desc repair\_table

| Name                | Null?    | Type           |
|---------------------|----------|----------------|
| OBJECT_ID           | NOT NULL | NUMBER         |
| TABLESPACE_ID       | NOT NULL | NUMBER         |
| RELATIVE_FILE_ID    | NOT NULL | NUMBER         |
| BLOCK_ID            | NOT NULL | NUMBER         |
| CORRUPT_TYPE        | NOT NULL | NUMBER         |
| SCHEMA_NAME         | NOT NULL | VARCHAR2(30)   |
| OBJECT_NAME         | NOT NULL | VARCHAR2(30)   |
| BASEOBJECT_NAME     |          | VARCHAR2(30)   |
| PARTITION_NAME      |          | VARCHAR2(30)   |
| CORRUPT_DESCRIPTION |          | VARCHAR2(2000) |
| REPAIR_DESCRIPTION  |          | VARCHAR2(200)  |
| MARKED_CORRUPT      | NOT NULL | VARCHAR2(10)   |
| CHECK_TIMESTAMP     | NOT NULL | DATE           |
| FIX_TIMESTAMP       |          | DATE           |
| REFORMAT_TIMESTAMP  |          | DATE           |

## 创建孤立关键字表

下面举例说明孤立关键字表的创建:

```
BEGIN
DBMS_REPAIR.ADMIN_TABLES (
    TABLE_NAME => 'ORPHAN_KEY_TABLE'
    TABLE_TYPE => dbms_repair.orphan_table,
    ACTION      => dbms_repair.create_action,
    TABLESPACE => 'USERS');
END;
/
```

下列查询语句描述了孤立关键字表:

```
SQL> desc orphan__key_table
```

| Name           | Null?    | Type         |
|----------------|----------|--------------|
| -----          | -----    | -----        |
| SCHEMA_NAME    | NOT NULL | VARCHAR2(30) |
| INDEX_NAME     | NOT NULL | VARCHAR2(30) |
| IPART_NAME     |          | VARCHAR2(30) |
| INDEX_ID       | NOT NULL | NUMBER       |
| TABLE_NAME     | NOT NULL | VARCHAR2(30) |
| PART_NAME      |          | VARCHAR2(30) |
| TABLE_ID       | NOT NULL | NUMBER       |
| KEYROWID       | NOT NULL | ROWID        |
| KEY            | NOT NULL | ROWID        |
| DUMP_TIMESTAMP | NOT NULL | DATE         |

#### 20. 4. 2 使用 CHECK\_OBJECT 过程检测讹误

CHECK\_OBJECT 过程检查指定的对象，填充修复表中讹误和修复指令等信息。当用户要检查对象的分区时可任意指定范围、分区名称和子分区名称。

证实操作包括检查对象中所有以前未标识为“讹误”的块。对每个块的事务处理和数据层分区做一致性检测。在执行 CHECK\_OBJECT 过程中，如果遇到一个块有讹误的缓冲头，这个块则被跳过。

下面是运行 CHECK\_OBJECT 过程的例子:

```
SET serveroutput on
DECLARE num_corrupt INT;
BEGIN
num_corrupt := 0;
DBMS_REPAIR.CHECK_OBJECT (
    SCHEMA_NAME => 'SCOTT',
    OBJECT_NAME => 'DEPT',
    REPAIR_TABLE_NAME => 'REPAIR_TABLE',
    corrupt_count => num_corrupt);
DBMS_OUTPUT.PUT_LINE('number corrupt: ' || TO_CHAR (num_corrupt));
END;
/
```

SQL\*PLUS 输出下面行，指明有一个讹误:

```
number corrupt: 1
```

查询修复表将产生描述讹误和提示修复行为的信息:

```
SELECT object_name, block_id, corrupt_type, marked_corrupt,
       corrupt_description, repair_description
FROM repair_table;
```

```
OBJECT_NAME          BLOCK_ID  CORRUPT_TYPE  MARKED_COR
```



| CORRUPT_DESCRIPTION         |  |            |       |
|-----------------------------|--|------------|-------|
| REPAIR_DESCRIPTION          |  |            |       |
| DEPT                        | 3                                      | 1          | FALSE |
| kdbchk:                     | row locked by non_existent transaction |            |       |
|                             | table=0                                | slot=0     |       |
|                             | lockid=32                              | ktbbhitc=1 |       |
| mark block software corrupt |  |            |       |

这时，讹误块还未被标识，所以这时应该取出任何有意义的数
据。讹误块被标识之后，整个块将被跳过。

20. 4. 3 使用 FIX\_CORRUPT\_BLOCKS 过程修复讹误块

根据由 CHECK\_OBJECT 产生的修复表信息使用 FIX\_CORRUPT\_BLOCKS 过程修复指定对象中讹误块。在对块做任何改变之前，检测该块，确保该块仍是讹误的。通过标记块讹误软件修复讹误块。进行修复时，修复表中相应的行用一修复时间信息更新。

下例修复了由 CHECK\_OBJECT 过程报告的 SCOTT.DEPT 表中的讹误块:

```
SET serveroutput on
DECLARE num_fix INT;
BEGIN
num_fix := 0;
DBMS_REPAIR.FIX_CORRUPT_BLOCKS (
    SCHEMA_NAME => 'SCOTT',
    OBJECT_NAME=> 'DEPT',
    OBJECT_TYPE => dbms_repair.table_object,
    REPAIR_TABLE_NAME => 'REPAIR_TABLE',
    FIX_COUNT=> num_fix);
DBMS_OUTPUT.PUT_LINE('num fix: ' || to_char(num_fix));
END;
/
```

SQL/PL 输出:

```
num fix: 1
```

下面查询语句进一步证实了修复操作:

```
SELECT object_name, block_id, marked_corrupt
FROM repair_table;
```

| OBJECT_NAME | BLOCK_ID | MARKED_COR |
|-------------|----------|------------|
| DEPT        | 3        | TRUE       |

20. 4. 4 找出指向讹误数据块的索引实体: DUMP\_ORPHAN\_KEYS

DUMP\_ORPHAN\_KEYS 过程报告指向讹误数据块中行的索引实体。对每个遇到的索引

实体，将一行插入指定的孤立关键字表中，在这之前孤立关键字表必须已创建。

如果指定了修复表，那么除被标识为软讹误的块之外的所有与基表关联的讹误块都将被处理。否则，只处理标识为讹误的块。

这些信息对于重建表中丢失的行和诊断是非常有用的。

**注意** 对于与在修复表中标识的表相关的每一个索引，应该运行这个操作。

在下面的例子中，PK\_DEPT 是 SCOTT.DEPT 表上的一个索引。扫描该索引，以判断是否存在指向讹误数据块中行的索引项。

```
SET serveroutput on
DECLARE num_orphans INT ;
BEGIN
num_orphans :=0 ;
DBMS_REPAIR.DUMP_ORPHAN_KEYS (
    SCHEMA_NAME => 'SCOTT',
    OBJECT_NAME => 'PK_DEPT',
    OBJECT_TYPE => dbms_repair.index_object,
    REPAIR_TABLE_NAME => 'REPAIR_TABLE',
    ORPHAN_TABLE_NAME => 'ORPHAN_KEY_TABLE',
    KEY_COUNT => num_orphans );
DBMS_OUTPUT.PUT_LINE('orphan key count: ' || to_char(num_orphans));
END ;
/
```

下面是其输出，表示有三个 orphan 键：

```
orphan key count: 3
```

在 orphan 键表中的索引项意味着索引必须重建，以保证表探针（probe）和索引探针返回相同的结果集。

#### 20.4.5 使用 REBUILD\_FREELISTS 过程来重建空闲列表

REBUILD\_FREELISTS 过程重建指定对象的空闲列表。所有的空闲块都放置在主空闲列表中。所有的其它列表都被零化。如果对象拥有多个空闲列表组，那么空闲块分布在所用的空闲列表之间，以 round-robin 方式分配到不同组中。

下面的例子重建 SCOTT.DEPT 的空闲列表。

```
BEGIN
DBMS_REPAIR.REBUILD_FREELISTS (
    SCHEMA_NAME => 'SCOTT',
    OBJECT_NAME => 'DEPT',
    OBJECT_TYPE => dbms_repair.table_object);
END ;
/
```

#### 20.4.6 启用或禁用讹误数据块的跳过：SKIP\_CORRUPT\_BLOCKS

SKIP\_CORRUPT\_BLOCKS 过程启用或禁用在指定对象的索引和表扫描期间讹误数据块的跳过。当对象是一个表时，跳过应用到表及其索引。当对象是一个簇时，跳过应用到簇

中的所有表及其相应的索引。

下列例子跳过 SCOTT.DEPT 表的软件讹误数据块:

```
BEGIN
DBMS_REPAIR.SKIP_CORRUPT_BLOCKS (
    SCHEMA_NAME => 'SCOTT',
    OBJECT_NAME => 'DEPT',
    OBJECT_TYPE => dbms_repair.table_object,
    FLAGS => dbms_repair.skip_flag);
END ;
/
```

使用 DBA\_TABLES 视图查询 SCOTT 表, 结果显示对于 SCOTT.DEPT 表, SKIP\_CORRUPT 是启用的。

```
SELECT owner, table_name, skip_corrupt FROM dba_tables
WHERE owner = 'SCOTT' ;
```

| OWNER | TABLE_NAME         | SKIP_COR |
|-------|--------------------|----------|
| SCOTT | ACCOUNT            | DISABLED |
| SCOTT | BONUS              | DISABLED |
| SCOTT | DEPT               | ENABLED  |
| SCOTT | DOCINDEX           | DISABLED |
| SCOTT | EMP                | DISABLED |
| SCOTT | RECEIPT            | DISABLED |
| SCOTT | SALGRADE           | DISABLED |
| SCOTT | SCOTT_EMP          | DISABLED |
| SCOTT | SYS_IOT_OVER_12255 | DISABLED |
| SCOTT | WORK_AREA          | DISABLED |

10 rows selected.

## 第五部分 数据库安全

---

第五部分提出了有关影响数据库安全的用户管理和权限管理的问题，包括以下章节：

- 第 21 章，“建立安全策略”
- 第 22 章，“用户和资源管理”
- 第 23 章，“用户权限和角色管理”
- 第 24 章，“数据库使用审计”

# 21

## 建立安全策略

本章提供了开发数据库操作安全性策略的指南，包括以下几个部分：

- 系统安全策略
- 数据安全策略
- 用户安全策略
- 口令管理策略
- 审计策略

### 21.1 系统安全策略

本节阐述了系统安全策略问题，包括以下几个部分：

- 数据库用户管理
- 用户身份验证
- 操作系统安全

每个数据库都有一个或多个管理员负责维护所有安全策略，称为安全管理员。如果数据库系统很小，则数据库管理员可兼作安全管理员。但是，如果数据库系统很大，则需要一个或一组特定的人担负安全管理员的职责。

确定系统安全管理员之后，必须为每个数据库开发安全策略，一个数据库的安全策略应该包括几个子策略，正如接下来的章节所描述的一样。

#### 21.1.1 数据库用户管理

数据库用户是指存取一个 Oracle 数据库信息的路径，因此对数据库用户管理来说，紧密的安全性应该是可维护的。根据数据库的大小以及管理数据库用户所需的工作量，安全管理员也许是唯一一个具有创建（create）、变更（alter）或删除（drop）数据库用户权限的用户。另一方面，也许有多个管理员具有管理数据库用户的权限。无论如何，只有可信任的人才有管理数据库用户的权限。

### 21.1.2 用户身份验证

数据库用户可通过使用数据库口令、主机操作系统、网络服务由 Oracle 或 Secure Sockets Layer (SSL) 进行身份验证 (验证是正确的用户)。

**注意** 使用网络鉴别服务或 SSL 进行身份验证需要安装 Oracle 高级安全系统 (Oracle Advanced Security), 请查阅 “Oracle Advanced Security Administrator’s Guide” 获取有关身份验证类型的信息。

有关用户身份验证以及如何运作的更多的信息, 参见第 22 章的 22.2 节“用户身份验证”。

### 21.1.3 操作系统安全

如果适用, 对运行 Oracle 和任何数据库应用程序的操作系统必须考虑下列安全问题:

- 数据库管理员必须具有创建或删除文件的操作系统权限;
- 典型数据库用户不具有创建或删除与数据库有关的文件的操作系统权限;
- 如果操作系统为用户标识数据库角色, 则安全管理员必须具有修改操作系统帐户的安全域的操作系统权限。

参见 有关操作系统针对 Oracle 的安全问题信息, 参见针对特定操作系统的 Oracle 文档。

## 21.2 数据安全策略 (Data Security Policy)

“数据安全”包括在对象级访问和使用数据库的控制机制。数据安全策略确定哪些用户可以访问一指定的模式对象以及用户对该对象可执行的操作类型。例如, 用户 SCOTT 对 EMP 表可执行 SELECT 和 INSERT 语句, 但不可执行 DELETE 语句。数据安全策略还应定义对每个模式对象进行审计的操作 (如果有的话)。

数据安全策略主要由用户希望为数据库中的数据建立的安全级来确定, 例如, 当你希望允许任何用户创建任何模式对象或给系统中的其他用户授予访问权限时, 应降低数据的安全性。或者, 当你希望指定一数据库管理员或安全管理员 (该管理员是唯一一个具有创建对象, 并为角色和用户授予访问该对象权限的人) 时, 严格控制数据安全是十分必要的。

总的来说数据安全应基于数据的敏感性 (sensitivity)。如果信息是不敏感的, 则数据安全策略可以更宽松。然而, 如果数据是敏感的, 则安全策略应设计成严格控制对对象的访问。

实现数据安全的方法包括系统和对象权限以及角色等。角色是可以授权给用户的成组的权限集, 权限和角色在 “管理用户和角色” 一章中论述。

由于视图的定义只限于访问表中的数据, 因此视图也可实现数据安全, 视图可以拒绝包含敏感数据的列。视图在第 18 章 “管理视图、序列和别名单元” 中论述。

实现数据安全的其它方法是通过 fine-grained 访问控制和使用一相关联的应用程序上下文。fine-grained 访问控制是 Oracle 的一个特征, 允许用户借助函数实现安全策略, 并将安全策略与表或视图关联起来。实际上, 安全策略函数生成一个附加到 SQL 语句的 WHERE 条件, 限制用户访问表或视图中的数据行。应用程序上下文是一安全的数据缓存器, 存储生成访问控制决策的信息。

参见 有关实现 fine-grained 访问控制和应用程序上下文的信息, 参见 “Oracle8i Application Developer’s Guide- Fundamentals” 和 “Oracle8i Supplied PL/SQL Packages Reference”。

## 21.3 用户安全策略

本节阐述了用户安全策略, 包括以下几个方面:

- 一般用户安全
- 最终用户安全
- 管理员安全
- 应用程序开发者安全
- 应用程序管理员安全

### 21.3.1 一般用户安全 (General User Security)

对所有类型的数据库用户来说, 需考虑下列一般用户安全问题:

- 口令安全
- 权限管理

#### 口令安全 (Password Security)

如果用户身份验证是由数据库管理的, 那么安全管理员应开发口令安全策略以维护数据库的访问安全性。例如, 数据库用户要求每隔一段时间更改口令, 当然, 当口令泄露时更应及时更改, 这种情况下通过强迫用户修改口令可减少对数据库的非法访问。

为更好地保护口令的机密性, 对客户/服务器和服务器/服务器的连接使用密码配置 Oracle。

**注意** 强烈建议用户在客户/服务器和服务器/服务器的连接中为 Oracle 配置密码。否则, 某个经常在网上窥探的不怀好意的用户可截取一不加保密的口令, 并使用该口令连接到数据库, 扮演其他用户角色。

通过设置下列数值, 用户可将用来证实连接的口令译成密码:

- 在客户机器上将 ORA\_ENCRYPT\_LOGIN 环境变量设置为 TRUE;
- 将服务器初始化参数 DBLINK\_ENCRYPT\_LOGIN 设置为 TRUE。

如果客户机和服务器上都做了相应的设置, 那么不会在网络上畅行无阻地发送口令, 而是使用 DES (数据加密标准) 算法将口令译成密码。

初始化参数 DBLINK\_ENCRYPT\_LOGIN 用来连接两个 Oracle 服务器 (例如, 执行分布式查询时)。如果与客户连接, Oracle 将检查环境变量 ORA\_ENCRYPT\_LOGIN。

无论何时用户使用口令与服务器连接时, Oracle 在将口令发送到服务器之前先将其译成密码。如果连接失败并启用了审计, 失败信息将记录在审计日志中, 然后 Oracle 检查 DBLINK\_ENCRYPT\_LOGIN 或 ORA\_ENCRYPT\_LOGIN 的值, 如果是设置为 FALSE, Oracle 则使用未译成密码的口令再次进行连接, 如果连接成功, 连接信息将替换审计日志中的失败信息, 连接继续进行。为防止不怀好意的用户再次使用未加密的口令进行连接, 用户必须将相应的变量设置为 TRUE。

## 权限管理 (Privilege Management)

安全管理员应考虑与所有用户有关的权限管理的问题。例如，一个数据库中有许多用户名，或许运用角色来管理用户相关权限是非常有益的，角色是授予用户或其他角色的相关权限的命名组；相反，如果数据库有意把用户名，或许显式地给用户授权显得更为容易，避免角色的使用。

管理一个拥有许多用户、应用程序或对象的数据库的安全管理员应该充分利用角色提供的好处，在复杂环境中，角色将极大简化权限管理的任务。

### 21.3.2 最终用户安全 (End-User Security)

安全管理员必须为最终用户定义安全策略。如果数据库很大，并有很多用户，安全管理员则可以决定哪些用户可以分类，为这些用户组创建用户角色，给每个用户角色授予必要的权限或应用程序角色，将用户角色分配到用户。为处理例外情况，安全管理员还必须确定什么权限必须显式地授给单个用户。

#### 为最终用户权限管理运用角色

角色是授予和管理不同数据库用户组所要求的共同权限的最容易的方法。设想某公司会计部门的每个用户都需要有运行 ACCTS\_RECEIVABLE 和 ACCTS\_PAYABLE 数据库应用程序的权限，角色应与这两个应用程序结合起来，包含执行这两个应用程序所必须的对象权限。

下列由数据库管理员或安全管理员执行的操作解决了这种简单的安全问题：

1. 创建一个名为 ACCOUNTANT 的角色；
2. 将授予给 ACCTS\_RECEIVABLE 和 ACCTS\_PAYABLE 数据库应用程序的角色授权给 ACCOUNTANT 角色；
3. 将 ACCOUNTANT 角色授权给会计部门的每个用户。

该安全模型如图 21-1 所示。

这个图解决了下列潜在的情况：

- 如果会计师以后需对新的数据库应用程序授权一个角色，那么那个应用程序的角色可以授权给 ACCOUNTANT 角色，会计部门的用户将自动接收与新数据库应用程序相关联的权限，该应用程序的角色不必授权给运用该程序的单个用户。
- 同样，如果会计部门不再需要某个特定的程序，该程序的角色可以从 ACCOUNTANT 角色中删除。
- 如果 ACCTS\_RECEIVABLE 或 ACCTS\_PAYABLE 程序要求的权限发生变化，新的权限可以授权给程序的角色，或从程序的角色中删除。ACCOUNTANT 角色的安全域和所有已授权 ACCOUNTANT 角色的用户自动反映权限的修改。

可能的话，优化所有情况下的角色，使得最终用户权限管理简单高效。

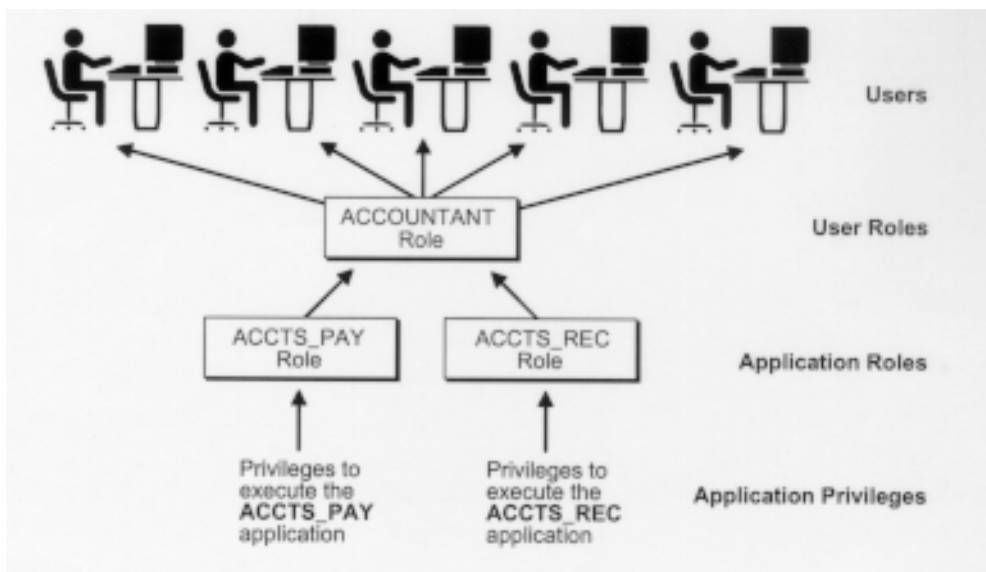


图 21-1 用户角色

### 为最终用户权限管理运用目录服务

在目录服务（directory service）中，你可通过企业用户和企业角色的 Oracle 高级安全特征集中管理用户以及身份验证。参见“Oracle Advanced Security Administrator's Guide”获取更多信息。

### 21.3.3 管理员安全（Administrator Security）

安全管理员应有处理管理员安全的策略。例如，当数据库很大并且有几个不同类型的数据库管理员时，安全管理员可以决定将相关的管理员权限分组成几个管理员角色，然后可以将管理员角色授权给适当的管理员用户。相反，当数据库很小并且只有少数管理员时，创建一个管理员角色并授权给所有管理员，这样做更为方便。

#### SYS 和 SYSTEM 连接保护

创建数据库之后应立即为 SYS 用户和 SYSTEM 用户设置口令，阻止非法用户访问数据库。以 SYS 用户身份和 SYSTEM 用户身份连接数据库使得用户有很多方式修改数据库，以 SYS 身份连接允许用户修改数据目录表。因此，对这些用户名的授权是非常敏感的，只是在选择数据库管理员时才有效。

可运用第 22 章的 22.3.2 节“改变用户”中介绍的过程来修改这些帐号的口令。

#### 管理员连接保护

只有数据库管理员才能使用管理员权限连接到数据库（例如，作为 SYSDBA/SYSOPER 连接），以 SYSOPER 身份连接使得用户能够执行某些基本操作（比如 STARTUP、SHUTDOWN 和 RECOVER）；以 SYSDBA 身份连接使得用户具有以上能力，再加上对数据库或数据库对象执行任何操作（比如 CREATE、DROP 和 DELETE）；以 SYSDBA 身份连接将用户放置在 SYS 模式中，可以改变数据目录表。



### 为管理员权限管理运用角色

角色是限制强大系统权限和数据库人事管理需要的角色最容易的方法。

设想在大型系统安装时几个数据库管理员共享数据库管理员职责，每个数据库管理员负责下列特定的数据库管理任务：

- 一个管理员负责对象创建和维护
- 一个管理员负责数据库性能调整
- 一个安全管理员负责创建新用户，给数据库用户授权角色和权限
- 一个数据库管理员负责数据库常规操作（例如，STARTUP、SHUTDOWN、BACKUP）
- 一个管理员负责紧急情况处理，比如数据库恢复
- 新的、无经验的数据库管理员给予有限的权限实验数据库管理

在这种情景中，安全管理员应为管理人员构造安全策略，如下所示：

1. 定义六个角色（例如，DBA\_OBJECTS、DBA\_TUNE、DBA\_SECURITY、DBA\_MAINTAIN、DBA\_RECOV、DBA\_NEW），这些角色包含完成每类工作所需的不同的权限。

2. 每个角色授予适当的权限。

3. 每类数据库管理员可以授权给相应的角色。

这种策略减少了下列问题出现的可能性：

- 如果一个数据库管理员的工作发生变化，担负更多的职责，那么可将该数据库管理员授权给与新职责相对应的其他管理角色。
- 如果一个数据库管理员的工作发生变化，只担负少量的职责，那么该数据库管理员可取消适当的管理角色。
- 数据字典总是存储每个角色和每个用户的信息，这些信息有助于公开每个管理员的任务。

### 21.3.4 应用程序开发者安全（Application Developer Security）

安全管理员必须为使用数据库的程序开发者定义一条特殊的安全策略，安全管理员可以授权程序开发者创建必要的对象，另外，创建对象权限可以只授权给数据库管理员，数据库管理员接收开发者创建对象的请求。

#### 应用程序开发者以及他们的权限

数据库应用程序开发者是唯一要求特殊权限来完成任务的数据库用户，不像最终用户，开发者需要系统权限，比如CREATE TABLE、CREATE PROCEDURE等等。但是，只有授权给开发者指定的系统权限才可限制开发者对数据库的全面操作。

#### 应用程序开发者环境：测试数据库和产品数据库

在很多种情况下，应用程序开发只限于测试数据库，而不允许在产品数据库上开发。这种限制确保程序开发者不与最终用户竞争资源，不会对产品数据库做出有害的影响。

应用程序完全开发完和测试之后，允许访问产品数据库，并供产品数据库的最终用户使

用。

#### 无约束和有约束应用开发环境

当数据库管理员决定将哪些权限授权给程序开发者时可以定义以下选项：

- |       |  |
|-------|--|
| 无约束开发 | 应用程序开发者允许创建新的模式对象，包括表、索引和过程等等，该选项允许开发者开发独立于其他用户的应用程序。                                |
| 有约束开发 | 程序开发者不允许创建新的模式对象，所有需要的表、索引、过程等由数据库管理员创建，由开发者提出申请。该选项允许数据库管理员完全控制数据库空间的使用以及对数据库信息的访问。 |

