Contents

Preface

- Audience
- Documentation Accessibility
- Related Documentation
- Conventions

1 Introduction to Oracle Database

- About Relational Databases
  - Database Management System (DBMS)
  - Relational Model
  - Relational Database Management System (RDBMS)
  - Brief History of Oracle Database
- Schema Objects
  - Tables
  - Indexes
- Data Access
  - Structured Query Language (SQL)
  - PL/SQL and Java
- Transaction Management
  - Transactions
  - Data Concurrency
  - Data Consistency
- Oracle Database Architecture
  - Database and Instance
  - Database Storage Structures
  - Database Instance Structures
  - Application and Networking Architecture
- Oracle Database Documentation Roadmap
  - Basic Group
  - Intermediate Group

目录

序言

- 听众
- 文档可访问性
- 相关文档
- 约定

第 1 章 Oracle 数据库简介

- 关于关系数据库
  - 数据库管理系统(DBMS)
  - 关系模型
  - 关系数据库管理系统(RDBMS)
  - Oracle 数据库发展简史
- 模式对象
  - 表
  - 索引
- 数据访问
  - 结构化查询语言(SQL)
  - PL/SQL 和 Java
- 事务管理
  - 事务
  - 数据并发性
  - 数据一致性
- Oracle 数据库体系结构
  - 数据库和实例
  - 数据库存储结构
  - 数据库实例结构
  - 应用程序和网络体系结构
- Oracle 数据库文档路线图
  - 基本组
  - 中级组

- Advanced Group

Part I Oracle Relational Data Structures

2 Tables and Table Clusters

- Introduction to Schema Objects
  - Schema Object Types
  - Schema Object Storage
  - Schema Object Dependencies
  - SYS and SYSTEM Schemas
  - Sample Schemas
- Overview of Tables
  - Columns and Rows
  - Example: CREATE TABLE and ALTER TABLE Statements
  - Oracle Data Types
  - Integrity Constraints
  - Object Tables
  - Temporary Tables
  - External Tables
  - Table Storage
- Overview of Table Clusters
  - Overview of Indexed Clusters
  - Overview of Hash Clusters

3 Indexes and Index-Organized Tables

- Overview of Indexes
  - Index Characteristics
  - B-Tree Indexes
  - Bitmap Indexes
  - Function-Based Indexes
  - Application Domain Indexes
  - Index Storage
- Overview of Index-Organized Tables
  - Index-Organized Table Characteristics
  - Index-Organized Tables with Row Overflow Area

- 高级组

第一部分 Oracle 关系数据结构

第 2 章 表和表簇

- 模式对象简介
  - 模式对象类型
  - 模式对象存储
  - 模式对象依赖
  - SYS 和 SYSTEM 模式
  - 示例模式
- 表概述
  - 列和行
  - 示例：CREATE TABLE 和 ALTER TABLE 语句
  - Oracle 数据类型
  - 完整性约束
  - 对象表
  - 临时表
  - 外部表
  - 表存储
- 表簇概述
  - 索引化表簇概述
  - 哈希簇概述

第 3 章 索引和索引组织表

- 索引概述
  - 索引特征
  - B-树索引
  - 位图索引
  - 基于函数的索引
  - 应用程序域索引
  - 索引存储
- 索引组织表概述
  - 索引组织表特征
  - 索引组织表的行溢出区

- Secondary Indexes on Index-Organized Tables

4 Partitions, Views, and Other Schema Objects

- Overview of Partitions
  - Partition Characteristics
  - Partitioned Tables
  - Partitioned Indexes
  - Partitioned Index-Organized Tables
- Overview of Views
  - Characteristics of Views
  - Updatable Join Views
  - Object Views
- Overview of Materialized Views
  - Characteristics of Materialized Views
  - Refresh Methods for Materialized Views
  - Query Rewrite
- Overview of Sequences
  - Sequence Characteristics
  - Concurrent Access to Sequences
- Overview of Dimensions
  - Hierarchical Structure of a Dimension
  - Creation of Dimensions
- Overview of Synonyms

5 Data Integrity

- Introduction to Data Integrity
  - Techniques for Guaranteeing Data Integrity
  - Advantages of Integrity Constraints
- Types of Integrity Constraints
  - NOT NULL Integrity Constraints
  - Unique Constraints
  - Primary Key Constraints
  - Foreign Key Constraints
  - Check Constraints
- States of Integrity Constraints

- 索引组织表的辅助索引

第 4 章 分区、 视图和其它模式对象

- 分区概述
  - 分区特征
  - 分区表
  - 分区索引
  - 分区索引组织表
- 视图概述
  - 视图特征
  - 可更新联合视图
  - 对象视图
- 物化视图概述
  - 物化视图的特征
  - 物化视图的刷新方法
  - 查询重写
- 序列概述
  - 序列概述
  - 对序列的并发访问
- 维度概述
  - 维度的层次结构
  - 创建维度
- 同义词概述

第 5 章 数据完整性

- 数据完整性介绍
  - 用于保证数据完整性的技术
  - 完整性约束的优势
- 完整性约束的类型
  - 非空完整性约束
  - 唯一性约束
  - 主键约束
  - 外键约束
  - 检查性约束
- 完整性约束的状态

- Checks for Modified and Existing Data
- Deferrable Constraints
- Examples of Constraint Checking

6 Data Dictionary and Dynamic Performance Views

- Overview of the Data Dictionary
  - Contents of the Data Dictionary
  - Storage of the Data Dictionary
  - How Oracle Database Uses the Data Dictionary
- Overview of the Dynamic Performance Views
  - Contents of the Dynamic Performance Views
  - Storage of the Dynamic Performance Views
- Database Object Metadata

Part II Oracle Data Access

7 SQL

- Introduction to SQL
  - SQL Data Access
  - SQL Standards
- Overview of SQL Statements
  - Data Definition Language (DDL) Statements
  - Data Manipulation Language (DML) Statements
  - Transaction Control Statements
  - Session Control Statements
  - System Control Statement
  - Embedded SQL Statements
- Overview of the Optimizer
  - Use of the Optimizer
  - Optimizer Components
  - Access Paths
  - Optimizer Statistics
  - Optimizer Hints
- Overview of SQL Processing
  - Stages of SQL Processing
  - How Oracle Database Processes DML

- 对现有数据和新数据的检查
- 可延迟约束
- 约束检查的示例

第 6 章 数据字典和动态性能视图

- 数据字典概述
  - 数据字典的内容
  - 数据字典存储
  - Oracle 数据库如何使用数据字典
- 动态性能视图概述
  - 动态性能视图的内容
  - 动态性能视图的存储
- 数据库对象元数据

第二部分 Oracle 数据访问

第 7 章 SQL

- SQL 简介
  - SQL 数据访问
  - SQL 标准
- SQL 语句概述
  - 数据定义语言 (DDL) 语句
  - 数据操作语言 (DML) 语句
  - 事务控制语句
  - 会话控制语句
  - 系统控制语句
  - 嵌入式的 SQL 语句
- 优化器概述
  - 使用优化器
  - 优化器组件
  - 访问路径
  - 优化器统计
  - 优化器提示
- SQL 处理的概述
  - SQL 处理的阶段

- How Oracle Database Processes DDL

8 Server-Side Programming: PL/SQL and Java

- Introduction to Server-Side Programming
- Overview of PL/SQL
  - PL/SQL Subprograms
  - PL/SQL Packages
  - PL/SQL Anonymous Blocks
  - PL/SQL Language Constructs
  - PL/SQL Collections and Records
  - How PL/SQL Runs
- Overview of Java in Oracle Database
  - Overview of the Java Virtual Machine (JVM)
  - Java Programming Environment
- Overview of Triggers
  - Advantages of Triggers
  - Types of Triggers
  - Timing for Triggers
  - Creation of Triggers
  - Execution of Triggers
  - Storage of Triggers

Part III Oracle Transaction Management
9 Data Concurrency and Consistency

- Introduction to Data Concurrency and Consistency
  - Multiversion Read Consistency
  - Locking Mechanisms
  - ANSI/ISO Transaction Isolation Levels
- Overview of Oracle Database Transaction Isolation Levels
  - Read Committed Isolation Level
  - Serializable Isolation Level
  - Read-Only Isolation Level
- Overview of the Oracle Database Locking Mechanism
  - Summary of Locking Behavior

- Oracle 数据库如何处理 DML
- Oracle 数据库如何处理 DDL

第 8 章 服务器端编程：PL/SQL 和 Java

- 服务器端编程简介
- PL/SQL 概述
  - PL/SQL 子程序
  - PL/SQL 包
  - PL/SQL 匿名块
  - PL/SQL 语言结构
  - PL/SQL 集合和记录
  - PL/SQL 如何运行
- Oracle 数据库中的 Java 概述
  - Java 虚拟机 (JVM) 的概述
  - Java 编程环境
- 触发器概述
  - 触发器的优势
  - 触发器类型
  - 触发时间
  - 创建触发器
  - 执行触发器
  - 触发器存储

第三部分 Oracle 事务管理
第 9 章 数据并发性和一致性

- 数据并发和一致性介绍
  - 多版本读一致性
  - 锁定机制
  - ANSI/ISO 事务隔离级别
- Oracle 数据库事务隔离级别概述
  - 读提交隔离级别
  - 可串行化隔离级别
  - 只读隔离级别
- 数据库锁定机制概述
  - 锁定行为总结

- Use of Locks
- Lock Modes
- Lock Conversion and Escalation
- Lock Duration
- Locks and Deadlocks
- Overview of Automatic Locks
  - DML Locks
  - DDL Locks
  - System Locks
- Overview of Manual Data Locks
- Overview of User-Defined Locks

10 Transactions

- Introduction to Transactions
  - Sample Transaction: Account Debit and Credit
  - Structure of a Transaction
  - Statement-Level Atomicity
  - System Change Numbers (SCNs)
- Overview of Transaction Control
  - Transaction Names
  - Active Transactions
  - Savepoints
  - Rollback of Transactions
  - Committing Transactions
- Overview of Autonomous Transactions
- Overview of Distributed Transactions
  - Two-Phase Commit
  - In-Doubt Transactions

Part IV Oracle Database Storage Structures
11 Physical Storage Structures

- Introduction to Physical Storage Structures
  - Mechanisms for Storing Database Files
  - Oracle Automatic Storage Management (Oracle ASM)
  - Oracle Managed Files and User-Managed Files

- 使用锁
- 锁模式
- 锁转换和锁升级
- 锁持续时间
- 锁和死锁
- 自动锁的概述
  - DML 锁
  - DDL 锁
  - 系统锁
- 手动数据锁概述
- 用户定义的锁的概述

第 10 章 事务

- 事务介绍
  - 示例事务： 帐户借贷
  - 事务的结构
  - 语句级原子性
  - 系统更改号 （SCNs）
- 事务控制概述
  - 事务名
  - 活动事务
  - 保存点
  - 事务回滚
  - 事务提交
- 自治事务概述
- 分布式事务概述
  - 两阶段提交
  - 可疑事务

第四部分 Oracle 数据库存储结构
第 11 章 物理存储结构

- 物理存储结构简介
  - 用于存储数据库文件的机制
  - Oracle 自动存储管理 （Oracle ASM）
  - Oracle 管理的文件和用户管理的文件

- Overview of Data Files
  - Use of Data Files
  - Permanent and Temporary Data Files
  - Online and Offline Data Files
  - Data File Structure
- Overview of Control Files
  - Use of Control Files
  - Multiple Control Files
  - Control File Structure
- Overview of the Online Redo Log
  - Use of the Online Redo Log
  - How Oracle Database Writes to the Online Redo Log
  - Structure of the Online Redo Log

12 Logical Storage Structures

- Introduction to Logical Storage Structures
  - Logical Storage Hierarchy
  - Logical Space Management
- Overview of Data Blocks
  - Data Blocks and Operating System Blocks
  - Data Block Format
  - Data Block Compression
  - Space Management in Data Blocks
- Overview of Extents
  - Allocation of Extents
  - Deallocation of Extents
  - Storage Parameters for Extents
- Overview of Segments
  - User Segments
  - Temporary Segments
  - Undo Segments
  - Segment Space and the High Water Mark
- Overview of Tablespaces
  - Permanent Tablespaces
  - Temporary Tablespaces
  - Tablespace Modes

- 数据文件的概述
  - 使用数据文件
  - 永久和临时数据文件
  - 联机和脱机数据文件
  - 数据文件结构
- 控制文件的概述
  - 使用控制文件
  - 多个控制文件
  - 控制文件结构
- 联机重做日志概述
  - 使用联机重做日志
  - Oracle 数据库如何写入到联机重做日志
  - 联机重做日志的结构

第 12 章 逻辑存储结构

- 逻辑存储结构简介
  - 逻辑存储层次结构
  - 逻辑空间管理
- 数据块概述
  - 数据块和操作系统块
  - 数据块格式
  - 数据块压缩
  - 数据块的空间管理
- 扩展区概述
  - 分配扩展区
  - 释放扩展区
  - 扩展区的存储参数
- 段概述
  - 用户段
  - 临时段
  - 撤消段
  - 段空间和高水位标记
- 表空间概述
  - 永久表空间
  - 临时表空间
  - 表空间模式

- Tablespace File Size

Part V Oracle Instance Architecture
13 Oracle Database Instance

- Introduction to the Oracle Database Instance
  - Database Instance Structure
  - Database Instance Configurations
- Overview of Instance Startup and Shutdown
  - Overview of Instance and Database Startup
  - Overview of Database and Instance Shutdown
- Overview of Checkpoints
  - Purpose of Checkpoints
  - When Oracle Database Initiates Checkpoints
- Overview of Instance Recovery
  - Purpose of Instance Recovery
  - When Oracle Database Performs Instance Recovery
  - Importance of Checkpoints for Instance Recovery
  - Instance Recovery Phases
- Overview of Parameter Files
  - Initialization Parameters
  - Server Parameter Files
  - Text Initialization Parameter Files
  - Modification of Initialization Parameter Values
- Overview of Diagnostic Files
  - Automatic Diagnostic Repository
  - Alert Log
  - Trace Files

14 Memory Architecture

- Introduction to Oracle Database Memory Structures
  - Basic Memory Structures
  - Oracle Database Memory Management
- Overview of the User Global Area
- Overview of the Program Global Area
  - Contents of the PGA

- 表空间文件大小

第五部分 Oracle 实例体系结构
第 13 章 Oracle 数据库实例

- Oracle 数据库实例简介
  - Oracle 实例结构
  - 数据库实例配置
- 实例启动和关闭概述
  - 实例和数据库启动概述
  - 数据库和实例关闭概述
- 检查点概述
  - 检查点的目的
  - Oracle 数据库何时启动检查点
- 实例恢复概述
  - 实例恢复的目的
  - Oracle 数据库何时执行实例恢复
  - 实例恢复检查点的重要性
  - 实例恢复阶段
- 参数文件的概述
  - 初始化参数
  - 服务器参数文件
  - 文本初始化参数文件
  - 修改初始化参数值的值
- 诊断文件概述
  - 自动诊断存储库
  - 警报日志
  - 跟踪文件

第 14 章 内存体系结构

- Oracle 数据库内存结构简介
  - 基本内存结构
  - Oracle 数据库内存管理
- 用户全局区概述
- 程序全局区概述

- PGA Usage in Dedicated and Shared Server Modes
- Overview of the System Global Area
  - Database Buffer Cache
  - Redo Log Buffer
  - Shared Pool
  - Large Pool
  - Java Pool
  - Streams Pool
  - Fixed SGA
- Overview of Software Code Areas

15 Process Architecture

- Introduction to Processes
  - Multiple-Process Oracle Database Systems
  - Types of Processes
- Overview of Client Processes
  - Client and Server Processes
  - Connections and Sessions
- Overview of Server Processes
  - Dedicated Server Processes
  - Shared Server Processes
- Overview of Background Processes
  - Mandatory Background Processes
  - Optional Background Processes
  - Slave Processes

16 Application and Networking Architecture

- Overview of Oracle Application Architecture
  - Overview of Client/Server Architecture
  - Overview of Multitier Architecture
  - Overview of Grid Architecture
- Overview of Oracle Networking Architecture
  - How Oracle Net Services Works
  - The Oracle Net Listener

  - PGA 的内容
  - 在专用和共享服务器模式中使用 PGA
- 系统全局区概述
  - 数据库缓冲区高速缓存
  - 重做日志缓冲区
  - 共享池
  - 大池
  - Java 池
  - 流池
  - 固定 SGA
- 软件代码区概述

第 15 章 进程体系结构

- 进程简介
  - 多进程 Oracle 数据库系统
  - 进程类型
- 客户端进程概述
  - 客户端和服务器进程
  - 连接和会话
- 服务器进程概述
  - 专有服务器进程
  - 共享服务器进程
- 后台进程概述
  - 强制性后台进程
  - 可选后台进程
  - 从属进程

第 16 章 应用程序和网络体系结构

- Oracle 应用程序体系结构概述
  - 客户端/服务器体系结构概述
  - 多层体系结构概述
  - 网格体系结构概述
- Oracle 网络体系结构概述
  - Oracle 网络服务的工作原理
  - Oracle 网络监听器

- Dedicated Server Architecture
- Shared Server Architecture
- Database Resident Connection Pooling
- Overview of the Program Interface
  - Program Interface Structure
  - Program Interface Drivers
  - Communications Software for the Operating System

Part VI Oracle Database Administration and Development

17 Topics for Database Administrators and Developers

- Overview of Database Security
  - User Accounts
  - Authentication
  - Encryption
  - Access Control
  - Monitoring
- Overview of High Availability
  - High Availability and Unplanned Downtime
  - High Availability and Planned Downtime
- Overview of Grid Computing
  - Database Server Grid
  - Database Storage Grid
- Overview of Data Warehousing and Business Intelligence
  - Data Warehousing and OLTP
  - Data Warehouse Architecture
  - Overview of Extraction, Transformation, and Loading (ETL)
  - Business Intelligence
- Overview of Oracle Information Integration
  - Federated Access
  - Information Sharing

18 Concepts for Database Administrators

- Duties of Database Administrators
  - 专有服务器体系结构
  - 共享服务器体系结构
  - 数据库驻留连接池
- 程序接口概述
  - 程序接口结构
  - 程序接口驱动程序
  - 操作系统的通信软件

第六部分 Oracle 数据库管理与开发

第 17 章 数据库管理员和开发人员的主题

- 数据库安全性概述
  - 用户账户
  - 身份验证
  - 加密
  - 访问控制
  - 监视
- 高可用性的概述
  - 高可用性和计划外停机时间
  - 高可用性和计划内停机时间
- 网格计算的概述
  - 数据库服务器网格
  - 数据库存储网格
- 数据仓库和商业智能概述
  - 数据仓库和 OLTP
  - 数据仓库体系结构
  - 提取、 转换，和加载 (ETL) 概述
  - 商务智能
- Oracle 信息集成概述
  - 联邦访问
  - 信息共享

第 18 章 数据库管理员相关概念

- 数据库管理员的职责

- Tools for Database Administrators
  - Oracle Enterprise Manager
  - SQL*Plus
  - Tools for Database Installation and Configuration
  - Tools for Oracle Net Configuration and Administration
  - Tools for Data Movement and Analysis
- Topics for Database Administrators
  - Backup and Recovery
  - Memory Management
  - Resource Management and Task Scheduling
  - Performance Diagnostics and Tuning

19 Concepts for Database Developers

- Duties of Database Developers
- Tools for Database Developers
  - SQL Developer
  - Oracle Application Express
  - Oracle JDeveloper
  - Oracle JPublisher
  - Oracle Developer Tools for Visual Studio .NET
- Topics for Database Developers
  - Principles of Application Design and Tuning
  - Client-Side Database Programming
  - Globalization Support
  - Unstructured Data

Glossary
Index

- 数据库管理员的工具
  - Oracle 企业管理器
  - SQL*Plus
  - 数据库安装和配置工具
  - Oracle 网络配置和管理工具
  - 数据移动和分析工具
- 数据库管理员的主题
  - 备份和恢复
  - 内存管理
  - 资源管理与任务调度
  - 性能诊断和调优

第 19 章 数据库开发人员相关概念

- 数据库开发人员的职责
- 数据库开发人员的工具
  - SQL Developer
  - Oracle Application Express
  - Oracle JDeveloper
  - Oracle JPublisher
  - Oracle 开发工具 for Visual Studio.NET
- 数据库开发人员的主题
  - 应用程序设计和优化的原则
  - 客户端数据库编程
  - 全球化支持
  - 非结构化数据

术语表
索引

Preface

This manual provides an architectural and conceptual overview of the Oracle database server, which is an object-relational database management system. It describes how the Oracle database server functions, and it lays a conceptual foundation for much of the practical information contained in other manuals. Information in this manual applies to the Oracle database server running on all operating systems.

This preface contains these topics:

- Audience
- Documentation Accessibility
- Related Documentation
- Conventions

Audience

Oracle Database Concepts is intended for technical users, primarily database administrators and database application developers, who are new to Oracle Database. Typically, the reader of this manual has had experience managing or developing applications for other relational databases.

To use this manual, you must know the following:

- Relational database concepts in general
- Concepts and terminology in Chapter 1, "Introduction to Oracle Database"
- The operating system environment under which you are running Oracle

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors

序言

Oracle 数据库服务器是一个对象-关系型数据库管理系统, 本手册提供其体系结构和概念概述。 它描述了 Oracle 数据库服务器如何工作，为包含在其他手册中的很多实用信息奠定了概念基础。本手册中的信息适用于在所有操作系统上运行的 Oracle 数据库服务器。

本序言包含以下主题：

- 听众
- 文档可访问性
- 相关文档
- 约定

听众

Oracle 数据库概念适用于技术型用户，主要是初次接触 Oracle 数据库的数据库管理员和数据库应用程序开发人员。 通常，本手册的读者已具有管理或开发其他关系数据库应用程序的经验。

若要使用本手册，你必须知道以下内容：

- 关系数据库的总体概念
- 第 1 章，"Oracle 数据库简介"中的概念和术语
- 正在运行 Oracle 的操作系统环境

文档可访问性

我们的目标是使 所有用户都可以访问 Oracle 的 产品、 服务和支持文档，包括残障人士。为此，我们的文档包括使用辅助技术向用户提供信息的功能。本文档的 HTML 格式也是可用的，并包含方便残疾人士访问的标记。辅助功能标准将继续随着时间的推移而改进，Oracle 正积极与其他市场领先的技术供应商一起，克服技术障碍，使我们所有的客户都可以访问我们的文档。欲知更多的信息，请访问 Oracle 辅助功能程序 的 Web 站点：

to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at http://www.oracle.com/accessibility/.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/support/contact.docxl or visit http://www.oracle.com/accessibility/support.docxl if you are hearing impaired.

Related Documentation

This manual is intended to be read with the following manuals:

- *Oracle Database 2 Day DBA*
- *Oracle Database 2 Day Developer's Guide*

For more related documentation, see "Oracle Database Documentation Roadmap".

Many manuals in the Oracle Database documentation set use the sample schemas of the seed database, which is installed by default when you install Oracle Database. Refer to *Oracle Database Sample Schemas* for information on how these schemas were created and how you can use them.

Conventions

The following text conventions are used in this manual:

http://www.oracle.com/accessibility/ 。

文档中示例代码的可访问性

屏幕阅读器可能并不总能正确读取此文档中的示例代码。代码编写约定要求右大括号应该另启一行；但是，某些屏幕阅读器可能并不总能读取只包含一个括号或大括号的一行文本。

在文档中对外部网站链接的可访问性

此文档可能包含指向并不由 Oracle 拥有或控制的其他公司或机构的 Web 站点。Oracle 不对这些网站的可访问性负责。

访问 Oracle 技术支持

Oracle 客户可以通过“我的 Oracle 技术支持” 访问电子化支持服务。有关信息，请访问 http://www.oracle.com/support/contact.html，或者，如果你有听力障碍，请访问 http://www.oracle.com/accessibility/support.html 。

相关文档

本手册适合与以下手册一起使用：

- *《Oracle 数据库 2 日 DBA》*
- 《*Oracle 数据库 2 日开发者指南*》

更多的相关文档，请参阅《Oracle 数据库文档路线图》。

数据库文档集中的很多手册都使用种子数据库中的示例模式, 它是在您安装 Oracle 数据库时默认安装的。 有关如何创建这些模式和如何使用它们的信息，请参阅《*Oracle 数据库示例模式*》

约定

本手册使用以下文本约定：

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates manual titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

约定	含义
粗体	粗体类型表示在一段文本或术语表中定义的一个操作或术语相关联的图形用户界面元素。
斜体	斜体类型表示手册标题、 强调、或让您提供特定值的占位符变量。
等宽字体	等宽类型表示一个段落、URL、示例代码、屏显文本或您输入的文本中的命令。

1 Introduction to Oracle Database

This chapter provides an overview of Oracle Database and contains the following sections:

- About Relational Databases
  - Database Management System (DBMS)
  - Relational Model
  - Relational Database Management System (RDBMS)
  - Brief History of Oracle Database
- Schema Objects
  - Tables
  - Indexes
- Data Access
  - Structured Query Language (SQL)
  - PL/SQL and Java
- Transaction Management
  - Transactions
  - Data Concurrency
  - Data Consistency
- Oracle Database Architecture
  - Database and Instance
  - Database Storage Structures
  - Database Instance Structures
  - Application and Networking Architecture
- Oracle Database Documentation Roadmap
  - Basic Group
  - Intermediate Group
  - Advanced Group

About Relational Databases

Every organization has information that it must store and manage to meet its requirements. For example, a corporation must collect and maintain human resources records for its employees. This information must be available to those who need it. An **information system** is a formal

第 1 章 Oracle 数据库简介

本章提供了 Oracle 数据库的概述，并包含以下各节：

- 关于关系数据库
  - 数据库管理系统(DBMS)
  - 关系模型
  - 关系数据库管理系统(RDBMS)
  - Oracle 数据库发展简史
- 模式对象
  - 表
  - 索引
- 数据访问
  - 结构化查询语言(SQL)
  - PL/SQL 和 Java
- 事务管理
  - 事务
  - 数据并发性
  - 数据一致性
- Oracle 数据库架构
  - 数据库和实例
  - 数据库存储结构
  - 数据库实例结构
  - 应用程序和网络架构
- Oracle 数据库文档路线图
  - 基本组
  - 中级组
  - 高级组

关于关系数据库

每个组织都有其必须存储和管理的信息，以满足其需求。 例如, 公司必须收集和维护其雇员的人力资源纪录。 此信息对需要它的人必须是可用的。 **信息系统**是一个正式的系统，用于存储和处理信息。

system for storing and processing information.
An information system could be a set of cardboard boxes containing manila folders along with rules for how to store and retrieve the folders. However, most companies today use a database to automate their information systems. A database is an organized collection of information treated as a unit. The purpose of a database is to collect, store, and retrieve related information for use by database applications.

Database Management System (DBMS)

A **database management system (DBMS)** is software that controls the storage, organization, and retrieval of data. Typically, a DBMS has the following elements:

- Kernel code

  This code manages memory and storage for the DBMS.
- Repository of metadata

  This repository is usually called a data dictionary.
- Query language

  This language enables applications to access the data.

A **database application** is a software program that interacts with a database to access and manipulate data.
The first generation of database management systems included the following types:

- Hierarchical

  A **hierarchical database** organizes data in a tree structure. Each parent record has one or more child records, similar to the structure of a file system.
- Network

  A **network database** is similar to a hierarchical database, except records have a many-to-many rather than a one-to-many relationship.

The preceding database management systems stored data in rigid, predetermined relationships. Because no data definition language existed, changing the structure of the data was difficult. Also, these systems lacked a simple query language, which hindered application development.

信息系统可能是一组文件柜, 其中包含许多文件夹，以及如何存储和检索文件夹的规则。 但是，大多数公司现在使用数据库来自动化其信息系统。 数据库是信息的一个有组织的集合, 被作为一个整体来看待。 数据库的目的是收集、 存储、和检索相关的信息, 以供数据库应用程序使用。

数据库管理系统(DBMS)

数据库管理系统 (DBMS) 是一种软件，用于控制对数据的存储、 组织和检索。 通常，一个 DBMS 具有以下元素：

- 内核代码

  此代码为 DBMS 管理内存和存储。
- 元数据的存储库

  此存储库通常称为数据字典。
- 查询语言

  此语言使应用程序能够访问数据

数据库应用程序是一个与数据库进行交互，以访问和操作其数据的软件程序。
第一代的数据库管理系统包括以下类型：

- 层次型

  层次数据库把数据组织在树状结构中。 每个父记录有一个或多个子记录，类似于文件系统的结构。

- 网络型

  网络数据库类似于层次数据库，但有一个区别，即记录之间是多对多的关系，而不是一对多的关系。

前面所述的数据库管理系统按照严格的、 预定义的关系存储数据。 由于不存在任何数据定义语言，更改数据结构是很困难的。 同时，这些系统缺乏一种简单的查询语言, 则阻碍了应用程序的开发。

Relational Model

In his seminal 1970 paper "A Relational Model of Data for Large Shared Data Banks," E. F. Codd defined a relational model based on mathematical set theory. Today, the most widely accepted database model is the relational model.

A **relational database** is a database that conforms to the relational model. The relational model has the following major aspects:

- Structures

  Well-defined objects store or access the data of a database.
- Operations

  Clearly defined actions enable applications to manipulate the data and structures of a database.
- Integrity rules

  Integrity rules govern operations on the data and structures of a database.

A relational database stores data in a set of simple relations. A **relation** is a set of tuples. A **tuple** is an unordered set of **attribute** values.

A **table** is a two-dimensional representation of a relation in the form of rows (tuples) and columns (attributes). Each row in a table has the same set of columns. A relational database is a database that stores data in relations (tables). For example, a relational database could store information about company employees in an employee table, a department table, and a salary table.

See Also:

http://portal.acm.org/citation.cfm?id=362685 for an abstract and link to Codd's paper

Relational Database Management System (RDBMS)

The relational model is the basis for a **relational database management system (RDBMS)**. Essentially, an RDBMS moves data into a database, stores the data, and retrieves it so that it can be manipulated by applications. An RDBMS distinguishes between the following types of operations:

- Logical operations

关系模型

E.F.Codd 页: 3

在他 1970 年发表的论文《大型共享数据库数据的关系模型》中， 定义了一个基于数学集合理论的关系模型。 目前，最广泛接受的数据库模型就是关系模型。

关系数据库是一个符合关系模型的数据库。 关系模型有以下主要方面：

- 结构

  定义良好的对象, 用于存储或访问数据库的数据。
- 操作

  清楚定义的操作, 使应用程序可以处理数据库中的数据和结构。
- 完整性规则

  完整性规则用于管理在数据库的数据和结构上的操作。

关系数据库将数据存储在一组简单的关系中。 **关系**是一个元组的集合。 一个**元组** 是一些**属性**值的无序的集合。

表是一个关系的二维表示, 关系由行 （元组） 和列 （属性） 的形式构成。 表中的每一行具有相同的列集。 关系数据库是一个将数据存储在关系（表）中的数据库。 例如，关系数据库可以在一个雇员表、 部门表和薪金表中存储有关公司雇员的信息。

另见：

http://portal.acm.org/citation.cfm?id=362685 关于 Codd 的论文摘要及链接

关系数据库管理系统(RDBMS)

关系模型是**关系数据库管理系统 (RDBMS)** 的基础。 实际上，RDBMS 将数据移入数据库、在数据库中存储和检索数据，以便应用程序能对其进行操作。 RDBMS 区分下列两种类型的操作：

- 逻辑操作

In this case, an application specifies *what* content is required. For example, an application requests an employee name or adds an employee record to a table.

- Physical operations

  In this case, the RDBMS determines *how* things should be done and carries out the operation. For example, after an application queries a table, the database may use an index to find the requested rows, read the data into memory, and perform many other steps before returning a result to the user. The RDBMS stores and retrieves data so that physical operations are transparent to database applications.

Oracle Database is an RDBMS. An RDBMS that implements object-oriented features such as user-defined types, inheritance, and polymorphism is called an **object-relational database management system (ORDBMS)**. Oracle Database has extended the relational model to an object-relational model, making it possible to store complex business models in a relational database.

Brief History of Oracle Database

The current version of Oracle Database is the result of over 30 years of innovative development. Highlights in the evolution of Oracle Database include the following:

- Founding of Oracle

  In 1977, Larry Ellison, Bob Miner, and Ed Oates started the consultancy Software Development Laboratories, which became Relational Software, Inc. (RSI). In 1983, RSI became Oracle Systems Corporation and then later Oracle Corporation.

- First commercially available RDBMS

  In 1979, RSI introduced Oracle V2 (Version 2) as the first commercially available SQL-based RDBMS, a landmark event in the history of relational databases.

- Portable version of Oracle Database

  Oracle Version 3, released in 1983, was the first relational database to run on mainframes, minicomputers, and PCs. The database was written in C, enabling the database to be ported to multiple

在这种情况下, 由应用程序指定需要*哪些*内容。 例如，应用程序请求一个雇员的姓名，或向表中添加一个雇员记录。

- 物理操作

  在这种情况下，由 RDBMS 确定*如何*操作，并执行该操作。 例如，应用程序对表发出查询后，数据库可能会使用索引来查找请求的行、将数据读入内存，并执行许多其他步骤，最后向用户返回一个结果。 由 RDBMS 存储和检索数据，因此物理操作对于数据库应用程序是透明的。

Oracle 数据库是一个 RDBMS。 实现了面向对象特征（比如用户定义类型、继承和多态性）的 RDBMS，被称为**对象-关系数据库管理系统 (ORDBMS)**。 Oracle 数据库已经将关系模型扩展为对象-关系模型，使其可以在关系数据库中存储复杂的业务模型。

Oracle数据库的发展简史

Oracle 数据库的当前版本是超过 30 年的创新发展的结果。 Oracle 数据库发展过程中的重要事件包括：

- 创立 Oracle 公司

  1977 年， 拉里 • 埃利森、 鲍勃 • 穆勒、和爱德 • 奥茨成立了“软件开发实验室”咨询公司，其后又叫做“关系软件”(RSI)公司，1983 年，RSI 公司成为 Oracle 系统公司, 再后来又成为 Oracle 公司。

- 第一个商用 RDBMS

  1979 年，RSI 公司发布了 Oracle V2 （版本 2），这是第一个商用的基于 SQL 的 RDBMS，它是关系数据库发展史中的一个里程碑事件。

- 可移植版本的 Oracle 数据库

  Oracle 版本 3 发布于 1983 年，是第一个可以同时在大型机、 小型机和个人电脑上运行的关系数据库。 该数据库用 C 语言编写，使其可以被移植到多种平台上。

platforms.

- Enhancements to concurrency control, data distribution, and scalability

  Version 4 introduced multiversion read consistency. Version 5, released in 1985, supported client/server computing and distributed database systems. Version 6 brought enhancements to disk I/O, row locking, scalability, and backup and recovery. Also, Version 6 introduced the first version of the PL/SQL language, a proprietary procedural extension to SQL.
- PL/SQL stored program units

  Oracle7, released in 1992, introduced PL/SQL stored procedures and triggers.
- Objects and partitioning

  Oracle8 was released in 1997 as the object-relational database, supporting many new data types. Additionally, Oracle8 supported partitioning of large tables.
- Internet computing

  Oracle8*i* Database, released in 1999, provided native support for internet protocols and server-side support for Java. Oracle8*i* was designed for internet computing, enabling the database to be deployed in a multitier environment.
- Oracle Real Application Clusters (Oracle RAC)

  Oracle9*i* Database introduced Oracle RAC in 2001, enabling multiple instances to access a single database simultaneously. Additionally, Oracle XML Database (Oracle XML DB) introduced the ability to store and query XML.
- Grid computing

  Oracle Database 10*g* introduced grid computing in 2003. This release enabled organizations to virtualize computing resources by building a grid infrastructure based on low-cost commodity servers. A key goal was to make the database self-managing and self-tuning. Oracle Automatic Storage Management (Oracle ASM) helped achieve this goal by virtualizing and simplifying database

- 并发控制、 数据分发、和可扩展性等增强功能

  版本 4 引入了多版本读一致性。版本 5 发布于 1985 年，支持客户端/服务器计算和分布式数据库系统。 版本 6 增强了磁盘 I/O、行锁定、可扩展性、以及备份和恢复。并且，版本 6 推出了 PL/SQL 语言第一版，这是专门针对 SQL 的过程化扩展。

- PL/SQL 存储程序单元

  Oracle7 发布于 1992 年，引入了 PL/SQL 存储过程和触发器。

- 对象和分区

  Oracle8 发布于 1997 年, 作为一种对象-关系数据库，支持许多新的数据类型。 此外，Oracle8 支持对大型表进行分区。

- 因特网计算

  Oracle8i 数据库发布于 1999 年，提供了互联网协议的本机支持，和服务器端的 Java 支持。 Oracle8i 是为因特网计算而设计的, 这使得数据库可以在多层环境中部署。

- Oracle 真正应用集群 （Oracle RAC）

  Oracle9i 数据库在 2001 年引入了 Oracle RAC, 使得多个实例可以同时访问单个数据库。 此外，Oracle XML 数据库 （Oracle XML DB） 引入了存储和查询 XML 的能力。

- 网格计算

  Oracle 数据库 10g 在 2003 年引入了网格计算。 此版本使得各个公司可以通过构建基于低成本服务器的网格基础设施, 来虚拟化计算资源。 一个关键的目标是使数据库可以自我管理和自我优化。 Oracle 自动存储管理 (Oracle ASM) 通过虚拟化和简化数据库存储管理, 有助于实现这一目标。

storage management.

- Manageability, diagnosability, and availability

  Oracle Database 11*g*, released in 2007, introduced a host of new features that enable administrators and developers to adapt quickly to changing business requirements. The key to adaptability is simplifying the information infrastructure by consolidating information and using automation wherever possible.

See Also:
http://www.oracle.com/technology/oramag/oracle/07-jul/o4730.docxl for a more detailed description of the evolution of Oracle Database

Schema Objects

One characteristic of an RDBMS is the independence of physical data storage from logical data structures. In Oracle Database, a database schema is a collection of logical data structures, or schema objects. A database schema is owned by a database user and has the same name as the user name.

Schema objects are user-created structures that directly refer to the data in the database. The database supports many types of schema objects, the most important of which are tables and indexes.

See Also:
"Introduction to Schema Objects"

Tables

A table describes an entity such as employees. You define a table with a table name, such as `employees`, and set of columns. In general, you give each column a name, a data type, and a width when you create the table.

A table is a set of rows. A column identifies an attribute of the entity described by the table, whereas a row identifies an instance of the entity. For example, attributes of the employees entity correspond to columns for employee ID and last name. A row identifies a specific employee.

You can optionally specify rules for each column of a table. These rules are called integrity constraints. One example is a `NOT NULL` integrity constraint. This constraint forces the column to contain a value in every row.

See Also:

- 可管理性、 可诊断性、和可用性

  Oracle 数据库 11g, 发布于 2007 年，引入了大量的新功能，使管理员和开发人员可以快速适应不断变化的业务需求。 这种适应性的关键，在于通过整合信息和尽可能使用自动化，以简化信息基础架构。

另见：
http://www.oracle.com/technology/oramag/oracle/07-jul/o4730.docx 关于 Oracle 数据库的演变的更详细说明

模式对象

RDBMS 的特点之一，是物理数据存储与逻辑数据结构的独立性。 在 Oracle 数据库中，数据库模式是逻辑数据结构或模式对象的集合。 数据库模式由一个数据库用户拥有，并与用户名具有相同的名称。

模式对象是用户创建的结构，它直接引用数据库中的数据。 数据库支持多种类型的模式对象，其中最重要的是表和索引

另见：
"模式对象介绍"

表

表描述一个实体，例如雇员。使用一个表名（如 `employees`）和一个列集来定义表。 一般地，当你创建表时，你应该给出每一列的列名、 数据类型、和列宽。

表是一些行的集合。 列标识实体的属性，而行标识实体的实例。 例如，雇员实体的属性对应雇员 ID 列和姓氏列。 行标识一个特定的雇员。

您可以选择性地为每个表列指定规则。 这些规则称为完整性约束。例如，“非空”即是一个完整性约束。 此约束强制每一行中的该列都包含一个值。

另见：

- "Overview of Tables"
- Chapter 5, "Data Integrity"

Indexes

An index is an optional data structure that you can create on one or more columns of a table. Indexes can increase the performance of data retrieval. When processing a request, the database can use available indexes to locate the requested rows efficiently. Indexes are useful when applications often query a specific row or range of rows.

Indexes are logically and physically independent of the data. Thus, you can drop and create indexes with no effect on the tables or other indexes. All applications continue to function after you drop an index.

See Also:

"Overview of Indexes"

Data Access

A general requirement for a DBMS is to adhere to accepted industry standards for a data access language.

Structured Query Language (SQL)

SQL is a set-based declarative language that provides an interface to an RDBMS such as Oracle Database. In contrast to procedural languages such as C, which describe *how* things should be done, SQL is nonprocedural and describes *what* should be done. Users specify the result that they want (for example, the names of current employees), not how to derive it. SQL is the ANSI standard language for relational databases.

All operations on the data in an Oracle database are performed using SQL statements. For example, you use SQL to create tables and query and modify data in tables. A SQL statement can be thought of as a very simple, but powerful, computer program or instruction. A SQL statement is a string of SQL text such as the following:

```
SELECT first_name, last_name FROM employees;
```

SQL statements enable you to perform the following tasks:

- Query data
- Insert, update, and delete rows in a table
- Create, replace, alter, and drop objects
- Control access to the database and its objects
- Guarantee database consistency and integrity

- "表概述"
- 第 5 章，数据完整性

索引

索引是页: 7

一个可选的数据结构，您可以在表中的一个或多个列上创建索引。 索引可以提高数据检索的性能。 在处理一个请求时，数据库可以使用可用索引有效地找到请求的行。 当应用程序经常查询某一特定行或特定范围的行时，索引很有用。

索引在逻辑上和物理上都独立于数据。 因此，可以删除和创建索引，而对表或其它索引没有任何影响。 在删除索引后，所有应用程序可以继续运行。

另见：

"索引概述"

数据访问

对 DBMS 的一个一般要求，是要遵循公认的行业标准数据访问语言。

结构化查询语言(SQL)

SQL 是一种基于集合的声明性语言，提供一个到关系数据库（如 Oracle 数据库）的接口。 相对于过程化语言（如 C）描述的是*怎么*做，SQL 作为非过程化的语言，它描述的是应做些*什么*。 用户可以指定他们所希望的结果（例如，最新的雇员名单），而不是如何获得它。 SQL 是关系数据库的 ANSI 标准语言。

在 Oracle 数据库中对数据所作的所有操作都使用 SQL 语句来执行的。 例如，您可以使用 SQL 来创建表，并查询和修改表中的数据。 SQL 语句可以认为是一种很简单、但功能强大的计算机程序或指令。 SQL 语句是一个字符串或 SQL 文本，例如：

```
SELECT first_name, last_name FROM employees;
```

SQL 语句使您可以执行以下任务：

- 查询数据
- 插入、更新、和删除表中的行。
- 创建、替换、 更改、和删除对象
- 控制对数据库及其对象的访问

- 保证数据库的一致性和完整性

SQL unifies the preceding tasks in one consistent language. **Oracle SQL** is an implementation of the ANSI standard. Oracle SQL supports numerous features that extend beyond standard SQL.

See Also:

Chapter 7, "SQL"

PL/SQL and Java

PL/SQL is a procedural extension to Oracle SQL. PL/SQL is integrated with Oracle Database, enabling you to use all of the Oracle Database SQL statements, functions, and data types. You can use PL/SQL to control the flow of a SQL program, use variables, and write error-handling procedures.

A primary benefit of PL/SQL is the ability to store application logic in the database itself. A procedure or function is a schema object that consists of a set of SQL statements and other PL/SQL constructs, grouped together, stored in the database, and run as a unit to solve a specific problem or to perform a set of related tasks. The principal benefit of server-side programming is that built-in functionality can be deployed anywhere. Oracle Database can also store program units written in Java. A Java stored procedure is a Java method published to SQL and stored in the database for general use. You can call existing PL/SQL programs from Java and Java programs from PL/SQL.

See Also:

Chapter 8, "Server-Side Programming: PL/SQL and Java" and "Client-Side Database Programming"

Transaction Management

Oracle Database is designed as a multiuser database. The database must ensure that multiple users can work concurrently without corrupting one another's data.

Transactions

An RDBMS must be able to group SQL statements so that they are either all **committed**, which means they are applied to the database, or all **rolled back**, which means they are undone. A transaction is a logical, atomic unit of work that contains one or more SQL statements.

SQL 用一种统一的语言了完成了所有上述任务。 Oracle SQL 是 ANSI 标准的一种实现。 Oracle SQL 支持许多标准 SQL 之外的扩展功能。

另见：

第 7 章, "SQL"

PL/SQL和Java

PL/SQL 是 Oracle SQL 的过程化扩展。 PL/SQL 与 Oracle 数据库集成，使您能够使用所有的 Oracle 数据库的 SQL 语句、 函数、和数据类型。 您可以使用 PL/SQL 来控制 SQL 程序流程、使用变量、以及编写错误处理程序。

PL/SQL 的主要优点，是能够将应用程序逻辑存储在数据库本身之中。 过程或函数是一个模式对象，它包含一组 SQL 语句和一些其他的 PL/SQL 结构，共同组合在一起，存储在数据库中， 并作为一个单元来运行，以解决特定的问题，或执行一组相关的任务。 服务器端编程的主要好处，是内置功能可以在任意位置部署。

Oracle 数据库还可以存储用 Java 编写的程序单元。 一个 Java 存储过程是一个发布到 SQL 的 Java 方法，被存储在数据库中用于一般用途。 您可以在 Java 中调用 PL/SQL 程序，或在 PL/SQL 中调用 Java 程序。

另见：

第 8 章, "服务器端编程：PL/SQL 和 Java" 和 "客户端数据库编程"

事务管理

Oracle 数据库被设计为多用户数据库。 数据库必须确保多个用户可以同时工作，而不会损坏彼此的数据。

事务

RDBMS 必须能够对 SQL 语句进行分组，以将它们一起提交，将其应用于数据库；或者将它们全部回滚，将其全部撤消。 事务是一个逻辑、 原子的工作单元，包含一条或多条 SQL 语句。

An illustration of the need for transactions is a funds transfer from a savings account to a checking account. The transfer consists of the following separate operations:

- 1. Decrease the savings account.
- 2. Increase the checking account.
- 3. Record the transaction in the transaction journal.

Oracle Database guarantees that all three operations succeed or fail as a unit. For example, if a hardware failure prevents a statement in the transaction from executing, then the other statements must be rolled back.

Transactions are one of the features that sets Oracle Database apart from a file system. If you perform an atomic operation that updates several files, and if the system fails halfway through, then the files will not be consistent. In contrast, a transaction moves an Oracle database from one consistent state to another. The basic principle of a transaction is "all or nothing": an atomic operation succeeds or fails as a whole.

See Also:

Chapter 10, "Transactions"

Data Concurrency

A requirement of a multiuser RDBMS is the control of concurrency, which is the simultaneous access of the same data by multiple users. Without concurrency controls, users could change data improperly, compromising data integrity. For example, one user could update a row while a different user simultaneously updates it.

If multiple users access the same data, then one way of managing concurrency is to make users wait. However, the goal of a DBMS is to reduce wait time so it is either nonexistent or negligible. All SQL statements that modify data must proceed with as little interference as possible. Destructive interactions, which are interactions that incorrectly update data or alter underlying data structures, must be avoided.

Oracle Database uses locks to control concurrent access to data. A lock is a mechanism that prevents destructive interaction between transactions accessing a shared resource. Locks help ensure data integrity while allowing maximum concurrent access to data.

