

“十五”国家重点电子出版物规划项目·计算机基础知识普及和软件开发系列

2002 宝典
开发人员系列 (11)

北京希望电子出版社 总策划
胡欣杰 编写

Oracle 9i

数据库管理员指南

(V9.0.1)

 中国科学出版集团
 北京希望电子出版社

“十五”国家重点电子出版物规划项目 计算机知识普及和软件开发系列

2002 开发人员宝典系列 (1)

Oracle 9i 数据库管理员指南 (V9.0.1)

胡欣杰 编写

本版特点

- (1) 由浅入深、有章可循、编排精细、可操作性较强
- (2) 提供了针对具体的数据库管理问题的、极具参考价值的解决方案

本版 CD 为配套书



北京希望电子出版社

Beijing Hope Electronic Press

www.bhp.com.cn

2002

内 容 简 介

这是一本 Oracle 9i 管理员指导手册。

本书共分 6 个部分，由 32 章构成。其中第 1 部分：**基本的数据库管理**，介绍 Oracle 数据库管理员的职责、创建 Oracle 数据库、使用 Oracle 管理的文件以及如何启动和关闭数据库。第 2 部分：**Oracle 服务器进程和存储结构**，详细讲解管理 Oracle 进程、管理控制文件、管理联机重作日志、管理作业对列、管理表空间、管理数据文件以及撤消空间的管理。第 3 部分：**模式对象**，内容包括管理模式对象空间、管理表、管理索引、管理分区表和分区索引、管理簇、管理散列、管理视图、序列和同义词、模式对象的常规管理、检测和修复数据块损坏。第 4 部分：**数据库的安全**，介绍建立安全策略、管理用户和资源、管理用户的权限和角色、审计数据库的使用。第 5 部分：**数据库资源管理**，讲述数据库资源管理器的使用。第 6 部分：**分布式数据库的管理**，重点介绍分布式数据库的概念、管理以及开发，和分布式事物的概念与管理。

本书面向 Oracle 初、中级数据库管理员、安全维护员、网络管理员以及 Oracle 数据库应用开发人员，也可作为高校数据库专业师生参考读物。

本书内容极为丰富，有章可循、编排精细、可操作性强；并提供了针对具体的数据库管理问题的、极具参考价值的解决方案。

本版 CD 为配套书。

系 列 盘 书： “十五”国家重点电子出版物规划项目 计算机知识普及和软件开发系列
2002 开发人员宝典系列（1）

盘 书 名： Oracle 9i 数据库管理员指南(V9.0.1)

总 策 划： 北京希望电子出版社

文 本 著 作 者： 胡欣杰

C D 制 作 者： 希望多媒体开发中心

C D 测 试 者： 希望多媒体测试部

责 任 编 辑： 周 艳

出版、发 行 者： 北京希望电子出版社

地 址： 北京中关村大街 26 号，100080

网址: www.bhp.com.cn E-mail: lwm@hope.com.cn

电话: 010-62562329, 62541992, 62637101, 62637102, 62633308, 62633309

（图书发行和技术支持）

010-62613322-215（门市） 010-62547735（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 全卫

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

开 本 / 规 格： 787 毫米×1092 毫米 16 开本 35.875 印张 838.8 千字

版 次 / 印 次： 2002 年 4 月第 1 版 2002 年 4 月第 1 次印刷

本 版 号： ISBN 7-900088-48-2

印 数： 1-3000

定 价： 58.00 元（本版 CD）

说明：凡我社产品如有残缺，可执相关凭证与本社调换。

声 明

本电子版不包括第 2 章内容，请参看配套图书相关章节。

目 录

Oracle 9i Release 1(9.0.1)的新特性

表的联机重定义

Oracle 的 9.0.1 版本在 LogMiner 方面新增加的内容

第 1 部分 基本的数据库管理

第 1 章 Oracle 数据库管理员

- 1.1 Oracle 的用户类型
- 1.2 数据库管理员的任务
- 1.3 识别 Oracle 数据库软件的版本
- 1.4 数据库管理员的安全和权限
- 1.5 数据库管理员验证
- 1.6 密码文件的管理
- 1.7 数据库管理员实用程序

第 2 章 创建 Oracle 数据库

- 2.1 创建数据库之前要考虑的问题
- 2.2 Oracle 数据库配置助理
- 2.3 手动创建 Oracle 数据库
- 2.4 Oracle 9i 中有关简化数据库创建和管理的特性
- 2.5 在数据库创建过程中遇到的问题及其解决办法
- 2.6 删除数据库
- 2.7 创建数据库之后要考虑的问题
- 2.8 初始化参数和数据库的创建
- 2.9 使用服务器参数文件管理初始化参数

第 3 章 使用 Oracle 管理的文件

- 3.1 什么是 Oracle 管理的文件
- 3.2 允许创建和使用 Oracle 管理的文件
- 3.3 创建 Oracle 管理的文件
- 3.4 Oracle 管理文件的行为
- 3.5 使用 Oracle 管理文件的案例

第 4 章 启动和关闭数据库

- 4.1 启动数据库
- 4.2 改变数据库的可用性
- 4.3 关闭数据库
- 4.4 禁止操作数据库

4.5 暂停和继续数据库

第 2 部分 Oracle 服务器进程和存储结构

第 5 章 管理 Oracle 进程

- 5.1 服务器进程
- 5.2 将 Oracle 配置为共享服务器
- 5.3 关于 Oracle 的后台进程
- 5.4 监视 Oracle 实例的进程
- 5.5 为并行执行管理进程
- 5.6 为外部过程管理进程
- 5.7 终止会话

第 6 章 管理控制文件

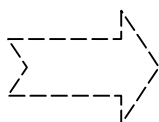
- 6.1 什么是控制文件
- 6.2 控制文件的准则
- 6.3 创建控制文件
- 6.4 在创建控制文件之后遇到的问题及其解决办法
- 6.5 备份控制文件
- 6.6 使用当前副本恢复控制文件
- 6.7 删除控制文件
- 6.8 显示控制文件信息

第 7 章 管理联机重做日志

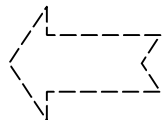
- 7.1 什么是联机重做日志
- 7.2 规划联机重做日志
- 7.3 创建联机重做日志组及其成员
- 7.4 重新定位和重新命名联机重做日志成员
- 7.5 取消联机重做日志组及其成员
- 7.6 强制实施日志切换
- 7.7 校验重做日志文件中的块
- 7.8 清除联机重做日志文件
- 7.9 查看联机重做日志信息

第 8 章 管理已存档的重做日志

- 8.1 什么是已存档的重做日志
- 8.2 在 NOARCHIVELOG 和 ARCHIVELOG 两种模式之间进行选择
- 8.3 控制存档模式
- 8.4 指定存档目标位置
- 8.5 指定日志传送模式
- 8.6 管理存档目标失败



8.7	通过指定多个 <code>ARCn</code> 进程调整存档性能
8.8	控制存档日志进程生成的跟踪输出
8.9	查看有关已存档的重做日志信息
第 9 章	使用 LogMiner 来分析重做日志文件
9.1	了解分析重做日志文件的价值
9.2	在开始之前需要知道的内容
9.3	使用 LogMiner
9.4	使用 LogMiner 的例子
第 10 章	管理作业队列
10.1	启动执行作业的进程
10.2	管理作业队列
10.3	查看作业队列信息
第 11 章	管理表空间
11.1	管理表空间的准则
11.2	创建表空间
11.3	管理表空间分配
11.4	修改表空间可用性
11.5	使用只读表空间
11.6	取消表空间
11.7	使用 <code>DBMS_SPACE_ADMIN</code> 解决表空间问题
11.8	在数据库之间传输表空间
11.9	查看表空间信息
第 12 章	管理数据文件
12.1	管理数据文件的准则
12.2	创建数据文件和向表空间添加数据文件
12.3	改变数据文件的大小
12.4	修改数据文件可用性
12.5	重命名和重定位数据文件
12.6	验证数据文件中的数据块
12.7	查看数据文件信息
第 13 章	管理撤消空间
13.1	什么是撤消
13.2	指定管理撤消空间的方式
13.3	管理撤消表空间
13.4	管理回滚段



第 3 部分 模 式 对 象

第 14 章 管理模式对象空间

- 14.1 管理数据块空间
- 14.2 设置存储参数
- 14.3 管理可恢复空间分配
- 14.4 回收空间
- 14.5 了解数据类型的空间使用

第 15 章 管理表

- 15.1 管理表的准则
- 15.2 创建表
- 15.3 更改表
- 15.4 联机重定义表
- 15.5 删除表
- 15.6 管理索引结构表
- 15.7 管理外部表
- 15.8 查看有关表的信息

第 16 章 管理索引

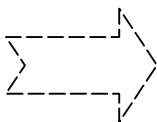
- 16.1 管理索引的准则
- 16.2 创建索引
- 16.3 更改索引
- 16.4 监视索引的空间使用
- 16.5 删除索引
- 16.6 查看索引信息

第 17 章 管理分区表和分区索引

- 17.1 什么是分区表和分区索引
- 17.2 分区的方法
- 17.3 创建分区表
- 17.4 维护分区表
- 17.5 分区表和索引的例子
- 17.6 查看有关分区表和索引的信息

第 18 章 管理簇

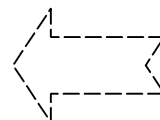
- 18.1 管理簇的准则
- 18.2 创建簇
- 18.3 更改簇
- 18.4 删除簇



18.5	查看有关簇的信息
第 19 章	管理散列簇
19.1	何时使用散列簇
19.2	创建散列簇
19.3	更改散列簇
19.4	删除散列簇
19.5	查看有关散列簇的信息
第 20 章	管理视图、序列和同义词
20.1	管理视图
20.2	管理序列
20.3	管理同义词
20.4	查看有关视图、同义词和序列的信息
第 21 章	模式对象的常规管理
21.1	在单个操作中创建多个表和视图
21.2	重命名模式对象
21.3	分析表、索引和簇
21.4	舍弃表和簇
21.5	启用和停用触发器
21.6	管理完整性约束
21.7	管理对象依赖性
21.8	管理对象名称解析
21.9	修改数据字典的存储参数
21.10	显示有关模式对象的信息
第 22 章	检测和修复数据块损坏
22.1	用于修复数据块损坏的选项
22.2	关于 DBMS_REPAIR 包
22.3	使用 DBMS_REPAIR 包
22.4	DBMS_REPAIR 的例子

第 4 部分 数据库的安全

第 23 章	建立安全策略
23.1	系统安全的策略
23.2	数据安全的策略
23.3	用户安全的策略
23.4	口令管理的策略
23.5	审计的策略
23.6	安全检查的清单



第 24 章 管理用户和资源

- 24.1 会话和用户的许可
- 24.2 用户鉴别的方法
- 24.3 管理 Oracle 用户
- 24.4 用配置文件来管理资源
- 24.5 查看有关数据库用户和配置文件的信息

第 25 章 管理用户的权限和角色

- 25.1 标识用户的权限
- 25.2 管理用户的角色
- 25.3 给用户授予权限和角色
- 25.4 撤消用户的权限和角色
- 25.5 授予和撤消行为何时生效
- 25.6 用操作系统或网络来授予角色
- 25.7 查看权限与角色的信息

第 26 章 审计数据库的使用

- 26.1 审计的准则
- 26.2 管理审计跟踪信息
- 26.3 细粒度审计
- 26.4 查看数据库的审计跟踪信息

第 5 部分 数据库资源管理

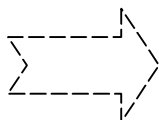
第 27 章 使用数据库资源管理器

- 27.1 什么是数据库资源管理器
- 27.2 管理数据库资源管理器
- 27.3 创建简单的资源规划
- 27.4 创建复杂的资源规划
- 27.5 管理资源用户组
- 27.6 启用数据库资源管理器
- 27.7 将所有内容结合在一起：数据库资源管理器的例子
- 27.8 监控和协调数据库资源管理器
- 27.9 查看数据库资源管理器信息

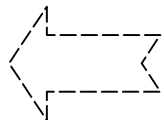
第 6 部分 分布式数据库的管理

第 28 章 分布式数据库的概念

- 28.1 分布式数据库的结构
- 28.2 数据库链接
- 28.3 分布式数据库的管理



28.4	分布式系统中的事务处理
28.5	分布式数据库的应用开发
28.6	字符集的支持
第 29 章	管理分布式数据库
29.1	管理分布式系统中的全局名
29.2	创建数据库链接
29.3	创建共享数据库链接
29.4	管理数据库链接
29.5	查看有关数据库链接的信息
29.6	创建位置透明性
29.7	管理语句透明性
29.8	管理分布式数据库：方案
第 30 章	开发用于分布式数据库系统的应用
30.1	管理应用数据的分布
30.2	控制靠数据库链接建立起来的连接
30.3	维护分布式系统中的参照完整性约束
30.4	协调分布式查询
30.5	管理远程过程中的错误
第 31 章	分布式事务的概念
31.1	什么是分布式事务
31.2	分布式事务的会话树
31.3	两阶段提交机制
31.4	悬而未决事务
31.5	分布式事务处理：案例学习
第 32 章	管理分布式事务
32.1	设置分布式事务的初始化参数
32.2	查看有关分布式事务的信息
32.3	决定如何处理悬而未决的事务
32.4	手动覆盖悬而未决的事务
32.5	清除数据字典中悬而未决的行
32.6	手动提交一个悬而未决的事务：例子
32.7	由于加锁而造成的数据访问失败
32.8	模拟分布式事务失败
32.9	管理读一致性



Oracle 9i Release 1 (9.0.1) 的新特性

Oracle 9i 带给我们一流的 Oracle 数据库服务器的新版本。它的特性包括使数据库更具可用性，以及更多的联机操作以减少脱机维护的需要。新的版本使得管理数据库更加方便。Oracle 9i 可以自动地创建和管理数据库所需的底层操作系统文件。下面主要谈一下自动管理的内容。

Oracle 9i 的性能得到增强。数据库资源管理器增加了允许对资源更为细微的控制的选项。资源客户组性能可以被更好地支持。分区方面新增加的功能允许表和索引从性能上更好地分区。安全方面新增的功能是这个版本的一个重要部分。该应用程序具有更多可以利用的以及更好的实现安全性和审计的方法。

下面总结一下本书所要讨论的 Oracle 9i 新增的特性。

表的联机重定义

新加的 DBMS_REDEFINITION PL/SQL 包提供了联机重定义表的机制。当联机重定义表时，在整个重定义的过程中都可以访问 DML。与传统的重定义表的方法相比，在可用性方面提供了更为重要的功能，从而不需要脱机进行操作。

- ANALYZE VALIDATE STRUCTURE 语句的 ONLINE 选项

当 DML 正在分析对象时，ANALYZE 语句现今可以执行有效性验证。

- 控制存档延迟

现今，为便于切换当前联机的重做日志组，Oracle 提供了一种基于时间的方法。在 primary/standby（当前使用/备用）配置中，将主站点的所有非当前日志都存储并转移到备用数据库中，这有效地限制了重做记录的数目，当需要及时进行调整时就不用加到备用数据库中了。

- 暂停数据库

Oracle 9i 包括数据库的暂停/继续特性。通过停止所有输入和输出到数据文件以及控制文件的方法，ALTER SYSTEM SUSPEND 语句可以暂停数据库的运作。当数据库暂停阶段，允许完成所有先前正在运行的 I/O 操作以及将新数据库访问放在排队等待状态。ALTER SYSTEM RESUME 语句则再继续正常的数据库操作。

- 禁止操作数据库

Oracle 9i 允许将数据库设置为禁止状态，这里只允许数据库管理员事务、查询或 PL/SQL 语句。这种禁止状态允许执行管理方面的动作，否则不能安全地完成动作。ALTER SYSTEM QUIESCE RESTRICTED 语句可以将数据库设置为禁止状态。

- 可恢复的空间分配

在空间分配失败事件之后，Oracle 为暂停、继续大型数据库操作的执行提供了一种方式。它可以允许用户纠正动作，而不是由 Oracle 数据库服务器返回一个错误给用户。在错误被纠正之后，被暂停的操作将自动继续执行。

- 更多的存档目标位置

联机重做日志的存档目标位置的最多数量已经从 5 增加到 10。

- 自动的段空间管理

本地管理的表空间允许扩展到由 Oracle 自动进行管理。对于存储在本地管理的表空间中的已用的和空闲的段，Oracle 9i 也允许自动管理。要为将要使用的 Oracle 指定段空间管理的类型，可以使用 CREATE TABLESPACE 的子句 SEGMENT SPACE MANAGEMENT 将类型指定为 AUTO（自动）或 MANUAL（手动）。

- 在执行分区维护时更新全局索引

默认情况下，已分区表上的一些表维护操作使得全局索引不可用（标记为 UNUSABLE）。所以，如果已经分区，就必须对所有的分区重新构建完整的全局索引。Oracle 9i 允许覆盖这种默认的行为。如果在 ALTER TABLE 语句中为维护操作指定了 UPDATE GLOBAL INDEX 子句，全局索引就会被更新为与基本表操作联结。

- 多种块大小

现今，Oracle 支持多种块大小。它有一个标准的块大小，由初始化参数 DB_BLOCK_SIZE 设置。除此之外，还有 4 种非标准的块大小。当创建表空间的时候指定这些非标准的块大小。标准的块大小用于 SYSTEM 表空间以及大多数其他类型的表空间。多种大小的块尺寸支持的特性允许选用不同于数据库之间的块大小来传输表空间。

- 动态高速缓冲区

现今，系统全局区的子组件高速缓冲区的大小是动态的。初始化参数 DB_BLOCK_BUFFERS 已经被一个新的动态参数 DB_CACHE_SIZE 所代替，在这里用户可以为标准的数据库块大小指定高速子缓冲区的大小。现今，高速缓冲区包含几个高速子缓冲区，这些高速子缓冲区是当为数据库指定了多种块大小时使用的。有 4 个初始化参数 DB_nK_CACHE_SIZE 可以为另外 4 种块大小指定高速子缓冲区的大小尺寸。

- 动态的 SGA

影响 SGA 大小的初始化参数已经被改成动态的。这样，就可以通过 ALTER SYSTEM SET 语句动态地更改 SGA 的大小。

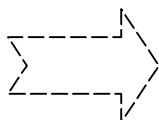
- 自动的撤消管理

以往 Oracle 都是使用回滚段存储撤消信息。Undo（撤消）可以定义为信息以便在回滚或者撤消动作中使用，进而当需要时更换到数据库。现今，Oracle 能够使用户创建一个撤消表空间以存储撤消信息。撤消表空间的使用消除了管理回滚段空间的复杂性，也使用户能够在撤消信息被覆盖之前控制更多的或是全部的撤消信息。

- Oracle 管理的文件

在 Oracle 9i 中，用 Oracle 管理文件的特性可以使用户不必直接管理这些包含 Oracle 数据库的文件。通过初始化参数 DB_CREATE_FILE_DEST 和 DB_CREATE_ONLINE_LOG_DEST_n 可以指定文件系统目录，以用做包含表空间的某种特殊类型的文件、联机重做日志文件或控制文件。之后，Oracle 可以确保只创建一个 Oracle 管理的文件，当不再需要时将它删除掉。

- 自动删除数据文件



Oracle 9i 提供一个选项,以便在使用 DROP TABLESPACE 语句撤消表空间时自动删除表空间的操作系统文件(数据文件)。ALTER DATABASE TEMPFILE 语句也有类似这样的选项,它可以删除掉与临时文件相关的操作系统文件。

- 元数据 API

新增了一个 PL/SQL 包, DBMS_METADATA.GET_DDL 允许用户获取关于模式对象的元数据(以用于创建该对象的 DDL 形式)。

- 外部表

Oracle 9i 允许用户以只读的方式访问外部表中的数据。所谓外部表就是不在数据库中的表,可以是任何一种格式,只要为它提供相应的驱动程序就可以实施访问。CREATE TABLE ... ORGANIZATION EXTERNAL 语句指定了描述外部表的元数据。现今的 Oracle 提供了 ORACLE_LOADER 访问驱动程序,这种驱动程序提供了数据映射的能力,因为其中有个 SQL*LOADER 控制文件语法的子集。

- 约束条件的增加

增强了 CREATE TABLE 或 ALTER TABLE 的 USING INDEX 子句,在创建了惟一码或主码或者使其处于有效状态时,允许用户使用该子句指定某个指定索引的创建或使用。另外,在约束条件被取消或禁止时,强制实施一个惟一码或主码约束以防止索引的丢失。

- 服务器参数文件

以往,Oracle 是将初始化参数存储在一个文本初始化参数文件中的,通常是存在客户机上。从 Oracle 9i 开始,可以选择在服务器参数文件中维护初始化参数,这是一个存储在数据库服务器上的二进制参数文件。存储在服务器参数文件中的初始化参数是持久不变的,当实例正在运行时,对这些参数所做的任何变动都将引起实例的停止运行以及重新启动。

- 默认的临时表空间

CREATE DATABASE 新增加了 DEFAULT TEMPORARY TABLESPACE 子句,该子句允许用户在创建数据库的时候创建一个默认的临时表空间。这个表空间用做默认的临时表空间,从而为没有另外分配临时表空间的用户分配一个临时表空间。

- 设置数据库时区

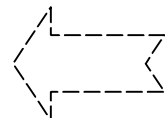
现今,CREATE DATABASE 新增加了 SET TIME_ZONE 子句,该子句允许用户将数据库的时区设置为不同于协调世界时(UTC, Coordinated Universal Time, 前身是格林尼治标准时间)的类型。当数据存储在磁盘上时,Oracle 将所有 TIMESTAMP WITH LOCAL TIME ZONE 数据标准化为数据库的时区。另外,ALTER SESSION 的 SET 子句新增加了一个 TIME_ZONE 会话参数。

- 事务命名

现今,Oracle 允许为事务指定名称。在对分布的事务进行分辨时,如果存在疑惑,事务的名称将非常有帮助,并且它将代替 COMMIT COMMENT。

- Oracle 数据库配置助理的变化

已经重新设计了 Oracle 数据库配置助理。现今这个版本,提供了几种模板,它们保存在数据库的定义中,通过这些模板可以生成数据库。Oracle 虽然提供了这些现成的模板,用户也可以对现有的模板进行修改并重新定义,从而创建自己的模板,也可以通过获取现



有的数据库的定义来创建新的模板。

当用数据库配置助理创建数据库时，可以开始就包含 Oracle 的新的示例模式，也可以在以后将它作为一个选项添加进去。这些模式是 Oracle 文档中使用的很多例子的基础。

- 监控索引的使用

ALTER INDEX 语句中新增加了一个 MONITORING USAGE 子句。它允许用户监控某个索引，以确定这个索引是否正在使用。

- 列表分区

Oracle 引入了列表分区，在对每个分区进行描述时，它允许用户为正在分区的列指定一个离散值的列表。这种列表分区方法是模型化具有离散值的分布式数据而特别设计的。它们通常不能通过排列或散列的分区方式很方便地进行设定。

- 索引结构表的散列分区

在这个版本中，添加了通过散列方式对索引结构表进行分区的内容。以前，这种表虽然能够分区，但是是通过排列的方式。

- 动态的作业队列进程

作业队列进程的创建已经成为动态的，以便于只创建所需的进程编号来执行准备好的作业。作业队列后台进程协调程序动态地产生 Jnnn 进程来执行作业。

- 数据库资源管理器为 Oracle 9i 新增的内容

下面是资源管理器新增的功能：

- 创建激活的会话组的能力。这个组包括为同一组用户中允许同时处于当前激活状态的用户会话指定最大数目。超出这个范围的其他会话需要排队才能执行，但是可以指定一个超时，超时后将放弃排队的作业。

- 基于管理员定义的条件自动从一个用户组切换到另一用户组。如果某个用户组中的成员创建的会话执行的时间超过了指定的最大时限，那么会话可以自动切换到另一个拥有不同资源需求的用户组中。

- 具有防止因为超过预设时限而被终止执行操作运行的能力。

- 具有创建撤消组的能力。这个组包括可以被用户组使用的撤消空间的数量。

- 代理验证和授权

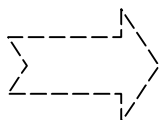
Oracle 9i 允许用户授权一个中级服务器来代表客户。ALTER USER 语句的 CONNECT THROUGH 子句可以指定这种功能。当作为客户连接时，也可以指定允许中级服务器激活的角色。

- 应用程序角色

Oracle 提供了一种机制，通过这种机制应用程序用户可以使用指定的 PL/SQL 包来授权角色。这个特性为 CREATE ROLE 语句引入了 IDENTIFIED USING *package* 子句。

- 细粒度审计

在 Oracle 传统的审计方法中，一套固定的事实被记录在审计踪迹中。只能将审计选项设置为检测对对象或权限的访问。DBMS_FGA 是一种新的 PL/SQL 包，它允许应用程序基于内容对数据访问执行细粒度审计。



Oracle 的 9.0.1 版本在 LogMiner 方面新增加的内容

9.0.1 版本的 LogMiner 添加了对许多新特性的支持。其中一些新特性从 Oracle 8.0 及以上版本的 Oracle 以来就可以与重做日志文件一起使用，而其他的特性只在 Oracle 9i 及以上版本的 Oracle 中才能与重做日志文件一起使用。

Oracle 9i 及以上版本的 Oracle 数据库生成的重做日志文件的新特性

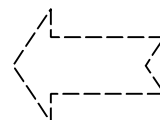
对于任何一个 Oracle 9i 及以上版本的 Oracle 生成的重做日志文件，LogMiner 都提供以下支持：

- 索引簇
- 链式和迁移的行
- 直接路径插入（此时 ARCHIVELOG 模式为有效）
- 将数据字典摘录到重做日志文件中。
- 使用联机对话框作为数据字典。
- 跟踪所有数据定义语言（DDL）的操作，它使得用户能够监视模式的进展变化。
- 在 SQL_REDO 列中查看用户执行的 DDL。还会返回有关初始的数据库用户的信息。
- 用主码信息为 updates 生成 SQL_REDO 和 SQL_UNDO。也就是说，更新行是通过主码和 ROWID 进行标识的（规定追加记录有效），从而使得将语句应用于另一个数据库更加容易。

Oracle 8.0 及以上版本的数据库生成的重做日志文件的新特性

对于任何一个 Oracle 8.0 及以上版本的 Oracle 生成的重做日志文件，LogMiner 都提供以下支持：

- 将 V\$LOGMNR_CONTENTS 数据限制到行只属于约定事务的行。这个选项使用户能够滤除掉回滚事务以及在进程中的事务。
- 基于重做日志文件中的真实数据执行查询。



第 1 部分 基本的数据库管理

第 1 部分对数据库的管理员职责进行了概述，并描述了数据库的创建以及如何启动和关闭数据库的实例方面的内容。这一部分包括：

- 第 1 章 Oracle 数据库管理员
- 第 2 章 创建 Oracle 数据库
- 第 3 章 使用 Oracle 管理的文件
- 第 4 章 启动和关闭数据库

第 1 章 Oracle 数据库管理员

本章描述数据库管理员（DBA）的职责，数据库管理员负责管理 Oracle 数据库服务器。

本章将要讨论以下内容：

- Oracle 的用户类型
- 数据库管理员的任务
- 识别 Oracle 数据库软件的版本
- 数据库管理员的安全和权限
- 数据库管理员验证
- 密码文件的管理
- 数据库管理员实用程序

1.1 Oracle 的用户类型

站点上的用户类型、角色和职责会有很大的变化。一个小站点可以只有一个数据库管理员来为应用程序开发人员和用户管理数据库。一个大型站点有必要将数据库管理员的职责划分给几个人以及几个专门的区域。

这一节包括以下几个内容：

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

1.1.1 数据库管理员

每个数据库至少需要一个数据库管理员（DBA）来管理。因为一个 Oracle 数据库系统可以很大，也可以有多个用户，通常这不是一个人就能完成的工作。在这种情况下，可以有一个 DBA 组来共同分担数据库管理员的职责。

数据库管理员的职责包括以下几种任务：

- 安装和升级 Oracle 服务器和应用程序工具
- 分配系统内存和计划将来数据库系统所需的内存
- 在应用程序开发人员设计了应用程序之后，创建主要的数据库存储结构（表空间）
- 在应用程序开发人员设计了应用程序之后，创建主要的对象（表、视图、索引）
- 如果需要，依据应用程序开发人员提供的信息修改数据库结构
- 注册用户和维护系统安全
- 保证遵守 Oracle 许可协议条款

- 控制和监控用户对数据库的访问
- 监控和优化数据库的性能
- 计划备份和恢复数据库信息
- 在磁带上保存大容量存储的数据
- 备份和恢复数据库
- 联络 Oracle 公司以获取技术支持

1.1.2 安全顾问

在某些情况下，一个站点会为数据库分配一个或多个安全顾问。安全顾问负责用户的注册，控制和监控用户对数据库的访问以及维护系统的安全。如果站点有单独的安全顾问，那么 DBA（数据库管理员）就不必担负这些责任。

1.1.3 网络管理员

一些站点有一个或多个网络管理员。网络管理员可以管理 Oracle 的网络产品，比如 Oracle Net。

1.1.4 应用程序开发人员

应用程序开发人员的任务是设计和实现数据库应用程序。他们的职责包括以下一些内容：

- 设计和开发数据库应用程序
- 设计应用程序的数据库结构
- 估算应用程序的存储需求量
- 指定对应用程序的数据库结构的修改内容
- 将以上信息传给数据库管理员
- 在整个开发阶段协调应用程序
- 在整个开发阶段建立应用程序的安全措施

应用程序开发人员可以和数据库管理员合作一起实施其中的某些任务。

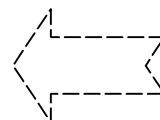
1.1.5 应用程序管理员

一个 Oracle 站点可以分配一个或多个应用程序管理员来管理某个特殊的应用程序。每个应用程序都可以有它自己的管理员。

1.1.6 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。典型的用户的职责包括以下内容：

- 在允许的地方，键入、修改和删除数据
- 根据数据生成报表



1.2 数据库管理员的任务

为了设计、实施和维护 Oracle 数据库，按先后次序的步骤，执行以下一些任务：

任务 1：评估数据库服务器的硬件设备

任务 2：安装 Oracle 软件

任务 3：规划数据库

任务 4：创建和打开数据库

任务 5：备份数据库

任务 6：注册系统用户

任务 7：具体实施数据库的设计

任务 8：备份具有完整功能的数据库

任务 9：协调数据库的性能

这些任务将在后续章节中加以讨论。

注意：如果要移植到一个新的版本，则需要在安装新版本之前备份现有的成品数据库。

1.2.1 任务 1：评估数据库服务器的硬件设备

评估的内容包括 Oracle 及其应用程序如何最大效用的使用可用的计算机资源。该评估应该揭示出以下信息：

- 有多少个磁盘驱动器可以用于 Oracle 及其数据库
- 如果有的话，有多少个专用磁带驱动器可以用于 Oracle 及其数据库
- 有多少内存容量可以用于运行 Oracle 的实例

1.2.2 任务 2：安装 Oracle 软件

作为数据库管理员，应安装 Oracle 数据库服务器软件，任何一种前台工具以及访问数据库所需的数据库应用程序。在一些分布式进程安装中，数据库由中央计算机控制，而数据库工具和应用程序则在远程计算机上执行。此时，还必须安装 Oracle Net 驱动程序，从而将远程计算机与执行 Oracle 的计算机相连。

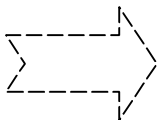
1.2.3 任务 3：规划数据库

作为数据库管理员，必须规划以下内容：

- 数据库的逻辑存储结构
- 全部的数据库设计
- 数据库的备份策略

对数据库的逻辑存储结构如何影响系统的性能和各种数据库管理操作进行规划，这非常重要。例如，在为数据库创建表空间之前，应该首先了解构成这个表空间的数据文件将有多少个，每个表空间中存储的信息是哪种类型以及数据文件物理存储在哪个磁盘驱动器中等这些问题。当规划数据库结构全部的逻辑存储时，要重视考虑当数据库真正创建和运行时这种结构所带来的效果。这样的考虑包括逻辑存储结构数据库如何影响以下内容：

- 执行 Oracle 的计算机的性能



- 在数据访问操作过程中数据库的性能
- 备份和恢复数据库进程的效率

为每个对象规划数据库对象和存储特性的相关性的设计。通过在创建对象之前规划每个对象和它的物理存储之间的关系，可以直接影响数据库作为一个整体的性能。要确信对数据库的增长进行规划。

在分布式数据库环境中，这个规划阶段是相当重要的。频繁地访问数据所在的物理位置将对应用程序的性能产生引人注目的影响。

在整个规划阶段，要为数据库开发一种备份策略。可以通过改变逻辑存储结构或数据库的设计来改善备份的效率。

对于关系和分布数据库的设计内容的讨论已经超出了本书的范围。如果还不熟悉这些设计问题，可以参考公认的工业标准文档。

第 2 部分“Oracle 服务器进程和存储结构”和第 3 部分“模式对象”将提供创建逻辑存储结构、对象和数据库完整性约束方面的信息。

1.2.4 任务 4：创建和打开数据库

通常，在完成数据库的设计任务之后，就可以开始创建数据库并将其打开了。可以使用 Oracle 数据库配置助理在安装时间创建数据库，也可以使用自己创作的脚本来创建数据库。

不管采用哪种方法，可以参考第 2 章“创建 Oracle 数据库”获取这方面的信息，还可以参考第 4 章“启动和关闭数据库”以助于数据库的创建。

1.2.5 任务 5：备份数据库

在创建数据库结构之后，执行为数据库规划好的备份策略。再创建任意一个重做日志文件，来获得第一个完整的数据库备份（联机或脱机），并计划将来定期备份数据库。

1.2.6 任务 6：注册系统用户

在备份了数据库结构之后，可以依照 Oracle 许可协议注册数据库的用户，并为用户创建相应的角色以及为这些角色授权。

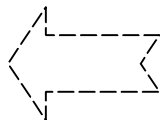
以下几个章节将有助于注册方面的内容：

- 第 23 章，建立安全策略
- 第 24 章，管理用户和资源
- 第 25 章，管理用户权限和角色

1.2.7 任务 7：具体实施数据库的设计

在创建和启动数据库并注册了系统用户之后，就可以通过创建所有所需的表空间来实施计划好的逻辑结构数据库。在这之后，可以为数据库创建各种对象。

第 2 部分“Oracle 服务器进程和存储结构”和第 3 部分“模式对象”将提供创建数据库的逻辑存储结构和对象方面的信息。



1.2.8 任务 8：备份具有完整功能的数据库

现在数据库已经全部实现，所以需要再次备份数据库。除了定期备份之外，应该在每次对数据库结构实施修改后立即备份数据库。

1.2.9 任务 9：协调数据库的性能

优化数据库的性能是数据库管理员的职责之一。此外，Oracle 提供了数据库资源管理特性，它允许数据库管理员对各种用户组进行资源分配控制。

数据库资源管理器将在第 27 章“数据库资源管理器的使用”中加以介绍。

1.3 识别 Oracle 数据库软件的版本

因为 Oracle 数据库服务器在不断地发展并且要求进一步的维护，所以 Oracle 会定期地产生新的版本。因为最初只有某些用户预订新版本或要求指定的维护功能，所以 Oracle 产品的多个版本会同时存在。

要完整地识别一个版本，需要 5 个数字。下面要讨论的就是这些数字分别代表的含义。

1.3.1 版本号码的格式

要想了解 Oracle 使用的版本级别的术语，仔细看下面的例子，该 Oracle 数据库服务器的版本号为 9.0.1.1.2。



图 1-1 Oracle 版本号说明

1.3.1.1 版本号

这是最普通的标识号。它表示软件的主要版本并包含非常重要的新的功能。

1.3.1.2 新特性版本号

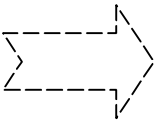
这个号码代表新特性方面的版本级别。

1.3.1.3 维护版本号

这个号码代表维护方面的版本级别。其中可能包含一些新的特性。

1.3.1.4 普通的补丁设置号码

该号码标识普通的补丁设置。这个补丁设置可以跨所有操作系统和硬件平台应用。



1.3.1.5 特殊的平台补丁设置号码

该号码代表只应用于特殊的操作系统和硬件平台的补丁设置。

1.3.2 检查当前的版本号

要识别当前安装的 Oracle 数据库服务器的版本以及弄清其他正在使用的 Oracle 组件的版本级别，可以查询数据字典，从 `PRODUCT_COMPONENT_VERSION` 中查看。下面举例说明查询的过程。其他产品的版本级别可以独立于数据库服务器而有所升级。

```
SELECT * FROM PRODUCT_COMPONENT_VERSION;
```

```
PRODUCT VERSION STATUS
```

```
-----
```

```
NLSRTL 9.0.1.0.0 Production
```

```
Oracle9i Enterprise Edition 9.0.1.0.0 Production
```

```
PL/SQL 9.0.1.0.0 Production
```

```
TNS for Solaris: 9.0.1.0.0 Production
```

当用软件报告问题时，使用查询传达出 Oracle 相关的信息是非常重要的。

而且，也可以查询 `V$VERSION` 视图来查看组件级信息。

1.4 数据库管理员的安全和权限

要完成 Oracle 中的管理任务，需要在数据库及其运行的服务器操作系统上拥有额外的权限。应该严格控制对数据库管理员账号的访问。

这一节包括以下两方面内容：

- 数据库管理员的操作系统账号
- 数据库管理员用户名

1.4.1 数据库管理员的操作系统账号

要对数据库实施一些管理员职责，就必须能够执行操作系统命令。根据运行 Oracle 的操作系统，需要一个操作系统账号或 ID 号来获得访问操作系统的权利。如果是这样，操作系统账号可能需要比数据库用户更多的操作系统权限或访问的权利（例如，执行 Oracle 软件的安装）。尽管不需要将 Oracle 文件存储在账号中，但是应该有权利访问这些文件。

1.4.2 数据库管理员用户名

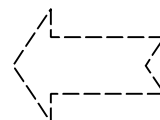
数据库将自动创建两个用户账号并授权数据库管理员角色。这两个用户账号是：

- `SYS`（最初的密码是：`CHANGE_ON_INSTALL`）
- `SYSTEM`（最初的密码是：`MANAGER`）

这两个用户名将在下面的章节中加以描述。

注意：要避免对数据字典表的不适当地访问，必须在创建 Oracle 数据库之后立即更改 `SYS` 和 `SYSTEM` 这两个用户名的密码。

建议至少另外创建一个管理员用户名，以便用来执行日常的管理任务。



注意安全部分增加的内容：在 Oracle 9i 及其后续版本中，增加了几个内容以确保默认的数据库用户账号的安全。

- 这个版本开始，在最初安装 Oracle 的数据库配置助理（DCBA）的过程中，除了 SYS、SYSTEM、SCOTT、DBSNMP、OUTLN、AURORA\$JIS\$UTILITY\$、AURORA\$ORB\$UNAUTHENTICATED 和 OSE\$HTTP\$ADMIN 以外的所有默认的数据库用户账号都将被锁定和终止。要激活某个锁定的账号，数据库管理员必须手动将其解锁并重新分配一个新的密码。
- 数据库服务器的下一个版本在最初安装数据库的过程中，Oracle 的数据库配置助理将提示 SYS 和 SYSTEM 用户输入密码，而不是给他们分配默认的密码。另外，手动发布的 CREATE DATABASE SQL 语句将需要为这两个用户指定密码。
- 用 SYSTEM 作为默认的数据库用户，不管是在任何一种安装类型中创建的还是由 CREATE DATABASE SQL 语句指定的，Oracle 9i 都将是最后一个支持它的主要版本。

1.4.3 DBA 角色

名为 DBA 的预先定义的角色，是由每个 Oracle 数据库自动创建的。这个角色包括大多数的数据库系统权限。因此，他的能力非常强大，他也应该是惟一授权操作整个数据库管理的对象。

注意：DBA 角色不包括 SYSDBA 或 SYSOPER 系统权限。这两种权限是非常特殊的管理权限，只允许一个管理员来执行基本的数据库管理任务，比如启动和关闭数据库。这些系统权限将在“管理权限”一节中加以讨论。

1.4.4 SYS

当创建任何一个数据库时，都将自动创建用户 SYS 并授权 DBA 角色。

所有为数据库的数据字典建立的基本表和视图都存储在 SYS 模式中。这些基本表和视图对于 Oracle 的操作是至关重要的。要维护数据字典的完整性，SYS 模式中的表只能由 Oracle 来操纵。任何用户或者数据库管理员都不能修改这些表，在 SYS 用户模式中没有一个人能创建表。（但是，如果需要，可以更改数据字典设置中的存储参数。）

要确保大多数数据库用户都不能使用 SYS 账号进行连接。

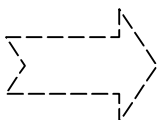
1.4.5 SYSTEM

当创建数据库时，也将自动创建 SYSTEM 用户并授权 DBA 角色。

用 SYSTEM 用户名创建显示管理信息的外部的表和视图以及各种 Oracle 选项和工具使用的内部的表和视图。在 SYSTEM 模式下，不要创建任何个人用户感兴趣的表。

1.5 数据库管理员验证

作为 DBA，经常要执行一些特殊的操作，比如关闭和启动数据库。因为只有 DBA 才能执行这些操作，所以数据库管理员的用户名需要一个安全的验证模式。



这一节包括以下一些内容：

- 管理权限
- 选择一种验证方法
- 使用操作系统验证
- 使用密码文件验证

1.5.1 管理权限

管理员需要管理权限才能执行基本的数据库操作，这些权限可以通过两种特殊的系统权限 `SYSDBA` 和 `SYSOPER` 来授权。管理员必须拥有其中一种权限，可以依照所需的授权级别进行选择。

注意：`SYSDBA` 和 `SYSOPER` 两种系统权限允许访问数据库实例，即便是当数据库还未打开时也可以访问。对这两种权限的控制已经完全超出了数据库本身的范围。

1.5.1.1 `SYSDBA` 和 `SYSOPER` 系统权限

这两种系统权限授权的操作包括以下一些内容：

系统权限	授权的操作
SYSDBA	■ 执行 <code>STARTUP</code> 和 <code>SHUTDOWN</code> 操作 ■ <code>ALERT DATABASE</code>：打开、登录、备份或更改字符表 ■ <code>CREATE DATABASE</code> ■ <code>CREATE SPFILE</code> ■ <code>ARCHIVELOG</code> 和 <code>RECOVERY</code> ■ 包括 <code>RESTRICTED SESSION</code> 权限 该系统权限可以将用户与 <code>SYS</code> 用户相连接，非常有效。
SYSOPER	■ 执行 <code>STARTUP</code> 和 <code>SHUTDOWN</code> 操作 ■ <code>CREATE SPFILE</code> ■ <code>ALERT DATABASE OPEN/MOUNT/BACKUP</code> ■ <code>ARCHIVELOG</code> 和 <code>RECOVERY</code> ■ 包括 <code>RESTRICTED SESSION</code> 权限 该权限允许用户执行基本的操作任务，但是不具备查看用户数据的能力。

使用所授的这些权限，依赖于选择使用的验证方法。

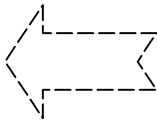
当使用用户名和密码连接 `SYSDBA` 或 `SYSOPER` 权限时，也就与默认的模式相连接，而不是与用户名相关联的模式相连。对于 `SYSDBA`，默认模式是 `SYS`；对于 `SYSOPER`，默认模式是 `PUBLIC`。

1.5.1.2 与管理权限相连接的例子

在这个例子中，当与 `SYSDBA` 系统权限相连接时，用户被分配了另一个 `SYS` 模式。假设用户 `scott` 已经发布以下一些语句：

```
CONNECT scott/tiger
CREATE TABLE scott_test (name VARCHAR2(20));
```

之后，用户 `scott` 发布以下一些语句：



```
CONNECT scott/tiger AS SYSDBA
SELECT * FROM scott_test;
```

现在，他收到了一个错误信息“scott_test 不存在”。这是因为 scott 引用了默认的 SYS 模式，而这个表是在 scott 模式下创建的。

1.5.2 选择一种验证方式

验证数据库管理员可以用以下几种方式：

- 操作系统（OS）验证方式
- 密码文件验证方式

注意：这些方式代替了早先的 Oracle 版本提供的 CONNECT INTERNAL 句法。现在，不再允许使用 CONNECT INTERNAL 句法。

可以在数据库所在的同一台机器上本地管理数据库，也可以从一个单独的远程客户机上管理许多不同的数据库，选择不同的方式将影响到验证方式的选择。图 1-2 阐明了数据库管理员验证模式的选择过程。

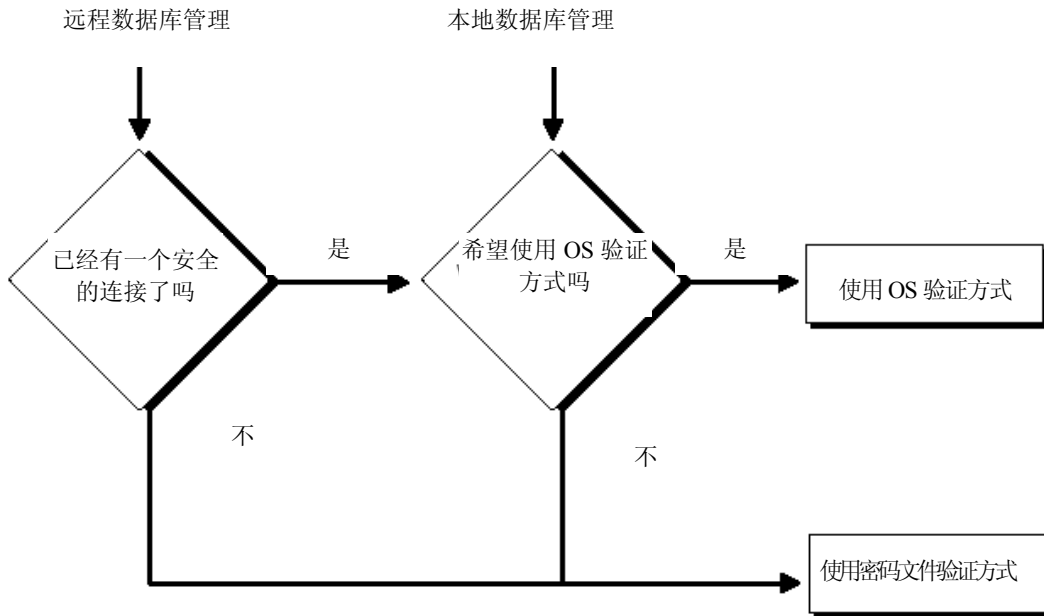
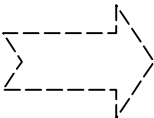


图 1-2 数据库管理员验证方式

如果正在实施的是远程数据库管理，则需要参考 Oracle Net 文档来决定正在使用的连接是否是一个安全的连接。大多数常用的连接协议，比如 TCP/IP 和 DECnet，都是不安全的。

1.5.2.1 不安全的远程连接

当有权限的用户通过一个不安全的连接与 Oracle 连接时，必须使用密码文件验证方式。使用这种验证方式时，数据库将使用一个密码文件来跟踪已经拥有 SYSDBA 或 SYSOPER 系统权限的数据库的用户名。



这种验证的形式将在“使用密码文件验证方法”一节中加以讨论。

1.5.2.2 本地连接和安全的远程连接

当有权限的用户通过一个本地连接或安全的远程连接与 Oracle 连接时,将有以下选项:

- 倘若数据库具备一个密码文件,并且数据库管理员已经拥有 SYSDBA 或 SYSOPER 系统权限,可以使用密码文件验证方式进行连接。
- 如果服务器没有使用密码文件,或者数据库管理员还没有 SYSDBA 或 SYSOPER 系统权限(因此也就不在密码文件中),那么可以使用 OS 验证方式。在大多数操作系统上,数据库管理员采用的 OS 验证都会将数据库管理员的 OS 用户名放在一个特殊的组中,通常用 OSDBA 表示。

1.5.3 使用操作系统验证方式

这一节主要描述如何使用操作系统来验证管理员。

1.5.3.1 使用 OS 验证所需的准备

要使用操作系统实施管理用户的验证,可以按如下步骤进行:

- ①为用户创建操作系统账号。
- ②将用户添加到定义为 OSDBA 或 OSOPER 的操作系统的组中。
- ③确保初始化参数 REMOTE_LOGIN_PASSWORDFILE 设置为 NONE,这是该参数的缺省值。

1.5.3.2 使用 OS 验证方式进行连接

用户可以作为管理用户接受验证,并与本地数据库相连,或者通过一个安全的连接与远程数据库相连,具体方法是键入以下 SQL*Plus 命令:

```
CONNECT / AS SYSDBA
```

```
CONNECT / AS SYSOPER
```

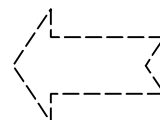
1.5.3.3 OSDBA 和 OSOPER

当使用 OS 验证方式时,有两种特殊的操作系统组控制着数据库管理员的登录。这些组通常命名为 OSDBA 和 OSOPER。组的创建以及命名是数据库安装进程的一部分。指定的名称依照操作系统的不同而有所差异,可以覆盖 Oracle Universal Installer (Oracle 通用安装程序)预先假定的缺省名称。如何创建 OSDBA 和 OSOPER 组是操作系统的内容,这里不再介绍。

如果数据库管理员是 OSDBA 组的成员,并在与数据库连接时被指定为 SYSDBA,那么他将拥有 SYSDBA 系统权限。

如果数据库管理员是 OSOPER 组的成员,并在与数据库连接时被指定为 SYSOPER,那么他将拥有 SYSOPER 系统权限。

如果数据库管理员不是相关的操作系统组的成员,也就没有 SYSDBA 或 SYSOPER 系统权限,而 CONNECT 命令也将失败。



1.5.4 使用密码文件验证方式

这一节主要描述如何使用密码文件来验证管理用户。

1.5.4.1 使用密码文件验证所需的准备

要使用密码文件实施管理用户的验证，可以按如下步骤进行：

①为用户创建操作系统账号。

②如果还没有创建，则使用 ORAPWD 实用程序来创建密码文件：

```
ORAPWD FILE=filename PASSWORD=password ENTRIES=max_users
```

③将初始化参数 REMOTE_LOGIN_PASSWORDFILE 设置为 EXCLUSIVE。

④作为 SYS 用户（或者是作为拥有管理权限的另一个用户）与数据库连接。

⑤如果用户还不在于数据库中，则需要先创建用户，并授权用户 SYSDBA 或 SYSOPER 系统权限：

```
GRANT SYSDBA to scott;
```

该语句将用户添加到密码文件中，从而可以作为 SYSDBA 进行连接操作。

1.5.4.2 使用密码文件验证方式进行连接

用户可以通过使用 SQL*Plus CONNECT 命令接受验证，并与本地或远程数据库相连。必须使用带 AS SYSDBA 或 AS SYSOPER 子句的用户名和密码进行连接。例如，用户 scott 已经被授予 SYSDBA 的权限，所以可以按如下语句进行连接：

```
CONNECT scott/tiger AS SYSDBA
```

然而，如果 scott 没有被授予 SYSOPER 的权限，那么下面的命令会导致失败：

```
CONNECT scott/tiger as SYSOPER;
```

1.6 密码文件的管理

可以使用密码文件创建实用程序 ORAPWD 来创建密码文件。对于一些操作系统，可以作为标准安装的一部分来创建该文件。

这一节包括以下一些内容：

- ORAPWD 的使用
- 设置 REMOTE_LOGIN_PASSWORDFILE
- 往密码文件中添加用户
- 密码文件的维护

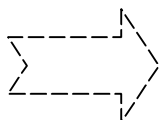
1.6.1 ORAPWD 的使用

如果在使用密码文件创建实用程序中没有提供任何参数，那么将会收到指向正确使用该命令的消息，下面举例说明：

```
orapwd
```

```
Usage: orapwd file=<fname> password=<password> entries=<users>
```

```
where
```



file - name of password file (mand),
 password - password for SYS (mand),
 entries - maximum number of distinct DBAs and OPERs (opt),
 There are no spaces around the equal-to (=) character.

例如, 下面的命令创建一个名为 acct.pwd 的密码文件, 该文件允许多达 30 个拥有不同密码的授权用户:

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

接下来, 介绍 ORAPWD 实用程序中的参数。

1.6.1.1 FILE

该参数设置创建的密码文件的名称。必须为文件指定一个完整的路径名。该文件的内容经过加密, 并且不能直接进行读取。该参数是具有强制性的。

密码文件允许的文件名的类型可以由操作系统指定。某些操作系统要求特定格式的密码文件并存储在特定的目录中。其他的操作系统允许使用环境变量来指定密码文件的名称和位置。

如果正在使用 Oracle 9i Real Application Clusters (Oracle 9i 实时应用程序簇) 运行多个 Oracle 实例, 那么每个实例的环境变量都应指向同一个密码文件。

警告: 保护密码文件及指向密码文件位置的环境变量, 对于系统的安全性能至关重要。访问这些内容的每个用户都有危害连接安全性的可能。

1.6.1.2 PASSWORD

该参数为 SYS 用户设置密码。如果在与数据库连接之后, 通过发布 ALTER USER 语句来更改 SYS 用户的密码, 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将被更改。这个参数也是强制性的。

1.6.1.3 ENTRIES

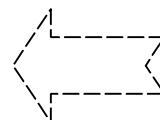
该参数指定密码文件可以接受的登录个数。这个数目和允许作为 SYSDBA 或 SYSOPER 与数据库相连的用户的数目相符。真正允许的登录的个数可以比这个数目多, 因为 ORAPWD 实用程序会继续分配密码直到操作系统块被添满。例如, 如果操作系统块的大小为 512 字节, 它可以包含 4 个密码入口。而通常分配的密码入口的数目是 4 的倍数。

当从密码文件中添加和删除用户时, 也可以重复使用入口。如果想指定为 REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE, 并且想将 SYSDBA 和 SYSOPER 权限授予用户时, 就需要这个参数。

警告: 当超出了分配的密码入口的数目时, 就必须创建一个新的密码文件。要避免这种情况的出现, 最好分配的入口数目比开始想像需要的入口数目多。

1.6.2 设置 REMOTE_LOGIN_PASSWORDFILE

除了要创建密码文件, 还必须设置初始化参数 REMOTE_LOGIN_PASSWORDFILE 为合适的值。可取的值如下所述:



值	说明
NONE	将参数设置为 NONE 会引起 Oracle 采取密码文件不存在时的举动。也就是说，不安全的连接方式不允许没有授权的连接。NONE 是该参数的缺省值。
EXCLUSIVE	设置为 EXCLUSIVE 的密码文件只能用于一个数据库。只有 EXCLUSIVE 的文件可以包含不是 SYS 的用户的名称。使用 EXCLUSIVE 的密码文件可以将 SYSDBA 和 SYSOPER 权限授予个人用户并使他们相连接。
SHARED	设置为 SHARED 的密码文件可以用于多个数据库。但是，SHARED 状态的密码文件认可的用户只有 SYS。而且，不能往这种密码文件中添加用户。所有需要 SYSDBA 和 SYSOPER 系统权限的用户都必须使用同一个名称 SYS 和密码进行连接。如果只有一个 DBA 管理着多个数据库，那么该选项将非常有用。

建议：要获得最大级别的安全性，在创建密码文件之后，应该立即将初始化参数 REMOTE_LOGIN_PASSWORDFILE 设置为 EXCLUSIVE 状态。

1.6.3 往密码文件中添加用户

当把 SYSDBA 和 SYSOPER 的权限授权给一个用户时，用户的名称和权限信息也就添加到密码文件中了。如果服务器还没有 EXCLUSIVE 状态的密码文件（也就是说，如果初始化参数 REMOTE_LOGIN_PASSWORDFILE 设置为 NONE 或 SHARED），那么在授予这些权限时将会收到一条错误信息。

只要用户至少拥有两个权限中的一个，则用户的名称就会保存在密码文件中。如果取消用户的这些权限，用户也就被从密码文件中删除。

1.6.3.1 创建密码文件和添加新的用户的步骤

- ①按照“ORAPWD 的使用”一节中介绍的步骤创建密码文件。
- ②将初始化参数 REMOTE_LOGIN_PASSWORDFILE 设置为 EXCLUSIVE 状态。
- ③使用 SYSDBA 权限进行连接，如下所示：

```
CONNECT SYS/password AS SYSDBA
```

- ④启动实例并创建所需的数据库或者登录和打开现有的数据库。
- ⑤创建所需的用户。将 SYSDBA 或 SYSOPER 权限授权给合适的用户或是管理员自己。

将 SYSDBA 或 SYSOPER 权限授权给用户，将引起用户名添加到密码文件的操作。这使得用户通过指定的用户名和密码作为 SYSDBA 或 SYSOPER 来连接数据库（而不是使用 SYS）。如果条件与 OS 验证匹配，密码文件的使用不会妨碍 OS 验证用户的连接。

1.6.3.2 授权和取消 SYSDBA 和 SYSOPER 权限

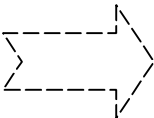
如果服务器使用的是 EXCLUSIVE 状态的密码文件，那么，可以使用 GRANT 语句将 SYSDBA 或 SYSOPER 权限授权给用户，如下所示：

```
GRANT SYSDBA TO scott;
```

使用 REVOKE 语句取消用户的 SYSDBA 或 SYSOPER 权限，如下所示：

```
REVOKE SYSDBA FROM scott;
```

因为 SYSDBA 和 SYSOPER 这两个权限是功能最强大的数据库权限，所以不必使用



ADMIN OPTION。当前只有作为 SYSDBA（或 INTERNAL）连接的用户才能有权给另一个用户授权和取消 SYSDBA 或 SYSOPER 系统权限。因为在数据库启动后只有角色是可用的，所以这些权限不能授予角色。不要将 SYSDBA 和 SYSOPER 数据库权限与操作系统角色混淆起来，它们是完全独立的特征。

1.6.3.3 查看密码文件成员

要查看数据库中哪些用户被授予了 SYSDBA 和/或 SYSOPER 权限，可以使用 V\$PFILE_USERS 视图。通过该视图显示以下的列，下面分别加以说明：

列	说明
USERNAME	该列包含密码文件认可的用户的名称。
SYSDBA	如果该列的值为 TRUE，那么该用户就被登记为拥有 SYSDBA 系统权限。
SYSOPER	如果该列的值为 TRUE，那么该用户就被登记为拥有 SYSOPER 系统权限。

1.6.4 密码文件的维护

这一节介绍如何做以下工作：

- 如果密码文件已经满员，需要扩展密码文件用户的数目
- 删除密码文件
- 避免改变密码文件的状态

1.6.4.1 扩展密码文件用户的数目

当试图将 SYSDBA 或 SYSOPER 系统权限授权给某个用户时，如果收到文件已满的错误信息（ORA-1996），那么就必须创建一个更大的密码文件并对用户重新授权。

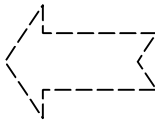
替换密码文件的步骤

- ①通过查询 V\$PFILE_USERS 视图，注意拥有 SYSDBA 或 SYSOPER 权限的用户。
- ②关闭数据库。
- ③删除现有的密码文件。
- ④按照“ORAPWD 的使用”一节中介绍的步骤使用 ORAPWD 实用程序创建新的密码文件。要确保 ENTRIES 参数设置为比需要的大一些的数目。
- ⑤按照“往密码文件中添加用户”一节中介绍的步骤添加用户。

1.6.4.2 删除密码文件

如果决定不再需要密码文件来验证用户，那么可以删除该密码文件并将初始化参数 REMOTE_LOGIN_PASSWORDFILE 设置为 NONE。在删除了文件之后，只有那些可以通过 OS 验证方式的用户才能执行数据库的管理操作。

警告：如果正有一个使用 REMOTE_LOGIN_PASSWORDFILE=EXCLUSIVE（或 SHARED）登录的数据库或实例，那么不要删除或修改密码文件。否则，不能使用密码文件进行远程的重新连接。即使是替换这个密码文件，也不能使用新的密码文件，原因是时间标记和校验和会出错。



1.6.4.3 改变密码文件的状态

密码文件的状态存储在密码文件中。当首次创建密码文件时，它的默认状态是 SHARED。可以通过设置初始化参数 REMOTE_LOGIN_PASSWORDFILE 来改变密码文件的状态。在启动一个实例时，Oracle 会从存储在客户机的参数文件中找回这个参数的值。当登录数据库时，Oracle 会将这个参数的值与存储在密码文件中的值进行比较。如果值不匹配，Oracle 会重写存储在文件中的值。

警告：使用警告信息来确保 EXCLUSIVE 状态的密码文件不会意外地被改写为 SHARED 状态。如果计划从多个客户机上启动实例，那么每个客户机都必须有一个初始化参数文件，并且每个文件中的初始化参数 REMOTE_LOGIN_PASSWORDFILE 的值必须一样。否则，密码文件的状态会随着实例启动的地方的不同而有所差异。

1.7 数据库管理员实用程序

有几个实用程序可以有助于维护 Oracle 数据库中的数据。这一节介绍其中的两个实用程序：

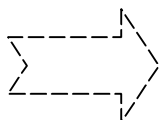
- SQL*Loader
- Export 和 Import

1.7.1 SQL*Loader 实用程序

数据库管理员和 Oracle 的其他用户都可以使用 SQL*Loader 实用程序。它将数据从标准的操作系统文件（比如，文本或 C 数据格式的文件）装载到 Oracle 数据库表中。

1.7.2 Export 和 Import 实用程序

Export 和 Import 实用程序可以将 Oracle 格式中现有的数据移出或移入 Oracle 数据库，而这些 Oracle 数据库可以运行在相同或不同的操作系统中。



第 2 章 创建 Oracle 数据库

HOPE



声 明

本电子版不包括本章内容，请看配套图书相关章节。

北京希望电子出版社

2001

第 3 章 使用 Oracle 管理的文件

本章讨论 Oracle 管理文件的使用问题，包含以下内容：

- 什么是 Oracle 管理的文件
- 允许创建和使用 Oracle 管理的文件
- 创建 Oracle 管理的文件
- Oracle 管理文件的行为
- 使用 Oracle 管理文件的案例

3.1 什么是 Oracle 管理的文件

使用 Oracle 管理文件可以简化对 Oracle 数据库的管理。这样，就不需要 DBA 直接管理 Oracle 数据库的操作系统文件。根据数据库对象而不是文件名来指定操作。Oracle 内部使用标准的文件系统接口创建和删除以下数据库结构所需的文件：

- 表空间
- 联机重做日志文件
- 控制文件

通过初始化参数，可以为特殊类型的文件指定文件系统目录。继而，Oracle 可以确保为其创建惟一的 Oracle 管理的文件，并在不需要的时候将该文件删除。

这个特性并不影响管理文件的创建或命名，这些管理文件可以是跟踪文件、审计文件、报警文件以及核心文件。

3.1.1 谁可以使用 Oracle 管理的文件

Oracle 管理的文件适用于以下类型的数据库：

- 低端或测试数据库
- 支持以下内容的数据库：
- 支持磁盘分片/RAID 和动态扩展逻辑卷的逻辑卷管理器
- 提供大型、可扩展文件的文件系统

Oracle 管理文件的特性还不能适用于方便地管理使用未加工过的磁盘的系统。这一特性在磁盘空间分配方面，提供了与操作系统功能的更好的一致性，由于没有操作系统支持未加工的磁盘分配（需要手动加工），这一特性无法补救。在其他方面，因为 Oracle 管理文件需要使用操作系统的文件系统（不像原始磁盘），所以 DBA 将失去控制文件如何放置在磁盘上的能力，从而也就失去某些 I/O 协调能力。

3.1.1.1 什么是逻辑卷管理器

逻辑卷管理器（LVM）是一种适用于大多数操作系统的软件包。有时，又称它为逻辑磁盘管理器（LDM）。可以将多个物理磁盘合并到一个连续或相邻的地址空间，从而高层软件视其为一个磁盘。与任何一个底层物理磁盘相比，LVM 使得逻辑卷拥有更好的容量、

性能、可靠性和可用性等特点。LVM 运用了一些技术来实现以上的特点，这些技术包括镜像、磁盘分片、级连和 RAID 5。

某些 LVM 还拥有在它创建后更改逻辑卷的特点，即使在使用 LVM 的时候也可以进行更改。此时，可以调整逻辑卷的大小或者镜像，还可以重新将逻辑卷放置在其他的物理磁盘上。

3.1.1.2 什么是文件系统

文件系统是一种构建在邻接的磁盘地址空间中的数据结构。文件管理器（FM）是操作文件系统的软件包，但是，有时也称为文件系统。所有操作系统都有文件管理器。文件管理器的主要任务是在文件系统中为文件分配和取消分配磁盘空间。

文件系统允许为大量的文件分配磁盘空间。每个文件对应用程序（比如，Oracle）而言，好像是有几个在邻接的地址空间中。实际上，文件在文件系统的磁盘空间上，可能并不邻接。可以创建、读、写、调整大小和删除文件。每个文件都有各自的名称。

通常，文件系统由 LVM 构建在逻辑卷的最上层。因此，一个特殊文件系统中的所有文件都具有从底层逻辑卷继承下来的相同的性能、可靠性和可用性特征。文件系统是其中的所有文件共享的存储池。如果文件系统超出了空间范围，那么其中的文件没有一个可以增加。文件系统可利用的空间并不影响其他文件系统的空间。但是，一些 LVM/FM 组合允许为文件系统添加或删除空间。

一个操作系统可以支持多个文件系统。这些文件系统可以赋予不同的文件以不同的存储特征，也可以将可利用的磁盘空间划分成互不影响的存储池。

3.1.2 使用 Oracle 管理文件的好处

使用 Oracle 管理文件将带来以下好处：

- 使得数据库的管理更加方便。

不需要构造文件名和定义特定的存储要求。有一套相容的规则可以用于命名相关的文件。文件系统可以定义存储特征并分配相应的存储池。

- 减少由管理员指定错误文件而产生的破坏。

每个 Oracle 管理的文件及其文件名都是惟一的。在两个不同的数据库中使用同一个文件是最常见的错误，这样将会导致花费大量的停机时间并丢失已经提交的事务。另一种错误是一个文件使用两个不同的名称，这也将导致较多的错误。

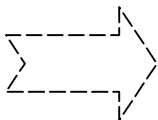
- 减少由废弃文件消耗引起的磁盘空间的浪费。

当某些旧的 Oracle 管理的文件不再有用时，Oracle 将自动删除这些文件。在大型系统中，往往因为人们不能确定某个文件是否还有用，从而浪费了大量的磁盘空间。删除磁盘上不用的文件将会简化管理的任务，并能防止在删除文件时可能出现的错误。

- 简化测试和开发数据库的创建过程。

使得花费在决定有关文件结构及其命名时的时间最少，并且执行更少的文件管理任务。这样，可以更好地将精力集中在满足测试或开发数据库真正的需求上。

- Oracle 管理文件使得便携式第三方工具的开发工作更容易展开。



Oracle 管理的文件不需要将操作系统特定的文件名称放到 SQL 脚本中。

3.1.3 Oracle 管理的文件和现有的功能

使用 Oracle 管理的文件并不消除任何一种现有的功能。现有的数据库操作将不受影响。新文件可以创建为 Oracle 管理的文件，而原来的文件依旧按照原来的方式进行管理。因此，数据库可以同时包含 Oracle 管理的文件和非 Oracle 管理的文件。

3.2 允许创建和使用 Oracle 管理的文件

以下的初始化参数允许数据库服务器使用 Oracle 管理文件的特性：

参数	说明
DB_CREATE_FILE_DEST	定义默认的文件系统目录位置，当在创建操作中没有给出文件指定时，Oracle 在此创建数据文件或临时文件。如果没有指定 DB_CREATE_ONLINE_LOG_DEST_n 参数，联机重做日志和控制文件也用它作为默认的文件系统目录。
DB_CREATE_ONLINE_LOG_DEST_n	当在创建操作中没有给出文件指定时，为联机重做日志和控制文件的创建定义默认的文件系统目录位置。可以多次使用这个初始化参数，这里的 n 指联机重做日志或控制文件的多路复用备份。最多可以指定 5 个多路复用备份。

由上面其中的一个参数指定的文件系统目录必须已经存在：Oracle 不会创建该目录。该目录必须允许 Oracle 在其中创建文件。

在创建文件时，如果没有明确为其指定位置，可以使用默认的位置。Oracle 创建了文件名，因此创建的文件也就是一个 Oracle 管理的文件。

这两个初始化参数都是动态的，而且可以使用 ALTER SYSTEM 或 ALTER SESSION 语句进行设置。

3.2.1 设置初始化参数 DB_CREATE_FILE_DEST

在初始化参数文件中包含 DB_CREATE_FILE_DEST 初始化参数以确定数据库服务器的默认位置，以便创建以下文件：

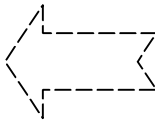
- 数据文件
- 临时文件
- 联机重做日志文件
- 控制文件

可以指定文件系统目录的名称，以便为上述操作系统文件的创建提供默认的位置。在下面的例子中，使用 “/u01/oradata/payroll” 作为创建 Oracle 管理文件的默认目录。

```
DB_CREATE_FILE_DEST = '/u01/oradata/payroll'
```

3.2.2 设置初始化参数 DB_CREATE_ONLINE_LOG_DEST_n

在初始化参数文件中包含 DB_CREATE_ONLINE_LOG_DEST_n 初始化参数以确定数据库服务器的默认位置，以便创建以下文件：



- 联机重做日志文件
- 控制文件

可以指定文件系统目录的名称，以便为上述操作系统文件的创建提供默认的位置。最多可以指定 5 个多路复用的位置。

仅仅对于联机重做日志文件和控制文件来讲，该参数可以重写在 DB_CREATE_FILE_DEST 初始化参数中为它们指定的任何默认位置。如果没有指定 DB_CREATE_FILE_DEST 参数，而确定了 DB_CREATE_ONLINE_LOG_DEST_n 参数，那么只有联机重做日志文件和控制文件可以被创建为 Oracle 管理的文件。

建议至少指定两个参数。例如：

```
DB_CREATE_ONLINE_LOG_DEST_1 = '/u02/oradata/payroll'  
DB_CREATE_ONLINE_LOG_DEST_2 = '/u03/oradata/payroll'
```

可以允许多路复用，如果其中的一个目标失败，这种方法将为联机重做日志文件和控制文件提供更大的容错性。

3.3 创建 Oracle 管理的文件

当创建操作中没有文件指定时，如果遇到下面的一种情况，Oracle 将创建适当的 Oracle 管理文件：

- 在初始化参数文件中已经包含初始化参数 DB_CREATE_FILE_DEST 或/和 DB_CREATE_ONLINE_LOG_DEST_n。
- 已经发出 ALTER SYSTEM 或 ALTER SESSION 语句来动态设置 DB_CREATE_FILE_DEST 或/和 DB_CREATE_ONLINE_LOG_DEST_n。

如果创建 Oracle 管理文件的语句发现错误或者因为某些方面的失败而不能完成操作，那么该语句创建的任何一个 Oracle 管理文件都将作为错误或失败的一部分自动被删除掉。但是，因为伴随文件系统和存储子系统可能发生的大量的潜在错误，可能需要使用操作系统命令手动删除这些文件。当创建 Oracle 管理文件时，它的文件名将被写入报警文件。如果需要手动删除该文件时，可以利用这条信息找到它。

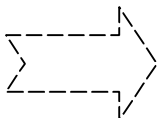
本节将讨论以下内容：

- 如何命名 Oracle 管理文件
- 在数据库创建期间创建 Oracle 管理文件
- 为表空间创建数据文件
- 为临时表空间创建临时文件
- 创建控制文件
- 创建联机重做日志文件

3.3.1 如何命名 Oracle 管理文件

Oracle 管理文件的文件名遵循 Oracle 柔性结构 (OFA, Oracle Flexible Architecture) 标准的文件命名规则。命名的名称将满足以下要求：

- 更容易把数据库文件和所有其他文件区别开。



- 同样也能识别出控制文件、联机重做日志文件和数据文件。
- 能够清晰地指出数据文件和表空间的联系。

两个 Oracle 管理文件不可能有相同的名称。用于创建 Oracle 管理文件的名称是由三部分构造的。

- 默认的文件系统目录位置
- 基于文件类型选择的特定端口的文件名称模板
- 由 Oracle 数据库服务器或操作系统创建的惟一的字符串。这就保证了文件的创建不会损害现有的文件，也不会将该文件误认为其他文件。

举例如下，Oracle 管理文件的文件名在 Solaris 上的格式为：

文件类型	格式	举例说明
数据文件	ora_%t_%u.dbf	/u01/oradata/payroll/ora_tbs1_2ixfh90q.dbf
临时文件	ora_%t_%u.tmp	/u01/oradata/payroll/ora_temp1_6dygh80r.tmp
重做日志文件	ora_%g_%u.log	/u01/oradata/payroll/ora_1_wo94n2xi.log
控制文件	ora_%u.ctl	/u01/oradata/payroll/ora_cmr_7t30p.ctl

其中：

- %t 是表空间的名称。其中，最多可以使用 8 个字符。如果 8 个字符还使得名称太长，那么名称将进行截取。把表空间名称放在惟一的字符串前意味着表空间的所有数据文件将按字母顺序进行排列。
- %u 是一个包含 8 个字符的字符串，它确保惟一性。
- %g 是联机重做日志文件的组号

在其他平台上，名称也类似这样，都要服从所在平台的命名规则的约束。

3.3.2 在数据库创建期间创建 Oracle 管理文件

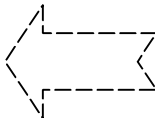
本节主要讨论当使用 Oracle 管理的文件时，为了创建数据库的结构而采用的 CREATE DATABASE 语句的行为。

3.3.2.1 在数据库创建期间指定控制文件

在数据库创建期间，控制文件创建在由 CONTROL_FILES 初始化参数指定的文件中。如果没有设置 CONTROL_FILES 参数，而至少设置了一个创建 Oracle 管理文件所需的初始化参数，那么将在默认的控制文件目标位置创建 Oracle 管理的控制文件。按优先权的次序，定义默认的目标位置如下：

- 如果指定了初始化参数 DB_CREATE_ONLINE_LOG_DEST_n，那么将在指定的每个目录中创建 Oracle 管理的控制文件的副本。第一个目录中的文件为主控制文件。
- 如果指定了初始化参数 DB_CREATE_FILE_DEST，而没有指定初始化参数 DB_CREATE_ONLINE_LOG_DEST_n，那么将在指定的目录中创建 Oracle 管理的控制文件。

如果没有设置 CONTROL_FILES 参数，也没有设置上述任何一个初始化参数，那么 Oracle 将根据操作系统的不同采取相应的默认行为。在操作系统相应的默认位置中，至少创建一个控制文件的副本。这种情形下创建的任何一个控制文件的副本都不是 Oracle 管理



文件，而且必须往初始化参数文件中添加 CONTROL_FILES 初始化参数。

如果 Oracle 创建了 Oracle 管理的控制文件，并且如果存在一个服务器参数文件，那么 Oracle 将在服务器参数文件中创建一个 CONTROL_FILES 初始化参数入口。如果没有服务器参数文件，则必须手动创建一个 CONTROL_FILES 初始化参数入口并把它放在文本初始化参数文件中。

3.3.2.2 在数据库创建期间指定联机重做日志文件

在 CREATE DATABASE 语句中不要求有 LOGFILE 子句，而且，将它省略掉可以提供更简单的方式来创建 Oracle 管理的联机重做日志文件。如果省略掉 LOGFILE 子句，那么将在默认的联机重做日志文件目标位置创建联机重做日志文件。按优先权的次序，默认的目标位置如下：

- 如果指定了初始化参数 DB_CREATE_ONLINE_LOG_DEST_*n*，那么 Oracle 将在指定的每个目录中创建两个联机重做文件。更特别的是，在指定的每个目录中，Oracle 将创建两个联机重做组，其中包含相应的成员。这些联机重做日志文件都是 Oracle 管理的文件。
- 如果指定了初始化参数 DB_CREATE_FILE_DEST，而没有指定初始化参数 DB_CREATE_ONLINE_LOG_DEST_*n*，那么将在指定的目录中创建两个联机重做日志文件（两个组，每个组一个成员）。这些联机重做日志文件都是 Oracle 管理的文件。
- 如果省略了 LOGFILE 子句，并且没有指定上述任何一个初始化参数，那么在操作系统相应的默认位置上将创建两个联机重做日志文件。这种情形下创建的任何一个是联机重做日志文件都不是 Oracle 管理的文件。

Oracle 管理的联机重做日志文件的默认大小是 100MB。

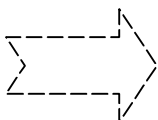
也可以通过包含 LOGFILE 子句但是省略文件名的方式，创建 Oracle 管理的联机重做日志文件，并且重写默认的属性。如上所述创建联机重做日志文件，稍有不同的是：如果 CREATE DATABASE 语句的 LOGFILE 子句中没有提供文件名，而且没有提供创建 Oracle 管理文件所需的初始化参数，那么执行 CREATE DATABASE 语句将会失败。

3.3.2.3 在数据库创建期间指定 SYSTEM 表空间数据文件

在 CREATE DATABASE 语句中不要求有 DATAFILE 子句，而且，将它省略掉可以为 SYSTEM 表空间提供更简单的方式来创建 Oracle 管理的数据文件。如果省略掉 DATAFILE 子句，将会发生下面的一种情况：

- 如果设置了 DB_CREATE_FILE_DEST，那么将在 DB_CREATE_FILE_DEST 指定的目录中为 SYSTEM 表空间创建 Oracle 管理的数据文件。
- 如果没有设置 DB_CREATE_FILE_DEST，那么 Oracle 将会创建一个 SYSTEM 表空间数据文件，该文件的名称和大小依据操作系统而定。在这种方式下创建的 SYSTEM 表空间数据文件都不是 Oracle 管理的数据文件。

Oracle 管理的数据文件的默认大小为 100MB，并且该文件是自动可扩展的，没有最大



尺寸方面的限制。

通过包含 DATAFILE 子句但是省略文件名的方式，也可以为 SYSTEM 表空间创建 Oracle 管理的数据文件，并且重写默认的属性。如果没有提供文件名，但是设置了 DB_CREATE_FILE_DEST 参数，那么将在 DB_CREATE_FILE_DEST 指定的目录中为 SYSTEM 表空间创建 Oracle 管理的数据文件。如果既没有提供文件名，也没有设置 DB_CREATE_FILE_DEST 参数，那么执行 CREATE DATABASE 语句将会失败。

3.3.2.4 在数据库创建期间指定撤消表空间数据文件

UNDO TABLESPACE 子句中的 DATAFILE 次子句是可选项，并且在文件指定中不要有文件名。如果没有提供文件名，但是设置了 DB_CREATE_FILE_DEST 参数，那么将在 DB_CREATE_FILE_DEST 指定的目录中创建 Oracle 管理的数据文件。如果没有设置 DB_CREATE_FILE_DEST 参数，那么执行这条语句将会出现语法错误，进而导致执行失败。

在 CREATE DATABASE 语句中 UNDO TABLESPACE 子句本身就是可选项。如果没有指定这一项，并且采用的是自动撤消管理模式，那么将创建一个名为 SYS_UNDOTBS 的表空间，并且分配一个 10MB 的数据文件，该文件可以自动扩展大小，如下所示：

- 如果设置了 DB_CREATE_FILE_DEST，那么 Oracle 管理的数据文件将创建在指定的目录中。
- 如果没有设置 DB_CREATE_FILE_DEST，那么数据文件将采用操作系统指定的位置。

3.3.2.5 在数据库创建期间指定默认的临时表空间的临时文件

DEFAULT TEMPORARY TABLESPACE 子句中的 TEMPFILE 次子句是可选项，并且在文件指定中不要求有文件名。如果没有提供文件名，但是设置了 DB_CREATE_FILE_DEST 参数，那么将在 DB_CREATE_FILE_DEST 指定的目录中创建 Oracle 管理的临时文件。如果没有设置 DB_CREATE_FILE_DEST 参数，那么执行这条语句将会出现语法错误，进而导致执行失败。

DEFAULT TEMPORARY TABLESPACE 子句本身就是可选项。如果没有指定这一项，那么将不会创建默认的临时表空间。

Oracle 管理的临时文件的默认大小是 100MB，并且该文件是自动可扩展的，没有最大尺寸方面的限制。

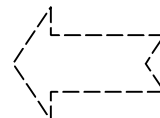
3.3.2.6 使用 Oracle 管理文件的 CREATE DATABASE 语句的例子

这一节介绍使用 Oracle 管理文件特性时的 CREATE DATABASE 语句的例子。

(1) CREATE DATABASE: 例 1

在初始化参数文件中包含以下内容：

```
DB_CREATE_FILE_DEST = '/u01/oradata/sample'
DB_CREATE_ONLINE_LOG_DEST_1 = '/u02/oradata/sample'
DB_CREATE_ONLINE_LOG_DEST_2 = '/u03/oradata/sample'
CREATE DATABASE 语句为：
```



```
SQL> CREATE DATABASE sample;
```

这个例子创建的数据库包含以下 Oracle 管理文件：

- 目录 “/u01/oradata/sample” 中的 SYSTEM 表空间数据文件，该文件是 100MB，并且具有自动可扩展大小的特性，没有最大尺寸限制。
- 两个联机日志组，每个组有两个成员，每个组员是 100MB，分别在 “/u02/oradata/sample” 和 “/u03/oradata/sample” 目录中。
- 如果自动撤消管理模式有效，那么在 “/u01/oradata/sample2” 目录中的撤消表空间数据文件的大小是 10MB，并且该文件具有自动可扩展大小的特性，没有最大尺寸限制。还创建有一个名为 SYS_UNDOTBS 的撤消表空间。
- 如果没有指定 CONTROL_FILES 初始化参数，那么有两个控制文件，分别在 “/u02/oradata/sample” 目录和 “/u03/oradata/sample” 目录中。“/u02/oradata/sample” 目录中的控制文件是主控制文件。

(2) CREATE DATABASE: 例 2

在这个例子中，做以下假设：

- 在初始化参数文件中不指定初始化参数 DB_CREATE_ONLINE_LOG_DEST_n。
- 在初始化参数文件中没有指定初始化参数 CONTROL_FILES。
- 自动撤消管理模式有效。

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u01/oradata/sample2';
```

```
SQL> CREATE DATABASE sample2;
```

这个例子创建的数据库包含以下 Oracle 管理文件：

- 定义在 “/u01/oradata/sample2” 目录中的 SYSTEM 表空间数据文件，该文件是 100MB。
- 两个 100MB 的联机重做日志文件，都在 “/u02/oradata/sample2” 目录中。它们不能多路复用。
- 定义在 “/u01/oradata/sample2” 目录中的撤消表空间数据文件，它的大小是 10MB，并且该文件具有自动可扩展大小的特性，没有最大尺寸限制。还创建有一个名为 SYS_UNDOTBS 的撤消表空间。
- 控制文件在目录 “/u02/oradata/sample” 中。

不建议这么配置数据库，这只适用于最低端的数据库或者简单的测试数据库。为了更好的保护这种数据库，以防失败，就需要创建至少一个或更多的控制文件，并且应该能够多路复用联机重做日志。

(3) CREATE DATABASE: 例 3

在初始化参数文件中包含以下内容：

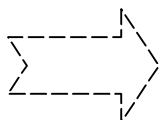
```
DB_CREATE_FILE_DEST = '/u01/oradata/sample3'
```

```
DB_CREATE_ONLINE_LOG_DEST_1 = '/u02/oradata/sample3'
```

```
DB_CREATE_ONLINE_LOG_DEST_2 = '/u03/oradata/sample3'
```

CREATE DATABASE 语句为：

```
SQL> CREATE DATABASE sample3 DATAFILE SIZE 400M
```




```
2> DEFAULT TEMPORARY TABLESPACE dflt_ts TEMPFILE SIZE 10M
3> UNDO TABLESPACE undo_ts DATAFILE SIZE 10M;
```

这个例子中，指定了默认临时表空间和撤消表空间的 Oracle 管理文件的大小。创建的数据库有以下特征：

这个例子创建的数据库包含以下 Oracle 管理文件：

- 在 “/u01/oradata/sample3” 目录中的 SYSTEM 表空间数据文件，该文件是 400MB。
- 两个联机日志组，每个组有两个成员，每个组员是 100MB，分别在 “/u02/oradata/sample3” 和 “/u03/oradata/sample3” 目录中。
- 在 “/u01/oradata/sample3” 目录中，为名为 dflt_ts 的默认临时表空间，创建一个 10MB 的临时文件。
- 在 “/u01/oradata/sample3” 目录中，为名为 undo_ts 的撤消表空间，创建一个 10MB 的数据文件。
- 如果没有指定 CONTROL_FILES 初始化参数，那么有两个控制文件，分别在 “/u02/oradata/sample3” 目录和 “/u03/oradata/sample3” 目录中。
“/u02/oradata/sample3” 目录中的控制文件是主控制文件。

3.3.3 为表空间创建数据文件

采用以下语句可以创建数据文件，在这一节中将逐个加以介绍：

- CREATE TABLESPACE
- CREATE UNDO TABLESPACE
- ALTER TABLESPACE ... ADD DATAFILE

创建表空间时，不管是普通的表空间还是撤消表空间，都可以使用 DATAFILE 子句来创建。如果在语句中包含了 DATAFILE 子句，那么文件名是可选项。如果没有提供 DATAFILE 子句或文件名，那么将应用如下准则：

- 如果指定了初始化参数 DB_CREATE_FILE_DEST，那么将在参数指定的位置上创建 Oracle 管理的数据文件。
- 如果没有指定初始化参数 DB_CREATE_FILE_DEST，那么执行创建数据文件的语句将会导致失败。

如果用 ALTER TABLESPACE ... ADD DATAFILE 语句在表空间中添加了数据文件，那么文件名将是可选项。如果没有指定文件名，将采用前面段落中介绍的准则。

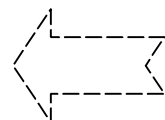
默认情况下，为普通表空间创建的 Oracle 管理的数据文件大小为 100MB，并且该文件的大小是可自动扩展的，而且没有最大尺寸的限制。

3.3.3.1 CREATE TABLESPACE 的例子

下面是一些创建具有 Oracle 管理文件的表空间的例子。

(1) CREATE TABLESPACE: 例 1

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u01/oradata/sample1';
SQL> CREATE TABLESPACE tbs_1;
```



这个例子为数据文件的创建设置了默认位置“/u01/oradata/sample”，然后创建了表空间“tbs_1”，其中有一个数据文件。数据文件的大小为 100MB，并且其大小是可自动扩展的，而且没有最大尺寸的限制。

(2) CREATE TABLESPACE: 例 2

在初始化参数文件中包含以下内容：

```
DB_CREATE_FILE_DEST = '/u01/oradata/sample2'
```

CREATE TABLESPACE 语句为：

```
SQL> CREATE TABLESPACE tbs_2 DATAFILE SIZE 400M AUTOEXTEND OFF;
```

这个例子在“/u01/oradata/sample2”目录中创建了一个名为“tbs_2”的表空间，其中包含一个数据文件，该文件的大小是不可以自动扩展的，大小为 400MB。

(3) CREATE TABLESPACE: 例 3

在初始化参数文件中包含以下内容：

```
DB_CREATE_FILE_DEST = '/u01/oradata/sample3'
```

CREATE TABLESPACE 语句为：

```
SQL> CREATE TABLESPACE tbs_3 DATAFILE AUTOEXTEND ON MAXSIZE 800M;
```

这个例子在“/u01/oradata/sample3”目录中创建了一个名为“tbs_3”的表空间，其中包含一个可以自动扩展大小的数据文件，该文件的起始大小为 100MB，最大尺寸为 800MB。

(4) CREATE TABLESPACE: 例 4

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u01/oradata/sample4';
```

```
SQL> CREATE TABLESPACE tbs_4 DATAFILE SIZE 200M, SIZE 200M;
```

这个例子为数据文件的创建设置了默认位置“/u01/oradata/sample4”，然后在这个目录中创建了一个名为“tbs_4”的表空间，其中包含两个数据文件。这些数据文件的大小可以自动扩展，起始大小为 200MB，没有上限。

3.3.3.2 CREATE UNDO TABLESPACE 的例子

下面的例子中，在“/u01/oradata/sample”目录中创建了一个名为“undotbs_1”的撤销表空间。其中的数据文件的大小为 100MB，并且大小可以自动扩展，没有上限。

初始化参数文件中包含以下内容：

```
DB_CREATE_FILE_DEST = '/u01/oradata/sample'
```

CREATE UNDO TABLESPACE 语句为：

```
SQL> CREATE UNDO TABLESPACE undotbs_1;
```

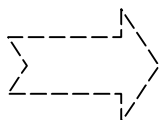
3.3.3.3 ALTER TABLESPACE 的例子

下面的例子中，在表空间“tbs_1”中添加了一个 Oracle 管理的可自动扩展的数据文件，该文件起始大小为 100MB，最大值为 800MB。

初始化参数文件中包含以下内容：

```
DB_CREATE_FILE_DEST = '/u01/oradata/sample'
```

ALTER TABLESPACE 语句为：



```
SQL> ALTER TABLESPACE tbs_1 ADD DATAFILE AUTOEXTEND ON MAXSIZE 800M;
```

3.3.4 为临时表空间创建临时文件

可以采用下面的语句创建临时文件，将在本节中逐个加以讨论：

- CREATE TEMPORARY TABLESPACE
- ALTER TABLESPACE ... ADD TEMPFILE

创建临时表空间时，TEMPFILE 子句是可选项。如果在语句中包含了 TEMPFILE 子句，那么文件名将是可选项。如果没有提供 TEMPFILE 子句或文件名，那么将遵循以下的准则：

- 如果指定了初始化参数 DB_CREATE_FILE_DEST，那么将在参数指定的位置上创建 Oracle 管理的临时文件。
- 如果没有指定初始化参数 DB_CREATE_FILE_DEST，那么执行创建临时文件的语句将会导致失败。

如果用 ALTER TABLESPACE ... ADD TEMPFILE 语句在表空间中添加了临时文件，那么文件名将是可选项。如果没有指定文件名，将遵循前面段落中介绍的准则。

默认情况下，为普通表空间创建的 Oracle 管理的临时文件大小为 100MB，并且该文件的大小是可自动扩展的，没有上限。

3.3.4.1 CREATE TEMPORARY TABLESPACE 的例子

在下面的例子中，为临时文件的创建设置了默认位置 “/u01/oradata/sample”，然后创建了一个名为 “temptbs_1” 的表空间，其中包含的临时文件的大小为 100MB，并且大小可以自动扩展，没有上限。

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u01/oradata/sample';
SQL> CREATE TEMPORARY TABLESPACE temptbs_1;
```

3.3.4.2 ALTER TABLESPACE ... ADD TEMPFILE 的例子

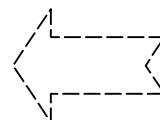
在下面的例子中，为临时文件的创建设置了默认位置 “/u03/oradata/sample”，然后在名为 “temptbs_1” 的表空间中添加了一个临时文件，该临时文件的起始大小为 100MB，并且大小可以自动扩展，没有上限。

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u03/oradata/sample';
SQL> ALTER TABLESPACE TBS_1 ADD TEMPFILE;
```

3.3.5 创建控制文件

当发布 CREATE CONTROLFILE 语句时，将会在 CONTROL_FILES 初始化参数指定的文件中创建一个控制文件（如果指定了 REUSE，也可以重新使用某个控制文件）。如果没有设置 CONTROL_FILES 参数，那么将在默认的控制文件目标位置创建控制文件。为了取得优先权，按如下方式定义默认的目标位置：

- 如果指定了初始化参数 DB_CREATE_ONLINE_LOG_DEST_n，那么 Oracle 将在指定的每个目录中创建 Oracle 管理的控制文件副本。在第一个目录中的文件是主要的控制文件。



- 如果指定了初始化参数 DB_CREATE_FILE_DEST，而没有指定初始化参数 DB_CREATE_ONLINE_LOG_DEST_n，那么将在指定的目录中创建 Oracle 管理的控制文件。
- 如果既没有指定初始化参数 DB_CREATE_ONLINE_LOG_DEST_n，也没有指定初始化参数 DB_CREATE_FILE_DEST，那么在操作系统特定的默认位置上将创建控制文件。该控制文件不是 Oracle 管理的文件。

如果 Oracle 创建了一个 Oracle 管理的控制文件，并且还有一个服务器参数文件，那么 Oracle 将为这个服务器参数文件创建 CONTROL_FILES 初始化参数。如果没有服务器参数文件，就必须手动创建一个 CONTROL_FILES 初始化参数，并且将它包含在初始化参数文件中。

如果数据库中的数据文件是 Oracle 管理的文件，那么在语句的 DATAFILE 子句中就必须提供 Oracle 为这些文件生成的文件名。

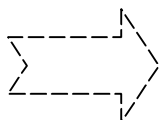
如果联机重做日志文件是 Oracle 管理的文件，那么将由[NO]RESETLOGS 关键字决定 LOGFILE 子句可以提供的内容：

- NORESETLOGS: 在 LOGFILE 子句中必须提供 Oracle 为 Oracle 管理的联机重做日志文件生成的文件名。
- RESETLOGS: 可以使用 CREATE DATABASE 语句提供联机重做日志文件的名称。下面的章节将包含使用 CREATE CONTROLFILE 语句创建 Oracle 管理文件的例子。

3.3.5.1 使用 NORESETLOGS 关键字的 CREATE CONTROLFILE 语句的例子

下面的 CREATE CONTROLFILE 语句由 ALTER DATABASE BACKUP CONTROLFILE TO TRACE 语句生成，将为数据库创建 Oracle 管理的数据文件和联机重做日志文件：

```
CREATE CONTROLFILE
DATABASE sample
LOGFILE GROUP 1 ('/u01/oradata/sample/ora_1_o220rtt9.log',
'/u02/oradata/sample/ora_1_v2o0b2i3.log') SIZE 100M,
GROUP 2 ('/u01/oradata/sample/ora_2_p22056iw.log',
'/u02/oradata/sample/ora_2_p02rcyg3.log') SIZE 100M
NORESETLOGS
DATAFILE '/u01/oradata/sample/ora_system_xu34ybm2.dbf' SIZE 100M
MAXLOGFILES 5
MAXLOGHISTORY 100
MAXDATAFILES 10
MAXINSTANCES 2
ARCHIVELOG;
```



3.3.5.2 使用 RESETLOGS 关键字的 CREATE CONTROLFILE 语句的例子

下面的例子是有 RESETLOGS 选项的 CREATE CONTROLFILE 语句。必须设置 DB_CREATE_ONLINE_LOG_DEST_*n* 或 DB_CREATE_FILE_DEST 初始化参数。

```
CREATE CONTROLFILE
DATABASE sample
RESETLOGS
DATAFILE '/u01/oradata/sample/ora_system_aawbmz51.dbf' SIZE 100M
MAXLOGFILES 5
MAXLOGHISTORY 100
MAXDATAFILES 10
MAXINSTANCES 2
ARCHIVELOG;
```

之后，必须发布 ALTER DATABASE OPEN RESETLOGS 语句以重新创建联机重做日志文件。这个内容将在下一节中讨论。如果前面的日志文件是 Oracle 管理的文件，那么将不会被删除。

3.3.6 创建联机重做日志文件

在数据库创建时期，会创建联机重做日志文件。也可以在发布以下语句时创建联机重做日志文件。

- ALTER DATABASE ADD LOGFILE
- ALTER DATABASE OPEN RESETLOGS

3.3.6.1 使用 ALTER DATABASE ADD LOGFILE 语句

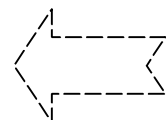
该语句允许往当前联机重做日志中添加新的组。如果正在使用 Oracle 管理的文件，则 ADD LOGFILE 子句中的文件名是可选项。如果没有提供文件名，那么将在默认的日志文件目标位置创建一个重做日志文件。为了取得优先权，按如下方式定义默认的目标位置：

- 如果指定了初始化参数 DB_CREATE_ONLINE_LOG_DEST_*n*，那么 Oracle 将在该参数指定的每个目录中创建 Oracle 管理的日志文件成员（直到创建 MAXLOGMEMBERS 成员）。
- 如果指定了初始化参数 DB_CREATE_FILE_DEST，而没有指定初始化参数 DB_CREATE_ONLINE_LOG_DEST_*n*，那么将在参数指定的目录中创建 Oracle 管理的日志文件成员。

如果没有提供文件名。也没有提供创建 Oracle 管理文件所需的上述任何一个初始化参数，那么该语句将返回一条错误信息。

Oracle 管理的日志文件的默认大小为 100MB。

通过指定完整的文件名可以继续添加或取消联机重做日志文件成员。



3.3.6.2 添加新的联机重做日志文件的例子

下面的例子在“/u01/oradata/sample”目录中创建一个日志文件的成员，在“/u02/oradata/sample”目录中创建另一个日志文件的成员。日志文件的大小为 100MB。

在初始化参数文件中包含以下内容：

```
DB_CREATE_ONLINE_LOG_DEST_1 = '/u01/oradata/sample'
```

```
DB_CREATE_ONLINE_LOG_DEST_2 = '/u02/oradata/sample'
```

ALTER DATABASE 语句为：

```
SQL> ALTER DATABASE ADD LOGFILE;
```

3.3.6.3 使用 ALTER DATABASE OPEN RESETLOGS 语句

如果以前使用 RESETLOGS 创建了一个控制文件，并且没有指定文件名，或者指定了不存在的文件名，那么，当发出 ALTER DATABASE OPEN RESETLOGS 语句时，Oracle 将创建联机重做日志文件。当没有为控制文件指定任何内容时，决定存储重做日志文件的目录的准则与“在数据库创建期间指定联机重做日志文件”一节中讨论的一样。

3.4 Oracle 管理文件的行为

不管在哪个 SQL 语句中提供的用于标识现有文件的文件名，都将接受 Oracle 管理文件的文件名。这些文件名，像其他文件名一样，存储在控制文件中；如果使用恢复管理器（RMAN）来备份和恢复文件，那么文件名也将存储在 RMAN 目录中。在所有常用的固定和动态性能视图中，这些文件都是可见的，这些视图也可以用来镜像数据文件和临时文件（例如，V\$DATAFILE 或 DBA_DATA_FILES 视图）。

使用 Oracle 生成文件名的语句，举例如下：

```
SQL> ALTER DATABASE RENAME FILE 'ora_tbs01_ziw3bopb.dbf'
```

```
2> TO 'tbs0101.dbf';
```

```
SQL> ALTER DATABASE DROP LOGFILE 'ora_1_wo94n2xi.log';
```

```
SQL> ALTER TABLE emp ALLOCATE EXTENT ( DATAFILE 'ora_tbs1_2ixfh90q.dbf' );
```

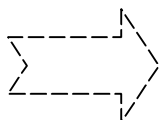
在与非 Oracle 管理文件进行协调时，可以备份和还原 Oracle 管理的数据文件、临时文件和控制文件。使用 Oracle 生成文件名并不影响逻辑备份文件的使用，比如导出文件。对于表空间及时恢复（TSPITR）和可传输的表空间导出文件，这一点相当重要。

也存在 Oracle 管理的文件行为不同的情况。这将在下一节中进行讨论。

3.4.1 取消数据文件和临时文件

不像非 Oracle 管理的文件，当取消 Oracle 管理的数据文件或临时文件时，将从控制文件中删除相应的文件名，并且这些文件将自动从文件系统中删除。当取消这些文件时，删除 Oracle 管理文件的语句如下所示：

- DROP TABLESPACE
- ALTER DATABASE TEMPFILE ... DROP



3.4.2 取消联机重做日志文件

当取消 Oracle 管理的联机重做日志文件时, 该文件相应的 Oracle 管理文件也将被删除。需要指定要取消的组或成员。下面这两个语句用于取消和删除联机重做日志文件:

- ALTER DATABASE DROP LOGFILE
- ALTER DATABASE DROP LOGFILE MEMBER

3.4.3 重新命名文件

下面的语句用于重新命名文件:

- ALTER DATABASE RENAME FILE
- ALTER TABLESPACE ...RENAME DATAFILE

这些语句并不能真正重新命名操作系统上的文件, 但是可以改变控制文件中相应的名称。如果旧的文件是一个 Oracle 管理的文件并且也确实存在, 那么该文件将被删除。当发布这个语句时, 必须使用相应操作系统的文件命名约定指定每个文件名。

3.4.4 管理备用的数据库

备用数据库中的数据文件、控制文件和联机重做日志文件可以是 Oracle 管理的文件。这与在主数据库上是否使用 Oracle 管理的文件无关。

当备用数据库的恢复操作遇到数据文件创建时的重做操作时, 如果数据文件是 Oracle 管理的文件, 那么, 恢复进程将在本地的默认文件系统位置中创建一个空的文件。这样就允许立即应用新文件的重做内容, 而不需要任何人工干预。

当备用数据库的恢复操作遇到需要重做表空间的删除操作时, 将在本地文件系统中删除所有 Oracle 管理的数据文件。注意, 这与在主数据库中发出的 INCLUDING DATAFILES 选项无关。

3.5 使用 Oracle 管理文件的案例

这一节将通过介绍使用 Oracle 管理文件的案例来进一步说明 Oracle 管理文件的用法。

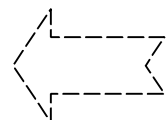
3.5.1 案例 1: 使用多路复用的联机重做日志创建和管理数据库

在这种情况下, DBA 创建的数据库中, 数据文件和联机重做日志文件分别是在单独的目录中创建的。其中, 联机重做日志文件和控制文件是多路复用的。数据库使用了一个撤销表空间, 并且有一个默认的临时表空间。下面是创建和维护这个数据库所涉及的任务。

3.5.1.1 设置初始化参数

在数据库创建之前, DBA 在初始化参数文件中包含了 3 个默认的正常文件创建默认值。还指定了自动撤销管理模式。

```
DB_CREATE_FILE_DEST = '/u01/oradata/sample'
DB_CREATE_ONLINE_LOG_DEST_1 = '/u02/oradata/sample'
DB_CREATE_ONLINE_LOG_DEST_2 = '/u03/oradata/sample'
```



```
UNDO_MANAGEMENT = AUTO
```

其中, DB_CREATE_FILE_DEST 参数为数据文件和临时文件设置了默认的文件系统目录。

DB_CREATE_ONLINE_LOG_DEST_1 和 DB_CREATE_ONLINE_LOG_DEST_2 为联机重做日志文件和控制文件的创建设置了默认的文件系统目录。每个联机重做日志文件和控制文件都可以多路复用这两个目录。

3.5.1.2 创建数据库

一旦设置了初始化参数, 就可以创建数据库:

```
SQL> CREATE DATABASE sample
2> DEFAULT TEMPORARY TABLESPACE dflt_tmp;
```

因为语句中没有出现 DATAFILE 子句, 而且设置了初始化参数 DB_CREATE_FILE_DEST, 所以将在默认的文件系统 (在这个例子中是 “/u01/oradata/sample”) 中创建 SYSTEM 表空间数据文件。文件名是由 Oracle 生成的唯一的文件名称。该文件的大小可以自动扩展, 起始大小为 100MB, 并且没有最大值的限制。该文件是一个 Oracle 管理的文件。

因为语句中也没有出现 LOGFILE 子句, 所以将创建两个联机重做日志组。每个日志组有两个成员, 一个成员位于 DB_CREATE_ONLINE_LOG_DEST_1 指定的位置, 另一个成员位于 DB_CREATE_ONLINE_LOG_DEST_2 指定的位置。文件名是由 Oracle 生成的唯一的文件名称。创建的日志文件的大小为 100MB。这些日志文件成员都是 Oracle 管理的文件。

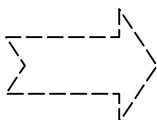
与上述原因类似, 因为语句中没有出现 CONTROL_FILES 初始化参数, 而且指定了两个 DB_CREATE_ONLINE_LOG_DEST_n 初始化参数, 所以将创建两个控制文件。位于 DB_CREATE_ONLINE_LOG_DEST_1 参数指定位置上的控制文件是主要的控制文件; 位于 DB_CREATE_ONLINE_LOG_DEST_2 参数指定位置上的控制文件是多路复用的备份文件。文件名是由 Oracle 生成的唯一的文件名称。这些文件都是 Oracle 管理的文件。如果有服务器参数文件, 还会出现 CONTROL_FILES 初始化参数。

指定了自动撤消管理模式, 但是因为指定了撤消表空间, 并且设置了初始化参数 DB_CREATE_FILE_DEST, 所以将在该参数指定的目录中创建名为 SYS_UNDOTBS 的默认的撤消表空间。其中, 数据文件的大小为 10MB, 并且大小可以自动扩展, 同时, 它还是一个 Oracle 管理的文件。

最后, 指定了名为 dflt_tmp 的默认临时表空间。因为初始化参数文件中包含了初始化参数 DB_CREATE_FILE_DEST, 所以将在该参数指定的目录中为 dflt_tmp 临时表空间创建临时文件。临时文件的大小为 100MB, 并且大小可以自动扩展, 而且没有最大尺寸的限制。同时, 它还是一个 Oracle 管理的文件。

最后合成的文件树如下所示, 用生成的文件名表示文件:

```
/u01
/oradata
```



```

/sample
/ora_system_cmr7t30p.dbf
/ora_sys_undo_2ixfh90q.dbf
/ora_dflt_tmp_157se6ff.tmp
/u02
/oradata
/sample
/ora_1_0orrm31z.log
/ora_2_2xyz16am.log
/ora_cmr7t30p.ctl
/u03
/oradata
/sample
/ora_1_ixfvm8w9.log
/ora_2_q89tmp28.log
/ora_xlsr8t36.ctl

```

当选择普通视图时，可以从中看到内部生成的文件名。例如：

```
SQL> SELECT NAME FROM V$DATAFILE;
```

```
NAME
```

```

-----
/u01/oradata/sample/ora_system_cmr7t30p.dbf
/u01/oradata/sample/ora_sys_undo_2ixfh90q.dbf
2 rows selected

```

当创建文件时，也将把文件名显示在报警文件中。

3.5.1.3 管理控制文件

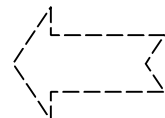
当生成数据库时，创建控制文件，而且 `CONTROL_FILES` 初始化参数也已经添加到参数文件中。如果需要，DBA 可以重建控制文件，或使用 `CREATE CONTROLFILE` 语句为数据库构建一个新的控制文件。

在 `DATAFILE` 和 `LOGFILE` 两个子句中必须使用正确的 Oracle 管理的文件名。`ALTER DATABASE BACKUP CONTROLFILE TO TRACE` 语句将生成带有正确的文件名的脚本。也可以通过选择 `V$DATAFILE`、`V$TEMPFILE` 和 `V$LOGFILE` 视图找到文件名。下面的例子中为示例数据库重建了一个控制文件：

```

SQL> CREATE CONTROLFILE REUSE
2> DATABASE sample
3> LOGFILE GROUP 1 ('/u02/oradata/sample/ora_1_0orrm31z.log',
4> '/u03/oradata/sample/ora_1_ixfvm8w9.log'),
5> GROUP 2 ('/u02/oradata/sample/ora_2_2xyz16am.log',

```




```
6> '/u03/oradata/sample/ora_2_q89tmp28.log')
7> NORESETLOGS
8> DATAFILE '/u01/oradata/sample/ora_system_cmr7t30p.dbf',
9> '/u01/oradata/sample/ora_sys_undo_2ixfh90q.dbf',
10> '/u01/oradata/sample/ora_dflt_tmp_157se6ff.tmp'
11> MAXLOGFILES 5
12> MAXLOGHISTORY 100
13> MAXDATAFILES 10
14> MAXINSTANCES 2
15> ARCHIVELOG;
```

数据库创建后产生了 CONTROL_FILES 初始化参数, 这个语句创建的控制文件就位于这个初始化参数指定的位置中。REUSE 子句将导致重写任何现有的文件。

3.5.1.4 管理联机重做日志

要创建新的联机重做日志组, DBA 可以使用 ALTER DATABASE ADD LOGFILE 语句。下面的语句中, 添加了一个日志文件, 其中一个成员位于 DB_CREATE_ONLINE_LOG_DEST_1 指定的位置上, 另一个成员位于 DB_CREATE_ONLINE_LOG_DEST_2 指定的位置上。这些文件都是 Oracle 管理的文件。

```
SQL> ALTER DATABASE ADD LOGFILE;
```

可以通过指定完整的文件名继续添加和取消日志文件成员。

可以用 GROUP 子句取消一个日志文件。在下面的例子中, 与 Oracle 管理的日志文件成员相关的操作系统文件将被自动删除。

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

3.5.1.5 管理表空间

在示例数据库中, 为了将来表空间的创建, 所有数据文件的默认存储位置都是由初始化参数 DB_CREATE_FILE_DEST 指定的 (在这个案例中为 “/u01/oradata/sample”)。任何一个没有指定文件名的数据文件都将创建在初始化参数 DB_CREATE_FILE_DEST 指定的文件系统中。例如:

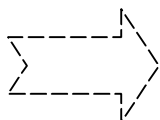
```
SQL> CREATE TABLESPACE tbs_1;
```

前面的语句创建了一个表空间, 位于 “/u01/oradata/sample”。创建的数据文件的起始大小为 100MB, 并且大小可以自动扩展, 而且没有最大尺寸的限制。该数据文件是一个 Oracle 管理的文件。

当表空间被删除时, 表空间相应的 Oracle 管理文件也将自动被删除。下面的语句将删除存储时使用的表空间及其所有相关的 Oracle 管理文件。

```
SQL> DROP TABLESPACE tbs_1;
```

一旦第一个数据文件写满, Oracle 将自动创建一个新的数据文件。通过添加另一个 Oracle 管理的数据文件, 可以给表空间增加更多的空间。下面的语句中, 在初始化参数



DB_CREATE_FILE_DEST 指定的位置上添加了另一个数据文件：

```
SQL> ALTER TABLESPACE tbs_1 ADD DATAFILE;
```

通过改变初始化参数 DB_CREATE_FILE_DEST 的值可以更改默认的文件系统。但不改变任何一个现有的数据文件，而只影响将来的创建工作。使用以下语句可以动态完成更改的任务：

```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST='/u04/oradata/sample';
```

3.5.1.6 存档重做信息

联机重做日志文件的存档不同于 Oracle 管理文件以及非管理文件。存档日志文件的文件系统位置可以使用 LOG_ARCHIVE_DEST_*n* 初始化参数指定。文件名基于 LOG_ARCHIVE_FORMAT 参数指定的内容或它的默认值来形成。

存档的日志文件不是 Oracle 管理的文件。

3.5.1.7 备份、还原和恢复

因为 Oracle 管理文件与标准的操作系统文件兼容，所以可以使用操作系统的工具来备份或还原 Oracle 管理文件。所有现有的备份、还原和恢复数据库的方法同样适用于 Oracle 管理文件。

3.5.2 案例 2：往现有数据库添加 Oracle 管理的文件

在这个例子中，假设现有数据库中没有 Oracle 管理的文件，但是 DBA 希望创建新的包含 Oracle 管理文件的表空间，并把它们放在 “/u03/oradata/sample2”。

3.5.2.1 设置初始化参数

为了允许自动创建数据文件，可以将初始化参数 DB_CREATE_FILE_DEST 设置为创建数据文件所在的文件系统目录中。下面的语句可以动态完成上述任务：

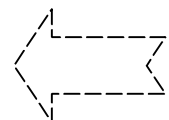
```
SQL> ALTER SYSTEM SET DB_CREATE_FILE_DEST = '/u03/oradata/sample2';
```

3.5.2.2 创建表空间

一旦设置了初始化参数 DB_CREATE_FILE_DEST，就可以忽略 CREATE TABLESPACE 语句中的 DATAFILE 子句。默认情况下，数据文件创建在 DB_CREATE_FILE_DEST 参数指定的位置中。例如：

```
SQL> CREATE TABLESPACE tbs_2;
```

当取消表空间 “tbs_2” 时，也将自动删除它的数据文件。



第 4 章 启动和关闭数据库

本章描述启动和关闭 Oracle 数据库的过程，包含以下内容：

- 启动数据库
- 改变数据库的可用性
- 关闭数据库
- 禁止操作数据库
- 暂停和继续数据库

4.1 启动数据库

启动数据库的同时也就创建了一个数据库的实例，并且可以选择数据库启动的状态。通常，可以通过装载和打开数据库来启动实例，这样任何一个有效的用户都能连接到该实例并执行典型的数据访问操作。但是，还可以选择另外的方法，这一节也将讨论这些内容。

本节包含有关启动数据库实例的以下内容：

- 启动数据库的方法
- 准备启动实例
- 使用 SQL*Plus 启动数据库
- 启动实例：各种方案

4.1.1 启动数据库的方法

启动（和管理）数据库实例的方法有很多，如下所示。

4.1.1.1 使用 SQL*Plus

可以先使用 SQL*Plus 连接到具有管理员权限的 Oracle，然后再发布 STARTUP 命令，从而启动数据库。虽然提出了启动数据库实例的三种方法，但是本书只涉及使用 SQL*Plus 这种方法。

4.1.1.2 使用 Recovery Manager

也可以使用 Recovery Manager (RMAN) 来执行 STARTUP（和 SHUTDOWN）命令。如果是在 RMAN 环境中，最好使用这种方法，而不要调用 SQL*Plus 方法。

4.1.1.3 使用 Oracle Enterprise Manager

可以选择使用 Oracle Enterprise Manager 来管理数据库，包括启动和关闭数据库。Oracle Enterprise Manager 是一个独立的 Oracle 产品，它与图形控制台、代理程序、公用服务以及各种工具一起为管理 Oracle 产品提供了一个集成的和复杂的系统管理平台。它允许用户使用 GUI 界面而不是命令行来执行本书中讨论的函数。

4.1.2 准备启动实例

在使用 SQL*Plus 启动数据库实例之前，必须采取一些预备步骤。

①以没有连接数据库的方式启动 SQL*Plus:

```
SQLPLUS /NOLOG
```

②作为 SYSDBA 连接到 Oracle:

```
CONNECT username/password AS SYSDBA
```

至此，就连接到了 Oracle 并做好准备启动数据库实例。

4.1.3 使用 SQL*Plus 启动数据库

使用 STARTUP 命令来启动数据库实例。要启动实例，Oracle 必须从服务器参数文件或传统的文本初始化参数文件中读取实例配置参数（初始化参数）。

当发布不带 PFILE 子句的 STARTUP 命令时，Oracle 将从平台指定的默认位置上的服务器参数文件（SPFILE）中读取初始化参数。

注意：对于 UNIX，平台为服务器参数文件（或文本初始化参数文件）指定的默认位置（目录）是：

```
$ORACLE_HOME/dbs
```

对于 Windows NT 和 Windows 2000，其位置是：

```
$ORACLE_HOME\database
```

在平台指定的默认位置上，Oracle 通过检查文件名按照如下次序放置初始化参数文件：

①spfile\$ORACLE_SID.ora

②spfile.ora

③init\$ORACLE_SID.ora

通过使用 STARTUP 命令的 PFILE 子句命令 Oracle 从传统的文本初始化参数文件中读取初始化参数。例如：

```
STARTUP PFILE = /u01/oracle/dbs/init.ora
```

此外，可以使用带有非默认服务器参数文件的 PFILE 子句来启动实例，如下所示：

（1）创建一个只包含 SPFILE 参数的单行文本初始化参数文件。参数的值为非默认服务器参数文件的位置。

例如，创建一个文本初始化参数文件/u01/oracle/dbs/spf_init.ora，它只包含以下参数：

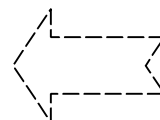
```
SPFILE = /u01/oracle/dbs/test_spfile.ora
```

注意：不能使用文本初始化参数文件中的 IFILE 初始化参数来指向服务器参数文件。在这种情况下，必须使用 SPFILE 初始化参数。

（2）启动指向这个初始化参数文件的实例。

```
STARTUP PFILE = /u01/oracle/dbs/spf_init.ora
```

因为服务器参数文件必须在运行数据库服务器的机器上，所以上面的方法也为客户机启动使用服务器参数文件的数据库提供了一种方式。这样，客户机也就不需要维护客户端的初始化参数文件。当客户机读取包含 SPFILE 参数的初始化参数文件时，它会将该值传递给读取指定的服务器参数文件的服务器。



启动实例可以采取不同的模式：

- 启动没有装载数据库的实例。这种模式不允许访问数据库，并且通常只适用于数据库创建或控制文件的重新创建的情况。
- 启动实例并装载数据库，但是保持数据库关闭状态。这种模式适用于某些 DBA 的动作，但是不允许对数据库的一般的访问动作。
- 启动实例，装载并打开数据库。这种模式可以在不受限制的方式下使用，允许所有用户访问，或者在受限制的方式下使用，只允许数据库管理员访问。

注意：如果是通过一个共享的服务器进程连接到数据库，就不能启动数据库实例。

另外，可以迫使实例启动，或者启动实例并立即开始完整的介质恢复过程。STARTUP 命令可以指定选项获取这些状态模式，在后面的章节中将加以介绍。

4.1.4 启动实例：各种方案

下面的各种方案描述和举例说明了启动实例的不同状态模式。当与 STARTUP 命令的选项一起使用时会有一些限制条件。

注意：如果没有可以利用的控制文件、数据库文件或重做日志文件，那么在启动实例时很可能会遇到一些问题。如果由 CONTROL_FILES 初始化参数指定的一个或多个文件不存在或在装载数据库时不能打开，那么 Oracle 将返回一条警告消息并不能装载数据库。如果数据文件或者重做日志文件，它的一个或多个不存在，或者在打开数据库时不能打开，那么 Oracle 将返回一条警告消息并不能装载数据库。

4.1.4.1 启动实例，装载并打开数据库

普通的数据库操作意味着某个实例已经启动并且数据库也已经装载并打开。这种模式允许任何一个有效的用户连接到数据库并执行典型的数据访问操作。

启动实例，接着从默认的服务器参数文件位置读取初始化参数，然后通过使用 STARTUP 命令来装载和打开数据库（当然，也可以选择由 PFILE 或 SPFILE 子句指定）：

```
STARTUP
```

4.1.4.2 启动实例，不装载数据库

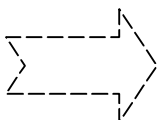
启动实例而不用装载数据库。通常，只在整个数据库创建过程中使用该模式。使用带 NOMOUNT 选项的 STARTUP 命令：

```
STARTUP NOMOUNT
```

4.1.4.3 启动实例并装载数据库

可以启动实例并装载数据库，不用打开数据库，该模式允许执行特定的维护操作。例如，在下面的任务中必须装载数据库而不用打开数据库：

- 重命名数据文件
- 添加、取消或重命名重做日志文件
- 允许和禁止重做日志存档选项
- 执行完整的数据库恢复操作



通过使用带 MOUNT 选项的 STARTUP 命令来启动实例并装载数据库，但是保持数据库关闭状态：

```
STARTUP MOUNT
```

4.1.4.4 在启动过程中限制访问数据库

可以在受限制的模式下启动实例并装载和打开数据库，以便只有管理人员（而不是一般的数据库用户）可以利用数据库。当需要完成以下一种任务时使用这种数据库启动模式：

- 执行数据库数据的导出或导入操作
- 执行数据装载操作（用 SQL*Loader）
- 暂时阻止一般的用户使用数据
- 在某个移植过程和升级操作过程中

通常，所有拥有 CREATE SESSION 系统权限的用户都能连接到一个打开的数据库上。以受限模式打开数据库可以允许拥有 CREATE SESSION 和 RESTRICTED SESSION 两种系统权限的用户访问数据库。只有数据库管理员拥有 RESTRICTED SESSION 系统权限。

通过使用带 RESTRICT 选项的 STARTUP 命令，可以在受限模式下启动实例（还可以装载和打开数据库）：

```
STARTUP RESTRICT
```

之后，使用 ALTER SYSTEM 语句来禁止 RESTRICTED SESSION 特性：

```
ALTER SYSTEM DISABLE RESTRICTED SESSION;
```

如果是在非受限模式下打开的数据库，后来会发现需要限制访问，此时可以使用 ALTER SYSTEM 语句来完成任务，这个内容将在本章的“限制对打开的数据库访问”一节中加以介绍。

4.1.4.5 迫使实例启动

在一些特殊环境下，可能会在启动数据库实例的时候遇到一些问题。一般不要迫使数据库启动，除非是在以下情形：

- 用 SHUTDOWN NORMAL，SHUTDOWN IMMEDIATE，SHUTDOWN TRANSACTIONAL 命令不能关闭当前的实例时。
- 在启动实例的时候遇到一些问题时。

如果产生了上面的一种情形，通常可以使用带 FORCE 选项的 STARTUP 命令来启动一个新的实例，从而解决这个问题：

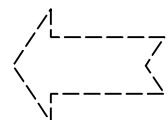
```
STARTUP FORCE
```

如果正在运行某个实例，那么在重新启动实例前 STARTUP FORCE 可以使用 ABORT 模式关闭该实例。

4.1.4.6 启动实例，装载数据库并启动完整的介质恢复过程

如果知道需要介质恢复过程，就可以启动实例，并为其装载数据库以及通过使用带 RECOVER 选项的 STARTUP 命令的方法来自动启动恢复进程：

```
STARTUP OPEN RECOVER
```



如果不需要任何的恢复进程时，在试图执行恢复进程，Oracle 将发布一条错误消息。
在操作系统启动过程中自动启动数据库

一些网站使用进程使得一个或多个 Oracle 实例和数据库在系统启动之后自动启动。执行这项任务的进程在不同的操作系统中是不同的。

4.1.4.7 启动远程实例

如果本地 Oracle 服务器是分布数据库的一部分，那么可以启动一个远程实例和数据库。根据通信协议和操作系统的不同，启动和停止远程实例的进程也会有所不同。

4.2 改变数据库的可用性

可以改变数据库的可用性。这么做的目的也许是因为维护的原因而限制访问或者使数据库为只读。以下这些章节将解释如何改变数据库的可用性：

- 为实例装载数据库
- 打开一个关闭的数据库
- 以只读模式打开数据库
- 限制对打开的数据库访问

4.2.1 为实例装载数据库

当需要执行特定的管理操作时，必须启动数据库并将其装载到实例上，但是数据库必须是关闭的。通过启动实例和装载数据库可以获得这种方案。

要装载一个先前打开的数据库，但是又不打开实例，可以使用带 MOUNT 选项的 SQL 语句 ALTER DATABASE，如下所示：

```
ALTER DATABASE MOUNT
```

4.2.2 打开一个关闭的数据库

通过打开数据库，可以使得已装载的但是关闭的数据库用于一般的用途。要打开已装载的数据库，可以使用带 OPEN 选项的 ALTER DATABASE 语句：

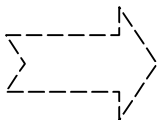
```
ALTER DATABASE OPEN
```

在执行这条语句之后，任何一个拥有 CREATE SESSION 系统权限的有效的 Oracle 用户都能连接到该数据库上。

4.2.3 以只读模式打开数据库

当消除任何对联机的数据内容更改的可能性时，以只读模式打开数据库可以使用户查询打开的数据库。以只读模式打开数据库可以保证数据文件和重做日志文件不被重写，但是这种模式不会限制数据库的恢复过程或者不生成重做的有关改变数据库状态的操作。例如，可以使数据文件脱机或者联机，因为这些操作不影响数据内容。

如果查询违背了数据库的只读模式而采用了临时表空间，例如磁盘排列，那么查询的发布者必须有一个本地管理的表空间作为默认的临时表空间。否则，查询将会失败。这种



情况会在第11章的“创建本地管理的临时表空间”一节中加以解释。

理想情况下，当为备用数据库选择只读模式或恢复模式时，选择采用只读模式打开数据库。要知道这些模式都是独占模式。

下面的语句以只读模式打开数据库：

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

也可以使用读写模式打开数据库，如下所示：

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

然而，读写模式是默认的模式。

注意：不能使用带 READ ONLY 子句的 RESETLOGS 子句。

4.2.4 限制对打开的数据库访问

要以受限模式安置实例，可以使用带 ENABLE RESTRICTED SESSION 子句的 SQL 语句 ALTER SYSTEM。在以受限模式安置实例后，可以考虑在执行任何一个管理任务之前取消所有当前的用户会话。要解除实例的受限模式，可以使用带 DISABLE RESTRICTED SESSION 选项的 ALTER SYSTEM 语句。

4.3 关闭数据库

要初始化数据库的关闭操作，可以使用 SQL*Plus SHUTDOWN 命令。在关闭操作完成之前，控制权不会返回到初始化数据库关闭操作的会话中。当正在进行关闭操作时，如果用户试图进行连接，那么他将收到类似下面的消息：

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

注意：如果你是通过一个共享服务器进程连接到一个数据库上的，你就不能关闭该数据库。

要关闭数据库和实例，必须首先以 SYSOPER 或 SYSDBA 进行连接。关闭数据库可以采取多种模式。以下章节中将讨论如下模式：

- 用 NORMAL 选项关闭数据库
- 用 IMMEDIATE 选项关闭数据库
- 用 TRANSACTIONAL 选项关闭数据库
- 用 ABORT 选项关闭数据库

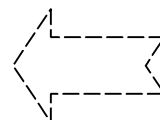
4.3.1 用 NORMAL 选项关闭数据库

要以普通模式关闭数据库，可以使用带 NORMAL 选项的 SHUTDOWN 命令：

```
SHUTDOWN NORMAL
```

要采用普通模式关闭数据库，需要以下几个条件：

- 在发布语句后不允许新的连接。
 - 在数据库关闭之前，Oracle 将等待所有当前连接的用户解除与数据库的连接。
- 数据库的下一启动将不需要任何实例的恢复过程。



4.3.2 用 IMMEDIATE 选项关闭数据库

只在以下情形中使用立即数据库关闭模式：

- 要初始化一个自动的并且未参与的备份
- 当要马上发生电源的关闭动作时
- 当数据库或其中一个应用程序的功能不正常，此时又不能联络到用户以请求注销操作或者这些用户不能注销时

要立即关闭数据库，可以使用带 IMMEDIATE 选项的 SHUTDOWN 命令：

```
SHUTDOWN IMMEDIATE
```

要采用立即模式关闭数据库，需要以下几个条件：

- 在发布语句后，不允许新的连接，也不允许启动新的事务。
- 任何没有提交的事务都会回滚。（如果存在很长的没有提交的事务，这种关闭数据库的方法也许不能快速完成，尽管它的名字是“立即”。）
- Oracle 不等待当前连接的用户解除与数据库的连接。Oracle 隐式回滚活动的事务并解除所有连接的用户。

数据库的下次启动将不需要任何实例的恢复过程。

4.3.3 用 TRANSACTIONAL 选项关闭数据库

如果在允许首先完成活动的事务的情况下，希望执行计划的实例关闭操作时，可以使用带 TRANSACTIONAL 选项的 SHUTDOWN 命令：

```
SHUTDOWN TRANSACTIONAL
```

要使用这种模式关闭数据库，需要以下几个条件：

- 在发布语句后，不允许新的连接，也不允许启动新的事务。
- 在完成所有事务后，任何一个仍连接到实例的客户都将被解除连接。
- 此时，实例的关闭操作就如同提交 SHUTDOWN IMMEDIATE 语句时所做的那样。

数据库的下次启动将不需要任何实例的恢复过程。

事务型的关闭模式可以使客户免于工作失败，同时，它不需要注销所有用户。

4.3.4 用 ABORT 选项关闭数据库

通过终止数据库的实例可以即刻关闭数据库。如果可能的话，只在以下情况执行这种类型的关闭操作：

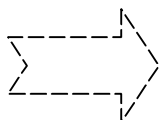
- 数据库或其中的一个应用程序功能不正常并且没有使用其他类型的关闭操作。
- 需要即刻关闭数据库（例如，知道会在一分钟内发生电源的关闭动作）。
- 当启动数据库实例时出现问题。

如果必须使用终止事务和用户连接的办法关闭数据库时，可以发布带 ABORT 选项的 SHUTDOWN 命令：

```
SHUTDOWN ABORT
```

终止数据库关闭模式需要以下一些条件：

- 在发布语句后，不允许新的连接，也不允许启动新的事务。



- 立即终止 Oracle 正在处理的当前客户 SQL 语句。
- 不回滚未提交的事务。
- Oracle 不等待当前连接的用户解除与数据库的连接。Oracle 隐式解除所有连接的用户。

数据库的下一启动将需要实例的恢复过程。

4.4 禁止操作数据库

有些时候需要将数据库置于一种状态，在该状态下只允许有 DBA 的事务、查询、读取或 PL/SQL 语句。该状态称为静止状态，在此状态的系统中没有正在运行的非 DBA 的事务、查询、读取或 PL/SQL 语句。这种状态允许管理员安全地执行一些操作，而这些操作的执行在其他状态是不安全的。这些操作包括以下几类：

- 如果并发用户事务访问同一个对象时会导致失败的操作。例如，改变数据库表的模式或者给需要立即锁的现有表添加一列。
- 并发用户事务可以看到操作中的不需要的中间结果类型的操作。例如，重组表的多步过程，首先是导出表，然后撤消表，最后导入表。如果一个并发用户想在表撤消之后、导入之前访问表，就会看到中间的干扰结果。

如果没有禁止操作数据库的能力，就需要关闭数据库并在受限模式下重新打开数据库。这是一个严格的限制条件，特别是对于那些需要 24 x 7 可用性的系统更是如此。相对来说，禁止操作数据库的限制少得多，原因是它消除了对用户的中断以及有关关闭和重启数据库的停机时间。

注意：对于 Oracle 9i 版本，在禁止操作数据库语境中 DBA 被定义为 SYS 或 SYSTEM。其他的用户，包括那些拥有 SYSDBA 系统权限或 DBA 角色的用户，不允许发布 ALTER SYSTEM QUIESCE DATABASE 语句，或者在数据库禁止操作后不能这么做。

4.4.1 将数据库置于静止状态

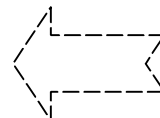
发布如下语句可以将数据库置于静止状态：

```
ALTER SYSTEM QUIESCE RESTRICTED
```

任何非 DBA 的活动会话将一直运行，直到这些会话变为非活动的。活动的会话是指当前在事务、查询、读取或 PL/SQL 语句中的会话；或者是当前持有共享资源的会话（例如，入队列）。不允许任何非活动会话变为活动的状态。例如，如果用户发布一个 SQL 查询来试图强制非活动会话变为活动的，那么查询将被中止。当数据库不处于静止状态时，将继续会话并且处理封锁的动作（例如，先前提到的 SQL 查询）。

一旦所有非 DBA 会话变为非活动的，ALTER SYSTEM QUIESCE RESTRICTED 语句的任务也随之结束，并且数据库也将进入静止状态。在 Oracle Real Application Clusters 环境中，这条语句将影响所有实例，而不仅仅是发布这条语句的实例。

注意：必须激活数据库资源管理器功能，并且自实例启动到成功发布 ALTER SYSTEM QUIESCE RESTRICTED 语句以后该管理器必须一直处于激活状态。可以通过数据库资源管理器的工具来防止非 DBA 会话变为活动状态。当这条语句生效时，任何要改变当前资源



计划的企图都将排队等候，直到系统处于非静止状态。

ALTER SYSTEM QUIESCE RESTRICTED 语句可以等很长的时间，以便活动会话变为非活动的。如果中断了这项请求，或者会话因为某种原因在所有活动会话静止之前异常终止，那么 Oracle 将自动撤消语句的任何部分的效果。

如果通过连续的多个 Oracle Call Interface(OCI)读取可以执行查询，那么 ALTER SYSTEM QUIESCE RESTRICTED 语句将不等待所有读取结束；它只会等待当前读取的结束。

对于专用服务器连接，ALTER SYSTEM QUIESCE RESTRICTED 语句不会给用户登录强加任何的限制条件。但是，对于共享服务器连接，在发布这条语句后，所有非 DBA 登录都将在数据库资源管理器中排队，而不允许处理这些非 DBA 登录。对于用户而言，这条语句就像登录被悬挂了一样。当数据库处于非静止状态时，将继续登录。

即使发布这条语句的会话还存在，数据库仍然会保持在排队状态。DBA 必须登录数据库以便发布指定数据库非静止状态的语句。

在静止状态，即使每个实例中都设置了检验点，也不能使用文件系统拷贝将数据库的数据文件做冷备份。不能这样做的原因是，静止状态下的联机数据文件的文件头就好像仍然处于正在访问的状态，看起来不完全像关闭后的样子。同样地，在数据库静止状态要执行任何联机表空间的数据文件的热备份，都需要首先使用 ALTER TABLESPACE ... BEGIN BACKUP 语句将表空间置于备份模式。

4.4.2 将系统还原为 Normal 操作状态

下面的语句将系统还原为 Normal 操作状态：

ALTER SYSTEM UNQUIESCE

此时，允许处理所有非 DBA 的活动。在 Oracle Real Application Clusters 环境中，不需要从同一个强加了静止状态的会话甚至是实例中发布这条语句。如果发布 ALTER SYSTEM UNQUIESCE 语句的会话异常终止，那么 Oracle 数据库服务器将确保结束非静止操作。

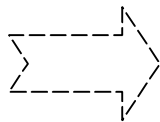
4.4.3 查看实例的禁止操作状态

可以使用 V\$INSTANCE 视图来查看实例的当前状态。该视图包含名为 ACTIVE_STATE 的列，其值如下表所示：

ACTIVE_STATE	说明
NORMAL	Normal 非禁止状态
QUIESCING	处于禁止状态，但是仍然激活非 DBA 会话的运行
QUIESCED	处于禁止状态，不激活或不允许任何非 DBA 会话

4.5 暂停和继续数据库

ALTER SYSTEM SUSPEND 语句通过中断所有数据文件（文件头部和文件数据）和控制文件的输入和输出（I/O）暂停数据库，因此允许数据库在没有任何 I/O 冲突的情况下进行备份。当数据库暂停时，允许完成所有先前存在的 I/O 操作并将任何新的数据库访问置于静止状态。



暂停命令将暂停数据库，并且该命令不是哪个实例所特有的。因此，在 Oracle Real Application Clusters 环境中，如果某个系统键入暂停命令，那么内部封锁机制将在实例间传播中断请求，从而将所有活动实例静止在一个已知的簇中。但是，在暂停另一个实例的时候不要启动新的实例，因为系统将不会暂停新的实例。

使用 ALTER SYSTEM RESUME 语句来继续普通的数据库操作。可以为不同的实例指定 SUSPEND 和 RESUME。例如，如果正在运行实例 1、2 和 3，那么从实例 1 中发布 ALTER SYSTEM SUSPEND 语句，然后从实例 1、2 或 3 中发布 RESUME，效果是一样的。

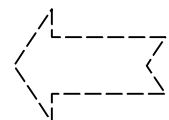
在系统中暂停/继续功能非常有用，它可以镜像磁盘或文件，然后分割该镜像，从而提供另一种备份和恢复解决方案。在正在写入时，如果使用的系统不能从现有的数据库中分割镜像的磁盘，那么可以使用暂停/继续功能来帮助分割。

暂停/继续功能不适合于代替普通的关闭操作，但是，已暂停的数据库的备份可以包含未提交的更改内容。

警告：不能使用 ALTER SYSTEM SUSPEND 语句代替将表空间置于热备份模式。在任何一种数据库暂停操作之前使用 ALTER TABLESPACE BEGIN BACKUP 语句。

下面的语句阐述了 ALTER SYSTEM SUSPEND/RESUME 的用途。查询 V\$INSTANCE 视图来确认数据库的状态。

```
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
```



第 2 部分 Oracle 服务器进程和存储结构

第 2 部分主要介绍 Oracle 服务器进程及支持其操作的底层数据库存储结构。
这部分包含以下几章内容：

- 第 5 章 管理 Oracle 进程
- 第 6 章 管理控制文件
- 第 7 章 管理联机重做日志
- 第 8 章 管理已存档的重做日志
- 第 9 章 使用 LogMiner 来分析重做日志文件
- 第 10 章 管理作业队列
- 第 11 章 管理表空间
- 第 12 章 管理数据文件
- 第 13 章 管理撤消空间

第 5 章 管理 Oracle 进程

本章描述如何管理 Oracle 实例的进程，主要包含以下内容：

- 服务器进程
- 将 Oracle 配置为共享服务器
- 关于 Oracle 的后台进程
- 监视 Oracle 实例的进程
- 为并行执行管理进程
- 为外部过程管理进程
- 终止会话

5.1 服务器进程

Oracle 创建服务器进程来处理连接到实例的用户进程的请求。服务器进程可以是一个专用服务器进程，此时该服务器进程只服务于一个用户进程；如果数据库服务器配置为共享服务器，那么服务器进程也可以是共享服务器进程，此时该服务器进程可以服务于多个用户进程。

5.1.1 专用服务器进程

图 5-1 “Oracle 专用服务器进程”说明了专用服务器进程是如何运转的。在这个图表中，有两个用户进程通过专用服务器进程连接到 Oracle 上。

通常，最好通过调度程序进行连接并使用共享服务器进程。这在图 5-2 “Oracle 共享服务器进程”中进行了说明。因为共享服务器进程保持了运行实例所需的最少数目的进程，所以共享服务器进程效率更高。

但是，在以下情形，用户和管理员需要直接使用专用服务器进程连接到实例中：

- 提交批处理作业（例如，当作业允许服务器进程少量或无空闲时间时）
- 使用 Recovery Manager（恢复管理器）备份、还原或恢复数据库

当 Oracle 配置为共享服务器时，如果要求一个专用服务器连接，那么用户必须使用配置为专用服务器的网络服务名称进行连接。需要明确的是，网络服务名称的值应该在连接描述符中包括 SERVER=DEDICATED 子句。

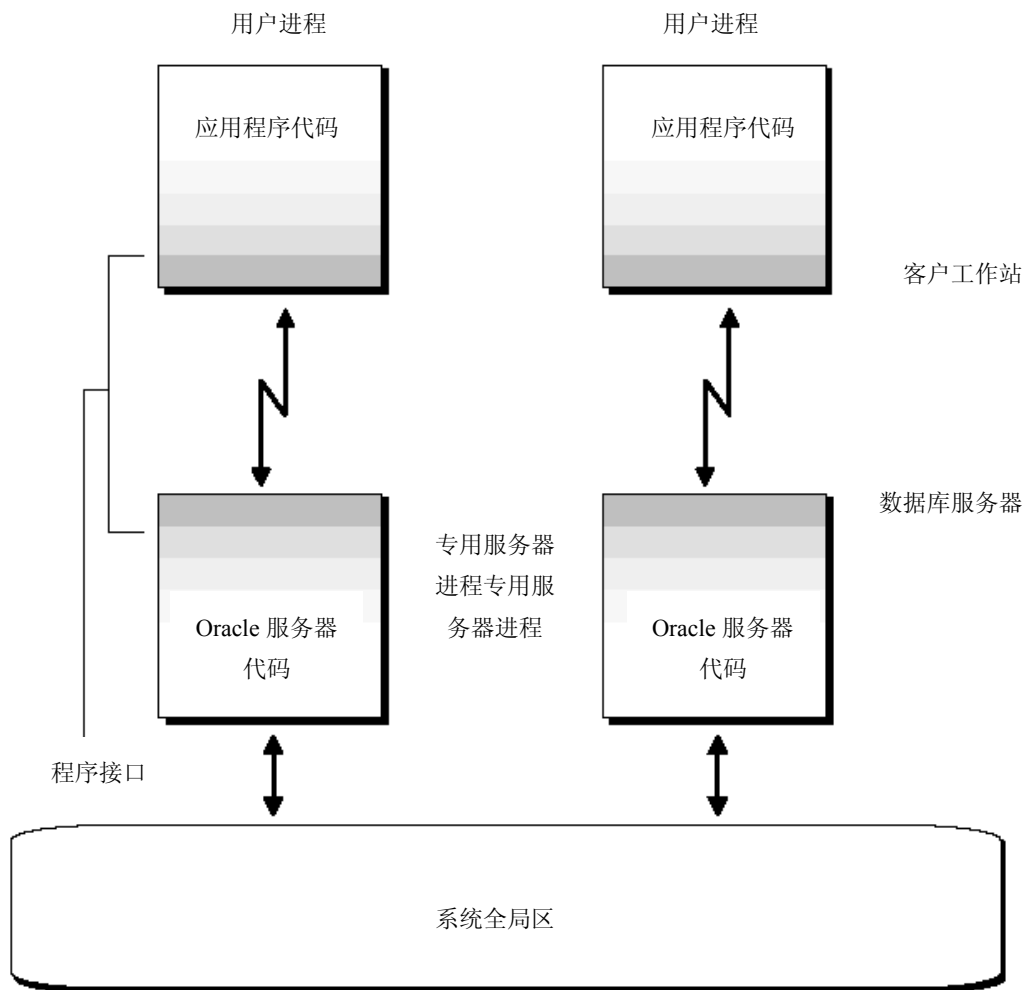
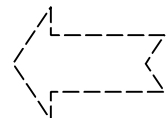


图 5-1 Oracle 专用服务器进程

5.1.2 共享服务器进程

设想一下专用服务器进程进入系统的次序。消费者发出一个定单，职员将定单输入到数据库中。对于大多数交易，职员是在电话上告之消费者的，而且专用于该职员的用户进程的服务器进程仍然保持空闲状态。在大多数交易中，并不一定需要服务器进程，而且，因为空闲的服务器进程占据了系统资源，所以对于输入定单的其他职员来说系统将会更慢。

共享服务器体系结构消除了每个连接都要有专用服务器进程的必要性(如图 5-2 所示)。



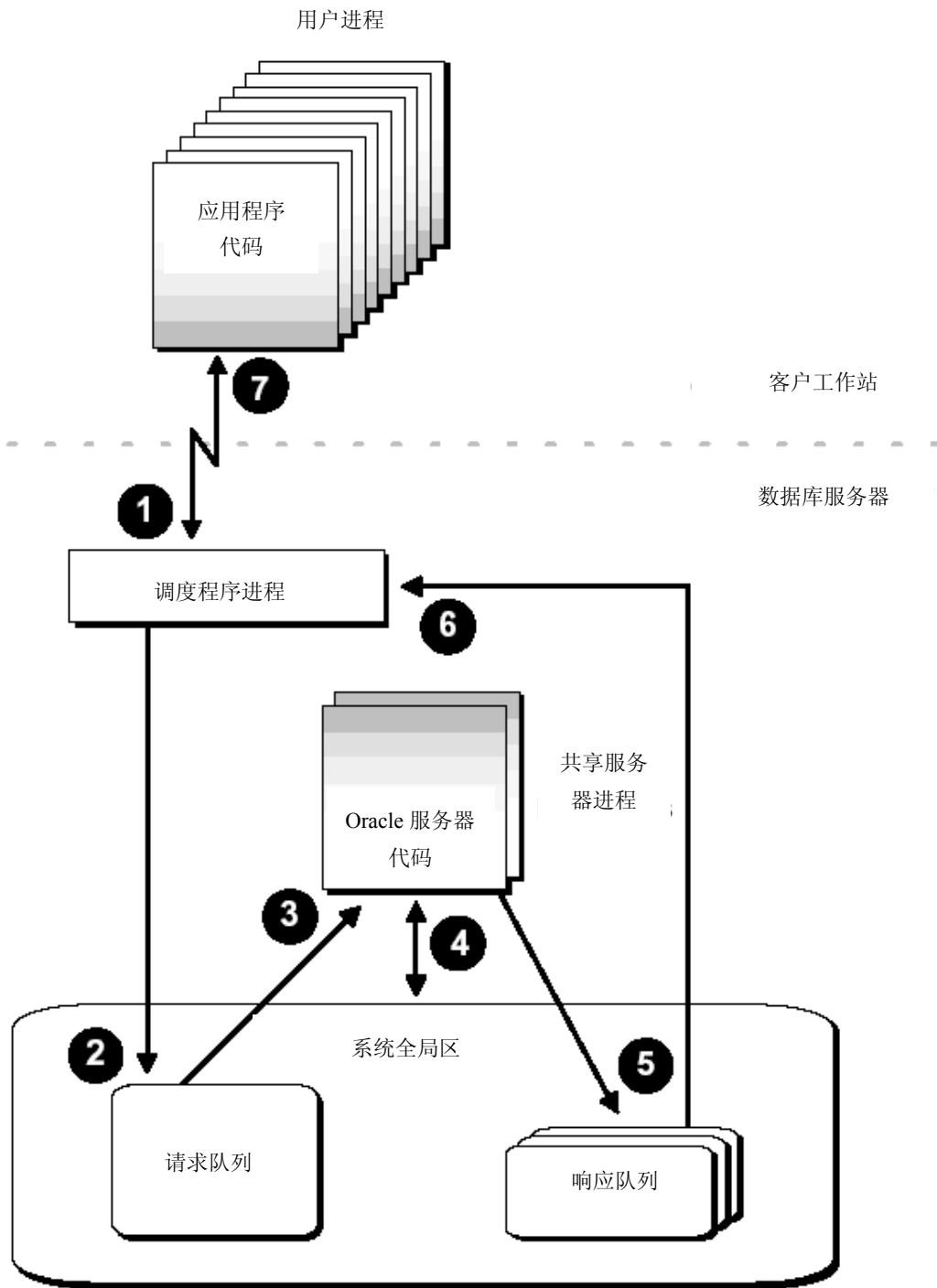
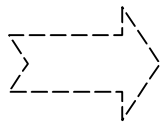


图 5-2 Oracle 共享服务器进程

在共享服务器的配置中，客户的用户进程连接到调度程序。调度程序可以支持多个并发客户连接。每个客户连接都是连接到一个虚拟回路上的。虚拟回路是一块共享内存，调



度程序在进行客户数据库连接的请求和答复时使用它。当请求到达时，调度程序将虚拟回路放置到一个公用的队列中。空闲的共享服务器将从公用队列中拾取虚拟回路，运行该项请求并在重新得到公用队列中的另一个虚拟回路之前放弃这个虚拟回路。这种方法使得一个很小的服务器进程的存储池就可以服务大量的客户。与专用服务器模式比较，共享服务器体系结构的一个主要优点在于用更少的系统资源来支持已经增长的用户数目。

共享服务器体系结构需要 Oracle Net Services。目标是共享服务器的用户进程必须通过 Oracle Net Services 进行连接，即使这些用户进程与 Oracle 实例是在同一台机器上，也是如此。

要将系统配置为共享服务器必须做几件事情，这些内容将在下一节中进行讨论。

5.2 将 Oracle 配置为共享服务器

通过设置数据库初始化参数可以启动共享服务器。共享服务器需要启动 Oracle Net Services 监听进程。这一节讨论共享服务器初始化参数的设置以及如何更改这些设置的问题。

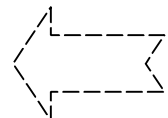
这一节包含以下一些内容：

- 配置共享服务器的初始化参数
- 设置调度程序的起始数目（DISPATCHERS）
- 设置共享服务器的起始数目（SHARED_SERVERS）
- 修改调度程序和服务器进程
- 监视共享服务器

5.2.1 配置共享服务器的初始化参数

控制共享服务器的初始化参数有以下一些参数：

参数	说明
必须设置的参数	
DISPATCHERS	在共享服务器体系结构中配置调度程序进程。
可选的参数。如果没有指定以下几个参数，Oracle 将选择适合的默认值。	
MAX_DISPATCHERS	指定同时运行的调度程序进程的最大数目。
SHARED_SERVERS	指定启动实例时可以创建的共享服务器进程的数目。
MAX_SHARED_SERVERS	指定可以同时运行的共享服务器进程的最大数目。
CIRCUITS	指定进出网络会话时可以利用的虚拟回路的总数。
SHARED_SERVER_SESSIONS	指定用户会话允许的共享服务器的总数。设置这个参数可以为专用服务器保留用户会话。
共享服务器影响的其他初始化参数可以按需要进行调整。	
LARGE_POOL_SIZE	指定大型存储池堆分配的字节大小。共享服务器也许会将默认值设置为很高的值，这样会导致在性能或启动数据库方面出现问题。
SESSIONS	指定系统可以创建的会话的最大数目。对于共享服务器也许需要调整。



5.2.2 设置调度程序的起始数目 (DISPATCHERS)

在实例启动时开始运行的调度程序进程的数目是由初始化参数 DISPATCHERS 控制的。在初始化文件中可以指定多个 DISPATCHERS 参数，但是它们必须是彼此相邻的。在 Oracle 内部将为每个 DISPATCHERS 参数分配一个 INDEX 索引值，这样在以后使用 ALTER SYSTEM 语句时就可以明确引用某个 DISPATCHERS 参数。

为每个实例设置适当的调度程序进程的数目依赖于希望从数据库中获得怎样的性能，主操作系统限制了每个进程（因操作系统而异）的连接数目以及每个网络协议需要的连接数目。实例必须能够提供与数据库系统上并发用户相同数目的连接个数。实例启动后，如果需要可以启动更多的调度程序进程。这个内容将在本章后面的“添加和删除调度程序进程”一节中加以介绍。

对于一般的系统，每 1000 个连接分配 1 个调度程序，就可以运转得很好，但是调度程序的个数也可以适当的多些。例如，如果预料在最高峰会有 1500 个连接，那么可以配置两个调度程序。因为配置太多的调度程序会降低性能，所以过高估计并无好处。使用这个比率作为参考，但也要特殊情况特殊处理。

下面是设置 DISPATCHERS 初始化参数的几个例子。

例子：典型情况

这是设置 DISPATCHERS 初始化参数的典型例子。

```
DISPATCHERS=(PROTOCOL=TCP)"
```

例子：强制调度程序使用 IP 地址

要强制调度程序使用 IP 地址，可以键入如下代码：

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

这个例子将启动两个调度程序，这两个调度程序将在上面的 IP 地址上进行监听，其中，该 IP 地址必须是实例所在的主机的有效 IP 地址。

例子：强制调度程序使用端口

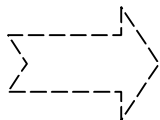
要强制调度程序使用精确的地址，可以为其添加 PORT，如下所示：

```
DISPATCHERS=(ADDRESS=(PROTOCOL=TCP)(PORT=5000))"
DISPATCHERS=(ADDRESS=(PROTOCOL=TCP)(PORT=5001))"
```

5.2.3 设置共享服务器的起始数目 (SHARED_SERVERS)

初始化参数 SHARED_SERVERS 指定实例启动时希望创建的共享服务器进程的数目。Oracle 将根据请求队列的长度动态调整共享服务器进程的数目。可以创建的共享服务器进程的数目在两个初始化参数 SHARED_SERVERS 和 MAX_SHARED_SERVERS 之间。通常，在每 10 个连接分配一个共享服务器的比例下系统似乎更趋于稳定。

对于 OLTP 应用程序，连接的数目和服务器数目的比例可能会更高。在请求的比率降低时，或者当服务器用于请求的比率降低时，就会出现连接的数目和服务器数目比例增高的现象。另一方面，在高比率请求或者服务器用于请求的比率很高的应用程序中，连接的数目和服务器数目的比例将会降低。



根据应用程序可以将 MAX_SHARED_SERVERS 设为合理的值。对于典型配置来说，Oracle 将为 SHARED_SERVERS 和 MAX_SHARED_SERVERS 两个参数提供了更好的默认值，但是这些参数的最佳值将根据应用程序的不同而有所差异。

注意：在 Windows NT 上，当 MAX_SHARED_SERVERS 参数的值设置的很高时要格外小心，因为每个服务器都是一个公用进程中的一个线程。

初始化参数 MAX_SHARED_SERVERS 是一个静态初始化参数，所以在没有关闭数据库时不能改变它的值。但是，SHARED_SERVERS 则是一个动态初始化参数，该参数的值可以使用 ALTER SYSTEM 语句进行修改。

5.2.4 修改调度程序和服务进程

运行实例时，可以动态修改 DISPATCHERS 和 SHARED_SERVERS 两个初始化参数的值。如果拥有 ALTER SYSTEM 权限，则可以使用 ALTER SYSTEM 语句对这两个初始化参数的值进行修改。

5.2.4.1 添加和删除调度程序进程

我们可以控制实例中的调度程序进程的数目。如果监视 V\$QUEUE、V\$DISPATCHER 和 V\$DISPATCHER_RATE 这三个视图，就可以知道调度程序进程的装载量一贯都很高，启动另外的调度程序进程来发送用户请求可以提高性能。相反，如果调度程序进程的装载量一贯都很低的话，减少调度程序的数目可以提高性能。

要改变调度程序进程的数目，使用 SQL 语句 ALTER SYSTEM 即可。可以为现有的 DISPATCHERS 值启动新的调度程序进程，或者添加新的 DISPATCHERS 值。可以添加的调度程序的最大数目由 MAX_DISPATCHERS 参数指定。

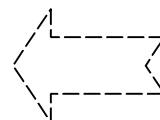
如果要为某个共享服务器减少调度程序的数目，那么调度程序将不能立即被删除。只有当用户解除连接时，Oracle 才会最终将调度程序的个数减少到 DISPATCHERS 参数指定的数目上。

下面的语句为 TCP/IP 协议动态更改调度程序进程的数目至 5，并为 TCP/IP 添加 SSL (TCPS) 协议的调度程序进程。这里没有为 TCPS 协议设置 DISPATCHERS 初始化参数(只有惟一一个 DISPATCHERS 初始化参数是为 TCP/IP 协议设定的)，所以下面的语句实际上只添加了一个 DISPATCHERS 初始化参数。

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

如果 TCP 协议的当前调度程序进程的数目少于 5 个，那么 Oracle 将创建新的调度程序，直到调度程序进程的数目有 5 个。如果当前调度程序进程的数目多于 5 个，那么在用户解除连接时 Oracle 将消除其中多余的调度程序进程。

注意：关键字 INDEX 可以用于识别要修改的是哪个 DISPATCHERS 参数。INDEX 的值可以从 0 到 n ，这里的 n 比定义的 DISPATCHERS 参数数目少 1。如果 ALTER SYSTEM



语句指定 INDEX 的值等于 $n+1$ ，这里的 n 是当前调度程序的数目，那么将添加一个新的 DISPATCHERS 参数。要确定分配给某个 DISPATCHERS 参数的索引值，可以查询 V\$DISPATCHERS 视图中的 CONF_INDEX 的值。

5.2.4.2 关闭指定的调度程序进程

可以关闭指定的调度程序进程。要识别指定关闭的调度程序进程的名称，可以使用 V\$DISPATCHERS 这种动态性能视图。

```
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))
```

每个调度程序的名称都是独一无二的，名称的格式为 *Dnnn*。

如果要关闭调度程序 D002，可以发布以下的语句：

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

关键字 IMMEDIATE 将阻止调度程序接受新的连接，并且 Oracle 会通过这个调度程序终止所有现有的连接。在清理完所有会话后，将关闭调度程序进程。如果没有指定 IMMEDIATE，那么调度程序将一直处于等待状态，直到它的所有用户解除连接，并且在关闭之前终止所有连接。

5.2.4.3 改变共享服务器进程的最少数目

启动实例之后，可以通过使用 SQL 语句 ALTER SYSTEM 来更改共享服务器进程的最少数目。当拥有的共享服务器的数目多于指定的最少数目限制时，Oracle 将终止处于空闲的服务器。

如果将 SHARED_SERVERS 参数设置为 0，那么 Oracle 将终止所有当前处于空闲状态的服务器，并且在 SHARED_SERVERS 参数值增加前不启动任何新的服务器。因此，将 SHARED_SERVERS 参数设置为 0 可以有效地禁止共享服务器。

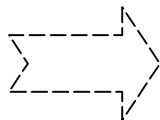
下面的语句将动态地设置两个共享服务器进程：