尽管有些数据库系统只使用一个选项，但还有些系统混合使用这两个选项。例如，允许程序开发者创建新的存储过程和程序包，但不允许创建表或索引。安全管理员关于这个问题的决定应基于以下几个方面：

- 控制数据库空间的使用
- 控制模式对象的访问路径
- 用来开发应用程序的数据库 如果是测试数据库用于应用程序开发，那么开发策略可以更为宽松。

#### 应用程序开发者的角色和权限

安全管理员可创建角色来管理典型程序开发者所要求的权限。例如，具有代表性的、名为 APPLICATION\_DEVELOPER 角色也许包含 CREATE TABLE、CREATE VIEW 和 CREATE PROCEDURE 等系统权限，为程序开发者定义角色时请考虑下列因素：

- 通常将 CREATE 系统权限授权给程序开发者，这样他们可以创建自己的对象，但是，通常不将 CREATE ANY 系统权限授权给开发者，该权限允许用户在其他任何用户模式中创建对象。这就限制了用户只能在自己的模式中创建新的对象。
- 很少将对象权限授权给开发者使用的角色，因为通过角色授权对象权限在创建其他对象（主要是视图和存储过程）将受到极大限制，允许开发者为程序开发创建自己的对象更为实际。

#### 强加于程序开发者的空间限制

应用程序开发者通常授权创建对象，作为开发过程的一部分，安全管理员必须限制什么空间可以让程序开发者使用，以及如何使用。例如，作为安全管理员应该为程序开发者特别设置以下限制：

- 开发者可创建表或索引的表空间
- 开发者可存取的表空间的配额

这两个限制可通过改变开发者的安全域来设置，它将在第 22 章“改变用户”一节中讨论。

### 21.3.5 应用程序管理员安全 (Application Administrator Security)

大型数据库中包含许多数据库应用程序（例如 Precompiler 和 Forms 应用程序），用户可能需要应用程序管理员，一个应用程序管理员负责以下任务：

- 为应用程序创建角色，管理每个角色的权限；
- 创建和管理数据库应用程序所使用的对象；
- 必要时，维护和更新应用程序代码、Oracle 过程和程序包。

应用程序管理员通常也是设计程序开发者，但是，这些工作不是开发者的职责所在，并可分配给熟悉数据库应用程序的其他人。

## 21.4 口令管理策略 (Password Management Policy)

数据库安全系统依赖于始终保密的口令，然而，口令是非常脆弱的，容易遭受偷窃、伪造和滥用。为最大限度地保证数据库的安全，Oracle 的口令管理策略由 DBAs 控制。

- 口令老化和终止
- 口令历史
- 口令复杂性验证

### 21.4.1 帐号锁定

当一用户多次登录都失败时，服务器自动锁定该用户的帐号，DBAs 使用 `CREATE PROFILE` 语句指定允许 `LOGIN` 失败的次数，DBAs 同时指定帐号锁定的时间。

在下例中，用户 `ASHWINI` 允许 `LOGIN` 失败的次数是 4，且该帐号的锁定时间是 30 天，30 天后该帐号自动解锁。

```
CREATE PROFILE prof LIMIT
  FAILED_LOGIN_ATTEMPTS 4
  PASSWORD_LOCK_TIME 30;
ALTER USER ashwini PROFILE prof;
```

如果 DBA 不指明帐号解锁时间间隔，则 `PASSWORD_LOCK_TIME` 假设该值在缺省 Profile 中指定；如果 DBA 将 `PASSWORD_LOCK_TIME` 指明为 `UNLIMITED`，则系统安全管理员必须显式地给该帐号解锁。

安全管理员同样可以显式地锁定该帐号，这时，该帐号不能自动解锁，只有安全管理员可将该帐号解锁。

参见 有关 `CREATE PROFILE` 语句的更多信息，参见 “Oracle8i SQL Reference”。

### 21.4.2 口令老化和终止

DBAs 使用 `CREATE PROFILE` 指明口令的最大生存周期，超过指定的生存周期，口令将终止，用户或 DBA 必须改变口令。下面语句指出 `ASHWINI` 在 90 天内可使用同一个口令，之后该口令将终止：

```
CREATE PROFILE prof LIMIT
  FAILED_LOGIN_ATTEMPTS 4
  PASSWORD_LOCK_TIME 30;
  PASSWORD_LIFE_TIME 90;
ALTER USER ashwini PROFILE prof;
```

DBAs 还可使用 `CREATE PROFILE` 语句指明宽限期 (`grace period`)，口令终止用户第一

次登录数据库帐号的时间即为宽限期开始之时，在宽限期内每当用户登录数据库时都会出现一条警告信息，直到口令终止。用户必须在宽限期内改变口令，如果在宽限期内不改变口令，帐号将终止，用户不允许再登录那个帐号，直至用户改变口令。图 21-2 显示了口令生存周期和宽限期的时序。

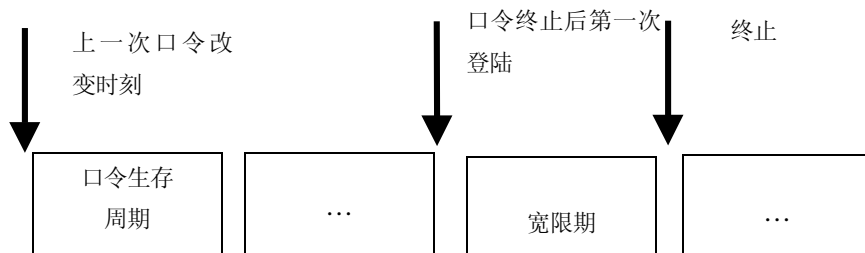


图 21-2 口令生存周期和宽限期的时序

例如，假设口令的生存周期是 60 天，宽限期是 3 天，如果用户 60 天之后试图登录数据库（有可能是第 70 天、第 100 天或其他时间，这个时间应是口令生存周期之后用户的第一次登录），该用户将收到一条警告信息，指出该口令将在 3 天之内终止，如果用户在宽限期内不改变口令，则该用户的帐号将终止。下面语句指出用户必须在宽限期（3 天）之内改变口令：

```
CREATE PROFILE prof LIMIT
  FAILED_LOGIN_ATTEMPTS 4
  PASSWORD_LOCK_TIME 30
  PASSWORD_GRACE_TIME 3;
ALTER USER ashwini PROFILE prof;
```

安全管理员亦可显式地终止帐号，这对于新的帐号来说非常有用。

#### 21.4.3 口令历史

DBAs 使用 CREATE PROFILE 语句指定用户不可重新使用同一口令的时间间隔。

下列语句中，DBA 指出用户 60 天内不可重新使用同一口令：

```
CREATE PROFILE prof LIMIT
  PASSWORD_REUSE_TIME 60
  PASSWORD_REUSE_MAX UNLIMITED;
```

下面语句指出用户必须将口令改变 3 次方可重新使用当前口令：

```
CREATE PROFILE prof LIMIT
  PASSWORD_REUSE_MAX 3
  PASSWORD_REUSE_TIME UNLIMITED;
```

**注意** 如果用户指定了 PASSWORD\_REUSE\_TIME 或 PASSWORD\_REUSE\_MAX 的值，则必须将另一个的值指定为 UNLIMITED 或根本不指定它。

#### 21.4.4 口令复杂性验证

Oracle 口令的复杂性验证程序可运用 PL/SQL 脚本（UTLPWDMG.SQL）来指定，该脚

本设置了 PROFILE 的缺省参数。

口令复杂性验证程序执行下列任务：

- 口令的最小长度为 4；
- 口令不同于 USERID；
- 口令至少包含一个阿尔法字母、一个数字和一个标点符号；
- 口令不可以是一些简单的单词，比如 welcome、account、database 或 user；
- 新口令与上一个口令至少有 3 个不同的字母。

**注意** 建议用户不要使用 ALTER USER 语句改变口令，因为它不完全支持口令验证函数，可使用 OCIPasswordChange ( ) 改变口令。

#### 口令验证程序格式指南

DBAs 可增强现有口令复杂性验证程序，或使用 PL/SQL（或其它第三方工具）创建自己的口令验证程序。

DBA 授权的 PL/SQL 系统调用必须遵循以下格式：

```
routine_name (  
  userid_parameter IN VARCHAR(30),  
  password_parameter IN VARCHAR (30),  
  old_password_parameter IN VARCHAR (30)  
)  
RETURN BOOLEAN
```

程序创建之后必须运用用户的 PROFILE 或系统的缺省 PROFILE 将其指定为口令验证程序：

```
CREATE/ALTER PROFILE profile_name LIMIT  
PASSWORD_VERIFY_FUNCTION routin_name
```

这个口令验证程序必须归 SYS 所有。

下面例子设置了缺省口令资源限制，提供了口令复杂性最低限度的检查，用户在为新口令创建自己的复杂性检查程序时可以以此为模板。

该脚本设置了缺省口令资源参数，必须在启用口令功能部件时运行它。然而，如果必要，用户可改变缺省资源参数。

缺省口令复杂性验证函数至少检查以下项目：

- 口令满足最小长度要求；
- 口令不是用户名，用户可根据需要修改这个函数。

该函数必须在 SYS 模式中创建，而且运行它之前以 SYSDBA 身份连接 SYS。

```
CREATE OR REPLACE FUNCTION verify_function  
(username varchar2,  
  password varchar2,  
  old_password varchar2)  
RETURN boolean IS  
  n boolean;  
  m integer;
```

```

differ      integer;
isdigit     boolean;
ischar      boolean;
ispunct     boolean;
digitarray  varchar2(20);
punctarray  varchar2(25);
chararray   varchar2(52);

BEGIN

digitarray:= '0123456789';
chararray:= 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
punctarray:= '!"#$%&()*+,-./:;<=>?_';
--Check if the password is same as the username
IF password = username THEN
    raise_application_error(-20001, 'Password same as user');
END IF;

--Check for the minimum length of the password
IF length(password) < 4 THEN
    raise_application_error(-20002, 'Password length less than 4');
END IF;

--Check if the password is too simple. A dictionary of words may be
--maintained and a check may be made so as not to allow the words
--that are too simple for the password.
IF NLS_LOWER(password) IN ('welcome', 'database', 'account', 'user',
    'password', 'oracle', 'computer', 'abcd')
    THEN raise_application_error(-20002, 'Password too simple');
END IF

--Check if the password contains at least one letter,
--one digit and one punctuation mark.
--1. Check for the digit
--You may delete 1. and replace with 2. or 3.
isdigit:=FALSE;
m := length(password);
FOR i IN 1..10 LOOP
    FOR j IN 1..m LOOP
        IF substr(password,j,1) = substr(digitarray,i,1) THEN
            isdigit := TRUE;
            GOTO findchar;
        END IF;
    END LOOP;
END LOOP;
IF isdigit = FALSE THEN
    raise_application_error(-20003, 'Password should contain at least one\

```

```

        digit, one character and one punctuation');
END IF;
--2. Check for the character

<<findchar>>
ischar := FALSE;
FOR i IN 1..length(chararray) LOOP
    FOR j IN 1..m LOOP
        IF substr(password,j,1) = substr(chararray,i,1) THEN
            ischar := TRUE;
            GOTO findpunct;
        END IF;
    END LOOP;
END LOOP;
IF ischar = FALSE THEN
    raise_application_error(-20003, 'Password should contain at least one digit,\
one character and one punctuation');
END IF;
--3. Check for the punctuation

<<findpunct>>
ispunct := FALSE;
FOR i IN 1..length(punctarray) LOOP
    FOR j IN 1..m LOOP
        IF substr(password,j,1) = substr(punctarray,i,1) THEN
            ispunct := TRUE;
            GOTO endsearch;
        END IF;
    END LOOP;
END LOOP;
IF ispunct = FALSE THEN raise_application_error(-20003, 'Password should \
contain at least one digit, one character and one punctuation');
END IF;

<<endsearch>>
--Check if the password differs from the previous password by at least 3 letters
IF old_password = " THEN
    raise_application_error(-20004, 'Old password is null');
END IF
--Everything is fine; return TRUE;
differ := length(old_password) - length(password);
IF abs(differ) < 3 THEN
    IF length(password) < length(old_password) THEN
        m := length(password);
    ELSE
        m := length(old_password);

```

```

        END IF;
        differ := abs(differ);
        FOR i IN 1..m LOOP
            IF substr(password,i,1) != substr(old_password,i,1) THEN
                differ := differ+1;
            END IF;
        END LOOP;
        IF differ < 3 THEN
            raise_application_error(-20004, 'Password should differ by at \
least 3 characters');
        END IF;
        END IF;
        --Everything is fine; return TRUE;
        RETURN (TRUE)
    END

```

## 21.5 审计策略

安全管理员应为每个数据库的审计过程定义安全策略，例如，你或许决定禁用数据库审计除非某些活动受到怀疑。当需要审计时，安全管理员必须确定审计数据库的级别，通常，确定受到怀疑的活动的起源之后，先做指定类型的审计再做常规的系统审计。审计将在第 24 章“数据库使用审计”中讨论。



# 22 用户和资源管理

本章描述如何控制对一个 Oracle 数据库的访问，包括以下主题：

- 会话(Session)和用户许可(User Licensing)
- 用户鉴别(User Authentication)
- Oracle 用户
- 用概要文件(Profiles)管理资源(Resources)
- 关于数据库用户和概要文件的列表信息
- 示例

关于建立用户和概要文件安全策略的指导请参见第 21 章“建立安全策略”。

在数据库中一个用户对数据库和模式对象的访问通过权限（Privileges）和角色（Roles）控制，关于权限和角色的详细信息参见第 23 章“用户权限和角色管理”

## 22.1 会话和用户许可

Oracle 帮助用户确保其组织遵循 Oracle 服务器许可协议。如果组织被授予并发使用（concurrent usage）许可，则可以在跟踪和限制同时连接到数据库的会话（Session）数。如果被授予指定用户许可，则可以限制数据库中指定用户的数量。对于这两种情况，控制授予许可工具，都必须使授予许可工具可用和设置合适的限制。通过在初始化参数文件中设置如下初始化参数来完成这项工作：

- LICENSE\_MAX\_SESSIONS
- LICENSE\_SESSIONS\_WARNING
- LICENSE\_MAX\_USERS

接下来讨论这些初始化参数。

要使用授予许可工具，必须知道组织有哪种类型的许可协议以及会话的最大数或指定的用户。用户的组织可以使用任意一种许可（并发使用或指定用户）类型，但不能同时使用两种。

**注意** 在少数情况，一个组织可能拥有非限制许可，既不是并发使用许可也不是指定用户许可。只有在这种情况下，让授予许可机制不可用，同时在初始化参数文件中省略 LICENSE\_MAX\_SESSIONS, LICENSE\_SESSIONS\_WARNING, LICENSE\_MAX\_USERS，或者将这三项的值全部设为 0。

这部分描述会话和用户许可的特性，包括以下部分：

- 并发使用许可
- 指定用户限制
- 查看许可限制和当前值

参见 关于初始化参数 LICENSE\_MAX\_SESSIONS, LICENSE\_SESSIONS\_WARNING, LICENSE\_MAX\_USERS 的描述和语法, 参见 “Oracle8i Reference”。

### 22.1.1 并发使用许可

并发使用许可限定在指定计算机上能同时连接到数据库的会话数。用户可以在启动一个实例前设定同时连接的会话数。事实上, 应该按照第 2 章 “创建 Oracle 数据库” 描述的那样, 作为初始化安装过程的一部分, 就设置好这项限制。用户也可以在数据库运行时改变同时连接的最大会话数。

#### 连接权限

在连接到数据库实例的会话数到极限后, 只有具有 RESTRICTED SESSION 权限的用户 (通常是 DBA) 能连接到数据库。当具有 RESTRICTED SESSION 权限的用户连接到数据库时, Oracle 发送一条信息给这个用户, 通知连接到数据库的会话数已到极限, 并把这条信息写入报警 (ALERT) 文件中。当会话数到达极限时, 你应连接到数据库终止那些不必要的进程。关于终止会话的内容, 参见第 4 章的 4.6 节 “终止会话”。

除非已升级 Oracle 许可协议, 否则不要提高许可限制。

除了可以设置最大并发会话数限制以外, 还可以设置一个并发会话警戒限制。在到达这一限制数后, 另外的用户可以继续连接 (直到最大限制数); 然而, Oracle 将在每一个连接的报警 (ALERT) 文件中写入一条相应的信息, 并通知每一个连接中的具有 RESTRICTED SESSION 权限的用户, 警告连接会话数要到极限了。

如果正在连接的用户具有系统管理员权限, 限制仍然适用; 只是, Oracle 在执行完该用户的第一条语句后强制执行该限制。

除了强制执行并发使用限制外, Oracle 跟踪每一个数据库实例的最高并发会话数。可以利用这一 “高水位标记” 帮助确定是否需要检查 Oracle 许可。关于 Oracle 许可限制升级的有关信息, 参见 22.1.3 节 “查看许可限制和当前值”

对于运行在并行服务器 (Parallel Server) 下的实例, 每一个实例都有自己的并发使用限制和警戒限制。但是, 这些实例限制的总和不能超过组织的并发使用许可。

**警告** 通过多路转换软件或硬件 (如 TP 监视器) 连接到数据库的每一个会话都提供一个单独的并发使用限制。然而, Oracle 许可机制不能区分通过这种方法连接的会话数。因此, 如果你的组织使用多路转换软件或硬件, 你必须考虑到这一点, 设置最大并发使用限制低于多路转换连接的会话数。

参见 关于在并行服务器环境下设置和更改限制的更多信息, 参见 “Oracle8i Parallel Server Administration, Deployment, and Performance”。

#### 设置最大会话数

通过赋值给初始化参数 LICENSE\_MAX\_SESSIONS, 设置一个实例的最大并发会话数, 下面这个例子设置的最大并发会话数是 80。

```
LICENSE_MAX_SESSIONS = 80
```

如果已经设置了这项限制, 也可以不设置警戒限制 (LICENSE\_SESSIONS\_WARN-

ING)。但是，使用警戒限制可以使最大限制易于管理，因为，它在组织使用接近最大值时给出一个预先的通知。

#### 设置会话的警戒限制

要给一个实例设置警戒限制，在实例启动使用的参数文件中设置初始化参数 `LICENSE_SESSIONS_WARNING`。

给警戒限制赋一个小于最大并发会话限制（`LICENSE_MAX_SESSIONS`）的值。

#### 在数据库运行时更改并发使用限制

要在数据库运行时更改最大并发使用限制或警戒限制，使用带有合适选项的 `ALTER SYSTEM` 语句。下面这条语句将最大限制数改为 100 个并发会话：

```
ALTER SYSTEM SET LICENSE_MAX_SESSIONS = 100;
```

下面的语句同时改变并发会话的警戒限制和最大限制：

```
ALTER SYSTEM  
  SET LICENSE_MAX_SESSIONS = 64  
  LICENSE_SESSIONS_WARNING = 54;
```

如果更改后的值低于当前会话数，那么当前的会话还要保持；但是，新的限制将对所有新的连接生效，直到实例关闭为止。要永久改变这个限制，必须改变初始化参数文件中的相应参数。

要在数据库运行时改变并发使用限制，必须有 `ALTER SYSTEM` 权限。同样，在实例的连接到达最大限制后要连接到该实例，必须有 `RESTRICTED SESSION` 权限。

**警告** 除非 Oracle 服务器许可已经升级，否则不要提高并发许可限制。关于更多的信息请与你的 Oracle 代理联系。

**参见** 关于使用 `ALTER SYSTEM` 语句更改初始化参数的内容，参见“Oracle8i SQL Reference”。

### 22.1.2 指定用户限制

指定用户许可限制在指定计算机上使用 Oracle 的已授权的单独用户数。要执行这项许可，可以在启动一个实例前设置在数据库中创建用户的数量限制。也可以在实例运行时改变用户的最大限制，或者完全关闭这一限制。在到达这一限制后你就不能再创建新的用户了，如果仍试图创建用户，Oracle 会返回一个错误信息告知已到最大用户极限了，并将这条信息写入报警（`ALERT`）文件中。

这一机制的运行假设每一个访问数据库的人都有唯一的用户名，并且没有共享用户。不要允许多个用户用同一个用户名进行连接。

#### 设置用户限制

要限制在数据库中创建用户的数量，在数据库参数文件中设置参数 `LICENSE_MAX_USERS`。下面这个例子将用户的最大数量设置为 200：

```
LICENSE_MAX_USERS = 200
```

如果数据库中的用户多于 `LICENSE_MAX_USERS` 的数值，那么当数据库启动时，Oracle 会返回一个警告并将信息写入报警（`ALERT`）文件中。除非将用户数删除到低于最大限制或者升级 Oracle 许可，否则就无法再创建新的用户。

**参见** 对于运行在并行服务器下的实例，连接到同一数据库的所有实例应享有统一的用户限制。参见 “Oracle8i Parallel Server Administration, Deployment, and Performance”。

### 更改用户限制

要改变最大用户限制，使用带有 `LICENSE_MAX_USERS` 选项的 `ALTER SYSTEM` 语句。下面的语句将定义的最大用户数改为 300：

```
ALTER SYSTEM SET LICENSE_MAX_SESSIONS = 300;
```

如果改变的限制低于当前用户数，Oracle 将返回一个错误并继续使用原来的限制。如果成功改变了这个限制，那么新的限制保持到关掉这一实例为止；要永久改变这个限制，需改变参数文件中 `LICENSE_MAX_USERS` 的值。

要改变指定用户的最大限制，你必须有 `ALTER SYSTEM` 权限。

**警告** 除非你的 Oracle 服务器许可已经升级，否则不要提高指定用户限制。关于更多的信息请与你的 Oracle 代理联系。

### 22.1.3 查看许可限制和当前值

通过查询实例的 `V$LICENSE` 数据字典视图，可以查看所有许可的当前限制、当前会话数和最大并发会话数。通过这些信息可以确定是否需要升级 Oracle 许可，以便允许更多的并发会话或指定用户：

```
SELECT sessions_max s_max,
       sessions_warning s_warning,
       sessions_current s_current,
       sessions_highwater s_high,
       users_max
FROM v$license;
```

| S_MAX | S_WARNING | S_CURRENT | S_HIGH | USERS_MAX |
|-------|-----------|-----------|--------|-----------|
| 100   | 80        | 65        | 82     | 50        |

另外，当数据库关闭时 Oracle 将会话的高水位标记写入数据库警告（`ALERT`）文件中，因此可以在那检查它。

要查看数据库中定义的指定用户的当前值，使用下面语句：

```
SELECT COUNT(*) FROM dba_users;
```

```
COUNT(*)
-----
      174
```

**参见** 关于 `V$LICENSE` 视图的完整描述，参见 “Oracle8i Reference”。

## 22.2 用户身份验证

在被允许建立数据库会话以前使用哪种方法定义用户，取决于你想怎样验证用户身份：

1. 可以定义用户的身份识别和身份验证都由数据库执行，称作“数据库身份验证 (database authentication)”。
2. 可以定义用户的身份验证由操作系统或网络服务器执行，称作“外部身份验证 (external authentication)”。
3. 可以定义用户身份验证由 SSL (Secure Sockets Layer) 执行。这些用户被称作全局用户 (global users)。对于全局用户，一个企业目录可以通过全局角色 (global roles) 用于授权对数据库的访问。
4. 可以指定用户允许通过代理服务器连接。这被称作多层验证和授权。

下面部分将讨论这些身份验证类型：

- 数据库身份验证
- 外部身份验证
- 全局身份验证和授权
- 多层身份验证和授权

在数据库中创建用户时要考虑的其它属性将在后面 22.3 节“Oracle 用户”中讨论。

### 22.2.1 数据库身份验证

如果为用户选择了数据库身份验证，那么用户帐号、口令管理以及用户的身份验证全部由 Oracle 完成。对于 Oracle 身份验证的用户，在创建和修改用户时指定用户的口令。用户可以在任何时候修改自己的口令。口令以加密的格式存储。每一个口令必须是由单字节字符组成，即使数据库使用的是多字节设置也是如此。

为了在使用数据库身份验证时提高安全性，Oracle 建议使用口令管理，包括帐户锁定、口令时效 (aging) 和终止、口令历史以及口令复杂校验。关于 Oracle 口令管理见第 21 章“建立安全策略”。

#### 建立由数据库进行身份验证的用户

下面这条语句创建了一个由 Oracle 识别和验证身份的用户。用户 SCOTT 无论何时要连接到数据库都必须指定口令 TIGER。

```
CREATE USER scott IDENTIFIED BY tiger;
```

参见 关于口令的有效性以及如何在 CREATE USER 和 ALTER USER 语句中指定 IDENTIFIED BY 子句的更多信息，参见“Oracle8i SQL Reference”。

## 数据库身份验证的优点

下面是数据库身份验证的优点：

- 用户帐户和全部身份验证完全由数据库控制。不依赖数据库以外的任何东西。
- 当使用数据库身份验证时 Oracle 提供了强大的口令管理特性以增强安全性。
- 当有较少的用户组时非常易于管理。

### 22.2.2 外部身份验证

当为用户选择了外部身份验证时，用户的帐号保留在 Oracle 中，但是口令的管理和用户的身份验证由外部服务器完成。这个外部服务器可能是操作系统或网络服务器，如 Net8。