See Also:

"Overview of the Oracle Database Locking Mechanism"

Data Consistency

举一个需要使用事务的例子，将资金从一个储蓄帐户转账到一个支票帐户。这笔转账由以下几个单独的操作组成：

- 从储蓄帐户中扣除资金。
- 将资金增加到支票帐户。
- 在事务日志中记录该事务。

Oracle 数据库确保将所有三个操作作为一个单元来处理, 要么全部成功，要么全部失败。 例如, 如果硬件故障阻止了事务中的某个语句的执行，那么其他语句必须回滚。

事务是 Oracle 数据库有别于文件系统的特征之一。 如果您执行一个原子操作，更新了几个文件，但是系统在处理过程中失败，那么文件将处于不一致状态。 相比之下，事务则会将 Oracle 数据库从一个一致状态转移到另一个一致状态。 事务的基本原则是，"要么全做，要么全都不做"： 一个原子操作作为一个整体，要么都成功，要么都失败。

另见：

第 10 章, "事务"

数据并发性

对多用户 RDBMS 的一个要求是并发控制，即多个用户同时访问同一数据。如果没有并发控制，则用户可能会不正确地更改数据，并影响数据完整性。例如，一个用户可能企图更新一行，而另一个不同的用户同时也在更新它。

如果多个用户访问相同的数据，那么管理并发的其中一种方法，就是让用户等待。而 DBMS 的目标却是减少等待时间，甚至于感觉不到等待的存在，或其等待时间可以少到忽略不计。 所有修改数据的 SQL 语句必须尽可能做到互不干扰。那些具有破坏性的交互性操作，可能会不正确地更新数据，或不正确地更改基础数据结构，必须予以避免。

Oracle 数据库使用锁来控制对数据的并发访问。 锁是一种机制，可以防止多个事务在访问共享资源时的相互破坏作用。锁有助于确保数据的完整性，同时允许对数据的最大并发访问。

另见：

"Oracle 数据库锁定机制概述"

数据一致性

In Oracle Database, each user must see a consistent view of the data, including visible changes made by a user's own transactions and committed transactions of other users. For example, the database must prevent dirty reads, which occur when one transaction sees uncommitted changes made by another concurrent transaction.
Oracle Database always enforces **statement-level read consistency**, which guarantees that the data returned by a single query is committed and consistent with respect to a single point in time. Depending on the transaction isolation level, this point is the time at which the statement was opened or the time the transaction began. The Flashback Query feature enables you to specify this point in time explicitly.
The database can also provide read consistency to all queries in a transaction, known as **transaction-level read consistency**. In this case, each statement in a transaction sees data from the same point in time, which is the time at which the transaction began.
See Also:

- Chapter 9, "Data Concurrency and Consistency"
- *Oracle Database Advanced Application Developer's Guide* to learn about Flashback Query

Oracle Database Architecture

A **database server** is the key to information management. In general, a server reliably manages a large amount of data in a multiuser environment so that users can concurrently access the same data. A database server also prevents unauthorized access and provides efficient solutions for failure recovery.

Database and Instance

An Oracle database server consists of a database and at least one database instance (commonly referred to as simply an **instance**). Because an instance and a database are so closely connected, the term **Oracle database** is sometimes used to refer to both instance and database. In the strictest sense the terms have the following meanings:

- Database

  A database is a set of files, located on disk, that store data. These files can exist independently of a database instance.
- Database instance

数据库中，每个用户必须看到数据的一致视图，包括用户自己的事务以及其他用户已提交的事务所做出的更改。 例如，数据库必须防止脏读，若一个事务看到另一个并发事务未提交的更改，即是这种情况。

Oracle 数据库始终强制执行**语句级读取的一致性**，保证单个查询所返回的数据是已提交的，并与某个时间点一致。基于不同的事务隔离级别，这个时间点要么是语句被打开时，要么是事务开始时。闪回查询功能使您能够显式指定此时间点。

数据库也可以为整个事务中的所有查询提供读一致性，这叫**事务级读一致性**。在这种情况下，事务中的每一个语句看见的都是同一个时间点的数据，这个时间点也就是事务开始之时。

另见：

- 第 9 章, "数据并发性和一致性"
- 《*Oracle 数据库高级应用开发者指南*》了解闪回查询

Oracle数据库架构

数据库服务器是信息管理的关键。 通常，服务器可靠地管理多用户环境中的大量数据，以便使用户可以同时访问这些相同的数据。 数据库服务器还可防止未经授权的访问，并为故障恢复提供有效的解决方案。

数据库和实例

Oracle 数据库服务器包括一个数据库和至少一个数据库实例 （通常是指只有一个实例）。 因为实例和数据库是如此的紧密关联，Oracle 数据库这个术语有时既指实例，也指数据库。 这个术语在严格意义上具有以下含义：

- 数据库

  数据库是一组文件，位于磁盘，用于存储数据。 这些文件可以独立于数据库实例存在。
- 数据库实例

An instance is a set of memory structures that manage database files. The instance consists of a shared memory area, called the system global area (SGA), and a set of background processes. An instance can exist independently of database files.

Figure 1-1 shows a database and its instance. For each user connection to the instance, the application is run by a client process. Each client process is associated with its own server process. The server process has its own private session memory, known as the program global area (PGA).

Figure 1-1 Oracle Instance and Database

Description of "Figure 1-1 Oracle Instance and Database"

A database can be considered from both a physical and logical

实例是管理数据库文件的一组内存结构。 该实例包含一个叫做系统全局区 （SGA） 的共享内存区和一组后台进程。 实例可以独立于数据库文件而存在。

图 1 显示了一个数据库和它的实例。 对连接到实例的每个用户，其应用程序以一个客户端进程运行。 每个客户端进程都具与它自己的服务器进程相关联。 服务器进程有自己的私有会话内存区，称为程序全局区 （PGA）。

图 1-1 Oracle 实例和数据库

Description of "Figure 1-1 Oracle Instance and Database"

一个数据库可以从物理和逻辑这两个角度来考察。 物理数据是在操作系统

perspective. Physical data is data viewable at the operating system level. For example, operating system utilities such as the Linux `ls` and `ps` can list database files and processes. Logical data such as a table is meaningful only for the database. A SQL statement can list the tables in an Oracle database, but an operating system utility cannot.

The database has **physical structures** and **logical structures**. Because the physical and logical structures are separate, the physical storage of data can be managed without affecting access to logical storage structures. For example, renaming a physical database file does not rename the tables whose data is stored in this file.

See Also:

Chapter 13, "Oracle Database Instance"

Database Storage Structures

An essential task of a relational database is data storage. This section briefly describes the physical and logical storage structures used by Oracle Database.

Physical Storage Structures

The physical database structures are the files that store the data. When you execute the SQL command `CREATE DATABASE`, the following files are created:

- Data files

  Every Oracle database has one or more physical data files, which contain all the database data. The data of logical database structures, such as tables and indexes, is physically stored in the data files.
- Control files

  Every Oracle database has a control file. A control file contains metadata specifying the physical structure of the database, including the database name and the names and locations of the database files.
- Online redo log files

  Every Oracle Database has an online redo log, which is a set of two or more **online redo log files**. An online redo log is made up of redo entries (also called **redo records**), which record all changes made to data.

级别可查看的数据。 例如，使用操作系统实用程序，如 Linux ls 和 ps，可以列出数据库文件和进程。 而象表这样的逻辑数据只在数据库层面有意义。 SQL 语句可以列出 Oracle 数据库中的表，但操作系统实用程序却不能。

数据库具有物理结构和逻辑结构。 因为物理结构和逻辑结构是分开的，因此对物理数据存储的管理，不会影响对逻辑存储结构的访问。 例如，重命名物理数据库文件并不会重命名表，尽管其数据存储在此文件中。

另见：

第 13 章, "Oracle 数据库实例"

数据库存储结构

数据存储是关系数据库的一个重要任务。 本部分简要介绍了 Oracle 数据库所使用的物理和逻辑存储结构。

物理存储结构

物理数据库结构即是那些存储数据的文件。 当您执行 SQL 命令 CREATE DATABASE，会创建下列文件：

- 数据文件

  每个 Oracle 数据库都有一个或多个物理数据文件，其中包含数据库的所有数据。数据库的逻辑数据结构，如表和索引，在物理上存储在这些数据文件中。

- 控制文件

  每个 Oracle 数据库都有一个控制文件。 控制文件包含指定数据库物理结构的元数据，包括数据库名称，及数据库文件的位置和名称等。

- 在线重做日志文件

  每个数据库都有一个联机重做日志，它包含两个或更多联机重做日志文件。 一个联机重做日志由许多重做项组成（也称为重做记录），以记录对数据所做的所有更改。

Many other files are important for the functioning of an Oracle database server. These files include parameter files and diagnostic files. Backup files and archived redo log files are offline files important for backup and recovery.

See Also:

Chapter 11, "Physical Storage Structures"

Logical Storage Structures

This section discusses logical storage structures. The following logical storage structures enable Oracle Database to have fine-grained control of disk space use:

- Data blocks

  At the finest level of granularity, Oracle Database data is stored in data blocks. One data block corresponds to a specific number of bytes on disk.
- Extents

  An extent is a specific number of logically contiguous data blocks, obtained in a single allocation, used to store a specific type of information.
- Segments

  A segment is a set of extents allocated for a user object (for example, a table or index), undo data, or temporary data.
- Tablespaces

  A database is divided into logical storage units called **tablespaces**. A tablespace is the logical container for a segment. Each tablespace contains at least one data file.

See Also:

Chapter 12, "Logical Storage Structures"

Database Instance Structures

An Oracle database uses memory structures and processes to manage and access the database. All memory structures exist in the main memory of the computers that constitute the RDBMS.

When applications connect to an Oracle database, they are connected to a database instance. The instance services applications by allocating other memory areas in addition to the SGA, and starting other processes in

很多其他文件对 Oracle 数据库服务器的正常运行也是很重要的。 这些文件包括参数文件和诊断文件。 备份文件和归档重做日志文件是用于备份和恢复的重要的脱机文件。

另见：

第 11 章, "物理存储结构"

逻辑存储结构

本部分讨论逻辑存储结构。 以下的逻辑存储结构使得 Oracle 数据库可以按更细的粒度控制磁盘空间的使用：

- 数据块

  在最细的粒度级别，Oracle 数据库将数据存储在数据块中。 一个数据块对应于磁盘上的特定数目的字节。
- 扩展区

  扩展区是一定数目的逻辑上连续的数据块，在单次分配中获得，用于存储特定类型的信息。
- 段

  段是为用户对象 （例如一个表或索引）、回滚数据、或临时数据等分配的一组扩展区。
- 表空间

  数据库由称为表空间的逻辑存储单位组成。 表空间是段的逻辑容器。 每个表空间至少包含一个数据文件。

另见：

第 12 章, "逻辑存储结构"

数据库实例结构

Oracle 数据库使用内存结构和进程，来管理和访问数据库。 所有内存结构都存在于构成关系数据库的计算机主内存中。

当应用程序连接到 Oracle 数据库时，他们也连接到一个数据库实例。 实例为该程序服务，为其分配除 SGA 之外的其他内存区，并启动除后台进程之外的其他进程。

addition to background processes.

Oracle Database Processes

A process is a mechanism in an operating system that can run a series of steps. Some operating systems use the terms *job*, *task*, or *thread*. For the purpose of this discussion, a thread is equivalent to a process. An Oracle database instance has the following types of processes:

- Client processes

  These processes are created and maintained to run the software code of an application program or an Oracle tool. Most environments have separate computers for client processes.
- Background processes

  These processes consolidate functions that would otherwise be handled by multiple Oracle Database programs running for each client process. Background processes asynchronously perform I/O and monitor other Oracle Database processes to provide increased parallelism for better performance and reliability.
- Server processes

  These processes communicate with client processes and interact with Oracle Database to fulfill requests.

Oracle processes include server processes and background processes. In most environments, Oracle processes and client processes run on separate computers.

See Also:

Chapter 15, "Process Architecture"

Instance Memory Structures

Oracle Database creates and uses memory structures for purposes such as memory for program code, data shared among users, and private data areas for each connected user. The following memory structures are associated with an instance:

- System Global Area (SGA)

  The SGA is a group of shared memory structures that contain data and control information for one database instance. Examples of SGA components include cached data blocks and shared SQL areas.

Oracle数据库进程

进程是操作系统的一种机制，用于运行一系列步骤。 有些操作系统使用工作、 任务、或线程等术语。 在本论述中，线程等价于进程。 Oracle 数据库实例具有以下类型的进程：

- 客户端进程

  这些进程被创建和维护，主要用来运行应用程序代码或 Oracle 工具。 多数环境中都由单独的计算机来运行客户端进程。
- 后台进程

  这些进程整合了多种功能，以避免为每个客户端进程运行多个 Oracle 数据库程序。 后台进程以异步方式执行 i/o 操作，并监控其他数据库进程，以提供更大的并行度，达到更好的性能和可靠性
- 服务端进程

  这些进程与客户端进程通信，并与 Oracle 数据库进行交互，以完成客户端请求。

Oracle 进程包括服务器进程和后台进程。 在大多数的环境中， Oracle 进程和客户端进程在不同的计算机上运行。

另见：

第 15 章, "进程架构"

实例内存结构

Oracle 数据库创建和使用内存结构有几个目的，例如，用于在多个用户之间共享程序代码和数据的内存，以及用于每个已连接用户的私有数据区域。以下是与实例相关的内存结构：

- 系统全局区(SGA)

  SGA 是一组共享的内存结构，包含一个数据库实例的数据和控制信息。 例如，SGA 组件包括数据块缓存和共享 SQL 区域。

- Program Global Areas (PGA)

  A PGA is a memory region that contain data and control information for a server or background process. Access to the PGA is exclusive to the process. Each server process and background process has its own PGA.

See Also:

Chapter 14, "Memory Architecture"

Application and Networking Architecture

To take full advantage of a given computer system or network, Oracle Database enables processing to be split between the database server and the client programs. The computer running the RDBMS handles the database server responsibilities while the computers running the applications handle the interpretation and display of data.

Application Architecture

The **application architecture** refers to the computing environment in which a database application connects to an Oracle database. The two most common database architectures are client/server and multitier.

In a client/server architecture, the client application initiates a request for an operation to be performed on the database server. The server runs Oracle Database software and handles the functions required for concurrent, shared data access. The server receives and processes requests that originate from clients.

In a traditional **multitier architecture**, one or more application servers perform parts of the operation. An **application server** contains a large part of the application logic, provides access to the data for the client, and performs some query processing, thus lessening the load on the database. The application server can serve as an interface between clients and multiple databases and provide an additional level of security.

Service-oriented architecture (SOA) is a multitier architecture in which application functionality is encapsulated in **services**. SOA services are usually implemented as Web services. Web services are accessible through HTTP and are based on XML-based standards such as Web Services Description Language (WSDL) and SOAP.

Oracle Database can act as a Web service provider in a traditional multitier or SOA environment.

See Also:

- 程序全局区(PGA)

  PGA 是一个内存区域，其中包含某个服务器进程或后台进程的数据和控制信息。 PGA 只由专有的过程访问。 每个服务器进程和后台进程都有其自己的 PGA。

另见：

第 14 章, "内存结构"

应用程序和网络架构

为了充分利用一个特定的计算机系统或网络，数据库把任务在数据库服务器和客户端程序之间分开处理。 运行关系数据库的计算机负责处理数据库服务器，而运行应用程序处理的计算机负责解释和显示数据。

应用程序体系结构

应用程序体系结构是指一个数据库应用程序连接到 Oracle 数据库的计算环境。最常见的两种数据库体系结构是客户端/服务器体系结构和多层体系结构。

在一个客户端/服务器体系结构中，客户端应用程序发起一个将在数据库服务器上执行操作的请求。 服务器运行 Oracle 数据库软件，并处理并发、 共享数据访问所需的功能。 服务器接收并处理来自客户端的请求。

在一个传统的**多层体系结构**中，由一个或多个应用程序服务器执行部分（由传统 C/S 服务器执行的）操作。 **应用程序服务器**包含应用程序逻辑的很大一部分，为该客户机提供对数据的访问，并执行一些查询处理，从而减轻在数据库上的负载。 应用程序服务器可以作为客户端和多个数据库之间的接口，并提供一个额外的安全级别。

面向服务的体系结构 (SOA) 是一种多层体系结构，应用程序的功能被封装在服务中。 SOA 服务通常是作为 Web 服务实现的。 Web 服务可通过 HTTP 访问，并基于 XML 标准，如 Web 服务描述语言 (WSDL) 和 SOAP。

Oracle 数据库可以在传统的多层环境或 SOA 环境中扮演一个 Web 服务提供者的角色。

另见：

- "Overview of Multitier Architecture"
- *Oracle XML DB Developer's Guide* for more information about using Web services with the database

Networking Architecture

Oracle Net Services is the interface between the database and the network communication protocols that facilitate distributed processing and distributed databases. Communication protocols define the way that data is transmitted and received on a network. Oracle Net Services supports communications on all major network protocols, including TCP/IP, HTTP, FTP, and WebDAV.

Oracle Net, a component of Oracle Net Services, establishes and maintains a network session from a client application to a database server. After a network session is established, Oracle Net acts as the data courier for both the client application and the database server, exchanging messages between them. Oracle Net can perform these jobs because it is located on each computer in the network.

An important component of Net Services is the **Oracle Net Listener** (called the **listener**), which is a separate process that runs on the database server or elsewhere in the network. Client applications can send connection requests to the listener, which manages the traffic of these requests to the database server. When a connection is established, the client and database communicate directly.

The most common ways to configure an Oracle database to service client requests are:

- Dedicated server architecture

  Each client process connects to a dedicated server process. The server process is not shared by any other client for the duration of the client's session. Each new session is assigned a dedicated server process.
- Shared server architecture

  The database uses a pool of shared processes for multiple sessions. A client process communicates with a **dispatcher**, which is a process that enables many clients to connect to the same database instance without the need for a dedicated server process for each client.

- "多级体系结构概述"
- 《*Oracle XML DB 开发人员指南*》关于使用数据库 Web 服务的详细信息

网络体系结构

Oracle 网络服务是数据库与网络通信协议之间的接口，它对分布式处理和分布式数据库很有用。 通信协议定义了数据在网络上被传输和接收的方式。 Oracle 网络服务支持所有主要的网络通信协议，包括 TCP/IP、 HTTP、 FTP 、和 WebDAV

Oracle Net，是 Oracle 网络服务的一个组件，用于建立和保持一个客户端应用程序到数据库服务器的网络会话。 网络会话建立后，Oracle Net 作为客户端应用程序和数据库之间的数据传递员，在它们之间交换消息。 Oracle Net 之所以可以执行这些工作，是因为它位于网络中的每台计算机上。

网络服务的一个重要组成部分是 **Oracle 网络监听器**（简称**监听器**），它是一个单独的进程，运行在数据库服务器上，或在网络中的其它地方。 客户端应用程序可以发送连接请求到监听器，而监听器管理这些发送到数据库服务器的请求的流量。 当一个连接建立之后，客户端和数据库进行直接通信（，而不再需要监听器的参与）。

为服务客户端请求，最常见的两种配置 Oracle 数据库的方式是：

- 专用服务器体系结构

  每个客户端进程连接到一个专用的服务器进程。 在与客户端的会话期间，服务器进程不可以被任何其他客户端共享。 Oracle 会为每个新会话分配一个专用的服务器进程

- 共享服务器体系结构

  数据库使用一个用于多个会话的共享进程池。 客户端进程与一个调度器通信，调度器是一个进程，它使许多客户端能够连接到相同的数据库实例，而无需为每个客户端建立一个专用的服务器进程。

See Also:

- "Overview of Oracle Networking Architecture"
- *Oracle Database Net Services Administrator's Guide* to learn more about Oracle Net architecture
- *Oracle XML DB Developer's Guide* for information about using WebDAV with the database

Oracle Database Documentation Roadmap

This section explains how this manual should be read and where it fits into the Oracle Database documentation set as a whole.

To a new user, the Oracle Database documentation library can seem daunting. Not only are there are over 175 manuals, but many of these manuals are several hundred pages long. However, the documentation is designed with specific access paths to ensure that users are able to find the information they need as efficiently as possible.

The documentation set is divided into three layers or groups: basic, intermediate, and advanced. Users begin with the manuals in the basic group (*Oracle Database 2 Day DBA*, *Oracle Database 2 Day Developer's Guide*, or this manual), proceed to the manuals in the intermediate group (the *2 Day +* series), and finally to the advanced manuals, which include the remainder of the documentation set.

Basic Group

Technical users who are new to Oracle Database begin by reading one or more manuals in the basic group from cover to cover. Each manual in this group is designed to be read in two days. In addition to this manual, the basic group includes:

- *Oracle Database 2 Day DBA*

  This manual is a task-based DBA quick start that teaches you how to perform day-to-day database administrative tasks. It teaches you how to perform all common administrative tasks needed to keep the database operational, including how to perform basic troubleshooting and performance monitoring activities.
- *Oracle Database 2 Day Developer's Guide*

  This manual is a task-based database developer quick start guide that explains how to use the basic features of Oracle Database through SQL and PL/SQL.

The manuals in the basic group are closely related, which is reflected in

另见：

- "Oracle 网络架构概述"
- 《*Oracle 数据库网络服务管理员指南*》有关 Oracle 网络体系结构的详细信息
- 《*Oracle XML DB 开发人员指南*》有关使用数据库 WebDAV 的信息

Oracle数据库文档路线图

此部分说明了应如何阅读本手册，以及本手册在哪些地方和其它数据库文档溶为一个整体。

对一个新的用户来说，数据库文档库可能看起来令人生畏。 不只是有超过 175 本手册，而且许多手册都长达几百页。 但是，该文档设计了特定的访问路径，确保用户能够尽可能高效地找到他们所需的信息。

文档集分为三个层次或组： 基本、中级、和高级。 用户可以从基本组开始（《*Oracle 数据库 2 日 DBA*》、《*Oracle 数据库 2 日开发者指南*》，或本手册），然后继续推进到中级组 (2 日 + 系列)，最后是高级手册，即文档集的其余部分。

基本组

新的 Oracle 数据库技术用户，从基本组中的一个或多个手册开始，从头到尾的阅读。 在此组中的每个手册被设计为在两天内看完。 除了本手册外，基本组包括：

- 《*Oracle 数据库 2 日 DBA*》

  本手册是一个基于任务的数据库管理员快速启动教程，教您如何执行日常的数据库管理任务。 它教您如何执行所有的常见管理任务，以保持数据库可操作，包括如何执行基本的故障排除和性能监测活动。

- 《*Oracle 数据库 2 日开发者指南*》

  本手册是基于任务的数据库开发人员快速启动指南，它解释了如何通过 SQL 和 PL/SQL 使用数据库的基本功能。

基本组的几本手册是密切相关的，这可以从大量存在的交叉引用看出来。

the number of cross-references. For example, *Oracle Database Concepts* frequently sends users to a *2 Day* manual to learn how to perform a task based on a concept. The *2 Day* manuals frequently references *Oracle Database Concepts* for conceptual background about a task.

Intermediate Group

The next step up from the basic group is the intermediate group. The manuals in this group are prefixed with the word *2 Day +* because they expand on and assume information contained in the *2 Day* manuals. These manuals cover topics in more depth than was possible in the basic manuals, or cover topics of special interest. As shown in Table 1-1, the *2 Day +* manuals are divided into manuals for DBAs and developers.

Table 1-1 Intermediate Group: 2 Day + Guides

Database Administrators

Oracle Database 2 Day + Performance Tuning Guide

Oracle Database 2 Day + Real Application Clusters Guide

Oracle Database 2 Day + Data Warehousing Guide

Oracle Database 2 Day + Data Replication and Integration Guide

Oracle Database 2 Day + Security Guide

Database Developers

Oracle Database 2 Day + Application Express Developer's Guide

Oracle Database 2 Day + Java Developer's Guide

Oracle Database 2 Day + .NET Developer's Guide for Microsoft Windows

Oracle Database 2 Day + PHP Developer's Guide

Advanced Group

The next step up from the intermediate group is the advanced group. These manuals are intended for expert users who require more detailed information about a particular topic than can be provided by the *2 Day +* manuals. Essential reference manuals in the advanced group include:

- *Oracle Database SQL Language Reference*

  This manual is the definitive source of information about Oracle SQL.
- *Oracle Database Reference*

  The manual is the definitive source of information about

例如 Oracle 数据库概念经常将用户引导到 2 日手册以了解如何执行一项基于某个概念的任务。 针对某一任务，2 日手册经常引用《*Oracle 数据库概念*》中的概念性背景知识。

中级组

基本组的下一步是中级组。 在此组中的手册带有“2 日+”前缀，因为他们扩展和承袭了 2 日手册中包含的信息。 这些手册相比包括基本手册而言，讨论的主题更加深入，或者更加侧重问题的某些特殊方面。 如表 1-1 中所示，2 日 + 手册分别针对数据库管理员和开发人员。

表 1-1 中级组: 2 日 + 指南

数据库管理员

《*Oracle 数据库 2 日+ 性能调整指南*》

《*Oracle 数据库 2 日+ 实际应用程序群集指南*》

《*Oracle 数据库 2 日+ 数据仓库指南*》

《*Oracle 数据库 2 日+ 数据复制和集成指南*》

《*Oracle 数据库 2 日+ 安全指南*》

数据库开发人员

《*Oracle 数据库 2 日+ 应用程序快速开发人员指南* 》

《*Oracle 数据库 2 日+ Java 开发人员指南*》

《*Oracle 数据库 2 日+.NET for Microsoft Windows 开发人员指南*》

《*Oracle 数据库 2 日+ PHP 开发人员指南*》

高级组

中级组的下一步是高级组。 这些手册供专家用户使用，它们相比 2 日 + 手册而言，提供了针对有关特定主题的更详细的信息。 高级组中的基本参考手册包括：

- 《*Oracle 数据库 SQL 语言参考*》

  本手册是关于 Oracle SQL 的权威信息来源。

- 《*Oracle 数据库参考*》

  本手册是关于初始化参数、 数据字典视图、和动态性能视图的权威

initialization parameters, data dictionary views, and dynamic performance views.

The advanced guides are too numerous to list in this section. Table 1-2 lists guides that are used by the majority of expert DBAs and developers at one time or another.

Table 1-2 Advanced Group

Database Administrators

Oracle Database Administrator's Guide

Oracle Database Performance Tuning Guide

Oracle Database Backup and Recovery User's Guide

Oracle Real Application Clusters Administration and Deployment Guide

Database Developers

Oracle Database Advanced Application Developer's Guide

Oracle Database PL/SQL Language Reference

Oracle Database PL/SQL Packages and Types Reference

Other advanced guides required by a particular user depend on the area of responsibility of this user. For example, a security officer will naturally refer to the *Oracle Database Security Guide*.

信息来源。

由于高级指南数目过多，不能一一列举。表 1-2 列出了大多数专家数据库管理员和开发人员会用到的指南。

表 1-2 高级组

数据库管理员

《Oracle 数据库管理员指南》

《Oracle 数据库性能优化指南》

《Oracle 数据库备份和恢复用户指南》

《Oracle 真正应用集群管理和部署指南》

数据库开发人员

《Oracle 数据库高级应用程序开发人员指南》

《Oracle 数据库 PL/SQL 语言参考》

《Oracle 数据库 PL/SQL 包和类型参考》

由特定用户所需的其他高级的指南，取决于该用户的责任范围。 例如安全人员自然会参考 《*Oracle 数据库安全指南*》。

2 Tables and Table Clusters

This chapter provides an introduction to schema objects and discusses tables, which are the most common types of schema objects.
This chapter contains the following sections:

- Introduction to Schema Objects
  - Schema Object Types
  - Schema Object Storage
  - Schema Object Dependencies
  - SYS and SYSTEM Schemas
  - Sample Schemas
- Overview of Tables
  - Columns and Rows
  - Example: CREATE TABLE and ALTER TABLE Statements
  - Oracle Data Types
  - Integrity Constraints
  - Object Tables
  - Temporary Tables
  - External Tables
  - Table Storage
- Overview of Table Clusters
  - Overview of Indexed Clusters
  - Overview of Hash Clusters

Introduction to Schema Objects

A database schema is a logical container for data structures, called **schema objects**. Examples of schema objects are tables and indexes. Schema objects are created and manipulated with SQL.
A database user has a password and various database privileges. Each user owns a single schema, which has the same name as the user. The schema contains the data for the user owning the schema. For example, the `hr` user owns the `hr` schema, which contains schema objects such as the `employees` table. In a production database, the schema owner usually represents a database application rather than a person.

第2章 表和表簇

本章介绍了模式对象，并讨论了最常见的模式对象类型——表。

本章包含以下各节：

- 模式对象简介
  - 模式对象类型
  - 模式对象存储
  - 模式对象依赖
  - SYS 和 SYSTEM 模式
  - 示例模式
- 表概述
  - 列和行
  - 示例：CREATE TABLE 和 ALTER TABLE 语句
  - Oracle 数据类型
  - 完整性约束
  - 对象表
  - 临时表
  - 外部表
  - 表存储
- 表簇概述
  - 索引化表簇概述
  - 哈希簇概述

模式对象简介

数据库模式，是被称为**模式对象**的数据结构的逻辑容器。 举例来说，表和索引即是模式对象。 模式对象通过 SQL 来创建和操作。

数据库用户具有口令并拥有各种数据库的权限。 每个用户拥有一个单一的模式，此模式与用户具有相同的名称。模式包含相应用户的全部数据。 例如，hr 用户拥有 hr 模式，此模式包含如雇员表之类的模式对象。 在生产数据库中，一个数据库模式的拥有者通常表示一个数据库应用程序，而不是一个人。

Within a schema, each schema object of a particular type has a unique name. For example, `hr.employees` refers to the table `employees` in the `hr` schema. Figure 2-1 depicts a schema owner named `hr` and schema objects within the `hr` schema.

Figure 2-1 HR Schema

Description of "Figure 2-1 HR Schema"

See Also:

"Overview of Database Security" to learn more about users and privileges

Schema Object Types

The most important schema objects in a relational database are tables. A table stores data in rows.

Oracle SQL enables you to create and manipulate many other types of schema objects, including the following:

- Indexes

  Indexes are schema objects that contains an entry for each indexed row of the table or table cluster and provide direct, fast access to rows. Oracle Database supports several types of index. An index-organized table is a table in which the data is stored in an index structure. See Chapter 3, "Indexes and Index-Organized Tables".
- Partitions

  Partitions are pieces of large tables and indexes. Each partition has its own name and may optionally have its own storage characteristics. See "Overview of Partitions".
- Views

在一个模式里，某一特定类型的每个模式对象都有一个唯一的名称。 例如 hr.employees 是指 hr 模式中的 employees 表。 图 2-1 描述了一个名为 hr 的模式拥有者，及其在 hr 模式内的模式对象。

图 2-1 HR 模式

Description of "Figure 2-1 HR Schema"

另见：

"数据库安全概述"了解用户和权限的更多信息

模式对象类型

在关系数据库中，最重要的模式对象是表。 表以行的形式存储数据。

Oracle SQL 使您能够创建和操作许多其他类型的模式对象，包括以下这些：

- 索引

  索引是一种模式对象，对于每一个被索引的表行或表簇行，索引都包含一个条目，以提供直接、快速的存取。 Oracle 数据库支持几种类型的索引。 一个索引组织表是一个表，其数据以一个索引结构来存储。 请参见第 3 章，"索引和索引组织表"。
- 分区

  分区是大型表和索引的分片。 每个分区有其自己的名称，并可能有其自己（可选）的存储特征。 请参阅"分区概述"。
- 视图

Views are customized presentations of data in one or more tables or other views. You can think of them as stored queries. Views do not actually contain data. See "Overview of Views".

- Sequences

  A sequence is a user-created object that can be shared by multiple users to generate integers. Typically, sequences are used to generate primary key values. See "Overview of Sequences".
- Dimensions

  A dimension defines a parent-child relationship between pairs of column sets, where all the columns of a column set must come from the same table. Dimensions are commonly used to categorize data such as customers, products, and time. See "Overview of Dimensions".
- Synonyms

  A synonym is an alias for another schema object. Because a synonym is simply an alias, it requires no storage other than its definition in the data dictionary. See "Overview of Synonyms".
- PL/SQL subprograms and packages

  PL/SQL is the Oracle procedural extension of SQL. A **PL/SQL subprogram** is a named PL/SQL block that can be invoked with a set of parameters. A **PL/SQL package** groups logically related PL/SQL types, variables, and subprograms. See "PL/SQL Subprograms" and "PL/SQL Packages".

Other types of objects are also stored in the database and can be created and manipulated with SQL statements but are not contained in a schema. These objects include database users, roles, contexts, and directory objects.

See Also:

- *Oracle Database 2 Day DBA* and *Oracle Database Administrator's Guide* to learn how to manage schema objects
- *Oracle Database SQL Language Reference* for more about schema objects and database objects

Schema Object Storage

视图是对一个或多个表、或其他视图中的数据的自定义表示。 你可以把它们看作存储的查询。 视图实际上并不存储数据。 请参见"视图概述"。

- 序列

  序列是一个由用户创建的对象，可以被多个用户共享，用于生成整数。 通常，序列用于生成主键值。 请参阅"序列概述"。

- 维度

  维度定义多个列集之间的父-子关系，列集中的所有列必须都来自同一个表。维度通常用于对客户、 产品、和时间之类的数据进行分类。 请参阅"维度概述"。

- 同义词

  同义词是另一个模式对象的别名。 因为同义词只是一个别名，它在数据字典中除了其定义之外，没有存储。 请参阅"同义词概述"。

- PL/SQL 子程序和包

  PL/SQL 是 Oracle 对 SQL 的过程化扩展。 **PL/SQL 子程序**是命名的 PL/SQL 块，可以带参数调用。 **PL/SQL 包**用于将逻辑上相关的 PL/SQL 类型、 变量、和子程序进行分组。 请参阅"PL/SQL 子程序"和"PL/SQL 包"。

还有一些其他类型的对象也存储在数据库中，并以 SQL 语句来创建和操作，但不包含在模式中。 这些对象包括数据库用户、 角色、 上下文、和目录对象。

另见：

- 《*Oracle 数据库 2 日 DBA*》 和《 *Oracle 数据库管理员指南*》了解如何管理模式对象
- 《*Oracle 数据库 SQL 语言参考*》关于模式对象和数据库对象的详细信息

模式对象存储

Some schema objects store data in logical storage structures called segments. For example, a nonpartitioned heap-organized table or an index creates a segment. Other schema objects, such as views and sequences, consist of metadata only. This section describes only schema objects that have segments.

Oracle Database stores a schema object logically within a tablespace. There is no relationship between schemas and tablespaces: a tablespace can contain objects from different schemas, and the objects for a schema can be contained in different tablespaces. The data of each object is physically contained in one or more data files.

Figure 2-2 shows a possible configuration of table and index segments, tablespaces, and data files. The data segment for one table spans two data files, which are both part of the same tablespace. A segment cannot span multiple tablespaces.

Figure 2-2 Segments, Tablespaces, and Data Files

一些模式对象将数据存储在称为段的逻辑存储结构中。例如，一个未分区的堆组织表或索引会创建一个段。 其他模式对象，如视图和序列，则只包含元数据。 这一节仅描述有段的模式对象。

Oracle 数据库逻辑上将模式对象存储在表空间中。 在模式和表空间之间没有任何的关系： 一个表空间可以包含来自不同的模式的对象，一个模式中的对象也可以包含在不同的表空间中。 每个对象的数据在物理上包含在一个或多个数据文件中。

图 2-2 显示了一种在表、索引段、表空间、和数据文件之间的可能配置。一个表的数据段跨越两个数据文件，这两个数据文件属于同一个表空间。一个段不能跨多个表空间。

图 2-2 段，表空间，和数据文件

Description of "Figure 2-2 Segments, Tablespaces, and Data Files"

See Also:

- Chapter 12, "Logical Storage Structures" to learn about tablespaces and segments
- *Oracle Database 2 Day DBA* and *Oracle Database Administrator's Guide* to learn how to manage storage for schema objects

Schema Object Dependencies

Some schema objects reference other objects, creating **schema object dependencies**. For example, a view contains a query that references tables or other views, while a PL/SQL subprogram invokes other subprograms. If the definition of object A references object B, then A is a **dependent object** with respect to B and B is a **referenced object** with respect to A.

Oracle Database provides an automatic mechanism to ensure that a dependent object is always up to date with respect to its referenced objects. When a dependent object is created, the database tracks dependencies between the dependent object and its referenced objects. When a referenced object changes in a way that might affect a dependent object, the dependent object is marked invalid. For example, if a user drops a table, no view based on the dropped table is usable.

An invalid dependent object must be recompiled against the new definition of a referenced object before the dependent object is usable. Recompilation occurs automatically when the invalid dependent object is referenced.

As an illustration of how schema objects can create dependencies, the following sample script creates a table `test_table` and then a procedure that queries this table:

Description of "Figure 2-2 Segments, Tablespaces, and Data Files"

另见:

- 第 12 章，"逻辑存储结构"了解表空间和段
- 《*Oracle 数据库 2 日 DBA*》 和 《*Oracle 数据库管理员指南*》了解如何管理模式对象的存储

模式对象依赖

一些模式对象会引用其它对象，这就产生了模式对象依赖。 例如，一个视图包含一个引用表或其他视图的查询，或一个 PL/SQL 子程序调用其他子程序。 如果对象 A 的定义引用了对象 B，那么 A 相对于 B 来说是依赖对象，或 B 相对于 A 来说是被引用对象。

Oracle 数据库提供了一种自动的机制，以确保依赖对象对于相应的被引用对象来说始终是最新的。 当依赖对象产生后，数据库将跟踪依赖对象和其被引用对象之间的依赖关系。当被引用对象的更改可能会影响依赖对象时，依赖对象被标记为无效。 例如，当用户删除一个表，所有基于这个被删除表的视图都将变得不可用。

在一个无效的依赖对象重新变得可用之前，必须基于对被引用对象的新定义重新编译。 当无效依赖对象被引用时，重新编译会自动发生。

为演示模式对象之间如何产生依赖，下面的示例脚本创建表 test_table，然后创建一个查询此表的存储过程：

```
CREATE TABLE test_table ( col1 INTEGER, col2 INTEGER );
CREATE OR REPLACE PROCEDURE test_proc
AS
BEGIN
 FOR x IN ( SELECT col1, col2 FROM test_table )
 LOOP
   -- process data
   NULL;
 END LOOP;
END;
/
```

The following query of the status of procedure `test_proc` shows that it is valid:

```
SQL> SELECT OBJECT_NAME, STATUS FROM USER_OBJECTS WHERE
OBJECT_NAME = 'TEST_PROC';

OBJECT_NAME STATUS
----------- -------
TEST_PROC   VALID
```

After adding the `col3` column to `test_table`, the procedure is still valid because the procedure has no dependencies on this column:

```
SQL> ALTER TABLE test_table ADD col3 NUMBER;

Table altered.

SQL> SELECT OBJECT_NAME, STATUS FROM USER_OBJECTS WHERE
OBJECT_NAME = 'TEST_PROC';

OBJECT_NAME STATUS
----------- -------
TEST_PROC   VALID
```

However, changing the data type of the `col1` column, which the `test_proc` procedure depends on in, invalidates the procedure:

```
CREATE TABLE test_table ( col1 INTEGER, col2 INTEGER );
CREATE OR REPLACE PROCEDURE test_proc
AS
BEGIN
 FOR x IN ( SELECT col1, col2 FROM test_table )
 LOOP
   -- process data
   NULL;
 END LOOP;
END;
/
```

下面的查询显示过程 test_proc 的状态是有效的：

```
SQL> SELECT OBJECT_NAME, STATUS FROM USER_OBJECTS WHERE
OBJECT_NAME = 'TEST_PROC';

OBJECT_NAME STATUS
----------- -------
TEST_PROC   VALID
```

向 test_table 表中添加 col3 列之后，该过程仍然是有效的，因为该过程不依赖于该列：

```
SQL> ALTER TABLE test_table ADD col3 NUMBER;

Table altered.

SQL> SELECT OBJECT_NAME, STATUS FROM USER_OBJECTS WHERE
OBJECT_NAME = 'TEST_PROC';

OBJECT_NAME STATUS
----------- -------
TEST_PROC   VALID
```

但是，更改 col1 列的数据类型后，test_proc 过程变得无效，因为该过程依赖于该列。

```
SQL> ALTER TABLE test_table MODIFY col1 VARCHAR2(20);

Table altered.

SQL> SELECT OBJECT_NAME, STATUS FROM USER_OBJECTS WHERE
OBJECT_NAME = 'TEST_PROC';

OBJECT_NAME STATUS
----------- -------
TEST_PROC   INVALID
```

Running or recompiling the procedure makes it valid again, as shown in the following example:

```
SQL> EXECUTE test_proc

PL/SQL procedure successfully completed.

SQL> SELECT OBJECT_NAME, STATUS FROM USER_OBJECTS WHERE
OBJECT_NAME = 'TEST_PROC';

OBJECT_NAME STATUS
----------- -------
TEST_PROC   VALID
```

See Also:

Oracle Database Administrator's Guide and *Oracle Database Advanced Application Developer's Guide* to learn how to manage schema object dependencies

SYS and SYSTEM Schemas

All Oracle databases include default administrative accounts. Administrative accounts are highly privileged and are intended only for DBAs authorized to perform tasks such as starting and stopping the database, managing memory and storage, creating and managing database users, and so on.

The administrative account `SYS` is automatically created when a database is created. This account can perform all database administrative functions. The `SYS` schema stores the base tables and views for the data dictionary. These base tables and views are critical for the operation of Oracle Database. Tables in the `SYS` schema are manipulated only by the

```
SQL> ALTER TABLE test_table MODIFY col1 VARCHAR2(20);

Table altered.

SQL> SELECT OBJECT_NAME, STATUS FROM USER_OBJECTS WHERE
OBJECT_NAME = 'TEST_PROC';

OBJECT_NAME STATUS
----------- -------
TEST_PROC   INVALID
```

运行或重新编译该过程使它重新有效，如下面的示例中所示：

```
SQL> EXECUTE test_proc

PL/SQL procedure successfully completed.

SQL> SELECT OBJECT_NAME, STATUS FROM USER_OBJECTS WHERE
OBJECT_NAME = 'TEST_PROC';

OBJECT_NAME STATUS
----------- -------
TEST_PROC   VALID
```

另见：

《*Oracle 数据库管理员指南*》和 《*Oracle 数据库高级应用开发者指南*》了解如何管理模式对象依赖性

SYS 和 SYSTEM 模式

所有 Oracle 数据库都包括默认管理帐户。 管理帐户享有很高的特权，仅用于授权的数据库管理员执行诸如启停数据库、 管理内存和存储、 创建和管理数据库用户等任务。

SYS 管理帐户在创建数据库时自动创建。 此帐户可以执行所有的数据库管理功能。SYS 模式存储数据字典基表和视图。 这些基表和视图对数据库的运行至关重要。 SYS 模式中的表只由数据库操作，绝不能被任何用户修改。

database and must never be modified by any user.
The SYSTEM account is also automatically created when a database is created. The SYSTEM schema stores additional tables and views that display administrative information, and internal tables and views used by various Oracle Database options and tools. Never use the SYSTEM schema to store tables of interest to nonadministrative users.

See Also:

- "User Accounts" and "Connection with Administrator Privileges"
- *Oracle Database 2 Day DBA* and *Oracle Database Administrator's Guide* to learn about SYS, SYSTEM, and other administrative accounts

Sample Schemas

An Oracle database may include **sample schemas**, which are a set of interlinked schemas that enable Oracle documentation and Oracle instructional materials to illustrate common database tasks. The hr schema is a sample schema that contains information about employees, departments and locations, work histories, and so on.
Figure 2-3 is an entity-relationship diagram of the tables in the hr schema. Most examples in this manual use objects from this schema.

Figure 2-3 HR Schema

Description of "Figure 2-3 HR Schema"

See Also:

在创建数据库时，也会自动创建 SYSTEM 帐户。 SYSTEM 模式存储其它一些用于显示管理信息的表和视图，以及用于各种数据库选项和工具的内部表和视图。 永远不要使用 SYSTEM 模式来存储非管理性用户的表。

另见:

- "用户帐户"和"以管理员权限连接"
- 《*Oracle 数据库 2 日 DBA*》 和 《*Oracle 数据库管理员指南*》了解 SYS, SYSTEM，以及其他管理帐户

示例模式

Oracle 数据库可能包含**示例模式**，这是一组相互关联的模式，在 Oracle 文档和 Oracle 教学材料中用于演示常见数据库任务。 hr 模式是一个包含有关雇员、 部门和位置、 工作历史等信息的示例模式。

图 2-3 是 hr 模式中表的一个实体关系图。 在本手册中的大多数示例都使用此模式中的对象。

图 2-3 HR 模式

Description of "Figure 2-3 HR Schema"

另见:

Overview of Tables

A table is the basic unit of data organization in an Oracle database. A table describes an **entity**, which is something of significance about which information must be recorded. For example, an employee could be an entity.

Oracle Database tables fall into the following basic categories:

- Relational tables

  Relational tables have simple columns and are the most common table type. Example 2-1 shows a `CREATE TABLE` statement for a relational table.
- Object tables

  The columns correspond to the top-level attributes of an object type. See "Object Tables".

You can create a relational table with the following organizational characteristics:

- A heap-organized table does not store rows in any particular order. The `CREATE TABLE` statement creates a heap-organized table by default.
- An index-organized table orders rows according to the primary key values. For some applications, index-organized tables enhance performance and use disk space more efficiently. See "Overview of Index-Organized Tables".
- An **external table** is a read-only table whose metadata is stored in the database but whose data in stored outside the database. See "External Tables".

A table is either **permanent** or **temporary**. A permanent table definition and data persist across sessions. A temporary table definition persists in the same way as a permanent table definition, but the data exists only for the duration of a transaction or session. Temporary tables are useful in applications where a result set must be held temporarily, perhaps because the result is constructed by running multiple operations.

This section contains the following topics:

表概述

表是 Oracle 数据库中的数据组织的基本单位。 一个表描述了一个**实体**，其相关重要信息必须被记录。 例如，雇员可能是一个实体。

Oracle 数据库表分为以下几个基本类别：

- 关系表

  关系表具有简单的列，是最常见的表类型。 示例 2-1 显示了一个用于创建关系表的 CREATE TABLE 语句。
- 对象表

  列对应于对象类型的顶层属性。 请参见“对象表”。

您可以创建一个具有下列组织特征的关系表：

- 堆组织表，它不会以任何特定顺序存储行。 默认情况下，CREATE TABLE 语句创建堆组织表。
- 索引组织表，它按主键值对行进行排序。 对于某些应用程序，索引组织表可以增强性能并更有效地使用的磁盘空间。 请参见"索引组织表概述"。
- **外部表**，是一个只读表，它的元数据存储在数据库中，但其数据存储在数据库外。 请参阅"外部表"。

表要么是**永久**的，要么是**临时**的。 永久表的定义和数据保持跨会话存在。 临时表的定义与永久表有相同的存在方式，但其数据仅在一个事务或会话的持续期间有效。临时表可用于暂存某些应用程序在运行过程中由多个操作所生成的中间结果集。

本部分包含下列主题：

- Columns and Rows
- Example: CREATE TABLE and ALTER TABLE Statements
- Oracle Data Types
- Integrity Constraints
- Object Tables
- Temporary Tables
- External Tables
- Table Storage

See Also:
Oracle Database 2 Day DBA and *Oracle Database Administrator's Guide* to learn how to manage tables

Columns and Rows

A table definition includes a **table name** and set of columns. A column identifies an attribute of the entity described by the table. For example, the column `employee_id` in the `employees` table refers to the employee ID attribute of an employee entity.

In general, you give each column a **column name**, a data type, and a **width** when you create a table. For example, the data type for `employee_id` is `NUMBER(6)`, indicating that this column can only contain numeric data up to 6 digits in width. The width can be predetermined by the data type, as with `DATE`.

A table can contain a virtual column, which unlike a nonvirtual column does not consume disk space. The database derives the values in a virtual column on demand by computing a set of user-specified expressions or functions. For example, the virtual column `income` could be a function of the `salary` and `commission_pct` columns.

After you create a table, you can insert, query, delete, and update rows using SQL. A row is a collection of column information corresponding to a record in a table. For example, a row in the `employees` table describes the attributes of a specific employee.

See Also:
Oracle Database Administrator's Guide to learn how to manage virtual columns

Example: CREATE TABLE and ALTER TABLE Statements

The Oracle SQL command to create a table is `CREATE TABLE`. Example 2-1 shows the `CREATE TABLE` statement for the `employees` table in

- 列和行
- 示例：CREATE TABLE 和 ALTER TABLE 语句
- Oracle 数据类型
- 完整性约束
- 对象表
- 临时表
- 外部表
- 表存储

另见:
《*Oracle 数据库 2 日 DBA*》和《*Oracle 数据库管理员指南*》了解如何管理表

列和行

表的定义包括表名称和列集。 列标识由表所描述的实体的一个属性。 例如，在雇员表中的列 employee_id，是指雇员实体中的雇员 ID 属性。

一般地，当您将创建一个表时，给每列一个列名称、 数据类型、和宽度。 例如 employee_id 的数据类型是 NUMBER(6)，这表明此列只可以包含长达 6 位数的数字数据宽度。与 DATE 类型一样，宽度可以由数据类型预先确定。

表可以包含虚拟的列，与非虚拟列不一样，虚拟列不占用磁盘空间。 数据库通过计算一组用户指定的表达式或函数，按需派生出虚拟的列值。 例如虚拟列 income 可能是 salary 列和 commission_pct 列的一个函数。

创建一个表后，可以使用 SQL 插入、 查询、 删除、和更新行。 行是对应于表中某条记录的列信息的集合。 例如雇员表中的一行描述一个特定的雇员的属性。

另见：
《*Oracle 数据库管理员指南*》了解如何管理虚拟列

示例：CREATE TABLE 和 ALTER TABLE 语句

用于创建表的 Oracle SQL 命令是 `CREATE TABLE`。 示例 2-1 显示了 hr 示例模式中用于创建雇员表的 CREATE TABLE 语句。 该语句指定了

the hr sample schema. The statement specifies columns such as employee_id, first_name, and so on, specifying a data type such as NUMBER or DATE for each column.