```
ALTER SYSTEM SET SHARED_SERVERS = 2;
```

5.2.5 监视共享服务器

以下视图将有助于获得有关共享服务器配置的信息以及监视性能。

视图	说明
V\$DISPATCHER	提供有关调度程序进程的信息，包括姓名、网址、状态、各种使用统计表和索引号码。
V\$DISPATCHER_RATE	提供调度程序进程的速率统计表。
V\$QUEUE	包含有关共享服务器消息队列的信息。
V\$SHARED_SERVER	包含有关共享服务器进程的信息。



(续表)

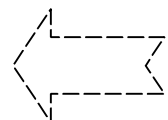
视图	说明
V\$CIRCUIT	包含有关虚拟回路的信息，这些虚拟回路是一些通过调度程序和服务器连接到数据库的用户连接。
V\$SHARED_SERVER_MONITOR	包含协调共享服务器的信息。
V\$SGA	包含各个系统全局区（SGA）组的容量信息。在协调共享服务器时也许会有用处。
V\$SGA_START	关于 SGA 的详细统计信息，用于协调共享服务器。
V\$SHARED_POOL_RESERVED	列出统计表以帮助协调共享存储池中的保留的存储池和空间。

5.3 关于 Oracle 的后台进程

要使性能最优化并容纳更多的用户，多进程 Oracle 系统就需要使用一些另外的进程，我们称这些进程为后台进程。后台进程合并了一些 Oracle 的功能，否则，就需要为每个用户进程运行多个 Oracle 程序。后台进程异步执行 I/O 操作并监视其他的 Oracle 进程，以便为获得更好的性能和可靠性提供更多的并行操作。

下面是一些基本的 Oracle 后台进程，其中某些后台进程将在本书的其他地方详细讨论。使用附加的 Oracle 数据库服务器特性或选项，将导致出现更多的后台进程。例如，如果使用高级排队，将会出现队列监视程序（QMn）这种后台进程。

进程名称	说明
数据库写入进程（DBWn）	数据库写入程序将数据库高速缓冲区中的修改的块写入数据文件。对于大多数系统来说，一个数据库写入程序（DBW0）就已经足够，但是，对于有大量修改数据任务的系统来说，就需要配置额外的进程（DBW1 到 DBW9）来改善写入的性能。初始化参数 DB_WRITER_PROCESSES 用来指定 DBWn 进程的个数。
日志写入进程（LGWR）	日志写入进程将重做日志条目写入磁盘中。重做日志条目在系统全局区（SGA）的重做日志缓冲区中生成，而且 LGWR 按顺序将重做日志条目写入联机重做日志文件。如果数据库有多元重做日志，那么 LGWR 会将重做日志条目写入联机重做日志文件的某个组中。
检验点进程（CKPT）	在特定时期，系统全局区的所有修改过的数据库缓冲区都是通过 DBWn 写入到数据文件中的。这种事件称为检验点。检验点进程负责在检验点通知 DBWn，并且更新数据库的所有数据文件和控制文件，文件中指明最新的检验点。
系统监视进程（SMON）	当失败的实例再次启动时，系统监视程序执行紧急恢复操作。在多实例系统（Oracle 9i Real Application Clusters）中，一个实例的 SMON 进程可以为其他失败的实例执行实例恢复操作。SMON 进程还将清除不再使用的临时段，以及恢复在紧急实例恢复过程中因为文件读取或脱机错误遗漏的停用的事务。当表空间或文件恢复为联机状态时，事务最终将由 SMON 进程进行恢复。 SMON 进程还合并了数据库字典管理表空间中的空闲区域，以便使空闲空间连接，从而更容易分配空闲空间。
进程监视进程（PMON）	当用户进程失败时，由进程监视程序执行恢复操作。PMON 进程负责清理缓存和进程使用的空闲资源。PMON 进程还检查调度程序进程和服务进程，并在这些进程失败后重新启动它们。



(续表)

进程名称	说明
存档进程 (ARCn)	一个或多个存档程序进程备份联机重做日志文件,以便在文件中充满日志或发生日志切换时存储档案。存档程序进程是第 8 章“管理已存档的重做日志”的主要内容。
恢复进程 (RECO)	恢复程序进程用于解决分布式事务,在分布式数据库中这些事务的悬而未决要归咎于网络或系统的失败。在计时间隔内,本地 RECO 会试图连接远程数据库并自动完成任何一个未解决的分布式事务在本地部分的提交或回滚操作。
调度进程 (Dnnn)	调度程序是个可以选择的后台进程,只在使用共享服务器配置时出现。共享服务器在“将 Oracle 配置为共享服务器”一节中已经做过介绍。
全局缓存服务 (LMS)	在 Oracle Real Application Clusters 环境中,用该进程管理资源并提供实例间的资源控制。
作业队列协调进程 (CJQ0)	这是一个协调实例的作业队列进程的程序。它监视 JOBS 表(作业队列中的作业表)并启动作业队列进程 (Jnnn)。当需要执行作业时, Jnnn 进程将执行由 DBMS_JOBS 包创建的作业请求。这是第 10 章“管理作业队列”的主要内容。 另外,视图最多可以查看 1000 个 Jnnn 进程,之后自动刷新视图的内容。进程定期唤醒和刷新计划刷新的视图内容。 Jnnn 进程的另外一个功能是将队列消息传播到另一个数据库上的队列。与其他很多 Oracle 后台进程不同的是,如果作业队列进程或协调程序 (CJQ0) 失败,并不会引起实例失败。

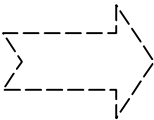
5.4 监视 Oracle 实例的进程

这一节列出了一些数据字典视图,使用这些视图可以监视 Oracle 实例。这些视图在它们各自的职责范围内是可以通用的。还有其他一些视图,主要对应于指定的某个进程,关于这些视图的内容将在本书有关进程的章节中加以介绍。另外。这一节还介绍了脚本和监视封锁状态的视图。

5.4.1 进程和会话视图

这些视图提供了进程和会话特定的信息。

视图	说明
V\$PROCESS	包含有关当前活动进程的信息。
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 统计表。
V\$SESSION_LONGOPS	该视图显示各种运行时间超过 6 秒(绝对时间)的操作的状态。通常,这些操作包括一些备份和恢复函数、统计表集合和查询执行。为每个 Oracle 版本添加了更多的操作。
V\$SESSION_WAIT	列出当前会话正在等待的资源或事件。
V\$SYSSTAT	包含会话统计表。
V\$RESOURCE_LIMIT	提供关于某些系统资源当前使用的全局资源及其最大值的信息。
V\$SQLAREA	包含有关共享 SQL 区域的统计表并且统计表中为每个 SQL 字符串列一行信息。还提供内存中经过语法分析并准备执行的有关 SQL 语句的统计表。
V\$LATCH	包含非父锁存器的统计表以及父锁存器的汇总统计表。



5.4.2 监视锁

utllockt.sql 脚本以树型结构的形式显示系统中正在等待封锁的会话以及正在等待使用的锁。使用特定的查询工具，比如 SQL*Plus，该脚本会打印出系统中正在等待封锁的会话以及相应的用于封锁的锁。该脚本文件的位置因操作系统而异；可以查看相应的 Oracle 文档。另一个脚本 catblock.sql 用于创建 utllockt.sql 脚本所需的锁视图，因此在运行 utllockt.sql 脚本之前必须先运行脚本 catblock.sql。

下面的视图可以用于监视锁。

视图	说明
V\$LOCK	列出 Oracle 服务器当前拥有的锁以及对锁或锁存器没有完成的请求。

5.4.3 跟踪文件和报警文件

每个服务器和后台进程都可以往有关的跟踪文件中写入信息。当进程检查出内部错误时，会将错误信息转储到它的跟踪文件中。写入到跟踪文件的某些信息是有意写给数据库管理员看的，而其他的信息是用于 Oracle 支持服务的。跟踪文件信息也可以用于协调应用程序和实例。

报警文件或报警日志是一种特殊的跟踪文件。数据库的报警文件是按年代顺序排列的消息和错误的日志，其中包含以下内容：

- 发生的所有内部错误（ORA-600），成组的损坏错误（ORA-1578）以及死锁错误（ORA-60）
- 管理操作，比如 CREATE，ALTER 和 DROP 语句以及 STARTUP，SHUTDOWN 和 ARCHIVELOG 语句
- 有关共享服务器和调度程序进程函数的一些消息和错误
- 在整个视图内容自动刷新的过程中发生的错误
- 数据库和实例启动时的所有初始化参数的值

Oracle 使用报警文件来保存这些特殊操作的日志，进而提供了另一种方法来显示控制台上的信息（虽然某些系统是在控制台上显示信息的）。如果操作成功，完成的消息“completed”会写入报警文件中，并且有相应的时间标志。

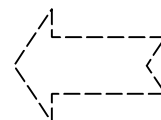
控制跟踪文件大小和位置的初始化参数有以下三个：

- BACKGROUND_DUMP_DEST
- USER_DUMP_DEST
- MAX_DUMP_FILE_SIZE

这些参数将在下面的章节中加以讨论。

5.4.3.1 使用跟踪文件

应该定期地检查实例的报警文件和其他跟踪文件，以便查看后台进程是否遇到错误。例如，当日志写入程序进程（LGWR）不能往某个组的某个成员写入信息时，能够说明该问题本质的错误消息，就会写入到 LGWR 跟踪文件和数据库的报警文件。如果看到类似这样的错误消息，就表明发生了介质或 I/O 方面的问题，应该立即进行纠正。



除了重要的统计表以外，Oracle 还将初始化参数的值写入到报警文件中。例如，当使用正常模式或立即模式关闭实例时（但不能采用放弃模式），Oracle 将自从实例开始连接到实例的最高峰时刻的并发会话的数目写入到报警文件。你可以使用这个数字来确定是否需要升级 Oracle 会话许可证。

5.4.3.2 指定跟踪文件的位置

后台进程的所有跟踪文件和报警文件都将写入到由初始化参数 `BACKGROUND_DUMP_DEST` 指定的目标目录中。服务器进程的所有跟踪文件都将写入到由初始化参数 `USER_DUMP_DEST` 指定的目标目录中。在不同的操作系统中，跟踪文件拥有不同的名称，但是每个文件通常包括写入文件的进程的名称（比如 LGWR 和 RECO）。

5.4.3.3 控制跟踪文件的大小

可以使用初始化参数 `MAX_DUMP_FILE_SIZE` 来控制跟踪文件（报警文件）除外的大小。这个限制条件是作为操作系统块的数目来设置的。要控制报警文件的大小，在不再需要该文件时必须手动删除它；否则，Oracle 将继续向这个文件追加内容。在实例运行时，可以很安全地删除掉报警文件，同时也可以首先为该文件做一个存档备份，然后再删除它。

5.4.3.4 控制 Oracle 写入跟踪文件的过程

后台进程通常在适当的时候将信息写入到跟踪文件。在使用 `ARCn` 后台进程时，可以通过一个初始化参数来控制产生的跟踪信息的类型和数量。这个内容将在第 8 章的“控制存档日志进程生成的跟踪输出”一节中加以介绍。其他的后台进程就没有这样的灵活性。

不管何时发生内部错误，错误信息都将代表服务器进程写入到跟踪文件。另外，为了处理实例的所有 SQL 语句，初始化参数 `SQL_TRACE = TRUE` 的设置会导致 SQL 跟踪工具生成性能统计表，并将这些内容写入初始化参数 `USER_DUMP_DEST` 指定的目标目录中。

在用户发出请求时，也可以为服务器进程生成跟踪文件。不管初始化参数 `SQL_TRACE` 当前设置的是什么值，通过使用 SQL 语句 `ALTER SESSION SET SQL_TRACE`，每个会话都可以代表有关的服务器进程允许或禁止跟踪记录。下面的例子中，允许某个指定的会话使用 SQL 跟踪工具：

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

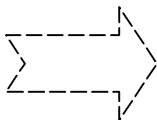
警告：因为服务器进程使用的 SQL 跟踪工具会引起系统显著的架空，从而对性能造成严重的影响，所以这个功能只在收集统计表时使用。

对于共享服务器，每个使用调度程序的会话都会路由到共享服务器进程，而且只在会话允许跟踪的时候（或者遇到错误时）将跟踪信息写入到服务器的跟踪文件。因此，要跟踪使用调度程序连接的某个特定的会话，就不得不研究几个共享服务器的跟踪文件。

`DBMS_SESSION` 和 `DBMS_SYSTEM` 两个包也可以用于控制会话的 SQL 跟踪。

5.5 为并行执行管理进程

这一节描述如何管理 SQL 语句的并行处理。这种配置下的 Oracle 可以将 SQL 语句的



处理任务在多个并行进程之间进行划分。

一些 SQL 语句的执行可以并行化。并行度用并行执行的服务器数目表示，而这些并行执行的服务器可以与某个单个的操作联系在一起。并行的程度由以下内容来确定：

- 语句中的 PARALLEL 子句
- 对于查询中引用的对象，在创建或更改该对象时使用的 PARALLEL 子句
- 插入到语句中的并行提示
- Oracle 确定的默认值

在第 15 章的“并行创建表”一节中有一个例子说明了并行执行的使用问题。

这一节包含以下内容：

- 管理并行执行的服务器
- 为某个会话更改并行执行

注意：本节中描述的并行执行功能在 Oracle 9i 企业版和 Oracle 9i 个人版中同样有效。

5.5.1 管理并行执行的服务器

拥有了并行执行特性，并行执行协调器进程就可以调度并行执行服务器的存储池的执行并且将所有来自于这些并行执行服务器的结果发送给用户。并行执行服务器进程通过它的程序执行段仍然保持与语句的联系。当完整地处理完这条语句后，就可以用这些进程处理其他的语句。

通过设置初始化参数 PARALLEL_AUTOMATIC_TUNING = TRUE 就可以自动协调并行执行。根据这个设置，Oracle 可以确定影响并行执行性能的其他初始化参数的默认值。

5.5.2 为某个会话更改并行执行

可以用 ALTER SESSION 语句来控制某个会话的并行执行。

5.5.2.1 停用并行执行

在发布 ALTER SESSION DISABLE PARALLEL DML [DDL] QUERY 语句之后，就不能将所有后面的 DML (INSERT、UPDATE、DELETE)、DDL (CREATE、ALTER) 或查询 (SELECT) 语句并行处理。不管与语句相联系的任何的 PARALLEL 子句或并行提示的存在，都将串行执行这些语句。

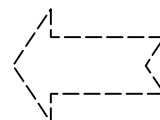
用下面的语句停用并行 DDL：

```
ALTER SESSION DISABLE PARALLEL DDL;
```

5.5.2.2 启用并行执行

在有与语句相联系的 PARALLEL 子句或并行提示的情况下，当发布 ALTER SESSION ENABLE PARALLEL DML [DDL] QUERY 语句之后，就可以并行处理那些 DML (INSERT、UPDATE、DELETE)、DDL (CREATE、ALTER) 或查询 (SELECT) 语句。而这种处理方式是 DDL 和查询语句的默认方式。

只有在特别发布 DML 语句的情况下才可以并行处理该语句。下面的语句使得能够并行处理 DML 语句：



```
ALTER SESSION ENABLE PARALLEL DML;
```

注意：只有安装了 Oracle 的 Partitioning（分区）选项才可以利用并行 DML。

5.5.2.3 强制并行执行

可以强制所有后续的 DML、DDL 或查询语句并行执行，因为使用 ALTER SESSION FORCE PARALLEL DML [DDL] QUERY 语句可以达到并行化的目的。另外，为了使一个特定的并行性程度强制生效，可以重写与后面的这些语句相关联的 PARALLEL 子句。如果在语句中没有指定并行性程度，会采用默认值。但是，在语句中通过提示指定的并行性程度将重写强制指定的程度。

下面的语句强制后面的语句并行执行，并且将并行性程度设置为 5，重写原来的并行性程度的值：

```
ALTER SESSION FORCE PARALLEL DDL PARALLEL 5;
```

要强制 DML 的并行执行，也必须像“允许并行执行”一节中介绍的那样允许 DML 的并行执行。

5.6 为外部过程管理进程

外部过程是从另外的程序中调用的过程，这些外部过程是采用另一种不同的语言编写的。举一个例子，一个 PL/SQL 程序可以调用一个或多个 C 子程序来实现特殊目的。

这些随时可以调用的子程序存储在动态链接库（DLL）中，如果使用的是 Java，也可以存储在类库中，并使用基本语言注册。Oracle 提供了一个具有特殊目的的接口“调用规范”，这个接口允许用户调用使用其他语言编写的外部过程。

简而言之，要调用外部过程，应用程序必须知道外部过程在哪个 DLL 或共享库中。应用程序将激活网络监听进程，该进程依次启动外部过程代理，默认情况下该代理的名称为 extproc。利用监听进程建立起的网络连接，应用程序可以将 DLL 的名称、外部过程的名称以及应用程序需要传递的参数传递给外部过程代理。然后，外部过程代理装载 DLL 并运行外部过程，还将返回值传回应用程序。

为了控制对 DLL 的访问，数据库管理员需要给应用程序开发人员授予相应的执行权限。应用程序开发人员编写外部过程并将指定的外部过程上的执行权限授予其他用户。

外部过程代理可以在作为数据库服务器的计算机上，也可以与监听程序一起位于远程计算机上。

本节讨论如何设置环境来调用外部过程。

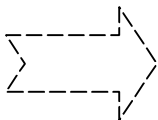
注意：尽管不是必须的，但还是要建议在 Oracle 数据库的安装过程中执行这些设置任务。

5.6.1 为调用外部过程设置一个环境

按如下步骤为调用外部过程设置一个环境。

①编辑 tnsname.ora 文件，通过添加一个入口连接到监听进程（以及随后的外部过程代理）。或者，可以使用 Oracle 网络配置助理来设置 tnsname.ora 文件。

②编辑 listener.ora 文件，为外部过程监听进程添加一个入口。或者，可以使用 Oracle



网络配置助理来设置 listener.ora 文件。

③启动一个单独的监听进程来专门处理外部过程。

④由于监听进程产生的外部过程代理继承了这个监听进程所有的操作系统权限，所以 Oracle 强烈建议限制单独的监听进程的权限。该进程应该没有权利读写数据库文件或 Oracle 服务器地址空间。

还有，这个单独的监听进程的属主不应该是 ORACLE（它是可执行服务器和数据库文件的默认属主）。

⑤如果外部过程代理是在一台远程计算机上，那么可以将可执行的外部过程代理设置在 \$ORACLE_HOME/BIN 路径中。

需要明确的是外部库（DLL 文件）必须是静态链接的。换句话说，它一定没有引用来自其他外部库中（DLL 文件）的任何外部符号。这是因为不能解析这些外部符号，而且这些外部符号将导致外部过程失败。

5.6.2 在 tnsname.ora 文件中为外部过程监听程序添加入口

以下是一个例子，目的是在 tnsname.ora 文件中为外部过程监听进程添加入口：

```
EXTPROC_CONNECTION_DATA=
(DESCRIPTION=
  (ADDRESS= (PROTOCOL=ipc) (KEY=extproc_key))
  (CONNECT_DATA=
    (SID=extproc_agent)))
```

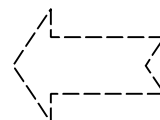
在这个例子中指定的关键字是 extproc_key，它必须与 listener.ora 文件中指定的 KEY 值一致。另外，在这个例子中指定的 SID 名称 extproc_agent 必须与 listener.ora 文件中指定的 SID_NAME 一致。

5.6.3 在 listener.ora 文件中为外部过程添加入口

以下的例子说明了如何在 listener.ora 文件中为外部过程添加入口：

```
LISTENER=
(DESCRIPTION=
  (ADDRESS=
    (PROTOCOL=ipc) (KEY=extproc_key)))
SID_LIST_LISTENER=
(SID_LIST=
  (SID_DESC=
    (SID_NAME=extproc_agent)
    (ORACLE_HOME=/u1/app/oracle/9.0)
    (PROGRAM=extproc)))
```

在这个例子中，PROGRAM 参数设置为 extproc，以便与默认的外部过程代理的名称一致。SID_NAME 是为外部过程代理设定的系统标识符，它可以是任何一个名字。可执行的



外部过程代理必须放在\$ORACLE_HOME/BIN 路径中。

5.7 终止会话

在某些情况下，可能希望终止当前的用户会话。例如，如果想执行一个管理操作，就需要终止所有的非管理会话。

这一节介绍有关终止会话的各个方面的情况，包含以下内容：

- 识别要终止的是哪个会话
- 终止处于激活状态的会话
- 终止处于非激活状态的会话

当某个会话被终止时，会话的事务将被回滚，并且将立即释放会话占有的资源以便别的会话可以利用。

使用 SQL 语句 ALTER SYSTEM KILL SESSION 可以终止当前的会话。

下面的语句终止了一个会话，该会话的系统标识符为 7，它的串行号为 15：

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

5.7.1 识别要终止的是哪个会话

要识别出要终止的是哪个会话，只需指定会话的索引号和串行号即可。要识别会话的系统标识符（sid）和串行号，可以查询 V\$SESSION 动态性能视图。

下面的查询将识别出所有用户名为 jward 的会话：

```
SELECT SID, SERIAL#, STATUS
FROM V$SESSION
WHERE USERNAME = 'JWARD';
SID SERIAL# STATUS
```

```
-----
7 15 ACTIVE
12 63 INACTIVE
```

当会话处理 Oracle 的一个 SQL 调用时，它的状态为 ACTIVE。状态为 INACTIVE 的会话不能处理 Oracle 的 SQL 调用。

5.7.2 终止处于激活状态的会话

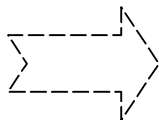
终止某个用户会话时，如果该会话正在处理某个事务（处于 ACTIVE 状态），那么事务将被回滚并且用户会立即收到如下消息：

```
ORA-00028: your session has been killed
```

收到 ORA-00028 消息后，如果用户在重新连接到数据库之前提交了另外的语句，那么 Oracle 将返回如下消息：

```
ORA-01012: not logged on
```

如果处于激活状态会话不能被终止（它正在执行网络 I/O 或回滚某个事务时），那么只有到操作完成后，才能终止该会话。在这种情况下，会话一直占有所有的资源直到会话被



终止。另外，利用 ALTER SYSTEM 语句来终止会话需要等待 60 秒的终止过程。如果不能被终止的操作执行超过 1 分钟，那么 ALTER SYSTEM 语句的发布者将收到一条消息，这条消息指出会话被标识为“已被终止”。在 V\$SESSION 视图中，标识为“已被终止”的会话将具有 KILLED 的状态和一个不是 PSEUDO 的服务器。

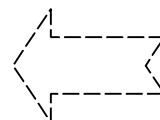
5.7.3 终止处于非激活状态的会话

终止某个用户会话时，如果该会话不在处理 Oracle 的 SQL 调用（会话的状态为 INACTIVE），那么将不会立即返回 ORA-00028 消息。直到用户后来试图使用已经终止的会话时才返回这条消息。

当一个处于非激活状态的会话被终止时，V\$SESSION 视图中的 STATUS 为 KILLED。在用户试图再次使用这个会话以及收到 ORA-00028 消息后，将从 V\$SESSION 视图中删除有关这个已经终止的会话的内容。

在下面的例子中，INACTIVE 状态的会话将被终止。首先，查询 V\$SESSION 视图来识别出会话的 SID 和 SERIAL#，然后终止会话。

```
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.
```



第 6 章 管理控制文件

本章解释如何创建和维护数据库的控制文件，包括以下内容：

- 什么是控制文件
- 控制文件的准则
- 创建控制文件
- 在创建控制文件之后遇到的问题及其解决办法
- 备份控制文件
- 使用当前副本恢复控制文件
- 取消控制文件
- 显示控制文件信息

6.1 什么是控制文件

每个 Oracle 数据库都有一个控制文件。控制文件是一个小型的二进制文件，它可以记录数据库的物理结构，其中包含以下内容：

- 数据库名称
- 相关的数据文件和联机重做日志文件的名称和位置
- 数据库创建的时标
- 当前日志的序号
- 检验点信息

无论何时打开数据库，控制文件必须能够由 Oracle 数据库服务器写入内容。没有控制文件，数据库就不能装载，并且恢复起来很困难。

Oracle 数据库的控制文件是在数据库创建的同时创建的。默认情况下，在数据库创建期间至少有一个控制文件副本。在某些操作系统中，默认情况是创建多个副本。应该在数据库创建期间创建两个或多个控制文件的副本。此后，如果丢失了控制文件或者希望改变控制文件中的特殊设置，也可能需要创建更多的控制文件。

6.2 控制文件的准则

这一节介绍管理数据库控制文件所采用的准则，包含以下内容：

- 提供控制文件的文件名
- 在不同磁盘上的复用控制文件
- 适当放置控制文件
- 备份控制文件
- 管理控制文件的大小

6.2.1 提供控制文件的文件名

使用数据库初始化参数文件中的初始化参数 `CONTROL_FILES` 可以指定控制文件名。实例启动过程将识别和打开所有列出的文件。实例将写入并维护整个数据库操作过程中所有列出的控制文件。

在数据库创建之前，如果没有指定 `CONTROL_FILES` 参数，并且没有使用 Oracle Managed Files (Oracle 管理文件) 特性，那么 Oracle 将创建一个控制文件并使用一个默认的文件名。在不同的操作系统中，控制文件的默认文件名也会有所不同。

6.2.2 在不同磁盘上的复用控制文件

每个 Oracle 数据库都应该至少有两个控制文件，它们分别存储在不同的磁盘上。如果因为磁盘失败的原因导致控制文件损坏，那么就必須关闭相关的实例。一旦修理好磁盘驱动器，可以使用另一个磁盘上保存完好的控制文件的副本来恢复损坏的控制文件，并且可以重新启动实例。在这种情况下，不需要任何形式的介质恢复。

下面的内容描述了复用控制文件的行为：

- 依据数据库初始化参数文件中的初始化参数 `CONTROL_FILES` 列出的所有文件名，Oracle 会将相应内容写入到所有这些控制文件中。
- 初始化参数 `CONTROL_FILES` 中列出的第一个文件是 Oracle 数据库服务器在数据库运行期间惟一可以读取的文件。
- 在整个数据库运行期间，如果任何一个控制文件变为不可用，那么实例就不能再继续运行，而且应该中止这个实例。

注意：Oracle 强烈建议数据库至少要有两个控制文件，并且它们各自位于单独的磁盘上。

6.2.3 适当放置控制文件

正像已经建议的那样，控制文件的每个副本都应该存储在不同的磁盘驱动器上。如果联机重做日志是多路复用的，有一个很实用的办法，就是将控制文件的副本存储在每个存储有联机重做日志组成员的磁盘驱动器上。通过在这些地方存储控制文件，可以尽量减少在一个磁盘的失败中丢失所有控制文件和所有联机重做日志组的风险。

6.2.4 备份控制文件

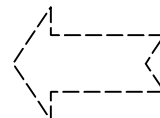
备份控制文件非常重要。这点建议是非常正确的，尤其在改变了数据库的物理结构之后。这类结构改变包括：

- 添加、取消或重命名数据文件
- 添加或撤消表空间，或者更改表空间的读写状态
- 添加或取消重做日志文件

备份控制文件的方法将在本章的“备份控制文件”一节中加以讨论。

6.2.5 管理控制文件的大小

控制文件大小的主要决定权在于设置的 `MAXDATAFILES`，`MAXLOGFILES`，



MAXLOGMEMBERS, MAXLOGHISTROY 参数以及在创建相关数据库的 CREATE DATABASE 语句中的 MAXINSTANCES 参数的值。增大这些参数的值将增大相关数据库的控制文件的大小。

6.3 创建控制文件

这一节描述创建控制文件的方法, 包含以下一些内容:

- 创建初始控制文件
- 创建额外的副本、重命名和重定位控制文件
- 创建新的控制文件

6.3.1 创建初始控制文件

当发布 CREATE DATABASE 语句时, 会创建 Oracle 数据库的初始控制文件。控制文件的名称是由数据库创建期间使用的初始化参数文件中的 CONTROL_FILES 参数指定的。CONTROL_FILES 参数指定的文件名应该是完整的, 并且这个名称应该因操作系统而异。下面举一个例子, 说明初始化参数 CONTROL_FILES 是如何指定初始控制文件的名称的:

```
CONTROL_FILES = (/u01/oracle/prod/control01.ctl,  
                /u02/oracle/prod/control02.ctl,  
                /u03/oracle/prod/control03.ctl)
```

在数据库创建期间, 如果当前已经存在与指定名称同名的文件, 那么必须在 CREATE DATABASE 语句中指定 CONTROLFILE REUSE 子句。而且, 如果原来的那个控制文件的大小与新的控制文件的 SIZE 参数设定值有差异, 就不能使用 REUSE 选项。

控制文件大小在不同版本的 Oracle 中可能会有所变化, 同样, 控制文件中指定的文件数目也会有所不同。MAXLOGFILES, MAXLOGMEMBERS, MAXLOGHISTROY, MAXDATAFILES 和 MAXINSTANCES 这些配置参数将会影响控制文件的大小。

之后可以改变初始化参数 CONTROL_FILES 的值, 以便添加更多的控制文件或者改变现有控制文件的名称或位置。

6.3.2 创建额外的副本、重命名和重定位控制文件

将某个现有文件备份到新的位置并将文件的名称添加到控制文件列表中, 这样就可以添加一个新的控制文件。同样地, 通过将文件备份为新名称或备份到新位置并在控制文件列表中更改文件名的方法, 可以重新命名现有的控制文件。在这两种情形下, 都要确保在整个过程中不要改变控制文件的内容, 并且在备份控制文件之前也不要关闭实例。

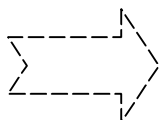
如果要多路复用或移动当前控制文件的另外的副本, 按如下步骤操作:

①关闭数据库。

②使用操作系统命令, 将现有控制文件备份到另外一个位置。

③编辑数据库初始化参数文件中的初始化参数 CONTROL_FILES, 进而添加新的控制文件的名称, 或者更改现有控制文件的文件名。

④重新启动数据库。



6.3.3 创建新的控制文件

这一节讨论何时以及如何创建新的控制文件。

6.3.3.1 何时创建新的控制文件

在以下情况下有必要创建新的控制文件：

- 所有数据库的控制文件都受到永久性的损坏，并且还没有任何控制文件的备份。
- 希望改变其中某个数据库参数的永久性设置，这些参数最初是在 CREATE DATABASE 语句中指定的。这些设置包括数据库的名称和以下一些参数的设置：MAXLOGFILES, MAXLOGMEMBERS, MAXLOGHISTORY, MAXDATAFILES 和 MAXINSTANCES。

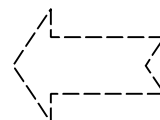
例如，在分布式环境中，如果数据库的名称与另一个数据库的名称产生冲突，就需要改变这个数据库的名称。或者，再举另外一个例子，如果最初设置 MAXLOGFILES 参数的值太低，可以更改这个参数的值。

6.3.3.2 CREATE CONTROLFILE 语句

使用 CREATE CONTROLFILE 语句可以为数据库创建一个新的控制文件。下面的代码将为 prod 数据库（就是先前那个使用不同的数据库名称的数据库）创建一个新的控制文件：

```
CREATE CONTROLFILE
SET DATABASE prod
LOGFILE GROUP 1 ('/u01/oracle/prod/redo01_01.log',
'/u01/oracle/prod/redo01_02.log'),
GROUP 2 ('/u01/oracle/prod/redo02_01.log',
'/u01/oracle/prod/redo02_02.log'),
GROUP 3 ('/u01/oracle/prod/redo03_01.log',
'/u01/oracle/prod/redo03_02.log')
NORESETLOGS
DATAFILE '/u01/oracle/prod/system01.dbf' SIZE 3M,
'/u01/oracle/prod/rbs01.dbs' SIZE 5M,
'/u01/oracle/prod/users01.dbs' SIZE 5M,
'/u01/oracle/prod/temp01.dbs' SIZE 5M
MAXLOGFILES 50
MAXLOGMEMBERS 3
MAXDATAFILES 200
MAXINSTANCES 6
ARCHIVELOG;
```

警告：CREATE CONTROLFILE 语句具有潜在损坏数据文件和联机重做日志文件的作用。省略文件名会引起文件中数据的丢失，或者失去访问完整数据库的能力。当使用这条语句时要格外小心，并且一定要遵循“创建新控制文件的步骤”一节中介绍的步骤说明。



6.3.3.3 创建新控制文件的步骤

完成以下这些步骤就可以创建一个新的控制文件。

①制作一个包含数据库的所有数据文件和联机重做日志文件的列表。

本章“备份控制文件”一节中讨论了控制文件的备份问题，如果按照建议的步骤，就具有数据文件和联机重做日志文件的列表，这个列表反映了当前数据库的结构。但是，如果没有这样的列表，可以执行以下语句来产生一个列表。

```
SELECT MEMBER FROM V$LOGFILE;
SELECT NAME FROM V$DATAFILE;
SELECT VALUE FROM V$PARAMETER WHERE NAME = 'CONTROL_FILES';
```

如果没有这样的列表而且控制文件又已经遭到损坏，那么将不能打开数据库，试试定位构成数据库的所有数据文件和联机重做日志文件，也许可以补救。一旦新的控制文件已经创建，而且在第5步中又没有指定任何文件，那么这些文件将不可恢复。此外，如果省略掉组成 SYSTEM 表空间的任何一个文件，将不能恢复数据库。

②关闭数据库。

如果数据库处于打开状态，尽可能采取正常模式关闭数据库。使用 IMMEDIATE 或 ABORT 选项作为最后的关闭手段。

③备份数据库的所有数据文件和联机重做日志文件。

④启动一个新的实例，但是不要装载或打开数据库：

```
STARTUP NOMOUNT;
```

⑤使用 CREATE CONTROLFILE 语句为数据库创建一个新的控制文件。

当创建新的控制文件的时候，如果除了控制文件外还丢失了某些联机重做日志组，那么可以选择使用 RESETLOGS 选项。在这种情况下，需要恢复丢失的重做日志（第8步）。如果重命名了数据库，也必须指定 RESETLOGS 选项。否则，选择 NORESETLOGS 选项。

⑥在离线存储设备上存储新控制文件的备份。

⑦为数据库编辑 CONTROL_FILES 初始化参数，以便指出现在所有控制文件都是第5步创建的数据库（不包括备份控制文件）的一部分。如果重新命名数据库，可以编辑 DB_NAME 参数来指定新的数据库名称。

⑧如果有必要可以恢复数据库。如果不需要恢复数据库，可以直接跳到第9步。

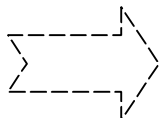
如果作为恢复过程的一部分正在创建控制文件，就可以对数据库进行恢复。如果新的控制文件是使用 NORESETLOGS 选项创建的（第5步），就可以使用完整的、封闭式数据库恢复过程来恢复数据库。

如果新的控制文件是使用 RESETLOGS 选项创建的，就必须指定 USING BACKUP CONTROL FILE。如果丢失了联机或存档的重做日志或者数据文件，可以使用这些过程来恢复这些文件。

⑨使用以下方法中的一种，可以打开数据库：

- 如果没有执行恢复过程，或者执行了第8步介绍的完整的封闭式数据库恢复过程，就可以以正常模式打开数据库。

```
ALTER DATABASE OPEN;
```



- 创建控制文件时如果指定了 RESETLOGS，就可以使用指明 RESETLOGS 的 ALTER DATABASE 语句。

```
ALTER DATABASE OPEN RESETLOGS;
```

按以上步骤操作，现在数据库就打开了，并且可以使用这个数据库。

6.4 在创建控制文件之后遇到的问题及其解决办法

在发布 CREATE CONTROLFILE 语句后，可能会遇到一些常见错误。这一节介绍控制文件使用中最常见的一些错误，主要包含以下内容：

- 检查丢失或多余的文件
- 在 CREATE CONTROLFILE 过程中处理错误

6.4.1 检查丢失或多余的文件

在创建新的控制文件并且使用它打开数据库之后，如果 Oracle 检查出数据字典和控制文件之间存在矛盾，比如数据字典包含了某个数据文件，而控制文件中没有列出这个数据文件，此时就可以检查报警文件来查找错误。

如果数据字典中存在某个数据文件，而新的控制文件中没有列出这个数据文件，那么 Oracle 将在控制文件中创建一个名为 MISSING nnn （这里的 nnn 是十进制的文件编号）的占位符入口。控制文件中的 MISSING nnn 作为脱机并需要介质恢复的标志。

通过重命名 MISSING nnn 可以访问到与 MISSING nnn 相对应的真正的数据文件，只有在数据文件是只读的或正常脱机时，MISSING nnn 才指向这个数据文件。另一方面，如果 MISSING nnn 对应的数据文件不是只读的或正常脱机的，那么重命名操作不能使得数据文件可以被访问，因为数据文件需要介质恢复，而 RESETLOGS 的结果排除了介质恢复的可能。在这种情况下，必须撤消包含数据文件的表空间。

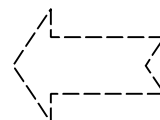
相反，如果控制文件中显示的数据文件并不在数据字典中，Oracle 会从新的控制文件中删除对数据文件的引用。在以上两种情况下，Oracle 会在 alert.log 文件中包含一条解释性消息，以便让用户知道报警文件发现了什么问题。

6.4.2 在 CREATE CONTROLFILE 过程中处理错误

在创建新的控制文件后，当用户试图装载和打开数据库时，如果 Oracle 发送了一条错误消息（通常的错误消息是 ORA-01173，ORA-01176，ORA-01177，ORA-01215，ORA-01216），那么最有可能的原因是 CREATE CONTROLFILE 语句中遗漏了一个文件，或者其中包含的某个文件在控制文件中并没有列出。在这种情况下，应该恢复本章前面介绍的第3步中的备份文件，并且使用正确的文件名重复从第4步开始的过程。

6.5 备份控制文件

使用 ALTER DATABASE BACKUP CONTROLFILE 语句备份控制文件。有两种可以选择的办法：



①使用以下语句将控制文件备份为二进制文件（现有控制文件的复制品）：

```
ALTER DATABASE BACKUP CONTROLFILE TO '/oracle/backup/control.bkp';
```

②制作以后可用于重新创建控制文件的 SQL 语句：

```
ALTER DATABASE BACKUP CONTROLFILE TO TRACE;
```

这个命令将 SQL 脚本写入数据库的跟踪文件，在这个跟踪文件中可以找到这个 SQL 脚本并对脚本进行编辑，以便重新产生控制文件。

6.6 使用当前副本恢复控制文件

这一节提出了恢复控制文件的方法，利用这些方法可以从当前备份或多路复用的副本中恢复控制文件。

6.6.1 使用控制文件的副本对受到损坏的控制文件进行恢复

这个过程假设 CONTROL_FILES 参数指定的某个控制文件受到损坏，该控制文件的目录仍然可以访问，并且有这个控制文件的一个多路复用副本。

①关闭实例，使用操作系统命令用一个完好的副本重写损坏的控制文件：

```
% cp /u01/oracle/prod/control03.ctl /u01/oracle/prod/control02.ctl;
```

②启动 SQL*Plus 并打开数据库：

```
SQL> STARTUP
```

6.6.2 使用控制文件的副本对永久性介质失败进行恢复

这个过程假设因为永久性介质失败的缘故不能访问 CONTROL_FILES 参数指定的某个控制文件，并且有这个控制文件的一个多路复用副本。

①关闭实例，使用操作系统命令将控制文件的当前副本复制到一个新的可访问的位置：

```
% cp /u01/oracle/prod/control01.ctl /u04/oracle/prod/control03.ctl;
```

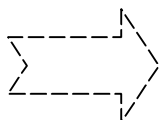
②编辑初始化参数文件中的 CONTROL_FILES 参数，以使用新的位置代替原来的被损坏的位置：

```
CONTROL_FILES = (/u01/oracle/prod/control01.ctl,  
                /u02/oracle/prod/control02.ctl,  
                /u04/oracle/prod/control03.ctl)
```

③启动 SQL*Plus 并打开数据库：

```
SQL> STARTUP
```

在拥有多个多路复用控制文件并且必须在最少时间内得到数据库的情况下，可以通过编辑 CONTROL_FILES 初始化参数来删除损坏了的控制文件并立即重新启动数据库。然后，在编辑初始化参数 CONTROL_FILES 以包含恢复的控制文件之后，可以执行已经损坏的控制文件的重建任务以及在以后的任何时间关闭和重新启动数据库。



6.7 删除控制文件

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再适合时，可以删除控制文件。记住数据库必须一直拥有至少两个控制文件。

①关闭数据库。

②编辑数据库初始化参数文件中的初始化参数 `CONTROL_FILES`，以便删除旧的控制文件的名称。

③重新启动数据库。

注意：这项操作并不能从磁盘中物理删除不想要的控制文件。在从数据库中删除控制文件之后，可以使用操作系统命令来删除不需要的文件。

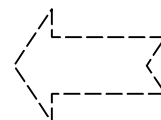
6.8 显示控制文件信息

以下视图用来显示有关控制文件的信息：

视图	说明
V\$CONTROLFILE	列出控制文件的名称
V\$CONTROLFILE_RECORD_SECTION	显示有关控制文件记录段的信息
V\$PARAMETER	显示初始化参数 <code>CONTROL_FILES</code> 指定的控制文件的名称

以下这个例子列出了控制文件的名称。

```
SQL> SELECT NAME FROM V$CONTROLFILE;
NAME
-----
/u01/oracle/prod/control01.ctl
/u02/oracle/prod/control02.ctl
/u03/oracle/prod/control03.ctl
```



第 7 章 管理联机重做日志

这一章主要解释如何管理联机重做日志，包含以下内容：

- 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及其成员
- 重新定位和重新命名联机重做日志成员
- 取消联机重做日志组及其成员
- 强制实行日志切换
- 校验重做日志文件中的块
- 清除联机重做日志文件
- 查看联机重做日志信息

7.1 什么是联机重做日志

在恢复操作中最重要的结构就是联机重做日志，它包括两个或更多个预先分配好的文件，在这些文件中存储自文件产生以来对数据库所做的所有更改。Oracle 数据库中的每一个实例都有一个相关联的联机重做日志，目的是保护数据库以防实例失败。

7.1.1 重做线程

每个数据库实例都有它自己的联机重做日志组。这些联机重做日志组，不管是不是多路复用，都称之为一个联机重做的实例线程。在典型的配置中，一个 Oracle 数据库只能被一个数据库实例访问，所以只会出现一个线程。但是，当运行 Oracle Real Application Clusters 时，同时可以有两个或更多个实例访问一个数据库，而且每个实例都有它自己的线程。

本章主要描述在没有使用 Oracle Real Application Clusters 特性的情况下如何配置和管理联机重做日志。因此，在所有讨论内容以及示例中，线程数都假设为 1。

7.1.2 联机重做日志目录

联机重做日志文件中填满了重做记录。重做记录又称为重做条目，它由一组修改向量组成，其中每个修改向量都是用来描述对数据库中的某个块所做的一种修改。例如，如果改变了雇员表中的薪水值，就会生成包含修改向量的重做记录，该修改向量描述了对表中这个数据段块、回滚段数据块以及回滚段的事务表的修改。

重做条目记录了可以用来对数据库重新构造的所有修改的数据，包括回滚段。因此，联机重做日志同样也会保护回滚数据。当使用重做数据恢复数据库时，Oracle 将读取重做记录中的修改向量并且将这些修改用于相关的块中。

重做记录采用循环方式在 SGA 的重做日志缓冲区中进行缓冲，并且由 Oracle 后台进程日志写入程序 (LGWR) 写入到其中某个联机重做日志文件。无论何时提交事务，LGWR 都会将事务的重做记录从 SGA 的重做日志缓冲区写入到某个联机重做日志文件中，而且为

提交的每个事务都分配一个识别重做记录的系统修改号码（SCN）。只有当与已知事务相关的所有重做记录都安全地位于联机日志所在的磁盘上时，用户进程才被告之事务已经提交。

在相应的事务提交之前，重做记录也可以写入到联机重做日志文件中。如果重做日志缓冲区已经填满，或者提交了另一个事务，那么即使某些重做记录还没有提交，LGWR 也会将重做日志缓冲区中的所有重做日志条目刷新到某个联机重做日志文件中。如果需要，Oracle 可以回滚这些修改。

7.1.3 Oracle 如何将内容写到联机重做日志

数据库的联机重做日志包括两个或更多个联机重做日志文件。Oracle 需要至少两个这种文件，以便确保其中一个一直用于写入，另一个用于存档（如果是在 ARCHIVELOG 模式）。

LGWR 以循环方式填写联机重做日志文件。当当前的联机重做日志文件填满时，LGWR 会开始向下一个联机重做日志文件填写。当最后一个可用的联机重做日志文件也被填满时，LGWR 会返回到第一个联机重做日志文件并进行填写，如此循环。图 7-1 举例说明了这种循环填写联机重做日志文件的方式。每一行旁边的号码表示 LGWR 填写每个联机重做日志文件的顺序。

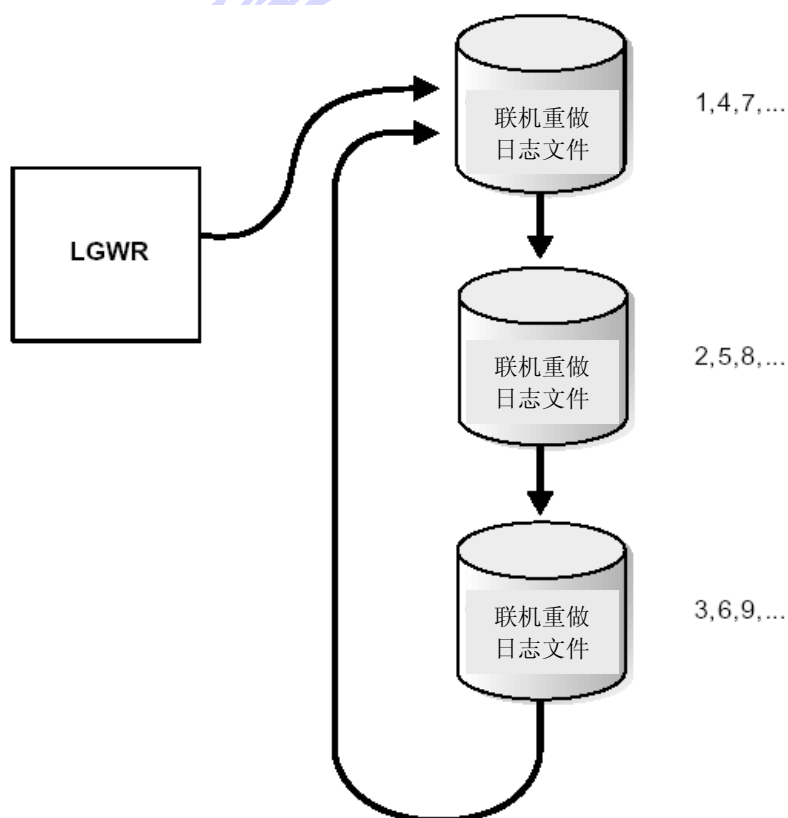
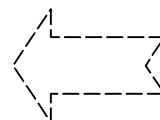


图 7-1 LGWR 循环使用联机重做日志文件



LGWR 是否可以重复利用填满的联机重做日志文件要依赖于是否能够存档。

- 如果不能进行存档 (NOARCHIVELOG 模式), 填满的联机重做日志文件只有在修改记录已经写入到数据文件时, 才可以重新得以利用。
- 如果能存档 (ARCHIVELOG 模式), 填满的联机重做日志文件可以在修改记录已经写入到数据文件并且联机重做日志文件已经存档时, 重新得以利用。

7.1.3.1 活动的 (当前的) 和非活动的联机重做日志文件

在任何时候, Oracle 都只使用其中一个联机重做日志文件来存储来自于重做日志缓冲区的重做记录。LGWR 当前写入的联机重做日志文件称为当前的联机重做日志文件。

实例恢复所需的联机重做日志文件称为活动的联机重做日志文件。实例恢复不需要的联机重做日志文件称为非活动的联机重做日志文件。

如果已经能够进行存档 (ARCHIVELOG 模式), Oracle 就不能重新使用或重写活动的联机重做日志文件, 直到 ARCn 存储了这个文件的内容。如果不能进行存档 (NOARCHIVELOG 模式), 当最后一个联机重做日志文件被填满时, 将通过重写的方法继续使用第一个可用的活动的联机重做日志文件。

7.1.3.2 日志切换和日志顺序号

Oracle 结束对某个联机重做日志文件的填写而开始填写另外一个联机重做日志文件的那一刻称为日志切换。通常, 日志切换发生在当前联机重做日志文件已经完全填满而且必须继续填写下一个联机重做日志文件的时候。然而, 可以用基于时间的方式指定日志切换发生的时刻, 此时不用理会当前联机重做日志文件是否已经完全填满。也可以手动强制执行日志切换。

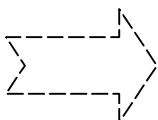
在每次发生日志切换以及 LGWR 开始进行填写时, Oracle 都会为每个联机重做日志文件分配一个新的日志顺序号。如果 Oracle 存档了联机重做日志文件, 那么存档的日志将保留它的日志顺序号。Oracle 将为循环重用的联机重做日志文件分配下一个可用的日志顺序号。

每个联机或存档的重做日志文件通过它的日志顺序号惟一标识。在崩溃、实例或介质恢复的过程中, 通过使用所需的存档和联机重做日志文件的日志顺序号, Oracle 将以升序的日志顺序号顺序正确应用重做日志文件。

7.2 规划联机重做日志

这一节描述在配置数据库实例的联机重做日志时应该考虑采纳的准则, 包含以下内容:

- 多路复用联机重做日志文件
- 将联机重做日志成员放置在不同的磁盘上
- 设置联机重做日志成员的大小
- 选择联机重做日志文件的号码
- 控制存档延迟



7.2.1 多路复用联机重做日志文件

Oracle 提供了多路复用联机重做日志文件的能力，以免损坏实例的联机重做日志文件。当多路复用联机重做日志文件时，LGWR 将同一个重做日志信息同时写到多个同样的联机重做日志文件中。这样就消除了因为一个重做日志文件的某种原因而导致失败的可能。

注意：Oracle 建议多路复用重做日志文件。如果需要进行恢复，那么日志文件数据的丢失将是一种灾难性的行为，会使恢复工作无法开展。

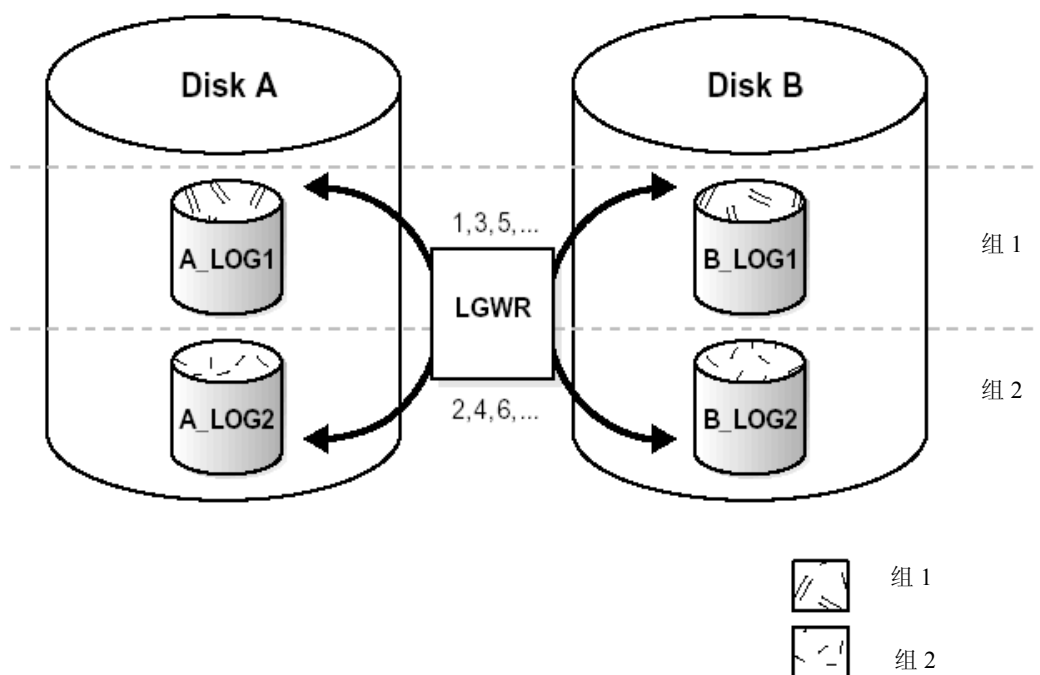


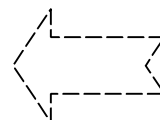
图 7-2 多路复用联机重做日志文件

对应的联机重做日志文件称为组。组中的每个联机重做日志文件称为成员。在图 7-2 中，A_LOG1 和 B_LOG1 是 1 组的两个成员，A_LOG2 和 B_LOG2 是 2 组的两个成员。组中的每个成员必须有相同的大小。

注意组中的每个成员都同时是活动的，或者由 LGWR 同时进行填写，如同 LGWR 分配的同个日志顺序号所表示的那样。在图 7-2 中，首先 LGWR 填写 A_LOG1 和 B_LOG1，然后填写 A_LOG2 和 B_LOG2。LGWR 绝不会同时填写不同组中的成员（例如，A_LOG1 和 B_LOG2）。

7.2.1.1 对联机重做日志失败作出响应

无论何时当 LGWR 不能填写组中的某一个成员时，Oracle 都会将这个成员标记为 INVALID，并且还往 LGWR 跟踪文件和数据库报警文件中写入错误消息，以指出这个不可访问的文件可能出现的问题。当某个联机重做日志成员不可用时，LGWR 会依据不可用的原因做出不同的反应。

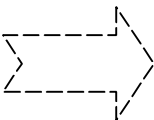


如果	那么
LGWR 可以成功地填写某个组中的至少一个成员	可以按正常过程进行填写。LGWR 仅仅写到组中的可利用成员并且忽略掉不可用的成员。
因为组需要存档的原因在日志切换时 LGWR 不能访问下一个组	数据库操作临时中断，直到组可以用或者组已经存档。
因为介质失败的缘故在日志切换时 LGWR 不能访问下一个组	Oracle 返回一条错误消息并且关闭数据库实例。在这种情况下，需要在数据库上执行介质恢复，以便从联机重做日志文件的丢失中恢复过来。 如果数据库的检验点已经移到丢失的重做日志那边，那么就不需要介质恢复，原因是 Oracle 已经将重做日志中记录的数据保存到数据文件中了。这时仅仅取消不可访问的重做日志组就可以了。如果 Oracle 没有存档损坏的日志，那么在取消该日志之前使用 ALTER DATABASE CLEAR UNARCHIVED LOG 语句禁止存档。
在 LGWR 正在进行填写时，如果突然不能访问组中的所有成员	Oracle 将返回一条错误消息并立即关闭数据库实例。在这种情况下，需要执行介质恢复。如果包含该日志的介质确实丢失了——例如，如果无意中关掉了存储该日志的驱动器——就不必进行介质恢复了。此时，只需要重新开启这个驱动器并让 Oracle 执行实例恢复即可。

7.2.1.2 合法和非法的配置

要避免因为单个联机重做日志而导致的失败，多路复用联机重做日志是一个解决方法，它具有理想的对称性：联机重做日志的所有组都有相同数目的成员。然而，Oracle 不需要多路复用联机重做日志这么对称。例如，一个组可以有一个成员，而另一个组可以有两个成员。这种配置避免了磁盘失败，而这种磁盘失败会暂时影响某些联机重做日志成员但是不影响其他的成员。

对于实例的联机重做日志的惟一要求是它至少要有两个组。图 7-3 说明了合法和非法的多路复用联机重做日志配置。第二种配置是非法的，因为它只有一个组。



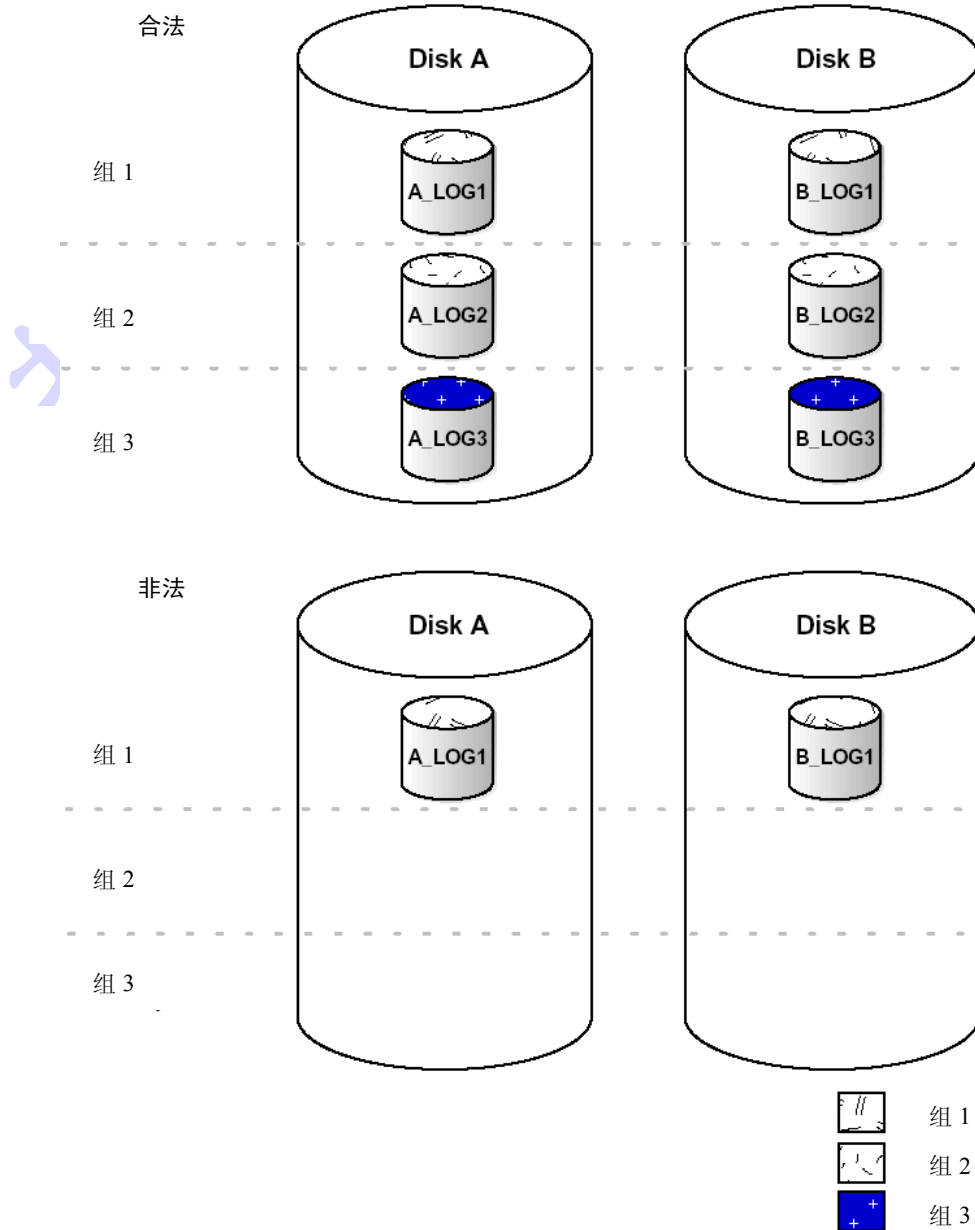


图 7-3 合法和非法的多路复用联机重做日志配置

7.2.2 将联机重做日志成员放置在不同的磁盘上

在设置多路复用联机重做日志时，将联机重做日志成员放置在不同的磁盘上。如果某个磁盘失败了，那么只有组中的一个成员变得不可以被 LGWR 利用，LGWR 仍然能够访问其他的成员，所以实例仍可以继续。

如果存档了重做日志，可以将联机重做日志成员散布在很多磁盘上，以消除 LGWR 和 ARC_n 后台进程对联机重做日志成员的争夺。例如，如果有两组多路复用联机重做日志成

员，可以将每个成员放置在不同的磁盘上，并将存档目的地设置在第 5 个磁盘上。这样的话，LGWR（填写成员）和 ARCn（读取成员）就不会产生争夺成员的现象了。

数据文件和联机重做日志文件也应该放在不同的磁盘上，以减少写数据块和写重做记录之间出现的竞争。

7.2.3 设置联机重做日志成员的大小

设置联机重做日志文件的大小时，要考虑将来是否需要存档重做日志。应该规定联机重做日志文件的大小，以便可以将填满的组存档到脱机存储介质（比如磁带或磁盘）的某个单元中，而这些介质上要至少留出一定的未用空间。例如，磁带上仅存储了一个填满的联机重做日志组，并且磁带还有 49% 的未用存储空间。此时，最好稍稍降低联机重做日志文件的大小，以便每个磁带都可以存档两个日志组。

采用多路复用的联机重做日志组时，同一个组的所有成员必须拥有同样的大小。不同组的成员的大小可以不同。但是，组之间拥有不同大小的文件尺寸并未带来任何好处。如果检验点并未设置在两个日志切换之间，就需要使所有组具有相同的大小，以便确保检验点发生在普通的时间间隔内。

7.2.4 选择联机重做日志文件的数量

要为数据库实例的联机重做日志文件确定合适的数量，最好的方法是测试不同的配置效果。最佳配置是拥有最少的组，而且不妨碍 LGWR 向重做日志写入信息。

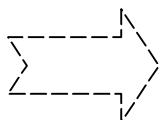
在某些情况下，数据库实例可能只需要两个组。在另外一些情况下，数据库实例可能还需要附加的组来确保 LGWR 一直都有可利用的循环组。在测试过程中，要确定当前的联机重做日志配置是否令人满意，最简单的方法是仔细检查 LGWR 跟踪文件和数据库报警文件的内容。如果其中的消息表明：因为检验点还没有完成或者组还没有存档的缘故，LGWR 不得不频繁地等待组，那么就可以添加组来解决问题。

在设置或更改实例的联机重做日志的配置之前，考虑使用参数来限制联机重做日志文件的数目。以下一些参数可以用来限制添加到数据库的联机重做日志文件的数目：

- 在 CREATE DATABASE 语句中使用的 MAXLOGFILES 参数，它可以为每个数据库指定联机重做日志文件的最大组数。组值的范围从 1 到 MAXLOGFILES。重写这个上限的惟一办法是重新创建数据库或数据库的控制文件。所以，在创建数据库之前，慎重考虑这个上限是非常重要的。如果 CREATE DATABASE 语句中没有指定 MAXLOGFILES，那么 Oracle 将使用操作系统指定的默认值。
- 在 CREATE DATABASE 语句中使用的 MAXLOGMEMBERS 参数，它可以为每个组指定成员的最大数目。和 MAXLOGFILES 参数一样，重写这个上限的惟一办法是重新创建数据库或数据库的控制文件。因此，在创建数据库之前，慎重考虑这个上限是非常重要的。如果 CREATE DATABASE 语句中没有指定 MAXLOGFILES，那么 Oracle 将使用操作系统指定的默认值。

7.2.5 控制存档延迟

可以强制所有有效的联机重做日志线程以时间的形式切换它们当前的日志。在现用/备



用配置中，通过存档现用站点的非当前日志并将这些日志传送到备用数据库中的方法，对备用数据库做些更改，从而可以使用备用数据库。这些应用到备用数据库的修改可以延迟发生在现用数据库上的改变。

因为备用数据库必须等待存储（存档到存档的重做日志中）现用数据库的联机重做日志中的变化以及将这些变化传送到备用数据库中，所以才会出现这个延迟。要控制或限制这个延迟，就需要设置 ARCHIVE_LAG_TARGET 初始化参数。对这个参数进行设置可以限制延迟多长时间。

7.2.5.1 设置 ARCHIVE_LAG_TARGET 初始化参数

当设置 ARCHIVE_LAG_TARGET 初始化参数的时候，会引起 Oracle 定期检查实例的当前联机重做日志。如果满足以下条件，实例将切换日志：

- 在 n 秒钟之前创建了当前日志，并且估计当前日志的存档时间为 m 秒钟（与当前日志中使用的重做块数目成比例），这里 $n+m$ 的值超过了 ARCHIVE_LAG_TARGET 初始化参数设定的值。
- 当前日志包含重做记录。

在 Oracle Real Application Clusters 环境中，如果其他线程落到了后面，那么实例也使这些线程进入切换状态和存档日志任务中。当簇中的某个实例比其他实例更加空闲时（正在运行的 Oracle Real Application Clusters 环境配置为两节点的现用/备用配置），这种做法特别有用。

初始化参数 ARCHIVE_LAG_TARGET 指定了一个时间指标，该时间指标表示在现用数据库关闭或崩溃事件中，重做备用需要多少秒的时间。这个参数还提供了上限时间，这个上限时间表示现用数据库的当前日志可以观测多长时间（单位为秒）。因为还需要考虑的存档时间，所以这个参数值并不是非常准确的日志切换时间。

下面的初始化参数设置将日志切换时间间隔设定为 30 分钟（典型值）。

```
ARCHIVE_LAG_TARGET = 1800
```

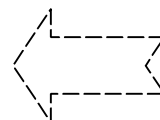
如果该参数的值为 0，则表示基于时间的日志切换功能被禁止。这是一个默认设置。

在一个空闲系统中，如果在很长一段时间都没有进行重做（例如，DBA 忘记关闭了数据库），那么 Oracle 将不会生成一堆空的存档日志。这是因为只有在当前日志包含重做时才会切换这个日志。

即使没有备用数据库，也可以设置初始化参数 ARCHIVE_LAG_TARGET。例如，可以特别指定 ARCHIVE_LAG_TARGET 参数来强制切换和存档日志。

ARCHIVE_LAG_TARGET 参数是一个动态参数，并且可以在 ALTER SYSTEM SET 语句中进行设置。

警告： 在 Oracle Real Application Clusters 环境下的所有实例中，ARCHIVE_LAG_TARGET 参数都必须设置成相同的值。如果不这样做将导致不可预料的行为并且后果非常糟糕。



7.2.5.2 影响 ARCHIVE_LAG_TARGET 参数设置的几个因素

当决定是否设置 ARCHIVE_LAG_TARGET 参数以及将该参数设置为何值时，考虑以下一些因素。

- 切换日志所消耗的系统开销
- 作为日志完整条件的结果，通常日志切换发生的频率是多少
- 备用数据库可以忍受多少重做损失

如果自然的日志切换经常发生，其频率高于指定的间隔，那么设置 ARCHIVE_LAG_TARGET 参数将不会有太大用处。但是，在产生重做的速率不规则的时候，设置的间隔将为每个当前日志覆盖的时间范围提供一个上限。如果数据库变成空闲的，并且重做记录停止生成，那么该时间间隔将强制日志切换，从而引起已经生成的所有重做记录的切换、存档并且可以使用备用数据库。

如果初始化参数 ARCHIVE_LAG_TARGET 设置为一个很低的值，那么会对性能产生负面影响。这种设置可以强制频繁切换日志。将该参数设置为一个合理的值，这样不会降低现用数据库的性能。

7.3 创建联机重做日志组及其成员

对数据库的联机重做日志进行规划，并且在数据库创建期间创建所有需要的联机重做日志文件的组和成员。然而，也会有一些情形需要额外创建另外的组或成员。例如，添加联机重做日志组可以校正重做日志组可用性的问题。

要创建新的联机重做日志组和成员，必须拥有 ALTER DATABASE 系统权限。一个数据库可以拥有多达 MAXLOGFILES 个组。

7.3.1 创建联机重做日志组

要创建联机重做日志文件的新组，使用带 ADD LOGFILE 子句的 SQL 语句 ALTER DATABASE 即可。

下面的语句向数据库添加了一个新的重做日志组：

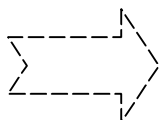
```
ALTER DATABASE ADD LOGFILE ('/oracle/dbs/log1c.rdo', '/oracle/dbs/log2c.rdo') SIZE 500K;
```

注意：新的日志成员使用完整的指定文件名来表示操作系统文件应该在哪里创建。否则，依据操作系统的不同，文件将在数据库服务器的默认目录或当前目录中创建。

也可以使用 GROUP 选项来指定标识组的编号：

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

使用组号可以更加方便地管理重做日志组的工作。但是，组号必须在 1 和 MAXLOGFILES 之间。不要跳跃式使用重做日志文件组号（也就是说，不要将组号编为 10, 20, 30, 等等），否则将耗费数据库控制文件中的空间。



7.3.2 创建联机重做日志成员

在某些情况下，也许不需要创建一个完整的联机重做日志文件组。某个组可能已经存在，但是它并不完整，原因是该组中的一个或多个成员已经被取消（例如，因为磁盘失败的缘故）。在这种情况下，可以给现有的这个组添加新的成员。

要为现有的某个组创建新的联机重做日志成员，使用带 `ADD LOG MEMBER` 参数的 SQL 语句 `ALTER DATABASE` 即可。下面的语句就为组号为 2 的组添加了一个新的重做日志成员：

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

注意必须指定文件名，但是可以不指定大小。新成员的大小是由组中的现有成员的大小决定的。

当使用 `ALTER DATABASE` 语句时，还可以通过在 `TO` 参数中指定组中所有其他成员的方法平来识别目标组，举例说明如下：

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

注意：完整地指定新的日志成员的文件名，以便指出应该在什么地方创建操作系统文件。否则，将根据操作系统的不同，在数据库服务器的默认目录或当前目录中创建这些文件。也许会注意到新日志成员的状态是 `INVALID`。这很正常，当首次使用日志成员时，该成员会变成活动的（标识处空白）。

7.4 重新定位和重新命名联机重做日志成员

可以使用操作系统命令重新定位联机重做日志，然后使用 `ALTER DATABASE` 语句使数据库知道联机重做日志的新名称（位置）。这个过程是非常必要的，例如，如果某些联机重做日志文件当前使用的磁盘将要被删除，或者如果数据文件和大量联机重做日志文件存储在同一个磁盘上并且将要被分开放置以减少对磁盘空间的竞争时，就需要这个过程。

要重新命名联机重做日志成员，必须拥有 `ALTER DATABASE` 系统权限。另外，也可能需要操作系统权限，以便将文件复制到理想的位置，以及用来打开和备份数据库的权限。

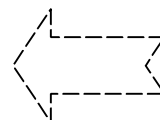
在重新定位重做日志之前，或者对数据库做出任何结构上的改变之前，需要完整地备份数据库以防在执行重新定位操作时出现问题。作为预防，在重新命名或重新定位一套联机重做日志文件之后，应立即备份数据库的控制文件。

按照如下步骤重新定位重做日志文件。同时用这个例子说明这些步骤的一些假设条件：

- 将日志文件定位在两个磁盘上：diska 和 diskb。
- 多路复用联机重做日志：其中一个组包含成员 /diska/logs/log1a.rdo 和 /diskb/logs/log1b.rdo，另一个组包含成员 /diska/logs/log2a.rdo 和 /diskb/logs/log2b.rdo。
- 位于 diska 的联机重做日志文件必须重新定位在 diskc 上。新的文件名将反映新的位置：/diskc/logs/log1c.rdo 和 /diskc/logs/log2c.rdo。

按照如下步骤重新命名联机重做日志成员。

①关闭数据库。



SHUTDOWN

②将联机重做日志文件复制到新的位置。

操作系统文件，比如联机重做日志成员，必须使用适当的操作系统命令进行复制。翻阅操作系统的某些专用文档来查看有关复制文件的详细内容。

注意：在没有退出 SQL*Plus 的情况下，通过使用 HOST 命令，可以执行操作系统命令来复制文件（或者执行其他的操作系统命令）。某些操作系统允许使用一个字符来代替 HOST。例如，在 UNIX 中可以使用 “!”。

```
mv /diska/logs/log1a.rdo /diskc/logs/log1c.rdo
```

```
mv /diska/logs/log2a.rdo /diskc/logs/log2c.rdo
```

③启动并装载数据库，但不打开数据库。

```
CONNECT / as SYSDBA
```

```
STARTUP MOUNT
```

④重新命名联机重做日志成员。

使用带 RENAME FILE 子句的 ALTER DATABASE 语句重新命名数据库的联机重做日志文件。

```
ALTER DATABASE RENAME FILE
```

```
'/diska/logs/log1a.rdo', '/diska/logs/log2a.rdo'
```

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

⑤打开数据库进行正常操作。

打开数据库时，对联机重做日志中的变更生效。

```
ALTER DATABASE OPEN;
```

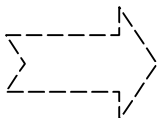
7.5 取消联机重做日志组及其成员

在某些情况下，也许希望取消联机重做日志的某个完整的组。例如，想减少实例的联机重做日志的组数。在另一种情况下，也许希望取消一个或多个特定的联机重做日志成员。例如，如果发生磁盘失败，就需要取消这个失败的磁盘上的所有联机重做日志文件，以便 Oracle 不会把某些内容写入到不可访问的文件中去。在其他情形下，可能会不再需要特别的联机重做日志文件。例如，某个文件可能存储在某个不恰当的位置上。

7.5.1 取消日志组

要取消某个联机重做日志组，就必须拥有 ALTER DATABASE 系统权限。在取消联机重做日志组之前，应该仔细斟酌以下的限制条件和预先警告：

- 不管组中要包含多少成员，实例需要至少两组联机重做日志文件。（一个组有一个或多个成员。）
- 只有当联机重做日志组的状态为非活动状态时，才可以取消这个组。如果需要取消当前组，那么首先要强制进行一个日志切换。
- 在取消联机重做日志组前要确保它已经存档（如果允许存档的话）。可以使用 V\$LOG 视图，查看是否发生过这种存档的操作，。



```
SELECT GROUP#, ARCHIVED, STATUS FROM V$LOG;
GROUP# ARC STATUS
```

```
-----
```

```
1 YES ACTIVE
2 NO CURRENT
3 YES INACTIVE
4 YES INACTIVE
```

用带 **DROP LOGFILE** 子句的 SQL 语句 **ALTER DATABASE** 取消联机重做日志组。

以下的语句取消了编号为 3 的重做日志组：

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

当从数据库中取消了联机重做日志组并且没有使用 Oracle 管理文件特性的时候，操作系统文件并没有从磁盘中删除掉。相反，此时更新了相关数据库的控制文件，以便从数据库结构中取消该组的成员。在取消联机重做日志组之后，要确定取消操作成功地完成，然后使用合适的操作系统命令来删除取消了的联机重做日志文件。

当使用 Oracle 管理文件时，将自动完成操作系统文件的清理工作。

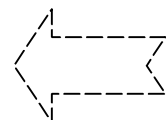
7.5.2 取消联机重做日志成员

要取消联机重做日志成员，必须拥有 **ALTER DATABASE** 系统权限。在取消联机重做日志成员之前，仔细斟酌以下的限制条件和预先警告：

- 取消联机重做日志文件是允许的，这样多路复用联机重做日志才可以暂时变得对称。例如，如果使用联机重做日志文件的多路复用组，就可以取消某个组的某个成员，即使所有其他的组都有两个成员，也可以这么做。但是，应该立即调整这种状况，以便所有组都至少有两个成员，以及减少由于单个成员的原因而导致联机重做日志失败的可能性。
- 不管组中包含多少成员，实例一直需要至少两个有效联机重做日志文件。（组有一个或多个成员。）如果要取消的成员是组的最后一个有效成员，那么就不能取消该成员，直到其他成员变为有效的为止。要查看重做日志文件的状态，使用 **V\$LOGFILE** 视图即可。如果 Oracle 不能访问重做日志文件，那么该文件将变成 **INVALID** 状态。如果 Oracle 查出重做日志文件不完整或不正确，那么该文件将变成 **STALE** 状态。当下次失效日志文件所属的组变成活动组时，该失效日志文件才会再次变成有效状态。
- 只有当要取消的联机重做日志成员属于活动组或当前组时，才可以取消它。如果想取消某个活动组的成员，首先需要强制进行日志切换。
- 在取消联机重做日志成员之前，要确定该成员所属的组是经过存档的（如果存档允许的话）。如果要查看组是否存档，可以使用 **V\$LOG** 视图。

要取消指定的非活动的联机重做日志成员，使用带 **DROP LOGFILE MEMBER** 子句的 **ALTER DATABASE** 语句即可。

下面的语句将取消重做日志 `/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中取消时，操作系统文件并未从磁盘中删除。相反，此时更新了与数据库相关的控制文件，以便将成员从数据库结构中取消掉。在取消联机重做日志文件后，要确定取消操作成功完成，然后使用合适的操作系统命令来删除取消了的联机重做日志文件。

要取消活动组的成员，首先必须强制进行日志切换。

7.6 强制实施日志切换

当 LGWR 停止向某个联机重做日志组写入而开始向另一个联机重做日志组写入时，将发生日志切换。默认情况下，当当前的联机重做日志组填满时将自动发生日志切换。

可以强制实行日志切换，使当前活动组变成非活动的以便对该组实施联机重做日志的维护操作。例如，如果想取消当前的活动组，但是并不能立即这么做，必须要等到该组变为非活动状态时才可以取消。在组成员完整地填充之前，如果需要在某个指定的时刻存档当前的活动组，那么也希望强制实施日志切换。因为填充大型的联机重做日志文件需要很长的时间，所以在配置这种文件时，日志切换选项非常有用。

要强制实施日志切换，必须拥有 ALTER SYSTEM 权限。如果有这种权限，使用带 SWITCH LOGFILE 子句的 ALTER SYSTEM 语句就可以强制实施日志切换。

下面的语句强制实施日志切换：

```
ALTER SYSTEM SWITCH LOGFILE;
```

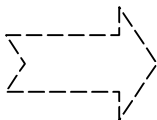
7.7 校验重做日志文件中的块

可以通过配置 Oracle，从而使用校验和来校验重做日志文件中的块。如果设置初始化参数 DB_BLOCK_CHECKSUM 为 TRUE，就可以校验写到磁盘上的所有 Oracle 数据库块，包括重做日志块。初始化参数 DB_BLOCK_CHECKSUM 的默认值为 FALSE。

如果允许块校验，Oracle 将为每个写到当前日志的重做日志块计算校验和。Oracle 将该校验和写到块的头部。Oracle 使用校验和来检测重做日志块中的损坏。当 Oracle 将块写入某个存档日志文件的时候，以及在恢复过程中从存档日志中读取块的时候，Oracle 都会校验重做日志块。

当试图存档时，如果 Oracle 在重做日志块中检测出存在损坏内容，那么系统将试着从组中的另一个成员中读取块。如果重做日志组中的所有成员中的块都受到损坏，那么将不能进行存档操作。

注意：如果 DB_BLOCK_CHECKSUM 参数设置允许校验，会有一些系统开销，并且数据库性能也会降低。监视数据库的性能，以便决定使用数据块校验和来检测存在损坏所付出的代价对性能造成的影响是否过重。



7.8 清除联机重做日志文件

当打开数据库时，联机重做日志文件也许已经受到损坏，并且因为不能继续存档最终停止数据库的各项活动。在这种情形下，可以不用关闭数据库而使用 ALTER DATABASE CLEAR LOGFILE 语句重新初始化联机重做日志文件。

下面的语句将清除组号为 3 的重做日志组中的日志文件：

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

要使用这条语句必须克服两种情况，在这两种情况下不可能取消重做日志：

- 如果仅有两个日志组
- 受到损坏的重做日志文件属于当前组

如果受到损坏的重做日志文件还没有存档，可以在语句中使用 UNARCHIVED 关键字。

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

这条语句将清除损坏了的重做日志并且避免存档这些日志。即使清除的重做日志没有存档，也可以使用。

如果清除了恢复备份所需的日志文件，那么就不能再从备份中进行恢复。Oracle 将在报警文件中写入一条消息来描述不能恢复的备份。

注意：如果清除了没有存档的重做日志文件，应该制作数据库的另一个备份。

如果想清除某个未存档的重做日志，而该日志可以将脱机表空间变成联机状态，那么可以在 ALTER DATABASE CLEAR LOGFILE 语句中使用 UNRECOVERABLE DATAFILE 子句。

如果清除了用于将脱机表空间变成联机状态的重做日志，将再也不能把脱机表空间变成联机状态。此时，将不得不撤消表空间或执行未完成的恢复过程。注意正常脱机的表空间不需要恢复。

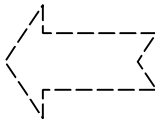
7.9 查看联机重做日志信息

使用下面的视图显示联机重做日志信息。

视图	说明
V\$LOG	显示来自于控制文件的重做日志文件信息
V\$LOGFILE	标识重做日志组及其成员以及成员的状态
V\$LOG_HISTROY	包含日志历史信息

以下查询将返回控制文件中关于数据库的联机重做日志的信息。

```
SELECT * FROM V$LOG;
GROUP#  THREAD#  SEQ  BYTES  MEMBERS  ARC  STATUS  FIRST_CHANGE#  FIRST_TIM
-----
1  1  10605  1048576  1  YES  ACTIVE  11515628  16-APR-00
2  1  10606  1048576  1  NO   CURRENT  11517595  16-APR-00
3  1  10603  1048576  1  YES  INACTIVE  11511666  16-APR-00
4  1  10604  1048576  1  YES  INACTIVE  11513647  16-APR-00
```

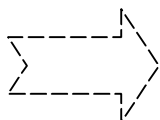


要查看组中所有成员的名称，可以使用类似下面的查询：

```
SELECT * FROM V$LOGFILE;  
GROUP# STATUS MEMBER
```

```
-----  
1 D:\ORANT\ORADATA\IDDB2\REDO04.LOG  
2 D:\ORANT\ORADATA\IDDB2\REDO03.LOG  
3 D:\ORANT\ORADATA\IDDB2\REDO02.LOG  
4 D:\ORANT\ORADATA\IDDB2\REDO01.LOG
```

如果成员的 STATUS 段是空白的，那么表示正在使用该文件。



第 8 章 管理已存档的重做日志

本章描述如何存档重做数据。这一章包括以下内容：

- 什么是已存档的重做日志
- 在 NOARCHIVELOG 和 ARCHIVELOG 两种模式之间进行选择
- 控制存档模式
- 指定存档目标位置
- 指定日志传送模式
- 管理存档目标失败
- 通过指定多个 ARCn 进程调整存档性能
- 控制存档日志进程生成的跟踪输出
- 查看有关已存档的重做日志信息

8.1 什么是已存档的重做日志

Oracle 允许将已填满的联机重做日志文件的组保存到一个或多个脱机目的地，这些日志全都称为已存档的重做日志，或者更简单来说称为存档日志。将联机重做日志文件变成已存档的重做日志文件的进程称为存档。仅当数据库运行在 ARCHIVELOG 模式时，才可能有这种进程。可以选择自动存档或手动存档方式。

已存档的重做日志文件是联机重做日志文件组的已填充成员的一个副本，它与被复制的成员完全相同。已存档的重做日志文件包括一些重做项，这些重做项与重做日志组的同一个成员中的重做项相同，并且保留了组的惟一的日志序号。例如，如果正在多路复用联机重做日志，并且如果 Group 1 包含成员文件 a_log1 和 b_log1，那么存档程序进程 (ARCn) 将存档这些相同成员中的一个。如果将来 a_log1 的信息受到损坏，那么 ARCn 仍然能存档同样的 b_log1。已存档的重做日志包含自从允许存档后创建的每个组的副本。

当运行在 ARCHIVELOG 模式时，并不允许日志写入进程 (LGWR) 使用，以及重写联机重做日志组，直到这些联机重做日志组已经存档为止。当允许自动存档时，后台进程 ARCn 将自动进行存档的操作。为了确保及时存档已填充的联机重做日志，Oracle 将启动所需的多个存档进程。

可以使用已存档的重做日志来做以下事情：

- 恢复数据库
- 更新备用数据库
- 使用 LogMiner 实用程序获取数据库的历史信息

8.2 在 NOARCHIVELOG 和 ARCHIVELOG 两种模式之间进行选择

这一节描述在选用 NOARCHIVELOG 或 ARCHIVELOG 模式来运行数据库时必须考虑的问题，主要包括以下内容：

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

8.2.1 在 NOARCHIVELOG 模式下运行数据库

当在 NOARCHIVELOG 模式下运行数据库时，需要禁止存档联机重做日志。数据库的控制文件表明已填充的组不需要存档。因此，在日志切换后，当已填充的组变成非活动状态时，该组可用于 LGWR 的重用。

是否允许存档联机重做日志文件的已填充的组，要根据运行在数据库上的应用程序的可用性和可靠性两方面要求作出决定。如果经不起磁盘失败事件所引起的数据库中任何数据的丢失，可以采用 ARCHIVELOG 模式。存档已填充的联机重做日志文件需要额外执行一些管理操作。

NOARCHIVELOG 模式只能保护数据库免于实例失败，但是不能避免介质失败。只有保存在联机重做日志组中的对数据库所做的最新更改可以用于实例的恢复过程。换句话说，如果正在使用 NOARCHIVELOG 模式，那么只能将数据库还原（不是恢复）到最近的完整的数据库备份。不能恢复后来的事务。

而且，在 NOARCHIVELOG 模式下不能执行联机表空间的文件备份。此外，不能使用先前在 ARCHIVELOG 模式下的联机表空间的文件备份。只能使用在 NOARCHIVELOG 模式下关闭数据库以还原数据库操作时所获得的完整数据库备份。因此，如果决定在 NOARCHIVELOG 模式下操作数据库，必须按照正常的时间间隔获得数据库的备份。

8.2.2 在 ARCHIVELOG 模式下运行数据库

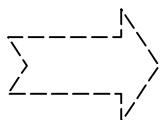
在 ARCHIVELOG 模式下运行数据库，可以指定联机重做日志的存档。数据库控制文件指出在已填充的联机重做日志文件组存档之前，LGWR 不能使用它们。在发生重做日志切换后，已填充的组可以立即存档。

存档已填充的组有以下好处：

- 数据库备份，联机重做日志文件也一起备份，这样可以保证在操作系统或磁盘失败事件后能够恢复所有已提交的事务。
- 如果有已存档的日志，可以使用在打开数据库时以及在正常系统使用过程中所获得的备份。
- 通过将原有的已存档的重做日志应用于备用数据库的方法，可以将当前的备用数据库与它原来的数据库一起保留。

如何存档联机重做日志已填满的组。可以将一个实例配置成自动存档已填满的联机重做日志文件，也可以配置成手动方式。从方便和有效方面来看，自动存档是最常用的方式。图 8-1 说明了存档进程（本例中使用的是 ARC0）是如何存档已填满的联机重做日志文件以生成数据库的存档重做日志的过程。

在分布式数据库中，如果所有数据库都在 ARCHIVELOG 模式下操作，那么可以并列执行分布式数据库的恢复过程。但是，如果分布式数据库中的某个数据库使用了 NOARCHIVELOG 模式，那么全局分布式数据库的恢复将受到一定的限制，这个限制来自



于操作在 NOARCHIVELOG 模式下的数据库的最后一个完整备份。

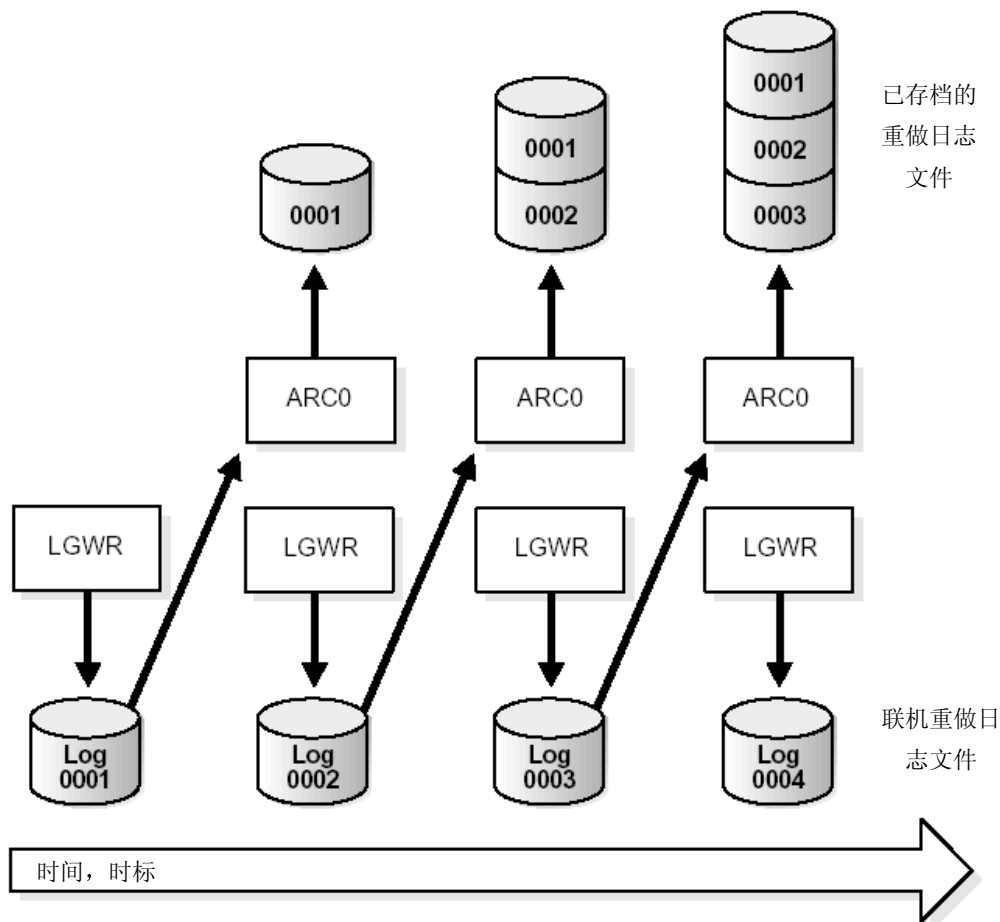


图 8-1 使用 ARCHIVELOG 模式的联机重做日志文件

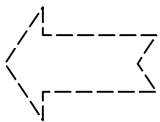
8.3 控制存档模式

这一节描述控制存档模式的方法，包含以下内容：

- 设置最初的数据库存档模式
- 改变数据库的存档模式
- 允许自动存档方式
- 禁止自动存档方式
- 执行手动存档

8.3.1 设置最初的数据库存档模式

作为数据库创建的一部分，在 CREATE DATABASE 语句中需要设置数据库最初的存档模式。通常，因为没必要存档数据库创建后生成的重做信息，所以可以使用 NOARCHIVELOG 作为默认模式。在创建数据库之后，要决定是否改变最初的存档模式。



注意：在 Oracle 安装过程中，如果数据库是自动创建的，那么该数据库最初的存档模式是由操作系统指定的。

8.3.2 改变数据库的存档模式

如果要在 NOARCHIVELOG 和 ARCHIVELOG 两种模式之间进行转换，使用带 ARCHIVELOG 或 NOARCHIVELOG 选项的 SQL 语句 ALTER DATABASE 即可。按照下面的步骤将数据库的存档模式从 NOARCHIVELOG 转换到 ARCHIVELOG：

①关闭数据库实例。

SHUTDOWN

在转换数据库的存档模式之前，打开的数据库必须首先处于关闭状态并且关闭相关的实例。如果介质恢复需要任何的数据文件，那么不能禁止存档功能。

②备份数据库。

在对数据库做出任何重要的改变之前，一直都要备份数据库以免出现任何问题。

③对初始化参数文件进行编辑，其中包含了指定是否允许自动存档以及指定存档日志文件的目标位置的初始化参数。

④启动一个新的实例并装载数据库，但是不打开数据库。

STARTUP MOUNT

要允许或禁止存档，必须装载但是不打开数据库。

⑤转换数据库的存档模式。然后打开数据库进行正常操作。

ALTER DATABASE ARCHIVELOG;

ALTER DATABASE OPEN;

8.3.3 允许自动存档方式

如果操作系统允许，可以自动存档联机重做日志。在这种情况下，在组填满后不需要任何的动作来复制这个组；Oracle 可以自动将这个组存档。因为这种方式很方便，所以一般会选择这种自动存档方式进行存档操作。但是，如果允许自动存档，仍然可以执行手动存档，这种存档方式将在本章的“执行手动存档”一节中加以介绍。

可以在实例启动之前或之后允许自动存档。要在实例启动之后允许自动存档，必须使用管理员权限连接到 Oracle。

在允许自动存档的情况下，经常指定已存档的重做日志的目标位置和文件名格式，这个内容将在本章的“指定存档目标位置”一节中加以介绍。

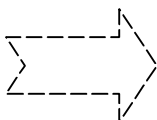
警告：除非数据库也在 ARCHIVELOG 模式下，否则 Oracle 不能自动存档日志文件。

8.3.3.1 在实例启动时允许自动存档方式

要在每次启动实例时允许自动存档已填满的组，可以在数据库初始化参数文件中包含初始化参数 LOG_ARCHIVE_START，并将该参数设置为 TRUE：

LOG_ARCHIVE_START=TRUE

该参数的值将在下次启动数据库时生效。



8.3.3.2 在实例启动后允许自动存档方式

要在没有关闭当前实例的情况下允许自动存档已填满的联机重做日志组，可以使用带 ARCHIVE LOG START 子句的 SQL 语句 ALTER SYSTEM。可以选择包含存档目标位置。

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

如果使用了 ALTER SYSTEM 方法，就不需要关闭实例来允许自动存档方式，但是，使用初始化参数文件的设置可以重新初始化该实例。这些设置可能是允许自动存档的，也可能是不允许自动存档的。

8.3.3.3 控制存档进程的数目

为了确保及时自动存档已填满的重做日志文件，Oracle 将另外启动所需的存档进程 (ARCn)。但是，如果要避免因为启动另外的 ARCn 进程而导致的运行期间的任何开销，就需要在实例启动时，使用 LOG_ARCHIVE_MAX_PROCESSES 初始化参数来指定需要启动的进程的数目。最多可以启动 10 个 ARCn 进程。

同时，这个参数也限制了可以启动的 ARCn 进程的数目。不能启动比这个参数指定的进程数目还多的进程。

初始化参数 LOG_ARCHIVE_MAX_PROCESSES 是动态的，并且可以使用 ALTER SYSTEM 语句进行更改。下面的语句增加（或减少）了当前运行的 ARCn 进程的数目。

```
ALTER SYSTEM SET LOG_ARCHIVE_MAX_PROCESSES=3;
```

通常没有必要改变初始化参数 LOG_ARCHIVE_MAX_PROCESSES 的默认值“1”，原因是 Oracle 会根据系统的工作量适当调整 ARCn 进程。

8.3.4 禁止自动存档

可以在任何时候禁止自动存档联机重做日志组。但是，一旦禁止了自动存档方式，就必须及时手动存档联机重做日志的组。如果在 ARCHIVELOG 模式下运行数据库而且还禁止了自动存档方式，并且如果联机重做日志文件的所有组都已填满但是未存档，那么 LGWR 不能重用联机重做日志组的任何非活动的组来继续写入重做日志条目。因此，数据库操作将暂时中断，直到执行了必要的存档为止。

可以在实例启动之前或之后禁止自动存档方式。要在实例启动后禁止自动存档方式，就必须使用管理员权限进行连接，并拥有 ALTER SYSTEM 权限。

8.3.4.1 在实例启动时禁止自动存档方式

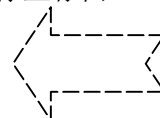
要在每次启动实例时使得不能自动存档已填满的组，可以将数据库初始化参数文件中的初始化参数 LOG_ARCHIVE_START 设置为 FALSE：

```
LOG_ARCHIVE_START=FALSE
```

该参数的值将在下次启动数据库时生效。

8.3.4.2 在实例启动后禁止自动存档方式

要在没有关闭当前实例的情况下不允许自动存档已填满的联机重做日志组，可以使用带 ARCHIVE LOG STOP 参数的 SQL 语句 ALTER SYSTEM。下面的语句将停止存档：



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

当打算禁止自动存档方式时，如果 `ARCn` 正在存档某个重做日志组，那么 `ARCn` 将完成当前组的存档操作，但是该进程将不再开始对下一个已填满的联机重做日志组的存档。

并不需要关闭实例来禁止自动存档方式。但是，如果关闭了实例并在禁止了自动存档方式后重新启动了该实例，那么使用初始化参数文件的设置可以重新初始化该实例。这些设置可能是允许自动存档的，也可能是不允许自动存档的。

8.3.5 执行手动存档

如果是在 `ARCHIVELOG` 模式下操作数据库，就必须存档已填满的联机重做日志文件的非活动组。不管是否允许自动存档都可以手动存档联机重做日志的组：

- 如果不允许自动存档，就必须及时手动存档联机重做日志的组。如果所有联机重做日志组都已经填满但没有存档，那么 `LGWR` 不能重用联机重做日志成员的任何非活动的组来继续写入重做日志条目。因此，数据库操作将暂时中断，直到执行了必要的存档为止。
- 如果允许自动存档，但是又希望将已填满的联机重做日志成员的某个非活动组重新存档到另一个位置，那么可以使用手动存档的方式。在完成手动存档之前，实例可以重用重做日志组并且因此重写这些文件。如果发生这种情况，Oracle 将在报警文件中放置一条错误消息。

如果要手动存档已填满的联机重做日志组，需要使用管理员权限进行连接。使用带 `ARCHIVE LOG` 子句的 `ALTER SYSTEM` 语句来手动存档已填满的联机重做日志文件。下面的语句用来存档所有未存档的日志文件：

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

8.4 指定存档目标位置

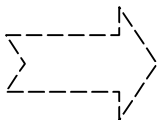
当存档重做日志时，要确定将这些日志进行存档的目标位置，并且熟悉各种目标位置的状态。一种实用的方法，就是使用动态性能（`V$`）视图来访问存档信息，这些视图在本章的“查看有关已存档的重做日志信息”一节中列出。

这一节包含以下内容：

- 指定存档目标位置
- 了解存档目标位置的状态

8.4.1 指定存档目标位置

必须确定日志存档的目标位置是一个还是多个。当多路复用这些日志时，也就是将日志存档到多于一个的位置上。根据以下其中一种方法，通过设置初始化参数可以指定存档的目标位置。



方法	初始化参数	主机	例子
1	LOG_ARCHIVE_DEST_ <i>n</i> 这里的 <i>n</i> 是从 1 到 10 的整数	本地或远程	LOG_ARCHIVE_DEST_1 = 'LOCATION = /disk1/arc' LOG_ARCHIVE_DEST_2 = 'SERVICE = standby1'
2	LOG_ ARCHIVE_DEST 和 LOG_ ARCHIVE_DUPLEX_DEST	本地	LOG_ ARCHIVE_DEST = '/disk1/arc' LOG_ ARCHIVE_DUPLEX_DEST = '/disk2/arc'

8.4.1.1 方法 1: 使用参数 LOG_ARCHIVE_DEST_ *n*

第一种方法是使用 LOG_ARCHIVE_DEST_ *n* 参数（这里的 *n* 是从 1 到 10 的整数）来指定 1 到 10 个不同的存档目标位置。每个带数字后缀的参数惟一标识某个目标位置。

使用以下这些关键字指定 LOG_ARCHIVE_DEST_ *n* 参数的位置:

关键字	表示	例子
LOCATION	本地文件系统位置	LOG_ARCHIVE_DEST_1 = 'LOCATION = /disk1/arc'
SERVICE	通过 Oracle Net 服务名称远程存档	LOG_ARCHIVE_DEST_2 = 'SERVICE = standby1'

如果使用 LOCATION 关键字，可以为操作系统指定一个有效的路径名称。如果指定了 SERVICE，Oracle 将通过 tnsname.ora 文件把网络服务名称称作一个连接描述符。该连接描述符包含连接远程数据库所需的信息。服务名称必须有一个相关联的数据库 SID，所以 Oracle 将为备用数据库正确地更新控制文件的日志历史。

执行以下的步骤，使用 LOG_ARCHIVE_DEST_ *n* 初始化参数来为已存档的重做日志设置目标位置:

①使用 SQL*Plus 来关闭数据库。

SHUTDOWN

②编辑 LOG_ARCHIVE_DEST_ *n* 参数来指定 1 到 10 个存档位置。关键字 LOCATION 将指定操作系统特定的路径名。例如，键入以下内容:

```
LOG_ARCHIVE_DEST_1 = 'LOCATION = /disk1/archive'
LOG_ARCHIVE_DEST_2 = 'LOCATION = /disk2/archive'
LOG_ARCHIVE_DEST_3 = 'LOCATION = /disk3/archive'
```

如果正在向一个备用数据库存档，可以使用关键字 SERVICE 从 tnsname.ora 文件中指定一个有效的网络服务名称。例如，键入以下内容:

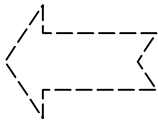
```
LOG_ARCHIVE_DEST_4 = 'SERVICE = standby1'
```

③编辑 LOG_ARCHIVE_FORMAT 初始化参数，使用%s 来包含日志序号作为文件名的一部分，并且使用%t 来包含线程号。使用大写字母（%S 和%T）来以 0 填补文件名左边的空处。例如，键入以下内容:

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

这些设置将生成已存档的日志，如下所示，这些日志的序号为 100，101 和 102:

```
/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
```

8.4.1.2 方法2:使用参数 LOG_ARCHIVE_DEST 和 LOG_ARCHIVE_DUPLEX_DEST

第二种方法,允许指定最多两个位置,一个是由 LOG_ARCHIVE_DEST 参数指定的主存档目标位置,另一个是由 LOG_ARCHIVE_DUPLEX_DEST 参数指定的辅助存档目标位置。无论何时 Oracle 存档重做日志,都会将该日志存档到由这两个参数指定的目标位置上。

执行以下的步骤来使用方法2。

①使用 SQL*Plus 来关闭数据库。

```
SHUTDOWN;
```

②为 LOG_ARCHIVE_DEST 和 LOG_ARCHIVE_DUPLEX_DEST 两个参数指定目标位置(也可以使用 ALTER SYSTEM 语句动态指定 LOG_ARCHIVE_DUPLEX_DEST 参数)。例如,键入以下内容:

```
LOG_ARCHIVE_DEST = '/disk1/archive'  
LOG_ARCHIVE_DUPLEX_DEST = '/disk2/archive'
```

③编辑 LOG_ARCHIVE_FORMAT 初始化参数,使用 %s 来包含日志序号作为文件名的一部分,并且使用 %t 来包含线程号。使用大写字母 (%S 和 %T) 来以 0 填补文件名左边的空处。例如,键入以下内容:

```
LOG_ARCHIVE_FORMAT = arch_%t_%s.arc  
例如,上面的设置生成的存档日志的序号为线程1中的100和101,如下所示:  
/disk1/archive/arch_1_100.arc, /disk1/archive/arch_1_101.arc  
/disk2/archive/arch_1_100.arc, /disk2/archive/arch_1_101.arc
```

8.4.2 了解存档目标位置的状态

每个存档目标位置都由以下变量来决定它的状态:

- Valid/Invalid——指出磁盘位置或服务名称是否是指定的和有效的
- Enabled/Disabled——指出目标位置的可用性状态以及 Oracle 是否可以使用这个目标位置
- Active/Inactive——指出在访问目标位置时是否会出现问题

这些特性也可以进行组合。要获得关于每个目标位置的当前状态及其他信息,可以查看 V\$ARCHIVE_DEST 视图。

在视图中会出现不同的特性,这些特性决定了目标位置的状态,如表 8-1 所示。注意对于使用的目标位置来说,它的特性必须是有效的、允许的和活动的。

初始化参数 LOG_ARCHIVE_DEST_STATE_*n* (这里的 *n* 为 1 到 10 的整数) 允许控制指定的目标位置 (*n*) 的可用状态。这种目标位置的状态有两个值: ENABLE 和 DEFER。ENABLE 表明 Oracle 可以使用这个目标位置,而 DEFER 表明暂时不能使用这个位置。

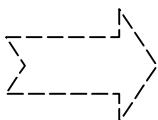


表 8-1 目标位置状态

		状态值		含义
STATUS (状态)	Valid (有效)	Enabled (允许)	Active (活动)	
VALID	TRUE	TRUE	TRUE	用户正确地初始化了目标位置，该目标位置可以用于存档。
INACTIVE	FALSE	n/a	n/a	用户还没有提供或者已经删除了目标位置的信息。
ERROR	TRUE	TRUE	FALSE	创建或写入目标文件时发生错误；请考虑错误信息。
DEFERRED	TRUE	FALSE	TRUE	用户手动并暂时不允许该目标位置。
DISABLED	TRUE	FALSE	FALSE	用户手动并暂时不允许出现错误的目标位置；请参考错误信息。
BAD PARAM	n/a	n/a	n/a	发生参数错误；请考虑错误信息。通常，只有当没有设置 LOG_ARCHIVE_START 参数时，才能看到这种状态。

8.5 指定日志传送模式

有两种模式可以将已存档的日志传送到它们的目标位置：正常存档传送模式和备用传送模式。正常传送包括将文件传送到本地磁盘上。备用传送包括通过网络将文件传送到本地或远程备用数据库上。

8.5.1 正常传送模式

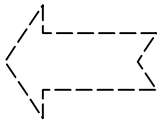
在正常传送模式下，存档目标是数据库服务器的另一个磁盘驱动器。在这种配置下，存档操作不用与实例所需的其他文件竞争空间并且能够更快地完成。可以用 LOG_ARCHIVE_DEST_n 或 LOG_ARCHIVE_DEST 参数来指定目标位置。

理想情况下，应该将已存档的重做日志文件以及相应的数据库备份从本地磁盘中永久地转移到并不昂贵的脱机存储介质中，比如存储到磁带中。因为存档日志最主要的价值在于用于数据库恢复，所以希望确保这些日志的安全，以便主数据库遭到灾难性打击时可以派上用场。

8.5.2 备用传送模式

在备用传送模式下，存档目标位置可以是本地的备用数据库，也可以是远程的备用数据库。

警告：可以在本地磁盘上维护备用数据库，但是 Oracle 强烈推荐在远程站点维护备用



数据库，以便尽量避免出现灾难性错误。

如果是在管理恢复模式下操作备用数据库，那么，通过自动应用传送的存档日志可以使备用数据库与源数据库保持同步。

要将文件成功地传送到备用数据库，ARC*n* 或服务器进程必须做以下事情：

- 识别远程位置
- 通过远程文件服务器（RFS）进程传送已存档的日志

ARC*n* 进程都将为每个备用目标位置创建一个相应的 RFS。例如，如果 3 个 ARC*n* 进程正在存档两个备用数据库，那么 Oracle 将会建立 6 个 RFS 连接。

可以使用 Oracle Net 方法通过网络将已存档的日志传送到一个远程位置。指定 Oracle Net 服务名称作为目标位置的一个属性，进而表明一个远程存档。然后，Oracle 将通过 tnsname.ora 文件把 SERVICE_NAME 参数设置的服务名称称作一个连接描述符。该描述符包含连接远程数据库所需的信息。服务名称必须有一个相关联的数据库 SID，这样 Oracle 才能为备用数据库正确更新控制文件的日志历史。

运行在目的节点的 RFS 进程担当 ARC*n* 客户的网络服务器。实质上，ARC*n* 首先将信息添加到 RFS，而 RFS 再将该信息传送到备用数据库。

存档到远程目标位置时，RFS 进程需要负责以下任务：

- 使用 ARC*n* 进程的网络 I/O 资源
- 在备用数据库上使用 STANDBY_ARCHIVE_DEST 参数来创建文件名
- 在远程站点添加日志文件
- 更新备用数据库的控制文件（恢复管理器可以使用它进行恢复工作）

已存档的日志文件主要用于维护备用数据库，该备用数据库是某个数据库的一个精确的复制品。可以在备用存档模式下操作数据库，用原数据库已存档的重做日志来自动更新备用数据库。

8.6 管理存档目标失败

有时，在自动存档模式下操作时，会因为某些问题而导致存档目标失败。为了使引起目标失败的问题最小化，Oracle 提供了一些选项。这些选项包含在以下章节中：

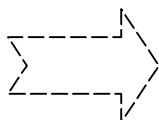
- 指定最小数目的成功的目标
- 重新存档到一个失败过的目标位置

8.6.1 指定最小数目的成功的目标

在 Oracle 重用联机日志文件之前，可选的初始化参数 LOG_ARCHIVE_MIN_SUCCEED_DEST=*n*（这里的 *n* 是 1 到 10 的整数，如果选择使用多路复用，也可以是 1 到 2）决定了 Oracle 必须成功存档重做日志组的目标的最小数目。默认值是 1。

8.6.1.1 指定强制目标和任选目标

使用 LOG_ARCHIVE_DEST_*n* 参数可以指定目标拥有 OPTIONAL（默认值）还是



MANDATORY 属性。LOG_ARCHIVE_MIN_SUCCEED_DEST=*n* 参数使用所有 MANDATORY 属性的目标以及一些 OPTIONAL 属性的非备用目标来决定 LGWR 是否可以重写联机日志。

当决定如何设置参数时，注意以下几点：

- 不为目标指定 MANDATORY 属性与为其指定 OPTIONAL 属性的功效是一样的。
- 必须至少拥有一个可以声明 OPTIONAL 或 MANDATORY 属性的本地目标。
- 在至少一个本地目标上使用 LOG_ARCHIVE_MIN_SUCCEED_DEST=*n* 参数时，将按 MANDATORY 方式进行操作，因为 LOG_ARCHIVE_MIN_SUCCEED_DEST 参数的最小值是 1。
- 任何 MANDATORY 属性目标的失败，包括 MANDATORY 属性的备用目标，都和 LOG_ARCHIVE_MIN_SUCCEED_DEST 参数没有关系。
- LOG_ARCHIVE_MIN_SUCCEED_DEST 参数值不能比目标的总数还大，也不能大于 MANDATORY 属性目标加上 OPTIONAL 属性的本地目标的总数。
- 如果使 MANDATORY 属性目标具有 DEFER 属性，并且 Oracle 在没有将已存档的日志传送到备用站点的情况下重写了联机日志，那么必须手动将日志传送到备用站点。

也可以通过 LOG_ARCHIVE_DEST 和 LOG_ARCHIVE_DUPLEX_DEST 参数来建立强制目标或是任选目标。注意以下几条惯例：

- 任何由 LOG_ARCHIVE_DEST 参数声明的目标都是强制性的。
- 如果 LOG_ARCHIVE_DUPLEX_DEST=1，那么任何由 LOG_ARCHIVE_DUPLEX_DEST 参数声明的目标都是任选的；如果 LOG_ARCHIVE_DUPLEX_DEST=2，那么任何由它来声明的目标都是强制性的。

8.6.1.2 示例方案：指定成功目标的数目

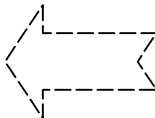
通过示例方案可以非常容易地看出 LOG_ARCHIVE_DEST_*n* 参数和 LOG_ARCHIVE_MIN_SUCCEED_DEST 参数之间的关系。

方案 1：在这种方案下，存档到 3 个本地目标位置，每个这种本地目标都声明为 OPTIONAL。表 8-2 说明了这种方案下 LOG_ARCHIVE_MIN_SUCCEED_DEST=*n* 参数可能的值。

表 8-2 第 1 种方案下 LOG_ARCHIVE_MIN_SUCCEED_DEST 的值

值	含义
1	仅在至少一个 OPTIONAL 目标成功的情况下，Oracle 可以重用日志文件。
2	仅在至少两个 OPTIONAL 目标成功的情况下，Oracle 可以重用日志文件。
3	仅在所有 OPTIONAL 目标成功的情况下，Oracle 可以重用日志文件。
4 或更大	出现错误：这个值大于目标的总数。

这种方案说明了即使 LOG_ARCHIVE_DEST_*n* 参数没有明确设置任何目标的属性为 MANDATORY，但是在 LOG_ARCHIVE_MIN_SUCCEED_DEST 设置为 1、2 或 3 时，Oracle 也必须成功存档一个或多个目标。



方案 2：在这种方案下，考虑以下情况：

- 指定两个 MANDATORY 属性的目标。
- 指定两个 OPTIONAL 属性的目标。
- 没有目标是备用数据库。

表 8-3 显示了 LOG_ARCHIVE_MIN_SUCCEED_DEST= n 参数可能的值。

表 8-3 第 2 种方案下 LOG_ARCHIVE_MIN_SUCCEED_DEST 的值

值	含义
1	Oracle 忽略这个值并使用 MANDATORY 目标的数目（在这个例子中，是 2）。
2	即使没有任何一个 OPTIONAL 目标成功，Oracle 也可以重用日志文件。
3	仅在至少一个 OPTIONAL 目标成功的情况下，Oracle 可以重用日志文件。
4	仅在这两个 OPTIONAL 目标都成功的情况下，Oracle 才可以重用日志文件。
5 或更大	出现错误：这个值大于目标的总数。

这种方案说明了：不管是否将 LOG_ARCHIVE_MIN_SUCCEED_DEST 参数设置为存档到最少数目的目标，Oracle 都必须存档到指定为 MANDATORY 属性的目标。

8.6.2 重新存档到一个失败过的目标位置

在出现错误之后，使用 LOG_ARCHIVE_DEST_ n 参数的 REOPEN 属性来指定 ARC n 能否以及何时重新存档到失败过的目标位置。REOPEN 属性适用于各种错误，不单单适用于 OPEN 错误。

REOPEN= n 设置了 ARC n 重新打开失败过的目标所需的最小秒数。 n 的默认值为 300 秒。如果 n 的值为 0，就表示关掉 REOPEN 选项。换句话说，在失败之后 ARC n 不会再进行存档，如果没有指定 REOPEN 关键字，在发生错误后，ARC n 将不再重新打开某个目标。

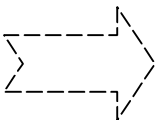
不能使用 REOPEN 来指定一个重新连接和传送已存档的日志的限制数目。REOPEN 可能成功，也可能失败，此时将重新设置 REOPEN 信息，

如果为 OPTIONAL 目标指定了 REOPEN 属性，那么，如果发生错误，Oracle 将重写联机日志。如果为 MANDATORY 目标指定了 REOPEN 属性，那么，当不能成功存档时，Oracle 将安装制作的数据库。在这种情况下，需要考虑使用以下办法：

- 手动存档到失败过的目标位置。
- 通过延期目标来更改目标，将目标指定为任选的，或者更改服务。
- 撤消目标。

当使用 REOPEN 关键字时，注意以下内容：

- 仅从日志文件开始处启动存档操作时，ARC n 可以重新打开目标，但是不是在整个当前操作过程中都可以重新打开目标。ARC n 经常从一开始就重试复制日志。
- 如果指定或默认了 REOPEN 时间，ARC n 将检查记录错误加上 REOPEN 时间间隔的时间是否小于当前时间。如果是，ARC n 将重试复制日志。
- REOPEN 子句将成功影响 ACTIVE=TRUE 目标状态。不会改变 VALID 和 ENABLED 状态。



8.7 通过指定多个 ARCn 进程调整存档性能

对于大多数数据库，ARCn 不会影响整个系统的性能。但是，在一些大型数据库站点上，存档将影响系统的性能。一方面，如果 ARCn 运转地很快，当 ARCn 运行时，因为 CPU 周期消耗在存档过程的缘故将会降低整个系统的性能。另一方面，如果 ARCn 运转地很慢，那么它不会对系统性能造成任何损害，但是会花费更长的时间存档重做日志文件，而且会产生瓶颈效应，因为所有重做日志组都在等着存档，这样就不能利用这些重做日志组。

最多可以为每个数据库实例指定 10 个 ARCn 进程。通过设置初始化参数 LOG_ARCHIVE_MAX_PROCESSES = n 可以允许在启动或运行期间使用多进程特性（这里的 n 是 1 到 10 的整数）。默认情况下，该参数设置为 1。

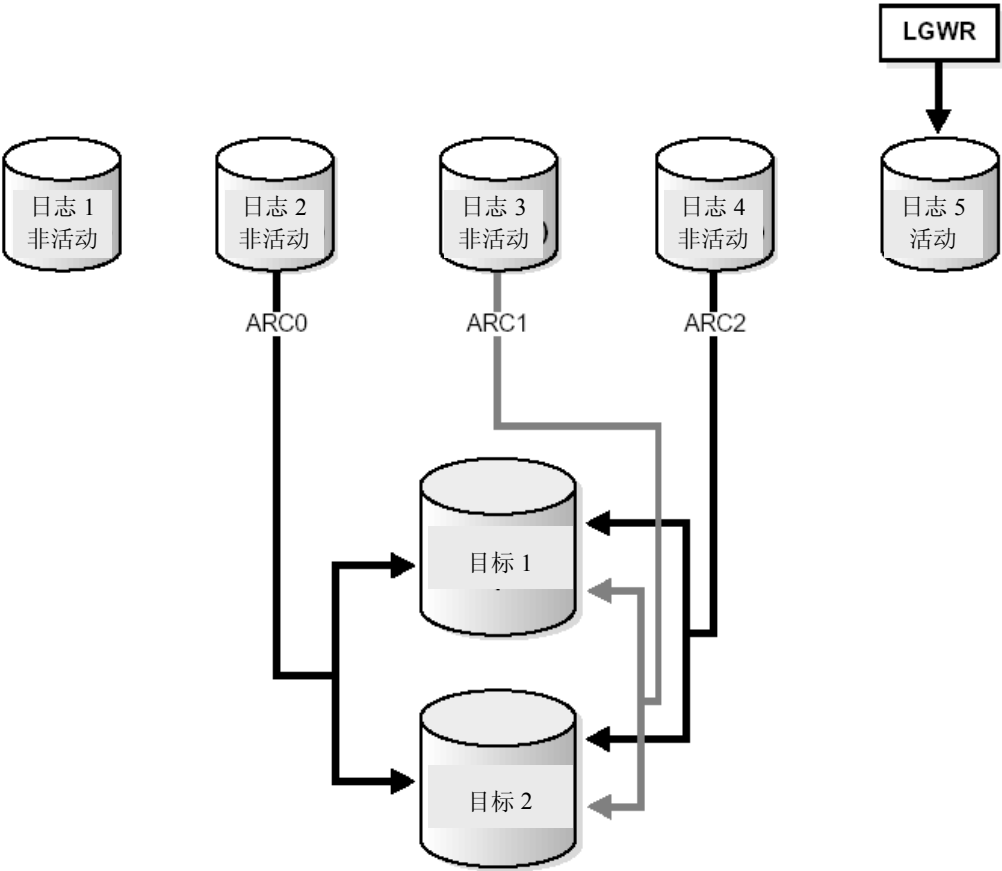
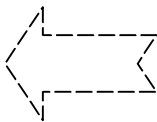


图 8-2 使用多个 ARCn 进程

因为如果当前数目的 ARCn 进程不足以处理当前工作量时，LGWR 会自动增加 ARCn 进程的数目，所以这个参数可以指定最初使用的 ARCn 进程数目，或者增加或减少当前的 ARCn 进程数目。假设最初设置的 ARCn 进程数目为 4，下面的语句将进程数目减少至 2。

```
ALTER SYSTEM SET LOG_ARCHIVE_MAX_PROCESSES=2
```

减少 ARCn 进程数目时，并不需要决定停止哪个进程。而且，不能将这个参数的值改



为 0，所以至少会有一个 `ARCn` 进程数目一直是活动的。通过 `V$ARCHIVE_PROCESSES` 视图来查看每个存档进程状态的信息。已经停止的进程显示为 `IDLE` 状态。

在以下情况下，创建多进程非常有用：

- 使用多于两个的联机重做日志时
- 存档到不止一个目标位置时

多个 `ARCn` 进程避免了当 `LGWR` 在多个联机重做日志之间切换时发生的瓶颈效应，多个 `ARCn` 进程要比一个 `ARCn` 进程快，并且可以将非活动日志写入多个目标。每个 `ARCn` 进程在某一时刻只能处理一个非活动日志，并且必须将其存档到每个指定的目标。

例如，如果维护 5 个联机重做日志文件，那么可以决定使用 3 个 `ARCn` 进程来启动实例。当 `LGWR` 正写入某个日志文件时，`ARCn` 进程可以同时最多 3 个非活动日志文件存档到各个目标中。如图 8-2 所示，`ARCn` 进程的每个实例都假设负责将一个日志文件存档到所有定义好的目标位置。

8.8 控制存档日志进程生成的跟踪输出

正如第 5 章的“跟踪文件和报警文件”一节中所讨论的那样，在适当时候后台进程经常会往跟踪文件中写入信息。对于存档日志进程来说，是有可能控制生成的输出的。

可以设置初始化参数 `LOG_ARCHIVE_TRACE` 来指定跟踪级别（`trace level`）。可以为以下的值：

跟踪级别	含义
0	不允许存档日志跟踪——默认设置。
1	跟踪重做日志文件的存档。
2	跟踪每个存档日志目标的存档状态。
4	跟踪存档操作的阶段。
8	跟踪存档日志目标的活动。
16	跟踪详细的存档日志目标的活动。
32	跟踪存档日志目标的参数修改。
64	跟踪 <code>ARCn</code> 进程的状态活动。

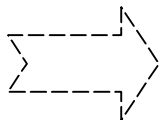
指定一个值，其值等于希望跟踪的各个级别的总和，这样就可以合并跟踪级别。例如，如果设置为 `LOG_ARCHIVE_TRACE=12`，那么将生成跟踪级别分别为 8 和 4 的输出。可以为主数据库和备用数据库设置不同的值。

`LOG_ARCHIVE_TRACE` 参数的默认值是 0，在这种级别下，根据出错条件仍然会生成适当的报警和跟踪条目。

可以使用 `ALTER SYSTEM` 语句动态改变这个参数的值。例如：

```
ALTER SYSTEM SET LOG_ARCHIVE_TRACE=12
```

按这种方法进行的改变将在下一次存档操作启动时生效。



8.9 查看有关已存档的重做日志信息

使用下面的方法可以显示有关已存档的重做日志信息：

- 固定的视图
- ARCHIVE LOG LIST 命令

8.9.1 固定的视图

有几个动态性能视图，其中包含了有关已存档的重做日志的信息，这些信息非常有用。

动态性能视图	说明
V\$DATABASE	指出数据库是在 ARCHIVELOG 模式还是 NOARCHIVELOG 模式。
V\$ARCHIVED_LOG	显示控制文件中的历史存档日志信息。如果使用了恢复一览表，RC_ARCHIVED_LOG 视图将包含类似的信息。
V\$ARCHIVE_DEST	描述当前的实例、所有存档目标以及这些目标的当前值、模式和状态。
V\$ARCHIVE_PROCESSES	显示实例用到的各种存档进程的状态信息。
V\$BACKUP_REDOLOG	包含有关已存档的日志的任何备份的信息。如果使用了恢复一览表，RC_BACKUP_REDOLOG 将包含类似的信息。
V\$ LOG	显示数据库所用的所有联机重做日志组并指出哪一个需要存档。
V\$ LOG_HISTROY	包含日志历史信息，比如哪些日志已经存档以及每个已存档的日志的 SCN 范围。

例如，以下的查询将显示哪一个联机重做日志组需要存档：

```
SELECT GROUP#, ARCHIVED
FROM SYS.V$LOG;
GROUP# ARC
-----
1 YES
2 NO
```

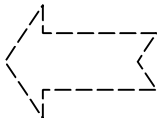
要查看当前的存档模式，查询 V\$DATABASE 视图即可：

```
SELECT LOG_MODE FROM SYS.V$DATABASE;
LOG_MODE
-----
NOARCHIVELOG
```

8.9.2 ARCHIVE LOG LIST 命令

可以使用 SQL*Plus 命令 ARCHIVE LOG LIST 来显示已连接实例的存档信息。例如：

```
SQL> ARCHIVE LOG LIST
Database log mode Archive Mode
Automatic archival Enabled
Archive destination D:\ORANT\oradata\IDDB2\archive
Oldest online log sequence 11160
```

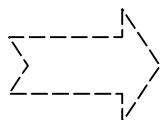


Next log sequence to archive 11163

Current log sequence 11163

上面显示的内容是有关当前实例的已存档重做日志设置的所有信息：

- 数据库当前操作在 ARCHIVELOG 模式。
- 允许自动存档。
- 已存档重做日志的目标位置是 D:\ORANT\oradata\IDDB2\archive。
- 最开始的已填充联机重做日志组的序号为 11160。
- 下一个要存档的已填充联机重做日志组的序号为 11163。
- 当前的联机重做日志文件的序号为 11163。



第 9 章 使用 LogMiner 来分析重做日志文件

Oracle 的 LogMiner 允许通过 SQL 接口查询重做日志文件。重做日志文件包含有关数据库的历史活动信息。

本章讨论以下内容：

- 了解分析重做日志文件的价值
- 在开始之前需要知道的内容
- 使用 LogMiner
- 使用 LogMiner 的例子

本章描述当用命令行方式使用 LogMiner 的功能。也可以通过 Oracle LogMiner Viewer 图形用户界面 (GUI) 来访问 LogMiner 的功能。该 LogMiner 查看器是 Oracle Enterprise Manager 的一部分。

9.1 了解分析重做日志文件的价值

Oracle 重做日志文件包含对用户数据和数据库的数据字典所做的所有改变。因此，重做日志文件是包含执行恢复操作所必需的所有信息的惟一源头。因为重做日志数据往往保存在已存档的文件中，所以这些数据已经可以利用。要获得 LogMiner 使用的数据，应该不需要另外的操作。

以下是重做日志文件包含的数据的潜在用处：

- 查明何时发生数据库逻辑上的损坏，比如在应用程序级造成的错误，也许已经开始损坏。举一个应用程序级造成错误的例子，如果用户在更新数据库时出错，本来打算将所有雇员的薪水增长 10%，但是错误地改成增长 100%，就会造成应用程序级的错误。了解发生信息损坏的准确时间是很重要的，以便于知道何时初始化基于时间的或者基于变化的恢复操作。这样才能把数据库刚好还原到信息损坏之前所处的状态。
- 决定在事务级执行细粒度的恢复所必需的动作。如果完全了解并考虑到现有的需要，可以执行基于表的撤消操作来回滚一组更改。通常，必需先将表还原到它先前的状态，然后再用一个已存档的日志文件向前回滚。
- 性能协调和容量规划。
- 实行事后审计。

9.2 在开始之前需要知道的内容

在开始使用 LogMiner 之前，了解 LogMiner 如何处理重做日志文件以及字典文件是很重要的。这有助于得到正确的结果和规划系统资源的使用。这一节将讨论以下概念：

- 重做日志文件
- 字典选项

- DDL 语句的跟踪
- 存储管理
- 从重做日志文件中获取数值
- LogMiner 的局限
- LogMiner 的视图

9.2.1 重做日志文件

当运行 LogMiner 时，需要指定待分析的重做日志文件的名称。LogMiner 从那些重做日志文件中重新获取信息并将通过 V\$LOGMNR_CONTENTS 视图返回这些信息。

可以用 SQL 来查询 V\$LOGMNR_CONTENTS 视图，类似查询其他视图那样。在 V\$LOGMNR_CONTENTS 视图中执行的每一项选择操作都会对重做日志文件进行顺序读取。

关于重做日志文件要牢记以下事项：

- 重做日志文件必须来自于 Oracle 数据库的 8.0 或更高版本。然而，某些 LogMiner 的 9.0.1 版本特性只处理在 Oracle 9i 或更新版本的数据库上生成的重做日志文件。
- 重做日志文件必须使用与运行 LogMine 相同的数据库的字符集。
- 通常，重做日志文件的分析需要生成重做日志文件的数据库所生成的字典。
- 如果正在平面文件格式或重做日志文件中使用字典，那么要分析的重做日志文件可以来自于 LogMiner 运行的数据库，也可以来自于其他数据库。如果正在使用联机目录作为 LogMiner 字典，那么可以只分析来自于 LogMiner 运行的数据库的重做日志文件。
- LogMiner 必须运行在生成正在分析的重做日志文件的硬件平台上。但是，它们并不一定在相同的系统上。

当运行 LogMiner 时，正确指定重做日志文件是非常重要的。如果遗漏了包含某些所需数据的重做日志文件，在查询 V\$LOGMNR_CONTENTS 视图时将会得到不正确的结果。

要决定正在分析的是哪个重做日志文件，可以看一下 V\$LOGMNR_LOG 视图，该视图为每个日志文件包含了一行信息。

9.2.2 字典选项

要完整地译出重做日志文件的内容，LogMiner 需要访问数据库字典。

LogMiner 使用字典将内部的对象标识符和数据类型称作对象名称和外部数据格式。假如没有字典，LogMiner 将返回内部的对象 ID 号并用十六进制字节表示数据。

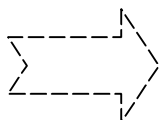
例如，SQL 语句：

```
INSERT INTO emp(name, salary) VALUES ('John Doe', 50000);
```

LogMiner 将代替上面的 SQL 语句显示为下面的内容：

```
insert into Object#2581(col#1, col#2) values (hextoraw('4a6f686e20446f65'),  
hextoraw('c306'));"
```

LogMiner 给出 3 种方式从源字典中取出数据：将字典中的数据提取到平面文件，将字



典中的数据提取到重做日志文件，或者使用联机目录（数据库当前使用的字典）。

9.2.2.1 将字典提取到平面文件或重做日志文件

LogMiner 字典文件包含识别创建这个文件的数据库以及该文件创建时间的信息。该信息用于确认所选重做日志文件而获取的字典。

字典文件必须与正在分析的日志文件一样拥有相同的数据库字符集以及数据库。总之，重做日志文件的分析需要字典，该字典从生成重做日志文件的数据库中生成。

DBMS_LOGMNR_D.BUILD 过程允许将字典提取到平面文件或重做日志文件。

将字典提取到平面文件 当数据字典提取到一个平面文件时，DDL 语句可以由其他用户发布。因此，也许提取的文件不包含与数据字典一致的快照。

当字典包含在平面文件中时使用的系统资源比字典包含在重做日志文件时使用的要少。

建议时常备份字典提取以确保正确地分析以前的重做日志文件。

将字典提取到重做日志文件 当正在将字典提取到重做日志流时，不能执行任何 DDL 语句。因此，字典快照能够保持一致。

将字典提取到重做日志文件的过程将耗费数据库资源，但是如果将提取限制在非高峰时间，那么就不会造成问题并且会比提取到平面文件更快。根据字典的大小，可以将它包含在多个重做日志文件中。

建议定期备份重做日志文件，以便保存信息并且在以后的时间使用这些信息。理想情况下，不用附加任何的步骤，因为数据库正在正确地实施管理，所以可以在适当位置用一个进程来备份和还原已存档的重做日志文件。而且，因为所需时间的关系，在非高峰时间内进行这些操作是一种不错的办法。

9.2.2.2 使用联机目录

要想使 LogMiner 使用数据库当前采用的字典，在启动 LogMiner 时，必须指定联机目录作为字典源。

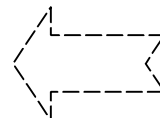
联机目录包含数据库的最新信息。但是，在重做日志文件生成后，如果联机目录已发生重大变更，那么就正分析的联机重做日志文件而言，该联机目录也许不正确。

9.2.3 DDL 语句的跟踪

根据启动时指定的字典源（平面文件字典，重做日志中的字典或者联机目录），LogMiner 自动构建它自己的内部字典。

如果字典源是平面文件字典或者重做日志文件中的字典，那么可以使用 DDL_DICT_TRACKING 选项设置 LogMiner 跟踪数据定义语言(DDL)语句。这样设置该选项后，LogMiner 会将在重做日志中可见的任何一条 DDL 语句应用到它自己的内部字典中。然后，更新过的信息会返回到 V\$LOGMNR_CONTENTS 视图的 SQL_REDO 列中。

因为可以处理发生在表的逻辑结构中的改变，所以跟踪 DDL 语句的能力将有助于监视模式的演变。另外，也可以显示数据处理语言（DML）在提取字典后创建的新表上执行的操作。



注意：LogMiner 的内部字典与包含在平面文件或重做日志文件中的 LogMiner 字典并不一样，了解这一点很重要。LogMiner 更新它的内部字典，但不更新包含在平面文件或重做日志文件中的字典。

通过 V\$LOGMNR_CONTENTS 视图查看时，如果只传送一遍，此时最好使用 DDL_DICT_TRACKING 选项。如果传送多遍，则需牢记以下几点：

- 如果还设置了 NO_DICT_RESET_ONSELECT 选项，那么在每个选择的操作后，字典将不刷新。因此，也许会得到重做日志文件中已修改对象的不正确的信息。
- 如果没有设置 NO_DICT_RESET_ONSELECT 选项，那么在 V\$LOGMNR_CONTENTS 视图上执行了每个选择的操作后，字典将花费一定的时间进行刷新。

9.2.4 存储管理

在 Oracle 9i 中，如果指定了 DDL_DICT_TRACKING 选项，那么当从重做日志文件中读取数据字典快照，或者从平面文件中读取数据字典时，LogMiner 可能会使用数据库来存储备份数据。

从重做日志文件中将数据字典快照读取到 SYSTEM 模式下的表中。因此，Oracle 建议通过添加新的数据文件来扩展 SYSTEM 表空间（使用 ALTER TABLESPACE ADD DATAFILE 语句）。

9.2.5 从重做日志文件中取出数据的值

LogMiner 能够基于真实的数据值来进行查询。例如，可以发布一个查询来选择 scott.emp 表中的所有更改操作或者所有由用户 scott 执行的删除操作。也可以执行一个查询来显示 scott.emp 表中增加 sal 超过一定数量的所有更改操作。类似这样的数据可以用于分析系统行为以及执行审计任务。

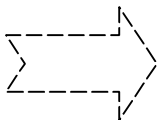
使用 mine 函数从重做日志文件中提取 LogMiner 所需的数据，mine 函数有 DBMS_LOGMNR.MINE_VALUE 和 COLUMN_PRESENT。这些函数是 DBMS_LOGMNR 包的一部分。

9.2.6 LogMiner 的局限

以下是 LogMiner 的局限：

- 不支持以下内容：
 - 数据类型 LONG 和 LOB
 - 简单的和嵌套的数据提取类型（ADTs）
 - 集合（嵌套表和 VARRAYs）
 - 对象 Refs
 - 索引结构的表（IOTs）

如果 LogMiner 看到以上这些内容，那么它将不能生成显示在 V\$LOGMNR_CONTENTS 视图的 SQL_REDO 列中的数据。但是，V\$LOGMNR_CONTENTS 视图中的所有其他数据仍然有效。



- 因为 LogMiner 支持直接路径插入操作，所以必须允许追加记录并打开 ARCHIVE 模式。
- 当在共享服务器环境运行 Oracle 数据库时，不能运行 LogMiner。（但是，可以分析共享服务器环境中创建的重做日志文件。）

9.2.7 LogMiner 的视图

LogMiner 提供以下视图。可以像使用其他视图一样用 SQL 进行查询。

- **V\$LOGMNR_CONTENTS**

显示对用户所做的修改以及表信息。

- **V\$LOGMNR_DICTIONARY**

显示有关 LogMiner 字典文件的信息，这些文件是由使用 STORE_IN_FLAT_FILE 选项创建的字典提供的。显示的信息包括数据库名称和状态信息。

- **V\$LOGMNR_LOGS**

显示有关指定的日志文件的信息。每个日志文件的信息占一行。

- **V\$LOGMNR_PARAMS**

显示有关可选的 LogMiner 参数的信息，包括启动和结束系统更改号码（SCNs）以及启动和结束的时间。

9.3 使用 LogMiner

要运行 LogMiner，可以使用两个 PL/SQL 包：

- **DBMS_LOGMNR**

包含初始化 LogMiner 和运行 LogMiner 必需的过程。这些过程包括指定日志文件名、筛选标准和 LogMiner 会话特性的接口。

- **DBMS_LOGMNR_D**

查询当前数据库的字典表并创建 LogMiner 字典文件。

LogMiner 包属于 SYS 模式。因此，如果没有按用户 SYS 身份进行连接，那么就必须调用中包括 SYS。例如：

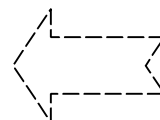
```
EXECUTE SYS.DBMS_LOGMNR.END_LOGMNR
```

这一节剩余部分介绍典型 LogMiner 会话中的步骤。每个步骤都用单独一段进行描述。

- ①提取字典
- ②指定要分析的重做日志文件
- ③启动 LogMiner
- ④分析 V\$LOGMNR_CONTENTS 视图中的输出
- ⑤使用 LogMiner 执行对象级的恢复操作
- ⑥结束 LogMiner 会话

9.3.1 提取字典

要使用 LogMiner，必须通过以下方法提供一个字典：



- 将数据库字典信息提取到平面文件
- 将数据库字典信息提取到重做日志文件
- 通过 `DICTIONARY_FROM_ONLINE_CATALOG` 选项来指定使用联机目录。

9.3.1.1 将数据库字典信息提取到平面文件

如果要将数据库字典信息提取到平面文件，使用 `DBMS_LOGMNR_D.BUILD` 过程即可。

`DBMS_LOGMNR_D.BUILD` 过程需要访问可以放置字典文件的某个目录。因为 PL/SQL 过程通常不访问用户目录，所以必须通过 `DBMS_LOGMNR_D.BUILD` 过程指定要使用的目录，否则过程会失败。要指定目录，可以设置初始化参数，即 `init.ora` 文件中的 `UTL_FILE_DIR` 参数。

例如，设置 `UTL_FILE_DIR` 参数为使用 `/oracle/database` 作为放置字典文件的目录，在 `init.ora` 文件中键入以下内容：

```
UTL_FILE_DIR = /oracle/database
```

要使对 `init.ora` 文件所做的修改生效，必须停止和重新启动数据库。

要确保 `DBMS_LOGMNR_D.BUILD` 过程成功完成，一定要指定必要的参数，如下面例子中所示。还要确定在构建字典时不进行 DDL 操作。

将字典提取到平面文件的步骤取决于该字典是为 Oracle 9i 还是为 Oracle 8i 创建的。

9.3.1.2 创建 Oracle 9i 数据库的字典平面文件

① 要为字典文件指定放置的位置，可以设置 `init.ora` 文件中的 `UTL_FILE_DIR` 参数。

例如，设置 `UTL_FILE_DIR` 参数为使用 `/oracle/database` 作为放置字典文件的目录，在 `init.ora` 文件中键入以下内容：

```
UTL_FILE_DIR = /oracle/database
```

记住对 `init.ora` 文件所做的修改要想生效，必须停止和重新启动数据库。

② 如果数据库处于关闭状态，可以使用 `SQL*Plus` 来装载并打开数据库，数据库中有希望分析的重做日志文件。例如，键入 `STARTUP` 命令来装载和打开数据库：

```
SQLPLUS> STARTUP
```

③ 执行 PL/SQL 过程 `DBMS_LOGMNR_D.BUILD`。指定字典的文件名和该文件的目录路径名称。这个过程就会创建该字典文件。例如，键入以下内容来创建 `/oracle/database` 中的文件 `dictionary.ora`：

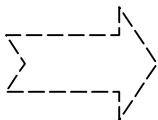
```
SQLPLUS>EXECUTE DBMS_LOGMNR_D.BUILD('dictionary.ora',
```

```
SQLPLUS>'/oracle/database/'
```

```
SQLPLUS>options => DBMS_LOGMNR_D.STORE_IN_FLAT_FILE);
```

也可以不用 `STORE_IN_FLAT_FILE` 选项来指定文件名和位置。结果是一样的。

创建 Oracle 8 数据库的字典平面文件 尽管 LogMiner 只运行在 Oracle 8.1 或更高版本的数据库上，但是可以用它来分析 8.0 版本数据库的重做日志文件。然而，依据日志文件的版本不同，可以用来分析日志文件的 LogMiner 功能也会有所不同。也就是说，Oracle



9i 的日志文件已经增加了利用 LogMiner 功能的特性，所以用老一些的 Oracle 版本创建的日志文件就会在 LogMiner 支持的操作和数据类型方面有所限制。

①使用操作系统的复制命令来复制 dbmslmd.sql 脚本，该脚本包含在 Oracle 8i 数据库的 \$ORACLE_HOME/rdbms/admin 目录中，这与该脚本在 Oracle 8 数据库中的目录相同。例如，键入以下内容：

```
% cp /8.1/oracle/rdbms/admin/dbmslmd.sql /8.0/oracle/rdbms/admin/dbmslmd.sql
```

②如果数据库处于关闭状态，可以使用 SQL*Plus 来装载并打开数据库，数据库中有希望分析的重做日志文件。例如，键入 STARTUP 命令来装载和打开数据库：

```
SQLPLUS> STARTUP
```

③执行 8.0 数据库中复制的 dbmslmd.sql 脚本来创建 DBMS_LOGMNR_D 包。例如，键入以下内容：

```
@dbmslmd.sql
```

需要键入脚本完整的路径。

④要指定字典文件放置的位置，可以设置 init.ora 文件中的 UTL_FILE_DIR 参数。

例如，设置 UTL_FILE_DIR 参数为使用 /oracle/database 作为放置字典文件的目录，在 init.ora 文件中键入以下内容：

```
UTL_FILE_DIR = /oracle/database
```

记住对 init.ora 文件所做的修改要想生效，必须停止和重新启动数据库。

⑤执行 PL/SQL 过程 DBMS_LOGMNR_D.BUILD。这个过程将会创建字典文件。指定字典文件的文件名和该文件的目录路径名称。例如，键入以下内容来创建 /oracle/database 中的文件 dictionary.ora：

```
SQLPLUS>EXECUTE DBMS_LOGMNR_D.BUILD('dictionary.ora'
```

```
SQLPLUS>'/oracle/database/')
```

```
SQLPLUS>options => DBMS_LOGMNR_D.STORE_IN_FLAT_FILE);
```

也可以不用 STORE_IN_FLAT_FILE 选项来指定文件名和位置。结果是一样的。

9.3.1.3 将数据库字典信息提取到重做日志文件

要将数据库字典信息提取到重做日志文件，数据库必须处于 ARCHIVELOG 模式。

当使用 STORE_IN_REDO_LOGS 选项指定文件名和位置时，将返回一条错误消息。

```
SQLPLUS>EXECUTE DBMS_LOGMNR_D.BUILD (
```

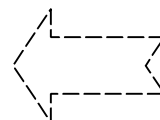
```
SQLPLUS>options => DBMS_LOGMNR_D.STORE_IN_REDO_LOGS);
```

9.3.2 指定要分析的重做日志文件

如果要指定希望分析的重做日志文件，执行 DBMS_LOGMNR.ADD_LOGFILE 过程即可，步骤如下所示。可以按照任意的次序添加和删除日志文件。

①使用 SQL*Plus 来启动 Oracle 实例，数据库可以装载，也可以不装载。例如，键入以下内容：

```
STARTUP
```



②通过指定 DBMS_LOGMNR.ADD_LOGFILE 过程的 NEW 选项来创建重做日志文件的列表。例如，键入以下内容来指定/oracle/logs/log1.f:

```
EXECUTE DBMS_LOGMNR.ADD_LOGFILE (
LOGFILENAME => '/oracle/logs/log1.f',
OPTIONS => DBMS_LOGMNR.NEW);
```

③如果需要，可以通过指定 DBMS_LOGMNR.ADD_LOGFILE 过程的 ADDFILE 选项来添加更多的重做日志文件。例如，键入以下内容来添加/oracle/logs/log2.f:

```
EXECUTE DBMS_LOGMNR.ADD_LOGFILE (
LOGFILENAME => '/oracle/logs/log2.f',
OPTIONS => DBMS_LOGMNR.ADDFILE);
```

④如果需要，还可以通过指定 DBMS_LOGMNR.ADD_LOGFILE 过程的 REMOVEFILE 选项来删除重做日志文件。例如，键入以下内容来删除/oracle/logs/log2.f:

```
EXECUTE DBMS_LOGMNR.ADD_LOGFILE (
LOGFILENAME => '/oracle/logs/log2.f',
OPTIONS => DBMS_LOGMNR.REMOVEFILE);
```

9.3.3 启动 LogMiner

在创建了字典文件并指定了要分析的重做日志文件之后，可以启动 LogMiner 并开始分析。按如下步骤进行操作：

①执行 DBMS_LOGMNR.START_LOGMNR 过程来启动 LogMiner。

建议指定字典选项。否则，LogMiner 将不能把内部对象的标识符和数据类型称作对象名称和外部数据格式。此时，LogMiner 将返回内部对象 ID 以及用十六进制字节表示的数据。

注意如果正在指定平面文件字典的名称，那么就必须为字典文件提供一个完全合格的文件名。例如，如果要使用/oracle/database/dictionary.ora 来启动 LogMiner，可以发布以下命令：

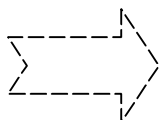
```
EXECUTE DBMS_LOGMNR.START_LOGMNR (
DICTFILENAME => '/oracle/database/dictionary.ora');
```

如果没有指定平面文件字典名称，那么指定 DICT_FROM_REDO_LOGS 或 DICT_FROM_ONLINE_CATALOG 选项。

②还可以设置 startTime 和 endTime 参数来根据时间筛选数据。这个过程需要日期值。使用 TO_DATE 函数来指定日期和时间，如下面这个例子所示：

```
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'));
不用时标来推断重做记录的次序。可以通过使用 SCN 来推断重做记录的次序。
```

③如果不指定开始时间和结束时间，也可以使用 SCN 的 startScn 和 endScn 参数来筛选



数据，如下面这个例子所示：

```
EXECUTE DBMS_LOGMNR.START_LOGMNR (
  DICTFILENAME => '/oracle/dictionary.ora',
  STARTSCN => 100,
  ENDSCN => 150);
```

在同时指定了 `startTime` 和 `endTime` 这两个参数的情况下，`startScn` 和 `endScn` 两个参数将分别重写 `startTime` 和 `endTime` 这两个参数的值。

如果没有指定开始或结束参数，对于每一个发布的 `SELECT` 语句来说，都将从头至尾读取整个日志文件。

④如果需要，还可以使用 `OPTIONS` 参数来设置以下会话特性：

9.3.3.1 COMMITTED_DATA_ONLY

在 `V$LOGMNR_CONTENTS` 视图中，只显示已提交事务的行。这个特性能够筛选出回滚事务以及进程中的事务。

如果正在分析的是重做日志文件中长时间运行的事务，那么使用这个选项将引起“Out of Memory”（内存溢出）这样的错误。

默认情况下，LogMiner 的行对应显示所有事务的信息。

9.3.3.2 SKIP_CORRUPTION

在 `V$LOGMNR_CONTENTS` 视图进行选择操作的过程中，会跳过重做日志文件的任何遭到损坏的地方。在信息损坏后重新找回的行都会标记为“Log File Corruption Encountered”（遇到日志文件损坏）消息。另外，对于每一个受到损坏的重做记录，都将返回一行信息，该信息指出跳过多少块。

对于选择的操作，如果遇到日志文件中的第一个信息损坏，默认情况下都会中断操作。

9.3.3.3 DDL_DICT_TRACKING

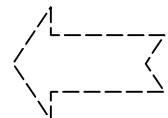
如果使用的字典是来自于平面文件或重做日志文件，那么在重做日志文件中找到 DDL 事件时，LogMiner 保证它的内部字典已经更新。这样对于在 LogMiner 内部字典构建后重做日志文件中修改过的对象来说，也就确保了 `SQL_REDO` 和 `SQL_UNDO` 信息的正确性。

这个选项默认情况下是不允许的。

如果设置了 `DICT_FROM_ONLINE_CATALOG` 选项，该选项就是无效的。

9.3.3.4 NO_DICT_RESET_ONSELECT

该选项仅在指定了 `DDL_DICT_TRACKING` 选项的情况下有效。在每一次 `V$LOGMNR_CONTENTS` 视图上的选择操作开始的时候，它可以避免 LogMiner 重新装载它的字典。这是非常有利的，因为如果 DDL 操作已经更新了内部的 LogMiner 字典，那么会花费一定的时间进行字典的更新。然而，应该意识到，因为使用了该选项，如果字典还没有更新，那么得到关于重做日志文件中修改过对象的不正确的 `SQL_REDO` 和 `SQL_UNDO` 信息是不正确的。



如果既然从随后的选择中返回全部重构的 SQL 语句，那么该选项就不能选。

在下面的例子中，显示了当指定 NO_DICT_RESET_ONSELECT 和 DDL_DICT_TRACKING 选项的时候 LogMiner 是如何做出反应的。

①启动 LogMiner，指定 NO_DICT_RESET_ONSELECT 和 DDL_DICT_TRACKING 选项，如下所示：

```
execute DBMS_LOGMNR.START_LOGMNR(OPTIONS =>
DBMS_LOGMNR.DDL_DICT_TRACKING +
DBMS_LOGMNR.NO_DICT_RESET_ONSELECT +
DBMS_LOGMNR.DICT_FROM_REDO_LOGS);
```

②发布以下 SQL 查询：

```
SELECT sql_redo FROM SYS.V$logmnr_contents;
```

③返回的 SQL_REDO 如下所示：

```
SQL_REDO
-----
create table scott.customer(name varchar2(32), phone_day varchar2(20),
phone_evening varchar2(20))
insert into "SCOTT"."CUSTOMER"("NAME","PHONE_DAY","PHONE_EVENING")
values ('Nadine Gordimer','847-123-1234','415-123-1234')
insert into "SCOTT"."CUSTOMER"("NAME","PHONE_DAY","PHONE_EVENING")
values ('Saul Bellow','847-123-1234','415-123-1234');
commit;
alter table scott.customer drop (phone_evening)
insert into "SCOTT"."CUSTOMER"("NAME","PHONE_DAY") values ('Gabriel
Garcia Marquez','044-1270-123-1234');
commit;
```

该 SELECT 语句正确地将 CREATE TABLE 和 ALTER TABLE 语句应用于 LogMiner 的内部字典并且重新构建了有效的 SQL_REDO。

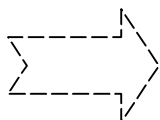
在这项选择操作结束时，因为 ALTER TABLE 语句撤消了 phone_evening 列，所以内部字典包含的 scott.customer 表的定义中仅有两列。

④再次发布与上次完全一样的 SQL 查询：

```
SELECT sql_redo FROM SYS.V$logmnr_contents;
```

⑤返回的 SQL_REDO 如下所示：

```
SQL_REDO
-----
create table scott.customer(name varchar2(32), phone_day varchar2(20),
phone_evening varchar2(20))
insert into "SCOTT"."CUSTOMER"("COL 1","COL 2","COL 3") values
(HEXTORAW('4e6164696e6520476f7264696d6572'),HEXTORAW('3834372d31323332d313233
```



```

34'),
HEXTORAW('3431352d3132332d31323334'));
insert into "SCOTT"."CUSTOMER" ("COL 1", "COL 2", "COL 3") values
(HEXTORAW('5361756c2042656c6c6677'),HEXTORAW('3834372d3132332d31323334'),
HEXTORAW('3431352d3132332d31323334'));
commit;
alter table scott.customer drop (phone_evening)
insert into "SCOTT"."CUSTOMER" ("NAME", "PHONE_DAY") values ('Gabriel
Garcia Marquez', '044-1270-123-1234');
commit;

```

因为当 LogMiner 启动时指定了 NO_DICT_RESET_ONSELECT 选项，所以当执行第 2 次的 SELECT 语句时，LogMiner 并不重新装载它的字典。因此，更新过的字典并不用于翻译重做流，而且该 SELECT 操作不能完全等同于最前面的两个 INSERT 语句。相反，这两个 INSERT 语句是用十六进制形式的 SQL_REDO 显示的。但是，可以完全等用于第 3 个 INSERT 语句，因为它符合 LogMiner 字典中的 scott.customer 表的定义。

因此，如果希望完全重新构建随后的选择操作返回的 SQL 语句，那么就不应该指定该选项。

9.3.3.5 DICT_FROM_ONLINE_CATALOG

如果设置了这个选项，LogMiner 将采用数据库当前使用的字典。如果指定了 DDL_DICT_TRACKING 选项，该选项就无效。

9.3.3.6 DICT_FROM_REDO_LOGS

如果设置了这个选项，LogMiner 会期望在 DBMS_LOGMNR.ADD_LOGFILE 过程指定的重做日志文件中找到一个字典。

要查看哪个重做日志文件包含字典，可以看一下 V\$LOGMNR_LOGS 视图。

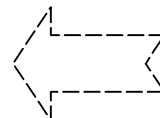
如果希望在特定时期分析重做日志文件，就必须使用与该时期开始时使用的字典一致的字典。如果已经执行了 DDL 操作，比如撤消列或表，那么字典也许不会与那些 DDL 操作之前创建的重做日志文件中的数据同步，认识到这一点非常重要。

9.3.4 分析 V\$LOGMNR_CONTENTS 视图中的输出

可以在 V\$LOGMNR_CONTENTS 视图中查看 LogMiner 输出。在 LogMiner 启动后，可以在命令行发布 SQL 语句来查询 V\$LOGMNR_CONTENTS 视图中包含的数据。

当执行了某个基于 V\$LOGMNR_CONTENTS 视图的 SQL 选择操作时，将顺序读取重做日志文件。从重做日志文件中获得的记录将返回到 V\$LOGMNR_CONTENTS 视图的行中。这样直到满足启动时 (endTime 或 endScn) 指定的筛选标准或者到达重做日志文件的末尾为止。

除非已经使用 COMMITTED_DATA_ONLY 选项来指定只重新找回已提交的事务，否则 LogMiner 将按 SCN 次序返回所有的行。



SCN 次序是应用在介质恢复过程中的标准次序。

以下示例查询将返回有关操作的信息：

```
SELECT operation, sql_redo FROM V$LOGMNR_CONTENTS;
OPERATION SQL_REDO
-----
INTERNAL
INTERNAL
START set transaction read write;
UPDATE update SYS.UNDO$ 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.
```

9.3.5 使用 LogMiner 执行对象级的恢复操作

LogMiner 处理重做日志文件，将文件中的内容称作 SQL 语句，这些语句代表数据库执行的逻辑操作。然后，V\$LOGMNR_CONTENTS 视图就会列出重新构建的 SQL 语句，这些语句代表原来的操作（SQL_REDO 列），以及对应撤消操作（SQL_UNDO 列）的 SQL 语句。

如果完全了解并考虑现有的依托关系，可以用 SQL_UNDO 语句来回滚原来对数据库所做的修改。

9.3.6 结束 LogMiner 会话

要正确地结束 LogMiner 会话，可以使用 DBMS_LOGMNR.END_LOGMNR 进程，如下所示：

```
EXECUTE DBMS_LOGMNR.END_LOGMNR;
```

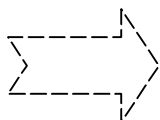
该进程将关闭所有日志文件并允许释放所有由 LogMiner 分配的数据库和系统资源。

如果没有执行该进程，LogMiner 将保留所有它分配的资源，直到启动 LogMiner 的 Oracle 会话结束为止。如果使用了 DDL_DICT_TRACKING 选项或者 DICT_FROM_REDO_LOGS 选项，那么使用该进程来结束 LogMiner 的方法就特别重要。

9.4 使用 LogMiner 的例子

这一节提供以下使用 LogMiner 的例子。

- 例子：跟踪某个特定用户实施的修改



- 例子：计算表访问统计表

9.4.1 例子：跟踪某个特定用户实施的更改

在这个例子中，将会看到在某个特定时间范围内由用户 joedevo 对数据库所做的所有修改。按如下步骤执行这项操作：

- 步骤 1：创建字典文件
- 步骤 2：添加重做日志文件并确定搜索范围
- 步骤 3：启动 LogMiner 并分析数据

9.4.1.1 步骤 1：创建字典文件

要使用 LogMiner 来分析用户 joedevo 的数据，就必须在启动 LogMiner 之前创建一个字典文件。按如下步骤进行操作：

- ①在 init.ora 文件中，将初始化参数 UTL_FILE_DIR 设置为 /user/local/dbs：

```
UTL_FILE_DIR = /user/local/dbs
```

- ②启动 SQL*Plus，然后连接数据库：

```
CONNECT SYSTEM/password
```

- ③打开数据库以便创建字典文件：

```
STARTUP
```

- ④将字典命名为 orcldict.ora 并把它放在 /user/local/dbs 目录中：

```
EXECUTE DBMS_LOGMNR.D.BUILD(
DICTIONARY_FILENAME => 'orcldict.ora',
DICTIONARY_LOCATION => '/usr/local/dbs');
```

- ⑤这样就创建了字典，并且可以用于以后的操作中。可以关闭数据库：

```
SHUTDOWN;
```

9.4.1.2 步骤 2：添加重做日志文件并确定搜索范围

上一步，字典创建完毕，此时就可以查看在特定时间进行的修改。按如下步骤进行操作：

- ①启动 SQL*Plus，按 SYSTEM 身份连接，然后启动实例：

```
CONNECT SYSTEM/password
```

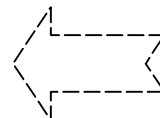
```
STARTUP NOMOUNT
```

- ②提供要分析的日志文件的列表。设置标记 Options 来表明这是一个新的列表：

```
EXECUTE DBMS_LOGMNR.ADD_LOGFILE(
LOGFILENAME => 'log1orcl.ora',
OPTIONS => DBMS_LOGMNR.NEW);
```

- ③往现有列表中添加文件。设置标记 OPTIONS 来表明正在往现有列表中添加文件：

```
EXECUTE DBMS_LOGMNR.ADD_LOGFILE(
LOGFILENAME => 'log2orcl.ora',
OPTIONS => DBMS_LOGMNR.ADDFILE);
```



9.4.1.3 步骤 3: 启动 LogMiner 并分析数据

此时, 可以使用 V\$LOGMNR_CONTENTS 视图进行查询。如果决定找到用户 joedevo 对薪水表所做的所有修改, 就会发现 joedevo 进行了两个操作: 删除原来的薪水, 然后添加新的、更高的薪水。现在, 就拥有了撤消该操作所必需的数据。按如下步骤进行操作:

①启动 LogMiner 并限制在指定时间范围内进行搜索:

```
EXECUTE DBMS_LOGMNR.START_LOGMNR (
  DICTFILENAME => 'orcldict.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'));
```

②查询 V\$LOGMNR_CONTENTS 视图来查看搜索结果:

```
SELECT sql_redo, sql_undo FROM V$LOGMNR_CONTENTS
WHERE USERNAME = 'joedevo' AND TABLENAME = 'salary';
```

③对于 SQL_REDO 和 SQL_UNDO 两列来说, 返回了两行 (显示的数据格式与屏幕上看到的有所不同):

```
SQL_REDO SQL_UNDO
-----
delete * from SALARY insert into SALARY (NAME,EMPNO, SAL)
where EMPNO = 12345 values ('JOEDEVO', 12345,500)
and ROWID = 'AAABOOAABAAEPCABA';
insert into SALARY (NAME, EMPNO, SAL) delete * from SALARY
values ('JOEDEVO',12345,2500) where EMPNO = 12345
and ROWID = 'AAABOOAABAAEPCABA';
2 rows selected
```

9.4.2 例子: 计算表访问统计信息

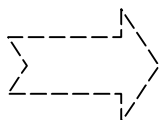
在这个例子中, 管理了一个直销数据库, 并且想弄清楚, 在 8 月份的两周内, 客户合同在生成收入方面有多大成效。假设已经创建了字典并添加了希望搜索的重做日志文件。按如下步骤进行操作:

①启动 LogMiner 并指定一个时间范围:

```
EXECUTE DBMS_LOGMNR.START_LOGMNR (
  STARTTIME => TO_DATE('07-Aug-1998 08:30:00', 'DD-MON-YYYY HH:MI:SS')
  ENDTIME => TO_DATE('15-Aug-1998 08:45:00', 'DD-MON-YYYY HH:MI:SS'),
  DICTFILENAME => '/usr/local/dict.ora');
```

②查询 V\$LOGMNR_CONTENTS 视图来确定在指定时间范围内哪些表进行了修改, 如本例中所示。(该查询筛选出名称中有 \$ 的系统表。)

```
SELECT seg_owner, seg_name, count(*) AS Hits FROM
V$LOGMNR_CONTENTS WHERE seg_name NOT LIKE '%$%' GROUP BY
seg_owner, seg_name;
```



③下面就是显示出的数据（适当地格式化了）：

```
SEG_OWNER SEG_NAME Hits
```

```
-----
```

```
CUST ACCOUNT 384
```

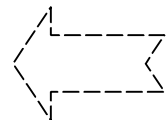
```
SCOTT EMP 12
```

```
SYS DONOR 12
```

```
UNIV DONOR 234
```

```
UNIV EXECDONOR 325
```

```
UNIV MEGADONOR 32
```



第 10 章 管理作业队列

本章描述如何使用作业队列来调度用户作业的定期执行，主要包含以下内容：

- 启动执行作业的进程
- 管理作业队列
- 查看作业队列信息

10.1 启动执行作业的进程

可以使用作业队列调度程序（作业）的定期运行。要调度提交到作业队列中的作业，可以使用 Oracle 提供的 DBMS_JOBS 包，并指定作业运行的频率。另外还有一些功能允许更改、禁用或删除先前提交的作业。

作业队列（Jnnn）进程执行作业队列中的作业。对于每一个实例，这些作业队列进程都是由作业队列协调程序（CJQ0）后台进程动态派生的。该协调程序定期从 DBA_JOBS 视图中选出准备运行的作业，并且将这些作业按时间进行排序，然后派生出 Jnnn 进程来运行选中的作业。每一个 Jnnn 进程执行选中作业中的一个。

初始化参数 JOB_QUEUE_PROCESSES 控制实例能否启动一个协调作业队列程序进程。如果这个参数设置为 0，就表明在数据库启动时不启动任何作业队列协调程序进程，并且因此不执行任何作业队列中的作业。初始化参数 JOB_QUEUE_PROCESSES 还可以指定在一个实例上能够并发运行 Jnnn 进程的最大数目。可以指定的最大进程数目是 1000。

下面的初始化参数设置是在数据库启动时启动作业队列协调程序进程，并且派生出最多 60 个的并发 Jnnn 进程。

```
JOB_QUEUE_PROCESSES = 60
```

在作业队列协调程序进程浏览 DBA_JOBS 视图中的作业的任何时候，该进程都会派生出执行选中作业所需的最多的 Jnnn 进程。在上面的这个例子中，允许并发 60 个 Jnnn 进程，如果仅选择执行 20 个作业，那么协调程序将仅仅派生或重用执行 20 个作业所需的 Jnnn 进程数目（至少 20 个）。任何空闲的现有 Jnnn 进程都可以用于重用过程。

当 Jnnn 进程执行完某个作业时，它将轮询另一个要执行的作业。如果没有要执行的选中作业，那么该进程将进入空闲状态，但是会再次定期轮询。在轮询了预定次数后，如果它还没有找到要执行的作业，那么该进程将终止。

初始化参数 JOB_QUEUE_PROCESSES 是动态的，并且可以用 ALTER SYSTEM 语句来进行修改。例如，下面的语句将并发 Jnnn 进程的最大数目设置为 20。

```
ALTER SYSTEM SET JOB_QUEUE_PROCESSES = 20;
```

如果这个新设的值比前面的设置要低并且少于并发执行的 Jnnn 进程数目，那么超出的进程终止之前允许完成进程任务。

如果实例运行在受限模式，那么 Jnnn 进程将不执行作业。

10.2 管理作业队列

这一节描述管理作业队列的各个方面，主要包含以下内容：

- DBMS_JOB 包
- 将作业提交到作业队列
- 作业如何执行
- 从作业队列中删除作业
- 更改作业
- 损坏的作业
- 强制执行作业
- 终止作业

10.2.1 DBMS_JOB 包

要调度和管理作业队列中的作业，可以使用 DBMS_JOB 包中的过程。使用作业队列不需要数据库权限。任何可以执行作业队列过程的用户都可以使用作业队列。

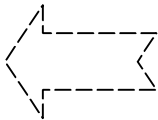
下面是 DBMS_JOB 包中的过程。这一节简单描述一下这些过程。

过程	说明
SUBMIT	将作业提交到作业队列。
REMOVE	从作业队列中删除指定的作业。
CHANGE	对已经提交到作业队列中的指定作业进行修改。可以更改该作业的说明，作业运行的时间或者执行作业的间隔。
WHAT	对指定的作业修改作业说明。
NEXT_DATE	修改指定作业的下一执行时间。
INTERVAL	修改指定作业的执行间隔。
BROKEN	设置或重新设置作业损坏标记。如果作业标记为损坏，那么 Oracle 不能执行该作业。
RUN	强制执行某个指定的作业。

10.2.2 将作业提交到作业队列

要将新作业提交到作业队列，可以使用 DBMS_JOB 包中的 SUBMIT 过程。需要指定 SUBMIT 过程的以下参数：

参数	说明
JOB	它是一个输出参数。是分配给正在创建的作业的标识符。无论何时更改或删除该作业，都必需使用这个作业号码。
WHAT	这是希望执行的 PL/SQL 代码。
NEXT_DATE	这是作业下一次运行的日期。默认值是 SYSDATE。
INTERVAL	这是用来计算下一次作业执行时间的日期函数。默认值为 NULL。该参数必须估算出下一次执行作业的时间或设置为 NULL。
NO_PARSE	这是一个标记。如果该参数设置为 FALSE（默认值），那么 Oracle 将分析与作业相关联的过程。如果该参数设置为 TRUE，那么 Oracle 将分析与首次执行的作业相关联的过程，例如，如果希望在创建与作业相关联的表完成之前提交作业，就需要将 NO_PARSE 参数设置为 TRUE。



例如，下面的语句将会把新作业提交到作业队列中去。作业调用过程 DBMS_DDL.ANALYZE_OBJECT 为表 dquon.accounts 生成优化程序统计表。该统计表基于 accounts 表的一半行数的示例而建立起来。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
```

10.2.2.1 作业环境

当把作业提交到作业队列或者更改作业的定义时，Oracle 记录了以下环境特征：

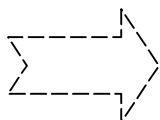
- 当前用户
- 提交或更改作业的用户
- 当前模式（如果发布了 ALTER SESSION SET CURRENT_SCHEMA 语句，当前模式可以与当前用户或提交用户的模式有所不同）
- MAC 权限（如果使用 Oracle Label Security 时）

Oracle 还记录以下 NLS 参数：

- NLS_LANGUAGE
- NLS_TERRITORY
- NLS_CURRENCY
- NLS_ISO_CURRENCY
- NLS_NUMERIC_CHARACTERS
- NLS_DATE_FORMAT
- NLS_DATE_LANGUAGE
- NLS_SORT

每次执行作业时，Oracle 都会还原所有这些环境特征。NLS_LANGUAGE 和 NLS_TERRITORY 两个参数决定未加指定的 NLS 参数的默认值。

可以通过使用 DBMS_SQL 包以及 ALTER SESSION 语句的方法来改变作业的环境。



10.2.2.2 作业和导入/导出

可以导入和导出作业。因此，如果在某个数据库中定义了一个作业，那么可以将它传送到另一个数据库。当导出和导入作业时，作业的号码、环境和定义将保持不变。

注意：如果想导入的作业的作业号与现有数据库中已经存在的某个作业的作业号相同，那么将不允许导入这个作业。该作业在数据库中将被作为新作业进行提交。

10.2.2.3 作业所有者

当把作业提交到作业队列中时，Oracle 将把提交者人标识为该作业的所有者。只有作业的所有者才能对该作业进行更改，强制运行或删除操作。

10.2.2.4 作业号码

队列中的作业用作业号码进行标识。当提交作业时，它的作业号码将从 SYS.JOBSEQ 序列中自动生成。一旦给作业分配了作业号码，就不能再改变这个号码。即使在导出和导入该作业时，它的作业号码也会保持不变。

10.2.2.5 作业定义

作业定义是 SUBMIT 过程的 WHAT 参数中指定的 PL/SQL 代码。通常，作业定义是对某个过程的一个调用。这个过程调用可以有任意个数的参数。

注意：在作业定义中，使用两个单引号括住字符串。通常在作业定义的末尾会加一个分号。

在下面的例子中，作业定义是有效的：

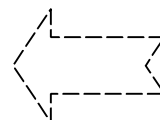
- 'myproc("10-JAN-99", next_date, broken);'
- 'scott.emppackage.give_raise("JFEE", 3000.00);'
- 'dbms_job.remove(job);'

注意：不能从一个作业中运行某个作业。否则，将会收到一条错误消息。例如，下面的语句将产生这样的错误消息 "ORA-32317 cannot run a job from another job":

```
declare
jobno number;
begin
dbms_job.submit(jobno, 'dbms_job.run(23587);');
dbms_job.run(jobno);
end;
```

10.2.2.6 作业执行间隔

如果作业应该按设定的间隔定期执行，可以在 INTERVAL 参数中使用类似于 'SYSDATE + 7' 这样的日期表达式。下面是一些常用于作业执行间隔的日期表达式。



日期表达式	赋值
'SYSDATE + 7'	与上一次执行相隔 7 天
'SYSDATE + 1/48'	每半个小时执行一次
'NEXT_DAY(TRUNC(SYSDATE), 'MONDAY') + 15/24'	在每个星期一的下午 3 点执行
'NEXT_DAY(ADD_MONTHS(TRUNC(SYSDATE), 'Q'), 3), 'THURSDAY')'	每季度的第一个星期四