对于外部身份验证，数据库依靠操作系统或网络身份验证服务器限制对数据库帐户的访问。这种类型的登录不需要数据库口令，如果操作系统或网络服务器允许，可以进行用户身份验证。如果要这样做，需设置初始化参数 OS\_AUTHENT\_PREFIX，同时在 oracle 用户名前使用这个前缀。OS\_AUTHENT\_PREFIX 参数定义一个前缀，Oracle 把它加到每个用户的操作系统帐户名前面。当一个用户要进行连接时 Oracle 将有前缀的用户名与数据库中的 Oracle 用户进行比较。

例如：假设 OS\_AUTHENT\_PREFIX 进行了如下设置：

```
OS_AUTHENT_PREFIX = OPS$
```

如果一个名叫“TSMITH”的操作系统用户要连接到 Oracle 数据库，同时由操作系统进行身份验证，那么 Oracle 则检查是否存在一个对应的数据库用户“OPS\$TSMITH”，如果是则允许用户连接。所有由操作系统进行身份验证的用户的引用都必须带有前缀，就像用户“OPS\$TSMITH”一样。对于向后兼容的 Oracle 早期版本，这个参数的缺省值是 OPS\$。但是，用户可以将其改为其它字符串或设置成空字符串（一对空的双引号：“”）。用空字符串代替了前缀的操作系统帐户名，那么 Oracle 用户名实际上就与操作系统的用户名一致。

如果设置了 OS\_AUTHENT\_PREFIX，那么它应该与数据库保持同样的生命周期。如果改变了这个前缀，那么，任何包含原来前缀的用户就都不能再建立连接了，除非改变用户的口令验证类型。

## 创建外部身份验证的用户

下面的语句创建了一个由 Oracle 识别、由操作系统或网络服务器验证身份的用户，这个例子假设：OS\_AUTHENT\_PREFIX = “”。

```
CREATE USER scott IDENTIFIED EXTERNALLY;
```

使用 CREATE USER...IDENTIFIED EXTERNALLY，创建的用户必须通过操作系统或网络服务器进行身份验证。Oracle 依靠这个外部登录验证来保证操作系统中指定的用户可以访问数据库中指定的用户。

参见 参数 OS\_AUTHENT\_PREFIX 的文本与操作系统有关，关于这个初始化参数的更多信息参见指定操作系统的 Oracle 文档；关于外部身份验证的更多信息，参见“Oracle Advanced Security Administrator’s Guide”和“Oracle8i Distributed Database System”。

## 操作系统身份验证

缺省时，Oracle 只允许操作系统验证的登录通过安全连接。因此，如果想让操作系统验证一个用户，那么缺省时用户不能通过 Net8 连接到数据库。这意味着该用户不能使用多线程服务器，因为这种连接使用了 Net8。这一缺省限制防止一个远程用户扮演另一个通过网络连接的操作系统用户。

如果不关心远程用户扮演另一个通过网络连接的操作系统用户。如果想要使用有网络客户的操作系统用户，将数据库初始化参数文件中的 REMOTE\_OS\_AUTHENT 设置为 TRUE（缺省为 FALSE）。将初始化参数 REMOTE\_OS\_AUTHENT 设置为 TRUE，则允许 RDBMS 接受通过非安全连接的客户端操作系统用户的帐号访问。这个改变将在下次启动实例或连接数据库时生效。

通常，通过主操作系统进行用户身份验证具有以下好处：

- 用户不用再专门指定数据库用户名和口令，可以更方便和快速地连接到 Oracle。
- 用户在数据库中和操作系统中的审计踪迹（audit trails）相对应。

## 网络身份验证

网络身份验证通过 Oracle 高级安全措施（Advanced Security）完成，还可以配置使用第三方服务器，如 Kerberos。如果使用 Oracle 安全措施作为唯一的外部身份验证服务器，那么参数 REMOTE\_OS\_AUTHENT 的设置就不重要了，因为 Oracle 高级安全措施只允许安全连接。

## 外部身份验证的优点

下面是外部身份验证的优点：

- 对于身份验证机制可以有更多的选择，如智能卡、指纹、Kerberos 或操作系统。
- 许多网络服务器，如 Kerberos 和 DCE，支持一次注册，这就意味用户可以少记一些口令。
- 如果已经使用了上面列出的一种外部身份验证机制，那么同使用数据库身份验证机制相比就可以有较少的管理开销。

## 22.2.3 全局身份验证和授权

Oracle 高级安全措施允许在基于 LDAP 的目录服务器中集中管理用户相关的信息，包括授权。用户在数据库中可以被标识为“全局用户”（global users），这意味着他们可以由 SSL 进行身份验证，同时，这些用户的管理由数据库外的中央目录服务器完成。“全局用户”由数据库定义，且只有该数据库知道，但是对这种角色的授权由目录服务器完成。

**注意** 用户也可以由 SSL 进行身份验证，其授权不在目录中管理；他们只有本地数据库的角色，更详细的信息见“Oracle 高级安全管理员指南”。

集中管理可以创建“企业用户（enterprise users）”和“企业角色（enterprise roles）”。企业用户的定义和管理均由目录服务器完成。在企业间他们有唯一的身份，同时可以被授予企业角色以确定他们在多个数据库间的访问权限。一个企业角色由一个或多个全局角色组成，可以把它看作一个全局角色的容器。

## 创建由目录服务器授权的用户

指定用户由目录服务器如何授权有两种选择。

创建全局用户：下面的语句示范了如何创建一个由 **SSL** 进行身份验证，由企业目录服务器授权全局用户：

```
CREATE USER scott
  IDENTIFIED GLOBALLY AS 'CN=scott,OU=division1,O=oracle,C=US'
```

在 **AS** 子句的字符串中为企业目录提供一个有意义的标识符（**DN**，或易区分的名字）。

在这种情况下，**SCOTT** 是一个真实的全局用户，但是，下面用户 **SCOTT** 必须在他要访问的数据库的每个数据库中创建，并添加到目录中。

创建独立模式的用户：创建独立模式的用户允许多企业用户在数据库中访问共享的模式。一个独立模式的用户是：

- 由 **SSL** 进行身份验证
- 不在数据库中使用任何类型的 **CREATE USER** 语句创建
- 权限的管理在目录中的用户
- 连接到共享模式的用户

创建独立模式用户的过程如下：

1). 像下面这样在数据库中创建一个共享模式：

```
CREATE USER appschema IDENTIFIED GLOBALLY AS '';
```

2). 现在在目录中创建多企业用户以及一个映射对象。

这个映射对象告诉数据库用户想如何映射用户的 **DN's** 到共享对象。你可以做一个完全的 **DN** 映射（一个目录对应一个唯一的 **DN**），也可以映射每一个包含 **DN** 部件的用户到 **APPSHEMA**：

```
OU=division,O=Oracle,C=US
```

关于这些映射的更多解释见 “**Oracle Internet Directory Administrator's Guide**”。

大多数用户不需要他们自己的模式，并将独立模式用户同数据库用户分离。将在一个数据库中创建的共享同一个模式的多用户作为企业用户，他们也能在其它数据库中访问共享模式。

## 全局身份验证和全局用户授权的优点

下面是全局用户身份验证和授权的一些优点：

- 可以提供使用 **SSL** 或 **NT** 强大的身份验证机制。
- 可以集中管理跨企业用户和权限。
- 更易于管理 不必为每一个用户在企业中的每一个数据库中创建一个模式。
- 方便的单次登录 用户登录一次就可以访问多个数据库和服务。
- **CURRENT\_USER** 数据库连接到全局用户。一个本地用户能够在存储过程的上下文中作为全局用户连接——在连接的定义中不需要存储这个全局用户的口令。

参见 关于全局身份验证和授权、企业用户和角色的更多内容参见下面几本书：

- “**Oracle Advanced Security Administrator's Guide**”



- “Oracle8i Distributed Database System”
- “Oracle Internet Directory Administrator's Guide”

#### 22.2.4 多层验证和授权

在一种安全方式中给代理客户设计一个应用服务器（中间层）是可能的。应用服务器代表客户进行连接时不必知道客户的口令。只要客户通过中间层验证，中间层通过数据验证，那么中间层就被系统管理员授权代表该客户，这个客户的身份可以被永远保留在数据库中而不损害客户的安全。

应用服务器必须是为这一功能设计。OCI（Oracle Call Interface）提供这种能力，这将在“Oracle 调用接口程序员指南”讲解。

将应用服务器授权给代理客户使用 `ALTER USER` 语句的 `GRANT CONNECT THROUGH` 子句。可以给应用服务器指定角色，当它作为客户连接被启动时。下面的例子授权应用服务器 APPSERVE 作为用户 BILL 连接，并允许 APPSERVE 激活除了 PAYROLL 以外 BILL 具有的全部角色。

```
ALTER USER bill
GRANT CONNECT THROUGH appserv
WITH ROLE ALL EXCEPT payroll;
```

使用 `REVOKE CONNECT THROUGH` 子句取消代理连接。

可以通过查询 `PROXY_USERS` 视图可以查看当前有那些用户被授权可以通过代理连接。

通过应用服务器代表客户的操作可以被审计，参见第 24 章的“在多层环境中的审计”一节。

参见 关于多层验证和授权的信息，参见“Oracle8i Concepts”和“Oracle8i Application Developer's Guide - Fundamentals”。

关于 `ALTER USER` 的“proxy clause”的语法和描述，参见“Oracle8i SQL Reference”。

### 22.3 Oracle 用户

每一个 Oracle 数据库都有一个有效的数据库用户列表。要访问数据库，一个用户必须运行一个数据库应用并使用一个在数据库中定义的有效用户名连接到数据库实例。这部分将解释如何管理数据库用户，包括以下几个部分：

- 创建用户
- 修改用户
- 删除用户

参见 关于出现在以下部分中的管理用户的 SQL 语句的更多信息，参见“Oracle8i SQL Reference”。

### 22.3.1 创建用户

要创建数据库用户，必须有 **CREATE USER** 系统权限。当创建一个新用户时，即使创建者在指定表空间上没有限额，也可以在数据库的表空间上指定表空间限额。因为它是非常强大的权限，通常一个安全系统管理员是唯一具有 **CREATE USER** 系统权限的用户。

使用 SQL 语句中的 **CREATE USER** 语句创建用户。下面的例子创建了一个用户，指定了用户的口令、缺省表空间、存放临时片段的临时表空间、表空间限额和概要文件。

```
CREATE USER jward
  IDENTIFIED BY AIRPLANE
  DEFAULT TABLESPACE data_ts
  TEMPORARY TABLESPACE temp_ts
  QUOTA 100M ON test_ts
  QUOTA 500K ON data_ts
  PROFILE clerk;
GRANT connect TO jward;
```

一个新建的用户在被授予 **CREATE SESSION** 系统权限前不能连接到数据库。通常给一个新建的用户授予一个角色，该角色指出访问一个数据库所需的 **CREATE CONNECT** 权限和其它基本权限，类似于预定义的 **CONNECT**，如该例所示。参见第 23 章的 23.3.1 节“授予系统权限和角色”获取具体信息。

#### 指定名称

每个数据库内用户名必须是独一无二的，不同其他用户和角色，一个用户和一个角色不能同名，而且每个用户都有一个相关联的模式。一个模式内的每个模式对象都有一个独一无二的名称。

#### 设置用户的身份验证

在前面的 **CREATE USER** 语句中，新用户必须经过身份验证方可使用数据库，因此，正在连接的用户必须输入正确的口令，才能成功连接到数据库。

指明用户类型身份验证的方法早在本章第 2 节“用户身份验证”中就介绍过了。

#### 分配缺省表空间

每个用户应该有一个缺省表空间，当用户创建一模式对象并且没有为它指定表空间时，Oracle 将该对象存储在用户的缺省表空间内。

每个用户的缺省表空间的缺省设置是 **SYSTEM** 表空间。如果用户不创建对象，缺省设置则没有问题，但是，如果用户创建任何类型的对象，则必须特别考虑设置用户缺省表空间。你可在创建用户时设置该用户的缺省表空间，在以后更改，更改用户的缺省表空间只影响设置更改后创建的对象。

确定指定那个表空间时请考虑下列问题：

- 只有当具有创建对象（比如表、视图和簇）的权限时才设置用户的缺省表空间。
- 将用户的缺省表空间设置为该用户有配额的表空间。
- 如果可能，将用户的缺省表空间设置为 **SYSTEM** 表空间以外的表空间，减少数据

字典对象和用户对象对同一数据文件的争用。

在前面的 **CREATE USER** 语句中，**JWARD** 的缺省表空间是 **DATA\_TS**。

### 分配临时表空间

也可以分配给每个用户一个临时表空间。当用户执行一条需要临时数据段的 **SQL** 语句时，**Oracle** 将该数据段存储在用户的临时表空间里，你不必为临时表空间设置配额。这些临时的数据段在做排序和连接时由系统创建并归 **SYS** 所有，**SYS** 在所有表空间内具有资源权限。

如果不显式地设置用户的缺省表空间，那么其缺省表空间是 **SYSTEM** 表空间。为每个用户设置临时表空间减少了临时数据段和其它数据段间的文件争用，因为临时表空间由临时数据段独占。你可在创建用户时设置用户的临时表空间，在以后更改。

在前面的 **CREATE USER** 语句中，**JWARD** 的临时表空间是 **TEMP\_TS**，它是一个显式创建的、只包含临时数据段的表空间。

### 分配表空间配额

你可给每个用户分配一个表空间配额，但对临时表空间来说，配额不是必须的。分配一个配额产生两个事件：

- 具有创建某种对象类型权限的用户可以在指定的表空间内创建这些对象；
- **Oracle** 限制指定表空间内用来存储用户对象的空间数量，不能超过配额。

缺省情况下，用户在数据库的任何表空间上没有配额。如果用户有权创建一模式对象，你必须给用户一个配额，允许用户创建对象，最少分配给一个缺省表空间的配额和创建对象所需的额外表空间配额。

你可给用户分配一个单独的配额，指定每个表空间中磁盘空间的数量，或者在所有表空间中没有限制，指定配额防止用户对象消耗太多的数据库空间。

可在创建用户时给该用户分配表空间配额，在以后增加或更改。如果新的配额少于旧的配额，那么下列条件为真：

- 如果用户使用的空间已超越新的表空间配额，那么不再给表空间中的用户对象分配空间，直到这些对象的联合空间少于新的配额。
- 如果用户使用的空间不超过新的表空间配额，或者如果表空间中用户的对象使用的空间少于新的表空间配额，那么可给用户的对象分配空间，直到达到新的配额为止。

**取消对表空间的访问** 你可通过将用户当前的配额改为 0 来取消用户对表空间的访问，配额变为 0 后，表空间中的用户的对象仍保留，但不再给对象分配任何新的空间。

**UNLIMITED TABLESPACE 系统权限** 授权用户 **UNLIMITED TABLESPACE** 系统权限可允许用户无限制地使用数据库中的任何表空间，用户可无视任何显式定义的配额，如果你以后取消这个权限，显式分配的配额又开始生效，你只可将该权限授权给用户，而不是角色。

授权 **INLIMITED TABLESPACE** 系统权限之前，考虑这样做的后果：

优点

- 你可只用一条语句授权用户无限制地访问数据库的所有表空间。

缺点

- 该权限无视所有显式地分配给用户的表空间配额;
- 你不可以使用 **UNLIMITED TABLESPACE** 权限有选择地取消用户对表空间的访问,只是在取消该权限后有选择地授权访问权限。

#### 指定概要文件 (Profile)

创建用户时也可指定 **Profile**。**Profile** 文件是一组数据库资源限制,如果不指定 **Profile**,指派给用户一个缺省 **Profile**。有关 **Profile** 的信息参见 22.4 节“使用概要文件管理资源”和第 21 章的 21.4 节“口令管理策略”。

#### 设置缺省角色

不可在 **CREATE USER** 语句中设置用户的缺省角色。第一次创建某个用户时,它的缺省角色设置为 **ALL**,这样随后授权给该用户的所有角色都是缺省角色。使用 **ALTER USER** 语句更改用户的缺省角色,参见第 23 章 23.5.2 节“指定缺省角色”。

**警告** 当你创建一个角色(而不是用户角色)时,该角色被隐式地授权给你并增添为缺省角色。如果你拥有的角色数超过 **MAX\_ENABLED\_ROLES** 指定的数,那么你在登录时将收到一条错误信息,改变用户缺省角色的个数,使其少于 **MAX\_ENABLED\_ROLES** 指定的数,这样则可避免错误的发生,因此,你应该在创建用户角色之前改变 **SYS** 和 **SYSTEM** 的 **DEFAULT ROLE** 设置。

### 22.3.2 改变用户

用户可以改变自己的口令,但是,你必须具有 **ALTER USER** 的系统权限才能改变用户安全域的其它选项,通常安全管理员是唯一具有该系统权限的用户,因为它可以对任何用户的安全域进行修改。该权限包括为用户在数据库的任何表空间内设置表空间配额,即使该用户执行修改操作时没有配额给指定的表空间。

可使用 **SQL** 语句 **ALTER USER** 改变用户的安全设置,对用户安全设置的改变只影响用户将来的会话,并不影响目前的会话。

下面语句用来改变用户 **AVYRROS** 的安全设置:

```
ALTER USER avyrros
  IDENTIFIED EXTERNALLY
  DEFAULT TABLESPACE data_ts
  TEMPORARY TABLESPACE temp_ts
  QUOTA 100M ON data_ts
  QUOTA 0 ON test_ts
  PROFILE clerk;
```

该 **ALTER USER** 语句所改变的 **AVYRROS** 的安全设置如下所示:

- 身份验证改变为使用 **AVYRROS** 的操作系统帐号;
- 显式设置 **AVYRROS** 的缺省表空间和临时表空间;
- 在 **DATA\_TS** 表空间上给予 **AVYRROS** 100M 的配额;
- 取消 **AVYRROS** 在 **TEST\_TS** 表空间上的配额;

■ 将 CLERK 概要文件指派给 AVYRROS。

#### 改变用户的身份验证机制

大多数非 DBA 用户可以使用 ALTER USER 语句改变自己的口令，如下所示：

```
ALTER USER andy  
IDENTIFIED BY swordfish;
```

用户可以用这种方法改变他们自己的口令，不再需要其它特殊的权限（连接到数据库的用户除外），建议用户经常改变自己的口令。

用户必须具有 ALTER USERS 权限才能在 Oracle 数据库身份验证间或外部的身份验证表格间进行切换，一般来说，只有 DBA 才有这个权限。

#### 改变用户的缺省角色

缺省角色是用户创建会话时自动为该用户启用的角色。你可以指派 0 个或更多个缺省角色给用户。有关改变用户缺省角色的更多信息参见第 23 章“用户权限管理和角色管理”。

### 22.3.3 删除用户

当一个用户被删除时，该用户和相关的模式都从数据字典和所有包含在该用户模式中的模式对象中删除，如果可能，立即删除。

**注意** 如果一个用户的模式和相关的对象必须保留而拒绝该用户访问数据库，则取消该用户的 CREATE SESSION 权限。

不可以删除正与数据库连接的用户。为了删除一个正在连接的用户，必须先使用带有 KILL SESSION 子句选项的 ALTER SYSTEM 语句终止用户的会话。有关中止会话的信息，参见第 4 章的 4.6 节“终止会话”。

可以使用 DROP USER 语句删除数据库中的用户，你必须具有 DROP USER 的系统权限才能删除一个用户和所有相关的对象。由于 DROP USER 系统权限功能强大，所以安全管理员是唯一一个具有这个权限的用户。

如果该用户的模式包含任何模式对象，那么使用 CASCADE 选项删除该用户、所有相关的对象和该用户的表上的外部键。如果你没指定 CASCADE 选项并且该用户的模式包含对象，那么将返回一条错误信息，且该用户不被删除。删除一个其模式包含对象的用户之前，仔细分析该用户的模式包含哪些对象，在删除用户之前删除这些对象会有什么隐含结果，注意任何可能产生的后果。例如，如果你想删除拥有一个表的表用户，检查一下是否有视图或过程依赖该表。

下面语句删除用户 JONES 及其所有相关的对象，以及依赖于 JONES 所有的表的外部键：

```
DROP USER jones CASCADE;
```

## 22.4 使用概要文件管理资源

PROFILE 是一组对资源的限制，用户的 PROFILE 文件限制数据库的使用情况和实例使

用的资源。你可以给每个用户指定一个 **PROFILE** 文件，给所有没有指定 **PROFILE** 的用户指定一个缺省 **PROFILE**。为了使 **PROFILE** 产生效应，必须在整个数据库内调整资源限制。

本节阐述了 **PROFILE** 的管理，包括以下主题：

- 启用和禁止资源限制
- 创建 **PROFILE** 文件
- 指定 **PROFILE** 文件
- 改变 **PROFILE** 文件
- 使用合成限制
- 删除 **PROFILE** 文件

参见 有关管理 **PROFILE** 的 SQL 语句的更多信息，参见 “Oracle8i Reference”。

#### 23.4.1 启用和禁止资源限制

任何数据库授权的用户随时可创建、更改、删除一个 **PROFILE** 文件，或将 **PROFILE** 文件指定给用户，但只有该用户对关联数据库启用资源限制时才实施 **PROFILE** 中设置的资源限制操作。可以使用两个方法来启用或禁止资源限制的实施，后面两节将介绍。

你必须具有 **ALTER SYSTEM** 系统权限才能改变资源限制的实施。

##### 在开机前启用和禁止资源限制

如果数据库暂时被关掉，那么可以使用数据库初始化参数文件中的 **RESOURCE\_LIMIT** 初始化参数启用或禁止资源限制。该参数的有效值是 **TRUE**（允许实施）或 **FALSE**，缺省值是 **FALSE**。一旦对参数文件进行编辑，则必须重新启动数据库实例才能使参数文件生效。任何时刻启动一个实例，新的参数值都将允许或禁止资源限制的实施。

##### 打开数据库时启用和禁止资源限制

如果不能暂时关闭数据库或者资源限制参数必须暂时改变，你可以使用 SQL 语句 **ALTER SYSTEM** 启动或禁止资源限制的实施，实例启动之后，**ALTER SYSTEM** 语句无视由 **RESOURCE\_LIMIT** 参数设置的值。例如，下面语句实施对数据库的资源限制：

```
ALTER SYSTEM
SET RESOURCE_LIMIT = TRUE;
```

**注意** 这个操作不适用于口令资源。

**ALTER SYSTEM** 语句不可永久性地实施资源限制，如果关闭数据库再启动，资源限制的实施由 **RESOURCE\_LIMIT** 参数设置的值来决定。

#### 22.4.2 创建 **PROFILE**

你必须具有 **CREATE PROFILE** 系统权限才能创建 **PROFILE**，使用 SQL 语句 **CREATE PROFILE** 创建概要文件，同时用户可显式地设置特殊的资源限制。

下面语句创建概要文件 **CLERK**：

```
CREATE PROFILE clerk LIMIT
SESSIONS_PER_USER 2
CPU_PER_SESSION unlimited
```

```
CPU_PER_CALL 6000
LOGICAL_READS_PER_SESSION unlimited
LOGICAL_READS_PER_CALL 100
IDLE_TIME 30
CONNECT_TIME 480;
```

新 **PROFILE** 中所有没有特地指明的资源限制取 **DEFAULT PROFILE** 中设置的值。

每个数据库都有一 **DEFAULT** 概要文件，在下面两种情况下使用 **DEFAULT** 中设置的缺省值：

- 如果用户没有显式地分配 **PROFILE**，那么该用户遵守 **DEFAULT** 概要文件中所有限制；
- 任何 **PROFILE** 中没指定的限制均使用 **DEFAULT PROFILE** 中相应的值。

开始时，**DEFAULT PROFILE** 的所有限制都设置为 **NOLIMITED**，但是，为阻止用户无限制地消耗资源，安全管理员应使用 **ALTER PROFILE** 语句 改变缺省设置：

```
ALTER PROFILE default LIMIT
```

任何具有 **ALTER PROFILE** 系统权限的用户可调整 **DEFAULT PROFILE** 中的限制，不可删除 **DEFAULT PROFILE**。

#### 22.4.3 分配 PROFILE

创建一个 **PROFILE** 之后，可以将它分配给数据库用户。任何时刻一个用户只能分配一个 **PROFILE**。如果将一个 **PROFILE** 分配给一个已有 **PROFILE** 的用户，新分配的 **PROFILE** 将覆盖旧的 **PROFILE**，但并不影响当前会话，只可将 **PROFILE** 分配给用户，不能分配给角色或其它 **PROFILE**。

使用 SQL 语句 **CREATE USER** 或 **ALTER USER** 将 **PROFILE** 分配给用户。有关分配一个 **PROFILE** 给用户的信息，参见 22.3.1 节“创建用户”和 22.3.2 节“改变用户”。

#### 22.4.4 改变 PROFILE

使用 SQL 语句 **ALTER PROFILE** 改变任何 **PROFILE** 对资源限制的设置，你必须具有 **ALTER PROFILE** 的系统权限才能改变 **PROFILE**。

任何改变后的 **PROFILE** 都覆盖以前 **PROFIEL** 中的资源限制。通过调整 **DEFAULT** 的设置，改变对数据库的缺省设置，其它 **PROFILE** 仍保留原来的设置。对某个 **PROFILE** 的更改并不影响当前会话，只是创建会话时才使用新的 **PROFILE** 设置。

下面语句改变 **CLERK** 的 **PROFILE**：

```
ALTER PROFILE clerk LIMIT
CPU_PER_CALL default
LOGICAL_READS_PER_SESSION 20000;
```

#### 22.4.5 使用合成限制

可以通过合成限制来限制总的会话资源成本。除了显式地对一个 **PROFILE** 设置指定的资源限制，你还可以设置一个合成限制，使用 SQL 语句 **CREATE PROFILE** 或 **ALTER PROFILE** 的 **COMPOSITE**——**LIMIT** 子句设置合成资源限制。通过服务单元（service units）