Example 2-1 CREATE TABLE employees

```
CREATE TABLE employees
    ( employee_id    NUMBER(6)
    , first_name     VARCHAR2(20)
    , last_name      VARCHAR2(25)
        CONSTRAINT     emp_last_name_nn  NOT NULL
    , email          VARCHAR2(25)
      CONSTRAINT     emp_email_nn  NOT NULL
    , phone_number   VARCHAR2(20)
    , hire_date      DATE
      CONSTRAINT     emp_hire_date_nn  NOT NULL
    , job_id         VARCHAR2(10)
      CONSTRAINT     emp_job_nn  NOT NULL
    , salary         NUMBER(8,2)
    , commission_pct NUMBER(2,2)
    , manager_id     NUMBER(6)
    , department_id  NUMBER(4)
    , CONSTRAINT     emp_salary_min
                     CHECK (salary > 0)
    , CONSTRAINT     emp_email_uk
                     UNIQUE (email)
    ) ;
```

Example 2-2 shows an ALTER TABLE statement that adds integrity constraints to the employees table. Integrity constraints enforce business rules and prevent the entry of invalid information into tables.

Example 2-2 ALTER TABLE employees

employee_id、 first_name 等列，并为每个列指定了数据类型，如 NUMBER 或 DATE 等。

示例 2-1 CREATE TABLE employees

```
CREATE TABLE employees
    ( employee_id    NUMBER(6)
    , first_name     VARCHAR2(20)
    , last_name      VARCHAR2(25)
        CONSTRAINT     emp_last_name_nn  NOT NULL
    , email          VARCHAR2(25)
      CONSTRAINT     emp_email_nn  NOT NULL
    , phone_number   VARCHAR2(20)
    , hire_date      DATE
      CONSTRAINT     emp_hire_date_nn  NOT NULL
    , job_id         VARCHAR2(10)
      CONSTRAINT     emp_job_nn  NOT NULL
    , salary         NUMBER(8,2)
    , commission_pct NUMBER(2,2)
    , manager_id     NUMBER(6)
    , department_id  NUMBER(4)
    , CONSTRAINT     emp_salary_min
                     CHECK (salary > 0)
    , CONSTRAINT     emp_email_uk
                     UNIQUE (email)
    ) ;
```

示例 2-2 显示了 ALTER TABLE 语句，它向雇员表添加了一个完整性约束。完整性约束强制实施业务规则和防止向表中输入无效信息。

示例 2-2 ALTER TABLE employees

```
ALTER TABLE employees
ADD ( CONSTRAINT     emp_emp_id_pk
                       PRIMARY KEY (employee_id)
    , CONSTRAINT     emp_dept_fk
                       FOREIGN KEY (department_id)
                         REFERENCES departments
    , CONSTRAINT     emp_job_fk
                       FOREIGN KEY (job_id)
                         REFERENCES jobs (job_id)
    , CONSTRAINT     emp_manager_fk
                       FOREIGN KEY (manager_id)
                         REFERENCES employees
    ) ;
```

Example 2-3 shows 8 rows and 6 columns of the `hr.employees` table.

Example 2-3 Rows in the employees Table

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	COMMISSION_PCT	DEPARTMENT_ID
100	Steven	King	24000		90
101	Neena	Kochhar	17000		90
102	Lex	De Haan	17000		90
103	Alexander	Hunold	9000		60
107	Diana	Lorentz	4200		60
149	Eleni	Zlotkey	10500	.2	80
174	Ellen	Abel	11000	.3	80
178	Kimberely	Grant	7000	.15	

The output in Example 2-3 illustrates some of the following important characteristics of tables, columns, and rows:

- A row of the table describes the attributes of one employee: name, salary, department, and so on. For example, the first row in the output shows the record for the employee named Steven King.

- A column describes an attribute of the employee. In the example, the `employee_id` column is the primary key, which means that every employee is uniquely identified by employee ID. Any two employees are guaranteed not to have the same employee ID.

- A non-key column can contain rows with identical values. In the example, the salary value for employees 101 and 102 is the same:


```
ALTER TABLE employees
ADD ( CONSTRAINT     emp_emp_id_pk
                       PRIMARY KEY (employee_id)
    , CONSTRAINT     emp_dept_fk
                       FOREIGN KEY (department_id)
                         REFERENCES departments
    , CONSTRAINT     emp_job_fk
                       FOREIGN KEY (job_id)
                         REFERENCES jobs (job_id)
    , CONSTRAINT     emp_manager_fk
                       FOREIGN KEY (manager_id)
                         REFERENCES employees
    ) ;
```


示例 2-3 显示了 hr.employees 表中的 8 行和 6 列。

示例 2-3 employees 表中的行

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	SALARY	COMMISSION_PCT	DEPARTMENT_ID
100	Steven	King	24000		90
101	Neena	Kochhar	17000		90
102	Lex	De Haan	17000		90
103	Alexander	Hunold	9000		60
107	Diana	Lorentz	4200		60
149	Eleni	Zlotkey	10500	.2	80
174	Ellen	Abel	11000	.3	80
178	Kimberely	Grant	7000	.15	

示例 2-3 中的输出说明了表、 行、和列的如下重要特性：

- 表中的行描述一名雇员的全部属性： 名称、 薪金、 部门，等等。 例如，输出的第一行显示名为 Steven King 的雇员记录。

- 列描述该雇员的一个属性。 在本例中， employee_id 列是主键，其含义是指每一位员工由其雇员 id 唯一地标识，保证任何两个雇员都不具有相同的雇员 id。

- 非键列可以包含具有相同值的行。 在本例中，雇员 101、 102 的薪金

17000.

- A foreign key column refers to a primary or unique key in the same table or a different table. In this example, the value of 90 in department_id corresponds to the department_id column of the departments table.

- A **field** is the intersection of a row and column. It can contain only one value. For example, the field for the department ID of employee 104 contains the value 60.

- A field can lack a value. In this case, the field is said to contain a null value. The value of the commission_pct column for employee 100 is null, whereas the value in the field for employee 149 is .2. A column allows nulls unless a NOT NULL or primary key integrity constraint has been defined on this column, in which case no row can be inserted without a value for this column.

See Also:
Oracle Database SQL Language Reference for CREATE TABLE syntax and semantics

Oracle Data Types

Each column has a **data type**, which is associated with a specific storage format, constraints, and valid range of values. The data type of a value associates a fixed set of properties with the value. These properties cause Oracle Database to treat values of one data type differently from values of another. For example, you can multiply values of the NUMBER data type, but not values of the RAW data type.

When you create a table, you must specify a data type for each of its columns. Each value subsequently inserted in a column assumes the column data type.

Oracle Database provides several built-in data types. The most commonly used data types fall into the following categories:

- Character Data Types
- Numeric Data Types
- Datetime Data Types
- Rowid Data Types

相同： 17000。

- 外键列会引用同一个表或另一个表中的主键或唯一键。 在本例中，department_id 值 90 对应于部门表中的 department_id 列等于 90 的行。

- 域是行和列的交叉点。 它只可以包含一个值。 例如，雇员 104 的部门 ID 域包含值 60。

- 域可以没有值。 在这种情况下，该域包含空值。 雇员 100 的 commission_pct 列的值为空，而雇员 149 的相应值是.2。一个列通常允许为空，除非已对该列定义了 NOT NULL 或主键完整性约束，在这种情况下如果该列无值，是不可以插入任何行的。

另见：
《*Oracle 数据库 SQL 语言参考*》关于 CREATE TABLE 的语法和语义

Oracle 数据类型

每个列都有一种数据类型，这与特定的存储格式、限制、和有效的值范围相关联。 一个值的数据类型与一组固定的属性相关联。 这些属性会使 Oracle 数据库区别对待不同数据类型的值。例如，你可以把 NUMBER 数据类型的值相乘，但不能把 RAW 数据类型的值相乘。

在创建表时，您必须为每个列指定数据类型。 随后在该列中插入的每个值都假定为该列的数据类型。

Oracle 数据库提供了几种内建的数据类型。 最常用的数据类型分为以下几类：

- 字符数据类型
- 数字数据类型
- 日期时间数据类型
- ROWID 数据类型

- Format Models and Data Types

Other important categories of built-in types include raw, large objects (LOBs), and collections. PL/SQL has data types for constants and variables, which include `BOOLEAN`, reference types, composite types (records), and user-defined types.

See Also:

- "Overview of LOBs"

- *Oracle Database SQL Language Reference* to learn about built-in SQL data types
- *Oracle Database PL/SQL Language Reference* to learn about PL/SQL data types
- *Oracle Database Advanced Application Developer's Guide* for information about how to use the built-in data types

Character Data Types

Character data types store character (alphanumeric) data in strings. The most commonly used character data type is `VARCHAR2`, which is the most efficient option for storing character data.

The byte values correspond to the character encoding scheme, generally called a **character set** or **code page**. The database character set is established at database creation. Examples of character sets are 7-bit ASCII, EBCDIC, and Unicode UTF-8.

The length semantics of character data types can be measured in bytes or characters. **Byte semantics** treat strings as a sequence of bytes. This is the default for character data types. **Character semantics** treat strings as a sequence of characters. A character is technically a code point of the database character set.

See Also:

- "Character Sets"
- *Oracle Database 2 Day Developer's Guide* and *Oracle Database Advanced Application Developer's Guide* and to learn how to select a character data type

VARCHAR2 and CHAR Data Types

The `VARCHAR2` data type stores variable-length character literals. The terms **literal** and **constant value** are synonymous and refer to a fixed data value. For example, `'LILA'`, `'St. George Island'`, and

- 格式模型和数据类型

其他几种重要的内建类型包括 RAW、大对象 (LOB)、 和集合。 PL/SQL 具有用于常量和变量的数据类型，包括布尔、 引用类型、 复合类型 （记录） 、和用户定义类型。

另见：

- "大对象概述"

- 《*Oracle 数据库 SQL 语言参考*》了解内置的 SQL 数据类型

- 《*Oracle 数据库 PL/SQL 语言参考*》了解 PL/SQL 数据类型

- 《*Oracle 数据库高级应用开发指南*》关于如何使用内置数据类型的信息

字符数据类型

字符数据类型存储在字符串中的字符 （字母或数字） 数据。 最常用的字符数据类型是 VARCHAR2，它是用于存储字符数据的最有效的选项。

与字符编码模式相对应的字节值，一般称为**字符集**或**代码页**。 数据库字符集是在数据库创建时建立的。字符集的例子有 7 位 ASCII 码、 EBCDIC 码、和 Unicode utf-8。

字符数据类型的长度语义可以以字节或字符为单位。 **字节语义**将字符串视为一个字节序列。 这是字符数据类型的默认值。 **字符语义**将字符串视为字符序列。 一个字符在技术上是数据库字符集的编码点。

另见：

- "字符集"
- 《*Oracle 数据库 2 日开发人员指南*》 和 《*Oracle 数据库高级应用开发指南*》学习如何选择字符数据类型

VARCHAR2 和 CHAR 数据类型

VARCHAR2 数据类型存储变长字符文本。**文本**和**常量值**这两个术语是同义词，指的是一个固定的数据值。例如'LILA'、 ' St. George Island ' 和 '101'，都是字符文本 ； 5001 是数字文本。字符文本包括在单引号中，以便

'101' are all character literals; 5001 is a numeric literal. Character literals are enclosed in single quotation marks so that the database can distinguish them from schema object names.

数据库可以将它们与模式对象名称区分开来。

Note:

This manual uses the terms **text literal**, **character literal**, and **string** interchangeably.

注意：

本手册中，**文本、 文本字符和字符串**等术语的使用具有互换性。

When you create a table with a VARCHAR2 column, you specify a maximum string length. In Example 2-1, the last_name column has a data type of VARCHAR2(25), which means that any name stored in the column can have a maximum of 25 bytes.

当创建一个有 VARCHAR2 列的表时，指定该列的最大字符串长度。在示例 2-1 中，last_name 列是 VARCHAR2(25)数据类型，这意味着该列中存储的名字最多可以长达 25 个 字节。

For each row, Oracle Database stores each value in the column as a variable-length field unless a value exceeds the maximum length, in which case the database returns an error. For example, in a single-byte character set, if you enter 10 characters for the last_name column value in a row, then the column in the row piece stores only 10 characters (10 bytes), not 25. Using VARCHAR2 reduces space consumption.

对每一行， Oracle 数据库将按可变长度字段来存储值，如果一个值超出最大长度，数据库将返回一个错误。例如，在单字节字符集中，如果您在某行的 last_name 列输入 10 个字符，那么在行片中该列只存储 10 个字符 （10 个字节） 而不是 25。使用 VARCHAR2 可以减少空间占用。

In contrast to VARCHAR2, CHAR stores fixed-length character strings. When you create a table with a CHAR column, the column requires a string length. The default is 1 byte. The database uses blanks to pad the value to the specified length.

与 VARCHAR2 不同， CHAR 存储固定长度的字符串。当创建一个有 CHAR 列的表时，需要为该列指定一个字符串长度。默认值是 1 个字节。数据库使用空格填充到指定的长度的值。

Oracle Database compares VARCHAR2 values using nonpadded comparison semantics and compares CHAR values using blank-padded comparison semantics.

Oracle 数据库将使用非填充比较语义来比较 VARCHAR2 值，而使用空白填充比较语义来比较 CHAR 值。

See Also:

Oracle Database SQL Language Reference for details about blank-padded and nonpadded comparison semantics

另见：

《*Oracle 数据库 SQL 语言参考*》了解有关空格填充和非填充比较语义的详细信息

NCHAR and NVARCHAR2 Data Types

The NCHAR and NVARCHAR2 data types store Unicode character data. **Unicode** is a universal encoded character set that can store information in any language using a single character set. NCHAR stores fixed-length character strings that correspond to the national character set, whereas NVARCHAR2 stores variable length character strings.

NCHAR 和 NVARCHAR2 数据类型

NVARCHAR2 和 NCHAR 数据类型存储 **Unicode** 字符数据。Unicode 是一种通用的编码字符集，可以将任何语言的信息存储在一种字符集中。NCHAR 存储对应于国家字符集的固定长度字符串，而 NVARCHAR2 存储可变长度的字符串。

You specify a national character set when creating a database. The character set of NCHAR and NVARCHAR2 data types must be either AL16UTF16 or UTF8. Both character sets use Unicode encoding.

当您创建数据库时，指定国家字符集。NCHAR 和 NVARCHAR2 数据类型的字符集必须是 AL16UTF16 或 UTF8。这两种字符集都使用 Unicode 编码。

When you create a table with an NCHAR or NVARCHAR2 column, the maximum size is always in character length semantics. Character length

当您创建一个有 NCHAR 或 NVARCHAR2 列的表时，其最大大小始终是字符长度语义。对于 NCHAR 或 NVARCHAR2，字符长度语义是默认的，也

semantics is the default and only length semantics for NCHAR or NVARCHAR2.

See Also:

Oracle Database Globalization Support Guide for information about Oracle's globalization support feature

Numeric Data Types

The Oracle Database numeric data types store fixed and floating-point numbers, zero, and infinity. Some numeric types also store values that are the undefined result of an operation, which is known as "not a number" or NAN.

Oracle Database stores numeric data in variable-length format. Each value is stored in scientific notation, with 1 byte used to store the exponent. The database uses up to 20 bytes to store the **mantissa**, which is the part of a floating-point number that contains its significant digits. Oracle Database does not store leading and trailing zeros.

NUMBER Data Type

The NUMBER data type stores fixed and floating-point numbers. The database can store numbers of virtually any magnitude. This data is guaranteed to be portable among different operating systems running Oracle Database. The NUMBER data type is recommended for most cases in which you must store numeric data.

You specify a fixed-point number in the form NUMBER(*p*,*s*), where *p* and *s* refer to the following characteristics:

- Precision

  The **precision** specifies the total number of digits. If a precision is not specified, then the column stores the values exactly as provided by the application without any rounding.

- Scale

  The **scale** specifies the number of digits from the decimal point to the least significant digit. **Positive scale** counts digits to the right of the decimal point up to and including the least significant digit. **Negative scale** counts digits to the left of the decimal point up to but not including the least significant digit. If you specify a precision without a scale, as in NUMBER(6), then the scale is 0.

In Example 2-1, the salary column is type NUMBER(8,2), so the precision is 8 and the scale is 2. Thus, the database stores a salary of

是唯一的长度语义。

另见：

《*Oracle 数据库全球支持指南*》了解 Oracle 全球化支持功能的信息

数字数据类型

Oracle 数据库的数字数据类型存储固定和浮点数字、零、或无穷。某些数值类型也可以存储未定义操作的结果值，叫做"非数字"或 NAN。

Oracle 数据库以变长格式存储数字数据。用科学计数法存储值，其中一个字节用于存储指数。数据库使用最多 20 个字节存储尾数，即浮点数的有效位数部分。Oracle 数据库不会存储前导零和结尾零。

NUMBER 数据类型

NUMBER 数据类型存储固定和浮点数字。数据库可以存储几乎任何规模的数字。此数据保证在运行 Oracle 数据库的不同操作系统之间可移植。大多数情况下，当您必须存储数值数据时，推荐使用 NUMBER 数据类型。

按 NUMBER(p,s)的形式定义定点数，p 和 s 有以下特征：

- 精度

  精度指定数字的总长度。如果不指定精度，则列按应用程序提供的数据存储值，不作任何舍入。

- 小数位数

  小数位数指从十进制的小数点到最小有效数字的位数。正小数位数从小数点向右计数直至最小有效位。负小数位数从小数点向左计数直至（但不包括）最小有效数字。如果您指定了精度但没有指定小数位数，如 NUMBER(6)，那么小数位数为 0。

在示例 2-1 中，工资列是 NUMBER(8,2)类型，精度为 8，而小数位数为 2。因此，数据库将 10 万元的薪金存储为 100000.00 。

100,000 as `100000.00`.

Floating-Point Numbers

Oracle Database provides two numeric data types exclusively for floating-point numbers: `BINARY_FLOAT` and `BINARY_DOUBLE`. These types support all of the basic functionality provided by the `NUMBER` data type. However, while `NUMBER` uses decimal precision, `BINARY_FLOAT` and `BINARY_DOUBLE` use binary precision, which enables faster arithmetic calculations and usually reduces storage requirements.

`BINARY_FLOAT` and `BINARY_DOUBLE` are approximate numeric data types. They store approximate representations of decimal values, rather than exact representations. For example, the value 0.1 cannot be exactly represented by either `BINARY_DOUBLE` or `BINARY_FLOAT`. They are frequently used for scientific computations. Their behavior is similar to the data types `FLOAT` and `DOUBLE` in Java and XMLSchema.

See Also:

Oracle Database SQL Language Reference to learn about precision, scale, and other characteristics of numeric types

Datetime Data Types

The **datetime** data types are `DATE` and `TIMESTAMP`. Oracle Database provides comprehensive time zone support for time stamps.

DATE Data Type

The `DATE` data type stores date and time. Although datetimes can be represented in character or number data types, `DATE` has special associated properties. The `hire_date` column in Example 2-1 has a `DATE` data type.

The database stores dates internally as numbers. Dates are stored in fixed-length fields of 7 bytes each, corresponding to century, year, month, day, hour, minute, and second.

Note:

Dates fully support arithmetic operations, so you add to and subtract from dates just as you can with numbers. See *Oracle Database Advanced Application Developer's Guide*.

The database displays dates according to the specified **format model**. A format model is a character literal that describes the format of a datetime in a character string. The standard date format is `DD-MON-RR`, which displays dates in the form `01-JAN-09`.

`RR` is similar to `YY` (the last two digits of the year), but the century of the

浮点数

Oracle 数据库为浮点数提供了两种互斥的数值数据类型： BINARY_FLOAT 和 BINARY_DOUBLE。这两种类型支持所有 NUMBER 数据类型提供的基本功能。然而，相比 NUMBER 使用十进制精度，BINARY_FLOAT 和 BINARY_DOUBLE 使用二进制精度，可以使算术计算更快并且通常可以降低存储需求。

BINARY_FLOAT 和 BINARY_DOUBLE 是近似数字数据类型。它们存储十进制的值的近似表示，而不是精确的表示形式。例如值 0.1 不能用 BINARY_DOUBLE 或 BINARY_FLOAT 完全精确地表示。他们经常用于科学计算。其行为类似于 Java 或 XMLSchema 中的 `FLOAT` 和 `DOUBLE` 数据类型。

另见：

《*Oracle 数据库 SQL 语言参考*》了解有关精度、小数位数、和数值类型的其他特性

日期时间数据类型

日期时间数据类型包括 DATE 和 TIMESTAMP。Oracle 数据库为时间戳提供全面的时区支持。

DATE 数据类型

DATE 数据类型存储日期和时间。尽管日期时间可以用字符或数字数据类型表示，DATE 具有特殊的相关属性。示例 2-1 中的 hire_date 列是一个 DATE 数据类型。

数据库在内部将日期存储为数字。日期被存储为固定长度的域，共 7 个字节，分别对应世纪、 年、 月、 日、 小时、分、和秒。

注意：

日期完全支持算术操作，您可以像对数字一样，对日期进行加减操作。请参阅《*Oracle 数据库高级应用程序开发人员指南*》

数据库根据指定的**格式模型**显示日期。格式模型是一个描述在字符串中的日期时间格式的字符文字。标准的日期格式是 DD-MON-RR，它会以 01-JAN-09 的形式显示日期。

RR 类似于 YY （年的最后两位），但返回值的世纪值会因指定的两位年份

return value varies according to the specified two-digit year and the last two digits of the current year. Assume that in 1999 the database displays `01-JAN-09`. If the date format uses `RR`, then `09` specifies `2009`, whereas if the format uses `YY`, then `09` specifies `1909`. You can change the default date format at both the instance and the session level.

Oracle Database stores time in 24-hour format—`HH:MI:SS`. If no time portion is entered, then by default the time in a date field is `00:00:00 A.M`. In a time-only entry, the date portion defaults to the first day of the current month.

See Also:

- *Oracle Database Advanced Application Developer's Guide* for more information about centuries and date format masks
- *Oracle Database SQL Language Reference* for information about datetime format codes

TIMESTAMP Data Type

The `TIMESTAMP` data type is an extension of the `DATE` data type. It stores fractional seconds in addition to the information stored in the `DATE` data type. The `TIMESTAMP` data type is useful for storing precise time values, such as in applications that must track event order.

The `DATETIME` data types `TIMESTAMP WITH TIME ZONE` and `TIMESTAMP WITH LOCAL TIME ZONE` are time-zone aware. When a user selects the data, the value is adjusted to the time zone of the user session. This data type is useful for collecting and evaluating date information across geographic regions.

See Also:

Oracle Database SQL Language Reference for details about the syntax of creating and entering data in time stamp columns

Rowid Data Types

Every row stored in the database has an address. Oracle Database uses a `ROWID` data type to store the address (rowid) of every row in the database. Rowids fall into the following categories:

- **Physical rowids** store the addresses of rows in heap-organized tables, table clusters, and table and index partitions.

- **Logical rowids** store the addresses of rows in index-organized tables.

及本年度的最后两位不同而不同。假设在 1999 年，数据库显示为 01-JAN-09。如果日期格式使用 RR，那么 09 指的是 2009，而如果格式使用 YY，那么 09 指的是 1909。您可以更改实例或会话级别的默认日期格式。

Oracle 数据库以 24 小时制格式存储时间 — — HH:MI:SS。如果不输入的任何时间部分，那么默认情况下，日期字段中的时间是凌晨 00: 00: 00，如果只输入了时间，则日期部分默认为当月的第一天。

另见：

- 《*Oracle 数据库高级应用开发者指南*》了解有关世纪和日期格式掩码的详细信息
- 《*Oracle 数据库 SQL 语言参考*》了解日期时间格式代码的有关信息

TIMESTAMP 数据类型

`TIMESTAMP` 数据类型是日期数据类型的扩展。除了存储在 `DATE` 数据类型的信息，它同时还存储秒的小数部分。`TIMESTAMP` 数据类型对于存储精确时间的值很有用，比如那些必须跟踪事件顺序的应用程序。

`DATETIME` 数据类型 `TIMESTAMP WITH TIME ZONE` 和 `TIMESTAMP WITH LOCAL TIME ZONE` 能感知时区。当用户中选择数据时，其值会被调整以适配用户会话的时区。此数据类型可用于收集和评估各地理区域的日期信息。

另见：

《*Oracle 数据库 SQL 语言参考*》了解有关创建时间戳列，和往其中输入数据的语法的详细信息

Rowid 数据类型

存储在数据库中的每一行都有一个地址。Oracle 数据库使用 ROWID 数据类型存储在数据库中的每一行的地址 (rowid)。Rowids 分为以下几类：

- **物理 rowids** 存储堆组织表、 表簇、表分区、和索引分区中的行地址。

- **逻辑 rowids** 存储索引组织表中的行地址。

- **Foreign rowids** are identifiers in foreign tables, such as DB2 tables accessed through a gateway. They are not standard Oracle Database rowids.

A data type called the **universal rowid**, or `UROWID`, supports all kinds of rowids.

Use of Rowids

Oracle Database uses rowids internally for the construction of indexes. A B-tree index, which is the most common type, contains an ordered list of keys divided into ranges. Each key is associated with a rowid that points to the associated row's address for fast access. End users and application developers can also use rowids for several important functions:

- Rowids are the fastest means of accessing particular rows.
- Rowids provide the ability to see how a table is organized.
- Rowids are unique identifiers for rows in a given table.

You can also create tables with columns defined using the `ROWID` data type. For example, you can define an exception table with a column of data type `ROWID` to store the rowids of rows that violate integrity constraints. Columns defined using the `ROWID` data type behave like other table columns: values can be updated, and so on.

ROWID Pseudocolumn

Every table in an Oracle database has a **pseudocolumn** named `ROWID`. A pseudocolumn behaves like a table column, but is not actually stored in the table. You can select from pseudocolumns, but you cannot insert, update, or delete their values. A pseudocolumn is also similar to a SQL function without arguments. Functions without arguments typically return the same value for every row in the result set, whereas pseudocolumns typically return a different value for each row.

Values of the `ROWID` pseudocolumn are strings representing the address of each row. These strings have the data type `ROWID`. This pseudocolumn is not evident when listing the structure of a table by executing `SELECT` or `DESCRIBE`, nor does the pseudocolumn consume space. However, the rowid of each row can be retrieved with a SQL query using the reserved word `ROWID` as a column name.

Example 2-4 queries the `ROWID` pseudocolumn to show the rowid of the row in the `employees` table for employee 100.

- **外部 rowids** 是外来表（如通过网关访问的 DB2 表）中的标识符。他们不是标准的 Oralce 数据库 rowids。

有一种数据类型称为**通用 rowid** 或 UROWID，支持各种 rowids。

使用 Rowid

Oracle 数据库在内部使用 rowids 用于构造索引，最常见的 B 树索引，包含一个被划分成多个键范围的排序列表。每个键都与一个指向关联的行地址的 rowid 相关联，用于快速访问。最终用户和应用程序开发人员也可以使用 rowids 的几个重要功能：

- Rowids 是访问特定行的最快方式。
- Rowids 提供了查看表的组织方式的能力。
- Rowids 是给定表中的行的唯一标识符。

您也可以创建带 ROWID 数据类型列的表。例如，您可以定义一个异常表，其中一列为 ROWID 数据类型，以存储违反了完整性约束的行 rowids。使用 ROWID 数据类型定义的列像其他表中的列一样， 其值可以更新，等等。

ROWID 伪列

Oracle 数据库中的每个表都有一个名为 ROWID 的 **伪列**。 伪列类似于表列，但实际上并不存储在表中。您可以从伪列中选择数据，但不能插入、更新、或删除它们的值。伪列也类似于不带参数的 SQL 函数。不带参数的函数对于结果集中的每一行通常返回相同的值，而伪列通常为每个行返回不同的值。

ROWID 伪列的值是表示的每个行的地址的字符串。这些字符串具有的数据类型 ROWID。在执行 SELECT 或 DESCRIBE 列出表的结构时，此伪列并不显示，它也不占用空间。但是，每个行的 rowid 可以通过使用保留字 ROWID 作为列名的 SQL 查询来检索。

示例 2-4 查询显示雇员表中雇员 ID 为 100 的行的 ROWID 伪列。

Example 2-4 ROWID Pseudocolumn

```
SQL> SELECT ROWID FROM employees WHERE employee_id = 100;

ROWID
------------------
AAAPecAAFAAAABSAAA
```

See Also:

- "Rowid Format"
- *Oracle Database Advanced Application Developer's Guide* to learn how to identify rows by address
- *Oracle Database SQL Language Reference* to learn about rowid types

Format Models and Data Types

A **format model** is a character literal that describes the format of datetime or numeric data stored in a character string. A format model does not change the internal representation of the value in the database. When you convert a character string into a date or number, a format model determines how the database interprets the string. In SQL, you can use a format model as an argument of the `TO_CHAR` and `TO_DATE` functions to format a value to be returned from the database or to format a value to be stored in the database.

The following statement selects the salaries of the employees in Department 80 and uses the `TO_CHAR` function to convert these salaries into character values with the format specified by the number format model `'$99,990.99'`:

```
SQL> SELECT last_name employee, TO_CHAR(salary, '$99,990.99')
  2  FROM   employees
  3  WHERE  department_id = 80 AND last_name = 'Russell';

EMPLOYEE                  TO_CHAR(SAL
------------------------- -----------
Russell                    $14,000.00
```

The following example updates a hire date using the `TO_DATE` function with the format mask `'YYYY MM DD'` to convert the string `'1998 05 20'` to a `DATE` value:

示例 2-4 ROWID 伪列

```
SQL> SELECT ROWID FROM employees WHERE employee_id = 100;

ROWID
------------------
AAAPecAAFAAAABSAAA
```

另见：

- "Rowid 格式"
- 《*Oracle 数据库高级应用开发指南*》了解如何通过地址标识行
- 《*Oracle 数据库 SQL 语言参考*》了解 rowid 类型

格式模型与数据类型

格式模型是一个字符文本，用来描述存储在一个字符串中的日期时间或数值数据的格式。格式模式不会更改数据库中的值的内部表示形式。

当您将一个字符串转换成日期或数字时，格式模型决定了数据库如何解释字符串。在 SQL 中，你可以使用格式模型作为 TO_CHAR 和 TO_DATE 函数的参数，来格式化将从数据库中返回的一个值，或将存储到数据库中的一个值。

下面的语句选择部门 80 中的雇员的薪水，并使用 TO_CHAR 函数来将这些薪水的值，转换为以数字格式模型 '$ 99,990.99' 指定的字符值：

```
SQL> SELECT last_name employee, TO_CHAR(salary, '$99,990.99')
  2  FROM   employees
  3  WHERE  department_id = 80 AND last_name = 'Russell';

EMPLOYEE                  TO_CHAR(SAL
------------------------- -----------
Russell                    $14,000.00
```

下面的示例更新一个雇佣日期，使用 TO_DATE 函数与格式掩码 `'YYYY MM DD'` 将字符串`'1998 05 20'`转换为日期值。

```
SQL> UPDATE employees
  2  SET hire_date = TO_DATE('1998 05 20','YYYY MM DD')
  3  WHERE last_name = 'Hunold';
```

See Also:

Oracle Database SQL Language Reference to learn more about format models

Integrity Constraints

Integrity constraints are named rules that restrict the values for one or more columns in a table. These rules prevent invalid data entry into tables. Also, constraints can prevent the deletion of a table when certain dependencies exist.

If a constraint is enabled, then the database checks data as it is entered or updated. Data that does not conform to the constraint is prevented from being entered. If a constraint is disabled, then data that does not conform to the constraint can be allowed to enter the database.

In Example 2-1, the `CREATE TABLE` statement specifies `NOT NULL` constraints for the `last_name`, `email`, `hire_date`, and `job_id` columns. The constraint clauses identify the columns and the conditions of the constraint. These constraints ensure that the specified columns contain no null values. For example, an attempt to insert a new employee without a job ID generates an error.

You can create a constraint when or after you create a table. Constraints can be temporarily disabled if needed. The database stores constraints in the data dictionary.

See Also:

- Chapter 5, "Data Integrity" to learn about integrity constraints
- *Oracle Database SQL Language Reference* to learn about SQL constraint clauses

Object Tables

An Oracle **object type** is a user-defined type with a name, attributes, and methods. Object types make it possible to model real-world entities such as customers and purchase orders as objects in the database.

An object type defines a logical structure, but does not create storage. Example 2-5 creates an object type named `department_typ`.

Example 2-5 Object Type

```
SQL> UPDATE employees
  2  SET hire_date = TO_DATE('1998 05 20','YYYY MM DD')
  3  WHERE last_name = 'Hunold';
```

另见：

《*Oracle 数据库 SQL 语言参考*》了解格式模型

完整性约束

完整性约束是一种命名规则，用于限制一个或多个表中的列值。这些规则可以防止向表中输入无效数据。此外，约束也可以防止当存在某些依赖项时删除表中的数据。

如果启用了约束，则输入或更新数据时，数据库会检查数据。不符合该约束的数据会被阻止输入。如果禁用了约束，则不符合该约束的数据可以被允许输入数据库。

在示例 2-1 中，CREATE TABLE 语句对 last_name、 email、 hire_date、和 job_id 列指定了非空约束。约束子句标识列和约束的条件。这些约束确保指定的列中不包含的空值。例如，试图插入一个没有 job ID 的新雇员将产生一个错误。

您可以在创建表时或创建表后创建一个约束。如果需要，可以暂时禁用约束。数据库将约束存储在数据字典中。

另见：

- 第 5 章，数据完整性 了解完整性约束
- 《*Oracle 数据库 SQL 语言参考*》了解 SQL 约束子句

对象表

Oracle 对象类型是具有名称、 属性、和方法的用户定义类型。对象类型使得对现实世界中的实体（如客户和采购单等），作为对象在数据库中进行建模成为可能。

对象类型定义逻辑结构，但不会创建存储。示例 2-5 创建了一个名为 `department_typ` 的对象类型。

示例 2-5 对象类型

```
CREATE TYPE department_typ AS OBJECT
   ( d_name      VARCHAR2(100),
     d_address   VARCHAR2(200) );
/
```

An **object table** is a special kind of table in which each row represents an object. The `CREATE TABLE` statement in Example 2-6 creates an object table named `departments_obj_t` of the object type `department_typ`. The attributes (columns) of this table are derived from the definition of the object type. The `INSERT` statement inserts a row into this table.

Example 2-6 Object Table

```
CREATE TABLE departments_obj_t OF department_typ;
INSERT INTO departments_obj_t
  VALUES ('hr', '10 Main St, Sometown, CA');
```

Like a relational column, an object table can contain rows of just one kind of thing, namely, object instances of the same declared type as the table. By default, every row object in an object table has an associated logical **object identifier (OID)** that uniquely identifies it in an object table. The OID column of an object table is a hidden column.

See Also:

- *Oracle Database Object-Relational Developer's Guide* to learn about object-relational features in Oracle Database
- *Oracle Database SQL Language Reference* for `CREATE TYPE` syntax and semantics

Temporary Tables

Oracle Database **temporary tables** hold data that exists only for the duration of a transaction or session. Data in a temporary table is private to the session, which means that each session can only see and modify its own data.

Temporary tables are useful in applications where a result set must be buffered. For example, a scheduling application enables college students to create optional semester course schedules. Each schedule is represented by a row in a temporary table. During the session, the schedule data is private. When the student decides on a schedule, the application moves the row for the chosen schedule to a permanent table. At the end of the session, the schedule data in the temporary data is

```
CREATE TYPE department_typ AS OBJECT
   ( d_name      VARCHAR2(100),
     d_address   VARCHAR2(200) );
/
```

对象表是一种特殊的表，其中每一行表示一个对象。在示例 2-6 中，CREATE TABLE 语句创建一个名为 departments_obj_t 的对象表，其对象类型为 department_typ 。此表的属性 （列） 派生自该对象类型的定义。使用 INSERT 语句将行插入到此表。

示例 2-6 对象表

```
CREATE TABLE departments_obj_t OF department_typ;
INSERT INTO departments_obj_t
  VALUES ('hr', '10 Main St, Sometown, CA');
```

与关系表列相似，对象表只能包含同种类型的行，即具有同种声明类型的对象实例。默认情况下，对象表中的每个行对象都有一个相关联的逻辑对象标识符 (OID)，以便在该对象表中唯一地标识该对象。对象表的 OID 列是一个隐藏列。

另见：

- 《*Oracle 数据库对象关系开发指南*》了解 Oracle 数据库中的对象-关系特性
- 《*Oracle 数据库 SQL 语言参考*》了解 `CREATE TYPE` 的语法和语义

临时表

Oracle 数据库的**临时表**，用于存放只存在于某个事务或会话期间的数据。临时表中的数据是会话私有的，这意味着每个会话只可以查看和修改自己的数据。

临时表对于必须缓冲中间结果集的应用程序非常有用。例如，一个计划应用程序使学生可以创建可选的学期课程计划。每个课程计划由临时表中的一行表示。在会话期间，课程计划数据是私有的。当某个学生确定了课程计划，应用程序会将其所选计划移入永久表。在会话结束时，临时表中的课程计划数据将被自动删除。

automatically dropped.

Temporary Table Creation

The `CREATE GLOBAL TEMPORARY TABLE` statement creates a temporary table. The `ON COMMIT` clause specifies whether the table data is transaction-specific (default) or session-specific.

Unlike temporary tables in some other relational databases, when you create a temporary table in an Oracle database, you create a static table definition. The temporary table is a persistent object described in the data dictionary, but appears empty until your session inserts data into the table. You create a temporary table for the database itself, not for every PL/SQL stored procedure.

Because temporary tables are statically defined, you can create indexes for them with the `CREATE INDEX` statement. Indexes created on temporary tables are also temporary. The data in the index has the same session or transaction scope as the data in the temporary table. You can also create a view or trigger on a temporary table.

See Also:

- *Oracle Database Administrator's Guide* to learn how create and manage temporary tables
- *Oracle Database SQL Language Reference* for `CREATE GLOBAL TEMPORARY TABLE` syntax and semantics
- "Overview of Views" and "Overview of Triggers"

Segment Allocation in Temporary Tables

Like permanent tables, temporary tables are defined in the data dictionary. However, temporary tables and their indexes do not automatically allocate a segment when created. Instead, temporary segments are allocated when data is first inserted. Until data is loaded in a session the table appears empty. Temporary segments are deallocated at the end of the transaction for transaction-specific temporary tables and at the end of the session for session-specific temporary tables.

See Also:

"Temporary Segments"

External Tables

An **external table** accesses data in external sources as if this data were in a table in the database. You can use SQL, PL/SQL, and Java to query

创建临时表

使用 `CREATE GLOBAL TEMPORARY TABLE` 语句创建一个临时表。`ON COMMIT` 子句指定表中的数据是特定于事务（默认值），还是特定于会话。

与其它一些关系数据库中的临时表不同，当你在 Oracle 数据库中创建一个临时表时，你只创建其静态表定义。临时表是在数据字典中所描述的一个持久对象，但在您的会话向表中插入数据之前，表显示为空。你是在为数据库本身创建一个临时表，而不是为每个 PL/SQL 存储过程。

因为临时表是静态定义的，您可以使用 CREATE INDEX 语句为其创建索引。临时表上创建的索引也是临时的。在索引中的数据与临时表中的数据具有相同的会话或事务范围。您还可以在临时表上创建一个视图或触发器。

另见：

- *《Oracle 数据库管理员指南》*了解如何创建和管理临时表
- *《Oracle 数据库 SQL 语言参考指南》*关于 `CREATE GLOBAL TEMPORARY TABLE` 的语法和语义
- "视图概述"和"触发器概述"

临时表中的段分配

与永久表类似，临时表被定义在数据字典中。但是，临时表和他们的索引不会在创建时自动分配段。相反，临时段是在第一次插入数据时分配的。在一个会话中加载数据之前，表显示为空。对特定事务的临时表，临时段在事务结束时释放空间，而对特定于会话的临时表，在会话结束时释放空间。

另见：

"临时段"

外部表

外部表访问外部数据源中的数据，如同此数据是在数据库中的表中一样。您可以使用 SQL 、 PL/SQL、和 Java 查询外部数据。

the external data.

External tables are useful for querying flat files. For example, a SQL-based application may need to access records in a text file. The records are in the following form:

```
100,Steven,King,SKING,515.123.4567,17-JUN-03,AD_PRES,31944,150,90
101,Neena,Kochhar,NKOCHHAR,515.123.4568,21-SEP-05,AD_VP,17000,100,90
102,Lex,De Haan,LDEHAAN,515.123.4569,13-JAN-01,AD_VP,17000,100,90
```

You could create an external table, copy the file to the location specified in the external table definition, and use SQL to query the records in the text file.

External tables are also valuable for performing ETL tasks common in data warehouse environments. For example, external tables enable the pipelining of the data loading phase with the transformation phase, eliminating the need to stage data inside the database in preparation for further processing inside the database. See "Overview of Data Warehousing and Business Intelligence".

External Table Creation

Internally, creating an external table means creating metadata in the data dictionary. Unlike an ordinary table, an external table does not describe data stored in the database, nor does it describe how data is stored externally. Rather, external table metadata describes how the external table layer must *present* data to the database.

A `CREATE TABLE ... ORGANIZATION EXTERNAL` statement has two parts. The **external table definition** describes the column types. This definition is like a view that enables SQL to query external data without loading it into the database. The second part of the statement maps the external data to the columns.

External tables are read-only unless created with `CREATE TABLE AS SELECT` with the `ORACLE_DATAPUMP` access driver. Restrictions for external tables include no support for indexed columns, virtual columns, and column objects.

External Table Access Drivers

An **access driver** is an API that interprets the external data for the database. The access driver runs inside the database, which uses the driver to read the data in the external table. The access driver and the external table layer are responsible for performing the transformations required on the data in the data file so that it matches the external table definition. Figure 2-4 represents how external data is accessed.

外部表可用于查询平面文件。例如，一个基于 SQL 的应用程序，可能需要访问一个文本文件中的记录。记录的形式如下：

```
100,Steven,King,SKING,515.123.4567,17-JUN-03,AD_PRES,31944,150,90
101,Neena,Kochhar,NKOCHHAR,515.123.4568,21-SEP-05,AD_VP,17000,100,90
102,Lex,De Haan,LDEHAAN,515.123.4569,13-JAN-01,AD_VP,17000,100,90
```

您可以创建一个外部表，把这个文件复制到外部表定义中指定的位置，并使用 SQL 查询该文本文件中的记录。页: 24

外部表对于在数据仓库环境中执行常见的 ETL 任务也是很有价值的。例如，外部表使得数据加载阶段和数据转换阶段对接，消除了为进一步处理数据库中的数据而在数据库内存放中间数据的需要。请参阅《数据仓库和商业智能概述》。

创建外部表

在数据库内部，创建一个外部表意味着在数据字典中创建元数据。与普通的表不同，外部表不描述存储在数据库中的数据，也不会描述数据在外部是如何存储的。外部表的元数据描述了外部表层必须如何提供数据给数据库。

`CREATE TABLE ... ORGANIZATION EXTERNAL` 语句包含两部分。**外部表定义**描述列类型。这个定义像一个视图，使您可以使用 SQL 查询外部数据，而不用将其加载到数据库。该语句的第二部分将外部数据映射到列。

外部表是只读的，除非它是使用 `CREATE TABLE AS SELECT` 和 `ORACLE_DATAPUMP` 访问驱动程序创建的。外部表有些限制，包括不支持索引列、 虚拟列、和列对象。

外部表访问驱动程序

访问驱动程序是一个 API，它为数据库解释外部数据。访问驱动程序在数据库内运行，数据库使用该驱动程序来读取外部表中的数据。访问驱动程序和外部表层负责对数据文件中的数据进行转换，使它与外部表定义匹配。图 2-4 表示外部数据是如何被访问的。

Figure 2-4 External Tables

Description of "Figure 2-4 External Tables"

Oracle provides the ORACLE_LOADER (default) and ORACLE_DATAPUMP access drivers for external tables. For both drivers, the external files are not Oracle data files.

ORACLE_LOADER enables read-only access to external files using SQL*Loader. You cannot create, update, or append to an external file using the ORACLE_LOADER driver.

The ORACLE_DATAPUMP driver enables you to **unload** external data. This operation involves reading data from the database and inserting the data into an external table, represented by one or more external files. After external files are created, the database cannot update or append data to them. The driver also enables you to **load** external data, which involves reading an external table and loading its data into a database.

See Also:

- *Oracle Database Administrator's Guide* to learn about managing external tables, external connections, and directory objects
- *Oracle Database Utilities* to learn about external tables
- *Oracle Database SQL Language Reference* for information about creating and querying external tables

Table Storage

Oracle Database uses a data segment in a tablespace to hold table data. As explained in "User Segments", a segment contains extents made up of data blocks.

图 2-4 外部表

Description of "Figure 2-4 External Tables"

Oracle 为外部表提供了 ORACLE_LOADER（缺省） 和 ORACLE_DATAPUMP 访问驱动程序。对于这两个驱动程序来说，外部文件不是 Oracle 数据文件（，而只是普通操作系统文件）。

ORACLE_LOADER 允许通过 SQL*Loader 对外部文件进行只读访问。您不能使用 ORACLE_LOADER 驱动程序创建、 更新、或追加数据到外部文件。

ORACLE_DATAPUMP 驱动程序使您能够**卸载**外部数据。此操作包括从数据库读取数据，并将其插入到由一个或多个外部文件所代表的外部表中。创建外部文件后，无法更新或将追加数据到外部文件。该驱动程序也使您能够**加载**外部数据，包括读取外部表并将其数据加载到数据库中。

另见：

- 《*Oracle 数据库管理员指南*》了解管理外部表、 外部连接、和目录对象等
- 《*Oracle 数据库实用程序*》了解外部表
- 《*Oracle 数据库 SQL 语言参考*》关于创建和查询外部表的信息

表存储

Oracle 数据库使用表空间中的数据段保存表数据。如"用户段"所述，段包含由数据块组成的扩展盘区。

The data segment for a table (or cluster data segment, when dealing with a table cluster) is located in either the default tablespace of the table owner or in a tablespace named in the `CREATE TABLE` statement.

Table Organization

By default, a table is organized as a **heap**, which means that the database places rows where they fit best rather than in a user-specified order. Thus, a heap-organized table is an unordered collection of rows. As users add rows, the database places the rows in the first available free space in the data segment. Rows are not guaranteed to be retrieved in the order in which they were inserted.

Note:

Index-organized tables use a different principle of organization. See "Overview of Index-Organized Tables".

The `hr.departments` table is a heap-organized table. It has columns for department ID, name, manager ID, and location ID. As rows are inserted, the database stores them wherever they fit. A data block in the table segment might contain the unordered rows shown in Example 2-7.

Example 2-7 Rows in Departments Table

```
50,Shipping,121,1500
120,Treasury,,1700
70,Public Relations,204,2700
30,Purchasing,114,1700
130,Corporate Tax,,1700
10,Administration,200,1700
110,Accounting,205,1700
```

The column order is the same for all rows in a table. The database usually stores columns in the order in which they were listed in the `CREATE TABLE` statement, but this order is not guaranteed. For example, if a table has a column of type `LONG`, then Oracle Database always stores this column last in the row. Also, if you add a new column to a table, then the new column becomes the last column stored.

A table can contain a **virtual column**, which unlike normal columns does not consume space on disk. The database derives the values in a virtual column on demand by computing a set of user-specified expressions or functions. You can index virtual columns, collect statistics on them, and create integrity constraints. Thus, virtual columns are much like nonvirtual columns.

See Also:

表数据段（或涉及表簇时的簇数据段）位于表所有者的默认空间中，或 CREATE TABLE 语句中所指定的表空间中。

表组织

默认情况下，表按**堆**的形式来组织数据，这意味着数据库将行存放在最适合他们的位置，而不是按用户指定的顺序来存放。因此，堆组织表是一个无序的行的集合。当用户往其中添加行时，数据库将行置于在数据段中第一个可用的空闲空间。不能保证行按照插入它们的顺序进行检索。

注意：

索引组织表使用不同的组织原则。请参见"索引组织表概述"。

hr.departments 表是一个堆组织表。它具有部门 ID、 名称、 经理 ID、位置 id 等列。当插入行时，数据库将其存储到适合的位置。表段中的某个数据块可能包含示例 2-7 所示的无序的行。

Example 2-7 Rows in Departments Table

```
50,Shipping,121,1500
120,Treasury,,1700
70,Public Relations,204,2700
30,Purchasing,114,1700
130,Corporate Tax,,1700
10,Administration,200,1700
110,Accounting,205,1700
```

列的顺序对于表中的所有行都是相同的。通常，数据库按它们被列在 CREATE TABLE 语句中的列顺序来存储列，但这个顺序是不保证的。例如，如果表中有 LONG 类型的列，那么数据库总是将其存储到行中的最后一列。此外，如果向表中添加一个新列，则新列将被存储为最后一列。

与常规列不同，表可以包含**虚拟列**，它不占用磁盘上的空间。数据库通过计算一组用户指定的表达式或函数，在虚拟列中按需派生一个值。您可以对虚拟列创建索引、 收集统计数据、并创建完整性约束。因此，虚拟列非常像非虚拟列。

另见：

Oracle Database SQL Language Reference to learn about virtual columns

Row Storage

The database stores rows in data blocks. Each row of a table containing data for less than 256 columns is contained in one or more **row pieces**.