注意：当指定 NEXT_DATE 或 INTERVAL 时，记住日期文字和字符串必须用单引号括起来。而且，INTERVAL 的值也必须用单引号括起来。

在执行作业之前，INTERVAL 日期函数将立即求出值。当作业成功完成时，从 INTERVAL 计算出的日期将成为新的 NEXT_DATE 的值。例如，如果在星期一设置执行间隔为 'SYSDATE + 7'，但是因为某种原因没有执行作业，一直到星期四才执行作业，那么 'SYSDATE + 7' 设置的结果将是在每周四执行作业，而不是每周一执行作业。如果 INTERVAL 日期函数求出的值为 NULL，并且作业成功完成，那么将从作业队列中删除这个作业。

如果一直希望在某个指定的时间自动执行作业，而不管上次执行是在什么时候（例如，每个星期一），INTERVAL 和 NEXT_DATE 参数都将指定类似于 'NEXT_DAY(TRUNC(SYSDATE), 'MONDAY')' 这样的日期表达式。

10.2.2.7 数据库链接和作业

如果要提交使用了数据库链接的作业，那么这个链接就必须包括用户名和密码。如果使用的是匿名的数据库链接，那么提交将不会成功。

10.2.3 作业如何执行

Jnnn 进程执行作业。要执行某个作业，进程会创建一个会话程序来运行它。当 Jnnn 进程执行某个作业时，作业运行在提交该作业的同一个环境中并拥有作业所有者的默认权限。所有者必须为作业定义中引用的所有对象直接授予所需的对象权限。

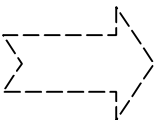
当使用 DBMS_JOB.RUN 过程强制执行作业时，作业将由用户进程运行并且只拥有所有者的默认权限。通过角色授予作业所有者的权限都是不可用的。必须为作业定义中引用的所有对象直接授予所需的对象权限。

10.2.3.1 作业队列锁

Oracle 使用作业队列锁来确保每次作业只在一个会话中执行。当正在运行某个作业时，它的会话需要为作业分配一个作业队列锁。可以使用数据字典中的锁视图来查看会话程序当前锁的有关信息。

下面的查询列出了所有拥有 JQ 锁的会话的标识符、锁类型和锁标识符：

```
SELECT SID, TYPE, ID1, ID2
FROM V$LOCK
WHERE TYPE = 'JQ';
SID TY ID1 ID2
```



```
-----
12 JQ 0 14144
```

```
1 row selected.
```

在上面的查询中，拥有这种锁的会话的标识符是 12。对于 JQ 锁来说，ID1 列通常都是 0。ID2 列是会话程序正在运行的作业的作业号码。这个锁视图可以与 DBA_JOBS_RUNNING 视图连接起来，以获取更多的作业信息。

10.2.3.2 作业执行错误

当作业失败时，有关失败的信息将记录在跟踪文件和报警日志中。Oracle 将写入编号为 ORA-12012 的消息并在失败信息中包括失败作业的作业号码。

在以下情况将不能成功执行队列中的作业：

- 网络失败或实例失败
- 当执行作业时发生异常情况

当 Oracle 正在试着执行作业时，如果作业返回一条错误消息，那么 Oracle 将再次尝试执行该作业。第一次尝试是在一分钟之后，第二次尝试是在两分钟以后，第三次尝试是在四分钟以后，以此类推，每次尝试都间隔双倍的时间。如果作业失败了 16 次，那么 Oracle 将会自动标记该作业为损坏并且不再继续尝试执行该作业。但是，在不同的尝试之间，仍然有机会纠正妨碍作业运行的问题。这样不会打乱再次尝试的循环，并且最终 Oracle 将会再次尝试运行该作业。

10.2.4 从作业队列中删除作业

要从作业队列中删除作业，可以使用 DBMS_JOB 包中的 REMOVE 过程。

下面的语句将从作业队列中删除作业号码为 14144 的作业：

```
DBMS_JOB.REMOVE(14144);
```

限制：

- 可以从作业队列中删除当前正在执行的作业。但是，作业将不会终止，而且还会完成当前的执行任务。
- 只能删除自己所有的作业。如果试图删除不属于自己的作业，那么将会收到一条消息，该消息表明要删除的作业不在作业队列中。

10.2.5 更改作业

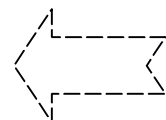
要更改已经提交到作业队列中的作业，可以使用 DBMS_JOB 包中的 CHANGE、WHAT、NEXT_DATE 或者 INTERVAL 过程。

限制：

- 只能改变自己所有的作业。如果试图改变不属于自己的作业，那么将会收到一条消息，该消息表明要删除的作业不在作业队列中。

10.2.5.1 CHANGE 过程

通过调用 DBMS_JOB.CHANGE 过程可以改变任何与作业相关联的用户可定义的参



数。

在这个例子中，标识符为 14144 的作业将变成每 3 天执行一次：

```
DBMS_JOB.CHANGE(14144, NULL, NULL, 'SYSDATE + 3');
```

当调用 DBMS_JOB.CHANGE 过程时,如果将 WHAT、NEXT_DATE 或者 INTERVAL 指定为 NULL，那么这个当前值将保持不变。

注意：当使用 DBMS_JOB 包中的 WHAT 参数改变作业的定义时，Oracle 将记录当前的环境。而记录的环境将成为作业的新环境。

10.2.5.2 WHAT 过程

通过调用 DBMS_JOB.WHAT 过程可以改变作业的定义。下面的例子将改变标识符为 14144 的作业的定义：

```
DBMS_JOB.WHAT(14144, 'scott.emppackage.give_raise(''RBAYLIS'', 6000.00);'
```

注意：当执行 DBMS_JOB.WHAT 过程时，Oracle 将记录当前的环境。而记录的环境将成为作业的新环境。

10.2.5.3 NEXT_DATE 过程

通过调用 DBMS_JOB.NEXT_DATE 过程可以改变作业下一次执行的时间，举例如下：

```
DBMS_JOB.NEXT_DATE(14144, 'SYSDATE + 1');
```

10.2.5.4 INTERVAL 过程

下面的例子举例说明了通过调用 DBMS_JOB.INTERVAL 过程来改变执行间隔的方法：

```
DBMS_JOB.INTERVAL(14144, 'NULL');
```

在这个例子中，成功执行作业之后将不再运行该作业。

10.2.6 损坏的作业

可以把作业标记为损坏或未损坏。Oracle 不会尝试运行已经损坏的作业。但是，可以通过调用过程 DBMS_JOB.RUN 来强制运行已经损坏的作业。

10.2.6.1 作业如何遭到损坏

当提交作业时，作业被认为是未损坏的。

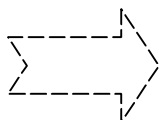
作业遭到损坏有两种方式：

- Oracle 尝试了 16 次去执行作业，并以失败告终。
- 使用 DBMS_JOB.BROKEN 过程把作业标记为损坏：

```
DBMS_JOB.BROKEN(14144, TRUE)
```

一旦作业已经标记为损坏，那么直到该作业标记为未损坏，或者通过调用 DBMS_JOB.RUN 过程来强制执行该作业，Oracle 才会尝试执行该作业。

在下面的例子中，将标识符为 14144 的作业标记为没有损坏并将它的下一次执行日期设置成星期一：



```
DBMS_JOB.BROKEN(14144, FALSE, NEXT_DAY(SYSDATE, 'MONDAY'));
```

限制:

- 只能将自己的作业标记为损坏。如果调用 DBMS_JOB.BROKEN 过程为不属于自己的作业进行标记, 那么将收到一条消息, 该消息表明要标记的作业不在作业队列中。

10.2.6.2 运行已经损坏的作业

如果因为某个问题导致 16 次执行作业失败, Oracle 将会标记该作业为损坏。一旦确定出了问题, 就可以通过以下方式之一运行该作业:

- 通过调用 DBMS_JOB.RUN 过程来强制运行该作业
- 通过调用 DBMS_JOB.BROKEN 过程将该作业标记为没有损坏, 并且等待 Oracle 执行这个作业

如果通过调用 DBMS_JOB.RUN 过程, 那么 Oracle 将立即运行该作业。如果作业执行成功, Oracle 会将该作业标记为没有损坏, 并且将执行作业的失败次数重新设置为 0。

一旦重新设置了作业的损坏标记 (通过调用 RUN 或 BROKEN 过程), 作业将继续按照为作业预先设定的执行间隔设置执行。

10.2.7 强制执行作业

可能很多时候要手动执行作业。例如, 如果已经改正了一个已经损坏的作业, 那么也许希望通过强制执行手段立即测试该作业。要强制作业立即执行, 可以使用 DBMS_JOB 中的 RUN 过程。

当使用 DBMS_JOB.RUN 过程运行作业时, Oracle 将重新计算下一次的执行日期。例如, 如果在星期一用 NEXT_DATE 的值 'SYSDATE' 和 INTERVAL 的值 'SYSDATE + 7' 创建了一个作业, 那么每隔 7 天在星期一将运行该作业。但是, 如果在星期三执行了 RUN 过程, 那么下一次执行日期将设置成下一个星期三。

下面的语句将使得在会话中运行标识符为 14144 的作业, 并且重新计算下一次执行的日期:

```
DBMS_JOB.RUN(14144);
```

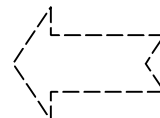
注意: 当强制运行某作业时, 该作业将在当前的会话中执行。作业的运行将重新初始化会话程序包。

限制:

- 只能运行自己的作业。如果运行不属于自己的作业, 那么将会收到一条消息, 该消息表明要运行的作业不在作业队列中。
- RUN 过程包含隐式提交。一旦使用 RUN 过程执行了作业, 那么将不能回滚该作业。

10.2.8 终止作业

可以通过将作业标记为损坏、识别正在运行作业的会话程序以及解除该会话连接的方法来终止正在运行的作业。应该将作业标记为损坏, 这样 Oracle 就不会再次试图运行该作



业。

识别了正在运行作业的会话后（使用 V\$SESSION 或 V\$LOCK），可以使用 SQL 语句 ALTER SYSTEM 解除会话连接。查看有关作业和会话的信息的例子，见下一节“查看作业队列信息”。

10.3 查看作业队列信息

可以使用下面列出的数据字典视图来查看作业队列中的作业信息：

视图	说明
DBA_JOBS	DBA 视图描述了数据库中的所有作业。ALL 视图描述了当前用户可以访问的所有作业。USER 视图描述了属于当前用户的所有作业。
ALL_JOBS	
USER_JOBS	
DBA_JOBS_RUNNING	列出数据库中当前运行的所有作业。这个视图可以与 V\$LOCK 视图连接，以便识别出已经上锁的作业。

10.3.1 显示有关作业的信息

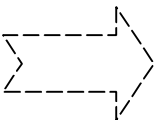
下面的查询将创建一个有关已经提交的作业信息的列表，该列表中包含每一个这种作业的作业号码、下一次执行时间、失败次数以及是否损坏的状态：

```
SELECT JOB, NEXT_DATE, NEXT_SEC, FAILURES, BROKEN
FROM USER_JOBS;
JOB NEXT_DATE NEXT_SEC FAILURES B
-----
9125 01-JUN-01 00:00:00 4 N
14144 24-OCT-01 16:35:35 0 N
41762 01-JUN-01 00:00:00 16 Y
3 rows selected.
```

10.3.2 显示有关正在运行的作业的信息

可以只显示有关当前正在运行的作业的信息。下面的查询为所有这些作业列出了会话的标识符、作业号码、提交作业的用户以及开始运行的时间：

```
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.1111
```



第 11 章 管理表空间

本章介绍表空间管理的各个方面，包括以下内容：

- 管理表空间的准则
- 创建表空间
- 管理表空间分配
- 修改表空间可用性
- 使用只读表空间
- 取消表空间
- 使用 DBMS_SPACE_ADMIN 解决表空间问题
- 在数据库之间传输表空间
- 查看表空间信息

11.1 管理表空间的准则

在使用一个 Oracle 数据库的表空间之前，应先熟悉以下章节所提供的一些准则：

- 使用多个表空间
- 指定表空间默认的存储参数
- 为用户指定表空间限额

11.1.1 使用多个表空间

使用多个表空间允许用户在执行数据库操作时有更大的灵活性。例如，当一个数据库有多个表空间时，可以执行下列任务：

- 将用户数据从数据字典数据中分离出来，以减少在同一个数据文件的字典对象和模式对象之间的竞争。
- 将一个应用程序的数据与另一个的分离，防止如果一个表空间必须被脱机，多个应用程序都受到影响。
- 在不同的磁盘驱动器上存储不同表空间的数据文件，以减少 I/O 竞争。
- 将回滚段数据从用户数据中分离出来，防止单个磁盘的故障造成数据的永久丢失。
- 当其他表空间保持联机时，使单个表空间脱机，提供更好的整体可用性。
- 为一个特定类型的数据库使用保留一个表空间，比如很快的更新动作、只读动作，或者临时段的存储。这使得用户可以优化表空间的使用。
- 备份单独的表空间。

一些操作系统对可以同时打开的文件数设置了一个限制。这些限制会影响可以同时联机的表空间数。为避免超过操作系统的限制，有效地计划表空间。只创建满足需求的足够的表空间，并使用尽可能少的文件来创建这些表空间。如果需要增加一个表空间的大小，添加一个或两个较大的数据文件，或者创建设置为自动扩展的数据文件，而不是创建很多较小的数据文件。

根据这些因素检查数据，并决定对于数据库设计需要多少表空间。

11.1.2 指定表空间默认的存储参数

当创建一个新的字典管理的表空间时，可以为表空间中将要创建的对象指定默认的存储参数。指定的存储参数在对象创建时覆盖包含该对象的表空间的默认存储参数。如果在创建对象时，没有指定存储参数，该对象的段自动使用该表空间的默认存储参数。

为一个表空间设置默认的存储参数，以说明该表空间将要包含的典型对象的大小（用户估计这个大小）。当创建一个不常见的或特殊的对象时，可以指定不同的存储参数。也可以在以后对默认的存储参数进行修改。

注意：如果没有为一个新的表空间指定默认的存储参数，Oracle 针对用户操作系统的默认存储参数就成为该表空间的默认存储参数。

11.1.3 为用户指定表空间限额

对将要创建表、簇、物化视图、索引和其他对象的用户，在预计容纳这些对象的段的表空间中，给予他们创建对象和一个限额（空间份额或限制）的权限。安全管理员负责将创建对象所需要的权限给予数据库用户，如果必要的话，并负责为数据库用户指定表空间限额。

11.2 创建表空间

在创建一个表空间之前，必须创建一个数据库来包含它。任何数据库中的第一个表空间，通常是 **SYSTEM** 表空间，并且任何数据库的第一个数据文件是在数据库创建期间，在 **SYSTEM** 表空间中自动分配的。

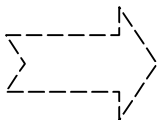
创建表空间的步骤因操作系统不同而不同。然而，在所有的情况中，用户都应该通过操作系统创建一个目录结构，用户的数据文件将被分配在该目录中。在大多数操作系统中，当通过添加数据文件来创建一个新的表空间或修改一个表空间时，应该指出大小和完全指定的文件名。在每一种情况中，都由 Oracle 按所指定的来自动分配和格式化数据文件。

可以创建块大小与初始化参数 **DB_BLOCK_SIZE** 指定的标准数据库块大小不同的的表空间。但是，在 **SGA** 内存中的缓冲区缓存必须按照非标准的块大小来配置。

要创建一个新的表空间，使用 SQL 语句 **CREATE TABLESPACE** 或 **CREATE TEMPORARY TABLESPACE**。用户必须拥有 **CREATE TABLESPACE** 系统权限来创建一个表空间。随后，可以使用 **ALTER TABLESPACE** 或 **ALTER DATABASE** 语句来修改该表空间。相应地，也必须拥有 **ALTER TABLESPACE** 或 **ALTER DATABASE** 系统权限。

在 Oracle8i 以前，所有的表空间都创建为字典管理的。字典管理的表空间依靠数据字典表来跟踪空间应用。从 Oracle8i 开始，用户能够创建本地管理的表空间，它们使用位图（代替了数据字典表）来跟踪已用的和空闲的空间。因为本地管理的表空间有更好的性能，并且更加容易管理，在 Oracle9i 中开始，当盘区管理的类型没有明确指定时，默认的非 **SYSTEM** 的永久表空间是本地管理的。

也可以创建一种称作撤消表空间的特殊表空间。这种表空间是专门设计来包含撤消记



录的。撤消记录是由 Oracle 生成的，用于为了恢复、读一致性而回滚或撤消对数据库所作的改动，或者是应 ROLLBACK 语句要求生成的记录。创建和管理撤消表空间是第 13 章“管理撤消空间”的主题。

永久的和临时的表空间在以下章节中讨论：

- 本地管理的表空间
- 字典管理的表空间
- 临时表空间

11.2.1 本地管理的表空间

本地管理的表空间使用位图跟踪该表空间本身的所有盘区信息，会有如下好处：

- 提高空间操作的速度和并发性，因为空间的分配和回收主要改变本地管理的资源（位图存储在头文件中），而不需要在中心管理的资源（比如 enqueues）。
- 提高性能，因为有时在字典管理的空间分配期间需要的递归操作被取消了。
- 允许可读的备用数据库，因为本地管理的临时表空间（例如，用于排序的）是本地管理的，并且因此不产生任何撤消或重作。
- 简化空间分配——当指定了 AUTOALLOCATE 子句时，会自动选择合适的盘区大小。
- 减少用户对数据字典的依赖，因为必要的信息被存储在文件头和位图块中了。

另外，DBMS_SPACE_ADMIN 包对本地管理的表空间提供维护过程。

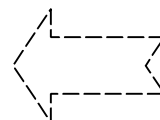
11.2.1.1 创建本地管理的表空间

要创建一个本地管理的表空间，在 CREATE TABLESPACE 语句的 EXTENT MANAGEMENT 子句中指定 LOCAL。用户可以随意地省略 EXTENT MANAGEMENT 子句；本地管理是默认的。然后，用户有两种选择。可以利用 AUTOALLOCATE 选项（默认的）使 Oracle 自动管理盘区，或者可以指定使用一个指定大小（UNIFORM SIZE）的统一盘区来管理该表空间。

注意：当用户没有对所创建的永久的非 SYSTEM 表空间明确指定盘区管理类型时，默认为本地管理的。应用下列规则：

- 如果在 INITIAL = NEXT 和 PCTINCREASE = 0 处指定了默认的存储子句。那么 Oracle 使用统一的盘区大小 = INITIAL 来创建一个统一的本地管理的表空间。
- 如果没有指定默认的存储子句，或者使用 PCTINCREASE 不等于 0 的值和/或 INITIAL 不等于 NEXT 的值指定了，那么 Oracle 使用自动盘区管理（AUTOALLOCATE）来创建一个本地管理的表空间。
- 外部的存储子句（例如，MINEXTENTS 和 MAXEXTENTS）被忽略。
- SYSTEM 表空间始终是字典管理的。

如果表空间被期望用于包含需要不同的盘区大小和拥有很多盘区的大小变动的对象，那么 AUTOALLOCATE 就是最佳选择。对用户来说，如果拥有对空间分配和回收的大量控制是不重要的话，AUTOALLOCATE 给出了一种管理表空间的简化方法。一些空间可能被



浪费了，但是让 Oracle 管理用户空间所带来的好处极有可能远远超过了这个缺点。

另一方面，如果想要准确控制未用的空间，并且能够精确地预计为一个或更多对象要分配的空间和盘区的数目和大小，那么 UNIFORM 是个不错的选择。它确保在表空间中永远不会有一定数量的未用空间。

如下的语句创建一个名为 lmtbsb 的本地管理的表空间，其中 AUTOALLOCATE 子句使 Oracle 自动管理盘区大小。

```
CREATE TABLESPACE lmtbsb DATAFILE '/u02/oracle/data/lmtbsb01.dbf' SIZE 50M
EXTENT MANAGEMENT LOCAL AUTOALLOCATE;
```

另一种选择是可以指定 UNIFORM 子句来创建该表空间。在本例中，盘区大小指定为 128K。在该文件的盘区位图中，每个 128K 的盘区（如果该表空间块大小为 2K，等于 64 个 Oracle 块）由一位（bit）来表示。

```
CREATE TABLESPACE lmtbsb DATAFILE '/u02/oracle/data/lmtbsb01.dbf' SIZE 50M
EXTENT MANAGEMENT LOCAL UNIFORM SIZE 128K;
```

注意：当为本地管理的表空间分配一个数据文件时，应该允许作为用户空间一部分的元数据空间用于空间管理（盘区位图或空间头部段）。例如，如果在指定 UNIFORM 子句时，没有在盘区管理子句中指定参数 SIZE，默认的盘区大小为 1MB。因此，在这种情况下，为数据文件指定的大小必须大于（至少在位图空间上加上一个块）1MB。

11.2.1.2 在本地管理的表空间中指定段空间管理

当使用 CREATE TABLESPACE 语句创建一个本地管理的表空间时，SEGMENT SPACE MANAGEMENT 子句允许指定如何管理在一个段中空闲的和已用的空间。其选择是：

- MANUAL

指定这个关键字通知 Oracle，用户想要使用段中管理空闲空间的空闲列表。空闲列表是有可用于插入行的空间的数据块的列表。MANUAL 是默认值。

- AUTO

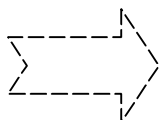
这个关键字通知 Oracle，用户想要使用位图来管理段中的空闲空间。一个位图，在这种情况下，是说明一个段中每个数据块与该块中可用于插入行的空间数量有关的状态的映射。随着数据块中的可用空间增加或减少，它的新状态在该位图中反映出来。位图允许 Oracle 管理空闲空间更加自动化，因而这种空间管理的形式被称作自动段空间管理。

空闲列表是管理段中空闲空间的传统方法。然而，位图提供了一种更简单、更有效的管理段空间的方法。它们提供了更好的空间应用，并且完全消除了为表空间中创建的段指定和调整 PCTUSED、FREELISTS 和 FREELISTS GROUPS 属性的任何必要。如果这些属性应该被指定，那么它们被忽略。

下面的语句使用自动段空间管理来创建表空间 lmtbsb：

```
CREATE TABLESPACE lmtbsb DATAFILE '/u02/oracle/data/lmtbsb01.dbf' SIZE 50M
EXTENT MANAGEMENT LOCAL
SEGMENT SPACE MANAGEMENT AUTO;
```

用户在表空间创建时指定的管理段中可用空间的方法，应用于所有随后在该表空间中



创建的段。另外，选择的方法以后不能改变。只有永久的、本地管理的表空间能够指定自动段空间管理。

注意：对于 LOB，不能指定自动段空间管理。

11.2.1.3 修改本地管理的表空间

不能将一个本地管理的表空间改变为一个本地管理的临时表空间，也不能改变它的段空间管理方法。对于本地管理的表空间，没有必要合并空闲盘区。

对本地管理的表空间使用 ALTER TABLESPACE 语句的一些原因包括：

- 添加一个数据文件。例如：

```
ALTER TABLESPACE lmtbsb
ADD DATAFILE '/u02/oracle/data/lmtbsb02.dbf' SIZE 1M;
```

- 改变一个表空间的可用性（ONLINE/OFFLINE）。
- 使一个表空间为只读或读写。
- 重命名数据文件，或者启用/禁用该表空间中的数据文件的自动盘区大小。

11.2.2 字典管理的表空间

从 Oracle9i 开始，创建表空间时默认的盘区管理为本地管理。但是，可以明确地指定想要创建一个字典管理的表空间。对于字典管理的表空间，当一个盘区被分配，或者被释放以重用，Oracle 更新数据字典中相应的表。

11.2.2.1 创建字典管理的表空间

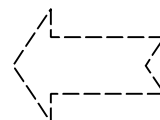
作为一个例子，下面的语句创建有如下的特性的表空间 tbsa：

- 新表空间的数据包含在单个数据文件中，文件大小为 50M。
- 通过指定 EXTENT MANAGEMENT DICTIONARY，该表空间被明确创建成一个字典管理的表空间。
- 该表空间中创建的任何段的默认存储参数都明确地设置。

下面的语句创建表空间 tbsb：

```
CREATE TABLESPACE tbsb
DATAFILE '/u02/oracle/data/tbsa01.dbf' SIZE 50M
EXTENT MANAGEMENT DICTIONARY
DEFAULT STORAGE (
  INITIAL 50K
  NEXT 50K
  MINEXTENTS 2
  MAXEXTENTS 50
  PCTINCREASE 0);
```

注意：如果没有为数据文件完整地指定文件名，Oracle 在默认的数据库目录或当前目录下，创建该数据文件，这取决于操作系统。Oracle 建议通常应该指定一个完全合格的名称。



下一个例子创建表空间 `tbsb`，但这次块大小不同于标准的数据库块大小（由初始化参数 `DB_BLOCK_SIZE` 所指定）。

```
CREATE TABLESPACE tbsb
DATAFILE '/u02/oracle/data/tbsb01.dbf' SIZE 50M
EXTENT MANAGEMENT DICTIONARY
BLOCKSIZE 8K
DEFAULT STORAGE (
INITIAL 50K
NEXT 50K
MINEXTENTS 2
MAXEXTENTS 50
PCTINCREASE 0);
```

注意：为了使 `BLOCKSIZE` 子句成功，必须设置 `DB_CACHE_SIZE` 和至少一个 `DB_nK_CACHE_SIZE` 初始化参数，并且在该子句中指定的整数必须与一个 `DB_nK_CACHE_SIZE` 参数设置一致。尽管是多余的，指定一个 `BLOCKSIZE` 与 `DB_BLOCK_SIZE` 初始化参数指定的标准块大小相等，是允许的。

11.2.2.2 修改字典管理的表空间

使用 `ALTER TABLESPACE` 语句的一个原因是添加数据文件。下面的语句为 `tbsa` 表空间创建一个新的数据文件：

```
ALTER TABLESPACE tbsa
ADD DATAFILE '/u02/oracle/data/tbsa02.dbf' SIZE 1M;
```

发布 `ALTER TABLESPACE` 语句的其他理由包括（但不局限于此）：

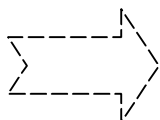
- 改变默认的存储参数。
- 合并表空间中的空闲空间。
- 改变表空间的可用性（`ONLINE/OFFLINE`）。
- 使表空间为只读或读写。
- 添加或重命名数据文件，或者启用/禁用该表空间中的数据文件的自动盘区大小。

11.2.3 临时表空间

为提高多个排序操作的并发能力、减小它们的开销，或避免 Oracle 空间管理操作一起进行，创建临时表空间。一个临时表空间可以被多个用户共享，并且可以在使用 `CREATE USER` 语句创建数据库中的用户时，分配给用户。

在一个临时表空间中，一个给定实例和表空间的所有排序操作共享一个单一的排序段。排序段为一个给定表空间中的每个执行排序操作的实例而存在。排序段由启动之后使用临时表空间用于排序的第一个语句创建，并且只能在关闭时被释放。一个盘区不能被多个事务所共享。

可以使用 `V$SORT_SEGMENT` 视图查看一个临时表空间排序段的空间分配和回收情



况，并且 V\$SORT_USAGE 视图标识出这些段的当前排序用户。

不能在临时表空间中明确地创建对象。

11.2.3.1 创建本地管理的临时表空间

因为在本地管理的表空间中，空间管理更加简单和有效，它们非常适用于临时表空间。本地管理的临时表空间使用临时文件，这种文件不修改临时表空间以外的数据，也不为临时表空间数据生成任何的重作。因而，它们可以用于备用的或只读的数据库。

还要使用与查看数据文件不同的视图来查看关于临时文件的信息。V\$TEMPFILE 和 DBA_TEMP_FILES 视图与 V\$DATAFILE 和 DBA_DATA_FILES 视图类似。

要创建一个本地管理的临时表空间，使用 CREATE TEMPORARY TABLESPACE 语句，这需要用户拥有 CREATE TABLESPACE 系统权限。

下面的语句创建一个临时表空间，其中的每个盘区为 16M。每个 16M 盘区（当标准块大小为 2K 时，它相当于 8000 个块）在该文件的位图中由一位（bit）来表示。

```
CREATE TEMPORARY TABLESPACE lmtmp TEMPFILE '/u02/oracle/data/lmtmp01.dbf'
SIZE 20M REUSE
EXTENT MANAGEMENT LOCAL UNIFORM SIZE 16M;
```

注意：在一些操作系统中，直到临时文件块被实际访问时，Oracle 才为临时文件分配空间。这种空间分配的延迟导致临时文件更快的创建和重新确定大小，但它在以后使用临时文件时，需要足够的可用磁盘空间。请参考所用的操作系统文档，以决定 Oracle 在系统中是否采用这种方法分配临时文件空间。

11.2.3.2 修改本地管理的临时表空间

除了像下面的例子中举例说明的那样添加一个临时文件，不能对一个本地管理的临时表空间使用 ALTER TABLESPACE 语句。

```
ALTER TABLESPACE lmtmp
ADD TEMPFILE '/u02/oracle/data/lmtmp02.dbf' SIZE 2M REUSE;
```

注意：不能使用带有 TEMPORARY 关键字的 ALTER TABLESPACE 语句，把本地管理的永久表空间变成本地管理的临时表空间。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 语句可用于修改临时文件。

如下的语句使临时文件脱机和联机：

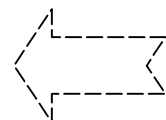
```
ALTER DATABASE TEMPFILE '/u02/oracle/data/lmtmp02.dbf' OFFLINE;
ALTER DATABASE TEMPFILE '/u02/oracle/data/lmtmp02.dbf' ONLINE;
```

如下的语句重新确定一个临时文件的大小：

```
ALTER DATABASE TEMPFILE '/u02/oracle/data/lmtmp02.dbf' RESIZE 4M;
```

如下的语句取消一个临时文件并删除相应的操作系统文件：

```
ALTER DATABASE TEMPFILE '/u02/oracle/data/lmtmp02.dbf' DROP
INCLUDING DATAFILES;
```



该临时文件所属的表空间仍旧保留。一条信息被写入被删除的数据文件的修改文件。如果一个操作系统错误使文件的删除没有成功，该语句仍成功执行，但一条说明该错误的信息被写入修改文件。

也有可能，但没有明确指出，可以 AUTOEXTEND 一个临时文件和重命名 (RENAME FILE) 一个临时文件。

11.2.3.3 创建字典管理的临时表空间

要在表空间创建期间，指定一个表空间为临时的，在 CREATE TABLESPACE 语句中指定 TEMPORARY 关键字。不能为一个以这种方式创建的临时表空间指定 EXTENT MANAGEMENT LOCAL。要创建一个本地管理的临时表空间，使用 CREATE TEMPORARY TABLESPACE 语句，这是创建一个临时表空间的首选方法。

如下的语句创建一个字典管理的临时表空间：

```
CREATE TABLESPACE sort
DATAFILE '/u02/oracle/data/sort01.dbf' SIZE 50M
DEFAULT STORAGE (
  INITIAL 2M
  NEXT 2M
  MINEXTENTS 1
  PCTINCREASE 0)
EXTENT MANAGEMENT DICTIONARY
TEMPORARY;
```

11.2.3.4 修改字典管理的临时表空间

可以发布 ALTER TABLESPACE 语句，禁止一个字典管理的临时表空间使用多个关键字和子句，与此相同的关键字和子句可用于一个字典管理的永久表空间。

注意：当使用 ALTER TABLESPACE ... OFFLINE 语句使字典管理的临时表空间脱机时，使它们重新联机不影响它们的临时状态。

使用 ALTER TABLESPACE 语句，可以将一个现有的字典管理的永久表空间改变为一个临时表空间。例如：

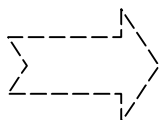
```
ALTER TABLESPACE tbsa TEMPORARY;
```

11.3 管理表空间分配

当创建一个表空间时，用户确定什么物理数据文件组成该表空间，以及对于字典管理的表空间，该表空间的默认存储特性是什么。该表空间的所有这些属性都可以在随后修改。一个表空间的默认存储特性在本节中讨论。

随着时间的推移，一个字典管理的表空间的空闲空间可能成为碎片，使得分配新的盘区变得困难。整理这种空闲空间碎片的方法也在本节讨论。

本节包括以下内容：



- 本地管理的表空间中的存储参数
- 字典管理的表空间的存储参数
- 合并字典管理的表空间的空闲空间

11.3.1 本地管理的表空间中的存储参数

用户不能为本地管理的表空间指定默认的存储参数，也不能指定 `MINIMUM_EXTENT`。如果指定了 `AUTOALLOCATE`，该表空间是使用最小盘区大小为 64K 系统管理的。如果指定了 `UNIFORM SIZE`，那么该表空间是使用指定 `SIZE` 的统一大小盘区管理的。默认的 `SIZE` 为 1M。

当在本地管理的表空间中分配段（创建对象）时，在创建时指定的存储子句与用于字典管理的表空间的子句有不一样的解释。当在本地管理的表空间中创建一个对象时，Oracle 使用它的 `INITIAL`、`NEXT` 和 `MINEXTENTS` 参数来计算该对象段的初始大小。

11.3.2 字典管理的表空间的存储参数

存储参数同时影响它访问存储在数据库中的数据所花的时间和数据库中空间的使用效率。

11.3.2.1 指定默认的存储参数

下面的参数影响表空间中段存储分配。它们被称作存储参数，并包含在 `CREATE TABLESPACE` 语句的 `storage_clause` 中。

<code>INITIAL</code>	定义该段中第一个盘区以字节计（K 或 M）的大小
<code>NEXT</code>	定义第二个盘区以字节计的大小（K 或 M）
<code>PCTINCREASE</code>	指定在第二个（ <code>NEXT</code> ）盘区以后的每个盘区增长的百分比
<code>MINEXTENTS</code>	指定在表空间中第一次创建一个段时所分配的盘区数
<code>MAXEXTENTS</code>	确定一个段可以拥有的最大盘区数。也可指定为 <code>UNLIMITED</code> 。

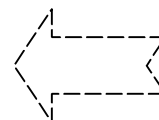
`CREATE TABLESPACE` 语句的另一个参数 `MINIMUM EXTENT` 也影响段的分配。如果指定了该参数，它确保该表空间中所有空闲和已分配的盘区至少与指定的字节数（K 或 M）的一个倍数一样大。这提供了一种在该表空间中控制空闲空间碎片的方法。

11.3.2.2 修改表空间的存储设置

可以修改一个表空间的默认存储参数，以改变该表空间中创建将来的对象的默认要求。要改变在该表空间中随后创建的对象默认存储参数，使用 `ALTER TABLESPACE` 语句。

```
ALTER TABLESPACE users
DEFAULT STORAGE (
NEXT 100K
MAXEXTENTS 20
PCTINCREASE 0);
```

不能在一条 `ALTER` 语句中指定 `INITIAL` 和 `MINEXTENTS` 关键字。一个表空间的默认存储参数新值只影响将来该表空间中为段分配的盘区。



11.3.3 合并字典管理的表空间的空闲空间

表空间中的一个空闲盘区由相邻的空闲块集合构成。当为一个表空间段分配新的盘区时，与需要的盘区在大小上最接近的空闲盘区被使用。在有些情况中，当段被取消时，它们的盘区被回收并被标记为空闲，但任何相邻的空闲盘区并不能立即被重组为更大的空闲盘区。成为碎片的结果使得分配更大的盘区更加困难。

这种碎片可以用几种方法来解决：

- 当试图为一个段分配一个新的盘区时，Oracle 首先试图找到一个比新盘区更大的空闲盘区。如果没有找到足够大的空闲盘区，那么 Oracle 合并该表空间中相邻的空闲盘区并再次查找。这种合并通常由 Oracle 在它找不到一个适合新盘区的空闲盘区时执行。
- 当一个表空间的 PCTINCREASE 值不为 0 时，SMON 后台进程定期合并相邻的空闲盘区。如果设置 PCTINCREASE=0，那么就不会有空闲盘区合并的发生。如果担心 SMON 进行合并的开销，那么一种变通的方法就是设置 PCTINCREASE=0，并且定期地手动合并空闲空间。
- 当一个段被取消或截断时，如果该段的 PCTINCREASE 值不为 0，那么将执行一种限制形式的合并。即使包含该段的表空间的 PCTINCREASE=0，这也可完成。
- 可以使用 ALTER TABLESPACE ... COALESCE 语句来手动合并任何相邻的空闲盘区。

合并空闲空间的进程在下图中说明。

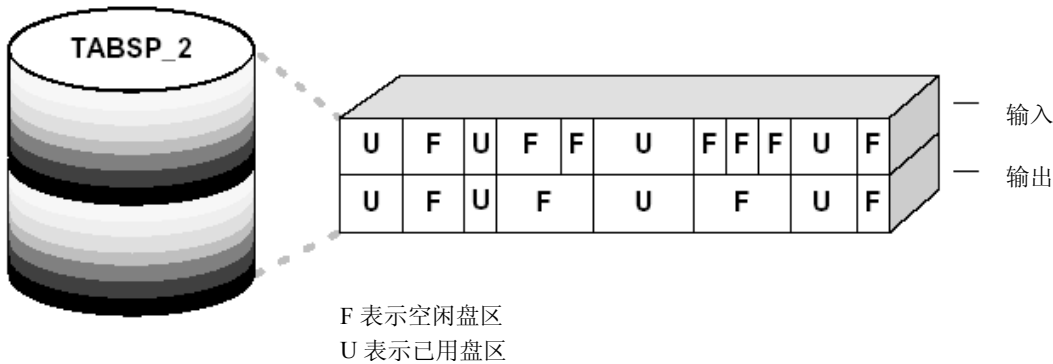


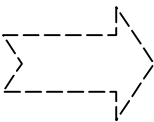
图 11-1 合并空闲空间

注意：对于本地管理的表空间，合并空闲空间不是必需的，因为位图自动跟踪相邻的空闲空间。

11.3.3.1 手动合并空闲空间

如果发现一个表空间中碎片空间很多（磁盘上相邻的空间看上去不相邻的），可以使用 ALTER TABLESPACE ... COALESCE 语句手动合并任何空闲空间。必须拥有 ALTER TABLESPACE 系统权限来合并表空间。

如果 PCTINCREASE=0，可能想要使用该语句，或者可以使用它来补充 SMON 和盘区



分配合并。如果该表空间中所有的盘区具有相同的大小，合并是没有必要的。可能是这种情况：如果该表空间的默认 PCTINCREASE 值已设置为 0，所有的段使用该表空间的默认存储参数，并且 INITIAL = NEXT = MINIMUM EXTENT。

如下的语句合并表空间 `tabsp_4` 的空闲空间。

```
ALTER TABLESPACE tabsp_4 COALESCE;
```

像 ALTER TABLESPACE 语句的其他选项一样，COALESCE 选项是排它的：当它被指定，它必须是唯一的选项。

该语句不能合并被数据盘区所分开的空闲盘区。如果观察到有很多空闲盘区位于数据盘区之间，必须重新组织该表空间（例如，通过导出和导入它的数据）来创建有用的空闲空间盘区。

11.3.3.2 监控空闲空间

可以使用下列视图来监控一个表空间中的空闲空间：

- DBA_FREE_SPACE
- DBA_FREE_SPACE_COALESCED

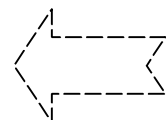
如下的语句显示表空间 `tabsp_4` 中的空闲空间：

```
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.
```

该视图显示了 `tabsp_4` 中有还未合并的相邻空闲空间（例如，从 BLOCK_ID 2、4、6、16 开始的块）。在使用前面显示的 ALTER TABLESPACE 语句合并该表空间后，这个查询



的结果将读取：

```
BLOCK_ID BYTES BLOCKS
-----
2 131072 16
27 311296 38
2 rows selected.
```

DBA_FREE_SPACE_COALESCED 视图显示合并动作的统计情况。它对确定是否需要合并空间也是有用的。

11.4 修改表空间可用性

可以使一个联机的表空间脱机，以便数据库的这部分对于一般的使用暂时不可用。数据库的其余部分是打开的，并可用于用户访问数据。相反地，可以将一个脱机的表空间联机使得该表空间中的模式对象可被数据库用户使用。该数据库必须是打开的。

要修改一个表空间的可用性，使用 SQL 语句 ALTER TABLESPACE。用户必须拥有 ALTER TABLESPACE 或 MANAGE TABLESPACE 系统权限来执行这个动作。

也可以使一个表空间的所有数据文件或临时文件脱机，并使它们重新联机，而不影响该表空间本身的 OFFLINE 或 ONLINE 状态。

11.4.1 使表空间脱机

可能基于下列理由，想要使一个表空间脱机：

- 使数据库的一部分不可用，而同时允许正常访问该数据库的剩余部分
- 执行一次脱机的表空间备份（尽管一个表空间可以在联机和使用中备份）
- 当更新和维护一个应用时，使得该应用和它的表组不可用

当一个表空间被脱机时，Oracle 使所有关联的文件脱机。SYSTEM 表空间永远不能被脱机。

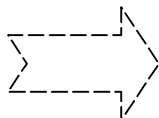
当使一个表空间脱机时，可以指定下列任何选项：

NORMAL 如果该表空间的任何数据文件不存在错误状态，一个表空间可以被正常地脱机。作为一个写错误的结果，该表空间中没有数据文件可以被当前脱机。当 OFFLINE NORMAL 被指定时，Oracle 在使该表空间的所有数据文件脱机时，为它们设置一个检测点。NORMAL 是默认的。

TEMPORARY 一个表空间可以被暂时脱机，即使该表空间的一个或更多文件存在错误状态。当 OFFLINE TEMPORARY 被指定时，Oracle 使还没有脱机的数据文件脱机，同时为它们设置检测点。

如果没有文件被脱机，但使用了暂时选项，使这些表空间重新联机不需要介质恢复。但是，如果该表空间的一个或更多文件因为写错误而脱机，并且使这些表空间暂时脱机，在能够使它重新联机之前，该表空间需要恢复。

IMMEDIATE 一个表空间可以被立即脱机，不用 Oracle 在任何数据文件上设置检测点。当指定 OFFLINE IMMEDIATE 时，在能够被重新联机之前，该表空间需要介质恢复。



如果数据库正在 NOARCHIVELOG 模式中运行，不能使一个表空间立即脱机。

FOR RECOVER 使在恢复中的数据库表空间为表空间 point-in-time 恢复设置为脱机。

警告：如果必须使一个表空间脱机，如果可能，使用 NORMAL 选项（默认的）。这保证了该表空间重新联机不需要恢复。即使在未完成的恢复后，使用一条 ALTER DATABASE OPEN RESETLOGS 语句重置了重作日志序列，它也不需要恢复。

只有在不能使表空间正常脱机的时候，才指定 TEMPORARY。在这种情况下，只有因为错误而被脱机的文件，在该表空间能够重新被联机之前需要恢复。只有在尝试了正常和暂时选项之后，才可以指定 IMMEDIATE。

下面的例子使 users 表空间正常地脱机：

```
ALTER TABLESPACE users OFFLINE NORMAL;
```

在使一个联机的表空间脱机之前，考虑采取以下的动作：

- 验证该表空间没有包含活动的回滚段。这样的表空间不能被脱机。
- 可能想要修改表空间的分配，该表空间已经被用户指定为默认的或临时的表空间。这正是建议采取的动作，因为他们在表空间脱机时，将不能访问其中的对象或排序区域。

11.4.2 使表空间联机

当数据库打开时，可以使 Oracle 数据库的任何表空间联机。一个表空间被正常地联机，以便其中包含的数据可以被数据库用户使用。

注意：如果一个将要联机的表空间原来没有被“干净地”脱机（即使用 ALTER TABLESPACE OFFLINE 语句的 NORMAL 选项），必须在使它联机之前，首先执行介质恢复表空间。否则，Oracle 返回一个错误，并且该表空间仍旧保持脱机。

如下的语句使 users 表空间联机：

```
ALTER TABLESPACE users ONLINE;
```

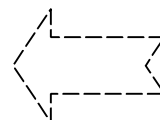
11.4.3 修改数据文件或临时文件的可用性

ALTER TABLESPACE 语句的子句使用户可以改变一个表空间中所有数据文件或临时文件的联机或脱机状态。特别地，影响联机/脱机状态的语句有：

- ALTER TABLESPACE ... DATAFILE { ONLINE | OFFLINE }
- ALTER TABLESPACE ... TEMPFILE { ONLINE | OFFLINE }

用户只需要输入该表空间名称，而不是单独的数据文件或临时文件。所有的数据文件或临时文件都受影响，但表空间本身的联机/脱机状态没有被改变。

在大多数情况下，上述的 ALTER TABLESPACE 语句可以在数据库被装载时发布，即使它没有被打开。如果该表空间是 SYSTEM 表空间、一个撤消表空间或默认的临时表空间，数据库必须是没有打开的。ALTER DATABASE DATAFILE 和 ALTER DATABASE TEMPFILE 语句也有 ONLINE/OFFLINE 子句，然而在这些语句中需要输入该表空间所有的文件名。



这种语法与修改表空间可用性的 `ALTER TABLESPACE ... ONLINE | OFFLINE` 语句不同，因为那是个不一样的操作。`ALTER TABLESPACE` 语句使数据文件以及表空间脱机，但它不能用于修改临时表空间或其临时文件的状态。

11.5 使用只读表空间

使一个表空间只读，就可以防止对该表空间中的数据文件进行写操作。只读表空间的主要目的是消除执行数据库大量的静态部分的备份和恢复的需要，但它们也提供一种完全保护历史数据的方法，使得没有人可以在事实之后修改数据。使一个表空间只读防止对该表空间中所有表的更新，不管一个用户的更新权限级别。

注意：使一个表空间只读不能以它自身用于满足存档或数据公布的要求，因为该表空间只能在创建它的数据库中被联机。但是，可以通过使用可传输的表空间特性来满足这类要求。

可以从只读表空间中取消项，比如表或索引，但不能在该表空间中创建或修改对象。可以执行更新数据字典中文件描述的语句，比如 `ALTER TABLE ... ADD` 或 `ALTER TABLE ... MODIFY`，但直到该表空间成为可读写的，才能使用新的描述。

只读表空间可以被传输到其他数据库。并且，由于只读表空间永远不能被更新，它们可以驻留在 CD-ROM 或 WORM（写一次-读多次）设备上。

在本节中讨论下列的内容：

- 使表空间只读
- 使只读表空间可写
- 在 WORM 设备上创建只读表空间
- 延迟只读表空间中数据文件的打开

11.5.1 使表空间只读

所有表空间初始创建时都是可读写的。使用 `ALTER TABLESPACE` 语句中的 `READ ONLY` 子句将一个表空间改变为只读的。必须拥有 `ALTER TABLESPACE` 或 `MANAGE TABLESPACE` 系统权限。

在能够使一个表空间只读之前，必须满足下列条件：

- 该表空间必须是联机的。

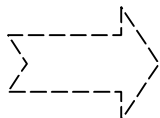
这是必要的，以确保没有需要用于该表空间的撤消信息。

- 该表空间必须没有包含任何活动的回滚段（这可能是正常情况，因为一个数据表空间应该不包含回滚段）。

基于这个原因，`SYSTEM` 表空间永远不能成为只读的，因为它包含 `SYSTEM` 回滚段。另外，因为一个只读表空间的任何回滚段都不能被访问，必须在使一个表空间只读前，取消回滚段。

- 该表空间当前必须没有涉及联机备份，因为在备份的最后，要更新该表空间中所有数据文件的头文件。

在访问只读表空间中的数据时，要获得更好的性能，可以恰好在使它只读之前，发布



一个访问该表空间中表的所有块的查询。一个简单的查询，比如 `SELECT COUNT (*)`，对每个表执行确保表空间中的数据块可以在随后被最有效地访问。这样消除了 Oracle 检查最近修改块的事务的状态的必要。

如下的语句使 `flights` 表空间只读：

```
ALTER TABLESPACE flights READ ONLY;
```

在发布 `ALTER TABLESPACE ... READ ONLY` 语句之前，不必等待事务完成。当语句发布时，目标表空间进入一种过渡的只读模式，在这种模式中不允许对该表空间的进一步写操作（DML 语句）。现有的修改该表空间的事务被允许提交或回滚。一旦（该数据库中的）所有的事务已经完成，该表空间成为只读的。

注意：这种过渡的只读状态只有在初始化参数 `COMPATIBLE` 的值为 8.1.0 或更高时才会出现。如果该参数被设置为一个小于 8.1.0 的值，`ALTER TABLESPACE ... READ ONLY` 语句在有任何活动事务存在的情况下失败。

如果发现该表空间停顿了很长时间，能识别阻止只读状态有效的事务。如果必要，可以通知这些事务的属主，并作出终止事务的决定。下面的例子说明了如何识别阻塞的事务：

- 识别 `ALTER TABLESPACE ... READ ONLY` 语句的事务入口，并注明它的会话地址（`saddr`）。

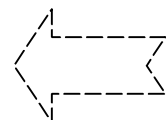
```
SELECT SQL_TEXT, SADDR
FROM V$SQLAREA, V$SESSION
WHERE V$SQLAREA.ADDRESS = V$SESSION.SQL_ADDRESS
AND SQL_TEXT LIKE 'alter tablespace%';
SQL_TEXT SADDR
```

```
-----
alter tablespace tbs1 read only 80034AF0
```

- 每个活动事务的启动 SCN 存储在 `V$TRANSACTION` 视图中。显示按启动 SCN 升序排列的这种视图，以执行顺序列出事务。由于知道只读语句事务入口的会话地址，它可以位于 `V$TRANSACTION` 视图中。所有具有较少启动 SCN 的事务可能阻塞该表空间的静态和随后的只读状态。

```
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
80034AF0 3628 --> this is the ALTER TABLESPACE statement
80037910 3629 --> don't care about this txn
```

使该表空间只读以后，建议立即将它备份。只要该表空间保持在只读状态，对其进一



步备份就没有必要，因为不会对它作出改变。

11.5.2 使只读表空间可写

使用 ALTER TABLESPACE 语句中的 READ WRITE 关键字，将一个表空间改变为允许写操作。必须拥有 ALTER TABLESPACE 或 MANAGE TABLESPACE 系统权限。

使该表空间成为可读写的一个先决条件是该表空间中所有的数据文件，以及表空间自身，必须是联机的。使用 ALTER DATABASE 语句的 DATAFILE ... INLINE 子句使一个数据文件联机。V\$DATAFILE 视图列出数据文件的当前状态。

如下的语句使 flights 表空间成为可写的：

```
ALTER TABLESPACE flights READ WRITE;
```

使一个只读表空间可写更新了该数据文件的控制文件入口，以便可以使用该数据文件的只读版本作为恢复的开端。

11.5.3 在 WORM 设备上创建只读表空间

按照这些步骤在 CD-ROM 或 WORM（写一次-读多次）设备创建一个只读表空间。

①在其他设备上创建一个可写的表空间。创建属于该表空间的对象，并且插入用户的数据。

②修改该表空间以使它只读。

③将该表空间的数据文件复制到 WORM 设备上。使用操作系统命令来复制这些文件。

④使该表空间脱机。

⑤重命名数据文件，使其与复制到 WORM 设备的数据文件名称一致。使用带有 RENAME DATAFILE 子句的 ALTER TABLESPACE 语句。重命名这些数据文件，改变它们在控制文件中的名称。

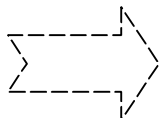
⑥使该表空间重新联机。

11.5.4 延迟只读表空间中数据文件的打开

当一个非常大的数据库的实质性部分存储在位于低速访问设备的只读表空间中或分级存储时，应该考虑将初始化参数 READ_ONLY_OPEN_DELAYED 设置为 TRUE。这将加速某些操作，主要是打开数据库，其原因是在需要读取存储在它们中的数据时，使得只读表空间的数据文件只在第一次被访问。

设置 READ_ONLY_OPEN_DELAYED = TRUE 带来下列负面影响：

- 在打开时检测不到缺少的或坏的只读文件。只有在试图访问它时，才会发现。
- ALTER DATABASE CHECK DATAFILES 不检查只读文件。
- ALTER TABLESPACE ... ONLINE 和 ALTER DATABASE DATAFILE ... ONLINE 不检查只读文件。它们只在第一次访问上被检查。
- V\$RECOVER_FILE、V\$BACKUP 和 V\$DATAFILE_HEADER 不访问只读文件。只读文件在有错误“DELAYED OPEN”的结果列表中显示，其他列的值为 0。
- V\$DATAFILE 不访问只读文件。只读文件以“0”大小列出。
- V\$RECOVER_LOG 不访问只读文件。它们可能需要的用于恢复的日志没有被添加



到该列表中。

- ALTER DATABASE NOARCHIVELOG 不访问只读文件。即使有一个需要恢复的只读文件，它也继续进行。

注意：

- RECOVER DATABASE 和 ALTER DATABASE OPEN RESETLOGS 继续访问所有的只读数据文件，而不管该参数值为何。如果想要避免为这些操作访问只读文件，这些文件应该被脱机。
- 如果使用了一个备份控制文件，那么一些文件的只读状态可能是不准确的。这可能导致一些这样的操作返回意想不到的结果。在这种情况下，应该小心。

11.6 取消表空间

如果不再需要一个表空间及其内容（该表空间所包含的段），可以从数据库中取消该表空间及其内容。一个 Oracle 数据库中的任何表空间，除 SYSTEM 表空间外，都可以被取消。必须拥有 DROP TABLESPACE 系统权限来取消一个表空间。

警告：一旦一个表空间已经被取消，该表空间的数据就不能恢复了。因而，要确保一个将被取消的表空间包含的所有数据将来都不再需要。另外，在从数据库撤消表空间前后，立即对数据库作一次完全备份。强烈建议采取这种方法，以便如果错误地取消了一个表空间，或者如果数据库在撤消表空间之后遇到问题，能够恢复该数据库。

当取消一个表空间时，相关数据库的控制文件中的文件指针被删除。可以随意地指示 Oracle 删除构成被取消的表空间的操作系统文件（数据文件）。如果在 Oracle 删除表空间的同时，没有指示它删除数据文件，就必须在以后使用合适的操作系统命令来删除它们。

不能取消包含任何活动段的表空间。例如，如果表空间中一个表当前正在被使用，或者表空间包含一个活动的回滚段，就不能取消该表空间。为了简单，使表空间在被取消之前脱机。

要取消一个表空间，使用 DROP TABLESPACE 语句。如下的语句取消 users 表空间，包括该表空间中的段：

```
DROP TABLESPACE users INCLUDING CONTENTS;
```

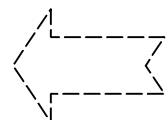
如果该表空间是空的（没有包含任何表、视图或其他结构），就不必指定 INCLUDING CONTENTS 选项。使用 CASCADE CONSTRAINTS 选项从表空间以外的表中取消所有指示完整性的约束条件，这些表空间引用了要撤消表空间中表的主键和唯一键。

要在取消一个表空间的同时删除与它相连的数据文件，使用 INCLUDING CONTENTS AND DATAFILES 子句。

如下的语句取消 users 表空间以及与它相连的数据文件：

```
DROP TABLESPACE users INCLUDING CONTENTS AND DATAFILES;
```

一条信息写入每个被删除的数据文件的修改文件中。如果一个操作系统错误使得文件删除没有成功，DROP TABLESPACE 语句仍然成功完成，但一条说明该错误的信息写入修改文件。



11.7 使用 DBMS_SPACE_ADMIN 解决表空间问题

注意: DBMS_SPACE_ADMIN 包为本地管理的表空间提供带有故障诊断和修复功能的管理程序。它不能用于字典管理的表空间。

DBMS_SPACE_ADMIN 包含有下列过程:

过程	说明
SEGMENT_VERIFY	验证该段盘区映射的一致性。
SEGMENT_CORRUPT	标注该段为损坏或有效, 以便执行恰当的错误恢复。
SEGMENT_DROP_CORRUPT	取消一个当前标注为损坏的段 (不回收空间)。
SEGMENT_DUMP	卸下一个给定段的段头部和盘区映射。
TABLESPACE_VERIFY	验证该表空间中段的位图和盘区映射是否同步。
TABLESPACE_REBUILD_BITMAPS	重建适当的位图。
TABLESPACE_FIX_BITMAPS	在位图中标注适当的数据块地址范围 (盘区) 为空闲或已用。
TABLESPACE_REBUILD_QUOTAS	为给定的表空间重建限额。
TABLESPACE_MIGRATE_FROM_LOCAL	将一个本地管理的表空间移植为字典管理的表空间。
TABLESPACE_MIGRATE_TO_LOCAL	将一个表空间从字典管理的格式移植为本地管理的格式。
TABLESPACE_RELOCATE_BITMAPS	将位图重定位到指定的目的地。
TABLESPACE_FIX_SEGMENT_STATES	修改移植被放弃的表空间中数据段的状态。

下列案例说明了可以利用 DBMS_SPACE_ADMIN 包来诊断和解决故障的典型情况。

注意: 其中的某些过程如果没有正确使用, 可能导致数据的丢失和不可恢复。如果对这些过程有疑问, 应该使用 Oracle Support Services。

11.7.1 案例 1: 当分配的块标注为空闲 (没有重叠) 时修改位图

TABLESPACE_VERIFY 过程发现一个段拥有在位图中标注为空闲的已分配的块, 但没有段之间重叠的报告。

在这种情形下, 执行下列任务:

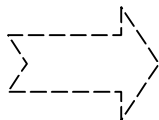
- ①调用 SEGMENT_DUMP 过程来卸下管理程序分配给该段的区域。
- ②对每个区域, 调用带有 TABLESPACE_EXTENT_MAKE_USED 选项的 TABLESPACE_FIX_BITMAPS 过程来将该空间标注为已用的。
- ③调用 TABLESPACE_REBUILD_QUOTAS 过程来确定限额。

11.7.2 案例 2: 取消一个损坏的段

不能取消一个段, 因为位图将段块标注为“空闲”。系统自动将该段标注为损坏的。

在这种情形下, 执行下列任务:

- ①调用带有 SEGMENT_VERIFY_EXTENTS_GLOBAL 选项的 SEGMENT_VERIFY 过程。如果没有报告有重叠, 那么进行步骤 2 至 5。
- ②调用 SEGMENT_DUMP 过程来卸下分配给该段的 DBA 区域。



③ 对每个区域，调用带有 TABLESPACE_EXTENT_MAKE_FREE 选项的 TABLESPACE_FIX_BITMAPS 过程来将该空间标注为空闲的。

④ 调用 SEGMENT_DROP_CORRUPT 过程来取消 SEG\$ 入口。

⑤ 调用 TABLESPACE_REBUILD_QUOTAS 过程来修改限额。

11.7.3 案例 3：在报告重叠处修改位图

TABLESPACE_VERIFY 过程报告一些重叠现象。一些实际数据因为原先的内部错误必须被牺牲。

在选好要牺牲的对象之后，在这个例子中就是表 t1，执行下列任务：

① 生成一个 t1 重叠的所有对象的列表。

② 取消表 t1。如果必要，通过调用 SEGMENT_DROP_CORRUPT 过程继续进行。

③ 对 t1 重叠的所有对象调用 SEGMENT_VERIFY 过程。如果必要，调用 TABLESPACE_FIX_BITMAPS 过程来将合适的位图块标注为已用的。

④ 返回到 TABLESPACE_VERIFY 过程来验证故障是否解决。

11.7.4 案例 4：纠正位图块的介质损坏

一组位图块有介质损坏。

在这种情形下，执行下列任务：

① 对所有位图块，或者对单个块，如果只有一个块损坏的话，调用 TABLESPACE_REBUILD_BITMAPS 过程。

② 调用 TABLESPACE_REBUILD_QUOTAS 过程来重建限额。

③ 调用 TABLESPACE_VERIFY 过程来验证该位图是否一致。

11.7.5 案例 5：从字典管理的移植为本地管理的表空间

将一个字典管理的表空间移植为本地管理的表空间。使用 TABLESPACE_MIGRATE_TO_LOCAL 过程。

假定该数据库块大小为 2K，并且表空间 tbs_1 的现有盘区大小为 10, 50 和 10,000 个块（已用、已用和空闲）。MINIMUM EXTENT 的值为 20K（10 个块）。在这种情形下，用户允许该位图分配单元由系统选择。10 个块的值被选中，因为它是最大的，并且没有超过 MINIMUM EXTENT 的公分母。

将 tbs_1 转变为一个本地管理的表空间的语句如下：

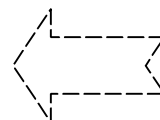
```
EXEC DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_TO_LOCAL ('tbs_1');
```

如果选择指定一个分配单元的大小，它必须是由系统计算的单元大小的一个因数，否则，就会出现错误信息。

11.8 在数据库之间传输表空间

本节介绍如何在数据库之间传输表空间，包括以下内容：

- 可传输表空间的介绍



- 约束条件
- 可传输表空间的兼容性考虑
- 在数据库之间传输表空间：一个过程
- 对象行为
- 使用可传输表空间

11.8.1 可传输表空间的介绍

注意：必须使用 Oracle8i（或更高版本）的企业版来生成一个可传输表空间集合。然而，可以使用 Oracle8i（或更高版本）的任何版本将一个可传输表空间集合插入到一个 Oracle 数据库中。

可以使用可传输表空间特性来移动 Oracle 数据库的一个子集并将它“插入”到另一个 Oracle 数据库中，实质上是在数据库之间移动表空间。要传输的表空间可以是字典管理的或本地管理的。从 Oracle9i 开始，传输的表空间不要求与目标数据库的标准块大小具有相同的块大小。传输表空间在以下方面特别有用：

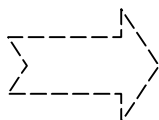
- 将数据从 OLTP 系统移动到数据仓库分级系统
- 从分级系统更新数据仓库和数据中心
- 从中心数据仓库装载数据中心
- 启用存档 OLTP 和数据仓库系统
- 对内部和外部使用者发布数据
- 执行表空间 Point-in-Time 恢复（TSPITR）

使用可传输表空间移动数据比执行相同数据的导出/导入或卸载/装载操作快得多，因为传输一个表空间只需要复制数据文件和集成该表空间结构信息。也可以使用可传输表空间来移动索引数据，从而避免当导入或装载表数据时必须执行索引重建。

11.8.2 约束条件

在计划可传输表空间的使用时，清楚以下约束条件：

- 源和目标数据库必须在相同的硬件平台上。例如，可以在 Sun Solaris Oracle 数据库之间传输表空间，或者可以在 Windows NT Oracle 数据库之间传输表空间。然而，不可以将一个表空间从一个 Sun Solaris Oracle 数据库传输到一个 Windows NT Oracle 数据库。
- 源和目标数据库必须使用相同的字符集和本国字符集。
- 不能将一个表空间传输到一个已有同名表空间存在的目标数据库。
- 可传输表空间不支持：
 - 物化视图/复制
 - 基于函数的索引
 - 规定范围的 REF
 - 8.0 兼容的带多个接受者的高级查询



11.8.3 可传输表空间的兼容性考虑

要使用可传输表空间特性，源和目标数据库的初始化参数 `COMPATIBLE` 必须同时设置为 8.1 或更高。如果任何要传输的表空间的块大小与目标数据库的标准块大小不同，必须将目标数据库的初始化参数 `COMPATIBLE` 必须设置为 9.0 或更高。没有要求同时为源和目标数据库运行 Oracle 的相同版本。Oracle 保证可传输数据库集合与目标数据库兼容。如果不是，在插入操作开始时，发信号通知错误。

通常可能将一个表空间从一个运行较早版本 Oracle (Oracle8i 起) 的数据库传输到一个运行较新版本 Oracle (例如, Oracle9i) 的数据库。

当创建一个可传输表空间集合时，Oracle 计算目标数据库必须运行的最低兼容级。这被看作可传输集合的兼容级。当插入可传输集合到一个目标数据库中时，如果该可传输集合的兼容级大于目标数据库的兼容级，Oracle 发信号通知错误。

11.8.4 在数据库之间传输表空间：一个过程

要移动或复制一个表空间集合，执行下列步骤。这些步骤在随后的章节中更充分地说明，这些章节详细描述了在数据库之间传输表空间 `sales_1` 和 `sales_2`。

①选择一个自含式（自包含）表空间集合。

②生成一个可传输表空间集合。

一个可传输表空间集合由要传输的表空间集合的数据文件和一个包含该表空间集合结构信息的文件组成。

③传输该表空间集合。

将数据文件和输出文件复制到目标数据库。可以使用任何复制平面文件的工具软件(例如，一个操作系统复制工具、ftp 或随 CD 发布的)来完成该操作。

④插入该表空间。

调用 `Import` 工具来将该表空间集合插入到目标数据库。

11.8.4.1 步骤 1: 选择一个自含式表空间集合

在可传输集合内的对象和集合外的对象之间，可能有逻辑的或物理的相关性。可以只传输一个自含式的表空间集合。在这里的上下文中，“自含式”表示没有从表空间集合内指向表空间外的引用。一些违反自含式表空间的例子是：

- 一个该表空间集合内部的索引用于一个该表空间集合以外的表。

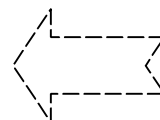
注意：如果表的对应索引在该表空间集合以外，就不是违例。

- 一个分区表部分包含在该表空间集合中。

想要复制的表空间集合必须要么包含一个分区表的所有分区，要么对分区表的一个分区也不包含。如果想要传输分区表的一个子集，必须将这些分区交换到表中。

- 指向一个表的指示完整性的约束条件越过了设置的界限。

当传输一个表空间集合时，可以选择包含指示完整性的约束条件。然而，这样做会影响一个表空间集合是否为自含式的。如果决定不传输约束条件，那么这些约束条件就不被考虑。



- 一个表空间集合内部的表包含一个指向表空间集合以外的 LOB 的 LOB 列。

要确定一个表空间集合是否为自含式的，可以调用 Oracle 提供的 DBMS_TTS 包中的 TRANSPORT_SET_CHECK 过程。用户必须被赋予 EXECUTE_CATALOG_ROLE 角色（初始被标记为 SYS）来执行该过程。

当调用 DBMS_TTS 包时，指定要被检查是否为自身密闭的可传输集合中表空间的列表。如果约束条件必须被包含，可以随意地指定。对于严格或完全密闭，必须额外设置 TTS_FULL_CHECK 参数为 TRUE。

严格或完全密闭检查针对那些不仅需要捕获指向可传输集合以外的引用，还要捕获指向集合内部的引用的情况。Tablespace Point-in-Time Recovery (TSPITR) 就是一个这样的情况，这里相关对象必须完全被可传输集合包含，或者完全位于集合之外。

例如，在一个包含表 t 但不包含其索引 i 的表空间上执行 TSPITR 就是一个违例，因为传输之后，索引和数据会不一致。一次完全密闭检查确保没有出去和进入该可传输集合的相关性。

注意：可传输表空间默认为检查自身密闭，而不是检查完全密闭。

这里确定表空间 sales_1 和 sales_2 是否为自含式的，同时考虑指示完整性的约束条件（由 TRUE 指出）。

```
EXECUTE dbms_tts.transport_set_check('sales_1,sales_2', TRUE);
```

调用该 PL/SQL 包之后，可以通过从 TRANSPORT_SET_VIOLATIONS 视图中选择看到所有的违例。如果该表空间集合是自含式的，这个视图为空。如下的查询显示有两个违例的情况：一个外部关键字约束条件 dept_fk 越过了该表空间设置的界限，以及一个分区表 jim.sales，它部分包含在该表空间集合中。

```
SELECT * FROM TRANSPORT_SET_VIOLATIONS;
```

```
-----
Constraint DEPT_FK between table JIM.EMP in tablespace SALES_1 and table
JIM.DEPT in tablespace OTHER
```

```
Partitioned table JIM.SALES is partially contained in the transportable set
```

这些违例必须在 sales_1 和 sales_2 成为可传输的之前被解决。正如在下一步中提到的那样，绕过完整性约束条件违例的一个选择就是不输出完整性约束条件。

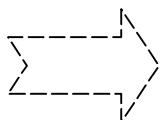
对象引用（比如 REF）越过表空间集合不被当作违例。REF 不由 TRANSPORT_SET_CHECK 例程检查。当一个包含虚悬 REF 的表空间被插入数据库时，紧随该虚悬 REF 的查询指出用户错误。

11.8.4.2 步骤 2: 生成一个可传输表空间集合

在确保有一个想要传输的自含式表空间集合之后，通过执行下列任务来生成一个可传输表空间集合：

- ①使正在复制的集合中的所有表空间为只读。

```
ALTER TABLESPACE sales_1 READ ONLY;
```




```
ALTER TABLESPACE sales_2 READ ONLY;
```

②调用 Export 工具，并指定哪个表空间在可传输集合中，如下：

```
EXP TRANSPORT_TABLESPACE=y TABLESPACES=(sales_1,sales_2)
```

```
TRIGGERS=y CONSTRAINTS=n GRANTS=n FILE=expdat.dmp
```

注意：尽管使用了 Export 工具，仍只有该表空间的数据字典结构信息（元数据）被输出。因此，即使对于一个较大的表空间，该操作执行得很快。

当被提示时，用具有 SYSDBA 系统权限的 SYS（或者其他管理用户）进行连接：

```
CONNECT SYS/password AS SYSDBA.
```

通常必须指定 TABLESPACES。在本例中，也指定它：

- 触发器被输出。

如果设置 TRIGGERS = y，触发器不用有效性检查就被输出。无效的触发器在随后的输入期间，引起编译错误。如果设置 TRIGGERS = n，触发器不被输出。

- 指示完整性的约束条件不被输出。
- 授权不被输出。
- 要创建的结构信息输出文件的名称是 expdat.dmp。

如果在执行 TSPITR 或带有严格密闭检查的传输，使用：

```
EXP TRANSPORT_TABLESPACE=y TABLESPACES=(sales_1,sales_2)
```

```
TTS_FULL_CHECK=Y FILE=expdat.dmp
```

如果在传输的表空间集合不是自含式的，输出失败并指出该可传输集合不是自含式的。必须返回到步骤 1 来解决所有的违例。

11.8.4.3 步骤 3：传输该表空间集合

传输该表空间的数据文件和输出文件两者到一个目标数据库可以访问的位置。可以使用任何复制平面文件的工具软件（例如，一个操作系统复制工具、ftp 或随 CD 发布的）。

11.8.4.4 步骤 4：插入该表空间集合

注意：如果传输一个表空间，它的块大小与接受该表空间集合的数据库标准块大小不同，那么必须首先在接受数据库的参数文件中有一个 DB_nK_CACHE_SIZE 初始化参数入口。

例如，如果传输一个块大小为 8K 的表空间到一个标准块大小为 4K 的数据库中，那么必须在参数文件中包含一个 DB_8K_CACHE_SIZE 初始化参数入口。如果它还没有包含在参数文件中，该参数可以使用 ALTER SYSTEM SET 语句来设置。

要插入一个表空间集合，执行下列任务：

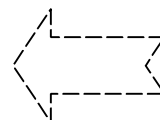
①使用 Import 工具插入该表空间和集成结构信息。

```
IMP TRANSPORT_TABLESPACE=y FILE=expdat.dmp
```

```
DATAFILES=('db/sales_jan','db/sales_feb',...)
```

```
TABLESPACES=(sales_1,sales_2) TTS_OWNERS=(dcranney,jfee)
```

```
FROMUSER=(dcranney,jfee) TOUSER=(smith,williams)
```



当被提示时,用具有 SYSDBA 系统权限的 SYS (或者其他管理用户) 进行连接:

```
CONNECT SYS/password AS SYSDBA.
```

在本例中指定如下内容:

- TRANSPORT_TABLESPACE = y 通知 Export 工具正在传输一个表空间。
- 该表空间包含元数据的输出文件是 expdat.dmp。
- DATAFILES 指定被传输表空间的数据文件, 并且必须指定。
- 表空间名为 sales_1 和 sales_2。

当指定 TABLESPACES 时, 提供的表空间名应与输出文件中的比较。如果有任何的不匹配, Import 返回一个错误。换句话说, 表空间名应从输出文件中提取。

- TTS_OWNERS 列出在该表空间集合中拥有数据的所有用户。

当指定 TTS_OWNERS 时, 用户名应与输出文件中的一致。如果有任何的不匹配, Import 返回一个错误。换句话说, 用户名应从输出文件中提取。

- 指定 FROMUSER 和 TOUSER 来改变数据库对象的所有权。

如果没有指定 FROMUSER 和 TOUSER, 所有数据库对象 (比如表和索引) 都以与源数据库相同的用户名义创建。这些用户必须已经在目标数据库中存在。如果没有, Import 返回一个错误指出一些需要的用户在目标数据库中不存在。

可以使用 FROMUSER 和 TOUSER 来改变对象的所有权。在本例中, 指定 FROMUSER = (dcranney, jfee) 和 TOUSER = (smith, williams)。该表空间集合被插入以后, 在源数据库中属于 dcranney 的表空间集合对象将在目标数据库中属于 smith。同样地, 在源数据库中属于 jfee 的对象将在目标数据库中属于 williams。在这种情况下, 目标数据库不需要拥有用户 dcranney 和 jfee, 但必须有用户 smith 和 williams。

在该语句成功执行以后, 该集合中所有的表空间仍以只读模式被复制。检查输入日志确保没有出现错误。

当处理大量的数据文件时, 在语句行中指定数据文件名的列表可能是个费力的过程。它甚至可能会超出语句行的限制。在这种情况下, 可以使用一个输入参数文件。例如, 可以调用如下的 Import 工具:

```
IMP PARFILE='par.f'
```

文件 par.f 包含如下内容:

```
TRANSPORT_TABLESPACE=y
```

```
FILE=expdat.dmp
```

```
DATAFILES=('/db/sales_jan','/db/sales_feb',...)
```

```
TABLESPACES=(sales_1,sales_2)
```

```
TTS_OWNERS=(dcranney,jfee)
```

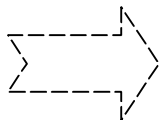
```
FROMUSER=(dcranney,jfee)
```

```
TOUSER=(smith,williams)
```

②如果必要, 使所复制的空间中的表空间重新回到可读写模式, 如下:

```
ALTER TABLESPACE sales_1 READ WRITE
```

```
ALTER TABLESPACE sales_1 READ WRITE
```



11.8.5 对象行为

大多数对象，不管在表空间中的数据还是与表空间相关联的结构信息，在被传输到一个不同的数据库后，都正常地工作。

然而，下列对象是例外：

- ROWID
- REF
- 权限
- 分区表
- 对象
- 高级队列
- 索引
- 触发器
- 物化视图/复制

11.8.5.1 ROWID

当一个数据库包含一个已经插入（从其他的数据库）的表空间，该数据库的 ROWID 就不再是唯一的了。ROWID 只在一个表中保证是唯一的。

11.8.5.2 REF

当 Oracle 确定一个表空间集合是否为自含式的时候，REF 不被检查。其结果是，一个插入的表空间可能包含虚悬 REF。任何紧跟虚悬 REF 的查询都返回一个用户错误。

11.8.5.3 权限

如果在输出期间指定了 GRANTS = y，权限被传输。在输入期间，一些授权可能失败。例如，被授予某种权利的用户可能不存在，或者一个被授予一项特殊的权利的角色可能不存在。

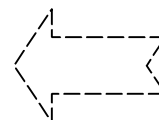
11.8.5.4 分区表

当只有一个分区表的子集包含在表空间集合中时，不能使用可传输表空间来移动一个分区表。必须确保表中所有分区都在该表空间集合中，或者在复制该表空间集合前，将分区交换到表中。但是，应该注意到与表交换分区将使该分区表的全局索引无效。

在目标数据库，如果已经有一个分区表与目标数据库中的列完全匹配，那么可以将表交换回分区。如果该表所有的分区来自同一个外部数据库，交换操作可以保证成功。如果它们不是，在极少情况中，交换操作可能返回一个错误指出有数据对象号冲突。

如果在交换表返回分区的时候，收到一个数据对象号冲突错误，可以使用 ALTER TABLE MOVE PARTITION 语句移走那个讨厌的分区。在这样做之后，重试该交换操作。

如果指定了交换语句的 WITHOUT VALIDATION 选项，该语句立即返回，因为它只处理结构信息。但是，移动分区表可能很慢，因为该分区表中的数据可以被复制。



11.8.5.5 对象

一个可传输表空间集合可以包含：

- 表
- 索引
- 域索引
- 位图索引
- 索引结构的表
- LOB
- 嵌套表
- Varray
- 带有用户定义类型的列的表

如果该表空间包含一个指向 BFILE 的指针，必须移动该 BFILE，并且在目标数据库中正确设置目录。

11.8.5.6 高级队列

可以使用可传输表空间来移动或复制 Oracle 高级队列，只要这些队列不是有多个接受者的 8.0 兼容队列。在一个队列被传输到目标数据库之后，该队列初始化是禁用的。在使得被传输的表空间在目标数据库中成为可读写的之后，可以通过使用内置的 PL/SQL 例程 DBMS_AQADM.START_QUEUE 启动它来启用该队列。

11.8.5.7 索引

可以传输常规索引、域索引和位图索引。当可传输集合完全包含一个分区表时，也可以传输该分区表的全局索引。

基于函数的索引不被支持。如果它们在一个表空间中存在，必须在能够传输该表空间之前取消它们。

11.8.5.8 触发器

触发器不用有效性检查就被输出。换句话说，Oracle 不检验触发器只引用该可传输集合的对象。在随后的输入期间，无效触发器产生一个编译错误。

11.8.5.9 物化视图/复制

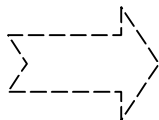
传输物化视图或复制结构信息不被支持。当传输一个表空间时，与该表空间中的表相关联的物化视图或复制元数据不被输出，并且因而不能为目标数据库所用。

11.8.6 使用可传输表空间

下面是可传输表空间的一些可能应用。

11.8.6.1 为数据仓库传输和附加分区

典型的企业数据仓库包含一个或更多的大型事实表。这些事实表可以由数据分区，使



得企业数据仓库成为一个历史数据库。可以建立索引来加速星形查询。事实上，Oracle 建议为这类历史分区表建立本地索引，以避免每次从该历史数据库中取消最古老的分区都要重建全局索引。

假定每个月都要将一个月的数据财产装载到数据仓库中。数据仓库中有一个名为 `sales` 的大型事实表，它有如下的列：

```
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));
创建一个本地的非前缀索引：
```

```
CREATE INDEX sales_index ON sales(invoice_no) LOCAL;
```

最初，所有的分区都是空的，并且在同一个默认的表空间中。每个月，想要创建一个分区并将它附加到分区的 `sales` 表。

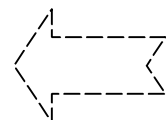
假定现在是 1998 年 7 月，并且想要将 7 月份的销售数据装载到分区表。在一个分级数据库中，创建一个新的表空间 `ts_jul`。还要在与 `sales` 表有完全相同的列类型的表空间中创建一个表 `jul_sales`。可以使用 `CREATE TABLE ... AS SELECT` 语句创建表 `jul_sales`。在创建和填充 `jul_sales` 之后，还可以为该表创建一个索引 `jul_sale_index`，编出与 `sales` 表中本地索引相同列的索引。建立索引之后，将表空间 `ts_jul` 传输到数据仓库。

在数据仓库中，为 7 月份的销售数据添加一个分区到 `sales` 表。这也为本地的非前缀索引创建另一个分区：

```
ALTER TABLE sales ADD PARTITION jul98 VALUES LESS THAN (1998, 8, 1);
通过将表传输的表 jul_sales 与新的分区交换，把它附加到表 sales:
ALTER TABLE sales EXCHANGE PARTITION jul98 WITH TABLE jul_sales
INCLUDING INDEXES
WITHOUT VALIDATION;
```

这条语句将 7 月份的销售数据放入新分区 `jul98`，将新的数据附加到该分区表。该语句也将索引 `jul_sale_index` 转换为 `sales` 表本地索引的一个分区。这条语句应该立即返回，因为它只对结构信息操作，只是交换数据库指针。如果知道新分区中的数据没有与原来分区的数据重叠，建议指定 `WITHOUT VALIDATION` 选项。否则，该语句仔细检查新分区的所有新数据，试图让该分区范围都有效。

如果 `sales` 表中的所有分区来自同一个分级数据库（该分级数据库从未被破坏），交换



语句通常会成功。然而，一般而言，如果一个分区表中的数据来自不同的数据库，交换操作就有可能失败。例如，如果 sales 表中的 jan98 分区不是来自同一个分级数据库，上述的交换操作可能失败，返回如下错误信息：

```
ORA-19728: data object number conflict between table JUL_SALES and partition
JAN98 in table SALES
```

要解决这种冲突，通过发布如下的语句将这个讨厌的分区移走：

```
ALTER TABLE sales MOVE PARTITION jan98;
```

然后重试该交换操作。

在交换成功以后，可以安全地取消 jul_sales 和 jul_sale_index（两者现在都是空的）。于是，已经成功地将 7 月份的销售数据装载到数据仓库中。

11.8.6.2 在 CD 上发布结构化数据

可传输表空间提供一种在 CD 上发布结构化数据的方法。一个数据提供者可以装载一个带有要发布数据的表空间，生成可传输集合，并且将该可传输集合复制到一张 CD 上。然后该 CD 可以被分发。

当消费者收到该 CD 时，他们可以将它插入到一个现有的数据库，而不必将数据文件从 CD 复制到磁盘存储区。例如，假定在一个 Windows NT 机器上，D: 驱动器是 CD 驱动器。可以插入一个带有数据文件 catalog.f 和输出文件 expdat.dmp 的可传输集合，如下：

```
IMP TRANSPORT_TABLESPACE=y DATAFILES='D:\catalog.f' FILE='D:\expdat.dmp'
```

可以将该 CD 移走而数据库仍然打开。随后对该表空间的查询返回一个错误指出 Oracle 不能打开 CD 上的数据文件。然而，对该数据库其他部分的操作不受影响。将该 CD 放回驱动器使得该表空间又可读了。

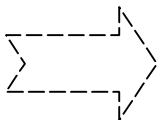
移走该 CD 与移走一个只读表空间的数据文件是一样的。如果关闭和重启动数据库，Oracle 指出它找不到移走的数据文件和不能打开该数据库（除非设置初始化参数 READ_ONLY_OPEN_DELAYED 为 TRUE）。当 READ_ONLY_OPEN_DELAYED 被设置为 TRUE 时，Oracle 只在有人查询插入的表空间时才读文件。因而，当插入一个 CD 上的表空间时，通常应该将初始化参数 READ_ONLY_OPEN_DELAYED 设置为 TRUE，除非该 CD 永久地连着数据库。

11.8.6.3 在多个数据库上装载同一个只读表空间

可以使用可传输表空间在多个数据库上装载一个只读表空间。这样，分开的数据库可以共享磁盘上的相同数据，而不用将数据复制到分开的磁盘上。该表空间数据文件必须可以被所有的数据库访问。要避免数据库损坏，该表空间必须在所有装载它的数据库中保持只读。

可以用下列任意一种方法，在多个数据库上装载同一个只读表空间：

- 将该表空间插入到每个想要装载它的数据库中。在单个的表空间中生成一个可传输集合。将数据文件放入可传输集合中，该可传输集合位于一个可被所有数据库访问的磁盘上。将结构信息输入到每个数据库中。



- 在其中一个数据库中生成可传输集合。并且将它插入到其他数据库中。如果使用这种方法，它假定了数据文件已经在共享磁盘上，并且它们属于其中一个数据库的现有表空间。可以使该表空间只读，生成可传输集合，然后将该表空间插入到其他数据库中，而数据文件仍保持在共享磁盘的同样位置。

可以用多种方法使该磁盘可以被多个计算机访问。可以使用簇文件系统或原始磁盘，因为这是 Oracle9i Real Application Clusters 要求的。因为 Oracle 只读取共享磁盘上这些类型的数据文件，也可以使用 NFS。但是，要注意，如果一个用户查询共享表空间而 NFS 停了，该数据库将暂停直到 NFS 操作超时。

随后，可以在一些这样的数据库中取消该只读表空间。这样做不改变该表空间的数据文件。因而，这个取消操作不会损坏该表空间。不要使该表空间成为可读写的，除非只有一个数据库装载该表空间。

11.8.6.4 使用可传输表空间存档历史数据

既然一个可传输表空间集合是一个可插入任何 Oracle 数据库中的文件的自含式集合，可以使用本章介绍的可传输表空间过程将旧/历史数据存档到一个企业数据仓库中。

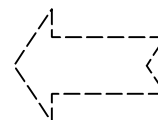
11.8.6.5 使用可传输表空间执行 TSPITR

可以使用可传输表空间来执行表空间 point-in-time 恢复（TSPITR）。

11.9 查看表空间信息

下列数据字典视图和动态性能视图提供关于一个数据库的表空间的有用信息。

视图	说明
V\$TABLESPACE	来自控制文件的所有表空间名称和编号。
DBA_TABLESPACE, USER_TABLESPACE	所有（或用户可访问的）表空间的说明。
DBA_SEGMENTS, USER_SEGMENTS	关于所有（或用户可访问的）表空间中的段的信息。
DBA_EXTENTS, USER_EXTENTS	关于所有（或用户可访问的）表空间中的数据盘区的信息。
DBA_FREE_SPACE, USER_FREE_SPACE	关于所有（或用户可访问的）表空间中的空闲盘区的信息。
V\$DATAFILE	关于所有数据文件的信息，包括所属表空间的表空间号。
V\$TEMPFILE	关于所有临时文件的信息，包括所属表空间的表空间号。
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	所有本地管理的临时表空间中的所有盘区的信息。
V\$TEMP_EXTENT_POOL	对于本地管理的临时表空间：由每个实例缓存和使用的临时表空间的状态。
V\$TEMP_SPACE_HEADER	显示每个临时文件的已用/空闲空间。



(续表)

视图	说明
DBA_USERS	所有用户的默认的和临时的表空间。
DBA_TS_QUOTAS	列出所有用户的表空间限额。
V\$SORT_SEGMENT	关于一个给定实例的每个排序段的信息。该视图只在表空间是 TEMPORARY 类型时更新。
V\$SORT_USER	用户使用的临时排序空间和临时的/永久的表空间。

下面是几个使用这些视图的例子。

11.9.1 列出表空间和默认存储参数的例子

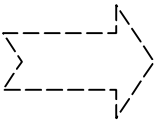
要列出一个数据库中所有表空间的名称和默认的存储参数，在 DBA_TABLESPACES 视图中使用如下的查询：

```
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
```

11.9.2 列出数据库的数据文件和关联的表空间的例子

要列出一个数据库中数据文件的名称、大小和关联的表空间，在 DBA_DATA_FILES 视图中输入如下的查询：

```
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
```



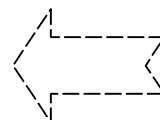
```
/U02/ORACLE/IDDB3/USERS01.DBF 384 USERS
```

11.9.3 为每个表空间的空闲空间（盘区）显示统计信息的例子

要为数据库中每个表空间生成关于空闲盘区和合并动作的信息，输入如下的查询：

```
SELECT TABLESPACE_NAME "TABLESPACE", FILE_ID,
COUNT(*) "PIECES",
MAX(blocks) "MAXIMUM",
MIN(blocks) "MINIMUM",
AVG(blocks) "AVERAGE",
SUM(blocks) "TOTAL"
FROM 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 显示该表空间文件中的空闲空间盘区数，MAXIMUM 和 MINIMUM 显示数据库块中最大和最小的空间相邻区域，AVERAGE 显示一个空闲空间盘区以块计的平均大小，而 TOTAL 显示每个表空间文件中空闲空间以块计的数量。当要创建一个新的对象或知道一个段要盘区，并且想要确保在包含的表空间中有足够的空间时，这条查询是有用的。



第 12 章 管理数据文件

本章介绍管理数据文件的各个方面，包括以下内容：

- 管理数据文件的准则
- 创建数据文件和向表空间添加数据文件
- 改变数据文件的大小
- 修改数据文件可用性
- 重命名和重定位数据文件
- 验证数据文件中的数据块
- 查看数据文件信息

12.1 管理数据文件的准则

数据文件是操作系统储存数据库中所有逻辑结构的数据的物理文件。它们必须明确地为每个表空间所创建。Oracle 为每个数据文件指定两个关联的文件号，一个绝对的文件号和一个相对的文件号，它们用于唯一地标识该数据文件。这些文件号在下表中进行说明：

文件号类型	说明
绝对的	在数据库中唯一地标识一个数据文件。在 Oracle 较早的版本中，绝对文件号被简单地称作“文件号”。
相对的	在表空间中唯一地标识一个数据文件。对于较小和中等大小的数据库，相对文件号通常与绝对文件号具有相同的值。但是，当一个数据库中的数据文件数超过阈值（典型的为 1023）时，相对文件号与绝对文件号就不一样了。

文件号可以在很多数据字典视图中显示。

本节介绍管理数据文件的方面，包括如下内容：

- 确定数据文件的编号
- 确定数据文件的大小
- 适当地放置数据文件
- 将数据文件与重做日志文件分开存储

12.1.1 确定数据文件的编号

一个数据库的 SYSTEM 表空间至少需要一个数据文件。一个较小的系统可能只有一个单一的数据文件。下面是为数据库确定数据文件编号时，要考虑的一些准则。

12.1.1.1 确定初始化参数 DB_FILES 的值

当启动一个 Oracle 实例时，初始化参数 DB_FILES 指出为数据文件信息保留的 SGA 空间的数量，因此，也指出了可以为该实例创建的数据文件的最大编号。这个限制适用于该实例的整个使用期间。可以改变 DB_FILES 的值（通过改变初始化参数设置），但是新值直到关闭和重新启动该实例后才会起作用。

注意：DB_FILES 的默认值与操作系统有关。

当为 DB_FILES 确定一个值时，考虑如下因素：

- 如果 DB_FILES 值太小，就不能在第一次关闭数据库之前，添加超过 DB_FILES 限制的数据文件。
- 如果 DB_FILES 值太高，内存被不必要地消耗。

12.1.1.2 添加数据文件到表空间时的限制

可以添加数据文件到表空间，服从下列限制条件：

- 操作系统经常对一个进程能够同时打开的文件数加以限制。当达到操作系统打开文件的限度时，就不能再创建数据文件了。
- 操作系统对数据文件的数量和大小加以限制。
- Oracle 对任何实例打开的任何 Oracle 数据库数据文件的最大数量加以限制。该限制与操作系统有关。
- 不能超过由初始化参数 DB_FILES 所指定的数据文件数量。
- 当发布 CREATE DATABASE 或 CREATE CONTROLFILE 语句时，参数 MAXDATAFILES 指定控制文件的数据文件部分的一个初始大小。如果 MAXDATAFILES 的值小于或等于初始化参数 DB_FILES 所指定的值，就只能添加一个新的数据文件。在这种情况下，控制文件自动盘区以允许数据文件部分容纳更多的文件。

12.1.1.3 考虑对性能的影响

组成表空间的数据文件的数目，归根结底是数据库，能够对性能产生影响。

Oracle 允许数据库中有比操作系统规定的限制更多的数据文件。Oracle 的 DBW n 进程能够打开所有联机的数据文件。Oracle 能够将打开文件描述符当成一个缓存，当打开文件的描述符数目达到操作系统规定的限制时，自动关闭文件。这对性能有负面的影响。在可能的时候，调整操作系统在打开文件描述符上的限制，使它大于数据库中联机数据文件的数目。

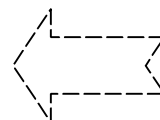
12.1.2 确定数据文件的大小

第一个数据文件（在原始的 SYSTEM 表空间中）必须至少有 150M 以包含初始的数据字典和回滚段。如果安装其他的 Oracle 产品，它们在 SYSTEM 表空间中可能需要额外的空间。参见这些产品的安装指南，了解关于它们对空间需求的信息。

12.1.3 适当地放置数据文件

表空间的位置由组成该表空间的数据文件的物理位置所确定。适当地使用计算机的硬件资源。

例如，如果数个磁盘驱动器可用于存储数据库，考虑将可能竞争的数据文件放置在分开的磁盘上。使用这种方法，当用户查询信息时，两个磁盘驱动器可以同时工作，同时检索数据。



12.1.4 将数据文件与重做日志文件分开存储

数据文件不应该与数据库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，并且该磁盘驱动器出现故障，这些文件就不能被数据库的恢复过程所使用。

如果重做日志文件有多个，那么丢失所有的重做日志文件的可能性就很小，因此可以将数据文件与一些重做日志文件存储在同一个驱动器上。

12.2 创建数据文件和向表空间添加数据文件

当创建表空间时，可以估计数据库对象的可能大小，并在多个设备上创建足够的文件，这样做是要确保数据均匀地分布在所有设备上。随后，如果需要，可以创建额外的数据文件并将它们添加到表空间，以增加分配给该表空间的磁盘空间的总量，并因此增加数据库的空间。

可以使用下表中列出的任一语句来创建数据文件并将它们与表空间相连。在所有的情况中，既可以为正在创建的数据库指定文件规格，也可以使用 Oracle Managed File（Oracle 管理文件）特性来创建由数据库服务器创建和管理的文件。下表包含对这些语句用于创建数据文件的简短说明，参考本书中使用这些语句的章节有最完整的说明：

SQL 语句	说明
CREATE TABLESPACE	创建一个表空间和组成它的数据文件
CREATE TEMPORARY TABLESPACE	创建一个本地管理的临时表空间和组成它的临时文件 (临时文件是一种特殊类型的数据文件)
ALTER TABLESPACE ... ADD DATAFILE	创建并添加一个数据文件到表空间
ALTER TABLESPACE ... ADD TEMPFILE	创建并添加一个临时文件到临时表空间
CREATE DATABASE	创建一个数据库和关联的数据文件
ALTER DATABASE ... CREATE DATAFILE	创建一个新的空数据文件代替原有的数据文件——有助于重新创建一个已丢失的且没有备份的数据文件

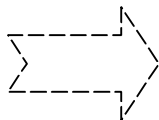
如果添加新的数据文件到表空间，且没有完全地指定文件名，Oracle 在默认的数据库目录或当前目录下创建这些文件，这取决于操作系统。Oracle 建议用户总是为数据文件指定完全合格的名称。除非用户想要重用现有的文件，否则，确保这些新文件名不与其他文件相冲突。以前已经被取消的原有文件将会被重写。

如果一条创建数据文件的语句失败，Oracle 删除任何创建的操作系统文件。但是，因为文件系统和存储子系统中存在大量可能出现的错误，在这种情况下，用户必须使用操作系统命令来手动删除这些文件。

12.3 改变数据文件的大小

本节说明修改一个数据文件大小的各种方法，包括如下内容：

- 启用和禁用数据文件的自动盘区
- 手动调整数据文件大小



12.3.1 启用/禁用数据文件的自动盘区

可以创建数据文件或修改现有数据文件，以便它们在数据库中需要更多的空间时，能够自动增加大小。文件以指定的增量增加到指定的最大值。

将数据文件设置为自动盘区可以提供如下优越性：

- 当表空间用完空间时，减少立即干预的必要
- 确保应用程序不会因分配盘区失败而暂停

要确定一个数据文件是否是自动盘区，查询 `DBA_DATA_FILES` 视图并检查 `AUTOEXTENSIBLE` 列。

在创建数据文件时，可以使用下列 SQL 语句，通过指定 `AUTOEXTEND ON` 子句来指定自动文件盘区：

- `CREATE DATABASE`
- `CREATE TABLESPACE`
- `ALTER TABLESPACE`

可以启用或禁用现有数据文件的自动文件盘区，或者使用 SQL 语句 `ALTER DATABASE` 手动调整一个数据文件的大小。

下面的例子启用一个添加到 `users` 表空间的数据文件的自动盘区：

```
ALTER TABLESPACE users
ADD DATAFILE '/u02/oracle/rbdb1/users03.dbf' SIZE 10M
AUTOEXTEND ON
NEXT 512K
MAXSIZE 250M;
```

`NEXT` 的值是文件盘区时所增加的增量的最小大小。`MAXSIZE` 的值是文件能够自动盘区的最大大小。

接下来的例子禁用该数据文件的自动盘区。

```
ALTER DATABASE DATAFILE '/u02/oracle/rbdb1/users03.dbf'
AUTOEXTEND OFF;
```

12.3.2 手动调整数据文件大小

可以使用 `ALTER DATABASE` 语句，手动地增加或减少一个数据文件的大小。

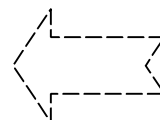
因为可以改变数据文件的大小，就可以为数据库增加更多的空间而不用添加更多的数据文件。如果关注的是接近数据库所允许的最大数据文件数，这是有用的。

手动减小数据文件的大小，使得能够回收数据库中没有使用的空间。这在估计空间需求时改正错误是有用的。

在接下来的例子中，假定数据文件 `/u02/oracle/rbdb1/stuff01.dbf` 已经盘区到 250M。但是，因为它的表空间现在存储的是较小的对象，该数据文件可以减小大小。

下面的语句减小数据文件 `/u02/oracle/rbdb1/stuff01.dbf` 的大小：

```
ALTER DATABASE DATAFILE '/u02/oracle/rbdb1/stuff01.dbf'
RESIZE 100M;
```



注意：不是总有可能将一个文件的大小减小到一个指定值。

12.4 修改数据文件可用性

可以使一个表空间的单个数据文件或临时文件脱机，或者同样地，使它们联机。脱机的数据文件不能被数据库使用，并且不能被访问，直到它们被重新联机。也可以选择只指定表空间的名称使组成一个表空间的所有数据文件或临时文件脱机或联机。

一个用户可能被要求修改数据文件的可用性的例子，是 Oracle 在写一个数据文件出现问题，并自动使该数据文件脱机。随后，问题解决之后，可以手动地使该数据文件重新联机。

一个只读表空间的文件可以被单独地脱机或联机，正如对读写的表空间那样。在一个只读表空间中使一个数据文件联机，将使该文件可读。不能对该文件写，除非它相连的表空间返回到读写状态。

要使一个数据文件脱机，或使它联机，必须拥有 ALTER DATABASE 系统权限。要使用 ALTER TABLESPACE 语句，使所有的数据文件或临时文件脱机，必须拥有 ALTER TABLESPACE 或 MANAGE TABLESPACE 系统权限。在 Oracle Real Application Clusters 环境中，数据库必须以独占模式打开。

本节介绍修改数据文件可用性的方法，包括如下内容：

- 在 ARCHIVELOG 模式中使数据文件联机或脱机
- 在 NOARCHIVELOG 模式中使数据文件脱机
- 修改表空间中所有数据文件或临时文件的可用性

注意：可以通过使表空间脱机，使除 SYSTEM 表空间中的文件以外的任何表空间中的所有数据文件临时不可用。必须将这些文件留在表空间中，以使该表空间重新联机。

12.4.1 在 ARCHIVELOG 模式中使数据文件联机或脱机

要使一个单独的数据文件联机，发布包括 DATAFILE 子句的 ALTER DATABASE 语句。下面的语句使指定的数据文件联机：

```
ALTER DATABASE DATAFILE '/u02/oracle/rbdb1/stuff01.dbf' ONLINE;
```

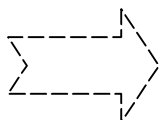
要使同一文件脱机，发布如下的语句：

```
ALTER DATABASE DATAFILE '/u02/oracle/rbdb1/stuff01.dbf' OFFLINE;
```

注意：要使用 ALTER DATABASE 语句的这个选项，数据库必须处于 ARCHIVELOG 模式。该要求防止用户意外丢失数据文件，因为在 NOARCHIVELOG 模式中使数据文件脱机，很可能导致丢失文件。

12.4.2 在 NOARCHIVELOG 模式中使数据文件脱机

当数据库处于 NOARCHIVELOG 模式时，要使一个数据文件脱机，使用同时带有 DATAFILE 和 OFFLINE DROP 子句的 ALTER DATABASE 语句。这使得用户可以使数据文件脱机并立即取消它。这是有用的，例如，如果数据文件只包含来自临时段的数据，并且没有进行备份，而数据库处于 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;
```

12.4.3 修改表空间中所有数据文件或临时文件的可用性

ALTER TABLESPACE 语句的子句允许用户改变一个表空间中所有数据文件或临时文件的联机或脱机状态。特别地，影响联机/脱机状态的语句有：

- ALTER TABLESPACE ... DATAFILE { ONLINE | OFFLINE }
- ALTER TABLESPACE ... TEMPFILE { ONLINE | OFFLINE }

只要求用户输入表空间名称，而不是单个的数据文件或临时文件。所有的数据文件或临时文件都受到影响，但是表空间本身的联机/脱机状态不改变。

在大多数情况下，上述的 ALTER TABLESPACE 语句都可以在装载数据库时发布，即使它没有被打开。但是，如果该表空间是系统表空间，一个撤消表空间或者默认的临时表空间，数据库必须是没有打开的。ALTER DATABASE DATAFILE 和 ALTER DATABASE TEMPFILE 语句也有 ONLINE/OFFLINE 子句，但是在这些语句中，必须输入表空间所有的文件名。

语法与修改一个表空间的可用性的 ALTER TABLESPACE ... ONLINE | OFFLINE 语句不一样，因为这是一个不一样的操作。ALTER TABLESPACE 语句使数据文件以及表空间脱机，但不能用于修改临时表空间或其临时文件的状态。

12.5 重命名和重定位数据文件

可以对数据文件重命名来改变它们的名称或者对它们重定位。一些选项和可以遵循的过程，在下面的章节中说明：

- 单个表空间的数据文件的重命名和重定位

例如，重命名 tablespace1 中的 filename1 和 filename2，而数据库的其余部分是打开的。

- 多个表空间的数据文件的重命名和重定位

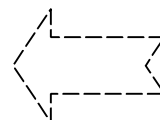
例如，重命名 tablespace1 中的 filename1 和 tablespace2 中的 filename2，而数据库被装载但是关闭的。

注意：要重命名或重定位 SYSTEM 表空间的数据文件，用户必须使用第二个选项，因为用户不能使 SYSTEM 表空间脱机。

当利用这些过程重命名和重定位数据文件时，只有数据文件的指针，像在数据库控制文件中记录的那样，被改变了。这些过程不能物理地重命名任何操作系统文件，也不能在操作系统层复制文件。重命名和重定位数据文件包含几个步骤。在执行上述过程之前，请仔细阅读这些步骤和例子。

12.5.1 单个表空间的数据文件的重命名和重定位

本节提供一些对单个表空间中的数据文件重命名和重定位的过程。必须拥有 ALTER TABLESPACE 系统权限来重命名单个表空间的数据文件。



12.5.1.1 单个表空间的数据文件的重命名

要重命名单个表空间的数据文件，完成下列步骤：

①使包含该数据文件的非 SYSTEM 表空间脱机。

例如：

```
ALTER TABLESPACE users OFFLINE NORMAL;
```

②利用操作系统重命名该数据文件。

③使用带有 RENAME DATAFILE 选项的 ALTER TABLESPACE 语句，在数据库中改变文件名。

例如，如下的语句将数据文件/u02/oracle/rbdb1/user1.dbf 和/u02/oracle/rbdb1/user2.dbf 分别重命名为/u02/oracle/rbdb1/users01.dbf 和/u02/oracle/rbdb1/users02.dbf：

```
ALTER TABLESPACE users
RENAME DATAFILE '/u02/oracle/rbdb1/user1.dbf',
'/u02/oracle/rbdb1/user2.dbf'
TO '/u02/oracle/rbdb1/users01.dbf',
'/u02/oracle/rbdb1/users02.dbf';
```

新文件必须已经存在；该语句不能创建文件。另外，提出提供完整的文件名（包括它们的路径）来正确地识别以前的数据文件和新的数据文件。特别是，准确指定以前的数据文件名称，跟它在数据字典的 DBA_DATA_FILES 视图中显示的一样。

④备份数据库。在对数据库作出结构上的改变之后，通常执行一次立即和完整的备份。

12.5.1.2 单个表空间的数据文件的重定位和重命名

这里用一个例子举例说明了重定位一个数据文件所包含的步骤。

假定下列的条件：

- 一个打开的数据库，有一个名为 users 的表空间，组成表空间的所有数据文件位于同一磁盘。
- users 表空间的数据文件将被重定位到不同的、分开的磁盘驱动器。
- 目前以管理员权限与该打开的数据库相连。
- 有一个该数据库的当前备份。

完成下列步骤：

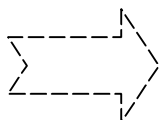
①标识感兴趣的数据文件名称。

数据字典视图 DBA_DATA_FILES 的如下查询列出了 users 表空间的数据文件名称和各自的大小（以字节计）：

```
SELECT FILE_NAME, BYTES FROM DBA_DATA_FILES
WHERE TABLESPACE_NAME = 'USERS';
FILE_NAME BYTES
```

```
-----
/U02/ORACLE/RBDB1/USERS01.DBF 102400000
```

```
/U02/ORACLE/RBDB1/USERS02.DBF 102400000
```



②使包含该数据文件的表空间脱机，或者关闭数据库并重新启动和装载它，但使数据库处于关闭状态。任一种选择都关闭了该表空间的数据文件。

③将数据文件复制到它们的新位置，并利用操作系统对它们重命名。

注意：可以通过使用 SQL*Plus HOST 命令来执行一个复制文件的操作系统命令。

④在 Oracle 中对数据文件重命名。

组成 users 表空间的那些文件的数据文件指针，记录在相连的数据库的控制文件中，现在必须从以前的名称改变为新的名称。

如果该表空间是脱机的，但数据库是打开的，使用 ALTER TABLESPACE ... RENAME DATAFILE 语句。如果数据库被装载，但是关闭的，使用 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';
```

⑤使该表空间联机，或者打开数据库。

如果 users 表空间是脱机的，并且数据库是打开的，使该表空间重新联机。如果数据库被装载，但是关闭的，打开该数据库。

⑥备份数据库。在对数据库作出任何结构上的改变之后，通常执行一次立即和完全的备份。

12.5.2 多个表空间的数据文件的重命名和重定位

使用带有 RENAME FILE 选项的 ALTER DATABASE 语句，可以重命名和重定位一个或更多表空间的数据文件。该选项是想要在一次操作中重命名或重定位几个表空间的数据文件，或者重命名或重定位 SYSTEM 表空间的数据文件的唯一选择。如果该数据库必须保持打开状态，就要考虑替换前面章节中概述的过程。

要在一次操作中重命名几个表空间的数据文件，或者重命名 SYSTEM 表空间的数据文件，必须拥有 ALTER DATABASE 系统权限。

要重命名多个表空间中的数据文件，遵循下列步骤：

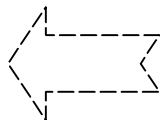
①确保数据库被装载，但是关闭的。

②利用操作系统，将要重命名的数据文件复制到新位置和新名称。

③使用 ALTER DATABASE，重命名数据库控制文件中该文件指针。

例如，如下的语句将数据文件/u02/oracle/rbdb1/sort01.dbf 和/u02/oracle/rbdb1/user3.dbf 分别重命名为/u02/oracle/rbdb1/temp01.dbf 和/u02/oracle/rbdb1/users03.dbf:

```
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;
```

新文件必须已经存在；该语句不能创建文件。另外，通常提供完全的文件名（包括它们的路径）来正确地识别以前的数据文件和新的数据文件。特别是，准确地指定以前的数据文件名称，与在数据字典的 DBA_DATA_FILES 视图中显示的一样。

④备份数据库。在对数据库作出任何结构上的改变之后，通常执行一次立即和完全的备份。

12.6 验证数据文件中的数据块

如果想要配置 Oracle 以使用校验和来验证数据块，将初始化参数 DB_BLOCK_CHECKSUM 设置为 TRUE。该参数的值可以动态改变，或者在初始化参数文件中设置。DB_BLOCK_CHECKSUM 的默认值是 FALSE。不管该参数的设置如何，校验和通常用于在系统表空间中验证数据块。

当启用块校验时，Oracle 为每个写到磁盘的块计算一个校验和。对所有的数据块都计算校验和，包括临时块。

DBWn 进程为每个块计算校验和，并将它存储在该块的头部。校验和也可由直接的装载程序来计算。

下一次 Oracle 读一个数据块时，它使用校验和来检测该块中的讹误。如果一个讹误被检测出来，Oracle 返回信息 ORA-01578，并将关于该讹误的信息写入一个跟踪文件。

警告：将 DB_BLOCK_CHECKSUM 设置为 TRUE 可能造成性能开销。只有在 Oracle Support 人员的建议下，将该参数设置为 TRUE 来诊断数据讹误问题。

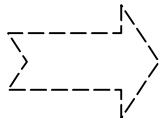
12.7 查看数据文件信息

下列数据字典视图提供关于一个数据库的数据文件的有用信息：

视图	说明
DBA_DATA_FILES	提供关于每个数据文件的说明信息，包括它所属的表空间和文件标识。该文件标识可以用于加入其他视图得到详细信息。
DBA_EXTENTS	DBA 视图说明组成数据库中所有段的盘区。包括包含该盘区的数据文件的文件标识。USER 视图描述了属于当前用户所拥有的对象的段的盘区。
USER_EXTENTS	
DBA_FREE_SPACE	DBA 视图列出所有表空间中的空闲盘区。包括包含该盘区的数据文件的文件标识。USER 视图列出可被当前用户访问的表空间中空闲盘区。
USER_FREE_SPACE	
V\$DATAFILE	包括来自控制文件的数据文件信息
V\$DATAFILE_HEADER	包括来自数据文件头部的信息

本例子举例说明这些视图中的 V\$DATAFILE 的使用。

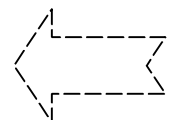
```
SELECT NAME,
       FILE#,
       STATUS,
```



```
CHECKPOINT_CHANGE# "CHECKPOINT"  
FROM V$DATAFILE;  
NAME FILE# STATUS CHECKPOINT
```

```
-----  
/u01/oracle/rbdb1/system01.dbf 1 SYSTEM 3839  
/u02/oracle/rbdb1/temp01.dbf 2 ONLINE 3782  
/u02/oracle/rbdb1/users03.dbf 3 OFFLINE 3782
```

FILE#列出每个数据文件的文件号；在该数据库的 SYSTEM 表空间中第一个创建的数据文件通常是文件 1。STATUS 列出关于一个数据文件的其他信息。如果一个数据文件是 SYSTEM 表空间的一部分，它的状态是 SYSTEM（除非它需要恢复）。如果一个非 SYSTEM 表空间的数据文件是联机的，它的状态是 ONLINE。如果一个非 SYSTEM 表空间的数据文件是脱机的，它的状态既可以是 OFFLINE，也可以是 RECOVER。CHECKPOINT 列出为数据文件最近的检测点写入的最终 SCN（系统改变数）。



第 13 章 管理撤消空间

本章介绍如何通过使用撤消表空间或回滚段管理撤消（Undo）空间。包括以下内容：

- 什么是撤消
- 指定管理撤消空间的方式
- 管理撤消表空间
- 管理回滚段

13.1 什么是撤消

每个 Oracle 数据库必须有一种维护用于回滚或撤消、修改数据库的信息的方法。这类信息主要是由提交前的事务动作的记录所组成。Oracle 将这些记录总称为 undo。

撤消记录用于：

- 当发布 ROLLBACK 语句时回滚事务
- 恢复数据库
- 提供读一致性

当发布一条回滚语句时，撤消记录用于撤消未提交事务对数据库所作的修改。在数据库恢复期间，撤消记录用于撤消从重做日志作用到数据文件的任何未提交的修改。撤消记录通过当另一用户正在修改数据时，为同时正在访问数据的用户维护数据修改前的映像提供读一致性。

过去，Oracle 使用回滚段来存储撤消。这些回滚段的空间管理已经被证实为非常复杂。Oracle 现在提供另一种存储撤消的方法，消除了管理回滚段空间的复杂性，并且能使 DBA 对撤消在被覆盖之前能保留的时间进行控制。该方法使用了撤消表空间。管理撤消空间的这两种方法在本章中都将要进行讨论。

不能在同一个数据库实例中同时使用这两种方法，尽管为了移植的目的，这是可以的，例如，在一个使用回滚段的数据库中创建撤消表空间，或者在一个使用撤消表空间的数据库中撤消回滚段。但是，为了使切换到另一种管理撤消的方法有效，必须关闭和重新启动数据库。

注意：Oracle 通常使用一个 SYSTEM 回滚段来执行系统事务。一个数据库中只有一个 SYSTEM 回滚段，它是在 CREATE DATABASE 时自动创建的，并且通常在实例启动时联机。在管理 SYSTEM 回滚段时，不要求执行任何选项。

13.2 指定管理撤消空间的方式

如果使用回滚段方法来管理撤消空间，被认为是以手动撤消管理方式在操作。如果使用撤消表空间方法，则是以自动撤消管理方式在操作。在实例启动时，使用初始化参数 UNDO_MANAGEMENT 决定采用的方式。

13.2.1 以自动撤消管理方式启动实例

如下的初始化参数设置致使 STARTUP 命令以自动撤消管理方式来启动一个实例：

```
UNDO_MANAGEMENT = AUTO
```

一个撤消表空间必须是可用的，Oracle 将在其中存储撤消记录。默认的撤消表空间在创建数据库时被创建，或者可以明确地创建一个撤消表空间。创建一个撤消表空间的方法在本章的“创建撤消表空间”一节中说明。

当实例启动时，Oracle 自动选择使用第一个可用的撤消表空间。如果没有可用的撤消表空间，实例启动，但使用 SYSTEM 回滚段。在正常情况下，这不是推荐使用的方法，并且一条告警信息写入告警文件，以警告系统正在没有撤消表空间的情况下运行。

可以选择在启动时指定一个 Oracle 实例使用某个特定的撤消表空间。这是通过设置初始化参数 UNDO_TABLESPACE 来完成的。例如：

```
UNDO_TABLESPACE = undotbs_01
```

在这种情形下，如果还没有创建撤消表空间（在本例中，是 undotbs_01），STARTUP 命令会失败。UNDO_TABLESPACE 参数可以用于为一个 Oracle Real Application Clusters 环境中的实例指定一个特定的撤消表空间。

下面是对可以用在自动撤消管理方式中的初始化参数的一个总结：

初始化参数	说明
UNDO_MANAGEMENT	如果为 AUTO，使用自动撤消管理方式。如果为 MANUAL，则使用手动撤消管理方式。
UNDO_TABLESPACE	一个指定所用撤消表空间名称的动态参数。
UNDO_RETENTION	一个指定保持撤消的时间长度的动态参数。默认值是 900 秒。
UNDO_SUPPRESS_ERRORS	如果为 TRUE，当操作以自动撤消管理方式进行时，如果发布手动撤消管理 SQL 语句，则取消出错信息。如果为 FALSE，发布出错信息。它是一个动态参数。

如果初始化参数文件包含与手动撤消管理有关的参数，它们被忽略。

要了解如何管理撤消表空间，参见本章的“管理撤消表空间”一节。

13.2.2 以手动撤消管理方式启动实例

如下的初始化参数设置致使 STARTUP 命令以手动撤消管理方式来启动一个实例：

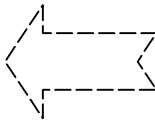
```
UNDO_MANAGEMENT = MANUAL
```

如果没有指定初始化参数 UNDO_MANAGEMENT，实例以手动撤消管理方式启动。如果发现初始化参数 UNDO_TABLESPACE，它会被忽略。对于想要以手动撤消管理方式运行数据库的 DBA，它们现有的初始化参数文件可以不作任何修改就使用。

当实例启动时，它按照下列参数之一所决定的那样，联机生成许多回滚段：

- 初始化参数 ROLLBACK_SEGMENTS
- 初始化参数 TRANSACTIONS 和 TRANSACTIONS_PER_ROLLBACK_SEGMENT

下面是对可以用在手动撤消管理方式中的初始化参数的一个总结：



初始化参数	说明
ROLLBACK_SEGMENTS	指定在启动时需要的回滚段
TRANSACTIONS	指定并发事务的最大个数
TRANSACTIONS_PER_ROLLBACK_SEGMENT	指定每个回滚段希望被处理的并发事务的个数
MAX_ROLLBACK_SEGMENTS	指定可以为任何实例联机服务的回滚段的最大个数

要了解如何管理回滚段，参见本章的“管理回滚段”一节。

13.3 管理撤消表空间

Oracle 强烈推荐以自动撤消管理方式进行操作。数据库服务器能够更加有效地管理撤消，并且自动撤消管理方式执行和管理更简单一些。下面几节指导用户如何管理撤消表空间：

- 创建撤消表空间
- 修改撤消表空间
- 取消撤消表空间
- 切换撤消表空间
- 为撤消空间确定用户限额
- 为撤消信息设置保留期
- 查看关于撤消空间的信息

13.3.1 创建撤消表空间

有两种创建撤消表空间的方法。第一种方法在发布 CREATE DATABASE 语句时，创建撤消表空间。这种情况发生在创建一个新数据库的时候，并且实例以自动撤消管理方式启动（UNDO_MANAGEMENT = AUTO）。第二种方法用在现有的数据库中。它使用 CREATE UNDO TABLESPACE 语句。

不能在撤消表空间中创建数据库对象。它是为系统管理的撤消数据保留的。

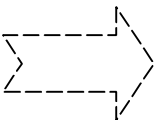
13.3.1.1 使用 CREATE DATABASE 创建撤消表空间

可以使用 CREATE DATABASE 语句的 UNDO TABLESPACE 子句来创建一个特定的撤消表空间。但是，该子句不是必须的。

如果没有指定 UNDO TABLESPACE 子句，并且 CREATE DATABASE 语句是以自动撤消管理方式执行的，将创建一个名为 SYS_UNDOTBS 的默认撤消表空间。该表空间从 CREATE DATABASE 语句使用的默认文件组中分配，并且其属性由 Oracle 决定。初始大小是 10M，并是自动盘区的。这种创建撤消表空间的方法只推荐给那些对撤消空间的分配没有特殊要求的用户使用。

如下语句举例说明了在 CREATE DATABASE 语句中使用 UNDO TABLESPACE 子句。撤消表空间命名为 undotbs_01，并且分配给它一个数据文件，/u01/oracle/rbdb1/undo0101.dbf。

```
CREATE DATABASE rbdb1
```



```
CONTROLFILE REUSE
```

```
...
```

```
UNDO TABLESPACE undotbs_01 DATAFILE '/u01/oracle/rbdb1/undo0101.dbf'
```

如果在 CREATE DATABASE 期间，创建撤消表空间没有成功，整个 CREATE DATABASE 操作就失败。必须清理数据库文件，修正错误并重试 CREATE DATABASE 操作。

13.3.1.2 使用 CREATE UNDO TABLESPACE 语句

CREATE UNDO TABLESPACE 语句与 CREATE TABLESPACE 语句相同，但指定了关键字 UNDO。Oracle 决定撤消表空间的绝大多数属性，用户只能指定 DATAFILE 子句。

下面的例子创建撤消表空间 undotbs_02:

```
CREATE UNDO TABLESPACE undotbs_02
DATAFILE '/u01/oracle/rbdb1/undo0201.dbf' SIZE 2M REUSE AUTOEXTEND ON;
```

13.3.2 修改撤消表空间

使用 ALTER TABLESPACE 语句可以修改撤消表空间。但是，既然撤消表空间的大多数方面都由系统来管理，所以，用户只需关注下列动作：

- 添加数据文件
- 数据文件的重命名
- 使数据文件联机或使其脱机
- 开始或结束数据文件的公开备份

也只有这些属性允许用户进行修改。

如果一个撤消表空间用完其空间，或者用户想要防止它这样做，可以为它添加更多的文件或是为现有的数据文件重新设置其大小。

下面的例子为撤消表空间 undotbs_01 添加了另一个数据文件：

```
ALTER TABLESPACE undotbs_01
ADD DATAFILE '/u01/oracle/rbdb1/undo0102.dbf' AUTOEXTEND ON NEXT 1M
MAXSIZE UNLIMITED;
```

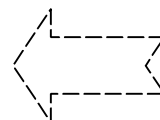
可以使用 ALTER DATABASE ... DATAFILE 语句为数据文件重新设置大小或盘区。

13.3.3 取消撤消表空间

使用 DROP TABLESPACE 语句可以取消一个撤消表空间。下面的例子取消了撤消表空间 undotbs_01:

```
DROP TABLESPACE undotbs_01;
```

只有一个撤消表空间当前没有被任何实例所使用，才可以被取消。如果撤消表空间包含任何未完成的事务（例如，一个事务死掉而且仍未恢复），DROP TABLESPACE 语句就失败。但是，既然 DROP TABLESPACE 取消一个撤消表空间即使它包含了未到期的撤消信息（在保留期内），用户必须小心，如果撤消信息是一些现有查询所需要的，就不要取消该撤消表空间。



DROP TABLESPACE 对撤消表空间的作用类似于 DROP TABLESPACE ... INCLUDING CONTENTS。撤消表空间的所有内容都被删除。

13.3.4 切换撤消表空间

可以从使用一个撤消表空间切换到另一个。因为初始化参数 UNDO_TABLESPACE 是一个动态参数，ALTER SYSTEM SET 语句可用于指定一个新的撤消表空间。

下面的语句有效地切换到一个新的撤消表空间：

```
ALTER SYSTEM SET UNDO_TABLESPACE = undotbs_02;
```

假定 undotbs_01 是当前的撤消表空间，在该命令成功执行后，该实例使用 undotbs_02 代替 undotbs_01 作为它的撤消表空间。

如果被切换到的表空间有下列任何一个条件存在，将会报告一个错误并且不会发生切换：

- 该表空间不存在，
- 该表空间不是一个撤消表空间
- 该表空间已经被另一个实例所使用

当切换操作执行时，数据库是联机的，而用户事务可以在该命令执行时执行。当切换操作成功完成时，在切换操作开始后启动的所有事务被指定给新的撤消表空间中的事务表。

切换操作并不等待原先的撤消表空间中的事务进行提交。如果在原先的撤消表空间中有挂起的事务，该撤消表空间进入一种 PENDING OFFLINE 模式（状态）。在这种模式中，现有的事务可以继续执行，但新的用户事务的撤消记录不能存储在该撤消表空间中。

一个撤消表空间能够以这种 PENDING OFFLINE 模式存在，即使在切换操作成功完成之后。一个 PENDING OFFLINE 撤消表空间不能被另一个实例所用，也不能被取消。最终，在所有活动事务提交完以后，该撤消表空间自动从 PENDING OFFLINE 模式进入 OFFLINE 模式。从那时起，该撤消表空间可以被其他实例使用（在 Oracle Real Application Cluster 环境中）。

如果 UNDO_TABLESPACE 的参数值被设置为 ‘ ’（两个单引号），当前撤消表空间将被切换出去，而没有任何其他的撤消表空间切换进来。这可能是有用的，例如，在想要还原到手动撤消管理方式的情况下，不指定撤消表空间。

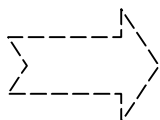
下面的例子不指定当前的撤消表空间：

```
ALTER SYSTEM SET UNDO_TABLESPACE = '';
```

13.3.5 为撤消空间确定用户限额

Oracle 的 Database Resource Manager（数据库资源管理器）可以用于为撤消空间建立用户限额。Database Resource Manager 指令，UNDO_POOL 允许 DBA 限制一个用户组（资源使用者组）所消耗的撤消空间的数量。

可以为每个使用者组指定一个撤消池。一个撤消池控制由一个使用者组产生的全部撤消的数量。当一个使用者组产生的总的撤消超出它的撤消限制时，产生重做的当前 UPDATE 事务被终止。该使用者组的其他成员不能执行进一步的更新，直到撤消空间从撤消池中被



释放出来。

当没有明确定义 UNDO_POOL 指令时，用户被允许使用没有限制的撤消空间。

13.3.6 为撤消信息设置保留期

当撤消空间被一个新的事务重写时，它所提交的撤消信息通常被丢掉。然而，为了一致性读取的目的，长时间运行的查询可能需要原先的撤消信息来撤消修改和生成数据块原先的映像。初始化参数 UNDO_RETENTION，提供一种明确指定要保留的撤消信息的数量的方法。利用一个适当的设置，长时间运行的查询能够完成，而不会收到“snapshot too old”的出错信息。

13.3.6.1 指定保留期

保留期以秒为单位来指定，例如 500 秒。它可以持续计算并不受系统崩溃的影响。这是指，实例崩溃前生成的撤消，一直保持到它的保留期过期，即使重新启动实例也是如此。当该实例恢复时，撤消信息会基于初始化参数 UNDO_RETENTION 的当前设置进行保留。

参数 UNDO_RETENTION 可以在进程 STARTUP 使用的初始化参数文件中初始化指定。

```
UNDO_RETENTION = 10
```

UNDO_RETENTION 参数值也可以使用 ALTER SYSTEM 命令在任何时候动态地修改。

```
ALTER SYSTEM SET UNDO_RETENTION = 5
```

参数 UNDO_RETENTION 立即有效，但只有在活动事务的当前撤消表空间有足够的空间时，它才起作用。如果一个活动事务需要撤消空间而该撤消表空间没有可用的空间，系统开始重用未到期的撤消空间。这类动作可能造成一些查询因“snapshot too old”错误而失败。

如果没有指定初始化参数 UNDO_RETENTION，默认值为 900 秒。

13.3.6.2 闪回查询的保留期

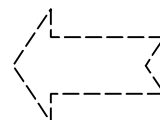
撤消信息的保留期是闪回查询执行的一个重要因素。Oracle 的闪回查询特性，使用户能够见到数据库在过去一个指定时间的一致的版本。用户可以在数据库中的一个过去的时间执行查询，甚至应用程序。Oracle 提供的 DBMS_FLASHBACK 包实现了这个功能。

保留期决定了为闪回查询建立的数据库版本可以回退多长时间。特别要注意的是，必须确定一个足够长的撤消保留间隔，它使用户可以建立其感兴趣的数据库的最古老版本的快照。例如，如果一个应用程序需要一个可用来反映其 12 小时以前内容的数据库版本，那么参数 UNDO_RETENTION 必须被设置为 43200。

13.3.6.3 撤消保留的空间要求

给定了一个特定的 UNDO_RETENTION 参数设置和一些系统统计信息，可以利用如下公式计算，满足撤消保留要求所需要的撤消空间数量：

$$\text{UndoSpace} = \text{UR} * \text{UPS} + \text{overhead}$$



其中：

UndoSpace = 撤消块数目

UR = 以秒计的 UNDO_RETENTION

UPS = 每秒的撤消块数

Overhead = 元数据的小开销（事务表、位图等等）

作为一个例子，如果 UNDO_RETENTION 被设置为 2 小时，并且事务速率（UPS）为每小时 200 个撤消块，使用 4K 块大小，所需要的撤消空间计算如下：

$$(2 * 3600 * 200 * 4K) = 5.8GBs.$$

这类计算可以使用 V\$UNDOSTAT 视图中的信息来完成。在稳定状态中，可以查询该视图来获得事务速率。开销的数字也可以从该视图得到。

13.3.7 查看关于撤消空间的信息

本节列出有助于以自动撤消管理方式查看关于撤消空间的信息的视图。除了这里列出的视图外，用户可以从用来查看表空间和数据文件信息的可用视图中获得信息。

13.3.7.1 撤消空间视图

下列视图可用来获得撤消空间的信息：

视图	说明
V\$UNDOSTAT	包括用来监控和调整撤消空间的统计信息。使用该视图有助于计算当前工作量所需要的撤消空间数量。Oracle 也使用这些信息来帮助调整系统中撤消的使用。该视图在自动撤消管理方式和手动撤消管理方式中都可使用。
V\$ROLLSTAT	用于自动撤消管理方式，信息反映了撤消表空间中撤消段的行为。
V\$TRANSACTION	包括撤消段的信息
DBA_UNDO_EXTENTS	显示撤消表空间中每个盘区的提交时间

13.3.7.2 监控撤消空间

V\$UNDOSTAT 视图有助于监控当前实例中在撤消空间上事务执行的效果。统计信息可用于撤消空间的消耗、事务并发和实例中查询的长度。

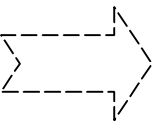
视图中每一行含有实例中以 10 分钟的间隔收集的统计信息。这些行按 BEGIN_TIME 列值降序排列。每行属于 (BEGIN_TIME, END_TIME) 所标明的时间间隔。每列表示在该时间间隔内为特定的统计信息所收集的数据。该视图的第一行包括当前（部分）时间期间的统计信息。该视图总共包括 144 行，跨越一个 24 小时的周期。

下面的例子显示了一个 V\$UNDOSTAT 视图的查询结果。

```
SELECT BEGIN_TIME, END_TIME, UNDOTSN, UNDOBLKS, TXNCOUNT,
MAXCONCURRENCY AS "MAXCON"
FROM V$UNDOSTAT;
```

结果是：

```
BEGIN_TIME END_TIME UNDOTSN UNDOBLKS TXNCOUNT MAXCON
```



```
07/28/2000 18:26:28 07/28/2000 18:32:13 2 709 55 2
07/28/2000 18:16:28 07/28/2000 18:26:28 2 448 12 2
07/28/2000 14:36:28 07/28/2000 18:16:28 1 0 0 0
07/28/2000 14:26:28 07/28/2000 14:36:28 1 1 1 1
07/28/2000 14:16:28 07/28/2000 14:26:28 1 10 1 1
...
```

上面的例子显示了从时间 18: 32: 13 算起，以前的 24 小时中系统的撤消空间是如何消耗的。

13.4 管理回滚段

如果选择使用回滚段来存储撤消，下面的章节对它们的管理进行指导：

- 管理回滚段的准则
- 创建回滚段
- 修改回滚段
- 将事务明确地指定到回滚段
- 取消回滚段
- 查看回滚段信息

注意：使用回滚段来管理撤消空间是不被赞成的。Oracle 强烈推荐使用自动撤消管理方式和使用 UNDO_TABLESPACE 管理撤消空间。

13.4.1 管理回滚段的准则

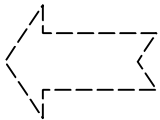
本节说明在数据库中创建或管理回滚段之前，应该考虑的准则，并包括如下内容：

- 使用多回滚段
- 在公共和私有回滚段之间做出选择
- 指定回滚段为自动获得
- 近似的回滚段大小
- 创建具有多个相同大小盘区的回滚段
- 为每个回滚段设置一个最佳的盘区数
- 将回滚段放入一个独立的表空间

13.4.1.1 使用多回滚段

使用多回滚段将回滚段争用分布到多个段并提高系统性能。Oracle 以 round-robin 方式为回滚段指定事务。其结果是将事务相当均匀地分布给每个回滚段。也有可能将一个事务指定给特定的回滚段，但通常不这样做。

当数据库被创建时，一个单一的名为 SYSTEM 的回滚段被创建在 SYSTEM 表空间中。该回滚段被 Oracle 数据库服务器以特殊的方法使用，并且不用于一般的用途。在向 non-SYSTEM 表空间中创建的对象写之前，必须在 non-SYSTEM 表空间中创建和联机生成至少一个额外的回滚段。



注意: 当初始化创建数据库时, 并且为了创建额外的表空间和回滚段, 必须在 SYSTEM 表空间中创建第二个回滚段。一旦这些额外的回滚段被创建, 应该激活这些新的回滚段并使第二个回滚段不可用。

在启动时, 一个实例除了获得它所需要的或直接获得的任何其他回滚段之外, 通常还获得 (联机生成) SYSTEM 回滚段。当有多回滚段时, Oracle 试图将 SYSTEM 回滚段只用于特殊的系统事务并将用户事务分布到其他的回滚段。如果对于 non-SYSTEM 回滚段有太多的事务, Oracle 使用 SYSTEM 段; 计划好回滚段的数目以避免这种情况。

当启动一个实例时, 有两个选项来激活多回滚段。

- 使用公共回滚段, 并且在初始化参数文件中包括初始化参数 TRANSACTIONS 和 TRANSACTIONS_PER_ROLLBACK_SEGMENT。
- 使用私有或公共回滚段, 并且以初始化参数 ROLLBACK_SEGMENTS 指定它们的名称。

这些选项在接下来的其他准则中讨论。

对同时打开的回滚段的数目有个限制。该限制由初始化参数 MAX_ROLLBACK_SEGMENTS 设定。确保该参数被设置为一个比在初始化参数 ROLLBACK_SEGMENTS 中指定的回滚段数目更大的数值。

13.4.1.2 在公共和私有回滚段之间做出选择

一个私有回滚段必须由一个实例明确地获得。当该回滚段的名称包含在初始化参数文件里的参数 ROLLBACK_SEGMENTS 中时, 这种情况可能在数据库启动时发生。一个私有回滚段也可以专门通过手工发布使其联机生成的语句来获得。在一个 Oracle Real Application Clusters 环境中, 私有回滚段允许一个实例获得特定的回滚段。

公共回滚段形成一个回滚段池, 可以被任何需要回滚段的实例所使用。一个实例根据初始化参数 TRANSACTIONS 和 TRANSACTIONS_PER_ROLLBACK_SEGMENT 的值, 决定在实例启动时自动获得多少个这样的回滚段。公共回滚段可以在 Oracle Real Application Cluster 实例之间共享。

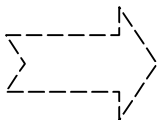
如果没有使用 Oracle9i Real Application Clusters 特性, 私有回滚段和公共回滚段功能相似。

13.4.1.3 指定回滚段为自动获得

当多个事务并发进行时, 它们同时产生回滚信息。在实例启动时, 一种指定自动获得的合适的回滚段数的方法是包含初始化参数 TRANSACTIONS 和 TRANSACTIONS_PER_ROLLBACK_SEGMENT。用户必须也正在使用公共回滚段。

可以使用初始化参数 TRANSACTIONS 来指明用户为实例所预期的并发事务的数目, 并使用初始化参数 TRANSACTIONS_PER_ROLLBACK_SEGMENT 指明用户期望每个回滚段将需要处理的事务的数目。然后, 当一个实例打开数据库时, 它试图获得至少 n 个回滚段, 这里 $n = \text{TRANSACTIONS} / \text{TRANSACTIONS_PER_ROLLBACK_SEGMENT}$ 。当创建数据库时, 或者随后, 用户应该已经创建了至少 n 个公共回滚段。

如果用户选择使用私有回滚段, 这些回滚段将由实例在启动时自动获得, 前提是用户在该



实例的参数文件中的初始化参数 `ROLLBACK_SEGMENTS` 中指定了这些回滚段的名称。

如果用户同时使用私有回滚段和公共回滚段，可能发生下面的情况。一个实例获得所有在初始化参数 `ROLLBACK_SEGMENTS` 中列出的回滚段，即使比 `TRANSACTIONS / TRANSACTIONS_PER_ROLLBACK_SEGMENT` 指定的段还要多。

13.4.1.4 近似的回滚段大小

总的回滚段大小应该根据相对于一个数据库所发布的最普通的事务的大小来设置。概括地说，当数据库拥有很多较小的回滚段时，简短的事务会有较好的性能，然而，长时间运行的事务，象批处理作业，使用较大的回滚段性能更好。通常，回滚段能够顺利地处理任何大小的事务。但是，在当一个事务非常简短或者时间十分长的极端情况下，一个用户可能想要使用一个合适大小的回滚段。

如果一个系统只运行简短的事务，回滚段应该小以便它们总能在主内存中缓存。如果回滚段足够小，它们越可能按照 LRU 算法缓存在 SGA 中，并且数据库性能因磁盘必需的 I/O 较少而提高。小回滚段的主要不足是当运行一个包括被其他事务频繁更新的记录的长时间查询时，不断增加的出现“snapshot too old”错误的可能性。这种错误发生是因为读一致性所需要的回滚段入口被回滚段四周的其他更新入口所重写。当设计一个应用事务时，考虑这种情况，并使它们作为工作的简短基本单位，以便能够避免这个问题。

相反地，长时间运行的事务使用较大的回滚段工作得更好，因为长时间运行事务的回滚入口能够适合一个大回滚段预先分配的盘区。

当数据库系统应用程序同时发布非常简短和时间十分长的混合事务时，如果根据事务/回滚段大小明确地将事务指定给一个回滚段，可以使性能得到优化。用户可以最小化动态盘区分配和切断回滚段。这不是大多数系统需要的，它适用于特别大或特别小的任务。

当发布特别小和特别大的混合事务时，为优化性能，针对每种类型的事务，安排一定数目的大小合适的回滚段（比如小的、中等的和大的）。大多数回滚段应该与典型事务相对应，少数回滚段对应非典型事务。然后，为每个这样的回滚段设置 `OPTIMAL`，以便如果回滚段不得不增长，可以返回它所计划的大小。

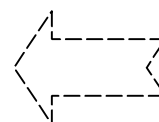
应该通知用户关于与不同类型事务相应的回滚段的不同设置。通常，明确地为特定的回滚段指定一个事务并没有好处。但是，可以将一个非典型事务指定给一个为这类事务创建的合适的回滚段。例如，可以将一个包含大的批处理作业的事务指定给一个大的回滚段。

当一个混合事务不是很普遍时，每个回滚段应该是数据库中最大的表大小的 10%，因为大多数 SQL 语句影响一个表的 10% 或更少。一个这种大小的回滚段应该足够存储大多数 SQL 语句执行的动作。

一般地说，应该为回滚段设置一个高的 `MAXEXTENTS`。这允许一个回滚段在它需要时，分配后来的盘区。

13.4.1.5 创建使用多个相同大小的盘区的回滚段

每个回滚段总共分配的空间应该分成多个相同大小的盘区。通常，如果一个实例的每个回滚段有 10 至 20 个相同大小的盘区，将观察到最优的回滚 I/O 性能。



在决定一个回滚段想要的总的初始大小和该段的初始盘区数之后，使用下面的公式来计算该回滚段的每个盘区的大小：

$$s = T / n$$

其中：

s = 计算出的每个盘区初始分配的大小，以字节计

T = 回滚段总的初始大小，以字节计

n = 初始分配的盘区数

计算出 s 以后，创建回滚段并将存储参数 INITIAL 和 NEXT 指定为 s，将 MINEXTENTS 指定为 n。不能为回滚段指定 PCTINCREASE，因此默认为 0。同样，如果一个盘区的大小 s 正好不是数据块大小的整数倍，那么它将上舍入到相邻的倍数。

13.4.1.6 为每个回滚段设置一个最佳的盘区数

当为每个回滚段设置 OPTIMAL 参数时，应该仔细地确定系统运行的事务的类型。对于一个频繁地执行长时间运行事务的系统，OPTIMAL 应该大一点，这样 Oracle 不必频繁地缩小和分配盘区。同样，对于一个执行长时间查询活动数据的系统，OPTIMAL 应该大一点以避免 “snapshot too old” 错误。对于主要执行简短事务和查询的系统，OPTIMAL 应该小一点，这样回滚段保持足够小，能够缓存在内存中，从而提高系统性能。

可以监控 V\$ROLLNAME 和 V\$ROLLSTAT 动态性能视图以收集用于决定 OPTIMAL 的合适设置的有用的统计信息。

13.4.1.7 将回滚段放入一个独立的表空间

如果可能，专门创建一个或更多的表空间来保存所有的回滚段。这种方法，所有的回滚段数据与其他类型的数据分开存储。创建这种 “回滚段” 表空间可以提供如下好处：

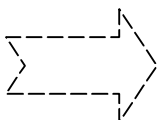
- 一个保存回滚段的表空间可以一直保持联机，从而一直最大化回滚段的混合存储容量。如果一些回滚段不可用，整个数据库操作都受到影响。
- 因为拥有活动回滚段的表空间不能脱机，指定一个保存数据库的所有回滚段的表空间确保存储在其他表空间中的数据可以脱机，而不涉及到数据库的回滚段。
- 如果表空间包含频繁地分配和回收盘区的回滚段，这个表空间的空闲盘区可能有更多的碎片。

13.4.2 创建回滚段

要创建回滚段，用户必须有 CREATE ROLLBACK SEGMENT 系统权限。用户使用 CREATE ROLLBACK SEGMENT 语句。要包含新的回滚段的表空间必须是联机的。回滚段的创建通常作为数据库创建脚本或进程的一部分，但用户可以在这之后添加更多的回滚段。

下面是本节中包含的与创建回滚段有关的内容：

- CREATE ROLLBACK SEGMENT 语句
- 使新的回滚段联机
- 在创建回滚段时设置存储参数



13.4.2.1 CREATE ROLLBACK SEGMENT 语句

下面的语句在表空间 `rbsspace` 中创建一个名为 `rbs_02` 的回滚段，使用该表空间的默认存储参数。由于这不是一个 Oracle Real Application Clusters 环境，没有必要指定 `PRIVATE` 或 `PUBLIC`。默认为 `PRIVATE`。

```
CREATE ROLLBACK SEGMENT rbs_02 TABLESPACE rbsspace;
```

13.4.2.2 使新的回滚段联机

新的回滚段初始时是脱机的。必须发布一条 `ALTER ROLLBACK SEGMENT` 语句使其联机并且使它们能够为一个实例的事务所使用。这将在本章的“改变回滚段的 `ONLINE/OFFLINE` 状态”一节中说明。

如果创建一个私有回滚段，将这个新的回滚段的名称添加到该数据库初始化参数文件的初始化参数 `ROLLBACK_SEGMENTS` 中。这样做使得该私有回滚段可由实例在启动时自动获得。例如，如果创建两个新的私有回滚段并命名为 `rbs_01` 和 `rbs_02`，那么初始化参数 `ROLLBACK_SEGMENTS` 可以如下指定：

```
ROLLBACK_SEGMENTS = (rbs_01, rbs_02)
```

13.4.2.3 在创建回滚段时设置存储参数

假定想要使用如下的存储参数和最佳大小设置创建一个回滚段 `rbs_01`：

- 该回滚段被分配一个 100K 的初始盘区。
- 该回滚段被分配第二个 100K 的盘区。
- 该回滚段的最佳大小为 4M。
- 该段创建时的最小盘区数和初始分配的盘区数为 20。
- 该回滚段可以分配的最大盘区数为 100，包括初始的盘区。

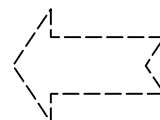
下面的语句创建一个带有这些特征的回滚段：

```
CREATE ROLLBACK SEGMENT rbs_01
TABLESPACE rbsspace
STORAGE (
INITIAL 100K
NEXT 100K
OPTIMAL 4M
MINEXTENTS 20
MAXEXTENTS 100 );
```

不能为存储参数 `PCTINCREASE` 设置一个值。对于回滚段它总是为 0。存储参数 `OPTIMAL` 对回滚段是唯一的。对于存储参数的讨论见第 14 章的“设置存储参数”一节。

Oracle 公司做出如下建议：

- 将 `INITIAL` 和 `NEXT` 设置为相同的值，以确保所有的盘区都是相同大小。
- 创建一个较多的初始盘区，将动态盘区的可能性降至最低。`MINEXTENTS = 20` 是个不错的值。



- 避免设置 `MAXEXTENTS = UNLIMITED`，因为这可能造成因程序错误引起的回滚段和可能的数据文件的不必要盘区。如果指定为 `UNLIMITED`，应意识到该回滚段的盘区必须最少有 4 个数据块。同样，如果以后想要转换一个 `MAXEXTENTS` 限制为 `UNLIMITED` 的回滚段，而在任一盘区中少于 4 个数据块，将不能被转换。如果想要从有限的转换为 `UNLIMITED`，并且在一个盘区中少于 4 个数据块，唯一的选择就是取消并重建该回滚段。

13.4.3 修改回滚段

本节讨论维护回滚段可以采取的各种动作。所有这些维护行为都使用 `ALTER ROLLBACK SEGMENT` 语句。用户必须拥有 `ALTER ROLLBACK SEGMENT` 系统权限来使用该语句。

对下面的内容进行了讨论：

- 改变回滚段的存储参数
- 手动缩小一个回滚段
- 改变回滚段的 `ONLINE/OFFLINE` 状态

13.4.3.1 改变回滚段的存储参数

在创建回滚段之后，可以改变回滚段的一些存储参数。用户可能想要改变 `OPTIMAL` 或 `MAXEXTENTS` 的值。如下的语句修改回滚段 `rbs_01` 能够分配的最大盘区数：

```
ALTER ROLLBACK SEGMENT rbs_01  
STORAGE (MAXEXTENTS 120);
```

可以修改 `SYSTEM` 回滚段的设置，包括参数 `OPTIMAL`，与修改任何回滚段的那些参数一样。

13.4.3.2 手动缩小一个回滚段

可以使用 `ALTER ROLLBACK SEGMENT` 语句手动减小回滚段的大小。想要缩小的回滚段必须是联机的。

如下语句将回滚段 `rbs1` 缩小至 100K：

```
ALTER ROLLBACK SEGMENT rbs1 SHRINK TO 100K;
```

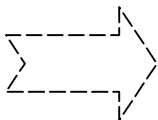
该语句试图将回滚段的大小减小到指定的大小，但是如果一个盘区因为处于活动状态而不能被重新分配，就很快停止了。

13.4.3.3 改变回滚段的 `ONLINE/OFFLINE` 状态

本节讨论使回滚段联机和脱机方面的内容，包括以下内容：

- 手动使回滚段联机
- 自动使回滚段联机
- 使回滚段脱机

一个回滚段要么联机并可为事务所用，要么脱机不可为事务所用。通常，回滚段联机并可为事务所用。



在下列情形中，用户可能想要使回滚段脱机：

- 想要使一个表空间脱机，并且该表空间包含回滚段。如果表空间包含事务正在使用的回滚段，将不能使其脱机。要防止相关联的回滚段正被使用，可以在使表空间脱机之前使它们脱机。
- 想要取消一个回滚段，但因为事务正在使用它而不能取消。要防止回滚段正被使用，可以在取消它之前使它脱机。

注意：不能使 SYSTEM 回滚段脱机。

用户可能在以后又想要使一个脱机的回滚段重新联机，以便事务可以使用它。当创建一个回滚段时，它最初是脱机的，并且必须在它被一个实例的事务使用之前，明确地使一个新建的回滚段联机。可以使用任何访问包含该回滚段的数据库的实例来使一个脱机的回滚段联机。

13.4.3.4 手动使回滚段联机

如果一个回滚段的当前状态（正如在 DBA_ROLLBACK_SEGS 数据字典视图中显示的那样）为 OFFLINE 或 PARTLY AVAILABLE，只能使它联机。要使一个脱机的回滚段联机，使用带有 ONLINE 选项的 ALTER ROLLBACK SEGMENT 语句。

如下的语句使回滚段 user_rs_2 联机：

```
ALTER ROLLBACK SEGMENT user_rs_2 ONLINE;
```

在使一个回滚段联机之后，它在数据字典视图 DBA_ROLLBACK_SEGS 中的状态是 ONLINE。要了解一个检查回滚段状态的查询，见本章的“显示回滚段信息”一节

一个处于 PARTLY AVAILABLE 状态的回滚段包含一个有问题的或恢复的分布式事务的数据，或者是仍在恢复的事务的数据。可以在数据字典视图 DBA_ROLLBACK_SEGS 查看到它的状态为 PARTLY AVAILABLE。回滚段通常保持在这种状态，直到该事务被 RECO 自动解决，或者被 DBA 手动解决。

用户可能会发现所有的回滚段都是 PARTLY AVAILABLE。在这种情况下，可以使这些 PARTLY AVAILABLE 的段联机。一些被有问题的事务的回滚段所使用的资源仍保持不能被访问，直到该事务被解决。其结果是，如果其他指定到该回滚段的事务需要额外的空间，该回滚段不得不增长。

作为使 PARTLY AVAILABLE 的段联机的另一种方法，可能会发现临时创建一个新的回滚段要有效得多，直到这个有问题的事务被解决。

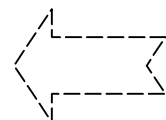
13.4.3.5 自动使回滚段联机

如果想要一个回滚段在启动数据库时，自动使其联机，将该段的名称添加到数据库参数文件的参数 ROLLBACK_SEGMENTS 中。或者，可以使用公共回滚段，并且使用初始化参数 TRANSACTIONS 和 TRANSACTIONS_PER_ROLLBACK_SEGMENT。

这些选项在本章的“指定回滚段为自动获得”一节中讨论。

13.4.3.6 使回滚段脱机

要使一个联机的回滚段脱机，使用带有 OFFLINE 选项的 ALTER ROLLBACK



SEGMENT 语句。该回滚段的状态在 DBA_ROLLBACK_SEGS 数据字典视图必须是 ONLINE，并且该回滚段必须是由当前实例所获得。

下面的例子使回滚段 user_rs_2 脱机：

```
ALTER ROLLBACK SEGMENT user_rs_2 OFFLINE;
```

如果试图使一个没有包含活动的回滚入口的回滚段脱机，Oracle 立即使该段脱机并将它的状态改为 OFFLINE。

相反地，如果试图使一个包含活动事务（本地的、远程的或者分布式的）的回滚数据的回滚段脱机，Oracle 使该回滚段对将来的事务为不可用，并且在所有使用该回滚段的的活动事务完成后使其脱机。直到该事务完成，除正在试图使它脱机以外的任何其他实例都不能使该回滚段联机。

在回滚段等待脱机期间，该回滚段在视图 DBA_ROLLBACK_SEGS 中的状态保持为 ONLINE。然而，该回滚段在视图 V\$ROLLSTAT 中的状态为 PENDING OFFLINE。

试图使一个回滚段脱机并将其状态改变为 PENDING OFFLINE 的实例，可以在任何时候使其重新联机。如果该回滚段重新联机，它运行正常。

在使一个公共回滚段或私有回滚段脱机之后，它保持脱机状态，直到明确地使它重新联机或重新启动该实例。

13.4.4 将事务明确地指定到回滚段

一项事务可以明确地指定到一个特定的回滚段。这样做的原因包括：

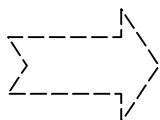
- 可以预计一项事务所生成的回滚信息的数量。当知道回滚信息适合该段的当前盘区时，可以将该事务指定给一个回滚段。因而，可以减少动态分配额外盘区并在随后截断的开销。
- 用户知道没有长时间运行的查询在同时读取相同的表，因此如果将这些较小的事务指定给较小的回滚段，这些段将很有可能保留在内存中。
- 有一些修改表的事务，而这些表正同时被长时间运行的查询读取。可以将这些事务指定给较大的回滚段，以便要求读一致性的查询所需要的回滚信息不会被重写。

要将一项事务明确地指定给一个回滚段，使用带有 USE ROLLBACK SEGMENT 子句的 SET TRANSACTION 语句。该回滚段对当前实例必须是联机的，并且 SET TRANSACTION USE ROLLBACK SEGMENT 语句必须是该事务的第一条语句。如果一个指定的回滚段没有联机，或者 SET TRANSACTION USE ROLLBACK SEGMENT 子句不是一项事务的第一条语句，将会返回一个错误。

例如，如果正打算开始一项包含大量工作的事务（比大多数事务要多），可以将该事务指定给一个较大的回滚段，如下：

```
SET TRANSACTION USE ROLLBACK SEGMENT large_rs1;
```

该事务提交之后，Oracle 自动将下一项事务指定给任何可用的回滚段，除非这项新的事务由用户明确地指定给了一个特定的回滚段。



13.4.5 取消回滚段

当一个段的盘区在磁盘上成为太多碎片，或者该段需要在一个不同的表空间中重新定位时，可以取消回滚段。在取消回滚段之前，确定该回滚段的状态是 OFFLINE。如果需要取消的回滚段是其他任何一种状态，都不能取消它。如果其状态是 INVALID，该段已经被取消了。

要取消一个回滚段，使用 DROP ROLLBACK SEGMENT 语句。用户必须拥有 DROP ROLLBACK SEGMENT 系统权限。如下的语句取消回滚段 rbs1：

```
DROP ROLLBACK SEGMENT rbs1;
```

注意：如果一个在 ROLLBACK_SEGMENTS 中指定的回滚段被取消，一定要编辑数据库的参数文件，从参数 ROLLBACK_SEGMENTS 中的列表里删除取消的回滚段的名称。如果这个步骤没有在下一个实例启动前执行，启动因为不能获得取消的回滚段而失败。

在一个回滚段被取消之后，它的状态改变为 INVALID。下一次创建一个回滚段，它占据由取消的回滚段空出的行（如果该行可用的话）并且取消的回滚段的行不再出现在 DBA_ROLLBACK_SEGS 视图中。

13.4.6 查看回滚段信息

本节介绍可用于获得和监控回滚段信息的视图，并且提供与它们的使用有关的信息和例子。

包括下列内容：

- 回滚段视图
- 显示回滚段信息
- 监控回滚段统计信息
- 显示所有的回滚段
- 显示一个回滚段是否已经脱机

13.4.6.1 回滚段视图

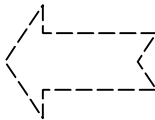
下列视图有助于显示关于回滚段的信息：

视图	说明
DBA_ROLLBACK_SEGS	描述回滚段，包括名称和表空间
DBA_SEGMENTS	将一个段标识为回滚段，并且包括额外的段信息
V\$ROLLNAME	列出所有联机的回滚段的名称
V\$ROLLSTAT	包括回滚段统计信息
V\$TRANSACTION	包括撤消段信息

13.4.6.2 显示回滚段信息

DBA_ROLLBACK_SEGS 数据字典视图存储关于一个数据库回滚段的信息。例如，如下的查询列出一个数据库中每个回滚段的名称、相关联的表空间和状态：

```
SELECT SEGMENT_NAME, TABLESPACE_NAME, STATUS
FROM DBA_ROLLBACK_SEGS;
```



SEGMENT_NAME TABLESPACE_NAME STATUS

SYSTEM SYSTEM ONLINE

PUBLIC_RS SYSTEM ONLINE

USERS_RS USERS ONLINE

另外，下列数据字典视图包含关于一个数据库的段的信息，包括回滚段：

- USER_SEGMENTS
- DBA_SEGMENTS

13.4.6.3 监控回滚段统计信息

V\$ROLLSTAT 动态性能视图可以被查询来监控回滚段的统计信息。它必须被加入到 V\$ROLLNAME 视图中，将它的段号映射到它的名称。

在 V\$ROLLSTAT 视图中，一些重要的特定列包括：

名称	说明
USN	回滚段编号。如果该视图加入到 V\$ROLLNAME 视图中，也确定了该回滚段的名称
WRITES	写入该回滚段的入口字节数。
XACTS	活动的事务数。
GETS	回滚段头部的请求数。
WAITS	导致等待的回滚段头部请求数。
OPTSIZE	该回滚段最佳参数的值。
HWMSIZE	该回滚段大小在使用期间达到的最高值（高水位标记），以字节计。
SHRINKS	为了保持在最佳大小，该回滚段必须执行的缩减数。
WRAPS	一个回滚段入口从一个盘区换到另一个的次数。
EXTENDS	该回滚段必须获得新盘区的次数。
AVESHRINK	在一次压缩期间所释放的平均字节数。
AVEACTIVE	该回滚段中活动盘区的平均字节数，超时测算。

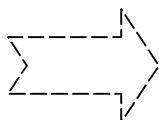
这些统计信息在系统启动时被重置。

该视图的 Ad hoc 查询有助于确定参数 OPTIMAL 的最佳设置。假定一个实例有与盘区同等大小的大小相同的回滚段，一个给定的回滚段 OPTIMAL 值应该设置得比 AVEACTIVE 稍高一点。下面的图表提供了在该视图中如何解释给出的统计数字的附加信息。

SHRINKS	AVESHRINK	分析和建议
低	低	如果 AVEACTIVE 值与 OPTSIZE 值接近，那么该 OPTIMAL 设置是正确的。否则，OPTIMAL 值就太大了（没有这么多压缩在执行）。
低	高	最好的：一个不错的 OPTIMAL 设置。
高	低	OPTIMAL 值太小：太多的压缩在执行。
高	高	定期长时间运行的事务可能生成这些统计信息。将 OPTIMAL 参数设得更高，直到 SHRINKS 值为低。

13.4.6.4 显示所有的回滚段

如下的查询返回每个回滚段的名称、包含它的表空间和它的大小：



```

SELECT SEGMENT_NAME, TABLESPACE_NAME, BYTES, BLOCKS, EXTENTS
FROM 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.

```

13.4.6.5 显示一个回滚段是否已经脱机

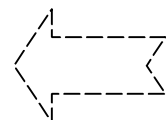
当使一个回滚段脱机时，它并没有真正脱机，直到其中所有的活动事务都完成。在试图使它脱机到它真正脱机这段时间里，它在 V\$ROLLSTAT 中的状态是 PENDING OFFLINE，并且不能被新的事务使用。要确定一个实例的任何回滚段是否处于这种状态，使用如下查询：

```

SELECT NAME, XACTS "ACTIVE TRANSACTIONS"
FROM V$ROLLNAME, V$ROLLSTAT
WHERE STATUS = 'PENDING OFFLINE'
AND V$ROLLNAME.USN = V$ROLLSTAT.USN;
NAME ACTIVE TRANSACTIONS
-----
RS2 3

```

如果用户的实例是 Oracle Real Application Clusters (Oracle 真实应用簇) 配置的一部分，该查询只显示当前实例的回滚段信息，而没有其他实例的。



第 3 部分 模 式 对 象

第 3 部分介绍了在 Oracle 数据库中创建和维护模式对象。包括下列章节：

- 第 14 章 管理模式对象空间
- 第 15 章 管理表
- 第 16 章 管理索引
- 第 17 章 管理分区表和分区索引
- 第 18 章 管理簇
- 第 19 章 管理散列簇
- 第 20 章 管理视图、序列和同义词
- 第 21 章 模式对象的常规管理
- 第 22 章 检测和修复数据块损坏

第 14 章 管理模式对象空间

本章提供对管理模式对象空间的指导。包括以下内容：

- 管理数据块空间
- 设置存储参数
- 管理可恢复空间分配
- 回收空间
- 了解数据类型的空间使用

在试着对后续章节所介绍的特定模式对象进行管理之前，应该先熟悉本章中的概念。

14.1 管理数据块空间

本节介绍管理数据块空间方面的内容。数据块是数据库数据存储在磁盘上的结构的最佳粒度。数据块的大小在创建数据库时指定（或默认）。

PCTFREE 和 PCTUSED 参数是物理属性，可以在模式对象创建或修改时指定。这些参数允许用户控制数据块中空闲空间的使用。空闲空间用于数据行的插入和更新。

PCTFREE 和 PCTUSED 参数允许用户：

- 改善写入和检索数据时的性能
- 减少数据块中未使用空间的数量
- 减少数据块之间行链接的数量

INITRANS 和 MAXTRANS 参数也是可在模式对象创建或修改时指定的物理属性。这些参数控制分配给一个模式对象的数据块的同时更新的事务的数量，它们依次影响数据块头部空间的使用和数据块空闲空间。

本节包括以下内容：

- 指定 PCTFREE 参数
- 指定 PCTUSED 参数
- 选择关联的 PCTFREE 和 PCTUSED 值
- 指定事务入口参数：INITRANS 和 MAXTRANS

14.1.1 指定 PCTFREE 参数

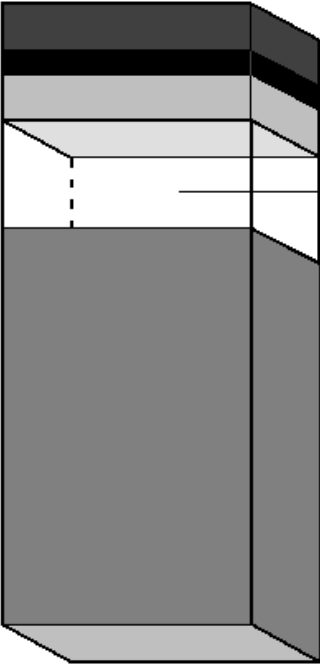
PCTFREE 参数用于设置为数据块中已有行的可能更新而保留的百分比。例如，假定在 CREATE TABLE 语句中指定如下参数：