设置合成限制，服务单元是每个资源的加权值。

下面 **CREATE PROFILE** 语句的定义中使用了 **COMPOSITE\_LIMIT** 子句：

```
CREATE PROFILE clerk LIMIT
  COMPOSITE_LIMIT 20000
  SESSIONS_PER_USER 2
  CPU_PER_CALL 1000;
```

显式资源限制和合成限制可同时在一个 **PROFILE** 中并存。第一个到达的限制是对会话的限制，在对系统资源的限制中合成限制更具灵活性。

#### 确定合成限制的值

正确的合成限制依赖于一个普通 **PROFILE** 用户所使用的总的资源。通过每个指定的资源限制和历史信息确定一个典型 **PROFILE** 用户所使用的合成资源限制的一般范围。

参见 有关如何计算合成限制的信息，参见 “Oracle8i SQL Reference”。

#### 设置资源价值 (resource costs)

每个系统都有自身的特点，某些系统资源可能比其它资源更有价值，Oracle 允许你给每个系统一个价值 (cost)，对数据库层的每个系统资源进行价值加权，价值只应用于 **PROFILE** 的合成限制，不适合于显式地设置单个的资源限制。

你必须具有 **ALTER RESOURCE** 系统权限才能设置资源价值。

只有某些资源才能给出价值：**CPU\_PER\_SESSION**、**LOGICAL\_READS\_PER\_SESSION**、**CONNECT\_TIME** 和 **PRIVATE\_SGA**。使用 SQL 语句 **ALTER RESOURCE COST** 设置数据库的价值：

```
ALTER RESOURCE COST
  CPU_PER_SESSION 1
  LOGICAL_READS_PER_SESSION 50;
```

价值高意味着该资源非常昂贵，价值低意味着该资源不太昂贵。缺省情况下，每个资源的初始价值为 0，价值为 0 意味着该资源不在合成限制考虑之中（也就是说，使用该资源不花费任何成本）。不能将资源的价值设置为 **NULL**。

参见 有关设置资源价值的其它信息和建议，参见用户操作系统指定的 Oracle 资料和 “Oracle8i SQL Reference”。

### 22.4.6 删除 PROFILE

你必须具有 **DROP PROFILE** 系统权限才能删除一个 **PROFILE**，使用 SQL 语句 **DROP PROFILE** 删除一个 **PROFILE**，使用 **CASCADE** 选项成功地删除一个分配给用户的 **PROFILE**。

下面语句删除 **CLERK** 概要文件，即使它已分配给了用户：

```
DROP PROFILE clerk CASCADE;
```

自动将 **DEFAULT** 概要文件分配给与被删除 **PROFILE** 关联的用户，**DEFAULT** 概要文件不可删除。当一个 **PROFILE** 被删除时，删除过程并不影响当前活动的会话，只影响以后创建的会话。



## 22.5 列举数据库用户和 PROFILE 的相关信息

数据字典存储了每个用户和 **PROFILE** 的相关信息，包括：

- 一个数据库中的所有用户
- 每个用户存放表、簇和索引的缺省表空间
- 每个用户存放临时数据段的表空间
- 每个用户的空间配额，如果必需
- 每个用户指定的 **PROFILE** 和资源限制
- 每个可用的系统资源的价值
- 每个会话的内存使用情况

当你操作数据库用户和 **PROFILE** 文件时应关心下列数据字典视图：

| 视图                   | 描述                                       |
|----------------------|--|
| ALL_USERS            | 列举当前可见的所有数据库用户。该视图并不描述用户，参见相关视图          |
| DBA_USERS            | 描述数据库的所有用户                               |
| USER_USERS           | 和 DBA_USERS 相同的列，但只描述当前用户                |
| DBA_TS_QUOTAS        | 描述所有用户的表空间配额                             |
| USER_TS_QUOTAS       | 和 DBA_TS_QUOTAS 相同的列，但只包含当前用户的表空间配额的相关信息 |
| USER_PASSWORD_LIMITS | 描述 <b>PROFILE</b> 中指定给用户的口令              |
| USER_RESOURCE_LIMITS | 显示对当前用户的资源限制                             |
| DBA_PROFILES         | 显示所有 <b>PROFILE</b> 文件及其限制               |
| RESOURCE_COST        | 列举每个资源的价值                                |
| V\$SESSION           | 列举每个当前会话的信息，包括用户名                        |
| V\$SESSTAT           | 列举用户会话的统计信息                              |
| V\$STATNAME          | 显示 V\$SESSTAT 视图中解密的统计信息名称               |
| PROXY_USER           | 该视图描述能识别其他用户的用户                          |

参见 有关每个视图的详细信息参见 “Oracle8i Reference”。

### 22.5.1 列举用户和 PROFILE 的相关信息：示例

本节的例子假设已经在数据库中执行了下列语句：

```
CREATE PROFILE clerk LIMIT
  SESSIONS_PER_USER 1
  IDLE_TIME 30
  CONNECT_TIME 600;

CREATE USER jfee
  IDENTIFIED BY wildcat
  DEFAULT TABLESPACE users
  TEMPORARY TABLESPACE temp_ts
```

```

        QUOTA 500K ON users
    PROFILE clerk;
CREATE USER dcranney
    IDENTIFIED BY bedrock
    DEFAULT TABLESPACE users
    TEMPORARY TABLESPACE temp_ts
    QUOTA unlimited ON users;
CREATE USER userscott
    IDENTIFIED BY scott1;

```

### 列举所有用户及其相关信息

下面查询语句列举数据库中定义的所有用户及其相关信息:

```

SELECT username, profile, account_status from dba_users;

```

| USERNAME  | PROFILE | ACCOUNT_STATUS |
|-----------|---------|----------------|
| SYS       | DEFAULT | OPEN           |
| SYSTEM    | DEFAULT | OPEN           |
| USERSCOTT | DEFAULT | OPEN           |
| JFEE      | CLERK   | OPEN           |
| DCRANNEY  | DEFAULT | OPEN           |

所有口令都译成密码以确保安全。如果用户查询 **PASSWORD** 列, 则该用户不能获知其他用户的口令。

### 列举所有表空间配额

下面查询语句列举分配给每个用户的表空间配额:

```

SELECT * FROM sys.dba_ts_quotas;

```

| TABLESPACE | USERNAME | BYTES | MAX_BYTES | BLOCKS | MAX_BLOCKS |
|------------|----------|-------|-----------|--------|------------|
| USERS      | JFEE     | 0     | 512000    | 0      | 250        |
| USERS      | DCRANNEY | 0     | -1        | 0      | -1         |

**MAX\_BYTES** 列指出配额的准确数, 该数值必须是数据库块大小的倍数, 如果一个表空间的配额不是数据库块大小的倍数, 那么按照四舍五入的原则进位, 无限制的配额指定为“-1”。

### 列举所有 PROFILE 和指定的限制

下面查询语句列举数据库中所有的 **PROFILE** 和每个 **PROFILE** 中的每个限制的相关设置:

```

SELECT * FROM sys.dba_profiles
    ORDER BY profile;

```

| PROFILE | RESOURCE_NAME         | RESOURCELIMIT    |
|---------|-----------------------|------------------|
| CLERK   | COMPOSITE_LIMIT       | KERNEL DEFAULT   |
| CLERK   | FAILED_LOGIN_ATTEMPTS | PASSWORD DEFAULT |

|         |                           |          |           |
|---------|---------------------------|----------|-----------|
| CLERK   | PASSWORD_LIFE_TIME        | PASSWORD | DEFAULT   |
| CLERK   | PASSWORD_RESUE_TIME       | PASSWORD | DEFAULT   |
| CLERK   | PASSWORD_REUSE_MAX        | PASSWORD | DEFAULT   |
| CLERK   | PASSWORD_VERIFY_FUNCTION  | PASSWORD | DEFAULT   |
| CLERK   | PASSWORD_LOCK_TIME        | PASSWORD | DEFAULT   |
| CLERK   | PASSWORD_GRACE_TIME       | PASSWORD | DEFAULT   |
| CLERK   | PRIVATE_SGA               | KERNEL   | DEFAULT   |
| CLERK   | CONNECT_TIME              | KERNEL   | 600       |
| CLERK   | IDLE_TIME                 | KERNEL   | 30        |
| CLERK   | LOGICAL_READS_PER_CALL    | KERNEL   | DEFAULT   |
| CLERK   | LOGICAL_READS_PER_SESSION | KERNEL   | DEFAULT   |
| CLERK   | CPU_PER_CALL              | KERNEL   | DEFAULT   |
| CLERK   | CPU_PER_SESSION           | KERNEL   | DEFAULT   |
| CLERK   | SESSIONS_PER_USER         | KERNEL   | 1         |
| DEFAULT | COMPOSITE_LIMIT           | KERNEL   | UNLIMITED |
| DEFAULT | PRIVATE_SGA               | KERNEL   | UNLIMITED |
| DEFAULT | SESSIONS_PER_USER         | KERNEL   | UNLIMITED |
| DEFAULT | CPU_PER_CALL              | KERNEL   | UNLIMITED |
| DEFAULT | LOGICAL_READS_PER_CALL    | KERNEL   | UNLIMITED |
| DEFAULT | CONNECT_TIME              | KERNEL   | UNLIMITED |
| DEFAULT | IDLE_TIME                 | KERNEL   | UNLIMITED |
| DEFAULT | LOGICAL_READS_PER_SESSION | KERNEL   | UNLIMITED |
| DEFAULT | CPU_PER_SESSION           | KERNEL   | UNLIMITED |
| DEFAULT | FAILED_LOGIN_ATTEMPTS     | PASSWORD | UNLIMITED |
| DEFAULT | PASSWORD_LIFE_TIME        | PASSWORD | UNLIMITED |
| DEFAULT | PASSWORD_REUSE_MAX        | PASSWORD | UNLIMITED |
| DEFAULT | PASSWORD_LOCK_TIME        | PASSWORD | UNLIMITED |
| DEFAULT | PASSWORD_GRACE_TIME       | PASSWORD | UNLIMITED |
| DEFAULT | PASSWORD_VERIFY_FUNCTION  | PASSWORD | UNLIMITED |
| DEFAULT | PASSWORD_REUSE_TIME       | PASSWORD | UNLIMITED |

32 rows selected.

### 查看每个用户会话的内存使用情况

下面查询语句列举了当前所有会话，显示了 Oracle 用户和每个会话当前 UGA（用户全局区域）内存使用情况：

```
SELECT username, value || 'bytes' "Current UGA memory"
FROM v$session sess, v$sesstat stat, v$statname name
WHERE sess.sid = stat.sid
AND stat.statistic# = name.statistic#
AND name.name = 'session uga memory';
```

| USERNAME | Current UGA memory |
|----------|--------------------|
| -----    | -----              |
|          | 18636bytes         |
|          | 17464bytes         |

```

19180bytes
18364bytes
39384bytes
35292bytes
17696bytes
15868bytes
USERSCOTT 42244bytes
SYS        98196bytes
SYSTEM     30648bytes
11 rows selected.

```

查看实例启动来分配每个会话的最大 UGA 内存，将上面查询语句中的 “session uga memory” 替换为 “session uga memory max”。

## 22.6 示例

下面是一些使用本章中讨论的 SQL 语句和视图的例子。

1. 下面语句创建概要文件 PROF:

```

CREATE PROFILE prof LIMIT
  FAILED_LOGIN_ATTEMPTS 5
  PASSWORD_LIFE_TIME 60
  PASSWORD_REUSE_MAX 60
  PASSWORD_REUSE_TIME UNLIMITED
  PASSWORD_VERIFY_FUNCTION verify_function
  PASSWORD_LOCK_TIME 1
  PASSWORD_GRACE_TIME 10;

```

2. 下面语句尝试创建一个具有相同的口令和用户名的用户:

```

CREATE USER userscott IDENTIFIED BY userscott PROFILE prof;
ORA-28003: Password verification for the specified password failed
ORA-20001: Password same as user

```

3. 下面语句创建一个标识为 SCOTT%、概要文件为 PROF 的 USERSCOTT 用户:

```

CREATE USER userscott IDENTIFIED BY "scott%" PROFILE prof;

```

4. 下面语句将用户的口令改变为 “SCOTT%”:

```

ALTER USER userscott IDENTIFIED BY "scott%";
ORA-28007: The password cannot be reused

```

5. 下面语句锁定用户帐号:

```

ALTER USER userscott ACCOUNT LOCK;

```

6. 下面语句检查用户帐号状态:

```

SELECT username, user_id, account_status, lock_date
  FROM dba_users
 WHERE username = 'USERSCOTT';

```

| USERNAME | USER_ID | ACCOUNT_STATUS | LOCK_DATE |
|----------|---------|----------------|-----------|
| -----    | -----   | -----          | -----     |

|           |    |        |           |
|-----------|----|--------|-----------|
| USERSCOTT | 38 | LOCKED | 11-OCT-99 |
|-----------|----|--------|-----------|

7. 下面语句给用户开锁:

```
ALTER USER userscott ACCOUNT UNLOCK;
```

8. 下面语句终止口令:

```
ALTER USER userscott PASSWORD EXPIRE;
```

# 23 用户权限管理和角色管理

本章说明如何使用权限和角色来控制对模式对象的访问以及对系统操作的执行能力，包括以下内容：

- 识别用户权限
- 管理用户角色
- 授予用户权限和角色
- 取消用户权限和角色
- 授予权限和取消权限何时开始生效
- 运用操作系统或运用网络授权
- 列举权限和角色信息

有关对数据库访问的控制，参见第 22 章“用户和资源管理”；有关数据库安全策略的建议，参见第 21 章“建立安全策略”。

## 23.1 识别用户权限

本节阐述了 Oracle 用户权限，包括以下内容：

- 系统权限
- 对象权限

用户权限是执行某一 SQL 语句或访问另一用户的对象的权力，Oracle 还提供成组的权限的快捷方式，通常可以一起授权或取消。

### 23.1.1 系统权限

Oracle 有 100 种不同的系统权限，每一种系统权限允许用户执行某一个数据库操作或某一类数据库操作。

**警告** 系统权限的权力非常强大，应十分小心地授权给角色和数据库的值得信赖的用户。

有关系统权限的总览和描述，参见“Oracle8i SQL Reference”。

#### 系统权限约束

由于系统权限的权力非常强大，Oracle 建议配置数据库，防止正常用户（非 DBA）在数据字典中使用 ANY 系统权限。为了保证数据字典的安全，将 O7\_DICTIONARY\_ACCESSIBILITY 初始化参数设置为 FALSE，这个特征叫做数据字典保护机制。

**注意** 当你从 Oracle7 移植到 Oracle8i 时，O7\_DICTIONARY\_ACCESSIBILITY 初始化参数控制对 SYSTEM 权限的约束。如果该参数设置为 TRUE，则允许访问 SYS 模式中

的对象；如果设置为 FALSE，允许访问“任何”模式中对象的系统权限不允许访问 SYS 模式中的对象。缺省值是 TRUE。

如果你不显式地将该初始化参数设置为 FALSE，那么 ANY 权限将应用于数据字典，具有 ANY 权限的不怀好意的用户将访问或改变数据字典表。

有关该参数的更多信息参见“Oracle8i SQL Reference”，参见“Oracle8i Migration”以掌握该参数的使用。

如果你启用数据字典保护，那么只限于 SYS 模式中的用户可访问 SYS 模式中的对象，包括 SYS 用户和以 SYSDBA 身份连接的用户。提供访问其它模式中的对象的系统权限并不给其他用户访问 SYS 模式中的对象，例如，SELECT ANY TABLE 权限允许用户访问其他用户模式中的表和视图，但不允许他们选择字典对象（动态性能视图、视图、程序包和别名单元的基表），然而这些用户可显式地授权访问 SYS 模式中的对象。

### 经常访问有效的字典对象

有明确对象权限或管理员（SYSDBA）权限的用户可访问字典对象，但是，如果其他用户需要访问字典对象并没有这些权限，那么可将以下角色授权给他们：

#### ■ SELECT\_CATALOG\_ROLE

使用户选择所有授权给该角色的输出目录（exported catalog）视图和表，将这个角色授权给必须访问数据字典中所有输出视图和表的用户。

#### ■ EXECUTE\_CATALOG\_ROLE

对数据字典中的输出程序包给予 EXECUTE 权限。

#### ■ DELETE\_CATALOG\_ROLE

允许用户删除 AUD\$ 表中的记录。

这些角色允许一些数据库管理员维护数据字典安全时访问数据字典中的某些对象。

**注意** 你不应将数据字典中的非输出对象的对象权限授权给任何用户，这样做可能损害数据库的完整性。

**参见** 有关输出表或视图的详细信息，参见“Oracle8i Reference”。

## 23.1.2 对象权限

每一类对象都有与之关联的不同的权限，有关对象和相关权限的详细清单参见“Oracle8i SQL Reference”。

用户可指定 ALL 为某一对象授予或取消所有有效对象权限，ALL 不是一个权限，它是一个快捷键，或是一种方式，在 GRANT 语句或 REVOKE 语句中用一个单词授予或取消所有对象权限。记住如果用 ALL 快捷键授权所有对象权限，仍可取消单个权限。

同样地，所有单个授权的权限也可通过指定 ALL 取消，但是如果用户取消 ALL，且取消引起删除完整性约束（因为它们依赖于正被取消的 REFERENCES 权限），则用户必须在 REVOKE 语句中包含 CASCADE CONSTRAINTS 选项。

## 23.2 管理用户角色

角色是将权限和角色组成组，这样可将它们同时授权给用户或同时从用户中取消，每个用户可启用或禁用角色。

本节阐述了角色的管理，包括：

- 预定义角色
- 创建角色
- 角色授权
- 删除角色

参见 有关角色的信息，参见 “Oracle8i Concepts”。

### 23.2.1 预定义角色

当用户运行标准脚本（这些脚本是数据库创建的一部分）时，Oracle 数据库自动定义表 23-1 中列举的角色，用户可为这些预定义角色授权权限和角色，其方式类似角色定义。

表 23-1 预定义角色

| 角色名   | 创建者（脚本）    | 描述  |
|---|------------|---|
| CONNECT   | SQL.BSQ    | 包括下列系统权限: ALTER SESSION, CREATE CLUSTER, CREATE DATABASE LINK, CREATE SEQUENCE, CREATE SESSION, CREATE SYNONYM, CREATE TABLE , CREATE VIEW  |
| RESOURCE  | SQL.BSQ    | 包括下列系统权限: CREATE CLUSTER, CREATE INDEXTYPE, CREATE OPERATOR , CREATE PROCEDURE , CREATE SEQUENCE, CREATE TABLE, CREATE TRIGGER, CREATE TYPE   |
| DBA   | SQL.BSQ    | 所有带有 ADMIN OPTION 选项的系统权限   |
| 注意 前面三种角色用来维护与 Oracle 以前版本的兼容性，在 Oracle 以后版本中可能不再自动创建这些角色，Oracle 公司建议用户为数据库安全设计自己的角色，而不是依赖于这些角色 |            |   |
| EXP_FULL_ DATABASE  | CATEXP.SQL | 提供执行不断增长的整个数据库输出所需的权限，包括: SELECT ANY TABLE , BACKUP ANY TABLE , EXECUTE ANY PROCEDURE, EXECUTE ANY PROCEDURE, EXECUTE ANY TYPE , ADMINISTER RESOURCE MANAGER , 以及对表 SYS.INCVID、SYS.INCFIL 和 SYS.INCEXP 的 INSERT、DELETE 和 UPDATE，还包括下列角色: EXECUTE_CATALOG_ROLE 和 SELECT_CATALOG_ROLE |
| IMP_FULL_ DATABASE  | CATEXP.SQL | 提供执行整个数据库输入所需的权限，包括大量的系统权限（利用 DBA_SYS_PRIVS 视图查阅这些权限）和下列角色: EXECUTE_CATALOG_ROLE 和 SELECT_CATALOG_ROLE  |
| DELETE_CATALOG_ROLE   | SQL.BSQ    | 数据字典中针对该角色的所有程序包上的 DELETE 权限  |
| EXECUTE_CATALOG_ROLE  | SQL.BSQ    | 数据字典中针对该角色的所有程序包上的 EXECUTE 权限，还有 HS_ADMIN_ROLE  |
| SELECT_CATALOG_ROLE   | SQL.BSQ    | 针对该角色的所有目录表和视图的 SELECT 权限，还有 HS_ADMIN_ROLE  |



\*续表)

| 角色名                            | 创建者 (脚本)         | 描述   |
|--------------------------------|------------------|--|
| RECOVERY_<br>CATALOG_OW<br>NER | CATALOG.SQL      | 为恢复目录的所有者提供权限, 包括: CREATE SESSION, ALTER SESSION, CREATE SYNONYM, CREATE VIEW, CREATE DATABASE LINK, CREATE TABLE, CREATE CLUSTER, CREATE SEQUENCE, CREATE TRIGGER 和 CREATE PROCEDURE                            |
| HS_ADMIN_<br>ROLE              | CATHS.SQL        | 用来保护对 HS (Heterogeneous Services) 数据字典表 (授权 SELECT) 和程序包 (授权 EXECUTE) 的访问, 将其授权给 SELECT_CATALOG_ROLE 和 EXECUTE_CATALOG_ROLE, 这样能访问一般数据字典的用户也能访问 HS 数据字典已作废的, 只为与 8.0 兼容而保留。提供对 DBMS_AQ 和 DBMS_AQIN 的 EXECUTE 权限。 |
| AQ_USER_<br>ROLE               | CATQUEUE.SQ<br>L | 为管理高级排序 (Advance Queuing) 提供权限, 包括: ENQUEUE ANY QUEUE, DEQUEUE ANY QUEUE, MANAGE ANY QUEUE, 对 AQ 表的 SELECT 和对 AQ 程序包的 EXECUTE  |
| AQ_ADMINIST<br>RATOR_ROLE      | CATQUEUE.SQ<br>L | 企业管理者/智能代理商使用该角色, 包括 ANALYZE ANY 和对不同视图授权 SELECT   |
| SNMPAGENT                      | CATSNMP.SQL      |  |

如果用户安装了其它选项或产品, 则可能预定义其它角色。

### 23.2.2 创建角色

用户可使用 CREATE ROLE 语句创建一个角色, 但用户必须具有这样做的 CREATE ROLE 系统权限, 具有代表性的是, 安全管理员拥有该系统权限。

**注意** 角色刚创建完时没有任何权限与之相关联, 为使一个新的角色拥有相关的权限, 用户必须授权权限或角色给该新的角色。

用户必须给自己创建的每一个角色一个唯一的名称, 该名称应不同于数据库已存在的用户名和角色名, 角色不包含在任何用户的模式中。在使用多字节字符设置的数据库中, Oracle 建议每个角色的名称至少包含一个单字节字符, 如果一个角色的名称只包含多字节字符, 则加密的角色名称/口令相对来说不太安全。

下面语句创建了 CLERK 角色, 该角色由数据库使用 BICENTENNIAL 口令授权:

```
CREATE ROLE clerk  
IDENTIFIED BY bicentennial;
```

IDENTIFIED BY 子句指定了授权的方式, 该方式必须在角色启用之前由一指定用户 (已将角色授权给该用户) 提供。如果没有指定这个子句, 或者指定 NOT IDENTIFIED, 则启用该角色不需经过授权。可指定角色由数据库进行外部或全局授权, 这些授权将在下一节讨论。

随后, 用户可使用 ALTER ROLE 语句设置或改变角色授权方式。下面语句将 CLERK 角色改变为外部授权:

```
ALTER ROLE clerk
```

IDENTIFIED EXTERNALLY;

为了改变角色的授权方式，用户拥有 ALTER ANY ROLE 的系统权限或已用 ADMIN OPTION 授权该角色。

参见 有关管理角色和权限的 SQL 语句的语法、限制和授权信息，参见 “Oracle8i SQL Reference”。

### 23.2.3 角色授权

当用户试图启用角色时，数据库角色可有选择地进行授权，本节将讨论角色的授权方式，角色启用将在本章“授权和取消何时开始生效”一节中讨论。

#### 由数据库进行角色授权

可通过相关口令保护由数据库进行的角色授权。如果授权一个口令保护的角色给用户，用户可在 SET ROLE 语句中通过提供正确的口令启用或禁止一个角色，参见本章第 5 节中的“SET ROLE 语句”。但是，如果角色是一缺省角色并在连接时启用，则用户无需输入口令。

前面已介绍了对数据库授权指定角色的例子。

**注意** 在使用多字节设置的数据库中，角色的口令必须只包括单字节字符，不接受多字节字符。

参见 有关口令有效性信息，参见 “Oracle8i SQL Reference”。

#### 由操作系统进行角色授权

下面语句创建了一个名为 ACCTS\_REC 的角色并要求操作系统授权它的使用：

```
CREATE ROLE accts_rec IDENTIFIED EXTERNALLY;
```

只有当操作系统能够动态将操作系统权限与应用程序连接起来时，通过操作系统进行角色授权才显得非常有用。当用户开始运行某一应用程序时，操作系统将一操作系统权限授权给用户，授权的操作系统权限符合与应用程序相关联的角色，从这个意义上说，应用程序可启用应用程序角色，当应用程序终止时，以前授权的操作系统权限也从用户的操作系统帐号中取消。

如果由操作系统进行角色授权，则必须在操作系统级为每个用户配置正确信息，该操作随操作系统而定。

如果由操作系统授权角色，用户不必让操作系统进行角色授权，这将产生冗余。有关由操作系统授权的角色的信息，参见本章第 6 节“利用操作系统或网络授权角色”。

#### 角色授权和网络客户

如果用户通过 Net8 连接到数据库，则缺省情况下他们的角色不可由操作系统进行身份验证，这包括通过多线程服务器连接，因为这种连接需要 Net8。因为一远程用户可通过网络连接扮演另一个操作系统用户，所以这种限制是缺省值。