If possible, Oracle Database stores each row as one row piece. However, if all of the row data cannot be inserted into a single data block, or if an update to an existing row causes the row to outgrow its data block, then the database stores the row using multiple row pieces (see "Data Block Format").

Rows in a table cluster contain the same information as rows in nonclustered tables. Additionally, rows in a table cluster contain information that references the cluster key to which they belong.

Rowids of Row Pieces

A rowid is effectively a 10-byte physical address of a row. As explained in "Rowid Data Types", every row in a heap-organized table has a rowid unique to this table that corresponds to the physical address of a row piece. For table clusters, rows in different tables that are in the same data block can have the same rowid.

Oracle Database uses rowids internally for the construction of indexes. For example, each key in a B-tree index is associated with a rowid that points to the address of the associated row for fast access (see "B-Tree Indexes"). Physical rowids provide the fastest possible access to a table row, enabling the database to retrieve a row in as little as a single I/O.

See Also:

"Rowid Format"

Table Compression

The database can use table compression to eliminate duplicate values in a data block. For tables with highly redundant data, compression saves disk space, reduces memory use in the database buffer cache, and in some cases speeds query execution. Table compression is transparent to database applications.

Basic and OLTP Table Compression

Dictionary-based table compression provides good compression ratios. Oracle Database supports the following types of table compression:

- Basic table compression

  This type of compression compresses data inserted by direct path load only and supports limited data types and SQL operations.

《*Oracle 数据库 SQL 语言参考*》了解虚拟列

行存储

数据库将行存储在数据块中。在表中，少于 256 个列的行，被存储在一个或多个**行片**中。

数据库尽可能将每一行作为一个行片来存储。但是，如果所有的行数据不能插入到一个单一数据块中，或者对现有的行的更新导致行溢出，则数据库使用多个行片来存储此行（参见"数据块格式"）。

表簇中的行除了包含非聚集表中的行相同的信息，还包含对其所属簇键的引用信息。

行片Rowids

rowid 实际上是行的一个 10 字节的物理地址。如"Rowid 数据类型"中所述，堆组织表中的每一行都有一个在该表中唯一的 rowid，与一个行片的物理地址相对应。对于表簇，处于同一个数据块的不同表中的行可以有相同的 rowid。

Oracle 数据库在内部使用 rowids 来构建索引。例如，在 B 树索引中的每个键与一个指向关联行地址的 rowid 相关联，以便快速访问（请参阅"B 树索引"）。物理 rowids 提供对表行尽可能最快的访问，使数据库几乎只需单次 I/O 就可以检索到行。

另见：

"Rowid 格式"

表压缩

数据库可以使用表压缩来消除数据块中的重复值。对于数据高度冗余的表，压缩可以节省磁盘空间，减少数据库高速缓存中的内存使用，并在某些情况下可以加快查询执行速度。表压缩对数据库应用程序是透明的。

基本/OLTP表压缩

基于字典的表压缩提供了很好的压缩率。Oracle 数据库支持以下类型的表压缩：

- 基本表压缩

  这种类型的压缩只能压缩由直接路径加载插入的数据，只支持有限的数据类型和 SQL 操作。

- OLTP table compression

  This type of compression is intended for OLTP applications and compresses data manipulated by any SQL operation.

The database stores compressed rows in **row-major format**. All columns of one row are stored together, followed by all columns of the next row, and so on (see Figure 12-7). Duplicate values are replaced with a short reference to a symbol table stored at the beginning of the block. Thus, information needed to re-create the uncompressed data is stored in the data block itself.

Compressed data blocks look much like normal data blocks. Most database features and functions that work on regular data blocks also work on compressed blocks.

See Also:

"SQL*Loader" to learn about using SQL*Loader for direct path loads

Compression Declaration

You can declare compression at the tablespace, table, partition, or subpartition level. If specified at the tablespace level, then all tables created in this tablespace are compressed by default.

The following statement applies OLTP compression to the `orders` table:

```
ALTER TABLE oe.orders COMPRESS FOR OLTP;
```

The following example of a partial `CREATE TABLE` statement specifies OLTP compression for one partition and basic compression for the other partition:

```
CREATE TABLE sales (
    prod_id     NUMBER     NOT NULL,
    cust_id     NUMBER     NOT NULL, ... )
 PCTFREE 5 NOLOGGING NOCOMPRESS
 PARTITION BY RANGE (time_id)
 ( partition sales_2008 VALUES LESS
THAN(TO_DATE(...)) COMPRESS BASIC,
   partition sales_2009 VALUES LESS THAN
(MAXVALUE) COMPRESS FOR OLTP );
```

See Also:

- "Data Block Compression" to learn about the format of compressed data blocks
- *Oracle Database Administrator's Guide* and *Oracle Database Performance Tuning Guide* to learn about table compression

- OLTP 表压缩

  这种类型的压缩用于 OLTP 应用程序，并可压缩任何 SQL 操作的数据。

数据库以**行主要格式**存储压缩行。一个行的所有列都存储在一起，接下来是下一行的所有列，依次类推。重复值被替换为一个存储在块开始处的符号表短引用。因此，重新创建未压缩数据所需的信息存储在数据块本身中。

压缩的数据块看起来跟正常的数据块差不多。在常规数据块上能正常工作的大多数数据库功能，也可以在压缩的数据块上正常工作。

另见：

"SQL*Loader "了解如何使用 SQL*Loader 进行直接路径加载

压缩声明

您可以在表空间、 表、 分区或子分区等级别声明压缩。如果在表空间级指定了压缩，则在该表空间中创建的所有表在缺省情况下都是压缩的。

下面的语句将 OLTP 压缩应用于 `orders` 表：

```
ALTER TABLE oe.orders COMPRESS FOR OLTP;
```

如下所示的 CREATE TABLE 语句示例片断，指定其中一个分区为 OLTP 压缩，而其他分区为基本压缩：

```
CREATE TABLE sales (
    prod_id     NUMBER     NOT NULL,
    cust_id     NUMBER     NOT NULL, ... )
 PCTFREE 5 NOLOGGING NOCOMPRESS
 PARTITION BY RANGE (time_id)
 ( partition sales_2008 VALUES LESS
THAN(TO_DATE(...)) COMPRESS BASIC,
   partition sales_2009 VALUES LESS THAN
(MAXVALUE) COMPRESS FOR OLTP );
```

另见：

- "数据块压缩"了解压缩数据块的格式

- 《*Oracle 数据库管理员指南*》和《*Oracle 数据库性能调整指南*》了

解表压缩

Storage of Null Values

A **null** is the absence of a value in a column. Nulls indicate missing, unknown, or inapplicable data.

Nulls are stored in the database if they fall between columns with data values. In these cases, they require 1 byte to store the length of the column (zero). Trailing nulls in a row require no storage because a new row header signals that the remaining columns in the previous row are null. For example, if the last three columns of a table are null, then no data is stored for these columns.

See Also:

Oracle Database SQL Language Reference to learn more about null values

空值存储

空值是指在某个列中值的缺失。空值表示缺少、 未知、或不适用的数据。

在数据库中，如果在空值的前后都有数据值，则空值会被存储。在这些情况下，他们需要 1 个字节来存储列的长度（零）。尾随的空值在行中不需要存储，因为新的行头标志着前一行中的剩余列都为空。例如，如果表的最后三列为空，则不为这些列存储数据。

另见：

《*Oracle 数据库 SQL 语言参考*》了解空值的更多信息

Overview of Table Clusters

A table cluster is a group of tables that share common columns and store related data in the same blocks. When tables are clustered, a single data block can contain rows from multiple tables. For example, a block can store rows from both the `employees` and `departments` tables rather than from only a single table.

The **cluster key** is the column or columns that the clustered tables have in common. For example, the `employees` and `departments` tables share the `department_id` column. You specify the cluster key when creating the table cluster and when creating every table added to the table cluster.

The **cluster key value** is the value of the cluster key columns for a particular set of rows. All data that contains the same cluster key value, such as `department_id=20`, is physically stored together. Each cluster key value is stored only once in the cluster and the cluster index, no matter how many rows of different tables contain the value.

For an analogy, suppose an HR manager has two book cases: one with boxes of employees folders and the other with boxes of departments folders. Users often ask for the folders for all employees in a particular department. To make retrieval easier, the manager rearranges all the boxes in a single book case. She divides the boxes by department ID. Thus, all folders for employees in department 20 and the folder for department 20 itself are in one box; the folders for employees in department 100 and the folder for department 100 are in a different box, and so on.

表簇概述

表簇是一组表，它们共享公共的列，并将相关的数据存储在相同的数据块中。当表被聚簇时，单个数据块可以包含多个表中的行。例如，一个块可以同时存储来自 `employees` 表和 `departments` 表的行，而不只是单个表中的行。

簇键是所有被聚簇的表的共有列或列集。例如，`employees` 表和 `departments` 表共享 department_id 列。您在创建表簇时，和创建被添加到表簇的每个表时，指定簇键。

簇键值是一组特定行的簇键列的值。包含相同簇键值的所有数据（例如 `department_id=20`），物理上存储在一起。每个簇键值在簇或簇索引中只存储一次，而无论在这些不同表中有多少行包含这个值。

打个比方，假设一个人力资源经理有两个书柜： 一个书柜装有一盒盒雇员文件夹，而另一个书柜装有一盒盒部门文件夹。用户经常要找一个特定部门的所有员工的文件夹。为便于检索，经理重新排列了书柜中的所有盒子。她将这些盒子按部门 id 划分。这样，部门 20 的所有雇员文件夹和该部门 20 的文件夹本身放在一个盒子中；部门 100 的所有雇员文件夹和该部门 100 的文件夹本身放在另一个盒子中，依次类推。

You can consider clustering tables when they are primarily queried (but not modified) and records from the tables are frequently queried together or joined. Because table clusters store related rows of different tables in the same data blocks, properly used table clusters offer the following benefits over nonclustered tables:

- Disk I/O is reduced for joins of clustered tables.
- Access time improves for joins of clustered tables.
- Less storage is required to store related table and index data because the cluster key value is not stored repeatedly for each row.

Typically, clustering tables is not appropriate in the following situations:

- The tables are frequently updated.
- The tables frequently require a full table scan.
- The tables require truncating.

See Also:

Oracle Database Performance Tuning Guide for guidelines on when to use table clusters

Overview of Indexed Clusters

An **indexed cluster** is a table cluster that uses an index to locate data. The **cluster index** is a B-tree index on the cluster key. A cluster index must be created before any rows can be inserted into clustered tables. Assume that you create the cluster `employees_departments_cluster` with the cluster key `department_id`, as shown in Example 2-8. Because the `HASHKEYS` clause is not specified, this cluster is an indexed cluster. Afterward, you create an index named `idx_emp_dept_cluster` on this cluster key.

Example 2-8 Indexed Cluster

```
CREATE CLUSTER employees_departments_cluster
   (department_id NUMBER(4))
SIZE 512;
CREATE INDEX idx_emp_dept_cluster ON CLUSTER
employees_departments_cluster;
```

You then create the `employees` and `departments` tables in the cluster, specifying the `department_id` column as the cluster key, as follows (the ellipses mark the place where the column specification goes):

如果多个表主要是被查询（而不是修改），且各表中的记录是经常被一起查询或联接，在这些情况下可以考虑将他们聚簇化。因为表簇将不同表中的相关行存储在同一个数据块中，被正确使用的表簇相比非聚簇表具有下列优点：

- 对于被聚簇表的联接，可以减少磁盘 I/O。
- 对于被聚簇表的联接，可以提高访问速度。
- 只需更少的空间来存储相关的表和索引数据，因为簇键值不会为每行重复存储。

通常，簇表不适合以下情况：

- 会经常被更新的表。
- 经常需要全表扫描的表。
- 需要被截断的表。

另见：

《*Oracle 数据库性能优化指南*》了解何时使用表群集的指导方针

索引化表簇概述

索引化聚簇是使用索引来查找数据的表簇。而**簇索引**是一个簇键上的 B 树索引。簇索引必须先被创建，然后才能将行插入到簇表中。

假定使用簇键 department_id 来创建聚簇 employees_departments_cluster，如例 2-8 所示。因为未指定 HASHKEYS 子句，这个簇是一个索引化表簇。然后，在这个簇键上创建一个名为 idx_emp_dept_cluster 的索引

例 2-8 索引化聚簇

```
CREATE CLUSTER employees_departments_cluster
   (department_id NUMBER(4))
SIZE 512;
CREATE INDEX idx_emp_dept_cluster ON CLUSTER
employees_departments_cluster;
```

然后在该簇中创建雇员表和部门表，并指定 `department_id` 列为簇键，如下所示（省略号表示放置列定义的地方）

```
CREATE TABLE employees ( ... )
   CLUSTER employees_departments_cluster
(department_id);

CREATE TABLE departments ( ... )
   CLUSTER employees_departments_cluster
(department_id);
```

Finally, you add rows to the `employees` and `departments` tables. The database physically stores all rows for each department from the `employees` and `departments` tables in the same data blocks. The database stores the rows in a heap and locates them with the index. Figure 2-5 shows the `employees_departments_cluster` table cluster, which contains `employees` and `departments`. The database stores rows for employees in department 20 together, department 110 together, and so on. If the tables are not clustered, then the database does not ensure that the related rows are stored together.

Figure 2-5 Clustered Table Data

```
CREATE TABLE employees ( ... )
   CLUSTER employees_departments_cluster
(department_id);

CREATE TABLE departments ( ... )
   CLUSTER employees_departments_cluster
(department_id);
```

最后，将行添加到 `employees` 表和 `departments` 表中。数据库在物理上将雇员表和部门表中每个部门的所有行都存储在相同的数据块中。数据库以堆的形式存储行，并使用索引定位他们。

图 2-5 显示了包含 `employees` 表和 `departments` 表所构成的 employees_departments_cluster 表簇。数据库将部门 20 的雇员所在的行存储在一起，部门 110 的行也存储在一起，依次类推。如果表未被聚簇，那么数据库将不保证相关的行被存储在一起。

图 2－5 簇表数据

Description of "Figure 2-5 Clustered Table Data"

The B-tree cluster index associates the cluster key value with the **database block address (DBA)** of the block containing the data. For example, the index entry for key 20 shows the address of the block that contains data for employees in department 20:

```
20,AADAAAA9d
```

The cluster index is separately managed, just like an index on a nonclustered table, and can exist in a separate tablespace from the table cluster.

See Also:

- "Overview of Indexes"
- *Oracle Database Administrator's Guide* to learn how to create and manage indexed clusters

Description of "Figure 2-5 Clustered Table Data"

B 树簇索引把簇键值与数据所在块的数据库块地址 (DBA) 关联起来。例如，键 20 的索引条目显示包含部门 20 的雇员数据所在块的地址：

```
20,AADAAAA9d
```

与非聚簇表上的索引类似，簇索引被单独管理，簇索引也可以与表簇存在于不同的表空间中。

另见：

- "索引概述"
- 《*Oracle 数据库管理员指南*》了解如何创建和管理索引化聚簇

- *Oracle Database SQL Language Reference* for `CREATE CLUSTER` syntax and semantics

Overview of Hash Clusters

A **hash cluster** is like an indexed cluster, except the index key is replaced with a hash function. No separate cluster index exists. In a hash cluster, the data is the index.

With an indexed table or indexed cluster, Oracle Database locates table rows using key values stored in a separate index. To find or store a row in an indexed table or table cluster, the database must perform at least two I/Os:

- One or more I/Os to find or store the key value in the index
- Another I/O to read or write the row in the table or table cluster

To find or store a row in a hash cluster, Oracle Database applies the hash function to the cluster key value of the row. The resulting hash value corresponds to a data block in the cluster, which the database reads or writes on behalf of the issued statement.

Hashing is an optional way of storing table data to improve the performance of data retrieval. Hash clusters may be beneficial when the following conditions are met:

- A table is queried much more often than modified.
- The hash key column is queried frequently with equality conditions, for example, `WHERE department_id=20`. For such queries, the cluster key value is hashed. The hash key value points directly to the disk area that stores the rows.
- You can reasonably guess the number of hash keys and the size of the data stored with each key value.

Hash Cluster Creation

The **cluster key**, like the key of an indexed cluster, is a single column or composite key shared by the tables in the cluster. The **hash key values** are actual or possible values inserted into the cluster key column. For example, if the cluster key is `department_id`, then hash key values could be 10, 20, 30, and so on.

Oracle Database uses a hash function that accepts an infinite number of hash key values as input and sorts them into a finite number of buckets.

- 《*Oracle 数据库 SQL 语言参考*》了解 `CREATE CLUSTER` 的语法和语义

哈希簇概述

除了索引键被替换为一个哈希函数之外，**哈希簇**就像一个索引化聚簇。它没有单独的簇索引存在。对一个哈希簇来说，数据本身就是索引。

对已索引的表或索引化聚簇，数据库用存储在一个单独的索引中的键值查找表行。要查找或存储已索引的表或索引化聚簇中的一个行，数据库必须执行至少两个 I/O 操作：

- 为查找或存储在索引中的键值，需要一个或多个 I/O
- 为读取或写入表或表簇中的行，还需要一个 I/O

为查找或存储在哈希簇中的一个行，数据库将哈希函数应用到行的簇键值。得出的哈希值对应到一个聚簇中的数据块，数据库则按发出的语句读写该块。

哈希是一种可选的表数据存储方法，用来提高数据检索的性能。在满足以下条件时，哈希簇可能是有益的：

- 经常被查询，但不经常被修改的表。
- 哈希键列经常使用等值条件查询，例如， WHERE department_id = 20。对于这样的查询，簇键值是已经过哈希运算的。哈希键值直接指向存储行的磁盘区域。
- 您可以合理地猜出哈希键的数目，和每个键值所存储的数据的大小。

创建哈希簇

与索引化聚簇的键类似，簇键是由簇中各表共享的单键列或复合键列。哈希键值是插入到簇键列的实际值或可能值。例如，如果簇键是 department_id，那么哈希键键值可能是 10、 20、 30，等等。

Oracle 数据库使用一个哈希函数，接受任意多个哈希键值作为输入，经过排序并哈希到有限数量的桶。每个桶都有一个唯一的数字 ID，称为**哈希值**。

Each bucket has a unique numeric ID known as a **hash value**. Each hash value maps to the database block address for the block that stores the rows corresponding to the hash key value (department 10, 20, 30, and so on).

To create a hash cluster, you use the same `CREATE CLUSTER` statement as for an indexed cluster, with the addition of a hash key. The number of hash values for the cluster depends on the hash key. In Example 2-9, the number of departments that are likely to exist is 100, so `HASHKEYS` is set to `100`.

Example 2-9 Hash Cluster

```
CREATE CLUSTER employees_departments_cluster
   (department_id NUMBER(4))
SIZE 8192 HASHKEYS 100;
```

After you create `employees_departments_cluster`, you can create the `employees` and `departments` tables in the cluster. You can then load data into the hash cluster just as in the indexed cluster described in Example 2-8.

See Also:

Oracle Database Administrator's Guide to learn how to create and manage hash clusters

Hash Cluster Queries

The database, not the user, determines how to hash the key values input by the user. For example, assume that users frequently execute queries such as the following, entering different department ID numbers for `p_id`:

```
SELECT *
FROM   employees
WHERE  department_id = :p_id;

SELECT *
FROM   departments
WHERE  department_id = :p_id;
SELECT *
FROM   employees e, departments d
WHERE  e.department_id = d.department_id
AND    d.department_id = :p_id;
```

If a user queries employees in `department_id=20`, then the database might hash this value to bucket 77. If a user queries employees in

每个哈希值都映射到哈希键值（部门 10、 20、 30，等等)对应行所在块的数据库块地址。

与创建索引化聚簇类似，使用 `CREATE CLUSTER` 语句创建一个哈希簇，但得加上一个哈希键。哈希值的数量取决于哈希键。示例 2-9 中可能存在的部门的数量是 100，所以 HASHKEYS 设置为 100。

示例 2-9 哈希簇

```
CREATE CLUSTER employees_departments_cluster
   (department_id NUMBER(4))
SIZE 8192 HASHKEYS 100;
```

创建聚簇 employees_departments_cluster 后，可以在该簇中创建 `employees` 表和 `departments` 表。然后可以将数据装载到哈希簇中，如同示例 2-8 中所述的索引化簇一样。

另见：

《*Oracle 数据库管理员指南*》了解如何创建和管理哈希簇

查询哈希簇

由数据库（而不是用户）确定如何哈希用户输入的键值。例如，假设用户经常执行如下的查询， 为 p_id 输入不同的部门 ID 号：

```
SELECT *
FROM   employees
WHERE  department_id = :p_id;

SELECT *
FROM   departments
WHERE  department_id = :p_id;
SELECT *
FROM   employees e, departments d
WHERE  e.department_id = d.department_id
AND    d.department_id = :p_id;
```

如果某个用户查询 department_id = 20 的雇员，那么数据库可能将此值哈希到桶 77。如果某个用户查询 department_id = 10 的雇员，那么数据库可能将

department_id=10, then the database might hash this value to bucket 15. The database uses the internally generated hash value to locate the block that contains the employee rows for the requested department.

Figure 2-6 depicts a hash cluster segment as a horizontal row of blocks. As shown in the graphic, a query can retrieve data in a single I/O.

Figure 2-6 Retrieving Data from a Hash Cluster

Description of "Figure 2-6 Retrieving Data from a Hash Cluster"

A limitation of hash clusters is the unavailability of range scans on nonindexed cluster keys (see "Index Range Scan"). Assume that no separate index exists for the hash cluster created in Example 2-9. A query for departments with IDs between 20 and 100 cannot use the hashing algorithm because it cannot hash every possible value between 20 and 100. Because no index exists, the database must perform a full scan.

Hash Cluster Variations

A **single-table hash cluster** is an optimized version of a hash cluster that supports only one table at a time. A one-to-one mapping exists between hash keys and rows. A single-table hash cluster can be beneficial when users require rapid access to a table by primary key. For example, users often look up an employee record in the employees table by employee_id.

A **sorted hash cluster** stores the rows corresponding to each value of the hash function in such a way that the database can efficiently return them in sorted order. The database performs the optimized sort internally. For applications that always consume data in sorted order, this technique can mean faster retrieval of data. For example, an application might always sort on the order_date column of the orders table.

此值哈希到桶 15。数据库使用内部生成的哈希值来定位包含请求的部门的雇员行所在的数据块。

图 2-6 将哈希簇段显示为一行水平排列的块。如图所示，查询只需单个 I/O 就可以检索到数据。

Figure 2-6 Retrieving Data from a Hash Cluster

Description of "Figure 2-6 Retrieving Data from a Hash Cluster"

哈希簇的一个限制是在非索引化簇键上的范围扫描不可用（请参见"索引范围扫描"）。假设在示例 2-9 中没有为哈希簇创建单独的索引。对部门 id 在 20 和 100 之间的查询不能使用哈希算法，因为它不能哈希 20 和 100 之间的每个可能的值。因为没有索引存在，数据库必须执行完全扫描。

哈希簇变体

单表哈希簇是哈希簇的一个优化版本，一次只支持一个表。哈希键和行之间存在一一映射。当用户需要通过主键快速访问表时，单表哈希簇会很有用。例如，用户经常通过 employee_id 查找一个雇员表中的雇员记录。

排序哈希簇存储哈希函数的每个值对应的行，通过某种方式，数据库可以有效地把他们按已排定的顺序返回。数据库在内部执行优化的排序。对于需要总是按排定顺序来消费数据的应用程序，这种技术可能会更快的检索到数据。例如，应用程序可能总是按订单表的 order_date 列进行排序。

See Also:
Oracle Database Administrator's Guide to learn how to create single-table and sorted hash clusters

Hash Cluster Storage

Oracle Database allocates space for a hash cluster differently from an indexed cluster. In Example 2-9, `HASHKEYS` specifies the number of departments likely to exist, whereas `SIZE` specifies the size of the data associated with each department. The database computes a storage space value based on the following formula:

```
HASHKEYS * SIZE / database_block_size
```

Thus, if the block size is 4096 bytes in Example 2-9, then the database allocates at least 200 blocks to the hash cluster.

Oracle Database does not limit the number of hash key values that you can insert into the cluster. For example, even though `HASHKEYS` is `100`, nothing prevents you from inserting 200 unique departments in the `departments` table. However, the efficiency of the hash cluster retrieval diminishes when the number of hash values exceeds the number of hash keys.

To illustrate the retrieval issues, assume that block 100 in Figure 2-6 is completely full with rows for department 20. A user inserts a new department with `department_id` 43 into the `departments` table. The number of departments exceeds the `HASHKEYS` value, so the database hashes `department_id` 43 to hash value 77, which is the same hash value used for `department_id` 20. Hashing multiple input values to the same output value is called a **hash collision**.

When users insert rows into the cluster for department 43, the database cannot store these rows in block 100, which is full. The database links block 100 to a new overflow block, say block 200, and stores the inserted rows in the new block. Both block 100 and 200 are now eligible to store data for either department. As shown in Figure 2-7, a query of either department 20 or 43 now requires *two* I/Os to retrieve the data: block 100 and its associated block 200. You can solve this problem by re-creating the cluster with a different `HASHKEYS` value.

Figure 2-7 Retrieving Data from a Hash Cluster When a Hash Collision Occurs

另见：
《*Oracle 数据库管理员指南*》了解如何创建单表簇和排序哈希簇

哈希簇存储

Oracle 数据库为哈希簇分配空间的方式不同于索引化簇。在示例 2-9 中，HASHKEYS 指定可能存在的部门数，而 `SIZE` 指定与每个部门相关联的数据的大小。则数据库将基于以下公式计算存储空间：

```
HASHKEYS * SIZE / database_block_size
```

因此，如果示例 2-9 中的块大小是 4096 字节，那么数据库至少要为该哈希簇分配 200 个数据块。

Oracle 数据库并不限制您可以向哈希簇中插入的哈希键值的数目。例如，即使 HASHKEYS 是 100，但这不会阻止你向部门表中插入 200 个不同的部门。但是，当（输入的）哈希键值的数目超过哈希键（HASHKEYS 的设定）数目时，哈希簇检索效率会降低。

为说明检索问题，假定图 2-6 中的块 100 内全是部门 20 的行。用户向部门表中插入一个新的部门，其 department_id 为 43。部门的数量超过 HASHKEYS 的值，因此数据库对 department_id 43 进行哈希处理得到哈希值 77，这与 department_id 20 的哈希值相同。多个输入的值被哈希为相同的输出值称为**哈希冲突**。

当用户为部门 43 往簇中插入行时，数据库不能将这些行存储在块 100 中，因为它已经满了。数据库将块 100 链接到一个新的溢出块，比如说块 200，并将插入的行存储在新块中。这两个块 100 和 200 现在都可以存储部门 20 或 43 的数据。如图 2-7，查询部门 20 或 43 现在需要两个 i/o 才能检索到数据： 块 100 和其相关联的块 200。您可以通过重新创建具有不同 HASHKEYS 值的表簇来解决这个问题。

图 2-7 当哈希冲突 发生时，从哈希簇检索数据

Description of "Figure 2-7 Retrieving Data from a Hash Cluster When a Hash Collision Occurs"

See Also:

Oracle Database Administrator's Guide to learn how to manage space in hash clusters

Description of "Figure 2-7 Retrieving Data from a Hash Cluster When a Hash Collision Occurs"

另见：

《*Oracle 数据库管理员指南*》了解如何在哈希簇中管理空间

Part I

Oracle Relational Data Structures

This part describes the basic data structures of an Oracle database, including data integrity rules, and the structures that store metadata.

This part contains the following chapters:

- Chapter 2, "Tables and Table Clusters"
- Chapter 3, "Indexes and Index-Organized Tables"
- Chapter 4, "Partitions, Views, and Other Schema Objects"
- Chapter 5, "Data Integrity"
- Chapter 6, "Data Dictionary and Dynamic Performance Views"

第一部分

Oracle关系数据结构

本部分介绍了 Oracle 数据库的基本数据结构，包括数据完整性规则，和存储元数据的结构。

本部分包含以下章节：

- 第 2 章，表和表簇
- 第 3 章，索引和索引组织表
- 第 4 章，分区、视图、和其它模式对象
- 第 5 章，数据完整性
- 第 6 章，数据字典和动态性能视图

3 Indexes and Index-Organized Tables

This chapter discusses indexes, which are schema objects that can speed access to table rows, and index-organized tables, which are tables stored in an index structure.

This chapter contains the following sections:

- Overview of Indexes
  - Index Characteristics
  - B-Tree Indexes
  - Bitmap Indexes
  - Function-Based Indexes
  - Application Domain Indexes
  - Index Storage
- Overview of Index-Organized Tables
  - Index-Organized Table Characteristics
  - Index-Organized Tables with Row Overflow Area
  - Secondary Indexes on Index-Organized Tables

Overview of Indexes

An **index** is an optional structure, associated with a table or table cluster, that can sometimes speed data access. By creating an index on one or more columns of a table, you gain the ability in some cases to retrieve a small set of randomly distributed rows from the table. Indexes are one of many means of reducing disk I/O.

If a heap-organized table has no indexes, then the database must perform a full table scan to find a value. For example, without an index, a query of location `2700` in the `hr.departments` table requires the database to search every row in every table block for this value. This approach does not scale well as data volumes increase.

For an analogy, suppose an HR manager has a shelf of cardboard boxes. Folders containing employee information are inserted randomly in the boxes. The folder for employee Whalen (ID 200) is 10 folders up from the bottom of box 1, whereas the folder for King (ID 100) is at the bottom of box 3. To locate a folder, the manager looks at every folder in box 1 from

第3章 索引和索引组织表

本章讨论索引，这种模式对象可以加快对常规表、及以索引结构存储的索引组织表的访问。

本章包含以下各节：

- 索引概述
  - 索引特征
  - B-树索引
  - 位图索引
  - 基于函数的索引
  - 应用程序域索引
  - 索引存储
- 索引组织表概述
  - 索引组织表特征
  - 索引组织表的行溢出区
  - 索引组织表的辅助索引

索引概述

索引是一种与表或表簇相关联的可选结构，有时可以提高数据访问速度。通过在表中的一个或多个列上创建索引，在某些情况下使您能够(快速地)从随机分布的表行中检索一小部分行。索引是减少磁盘 I/O 的许多手段之一。

如果一个堆组织表没有索引，数据库必须执行全表扫描来查找值。例如，如果没有索引，为查询 hr.departments 表中的位置 2700，数据库需要搜索每个表块中的每一行，以找到该值。当数据量增加时，这种方法不具备良好的可扩展性。

打个比方，假设一个人力资源经理有一个搁放纸盒的架子。在框中包含雇员信息的文件夹随机放置在纸盒中。员工 Whalen (ID 200) 的文件夹是从 10 盒从底往上的盒子中的第 1 盒，而 King 是(ID 100） 的文件夹是从底往上的第 3 盒。要查找某个文件夹，经理从底往上查看这些盒子中的每个文件夹，看完一盒又看下一盒，直到找到该文件夹。为加快查找速度，经理可创建一个

bottom to top, and then moves from box to box until the folder is found. To speed access, the manager could create an index that sequentially lists every employee ID with its folder location:

```
ID 100: Box 3, position 1 (bottom)
ID 101: Box 7, position 8
ID 200: Box 1, position 10
.
.
.
```

Similarly, the manager could create separate indexes for employee last names, department IDs, and so on.

In general, consider creating an index on a column in any of the following situations:

- The indexed columns are queried frequently and return a small percentage of the total number of rows in the table.
- A referential integrity constraint exists on the indexed column or columns. The index is a means to avoid a full table lock that would otherwise be required if you update the parent table primary key, merge into the parent table, or delete from the parent table.
- A unique key constraint will be placed on the table and you want to manually specify the index and all index options.

See Also:

Chapter 5, "Data Integrity"

Index Characteristics

Indexes are schema objects that are logically and physically independent of the data in the objects with which they are associated. Thus, an index can be dropped or created without physically affecting the table for the index.

Note:

If you drop an index, then applications still work. However, access of previously indexed data can be slower.

The absence or presence of an index does not require a change in the wording of any SQL statement. An index is a fast access path to a single row of data. It affects only the speed of execution. Given a data value that has been indexed, the index points directly to the location of the rows containing that value.

The database automatically maintains and uses indexes after they are created. The database also automatically reflects changes to data, such as

索引，按顺序列出每个雇员 ID 和它的文件夹位置：

```
ID 100: Box 3, position 1 (bottom)
ID 101: Box 7, position 8
ID 200: Box 1, position 10
.
.
.
```

同样，经理可以为雇员姓氏，部门 id 等创建单独的索引。

通常，在下列情况下可以考虑在某列上创建索引：

- 要索引的列经常被查询，并只返回表中的行的总数的一小部分。
- 在索引的列或列集上存在引用完整性约束。索引可以避免当你更新父表主键、合并父表、从父表删除行时可能引起的全表锁定。
- 要在表上设置唯一键约束，并且您想手动指定索引和所有索引选项。

另见:

第 5 章, "数据完整性"

索引特征

索引是一种模式对象，它在逻辑上和物理上都与其相关联的对象中的数据保持独立。因此，可以删除或创建索引而不会实际影响相关的表。

注意：

如果您删除一个索引，应用程序将仍然可以工作。不过，访问之前索引过数据可能变慢。

索引的存在与否，不需要改变任何 SQL 语句的写法。索引是到单一行数据的快速访问路径。它只影响执行的速度。对于一个已被索引的给定的数据值，索引直接指向包含该值的行的位置。

在创建索引后，数据库会自动维护并使用它们。数据库还会自动反映对相关表数据的更改，如添加、 更新、和删除行等，用户不需要对所有相关索引

adding, updating, and deleting rows, in all relevant indexes with no additional actions required by users. Retrieval performance of indexed data remains almost constant, even as rows are inserted. However, the presence of many indexes on a table degrades DML performance because the database must also update the indexes.
Indexes have the following properties:

- Usability

  Indexes are usable (default) or unusable. An **unusable index** is not maintained by DML operations and is ignored by the optimizer. An unusable index can improve the performance of bulk loads. Instead of dropping an index and later re-creating it, you can make the index unusable and then rebuild it. Unusable indexes and index partitions do not consume space. When you make a usable index unusable, the database drops its index segment.
- Visibility

  Indexes are visible (default) or invisible. An **invisible index** is maintained by DML operations and is not used by default by the optimizer. Making an index invisible is an alternative to making it unusable or dropping it. Invisible indexes are especially useful for testing the removal of an index before dropping it or using indexes temporarily without affecting the overall application.

See Also:

- "Overview of the Optimizer"
- *Oracle Database 2 Day DBA* and *Oracle Database Administrator's Guide* to learn how to manage indexes
- *Oracle Database Performance Tuning Guide* to learn how to tune indexes

Keys and Columns

A **key** is a set of columns or expressions on which you can build an index. Although the terms are often used interchangeably, indexes and keys are different. **Indexes** are structures stored in the database that users manage using SQL statements. Keys are strictly a logical concept.
The following statement creates an index on the `customer_id` column of the sample table `oe.orders`:

```
CREATE INDEX ord_customer_ix ON orders
```

做任何操作。即使在插入行时，被索引数据的检索性能仍然几乎不变。但是，在表上存在过多的索引，会降低 DML 性能，因为数据库还必须更新索引。

索引具有以下属性：

- 可用性

  索引可能是可用的（默认值）或不可用的。**不可用索引**在 DML 操作中不会被维护，并会被优化程序忽略。不可用索引可以提高大容量加载的性能。你不用删除索引并稍后重新创建它，你可以使索引不可用，最后再重新生成它。不可用索引或索引分区不会占用空间。当你把一个可用的索引置为不可用时，数据库将删除其索引段。

- 可见性

  索引可能是可见的（默认值）或不可见的。**不可见索引**在 DML 操作中会被维护，但在默认情况下优化程序不会使用它。使索引不可见是使其不可用或删除它的一种替代方法。不可见索引有时特别有用，比如在删除索引前测试移除后果，或临时用一下索引而不会影响整个应用程序。

另见：

- "优化器概述"
- 《*Oracle 数据库 2 日 DBA*》 和 《*Oracle 数据库管理员指南*》了解如何管理索引
- 《*Oracle 数据库性能优化指南*》了解如何优化索引页: 3

键和列

键是一个列集或表达式，您可以在它上面创建索引。虽然这两个术语通常互换使用，但索引和键还是不同的。索引是存储在数据库中的一种结构，用户使用 SQL 语句来管理它们。而键严格来讲只是一个逻辑概念。

下面的语句在示例表 oe.orders 的列 customer_id 上创建索引：

```
CREATE INDEX ord_customer_ix ON orders
```

```
(customer_id);
```

In the preceding statement, the `customer_id` column is the index key. The index itself is named `ord_customer_ix`.

Note:

Primary and unique keys automatically have indexes, but you might want to create an index on a foreign key.

See Also:

Oracle Database SQL Language Reference `CREATE INDEX` syntax and semantics

Composite Indexes

A **composite index**, also called a **concatenated index**, is an index on multiple columns in a table. Columns in a composite index should appear in the order that makes the most sense for the queries that will retrieve data and need not be adjacent in the table.

Composite indexes can speed retrieval of data for `SELECT` statements in which the `WHERE` clause references all or the leading portion of the columns in the composite index. Therefore, the order of the columns used in the definition is important. In general, the most commonly accessed columns go first.

For example, suppose an application frequently queries the `last_name`, `job_id`, and `salary` columns in the `employees` table. Also assume that `last_name` has high cardinality, which means that the number of distinct values is large compared to the number of table rows. You create an index with the following column order:

```
CREATE INDEX employees_ix
   ON employees (last_name, job_id, salary);
```

Queries that access all three columns, only the `last_name` column, or only the `last_name` and `job_id` columns use this index. In this example, queries that do not access the `last_name` column do not use the index.

Note:

In some cases, such as when the leading column has very low cardinality, the database may use a skip scan of this index (see "Index Skip Scan").

Multiple indexes can exist for the same table if the permutation of columns differs for each index. You can create multiple indexes using the same columns if you specify distinctly different permutations of the columns. For example, the following SQL statements specify valid permutations:

```
(customer_id);
```

在前面的语句中， customer_id 列是索引键。索引本身被命名为 ord_customer_ix。

注意：

主键和唯一键会自动生成索引，但您可能还需要在外键上创建索引

另见：

《*Oracle 数据库 SQL 语言参考*》关于 CREATE INDEX 的语法和语义

复合索引

复合索引，也称为**连接索引**，是在某个表中的多个列上的索引。复合索引中的列应该以在检索数据的查询中最有意义的顺序出现，但在表中不必是相邻的。

若 WHERE 子句引用了复合索引中的所有列或前导列，复合索引可以加快 SELECT 语句的数据检索速度。所以，在定义中所使用的列顺序很重要。一般地，最常被访问的列放在前面。

例如，假定应用程序经常查询雇员表中的 last_name、 job_id 和薪金列。同时假定 last_name 有很高的基数，也就是相对于表的总行数来说，不同值的数目很大。按以下的列顺序创建索引：

```
CREATE INDEX employees_ix
   ON employees (last_name, job_id, salary);
```

如果查询会访问所有的三个列，或仅 last_name 列，或仅 last_name 和 job_id 列，则会使用此索引。在本例中，不访问 last_name 列的查询，将不会使用索引。

注意：

在某些情况下，例如，若前导列的基数很低，则数据库可能使用索引跳跃扫描 （请参见"索引跳跃扫描"）。

在同一个表上可以存在多个索引，只要其索引列具有不同的排列顺序。如果您对多个列指定不同的排列顺序，您可以在这些相同的列上创建多个索引。例如，下面的 SQL 语句指定的排列都是有效的：

```
CREATE INDEX employee_idx1 ON employees (last_name, job_id);
CREATE INDEX employee_idx2 ON employees (job_id, last_name);
```

See Also:

Oracle Database Performance Tuning Guide for more information about using composite indexes

Unique and Nonunique Indexes

Indexes can be **unique** or nonunique. Unique indexes guarantee that no two rows of a table have duplicate values in the key column or column. For example, no two employees can have the same employee ID. Thus, in a unique index, one rowid exists for each data value. The data in the leaf blocks is sorted only by key.

Nonunique indexes permit duplicates values in the indexed column or columns. For example, the `first_name` column of the `employees` table may contain multiple `Mike` values. For a nonunique index, the rowid is included in the key in sorted order, so nonunique indexes are sorted by the index key and rowid (ascending).

Oracle Database does not index table rows in which all key columns are null, except for bitmap indexes or when the cluster key column value is null.

Types of Indexes

Oracle Database provides several indexing schemes, which provide complementary performance functionality. The indexes can be categorized as follows:

- B-tree indexes

  These indexes are the standard index type. They are excellent for primary key and highly-selective indexes. Used as concatenated indexes, B-tree indexes can retrieve data sorted by the indexed columns. B-tree indexes have the following subtypes:
  - Index-organized tables

    An index-organized table differs from a heap-organized because the data is itself the index. See "Overview of Index-Organized Tables".
  - Reverse key indexes

    In this type of index, the bytes of the index key are reversed, for example, 103 is stored as 301. The reversal of

```
CREATE INDEX employee_idx1 ON employees (last_name, job_id);
CREATE INDEX employee_idx2 ON employees (job_id, last_name);
```

另见：

《*Oracle 数据库性能优化指南*》有关使用复合索引的详细信息

唯一索引和非唯一索引

索引可以是唯一的或非唯一的。唯一索引保证在表的键列或键列集上没有具有重复的值的行。例如，没有任何两名雇员，可以有相同的雇员 id。因此，在一个唯一索引中，对每个数据值都存在一个 rowid。叶块中的数据仅根据键排序。

非唯一索引允许在索引的列或列集中有重复的值。例如，雇员表的 first_name 列中可能包含多个 Mike 值。对于非唯一索引，rowid 被包含在键中且已排序，因此非唯一索引按索引键和 rowid （升序） 进行排序。

除了位图索引或簇键列值为空的情况之外，Oracle 数据库不会索引所有键列都为空的表行。

索引类型

Oracle 数据库提供了几种索引模式，以提供增强性能的功能。索引可以分为以下几类：

- B-树索引

  这是索引的标准类型。他们对于主键和高选择性索引非常适合。在复合索引中使用时，B-树索引可以按多个索引列以排序方式检索数据。B-树索引具有下列子类型：
  - 索引组织表

    索引组织表不同于堆组织表，因为数据本身就是索引。请参见"索引组织表概述"。
  - 反键索引

    在这种类型的索引中，索引键中的字节被反转了，例如， 103 被存储为 301。反转字节可以把对索引记录的插入分散到的很

bytes spreads out inserts into the index over many blocks. See "Reverse Key Indexes".

- Descending indexes

  This type of index stores data on a particular column or columns in descending order. See "Ascending and Descending Indexes".

- B-tree cluster indexes

  This type of index is used to index a table cluster key. Instead of pointing to a row, the key points to the block that contains rows related to the cluster key. See "Overview of Indexed Clusters".

- Bitmap and bitmap join indexes

  In a bitmap index, an index entry uses a bitmap to point to multiple rows. In contrast, a B-tree index entry points to a single row. A bitmap join index is a bitmap index for the join of two or more tables. See "Bitmap Indexes".

- Function-based indexes

  This type of index includes columns that are either transformed by a function, such as the `UPPER` function, or included in an expression. B-tree or bitmap indexes can be function-based. See "Function-Based Indexes".

- Application domain indexes

  This type of index is created by a user for data in an application-specific domain. The physical index need not use a traditional index structure and can be stored either in the Oracle database as tables or externally as a file. See "Application Domain Indexes".

See Also:

Oracle Database Performance Tuning Guide to learn about different index types

B-Tree Indexes

B-trees, short for **balanced trees**, are the most common type of database index. A B-tree index is an ordered list of values divided into ranges. By associating a key with a row or range of rows, B-trees provide

多数据块。请参阅"反键索引"。

- 降序索引

  这种类型的索引将存储在一个特定的列或多列中的数据按降序排序。请参阅"升序和降序索引"。

- B-树簇索引

  这种类型的索引用于索引(vt)表簇键。它的键指向包含簇键相关行所在块，而不是指向行。请参见"索引化簇概述"。

- 位图索引和位图联接索引

  在位图索引中，索引条目使用位图来指向多个行。相比之下， B-树索引条目指向单个行。位图联接索引是在两个或更多表的联接上的位图索引。请参阅"位图索引"。

- 基于函数的索引

  这种类型的索引包括经过一个函数（如 UPPER 函数）转换过的列，或包括在表达式中的列。B-树索引或位图索引都可以是基于函数的。请参阅"基于函数的索引"。

- 应用程序域索引

  这种类型的索引是由用户为一个特定的应用程序域中的数据创建的。其物理索引不需要使用传统的索引结构，可以存储为 Oracle 数据库表，或外部文件。请参见"应用程序域索引"。

另见：

《*Oracle 数据库性能优化指南*》了解不同的索引类型

B-树索引

平衡树，简称 B-树，是最常见的数据库索引类型。一个 B-树索引是被划分为多个范围的已排序的值列表。通过将键与一行或行范围关联起来，B-树可以对多种类型的查询提供优秀的检索性能，包括精确匹配和范围搜索等。

excellent retrieval performance for a wide range of queries, including exact match and range searches.

Figure 3-1 illustrates the structure of a B-tree index. The example shows an index on the `department_id` column, which is a foreign key column in the `employees` table.

Figure 3-1 Internal Structure of a B-tree Index

Description of "Figure 3-1 Internal Structure of a B-tree Index"

Branch Blocks and Leaf Blocks

A B-tree index has two types of blocks: **branch blocks** for searching and **leaf blocks** that store values. The upper-level branch blocks of a B-tree index contain index data that points to lower-level index blocks. In Figure 3-1, the root branch block has an entry `0-40`, which points to the

图 3-1 显示了一个 B-树索引的结构。该示例显示在 department_id 列上的一个索引，它是雇员表的一个外键。 页: 7

图 3-1 B-树索引的内部结构

Description of "Figure 3-1 Internal Structure of a B-tree Index"

分支块和页块

B-树索引有两种类型的块： 用于查找的分支块和用于存储值的叶块。B-树索引的上层分支块包含指向下层索引块的索引数据。在图 3-1 中，根分支块包含条目 0-40， 指向下一个分支级别中最左边的块。此分支块包含条目如 0-10 和 11-19 等。每个这些条目指向包含在该范围内键值的叶块。

leftmost block in the next branch level. This branch block contains entries such as `0-10` and `11-19`. Each of these entries points to a leaf block that contains key values that fall in the range.

A B-tree index is balanced because all leaf blocks automatically stay at the same depth. Thus, retrieval of any record from anywhere in the index takes approximately the same amount of time. The **height** of the index is the number of blocks required to go from the root block to a leaf block. The **branch level** is the height minus 1. In Figure 3-1, the index has a height of 3 and a branch level of 2.

B-树索引之所以是平衡的，是因为所有叶块都自动处于相同的深度。因此，在索引中从任意位置检索任意记录需要的时间基本上是相同的。索引的高度是从根块到叶块所需的块的数量。分支级别等于其高度减 1。在图 3-1 中，索引的高度为 3，分支级别为 2。

Branch blocks store the minimum key prefix needed to make a branching decision between two keys. This technique enables the database to fit as much data as possible on each branch block. The branch blocks contain a pointer to the child block containing the key. The number of keys and pointers is limited by the block size.

分支块用于存储在两个键之间作出分支决定所需的最小键前缀。这种技术使数据库在每个分支块上能够尽可能存放更多的数据。分支块包含一个指针，该指针指向包含该键的子块。键和指针的数量受限于块大小。

The leaf blocks contain every indexed data value and a corresponding rowid used to locate the actual row. Each entry is sorted by (key, rowid). Within a leaf block, a key and rowid is linked to its left and right sibling entries. The leaf blocks themselves are also doubly linked. In Figure 3-1 the leftmost leaf block (`0-10`) is linked to the second leaf block (`11-19`).

叶块包含每个被索引的数据值，和一个相应的用来定位实际行的 rowid。每个条目按（rowid，键）排序。在一个叶块内，键和 rowid 链接到其左右同级条目。这些叶块本身也双向链接在一起。如图 3-1，最左边的叶块 (0-10) 链接到第二个叶块 (11-19)。

Note:

Indexes in columns with character data are based on the binary values of the characters in the database character set.

注意：

在字符数据列上的索引，基于数据库字符集中字符的二进制值。页: 8

Index Scans

In an **index scan**, the database retrieves a row by traversing the index, using the indexed column values specified by the statement. If the database scans the index for a value, then it will find this value in *n* I/Os where *n* is the height of the B-tree index. This is the basic principle behind Oracle Database indexes.

索引扫描

在**索引扫描**中，数据库使用语句指定的索引列，通过遍历索引来检索行。数据库扫描索引，将使用 n 个 I/O 就能找到其要查找的值，其中 n 即是 B-树索引的高度。这是数据库索引背后的基本原理。

If a SQL statement accesses only indexed columns, then the database reads values directly from the index rather than from the table. If the statement accesses columns in addition to the indexed columns, then the database uses rowids to find the rows in the table. Typically, the database retrieves table data by alternately reading an index block and then a table block.