```
PCTFREE 20
```

这表明该表数据段所用的每个数据块中 20% 的空间被保留，并用于每个数据块中已有行的可能更新。图 14-1 举例说明了 PCTFREE。

数据库块

PCTFREE = 20



20%空闲空间

数据块允许插入行直到占用 80% 的空间，保留 20% 的空间用于数据块中已有行的更新

图 14-1 PCTFREE

注意在数据块到达 PCTFREE 指定的百分比之前，其空闲空间同时被新插入的行和数据块头部的增长所填充。

在设置 PCTFREE 参数之前，确保已经对表或索引数据的特性有所了解。更新可能导致行的增长。新值与所取代的值可能并不是同样大小。如果有很多数据值变大的更新，PCTFREE 参数应该增大。如果行的更新不影响总的行宽，PCTFREE 参数应该低一点。目的是在高度压缩的数据和良好的更新性能之间找到令人满意的平衡。

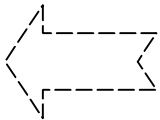
注意：如果修改表空间中一个对象的 PCTFREE 参数值，且该表空间的段管理被指定为 AUTO，也应该调用 DBMS_REPAIR.SEGMENT_FIX_STATUS 过程。这确保为数据块重新计算位图状态以反映新的 PCTFREE 参数值。

PCTFREE 参数的默认值是 10%。可以使用 0~99（包括在内）之间的任何整数，只要 PCTFREE 和 PCTUSED 参数取值之和不超过 100 即可。

14.1.1.1 指定较小 PCTFREE 参数值的作用

一个较小的 PCTFREE 参数值有如下作用：

- 为盘区表中现有行的更新保留较少的空间
- 允许通过插入来更完整地填充数据块
- 可以节省空间，因为表或索引的所有数据存储在较少的数据块中（每个数据块可以有更多的行和入口）



例如，对一个很少改变的段来说，一个较小的 PCTFREE 参数值也许是合适的。

14.1.1.2 指定较大 PCTFREE 参数值的作用

一个较大的 PCTFREE 参数值有如下作用：

- 为表中现有行的将来更新保留更多的空间
- 可以为同样数量的插入数据申请更多的数据块（为每个数据块插入较少的行）
- 可以提高更新性能，因为 Oracle 不必频繁地链接行片

例如，对一个频繁更新的段来说，一个较大的 PCTFREE 参数值是合适的。

14.1.1.3 非簇表的 PCTFREE 参数

如果在非簇表的行中的数据可能会随着时间的推移增大，为这些更新保留一点空间。否则，更新后的行可能在数据块之间链接。

14.1.1.4 簇表的 PCTFREE 参数

对非簇表的讨论也可用于簇表。然而，如果达到 PCTFREE 参数指定的百分比，来自同一簇码中所包含的任何一个表的新行进入到一个新的数据块，该数据块被链接到现有的簇码。

14.1.1.5 索引的 PCTFREE 参数

只能在初始创建索引时指定 PCTFREE 参数。

14.1.2 指定 PCTUSED 参数

在一个数据块按 PCTFREE 参数所确定的百分比填满之后，Oracle 不考虑为该数据块插入新行，直到数据块所用空间的百分比降低到 PCTUSED 参数值之下。在达到该值之前，Oracle 只使用数据块的空闲空间来更新数据块中已有的行。例如，假定在 CREATE TABLE 语句中指定如下参数：

PCTUSED 40

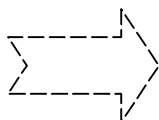
在这种情况下，该表数据段所用数据块不考虑插入任何新行，直到数据块中所用空间的数量降低到 39%或更低（假定数据块所用空间原先已经达到 PCTFREE 参数指定的值）。

图 14-2 举例说明了这种情况。

PCTUSED 参数的默认值是 40%。在数据块中的空闲空间达到 PCTFREE 指定值之后，该数据块中不插入新行，直到所用空间的百分比降低到 PCTUSED 参数值之下。该百分比值是数据块总空间中减去系统开销后用于数据的空间。

可以为 PCTUSED 参数指定 0~99（包括在内）之间的任何整数，只要 PCTFREE 和 PCTUSED 参数取值之和不超过 100 即可。

注意：如果对象是在本地管理的表空间中使用段空间管理指定为 AUTO 的方式所创建，则 PCTUSED 参数被忽略。



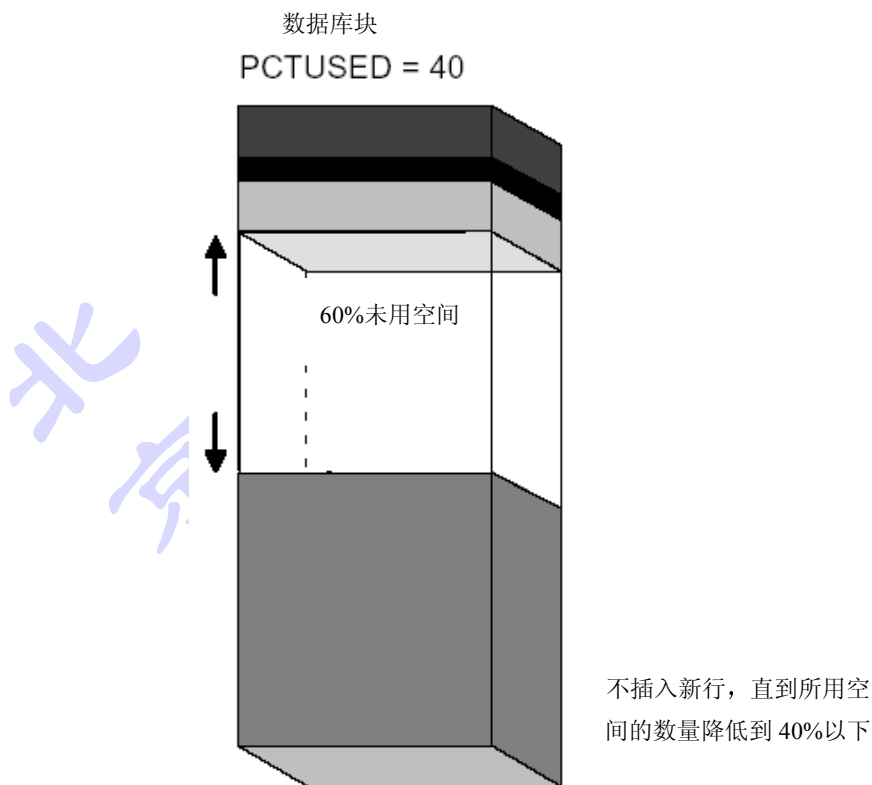


图 14-2 PCTUSED

14.1.2.1 指定较小 PCTUSED 参数值的作用

一个较小的 PCTUSED 参数值有如下作用：

- 当它降低到使用百分比以下时，减少在 UPDATE 和 DELETE 语句中因将数据块移动到自由列表引起的处理开销
- 增加数据库中的未用空间

14.1.2.2 指定较大 PCTUSED 参数值的作用

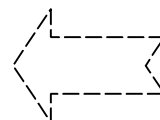
一个较大的 PCTUSED 参数值有如下作用：

- 提高空间利用率
- 在 INSERT 和 UPDATE 语句中增加处理开销

14.1.3 选择关联的 PCTUSED 和 PCTFREE 参数值

如果决定不使用 PCTFREE 或 PCTUSED 参数的默认值，记住如下要点：

- PCTFREE 和 PCTUSED 参数取值之和必须等于或小于 100。
- 如果两参数值之和等于 100，Oracle 只是保持 PCTFREE 参数指定的空闲空间，并且处理开销是最大的。
- PCTFREE 和 PCTUSED 参数值之和与 100 的差值越小(如 PCTUSED 参数值为 75，



PCTFREE 参数值为 25)，空间利用率越高，性能上受些影响。

下表包含一些例子，说明如何和为何要为表指定特定的 PCTFREE 和 PCTUSED 参数值。

例子	方案	设置	说明
1	包括增大行大小的 UPDATE 语句的普通动作。	PCTFREE=20 PCTUSED=40	PCTFREE 设置为 20 为作为更新结果的行大小增长提供足够的空间。PCTUSED 设置为 40 以便在更新动作很多时少做处理，从而提高性能。
2	包含 INSERT 和 DELETE 语句、以及不增大更新行大小的 UPDATE 语句的大多数动作。	PCTFREE=5 PCTUSED=60	PCTFREE 设置为 5，因为大多数 UPDATE 语句不增大行大小。PCTUSED 设置为 60 以便由 DELETE 语句释放的空间很快可以使用，而处理开销是最小的。
3	表非常大并且存储是首要考虑。包含只读事务的大多数动作。	PCTFREE=5 PCTUSED=40	PCTFREE 设置为 5，因为这是个大表并且想要完全填充每个数据块。

14.1.4 指定事务入口参数：INITRANS 和 MAXTRANS

INITRANS 参数指定 DML 事务入口的数目，在数据块头部中最初保留了空间。在关联段中所有数据块的头部保留了空间。

当多个事务同时访问同一数据块的行时，空间被分配给数据块中每个 DML 事务入口。一旦 INITRANS 参数所保留的空间被用光，在数据块中分配的附加事务入口空间是不包括空闲空间在内的可用空间。一旦被分配，该空间有效地成为数据块头部中固定的一部分。MAXTRANS 参数限制在一个数据块中同时使用数据的事务入口的数目。因此，使用 MAXTRANS 参数可以限制在一个数据块中分配给事务入口的空闲空间的数目。

分配给一个特定模式对象的数据块的 INITRANS 和 MAXTRANS 参数，应该基于如下标准，针对每个模式对象单独设置：

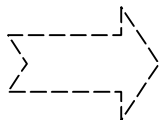
- 为事务入口保留的空间与为数据库数据保留的空间相比较
- 任何时候可能访问同一数据块的并发事务的数目

例如，如果一个表非常大，并且只有少数的用户同时访问该表，多个并发事务请求访问同一数据块的机会就很小。因此，INITRANS 参数可以设置得小一点，特别是如果数据库中的空间非常宝贵的情况。

另一种情况是，假定一个表通常是被很多用户在同一时间访问。在这种情况下，可以考虑使用较大的 INITRANS 参数来预分配事务入口空间。这省去了必须分配事务入口空间的开销，正如对象在使用中要求的那样。另外，设置一个较大的 MAXTRANS 参数值以便没有用户为访问必要的数据库块而必须等待。

14.2 设置存储参数

本节介绍可以为各种数据结构设置的存储参数。这些存储参数用于下列类型的结构和模式对象：



- 表空间（用作所有段的存储参数默认值）
- 表、分区、簇、物化视图和物化视图日志（数据段）
- 索引（索引段）
- 回滚段

本节将讨论如下内容：

- 确定存储参数
- 为表空间中的段设置默认的存储参数
- 为数据段设置存储参数
- 为索引段设置存储参数
- 为 LOB（大型对象）、VARRAY 和嵌套表设置存储参数
- 修改存储参数值
- 了解存储参数的先后次序
- 存储参数如何影响空间分配的例子

14.2.1 确定存储参数

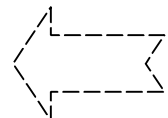
每个数据库都有存储参数的默认值。但是，可以为表空间指定新的默认值，它将覆盖系统默认值，成为只是在该表空间中所创建对象的默认值。这些默认存储参数值在 CREATE 或 ALTER TABLESPACE 语句的 DEFAULT STORAGE 子句中指定。

更进一步，可以为每个单独的模式对象指定存储设置，它们覆盖任何默认的存储设置。要这样做，使用单个对象的 CREATE 或 ALTER 语句的 STORAGE 子句。下面的例子举例说明了在创建表时指定存储参数：

```
CREATE TABLE players
(code NUMBER(10) PRIMARY KEY,
lastname VARCHAR(20),
firstname VARCHAR(15),
position VARCHAR2(20),
team VARCHAR2(20))
PCTFREE 10
PCTUSED 40
STORAGE
(INITIAL 25K
NEXT 10K
MAXEXTENTS 10
MINEXTENTS 3);
```

不是每种类型的数据库对象都可以指定所有的存储参数，并且也不是所有的存储参数都可以同时在 CREATE 和 ALTER 语句中指定。要设置或修改一个存储参数的值，必须具有使用合适的 CREATE 或 ALTER 语句的必要权限。

下面章节确定了可以指定的存储参数。



14.2.1.1 参数 INITIAL

以字节计，当创建段时分配的第一个盘区的大小。该参数不能在 ALTER 语句中指定。

默认值:	5 个数据块
最小值:	2 个数据块（在字典管理的表空间中），3 个数据块（在本地管理的表空间中）
最大值:	操作系统指定

14.2.1.2 参数 NEXT

以字节计，分配给一个段的下一个增加的盘区的大小。第二个盘区等于 NEXT 的原始设置。从此往后，NEXT 设置为上一次 NEXT 乘上 $(1 + \text{PCTINCREASE}/100)$ 的大小。

默认值:	5 个数据块
最小值:	1 个数据块
最大值:	操作系统指定

14.2.1.3 参数 PCTINCREASE

分配给一个段的每次增加的盘区在上一次增加盘区的基础上增长的百分比。如果 PCTINCREASE 参数值为 0，那么所有的增加的盘区大小都相同。如果 PCTINCREASE 参数值比 0 大，那么每次 NEXT 要计算，它增长 PCTINCREASE 参数的指定值。PCTINCREASE 参数不能为负值。

新的 NEXT 参数值等于 $1 + \text{PCTINCREASE}/100$ ，乘以上一次增加盘区的大小（旧的 NEXT 参数值），并且上舍入到相邻的数据块大小的倍数。

默认值:	50 (%)
最小值:	0 (%)
最大值:	操作系统指定

14.2.1.4 参数 MINEXTENTS

创建段时分配的盘区的总数。允许在创建时分配一个大的空间，即使连续空间不够用。

默认值:	1 个（盘区）；回滚段为 2 个（盘区）
最小值:	1 个（盘区）；回滚段为 2 个（盘区）
最大值:	操作系统指定

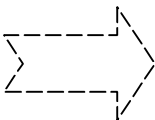
14.2.1.5 参数 MAXEXTENTS

所有能够分配给段的，包括第一次的盘区的总数。

默认值:	取决于数据块大小和操作系统
最小值:	1 个（盘区）；回滚段为 2 个（盘区）
最大值:	没有限制

14.2.1.6 参数 FREELIST GROUPS

正在创建的数据库对象的空闲表组数。Oracle 使用 Oracle Real Application Cluster 实例的实例数将每个实例映射到一个空闲表组。



默认值:	1
最小值:	1
最大值:	取决于 Oracle Real Application Cluster 实例的数目

在使用指定为 AUTO 的段空间管理的本地管理的表空间中创建对象时,该参数被忽略。

14.2.1.7 参数 FREELISTS

指定模式对象的每个空闲表组中空闲表的数目。对表空间无效。

默认值:	1
最小值:	1
最大值:	取决于数据块大小

在使用指定为 AUTO 的段空间管理的本地管理的表空间中创建对象时,该参数被忽略。

14.2.1.8 参数 OPTIMAL

只与回滚段有关。

14.2.1.9 参数 BUFFER_POOL

为模式对象定义一个默认的缓冲池（高速缓存）。对表空间或回滚段无效。

14.2.2 为表空间中的段设置默认的存储参数

可以为数据库的每个表空间设置默认的存储参数。任何在表空间中创建或随后自动修改一个段时没有明确设置的存储参数，被设置为该段所驻留的表空间的相应的默认存储参数。

当在表空间层指定 MINEXTENTS 参数时，在该表空间中分配的任何盘区舍入到一个最小盘区的倍数。

14.2.3 为数据段设置存储参数

使用表、物化视图或物化视图日志的 CREATE 或 ALTER 语句的 STORAGE 子句，可以为非簇表、物化视图或物化视图日志的数据段设置存储参数。

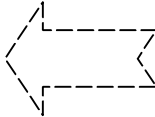
相反地，使用 CREATE CLUSTER 或 ALTER CLUSTER 语句的 STORAGE 子句，可以为一个簇的数据段设置存储参数，而不是单独的将表和物化视图插入到该簇的 CREATE 或 ALTER 语句。当创建或修改簇表或物化视图时，指定的存储参数被忽略。为簇设置的存储参数覆盖表的存储参数。

使用分区表，可以在表的层次上设置默认的存储参数。当创建表的一个新分区时，默认的存储参数从表层继承得来（除非为单个的分区指定存储参数）。如果在表的层次上没有指定存储参数，那么它们从表空间继承得来。

14.2.4 为索引段设置存储参数

为一个表的索引所创建的索引段的存储参数，可以使用 CREATE INDEX 或 ALTER INDEX 语句的 STORAGE 子句来设置。

为一个用来增强主键或唯一键约束的索引创建的索引段的存储参数，可以用下列两种



方法来设置：

- 在 CREATE TABLE 或 ALTER TABLE 语句的 ENABLE ... USING INDEX 子句中设置
- 在 ALTER INDEX 语句的 STORAGE 子句中设置

14.2.5 为 LOB（大型对象）、VARRAY 和嵌套表设置存储参数

一个表或物化视图可以包含 LOB、VARRAY 或嵌套表列类型。这些实体存储在它们自己的段中。LOB 和 VARRAY 存储在 LOB 段中，而嵌套表存储在存储表中。可以为这些段指定一个 STORAGE 子句，从而覆盖在表的层次上指定的存储参数。

14.2.6 修改存储参数值

可以修改表空间默认的存储参数和单个段的特定存储参数。可以为表空间重置默认的存储参数。然而，修改只影响表空间中新创建的对象，或者为段新分配的盘区。

不能为现有的表、簇、索引或回滚段修改 INITIAL 和 MINEXTENTS 存储参数。如果一个段只有 NEXT 参数被修改，下一次增加盘区就是新 NEXT 的大小，随后的盘区像通常那样按 PCTINCREASE 参数增长。

如果一个段 NEXT 和 PCTINCREASE 两个参数都被修改，下一个盘区就是 NEXT 的新值，从此开始 NEXT 像通常那样按 PCTINCREASE 参数计算。

14.2.7 了解存储参数的先后次序

在给定时间内有效的存储参数由下列类型的 SQL 语句决定，按照先后次序列出（先后次序按数字从大到小排列）：

- ① ALTER [TABLE|CLUSTER|MATERIALIZED VIEW|MATERIALIZED VIEW LOG|INDEX|ROLLBACK] SEGMENT 语句
- ② CREATE [TABLE|CLUSTER|MATERIALIZED VIEW|MATERIALIZED VIEW LOG|INDEX|ROLLBACK] SEGMENT 语句
- ③ ALTER TABLESPACE 语句
- ④ CREATE TABLESPACE 语句
- ⑤ Oracle 默认值

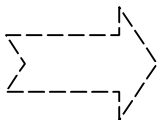
任何在对象层设置的存储参数覆盖在表空间层设置的相应选项。当存储参数在对象层没有明确设置时，它们默认为表空间层的取值。当存储参数在表空间层没有设置时，使用 Oracle 系统默认值。如果存储参数被修改，新的选项只用于还没有分配的盘区。

注意：临时段的存储参数通常使用相关联的表空间所设置的默认存储参数。

14.2.8 存储参数如何影响空间分配的例子

假定下列语句已经被执行：

```
CREATE TABLE test_storage
( . . . )
STORAGE (INITIAL 100K NEXT 100K
```



```
MINEXTENTS 2 MAXEXTENTS 5
PCTINCREASE 50);
```

还假定初始化参数 DB_BLOCK_SIZE 被设置为 2K。下表说明了如何为表 TEST_STORAGE 分配盘区。也说明了增加盘区的取值，如在数据字典视图 USER_SEGMENTS 或 DBA_SEGMENTS 的 NEXT 列中所见的那样：

表 14-1 盘区分配

盘区号	盘区大小	NEXT 取值
1	50 个块或 102400 字节	50 个块或 102400 字节
2	50 个块或 102400 字节	75 个块或 153600 字节
3	75 个块或 153600 字节	113 个块或 231424 字节
4	115 个块或 235520 字节	170 个块或 348160 字节
5	170 个块或 348160 字节	没有下一个值，MAXEXTENTS=5

如果在 ALTER 语句（例如 ALTER TABLE）中修改 NEXT 或 PCTINCREASE 存储参数，指定值取代存储在数据字典中的当前值。例如，下面语句在为表 test_storage 分配第三个盘区之前，修改该表的 NEXT 存储参数：

```
ALTER TABLE test_storage STORAGE (NEXT 500K);
```

结果是第三个盘区在分配时为 500K，第四个为 (500K*1.5) =750K，以此类推。

14.3 管理可恢复空间分配

Oracle 为大型数据库操作的执行，提供在空间分配失败的情况下暂停，并在以后恢复的方法。这使得用户可以采取正确的动作，而不是由 Oracle 数据库服务器为用户返回错误信息。在出错条件得到改正之后，暂停的操作自动恢复。这种特性被称作可恢复空间分配。与之相关的语句被称作可恢复语句。

本节包括以下内容：

- 可恢复空间分配的概述
- 启用和禁用可恢复空间分配
- 检测暂停的语句
- 可恢复空间分配的例子：注册一个 AFTER SUSPEND 触发器

14.3.1 可恢复空间分配的概述

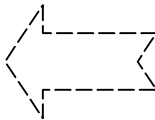
本节提供可恢复空间分配的概述。它介绍了可恢复语句如何工作，特别是定义限制语句和出错条件。

14.3.1.1 可恢复语句如何工作

下面是可恢复语句如何工作的概述。细节包含在后续的章节中。

①只有当客户使用 ALTER SESSION 语句明确地启用会话的可恢复语义时，一个语句才能以可恢复方式执行。

②当下列条件（这些条件导致告知非恢复语句的相应的错误）之一出现时，可恢复语



句被暂停：

- 没有空间条件
- 达到最大盘区条件
- 超过空间限额条件

③在可恢复语句执行被暂停期间，有一些机制来实现用户提供的操作、错误日志和查询语句执行的状态。当一个可恢复语句被暂停时，采取下列动作：

- 在修改日志中报告错误。
- 如果用户在 AFTER SUSPEND 系统事件上注册了一个触发器，那么用户的触发器被执行。一个用户提供的 PL/SQL 过程能够使用 DBMS_RESUMABLE 包和 DBA/USER_RESUMABLE 视图访问错误信息数据。

④暂停一个语句自动导致暂停该事务。因而从一个语句暂停到恢复期间，所有的事务资源被保持。

⑤当出错条件消失（例如，作为用户干预或也许是其他查询所释放空间进行排序的结果）时，暂停的语句自动恢复执行。

⑥使用 DBMS_RESUMABLE.ABORT（）过程，一个被暂停的语句可以被强制发出异常。该过程可以被 DBA 或者是发布这条语句的用户调用。

⑦暂停超时间隔与可恢复语句相关联。一个暂停了超时间隔（默认是 2 小时）的可恢复语句被唤醒并对用户返回异常。

⑧一个可恢复语句在执行期间可被多次暂停和恢复。

14.3.1.2 哪些操作是可恢复的

注意：当使用本地管理的表空间时，可恢复空间分配是被完全支持的。当使用字典管理的表空间时，有某些限制。

下列操作是可恢复的：

- 查询

用尽临时空间（用作排序区）的 SELECT 语句是可恢复执行的候选项。当使用 OCI 时，OCISstmtExecute（）和 OCISstmtFetch（）调用是候选项。

- DML

INSERT、UPDATE 和 DELETE 语句是候选项。与执行它们的接口无关；可以是 OCI、JSQL、PL/SQL 或者其他接口。另外，来自外部表的 INSERT INTO ... SELECT 可以是可恢复的。

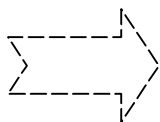
- 导入/导出

对于 SQL*装载器，一个命令行参数控制语句在出现可改正错误之后是否是可恢复的。

- DDL

下列语句是可恢复执行的候选项：

- CREATE TABLE ... AS SELECT
- CREATE INDEX
- ALTER INDEX ... REBUILD



- ALTER TABLE ... MOVE PARTITION
- ALTER TABLE ... SPLIT PARTITION
- ALTER INDEX ... REBUILD PARTITION
- ALTER INDEX ... SPLIT PARTITION
- CREATE MATERIALIZED VIEW
- CREATE MATERIALIZED VIEW LOG

14.3.1.3 哪些错误是可改正的

有 3 类可改正错误：

- 没有空间条件

该操作不能再为表空间中的表/索引/临时段/回滚段/撤消段/簇/LOB/表分区/索引分区获取更多的盘区。例如，下面出现的错误属于此类：

```
ORA-1650 unable to extend rollback segment ... in tablespace ...
```

```
ORA-1653 unable to extend table ... in tablespace ...
```

```
ORA-1654 unable to extend index ... in tablespace ...
```

- 达到最大盘区条件

一个表/索引/临时段/回滚段/撤消段/簇/LOB/表分区/索引分区的盘区数等于该对象所定义的最大盘区数。例如，下面出现的错误属于此类：

```
ORA-1628 max # extents ... reached for rollback segment ...
```

```
ORA-1631 max # extents ... reached in table ...
```

```
ORA-1654 max # extents ... reached in index ...
```

- 超过空间限额条件

用户超出了表空间中分配给他的空间限额。特别地，在下面的错误中记录：

```
ORA-1536 space quota exceeded for tablespace string
```

14.3.1.4 字典管理的表空间的可恢复空间分配的限制

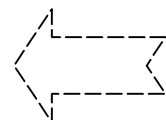
当使用字典管理的表空间时，有一些限制。这些限制在下面列出：

1. 一个像 CREATE TABLE 或 CREATE INDEX 这样的 DDL 操作在执行时，使用了在执行期间导致用完空间错误的明确的 MAXEXTENTS 参数设置，该操作不会被暂停。而是被放弃。该错误被当作是不能修复的，因为一个对象的属性（例如 MAXEXTENTS）不能在它被创建之前就修改。但是，如果一个 DML 操作导致一个已经存在的表或索引达到 MAXEXTENTS 参数的限制，它会被暂停并在以后恢复。这种限制可以通过设置 MAXEXTENTS 子句为 UNLIMITED 或使用本地管理的表空间来克服。

2. 如果回滚段位于字典管理的表空间中，那么对回滚段的空间分配是不可恢复的。然而，对用户对象（表、索引这类的）的空间分配仍然是可恢复的。要克服这种限制，推荐使用自动撤消管理或将回滚段放置在本地管理的表空间中。

14.3.1.5 可恢复语句和分布式操作

在可恢复方式中，不支持远程操作。



14.3.1.6 并行执行和可恢复语句

在并行执行中，如果并行执行服务器进程中的一个遇到可改正错误，该服务器进程暂停它的执行。其他的并行执行服务器进程将继续执行它们各自的任务，直到它们遇到一个错误或被暂停的服务器进程所阻塞（直接或间接）。当可改正错误被解决，暂停的进程恢复执行并且该并行操作继续执行。如果该暂停的操作被结束，该并行操作被放弃，将错误发给用户。

不同的并行执行服务器进程可能遇到一个或更多的可改正错误。这可能导致多次以并行方式激发 AFTER SUSPEND 触发器。另外，如果一个并行执行服务器进程遇到一个不可改正的错误，而另一个并行执行服务器进程被暂停，暂停的语句立即被放弃。

对于并行执行，每个并行执行协调程序和服务器进程在 DBA/USER_RESUMABLE 视图中都有自己的入口。

14.3.2 启用和禁用可恢复空间分配

只有当会话已经启用可恢复方式且语句在该会话中执行时，可恢复空间分配才可能实现。

要启用一个会话的可恢复方式，使用如下的 SQL 语句：

```
ALTER SESSION ENABLE RESUMABLE;
```

因为暂停的语句可以保持一些系统资源，用户必须在启用可恢复语句和执行前，被授予 RESUMABLE 系统权限。

要禁用可恢复方式，发布如下语句：

```
ALTER SESSION DISABLE RESUMABLE;
```

一个新的会话默认地禁用可恢复方式。

还可以指定一个超时间隔，并且提供一个用于标识一个可恢复语句的名称。这些将在以下章节中分别讨论。

14.3.2.1 指定超时间隔

当启用会话的可恢复方式时，也可以指定一个超时间隔，超过该间隔如果没有采取干预，暂停的语句会出错。如下的语句指定可恢复事务在 3600 秒后将超时和出错：

```
ALTER SESSION ENABLE RESUMABLE TIMEOUT 3600;
```

TIMEOUT 值保持有效直到它被另一个 ALTER SESSION ENABLE RESUMABLE 语句所修改，它被另一种方法修改，或者会话结束。超时间隔的默认值是 7200 秒。

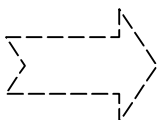
14.3.2.2 为可恢复语句命名

可以通过名称来识别可恢复语句。如下的语句为可恢复语句指定一个名称：

```
ALTER SESSION ENABLE RESUMABLE TIMEOUT 3600 NAME 'insert into table';
```

NAME 值保持有效直到它被另一个 ALTER SESSION ENABLE RESUMABLE 语句所修改，或者会话结束。NAME 的默认值是：

```
User USERNAME(USERID), Session SESSIONID, Instance INSTANCEID
```



语句的名称用于在 DBA_RESUMABLE 和 USER_RESUMABLE 视图中标识该可恢复语句。

14.3.2.3 设置默认的可恢复方式

要设置默认的可恢复方式,一个 DBA 可以注册一个数据库层的 LOGON 触发器来修改用户的会话,启用可恢复方式并设置一个超时间隔。

注意: 如果有多个注册的触发器修改可恢复语句的默认方式和超时间隔,其结果将会是没有指定,因为 Oracle 不能保证触发器启用的顺序。

14.3.2.4 修改超时间隔

除 ALTER SESSION ENABLE RESUMABLE 语句外,还有别的方法设置或修改超时间隔。

DBMS_RESUMABLE 包带有一个为特定会话或当前会话设置超时间隔的过程。一个 DBA 可以通过创建一个系统范围的 AFTER SUSPEND 触发器来修改默认的系统超时间隔,该触发器调用 DBMS_RESUMABLE 来设置超时间隔。例如,如下的代码示例设置一个系统范围的默认超时间隔为 1 小时:

```
CREATE OR REPLACE TRIGGER resumable_default_timeout
AFTER SUSPEND
ON DATABASE
BEGIN
  DBMS_RESUMABLE.SET_TIMEOUT(3600);
END;
```

14.3.3 检测暂停的语句

当一个可恢复语句被暂停时,错误不出现在客户端。为了采取纠正的动作,Oracle 提供可选择的方法通知用户这个错误并提供关于环境的信息。

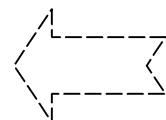
14.3.3.1 AFTER SUSPEND 系统事件和触发器

当可恢复语句遇到一个可改正错误时,系统在内部产生 AFTER SUSPEND 系统事件。用户可以同时在数据库层和模式层为该事件注册触发器。如果一个用户注册一个触发器来处理该系统事件,该触发器在 SQL 语句被暂停之后执行。

在 AFTER SUSPEND 触发器中执行的 SQL 语句通常是不可恢复的和独立的。在触发器中启动的事务使用 SYSTEM 回滚段。这些条件被用来克服死锁,并且减少触发器遭遇与语句同样的出错条件的机会。

用户可以在触发器中使用 USER_RESUMABLE 或 DBA_RESUMABLE 视图,或 DBMS_RESUMABLE.SPACE_ERROR_INFO 函数,来获得关于可恢复语句的信息。

触发器也可以调用 DBMS_RESUMABLE 包来放弃暂停的语句和可恢复超时间隔值。



14.3.3.2 包含关于可恢复语句信息的视图

下列视图可以被查询以获得关于可恢复语句状态的信息。

视图	说明
DBA_RESUMABLE USER_RESUMABLE	这些视图包含所有当前正在执行或被暂停的可恢复语句的行。它们可以用于 DBA、AFTER SUSPEND 触发器，或者另一个会话来监控可恢复语句的进行情况，或获得关于可恢复语句的特定信息。
V\$SESSION_WAIT	当一个语句被暂停，调用该语句的会话进入到等待状态。使用包含 “suspended on space error” 的 EVENT 列，在该视图中为会话插入一行。

14.3.3.3 DBMS_RESUMABLE 包

DBMS_RESUMABLE 包有助于控制可恢复语句。下列过程是可用的：

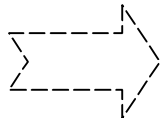
过程	说明
ABORT (sessionID)	该过程放弃一个暂停的可恢复语句。参数 sessionID 是该语句在其中执行的会话的标识。对于并行 DML/DDI，会话标识是参与并行 DML/DDI 的任何一个会话的标识。Oracle 保证 ABORT 操作通常是成功的。它既可以在 AFTER SUSPEND 触发器内，也可以在触发器外被调用。ABORT 的调用者必须是使用 sessionID 的会话的拥有者，有 ALTER SYSTEM 权限，或者有 DBA 权限。
GET_SESSION_TIMEOUT (sessionID)	该函数为使用 sessionID 的会话返回可恢复语句的当前超时间隔值。返回的超时间隔以秒为单位。如果该会话不存在，函数返回-1。
SET_SESSION_TIMEOUT (sessionID, timeout)	该过程为使用 sessionID 的会话设置可恢复语句的超时间隔值。参数 timeout 以秒为单位。新的 timeout 设置将立即用于该会话。如果该会话不存在，不采取任何动作。
GET_TIMEOUT ()	该函数返回当前会话的可恢复语句的当前超时间隔值。返回值以秒为单位。
SET_TIMEOUT (timeout)	该过程为当前会话的可恢复语句设置一个超时间隔值。参数 timeout 以秒为单位。新的 timeout 设置将立即用于该会话。

14.3.4 可恢复空间分配的例子：注册一个 AFTER SUSPEND 触发器

本例子说明了可恢复语句的用法。一个系统范围的 AFTER SUSPEND 触发器作为用户 SYS 在数据库层创建和注册。在任何会话中，只要一个可恢复语句被暂停，该触发器有以下两个作用：

- 如果回滚段达到它的空间限制，那么一条消息发给 DBA 并且该语句被放弃。
- 如果任何其他的可改正错误发生，超时间隔被重置为 8 小时。

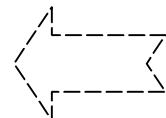
```
CREATE OR REPLACE TRIGGER resumable_default
AFTER SUSPEND
ON DATABASE
DECLARE
```



```

/* declare transaction in this trigger is autonomous */
/* this is not required because transactions within a trigger
are always autonomous */
PRAGMA AUTONOMOUS_TRANSACTION;
cur_sid NUMBER;
cur_inst NUMBER;
errno NUMBER;
err_type VARCHAR2;
object_owner VARCHAR2;
object_type VARCHAR2;
table_space_name VARCHAR2;
object_name VARCHAR2;
sub_object_name VARCHAR2;
error_txt VARCHAR2;
msg_body VARCHAR2;
ret_value BOOLEAN;
mail_conn UTL_SMTP.CONNECTION;
BEGIN
-- Get session ID
SELECT DISTINCT(SID) INTO cur_SID FROM V$MYSTAT;
-- Get instance number
cur_inst := userenv('instance');
-- Get space error information
ret_value :=
DBMS_RESUMABLE.SPACE_ERROR_INFO(err_type,object_type,object_owner,
table_space_name,object_name, sub_object_name);
/*
-- If the error is related to rollback segments, log error, send email
-- to DBA, and abort the statement. Otherwise, set timeout to 8 hours.
--
-- sys.rbs_error is created by DBA manually and defined as
-- sql_text VARCHAR2(1000), error_msg VARCHAR2(4000),
-- suspend_time DATE)
*/
IF OBJECT_TYPE = 'ROLLBACK SEGMENT' THEN
/* LOG ERROR */
INSERT INTO sys.rbs_error (
SELECT SQL_TEXT, ERROR_MSG, SUSPEND_TIME

```



```

FROM DBMS_RESUMABLE
WHERE SESSION_ID = cur_sid AND INSTANCE_ID = cur_inst
);
SELECT ERROR_MSG INTO error_txt FROM DBMS_RESUMABLE
where SESSION_ID = cur_sid and INSTANCE_ID = cur_inst;
-- Send email to receiptent via UTL_SMTP package
msg_body:='Subject: Space Error Occurred
Space limit reached for rollback segment ' || object_name ||
on ' ' || TO_CHAR(SYSDATE, 'Month dd, YYYY, HH:MIam') ||
'. Error message was ' || error_txt;
mail_conn :- UTL_SMTP.OPEN_CONNECTION('localhost', 25);
UTL_SMTP.HELO(mail_conn, 'localhost');
UTL_SMTP.MAIL(mail_conn, 'sender@localhost');
UTL_SMTP.RCPT(mail_conn, 'recipient@localhost');
UTL_SMTP.DATA(mail_conn, msg_body);
UTL_SMTP.QUIT(mail_conn);
-- Abort the statement
DBMS_RESUMABLE.ABORT(cur_sid);
ELSE
-- Set timeout to 8 hours
DBMS_RESUMABLE.SET_TIMEOUT(28800);
END IF;
/* commit autonomous transaction */
COMMIT;
END;

```

14.4 回收空间

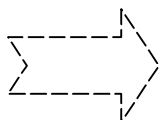
为一个段分配空间并不罕见，只是以后发现所分配的空间没有使用。例如，可以设置 PCTINCREASE 参数为一个较大的值，这可以创建一个只有部分被使用的大型盘区。或者，可以通过发布 ALTER TABLE ... ALLOCATE EXTENT 语句明确地多分配空间。如果发现有未使用的或多分配的空间，可以将其释放以便这些未使用的空间能够被其他的段使用。

本节说明回收未使用空间的方法。

14.4.1 观察高水位标记

在回收之前，可以使用包含一个过程 (UNUSED_SPACE) 的 DBMS_SPACE 包，该过程返回关于一个段中高水位标记的位置和未使用空间的数量的信息。

在一个段中，高水位标记指出已使用空间的数量，或者已经被格式化准备接收数据的空间数量。低于高水位标记的空间不能被释放（即使想要回收的空间中没有数据）。但是，



如果该段全部是空的，可以使用 `TRUNCATE ... DROP STORAGE` 语句来释放空间。

对于使用段空间管理指定为 `AUTO` 的本地管理的表空间中的段，下列输出参数仍能确定高水位标记，但稍微改变了它们的意义：

- `LAST_USED_EXTENT_FILE_ID`
- `LAST_USED_EXTENT_BLOCK_ID`
- `LAST_USED_BLOCK`

特别是，一些低于高水位标记的块没有格式化是可能的。`DBMS_SPACE` 的 `UNUSED_SPACE` 和 `FREE_SPACE` 过程都不能在段空间管理被指定为 `AUTO` 时，准确计算出未使用的空间。使用 `SPACE_USAGE` 过程来代替它们。

14.4.2 发布空间回收语句

下列语句回收一个段（表、索引或簇）中未使用的空间。`KEEP` 子句是可选的。

```
ALTER TABLE table DEALLOCATE UNUSED KEEP integer;
ALTER INDEX index DEALLOCATE UNUSED KEEP integer;
ALTER CLUSTER cluster DEALLOCATE UNUSED KEEP integer;
```

当明确指定 `KEEP` 一定数量的未使用空间，这些空间被保留而剩余的未使用空间被回收。如果剩下的盘区个数变得比 `MINEXTENTS` 参数值小，那么 `MINEXTENTS` 参数值将改变以反映新的个数。如果初始盘区变小，将修改 `INITIAL` 参数值以反映初始盘区的新大小。

如果没有指定 `KEEP` 子句，所有未使用空间（高于高水位标记的一切空间）都被回收，只要初始盘区的大小和 `MINEXTENTS` 参数值被保存。因而，即使高水位标记出现在 `MINEXTENTS` 边界内，`MINEXTENTS` 保持不变并且初始盘区大小不缩减。

可以通过检查 `DBA_FREE_SPACE` 视图来验证回收的空间已被释放。

14.4.3 回收空间的例子

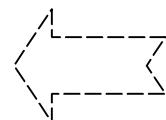
本节提供一些空间回收的例子。

14.4.3.1 回收空间例子 1

一个表由 3 个盘区组成。第一个盘区是 10K，第二个是 20K，第三个是 30K。高水位标记在第二个盘区的中部，并且有 40K 的未使用空间。图 14-3 举例说明了发布如下语句的作用：

```
ALTER TABLE dquon DEALLOCATE UNUSED
```

所有未使用的空间被回收，留下表 `dquon`，它剩下两个盘区。第三个盘区消失，而第二个盘区的大小为 10K。



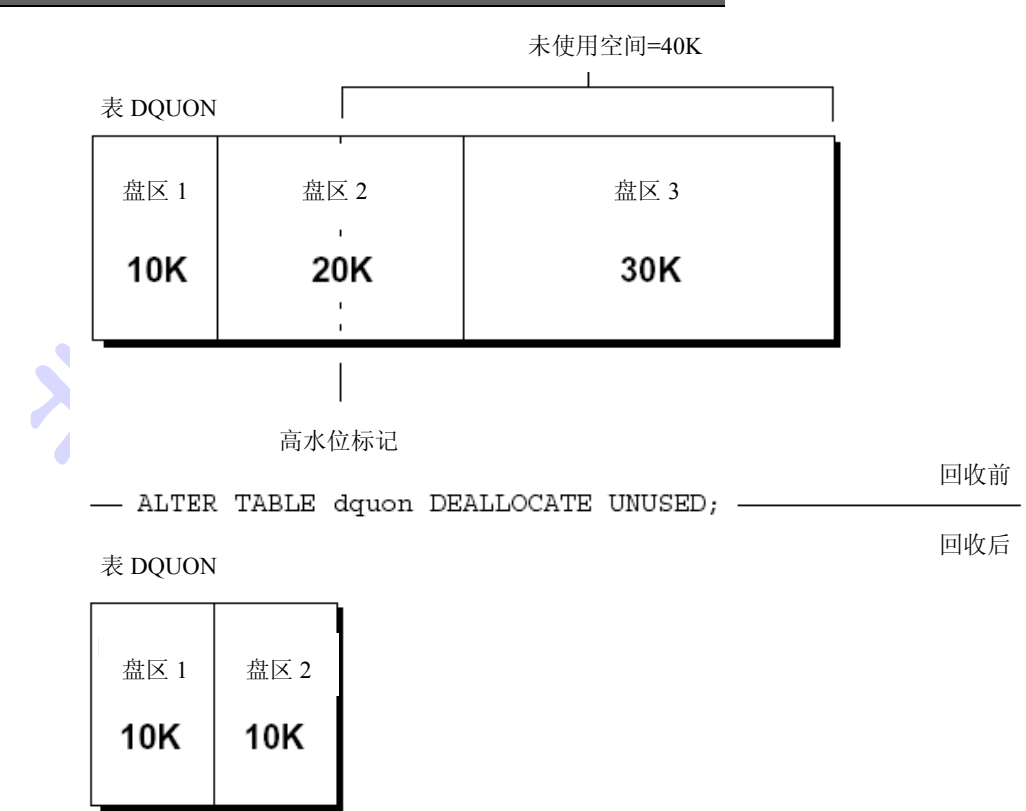


图 14-3 回收所有的未使用空间

但是，如果曾经发布了指定 **KEEP** 关键字的如下语句，那么高于高水位标记的 10K 空间将被保留，表 **dquon** 中的剩余未使用空间将被回收。

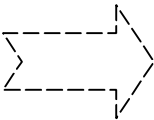
```
ALTER TABLE dquon DEALLOCATE UNUSED KEEP 10K;
```

实际上，第三个盘区被回收，而第二个盘区保持原样。

图 14-4 举例说明了这种情况。

更进一步，如果从表 **dquon** 中回收所有的未使用空间并保留 20K，像如下语句中指定的那样，第三个盘区被减为 10K，而第二个盘区的大小保持原样。

```
ALTER TABLE dquon DEALLOCATE UNUSED KEEP 20K;
```



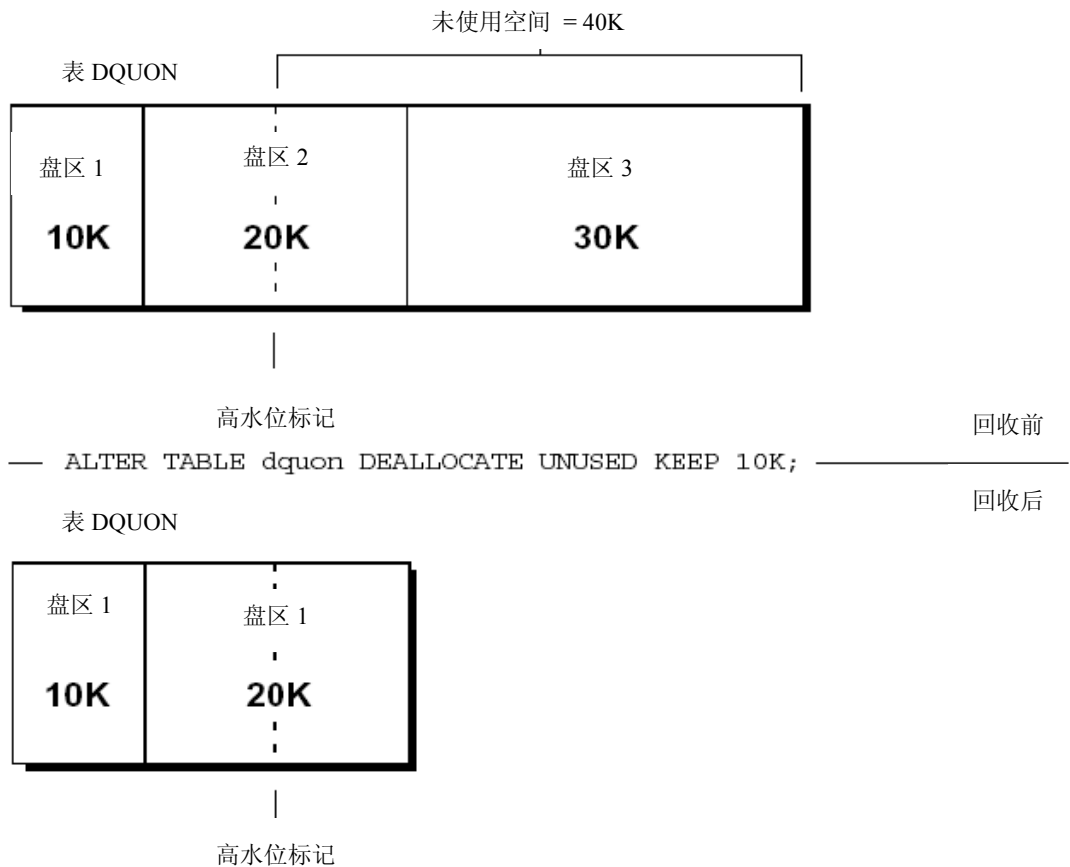


图 14-4 回收未使用空间，KEEP 10K

14.4.3.2 回收空间例子 2

考虑图 14-3 举例说明的这种情况。盘区 3 完全被回收，并且第二个盘区剩下 10K。更进一步，下一个分配的盘区的大小默认为最后完全回收的盘区的大小，就此而言是 30K。如果这不是所希望的，可以使用 ALTER TABLE 语句明确设置下一个盘区的大小，在存储子句中为下一个盘区指定一个新值。

如下语句为表 dquon 设置下一个盘区大小为 20K：

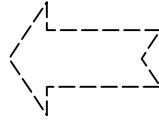
```
ALTER TABLE dquon STORAGE (NEXT 20K)
```

14.4.3.3 回收空间例子 3

为保持盘区的 MINEXTENTS 个数，DEALLOCATE 可以保留原先分配给一个段的盘区。这种功能受 KEEP 参数的影响并且在前面解释过了。

如果表 dquon 有值为 2 的 MINEXTENTS 参数，在图 14-3 和图 14-4 中举例说明的语句仍然产生如图所示的结果，并且更进一步，MINEXTENTS 参数的初始值被保留。

但是，如果 MINEXTENTS 参数值为 3，那么在图 14-4 中举例说明的语句仍然产生如图所示的相同结果（第三个盘区被删除），但 MINEXTENTS 参数值改为 2。然而，在图 14-3

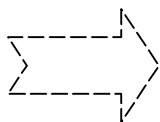


中举例说明的语句不会产生同样的结果。就此而言，该语句无效。

14.5 了解数据类型的空间使用

当创建表和其他数据结构时，必须知道它们需要多少空间。每个数据类型有不同的空间需求。

北京拓墣电子书出版社



第 15 章 管 理 表

本章描述管理表的几个方面，包括如下内容：

- 管理表的准则
- 创建表
- 更改表
- 联机重定义表
- 删除表
- 管理索引结构表
- 管理外部表
- 查看有关表的信息

15.1 管理表的准则

本节描述管理表时所遵循的准则。遵循这些准则就能更容易地管理表，并且能改善创建表以及其后查询或更新表时的性能。

讨论的内容如下：

- 创建表之前设计表
- 指定如何使用数据块空间
- 指定每个表的位置
- 考虑并行创建表
- 考虑创建表时使用 NOLOGGING
- 估计表的大小并设置存储参数
- 规划大表
- 表的限制

15.1.1 创建表之前设计表

通常，应用的开发者负责设计应用的各个组件，包括表。基于来自应用的开发者的有关应用如何运作和所期望的数据类型，数据库管理员负责设置存储参数和给表定义簇。

与应用开发者一起工作，并认真规划每个表，以致于做到：

- 表被规范化
- 每个列具有合适的数据类型
- 最后定义允许空值的列，以便节约存储空间
- 合适的时候对表分簇，以便节省存储空间和优化 SQL 语句的性能。簇表是第 18 章“管理簇”的内容。

15.1.2 指定如何使用数据块空间

通过在创建每个表时指定 PCTFREE 和 PCTUSED 参数，可以影响空间的利用效率和

在表的数据段的数据块中用于更新当前数据所保留的空间数量。PCTFREE 和 PCTUSED 参数在第 14 章“管理数据块空间”中讨论。

15.1.3 指定每个表的位置

如果具有合适的权限和表空间限额，就可以在任何当前联机的表空间中创建一个新表。建议在 CREATE TABLE 语句中指定 TABLESPACE 子句，以便指定存储新表的表空间。如果在 CREATE TABLE 语句中没有指定表空间，则该表将被创建在默认表空间中。

在为新表指定表空间时，应该明白所做选择的实质。在创建每个表时，通过指定合适的表空间，就能：

- 提高数据库系统的性能
- 减少数据库管理的时间

下面的情况描述了，当为模式对象指定了不正确的位置时会对数据库产生的影响：

- 如果用户对象被创建在 SYSTEM 表空间，将影响 Oracle 的性能，因为数据字典对象和用户对象必然会竞争相同的数据文件。
- 如果与应用相关联的表被随机地存储在多个表空间，那么，为那个应用的数据所进行的管理操作（如备份和恢复）所需的时间就会增加。

第 24 章“管理用户和资源”包含了有关给用户分配默认表空间和表空间限额的信息。

15.1.4 考虑并行创建表

当在 CREATE TABLE 语句中使用子查询（如 SELECT）来创建表时，就可以利用并行执行。因为多个进程一起来创建表，所以就改善了创建表的效率。

并行创建表在本章的“并行创建表”一节中讨论。

15.1.5 考虑创建表时使用 NOLOGGING

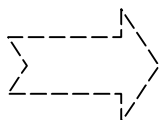
为了最有效地创建表，在 CREATE TABLE...AS SELECT 语句中使用 NOLOGGING 子句。NOLOGGING 子句能使创建表时所产生的重做信息量最小。这有如下好处：

- 节省了重做日志文件的空间。
- 减少了创建表的时间
- 改善了大表的并行创建的性能

NOLOGGING 子句还指定，不登记后续的用 SQL*Loader 和 INSERT 操作所做的直接加载操作。后续的 DML 语句（UPDATE，DELETE 和常规的路径插入）不受表的 NOLOGGING 属性的影响，并产生重做登记。

如果创建表后不能承担丢失表的损失（如，不再能够访问创建表的数据）就应该在创建表后立即备份。在某些情况下，如为临时使用而创建的表，就不需要做这个防范了。

通常，指定了 NOLOGGING 后，大表的性能改善相对于小表而言要好些。对小表而言，NOLOGGING 对创建表所用的时间只有点点影响。但对大表的性能改善可能很明显，特别是还用并行表创建方法时。



15.1.6 估计表的大小和设置存储参数

因为如下原因，在创建表之前估计表的大小是有用的：

- 可以对表的大小做综合估计，包括对索引、回滚段和重做日志文件的估计，以决定支持期望的数据库所需的磁盘空间的大小。通过这些估计，就可以购买合适的硬件和做其他决定。
- 可以估计单个表的大小，以便更好地管理该表将使用的磁盘空间。创建表时设置合适的存储参数，就可以改善使用该表的应用的 I/O 性能。例如，假设在创建表之前估计了表的最大大小，然后，在创建表时设置了存储参数，就会为表的数据段分配很少的盘区，表的所有数据都被存储在相关的连续的磁盘空间扇区。这就减少了涉及这个表的磁盘 I/O 操作所需的时间。

无论是否在创建表之前估计过表的大小，都可以在创建每个表时明确地设置存储参数。（簇表，在第 18 章“管理簇”中讨论，自动使用簇的存储参数）。在创建表或后续的表更改时没有明确地设置的任何存储参数，都自动地使用该表所在的表空间的相应的默认存储参数设置。存储参数在第 14 章的“设置存储参数”中讨论。

如果明确地为表的数据段的盘区设置存储参数，应尽量将表的数据存储在少量的大盘区中而不是大量的小盘区中。

15.1.7 规划大表

对表和盘区的物理大小是没有限制的。可以将关键词 `UNLIMITED` 赋给 `MAXEXTENTS`，来简单地规划大表，减少消耗的空间和碎片，改善空间的再使用。但是，当表的盘区数量增长到很大时，在那个表上执行任何操作都会感觉到其对性能的影响。

注意：不能更改数据字典表来使 `MAXEXTENTS` 大于允许的数据库块的最大值。

如果数据库中有大表，请考虑如下建议：

- 将表和索引分开

将索引放到与其他对象不同的表空间，可能的话放到不同的磁盘。如果非要撤消并重建一个大表的索引（如停用和启动一个约束，或重建表），已隔离到不同的表空间的索引，就能比将索引安排到包含其他对象的表空间来说更容易找到连续的空间。

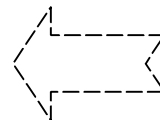
- 分配足够的临时空间

如果在一个很大的表中访问数据的应用执行大的排序，应保证足够的空间以用于大的临时段（临时段总是使用其表空间的默认的 `STORAGE` 设置）。

15.1.8 表的限制

创建表之前应注意如下的限制：

- 包含对象类型的表不能导入到 Oracle8 以前的数据库。
- 当原始数据还在数据库中时，不能将类型和盘区表移动到一个不同的模式中。
- 不能将一个导出的表合并到一个在不同模式中的具有相同名字的早已存在的表中。
- Oracle 对表所能具有的列数（或对象类型的属性）有限制。



进一步，当创建一个包含用户定义类型的数据的表时，Oracle 将用户定义类型映射成相关的列，以便存储用户定义类型的数据。这导致创建了另外的相关的列。这产生了在 DESCRIBE 表语句中不可见的，并且在 SELECT * 语句中不能返回的“隐藏”的相关的列。因此，当创建对象表、或具有 REF 列、varray、嵌入表、或对象类型的关系表时，应注意到 Oracle 实际为该表所创建的列的总数可能比你所指定的要多。

15.2 创 建 表

要想在你的模式中创建新表，需要有 CREATE TABLE 系统权限。要想在其他用户的模式中创建新表，需要有 CREATE ANY TABLE 系统权限。另外，表的拥有者还必须具有包含该表的表空间的限额，或 UNLIMITED TABLESPACE 系统权限。

用 SQL 语句 CREATE TABLE 创建表。

15.2.1 创建一个表

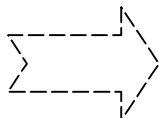
当用户 scott 发布如下语句时，就在他的模式中创建了一个名为 emp 的表，并将其存储在 user 表空间：

```
CREATE TABLE emp (  
  empno NUMBER(5) PRIMARY KEY,  
  ename VARCHAR2(15) NOT NULL,  
  job VARCHAR2(10),  
  mgr NUMBER(5),  
  hiredate 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，在表的几个列上定义了完整性约束。完整性约束在第 21 章的“管理完整性约束”中讨论。还为该表明确地指定了几个段属性。这些在第 14 章的“管理模式对象的空间”中讨论。

15.2.2 创建临时表

可以创建一个临时表。所有会话都可以看到临时表的定义，但只有将数据插入该表中



的会话才能看见临时表中的数据。用 CREATE GLOBAL TEMPORARY TABLE 语句来创建临时表。ON COMMIT 关键词指定是否表中的数据是事务特殊的（默认）或会话特殊的：

- ON COMMIT DELETE ROWS 指定临时表是事务特殊的。每次提交之后，Oracle 都舍弃表（删除所有行）。
- ON COMMIT PRESERVE ROWS 指定临时表是会话特殊的。结束会话时，Oracle 才舍弃表。

下面的例子创建一个事务特殊的临时表：

```
CREATE GLOBAL TEMPORARY TABLE work_area
(startdate DATE,
enddate DATE,
class CHAR(20))
ON COMMIT DELETE ROWS;
```

可以在临时表上创建索引。这些索引也是临时的，并且索引中的数据与其所基于的表的数据具有相同的会话或事务作用域。

15.2.3 并行创建表

如果在创建表时指定 AS SELECT 子句，就可以利用并行执行。CREATE TABLE AS SELECT 语句包括两部分：CREATE 部分（DDL）和 SELECT 部分（查询）。Oracle 能使该语句的两部分并行执行。如果下面的条件之一成立，则 CREATE 部分就是并行的。

- 在 CREATE TABLE AS SELECT 语句中包含了 PARALLEL 子句
- 指定了 ALTER SESSION FORCE PARALLEL DDL 语句

如果下面的所有条件成立，则查询部分就是并行的：

- 查询中包含了一个并行暗示说明(PARALLEL 或 PARALLEL_INDEX), 或 CREATE 部分包含 PARALLEL 子句，或查询所涉及的模式对象具有和它们相关的 PARALLEL 声明。
- 至少在一个查询中指定的表需要完整的表扫描或索引范围扫描跨越多个分区。

如果并行创建表，那么，该表就具有与其相关的并行声明（PARALLEL 子句）。任何后续的、可能并行执行的对该表的 DML 或查询就将尽量使用并行执行。

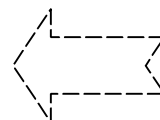
下面的简单例子并行创建表：

```
CREATE TABLE emp_dept
PARALLEL
AS SELECT * FROM scott.emp
WHERE deptno = 10;
```

该例子中的 PARALLEL 子句告诉 Oracle，在创建表时选择一个优化的并行执行服务器的个数。

15.2.4 自动收集表上的统计数据

PL/SQL 的包 DBMS_STATS 可以产生和管理用于基于开销的优化方法的统计数据。可



以使用这个包来收集、修改、查看、导出、导入和删除统计数据。还可以使用这个包来标识或命名收集到的统计数据。

通过在 CREATE（或 ALTER）TABLE 语句中指定 MONITORING 子句来使 DBMS_STATS 自动收集统计数据。那么就可以通过，如设置一个带有 GATHER STALE 选项的调用 DBMS_STATS.GATHER_TABLE_STATS 的递归作业（或许要靠使用作业队列），某些方法在合适的期间为应用自动收集统计数据。

监控进程跟踪自从上次收集统计数据以来对表的 INSERT、UPDATE 和 DELETE 操作的大约数量。有关有多少行受到影响的信息是在 SGA 中维持的，直到 SMON 周期性地（大约每 3 个小时）将这些数据合并到数据字典中。可以通过 DBA | ALL | USER_TAB_MODIFICATIONS 视图来查看这个数据字典的信息。Oracle 用这个视图来识别具有陈旧统计数据的表。

用 MONITORING 子句和 DBMS_STATS 包可使优化器产生精确的执行方案，而不需要运行传统和费事的 ANALYZE 语句来识别已被修改过的表。

为了停用对表的监控，可以指定 NOMONITORING 子句。

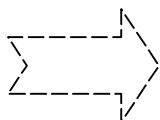
15.3 更 改 表

要想更改表，该表必须保存在你的模式中，或者你必须具有对该表的 ALTER 对象权限或者 ALTER ANY TABLE 系统权限。

因为如下的原因，可以更改 Oracle 数据库中的表：

- 为了添加或删除列，或修改现有的列的定义（数据类型、长度、默认值和 NOT NULL 完整性约束）
- 为了修改数据块空间的使用参数（PCTFREE、PCTUSED）
- 为了修改事务入口的设置（INITRANS、MAXTRANS）
- 为了修改存储参数
- 为了将表移动到一个新段或表空间
- 为了明确地分配一个盘区或收回一个未使用的空间。
- 为了修改表的日志属性
- 为了修改 CACHE/NOCACHE 属性
- 为了添加、修改或删除与表相关的完整性约束
- 为了启动或停用与表相关的完整性约束或触发器
- 为了修改表的并行度
- 为了重命名一个表
- 为了添加或修改索引结构表的特征
- 为了添加或修改 LOB 列
- 为了添加或修改对象类型、嵌入的表、或 Varray 列
- 为了启动或停用统计数据采集（MONITORING/NOMONITORING）

如果当前存在的数据满足新的长度，就可以增加或减少当前存在的列的长度。可以将



一个列从字节语义改变成 CHAR 语义，或反之。为了减少非空 CHAR 列的长度，必须设置初始参数 BLANK_TRIMMING=TURE。

当修改一个表以增加 CHAR 数据类型的列的长度时，要知道，这可能是一个费时的操作，还可能需充足的附加存储空间，特别是当该表包含很多行时。这是因为每行中的 CHAR 值都必须在其后添加空格以满足新的列长度。

当更改表的数据块空间使用参数（PCTFREE 和 PCTUSED）时，要注意，新的设置适用于该表所使用的所有数据块，包括给该表已经分配的和以后分配的数据块。但是，当更改空间的使用参数时，已经给表分配的数据块不会立即重组，而在需要时才重组。数据块存储参数在第 14 章的“管理数据块空间”中描述。

当更改表的事务入口设置（INITRANS、MAXTRANS）时，要注意，对 INITRANS 的新设置仅适用于以后给表分配的数据块，而对 MAXTRANS 的新设置适用于所有数据块（已经和以后分配的数据块）。为了更好地理解这些事务入口的参数设置，参见第 14 章的“指定事务入口参数：INITRANS 和 MAXTRANS”。

不能更改存储参数 INITIAL 和 MINEXTENTS。其他存储参数（如 NEXT、PCTINCREASE）的所有新设置仅适用于以后给该表分配的盘区。下一个被分配的盘区的大小取决于 NEXT 和 PCTINCREASE 的当前值而不是这些参数以前的值。存储参数在第 14 章的“设置存储参数”中讨论。

用 ALTER TABLE 语句更改表。下面的语句更改 emp 表。它更改数据块存储参数并添加一个名为 bonus 的列。

```
ALTER TABLE emp
ADD (bonus NUMBER (7,2))
PCTFREE 30
PCTUSED 60;
```

ALTER TABLE 语句的其他用途是：

- 将表移动到一个新段或表空间
- 手动给表分配存储空间
- 删除列

警告：在更改表之前，自己必须清楚这样做的结果。

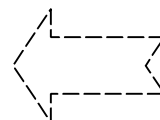
如果给表添加一个新列时，列的初值是空的。仅当表中不包含任何行时才能给表添加一个具有 NOT NULL 约束的列。

如果一个视图或 PL/SQL 程序单元依赖于一个基本表，更改该基本表会影响依赖的对象。参见第 21 章的“管理对象依赖性”中有关 Oracle 是如何管理依赖性的信息。

15.3.1 将表移动到一个新段或表空间

可以使用 ALTER TABLE A MOVE 语句将非分区表的数据重新放置到一个新段，还可以放置到具有有限额的不同的表空间。这个语句还可以修改表的任何存储属性，包括那些不能用 ALTER TABLE 语句修改的存储属性。

如下语句将 emp 表移动到一个新段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 列，这个语句可用于将表连同用户明确指定的 LOB 数据和 LOB 索引段（与该表相关的）一起移动。如果没有指定，则默认不移动 LOB 数据和 LOB 索引段。

15.3.2 手动给表分配存储空间

当需要时，Oracle 动态地给表的数据段分配附加的盘区。但是，也许想要明确地给表分配附加的盘区。例如，在 Oracle 真实应用簇环境中，可以为一个特定的实例明确地给表分配一个盘区。

可以用 ALTER TABLE 语句的 ALLOCATE EXTENT 子句给表分配新的盘区。

还可以用带有 DEALLOCATE UNUSED 子句的 ALTER TABLE 语句明确地收回未使用的空间。这在第 14 章的“回收空间”中讨论。

15.3.3 删除列

可以从表（包括索引结构表）中删除不再需要的列。这提供了一个释放数据库中的空间的方便手段，同时避免了不得不导出/导入数据，然后再重建索引和约束的工作。为了能发布下面与删除列相关的任何语句，用户需要在目标表上具有 ALTER 权限或 ALTER ANY TABLE 系统权限。

不能删除表中的所有列，也不能删除 SYS 拥有的表的列。如果这样做就产生错误。

15.3.3.1 从表中删除列

如果发布 ALTER TABLE A DROP COLUMN 语句，则列的描述符和与目标列相关的数据将被从表中的每行删除。可以用一个语句删除多个列。下面的语句是从 emp 表中删除列的例子。

如下语句只删除 sal 列：

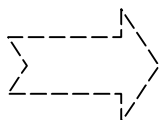
```
ALTER TABLE emp DROP COLUMN sal;
```

如下语句删除 sal 和 comm 两个列：

```
ALTER TABLE emp DROP (sal, comm);
```

15.3.3.2 将列标记成未使用

如果在意从一个大表中删除所有行的列数据所花费的时间，可以使用 ALTER TABLE A SET UNUSED 语句。该语句将一个或多个列标记成未使用，而不会真的将目标列数据删除或恢复这些列所占据的磁盘空间。然而，被标记为未使用的列在查询中或数据字典视图中不会显示出来，它的名字被删除了，因此，新的列可以重用该名字。所有定义在该列上的约束、索引和统计数据也被删除了。



为了将 `sal` 和 `comm` 列标记成未使用，执行如下语句：

```
ALTER TABLE emp SET UNUSED (sal, comm);
```

以后，可以发布 `ALTER TABLE A DROP UNUSED COLUMNS` 语句来删除被标记为未使用的列。无论何时发布一个明确的删除表的特定的列，未使用的列也将从目标表中被删除。

数据字典视图 `USER_UNUSED_COL_TABS`、`ALL_UNUSED_COL_TABS` 或 `DBA_UNUSED_COL_TABS` 可用于列出所有的包含未使用列的表。`COUNT` 域显示了表中未使用列的个数。

```
SELECT * FROM DBA_UNUSED_COL_TABS;
OWNER TABLE_NAME COUNT
```

```
-----
```

```
SCOTT EMP 1
```

```
1 row selected.
```

15.3.3.3 删除未使用的列

`ALTER TABLE A DROP UNUSED COLUMNS` 语句是唯一允许在未使用列上进行的操作。它从表中物理地删除未使用列，并且收回磁盘空间。

在如下例子中指定了可选的关键词 `CHECKPOINT`。这个选项使得在处理了指定数量的行（本例为 250）后产生一个检查点。在删除列的操作期间，检查点减少重做日志积累的空间，以免潜在地耗尽回滚段空间。

```
ALTER TABLE emp DROP UNUSED COLUMNS CHECKPOINT 250;
```

15.4 联机重定义表

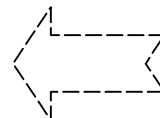
在高可用系统中，偶尔需要重新定义大的“热”表，以改善针对这些表的查询或 DML 执行的性能。Oracle 提供了一个联机重定义表的机制。这个机制相对于传统的需要将表脱机后才能重定义的方法而言，明显地提高了可用性。

联机定义表时，在许多重定义处理期间是可以对 DML 进行访问的。表仅在一个很小的时间窗期间以排它方式加锁，这个时间窗与表的大小和重定义的复杂性无关。

联机重定义表可以：

- 修改表的存储参数
- 将表移动到相同模式中的不同的表空间
- 添加对并行查询的支持
- 添加或删除分区支持
- 重建表以减少碎片
- 将普通表（堆积结构的）的结构改变成索引结构表，或反之
- 添加或删除列

执行联机重定义的机制是 PL/SQL 包 `DBMS_REDEFINITION`。这个包的执行权限被授予 `EXECUTE_CATALOG_ROLE`。除了具有这个包的执行权限之外，还必须被授予如下权



限：

- CREATE ANY TABLE（创建任何表）
- ALTER ANY TABLE（更改任何表）
- DROP ANY TABLE（删除任何表）
- LOCK ANY TABLE（加锁任何表）
- SELECT ANY TABLE（选取任何表）

重定义的过程有几个步骤。

15.4.1 表的联机重定义的步骤

为了进行表的联机重定义，用户必须执行如下几步。

①调用 DBMS_REDEFINITION.CAN_REDEF_TABLE（）过程，确认该表是可以联机重定义的。如果该表不适于联机重定义，这个过程就产生一个错误，表明该表为何不能被联机重定义。

②创建一个具有所有所需属性的空的临时过渡表（在要被重定义的表的相同模式中）。

③调用 DBMS_REDEFINITION.START_REDEF_TABLE（），启动重定义进程，这需要提供：

- 要被重定义的表
- 临时过渡表的名字
- 列的映射

如果没有提供列的映射信息，则是假设所有的列（列名不变）都包含在临时过渡表中。如果提供了列的映射，则仅只考虑在列的映射中被明确指定的那些列。

④在临时过渡表上创建触发器、索引、授权和约束。涉及临时过渡表（即临时过渡表是参照约束的父或子表）的任何参照约束必须按停用状态创建。在重定义进程结束或终止之前，在临时过渡表上的任何触发器都不会执行。

重定义完成后，与临时过渡表相关的触发器、约束、索引和授权将代替正被重定义的表上的触发器、约束、索引和授权，涉及临时过渡表的参照约束（按停用状态创建），将被转换到正被重定义的表上，并且在完成重定义之后变成启用状态。

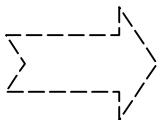
⑤执行 DBMS_REDEFINITION.FINISH_REDEF_TABLE（）过程，完成表的重定义。在这个过程期间，以排它模式给原始表加锁一会（非常小的时间窗口）。这个时间窗口不依赖于原始表中数据量的大小。同时作为这个过程的一部分，将会：

- a. 重定义原始表，使其具有临时过渡表的所有属性、索引、约束、授权和触发器。
- b. 包含临时过渡表的参照约束现在包含最终的被重定义的表并被启用。

⑥在第 4 步中，可选择修改被创建到临时过渡表中的任何索引名称。现在这些索引已被定义到重定义表中了。

重定义过程的最终结果是：

- 原始表按照临时过渡表的属性和特点被重定义。
- 在 START_REDEF_TABLE（）和 FINISH_REDEF_TABLE（）期间在临时过渡表上定义的触发器、授权、索引和约束现在已被定义到被重定义之后的表上了。在重



定义过程结束之前任何包含临时过渡表的参照约束，现在包含到被重定义之后的表了并被启用。

- 在原始表（重定义之前的表）上定义的任何索引、触发器、授权和约束被转换到临时过渡表了，当用户删除临时过渡表时会将它们删除。在重定义之前的任何包含原始表的参照约束现在包含到临时过渡表并被停用。
- 在原始表（重定义之前的表）上定义的任何 PL/SQL 过程和游标都无效了。只要下次使用它们，它们都会自动重新生效（如果重定义过程导致表的形式被改变了，这个重新生效可能会失败）。

15.4.2 中间同步

在调用 `START_REDEF_TABLE()` 启动重定义过程和调用 `FINISH_REDEF_TABLE()` 结束重定义过程期间，可以在原始表上执行大量的 DML 语句。如果这是事实，建议周期性地使临时过渡表与原始表同步。可以调用 `DBMS_REDEFINITION.SYNC_INTERIM_TABLE()` 过程来做这件事。调用这个过程可以减少为完成重定义过程而调用 `FINISH_REDEF_TABLE()` 时所花的时间。

在 `FINISH_REDEF_TABLE()` 期间原始表被加锁的那小段时间与是否调用过 `SYNC_INTERIM_TABLE()` 无关。

15.4.3 出错后的中止和清除

在重定义的过程中，如果出现错误或想终止重定义过程，就调用 `DBMS_REDEFINITION.ABORT_REDEF_TABLE()`。该过程删除与重定义进程相关的临时日志和表。调用该过程之后，用户可以删除临时过渡表及其相关的对象。

15.4.4 联机重定义表的例子

该例子描述了具有列：empno, name, salary, phone 的非分区表 emp 的联机重定义。该表被重定义成：

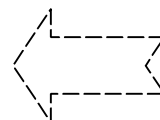
- 列 salary 乘以系数 1.10 并重命名为 sal。
- 删除 phone 列。
- 添加默认值为 10 的一个新的 deptno 列。
- 重定义的表按 empno 列的范围分区。

假设已经运行了 `DBMS_REDEFINITION.CAN_REDEF_TABLE()` 过程，并且表 emp 可以被重定义。

该重定义的步骤如下所述：

①创建临时过渡表 int_emp。

```
CREATE TABLE int_emp
(empno NUMBER PRIMARY KEY,
name VARCHAR2(100),
sal NUMBER,
deptno NUMBER DEFAULT 10)
```



```
PARTITION BY RANGE(empno)
(PARTITION emp1000 VALUES LESS THAN (1000) TABLESPACE tbs_1,
PARTITION emp2000 VALUES LESS THAN (2000) TABLESPACE tbs_2);
```

②启动重定义进程。

```
DBMS_REDEFINITION.START_REDEF_TABLE('u1', 'emp', 'int_emp',
'empno empno, name name, salary*1.10 sal');
```

③在 `int_emp` 上创建触发器、索引和约束。在重定义的最后一步中，这些将被转换回原始表。在 `int_emp` 上的任何参照约束应被停用。可以定义与临时过渡表相关的任何授权。重定义之后，这些授权将取代原始表上的授权。

④可选地，同步临时过渡表 `int_emp`。

```
DBMS_REDEFINITION.SYNC_INTERIM_TABLE('u1', 'emp', 'int_emp');
```

⑤完成重定义。

```
DBMS_REDEFINITION.FINISH_REDEF_TABLE('u1', 'emp', 'int_emp');
```

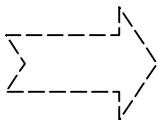
在这一步的后期，安排它模式将表 `emp` 加锁一小段窗口。这个调用之后，表 `emp` 被重定义，它具有了 `int_emp` 表的所有属性

⑥删除临时过渡表。

15.4.5 限制

下面的限制适用于表的联机重定义：

- 表必须有主键才能被联机重定义。
- 将被重定义的表和最终被重定义的表必须有相同的主键列。
- 在其上定义过物化视图和物化视图日志的表不能被联机重定义。
- 被物化成视图容器表的表和高级队列表不能被联机重定义。
- 索引结构表的溢出表不能被联机重定义。
- 具有用户定义类型（对象、REF、收集、类型化表）的表不能被联机重定义。
- 具有 FILE 列的表不能被联机重定义。
- 具有 LONG 列的表不能被联机重定义。具有 LOB 列的表还可以。
- 要重定义的表不能是簇的一部分。
- SYS 和 SYSTEM 模式中的表不能被联机重定义。
- 临时表不能被重定义。
- 没有同一级别的子设置值支持。
- 在将临时过渡表中的列映射到原始表中的列时，只能使用简单的确切的表达式。例如，不允许使用子查询。
- 如果在重定义时添加新列（不能用原始表中的现存数据来生成），在重定义完成之前，不能将其声明为 NOT NULL。
- 在正被重定义的表和临时过渡表之间没有任何参照约束。



15.5 删 除 表

要删除表，该表必须保存在你的模式中或你必须具有 **DROP ANY TABLE** 系统权限。

要删除不再需要的表，可使用 **DROP TABLE** 语句。如下语句删除 **emp** 表。

```
DROP TABLE emp;
```

如果要删除的表包含其他表的外键所参照的任何主键或唯一键，你又打算删除子表中的外键约束 (**FOREIGN KEY**)，请在 **DROP TABLE** 语句中包含 **CASCADE** 选项，如：

```
DROP TABLE emp CASCADE CONSTRAINTS;
```

警告：在删除表之前，必须清楚这样做的后果

- 删除表就会从数据字典中删除该表的定义。就再也不能访问该表的任何行了。
- 与表相关的所有索引和触发器也被删除。
- 依赖于被删除表的所有视图和 **PL/SQL** 程序单元仍保留，但已无效（不可使用了）。参见第 21 章的“管理对象依赖性”中有关 Oracle 是如何管理依赖性的信息。
- 被删除表的所有同义词仍保留，但使用时会返回错误。
- 已经分配给被删除表的所有盘区都划归表空间的空闲空间，当任何其他对象需要新盘区或新对象时就能使用它。对应于簇表的所有行被从簇的数据块中删除，簇表是第 18 章“管理簇”的内容。

也许你想用舍弃表来代替删除表。**TRUNCATE** 语句提供了一个快速、有效的方法来从表中删除所有行，但它不影响与该正被舍弃的表相关的任何结构（列定义、约束、触发器等等）或授权。**TRUNCATE** 语句在第 21 章的“舍弃表和簇”中讨论。

15.6 管理索引结构表

本节描述管理索引结构表，包括如下内容：

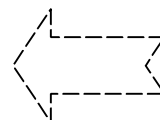
- 什么是索引结构表
- 创建索引结构表
- 维护索引结构表
- 分析索引结构表
- 随同索引结构表使用 **ORDER BY** 语句
- 将索引结构表转换成普通表

15.6.1 什么是索引结构表

索引结构表具有一个是主 **B-树** 的变形的存储结构。与按末排序的集合（堆积）存储数据的普通的（堆积结构的）表不同，索引结构表的数据按主键排序手段被存储在 **B-树** 索引结构中。除了存储索引结构表的行的主键列的值之外，在 **B-树** 中的每个索引项还存储非键列的值。

15.6.1.1 为何使用索引结构表

索引结构表为包含精确匹配和范围搜索的查询提供了快速的基于键的对表中数据的访



问。对表中数据的修改（如加新行、更改行或删除行）仅导致更改索引结构（因为没有单独的表存储区域）。

同时，因为没有在表和索引中复制键列，所以也就减少了存储需求。剩下的非键列被存储在索引结构中。

当使用必须基于主键以获取数据的应用时，索引结构表特别有用。索引结构表也适用于模块化应用特殊的索引结构。如，基于内容的包含文本、图像和音响数据的信息检索应用需要已逆转的索引，用索引结构表就能有效地模块化这种索引。

15.6.1.2 索引结构表和普通表之间的差异

如图 15-1 所示，索引结构表有些象是由一个普通表和一个在一个或多个表列上的索引所组成的结构，但不是维护两个独立的存储结构（一个用于表，一个用于 B-树索引），数据库系统仅仅是维护单个 B-树索引。同时，不仅将行的 rowid（行标识）存在索引项中，而且还保存了非键列的值。因此，每个 B-树索引项包含：

<primary_key_value, non_primary_key_column_values>。

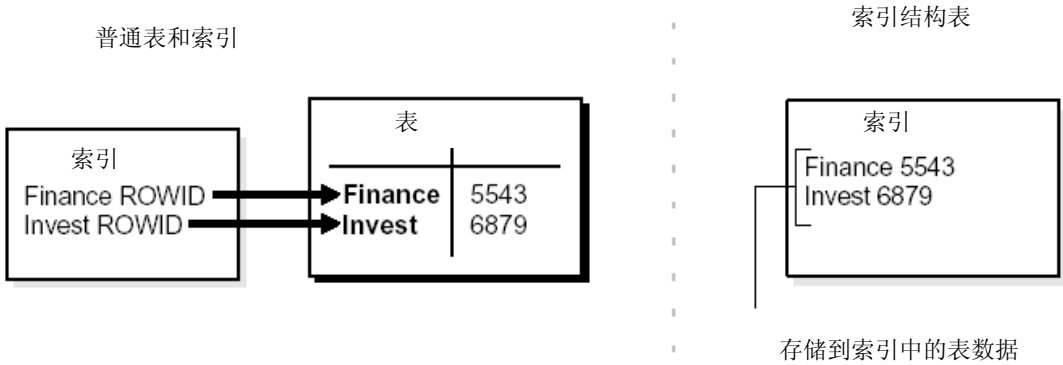


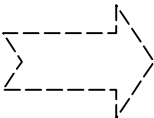
图 15-1 普通表的结构与索引结构表的结构对比

应用管理索引结构表就像管理普通表那样，使用 SQL 语句。但数据库系统是靠管理对应的 B-树索引来执行所有操作的。

15.6.2 创建索引结构表

使用 CREATE TABLE 语句创建索引结构表，但还需提供如下附加信息：

- ORGANIZATION INDEX 限定词，它表示这是一个索引结构表。
- 主键是通过列约束子句（单个列主键时）或表约束子句（多个列主键时）指定的。必须给索引结构表指定主键。
- 可选的行溢出说明子句（OVERFLOW），它依靠将超过指定门限值的行的列值存储到一个独立的溢出数据段来保存 B-树索引的分簇密度。INCLUDING 子句也能被指定识别什么（没有关键字）列是被保存在溢出数据段中。
- PCTTHRESHOLD 值，它定义在索引数据块中为索引结构表保留的空间的百分比。超过指定门限值的行的任何部分都被存储到溢出段。换句话说，行在一个列的边界上被分成两段，头段和尾段。头段满足指定的门限值并与键一起存储在索引叶子数



据块中。尾段按一个或多个行段被存储在溢出区域中。因此，索引项包含键值、满足指定门限值的非键列值、和指向剩余行的指针。

下面的例子创建一个索引结构表：

```
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
OVERFLOW TABLESPACE ovf_tbs;
```

上面的例子显示，ORGANIZATION INDEX 限定词指定了索引结构表，键列和非键存在于一个索引中，该索引定义在为表指明主键 (token, doc_id) 的列上。

索引结构表可以存储对象类型。下面的例子创建一个包含对象类型 mytype 的列的索引结构表：

```
CREATE TABLE iot (c1 NUMBER primary key, c2 mytype)
ORGANIZATION INDEX;
但不能创建对象类型的索引结构表。如，下面的语句无效：
CREATE TABLE iot OF mytype ORGANIZATION INDEX;
```

15.6.2.1 使用 AS 子查询

可以用 AS 子查询创建索引结构表。用这种方式创建索引结构表时，可用 PARALLEL 选项并行地装载表。

下面的语句利用从常规表 rt 中选出的行创建（并行地）一个索引结构表：

```
CREATE TABLE iot(i PRIMARY KEY, j) ORGANIZATION INDEX PARALLEL (DEGREE 2)
AS SELECT * FROM rt;
```

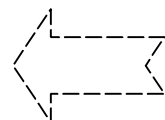
15.6.2.2 使用 Overflow 子句

前面例子中的 Overflow 子句表示，超过数据块大小的 20% 的行的任何非键列都将被放置到 OVF_TBS 表空间中的数据段中。键列应该满足指定的门限值。

如果更新非键列，使行减小大小，Oracle 会识别适合对其进行更新的段（头或尾），并重写那个行段。

如果更新非键列，使行增加大小，Oracle 会识别适合对其进行更改的段（头或尾），并重写那个行段。如果更改的目标是头段，注意，这个段还可以再次被分成两段以保持行的大小低于指定的门限值。

适合索引叶子数据块的非键列被按行的头段存储，该头段包含一个 ROWID 域，该域将该段连接到存储在溢出数据段中的下一个行段。存储在溢出区域中的列是那些不适合的



列。

15.6.2.3 选择和监测门限值

最好选择一个可以容纳键列和头几个非键列（如果常常访问）的门限值。

选择好门限值之后，可以监测各个表以确认所指定的值是合适的。可以用 `ANALYZE TABLE...LIST CHAINED ROWS` 语句来确定超过门限值的行数和行的标识。

15.6.2.4 使用 INCLUDING 子句

除了指定 `PCTTHRESHOLD` 之外，还可用 `INCLUDING` 子句来控制哪些非键列要与键列存储在一起。Oracle 在索引叶子数据块中容纳所有的直到在 `INCLUDING` 子句中指定的非键列，只要它不超过指定的门限值。超过在 `INCLUDING` 子句中指定的非键列被存储在溢出区域。

注意：Oracle 将索引结构表的所有主键列移动到表的前端（按键排序），以便提供高效的基于主键的访问。如：

```
CREATE TABLE io(a INT, b INT, c INT, d INT,
primary key(c,b))
ORGANIZATION INDEX;
```

存储后的列的排序是：**c b a d**（而不是：**a b c d**）。基于存储后的列的排序，最后的主键列是 **b**。`INCLUDING` 列可以是最后的主键列（本例是 **b**），或任何非键列（即，在存储后的列的排序中，**b** 之后的任何列）。

可以修改前面的例子，来创建一个索引结构表，其 `token_offsets` 列的值总是存储在溢出区域：

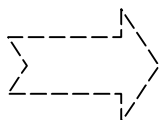
```
CREATE TABLE docindex(
token CHAR(20),
doc_id NUMBER,
token_frequency NUMBER,
token_offsets VARCHAR2(512),
CONSTRAINT pk_docindex PRIMARY KEY (token, doc_id))
ORGANIZATION INDEX TABLESPACE ind_tbs
PCTTHRESHOLD 20
INCLUDING token_frequency
OVERFLOW TABLESPACE ovf_tbs;
```

此时，仅仅直到 `token_frequency` 列（在本例中仅只有一个列）的非键列才与键列的值一起被存储在索引叶子数据块中。

15.6.2.5 使用键压缩

用键压缩方法创建索引结构表能够消除键列前缀值的重复出现。

键压缩将索引键分成前缀和后缀项。压缩是靠共享索引数据块中的所有后缀项的前缀项来实现的。该共享能节省很多空间，在改善性能的同时允许在每个索引数据块中存储更



多的键。

可以用 COMPRESS 子句启用键压缩，当：

- 创建索引结构表时
- 移动索引结构表时

还可以指定前缀的长度（就像键列的个数），它表明如何将键列分成前缀和后缀项。

```
CREATE TABLE iot(i INT, j INT, k INT, l INT, PRIMARY KEY (i, j, k))
ORGANIZATION INDEX COMPRESS;
```

上面的语句与下面的语句等效：

```
CREATE TABLE iot(i INT, j INT, k INT, l INT, PRIMARY KEY(i, j, k))
ORGANIZATION INDEX COMPRESS 2;
```

对于值 (1, 2, 3), (1, 2, 4), (1, 2, 7), (1, 3, 5), (1, 3, 4), (1, 4, 4) 的列表而言，重复出现的 (1, 2), (1, 3) 就被压缩掉了。

可以覆盖用于压缩的默认的前缀长度，如

```
CREATE TABLE iot(i INT, j INT, k INT, l INT, PRIMARY KEY (i, j, k))
ORGANIZATION INDEX COMPRESS 1;
```

对于值 (1, 2, 3), (1, 2, 4), (1, 2, 7), (1, 3, 5), (1, 3, 4), (1, 4, 4) 的列表而言，重复出现的 1 就被压缩掉了。

可以停用压缩，如：

```
ALTER TABLE A MOVE NOCOMPRESS;
```

15.6.3 维护索引结构表

索引结构表仅仅只是在物理结构上不同于普通的表，在逻辑上被以相同的方式管理。在 INSERT, SELECT, DELETE 和 UPDATE 语句中可以用索引结构表代替普通表。

15.6.3.1 更改索引结构表

可以用 ALTER TABLE 语句更改主键索引和溢出数据段两者的物理和存储属性。在 OVERFLOW 关键词之前指定的所有属性都适用于主键索引段。在 OVERFLOW 关键词之后指定的所有属性都适用于溢出数据段。例如，可以像下面这样，将主键索引段的 INITRANS 设为 4，将溢出数据段的 INITRANS 设为 6：

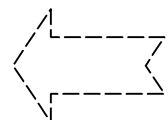
```
ALTER TABLE docindex INITRANS 4 OVERFLOW INITRANS 6;
```

还可以更改 PCTTHRESHOLD 和 INCLUDING 列的值。在后续操作中，新的设置被用于将行分成头段和溢出尾段。例如，可以像下面这样，更改 DOCINDEX 表的 PCTTHRESHOLD 和 INCLUDING 列的值：

```
ALTER TABLE docindex PCTTHRESHOLD 15 INCLUDING doc_id;
```

通过将 INCLUDING 列设为 doc_id, token_frequency 和 token_offsets 之后的所有列都被存储到溢出数据段中了。

如果被创建的索引结构表没有溢出数据段，就可以用 ADD OVERFLOW 子句来加一个溢出数据段。例如，如果 DOCINDEX 表确实没有溢出数据段，那么，可以像下面一样加



一个溢出数据段：

```
ALTER TABLE docindex ADD OVERFLOW TABLESPACE ovf_tbs;
```

15.6.3.2 移动（重建）索引结构表

因为索引结构表主要按 B-树索引存储，所以，更改多了就使得碎片也多了。但可以用 ALTER TABLE……MOVE 语句来重建索引和减少这些碎片。

下面的语句在将其 INITRANS 设为 10 之后，重建索引结构表 DOCINDEX：

```
ALTER TABLE docindex MOVE INITRANS 10;
```

可以使用 ONLINE 选项来联机移动没有溢出数据段的索引结构表。例如，如果 DOCINDEX 表没有溢出数据段，就可以像下面一样联机执行移动

```
ALTER TABLE docindex MOVE ONLINE INITRANS 10;
```

下面的语句重建索引结构表 DOCINDEX 及其溢出数据段：

```
ALTER TABLE docindex MOVE TABLESPACE ix_tbs OVERFLOW TABLESPACE ov_tbs;
```

在这最后一个语句中，移动索引结构表 IOT，同时重建用于 C2 的 LOB 索引和数据段：

```
ALTER TABLE iot MOVE LOB (C2) STORE AS (TABLESPACE lob_ts);
```

15.6.3.3 方案：更新键列

从逻辑上讲，更新键列等效于删除具有旧键值的行，然后在适当的位置以新键值插入行以维持主键的排序。

逻辑上，在如下例子中，删除了 dept_id = 20 和 e_id = 10 的职员行，同时，插入了 dept_id = 23 和 e_id = 10 的职员行。

```
UPDATE employees
SET dept_id=23
WHERE dept_id=20 and e_id=10;
```

15.6.4 分析索引结构表

正如普通表一样，使用 ANALYZE 语句来分析索引结构表：

```
ANALYZE TABLE docindex COMPUTE STATISTICS;
```

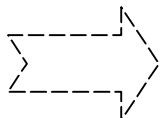
ANALYZE 语句分析主键索引段和溢出数据段两者，并且计算该表的逻辑和物理统计数据。

- 可以用 USER_TABLES, ALL_TABLE 或 DBA_TABLES 来查询逻辑统计数据。
- 可以用 USER_INDEXES, ALL_INDEXES 或 DBA_INDEXES (并使用主键索引名) 来查询主键索引段的物理统计数据。例如，可以像下面一样，获得表 docindex 的主键索引段的物理统计数据：

```
SELECT * FROM DBA_INDEXES WHERE INDEX_NAME= 'PK_DOCINDEX';
```

- 可以用 USER_TABLES, ALL_TABLES 或 DBA_TABLES 来查询溢出数据段的物理统计数据。可以用查找 IOT_TYPE='IOT_OVERFLOW' 来识别溢出项。例如，可以像下面一样，获得与 DOCINDEX 表相关的溢出数据段的物理属性：

```
SELECT * FROM DBA_TABLES WHERE IOT_TYPE='IOT_OVERFLOW'
```



```
and IOT_NAME= 'DOCINDEX';
```

15.6.5 随同索引结构表使用 ORDER BY 子句

如果 ORDER BY 子句仅仅参照主键列或其前缀，那么，只要返回的行在主键上已排序了，优化器就会避免排序的开销。

例如，创建下面的表：

```
CREATE TABLE employees (dept_id INTEGER, e_id INTEGER, e_name
    VARCHAR2, PRIMARY KEY (dept_id, e_id)) ORGANIZATION INDEX;
```

因为数据在主键上已经排序了，所以，下面的查询避免了排序的开销：

```
SELECT * FROM employees ORDER BY (dept_id, e_id);
SELECT * FROM employees ORDER BY (dept_id);
```

但是，如果让 ORDER BY 子句参照主键列的后缀或非键列，就需要额外的分类（假设没有定义其他的次要索引）。

```
SELECT * FROM employees ORDER BY (e_id);
SELECT * FROM employees ORDER BY (e_name);
```

15.6.6 将索引结构表转换成普通表

可以用 Oracle 的 IMPORT 或 EXPORT 工具，或 CREATE TABLE ... AS SELECT 语句，将索引结构表转换成普通表。

为将索引结构表转换成普通表，应该：

- 用常规的途径导出索引结构表的数据。
- 用相同的定义创建普通表的定义。
- 导入索引结构表的数据，要使 IGNORE=y（以保证忽略存在的对象错误）。

注意：在将索引结构表转换成普通表之前，要注意，索引结构表不能被 Oracle8 以前的版本的导出工具导出。

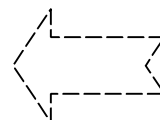
15.7 管理外部表

Oracle 允许以只读方式访问外部表中的数据。外部表是指不在数据库中的表，可能以某种方式为其提供了一个访问驱动程序。通过向 Oracle 提供描述外部表的元数据，Oracle 就可以显露外部表中的数据，就像这些数据存储在一个普通数据库表中一样。可以用 SQL 来直接和并行查询外部表的数据。

例如，可以选取、联结或排序外部表的数据。还可以为外部表创建视图和同义词。但在外部表上不可能执行 DML 操作（UPDATE，INSERT 或 DELETE），也不可能创建索引。

注意：ANALYZE 语句不支持采集外部表的统计数据，应该使用 DBMS_STATS 包来采集外部表的统计数据。

为外部表定义元数据的方法是使用 CREATE TABLE ... ORGANIZATION EXTERNAL 语句。该外部表的定义可以被想象成一个视图，该视图允许在外部数据上运行任何 SQL 查询，而不需要事先将外部数据装载进数据库。实际上用来读取表中的外部数据



的机制是访问驱动程序。

Oracle 为外部表提供了访问驱动程序。它允许使用 Oracle 的装载技术从外部文件中读取数据。ORACLE_LOADER 访问驱动程序提供了一个是 SQL*Loader 工具的控制文件句法的子集的数据映射能力。

Oracle 的外部表特性为在数据仓库中经常执行的基本的提取、转换、传输 (ETT) 任务提供了一个有价值的方法。

下面的几节讨论支持外部表的 DDL 语句。仅仅只是被讨论了的 DDL 语句才被支持而不是这些语句的所有子句都被支持。

- 创建外部表
- 更改外部表
- 删除外部表
- 用于外部表的系统权限和对象权限

15.7.1 创建外部表

使用 CREATE TABLE 语句的 ORGANIZATION EXTERNAL 子句创建外部表。实际上没有创建表；即，外部表没有任何与其相关的盘区。确切地说，是在数据字典中创建元数据以便访问外部数据。

下面的例子创建一个外部表，然后将数据装载到数据库表中。

15.7.1.1 例：创建外部表和装载数据

文件 empxt1.dat 包含如下样本数据：

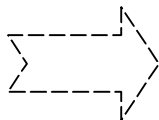
```
7369,SMITH,CLERK,7902,17-DEC-1980,800,0,20
7499,ALLEN,SALESMAN,7698,20-FEB-1981,1600,300,30
7521,WARD,SALESMAN,7698,22-FEB-1981,1250,500,30
7566,JONES,MANAGER,7839,02-APR-1981,2975,0,20
7654,MARTIN,SALESMAN,7698,28-SEP-1981,1250,1400,30
7698,BLAKE,MANAGER,7839,01-MAY-1981,2850,0,30
7782,CLARK,MANAGER,7839,09-JUN-1981,2450,0,10
```

...

文件 empxt2.dat 包含如下样本数据：

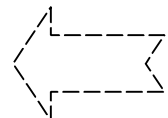
```
7788,SCOTT,ANALYST,7566,19-APR-1987,3000,0,20
7839,KING,PRESIDENT,,17-NOV-1981,5000,0,10
7844,TURNER,SALESMAN,7698,08-SEP-1981,1500,0,30
7876,ADAMS,CLERK,7788,23-MAY-1987,1100,0,20
7900,JAMES,CLERK,7698,03-DEC-1981,950,0,30
7902,FORD,ANALYST,7566,03-DEC-1981,3000,0,20
7934,MILLER,CLERK,7782,23-JAN-1982,1300,0,10
```

...



下面的 SQL 语句创建一个外部表并将其数据装载到数据库表 `scott.emp` 中。

```
SET ECHO ON;
CONNECT / AS SYSDBA;
CREATE OR REPLACE DIRECTORY dat_dir AS '/flatfiles/data';
CREATE OR REPLACE DIRECTORY log_dir AS '/flatfiles/log';
CREATE OR REPLACE DIRECTORY bad_dir AS '/flatfiles/bad';
GRANT READ ON DIRECTORY dat_dir TO scott;
GRANT WRITE ON DIRECTORY log_dir TO scott;
GRANT WRITE ON DIRECTORY bad_dir TO scott;
CONNECT scott/tiger;
DROP TABLE empxt;
CREATE TABLE empxt (empno NUMBER(4),
    ename VARCHAR2(10),
    job VARCHAR2(9),
    mgr NUMBER(4),
    hiredate DATE,
    sal NUMBER(7,2),
    comm NUMBER(7,2),
    deptno NUMBER(2)
)
ORGANIZATION EXTERNAL
(
    TYPE ORACLE_LOADER
    DEFAULT DIRECTORY dat_dir
    ACCESS PARAMETERS
    (
        records delimited by newline
        badfile bad_dir:'empxt%a_%p.bad'
        logfile log_dir:'empxt%a_%p.log'
        fields terminated by ','
        missing field values are null
        ( empno, ename, job, mgr,
        hiredate char date_format date mask "dd-mm-yyyy",
        sal, comm, deptno
        )
    )
)
LOCATION ('empxt1.dat', 'empxt2.dat')
)
```



```
PARALLEL
REJECT LIMIT UNLIMITED;
ALTER SESSION ENABLE PARALLEL DML;
INSERT INTO TABLE emp SELECT * FROM empxt;
```

下面几段解释这个例子。

本例的前面几条语句，为包含数据源的操作系统目录创建目录对象，为在访问参数中指定的坏记录文件和日志文件创建目录对象。还必须授予合适的 READ(读)或 WRITE(写)目录对象的权限。

TYPE 说明仅用于解释其使用。如果没有指定，ORACLE_LOADER 就是默认的访问驱动程序。Oracle 是不懂这些在 ACCESS PARAMETERS 子句中指定的访问参数的。这些访问参数是由访问驱动程序定义的，当访问外部表时由 Oracle 提供给访问驱动程序。

PARALLEL 子句使得可以对数据源进行并行查询。并行机制的粒度默认是数据源，但是，只要可能就执行并行存取数据源。例如，如果指定 PARALLEL=3，那么就可以有多个并行执行服务器在数据源上工作。但是，仅仅在满足如下所有条件时，访问驱动程序才能提供对数据源的并行访问。

- 媒体允许在数据源中随机定位
- 可以从随机位置找到记录的边界
- 数据文件足够大，以致于值得将其分成多个块。

注意：只有在处理大量的数据时，指定 PARALLEL 子句才是有价值的。否则，指定 PARALLEL 子句是不明智的，这么做可能是有害的。

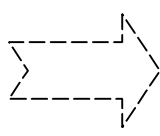
REJECT LIMIT 子句指定对在查询外部数据时可能产生的错误的数量是没有限制的。针对并行访问，这个从属无关地适用于每个并行查询进程。例如，如果将 REJECT LIMIT 指定为 10，则每个并行查询进程允许 10 次拒绝。因此，在并行查询上精确地被强制的 REJECT LIMIT 值仅有 0 和 UNLIMITED。

本例中，INSERT INTO TABLE 语句产生一个从外部数据源到处理数据的 Oracle SQL 引擎的数据流。当访问驱动程序分析外部表数据源中的数据并将其提供给外部表接口时，外部数据被从其外部表示转换成其 Oracle 的内部数据类型。

15.7.2 更改外部表

可以使用如下任何 ALTER TABLE 子句来更改外部表的特征。不允许其他子句。

ALTER TABLE 子句	说明	例子
REJECT LIMIT	更改拒绝限制	ALTER TABLE empxt REJECT LIMIT 100;
DEFAULT	更改默认目录说明	ALTER TABLE empxt
DIRECTORY		DEFAULT DIRECTORY newemp_dir;



(续表)

ALTER TABLE 子句	说明	例子
ACCESS PARAMETERS	允许更改访问参数而不需删除并重建外部表元数据	ALTER TABLE empxt ACCESS PARAMETERS (FIELDS TERMINATED BY ‘;’);
LOCATION	允许更改数据源而不需删除并重建外部表元数据	ALTER TABLE empxt LOCATION (‘empxt3.txt’, ‘empxt4.txt’);
PARALLEL	与普通表没有区别。允许更改并行度。	没有新语法
ADD COLUMN	与普通表没有区别。允许在外部表中加一列。	没有新语法
MODIFY COLUMN	与普通的表没有区别。允许修改外部表的列。	没有新语法
DROP COLUMN	与普通表没有区别。允许删除外部表的列。	没有新语法
RENAME TO	与普通表没有区别。允许更改外部表的名字。	没有新语法

15. 7. 3 删除外部表

对于外部表，DROP TABLE 语句仅删除数据库中表的元数据。这不影响实际的数据，它仍存在数据库的外面。

15. 7. 4 用于外部表的系统权限和对象权限

用于外部表的系统和对象权限是用于普通表的权限的子集。仅有如下系统权限适用于外部表：

- CREATE ANY TABLE（创建任何表）
- ALTER ANY TABLE（更改任何表）
- DROP ANY TABLE（删除任何表）
- SELECT ANY TABLE（选取任何表）

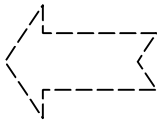
仅有如下对象权限适用于外部表：

- ALTER（更改）
- SELECT（选取）

然而，与目录相关的对象权限是：

- READ（读）权限
- WRITE（写）权限

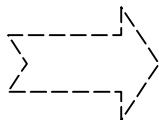
对于外部表，需要具有对包含数据源的目录的 READ（读）权限，并且具有对包含坏记录文件、日志文件或遗弃文件的目录的 WRITE（写）权限。



15.8 查看有关表的信息

如下视图能用于访问有关表的信息：

视图	描述
DBA_TABLES	DBA 视图描述数据库中的所有关系表。ALL 视图描述用户可访问的所有表。USER 视图局限于用户拥有的表。这些视图中的某些列包含有 DBMS_STATS 包或 ANALYZE 语句产生的统计数据。
ALL_TABLES	
USER_TABLES	
DBA_TAB_COLUMNS	这些视图描述数据库中的表、视图和簇的列。这些视图中的某些列包含有 DBMS_STATS 包或 ANALYZE 语句产生的统计数据。
ALL_TAB_COLUMNS	
USER_TAB_COLUMNS	
DBA_ALL_TABLES	这些视图描述数据库中的所有关系表和对象表。本书不特别讨论对象表。
ALL_ALL_TABLES	
USER_ALL_TABLES	
DBA_TAB_COMMENTS	这些视图显示表和视图的注释。注释是用 COMMENT 语句输入的。
ALL_TAB_COMMENTS	
USER_TAB_COMMENTS	
DBA_COL_COMMENTS	这些视图显示表的列和视图的列的注释。注释是用 COMMENT 语句输入的。
ALL_COL_COMMENTS	
USER_COL_COMMENTS	
DBA_EXTERNAL_TABLES	这些视图列出数据库中的外部表的特殊的属性。
ALL_EXTERNAL_TABLES	
USER_EXTERNAL_TABLES	
DBA_EXTERNAL_LOCATIONS	这些视图列出外部表的数据源。
ALL_EXTERNAL_LOCATIONS	
USER_EXTERNAL_LOCATIONS	
DBA_TAB_HISTOGRAMS	这些视图描述表和视图上的直方图。
ALL_TAB_HISTOGRAMS	
USER_TAB_HISTOGRAMS	
DBA_TAB_COL_STATISTICS	这些视图提供从相关的 TAB_COLUMNS 视图中提取出来的列的统计数据 and 直方图信息。
ALL_TAB_COL_STATISTICS	
USER_TAB_COL_STATISTICS	
DBA_TAB_MODIFICATIONS	这些视图描述自从上次对它们收集统计数据之后已经被更改过的表。这些视图仅填充具有 MONITORING 属性的表。并不立即填充它们，而是在一个时间间隔（通常是 3 小时）之后。
ALL_TAB_MODIFICATIONS	
USER_TAB_MODIFICATIONS	
DBA_UNUSED_COL_TABS	这些视图列出具有未使用列的表，这些列是用 ALTER TABLE ... SET UNUSED 语句标记的。
ALL_UNUSED_COL_TABS	
USER_UNUSED_COL_TABS	
DBA_PARTIAL_DROP_TABS	这些视图列出部分地完成 DROP COLUMN 操作的表。这些操作可能是由于用户终止了操作或系统崩溃而未完成的。
ALL_PARTIAL_DROP_TABS	
USER_PARTIAL_DROP_TABS	



第 16 章 管理索引

本章讨论索引的管理，包括如下内容：

- 管理索引的准则
- 创建索引
- 更改索引
- 监视索引的空间使用
- 删除索引
- 查看索引信息

16.1 管理索引的准则

索引是与表和簇相关的可选的结构，它能使对应于表的 SQL 语句执行得更快。正如有索引比没有索引能更快地定位信息一样，Oracle 索引能提供访问表的数据的更快路径。可以不用重写任何查询而使用索引。其结果是一样的，但感觉更快。

Oracle 提供了几种索引模式，它们提供复杂的效率功能，它们是：

- B-树索引 —— 默认的和最常用的
- B-树簇索引 —— 特意为簇定义的索引
- 散列簇索引 —— 特意为散列簇定义的索引
- 全局和局部索引 —— 相应于分区表和索引
- 逆转键索引 —— 特别适用于 Oracle 真实应用簇的应用
- 位图索引 —— 紧凑的，特别适用于具有少量值集的列
- 基于函数的索引 —— 包含函数/表达式的预先计算的值
- 域索引 —— 特别针对应用或插件

索引在逻辑上和物理上都与相关的表的数据无关。作为无关的结构，索引需要存储空间。创建或删除一个索引，不会影响基本的表、数据库应用、或其他索引。当插入、更改和删除相关的表的行时，Oracle 会自动管理索引。如果删除索引，所有的应用仍继续工作。但访问以前被索引了的数据时就可能慢了。

这一节讨论管理索引的准则，包括如下内容：

- 在表中插入数据后创建索引
- 索引正确的表和列
- 为性能而安排索引列
- 限制每个表的索引的数量
- 删除不再需要的索引
- 指定索引数据块空间使用
- 估计索引大小并设置存储参数
- 为每个索引指定表空间

- 考虑并行创建索引
- 考虑创建具有 NOLOGGING 的索引
- 考虑合并或重建索引时的损益
- 在停用或删除约束之前，考虑开销

16.1.1 在表中插入数据后创建索引

数据通常是用 SQL*Loader 或导入工具插入或装载到表中的。在插入或装载了数据后为表创建索引更加有效率。如果在装载数据之前创建了一个或多个索引，那么当插入每行时 Oracle 都必须更改每个索引。

在一个已经有数据的表上创建索引需要分类空间。某些分类空间来自于分配给索引的创建者的内存空间。每个用户的使用量是由初始化参数 SORT_AREA_SIZE 决定的。Oracle 还在创建索引时才在用户的临时表空间中分配的临时段中交换分类信息。

在某些情况下，数据可以用 SQL*Loader 的直接路径装载方法装载进表中，同时，索引也可以被创建。

16.1.2 索引正确的表和列

使用下面的准则来决定何时创建索引：

- 如果需要经常地检索大表中的少于 15% 的行，就创建索引。这个百分比的变化很大，取决于表扫描的相关速度和索引键是如何将行数据分簇的。表扫描越快，百分比越低；行数据的簇越多，百分比越高。
- 为了改善多个表的联结的性能，索引列常用于联结。

注意：主键和唯一键自动具有索引，但应该在外键上创建索引。

- 小表不需要索引。如果查询所花的时间太长了，可能这个表已经由小变大了。

某些列特别适合作索引。具有如下一个或多个特征的列就适合作索引：

- 列中的值相对比较唯一。
- 取值范围大（适合作常规索引）。
- 取值范围小（适合作位图索引）。
- 列中有许多空值，但经常查询所有具有值的行。此时，使用如下语法：

```
WHERE COL_X > -9.99 * power(10,125)
```

上面的语法比下面的好：

```
WHERE COL_X IS NOT NULL
```

这是因为前一句使用了 COL_X 列上的索引（假设 COL_X 是一个数值列）。

具有如下特征的列不太适合作索引：

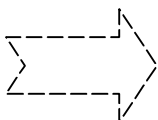
- 列中有许多空值，但又不查询非空值。

LONG 和 LONG RAW 列不能被索引。

单个索引项的大小不能比数据块中可用空间的一半多太多（减去某些杂项开销）。

16.1.3 为性能而安排索引列

在 CREATE INDEX 语句中列的排序会影响查询的性能。通常，将最常用的列放在最前面。



如果创建了一个索引来加速存取多列的查询，如 col1, col2 和 col3 的单个索引来加速查询；则仅存取 col1 或仅存取 col1 和 col2 的查询也被加速。但仅存取 col2 或 col3，或 col2 和 col3 的查询不会使用索引。

16.1.4 限制每个表的索引的数量

一个表可以有任意数量的索引。但是，索引越多，修改表的杂项开销就越多。特别是当插入或删除行时，表上的所有索引也要被更改。同样，当更改一个列时，包含该列的所有索引也要被更改。

因此，在从表中检索数据的速度和更改表的速度之间有一个折衷。例如，如果一个表主要仅用于读，则索引多就有好处；但是，如果一个表经常被更改，则索引少点更好。

16.1.5 删除不再需要的索引

考虑删除索引，如果：

- 索引不能加速查询。表可能很小，或虽表中有很多行，但几乎没有索引项。
- 应用中的查询不使用索引。
- 重建索引之前必须先删除索引。

16.1.6 指定索引数据块空间使用

为表创建索引时，索引的数据块是用表中的现存的值填充的，直到达到 PCTFREE 为止。当向表中插入一个行并且相应的索引项必须被放在正确的索引数据块（即，在前一个和后一个索引项之间）中时，才使用由 PCTFREE 为索引数据块保留的空间。

如果在适当的索引数据块中没有可用空间了，被索引的值就被放在其所属之地（基于字典词汇集合排序）。因此，如果打算将许多行插入到被索引的表中，PCTFREE 就应大点以放置新的索引值。如果表相对地静止，没有许多插入动作，用于相关索引的 PCTFREE 就可以小点，以使保持索引数据所需的数据块就少点。

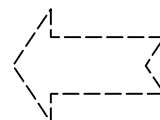
不能给索引指定 PCTUSED。

16.1.7 估计索引大小和设置存储参数

创建索引之前先估计索引的大小能更好地促进规划和管理磁盘空间。可以用索引以及表、回滚段、重做日志文件的组合估计的大小来决定支持所期望的数据库所需的磁盘空间的大小。通过这些估计，就可以购买合适的硬件和做出其他正确的决定。

用单个索引估计的大小就能更好地管理索引使用的磁盘空间。创建索引时，可以设置适当的存储参数，并改善使用该索引的应用的 I/O 性能。例如，假设在创建索引之前估计索引的最大大小，然后，如果在创建该索引时设置存储参数，就能很少为表的数据段分配盘区，并且，所有的该索引的数据都被保存在相对连续的磁盘空间扇区中。这就减少了使用该索引的磁盘 I/O 操作所需的时间。

单个索引项的最大大小大约是数据块大小的一半。



16.1.8 为每个索引指定表空间

可以在任何表空间中创建索引。可以在其索引的表的相同或不同的表空间中创建索引。如果表及其索引使用相同的表空间，就能更方便地对数据库进行管理（如表空间或文件备份），或保证应用的可用性。所有有关的数据总是在一起联机。

将表及其索引放在不同的表空间（在不同磁盘上）产生的性能比放在相同的表空间要好。这减少了磁盘竞争。但是将表及其索引放在不同的表空间，如果一个表空间（包含数据或索引）脱机了，那么，就不能保证参照那个表的语句正常工作。

16.1.9 考虑并行创建索引

就像并行创建表一样，可以并行创建索引。因为多个进程一起来创建索引，所以，此时 Oracle 就比单个服务器进程顺序地创建索引要快许多。

当并行创建索引时，其存储参数被每个查询服务器进程分别使用。因此，创建一个 INITIAL 为 5M、并行度为 12 的索引，在创建索引期间至少要消耗 60M 的存储空间。

16.1.10 考虑用 NOLOGGING 创建索引

可以在 CREATE INDEX 语句中指定 NOLOGGING 来创建索引并产生最少的重做日志记录。

注意：因为用 NOLOGGING 创建索引时不存档，所以创建索引后应进行备份。

用 NOLOGGING 创建索引有如下好处：

- 节省了重做日志文件的空间
- 缩短了创建索引的时间
- 改善了并行创建大索引时的性能

通常，不用 LOGGING 创建大索引比创建小索引时的性能改善要好些。不用 LOGGING 创建小索引对创建索引的时间只有一点点影响。但创建大索引时对性能的改善就明显了，尤其是做并行创建时。

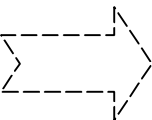
16.1.11 考虑合并或重建索引时的损益

不适当的筛分或被增加的增量可能产生索引碎片。为了消除或减小碎片，可以重建或合并索引。但在此之前，应权衡一下所做的每个选择的损益，再选择最适合你的情况的选择。表 16-1 是与重建和合并索引相关的损益对照表。

表 16-1 是重建还是合并索引...是个问题

重建索引	合并索引
迅速将索引移动到另一个表空间	不能将索引移动到另一个表空间
高开销：需要更多的磁盘空间	低开销：不需要更多的磁盘空间
创建新树，如果可能就收缩高度	在树的相同分支中合并叶子数据块
可以迅速地更改存储参数和表空间参数，而不需 要删除原始索引。	迅速地为使用而释放索引的叶子数据块。

在 B-树叶子数据块能被释放重用的情况下，可以用如下语句将这些叶子数据块合并：



```
ALTER INDEX vmoore COALESCE;
```

图 16-1 解释了在索引 vmoore 上执行 ALTER INDEX COALESCE 语句的效果。执行该操作之前，头两片叶子数据块的 50% 是满的。这说明可以减少碎片而使第一个数据块被完全填满，而释放第二个数据块。在这个例子中，假设 PCTFREE=0。

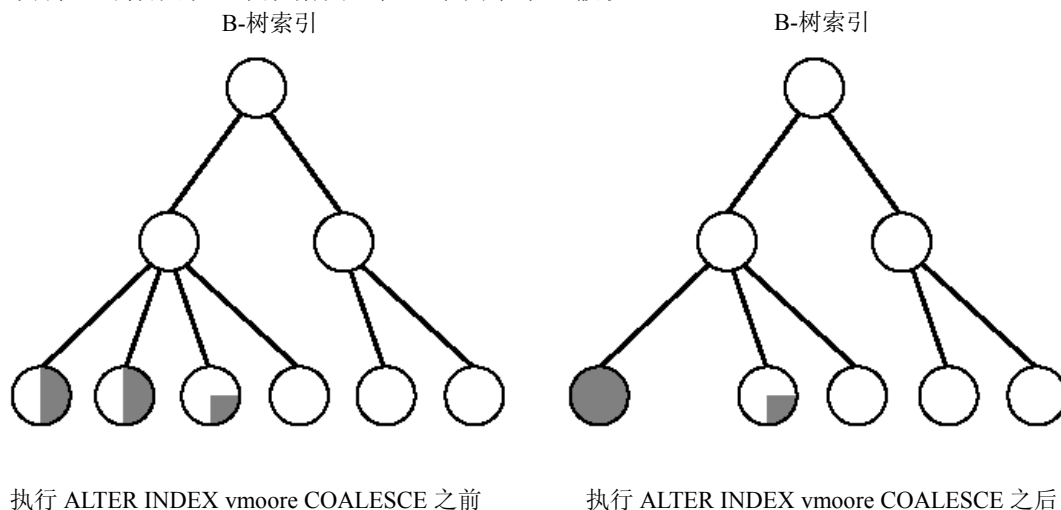


图 16-1 合并索引

16.1.12 在停用或删除约束之前，考虑开销

因为唯一键和主键有相关的索引，所以，在考虑或是停用或是删除 UNIQUE 键或 PRIMARY 键约束时，应该考虑删除和创建索引的开销。如果与 UNIQUE 键或 PRIMARY 键约束相关的索引相当大，则保持约束启用而不是删除后再重建该大索引，就能节省时间。在删除或停用 UNIQUE 键或 PRIMARY 键约束时，还可以明确指定你是希望保持还是删除索引。

16.2 创建索引

本节描述如何创建索引。要在你自己的模式中创建索引，至少要满足如下条件之一：

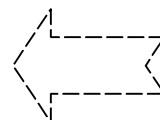
- 要被索引的表或簇是在你自己的模式中。
- 你在要被索引的表上有 INDEX 权限。
- 你具有 CREATE ANY INDEX 系统权限。

要在其他模式中创建索引，要满足如下所有条件：

- 你具有 CREATE ANY INDEX 系统权限。
- 其他模式的拥有者在保存索引或索引分区的表空间中有限额，或者他具有 UNLIMITED TABLESPACE 系统权限。

本节包括如下内容：

- 明确地创建索引
- 明确地创建唯一索引



- 创建与约束相关的索引
- 采集创建索引时附带的统计数据
- 创建大索引
- 联机创建索引
- 创建基于函数的索引
- 创建键压缩索引

16.2.1 明确地创建索引

可以用 CREATE INDEX 语句明确地创建索引（不含完整性约束）。如下语句在 emp 表的 ename 列上创建一个名为 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，为该索引明确地指定了几个存储设置和一个表空间。如果没有给索引指定存储选项（如 INITIAL 和 NEXT），就自动使用默认或指定的表空间的默认存储选项。

16.2.2 明确地创建唯一索引

索引可以是唯一的或非唯一的。唯一索引可以保证在键列上表不会有两行有重复的值。非唯一索引不会在列值上施加这个限制。

可用 CREATE UNIQUE INDEX 语句来创建唯一索引。如下例子创建一个唯一索引：

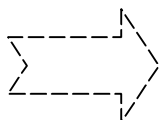
```
CREATE UNIQUE INDEX dept_unique_index ON dept (dname)
TABLESPACE indx;
```

换句话说，可以在希望的列上定义 UNIQUE 完整性约束。Oracle 通过自动地在唯一键上定义一个唯一索引来保证 UNIQUE 完整性约束。在下面章节中讨论这些问题。但是，任何为查询性能而存在的索引，包括唯一索引，最好是采用明确地创建方式。

16.2.3 创建与约束相关的索引

Oracle 通过在唯一键或主键上创建一个唯一索引，来在表上实施 UNIQUE KEY 或 PRIMARY KEY 完整性约束。当启用约束时，Oracle 自动创建该索引。当发布 CREATE TABLE 或 ALTER TABLE 语句而创建索引时不需要你做什么，但可以指定 USING INDEX 子句来对其创建实施控制。这包括两种情况：当定义约束并启用约束时，当启用已定义但被停用的约束时。

为启用 UNIQUE 或 PRIMARY KEY 约束，就创建一个相关索引，表的拥有者必须在打算保存索引的表空间中有限额，或者具有 UNLIMITED TABLESPACE 系统权限。约束的相关索引总是采用约束的名字，除非你选择地指定了其他名字。



16.2.3.1 为与约束相关的索引指定存储选项

可以用 USING INDEX 子句, 为与 UNIQUE 和 PRIMARY KEY 约束相关的索引, 设置存储选项。如下 CREATE TABLE 语句启用一个 PRIMARY KEY 约束, 并指定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
PCTFREE 0;
```

16.2.3.2 指定与约束相关的索引

如果需要对 UNIQUE 和 PRIMARY KEY 约束相关的索引进行更明确的控制, Oracle 允许你:

- 指定一个 Oracle 用来实施约束的现存的索引
 - 指定一个 Oracle 用来创建索引并实施约束的创建索引的语句
- 用 USING INDEX 子句来指定这些选项, 如下几个语句是几个例子:

例 1:

```
CREATE TABLE a (
a1 INT PRIMARY KEY USING INDEX (create index ai on a (a1)));
```

例 2:

```
CREATE TABLE b(
b1 INT,
b2 INT,
CONSTRAINT bu1 UNIQUE (b1, b2)
USING INDEX (create unique index bi on b(b1, b2)),
CONSTRAINT bu2 UNIQUE (b2, b1) USING INDEX bi);
```

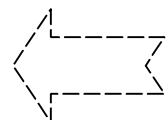
例 3:

```
CREATE TABLE c(c1 INT, c2 INT);
CREATE INDEX ci ON c (c1, c2);
ALTER TABLE c ADD CONSTRAINT cpk PRIMARY KEY (c1) USING INDEX ci;
```

当单独一条语句创建带有一个约束并又将该索引用于另一个约束时, 系统将试图重新安排该子句, 以便在使用索引之前先创建索引。

16.2.4 采集创建索引时伴随的统计数据

在创建索引或重建索引期间, Oracle 提供了一个使用很小的资源开销就能采集统计数据的机会。这些统计数据被存储在数据字典中, 以便优化器在为执行 SQL 语句而选择方案时随机使用。如下语句计算在表 emp 的列 ename 上创建索引 emp_ename 时, 有关索引、表和列的统计数据。



```
CREATE INDEX emp_ename ON emp(ename)
COMPUTE STATISTICS;
```

16.2.5 创建大索引

创建很大的索引时，考虑采用如下过程，为创建索引分配一个较大的临时表空间：

①用 CREATE TABLESPACE 或 CREATE TEMPORARY TABLESPACE 语句创建一个新的临时表空间。

②用 ALTER USER 语句的 TEMPORARY TABLESPACE 选项制作这个新的临时表空间。

③用 CREATE INDEX 语句创建索引。

④用 DROP TABLESPACE 语句删除该表空间。然后，用 ALTER USER 语句将临时表空间重新设置成原始的临时表空间。

使用该过程能避免将平常的、一般是共享的临时表空间扩展到一个不合理的大小，而影响以后的性能的问题。

16.2.6 联机创建索引

可以联机创建和重建索引。这使得你在表上创建或重建索引的同时，更改基础表。创建索引时可以执行 DML 操作，但不允许执行 DDL 操作。联机创建或重建索引时不支持并行执行。

如下语句描述联机创建索引的操作：

```
CREATE INDEX emp_name ON emp (mgr, emp1, emp2, emp3) ONLINE;
```

注意：尽管在联机创建索引期间可以执行 DML 操作，但 Oracle 建议，在这个过程中不要执行主要的/大的 DML 操作。这是因为在基础表上做 DML 操作时，该操作会对那个资源加锁。在基础表上的事务提交或回滚之前，即释放锁之前，DDL 是不能进行创建索引的。

例如，如果想装载总行数多达现存的表的 30% 的行，就应该先装载，然后才联机创建索引。

16.2.7 创建基于函数的索引

基于函数的索引促进了限定函数或表达式的返回值的查询。该函数或表达式的值被预先计算出来并存储在索引中。

16.2.7.1 基于函数的索引的特点

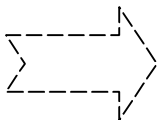
基于函数的索引可以：

- 创建更强有力的分类

可以用 UPPER 和 LOWER 函数执行区分大小写的分类，用 DESC 关键词执行降序分类，用 NLSSORT 函数执行基于语言的分类。

- 预先计算出计算密集的函数的值，并在索引中将其分类。

可以在索引中存储要经常访问的、计算密集的函数。当需要访问值时，该值已经计算



出来了，因此，极大地改善了查询的执行性能。

- 增加优化器可以执行范围扫描而不是全表扫描的情况的数量。

例如，考虑如下 WHERE 子句中的表达式：

```
CREATE INDEX idx ON Example_tab(column_a + column_b);
SELECT * FROM example_tab WHERE column_a + column_b < 10;
```

因为该索引是建立在 (column_a + column_b) 之上的，所以优化器可以为该查询使用范围扫描。如果预计选取的行数会少于大表的 15%，范围扫描一般会产生快的响应时间。如果表达式被用于基于函数的索引，优化器就能靠表达式更精确地估计出选中了多少行。（基于函数的表达式被称为虚列，ANALYZE 语句可以在这些列上建立直方图）

- 使真正的降序索引成为可能

它们被视为基于函数的索引的特例。

注意：Oracle 用 DESC 关键词按降序来分类各个列。这些索引被视为基于函数的索引。降序索引不能做位图或逆转，并且不能被用于位图优化。为获得早于 Oracle8.1 版的 DESC 特性，要在 CREATE INDEX 语句中删除 DESC 关键词。

- 在对象列和 REF 列上创建索引

描述对象的方法可以被用作创建索引的函数。例如，可以用 MAP 方法在一个对象类型的列上创建索引。

16.2.7.2 基于函数的索引如何工作

为了在你自己的模式中创建基于函数的索引，你必须被授予 QUERY REWRITE 系统权限。为了在另一个模式中或在另一个模式的表上创建索引，你必须具有 CREATE ANY INDEX 和 GLOBAL QUERY REWRITE 权限。

必须定义如下初始化参数，才能创建基于函数的索引：

- 将 QUERY_REWRITE_INTEGRITY 设置为 TRUSTED
- 将 QUERY_REWRITE_ENABLED 设置为 TRUE
- 将 COMPATIBLE 设置为 8.1.0.0.0 或更大的值

另外，为使用基于函数的索引，需要：

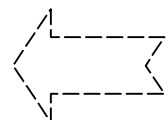
- 在创建索引后必须分析表。
- 因为 NULL 值不会被保存在索引中，所以必须保证查询不需要被索引的表达式中的任何 NULL 值。

注意：CREATE INDEX 存储在基于函数的索引中使用的，最近的函数的时间戳。当索引被校验时就更改该时间戳。当执行基于函数的索引的表空间的时间点恢复时，如果在索引中使用的最近的函数的时间戳，比存储在索引中的时间戳要新，那么就将索引标记为无效。必须使用 ANALYZE VALIDATE INDEX 语句来校验该索引。

为了说明基于函数的索引，考虑如下语句，它定义一个被定义在函数 area(geo) 上的基于函数的索引 (area_index)：

```
CREATE INDEX area_index ON rivers (area(geo));
```

在如下 SQL 语句中，当在 WHERE 子句中参照 area(geo) 时，优化器就考虑使用索引



area_index。

```
SELECT id, geo, area(geo), desc
FROM rivers
WHERE Area(geo) >5000;
```

表的拥有者应该在基于函数的索引所使用的函数上具有 EXECUTE 权限。

因为基于函数的索引依赖于它所使用的各个函数，所以当更改一个函数后，它可能会变成未被校验状态。如果函数是有效的，就可以用 ALTER INDEX...ENABLE 语句来启用已被停用的基于函数的索引。可用 ALTER INDEX...DISABLE 语句来停用基于函数的索引。如果正在对函数本身做工作时就要考虑做这件事。

16.2.7.3 基于函数的索引的例子

某些使用基于函数的索引的例子如下。

例：用于大小写敏感搜索的基于函数的索引 如下语句基于 ename 列的大写字母计算，在表 emp 上创建基于函数的索引 idx：

```
CREATE INDEX idx ON emp (UPPER(ename));
```

现在，SELECT 语句使用 UPPER(ename) 上的基于函数的索引来检索所有的名字开头为 JOH 的职员：

```
SELECT * FROM emp WHERE UPPER(ename) LIKE 'JOH%';
```

这个例子也说明了一个大小写敏感搜索。

例：用基于函数的索引来预先计算算术表达式 这条语句基于一个表达式创建一个基于函数的索引：

```
CREATE INDEX idx ON t (a + b * (c - 1), a, b);
```

SELECT 语句可以使用索引范围扫描（在下面的 SELECT 语句中，表达式是该索引的前缀）或者索引完全扫描（当索引指定了一个高并行度时更可取）。

```
SELECT a FROM t WHERE a + b * (c - 1) < 100;
```

例：将基于函数的索引用于依赖于语言的分类 可以用基于函数的索引来支持语言学分类索引。NLSSORT 是一个用于返回被赋予了字符串的分类键的函数。因此，如果想用 NLSSORT 在 name 上创建一个索引，就发布如下语句：

```
CREATE INDEX nls_index ON t_table (NLSSORT(name, 'NLS_SORT = German'));
```

这条语句在带有校勘序列 German 的表 t_table 上创建索引 nls_index。

现在，如下用 NLS_SORT 索引从表 t_table 中选取信息：

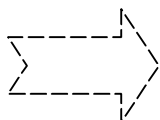
```
SELECT * FROM t_table ORDER BY name;
```

用校勘序列 German 对行进行排序。

如下例子组合了大小写敏感的分类和按语言的分类：

```
CREATE INDEX empi ON emp
UPPER ((ename), NLSSORT(ename));
```

这里，NLS_SORT 说明没有出现在 NLSSORT 的参数中，因为，NLSSORT 使用针对语言学分类键的语言的会话设置值。前面的例子说明了指定 NLS_SORT 的一种情况。



16.2.8 创建键压缩索引

用键压缩来创建索引可以消除重复的键列的前缀值。

键压缩将索引键分成前缀项和后缀项。压缩是通过使索引数据块中的所有后缀项共享前缀项来实现的。这个共享可以节省巨大的空间，使你可以在每个索引数据块中存储更多的键，并改善性能。

键压缩有益于如下情况：

- 具有一个用附加 ROWID 才使键唯一的非唯一的索引。如果在此使用键压缩，则重复的键被作为一个前缀项存储在具有 ROWID 的索引数据块中。剩余的行变成仅由 ROWID 组成的后缀项。
- 具有唯一的多列的索引。

用 COMPRESS 子句来启用键压缩。还可以指定前缀长度（即键列的个数），以便说明如何将键列分成前缀项和后缀项。例如，如下语句压缩在索引叶子数据块中的键的重复值。

```
CREATE INDEX emp_ename ON emp(ename)
TABLESPACE users
COMPRESS 1;
```

还可以在重建的时候指定 COMPRESS 子句。例如，重建期间可以像下面这样停用压缩：

```
ALTER INDEX emp_ename REBUILD NOCOMPRESS;
```

16.3 更改索引

为了更改索引，你的模式必须包含该索引，或者你必须具有 ALTER ANY INDEX 系统权限。

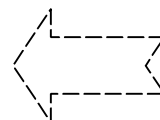
允许在 ALTER INDEX 语句中包含的动作是：

- 重建或合并现存的索引
- 回收未使用的空间或分配新盘区
- 指定并行执行（或相反）和更改并行度
- 更改存储参数或物理属性
- 指定 LOGGING 或 NOLOGGING
- 启动或停用键压缩
- 将索引标记成不可用
- 开始或停止对索引使用的监视

不能更改索引的列结构。

有关这些操作的更详细的讨论包括在下面的章节中：

- 更改索引的存储特征
- 重建现存的索引
- 监视索引使用



16.3.1 更改索引的存储特征

用 ALTER INDEX 语句来更改任何索引的存储参数，包括那些由 Oracle 创建的用于实施主键和唯一键的完整性约束的索引。例如，如下语句更改 emp_name 索引：

```
ALTER INDEX emp_name  
STORAGE (PCTINCREASE 50);
```

不能更改存储参数 INITIAL 和 MINEXTENTS。给其他存储参数新设置的值只会影响以后分配给该索引的分区。

对于实现完整性约束的索引，可以用发布包含 ENABLE 子句的 USING INDEX 子句的 ALTER TABLE 语句来调整其存储参数。例如，下面的语句修改用于实施主键约束，而在 emp 表上创建的索引的存储选项：

```
ALTER TABLE emp  
ENABLE PRIMARY KEY USING INDEX  
PCTFREE 5;
```

16.3.2 重建现存的索引

在重建现存的索引之前，应该按照表 16-1 中所描述的那样，比较一下重建该索引和合并该索引之间的损益。

重建索引时，是将现存的索引作为数据源。用这种方法创建索引，允许修改存储特征或将索引移动到新的表空间。基于现存数据源来重建索引会删除额外数据块的碎片。相对于删除索引再用 CREATE INDEX 语句来创建索引而言，重建现存的索引提供了更好的性能。

如下语句重建现存的索引 emp_name：

```
ALTER INDEX emp_name REBUILD;
```

REBUILD 子句必须立即地紧跟在索引名之后，并出现在任何其他选项之前。它不能与 DEALLOCATE UNUSED 子句一起使用。

如果具有联机重建索引的选项，就可联机重建索引。如下语句联机重建 emp_name 索引：

```
ALTER INDEX REBUILD ONLINE;
```

如果没有重建索引所需的空闲空间，可以用合并索引来代替。合并索引也可以联机执行。

16.3.3 监视索引使用

Oracle 提供了一种监视索引的方法，以便决定它们正在被使用或没被使用。如果一个索引目前没被使用，那么可将其删除，因而减少了不需要的语句负荷。

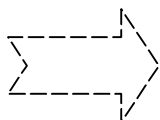
为了开始监视一个索引的使用，发布如下语句：

```
ALTER INDEX index MONITORING USAGE
```

以后，发布如下语句来停止该监视：

```
ALTER INDEX index NOMONITORING USAGE
```

可用查询视图 V\$OBJECT_USAGE 来查询正在被监视的索引，以便查看是否该索引已



经被使用。该视图包含一个 USED 列，其值是 YES 或 NO，取决于该索引在被监视期间是否已经被使用。该视图还包含该监视阶段的开始和结束时间，还包含一个 MONITORING 列（YES/NO），以标识是否现在激活了使用监视。

每次指定 MONITORING USAGE 时，V\$OBJECT_USAGE 视图都被针对指定的索引进行重新设置。以前的使用信息被清除或重新设置，新的开始时间被记录下来。当指定 NOMONITORING USAGE，又不执行进一步的监视时，则该监视阶段的结束时间被记录下来。在下次发布 ALTER INDEX ... MONITORING USAGE 语句之前，该视图的信息保持不变。

16.4 监视索引的空间使用

如果经常插入、更新和删除索引中的键值，那么，随着时间的推移，该索引可能会失去其已获得的空间效率。为了时常地监视索引的空间使用效率，可以首先分析该索引的结构，使用 ANALYZE INDEX ... VALIDATE STRUCTURE 语句，然后查询 INDEX_STATE 视图：

```
SELECT PCT_USED FROM INDEX_STATS WHERE NAME = 'index';
```

根据插入、更新和删除索引键的频繁程度，索引的空间使用的百分比是变化的。为了开发出索引的空间使用的平均效率的历史，可以执行几次下面的操作序列：

- 分析统计数据
- 校验索引
- 检查 PCTUSED
- 删除和重建（或合并）索引

当发现一个索引的空间使用效率降到了其平均值之下，就可以通过删除索引并重建（或合并）该索引来精简该索引的空间。

16.5 删除索引

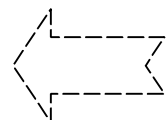
要想删除索引，则该索引必须包含在你的模式中，或你必须具有 DROP ANY INDEX 系统权限。

要删除索引的某些原因如下：

- 不再需要该索引
- 该索引没有为针对其相关的表所发布的查询提供所期望的性能改善。例如，表可能很小，或者尽管表中有许多行但只有很少的索引项。
- 应用没有用该索引来查询数据。
- 该索引已经变成无效了，必须在重建之前删除该索引。
- 该索引已经变得太碎了，必须在重建之前删除该索引。

删除索引之后，该索引的段的所有盘区都回归给包含它的表空间，并可用于表空间中的其他对象。

如何删除索引，取决于是否是用 CREATE INDEX 语句来明确地创建该索引的，或是由



于在一个表上定义键约束而隐含地创建该索引的。如果是用 CREATE INDEX 语句来明确地创建该索引的，则可以用 DROP INDEX 语句删除该索引。如下语句删除 emp_ename 索引：

```
DROP INDEX emp_ename;
```

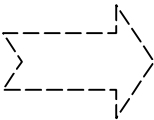
不能仅仅删除与已启用的 UNIQUE KEY 键或 PRIMARY KEY 键约束相关的索引。要删除一个与约束相关的索引，必须停用或删除该约束本身。

注意：删除表也就自动也删除了所有与其相关的索引。

16.6 查看索引信息

下面的视图显示有关索引的信息：

视图	说明
DBA_INDEXES	DBA 视图描述数据库中所有表上的索引。ALL 视图描述用户可以访问的所有表上的索引。USER 视图局限于用户所拥有的索引。这些视图中的某些列包含有由 DBMS_STATS 包或 ANALYZE 语句产生的统计数据。
ALL_INDEXES	
USER_INDEXES	
DBA_IND_COLUMNS	这些视图描述表上的索引的列。这些视图中的某些列包含有由 DBMS_STATS 包或 ANALYZE 语句产生的统计数据。
ALL_IND_COLUMNS	
USER_IND_COLUMNS	
DBA_IND_EXPRESSIONS	这些视图描述表上的基于函数的索引的表达式。
ALL_IND_EXPRESSIONS	
USER_IND_EXPRESSIONS	
INDEX_STATS	存储最后一条 ANALYZE INDEX...VALIDATE STRUCTURE 语句所产生的信息。
INDEX_HISTOGRAM	存储最后一条 ANALYZE INDEX...VALIDATE STRUCTURE 语句所产生的信息。
V\$OBJECT_USAGE	包含由 ALTER INDEX...MONITORING USAGE 语句所产生的索引使用的信息。



第 17 章 管理分区表和分区索引

本章描述管理分区表和分区索引的几个方面，包括如下内容：

- 什么是分区表和分区索引
- 分区的方法
- 创建分区表
- 管理分区表
- 分区表和分区索引的例子
- 查看有关分区表和分区索引的信息

17.1 什么是分区表和分区索引

今天的企业经常运行包含多达几百个 GB，某些时候多达几个 TB 数据的任务关键的数据库。这些企业面临需要支持和维护非常大的数据库（VLDB）的挑战，并要谋划出对付那些挑战的方法。

满足 VLDB 要求的一个方法是创建和使用分区表和分区索引。分区表允许将数据分成被称为分区甚至子分区的更小的、更好管理的块。索引也可以这么分区。每个分区可以被单独管理，可以不依赖于其他分区而单独发挥作用，因此提供了一个更有利于可用性和性能的结构。

如果打算使用并行执行，则分区提供了另一种并行的方法。通过给表或索引的不同分区分配不同的并行执行服务器，就可以并行执行对分区表和分区索引的操作。

表或索引的分区和子分区都共享相同的逻辑属性。例如，表的所有分区（或子分区）共享相同的列和约束定义，一个索引的分区（或子分区）共享相同的索引选项。然而，它们可以具有不同的物理属性（如表空间）。

尽管不需要将表或索引的每个分区（或子分区）放在不同的表空间，但这样做更好。将分区存储到不同的表空间能够：

- 减少数据在多个分区中冲突的可能性
- 可以单独备份和恢复每个分区
- 控制分区与磁盘驱动器之间的映射（对平衡 I/O 负载是重要的）
- 改善可管理性、可用性和性能

分区操作对现存的应用和运行在分区表上的标准 DML 语句来说是透明的。但是，可以通过在 DML 中使用分区扩展表或索引的名字来对应用编程，使其利用分区的优点。

可以使用 SQL*Loader, Import 和 Export 工具来装载或卸载分区表中的数据。这些工具都是支持分区和子分区的。

17.2 分区的方法

Oracle 提供了如下几种分区方法：

- 范围分区
- 散列分区
- 列表分区
- 组合分区

可对索引和表分区。全局索引只能按范围分区，但可以将其定义在任何类型的分区或非分区表上。通常全局索引比局部索引需要更多的维护。

组建局部索引，以便反映其基础表的结构。它与基础表是等同分区的，即，它与基础表在同样的列上分区，创建同样数量的分区或子分区，设置与基础表相对应的同样的分区边界。对局部索引而言，当维护活动影响分区时，会自动维护索引分区。这保证了索引与基础表之间的等同分区。

下面几节能帮你决定适合于你的需求的分区方法：

- 何时使用范围分区方法
- 何时使用散列分区方法
- 何时使用列表分区方法
- 何时使用组合分区方法

17.2.1 何时使用范围分区方法

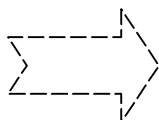
要想将行映射到基于列值范围的分区，就使用范围分区方法。当数据可以被划分成逻辑范围时，如年度中的月份，这种类型的分区就有用了。当数据在整个范围中能被均等地划分时性能最好。如果靠范围的分区会由于不均等的划分而导致分区在大小上明显不同时，你就需要考虑其他的分区方法了。

创建范围分区时，必须指定：

- 分区方法：范围
- 分区列
- 标识分区边界的分区描述

如下例子创建一个4个分区的表，一个分区对应一个季度的销售情况。列 `sale_year`, `sale_month` 和 `sale_day` 是分区列，即它们的值组成一个特殊的行的分区键。`VALUES LESS THAN` 子句决定分区边界：分区键值比在子句中指定的，已排序的值的列表还要小的行，将被存储在分区中。每个分区都有名字 (`sales_q1`, `sales_q2`, ...), 每个分区都在单独的表空间 (`tsa`, `tsb`, ...).