如果用户不关心由此带来的安全问题并希望为网络客户使用操作系统角色授权，那么将数据库参数文件中的 REMOTE\_OS\_ROLES 参数设置 TRUE，当用户下次启动该实例并登录数据库时这个改变将生效（该参数的缺省值是 FALSE）。

参见 有关网络角色的更多信息，参见“Oracle8i Distributed Database Systems”。

#### 由企业目录服务器进行角色授权

可将角色定义为全局角色，通过企业目录服务器只授权全球用户使用该角色，用户可通过授权权限和角色给本地数据库来定义一个全局角色，但用户不可将全局角色授权给数据库中的任何用户或其它角色。当一全球用户尝试连接到数据库时，查询企业目录，获得与该用户相关联的全局角色。

下面语句创建一全局角色：

```
CREATE ROLE supervisor IDENTIFIED GLOBALLY
```

全局角色是企业用户管理的一部分，一个全局角色只应用于一个数据库，但它可以授权给企业目录中定义的企业角色。一个企业角色是一个目录结构，包含多数据库中的全局角色，可以授权给企业用户。

有关全局角色身份验证、用户授权和企业用户管理中的角色的讨论在第 22 章的 22.2.3 节“全局身份验证和授权”中提供。

参见 有关企业用户管理以及如何改进它的信息，参见“Oracle Advanced Security Administrator's Guide”和“Oracle Internet Directory Administrator's Guide”。

#### 23.2.4 删除角色

在某些情况下，或许应该从数据库中删除一角色，删除角色授权的所有用户和角色的安全域会立即随着删除角色权限的消失而发生变化，所有删除间接授权的角色也将从受影响安全域中取消，删除一个角色将自动从所有用户的缺省角色中取消该角色。

因为对象的创建并不依赖通过角色接收的权限，所以当角色被删除时，表和其它对象并不会被删除。

使用 **DROP ROLE** 语句删除角色。为删除一角色，用户必须具有 **DROP ANY ROLE** 的系统权限或已经用 **ADMIN OPTION** 授权该角色。

下面语句删除 **CLERK** 角色：

```
DROP ROLE clerk;
```

### 23.3 授权用户权限和角色

本节阐述了授权用户权限和角色，包括：

- 授权系统权限和角色
- 授权对象权限和角色
- 对列授权权限

也有可能将角色授权给通过中介代理连接的用户，这已在第 22 章“多层身份验证和授权”一节中讨论过。

### 23.3.1 授权系统权限和角色

用户可使用 **GRANT** 语句授权系统权限和角色给其他角色和用户。为授权一系统权限或角色，用户必须对所授权的系统权限和角色具有 **ADMIN OPTION**，同样，任何具有 **GRANT ANY ROLE** 系统权限的用户可授权一个数据库中的任何角色。

**注意** 用户不可授权 **IDENTIFIED GLOBALLY** 的角色，全球角色的授权完全由企业目录服务器控制。

下面语句将 **CREATE SESSION** 系统权限和 **ACCTS\_PAY** 授权给用户 **JWARD**:

```
GRANT CREATE SESSION, accts_pay  
TO jward;
```

**注意** 在同一条 **GRANT** 语句中，不可将对象权限和系统权限、角色一起授权。

当用户创建一角色时，使用 **ADMIN OPTION** 方式自动将该角色授权给创建者，使用 **ADMIN OPTION** 的被授权者拥有几个扩展能力：

- 被授权者可授权系统权限或角色给数据库中的任何用户或其它角色，或从数据库中的任何用户和其他角色中取消系统权限（用户不可取消自己的角色。）
- 被授权者更可使用 **ADMIN OPTION** 授权系统权限或角色。
- 角色的被授权者可改变或删除角色。

下面语句中，安全管理员将 **NEW\_DBA** 角色授权给 **MICHAEL**:

```
GRANT new_dba TO michael WITH ADMIN OPTION;
```

用户 **MICHAEL** 不但可以使用隐含在 **NEW\_DBA** 中的所有权限，必要时还可授权、取消或删除 **NEW\_DBA** 角色。鉴于这些原因，使用 **ADMIN OPTION** 授权系统权限或角色时应十分小心，这种权限通常由安全管理员保留，很少授权给系统的其他管理员或用户。

### 23.3.2 授权对象权限和角色

使用 **GRANT** 语句授权对象权限给用户和角色，为授权一对象权限，用户必须满足下列条件之一：

- 用户拥有指定的对象
- 授予 **GRANT OPTION** 的对象权限已授权给用户

**注意** 在同一条 **GRANT** 语句中，不可将对象权限和系统权限、角色一起授权。

下面语句将 **EMP** 表所有列的 **SELECT**、**INSERT** 和 **DELETE** 对象权限授权给用户 **JFEE** 和 **TSMITH**:

```
GRANT SELECT, INSERT, DELETE ON emp TO jfee, tsmith;
```

只将 **EMP** 表的 **ENAME** 列和 **JOB** 列的 **INSERT** 对象权限授权给用户 **JFEE** 和 **TSMITH**，用下面语句实现：

```
GRANT INSERT(ename, job) ON emp TO jfee, tsmith;
```

将 **SALARY** 视图的所有对象权限授权给用户 **JFEE**，使用 **ALL** 关键字，如下例所示：

```
GRANT ALL ON salary TO jfee;
```

与 **GRANT OPTION** 关联的对象权限自动授权的模式包含一个对象的用户，这个特殊的权限给予被授权者几个扩展权限：

- 被授权者可将对象权限授权给数据库中的任何用户（带有或不带有 GRANT OPTION）或角色。
- 如果被授权者收到关于某一表的带有 GRANT OPTION 的对象权限并且被授权者拥有 CREATE VIEW 或 CREATE ANY VIEW 的系统权限，则被授权者可在该表上创建视图并将视图上相应的权限授权给数据库中的任何用户或角色。

授权一对象权限给角色时 GRANT OPTION 无效。

Oracle 防止通过角色传播对象权限，这样角色的被授权者不能传播角色收到的对象权限。

### 23.3.3 对列授权权限

用户可对表中单个授权 INSERT、UPDATE 或 REFERENCES 权限。

**警告** 授权一指定列 INSERT 权限之前，确保该表是否包含任何定义为 NOT NULL 的列。对不包括 NOT NULL 列授权 INSERT 权限可防止用户往表中插入任何行，为避免发生这种情况，应确保每个 NOT NULL 列是可插入的或有一非 NULL 缺省值，否则，被授权者不能往表中插入行并收到一条错误信息。

下面语句将对表 ACCOUNTS 的 ACCT\_NO 列的 INSERT 权限授权给 SCOTT:

```
GRANT INSERT (acct_no)
ON accounts TO scott;
```

## 23.4 取消用户权限和角色

本节阐述如何取消用户权限和角色，包括：

- 取消系统权限和角色
- 取消对象权限和角色
- 取消权限的效应
- 授权和取消 PUBLIC 用户组

### 23.4.1 取消系统权限和角色

使用 REVOKE 语句取消系统权限和角色。

具有 ADMIN OPTION 系统权限或角色的任何用户可以取消任何其他数据库用户或角色的权限或角色，取消者不必必须是最初授权权限或角色的用户，同样，具有 GRANT ANY ROLE 权限的用户可取消任何角色。

下面语句取消 TSMITH 的 CREATE TABLE 系统权限和 ACCTS\_REC 角色：

```
REVOKE CREATE TABLE, accts_rec FROM tsmith;
```

**注意** 不可有选择地取消系统权限或角色的 ADMIN OPTION，必须先取消该系统权限或角色，然后再重新授权不含 ADMIN OPTION 的系统权限或角色。

### 23.4.2 取消对象权限和角色

REVOKE 语句也可用来取消对象权限，为了取消对象权限，取消者必须是正被取消的

对象权限的原始授权者。

例如，假设你是原始授权者，为了取消 JFEE 用户和 TSMITH 用户对 EMP 表的 SELECT 和 INSERT 权限，你将执行以下语句：

```
REVOKE SELECT, insert ON emp
FROM jfee, tsmith;
```

下面语句取消 DEPT 表的所有权限（最初授权给 HUMAN\_RESOURCE 角色的权限）：

```
REVOKE ALL ON dept FROM human_resource;
```

**注意** 上面这条语句只取消授权者已授权的权限，而不是其他用户的授权。不可有选择地取消一对象权限的 GRANT OPTION，必须先取消该对象权限，然后再重新授权不带 GRANT OPTION 的对象权限，用户不可取消自己的对象权限。

#### 取消选定列的权限

尽管用户可选择表和视图的列授权 INSERT、UPDATE 和 REFERENCES 权限，当用户不能使用类似于 REVOKE 的语句有选择地取消列指定的权限，授权者必须先取消表或视图所有列的对象权限，然后再重新有选择地授权应该保留的列的指定权限。

例如，假设已将 DEPT 表的 DEPTNO 列和 DNAME 列的 UPDATE 权限授权给角色 HUMAN\_RESOURCE，为了仅仅取消角色 HUMAN\_RESOURCE 针对 DEPTNO 列的 UPDATE 权限，用户将执行下面两条语句：

```
REVOKE UPDATE ON dept FROM human_resources;
GRANT UPDATE (dname) ON dept TO human_resources;
```

REVOKE 语句取消角色 HUMAN\_RESOURCE 对 DEPT 表所有列的 UPDATE 权限，GRANT 语句重新将 DNAME 列的 UPDATE 权限授权给角色 HUMAN\_RESOURCE。

#### 取消 REFERENCES 对象权限

如果 REFERENCES 对象权限的被授权者已利用该权限创建了一外部键约束（当前已存在），那么授权者可通过在 REVOKE 语句中指明 CASCADE CONSTRAINTS 选项取消该权限：

```
REVOKE REFERENCES ON dept FROM jward CASCADE CONSTRAINTS;
```

当指定 CASCADE CONSTRAINTS 选项时，使用已取消的 REFERENCES 权限定义的外部键约束将被删除。

### 23.4.3 取消权限的效应

当一权限被取消时，根据权限的类型将会产生一连串的反应。

#### 系统权限

取消与 DDL 操作相关的系统权限时不会带来一连串的反应，不管该权限是否使用还是不使用 ADMIN OPTION 授权。例如，假设：

1. 安全管理员使用 ADMIN OPTION 将 CREATE TABLE 系统权限授权给 JFEE;
2. JFEE 创建一个表;

3. JFEE 将 CREATE TABLE 系统权限授权给 TSMITH;
4. TSMITH 创建一个表;
5. 安全管理员取消 JFEE 的 CREATE TABLE 系统权限;
6. JFEE 创建的表继续存在, TSMITH 继续保留其创建的表和 CREATE TABLE 系统权限。

取消一个与 DML 操作相关的系统权限时可观察到一连串的反应, 例如, 如果将 SELECT ANY TABLE 授权给一用户而且该用户已创建任何过程, 则包含在用户模式中的所有过程在再次使用之前必须重新授权。

### 对象权限

取消一个对象权限或许会引起一连串的反应, 所以在执行 REVOKE 语句之前应进行调研。

- 如果 DML 对象权限被取消, 那么依赖于它的对象定义将受到影响。例如, 假设 TEST 过程的过程体包含查询 EMP 表数据的 SQL 语句, 如果取消 TEST 过程所有者对 EMP 表的 SELECT 权限, 则该过程不再能成功执行。
- 如果取消 ALTER 或 INDEX 对象权限, 则要求 ALTER 和 INDEX DDL 对象权限的对象定义不会受到影响。例如, 如果取消一个用户的 INDEX 权限, 而且该用户已对某表创建了索引, 则该权限被取消后索引仍然存在。
- 当取消某一用户对某表的 REFERENCES 权限时, 任何由该用户定义的、要求被删除的 REFERENCES 权限的外部键完整性约束都将被自动删除。例如, 已授权用户 JWARD 具有对 DEPT 表的 DEPTNO 列的 REFERENCES 权限, 且该用户在 EMP 表的 DEPTNO 列上创建了一外部键, 如果取消 EMP 表的 DEPTNO 列上的 REFERENCES 权限, 则在同一操作中删除 EMP 表的 DEPTNO 列上的外部键。
- 如果取消授权者的对象权限, 那么使用 GRANT OPTION 传播的对象权限授权将被取消。例如, 假设使用 GRANT OPTION 授权给 USER1 SELECT 对象权限, 且将 EMP 的 SELECT 权限授权给 USER2, 后来将 USER1 的 SELECT 对象权限取消, 同样也取消 USER2 的 SELECT 对象权限, 则任何依赖于已取消 SELECT 权限的 USER1 和 USER2 的对象也将受到影响, 正如前面所阐述的一样。

### 23.4.4 授权和取消 PUBLIC 用户组

可将权限和角色授权给 PUBLIC 用户组, 或从 PUBLIC 用户组中取消。因为每个数据库用户都可访问 PUBLIC, 所以每个数据库用户都可取得授权给 PUBLIC 的权限和角色。

只有当每个数据库用户都需要某个权限或角色时, 安全管理员和数据库用户才会将该权限或角色授权给 PUBLIC, 请谨记每个数据库用户都应拥有成功完成用户组的当前任务所需的权限。

取消 PUBLIC 的某个权限都会引起明显的连锁反应。如果取消 PUBLIC 的与 DML 操作相关的任何权限 (例如, SELECT ANY TABLE, UPDATE ON emp), 那么数据库中的所有过程 (包括函数和程序包) 在再次使用之前都必须重新授权, 因此, 将与 DML 相关的权限授权给 PUBLIC 时应十分小心。

有关对象相关性的更多信息，参见第 19 章的“管理对象相关性”一节。

## 23.5 授予权限和取消权限何时生效

根据被授权或被取消的对象的不同，授权或取消操作将在不同时刻发生作用：

- 所有对任何对象（用户、角色和 **PUBLIC**）的系统权限和对象权限的授权/取消可立即观察到。
- 所有对任何对象（用户、其他角色和 **PUBLIC**）的角色的授权/取消只有当当前用户对话（**session**）发布 **SET ROLE** 语句给授权取消之后的角色，或授权/取消之后创建了一个新的用户对话（**session**）时才可观察到。

用户可通过检查 **SESSION\_ROLES** 数据字典视图查看哪些角色目前正被启用。

### 23.5.1 SET ROLE 语句

在对话期间，用户或应用程序可使用 **SET ROLE** 语句对任意目前对话启用角色，当然 **SET ROLE** 语句中的角色必须已经授权给用户，初始化参数 **MAX\_ENABLED\_ROLES** 限制了当前可启用的角色的个数。

下例启用角色 **CLERK**（已将该角色授权给你）并指明口令：

```
SET ROLE clerk IDENTIFIED BY bicentennial;
```

用户使用下面语句禁用所有角色：

```
SET ROLE NONE;
```

### 23.5.2 指定缺省角色

当用户登录数据库时，**Oracle** 启用所有已显式授权给该用户的权限以及该用户缺省角色中的所有权限。

使用 **ALTER USER** 语句设置和改变用户缺省角色列表。**ALTER USER** 语句允许你指定一个用户连接到数据库时即将启用的角色，而不要求用户指定角色的口令。必须使用 **GRANT** 语句直接将角色授权给该用户，用户不可将由包含目录服务器的外部服务器管理的角色（外部角色或全局角色）设置为缺省角色。

下例为 **JANE** 用户建立缺省角色：

```
ALTER USER jane DEFAULT ROLE payclerk, pettycash;
```

不可在 **CREATE USER** 语句中设置用户的缺省角色。当你第一次创建某一用户时，用户的缺省角色的设置为 **ALL**，以后授权给该用户的所有角色都是缺省角色，使用 **ALTER USER** 语句来限制用户的缺省角色。

### 23.5.3 限定一个用户可启用的角色的个数

一个用户可启用和初始化参数 **MAX\_ENABLED\_ROLE** 指定的角色的个数一样多的角



色，该计数器包含了所有间接授权的角色，数据库管理员可以改变这个限制值，该值越大，每个用户对话（session）可启用的角色就越多，但是，该参数值越大，需要越多的内存空间，这是因为对每个用户对话来说 PGA 的大小都会受到影响，并且每个角色需要 4 个字节。确定每个用户可启用的角色的最大个数之后，将该值设置为 MAX\_ENABLED\_ROLES 参数的值。

## 23.6 运用操作系统或运用网络授权

本节阐述通过用户的操作系统或网络授权角色，包括：

- 运用操作系统识别角色
- 运用操作系统管理角色
- 当 OS\_ROLES=TRUE 时授权和取消角色
- 当 OS\_ROLES=TRUE 时启用和禁用角色
- 运用网络与操作系统相联合进行角色管理

运行 Oracle 的操作系统在连接时，将角色授权给用户代替安全管理员使用 GRANT 和 REVOKE 语句显式地将数据库角色授权给用户或从用户中取消。当用户创建一个对话时，可运用操作系统管理角色并将角色传递给 Oracle。作为该机制的一部分，可对每个用户的缺省角色和使用 ADMIN OPTION 授权给该用户的角色进行身份识别。即使运用操作系统授权用户角色，但是必须在数据库中创建所有角色并使用 GRANT 语句分配给角色权限。

也可通过网络服务器授权角色，有关网络角色的信息参见“Oracle8i Distributed Database System”。

运用操作系统识别用户数据库角色的优点是使 Oracle 数据库权限管理具体化，由操作系统提供的安全措施控制管理用户的权限，该选项能集中管理许多系统活动的安全性。例如，MVS Oracle 管理员也许希望 RACF 组识别数据库用户的角色，UNIX Oracle 管理员也许希望 UNIX 组识别数据库用户的角色，或者 VMS Oracle 管理员也许希望使用权利标识符识别数据库用户的角色。

运用操作系统识别数据库用户角色的主要缺点是只能在角色级执行权限管理，不能运用操作系统授权单个权限，但仍可在数据库中使用 GRANT 语句进行授权。

第二个缺点是如果操作系统管理这些角色，那么该角色的缺省用户不可通过多线程服务器连接到数据库或连接到其它网络，可是，用户可改变这个缺省值，参见 23.6.5 节“运用网络与操作系统联合进行角色管理”。

参见 本节阐述的特征只对某些操作系统有用，由操作系统来决定这些特征是否可用，参见指定操作系统的 Oracle 文档。

### 23.6.1 运用操作系统识别角色

将初始化参数 OS\_ROLES 设置为 TRUE，运行数据库（如果数据库正在运行，重新启动它），当创建了一个对话时可运用操作系统来识别用户的数据库角色。当用户试图创建一个与数据库相连的对话时，Oracle 使用由操作系统识别的数据库角色将用户的安全域初始化。

为了识别用户的数据库角色，每个 Oracle 用户的操作系统帐号持有操作系统标识符（或许称作组、权利标识符或其它类似的名称），该标识符指出哪些数据库角色对该用户有用。角色的详细设计亦可指出哪些角色是该用户的缺省角色，哪些带有 ADMIN OPTION 选项的角色是可用的，无论运用哪种操作系统，按以下格式设计操作系统级的角色：

```
ora_<ID>_<ROLE>[_[d][a]]
```

其中：

**ID**

ID 的定义根据操作系统的不同而不同，例如，在 VMS 操作系统中，ID 是数据库的实例的标识符；在 MVS 操作系统中，ID 是机器的类型；在 UNIX 操作系统中，ID 是系统 ID。

**注意** ID 对用户的 ORACLE\_SID 敏感，但 ROLE 就不会出现这种情况。

**D**

该选项指出这个角色是数据库用户的缺省角色。

**A**

该选项指出这个角色将被授权给具有 ADMIN OPTION 选项的用户，这样允许用户将该角色授权给其他角色（如果运用操作系统管理角色，则不可将角色授权给用户）。

**注意** 如果需要指定 D 或指定 A，在 A 或 D 之前必须先写上下划线“-”。

例如，一个操作系统帐号或许拥有在其 PROFILE 文件中识别的下列对象：

```
ora_PAYROLL_ROLE1
ora_PAYROLL_ROLE2_a
ora_PAYROLL_ROLE3_d
ora_PAYROLL_ROLE4_da
```

当相应的用户连接到 Oracle 的 PAYROLL 实例时，ROLE3 和 ROLE4 是缺省角色，同时 ROLE2 和 ROLE4 对拥有 ADMIN OPTION 选项的用户有效。

### 23.6.2 运用操作系统进行角色管理

当用户运用操作系统管理角色时，值得注意的是数据库角色正被授权给操作系统用户。任何操作系统用户可连接的数据库用户都可启用已授权的数据库角色。基于这个原因，如果 OS\_ROLE=TRUE，用户应考虑将所有 Oracle 用户定义为 IDENTIFIED EXTERNALLY，这样数据库帐号就可以连接到已授予权限的操作系统帐号。

### 23.6.3 当 OS\_ROLES=TRUE 时授权和管理角色

如果 OS\_ROLES 设置为 TRUE，则操作系统完全管理该角色对用户的授权和取消，任何以前使用 GRANT 语句对用户的角色授权都不再使用，但它们仍保留在数据字典中，只有操作系统级的角色授权仍可用，用户仍可以将权限授权给角色和用户。

**注意** 如果操作系统使用 ADMIN OPTION 选项将一角色授权给一用户，则该用户只能将该角色再授权给其他角色。

### 23.6.4 当 OS\_ROLES=TRUE 时启用和禁用角色

如果 OS\_ROLES 设置为 TRUE，可使用 SET ROLE 语句动态地启用任何由操作系统授权的角色。如果角色定义中要求口令或操作系统授权，则口令仍有效，但是，不可在 SET ROLE

语句中指定未经用户操作系统帐号识别的任何角色，即使当 `OS_ROLES=FALSE` 时已使用 `GRANT` 语句授权该角色（如果你指定这样一个角色，Oracle 将忽略它。）

当 `OS_ROLES=TRUE` 时，用户可以启用和 `MAX_ENABLED_ROLES` 参数指定的个数一样多的角色。

### 23.6.5 运用网络与操作系统联合管理角色

如果你希望让操作系统管理角色，作为缺省用户不能通过多线程服务器连接到数据库。由于一个远程用户可通过非安全连接扮演另一个操作系统用户，所以这个限制是缺省的。

如果你不关心安全问题并希望使用多线程服务器或其它网络连接实现操作系统角色管理，那么将数据库参数文件中的 `REMOTE_OS_ROLESC` 参数设置为 `TRUE`，在用户下次启动该实例和登录数据库之前该变化一直生效（该参数的缺省值是 `FALSE`）。

## 23.7 权限和角色信息列表

为列举对象的权限，用户可查询下列数据字典视图：

- `ALL_COL_PRIVS`, `USER_COL_PRIVS`, `DBA_COL_PRIVS`
- `ALL_COL_PRIVS_MADE`, `USER_COL_PRIVS_MADE`
- `ALL_COL_PRIVS_RECD`, `USER_COL_PRIVS_RECD`
- `ALL_TAB_PRIVS`, `USER_TAB_PRIVS`, `DBA_TAB_PRIVS`
- `ALL_TAB_PRIVS_MADE`, `USER_TAB_PRIVS_MADE`
- `ALL_TAB_PRIVS_RECD`, `USER_TAB_PRIVS_RECD`
- `DBA_ROLES`
- `USER_ROLE_PRIVS`, `DBA_ROLE_PRIVS`
- `USER_SYS_PRIVS`, `DBA_SYS_PRIVS`
- `COLUMN_PRIVILEGES`
- `ROLE_ROLE_PRIVS`, `ROLE_SYS_PRIVS`, `ROLE_TAB_PRIVS`
- `SESSION_PRIVS`, `SESSION_ROLES`

下面是使用这些视图的一些例子，假设所有这些例子发布以下语句：

```
CREATE ROLE security_admin IDENTIFIED BY honcho;
```

```
GRANT CREATE PROFILE, ALTER PROFILE, DROP PROFILE,  
      CREATE ROLE, DROP ANY ROLE, GRANT ANY ROLE, AUDIT ANY,  
      AUDIT SYSTEM, CREATE USER, become user, ALTER USER, DROP USER  
      TO security_admin WITH ADMIN OPTION;
```

```
GRANT SELECT, DELETE ON sys.aud$ TO security_admin;
```

```
GRANT security_admin, CREATE SESSION TO swilliams;
```

```
GRANT security_admin TO system_administrator;
```

GRANT CREATE SESSION TO jward;

GRANT SELECT, DELETE ON emp TO jward;

GRANT INSERT (ename, job) ON emp TO swilliams, jward;

参见 有关这些数据字典视图的详细描述参见 “Oracle8i Reference”。

### 23.7.1 列举所有可授权的系统权限

下面查询语句指出了所有可授权给用户和角色的系统权限:

SELECT \* FROM sys.dba\_sys\_privs;

| GRANTEE        | PRIVILEGE      | ADM   |
|----------------|----------------|-------|
| -----          | -----          | ----- |
| SECURITY_ADMIN | ALTER PROFILE  | YES   |
| SECURITY_ADMIN | ALTER USER     | YES   |
| SECURITY_ADMIN | AUDIT ANY      | YES   |
| SECURITY_ADMIN | AUDIT SYSTEM   | YES   |
| SECURITY_ADMIN | BECOME USER    | YES   |
| SECURITY_ADMIN | CREATE PROFILE | YES   |
| SECURITY_ADMIN | CREATE ROLE    | YES   |
| SECURITY_ADMIN | CREATE USER    | YES   |
| SECURITY_ADMIN | DROP ANY ROLE  | YES   |
| SECURITY_ADMIN | DROP PROFILE   | YES   |
| SECURITY_ADMIN | DROP USER      | YES   |
| SECURITY_ADMIN | GRANT ANY ROLE | YES   |
| SWILLIAMS      | CREATE SESSION | NO    |
| JWARD          | CREATE SESSION | NO    |