如果 SQL 语句仅访问被索引的列，那么数据库只需直接从索引中读取值，而不用读取表。如果该语句同时还需要访问除索引列之外的列，那么数据库会使用 rowids 来查找表中的行。通常，为检索表数据，数据库以交替方式先读取索引块，然后读取相应的表块。

See Also:

Oracle Database Performance Tuning Guide for detailed information about index scans

另见：

《*Oracle 数据库性能优化指南*》有关索引扫描的详细信息

Full Index Scan

完全索引扫描

In a **full index scan**, the database reads the entire index in order. A full index scan is available if a predicate (`WHERE` clause) in the SQL statement references a column in the index, and in some circumstances when no predicate is specified. A full scan can eliminate sorting because the data is ordered by index key.

Suppose that an application runs the following query:

```
SELECT department_id, last_name, salary
FROM   employees
WHERE  salary > 5000
ORDER BY department_id, last_name;
```

Also assume that `department_id`, `last_name`, and `salary` are a composite key in an index. Oracle Database performs a full scan of the index, reading it in sorted order (ordered by department ID and last name) and filtering on the salary attribute. In this way, the database scans a set of data smaller than the `employees` table, which contains more columns than are included in the query, and avoids sorting the data.

For example, the full scan could read the index entries as follows:

```
50,Atkinson,2800,rowid
60,Austin,4800,rowid
70,Baer,10000,rowid
80,Abel,11000,rowid
80,Ande,6400,rowid
110,Austin,7200,rowid
.
.
.
```

Fast Full Index Scan

A **fast full index scan** is a full index scan in which the database reads the index blocks in no particular order. The database accesses the data in the index itself, without accessing the table.

Fast full index scans are an alternative to a full table scan when the index contains all the columns that are needed for the query, and at least one column in the index key has the `NOT NULL` constraint.

For example, an application issues the following query, which does not include an `ORDER BY` clause:

```
SELECT last_name, salary
FROM   employees;
```

If the last name and salary are a composite key in an index, then a fast

在**完全索引扫描**中，数据库顺序读取整个索引。如果在 SQL 语句中的谓词（WHERE 子句）引用了一个索引列，或者在某些情况下未不指定任何谓词，此时可能使用完全索引扫描。完全扫描可以消除排序，因为数据本身就是基于索引键排过序的。

假设应用程序运行以下查询：

```
SELECT department_id, last_name, salary
FROM   employees
WHERE  salary > 5000
ORDER BY department_id, last_name;
```

此外假定 department_id last_name，和 salary 是一个复合索引键。Oracle 数据库会执行完全索引扫描，按顺序读取（按部门 ID 和姓氏顺序）并基于薪金属性进行筛选。通过这种方式，数据库只需扫描一个小于雇员表的数据集，而不用扫描那些未包含在查询中的列，并避免了对该数据进行排序。

例如，完全扫描可能这样读取索引条目，如下所示：

```
50,Atkinson,2800,rowid
60,Austin,4800,rowid
70,Baer,10000,rowid
80,Abel,11000,rowid
80,Ande,6400,rowid
110,Austin,7200,rowid
.
.
.
```

快速完全索引扫描

快速完全索引扫描是一种完全索引扫描，数据库并不按特定的顺序读取索引块。数据库仅访问索引本身中的数据，而无需访问表。 页: 9

当索引包含了查询所需的所有列，且索引键中至少一列具有 NOT NULL 约束时，快速完全索引扫描可以替代全表扫描。

例如，应用程序发出以下查询，不包含 ORDER BY 子句：

```
SELECT last_name, salary
FROM   employees;
```

如果姓氏和工资是一个复合索引键，那么快速完全索引扫描只需读取索引条

full index scan can read the index entries to obtain the requested information:

```
Baida,2900,rowid
Zlotkey,10500,rowid
Austin,7200,rowid
Baer,10000,rowid
Atkinson,2800,rowid
Austin,4800,rowid
.
.
.
```

Index Range Scan

An **index range scan** is an ordered scan of an index that has the following characteristics:

- One or more leading columns of an index are specified in conditions. A **condition** specifies a combination of one or more expressions and logical (Boolean) operators and returns a value of `TRUE`, `FALSE`, or `UNKNOWN`.
- 0, 1, or more values are possible for an index key.

The database commonly uses an index range scan to access selective data. The selectivity is the percentage of rows in the table that the query selects, with 0 meaning no rows and 1 meaning all rows. Selectivity is tied to a query predicate, such as `WHERE last_name LIKE 'A%'`, or a combination of predicates. A predicate becomes more selective as the value approaches 0 and less selective (or more unselective) as the value approaches 1.

For example, a user queries employees whose last names begin with `A`. Assume that the `last_name` column is indexed, with entries as follows:

```
Abel,rowid
Ande,rowid
Atkinson,rowid
Austin,rowid
Austin,rowid
Baer,rowid
.
.
```

目，就可以获取所需的信息：

```
Baida,2900,rowid
Zlotkey,10500,rowid
Austin,7200,rowid
Baer,10000,rowid
Atkinson,2800,rowid
Austin,4800,rowid
.
.
.
```

索引范围扫描

索引范围扫描是对索引的有序扫描，具有以下特点：

- 在条件中指定了一个或多个索引前导列。**条件**指定一个或多个表达式和逻辑（布尔）运算符的组合，并返回一个值（TRUE、FALSE，或 UNKNOWN）。
- 一个索引键可能对应 0 个、1 个或更多个值。

通常，数据库使用索引范围扫描来访问选择性的数据。选择性是查询所选择的数据占总行数的百分比，0 意味着没有任何行，1 表示所有行。选择性与一个（或多个）查询谓词相关，比如 `WHERE last_name LIKE 'A%'`。值越接近 0 的谓词越具有选择性，相反，越接近 1 的谓词则越不具有选择性。

例如，用户查询姓氏以 A 开头的雇员。假定 last_name 列已被索引，其索引条目如下所示：

```
Abel,rowid
Ande,rowid
Atkinson,rowid
Austin,rowid
Austin,rowid
Baer,rowid
.
.
```

.

The database could use a range scan because the `last_name` column is specified in the predicate and multiples rowids are possible for each index key. For example, two employees are named Austin, so two rowids are associated with the key `Austin`.

An index range scan can be bounded on both sides, as in a query for departments with IDs between 10 and 40, or bounded on only one side, as in a query for IDs over 40. To scan the index, the database moves backward or forward through the leaf blocks. For example, a scan for IDs between 10 and 40 locates the first index leaf block that contains the lowest key value that is 10 or greater. The scan then proceeds horizontally through the linked list of leaf nodes until it locates a value greater than 40.

Index Unique Scan

In contrast to an index range scan, an **index unique scan** must have either 0 or 1 rowid associated with an index key. The database performs a unique scan when a predicate references all of the columns in a `UNIQUE` index key using an equality operator. An index unique scan stops processing as soon as it finds the first record because no second record is possible.

As an illustration, suppose that a user runs the following query:

```
SELECT *
FROM   employees
WHERE  employee_id = 5;
```

Assume that the `employee_id` column is the primary key and is indexed with entries as follows:

```
1,rowid
2,rowid
4,rowid
5,rowid
6,rowid
.
.
.
```

In this case, the database can use an index unique scan to locate the rowid for the employee whose ID is 5.

Index Skip Scan

An **index skip scan** uses logical subindexes of a composite index. The

.

数据库可以使用范围扫描，因为在谓词中指定了 last_name 列，而且每个索引键可能对应多个 rowids。例如有两个雇员名叫 Austin，所以有两个 rowids 都与索引键 Austin 相关联。

索引范围扫描可以在两边都有边界，比如部门 ID 在 10 至 40 之间的查询，或只在一边有界，比如部门 id 在 40 以上的查询。为扫描索引，数据库将在叶块之间前后移动。例如，对于 ID 在 10 到 40 之间的扫描，将先定位到包含最低键值（大于或等于 10）的第一个索引叶块。然后顺着各个被链接的页结点水平推进，直到它找到一个大于 40 的值为止。

唯一索引扫描

相对于索引范围扫描，**唯一索引扫描**必须是 有 0 个 或 1 个 rowid 与索引键相关联。当一个谓词使用相等运算符引用了唯一索引键的所有列时，数据库将执行唯一扫描。只要找到了第一个记录，唯一索引扫描就停止处理，因为不可能有第二个记录满足条件。

作为演示，假定用户运行如下查询：

```
SELECT *
FROM   employees
WHERE  employee_id = 5;
```

假定 employee_id 列是主键，并具有如下的索引条目：

```
1,rowid
2,rowid
4,rowid
5,rowid
6,rowid
.
.
.
```

在这种情况下，数据库可以使用唯一索引扫描来定位 rowid，以找到 ID 为 5 的雇员

索引跳跃扫描

索引跳跃扫描使用复合索引的逻辑子索引。数据库“跳跃地”通过单个索

database "skips" through a single index as if it were searching separate indexes. Skip scanning is beneficial if there are few distinct values in the leading column of a composite index and many distinct values in the nonleading key of the index.

引，好像它在多个单独的索引中搜索一样。如果在复合索引前导键列中有少量不同值，而在非前导键列中有大量不同值，此时使用跳跃扫描是有益的。

The database may choose an index skip scan when the leading column of the composite index is not specified in a query predicate. For example, assume that you run the following query for a customer in the `sh.customers` table:

当在查询谓词中未指定组合索引的前导列时，数据库可能选择索引跳跃扫描。例如，假定您要在 sh.customers 表中查找一个客户，运行如下查询：

```
SELECT * FROM sh.customers WHERE cust_email =
'Abbey@company.com';
```

The `customers` table has a column `cust_gender` whose values are either `M` or `F`. Assume that a composite index exists on the columns (`cust_gender`, `cust_email`). Example 3-1 shows a portion of the index entries.

customers 表中有一列 cust_gender，其值为 M 或 F。假定在列 cust_gender 和 cust_email 上存在一个复合索引。示例 3-1 显示了索引条目的一部分。

Example 3-1 Composite Index Entries

```
F,Wolf@company.com,rowid
F,Wolsey@company.com,rowid
F,Wood@company.com,rowid
F,Woodman@company.com,rowid
F,Yang@company.com,rowid
F,Zimmerman@company.com,rowid
M,Abbassi@company.com,rowid
M,Abbey@company.com,rowid
```

示例 3-1 复合索引条目

```
F,Wolf@company.com,rowid
F,Wolsey@company.com,rowid
F,Wood@company.com,rowid
F,Woodman@company.com,rowid
F,Yang@company.com,rowid
F,Zimmerman@company.com,rowid
M,Abbassi@company.com,rowid
M,Abbey@company.com,rowid
```

The database can use a skip scan of this index even though `cust_gender` is not specified in the `WHERE` clause.

虽然未在 WHERE 子句中指定 cust_gender，数据库可以使用跳跃索引扫描。

In a skip scan, the number of logical subindexes is determined by the number of distinct values in the leading column. In Example 3-1, the leading column has two possible values. The database logically splits the index into one subindex with the key `F` and a second subindex with the key `M`.

在跳跃扫描中，逻辑子索引的数目决定于前导列中的非重复值的数目。在示例 3-1 中，前导列中有两个可能值。数据库在逻辑上将该索引拆分为一个具有 F 键的子索引和另一个具有 M 键的子索引。

When searching for the record for the customer whose email is `Abbey@company.com`, the database searches the subindex with the value `F` first and then searches the subindex with the value `M`. Conceptually, the database processes the query as follows:

当搜索电子邮件为 Abbey@company.com 的客户的记录时, 数据库首先搜索 F 值的子索引，然后搜索 M 值的子索引。从概念上讲，数据库这样处理查询，如下所示：

```
SELECT * FROM sh.customers WHERE cust_gender = 'F'
  AND cust_email = 'Abbey@company.com'
UNION ALL
SELECT * FROM sh.customers WHERE cust_gender = 'M'
  AND cust_email = 'Abbey@company.com';
```

See Also:

Oracle Database Performance Tuning Guide to learn more about skip scans

Index Clustering Factor

The **index clustering factor** measures row order in relation to an indexed value such as employee last name. The more order that exists in row storage for this value, the lower the clustering factor.

The clustering factor is useful as a rough measure of the number of I/Os required to read an entire table by means of an index:

- If the clustering factor is high, then Oracle Database performs a relatively high number of I/Os during a large index range scan. The index entries point to random table blocks, so the database may have to read and reread the same blocks over and over again to retrieve the data pointed to by the index.
- If the clustering factor is low, then Oracle Database performs a relatively low number of I/Os during a large index range scan. The index keys in a range tend to point to the same data block, so the database does not have to read and reread the same blocks over and over.

The clustering factor is relevant for index scans because it can show:

- Whether the database will use an index for large range scans
- The degree of table organization in relation to the index key
- Whether you should consider using an index-organized table, partitioning, or table cluster if rows must be ordered by the index key

For example, assume that the `employees` table fits into two data blocks. Table 3-1 depicts the rows in the two data blocks (the ellipses indicate data that is not shown).

Table 3-1 Contents of Two Data Blocks in the Employees Table

Data Block 1	Data Block 2

```
SELECT * FROM sh.customers WHERE cust_gender = 'F'
  AND cust_email = 'Abbey@company.com'
UNION ALL
SELECT * FROM sh.customers WHERE cust_gender = 'M'
  AND cust_email = 'Abbey@company.com';
```

另见：

《*Oracle 数据库性能优化指南*》了解跳跃扫描的更多信息

索引聚簇因子

索引聚簇因子用于测量相对于某个索引值（如雇员姓氏）的行顺序。被索引值的行存储得越有序，则聚簇因子越低。

作为一种粗略测量通过索引读取整个表所需的 I/O 数，聚簇因子非常有用：

- 如果聚簇因子较高，则在大型索引范围扫描过程中，数据库将执行相对较高数目的 I/O。索引条目指向随机表块，因此数据库可能必须一遍又一遍地来回重读索引所指向的同一数据块。
- 如果聚簇因子较低，则在大型索引范围扫描过程中数据库将执行相对较低数目的 I/O。在一个范围内的索引键倾向于指向相同的数据块，因此该数据库不必来回重读相同的数据块。

聚簇因子与索引扫描关系密切，因为它可以显示：

- 数据库是否会在大范围扫描中使用索引
- 相对于索引键的表组织程度
- 如果行必须按索引键顺序排列，是否应考虑使用索引组织表、 分区、或表簇

例如，假定雇员表存放在两个数据块中。表 3-1 描述了两个数据块中的行（省略号表示未显示的数据）。

表 3-1 雇员表中某两个数据块的内容

数据块 1	数据块 2

100	Steven	**King**	SKING	...	149	Eleni	**Zlotkey**	EZLOTKEY	...
156	Janette	**King**	JKING	...	200	Jennifer	**Whalen**	JWHALEN	...
115	Alexander	**Khoo**	AKHOO	...	.				
.					.				
.					.				
.					137	Renske	**Ladwig**	RLADWIG	...
116	Shelli	**Baida**	SBAIDA	...	173	Sundita	**Kumar**	SKUMAR	...
204	Hermann	**Baer**	HBAER	...	101	Neena	**Kochar**	NKOCHHAR	...
105	David	**Austin**	DAUSTIN	...					
130	Mozhe	**Atkinson**	MATKINSO	...					
166	Sundar	**Ande**	SANDE	...					
174	Ellen	**Abel**	EABEL	...					

Rows are stored in the blocks in order of last name (shown in bold). For example, the bottom row in data block 1 describes Abel, the next row up describes Ande, and so on alphabetically until the top row in block 1 for Steven King. The bottom row in block 2 describes Kochar, the next row up describes Kumar, and so on alphabetically until the last row in the block for Zlotkey.

Assume that an index exists on the last name column. Each name entry corresponds to a rowid. Conceptually, the index entries would look as follows:

```
Abel,block1row1
Ande,block1row2
Atkinson,block1row3
Austin,block1row4
Baer,block1row5
.
.
.
```

Assume that a separate index exists on the employee ID column. Conceptually, the index entries might look as follows, with employee IDs distributed in almost random locations throughout the two blocks:

```
100,block1row50
101,block2row1
102,block1row9
103,block2row19
104,block2row39
105,block1row4
```


100	Steven	**King**	SKING	...	149	Eleni	**Zlotkey**	EZLOTKEY	...
156	Janette	**King**	JKING	...	200	Jennifer	**Whalen**	JWHALEN	...
115	Alexander	**Khoo**	AKHOO	...	.				
.					.				
.					.				
.					137	Renske	**Ladwig**	RLADWIG	...
116	Shelli	**Baida**	SBAIDA	...	173	Sundita	**Kumar**	SKUMAR	...
204	Hermann	**Baer**	HBAER	...	101	Neena	**Kochar**	NKOCHHAR	...
105	David	**Austin**	DAUSTIN	...					
130	Mozhe	**Atkinson**	MATKINSO	...					
166	Sundar	**Ande**	SANDE	...					
174	Ellen	**Abel**	EABEL	...					

行以姓氏顺序（以粗体显示）存储在块中。例如，数据块 1 的最下面一行描述了 Abel, 往上一行描述了 Ande，如此等等，按字母顺序直到最上面的一行描述了 Steven King。数据块 2 中的最下面行描述了 Kochar，往上一行描述了 Kumar ，如此等等，按字母顺序直到最后一行描述了 Zlotkey。

假设在姓氏列上存在一个索引。每个姓氏条目对应于一个 rowid。从概念上讲，索引条目看起来如下所示：


```
Abel,block1row1
Ande,block1row2
Atkinson,block1row3
Austin,block1row4
Baer,block1row5
.
.
.
```


假设在雇员 ID 列上存在另一个单独的索引。从概念上讲，索引条目可能看起来像下面这样，雇员 id 几乎分布在这整个两个数据块的任意位置：


```
100,block1row50
101,block2row1
102,block1row9
103,block2row19
104,block2row39
105,block1row4
```

.
.
.

Example 3-2 queries the ALL_INDEXES view for the clustering factor for these two indexes. The clustering factor for EMP_NAME_IX is low, which means that adjacent index entries in a single leaf block tend to point to rows in the same data blocks. The clustering factor for EMP_EMP_ID_PK is high, which means that adjacent index entries in the same leaf block are much less likely to point to rows in the same data blocks.

Example 3-2 Clustering Factor

```
SQL> SELECT INDEX_NAME, CLUSTERING_FACTOR
  2  FROM ALL_INDEXES
  3  WHERE INDEX_NAME IN
('EMP_NAME_IX','EMP_EMP_ID_PK');

INDEX_NAME           CLUSTERING_FACTOR
-------------------- -----------------
EMP_EMP_ID_PK                       19
EMP_NAME_IX                          2
```

See Also:

Oracle Database Reference to learn about ALL_INDEXES

Reverse Key Indexes

A **reverse key index** is a type of B-tree index that physically reverses the bytes of each index key while keeping the column order. For example, if the index key is 20, and if the two bytes stored for this key in hexadecimal are C1,15 in a standard B-tree index, then a reverse key index stores the bytes as 15,C1.

Reversing the key solves the problem of contention for leaf blocks in the right side of a B-tree index. This problem can be especially acute in an Oracle Real Application Clusters (Oracle RAC) database in which multiple instances repeatedly modify the same block. For example, in an orders table the primary keys for orders are sequential. One instance in the cluster adds order 20, while another adds 21, with each instance writing its key to the same leaf block on the right-hand side of the index.

In a reverse key index, the reversal of the byte order distributes inserts across all leaf keys in the index. For example, keys such as 20 and 21 that would have been adjacent in a standard key index are now stored far

.
.
.

示例 3-2 所示的查询，通过 ALL_INDEXES 查看这两个索引的聚簇因子。EMP_NAME_IX 的聚簇因子较低，这意味着在一个单一叶块中的相邻索引条目倾向于指向同一个数据块中的行。EMP_EMP_ID_PK 的聚簇因子较高，这意味着在相同的叶块中的相邻索引条目不太可能指向同一个数据块中的行。

示例 3－2 聚簇因子

```
SQL> SELECT INDEX_NAME, CLUSTERING_FACTOR
  2  FROM ALL_INDEXES
  3  WHERE INDEX_NAME IN
('EMP_NAME_IX','EMP_EMP_ID_PK');

INDEX_NAME           CLUSTERING_FACTOR
-------------------- -----------------
EMP_EMP_ID_PK                       19
EMP_NAME_IX                          2
```

另见：

《*Oracle 数据库引用*》了解 ALL_INDEXES

反向键索引

反向键索引是一种 B-树索引，它在物理上反转每个索引键的字节，但保持列顺序不变。例如，如果索引键是 20，，并且在一个标准的 B-树索引中此键被存为十六进制的两个字节 C1 15， 那么反向键索引会将其存为 15，C1。

反向键解决了在 B-树索引右侧的的叶块争用问题。在 Oracle 真正应用集群（Oracle RAC） 数据库中的多个实例重复不断地修改同一数据块时，这个问题尤为严重。例如，在订单表中，订单的主键是连续的。在集群中的一个实例添加订单 20，而另一个实例添加订单 21，每个实例都将它的键写入索引右侧的相同叶块。

在一个反向键索引中，对字节顺序反转，会将插入分散到索引中的所有叶块。例如键 20 和 21，本来在一个标准键索引中会相邻，现在存储在相隔很远的独立的块中。这样，顺序插入产生的 I/O 被更均匀地分布了。

apart in separate blocks. Thus, I/O for insertions of sequential keys is more evenly distributed.

Because the data in the index is not sorted by column key when it is stored, the reverse key arrangement eliminates the ability to run an index range scanning query in some cases. For example, if a user issues a query for order IDs greater than 20, then the database cannot start with the block containing this ID and proceed horizontally through the leaf blocks.

See Also:

Oracle Database Performance Tuning Guide to learn about design considerations for reverse key indexes

Ascending and Descending Indexes

In an **ascending index**, Oracle Database stores data in ascending order. By default, character data is ordered by the binary values contained in each byte of the value, numeric data from smallest to largest number, and date from earliest to latest value.

For an example of an ascending index, consider the following SQL statement:

```
CREATE INDEX emp_deptid_ix ON
hr.employees(department_id);
```

Oracle Database sorts the `hr.employees` table on the `department_id` column. It loads the ascending index with the `department_id` and corresponding rowid values in ascending order, starting with `0`. When it uses the index, Oracle Database searches the sorted `department_id` values and uses the associated rowids to locate rows having the requested `department_id` value.

By specifying the `DESC` keyword in the `CREATE INDEX` statement, you can create a **descending index**. In this case, the index stores data on a specified column or columns in descending order. If the index in Figure 3-1 on the `employees.department_id` column were descending, then the leaf blocking containing `250` would be on the left side of the tree and block with `0` on the right. The default search through a descending index is from highest to lowest value.

Descending indexes are useful when a query sorts some columns ascending and others descending. For an example, assume that you create a composite index on the `last_name` and `department_id` columns as follows:

```
CREATE INDEX emp_name_dpt_ix ON
```

因为存储数据时，并没有按照键列排序，因此在某些情况下，反向键格式丧失了执行索引范围扫描查询的能力。例如，如果用户发出一个订单 id 大于 20 的查询，但数据库不能从包含此 ID 的块开始扫描并沿着叶块水平推进。

另见：

《*Oracle 数据库性能优化指南*》了解反键索引设计注意事项

升序和降序索引

对于**升序索引**，数据库按升序排列的顺序存储数据。默认情况下，字符数据按每个字节中包含的二进制值排序， 数值数据按从小到大排序，日期数据从早到晚排序。

举一个升序索引的例子，请考虑下面的 SQL 语句：

```
CREATE INDEX emp_deptid_ix ON
hr.employees(department_id);
```

Oracle 数据库对 hr.employees 表按 department_id 列进行排序。从 0 开始，按 department_id 列及相应的 rowid 值的升序顺序加载索引。使用此索引，数据库搜索已排序的 department_id 值，并使用相关联的 rowids 来定位包含所请求的 department_id 值的行。

通过在 CREATE INDEX 语句中指定 `DESC` 关键字，您可以创建一个**降序索引**。在这种情况下，索引在指定的一列或多列上按降序顺序存储数据。如果在图 3-1 中 employees.department_id 列上的索引为降序，则包含 250 的叶块会在树的左侧，而 0 在右侧。降序索引的默认搜索顺序是从最高值到最低值。

当要求查询按一些列升序而另一些列降序排序时，降序索引非常有用。例如，假定您在 last_name 列和 department_id 列上创建一个复合索引，如下所示：

```
CREATE INDEX emp_name_dpt_ix ON
```

```
hr.employees(last_name ASC, department_id DESC);
```

If a user queries `hr.employees` for last names in ascending order (A to Z) and department IDs in descending order (high to low), then the database can use this index to retrieve the data and avoid the extra step of sorting it.

See Also:

- *Oracle Database Performance Tuning Guide* to learn more about ascending and descending index searches
- *Oracle Database SQL Language Reference* for descriptions of the `ASC` and `DESC` options of `CREATE INDEX`

Key Compression

Oracle Database can use key compression to compress portions of the primary key column values in a B-tree index or an index-organized table. Key compression can greatly reduce the space consumed by the index. In general, index keys have two pieces, a **grouping piece** and a **unique piece**. Key compression breaks the index key into a **prefix entry**, which is the grouping piece, and a **suffix entry**, which is the unique or nearly unique piece. The database achieves compression by sharing the prefix entries among the suffix entries in an index block.

Note:

If a key is not defined to have a unique piece, then the database provides one by appending a rowid to the grouping piece.

By default, the prefix of a unique index consists of all key columns excluding the last one, whereas the prefix of a nonunique index consists of all key columns. For example, suppose that you create a composite index on the `oe.orders` table as follows:

```
CREATE INDEX orders_mod_stat_ix ON orders
( order_mode, order_status );
```

Many repeated values occur in the `order_mode` and `order_status` columns. An index block may have entries as shown in Example 3-3.

Example 3-3 Index Entries in Orders Table

```
online,0,AAAPvCAAFAAAAFaAAa
online,0,AAAPvCAAFAAAAFaAAg
online,0,AAAPvCAAFAAAAFaAAl
online,2,AAAPvCAAFAAAAFaAAm
online,3,AAAPvCAAFAAAAFaAAq
```

```
hr.employees(last_name ASC, department_id DESC);
```

一个对 hr.employees 用户查询，要求按姓氏以升序顺序 （A 到 Z） 而部门 id 以降序 （从高到低)排序，则数据库可以使用此索引检索数据并避免额外的排序步骤。

另见：

- 《*Oracle 数据库性能优化指南*》了解升序和降序索引搜索的详细信息的
- 《*Oracle 数据库 SQL 语言参考*》有关 `CREATE INDEX ASC` 和 `DESC` 选项的说明

键压缩

Oracle 数据库可以使用键压缩来压缩 B-树索引或索引组织表中的主键列值的部分。键压缩可以大大减少索引所使用的空间。

一般地，索引键包含两个片断，一个**分组片断**和一个**唯一片断**。键压缩将索引键分成两部分，即作为分组片断的前缀条目，和作为唯一或几乎唯一片断的后缀条目。数据库通过在一个索引块中的多个后缀条目之间共享前缀条目来实现压缩。

注意：

如果某个键未被定义为具有唯一片断，那么数据库会通过将一个 rowid 附加到分组片断来提供唯一片断。

默认情况下，一个唯一索引的前缀由除最后一列之外的所有键列组成，而非唯一索引的前缀包含所有键列。例如，假设您在 oe.orders 表上创建一个复合索引，如下所示：

```
CREATE INDEX orders_mod_stat_ix ON orders
( order_mode, order_status );
```

在 order_mode 和 order_status 列中有许多重复值出现。一个索引块中的条目可能如示例 3-3 中所示。

示例 3-3 订单表中的索引条目

```
online,0,AAAPvCAAFAAAAFaAAa
online,0,AAAPvCAAFAAAAFaAAg
online,0,AAAPvCAAFAAAAFaAAl
online,2,AAAPvCAAFAAAAFaAAm
online,3,AAAPvCAAFAAAAFaAAq
```

```
online,3,AAAPvCAAFAAAAFaAAt
```

In Example 3-3, the key prefix would consist of a concatenation of the `order_mode` and `order_status` values. If this index were created with default key compression, then duplicate key prefixes such as `online,0` and `online,2` would be compressed. Conceptually, the database achieves compression as shown in the following example:

```
online,0
AAAPvCAAFAAAAFaAAa
AAAPvCAAFAAAAFaAAg
AAAPvCAAFAAAAFaAAl
online,2
AAAPvCAAFAAAAFaAAm
online,3
AAAPvCAAFAAAAFaAAq
AAAPvCAAFAAAAFaAAt
```

Suffix entries form the compressed version of index rows. Each suffix entry references a prefix entry, which is stored in the same index block as the suffix entry.

Alternatively, you could specify a prefix length when creating a compressed index. For example, if you specified prefix length `1`, then the prefix would be `order_mode` and the suffix would be `order_status,rowid`. For the values in Example 3-3, the index would factor out duplicate occurrences of `online` as follows:

```
online
0,AAAPvCAAFAAAAFaAAa
0,AAAPvCAAFAAAAFaAAg
0,AAAPvCAAFAAAAFaAAl
2,AAAPvCAAFAAAAFaAAm
3,AAAPvCAAFAAAAFaAAq
3,AAAPvCAAFAAAAFaAAt
```

The index stores a specific prefix once per leaf block at most. Only keys in the leaf blocks of a B-tree index are compressed. In the branch blocks the key suffix can be truncated, but the key is not compressed.

See Also:

- *Oracle Database Administrator's Guide* to learn how to use compressed indexes

```
online,3,AAAPvCAAFAAAAFaAAt
```

示例 3-3 中，键前缀将包括 order_mode 和 order_status 的串联值。如果此索引按默认键压缩创建，那么重复键前缀（如 `online,0` 和 `online,2`）将会被压缩。从概念上讲，数据库按如下示例中所示实现压缩：

```
online,0
AAAPvCAAFAAAAFaAAa
AAAPvCAAFAAAAFaAAg
AAAPvCAAFAAAAFaAAl
online,2
AAAPvCAAFAAAAFaAAm
online,3
AAAPvCAAFAAAAFaAAq
AAAPvCAAFAAAAFaAAt
```

后缀条目形成索引行的压缩版本。每个后缀条目引用一个与其存储在相同索引块中的前缀条目。

或者，创建一个压缩索引时，您也可以指定前缀长度。例如，如果指定了前缀长度为 1 ，那么前缀将会是 order_mode，而后缀将会是 order_status，rowid。对于示例 3-3 中的数值，索引会提取重复出现的 `online` 作为前缀，如下所示：

```
online
0,AAAPvCAAFAAAAFaAAa
0,AAAPvCAAFAAAAFaAAg
0,AAAPvCAAFAAAAFaAAl
2,AAAPvCAAFAAAAFaAAm
3,AAAPvCAAFAAAAFaAAq
3,AAAPvCAAFAAAAFaAAt
```

对索引中的每个叶块，一个特定前缀最多被存储一次。只有在 B-树索引的叶块中的键会被压缩。在分支块中的键后缀可以被截断，但不会被压缩。

另见：

- 《*Oracle 数据库管理员指南*》了解如何使用压缩索引

- *Oracle Database VLDB and Partitioning Guide* to learn how to use key compression for partitioned indexes
- *Oracle Database SQL Language Reference* for descriptions of the `key_compression` clause of `CREATE INDEX`

Bitmap Indexes

In a **bitmap index**, the database stores a bitmap for each index key. In a conventional B-tree index, one index entry points to a single row. In a bitmap index, each index key stores pointers to multiple rows.

Bitmap indexes are primarily designed for data warehousing or environments in which queries reference many columns in an ad hoc fashion. Situations that may call for a bitmap index include:

- The indexed columns have low **cardinality**, that is, the number of distinct values is small compared to the number of table rows.
- The indexed table is either read-only or not subject to significant modification by DML statements.

For a data warehouse example, the `sh.customer` table has a `cust_gender` column with only two possible values: `M` and `F`. Suppose that queries for the number of customers of a particular gender are common. In this case, the `customer.cust_gender` column would be a candidate for a bitmap index.

Each bit in the bitmap corresponds to a possible rowid. If the bit is set, then the row with the corresponding rowid contains the key value. A mapping function converts the bit position to an actual rowid, so the bitmap index provides the same functionality as a B-tree index although it uses a different internal representation.

If the indexed column in a single row is updated, then the database locks the index key entry (for example, `M` or `F`) and not the individual bit mapped to the updated row. Because a key points to many rows, DML on indexed data typically locks all of these rows. For this reason, bitmap indexes are not appropriate for many OLTP applications.

See Also:

- *Oracle Database Performance Tuning Guide* to learn how to use bitmap indexes for performance
- *Oracle Database Data Warehousing Guide* to learn how to use bitmap indexes in a data warehouse

Bitmap Indexes on a Single Table

- 《*Oracle 数据库 VLDB 和分区指南*》了解如何对已分区索引使用键压缩
- 《*Oracle 数据库 SQL 语言参考*》有关的 CREATE INDEX 的 key_compression 子句的说明

位图索引

在**位图索引**中，数据库为每个索引键存储一个位图。在传统的 B-树索引中，一个索引条目指向单个行。在位图索引中，每个索引键存储指向多个行的指针。

位图索引主要用于数据仓库，或在以特定方式引用很多列的查询环境中。可能需要一个位图索引的情况包括：

- 索引列的**基数**较低，也就是说，不同值的数目相比表的总行数很小。
- 被索引的表是只读的，或 DML 语句不会对其进行重大修改。

举一个数据仓库的例子， sh.customer 表的 cust_gender 列只有两个可能的值：`M` 和 `F`。假设会经常查询某一性别的客户数目。在这种情况下，可以考虑在 customer.cust_gender 列上创建位图索引。

位图中的每一位对应于一个可能的 rowid。如果设置了某位，那么与其相应的 rowid 行包含该键值。映射函数将位的位置转换为一个实际的 rowid，所以，虽然位图索引使用不同的内部表示形式，但它提供了与 B-树索引相同的功能。

如果更新了某个单行中的索引列，那么数据库将锁定整个索引键条目（例如 M 或 F），而不只是该位映射到的更新行。因为一个键指向多个行，DML 通常会锁定索引数据的所有这些行。因此，位图索引并不适合许多 OLTP 应用程序。

另见：

- 《*Oracle 数据库性能优化指南*》了解如何使用位图索引以增强性能
- 《*Oracle 数据库数据存储指南*》了解如何使用数据仓库中的位图索引

单表的位图索引

Example 3-4 shows a query of the `sh.customers` table. Some columns in this table are candidates for a bitmap index.

Example 3-4 Query of customers Table

```
SQL> SELECT cust_id, cust_last_name,
cust_marital_status, cust_gender
  2  FROM   sh.customers
  3  WHERE  ROWNUM < 8 ORDER BY cust_id;
   CUST_ID CUST_LAST_ CUST_MAR C
---------- ---------- -------- -
         1 Kessel              M
         2 Koch                F
         3 Emmerson            M
         4 Hardy               M
         5 Gowen               M
         6 Charles    single   F
         7 Ingram     single   F

7 rows selected.
```

The `cust_marital_status` and `cust_gender` columns have low cardinality, whereas `cust_id` and `cust_last_name` do not. Thus, bitmap indexes may be appropriate on `cust_marital_status` and `cust_gender`. A bitmap index is probably not useful for the other columns. Instead, a unique B-tree index on these columns would likely provide the most efficient representation and retrieval.

Table 3-2 illustrates the bitmap index for the `cust_gender` column output shown in Example 3-4. It consists of two separate bitmaps, one for each gender.

Table 3-2 Sample Bitmap

Value	Row 1	Row 2	Row 3	Row 4	Row 5	Row 6	Row 7
M	1	0	1	1	1	0	0
F	0	1	0	0	0	1	1

A mapping function converts each bit in the bitmap to a rowid of the `customers` table. Each bit value depends on the values of the corresponding row in the table. For example, the bitmap for the `M` value contains a `1` as its first bit because the gender is `M` in the first row of the `customers` table. The bitmap `cust_gender='M'` has a `0` for its the

示例 3-4 显示了对 sh.customers 表的查询。此表中的某些列是创建位图索引的候选对象。

示例 3-4 客户表查询

```
SQL> SELECT cust_id, cust_last_name,
cust_marital_status, cust_gender
  2  FROM   sh.customers
  3  WHERE  ROWNUM < 8 ORDER BY cust_id;
   CUST_ID CUST_LAST_ CUST_MAR C
---------- ---------- -------- -
         1 Kessel              M
         2 Koch                F
         3 Emmerson            M
         4 Hardy               M
         5 Gowen               M
         6 Charles    single   F
         7 Ingram     single   F

7 rows selected.
```

cust_marital_status 和 cust_gender 列的基数较低，而 cust_id 和 cust_last_name 列的基数较高。因此，在 cust_marital_status 和 cust_gender 列上创建位图索引可能是恰当的。而在其它列上创建位图索引则可能没什么用处。相反，在这些列上创建 B-树唯一索引可能会提供最有效的表示形式和检索性能。

表 3-2 说明了示例 3-4 中所示的 cust_gender 列的位图索引。它包括两个分别针对每个性别的单独位图。

表 3－2 位图示例

值	行 1	行 2	行 3	行 4	行 5	行 6	行 7
M	1	0	1	1	1	0	0
F	0	1	0	0	0	1	1

映射函数将位图中的每一位转换为客户表中的一个 rowid。每一位的值取决于表中的相应行中的值。例如， M 值的位图其第一位包含 1，因为客户表的第一行的性别是 M。对于位图 cust_gender ='M '， 与第 2、 6，7 行中相应的位为 0，因为这些行上包含的值不是 M。

bits in rows 2, 6, and 7 because these rows do not contain M as their value.

Note:

Bitmap indexes can include keys that consist entirely of null values, unlike B-tree indexes. Indexing nulls can be useful for some SQL statements, such as queries with the aggregate function COUNT.

An analyst investigating demographic trends of the customers may ask, "How many of our female customers are single or divorced?" This question corresponds to the following SQL query:

```
SELECT COUNT(*)
FROM   customers
WHERE  cust_gender = 'F'
AND    cust_marital_status IN ('single',
'divorced');
```

Bitmap indexes can process this query efficiently by counting the number of 1 values in the resulting bitmap, as illustrated in Table 3-3. To identify the customers who satisfy the criteria, Oracle Database can use the resulting bitmap to access the table.

Table 3-3 Sample Bitmap

Value	Row 1	Row 2	Row 3	Row 4	Row 5	Row 6	Row 7
M	1	0	1	1	1	0	0
F	0	1	0	0	0	1	1
single	0	0	0	0	0	1	1
divorced	0	0	0	0	0	0	0
single or divorced, and F	0	0	0	0	0	1	1

Bitmap indexing efficiently merges indexes that correspond to several conditions in a WHERE clause. Rows that satisfy some, but not all, conditions are filtered out before the table itself is accessed. This technique improves response time, often dramatically.

Bitmap Join Indexes

A **bitmap join index** is a bitmap index for the join of two or more tables. For each value in a table column, the index stores the rowid of the corresponding row in the indexed table. In contrast, a standard bitmap

注意：

与 B-树索引不同，位图索引可以包括完全由空值组成的键。对空值建立索引对于某些的 SQL 语句是有用的，比如使用 COUNT 聚合函数的查询。

调查客户的人口趋势的分析人员可能会问，"我们的女性客户中，有多少是单身或离婚的？" 这一问题对应于以下的 SQL 查询：

```
SELECT COUNT(*)
FROM   customers
WHERE  cust_gender = 'F'
AND    cust_marital_status IN ('single',
'divorced');
```

位图索引可以高效地处理此查询，通过计算得到的位图作为插图表 3-3 在 1 值的数量。要确定满足条件的客户，数据库可以访问表使用得到的位图。

表 3—3 位图示例

值	行 1	行 2	行 3	行 4	行 5	行 6	行 7
M	1	0	1	1	1	0	0
F	0	1	0	0	0	1	1
single	0	0	0	0	0	1	1
divorced	0	0	0	0	0	0	0
single 或 divorced，和 F	0	0	0	0	0	1	1

位图索引可以有效地合并与 WHERE 子句中的多个条件相对应的索引。满足一些但不是全部条件的行，将在表本身被访问之前被过滤掉。这种技术往往大大提高了响应的时间。

位图联接索引

位图联接索引是建立在两个或更多表的联接之上的位图索引。对于表列中的每个值，索引存储被索引表中的相应行的 rowid。相比之下，在标准位图索引中，索引是建立在一个表上的。

index is created on a single table.

A bitmap join index is an efficient means of reducing the volume of data that must be joined by performing restrictions in advance. For an example of when a bitmap join index would be useful, assume that users often query the number of employees with a particular job type. A typical query might look as follows:

```
SELECT COUNT(*)
FROM   employees, jobs
WHERE  employees.job_id = jobs.job_id
AND    jobs.job_title = 'Accountant';
```

The preceding query would typically use an index on `jobs.job_title` to retrieve the rows for `Accountant` and then the job ID, and an index on `employees.job_id` to find the matching rows. To retrieve the data from the index itself rather than from a scan of the tables, you could create a bitmap join index as follows:

```
CREATE BITMAP INDEX employees_bm_idx
ON     employees (jobs.job_title)
FROM   employees, jobs
WHERE  employees.job_id = jobs.job_id;
```

As illustrated in Figure 3-2, the index key is `jobs.job_title` and the indexed table is `employees`.

Figure 3-2 Bitmap Join Index

通过在联接前预先执行限制条件，位图联接索引是减少数据量的有效方式。举一个可能会用到位图联接索引的示例，假定用户经常查询某种特定职位类型的雇员数。一个典型的查询可能如下所示：

```
SELECT COUNT(*)
FROM   employees, jobs
WHERE  employees.job_id = jobs.job_id
AND    jobs.job_title = 'Accountant';
```

上述查询通常使用 jobs.job_title 上的一个索引来检索职位表中的 `Accountant` 行，然后通过（公共列）职位 ID，和员工表中 `employees.job_id` 列上的索引来找到匹配的行。若要从索引本身检索数据，而不是从表中扫描，您可以创建一个位图联接索引，如下所示：

```
CREATE BITMAP INDEX employees_bm_idx
ON     employees (jobs.job_title)
FROM   employees, jobs
WHERE  employees.job_id = jobs.job_id;
```

如图 3-2，索引键是 `jobs.job_title`， 而被索引的表是 `employees`。

*图 **3－2** 位图联接索引*

Description of "Figure 3-2 Bitmap Join Index"

Conceptually, `employees_bm_idx` is an index of the `jobs.title` column in the SQL query shown in Example 3-5 (sample output included). The `job_title` key in the index points to rows in the `employees` table. A query of the number of accountants can use the index to avoid accessing the `employees` and `jobs` tables because the index itself contains the requested information.

Example 3-5 Join of employees and jobs Tables

```
SELECT jobs.job_title AS "jobs.job_title",
employees.rowid AS "employees.rowid"
FROM   employees, jobs
WHERE  employees.job_id = jobs.job_id
ORDER BY job_title;
```

Description of "Figure 3-2 Bitmap Join Index"

从概念上讲，employees_bm_idx 是建立在示例 3-5（包括示例输出）所示的 SQL 查询中的 jobs.title 列上的索引。在索引中的 job_title 键指向雇员表中的行。对会计师数目的查询可以仅使用索引，而不用访问 `employees` 表和 `jobs` 表，因为该索引本身已包含所请求的信息。

例 3－5 员工表和职位表的连接

```
SELECT jobs.job_title AS "jobs.job_title",
employees.rowid AS "employees.rowid"
FROM   employees, jobs
WHERE  employees.job_id = jobs.job_id
ORDER BY job_title;
```

```
jobs.job_title                       employees.rowid
------------------------------------ ------------------
Accountant                           AAAQNKAAFAAAABSAAL
Accountant                           AAAQNKAAFAAAABSAAN
Accountant                           AAAQNKAAFAAAABSAAM
Accountant                           AAAQNKAAFAAAABSAAJ
Accountant                           AAAQNKAAFAAAABSAAK
Accounting Manager                   AAAQNKAAFAAAABTAAH
Administration Assistant             AAAQNKAAFAAAABTAAC
Administration Vice President        AAAQNKAAFAAAABSAAC
Administration Vice President        AAAQNKAAFAAAABSAAB
.
.
.
```

In a data warehouse, the join condition is an **equijoin** (it uses the equality operator) between the primary key columns of the dimension tables and the foreign key columns in the fact table. Bitmap join indexes are sometimes much more efficient in storage than materialized join views, an alternative for materializing joins in advance.

See Also:

Oracle Database Data Warehousing Guide for more information on bitmap join indexes

Bitmap Storage Structure

Oracle Database uses a B-tree index structure to store bitmaps for each indexed key. For example, if `jobs.job_title` is the key column of a bitmap index, then the index data is stored in one B-tree. The individual bitmaps are stored in the leaf blocks.

Assume that the `jobs.job_title` column has unique values `Shipping Clerk`, `Stock Clerk`, and several others. A bitmap index entry for this index has the following components:

- The job title as the index key
- A low rowid and high rowid for a range of rowids
- A bitmap for specific rowids in the range

Conceptually, an index leaf block in this index could contain entries as follows:

```
Shipping Clerk,AAAPzRAAFAAAABSABQ,AAAPzRAAFAAAABSABZ,0010000100
Shipping Clerk,AAAPzRAAFAAAABSABa,AAAPzRAAFAAAABSABh,010010
Stock Clerk,AAAPzRAAFAAAABSAAa,AAAPzRAAFAAAABSAAc,1001001100
Stock Clerk,AAAPzRAAFAAAABSAAd,AAAPzRAAFAAAABSAAt,0101001001
```


```
jobs.job_title                       employees.rowid
------------------------------------ ------------------
Accountant                           AAAQNKAAFAAAABSAAL
Accountant                           AAAQNKAAFAAAABSAAN
Accountant                           AAAQNKAAFAAAABSAAM
Accountant                           AAAQNKAAFAAAABSAAJ
Accountant                           AAAQNKAAFAAAABSAAK
Accounting Manager                   AAAQNKAAFAAAABTAAH
Administration Assistant             AAAQNKAAFAAAABTAAC
Administration Vice President        AAAQNKAAFAAAABSAAC
Administration Vice President        AAAQNKAAFAAAABSAAB
.
.
.
```


在数据仓库中，连接条件是在维度表主键列和事实数据表的外键列之间的一个**等值连接** （使用相等运算符）。位图联接索引有时比物化视图的存储效率高得多，这是一种替代*提前物化连接*的方法。

另见：

《*Oracle 数据库数据仓库指南*》了解位图联合索引的详细信息

位图存储结构

Oracle 数据库使用一个 B-树索引结构来为每个索引键存储位图。例如 ， 如果 jobs.job_title 是一个位图索引的键列， 那么索引数据存储在一个 B-树中。单个位图存储在叶块中。

假定 jobs.job_title 列具有 `Shipping Clerk`、`Stock Clerk`、及其他几个唯一值。此位图索引的索引条目具有以下组件：

- 作为索引键的职位
- 一个 rowids 范围的低值 rowid 和高值 rowid
- 在该范围内的特定 rowids 的位图

从概念上讲，该索引中的一个索引叶块，可能包含如下所示的条目：


```
Shipping Clerk,AAAPzRAAFAAAABSABQ,AAAPzRAAFAAAABSABZ,0010000100
Shipping Clerk,AAAPzRAAFAAAABSABa,AAAPzRAAFAAAABSABh,010010
Stock Clerk,AAAPzRAAFAAAABSAAa,AAAPzRAAFAAAABSAAc,1001001100
Stock Clerk,AAAPzRAAFAAAABSAAd,AAAPzRAAFAAAABSAAt,0101001001
```

```
Stock Clerk,AAAPzRAAFAAAABSAAu,AAAPzRAAFAAAABSABz,100001
.
.
.
```

The same job title appears in multiple entries because the rowid range differs.

Assume that a session updates the job ID of one employee from `Shipping Clerk` to `Stock Clerk`. In this case, the session requires exclusive access to the index key entry for the old value (`Shipping Clerk`) and the new value (`Stock Clerk`). Oracle Database locks the rows pointed to by these two entries—but not the rows pointed to by `Accountant` or any other key—until the `UPDATE` commits.

The data for a bitmap index is stored in one segment. Oracle Database stores each bitmap in one or more pieces. Each piece occupies part of a single data block.

See Also:

"User Segments"

Function-Based Indexes

You can create indexes on functions and expressions that involve one or more columns in the table being indexed. A **function-based index** computes the value of a function or expression involving one or more columns and stores it in the index. A function-based index can be either a B-tree or a bitmap index.

The function used for building the index can be an arithmetic expression or an expression that contains a SQL function, user-defined PL/SQL function, package function, or C callout. For example, a function could add the values in two columns.