```
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 );

```

具有 `sale_year = 1999`, `sale_month = 8` 和 `sale_day = 1` 的行的分区键是 (1999, 8, 1), 将被存储在分区 `sales_q3` 中。

范围分区表的每个分区都被存储在单独的段中。

注意: 如果你的企业已经或将要使用具有不同字符集的数据库, 那么, 在字符列上分区时就需留意了, 因为字符的分类序列在不同字符集中是不同的。

17.2.2 何时使用散列分区方法

如果数据不那么容易进行范围分区, 但为了性能和管理的原因又想分区时, 就使用散列分区方法。散列分区提供了一种在指定数量的分区中均等地划分数据的方法。基于分区键的散列值将行映射到分区中。创建和使用散列分区会给你提供了一种很灵活的放置数据的方法, 因为你可以通过在 I/O 驱动器之间播撒 (摘掉) 这些均等定量的分区, 来影响可用性和性能。

为了创建散列分区, 你需要指定:

- 分区方法: 散列
- 分区列
- 分区数量或单独的分区描述

如下例子创建一个散列分区表。分区列是 `id`, 创建 4 个分区并赋予系统产生的名字, 并将其放到 4 个命名表空间 (`gear1`, `gear2`, ...) 中。

```

CREATE TABLE scubagear
(id NUMBER,
name VARCHAR2 (60))
PARTITION BY HASH (id)
PARTITIONS 4
STORE IN (gear1, gear2, gear3, gear4);

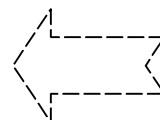
```

散列分区表的每个分区都被存储在单独的段中。

17.2.3 何时使用列表分区方法

当你需要明确地控制如何将行映射到分区时, 就使用列表分区方法。可以在每个分区的描述中为该分区列指定一系列离散值。这不同于范围分区, 在那里一个范围与一个分区相关; 这也不同于散列分区, 在那里用户不能控制如何将行映射到分区。

列表分区方法是特意为遵从离散值的模块化数据划分而设计的。范围分区或散列分区不那么容易做到这一点, 这是因为:



- 范围分区为分区列假设了一个值的自然范围。这就不可能将该值的范围以外的分区组织到一起。
- 散列分区不允许对数据的划分进行控制，因为是用系统的散列函数将数据划分到各种分区的。此外，这就不可能从逻辑上将用于分区列的离散值按分区组织到一起。

进一步说，列表分区可以非常自然地将无序的和不相关的数据集进行分组和组织到一起。

与范围分区和散列分区所不同，列表分区不支持多列分区。如果要将表按列分区，那么分区键就只能由表的一个单独的列组成。然而，可以用范围分区或散列分区方法进行分区的所有的列，都可以用列表分区方法进行分区。

当创建列表分区时，你必须指定：

- 分区方法：列表
- 分区列
- 分区描述，每个描述指定一串文字值（值的列表），它们是分区列（它们限定将被包括在分区中的行）的离散值。

如下例子创建一个列表分区表。创建一个按地区进行分区的表 `sales_by_region`；即，按地理位置将州分组到一起。

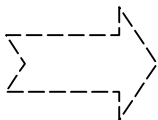
```
CREATE TABLE sales_by_region
(deptno number,
deptname varchar2(20),
quarterly_sales number(10, 2),
state varchar2(2))
PARTITION BY LIST (state)
(PARTITION q1_northwest VALUES ('OR', 'WA'),
PARTITION q1_southwest VALUES ('AZ', 'UT', 'NM'),
PARTITION q1_northeast VALUES ('NY', 'VM', 'NJ'),
PARTITION q1_southeast VALUES ('FL', 'GA'),
PARTITION q1_northcentral VALUES ('SD', 'WI'),
PARTITION q1_southcentral VALUES ('OK', 'TX'));
```

通过检查一个行的分区列的值是否属于描述该分区的值的集合，就可以将该行映射到该分区。

例如，下面的行按如下方式插入：

- (10, 'accounting', 100, 'WA') 映射到分区 `q1_northwest`
- (20, 'R&D', 150, 'OR') 映射到分区 `q1_northwest`
- (30, 'sales', 100, 'FL') 映射到分区 `q1_southeast`
- (40, 'HR', 10, 'TX') 映射到分区 `q1_southwest`
- (50, 'systems engineering', 10, 'CA') 不会映射到表中的任何分区。

有关列表分区的一件有趣的事是，各个分区之间没有明显的排序（不象范围分区）。



17.2.4 何时使用组合分区方法

组合分区方法是在分区中使用范围分区方法分区数据，而在子分区中则使用散列分区方法。组合分区很适合于历史数据和条块数据两者，它改善了范围分区及其数据放置的管理性，并提供了散列分区的并行机制的优点。

当创建组合分区时，你要指定：

- 分区方法：范围
- 分区列
- 标识分区边界的分区描述
- 子分区方法：散列
- 子分区列
- 每个分区的子分区数量，或子分区的描述

如下语句创建一个组合分区表。在本列中，创建了 3 个范围分区，每个范围分区包含 8 个子分区。因为没有给予分区命名，所以系统为其命名，但 STORE IN 子句将其划分到 4 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, ts2, ts3, ts4)
(PARTITION p1 VALUES LESS THAN (1000),
PARTITION p2 VALUES LESS THAN (2000),
PARTITION p3 VALUES LESS THAN (MAXVALUE));
```

组合分区表的每个子分区都被存储在自己的段中。因为它们的数据被存储在它们的子分区的段中，所以组合分区表的分区仅仅是一个逻辑结构。因为有了分区，所以这些子分区才共享相同的逻辑属性。与范围分区表中的范围分区所不同，子分区不能具有与其所属分区所不同的物理属性，尽管它们不需要被存储在相同的表空间中。

17.3 创建分区表

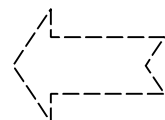
除了包括一个分区方法的子句之外，创建分区表或分区索引很像创建非分区表或非分区索引（在第 15 章“管理表”中描述）。你所包括的分区子句，和下级子句，取决于你想进行的分区的类型。

你可以对普通表（堆积结构表）和索引结构表两者，包括那些具有 LOB 列的表，进行分区。你可以在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。

创建（或更改）分区表时，可以指定行移动子句，即 ENABLE ROW MOVEMENT 或者 DISABLE ROW MOVEMENT。该子句启用或停用将行迁移到一个新的分区（当其键被更改时）。默认值为 DISABLE ROW MOVEMENT。

下面几节描述，为各种类型的分区表和分区索引，创建分区的细节和例子：

- 创建范围分区表
- 创建散列分区表



- 创建列表分区表
- 创建组合分区表
- 创建分区索引结构表
- 用于多个数据块大小的分区限制

17.3.1 创建范围分区表

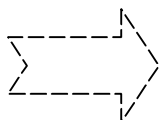
CREATE TABLE 语句的 PARTITION BY RANGE 子句表示将表进行范围分区。PARTITION 子句标识各个分区范围，可选的 PARTITION 子句的下级子句可以指定特别用于该分区段的物理的和其他的属性。如果在分区层次上没有重载，则分区继承其基础表的属性。

如下例子比前面用于范围分区表的例子复杂一些。在表层次上指定了存储参数和 LOGGING 属性。这些属性代替了相应的从表空间层次继承的默认值（可用于表自身，并可被范围分区继承）。当然，因为第一季度没有什么业务，所以分区 sales_q1 的存储属性安排得小些。ENABLE ROW MOVEMENT 子句表示，如果键值被更改了（它会将行放置到不同的分区），就允许将行迁移到新分区。

```
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相似于创建范围分区表。下面是一个在上面的表的 sales_month 列上，创建一个范围分区全局索引的例子。每个索引分区都被命名，但存储在索引的默认表空间中。

```
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));

```

17.3.2 创建散列分区表

CREATE TABLE 语句的 PARTITION BY HASH 子句表示将表进行散列分区。PARTITIONS 子句可用于指定要创建的分区数量，可选地，以及用于存储这些分区的表空间。换句话说，可以使用 PARTITION 子句来命名各个分区及其表空间。

可以为散列分区指定的属性只有 TABLESPACE。一个表的所有散列分区都必须共享从表层次上继承来的相同的属性（除 TABLESPACE 之外）。

下面的例子说明了创建名为 dept 的散列分区表的 2 种方法。在第一个例子中指定了分区的数量，但是由系统给它们命名，并将它们存储在表的默认表空间中。

```

CREATE TABLE dept (deptno NUMBER, deptname VARCHAR(32))
PARTITION BY HASH(deptno) PARTITIONS 16;

```

在这第二个例子中，指定了各个分区的名字及其所属的表空间。在表层次上还明确地为每个散列分区（段）指定了初始盘区的大小，并且所有分区都继承这个属性。

```

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);

```

如果为上面的表创建局部索引，则 Oracle 会创建一个与基础表等同分区的索引。Oracle 还保证，当对基础表进行维护时，该索引也被自动维护。下面是一个在表 dept 上创建局部索引的例子。

```

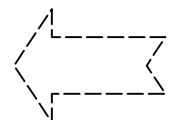
CREATE INDEX locd_dept_ix ON dept(deptno) LOCAL

```

可选地，可以对散列分区和存储局部索引分区的表空间进行命名。如果没做这件事，Oracle 就将对应的基础分区的名字作为索引分区的名字，并将索引分区存储在表分区的相同的表空间中。

17.3.3 创建列表分区表

创建列表分区的过程很像创建范围分区。但创建列表分区时，在 CREATE TABLE 语



句中是指定 **PARTITION BY LIST** 子句，并且该 **PARTITION** 子句会指定一串文字值，它们是分区列的离散值（限定在分区中包括的行）。与范围分区相同，**PARTITION** 子句的、可选的下级子句可以为分区段指定特殊的物理的和其他的属性。如果在分区层次上没有重载，则分区继承其基础表的属性。

在如下例子中创建表 `sales_by_region`，并用列表分区方法将其分区。前面 2 个 **PARTITION** 子句指定物理属性，它们重载了表层次的默认值。剩下的 **PARTITION** 子句没有指定属性，那些分区就从表层次上继承它们的默认的物理属性。

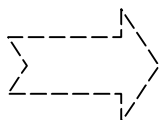
```
CREATE TABLE sales_by_region (item# INTEGER, qty INTEGER,
store_name VARCHAR(30), state_code VARCHAR(2),
sale_date DATE)
STORAGE(INITIAL 10K NEXT 20K) TABLESPACE tbs5
PARTITION BY LIST (state_code)
(
PARTITION region_east
VALUES ('MA','NY','CT','NH','ME','MD','VA','PA','NJ')
STORAGE (INITIAL 20K NEXT 40K PCTINCREASE 50)
TABLESPACE tbs8,
PARTITION region_west
VALUES ('CA','AZ','NM','OR','WA','UT','NV','CO')
PCTFREE 25 NOLOGGING,
PARTITION region_south
VALUES ('TX','KY','TN','LA','MS','AR','AL','GA'),
PARTITION region_central
VALUES ('OH','ND','SD','MO','IL','MI', null, 'IA')
);
```

17.3.4 创建组合分区表

为了创建组合分区表，先用 **CREATE TABLE** 语句的 **PARTITION BY RANGE** 子句，然后指定一个与 **PARTITION BY HASH** 语句遵从相似语法和规则的 **SUBPARTITION BY HASH** 子句。各个 **PARTITION** 后面紧跟 **SUBPARTITION** 或 **SUBPARTITIONS** 子句。

为（范围）分区指定的属性适用于那个分区的所有子分区。可以为每个（范围）分区指定不同的属性，并且，如果分区的子分区要分布的表空间与其他分区不同的话，还可以在分区层次上指定 **STORE IN** 子句。下面的例子对此作解释。

```
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) PCTFREE 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));
```

如下语句在 emp 表上创建一个局部索引，该表的索引段分布在表空间 ts7, ts8 和 ts9。

```
CREATE INDEX emp_ix ON emp(deptno)
LOCAL STORE IN (ts7, ts8, ts9);
```

该局部索引在如下方面与基础表是等同分区的：

- 分区数量与基础表相同。
- 每个索引分区的子分区数量与相应的基础表分区相同。
- 在给定的基础表的子分区中的行的索引项，被存储在该索引的相应的子分区中。

17.3.5 创建分区索引结构表

可以对索引结构表使用范围分区或散列分区方法。但只有范围分区索引结构表才能包含 LOB 数据类型的列。创建范围分区或散列分区索引结构表的过程与创建普通表相似，区别在于：

- 创建该表时要指定 ORGANIZATION INDEX 子句，需要时还要指定 INCLUDING 和 OVERFLOW 子句。
- PARTITION 或 PARTITIONS 子句可以有 OVERFLOW 下级子句，它允许你在分区层次上指定溢出段的属性。

指定 OVERFLOW 子句使得溢出数据段本身是与主键索引段等同分区的。因此，对具有溢出数据段的分区索引结构表而言，每个分区具有一个索引段和一个溢出数据段。

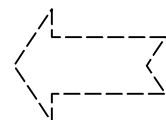
索引结构表的分区列的集合必须是主键列的子集。因为索引结构表的行是按表的主键索引存储的，所以分区方法的标准对可用性有作用。通过将分区键选成主键的子集，插入操作就只需要校验在单个分区中的主键的唯一性，因此，对分区的维护就互不依赖了。

对索引结构表的辅助索引的支持相似于对普通表的支持，但是，某些维护操作不会将全局索引标记成 UNUSABLE，这不同于普通表。

17.3.5.1 创建范围分区索引结构表

可以用范围分区方法对索引结构表，及其辅助索引进行分区。如下例子创建一个范围分区索引结构表 sales。INCLUDING 子句指定将 week_no 列之后的所有列都存储在溢出段中。每个分区有一个溢出段，都存储在相同的表空间（overflow_here）中。可选地，可以在各个分区层次上指定 OVERFLOW TABLESPACE，这样，某个或所有溢出段可以有不同的 TABLESPACE 属性。

```
CREATE TABLE sales(acct_no NUMBER(5),
acct_name CHAR(30),
amount_of_sale 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 OVERFLOW TABLESPACE overflow_ts2,
...
PARTITION VALUES LESS THAN (MAXVALUE)
TABLESPACE ts13);

```

17.3.5.2 创建散列分区索引结构表

对索引结构表进行分区其他做法是使用散列分区方法。如下例子用散列分区方法对索引结构表 sales 进行分区。

```

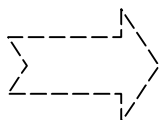
CREATE TABLE sales(acct_no NUMBER(5),
acct_name CHAR(30),
amount_of_sale NUMBER(6),
week_no INTEGER,
sale_details VARCHAR2(1000),
PRIMARY KEY (acct_no, acct_name, week_no))
ORGANIZATION INDEX
INCLUDING week_no
OVERFLOW
PARTITION BY HASH (week_no)
PARTITIONS 16
STORE IN (ts1, ts2, ts3, ts4)
OVERFLOW STORE IN (ts3, ts6, ts9);

```

注意：因为一个设计成熟的散列函数被认为会将行在各个分区中作很好的平衡分布，所以，更改一个行的主键列就很可能将该行移动到不同的分区。因此，建议在创建一个具有可变的分区键的散列分区索引结构表时，明确指定 ROW MOVEMENT ENABLE 子句。默认时该特征是停用的。

17.3.6 用于多个数据块大小的分区限制

当在具有多个数据块大小的表空间的数据库中创建分区对象时要留意。将分区对象存储到这些表空间中时，会受到某些限制。特别地，如下物化的所有分区，必须存储在具有



相同数据块大小的表空间中：

- 常规表
- 索引
- 索引结构表的主键索引段
- 索引结构表的溢出段
- 在外存储的 LOB 列

因此：

- 每个常规表的所有分区，必须被存储在具有相同数据块大小的表空间中。
- 每个索引结构表的所有主键索引分区，必须被存储在具有相同数据块大小的表空间中，并且，那个表的所有溢出分区也必须被存储在具有相同数据块大小的表空间中。当然，索引分区和溢出分区可以存储在具有不同数据块大小的表空间中。
- 每个索引（全局或局部）的每个分区必须被存储在具有相同数据块大小的表空间中。当然，定义在同一个对象上的不同的索引的分区可以被存储在具有不同数据块大小的表空间中。
- 每个 LOB 列的每个分区必须被存储在具有相同数据块大小的表空间中。当然，不同的 LOB 列可以被存储在具有不同数据块大小的表空间中。

当创建或更改分区表或分区索引时，为每个物化的分区和子分区明确指定的所有表空间，都必须具有相同的数据块大小。如果没有为一个物化明确指定存储表空间，那么，Oracle 默认使用的表空间，就必需具有相同的数据块大小。因此，必须在分区对象的每个层次上留意默认表空间。

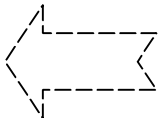
17.4 维护分区表

本节描述如何对表和索引进行分区和子分区的维护操作。

表 17-1 列出了可以对表的分区（或子分区）进行的维护操作，并且，对每种分区类型，列出了用于执行维护操作的 ALTER TABLE 语句的特殊子句。

表 17-1 用于表的分区的 ALTER TABLE 维护操作

维护操作	范围	散列	列表	组合
增加分区	ADD	ADD	ADD PARTITION	ADD PARTITION
	PARTITION	PARTITION		MODIFY PARTITION ... ADD SUBPARTITION
合并分区	n/a	COALESCE PARTITION	n/a	MODIFY PARTITION ... COALESCE SUBPARTITION
删除分区	DROP	n/a	DROP	DROP PARTITION
	PARTITION		PARTITION	
交换分区	EXCHANGE	EXCHANGE	EXCHANGE	EXCHANGE PARTITION
	PARTITION	PARTITION	PARTITION	EXCHANGE SUBPARTITION
融合分区	MERGE PARTITIONS	n/a	MERGE PARTITIONS	MERGE PARTITIONS



(续表)

维护操作	范围	散列	列表	组合
修改分区： 增加值	n/a	n/a	MODIFY PARTITION ... ADD VALUES	n/a
修改分区： 删除值	n/a	n/a	MODIFY PARTITION ... DROP VALUES	n/a
修改默认 属性	MODIFY DEFAULT ATTRIBUTES	MODIFY DEFAULT ATTRIBUTES	MODIFY DEFAULT ATTRIBUTES	MODIFY DEFAULT ATTRIBUTES MODIFY DEFAULT ATTRIBUTES FOR PARTITION
修改分区 的真实属 性	MODIFY PARTITION	MODIFY PARTITION	MODIFY PARTITION	MODIFY PARTITION MODIFY SUBPARTITION
移动分区	MOVE PARTITION	MOVE PARTITION	MOVE PARTITION	MOVE PARTITION
更名分区	RENAME PARTITION	RENAME PARTITION	RENAME PARTITION	RENAME PARTITION RENAME SUBPARTITION
分割分区	SPLIT PARTITION	n/a	SPLIT PARTITION	SPLIT PARTITION
舍弃分区	TRUNCATE PARTITION	TRUNCATE PARTITION	TRUNCATE PARTITION	TRUNCATE PARTITION TRUNCATE SUBPARTITION

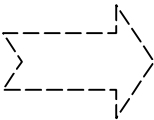
表 17-2 列出了可以对索引分区进行的维护操作,并且指明了可以对何种类型的索引(全局或局部)执行这些维护操作。显示了可用于维护操作的 ALTER INDEX 子句。

全局索引不反映基础表的结构,如果要分区,就只能对其进行范围分区。范围分区索引共享某些(并非全部)可以在范围分区表上进行的分区维护操作。

因为局部索引反映基础表的结构,因此当对表的分区和子分区进行维护活动时,会自动地对局部索引的分区进行维护。因此不太需要对局部索引的分区进行维护,选项也很少。

表 17-2 用于索引分区的 ALTER INDEX 维护操作

维护操作	索引 类型	索引分区的类型		
		范围	散列/列表	组合
删除索引分 区	全局	DROP PARTITION		
	局部	n/a	n/a	n/a
修改索引分 区的默认属 性	全局	MODIFY DEFAULT ATTRIBUTES		



(续表)

维护操作	索引类型	索引分区的类型		
		范围	散列/列表	、
	局部	MODIFY DEFAULT ATTRIBUTES	MODIFY DEFAULT ATTRIBUTES	MODIFY DEFAULT ATTRIBUTES MODIFY DEFAULT ATTRIBUTES FOR PARTITION
修改索引分区的真实属性	全局	MODIFY PARTITION		
	局部	MODIFY PARTITION	MODIFY PARTITION	MODIFY PARTITION MODIFY SUBPARTITION
重建索引分区	全局	REBUILD PARTITION		
	局部	REBUILD PARTITION	REBUILD PARTITION	REBUILD SUBPARTITION
更名索引分区	全局	RENAME PARTITION		
	局部	RENAME PARTITION	RENAME PARTITION	RENAME PARTITION RENAME SUBPARTITION
分割索引分区	全局	SPLIT PARTITION		
	局部	n/a	n/a	n/a

注意：下面几节讨论对分区表的维护操作。并讨论维护操作对索引或索引分区的可用性的影响，包括：

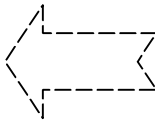
- 只有非空的索引和索引分区可以被标记成 UNUSABLE。如果是空的，则 USABLE/ UNUSABLE 就处于释放状态。
- 只有处于 USABLE 状态的索引或索引分区可以被后续的 DML 语句更改。

17.4.1 自动更新全局索引

在讨论对分区表和分区索引的各种维护操作之前，最好先讨论一下可以在 ALTER TABLE 语句中指定的 UPDATE GLOBAL INDEX 子句的作用。

默认时，对分区表的许多表维护操作会使全局索引不可用（标记成 UNUSABLE）。那么，就必须重建整个全局索引或其全部分区（如果已被分区）。Oracle 允许在用于维护操作的 ALTER TABLE 语句中，指定 UPDATE GLOBAL INDEXES，来重载这个默认特性。指定这个子句也就告诉 Oracle，当它执行维护操作的 DDL 语句时，更新全局索引。这提供了如下好处：

- 在操作基础表的同时更新全局索引。这就不需要后来单独地重建全局索引。
- 因为没有被标记成 UNUSABLE，所以全局索引的可用性更高了。甚至正在执行分区的 DDL 语句时，仍然可用索引来访问表中的其他分区。



- 避免了查询所有失效的全局索引的名字，以便重建它们。

另外，在指定 UPDATE GLOBAL INDEXES 之前，还要考虑如下性能因素：

- 因为要更新事先被标记成 UNUSABLE 的索引，所以分区的 DDL 语句要执行更长时间。当然，这要与先不更新索引而执行 DDL，然后再重建索引所花的时间做个比较。一个适用的规则是，如果分区的大小小于表的大小的 5%，则更新索引更快一点。
- DROP、TRUNCATE 和 EXCHANGE 操作也不那么快了。同样，这必须与先执行 DDL，然后再重建所有全局索引所花的时间做个比较。
- 要登记对索引的更新，并产生重做记录和撤消记录。重建整个索引时，可选择 NOLOGGING。
- 重建整个索引产生一个更有效的索引，因为这更利于使用空间。再者，重建索引时允许修改存储选项。

注意：分区索引结构表不支持 UPDATE GLOBAL INDEXES 子句。

如下操作支持 UPDATE GLOBAL INDEXES 子句：

- ADD PARTITION | SUBPARTITION（仅限散列分区）
- COALESCE PARTITION | SUBPARTITION
- DROP PARTITION
- EXCHANGE PARTITION | SUBPARTITION
- MERGE PARTITION
- MOVE PARTITION | SUBPARTITION
- SPLIT PARTITION
- TRUNCATE PARTITION | SUBPARTITION

17.4.2 增加分区

本节描述如何在分区表中增加新分区，并解释为何不能专门给全局分区索引或局部索引增加分区。

17.4.2.1 给范围分区表增加分区

用 ALTER TABLE ... ADD PARTITION 语句将新分区增加到“高”端（最后一个现存分区的后面）。要想在表的开始或中间增加分区，就要使用 SPLIT PARTITION 子句。

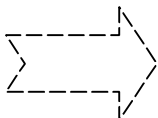
例如，考虑表 sales，它包括当前月份和前 12 个月份的数据。在 1999 年 1 月 1 日，你为 1 月份增加一个分区，并将其存储在表空间 tsx 中。

```
ALTER TABLE sales
ADD PARTITION jan96 VALUES LESS THAN ( '01-FEB-1999' )
TABLESPACE tsx;
```

与范围分区表相关的局部索引和全局索引仍然可用。

17.4.2.2 给散列分区表增加分区

当给散列分区表增加一个新分区时，Oracle 会在新分区中放入，来自一个现存分区（由



Oracle 选定的) 的, 由散列函数决定的, 被重新做过散列操作后的行。

下面的例子显示了给表 `scubagear` 增加一个散列分区的两种方法。第一个语句增加一个新散列分区, 其分区名是由系统产生的, 并被放置在表的默认表空间中。第二个语句也增加一个新散列分区, 但分区被明确命名为 `p_named`, 并被创建到表空间 `gear5` 中。

```
ALTER TABLE scubagear ADD PARTITION;  
  
ALTER TABLE scubagear  
ADD PARTITION p_named TABLESPACE gear5;  
按下表所解释的那样, 索引可能被标记成 UNUSABLE:
```

表类型	索引特性
普通的 (堆积的)	▪ 用于新分区的和用于现存分区的, 重新划分行的局部索引被标记成 UNUSABLE, 并必须被重建。
	▪ 除非指定 UPDATE GLOBAL INDEXES, 否则所有全局索引或分区全局索引的所有分区, 被标记成 UNUSABLE, 并必须被重建
索引结构的	▪ 局部索引与普通表一样
	▪ 所有全局索引保持可用

17.4.2.3 给列表分区表增加分区

如下语句解释了, 给列表分区表增加一个新分区。在本例中, 给将增加的分区指定了物理属性和 NOLOGGING。

```
ALTER TABLE sales_by_region  
ADD PARTITION region_nonmainland VALUES ('HI', 'PR')  
STORAGE (INITIAL 20K NEXT 20K) TABLESPACE tbs_3  
NOLOGGING;
```

描述将要增加的分区的文字值的集合中的任何值, 都不能在该表的任何其他分区中存在。

与列表分区表相关的局部索引和全局索引仍然可用。

17.4.2.4 给组合分区表增加分区

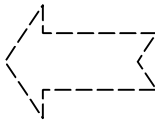
可以在范围分区层次和散列子分区层次两个层次上增加分区。

增加分区 给组合分区表增加一个范围分区与前面在“给范围分区表增加分区”中描述的一样。然而, 可以使用 SUBPARTITIONS 子句, 来增加指定数量的子分区, 或使用 SUBPARTITION 子句来给特殊的子分区命名。如果没有指定 SUBPARTITIONS 或 SUBPARTITION 子句, 则该分区继承用于子分区的表层次的默认值。

本例给表 `sales` 增加一个范围分区 `q1_2000`, 用于放置 2000 年第一季度的数据。在表空间 `tbs5` 中有 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 可以用 ALTER TABLE 语句的 MODIFY PARTITION...ADD



SUBPARTITION 子句，给组合分区表增加散列子分区。在新增加的子分区中，放入来自相同分区的其他子分区的，由散列函数决定的，被重新做过散列操作后的行。

在如下例子中，存储在表空间 us1 中的，新散列子分区 us_loc5，被增加到表 diving 中的范围分区 locations_us 中。

```
ALTER TABLE diving MODIFY PARTITION locations_us
ADD SUBPARTITION us_locs5 TABLESPACE us1;
```

必须重建对应于已增加的，和重新做过散列操作的子分区的，局部索引子分区。除非指定 UPDATE GLOBAL INDEXES，否则，所有全局索引，或分区全局索引的所有分区，将被标记成 UNUSABLE，并必须被重建。

17.4.2.5 增加索引分区

不能明确地给局部索引增加分区。替代方法是，只有当为基础表增加分区时，才能给局部索引增加了一个新分区。特别是，当表中定义了一个局部索引，并发布 ALTER TABLE 语句来增加分区时，也就给该局部索引增加了一个匹配的分区。因为 Oracle 会给新的索引分区分名字和默认物理存储属性，所以在完成 ADD 操作之后，可能想要给它们重命名或更改它们。

因为最高分区总是具有 MAXVALUE 分区边界，所以不能给全局索引增加分区。如果要想增加一个新的最高分区，就要使用 ALTER INDEX...SPLIT PARTITION 语句了。

17.4.3 合并分区

合并分区是一种减少散列分区表中的分区数量，或组合分区表中的子分区数量的一种方法。合并散列分区时，由散列函数决定将其内容重新划分到一个或多个剩余的分区中。由 Oracle 选择要被合并的特殊的分区，并在重新划分其内容后将其删除。

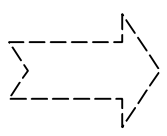
按下表所解释的那样，索引可能被标记成 UNUSABLE:

表类型	索引特性
普通的（堆积的）	▪ 对应于被选中分区的任何局部索引分区也被删除。
	▪ 对应于一个或多个兼并的分区的局部索引分区被标记成 UNUSABLE，并必须被重建。
索引结构的	▪ 除非指定 UPDATE GLOBAL INDEXES，否则，所有全局索引，或分区全局索引的所有分区，都被标记成 UNUSABLE，并必须被重建。
	▪ 像上面所注，某些局部索引被标记成 UNUSABLE。
	▪ 所有全局索引仍然可用。

17.4.3.1 合并散列分区表中的分区

用 ALTER TABLE ... COALESCE PARTITION 语句，来合并散列分区表中的分区。如下语句靠合并一个分区来减少该表的一个分区。

```
ALTER TABLE ouu1
COALESCE PARTITION;
```



17.4.3.2 合并组合分区表中的分区

如下语句将分区 `us_locations` 中的一个子分区的内容，划分到相同分区的一个或多个子分区中（由散列函数决定）。基本上说，该操作是，在第 17 章的“增加子分区”中讨论的，`MODIFY PARTITION ... ADD SUBPARTITION` 子句的逆操作。

```
ALTER TABLE diving MODIFY PARTITION us_locations
COALESCE SUBPARTITION;
```

17.4.4 删除分区

可以删除范围、组合或列表分区表中的分区。对散列分区表，或组合分区表的散列子分区来说，必须用执行合并操作来替代删除操作。

17.4.4.1 删除表分区

使用 `ALTER TABLE ... DROP PARTITION` 语句来删除表分区。如果想保留分区中的数据，则用 `MERGE PARTITION` 语句来代替 `DROP PARTITION` 语句。

如果给表定义过局部索引，这个语句还会从局部索引中删除匹配的分区或子分区。所有的全局索引，或分区全局索引的所有分区都将被标记成 `UNUSABLE`，除非满足如下条件之一：

- 指定了 `UPDATE GLOBAL INDEXES`（不能对索引结构表做这个指定）
- 要删除的分区或其子分区是空的

注意：不能删除表中唯一的分区。那么，就删除该表。

如下几节包含删除表分区的某些方案。

从包含数据和全局索引的表中删除分区 如果在分区中包含数据和在表上定义了一个或多个全局索引，就用下面的方法之一来删除该表的分区。

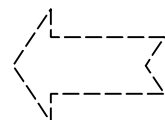
方法 1：

在发布 `ALTER TABLE ... DROP PARTITION` 语句期间先不管全局索引。然后，必须重建任何全局索引（无论是否分区），因为索引（或索引分区）可能已被标记成了 `UNUSABLE`。下面的语句先从 `sales` 表中删除分区 `dec98`，然后重建其全局非分区索引。

```
ALTER TABLE sales DROP PARTITION dec98;
ALTER INDEX sales_area_ix REBUILD;
```

假如索引 `sales_area_ix` 是一个范围分区全局索引，那么，就需要重建该索引的所有分区。进一步说，不可能在一条语句中重建一个索引的所有分区。必须为索引中的每个分区写一条单独的 `REBUILD` 语句。下面的语句重建索引分区 `jan99_ix`, `feb99_ix`, `mar99_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;
```



这种方法最适合于要删除的分区中包含该表中很多数据的大表。

方法 2:

在发布 ALTER TABLE ... DROP PARTITION 语句之前,先发布 DELETE 语句来删除分区中的所有行。DELETE 语句更改全局索引,并启动触发器和产生重做及撤消日志。

例如,要删除分区边界为 10000 的第一个分区,就发布如下语句:

```
DELETE FROM sales WHERE TRANSID < 10000;
ALTER TABLE sales DROP PARTITION dec98;
```

这种方法最适合于小表,或者要删除的分区只包含该表中较少数据的大表。

方法 3:

在 ALTER TABLE 语句中指定 UPDATE GLOBAL INDEXES。这导致在删除分区的同时更改全局索引。

```
ALTER TABLE sales DROP PARTITION dec98
UPDATE GLOBAL INDEXES;
```

删除包含数据和参照完整性约束的分区 如果分区包含数据,并且该表具有参照完整性约束,那就选择下面的方法之一来删除该表的分区。该表只有局部索引,所以不需重建任何索引。

方法 1:

停用完整性约束,发布 ALTER TABLE ... DROP PARTITION 语句,然后再启用完整性约束:

```
ALTER TABLE sales
DISABLE CONSTRAINT dname_sales1;
ALTER TABLE sales DROP PARTITION dec98;
ALTER TABLE sales
ENABLE CONSTRAINT dname_sales1;
```

这种方法最适合于要删除的分区中包含该表中很多数据的大表。

方法 2:

在发布 ALTER TABLE ... DROP PARTITION 语句之前,先发布 DELETE 语句来删除分区中的所有行。DELETE 语句实施参照完整性约束,并启动触发器和产生重做及撤消日志。

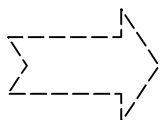
```
DELETE FROM sales WHERE TRANSID < 10000;
ALTER TABLE sales DROP PARTITION dec94;
```

这种方法最适合于小表,或者要删除的分区只包含该表中较少数据的大表。

17.4.4.2 删除索引分区

可以明确地删除局部索引的分区。要么,只有当删除基础表的分区时才删除局部索引分区。

如果全局索引是空的,就可以发布 ALTER INDEX ... DROP PARTITION 语句,来明确地删除它。但是,如果全局索引分区中包含数据,那么,删除该分区就将导致下一个最



高分区被标记成 UNUSABLE。例如，如想删除索引分区 P1，并且 P2 是下一个最高分区，就必须发布如下语句：

```
ALTER INDEX npr DROP PARTITION P1;
ALTER INDEX npr REBUILD PARTITION P2;
注意：不能删除全局索引中的最高分区。
```

17.4.5 交换分区

可以通过交换数据段的方法，将一个分区（或子分区）转换成一个非分区表，或将一个非分区表转换成一个分区表的一个分区（或子分区）。还可以将一个散列分区表转换成一个组合分区表的一个分区，或将组合分区表的分区转换成一个散列分区表。

当有一个使用非分区表的应用，又想将该非分区表转换成一个分区表的分区时，交换表分区就非常有用。例如，你可能已经有了打算将其融合到分区表中的分区视图了。

当与可传输表空间一起使用时，交换分区还有助于高速数据装载。

交换分区时会保留登记属性。可以选择指定是否也交换局部索引，是否为适当的映射而对行进行校验。除非指定 UPDATE GLOBAL INDEXES（不能用于索引结构表），否则 Oracle 会将正在交换分区的表上的全局索引、或所有全局索引分区，标记成 UNUSABLE。

正在交换的表上的任何全局索引或全局索引分区都被标记成 UNUSABLE。

17.4.5.1 交换范围、散列、列表分区

要想将范围、散列、列表分区表的分区转换成非分区表，或反之，就要使用 ALTER TABLE ... EXCHANGE PARTITION 语句。下面是一个将一个分区转换成一个非分区表的例子。本例中，表 stocks 可以是范围、散列或列表分区表。

```
ALTER TABLE stocks
EXCHANGE PARTITION p3 WITH stock_table_3;
```

17.4.5.2 将散列分区表转换成组合分区

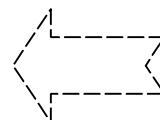
本例要将整个散列分区表，及其所有子分区，转换成组合分区表的范围分区及其所有散列子分区。如下例所示。

首先，创建一个散列分区表

```
CREATE TABLE t1 (i NUMBER, j NUMBER)
PARTITION BY HASH(i)
(PARTITION p1, PARTITION p2);
```

填充该表，然后按下面所示创建组合分区表：

```
CREATE TABLE t2 (i NUMBER, j NUMBER)
PARTITION BY RANGE(j)
SUBPARTITION BY HASH(i)
(PARTITION p1 VALUES LESS THAN (10)
SUBPARTITION t2_pls1
SUBPARTITION t2_pls2,
```



```
PARTITION p2 VALUES LESS THAN (20)
SUBPARTITION t2_p2s1
SUBPARTITION t2_p2s2));
重要的事是，表 t1 的分区键与表 t2 的子分区键相同。
为了将数据从 t1 迁移到 t2 并校验行，要使用如下语句：
ALTER TABLE t1 EXCHANGE PARTITION p1 WITH TABLE t2
WITH VALIDATION;
```

17.4.5.3 交换组合分区表的子分区

要想将组合分区表的散列子分区转换成非分区表，或反之，就要使用 ALTER TABLE ... EXCHANGE SUBPARTITION 语句。如下例子将表 sales 的子分区 q3_1999_s1 转换成非分区表 q3_1999。局部索引分区被转换成 q3_1999 上对应的索引。

```
ALTER TABLE sales EXCHANGE SUBPARTITIONS q3_1999_s1
WITH TABLE q3_1999 INCLUDING INDEXES;
```

17.4.6 融合分区

要想将两个分区的内容融合到一个分区，就要使用 ALTER TABLE ... MERGE PARTITION 语句。两个原始分区及其任何对应的局部索引都将被删除。

不能将该语句用于散列分区表或组合分区表的散列子分区。

除非涉及到的分区或子分区是空的，否则，按下表所解释的那样，索引可能被标记成 UNUSABLE:

表类型	索引特性
普通的（堆积的）	- Oracle 将所有结果所对应的局部索引分区或子分区都标记成 UNUSABLE。 - 除非指定 UPDATE GLOBAL INDEXES，否则，所有全局索引，或分区全局索引的所有分区都被标记成 UNUSABLE，并必须被重建。
索引结构的	- Oracle 将所有结果所对应的局部索引分区或子分区都标记成 UNUSABLE。 - 所有全局索引仍然可用。

17.4.6.1 融合范围分区

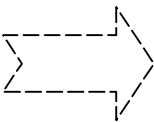
允许将两个相邻的范围分区中的内容融合到一个分区。不相邻的范围分区不能融合。结果分区继承两个被融合分区的较高的上边界。

融合范围分区的一个原因是希望联机地将历史数据保留在一个大分区中。例如，有日分区，以及将最早的日分区累积到周分区，还可以将最早的周分区积累到月分区，等等。

如下脚本是融合范围分区的一个例子。

首先，创建一个分区表并创建局部索引。

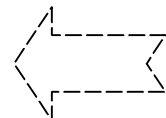
```
-- Create a Table with four partitions each on its own tablespace
-- Partitioned by range on the data column.
```



```

--
CREATE TABLE four_seasons
(
  one DATE,
  two VARCHAR2(60),
  three NUMBER
)
PARTITION BY RANGE ( one )
(
  PARTITION quarter_one
  VALUES LESS THAN ( TO_DATE('01-apr-1998','dd-mon-yyyy'))
  TABLESPACE quarter_one,
  PARTITION quarter_two
  VALUES LESS THAN ( TO_DATE('01-jul-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-jan-1999','dd-mon-yyyy'))
  TABLESPACE quarter_four
)
/
--
-- Create local PREFIXED index on Four_Seasons
-- Prefixed because the leftmost columns of the index match the
-- Partition key
--
CREATE INDEX i_four_seasons_l 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
)
/
  下一步，融合分区。
--

```



```
-- Merge the first two partitions
--
ALTER TABLE four_seasons
MERGE PARTITIONS quarter_one, quarter_two INTO PARTITION quarter_two
/
然后，为受影响的分区重建局部索引。
-- 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
/
```

17.4.6.2 融合列表分区

融合列表分区时，要融合的分区可以是任何两个分区。因为列表分区方法并不假设分区的任何排序，所以这两个分区不需要像范围分区中那样是相邻的。结果分区由两个原始分区中的所有数据组成。

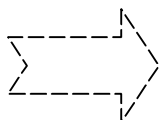
如下语句用列表分区方法，将一个列表分区表的两个分区融合成一个分区，该分区继承了它们两者在表层次上的所有默认属性，除了在语句中指定的 `PCTFREE` 和 `MAXEXTENTS` 之外。

```
ALTER TABLE sales_by_region
MERGE PARTITIONS sales_northwest, sales_southwest
INTO PARTITION sales_west
PCTFREE 50 STORAGE(MAXEXTENTS 20);
两个原始分区的值列表被指定为：
PARTITION sales_northwest VALUES ('WA','OR','WY','MT')
PARTITION sales_southwest VALUES ('AZ','NM','CO')
而结果 sales_west 分区的值列表是由这两个分区的值列表的并集组成，即：
('WA','OR','WY','MT','AZ','NM','CO')
```

17.4.6.3 融合范围组合分区

融合范围组合分区时，子分区被重做散列操作，形成或者在 `SUBPARTITIONS` 或 `SUBPARTITION` 子句中指定的子分区数量，或者（如没有包含这些子句时）就用表层次上默认的子分区数量。

要注意，分割范围组合分区（在第17章的“分割范围组合分区”中讨论）与融合两个范围组合分区两者，对属性的继承是有区别的。分割分区时，因为仅有一个双亲，所以新的分区可以从原始分区中继承属性。然而，当融合分区时，因为有两个双亲，所以必须继



承表层次上的默认属性，即，新分区不能继承其一而损害其二的利益。

如下例子融合两个范围组合分区：

```
ALTER TABLE all_seasons
MERGE PARTITIONS quarter_1, quarter_2 INTO PARTITION quarter_2
SUBPARTITIONS 8;
```

17.4.7 修改默认属性

可以修改表，或组合分区表的分区的默认属性。修改默认属性时，新属性只影响以后创建的分区，或子分区。创建新分区或子分区时还可以特意重载默认值。

17.4.7.1 修改表的默认属性

可以用 ALTER TABLE 语句的 MODIFY DEFAULT ATTRIBUTES 子句来修改会被范围分区、列表分区或散列分区继承的默认属性。如下例子修改表 emp 的 PCTFREE 属性的默认值，以便创建任何新分区。

```
ALTER TABLE emp
MODIFY DEFAULT ATTRIBUTES PCTFREE 25;
```

对散列分区表而言，只能修改 TABLESPACE 属性。

17.4.7.2 修改分区的默认属性

为了修改在创建子分区时要继承的默认属性，就使用 ALTER TABLE ... MODIFY DEFAULT ATTRIBUTES FOR PARTITION 语句。如下语句修改 TABLESPACE 属性，该表空间将存储组合分区表 emp 中的分区 p1 的以后的子分区。

```
ALTER TABLE emp
MODIFY DEFAULT ATTRIBUTES FOR PARTITION p1 TABLESPACE ts1;
```

因为所有子分区必须共享除 TABLESPACE 之外的相同的属性，所以它是仅能被修改的属性。

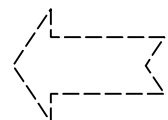
17.4.7.3 修改索引分区的默认属性

与表分区相似，可以修改会被范围分区全局索引的分区、或范围分区表、散列分区表或组合分区表的局部索引分区继承的默认属性，可以用 ALTER INDEX ... MODIFY DEFAULT ATTRIBUTES 语句来做此事。如果要修改被组合分区表的子分区继承的默认属性，就用 ALTER INDEX ... MODIFY DEFAULT ATTRIBUTES FOR PARTITION 语句来做此事。

17.4.8 修改分区的真实属性

可以修改表或索引的现存分区的属性。

不能修改 TABLESPACE 属性。要将分区或子分区移动到新的表空间，就使用 ALTER TABLESPACE ... MOVE PARTITION/SUBPARTITION 语句。



17.4.8.1 修改范围分区或列表分区的真实属性

使用 `ALTER TABLE ... MODIFY PARTITION` 语句来修改范围分区的现存属性。可以修改段属性（除 `TABLESPACE` 之外），可以分配和回收盘区，将局部索引分区标记成 `UNUSABLE`，或重建被标记成 `UNUSABLE` 的局部索引。

如果组合分区表具有范围分区，就要注意：

- 如果分配或回收盘区，该动作将在指定分区的每个子分区上执行。
- 而且，修改任何其他属性都将导致修改那个分区的所有子分区的对应的那些属性。还会修改分区层次的默认属性。为了避免修改现存子分区的属性，就要使用 `MODIFY DEFAULT ATTRIBUTES` 语句的 `FOR PARTITION` 子句。

下面是修改分区的真实属性的某些例子。

本例修改表 `sales` 的范围分区 `sales_q1` 的 `MAXEXTENTS` 存储属性：

```
ALTER TABLE sales MODIFY PARTITION sales_q1
STORAGE (MAXEXTENTS 10);
```

如下例子会将组合分区表 `scubagear` 的分区 `ts1` 中的所有局部索引分区都标记成 `UNUSABLE`：

```
ALTER TABLE scubagear MPDIFY PARTITION ts1 UNUSABLE LOCAL INDEXES;
```

17.4.8.2 修改散列分区的真实属性

仍然使用 `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都必须相同（除 `TABLESPACE` 之外），所以，可以做的事情被限制到：

- 分配新盘区
- 回收未被使用的盘区
- 将局部索引子分区标记成 `UNUSABLE`
- 重建被标记成 `UNUSABLE` 的子分区

如下例子重建所有与表 `dept` 的散列分区 `P1` 相关的，不可用的局部索引分区：

```
ALTER TABLE dept MODIFY PARTITION p1
REBUILD UNUSABLE LOCAL INDEXES;
```

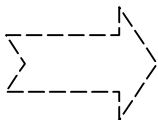
17.4.8.3 修改子分区的真实属性

可以用 `ALTER TABLE` 语句的 `MODIFY SUBPARTITION` 子句，来执行前面针对散列分区的那些动作，只是现在是在特别的组合分区表的子分区层次上进行操作。例如：

```
ALTER TABLE emp MODIFY SUBPARTITION p3_s1
REBUILD UNUSABLE LOCAL INDEXES
```

17.4.8.4 修改索引分区的真实属性

可用 `ALTER INDEX` 语句的 `MODIFY PARTITION` 子句，来修改索引分区或它的子分区的真实属性。其规则非常像用于表分区的规则，但不象 `ALTER TABLE` 语句的 `MODIFY`



PARTITION 子句，因为这里没有重建不可用索引分区下级子句，但这里有合并索引分区或它的子分区下级子句。此就是说，合并意味着融合那些可以被释放以便被重新使用的索引数据块。

还可以用 MODIFY SUBPARTITION 子句，来分配或回收用于局部索引的子分区的存储体，或将它标记成 UNUSABLE。

17.4.9 修改列表分区：增加或删除值

列表分区方法允许你选择增加或删除定义的值列表中的文字值。

17.4.9.1 修改分区：增加值

为了扩展现存分区的值列表，使用 ALTER TABLE 语句的 MODIFY PARTITION ... ADD VALUES 子句。要增加的文字值必须没有被包含在任何其他分区的值列表中。任何对应的局部索引分区的分区值列表也被对应地扩展，并且，任何全局索引，或全局索引分区或局部索引分区，仍然可用。

如下语句将新的州代码集合（'OK'，'KS'）增加到一个现存分区的列表中。

```
ALTER TABLE sales_by_region
MODIFY PARTITION region_east
ADD VALUES ('OK', 'KS');
```

17.4.9.2 修改分区：删除值

为了从现存分区的值列表中删除文字值，使用 ALTER TABLE 语句的 MODIFY PARTITION ... DROP VALUES 子句。该语句总是随着校验一起执行，即，它检查是否有任何行存在于要被删除的值的集合所对应的分区中。如果发现了这种行，那么 Oracle 就返回一条错误消息并且操作失败。需要时，在打算删除文字值之前，先用 DELETE 语句从表中删除对应的行。

注意：不能删除描述分区的值列表中的所有的文字值。那么，就必须使用 ALTER TABLE ... DROP PARTITION 语句。

任何对应的局部索引分区的值列表会反映该新的值列表，并且，任何全局索引，或全局索引分区或局部索引分区，仍然可用。

如下语句从一个现存分区的值列表中删除州代码集合（'OK'和'KS'）。

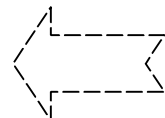
```
ALTER TABLE sales_by_region
MODIFY PARTITION region_south
DROP VALUES ('OK', 'KS');
```

注意：因为为了检查对应于要被删除的文字值的分区中的行的存在性，需要执行查询，所以最好在该表上创建一个局部前缀索引。这会加速查询和所有操作的执行速度。

17.4.10 移动分区

使用 ALTER TABLE 语句的 MOVE PARTITION 子句来：

- 重新分簇数据并减少碎片



- 将一个分区移动到另一个表空间
- 修改创建时的属性

典型地，可以使用 ALTER TABLE/INDEX ... MODIFY PARTITION 语句，只用一步，就可以修改一个分区的物理存储属性。但是，某些物理属性，如 TABLESPACE，不能用 MODIFY PARTITION 语句来修改。这时，就要用 MOVE PARTITION 子句了。

除非要被移动的分区不包含任何数据，否则，根据下表，索引可能被标记成 UNUSABLE:

表类型	索引特性
普通的（堆积的）	▪ 每个局部索引中的匹配分区被标记成 UNUSABLE。发布 MOVE PARTITION 之后必须重建这些索引 ▪ 除非指定 UPDATE GLOBAL INDEXES，否则，任何全局索引，或分区全局索引的所有分区被标记成 UNUSABLE。
索引结构的	在要被移动的分区上定义的任何局部索引或全局索引仍然可用，因为它们是基于逻辑行标识（rowids）的主键。然而对这些行标识的推测信息就不正确了。

17.4.10.1 移动表分区

使用 MOVE PARTITION 子句来移动分区。例如，为了将最活动的分区移动到它自己的磁盘上的表空间中（为了平衡 I/O）并且不登记该动作，就发布如下语句：

```
ALTER TABLE parts MOVE PARTITION depot2
TABLESPACE ts094 NOLOGGING;
```

该语句总是删除该分区的旧段并创建一个新段，甚至于在没有指定新的表空间时也这么做。

17.4.10.2 移动子分区

如下语句说明如何移动一个表的子分区中的数据。本例还指定了一个 PARALLEL 子句。

```
ALTER TABLE scuba_gear MOVE SUBPARTITION bcd_types
TABLESPACE tbs23 PARALLEL (DEGREE 2);
```

17.4.10.3 移动索引分区

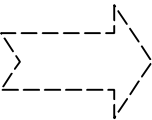
对于普通表，ALTER TABLE ... MOVE PARTITION 语句将全局索引的所有分区都标记成 UNUSABLE。可以用 ALTER INDEX ... REBUILD PARTITION 语句来逐个重建每个分区，以便重建整个索引。可以同时进行这些重建工作。

还可以简单地删除索引并重建该索引。

17.4.11 重建索引分区

重建索引分区的某些原因如下：

- 为了恢复空间和改善性能
- 为了修复由于媒体故障而导致损坏的索引分区



- 为了在用 Import 或 SQL*Loader 装载完基础表分区之后重建局部索引分区
- 为了重建被标记成 UNUSABLE 的索引分区

如下几节讨论重建索引分区和子分区的方法。

17.4.11.1 重建全局索引分区

可以用两种方法来重建全局索引分区

①发布 ALTER INDEX ... REBUILD PARTITION 语句，重建每个分区（可以同时运行重建）。

②删除整个全局索引，重建该全局索引。

注意：第二种方法更有效，因为只扫描一次表。

对于在具有全局索引的分区表上进行的大多数维护操作而言，可以选择通过在 DDL 语句中指定 UPDATE GLOBAL INDEXES 来避免重建全局索引的工作。

17.4.11.2 重建局部索引分区

如下所示，可以用 ALTER INDEX 或者 ALTER TABLE 语句来重建局部索引：

- ALTER INDEX ... REBUILD PARTITION/SUBPARTITION

该语句无条件地重建索引分区或子分区

- ALTER TABLE ... MODIFY PARTITION/SUBPARTITION ... REBUILD UNUSABLE LOCAL INDEXES

该语句在给定表分区或子分区中寻找所有不可用的索引并重建这些索引。该语句只重建被标记成 UNUSABLE 的索引分区。

用更改索引来重建分区 ALTER INDEX ... REBUILD PARTITION 语句重建一个索引的一个分区。它不能用于组合分区表。重建索引时，还可以选择将该分区移动到一个新的表空间中或修属性。

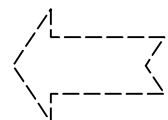
对于组合分区表，使用 ALTER INDEX ... REBUILD SUBPARTITION 语句来重建一个索引的一个子分区。可以将该子分区移动到另一个表空间中或指定一个并行子句。如下语句重建一个表上的一个局部索引的一个子分区，并将该索引的子分区移动到另一个表空间中。

```
ALTER INDEX scuba
REBUILD SUBPARTITION bcd_types
TABLESPACE tbs23 PARALLEL (DEGREE 2);
```

用更改表来重建索引分区 ALTER TABLE ... MODIFY PARTITION 语句的 REBUILD UNUSABLE LOCAL INDEXES 子句不允许给被重建的索引分区指定任何新的属性。如下例子在表 scubagear 的分区 p1 中查找和重建任何不可用的局部索引分区。

```
ALTER TABLE scubagear
MODIFY PARTITION p1 REBUILD UNUSABLE LOCAL INDEXES;
```

为了重建不可用的局部索引子分区，可以使用对应的 ALTER TABLE ... MODIFY SUBPARTITION 子句。



17.4.12 重命名分区

可以重命名表和索引两者的分区和子分区的名字。给分区重命名的一个原因是为了起一个有意义的名字，用以替换在另外的维护操作中赋给该分区的默认的系统名字。

17.4.12.1 重命名表分区

用 ALTER TABLE ... RENAME PARTITION 语句来重命名范围分区、散列分区、或列表分区的名字。例如：

```
ALTER TABLE scubagear RENAME PARTITION sys_p636 TO tanks;
```

17.4.12.2 重命名表子分区

相似地，可以给表的子分区起一个新名字。这时，使用 ALTER TABLE ... RENAME SUBPARTITION 语法。

17.4.12.3 重命名索引分区

可以用相似的方法重命名索引分区和索引子分区的名字，但使用 ALTER INDEX 语法。
重命名一个索引分区 用 ALTER INDEX ... RENAME PARTITION 语句来更改一个索引分区的名字。

重命名一个索引子分区 如下语句简单地表示了如何更改一个子分区的名字，该子分区具有一个系统产生的名字，它是在基础表上增加一个分区时产生的：

```
ALTER INDEX scuba RENAME SUBPARTITION sys_subp3254 TO bcd_types;
```

17.4.13 分割分区

ALTER TABLE 或 ALTER INDEX 语句的 SPLIT PARTITION 子句被用于将一分区中的内容重新划分成两个新的分区。当一个分区变得太大以至于要用很长时间才能完成备份、恢复、或维护操作时，就应考虑做分割分区的工作。还可以用 SPLIT PARTITION 子句来重新划分为 I/O 负载。

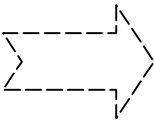
这个语句不能用于散列分区或子分区。

除非要分割的分区不包含任何内容，否则，根据下表，索引可能被标记成 UNUSABLE：

表类型	索引特性
普通的（堆积的）	- Oracle 将每个局部索引中的新分区（有两个）标记成 UNUSABLE。 - 除非指定了 UPDATE GLOBAL INDEXES，否则，任何全局索引、或分区全局索引的所有分区都被标记成 UNUSABLE，并必须重建。
索引结构的	- Oracle 将每个局部索引中的新分区（有两个）标记成 UNUSABLE。 - 所有全局索引仍然可用。

17.4.13.1 分割范围分区表的分区

用 ALTER TABLE ... SPLIT PARTITION 语句来分割范围分区。可选地，可以给分割产生的两个分区指定新的属性。如果在该表上定义了局部索引，这条语句也分割在每个局部索引中的匹配的分区的。



在如下例子中, `fee_katy` 是表 `vet_cats` 的一个分区, 它有一个局部索引 `jaf1`。在该表上还有一个全局索引 `vet`。Vet 包含两个分区, `vet_parta` 和 `vet_partb`。

为了分割分区 `fee_katy`, 并重建索引分区, 发布如下语句:

```
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: 如果没有指定新分区名字, 则 Oracle 赋予的名字的形式为 `SYS_pn`。可检查数据字典来找到赋给该新局部索引分区名字。也许想给它们更名。没有指定的任何属性都从原始分区中继承。

17.4.13.2 分割列表分区表的分区

用 `ALTER TABLE ... SPLIT PARTITION` 语句来分割列表分区。`SPLIT PARTITION` 子句允许你为那些具有对应的分区键值, 被插入到第一个新分区中的行, 指定一个文字值的值列表。原始分区中剩余的行被插入到第二个分区中。

可以选择为分割所产生的两个分区指定新的属性。

如下语句将分区 `region_east` 分割成 2 个分区:

```
ALTER TABLE sales_by_region
SPLIT PARTITION region_east VALUES ('CT', 'VA', 'MD')
INTO
( PARTITION region_east_1
PCTFREE 25 TABLESPACE tbs2,
PARTITION region_east_2
STORAGE (NEXT 2M PCTINCREASE 25))
PARALLEL 5;
```

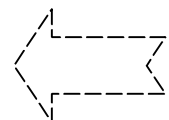
原始 `region_east` 分区的文字值列表被指定为:

```
PARTITION region_east VALUES ('CT','VA','MD','NY','NH','ME','VA','PA','NJ')
两个新分区的是:
```

- `region_east_1` 的文字值列表是('CT','VA','MD')
 - `region_east_2` 继承剩余的文字值列是, 是 ('NY', 'NH', 'ME', 'VA', 'PA', 'NJ')
- 各个分区具有在分区层次上指定的新的物理属性, 该操作使用程度 5 的并行机制执行。

17.4.13.3 分割范围组合分区

这与融合范围组合分区相反。分割范围组合分区时, 新的子分区被重做散列操作, 形成在 `SUBPARTITIONS` 子句或者在 `SUBPARTITION` 子句中指定的子分区的数量。或者,



如果没有包含这些子句，新的分区就继承正被分割的分区中的子分区（和表空间）的数量。

注意，分割范围组合分区和融合两个范围组合分区，在属性的继承上是有区别的。分割分区时，因为只有一个双亲，所以新的分区可以继承原始分区的属性。然而，融合分区时，属性必须继承表层次的默认值，因为此时有两个双亲，新的分区不能继承其一而损害其二的利益。

如下例子分割一个范围组合分区：

```
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);
```

17.4.13.4 分割索引分区

不能明确地分割局部索引中的分区。只有当分割基础表的分区时，才能分割局部索引分区。然而，可以像下面一样分割全局索引分区：

```
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;
```

正被分割的索引可以包含索引数据，并且不需要重建所产生的分区，除非原始分区事先已被标记成了 UNUSABLE。

17.4.14 舍弃分区

用 ALTER TABLE ... TRUNCATE PARTITION 语句，来移出表分区中的所有行。舍弃分区有些像删除分区，区别在于这是清空分区中的数据而不是物理地将其删除以外。

不能舍弃索引分区。但是，如果为该表定义了局部索引，则 ALTER TABLE TRUNCATE PARTITION 语句也舍弃在每个局部索引中匹配的分区。除非指定 UPDATE GLOBAL INDEXES（不能用于索引结构表），否则，全局索引，或分区全局索引的所有分区，都被标记成 UNUSABLE，并必须被重建。

17.4.14.1 舍弃一个表分区

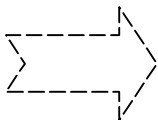
用 ALTER TABLE ... TRUNCATE PARTITION 语句，来移出表空间中的所有行，伴随或不伴随回收空间。

舍弃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，就用如下方法之一来舍弃表分区。

方法 1：

在执行 ALTER TABLE TRUNCATE PARTITION 语句期间，不管全局索引。本例中，表 sales 有一个全局索引 sales_area_ix，该索引被重建。

```
ALTER TABLE sales TRUNCATE PARTITION dec98;
```




```
ALTER INDEX sales_area_ix REBUILD;
```

这个方法最适用于要舍弃的分区包含了表中大部分数据的大表。

方法 2:

在发布 ALTER TABLE ... TRUNCATE PARTITION 语句之前,先发布 DELETE 语句,删除分区中的所有行。DELETE 语句更改全局索引,并启动触发器和产生重做与撤消日志。

例如,为舍弃分区边界为 1000 的第一个分区,发布如下语句:

```
DELETE FROM sales WHERE TRANSID < 10000;
```

```
ALTER TABLE sales TRUNCATE PARTITION dec98;
```

这个方法最适用于小表、或要舍弃的分区包含了表中大部分数据的大表。

方法 3:

在 ALTER TABLE 语句中指定 UPDATE GLOBAL INDEXES。这使得在舍弃分区的同时也舍弃了全局索引。

```
ALTER TABLE sales TRUNCATE PARTITION dec98
```

```
UPDATE GLOBAL INDEXES;
```

舍弃包含数据和参照完整性约束的分区 如果分区包含数据和具有参照完整性约束,就用如下方法之一来舍弃该表分区。

方法 1:

停用完整性约束,发布 ALTER TABLE ... TRUNCATE PARTITION 语句,然后,重新启用完整性约束:

```
ALTER TABLE sales
```

```
DISABLE CONSTRAINT dname_sales1;
```

```
ALTER TABLE sales TRUNCATE PARTITION dec94;
```

```
ALTER TABLE sales
```

```
ENABLE CONSTRAINT dname_sales1;
```

这种方法最适用于要舍弃的分区包含了表中很多数据的大表。

方法 2:

在发布 ALTER TABLE ... TRUNCATE PARTITION 语句之前,先发布 DELETE 语句删除分区中的所有行。DELETE 语句实施参照完整性约束,并启动触发器和产生重做与撤消日志。

注意:可以通过在删除分区中的所有行之前,给分区设置 NOLOGGING 属性(用 ALTER TABLE ... MODIFY PARTITION ... NOLOGGING),来充分地减少登记的数量。

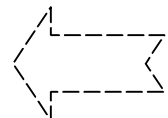
```
DELETE FROM sales WHERE TRANSID < 10000;
```

```
ALTER TABLE sales TRUNCATE PARTITION dec94;
```

这种方法最适用于小表,或要舍弃的分区包含了表中很少数据的大表。

17.4.14.2 舍弃一个子分区

用 ALTER TABLE ... TRUNCATE SUBPARTITION 语句,来移出组合分区表的子分区



中的所有行。还舍弃对应的局部索引子分区。

如下语句显示如何舍弃表的子分区中的数据。本例中，被删除的行所占据的空间就可以被表空间中其他模式的对象使用了。

```
ALTER TABLE diving
TRUNCATE SUBPARTITION us_locations
DROP STORAGE;
```

17.5 分区表和索引的例子

本节展示某些用于分区表和索引的例子。

17.5.1 移动历史表中的时间窗口

历史表描述一个企业在一段时间间隔之中的业务事务。历史表可以是基础表，它包含基础信息，如销售、支票和订单。历史表还可以是积累表，它是用诸如 GROUP BY、AVERAGE 或 COUNT 等操作，从基础信息中提取的汇总信息。

历史表中的时间间隔常常是一个滚动窗口。DBA 周期性地删除描述最早的事务的行的集合，然后为描述最近的事务的行的集合分配空间。例如，1995 年 4 月 31 日业务结束时，DBA 删除描述 1994 年 4 月以来的事务的行（及其支持的索引项），并为 1995 年 4 月的事务分配空间。

现在考虑一个特例。有一个表 order，它包含 13 个月的事务：1 年的历史数据和本月的订单。每个月有一个分区。这些按月的分区被命名成 order_yymm，亦即它们所在的表空间。

order 表包含两个局部索引，order_ix_onum 是一个订单号码上的局部的、前缀的、唯一索引，order_ix_supp 是一个供应商号码上的局部的、非前缀索引。局部索引分区用匹配基础表的后缀命名。还有一个用于客户姓名的全局唯一索引 order_ix_cust。order_ix_cust 包含 3 个分区，每一个分区用于 1/3 的字母。1994 年 10 月 31 日，按如下方法修改时间窗：

①备份最早的时间间隔的数据。

```
ALTER TABLESPACE order_9310 BEGIN BACKUP;
...
```

```
ALTER TABLESPACE order_9310 END BACKUP;
```

②删除最早的时间间隔的分区。

```
ALTER TABLE order DROP PARTITION order_9310;
```

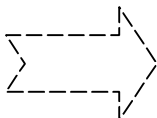
③增加最近的时间间隔的分区。

```
ALTER TABLE order ADD PARTITION order_9411;
```

④重建全局索引分区。

```
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;
```

通常，Oracle 获得足够的锁以保证没有什么操作（DML，DDL 或工具）会干扰单个的 DDL 语句，如 ALTER TABLE ... DROP PARTITION。然而，如果分区维护操作需要几步，



则 DBA 就要负责保证应用（或其他维护操作）不会干扰多步操作的进行。为此，某些针对性的方法是：

- 在一个被定义好的批处理窗口期间，关闭所有用户层次的应用。
- 通过撤消所有应用使用的角色的访问权限，来保证没有谁可以访问表 order。

17.5.2 将分区视图转换成分区表

本方案描述如何将一个分区视图（也称为“手动分区”）转换成一个分区表。分区视图按如下定义：

```
CREATE VIEW accounts AS
SELECT * FROM accounts_jan98
UNION ALL
SELECT * FROM accounts_feb98
UNION ALL
...
SELECT * FROM accounts_dec98;
```

为了增量式地将该分区视图迁移到一个分区表，采用如下步骤：

①首先，通过创建分区表，仅仅两个最近的分区（account_nov98 和 accounts_dec98）会被从该视图迁移到该分区表中。每个分区获得两数据块的一个段（作为一个占位器）

```
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-JAN-1999'));
```

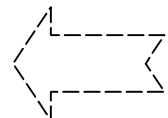
②用 EXCHANGE PARTITION 语句，将该表迁移到对应的分区。

```
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 分区相关的占位器数据段就同与 accounts_nov98 和 accounts_dec98 表相关的数据段作了交换。

③重新定义 accounts 视图

```
CREATE OR REPLACE VIEW accounts AS
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，它们拥有最初附属于 nov98 和 dec98 分区的占位器数据段。

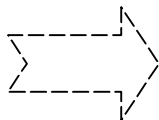
⑤在将 UNION ALL 视图中的所有表都转换到分区中之后，删除该视图，并将该分区表更名成被删除视图的名字。

```
DROP VIEW accounts;
RENAME accounts_new TO accounts;
```

17.6 查看有关分区表和索引的信息

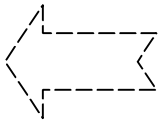
下面的视图显示有关分区表和索引的信息：

视图	说明
DBA_PART_TABLES	DBA 视图显示数据库中所有分区表的分区信息。
ALL_PART_TABLES	ALL 视图显示可被用户访问的所有分区表的分区信息。USER 视图局限于显示用户所拥有的分区表的分区信息。
USER_PART_TABLES	
DBA_TAB_PARTITIONS	显示分区层次的分区信息、分区存储参数、和由 ANALYZE 语句决定的用于分区的分区统计数据。
ALL_TAB_PARTITIONS	
USER_TAB_PARTITIONS	
DBA_TAB_SUBPARTITIONS	显示子分区层次的分区信息、子分区存储参数、和由 ANALYZE 操作决定的用于分区的子分区统计数据。
ALL_TAB_SUBPARTITIONS	
USER_TAB_SUBPARTITIONS	
DBA_PART_KEY_COLUMNS	显示分区表的分区键列。
ALL_PART_KEY_COLUMNS	
USER_PART_KEY_COLUMNS	
DBA_SUBPART_KEY_COLUMNS	显示组合分区表（和组合分区表上的局部索引）的子分区键列。
ALL_SUBPART_KEY_COLUMNS	
USER_SUBPART_KEY_COLUMNS	
DBA_PART_COL_STATISTICS	显示表的分区的，列统计数据和直方图信息。
ALL_PART_COL_STATISTICS	
USER_PART_COL_STATISTICS	
DBA_SUBPART_COL_STATISTICS	显示表的子分区的，列统计数据和直方图信息。
ALL_SUBPART_COL_STATISTICS	
USER_SUBPART_COL_STATISTICS	



(续表)

视图	说明
DBA_PART_HISTOGRAMS	显示表的分区上的直方图的直方图数据（每个直方图的端点）。
ALL_PART_HISTOGRAMS	
USER_PART_HISTOGRAMS	
DBA_SUBPART_HISTOGRAMS	显示表的子分区上的直方图的直方图数据（每个直方图的端点）。
ALL_SUBPART_HISTOGRAMS	
USER_SUBPART_HISTOGRAMS	
DBA_PART_INDEXES	显示分区索引的分区信息。
ALL_PART_INDEXES	
USER_PART_INDEXES	
DBA_IND_PARTITIONS	显示索引分区的如下信息：分区层次的分区信息、分区的存储参数、ANALYZE 语句采集的统计数据。
ALL_IND_PARTITIONS	
USER_IND_PARTITIONS	
DBA_IND_SUBPARTITIONS	显示索引子分区的如下信息：分区层次的分区信息、分区的存储参数、ANALYZE 语句采集的统计数据。
ALL_IND_SUBPARTITIONS	
USER_IND_SUBPARTITIONS	



第 18 章 管 理 簇

本章描述管理簇的几个方面，包括如下几个与索引簇、簇表和簇索引的管理相关的内容：

- 管理簇的准则
- 创建簇
- 更改簇
- 删除簇
- 查看有关簇的信息

18.1 管理簇的准则

簇提供了一种可选的、存储表的数据的方法。一个簇是由共享相同数据块的一组表组成的。因为这些表共享公共的列并且经常一起被使用，所以将这些表组合在一起。例如，emp 和 dept 表共享 deptno 列。当将 emp 和 dept 表组成簇（见图 18-1）时，Oracle 物理将 emp 和 dept 表中有关每个部门的所有行存储到相同的数据块中。

因为簇将不同表的相关的行一起存储到相同的数据块，所以，合理使用簇可以获得两个主要益处：

- 减少了磁盘 I/O，并改善了访问簇表的联结所花的时间。
- 簇键是列或列的组，它们是簇表所共有的。创建簇时指定簇键的列。以后在创建增加到簇中的每个表时，指定相同的列。每个簇键值在簇和簇索引中仅存储一次，而不管不同的表有多少行包含这个值。

因此，在簇中存储相关的表和索引数据与在非簇表中相比，需要较少的存储介质。例如，在图 18-1 中，注意，对于 emp 和 dept 表中包含相同值的许多行来说，每个簇键（每个 deptno）是如何只存储一次的。

创建簇之后，可以在簇中创建表。但是，在将任何行插入到簇表中之前，必须创建簇索引。使用簇不影响在簇表上创建另外的索引；可以照常创建和删除这些索引。

对于经常单独使用的表来说，不应该使用簇。

下面几节描述何时考虑管理簇的准则，包括如下内容：

- 为簇选择合适的表
- 为簇键选择合适的列
- 指定数据块空间使用
- 指定平均簇键及其相关的行所需的空間
- 指定每个簇和簇索引行的位置
- 估计簇大小并设置存储参数

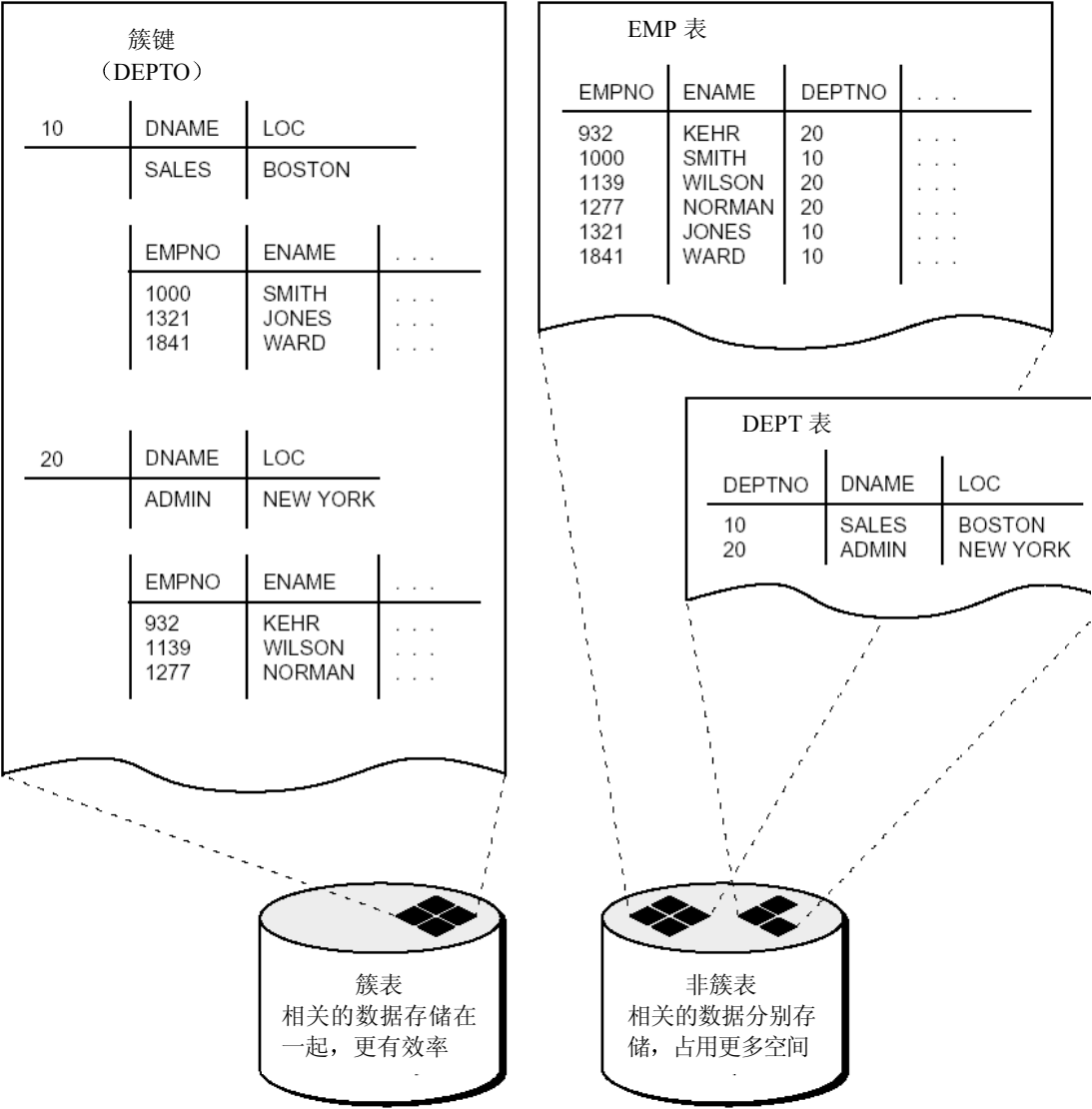


图 18-1 簇表数据

18.1.1 为簇选择合适的表

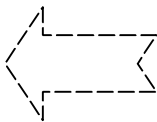
对于满足如下条件的表, 可使用簇:

- 表主要被查询 —— 即表不总是被插入或更改。
- 表中的记录经常被一起或联结查询。

18.1.2 为簇键选择合适的列

认真选择簇键列。如果在联结表的查询中使用多个列, 就用组合键做簇键。通常, 一个好的簇索引的特征与任何好的索引的特征相同。

好的簇键具有足够的唯一值, 以便相应于每个键值的行的组大概会填充同一个数据块。



每个簇键值的行太少了就会浪费空间，并导致一点性能损失。簇键太特殊了以致于仅有很少的行共享相同的值就会导致浪费数据块空间，除非在创建簇时指定了一个小的 `SIZE`（参见第 18 章的“指定平均簇键及其相关行所需的空间”）。

每个簇键值的行太多可能会导致额外的搜索以发现该键的行。簇键值太普遍了（如，`male`（男）和 `female`（女））也可能导致过份的搜索，并可能导致比没成簇时更差的性能。

簇索引不能是唯一的或包含一个 `long` 类型的列。

18.1.3 指定数据块空间使用

通过在创建簇时指定 `PCTFREE` 和 `PCTUSED` 参数，可以影响空间的利用，和为了更改在簇数据段的数据块中的当前行而保留的空间的数量。如果在簇中创建表时没有指定 `PCTFREE` 和 `PCTUSED` 参数，则簇表自动地使用为簇指定的设置值。

18.1.4 指定平均簇键及其相关的行所需的空间

`CREATE CLUSTER` 语句有一个可选的参数，`SIZE`，它是一个估计的平均簇键及其相关的行所需的字节数。Oracle 在执行如下任务时使用 `SIZE` 参数：

- 在估计可以填充到簇数据块的簇键（及其相关的行）的数量时
- 在限制放置到簇数据块中的簇键的数量时。这在簇中最大化键的存储效率。

`SIZE` 不限制给定簇键可以使用的空间。例如，如果 `SIZE` 被设置成两个簇键都可以填充进一个数据块，则每个簇键仍可以使用任何数量的可用数据块空间。

默认时，Oracle 在簇的数据段的每个数据块中仅仅存储一个簇键及其相关的行。尽管数据块的大小在各个操作系统中各不相同，当簇表被导入到其他机器上的其他数据库中时，仍维持一个键一个数据块的规则。

如果给定簇键值的所有行不能填充到一个数据块，则数据块被链接在一起以加快对具有给定键的所有值的访问。簇索引指向数据块的链的开始处，每个数据就包含簇键值及其相关的行。如果簇的 `SIZE` 使得有多个键填充到一个数据块，则数据块可以属于多个链。

18.1.5 指定每个簇和簇索引行的位置

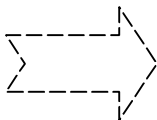
如果具有合适的权限和表空间限额，就可以在当前联机的任何表空间中创建一个新簇和相关的簇索引。为了标识存储新簇或索引的表空间，就要在 `CREATE CLUSTER/INDEX` 语句中指定 `TABLESPACE` 选项。

可以在不同的表空间中创建簇及其簇索引。实际上，将簇及其索引创建在存储在不同存储设备上的不同表空间中，就可能用最小的磁盘竞争而同步地检索表数据和索引数据。

18.1.6 估计簇大小和设置存储参数

在创建簇之前估计一下簇的大小有如下的益处：

- 可以用对簇，连同对索引、回滚段和重做日志文件的估计，的综合估计的大小，来决定支持一个希望的数据库所需的磁盘空间的数量。从这些估计中，就可以购买正确的硬件并做出其他正确的决定。
- 可以用单个簇的估计大小，来更好地管理该簇要使用的磁盘空间。当创建簇时，



就可以设置合适的存储参数，并改善使用该簇的应用的 I/O 性能。

无论是否在创建表之前估计了表的大小，当创建每个非簇表时都可以明确地设置其存储参数。在创建表或以后更改表时没有明确地设置的任何存储参数，都自动地使用该表所在的表空间的对应的默认存储参数设置值。簇表也自动使用其簇的存储参数设置值。

18.2 创 建 簇

要想在你的模式中创建簇，必须具有 CREATE CLUSTER 系统权限，以及打算包含该簇的表空间的限额，或具有 UNLIMITED TABLESPACE 系统权限。

要想在另外用户的模式中创建簇，你必须具有 CREATE ANY CLUSTER 系统权限，并且该拥有者必须具有打算包含该簇的表空间的限额，或 UNLIMITED TABLESPACE 系统权限。

用 CREATE CLUSTER 语句创建簇。如下语句创建一个存储 emp 和 dept 表的、按 deptno 列成簇的、名为 emp_dept 的簇。

```
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);
```

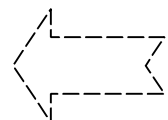
如果没有指定 INDEX 关键词，像本例，就默认地创建一个索引簇。当指定了散列参数 (HASHKEYS, HASH IS 或 SINGLE TABLE HASHKEYS) 时，还可以创建 HASH (散列) 簇。散列簇在第 19 章“管理散列簇”中描述。

18.2.1 创建簇表

要想在簇中创建表，你必须具有或者 CREATE TABLE 或者 CREATE ANY TABLE 系统权限。为了在簇中创建表，你不需要表空间限额或 UNLIMITED TABLESPACE 系统权限。

用带有 CLUSTER 选项的 CREATE TABLE 语句在簇中创建表。使用如下语句可以将 emp 和 dept 表创建到 emp_dept 簇中。

```
CREATE TABLE emp (
empno NUMBER(5) PRIMARY KEY,
ename VARCHAR2(15) NOT NULL,
. . .
deptno NUMBER(3) REFERENCES dept)
CLUSTER emp_dept (deptno);
```



```
CREATE TABLE dept (
deptno NUMBER(3) PRIMARY KEY, . . . )
CLUSTER emp_dept (deptno);
```

注意：可以在 CREATE TABLE 语句中为簇表指定模式。簇表可以在与包含该簇的模式所不同的模式中。列的名称也不需要匹配，但它们的结构必须匹配。

18.2.2 创建簇索引

要创建簇索引，必须满足如下条件之一：

- 你的模式包含簇。
- 你具有 CREATE ANY INDEX 系统权限。

在任一情况之下，你还必须或者具有打算用来包含该簇索引的表空间的限额，或者具有 UNLIMITED TABLESPACE 系统权限。

在任何行能被插入到任何簇表之前，必须先创建簇索引。如下语句为 emp_dept 簇创建一个簇索引：

```
CREATE INDEX emp_dept_index
ON CLUSTER emp_dept
INITRANS 2
MAXTRANS 5
TABLESPACE users
STORAGE (INITIAL 50K
NEXT 50K
MINEXTENTS 2
MAXEXTENTS 10
PCTINCREASE 33)
PCTFREE 5;
```

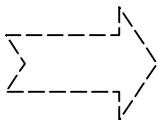
簇索引子句 (ON CLUSTER) 标识正为其创建索引的簇 emp_dept。该语句还明确地为簇和簇索引指定了几个存储参数设置值。

18.3 更 改 簇

要想更改簇，你的模式必须包含该簇，或你必须具有 ALTER ANY CLUSTER 系统权限。可以更改现存簇的如下设置值：

- 物理属性 (PCTFREE, PCTUSED, INITRANS, MAXTRANS 和存储特征)
- 为了存储簇键值的所有行所需空间的平均值 (SIZE)
- 默认的并行度

另外，可以明确地给簇分配新的盘区，或回收在簇的末端的任何未使用的盘区。在需要时，Oracle 动态地为簇的数据段分配附加的盘区。然而在某种情况下，可能想明确地为簇分配一个附加的盘区。例如，当使用 Oracle9i 真实应用簇时，就可以明确地为一个特殊的实例分配一个簇的盘区。用带有 ALLOCATE EXTENT 子句的 ALTER CLUSTER 语句来



为簇分配新的盘区。

当更改簇的数据块空间使用参数（PCTFREE 和 PCTUSED）或簇的大小参数（SIZE）时，新的设置值适用于该簇所使用的所有数据块，包括已经分配给该簇的和以后分配给该簇的数据块。需要的时候（并不是立即地），重新组织已经给表分配的数据块。

当更改簇的事务项设置值（INITRANS 和 MAXTRANS）时，INITRANS 的新设置值仅适用于以后分配给该簇的数据块，而 MAXTRANS 的新设置值适用于该簇的所有数据块（已经分配的和以后分配的数据块）。

不能更改存储参数 INITIAL 和 MINEXTENTS。其他存储参数的所有新设置值仅仅影响以后分配给簇的盘区。

用 ALTER CLUSTER 语句来更改簇。如下语句更改 emp_dept 簇：

```
ALTER CLUSTER emp_dept
PCTFREE 30
PCTUSED 60;
```

18.3.1 更改簇表

可以用 ALTER TABLE 语句来更改簇表。但是，如在 ALTER TABLE 语句中给簇表设置任何数据块空间参数、事务项参数或存储参数，都将产生一条错误消息（ORA_01771, illegal option for a clustered table（对于簇表而言，是违法选项））。Oracle 将簇的参数用于所有簇表。因此，对于簇表而言，可以用 ALTER TABLE 语句所做的事，仅仅是增加或修改列、删除非簇键列，或是增加、删除、启用或停用完整性约束或触发器，有关更改表的信息，参见第 15 章的“更改表”。

18.3.2 更改簇索引

更改簇索引与更改其他索引一样，参见第 16 章的“更改索引”。

注意：在估计簇索引的大小时，要记住索引是建立在每个簇键上的而不是建立在实际的行上的。因此，每个键仅在索引中出现一次。

18.4 删 除 簇

如果不再需要簇中的表，则可以将簇删除。删除簇时，也将簇中的表及其对应的簇索引删除。属于簇的数据段的盘区和属于簇索引的索引段的盘区都将返还给它们所处的表空间，并可被表空间中的其他段使用。

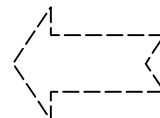
为了删除不包含表的簇及其簇索引，使用 DROP CLUSTER 语句。例如，如下语句删除名为 emp_dept 的空簇：

```
DROP CLUSTER emp_dept;
```

如果簇中包含一个或多个簇表，并打算也将表删除时，就在 DROP CLUSTER 语中增加 INCLUDING TABLES 选项，如：

```
DROP CLUSTER emp_dept INCLUDING TABLES;
```

如果簇中包含表，又没有增加 INCLUDING TABLES 选项时，就会产生错误。



如果簇中一个或多个表包含有被簇之外的表的 FOREIGN KEY 约束所参照的主键或唯一键，就不能删除该簇，除非依赖的 FOREIGN KEY 约束也被删除。可以使用 DROP CLUSTER 语句的 CASCADE CONSTRAINTS 选项来容易地做此事，就像下面的例子：

```
DROP CLUSTER emp_dept INCLUDING TABLES CASCADE CONSTRAINTS;
```

如果存在约束，你又没有使用 CASCADE CONSTRAINTS 选项，Oracle 就返回一个错误。

18.4.1 删除簇表

要想删除一个簇，你的模式必须包含该簇，或你必须具有 DROP ANY CLUSTER 系统权限。删除一个包含表的簇不需要另外的权限，甚至于簇的拥有者不拥有簇表时也不需要另外的权限。

簇表可以被单独地删除，而不影响表的簇、其他簇表或簇索引。删除簇表就像删除非簇表一样——用 DROP TABLE 语句。参见第 15 章的“删除表”。

注意：当从簇中删除单个表时，Oracle 单独地删除该表的每一行。为了最有效地删除整个簇，即删除包含所有表的簇，使用带有 INCLUDING TABLES 选项的 DROP CLUSTER 语句。仅当你希望保留簇的剩余部分时，才从簇中删除单独的表（使用 DROP TABLE 语句）

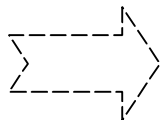
18.4.2 删除簇索引

簇索引可以被删除，而不影响簇或它的簇表。但是，如果没有簇索引就不能使用簇表；为了允许访问簇，就必须重建簇索引。有些时候，作为重建碎片簇索引过程的一部分，也删除簇索引。

18.5 查看有关簇的信息

下面的视图显示有关簇的信息：

视图	说明
DBA_CLUSTERS	DBA 视图描述数据库中的所有簇。ALL 视图描述用户可以访问的所有簇。
ALL_CLUSTERS	USER 视图局限于用户拥有的簇。这些视图中的某些列包含有由
USER_CLUSTERS	DBMS_STATS 包或 ANALYZE 语句产生的统计数据。
DBA_CLU_COLUMNS	这些视图将表列映射到簇列。
USER_CLU_COLUMNS	



第 19 章 管理散列簇

本章描述如何管理散列簇，包括如下内容：

- 何时使用散列簇
- 创建散列簇
- 更改散列簇
- 删除散列簇
- 查看有关散列簇的信息

19.1 何时使用散列簇

为了改善数据检索的性能，一种可选的方法是将表存储到一个散列簇中。散列簇是一种为非簇表提供索引或索引簇的可选方法。对于索引表或索引簇，Oracle 使用存储在单独的索引中的键值来定位表中的行。为了使用散列方法，就要创建散列簇并将表装入其中。Oracle 物理地将表中的行存储到散列簇，并根据散列函数的结果来检索它们。

Oracle 使用散列函数来产生被称为散列值的、基于特殊的簇键值的数字值的分布。散列簇的键类似于索引簇的键，可以是单个列或组合键（多列键）。为了在散列簇中查找或存储行，Oracle 将散列函数应用到行的簇键值。产生的散列值对应于簇中的数值块，然后 Oracle 根据发布的语句在这些数据块上读或写。

为了在簇表或簇中查找或存储行，最少要执行两个 I/O（经常更多）：

- 一个或多个 I/O，在索引中查找或存储键值。
- 另一个 I/O，在表或簇中读或写行

相反，Oracle 使用散列函数在散列簇中定位行时不需要 I/O。其结果是，在散列簇中读或写行需要至少 1 个 I/O 操作。

本节通过对照散列方法最有用的情况与没什么好处的情况，来帮助你决定何时使用散列簇。如果发现你的决定是，使用索引方法比使用散列方法更好，那么，你就应该考虑或是单独地存储表或是将其作为簇的一部分。

注意：即便决定使用散列方法，表仍然可以在任何列（包括簇键）上具有单独的索引。

19.1.1 散列方法有用的情况

在具有如下条件时，散列方法有用：

- 大多数查询是簇键上的相等查询：

```
SELECT ... WHERE cluster_key = ...;
```

此时，相等条件中的簇键被散列，并且，通常读一次就能找到对应的散列键。相对照，对于索引表，必须首先在索引中找到其键值（通常要读几次），然后再从表中读行（另一次读）。

- 散列簇中的表的大小基本是不变的，因此，可以决定簇中表的行数及其所需的空间数量。如果散列簇中的表需要比初始分配给该簇的更多的空间，则性能肯定会降

低，因为需要溢出数据块了。

19.1.2 散列方法无益的情况

在如下情况时，散列方法是无益的：

- 表上的大多数查询是在簇键值的一个范围中检索行。例如，在全表扫描或类似下面的查询中，散列函数不能被用于决定特定的散列键的位置。代之以，为了找到查询的行，必须做全表扫描那种事。

```
SELECT . . . WHERE cluster_key < . . . ;
```

用索引的话，键值在索引中被排序，因此，就能用相对少的 I/O 找到满足查询中的 WHERE 子句的簇键值。

- 表不是静止的，而是经常增长。如果表的增长没有尽头，在表（及其簇）的生命期中所需的空间就不能预先决定了。
- 应用经常对表进行全表的扫描，并且很少填充表。在此情况下，用散列方法的全表扫描需要更长时间。
- 不能提供散列簇最终需要的预分配空间。

19.2 创建散列簇

用 CREATE CLUSTER 语句来创建散列簇，但要指定 HASHKEYS 子句。如下语句创建存储 trial 表的、按 trialno 列成簇的、名为 trial_cluster 的簇；另一个语句在该簇中创建表。

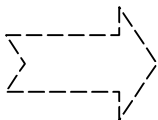
```
CREATE CLUSTER trial_cluster (trialno NUMBER(5,0))
PCTUSED 80
PCTFREE 5
TABLESPACE users
STORAGE (INITIAL 250K NEXT 50K
MINEXTENTS 1 MAXEXTENTS 3
PCTINCREASE 0)
HASH IS trialno HASHKEYS 150;
CREATE TABLE trial (
trialno NUMBER(5,0) PRIMARY KEY,
...)
CLUSTER trial_cluster (trialno);
```

与索引簇相似，散列簇的键可以是单个列或组合键（多列键）。本例中是单个列。

HASHKEYS 值，本例中为 150，指定和限制该簇所使用的散列函数可以产生的唯一的散列值的数量。Oracle 将该指定的数量舍入到最接近的素数。

如果没有指定 HASH IS 子句，Oracle 就使用内部散列函数。如果簇键已经是一个在其范围内均一分布的唯一的标识，就可以不用内部散列函数并将该簇键指定为散列值，就像上例一样。你还可以用 HASH IS 子句来指定用户定义的散列函数。

在散列簇上不能创建簇索引，并且不需要在散列簇键上创建索引。



有关在簇中创建表、设置索引和散列簇共用的 CREATE CLUSTER 语句的参数的准则、以及创建任何簇所需的权限，参见第 18 章“管理簇”。下面几节为特别针对散列簇的设置 CREATE CLUSTER 语句的参数进行解释和提供准则：

- 创建单个表散列簇
- 控制散列簇的空间使用
- 估计散列簇所需的空间大小

19.2.1 创建单个表散列簇

你还可以创建单个表散列簇，它提供了对表中的行的快速访问。当然，这个表必须是散列簇中仅有的表。本质上说，必须在散列键和数据行之间存在一对一的映射。如下语句创建一个簇键为 variety 的、名为 peanut 单个表散列簇：

```
CREATE CLUSTER peanut (variety NUMBER)
SIZE 512 SINGLE TABLE HASHKEYS 500;
```

Oracle 将 HASHKEY 值舍入成最接近的素数，所以这个簇具有最大 503 个散列键值，每个有 512 字节。

注意：SINGLE TABLE 选项仅对散列簇有效。还必须指定 HASHKEYS。

19.2.2 控制散列簇的空间使用

创建散列簇时，重要的事是，正确地选择簇键和设置 HASH IS, SIZE 和 HASHKEYS 参数，以便性能和空间使用是最优的。下面的准则描述如何设置这些参数。

19.2.2.1 选择键

选择正确的簇键取决于针对簇表所发布的最常用的查询的类型。例如，考虑散列簇中的 emp 表。如果查询常常是按职员号码来选择行，则 empno 列就应该是簇键。如果查询常常是按部门号码来选择行，则 deptno 列就应该是簇键。对于包含单个表的散列簇，簇键典型地是其所包含的表的整个主键。

散列簇的键，就像索引簇的键，可以是单个列或组合键（多列键）。具有组合键的散列簇必须使用 Oracle 的内部散列函数。

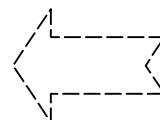
19.2.2.2 设置 HASH IS

仅仅当簇键是 NUMBER 数据类型的、包含均匀分布的整数的、单个的列时，才指定 HASH IS 参数。如果上述条件成立，你就可以在簇中分配行，以便每个唯一簇键值会无冲突（两个簇键值具有相同的散列值）地分配到唯一的散列值。如果上述条件不成立，就忽略这个选项，以便使用内部散列函数。

19.2.2.3 设置 SIZE

应将 SIZE 设置成，为保持任何给定散列键的所有行所需的空间的平均数。因此，为了合理地决定 SIZE，必须注意数据的特征：

- 如果散列簇仅包含单个表，并且该表中的行的散列键值是唯一的（一个值一行），



就可以将 **SIZE** 设置成簇中的平均的行大小。

- 如果散列簇包含多个表，就可以将 **SIZE** 值设置成，为保持与有代表性的散列值相关的所有行所需的空间的平均数。

进一步，当决定了一个 **SIZE** 的（初步）值时，考虑下面的情况。如果 **SIZE** 值小（可以给每个数据块赋予 4 个以上的散列键），就可在 **CREATE CLUSTER** 语句中使用这个 **SIZE** 值。然而，如果 **SIZE** 值大（只能给每个数据赋予 4 个以下散列键），那么，就还应该考虑所期望的冲突频率、以及是否数据检索的性能、或空间使用的效率更重要。

- 如果散列簇不使用内部散列函数（如果指定了 **HASH IS**），并期望很少或没有冲突，就可以使用 **SIZE** 的初始值。无冲突，并且尽可能有效地使用空间。
- 如果预料到插入时会经常冲突，则为了存储行而分配溢出数据块的可能性就很高。为了减少溢出数据块的可能性，并在频繁冲突时使性能最好，就应按下图所示调整 **SIZE**。

每个数据块的可用空间/计算值 SIZE	SIZE 的设置值
1	SIZE
2	SIZE+15%
3	SIZE+12%
4	SIZE+8%
>4	SIZE

估计过高的 **SIZE** 值增加簇中未使用空间的数量。如果空间的效率比数据检索的性能更重要，就忽略上面的调整并使用 **SIZE** 的初始值。

19.2.2.4 设置 HASHKEYS

为了散列簇中的行的最大分布，Oracle 将 **HASHKEYS** 值舍入成最接近的素数。

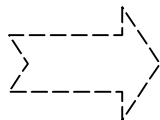
19.2.2.5 控制散列簇的空间：例子

下面的例子显示如何正确地选择簇键，并设置 **HASH IS**、**SIZE** 和 **HASHKEYS** 参数。在所有例子中，假设数据块大小是 2K，并且平均来说，每个数据块的可用数据空间（数据块大小减系统开销）是 1950 字节。

例 1 你打算将 **emp** 表装载进一个散列簇。大多数查询按职员号码检索职员记录。你估计，在任何给定时间，**emp** 表中的最大行数是 10000，并且平均行大小是 55 个字节。

这时，应将 **empno** 作为簇键。因为该列包含唯一的整数，所以，可以不使用内部散列函数。**SIZE** 可被设置成平均行大小，55 字节。注意，给每个数据块赋予了 34 个散列键。**HASHKEYS** 可被设置成表中的行数，10000。Oracle 将该值尽似成最接近的素数：10007。

```
CREATE CLUSTER emp_cluster (empno
NUMBER)
. . .
SIZE 55
HASH IS empno HASHKEYS 10000;
```



例 2 条件与上例相似。然而此时，通常按部门号码检索行。常常是，有 1000 个部门，每个部门平均有 10 个职员。部门号码按 10 递增（0, 10, 20, 30, …）。

这时，应将 deptno 作为簇键。因为该列包含均匀分布的整数，所以可以不使用内部散列函数。SIZE 的初始值（为保持每个部门的所有行所需的空间的平均数）是 55 字节*10，或 550 字节。使用这个 SIZE 值，仅仅可以给每个数据块赋予 3 个散列键。如果预料到某些冲突，并且希望数据检索的性能最好。稍微地更改所估计的 SIZE 值，以预防需要溢出数据块而带来的冲突。调整 12% 的 SIZE，到 620 字节（参见第 19 章的“设置 SIZE”），就为可能冲突的行提供了更多的空间。

HASHKEYS 可被设置成唯一的部门号码的个数，1000。Oracle 将该值舍入成最接近的素数：1009。

```
CREATE CLUSTER emp_cluster (deptno NUMBER)
. . .
SIZE 620
HASH IS deptno HASHKEYS 1000;
```

19.2.3 估计散列簇所需的空间大小

就像索引簇，估计散列簇中数据所需的存储量很重要。

依据 SIZE 和 HASHKEYS 的设置值，Oracle 保证最初分配的空间足够存储散列表。如果对存储参数 INITIAL、NEXT 和 MINEXTENTS 的设置值不能满足簇表的大小，则分配增加的（附加的）盘区直到至少满足 SIZE*HASHKEYS。例如，假设数据块大小是 2K，每个数据块可用的数据空间大约是 1900 字节（数据块大小减系统开销），并且，在 CREATE CLUSTER 语句中按如下方式指定了 STORAGE 和 HASH 参数：

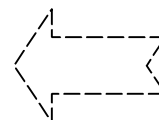
```
STORAGE (INITIAL 100K
NEXT 150K
MINEXTENTS 1
PCTINCREASE 0)
SIZE 1500
HASHKEYS 100
```

本例中，仅仅可以给每个数据块赋予一个散列键。因此，该散列簇所需的初始空间至少是 100*2K 或 200K。对存储参数的设置值不能满足这个需求。因此，分配给该散列簇一个 100K 的初始盘区和一个 150K 的第 2 个盘区。

可选地，假设 HASH 参数被指定为：

```
SIZE 500 HASHKEYS 100
```

这时，就可以给每个数据块赋予三个散列键了。因此，该散列簇所需的初始空间至少是 34*2K 或 68K。存储参数的最初设置值满足了这个需求（分配给该散列簇 100K 的初始盘区）。



19.3 更改散列簇

可以用 ALTER CLUSTER 语句来更改散列簇：

```
ALTER CLUSTER emp_dept . . . ;
```

更改散列簇的含意与在第 18 章的“更改簇”中描述的更改索引簇一样。但是，在 ALTER CLUSTER 语句中不能指定 SIZE、HASHKEYS 和 HASH IS 参数。为了更改这些参数，必须重建簇，然后从原始簇中复制数据。

19.4 删除散列簇

可以用 DROP CLUSTER 语句来删除散列簇：

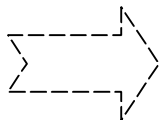
```
DROP CLUSTER emp_dept;
```

用 DROP TABLE 语句来删除散列簇中的表。删除散列簇和删除散列簇中的表的含意与删除索引簇一样。

19.5 查看有关散列簇的信息

下面的视图显示有关散列簇的信息：

视图	说明
DBA_CLUSTERS	DBA 视图描述数据库中的所有簇(包括散列簇)。ALL 视图描述用户可以访问的所有簇。USER 视图局限于用户拥有的簇。这些视图中的某些列包含有由 DBMS_STATS 包或 ANALYZE 语句产生的统计数据。这些视图将表列映射到簇列。
ALL_CLUSTERS	
USER_CLUSTERS	
DBA_CLU_COLUMNS	这些视图列出用于散列簇的散列函数。
USER_CLU_COLUMNS	
DBA_CLUSTER_HASH_EXPRESSIONS	
ALL_CLUSTER_HASH_EXPRESSIONS	
USER_CLUSTER_HASH_EXPRESSIONS	



第 20 章 管理视图、序列和同义词

本章介绍视图、序列和同义词的管理，包括如下内容：

- 管理视图
- 管理序列
- 管理同义词
- 查看有关视图、同义词和序列的信息

20.1 管 理 视 图

视图是一个或多个表（或其他视图）中所包含数据的裁剪后的表示，存放查询的输出并把视图看作一个表。也可以把一个视图看作“存储的查询”或“虚拟的表”。可以在许多使用表的地方使用视图。

本节介绍视图管理的有关方面，包含如下内容：

- 创建视图
- 更新联结视图
- 更改视图
- 删除视图
- 替换视图

20.1.1 创建视图

为了创建视图，必须满足如下需求：

- 为了在你的模式中创建视图，必须拥有 `CREATE VIEW` 权限。为了在其他用户的模式中创建视图，必须拥有 `CREATE ANY VIEW` 系统权限。能直接地或者通过一个角色获得这些权限。
- 视图的拥有者（无论是你还是其他用户）必须被明确授予访问在视图定义中所参考的所有对象的权限。视图拥有者不能通过角色获得这些权限。同样，视图的功能取决于视图拥有者的权限。例如：如果视图的拥有者仅仅有 `Scott` 的 `emp` 表的 `INSERT` 权限，那么此视图仅仅用于在 `emp` 表中插入新行，而不能用于 `SELECT`、`UPDATE` 或者 `DELETE` 行。
- 如果视图的拥有者打算授权其他用户访问视图，则视图的拥有者必须已经获得了对于基础对象的带有 `GRANT OPTION` 的对象权限，或者带有 `ADMIN OPTION` 的系统权限。

可用 `CREATE VIEW` 语句创建视图。每个视图是由参考表的查询、物化的视图或其他视图定义的。就像所有的子查询，定义视图的查询不能包含 `FOR UPDATE` 子句。

下面的语句基于 `emp` 表中数据的子集创建一个视图：

```
CREATE VIEW sales_staff AS
SELECT empno, ename, deptno
```

```
FROM emp
WHERE deptno = 10
WITH CHECK OPTION CONSTRAINT sales_staff_cnst;
```

定义 `sales_staff` 视图的查询仅参考部门号为 10 的行。此外，`CHECK OPTION` 创建带有约束（名字为 `sales_staff_cnst`）的视图，针对该视图发布的 `INSERT` 和 `UPDATE` 语句不能产生该视图不能选择的行。例如：下面的 `INSERT` 语句，依靠 `sales_staff` 视图，成功地往 `emp` 表中插入一行，该视图包含了部门号为 10 的所有行：

```
INSERT INTO sales_staff VALUES (7584, 'OSTER', 10);
```

但是，下面的 `INSERT` 语句被回滚并且返回一个错误，因为此语句想在部门号为 30 处插入一行，该部门不能用 `sales_staff` 视图选择出来。

```
INSERT INTO sales_staff VALUES (7591, 'WILLIAMS', 30);
```

可选的，可用指定 `WITH READ ONLY` 子句的方式构造视图，从而防止通过视图对基础表的任何更改、插入或删除。如果没有指定 `WITH` 子句，带有某些限制的视图，自然是可更新的。

20.1.1.1 联结视图

在 `FROM` 子句中，也可以指定多个基础表或视图来创建视图。这种视图就叫联结视图（join views）。下面的语句创建联结 `emp` 表和 `dept` 表中数据的 `division1_staff` 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;
```

可更新的联结视图（`updatable join view`）是允许进行 `UPDATE`、`INSERT` 和 `DELETE` 操作的联结视图。关于此问题的更深入地讨论，参见第 21 章的“更新联结视图”。

20.1.1.2 在创建视图时定义查询的扩展

创建视图时，Oracle 将顶层视图查询中的所有通配符（*）展开成列的列表。形成的查询被存储在数据字典中；任何子查询都被完整无缺地保留下来。在一个展开列的列表中的列名被放在括号中，以便处理基础对象的列最初输入时就带有括号而在查询中又需要它们在语法上正确的可能性。

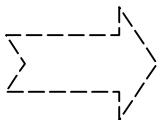
作为例子，假设按下面的语句创建 `dept` 视图：

```
CREATE VIEW dept AS SELECT * FROM scott.dept;
```

Oracle 按如下格式保存 `dept` 视图的定义查询：

```
SELECT "DEPTNO", "DNAME", "LOC" FROM scott.dept;
```

创建存在着错误的视图，不能展开通配符。但是，如果视图在编译过程中没有错误，则可以展开在定义查询中的通配符。



20.1.1.3 带错误创建视图

如果在 CREATE VIEW 语句中没有语法错误，即使不能执行视图的定义查询，Oracle 也能创建视图。在这种情况下，该视图就被认为“带错误创建”。例如，当创建视图时，涉及到不存在的表或现存表中的无效的列，或者当视图的拥有者不具备所需的权限时，还是能创建视图并输入数据字典。但是，这种视图还是不能用的。

要能创建带有错误的视图，必须在 CREATE VIEW 语句中包括 FORCE 选项。

```
CREATE FORCE VIEW AS ...;
```

默认时，带有错误的视图不能被创建为 VALID。当试图创建这种视图时，Oracle 返回一个消息，表示带错误创建视图。带错误所创建的视图的状态为 INVALID。如果后来条件改变了，使得无效视图能被执行了，就可以重新编译视图，使之有效（可用的）。有关如何改变条件以及它们对视图的影响，参见第 21 章的“管理对象依赖性”。

20.1.2 更新联结视图

可更新的（updatable）联结视图（也称为可修改的（modifiable）联结视图）是一种视图，该视图在 SELECT 语句的顶层 FROM 子句中包含多个表，并且不受 WITH READ ONLY 子句限制。

注意：某些限制和条件可能影响联结视图是否可更新。

另外，如果视图是在其他嵌套视图之上联结的，那么，其他嵌套视图必须合并到顶层视图。

有些数据字典视图指出联结视图中的列是否是可更新的。参见第 20 章的表 20-1 “UPDATABLE_COLUMNS 视图”中关于这类视图的描述。

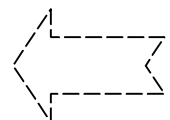
用于可更新的联结视图的规则如下：

规则	说明
一般规则	在联结视图上的任何 INSERT、UPDATE 或 DELETE 操作一次只能修改一个下层基础表。
UPDATE 规则	联结视图中的所有可更新的列必须映射到键预留表的列中。参见第 20 章的“键预留表”一节中关于键预留表问题的讨论。如果用 WITH CHECK OPTION 子句定义视图，那么所有的联结列和重复表中的所有列是不可更新的。
DELETE 规则	只要在联结中有一个准确的键预留表，就能从联结视图中删除行。如果用 WITH CHECK OPTION 子句定义视图，并且重复键预留表，那么就不能通过视图删除行。
INSERT 规则	INSERT 语句不必直接或明确地参照非键预留表的列。如果联结视图是用 WITH CHECK OPTION 子句定义的，那么不允许使用 INSERT 语句。

下面的例子说明了上述规则，而在后续章节中讨论有关键预留表的问题。

如果在表中明确地定义了主键和外键，或者定义了唯一的索引，下面的例子才能工作。下面是针对 emp 和 dept 表的合适的但受约束的表定义。

```
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));

```

也可以省略上述所列出的主键和外键约束，而在 dept(deptno) 上创建 UNIQUE INDEX (唯一索引) 以使下面的例子工作。

下面的语句创建例子中所参照的 emp_dept 联结视图：

```

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');

```

20.1.2.1 键预留表

键预留表的概念是理解修改联结视图的限制的基础。如果表中的每个键也是联结的结果键，则该表是键预留的。所以，一个键预留表通过联结预留它自己的键。

注意：不需要将表中一个或多个键选择作为预留键。选择了键就可以了，它们也就作为该联结的结果的键了。

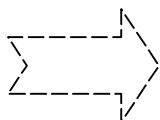
表中的键的预留属性不依赖于表中的实际数据。它更是其模式的属性。例如，如果在 emp 表中每个部门最多有一名雇员，那么 deptno 在 emp 和 dept 的联结结果中是唯一的，但 dept 还不是一个键预留表。

如果从 emp_dept 中 SELECT 所有行，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

```



8 rows selected.

在此视图中，**emp** 是键预留表，因为 **empno** 是 **emp** 表的一个键，也是一个联结的结果键。**dept** 不是键预留表，因为尽管 **deptno** 是 **dept** 表的一个键，但它不是一个联结的键。

20.1.2.2 DML 语句和联结视图

一般的规则是，联结视图上的任何 **UPDATE**，**DELETE** 或者 **INSERT** 语句仅能修改一个下层基础表。下面的例子说明了针对 **UPDATE**，**DELETE** 和 **INSERT** 语句的规则。

UPDATE 语句 下面的例子显示了正确修改 **emp_dept** 视图的 **UPDATE** 语句：

```
UPDATE emp_dept
SET sal = sal * 1.10
WHERE deptno = 10;
```

在 **emp_dept** 视图中，不允许下面的 **UPDATE** 语句：

```
UPDATE emp_dept
SET loc = 'BOSTON'
WHERE ename = 'SMITH';
```

该语句失败了，产生一个错误（ORA-01779 cannot modify a column which maps to a non key-preserved table（不能修改映射到非键预留表中的列）），因为该语句试图修改基础 **dept** 表，但 **dept** 表不是 **emp_dept** 视图中的键预留表。

通常，联结视图中的所有可更新的列必须映射到键预留表中列。如果视图是使用 **WITH CHECK OPTION** 子句定义的，那么所有联结的列和所有从在视图中参照多次的表中取出的列，都是不可修改的。

所以，例如，如果 **emp_dept** 视图是使用 **WITH CHECK OPTION** 子句定义的，下面的 **UPDATE** 语句会失败：

```
UPDATE emp_dept
SET deptno = 10
WHERE ename = 'SMITH';
```

该语句失败是因为它试图更新一个联结的列。

DELETE 语句 能够通过所提供的在联结中有一个而且仅有一个键预留表的联结视图进行删除工作。

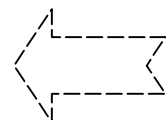
下面的 **DELETE** 语句在 **emp_dept** 视图上工作：

```
DELETE FROM emp_dept
WHERE ename = 'SMITH';
```

这个在 **emp_dept** 视图上的 **DELETE** 语句是合法的，因为该语句能被转换成在基础 **emp** 表上的 **DELETE** 操作，并且 **emp** 表是该联结中的唯一的键预留表。

如果假使创建了下面的视图，则在该视图上就不能执行 **DELETE** 操作，因为 **e1** 和 **e2** 二者都是键预留表：

```
CREATE VIEW emp_emp AS
SELECT e1.ename, e2.empno, deptno
```



```
FROM emp e1, emp e2
WHERE e1.empno = e2.empno;
```

如果视图是用 WITH CHECK OPTION 子句定义的，并且键预留表是重复的，那么就不能通过这样的视图来删除行。

```
CREATE VIEW emp_mgr AS
SELECT e1.ename, e2.ename mname
FROM emp e1, emp e2
WHERE e1.mgr = e2.empno
WITH CHECK OPTION;
```

在该视图上不能执行删除，因为该视图包括了键预留表的自身联结。

INSERT 语句 在 emp_dept 视图上的如下 INSERT 语句是成功的：

```
INSERT INTO emp_dept (ename, empno, deptno)
VALUES ('KURODA', 9010, 40);
```

这条语句能运行是因为仅修改一个键预留基础表 emp，40 是 dept 表中的有效的 deptno（即满足在 emp 表上的 FOREIGN KEY 完整性约束）。

象下面那样的 INSERT 语句会失败，失败的原因与 UPDATE 在基础 emp 表中失败的原因相同：违反了 emp 表中 FOREIGN KEY 完整性约束（因为没有 deptno 77）。

```
INSERT INTO emp_dept (ename, empno, deptno)
VALUES ('KURODA', 9010, 77);
```

下面的 INSTER 语句也会失败，产生一个错误(ORA-01776 cannot modify more than one base table through a view（不能通过视图修改多个基础表））：

```
INSERT INTO emp_dept (empno, ename, loc)
VALUES (9010, 'KURODA', 'BOSTON');
```

INSERT 语句不能隐含地或明确地涉及非键预留表的列。如果联结视图是用 WITH CHECK OPTION 子句定义的，那么就不能对该视图执行 INSERT 操作。

20.1.2.3 使用 UPDATABLE_COLUMNS 视图

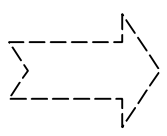
当要修改联结视图时，表 20-1 中所描述的视图对你有帮助。

表 20-1 UPDATABLE_COLUMNS 视图

视图名称	说明
DBA_UPDATABLE_COLUMNS	显示所有表及视图中的所有可修改的列。
ALL_UPDATABLE_COLUMNS	显示用户可访问的所有表及视图中的所有可修改的列。
USER_UPDATABLE_COLUMNS	显示用户模式中所有表及视图中的所有可修改的列。

下面列出了视图 emp_dept 中可更新的列。

```
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.

```

20.1.3 更改视图

用 ALTER VIEW 语句仅能明确地重新编译一个无效的视图。如果要改变视图的定义，参见第 20 章的“替换视图”。

ALTER VIEW 语句能够在运行前发现重新编译的错误。为了确保这种改变不影响依赖于它的视图和其他对象，在改变了它的基础表中的任意一个之后，应该明确地重新编译视图。

要想使用 ALTER VIEW 语句，该视图必须要在你的模式中，或者你必须拥有 ALTER ANY TABLE 系统权限。

20.1.4 删除视图

可以删除包含在你的模式中的任何视图。为了删除其他用户模式中的视图，你必须拥有 DROP ANY VIEW 系统权限。使用 DROP VIEW 语句删除视图。例如，下面的语句删除 emp_dept 视图：

```
DROP VIEW emp_dept;
```

20.1.5 替换视图

要想替换视图，必须拥有删除和创建一个视图所需的所有权限。如果必须修改视图的定义，就必须替换视图；而不能修改视图的定义。可用下面的几种方式替换视图：

- 删除并重建视图。

警告：当视图被删除时，相应的对象权限的所有授权也被从角色和用户中撤消。重新创建视图之后，必须重新授予权限。

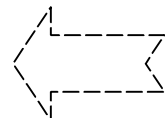
- 可以用包含 OR REPLACE 选项的 CREATE VIEW 语句重新定义视图。OR REPLACE 选项替换视图的当前定义并且保留当前的安全授权。例如，假设像前面介绍的那样创建过 sales_staff 视图，另外，为角色和其他用户授予过几种对象权限。但是，现在需要重新定义 sales_staff 视图来改变在 WHERE 子句中指定的部门号。

可用下面的语句替换当前版本的 sales_staff 视图。

```

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，要考虑如下几个方面：

- 替换视图也就替换了数据字典中视图的定义。并不影响视图所涉及的所有下层对象。
- 如果前面已经定义了 CHECK OPTION 中的约束，而新的视图定义中又不包含这种约束，该约束就被删除了。
- 依赖于被替换视图的所有视图及 PL/SQL 程序单元都变成无效（不能使用）。关于 Oracle 如何管理这种依赖性的详细内容，参见第 21 章的“管理对象依赖性”。

20.2 管理序列

序列是多个用户能够以此产生唯一整数的数据库对象。可用序列来自动产生主键值。本节描述管理序列的多个方面，包含如下内容：

- 创建序列
- 更改序列
- 删除序列

20.2.1 创建序列

要想在你的模式中创建序列，你必须拥有 CREATE SEQUENCE 系统权限。为了在其他用户的模式中创建序列，必须拥有 CREATE ANY SEQUENCE 权限。

使用 CREATE SEQUENCE 语句创建序列。例如，下面的语句创建一个用于给 emp 表中 empno 列产生雇员号的序列：

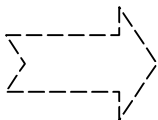
```
CREATE SEQUENCE emp_sequence  
INCREMENT BY 1  
START WITH 1  
NOMAXVALUE  
NOCYCLE  
CACHE 10;
```

CACHE 选项预分配一组序列号，并且把它们存在内存中，以便更快地访问序列号。当高速缓存中最后一个序列号被使用了，Oracle 就再读入另外一组序列号，并将其放入到高速缓存中。

如果选择用高速缓存存放一组序列号，Oracle 可能跳过序列号。例如，当实例不正常关机时（例如，当实例出现失败或发布了 SHUTDOWN ABORT 语句），已经被放在高速缓存中但没有被使用的序列号就会丢失。同样，已经被使用但没有被保存的序列号也会丢失。在导入和导出之后，Oracle 也可能跳过已在高速缓存中的序列号。

20.2.2 更改序列

要想更改序列，你的模式中必须包含序列，或者你必须拥有 ALTER ANY SEQUENCE 系统权限。能更改序列，以修改定义它是如何产生序列号的任何参数（除序列的起始号以



外)。为了更改序列的起始点，可以先删除序列，然后再重新创建序列。当在序列号上执行 DDL 时，就会丢失高速缓存中的值。

使用 ALTER SEQUENCE 语句更改序列。例如，下面的语句改变 emp_sequence 序列：

```
ALTER SEQUENCE emp_sequence
INCREMENT BY 10
MAXVALUE 10000
CYCLE
CACHE 20;
```

20.2.3 删除序列

可以删除你的模式中的任何序列。要想删除其他模式中的序列，你必须拥有 DROP ANY SEQUENCE 系统权限。如果不再需要某个序列，可用 DROP SEQUENCE 语句删除它。例如：下面的语句删除 order_seq 序列：

```
DROP SEQUENCE order_seq;
```

删除序列时，它的定义就会从数据字典中删除。该序列的所有同义词保留下来，但参照这些同义词时，会返回一个错误。

20.3 管理同义词

同义词是模式对象的别名。同义词通过屏蔽对象的名称和拥有者，以及通过为分布式数据库的远程对象提供位置透明性来提供一个安全层。同义词使用方便，为数据库用户降低了 SQL 语句的复杂性。

同义词允许重新命名或移动下层对象，在那儿仅需要重新定义同义词，而基于同义词的应用无须做任何修改仍能继续工作。

即能创建公共的也能创建私有的同义词。公共同义词由名为 PUBLIC 的特殊用户组所拥有，并且数据库中的每个用户都可访问。私有同义词被包含在特殊用户的模式中，并且仅仅用户和用户的受让人可使用。

本节包括如下同义词管理信息：

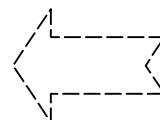
- 创建同义词
- 删除同义词

20.3.1 创建同义词

要想在你拥有的模式中创建私有同义词，你必须拥有 CREATE SYNONYM 权限。要想在其他用户的模式中创建私有同义词，你必须拥有 CREATE ANY SYNONYM 权限。要想创建公共同义词，必须拥有 CREATE PUBLIC SYNONYM 系统权限。

使用 CREATE SYNONYM 语句创建同义词。在创建同义词时，下层模式对象不必存在，你也不需要权限以访问对象。下面的语句在包含在 jward 模式中的 emp 表上创建名为 public_emp 的公共同义词：

```
CREATE PUBLIC SYNONYM public_emp FOR jward.emp;
```



20.3.2 删除同义词

你能删除你的模式中的任何私有同义词。要想删除其他用户的模式中的私有同义词，你必须拥有 **DROP ANY SYNONYM** 系统权限。要想删除公共同义词，你必须拥有 **DROP PUBLIC SYNONYM** 系统权限。

使用 **DROP SYNONYM** 语句删除不再需要的同义词。要想删除私有同义词，就省略 **PUBLIC** 关键字。要想删除公共同义词，就要包括 **PUBLIC** 关键字。

例如，下面的语句删除名为 **emp** 的私有同义词：

```
DROP SYNONYM emp;
```

下面的语句删除名为 **public_emp** 的公共同义词：

```
DROP PUBLIC SYNONYM public_emp;
```

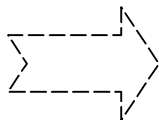
删除序列时，它的定义就会从数据字典中删除。该序列的所有同义词保留下来，但参照这些同义词时，会返回一个错误。

删除同义词时，它的定义就会从数据字典中删除。所有参照被删除同义词的对象仍保留下来，但它们变成无效（不能使用）。有关删除同义词对其他模式对象产生的影响的更详细信息，参见第 21 章的“管理对象依赖性”。

20.4 查看有关视图、同义词和序列的信息

下面的视图显示了关于视图、同义词和序列的信息：

视图	说明
DBA_VIEWS	DBA 视图描述了数据库中的所有视图。ALL 视图局限于当前用户可访问的视图。USER 视图局限于当前用户所拥有的视图。
ALL_VIEWS	
USER_VIEWS	
DBA_SYNONYMS	这些视图描述同义词。
ALL_SYNONYMS	
USER_SYNONYMS	
DBA_SEQUENCES	这些视图描述序列。
ALL_SEQUENCES	
USER_SEQUENCES	
DBA_UPDATABLE_COLUMNS	这些视图描述联结视图中的所有可更新的列。
ALL_UPDATABLE_COLUMNS	
USER_UPDATABLE_COLUMNS	



第 21 章 模式对象的常规管理

本章描述模式对象的管理问题，这种问题常见于多种模式对象中。包含如下内容：

- 在单个操作中创建多个表和视图
- 更名模式对象
- 分析表、索引和簇
- 舍弃表和簇
- 启用和停用触发器
- 管理完整性约束
- 管理对象依赖性
- 管理对象名称解析
- 改变数据字典的存储参数
- 显示有关模式对象的信息

21.1 在单个操作中创建多个表和视图

可用 `CREATE SCHEMA` 语句，在一个操作中创建几个表和视图并授予权限。如果想在单个操作中创建几个表、视图并授权，则 `CREATE SCHEMA` 语句就有用了。如果单个的表、视图、或授权失败，则全部语句回滚。既没有创建对象，也没有授予权限。

特别是，`CREATE SCHEMA` 语句仅能包括 `CREATE TABLE`、`CREATE VIEW` 和 `GRANT` 语句。必须具有所需的权限才能发布所包含的语句。使用 `CREATE USER` 语句创建了用户，就不能再创建模式，而更合适于组装模式。

下面的例子创建了两个表和一个视图，此视图联结了两个表中的数据：

```
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 语句不支持针对 ANSI CREATE TABLE、CREATE VIEW 语句的 Oracle 扩展，包括 STORAGE 子句。

21.2 重命名模式对象

要想重命名对象，该对象必须要在你的模式中。重命名模式对象可以使用下面两种方法中的任何一种：

- 删除和重建对象
- 使用 RENAME 语句重命名对象

如果删除和重建一个对象，授予给对象的所有权限将丢失。当重新创建对象时，必须再重新授予权限。

做为选择，可用 RENAME 语句，来重命名一个表、视图、序列或表、视图、序列的私有同义词。使用 RENAME 语句时，对对象进行的完整性约束、索引、授权也将改成新名称。例如，下面的语句对 sales_staff 视图进行更名，新名字为 dept_30：

```
RENAME sales_staff TO dept_30;
```

注意：不能对存储 PL/SQL 程序单元、公共同义词、索引或簇使用 RENAME 语句。要重命名这种对象，必需删除它再重新创建它。

在重命名一个模式对象之前，考虑如下效果：

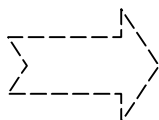
- 依赖于被重命名对象的所有视图和 PL/SQL 程序单元变为无效，在下次使用前，需要重新编译。
- 当使用被重命名对象的所有同义词时，返回一个错误。
- 有关 Oracle 如何管理对象依赖性的详细内容参见本章“管理对象依赖性”。

21.3 分析表、索引和簇

可以分析表、索引或簇来采集其有关的数据，或校验其存储格式的合法性。

还可以分析这些模式对象以便收集或更新指定对象的统计数据。当发布 DDL 语句时，参照对象的统计数据被用于确定该语句的最有效的执行方案。这种优化叫做“基于成本的优化”。统计数据存储在数据字典中。

可以分析表、索引、簇以校验对象的结构。例如，在罕见的情况下，如硬件或其他系统失败，索引被破坏了，不能正确执行。校验索引，就能确认索引中的每一项指向相关表的正确的行，如果模式对象被破坏了，就要删除此模式对象再重新创建它。



可以分析表或簇，来收集有关表或簇的链接列的信息。分析结果可用于确定是否有足够的空间更新行。例如，这些信息能够显示是否适当设置了表或簇的 PCTFREE。

为了分析表、簇或索引，就必须拥有该表、簇或索引，或者具有 ANALYZE ANY 系统权限。

本节讨论如下内容：

- 使用表、索引和簇的统计表
- 校验表、索引、簇和物化视图
- 列出表和簇的链接列

有关分析索引结构表的详细信息，参见第 15 章的“分析索引结构表”

21.3.1 使用表、索引和簇的统计表

使用 ANALYZE 语句，能把有关表、索引和簇的物理存储特性的统计数据，收集并存储在数据字典中。使用基于成本的优化，为访问被分析对象的 SQL 语句选择最有效率的执行方案时，Oracle 就能利用这些统计数据。也可用该语句产生的统计数据来编写访问被分析对象的有效率的 SQL 语句。

可以选择如下 ANALYZE 语句的子句中的任何一个来收集统计数据。

• COMPUTER STATISTICS

当计算统计数字时，扫描整个对象，收集关于此对象的数据。Oracle 用这些数据来计算此对象的精确的统计数据。在这些计算出的统计数据中，整个对象的微小变化都被计算出来。因为为了收集计算统计数据的信息要扫描整个对象，所以对象的体积越大，收集所需信息所要做的工作就越多。

• ESTIMATE STATISTICS

当估计统计数据时，Oracle 收集对象的各部分有代表性的信息。该信息的子集提供了有关该对象合理的、估计的统计数据。估计出的统计数据的精确度取决于 Oracle 所使用的样例的代表性。因为收集估计统计数据，只是扫描对象的几个部分，所以能快速地分析一个对象。也可以随意地指定 Oracle 在做估计时，所使用的行的数量和百分比。

注意：当对一个表或簇计算统计数据时，执行这种计算所需的临时空间的大小与指定的行数有关。对于 COMPUTER STATISTICS 来说，需要有足够的临时空间来容纳和分类整个表和每个行的一点覆盖。对于 ESTIMATE STATISTICS 来说，需要有足够的临时空间来容纳和分类所需行的样例和每行的一点覆盖。对于索引来说，分析时不需要临时空间。

21.3.1.1 使用 ANALYZE 语句计算统计数据

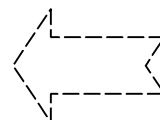
下面的语句计算 emp 表的统计数据：

```
ANALYZE TABLE emp COMPUTE STATISTICS;
```

下面的查询估计 emp 表的统计数据，使用默认的 1064 行的统计样例：

```
ANALYZE TABLE emp ESTIMATE STATISTICS;
```

为了指定 Oracle 将使用的统计样例，使用带有 ESTIMATE STATISTICS 选项的包含 SAMPLE 选项的语句。可以指定一个整数，代表行或索引值的数目、或表中行或索引值的



百分比。如下例语句显示了每种选项的例子：

```
ANALYZE TABLE emp
ESTIMATE STATISTICS
SAMPLE 2000 ROWS;
ANALYZE TABLE emp
ESTIMATE STATISTICS
SAMPLE 33 PERCENT;
```

无论什么情况，如果指定的百分比大于 50，或者行的数目、或索引值大于对象中该值的 50%，Oracle 就计算精确统计数据，而不是估计统计数据。

21.3.1.2 收集什么样的统计数据

本节列出了从表、索引、簇收集到的统计数据。

注意：符号*表示当计算统计数据时，数目总是一个精确的值。

表的统计数据

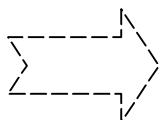
- 行数
- 已经使用的数据块数*
- 从未使用的数据块数
- 平均可用空闲空间
- 链接行的数目
- 平均行长度
- 列中不同的值的数目
- 列的下限值*
- 列的上限值*

注意：当分析表时，自动生成与表相关的所有索引的统计数据。

21.3.1.3 索引的统计数据

- 索引层次*
- 叶子数据块的数目
- 不同的键的数目
- 每个键的叶子数据块的平均数目
- 每个键的数据块的平均数目
- 分簇因子

注意：如果在一个已经标记为 UNUSABLE 的索引上，使用 ANALYZE 语句，会收到一个错误信息。当分析一个表的时候，Oracle 也为该表的每个索引搜集统计数据，所以如果任何索引标记成 UNUSABLE，就会收到一个错误。为了使 ANALYZE 语句成功，必须删除和重建标记为 UNUSABLE 的索引。可任选地，指定一个 FOR 子句，使分析操作跳过采集索引的统计数据。



21.3.1.4 簇的统计数据

可以从簇中收集到的唯一统计数据是簇键链的平均长度。能估计或计算这种统计数据。当分析簇的统计数据时，簇中表的统计数据 and 所有与簇中表相关的索引（包括簇键索引）的统计数据也被自动收集。

21.3.1.5 查看对象的统计数据

对于对象的统计数据无论是计算的还是估计的，其统计数据都存储在数据字典中。使用下面的数据字典视图能查询这些统计数据。

视图	说明
USER_INDEXES	该视图包含数据库中索引的描述，包括用 ANALYZE 收集到的索引统计数据。视图的类型（USER，ALL，DBA）决定显示什么索引项。
ALL_INDEXES	
DBA_INDEXES	
USER_TABLES	该视图包含数据库中相关表的描述，包括用 ANALYZE 收集的统计数据。
ALL_TABLES	
DBA_TABLES	
USER_TAB_COLUMNS	该视图包含数据库中列、视图和簇的列的描述，包括用 ANALYZE 收集的统计数据。
ALL_TAB_COLUMNS	
DBA_TAB_COLUMNS	

注意：这些视图中的行只包含已经采集了统计数据的索引、表和簇的统计数据列中的项目。每次 ANALYZE 对象时，这些项目都被更新。

21.3.1.6 删除模式对象的统计数据

从数据字典中删除表、索引或簇的统计数据，可用带有 DELETE STATISTICS 选项的 ANALYZE 语句。例如，如果不想将基于成本的优化用于涉及该对象的语句，就可能想删除该对象的统计数据。下面的语句就是从数据字典中删除 emp 表的统计数据：

```
ANALYZE TABLE emp DELETE STATISTICS;
```

21.3.1.7 共享 SQL 和分析统计数据

分析一个表、簇、索引可能会影响当前位于共享池中的当前共享 SQL 语句。无论什么时候分析对象，以便更新或删除统计数据，所有参照被分析对象的共享 SQL 语句都被从内存中清空，所以，下一步执行的语句利用新的统计数据。

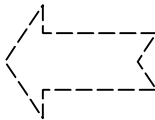
21.3.1.8 计算统计数据的某些可选的方法

某些 PL/SQL 包，事实上允许执行 ANALYZE 语句，此处简要地讨论一下：

DBMS_STATS 这是非常强大的包，它即允许收集统计数据，包括利用并行执行，又能进行统计数据的外部处理。统计数据可以被存储在数据字典外部的表中，此时处理统计数据不影响优化器。统计数据能在数据库间拷贝或者备份。

DBMS_UTILITY 此包包含 ANALYZE_SCHEMA 过程，带有两个参数：

- 模式的名称



- 分析的方法 (COMPUTE、ESTIMATE 或 DELETE)。

此包收集模式中所有对象的统计数据。

DBMS_DDL 此包包含 ANALYZE_OBJECT 过程，带有四个参数：

- 对象的类型 (CLUSTER, TABLE 或 INDEX)
- 对象的模式
- 对象的名称
- 分析方法 (COMPUTE, ESTIMATE 或 DELETE)。

此包收集对象的统计数据。

21.3.2 校验表、索引、簇和物化视图

为了校验表、索引、簇和物化视图的结构完整性，使用带有 VALIDATE STRUCTURE 选项的 ANALYZE 语句。如果结构是好的，则不会返回错误；但如果结构是坏的，则会收到一个错误消息。

如果表、索引、簇是坏的，就应删除并重新创建它。如果物化视图是坏的，就执行一个完全刷新，并确保你已经解决了这个问题，如果这个问题没有解决，只好删除并重新创建该物化视图。

下面的语句分析 emp 表：

```
ANALYZE TABLE emp VALIDATE STRUCTURE;
```

通过包含 CASCADE 选项，能校验一个对象和所有与之相关的对象（如索引），下面的语句验证 emp 表和所有的与之相关的索引：

```
ANALYZE TABLE emp VALIDATE STRUCTURE CASCADE;
```

可以表明，你想在 DML 作用于被校验对象上时，执行联机结构校验。用正在进行的作用于对象的 DML 来校验，可能对性能有一点影响，但这通过联机执行 ANALYZE 的灵活性来补偿。

下面的语句联机地校验 emp 表及其所有相关的索引。

```
ANALYZE TABLE emp VALIDATE STRUCTURE CASCADE ONLINE;
```

21.3.3 列出表和簇的链接行

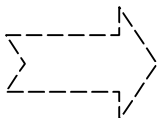
使用带 LIST CHAINED ROWS 选项的 ANALYZE 语句，能查看表或簇中链接的或迁移的行。这条语句的执行结果存储在指定的表中，该表被明确地创建，以便直接接收由 LIST CHAINED ROWS 子句返回的信息。

21.3.3.1 创建 CHAINED_ROWS 表

为了创建用于接收由 ANALYZE LIST CHAINED ROWS 语句返回的数据的表，执行 UTLCHAIN.SQL 或者 UTLCHN1.SQL 脚本即可。这些脚本是由 Oracle 提供的。在提交脚本的用户模式中，创建名为 CHAINED_ROWS 的表。

注意：为了创建 CHAINED_ROWS 表而选择执行的脚本依赖于数据库的兼容级别和所分析的表的类型。

创建了 CHAINED_ROWS 表之后，可用 ANALYZE 语句的 INTO 子句指定该表。例如，



下面的语句将包含有关在 emp_dept 簇中链接行信息的行插入到 CHAINED_ROWS 表中。

```
ANALYZE CLUSTER emp_dept LIST CHAINED ROWS INTO CHAINED_ROWS;
```

21.3.3.2 删除表中的迁移或链接行

使用 CHAINED_ROWS 表中的信息，能减少或删除现存表中的迁移或链接行。步骤如下：

①使用 ANALYZE 语句收集迁移或链接行的信息。

```
ANALYZE TABLE order_hist LIST CHAINED ROWS;
```

②查询输出表：

```
SELECT *
FROM CHAINED_ROWS
WHERE TABLE_NAME = 'ORDER_HIST'
OWNER_NAME TABLE_NAME CLUST... HEAD_ROWID TIMESTAMP
```

```
-----
```

```
SCOTT ORDER_HIST ... AAAA1uAAHAAAAA1AAA 04-MAR-96
```

```
SCOTT ORDER_HIST ... AAAA1uAAHAAAAA1AAB 04-MAR-96
```

```
SCOTT ORDER_HIST ... AAAA1uAAHAAAAA1AAC 04-MAR-96
```

该输出列出了或者是迁移或者是链接的所有行。

③如果输出表显示出有许多迁移或链接行，那么就继续完成下面步骤来删除迁移行。

④创建一个与现存表有相同列的中间表，以便保留迁移或链接行：

```
CREATE TABLE int_order_hist
AS SELECT *
FROM order_hist
WHERE ROWID IN
(SELECT HEAD_ROWID
FROM CHAINED_ROWS
WHERE TABLE_NAME = 'RDER_HIST');
```

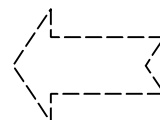
⑤从现存的表中删除迁移或链接行：

```
DELETE FROM order_hist
WHERE ROWID IN
(SELECT HEAD_ROWID
FROM CHAINED_ROWS
WHERE TABLE_NAME = 'RDER_HIST');
```

⑥把中间表中的行插入到现存表中：

```
INSERT INTO order_hist
SELECT *
FROM int_order_hist;
```

⑦删除中间表



```
DROP TABLE int_order_history;
```

⑧从输出表中删除步骤1所收集的信息：

```
DELETE FROM CHAINED_ROWS  
WHERE TABLE_NAME = 'RDER_HIST'
```

⑨再次使用 ANALYZE 语句，查询输出表：

⑩出现在输出表中的所有行都是链接行。通过增加数据块的大小就能消除链接行，在所有情况下都避免链接不太可能。对于具有属性为 LONG，或长 CHAR 或者 VARCHAR2 列的表而言，链接问题常常是不可避免的。

21.4 舍弃表和簇

能删除表中的所有行，也能删除一组成簇的表（一簇表）中的所有的行，删除行之后，表（或簇）仍然存在，但完全是空的。例如，考虑一个包含每月数据的表，在每个月的月末，数据存档之后，就需要清空表（删除所有行）。

可以用下面的几种方法，从表中删除所有的行：

- 使用 DELECT 语句。
- 使用 DROP 或者 CREATE 语句。
- 使用 TRUNCATE 语句。

下面的几节中将分别讨论这几种方法。

21.4.1 使用 DELETE

使用 DELETE 语句能删除表中的行。例如，下面的语句删除 emp 表中的所有行：

```
DELETE FROM emp;
```

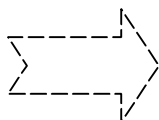
如果表或簇中有很多行，当使用 DELETE 语句删除行的时候，会消耗很多系统资源。例如，CPU 时间、重做日志空间、表和与之相关联的索引的回滚段空间都需要资源。同样，每删除一行，就启动一次触发器。原来分配给所产生的空表和空簇的空间仍保持与那个对象相关。使用 DELETE 语句，可以有选择地删除某些行，然而，TRUNCATE 和 DROP 语句影响整个对象。

21.4.2 使用 DROP 和 CREATE

能删除一个表，然后再重新创建该表，例如，下面的语句先删除 emp 表，之后再重新创建 emp 表：

```
DROP TABLE emp;  
CREATE TABLE emp ( ... );
```

当删除和重新创建表或簇时，所有与之相关联的索引、完整性约束和触发器也被删除，所有依赖于被删除的表或簇表的对象变成无效。同样，所有针对被删除表或簇表的授权也被删除。



21.4.3 使用 TRUNCATE

使用 TRUNCATE 语句能删除表中的所有行。例如，下面的语句删除 emp 表：

```
TRUNCATE TABLE emp;
```

使用 TRUNCATE 语句，为我们提供了一种快速、有效地删除表或簇的所有行的方法。TRUNCATE 语句不产生任何回滚信息，并且立即提交。此语句是一个 DDL 语句，而且不能回滚。TRUNCATE 语句并不影响与被删除的表相关联的任何结构（约束和触发器）或者授权。TRUNCATE 语句还指定在删除之后，当前分配给该表的空间是否返回给容纳该表的空间。

能在用户的相关模式中删除任何表和簇。同样，拥有 DROP ANY TABLE 系统权限的任何用户，都能删除任何模式中的表或簇。

在删除包含一个父键的表或簇表之前，在不同表中的所有的所参照的外键必须被停用。作为自参照的约束不必停用。

在使用 TRUNCATE 语句删除表中的行时，不触发与表相关联的的触发器。另外，尽管启用了审计，TRUNCATE 语句也不产生相应于 DELETE 语句的任何审计信息。然而，会为正发布的 TRUNCATE 语句产生单个审计记录。关于审计的更为详细的内容，参见第 26 章的“审计数据库的使用”。

不能删除散列簇。同样，也不能单独删除散列簇或者索引簇中的表，删除索引簇就是删除簇中的所有表的所有行。如果必须删除一个单独簇表中的所有行，就使用 DELETE 语句，或者先删除表再重新创建该表。

TRUNCATE 语句的 REUSE STORAGE 或者 DROP STORAGE 选项，控制是否当前分配给表或簇的空间，在删除之后，返还给容纳该表的空间。默认选项为 DROP STORAGE，将分配给结果表盘区的数量，减少到 MINEXTENTS 的初始设置值。然后，被释放的盘区返回给系统，就可以由其他对象使用。

另一个，REUSE STORAGE 选项指定将当前分配给表或簇的空间，仍然分配给它。例如，下面的语句删除 emp_dept 簇，将以前分配给簇的所有可用的盘区，用于以后的插入和删除。

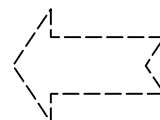
```
TRUNCATE CLUSTER emp_dept REUSE STORAGE;
```

REUSE 或者 DROP STORAGE 选项也适用于任何与之相关联的索引。当删除表或簇时，所有的与之相关联的索引也被删除。当删除表、簇或者与之相关联的索引时，存储参数不会改变。

21.5 启用和停用触发器

数据库触发器是存储在数据库中的过程，当满足特定的条件时，如往表中增加一行时，就能被激活（触发）。使用触发器能补充 Oracle 标准功能，提供高客户化的数据库管理系统。例如，能创建一个触发器来限制对一个表的 DML 操作，仅允许在正规的商业时间内发布的语句。

数据库触发器能与表、模式或数据库相关联。在如下情况下，触发器被隐含地激活：



- 在一个相关表上执行 DML 语句 (INSERT, UPDATE, DELETE)
- 在数据库或模式中的对象上, 执行某个 DDL 语句 (如 ALTER, CREATE, DROP)
- 特殊的数据库事件发生 (STARTUP, SHUTDOWN, SERVERERROR)。

上面没有列出所有的情况。

用 CREATE TRIGGER 语句创建触发器, 可以将它们定义成在触发事件之前 (BEFORE) 或之后 (AFTER) 或代替 (INSTEAD) 它进行触发。下面的语句在表 scott.emp 上创建触发器 scott.emp_permit_changes, 在执行任何指定的语句之前, 触发触发器。

```
CREATE TRIGGER scott.emp_permit_changes
BEFORE
DELETE OR INSERT OR UPDATE
ON scott.emp
.
pl/sql block
.
```

可用发布 DROP TRIGGER 语句, 来从数据库中删除触发器。

触发器有两种不同的模式:

- 启用

如果发布了触发语句, 并且触发约束条件 (如果有的话) 评估为真, 一个启用的触发器就执行触发体。默认时, 第一次创建触发器时, 触发器就被启用。

- 停用

一个停用的触发器不能执行触发体, 即便是发布了触发语句, 并且触发约束条件 (如果有的话) 评估为真时, 也不能执行。

使用 ALTER TABLE 语句启用或停用触发器时, 必须拥有表及对表的 ALTER 对象的权限, 或者拥有 ALTER ANY TABLE 系统权限。使用 ALTER TRIGGER 语句启用或停用一个单独的触发器时, 必须拥有触发器或者拥有 ALTER ANY TRIGGER 系统权限。

21.5.1 启用触发器

使用带有 ENABLE 选项的 ALTER TRIGGER 语句, 启用一个停用的触发器。为了启用一个建于 inventory 表上的名为 reorder 的停用触发器, 输入下面的语句:

```
ALTER TRIGGER reorder ENABLE;
```

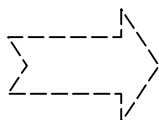
为了启用定义在指定表上的所有触发器, 使用带有 ENABLE ALL TRIGGERS 选项的 ALTER TABLE 语句。为了启用定义在 INVENTORY 表上的所有触发器, 输入下面的语句:

```
ALTER TABLE inventory
ENABLE ALL TRIGGERS;
```

21.5.2 停用触发器

如果下列条件之一成立, 就考虑临时停用一个触发器:

- 触发器参照的对象是不可用的。



- 必须执行装载大量数据并希望快速处理数据而不激活触发器。
- 正在往使用触发器的表中装载数据。

使用带有 `DISABLE` 选项的 `ALTER TRIGGER` 语句，停用触发器。为了停用建于 `inventory` 表上的触发器 `reorder`，输入下面的语句：

```
ALTER TRIGGER reorder DISABLE;
```

使用带有 `DISABLE ALL TRIGGERS` 选项的 `ALTER TABLE` 语句，可以同时停用与表相关联的所有触发器。例如，为了停用为 `INVENTORY` 表所定义的所有触发器，输入下面的语句：

```
ALTER TABLE inventory
DISABLE ALL TRIGGERS;
```

21.6 管理完整性约束

完整性约束是约束表中一个或多个列值的规则。约束子句可以出现在 `CREATE TABLE` 语句或 `ALTER TABLE` 语句中，标明完整性约束所涉及的列和标明约束条件。

本节简要地讨论约束的概念，及说明定义并管理完整性约束的 SQL 语句。本节包括如下内容：

- 完整性约束的状态
- 在定义中设置完整性约束
- 修改或删除现存的完整性约束
- 延时约束检查
- 报告约束异常

21.6.1 完整性约束的状态

能指定启用 (`ENABLE`) 或者停用 (`DISABLE`) 约束。如果启用约束，当在数据库中输入或者更新数据时要进行数据检查，不符合约束规则的数据不能输入数据库。如果停用约束，那么不符合规则的数据也能输入到数据库中。

另外，也能指定表中的现有数据必须符合约束 (`VALIDATE`)；相反地，如果指定为 (`NOVALIDATE`)，就不能确保现有的数据符合约束。

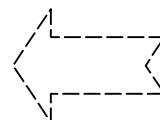
定义在表上的完整性约束可能处于下面几种状态之一：

- `ENABLE, VALIDATE`
- `ENABLE, NOVALIDATE`
- `DISABLE, VALIDATE`
- `DISABLE, NOVALIDATE`

21.6.1.1 停用约束

为了强制执行由完整性约束所定义的规则，约束应总是处于启用状态。然而，由于下面执行时的几个原因，可以考虑临时停用表的完整性约束。

- 当往表中装载大量数据时



- 当执行批操作使得表有大规模的改变（例如，往现有的序号中增加 1000，改变每个雇员的序号）时
- 当一次性导入或导出一个表时

在上面三种情况下，临时停用完整性约束能改善操作性能，特别是在配置数据仓库时更能改善其操作性能。

当停用约束时，很可能输入违反约束的数据。所以，在完成上述所列的任何操作之后，应该总是启用约束。

21.6.1.2 启用约束

当启用约束时，违反约束的行不能插入到表中。但是，当停用约束时，这样的行就能插入到表中。这种行被认为对约束异常。如果约束处于启用非校验状态，那些在禁用约束时所输入的数据中，违反约束的结果仍然保留，违反约束的那些行一定要更新或者删除，以便约束处于已校验状态。

当启用一个约束时，能识别一个特殊的完整性约束的异常，参见本章的“报告约束异常”。违反约束的所有行都被记录在 EXCEPTIONS 表中，你可以细查此表。

21.6.1.3 启用非校验约束状态

当约束处于启用非校验状态时，为了符合约束，检查所有的后续语句。但是，不检查现有表中的任何数据。一个带有启用非校验约束的表可能包含无效数据，但不可能给它增加新的无效数据。在非校验状态下，启用约束对上载有效的 OLTP 数据的数据仓库配置中非常有用。

启用约束不需要校验。启用非校验约束比启用并且校验一个约束要快，还有，校验一个已经启用的约束在校验期间不需要任何 DML 锁定（这与校验一个以前被停用的约束不同）。该实施保证了在校验期间没有引入违反约束的数据。因此允许不带校验能够减少与启用约束相关的典型的停工时间。

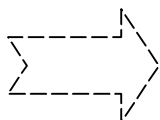
21.6.1.4 完整性约束的状态：过程和效益

按照下面的顺序使用完整性约束，确保最佳效益。

- ①停用状态
- ②执行操作（装载、导出、导入）
- ③启用非校验状态
- ④启用状态

按照如下顺序使用约束的几点好处是：

- 不留锁
- 所有的约束能够同时处于启用状态
- 约束启用是并行实现的
- 允许在表上进行同步操作



21.6.2 在定义中设置完整性约束

当一个完整性约束是用 CTEATE TABLE 或者 ALTER TABLE 语句定义的,它就能启用、停用或者校验或不校验,取决于你使用 ENABLE/DISABLE 子句时的说明。如果在约束的定义中,没有指定 ENABLE/DISABLE 子句,Oracle 自动启用和校验约束。

21.6.2.1 在定义中停用约束

下面的 CTEATE TABLE 和 ALTER TABLE 语句都定义并停用完整性约束:

```
CREATE TABLE emp (
empno NUMBER(5) PRIMARY KEY DISABLE, . . . ;
ALTER TABLE emp
ADD PRIMARY KEY (empno) DISABLE;
```

定义和停用一个完整性约束的 ALTER TABLE 语句,从来不会因为表中的行违反完整性约束而失败。由于其规则不是强制的,所以允许这种约束的定义。

21.6.2.2 在定义中启用约束

下面的 CTEATE TABLE 和 ALTER TABLE 语句都定义并启用完整性约束:

```
CREATE TABLE emp (
empno NUMBER(5) CONSTRAINT emp.pk PRIMARY KEY, . . . ;
ALTER TABLE emp
ADD CONSTRAINT emp.pk PRIMARY KEY (empno);
```

定义和试图启用一个完整性约束的 ALTER TABLE 语句,可能会由于表中的行违反完整性约束而失败。在这种情况下,此语句回滚,并且不存储也不启用约束定义。

当启用一个 UNIQUE 或者 PRIMARY KEY 约束时,就创建了一个与之相关的索引。参见第 21 章的“创建与约束相关的索引”。

21.6.3 修改或删除现存的完整性约束

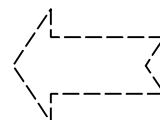
可用 ALTER TABLE 语句来启用、停用、修改或删除一个约束。当 Oracle 使用一个 UNIQUE 或者 PRIMARY KEY 索引实施一个约束,并且与那个索引相关的约束被删除或停用时,这个索引被删除,除非指定了其他情况。

当启用的外键参考 PRIMARY 或 UNIQUE 键时,不能停用或删除 PRIMARY 或 UNIQUE 键约束或索引。

21.6.3.1 停用已启用的约束

下面的语句停用完整性约束。第二个语句指定保留与之相关的索引。

```
ALTER TABLE dept
DISABLE CONSTRAINT dname_ukey;
ALTER TABLE dept
DISABLE PRIMARY KEY KEEP INDEX,
DISABLE UNIQUE (dname, loc) KEEP INDEX;
```



下面的语句启用非校验已停用的完整性约束：

```
ALTER TABLE dept
ENABLE NOVALIDATE CONSTRAINT dname_ukey;
ALTER TABLE dept
ENABLE NOVALIDATE PRIMARY KEY,
ENABLE NOVALIDATE UNIQUE (dname, loc);
```

下面的语句启用或校验已停用的完整性约束：

```
ALTER TABLE dept
MODIFY CONSTRAINT dname_key VALIDATE;
ALTER TABLE dept
MODIFY PRIMARY KEY ENABLE NOVALIDATE;
```

下面的语句启用已停用的完整性约束：

```
ALTER TABLE dept
ENABLE CONSTRAINT dname_ukey;
ALTER TABLE dept
ENABLE PRIMARY KEY,
ENABLE UNIQUE (dname, loc);
```

为了在一步中停用或删除 UNIQUE 键或 PRIMARY KEY 约束及其所有相关的 FOREIGN KEY 约束。可使用 DISABLE 或 DROP 子句的 CASCADE 选项。例如，下面的语句停用 PRIMARY KEY 约束和任何依赖于它的 FOREIGN KEY 约束。

```
ALTER TABLE dept
DISABLE PRIMARY KEY CASCADE;
```

21.6.3.2 删除完整性约束

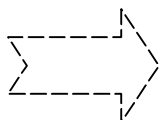
如果完整性约束实施的规则不再为真，或者不再需要该约束时，就可以删除一个完整性约束。可用带有 DROP 子句的 ALTER TABLE 语句来删除约束。下面的两条语句删除完整性约束，第二条语句保留了与 PRIMARY KEY 约束相关的索引：

```
ALTER TABLE dept
DROP UNIQUE (dname, loc);
ALTER TABLE emp
DROP PRIMARY KEY KEEP INDEX,
DROP CONSTRAINT dept_fkey;
```

如果有 FOREIGN KEY（外键）参照了 UNIQUE（唯一键）或 PRIMARY KEY（主键），那么在 DROP 语句中必须包括 CASCADE CONSTRAINT 子句，不然，就不能删除约束。

21.6.4 延时约束检查

当 Oracle 检查约束时，如果发现不满足该约束，就发出一个错误信号。可以延时检查约束的合法性，直到事务结束时。



当发布 SET CONSTRAINTS 语句时，SET CONSTRAINTS 模式延长事务的持续时间，直到另一个 SET CONSTRAINTS 语句重新设置模式。

注意：

- 不能在触发器内发布 SET CONSTRAINTS 语句。
- 可延时的唯一键和主键必须使用非唯一索引。

21.6.4.1 将所有约束设置成延时

在用于管理数据的应用中，在实际开始处理任何数据之前，必须将所有约束设置成延时。使用如下 DML 语句将所有可延时的约束设置成延时：

```
SET CONSTRAINTS ALL DEFERRED;
```

注意：SET CONSTRAINTS 语句仅适用于当前的事务。只要约束存在，则创建约束时指定的默认值就保留。ALTER SESSION SET CONSTRAINTS 语句只适用于当前会话。

21.6.4.2 检查提交（可选的）

通过在发布 COMMIT 语句之前，发布 SET CONSTRAINTS ALL IMMEDIATE 语句，就能在提交之前检查约束违反情况。如果存在任何约束问题，此语句失败，导致错误的约束被标识出来。如果提交违反约束的事务，则该事务被回滚，并收到一个错误消息。

21.6.5 报告约束异常

如果校验约束时存在异常，则返回一个错误而且完整性约束仍保持为未被校验。由于存在完整性约束异常，使一个语句不能正确执行时，则此语句被回滚。如果存在异常，直到更新或删除了约束的所有异常之后，才能校验约束。

使用 CREATE TABLE 语句不能确定哪一行违反完整性校验。为了确定哪一行违反了完整性约束，发布在 ENABLE 子句中带有 EXCEPTION 选项的 ALTER TABLE 语句。

EXCEPTION 选项将 ROWID、表拥有者、表名称和所有异常行的约束名称放到一个指定的表中。

在启用约束之前，应创建一个合适的异常报告表，用来接收来自于 ENABLE 子句的 EXCEPTION 选项的信息。也可以通过执行 UTLEXCPT.SQL 脚本或者 UTLEXPT1.SQL 脚本创建一个异常表。

注意：为创建 EXCEPTIONS 表所选择执行的脚本，取决于数据库的兼容性级别和所分析的表的类型。

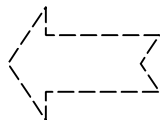
这两个脚本创建名为 EXCEPTIONS 的异常表。通过修改和重新提交该脚本，也能创建别的具有不同名称的异常表。

下面的语句尝试校验 dept 表的 PRIMARY KEY，如果存在异常，就将异常信息插入到 EXCEPTIONS 表中。

```
ALTER TABLE dept ENABLE PRIMARY KEY EXCEPTIONS INTO EXCEPTIONS;
```

如果在 dept 表中存在重复的主键值，并且在 dept 上 PRIMARY KEY 约束的名字是 sys_c00610，则上面的语句会把下面的行放到 EXCEPTIONS 表中：

```
SELECT * FROM EXCEPTIONS;
```



```
ROWID OWNER TABLE_NAME CONSTRAINT
```

```
-----
```

```
AAAAZ9AABAAABvqAAB SCOTT DEPT SYS_C00610
```

```
AAAAZ9AABAAABvqAAG SCOTT DEPT SYS_C00610
```

能提供更多信息的查询应该将异常报告中的行与主表中的行相连接，以便列出违反指定约束的实际行，如下例所示：

```
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
```

应从包含约束的表中，更新或者删除违反约束条件的所有行。当更新异常时，就必须把违反约束的值更改成与约束相符的值或空值。更新或删除了主表中的行以后，相应的异常报告表中的异常行也应删除，以避免与后来的异常表相混。更新主表和异常报告表的语句应在同一个事务中，以确保事物的一致性。

为了纠正前面例子中的异常，可以发布如下事务：

```
UPDATE dept SET deptno = 20 WHERE dname = 'RESEARCH';
```

```
DELETE FROM EXCEPTIONS WHERE constraint = 'SYS_C00610';
```

```
COMMIT;
```

管理异常的目标就是删除异常报告表中的所有异常。

注意：对于一个带有停用约束的表而言，当纠正其当前的异常时，很可能其他的用户发布的语句又产生了新的异常。通过在开始删除异常之前，启用非校验约束可以避免这种情况。

21.7 管理对象依赖性

本节描述各种对象的依赖性，包括如下内容：

- 手动重新编译视图
- 手动重新编译过程和函数
- 手动重新编译包

首先，看表 21-1，它说明了对象是如何受到它们所依赖的对象的改变的影响的。

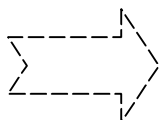


表 21-1 影响对象状态的操作

操作	对象的结果状态	依赖对象的结果状态
CREATE [TABLE SEQUENCE SYNONYM]	VALID 如果没有错误	没有变化 ¹
ALTER TABLE (ADD, MODIFY columns)	VALID 如果没有错误	INVALID
RENAME [TABLE SEQUENCE SYNONYM VIEW]		
DROP [TABLE SEQUENCE SYNONYM VIEW PROCEDURE FUNCTION PACKAGE]	无。对象被删除了。	INVALID
CREATE [VIEW PROCEDURE] ²	VALID 如果没有错误；INVALID 如果有语法或授权错误。	没有变化 ¹
CREATE OR REPLACE [VIEW PROCEDURE] ²	VALID 如果没有错误；INVALID 如果有语法或授权错误。	INVALID
REVOKE <i>object privilege</i> ³ ON <i>object</i> TO FROM <i>user</i>	没有变化	依赖于对象的所有用户对象是 INVALID ³
REVOKE <i>object privilege</i> ³ ON <i>object</i> TO FROM PUBLIC	没有变化	数据库中依赖于对象的所有对象是 INVALID ³
REVOKE <i>system privilege</i> ⁴ TO FROM <i>user</i>	没有变化	所有用户对象是 INVALID ⁴
REVOKE <i>system privilege</i> ⁴ TO FROM PUBLIC	没有变化	数据库中的所有对象是 INVALID ⁴

注：

¹ 如果以前对象不存在，能引起依赖对象变成 INVALID。

² 单独的过程和函数、包、触发器。

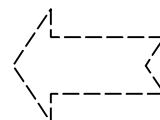
³ 仅为 DML 对象权限，包括 SELECT，INSERT，UPDATE，DELETE 和 EXECUTE；重新校验不需要重新编译。

⁴ 仅为 DML 系统权限，包括 SELECT，INSERT，UPDATE，DELETE ANY TABLE 和 EXECUTE ANY PROCEDURE；重新校验不需要重新编译。

在下次使用一个无效的视图或 PL/SQL 程序单元时，Oracle 会自动将其重新编译。另外，使用适当的带 COMPILE 子句的 SQL 语句，用户能强制 Oracle 重新编译视图或程序单元。当依赖视图或程序单元无效时，强制编译是经常使用的测试错误的方法，但当前不使用。在这些情况下，直到执行视图或程序单元时，Oracle 才自动重新编译。为了识别无效的依赖对象，查询视图 USER/ALL/DBA_OBJECTS。

21.7.1 手动重新编译视图

为了手动重新编译视图，必须拥有 ALTER ANY TABLE 系统权限，或者该视图必须包含在你的模式中。用带有 COMPILE 子句的 ALTER VIEW 语句来重新编译视图。下面的



语句重新编译包含在你的模式中的视图 emp_dept:

```
ALTER VIEW emp_dept COMPILE;
```

21.7.2 手动重新编译过程和函数

为了手动重新编译单独的过程，必须拥有 ALTER ANY PROCEDURE 系统权限，或者该过程必须包含在你的模式中。用带有 COMPILE 子句的 ALTER PROCEDURE/FUNCTION 语句来重新编译单独的过程和函数。下面的语句重新编译包含在你的模式中的存储过程 update_salary:

```
ALTER PROCEDURE update_salary COMPILE;
```

21.7.3 手动重新编译包

为了手动重新编译包，必须拥有 ALTER ANY PROCEDURE 系统权限，或者该包必须包含在你的模式中。用带有 COMPILE 子句的 ALTER PACKAGE 语句来重新编译包体或者包说明和包体。下面的语句只重新编译包 acct_mgmt 的包体:

```
ALTER PACKAGE acct_mgmt COMPILE BODY;
```

下面的例子编译 acct_mgmt 包的包体和包说明:

```
ALTER PACKAGE acct_mgmt COMPILE PACKAGE;
```

21.8 管理对象名称解析

在 SQL 语句中参照的对象名称是由分成的几段组成的。下面描述了 Oracle 是如何分解对象名称的。

①Oracle 试图修饰在 SQL 语句中所参照的名称的第一段。例如，在 scott.emp 中 scott 是第一段，如果仅有一段，就认为这段就是第一段。

a. Oracle 在当前的模式中，搜索名字与对象名称的第一段匹配的对象。如果没有找到这个对象，就继续执行步骤 b。

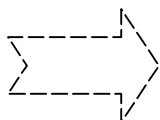
b. Oracle 搜索匹配对象名称第一段的公共同义词。如果没有找到，就继续执行步骤 c。

c. Oracle 搜索匹配对象名称第一段的模式。如果找到了，就返回到步骤 b，现在，用名称的第二段作为对象，在修饰的模式中查找。如果在前面修饰的模式中，第二段没有与一个对象相符合，或者没有第二段，Oracle 返回一个错误。

如果在步骤 c 中没有发现模式，则不能修饰对象，并且 Oracle 返回一个错误。

②模式对象已被修饰。名称的任何其他段必须与已找到的对象的一个有效部分相匹配。例如，如果 scott.emp.deptno 是一个名称，scott 被修饰为模式，emp 被修饰为表，则 deptno 就必须与列相对应（因为 emp 是表）。如果 emp 被修饰为包，则 deptno 就必须与包的公共常量、变量、过程、或函数相对应。

当全局对象名称用在分布式数据库中时，或者明确或者间接地在一个同义词内，本地 Oracle 按本地解析参照。例如，它把一个同义词解析成远程表的全局对象名称。部分被解析的语句被发送到远程数据库，远程的 Oracle 就像上面描述的那样，完成对象的解析。



21.9 修改数据字典的存储参数

如果你的数据库很大，或者包含异乎寻常的大量的对象、表的列、约束定义、用户、或其他定义，形成数据字典的表在某些情况可能不能获得附加的盘区。例如，数据字典表需要附加盘区，但是在 SYSTEM 表空间中，没有足够的连续空间。如果发生这种情况时，即便是打算用于装载对象的表空间有足够的空间，也不能创建新的对象。为了补救这种情况，就要改变基础数据字典表的存储参数，正象为用户创建的段改变存储设置一样。这样就允许为数据字典表分配更多的盘区。例如，可以调整数据字典表的 NEXT 值或者 PCTINCREASE 的值。

警告：当改变数据字典对象的存储设置时要多加小心。如果选择了不适当的设置，可能会破坏数据字典的结构，并被迫重建全部数据库。例如，将数据字典表 USERS\$ 的 PCTINCREASE 设置成 0 并将 NEXT 的值设置成 2K，则此表迅速到达段的最大盘区数。那样，要是不导出、重新创建、导入全部数据库的话，就不能创建任何更多的用户和角色。

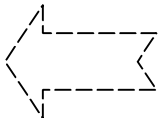
本节描述改变数据字典存储参数的有关方面，包括如下内容：

- 数据字典的结构
- 需要修改数据字典存储的错误

21.9.1 数据字典的结构

下面的表和簇包含了数据库中用户创建的所有对象的定义：

对象	说明
SEG\$	在数据库中定义的段（包括临时段）
OBJ\$	在数据库中用户定义的对象（包括簇表）；通过 I_OBJ1 和 I_OBJ2 索引
UNDOS	在数据库中定义的回滚段；通过 I_UNDO 索引
FET\$	没有分配给任何段的可用的空闲盘区
UET\$	已分配给段的盘区
TSS	在数据库中定义的表空间
FILES	形成数据库的文件；通过 I_FILE1 索引
FILEXTS	带有 AUTOEXTEND 选项设置的数据文件
TAB\$	在数据库中定义的表（包括簇表）；通过 I_TAB1 索引
CLUS	在数据库中定义的簇
IND\$	在数据库中定义的索引；通过 I_IND1 索引
ICOL\$	在其上定义过索引的列（包括在组合索引中每个列的单独的项）；通过 I_ICOL1 索引
COL\$	在数据库的表中定义的列；通过 I_COL1 和 I_COL2 索引
CON\$	在数据库中定义的约束（包括约束拥有者的信息）；通过 I_CON1 和 I_CON2 索引
CDEF\$	在 CON\$ 中的约束的定义；通过 I_CDEF1、 I_CDEF2 和 I_CDEF3 索引
CCOL\$	在其上定义过索引的列（包括在组合键中每个列的单独的项）；通过 I_CCOL1 索引
USERS	在数据库中定义的用户和角色；通过 I_USER1 索引



(续表)

对象	说明
TSQ\$	用户的表空间限额（包括为每个用户定义的每个表空间的限额）
C_OBJ#	包含 TAB\$、CLU\$、ICOL\$、IND\$和 COL\$的簇；通过 I_OBJ#索引
C_TS#	包含 FET\$、TSS\$和 FILE\$的簇；通过 I_TS#索引
C_USER#	包含 USER 和 TSQ\$的簇；通过 I_USER#索引
C_COBJ#	包含 CDED\$和 CCOL\$的簇；通过 I_COBJ#索引
对于所有数据字典段，很可能需要改变如下几项：	
C_TS#	如果数据库中的空闲空间很碎时
C_OBJ#	如果表中有许多索引和许多列时
CONS, C_COBJ#	如果大量地使用完整性约束时
C_USER#	
对于簇表，必须改变簇的存储设置，而不是表的存储设置。	

21.9.2 需要修改数据字典存储的错误

如果一个用户试图创建一个新的对象，该对象需要 Oracle 在数据字典中为其分配附加的盘区，而 Oracle 又不能分配盘区时，Oracle 就返回一个错误。下面的错误消息就说明这种问题：

ORA-1653 unable to extend table name by num in tablespace name

如果收到了这种错误消息，而试图改变（例如表或回滚段）的段并没有到达在其定义中所指定的极限，请检查包含它的定义的对象存储设置。

例如，如果试图在表上定义新的 PRIMARY KEY 约束时，收到了一条 ORA-1653，并且有足够的空间供 Oracle 为该键所必须创建的索引使用，就请检查是否不能给 CONS\$或 C_COBJ#分配盘区。为了做到这一点，查询 DBA_SEGMENT。如果不能分配附加的盘区，考虑改变 CONS\$或 C_COBJ#的存储参数，参见第 21 章的“例 7：显示不能分配附加盘区的段”。

21.10 显示有关模式对象的信息

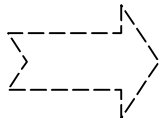
Oracle 提供了数据字典视图和 PL/SQL 包，以便于显示有关模式对象的信息。与特殊的模式对象相对应的视图和包会在本书的与那个对象相关的章节中描述。

本节描述普通的和应用到多种模式对象的视图和包。

21.10.1 使用 PL/SQL 包显示有关模式对象信息

这些 Oracle 所提供的 PL/SQL 包提供有关模式对象的信息：

包和过程/函数	说明
DBMS_EMTADATA.GET_DDL	用于获得关于模式对象的元数据（按用于创建该对象的 DDL 的格式）。
下面的包过程提供了有关模式对象的空间使用和空闲数据块的信息：	
DBMS_SPACE.UNUSED_SPACE	返回对象（表、索引或簇）中未使用空间的信息



(续表)

包和过程/函数	说明
DBMS_SPACE.FREE_BLOCK	返回对象（表、索引或簇）中空闲数据块的信息，对象的段的空闲空间由释放列表管理（段空间管理是 MANUAL（手动的））。
DBMS_SPACE.SPACE_USAGE	返回对象（表、索引或簇）空闲数据块的信息，对象的段的空闲空间管理是 AUTO（自动的）。

下面是使用上述包中某几个包的例子：

21.10.1.1 例 1：使用 DBMS_METADATA 包

DBMS_METADATA 包是获取模式对象完整定义的强有力工具。它能使我们一次就获得一个对象的全部属性。对象按 DDL 描述，可用于重新创建它。

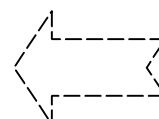
在此例中，GET_DDL 函数被用于从当前模式中提取所有表的 DDL，过滤出嵌套的表和溢出的段。SET_TRANSFORM_PARAM（其句柄值等于 DBMS_METADATA.SESSION_TRANSFORM，意思是“针对当前会话”）被用于指定在 SQL DDL 中不返回存储子句。之后，将会话层的传输参数设置成默认值。只要一设置，则传输参数保持有效，直到又重新设置成默认值。

```
EXECUTE DBMS_METADATA.SET_TRANSFORM_PARAM(
DBMS_METADATA.SESSION_TRANSFORM, 'STORAGE',false);
SELECT DBMS_METADATA.GET_DDL('TABLE',u.table_name)
FROM USER_ALL_TABLES u
WHERE u.nested='NO'
AND (u.iot_type is null or u.iot_type='IOT');
EXECUTE DBMS_METADATA.SET_TRANSFORM_PARAM(
DBMS_METADATA.SESSION_TRANSFORM,'DEFAULT';
```

21.10.1.2 例 2：使用 DBMS_SPACE.UNUSED_SPACE 包

下面的 SQL*Plus 例子使用 DBMS_SPACE 包来获得未使用空间信息。

```
SQL> VARIABLE total_blocks NUMBER
SQL> VARIABLE total_bytes NUMBER
SQL> VARIABLE unused_blocks NUMBER
SQL> VARIABLE unused_bytes NUMBER
SQL> VARIABLE lastextf NUMBER
SQL> VARIABLE last_extb NUMBER
SQL> exec DBMS_SPACE.UNUSED_SPACE('SCOTT', 'EMP', 'TABLE', :total_blocks, ->
:total_bytes,:unused_blocks, :unused_bytes, :lastextf, ->
:last_extb, :lastusedblock);
PL/SQL procedure successfully completed.
SQL> PRINT
```



```
TOTAL_BLOCKS
-----
5
TOTAL_BYTES
-----
10240
...
LASTUSEDBLOCK
-----
3
```

21.10.2 使用视图来显示有关模式对象的信息

下面的视图显示有关模式对象的信息：

视图	说明
DBA_OBJECTS	DBA 视图描述了数据库中的所有模式对象。ALL 视图描述了当前用户可访问的对象。USER 视图描述了当前用户拥有的对象。
ALL_OBJECTS	
USER_OBJECTS	
DBA_CATALOG	列出了数据库中所有表、视图、同义词和序列的名称、类型和所有者（USER 视图不显示所有者）。
ALL_CATALOG	
USER_CATALOG	
DBA_DEPENDENCIES	描述了过程、包、函数、包体和触发器之间的所有依赖性，包括没有任何数据库链接的视图间的依赖性。
ALL_DEPENDENCIES	
USER_DEPENDENCIES	
下面的视图包含数据库的段信息	
DBA_SEGMENTS	描述了分配给所有数据库段的存储，或分配给当前用户的段。
USER_SEGMENTS	
下面的视图包含数据库的范围信息	
DBA_EXTENTS	描述了数据库中组成所有段的盘区，或组成当前用户的段的盘区。
USER_EXTENTS	
DBA_FREE_SPACE	列出了所有表空间的空闲盘区，或者是当前用户所拥有的表空间的空闲盘区。
USER_FREE_SPACE	

下面是使用上述视图中某几个视图的例子：

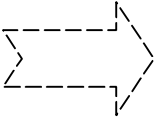
21.10.2.1 例 1：通过类型显示模式对象

下面的查询列出了发布查询的用户所拥有的全部对象：

```
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

```

21.10.2.2 例 2: 显示列信息

列信息，如名称、数据类型、长度、精度、范围和默认的数据值，可通过使用结尾带有 `_COLUMNS` 后缀的视图列出来。例如，下面的查询列出了 `emp` 和 `dept` 表的全部默认列值：

```

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 MGR
EMP HIREDATE SYSDATE
EMP SAL
EMP COMM
EMP DEPTNO

```

注意不是所有的列都有用户指定的默认值。在默认情况下，这些列自动取 `NULL` 值。

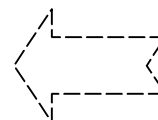
21.10.2.3 例 3: 显示视图和同义词的依赖性

当创建视图或同义词时，视图或同义词是基于其下层基础对象的。`ALL/USER/DBA_DEPENDENCIES` 数据字典视图揭示了视图的依赖性；`ALL/USER/DBA_SYNONYMS` 数据字典视图列出同义词的基础对象。例如，下面的查询列出了由用户 `jward` 所创建的同义词的基础对象：

```

SELECT TABLE_OWNER, TABLE_NAME, SYNONYM_NAME
FROM DBA_SYNONYMS
WHERE OWNER = 'JWARD';
TABLE_OWNER TABLE_NAME SYNONYM_NAME
-----

```



```
SCOTT DEPT DEPT
SCOTT EMP EMP
```

21.10.2.4 例 4: 显示普通段信息

下面的查询返回每个回滚段的名称，包含每个回滚段的表空间的名称、每个回滚段的大小：

```
SELECT SEGMENT_NAME, TABLESPACE_NAME, BYTES, BLOCKS, EXTENTS
FROM 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
```

21.10.2.5 例 5: 显示普通盘区的信息

数据库中有关当前已分配的盘区的普通信息存储在 DBA_EXTENTS 数据字典视图中。例如，下面的查询标识与回滚段相关的盘区和每个盘区的大小：

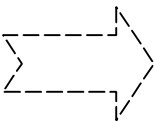
```
SELECT SEGMENT_NAME, BYTES, BLOCKS
FROM DBA_EXTENTS
WHERE SEGMENT_TYPE = 'ROLLBACK';
SEGMENT_NAME BYTES BLOCKS
-----
RS1 10240 5
RS1 10240 5
SYSTEM 51200 25
SYSTEM 51200 25
SYSTEM 51200 25
```

注意 rs1 回滚段由两个盘区组成，每个 10K，而 SYSTEM 回滚段由三个尺寸相等的盘区组成，每个 50K。

21.10.2.6 例 6: 显示数据库的空闲空间（盘区）

数据库中空闲盘区（指没有分配给任何段的盘区）存储在 DBA_FREE_SPACE 数据字典视图中。例如，下面的查询揭示了在每个表空间中可用于空闲盘区的可用的空闲空间大小。

```
SELECT TABLESPACE_NAME, FILE_ID, BYTES, BLOCKS
FROM DBA_FREE_SPACE;
TABLESPACE_NAME FILE_ID BYTES BLOCKS
-----
```



```
SYSTEM 1 8120320 3965
```

```
SYSTEM 1 10240 5
```

```
TS1 2 10432512 5094
```

21.10.2.7 例 7: 显示不能分配附加盘区的段

可用 DBA_FREE_SPACE，并与视图 DBA_SEGMENT，DBA_TABLES，DBA_CLUSTERS，DBA_INDEXES 和 DBA_ROLLBACK_SEGS 相结合，来确定是否是任何其他的段都不能仅仅为数据字典对象分配附加盘区。

由于下列原因，段不能分配到盘区：

- 包含段的表空间没有足够的空间分配下一个盘区。
- 段中盘区的数量已达到最大值，它被记录在数据字典中（在 SEG.MAX_EXTENTS）。
- 段中盘区的数量已达到数据块尺寸所允许的最大值，它与操作系统有关。

注意：尽管 MAXEXTENTS 的 STORAGE 子句的值可以是 UNLIMITED，但数据字典表不能使 MAXEXTENTS 大于所允许的数据块的最大值。所以数据字典表不能被转换成无限制的格式。

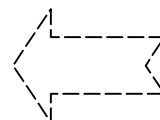
下面的查询返回满足上述标准的全部段的名称、拥有者和表空间：

```
SELECT a.SEGMENT_NAME, a.SEGMENT_TYPE, a.TABLESPACE_NAME, a.OWNER
FROM DBA_SEGMENTS a
WHERE a.NEXT_EXTENT >= (SELECT MAX(b.BYTES)
FROM DBA_FREE_SPACE b
WHERE b.TABLESPACE_NAME = a.TABLESPACE_NAME)
OR a.EXTENTS = a.MAX_EXTENTS
OR a.EXTENTS = 'data_block_size' ;
```

注意：当使用此查询时，用你的系统的数据块尺寸代替 *data_block_size*。

一旦识别出不能分配附加盘区的段，就可用下面两种方法中的一种解决问题，使用那种方法取决于形成原因：

- 如果表空间满了，就往表空间中加入数据文件。
- 如果段中包含很多盘区，又不能加大段的 MAXEXTENTS 值，则执行下列步骤：
 - ① 导出段中的数据
 - ② 删除并重新创建段，给段设置一个较大的 INITIAL 值，以便其不需分配太多的盘区
 - ③ 将数据再导入到段中。



第 22 章 检测和修复数据块损坏

本章解释使用 DBMS_REPAIR PL/SQL 包来修复数据库模式对象中的数据块损坏，包括如下内容：

- 用于修复数据块损坏的选项
- 关于 DBMS_REPAIR 包
- 使用 DBMS_REPAIR 包
- DBMS_REPAIR 的例子

注意：如果不熟悉 DBMS_REPAIR 包，建议在执行这个包中的修复过程时，与 Oracle 支持服务分析员一起工作。

22.1 用于修复数据块损坏的选项

Oracle 为检测和纠正数据块损坏提供了不同的方法。纠正的方法之一是：在检测到损坏之后，就删除并重建该对象。但这不总是可能的和理想的。如果数据块损坏局限于行的子集，则另一个方法是：选取除了损坏的行之外的所有行来重建表。

管理数据块损坏的另一个方法是：使用 DBMS_REPAIR 包。可以用 DBMS_REPAIR 来检测和修复表中和索引中的损坏的数据块。使用这个方法，可以确定损坏可能出现的位置，还可以在重建或修复它们时继续使用对象。

注意：任何包含数据丢失的损坏都需要分析和理解数据是如何填充进整个数据库系统中的。DBMS_REPAIR 不是魔杖——所以还必须决定，是否这个包所提供的修复方法适用于每个特殊的损坏问题。基于损坏的本质，是可能丢失数据或引入逻辑矛盾的。因此，必须权衡使用 DBMS_REPAIR 所带来的得与失的问题。

22.2 关于 DBMS_REPAIR 包

本节描述包含在这个包中的 DBMS_REPAIR 过程，并注明使用它们时的某些限制和约束。

22.2.1 DBMS_REPAIR 过程

下表列出了包含在 DBMS_REPAIR 包中的过程：

过程名	说明
CHECK_OBJECT	检测和报告表中或索引中的损坏。
FIX_CORRUPT_BLOCKS	将数据块（以前已被 CHECK_OBJECT 过程标记出来）标记成软件错误。
DUMP_ORPHAN_KEYS	报告指出损坏数据块中的行的索引项（在一个孤立键表中）。
REBUILD_FREELISTS	重建对象的空闲列表。
SEGMENT_FIX_STATUS	当段空间管理是 AUTO 时，提供确定位图项的损坏状态的能力。
SKIP_CORRUPT_BLOCKS	当使用它时，就在扫描表和索引期间，忽略被标记成损坏的数据块。当不使用它时，当遇到被标记成损坏的数据块时就得到错误 ORA-1578

(续表)

过程名	说明
ADMIN_TABLES	提供用于修复或隔离键表的管理函数（创建、删除、净化）。 注意：这些表总是被创建在 SYS 模式中。

这些过程将在第 22 章的“DBMS_REPAIR 的例子”中作进一步的描述，包含其使用例子。

22.2.2 限制和约束

DBMS_REPAIR 过程有如下限制：

- 支持具有 LOB 列、嵌入表和 varray 列的表，但是忽略行之外的列（out of line columns）。
- SKIP_CORRUPT_BLOCKS 和 REBUILD_FREELISTS 过程支持簇，但 CHECK_OBJECT 过程不支持簇。
- 不支持索引结构表和 LOB 索引。
- DUMP_ORPHAN_KEYS 过程不能用于位图索引或基于函数的索引。
- DUMP_ORPHAN_KEYS 过程最多处理 3950 字节长的键。

22.3 使用 DBMS_REPAIR 包

当考虑用 DBMS_REPAIR 来定位数据块损坏时，建议使用如下步骤：

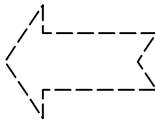
- 任务 1：检测和报告损坏
 - 任务 2：评估使用 DBMS_REPAIR 的损益
 - 任务 3：使对象可用
 - 任务 4：修复损坏并重建丢失的数据
- 在后续几节中讨论这些任务。

22.3.1 任务 1：检测和报告损坏

在使用 DBMS_REPAIR 之前的第一项任务，就是应该检测和报告损坏。报告不仅仅指出数据块出什么错了，还要指出相关的修复方针。为了检测损坏，除了 DBMS_REPAIR 之外，还有几个选择。表 22-1 描述了不同的检测方法学。

表 22-1 各种损坏检测方法的比较

检测方法	说明
DBMS_REPAIR	对一个指定的表、分区或索引执行数据块检查。用结果填充一个修复表。
DB_VERIFY	外部命令行工具，它在脱机数据库上执行数据块检查。
ANALYZE	与 VALIDATE STRUCTURE 选项一起使用，确认索引、表或簇的结构完整性；检查或确认表和索引是同步的。
DB_BLOCK_CHECKING	当初始参数 DB_BLOCK_CHECKING = TRUE 时被执行。在实际被标记成损坏之前识别损坏的数据块。当数据块被修改时执行检查。



22.3.1.1 DBMS_REPAIR: 使用 CHECK_OBJECT 和 ADMIN_TABLES 过程

CHECK_OBJECT 过程检查和报告指定对象的数据块损坏。与用于索引和表的 ANALYZE ... VALIDATE STRUCTURE 语句相同，数据块检查是用于索引和数据块的。

CHECK_OBJECT 不仅报告损坏，如果以后在该对象上运行 FIX_CORRUPT_BLOCKS 时，它还标识任何方位。通过将这些信息填充进修复表而使这些信息可用，必须先用 ADMIN_TABLES 过程创建修复表。

运行 CHECK_OBJECT 过程之后，在修复表上运行一个简单查询，就显示对象的损坏和修复方针。用这些信息，就可以知道如何最好地解决所报告的问题。

22.3.1.2 DB_VERIFY: 执行脱机数据库检查

通常，当遇到数据损坏问题时，用 DB_VERIFY 作为脱机诊断工具。

22.3.1.3 ANALYZE: 损坏报告

ANALYZE TABLE ... VALIDATE STRUCTURE 语句校验被分析对象的结构。如果 Oracle 成功地校验了该结构，则返回一条确认该校验的消息。如果 Oracle 遇到了对象结构中的损坏，则返回一条错误消息。这时，就应删除并重建该对象。

22.3.1.4 DB_BLOCK_CHECKING (数据块检查初始参数)

可以使用 DB_BLOCK_CHECKING 初始参数（默认值是 FALSE）来为实例设置数据块检查任务。无论何时修改数据和索引时，它就检查数据和索引数据块。DB_BLOCK_CHECKING 是个动态参数，可用 ALTER SYSTEM SET 语句进行修改。对于系统表空间，总是启用数据块检查。

22.3.2 任务 2: 评估使用 DBMS_REPAIR 的损益

在使用 DBMS_REPAIR 之前，必须权衡使用它的好处及其不利因素。还应该测试一下其他可用于处理损坏对象的方法。

第一步是回答如下问题：

①损坏的盘区是什么？

为了确定是否存在损坏和修复动作，就执行 CHECK_OBJECT 过程，并查询修复表。

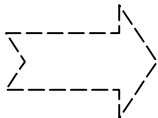
②其他可用于处理数据块损坏的方法是什么？考虑：

- 假设可从另一个来源获得数据，就删除、重建并重新填充该对象。
- 对损坏的表发布 CREATE TABLE ... AS SELECT 语句以便重建一个新表。
- 通过从选取 (select) 语句中排除损坏的行来忽略损坏。
- 执行媒体恢复。

③当使用 DBMS_REPAIR 来使一个对象可用时，会引入什么逻辑矛盾或副作用？可以处理这些问题吗？需要花多大精力？

可能不能访问被标记成损坏的数据块中的行。当然尽管还存在可以被有效访问的行，但数据块也可以被标记成损坏。

当数据块被标记成损坏时也可能破坏参照完整性约束。的确如此的话，就应停用和



重新启用该约束；任何矛盾都被报告。确定所有问题之后，就可能成功地重新启用该约束。

当在表上定义过触发器时会产生逻辑矛盾。例如，如果重新插入行，则应不应该启动插入（insert）触发器呢？只有在理解了触发器及其在你的安装中的使用时，才能处理这些问题。

空闲列表数据块可能变成不可访问了。如果损坏数据块是在空闲列表的头部或尾部，则空间管理会重新初始化该空闲列表。这就可能存在应该放入空闲列表中但又没有放入的数据块。运行 REBUILD_FREELISTS 过程可以处理这个问题。

索引和表不同步了。可以处理这个问题：首先执行 DUMP_ORPHAN_KEYS 过程（从键中获得信息，这在重建被损坏的数据时可能有用），然后发布 ALTER INDEX ... REBUILD ONLINE 语句以使表及其索引继续同步。

④如果修复中丢失了数据，这些数据还能被检索吗？

当数据块被标记成损坏时仍可以从索引中检索数据。DUMP_ORPHAN_KEYS 过程能有助于检索该信息。当然，用这种方法检索数据取决于索引和表之间数据冗余的数量。

22.3.3 任务 3：使对象可用

在这个任务中，通过在扫描表和索引期间忽略损坏，DBMS_REPAIR 使对象可用。

22.3.3.1 损坏修复：使用 FIX_CORRUPT_BLOCKS 和 SKIP_CORRUPT_BLOCKS 过程

通过建立一个环境，跳过 DBMS_REPAIR 的修复能力范围之外的损坏，使损坏对象可用。

如果该损坏包括了数据的丢失，如数据块中的坏行，所有这样的数据块都被 FIX_CORRUPT_BLOCKS 过程标记成损坏。然后，运行 SKIP_CORRUPT_BLOCKS 过程，它跳过对象中被标记成损坏的数据块。当设置跳过之后，表和索引扫描就跳过所有被标记成损坏的数据块。这适用于媒体和软件损坏数据块两者。

22.3.3.2 跳过损坏数据块时的含意

如果索引和表不同步了，那么，当一个查询仅仅探查索引，而下一个查询探查索引和表两者时，SET TRANSACTION READ ONLY 事务就可能有矛盾。如果表数据块被标记成损坏，那么，这两个查询就会返回不同的结果，这就破坏了只读事务的规则。处理这个问题的一个方法是，在 SET TRANSACTION READ ONLY 事务中不跳过损坏数据块。

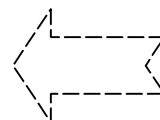
当选取被链接在一起的行时也会发生相同的问题。本质上说，对相同行的查询既可以不可以访问损坏数据块，因此产生不同的结果。

22.3.4 任务 4：修复损坏和重建丢失的数据

使对象可用之后，就可以进行如下修复活动。

22.3.4.1 用 DUMP_ORPHAN_KEYS 过程恢复数据

DUMP_ORPHAN_KEYS 过程报告指向损坏数据块中的行的索引项。所有这些索引项



都被插入到一个孤立键表中，该表存储了损坏数据块的键和行标识。

检索出了索引项信息之后，就可以用 ALTER INDEX ... REBUILD ONLINE 语句重建该索引。

22.3.4.2 用 REBUILD_FREELISTS 过程修复空闲列表

如果是用空闲列表 (SEGMENT SPACE MANAGEMENT MANUAL) 来管理段中的空闲空间时，就使用该过程。

当在空闲列表的头部或尾部上找到了被标记成“损坏”的数据块时，将重新初始化空闲列表，并返回一个错误。尽管这从空闲列表中清除了讨厌的数据块，但它使得空闲列表不能访问紧跟损坏数据块之后的所有数据块。

可以用 REBUILD_FREELISTS 过程来重新初始化空闲列表。对象被扫描，如果它适合作为空闲列表上的一个数据块，则将其增加到主空闲列表中。用等分数据块的方式来管理空闲列表组，一次一个数据块。在对象中，任何被标记成“损坏”的数据块，在重建期间都被忽略。

22.3.4.3 用 SEGMENT_FIX_STATUS 过程修理段位图

如果用位图 (SEGMENT SPACE MANAGEMENT AUTO) 来管理段中的空闲空间时，就使用该过程。

这个过程或者基于对应的数据块的当前内容重新计算位图的状态，或者指定将位图项设置成特殊的值。通常，状态被正确地重新计算而不需要强行设置。

22.4 DBMS_REPAIR 的例子

本节用例子解释如何使用 DBMS_REPAIR 过程：

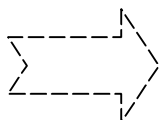
- 用 ADMIN_TABLES 建立修复表或孤立键表
- 用 CHECK_OBJECT 过程检测损坏
- 用 FIX_CORRUPT_BLOCKS 过程修理损坏数据块
- 用 DUMP_ORPHAN_KEYS 查找指向损坏数据块的索引项
- 用 REBUILD_FREELISTS 过程重建空闲列表
- 用 SKIP_CORRUPT_BLOCKS 启用或停用跳过损坏数据块

22.4.1 用 ADMIN_TABLES 建立修复表或孤立键表

修复表提供有关 CHECK_OBJECT 过程发现了什么损坏，以及 FIX_CORRUPT_BLOCKS 过程如何修理这些损坏的信息。进一步，它被用于驱动 FIX_CORRUPT_BLOCKS 过程的执行。

当执行 DUMP_ORPHAN_KEYS 过程，并且找到了指向损坏行的索引项时，就使用孤立键表。DUMP_ORPHAN_KEYS 过程用登记其活动，并以可用的方式提供索引信息，来填充孤立键表。

ADMIN_TABLE 过程用于创建、净化或删除修复表或孤立键表。



22.4.1.1 创建修复表

如下例子为 users 表空间创建修复表。