### 23.7.2 列举所有可授权的角色

下面查询语句返回所有可授权给用户和其他角色的角色:

SELECT \* FROM sys.dba\_role\_privs;

| GRANTEE   | GRANTED_ROLE   | ADM   |
|-----------|----------------|-------|
| -----     | -----          | ----- |
| SWILLIAMS | SECURITY_ADMIN | NO    |

### 23.7.3 列举所有可授权给用户的对象权限

下面查询语句返回所有授权给指定用户的对象权限（不包括指定列的权限）:

SELECT table\_name, privilege, grantable FROM sys.dba\_tab\_privs  
WHERE grantee = 'JWARD';

| TABLE_NAME | PRIVILEGE | GRANTABLE |
|------------|-----------|-----------|
| -----      | -----     | -----     |
| EMP        | SELECT    | NO        |
| EMP        | DELETE    | NO        |

使用下面语句列举授权给指定列的权限：

```
SELECT grantee, table_name, column_name, privilege
FROM sys.dba_col_privs;
```

| GRANTEE   | TABLE_NAME | COLUMN_NAME | PRIVILEGE |
|-----------|------------|-------------|-----------|
| -----     | -----      | -----       | -----     |
| SWILLIAMS | EMP        | ENAME       | INSERT    |
| SWILLIAMS | EMP        | JOB         | INSERT    |
| JWARD     | EMP        | NAME        | INSERT    |
| JWARD     | EMP        | JOB         | INSERT    |

### 23. 7. 4 列举当前对话的权限域

下面查询语句列举了发行者（issuer）当前启用的角色：

```
SELECT * FROM session_roles;
```

如果 SWILLIAMS 已启用 SECURITY\_ADMIN 角色并执行该条语句，则 Oracle 将返回下面信息：

```
ROLE
-----
SECURITY_ADMIN
```

下面查询语句列举了发行者安全域中当前所有可用系统权限，包括来自显式授权的权限和来自被启用的角色授权的权限：

```
SELECT * FROM session_privs;
```

如果 SWILLIAMS 启用了 SECURITY\_ADMIN 角色并执行上述查询语句，则 Oracle 将返回下列结果：

```
PRIVILEGE
-----
AUDIT SYSTEM
CREATE SESSION
CREATE USER
BECOME USER
ALTER USER
DROP USER
CREATE ROLE
DROP ANY ROLE
GRANT ANY ROLE
AUDIT ANY
CREATE PROFILE
ALTER PROFILE
DROP PROFILE
```

如果对 SWILLIAMS 禁用 SECURITY\_ADMIN 角色，则第一条查询语句不返回任何行，第二条查询语句只返回一行 CREATE SESSION。

### 23.7.5 列举数据库角色

可利用 `DBA_ROLES` 视图列举数据库的所有角色和对每个角色的身份验证，例如，下面查询语句列举了数据库中的所有角色：

```
SELECT * FROM sys.dba_roles;
```

| ROLE           | PASSWORD |
|----------------|----------|
| CONNECT        | NO       |
| RESOURCE       | NO       |
| DBA            | NO       |
| SECURITY_ADMIN | YES      |

### 23.7.6 列举角色的权限域的相关信息

数据字典视图 `ROLE_ROLE_PRIVS`、`ROLE_SYS_PRIVS` 和 `ROLE_TAB_PRIVS` 包含了角色的权限域的相关信息。

例如，下面查询语句列举了所有授权给 `SYSTEM_ADMIN` 角色的角色：

```
SELECT granted_role, admin_option
FROM role_role_privs
WHERE role='SYSTEM_ADMIN';
```

| GRANTED_ROLE   | ADM |
|----------------|-----|
| SECURITY_ADMIN | NO  |

下面查询语句列举了所有授权给 `SECURITY_ADMIN` 角色的系统权限：

```
SELECT * FROM role_sys_privs WHERE role='SECURITY_ADMIN';
```

| ROLE           | PRIVILEGE      | ADM |
|----------------|----------------|-----|
| SECURITY_ADMIN | ALTER PROFILE  | YES |
| SECURITY_ADMIN | ALTER USER     | YES |
| SECURITY_ADMIN | AUDIT ALL      | YES |
| SECURITY_ADMIN | AUDIT SYSTEM   | YES |
| SECURITY_ADMIN | BECOME USER    | YES |
| SECURITY_ADMIN | CREATE PROFILE | YES |
| SECURITY_ADMIN | CREATE ROLE    | YES |
| SECURITY_ADMIN | CREATE USER    | YES |
| SECURITY_ADMIN | DROP ANY ROLE  | YES |
| SECURITY_ADMIN | DROP PROFILE   | YES |
| SECURITY_ADMIN | DROP USER      | YES |
| SECURITY_ADMIN | GRANT ANY ROLE | YES |

下面查询语句列举了授权给 `SECURITY_ADMIN` 角色的所有对象权限：

```
SELECT table_name, privilege FROM role_tab_privs
WHERE role = 'SECURITY_ADMIN';
```

| TABLE_NAME | PRIVILEGE |
|------------|-----------|
| -----      | -----     |
| AUD\$      | DELETE    |
| AUD\$      | SELECT    |

# 24 数据库使用审计

本章阐述如何使用数据库审计工具，包括以下内容：

- 审计准则
- 创建和删除数据库审计踪迹视图
- 审计踪迹信息管理
- 查看数据库审计踪迹信息
- 通过数据库触发器进行审计

## 24.1 审计准则

本节阐述了有关审计的准则，包括：

- 通过数据库或操作系统进行审计
- 使得审计信息易管理

### 24.1.1 通过数据库或操作系统进行审计

每个数据库的数据字典都有一与数据库“审计踪迹”有关的表 `SYS.AUD$`，该表用来存储审计数据库语句、权限或模式对象的记录。

用户的操作系统也可以保留一审计踪迹，该审计踪迹存储由操作系统审计工具生成的审计记录。这种指定操作系统的审计工具可能支持从数据库审计到操作系统审计踪迹，也可能不支持。如果该选项可用，则考虑数据库或操作系统审计踪迹、存储数据库审计记录的优缺点。

使用数据库审计踪迹具有以下优点：

- 用户可借助数据字典的预定义的审计踪迹视图有选择地查看审计踪迹。
- 用户可使用 Oracle 工具（比如 Oracle Reports）生成审计记录。

另外，用户的操作系统审计踪迹允许用户合并来自多源点的审计记录，包括 Oracle 和其它应用程序，因此，审查系统活动可能会更有效，因为所有审计记录都在一个地方。

参见 查阅指定操作系统的文档获取其审计能力的信息。

### 24.1.2 使得审计信息易管理

尽管审计相对廉价，但也要尽可能地限制审计的次数，这将降低对被审计语句的性能的影响，减少审计踪迹的大小。

设计审计策略时使用下列一般性准则：

- 评估审计目的

用户明确审计理由之后可设计正确的审计策略，避免不必要的审计。

例如，假设用户准备审计调查可疑的数据库活动，它本身提供的信息并不充分，用户



所怀疑的或所注意的数据库活动是什么类型的活动？更常见的审计目的或许是审计对数据库表未经授权的删除，这样缩小了被审计活动的类型和受可疑活动影响的对象的类型。

#### ■ 明智地审计

审计最少量的语句、用户或对象，获得目标信息，防止不必要的审计信息扰乱有用信息和消耗 **SYSTEM** 表空间中宝贵的空间，均衡一下你对采集足够多安全信息的需求和你存储及处理它们的能力。

例如，如果你通过采集有关数据库行为的信息进行审计，那么准确地确定所跟踪的活动类型，只审计感兴趣的活动，只进行必要次数的审计以采集你所需要的信息。如果你只对每个会话（session）的逻辑 I/O 信息感兴趣，则不必审计对象。

#### 审计可疑的数据库活动

当用户审计监视可疑的数据库活动时，遵循下列准则：

##### ■ 先做一般性审计，再做明确审计。

开始审计可疑的数据库活动时，通常没有太多的信息可供目标指定用户或模式对象使用，因此，刚开始只是做一般性的审计，一旦记录和分析了初步的审计信息，则结束一般性审计，转而进行明确的审计。这个过程不断执行直到采集到足够的证据，能对可疑数据库活动的起源做出明确的结论。

##### ■ 保护审计踪迹。

当对可疑数据库活动进行审计时，保护审计踪迹，这样不审计时不会增加、改变或删除审计信息。有关更多信息和指导，参见 24.3.5 节“保护审计踪迹”。

#### 审计正常数据库活动

当用户审计正常的数据库活动、采集指定数据库活动的历史信息时，遵循下列准则：

##### ■ 只审计相关的活动。

为避免将有用的信息和无用的审计记录弄混、减少审计踪迹管理的数量，只审计目标数据库活动。

##### ■ 存档审计记录和整理审计踪迹。

用户收集到所需的信息之后，将感兴趣的审计记录存档，整理有关审计踪迹信息。

## 24.2 创建和删除数据库审计踪迹视图

数据库审计踪迹（**SYS.AUD\$**）是每个 Oracle 数据库的数据字典中的单独的一个表。为帮助用户查看该表中的有意义的审计信息，提供了几个预定义视图，这些视图必须是为用户使用审计而创建的，如果以后你决定不使用审计，可将它们删除。

当你运行 **CATALOG.SQL** 脚本时，审计踪迹视图自动被创建，**USER** 视图由 **CATALOG.SQL** 脚本创建。

### 24.2.1 创建审计踪迹视图

下列视图由 **CATALOG.SQL** 脚本和 **CATAUDIT.SQL** 脚本创建。

| 视图                    | 描述  |
|-----------------------|---|
| STMT_AUDIT_OPTION_MAP | 包含关于审计选项编码的信息   |
| AUDIT_ACTIONS         | 包含对审计跟踪活动类型编码的描述  |
| ALL_DEF_AUDIT_OPTS    | 包含对象创建时将应用的缺省对象审计选项   |
| DBA_STMT_AUDIT_OPTS   | 描述目前系统的系统级和用户的审计选项  |
| DBA_PRIV_AUDIT_OPTS   | 描述目前正被审计的的系统级和用户的系统权限   |
| DBA_OBJ_AUDIT_OPTS,   | 描述所有对象上的审计选项， <b>USER</b> 视图描述由目前用户   |
| USER_OBJ_AUDIT_OPTS   | 拥有的对象上的审计选项   |
| DBA_AUDIT_TRAIL,      | 列举所有审计踪迹表目， <b>USER</b> 视图显示与当前用户相  |
| USER_AUDIT_TRAIL      | 关的审计踪迹表目  |
| DBA_AUDIT_OBJECT,     | 包含对系统中所有对象的审计踪迹记录， <b>USER</b> 视图列  |
| USER_AUDIT_OBJECT     | 举了对有关当前用户可访问的对象语句的审计跟踪记录  |
| DBA_AUDIT_SESSION,    | 列举所有关于 <b>CONNECT</b> 和 <b>DISCONNECT</b> 的审计踪迹                                 |
| USER_AUDIT_SESSION    | 记录， <b>USER</b> 视图列举所有关于当前用户连接和非连接的   |
|                       | 审计踪迹记录  |
| DBA_AUDIT_STATEMENT,  | 列举关于由用户发布的、贯穿整个数据库或只针对 <b>USER</b>  |
| USER_AUDIT_STATEMENT  | 视图的 <b>GRANT</b> 、 <b>REVOKE</b> 、 <b>AUDIT</b> 、 <b>NOAUDIT</b> 和 <b>ALTER</b> |
|                       | 语句的审计跟踪记录   |
| DBA_AUDIT_EXISTS      | 列举由 <b>AUDIT EXISTS</b> 和 <b>AUDIT NOT EXISTS</b> 产生的                           |
|                       | 审计踪迹表目（entry）   |

关于对这些例子的审计信息的说明，参见 24.4 节“查看数据库审计踪迹信息”。

参见 有关这些视图的信息，参见“Oracle8i Reference”。

### 24.2.2 删除审计踪迹视图

如果你禁止审计，而且不再需要审计踪迹视图，那么以 **SYS** 身份连接到数据库，运行脚本文件 **CATNOAUD.SQL** 就可以删除所有审计踪迹视图，**CATNOAUD.SQL** 的名称和定位由操作系统确定。

## 24.3 管理审计踪迹信息

审计踪迹记录可以包含不同类型的信息，根据被审计的事件和审计选项设置而确定。倘若下列信息对特别的审计活动是有意义的，则它们总是包含在每个审计踪迹记录中。

- 用户名
- 会话标识符（session identifier）
- 终端标识符
- 可访问的模式对象的名称
- 被执行的或被试图执行的操作
- 操作的完成码

- 日期和时间标记
- 所使用的系统权限

写入操作系统审计踪迹的审计踪迹记录被译成密码，并不可读，但可在数据字典文件和错误信息中将它们解密，如下所示：

- |         |   |
|---------|---|
| 操作代码    | 该代码描述被执行或被试图执行的操作， <b>AUDIT_ACTIONS</b> 数据字典表包含了一系列这样的代码以及对它们的描述。                       |
| 计算机使用特权 | 所使用的权限，该代码描述用来执行操作的系统权限， <b>SYSTEM_PRIVILEGE_MAP</b> 表列举所有这样的代码以及对它们的描述。                |
| 完成码     | 该代码描述试图执行操作的结果，成功的操作返回 0；失败的操作返回描述为什么失败的 Oracle 错误代码，这些代码在“Oracle8i Error Messages”中列出。 |

本节阐述了管理审计踪迹信息的不同方面，包括：

- 缺省审计事件
- 设置审计选项
- 启动和禁止数据库审计
- 控制审计踪迹的增长和大小
- 保护审计踪迹

#### 24.3.1 缺省审计事件

不管是否启用数据库审计，Oracle 总是审计某些与数据库相关的操作，并写入操作系统审计踪迹，这些事件是：

##### ■ 实例启动

产生一个列举启动该实例的操作系统用户、用户的终端标识符、日期和时间标志以及是否启用或禁止数据库审计的审计记录，该记录存储在操作系统审计踪迹中，因为启动完全成功完成后，数据库审计踪迹才可用。启动时记录数据库审计的状态也可以防止管理员在数据库审计禁止的情况下重新启动数据库（所以他们可以执行不审计的操作）。

##### ■ 实例停止运行

产生一个列举停止该实例的操作系统用户、用户终端标识符、日期和时间标志的审计记录。

##### ■ 以管理员身份连接到数据库

产生一个列举以 **SYSOPER** 或 **SYSDBA** 身份连接到数据库的操作系统用户的审计记录，提供具有管理员权限的用户的职责。

对于那些不能让 Oracle 访问审计踪迹的操作系统，这些审计跟踪记录和后台处理跟踪文件一起被存放在同一目录下的 Oracle 审计踪迹文件中。

参见 使用 **AUDIT** 语句指定需审计的语句或模式对象。有关 **AUDIT** 语句的其他信息，包括语法和所要求的授权级别，参见“Oracle8i SQL Reference”。

#### 24.3.2 设置审计选项

针对不同的审计选项设置，审计记录将包含不同类型的信息，但是，所有审计选项都将

产生以下信息：

- 执行审计语句的用户
- 操作编码，指出用户所执行的审计语句
- 审计语句中涉及到的对象
- 审计语句执行的日期和时间

审计踪迹不存储审计语句中任何可能涉及的数据值信息，例如，审计 UPDATE 语句时，不存储所更新行的新、旧数据值，但可以在 DML 语句中包含有关表，通过使用数据库触发器执行特殊类型的审计，使用触发器进行特殊审计的例子见 24.5 节“通过数据库触发器进行审计”。

Oracle 允许用户设置以下三级审计选项：

- |     |  |
|-----|--|
| 语句级 | 对所使用 SQL 语句类型的审计，比如一个表上的任何 SQL 语句（CREATE、TRUNCATE 和 DROP TABLE 语句） |
| 权限级 | 审计特殊系统权限的使用，比如 CREATE TABLE  |
| 对象级 | 审计指定对象的指定语句，比如 EMP 表上的 ALTER TABLE                                 |

#### 语句审计选项

包含在 AUDIT 和 NOAUDIT 语句中的有效的语句审计选项都在“Oracle8i SQL Reference”中列出。

#### 审计连接和断开

SESSION 语句选项是特殊选项，因为发出特殊类型语句时它不生成审计记录，该选项为每一个通过连接到一个实例而创建的会话分别生成一个审计记录，连接时将审计记录插入到审计踪迹中，断开时更新审计记录，有关某个会话的累积信息，比如连接时间、断开时间、逻辑和物理 I/Os 处理等等都存储在与该会话相连通的审计记录中。

#### 权限审计选项

权限审计选项与相应的系统选项完全匹配，例如，审计 DELETE ANY TABLE 权限的选项是 DELETE ANY TABLE。为开启该选项，用户应使用类似于下例的语句：

```
AUDIT DELETE ANY TABLE
  BY ACCESS
  WHENEVER NOT SUCCESSFUL;
```

“Oracle8i SQL Reference”中列举了所有 Oracle 的系统权限。

#### 对象审计选项

“Oracle8i SQL Reference”列举了所有可用的对象审计选项和针对每个选项可用的模式对象类型。

#### 启用审计选项

AUDIT 语句启用语句、权限和对象审计选项。使用它设置语句和权限选项，用户必须拥有 AUDIT SYSTEM 的权限；使用它设置对象审计选项，用户必须拥有被审计的对象或拥有 AUDIT ANY 的权限。设置语句和权限审计选项的审计语句可以包含一个 BY 子句，用来

指定一系列用户或应用程序，限制语句和权限审计选项的范围。

用户可设置任何审计选项，并指定下列审计条件：

■ **WHENEVER SUCCESSFUL/WHENEVER NOT SUCCESSFUL**

■ **BY SESSION/BY ACCESS**

创建一个新的会话时，它从数据字典中获得审计选项，大批审计选项在数据库连接期间继续保留，设置新的系统或对象审计选项使得随后的数据库会话使用这些选项，现存会话将继续使用会话创建时的审计选项。

**注意** AUDIT 语句只指定审计选项，总体上来说它不能启用审计，为了启动审计和控制 Oracle 是否产生基于当前审计选项的审计记录，设置数据库参数文件中的 AUDIT\_TRAIL 参数。

有关启动和禁止审计的更多的信息，参见 23.3.3 节“启用和禁用数据库审计”。

**参见** 有关 AUDIT 语句的完整描述，参见“Oracle8i SQL Reference”。

**启用语句权限审计** 为了审计所有成功的或不成功的对数据库的连接和断开，不考虑用户，BY SESSION（该选项的缺省值和唯一值），输入下面语句：

```
AUDIT SESSION;
```

你也可为用户选择性地设置选项，如下例所示：

```
AUDIT SESSION
```

```
BY scott, lori;
```

为了审计 DELETE ANY TABLE 系统权限所有成功的和不成功的使用，输入下面语句：

```
AUDIT DELETE ANY TABLE;
```

为了审计所有表上不成功的 SELECT、INSERT、DELETE 语句和 EXECUTE PROCEDURE 系统权限不成功的使用，输入下列语句：

```
AUDIT SELECT TABLE, INSERT TABLE, DELETE TABLE,  
EXECUTE PROCEDURE  
BY ACCESS  
WHENEVER NOT SUCCESSFUL;
```

设置任何语句或权限审计选项时要求具有 AUDIT SYSTEM 系统权限，通常，安全管理员是唯一能授权该系统权限的用户。

**启用对象审计** 为了审计 SCOTT.EMP 表上的所有成功和不成功的 DELETE 语句，BY SESSION（缺省值），输入下面语句：

```
AUDIT DELETE ON scott.emp;
```

为了审计 JWARD 用户拥有的 DEPT 表上的所有成功的 SELECT、INSERT 和 DELETE 语句，BY ACCESS，输入下面语句：

```
AUDIT SELECT, INSERT, DELETE  
ON jward.dept  
BY ACCESS  
WHENEVER SUCCESSFUL;
```

为设置缺省对象选项来审计所有不成功的 SELECT 语句，BY SESSION（缺省值），输

入下面语句：

```
AUDIT SELECT
ON DEFAULT
WHENEVER NOT SUCCESSFUL;
```

用户可为包含在用户模式中对象设置任何对象选项，给包含在其他用户模式中的对象设置对象审计选项或设置缺省对象审计选项时要求用户有 **AUDIT ANY** 的系统权限，通常，安全管理员是唯一能授权这种权限的用户。

### 多层环境中的审计

在多层环境中，Oracle 保留穿过所有层的客户的标识符，这将启用为该客户采用的审计活动。为此，你应在自己的 **AUDIT** 语句中使用 **BY PROXY** 子句。

该子句允许给予用户一些选择，用户可以：

- 审计由指定代理为它自己发布的 **SQL** 语句。
- 审计为指定用户执行的语句。
- 审计所有任何用户执行的语句。

下例审计通过代理应用程序服务器 **APPSERVE** 为客户 **JACKSON** 发布的 **SELECT TABLE** 语句：

```
AUDIT SELECT TABLE
BY appserve ON BEHALF OF jackson;
```

参见 参见 “Oracle8i Concepts” 和 “Oracle8i Application Developer’s Guide – Fundamentals” 获取关于代理和多层应用程序的更多的信息。

### 禁止审计选项

**NOAUDIT** 语句关掉 Oracle 的不同的审计选项，可使用它重新设置语句和权限审计选项、对象审计选项。设置语句和权限审计选项的 **NOAUDIT** 语句可包含 **BY USER** 选项来指定一系列用户，限制语句和权限审计选项的范围。

用户可使用 **NOAUDIT** 语句和 **WHENEVER** 子句有选择地禁止某一审计选项，如果不指定该子句，所有审计选项都被禁止，包括成功和不成功两种情况。

**NOAUDIT** 不支持 **BY SESSION/BY ACCESS** 选项对，通过适当的 **NOAUDIT** 语句关闭审计选项，不管它们是否已被启用。

**警告** **NOAUDIT** 只指定审计选项，它不禁止整个审计。为了关闭审计和从生成的审计记录中停止 Oracle，即使你已有当前审计选项，必须设置数据库参数文件中的 **AUDIT\_TRAIL** 参数。

参见 有关 **NOAUDIT** 语句的完整语法，参见 “Oracle8i SQL Reference”。

### 禁止语句和权限审计

下列语句关闭相应的审计选项：

```
NOAUDIT session;
NOAUDIT session BY scott, lori;
NOAUDIT DELETE ANY TABLE;
```

```
NOAUDIT SELECT TABLE, INSERT TABLE, DELETE TABLE,  
EXECUTE PROCEDURE;
```

下列语句关闭语句（系统）和权限审计选项：

```
NOAUDIT ALL;  
NOAUDIT ALL PRIVILEGES;
```

为禁止语句或权限审计选项，用户必须有 **AUDIT SYSTEM** 系统权限。

**禁止对象审计** 下列语句关闭相应的审计选项：

```
NOAUDIT DELETE  
ON emp;  
NOAUDIT SELECT, INSERT, DELETE  
ON jward,dept;
```

此外，为了关闭 **EMP** 表上的所有对象审计选项，输入下面语句：

```
NOAUDIT ALL  
ON emp;
```

**禁止缺省对象审计选项** 为关闭所有缺省对象审计选项，输入下面语句：

```
NOAUDIT ALL  
ON DEFAULT;
```

注意执行 **NOAUDIT** 语句之前创建的模式对象继续使用对象创建时生效的缺省对象审计选项，除非对象创建之后由一条显式的 **NOAUDIT** 语句重写。

为禁止一指定对象的对象审计权限，用户必须是该模式对象的所有者。为禁止其他用户模式中的某一对象的对象审计权限或禁止缺省对象审计选项，用户必须拥有 **AUDIT ANY** 系统权限。具有禁止某一对象的对象审计选项权限的用户可以无视其他用户设置的选项。

### 24.3.3 启用和禁用数据库审计

任何已授权的数据库用户在任何时候都可设置语句、权限和对象审计选项，但 **Oracle** 并不生成和存储审计踪迹中的审计记录，除非启动数据库审计，通常由安全管理员负责这个操作。

通过数据库参数文件中的 **AUDIT\_TRAIL** 初始化参数启动和禁止数据库审计，该参数可设置为以下值：

|             |                            |
|-------------|----------------------------|
| <b>DB</b>   | 启动数据库审计并将所有审计记录写到数据库审计踪迹中  |
| <b>OS</b>   | 启动数据库审计并将所有审计记录写到操作系统审计踪迹中 |
| <b>NONE</b> | 禁止审计（该值是缺省值）               |

可利用 **AUDIT\_FILE\_DEST** 参数指定存储审计文件的目录

如果你编辑了该参数，重新启动数据库实例，按计划启动或禁止数据库审计，这些参数不是动态的。

参见 有关这些参数和如何编辑参数文件的更多信息，参见 “Oracle8i Reference”。

### 24.3.4 控制审计踪迹的增长和大小

如果审计踪迹完全充满了，再也插不进去任何审计记录，则无法成功执行审计语句，直

到重新清理审计踪迹，发出审计语句时会给所有用户返回警告信息，因此，安全管理员必须控制审计踪迹的增长和大小。

当启动审计并生成审计记录时，审计踪迹按以下两个因素增长：

- 启动的审计选项的个数

- 执行审计语句的频度

用户可采用以下方法来控制审计踪迹的增长：

- 启动和禁止数据库审计。如果启动数据库审计，则将产生审计记录，并将审计记录存储在审计踪迹中；如果禁止数据库审计，则不产生审计记录。

- 精心挑选将开启的审计选项。如果只执行精心挑选的审计，则不会产生一些无用和没必要的信息，也不会存储在审计踪迹中。

- 紧紧控制执行对象审计的能力，这可通过两种不同的方式实现：

1. 安全管理员拥有所有对象，AUDIT ANY 系统权限从来没有授权给其他用户。或者，所有模式对象属于一个模式，该模式相应的用户不拥有 CREATE SESSION 权限。