See Also:

- *Oracle Database Administrator's Guide* to learn how to create function-based indexes
- *Oracle Database Performance Tuning Guide* for more information about using function-based indexes
- *Oracle Database SQL Language Reference* for restrictions and usage notes for function-based indexes

Uses of Function-Based Indexes

Function-based indexes are efficient for evaluating statements that contain functions in their `WHERE` clauses. The database only uses the function-

```
Stock Clerk,AAAPzRAAFAAAABSAAu,AAAPzRAAFAAAABSABz,100001
.
.
.
```

同一职位可能出现在多个条目中，这是因为其 rowid 范围不同。

假定某个会话更新了一名雇员的职务 ID，从 `Shipping Clerk` 到 `Stock Clerk`。在这种情况下，该会话需要对旧值（`Shipping Clerk`）和新值（`Stock Clerk`）所在索引键条目的独占访问权限。直到 `UPDATE` 提交之前，Oracle 数据库会锁定这两个条目所指向的所有行，而不是由 `Accountant` 或其他任何键所指向的行。

位图索引的数据存储在一个段中。Oracle 数据库将每个位图存储在一个或多个片断中。每个片断占一个单一数据块的一部分。

另见：

"用户段"

基于函数的索引

您可以基于函数、或涉及相关表的一个或多个列的表达式来创建索引。**基于函数的索引**计算函数或涉及一个或多个列的表达式的值，并将其存储在索引中。基于函数的索引可以是一个 B-树索引或位图索引。

用于生成索引的函数可以是算术表达式，或一个包含 SQL 函数、 用户定义 PL/SQL 函数、 包函数，或 C 调用的表达式。 例如，函数可以将两个列中的值相加。

另见：

- 《*Oracle 数据库管理员指南*》了解如何创建基于函数的索引
- 《*Oracle 数据库性能优化指南*》有关使用基于函数的索引的详细信息
- 《*Oracle 数据库 SQL 语言参考*》了解基于函数的索引的限制和使用注意事项

使用基于函数的索引

基于函数的索引对于在 WHERE 子句中包含函数计算的语句是有效的。仅当在查询中包含该函数时，数据库才使用基于函数的索引。当数据库处理

based index when the function is included in a query. When the database processes INSERT and UPDATE statements, however, it must still evaluate the function to process the statement.

For example, suppose you create the following function-based index:

```
CREATE INDEX emp_total_sal_idx
  ON employees (12 * salary * commission_pct,
salary, commission_pct);
```

The database can use the preceding index when processing queries such as Example 3-6 (partial sample output included).

Example 3-6 Query Containing an Arithmetic Expression

```
SELECT   employee_id, last_name, first_name,
         12*salary*commission_pct AS "ANNUAL SAL"
FROM     employees
WHERE    (12 * salary * commission_pct) < 30000
ORDER BY "ANNUAL SAL" DESC;

EMPLOYEE_ID LAST_NAME                 FIRST_NAME           ANNUAL SAL
----------- ------------------------- -------------------- ----------
        159 Smith                     Lindsey                   28800
        151 Bernstein                 David                     28500
        152 Hall                      Peter                     27000
        160 Doran                     Louise                    27000
        175 Hutton                    Alyssa                    26400
        149 Zlotkey                   Eleni                     25200
        169 Bloom                     Harrison                  24000
```

Function-based indexes defined on the SQL functions UPPER(*column_name*) or LOWER(*column_name*) facilitate case-insensitive searches. For example, suppose that the first_name column in employees contains mixed-case characters. You create the following function-based index on the hr.employees table:

```
CREATE INDEX emp_fname_uppercase_idx
ON employees ( UPPER(first_name) );
```

The emp_fname_uppercase_idx index can facilitate queries such as the following:

```
SELECT *
FROM   employees
WHERE  UPPER(first_name) = 'AUDREY';
```

A function-based index is also useful for indexing only specific rows in a table. For example, the cust_valid column in the sh.customers table has either I or A as a value. To index only the A rows, you could

INSERT 和 UPDATE 语句时，它仍然必须计算函数才能完成对语句的处理。

例如，假设您创建如下基于函数的索引：

```
CREATE INDEX emp_total_sal_idx
  ON employees (12 * salary * commission_pct,
salary, commission_pct);
```

当数据库处理如示例 3-6 （包括部分示例输出）所示的查询时，它可以使用前面定义的索引。

示例 3-6 包含算术表达式的查询

```
SELECT   employee_id, last_name, first_name,
         12*salary*commission_pct AS "ANNUAL SAL"
FROM     employees
WHERE    (12 * salary * commission_pct) < 30000
ORDER BY "ANNUAL SAL" DESC;

EMPLOYEE_ID LAST_NAME                 FIRST_NAME           ANNUAL SAL
----------- ------------------------- -------------------- ----------
        159 Smith                     Lindsey                   28800
        151 Bernstein                 David                     28500
        152 Hall                      Peter                     27000
        160 Doran                     Louise                    27000
        175 Hutton                    Alyssa                    26400
        149 Zlotkey                   Eleni                     25200
        169 Bloom                     Harrison                  24000
```

基于 SQL 函数 UPPER(column_name) 或 LOWER(column_name)定义的索引，使对大小写无关的搜索变得非常方便。例如，假设雇员表的 first_name 列包含混合大小写字符。您在 hr.employees 表上创建了如下基于函数的索引：

```
CREATE INDEX emp_fname_uppercase_idx
ON employees ( UPPER(first_name) );
```

emp_fname_uppercase_idx 索引使如下所示的查询变得非常方便：

```
SELECT *
FROM   employees
WHERE  UPPER(first_name) = 'AUDREY';
```

基于函数的索引对于只在一个表中的特定行上建立索引也是有用的。例如 sh.customers 表中的 cust_valid 列有 I 或 A 两个值。如果只想对含 A 值的行建立索引，您可以编写一个函数，让不含 A 值的其它任何行都返回一个

write a function that returns a null value for any rows other than the A rows. You could create the index as follows:

```
CREATE INDEX cust_valid_idx
ON customers ( CASE cust_valid WHEN 'A' THEN 'A'
END );
```

See Also:

- *Oracle Database Globalization Support Guide* for information about linguistic indexes
- *Oracle Database SQL Language Reference* to learn more about SQL functions

Optimization with Function-Based Indexes

The optimizer can use an index range scan on a function-based index for queries with expressions in WHERE clause. The range scan access path is especially beneficial when the predicate (WHERE clause) has low selectivity. In Example 3-6 the optimizer can use an index range scan if an index is built on the expression 12*salary*commission_pct.

A virtual column is useful for speeding access to data derived from expressions. For example, you could define virtual column annual_sal as 12*salary*commission_pct and create a function-based index on annual_sal.

The optimizer performs expression matching by parsing the expression in a SQL statement and then comparing the expression trees of the statement and the function-based index. This comparison is case-insensitive and ignores blank spaces.

See Also:

- "Overview of the Optimizer"
- *Oracle Database Performance Tuning Guide* for more information about gathering statistics
- *Oracle Database Administrator's Guide* to learn how to add virtual columns to a table

Application Domain Indexes

An **application domain index** is a customized index specific to an application. Oracle Database provides **extensible indexing** to do the following:

- Accommodate indexes on customized, complex data types such as documents, spatial data, images, and video clips (see "Unstructured

空值。可以这样创建索引，如下所示：

```
CREATE INDEX cust_valid_idx
ON customers ( CASE cust_valid WHEN 'A' THEN 'A'
END );
```

另见：

- 《*Oracle 数据库全球支持指南*》有关语言学索引的信息
- 《*Oracle 数据库 SQL 语言参考*》了解 SQL 函数的更多信息

优化基于函数的索引

对于在 WHERE 子句中包含表达式的查询，优化程序可以在一个基于函数的索引上使用索引范围扫描。当谓词（WHERE 子句）的选择性很低时，范围扫描的访问路径显得特别有优势。示例 3-6 中，如果索引是基于表达式 12*salary*commission_pct 来创建的，则优化程序可以使用索引范围扫描。

虚拟列可用于快速访问由表达式生成的数据。例如，您可以为表达式 12*salary*commission_pct 定义虚拟列 annual_sal，并在该 annual_sal 列上创建一个基于函数的索引。

优化程序通过分析在 SQL 语句中的表达式来执行表达式匹配，然后比较的语句表达式目录树和基于函数的索引。这种比较不区分大小写，并忽略空格。

另见：

- "优化器概述"
- 《*Oracle 数据库性能优化指南*》关于收集统计信息的详细信息
- 《*Oracle 数据库管理员指南*》了解如何向表中添加虚拟列

应用程序域索引

应用程序域索引是一个特定于应用程序的自定义索引。Oracle 数据库提供了**可扩展的索引**，以执行下列操作：

- 可以在自定义的、 复杂的数据类型（如文档、 空间数据、 图像、和

Data")

- Make use of specialized indexing techniques

You can encapsulate application-specific index management routines as an **indextype** schema object and define a domain index on table columns or attributes of an object type. Extensible indexing can efficiently process application-specific operators.
The application software, called the **cartridge**, controls the structure and content of a domain index. The database interacts with the application to build, maintain, and search the domain index. The index structure itself can be stored in the database as an index-organized table or externally as a file.
See Also:
Oracle Database Data Cartridge Developer's Guide for information about using data cartridges within the Oracle Database extensibility architecture

Index Storage

Oracle Database stores index data in an index segment. Space available for index data in a data block is the data block size minus block overhead, entry overhead, rowid, and one length byte for each value indexed.

The tablespace of an index segment is either the default tablespace of the owner or a tablespace specifically named in the `CREATE INDEX` statement. For ease of administration you can store an index in a separate tablespace from its table. For example, you may choose not to back up tablespaces containing only indexes, which can be rebuilt, and so decrease the time and storage required for backups.
See Also:
Chapter 12, "Logical Storage Structures"

Overview of Index-Organized Tables

An **index-organized table** is a table stored in a variation of a B-tree index structure. In a heap-organized table, rows are inserted where they fit. In an index-organized table, rows are stored in an index defined on the primary key for the table. Each index entry in the B-tree also stores the non-key column values. Thus, the index is the data, and the data is the index. Applications manipulate index-organized tables just like heap-organized tables, using SQL statements.
For an analogy of an index-organized table, suppose a human resources

视频剪辑等）之上建立索引，请参阅"非结构化数据"

- 使用专门的索引技术

可以将应用程序特定的索引管理例程封装为 **indextype** 的模式对象，并在表中的列或对象类型属性上定义一个域索引。可扩展的索引可以有效地处理特定于应用程序的操作。

由称为**模块**的应用软件，来控制域索引的结构和内容。数据库与应用程序进行交互，来建立、 维护、并搜索域索引。索引结构本身可以作为一个索引组织表存储在数据库中，或作为一个文件存储在数据库外部。

另见：
《*Oracle 数据库数据模块开发者指南*》关于在 Oracle 数据库可扩展性体系结构中使用数据模块的信息

索引存储

Oracle 数据库将索引数据存储在一个索引段中。在一个数据块中对索引数据可用的空间是，数据块大小减去块开销、 条目开销、 rowid 、和每个被索引的值所需的一个字节。
索引段的表空间，是所有者的默认表空间，或在 CREATE INDEX 语句中明确指定的表空间。为了便于管理，您可以在一个独立于表所在表空间的单独表空间中存储索引。例如，您可以选择不备份仅包含索引的表空间，因为索引表空间可以重建，这样可以减少备份所需要的时间和存储空间。

另见：
第 12 章, "逻辑存储结构"

索引组织表概述

索引组织表是存储在 B-树索引结构的一种变体中的表。在堆组织表中，行被插入到适合他们的位置。在索引组织表中，行被存储在表的主键索引中。在 B-树中的每个索引条目也存储非键列值。因此，索引即是数据，数据也是索引。同堆组织表一样，应用程序使用 SQL 语句操作索引组织表。

给索引组织表打个比喻，假设人事经理有一箱纸盒。给每个盒子贴上 1、

manager has a book case of cardboard boxes. Each box is labeled with a number—1, 2, 3, 4, and so on—but the boxes do not sit on the shelves in sequential order. Instead, each box contains a pointer to the shelf location of the next box in the sequence.

Folders containing employee records are stored in each box. The folders are sorted by employee ID. Employee King has ID 100, which is the lowest ID, so his folder is at the bottom of box 1. The folder for employee 101 is on top of 100, 102 is on top of 101, and so on until box 1 is full. The next folder in the sequence is at the bottom of box 2.

In this analogy, ordering folders by employee ID makes it possible to search efficiently for folders without having to maintain a separate index. Suppose a user requests the records for employees 107, 120, and 122. Instead of searching an index in one step and retrieving the folders in a separate step, the manager can search the folders in sequential order and retrieve each folder as found.

Index-organized tables provide faster access to table rows by primary key or a valid prefix of the key. The presence of non-key columns of a row in the leaf block avoids an additional data block I/O. For example, the salary of employee 100 is stored in the index row itself. Also, because rows are stored in primary key order, range access by the primary key or prefix involves minimal block I/Os. Another benefit is the avoidance of the space overhead of a separate primary key index.

Index-organized tables are useful when related pieces of data must be stored together or data must be physically stored in a specific order. This type of table is often used for information retrieval, spatial (see "Overview of Oracle Spatial"), and OLAP applications (see "OLAP").

See Also:

- *Oracle Database Administrator's Guide* to learn how to manage index-organized tables
- *Oracle Database Performance Tuning Guide* to learn how to use index-organized tables to improve performance
- *Oracle Database SQL Language Reference* for `CREATE TABLE ... ORGANIZATION INDEX` syntax and semantics

Index-Organized Table Characteristics

The database system performs all operations on index-organized tables by manipulating the B-tree index structure. Table 3-4 summarizes the differences between index-organized tables and heap-organized tables.

2、 3、 4 等标签，但这些盒子在书架上并不是顺序排放的。相反，每个盒子包含一个指针，指示序列中的下一个盒子的位置。

包含雇员记录的文件夹存储在每个盒子中。文件夹按雇员 id 排序。员工 King 的 ID 最低，为 100，所以他的文件夹在第 1 盒的底部。雇员 101 的文件夹在 100 的之上， 102 的在 101 的之上，如此等等，直到盒子 1 已满。序列中的下一个文件夹在盒子 2 的底部。

在这个比喻中，将文件夹按雇员 ID 排序，使得能够有效地搜索文件夹，而无需维护一个单独的索引。假设用户请求雇员 107、 120 、和 122 的记录。人事经理不需在一个步骤中搜索索引、然后又在另一步骤中检索文件夹，他可以按顺序搜索文件夹，并同时检索每个找到的文件夹。

通过主键或有效的前缀，索引组织表提供了对表行更快的访问。非键列存在于索引叶块中，避免了额外的数据块 I/O。例如，雇员 100 的薪酬存储在索引行本身中。此外，因为行是以主键顺序存储的，通过主键或前缀进行范围访问只涉及最少的块 I/O。另一个好处是避免一个单独的主键索引的空间开销。

当有关的数据片断必须存储在一起，或数据必须按特定的顺序物理地存储时，索引组织表很有用。这种类型的表通常用于信息检索、 空间 （请参阅"Oracle 空间概述"）、 和 OLAP 应用程序 （请参阅"OLAP"）。

另见

- 《*Oracle 数据库管理员指南*》了解如何管理组织索引表
- 《*Oracle 数据库性能优化指南*》了解如何使用索引组织表来提高性能
- 《*Oracle 数据库 SQL 语言参考*》关于 `CREATE TABLE ... ORGANIZATION INDEX` 的语法和语义

索引组织表特征

数据库系统通过操作 B-树索引结构，来在索引组织表上执行所有操作。表 3-4 总结了索引组织表和堆组织表之间的差异。

Table 3-4 Comparison of Heap-Organized Tables with Index-Organized Tables

Heap-Organized Table	Index-Organized Table
The rowid uniquely identifies a row. Primary key constraint may optionally be defined.	Primary key uniquely identifies a row. Primary key constraint must be defined.
Physical rowid in `ROWID` pseudocolumn allows building secondary indexes.	Logical rowid in `ROWID` pseudocolumn allows building secondary indexes.
Individual rows may be accessed directly by rowid.	Access to individual rows may be achieved indirectly by primary key.
Sequential full table scan returns all rows in some order.	A full index scan or fast full index scan returns all rows in some order.
Can be stored in a table cluster with other tables.	Cannot be stored in a table cluster.
Can contain a column of the `LONG` data type and columns of LOB data types.	Can contain LOB columns but not `LONG` columns.
Can contain virtual columns (only relational heap tables are supported).	Cannot contain virtual columns.

Figure 3-3 illustrates the structure of an index-organized `departments` table. The leaf blocks contain the rows of the table, ordered sequentially by primary key. For example, the first value in the first leaf block shows a department ID of `20`, department name of `Marketing`, manager ID of `201`, and location ID of `1800`.

Figure 3-3 Index-Organized Table

表 3-4 堆组织表和索引组织表的比较

堆组织表	索引组织表
由 rowid 唯一地标识行。定义主键约束是可选选项。	主键唯一地标识行。必须定义主键约束。
允许在 ROWID 伪列中的物理 rowid 上创建辅助索引。	允许在 ROWID 伪列中的逻辑 rowid 上创建辅助索引。
可直接由 rowid 访问单个行。	可间接通过主键访问单个行。
顺序全表扫描按一定顺序返回所有行。	完全索引扫描或快速完全索引扫描按一定顺序返回所有行。
可与其他表一起存储在表簇中。	不能存储在表簇中。
可以包含一个 LONG 数据类型的列，和多个 LOB 数据类型的列。	可以包含 LOB 列，但不能包含 `LONG` 列。
可以包含虚拟列 （只支持关系型堆表）。	不能包含虚拟列。

图 3-3 显示了一个索引组织表 `departments` 的结构。叶块包含的表行，按主键顺序排序。例如，在第一个叶块中的第一个值显示其部门 ID 20，部门名称为 `Marketing`、经理 ID 为 201、 位置 ID 为 1800。

图 3-3 索引组织表

Description of "Figure 3-3 Index-Organized Table"

An index-organized table stores all data in the same structure and does not need to store the rowid. As shown in Figure 3-3, leaf block 1 in an index-organized table might contain entries as follows, ordered by primary key:

```
20,Marketing,201,1800
30,Purchasing,114,1700
```

Leaf block 2 in an index-organized table might contain entries as follows:

```
50,Shipping,121,1500
60,IT,103,1400
```

A scan of the index-organized table rows in primary key order reads the blocks in the following sequence:

Description of "Figure 3-3 Index-Organized Table"

索引组织表将所有数据都存储在相同的结构中，且不需要存储 rowid。如图 3-3 ，索引组织表中的叶块 1 可能包含如下按主键排序的条目：

```
20,Marketing,201,1800
30,Purchasing,114,1700
```

索引组织表中的叶块 2 可能包含如下所示的条目：

```
50,Shipping,121,1500
60,IT,103,1400
```

按主键顺序对索引组织表行的扫描，依照如下顺序读取块：

- 1. Block 1
- 2. Block 2

To contrast data access in a heap-organized table to an index-organized table, suppose block 1 of a heap-organized `departments` table segment contains rows as follows:

```
50,Shipping,121,1500
20,Marketing,201,1800
```

Block 2 contains rows for the same table as follows:

```
30,Purchasing,114,1700
60,IT,103,1400
```

A B-tree index leaf block for this heap-organized table contains the following entries, where the first value is the primary key and the second is the rowid:

```
20,AAAPeXAAFAAAAAyAAD
30,AAAPeXAAFAAAAAyAAA
50,AAAPeXAAFAAAAAyAAC
60,AAAPeXAAFAAAAAyAAB
```

A scan of the table rows in primary key order reads the table segment blocks in the following sequence:

- 1. Block 1
- 2. Block 2
- 3. Block 1
- 4. Block 2

Thus, the number of block I/Os in this example is double the number in the index-organized example.

See Also:

- "Table Organization"
- "Introduction to Logical Storage Structures"

Index-Organized Tables with Row Overflow Area

When creating an index-organized table, you can specify a separate segment as a **row overflow area**. In index-organized tables, B-tree index entries can be large because they contain an entire row, so a separate segment to contain the entries is useful. In contrast, B-tree entries are usually small because they consist of the key and rowid.

If a row overflow area is specified, then the database can divide a row in

- 1. 块 1
- 2. 块 2

对比索引组织表和堆组织表中的数据访问，假定 `departments` 堆组织表段的块 1 包含的行，如下所示：

```
50,Shipping,121,1500
20,Marketing,201,1800
```

同一个表中块 2 包含的行，如下所示：

```
30,Purchasing,114,1700
60,IT,103,1400
```

此堆组织表的 B-树索引中的一个叶块包含如下条目，其中的第一个列值是主键，第二个列值是 rowid：

```
20,AAAPeXAAFAAAAAyAAD
30,AAAPeXAAFAAAAAyAAA
50,AAAPeXAAFAAAAAyAAC
60,AAAPeXAAFAAAAAyAAB
```

按主键顺序对表行的扫描，依照如下顺序读取块：

- 1. 块 1
- 2. 块 2
- 3. 块 1
- 4. 块 2

因此，在此示例中的块 I/O 数目是在索引组织表示例中的 2 倍。

另见：

- "表组织"
- "逻辑存储结构简介"

索引组织表的行溢出区

在创建一个索引组织表时，您可以指定一个单独的段为**行溢出区**。索引组织表的 B-树索引条目可能比较大，因为它们包含整个行，因此用一个单独的段来包含这些条目是很有用的。相比之下，常规 B-树条目则通常很小，因为它们仅包含键和 rowid。

如果指定了行溢出区，那么数据库可以将索引组织的表中的行分成以下两个

an index-organized table into the following parts:

- The index entry

  This part contains column values for all the primary key columns, a physical rowid that points to the overflow part of the row, and optionally a few of the non-key columns. This part is stored in the index segment.
- The overflow part

  This part contains column values for the remaining non-key columns. This part is stored in the overflow storage area segment.

See Also:

- *Oracle Database Administrator's Guide* to learn how to use the `OVERFLOW` clause of `CREATE TABLE` to set a row overflow area
- *Oracle Database SQL Language Reference* for `CREATE TABLE ... OVERFLOW` syntax and semantics

Secondary Indexes on Index-Organized Tables

A **secondary index** is an index on an index-organized table. In a sense, it is an index on an index. The secondary index is an independent schema object and is stored separately from the index-organized table.

As explained in "Rowid Data Types", Oracle Database uses row identifiers called **logical rowids** for index-organized tables. A logical rowid is a base64-encoded representation of the table primary key. The logical rowid length depends on the primary key length.

Rows in index leaf blocks can move within or between blocks because of insertions. Rows in index-organized tables do not migrate as heap-organized rows do (see "Chained and Migrated Rows"). Because rows in index-organized tables do not have permanent physical addresses, the database uses logical rowids based on primary key.

For example, assume that the `departments` table is index-organized. The `location_id` column stores the ID of each department. The table stores rows as follows, with the last value as the location ID:

```
10,Administration,200,1700
20,Marketing,201,1800
30,Purchasing,114,1700
40,Human Resources,203,2400
```

A secondary index on the `location_id` column might have index

部分：

- 索引条目

  本部分包含所有主键列的值、指向该行溢出部分的物理 rowid、 或（可选的）几个非键列的值。这部分存储在索引段中。

- 溢出部分

  此部分包含剩余的非键列的值。这部分存储在溢出存储区段中。

另见：

- 《*Oracle 数据库管理员指南*》了解如何使用 CREATE TABLE 的 `OVERFLOW` 子句设置行溢出区
- 《*Oacle 数据库 SQL 语言参考*》关于 `CREATE TABLE ... OVERFLOW` 的语法和语义

索引组织表的辅助索引

辅助索引是一个建立在索引组织表上的索引。在某种意义上，它是一个索引的索引。辅助索引是一个独立的模式对象，并与索引组织表分开存储。

如"Rowid 数据类型"中所述，数据库使用叫做**逻辑 rowids** 的行标识符来访问索引组织表。逻辑的 rowid 是表主键的 base64 编码表示形式。逻辑 rowid 的长度取决于主键长度。

由于插入操作，索引叶块中的行可以在块内或块之间移动。索引组织表中的行像堆组织表那样迁移行（请参阅"链接行和迁移行"）。因为索引组织表中的行并没有永久的物理地址，数据库使用基于主键的逻辑 rowids。

例如，假设 `departments` 表是索引组织表。`location_id` 列中存储每个部门的 ID。表像如下所示这样存储行，最后一个值是是位置 ID：

```
10,Administration,200,1700
20,Marketing,201,1800
30,Purchasing,114,1700
40,Human Resources,203,2400
```

在 location_id 列上的辅助索引可能有类似如下的索引条目，逗号之后的值

entries as follows, where the value following the comma is the logical rowid:

```
1700,*BAFAJqoCwR/+
1700,*BAFAJqoCwQv+
1800,*BAFAJqoCwRX+
2400,*BAFAJqoCwSn+
```

Secondary indexes provide fast and efficient access to index-organized tables using columns that are neither the primary key nor a prefix of the primary key. For example, a query of the names of departments whose ID is greater than 1700 could use the secondary index to speed data access.

See Also:

- *Oracle Database Administrator's Guide* to learn how to create secondary indexes on an index-organized table
- *Oracle Database VLDB and Partitioning Guide* to learn about creating secondary indexes on indexed-organized table partitions

Logical Rowids and Physical Guesses

Secondary indexes use the logical rowids to locate table rows. A logical rowid includes a **physical guess**, which is the physical rowid of the index entry when it was first made. Oracle Database can use physical guesses to probe directly into the leaf block of the index-organized table, bypassing the primary key search. When the physical location of a row changes, the logical rowid remains valid even if it contains a physical guess that is stale. For a heap-organized table, access by a secondary index involves a scan of the secondary index and an additional I/O to fetch the data block containing the row. For index-organized tables, access by a secondary index varies, depending on the use and accuracy of physical guesses:

- Without physical guesses, access involves two index scans: a scan of the secondary index followed by a scan of the primary key index.
- With physical guesses, access depends on their accuracy:
  - With accurate physical guesses, access involves a secondary index scan and an additional I/O to fetch the data block containing the row.
  - With inaccurate physical guesses, access involves a secondary index scan and an I/O to fetch the wrong data block (as indicated by the guess), followed by an index

是逻辑 rowid：

```
1700,*BAFAJqoCwR/+
1700,*BAFAJqoCwQv+
1800,*BAFAJqoCwRX+
2400,*BAFAJqoCwSn+
```

辅助索引对索引组织表提供快速、 高效的访问，但使用的列既不是主键也不是主键前缀。例如，查询其 ID 大于 1700 的部门的名称，可以使用辅助索引以加快数据访问。

另见：

- 《*Oracle 数据库管理员指南*》了解如何在索引组织表上创建辅助索引
- 《*Oracle 数据库 VLDB 和分区指南*》了解如何在索引组织表分区上创建辅助索引

逻辑 Rowids 和物理猜想

辅助索引使用逻辑 rowids 来查找表行。逻辑的 rowid 包括一个**物理猜测**，是索引条目第一次被创建时的物理 rowid 。Oracle 数据库可以使用物理猜测，直接探入索引组织表的叶块，以绕过主键搜索。当行的物理位置更改了，即使它包含的物理猜测已经过时，其逻辑 rowid 仍然保持有效。

对于堆组织表，按辅助索引访问涉及对辅助索引的扫描，和一个读取行所在数据块的额外 I/O。对于索引组织表，按辅助索引访问取决于物理猜测的使用及其准确性：

- 不用物理猜测，则访问包括两个索引扫描： 先是一个对辅助索引的扫描，然后是一个对主键索引的扫描。
- 使用物理猜测，则访问取决于其准确性：
  - 如果物理猜测准确，则访问包括一个辅助索引扫描，和一个读取行所在数据块的额外 I/O。
  - 如果物理猜测不准确，则访问包括一个辅助索引扫描，和一个读取错误的数据块 I/O （即猜测所指出的)，再是一个按主键

unique scan of the index organized table by primary key value.

Bitmap Indexes on Index-Organized Tables

A secondary index on an index-organized table can be a bitmap index. As explained in "Bitmap Indexes", a bitmap index stores a bitmap for each index key.

When bitmap indexes exist on an index-organized table, all the bitmap indexes use a heap-organized **mapping table**. The mapping table stores the logical rowids of the index-organized table. Each mapping table row stores one logical rowid for the corresponding index-organized table row. The database accesses a bitmap index using a search key. If the database finds the key, then the bitmap entry is converted to a physical rowid. With heap-organized tables, the database uses the physical rowid to access the base table. With index-organized tables, the database uses the physical rowid to access the mapping table, which in turn yields a logical rowid that the database uses to access the index-organized table. Figure 3-4 illustrates index access for a query of the `departments_iot` table.

Figure 3-4 Bitmap Index on Index-Organized Table

Description of "Figure 3-4 Bitmap Index on Index-Organized Table"

Note:

Movement of rows in an index-organized table does not leave the bitmap indexes built on that index-organized table unusable.

值对索引组织表的唯一索引扫描。

索引组织表上的位图索引

索引组织表上的辅助索引可以是位图索引。如"位图索引"所述，位图索引为每个索引键存储一个位图。

当索引组织表上存在位图索引时，所有位图索引都使用堆组织**映射表**。映射表将存储索引组织表的逻辑 rowids。每个映射表行会为相应的索引组织表行存储一个逻辑 rowid。

数据库使用搜索键来访问位图索引。如果数据库找到了该键，则该位图条目被转换为一个物理 rowid。对堆组织表，数据库使用物理 rowid 来访问基表。对索引组织表，数据库使用物理 rowid 来访问映射表，这样会生成一个逻辑 rowid ，以被数据库用来访问索引组织表。图 3-4 演示了一个通过索引访问 departments_iot 表的查询。

图 3-4 索引组织表上的位图索引

Description of "Figure 3-4 Bitmap Index on Index-Organized Table"

注意：

索引组织表中的行移动，不会使建立在该索引组织表上的位图索引变得不可用。

See Also:
"Rowids of Row Pieces"

另见：
"行片 Rowid"

4 Partitions, Views, and Other Schema Objects

Although tables and indexes are the most important and commonly used schema objects, the database supports many other types of schema objects, the most common of which are discussed in this chapter.

This chapter contains the following sections:

- Overview of Partitions
  - Partition Characteristics
  - Partitioned Tables
  - Partitioned Indexes
  - Partitioned Index-Organized Tables
- Overview of Views
  - Characteristics of Views
  - Updatable Join Views
  - Object Views
- Overview of Materialized Views
  - Characteristics of Materialized Views
  - Refresh Methods for Materialized Views
  - Query Rewrite
- Overview of Sequences
  - Sequence Characteristics
  - Concurrent Access to Sequences
- Overview of Dimensions
  - Hierarchical Structure of a Dimension
  - Creation of Dimensions
- Overview of Synonyms

Overview of Partitions

Partitioning enables you to decompose very large tables and indexes into smaller and more manageable pieces called **partitions**. Each partition is an independent object with its own name and optionally its own storage characteristics.

第 4 章 分区、 视图、和其它模式对象

虽然表和索引是最重要的和最常用的模式对象，但数据库也支持许多其他类型的模式对象，其中最常见的一些将在本章讨论。

本章包含以下各节：

- 分区概述
  - 分区特征
  - 分区表
  - 分区索引
  - 分区索引组织表
- 视图概述
  - 视图特征
  - 可更新联合视图
  - 对象视图
- 物化视图概述
  - 物化视图的特征
  - 物化视图的刷新方法
  - 查询重写
- 序列概述
  - 序列概述
  - 对序列的并发访问
- 维度概述
  - 维度的层次结构
  - 创建维度
- 同义词概述

分区概述

分区使你能够将非常大的表和索引分解成更小、 更易于管理的叫做**分区**的片断。每个分区是一个独立的对象，具有其自己的名称和 （可选的）的存储特征。

For an analogy that illustrates partitioning, suppose an HR manager has one big box that contains employee folders. Each folder lists the employee hire date. Queries are often made for employees hired in a particular month. One approach to satisfying such requests is to create an index on employee hire date that specifies the locations of the folders scattered throughout the box. In contrast, a partitioning strategy uses many smaller boxes, with each box containing folders for employees hired in a given month.

为说明分区，打个比方，假设一个人事经理有一只大箱子，其中包含雇员的文件夹。每个文件夹列出了雇员的雇用日期。经常会查询在一个特定月份雇用的雇员。要满足这些要求的一种方法是，对雇员的雇佣日期创建索引，它指向分散在箱子中的这些文件夹的位置。相比之下，分区策略将使用许多更小的箱子，每个小箱子仅包括在一个给定月份内雇用的雇员的文件夹。

Using smaller boxes has several advantages. When asked to retrieve the folders for employees hired in June, the HR manager can retrieve the June box. Furthermore, if any small box is temporarily damaged, the other small boxes remain available. Moving offices also becomes easier because instead of moving a single heavy box, the manager can move several small boxes.

使用更小的箱子具有几个优势。当要检索在 6 月份雇用的员工的文件夹时，人事经理只需检索 6 月份的箱子。此外，如果任何小箱子损坏，则并不影响其他小箱子保持可用。办公室搬家也会变得更容易，因为他不必移动一个很大的箱子，而只需移动几个小箱子。

From the perspective of an application, only one schema object exists. DML statements require no modification to access partitioned tables.

从应用程序的角度来看，只有一个模式对象已存在。DML 语句不需要作任何修改就可以访问该分区表。分区对许多不同类型的应用程序都很有用，特别是那些管理大量数据的应用程序。其好处包括：

Partitioning is useful for many different types of database applications, particularly those that manage large volumes of data. Benefits include:

- Increased availability

  The unavailability of a partition does not entail the unavailability of the object. The query optimizer automatically removes unreferenced partitions from the query plan so queries are not affected when the partitions are unavailable.

- 可用性增强

  其中某个分区不可用并不意味着整个对象不可用。当部分分区不可用时，查询优化器自动从查询计划中删除未引用的分区，而查询不会受影响。

- Easier administration of schema objects

  A partitioned object has pieces that can be managed either collectively or individually. DDL statements can manipulate partitions rather than entire tables or indexes. Thus, you can break up resource-intensive tasks such as rebuilding an index or table. For example, you can move one table partition at a time. If a problem occurs, then only the partition move must be redone, not the table move. Also, dropping a partition avoids executing numerous `DELETE` statements.

- 更轻松的管理模式对象

  已分区对象具有多个分片，可以将其作为一个整体来管理，也可以单独管理各个分片。DDL 语句可以处理分区，而不是整个表或索引。因此，您可以分解大量占用资源的任务，如重建索引或表。例如，您可以一次只移动一个表分区。如果发生了问题，则只需重新移动该分区，而不是整个表。而且，删除一个分区可以避免执行许多 DELETE 语句。

- Reduced contention for shared resources in OLTP systems

  In some OLTP systems, partitions can decrease contention for a shared resource. For example, DML is distributed over many

- 在 OLTP 系统中减少对共享资源的争用

  在一些 OLTP 系统中，分区可以减少对共享资源的争用。例如，DML 被分散到很多段，而不只是一个段。

segments rather than one segment.

- Enhanced query performance in data warehouses

  In a data warehouse, partitioning can speed processing of ad hoc queries. For example, a sales table containing a million rows can be partitioned by quarter.

See Also:

Oracle Database VLDB and Partitioning Guide for an introduction to partitioning

Partition Characteristics

Each partition of a table or index must have the same logical attributes, such as column names, data types, and constraints. For example, all partitions in a table share the same column and constraint definitions, and all partitions in an index share the same indexed columns. However, each partition can have separate physical attributes, such as the tablespace to which it belongs.

Partition Key

The **partition key** is a set of one or more columns that determines the partition in which each row in a partitioned table should go. Each row is unambiguously assigned to a single partition.

In the `sales` table, you could specify the `time_id` column as the key of a range partition. The database assigns rows to partitions based on whether the date in this column falls in a specified range. Oracle Database automatically directs insert, update, and delete operations to the appropriate partition by using the partition key.

Partitioning Strategies

Oracle Partitioning offers several partitioning strategies that control how the database places data into partitions. The basic strategies are range, list, and hash partitioning.

A **single-level** partitioning strategy uses only one method of data distribution, for example, only list partitioning or only range partitioning. In **composite partitioning**, a table is partitioned by one data distribution method and then each partition is further divided into subpartitions using a second data distribution method. For example, you could use a list partition for `channel_id` and a range subpartition for `time_id`.

Range Partitioning

In **range partitioning**, the database maps rows to partitions based on

- 在数据仓库中增强的查询性能

  在数据仓库中，分区可以加快处理即席查询。例如，包含一百万行的销售表可以按季度进行分区。

另见：

《*Oracle 数据库 VLDB 和分区指南*》关于分区的介绍

分区特征

每个表或索引的分区必须具有相同的逻辑属性，如列名称、 数据类型、和约束。例如，在一个表中的所有分区都共享相同的列和约束定义，并在索引中的所有分区都共享相同的索引列。但是，每个分区可以有单独的物理属性，如其所属的表空间。

分区键

分区键是一个列或列集，以确定分区表中的每一行应该所在的分区。每个行会被确定地（而不是模棱两可地）分配到某个分区。

在 `sales` 表中，您可以指定 time_id 列作为范围分区的键。基于此列中的日期是否属于某一特定范围，数据库将行分配到所属分区。通过使用分区键，Oracle 数据库将插入、 更新、和删除操作自动地指向适当的分区。

分区策略

Oracle 分区提供了几个分区策略，来控制数据库如何将数据放置到分区。基本策略有范围分区、 列表分区、和哈希分区等。

单一分区策略只使用一种数据分布方法，例如，仅使用列表分区，或仅使用范围分区。在**复合分区**中，表先按一种数据分布方法分区，然后每个分区使用第二种数据分布方法进一步分成子分区。例如，您可以使用 channel_id 划分列表分区，并使用 time_id 划分范围子分区。

范围分区

在**范围分区**中，数据库基于分区键的值范围将行映射到各个分区。范围分区

ranges of values of the partitioning key. Range partitioning is the most common type of partitioning and is often used with dates.

Suppose that you want to populate a partitioned table with the `sales` rows shown in Example 4-1.

Example 4-1 Sample Row Set for Partitioned Table

PROD_ID	CUST_ID	TIME_ID	CHANNEL_ID	PROMO_ID	QUANTITY_SOLD	AMOUNT_SOLD
116	11393	05-JUN-99	2	999	1	12.18
40	100530	30-NOV-98	9	33	1	44.99
118	133	06-JUN-01	2	999	1	17.12
133	9450	01-DEC-00	2	999	1	31.28
36	4523	27-JAN-99	3	999	1	53.89
125	9417	04-FEB-98	3	999	1	16.86
30	170	23-FEB-01	2	999	1	8.8
24	11899	26-JUN-99	4	999	1	43.04
35	2606	17-FEB-00	3	999	1	54.94
45	9491	28-AUG-98	4	350	1	47.45

You create `time_range_sales` as a partitioned table using the statement in Example 4-2. The `time_id` column is the partition key.

Example 4-2 Range-Partitioned Table

```
CREATE TABLE time_range_sales
   ( prod_id        NUMBER(6)
   , cust_id        NUMBER
   , time_id        DATE
   , channel_id     CHAR(1)
   , promo_id       NUMBER(6)
   , quantity_sold  NUMBER(3)
   , amount_sold    NUMBER(10,2)
   )
PARTITION BY RANGE (time_id)
 (PARTITION SALES_1998 VALUES LESS THAN
(TO_DATE('01-JAN-1999','DD-MON-YYYY')),
  PARTITION SALES_1999 VALUES LESS THAN
(TO_DATE('01-JAN-2000','DD-MON-YYYY')),
  PARTITION SALES_2000 VALUES LESS THAN
(TO_DATE('01-JAN-2001','DD-MON-YYYY')),
  PARTITION SALES_2001 VALUES LESS THAN (MAXVALUE)
 );
```

Afterward, you load `time_range_sales` with the rows from Example 4-1. Figure 4-1 shows the row distributions in the four partitions. The database chooses the partition for each row based on the `time_id` value according to the rules specified in the `PARTITION BY RANGE` clause.

是最常见的分区类型，通常与日期一起使用。

假设您要用示例 4-1 中所示的销售行填充分区表。

示例 4-1 分区表的示例行集

PROD_ID	CUST_ID	TIME_ID	CHANNEL_ID	PROMO_ID	QUANTITY_SOLD	AMOUNT_SOLD
116	11393	05-JUN-99	2	999	1	12.18
40	100530	30-NOV-98	9	33	1	44.99
118	133	06-JUN-01	2	999	1	17.12
133	9450	01-DEC-00	2	999	1	31.28
36	4523	27-JAN-99	3	999	1	53.89
125	9417	04-FEB-98	3	999	1	16.86
30	170	23-FEB-01	2	999	1	8.8
24	11899	26-JUN-99	4	999	1	43.04
35	2606	17-FEB-00	3	999	1	54.94
45	9491	28-AUG-98	4	350	1	47.45

使用示例 4-2 中的语句创建分区表 time_range_sales。time_id 列是分区键。

示例 4-2 范围分区表

```
CREATE TABLE time_range_sales
   ( prod_id        NUMBER(6)
   , cust_id        NUMBER
   , time_id        DATE
   , channel_id     CHAR(1)
   , promo_id       NUMBER(6)
   , quantity_sold  NUMBER(3)
   , amount_sold    NUMBER(10,2)
   )
PARTITION BY RANGE (time_id)
 (PARTITION SALES_1998 VALUES LESS THAN
(TO_DATE('01-JAN-1999','DD-MON-YYYY')),
  PARTITION SALES_1999 VALUES LESS THAN
(TO_DATE('01-JAN-2000','DD-MON-YYYY')),
  PARTITION SALES_2000 VALUES LESS THAN
(TO_DATE('01-JAN-2001','DD-MON-YYYY')),
  PARTITION SALES_2001 VALUES LESS THAN (MAXVALUE)
 );
```

然后，您可以将示例 4-1 中的示例行加载到 time_range_sales 表。图 4-1 显示了四个分区中的行分布。数据库根据 `PARTITION BY RANGE` 子句中指定的规则，基于 time_id 值为每个行选择适当的分区。

Figure 4-1 Range Partitions

Table Partition SALES_1998

PROD_ID	CUST_ID	TIME_ID	CHANNEL_ID	PROMO_ID	QUANTITY_SOLD	AMOUNT_SOLD
40	100530	30-NOV-98	9	33	1	44.99
125	9417	04-FEB-98	3	999	1	16.86
45	9491	28-AUG-98	4	350	1	47.45

Table Partition SALES_1999

PROD_ID	CUST_ID	TIME_ID	CHANNEL_ID	PROMO_ID	QUANTITY_SOLD	AMOUNT_SOLD
116	11393	05-JUN-99	2	999	1	12.18
36	4523	27-JAN-99	3	999	1	53.89
24	11899	26-JUN-99	4	999	1	43.04

Table Partition SALES_2000

PROD_ID	CUST_ID	TIME_ID	CHANNEL_ID	PROMO_ID	QUANTITY_SOLD	AMOUNT_SOLD
133	9450	01-DEC-00	2	999	1	31.28
35	2606	17-FEB-00	3	999	1	54.94

Table Partition SALES_2001

PROD_ID	CUST_ID	TIME_ID	CHANNEL_ID	PROMO_ID	QUANTITY_SOLD	AMOUNT_SOLD
118	133	06-JUN-01	2	999	1	17.12
30	170	23-FEB-01	2	999	1	8.8

Description of "Figure 4-1 Range Partitions"

The range partition key value determines the high value of the range partitions, which is called the **transition point**. In Figure 4-1, the `SALES_1998` partition contains rows with partitioning key `time_id` values less than the transition point `01-JAN-1999`.

The database creates **interval partitions** for data beyond that transition point. Interval partitions extend range partitioning by instructing the database to create partitions of the specified range or interval automatically when data inserted into the table exceeds all of the range

图 4-1 范围分区

Table Partition SALES_1998

PROD_ID	CUST_ID	TIME_ID	CHANNEL_ID	PROMO_ID	QUANTITY_SOLD	AMOUNT_SOLD
40	100530	30-NOV-98	9	33	1	44.99
125	9417	04-FEB-98	3	999	1	16.86
45	9491	28-AUG-98	4	350	1	47.45

Table Partition SALES_1999

PROD_ID	CUST_ID	TIME_ID	CHANNEL_ID	PROMO_ID	QUANTITY_SOLD	AMOUNT_SOLD
116	11393	05-JUN-99	2	999	1	12.18
36	4523	27-JAN-99	3	999	1	53.89
24	11899	26-JUN-99	4	999	1	43.04

Table Partition SALES_2000

PROD_ID	CUST_ID	TIME_ID	CHANNEL_ID	PROMO_ID	QUANTITY_SOLD	AMOUNT_SOLD
133	9450	01-DEC-00	2	999	1	31.28
35	2606	17-FEB-00	3	999	1	54.94

Table Partition SALES_2001

PROD_ID	CUST_ID	TIME_ID	CHANNEL_ID	PROMO_ID	QUANTITY_SOLD	AMOUNT_SOLD
118	133	06-JUN-01	2	999	1	17.12
30	170	23-FEB-01	2	999	1	8.8

Description of "Figure 4-1 Range Partitions"

范围分区键值确定各个范围分区的高值，被称为**跃点**。在图 4-1 中，分区 SALES_1998 包含分区键 time_id 值小于跃点 01-JAN-1999 的行。

数据库为超出跃点的数据创建**间隔分区**。当插入到表中的数据超出所有已定义的表分区时，间隔分区指示数据库自动创建特定范围或特定间隔的分区，来扩展范围分区。在图 4-1 中，分区 SALES_2001 包含分区键 time_id 值大于或等于 01-JAN-2001 的行。

partitions. In Figure 4-1, the SALES_2001 partition contains rows with partitioning key time_id values greater than or equal to 01-JAN-2001.

List Partitioning

In **list partitioning**, the database uses a list of discrete values as the partition key for each partition. You can use list partitioning to control how individual rows map to specific partitions. By using lists, you can group and organize related sets of data when the key used to identify them is not conveniently ordered.

Assume that you create list_sales as a list-partitioned table using the statement in Example 4-3. The channel_id column is the partition key.

Example 4-3 List-Partitioned Table

```
CREATE TABLE list_sales
   ( prod_id        NUMBER(6)
   , cust_id        NUMBER
   , time_id        DATE
   , channel_id     CHAR(1)
   , promo_id       NUMBER(6)
   , quantity_sold  NUMBER(3)
   , amount_sold    NUMBER(10,2)
   )
PARTITION BY LIST (channel_id)
 (PARTITION even_channels VALUES (2,4),
  PARTITION odd_channels VALUES (3,9)
 );
```

Afterward, you load the table with the rows from Example 4-1. Figure 4-2 shows the row distribution in the two partitions. The database chooses the partition for each row based on the channel_id value according to the rules specified in the PARTITION BY LIST clause. Rows with a channel_id value of 2 or 4 are stored in the EVEN_CHANNELS partitions, while rows with a channel_id value of 3 or 9 are stored in the ODD_CHANNELS partition.

Figure 4-2 List Partitions

列表分区

在**列表分区**中，数据库使用一些具体值的列表作为每个分区的分区键。你可以使用列表分区来控制单个行如何映射到特定的分区。当用来区分数据集的键不方便排序时，可以通过使用列表来分组和组织相关的数据集。

假设您使用示例 4-3 中的语句创建列表分区表 list_sales。channel_id 列为分区键。

示例 4-3 列表分区表

```
CREATE TABLE list_sales
   ( prod_id        NUMBER(6)
   , cust_id        NUMBER
   , time_id        DATE
   , channel_id     CHAR(1)
   , promo_id       NUMBER(6)
   , quantity_sold  NUMBER(3)
   , amount_sold    NUMBER(10,2)
   )
PARTITION BY LIST (channel_id)
 (PARTITION even_channels VALUES (2,4),
  PARTITION odd_channels VALUES (3,9)
 );
```

随后，将示例 4-1 中的表行加载到表中。图 4-2 显示了在两个分区中的行分布。数据库根据 PARTITION BY LIST 子句中指定的规则，基于 channel_id 为每个行选择适当的分区。channel_id 值为 2 或 4 的行存储在 EVEN_CHANNELS 分区中，而 channel_id 值 为 3 或 9 的行存储在 ODD_CHANNELS 分区中。

图 4-2 列表分区

Table Partition EVEN_CHANNELS

PROD_ID	CUST_ID	TIME_ID	CHANNEL_ID	PROMO_ID	QUANTITY_SOLD	AMOUNT_SOLD
116	11393	05-JUN-99	2	999	1	12.18
118	133	06-JUN-01	2	999	1	17.12
133	9450	01-DEC-00	2	999	1	31.28
30	170	23-FEB-01	2	999	1	8.8
24	11899	26-JUN-99	4	999	1	43.04
45	9491	28-AUG-98	4	350	1	47.45

Table Partition ODD_CHANNELS

PROD_ID	CUST_ID	TIME_ID	CHANNEL_ID	PROMO_ID	QUANTITY_SOLD	AMOUNT_SOLD
40	100530	30-NOV-98	9	33	1	44.99
36	4523	27-JAN-99	3	999	1	53.89
125	9417	04-FEB-98	3	999	1	16.86
35	2606	17-FEB-00	3	999	1	54.94

Description of "Figure 4-2 List Partitions"

Hash Partitioning

In **hash partitioning**, the database maps rows to partitions based on a hashing algorithm that the database applies to the user-specified partitioning key. The destination of a row is determined by the internal hash function applied to the row by the database. The hashing algorithm is designed to evenly distributes rows across devices so that each partition contains about the same number of rows.