```
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键表，还创建一个视图，它排除属于不再存在的对象的任何行。视图的名字对应于修复表或孤立键表，但加上前缀 DBA_（如 DBA_REPAIR_TABLE 或 DBA_ORPHAN_KEY_TABLE）。

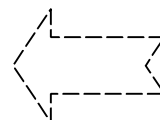
如下查询描述前面例子中创建的修复表。

```
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
```

22.4.1.2 创建孤立键表

本例说明为 users 表空间创建孤立键表。

```
BEGIN
DBMS_REPAIR.ADMIN_TABLES (
TABLE_NAME => 'ORPHAN_KEY_TABLE',
```



```
TABLE_TYPE => dbms_repair.orphan_table,
ACTION => dbms_repair.create_action,
TABLESPACE => 'USERS');
END;
/
```

如下查询描述孤立键表:

```
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
```

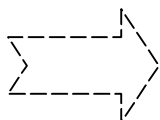
22.4.2 用 CHECK_OBJECT 过程检测损坏

CHECK_OBJECT 过程检查指定的对象,并用有关损坏和修复方针的信息填充修复表。当打算检查一个对象的某个部分时,可以选择性地指定范围、分区名或子分区名。

校验表示检查对象中的已前还未被标记成损坏的所有数据块。对于每个数据块,就检查事务和数据层部分的自身的一致性。在执行 CHECK_OBJECT 期间,如果遇到一个具有损坏的缓冲区存储器头部的数据块,那就跳过该数据块。

这是一个用于 scott.dept 表的执行 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 输出如下行，指出了损坏：

```
number corrupt: 1
```

查询修复表，会产生描述损坏和建议修复动作的信息。

```
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
```

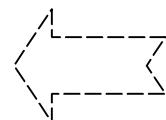
此时，已损坏的数据块还没有被标记成损坏，所以这是一个提取任何有价值数据的机会。当数据块被标记成损坏之后，就必须跳过整个数据块了。

22.4.3 用 FIX_CORRUPT_BLOCKS 过程修理损坏数据块

基于前面用 CHECK_OBJECT 过程产生的修复表中的信息，使用 FIX_CORRUPT_BLOCK 过程来修理在指定对象中的、损坏的数据块。在对数据块做任何修理之前，先检查该数据块以保证该数据块仍然是损坏的。通过将数据块标记成软件错误而修复该损坏的数据块。执行修复后，会更新修复表中的相关行的修理时间戳。

本例修理用 CHECK_OBJECT 过程报告的、在 scott.dept 表中的损坏数据块。

```
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*Plus 输出如下行:

```
num fix: 1
```

如下查询确认已做了修复。

```
SELECT OBJECT_NAME, BLOCK_ID, MARKED_CORRUPT
```

```
FROM REPAIR_TABLE;
```

```
OBJECT_NAME BLOCK_ID MARKED_COR
```

```
-----
```

```
DEPT 3 TRUE
```

22.4.4 用 DUMP_ORPHAN_KEYS 查找指向损坏数据块的索引项

DUMP_ORPHAN_KEYS 过程报告指向损坏数据块中的行的索引项。对于每个遇到的这样的索引项,就在指定的孤立键表中插入一行。必须在这之前创建好孤立键表。

这个信息可能有利于重建表中丢失的行,以及有利于诊断的目的。

注意: 应该对每个与修复表中指出的表相关的索引都运行该过程。

本例中, pk_dept 是 scott.dept 表上的一个索引。扫描该索引以确定是否存在任何指向损坏数据块中的行的索引项。

```
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));
```

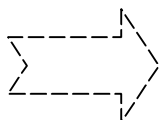
```
END;
```

```
/
```

下面是输出行,指出存在 3 个孤立键:

```
orphan key count: 3
```

孤立键表中的索引项表示应该重建该索引。这保证表探查和索引探查会返回同样的结果集。



22.4.5 用 REBUILD_FREELISTS 过程重建空闲列表

REBUILD_FREELISTS 过程为指定的对象重建空闲列表。所有的空闲数据块都被放在主空闲列表中。所有其他的空闲表都被清零。如果对象有多个空闲列表组，则空闲数据块被划分到所有空闲列表，以轮回方式分配给不同的组。

本例为表 scott.dept 重建空闲列表。

```
BEGIN
DBMS_REPAIR.REBUILD_FREELISTS (
  SCHEMA_NAME => 'SCOTT',
  OBJECT_NAME => 'DEPT',
  OBJECT_TYPE => dbms_repair.table_object);
END;
/
```

22.4.6 用 SKIP_CORRUPT_BLOCKS 启用或停用跳过损坏数据块

SKIP_CORRUPT_BLOCKS 过程启用或停用是否在对指定对象的表和索引进行扫描期间跳过损坏的数据块。当对象是表时，就跳过表及其索引。当对象是簇时，就跳过簇中所有的表及其他各自的索引。

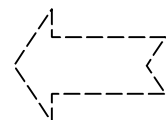
如下例子为 scott.dept 表启用跳过软件错误数据块：

```
BEGIN
DBMS_REPAIR.SKIP_CORRUPT_BLOCKS (
  SCHEMA_NAME => 'SCOTT',
  OBJECT_NAME => 'DEPT',
  OBJECT_TYPE => dbms_repair.table_object,
  FLAGS => dbms_repair.skip_flag);
END;
/
```

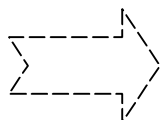
用 DBA_TABLES 视图查询 scott 的表，显示出已经为表 scott_dept 启用了 SKIP_CORRUPT。

```
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 4 部分 数据库的安全

第 4 部分着重介绍影响数据库安全的用户和权限管理问题，包括如下章节：

- 第 23 章 建立安全策略
- 第 24 章 管理用户和资源
- 第 25 章 管理用户的权限和角色
- 第 26 章 审计数据库的使用

第 23 章 建立安全策略

本章为数据库操作提供了用于开发安全策略的指南，包含如下内容：

- 系统安全的策略
- 数据安全的策略
- 用户安全的策略
- 口令管理的策略
- 审计的策略
- 安全检查的清单

23.1 系统安全的策略

本节描述有关系统安全策略的几个方面，包括如下内容：

- 数据库用户的管理
- 用户的鉴别
- 操作系统的安全

每个数据库系统都有一个或几个管理员，他（们）负责维护所有的安全策略方面的问题：即安全管理员。如果数据库系统很小，数据库管理员也就担当起安全管理员的责任。但是，如果数据库系统很大，通常就要指定一个人或一群人专门担当安全管理员的责任。

在确定谁将管理系统的安全之后，就要为每个数据库开发安全策略。数据库的安全策略应包含几个子策略，这个问题将在下面几节中说明。

23.1.1 数据库用户的管理

数据库用户是访问 Oracle 数据库中的信息的途径。因此，就应该对数据库用户进行严格的安全管理。根据数据库系统的大小和管理数据库用户所需的工作量，安全管理员是唯一具有创建、更改或删除用户的权限的用户。另一方面，可能有一些管理员也有权管理数据库用户。不管怎么说，只有被信赖的人才应该具有这种强大的管理数据库用户的权限。

23.1.2 用户的鉴别

Oracle 使用数据库口令、宿主操作系统、网络服务或安全套接字协议层（Secure Sockets Layer, SSL）来鉴别（核实合法用户）数据库用户。

注意：要想用网络鉴别服务或 SSL 来鉴别数据库用户，就要先安装 Oracle 高级安全组件（Oracle Advanced Security）。

用户鉴别和如何指定鉴别会在第 24 章的“用户鉴别的方法”中讨论。

23.1.3 操作系统的安全

如果可以的话，对于执行 Oracle 和任何数据库应用的操作系统环境来说，还应该考虑

如下安全问题：

- 数据库管理员必须具有操作系统权限，以便创建和删除文件。
- 一般数据库用户不应该具有操作系统权限，以便创建或删除涉及数据库的文件。
- 如果操作系统会识别用户的数据库角色，那么，安全管理员就必须有操作系统权限，以便修改操作系统帐户的安全域。

23.2 数据安全的策略

数据安全包括在对象层上控制访问和使用数据库的机制。你的数据安全策略应确定，哪些用户能访问特殊的模式对象、允许每个用户在对象上所做的特殊的动作类型。例如：使用 emp 表时，用户 scot 只能发布 SELECT 和 INSERT 语句，而不能发布 DELETE 语句。数据安全还应该定义每个模式对象审计的动作，如果有的话。

数据安全策略主要是由你想在数据库中为数据建立的安全的层次来确定的。例如：当你想允许任何用户创建任何模式对象，或者将针对他们的访问权限授予系统中的任何其他用户时，数据库中的数据安全少点也是可以接受的。当然，当你只想让数据库管理员或安全管理员才能创建对象，并将访问该对象的权限授予角色和用户时，就很有必要对数据安全进行很好地控制了。

总体的数据安全应该基于数据的敏感性。如果信息是不敏感的，那么数据的安全策略可以松一些；然而，如果数据是敏感的，那么，就应该开发出安全策略，以便对访问对象进行严密的控制。

实现数据安全的方法包括系统权限和对象权限、或通过角色实现。角色是可以授予用户的，组合在一起的一组权限。在第 25 章“管理用户的权限和角色”中讨论有关权限和角色的问题。

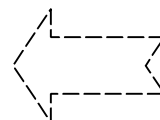
视图也能实现数据安全，因为视图的定义能限制对表中的数据访问。它们可以排出包含敏感数据的列。在第 20 章“管理视图、序列和同义词”中讨论有关视图的问题。

实现数据安全的其他方法是，成熟的访问控制和使用应用上下文。成熟的访问控制是 Oracle 的特性，它允许实现带有函数的安全策略，将安全策略与表和视图相关联。实际上，安全策略函数会产生一个 WHERE 条件，附加到 SQL 语句后，从而限制用户访问表或视图中的数据行。应用上下文是一个安全数据高速缓存，用于存储做出访问控制决定的相关的信息。

23.3 用户安全的策略

本节描述用户安全策略的几个方面，包括如下内容：

- 普通用户的安全
- 终端用户的安全
- 管理员的安全
- 应用开发者的安全
- 应用管理员的安全



23.3.1 普通用户的安全

对于所有类型的数据库用户，都要考虑如下的普通用户的安全问题：

- 口令的安全
- 权限管理

23.3.1.1 口令的安全

如果用户鉴别是由数据库来管理的，那么，安全管理员就应该开发出一个口令安全策略，以维护数据库访问的安全。例如：要求数据库用户定期地改变它们的口令。当然了，当它们的口令被其他人知道了时，就要强制用户修改口令，以便减少非授权的用户访问数据库。

为了更好的保证口令的机密性，对于客户/服务器和服务器/服务器连接来说，可以将 Oracle 配置成使用加密后的口令。

注意：在客户/服务器和服务器/服务器连接中，强烈建议将 Oracle 配置成加密口令。否则，恶意的用户“侦听”网络，攫取未加密的口令，然后使用该口令按另外的用户身份连接到数据库，从而“假装”那个用户。

通过设置下面的值，可以要求总是将用于校验一个连接的口令进行加密。

- 在客户机上，将 ORA_ENCRYPT_LOGIN 环境变量设置成 TURE。
- 将 DBLINK_ENCRYPT_LOGIN 服务器初始化参数设置成 TURE。

如果在客户和服务器两者上都进行了设置，那么，口令就不会在网络上“清晰”地发送了，而是用修改后的 DES（Data Encryption Standard；数据加密标准）算法进行了加密。

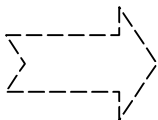
DBLINK_ENCRYPT_LOGIN 初始化参数用于两个 Oracle 服务器之间的连接（例如：在执行分布式查询时）。如果正在同客户进行连接，那么，Oracle 就会去检查 ORA_ENCRYPT_LOGIN 环境变量了。

无论什么时候，尝试使用口令来连接到一个服务器，Oracle 就会在把口令发送给服务器之前，先对其进行加密。如果连接失败了并且启用了审计，就会在审计日志中记录该失败。Oracle 然后检查相应的 DBLINK_ENCRYPT_LOGIN 或者 ORA_ENCRYPT_LOGIN 的值。如果它们被设置成 FALSE，那么，Oracle 就会尝试使用未加密形式的口令，再进行连接。如果连接成功，该连接就将替代审计日志中的前面的失败记录，该连接继续下去。为了避免恶意的用户强迫 Oracle 使用一个未加密形式的口令重新尝试连接，就必须将相应的值设置成 TURE。

23.3.1.2 权限管理

安全管理员应该考虑涉及所有类型用户的权限管理问题。例如：在一个有许多用户名的数据库中，使用角色（可以授予用户或其他角色的、相关权限的命名的集合）来管理用户可用的权限是非常有益的。否则，如果数据库中只有少数用户名，就将权限明确地赋予用户，避免使用角色，反而更容易。

管理具有很多用户、应用或对象的数据库的安全管理员，应该利用角色提供的益处。角色在很大程度上简化了在复杂环境中的权限管理任务。



23.3.2 终端用户的安全

安全管理员必须定义终端用户的安全策略。如果一个数据库有许多用户，安全管理员可以决定将哪些用户的组分类成用户组，然后为这些组创建用户角色。安全管理员可以给每个用户角色赋予必要的权限或应用角色，再将用户角色赋予给这些用户。对于例外情况，安全管理员还必须确定，必须将什么权限明确地授予各个用户。

23.3.2.1 使用角色实现终端用户的权限管理

角色是授予和管理不同的数据库用户组所需的公共权限的最简单的方法。

考虑一种情况，公司财务部门的每个用户都需要权限，去运行 `accts_receivable` 和 `accts_payable` 数据库应用。角色就与这两个应用相关了，它包含执行这些应用所需的对象权限。

由数据库管理员或安全管理员执行的如下动作，处理这个简单的安全情况：

- ①创建 `accountant` 角色。
- ②将 `accts_receivable` 和 `accts_payable` 数据库应用的角色，授予 `accountant` 角色。
- ③将 `accountant` 角色授予财务部门的每个用户。

其安全模型如图 23-1 所示。

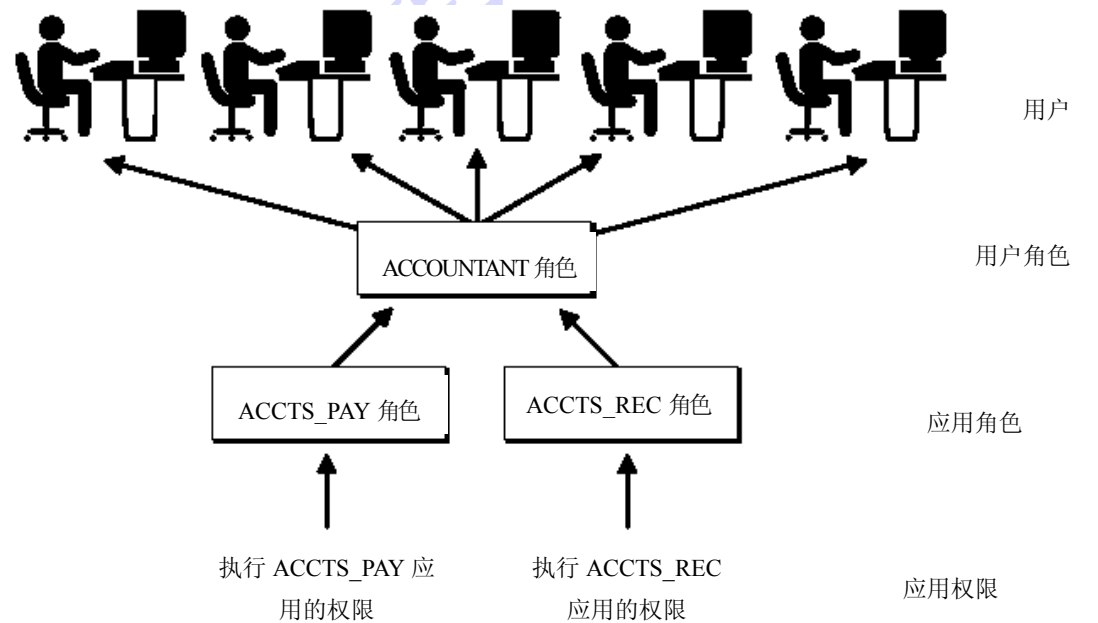
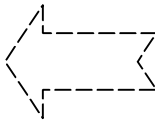


图 23-1 用户角色

这个设计处理下面可能的情况：

- 如果会计师后来需要一个新的数据库应用的角色，就可以将那个应用的角色授予 `accountant` 角色，而在财务部门的所有用户，都将自动接受与该新的数据库应用相关联的权限。而不需要将该应用的角色授予需要使用该应用的各个用户。
- 同样地，如果财务部门不再需要某个特定的应用，就可以从 `accountant` 角色中删除



该应用的角色。

- 如果 `accts_receivable` 或 `accts_payable` 所需要的权限改变了，则可以给该应用的角色授予新权限，或者从该应用的角色中删除权限。`accountant` 角色的安全域和被授予 `accountant` 角色的所有用户，都自动地反映出权限的改变。

在所有可能的情况下利用角色，都使得对终端用户的权限管理变得有效和简单。

23.3.2.2 使用目录服务实现终端用户的权限管理

通过 Oracle 高级安全 (Oracle Advanced Security) 中的企业用户和企业角色特性，还可以在目录服务中，集中管理用户和它们的权限。

23.3.3 管理员的安全

安全管理员应该有一个处理数据库管理员的安全的策略。例如：当数据库很大并且有几种类型的数据库管理员时，安全管理员就应该将相关的管理权限划分成几个管理角色。然后将这些管理角色授予适当的管理员用户。相反，当数据库很小而且只有几个管理员时，只创建一个管理角色并把这个管理角色授予所有管理员，可能就更方便些了。

23.3.3.1 针对按 SYS 和 SYSTEM 进行的连接的保护

创建数据库之后，应立即改变 `SYS` 和 `SYSTEM` 管理用户名的口令，以避免未授权的访问数据库。

按 `SYS` 或 `SYSTEM` 进行的连接，给用户提供了一个强有力的权限，使其能有许多方法修改数据库。按 `SYS` 进行的连接允许用户更改数据字典表。与这些用户名相关联的权限非常敏感，应该只有数据库管理员才可以使用。

如果已经安装了可以创建其他管理用户名的选项，则这些被创建的用户帐号最初是被锁定的。要想将这些帐号解锁，就使用 `ALTER USER` 语句。也可以用 `ALTER USER` 语句来改变与这些帐号相关联的口令。

可以使用过程来改变这些帐号的口令，详见第 24 章的“更改用户”。

23.3.3.2 针对管理员的连接的保护

只有数据库管理员才应该具有连接到数据库进行管理的权限的能力。例如：

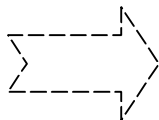
```
CONNECT username/password AS SYSDBA/SYSOPER
```

按 `SYSOPER` 进行的连接，给用户提供了执行基本操作任务（例如 `STARTUP`，`SHUTDOWN` 和恢复操作）的能力。按 `SYSDBA` 进行的连接，又给用户加上了无限的权限，使其能对数据库或数据库中的对象做任何操作（包括 `CREATE`，`DROP`，`DELETE`）。按 `SYSDBA` 进行的连接，将用户放入 `SYS` 模式中，使其能改变数据字典表。

23.3.3.3 使用针对管理员权限管理的角色

为了限制强大的系统权限和管理数据库的人员所需的角色，最简单的方法就是使用角色。

考虑一种情况，在一个大型安装中，几个数据库管理员分担数据库管理员的责任，而



每一个都负责如下特定的数据库管理工作：

- 对象的创建和维护
- 数据库的调整和性能
- 创建新用户，并将角色和权限授予数据库用户
- 常规数据库操作（例如：STARTUP，SHUTDOWN，备份和恢复操作）
- 突发情况，如数据库恢复

当新的、业务不熟练的数据库管理员在试验管理数据库时，也需要限制其能力。

在这种情况下，安全管理员应该按如下方法，构造用于管理人员的安全策略：

①定义 6 个角色，以便包括不同的、完成每种类型工作所需的权限（例如：dba_objects，dba_tune，dba_security，dab_maintain，dba_recov，dba_new）。

②给每个角色授予合适的权限。

③将相应的角色授予每种类型的数据库管理员。

这个方案按如下方式，减少了进一步的问题的可能性：

- 如果一个数据库管理员的工作种类改变了，包括了更多的责任，就可以给那个数据库管理员授权与新的责任相对应的其他管理角色。
- 如果一个数据库管理员的工作种类改变了，包括了较少的责任，就可以删除那个数据库管理员的相应的管理角色。
- 数据字典中已经存储了有关每个角色和每个用户的信息，可用这些信息来揭示每个管理员的任务。

23.3.4 应用开发者的安全

安全管理员必须为使用数据库的应用开发者定义一个特殊的安全策略。安全管理员应该给应用开发者授予创建所需对象的权限。或者，相反地，只将创建对象的权限授予数据库管理员，然后他再从开发者那里接受创建对象的请求。

23.3.4.1 应用开发者及其权限

数据库开发者是唯一的需要特殊的权限组合，以便完成他们的工作的数据库用户。不同于终端用户，开发者需要系统权限，例如 CREATE TABLE，CREATE PROCEDURE，等等。但是，只将特殊的系统权限授予给开发者，以限制他们在数据库中的总体能力。

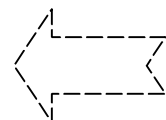
23.3.4.2 应用开发者的环境：测试数据库和产品数据库

在很多情况下，应用开发仅限于测试数据库，而不允许针对产品数据库。这种限制确保了应用开发者不与终端用户竞争数据库资源，而且这也不会对产品数据库带来有害的影响。

当一个应用已经全部开发和测试完成之后，就允许它访问产品数据库了，就可以让合适的产品数据库的终端用户使用了。

23.3.4.3 自由的还是受控的应用开发

数据库管理员通过定义如下选项，来决定将哪些权限授予应用开发者：



自由开发 允许应用开发者创建新的模式对象，包括表、索引、过程、包等等。该选项允许应用开发者开发应用，而不依赖于其他对象。

受控开发 不允许应用开发者创建新的模式对象。所有所需的表、索引、过程等等是由应用开发者申请，再由数据库管理员创建。该选项允许数据库管理员完全地控制数据库空间的使用，以及访问数据库中的信息的途径。

尽管某些数据库系统只使用其中之一，但其他系统可能把他们混合在一起。例如：允许应用开发者创建新的存储过程和包，但不允许创建表和索引。关于这个问题，安全管理员应基于如下方面来做决定：

- 希望对数据库的使用空间所进行的控制。
- 希望对模式对象的访问途径所进行的控制。
- 用于开发应用的数据库——如果一个测试数据库正被用于应用开发，则自由的开发策略可能更好一些。

23.3.4.4 应用开发者的角色和权限

为了管理一般应用开发者所需的权限，安全管理员可以创建角色。例如：一个名字为 APPLICATION_DEVELOPER 的一般角色，可以包括 CREATE TABLE, CREATE VIEW 和 CREATE PROCEDURE 系统权限。当为应用开发者定义角色时，应考虑如下方面：

- 通常将 CREATE 系统权限授予应用开发者，以便它们可以创建它们自己的对象。但是，CREATE ANY 系统权限，会允许一个用户在任何用户的模式中创建对象，因此，通常不将该权限授予开发者。这就将新对象的创建限制到了开发者的用户帐户。
- 很少将对象权限授予应用开发者所使用的角色。这是因为通过角色授予对象权限，常常会限制他们在创建其他对象（主要是视图和存储过程）时的可用性。允许应用开发者为了开发的目的，去创建它们自己的对象，这种方法是很实用的。

23.3.4.5 对应用开发者的空间限制

当授予应用开发者权限，使之作为开发过程的一部分而创建对象时，安全管理员必须维护一些界限，即每个应用开发者能使用什么样的、多大的数据库空间。例如：作为一个安全管理员，就应该特别地为每个应用开发者设置或者限制如下的界限。

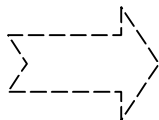
- 开发者能创建表和索引的表空间。
- 开发者可以访问的每个表空间的限额。

可以通过更改开发者的安全域，来设置上述两种界限。详见第 24 章的“更改用户”中的讨论。

23.3.5 应用管理员的安全

在具有许多数据库应用的大型数据库系统中，就要考虑分配应用管理员。一个应用管理员负责下面几种类型的任务：

- 为应用创建角色和管理每个应用角色的权限。
- 创建和管理数据库应用所使用的对象。



- 如果必要的话，维护和更新应用代码和 Oracle 的过程及包。

通常，应用管理员也是设计应用的应用开发者。但是，应用管理员也可能是熟悉数据库操作的任何人。

23.4 口令管理的策略

依赖于口令的数据库安全系统，需要口令在任何时候都是保密的。但是，口令是容易受到偷窃、伪造和滥用。为了更好的控制数据库的安全，Oracle 的口令管理的策略是由 DBA 和安全员，通过用户的配置文件来控制的。

使用 CREATE PROFILE 语句，来创建用户配置文件。用 CREATE USER 和 ALTER USER 语句，把用户配置文件分配给用户。本节不详细讨论创建和更改数据库用户。本节所关心的是使用 CREATE PROFILE（或 ALTER PROFILE）语句来设定口令参数。

本节包括如下与口令管理相关的内容：

- 口令的期限和到期
- 口令的历史
- 口令的复杂性校验

23.4.1 账户加锁

当一个特殊的用户超过了指定的失败的登录次数时，服务器就会自动地锁定那个用户帐号。使用 CREATE PROFILE 语句，来指定允许的失败的登录次数。还可以指定帐户保持锁定的时间。

在如下例子中，用户 ashwini 的失败的登录次数最多为 4，帐户保持锁定的时间为 30 天。超过 30 天这段时间之后，帐户自动解锁。

```
CREATE PROFILE prof LIMIT
FAILED_LOGIN_ATTEMPTS 4
PASSWORD_LOCK_TIME 30;
ALTER USER ashwini PROFILE prof;
```

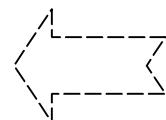
如果没有指定帐户解锁的时间间隔，PASSWORD_LOCK_TIME 就采用默认的用户配置文件中指定的值。如果将 PASSWORD_LOCK_TIME 指定为 UNLIMITED，就必须使用 ALTER USER 语句来明确地将帐户解锁。例如：假设给 ashwini 指定了 PASSWORD_LOCK_TIME 为 UNLIMITED，那么，就必须用如下语句来给该帐户解锁：

```
ALTER USER ashwini ACCOUNT UNLOCK;
```

在用户成功地登录到帐户之后，那个用户的不成功的登录次数（如果有的话），就被重新设置成 0。

安全员也能明确地锁定用户帐号。当发生这种情况时，该帐号就不能被自动解锁，只有该安全员才可以将该帐号解锁。CREATE USER 或者 ALTER USER 语句被用于明确地加锁或解锁帐号。例如：如下语句锁定用户帐号 susan：

```
ALTER USER susan ACCOUNT LOCK;
```



23.4.2 口令期限和到期

用 CREATE PROFILE 语句指定口令的最长生存期。当指定的时间期限到了，口令也就到期了，用户或 DBA 就必须改变该口令了。如下语句为用户 ashwini 创建和分配一个配置文件，其 PASSWORD_LIFE_TIME 子句指定，ashwini 在口令期满之前的 90 天内使用同样的口令。

```
CREATE PROFILE prof LIMIT
FAILED_LOGIN_ATTEMPTS 4
PASSWORD_LOCK_TIME 30
PASSWORD_LIFE_TIME 90;
ALTER USER ashwini PROFILE prof;
```

也可以为到期的口令指定延长期。在用户的口令已经到期之后，第一次登录数据库帐户时，用户输入延长期。在延长期期间，在用户每次登录他们的帐户时，都会出现一个警告消息，警告消息一直持续出现，直到延长期期满。在延长期期间，用户必须改变该口令。如果在延长期期间没有改变该口令，从那以后，用户每次访问他们的帐户时，都被提示输入新的口令。直到提供了一个新的口令，才允许访问该帐户。

图 23-2 表示了口令生存期和口令延长期的时间表。

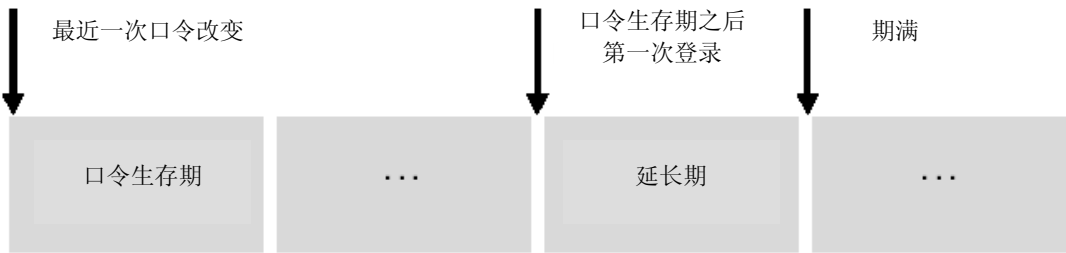
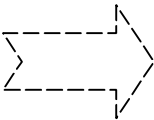


图 23-2 口令生存期和口令延长期的时间表

在如下例子中，分配给 ashwini 的用户配置文件中包括延长期的说明：PASSWORD_GRACE_TIME = 3。90 天（可能是第 90 天之后的任何一天，即，第 70 天、第 100 天，或其他某一天）之后，当 ashwini 第一次试图登录该数据库时，她就会收到了一条警告消息，说她的口令还有三天就将过期了。如果 3 天过后，她还没有改变她的口令，该口令就过期了。从那以后，只要她打算登录时，都会收到一条提示消息，要她改变她口令。直到她改变了她的口令，她才能登录。

```
CREATE PROFILE prof LIMIT
FAILED_LOGIN_ATTEMPTS 4
PASSWORD_LOCK_TIME 30
PASSWORD_LIFE_TIME 90
PASSWORD_GRACE_TIME 3;
ALTER USER ashwini PROFILE prof;
```

Oracle 提供了一个明确地将口令过期的方法。CREATE USER 和 ALTER USER 语句提



供了这种功能。如下语句用一个过期的口令创建一个用户。这种设置会强迫该用户在登录该数据库之前先改变该口令。

```
CREATE USER jbrown
IDENTIFIED BY zX83yT
...
PASSWORD EXPIRE;
```

23.4.3 口令的历史

为了指定一个用户不能重用口令的时间间隔,就要使用 CREATE PROFILE 语句。在如下语句中,定义了一个用户配置文件,该文件中的 PASSWORD_REUSE_TIME 子句指定了用户在 60 天内不能重用该口令。

```
CREATE PROFILE prof LIMIT
PASSWORD_REUSE_TIME 60
PASSWORD_REUSE_MAX UNLIMITED;
```

在下一条语句中, PASSWORD_REUSE_MAX 子句指定,在可以再次使用当前口令之前,用户必须改变该口令的次数是 3。

```
CREATE PROFILE prof LIMIT
PASSWORD_REUSE_MAX 3
PASSWORD_REUSE_TIME UNLIMITED;
```

注意: 如果指定了 PASSWORD_REUSE_TIME 或 PASSWORD_REUSE_MAX 中的任何一个,就必须将另外一个设置成 UNLIMITED,或根本就不设置它。

23.4.4 口令的复杂性校验

可以用 PL/SQL 脚本(UTLPWDMG.SQL),来指定校验 Oracle 的口令的复杂性的例程。

口令复杂性校验例程执行如下检查:

- 口令的最小长度为 4。
- 口令不要与用户名相同。
- 口令至少要有字母、一个数字、和一个标点符号。
- 口令不应该是简单的或明显的字,如 welcome, account, database 或 user。
- 口令与其先前的口令至少有三个字符不一样。

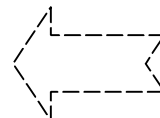
注意: Oracle 建议不要用 ALTER USER 语句来修改口令,因为它并不完全支持这个口令校验功能。而是应该用 OCIPasswordChange()来修改口令。

23.4.4.1 口令校验例程的格式化准则

可以用 PL/SQL 或第三方工具来增强现存的口令复杂性校验例程或创建其他的口令校验例程。

PL/SQL 调用必须采用如下格式:

```
routine_name
(
```



```

userid_parameter IN VARCHAR(30),
password_parameter IN VARCHAR (30),
old_password_parameter IN VARCHAR (30)
)
RETURN BOOLEAN

```

创建了一个新的例程之后，还必须用用户的配置文件或系统默认的配置文件，将其指定为口令校验例程。

```

CREATE/ALTER PROFILE profile_name LIMIT
PASSWORD_VERIFY_FUNCTION routine_name
SYS 必须拥有口令校验例程。

```

23.4.4.2 口令校验例程的例子

当你开发你自己的新口令的复杂性检查的例程时，可以将这个口令校验例程的例子作为模型。

默认的口令复杂性校验功能执行如下最小的复杂性检查：

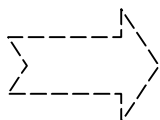
- 口令满足最小长度的需要。
- 口令不是用户名。可根据需要而改变此功能。

必须将这个功能创建到 SYS 模式中，而且在运行该脚本之前，必须用 SYS/SYSDBA (用户名/口令) 进行连接。

```

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

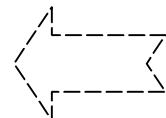
```



```

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

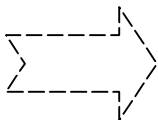
```



```
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;

```

23.5 审计的策略

安全管理员应该为每个数据库的审计过程都定义一个策略。当然，可以停用数据库审计，除非怀疑有可疑的行为。当需要审计时，安全管理员必须确定出要审计数据库的什么级别的细节；通常，当找到可疑行为的起因之后，在进行普通系统审计之后还要进行更特殊的审计。在第 26 章的“审计数据库的使用”中讨论审计。

23.6 安全清单

在商业活动中，信息安全、隐私、集体财产和数据的保护都是至关重要的。通过提供犀利的安全特性（如深层数据保护、审计、可升级的安全、安全寄托、数据交换），Oracle9i 全面考虑了信息安全的需要。

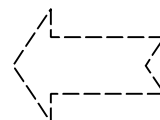
Oracle9i 数据库服务器在安全方面引导行业。然而，为了在任何商业环境中都充分利用 Oracle9i 提供的安全特性，Oracle9i 自身就应该被很好地保护。此外，正确地使用它的安全特性和遵循基本的安全惯例，将有助于使数据库免遭与数据库相关的威胁和攻击，并为 Oracle9i 提供更安全的操作环境。

这种安全检查列表，通过遵循和建议行业标准的“最佳安全实践”，为运作的数据库开发提供了以安全的方式配置 Oracle9i 的准则。

可以在 Oracle 文档中查到特殊的与数据库相关联的任务和行为的详细情况。

① 只安装所需的部分

除数据库服务器之外，Oracle9i CD 套件还包含了主要的选项和产品。仅在必须时，才安装附加的产品和选项。或者，按照典型安装（如果不想进行定制安装），就不会安装不需要的选项和产品。如果不使用附加的产品和选项，也就不必维护它们了。需要它们的时候，



总是可以合适地并容易地重新安装它们。

②锁定和过期默认的用户帐户

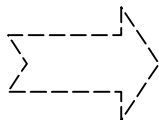
安装好的 Oracle9i 带有几个默认的（预先设置的）数据库服务器用户帐户。如果成功地安装了数据库服务器，那么，数据库客户管理工具（Database Client Administration, DBCA）就会自动地锁定和过期所有的默认的数据库用户帐户，除了如下帐户之外。

- SYS
- SYSTEM
- SCOOT
- DBSNMP
- OUTLN
- 三个 JSERV 用户

如果进行人工（不是利用 DBCA）安装 Oracle9i，则在成功地安装了数据库服务器之后，默认的数据库用户是不会被锁定的。如果任凭其处于默认的打开状态，就能用这些用户帐户进行非授权的数据访问或进行破坏数据库的操作。因此，在做完没有使用 DBCA 的任何形式的初始化安装之后，就应该锁定和过期除了 SYS, SYSTEM, SCOTT, DBSNMP, OUTLN 和三个 JSERV 数据库用户之外的，所有的默认的数据库用户帐户。Oracle9i 提供了 SQL 语句来执行这种操作。

在利用 DBCA 完成了 Oracle9i 的典型安装之后，其数据库用户的列表如下。

USERNAME（用户名）	ACCOUNT_STATUS（帐户状态）
ADAMS	EXPIRED & LOCKED
AURORA\$UTILITY\$	OPEN
AURORA\$ORB\$UNAUTHENTICATED	OPEN
BLAKE	EXPIRED & LOCKED
CLARK	EXPIRED & LOCKED
CTXSYS	EXPIRED & LOCKED
DBSNMP	OPEN
HR	EXPIRED & LOCKED
JONES	EXPIRED & LOCKED
LBACSYS	EXPIRED & LOCKED
MDSYS	EXPIRED & LOCKED
OE	EXPIRED & LOCKED
OLAPDBA	EXPIRED & LOCKED
OLAPSVR	EXPIRED & LOCKED
OLAPSYS	EXPIRED & LOCKED
ORDPLUGINS	EXPIRED & LOCKED
ORDSYS	EXPIRED & LOCKED
OS\$HTTP\$ADMIN	OPEN



(续表)

USERNAME (用户名)	ACCOUNT_STATUS (帐户状态)
OUTLN	OPEN
PM	EXPIRED & LOCKED
QS	EXPIRED & LOCKED
QS_ADM	EXPIRED & LOCKED
QS_CB	EXPIRED & LOCKED
QS_CBADM	EXPIRED & LOCKED
QS_CS	EXPIRED & LOCKED
QS_ES	EXPIRED & LOCKED
QS_OS	EXPIRED & LOCKED
QS_WS	EXPIRED & LOCKED
SCOTT	OPEN
SH	EXPIRED & LOCKED
SYS	OPEN
SYSTEM	OPEN

如果由于某种原因，需要不处于打开状态的任何默认的数据库服务器用户帐户时，数据库管理员 (DBA) 就只需要简单地将那个帐户解锁并用一个新的、有意义的口令来激活那个帐户。

③改变默认的用户口令

最烦人的可能危及 Oracle9i 的事是，即使安装之后，一个默认的数据库服务器用户帐户也还有一个与之相关联的默认的口令。

a. 改变管理用户的默认口令

在 Oracle9i 中，安装时，SYS 的默认口令是 CHANGE_ON_INSTALL，而 SYSTEM 的默认口令是 MANAGER。在安装数据库服务器之后，应立即改变用户 SYS 和用户 SYSTEM 的默认口令。

b. 改变所有用户的默认口令

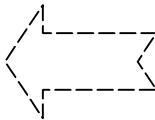
在 Oracle9i 中，安装时，SCOTT 的默认口令是 TIGER，三个 JSERV 帐户 (AURORA\$JIS\$UTILITY\$, AURORA\$ORB\$UNAUTHENTICATED 和 OSE\$HTTP\$ADMIN) 的默认口令是随机产生的。其他每个帐户的默认口令与该用户的帐户完全一样 (例如：用户 MDSYS 的默认口令是 MDSYS)。

安装之后，应立即改变 SCOTT，DBSNMP，OUTLN 和三个 JSERV 用户帐户的口令。如果安装之后，需要激活其他任何默认的被锁定或过期的用户帐户，就给该用户帐户分配一个新的、有意义的口令。

尽管 Oracle 并不明确地要求改变用户 SCOTT 的默认口令，然而 Oracle 建议也应锁定这个用户帐户，除非它正被频繁使用。

b. 实施口令管理

Oracle 建议，应该将数据库所提供的基本口令管理准则 (例如：口令长度、历史记录、复杂性等等) 运用于所有的用户口令，并要求所有的用户定期地改变他们的口令。



Oracle 还建议，如果可能的话，要利用带有网络鉴别服务（如 Kerberos）、令牌卡、灵巧卡或 X.509 证书的 Oracle 高级安全（Oracle 企业版的一个选项）。这些服务提供严格的用户鉴别，以便更好地防止对 Oracle9i 进行未经授权访问。

④启用数据字典保护

Oracle 建议，由用户实现对数据字典的保护，以防止拥有 ANY 系统权限的用户，在数据字典上使用此权限。

为了启用数据字典保护，就用如下方式设置 O7_DICTIONARY_ACCESSIBILITY 初始化参数：

```
O7_DICTIONARY_ACCESSIBILITY = FALSE
```

这样做之后，只有那些被授权的按 DBA 权限进行连接的用户（例如：CONNECT / AS SYSDBA），才可以在数据字典上使用 ANY 系统权限。如果此参数没有被设置成上述建议的值，那么举例来说，具有 DROP ANY TABLE 系统权限的任何用户，都能恶意地删除数据字典中的某些部分。

然而，如果一个用户需要用视图来访问数据字典，还是可以给那个用户授予 SELECT ANY DICTIONARY 系统权限的。

注意，在 Oracle9i 中，默认 O7_DICTIONARY_ACCESSIBILITY = FALSE；而在 Oracle8i 中，该参数的默认值是 TRUE，所以，要想启用这个安全特性，就必须特意地将其改成 FALSE。

⑤实施最少权限原则

a. 仅授予所需的权限

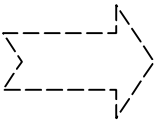
不要给数据库用户提供比其所需还要多的权限。换句话说，最少权限原则就是：只给用户真正需要的、为了高效和简洁地执行他们的工作所需的权限。

为了实现最少权限，就要限制：1）授予给数据库用户的 SYSTEM 权限和 OBJECT 权限的数量，2）都用 SYS 权限连接到数据库的数量。例如：通常不需要将 CREATE ANY TABLE 权限授予非 DBA 权限的用户，

b. 撤消 PUBLIC 中的不需要的权限

撤消数据库服务器用户组 PUBLIC 中的所有不需要的权限和角色。PUBLIC 是授予给 Oracle 数据库中的每一个用户的默认角色。任何数据库用户都可以行使授予给 PUBLIC 的权限。这些权限包括执行各种 PL/SQL 包的 EXECUTE 权限，这就会允许一个只最少权限的用户可以访问或执行那些不允许他直接访问的包。下表列出了潜在的会被误用的更强大的包：

包	说明
UTL_SMPT	此包允许将任意的邮件消息，从一个任意的用户发送到另外一个任意的用户。将此包授予 PUBLIC，可能就会允许邮件消息的未授权交换。
UTL_TCP	此包允许数据库服务器建立向外的、通向任何接收（或等待）的网络服务的网络连接。所以，就可能在数据库服务器和任何等待的网络服务之间发送任意的数据。



(续表)

包	说明
UTL_HTTP	此包允许数据库服务器使用 HTTP 来请求或检索数据。将此包授予 PUBLIC，可能就会允许使用 HTML 格式将数据发送给怀有恶意的网络站点。
UTL_FILE	如果配置不合适，此包就会允许在文本层上访问宿主操作系统上的任何文件。即使配置得合适，此包也不会区分它所调用的应用和一个应用的结果。UTL_FILE 可能会将任意数据写到其他应用所写的同样位置上。
DBMS_RANDOM	此包用于加密已存储的数据。一般情况下，大多数用户不应该有加密数据的权限，因为如果密钥的生成、存储、管理是不安全的话，可能就不能恢复被加密的数据了。

这些包，对于那些需要它们并且需要合适配置和使用它们的应用来说，是非常有用的。但这些包可能也会不适合于其他应用。所以，除非绝对的需要，就应从 PUBLIC 中将它们撤消。

c. 制运行时工具中的许可

不要将“all permissions”(所有许可)权限分配给任何数据库服务器运行工具，如 Oracle Java 虚拟机 (Oracle Java Virtual Machine, OJVM)。对于这些可能在数据库服务器外面执行文件和包的工具来说，应该将特殊的许可权限授予明确的文档根文件路径。

一个易受攻击的运行时调用的例子：

```
call dbms_java.grant_permission('SCOTT',
                                'SYS:java.io.FilePermission','<<ALL FILES>>','read');
```

一个更好的（更安全）的运行时调用的例子：

```
call dbms_java.grant_permission('SCOTT',
                                'SYS:java.io.FilePermission','<<actual directory path>>','read');
```

⑥有效地实施访问控制

a. 正确地鉴别客户

远程鉴别是由 Oracle9i 提供的一个安全特性。如果启用 (TRUE) 它，它就会处理连接到 Oracle 数据库的远程客户的用户鉴别问题。所以，数据库绝对相信任何客户都已经进行了正确的鉴别。注意，在一般情况下，并不相信客户（如 PC 机）会正确地执行操作系统鉴别，所以启用这个特性是一个比较差的安全习惯。

一个更安全的配置是，将此特性关闭 (FALSE)，这就会对连接到 Oracle 数据库的客户，实施合适的、基于服务器的鉴别。

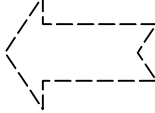
为了限制远程鉴别，从而由数据库来处理客户的信任问题，就用如下方式设置 REMOTE_OS_AUTHENT 初始化参数：

```
REMOTE_OS_AUTHENT = FALSE
```

b. 限制操作系统的用户的数量

将 Oracle9i 主机（物理机器）上的有操作系统帐户（管理的、根权限的、最少权限的）的用户数量尽可能地减少到最少。

Oracle 还建议，除非有 Oracle 公司的指示，否则不允许任何有权限的操作系统用户或者 Oracle 的拥有者，修改 Oracle9i 家族（安装）目录的默认文件和目录的许可。



⑦限制网络访问

a. 用防火墙

将数据库服务器放在防火墙的后面。Oracle9i 的网络基本设施, Oracle Net (以前所知的 Net8 和 SQL* Net), 提供了对来自于各个厂家的各种防火墙的支持。被支持的代理防火墙包括 Network Associates 公司的"Gauntlet"和 Axent 公司的"Raptor"。被支持的包过滤防火墙有 Cisco 公司的"PIX"防火墙。被支持的状态检查防火墙(更复杂的包过滤防火墙)包括 CheckPoint 公司的 Firewall-1。

b. 永远不要穿透防火墙

如果 Oracle9i 是在一个防火墙的后面, 那么, 在任何情况下都不要穿透该防火墙; 例如: 不要打开 Oracle 监听器的 1521 端口来与 Internet 连接, 反之亦然。

这么做就会引入许多重要的安全弱点, 包括打开更多的穿透防火墙的端口、多线程操作系统服务器的问题、以及泄露防火墙后面的数据库中的重要信息。另外, 一个没有用已建立的口令来运行的 Oracle 监听器, 就可能被用于探查有关数据库的关键的细节, 因为监听器会监听该数据库的轨迹、登录信息、标识信息、数据库描述符、服务名等。

如此多余的信息以及配置不好的防火墙的可用性, 就会给攻击者提供更多的, 对目标数据库进行恶意攻击的机会。

c. 防止对 Oracle 监听器进行未授权的管理

为了防止对 Oracle 监听器进行远程配置, 就要给 Oracle 监听器建立一个有意义的、构造良好的口令。另外, 可以用如下方式设置 listener.ora (Oracle 监听器的控制文件) 中的安全配置参数:

```
ADMIN_RESTRICTIONS_listener_name = ON
```

这么做也能防止对 Oracle 监听器进行未授权的管理。

d. 检查网络 IP 地址

利用 Oracle Net 的"valid node checking"(有效节点检查)安全特性, 可以允许或拒绝从指定 IP 地址的网络客户来访问 Oracle 服务器进程。要想使用这种特性, 就要将 protocol.ora (Oracle Net 配置文件) 中的参数设置如下:

```
tcp.validnode_checking = YES
tcp.excluded_nodes = {list of IP addresses}
tcp.invited_nodes = {list of IP addresses}
```

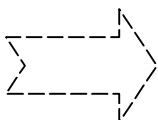
第一个参数启用这种特性, 而后两个参数分别拒绝或允许通过指定的客户 IP 地址连接到 Oracle 监听器 (从而防止了潜在的拒绝服务的攻击)。

e. 加密网络通信

如果可能的话, 就利用 Oracle 高级安全 (Oracle Advanced Security) 选项, 对客户、数据库和应用服务器之间的网络通信进行加密。(注意, Oracle 高级安全选项仅在 Oracle 数据库的企业版中才可用)。

f. 使操作系统健壮

通过停止不必要的操作系统服务, 可使宿主机的操作系统健壮。UNIX 和 Windows 两个平台都提供了各种操作系统服务, 但对许多配置来说, 它们中的大多数是不必要的。这



些服务包括：FTP，TFTP，TELNET 等等。对每一种被停止的服务来说，应确信关闭了相应的 UDP 和 TCP 两种端口。光是停止一种端口而不停止另一种端口，并不能使操作系统更安全。

⑧应用所有的安全补丁和安全工作区

对于 Oracle9i 所驻留的操作系统和 Oracle9i 本身，以及所有已安装了的 Oracle 选项和组件来说，应该总是使用所有与其相关的和当前的安全补丁。

请定期查阅 Oracle 技术网络中的安全站点，以便获得 Oracle 公司发布的安全警告的详细信息。

<http://otn.oracle.com/deploy/security/alerts.htm>

<http://technet.oracle.com/deploy/security/alerts.htm>

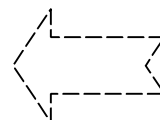
还要查阅 Oracle 全球技术支持服务(Oracle Worldwide Support Services)站点, Metalink, 以便获得可用的和即将发布的有关安全的补丁的详细信息。

<http://metalink.oracle.com>

⑨运用 Oracle 的安全产品

如果你确信发现了 Oracle9i 的安全弱点，就用 Metalink 向 Oracle 全球技术支持服务(Oracle Worldwide Support Services) 站点提交一份 iTAR，或者用 e-mail 向如下地址发送一个完整的问题描述，包括产品版本和平台，以及任何使用的脚本和（或者）例子。

secalert_us@oracle.com



第 24 章 管理用户和资源

本章描述如何控制对 Oracle 数据库的访问，包括如下内容：

- 会话和用户的许可
- 用户鉴别的方法
- 管理 Oracle 用户
- 用配置文件来管理资源
- 查看有关数据库用户和配置文件的信息

24.1 会话和用户的许可

Oracle 有助于使你保证你的站点遵守其 Oracle 数据库服务器的许可证协议。如果你的站点是按并发使用方式来验证许可，你就能跟踪和限制并发地连接到数据库的会话的数量。如果你的站点是按命名用户来验证许可，你就能限制在数据库中创建的命名用户的数量。在这两种情况下，你都要控制许可证工具，并必须启用该工具和设置合适的限制。通过在初始化参数文件中设置如下初始化参数来做这件事：

- LICENSE_MAX_SESSIONS
- LICENSE_SESSIONS_WARNING
- LICENSE_MAX_USERS

为了使用许可证工具，你必须知道你的站点具有什么类型的许可证协议、会话或命名用户的最大数量。你的站点可以使用任何类型的许可方式（并发使用或命名用户），但不能同时使用这两者。

注意：在个别情况下，站点最好是采用无限的许可证，而不采用并发使用或命名用户许可方式。仅在这些情况下才停用许可机制，并忽略初始化文件中的 LICENSE_MAX_SESSIONS，LICENSE_SESSIONS_WARNING 和 LICENSE_MAX_USERS，或将它们全都设置成 0。

本节描述有关会话的几个方面和用户许可方式，包括如下内容：

- 并发使用许可
- 命名用户的限制
- 查看许可方式的限制和当前值

24.1.1 并发使用许可

并发使用许可的方式能够限制同时连接到在指定计算机上的数据库的会话的数量。在启动一个实例之前，可以设置并发会话的数量限制。当数据库运行时还可以修改并发会话的最大数量。

24.1.1.1 连接权限

当达到实例的会话限制之后，仅仅具有 RESTRICTED SESSION 权限（一般是 DBA）的用户才能连接到数据库。当一个具有 RESTRICTED SESSION 权限的用户进行连接时，

Oracle 给该用户发送一条消息，指出已经到达最大的限制了，并且在报警文件中写一条消息。当到达最大值时，应该仅仅为了终止不再需要的进程，才进行连接。

不要抬高对许可方式的限制，除非已经对 Oracle 许可证协议做了升级。

另外，为了设置最大并发会话限制，可以设置并发会话数量的警告限制。当达到这个限制时，另外的用户还可以继续连接（直到最大限制）。但是，每次连接时，Oracle 都会在报警文件中写一条消息，并且给每个具有 RESTRICTED SESSION 权限的正在连接的用户发送一条警告消息，指出，就要达到最大值了。

如果用户用管理员权限进行连接，该限制仍然有效。但是，Oracle 是在用户执行了第一条语句后才实施该限制。

除了实施并行使用限制之外，Oracle 还跟踪每个实例的并行会话的最高数量。可以使用这个“高水位标记”来帮助你决定，是否需要检查一下你的 Oracle 许可证。

对于运行 Oracle 真实应用簇的实例而言，每个实例可以有其自己的并发使用限制和警告限制。但是，所有实例的限制之和不能超过该站点的并发使用许可证限制。

警告：每个通过组合软件或硬件（如 TP 监测器）连接到 Oracle 的会话都单独地使用并发用户限制。但是 Oracle 许可证机制不能区别这种连接方式的会话数量。如果站点使用组合软件或硬件，就必须考虑这个问题并将最大并发使用限制设置得小点，以处理组合会话的问题。

24.1.1.2 设置会话的最大数量

为了给实例设置并发会话的最大数量，就要设置 LICENSE_MAX_SESSIONS 初始化参数。本例将并发会话的最大数量设置成 80。

```
LICENSE_MAX_SESSIONS = 80
```

如果设置了这个限制，就不需要设置警告限制（LICENSE_SESSIONS_WARNING）了。然而，使用警告限制会使最大限制更好管理，因为，它提供了一个站点就要达到最大使用限制的预先的通知。

24.1.1.3 设置会话警告限制

为了给实例设置警告限制，就要在用来启动该实例的初始化参数文件中设置 LICENSE_SESSIONS_WARNING 初始化参数。在如下例子中，从第 60 个开始的每个新的连接，都会在报警文件中写一条警告消息。

```
LICENSE_SESSIONS_WARNING = 60
```

要将会话警告限制值设置成小于并发使用最大限制（LICENSE_MAX_SESSION）的值。

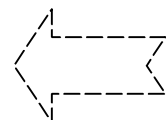
24.1.1.4 在数据库运行时修改并发使用限制

为了在数据库运行时修改最大并发使用限制或警告限制，使用带有合适选项的 ALTER SYSTEM 语句。如下语句将并发会话的最大限制修改成 100：

```
ALTER SYSTEM SET LICENSE_MAX_SESSIONS = 100;
```

如下语句修改警告限制和最大限制两者：

```
ALTER SYSTEM
```



```
SET LICENSE_MAX_SESSIONS = 64  
LICENSE_SESSIONS_WARNING = 54;
```

如果将任何限制修改成了小于当前会话数量的值，则当前会话仍然保留。但是，新的限制会施加于所有后来的连接，直到关闭该实例。为了永久地改变该限制，就应在初始化参数文件中改变相应参数的值。

要想在数据库运行时修改并发使用限制，你就必须要具有 ALTER SYSTEM 权限。同样，要想在已经达到实例的最大限制时还能连接到实例，你就必须要具有 RESTRICTED SESSION 权限。

警告：不要抬高并发使用限制，除非已经合理地升级了 Oracle 服务器许可证。请与你的 Oracle 代理商联系以获得更多的信息。

24.1.2 命名用户的限制

命名用户许可方式限制被授权使用指定计算机上的 Oracle 的人数。为了实施这个许可，可以在启动实例之前，设置能在数据库中创建的用户数的限制。还可以在实例运行期间修改该最大用户数或完全停用这个限制。当达到这个限制后，你就不能再创建用户了。如果你真这么做，Oracle 就返回一个错误，指出已经创建了最大数量的用户，并在报警文件中写一条消息。

该机制的实现基于一个假设，即访问数据库的每个用户都有唯一的用户名，并且不存在相同的用户名。如果使用命名用户许可方式，就不允许多个用户用相同的用户名来连接到数据库了。

24.1.2.1 设置用户限制

为了限制在数据库中创建的用户数，就要在数据库的初始化参数文件中设置 LICENSE_MAX_USERS 初始化参数。如下例子将最大用户数设置成 200：

```
LICENSE_MAX_USERS = 200
```

如果在启动数据库时，数据库包含了多于 LICENSE_MAX_USERS 的用户，则 Oracle 返回一条警告消息并在报警文件中写一条相应的消息。在用户数降到限制之下以前，或者在删除用户或者在升级 Oracle 许可证之前，是不能创建另外的用户的。

对于运行 Oracle 真实应用簇的实例而言，连接到相同数据库的所有实例应该具有相同的命名用户限制。

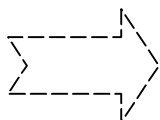
24.1.2.2 修改用户限制

为了修改最大命名用户限制，使用带有 LICENSE_MAX_USERS 选项的 ALTER SYSTEM 语句。如下语句将被定义用户的最大数量修改成 300：

```
ALTER SYSTEM SET LICENSE_MAX_USERS = 300;
```

如果试图将该限制修改成比当前用户数还小的值，则 Oracle 返回一个错误并继续使用以前的限制。如果成功地修改了该限制，则新的限制在关闭实例之前都有效。若要永久地改变该限制，则要在初始化参数文件中修改 LICENSE_MAX_USERS 的值。

要想能修改最大命名用户限制，你必须要具有 ALTER SYSTEM 权限。



警告：不要抬高命名用户限制，除非已经合理地升级了 Oracle 许可证。请与你的 Oracle 代理商联系以获得更多的信息。

24.1.3 查看许可方式的限制和当前值

V\$LICENSE 数据字典视图允许你查看所有许可证设置的当前限制、当前的用户会话数、自从实例被创建以来的最大的并发用户会话数。下面有一个例子。你可以用这个信息来决定是否需要升级 Oracle 许可证，以允许更多的并发会话或命名用户。

```
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，当关闭数据库时，Oracle 会将会话的高水位标记写到数据库的报警文件中，这样，你就可以在此检查它了。

要想查看当前在数据库中定义的命名用户数，可用如下查询：

```
SELECT COUNT(*) FROM DBA_USERS;
COUNT(*)
-----
174
```

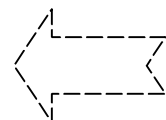
24.2 用户鉴别的方法

在允许用户创建数据库会话之前，Oracle 提供了几个鉴别用户的方法：

- ①可以定义由数据库执行用户识别和鉴别的用户。这被称为数据库鉴别。
- ②可以定义由操作系统或网络服务器执行鉴别的用户。这被称为外部鉴别。
- ③可以定义由 SSL (Secure Sockets Layer, 安全套接字层) 进行全局鉴别的用户。这些用户被称为全局用户。对于全局用户，可用企业目录来允许他们通过全局角色访问数据库。
- ④可以指定允许通过中间层服务器进行连接的用户。中间层服务器鉴别和呈现用户的身份，并允许为用户启用特殊的角色。这被称为代理鉴别和授权。

下面几节讨论这些鉴别方法：

- 数据库鉴别
- 外部鉴别
- 全局鉴别和授权
- 代理鉴别和授权



24.2.1 数据库鉴别

如果为用户选择了数据库鉴别，则 Oracle 全权管理用户帐户、口令和该用户的鉴别。要使 Oracle 鉴别一个用户，就要在创建或更改用户时为该用户指定一个口令。用户可在任何时候修改其口令。口令用加密格式存储。每个口令必须用单个字节的字符组成，即使数据库使用多字节字符集也要这样。

为了在使用数据库鉴别时加强安全性，Oracle 建议使用口令管理，包括帐户加锁、口令使用期和过期、口令历史和口令复杂性校验。

24.2.1.1 创建由数据库鉴别的用户

如下语句创建一个由 Oracle 识别和鉴别的用户。无论何时连接到 Oracle 时，用户 scott 必须指定口令 tiger。

```
CREATE USER scott IDENTIFIED BY tiger;
```

24.2.1.2 数据库鉴别的优点

数据库鉴别的优点如下：

- 由数据库控制用户帐户和所有鉴别。不需要依赖数据库之外的任何东西。
- 使用数据库鉴别时，Oracle 提供强大的口令管理特性以加强安全性。
- 当只有较少用户群时，这个方法对管理员来说较容易。

24.2.2 外部鉴别

当为用户选择了外部鉴别时，则 Oracle 维护用户的帐户，但由外部服务来执行口令管理和用户鉴别。这个外部服务可以是操作系统或网络服务，如 Oracle Net。

使用外部鉴别后，数据库就靠基础操作系统或网络鉴别服务来限制对数据库帐户的访问了。数据库口令不用于这种类型的登录。如果操作系统或网络服务允许，就可以用它们来鉴别用户。如果要这么做，就请设置初始化参数 OS_AUTHENT_PREFIX，并在 Oracle 用户名中使用这个前缀。OS_AUTHENT_PREFIX 参数定义了一个前缀，Oracle 会将其增加到每个用户的操作系统帐户名之前。当一个用户尝试连接时，Oracle 就会将这个带前缀的用户名与数据库中的 Oracle 用户名进行比较。

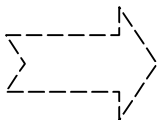
例如，假设 OS_AUTHENT_PREFIX 被设置成：

```
OS_AUTHENT_PREFIX=OPS$
```

注意：在某些操作系统中，OS_AUTHENT_PREFIX 初始化参数的文本是大小写敏感的。有关该初始化参数的更多的信息，参见你的相关操作系统的 Oracle 文档。

当一个操作系统帐户名为 tsmith 的用户打算连接到 Oracle 数据库，并由操作系统进行鉴别时，Oracle 就检查是否存在一个对应的数据库用户 OPS\$tsmith，如有，则允许用户进行连接。所有对由操作系统鉴别的用户的参照都必须包含该前缀，正如所见到的 OPS\$tsmith。

这个参数的默认值是 OPS\$，以保证与 Oracle 以前版本的后向兼容。当然，可以随意将该前缀设置某个其他字符串或空字符串（双引号的空集：“”）。使用空字符串就不在操作



系统帐户名之前增加任何前缀了，因此，Oracle 用户名就与操作系统用户名完全一样了。

设置 OS_AUTHENT_PREFIX 之后，就应在数据库的生命期中使其保持不变。如果修改了前缀，就不能用任何包含老前缀的用户名建立连接了，除非更改用户名使其使用口令鉴别。

24.2.2.1 创建一个外部鉴别的用户

如下语句创建一个由 Oracle 识别并由操作系统或网络服务鉴别的用户。本例假设 OS_AUTHENT_PREFIX=""。

```
CREATE USER scott IDENTIFIED EXTERNALLY;
```

用 CREATE USER ... IDENTIFIED EXTERNALLY 语句，创建必须由操作系统或网络服务鉴别的数据库帐户。Oracle 靠这个外部的登录鉴别，来保证特殊的操作系统用户能够访问特殊的数据库用户。

24.2.2.2 操作系统鉴别

默认时，Oracle 仅允许操作系统鉴别在安全连接之上的登录。因此，如果希望操作系统鉴别一个用户，则默认时该用户不能通过 Oracle Net 连接到数据库。这意味着该用户不能进行使用共享服务配置的连接，因为该连接使用了 Oracle Net。这个默认的限制可防止一个远程用户通过一个网络连接而模仿另一个操作系统用户。

如果不介意远程用户通过网络连接而模仿另一个操作系统用户，并且希望使用操作系统用户鉴别来鉴别网络客户，就在数据库的初始化参数文件中将初始化参数 REMOTE_OS_AUTHENT 设置成 TRUE（默认是 FALSE）。将初始化参数 REMOTE_OS_AUTHENT 设置成 TRUE，就允许 RDBM（关系数据库管理系统）接受通过非安全连接接收到的客户操作系统用户名，并将其用于帐户访问。这个修改会在下一次启动实例，并加载数据库后生效。

通常，通过宿主操作系统的用户鉴别提供如下好处：

- 用户可以更快和更方便地连接到 Oracle，不需要指定单独的数据库用户名或口令。
- 在数据库和操作系统审计跟踪中的用户项是相一致的。

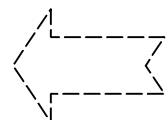
24.2.2.3 网络鉴别

网络鉴别是用 Oracle 高级安全来执行的，可以配置 Oracle 高级安全以便使用第 3 方服务，如 Kerberos。如果用 Oracle 高级安全作为仅有的外部鉴别服务，则对 REMOTE_OS_AUTHENT 的设置就无关了。因为 Oracle 高级安全仅允许安全连接。

24.2.2.4 外部鉴别的优点

外部鉴别的优点如下：

- 可有更多的可选的鉴别机制，如灵巧卡、指纹、Kerberos 或操作系统。
- 许多网络鉴别服务，如 Kerberos 和 DCE，支持单一签约（signon）。这意味着用户只需记忆更少的口令。
- 如果已经使用某些外部机制来鉴别，如上述所列之一，这与在数据库中也使用该



机制相比，需要较少的管理开销。

24.2.3 全局鉴别和授权

Oracle 高级安全选项允许在一个基于 LDAP 的目录服务中，对与用户相关的信息（包括授权）进行集中管理。可以在数据库中将用户标记为全局用户，即他们被 SSL 鉴别，并且对这些用户的管理是在数据库之外、由集中式目录服务来做。全局角色在数据库中被定义，并仅仅被那个数据库所认识，但由目录服务对这些角色进行授权。

注意：也可以让 SSL 来鉴别用户，不在目录中管理其授权；即他们仅仅具有局部数据库角色。

这种集中式管理方式允许创建企业用户和企业角色。企业用户是在目录中定义和管理的。他们在整个企业中具有唯一标识，并能被赋予决定他们在多个数据库中的访问权限的企业角色。企业角色由一个或多个全局角色组成，可以被看作全局角色的容器。

24.2.3.1 创建由目录服务授权的用户

可以用两种方法来指定由目录服务授权的用户。

创建全局用户 如下语句说明全局用户的创建，它由 SSL 鉴别并由企业目录服务授权：

```
CREATE USER scott
IDENTIFIED GLOBALLY AS 'CN=scott,OU=division1,O=oracle,C=US'
```

在 AS 子句中提供的字符串提供一个意味着企业目录的标识（有区别的名字，或 DN）。

此时，scott 是真正的全局用户了。但是，这里的缺点是，以后必须在每个需要访问的数据库中（包括目录）创建 scott。

创建不依赖于模式的用户 创建不依赖于模式的用户就允许多个企业用户访问数据库中的共享模式。不依赖于模式的用户是：

- 由 SSL 鉴别
- 不是用任何类型的 CREATE USER 语句创建到数据库中的
- 其权限是在目录中管理的用户
- 连接到共享模式的用户

创建不依赖于模式的用户的过程如下：

①按如下方法在数据库中创建共享模式。

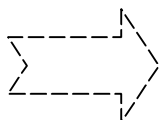
```
CREATE USER appschema IDENTIFIED GLOBALLY AS '';
```

②现在，在目录中创建多个企业用户和一个映射对象。

映射对象告诉数据库如何将用户的 DN 映射到共享模式。可以或者做全部映射（每个唯一 DN 对应一个目录项），或者，例如，将包含如下 DN 组成的每个用户映射到 appschema：

```
OU=division,O=Oracle,C=US
```

由于大多数用户不需要他们自己的模式，因此不依赖于模式的用户实现了将用户从数据库中分离出来。在数据库中创建多个共享相同模式的模式的用户，并作为企业用户，他们还可以访问在其他数据库中的共享模式。



24.2.3.2 全局鉴别和全局授权的优点

全局用户鉴别和全局授权的某些优点如下：

- 提供了强大的使用 SSL 或 Windows NT 本身鉴别的鉴别方法。
- 允许在企业范围内对用户和权限进行集中式管理
- 对管理员更容易了——因为不再需要在企业的每个数据库中为每个用户创建一个模式。
- 实现了单一签约——用户只需签约一次就可以访问多个数据库和服务
- `CURRENT_USER` 数据库链接是按全局用户进行连接的。按存储过程的含义，一个局部用户可按全局用户进行连接——不需要在该链接的定义中存储全局用户的口令。

24.2.4 代理鉴别和授权

可以设计一个中间层服务器以安全的方式来代理客户。

Oracle 提供了三种形式的代理鉴别：

- 中间层服务器用数据库服务器和客户来鉴别它自己，此时，应用的用户或另一个应用就是用中间层服务器来鉴别它自己。客户身份可以在直到数据库的整个过程中维持不变。
- 客户，此时为数据库用户，不是由中间层服务器鉴别的。客户身份和数据库口令通过中间层服务器被传送到数据库服务器用于鉴别。
- 客户，此时为全局用户，是由中间层服务器鉴别的，并通过中间层传送下面所列之一，以便检索客户的用户名。
- 有区别的名称（DN）
- 证书

在所有情况下，管理员必须授权中间层服务器扮演客户的特征。

要授权中间层服务器代理客户，使用 `ALTER USER` 语句的 `GRANT CONNECT THROUGH` 子句。还可以为中间层指定一个，在作为客户进行连接时，允许其扮演的角色。

可以审计按客户的特征由中间层服务器所做的操作。

可以查询 `PROXY_USERS` 数据字典视图，以便查看哪个用户当前被授权通过中间层进行连接。

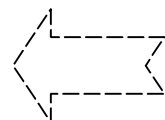
用 `ALTER USER` 语句的 `REVOKE CONNECT THROUGH` 子句来禁止代理连接。

24.2.4.1 授权中间层代理和鉴别用户

如下语句授权中间层服务器 `appserve` 按用户 `bill` 进行连接。它使用 `WITH ROLE` 子句来指定 `appserve` 激活的、与 `bill` 有关的、除 `payroll` 之外的角色。

```
ALTER USER bill
GRANT CONNECT THROUGH appserve
WITH ROLE ALL EXCEPT payroll;
```

为了撤消按用户 `bill` 进行连接的对中间层服务器（`appserve`）的授权，使用如下语句：



```
ALTER USER bill REVOKE CONNECT THROUGH appserve;
```

24.2.4.2 授权中间层代理用其他方法鉴别的用户

用 `ALTER USER ... GRANT CONNECT THROUGH` 语句的 `AUTHENTICATED USING` 子句，来授权中间层代理用户但不鉴别用户。此时，仅支持 `PASSWORD` 方法。

如下语句说明这种形式的鉴别：

```
ALTER USER mary
GRANT CONNECT THROUGH midtier
AUTHENTICATED USING PASSWORD;
```

在上面的语句中，中间层服务器 `midtier` 被授权按 `mary` 进行连接，并且 `midtier` 还要将 `mary` 的口令传送到数据库服务器以便用于鉴别。

24.2.4.3 授权中间层代理用有区别的名字标识的用户

此时，如下语句授权中间层服务器 `WebDB`，将全局用户 `jeff` 的有区别的名字呈现给数据库服务器。有区别的名字被用于检索用户名。用户 `jeff` 已由中间层服务器 `WebDB` 鉴别了。

```
ALTER USER jeff
GRANT CONNECT THROUGH WebDB
AUTHENTICATED USING DISTINGUISHED NAME;
```

可选地，可以授权中间层服务器呈现整个证书（包括有区别的名字）。如下语句所示：

```
ALTER USER jeff
GRANT CONNECT THROUGH WebDB
AUTHENTICATED USING CERTIFICATE;
```

传送整个证书会花费鉴别时间。但是，某些应用会使用证书中的其他信息。

24.3 管理 Oracle 用户

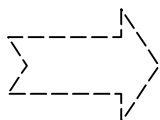
每个 Oracle 数据库都有一串有效的数据库用户。为了访问数据库，用户必须运行数据库应用，并通过一个定义在数据库中的有效的用户名来连接到数据库实例。这一节解释如何管理数据库的用户，包括如下内容：

- 创建用户
- 更改用户
- 删除用户

24.3.1 创建用户

用 `CREATE USER` 语句来创建用户。要想创建用户，你必须要有 `CREATE USER` 系统权限。因为这是一个强大的权限，所以通常 `DBA` 或安全管理员才是具有 `CREATE USER` 系统权限的仅有的用户。

如下语句创建一个用户，并指定该用户的口令、默认表空间、创建临时段所用的临时表空间、表空间限额和配置文件。



```

CREATE USER jward
IDENTIFIED BY aZ7bC2
DEFAULT TABLESPACE data_ts
QUOTA 100M ON test_ts
QUOTA 500K ON data_ts
TEMPORARY TABLESPACE temp_ts
PROFILE clerk;
GRANT connect TO jward;

```

新创建的用户直到被授予 CREATE SESSION 系统权限之后才能连接到数据库。通常，给新创建的用户授予一个角色，就像预定义的角色 CONNECT，如本例所示，以便指定访问数据库所需的 CREATE SESSION 和其他基本的权限。

参考上面的例子，本节讨论创建用户的如下几个方面：

- 指定名字
- 设置用户的鉴别方式
- 分配默认表空间
- 分配表空间限额
- 分配临时表空间
- 指定配置文件
- 设置默认角色

24.3.1.1 指定名字

在每个数据库中，相对于其他用户名和角色而言，用户名必须是唯一的。用户和角色不能用相同的名字。进一步说，每个用户有一个相关的模式。在该模式中，每个模式对象都有一个唯一的名字。

24.3.1.2 设置用户的鉴别方式

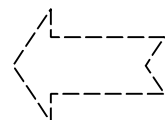
在前一个 CREATE USER 语句中，该新用户是由数据库鉴别的。此时，正在连接的用户必须向数据库提供正确的口令才能成功地连接。

24.3.1.3 分配默认表空间

每个用户都应该有一个默认表空间。当用户创建模式对象，可又没有为其指定容纳它的表空间时，Oracle 就在该用户的默认表空间中存储该对象。

每个用户的默认表空间的默认设置是 SYSTEM 表空间。如果用户不创建对象，也没权限这么做时，则这个默认设置就行了。然而，如果用户会创建任何类型的对象，就应该专门地为该用户分配一个默认表空间。使用不同于 SYSTEM 的表空间，会减少数据字典对象和用户对象之间，为使用相同数据文件而带来的冲突。通常，将用户数据存储在 SYSTEM 表空间中是不明智的。

可以在创建用户时给该用户设置一个默认表空间，以后再用 ALTER USER 语句对其进行修改。修改用户的默认表空间仅仅影响设置被改变之后才被创建的对象。



当指定用户的默认表空间时，还要指定在那个表空间上的限额。

在前面的 CREATE USER 语句中，jward 的默认表空间是 data_ts，他在那个表空间上的限额是 500K。

24.3.1.4 分配表空间限额

可以给每个用户在任何表空间（除临时表空间以外）上分配限额。分配限额做两件事：

- 具有创建某种类型的对象的权限的用户，可以在指定的表空间上创建那种对象。
- Oracle 将在指定的表空间中，限制可以分配用于存储用户对象的空间数量，使其被限制在限额的数量以内。

默认时，用户在数据库中的任何表空间上都有限额。如果用户具有创建模式对象的权限，你就必须给他分配一个限额，以允许该用户创建对象。至少是，为该用户在默认表空间中分配一个限额，并在他们可以创建对象的其他表空间中分配附加的限额。

可以或者在每个表空间中为用户分配单独的、指定数量的磁盘空间限额，或者在所有表空间中为用户分配无限的磁盘空间限额。指定的限额能防止用户对象在数据库中消耗太多的空间。

可以在创建用户时为该用户分配表空间限额，以后再增加或修改该限额。如果新的限额少于老的限额，则发生如下情况：

- 如果用户已经超过了新的表空间限额，就不能在该表空间中为该用户的对象分配更多的空间了，直到这些对象的合并后的空间降到新的限额之下。
- 如果用户还没有超过新的表空间限额，或者如果在该表空间中的该用户对象所使用的空间降到了新的表空间限额之下了，则可以为该用户的对象分配空间直到新的限额。

撤消用户在表空间中创建对象的能力 通过将用户的当前限额修改成零，就可以撤消该用户在该表空间中创建对象的能力。限额为零之后，在该表空间中的该用户的对象仍保留，但不能创建新的对象，也不能给现存的对象分配新的空间。

UNLIMITED TABLESPACE 系统权限 为了允许用户使用数据库中任何表空间中的所有空间，就要给该用户授予 UNLIMITED TABLESPACE 系统权限。这将覆盖该用户的所有明确的表空间限额。如果以后撤消了该权限，则明确的限额仍发挥作用。只能给用户分配该权限，而不能给角色分配该权限。

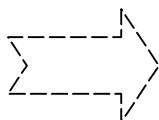
在授予 UNLIMITED TABLESPACE 系统权限之前，先考虑这样做的后果：

优点：

- 可以授予用户在一个语句中无限制地访问数据库的所有表空间。

缺点：

- 该权限覆盖该用户的所有明确的表空间限额。
- 不能有选择地撤消一个具有 UNLIMITED TABLESPACE 权限的用户的表空间访问。只有在撤消该权限之后才能有选择地授予访问。



24.3.1.5 分配临时表空间

还应该给每个用户分配临时表空间。当用户执行一条需要临时段的 SQL 语句时, Oracle 就将该段存储在用户的临时表空间中。当做分类或联接时, 系统创建临时段, 并由 SYS 所拥有, SYS 在所有表空间都具有资源权限。

在前面的 CREATE USER 语句中, jward 的临时表空间是 temp_ts, 它被明确地创建以便仅仅包含临时段。这种表空间是用 CREATE TEMPORARY TABLESPACE 语句创建的。

如果没有明确地为用户设置临时表空间, 则给该用户分配在创建数据库时指定的默认临时表空间, 或后来用 ALTER DATABASE 语句指定的默认临时表空间。如果没有默认临时表空间, 则默认为 SYSTEM 表空间。正如前面所讨论的用户的默认表空间一样, 将用户数据存储在 SYSTEM 表空间中是不明智的。同样, 将一个表空间专门用作临时表空间, 就会减少临时段和其他类型的段之间的文件冲突。

可以在创建用户时为该用户设置临时表空间, 以后再用 ALTER USER 语句修改它。不要为临时表空间设置限额。

24.3.1.6 指定配置文件

创建用户时还可以指定配置文件。配置文件是一组对数据库资源和访问数据库的口令的限制。如果没有指定配置文件, 则给用户分配默认的配置文件。

24.3.1.7 设置默认角色

在 CREATE USER 语句中不能设置用户的默认角色。第一次创建用户时, 该用户的默认角色被设置成 ALL, 它使得以后授予给该用户的所有角色都是默认角色。用 ALTER USER 语句来修改用户的默认角色。

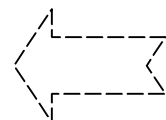
24.3.2 更改用户

用户可以修改自己的口令。但是, 要想修改用户的安全域中的其他选项, 就必须要有 ALTER USER 系统权限。通常, 仅仅只有安全管理员才具有这个系统权限, 因为它允许修改任何用户的安全域。这个权限包括为用户在数据库中的任何表空间中设置表空间限额的能力, 甚至于执行修改的用户在指定的表空间中没有限额时也行。

可以用 ALTER USER 语句更改用户的安全设置。修改用户的安全设置会影响该用户的以后的会话, 但不影响当前的会话。

如下语句更改用户 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 语句按如下方式修改 avyrros 的安全设置：

- 鉴别被改变成了使用 avyrros 的操作系统帐户。
- 明确地设置了 avyrros 的默认和临时表空间。
- 在 data_ts 表空间中为 avyrros 分配了 100M 的限额。
- 撤消了 avyrros 在 test_ts 表空间上的限额。
- 为 avyrros 分配了 clerk 配置文件。

24.3.2.1 修改用户的鉴别机制

大多数非 DBA 用户还可以用 ALTER USER 语句修改他们自己的口令，如

```
ALTER USER andy  
IDENTIFIED BY swordfish;
```

用户可用此法修改他们自己的口令，而不需任何特别的权限（不同于连接到数据库的那些操作）。应该鼓励用户经常修改他们的口令。

为了能在 Oracle 数据库鉴别和一种外部鉴别之间进行转换，用户必须要具有 ALTER USER 权限。通常，只有管理员才具有这个权限。

24.3.2.2 修改用户的默认角色

默认角色是在用户创建会话时被自动启用的角色。可以给用户分配零个或多个默认角色。

24.3.3 删除用户

删除用户时，会从数据字典中删除该用户及其相关的模式，同时，如果在该用户的模式中包含任何模式对象，也将其立即删除。

注意：如果必须保留用户的模式及其相关的对象，但又必须拒绝用户访问数据库，就撤消该用户的 CREATE SESSION 权限。

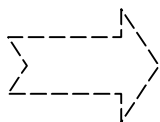
不能删除当前已连接到数据库的用户。要删除一个已连接的用户，必须先用带有 KILL SESSION 子句的 SQL 语句 ALTER SYSTEM 终止该用户的会话。

可以用 DROP USER 语句删除数据库中的用户。要想删除用户和该用户的所有模式对象（如果有），就必须要有 DROP USER 系统权限。因为 DROP USER 系统权限如此强大，所以一般仅仅是安全管理员才具有该权限。

如果用户的模式包含任何模式对象，就用 CASCADE 选项来成功地删除该用户以及所有相关的对象和依赖于该用户的表的外键。如果没有指定 CASCADE，并且用户的模式又包含对象，则返回一条错误消息，也不删除该用户。在删除其模式包含对象的用户之前，彻底地调查一下该用户的模式包含什么对象和删除它们的含意。留意任何未知的级联影响。例如，如果打算删除一个拥有表的表，请先检查一下是否有任何视图或过程依赖于那个特殊的表。

如下语句删除用户 jones 以及所有相关的对象和依赖于 jones 所拥有的表的外键。

```
DROP USER jones CASCADE;
```



24.4 用配置文件来管理资源

配置文件是一个命名的资源限制的集合。用户的配置文件会按配置文件中所定义的那样，来限制数据库使用和实例资源。可以给每个用户分配配置文件，并且给所有没有专门的配置文件的用户分配一个默认配置文件。为使配置文件起作用，必须从总体上启用数据库资源限制。

本节描述配置文件管理的几个方面，包括如下内容：

- 启用和停用资源限制
- 创建配置文件
- 分配配置文件
- 更改配置文件
- 使用组合限制
- 删除配置文件

24.4.1 启用和停用资源限制

可以为用户创建和分配配置文件。任何被授权的数据库用户都可以在任何时候更改、删除配置文件。但是，只有当对相关的数据库启用了资源限制时，才实施配置文件中的对资源限制的设置。可用两种不同的方法启用或停用实施资源限制，在下面两节中讨论这个内容。

要想在数据库保持打开时更改资源限制的实施，就必须要有 ALTER SYSTEM 系统权限。

24.4.1.1 在启动之前启用和停用资源限制

如果可以临时地关闭数据库，则可以使用在数据库初始化参数文件中的 RESOURCE_LIMIT 初始化参数，来启用或停用资源限制。该参数的有效值是 TRUE（启用实施）和 FALSE。默认时，该参数的值被设置成 FALSE。只要编辑了初始化文件，就要重新启动数据库实例以使其生效。每次启动实例，新的参数值就启用或停用资源限制的实施。

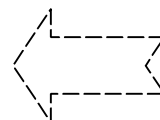
24.4.1.2 当数据库打开时启用和停用资源限制

如果不可以临时地关闭数据库，或者必须临时地更改资源限制特性，就可以用 SQL 语句 ALTER SYSTEM 来启用或停用资源限制的实施。实例启动之后，ALTER SYSTEM 语句会覆盖由 RESOURCE_LIMIT 初始化参数设置的值。例如，如下语句为数据库启用资源限制的实施：

```
ALTER SYSTEM  
SET RESOURCE_LIMIT = TRUE;
```

注意：这条语句不适用于口令参数。

ALTER SYSTEM 语句并不永久性地决定资源限制的实施。如果关闭并重新启动了该数据库，则资源限制的实施就取决于 RESOURCE_LIMIT 参数的设置值了。



24.4.2 创建配置文件

要想创建配置文件，就必须要有 `CREATE PROFILE` 系统权限。可以用 SQL 语句 `CREATE PROFILE` 来创建配置文件。同时还可以明确地设置特殊的资源限制。

如下语句创建配置文件 `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;
```

在新的配置文件中没有指定的所有资源限制，都使用 `DEFAULT` 配置文件中的限制设置。

每个数据库都有一个 `DEFAULT` 配置文件，其中的限制用于两种情况：

- 如果没有明确地给用户分配配置文件，则该用户遵守所有的 `DEFAULT` 配置文件的限制。
- 任何配置文件中的所有没有指定的限制，都使用 `DEFAULT` 配置文件中的对应的限制。

初始时，`DEFAULT` 配置文件中的所有限制都被设置成 `UNLIMITED`。但是为了防止 `DEFAULT` 配置文件的用户无限制地消耗资源，安全管理员就应该用 `ALTER PROFILE` 语句修改默认的限制：

```
ALTER PROFILE default LIMIT
...;
```

任何具有 `ALTER PROFILE` 系统权限的用户都可以调整 `DEFAULT` 配置文件中的限制。不能删除 `DEFAULT` 配置文件。

24.4.3 分配配置文件

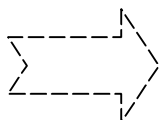
创建配置文件之后，可以将其分配给数据库用户。在任何时候都只能给每个用户分配一个配置文件。如果将一个配置文件分配给一个已经具有配置文件的用户，则新的配置文件将取代以前分配的配置文件。分配配置文件不会影响当前的会话。只能将配置文件分配给用户，而不能分配给角色或其他配置文件。

可以用 `CREATE USER` 和 `ALTER USER` 语句来给用户分配配置文件。

24.4.4 更改配置文件

可以用 SQL 语句 `ALTER PROFILE` 来更改任何配置文件中的资源限制设置。要想更改配置文件，你必须要有 `ALTER PROFILE` 系统权限。

任何调整后的配置文件中的限制都会覆盖那个配置文件中以前的限制。如果用



DEFAULT 值调整一个限制, 则该资源限制就使用用于该数据库的默认设置。当更改一个配置文件时, 所有没有被调整的配置文件仍然保留其原来的值。对配置文件的任何修改都不影响当前的会话。新的配置文件的设置只用于修改完配置文件之后才创建的会话。

如下语句更改 clerk 配置文件:

```
ALTER PROFILE clerk LIMIT
CPU_PER_CALL default
LOGICAL_READS_PER_SESSION 20000;
```

24.4.5 使用组合限制

除了可以用 CREATE 或 ALTER PROFILE 语句, 来为特殊的资源分配资源限制之外, 还可以用组合限制来限制一个会话的总的资源开销。组合限制被表示成为一个用服务单位计量的某些资源的加权和。

可以用 CREATE PROFILE 或 ALTER PROFILE 语句的 COMPOSITE_LIMIT 子句, 来设置配置文件的组合限制。如下 CREATE PROFILE 语句指定了 COMPOSITE_LIMIT 子句:

```
CREATE PROFILE clerk LIMIT
COMPOSITE_LIMIT 20000
SESSIONS_PER_USER 2
CPU_PER_CALL 1000;
```

注意, 明确的资源限制和组合限制两者可以同时存在于一个配置文件中。首先达到的限制就会终止会话中的活动。在限制对系统资源的使用时, 组合限制提供了附加的灵活性。

24.4.5.1 决定组合限制的值

正确的组合限制的值取决于大多数配置文件的用户所使用的总的资源数量。就每个特殊的资源限制而言, 应该收集历史信息, 以便为典型的配置文件用户决定正常的组合资源使用的范围。

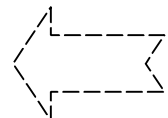
24.4.5.2 设置资源开销

每个 Oracle 数据库服务器环境都有其自身的特征。某些系统资源可能在一个环境中比在另一个环境中更有价值。Oracle 允许给下面的资源分配一个加权值, 以影响它们计入总的资源开销:

- CPU_PER_SESSION
- LOGICAL_READS_PER_SESSION
- CONNECT_TIME
- PRIVATE_SGA

如果没有给一个资源分配加权值, 则其默认加权值为 0, 即对该资源的使用不计入总的资源开销。

Oracle 这样计算总的资源开销: 首先将会话中所使用的每个资源的数量乘以该资源的加权值, 然后将所有四种资源的乘积相加。对于任何会话, 该开销是用该用户的配置文件中的 COMPOSITE_LIMIT 参数的值来限制的。乘积和总开销这两者, 是用被称为服务单位



的单位来计量的。

要想用 ALTER RESOURCE COST 语句给资源设置加权值，你必须要有 ALTER RESOURCE 系统权限。如下语句给 CPU_PER_SESSION 和 LOGICAL_READS_PER_SESSION 资源分配加权值。

```
ALTER RESOURCE COST
CPU_PER_SESSION 1
LOGICAL_READS_PER_SESSION 50;
```

这些加权值为会话建立这个开销公式：

```
cost = (1 * CPU_PER_SESSION) + (50 * LOGICAL_READS_PER_SESSION)
```

这里，CPU_PER_SESSION 和 LOGICAL_READS_PER_SESSION 的值，或者取自 DEFAULT 配置文件中的值，或者取自该会话的用户的配置文件中的值。因为在上面的语句中没有给资源 CONNECT_TIME 和 PRIVATE_SGA 分配加权值，所以这些资源没有出现在该公式中。

24.4.6 删除配置文件

要想删除配置文件，你必须要有 DROP PROFILE 系统权限。可以用 SQL 语句 DROP PROFILE 来删除配置文件。为了成功地删除当前分配给用户的配置文件，要用 CASCADE 选项。

如下语句删除配置文件 clerk，尽管已将它分配给了用户：

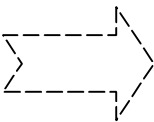
```
DROP PROFILE clerk CASCADE;
```

自动地将 DEFAULT 配置文件分配给当前使用被删除配置文件的任何用户。可以删除 DEFAULT 配置文件。删除配置文件时，该删除不会影响当前激活的会话。只有在配置文件被删除以后，才建立的会话才遵守任何修改后的配置文件的分配。

24.5 查看有关数据库用户和配置文件的信息

下面的数据字典视图包含有关数据库用户和配置文件的信息：

视图	说明
DBA_USERS	DBA 视图描述数据库中的所有用户。ALL 视图列出当前用户可以见到的用户，但不描述他们。USER 视图仅仅描述当前的用户
ALL_USERS	
USER_USERS	
DBA_TS_QUOTAS	描述用户的表空间限额
USER_TS_QUOTAS	
USER_PASSWORD_LIMITS	描述分配给该用户的口令配置文件参数



(续表)

视图	说明
USER_RESOURCE_LIMITS	显示当前用户的资源限制
DBA_PROFILES	显示所有的配置文件及其限制
RESOURCE_COST	列出每个资源的开销
V\$SESSION	列出每个当前会话的会话信息。包括用户名
V\$SESSTAT	列出用户会话的统计数据
V\$STATNAME	显示在 V\$SESSTAT 视图中显示的统计数据的解码后的统计数据名称
PROXY_USERS	描述可以呈现其他用户身份的用户

下面几节展示几个使用这些视图的例子，并假设已经在数据库中执行了下面的语句：

```
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;

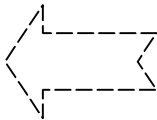
CREATE USER dcranney
IDENTIFIED BY bedrock
DEFAULT TABLESPACE users
TEMPORARY TABLESPACE temp_ts
QUOTA unlimited ON users;

CREATE USER userscott
IDENTIFIED BY scott1;
```

24.5.1 列出所有的用户及其相关的信息

如下查询列出数据库中所定义的用户及其相关的信息::

```
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
```

为了安全起见，所有的口令都被加密了。如果一个用户查询 PASS WORD 列，则该用户不可能知道另一个用户的口令。

24.5.2 列出所有的表空间限额

如下查询列出了专门分配给每个用户的表空间限额：

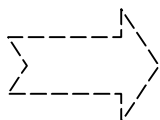
```
SELECT * FROM DBA_TS_QUOTAS;
TABLESPACE USERNAME BYTES MAX_BYTES BLOCKS MAX_BLOCKS
-----
USERS JFEE 0 512000 0 250
USERS DCRANNEY 0 -1 0 -1
```

当分配了专门的限额时，精确的数字就被表示在 MAX_BYTES 列。注意，该值总是数据库的数据块大小的一个倍数。因此，如果指定了一个不是数据库数据块大小的倍数的表空间限额，则会做相应的四舍五入。无限制的限额被表示成“-1”。

24.5.3 列出所有的配置文件及其分配的限制

如下查询列出数据库中所有的配置文件及其在每个配置文件中对每个限制的设置：

```
SELECT * FROM DBA_PROFILES
ORDER BY PROFILE;
PROFILE RESOURCE_NAME RESOURCE LIMIT
-----
CLERK COMPOSITE_LIMIT KERNEL DEFAULT
CLERK FAILED_LOGIN_ATTEMPTS PASSWORD DEFAULT
CLERK PASSWORD_LIFE_TIME PASSWORD DEFAULT
CLERK PASSWORD_REUSE_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.

```

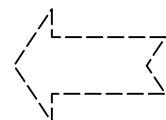
24.5.4 查看每个用户会话的内存使用

如下查询列出所有当前的会话，显示 Oracle 用户和每个会话当前使用的 UGA（用户全局区）内存的使用情况：

```

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

```

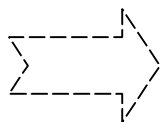


```
SYS 98196bytes
```

```
SYSTEM 30648bytes
```

```
11 rows selected.
```

若要查看启动实例以来曾经分配给每个会话的最大的 UGA 内存，就用'session uga memory max'代替上面查询中的'session uga memory'。



第 25 章 管理用户的权限和角色

本章说明如何使用权限和角色来控制对模式对象的访问，以及如何控制执行系统操作的能力。将讨论以下内容：

- 标识用户的权限
- 管理用户的角色
- 授予用户权限和角色
- 撤消用户的权限和角色
- 授予和撤消行为何时生效
- 用操作系统或网络来授予角色
- 查看权限和角色信息

参见：

- 第 23 章的“建立安全策略”中所建议的常用的数据库安全策略
- 第 24 章的“管理用户和资源”中的有关控制对数据库的访问的信息

25.1 标识用户的权限

本节描述了 Oracle 的用户权限，包括如下内容：

- 系统权限
- 对象权限

用户权限是执行特殊类型的 SQL 语句，或访问其他用户对象的一种权力。Oracle 也提供了一种将经常一起授予或撤消的权限组合在一起的捷径。

25.1.1 系统权限

有一百多种不同的系统权限。每种系统权限都允许用户执行某一个特殊的数据库操作或某一类数据库操作。

警告：系统权限可能非常强大，只应该在需要时才授予数据库的角色和受信任的用户。

25.1.1.1 系统权限的限制

由于系统权限太强大了，所以 Oracle 建议：对数据库进行配置，以防止一般用户（非 DBA）在数据字典上使用 ANY 系统权限(如 UPDATE ANY TABLE)。为了保证数据字典的安全，应确保将 O7_DICTIONARY_ACCESSIBILITY 初始化参数设置成 FALSE。这种特性被称为字典保护机制。

注意：当你从 Oracle7 升级为 Oracle8i 或更高版本时，O7_DICTIONARY_ACCESSIBILITY 初始化参数就会对系统权限进行控制了。如果该参数被设置成 TRUE，则允许访问在 SYS 模式中的对象（Oracle7 的特性）。如果该参数被设置成 FALSE，那么，允许访问在“任何模式”中的对象的系统权限，将不再允许访问在 SYS 模式中的对象了。O7_DICTIONARY_ACCESSIBILITY 的默认值是 FALSE。

当该参数没有被设置成 FALSE 时, ANY 权限适用于数据字典, 具有 ANY 权限的恶意用户就可以访问或改变数据字典表了。

如果启用字典保护 (O7_DICTIONARY_ACCESSIBILITY 被设置成 FALSE), 则只有拥有 SYS 模式的用户才能访问在 SYS 模式 (字典对象) 中的对象。这些用户是 SYS 和那些按 SYSDAB 进行连接的用户。可以提供访问其他模式中的对象的系统权限, 并不提供访问在 SYS 模式中的对象的权限。例如, SELECT ANY TABLE 权限允许用户访问在其他模式中的视图和表, 但并不允许他们选取字典对象 (动态性能视图的基础表、视图、包、同义词)。然而, 却可以授予这些用户明确的对象权限, 使其访问在 SYS 模式中的对象。

25.1.1.2 访问在 SYS 模式中的对象

具有明确的对象权限或具有管理权限 (SYSDBA) 的用户, 可以访问在 SYS 模式中的对象。允许访问在 SYS 模式中的对象的另一种方法是, 给用户授予如下角色:

- SELECT_CATALOG_ROLE

可以将该权限授予用户, 以便使其具有选取所有数据字典视图的 SELECT 权限。

- EXECUTE_CATALOG_ROLE

可以将该权限授予用户, 以便使其具有执行数据字典中的包和过程的 EXECUTE 权限。

- DELETE_CATALOG_ROLE

可以将该权限授予用户, 以便使其能够删除系统审计表 (AUD\$) 中的记录。

另外, 若用户需要访问在 SYS 模式中创建的表, 就要给他授予下面的权限:

- SELECT ANY DICTIONARY

该系统权限允许查询在 SYS 模式中的任何对象, 包括在此模式中创建的表。必须单独将该权限授予需要该权限的用户。该权限没有被包含在 GRANT ALL PRIVILEGES 中, 也不能通过角色来授予该权限。

警告: 在授予这些权限和 SELECT ANY DICTIONARY 系统权限时应该特别小心, 因为错误地使用他们, 就会危及系统的完整性。

25.1.2 对象权限

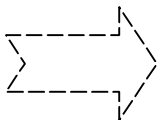
每种对象都有与之相关的不同的权限。

为了授予或撤消针对一个对象的所有可用的对象权限, 可以指定 ALL[PRIVILEGES]。ALL 不是一个权限, 而只是一种捷径, 或是可以在 GRANT 或 REVOKE 语句中用一个词来授予或撤消所有对象权限的一个方法。注意, 如果所有对象权限是用 ALL 捷径来授予的, 仍然可以撤消单个的权限。

同样, 通过指定 ALL, 可以撤消所有单独授予的权限。然而, 如果你 REVOKE ALL, 该撤消行为会删除完整性约束 (因为完整性约束依赖于正在撤消的 REFERENCES 权限), 所以必须在 REVOKE 语句中包含 CASCADE CONSTRAINTS 选项。

25.2 管理用户的角色

一个角色可以组合多个权限和角色, 以便将它们同时授予用户或从用户那里撤消。在



用户可以使用角色之前，必须为该用户启用该角色。

本节描述了管理角色的几个方面，包括如下内容：

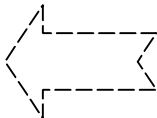
- 预定义的角色
- 创建角色
- 指定角色授权的类型
- 删除角色

25. 2. 1 预定义的角色

当运行作为数据库创建的一部分的标准脚本时，会自动为 Oracle 数据库定义在表 25-1 中所列出来的角色。你可以按照处理你所定义的任何角色的方法，来给这些预定义的角色授予或撤消权限和角色。

表 25-1 预定义的角色

角色名	创建者（脚本）	说明
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 WITH ADMIN OPTION 的系统权限
注意：前三个角色用来保持与 Oracle 的较早版本的兼容性，可能不会在 Oracle 的未来版本中自动创建了。Oracle 公司建议，为数据库的安全而设计自己的角色，而不要依赖于这些角色。		
EXP_FULL_DATABASE	CATEXP.SQL	提供执行完整的和增量的数据库输出所需要的 权限 包括下列权限：SELECT ANY TABLE， BACKUP ANY TABLE， EXECUTE ANY PROCEDURE， EXECUTE ANY TYPE， ADMINISTER RESOURCE MANAGER，和对表 SYS.INCVID， SYS.INCFIL， SYS.INCEXP 进行 的 INSERT， DELETE 和 UPDATE。也包含下面 的 角 色 ： EXECUTE_CATALOG_ROLE 和 SELECT_CATALOG_ROLE。
IMP_FULL_DATABASE	CATEXP.SQL	提供执行完整的数据库输入所需要的权限，包括 一个系统权限的扩展表（使用视图 DBA_SYS_PRIVS 来查询这些权限）和如下角 色 ： EXECUTE_CATALOG_ROLE 和 SELECT_CATALOG_ROLE。



(续表)

角色名	创建者 (脚本)	说明
DELETE_CATALOG_ROLE	SQL.BSQ	提供对系统审计表 (AUD\$) 的 DELETE 权限
EXECUTE_CATALOG_ROLE	SQL.BSQ	提供执行在数据字典中的对象的 EXECUTE 权限, 还包括 HS_ADMIN_ROLE
SELECT_CATALOG_ROLE	SQL.BSQ	提供选取在数据字典中的对象的 SELECT 权限, 还包括 HS_ADMIN_ROLE
RECOVERY_CATALOG_OWNER	CATALOG.SQL	为恢复目录的拥有者提供权限, 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_ROLE	CATHS.SQL	用来防止对 HS (异构服务) 数据字典表 (授予 SELECT) 和包 (授予 EXECUTE) 的访问。它被授予 SELECT_CATALOG_ROLE 角色和 EXECUTE_CATALOG_ROLE 角色, 使得有一般数据字典访问权限的用户都可以访问 HS 数据字典。
AQ_USER_ROLE	CATQUEUE.SQL	已过时了, 但主要被用于 8.0 版本的兼容性。提供执行 DBMS_AQ 和 DBMS_AQIN 的权限。
AQ_ADMINISTRATOR_ROLE	CATQUEUE.SQL	为管理员高级排队提供权限, 包括 ENQUEUE ANY QUEUE, DEQUEUE ANY QUEUE, 和 MANAGE ANY QUEUE, 选取 AQ 表的 SELECT 权限和执行 AQ 包的 EXECUTE 权限
SNMPAGENT	CATSNMP.SQL	企业管理器/智能代理使用这个角色。包括 ANALYZE ANY 和 授予选取各种视图的 SELECT 权限

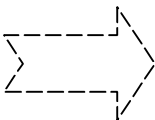
如果安装了其他选项和产品, 可能还会创建其他预定义的角色。

25.2.2 创建角色

可以用 CREATE ROLE 语句来创建角色, 但在这么做之前必须要具有 CREATE ROLE 系统权限。一般来说, 只有安全管理员才具有这个系统权限。

注意: 刚刚创建的角色没有任何相关的权限。为使一个新角色与权限相关联, 必须给该新角色授予权限或其他角色。

必须在数据库现有的用户名和角色名中, 为你所创建的每一个角色起一个唯一的名字。任何用户的模式中都不包含角色。在使用多字节字符集的数据库中, Oracle 建议, 每个角色名至少包含一个单字节的字符。如果一个角色名只包含多字节的字符, 则加密后的用户名/口令的组合的安全性就会低一些。



如下语句创建 `clerk` 角色，数据库使用口令 `bicentennial` 来授权他：

```
CREATE ROLE clerk IDENTIFIED BY bicentennial;
```

`IDENTIFIED BY` 子句说明了，在为一个已经被授予该角色的特殊的用户启用该角色之前，是如何授权该用户的。如果没有指定这个子句，或指定了 `NOT IDENTIFIED`，则启用该角色时就不需要授权了。可以指定用如下方式来授权角色：

- 使用口令的数据库
- 使用一个指定的包的应用程序
- 通过操作系统、网络或其他外部资源进行外部授权
- 通过企业目录服务进行全局授权

在如下几节中讨论这些授权方法。

以后，可以用 `ALTER ROLE` 语句来为角色设置或改变授权的方法。如下语句更改 `clerk` 角色并指出：在启用该角色之前，必须用一个外部资源来授权用户。

```
ALTER ROLE clerk IDENTIFIED EXTERNALLY;
```

为了更改角色的授权方法，需要具有 `ALTER ANY ROLE` 系统权限或已经被授予了具有 `ADMIN OPTION` 权限的角色。

25.2.3 指定角色授权的类型

本节提供了授权角色的方法。在使用角色之前必须先启用该角色。

25.2.3.1 通过数据库授权角色

通过数据库授权的角色可以用相关的口令来保护。如果给你授予一个用口令保护的角色，你就可以通过在 `SET ROLE` 语句中，为该角色提供正确的口令，来启用或停用该角色。当然，如果该角色是默认的了，并且在连接时启用了该角色，那用户就不需要再输入口令了。

如下语句创建角色 `manager`。当启用该角色时，必须提供口令 `morework`。

```
CREATE ROLE manager IDENTIFIED BY morework;
```

注意：在使用多字节字符集的数据库中，角色的口令只能包含单字节的字符。口令中不允许有多字节的字符。

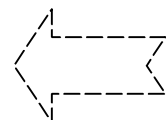
25.2.3.2 通过应用程序授权角色

`IDENTIFIED USING package_name` 子句允许你创建应用角色，它是只能由使用授权包的应用才能启用的角色。应用开发商不需要通过应用嵌入口令来保护角色的安全。相反，他们可以创建应用角色，并指定哪个 `PL/SQL` 包被授权用于启用该角色。

如下例子表明，角色 `admin_role` 是一个应用角色，只有在 `PL/SQL` 包 `hr.admin` 中定义的模块才能启用该角色。

```
CREATE ROLE admin_role IDENTIFIED USING hr.admin;
```

在登录时启用用户的默认角色（在该用户的配置文件中指定）时，不对应用角色做任何检查。



25.2.3.3 通过外部资源授权角色

如下语句创建一个名为 `accts_rec` 的用户，在启用该角色之前，需要外部资源授权用户：

```
CREATE ROLE accts_rec IDENTIFIED EXTERNALLY;
```

25.2.3.4 通过操作系统授权角色

只有在操作系统能够将操作系统权限与应用进行动态的链接的情况下，通过操作系统的角色授权才有用。当一个用户启动一个应用时，操作系统就会将操作系统权限授予该用户。被授予的操作系统权限对应于与该应用相关的角色。这样，该应用就能启用该应用角色。当该应用被终止时，就会从该用户的操作系统帐户中撤消以前授予给该用户的操作系统权限。

如果通过操作系统来授权角色，就必须在操作系统一级上为每一个用户配置信息。这个操作依赖于操作系统。

如果通过操作系统来授予角色，那就不必再让操作系统对其进行授权了，这是多余的。

25.2.3.5 角色授权和网络客户

如果用户是通过 Oracle Net 连接到数据库的，则在默认情况下，他们的角色不能由操作系统进行授权了。这包括通过共享服务器配置进行的连接，因为这种连接需要 Oracle Net。因为一个远程用户可能通过一个网络连接来模仿其他操作系统用户，所以这种限制应该是默认的。

如果不在乎这种安全风险，只是想对网络客户进行操作系统角色授权，就把数据库的初始化参数文件中的初始化参数 `REMOTE_OS_ROLES` 设置成 `TRUE`。在下次启动实例和加载数据库时，这种改变才会生效。该参数的默认值是 `FALSE`。

25.2.3.6 通过企业目录服务授权角色

可以将角色定义为全局角色，据此，（全局）用户只能由企业目录服务来授权使用该角色。通过将权限和角色授予一个角色，你就可以在数据库中定义本地全局角色，但是你不能将该全局角色本身授予在数据库中的任何用户或其他角色。当一个全局用户试图连接到数据库时，会查询企业目录以便获得与该用户相关的任何全局角色：

如下语句创建一个全局角色：

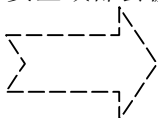
```
CREATE ROLE supervisor IDENTIFIED GLOBALLY
```

全局角色是企业用户管理的一个组件。全局角色只适用于一个数据库，但可以将它授予在企业目录中定义的一个企业角色。一个企业角色是包含多个数据库上的全局角色的一个目录结构，可以将它授予企业用户。

早在第 24 章的“全局鉴别和授权”中，就对用户的全局鉴别和授权，以及用户在企业用户管理中的角色进行了一般的讨论。

25.2.4 删除角色

在某些情况下，要删除数据库中的角色。所有被授予了该被删除角色的用户和角色的安全域都会被立即改变，以反映该被删除角色的权限的终止。所有间接授予该被删除角色



的角色也会被从受影响的安全域中删除。删除一个角色时会自动地从所有用户的默认角色列表中删除该角色。

因为创建对象时并不依赖于通过角色获得的权限，所以在删除角色时，也就不会删除表和其他对象。

可以用 SQL 语句 `DROP ROLE` 来删除角色。要想删除角色，就需要具有 `DROP ANY ROLE` 系统权限，或已经被授予了带有 `ADMIN OPTION` 的角色。

如下语句删除角色 `clerk`：

```
DROP ROLE clerk;
```

25.3 给用户授予权限和角色

本节描述授予权限和角色的几个方面，包括如下内容：

- 授予系统权限和角色
- 授予对象权限和角色
- 授予列上的权限

还可以将角色授予给通过中间层或代理而连接的用户。这在第 24 章的“代理鉴别和授权”中已讨论了。

25.3.1 授予系统权限和角色

可以用 SQL 语句 `GRANT` 将系统权限和角色授予其他角色和用户。要想授予系统权限和角色，必须要具有打算授予的所有系统权限和角色的 `ADMIN OPTION` 权限。另外，具有 `GRANT ANY ROLE` 系统权限的任何用户，都可以授予数据库中的任何角色。

注意：不能给带有 `IDENTIFIED GLOBALLY` 的角色授予任何权限和角色。全局角色的授予和撤消完全是由企业目录服务控制的。

如下语句将系统权限 `CREATE SESSION` 和 `accts_pay` 角色授予用户 `jward`：

```
GRANT CREATE SESSION, accts_pay
TO jward;
```

注意：在同一个 `GRANT` 语句中，不能将系统权限和对象权限一起授予。

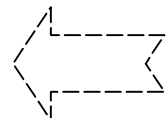
当用户创建角色时，该角色会被自动地授予给具有 `ADMIN OPTION` 权限的该创建者。一个具有 `ADMIN OPTION` 权限的被授予者有几种扩展的能力：

- 被授予者能够将系统权限或角色授予给数据库中的任何用户或其他角色，或从中撤消他们。用户不能撤消自身的角色。
- 被授予者可以进一步地授予具有 `ADMIN OPTION` 权限的系统权限或角色。
- 角色的被授予者可以更改或删除该角色。

在如下语句中，安全管理员将 `new_dba` 角色授予 `michael`：

```
GRANT new_dba TO michael WITH ADMIN OPTION;
```

用户 `michael` 不仅可以使使用包含在 `new_dba` 角色中的所有权限，需要时，他还可以授予、撤消、删除该 `new_dba` 角色。由于这些强大的功能，所以，当授予具有 `ADMIN OPTION` 权限的系统权限或角色时就应特别小心。这些权限通常是安全管理员而保留的，很少被



授予给其他管理员或系统用户。

25.3.2 授予对象权限和角色

还可以用 GRANT 语句将对象权限授予给角色或用户。要想授予对象权限，必须满足如下条件之一：

- 拥有被指定的对象
- 已经被授予了具有 GRANT OPTION 权限的对象权限

注意：在同一个 GRANT 语句中，不能将系统权限和对象权限一起授予。

如下语句把针对 emp 表的所有列的 SELECT, INSERT 和 DELETE 对象权限授予用户 jfee 和 tsmith。

```
GRANT SELECT, INSERT, DELETE ON emp TO jfee, tsmith;
```

为了只把针对 emp 表中的 ename 和 job 列的 INSERT 对象权限授予用户 jfee 和 tsmith，可发布如下语句：

```
GRANT INSERT(ename, job) ON emp TO jfee, tsmith;
```

为了把针对 salary 视图的所有对象权限授予用户 jfee，使用 ALL 关键字，如下面例子所示：

```
GRANT ALL ON salary TO jfee;
```

其模式中包含对象的用户被自动授予具有 GRANT OPTION 权限的所有相关的对象权限。这种特殊的权限允许被授予者具有几种扩展的权限：

- 被授予者可以将对象权限授予数据库中的任何用户（无论是否具有 GRANT OPTION 权限），或数据库中的任何角色。
- 如果下面两项都为真，被授予者就可以在表上创建视图，并将该视图上的相关权限授予数据库的任何用户和角色。
 - 被授予者接受了该表的具有 GRANT OPTION 权限的对象权限。
 - 被授予者具有 CREATE VIEW 或 CREATE ANY VIEW 系统权限。

当将一个对象权限授予一个角色时，GRANT OPTION 权限是无效的。Oracle 会防止通过角色来传播对象权限，因此，一个角色的被授予者就不能传播通过角色获得的对象权限。

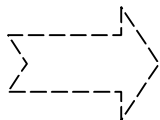
25.3.3 授予列上的权限

可以授予在表中的单个列上的 INSERT, UPDATE 或 REFERENCES 权限。

警告：在授予一个列特殊的 INSERT 权限之前，要先确定该表是否包含定义过 NOT NULL 约束关系的列。授予选择性的不包含 NOT NULL 列的插入能力就会使用户不能在表中插入任何行。为了避免这种情况的发生，应确保每一个 NOT NULL 列要么是可插入的，要么具有非空的默认值。否则，被授予者就不能在表中插入行，这么做就会收到出错消息。

将 accounts 表中的 acct_no 列的 INSERT 权限授予 scott:

```
GRANT INSERT (acct_no)
ON accounts TO scott;
```



25.4 撤消用户的权限和角色

本节描述撤消用户的权限和角色的几个方面，包括如下内容：

- 撤消系统权限和角色
- 撤消对象权限和角色
- 撤消权限的连带效应
- 授予和撤消用户组 PUBLIC 中的权限

25.4.1 撤消系统权限和角色

可以使用 SQL 语句 REVOKE 来撤消系统权限和角色

具有 ADMIN OPTION 权限的系统权限或角色的用户，可以撤消其他数据库用户或角色的权限和角色。撤消者不必是最初授予该权限或角色的用户。另外，具有 GRANT ANY ROLE 权限的用户可以撤消任意角色。

如下语句从 tsmith 撤消 CREATE TABLE 系统权限和 accts_rec 角色：

```
REVOKE CREATE TABLE, accts_rec FROM tsmith;
```

注意：不能选择性地撤消系统权限或角色的 ADMIN OPTION 权限。只能先撤消该权限和角色，然后再重新授予不具有 ADMIN OPTION 权限的该权限和角色。

25.4.2 撤消对象权限和角色

用 REVOKE 语句来撤消对象权限。要想撤消对象权限，撤消者必须是该被撤消对象权限的初始授予者。

例如，假设你是初始授予者，为了撤消用户 jfee 和 tsmith 的 SELECT(选取)和 INSERT(插入) emp 表的权限，应发布如下语句：

```
REVOKE SELECT, insert ON emp
FROM jfee, tsmith;
```

如下语句撤消对表 dept 的所有权限（最初被授予角色 human_resource 的权限）：

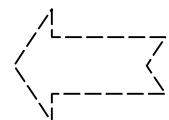
```
REVOKE ALL ON dept FROM human_resources;
```

注意：上述语句只会撤消授予者所授予的权限，不包括其他用户所授予的权限。不能选择性地撤消一个对象权限的 GRANT OPTION 权限。只能先撤消该对象权限，然后再重新授予不具有 GRANT OPTION 权限的该对象权限。用户不能撤消自身的对象权限。

25.4.2.1 撤消列选择性的对象权限

尽管用户可以授予针对表和视图的列选择性的 INSERT, UPDATE 和 REFERENCES 权限，但却不能用一条简单的 REVOKE 语句来选择性地撤消列特殊的权限。相反，授予者必须首先撤消表或视图的所有列的对象权限，然后再选择性地重新授予应该保留的列特殊的权限。

例如，假设已经给角色 human_resources 授予了在 dept 表的 deptno 和 dname 列上进行 UPDATE（更新）的权限。为了只撤消对 deptno 列的 UPDATE 权限，就要发布如下两条语句：



```
REVOKE UPDATE ON dept FROM human_resources;  
GRANT UPDATE (dname) ON dept TO human_resources;
```

该 REVOKE 语句撤销角色 human_resources 对 dept 表的所有列的 UPDATE 权限。而该 GRANT 语句将对 dname 列的 UPDATE 权限再次授予角色 human_resources。

25.4.2.2 撤销 REFERENCES 对象权限

如果 REFERENCES 对象权限的被授予者已经使用该权限创建了外键约束（当前已存在），那么授予者可以只在 REVOKE 语句中指定 CASCADE CONSTRAINTS 选项来撤销该权限：

```
REVOKE REFERENCES ON dept FROM jward CASCADE CONSTRAINTS;
```

当指定了 CASCADE CONSTRAINTS 子句时，就会删除当前被定义的、使用被撤销的 REFERENCES 权限的所有外键约束。

25.4.3 撤销权限的连带效应

根据权限的类型，当撤销权限时，可能会发生连带效应。

25.4.3.1 系统权限

当撤销与 DDL 操作相关的系统权限时，那么，无论该权限是否被授予了 ADMIN OPTION 权限，都不会发生连带效应，例如，假设：

①安全管理员将 CREATE TABLE 系统权限授予具有 ADMIN OPTION 权限的 jfee。

②jfee 创建了一个表。

③jfee 将 CREATE TABLE 系统权限授予 tsmith。

④tsmith 创建了一个表。

⑤安全管理员撤销了 jfee 的 CREATE TABLE 系统权限。

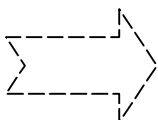
⑥jfee 的表依然存在，tsmith 的表也存在，而且 tsmith 依然拥有 CREATE TABLE 系统权限。

当撤销与 DML 操作相关的系统权限时，会出现连带效应。如果撤销一个用户的 SELECT ANY TABLE 权限，那么依赖于该权限的、在该用户的模式中的过程就会失败，直到给该用户重新授予该权限。

25.4.3.2 对象权限

撤销一个对象权限会发生连带效应，因此在发布 REVOKE 命令前，应对可能的连带效应加以分析。

- 如果撤销 DML 对象权限，就会影响到依赖于该 DML 对象权限的对象定义。例如，假设 test 过程的过程体含有从 emp 表中查询数据的 SQL 语句。如果撤销了 test 程序的拥有者在 emp 表上的 SELECT 权限，就不能成功地执行该程序了。
- 当撤销一个用户的对一个表的 REFERENCES 权限时，就会自动删除该用户定义的、需要该被删除的 REFERENCES 权限的所有外键完整性约束。例如，假设用户 jward 被授予了在 dept 表中的 deptno 列上的 REFERENCES 权限，并且在 emp 表中



的 deptno 列上创建了一个参照 deptno 列的外键。如果撤消在 dept 表中的 deptno 列上的 REFERENCES 权限，那么，这个操作还会删除 emp 表中的 deptno 列上的外键约束。

- 如果撤消授予者的对象权限，就会撤消使用 GRANT OPTION 权限传播的对象权限。例如，假设 user1 被授予了具有 GRANT OPTION 权限的 SELECT 对象权限，同时将 emp 上的 SELECT 权限授予了 user2。然后，撤消 user1 的 SELECT 权限。那么，这个 REVOKE（撤消）操作还会牵连到 user2。任何依赖于 user1 的和 user2 的、被撤消的 SELECT 权限的对象都会受到影响，就象前面条款所描述的一样。

撤消 ALTER 或者 INDEX 对象权限，不会影响到需要 ALTER 和 INDEX 的 DDL 对象权限的对象定义。例如，如果撤消一个用户的 INDEX 权限（该用户已经在其他人的表中创建了一个索引），那么，当撤消该用户的该权限时，该索引依旧存在。

25.4.4 授予和撤消用户组 PUBLIC 中的权限

也可以授予和撤消用户组 PUBLIC 中的权限与角色。因为每个数据库用户都可以访问 PUBLIC，所以每个数据库用户都可以获得授予 PUBLIC 的所有权限与角色。

只有在每个数据库用户都需要某个权限或角色的条件下，安全管理员与数据库用户才应当将该权限或角色授予 PUBLIC。这个建议加强了这样一个常规的规则：在任何时候，每个数据库用户只应该拥有能顺利完成群体的当前工作所需的权限。

撤消 PUBLIC 中的权限会造成巨大的连带效应。如果撤消 PUBLIC 中的、与 DML 操作相关的权限（如，在 emp 上的 SELECT ANY TABLE、UPDATE 等），那么，在该数据库中的所有过程（包括函数和包），必须被重新授权后才能被重新使用。因此，当授予或撤消 PUBLIC 中的、与 DML 相关的权限时必须特别小心。

25.5 授予和撤消行为何时生效

这依赖于授予或撤消的是什么权限，授予或撤消会在不同的时间生效。

- 可以立即察觉到授予/撤消任何对象（用户，角色和 PUBLIC）的系统权限和对象权限。
- 只有在授予/撤消之后，当前用户会话发布了 SET ROLE 语句来重新启用该角色，或创建了新的用户会话之时，才能察觉到授予/撤消任何对象（用户，角色和 PUBLIC）的角色。

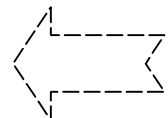
通过检查 SESSION_ROLES 数据字典视图，可以看到当前启用了哪些角色。

25.5.1 SET ROLE 语句

在会话期间，用户或应用能任意次的使用 SET ROLE 语句，以改变用于该会话的当前被启用的角色。在 SET ROLE 语句指定的角色必须是已经授予给你的角色。可以同时启用的角色的数量是由初始化参数 MAX_ENABLED_ROLES 来限定的。

本例启用已经授予给你的角色 clerk，并指定了口令。

```
SET ROLE clerk IDENTIFIED BY bicentennial;
```



可以用如下语句来停用所有角色：

```
SET ROLE NONE;
```

25.5.2 指定默认角色

当用户登录时，Oracle 会启用所有明确地授予该用户的权限，并启用该用户的默认角色的所有权限。

可以用 ALTER USER 语句来设置和改变一个用户的默认角色的列表。允许在 ALTER USER 语句中指定那些在用户连接到数据库时就启用的、不需要用户指定该角色的口令的角色。必须已经用 GRANT 语句将这些角色直接地授予给了该用户。不能将外部服务（包括目录服务）管理的任何角色（外部角色或全局角色）指定为默认角色。

如下例子为用户 jane 建立默认角色：

```
ALTER USER jane DEFAULT ROLE payclerk, pettycash;
```

不能在 CREATE USER 语句中设置用户的默认角色。当你第一次创建一个用户时，该用户的默认角色的设置值是 ALL，这使得以后授予给该用户的角色都是默认角色。使用 ALTER USER 语句来限制用户的默认角色。

警告：当你创建一个角色时（除了用户角色），其实已经隐含地将该角色授予给你了，并且是一个默认角色。如果你的角色数量多于 MAX_ENABLED_ROLES，那么，你在登录时就会收到一条出错消息。可通过更改该用户的默认角色的数量，使之少于 MAX_ENABLED_ROLES 来避免这个错误。因此，应该在创建用户角色前，先改变 SYS 和 SYSTEM 的 DEFAULT ROLE 的设置值。

25.5.3 限制一个用户可以启用的角色数量

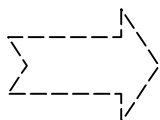
一个用户可以启用在初始化参数 MAX_ENABLED_ROLES 中所定义那么多个角色。这个数目包括启用直接角色而被启用的间接角色的数目在内。数据库管理员可以通过修改这个参数的值来改变这个限制。这个值越大，就允许每个用户会话可以同时启用更多的角色。然而，这个参数的值越大，每个用户会话就需要更多的存储空间；这是因为，PGA 的大小会受到每个用户会话的影响，每个角色需要四个字节。确定可能被任何用户同时启用的角色的最大数量，然后把这个值作为 MAX_ENABLED_ROLES 参数的值。

25.6 用操作系统或网络来授予角色

本节描述通过操作系统或网络来授予角色的几个方面，包括如下内容：

- 使用操作系统的角色识别
- 使用操作系统的角色管理
- 当 OS_ROLE = TRUE 时授予和撤消角色
- 当 OS_ROLES = TRUE 时启用和停用角色
- 使用具有操作系统角色管理的网络连接

与安全管理员使用 GRANT 和 REVOKE 语句来授予和撤消用户的数据库角色所不同，操作 Oracle 的操作系统会在连接时把角色授予用户。当一个用户创建一个会话时，操作系



统可以管理角色，并把角色传递给 Oracle。作为这种机制的一部分，可以识别每个用户的默认角色和授予给该用户的具有 ADMIN OPTION 权限的角色。尽管操作系统被用来将角色授权给用户，但所有的角色必须在数据库中创建，并且是用 GRANT 语句将权限授予给角色的。

还可以用网络服务来授予角色。

用操作系统来识别用户的数据库角色的好处是，Oracle 数据库的权限管理能够被具体化。操作系统提供的安全工具会控制用户的权限。这个选择权可以为许多系统行为提供集中的安全管理的好处，就像以下情况那样：

- MVS Oracle 管理员希望用 RACF 组来标识数据库用户的角色时。
- UNIX Oracle 管理员希望用 UNIX 组来标识数据库用户的角色时。
- VMS Oracle 管理员希望用权限标识符来标识数据库用户的角色时。

用操作系统来识别用户的数据库角色的主要缺点是，只能执行角色一级的权限管理。不能用操作系统来授予单个的权限，而只能用数据库中的 GRANT 语句来授予单个的权限。

使用这个特性的第二个缺点是，如果用操作系统来管理角色的话，默认情况下，用户就不能通过共享服务器连接到数据库，或连接到其他的网络连接。然而，可以改变这种默认情况。参见第 25 章的“使用具有操作系统角色管理的网络连接”。

注意：本节描述的这些特性只适用于某些操作系统。参见你的操作系统特殊的 Oracle 文档以确定是否你可以使用这些特性。

25.6.1 使用操作系统的角色标识

为了在创建会话时，让数据库使用操作系统来识别每个用户的数据库角色，就要把初始化参数 OS_ROLES 设置成 TRUE(如果实例正在运行，就应重新启动该实例)。当用户试图创建与数据库的会话时，Oracle 就用操作系统标识的数据库角色来初始化该用户的安全域。

为了标识用户的数据库角色，每个 Oracle 用户的操作系统帐户就必须要有有一个操作系统标识符（可以被称为组权限标识符，或其他相似的名字）。该标识符指出该用户可以使用哪些数据库角色。角色说明还可以指出哪些角色是用户的默认角色，哪些角色可用于 ADMIN OPTION 权限。无论使用什么样的操作系统，操作系统层次的角色说明的格式如下：

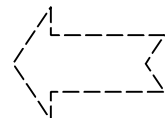
```
ora_ID_ROLE[_[d][a]]
```

其中：

- 不同操作系统上的 ID 定义也不同。例如，在 VMS 上，ID 是数据库的实例标识符；在 MVS 上，是机器类型；在 UNIX 上，是系统 ID。

注意：为了与 ORACLE_SID 相匹配，所以 ID 是大小写敏感的，而 ROLE 不是大小写敏感的。

- ROLE 是数据库角色的名字。
 - d 是一个可选字符，它表示这个角色是数据库用户的默认角色。
 - a 是一个可选字符，它表示将这个角色授予用户时会具有 ADMIN OPTION 权限。
- 这就允许该用户将该角色授予其他角色。如果用操作系统来管理角色，就不能将角



色授予用户。

注意：如果要指定 d 或 a 字符，就要在其前面加下划线。

例如，一个操作系统帐户可能具有如下的、在其配置文件中标识的角色：

```
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 是默认的角色，而 role2 和 role4 具有 ADMIN OPTION 权限。

25.6.2 使用操作系统的角色管理

当使用操作系统来管理角色时，要注意，数据库角色被授予给了操作系统用户。OS 用户能够连接到的数据库用户，都会启用被授权的数据库角色。因此，如果 OS_ROLES = TRUE 时，就应该考虑将所有的 Oracle 用户定义成 IDENTIFIED EXTERNALLY(外部识别)，以便数据库帐户与已经被授予了权限的 OS 帐户相关联。

25.6.3 当 OS_ROLES = TRUE 时授予和撤消权限

如果 OS_ROLES 被设置成 TRUE，那么操作系统就会完全管理给用户授予或撤消角色。不能再用以前的 GRANT 语句来给用户授予角色了；但仍然将它们放在数据字典中。只能在操作系统层次上将角色授予用户。用户仍可以将权限授予角色和用户。

注意：如果操作系统将一个角色连同 ADMIN OPTION 权限授予用户，那么，该用户就只能将该角色授予其他角色。

25.6.4 当 OS_ROLES = TRUE 时启用和停用角色

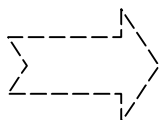
如果 OS_ROLES 被设置成 TRUE，就可以用 SET ROLE 语句来动态地启用操作系统授予的任何角色。即使该角色需要口令或需要操作系统授权时，仍然可以这么做。然而，没有在用户的操作系统帐户中标识的任何角色，都不能在 SET ROLE 语句中指定，即使当 OS_ROLES = FALSE 时已经用 GRANT 语句授予了该角色。（如果指定了这样一个角色，Oracle 就会忽略它）。

当 OS_ROLES = TRUE 时，用户可以启用在初始化参数 MAX_ENABLED_ROLES 中指定的那么多的角色。

25.6.5 使用具有操作系统角色管理的网络连接

如果选择让操作系统来管理角色，那么在默认情况下，用户是不能通过共享服务器来连接到数据库的。因为一个远程用户可以在非安全的连接的基础上，模仿其他操作系统用户，所以这种限制是默认的。

如果你不在乎这个安全隐患，并且想使用带有共享服务器的操作系统角色管理，或者使用其他网络连接，那么，你可将该数据库的初始化参数文件中的 REMOTE_OS_ROLES 初始化参数设置成 TRUE。当你下一次启动实例和装载该数据库时，这个改变就会生效。



该参数的默认值是 FALSE。

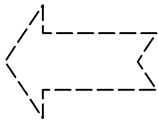
25.7 查看权限与角色的信息

如果想查看有关权限和角色的授予的信息，可以查询如下数据字典视图：

视图	说明
DAB_COL_PRIVS	DBA 视图描述了数据库中的所有的列对象的授权。ALL 视图描述了当前用户或 PUBLIC 是其所有者、授予者或被授予者的，所有的列对象的授权。USER 视图描述了当前用户是其所有者、授予者或被授予者的，所有的列对象的授权。
ALL_COL_PRIVS	
USER_COL_PRIVS	
ALL_COL_PRIVS_MADE	ALL 视图列出了当前用户是其所有者或授予者的，所有的列对象的授权。USER 视图描述了当前用户是其授予者的所有的列对象的授权。
USER_COL_PRIVS_MADE	
ALL_COL_PRIVS_RECD	ALL 视图描述了当前用户或 PUBLIC 是其被授予者的所有的列对象的授权。USER 视图描述了当前用户是其被授予者的所有的列对象的授权。
USER_COL_PRIVS_RECD	
DBA_TAB_PRIVS	DBA 视图列出了数据库中所有的对象的授权。ALL 视图列出了用户或 PUBLIC 是其授予者的对象的授权。USER 视图列出当前用户是其被授予者的所有的对象的授权。
ALL_TAB_PRIVS	
USER_TAB_PRIVS	
ALL_TAB_PRIVS_MADE	ALL 视图列出了当前用户所做的所有的对象授权或者是在当前用户所拥有的对象上的对象授权。USER 视图列出了在当前用户所拥有的所有对象上的授权。
USER_TAB_PRIVS_MADE	
ALL_TAB_PRIVS_RECD	ALL 视图列出了当前用户或 PUBLIC 是其被授予者的对象授权。USER 视图列出了当前用户是其被授予者的对象授权。
USER_TAB_PRIVS_RECD	
DBA_ROLES	此视图列出了数据库中存在的角色。
DBA_ROLE_PRIVS	DBA 视图列出了授予给用户或角色的角色。USER 视图列出了授予给当前用户的角色。
USE_ROLE_PRIVS	
DBA_SYS_PRIVS	DBA 视图列出了授予给用户和角色的系统权限。USER 视图列出了授予给当前用户的系统权限
USER_SYS_PRIVS	
ROLE_ROLE_PRIVS	此视图描述了授予给其他角色的角色。它提供的只是该用户可以访问的角色的信息。
ROLE_SYS_PRIVS	此视图包含了授予给角色的系统权限的信息。它提供的只是该用户可以访问的角色的信息。
ROLE_TAB_PRIVS	此视图包含了授予给角色的对象权限的信息。它提供的只是该用户可以访问的角色的信息。
SESSION_PRIVS	此视图列出了用户当前启用的权限
SESSION_ROLES	此视图列出了用户当前启用的角色。

下面是使用这些视图的一些例子。在这些示例中，我们均假设已经发布了如下语句：

```
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;
```

25.7.1 列出所有系统权限的授权

下列查询返回的是，给角色和用户授予的所有系统权限：

```
SELECT * FROM DBA_SYS_PRIVS;
GRANTED 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
```

25.7.2 列出所有角色的授权

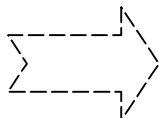
下列查询返回的是，给用户和其他角色授予的所有角色：

```
SELECT * FROM DBA_ROLE_PRIVS;
GRANTEE GRANTED_ROLE ADM
-----
SWILLIAMS SECURITY_ADMIN NO
```

25.7.3 列出授予给用户的对象权限

下列查询返回的是，授予给指定用户的所有对象权限（不包括列特殊的权限）：

```
SELECT TABLE_NAME, PRIVILEGE, GRANTABLE FROM DBA_TAB_PRIVS
```



```
WHERE GRANTEE = 'JWARD';
TABLE_NAME PRIVILEGE GRANTABLE
-----
```

```
EMP SELECT NO
```

```
EMP DELETE NO
```

要想列出已经被授予了的、所有的列特殊的权限，就使用如下查询：

```
SELECT GRANTEE, TABLE_NAME, COLUMN_NAME, PRIVILEGE
```

```
FROM 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
```

25.7.4 列出会话的当前权限域

如下查询列出该语句的发布者当前所启用的所有角色：

```
SELECT * FROM SESSION_ROLES;
```

如果 swilliams 已经启用了角色 security_admin，并且发布该查询语句，则 Oracle 返回如下信息：

```
ROLE
-----
```

```
SECURITY_ADMIN
```

下列查询列出了，在该命令的发布者的安全域中，当前可用的、明确的权限授权的和被启用的角色的所有系统权限：

```
SELECT * FROM SESSION_PRIVS;
```

如果 swilliams 启用了 security_admin 角色，并且发布该查询语句，则 Oracle 返回如下信息：

```
PRIVILEGE
-----
```

```
AUDIT SYSTEM
```

```
CREATE SESSION
```

```
CREATE USER
```

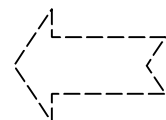
```
BECOME USER
```

```
ALTER USER
```

```
DROP USER
```

```
CREATE ROLE
```

```
DROP ANY ROLE
```



```
GRANT ANY ROLE
AUDIT ANY
CREATE PROFILE
ALTER PROFILE
DROP PROFILE
```

如果 swilliams 停用了 security_admin 角色，则第一个查询不返回任何行，而第二个查询只会为 CREATE SESSION 权限的授权返回一行。

25.7.5 列出数据库角色

可以用 DBA_ROLES 数据字典视图，来列出数据库的所有角色和用于每个角色的授权。例如，如下查询将列出数据库中的所有角色：

```
SELECT * FROM DBA_ROLES;
ROLE PASSWORD
-----
CONNECT NO
RESOURCE NO
DBA NO
SECURITY_ADMIN YES
```

25.7.6 列出有关角色的权限域的信息

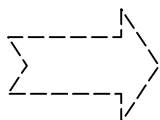
ROLE_ROLE_PRIVS, ROLE_SYS_PRIVS 和 ROLE_TAB_PRIVS 数据字典视图包括角色的权限域的相关信息。

例如，如下查询将列出授予给 system_admin 角色的所有角色：

```
SELECT GRANTED_ROLE, ADMIN_OPTION
FROM ROLE_ROLE_PRIVS
WHERE ROLE = 'SYSTEM_ADMIN';
GRANTED_ROLE ADM
-----
SECURITY_ADMIN NO
```

如下查询将列出授予给 security_admin 角色的所有系统权限：

```
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 ANY YES
SECURITY_ADMIN AUDIT SYSTEM YES
SECURITY_ADMIN BECOME USER YES
```



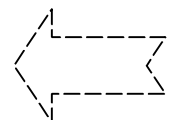
```
SECURITY_ADMIN CREATE PROFILE YES
SECURITY_ADMIN CREATE ROLE YES
SECURITY_ADMIN CREATE USER YES
SECURITY_ADMIN DROP ANY ROLE YES
SECURITY_ADMIN DROP PROFILE YES
SECURITY_ADMIN DROP USER YES
SECURITY_ADMIN GRANT ANY ROLE YES
```

如下查询将列出授予给 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
```



第 26 章 审计数据库的使用

本章描述如何使用 Oracle 数据库服务器的审计功能，包含以下内容：

- 审计的准则
- 管理审计跟踪信息
- 细粒度的审计
- 查看数据库的审计跟踪信息

26.1 审计的准则

这一节介绍审计的准则，包含以下内容：

- 决定是否使用数据库或操作系统的审计跟踪功能
- 保持审计过的信息可管理
- 审计可疑的数据库活动的准则
- 审计正常的数据库活动的准则

26.1.1 决定是否使用数据库或操作系统的审计跟踪功能

每个数据库的数据字典都有一个名为 `SYS.AUD$` 的表，通常被称为数据库审计跟踪，用这个表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包含审计跟踪，用它来存储由操作系统审计程序生成的审计记录。操作系统特定的审计程序也许支持或者不支持对操作系统审计跟踪的数据库审计功能。如果这个选项可用，就需要考虑使用数据库或者操作系统审计跟踪来存储数据库审计记录所带来的有利和不利因素。

使用数据库审计跟踪会带来下列好处：

- 根据预先定义的数据字典的审计跟踪视图来查看所选的审计跟踪部分。
- 可以使用 Oracle 工具（比如 Oracle 报表）生成审计报表。

操作系统的审计跟踪也可以合并来自多个源的审计记录，这些源包括 Oracle 和其他应用程序。因此，如果所有审计记录都来自同一个地方，那么检验系统活动将会更加有效。

26.1.2 保持审计过的信息可管理

尽管相对来说审计的花费不是很多，但是要尽可能地限制经过审计的事件的数目。这样将会最低限度地减小执行经过审计的语句所带来的性能方面的影响，并且减小审计跟踪的大小。

当设计审计策略时使用下面的常用准则：

- 评价审计的目的。

在对审计的原因有清晰地了解之后，可以设计合适的审计策略并避免不必要的审计。

例如，假设正在执行审计以便调查可疑的数据库活动。这种信息不是特别够。开始怀疑或已经注意到什么类型的可疑数据库活动？审计的更中心的目的是审计未授权的从数据

库中删除任意表的活动。这个目的缩小了正在接受审计的动作的类型范围以及可疑活动影响的对象的类型范围。

- 理智地进行审计。

审计必要的最少数目的语句、用户或对象来获得目标信息。这样，避免了从混乱的重要的信息中审计不必要的信息并因此防止消耗 `SYSTEM` 表空间中的宝贵空间。平衡一下你的需求来收集足够的信息，以便有能力存储和处理这些信息。

例如，如果正在审计以便收集有关数据库活动的信息，则需要准确地确定出正在跟踪的活动的类型，只审计感兴趣的活动的，并且只在必要的时间内进行审计以便收集想要的信息。如果只对每个会话的逻辑 I/O 信息感兴趣，就不要审计对象。

26.1.3 审计可疑的数据库活动的准则

当执行审计以便监控可疑的数据库活动时，使用以下一些准则：

- 先进行一般的审计，再进行特别的审计。

当开始审计可疑的数据库活动时，通常，可以用于目标指定用户或模式对象的信息并不是很多。因此，必须首先设置一些更加普通的审计选项。一旦记录并分析了初步的审计信息，就应该关掉这些普通的审计选项并启用更特殊的审计选项。这种处理过程一直继续，直到收集了足够证据可以对可疑数据库活动的可疑起因做明确而具体的结论时为止。

- 保护审计跟踪。

当审计可疑的数据库活动时，需要保护审计跟踪以便在没有经过审计时不能添加、更改或删除审计信息。

26.1.4 审计正常的数据库活动的准则

当审计的目的是收集特定的数据库活动的历史信息时，使用以下准则：

- 只审计适当的操作。

要避免在混乱的重要信息中得到无用的审计记录并减少审计跟踪管理的数量，只需要审计目标数据库活动就可以。

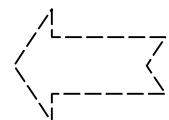
- 存档审计记录并清除审计跟踪。

在收集到所需的信息之后，将感兴趣的审计记录存档并清除这些信息的审计跟踪。

26.2 管理审计跟踪信息

这一节介绍管理审计跟踪信息的各个方面，包含以下内容：

- 审计跟踪中包含哪些信息？
- 默认的审计事件
- 设置审计选项
- 启用和禁用数据库审计
- 控制审计跟踪的增长和大小
- 保护审计跟踪



26.2.1 审计跟踪中包含哪些信息

审计跟踪记录可以包含不同类型的信息，这要根据审计的事件和设置的审计选项来定。通常每个审计跟踪记录要包括下面的信息：

- 操作系统登录用户名
- 用户名
- 会话标识符
- 终端标识符
- 访问的模式对象的名称
- 已经执行的操作或者计划执行的操作
- 操作的完成代码
- 日期和时间戳

写入操作系统审计跟踪中的审计跟踪记录是加密的并且不可读，但是这些记录可以在数据字典文件中解密，可能出现以下错误消息：

Action Code（操作代码） 它描述已经执行或者计划执行的操作。AUDIT_ACTION 数据字典表包含这些代码以及各自说明的一个列表。

Privileges Used（使用的权限） 它描述执行操作所使用的系统权限。SYSTEM_PRIVILEGE_MAP 表罗列出了所有这些代码及其说明。

Completion Code（完成代码） 它描述计划操作的结果。成功的操作将返回零值；不成功的操作将返回 Oracle 错误代码，这些代码将描述操作之所以不成功的原因。这些代码列在 Oracle 9i Database Error Messages（Oracle 9i 数据库错误消息）中。

审计跟踪不存储有关数据值的信息，因为这可能涉及到经过审计的语句。例如，当审计 UPDATE 语句时，就不存储更新过的行的新数据值和原来的旧数据值。但是，使用细粒度审计方法可以执行这种特定类型的审计。

26.2.2 默认的审计事件

不管是否启用数据库审计，Oracle 通常会审计某些关系数据库动作并写入操作系统审计跟踪中。这些事件包括以下一些内容：

- 启动实例

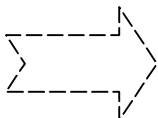
将生成一个审计记录，该记录会列出启动实例的操作系统用户、该用户的终端标识符、日期和时间戳以及是否启用或禁用数据库审计等内容。记录将存储在操作系统审计跟踪中，因为直到启动成功完成之后才能利用数据库审计跟踪。记录启动时的数据库数据状态将有助于检测出管理员何时在禁用数据库审计的情况下重新启动数据库（因此可以使得管理员执行未审计的动作）。

- 关闭实例

将生成一个审计记录，该记录会列出关闭实例的操作系统用户、用户的终端标识符以及日期和时间戳。

- 使用管理员权限连接数据库

将生成一个审计记录，该记录会列出作为 SYSOPER 或 SYSDBA 连接到 Oracle 的操作



系统用户。它将提供拥有管理员权限的用户的可计算性。

在某些操作系统上，如果 Oracle 不能访问审计跟踪，那么这些审计跟踪记录将放置在与后台进程跟踪文件相同路径的 Oracle 审计跟踪文件中。

26.2.3 设置审计选项

使用 AUDIT 语句来指定审计选项。AUDIT 语句允许用户在下列三种级别下设置选项：

级别	效果
语句	导致影响特定类型数据库对象的特殊的 SQL 语句或语句组的审计。例如，AUDIT TABLE 审计 CREATE TABLE, TRUNCATE TABLE, COMMENT ON TABLE 和 DELETE [FROM] TABLE 语句。
权限	审计授予特殊系统权限的语句。例如，AUDIT CREATE ANY TRIGGER 审计使用 CREATE ANY TRIGGER 系统权限发布的语句。
对象	审计作用在特殊对象上的语句，比如 emp 表上的 ALTER TABLE 语句。

要使用 AUDIT 语句来设置语句以及权限选项，就必须拥有 AUDIT SYSTEM 权限。要使用 AUDIT 语句来设置对象审计选项，就必须拥有经过审计的对象或者拥有 AUDIT ANY 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，用这个子句来指定用户或者应用程序代理的列表，以便限制语句和权限审计选项的范围。

当设置审计选项时，也可以为审计指定以下条件：

- BY SESSION/BY ACCESS

BY SESSION 将使得 Oracle 为在相同会话中发布的同一种类型的所有 SQL 语句写入一条记录。BY ACCESS 将使得 Oracle 为每一个访问写入一条记录。

- WHENEVER SUCCESSFUL/ WHENEVER NOT SUCCESSFUL

WHENEVER SUCCESSFUL 选择只对成功的语句进行审计。WHENEVER NOT SUCCESSFUL 选择只对失败或导致错误的语句进行审计。

在后面的章节中将对审计选项的含义以及 AUDIT 语句的子句进行讨论。

在新的数据库会话创建的时候，该会话将从数据字典中获得审计选项。在数据库连接期间这些审计选项将保持有效。新系统或对象审计选项的设置将会引起所有后续数据库会话使用这些选项；现有的会话继续使用会话创建时设置的审计选项。

警告：AUDIT 语句只能指定审计选项；它不能从整体上启用审计。如果要打开审计功能并基于当前设置的审计选项控制 Oracle 是否生成审计记录，就需要设置初始化参数 AUDIT_TRAIL，该参数将在本章后面的“启用和禁用数据库审计”一节中加以介绍。

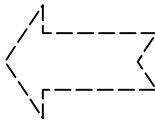
26.2.3.1 指定语句审计

有效的语句审计选项可以包含在 AUDIT 和 NOAUDIT 语句中。

在下面的章节中将讨论语句审计的两个特殊的例子。

26.2.3.2 审计连接和审计断开连接

SESSION 语句选项是一个特别的选项，因为当发布特殊类型的语句时它不能生成审计



记录；这个选项为由实例的连接创建的每个会话生成一条审计记录。审计记录将在连接时期插入到审计跟踪中，并且在断开连接时期进行更新。累积的有关会话的信息，比如连接时期、断开连接时期、处理的逻辑和物理 I/O 以及更多信息将存储在单独一条审计记录中，该审计记录与会话相对应。

要审计所有成功和不成功的数据库连接以及断开连接，不管用户是谁，均可采用 **BY SESSION**（这个选项的默认值，并且是唯一的值），键入以下语句：

```
AUDIT SESSION;
```

也可以单独为用户设置这个选项，如同下面的例子：

```
AUDIT SESSION
```

```
BY scott, lori;
```

26.2.3.3 审计因为对象不存在而导致失败的语句

NOT EXISTS 语句选项指定审计因为目标对象不存在而导致失败的所有 SQL 语句。

26.2.3.4 指定权限审计

权限审计选项准确地与对应的系统权限匹配。例如，使用 **DELETE ANY TABLE** 权限进行审计的选项：**DELETE ANY TABLE**。要启用该选项，使用类似下面例子中的语句即可：

```
AUDIT DELETE ANY TABLE
```

```
BY ACCESS
```

```
WHENEVER NOT SUCCESSFUL;
```

要审计 **DELETE ANY TABLE** 系统权限的所有成功和不成功使用，键入下列语句：

```
AUDIT DELETE ANY TABLE;
```

要审计在所有表上不成功的所有 **SELECT**、**INSERT** 和 **DELETE** 语句以及 **EXECUTE PROCEDURE** 系统权限的不成功使用，需要由所有数据库用户和个别经过审计的语句发布以下语句：

```
AUDIT SELECT TABLE, INSERT TABLE, DELETE TABLE,
```

```
EXECUTE PROCEDURE
```

```
BY ACCESS
```

```
WHENEVER NOT SUCCESSFUL;
```

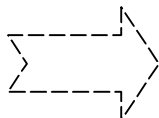
设置任何语句或权限审计选项都必须拥有 **AUDIT SYSTEM** 系统权限。通常，安全管理员是唯一授予这个系统权限的用户。

26.2.3.5 指定对象审计

用户可以为包含在用户模式中的对象设置任何的对象审计选项。为包含在另一个用户模式中的对象设置对象审计选项或者设置默认对象审计选项，就必须有 **AUDIT ANY** 系统权限。通常，安全管理员是唯一授予 **AUDIT ANY** 系统权限的用户。

要审计 **scott.emp** 表上的所有成功和不成功的 **DELETE** 语句，采用 **BY SESSION**（默认值），键入下列语句：

```
AUDIT DELETE ON scott.emp;
```



要审计属于用户 `jward` 的 `dept` 表上的所有成功的 `SELECT`, `INSERT` 和 `DELETE` 语句, 采用 `BY ACCESS`, 键入下列语句:

```
AUDIT SELECT, INSERT, DELETE
ON jward.dept
BY ACCESS
WHENEVER SUCCESSFUL;
```

要将默认对象审计选项设置为审计所有不成功的 `SELECT` 语句, 采用 `BY SESSION` (默认值), 键入下列语句:

```
AUDIT SELECT
ON DEFAULT
WHENEVER NOT SUCCESSFUL;
```

26.2.3.6 在多层环境中进行审计

在多层环境中, Oracle 通过所有层保存客户的标识符。这样就能够审计代表客户实施的操作。如果这么做, 只要在 `AUDIT` 语句中使用 `BY proxy` 子句即可。

该子句允许带一些选项。用户可以利用这个子句做以下工作:

- 审计由指定代理为自己发布的 `SQL` 语句
- 审计代表指定的一个或多个用户执行的语句
- 审计代表任何一个用户执行的所有语句

下面的例子审计了由代理应用程序服务器 `appserve` 代表客户 `jackson` 发布的 `SELECT TABLE` 语句。

```
AUDIT SELECT TABLE
BY appserve ON BEHALF OF jackson;
```

26.2.4 关闭审计选项

`NOAUDIT` 语句可以关闭 Oracle 的各种审计选项。使用该语句重新设置语句、权限审计选项以及对象审计选项。设置语句和权限审计选项的 `NOAUDIT` 语句可以包含 `BY user` 或 `BY proxy` 选项来指定用户列表, 以便限制语句和权限审计选项的范围。

可以使用 `NOAUDIT` 语句来禁用使用 `WHENEVER` 子句选择的审计选项。如果没有指定子句, 审计选项将被完全禁用, 对成功和不成功的情况都是如此。

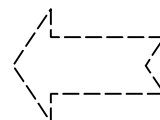
`NOAUDIT` 语句不支持 `BY SESSION`/`BY ACCESS` 选项对; 不论这些审计选项是如何打开的, 都可以由一个适当的 `NOAUDIT` 语句关闭掉这些选项。

警告: `NOAUDIT` 语句只指定审计选项; 它不能从整体上禁用审计。要关闭审计并停止 Oracle 生成审计记录, 可以在数据库初始化参数文件中设置初始化参数 `AUDIT_TRAIL`, 该参数将在本章后面的“启用和禁用数据库审计”一节中加以描述。

26.2.4.1 关闭语句和权限审计

下面的语句将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;
```

下面的语句将关闭所有语句审计选项:

```
NOAUDIT ALL;
```

下面的语句将关闭所有权限审计选项:

```
NOAUDIT ALL PRIVILEGES;
```

要禁用语句或权限审计选项, 就必须拥有 AUDIT SYSTEM 系统权限。

26.2.4.2 关闭对象审计

下面的语句将关闭相应的审计选项:

```
NOAUDIT DELETE  
ON emp;  
NOAUDIT SELECT, INSERT, DELETE  
ON jward.dept;
```

此外, 要关闭 emp 表上的所有对象审计选项, 可以键入下列语句:

```
NOAUDIT ALL  
ON emp;
```

要关闭所有默认的对象审计选项, 可以键入下列语句:

```
NOAUDIT ALL  
ON DEFAULT;
```

所有在发布 NOAUDIT 语句之前创建的模式对象除非在它们创建之后由一个明确的 NOAUDIT 语句重写了对象审计选项, 否则都将继续使用在它们创建时有效的默认的对象审计选项。

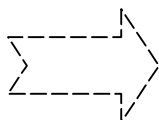
要禁用某个特定对象的对象审计选项, 用户就必须是模式对象的所有者。要禁用另一个用户的模式中对对象的对象审计选项, 或者禁用默认的对象审计选项, 就必须拥有 AUDIT ANY 系统权限。赋予禁用对象的对象审计选项权限的用户可以重写由任何用户设置的选项。

26.2.5 启用和禁用数据库审计

任何授权的数据库用户都可以在任何时候设置语句、权限和对象审计选项, 但是, 除非启用数据库审计, 否则 Oracle 不能生成并在审计跟踪中存储审计记录。安全管理员通常负责这项操作。

通过数据库初始化参数文件中的 AUDIT_TRAIL 初始化参数启用和禁用数据库审计。这个参数可以设置为下列值:

DB	启用数据库审计并引导所有审计记录到数据库的审计跟踪
OS	启用数据库审计并引导所有审计记录到操作系统的审计跟踪



NONE 禁用审计（这个值是默认值。）

可以用 `AUDIT_FILE_DEST` 初始化参数来指定审计文件存储的目录。

如果编辑了初始化参数文件，需要重新启动数据库实例以便按照计划启用或禁用数据库审计。这些参数不是动态的。

26.2.6 控制审计跟踪的增长和大小

如果审计跟踪已经完全装满，而且不能继续插入更多的审计跟踪，那么将不能成功执行经过审计的语句，直到清除审计跟踪。所有发布了经过审计的语句的用户都会收到警告信息。因此，安全管理员必须控制审计跟踪的增长和大小。

当启用审计并且正在生成审计记录时，审计跟踪将依据以下两个因素增长：

- 打开的审计选项的数目
- 经过审计的语句的执行频率

要控制审计跟踪的增长，可以采用以下方法：

- 启用和禁用数据库审计。如果启用了数据库审计，那么将生成审计记录并将其存储在审计跟踪中；如果禁用了数据库审计，那么将不能生成审计记录。
- 要精挑细选需要打开的审计选项。如果执行了很多经过选择的审计，那么将不会生成无用或不必要的审计信息，并将它们存储在审计跟踪中。
- 要严格控制执行对象审计的能力。可以采用下列两种不同的方式进行控制：

— 安全管理员拥有所有对象和 `AUDIT ANY` 系统权限，并且从未将该系统权限授予其他用户。所有模式对象也可以属于某个相应的用户没有 `CREATE SESSION` 权限的模式。

— 所有对象都包含在没有对应真实的数据库用户的模式中（也就是说，没有授予 `CREATE SESSION` 权限给相应的用户），而且安全管理员是唯一授予 `AUDIT ANY` 系统权限的用户。

在这两种方案中，完全由安全管理员控制对象审计。

数据库审计跟踪的最大大小是在数据库创建期间预先确定的。默认情况下，最多 99 个区域，每个区域为 10K 大小，可以在表中进行分配。作为控制审计跟踪增长和大小的一种手段，用户不应将 `SYS.AUD$` 移动到另一个表空间。但是，可以在 `SYS.AUD$` 中修改默认的存储参数。

注意：因为 Oracle 代码隐含关于数据字典表的假设，比如 `SYS.AUD$`，所以不支持从 `SYSTEM` 表空间中移走这个表，否则将引起升级和备份/恢复方案方面的问题。

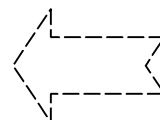
26.2.6.1 从审计跟踪中清除审计记录

在某些时候启用了审计之后，安全管理员也许想从数据库审计跟踪中删除记录，一方面腾出审计跟踪空间，另一方面方便审计跟踪管理。

例如，要从审计跟踪中删除所有审计记录，可以键入下列语句：

```
DELETE FROM SYS.AUD$;
```

还可以从作为审计表 `emp` 的结果产生的审计跟踪中删除所有审计记录，可以键入下列语句：



```
DELETE FROM SYS.AUD$
WHERE obj$name='EMP';
```

如果为了留下历史记录的目的而必须存档审计跟踪信息, 安全管理员可以将相关的记录复制到普通的数据库表(例如, 使用 `INSERT INTO table SELECT...FROM SYS.AUD$...`) 或将审计跟踪表导出到某个操作系统文件中。

只有用户 SYS、拥有 DELETE ANY TABLE 权限的用户或者在 SYS.AUD\$ 上被 SYS 授予 DELETE 权限的用户才能将记录从数据库审计跟踪中删除。

注意: 如果审计跟踪已经完全装满并且正在审计连接(也就是说, 如果设置了 SESSION 选项), 那么会因为与连接相关联的审计记录不能插入到审计跟踪中而导致特有的用户不能连接到数据库。在这种情况下, 安全管理员必须作为 SYS 进行连接(SYS 执行的操作不审计)并且在审计跟踪中腾出可用空间。

26.2.6.2 缩减审计跟踪的大小

对于任何数据库表来说, 从数据库审计跟踪中删除记录之后, 为这个表分配的区域依然存在。

如果数据库审计跟踪已经有很多分配的区域, 但是其中有一些当前没有使用, 那么可以通过以下步骤减小分配给数据库审计跟踪的空间:

①如果当前想把信息保存到审计跟踪中, 可以将它复制到另一个数据库表或使用 EXPORT 实用程序将它导出。

②作为拥有管理员权限的用户进行连接。

③使用 TRUNCATE 语句截取 SYS.AUD\$。

④重新装载第 1 步生成的存档的审计跟踪记录。

分配给新版本 SYS.AUD\$ 的区域和包含当前审计跟踪记录所需要的区域一样多。

注意: SYS.AUD\$ 是唯一一个可以直接修改的 SYS 对象。

26.2.7 保护审计跟踪

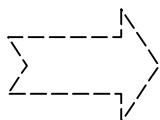
当审计可疑的数据库活动时, 需要保护审计跟踪记录的完整性, 以便确保审计信息的准确性和完备性。

为了保护数据库审计跟踪免于未经授权的删除操作, 可以只给安全管理员授予 DELETE ANY TABLE 系统权限。

要审计对数据库审计跟踪所做的修改, 可以使用以下语句:

```
AUDIT INSERT, UPDATE, DELETE
ON sys.aud$
BY ACCESS;
```

作为为 SYS.AUD\$ 表设置的对象审计选项的结果而生成的审计记录, 只能由拥有管理员权限的连接者才能从审计跟踪中将它们删除掉, 审计跟踪本身有免于未经授权使用的保护能力。如果可以的话, 作为保护审计跟踪的最后一道防线, 需要保证在用管理员权限连接的时候执行的任何操作在操作系统审计跟踪中都被审计。



26.3 细粒度审计

迄今为止，在讨论的审计方法中，都有一套固定的事实记录在审计跟踪中。而且，审计选项只能设置为监控对象或权限的访问。已经讨论的内容中还没有涉及获得有关环境或查询结果的更多特殊信息方面的内容，也没有任何机制来指定审计条件以降低错误审计出现的概率。为此，Oracle 提供了细粒度审计来解决这些问题。

细粒度的审计允许基于内容的数据访问的监控。例如，国税管理局需要跟踪访问返还的税收以免雇员偷窃。需要有足够的细节才能决定访问过什么数据，而不仅仅是知道在某个特殊表上由某个特定的用户使用了 SELECT 权限。细粒度审计将提供这种功能。

通常，细粒度审计策略是基于简单的用户定义的 SQL 依据表对象作为选择的审计的条件。在整个提取过程中，无论何时满足策略条件而返回一行，都会审计这个查询。之后，Oracle 执行用户定义的审计事件处理程序，使用自主事务来处理事件。

可以使用 DBMS_FGA 包或使用数据库触发器在用户应用程序中实现细粒度审计。

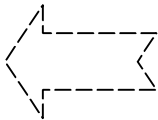
26.4 查看数据库的审计跟踪信息

数据库审计跟踪 (SYS.AUD\$) 是每个 Oracle 数据库的数据字典中的一个单独的表。为了帮助用户有目的的查看这个表中的审计信息，可以利用几个预先定义好的视图。这些视图必须是由用户创建。如果决定不再使用审计，可以在以后将它们删除。

26.4.1 创建审计跟踪视图

下列视图（除了 STMT_AUDIT_OPTION_MAP）是由 CATALOG.SQL 和 CATAUDIT.SQL 脚本创建的：

视图	说明
STMT_AUDIT_OPTION_MAP	包含有关审计选项类型代码的信息。由 SQL.BSQ 脚本在 CREATE DATABASE 的时候创建。
AUDIT_ACTIONS	包含对审计跟踪动作类型代码的说明。
ALL_DEF_AUDIT_OPTS	包含默认对象审计选项，当创建对象时将应用这些选项。
DBA_STMT_AUDIT_OPTS	描述由用户设置的跨系统的当前系统审计选项。
DBA_PRIV_AUDIT_OPTS	描述由用户正在审计的跨系统的当前系统权限。
DBA_OBJ_AUDIT_OPTS	描述在所有对象上的审计选项。USER 视图描述当前用户拥有的所有对象上的审计选项。
USER_OBJ_AUDIT_OPTS	描述在所有对象上的审计选项。
DBA_AUDIT_TRAIL USER_AUDIT_TRAIL	列出所有审计跟踪条目。USER 视图显示与当前用户有关的审计跟踪条目。



(续表)

视图	说明
DBA_AUDIT_OBJECT USER_AUDIT_OBJECT	包含系统中所有对象的审计跟踪记录。USER 视图列出一些审计跟踪记录，而这些记录涉及当前用户可以访问的对象的语句。
DBA_AUDIT_SESSION USER_AUDIT_SESSION	列出涉及 CONNECT 和 DISCONNECT 的所有审计跟踪记录。USER 视图列出涉及当前用户的 CONNECT 和 DISCONNECT 的所有审计跟踪记录。
DBA_AUDIT_STATEMENT USER_AUDIT_STATEMENT	列出涉及数据库全部的 GRANT、REVOKE、AUDIT、NOAUDIT 和 ALTER SYSTEM 语句的审计跟踪记录，对于 USER 视图来说，这些语句应是用户发布的。
DBA_AUDIT_EXISTS	列出 BY AUDIT NOT EXISTS 产生的审计跟踪条目。
下面的视图用于细粒度审计：	
DBA_AUDIT_POLICIES	显示系统上的所有审计策略。
DBA_FGA_AUDIT_TRAIL	列出基于值的审计的审计跟踪记录。

26. 4. 2 删除审计跟踪视图

如果禁用了审计并且不再需要审计跟踪视图，就可以通过作为 SYS 连接到数据库并运行脚本文件 CATNOAUD.SQL 的办法删除这些视图。CATNOAUD.SQL 脚本的名称和位置因操作系统而异。

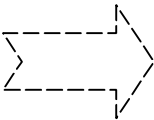
26. 4. 3 使用审计跟踪视图调查可疑的活动

这一节提供一些例子来说明如何检查和解释审计跟踪中的信息。考虑以下情况。用户也许希望审计数据库的以下可疑活动：

- 因为某些数据库用户未经授权就修改了密码、表空间设置和配额。
- 发生大量数目的死锁，大多是因为用户获得了独占式表锁。
- 在 scott 的模式中正在从 emp 表中任意地删除行。

如果怀疑用户 jward 和 swilliams 做了这些危险的动作，要启用调查，可以发布以下语句（按次序）：

```
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;
```

下面的语句是后来由用户 jward 发布的:

```
ALTER USER tsmith QUOTA 0 ON users;
```

```
DROP USER djones;
```

下面的语句是后来由用户 swilliams 发布的:

```
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);
```

接下来的几节将显示有关调查的信息, 这些信息可以使用数据字典中的审计跟踪视图进行查看:

- 列出活动的语句审计选项
- 列出活动的权限审计选项
- 列出指定对象的活动的对象审计选项
- 列出默认的对象审计选项
- 列出审计记录
- 列出 AUDIT SESSION 选项的审计记录

26.4.3.1 列出活动的语句审计选项

下面的查询将返回所有设置的语句审计选项:

```
SELECT * FROM DBA_STMT_AUDIT_OPTS;
```

```
USER_NAME AUDIT_OPTION SUCCESS FAILURE
```

```
-----
```

```
JWARD SESSION BY SESSION BY SESSION
```

```
SWILLIAMS SESSION BY SESSION BY SESSION
```

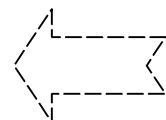
```
LOCK TABLE BY ACCESS NOT SET
```

不管语句审计选项设置成功与否, 以及是否是因 BY SESSION 或 BY ACCESS 而设置, 都会注意到视图显示出了设置的这些选项。

26.4.3.2 列出活动的权限审计选项

下面的查询将返回所有设置的权限审计选项:

```
SELECT * FROM DBA_PRIV_AUDIT_OPTS;
```



```
USER_NAME PRIVILEGE SUCCESS FAILURE
```

```
ALTER USER BY SESSION BY SESSION
```

26.4.3.3 列出指定对象的活动的对象审计选项

下面的查询将返回为名称以 **emp** 开头并且包含在 **scott** 模式中的任何对象而设置的所有审计选项：

```
SELECT * FROM DBA_OBJ_AUDIT_OPTS
WHERE OWNER = SCOTT? AND OBJECT_NAME LIKE %MP%?
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 -/- -/- ...
```

注意到视图返回了有关指定对象的所有审计选项的信息。对视图中的信息做如下解释：

- 符号“-”说明没有设置这个审计选项。
- 符号“S”说明审计选项是由 **BY SESSION** 设置的。
- 符号“A”说明审计选项是由 **BY ACCESS** 设置的。
- 每一个审计选项都有两种可能的设置，**WHENEVER SUCCESSFUL** 和 **WHENEVER NOT SUCCESSFUL**，用“/”隔开。例如，为 **scott.emp** 设置的 **DELETE** 审计选项为成功删除语句而由 **BY ACCESS** 进行设置，并且对不成功的删除语句不进行设置。

26.4.3.4 列出默认的对象审计选项

下面的查询返回所有默认的对象审计选项：

```
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 -/- -/- -/- -/-
```

注意到视图返回的信息与 **USER_OBJ_AUDIT_OPTS** 和 **DBA_OBJ_AUDIT_OPTS** 视图返回的类似（参见前面的例子）。

26.4.3.5 列出审计记录

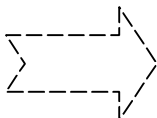
下面的查询列出了由语句审计选项和对象审计选项生成的审计记录：

```
SELECT * FROM DBA_AUDIT_OBJECT;
```

26.4.3.6 列出 AUDIT SESSION 选项的审计记录

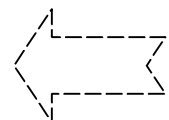
下面的查询列出了对应 **AUDIT SESSION** 语句审计选项的审计信息：

```
SELECT USERNAME, LOGOFF_TIME, LOGOFF_LREAD, LOGOFF_PREAD,
LOGOFF_LWRITE, LOGOFF_DLOCK
```




```
FROM 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
```

北京拓鼎电子出版社



第 5 部分 数据库资源管理

第 5 部分将讨论数据库资源管理，这部分包含下列章节：

- 第 27 章 使用数据库资源管理器

第 27 章 使用数据库资源管理器

Oracle 通过它的数据库资源管理器提供数据库资源管理能力。本章介绍数据库资源管理器的使用。

将讨论以下内容：

- 什么是数据库资源管理器？
- 管理数据库资源管理器
- 创建简单的资源规划
- 创建复杂的资源规划
- 管理资源用户组
- 启用数据库资源管理器
- 将所有内容结合在一起：数据库资源管理器的例子
- 监控和协调数据库资源管理器
- 查看数据库资源管理器信息

注意：这一章讨论 Oracle 提供的 DBMS_RESOURCE_MANAGER 和 DBMS_RESOURCE_MANAGER_PRIVS 包的使用，以便管理数据库资源管理器。通过 Oracle Enterprise Manager (OEM) 能更方便地管理数据库资源管理器。OEM 为管理数据库资源管理器提供了更方便的图形界面。

27.1 什么是数据库资源管理器

数据库资源管理器的主要目的是给予 Oracle 数据库服务器更多控制资源管理决策的权利，因此避免了因低效率的操作系统管理而导致的问题。

这一节包含以下内容：

- 数据库资源管理器着重处理哪些问题
- 数据库资源管理器如何处理这些问题
- 数据库资源管理器有哪些组成部分
- 了解资源规划

27.1.1 数据库资源管理器着重处理哪些问题

当把数据库资源分配决策权留给操作系统时，可能会遇到以下问题：

- 过多的系统开销

当服务器进程数目很高时，操作系统上下文在 Oracle 服务器进程之间的切换将导致过多的系统开销。

- 低效率的程序调度

当 Oracle 数据库服务器占据锁存器时，此时效率低下，操作系统将解除对 Oracle 数据库服务器的程序调度。

- 不恰当的资源分配
操作系统在所有活动进程中平均分配资源并且不能将某项任务优先列在另一项任务之上。
- 没有能力管理数据库指定的资源，比如并行从属装置和活动的会话。

27.1.2 数据库资源管理器如何处理这些问题

通过允许数据库更多的控制机器资源如何分配，Oracle 的数据库资源管理器可以帮助克服以上这些问题。

特别是使用数据库资源管理器可以做下列事情：

- 不管系统的装载量和用户数目是多少，都可以保证某些用户占据最少数量的处理资源。
- 通过将一定百分比的 CPU 时间分配给不同的用户和应用程序的办法分配可利用的处理资源。在一个数据仓库中，可能给予 ROLAP（与联机分析处理有关）应用程序的 CPU 时间百分比要大于批处理作业的百分比。
- 限制用户组成员执行的任何操作的并行度。
- 创建活动会话池。这个存储池由指定的最大数目的用户会话组成，这些会话允许在用户组中并发活动。超过这个最大数目的另外的会话将排队等候执行，但是可以指定超时时间，超时之后排队的作业将被中止。
- 允许基于管理员定义的准则在组之间进行自动切换。如果某个特殊的用户组中的成员创建了一个会话，会话的执行时间要长于指定时间，那么将会被自动切换到拥有不同资源需求的另一个用户组中。
- 防止执行估计运行时间长于预先定义的限制时间的操作。
- 创建撤消池。这个存储池由一定数量的可以在某个用户组中使用的撤消空间组成。
- 配置实例以便使用一种特殊的分配资源的方法。可以动态地修改这种方法，例如，不用必须关闭和重新启动实例就可以将白天的设置改成晚上的设置。

27.1.3 数据库资源管理器有哪些组成部分

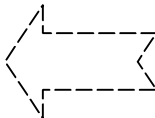
Oracle 的数据库资源管理的组成部分，可以通过数据库资源管理器包来定义，介绍如下：

组成部分	说明
资源用户组	基于资源处理需求的用户会话组合。
资源规划	包含指定资源如何分配给资源用户组的指令。
资源分配方法	当分配特殊资源时数据库资源管理器使用的方法/策略；资源用户组和资源规划使用的方法/策略。Oracle 提供可用的资源分配方法，但是用户可以决定使用哪一种方法。
资源规划指令	管理员用它来与拥有特殊规划的资源用户组联系，并将资源分配给各资源用户组。

在本章的后面章节中将学习如何创建和使用这些组成部分。

27.1.4 了解资源规划

这一节简要介绍资源规划的概念。包括简单资源规划的一些图解说明。在解释了如何



构建和维护数据库资源管理器的组成部分之后，将在后面的例子中（本章后面的“将所有内容结合在一起：数据库资源管理器的例子”一节）介绍更复杂的规划。

资源规划指定资源用户组属于某个规划并且包含如何在这些组中分配资源的指令。使用 DBMS_RESOURCE_MANAGER 包来创建和维护数据库资源管理器的这些组成部分：资源规划、资源用户组以及资源规划指令。规划信息存储在数据字典的表中。可以用一些视图来查看规划数据。

27.1.4.1 单级的资源规划

图 27-1 显示的第一个图解讲述的是一个单级的规划，这种规划在各资源用户组中分配资源。Great Bread 公司有一个规划，名为 `great_bread`，该规划将 CPU 资源分配给 3 个资源用户组。具体分配比例是，分配给 `sales` 用户组 60% 的 CPU 时间，分配给 `market` 用户组 20% 的 CPU 时间，剩下的 20% 的 CPU 时间分给了 `develop` 用户组。

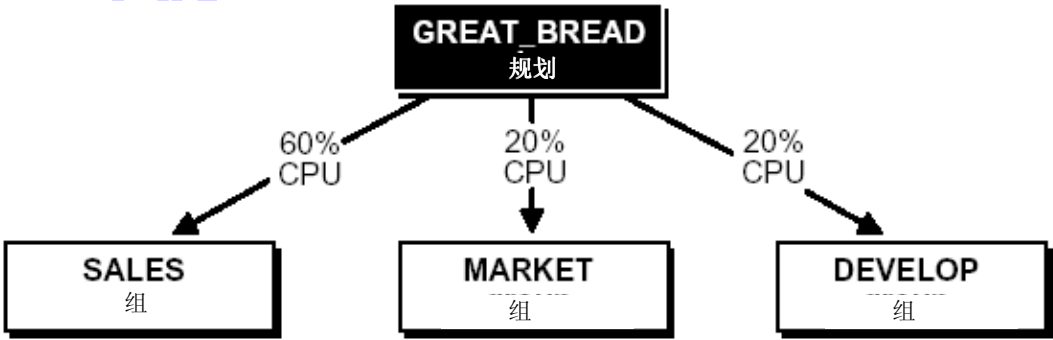


图 27-1 简单的资源管理规划

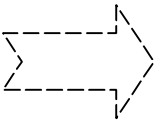
Oracle 提供了一个过程 (`CREATE_SIMPLE_PLAN`)，从而使用户能够快速创建简单的资源规划。这个过程将在本章后面的“创建简单的资源规划”一节中加以讨论。

27.1.4.2 多级的资源规划

规划可以不仅包含资源用户组，还可以包含其他的规划即子规划。Great Bread 公司也可能选择如图 27-2 所示的那样划分他们的 CPU 资源。

在这个例子中，`great_bread` 规划仍然会将 CPU 资源分配给用户组 `market`，但是现在它将 CPU 资源也分配给了子规划 `sales_team` 和 `develop_team`，这些子规划反过来又将资源分配给用户组。图 27-2 就阐述了一种规划模式，它包含顶级规划 (`great_bread`) 和所有它的分支。

子规划或用户组也可能有比一个还多的父规划（所属的规划），但是规划模式中不能有任何循环。举个多于一个父规划的子规划的例子，假如 Great Bread 公司有一个晚上的规划，还有一个白天的规划。这两个规划都包含 `sales` 子规划作为一个成员，但是每个实例中分配的 CPU 资源也许会有所不同。



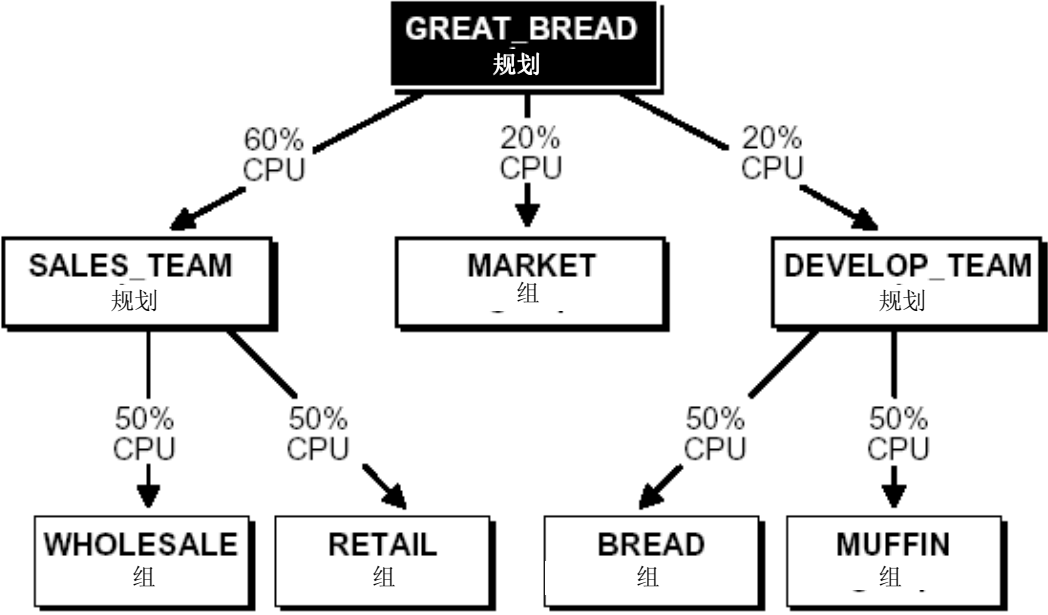


图 27-2 带有子规划的多级规划

注意：如同后面要解释的那样，上面的规划也应该为 OTHER_GROUPS 包含一个规划指令。但是，为了简化视图，视图中没有显示出这个规划指令。

27.1.4.3 资源用户组

资源用户组是一组一组的用户或者会话，它们基于处理需求进行分组。资源规划指令（接下来会介绍）指定如何将资源分配给各用户组和规划模式中的子规划。

27.1.4.4 资源规划指令

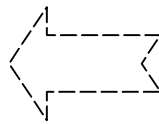
如何将资源分配给资源用户组的内容是由资源分配指令指定的。数据库资源管理器提供几种分配资源的方法。

27.1.4.5 CPU 方法

这种方法可以指定如何将 CPU 资源分配给各用户组或子规划。CPU 资源的多级（最多 8 级）分配提供了一种在规划模式中区分 CPU 使用优先次序的方式。第 2 级只能在第 1 级未用完所有资源的情况下得到资源，而多级不仅提供了区分优先次序的方式，而且还提供了明确指定如何使用所有第 1 级和剩下的资源的方式。

27.1.4.6 带排队的活动会话池

可以控制某用户组中允许并发的活动会话的最大数目。这个最大值指定了活动会话池。当因为会话池已满而不能启动某个会话时，会话将放入队列中。当某个活动会话完成时，将调度执行队列中的第一个会话。还可以指定超时时间，在超时这段时间之后，执行队列中的作业（等候执行的作业）被认定为超时，从而引起异常中止，并出现错误消息。



一个完整的并行执行会话将被算作一个活动会话。

27.1.4.7 并行度界限

指定并行度界限可以控制用户组中任何操作的最大并行度。

27.1.4.8 自动的用户组切换

这种方法可以通过指定准则来控制资源，如果满足条件将会把会话自动切换到另一个用户组。用来决定切换的准则包括以下内容：

- 切换组——指定在其他条件（如下所示）满足的情况下会话要切换到的用户组。
- 切换时间——指定在切换到另一个用户组之前会话可以执行的时间长度。
- 使用估计——指定 Oracle 是否使用它对操作将要执行多长时间所做的估计。

如果会话活动超过切换时间，那么数据库资源管理器将把运行中的会话切换到切换组。活动的意思是会话正在运行并且正在使用资源，不是在空闲地等待用户的输入，也不是在等候 CPU 周期。即使活动会话池因为新组的缘故已经装满，也会允许这个会话继续运行。在这些条件下，用户组拥有运行中的会话数目可以比它的活动会话池指定的还多。一旦会话完成它的操作并变成空闲状态，那么将切换回原来的组。

如果使用估计设置为 TRUE，那么数据库资源管理器将使用一个预报的操作完成时间的估计。如果 Oracle 的预报估计长于切换时间指定的值，那么 Oracle 将在执行开始之前切换会话。如果没有设置这个参数，操作将正常开始并且只在满足切换条件的时候才切换到其他组。

27.1.4.9 执行时间界限

可以指定操作允许的执行时间的最大值。如果 Oracle 估计操作运行时间将会长于指定的执行时间的最大值，那么操作将会中断并出现错误。可以中断这个错误并重新调度这个操作。

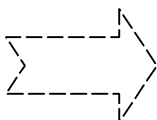
27.1.4.10 撤消池

可以为每个用户组指定一个撤消池。撤消池将控制用户组生成的整个撤消的量。当用户组生成的整个撤消超过它的撤消界限时，将中断生成该撤消的当前的 DML 语句。用户组的其他成员不能执行进一步的数据操作，直到存储池释放出撤消空间为止。

27.2 管理数据库资源管理器

必须拥有系统权限 ADMINISTER_RESOURCE_MANAGER 才能管理数据库资源管理器。通常，作为 DBA 角色的一部分，数据库管理员拥有带 ADMIN 选项的这个权限。

作为数据库资源管理器的管理员，可以执行 DBMS_RESOURCE_MANAGER 包中的所有过程。下面的表中列出了这些过程，我们将在这一章的后面章节中解释这些过程的使用。



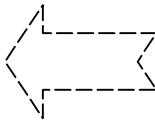
过程	说明
CREATE_SIMPLE_PLAN	创建一个简单的资源规划，在一步中可以包含多达 8 个的用户组。这是最快学习使用这个包的方式。
CREATE_PLAN	创建资源规划并指定它的分配方法。
UPDATE_PLAN	更新资源规划的注释信息。
DELETE_PLAN	删除资源规划和它的分支。
DELETE_PLAN_CASCADE	删除资源规划以及所有它的分支。
CREATE_CONSUMER_GROUP	创建资源用户组。
UPDATE_CONSUMER_GROUP	更新资源用户组的注释信息。
DELETE_CONSUMER_GROUP	删除资源用户组。
CREATE_PLAN_DIRECTIVE	指定资源规划指令，这些指令将把资源分配给某个规划中的资源用户组或者分配给多级规划模式中的子规划。
UPDATE_PLAN_DIRECTIVE	更新规划指令。
DELETE_PLAN_DIRECTIVE	删除规划指令。
CREATE_PENDING_AREA	创建一个暂挂区域（暂存区），在这个区域中可以对规划模式进行修改。
VALIDATE_PENDING_AREA	使得在暂挂区域对规划模式所做的修改有效。
CLEAR_PENDING_AREA	从暂挂区域清除所有起草的修改。
SUBMIT_PENDING_AREA	将所有修改内容提交给规划模式。
SET_INITIAL_CONSUMER_GROUP	为用户设置最初的用户组。
SWITCH_CONSUMER_GROUP_FOR_SESS	切换特定会话的用户组。
SWITCH_CONSUMER_GROUP_FOR_USER	切换属于某个特定用户的所有会话的用户组。

作为拥有带 ADMIN 选项的系统权限的管理员，可以选择将管理权限授予其他用户或角色。这里可以使用 DBMS_RESOURCE_MANAGER_PRIVS 包。这个包包含下面这个表所列出的过程。

过程	说明
GRANT_SYSTEM_PRIVILEGE	将 ADMINISTER_RESOURCE_MANAGER 系统权限授予某个用户或角色。
REVOKE_SYSTEM_PRIVILEGE	撤回某个用户或角色的 ADMINISTER_RESOURCE_MANAGER 系统权限。
GRANT_SWITCH_CONSUMER_GROUP	许可用户、角色或 PUBLIC 切换到指定的资源用户组。
REVOKE_SWITCH_CONSUMER_GROUP	撤回对用户、角色或 PUBLIC 切换到指定的资源用户组的许可。

在下面的例子中，将管理权限授予了用户 scott，但是没有给 scott 授予 ADMIN 选项。因此，scott 可以执行 DBMS_RESOURCE_MANAGER 包中的所有过程，但是 scott 不能使用 GRANT_SYSTEM_PRIVILEGE 过程将管理权限授予其他人。

```
EXEC DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SYSTEM_PRIVILEGE -  
  (GRANTEE_NAME => 'scott', PRIVILEGE_NAME => 'ADMINISTER_RESOURCE_MANAGER', -  
   ADMIN_OPTION => FALSE);
```



可以使用 REVOKE_SYSTEM_PRIVILEGE 过程撤回这个权限。

注意：只能使用 DBMS_RESOURCE_MANAGER_PRIVS 包授予或撤回 ADMINISTER_RESOURCE_MANAGER 系统权限。不能通过 SQL GRANT 或 REVOKE 语句授予或撤回该权限。

DBMS_RESOURCE_MANAGER_PRIVS 包中的其他过程将在本章后面的“管理切换权限”一节中加以讨论。

27.3 创建简单的资源规划

使用 CREATE_SIMPLE_PLAN 过程可以快速创建简单的资源规划，对于大多数情形，这种规划已经足够了。这个过程可以创建用户组，并通过执行简单的语句来给它们分配资源。使用这个过程，不需要撤回后续章节中将要介绍的过程，包括创建暂挂区域的过程、分别单独创建每个用户组的过程以及指定资源规划指令的过程。

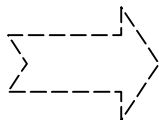
可以为 CREATE_SIMPLE_PLAN 过程指定以下参数：

参数	说明
SIMPLE_PLAN	规划的名称
CONSUMER_GROUP1	第 1 组的用户组名称
GROUP1_CPU	分配给第 1 组的 CPU 资源
CONSUMER_GROUP2	第 2 组的用户组名称
GROUP2_CPU	分配给第 2 组的 CPU 资源
CONSUMER_GROUP3	第 3 组的用户组名称
GROUP3_CPU	分配给第 3 组的 CPU 资源
CONSUMER_GROUP4	第 4 组的用户组名称
GROUP4_CPU	分配给第 4 组的 CPU 资源
CONSUMER_GROUP5	第 5 组的用户组名称
GROUP5_CPU	分配给第 5 组的 CPU 资源
CONSUMER_GROUP6	第 6 组的用户组名称
GROUP6_CPU	分配给第 6 组的 CPU 资源
CONSUMER_GROUP7	第 7 组的用户组名称
GROUP7_CPU	分配给第 7 组的 CPU 资源
CONSUMER_GROUP8	第 8 组的用户组名称
GROUP8_CPU	分配给第 8 组的 CPU 资源

使用这个过程可以指定多达 8 个的用户组，而且只有为 CPU 才能指定一个规划指令。在这个规划中指定的每一个用户组都是在第 2 级给它们分配的 CPU 百分比。在这个规划中还包括 SYS_GROUP（它是 Oracle 最初为用户 SYS 和 SYSTEM 定义的组）和 OTHER_GROUPS。

例子：CREATE_SIMPLE_PLAN 过程的使用

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_SIMPLE_PLAN(SIMPLE_PLAN => 'simple_plan1',
CONSUMER_GROUP1 => 'mygroup1', GROUP1_CPU => 80,
```



```
CONSUMER_GROUP2 => 'mygroup2', GROUP2_CPU => 20);  
END;
```

执行上面的语句将创建下面的规划:

用户组	第 1 级	第 2 级	第 3 级
SYS_GROUP	100%		
mygroup1		80%	
mygroup2		20%	
OTHER_GROUPS			100%

27.4 创建复杂的资源规划

这一节介绍在需要创建更加复杂的资源规划的情形下可以使用的动作和 DBMS_RESOURCE_MANAGER 过程。这一节包含以下内容:

- 使用暂挂区域创建规划模式
- 创建资源规划
- 创建资源用户组
- 指定资源规划指令

27.4.1 使用暂挂区域创建规划模式

创建或修改规划模式要做的第一件事就是创建一个暂挂区域。在这个暂存区中可以实施修改并在修改内容变成活动的之前确认所做的修改。

27.4.1.1 创建暂挂区域

要创建暂挂区域, 可以使用以下语句:

```
EXEC DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA;
```

实际上, 此时真正发生的动作是正在使得暂挂区域变成活动的, 并且正在将所有现有的或活动的规划模式装载到暂挂区域, 这样可以更新规划或添加新的规划。活动规划模式就是那些已经存储在数据字典中的模式, 数据库资源管理器可以使用这些模式。如果打算在没有激活(创建)暂挂区域的情况下更新规划或添加新规划, 那么将收到错误消息, 该消息通知用户暂挂区域不是活动的。

可以利用视图来检查所有活动的以及挂起的资源规划模式。本章后面的“查看数据库资源管理器信息”一节中列出了这些视图。

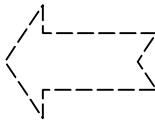
27.4.1.2 确认改变的内容

在任何时候当用户正在暂挂区域中进行修改时, 都可以调用确认过程, 如下所示。

```
EXEC DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA;
```

这个过程检测所做的修改是否有效。必须遵循以下规则, 而且确认过程还要检测这些内容:

- ①规划模式不包含任何循环。



- ②规划指令引用的所有规划和/或资源用户组必须存在。
- ③所有规划都必须有规划指令来指向规划或者资源用户组。
- ④在任何已知的级中所有百分比都必须不大于 100。
- ⑤不能删除某活动实例当前正在作为顶层规划使用的规划。
- ⑥下列规划指令参数只能出现在引用资源用户组（而不是其他资源规划）的规划指令

中：

- PARALLEL_DEGREE_LIMIT_P1
- ACTIVE_SESS_POOL_P1
- QUEUEING_P1
- SWITCH_GROUP
- SWITCH_TIME
- SWITCH_ESTIMATE
- MAX_EST_EXEC_TIME
- UNDO_POOL

⑦在任何活动的规划模式中不得超过 32 个资源用户组。而且，规划最多只能有 32 个子规划。顶层规划的所有级都必须是资源用户组；在规划模式的最低级中规划指令必须引用用户组。

⑧规划和资源用户组不能拥有相同的名称。

⑨在任何活动的规划模式中必须有 OTHER_GROUPS 的规划指令。这样能够确保把资源（由 OTHER_GROUPS 指令指定）分配给不是包含在任何当前活动规划中的用户组的一部分的会话。

如果违反了上面的任何一条规则都将收到一条错误消息。然后可以针对问题做一些修改并再次调用确认过程。

很有可能创建没有规划指令引用的“孤立”用户组。这样可以允许创建当前不会被使用的用户组，也许是将来会执行的规划的一部分。

27.4.1.3 提交改变的内容

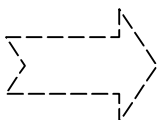
在确认改变的内容之后，调用提交过程激活改变的内容。

```
EXEC DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA;
```

提交过程也执行确认动作，所以不需要单独调用确认过程。然而，在对规划模式做重大修改时，如果不断确认改变的内容，调试问题将会变得更加简单。直到暂挂区域中的所有改变的内容都成功确认，否则不提交（变成活动的）任何改变的内容。

在成功确认和提交改变的内容后，SUBMIT_PENDING_AREA 过程将清除暂挂区域（使得该区域无效）。

注意：即使 VALIDATE_PENDING_AREA 成功，对 SUBMIT_PENDING_AREA 过程的调用也可能失败。例如，在调用 VALIDATE_PENDING_AREA 之后，但是在调用 SUBMIT_PENDING_AREA 之前，实例装载正在删除的规划时就会出现这种情况。



27.4.1.4 清除暂挂区域

也有一个过程用于在任何时候清除暂挂区域。下面这条语句将导致异常中止所有变更操作。

```
EXEC DBMS_RESOURCE_MANAGER.CLEAR_PENDING_AREA;  
在试图再次进行修改之前，必须调用 CREATE_PENDING_AREA 过程。
```

27.4.2 创建资源规划

当创建资源规划时，可以指定下列参数：

参数	说明
PLAN	规划的名称。
COMMENT	任何注释都可以。这个域是可选的。
下列参数不要求指定。默认值就比较合适，并且是此时唯一允许的值。	
CPU_MTH	CPU 资源分配方法。EMPHASIS 是默认值并且是资源规划级别上允许的唯一一种 CPU 方法。
ACTIVE_SESS_POOL_MTH	活动会话池资源分配方法。控制着并发用户的最大值。ACTIVE_SESS_POOL_ABSOLUTE 是默认值并且是唯一可用的方法。
PARALLEL_DEGREE_LIMIT_MTH	用于指定任何操作的并行度界限的资源分配方法。PARALLEL_DEGREE_LIMIT_ABSOLUTE 是默认值并且是唯一可用的方法。
QUEUEING_MTH	排队资源分配方法。控制排队等候执行的会话的次序。FIFO_TIMEOUT 是默认值并且是唯一可用的方法。

Oracle 提供了一种资源规划“SYSTEM_PLAN”，它包含一种简单的结构，这在某些环境下已经足够。将在本章后面的“Oracle 提供的一种规划”一节中加以介绍。

27.4.2.1 创建规划

可以使用 CREATE_PLAN 过程创建规划。下面的语句调用 great_bread 创建了一个规划。选择使用默认的资源分配方法。

```
EXEC DBMS_RESOURCE_MANAGER.CREATE_PLAN (PLAN => 'great_bread', -  
COMMENT => 'great plan');
```

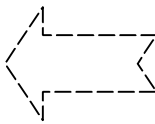
27.4.2.2 更新规划

使用 UPDATE_PLAN 过程更新规划信息。如果没有为 UPDATE_PLAN 过程指定参数，那么这些参数的设置将在数据字典中保持不变。下列语句更新了 COMMENT 参数。

```
EXEC DBMS_RESOURCE_MANAGER.UPDATE_PLAN (PLAN => 'great_bread', -  
NEW_COMMENT => 'great plan for great bread');
```

27.4.2.3 删除规划

DELETE_PLAN 过程可以删除指定的规划以及和规划相关联的规划指令。下面的语句



删除了 `great_bread` 规划和它的指令。

```
EXEC DBMS_RESOURCE_MANAGER.DELETE_PLAN (PLAN => 'great_bread');
资源用户组本身没有被删除，但是这些组将不再与 great_bread 规划有任何关联。
```

`DELETE_PLAN_CASCADE` 过程删除指定的规划以及它的分支（规划指令、子规划、资源用户组）。如果 `DELETE_PLAN_CASCADE` 过程遇到错误，它将回滚，而规划模式则保持不变。

27.4.3 创建资源用户组

当创建资源用户组时，可以指定以下参数：

参数	说明
CONSUMER_GROUP	用户组的名称。
COMMENT	任何注释。
CPU_MTH	用户组的 CPU 资源分配方法。默认值是 <code>ROUND-ROBIN</code> 。这也是资源用户组当前可以使用的唯一一种方法。

这里有两个特殊的用户组，它们经常出现在数据字典中，而且不能对它们进行修改和删除。它们是：

- `DEFAULT_CONSUMER_GROUP`

它是为还没有明确分配初始用户组的所有用户/会话准备的初始用户组。`DEFAULT_CONSUMER_GROUP` 已经将切换权限授予了 `PUBLIC`；因此，会自动给所有用户授予这个用户组切换权限。

- `OTHER_GROUPS`

不能给用户明确地分配这个用户组。`OTHER_GROUPS` 必须有一个在任何活动规划的模式中指定的资源指令。这个组应用于属于用户组但不是当前活动规划模式的一部分的所有会话，包括 `DEFAULT_CONSUMER_GROUP` 也是一样。

此外，作为 Oracle 提供的 `SYSTEM_PLAN` 的一部分，还提供了两个其他的组，`SYS_GROUP` 和 `LOW_GROUP`，`SYSTEM_PLAN` 将在本章后面的“Oracle 提供的一种规划”一节中加以介绍。

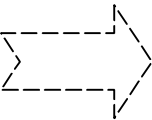
27.4.3.1 创建用户组

使用 `CREATE_CONSUMER_GROUP` 过程可以创建用户组。下面的语句就创建了一个名为 `sales` 的用户组，暂挂区域必须是活动的才能成功执行这条语句。

```
EXEC DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (CONSUMER_GROUP => 'sales', -
COMMENT => 'retail and wholesale sales');
```

27.4.3.2 更新用户组

使用 `UPDATE_CONSUMER_GROUP` 过程来更新用户组信息。如果没有为 `UPDATE_CONSUMER_GROUP` 过程指定参数，那么这些参数设置在数据字典中将保持不变。



27.4.3.3 删除用户组

DELETE_CONSUMER_GROUP 过程可以删除指定的用户组。用户组刚一删除，以删除的组作为初始用户组的所有用户将把 DEFAULT_CONSUMER_GROUP 设置为他们的初始用户组。属于已删除用户组的所有当前运行的会话都将切换到 DEFAULT_CONSUMER_GROUP。

27.4.4 指定资源规划指令

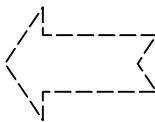
资源规划指令将用户组分配给资源规划并为每一种资源分配方法提供参数。当创建资源规划指令时，可以指定以下参数：

参数	说明
PLAN	资源规划的名称。
GROUP_OR_SUBPLAN	用户组或子规划的名称。
COMMENT	任何注释。
CPU_P1	指定第 1 级上的百分比。为所有 CPU 参数设定的默认值是 NULL。
CPU_P2	指定第 2 级上的百分比。
CPU_P3	指定第 3 级上的百分比。
CPU_P4	指定第 4 级上的百分比。
CPU_P5	指定第 5 级上的百分比。
CPU_P6	指定第 6 级上的百分比。
CPU_P7	指定第 7 级上的百分比。
CPU_P8	指定第 8 级上的百分比。
ACTIVE_SESS_POOL_P1	指定用户组的并发活动会话的最大数目。默认值是 UNLIMITED。
QUEUEING_P1	指定时间（以秒为单位），如果超过了这个时间，执行队列（等待执行）中的作业将被认定为超时。默认值是 UNLIMITED。
PARALLEL_DEGREE_LIMIT_P1	指定任何操作的并行度界限。默认值是 UNLIMITED。
SWITCH_GROUP	指定如果满足其他切换条件会话要被切换到的用户组。默认值是 NULL。
SWITCH_TIME	指定会话在切换到另一个用户组之前可以执行的时间（以秒为单位）。默认值是 UNLIMITED。
SWITCH_ESTIMATE	如果设定为 TRUE，即告诉 Oracle 使用它的执行时间估计在开始执行之前自动切换某个操作的用户组。默认值是 FALSE。
MAX_EST_EXEC_TIME	指定会话允许的最长的执行时间（以秒为单位）。默认值是 UNLIMITED。
UNDO_POOL	设置用户组生成的撤消总量（以千字节 K 为单位）。默认值是 UNLIMITED。

27.4.4.1 创建资源规划指令

使用 CREATE_PLAN_DIRECTIVE 来创建资源规划指令。下面的语句将为规划 great_bread 创建一个资源规划指令。

```
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);
要完成这个规划，与图 27-1 类似，执行以下语句：

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;

```

在这个规划中，用户组 **sales** 对任何操作的最大并行度为 4，此时没有任何其他的用户组限制它们的并行度。而且，不管何时剩下的第 1 级的 CPU 资源，都会分配（100%）给 **OTHER_GROUPS**。

27.4.4.2 更新资源规划指令

使用 **UPDATE_PLAN_DIRECTIVE** 过程可以更新规划指令。下面这个例子就将 CPU 改为分配给资源用户组 **develop**。

```

EXEC DBMS_RESOURCE_MANAGER.UPDATE_PLAN_DIRECTIVE (PLAN => 'great_bread', -
GROUP_OR_SUBPLAN => 'develop', NEW_CPU_P1 => 15);

```

如果没有为 **UPDATE_PLAN_DIRECTIVE** 过程指定参数，那么这些参数的设置将在数据字典中保持不变。

27.4.4.3 删除资源规划指令

要删除资源规划指令，可以使用 **DELETE_PLAN_DIRECTIVE** 过程。

27.4.4.4 资源规划指令如何相互配合

如果有多个引用同一个用户组的资源规划指令，那么以下规则适用于一些特殊的情况：

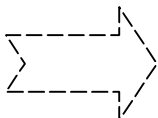
①用户组的并行度界限将是所有输入值中最小的值。

②用户组的活动会话池将是所有输入值的总和，并且排队超时将是所有输入的超时值的最小值。

③如果切换组多于一个，而且切换时间也多于一个，那么数据库资源管理器将选择所有输入值中的最具限制性的值。具体如下：

- **SWITCH_TIME** = *min*（所有输入的切换时间值）
- **SWITCH_ESTIMATE** = TRUE 覆盖 **SWITCH_ESTIMATE** = FALSE

注意：如果两个规划指令指定同一个切换时间，但是却指定不同的切换组，那么将由



数据库资源管理器静态的而且是任意的决定切换到哪个组。

④如果会话因为超出了切换时间而切换到了另一个用户组，那么，即使为新用户组准备的会话池已满，会话也将会执行。

⑤估计的执行时间的最大值将是所有输入值中最具限制性的。具体是：

- $\max_estimated_exec_time = \min(\text{所有输入的 } \max_estimated_exec_time \text{ 值})$

27.5 管理资源用户组

在启用数据库资源管理器之前，必须给用户分配资源用户组。除了提供过程去创建、修改或删除数据库资源管理器使用的组成部分以外，DBMS_RESOURCE_MANAGER 包还包含给用户分配资源用户组的过程。它还提供一些过程以便临时将用户会话切换到另一个用户组。

如同前面在授予数据库资源管理器系统权限时介绍的那样，也可以用 DBMS_RESOURCE_MANAGER_PRIVS 包将切换权限授予另一个用户，之后，这个用户就可以更改它们自己的用户组。

下面讨论的任何一种过程都不使用暂挂区域。

27.5.1 分配一个初始的资源用户组

用户的初始用户组是由用户创建的任何会话最初所属的那个用户组。当创建用户时，自动设置用户的初始用户组为 DEFAULT_CONSUMER_GROUP。

在某个用户组成为用户的初始用户组之前，必须准许用户（或 PUBLIC）切换到这个特定的用户组。这种许可权称为切换权限，我们将在本章后面的“管理切换权限”一节中加以解释。切换到初始用户组的切换权限不能从角色授权给用户。

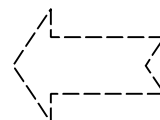
下面的语句说明如何设置用户的初始用户组。

```
EXEC DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SWITCH_CONSUMER_GROUP ('scott',
'sales', -
TRUE);
EXEC DBMS_RESOURCE_MANAGER.SET_INITIAL_CONSUMER_GROUP('scott', 'sales');
```

27.5.2 修改资源用户组

这里有两个过程，它们是 DBMS_RESOURCE_MANAGER 包的一部分，它们允许管理员修改正在运行的会话的资源用户组。这两个过程都可以改变与协调程序的会话相关联的任何并行查询从属会话的用户组。用这些过程所做的修改只适合当前的会话；它们不会持续影响其他的会话。这两个过程也不改变用户的初始用户组。

如果某个用户的会话使用了过多的 CPU 资源，那么管理员可以将用户的用户组改成另一个使用较少 CPU 资源的用户组，而不用删除这个用户的会话。否则将使用自动的用户组切换资源规划指令自动强制执行这个切换。



27.5.2.1 切换会话

SWITCH_CONSUMER_GROUP_FOR_SESS 将导致指定会话立即移入到指定资源用户组中。实际上，这条语句可以提高或降低优先级。下面的语句将把某个特殊会话的资源用户组改成一个新的用户组。这个会话的标识符 (SID) 为 17，会话的序列码 (SERIAL#) 是 12345，这个会话的资源用户组将改为 high_priority 用户组。

```
EXEC DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_SESS ('17', '12345', -
'high_priority');
```

可以使用 V\$SESSION 数据字典视图来查看会话的 SID、会话序列码和当前的资源用户组。

27.5.2.2 切换用户的会话

SWITCH_CONSUMER_GROUP_FOR_USER 过程用一个已知的用户名称来改变所有会话的资源用户组。

```
EXEC DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_USER ('scott', -
'low_group');
```

27.5.3 管理切换权限

使用 DBMS_RESOURCE_MANAGER_PRIVS 包可以授予或撤回用户的切换权限。切换权限赋予了用户可以从当前资源用户组切换到指定资源用户组的权限。这个包也允许撤回切换权限。

真正的切换是由执行 DBMS_SESSION 包中的某个过程来完成的。已经授予切换权限的用户（或者用户所属的某个过程）可以使用 SWITCH_CURRENT_CONSUMER_GROUP 过程切换到另一个资源用户组。这个新的组必须是特别授权用户要切换到的组。

27.5.3.1 授予切换权限

下面的例子授予了切换到用户组的权限。用户 scott 被授予切换到用户组 bug_batch_group 的权限。

```
EXEC DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SWITCH_CONSUMER_GROUP ('scott', -
'bug_batch_group', TRUE);
```

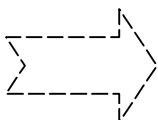
也可以授予用户 scott 切换到其他不同于 bug_batch_group 用户组的权限。

如果同意了用户切换到某个特殊的用户组，那么用户可以从当前用户组切换到新的用户组。

如果同意了角色切换到某个特殊的资源用户组，那么已经授予这个角色并已经启用这个角色的任何一个用户都会立即从当前用户组切换到新的用户组。

如果同意了 PUBLIC 切换到某个特殊的用户组，那么任何用户都可以切换到那个新的用户组。

如果 GRANT_OPTION 参数设置为 TRUE，那么被授予切换用户组权限的用户也可以授予该用户组用户切换到另外的用户组的切换权限。



27.5.3.2 撤消切换权限

下面这个例子将撤消用户 `scott` 的切换到用户组 `bug_batch_group` 的权限。

```
EXEC DBMS_RESOURCE_MANAGER_PRIVS.REVOKE_SWITCH_CONSUMER_GROUP ('scott', -
'bug_batch_group');
```

如果撤消了用户的切换到某个特殊用户组的切换权限，那么该用户以后任何打算切换到这个用户组的动作都将失败。如果撤消了用户的初始用户组，那么登录时那个用户将自动成为 `DEFAULT_CONSUMER_GROUP` 的一部分。

如果撤消了角色的切换到某个用户组的切换权限，那么，只能通过这个角色切换到该用户组的任何用户在以后都不能切换到这个用户组。

如果撤消了 `PUBLIC` 的切换到某个用户组的切换权限，那么，除了通过直接方式或 `PUBLIC` 方式明确分配切换权限的那些用户，任何其他用户在以后都不能切换到这个用户组。

27.5.3.3 使用 `DBMS_SESSION` 包来切换用户组

如果授予了切换权限，用户就可以使用 `DBMS_SESSION` 包中的 `SWITCH_CURRENT_CONSUMER_GROUP` 过程切换当前的用户组。

这个过程使得拥有切换权限的用户可以切换到某个用户组。如果调用这个过程的是另外一个过程，那么这个过程允许拥有切换权限并且是调用它的过程的所有者才能切换到某个用户组。

下面列出了这个过程的几个参数：

参数	说明
<code>NEW_CONSUMER_GROUP</code>	用户要切换到的用户组。
<code>OLD_CONSUMER_GROUP</code>	这是一个输出参数。它存储了用户切换之前的用户组的名称。以后还可以用它切换回原来的用户组。
<code>INITIAL_GROUP_ON_ERROR</code>	控制发生切换错误时的行为。如果为 <code>TRUE</code> ，那么在发生错误时，用户将切换到初始用户组。如果为 <code>FALSE</code> ，将产生错误。

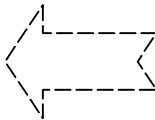
下面这个例子说明了切换到新用户组的过程。通过打印输出参数 `old_group` 的值，将解释原来的用户组名称如何保存。

```
SET serveroutput on
DECLARE
old_group varchar2(30);
BEGIN
DBMS_SESSION.SWITCH_CURRENT_CONSUMER_GROUP('sales', old_group, FALSE);
DBMS_OUTPUT.PUT_LINE('OLD GROUP = ' || old_group);
END;
```

下面这一行是输出的内容：

```
OLD GROUP = DEFAULT_CONSUMER_GROUP
```

可以使用 `PL/SQL` 应用程序中的 `DBMS_SESSION` 包，这样可以允许应用程序动态地



改变用户组或有效地改变优先级。

注意：数据库资源管理器也可以工作在用普通的用户名登录到某应用程序的环境。可以在会话启动时或作为特殊的模块调用 DBMS_SESSION 包来切换会话的用户组分配。

27.6 启用数据库资源管理器

通过设置初始化参数 RESOURCE_MANAGER_PLAN 可以启用数据库资源管理器。这个参数指定顶层规划、确定实例使用的规划模式。如果没有用这个参数指定规划，数据库资源管理器将不是活动的。下面这个例子激活数据库资源管理器并分配顶层规划为 mydb_plan。

```
RESOURCE_MANAGER_PLAN = mydb_plan
```

也可以使用 ALTER SYSTEM 语句激活或停用数据库资源管理器，或者改变当前的顶层规划。在这个例子中，顶层规划指定为 mydb_plan。

```
ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = mydb_plan;
```

如果指定的规划在数据字典中不存在，将返回一条错误消息。

要停用数据库资源管理器，发布下列语句即可：

```
ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = '';
```

27.7 将所有内容结合在一起：数据库资源管理器的例子

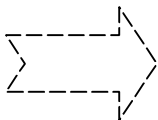
这一节提供一些资源规划模式的例子。包括以下这些例子：

- 多级模式的例子
- 使用几种资源分配方法的例子
- Oracle 提供的一种规划

27.7.1 多级模式的例子

下面的语句将创建一个多级模式，如图 27-3 所示。这个例子使用的是默认规划和资源用户组方法。

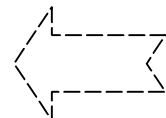
```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'bugdb_plan',
COMMENT => 'Resource plan/method for bug users sessions');
DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'maildb_plan',
COMMENT => 'Resource plan/method for mail users sessions');
DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => '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 maintain
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 2', 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',

```

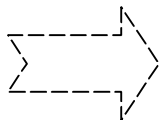


```

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;

```

因为确认动作是在 SUBMIT_PENDING_AREA 中暗中进行的，所以前面对 VALIDATE_PENDING_AREA 的调用是可选的。



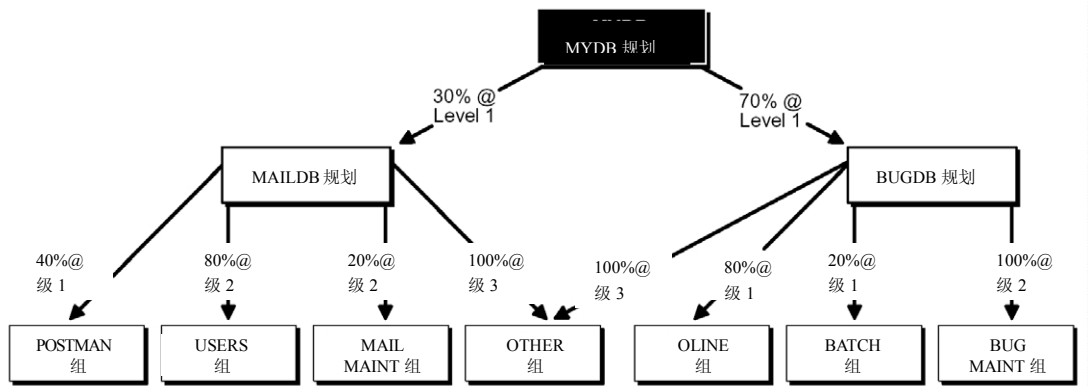


图 27-3 多级模式

27.7.2 使用几种资源分配方法的例子

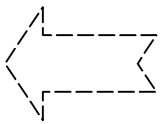
这个例子中的规划可以提供给支持打包的 ERP（企业资源规划）或 CRM（用户关系管理）的数据库。在这种环境中的工作有非常大的不同。这个例子混合了短小的事务和快速的查询，还结合了包括大量并行查询的更长运行时间的批处理作业。它的目的就是当允许批处理作业并行运行时给 OLTP（联机事务处理）以最好的反应时间。

规划总结如下表。

组	CPU 资源分配%	活动会话池参数	自动切换参数	执行时间的最大估计值	撤消池
oltp	Level1:80%		切换到组:batch 切换时间:3 使用估计:TRUE		大 小:200K
batch	Level2:100%	池大小:5 超时:600		时间:3600	
OTHER_GROUPS	Level3:100%				

下面的语句将创建上面的规划，规划名称为 `erp_plan`：

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'erp_plan',
COMMENT => 'Resource plan/method for ERP Database');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'oltp',
COMMENT => 'Resource consumer group/method for OLTP jobs');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'batch',
COMMENT => 'Resource consumer group/method for BATCH jobs');
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'erp_plan',
GROUP_OR_SUBPLAN => 'oltp', COMMENT => 'OLTP sessions', CPU_P1 => 80,
SWITCH_GROUP => 'batch', SWITCH_TIME => 3, SWITCH_ESTIMATE => TRUE,
```



```
UNDO_POOL => 200);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'erp_plan',
GROUP_OR_SUBPLAN => 'batch', COMMENT => 'BATCH sessions', CPU_P2 => 100,
ACTIVE_SESS_POOL_P1 => 5, QUEUEING_P1 => 600,
MAX_EST_EXEC_TIME => 3600);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'erp_plan',
GROUP_OR_SUBPLAN => 'OTHER_GROUPS', COMMENT => 'mandatory', CPU_P3 => 100);
DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
```

27.7.3 Oracle 提供的一种规划

Oracle 提供一种默认的资源管理器规划即 SYSTEM_PLAN，它将给出系统会话的优先级。定义 SYSTEM_PLAN 如下：

CPU 资源分配			
资源用户组	第 1 级	第 2 级	第 3 级
SYS_GROUP	100%	0%	0%
OTHER_GROUPS	0%	100%	0%
LOW_GROUP	0%	0%	100%

在这个规划中 Oracle 提供以下内容：

- SYS_GROUP 是用户 SYS 和 SYSTEM 的初始用户组。
- OTHER_GROUPS 应用于属于某个用户组的所有会话，而这个用户组不是当前活动规划模式的一部分。
- LOW_GROUP 提供在这个规划中比 SYS_GROUP 和 OTHER_GROUPS 优先级还低的组。由用户决定哪些用户会话将成为 LOW_GROUP 的一部分。按 PUBLIC 形式授予这个组切换权限。

可以使用这些组，或者不使用这些组，也可以修改或删除这些组。

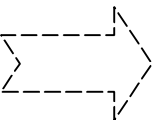
如果适合用户的环境，可以使用这个 Oracle 提供的简单的规划。

27.8 监控和协调数据库资源管理器

要有效地监控和协调数据库资源管理器，就必须设计一个有代表性的环境。在大型环境中数据库资源管理器可以发挥得很好，在这种环境下系统利用率会比较高。如果在系统上放置了不足的装载量，那么测试出的 CPU 分配情况将与在活动资源规划中指定的分配方案有较大的不同。

27.8.1 创建环境

要创建一个有代表性的环境，就必须有足够的装载量（对 CPU 资源的要求）以免 CPU 资源缺乏。如果遵循了以下规则，那么测试环境将生成实际的（经过测试的）资源分配，



而这种分配满足在活动资源规划中指定的那些内容。

①创建生成足够装载量所需的最小数目的并发运行进程。它要大于以下要求：

- 每个用户组有 4 个进程
- 每个用户组有 $1.5 \times$ (处理进程的数目)。如果结果不是整数，进行舍入。

②每一个进程都必须有能力消耗分配给它所属用户组的所有 CPU 资源。继续编写程序。如下所示：

```
BEGIN
DECLARE
m NUMBER;
BEGIN
FOR i IN 1..100000 LOOP
FOR j IN 1..100000 LOOP
m := sqrt(4567);
END LOOP;
END LOOP;
END;
END;
/
```

27.8.2 为什么需要生成期望的结果

当每个组按照要求的量获得自己的 CPU 资源时，数据库资源管理器首先调查系统的最大吞吐量，而不是按要求的比例强制分配。例如，考虑以下情形：

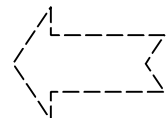
- 每个用户组只有一个可以运行的进程。
- 每个进程不断地运行。
- 有 4 个 CPU。

在这种情况下，不管在活动资源规划中指定的分配量是多少，测试出的每个用户组的 CPU 分配量都将是 25%。

另一个因素决定了上面第 (1) 步中的计算。与操作系统级别上的调度相似的处理程序可能误报 CPU 在未充分利用的系统上的分配量。这将在下面的段落中加以解释。

在并发运行的进程数目达到一定标准之前，典型的操作系统调度算法将避免完全使用 CPU 资源。数据库资源管理器通过限制运行进程的数目来控制 CPU 的使用。通过决定在多长时间允许运行哪些进程的办法，数据库资源管理器控制着 CPU 资源的分配。当 CPU 有可以利用的资源并且完全利用了其他进程时，操作系统将会把进程移植到未充分利用的处理程序中，但并不是立即这么做。

由于处理程序的相似性，操作系统将等待（一段时间）移植进程，但仍然期望调度另一个进程运行，而不是强制进程从一个 CPU 移植到另一个 CPU。在一个完全装载的系统上，如果有足够多处于等待状态的进程，就可以采用这种策略。在大型环境中，因为清除当前 CPU 缓存然后装载新的 CPU 缓存代价太高，所以处理程序最重要的是提高性能。因为在



多数平台上进程都有相似的处理程序，所以每个用户组必须运行比 CPU 还多的进程。否则，不可能完全使用系统。

27.8.3 监控结果

使用 V\$RSRC_CONSUMER_GROUP 视图可以监控 CPU 的使用情况。这个视图提供每个用户组中所有会话使用的 CPU 时间的累积量。它还提供有助于协调的其他方法。

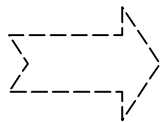
```
SQL> SELECT NAME, CONSUMED_CPU_TIME FROM V$RSRC_CONSUMER_GROUP;  
NAME CONSUMED_CPU_TIME  
-----  
OTHER_GROUPS 14301  
TEST_GROUP 8802  
TEST_GROUP2 0  
3 rows selected.
```

27.9 查看数据库资源管理器信息

下面这个表列出了与数据库资源管理器有关的一些视图：

视图	说明
DBA_RSRC_CONSUMER_GROUP_PRIVS USER_RSRC_CONSUMER_GROUP_PRIVS DBA_RSRC_CONSUMER_GROUPS DBA_RSRC_MANAGER_SYSTEM_PRIVS USER_RSRC_MANAGER_SYSTEM_PRIVS	DBA 视图列出了所有资源用户组以及已经授权的用户和角色。USER 视图列出了授权给用户的所有资源用户组。列出数据库中存在的的所有资源用户组。 DBA 视图列出了所有已经授予数据库资源管理器权限的用户和角色。USER 视图列出了所有为了使用 DBMS_RESOURCE_MANAGER 包而被授予系统权限的用户。
DBA_RSRC_PLAN_DIRECTIVES DBA_RSRC_PLANS DBA_USERS USERS_USERS	列出了数据库中存在的的所有资源规划指令。 列出了数据库中存在的的所有资源规划。 DBA 视图包含数据库的所有用户的信息。特别是，对于数据库资源管理器来说，它还包含用户的初始的资源用户组。USER 视图包含当前用户的信息，特别是，对于数据库资源管理器来说，它还包含当前用户的初始的资源用户组。
V\$ACTIVE_SESS_POOL_MTH V\$PARALLEL_DEGREE_LIMIT_MTH V\$QUEUEING V\$RSRC_CONSUMER_GROUP V\$RSRC_CONSUMER_GROUP_CPU_MTH V\$RSRC_PLAN V\$RSRC_PLAN_CPU_MTH V\$SESSION	显示所有可用的活动会话池资源分配方法。 显示所有可用的并行度界限资源分配方法。 显示所有排队等候的资源分配方法。 显示活动资源用户组的信息。这个视图可以用于协调。 显示资源用户组可以利用的所有 CPU 资源分配方法。 显示所有当前活动资源规划的名称。 显示资源规划可以利用的所有 CPU 资源分配方法。 列出了每个当前会话的会话信息。特别是，列出了每个当前会话的资源用户组名称。

可以使用这些视图查看权限，查看规划模式，或者可以监控这些视图以便收集协调数



数据库资源管理器的信息。下面有几个例子介绍这些视图的使用。

27.9.1 查看授予用户或角色的用户组

DBA_RSRC_CONSUMER_GROUP_PRIVS 视图显示授予用户或角色的用户组。特别是，显示了允许用户或角色属于或切换到的组。例如，在下面的视图中，用户 scott 可以属于用户组 market 或 sales，该用户有能力将其他用户分配到 sales 组而不是 market 组。这两个组都不是用户 scott 的初始用户组。

```
SQL> SELECT * FROM DBA_RSRC_CONSUMER_GROUP_PRIVS;
GRANTEE GRANTED_GROUP GRA INI
```

```
-----
PUBLIC DEFAULT_CONSUMER_GROUP YES YES
PUBLIC LOW_GROUP NO NO
SCOTT MARKET NO NO
SCOTT SALES YES NO
SYSTEM SYS_GROUP NO YES
```

Scott 被授予了使用 DBMS_RESOURCE_MANAGER_PRIVS 包切换到这些组的能力。

27.9.2 查看规划模式信息

在这个例子中，使用 DBA_RSRC_PLANS 视图来显示数据库中定义的所有资源规划。显示的所有这些规划都是活动的，这意味着它们不是在暂挂区域中。

```
SQL> SELECT PLAN,COMMENTS,STATUS FROM DBA_RSRC_PLANS;
PLAN COMMENTS STATUS
```

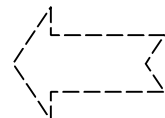
```
-----
SYSTEM_PLAN Plan to give system sessions priority ACTIVE
BUGDB_PLAN Resource plan/method for bug users sessions ACTIVE
MAILDB_PLAN Resource plan/method for mail users sessions ACTIVE
MYDB_PLAN Resource plan/method for bug and mail users sessions ACTIVE
GREAT_BREAD Great plan for great bread ACTIVE
ERP_PLAN Resource plan/method for ERP Database ACTIVE
6 rows selected.
```

27.9.3 查看会话的当前用户组

可以使用 V\$SESSION 视图来显示当前分配给会话的用户组。

```
SQL> SELECT SID,SERIAL#,USERNAME,RESOURCE_CONSUMER_GROUP FROM V$SESSION;
SID SERIAL# USERNAME RESOURCE_CONSUMER_GROUP
```

```
-----
.
.
.
```

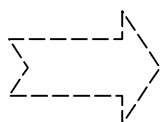


```
11 136 SYS SYS_GROUP
13 16570 SCOTT SALES
10 rows selected.
```

27.9.4 查看当前活动的规划

在这个例子中将 `myd_bplan` 设置为顶层规划，这个规划由本章前面的“多级模式例子”一节中的语句创建。查询 `V$RSRC_PLAN` 视图将得到当前活动的规划。

```
SQL> ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = mydb_plan;
System altered.
SQL> SELECT * FROM V$RSRC_PLAN;
NAME
-----
MYDB_PLAN
MAILDB_PLAN
BUGDB_PLAN
```



第 6 部分 分布式数据库的管理

第 6 部分讨论分布式数据库环境的管理。包括如下章节：

- 第 28 章 分布式数据库的概念
- 第 29 章 管理分布式数据库
- 第 30 章 开发用于分布式数据库系统的应用
- 第 31 章 分布式事务的概念
- 第 32 章 管理分布式事务

第 28 章 分布式数据库的概念

本章描述 Oracle 的分布式数据库结构的基本概念和技术。包括如下内容：

- 分布式数据库的结构
- 数据库链接
- 分布式数据库的管理
- 分布式系统中的事务处理
- 分布式数据库应用的开发
- 字符集的支持

28.1 分布式数据库的结构

分布式数据库系统允许应用访问本地和远程数据库中的数据。在同构分布式数据库系统中，每个数据库都是一个 Oracle 数据库。在异构分布式数据库系统中，至少有一个数据库是非 Oracle 数据库。分布式数据库使用客户/服务器结构来处理信息请求。

本节包含如下内容：

- 同构分布式数据库系统
- 异构分布式数据库系统
- 客户/服务器数据库结构

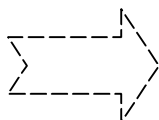
28.1.1 同构分布式数据库系统

同构分布式数据库系统，是一个由一个或多个机器上的，两个或多个 Oracle 数据库组成的网络。图 28-1 表示了一个连接着三个数据库 `hq`，`mfg`，`sales` 的分布式系统。在一个单独的分布式环境中，一个应用能同时访问或更改几个数据库中的数据。例如，来自本地数据库 `mfg` 上的一个制造商客户的一个单独的查询，可以检索本地数据库上的 `products` 表中的和远程 `hq` 数据库上的 `dept` 表中的联结后的数据。

对于客户应用而言，数据库的位置和平台是透明的。还可以给分布式系统中的远程对象创建同义词，以便用户可以用与访问本地对象相同的语法来访问它们。例如，如果连接到 `mfg` 数据库，但是想访问 `hq` 数据库中的数据，就在 `mfg` 上为远程 `dept` 表创建一个同义词，以便允许发布如下查询：

```
SELECT * FROM dept;
```

用这种方法，分布式系统会给出本身数据访问的位置。`Mfg` 上的用户不必知道他们所访问的数据是存储在远程数据库中的。



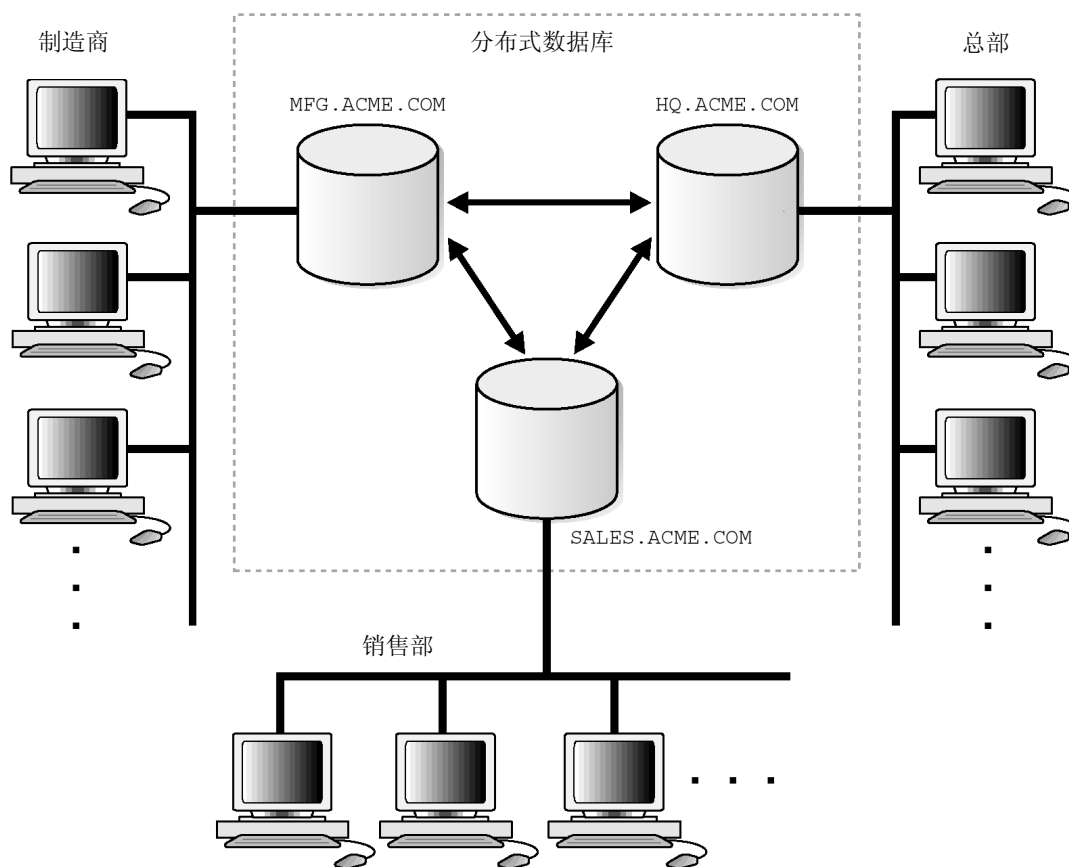


图 28-1 同构分布式数据库系统

Oracle 分布式数据库系统可以融合不同版本的 Oracle 数据库。所有被支持的 Oracle 版本都能参与到分布式数据库系统中。然而，工作在分布式数据库中的应用，就必须要了解该系统中每个节点上可用的功能了。一个分布式数据库应用不能要求一个 Oracle7 数据库能够理解只能在 Oracle9i 中使用的 SQL 扩展。

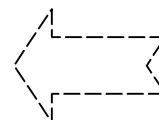
28.1.1.1 分布式数据库与分布式处理

分布式数据库和分布式处理这两个术语是紧密相连的，但它们有截然不同的含义。

分布式数据库 它是在分布式系统中的数据库的集合。对应用而言，它们就像一个单独的数据源。

分布式处理 它是当应用把它的任务分配到网络中的不同计算机上时发生的操作。例如，数据库应用一般会将前端表示的任务分配给客户计算机，并让后端数据库服务器来管理对数据库的共享的访问。其结果是，分布式数据库应用处理系统就常常被称为客户/服务器数据库应用系统。

Oracle 分布式数据库系统使用分布式处理结构。例如，当一个 Oracle 数据库服务器请求由另外的 Oracle 数据库服务器管理的数据时，它就是一个客户。



28.1.1.2 分布式数据库与复制式数据库

分布式数据库系统和数据库复制的术语是有关系的，但截然不同。在一个纯粹的（也就是说未被复制）分布式数据库中，系统管理所有数据的一个拷贝并且支持数据库对象。一般地，分布式数据库应用使用分布式事务来访问本地的和远程的数据，并且实时修改全局数据库。

注意：本书只讨论纯粹的分布式数据库。

术语复制是指在属于分布式系统的多个数据库中备份和管理数据库对象的操作。同时，复制依赖于分布式数据库技术，数据库复制技术给应用提供了一种在纯粹的分布式数据库环境中不可能提供的好处。

更常见的是，复制被用于改善本地数据库的性能，并保护应用的可用性，因为有了可以改变的数据访问选项了。例如，一个应用可能通常访问本地数据库，而不是远程服务器，以减小网络流量并实现最佳性能。此外，如果本地服务器出现故障了，但其他的具有复制数据的服务器还可以被访问时，应用就还能继续运行，

28.1.2 异构分布式数据库系统

在一个异构分布式数据库系统中，至少有一个数据库是非 Oracle 系统。对于应用来说，该异构分布式数据库系统就像是一个单独的、本地的、Oracle 数据库。本地 Oracle 数据库服务器隐藏了数据的分布和异构的情况。

Oracle 数据库服务器，通过使用 Oracle 异构服务（Oracle Heterogeneous Services）并结合一个代理，来访问非 Oracle 系统。如果使用 Oracle 透明网关（Transparent Gateway）来访问非 Oracle 数据存储，那么该代理就是一个系统特殊的应用。例如，如果在 Oracle 分布式系统中包含一个 Sybase 数据库，那么就需要获得一个 Sybase 特殊的透明网关，以便该系统中的 Oracle 数据库可以与它通信。

可选择地，可以使用普通的连通性（generic connectivity）来访问非 Oracle 数据存储，只要该非 Oracle 系统支持 ODBC 或 OLE DB 协议。

注意：除了本章所介绍的内容之外，本书不再讨论 Oracle 异构服务。

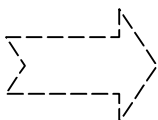
28.1.2.1 异构服务

异构服务（HS）是 Oracle 数据库服务器和用于当前的 Oracle 透明网关产品的运做技术的一个集成后的组件。HS 为 Oracle 网关产品和其他的异构访问工具提供了公共的体系结构和管理机制。同样，它也为使用早期的 Oracle 透明网关版本的很多用户提供了向上的兼容性。

28.1.2.2 透明网关代理

对于所访问的每一个非 Oracle 系统，异构服务都能使用透明网关代理，来与指定的非 Oracle 系统进行接口。该代理是特别针对该非 Oracle 系统的，所以每种类型的系统都需要一个不同的代理。

透明网关代理有助于 Oracle 数据库和非 Oracle 数据库之间的通信，并且在 Oracle 数据



库服务器中使用异构服务组件。该代理代表 Oracle 数据库服务器，在非 Oracle 系统中执行 SQL 语句和事务的请求。

28.1.2.3 普通的连通性

普通的连通性使你可以通过使用异构服务 ODBC 代理或异构服务 OLE DB 代理（它们都作为标准特征，包含在 Oracle 数据库产品中），而连接到非 Oracle 数据存储上。任何与 ODBC 或 OLE DB 标准相兼容的数据源，都能使用普通的连通性代理来访问。

普通的连通性的优点是不需要购买或配置单独的系统特殊的代理。使用能与该代理接口的 ODBC 或 OLE DB 驱动程序。当然，某些数据访问特性只适用于透明网关代理。

28.1.3 客户/服务器结构

数据库服务器就是管理数据库的 Oracle 软件，客户就是请求服务器中的信息的应用。网络中的每一台计算机都是一个能容纳一个或多个数据库的节点。根据不同情形，分布式数据库系统中的每个节点都能担当客户、服务器、或二者兼有。

在图 28-2 中，当发布一条语句访问它的本地数据时，hq 数据库的主机就表现为数据库服务器（例如，每个事务的第二条语句是一条访问本地 dept 表的语句），但是，当它发布一条访问远程数据的语句时，它就表现为客户（例如，每个事务的第一条语句是一条访问 sales 数据库中的远程表 emp 的语句）。

客户能够直接地或间接地连接到数据库服务器。当一个客户连接到一个服务器并且访问那个服务器中的信息时，就产生了直接连接。例如，如果连接到 hq 数据库，并且访问该数据库中的 dept 表，如图 28-2 所示，你可以发布如下语句：

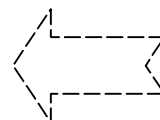
```
SELECT * FROM dept;
```

该查询是直接的，因为你现在没有访问远程数据库中的对象。

相反，当一个客户连接到一个服务器，然后访问包含在不同服务器中的数据库的信息时，就产生了间接连接。例如，如果连接到 hq 数据库，但是又访问远程 sales 数据库中的 emp 表，如图 28-2 所示，你可以发布如下语句：

```
SELECT * FROM emp@sales;
```

该查询是间接的，因为你现在访问的对象不是在你直接连接的数据库上。



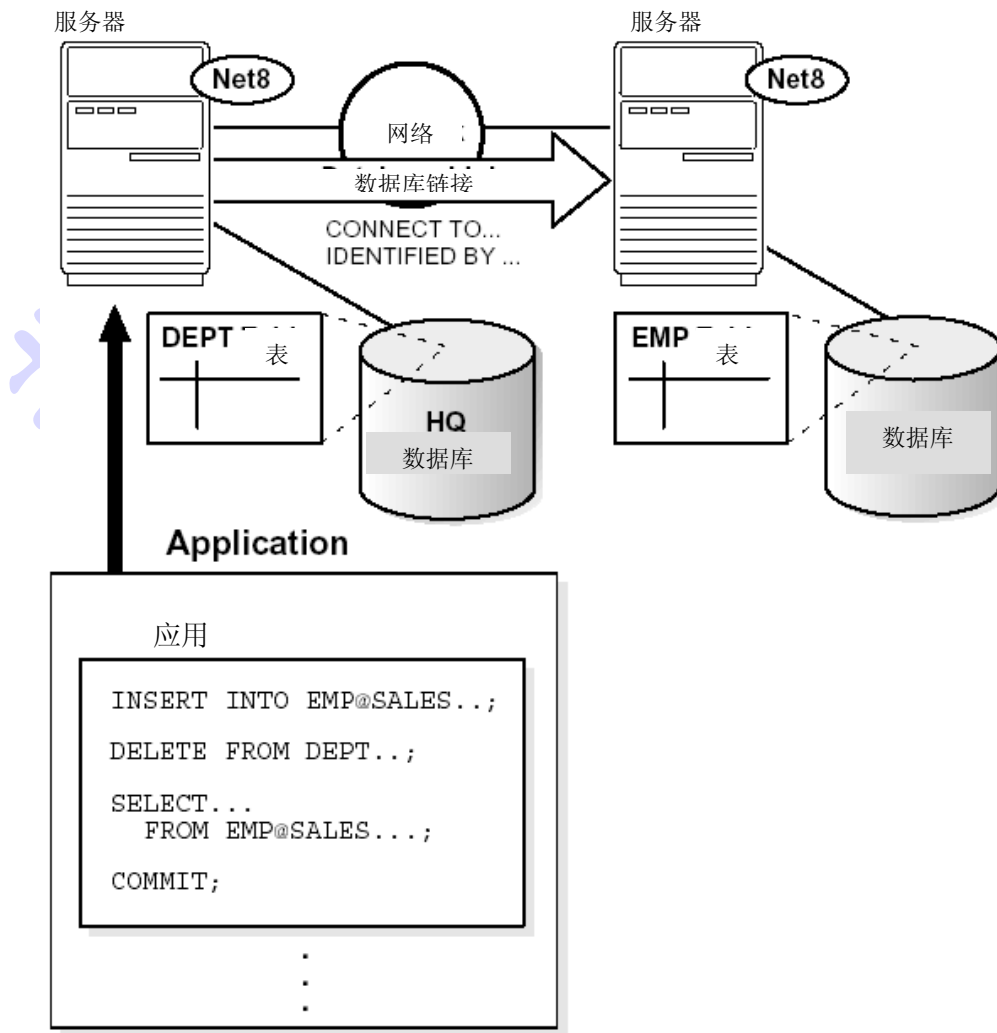


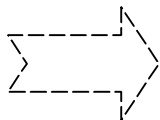
图 28-2 一个 Oracle 分布式数据库系统

28.2 数据库链接

分布式数据库系统中的一个核心的概念是数据库链接。数据库链接是在两个物理数据库服务器之间的连接，该链接允许客户像访问一个逻辑数据库一样来访问它们。

本节包括如下内容：

- 什么是数据库链接
- 为什么使用数据库链接
- 数据库链接中的全局数据库名
- 数据库链接的名称
- 数据库链接的类型
- 数据库链接的用户



- 数据库链接的创建：例子
- 模式对象和数据库链接
- 数据库链接的限制

28.2.1 什么是数据库链接

数据库链接是一个指针，它定义了一个从一个 Oracle 数据库服务器到另一个数据库服务器的单向的通信路径。该链接指针在数据字典表中实际上是被定义为一个入口。为了访问该链接，你必须连接到包含该数据字典入口的本地数据库。

因为连接到本地数据库 A 上的一个客户，可以使用存储在数据库 A 中的链接来访问远程数据库 B 中的信息，但是连接到数据库 B 中的用户就不能使用相同的链接来访问数据库 A 中的数据了，因此，数据库链接连接是单向的。如果数据库 B 中的本地用户，想去访问数据库 A 中的数据，那么他们就必须定义一个链接，并将该链接存储在数据库 B 的数据字典中。

数据库链接连接允许本地用户访问远程数据库中的数据。为了产生这种连接，分布式数据库系统中的每个数据库，就必须在网络域中有惟一的全局数据库名。该全局数据库名惟一地标识分布式系统中的数据库服务器。

图 28-3 说明了用户 scott 用全局名称 hq.acme.com 访问远程数据库 emp 表的例子：

数据库链接或者是私有的或者是公共的。如果它们是私有的，那么就只有创建该链接的用户可以访问；如果它们是公共的，那么所有的数据库用户都可以访问。

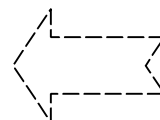
各种数据库链接的主要不同点，就是连接到远程数据库的方式。用户通过如下类型的链接来访问远程数据库。

已连接用户链接 用户以他们自己的身份进行连接，这就意味着他们在远程数据库上必须有一个帐户，该帐户与他们在本地数据库中的帐户具有相同的用户名。

固定用户链接 用户使用在链接中所参照的用户名和口令进行连接。例如，如果 Jane 使用用户名和口令为 scott/tiger 的、连接到 hq 数据库的固定用户链接，那么她就按 scott 进行连接。Jane 就会拥有在 hp 中直接授予给 scott 的所有权限，和 scott 在 hq 数据库中被授予的所有默认的角色。

当前用户链接 用户按全局用户进行连接。一个本地用户可以在存储过程的上下文中按全局用户进行连接——不需要在链接定义中保存该全局用户的名称。例如，Jane 可以访问 Scott 写的过程，访问 hq 数据库中的 Scott 的帐户和 Scott 的模式。当前用户连接是 Oracle 高级安全的一个方面。

使用 CREATE DATABASE LINK 语句来创建数据库链接。创建了一个链接之后，就可以用它来指定 SQL 语句中的模式对象了。



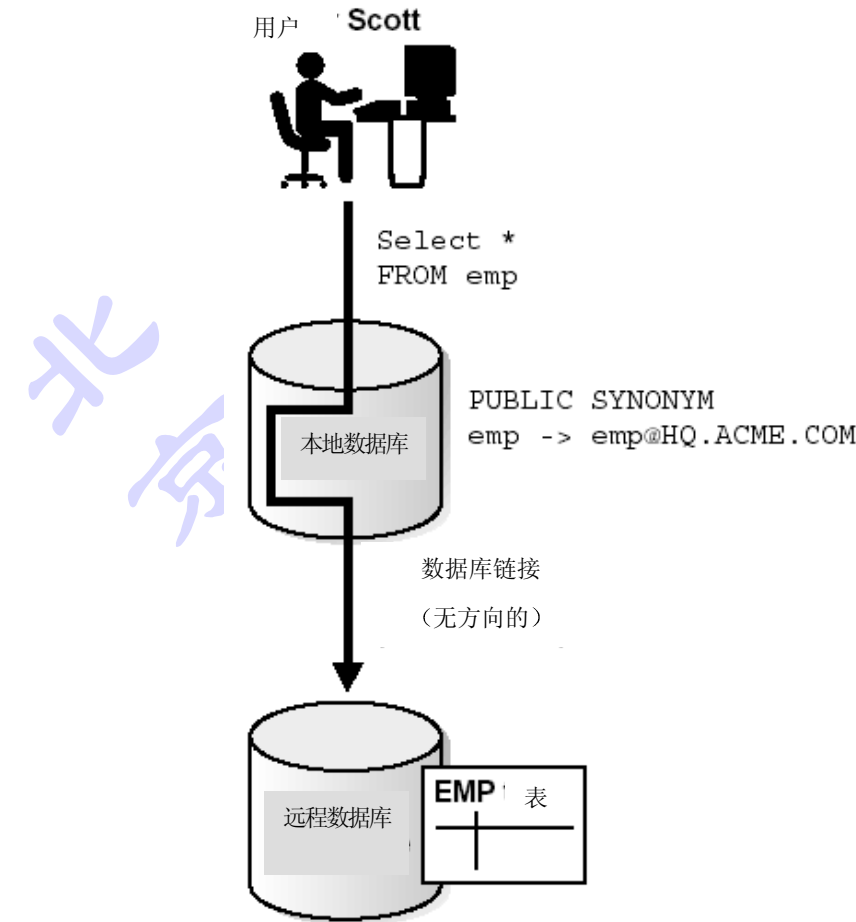


图 28-3 数据库链接

28.2.2 什么是共享数据库链接

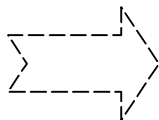
共享数据库链接是本地服务器进程和远程数据库之间的一个链接。该链接是共享的，因为多个客户进程可以同时使用这个相同的链接。

当通过数据库链接，将一个本地数据库连接到远程数据库时，该数据库既可以以专用模式又可以以共享服务器模式运行。下面的表说明了这种可能性：

本地数据库模式	远程数据库模式
专用	专用
专用	共享服务器
共享服务器	专用
共享服务器	共享服务器

共享数据库链接能以上述四种配置中的任何一种方式存在。共享数据库链接与标准数据库链接有如下几个方面的差别：

- 通过一个数据库链接来访问相同模式对象的不同用户，可以共享一个网络连接。
- 当一个用户需要建立从一个特殊的服务器进程到一个远程服务器的连接时，该进



程就可以重新使用已经建立的，到该远程服务器的连接。如果已经在具有相同数据库链接的相同的服务器进程上建立了连接——可能是在不同的会话中，就会重新使用该连接。而在非共享数据库链接中，多路会话是不能共享一个连接的。

- 当在共享服务器配置中使用共享数据库链接时，就会在本地服务器的共享服务器进程之外，直接地建立网络连接。对于本地共享服务器上的非共享数据库链接而言，本地调度程序可能已经建立了该连接，这需要针对本地调度程序的上下文转换，并且需要数据通过该调度程序。

28.2.3 为何使用数据库链接

数据库链接的最大优点就是，它允许用户访问远程数据库中的其他用户的对象，所以这些用户就具有了这些对象的拥有者的权限。换句话说，一个本地用户可以访问一个到远程数据库的链接，而不必是该远程数据库中的一个用户。

例如，假设雇员们将开支报告提交给了应付款（Accounts Payable, A/P）应用，并且进一步假设一个使用 A/P 应用的用户需要从 hq 数据库中检索有关雇员们的信息。那么该 A/P 用户就应该能够连接到 hq 数据库，并且执行远程 hq 数据库中的检索所需信息的存储过程。而该 A/P 用户并不需要是 hq 数据库的用户才能做这件事；当然，该用户仅仅能够访问该存储过程所能够访问的 hp 数据库中的信息。

数据库链接允许将有限的访问远程数据库的权限授予给本地用户。通过使用当前的用户链接，就能创建集中管理的全局用户，该用户的口令信息对管理员和非管理员来说都是隐藏的。例如 A/P 用户能按 scott 访问 hq 数据库，但与固定用户链接不同，scott 的信任书不会被存储在数据库用户能够看见他们的地方。

通过使用固定用户数据库链接，就能创建非全局用户，其口令信息按非加密格式被存储在 LINKS 数据字典表中。固定用户链接比较容易创建并只需要低的系统开销，因为不需要 SSL 或目录，但是由于在数据字典中存储了口令信息产生了安全风险。

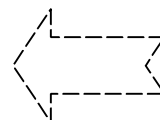
28.2.4 数据库链接中的全局数据库名

为了理解数据库链接是如何工作的，必须首先要知道什么是全局数据库名。分布式数据库系统中的每个数据库，都惟一地由它的全局数据库名所标识。Oracle 是这么构成数据库的全局数据库名的：将单独的数据库名称（在 DB_NAME 初始化参数中指定）作为前缀加到该数据库的网络域名称（在 DB_DOMAIN 初始化参数中指定）之前。

例如，图 28-4 是一个网络中的数据库的层次排列的示意图。

数据库的名称是这样构成的：由树叶开始沿着一条路径一直到根。例如，mfg 数据库是在 com 域的 acme_tools 分支的 division3 中。则，mfg 的全局数据库名是通过串联该树中的各个节点而创建的，如下所示：

```
mfg.division3.acme_tools.com
```



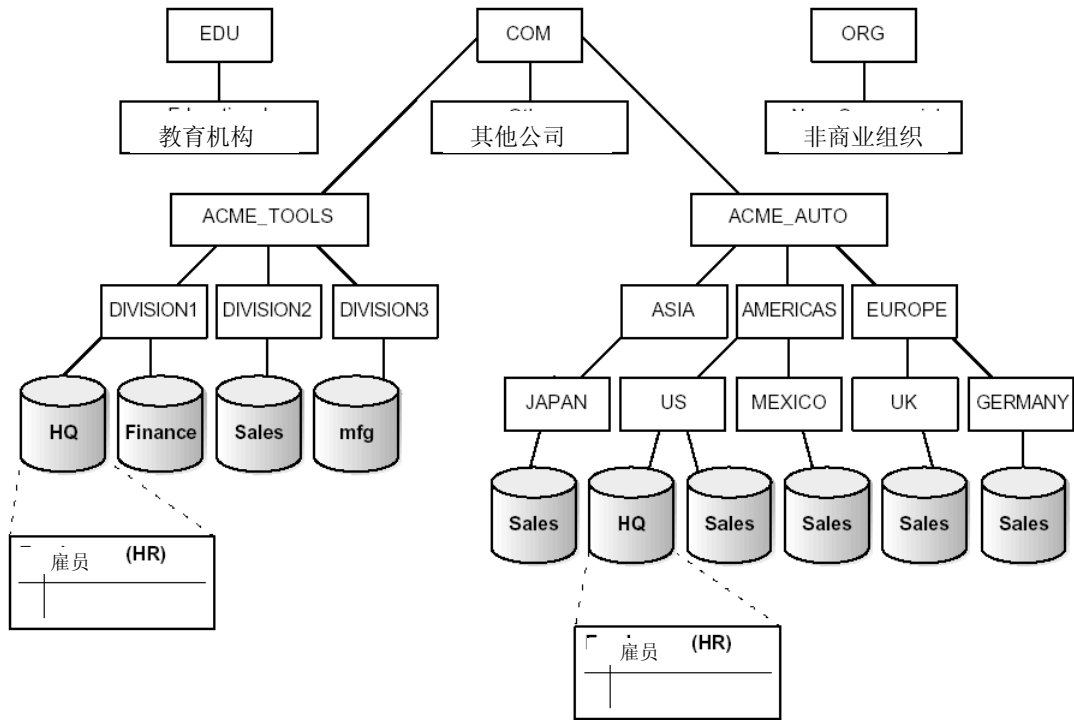


图 28-4 网络数据库的层次排列

尽管几个数据库可以共享一个个体的名称，但每个数据库都必须要有个惟一的全局数据库名。例如，网络域为 `us.americas.acme_auto.com` 和 `uk.europe.acme_auto.com`，每个都包含 `sales` 数据库。但全局数据库命名系统是按照如下方式进行命名的，以便区分 `americas` 部门中的 `sales` 数据库和 `europe` 部门中的 `sales` 数据库：

```
sales.us.americas.acme_auto.com
sales.uk.europe.acme_auto.com
```

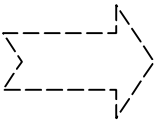
28.2.5 数据库链接的名称

一般来谈的，数据库链接的名称与其所参照的远程数据库的全局数据库名是相同的。例如，如果一个数据库的全局数据库名是 `sales.us.oracle.com`，那么，其数据库链接也被称为 `sales.us.oracle.com`。

当将初始化参数 `GLOBAL_NAMES` 设为 `TRUE` 时，Oracle 就会保证数据库链接的名称与远程数据库的全局数据库名相同。例如，如果 `hq` 的全局数据库名为 `hq.acme.com` 并且 `GLOBAL_NAMES` 为 `TRUE`，那么链接名称一定是 `hq.acme.com`。注意，Oracle 只是检查存储在数据字典中的全局数据库名的域部分，而不是初始化参数文件中的所设置的 `DB_DOMAIN`（参见第 29 章的“改变全局数据库名中的域”）。

如果将初始化参数 `GLOBAL_NAMES` 设为 `FALSE`，那么也就不需要使用全局名称了。你想将数据库链接命名成什么都可以。例如，可以将数据库链接 `hq.acme.com` 命名为 `foo`。

注意：Oracle 公司建议，因为有许多有用的特性（包括复制）需要全局命名方式，所



以，应使用全局名称。

在已经启用全局命名方式之后，对分布式数据库的用户而言，数据库链接本质上是透明的，因为一个数据库链接的名称与该链接所指向的数据库的全局名称是相同的。例如，如下语句在本地数据库中创建了一个到远程数据库 sales 的数据库链接。

```
CREATE PUBLIC DATABASE LINK sales.division3.acme.com USING 'sales1';
```

28.2.6 数据库链接的类型

Oracle 允许用户创建 Private（私有），public（公共）和 global（全局）数据库链接。这些基本的链接类型按照允许哪些用户访问远程数据库而有所不同。

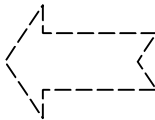
类型	拥有者	说明
Private	创建该链接的用户。 视图的物主身份数据来自： ▪ DBA_DB_LINKS ▪ ALL_DB_LINKS ▪ USER_DB_LINKS	在本地数据库的特殊模式中创建链接。只有私有数据库链接的拥有者或该模式中的 PL/SQL 子程序，可以使用该链接来访问相应的远程数据库中的数据库对象。
Public	被称为 PUBLIC 的用户。视图的物主身份数据来自上述视图。	创建数据库宽松的链接。数据库中的所有用户和 PL/SQL 子程序，都能使用该链接来访问相应的远程数据库中的数据库对象。
Global	被称为 PUBLIC 的用户。视图的物主身份数据来自上述视图。	创建网络宽松的链接。当 Oracle 网络使用 Oracle 名称时，该系统中的名称服务器就会为网络中的每个 Oracle 数据库，自动地创建和管理全局数据库链接。任何数据库中的用户和 PL/SQL 子程序，都能使用全局链接来访问相应的远程数据库中的对象。

究竟在分布式数据库中使用什么类型的数据库链接，依赖于使用该系统的应用的特殊需要。要考虑如下优点和缺点：

- 私有数据库链接 这种链接比公共或全局链接更安全。因为只有私有链接的拥有者，或相同模式中的子程序，才能使用该链接来访问远程数据库。
- 公共数据库链接 当很多用户都需要一个访问远程 Oracle 数据库的路径时，就可以为数据库中的所有用户，创建一个单独的公共数据库链接。
- 全局数据库链接 当一个 Oracle 网络使用 Oracle 名称时，管理员就能方便地，为该系统中的所有数据库，管理全局数据库链接。数据库链接的管理是集中式的和简单的。

28.2.7 数据库链接的用户

当创建数据库链接时，你要确定哪个用户应该连接到远程数据库以访问数据。下表说明了数据库链接所涉及的用户的类型之间的差别：



用户类型	含义	创建链接的语法的例子
已连接用户	访问数据库链接的本地用户（没有指定固定的用户名和口令）。如果 SYSTEM 在一个查询中访问公共链接，那么已连接用户就是 SYSTEM ，而且 Oracle 会连接到远程数据库中的 SYSTEM 模式。 注意：已连接用户并不一定是创建该链接的用户，而是访问该链接的任何用户。	<code>CREATE PUBLIC DATABASE LINK hq USING 'hq';</code>
当前用户	在 CURRENT_USER 数据库链接中的全局用户。该全局用户必须用 X.509 证书（SSL 鉴别的企业用户）或者口令（口令鉴别的企业用户）来鉴别，而且必须是该链接所包含的两个数据库中的用户。当前用户链接是 Oracle 高级安全选项的一个方面。	<code>CREATE PUBLIC DATABASE LINK hq CONNECT TO CURRENT_USER using 'hq';</code>
固定用户	用户名/口令是链接定义的一部分的用户。如果一个链接包含一个固定用户，那么，该固定用户的用户名和口令就会被用于连接到远程数据库。	<code>CREATE PUBLIC DATABASE LINK hq CONNECT TO jane IDENTIFIED BY doe USING 'hq';</code>

28.2.7.1 已连接用户数据库链接

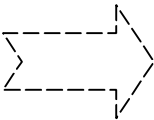
已连接用户链接没有与之相关的连接字符串。已连接用户链接的优点是：参照该链接的用户会按相同用户而连接到远程数据库。而且，由于该链接没有相关的连接字符串，口令也就不会按清晰文本方式存储在数据字典中了。

已连接用户链接也有某些缺点。由于这种链接需要用户具有他试图连接的远程数据库的帐户和权限，对管理员来说，这就需要对他们进行更多的权限管理了。再者，这就给用户赋予了比其所需的还要多的权限了，因而违反了最少权限的基本安全概念：只应该给用户赋予他们工作所需的权限。

使用已连接用户数据库链接的能力取决于几个因素，主要是，该用户是 Oracle 用口令来鉴别的，或是用操作系统或网络鉴别服务从外部鉴别的。如果该用户是外部鉴别的，那么，使用已连接用户链接的能力还取决于，该远程数据库是否认可用户的远程鉴别，这是通过 **REMOTE_OS_AUTHENT** 初始化参数来设置的。

REMOTE_OS_AUTHENT 参数操作如下：

如果 REMOTE_OS_AUTHENT 是 N	那么 N
TRUE 对远程数据库。	外部鉴别的用户可以使用已连接用户数据库链接，来连接到该远程数据库。
FALSE 对远程数据库。	外部鉴别的用户不可以使用已连接用户数据库链接，来连接到远程数据库。除非使用 Oracle 高级安全选项所支持的安全协议或网络鉴别服务。



28.2.7.2 固定用户数据库链接

固定用户链接的好处是：它按照在连接字符串中指定的该用户的安全语义，来将初始数据库中的用户连接到远程数据库。例如，本地用户 `joe` 可以在 `joe` 的模式中创建一个公共数据库链接，该链接指定固定用户 `scott` 及其口令 `tiger`。如果 `jane` 在一个查询中使用了该固定用户链接，那么尽管 `jane` 是本地数据库上的用户，但她是按 `scott/tiger` 连接到远程数据库的。

固定用户数据库链接有一个与连接字符串相关的用户名和口令。用户名和口令以非加密格式被存储在数据字典的 `LINK$` 表中。

警告：将用户名和口令按非加密格式存储在数据字典中的事实，是固定用户数据库链接的潜在的安全隐患。

如果 `O7_DICTIONARY_ACCESSIBILITY` 初始化参数被设置成 `TRUE`，那么具有 `SELECT ANY TABLE` 系统权限的用户就可以访问数据字典了，因此也就危及到了与固定用户相关的鉴别。

`O7_DICTIONARY_ACCESSIBILITY` 初始化参数的默认值是 `FALSE`。

这种安全问题的一个例子是，假设 `jane` 没有权限使用按 `scott/tiger` 连接到 `hq` 数据库的私有链接，但却在 `O7_DICTIONARY_ACCESSIBILITY` 初始化参数是 `TURE` 数据库中拥有 `SELECT ANY TABLE` 的权限。那么，她就可以查询 `LINK$` 并且读出通向 `hq` 的连接字符串是 `scott/tiger`。如果 `jane` 在存储 `hq` 的主机上有一个帐号，那么她就能连接到该主机上，之后用口令 `tiger` 按 `scott` 连接到 `hq`。就算她从本地连接，她也会具有 `scott` 的所有权限，并且任何审计记录都被记录下来，就好像她就是 `scott`。

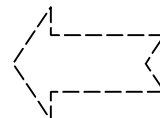
28.2.7.3 当前用户数据库链接

当前用户数据库链接使用全局用户。全局用户必须是用 `X.509` 证书或口令所鉴别的用户，并且是包含在该链接中的两个数据库中的用户。

激活 `CURRENT_USER` 链接的用户不必是全局用户。例如，如果 `jane` 是用应付帐户数据库的口令鉴别（不作为一个全局用户）的，她就可以通过访问存储过程，来检索 `hq` 中的数据。该过程使用当前用户数据库链接，它按全局用户 `scott` 将她连接到 `hq` 数据库。用户 `scott` 是全局用户，并且通过 `SSL` 的证书被鉴别，但 `jane` 不是。

注意，当前用户数据库链接有如下结论：

- 如果在一个存储对象中不访问当前用户数据库链接，那么该当前用户就与访问该链接的已连接的用户相同。例如，如果 `scott` 通过当前用户链接发布 `SELECT` 语句，那么当前用户就是 `scott`。
- 当执行一个存储对象（如访问数据库链接的过程、视图、或触发器）时，当前用户就是拥有该存储对象的用户，而不是调用该对象的用户。例如，如果 `jane` 调用过程 `scott.p`（由 `scott` 创建），并且当前用户链接出现在被调用的过程中，那么 `scott` 就是该链接的当前用户。
- 如果存储对象是一个调用者权力函数、过程或包，那么该调用者的鉴别 ID 就被用于按远程用户进行连接。例如，如果用户 `jane` 调用过程 `scott.p`（由 `scott` 创建的调



用者权力过程), 并且该链接出现在过程 `scott.p` 中, 那么 `jane` 就是当前用户。

- 不能按企业用户连接到数据库, 并且使用在共享、全局模式中的存储过程中的当前用户链接。例如, 如果用户 `jane` 访问数据库 `hq` 中的共享模式 `quest` 中的一个存储过程, 她就不能使用该模式中的当前用户链接来登录到远程数据库。

28.2.8 数据库链接的创建: 例子

数据库链接由 `CREATE DATABASE LINK` 语句来创建。下表给出了在本地数据库中创建通向远程 `sales.us.americas.acme_auto.com` 数据库的数据库链接的 SQL 语句的例子。

SQL 语句	链接到数据库	链接用户	链接类型
<code>CREATE DATABASE LINK sales.us.americas.acme_auto.com USING 'sales_us'</code>	使用网络服务名 <code>sales_us</code> 的 <code>sales</code>	已连接用户	私有的已连接用户
<code>CREATE DATABASE LINK foo CONNECT TO CURRENT_USER USING 'am_sls';</code>	使用网络服务名 <code>am_sls</code> 的 <code>sales</code>	已授权用户	私有当前用户
<code>CREATE DATABASE LINK sales.us.americas.acme_auto.com CONNECT TO scott IDENTIFIED BY tiger USING 'sales_us'</code>	使用网络服务名 <code>sales_us</code> 的 <code>sales</code>	使用口令 <code>tiger</code> 的 <code>scott</code>	私有固定用户
<code>CREATE PUBLIC DATABASE LINK sales CONNECT TO scott IDENTIFIED BY tiger USING 'rev';</code>	使用网络服务名 <code>rev</code> 的 <code>sales</code>	使用口令 <code>tiger</code> 的 <code>scott</code>	公共固定用户
<code>CREATE SHARED PUBLIC DATABASE LINK sales.us.americas.acme_auto.com CONNECT TO scott IDENTIFIED BY tiger AUTHENTICATED BY anupam IDENTIFIED BY bhide USING 'sales';</code>	使用网络服务名 <code>sales</code> 的 <code>sales</code>	使用口令 <code>bhide</code> 按 <code>anupam</code> 鉴别的, 口令为 <code>tiger</code> 的 <code>scott</code>	共享公共固定用户

28.2.9 模式对象和数据库链接

在创建了数据库链接之后, 就可以执行访问远程数据库中的对象的 SQL 语句了。例如, 要想使用数据库链接 `foo` 来访问远程对象 `emp`, 就可以发布如下语句:

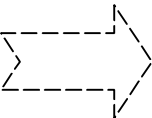
```
SELECT * FROM emp@foo;
```

用数据库链接来构造合理形式的对象名称, 是在分布系统中进行数据管理的一个重要方面。

28.2.9.1 用数据库链接来命名模式对象

Oracle 使用全局数据库名来全局性地命名模式对象, 其方案如下:

```
schema.schema_object@global_database_name
```



其中：

schema 是数据或模式对象的逻辑结构的集合。模式被数据库用户所拥有，并且那个用户的名称相同。每个用户拥有一个单独的模式。

schema_object 是像表、索引、视图、同义词、过程、包或数据库链接那样的逻辑数据结构。

global_database_name 是惟一标识远程数据库的名称。该名称必须与远程数据库的初始化参数 **DB_NAME** 和 **DB_DOMAIN** 连接在一起的字符串相同，除非将参数 **GLOBAL_NAMES** 设置成 **FALSE**，此时，任何名称都可以。

例如，使用到数据库 **sales.division3.acme.com** 的数据库链接，用户或应用就能像下面那样参照远程数据：

```
SELECT * FROM scott.emp@sales.division3.acme.com; # emp table in scott担
schema
```

```
SELECT loc FROM scott.dept@sales.division3.acme.com;
```

如果将 **GLOBAL_NAMES** 设置成 **FALSE**，那么能使用任何名称来作为到 **sales.division3.acme.com** 的链接。例如，能将链接称作 **foo**。那么，就可以如下所示访问远程数据库：

```
SELECT name FROM scott.emp@foo; # link name different from global name
```

28.2.9.2 模式对象的同义词

Oracle 允许用户创建同义词，以便对用户隐藏数据库链接的名称。同义词允许使用与访问本地数据库中的表所用的相同的语法来访问远程数据库中的表。例如，假设对远程数据库中的表，发布下面的查询：

```
SELECT * FROM emp@hq.acme.com;
```

可以为 **emp@hq.acme.com** 创建同义词 **emp**，以便发布下面的查询来代替访问同样的数据：

```
SELECT * FROM emp;
```

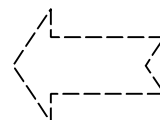
28.2.9.3 模式对象的名称解析

为了解析应用对模式对象的参照（被称为名称解析的过程），Oracle 会按层次结构形成对象的名称。例如，Oracle 保证数据库中的每个模式都有惟一的名称，并且模式中的每个对象也有惟一的名称。其结果是，模式对象的名称在数据库中总是惟一的。进一步，Oracle 解析应用对一个对象的本地名称的参照。

在分布式数据库中，系统中的所有应用都可访问模式对象，比如表。Oracle 用全局数据库名来扩展层次化的命名模式，以便有效地创建全局对象名，并在分布式数据库系统中解析对模式对象的参照。例如，一个查询通过指定其完整的被限定的名称（包括其所属的数据库）就能参照远程表。

例如，假设用用户 **SYSTEM** 连接到本地数据库：

```
CONNECT SYSTEM/password@sales1
```



然后，使用数据库链接 `hq.acme.com` 发布下面的语句，来访问远程数据库 `hq` 中的 `scott` 和 `jane` 模式中的对象：

```
SELECT * FROM scott.emp@hq.acme.com;
INSERT INTO jane.accounts@hq.acme.com (acc_no, acc_name, balance)
VALUES (5001, 'BOWER', 2000);
UPDATE jane.accounts@hq.acme.com
SET balance = balance + 500;
DELETE FROM jane.accounts@hq.acme.com
WHERE acc_name = 'BOWER';
```

28.2.10 数据库链接的限制

使用数据库链接时，不能执行下面的操作：

- 授予远程对象上的权限
- 在某些远程对象上执行 `DESCRIBE` 操作。但是，下面的远程对象是支持 `DESCRIBE` 操作的：
 - 表
 - 视图
 - 过程
 - 函数
- `ANALYZE` 远程对象
- 定义或者实施参照的完整性
- 给远程数据库中的用户授予角色
- 获得远程数据库上的非默认的角色。例如，如果 `jane` 连接到本地数据库，并且执行以 `scott` 名义连接的使用、固定用户链接的存储过程，`jane` 就会采用 `scott` 在远程数据库上的默认角色。但 `Jane` 不能靠发布 `SET ROLE` 语句来获得非默认角色。
- 执行使用共享服务器连接的散列查询联结。
- 使用当前用户的不用通过 `SSL`、口令或者 `NT` 本身的鉴别进行鉴别的链接。

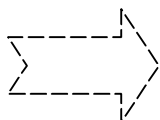
28.3 分布式数据库的管理

下面几节说明有关在 `Oracle` 分布式数据库系统中的数据库管理的某些内容：

- 站点自治
- 分布式数据库的安全
- 审计数据库链接
- 管理工具

28.3.1 站点自治

站点自治是指在分布式数据库中，独立地管理所参与的每个服务器，而与所有其他数据库无关。尽管几个数据库能同时工作，但每个数据库都有一个单独的、被分别管理的数



据的仓库。在 Oracle 分布式数据库中，站点自治的好处是：

- 系统的节点能反映出需要保持独立的公司或组的逻辑结构。
- 本地管理员控制相应的本地数据。所以，每个数据库管理员的责任域就更小并且更容易管理了。
- 不相关的失败就不太可能破坏分布式数据库中的其他节点。单个的数据库失败就不必中断所有分布式操作或是一个性能瓶颈。
- 管理员能从系统的其他节点上独立地恢复孤立的系统失败。
- 每个本地数据库都有数据字典 —— 访问本地数据不需要全局分类。
- 各节点能够各自升级软件。

尽管 Oracle 允许在分布式数据库系统中各自管理每个数据库，但还是不要忽视系统的整体需要。例如，可能需要：

- 在每个数据库中创建附加用户帐户，以支持创建服务器到服务器连接的链接。
- 设置附加的初始化参数如 `DISTRIBUTED_TRANSACTIONS` 和 `COMMIT_POINT_STRENGTH`。

28.3.2 分布式数据库的安全

对于分布式数据库系统，Oracle 支持在非分布式数据库环境中可使用的所有安全特性，包括：

- 用户和角色的口令鉴别
- 用户和角色的某种外部鉴别，包括
 - 用于已连接用户链接的 Kerberos 版本 5.0
 - 用于已连接用户链接的 DCE
 - 用于客户到服务器和服务器到服务器的连接的注册信息包加密。

下面几节说明了，在配置 Oracle 分布式数据库系统时要考虑的几个附加内容：

- 通过数据库链接的鉴别
- 不用口令的鉴别
- 支持的用户帐户和角色
- 集中式用户和权限的管理
- 数据加密

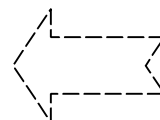
28.3.2.1 通过数据库链接的鉴别

数据库链接要么是私有的要么是公共的，或是被鉴别的或是非鉴别的。通过在创建链接的语句中指定 `PUBLIC` 关键字来创建公共链接。例如，能发布如下语句：

```
CREATE PUBLIC DATABASE LINK foo USING 'sales';
```

可以通过在创建数据库链接的语句中，指定 `CONNECT TO` 子句，`AUTHENTICATED BY` 子句、或同时使用两个子句，来创建可鉴别的链接。例如，能发布如下语句：

```
CREATE DATABASE LINK sales CONNECT TO scott IDENTIFIED BY tiger USING 'sales';
CREATE SHARED PUBLIC DATABASE LINK sales CONNECT TO mick IDENTIFIED BY jagger
```



AUTHENTICATED BY david IDENTIFIED BY bowie USING 'sales';

下表说明了用户如何通过链接访问远程数据库:

链接类型	是否被鉴别	安全访问
私有	不是	当连接到远程数据库时, Oracle 使用从本地会话获得的安全信息 (用户标识/口令)。因此, 该链接是一个已连接用户数据库链接。口令必须在两个数据库之间同步。
私有	是	用户标识/口令是取自链接的定义而不是本地会话文本。因此, 该链接是一个固定用户数据库链接。 这种配置允许两个数据库上的口令不同, 但是本地数据库链接的口令必须与远程数据库的口令匹配。口令是以清晰文本的方式存储在本地系统目录中的, 这增加了安全风险。
公共	不是	除了所有用户能参照这个指向远程数据库的指针以外, 其它操作与私有的非鉴别的链接一样。
公共	是	本地数据库中的所有用户都能访问远程数据库, 并且都使用同样的用户标识/口令来实现连接。同样, 口令也是以清晰文本方式存储在本地目录中的。所以, 如果在本地数据库中拥有足够的权限, 就能看见口令。

28.3.2.2 不用口令的鉴别

当使用已连接用户或当前用户数据库链接时, 就能使用外部鉴别资源如 Kerberos, 来获得端到端的安全。在端到端鉴别中, 信任是通过从服务器传到服务器的, 并可以通过属于同一个域的数据库服务器来鉴别。例如, 如果 jane 在本地数据库上被外部地鉴别, 并且 jane 想使用已连接用户链接, 用自己来连接到远程数据库, 那么本地服务器就将该安全凭据传给远程数据库。

28.3.2.3 支持的用户帐户和角色

在分布式数据库系统中, 必须仔细设计为支持使用本系统的应用所必须的用户帐户和角色。注意以下几点:

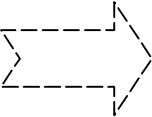
- 为了建立服务器到服务器连接所必须的用户帐户, 分布式数据库系统中的所有数据库中必须是可用的。
- 使分布式数据库应用的用户可以利用应用的权限所必须的角色, 必须出现在分布式数据库系统中的所有数据库中。

当创建针对分布式数据库系统的节点的数据库链接时, 要确定每个站点需要哪个用户帐户和角色, 以支持使用该链接的服务器到服务器的连接。

在分布式环境中, 一般来说, 用户需要访问很多网络服务。当必须为每个用户配置单独的鉴别以便访问每个网络服务时, 安全管理就变得难以控制, 尤其是在大型系统中。

28.3.2.4 集中式用户和权限的管理

Oracle 提供各种方法来管理分布式系统中所包含的用户和权限。例如, 可做如下选择:



- 企业用户管理。可以创建通过 SSL 或使用口令来鉴别的全局用户，然后通过一个单独的企业目录服务而在一个目录中来管理这些用户和他们的权限。
- 网络鉴别服务。这种常用的技术简化了分布式环境中的安全管理。可以使用 Oracle 高级安全选项来增强 Oracle Net 和 Oracle 分布式数据库系统的安全。Windows NT 自身鉴别就是一个非 Oracle 鉴别解决方案的例子。

依赖于模式的全局用户 用于集中管理用户和权限的一个方案是创建下面的用户：

- 一个集中式目录中的全局用户
- 该全局用户必须连接到的每个数据库中的用户

例如，用下面的 SQL 语句，可以创建一个被称为 fred 的全局用户：

```
CREATE USER fred IDENTIFIED GLOBALLY AS 'CN=fred adams,O=Oracle,C=England';
```

该解决方案允许通过集中式目录来鉴别单个全局用户：

依赖于全局用户的解决方案的结论是，必须在该用户必需访问的每个数据库上都创建一个被称为 fred 的用户。因为大多数用户需要许可访问一个应用模式，而又不需要他们拥有该模式，所以，在每个数据库中为每个全局用户建立单独的帐户造成了很大的系统开销。因为这个问题，Oracle 也支持不依赖于模式的用户，该用户是可以访问每个数据库中的单独的、普通的模式的全局用户。

不依赖于模式的全局用户 Oracle 支持这样的功能，即允许通过企业目录服务来集中地管理全局用户。在该目录中被管理的用户，被称为企业用户。该目录所包含的信息如下：

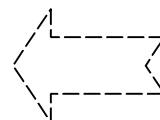
- 企业用户能访问分布式系统中的哪些数据库
- 企业用户能使用每个数据库中的哪些角色
- 企业用户能连接到每个数据库中的哪些模式

每个数据库的管理员不需要为需要连接的每个数据库上的每个企业用户都创建一个全局用户帐户。相反是，多个企业用户能够连接到同一个被称为共享模式的数据库模式。

注意：不能访问共享模式中的当前用户数据库链接。

例如，假设 jane, bill 和 scott 都使用人力资源应用。hq 应用的所有对象都被包含在 hq 数据库的 quest 模式中。在这种情况下，就可以创建一个本地全局用户帐户做为一个共享模式。该全局用户名，也就是一个共享模式名称，为 guest。Jane, bill 和 scott 都是被创建在目录服务中的企业用户。他们也被映射成目录中的 quest 模式，而且可以在 hq 的应用中为其被分配不同的身份。

图 28-5 说明使用企业目录服务的全局用户的安全的例子：



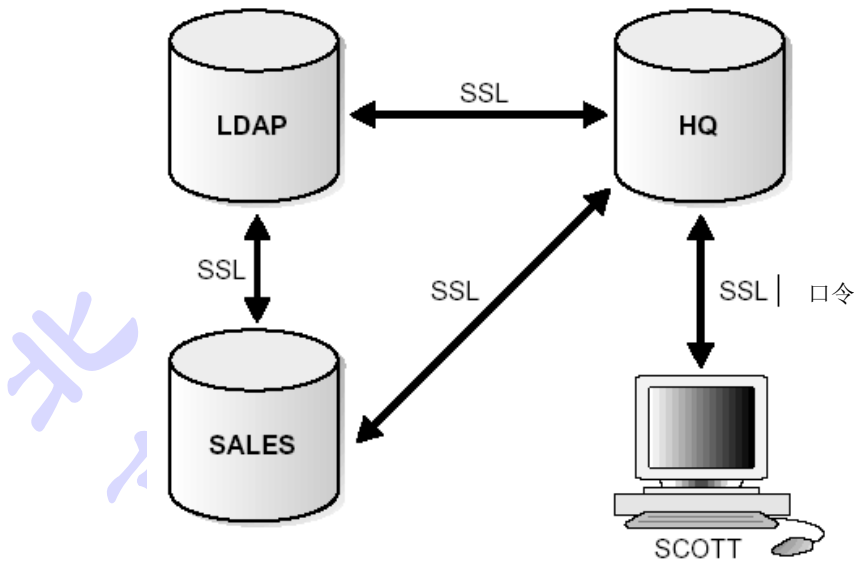


图 28-5 全局用户的安全

假设企业目录服务包含如下关于 hq 和 sales 数据库的企业用户的信息：

数据库	角色	模式	企业用户
Hq	clerk1	Quest	bill
			scott
Sales	clerk2	Quest	jane
			scott

同样，假设 hq 和 sales 数据库的本地管理员发布了如下语句：

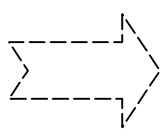
数据库	CREATE 语句
hq	CREATE USER guest IDENTIFIED GLOBALLY AS '';
	CREATE ROLE clerk1 GRANT select ON emp;
	CREATE PUBLIC DATABASE LINK sales_link CONNECT AS CURRENT_USER USING 'sales'
sales	CREATE USER guest IDENTIFIED GLOBALLY AS '';
	CREATE ROLE clerk2 GRANT select ON dept;

假设企业用户 scott 需要连接到本地数据库 hq，以便执行包括 sales 数据库的分布式事务。发生如下步骤（不一定严格按照此顺序）：

- ①使用 SSL 或口令鉴别企业用户 scott。
- ②用户 scott 发布下面语句：

```
SELECT e.ename, d.loc
FROM emp e, dept@sales_link d
WHERE e.deptno=d.deptno
```

- ③数据库 hq 和 sales 使用 SSL 彼此相互鉴别。



④数据库 hq 查询企业目录服务，以确定企业用户 scott 是否已经访问 hq 数据库，并且发现 scott 可以使用角色 clerk1 来访问本地模式 quest。

⑤数据库 sales 查询企业目录服务，以确定企业用户 scott 是否已经访问 sales，并且发现 scott 可以使用角色 clerk2 来访问本地模式 quest。

⑥企业用户 scott 用角色 clerk2 登录到 sales 数据库的模式 quest 中，并且发布 SELECT 语句以获得所需信息，并把该信息传递给 hq 数据库。

⑦数据库 hq 从 sales 中接收所需的数据，并把数据返回给客户 scott。

28.3.2.5 数据加密

Oracle 高级安全选项还使 Oracle Net 和相关产品能使用网络数据加密与校验求和技术，使得数据不能读出或改变。通过使用 RAS 数据安全 RC4 (RAS Data Security RC4) 或数据加密标准 (Data Encryption Standard, DES) 加密算法来保护数据免遭非授权查看。

为了确保数据在传输期间不被修改、删除或重放，Oracle 高级安全选项的安全服务，能产生一个加密安全消息摘要，而且在网络中发送的每个包都包含该摘要。

28.3.3 审计数据库链接

必须在本地经常执行局部审计操作。也就是说，如果一个用户操作本地数据库，并通过数据库链接访问远程数据库，则在本地数据库中审计本地动作，在远程数据库中审计远程动作——在各自的数据库中设置了合适的审计选项。

远程数据库不能确定一个成功的连接需求和后来的 SQL 语句是来自于其他的服务器，或者是来自于本地已连接的客户。例如，假设：

- 固定用户链接 hq.acme.com 将本地用户 jane 按远程用户 scott 连接到远程 hq 数据库。
- 在远程数据库中审计用户 scott。

审计远程数据库会话期间执行的动作，就好像将 scott 连接到本地 hq，并且在那儿执行同样的动作一样。为了捕获嵌入在该链接中的该用户名——此时是 hp 数据库的 scott——的操作，就必须在远程数据库中设置审计选项，如果所希望的结果是审计 jane 正在远程数据库中做什么时。

注意：可以审计全局用户的全局用户名。

不能在远程对象上设置本地审计选项。因此，尽管可以在远程数据库上审计对远程对象的访问，但不能审计对数据库链接的使用。

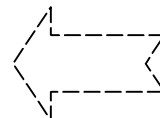
28.3.4 管理工具

当管理 Oracle 分布式数据库系统时，数据库管理者可选择使用几种工具：

- 企业管理器
- 第三方管理工具
- SNMP 支持

28.3.4.1 企业管理器

企业管理器是 Oracle 的数据库管理工具，该工具提供了图形用户界面 (GUI)。企业管



理器通过一个易于使用的界面，为分布式数据库提供管理功能。能使用企业管理器做如下事情：

- 管理多个数据库。可以用企业管理器来管理单个数据库，或同时管理多个数据库。
- 集中数据库管理任务。可以管理运行在全世界任何位置、任何 Oracle 平台上的本地或者远程数据库。另外，通过 Oracle Net 所支持的各种网络协议，能将这些 Oracle 平台互相连接。
- 动态地执行 SQL、PL/SQL 和企业管理器命令。可以用企业管理器输入、编辑和执行语句。企业管理器还能保持已执行过的语句的历史记录。

所以，不用重新键入原来的语句，就能重新执行这些语句，如果想在分布式数据库系统中，重复执行的很长的语句，这种特性就特别有用了。

- 管理安全特性。如全局用户、全局角色和企业目录服务。

28.3.4.2 第三方管理工具

目前，超过 60 家公司在生产超过 150 多种的能帮助我们管理 Oracle 数据库和网络的产品，提供了一个真正的开放环境。

28.3.4.3 SNMP 支持

除了网络管理能力之外，Oracle 对简单网络管理协议（Simple Network Management Protocol, SNMP）的支持，允许任何基于 SNMP 的网络管理系统查找和查询一个 Oracle 数据库服务器。SNMP 是被很多流行的网络管理系统所承认的标准，如：

- HP 的 OpenView
- Digital 的在 NetView 上的 POLYCENTER 管理器
- IBM 的 NetView/6000
- Novell 的 NetWare 管理系统
- SunSoft 的 SunNet 管理器

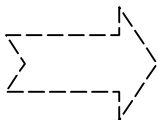
28.4 分布式系统中的事务处理

事务是一个由单个用户所执行的一个或多个 SQL 语句构成的逻辑工作单元。事务开始于用户的第一个可执行的 SQL 语句，结束于该用户提交或回滚该事务。

一个远程事务仅包含访问单个远程节点的语句。一个分布式事务包含访问多个节点的语句。

下面几节定义了事务处理中的重要概念，并说明了事务是如何在分布式数据库中访问数据的：

- 远程 SQL 语句
- 分布式 SQL 语句
- 用于远程和分布式语句的共享 SQL
- 远程事务
- 分布式事务



- 两阶段提交机制
- 数据库链接的名称解析
- 模式对象的名称解析

28.4.1 远程 SQL 语句

一条远程查询语句是从一个或多个远程表中选取信息的查询，这些所有的表属于相同的远程节点。例如，下面的查询访问远程 sales 数据库的 scott 模式中的 dept 表中的数据：

```
SELECT * FROM scott.dept@sales.us.americas.acme_auto.com;
```

一条远程更新语句是一个这样的更新，即修改位于相同的远程节点中的一个或多个表中的数据。例如，下面的查询更新远程 sales 数据库的 scott 模式中的 dept 表中的数据：

```
UPDATE scott.dept@mktng.us.americas.acme_auto.com
SET loc = 'NEW YORK'
WHERE deptno = 10;
```

注意：一条远程更新语句可以包括一条从一个或多个远程节点中检索数据的子查询，但是由于更新仅能发生在单个远程节点上，所以把该语句归类为远程更新。

28.4.2 分布式 SQL 语句

分布式查询语句从两个或多个节点上检索信息。例如：下面的查询既访问本地数据库也访问远程数据库中的数据。

```
SELECT ename, dname
FROM scott.emp e, scott.dept@sales.us.americas.acme_auto.com d
WHERE e.deptno = d.deptno;
```

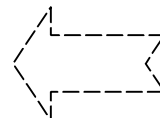
分布式更新语句修改两个或多个节点上的数据。使用 PL/SQL 子程序单元，如包含两个或多个访问不同节点上的数据的过程或触发器的远程更新，就可能实现分布式更新。例如：下面的 PL/SQL 程序单元更新本地数据库和远程 sales 数据库中的表：

```
BEGIN
UPDATE scott.dept@sales.us.americas.acme_auto.com
SET loc = 'NEW YORK'
WHERE deptno = 10;
UPDATE scott.emp
SET deptno = 11
WHERE deptno = 10;
END;
COMMIT;
```

Oracle 将程序中的语句发送给远程节点，并且它们按一个程序单元执行成功或失败。

28.4.3 用于远程和分布式语句的共享 SQL

使用共享 SQL 的远程或分布式语句的机制与本地语句的机制是相同的。SQL 文本必须匹配，而且所参照的对象也必须匹配。如果可用的话，共享 SQL 能用于任何语句或已分解



查询的本地和远程管理

28.4.4 远程事务

远程事务包含一个或多个远程语句，所有这些语句参照一个单独的远程节点。例如，下面的事务包含两条语句，每一条都访问远程 sales 数据库：

```
UPDATE scott.dept@sales.us.americas.acme_auto.com
SET loc = 'NEW YORK'
WHERE deptno = 10;
UPDATE scott.emp@sales.us.americas.acme_auto.com
SET deptno = 11
WHERE deptno = 10;
COMMIT;
```

28.4.5 分布式事务

分布式事务是包含一条或多条语句的事务，这些语句单独地或作为一个语句组，更新分布式数据库中两个或多个不同的节点上的数据。例如，下面的事务更新本地数据库和远程 sales 数据库：

```
UPDATE scott.dept@sales.us.americas.acme_auto.com
SET loc = 'NEW YORK'
WHERE deptno = 10;
UPDATE scott.emp
SET deptno = 11
WHERE deptno = 10;
COMMIT;
```

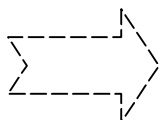
注意：如果事务中的所有语句仅参照一个单独的远程节点，则该事务是远程事务，而不是分布式事务。

28.4.6 两阶段提交机制

数据库必须确保事务中的所有语句（分布式的或非分布式的），作为一个单元或者提交或者回滚。正在进行中的事务的作用对所有节点上的所有其他事务应该是不可见的；这种透明性对所有事务（包括任何类型操作，如查询、更新或远程过程调用）来说的，都应该成立。

在分布式数据库中，Oracle 必须在整个网络中用相同的特征来协调事务控制并管理数据的一致性，甚至在网络或系统出现故障时。

Oracle 的两阶段提交机制确保了参与分布式事务的所有数据库服务器，或者全部提交或者全部回滚事物中的语句。两阶段提交机制也保护了由完整性约束、远程过程调用和触发器所执行的隐含的 DML 操作。



28.4.7 数据库链接的名称解析

全局对象名是一个用数据库链接来指定的对象。全局对象名的基本组成部分如下：

- 对象名
- 数据库名
- 域

下面的表显示了一个明确指定的全局数据库对象名的组成部分：

语句	对象	数据库	域
SELECT * FROM joan.dept@sales.acme.com	Dept	sales	acme.com
SELECT * FROM emp@mktg.us.acme.com	Emp	mktg	us.acme.com

无论何时，如果一条 SQL 语句包含了对全局对象名的参照，Oracle 就用与在全局对象名中指定的数据库名称的一个名称相匹配来搜索数据库链接。例如，如果发布下面的语句：

```
SELECT * FROM scott.emp@orders.us.acme.com;
```

Oracle 就搜索被称为 orders.us.acme.com 的数据库链接。Oracle 执行这个操作，以便确定指向该被指定的远程数据库的路径。

Oracle 总是按照下面的顺序搜索匹配的数据库链接：

- ①发布 SQL 语句的用户模式中的私有数据库链接。
- ②本地数据库中的公共数据库链接。
- ③全局数据库链接（仅当可以使用 Oracle 名称服务器时）。

28.4.7.1 全局数据库名是完整时的名称解析

假设发布下面的 SQL 语句，它指定了一个完整的全局数据库名：

```
SELECT * FROM emp@prod1.us.oracle.com
```

此时，指定了数据库名称（prod1）和域（us.oracle.com）两部分，所以 Oracle 搜索私有的、公共和全局数据库链接。Oracle 仅搜索匹配指定的全局数据库名的链接。

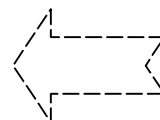
28.4.7.2 全局数据库名是部分时的名称解析

如果指定了域的某一部分，Oracle 就假设指定了一个完整的全局数据库名。如果一条 SQL 语句指定了部分全局数据库名（即，只是指定了数据库部分），Oracle 就把 DB_DOMAIN 初始化参数中的值附加到 DB_NAME 初始化参数的值，以构成一个完整的名称。例如，假设发布下面的语句：

```
CONNECT scott/tiger@locdb
```

```
SELECT * FROM scott.emp@orders;
```

如果 locdb 的网络域是 us.acme.com，那么 Oracle 把此域附加到 orders，以构成 orders.us.acme.com 的完整全局数据库名。Oracle 搜索匹配构成的全局名称的数据库链接。如果没有找到一个匹配的链接，Oracle 返回一个错误，并且不能执行该 SQL 语句。



28.4.7.3 没有指定全局数据库名时的名称解析

如果全局对象名称参照一个本地数据库中的对象，并且数据库链接名称没有用@符号指定，那么 Oracle 自动认为该对象是本地的，而且不搜索或者不使用数据库链接来解析该对象参照。例如，假设发布如下语句：

```
CONNECT scott/tiger@locdb
SELECT * from scott.emp;
```

因为第二个语句没有用数据库链接的连接字符串来指定一个全局数据库名，所以 Oracle 就不搜索数据库链接。

28.4.7.4 名称解析搜索的结束

当 Oracle 找到了第一个匹配时，并不停止搜索匹配的数据库链接。Oracle 必须搜索匹配的私有、公共和网络数据库链接，直到确定了到远程数据库的完整路径（远程帐户和服务名两者）为止。

正如下表所示，第一个匹配确定了远程模式：

如果Λ	那么 Oracle Λ	举例 Λ
不指定 CONNECT 子句	使用一个已连接用户数据库链接	CREATE DATABASE LINK k1 USING 'prod'
不指定 CONNECT TO Λ IDENTIFIED BY 子句	使用一个固定用户数据库链接	CREATE DATABASE LINK k2 CONNECT TO scott IDENTIFIED BY tiger USING 'prod'
指定 CONNECT TO CURRENT_USER 子句	使用一个当前用户数据库链接	CREATE DATABASE LINK k3 CONNECT TO CURRENT_USER USING 'prod'
不指定 USING 子句	搜索直到找到一个指定数据库字符串的链接。如果没有找到匹配的数据库链接，并且字符串从来也没有被识别，Oracle 就返回一个错误。	CREATE DATABASE LINK k4 CONNECT TO CURRENT_USER

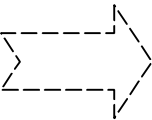
在 Oracle 确定了一个完整的路径之后，就创建一个远程会话——假设代表相同本地会话的同样的连接还没有打开。如果会话已经存在，Oracle 就重新使用该会话。

28.4.8 模式对象的名称解析

当本地 Oracle 数据库按照发布 SQL 语句的本地用户的要求链接到指定的远程数据库之后，对象解析继续执行，就好象远程用户发布了的相关 SQL 语句一样。根据下面的原则，第一个匹配确定了远程模式：

如果使用 Λ	那么对象解析处理就Λ
固定用户数据库链接	在链接创建语句中指定的模式中
已连接用户数据库链接	在已连接用户的远程模式中
当前用户数据库链接	在当前用户的模式中

如果 Oracle 不能找到该对象，那么它就检查远程数据库中的公共对象。如果它不能解



析该对象，那么就保留已建立的远程会话，但不能执行 SQL 语句并返回一个错误。

下面是在分布式数据库系统中的全局对象名称解析的一些例子。对于所有的例子，假设：

28.4.8.1 全局对象的名称解析的例子：完整的对象名称

这个例子说明了 Oracle 怎样解析一个完整的全局对象名，并且使用私有和公共数据库链接两者来决定通向远程数据库的合适路径。对于此例，假设：

- 远程数据库的名称为 sales.division3.acme.com。
- 本地数据库的名称为 hq.division3.acme.com。
- 不能使用 Oracle 名称服务器（因此，也不能使用全局数据库链接）。
- 在模式 tsmith 中包含一个远程表 emp。

考虑本地数据库中的 scott 发布如下语句：

```
CONNECT scott/tiger@hq
CREATE PUBLIC DATABASE LINK sales.division3.acme.com
CONNECT TO guest IDENTIFIED BY network
USING 'dbstring';
```

后来，JWARD 连接并且发布如下语句：

```
CONNECT jward/bronco@hq
CREATE DATABASE LINK sales.division3.acme.com
CONNECT TO tsmith IDENTIFIED BY radio;
UPDATE tsmith.emp@sales.division3.acme.com
SET deptno = 40
WHERE deptno = 10;
```

Oracle 按照如下步骤处理最后的语句：

①Oracle 确认在 jward 的更新语句中参照了一个完整的全局对象名称。因此，系统在本地数据库中搜索具有匹配名称的数据库链接。

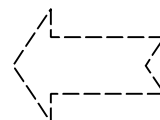
②Oracle 在模式 jward 中找到一个匹配的私有数据库链接。然而，私有数据库链接 jward.sales.division3.acme.com，并不能代表通向远程 sales 数据库的完整路径，仅是一个远程帐户。因此，现在 Oracle 就搜索匹配的公共数据库链接。

③Oracle 在 scott 的模式中，找到了公共数据库链接。从这个公共数据库链接中，Oracle 获得了服务名 dbstring。

④从匹配的私有固定用户数据库链接中获得的远程帐户后，Oracle 确认了一个完整的路径，并按用户 tsmith/radio 建立到远程 sales 数据库的连接。

⑤远程数据库现在能解析针对 emp 表的参照。Oracle 在 tsmith 模式中搜索，并且找到被参照的 emp 表。

⑥远程数据库完成语句的执行，并将结果返回给本地数据库。



28.4.8.2 全局对象的名称解析的例子：部分的对象名称

这个例子说明了 Oracle 怎样解析一个部分的全局对象名，并且使用私有和公共数据库链接两者来决定通向远程数据库的合适路径。

对于此例，假设：

- 远程数据库的名称为 sales.division3.acme.com。
- 本地数据库的名称为 hq.division3.acme.com。
- 不能使用 Oracle 名称服务器（因此，也不能使用全局数据库链接）。
- 远程数据库 sales 中的表 emp 被包含在模式 tsmith 中，但该表不在模式 scott 中。
- 名称为 emp 的公共同义词属于远程数据库 sales，并且指向远程数据库 sales 中的 tsmith.emp。
- 在“全局对象名解析的例子：完整的对象名称”部分的公共数据库链接中，已经在本地数据库 hq 上创建了。

```
CREATE PUBLIC DATABASE LINK sales.division3.acme.com
CONNECT TO guest IDENTIFIED BY network
USING 'dbstring';
```

考虑在本地数据库 hq 上发布如下语句：

```
CONNECT scott/tiger@hq
CREATE DATABASE LINK sales.division3.acme.com;
DELETE FROM emp@sales
WHERE empno = 4299;
```

Oracle 按照如下步骤处理最后的 DELETE 语句：

①Oracle 认为在 scott 的 DELETE 语句中参照了部分的全局对象名。它按照如下方式，用本地数据库域，将部分的全局对象名扩展成完整的全局对象名。

```
DELETE FROM emp@sales.division3.acme.com
WHERE empno = 4299;
```

②Oracle 本地数据库中搜索匹配名称的数据库链接。

③Oracle 在模式 scott 中找到了一个匹配的私有已连接用户链接，但该私有数据库链接并不表示路径。Oracle 将已连接用户名/口令作为路径的远程帐户的一部分，然后搜索并且找到一个匹配的公共数据库链接。

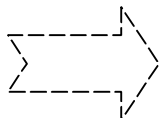
```
CREATE PUBLIC DATABASE LINK sales.division3.acme.com
CONNECT TO guest IDENTIFIED BY network
USING 'dbstring';
```

④Oracle 从公共数据库链接中获得数据库网络服务名 dbstring。此时，Oracle 就已经确定了一个完整的路径。

⑤Oracle 按 scott/tiger 连接到远程数据库，在模式 scott 中搜索，但并没有找到名称为 emp 的对象。

⑥远程数据库搜索名称为 emp 的公共同义词，并且找到了它。

⑦远程数据库执行该语句，并且将结果返回给本地数据库。



28.4.9 在视图、同义词、过程中的全局名称解析

视图、同义词或 PL/SQL 程序单元（例如，过程、函数或触发器），能通过它们的全局对象名来参照远程模式对象。如果全局对象名是完整的，那么 Oracle 就存储对象的定义，而不用扩展全局对象名。如果全局对象名是部分的，那么 Oracle 就使用本地数据库名的域来扩展该名称。

下表说明了 Oracle 是如何为视图、同义词和程序单元扩展全局对象名称的：

如果 Δ	那么 Oracle 就 Δ
创建视图	不扩展部分全局名称——数据字典中存储着定义查询的精确文本。相反，Oracle 每当对使用视图的语句做语法分析时就扩展部分全局对象名。
创建同义词	扩展部分全局名称。存储在数据字典中的同义词的定义包含了被扩展的全局对象名。
编译程序单元	扩展部分全局名称。

28.4.9.1 当全局名称改变时会发生什么

全局名称的改变，可能影响使用部分全局对象名来参照远程数据的视图、同义词和过程。如果被参照的数据库的全局名称改变了，视图和过程就可能尝试参照一个不存在或不正确的数据库。然而，因为同义词在运行时不扩展数据库链接名称，所以它们不改变。

28.4.9.2 针对全局名称改变的方案

例如，考虑两个数据库 sales.uk.acme.com 和 hq.uk.acme.com。还假设 sales 数据库包含如下视图和同义词：

```
CREATE VIEW employee_names AS
SELECT ename FROM scott.emp@hr;
CREATE SYNONYM employee FOR scott.emp@hr;
Oracle 扩展 employee 同义词的定义并将其存储成：
scott.emp@hr.uk.acme.com
```

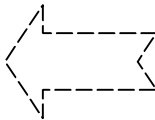
方案 1：两个数据库改变名称 首先，考虑这种情况，销售部门和人力资源部门重新部署到美国。因此，也就改变了两个全局数据库名，如下所示：

旧的全局名称	新的全局名称
sales.uk.acme.com	sales.us.acme.com
hq.uk.acme.com	hq.us.acme.com

下表说明了全局名称改变前后的查询的扩展：

查询 sales	改变之前的扩展	改变之后的扩展
SELECT * FROM Employee_names	SELECT * FROM scott.emp@hr.uk.acme.com	SELECT * FROM scott.emp@hr.us.acme.com
SELECT * FROM employee	SELECT * FROM scott.emp@hr.uk.acme.com	SELECT * FROM scott.emp@hr.us.acme.com

方案 2：一个数据库改变名称 现在考虑仅仅是将销售部门搬到了美国，人力资源部门仍留在英国。因此，也就改变了全局数据库名，如下所示：



旧的全局名称	新的全局名称	
sales.uk.acme.com	sales.us.acme.com	
hq.uk.acme.com	没有改变	
下面的表说明了全局名称改变之前和改变之后的查询扩展：		
查询 sales	改变之前的扩展	改变之后的扩展
SELECT * FROM	SELECT * FROM	SELECT * FROM
employee_names	scott.emp@hr.uk.acme.com	scott.emp@hr.us.acme.com
SELECT * FROM employee	SELECT * FROM	SELECT * FROM
	scott.emp@hr.uk.acme.com	scott.emp@hr.us.acme.com

此时，`employee_name` 视图定义的查询被扩展成一个不存在的全局数据库名。然而，`employee` 同义词继续参照正确的数据库 `hq.uk.acme.com`。

28.5 分布式数据库的应用开发

分布式系统中的应用开发的问题是不能应用到非分布式系统中。本节包括如下与分布式应用开发相关的内容：

- 分布式数据库系统中的透明性
- 远程过程调用（RPCs）
- 分布式查询的优化

28.5.1 分布式数据库系统中的透明性

稍做努力，就能开发出使 Oracle 分布式数据库系统对使用该系统的用户透明的应用。透明的目的，就是使分布式数据库系统看起来就好像它是一个单独的 Oracle 数据库一样。其结果是，系统并不使该系统的开发者和用户承担可能使分布式数据库应用开发更具挑战性和减损用户生产力的复杂性。

下面几节主要说明分布式数据库系统中的透明性

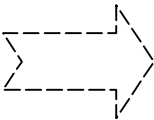
28.5.1.1 位置透明性

Oracle 分布式数据库系统有这样的特性，它允许应用开发者和管理员，对应用和用户隐藏数据库对象的物理位置。当用户可以到处参考数据库对象（如表），而不管应用所连接的节点时，就是位置透明性。位置透明性有几个好处，包括：

- 访问远程数据就简单了，因为数据库用户并不需要知道数据库对象的物理位置。
- 管理员能移动数据库对象，而不影响终端用户和现有的数据库应用。

典型的，管理员或开发者使用同义词来给表建立位置透明性，以及支持应用模式中的对象。例如，下面的语句在数据库中给另一个远程数据库中的表创建同义词：

```
CREATE PUBLIC SYNONYM emp
FOR scott.emp@sales.us.americas.acme_auto.com
CREATE PUBLIC SYNONYM dept
FOR scott.dept@sales.us.americas.acme_auto.com
```



现在，并非用一个如下的查询来访问远程表：

```
SELECT ename, dname
FROM scott.emp@sales.us.americas.acme_auto.com e,
scott.dept@sales.us.americas.acme_auto.com d
WHERE e.deptno = d.deptno;
```

而是应用可以发布比较简单的，不必介意远程表的位置的查询。

```
SELECT ename, dname
FROM emp e, dept d
WHERE e.deptno = d.deptno;
```

除了同义词之外，开发者还能使用视图和存储过程，为分布式数据库系统中的应用建立位置透明性。

28.5.1.2 SQL 和 COMMIT 透明性

Oracle 的分布式数据库结构还提供了查询、更新和事务透明性。例如，标准 SQL 语句（如 SELECT、INSERT、UPDATE 和 DELETE）就像在非分布式数据库环境中一样运作。另外，应用是使用标准的 SQL 语句 COMMIT、SAVEPOINT 和 ROLLBACK 来控制事务的——而不需要复杂的编程或为分布式事务控制提供其他的特殊的操作。

- 在单个事务中的语句，可以参照任意数量的本地或远程表。
- Oracle 保证包含在分布式事务中的所有节点做同样的动作：它们或者全部提交或者全部回滚事务。
- 如果在分布式事务提交的过程中，发生网络或系统故障，就自动地或透明地完全解析事务。特别地，当网络或系统恢复后，各个节点或者全部提交事务或者全部回滚事务。

在 Oracle 内部，每一个已提交的事务都有一个相关的系统修改号 SCN (system change number)，来惟一地标识事务内部的语句所做的修改。在分布式数据库中，调整通信节点的各个 SCN 的时机是，当：

- 用一个或多个数据库链接所描述的路径建立链接时。
- 执行分布式 SQL 语句时。
- 提交分布式事务时。

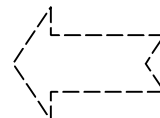
其他的好处是，在分布式数据库系统的各个节点中调整各个 SCN，就允许在语句和事务层上实现全局分布式的读一致性。如果需要的话，还可以实现全局分布式基于时间的恢复。

28.5.1.3 复制透明性

Oracle 还提供了许多在系统的节点之间透明地复制数据的特性。

28.5.2 远程过程调用 (RPC)

开发者可以编写 PL/SQL 包和过程，以支持用于分布式系统的应用。应用能使本地过程调用执行本地数据库的操作，远程过程调用 (RPC) 执行远程数据库的操作。



当一个程序调用一个远程过程时，本地服务器将所有的过程参数传递给该调用中的远程服务器。例如，下面的 PL/SQL 程序单元，调用位于远程 sales 数据库中的已打包的过程 del_emp，并将参数 1257 传送给它：

```
BEGIN
emp_mgmt.del_emp@sales.us.americas.acme_auto.com(1257);
END;
```

要使 RFC 成功执行，则被调用的过程必须存在于远程站点，被连接的用户必须拥有合适的权限以执行该过程。

当为分布式数据库系统开发包或过程时，开发者在编程时必须理解：程序单元在远程位置中做什么，以及如何将结果返回该调用应用。

28.5.3 分布式查询的优化

分布式查询优化是 Oracle 的特性，当一个事务从分布式 SQL 语句所参照的远程表中检索数据时，分布式查询优化减少了在各个站点之间所需传递的数据量。

分布式查询优化使用 Oracle 的基于开销的优化策略，来查找或生成 SQL 表达式，以便仅从远程表中提取必要的数 据，在远程站点或有时在本地站点处理那些数据，然后就将结果发送给本地站点做最后处理。这种操作相对于将所有的表的数据传给本地站点来做处理而言，减少了所需的数据传递量。

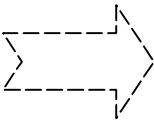
使用各种基于开销的优化器，隐含着如 DRIVING_SITE、NO_MERGE 和 INDEX，就能控制 Oracle 在什么地方处理数据或者怎样访问数据。

28.6 字符集的支持

Oracle 支持客户、Oracle 数据库服务器和非 Oracle 服务器使用不同字符集的环境。Oracle 为异构环境提供了 NCHAR 支持。可以设置各个国家的语言支持（National Language Support, NLS）和异构服务（Heterogeneous Services, HS）环境变量，以及初始化参数，以便控制在各个不同字符集之间的数据转换。

用如下的 NLS 和 HS 参数定义字符设置。

参数	环境	为谁定义
NLS_LANG (环境变量)	客户-服务器	客户
NLS_LANGUAGE	客户-服务器	Oracle 数据库服务器
NLS_CHARACTERSET	非异构分布式	
NLS_TERRITORY	异构分布式	



(续表)

参数	环境	为谁定义
HS_LANGUAGE	异构分布式	非 Oracle 服务器 透明网关
NLS_NCHAR (环境变量)	异构分布式	Oracle 数据库服务器 透明网关
HS_NLS_NCHAR		

28.6.1 客户/服务器环境

在一个客户/服务器环境中，将客户字符集设置成与 Oracle 数据库服务器字符集一样，或者是其字符集的一个子集，举例说明如图 28-6 所示：

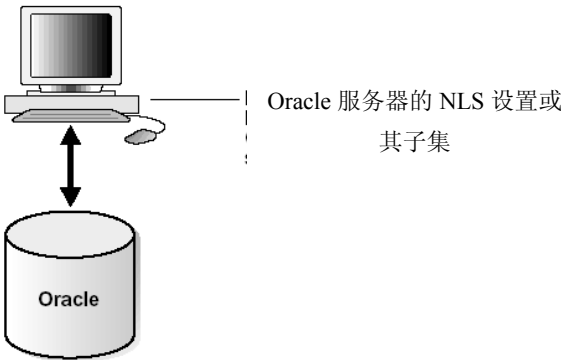


图 28-6 在客户-服务器环境中的 NLS 参数设置

28.6.2 同构分布式环境

在一个非异构环境中，客户和服务器的字符集应该与主服务器的字符集相同，或者是其字符集的一个子集，举例说明如图 28-7 所示：

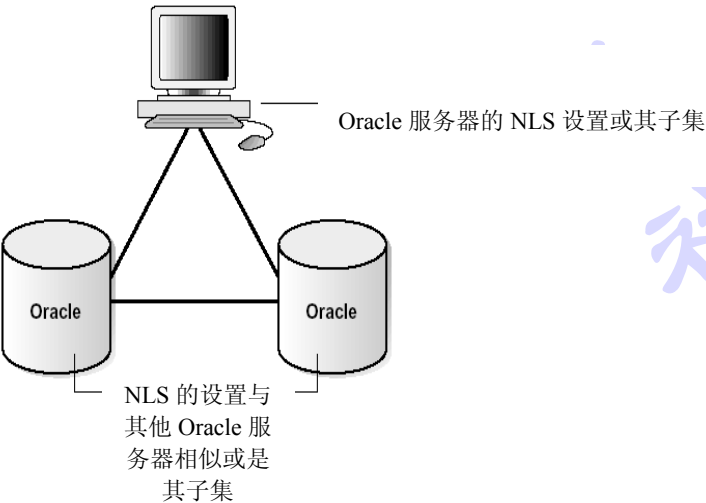
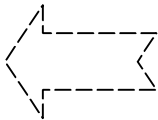


图 28-7 在同构环境中的 NLS 参数设置



28.6.3 异构分布式环境

在一个异构环境中，客户、透明网关和非 Oracle 数据源的 NLS 设置，应该与 Oracle 数据库服务器字符集一样，或者是其字符集的一个子集，举例说明如图 28-8 所示。透明网关支持所有 NLS。

在一个异构环境中，只有用 HS 技术建立的透明网关支持完整的 NCHAR 性能。一个特殊的透明网关是否支持 NCHAR，依赖于它所指向的非 Oracle 数据源。关于一个特殊的透明网关是如何处理对 NCHAR 的支持的有关信息，请参照特定系统的透明网关的文档。

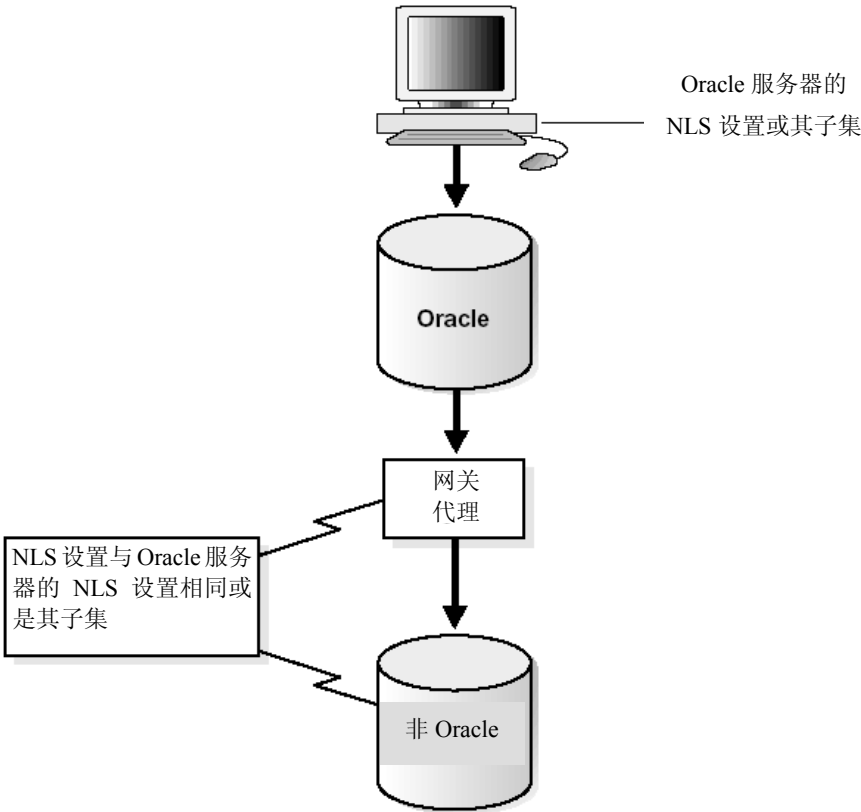
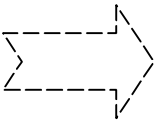


图 28-8 在异构环境中的 NLS 参数设置



第 29 章 管理分布式数据库

本章描述怎样管理和维护分布式数据库系统，包括如下内容：

- 管理分布式系统中的全局名
- 创建数据库链接
- 创建共享数据库链接
- 管理数据库链接
- 查看有关数据库链接的信息
- 创建位置透明性
- 管理语句透明性
- 管理一个分布式数据库：方案

29.1 管理分布式系统中的全局名

在分布式数据库系统中，每个数据库都应该有一个唯一的全局数据库名。全局数据库名唯一地标识系统中的数据库。在分布式系统中，主要的管理任务就是管理全局数据库名的创建和更改。

本节包括如下内容：

- 理解全局数据库名是怎样构成的
- 确定是否实施全局命名
- 查看全局数据库名
- 修改全局数据库名中的域
- 修改全局数据库名：方案

29.1.1 理解全局数据库名是怎样构成的

全局数据库名由两部分构成：数据库名和域。数据库创建时，下面的初始化参数确定数据库名和域名：

组成部分	参数	要求	例子
数据库名	DB_NAME	必须是 8 个或少于 8 个字符。	sales
包含数据库的域	DB_DOMAIN	必须遵循标准的 Internet 习惯。域名称中的层次用点分开，域名称的顺序是从叶子到根、从左到右。	us.acme.com

下面是有效的全局数据库名的例子：

DB_NAME	DB_DOMAIN	全局数据库名
sales	au.oracle.com	sales.au.oracle.com
sales	us.oracle.com	sales.us.oracle.com
mktg	us.oracle.com	mktg.us.oracle.com
payroll	nonprofit.org	payroll.nonprofit.org

DB_DOMAIN 初始化参数在数据库创建时是最重要的，使用它，连同 DB_NAME 参数一起，构成了数据库的全局名。此时，数据库的全局名存储在数据字典中。必须使用 ALTER DATABASE 语句修改全局名，而不能通过修改初始化参数文件中的 DB_NAME 参数来修改全局名。当然，在下次启动数据库之前，修改 DB_DOMAIN 参数以便反映域名的变化是个好习惯。

29.1.2 确定是否实施全局命名

在本地数据库上给链接提供的名称，依赖于要访问的远程数据库是否实施全局命名。如果远程数据库实施全局命名，那么就必须用远程数据库的全局数据库名作为链接的名称。例如，如果你被连接到本地 hq 服务器，又想创建一个到远程 mfg 数据库的链接，且 mfg 实施全局命名，那么你就必须使用 mfg 的全局数据库名作为链接名称。

也可以用服务名作为数据库链接名称的一部分。例如，如果使用服务名 sn1 和 sn2 连接到数据库 hq.acme.com，但 hq 强迫全局名称，那么你能创建下面的到 hq 的链接名称：

```
HQ.ACME.COM@SN1
HQ.ACME.COM@SN2
```

为了确定数据库是否实施数据库全局命名，要么检查数据库的初始化参数文件，要么查询 V\$PARAMETER 视图。例如，要看一看 mfg 数据库是否实施全局命名，可以在 mfg 上启动一个会话，然后创建并执行下面的 globalnames.sql 脚本（包括例子的输出）：

```
COL NAME FORMAT A12
COL VALUE FORMAT A6
SELECT NAME, VALUE FROM V$PARAMETER
WHERE NAME = 'global_names'
/
SQL> @globalnames
NAME VALUE
-----
global_names FALSE
```

29.1.3 查看全局数据库名

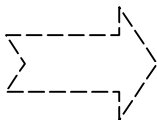
使用数据字典视图 GLOBAL_NAME 来查看数据库的全局名。例如，发布如下语句：

```
SELECT * FROM GLOBAL_NAME;
GLOBAL_NAME
-----
SALES.AU.ORACLE.COM
```

29.1.4 修改全局数据库名中的域

使用 ALTER DATABASE 语句来修改数据库的全局名中的域。注意数据库被创建之后，修改初始化参数 DB_DOMAIN 不影响全局数据库名或数据库链接名称的解析。

下面的例子说明重命名语句的语法，此处的 *database* 是数据库名，*domain* 是网络域：



```
ALTER DATABASE RENAME GLOBAL_NAME TO database.domain;
```

使用如下过程修改全局数据库名中的域：

①确定当前的全局数据库名。例如：发布如下语句：

```
SELECT * FROM GLOBAL_NAME;
GLOBAL_NAME
```

```
-----
SALES.AU.ORACLE.COM
```

②使用 ALTER DATABASE 语句重新命名全局数据库名。例如，输入：

```
ALTER DATABASE RENAME GLOBAL_NAME TO sales.us.oracle.com;
```

③查询 GLOBAL_NAME 表，检查新的名称。例如，输入：

```
SELECT * FROM GLOBAL_NAME;
GLOBAL_NAME
```

```
-----
SALES.US.ORACLE.COM
```

29.1.5 修改全局数据库名：方案

在此方案中，修改本地数据库的全局数据库名的域部分。同时还用部分指定的全局名创建数据库链接，以测试 Oracle 如何解析名称。结果发现，Oracle 是使用本地数据库的当前的全局数据库名的域部分，而不是初始化参数 DB_DOMAIN 的值来解析该部分名称的。

①连接到 SALES.US.ACME.COM 并查询 GLOBAL_NAME 数据字典视图，以确定数据库的当前的全局名：

```
CONNECT SYSTEM/password@sales.us.acme.com
SELECT * FROM GLOBAL_NAME;
GLOBAL_NAME
```

```
-----
SALES.US.ACME.COM
```

②查询 V\$PARAMETER 视图，以确定 DB_DOMAIN 初始化参数的当前设置值：

```
SELECT NAME, VALUE FROM V$PARAMETER WHERE NAME = 'db_domain';
NAME VALUE
```

```
-----
db_domain US.ACME.COM
```

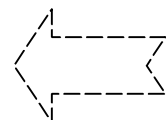
③然后，仅使用部分指定的全局名，创建到数据库 hq 的数据库链接：

```
CREATE DATABASE LINK hq USING 'sales';
```

对于这种链接，Oracle 通过把本地数据库的全局数据库名的域部分，添加到在链接中指定的数据库的名称上，来展开全局数据库名。

④查询 USER_DB_LINKS，以确定 Oracle 是使用哪个域名称来解析部分指定的全局数据库名：

```
SELECT DB_LINK FROM USER_DB_LINKS;
```



```
DB_LINK
```

```
-----
HQ.US.ACME.COM
```

此结果表明本地数据库的全局数据库名的域部分是 `us.acme.com`。当创建数据库链接时，Oracle 使用该域来解析部分数据库链接名。

⑤因为已经收到了消息，`sales` 数据库将移动到日本，所以将 `sales` 数据库重新命名为 `sales.jp.acme.com`：

```
ALTER DATABASE RENAME GLOBAL_NAME TO sales.jp.acme.com;
SELECT * FROM GLOBAL_NAME;
GLOBAL_NAME
-----
```

```
SALES.JP.ACME.COM
```

⑥再次查询 `V$PARAMETER` 视图，会发现尽管重新命名了全局数据库名的域部分，但是 `DB_DOMAIN` 的值并没有改变：

```
SELECT NAME, VALUE FROM V$PARAMETER
WHERE NAME = 'db_domain';
NAME VALUE
-----
```

```
db_domain US.ACME.COM
```

此结果表明 `DB_DOMAIN` 初始化参数的值不依赖于 `ALTER DATABASE RENAME GLOBAL_NAME` 语句。`ALTER DATABASE` 语句决定全局数据库名的域，而不是 `DB_DOMAIN` 初始化参数（尽管修改 `DB_DOMAIN` 以便反映新的域名是个好习惯）。

⑦创建另一个到数据库 `supply` 的数据库链接，然后查询 `USER_DB_LINKS`，看一看 Oracle 是如何解析 `supply` 的全局数据库名的域部分的：

```
CREATE DATABASE LINK supply USING 'supply';
SELECT DB_LINK FROM USER_DB_LINKS;
DB_LINK
-----
```

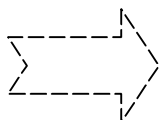
```
HQ.US.ACME.COM
```

```
SUPPLY.JP.ACME.COM
```

此结果表明 Oracle 是用域 `jp.acme.com` 来解析部分指定的链接名称的。因为该域是本地数据库的全局数据库名的域部分，所以创建链接时，使用此域。当解析部分链接名称时，Oracle 并不使用 `DB_DOMAIN` 初始化参数设置值。

⑧后来，收到消息，说以前的信息不对：`sales` 应该在 `ASIA.JP.ACME.COM` 域中，而不是在 `JP.ACME.COM` 域中。其结果是，重新命名全局数据库名如下：

```
ALTER DATABASE RENAME GLOBAL_NAME TO sales.asia.jp.acme.com;
SELECT * FROM GLOBAL_NAME;
GLOBAL_NAME
```



```
SALES.ASIA.JP.ACME.COM
```

⑨查询 V\$PARAMETER 视图，再次检查参数 DB_DOMAIN 的设置值：

```
SELECT NAME, VALUE FROM V$PARAMETER
WHERE NAME = 'db_domain';
NAME VALUE
```

```
db_domain US.ACME.COM
```

此结果表明在参数文件中的域设置值，与发布任意一个 ALTER DATABASE RENAME 语句之前的设置值完全一样。

⑩最后，创建到 warehouse 数据库的链接，并再次查询 USER_DB_LINKS 以确定 Oracle 是如何解析部分指定的全局名：

```
CREATE DATABASE LINK warehouse USING 'warehouse';
SELECT DB_LINK FROM USER_DB_LINKS;
DB_LINK
-----
HQ.US.ACME.COM
SUPPLY.JP.ACME.COM
WAREHOUSE.ASIA.JP.ACME.COM
```

再次看到，Oracle 在创建链接期间是使用的本地数据库的全局数据库名的域部分来展开部分链接名称的。

注意：为了改正 supply 数据库链接，必须先删除它，然后再重新创建。

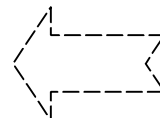
29.2 创建数据库链接

为了支持应用通过分布式数据库系统访问数据和模式对象，必须创建所有必要的数据库链接。本节包括如下内容：

- 获取创建数据库链接所需的权限
- 指定链接类型
- 指定链接用户
- 使用连接限定词在链接名称中指定服务名

29.2.1 获取创建数据库链接所需的权限

数据库链接是本地数据库中的一个指针，允许访问远程数据库中的对象。要想创建一个私有数据库链接，你必须被授予了相应的权限。下表说明在何种数据库上对于哪种类型的链接需要哪种权限：



权限	数据库	所需的
CREATE DATABASE LINK	本地	创建私有数据库链接。
CREATE PUBLIC DATABASE LINK	本地	创建公共数据库链接。
CREATE SESSION	远程	创建所有类型的数据库链接。

为了看一看你当前可用的权限，就查询 `ROLE_SYS_PRIVS`。例如，你可以创建并且执行下面的 `privs.sql` 脚本（包括例子的输出）：