2. 所有对象都包含在与实际数据库用户不相符的模式中（也就是说，CREATE SESSION 权限没有授权给相应的用户），安全管理员是授权 AUDIT ANY 系统权限的唯一用户。

在这两种情况下，对象审计完全由安全管理员控制。

数据库审计踪迹（SYS.AUD\$表）的最大长度在数据库创建时就预定好了，该表的缺省大小可达到 99 个区域（extent），每个区域的大小为 10K。作为控制审计踪迹增长和大小的工具，用户不可将 SYS.AUD\$移动到其它表空间，用户可修改 SYS.AUD\$中的缺省存储参数。

**注意** 不鼓励用户将 SYS.AUD\$移出 SYSTEM 表空间，因为 Oracle 代码对数据字典表做了隐含的假设，比如对 SYS.AUD\$，随着升级和备份/恢复该表可能会引起一些问题。

**参见** 如果用户希望将审计记录写到操作系统审计踪迹中，参见用户特定操作系统的 Oracle 文档获取如何管理操作系统审计踪迹的更多的信息。

### 清理审计踪迹中的审计记录

启动审计一段时间之后，安全管理员可能希望删除数据库审计踪迹中部分记录，释放审计踪迹空间，方便审计踪迹管理。

例如，输入下面语句删除审计踪迹中的所有记录：

```
DELETE FROM sys.aud$
```

或者，输入下面语句删除审计 EMP 表产生的审计踪迹中的所有审计记录：

```
DELETE from sys.aud$  
WHERE obj$name='EMP';
```

如果审计踪迹信息必须作为历史存档，那么安全管理员可将有关记录拷贝到一普通数据库表中（例如，使用“INSERT INTO table SELECT...FROM sys.auds ...”）或将审计踪迹表输出到一操作系统文件中。

只有 SYS 用户（具有 DELETE ANY TABLE 权限）或 SYS 已授权其对 SYS.AUD\$表具有 DELETE 权限的用户可以删除数据库审计踪迹中的记录。



**注意** 如果审计踪迹已满，并且审计连接仍在继续（也就是说，如果设置了 `SESSION` 选项），则典型用户不能连接到数据库，因为与连接相关联的审计记录不能插入到审计踪迹中。这种情况下，安全管理员必须以 `SYS` 身份连接（`SYS` 的操作不审计），并给审计踪迹提供有效空间。

**参见** 有关表输出的信息，参见“Oracle8i Utilities”。

### 减少审计踪迹的大小

正如任何数据库表，记录从数据库审计踪迹中删除之后，分配给该表区域继续存在。

如果为数据库审计踪迹分配了许多区域，可其中有很多区域不再使用，那么按照以下步骤减少分配给数据库审计踪迹的空间：

1. 如果你希望保存当前数据库审计踪迹中的信息，那么将它拷贝到另一个数据库表中或使用 `EXPORT` 工具输出。

2. 如同管理员一样连接。

3. 使用 `TRUNCATE` 语句截取 `SYS.AUD$`。

4. 再装载步骤产生的、已存档的审计踪迹记录。

只分配包含当前审计踪迹记录所必需的盘区给新版本的 `SYS.AUD$`。

**注意** `SYS.AUD$`是唯一一个不可直接修改的 `SYS` 对象。

### 24.3.5 保护审计踪迹

审计可疑数据库操作时，保护审计踪迹记录的完整性，确保审计信息准确和完全。

为防止数据库审计踪迹未经授权被删除，只将 `DELETE ANY TABLE` 系统权限授权给安全管理员。

使用下面语句审计对数据库审计踪迹的改变：

```
AUDIT INSERT, UPDATE, DELETE
ON sys.aud$
BY ACCESS;
```

某个具有管理员权限的用户可删除审计踪迹中的由 `SYS.AUD$` 表的对象审计选项而产生的审计记录。作为最后一个保护审计踪迹措施，如果可用的话，在操作系统审计踪迹中审计以管理员权限连接时所执行的任何操作。

**参见** 有关操作系统审计踪迹的可用性和所有可能的用途，参见用户操作系统指定 Oracle 文档。

## 24.4 查看数据库审计踪迹信息

本节提供了一些例子，证明如何分析和翻译审计踪迹中的信息，考虑以下状况。

用户希望审计下列可疑数据库操作：

- 未经授权审计口令、表空间设置和数据库用户的配额。
- 大量死锁出现，主要是因为用户拥有专用的表锁。
- 任意地删除 `SCOTT` 模式中的 `EMP` 表中的行。

你怀疑用户 JWARD 和 SWILLIAMS 可能有以上几种有害的行为。  
按顺序输入下列语句进行调查：

```
AUDIT ALTER, INDEX, RENAME ON DEFAULT
    BY SESSION;
CREATE VIEW scott.employee AS SELECT * FROM scott.emp;
AUDIT SESSION BY jward, swilliams;
AUDIT ALTER USER;
AUDIT LOCK TABLE
    BY ACCESS
    WHENEVER SUCCESSFUL;
AUDIT DELETE ON scott.emp
    BY ACCESS
    WHENEVER SUCCESSFUL;
```

接下来由 JWARD 发出下列语句：

```
ALTER USER tsimth QUOTA 0 ON users;
DROP USER djones;
```

接下来由用户 SWILLIAMS 发布以下语句：

```
LOCK TABLE scott.emp IN EXCLUSIVE MODE;
DELETE FROM scott.emp WHERE mgr = 7698;
ALTER TABLE scott.emp ALLOCATE EXTENT (SIZE 100K);
CREATE INDEX scott.ename_index ON scott.emp (ename);
CREATE PROCEDURE scott.fire_employee (empid NUMBER) AS
BEGIN
    DELETE FROM scott.emp WHERE empno = empid;
END;
/
```

```
EXECUTE scott.fire_employee(7902);
```

下面几节显示使用数据字典中的审计踪迹视图可列举的信息：

- 列举有效的语句审计选项
- 列举有效的权限审计选项
- 列举指定对象的有效对象审计选项
- 列举缺省对象审计选项
- 列举审计记录
- 列举针对 AUDIT SESSION 选项的审计记录

**24. 4. 1 列举有效的语句审计选项**

下面查询语句返回所有设置了的语句审计选项：

```
SELECT * FROM sys.dba_stmt_audit_opts;
```

| USER_NAME | AUDIT_OPTIONS | SUCCESS | FAILURE |
|-----------|---------------|---------|---------|
| -----     | -----         | -----   | -----   |

|           |            |            |            |
|-----------|------------|------------|------------|
| JWARD     | SESSION    | BY SESSION | BY SESSION |
| SWILLIAMS | SESSION    | BY SESSION | BY SESSION |
|           | LOCK TABLE | BY ACCESS  | NOT SET    |

该视图显示了可设置的语句审计选项，无论是为“SUCCESS”还是为“FAILURE”（或为这两者）设置的，无论是为“BY SESSION”还是为“BY ACCESS”设置的。

#### 24.4.2 列举有效的权限审计选项

下面查询语句返回所有设置了的权限审计选项：

```
SELECT * FROM sys.dba_priv_audit_opts;
```

| USER_NAME  | PRIVILEGE  | SUCCESS    | FAILURE |
|------------|------------|------------|---------|
| -----      | -----      | -----      | -----   |
| ALTER USER | BY SESSION | BY SESSION |         |

#### 24.4.3 列举指定对象的有效对象审计选项

下面查询语句返回所有为 SCOTT 模式中任何对象设置的审计选项：

```
SELECT * FROM sys.dba_obj_audit_opts
WHERE owner = 'SCOTT' AND object_name LIKE 'EMP%';
```

| OWNER | OBJECT_NAME | OBJECT_TY | ALT | AUD | COM | DEL | GRA | IND   | INS | LOC | ... |
|-------|-------------|-----------|-----|-----|-----|-----|-----|-------|-----|-----|-----|
| ----- | -----       | -----     | --- | --- | --- | --- | --- | ----- | --- | --- | ... |
| SCOTT | EMP         | TABLE     | S/S | -/- | -/- | A/- | -/- | S/S   | -/- | -/- | ... |
| SCOTT | EMPLOYEE    | VIEW      | -/- | -/- | -/- | A/- | -/- | S/S   | -/- | -/- | ... |

注意该视图返回了为指定对象设置的所有审计选项，视图中的信息翻译如下：

- 字符“-”指出没设置该选项；
- 字符“S”指出审计选项设置为“BY SESSION”；
- 字符“A”指出审计选项设置为“BY ACCESS”；
- 每个审计选项有两种设置：WHENEVER SUCCESSFUL 和 WHENEVER NOT SUCCESSFUL，用“/”隔开。例如，对所有成功的删除语句，将 SCOTT.EMP 的 DELETE 审计选项设置为“BY ACCESS”；对那些不成功的删除语句没做设置。

#### 24.4.4 列举缺省对象审计选项

下面查询语句返回所有缺省对象审计选项：

```
SELECT * FROM all_def_audit_opts;
```

| ALT | AUD | COM | DEL | GRA | IND | INS | LOC | REN | SEL | UPD | REF | EXE |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| S/S | -/- | -/- | -/- | -/- | S/S | -/- | -/- | S/S | -/- | -/- | -/- | -/- |

注意该视图返回类似于 USER\_OBJ\_AUDIT\_OPTS 视图和 DBA\_OBJ\_AUDIT\_OPTS 视图的信息（见前例）。

#### 24.4.5 列举审计记录

下面查询语句列举了由语句和对象审计选项产生的审计记录：

```
SELECT * FROM sys.dba_audit_object;
```

### 24.4.6 列举针对 AUDIT SESSION 选项的审计记录

下面查询语句列举了与 AUDIT SESSION 语句审计选项相对应的审计信息：

```
SELECT username, logoff_time, logoff_lread, logoff_pread,
       logoff_lwrite, logoff_dlock
FROM sys.dba_audit_session;
```

| USERNAME  | LOGOFF_TI | LOGOFF_LRE | LOGOFF_PRE | LOGOFF_LWR | LOGOFF_DLO |
|-----------|-----------|------------|------------|------------|------------|
| JWARD     | 02-AUG-91 | 53         | 2          | 24         | 0          |
| SWILLIAMS | 02-AUG-91 | 3337       | 256        | 630        |            |
| 0         |           |            |            |            |            |

## 24.5 通过数据库触发器进行审计

用户可以使用触发器实现 Oracle 内部审计。尽管用户可以将触发器记录的信息（类似于 AUDIT 语句记录的信息）写入审计记录信息中，但只是当你需要更详细地审计信息时才这样做。例如，用户可使用触发器提供表中每一行的数字化（value-based）审计信息。

**注意** 在一些领域，认为 Oracle AUDIT 语句是“安全”审计工具，而触发器提供“财政”审计工具。

决定是否创建一触发器审计数据库活动时，将标准 Oracle 数据库审计功能部件的优点和触发器审计的优点进行比较：

- 标准审计选项覆盖所有模式对象类型和结构类型的 DML 和 DDL 语句；
- 使用 Oracle 审计功能部件将自动地、集中地记录数据库审计信息；
- 使用标准 Oracle 功能部件启用的审计功能部件比使用触发器定义审计函数更容易说明和维护，错误更少；
- 也可以审计对现存审计选项做的任何变化，防止有害的数据库操作；
- 使用数据库审计功能部件，用户可以每发布一条审计语句（BY ACCESS）产生一条记录，或每为会话发布一条审计语句（BY SESSION）产生一条信息。触发器不能通过会话进行审计，每次涉及触发器审计表时都产生一条审计记录；
- 数据库审计可以审计不成功的访问，相比较之下，如果触发器审计语句回退，那么由触发器产生的审计记录也将回退；
- 标准数据库审计可记录连接和断开以及会话活动（比如物理 I/Os、逻辑 I/Os 和死锁）。

使用触发器进行复杂审计时，通常使用 AFTER 触发器。通过使用 AFTER 触发器，用户在触发器语句经受完整性约束检查之后记录审计信息，防止对完整性约束之外的语句执行审计进程。

用户何时使用 AFTER 行与 AFTER 语句触发器相对比取决于被审计的信息，例如，行触发器为表的每一行提供数字化（value-based）审计。触发器还允许用户为被审计的 SQL 语句提供“理由代码（reason code）”，这在行和语句级的审计状况中非常有用。

下面触发器审计对 EMP 表进行行级修改，它要求“理由代码”在更新之前存储在全局包变量中，触发器说明如下：

- 触发器如何提供数字化审计

- 如何使用公共程序包变量

程序代码中的注释解释了触发器的功能。

```
CREATE TRIGGER audit_employee
AFTER INSERT OR DELETE OR UPDATE ON emp
FOR EACH ROW
BEGIN
/* AUDITPACKAGE is a package with a public package
   variable REASON. REASON could be set by the
   application by a command such as EXECUTE
   AUDITPACKAGE.SET_REASON(reason_string). Note that a
   package variable has state for the duration of a
   session and that each session has a separate copy of
   all package variables. */
IF auditpackage.reason IS NULL THEN
    raise_application_error(-20201, "Must specify reason with ',
    'AUDITPACKAGE.SET_REASON(reason_string)');
END IF;

/* If the above conditional evaluates to TRUE, the
   user-specified error number and message is raised,
   the trigger stops execution, and the effects of the
   triggering statement are rolled back. Otherwise, a
   new row is inserted into the pre-defined auditing
   table named AUDIT_EMPLOYEE containing the existing
   and new values of the EMP table and the reason code
   defined by the REASON variable of AUDITPACKAGE. Note
   that the "old" values are NULL if triggering
   statement is an INSERT and the "new" values are NULL
   if the triggering statement is a DELETE. */
INSERT INTO audit_employee VALUES
    (:old.ssn, :old.name, :old.job_classification, :old.sal,
    :new.ssn, :new.name, :new.job_classification, :new.sal,
    auditpackage.reason, user, sysdate);
END;
```

如果用户希望强制性地对每次更新设置“理由代码”，用户也可任意地将“理由代码”设置为 NULL。下面 AFTER 语句触发器在触发器语句执行之后将“理由代码”设置为 NULL：

```
CREATE TRIGGER audit_employee_reset
AFTER INSERT OR DELETE OR UPDATE ON emp
BEGIN
    auditpackage.set_reason(NULL);
END;
```

前两个触发器由同一类型的 SQL 语句激活。然而，一旦对受触发器语句影响的表的每一行进行操作，就激活 AFTER 行触发器，而只当触发器语句完成之后才激活 AFTER 语句触发器。

## 第六部分 数据库资源管理

---

第六部分讨论数据库资源管理，包括以下章节：

- 第 25 章，“数据库资源管理器”。

# 25

## 数据库资源管理器

Oracle8i 通过数据库资源管理器(Database Resource Manager)提供了数据库资源管理能力，本章将向用户介绍它的用途。

本章包括以下主题：

- 什么是数据库资源管理器
- 管理数据库资源管理器
- 创建和管理资源规划
- 管理资源用户组
- 启动数据库资源管理器
- 举例
- 数据库资源管理器视图

### 25.1 什么是数据库资源管理器

当数据库资源分配决策留给操作系统处理时，用户可能遇到下列问题：

- 庞大的日常开支

当服务器数目很多时，庞大的日常开支导致操作系统在 Oracle 服务器间不断切换。

- 低效率调度

当 Oracle 服务器装有锁存器（latch）时，操作系统减少对它们的调度，这种调度是低效率的。

- 无价值的资源分配

操作系统无法在不同任务间分配 CPU 资源。

- 无法管理数据库指定资源，比如并行移动（parallel slaves）和活动的会话（sessions）。

Oracle 数据库资源管理器通过允许数据库更多地控制分配机器资源，帮助解决以上问题。

特别地，使用数据库资源管理器，可以：

- 确保某些用户处理少量的资源，不考虑对系统的加载和用户的数量。
- 按比例将 CPU 时间分配给不同的用户和应用程序，分配有效的处理资源。在数据

仓库中，分配给 ROLAP 应用程序的时间百分比可能比批作业的高。

- 限制一组用户可使用的并行程度。
- 对实例进行配置，使其能使用特殊的资源分配方法。例如，DBA 可动态地改变这个方法，不用关机和重新启动实例。

用户通过数据库资源管理器定义的 Oracle 数据库资源管理元素如下所示：

| 元素 (Element)                        | 描述                                     |
|-------------------------------------|--|
| 资源用户组 (Resource consumer group)     | 根据资源处理需求，将用户会话成组                       |
| 资源规划 (Resource plan)                | 包含指出将哪些资源分配给资源用户组的命令                   |
| 资源分配方法 (Resource allocation method) | 数据库资源管理器分配特殊资源时采用的方法/策略，由资源用户组和资源规划使用  |
| 资源规划命令 (Resource plan directives)   | 管理员使用这些命令将资源用户组与特殊规划连接起来，并在资源用户组之间分配资源 |

在下面几节中你将学习如何创建和使用这些元素：

- 管理数据库资源管理器
- 创建和管理资源规划
- 管理资源用户组
- 启动数据库资源管理器
- 举例
- 数据库资源管理器视图

参见 有关数据库资源管理器的详细概念信息，参见 “Oracle8i Concepts”，建议用户在使用数据库资源管理器之前阅读该书中相关的内容。

## 25.2 管理数据库资源管理器

用户必须具有系统权限 ADMINISTER\_RESOURCE\_MANAGER 方能管理数据库资源管理器。最有代表性的是具有 ADMIN 选项的数据库管理员具有这个权利，作为（或等价于）DBA 角色。

作为数据库资源管理器的管理员，允许执行 DBMS\_RESOURCE\_MANAGER 程序包中的所有过程，这些过程在表 25-1 中列出，在本章的后几节说明它们的用途。

表 25-1 DBMS\_RESOURCE\_MANAGER 过程

| 过程                    | 描述              |
|-----------------------|-----------------|
| CREATE_PLAN           | 命名资源规划并指定其分配方法  |
| UPDATE_PLAN           | 更新资源规划的注释       |
| DELETE_PLAN           | 删除资源规划及其指令      |
| DELETE_PLAN_CASCADE   | 删除资源规划及其所有分支    |
| CREATE_CONSUMER_GROUP | 命名资源用户组并指定其分配方法 |
| UPDATE_CONSUMER_GROUP | 更新资源用户组的注释      |



(续表)

| 过程                             | 描述                                    |
|--------------------------------|---------------------------------------|
| DELETE_CONSUMER_GROUP          | 删除资源用户组                               |
| CREATE_PLAN_DIRECTIVE          | 指出一个规划中或多级规划模式的子规划间将资源分配给资源用户组的资源规划指令 |
| UPDATE_PLAN_DIRECTIVE          | 更新规划指令                                |
| DELETE_PLAN_DIRECTIVE          | 删除规划指令                                |
| CREATE_PENDING_AREA            | 在对规划模式的改变中创建一挂起区域（临时区域）               |
| VALIDATE_PENDING_AREA          | 确认所有对规划模式的挂起的改变                       |
| CLEAR_PENDING_AREA             | 清除待解决区域中的所有的挂起的改变                     |
| SUBMIT_PENDING_AREA            | 提交所有对规划模式的改变                          |
| SET_INITIAL_CONSUMER_GROUP     | 为用户设置初始化用户组                           |
| SWITCH_CONSUMER_GROUP_FOR_SESS | 转换指定会话的用户组                            |
| SWITCH_CONSUMER_GROUP_FOR_USER | 转换属于指定用户的所有会话的用户组                     |

这些过程用途将在本章后几节解释。

作为 ADMIN 管理员，你也许选择将管理员权限授权给其他用户或角色，这样就可能使用 DBMA\_RESOURCE\_MANAGER\_PRIVS 程序包，该程序包包含了表 25-2 中列举的过程。

表 25-2 DBMA\_RESOURCE\_MANAGER\_PRIVS 过程

| 过程                           | 描述   |
|------------------------------|--|
| GRANT_SYSTEM_PRIVILEGE       | 授权 ADMINISTER_RESOURCE_MANAGER 系统权限给一用户或角色 |
| REVOKE_SYSTEM_PRIVILEGE      | 取消用户或角色的 ADMINISTER_RESOURCE_MANAGER 系统权限  |
| GRANT_SWITCH_CONSUMER_GROUP  | 授权“许可”给用户、角色或 PUBLIC，以便切换到指定的资源用户组         |
| REVOKE_SWITCH_CONSUMER_GROUP | 取消用户、角色或 PUBLIC 切换到指定的资源用户组的“许可”           |

下例中，将管理员权限授权给 SCOTT 用户，但不授权 SCOTT ADMIN 选项，因此，SCOTT 能够执行 DBMA\_RESOURCE\_MANAGER 程序包中的所有过程，但 SCOTT 不能使用 GRANT\_SYSTEM\_PRIVILEGE 过程将管理员权限授权给其他用户。

```
EXEC DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SYSTEM_PRIVILEGE -  
    (GRANTEE_NAME => 'scott', PRIVILEGE_NAME => 'ADMINISTER_RESOURCE_MANAGER',  
    ADMIN_OPTION => FALSE);
```

用户可使用 REVOKE\_SYSTEM\_PRIVILEGE 过程取消这个权限。

**注意** 只能通过使用 DBMS\_RESOURCE\_MANAGER\_PRIVS 程序包对 ADMINISTER\_RESOURCE\_MANAGER 系统权限进行授权或取消，不可通过 SQL GRANT 或 REVOKE 语句授权。

DBMS\_RESOURCE\_MANAGER\_PRIVS 程序包中其它过程将在 25.4.3 节“授权转换

权限”中讨论。

参见 参见 “Oracle8i Supplied PL/SQL Packages Reference”，以获取下列数据库资源管理器程序包的详情：

- DBMS\_RESOURCE\_MANAGER
- DBMS\_RESOURCE\_MANAGER\_PRIVS

## 25.3 创建和管理资源规划

资源规划指明属于该规划的资源用户组，和在用户组之间分配资源的指令。用户使用 DBMS\_RESOURCE\_MANAGER 程序包创建和维护数据库资源管理器的元素：资源用户组、资源规划指令和资源规划，规划信息存储在数据字典的表中，有几个视图可用来查看规划数据。

也可以使用这个程序包将初始化用户组分配给用户，为指定的会话和用户转换用户组，这将在 25.4 节“管理资源用户组”中讨论。

第一个例子只有一级规划，如图 25-1 所示，该规划为资源用户组分配资源。Great Bread 公司有一个 GREAT\_BREAD 规划，该规划在三个资源用户组之间分配 CPU 资源，分别是，分配 SALES 60% 的 CPU 时间，分配 MARKET 20% 的 CPU 时间，剩下的 20% 分配给 DEVELOP。

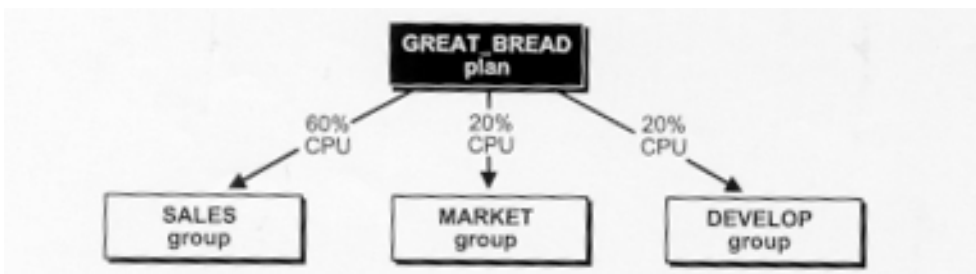


图 25-1 一个简单的规划

但是一个规划不可能只包含资源用户组，它还可包含其它规划（称为子规划）。那么 Great Bread 公司将如图 25-2 那样分配 CPU 时间。

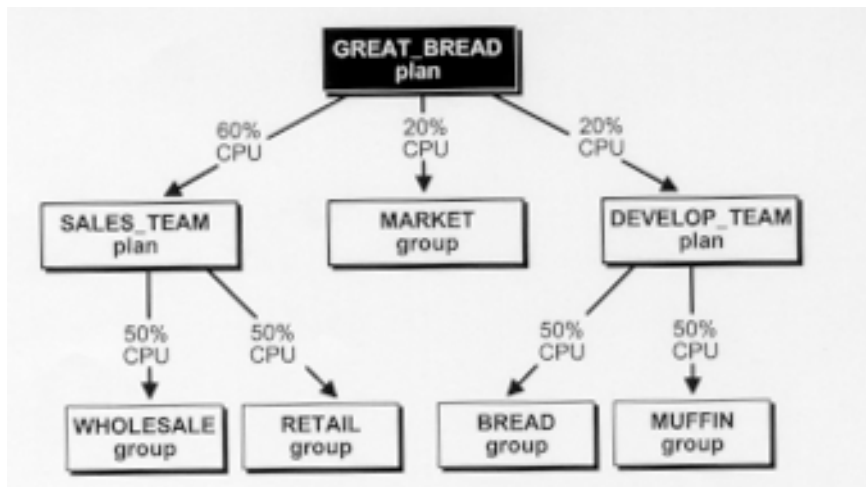


图 25-2 一个包含子规划的简单规划

这种情况下，GREAT\_BREAD 规划仍将 CPU 资源分配给 MARKET 用户组，但它现在还将 CPU 资源分配给 SALES\_TEAM 子规划和 DEVELOP\_TEAM 子规划，子规划再依次将 CPU 资源分配给用户组。图 25-2 举例说明了规划模式（plan schema），这种模式包含一个 top plan（GREAT\_BREAD）和所有分支。

一个子规划或用户组有可能有多个父规划，但规划模式中不会出现循环。如果 Great Bread 公司有日规划和夜规划，那么一个子规划有可能有多个父规划，比如日规划和夜规划都包含 SALES 子规划，但它们分别分配给 SALES 不同的 CPU 资源。