Hash partitioning is useful for dividing large tables to increase manageability. Instead of one large table to manage, you have several smaller pieces. The loss of a single hash partition does not affect the remaining partitions and can be recovered independently. Hash partitioning is also useful in OLTP systems with high update contention. For example, a segment is divided into several pieces, each of which is updated, instead of a single segment that experiences contention.

Assume that you create the partitioned `hash_sales` table using the statement in Example 4-4. The `prod_id` column is the partition key.

Example 4-4 Hash-Partitioned Table

Table Partition EVEN_CHANNELS

PROD_ID	CUST_ID	TIME_ID	CHANNEL_ID	PROMO_ID	QUANTITY_SOLD	AMOUNT_SOLD
116	11393	05-JUN-99	2	999	1	12.18
118	133	06-JUN-01	2	999	1	17.12
133	9450	01-DEC-00	2	999	1	31.28
30	170	23-FEB-01	2	999	1	8.8
24	11899	26-JUN-99	4	999	1	43.04
45	9491	28-AUG-98	4	350	1	47.45

Table Partition ODD_CHANNELS

PROD_ID	CUST_ID	TIME_ID	CHANNEL_ID	PROMO_ID	QUANTITY_SOLD	AMOUNT_SOLD
40	100530	30-NOV-98	9	33	1	44.99
36	4523	27-JAN-99	3	999	1	53.89
125	9417	04-FEB-98	3	999	1	16.86
35	2606	17-FEB-00	3	999	1	54.94

Description of "Figure 4-2 List Partitions"

哈希分区

在**哈希分区**中，基于用户所指定的将在分区键上应用的哈希算法，数据库将行映射到各个分区。行的目标分区是由数据库应用于行的内部哈希函数所决定的。哈希算法被设计为可以跨设备均匀分布行，以使每个分区包含大致相同的行数。

哈希分区可用于划分大表，以提高可管理性。你只需管理几个较小的表片断，而不是管理一个大型表。一个哈希分区的缺失并不影响其余分区，并且可以独立地恢复。在更新争用较高的 OLTP 系统中，哈希分区也是非常有用的。例如，一个段被分为几个片断，每个片断都可以被更新，而不像单个段那样会遭受争用。

假定您使用示例 4-4 中的语句创建 hash_sales 分区表。prod_id 列为分区键。

*示例 **4-4** 哈希分区表*

```
CREATE TABLE hash_sales
   ( prod_id        NUMBER(6)
   , cust_id        NUMBER
   , time_id        DATE
   , channel_id     CHAR(1)
   , promo_id       NUMBER(6)
   , quantity_sold  NUMBER(3)
   , amount_sold    NUMBER(10,2)
   )
PARTITION BY HASH (prod_id)
PARTITIONS 2;
```

Afterward, you load the table with the rows from Example 4-1. Figure 4-3 shows a possible row distribution in the two partitions. Note that the names of these partitions are system-generated.

As you insert rows, the database attempts to randomly and evenly distribute them across partitions. You cannot specify the partition into which a row is placed. The database applies the hash function, whose outcome determines which partition contains the row. If you change the number of partitions, then the database redistributes the data over all of the partitions.

Figure 4-3 Hash Partitions

Table Partition SYS_P33

PROD_ID	CUST_ID	TIME_ID	CHANNEL_ID	PROMO_ID	QUANTITY_SOLD	AMOUNT_SOLD
40	100530	30-NOV-98	9	33	1	44.99
118	133	06-JUN-01	2	999	1	17.12
36	4523	27-JAN-99	3	999	1	53.89
30	170	23-FEB-01	2	999	1	8.8
35	2606	17-FEB-00	3	999	1	54.94

Table Partition SYS_P34

PROD_ID	CUST_ID	TIME_ID	CHANNEL_ID	PROMO_ID	QUANTITY_SOLD	AMOUNT_SOLD
116	11393	05-JUN-99	2	999	1	12.18
133	9450	01-DEC-00	2	999	1	31.28
125	9417	04-FEB-98	3	999	1	16.86
24	11899	26-JUN-99	4	999	1	43.04
45	9491	28-AUG-98	4	350	1	47.45

Description of "Figure 4-3 Hash Partitions"

```
CREATE TABLE hash_sales
   ( prod_id        NUMBER(6)
   , cust_id        NUMBER
   , time_id        DATE
   , channel_id     CHAR(1)
   , promo_id       NUMBER(6)
   , quantity_sold  NUMBER(3)
   , amount_sold    NUMBER(10,2)
   )
PARTITION BY HASH (prod_id)
PARTITIONS 2;
```

随后，将示例 4-1 中的表行加载到表中。图 4-3 显示两个分区中可能的行分布。请注意这些分区的名称是系统自动生成的。

当您插入行时，数据库会尝试随机、 均匀地将它们分布在各个分区之间。您不能指定某行被放置在哪一个分区。数据库应用哈希函数，其结果确定由哪个分区包含此行。如果重新更改了分区数目，那么数据库会在所有分区上重新分布数据。

图 4-3 哈希分区

Table Partition SYS_P33

PROD_ID	CUST_ID	TIME_ID	CHANNEL_ID	PROMO_ID	QUANTITY_SOLD	AMOUNT_SOLD
40	100530	30-NOV-98	9	33	1	44.99
118	133	06-JUN-01	2	999	1	17.12
36	4523	27-JAN-99	3	999	1	53.89
30	170	23-FEB-01	2	999	1	8.8
35	2606	17-FEB-00	3	999	1	54.94

Table Partition SYS_P34

PROD_ID	CUST_ID	TIME_ID	CHANNEL_ID	PROMO_ID	QUANTITY_SOLD	AMOUNT_SOLD
116	11393	05-JUN-99	2	999	1	12.18
133	9450	01-DEC-00	2	999	1	31.28
125	9417	04-FEB-98	3	999	1	16.86
24	11899	26-JUN-99	4	999	1	43.04
45	9491	28-AUG-98	4	350	1	47.45

Description of "Figure 4-3 Hash Partitions"

See Also:

- *Oracle Database VLDB and Partitioning Guide* to learn how to create partitions
- *Oracle Database SQL Language Reference* for `CREATE TABLE ... PARTITION BY` examples

Partitioned Tables

A **partitioned table** consists of one or more partitions, which are managed individually and can operate independently of the other partitions. A table is either partitioned or nonpartitioned. Even if a partitioned table consists of only one partition, this table is different from a nonpartitioned table, which cannot have partitions added to it. "Partition Characteristics" gives examples of partitioned tables.

A partitioned table is made up of one or more table partition segments. If you create a partitioned table named `hash_products`, then no table segment is allocated for this table. Instead, the database stores data for each table partition in its own partition segment. Each table partition segment contains a portion of the table data.

Some or all partitions of a heap-organized table can be stored in a compressed format. Compression saves space and can speed query execution. Thus, compression can be useful in environments such as data warehouses, where the amount of insert and update operations is small, and in OLTP environments.

The attributes for table compression can be declared for a tablespace, table, or table partition. If declared at the tablespace level, then tables created in the tablespace are compressed by default. You can alter the compression attribute for a table, in which case the change only applies to new data going into that table. Consequently, a single table or partition may contain compressed and uncompressed blocks, which guarantees that data size will not increase because of compression. If compression could increase the size of a block, then the database does not apply it to the block.

See Also:

- "Table Compression" and "Overview of Segments"
- *Oracle Database Data Warehousing Guide* to learn about table compression in a data warehouse

另见：

- 《*Oracle 数据库 VLDB 和分区指南*》了解如何创建分区
- 《*Oracle 数据库 SQL 语言参考*》关于 `CREATE TABLE ... PARTITION BY` 的示例

分区表

分区表包含一个或多个分区，它们可以单独进行管理，并且可以独立于其他分区进行操作。表要么是分区表，要么是未分区表。即使分区表只包含一个分区，此表也是与一个未分区表不同的，不能将分区加入到未分区表中。请参见"分区特征"中给出的一些分区表的示例。

分区表由一个或多个表分区段组成。如果您创建了一个名为 hash_products 的分区表，但并没有为此表分配表段。相反，数据库将每个表分区的数据存储在其自己的分区段中。每个表分区段包含表数据的一部分。

堆组织表中的某些分区或所有分区可以存储为压缩格式。压缩可以节省空间，并可以加快执行查询的速度。因此，压缩可用于在插入和更新操作很少的数据仓库环境中，也可以用于 OLTP 环境中。

可以为表空间、 表、或表分区声明表压缩属性。如果在表空间级别声明，则默认情况下在该表空间中创建的表是被压缩的。您可以更改一个表的压缩属性，在这种情况下，此更改仅适用于插入到该表的新数据。因此，一个单表或分区可能包含压缩和未压缩的块，它保证数据的大小不会因为压缩反而增大。如果压缩会增加一个数据块的大小，那么数据库就不压缩该块。

另见：

- "表压缩" 和 "段概述"
- 《*Oracle 数据库数据存储指南*》了解数据仓库中的表压缩

Partitioned Indexes

A **partitioned index** is an index that, like a partitioned table, has been

分区索引

与分区表类似，**分区索引**被分解成更小、 更易于管理的索引片断。**全局索**

decomposed into smaller and more manageable pieces. **Global indexes** are partitioned independently of the table on which they are created, whereas **local indexes** are automatically linked to the partitioning method for a table. Like partitioned tables, partitioned indexes improve manageability, availability, performance, and scalability.
The following graphic shows index partitioning options.

Description of the illustration cncpt301.gif

See Also:

- "Overview of Indexes"
- *Oracle Database VLDB and Partitioning Guide* and *Oracle Database Performance Tuning Guide* for more information about partitioned indexes and how to decide which type to use

Local Partitioned Indexes

In a **local partitioned index**, the index is partitioned on the same columns, with the same number of partitions and the same partition bounds as its table. Each index partition is associated with exactly one partition of the underlying table, so that all keys in an index partition refer only to rows stored in a single table partition. In this way, the database automatically synchronizes index partitions with their associated table partitions, making each table-index pair independent.
Local partitioned indexes are common in data warehousing environments. Local indexes offer the following advantages:

- Availability is increased because actions that make data invalid or unavailable in a partition affect this partition only.

- Partition maintenance is simplified. When moving a table partition,

引独立于它们依赖的表进行分区；而**局部索引**则依据不同的表分区方法，自动链接到相应的表分区。与分区表类似，分区索引提高了可管理性、 可用性、 性能、和可扩展性。

下图显示索引的分区选项。

Description of the illustration cncpt301.gif

另见：

- "索引概述"
- 《*Oracle 数据库 VLDB 和分区指南*》和《*Oracle 数据库性能优化指南*》关于已分区索引，以及如何确定要使用的类型的详细信息

局部分区索引

在**局部分区索引**中，索引基于表上相同的列来分区，与表分区具有相同分区数目和相同的分区边界。每个索引分区仅与底层表的一个分区相关联，所以一个索引分区中的所有键都只引用存储在某个单一表分区中的行。通过这种方式，数据库会自动同步索引分区及其关联的表分区，使每个表-索引对相持独立。

局部分区索引在数据仓库环境中很常见。局部索引提供了以下优点：

- 因为使分区中的数据无效或不可用的操作只会影响当前分区，这有助于提高可用性。

- 简化了分区维护。当移动一个表分区，或某个分区的数据老化时，只

or when data ages out of a partition, only the associated local index partition must be rebuilt or maintained. In a global index, all index partitions must be rebuilt or maintained.

- If **point-in-time recovery** of a partition occurs, then the indexes can be recovered to the recovery time (see "Data File Recovery"). The entire index does not need to be rebuilt.

Example 4-4 shows the creation statement for the partitioned `hash_sales` table, using the `prod_id` column as partition key. Example 4-5 creates a local partitioned index on the `time_id` column of the `hash_sales` table.

Example 4-5 Local Partitioned Index

```
CREATE INDEX hash_sales_idx ON hash_sales(time_id) LOCAL;
```

In Figure 4-4, the `hash_products` table has two partitions, so `hash_sales_idx` has two partitions. Each index partition is associated with a different table partition. Index partition `SYS_P38` indexes rows in table partition `SYS_P33`, whereas index partition `SYS_P39` indexes rows in table partition `SYS_P34`.

Figure 4-4 Local Index Partitions

须重建或维持相关联的局部索引分区。而在全局索引中所有索引分区必须被全部重建或维护。

- 如果分区发生**时间点恢复**，则可以将局部索引恢复到恢复时间 （请参阅"数据文件恢复"）。而不需要重建整个索引。

示例 4-4 显示了使用 prod_id 列作为分区键的 hash_sales 分区表的创建语句。示例 4-5 在 hash_sales 表的 time_id 列上创建一个局部分区索引。

示例 4-5 局部分区索引

```
CREATE INDEX hash_sales_idx ON hash_sales(time_id) LOCAL;
```

在图 4－4 中， hash_products 表中有两个分区，因此 hash_sales_idx 也有两个分区。每个索引分区与一个不同的表分区相关联。索引分区 SYS_P38 对表分区 SYS_P33 中的行建立索引，而索引分区 SYS_P39 对表分区 SYS_P34 中的行建立索引。

图 4-4 局部索引分区

Description of "Figure 4-4 Local Index Partitions"

You cannot explicitly add a partition to a local index. Instead, new partitions are added to local indexes only when you add a partition to the underlying table. Likewise, you cannot explicitly drop a partition from a local index. Instead, local index partitions are dropped only when you drop a partition from the underlying table.

Like other indexes, you can create a bitmap index on partitioned tables. The only restriction is that bitmap indexes must be local to the partitioned table—they cannot be global indexes. Global bitmap indexes are supported only on nonpartitioned tables.

Local Prefixed and Nonprefixed Indexes

Local partitioned indexes are divided into the following subcategories:

Description of "Figure 4-4 Local Index Partitions"

你不能明确将一个分区添加到一个局部索引中。相反，仅当您将一个表分区添加到基础表时，新的索引分区会被自动添加到局部索引。同样，不能明确从局部索引删除一个分区。相反，仅当你从基础表中删除一个表分区时，相应的局部索引分区会被自动删除。

与其他索引一样，您可以在分区表上创建位图索引。唯一的限制是位图索引必须是局部分区索引 — — 而不能是全局分区索引。全局位图索引只支持非分区表。

局部前缀索引和局部非前缀索引

局部分区索引可分为以下类别：

- Local prefixed indexes

  In this case, the partition keys are on the leading edge of the index definition. In Example 4-2, the table is partitioned by range on `time_id`. A local prefixed index on this table would have `time_id` as the first column in its list.
- Local nonprefixed indexes

  In this case, the partition keys are not on the leading edge of the indexed column list and need not be in the list at all. In Example 4-5, the index is local nonprefixed because the partition key `product_id` is not on the leading edge.

Both types of indexes can take advantage of **partition elimination** (also called **partition pruning**), which occurs when the optimizer speeds data access by excluding partitions from consideration. Whether a query can eliminate partitions depends on the query predicate. A query that uses a local prefixed index always allows for index partition elimination, whereas a query that uses a local nonprefixed index might not.

See Also:

Oracle Database VLDB and Partitioning Guide to learn how to use prefixed and nonprefixed indexes

Local Partitioned Index Storage

Like a table partition, a local index partition is stored in its own segment. Each segment contains a portion of the total index data. Thus, a local index made up of four partitions is not stored in a single index segment, but in four separate segments.

See Also:

Oracle Database SQL Language Reference for `CREATE INDEX ... LOCAL` examples

Global Partitioned Indexes

A **global partitioned index** is a B-tree index that is partitioned independently of the underlying table on which it is created. A single index partition can point to any or all table partitions, whereas in a locally partitioned index, a one-to-one parity exists between index partitions and table partitions.

In general, global indexes are useful for OLTP applications, where rapid access, data integrity, and availability are important. In an OLTP system, a table may be partitioned by one key, for example, the

- 局部前缀索引

  在这种情况下，分区键处于索引定义的前导部分。示例 4-2 中， 该表在 time_id 上按范围分区。此表上的局部前缀索引会以 time_id 作为其索引列列表中的第一列。

- 局部非前缀索引

  在这种情况下，分区键不是索引列列表的前导部分，甚至根本不必在该列表中。在示例 4-5 中，由于分区键 product_id 不属于前导列，所以该索引是局部非前缀索引。

这两种类型的索引都可以充分利用**分区消除** （也称为**分区剪除**），此时，优化程序将不予考虑无关分区，以加快数据访问速度。查询是否可以消除分区取决于查询谓词。使用局部前缀索引的查询始终允许索引分区消除，而使用一个局部非前缀索引的查询可能不会。

另见：

《*Oracle 数据库 VLDB 和分区指南*》了解如何使用前缀和非前缀的索引

局部分区索引存储

与表分区类似，局部索引分区被存储在其自己的段中。每个段包含整个索引数据的一部分。因此，由四个分区组成的局部索引，不是存储在一个单一索引段中，而是在四个单独的段中。

另见：

《*Oracle 数据库 SQL 语言参考*》关于 `CREATE INDEX ... LOCAL` 的示例

全局分区索引

全局分区索引是一个 B-树索引，其分区独立于所依赖的基础表。某个索引分区可以指向任意或所有的表分区，而在一个局部分区索引中，索引分区与分区表之间却存在一对一的配对关系。

通常，对于强调快速访问、 数据完整性、和可用性的 OLTP 应用程序来说，全局索引很有用。在一个 OLTP 系统中，表可能会基于某个键（如 employees.department_id 列）来分区，但应用程序可能需要基于许多不同

employees.department_id column, but an application may need to access the data with many different keys, for example, by employee_id or job_id. Global indexes can be useful in this scenario.

You can partition a global index by range or by hash. If partitioned by range, then the database partitions the global index on the ranges of values from the table columns you specify in the column list. If partitioned by hash, then the database assigns rows to the partitions using a hash function on values in the partitioning key columns.

As an illustration, suppose that you create a global partitioned index on the time_range_sales table from Example 4-2. In this table, rows for sales from 1998 are stored in one partition, rows for sales from 1999 are in another, and so on. Example 4-6 creates a global index partitioned by range on the channel_id column.

Example 4-6 Global Partitioned Index

```
CREATE INDEX time_channel_sales_idx ON
time_range_sales (channel_id)
   GLOBAL PARTITION BY RANGE (channel_id)
      (PARTITION p1 VALUES LESS THAN (3),
       PARTITION p2 VALUES LESS THAN (4),
       PARTITION p3 VALUES LESS THAN (MAXVALUE));
```

As shown in Figure 4-5, a global index partition can contain entries that point to multiple table partitions. Index partition p1 points to the rows with a channel_id of 2, index partition p2 points to the rows with a channel_id of 3, and index partition p3 points to the rows with a channel_id of 4 or 9.

Figure 4-5 Global Partitioned Index

的键（如 employee_id 或 job_id）来访问数据。全局索引在这种情况下可能很有用。

你可以按范围或哈希方式建立全局分区索引。如果按范围分区，则数据库按你指定的表列的值范围对全局索引进行分区。如果按哈希分区，则数据库对分区键列中的值使用哈希函数，并将行分配到相应的分区。

作为演示，假设您在示例 4-2 中的 time_range_sales 表上创建全局分区索引。在该表中， 1998 年销售的行存储在一个分区中，1999 年销售的行在另一个分区中，如此等等。示例 4-6 在 channel_id 列上按范围创建了一个全局分区索引。

示例 4－6 全局分区索引

```
CREATE INDEX time_channel_sales_idx ON
time_range_sales (channel_id)
   GLOBAL PARTITION BY RANGE (channel_id)
      (PARTITION p1 VALUES LESS THAN (3),
       PARTITION p2 VALUES LESS THAN (4),
       PARTITION p3 VALUES LESS THAN (MAXVALUE));
```

如图 4-5 ，全局索引分区可以包含指向多个表分区的条目。索引分区 p1 指向 channel_id 为 2 的行，索引分区 p2 指向 channel_id 为 3 的行，而索引分区 p3 指向 channel_id 为 4 或 9 的行。

图 4-5 全局分区索引

Description of "Figure 4-5 Global Partitioned Index"

See Also:

- *Oracle Database VLDB and Partitioning Guide* to learn how to use global partitioned indexes
- *Oracle Database SQL Language Reference* for `CREATE INDEX ... GLOBAL` examples

Description of "Figure 4-5 Global Partitioned Index"

另见：

- 《*Oracle 数据库 VLDB 和分区指南*》了解如何使用全局分区索引
- 《*Oracle 数据库 SQL 语言参考*》关于 `CREATE INDEX ...`

Partitioned Index-Organized Tables

You can partition an index-organized table (IOT) by range, list, or hash. Partitioning is useful for providing improved manageability, availability, and performance for IOTs. In addition, data cartridges that use IOTs can take advantage of the ability to partition their stored data.

Note the following characteristics of partitioned IOTs:

- Partition columns must be a subset of primary key columns.
- Secondary indexes can be partitioned locally and globally.
- `OVERFLOW` data segments are always equipartitioned with the table partitions.

Oracle Database supports bitmap indexes on partitioned and nonpartitioned index-organized tables. A mapping table is required for creating bitmap indexes on an index-organized table.

See Also:

"Overview of Index-Organized Tables"

Overview of Views

A view is a logical representation of one or more tables. In essence, a view is a stored query. A view derives its data from the tables on which it is based, called **base tables**. Base tables can be tables or other views. All operations performed on a view actually affect the base tables. You can use views in most places where tables are used.

Note:

Materialized views use a different data structure from standard views. See "Overview of Materialized Views".

Views enable you to tailor the presentation of data to different types of users. Views are often used to:

- Provide an additional level of table security by restricting access to a predetermined set of rows or columns of a table

  For example, Figure 4-6 shows how the `staff` view does not show the `salary` or `commission_pct` columns of the base table `employees`.
- Hide data complexity

  For example, a single view can be defined with a join, which is a

`GLOBAL` 的示例

分区索引组织表

您可以对一个索引组织表 （IOT) 按范围、 列表、或哈希值进行分区。分区有助于改进 IOT 的可管理性、 可用性、和性能。另外，使用 IOT 的数据模块可以利用此功能，对其存储的数据进行分区。

请注意分区 IOT 的以下特征：

- 分区列必须是主键列的子集。
- 可以对辅助索引进行局部或全局分区。
- 溢出数据段总是和表分区一样具有相应的对等分区。

Oracle 数据库支持在分区和未分区索引组织表上的位图索引。在索引组织表上创建位图索引需要一个映射表。

另见：

"分区索引组织表概述"

视图概述

视图是一个或多个表的逻辑表示形式。视图在本质上是一个存储的查询。视图的数据来自它所依赖的称为**基表**的表。基表可以是表或其他视图。在视图上执行的所有操作实际上都影响到基表。在大多数可以使用表的地方，您也可以使用视图。

注意：

物化视图使用了与标准视图不同的数据结构。请参阅"物化视图概述"。

视图使您能够为不同类型的用户定制数据表示形式。视图通常用于：

- 通过限制对一组预定义的表行或表列的访问，提供一个额外的表安全级别

  例如，图 4-6 显示了 `staff` 视图如何做到不显示 `employees` 基表的 `salary` 列或 `commission_pct` 列。
- 隐藏数据复杂性

  例如，一个单一视图的定义可以包含一个联接，它是多个表中的相关

collection of related columns or rows in multiple tables. However, the view hides the fact that this information actually originates from several tables. A query might also perform extensive calculations with table information. Thus, users can query a view without knowing how to perform a join or calculations.

- Present the data in a different perspective from that of the base table

  For example, the columns of a view can be renamed without affecting the tables on which the view is based.
- Isolate applications from changes in definitions of base tables

  For example, if the defining query of a view references three columns of a four column table, and a fifth column is added to the table, then the definition of the view is not affected, and all applications using the view are not affected.

For an example of the use of views, consider the `hr.employees` table, which has several columns and numerous rows. To allow users to see only five of these columns or only specific rows, you could create a view as follows:

```
CREATE VIEW staff AS
  SELECT employee_id, last_name, job_id,
manager_id, department_id
  FROM   employees;
```

As with all subqueries, the query that defines a view cannot contain the `FOR UPDATE` clause. Figure 4-6 graphically illustrates the view named `staff`. Notice that the view shows only five of the columns in the base table.

Figure 4-6 View

列或行的集合。不过，视图隐藏了一个事实，即此信息实际上来自于几个表。查询还可能会对表的信息执行大量计算。因此，用户可以查询视图，而不用知道如何执行联接或计算。

- 以一个不同于基表的角度来呈现数据

  例如，可以重命名视图的列，而不会影响视图所基于的表。

- 隔离应用程序和对基表定义的更改

  例如，如果视图的查询定义引用了一个四列表的其中三列，若添加第五列到该表，则视图的定义不会受到影响，并且使用该视图的所有应用程序也不会受到影响。

举一个使用视图的例子，考虑表 hr.employees，它具有多个列和许多行。要允许用户只能查看其中 5 列，或只能查看某些特定的行，您可以创建一个视图，如下所示：

```
CREATE VIEW staff AS
  SELECT employee_id, last_name, job_id,
manager_id, department_id
  FROM   employees;
```

与所有子查询一样，定义视图的查询中不能包含 `FOR UPDATE` 子句。图 4-6 以图形方式显示了 `staff` 视图。请注意，该视图只显示了基表中的五列。

图 4-6 视图

Description of "Figure 4-6 View"

See Also:

- *Oracle Database Administrator's Guide* to learn how to manage views
- *Oracle Database SQL Language Reference* for `CREATE VIEW` syntax and semantics

Characteristics of Views

Unlike a table, a view is not allocated storage space, nor does a view contain data. Rather, a view is defined by a query that extracts or derives data from the base tables referenced by the view. Because a view is based on other objects, it requires no storage other than storage for the query that defines the view in the data dictionary.

A view has dependencies on its referenced objects, which are automatically handled by the database. For example, if you drop and re-create a base table of a view, then the database determines whether the new base table is acceptable to the view definition.

Data Manipulation in Views

Because views are derived from tables, they have many similarities. For example, a view can contain up to 1000 columns, just like a table. Users can query views, and with some restrictions they can perform DML on views. Operations performed on a view affect data in some base table of the view and are subject to the integrity constraints and triggers of the base tables.

The following example creates a view of the `hr.employees` table:

Description of "Figure 4-6 View"

另见：

- 《*Oracle 数据库管理员指南*》了解如何管理视图
- 《*Oracle 数据库 SQL 语言参考*》有关 `CREATE VIEW` 的语法和语义

视图特征

与表不同，视图既不分配存储空间，也不包含数据。而是通过定义一个查询，从它所引用的基表中提取或派生出数据。视图基于其他对象，除了只需要在数据字典中存储定义视图的查询，它不需要其他存储。

视图对其所引用的对象存在的依赖关系，由数据库自动处理。例如，如果您删除并重新创建了视图的基表，则数据库会确定新的基表是否仍然适合该视图的定义。

视图中的数据操作

由于视图是从表派生的，所以他们有很多相似之处。例如，与表一样，一个视图可以包含多达 1000 个列。用户可以查询视图，在某些限制条件下他们也可以在视图上执行 DML。在视图上执行的操作会影响到视图的基表中的数据，并遵从基表的完整性约束和触发器。

下面的示例创建 hr.employees 表上的一个视图：

```
CREATE VIEW staff_dept_10 AS
SELECT employee_id, last_name, job_id,
       manager_id, department_id
FROM   employees
WHERE  department_id = 10
WITH CHECK OPTION CONSTRAINT staff_dept_10_cnst;
```

The defining query references only rows for department 10. The `CHECK OPTION` creates the view with a constraint so that `INSERT` and `UPDATE` statements issued against the view cannot result in rows that the view cannot select. Thus, rows for employees in department 10 can be inserted, but not rows for department 30.

See Also:

Oracle Database SQL Language Reference to learn about subquery restrictions in `CREATE VIEW` statements

How Data Is Accessed in Views

Oracle Database stores a view definition in the data dictionary as the text of the query that defines the view. When you reference a view in a SQL statement, Oracle Database performs the following tasks:

- 1. Merges a query (whenever possible) against a view with the queries that define the view and any underlying views

  Oracle Database optimizes the merged query as if you issued the query without referencing the views. Therefore, Oracle Database can use indexes on any referenced base table columns, whether the columns are referenced in the view definition or in the user query against the view.
  Sometimes Oracle Database cannot merge the view definition with the user query. In such cases, Oracle Database may not use all indexes on referenced columns.
- 2. Parses the merged statement in a shared SQL area

  Oracle Database parses a statement that references a view in a new shared SQL area *only* if no existing shared SQL area contains a similar statement. Thus, views provide the benefit of reduced memory use associated with shared SQL.
- 3. Executes the SQL statement

The following example illustrates data access when a view is queried.

```
CREATE VIEW staff_dept_10 AS
SELECT employee_id, last_name, job_id,
       manager_id, department_id
FROM   employees
WHERE  department_id = 10
WITH CHECK OPTION CONSTRAINT staff_dept_10_cnst;
```

该查询定义只引用了部门 10 中的行。使用 `CHECK OPTION` 创建的视图具有一个约束，如果对该视图发出的 INSERT 和 UPDATE 语句所生成的新行不符合该视图的选择条件，则该语句将不会成功。因此，行 10 部门的雇员可以插入，但是部门 30 的行不可以。

另见：

《*Oracle 数据库 SQL 语言参考*》了解 `CREATE VIEW` 语句中的子查询限制

在视图中，数据是如何访问的

Oracle 数据库将视图定义存储在数据字典中，存的是定义视图查询的文本。当在 SQL 语句中引用视图时，数据库将执行以下任务：

- 1. （尽可能）将所发出的查询，与定义该视图（或其它任何底层视图）的查询进行合并

  Oracle 数据库会优化合并的查询，就好像您发出的查询并没有引用视图一样。因此，数据库可以使用任何被引用基表列上的索引，无论在视图定义中或在针对该视图的用户查询中是否引用了该列。

  有时数据库不能将视图定义与用户查询合并。在这种情况下，数据库可能不会使用所有引用列上的索引。
- 2. 在一个共享 SQL 区中，解析合并后的语句

  *只要*还不存在包含相似语句的共享 SQL 区，oracle 数据库就会在一个共享 SQL 区中解析这个引用了视图的语句。因此，视图提供了减少与共享 SQL 相关的内存使用的好处。
- 3. 执行 SQL 语句

下面的示例演示了当查询视图时的数据访问。假设您创建基于 `employees`

Assume that you create employees_view based on the employees and departments tables:

```
CREATE VIEW employees_view AS
  SELECT employee_id, last_name, salary,
location_id
  FROM   employees JOIN departments USING
(department_id)
  WHERE  departments.department_id = 10;
```

A user executes the following query of employees_view:

```
SELECT last_name
FROM   employees_view
WHERE  employee_id = 9876;
```

Oracle Database merges the view and the user query to construct the following query, which it then executes to retrieve the data:

```
SELECT last_name
FROM   employees, departments
WHERE  employees.department_id =
departments.department_id
AND    departments.department_id = 10
AND    employees.employee_id = 9876;
```

See Also:

- "Overview of the Optimizer" and *Oracle Database Performance Tuning Guide* to learn about query optimization
- "Shared SQL Areas"

Updatable Join Views

A **join view** is defined as a view that has multiple tables or views in its FROM clause. In Example 4-7, the staff_dept_10_30 view joins the employees and departments tables, including only employees in departments 10 or 30.

Example 4-7 Join View

```
CREATE VIEW staff_dept_10_30 AS
SELECT employee_id, last_name, job_id,
e.department_id
FROM   employees e, departments d
WHERE  e.department_id IN (10, 30)
AND    e.department_id = d.department_id;
```

表和 departments 表 employees_view 视图：

```
CREATE VIEW employees_view AS
  SELECT employee_id, last_name, salary,
location_id
  FROM   employees JOIN departments USING
(department_id)
  WHERE  departments.department_id = 10;
```

用户对 employees_view 执行如下的查询：

```
SELECT last_name
FROM   employees_view
WHERE  employee_id = 9876;
```

Oracle 数据库将视图和用户查询合并，构建如下面的查询，然后执行此查询以检索数据：

```
SELECT last_name
FROM   employees, departments
WHERE  employees.department_id =
departments.department_id
AND    departments.department_id = 10
AND    employees.employee_id = 9876;
```

另见：

- "优化器概述"和《*Oracle 数据库性能优化指南*》了解查询优化
- "共享 SQL 区"

可更新联接视图

联接视图被定义为在其 FROM 子句中有多个表或视图的视图。在例 4-7 中 staff_dept_10_30 视图将 employees 表和 departments 表相连接，只包括部门 10 或 30 的雇员。

示例 4-7 连接视图

```
CREATE VIEW staff_dept_10_30 AS
SELECT employee_id, last_name, job_id,
e.department_id
FROM   employees e, departments d
WHERE  e.department_id IN (10, 30)
AND    e.department_id = d.department_id;
```

An **updatable join view**, also called a **modifiable join view**, involves two or more base tables or views and permits DML operations. An updatable view contains multiple tables in the top-level FROM clause of the SELECT statement and is not restricted by the WITH READ ONLY clause.

To be inherently updatable, a view must meet several criteria. For example, a general rule is that an INSERT, UPDATE, or DELETE operation on a join view can modify only one base table at a time. The following query of the USER_UPDATABLE_COLUMNS data dictionary view shows that the view created in Example 4-7 is updatable:

```
SQL> SELECT TABLE_NAME, COLUMN_NAME, UPDATABLE
  2  FROM   USER_UPDATABLE_COLUMNS
  3  WHERE  TABLE_NAME = 'STAFF_DEPT_10_30';

TABLE_NAME                     COLUMN_NAME                    UPD
------------------------------ ------------------------------ ---
STAFF_DEPT_10_30               EMPLOYEE_ID                    YES
STAFF_DEPT_10_30               LAST_NAME                      YES
STAFF_DEPT_10_30               JOB_ID                         YES
STAFF_DEPT_10_30               DEPARTMENT_ID                  YES
```

All updatable columns of a join view must map to columns of a key-preserved table. A **key-preserved table** in a join query is a table in which each row of the underlying table appears at most one time in the output of the query. In Example 4-7, department_id is the primary key of the departments table, so each row from the employees table appears at most once in the result set, making the employees table key-preserved. The departments table is not key-preserved because each of its rows may appear many times in the result set.

See Also:

Oracle Database Administrator's Guide to learn how to update join views

Object Views

Just as a view is a virtual table, an **object view** is a virtual object table. Each row in the view is an **object**, which is an instance of an object type. An object type is a user-defined data type.

You can retrieve, update, insert, and delete relational data as if it was stored as an object type. You can also define views with columns that are object data types, such as objects, REFs, and collections (nested tables and VARRAYs).

可更新联接视图，也叫做**可修改联接视图**，涉及两个或多个基表或视图，并允许 DML 操作。可更新视图在 SELECT 语句的顶层 FROM 子句中包含多个表，并且没有 WITH READ ONLY 子句限制。

要使视图天生就可更新，必须满足几个条件。例如，一般的原则是联接视图上的插入、 更新、或删除操作只可一次修改一个基表。如下的对数据字典视图 USER_UPDATABLE_COLUMNS 的查询显示在示例 4-7 中创建的视图是可更新的：

```
SQL> SELECT TABLE_NAME, COLUMN_NAME, UPDATABLE
  2  FROM   USER_UPDATABLE_COLUMNS
  3  WHERE  TABLE_NAME = 'STAFF_DEPT_10_30';

TABLE_NAME                     COLUMN_NAME                    UPD
------------------------------ ------------------------------ ---
STAFF_DEPT_10_30               EMPLOYEE_ID                    YES
STAFF_DEPT_10_30               LAST_NAME                      YES
STAFF_DEPT_10_30               JOB_ID                         YES
STAFF_DEPT_10_30               DEPARTMENT_ID                  YES
```

连接视图中的所有可更新列，必须映射到键保留表中的列。联接查询中的**键保留表**，其基表中的每一行在查询输出中最多出现一次。在示例 4-7 中，department_id 是 departments 表的主键，所以 employees 表中的每一行在结果集中最多出现一次，使得 employees 表是键保留的。而 departments 表不是键保留的，因为其每个行可能会在结果集中多次出现。

另见：

《*Oracle 数据库管理员指南*》了解如何更新连接视图

对象视图

正如一个视图是一个虚拟表，一个**对象视图**是一个虚拟对象表。在视图中的每一行都是一个**对象**，对象是某种对象类型的一个实例。对象类型是一种用户定义的数据类型。

您可以检索、 更新、 插入、和删除关系数据，如同它被存储为一个对象类型。您还可以用对象数据类型的列来定义视图，如对象、 REF、和集合（嵌套表和 VARRAYs）等。

Like relational views, object views can present only the data that you want users to see. For example, an object view could present data about IT programmers but omit sensitive data about salaries. The following example creates an `employee_type` object and then the view `it_prog_view` based on this object:

```
CREATE TYPE employee_type AS OBJECT
(
  employee_id  NUMBER (6),
  last_name    VARCHAR2 (25),
  job_id       VARCHAR2 (10)
);
/
CREATE VIEW it_prog_view OF employee_type
  WITH OBJECT IDENTIFIER (employee_id) AS
SELECT  e.employee_id, e.last_name, e.job_id
FROM    employees e
WHERE   job_id = 'IT_PROG';
```

Object views are useful in prototyping or transitioning to object-oriented applications because the data in the view can be taken from relational tables and accessed as if the table were defined as an object table. You can run object-oriented applications without converting existing tables to a different physical structure.

See Also:

- *Oracle Database Object-Relational Developer's Guide* to learn about object types and object views
- *Oracle Database SQL Language Reference* to learn about the `CREATE TYPE` command

Overview of Materialized Views

Materialized views are query results that have been stored or "materialized" in advance as schema objects. The `FROM` clause of the query can name tables, views, and materialized views. Collectively these objects are called **master tables** (a replication term) or **detail tables** (a data warehousing term).

Materialized views are used to summarize, compute, replicate, and distribute data. They are suitable in various computing environments, such as the following:

与关系型视图类似，对象视图可以只显示您希望用户看到的数据。例如，一个对象视图可以只呈现 IT 程序员的有关信息，但忽略有关薪水等敏感数据。下面的示例创建一个 employee_type 对象，然后创建基于此对象的视图 it_prog_view：

```
CREATE TYPE employee_type AS OBJECT
(
  employee_id  NUMBER (6),
  last_name    VARCHAR2 (25),
  job_id       VARCHAR2 (10)
);
/
CREATE VIEW it_prog_view OF employee_type
  WITH OBJECT IDENTIFIER (employee_id) AS
SELECT  e.employee_id, e.last_name, e.job_id
FROM    employees e
WHERE   job_id = 'IT_PROG';
```

对象视图在原型程序或向面向对象转换的应用程序中非常有用，因为视图中的数据可以取自关系表，而访问起来好像是一个对象表。您可以运行面向对象的应用程序，而不用将现有表转换为不同的物理结构。

另见：

- 《*Oracle 数据库对象关系开发者指南*》了解对象类型和对象视图
- 《*Oracle 数据库 SQL 语言参考*》了解 `CREATE TYPE` 命令

物化视图概述

物化视图是查询结果被提前存储或"物化"的模式对象。在查询的 FROM 子句中可以是命名表、 视图、和物化视图。总体上，这些对象被称作**主表**（复制术语） 或**明细表** （数据仓库术语）。

物化视图用于汇总、 计算、 复制、和分发数据。它们适用于如下各种计算环境：

- In data warehouses, you can use materialized views to compute and store data generated from aggregate functions such as sums and averages.

  A **summary** is an aggregate view that reduces query time by precalculating joins and aggregation operations and storing the results in a table. Materialized views are equivalent to summaries (see "Data Warehouse Architecture (Basic)"). You can also use materialized views to compute joins with or without aggregations. If compatibility is set to Oracle9*i* or higher, then materialized views are usable for queries that include filter selections.
- In materialized view replication, the view contains a complete or partial copy of a table from a single point in time. Materialized views replicate data at distributed sites and synchronize updates performed at several sites. This form of replication is suitable for environments such as field sales when databases are not always connected to the network.
- In mobile computing environments, you can use materialized views to download a data subset from central servers to mobile clients, with periodic refreshes from the central servers and propagation of updates by clients to the central servers.

In a replication environment, a materialized view shares data with a table in a different database, called a **master database**. The table associated with the materialized view at the master site is the **master table**. Figure 4-7 illustrates a materialized view in one database based on a master table in another database. Updates to the master table replicate to the materialized view database.

Figure 4-7 Materialized View

- 在数据仓库中，可以使用物化视图来计算和存储由聚合函数（如求和或平均值）所生成的数据。

  汇总是一个聚合的视图，它通过预先计算联接和聚合操作，并将结果存储在一个表中，来减少查询时间。物化视图相当于汇总 （请参阅"数据仓库体系结构 （基本）"）。您也可以使用物化视图来计算带或不带聚合的联接。如果兼容性设置为 Oracle9i 或更高，则物化视图可用于包括筛选器选择的查询。

- 在物化视图复制中，该视图包含从某个表的单一时间点的完整或部分拷贝。物化视图在分布式站点上复制数据，并将在多个站点上执行的更新同步。这种形式的复制适用于如数据库并不始终连接到网络的现场销售环境。

- 在移动的计算环境中，可以使用物化视图将数据子集从中央服务器下载到移动客户端，从中央服务器定期刷新客户端，并定期将客户端更新传输回中央服务器。

在复制环境中，物化视图与一个称为**主数据库**的不同数据库中的表共享数据。在主站点上与物化视图关联的表是**主表**。图 4-7 演示了一个数据库中的物化视图，它基于另一个数据库中的主表。对主表的更新被复制到物化视图。

图 4-7 物化视图

Description of "Figure 4-7 Materialized View"

See Also:

- "Information Sharing" to learn about replication with Oracle Streams
- *Oracle Database 2 Day + Data Replication and Integration Guide* and *Oracle Database Advanced Replication* to learn how to use materialized views
- *Oracle Database SQL Language Reference* to learn about the `CREATE MATERIALIZED VIEW` statement

Characteristics of Materialized Views

Materialized views share some characteristics of nonmaterialized views and indexes. Materialized views are similar to indexes in the following ways:

- They contain actual data and consume storage space.
- They can be refreshed when the data in their master tables changes.
- They can improve performance of SQL execution when used for query rewrite operations.
- Their existence is transparent to SQL applications and users.

Description of "Figure 4-7 Materialized View"

另见：

- "信息共享"了解使用 Oracle 流复制
- 《*Oracle 数据库 2 日 + 数据复制和集成指南*》和 《*Oracle 数据库高级复制*》了解如何使用物化视图
- 《*Oracle 数据库 SQL 语言参考*》了解 `CREATE MATERIALIZED VIEW` 语句

物化视图的特征

物化视图与普通视图和索引具有某些共同特性。物化视图在以下几方面与索引类似：

- 它们包含实际数据，并且占用存储空间。
- 当其主表中数据更改时，他们可以被刷新。
- 当用于查询重写操作，它们可以提高 SQL 的执行性能。
- 它们的存在对 SQL 应用程序和用户是透明的。

A materialized view is similar to a nonmaterialized view because it represents data in other tables and views. Unlike indexes, users can query materialized views directly using `SELECT` statements. Depending on the types of refresh that are required, the views can also be updated with DML statements.

The following example creates and populates a materialized aggregate view based on three master tables in the `sh` sample schema:

```
CREATE MATERIALIZED VIEW sales_mv AS
  SELECT t.calendar_year, p.prod_id,
SUM(s.amount_sold) AS sum_sales
  FROM   times t, products p, sales s
  WHERE  t.time_id = s.time_id
  AND    p.prod_id = s.prod_id
  GROUP BY t.calendar_year, p.prod_id;
```

The following example drops table `sales`, which is a master table for `sales_mv`, and then queries `sales_mv`. The query selects data because the rows are stored (materialized) separately from the data in the master tables.

```
SQL> DROP TABLE sales;
Table dropped.
SQL> SELECT * FROM sales_mv WHERE ROWNUM < 4;

CALENDAR_YEAR    PROD_ID  SUM_SALES
------------- ---------- ----------
         1998         13  936197.53
         1998         26  567533.83
         1998         27  107968.24
```

A materialized view can be partitioned. You can define a materialized view on a partitioned table and one or more indexes on the materialized view.

See Also:

Oracle Database Data Warehousing Guide to learn how to use materialized views in a data warehouse

Refresh Methods for Materialized Views

The database maintains data in materialized views by refreshing them after changes to their master tables. The refresh method can be incremental, known as **fast refresh**, or a **complete refresh**.

A complete refresh occurs when the materialized view is initially defined as `BUILD IMMEDIATE`, unless the materialized view references a prebuilt

物化视图也类似于非物化视图，因为它也呈现其他的表和视图中的数据。与索引不同的是，用户可以直接使用 SELECT 语句查询物化视图。取决于所需的刷新类型，物化视图也可以用 DML 语句进行更新。

下面的示例创建并填充基于 sh 示例模式中的三个主表的聚合物化视图：

```
CREATE MATERIALIZED VIEW sales_mv AS
  SELECT t.calendar_year, p.prod_id,
SUM(s.amount_sold) AS sum_sales
  FROM   times t, products p, sales s
  WHERE  t.time_id = s.time_id
  AND    p.prod_id = s.prod_id
  GROUP BY t.calendar_year, p.prod_id;
```

下面的示例删除物化视图 sales_mv 的主表 `sales`，然后查询 `sales_mv`。该查询能选出数据，是因为其行数据与主表中的数据是分开存储的。

```
SQL> DROP TABLE sales;
Table dropped.
SQL> SELECT * FROM sales_mv WHERE ROWNUM < 4;

CALENDAR_YEAR    PROD_ID  SUM_SALES
------------- ---------- ----------
         1998         13  936197.53
         1998         26  567533.83
         1998         27  107968.24
```

物化视图可以进行分区。您可以在分区表上定义一个物化视图，并可以在该物化视图上创建一个或多个索引。

另见：

《*Oracle 数据库数据仓库指南*》了解如何在数据仓库中使用物化视图

物化视图的刷新方法

在主表中的数据更改后，数据库通过刷新物化视图来对其进行维护。刷新方法可以是增量刷新（也称**快速刷新**），或**完全刷新**。

当物化视图最初被定义为 `BUILD IMMEDIATE` 时，会发生完全刷新，除非物化视图引用了一个预建表。刷新涉及执行物化视图定义的查询。此过程可

table. The refresh involves executing the query that defines the materialized view. This process can be slow, especially if the database must read and process huge amounts of data.

能很慢，尤其是当数据库必须读取和处理大量的数据时。

A fast refresh eliminates the need to rebuild materialized views from scratch. Thus, processing only the changes can result in a very fast refresh time. Materialized views can be refreshed either on demand or at regular time intervals. Alternatively, materialized views in the same database as their master tables can be refreshed whenever a transaction commits its changes to the master tables.

快速刷新消除了从零开始重新生成物化视图的需要。因此，它仅处理更改，这样可能刷新速度会非常快。可以按需或按定期的时间间隔刷新物化视图。另外，与其主表在同一数据库中的物化视图，只要当主表的事务提交其所做的更改，就可以刷新物化视图。

For materialized views that use the fast refresh method, a **materialized view log** or **direct loader log** keeps a record of changes to the master tables. A materialized view log is a schema object that records changes to master table data so that a materialized view defined on the master table can be refreshed incrementally. Each materialized view log is associated with a single master table. The materialized view log resides in the same database and schema as its master table.

对于使用快速刷新方法的物化视图，由**物化视图日志**或**直接加载器日志**来记录对主表的更改。物化视图日志是一个用于记录主表数据更改的模式对象，以便可以以增量方式刷新物化视图。每个物化视图日志都与单个主表相关联。物化视图日志与其主表驻留在同一个数据库和模式内。

See Also:

- *Oracle Database Data Warehousing Guide* to learn how to refresh materialized views
- *Oracle Database Advanced Replication* to learn about materialized view logs

另见：

- 《*Oracle 数据库数据仓库指南*》了解如何刷新物化视图
- 《*Oracle 数据库高级复制*》了解物化视图日志

Query Rewrite

Query rewrite is an optimization technique that transforms a user request written in terms of master tables into a semantically equivalent request that includes materialized views. When base tables contain large amounts of data, computing an aggregate or join is expensive and time-consuming. Because materialized views contain precomputed aggregates and joins, query rewrite can quickly answer queries using materialized views.

查询重写

查询重写是一种优化技术，它将一个按主表写的用户请求，转换为在语义上等效的、包含物化视图的请求。当基表包含大量的数据时，计算一个聚合或联接是非常昂贵和费时的。因为物化视图包含预计算的聚合和联接，所以查询重写能使用物化视图迅速响应查询。

The optimizer **query transformer** transparently rewrites the request to use the materialized view, requiring no user intervention and no reference to the materialized view in the SQL statement. Because query rewrite is transparent, materialized views can be added or dropped without invalidating the SQL in the application code.