```
SELECT DISTINCT PRIVILEGE AS "Database Link Privileges"
FROM ROLE_SYS_PRIVS
WHERE PRIVILEGE IN ( 'CREATE SESSION','CREATE DATABASE LINK',
'CREATE PUBLIC DATABASE LINK')
/

SQL> @privs
Database Link Privileges
-----
CREATE DATABASE LINK
CREATE PUBLIC DATABASE LINK
CREATE SESSION
```

29.2.2 指定链接类型

创建数据库链接时，必须确定谁将访问该数据库链接。如下各节描述如何创建三种基本类型的链接：

- 创建私有数据库链接
- 创建公共数据库链接
- 创建全局数据库链接

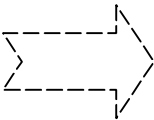
29.2.2.1 创建私有数据库链接

为了创建私有数据库链接，指定下面的语句（此处的 `link_name` 是全局数据库名或一个任意的链接名称）：

```
CREATE DATABASE LINK link_name ...;
```

下面是私有数据库链接的例子：

SQL 语句Λ	创建Λ
CREATE DATABASE LINK supply.us.acme.com;	一个使用全局数据库名的、通向远程 <code>supply</code> 数据库的私有链接。 该链接使用已连接用户的用户标识/口令（ <code>userid/password</code> ）。 所以，如果 <code>scott</code> （由 <code>tiger</code> 标识）在查询中使用该链接，则该链接就按 <code>scott/tiger</code> 来建立到远程数据库的连接。



(续表)

SQL 语句A	创建A
CREATE DATABASE LINK link_2 CONNECT TO jane IDENTIFIED BY doe USING 'us_supply';	一个被称为 link_2 的、通向服务名为 us_supply 的数据库的私有固定用户链接。该链接用 jane/doe 的用户标识/口令连接到远程数据库，而不考虑已连接的用户。
CREATE DATABASE LINK link_1 CONNECT TO CURRENT_USER USING 'us_supply';	一个被成为 link_1 的、通向服务名为 us_supply 的数据库的私有链接。该链接使用当前用户的用户标识/口令来登录到远程数据库。 注意：当前用户与已连接用户可能不是同一个，但必须是链接中所包括的两个数据库中的全局用户（参见第 28 章的“数据库链接的用户”）。当前用户链接是 Oracle 高级安全选项的一部分。

29.2.2.2 创建公共数据库链接

为了创建公共数据库链接，使用关键字 PUBLIC（此处的 link_name 是全局数据库名或一个任意的链接名称）：

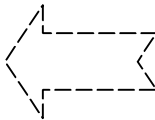
```
CREATE PUBLIC DATABASE LINK link_name ...;
```

下面是公共数据库链接的例子：

SQL 语句A	创建A
CREATE PUBLIC DATABASE LINK supply.us.acme.com;	一个通向远程 supply 数据库的公共链接。该链接使用已连接用户的用户标识/口令。所以，如果 scott（由 tiger 标识）在查询中使用该链接，则该链接就按 scott/tiger 来建立到远程数据库的连接。
CREATE PUBLIC DATABASE LINK pu_link CONNECT TO CURRENT_USER USING 'supply';	一个被称为 up_link 的、通向服务名为 supply 的数据库的公共链接。该链接使用当前用户的用户标识/口令来登录到远程数据库。 注意：当前用户与已连接用户可能不是同一个，但必须是链接中所包括的两个数据库中的全局用户（参见第 28 章“数据库链接的用户”）
CREATE PUBLIC DATABASE LINK sales.us.acme.com CONNECT TO jane IDENTIFIED BY doe ;	一个通向远程 sales 数据库的公共的固定用户链接。该链接用 jane/doe 的用户标识/口令连接到远程数据库。

29.2.2.3 创建全局数据库链接

必须在 Oracle 名称服务器（Oracle Names Server）中定义全局数据库链接。



29.2.3 指定链接用户

数据库链接定义了一个从一个数据库到另一个数据库的通信路径。当应用使用数据库链接来访问远程数据库时，Oracle 按本地应用的需要，在远程数据库中创建一个数据库会话。

当创建私有或公共数据库链接时，通过创建固定用户、当前用户和已连接用户的数据库链接，就能确定该链接将建立到远程数据库中的哪个模式的连接。

29.2.3.1 创建固定用户数据库链接

为了创建固定用户数据库链接，就要在链接定义中嵌入访问远程数据库所需要的信用证书（在这种情况下就是用户名称或口令）：

```
CREATE DATABASE LINK ... CONNECT TO username IDENTIFIED BY password ...;
```

下面是固定用户数据库链接的例子：

SQL 语句 A	创建 A
CREATE PUBLIC DATABASE LINK supply.us.acme.com CONNECT TO scott AS tiger;	一个使用全局数据库名称、通向远程 supply 数据库的公共链接。该链接用用户标识/口令 scott/tiger 来连接到远程数据库。
CREATE DATABASE LINK foo CONNECT TO jane IDENTIFIED BY doe USING 'finance';	一个被称为 foo 的、通向服务名为 finance 的数据库的私有的固定用户链接。该链接用用户标识/口令 jane/doe 来连接到远程数据库。

当一个应用使用固定用户数据库链接时，本地服务器总是建立到远程数据库中固定的远程模式的连接。当应用使用该链接来访问远程数据库时，本地服务器还通过网络发送固定用户的信用证书。

29.2.3.2 创建已连接用户与当前用户的数据库链接

已连接用户与当前用户的数据库链接不包括在链接定义中的信用证书。用于连接到远程数据库的信用证书能修改，这依赖于参照该数据库链接的用户和应用所执行的操作。

注意：对于许多分布式应用，可能不想让用户在远程数据库中拥有权限。实现这个目的的一种简单方法是：创建一个在其中包含固定用户与当前用户的数据库链接的过程。用这种方法，访问该过程的用户临时使用其他人的权限。

关于已连接用户与当前用户之间差别的展开的概念讨论，参见第 28 章的“数据库链接的用户”。

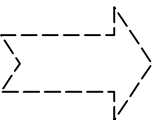
创建一个已连接用户的数据库链接 为了创建已连接用户的数据库链接，省略 CONNECT TO 子句。下面的语法创建了一个已连接用户的数据库链接，此处的 *dblink* 是链接的名称，*net_service_name* 是可选择的连接字符串：

```
CREATE [SHARED] [PUBLIC] DATABASE LINK dblink ... [USING 'net_service_name'];
```

例如，创建一个已连接用户的数据库链接，使用下面的语法：

```
CREATE DATABASE LINK sales.division3.acme.com USING 'sales';
```

创建一个当前用户的数据库链接 为了创建当前用户的数据库链接，在创建链接的语句中，使用 CONNECT TO CURRENT_USER 子句。当前用户的链接只有通过 Oracle 高级安



全才可以使用。

下面的语法创建一个当前用户的数据库链接，此处的 *dblink* 是链接的名称，*net_service_name* 是可选的连接字符串：

```
CREATE [SHARED] [PUBLIC] DATABASE LINK dblink CONNECT TO CURRENT_USER
[USING 'net_service_name'];
```

例如，创建一个到 *sales* 数据库的当前用户的数据库链接，可以使用如下语法：

```
CREATE DATABASE LINK sales CONNECT TO CURRENT_USER USING 'sales';
```

注意：为了使用当前用户数据库链接，当前用户必须是链接中所包含的两个数据库中的全局用户。

29.2.4 使用连接限定词在链接名称中指定服务名

在某些情况下，可能希望有几个相同类型的、指向相同远程数据库的数据库链接（例如，公共的），但是使用不同的通信途径来建立到远程数据库的连接。这种策略在如下情况很有效：

- 远程数据库是 Oracle 真实应用簇（Oracle Real Application Clusters）配置中的一部分，所以，在本地节点上定义了几个公共数据库链接，以便建立到远程数据库中的指定实例的连接。
- 某些客户使用 TCP/IP 协议连接到 Oracle 服务器，而其他客户则使用 DECNET 协议。

为了促进这种功能，Oracle 允许在数据库链接名称中带上可选的服务名来创建一个数据库链接。当创建数据库链接时，服务名被指定作为数据库链接名称的尾部，由一个 @ 符号分开，例如 @sales。将该字符串叫做连接限定词。

例如，假设远程数据库 *hq.acme.com* 是在 Oracle 真实应用簇环境中被管理的。*hq* 数据库有两个实例，名为 *hq_1* 和 *hq_2*。本地数据库能包含如下公共数据库的链接，以定义到 *hq* 数据库的远程实例的路径：

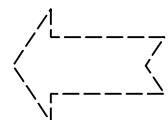
```
CREATE PUBLIC DATABASE LINK hq.acme.com@hq_1
USING 'string_to_hq_1';
CREATE PUBLIC DATABASE LINK hq.acme.com@hq_2
USING 'string_to_hq_2';
CREATE PUBLIC DATABASE LINK hq.acme.com
USING 'string_to_hq';
```

注意在前两个例子中，服务名的确是数据库链接名称的一部分。服务名的正文不需要表明如何建立链接；而是在 USING 子句的服务名中指定此信息。还要注意，在第三个例子中，没将服务名指定为链接名称的一部分。在这种情况下，就像将服务名指定为链接名称的一部分那样，是通过 USING 字符串来确定实例的。

为了使用服务名来指定一个特殊的实例，应在全局对象名的末尾包括服务名：

```
SELECT * FROM scott.emp@hq.acme.com@hq_1
```

注意在此例子中，有两个 @ 符号。



29.3 创建共享数据库链接

使用标准数据库链接来参照远程服务器每个应用，会在本地数据库和远程数据库之间建立一个链接。同时运行应用的多个用户可能引起本地数据库和远程数据库之间存在大量的连接。

共享数据库链接能限制在本地服务器和远程服务器之间所需的网络连接数量。

本节包括如下内容：

- 确定是否使用共享数据库链接
- 创建共享数据库链接
- 配置共享数据库链接

29.3.1 确定是否使用共享数据库链接

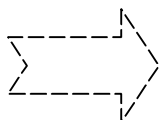
仔细查看应用和共享服务器配置，以确定是否使用共享链接。一个基本的方针是，当访问数据库链接的用户数量可能远远大于本地数据库中服务器进程的数量时，就使用共享数据库链接。

下表说明了有关数据库链接的三种可能的配置：

链接类型	服务器模式	结果
非共享 (Non-Shared)	专用/共享服务器	如果应用使用标准的公共数据库链接，而且 100 个用户同时都需要一个连接，那么就需要 100 个直接通向远程数据库的网络连接。
共享 (Shared)	共享服务器	如果在本地共享服务器模式数据库中，存在 10 个共享服务器进程，那么 100 个使用相同数据库链接用户，就需要 10 个或更少的到远程服务器的网络连接。每个本地共享服务器进程仅需要一个通向远程服务器的连接。
共享 (Shared)	专用	如果 10 个客户连接到本地专用服务器上，且每个客户在相同的连接上都有 10 个会话（所以全部合起来，建立 100 个会话），而且每一个会话参照相同的远程数据库，那么仅需要 10 个连接。如用非共享数据库链接，就需要 100 个连接。

共享数据库链接并不是在所有情况下都有用。假设仅有一个用户访问远程服务器，如果该用户定义一个共享数据库链接，并且在本地数据库中，存在 10 个共享服务器进程，那么该用户就能获得最多 10 个通向远程服务器的网络连接。因为用户能使用每个共享服务器进程，而每个进程都能建立一个到远程服务器的连接。

很明显，在这种情况下非共享数据库链接更好些，因为它仅需要一个网络连接。在单一用户情况下，共享数据库链接导致了更多的网络连接，所以，仅当很多用户需要使用相同的链接时，才使用共享链接。典型地，共享链接适用于公共数据库链接，但是当很多客户访问相同的本地模式（因此也是相同的私有数据库链接）时，共享链接也能用于私有数据库链接。



29.3.2 创建共享数据库链接

为了创建共享数据库链接，在 CREATE DATABASE LINK 语句中，使用关键字 SHARED。

```
CREATE SHARED DATABASE LINK dblink_name
[CONNECT TO username IDENTIFIED BY password] | [CONNECT TO CURRENT_USER]
AUTHENTICATED BY schema_name IDENTIFIED BY password
[USING 'service_name'];
```

下面的例子创建了一个固定用户的、到 sales 数据库的共享链接，按 scott 连接并按 keith 验证：

```
CREATE SHARED DATABASE LINK link2sales
CONNECT TO scott IDENTIFIED BY tiger
AUTHENTICATED BY keith IDENTIFIED BY richards
USING 'sales';
```

无论何时使用关键字 SHARED，都需要 AUTHENTICATED BY 子句。在 AUTHENTICATED BY 子句中指定的模式，仅仅是出于安全的原因，可被看作是虚设的模式。当使用共享数据库链接时，它并不受影响，它也不影响共享数据库链接的用户。

需要 AUTHENTICATED BY 子句是为了防止未被授权的客户伪装成数据库链接用户并且获得访问有权限的信息。

29.3.3 配置共享数据库链接

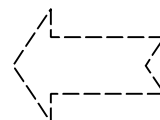
用下面的方法能配置共享数据库链接：

- 创建通向专用服务器的共享链接
- 创建通向共享服务器的共享链接

29.3.3.1 创建通向专用服务器的共享链接

图 29-1 说明了这种配置，即本地服务器中的共享服务器进程拥有一个专用远程服务器进程。这种配置的优点是，在本地共享服务器和远程专用服务器之间存在直接的网络传输。其缺点是需要一个额外的后台服务器进程。

注意：远程服务器既可以是一个共享服务器也可以是一个专用服务器。在本地服务器和远程服务器之间存在一个专用连接。当远程服务器是一个共享服务器时，通过在服务名称定义中使用（SERVER = DEDICATED）子句，就能实施一个专用服务器连接。



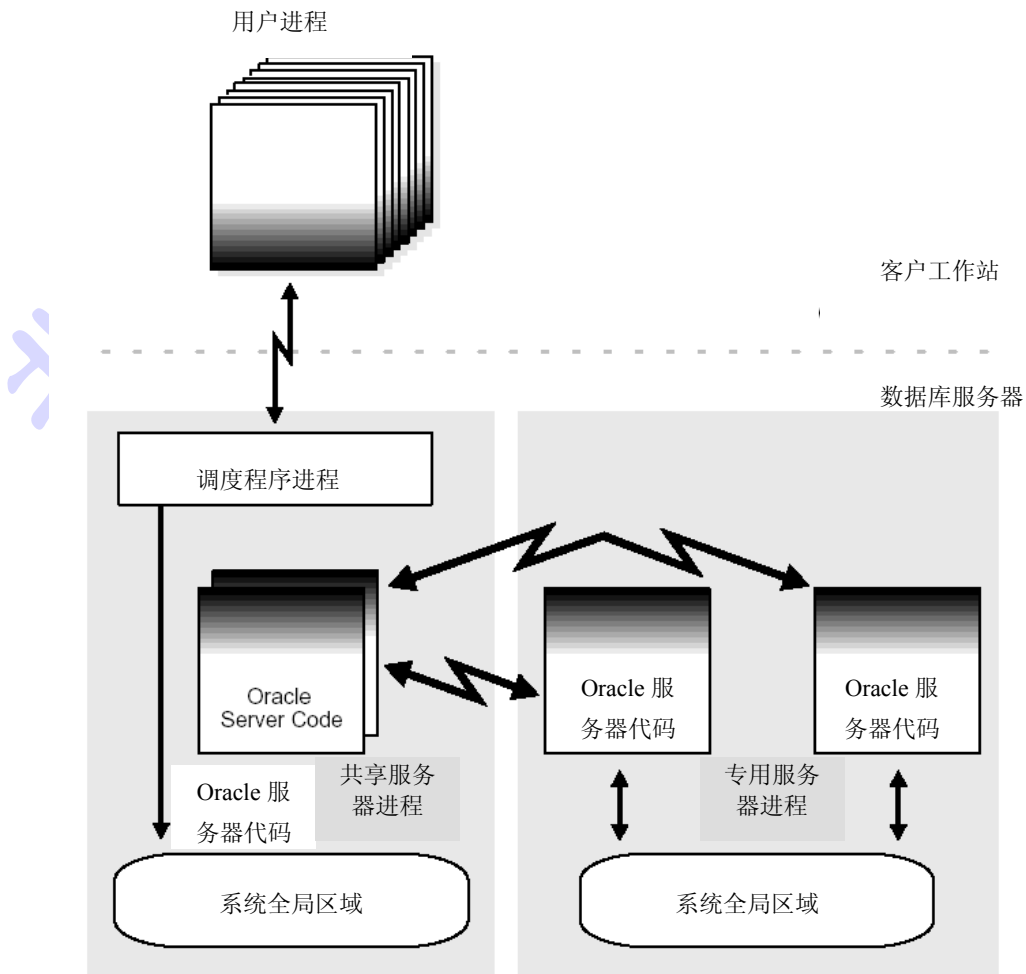
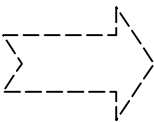


图 29-1 通向专用服务器进程的共享数据库链接

29.3.3.2 创建通向共享服务器的共享链接

图 29-2 说明了这种配置，即在远程服务器上使用共享服务器进程。这种配置消除了对更多专用服务器的需要，但需要通过远程服务器上的调度程序的连接。注意，必须将本地服务器和远程服务器二者配置成共享服务器。



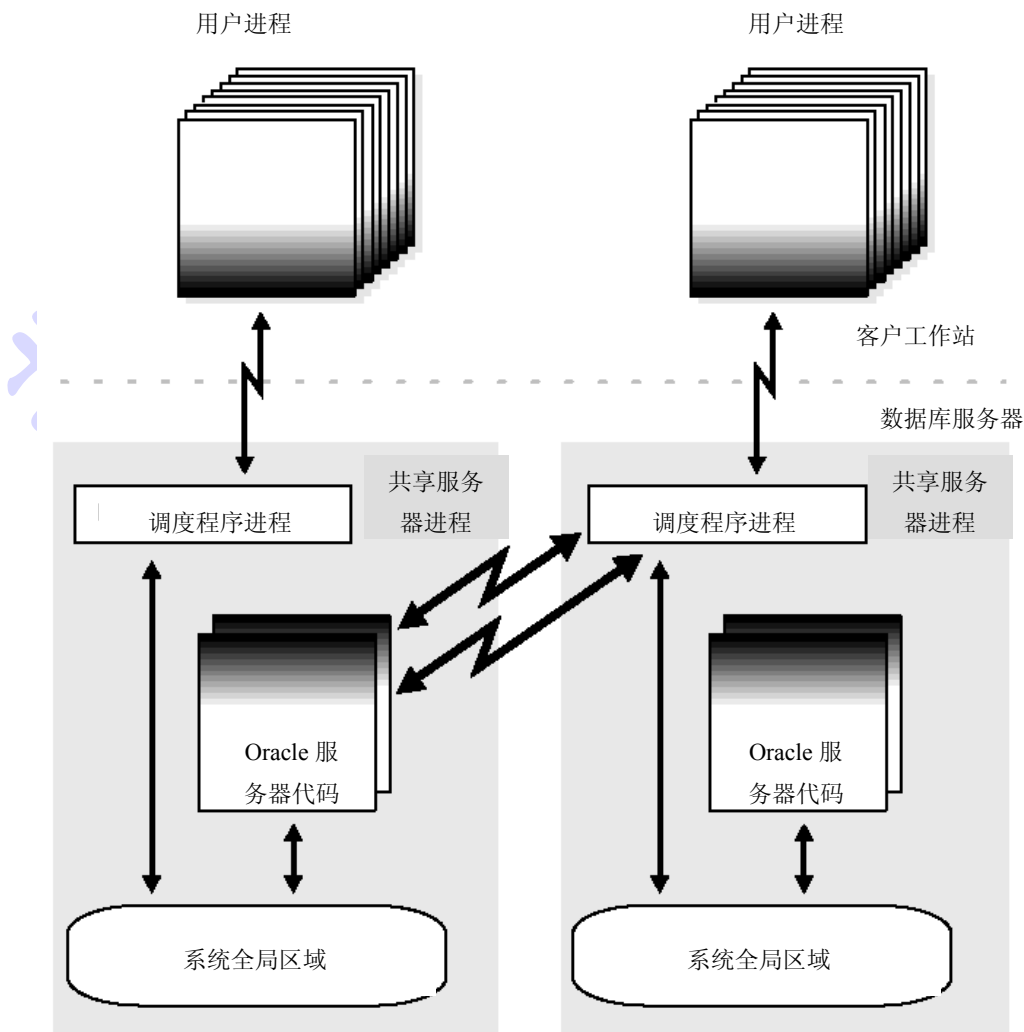


图 29-2 通向共享服务器的共享数据库链接

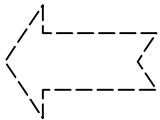
29.4 管理数据库链接

本节包括如下内容：

- 关闭数据库链接
- 删除数据库链接
- 限制激活的数据库链接的连接数量

29.4.1 关闭数据库链接

如果在会话中访问数据库链接，那么该链接保持打开状态，直到关闭此对话。如果一个连接是打开的，那么通过该链接所访问的每个远程数据库上都有一个激活的进程。这种情况会产生如下的后果：



- 如果 20 个用户打开会话，并且访问本地数据库中相同的公共链接，那么就打开了 20 个数据库链接的连接。
- 如果 20 个用户打开会话，并且每个用户访问一个私有链接，那么就打开了 20 个数据库链接的连接。
- 如果一个用户启动一个会话，并且访问 20 个不同的链接，那么就打开了 20 个数据库链接的连接。

关闭一个会话，就自动关闭了会话中的激活的链接。有些场合可能要手动关闭链接。

例如：当出现下面的情况时，就要手动关闭链接：

- 在应用中很少再使用的由链接所建立的网络连接。
- 必须结束用户会话。

如果要关闭链接，可以发布下面的语句，此处的 *linkname* 为链接的名称：

```
ALTER SESSION CLOSE DATABASE LINK linkname;
```

注意该语句仅关闭在当前会话中处于激活状态的链接。

29.4.2 删除数据库链接

就像能删除表或视图一样，也能删除数据库链接。如果链接是私有的，那么该链接必须要在你的模式中。如果链接是公共的，那么就必须拥有 DROP PUBLIC DATABASE LINK 系统权限。

删除数据库链接的语句的语法如下，此处的 *dblink* 是链接的名称：

```
DROP [PUBLIC] DATABASE LINK dblink;
```

29.4.2.1 删除私有数据库链接的过程

①使用 SQL*Plus 连接到本地数据库。例如，输入：

```
CONNECT scott/tiger@local_db
```

②查询 USER_DB_LINKS，以便查看你所拥有的链接。例如，输入：

```
SELECT DB_LINK FROM USER_DB_LINKS;
DB_LINK
```

```
-----
SALES.US.ORACLE.COM
```

```
MKTG.US.ORACLE.COM
```

```
2 rows selected.
```

③使用 DROP DATABASE LINK 语句删除想删的链接。例如，输入：

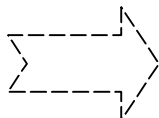
```
DROP DATABASE LINK sales.us.oracle.com;
```

29.4.2.2 删除公共数据库链接的过程

①以具有 DROP PUBLIC DATABASE LINK 权限的用户身份连接到本地数据库。例如，输入：

```
CONNECT SYSTEM/password@local_db AS SYSDBA
```

②查询 DBA_DB_LINKS，以便查看公共链接。例如，输入：



```

SELECT DBA_DB_LINK FROM USER_DB_LINKS
WHERE OWNER = 'PUBLIC';

DB_LINK
-----
DBL1.US.ORACLE.COM
SALES.US.ORACLE.COM
INST2.US.ORACLE.COM
RMAN2.US.ORACLE.COM
4 rows selected.

```

③使用 `DROP PUBLIC DATABASE LINK` 语句删除想删的连接。例如，输入：

```
DROP PUBLIC DATABASE LINK sales.us.oracle.com;
```

29.4.3 限制激活的数据库链接的连接数量

使用静态的初始化参数 `OPEN_LINKS`，能限制从用户进程到远程数据库连接的数量。该参数在分布式事务中控制单个用户会话能够同时连接的远程连接的数量。

设置 `OPEN_LINKS` 参数时，要注意考虑如下问题：

- 该值应大于或等于参照多个数据库的、在单条 SQL 语句中所涉及的数据库的数量。
- 如果以后要访问多个分布式数据库，就增加该值。所以，如果你一般会访问三个数据库，就将 `OPEN_LINKS` 设置为 3 或者更大。
- `OPEN_LINKS` 默认值为 4。如果将 `OPEN_LINKS` 设置成 0，那就不允许有分布式事物了。

29.5 查看有关数据库链接的信息

每个数据库的数据字典存储着数据库中所有数据库链接的定义。能使用数据字典的表及视图来获得有关链接的信息。本节包括如下内容：

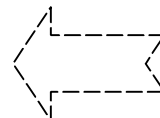
- 确定数据库中有哪些链接
- 确定哪些链接的连接是打开的

29.5.1 确定数据库中有哪些链接

下面的视图说明了已经在本地数据库中定义了的、并且存储在数据字典中的数据库链接：

视图	目的
DBA_DB_LINKS	列出了已连接用户所能访问的所有数据库链接。
ALL_DB_LINKS	列出了数据库中所有的数据库链接。
USER_DB_LINKS	列出了已连接用户所拥有的所有数据库链接。

这些数据字典视图包含了关于数据库链接的相同的基本信息，只有某些例外：



列	哪个视图	说明
OWNER	除 USER_* 之外，全部	创建数据库链接的用户。如果链接是公共的，那么用户被列为 PUBLIC。
DB_LINK	全部	数据库链接的名称。
USERNAME	全部	如果链接定义包括固定用户，那么该列显示固定用户的用户名。如果没有固定用户，则此列为 NULL。
PASSWORD	仅仅 USER_*	登录到远程数据库的口令。
HOST	全部	用于连接到远程数据库的网络服务名。
CREATED	全部	数据库链接的创建时间。

任何用户都能查询 USER_DB_LINKS，以确定该用户可以使用哪一个数据库链接。只有带有附加权限的那些用户才能使用 ALL_DB_LINKS 或 DBA_DB_LINKS 视图。

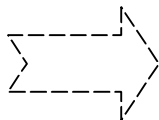
下面的脚本查询 DBA_DB_LINKS 视图，以获取链接信息：

```
COL OWNER FORMAT a10
COL USERNAME FORMAT A8 HEADING "USER"
COL DB_LINK FORMAT A30
COL HOST FORMAT A7 HEADING "SERVICE"
SELECT * FROM DBA_DB_LINKS
/
这里，激活该脚本，其结果输出如下：
SQL>@link_script
OWNER DB_LINK USER SERVICE CREATED
-----
SYS TARGET.US.ACME.COM SYS inst1 23-JUN-99
PUBLIC DBL1.UK.ACME.COM BLAKE ora51 23-JUN-99
PUBLIC RMAN2.US.ACME.COM inst2 23-JUN-99
PUBLIC DEPT.US.ACME.COM inst2 23-JUN-99
JANE DBL.UK.ACME.COM BLAKE ora51 23-JUN-99
SCOTT EMP.US.ACME.COM SCOTT inst2 23-JUN-99
6 rows selected.
```

29.5.1.1 用于查看口令信息的授权

仅仅是 USER_DB_LINKS 包含了一列口令信息。但是，如果你是管理级用户（按 AS SYSDBA 身份连接的 SYS 或用户），那么就能通过查询 LINKS 表，查看到数据库中所有链接的口令。如果你不是管理级用户，那么可以通过下面两种方式中的一种来授权以查询 LINKS 表：

- 授予针对 LINKS 表的特殊的对象权限
- 授予 SELECT ANY DICTIONARY 系统权限



29.5.1.2 查看口令信息

可以在 SQL*Plus 中创建并运行下面的脚本，以获取口令信息（包括例子输出）：

```
COL USERID FORMAT A10
COL PASSWORD FORMAT A10
SELECT USERID,PASSWORD
FROM SYS.LINK$
WHERE PASSWORD IS NOT NULL
/
SQL>@linkpwd
USERID PASSWORD
-----
SYS ORACLE
BLAKE TYGER
SCOTT TIGER
3 rows selected.
```

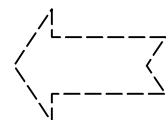
29.5.1.3 查看鉴别口令

通过查询 LINK\$ 表，可以查看数据库中的所有链接的、AUTHENTICATED BY（由...鉴别）、IDENTIFIED BY（由...标识）的用户名称和口令。可以在 SQL*Plus 中创建并运行下面的脚本，以获取口令信息（包括例子输出）：

```
COL AUTHUSR FORMAT A10
COL AUTHPWD FORMAT A10
SELECT AUTHUSR AS userid, AUTHPWD AS password
FROM SYS.LINK$
WHERE PASSWORD IS NOT NULL
/
SQL> @authpwd
USERID PASSWORD
-----
ELLIE MAY
1 row selected.
```

还可以通过创建并执行下面的脚本，在一个链接中查看合在一起的链接和口令信息（包括例子的输出）：

```
COL OWNER FORMAT A8
COL DB_LINK FORMAT A15
COL USERNAME FORMAT A8 HEADING "CON_USER"
COL PASSWORD FORMAT A8 HEADING "CON_PWD"
COL AUTHUSR FORMAT A8 HEADING "AUTH_USER"
```




```
COL AUTHPWD FORMAT A8 HEADING "AUTH_PWD"
COL HOST FORMAT A7 HEADING "SERVICE"
COL CREATED FORMAT A10
SELECT DISTINCT d.OWNER,d.DB_LINK,d.USERNAME,l.PASSWORD,
l.AUTHUSR,l.AUTHPWD,d.HOST,d.CREATED
FROM DBA_DB_LINKS d, SYS.LINK$ l
WHERE PASSWORD IS NOT NULL
AND d.USERNAME = l.USERID
/

SQL> @user_and_pwd
OWNER DB_LINK CON_USER CON_PWD AUTH_USE AUTH_PWD SERVICE CREATED
-----
JANE DBL.ACME.COM BLAKE TYGER ELLIE MAY ora51 23-JUN-99
PUBLIC DBL1.ACME.COM SCOTT TIGER ora51 23-JUN-99
SYS TARGET.ACME.COM SYS ORACLE inst1 23-JUN-99
```

29.5.2 确定哪些链接的连接是打开的

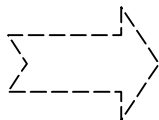
你可能会发现，确定出在你会话中当前所打开的数据库链接的连接是有用的。注意，如果是按 SYSDBA 连接的，就不能通过查询视图来确定所有的会话所打开的链接；而仅能获取正在使用的会话中的链接信息。

下面的视图显示了在当前会话中当前打开的数据库链接的连接：

视图	目的
V\$DBLINK	列出了会话中所有打开的数据库链接，即所有的 IN_TRANSACTION 列被设置成 YES 的数据库链接。
GV\$DBLINK	列出了会话中（连同它们相应的实例）所有打开的数据库链接。在 Oracle 真实应用簇配置中，该视图很有用。

下面的数据字典视图包含了有关数据库链接的相同的基本信息，只有一个例外：

列	哪个视图	说明
DB_LINK	全部	数据库链接的名称。
OWNER_ID	全部	数据库链接的拥有者。
LOGGED_ON	全部	是否数据库链接当前被登录。
HETEROGENEOUS	全部	数据库链接是同构的（NO）还是异构的（YES）。
PROTOCOL	全部	用于数据库链接的通信协议。
OPEN_CURSORS	全部	是否打开了用于数据库链接的游标。
IN_TRANSACTION	全部	是否在还没有提交或回滚的事物中访问数据库链接。
UPDATE_SENT	全部	是否在数据库链接中存在更新。



(续表)

列	哪个视图	说明
COMMIT_POINT_STRENGTH	全部	使用数据库链接的事务的提交点强度。
INST_ID	仅为 GV\$DBLINK	从其获取视图信息的实例。

例如：可以创建并执行下面的脚本，以确定哪些链接是打开的（包括例子输出）：

```
COL DB_LINK FORMAT A25
COL OWNER_ID FORMAT 99999 HEADING "OWNID"
COL LOGGED_ON FORMAT A5 HEADING "LOGON"
COL HETEROGENEOUS FORMAT A5 HEADING "HETER"
COL PROTOCOL FORMAT A8
COL OPEN_CURSORS FORMAT 999 HEADING "OPN_CUR"
COL IN_TRANSACTION FORMAT A3 HEADING "TXN"
COL UPDATE_SENT FORMAT A6 HEADING "UPDATE"
COL COMMIT_POINT_STRENGTH FORMAT 99999 HEADING "C_P_S"
SELECT * FROM V$DBLINK
/
SQL> @dblink
DB_LINK OWNID LOGON HETER PROTOCOL OPN_CUR TXN UPDATE C_P_S
-----
INST2.ACME.COM 0 YES YES UNKN 0 YES YES 255
```

29.6 创建位置透明性

当已经配置了所需的数据库链接之后，就能使用各种工具来对用户隐藏数据库系统的分布式特性。换句话说，用户能访问远程对象就好像这些对象是本地对象一样。本节说明如何对用户隐藏分布式功能：

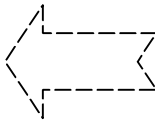
- 使用视图创建位置透明性
- 使用同义词创建位置透明性
- 使用过程创建位置透明性

29.6.1 使用视图创建位置透明性

本地视图能为分布式数据库系统中的本地和远程表提供位置透明性。

例如：假设表 emp 存储在本地数据库中、表 dept 存储在远程数据库中。为了使这些表对系统中的用户透明，可以在本地数据库中创建视图，该视图联结本地和远程数据：

```
CREATE VIEW company
AS
SELECT a.empno, a.ename, b.dname
FROM scott.emp a, jward.dept@hq.acme.com b
```



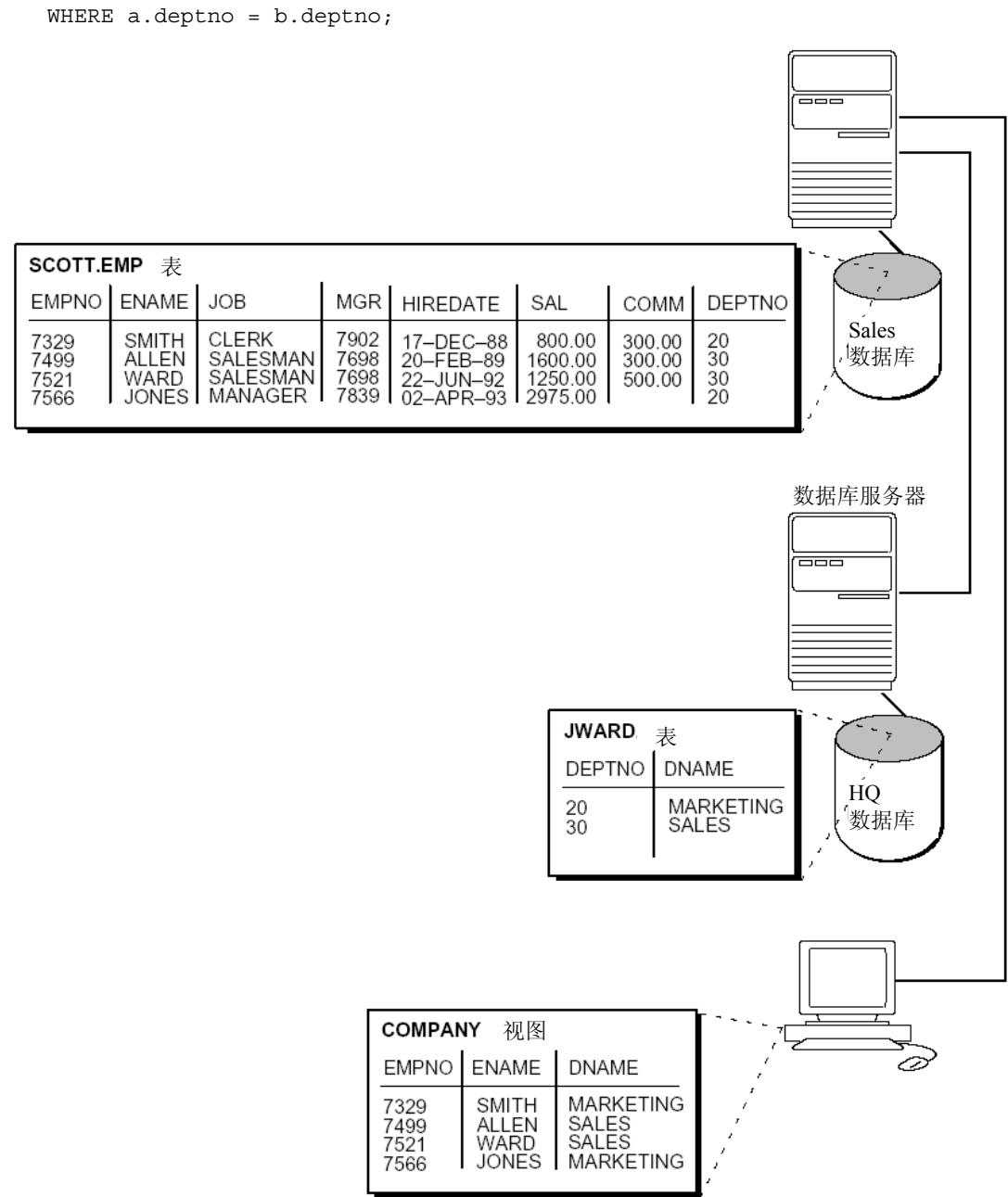
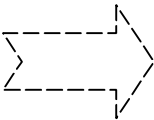


图 29-3 视图和位置的透明性

当用户访问该视图时，它们并不需要知道存储数据的物理位置，或者是否访问来自于几个表的数据。所以，对于用户来说，比较容易获得所需要的信息。例如，下面的查询提供了来自于本地和远程数据库表两者的数据：

```
SELECT * FROM company;
```

本地视图的拥有者仅能授予由远程用户所授权的在本地视图上的那些对象权限。（远程



用户是由数据库链接类型来暗指的), 这相似于参照本地数据的视图的权限管理。

29.6.2 使用同义词创建位置透明性

在分布式和非分布式两种环境中, 同义词都是很有用的, 因为同义词隐藏了下层对象的标识, 也包括它在分布式数据库系统中的位置。如果必须重新命名或移动下层对象, 仅需要重新定义同义词; 基于同义词的应用仍然正常运行。在分布式数据库系统中, 同义词也为用户简化了 SQL 语句。

29.6.2.1 创建同义词

能为下面的对象创建同义词:

- 表
- 类型
- 视图
- 物化视图
- 序列
- 过程
- 函数
- 包

任何同义词都是模式对象, 它们被存储在创建它们的数据库的数据字典中。为了通过数据库链接来简化对远程表的访问, 同义词能实现用单个字访问远程数据, 即对同义词的用户隐藏了特殊的对象名称和位置。

创建同义词的语法为:

```
CREATE [PUBLIC] synonym_name
FOR [schema.]object_name[@database_link_name]
```

其中:

PUBLIC 是关键字, 它指出所有用户都可以使用该同义词。省略该参数就使得该同义词为私有, 并且仅由创建者使用。公共同义词只能由具有 **CREATE PUBLIC SYNONYM** 系统权限的用户所创建。

synonym_name 指定被用户和应用所参照的替代对象名称。

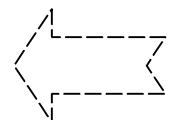
schema 指定在 *object_name* 中指定的对象的模式。省略此参数, 就用创建者的模式作为该对象的模式。

object_name 按实际情况, 指定一个表、视图、序列、物化视图、类型、过程、函数或包。

database_link_ 指定标识在 *object_name* 中指定的对象所在的远程数据库和模式的数。

name 数据库链接。

同义词必须是其模式中的唯一的命名的对象。如果一个模式包含了同名的模式对象和公共同义词, 那么当拥有该模式的用户参照那个名称时, Oracle 总是找到该模式对象。



例：创建公共同义词

假设在分布式数据库系统中的每个数据库中，为存储在 `hq` 数据库中 `scott.emp` 表定义一个公共同义词：

```
CREATE PUBLIC SYNONYM emp FOR scott.emp@hq.acme.com;
```

可以设计一个雇员管理应用，而不关心什么地方使用该应用，因为公共同义词隐藏了表 `scott.emp@hq.acme.com` 的位置。该应用中的 SQL 语句通过参照公共同义词 `emp` 来访问表。

此外，如果将 `emp` 表从 `hq` 数据库移动到 `hr` 数据库，那么仅需修改系统节点上的公共同义词。雇员管理应用在所有节点上仍然正常运行。

29.6.2.2 管理权限和同义词

同义词是对实际对象的一个参考。一个使用针对特殊的模式对象的同义词的用户，其本身还必须在下层模式对象上具有权限。例如，如果用户打算使用同义词，但是在同义词所标识的表上没有权限，就会产生一个表示表或视图不存在的错误。

假设 `scott` 为远程对象 `scott.emp@sales.acme.com` 创建了本地同义词 `emp` 当作别名。`scott` 不能将同义词上的对象权限授予其他本地用户。`scott` 不能给同义词授予本地权限是因为这种操作等于给 `sales` 数据库中的远程 `emp` 表授予权限，这是不允许的。这种行为不同于对用作本地表或视图的别名的同义词的权限管理。

因此，当同义词被用于位置透明性时，就不能管理本地权限。完全在远程节点上控制基础对象的安全。例如，用户 `admin` 不能给 `EMP_SYN` 同义词授予对象权限。

与在视图或过程定义中所参照的数据库链接不同，在同义词中所参照的数据库链接，在分析对同义词的参照时，是通过首先查找该模式所拥有的私有链接而被解析的。所以，为了确保所希望的对象解析，在同义词的定义中指定下层对象的模式是非常重要的。

29.6.3 使用过程创建位置透明性

被称为过程的 PL/SQL 程序单元能提供位置透明性。可做如下选择：

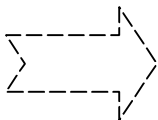
- 使用本地过程参照远程数据
- 使用本地过程调用远程过程
- 使用本地同义词参照远程过程

29.6.3.1 使用本地过程参照远程数据

过程或函数（或是独立的或是在包中的）可以包含参照远程数据的 SQL 语句。例如，考虑由下面的语句所创建的过程：

```
CREATE PROCEDURE fire_emp (enum NUMBER) AS
BEGIN
DELETE FROM emp@hq.acme.com
WHERE empno = enum;
END;
```

当用户或应用调用 `fire_emp` 过程时，感觉不到正在修改远程表。



当过程中的语句使用本地过程、视图或同义词间接地参照远程数据时，可能运用位置的第二层透明性。例如，下面的语句定义了一个本地同义词：

```
CREATE SYNONYM emp FOR emp@hq.acme.com;
```

给出该同义词，就能使用下面的语句创建 `fire_emp` 过程：

```
CREATE PROCEDURE fire_emp (enum NUMBER) AS
BEGIN
DELETE FROM emp WHERE empno = enum;
END;
```

如果重新命名或移动表 `emp@hq`，那么仅需要修改参照该表的本地同义词，而不需要修改任何调用该过程的过程和应用。

29.6.3.2 使用本地过程调用远程过程

可以使用本地过程来调用远程过程。之后远程过程就能执行所需的 DML。例如，假设 `scott` 连接到 `local_db`，并创建下面的过程：

```
CONNECT scott/tiger@local_db
CREATE PROCEDURE fire_emp (enum NUMBER)
AS
BEGIN
EXECUTE term_emp@hq.acme.com;
END;
```

现在，假设 `scott` 连接到远程数据库，并创建远程过程：

```
CONNECT scott/tiger@hq.acme.com
CREATE PROCEDURE term_emp (enum NUMBER)
AS
BEGIN
DELETE FROM emp WHERE empno = enum;
END;
```

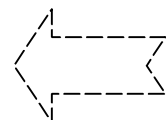
当被连接到 `local_db` 的用户或应用调用 `fire_emp` 过程时，该过程转而调用 `hq.acme.com` 中的远程 `term_emp` 过程。

29.6.3.3 使用本地同义词参照远程过程

例如，`scott` 连接到本地 `sales.acme.com` 数据库上，并创建下面的过程：

```
CREATE PROCEDURE fire_emp (enum NUMBER) AS
BEGIN
DELETE FROM emp@hq.acme.com
WHERE empno = enum;
END;
```

然后用户 `peggy` 连接到 `supply.acme.com` 数据库上，并且为 `scott` 在远程 `sales` 数据库上



创建的过程创建下面的同义词：

```
SQL> CONNECT peggy/hill@supply
```

```
SQL> CREATE PUBLIC SYNONYM emp FOR scott.fire_emp@sales.acme.com;
```

那么，Supply 中的本地用户就能用该同义词来执行 sales 中的过程。

29.6.3.4 管理过程和权限

假设本地过程包含了一条参照远程表或视图的语句。本地过程的拥有者可以给任何用户授予 execute 权限，从而，使那个用户具有了执行过程和间接地访问远程数据的能力。

一般来说，过程有助于安全。在过程中所参照的对象权限不必明确地授予给调用用户。

29.7 管理语句透明性

Oracle 允许下面的标准 DML 语句参照远程表：

- SELECT (查询)
- INSERT
- UPDATE
- DELETE
- SELECT A FOR UPDATE (在异构系统中不总是支持此语句)
- LOCK TABLE

包含联结、合计、子查询和 SELECT A FOR UPDATE 的查询，可以参照任何数量的本地和远程表及视图。例如，下面的查询联结了两个远程表中的信息：

```
SELECT e.empno, e.ename, d.dname
FROM scott.emp@sales.division3.acme.com e, jward.dept@hq.acme.com d
WHERE e.deptno = d.deptno;
```

UPDATE, INSERT, DELETE 和 LOCK TABLE 语句可以参照本地和远程表两者。不需要用程序来更新远程表。例如，下面的语句，通过从本地数据库的 jward 模式中的 emp 表中选择行，并将其插入到 scoot.sales 模式的远程表 emp 中作为新行：

```
INSERT INTO scott.emp@sales.division3.acme.com
SELECT * FROM jward.emp;
```

限制：

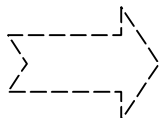
应用语句透明性的几个限制如下：

- 在单个 SQL 语句中，所有被参照的 LONG 和 LONG RAW 列、序列、被更新的表和被锁定的表，都必须位于同一节点上。
- 在同构系统中，除了通过远程执行 DBMS_SQL 包过程之外，Oracle 不允许执行远程 DDL 语句（例如，CREATE、ALTER 和 DROP），如下例所示：

```
DBMS_SQL.PARSE@link_name(crs, 'drop table emp', v7);
```

注意在异构系统中，通过工具允许执行 DDL。

- ANALYZE 语句的 LIST CHAINED ROWS 子句不能参照远程表。
- 在分布式数据库系统中，Oracle 总是计算与环境有关的 SQL 函数，如与本地服务



器有关的 SYSDATE, USER, UID 和 USERENV, 不管在什么地方执行该语句 (或语句的一部分)。

注意: Oracle 只对查询支持 USERENV 函数。

- 与远程对象访问相关的各种性能限制
- 远程视图没有统计数据。
- 不能优化分区表中的查询。
- 对远程表不能创建 20 个以上的索引。
- 对组合索引不能使用 20 个以上的列。
- 在 Oracle 的分布式读一致性中有一个局限, 它会引起一个节点相对于另一个来说是过时的。根据读一致性, 一个查询可能会结束检索的一致性, 而查询出过期的数据。参见第 32 章的“管理读一致性”中是如何管理这个问题的。

29.8 管理分布式数据库: 方案

本节给出了各种包含数据库链接管理语句的例子:

- 创建公共固定用户数据库链接
- 创建公共固定用户共享数据库链接
- 创建公共已连接用户数据库链接
- 创建公共已连接用户共享数据库链接
- 创建公共当前用户数据库链接

29.8.1 创建公共固定用户数据库链接

下面的例子用 jane 连接到本地数据库, 并且为 scott 创建一个到数据库 sales 的公共固定用户数据库链接。通过数据库的网络服务名 sldb 访问该数据库:

```
CONNECT jane/does@local
CREATE PUBLIC DATABASE LINK sales.division3.acme.com
CONNECT TO scott IDENTIFIED BY tiger
USING 'sldb';
```

结果:

连接到本地数据库的任何用户, 都能使用 sales.division3.acme.com 数据库链接, 连接到远程数据库。每个用户连接到远程数据库的模式 scott。

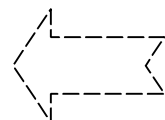
为了访问在 scott 的远程模式中的 emp 表, 用户可以发布下面的 SQL 查询:

```
SELECT * FROM emp@sales.division3.acme.com;
```

注意每个应用或用户会话都创建了一个到服务器上的公共帐号的单独连接。在应用或用户会话的持续时间内, 到远程数据库的连接保持打开。

29.8.2 创建公共固定用户共享数据库链接

下面的例子用 dana 连接到本地数据库, 并且创建一个到 sales 数据库的公共链接 (使用数据库的网络服务名 sldb)。该链接允许用 scott 连接到远程数据库, 并且用 scott 验证该



用户:

```
CONNECT dana/sculley@local
CREATE SHARED PUBLIC DATABASE LINK sales.division3.acme.com
CONNECT TO scott IDENTIFIED BY tiger
AUTHENTICATED BY scott IDENTIFIED BY tiger
USING 'sldb';
```

结果:

连接到本地共享服务器的任何用户都能使用该数据库链接, 通过一个共享服务器进程, 连接到远程 sales 数据库。然后用户能查询在 scott 模式中的表。

在上面的例子中, 每个本地共享服务器都能建立一个到远程服务器的连接。无论什么时候, 本地共享服务器进程需要通过 sales.division3.acme.com 数据库链接来访问远程服务器时, 该本地共享服务器进程就重新使用已建立的网络连接。

29.8.3 创建公共已连接用户数据库链接

下面的例子用 larry 连接到本地数据库, 并且创建一个到网络服务名为 sldb 的数据库的公共链接:

```
CONNECT larry/oracle@local
CREATE PUBLIC DATABASE LINK redwood
USING 'sldb';
```

结果:

连接到本地数据库的任何用户都能使用 redwood 数据库链接。本地数据库中用该数据库链接连接的用户来确定远程模式。

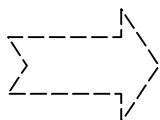
如果 scott 是一个已连接用户, 并且使用数据库链接, 那么该数据库链接就连接到远程模式 scott。如果 fox 是一个已连接用户, 并且使用数据库链接, 那么该数据库链接就连接到远程模式 fox。

当远程模式 fox 不能解析 emp 模式对象时, 对于本地数据库中的本地用户 fox, 下面的语句失败。也就是说, 如果 sales.division3.acme.com 中的 fox 模式, 没有用 emp 作为表、视图或(公共)同义词, 就将返回一个错误。

```
CONNECT fox/mulder@local
SELECT * FROM emp@redwood;
```

29.8.4 创建公共已连接用户共享数据库链接

下面的例子用 neil 连接到本地数据库, 并且创建一个到 sales 数据库(使用其网络服务名 sldb)的共享的、公共的链接。此用户是用 crazy/horse 用户标识/口令来验证的。下面的语句创建了公共的、已连接用户的、共享的数据库链接:



```
CONNECT neil/young@local
CREATE SHARED PUBLIC DATABASE LINK sales.division3.acme.com
AUTHENTICATED BY crazy IDENTIFIED BY horse
USING 'sldb';
```

结果:

连接到本地服务器的每个用户都能使用该共享数据库链接, 连接到远程数据库, 并在相应的远程模式中查询表。

每个本地的、共享的服务器进程都建立一个到远程服务器的连接。无论什么时候, 一个本地服务器过程需要通过 `sales.division3.acme.com` 数据库链接来访问远程服务器时, 该本地进程就重新使用已建立的网络连接, 甚至于在被连接的用户是不同的用户时也如此。

如果频繁地使用该数据库链接, 最终, 本地数据库中的每个共享服务器都会有一个远程连接。这就是说, 即使是新的用户使用该共享数据库链接, 也不需要更多的到远程服务器的物理连接。

29.8.5 创建公共当前用户数据库链接

下面的例子用已连接用户连接到本地数据库, 并且创建一个到 `sales` 数据库 (使用其网络服务名 `sldb`) 的公共链接。下面的语句创建公共当前用户数据库链接:

```
CONNECT bart/simpson@local
CREATE PUBLIC DATABASE LINK sales.division3.acme.com
CONNECT TO CURRENT_USER
USING 'sldb';
```

注意: 为了使用该链接, 当前的用户必须是全局用户。

结果:

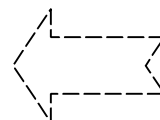
假设 `scott` 创建本地过程 `fire_emp`, 该过程从远程 `emp` 表中删除一行, 并且给 `ford` 授予在过程 `fire_emp` 上的执行权限。

```
CONNECT scott/tiger@local_db
CREATE PROCEDURE fire_emp (enum NUMBER)
AS
BEGIN
DELETE FROM emp@sales.division3.acme.com
WHERE empno=enum;
END;
GRANT EXECUTE ON fire_emp TO ford;
```

现在, 假设 `ford` 连接到本地数据库, 并且运行 `scott` 的过程:

```
CONNECT ford/fairlane@local_db
EXECUTE PROCEDURE scott.fire_emp (enum 10345);
```

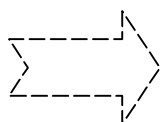
当 `ford` 执行过程 `scott.fire_emp` 时, 该过程在 `scott` 的权限之下运行。因为使用当前用户数据库链接, 所以所建立的连接是针对 `scott` 的远程模式的, 而不是 `ford` 的远程模式。注



意 scott 必须是全局用户，而 ford 不必一定是全局用户。

注意：如果使用已连接用户数据库链接来替换，则该连接将是针对 ford 的远程模式的。

通过使用到 scott 的远程模式的固定用户数据库链接，也能实现同样的结果。然而，用固定用户数据库链接时，可能损害安全问题，因为可以用可读的格式使用数据库中的 scott 的用户名称和口令。



第 30 章 开发用于分布式数据库系统的应用

本章描述了在开发运行在分布式数据库系统中的应用时要考虑的重点问题。包括如下内容：

- 管理应用数据的分布
- 控制靠数据库链接建立起来的连接
- 维护分布式系统中的参照完整性约束
- 协调分布式查询
- 处理远程过程中的错误

30.1 管理应用数据的分布

在一个分布式数据库环境中，要与数据库管理员相协调，以决定存放数据的最佳位置。要考虑的某些问题是：

- 从每个位置发出的事务的数量
- 每个节点所使用的数据（表的部分）的数量
- 网络的性能特征和可靠性
- 各个节点的速度、磁盘的容量
- 当一个节点或链接不可用时的重要性
- 对表之间的参照完整性的需求

30.2 控制靠数据库链接建立起来的连接

当在一条 SQL 语句或远程过程调用中参照了一个全局对象名时，数据库链接就按本地用户建立起一个到远程数据库中的会话的连接。只有在这之前还没有为该本地用户会话建立连接时，才会创建该远程连接和会话。

在本地用户的会话期间，已经建立的到远程数据库的连接和会话，会一直保持着，除非应用或用户明确地终止它们。注意，如果通过数据库链接发布一条 SELECT 语句，就会在回滚段放置一个事务锁。要想重新释放该段，就必须发布一条 COMMIT 或 ROLLBACK 语句。

终止用数据库链接建立起来的远程连接，有益于断开应用不再需要的高开销的连接。可以用带有 CLOSE DATABASE LINK 子句的 ALTER SESSION 语句，来终止远程连接和会话。例如，假设发布了如下事务：

```
SELECT * FROM emp@sales;  
COMMIT;
```

如下语句终止由 sales 数据库链接所指向的远程数据库中的会话：

```
ALTER SESSION CLOSE DATABASE LINK sales;
```

要想关闭用户会话中的数据库链接连接，必须要具有 ALTER SESSION 系统权限。

注意：在关闭数据库链接之前，首先关闭使用该链接的所有的游标，然后终止使用该链接的当前的事务。

30.3 维护分布式系统中的参照完整性约束

如果一条分布式语句中的一部分失败了（如违背了完整性约束），Oracle 就会返回错误号 ORA-02055。以后的语句或过程调用就会返回错误号 ORA-02067，直到发布一条回滚或回滚到保存点的语句为止。

要在应用中设计出能检查任何表示分布式更新的一部分失败的返回信息的功能。如果探测到一个失败，就应该在允许应用继续运行之前回滚整个事务。

Oracle 不允许在分布式系统的各个节点之间，定义公布的参照完整性约束。换句话说，在一个表上公布的参照完整性约束，不能指定一个参照远程表的主键或唯一键的外键。然而，可以用触发器来维护各节点的父/子表之间的关系。

如果打算用触发器来定义分布式数据库的各节点之间的参照完整性，要注意，网络故障可能不仅会限制对父表，还会限制对子表的访问可能性。例如，假设子表在 sales 数据库中，父表在 hq 数据库中。如果在这两个数据库之间的网络连接失败了，则某些针对子表的 DML 语句（如在子表中插入行或更新子表中的外键值），会由于参照完整性触发器必须要访问在 hq 数据库中的父表而不能继续执行。

30.4 协调分布式查询

本地 Oracle 数据库服务器将分布式查询分成相应个数的远程查询，并将其发送到远程节点中去执行。远程节点执行查询并将结果返回给本地节点。然后本地节点执行任何必要后续处理并将结果返回给用户或应用。

为了优化查询处理，可以用几种方法来设计应用。本节包括如下内容：

- 使用并行行中视图
- 使用基于开销的优化策略
- 使用暗示
- 分析执行方案

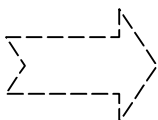
30.4.1 使用并行行中视图

优化分布式查询的最有效的方法，是尽可能少地访问远程数据库并只检索所需的数据。

例如，假设在一个分布式查询中参照了两个不同远程数据库中的五个远程表，并有一个复杂的过滤条件（如，WHERE r1.salary + r2.salary > 50000）。可以通过重写该查询，使得只访问一次远程数据库，和在远程站点上运用过滤条件，来改善该查询的性能。通过重写该查询，使得更少的数据被传送到查询的执行站点。

可以用并行行中视图来实现重写该查询，以便只访一次远程数据库。需要定义如下术语：

Collocated 位于相同数据库中的两个或多个表。



Inline view 一个在父 SELECT 语句中用于替代一个表的 SELECT 语句。嵌入的 SELECT 语句，在括号中显示的语句，就是行中视图的例子：

```
SELECT e.empno, e.ename, d.deptno, d.name
FROM (SELECT empno, ename from
      emp@orc1.world) e, dept d;
```

Collocated inline view 仅从单个数据库中的多个表中选取数据的一个行中视图。它减少了访问远程数据库的时间，改善了分布式查询的性能。

Oracle 公司建议，用并行行中视图来形成分布式查询，以便提高分布式查询的性能。Oracle 的基于开销的优化策略，可以透明地重写大多数分布式查询，以便获得由并行行中视图所提供的性能。

30.4.2 使用基于开销的优化策略

除了用并行行中视图重写查询之外，基于开销的优化方法根据收集到的被参照表的统计数据，和优化器所执行的计算，来优化分布式查询。

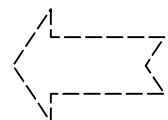
例如，基于开销的优化策略分析如下查询。该例子假设该表的统计数据是可用的。注意，它分析在 CREATE TABLE 语句中的查询：

```
CREATE TABLE AS (
SELECT l.a, l.b, r1.c, r1.d, r1.e, r2.b, r2.c
FROM local l, remotel r1, remote2 r2
WHERE l.c = r.c
AND r1.c = r2.c
AND r.e > 300
);
```

并将其重写为：

```
CREATE TABLE AS (
SELECT l.a, l.b, v.c, v.d, v.e
FROM (
SELECT r1.c, r1.d, r1.e, r2.b, r2.c
FROM remotel r1, remote2 r2
WHERE r1.c = r2.c
AND r1.e > 300
) v, local l
WHERE l.c = r1.c
);
```

给该行中视图分配了一个别名 v，然后就可以在上述 SELECT 语句中象表一样参照该别名。创建并行行中视图减少了在远程站点中执行查询的数量。因此减少了网络传输的开销。



30.4.2.1 基于开销的优化策略是如何工作的

优化器的主要任务是，重写分布式查询以便使用并行行中视图。分三步执行该优化策略：

- ①融合所有可融合的视图。
- ②优化器执行并行查询数据块测试。
- ③优化器用并行行中视图重写查询。

重写查询之后，执行该查询并将数据集返回给用户。

因为基于开销的优化策略的执行是对用户透明的，所以不能改善多个分布式查询方案的性能。特别地，如果分布式查询包含如下之一，则基于开销的优化策略不起作用：

- 合计
- 子查询
- 复杂 SQL

如果分布式查询包含上述之一，参见第 30 章的“使用暗示”，学习如何修改查询和使用暗示来改善分布式查询的性能。

30.4.2.2 设置基于开销的优化策略

当为了改善分布式查询的性能，而将系统设置成使用基于开销的优化策略之后，该操作对该用户就是透明的了。换句话说，当发布查询时，自动进行优化。

为了将系统设置成使用 Oracle 的优化器，需要完成如下任务：

- 设置环境
- 分析表

设置环境 为了启用基于开销的优化策略，需要将 OPTIMIZER_MODE 初始化参数设置成 CHOOSE 或 COST。为了设置该参数，可以：

- 在初始化参数文件中修改 OPTIMIZER_MODE 参数
- 发布一条 ALTER SESSION 语句，在会话层设置该参数

为了在会话层设置 OPTIMIZER_MODE 初始化参数，需要发布如下语句之一：

```
ALTER SESSION OPTIMIZER_MODE = CHOOSE;
```

```
ALTER SESSION OPTIMIZER_MODE = COST;
```

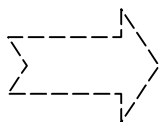
分析表 为了使基于开销的优化策略能为分布式查询选择最有效的路径，就必须提供所包含的表的精确的统计数据。

为表产生统计数据的一种方法是执行 ANALYZE 语句。注意，当执行该语句时，Oracle 会将被分析的表加锁。例如，如果在分布式查询中参照了 emp 和 dept 表，就执行如下语句，来产生所需的统计数据：

```
ANALYZE TABLE emp COMPUTE STATISTICS;
```

```
ANALYZE TABLE dept COMPUTE STATISTICS;
```

统计数据被存储在如下位置：



统计的类型	表
表	DBA/ALL/USER_TABLES
列	DBA/ALL/USER_TAB_COL_STATISTICS
	DBA/ALL/USER_TAB_COLUMNS
直方图	DBA/ALL/USER_TAB_HISTOGRAMS
	DBA/ALL/USER_PART_HISTOGRAMS
	DBA/ALL/USER_SUBPART_HISTOGRAMS
用户定义的统计数据	DBA/ALL/USER_USTATS

注意：为了执行 ANALYZE 语句，必须在本地连接到相关的表。不能执行如下语句
ANALYZE TABLE remote@remote.com COMPUTE STATISTICS;
必须先连接到远程站点，然后才能执行上面的 ANALYZE 语句。
还可以用 DBMS_STATS 包来采集统计数据。下面的过程启用采集某种优化器的统计数据，可能比 ANALYZE 语句的性能好些：

- GATHER_INDEX_STATS
- GATHER_TABLE_STATS
- GATHER_SCHEMA_STATS
- GATHER_DATABASE_STATS

例如，假设分布式事务照样访问 scott.dept 表。为了保证基于开销的优化器仍然挑选出最佳方案，执行如下过程：

```
BEGIN
DBMS_STATS.GATHER_TABLE_STATS ('scott', 'dept');
END;
```

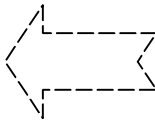
30.4.3 使用暗示

如果语句未被充分地优化，那么可以用暗示来扩展基于开销的优化策略的能力。特别地，如果编写你自己的查询以便使用并行行中视图，可以指示基于开销的优化器不要重写你的分布式查询。

另外，如果具有有关数据库环境的特殊的知识（如统计数据、负载、网络 and CPU 限制、分布式查询等等），就可以指定暗示以便指导基于开销的优化策略。例如，如果使用采集到的基于数据库环境的并行行中视图，编写了你自己的已经优化了的查询，就可以指定 NO_MERGE 暗示，以防止优化器再重写你的查询。

如果分布式查询中包含合计、子查询或复杂 SQL 时，这个技术特别有帮助。因为优化器不能重写这种类型的分布式查询，指定 NO_MERGE 就使优化器跳过了在第 30 章的“基于开销的优化策略是如何工作的”中所描述的步骤。

DRIVING_SITE 暗示用于定义作为查询执行站点的远程站点。这样，就在该远程站点执行该查询，然后将数据返回给本地站点。当该远程站点包含了大多数数据时，这个暗示特别有帮助。



30.4.3.1 使用 NO_MERGE 暗示

NO_MERGE 暗示能防止 Oracle 将行中视图融合到一个潜在的非并列 SQL 语句中（参见第 30 章的“使用暗示”）。该暗示被嵌入在 SELECT 语句中，或者作为一个参数出现在带有行中视图的 SELECT 语句的开头，或者出现在定义该行中视图的查询块中。

```
/* with argument */
SELECT /*+NO_MERGE(v)*/ t1.x, v.avg_y
FROM t1, (SELECT x, AVG(y) AS avg_y FROM t2 GROUP BY x) v,
WHERE t1.x = v.x AND t1.y = 1;
/* in query block */
SELECT t1.x, v.avg_y
FROM t1, (SELECT /*+NO_MERGE*/ x, AVG(y) AS avg_y FROM t2 GROUP BY x) v,
WHERE t1.x = v.x AND t1.y = 1;
```

典型地，当你基于你的数据库环境的知识，已经开发出了经过优化的查询后，就使用这个暗示。

30.4.3.2 使用 DRIVING_SITE 暗示

DRIVING_SITE 暗示用于指定执行查询的站点。最好是让基于开销的优化器策略来决定应该在何处执行查询，但是如果覆盖优化器，就可以手动指定执行站点。

如下是一条带有 DRIVING_SITE 暗示的 SELECT 语句的例子：

```
SELECT /*+DRIVING_SITE(dept)*/ * FROM emp, dept@remote.com
WHERE emp.deptno = dept.deptno;
```

30.4.4 分析执行方案

协调分布式查询的一个重要方面是分析执行方案。从分析中得到的反馈是用于测试和确认数据库的一个重要成份。当打算比较方案时，确认就变得特别重要了。例如，将基于开销的优化策略所优化的分布式查询执行方案，和使用暗示、并行行中视图、其他技术的手动优化查询的执行方案相比较时。

30.4.4.1 准备数据库以便存储方案

在能够查看分布式查询的执行方案之前，要先准备数据库，以便存储执行方案。可以用执行脚本的方法来完成该准备工作。执行如下脚本，来准备数据库，以便存储执行方案：

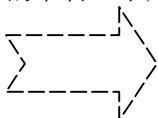
```
SQL> @UTLXPLAN.SQL
```

注意：可以在 \$ORACLE_HOME/rdbms/admin 目录中找到该 utlxplan.sql 文件。

执行完 utlxplan.sql 之后，就在当前的模式中创建表 PLAN_TABLE，以便临时地存储该执行方案。

30.4.4.2 产生执行方案

在已经准备好了用于存储执行方案的数据库之后，就已经做好了查看指定查询的方案



中。例如，可以执行如下语句：

```
EXPLAIN PLAN FOR
SELECT d.dname
FROM dept d
WHERE d.deptno
IN (SELECT deptno
FROM emp@orc2.world
GROUP BY deptno
HAVING COUNT (deptno) >3
)
/
```

30.4.4.3 查看执行方案

执行完上述 SQL 语句之后，执行方案就被临时地存储在早先创建的 PLAN_TABLE 中了。若要查看执行方案的结果，执行如下脚本：

```
@UTLXPLS.SQL
```

注意：可以在\$ORACLE_HOME/rdbms/admin 目录中找到该 utlxplx.sql 文件。
执行 utlxplx.sql 脚本就会显示指定的 SELECT 语句的执行方案。其结果被格式化成：

Operation	Name	Rows	Bytes	Cost	Pstart	Pstop
SELECT STATEMENT						
NESTED LOOPS						
VIEW						
REMOTE						
TABLE ACCESS BY INDEX ROWID	DEPT					
INDEX UNIQUE SCAN	PK_DEPT					

如果用编写自己的并行行中视图或用暗示的方法，来手动地优化分布式查询，最好在手动优化之前和之后生成执行方案。有了两个执行方案之后，就可以比较手动优化的效率，并做适当的修改，以便改善分布式查询的性能。

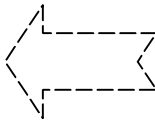
若要查看将在远程站点上执行的 SQL 语句，就执行如下 SELECT 语句：

```
SELECT OTHER
FROM PLAN_TABLE
WHERE operation = 'REMOTE';
```

如下是一个样例输出：

```
SELECT DISTINCT "A1"."DEPTNO" FROM "EMP" "A1"
GROUP BY "A1"."DEPTNO" HAVING COUNT("A1"."DEPTNO")>3
```

注意：如果很难看全 OTHER 列的内容，就执行如下 SQL*Plus 命令：
SET LONG 9999999



30.5 管理远程过程中的错误

当 Oracle 在本地或远程位置中执行过程时，可能发生四种类型的例外：

- PL/SQL 用户定义的例外，必须用关键词 EXCEPTION 进行发布。
- PL/SQL 预定义的例外，如 NO_DATA_FOUND 关键词
- SQL 错误，如 ORA-00900 和 ORA-02015
- 用 RAISE_APPLICATION_ERROR () 过程产生的应用例外。

当使用本地过程时，可以写一个如下的例外管理器，来捕获这些信息：

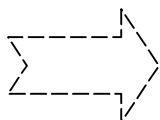
```
BEGIN
...
EXCEPTION
WHEN ZERO_DIVIDE THEN
/* ... handle the exception */
END;
```

注意，WHEN 子句需要一个例外名。如果例外没有名字，如 RAISE_APPLICATION_ERROR 产生的例外，就可以用 PRAGMA_EXCEPTION_INIT 分配一个名字，如：

```
DECLARE
null_salary EXCEPTION;
PRAGMA EXCEPTION_INIT(null_salary, -20101);
BEGIN
...
RAISE_APPLICATION_ERROR(-20101, 'salary is missing');
...
EXCEPTION
WHEN null_salary THEN
...
END;
```

调用远程过程时，可以用一个本地过程中的例外管理器来管理例外。远程过程必须将错误号返回给本地，即调用过程，然后本地就像前面的例子一样来管理例外。注意，PL/SQL 用户定义的例外总是将 ORA-06510 返回给本地过程。

因此，不可能基于错误号来区分两个不同的用户定义的例外。所有其他的远程例外可以像本地过程一样来管理。



第 31 章 分布式事务的概念

本章描述什么是分布式事务和 Oracle 如何管理他们的完整性。本章包括如下内容：

- 什么是分布式事务
- 用于分布式事务的会话树
- 两阶段提交机制
- 悬而未决事务
- 分布式事务处理：案例学习

31.1 什么是分布式事务

分布式事务包含一条或多条语句，单独的或作为一组，这些语句更新在分布式数据库的两个或多个不同的节点上的数据。例如，假设数据库配置如下图 31-1 所示：

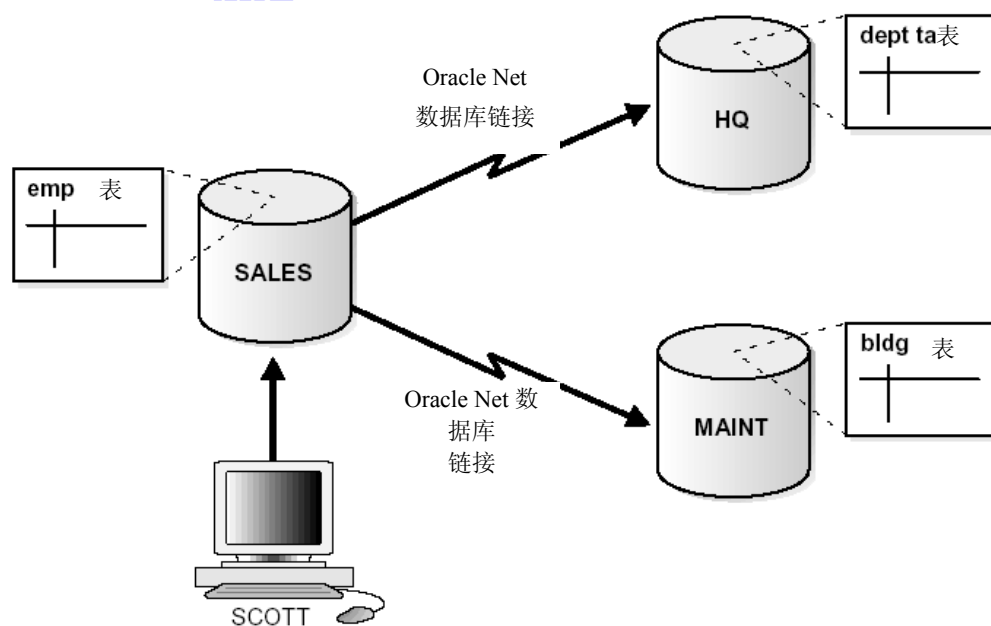


图 31-1 分布式系统

scott 执行的如下分布式事务，更新本地 sales 数据库、远程 hq 数据库和远程 maint 数据库：

```
UPDATE scott.dept@hq.us.acme.com
SET loc = 'REDWOOD SHORES'
WHERE deptno = 10;
UPDATE scott.emp
SET deptno = 11
WHERE deptno = 10;
```

```
UPDATE scott.bldg@maint.us.acme.com  
SET room = 1225  
WHERE room = 1163;  
COMMIT;
```

注意：如果一个事务的所有语句仅参照一个远程节点，则该事务是远程的，而不是分布式的。

在分布式事务中存在两种允许的操作：

- DML 和 DDL 事务
- 事务控制语句

31.1.1 DML 和 DDL 事务

如下列表描述了在分布式事务中所支持的 DML 和 DDL 操作：

- CREATE TABLE AS SELECT
- DELETE
- INSERT（默认和直接装载）
- LOCK TABLE
- SELECT
- SELECT FOR UPDATE

可以并行执行 DML 和 DDL 语句，串行执行 INSERT 直接装载语句，但要注意如下限制：

- 所有远程操作必须是 SELECT 语句
- 这些语句不能是另外的分布式事务中的子句
- 如果在 INSERT，UPDATE 或 DELETE 语句的 `table_expression_clause` 中参照的表是远程的，则该执行是串行的而不是并行的。
- 发布并行 DML/DDL 或直接装载 INSERT 语句之后，不能执行远程操作。
- 如果事务是用 XA 或 OCI 开始的，则串行执行该事务。
- 不能在发起并行操作的事务上执行循环操作。例如，不能参照一个实际上是本地对象的同义词的远程对象。
- 如果在事务中执行不是 SELECT 语句的分布式操作，就不会并行执行任何 DML。

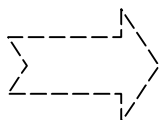
31.1.2 事务控制语句

如下列表描述了被支持的事务控制语句：

- COMMIT
- ROLLBACK
- SAVEPOINT

31.2 分布式事务的会话树

当发布分布式事务中的语句时，Oracle 就会定义一个参与到该事务中的所有节点的会



话树。会话树是一个层次模式，它描述各个会话之间的关系和它们的角色。图 31-2 说明了一个会话树：

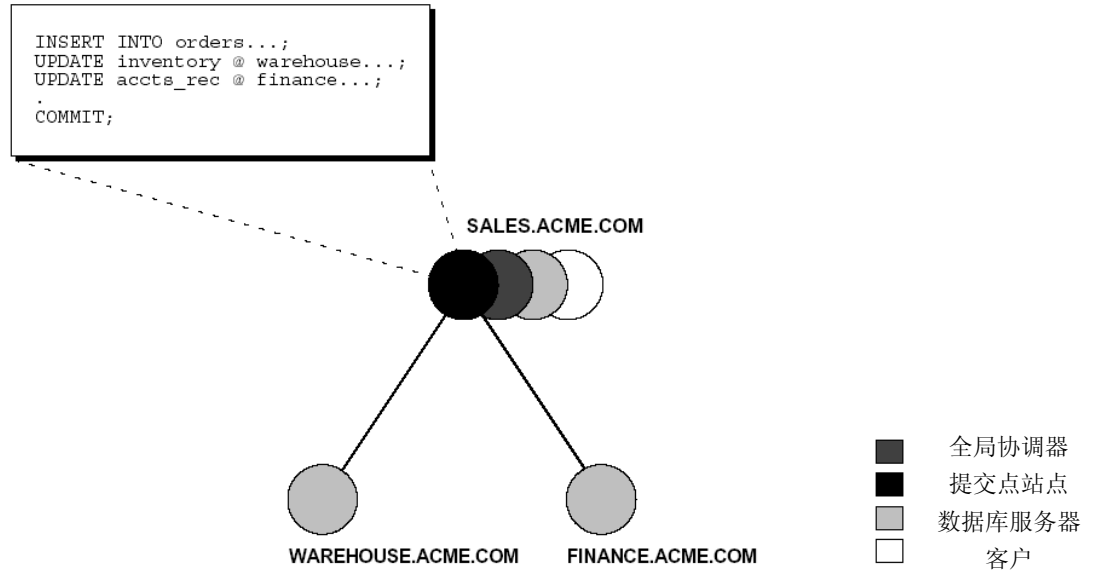


图 31-2 会话树的例子

所有参与到分布式事务的会话树中的节点，都会扮演如下的一个或多个角色：

角色	说明
Client（客户）	参照属于一个不同节点的数据库中的信息的节点
Database server（数据库服务器）	接收来自另一个节点的信息需求的节点
Global coordinator（全局协调器）	发起分布式事务的节点
Local coordinator（本地协调器）	被强制参照其他节点上的数据，以便完成事务中它那部分工作的节点
Commit point site（提交点站点）	按照全局协调器的指示，提交或回滚事务的节点

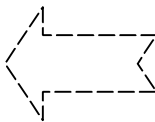
- 一个节点会在分布式事务中扮演什么角色取决于：
- 该事务是本地的还是远程的
 - 该节点的 commit point strength（提交点强度）（参见第 31 章的“提交点站点”）
 - 是否可以在一个节点取得所有所需的数据，还是需要参照其他节点来完成该事务
 - 该节点是否是只读的。

31.2.1 客户

当一个节点需要参照另一个节点的数据库中的信息时，该节点就扮演客户角色。而被参照的节点就是数据库服务器。在图 31-2 中，节点 sales 是寄存 warehouse 和 finance 数据库的节点的客户。

31.2.2 数据库服务器

数据库服务器是一个寄存客户所需数据的数据库的节点。



在图 31-2 中，在 sales 节点上的一个应用发起一个访问 warehouse 和 finance 节点中的数据的分布事务。因此，sales.acme.com 扮演客户节点的角色，而 warehouse 和 finance 两者都是数据库服务器。在本例中，sales 是一个数据库服务器而且还是一个客户，因为该应用还会修改 sales 数据库中的数据。

31.2.3 本地协调器

必须参照其他节点上的数据，才能完成它在分布式事务中的它那部分工作的节点被称为本地协调器。在图 31-2 中，sales 是一个本地协调器，因为它要协调它直接参照的各个节点：warehouse 和 finance。节点 sales 还是一个全局协调器，因为它协调所有包含在该事务中的各个节点。

本地协调器负责在它直接通讯的节点中调度事务，并且：

- 从那些节点接收和向那些节点传播事务状态信息
- 将查询传送到那些节点
- 从那些节点接收查询并将他们传送到其他节点
- 将查询的结果返回给发起查询的节点

31.2.4 全局协调器

发起分布式事务的节点被称为全局协调器。发布分布式事务的数据库应用被直接连接到扮演全局协调器的节点上。例如，在图 31-2 中，在节点 sales 上发布的事务，要参照数据库服务器 warehouse 和 finance 中的信息。因此，sales.acme.com 是这个分布式事务的全局协调器。

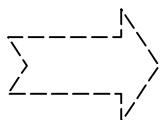
全局协调器是会话树的父或根。在分布式事务中，全局协调器执行如下操作：

- 将所有分布式事务的 SQL 语句、远程过程调用等，发送到直接参照的各个节点，从而形成会话树。
- 指示所有直接参照的节点（但不是提交点站点），准备该事务
- 如果所有节点都准备成功了，则指示提交点站点，发起该事务的全局提交，
- 如果存在失败响应，则指示所有节点，发起该事务的全局回滚操作。

31.2.5 提交点站点

提交点站点的工作是，按照全局协调器的指示，发起提交或回滚操作。系统管理员总是靠给各个节点分配一个提交点强度，而将会话树中一个节点设计成为提交点站点。被选为提交点站点的节点应该是存储最关键数据的节点。

图 31-3 是一个分布式系统的例子，sales 扮演提交点站点的角色：



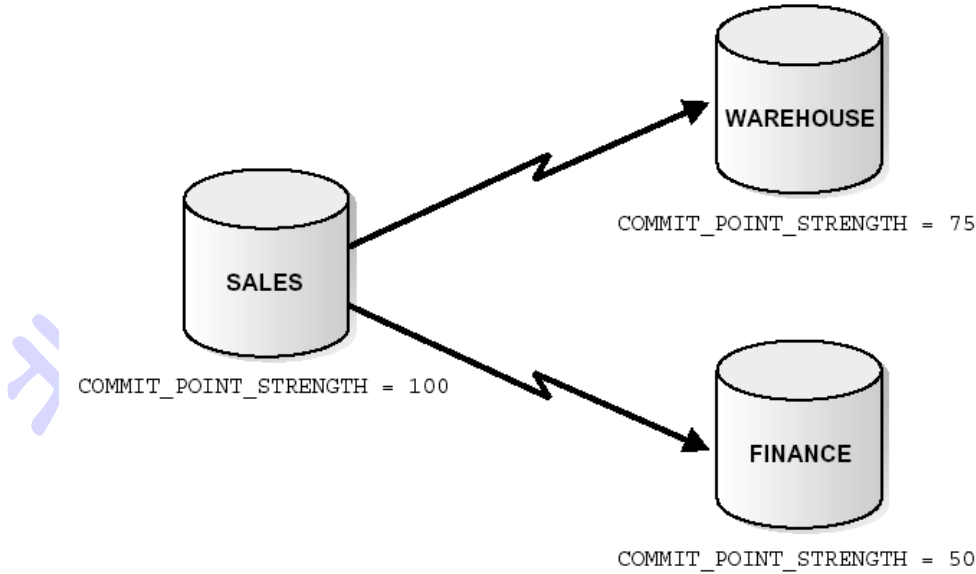


图 31-3 提交点站点

由于如下原因，提交点站点不同于分布式事务中的所有其他站点：

- 提交点站点从不进入准备好状态。因此，如果提交点站点存储着最关键的数据，则这些数据从不保持悬而未决状态，甚至在发生失败时也是如此。在失败的情况下，失败的节点保持准备好状态，保持数据上所需的锁，直到处理完悬而未决的事务。
- 提交点站点会在事务中所包含的其他节点提交之前，先进行提交。实际上，一个分布式事务在提交点站点上的结果，决定了是否提交或回滚所有节点上的事务：即其他节点紧跟提交点站点。全局协调器保证所有的节点都按提交点站点的方式完成事务。

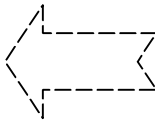
31.2.5.1 分布式事务是如何提交的

当所有非提交点站点已准备好了，并且该事务实际上已经在提交点被提交之后，才认为该分布式事务被提交了。只要提交点站点提交了事务，就会更新该节点上的联机重做日志。

因为提交点站点的日志包含了该提交的记录，所以，即使某些参与的节点可能仍然还处于准备好状态，并且在这些节点上实际上还没有提交该事务，该事务也被认为是被提交了。同样，如果提交还没有被登记在提交点站点中，则该分布式事务还是被认为是未被提交。

31.2.5.2 提交点强度

必须给每个数据库服务器分配一个提交点强度。如果一个数据库服务器在分布式事务中被参照，则其提交点强度的值就决定了它在两阶段提交中会扮演什么角色。特别是，提交点强度决定了所给的节点是否是分布式事务中的提交点站点，因此而在所有其他节点提交之前先进行提交。该值是用初始化参数 COMMIT_POINT_STRENGTH 来指定的。



本节解释 Oracle 是如何决定提交点站点的。

在准备阶段的开始确定的提交点站点，只从参与到事务的节点中选取。发生如下顺序的事件：

①在全局协调器直接参照的节点中，Oracle 将具有最高提交点强度的节点选为提交点站点。

②最初选取的节点决定了，是否为了该事务它不得不从中获取信息的所有节点，具有更高的提交点强度。

③或者是在事务中直接参照的具有最高提交点强度的节点成为提交点站点，或者是它的具有更高提交点强度的服务器之一成为提交点站点。

④当决定了最终的提交点站点之后，全局协调器就向参与到事务的所有节点发送准备响应。

图 31-4 在一个样例会话树中显示了每个节点的提交点强度（在括号中），以及被选为提交点站点的节点：

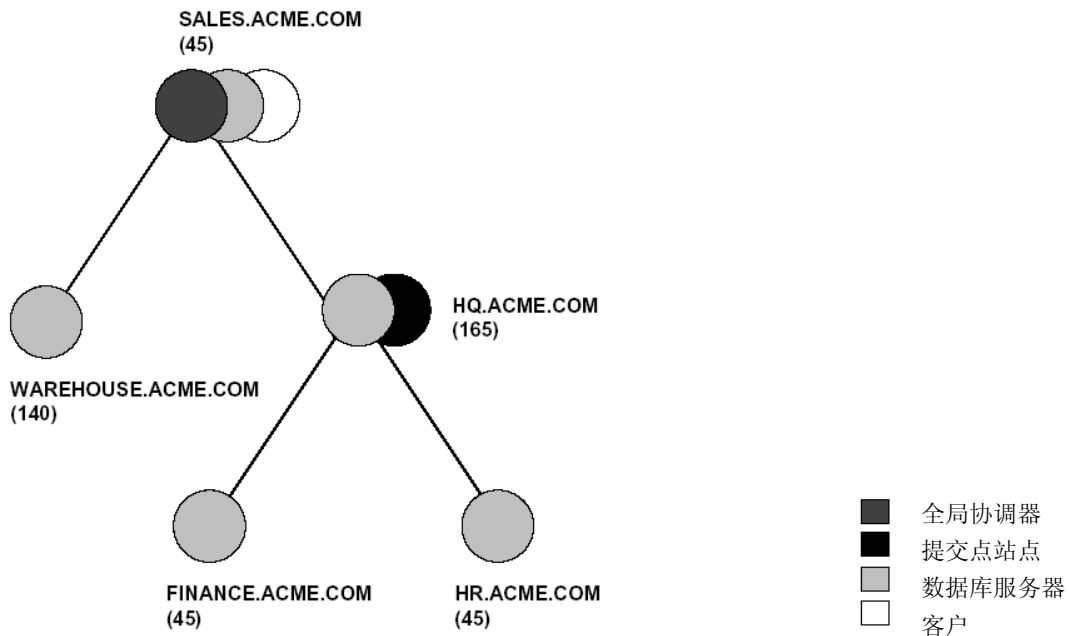
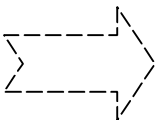


图 31-4 提交点强度和提交点站点的决定

在确定提交点站点时，会运用如下条件：

- 只读节点不能是提交点站点
- 如果全局协调器直接参照的多个节点具有相同的提交点强度，那么 Oracle 就会将其中之一设定成提交点站点。
- 如果一个分布式事务是回滚结束的，那么就不需要准备阶段和提交阶段了。因此，Oracle 就不再决定提交点站点了。而是由全局协调器向所有节点发送一条 ROLLBACK 语句，并结束该分布式事务的处理。

如图 31-4 所示，提交点站点和全局协调器可以是会话树中的不同节点。在做初始化连



接时，会将每个节点的提交点强度传送到协调器。协调器保留与其直接通话的每个节点的提交点强度，以便于在两阶段提交期间能够有效地选取提交点站点。因此，就不需要在每次提交时，在协调器和节点之间交换提交点强度了。

31.3 两阶段提交机制

与本地数据库上的事务所不同的是，一个分布式事务要更改多个数据库上的数据。因此，分布式事务处理更加复杂了。因为 Oracle 就必须像一个自包含单元那样，协调事务中修改的提交和回滚。换句话说，要么整个事务提交，要么整个事务回滚。

Oracle 用两阶段提交机制来保证分布式事务中的数据完整性。在准备阶段中，事务中的发起节点请求其他参与节点承诺提交或回滚事务。在提交阶段中，发起节点请求所有参与节点提交事务。如果办不成这件事，那么就请求所有节点进行回滚。

所有参与分布式事务的节点都应该执行相同的动作：它们应该或者都提交事务或者都回滚事务。Oracle 用两阶段提交机制，来自动地控制和监视分布式事务的提交或回滚，并维护全局数据库（参与到事务中的数据库的集合）的完整性。这个机制是完全透明的，不需要用户方或应用开发者编程。

该提交机制有如下不同阶段，无论何时用户提交了一个分布式事务，Oracle 都会自动地执行这些阶段：

- 准备阶段 被称为全局协调器的发起节点，请求不是提交点站点的参与节点承诺提交或回滚事务，甚至当存在失败时。如果任何节点不能准备好，则回滚事务。
- 提交阶段 如果所有参与者回答协调器，说它们已准备好了，那么协调器就请求提交点站点进行提交。在提交点站点提交之后，协调器就请求所有其他节点提交事务。
- 遗忘阶段 全局协调器遗忘事务。

31.3.1 准备阶段

提交分布式事务的第一个阶段是准备阶段。在这个阶段中，Oracle 并不实际地提交或回滚事务。而是告诉在分布式事务中所参照的所有节点（除提交点站点之外，在第 31 章的“提交点站点”中描述）准备提交。通过准备，节点就会：

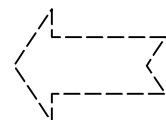
- 在联机重做日志中记录信息，以便以后可以提交或者回滚事务，不管中间的失败。
- 将分布式锁放到被修改的表上，以防止读操作。

当一个节点回答全局协调器，说它已经准备好提交时，该准备好的节点就承诺了以后提交或者回滚事务——但不做单方面的是否提交或回滚事务的决定。该承诺是说，如果实例在该节点失败了，该节点可以用联机日志中的记录将数据库恢复到准备阶段。

注意：在节点已经准备好之后才启动的查询，不能访问相关的已被加锁的数据，直到所有阶段都完成。这个时间是可以忽略的，除非发生了失败。

31.3.1.1 准备阶段的响应类型

当一个节点被告之做准备时，可能用如下方式作回答：



- 准备好 节点上的数据已经被分布式事务中的语句修改了，并且节点已被成功地准备好了。
- 只读 节点上的数据没有被或不能被修改（只能被查询），所以不需要做准备。
- 异常中断 节点不能成功地准备好。
- 准备好响应 当一个节点成功地准备好时，它就发布一条准备好消息。该消息表示该节点在联机日志中有修改的记录，因此它准备好了提交或者回滚。该消息还保证用于该事务的锁可以幸免失败。
- 只读响应 当一个节点被请求做准备时，并且涉及数据库的 SQL 语句并不修改该节点的数据时，该节点就回答一条只读消息。该消息表示该节点不会参与到提交阶段。

全部或部分分布式事务存在如下三种只读的情况：

情况	条件	结果
部分只读	满足任何如下一种情况： ■ 在一个或多个节点上只发布了查询。 ■ 没有修改数据。 ■ 由于启动了触发器或破坏了约束，而使修改被回滚了。	当被请求做准备时，只读节点承认它们的状态。它们给它们的本地协调器发送一条只读响应。因此，提交阶段完成得更快了，因为 Oracle 会从以后的处理中排除只读节点。
带有准备阶段的完全只读	满足所有如下情况： ■ 没有修改数据。 ■ 不是用 SET TRANSACTION READ ONLY 语句来启动事务的。	所有的节点承认它们在准备阶段是只读的，所以不需要提交阶段了。在不知道是否所有的节点都是只读的情况下，全局协调器还必须执行准备阶段。
不带有两阶段提交的全只读	满足所有如下情况： ■ 没有修改数据。 ■ 是用 SET TRANSACTION READ ONLY 语句来启动事务的。	在事务中只允许查询，因此，全局协调器不需要执行两阶段提交。其他事务所做的修改也不会降低全局的事务级的读一致性，因为全局的 SCN 会在各个节点之间作协调。事务不使用回滚段。

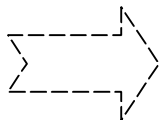
注意，如果一个分布式事务被设置成了只读，那么，它就不会使用回滚段。如有多个用户连接到数据库，并且他们的事务没有被设置成了只读，那么甚至在它们只执行查询的情况下，也会给它们分配回滚空间。

- 异常中断响应 当一个节点不能成功地准备好时，它就执行如下动作：
 - ①释放事务当前所占用的资源，并回滚事务的本地部分。
 - ②向在分布式事务中参照它的各个节点回答一条异常中断消息。

然后，这些动作被传播到分布式事务中所包含的其他节点，以便于它们能够回滚该事务，并保证全局数据库中的数据的完整性。这条响应实施了分布式事务的首要规则：事务中的所有节点，在相同的逻辑时间上，要么都提交事务要么都回滚事务。

31.3.1.2 准备阶段的步骤

为了完成准备阶段，除提交点站点之外的其他节点要执行如下步骤：



①节点请求其后代（即，以后参照的节点）准备提交。

②节点查看是否事务修改了它自己或其后代上的数据。如果没有修改数据，那么该节点就跳过剩下的步骤，并返回一条只读响应（参见第 31 章的“只读响应”）。

③如果数据被修改了，则该节点分配它需要用于提交事务的资源。

④节点将对应于事务所做的修改的重做记录，保存到它的联机重做日志中。

⑤节点保证用于事务的锁不会失败。

⑥节点用准备好响应来回答发起结点（参见第 31 章的“准备好响应”），或者如果它或其后代在做准备时不成功，那么就用异常中断响应来回答（参见第 31 章的“异常中断响应”）。

这些动作保证该节点能够有序地提交或回滚节点上的事务。然后，准备好的节点就一直等到从全局协调器收到 COMMIT 或 ROLLBACK 请求。

当各个节点都准备好之后，分布式事务就被称作悬而未决（参见第 31 章的“悬而未决的事务”）。它保持悬而未决状态，直到所有的修改被提交或者被回滚。

31.3.2 提交阶段

提交分布式事务的第二个阶段是提交阶段。在进入这个阶段之前，除了提交点站点之外，分布式事务所参照的所有节点都已保证它们都准备好了，即它们具有了提交该事务所需的资源。

31.3.2.1 提交阶段的步骤

提交阶段由如下步骤组成：

①全局协调器指示提交点站点进行提交。

②提交点站点进行提交。

③提交点站点告诉全局协调器它已经提交了。

④全局和本地协调器向所有节点发送一条消息，指示它们提交事务。

⑤在每个节点上，Oracle 提交分布式事务的本地部分并释放锁。

⑥在每个节点上，Oracle 在本地重做日志中记录一条附加的重做项，表示事务已经被提交了。

⑦各个参与节点通知全局协调器，它们已经提交了。

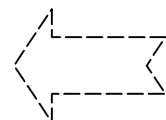
当提交阶段完成后，分布式系统的所有节点上的数据就是一致的了。

31.3.2.2 保证全局数据库的一致性

每个已提交的事务都有一个相关的系统修改号码（SCN），以便唯一地标识那个事务中的 SQL 语句所做的修改。SCN 功能就像唯一标识已提交的数据库版本的一个内部的 Oracle 时间戳一样。

在分布式系统中，当发生所有如下动作时，就会协调各个通讯节点的 SCN：

- 发生使用一个或多个数据库链接所描述的路径的连接。
- 执行分布式 SQL 语句
- 提交分布式事务



其他的好处是，在分布式系统的各个节点之间协调 SCN，确保了在语句和事务两个层次上的全局读一致性。如果需要，还可以完成全局的基于时间的恢复。

在准备阶段中，Oracle 会在事务所包含的所有节点中，决定最高的 SCN。然后，在提交点站点用最高的 SCN 提交该事务。然后，将该提交的 SCN 发送到所有做好提交决定的，已经准备好了的节点。

31.3.3 遗忘阶段

当参与节点通知提交点站点，它们已经提交了之后，该提交点站点就可以遗忘该事务了。发生如下步骤：

- ①当从全局协调器那里接收到了所有节点都已经提交了的通知之后，提交点站点就抹去有关这个事务的状态信息。
- ②提交点站点通知全局协调器，它已经抹去了状态信息。
- ③全局协调器抹去它自己的有关这个事务的信息。

31.4 悬而未决事务

两阶段提交机制保证了所有的节点，要么一起提交要么一起回滚。如果因为系统或网络故障而使这第三阶段的任何阶段失败，那会发生什么呢？该事务就会变成悬而未决。

分布式事务可能会在如下情况发生时，变成悬而未决：

- 运行 Oracle 软件的服务器机器崩溃。
- 在分布式处理中所包含的两个或多个 Oracle 数据库之间的网络连接被断开了。
- 未处理的软件错误发生了。

当机器、网络或软件问题解决之后，RECO 就会自动地处理悬而未决的事务。在 RECO 完全解决该事务之前，数据对读和写两者都加锁。因为 Oracle 还不能决定应给查询显示哪个版本的数据，所以 Oracle 也冻结了读操作。

本节包括如下内容：

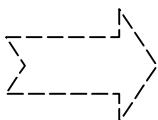
- 悬而未决事务的自动解决
- 悬而未决事务的手动解决
- 悬而未决事务的系统修改号码的实用性

31.4.1 悬而未决事务的自动解决

在大多数情况下，Oracle 会自动地解决悬而未决事务。假设在下面的方案中有两个节点，local（本地）和 remote（远程）。local 节点是提交点站点。用户 scott 连接到 local 站点，并且执行和提交更新 local 和 remote 的分布式事务。

31.4.1.1 准备阶段失败

图 31-5 说明了，当在分布式事务的准备阶段出现失败时，所发生的事件序列：



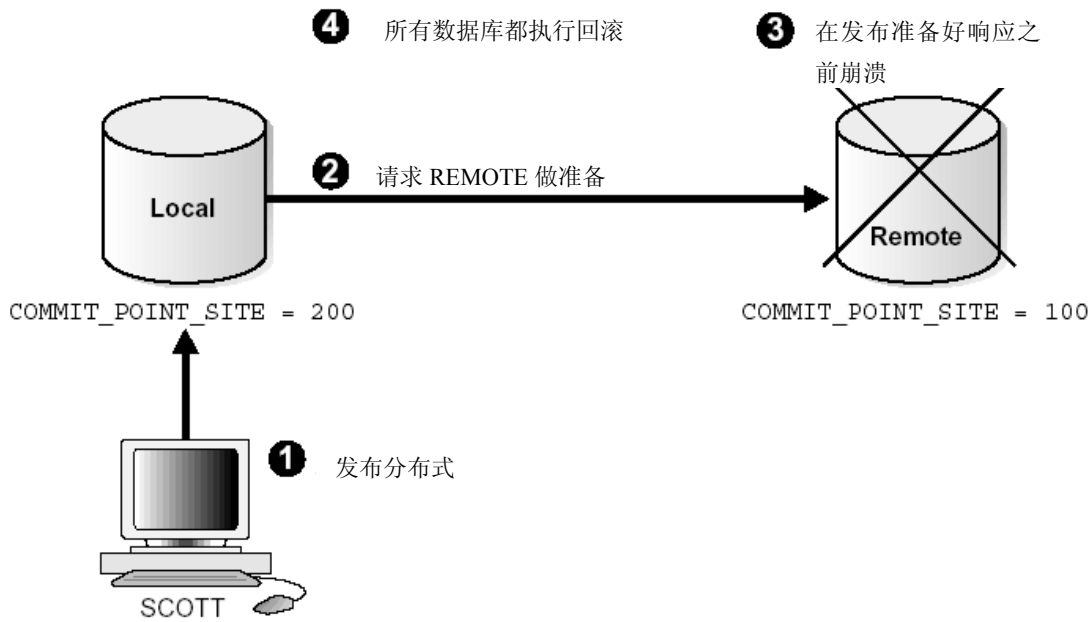


图 31-5 准备阶段失败

发生如下步骤：

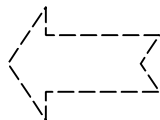
- ①scott 连接到 local 并执行分布式事务
- ②全局协调器，在本例中还是提交点站点，请求除提交点站点之外的所有站点承诺，当需要时能提交或回滚。
- ③remote 在向 local 发布准备好响应之前崩溃。
- ④当 remote 站点恢复后，通过 RECO 处理，最后该事务在每个数据库上被回滚。

31.4.1.2 提交阶段失败

图 31-6 说明了，当在分布式事务的提交阶段出现失败时，所发生的事件序列：

发生如下步骤：

- ①scott 连接到 local 并执行分布式事务。
- ②全局协调器，在本例中还是提交点站点，请求除提交点站点之外的所有数据库承诺，当需要时能提交或回滚。
- ③提交点站点接收到从 remote 发来的准备好消息，表示它可以提交。
- ④提交点站点在本地提交该事务，然后向 remote 发送一条提交消息，请求它提交。
- ⑤remote 接收到该提交消息，但由于网络故障，所以它不能作回答。
- ⑥当网络恢复后，通过 RECO 处理，最后事务在 remote 上被提交。



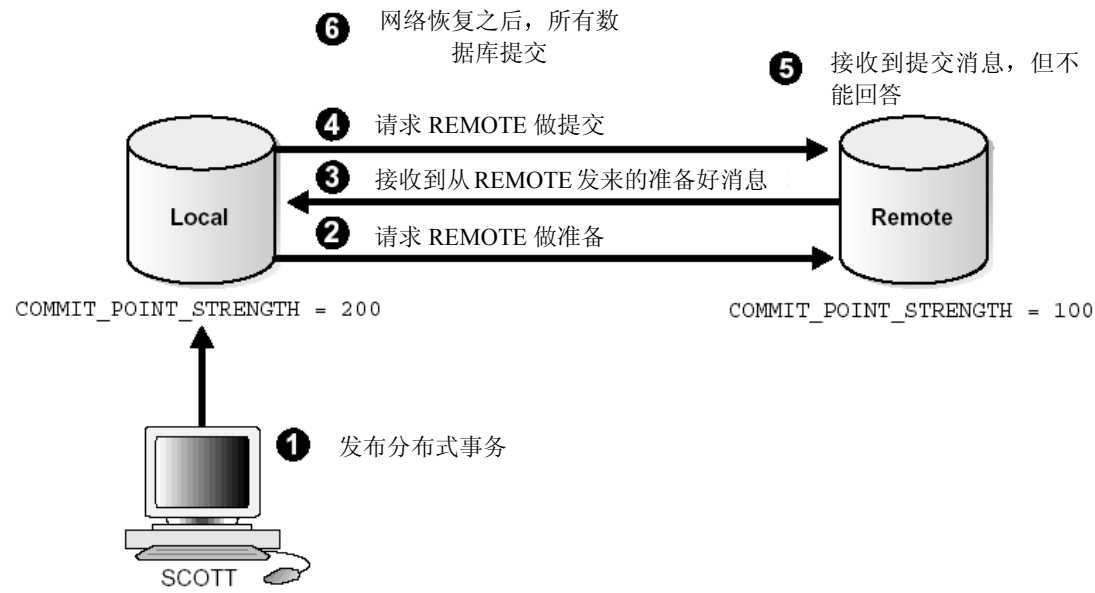


图 31-6 提交阶段失败

31.4.2 悬而未决事务的手动解决

在如下情况时，才需要手动解决悬而未决事务：

- 悬而未决事务在关键数据或回滚段上有锁。
- 不能迅速修复机器、网络或软件失败的原因。

悬而未决事务的解决可能是复杂的。该过程需要你做如下事情：

- 识别悬而未决事务的事务标识号码。
- 查询 DBA_2PC_PENDING 和 DBA_2PC_NEIGHBORS 视图，确定是否该事务所包含的数据库已经提交了。
- 如果需要，用 COMMIT FORCE 语句强行提交，或用 ROLLBACK FORCE 语句强行回滚。

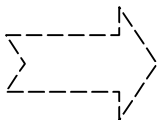
31.4.3 悬而未决事务的系统修改号码的实用性

系统修改号（System Change Number (SCN)）是一个用于已提交的数据库版本的内部时间戳。Oracle 数据库服务器用该 SCN 时钟值来保证事务的一致性。例如，当一个用户提交一个事务时，Oracle 就会在联机重做日志中为这个提交记录一个 SCN。

Oracle 用这些 SCN 来协调在不同数据库之间的分布式事务。例如，Oracle 按如下方式使用这些 SCN：

- ①一个应用用一个数据库链接来建立一个连接。
- ②分布式事务用所包含的所有数据库中的最高的全局 SCN 进行提交。
- ③将提交的全局 SCN 发送到事务所包含的所有数据库。

对于分布式事务来说，这些 SCN 是重要的，因为其作用就像是事务的同步提交时间戳一样——甚至在事务失败时。如果一个事务变成悬而未决了，管理员就可以用 SCN 来协调



对全局数据库的修改。用于事务提交的全局 SCN 还可以在以后用于标识该事务，例如，在分布式恢复时。

31.5 分布式事务处理：案例学习

在这个方案中，一个公司有单独的 Oracle 数据库服务器，sales.acme.com 和 warehouse.acme.com。当用户将销售记录插入到 sales 数据库中时，就会对 warehouse 数据库中的相关的记录做更新。

这个分布式处理的案例学习说明：

- 会话树的定义
- 如何决定提交点站点
- 何时发送准备消息
- 何时实际地提交事务
- 在本地存储有关事务的什么信息

31.5.1 阶段 1：客户应用发布 DML 语句

在销售部门，一个销售员使用 SQL*Plus 输入了一条销售订单，然后提交。该应用就会发布几条 SQL 语句，以便将该订单输入到 sales 数据库，并更新 warehouse 数据库中的货存：

```
CONNECT scott/tiger@sales.acme.com ...;
INSERT INTO orders ...;
UPDATE inventory@warehouse.acme.com ...;
INSERT INTO orders ...;
UPDATE inventory@warehouse.acme.com ...;
COMMIT;
```

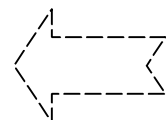
这些 SQL 语句是一个单独的分布式事务的一部分，保证所有发布的 SQL 语句，作为一个单元，或者成功或者失败。将这些语句当作一个单元就能防止，输入了订单然而又没有更新货存以反映订单的可能性。实际上，事务会保证在全局数据库中的数据的一致性。

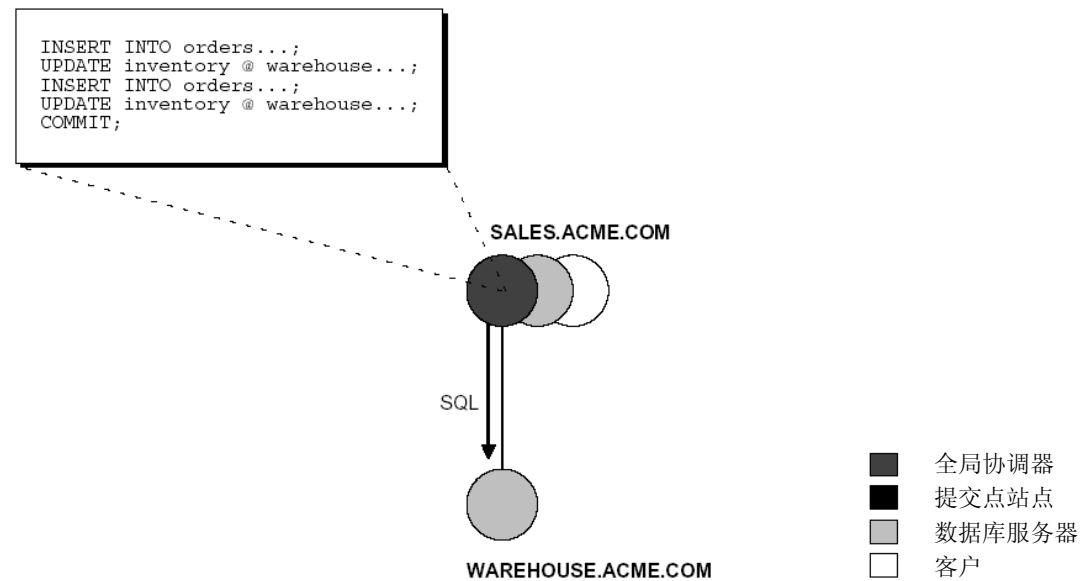
当执行事务中的每条 SQL 语句时，就会定义会话树，如图 31-7 所示。

注意事务的如下几个方面：

- 运行在 sales 数据库上的一个订单输入应用发起该事务。因此，sales.acme.com 是该分布式事务的全局协调器。
- 该订单输入应用在 sales 数据库中插入一条新的销售记录，并更新 warehouse 的货存。因此，节点 sales.acme.com 和 warehouse.acme.com 两者都是数据库服务器。
- 因为 sales.acme.com 更新存货，所以它是 warehouse.acme.com 的客户。

这个阶段为这个分布式事务完成会话树的定义。该会话树中的每个节点已经获得了所需的数据块，以便执行参照本地数据的各种 SQL 语句。甚至在这些 SQL 语句已经被执行完了之后，仍然保留这些锁，直到完成两阶段提交。





31.5.2 阶段 2: Oracle 决定提交点站点

在发布 COMMIT 语句之后，Oracle 立即决定提交点站点。全局协调器 sales.acme.com 被决定为提交点站点，如图 31-8 所示。

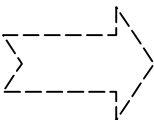


图 31-8 决定提交点站点

31.5.3 阶段 3: 全局协调器发送准备响应

准备阶段包括如下几步：

①当 Oracle 决定了提交点站点之后，全局协调器就向该会话树直接参照的所有结点（除了提交点站点之外）发送准备消息。在本例中，warehouse.acme.com 是唯一被请求做准备



的节点。

②节点 `warehouse.acme.com` 尝试做准备。如果一个节点能够保证它能够提交该事务的本地的相关部分，并且能够在它的本地重做日志中记录该提交信息，那么该节点就能成功地作好准备。在本例中，只有 `warehouse.acme.com` 接收到了准备消息，因为 `sales.acme.com` 是提交点站点。

③节点 `warehouse.acme.com` 用准备好消息回答 `sales.acme.com`。

当每个节点做准备时，它向请求它做准备的节点发送一条返回消息。根据这个响应，可能会发生如下情况之一：

- 如果任何一个被请求做准备的节点，向全局协调器回答一条异常中断消息，那么全局协调器就会告诉所有节点回滚该事务，并且结束该操作。
- 如果所有被请求做准备的节点，都向全局协调器回答一条已准备好或只读消息，即，它们已经做好了，那么全局协调器就会请求提交点站点提交该事务。

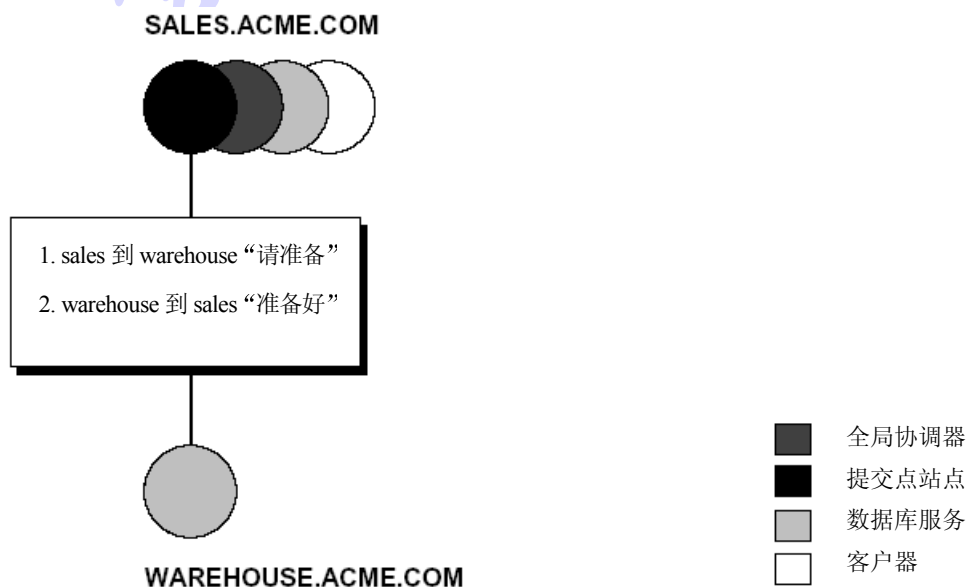


图 31-9 发送和应答准备消息

31.5.4 阶段 4：提交点站点提交

提交点站点的事务的提交包括如下几步：

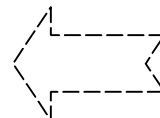
①节点 `sales.acme.com` 接收到 `warehouse.acme.com` 发来的已准备好的应答，指示提交点站点提交该事务。

②提交点站点马上在本地提交该事务，并在它的本地重做日志中记录这件事。

尽管 `warehouse.acme.com` 还没有提交，但是该事务的输出已被预先确定了。换句话说，该事务一定会在所有节点被提交，即使一个给定节点的提交能力被延迟了。

31.5.5 阶段 5：提交点站点通知全局协调器有关提交之事

这个阶段包括如下几步：



①提交点站点告诉全局协调器该事务已经被提交了。因为在本例中，提交点站点和全局协调器是同一个节点，所以就不需要做什么操作了。因为提交点站点在它的联机日志中记录了提交这件事，所以它知道该事务已经被提交了。

②全局协调器确认该事务已经在分布式事务所包含的所有其他节点上被提交了。

31.5.6 阶段 6：全局和本地协调器告诉所有节点进行提交

事务中的所有节点的事务提交包括如下几步：

①当提交点站点已经通知了全局协调器有关提交之事之后，全局协调器就通知所有其他直接参照的节点进行提交。

②然后，任何本地协调器就指示其服务器进行提交，等等。

③每个节点（包括全局协调器）提交该事务，并在本地记录相应的重做日志项。当每个节点提交时，就释放本地为那个事务而一直占用的资源锁。

在图 31-10 中，sales.acme.com（既是提交点站点又是全局协调器）已经在本地提交了该事务。现在，sales 指示 warehouse.acme.com 提交该事务。

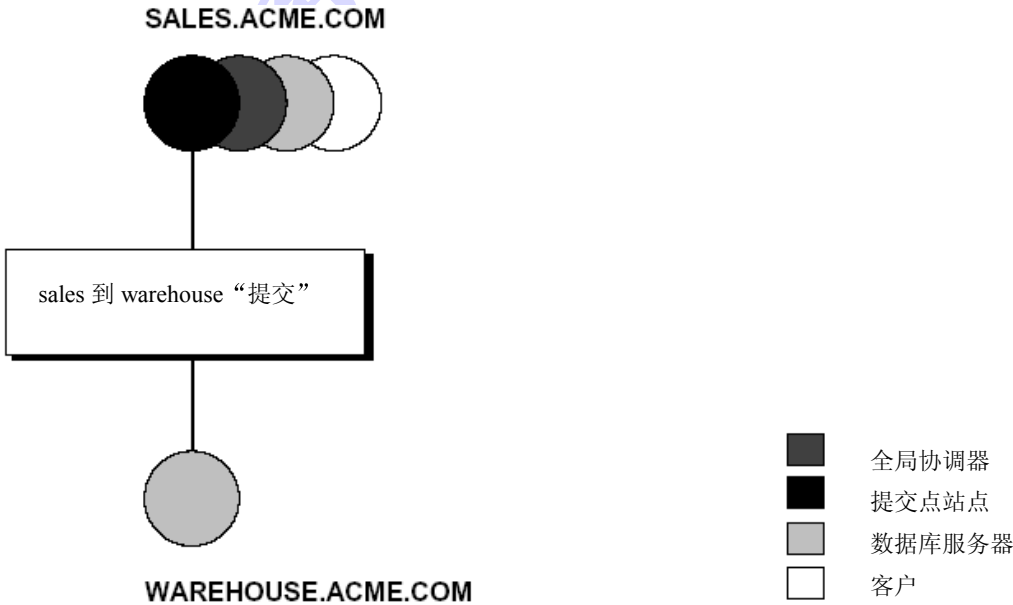


图 31-10 指示节点进行提交

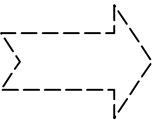
31.5.7 阶段 7：全局协调器和提交点站点完成提交

按如下几步完成事务的提交：

①当所有被参照的节点和全局协调器已经提交了事务之后，全局协调器就将此事通知提交点站点。

②等待这条消息的提交点站点抹去有关这个分布式事务的状态信息。

③提交点站点告诉全局协调器，此事已结束了。换句话说，提交点站点遗忘了有关提交该分布式事务的事了。因为包括在该两阶段提交中的所有节点，已经成功地提交了该事

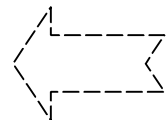


务，所以这个动作是被允许的。因此，它们将来也就不再需要决定该事务的状态了。

④通过遗忘该事务，全局协调器终结该事务。

当 COMMIT（提交）阶段完成之后，分布式事务本身也就完成了。上面所描述的这几步都是自动地实现的，并且是在一转眼的工夫中完成的。

北京拓墣电子出版社



第 32 章 管理分布式事务

本章描述怎么管理和解决分布式事务。本章包括以下内容：

- 设置分布式事务的初始化参数
- 查看有关分布式事务的信息
- 决定如何处理悬而未决的事务
- 手动处理悬而未决的事务
- 清除数据字典中悬而未决的行
- 手动提交一个悬而未决的事务：例子
- 由于加锁而造成的数据访问失败
- 模拟分布式事务失败
- 管理读一致性

32.1 设置分布式事务的初始化参数

可以设置分布式事务的初始化参数，来控制事务处理的行为。下面的表说明了与分布式事务处理相关的初始化参数。

参数	说明
DISTRIBUTED_TRANSACTIONS	指定数据库能够同时参与的分布式事务的最大数量。
COMMIT_POINT_STRENGTH	指定用于决定分布式事务的提交点站点的值。

这部分包括如下两个内容：

- 限制分布式事务的数量
- 指定一个节点的提交点强度

32.1.1 限制分布式事务的数量

初始化参数 `DISTRIBUTED_TRANSACTIONS` 用来限制一个给定的实例（作为客户或服务器）能够同时参与的分布式事务的最大数量。这个参数的默认值是依赖于操作系统的。

如果 Oracle 已经到达了 this 限制值，那么一个后来的用户发布一条参照远程数据库的 SQL 语句时，系统就会回滚这条语句并返回以下错误信息：

```
ORA-2042: too many global transactions
```

例如，假设将一个给定实例的该参数设置如下：

```
DISTRIBUTED_TRANSACTIONS = 10
```

此时，最多 10 个会话能够同时处理一个分布式事务。如果一个新增的会话试图发布一条需要分布式访问的 DML 语句时，那么，Oracle 就会给该会话返回一个错误消息，并回滚该语句。

注意：Oracle 建议将 `DISTRIBUTED_TRANSACTIONS` 的值，设置成你的环境中的分布式数据库站点的总个数。

32.1.1.1 增加事务限制

当一个实例有规律地参与到几个分布式事务中，并且系统经常返回 ORA-2042 消息时，就应该考虑增加 DISTRIBUTED_TRANSACTIONS 的值了。增加这个限制值就允许更多的用户能同时发布分布式事务了。

32.1.1.2 减少事务限制

如果你的站点正遇到不寻常的多次网络失败，你就可以临时性地减少 DISTRIBUTED_TRANSACTIONS 的值。这个操作能够限制你的站点所涉及的悬而未决事务的数量，因而限制了你的站点的被加锁的数据的数量，以及限制了可能不得不解决的悬而未决事务的数量。

32.1.1.3 停用分布式事务处理

如果将 DISTRIBUTED_TRANSACTIONS 设为 0，那么，任何会话都不能发布分布式 SQL 语句了。同时，在本地实例启动时，RECO 后台进程是不会启动的。Oracle 将不会自动解决那些可能出现的悬而未决的事务。所以，仅仅当一个新的实例启动的时候，并且确认上一个实例被关闭之后没有留下悬而未决的事务的时候，才将这个初始化参数设为 0，以便防止分布式事务。

32.1.2 指定一个节点的提交点强度

具有最高提交点强度的数据库决定了，在一个分布式事务中，哪个节点首先提交。当给每个节点指定提交点强度时，应确保在准备阶段或提交阶段发生失败时，最关键的服务器仍然是非冻结的。如下初始化参数决定了一个节点的提交点强度：

COMMIT_POINT_STRENGTH

该参数的默认值是依赖于操作系统的。它的取值范围是 0 到 255 之间的任何整数。例如，为了将一个数据库的提交点强度设为 200，就要在那个数据库的初始化参数文件中，包括下面这行：

COMMIT_POINT_STRENGTH = 200

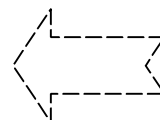
提交点强度仅仅被用来决定在一个分布式事务中的提交点站点。

当给一个数据库设置提交点强度时，要注意考虑以下问题：

- 因为提交点站点存储了有关事务的状态信息，所以，为了使其他节点能够获得有关事务的状态的信息，提交点站点就不应该是一个经常不可靠或经常不可用的节点。
- 数据库的提交点强度的设置，与数据库中的关键的共享数据的数量是相关的。例如，大型计算机上的数据库通常比 PC 机上的数据库共享用户的数据。因此，就应该给大型机设置一个比 PC 机更高的提交点强度。

32.2 查看有关分布式事务的信息

每个数据库的数据字典都存储了有关所有打开的分布式事务的信息。可以使用数据字典



典表和视图来获得有关事务的信息。本节包括以下内容：

- 事务命名
- 确定已准备好事务的 ID 号码和状态
- 跟踪悬而未决事务的会话树

32.2.1 事务命名

启动 Oracle9i 时，可以给一个事务命名。这有助于标识一个特别的分布式事务，它代替了用 COMMIT COMMENT 语句来实现这个目的。

要给一个事务命名，就使用 SET TRANSACTION.....NAME 语句，例如：

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE  
NAME 'update inventory checkpoint 0';
```

这个例子表明，该用户启动了一个隔离水平等于 SERIALIZABLE 的新的事务，并将该事务命名为 ‘update inventory checkpoint 0’。

对于分布事务来说，当事务被提交时，该名字被送到各个参与的站点。当事务的名字存在时，就会忽略存在的 COMMIT COMMENT。

事务的名字在 V\$TRANSACTION 视图中的 NAME 列中显示，并且当事务被提交时，会将该事务的名字放置在这个视图的 TRAN_COMMENT 列中。

32.2.2 确定已准备好事务的 ID 号码和状态

如下视图显示已经定义在本地数据库中定义并存储在数据字典中的数据库链接：

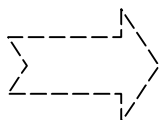
视图	用途
DBA_2PC_PENDING	列出了所有的悬而未决事务。这个视图在填入悬而未决事务之前是空的；当该事务被解决后，该视图也将被清空。

使用该视图可以为一个特殊的事务 ID 决定全局提交数。当手动处理一个悬而未决事务时，可以使用这个全局提交数。

下面的表显示了大多数有关的列：

表 32-1 DBA_2PC_PENDING

列	说明
LOCAL_TRAN_ID	本地事务的标识，格式为 integer.integer.integer。 注意：当一个连接的 LOCAL_TRAN_ID 和 GLOBAL_TRAN_ID 相同时，那么该节点就是该事务的全局协调器。
GLOBAL_TRAN_ID	全局数据库的标识，格式为 global_db_name.db_hex_id.local_tran_id，其中 db_hex_id 是用来唯一标识该数据库的八个字符长的十六进制数。这个公共的事务 ID 号在分布式事务的每个节点中都是相同的。 注意：当一个连接的 LOCAL_TRAN_ID 和 GLOBAL_TRAN_ID 相同时，那么该节点就是该事务的全局协调器。



(续表)

列	说明
STATE	见表 32-2，“DBA_2PC_PENDING 的 STATE 列”。
MIXED	“YES”意味着事务的一部分已经在一个节点上被提交，而在另一个节点上被回滚。
TRAN_COMMENT	事务的注释，或者如果使用了事务命名，那么当事务被提交时，事务的名字就会出现在此处。
HOST	主机名
COMMIT#	已提交事务的全局提交数。

表 32-2 DBA_2PC_PENDING 的 STATE 列

列	说明
Collecting	通常情况下，只有全局协调器或者本地协调器才使用这个条目。节点在能够决定它是否能够准备好之前，要收集来自其他数据库服务器的信息。
Prepared	节点已经准备好，可能或者也可能没有将已准备好的消息通知本地协调器。但此时，该节点还没有接收到提交的请求。仍保持着准备好的状态，控制着提交该事务所必需的任何本地资源锁，
Committed	节点（任何类型）已经提交了事务，但是该事务所包含的其他节点可能并没有提交。也就是说，该事务在一个或多个节点上仍然是悬而未决的。
Forced commit	数据库管理员进行判断后，可以强行提交悬而未决的事务。如果一个事务由数据库管理员在本地节点进行手动提交时，就会产生该项目。
Forced abort (rollback)	数据库管理员进行判断后，可以强行回滚悬而未决的事务。如果一个事务由数据库管理员在本地节点进行手动回滚时，就会产生这个项目。

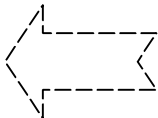
执行如下脚本可以查询 DBA_2PC_PENDING 中的相关信息（包括范例输出）：

```
COL LOCAL_TRAN_ID FORMAT A13
COL GLOBAL_TRAN_ID FORMAT A30
COL STATE FORMAT A8
COL MIXED FORMAT A3
COL HOST FORMAT A10
COL COMMIT# FORMAT A10
SELECT LOCAL_TRAN_ID, GLOBAL_TRAN_ID, STATE, MIXED, HOST, COMMIT#
FROM DBA_2PC_PENDING
/
```

```
SQL> @pending_txn_script
LOCAL_TRAN_ID GLOBAL_TRAN_ID STATE MIX HOST COMMIT#
```

```
-----
1.15.870 HQ.ACME.COM.ef192da4.1.15.870 commit no dlsun183 115499
```

这个输出表明，本地事务 1.15.870 已经在这个节点提交过了，但是它有可能在其他一



个或多个节点上是悬而未决的。因为 LOCAL_TRAN_ID 和 GLOBAL_TRAN_ID 的本地部分是相同的，所以这个节点是该事务的全局协调器。

32.2.3 跟踪悬而未决事务的会话树

下面的视图可以显示，哪些悬而未决的事务是从远程客户那里获得的，哪些是送出给远程服务器的：

视图	用途
DBA_2PC_NEIGHBORS	列出所有获得的（从远程客户）和送出的（给远程服务器）悬而未决的分布式事务。它也表明了该本地节点是不是事务的提交点站点。 在填入悬而未决事务之前，这个视图是空的。在该事务被解决后，视图将被清空。

当一个事务是悬而未决的时候，可能需要确定在会话树中哪个节点扮演哪个角色，就可使用这个视图来确定：

- 一个给定事务所有的获得的和送出的连接。
- 这个节点是否是一个给定事务的提交点站点。
- 这个节点是否是一个给定事务的全局协调器。（如果它的本地事务 ID 号和全局事务 ID 号相同）

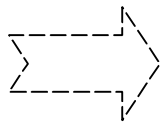
下面的表显示了大部分有关的列：

表 32-3 DBA_2PC_NEIGHBORS

列	说明
LOCAL_TRAN_ID	本地事务的标识，格式为 integer, integer, integer。 注 意：当一个连接的 LOCAL_TRAN_ID 和 GLOBAL_TRAN_ID、DBA_2PC_PEND 相同时，那么这个节点就是该事务的全局协调器。
IN_OUT	获得的事务为 IN；送出的事务为 OUT。
DATABASE	对获得的事务来说，是指请求本地节点信息的客户数据库的名称；对送出的事务来说，是指用于访问远程服务器上信息的数据库链接的名称。
DBUSER_OWNER	对获得的事务来说，是指被远程数据库链接用于连接的本地帐户；对送出的事务来说，是指该数据库链接的拥有者。
INTERFACE	‘C’ 代表提交信息；‘N’ 或者是表示已准备好状态的一条消息，或者是一条请求只读提交的请求。 当 ‘IN-OUT’ 为 ‘OUT’ 时，‘C’ 表示，该连接的远程端的后代是提交点站点，并且知道是提交还是中断。‘N’ 表示，本地节点正在通知远程节点，说它已准备好了。 当 ‘IN-OUT’ 为 ‘IN’ 时，‘C’ 表示，本地节点或送出的连接的远程端的一个数据库是提交点站点。‘N’ 表示，远程节点正在通知本地节点，说它已准备好了。

执行如下脚本可以查询 DBA-2PC-PENDING 中的有关信息（包括范例输出）：

```
COL LOCAL_TRAN_ID FORMAT A13
COL IN_OUT FORMAT A6
COL DATABASE FORMAT A25
```



```

COL DBUSER_OWNER FORMAT A15
COL INTERFACE FORMAT A3
SELECT LOCAL_TRAN_ID, IN_OUT, DATABASE, DBUSER_OWNER, INTERFACE
FROM DBA_2PC_NEIGHBORS
/
SQL> CONNECT SYS/password@hq.acme.com
SQL> @neighbors_script
LOCAL_TRAN_ID IN_OUT DATABASE DBUSER_OWNER INT
-----
1.15.870 out SALES.ACME.COM SYS C

```

这个输出表明，本地节点向远程服务器 **sales** 发出了一条送出的请求，以便提交事务 1.15.870。如果 **sales** 提交了这个事务，而其他节点没有提交，那么，你就知道 **sales** 是提交点站点了——因为提交点站点总是最早提交的。

32.3 决定如何处理悬而未决的事务

当两阶段提交的任何方面出现故障时，事务就变成悬而未决的了。分布式事务在下面情况时将会变成悬而未决：

- 运行 **oracle** 软件的服务器发生崩溃。
- 在分布式处理中所包含的两个或多个 **oracle** 数据库之间的网络连接被断开了。
- 产生了一个未被管理的软件错误。

可以手动地强行提交或回滚一个本地的悬而未决的分布式事务。因为这个操作可能会产生一致性方面的问题，所以仅仅在特殊情况下才这样做。

本节包括如下内容：

- 发现两阶段提交的问题
- 决定是否执行手动覆盖
- 分析事务数据

32.3.1 发现两阶段提交的问题

用下面的错误消息之一，来将出现的问题通知给提交分布式事务的用户应用：

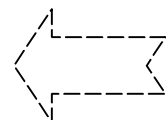
```

ORA-02050: transaction ID rolled back,
some remote dbs may be in-doubt
ORA-02051: transaction ID committed,
some remote dbs may be in-doubt
ORA-02054: transaction ID in-doubt

```

一个健壮的应用，应当在收到任何以上错误之一时，就保存有关事务的信息。如果以后希望进行分布式事务的手动恢复时，就可以使用这些信息。

由于网络或系统故障而导致的一个或多个悬而未决的事务，不需要任何节点的管理员进行操作。**Oracle** 的自动恢复特性会透明地完成任何悬而未决的事务，以保证，在网络或



系统的故障被解决后，会话树上的所有节点都会发生相同的结果（即，全部提交或全部回滚）。

但是，对于延伸出的损耗，可以强行提交或回滚一个事务，以释放所有被加锁的数据。应用必须处理这种可能性。

32.3.2 决定是否执行手动覆盖

仅仅当存在下列情形之一时，才需要手动覆盖一个特定的悬而未决的事务：

- 一个悬而未决的事务锁住了其他事务所需要的数据。当 ORA-01591 错误信息妨碍了用户的事务时，就会产生这种情况。
- 一个悬而未决的事务妨碍了其他事务使用回滚段的盘区。一个悬而未决的事务的本地事务 ID 的开头部分对应于回滚段的 ID，就像在数据字典视图 DBA_2PC_PENDING 和 DBA_ROLLBACK_SEGS 中所列出的那样。
- 妨碍两阶段提交阶段完成的故障，不能在可接收的时间内改正。这种情况的例子包括，通信网络已经被破坏，或一个被破坏了的数据库需要很长的恢复时间。

通常，应该先与其他位置的管理员商量后，再决定在本地强行恢复一个悬而未决的分布式事务。一个错误的决定会导致数据库的不一致性，从而造成回溯困难，而必须进行手动改正错误。

如果上述的情况没有出现，总是使用 Oracle 的自动恢复功能来完成该事务。但如果遇到上面任何条件，就要考虑在本地覆盖悬而未决的事务。

32.3.3 分析事务数据

如果决定强行完成一个事务，就要注意按照下列目标来分析有用的信息。

32.3.3.1 查找已经提交或回滚的节点

使用 DBA_2PC_PENDING 视图来查找已经提交或回滚过事务的节点。如果能发现一个已解决了该事务的节点，那么就可以跟踪那个节点所发生的动作。

32.3.3.2 寻找事务注释

看一看是否有分布式事务的信息出现在 DBA_2PC_PENDING 的 TRAN_COMMENT 列中。注释被包括在 COMMIT 语句的 COMMENT 子句中，或者如果使用了事务名称，当提交该事务时，该事务的名称就会被放置到 TRAN_COMMENT 域中。

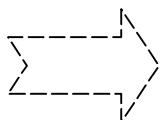
例如，一个悬而未决的分布式事务的注释，可以显示出该事务的来源，以及它是什么类型的事务：

```
COMMIT COMMENT 'Finance/Accts_pay/Trans_type 10B';
```

一个 SET TRANSACTION...NAME 语句可能已经被用来在一个事务名称中提供这些信息了（这是更可取的）。

32.3.3.3 寻找事务建议

看一看在 DBA_2PC_PENDING 的 ADVICE 列中，是否为分布式事务提供了任何信息。



一个应用可能用 ALTER SESSION 语句的 ADVISE 子句来指示建议，即是否强行提交或强行回滚该分布式事务的单独的部分。

在准备阶段发送给各个节点的建议，只有在最近的 DML 语句在当前事务的数据库中执行时，才有效。

例如，考虑一个分布式事务，它从一个节点的 emp 表中移动一条职工的记录到另一个节点的 emp 表中。该事务能够保护这条记录 —— 即使管理员在每个节点上独自地强行恢复悬而未决的事务 —— 通过包括如下 SQL 语句的序列：

```
ALTER SESSION ADVISE COMMIT;
INSERT INTO emp@hq ... ; /*advice to commit at HQ */
ALTER SESSION ADVISE ROLLBACK;
DELETE FROM emp@sales ... ; /*advice to roll back at SALES*/
ALTER SESSION ADVISE NOTHING;
```

如果你按照给出的建议手动地强行恢复悬而未决的事务，可能发生的最坏结果就是，每个节点都有一个职工记录的备份，但记录不会被丢失。

32.4 手动覆盖悬而未决的事务

使用带有 FORCE 选项的、表明悬而未决事务的本地或全局事务 ID 字符串的、COMMIT 或 ROLLBACK 语句来提交。

注意：在所有的例子中，事务都是在本地节点上提交或回滚的，并且本地悬而未决的事务表，为该事务的行的 STATE 列，记录了表明被强行提交或被强行中断的值。

本节包括以下内容：

- 手动提交一个悬而未决的事务
- 手动回滚一个悬而未决的事务

32.4.1 手动提交一个悬而未决的事务

当你尝试提交事务之前，确信你拥有合适的权限。注意以下要求：

事务的提交者	你必须拥有的权限
你	FORCE TRANSACTION
其他用户	FORCE ANY TRANSACTION

32.4.1.1 只使用事务 ID 来提交

如下 SQL 语句提交一个悬而未决的事务：

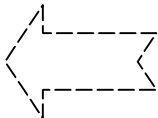
```
COMMIT FORCE 'transaction_id';
```

变量 transaction_id 是该事务的标识，在 DBA_2PC_PENDING 数据字典视图的 LOCAL_TRAN_ID 列或 GLOBAL_TRAN_ID 列中指定。

例如，假设查询了 DBA_2PC_PENDING，确定一个分布式事务的本地事务 ID 为：

```
LOCAL_TRAN_ID 1.45.13
```

然后，发布如下 SQL 语句，来强行提交这个悬而未决的事务：




```
COMMIT FORCE '1.45.13';
```

32.4.1.2 使用一个 SCN 来提交

可选地，当强行提交一个事务时，可以指定该事务的 SCN。当一个悬而未决的事务是在其他节点上被提交时，这项功能允许使用这个被赋予的 SCN 来提交该事务。

其结果是，即使存在故障，这个分布式事务的同步提交时间仍然被维持了。只有当你能够确定，已经在其他节点上提交过相同事务的 SCN 时，才指定一个 SCN。

例如，假设想手动提交一个具有如下全局事务 ID 号的事务：

```
SALES.ACME.COM.55d1c563.1.93.29
```

首先，查询同样包括在问题中的该事务的远程数据库的 DBA_2PC_PENDING 视图。注意那个节点提交该事务时所使用的 SCN。当在本地节点提交该事务时，就指定这个 SCN。例如，如果 SCN 为 829381993，就发布：

```
COMMIT FORCE 'SALES.ACME.COM.55d1c563.1.93.29', 829381993;
```

32.4.2 手动回滚一个悬而未决的事务

当你尝试回滚悬而未决的分布式事务之前，确信你拥有正确的权利。注意以下要求：

事务的提交者	你必须拥有的权限
你	FORCE TRANSACTION
其他用户	FORCE ANY TRANSACTION

如下 SQL 语句回滚一个悬而未决的事务：

```
ROLLBACK FORCE 'transaction_id';
```

变量 *transaction_id* 是该事务的标识，在 DBA_2PC_PENDING 数据字典视图的 LOCAL_TRAN_ID 列或 GLOBAL_TRAN_ID 列中指定。

例如，要想回滚本地事务 ID 为 2.9.4 的悬而未决的事务，就使用如下语句：

```
ROLLBACK FORCE '2.9.4';
```

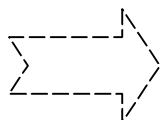
注意：不能将一个悬而未决的事务回滚到储存点位置。

32.5 清除数据字典中悬而未决的行

在 RECO 恢复一个悬而未决的事务之前，该事务会以 COLLECTING, COMMITTED 或 PREPARED，出现在 DBA_2PC_PENDING.STATE 中。如果使用 COMMIT FORCE 或 ROLLBACK FORCE 来强行恢复一个悬而未决的事务，那么就可能会出现 FORCED COMMIT 或 FORCED ROLLBACK 状态。

通常，自动恢复会删除处于这些状态的条目。唯一的例外是当恢复发现一个被强行的事务处于与该事务其他站点不一致的状态时。在这种情况下，这个条目将被保留在表中，DBA_2PC_PENDING 表的 MIXED 列的值被设置成 YES。可以用 DBMS_TRANSACTION.PURGE_MIXED 过程来清除这些条目。

如果一个远程数据库已经被永远的丢失了，恢复就不能识别重建的数据库了，这是因为重建它时，它得到了一个新的数据库 ID，所以就不可能进行自动恢复了。在这种情况下，



必须使用 DBMS_TRANSACTION 包中的 PURGE_LOST_DB_ENTRY 过程来清除这些条目。因为这些条目并不占用数据库资源，所以不用着急清除它们。

32.5.1 执行 PURGE_LOST_DB_ENTRY 过程

为了从数据字典中手动删除一个项目，使用如下语法（其中 *trans_id* 是事务的标识）：

```
DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('trans_id');
```

例如，要清除悬而未决的分布式事务 1.44.99，就在 SQL*Plus 中输入如下语句：

```
EXECUTE DBMS_TRANSACTION.PURGE_LOST_DB_ENTRY('1.44.99');
```

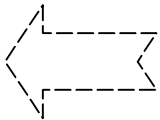
只有当进行了重大的重新配置，使得自动恢复不能解决事务时，才执行这个过程。范例包括：

- 丢失了全部的远程数据库
- 重新配置了软件，导致失去了两阶段提交的能力
- 丢失了类似 TPMonitor 的外部事务协调器的信息

32.5.2 确定何时使用 DBMS_TRANSACTION

下面的表显示分布式事务有哪些不同的状态，管理员应该进行哪些操作：

STATE 列	全局事务 的状态	本地事务 的状态	通常的动作	可选择的动作
Collecting	Rolled back	Rolled back	无	PURGE_LOST_DB_ENTRY （只有当自动恢复不能解决事务时）
Committed	Committed	Committed	无	PURGE_LOST_DB_ENTRY （只有当自动恢复不能解决事务时）
Prepared	Unknown	Prepared	无	强行提交或回滚
Forced commit	Unknown	Committed	无	PURGE_LOST_DB_ENTRY （只有当自动恢复不能解决事务时）
Forced rollback	Unknown	Rolled back	无	PURGE_LOST_DB_ENTRY （只有当自动恢复不能解决事务时）
Forced commit	Mixed	Committed	手动删除不一致性，然后使用 PURGE_MIXED	



(续表)

STATE 列	全局事务的状态	本地事务的状态	通常的动作	可选择的动作
Forced rollback	Mixed	Rolled back	手动删除不一致性，然后使用 PURGE_MIXED	

32.6 手动提交一个悬而未决的事务：例子

下面的例子，如图 32-1 所示，显示了一个在分布式事务提交时发生的故障。它说明了在手动强行提交或回滚一个悬而未决的分布式事务的本地部分之前，怎样着手获得信息。

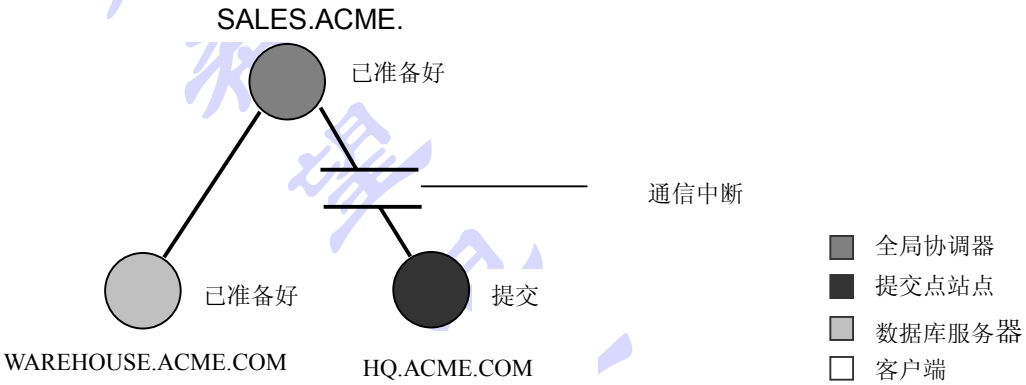


图 32-1 一个悬而未决的分布式事务的例子

在这个失败的案例中，准备阶段已经完成。然而，在提交阶段，尽管提交点站点已经提交了该事务，但提交点站点的提交确认还没有到达全局协调器。

假设你是 warehouse 数据库的管理员。因为悬而未决的事务比其他事务关键，所以货存数据被加锁。但是，因为这些锁必须保持到提交或回滚悬而未决的事务，所以就不能访问这些数据了。而且，你了解到销售部和总部之间的通信链接不能被马上解决。

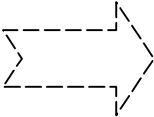
因此决定使用以下步骤，手动强行恢复悬而未决事务的本地部分：

- ①用户的反馈。
- ②本地的 DBA_2PC_PENDING 视图，获得全局事务 ID 号，以及有关该悬而未决事务的其他信息。
- ③本地的 DBA_2PC_NEIGHBORS 视图，开始跟踪会话树，以便能够找到已经解决了悬而未决事务的节点。
- ④新建立了正常的通信之后，检查混合结果标志。

下面几节详细说明这个例子的每一个步骤：

步骤 1：记录用户的反馈

步骤 2：查询 DBA_2PC_PENDING



步骤 3: 查询本地节点的 DBA_2PC_NEIGHBORS

步骤 4: 查询所有节点的数据字典视图

步骤 5: 提交悬而未决的事务

步骤 6: 利用 DBA_2PC_PENDING 检查混合结果

32.6.1 步骤 1: 记录用户的反馈

与悬而未决事务的锁产生冲突的本地数据库系统的用户，会接收到以下错误消息：

```
ORA-01591: lock held by in-doubt distributed transaction 1.21.17
```

在这种情况下，1.21.17 是该悬而未决分布式事务的本地事务 ID 号。你应该从报告问题的用户那里寻求和记录这个 ID 号，以便标识应该强行恢复哪个悬而未决事务。

32.6.2 步骤 2: 查询 DBA_2PC_PENDING

在将 SQL*Plus 与 warehouse 相连接之后，查询本地的 DBA_2PC_PENDING 数据字典视图，以获得有关该悬而未决事务的信息：

```
CONNECT SYS/password@warehouse.acme.com
```

```
SELECT * FROM DBA_2PC_PENDING WHERE LOCAL_TRAN_ID = '1.21.17';
```

Oracle 返回以下信息：

```
Column Name Value
```

```
-----
```

```
LOCAL_TRAN_ID 1.21.17
```

```
GLOBAL_TRAN_ID SALES.ACME.COM.55d1c563.1.93.29
```

```
STATE prepared
```

```
MIXED no
```

```
ADVICE
```

```
TRAN_COMMENT Sales/New Order/Trans_type 10B
```

```
FAIL_TIME 31-MAY-91
```

```
FORCE_TIME
```

```
RETRY_TIME 31-MAY-91
```

```
OS_USER SWILLIAMS
```

```
OS_TERMINAL TWA139:
```

```
HOST system1
```

```
DB_USER SWILLIAMS
```

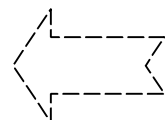
```
COMMIT#
```

32.6.2.1 确定全局事务 ID

全局事务 ID 是一个公共的事务 ID，对于一个分布式事务来说，它在每个节点上都是相同的，它具有以下格式：

```
global_database_name.hhhhhhhh.local_transaction_id
```

其中：



`global_database_name` 是全局协调器的数据库名称
`hhhhhhhh` 是全局协调器的内部数据库标识（十六进制）

`local_transaction_id` 赋予全局协调器的相应的本地事务 ID

注意，在全局协调器上，全局事务 ID 的最后部分与本地事务 ID 相匹配。在这个例子中，因为这些号码不相匹配，所以就能知道 warehouse 不是全局协调器：

```
LOCAL_TRAN_ID 1.21.17
GLOBAL_TRAN_ID ... 1.93.29
```

32.6.2.2 确定事务的状态

这个节点上的事务处于已准备好的状态：

```
STATE prepared (已准备好)
```

因此，warehouse 等待它的协调器送来一个提交或回滚请求。

32.6.2.3 寻找注释或建议

事务的注释或建议能包含有关该事务信息。如果真的如此，就要利用这些注释。在这个例子中，事务的来源和类型都在该事务的注释中：

```
TRAN_COMMENT Sales/New Order/Trans_type 10B
```

可以用 `SET TRANSACTION ... NAME` 语句，用一个事务名称来提供这些信息。

这些信息能够揭示一些内容，来帮助我们决定是提交还是回滚该事务的本地部分。如果有用的注释并没有和该悬而未决的事务一起存在，那你就必须完成一些额外的管理工作，才能跟踪会话树并找到一个已经解决该事务的节点。

32.6.3 步骤 3：查询本地节点上的 DBA_2PC_NEIGHBORS

这一步的目的是顺着会话树，以便发现协调器，最终达到全局协调器。按照这个方法，也许能发现一个已经解决了该事务的协调器。如果不能的话，最终也能照这种方法，到达提交点站点，它们总是已经解决了该悬而未决的事务。要想跟踪会话树，就要查询每个节点的 `DBA_2PC_NEIGHBORS` 视图。

在这个例子中，查询 warehouse 数据库上的这个视图：

```
CONNECT SYS/password@warehouse.acme.com
```

```
SELECT * FROM DBA_2PC_NEIGHBORS
```

```
WHERE LOCAL_TRAN_ID = '1.21.17'
```

```
ORDER BY SESS#, IN_OUT;
```

```
Column Name Value
```

```
-----
```

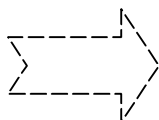
```
LOCAL_TRAN_ID 1.21.17
```

```
IN_OUT in
```

```
DATABASE SALES.ACME.COM
```

```
DBUSER_OWNER SWILLIAMS
```

```
INTERFACE N
```



```
DBID 000003F4
SESS# 1
BRANCH 0100
```

32.6.3.1 获得数据库角色和数据库链接的信息

DBA_2PC_NEIGHBORS 视图提供了与悬而未决事务相关的连接的信息。每个连接的信息都是不同的，这取决于该连接是 inbound (IN_OUT = in) 还是 outbound (IN_OUT = out):

IN_OUT	意义	DATABASE	DBUSER_OWNER
in	你的节点是另一个节点的服务器。	列出与你的节点相连接的客户数据库的名称。	列出用于数据库链接连接的本地帐户。 (对应于该悬而未决事务)
out	你的节点是其他服务器的一个客户。	列出与远程节点相连接的数据库链接的名称。	列出用于悬而未决事务的数据库链接的拥有者。

在这个例子中，IN_OUT 列表明，warehouse 数据库是针对 sales 客户的服务器，就像在 DATABASE 列中所指出的那样：

```
IN_OUT in
DATABASE SALES.ACME.COM
```

到 warehouse 的连接是通过来自 swilliams 帐户的数据库链接而建立起来，就像 DBUSER_OWNER 列所显示的那样：

```
DBUSER_OWNER SWILLIAMS
```

32.6.3.2 确定提交点站点

此外，INTERFACE 列说明，是否本地节点或下级从属节点是提交点站点：

```
INTERFACE N
```

就像 INTERFACE 列所示的那样，warehouse 和它的任何后代都不是提交点站点。

32.6.4 步骤 4: 查询所有节点的数据字典视图

在这一步中，可以与固定节点的管理员相联系，请求每个人都用全局事务 ID 来重复步骤 2 和步骤 3。

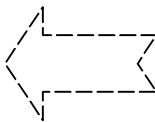
注意：如果能够通过另一个网络直接连接到的这些节点，就可以自己重复步骤 2 和步骤 3 了。

例如，下面是在 sales 和 hq 上执行步骤 2 和步骤 3 后返回的结果。

32.6.4.1 检查 sales 上的悬而未决事务的状态

在这个阶段，sales 的管理员查询 DBA_2PC_PENDING 数据字典视图：

```
SQL> CONNECT SYS/password@sales.acme.com
SQL> SELECT * FROM DBA_2PC_PENDING
> WHERE GLOBAL_TRAN_ID = 'SALES.ACME.COM.55d1c563.1.93.29';
Column Name Value
```



```
-----  
LOCAL_TRAN_ID 1.93.29  
GLOBAL_TRAN_ID SALES.ACME.COM.55d1c563.1.93.29  
STATE prepared  
MIXED no  
ADVICE  
TRAN_COMMENT Sales/New Order/Trans_type 10B  
FAIL_TIME 31-MAY-91  
FORCE_TIME  
RETRY_TIME 31-MAY-91  
OS_USER SWILLIAMS  
OS_TERMINAL TWA139:  
HOST system1  
DB_USER SWILLIAMS  
COMMIT#
```

32.6.4.2 确定 sales 的协调器和提交点站点

接着, sales 的管理员查询 DBA_2PC_NEIGHBORS, 以确定全局协调器和本地协调器, 以及提交点站点:

```
SELECT * FROM DBA_2PC_NEIGHBORS  
WHERE GLOBAL_TRAN_ID = 'SALES.ACME.COM.55d1c563.1.93.29'  
ORDER BY SESS#, IN_OUT;
```

这个查询返回 3 行:

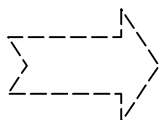
- 与 warehouse 的连接
- 与 hq 的连接
- 该用户自己建立的连接

与 warehouse 的连接相对应的行的, 被重新格式化后的信息显示如下:

```
Column Name Value  
-----  
LOCAL_TRAN_ID 1.93.29  
IN_OUT OUT  
DATABASE WAREHOUSE.ACME.COM  
DBUSER_OWNER SWILLIAMS  
INTERFACE N  
DBID 55d1c563  
SESS# 1  
BRANCH 1
```

与 hq 的连接相对应的行的, 被重新格式化后的信息显示如下:

```
Column Name Value
```



```

-----
LOCAL_TRAN_ID 1.93.29
IN_OUT OUT
DATABASE HQ.ACME.COM
DBUSER_OWNER ALLEN
INTERFACE C
DBID 00000390
SESS# 1
BRANCH 1

```

以上查询出的信息表明：

- 因为本地事务 ID 与全局事务 ID 相匹配，所以 sales 是全局协调器。
- 这个节点建立了 2 个向外的连接，但是没有向内的连接。因此 sales 不是其他节点的服务器。
- hq 或它的一个服务器是提交点站点。

32.6.4.3 检查 HQ 上的悬而未决事务的状态

在这个阶段，hq 的管理员查询 DBA_2PC_PENDING 数据字典视图：

```

SELECT * FROM DBA_2PC_PENDING@hq.acme.com
WHERE GLOBAL_TRAN_ID = 'SALES.ACME.COM.55d1c563.1.93.29';
Column Name Value

```

```

-----
LOCAL_TRAN_ID 1.45.13
GLOBAL_TRAN_ID SALES.ACME.COM.55d1c563.1.93.29
STATE COMMIT
MIXED NO
ACTION
TRAN_COMMENT Sales/New Order/Trans_type 10B
FAIL_TIME 31-MAY-91
FORCE_TIME
RETRY_TIME 31-MAY-91
OS_USER SWILLIAMS
OS_TERMINAL TWA139:
HOST SYSTEM1
DB_USER SWILLIAMS
COMMIT# 129314

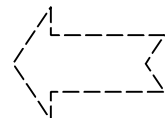
```

在这里，已经发现了一个已经解决了该事务的节点。正如该视图所揭示的那样，该事务已经被提交，并且被赋予了一个提交 ID 号：

```

STATE COMMIT
COMMIT# 129314

```



因此，你可以在你的本地数据库上，强行提交该悬而未决的事务。与你所知道的其他管理员相联系，让他们也从你的研究中获益，这是一个好主意。

32.6.5 步骤 5：提交悬而未决定事务

与 sales 数据库的管理员相联系，他用全局 ID 号，手动提交该悬而未决的事务：

```
SQL> CONNECT SYS/password@sales.acme.com
```

```
SQL> COMMIT FORCE 'SALES.ACME.COM.55d1c563.1.93.29';
```

和 warehouse 数据库的管理员一样，你也使用这个全局 ID 号，手动提交这个悬而未决的事务：

```
SQL> CONNECT SYS/password@warehouse.acme.com
```

```
SQL> COMMIT FORCE 'SALES.ACME.COM.55d1c563.1.93.29';
```

32.6.6 步骤 6：利用 DBA_2PC_PENDING 检查混合结果

当手动强行提交或回滚一个事务之后，在悬而未决事务表中的相应的记录仍然存在。该事务的状态的改变取决于怎样强行恢复该事务。

每一个 Oracle 数据库都有一个悬而未决事务表。这是一个特殊的表，当分布式事务经过两阶段提交的各个阶段时，它存储着有关该分布式事务的信息。可以通过 DBA_2PC_PENDING 数据字典视图来查询一个数据库的悬而未决事务表。（见表 32-1 “DBA_2PC_PENDING”）。

在悬而未决事务表中，特别值得关注的是，在 DBA_2PC_PENDING.MIXED 中所显示的混合结果标志。如果一个悬而未决事务被强行提交或回滚，就可能做出错误的选择。例如，本地管理员回滚该事务，但其他节点却提交该事务。错误的决定将被自动探测到，针对相应的悬而未决事务的记录的破坏标志被设置成（MIXED = yes）。

RECO（恢复器）后台程序使用在悬而未决事务表中的信息，来最后确定悬而未决事务的状态。也可以使用悬而未决事务表中的信息，来手动覆盖悬而未决分布式事务的自动恢复过程。

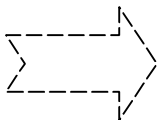
由 RECO 自动解决的所有事务，将从悬而未决事务表中删除。此外，被管理员正确解决的有关悬而未决事务的所有信息（正如 RECO 重新建立通信时所检查到的那样），也将从悬而未决事务表中自动删除。但是，由管理员解决的，在节点之间造成混合结果的所有行，将保留在所包含的所有相关节点的悬而未决事务表中，直到用 DBMS_TRANSACTION.PURGE_MIXED 手动将它们删除。

32.7 由于加锁而造成的数据访问失败

当发布一条 SQL 语句时，Oracle 就会给要成功执行这个语句所需要的资源加锁。如果所需的数据目前正被其他未提交事务的语句所占据，而且长时间保持着加锁，那么就会出现超时。

考虑如下造成数据访问失败的情况：

- 事务超时



- 锁使你陷于悬而未决的事务

32.7.1 事务超时

如果另一个事务自己已经锁住了所需要的数据，则一个请求远程数据库锁的 DML 语句可能被冻结。如果这些锁继续冻结该请求的 SQL 语句，那么，将会发生下列事件序列：

- ①产生一个超时。
- ②Oracle 回滚该语句。
- ③Oracle 给用户返回这个错误信息：

ORA-02049: time-out: distributed transaction waiting for lock

因为该事务还没修改数据，所以超时之后，不需要做任何操作。如果遇到了死锁情况，应用程序应该继续运行。执行该语句的用户随后可以尝试重新执行该语句。如果锁仍然存在，那么该用户就应该向管理员报告这个问题。

32.7.2 锁使你陷于悬而未决的事务

一个请求本地数据库锁的查询或 DML 语句，可能会不确定地由于一个悬而未决分布式事务锁住了资源而被冻结。在这种情况下，Oracle 发布如下错误信息：

ORA-01591: lock held by in-doubt distributed transaction identifier

在这种情况下，Oracle 立即回滚该 SQL 语句。执行该语句的用户稍后可以试着再重新执行该语句。如果锁仍然存在，那么，该用户就应该向管理员报告这个问题，包括这个悬而未决分布式事务的 ID 号。

发生上面情况的机会非常少，因为在两阶段提交期间的关键部分失败的可能性极小。就算发生了这样一个失败，并假设网络和系统失败迅速得到恢复，这些问题就将会自动解决，而不需要进行手动干预。因此，问题通常在他们被用户或数据库管理员探测到以前就已经被解决了。

32.8 模拟分布式事务失败

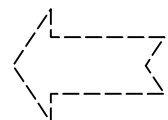
因为如下原因，可能实施分布式事务的失败：

- 为了观察 RECO 自动解决该事务的本地部分
- 为了实践手动解决悬而未决分布式事务，并观察结果

一个 Oracle 实例的 RECO 后台进程，自动地解决涉及分布式事务的失败。一个节点的 RECO 后台进程，尝试恢复一个悬而未决分布式事务的本地部分的时间间隔，呈指数级增长。

RECO 可以使用一个现存的连接，或创建一个新的连接，以便连接到已失败的事务所包含的其他节点上。当一个连接被建立时，RECO 就自动地解决所有悬而未决的事务。对应于任何已被解决的悬而未决事务的行，将自动地从每个数据库的悬而未决事务表中删除。

可以用带有 ENABLE/DISABLE DISTRIBUTED RECOVERY 选项的 ALTER SYSTEM 语句，来启用或停用 RECO。例如，可以临时性地不让 RECO 实施处理两阶段提交的失败，而手动解决该悬而未决事务。



如下语句停用 RECO:

```
ALTER SYSTEM DISABLE DISTRIBUTED RECOVERY;
```

相反, 如下语句启用 RECO, 以便悬而未决事务能被自动解决:

```
ALTER SYSTEM ENABLE DISTRIBUTED RECOVERY;
```

注意: 单进程实例 (例如, 一台 PC 机运行 MS-DOS) 没有单独的后台进程, 因此没有 RECO 进程。因此, 当参与分布式系统的一个单进程实例开始运行时, 就必须使用上面的语句手动启用分布式恢复。

32.9 管理读一致性

在 Oracle 的分布式读一致性的实现中, 存在着一个重要的限制。这个问题的发生是因为每个系统都有它自己的 SCN, 你可以将他们当作数据库的内部时间戳。Oracle 数据库服务器使用 SCN, 来决定一个查询返回哪个版本的数据。

一个分布式事务的各个 SCN, 在每个远程 SQL 语句结束时, 在每个事务的开始和结束时, 都是同步的。在有着大量的通信, 特别是有着大量的分布式更新的两个节点之间, 该同步是频繁发生的。然而, 不存在一个实用的方法, 能使一个分布式系统的各个 SCN 保持绝对的同步: 总是存在这样一个窗口, 在这个窗口中, 一个节点可能有一个 SCN, 它相对于另一个节点的 SCN 来说, 有点过时了。

因为 SCN 的间隙, 可能会执行一个使用稍旧了点的快照的查询, 所以看不到远程数据库上的最新的改变。依照读一致性, 一个查询可以因此检索到一致的、但过时的数据。注意, 该查询检索到的所有数据将来自于旧的 SCN, 所以, 如果一个在本地执行的更改事务更改一个远程节点上的两个表时, 那么在下一个远程访问中, 从这两个表中选取到的数据, 可能会包含更改之前的数据。

SCN 的间隙的一个后果是, 两个相继的 SELECT 语句可能检索到了不同的数据, 即使在这两条语句之间没有执行过 DML。例如, 可以发布一条更改语句, 然后在远程数据库上提交该更改语句。当在基于该远程表的视图上, 发布一条 SELECT 语句时, 该视图没有显示对行的更改情况。下一次再发布该 SELECT 语句时, 更改情况出现了。

可以使用下面的技术来保证在一个查询之前, 两台机器的 SCN 是同步的:

- 因为各个 SCN 在一个远程查询结束时是同步的, 所以在每个远程查询之前, 先发布一条针对同一站点的虚拟远程查询, 例如,

```
SELECT * FROM DUAL@REMOTE.
```

- 因为各个 SCN 在每个远程事务的开始时是同步的, 所以在发布远程查询之前, 先提交或回滚当前的事务。