**注意** 如后面所说明的，上面的规划也可以包含一个规划指令，如 OTHER\_GROUPS，但是为了使视图更简洁，这里不显示规划指令。

### 25.3.1 为创建规划模式使用挂起区域

为了创建或修改规划模式，用户必须做的第一件事是创建一挂起区域（pending area），这是一个临时的区域，允许用户在变化激活之前升级变化和验证变化。

#### 创建一挂起区域（Creating a Pending Area）

使用下面语句创建一挂起区域：

```
EXEC DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA;
```

实际上，真正发生的事情是用户正激活挂起区域，并将所有现存的、活动的规划模式加载到挂起区域，这样可以进行更新或增加新的规划。活动规划模式（active plan schema）是那些已由数据库资源管理器存储在数据库数据字典中准备使用的模式，如果用户不激活（或创建）一挂起区域就想更新规划或增加一新的规划，那么用户将收到一条错误信息，提醒用户挂起区域不在活动状态。

视图有助于检查所有活动的资源规划模式和临时的资源规划模式，这些视图将在 25.7 节“数据库资源管理器视图”中列出。

### 验证变化 (Validating Changes)

每当用户在挂起区域有所改变时，调用如下过程进行验证：

EXEC DBMS\_RESOURCE\_MANAGER.VALIDATE\_PENDING\_AREA;

该过程检查所做的变化是否有效，运用验证过程进行验证时必须遵循以下规则：

1. 规划模式不可包含任何循环；
2. 与规划指令相关的所有规划和资源用户组必须存在；
3. 所有规划必须有指向其它规划或资源用户组的规划指令；
4. 根据 EMPHASIS 资源分配方法，任何一级的百分数加起来不可以超过 100；
5. 不能删除某活动实例正在使用的、作为 TOP PLAN 的规划；
6. 规划指令参数 PARALLEL\_DEGREE\_LIMIT\_P1 只能出现在与资源用户组相关的规划指令中（不是其它资源规划）；
7. 任何活动规划模式中的资源用户组不能超过 32 个，也就是说，一个规划至多只能有 32 子规划，最高层规划的所有叶子必须是资源用户组，在规划模式的最底层，规划指令必须针对用户组；
8. 规划和资源用户组不能同名；
9. 任何规划模式中必须有 OTHER\_GROUPS 规划指令，确保某个不被当前活动规划覆盖的会话按指定的 OTHER\_GROUPS 指令分配资源。

如果不遵循上述任何一条规则，用户将收到一条错误信息，然后用户做一些变化来解决这个问题，并重新调用验证过程。

可以创建一个“独立”的用户组，没有任何规划指令与之相关，这样允许创建的一个目前不使用的用户组，将来也许是某个规划的一部分。

### 提交变化 (Submitting Changes)

用户验证完变化之后可调用提交过程激活变化：

EXEC DBMS\_RESOURCE\_MANAGER.SUBMIT\_PENDING\_AREA;

该提交程序也进行验证，所以用户不必单独调用验证过程，但是，如果用户对规划模式做了较多的改变，调用验证过程会使调试更容易些。

SUBMIT\_PENDING\_AREA 过程在成功完成对变化的验证和提交之后清除挂起区域。

**注意** 即使 VALIDATE\_PENDING\_AREA 过程成功完成，对 SUBMIT\_PENDING\_AREA 的调用也可能失败。例如，如果在调用 VALIDATE\_PENDING\_AREA 之后，但在调用 SUBMIT\_PENDING\_AREA 之前由一个实例装载一个正被删除的规划，则可能发生这种情况。

### 清除挂起区域 (Clearing the Pending Area)

有一个过程是用来清除挂起区域的，该语句终止用户所做的所有变化：

EXEC DBMS\_RESOURCE\_MANAGER.CLEAR\_PENDING\_AREA;

用户在重做改变之前必须调用 CREATE\_PENDING\_AREA 过程。

### 25.3.2 创建资源规划

用户创建资源规划时可以指定下列参数:

| 参数                         | 描述   |
|----------------------------|--|
| PLAN                       | 规划的名称  |
| COMMENT                    | 任何注释   |
| CPU_MTH                    | CPU 资源分配方法。EMPHASIS 是 Oracle 版本的缺省方法，也是唯一的方法               |
| MAX_ACTIVE_SESS_TARGET_MTH | 为以后版本预留的参数   |
| PARALLEL_DEGREE_LIMIT_MTH  | 指出对操作的并行程度限制的一种资源分配方法，缺省方法是 PARALLEL_DEGREE_LIMIT_ABSOLUTE |

Oracle 提供一个资源规划——SYSTEM\_PLAN，该规划包含一个适应某些环境的简单的结构，其将在 25.6.2 节“Oracle 提供的规划”中举例说明。

#### 创建一个规划 (create a plan)

使用 CREATE\_PLAN 过程创建一个规划，下面创建一个名为 GREAT\_BREAD 的规划:

```
EXEC DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'great_bread', -  
comment => 'great plan');
```

#### 更新一个规划 (updating a plan)

使用 UPDATE\_PLAN 过程更新规划信息。如果用户不为 UPDATE\_PLAN 指定自变量，则它们留在数据字典中。下面语句更新 COMMENT 参数:

```
EXEC DBMS_RESOURCE_MANAGER.UPDATE_PLAN(PLAN => 'great_bread', -  
NEW_COMMENT => 'great plan for great bread');
```

#### 删除一个规划 (deleting a plan)

DELETE\_PLAN 过程删除指定的规划和所有与其关联的规划指令，下面语句删除 GREAT\_BREAD 规划及其指令:

```
EXEC DBMS_RESOURCE_MANAGER.DELETE_PLAN(PLAN => 'great_bread');
```

资源用户组本身并没有被删除，但它们不再与 GREAT\_BREAD 规划相关。

DELETE\_PLAN\_CASCADE 过程删除指定的规划及其所有分支（规划指令、子规划、资源用户组），如果 DELETE\_PLAN\_CASCADE 遇到错误，它将回退，规划模式不受改变。

### 25.3.3 创建用户组

创建用户组时可以指定下列参数:

| 参数             | 描述                              |
|----------------|---------------------------------|
| CONSUMER_GROUP | 用户组名称                           |
| COMMENT        | 任何注释                            |
| CPU_MTH        | 为资源用户组分配 CPU 资源的方法，缺省方法是 ROUND- |

数据字典中提出了两种特殊的用户组，它们不可被修改或删除。

#### ■ OTHER\_GROUPS

该组应用于属于一个用户组的会话，该用户组不是当前活动规划模式中的一部分。

OTHER\_GROUPS 必须有在任何活动规划模式中指定的资源指令。

#### ■ DEFAULT\_CONSUMER\_GROUP

这是所有还没有明确地分配初始化用户组的用户/会话的初始化用户组。

DEFAULT\_CONSUMER\_GROUP 将授权给 PUBLIC 的权限进行转换，因此，该用户组的所有用户自动被授予转换权限（参见 25.4.3 节“授予权限”）。

另外两个用户组 SYS\_GROUP 和 LOW\_GROUP (SYSTEM\_PLAN 的一部分) 将在 20.6.2 节“Oracle 提供的规划”中讨论。

#### 创建一个用户组 (creating a consumer group)

使用 CREATE\_CONSUMER\_GROUP 过程创建一个资源用户组，下面创建一个名为 SALES 的资源用户组，记住，挂起区域必须处于活动状态才能成功执行该语句。

```
EXEC DEMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (CONSUMER_GROUP =>
'sales', -COMMENT => 'retail and wholesale sales');
```

#### 更新一个用户组 (updating a consumer group)

使用 UPDATE\_CONSUMER\_GROUP 过程更新用户组的信息。如果用户不为 UPDATE\_CONSUMER\_GROUP 过程指定自变量，则这些自变量仍留在数据字典中，不发生变化。

#### 删除一个用户组 (deleting a consumer group)

DELETE\_CONSUMER\_GROUP 过程删除一指定的资源用户组。用户组删除之后，所有将该用户组作为初始用户组的用户将使用 DEFAULT\_CONSUMER\_GROUP 设置作为他们的初始用户组，所有正在运行的、属于被删除用户组的会话将转换到 DEFAULT\_CONSUMER\_GROUP。

### 25.3.4 指定资源规划指令

资源规划指令将用户组指定给资源规划，并为每个资源分配方法提供参数。当用户创建资源规划指令时，可指定下列参数：

| 参数               | 描述             |
|------------------|----------------|
| PLAN             | 资源规划名称         |
| GROUP_OR_SUBPLAN | 资源用户组或子规划名称    |
| COMMENT          | 任何注释           |
| CPU_P1           | 指出第一级 CPU 的百分比 |
| CPU_P2           | 指出第二级 CPU 的百分比 |
| CPU_P3           | 指出第三级 CPU 的百分比 |
| CPU_P4           | 指出第四级 CPU 的百分比 |

|                           |                |
|---------------------------|----------------|
| CPU_P5                    | 指出第五级 CPU 的百分比 |
| CPU_P6                    | 指出第六级 CPU 的百分比 |
| CPU_P7                    | 指出第七级 CPU 的百分比 |
| CPU_P8                    | 指出第八级 CPU 的百分比 |
| MAX_ACTIVE_SESS_TARGET_P1 | 为以后版本保留的参数     |
| PARALLEL_DEGREE_LIMIT_P1  | 设置对任何操作并行程度的限制 |

多级 CPU 资源分配提供了在一规划模式中按优先次序使用 CPU 的方法，第二级只得到第一级分配之后的资源。记住任何用户组都不能使用超过指定百分比的可用的 CPU 资源，多级不但提供一种按优先级分配的方法，还明确地指出如何使用主要的和剩余的 CPU 资源。

#### 创建资源规划指令 (creating a resource plan directive)

使用 CREATE\_PLAN\_DIRECTIVE 创建资源规划指令。下面语句为 GREAT\_BREAD 规划创建了一资源规划指令：

```
EXEC DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'great_bread', -
    GROUP_OR_SUBPLAN => 'sales', COMMENT => 'Sales group',
    CPU_P1 => 60, PARALLEL_DEGREE_LIMIT_P1 => 4);
```

类似于图 25-1 所示，用户执行下列语句完成该规划：

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (PLAN => 'great_bread',
    GROUP_OR_SUBPLAN => 'market', COMMENT => 'marketing group',
    CPU_P1 => 20);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (PLAN => 'great_bread',
    GROUP_OR_SUBPLAN => 'develop', COMMENT => 'development group',
    CPU_P1 => 20);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (PLAN => 'great_bread',
    GROUP_OR_SUBPLAN => 'OTHER_GROUPS', COMMENT => 'this one is required',
    CPU_P1 => 0, CPU_P2 => 100);
END;
```

在该规划中，SALES 用户组拥有最大的并行操作度 4，其它用户组则没有限制；同样，第一级 CPU 剩余的资源都百分之百地分配给 OTHER\_GROUP。

#### 更新资源规划指令 (updating resource plan directives)

使用 UPDATE\_PLAN\_DIRECTIVE 过程更新规划指令。下例为 DEVELOP 资源用户组改变 CPU 资源分配：

```
EXEC DBMS_RESOURCE_MANAGER.UPDATE_PLAN_DIRECTIVE (PLAN => 'great_bread', -
    GROUP_OR_SUBPLAN => 'develop', NEW_CPU_P1 => 15);
```

如果用户没有为 UPDATE\_PLAN\_DIRECTIVE 过程指定变量，它们将保留在数据字典中，不发生任何改变。

#### 删除资源规划指令 (deleting resource plan directives)

使用 DELETE\_PLAN\_DIRECTIVE 过程删除资源规划指令。

## 25.4 管理资源用户组

用户在启用数据库资源管理器之前，必须将资源用户组分配给用户。DBMS\_RESOURCE\_MANAGER 程序包除了提供过程创建、更新或删除数据库资源管理器使用的元素之外，还包含将资源用户组分配给用户的过程，和提供临时将用户会话转换给其他资源用户的过程。

DBMS\_RESOURCE\_MANAGER 程序包还可以用来将“转换”权限授权给其他用户，这样其他用户可以改变它们自己的用户组。

用户不用为下面讨论的过程使用挂起区域。

### 25.4.1 指定初始资源用户组

一个用户的初始用户组是该用户最初创建会话所归属的用户组。如果没有为用户指定初始用户组，那么该用户的初始用户组自动设置为 DEFAULT\_CONSUMER\_GROUP 用户组。

在那个用户组成为该用户的初始用户组之前，必须将“转换”权限授权给属于该用户或 PUBLIC 用户的资源用户组（参见 25.4.3 节“授权转换权限”）。初始资源用户组的转换权限不能是授权给他们的用户。

下面语句说明如何设置一个用户的初始用户组：

```
EXEC DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SWITCH_CONSUMER_GROUP ('scott',  
    'sales', - TRUE);  
EXEC DBMS_RESOURCE_MANAGER.SET_INITIAL_CONSUMER_GROUP('scott', 'sales');
```

### 25.4.2 改变资源用户组

有两个过程（DBMS\_RESOURCE\_MANAGER 程序包的一部分）允许管理员改变操作会话的资源用户组，这两个过程也可改变任何与协调程序的会话关联的并行查询从属会话（parallel query slave sessions）的用户组，但它们不能改变初始用户组。

#### 转换会话

SWITCH\_CONSUMER\_GROUP\_FOR\_SESS 过程将指定的会话立即移动到指定资源用户组，实际上，该语句可提高或降低优先权。下面语句将指定会话的资源用户变换到一个新用户组，会话标识符（SID）是 7，会话序列号（SERIAL#）是 12345，将会话变换到 HIGH\_PRIORITY 用户组。

```
EXEC DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_SESS ('17', '12345', -  
    'high_priority');
```

#### 为用户转换会话

SWITCH\_CONSUMER\_GROUP\_FOR\_SESS 过程为指定用户的会话改变资源用户组：

```
EXEC DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_SESS ('scott', -  
    'low_group');
```

### 25.4.3 授予切换权限

使用 DBMS\_RESOURCE\_MANAGER\_PRIVS 程序包将切换权限授权给一个用户、角



色或 PUBLIC，或从用户、角色或 PUBLIC 中取消。切换权限给予用户将当前资源用户组切换到指定资源用户组的权限，该程序包也允许取消切换权限。

真正的切换是通过执行 DBMS\_SESSION 程序包的一个过程实现的。已授予切换权限的用户（过程被该用户所有）可以使用 SWITCH\_CURRENT\_CONSUMER\_GROUP 过程切换到另一个资源用户组，新的用户组必须是特别授予该用户可以切换的用户组。

#### 授予切换权限

下例授权切换到资源用户组的权限。授权用户 SCOTT 可以切换到资源用户组 BUG\_BATCH\_GROUP:

```
EXEC DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SWITCH_CONSUMER_GROUP ('scott', -  
    'bug_batch_group', TRUE);
```

同时允许 SCOTT 将 BUG\_BATCH\_GROUP 的切换权限授权给其他用户。

如果授权一个用户，允许他切换到一特殊的用户组，那么该用户可将他的当前资源用户组切换到新的用户组。

如果授权一个角色，允许该角色切换到一特殊的用户组，那么该角色授权的并已启用该角色的用户可以立即他们当前的用户组切换到新的用户组。

如果授权 PUBLIC，允许切换到一特殊的用户组，那么任何用户都可切换到那个组。

如果 grant\_option 变量的值为 TRUE，那么已授予切换权限的用户同样可以将切换权限授权给其他用户。

#### 取消切换权限

下例取消用户 SCOTT 切换到 BUG\_BATCH\_GROUP 资源用户组的权限:

```
EXEC DBMS_RESOURCE_MANAGER_PRIVS.REVOKE_SWITCH_CONSUMER_GROUP ('scott', -  
    'bug_batch_group');
```

如果取消用户切换到一特别的用户组的权限，那么以后该用户切换到该用户组的操作都将失败。如果取消用户的初始资源用户组，那么该用户登陆时自动成为 DEFAULT\_CONSUMER\_GROUP 的一部分。

如果取消一角色切换到一用户组的权限，那么通过角色授予某用户切换到该用户组权限的用户以后不能再切换到该用户组。

如果取消 PUBLIC 切换到一用户组的权限，那么任何以前通过 PUBLIC 只使用该用户组的用户以后不能再切换到该用户组。

#### 使用 DBMS\_SESSION 程序包切换用户组

如果授予切换权限，用户可以使用 DBMS\_SESSION 程序包中的 SWITCH\_CURRENT\_CONSUMER\_GROUP 过程切换他们当前的用户组。

该过程允许用户切换到某用户组——该用户具有切换到这个用户组的切换权限。如果调用者是另一个过程，那么这个过程允许用户切换到一用户组——那个过程的所有者对该用户组具有切换权限。

该过程的参数有:

| 参数                     | 描述  |
|------------------------|---|
| NEW_CONSUMER_GROUP     | 目标切换用户组   |
| OLD_CONSUMER_GROUP     | 一个输出参数，存储用户组切换源的名称，今后可以切换回去   |
| INITIAL_GROUP_ON_ERROR | 如果切换出现错误，控制切换行为<br>如果为 TRUE，万一有错误，将用户切换到初始用户组。<br>如果为 FALSE，则引发错误（raise an error） |

参见 有关 DBMS\_SESSION 程序包的详情，参见 “Oracle8i Supplied PL/SQL Packages Reference”。

## 25.5 启用数据库资源管理器

通过设置 `RESOURCE_MANAGER_PLAN` 初始化参数启用数据库资源管理器，该参数指出最顶层的规划，标明实例使用的规划模式。如果该参数没指定任何规划，则数据库资源管理没有激活。

亦可使用 `ALTER SYSTEM` 语句激活或撤消数据库资源管理器，或改变当前顶层规划。下例中，顶层规划指定为 `MYDB_PLAN`：

```
ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = mydb_plan;
```

如果数据字典中不存在指定的规划，则将返回一条错误信息。

## 25.6 举例

本节提供一些资源规划模式的例子。

### 25.6.1 多级模式

下列语句创建一个多级模式，如图 25-2 所说明的，它们使用缺省规划和资源用户组方法。

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'bugdb_plan',
    COMMENT => 'Resource plan/method for bug users sessions');
DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'maildb_plan',
    COMMENT => 'Resource plan/method for mail users sessions');
DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'mydb_plan',
    COMMENT => 'Resource plan/method for bug and mail users sessions');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP =>
'Bug_Online_group',
    COMMENT => 'Resource consumer group/method for online bug users sessions');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP =>
'Bug_Batch_group',
    COMMENT => 'Resource consumer group/method for bug users sessions who run batch jobs');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP =>
```

```

'Bug_Maintenance_group',
    COMMENT => 'Resource consumer group/method for users sessions who maintane
the bug db');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP =>
'Mail_users_group',
    COMMENT => 'Resource consumer group/method for mail users sessions');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP =>
'Mail_Postman_group',
    COMMENT => 'Resource consumer group/method for mail postman');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP =>
'Mail_Maintenance_group',
    COMMENT => 'Resource consumer group/method for users sessions who maintain the mail
db');
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan',
GROUP_OR_SUBPLAN =>
'Bug_Online_group',
    COMMENT => 'online bug users sessions at level 1', CPU_P1 => 80, CPU_P2 => 0,
    PARALLEL_DEGREE_LIMIT_P1 => 8);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan',
GROUP_OR_SUBPLAN =>
'Bug_Batch_group',
    COMMENT => 'batch bug users sessions at level 1', CPU_P1 => 20, CPU_P2 => 0,
    PARALLEL_DEGREE_LIMIT_P1 => 2);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan',
GROUP_OR_SUBPLAN =>
'Bug_Maintenance_group',
    COMMENT => 'bug maintenance users sessions at level 1', CPU_P1 => 0, CPU_P2 => 100,
    PARALLEL_DEGREE_LIMIT_P1 => 3);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan',
GROUP_OR_SUBPLAN =>
'OTHER_GROUPS',
    COMMENT => 'all other users sessions at level 3', CPU_P1 => 0, CPU_P2 => 0, CPU_P3 =>
100);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan',
GROUP_OR_SUBPLAN =>
'Mail_Postman_group',
    COMMENT => 'mail postman at level 1', CPU_P1 => 40, CPU_P2 => 0,
    PARALLEL_DEGREE_LIMIT_P1 => 4);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan',
GROUP_OR_SUBPLAN =>
'Mail_users_group',
    COMMENT => 'mail users sessions at level 2', CPU_P1 => 0, CPU_P2 => 80,
    PARALLEL_DEGREE_LIMIT_P1 => 4);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan',
GROUP_OR_SUBPLAN =>
'Mail_Maintenance_group',

```

```
COMMENT => 'mail maintenance users sessions at level 2', CPU_P1 => 0, CPU_P2 => 20,
PARALLEL_DEGREE_LIMIT_P1 => 2);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan',
GROUP_OR_SUBPLAN =>
'OTHER_GROUPS',
COMMENT => 'all other users sessions at level 3', CPU_P1 => 0, CPU_P2 => 0, CPU_P3 =>
100);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'mydb_plan',
GROUP_OR_SUBPLAN =>
'maildb_plan',
COMMENT => 'all mail users sessions at level 1', CPU_P1 => 30);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'mydb_plan',
GROUP_OR_SUBPLAN =>
'bugdb_plan',
COMMENT => 'all bug users sessions at level 1', CPU_P1 => 70);
DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
```

对 VALIDATE\_PENDING\_AREA 的调用是可选的，因为在 SUBMIT\_PENDING\_AREA 中已隐式地执行了验证。

25. 6. 2 Oracle 提供的规划

Oracle 提供一缺省的资源管理规划--SYSTEM\_PLAN，为系统会话提供优先权。SYSTEM\_PLAN 定义如下：

|              |      | CPU 资源分配 |      |
|--------------|------|----------|------|
| 资源用户组        | 第一级  | 第二级      | 第三级  |
| SYS_GROUP    | 100% | 0%       | 0%   |
| OTHER_GROUPS | 0%   | 100%     | 0%   |
| LOW_GROUP    | 0%   | 0%       | 100% |

下面介绍 Oracle 提供的两个新的资源用户组，它们定义如下：

- SYS\_GROUP 是 SYS 用户和 SYSTEM 用户的初始资源用户组。
- LOW\_GROUP 提供一个优先权比该规划中的 SYS\_GROUP 和 OTHER\_GROUPS 低的用户组。它将帮助你决定哪些用户会话将是 LOW\_GROUP 的一部分，对该组的切换权限被授权给 PUBLIC。

可以使用，也可以修改或删除这三个用户组。

如果 Oracle 提供的简单的规划适合你的环境，你可使用它。

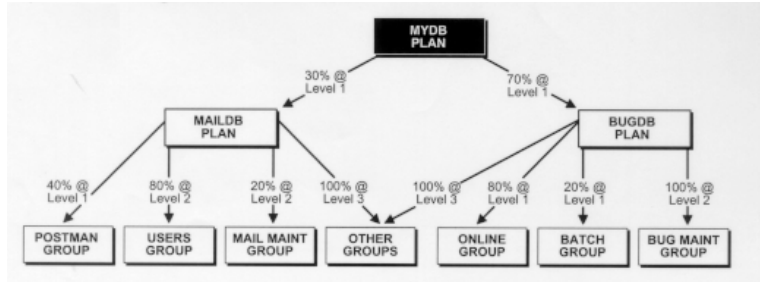


图 25-3 多级模式

## 25.7 数据库资源管理器视图

表 25-3 列出了与数据库资源管理器有关的视图:

表 25-3 数据库资源管理器视图

| 视图                             | 描述  |
|--------------------------------|---|
| DBA_RSRC_CONSUMER_GROUP_PRIVS  | 列举所有资源用户组和用户以及授权给它们的角色                      |
| DBA_RSRC_XONSUMER_GROUP        | 列举数据库中所有现存的资源用户组                            |
| DBA_RSRC_MANAGER_SYSTEM_PRIVS  | 列举所有授予数据库资源管理器权限的用户和角色                      |
| DBA_RSRC_PLAN_DIRECTIVES       | 列举数据库中所有现存的资源规划指令                           |
| DBA_RSRC_PLANS                 | 列举数据库中所有现存的资源规划                             |
| DBA_USERS                      | 包含数据库所有用户的相关信息。特别是对数据库资源管理器来说，它包含用户的初始资源用户组 |
| DBA_RSRC_CONSUMER_GROUP_PRIVS  | 列举所有授权给用户的资源用户组                             |
| DBA_RSRC_MANAGER_SYSTEM_PRIVS  | 显示 DBMS_RESOURCE_MANAGER 程序包中的所有授予系统权限的用户   |
| USERS_USERS                    | 包含当前用户的有关信息。特别是对数据库资源管理器来说，它包含当前用户的初始资源用户组  |
| V\$PARALLEL_DEGREE_LIMIT_MTH   | 显示所有有效的有并行程度限制的资源分配方法                       |
| V\$RSRC_CONSUMER_GROUP         | 显示可用来调整的活动资源用户组的有关信息                        |
| V\$RSRC_CONSUMER_GROUP_CPU_MTH | 显示资源用户组的所有有效的 CPU 资源分配方法                    |
| V\$RSRC_PLAN                   | 显示所有当前活动的资源规划的名称                            |
| V\$RSRC_PLAN_CPU_MTH           | 显示资源规划的所有有效的 CPU 资源分配方法                     |
| V\$SESSION                     | 列举当前每个会话的会话信息。特别是，列举当前每个会话的资源用户组的名称         |

可以使用这些视图查看规划模式，或者可以监控这些视图来获取信息以便调整数据库资源管理器。

**参见** 有关这些视图的详细内容，参见“Oracle8i Reference”。