优化器的查询变换器透明地重写用户请求，以便使用物化视图，而无需用户干预或在 SQL 语句中引用物化视图。因为查询重写是透明的，可以添加或删除物化视图，而不会使应用程序中的 SQL 代码无效。

In general, rewriting queries to use materialized views rather than detail tables improves response time. Figure 4-8 shows the database generating an execution plan for the original and rewritten query and choosing the lowest-cost plan.

通常，启用重写查询以使用物化视图，而不是使用明细表，能减少响应时间。图 4-8 显示了数据库同时生成原始的和经过重写的查询执行计划，然后选择成本最低的计划。

Figure 4-8 Query Rewrite

Description of "Figure 4-8 Query Rewrite"

See Also:

- "Overview of the Optimizer"
- *Oracle Database Data Warehousing Guide* to learn how to use query rewrite

Overview of Sequences

A **sequence** is a schema object from which multiple users can generate unique integers. A sequence generator provides a highly scalable and well-performing method to generate surrogate keys for a number data type.

Sequence Characteristics

A sequence definition indicates general information, such as the following:

- The name of the sequence
- Whether the sequence ascends or descends
- The interval between numbers
- Whether the database should cache sets of generated sequence numbers in memory
- Whether the sequence should cycle when a limit is reached

The following example creates the sequence `customers_seq` in the

图 4－8 查询重写

Description of "Figure 4-8 Query Rewrite"

另见：

- "优化器概述"
- 《*Oracle 数据库数据仓库指南*》了解如何使用查询重写

序列概述

序列是一种模式对象，多个用户可以用它来生成唯一整数。序列发生器提供高可扩展性的和性能良好的方法，来为数字数据类型生成代理键。

序列特征

序列定义需要如下所示的常规信息：

- 序列的名称
- 序列是升序还是降序
- 编号之间的间隔
- 数据库是否要在内存中缓存生成的序列号集合
- 当超限时序列是否要重复

下面的示例在示例模式 oe 中创建序列 customers_seq。当向 `customers`

sample schema oe. An application could use this sequence to provide customer ID numbers when rows are added to the customers table.

```
CREATE SEQUENCE customers_seq
START WITH      1000
INCREMENT BY    1
NOCACHE
NOCYCLE;
```

The first reference to customers_seq.nextval returns 1000. The second returns 1001. Each subsequent reference returns a value 1 greater than the previous reference.

See Also:

- *Oracle Database 2 Day Developer's Guide* and *Oracle Database Administrator's Guide* to learn how to manage sequences
- *Oracle Database SQL Language Reference* for CREATE SEQUENCE syntax and semantics

Concurrent Access to Sequences

The same sequence generator can generate numbers for multiple tables. In this way, the database can generate primary keys automatically and coordinate keys across multiple rows or tables. For example, a sequence can generate primary keys for an orders table and a customers table.

The sequence generator is useful in multiuser environments for generating unique numbers without the overhead of disk I/O or transaction locking. For example, two users simultaneously insert new rows into the orders table. By using a sequence to generate unique numbers for the order_id column, neither user has to wait for the other to enter the next available order number. The sequence automatically generates the correct values for each user.

Each user that references a sequence has access to his or her current sequence number, which is the last sequence generated in the session. A user can issue a statement to generate a new sequence number or use the current number last generated by the session. After a statement in a session generates a sequence number, it is available only to this session. Individual sequence numbers can be skipped if they were generated and used in a transaction that was ultimately rolled back.

表中添加行时，应用程序可以使用此序列提供客户 ID 号。

```
CREATE SEQUENCE customers_seq
START WITH      1000
INCREMENT BY    1
NOCACHE
NOCYCLE;
```

对 customers_seq.nextval 的第一次引用会返回 1000。第二次引用返回 1001。每个后续引用的返回值比前一个引用大 1。

另见：

- 《*Oracle 数据库 2 日开发人员指南*》 和 《*Oracle 数据库管理员指南*》了解如何管理序列
- 《*Oracle 数据库 SQL 语言参考*》有关 CREATE SEQUENCE 的语法和语义

对序列的并发访问

同一序列发生器可以为多个表生成编号。以这种方式，数据库可以自动生成主键，并协调跨多个行或表的键。例如，序列可以为 orders 表和 customers 表生成主键。

序列发生器可用于在多用户环境中生成唯一编号，不会引起磁盘 I/O 开销或事务锁定。例如，两个用户同时向 orders 表中插入新行。通过使用序列来为 order_id 列生成唯一编号，任何一个用户都不必等待别人输入下一个可用的订单号。序列自动为每个用户生成正确的值。

每个引用序列的用户都有权访问他或她的当前序列号，即在该会话中生成的最后一个序列值。用户可以发出一个语句生成一个新的序列号，或使用该会话最后生成的当前序列号。在某个会话中的一个语句生成了一个序列号后，只有该会话可以使用它。单个序列号可以跳过，如果个别序列号在某个事务中被生成使用，而最终被回滚，则它们会被跳过（而变得不连续）。

Caution:
If your application requires a gap-free set of numbers, then you cannot use Oracle sequences. You must serialize activities in the database using your own developed code.
See Also:
Chapter 9, "Data Concurrency and Consistency"

Overview of Dimensions

A typical data warehouse has two important components: dimensions and facts. A dimension is any category used in specifying business questions, for example, time, geography, product, department, and distribution channel. A **fact** is an event or entity associated with a particular set of dimension values, for example, units sold or profits.
Examples of multidimensional requests include the following:

- Show total sales across all products at increasing aggregation levels for a geography dimension, from state to country to region, for 2007 and 2008.
- Create a cross-tabular analysis of our operations showing expenses by territory in South America for 2007 and 2008. Include all possible subtotals.
- List the top 10 sales representatives in Asia according to 2008 sales revenue for automotive products, and rank their commissions.

Many multidimensional questions require aggregated data and comparisons of data sets, often across time, geography or budgets.
Creating a dimension permits the broader use of the query rewrite feature. By transparently rewriting queries to use materialized views, the database can improve query performance.
See Also:
"Overview of Data Warehousing and Business Intelligence"

Hierarchical Structure of a Dimension

A **dimension table** is a logical structure that defines hierarchical relationships between pairs of columns or column sets. A dimension has no data storage assigned to it. Dimensional information is stored in dimension tables, whereas fact information is stored in a **fact table**. Within a customer dimension, customers could roll up to city, state, country, subregion, and region. Data analysis typically starts at higher levels in the dimensional hierarchy and gradually drills down if the

警告：
如果你的应用程序需要没有间隙的一组数字，则您不能使用 Oracle 序列。您必须在数据库中使用自己开发的代码来序列化你的程序行为。

另见：
第 9 章，"数据并发和一致性"

维度概述

一个典型的数据仓库有两个重要组成部分： 维度和事实。**维度**是用于阐述业务问题的类别，例如，时间、 地理位置、 产品、 部门、和分销渠道等。**事实**是与一组特定的维度的值相关联的事件或实体，例如销售量或利润。

需要使用多维度的示例包括以下：

- 以上升聚合级别方式，显示从 2007 年到 2008 年度，在地理维度上从州到国家到地区，跨越所有产品的总销售额。
- 对 2007 年到 2008 年度，我们在南美的运营情况，创建一个跨表格分析，按地区显示开支。包括所有可能的分类汇总。
- 根据 2008 年汽车产品销售收入，列出在亚洲的前 10 位销售代表，并按其佣金分级。

很多的多维问题往往需要跨时间、 地理位置、或预算，来聚合数据和比较数据集。
创建维度允许更广泛地使用查询重写功能。数据库可以通过透明地重写查询以使用物化视图来提高查询性能。

另见：
"数据仓库和商业智能的概述"

维度的层次结构

维度表是一个逻辑结构，用来定义列对或列集之间的层次关系。维度没有分配数据存储。维度信息存储在维度表中，而事实信息存储在**事实表**中。

在客户维度中，能将客户上卷到市、 州、 国家、 亚区、和地区。数据分析通常在多维层次结构中从较高级别开始，并在必要时可以逐步下钻。

situation warrants such analysis.
Each value at the child level is associated with one and only one value at the parent level. A hierarchical relationship is a **functional dependency** from one level of a hierarchy to the next level in the hierarchy.

See Also:

- *Oracle Database Data Warehousing Guide* to learn about dimensions
- *Oracle OLAP User's Guide* to learn how to create dimensions

Creation of Dimensions

Dimensions are created with SQL statements. The `CREATE DIMENSION` statement specifies:

- Multiple `LEVEL` clauses, each of which identifies a column or column set in the dimension
- One or more `HIERARCHY` clauses that specify the parent/child relationships between adjacent levels
- Optional `ATTRIBUTE` clauses, each of which identifies an additional column or column set associated with an individual level

The following statement was used to create the `customers_dim` dimension in the sample schema `sh`:

```
CREATE DIMENSION customers_dim
   LEVEL customer   IS (customers.cust_id)
   LEVEL city       IS (customers.cust_city)
   LEVEL state      IS
(customers.cust_state_province)
   LEVEL country    IS (countries.country_id)
   LEVEL subregion  IS
(countries.country_subregion)
   LEVEL region     IS (countries.country_region)
   HIERARCHY geog_rollup (
      customer      CHILD OF
      city          CHILD OF
      state         CHILD OF
      country       CHILD OF
      subregion     CHILD OF
      region
```

在子级别的每个值在父级别有且只有一个与其相关联的值。层次结构关系是层次结构中的一个级别对该层次结构中的下一个级别的**函数依赖**。

另见：

- 《*Oracle 数据库数据仓库指南*》了解维度
- 《*Oracle OLAP 用户指南*》了解如何创建维度

创建维度

维度是用 SQL 语句来创建的。`CREATE DIMENSION` 语句会指定：

- 多个 `LEVEL` 子句，每个 `LEVEL` 标识维度中的一个列或列集
- 一个或多个指定相邻级别之间的父/子关系的 `HIERARCHY` 子句
- 可选的 `ATTRIBUTE` 子句，每个 `ATTRIBUTE` 标识与某个单个级别相关联的额外的列或列集

下面的语句用来在示例模式 sh 中创建 customers_dim 维度：

```
CREATE DIMENSION customers_dim
   LEVEL customer   IS (customers.cust_id)
   LEVEL city       IS (customers.cust_city)
   LEVEL state      IS
(customers.cust_state_province)
   LEVEL country    IS (countries.country_id)
   LEVEL subregion  IS
(countries.country_subregion)
   LEVEL region     IS (countries.country_region)
   HIERARCHY geog_rollup (
      customer      CHILD OF
      city          CHILD OF
      state         CHILD OF
      country       CHILD OF
      subregion     CHILD OF
      region
```

```
   JOIN KEY (customers.country_id) REFERENCES
country )
   ATTRIBUTE customer DETERMINES
   (cust_first_name, cust_last_name, cust_gender,
    cust_marital_status, cust_year_of_birth,
    cust_income_level, cust_credit_limit)
   ATTRIBUTE country DETERMINES
(countries.country_name);
```

The columns in a dimension can come either from the same table (**denormalized**) or from multiple tables (**fully** or **partially normalized**). For example, a normalized time dimension can include a date table, a month table, and a year table, with join conditions that connect each date row to a month row, and each month row to a year row. In a fully denormalized time dimension, the date, month, and year columns are in the same table. Whether normalized or denormalized, the hierarchical relationships among the columns must be specified in the `CREATE DIMENSION` statement.

See Also:

- *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide* for information about how dimensions are used in a warehousing environment
- *Oracle Database SQL Language Reference* for `CREATE DIMENSION` syntax and semantics

Overview of Synonyms

A **synonym** is an alias for a schema object. For example, you can create a synonym for a table or view, sequence, PL/SQL program unit, user-defined object type, or another synonym. Because a synonym is simply an alias, it requires no storage other than its definition in the data dictionary.

Synonyms can simplify SQL statements for database users. Synonyms are also useful for hiding the identity and location of an underlying schema object. If the underlying object must be renamed or moved, then only the synonym must be redefined. Applications based on the synonym continue to work without modification.

You can create both private and public synonyms. A **private** synonym is

```
   JOIN KEY (customers.country_id) REFERENCES
country )
   ATTRIBUTE customer DETERMINES
   (cust_first_name, cust_last_name, cust_gender,
    cust_marital_status, cust_year_of_birth,
    cust_income_level, cust_credit_limit)
   ATTRIBUTE country DETERMINES
(countries.country_name);
```

维度中的列可以来自一个表 （**反规范化**） 或多个表 （**完全**或**部分规范化**）。例如，一个规范化的时间维度可以包含日期表、 月份表、和一个年度表，使用联接条件，将每个日期行连接到一个月份行，每个月份行联接到一个年度行。在一个完全反规范化的时间维度中， 日期、 月份、和年度都在同一个表中。无论规范化或反规范化，列之间的层次关系必须在 `CREATE DIMENSION` 语句中指定。

另见：

- 《*Oracle 仓库生成器数据建模、ETL 与数据质量指南*》有关在数据仓库环境中如何使用维度的信息
- 《*Oracle 数据库 SQL 语言参考*》关于 `CREATE DIMENSION` 的语法和语义

同义词概述

同义词是一个模式对象的别名。例如，您可以为一个表或视图、 序列、PL/SQL 程序单元、 用户定义的对象类型、或另一个同义词等创建同义词。因为同义词只是一个别名，因此除了要在数据字典存储其定义之外，不需要其它存储。

同义词可以为数据库用户简化 SQL 语句。同义词也可以用于隐藏底层模式对象的标识和位置。如果必须重命名或移动底层对象，仅需要重新定义同义词。基于同义词的应用程序，可以无需修改而继续工作。

您可以创建私有同义词和公共同义词。**私有**同义词与其所有者在同一个模式

in the schema of a specific user who has control over its availability to others. A **public** synonym is owned by the user group named PUBLIC and is accessible by every database user.

In Example 4-9, a database administrator creates a public synonym named people for the hr.employees table. The user then connects to the oe schema and counts the number of rows in the table referenced by the synonym.

Example 4-8 Public Synonym

```
SQL> CREATE PUBLIC SYNONYM people FOR
hr.employees;

Synonym created.

SQL> CONNECT oe
Enter password: password
Connected.
SQL> SELECT COUNT(*) FROM people;
  COUNT(*)
----------
       107
```

Use public synonyms sparingly because they make database consolidation more difficult. As shown in Example 4-9, if another administrator attempts to create the public synonym people, then the creation fails because only one public synonym people can exist in the database. Overuse of public synonyms causes namespace conflicts between applications.

Example 4-9 Public Synonym

```
SQL> CREATE PUBLIC SYNONYM people FOR
oe.customers;
CREATE PUBLIC SYNONYM people FOR oe.customers
                       *
ERROR at line 1:
ORA-00955: name is already used by an existing
object
SQL> SELECT OWNER, SYNONYM_NAME, TABLE_OWNER,
TABLE_NAME
  2  FROM DBA_SYNONYMS
```

中，只有其所有者对其可用性具有控制权。公共同义词由名为 PUBLIC 的用户组所有，并且能被每一个数据库用户访问。

在示例 4-9 中，数据库管理员为 hr.employees 表创建了一个叫做 people 的公共同义词。然后，用户连接到 oe 模式，并计算通过同义词引用的表中的行的数目。

示例 4—8 公共同义词

```
SQL> CREATE PUBLIC SYNONYM people FOR
hr.employees;

Synonym created.

SQL> CONNECT oe
Enter password: password
Connected.
SQL> SELECT COUNT(*) FROM people;
  COUNT(*)
----------
       107
```

请尽量少用公共同义词，因为它们使数据库整合更困难。如 4-9 例中所示，如果另一个管理员尝试创建 people 公共同义词，则会创建失败，因为在一个数据库中只能存在一个叫做 people 的公共同义词。过度使用公共同义词会导致应用程序之间的命名空间冲突。

示例 4—9 公共同义词

```
SQL> CREATE PUBLIC SYNONYM people FOR
oe.customers;
CREATE PUBLIC SYNONYM people FOR oe.customers
                       *
ERROR at line 1:
ORA-00955: name is already used by an existing
object
SQL> SELECT OWNER, SYNONYM_NAME, TABLE_OWNER,
TABLE_NAME
  2  FROM DBA_SYNONYMS
```

```
  3  WHERE SYNONYM_NAME = 'PEOPLE';

OWNER      SYNONYM_NAME TABLE_OWNER TABLE_NAME
---------- ------------ ----------- ----------
PUBLIC     PEOPLE       HR          EMPLOYEES
```

Synonyms themselves are not securable. When you grant object privileges on a synonym, you are really granting privileges on the underlying object. The synonym is acting only as an alias for the object in the `GRANT` statement.

See Also:

- *Oracle Database Administrator's Guide* to learn how to manage synonyms
- *Oracle Database SQL Language Reference* for `CREATE SYNONYM` syntax and semantics

```
  3  WHERE SYNONYM_NAME = 'PEOPLE';

OWNER      SYNONYM_NAME TABLE_OWNER TABLE_NAME
---------- ------------ ----------- ----------
PUBLIC     PEOPLE       HR          EMPLOYEES
```

同义词本身不是安全可控的。当您在一个同义词上授予对象权限时，你其实是在底层对象上授予权限。同义词在 GRANT 语句中只作为对象的别名。

另见：

- 《*Oracle 数据库管理员指南*》了解如何管理同义词
- 《*Oracle 数据库 SQL 语言参考*》关于 `CREATE SYNONYM` 的语法和语义

5 Data Integrity

This chapter explains how integrity constraints enforce the business rules associated with a database and prevent the entry of invalid information into tables.

This chapter contains the following sections:

- Introduction to Data Integrity
  - Techniques for Guaranteeing Data Integrity
  - Advantages of Integrity Constraints
- Types of Integrity Constraints
  - NOT NULL Integrity Constraints
  - Unique Constraints
  - Primary Key Constraints
  - Foreign Key Constraints
  - Check Constraints
- States of Integrity Constraints
  - Checks for Modified and Existing Data
  - Deferrable Constraints
  - Examples of Constraint Checking

See Also:

"Overview of Tables"

Introduction to Data Integrity

Business rules specify conditions and relationships that must always be true or must always be false. For example, each company defines its own policies about salaries, employee numbers, inventory tracking, and so on. It is important that data maintain **data integrity**, which is adherence to these rules, as determined by the database administrator or application developer.

Techniques for Guaranteeing Data Integrity

When designing a database application, developers have various options for guaranteeing the integrity of data stored in the database. These options include:

- Enforcing business rules with triggered stored database procedures,

第 5 章 数据完整性

本章介绍完整性约束如何强制执行与数据库关联的业务规则，并防止向表中输入无效信息。

本章包含以下各节：

- 数据完整性简介
  - 用于保证数据完整性的技术
  - 完整性约束的优势
- 完整性约束的类型
  - 非空完整性约束
  - 唯一性约束
  - 主键约束
  - 外键约束
  - 检查性约束
- 完整性约束的状态
  - 对现有数据和新数据的检查
  - 可延迟约束
  - 约束检查的示例

另见：

"表概述"

● 数据完整性简介

业务规则会指定一些条件和关系，它们要么必须始终为真，要么必须始终为假。例如，每个公司会定义其有关工资、 雇员人数、 库存跟踪，等方面的政策。保持数据的**数据完整性**很重要，它由数据库管理员或应用程序开发人员来确定，以遵从业务规则。

用于保证数据完整性的技术

在设计数据库应用程序时，开发人员有多种选项用于保证存储在数据库中的数据的完整性。这些选项包括：

- 如"触发器概述"中所述，通过数据库触发器存储过程，强制实施业务

as described in "Overview of Triggers"

- Using stored procedures to completely control access to data, as described in "Introduction to Server-Side Programming"
- Enforcing business rules in the code of a database application
- Using Oracle Database **integrity constraints**, which are rules defined at the column or object level that restrict values in the database

This chapter explains the basic concepts of integrity constraints.

Advantages of Integrity Constraints

An integrity constraint is a schema object that is created and dropped using SQL. To enforce data integrity, use integrity constraints unless it is not possible. Advantages of integrity constraints over alternatives for enforcing data integrity include:

- Declarative ease

  Because you define integrity constraints using SQL statements, no additional programming is required when you define or alter a table. The SQL statements are easy to write and eliminate programming errors.
- Centralized rules

  Integrity constraints are defined for tables and are stored in the data dictionary (see "Overview of the Data Dictionary"). Thus, data entered by all applications must adhere to the same integrity constraints. If the rules change at the table level, then applications need not change. Also, applications can use metadata in the data dictionary to immediately inform users of violations, even before the database checks the SQL statement.
- Flexibility when loading data

  You can disable integrity constraints temporarily to avoid performance overhead when loading large amounts of data. When the data load is complete, you can re-enable the integrity constraints.

See Also:

- *Oracle Database 2 Day Developer's Guide* and *Oracle Database*

规则

- 如"服务器端编程介绍"中所述，使用存储过程完全控制数据访问
- 在数据库应用程序的代码中执行业务规则
- 使用 Oracle **完整性约束**，它们是定义在列级或对象级上的，用来限制数据库中的值的规则

本章介绍完整性约束的基本概念。

完整性约束的优势

完整性约束是一个模式对象，它使用 SQL 来创建和删除。若要强制实施数据完整性，请尽可能使用完整性约束。相对于其他强制数据完整性替代方案，完整性约束包括如下优点：

- 容易声明

  在您定义或更改表时，使用 SQL 语句定义完整性约束，而无需任何额外的编程。SQL 语句易于编写，也易于排除编程错误。
- 集中化的规则

  完整性约束定义在表上，并存储在数据字典中 （请参阅"数据字典概述）。因此，由所有应用程序输入的数据都必须遵守相同的完整性约束。如果约束规则在表级发生了更改，应用程序不需要变更。此外，甚至在数据库检查 SQL 语句之前，应用程序就可以使用数据字典中的元数据立即告知用户的违例行为。
- 加载数据时的灵活性

  当加载大量数据时，您可以暂时禁用完整性约束，以避免性能开销。当数据加载完成后，您可以重新启用完整性约束。

另见：

- 《*Oracle 数据库 2 日开发人员指南*》 和 《*Oracle 数据库高级应用*

Advanced Application Developer's Guide to learn how to maintain data integrity

- *Oracle Database 2 Day DBA*and *Oracle Database Administrator's Guide* to learn how to manage integrity constraints

Types of Integrity Constraints

Oracle Database enables you to apply constraints both at the table and column level. A constraint specified as part of the definition of a column or attribute is called an **inline** specification. A constraint specified as part of the table definition is called an **out-of-line** specification.

The term **key** is used in the definitions of several types of integrity constraints. A key is the column or set of columns included in the definition of certain types of integrity constraints. Keys describe the relationships between the tables and columns of a relational database. Individual values in a key are called **key values**.

Table 5-1 describes the types of constraints. Each can be specified either inline or out-of-line, except for `NOT NULL`, which must be inline.

Table 5-1 Types of Constraints

Constraint Type	Description	See Also
`NOT NULL`	Allows or disallows inserts or updates of rows containing a null in a specified column.	"NOT NULL Integrity Constraints"
Unique key	Prohibits multiple rows from having the same value in the same column or combination of columns but allows some values to be null.	"Unique Constraints"
Primary key	Combines a `NOT NULL` constraint and a unique constraint. It prohibits multiple rows from having the same value in the same column or combination of columns and prohibits values from being null.	"Primary Key Constraints"
Foreign key	Designates a column as the foreign key and establishes a relationship between the foreign key and a primary or unique key, called the **referenced key**.	"Foreign Key Constraints"

程序开发人员指南》了解如何维护数据完整性

- 《*Oracle 数据库 2 日 DBA*》和《*Oracle 数据库管理员指南*》 了解如何管理完整性约束

完整性约束的类型

Oracle 数据库使您能够在表级或列级应用约束。作为列或属性定义的一部分而指定的约束，称为行内规范约束。作为表定义的一部分而指定的约束称为行外规范约束。

好几种类型的完整性约束定义中都使用**键**这个术语。键是某种类型的完整性约束的定义中包含的列或列集。键描述关系数据库中的表与列之间的关系。键中的单个值称为**键值**。

表 5-1 描述了约束的类型。除 NOT NULL 必须指定为行内规范，其它每一个都可以指定为行内或行外规范。

表 5－1 约束的类型

约束类型	描述	另见
`NOT NULL`	允许或不允许在指定的列插入或更新包含空值的行。	"非空完整性约束"
唯一键	在相同的列、或多个列的组合中，不允许多个行具有相同的值，但允许一些值为空。	"唯一性约束"
主键	合并 NOT NULL 约束和唯一性约束。在相同的列、或多个列的组合中，不允许多个行具有相同的值，也不允许为空。	"主键约束"
外键	指定一个列作为外键，在外键和主键或唯一键之间建立关系，也称为**引用键**。	"外键约束"

Check	Requires a database value to obey a specified condition.	"Check Constraints"
REF	Dictates types of data manipulation allowed on values in a REF column and how these actions affect dependent values. In an object-relational database, a built-in data type called a REF encapsulates a reference to a row object of a specified object type. Referential integrity constraints on REF columns ensure that there is a row object for the REF.	*Oracle Database Object-Relational Developer's Guide* to learn about REF constraints

See Also:

- "Overview of Tables"
- *Oracle Database SQL Language Reference* to learn more about the types of constraints

NOT NULL Integrity Constraints

A NOT NULL constraint requires that a column of a table contain no null values. A **null** is the absence of a value. By default, all columns in a table allow nulls.

NOT NULL constraints are intended for columns that must not lack values. For example, the hr.employees table requires a value in the last_name column. An attempt to insert an employee row without a last name generates an error:

```
SQL> INSERT INTO hr.employees (employee_id,
last_name) values (999, 'Smith');
.
.
.
ERROR at line 1:
ORA-01400: cannot insert NULL into
("HR"."EMPLOYEES"."LAST_NAME")
```

~~You can only add a column with a NOT NULL constraint if the table does not contain any rows or if you specify a default value.~~

You cannot add a column with a NOT NULL constraint if table has any rows unless you also specify the DEFAULT clause

检查性约束	要求某个数据库值服从指定的条件。	"检查性约束"
REF	规定对 REF 列中的值上的允许的数据操作类型，这些操作如何影响依赖值。在一个对象-关系的数据库中，一个叫做 REF 的内置数据类型，封装了对一个某种指定对象类型的行对象的引用。在 REF 列上的引用完整性约束确保该 REF 有一个行对象。	《*Oracle 数据库对象关系开发者指南*》了解 REF 约束

另见：

- "表概述"
- 《*Oracle 数据库 SQL 语言参考*》了解类型的约束的更多信息

非空完整性约束

NOT NULL 约束要求表中的列不包含空值。**空值**即值的缺失。默认情况下，一个表中的所有列都允许空值。

NOT NULL 约束主要用于不能缺少值的列。例如，hr.employees 表在 last_name 列上需要有值。试图插入一个没有姓氏的雇员行会生成一个错误：

```
SQL> INSERT INTO hr.employees (employee_id,
last_name) values (999, 'Smith');
.
.
.
ERROR at line 1:
ORA-01400: cannot insert NULL into
("HR"."EMPLOYEES"."LAST_NAME")
```

如果表已包含有任何行，则您不能添加一个具有 NOT NULL 约束的列，除非您还为其指定了 DEFAULT 子句

See Also:

- *Oracle Database 2 Day Developer's Guide* for examples of adding `NOT NULL` constraints to a table
- *Oracle Database SQL Language Reference* for restrictions on using `NOT NULL` constraints
- *Oracle Database Advanced Application Developer's Guide* to learn when to use the `NOT NULL` constraint

Unique Constraints

A **unique key constraint** requires that every value in a column or set of columns be unique. No rows of a table may have duplicate values in a column (the **unique key**) or set of columns (the **composite unique key**) with a unique key constraint.

Note:

The term **key** refers only to the columns defined in the integrity constraint. Because the database enforces a unique constraint by implicitly creating or reusing an index on the key columns, the term **unique key** is sometimes incorrectly used as a synonym for **unique key constraint** or **unique index**.

Unique key constraints are appropriate for any column where duplicate values are not allowed. Unique constraints differ from primary key constraints, whose purpose is to identify each table row uniquely, and typically contain values that have no significance other than being unique. Examples of unique keys include:

- A customer phone number, where the primary key is the customer number
- A department name, where the primary key is the department number

As shown in Example 2-1, a unique key constraint exists on the `email` column of the `hr.employees` table. The relevant part of the statement is as follows:

```
CREATE TABLE employees     ( ...
    , email           VARCHAR2(25)
        CONSTRAINT    emp_email_nn   NOT NULL ...
    , CONSTRAINT      emp_email_uk   UNIQUE
(email) ... );
```

The `emp_email_uk` constraint ensures that no two employees have the same email address, as shown in Example 5-1.

另见：

- 《*Oracle 数据库 2 日开发人员指南*》 关于向表中添加 NOT NULL 约束的示例
- 《*Oracle 数据库 SQL 语言参考*》关于使用 NOT NULL 约束的限制
- 《*Oracle 数据库高级应用开发指南*》了解何时使用 NOT NULL 约束

唯一性约束

唯一键约束要求在一个列或列集中的每个值是唯一的。在一个表中，不允许多个行在有唯一键约束的列（**唯一键**）或列集（**复合唯一键**）上具有重复值。

注意：

术语**键**仅指在完整性约束中定义的列。因为数据库通过在键列上隐含创建或重用索引来强制执行唯一性约束，术语**唯一键**有时会被错误地用作**唯一键约束**或**唯一索引**的同义词。

唯一键约束适合于任何不允许重复值的列。唯一约束与主键约束不同，主键的目的是唯一地标识表中的每一行，通常它只要求唯一，而并不一定要有什么实际意义。唯一键的示例包括：

- 客户电话号码，其主键是客户号
- 部门名称，其主键是部门编号

如示例 2-1 所示， 在 hr.employees 表的 `email` 列上存在一个唯一键约束。相关部分的语句如下所示：

```
CREATE TABLE employees     ( ...
    , email           VARCHAR2(25)
        CONSTRAINT    emp_email_nn   NOT NULL ...
    , CONSTRAINT      emp_email_uk   UNIQUE
(email) ... );
```

如示例 5-1 所示，emp_email_uk 约束可以确保没有任何两名雇员具有相同的电子邮件地址。

Example 5-1 Unique Constraint

```
SQL> SELECT employee_id, last_name, email FROM employees WHERE
email = 'PFAY';

EMPLOYEE_ID LAST_NAME                 EMAIL
----------- ------------------------- -------------------------
        202 Fay                       PFAY
SQL> INSERT INTO employees (employee_id, last_name, email,
hire_date, job_id)
  1  VALUES (999,'Fay','PFAY',SYSDATE,'ST_CLERK');
.
.
.
ERROR at line 1:
ORA-00001: unique constraint (HR.EMP_EMAIL_UK) violated
```

Unless a NOT NULL constraint is also defined, a null always satisfies a unique key constraint. Thus, columns with both unique key constraints and NOT NULL constraints are typical. This combination forces the user to enter values in the unique key and eliminates the possibility that new row data conflicts with existing row data.

Note:

Because of the search mechanism for unique key constraints on multiple columns, you cannot have identical values in the non-null columns of a partially null composite unique key constraint.

See Also:

- "Unique and Nonunique Indexes"
- *Oracle Database 2 Day Developer's Guide* for examples of adding UNIQUE constraints to a table

Primary Key Constraints

In a **primary key constraint**, the values in the group of one or more columns subject to the constraint uniquely identify the row. Each table can have one **primary key**, which in effect names the row and ensures that no duplicate rows exist.

A primary key can be natural or a surrogate. A **natural key** is a meaningful identifier made of existing attributes in a table. For example, a natural key could be a postal code in a lookup table. In contrast, a **surrogate key** is a system-generated incrementing identifier that ensures

示例 5-1 唯一约束

```
SQL> SELECT employee_id, last_name, email FROM employees WHERE
email = 'PFAY';

EMPLOYEE_ID LAST_NAME                 EMAIL
----------- ------------------------- -------------------------
        202 Fay                       PFAY
SQL> INSERT INTO employees (employee_id, last_name, email,
hire_date, job_id)
  1  VALUES (999,'Fay','PFAY',SYSDATE,'ST_CLERK');
.
.
.
ERROR at line 1:
ORA-00001: unique constraint (HR.EMP_EMAIL_UK) violated
```

除非也指定了 NOT NULL 约束，否则空值也始终满足唯一键约束。因此，典型的情况是在列上同时具有唯一键约束和非空约束。这种组合强制用户输入的是唯一值，并消除新行数据与现有行数据发生冲突的可能性。

注意：

鉴于在多个列上的唯一键约束的搜索机制，对含有部分空值的复合唯一键约束中，在非空列中不能有相同的值。

另见：

- "唯一索引和非唯一索引"
- 《*Oracle 数据库 2 日开发人员指南*》 有关向表中添加 UNIQUE 约束的示例

主键约束

在一个**主键约束**中的列或列集，其值能唯一地标识行。每个表只能有一个**主键**，起到确定行的作用，并确保不存在任何重复的行。

主键可以是自然键或代理键。**自然键**是由表中的现有属性组成的一个有意义的标识符。例如，一个自然键可能是查找表中的邮政编码。相比之下，**代理键**是一个系统生成的递增标识符，以确保在一个表中的唯一性。通常，由一

uniqueness within a table. Typically, surrogate keys are generated by a sequence.

The Oracle Database implementation of the primary key constraint guarantees that the following statements are true:

- No two rows have duplicate values in the specified column or set of columns.
- The primary key columns do not allow nulls.

A typical situation calling for a primary key is the numeric identifier for an employee. Each employee must have a unique ID. A employee must be described by one and only one row in the `employees` table.

Example 5-1 indicates that an existing employee has the employee ID of 202, where the employee ID is the primary key. The following example shows an attempt to add an employee with the same employee ID and an employee with no ID:

```
SQL> INSERT INTO employees (employee_id, last_name, email,
hire_date, job_id)
  1  VALUES (202,'Chan','ICHAN',SYSDATE,'ST_CLERK');
.
.
.
ERROR at line 1:
ORA-00001: unique constraint (HR.EMP_EMP_ID_PK) violated
SQL> INSERT INTO employees (last_name) VALUES ('Chan');
.
.
.
ERROR at line 1:
ORA-01400: cannot insert NULL into
("HR"."EMPLOYEES"."EMPLOYEE_ID")
```

The database enforces primary key constraints with an index. Usually, a primary key constraint created for a column implicitly creates a unique index and a `NOT NULL` constraint. Note the following exceptions to this rule:

- In some cases, as when you create a primary key with a deferrable constraint, the generated index is not unique.

Note:

个序列生成代理键。

Oracle 数据库实现的主键约束可以保证如下行为：

- 任何两行在指定的列或列集上都不具有重复值。
- 主键列不允许空值。

需要用到主键的典型情况是雇员的编号标识。每个员工必须具有唯一的 id。一名雇员只能由 `employees` 表中的有且仅有的一行来描述

示例 5-1 指示一个雇员 ID 为 202 的现有员工，其雇员 ID 是主键。下面的示例显示了试图添加一名具有相同雇员 ID 的雇员，和一名没有雇员 ID 的雇员：

```
SQL> INSERT INTO employees (employee_id, last_name, email,
hire_date, job_id)
  1  VALUES (202,'Chan','ICHAN',SYSDATE,'ST_CLERK');
.
.
.
ERROR at line 1:
ORA-00001: unique constraint (HR.EMP_EMP_ID_PK) violated
SQL> INSERT INTO employees (last_name) VALUES ('Chan');
.
.
.
ERROR at line 1:
ORA-01400: cannot insert NULL into
("HR"."EMPLOYEES"."EMPLOYEE_ID")
```

数据库使用索引来强制主键约束。通常，在某列上创建的主键约束会隐含创建一个唯一索引和一个非空约束。但请注意，此规则有如下例外情况：

- 有时，当您使用一个可延迟的约束选项来创建一个主键时，其生成的索引不是唯一的。

注意：

You can explicitly create a unique index with the CREATE UNIQUE INDEX statement.

- If a usable index exists when a primary key constraint is created, then the constraint reuses this index and does not implicitly create a new one.

By default the name of the implicitly created index is the name of the primary key constraint. You can also specify a user-defined name for an index. You can specify storage options for the index by including the ENABLE clause in the CREATE TABLE or ALTER TABLE statement used to create the constraint.

See Also:

Oracle Database 2 Day Developer's Guide and *Oracle Database Advanced Application Developer's Guide* to learn how to add primary key constraints to a table

Foreign Key Constraints

Whenever two tables contain one or more common columns, Oracle Database can enforce the relationship between the two tables through a **foreign key constraint**, also called a **referential integrity constraint**. The constraint requires that for each value in the column on which the constraint is defined, the value in the other specified other table and column must match. An example of a referential integrity rule is an employee can work for only an existing department.

Table 5-2 lists terms associated with referential integrity constraints.

Table 5-2 Referential Integrity Constraint Terms

Term	Definition
Foreign key	The column or set of columns included in the definition of the constraint that reference a referenced key. For example, the department_id column in employees is a foreign key that references the department_id column in departments. Foreign keys may be defined as multiple columns. However, a **composite foreign key** must reference a composite primary or unique key with the same number of columns and the same data types. The value of foreign keys can match either the referenced

您可以使用 CREATE UNIQUE INDEX 语句，显式创建一个唯一索引。

- 当你创建主键约束时，如果有一个现成的索引可用，则该主键约束会重用此索引，而不会隐式创建一个额外的新索引。

默认情况下，隐式创建的索引名称即是主键约束的名称。您也可以为索引指定一个用户定义的名称。您可以通过在用来创建该约束的 CREATE TABLE 或 ALTER TABLE 语句中包括 ENABLE 子句，来为索引指定存储选项。

另见：

《*Oracle 数据库 2 日开发人员指南*》 和 《*Oracle 数据库高级应用开发指南*》了解如何向表中添加主键约束

外键约束

只要两个表包含一个或多个公共列，则数据库可以通过一个**外键约束**（也称为**参照完整性约束**）来强制建立两个表之间的关系。该约束要求定义约束的列中的每个值，必须与另一个指定表中的指定列中的值相匹配。参照完整性规则的一个例子是，雇员只可以为一个现有的部门工作。

表 5-2 列出了与参照完整性约束相关的术语。

表 5-2 参照完整性约束术语

术语	定义
外键	约束定义中包含的列或列集，它引用了参考键。例如，employees 表中的 department_id 列是一个外键，它引用了 departments 表中的 department_id 列。 外键可以定义在多个列上。但是，**复合外键**必须引用具有相同数量和相同数据类型列的复合主键或复合唯一键。 外键的值，可以要么匹配被引用主键或唯一键的值，要么为空。如果一个复合外键的任何列为空，则该键的非空部分不一定要匹配父项中的任何相应部分。

	primary or unique key value, or be null. If any column of a composite foreign key is null, then the non-null portions of the key do not have to match any corresponding portion of a parent key.		
Referenced key	The unique key or primary key of the table referenced by a foreign key. For example, the `department_id` column in `departments` is the referenced key for the `department_id` column in `employees`.	被引用键	被外键所引用的表中的唯一键或主键。例如，相对于 `employees` 表中的 `department_id` 列来说，`departments` 表中的 `department_id` 列是被引用键。
Dependent or child table	The table that includes the foreign key. This table is dependent on the values present in the referenced unique or primary key. For example, the `employees` table is a child of `departments`.	依赖表或子表	包含外键的表。此表依赖于父表中被引用的唯一键或主键的值。例如，`employees` 表是 `departments` 表的一个子表。
Referenced or parent table	The table that is referenced by the foreign key of the child table. It is this table's referenced key that determines whether specific inserts or updates are allowed in the child table. For example, the `departments` table is a parent of `employees`.	被引用表或父表	由子表的外键引用的表。正是该表中的被引用值决定了在子表中特定的插入或更新是否可被允许。`Departments` 表是 `employees` 表的父表。

Figure 5-1 shows a foreign key on the `employees.department_id` column. It guarantees that every value in this column must match a value in the `departments.department_id` column. Thus, no erroneous department numbers can exist in the `employees.department_id` column.

图 5-1 显示在 employees.department_id 列上的一个外键。它保证在此列中的每个值必须与 departments.department_id 列中的某个值相匹配。因此，在 employees.department_id 列中不会存在错误的部门编号。

Figure 5-1 Referential Integrity Constraints

图 5-1 引用完整性约束

Description of "Figure 5-1 Referential Integrity Constraints"

See Also:

Oracle Database 2 Day Developer's Guide and *Oracle Database Advanced Application Developer's Guide* to learn how to add foreign key constraints to a table

Self-Referential Integrity Constraints

Figure 5-2 shows a **self-referential integrity constraint**. In this case, a foreign key references a parent key in the same table.

In Figure 5-2, the referential integrity constraint ensures that every value

Description of "Figure 5-1 Referential Integrity Constraints"

另见：

《*Oracle 数据库 2 日开发人员指南*》 和 《*Oracle 数据库高级应用开发指南*》了解如何向表中添加外键约束

自引用完整性约束

图 5-2 显示了一个自引用的完整性约束。在这种情况下，外键引用同一个表中的父键。

在图 5-2 中，参照完整性约束可以确保在 employees.manager_id 列中的每

in the `employees.manager_id` column corresponds to an existing value in the `employees.employee_id` column. For example, the manager for employee 102 must exist in the `employees` table. This constraint eliminates the possibility of erroneous employee numbers in the `manager_id` column.

Figure 5-2 Single Table Referential Constraints

Description of "Figure 5-2 Single Table Referential Constraints"

Nulls and Foreign Keys

The relational model permits the value of foreign keys to match either the referenced primary or unique key value, or be null. For example, a user could insert a row into `hr.employees` without specifying a department ID.

If any column of a composite foreign key is null, then the non-null portions of the key do not have to match any corresponding portion of a parent key.

Parent Key Modifications and Foreign Keys

The relationship between foreign key and parent key has implications for deletion of parent keys. For example, if a user attempts to delete the record for this department, then what happens to the records for employees in this department?

个值对应于 employees.employee_id 列中的一个现有值。例如，员工 102 的经理必须存在于 `employees` 表中。此约束消除了在 manager_id 列中存在错误的雇员人数的可能性。

图 5-2 单表引用约束

Description of "Figure 5-2 Single Table Referential Constraints"

空值和外键

关系模型允许外键的值可以匹配被引用主键或唯一键值，或者为空。例如，用户可以往 `hr.employees` 表插入行，而无需指定一个部门 id。

如果一个复合外键的任何列为空，则该键的非空部分不一定要匹配父项中的任何相应部分。

父键修和外键

删除父键会影响外键和父键之间的关系。例如，如果用户试图删除某个部门的记录，那么属于这个部门的雇员记录会发生事情呢？

When a parent key is modified, referential integrity constraints can specify the following actions to be performed on dependent rows in a child table:

- No action on deletion or update

  In the normal case, users cannot modify referenced key values if the results would violate referential integrity. For example, if `employees.department_id` is a foreign key to `departments`, and if employees belong to a particular department, then an attempt to delete the row for this department violates the constraint.
- Cascading deletions

  A deletion **cascades** (`DELETE CASCADE`) when rows containing referenced key values are deleted, causing all rows in child tables with dependent foreign key values to also be deleted. For example, the deletion of a row in `departments` causes rows for all employees in this department to be deleted.
- Deletions that set null

  A deletion **sets null** (`DELETE SET NULL`) when rows containing referenced key values are deleted, causing all rows in child tables with dependent foreign key values to set those values to null. For example, the deletion of a department row sets the `department_id` column value to null for employees in this department.

Table 5-3 outlines the DML statements allowed by the different referential actions on the key values in the parent table, and the foreign key values in the child table.

Table 5-3 DML Statements Allowed by Update and Delete No Action

DML Statement	Issued Against Parent Table	Issued Against Child Table
`INSERT`	Always OK if the parent key value is unique	OK only if the foreign key value exists in the parent key or is partially or all null
`UPDATE`	Allowed if the statement does	Allowed if the new foreign

在父键被修改时，参照完整性约束可以指定在子表中的相关行上，执行以下某种操作之一：

- 对删除或更新操作，不采取任何动作

  在正常的情况下，如果结果会违反参照完整性，用户不能修改被引用的键值。例如，如果 `employees.department_id` 是对 `departments` 表的一个外键，且有一些雇员属于某个特定部门，则试图删除该部门所在的行会违反约束。

- 级联删除

  级联删除(`DELETE CASCADE`)即是当包含被引用键值的行被删除时，导致子表中的所有的外键依赖值所在行也会被删除。例如，删除 `departments` 表中的某行，会导致这个部门中的所有雇员行也被删除。

- 对删除置空

  删除**置空**(`DELETE SET NULL`)即是当包含被引用键值的行被删除时，导致子表中的所有的外键依赖值被全部置空。例如，删除部门表中的某行，会导致将该部门中的所有雇员行的 `department_id` 列被置为空。

表 5-3 列出了在父表中的键值及子表中的外键值上，针对不同的引用性操作，所允许的 DML 语句。

表 5-3 Update 和 Delete No Action 所允许的 DML 语句

DML 语句	对父表发出	对子表发出
`INSERT`	若该父键值是唯一的，则总是可以	只有当外键值存在相应父键、或部分为空、或全为空时才可以

NO ACTION	not leave any rows in the child table without a referenced parent key value	key value still references a referenced key value
DELETE NO ACTION	Allowed if no rows in the child table reference the parent key value	Always OK
DELETE CASCADE	Always OK	Always OK
DELETE SET NULL	Always OK	Always OK

Note:
Other referential actions not supported by FOREIGN KEY integrity constraints of Oracle Database can be enforced using database triggers. See "Overview of Triggers".

See Also:
Oracle Database SQL Language Reference to learn about the ON DELETE clause

Indexes and Foreign Keys

As a rule, foreign keys should be indexed. The only exception is when the matching unique or primary key is never updated or deleted. Indexing the foreign keys in child tables provides the following benefits:

- Prevents a full table lock on the child table. Instead, the database acquires a row lock on the index.

- Removes the need for a full table scan of the child table. As an illustration, assume that a user removes the record for department 10 from the departments table. If employees.department_id is not indexed, then the database must scan employees to see if any employees exist in department 10.

See Also:
"Locks and Foreign Keys" and "Overview of Indexes"

Check Constraints

A **check constraint** on a column or set of columns requires that a specified condition be true or unknown for every row. If DML results in the

UPDATE NO ACTION	只要该语句在子表中不留下任何这样的行——其外键值在父表中不存在，则允许	只要新的外键值仍可以引用父表中的某个被引用键值，则允许
DELETE NO ACTION	只要子表中没有行引用该父键值，则允许	总是可以
DELETE CASCADE	总是可以	总是可以
DELETE SET NULL	总是可以	总是可以

注意：
其他 Oracle 数据库外键完整性约束不支持的引用性动作，可以使用数据库触发器强制执行。请参阅"触发器概述"。

另见：
《*Oracle 数据库 SQL 语言参考*》了解 ON DELETE 子句

索引和外键

作为一条规则——总是应该为外键编制索引。唯一的例外是，当匹配的唯一键或主键永远不会被更新或删除时。在子表中的外键上建立索引提供了下列好处：

- 可防止在子表上的全表锁定。相反，数据库只需要在索引上获取一个行锁。

- 消除在子表上进行全表扫描的需要。作为一个说明，假定用户从部门表中删除部门 10 的记录。如果没有为 employees.department_id 建立索引，则数据库必须扫描 employees 表，以查看是否存在属于部门 10 的雇员。

另见：
"锁和外键"和"索引概述"

检查性约束

在一个列或列集上的**检查约束**，要求所指定的条件为真，或对每一行来说是未知的。如果 DML 语句导致约束条件的计算结果为假，则 SQL 语句将被回

condition of the constraint evaluating to false, then the SQL statement is rolled back.

The chief benefit of check constraints is the ability to enforce very specific integrity rules. For example, you could use check constraints to enforce the following rules in the `hr.employees` table:

- The `salary` column must not have a value greater than 10000.
- The `commission` column must have a value that is not greater than the salary.

The following example creates a maximum salary constraint on `employees` and demonstrates what happens when a statement attempts to insert a row containing a salary that exceeds the maximum:

```
SQL> ALTER TABLE employees ADD CONSTRAINT
max_emp_sal CHECK (salary < 10001);
SQL> INSERT INTO employees
(employee_id,last_name,email,hire_date,job_id,sala
ry)
  1  VALUES
(999,'Green','BGREEN',SYSDATE,'ST_CLERK',20000);
.
.
.
ERROR at line 1:
ORA-02290: check constraint (HR.MAX_EMP_SAL)
violated
```

A single column can have multiple check constraints that reference the column in its definition. For example, the `salary` column could have one constraint that prevents values over 10000 and a separate constraint that prevents values less than 500.

If multiple check constraints exist for a column, then they must be designed so their purposes do not conflict. No order of evaluation of the conditions can be assumed. The database does not verify that check conditions are not mutually exclusive.

See Also:

Oracle Database SQL Language Reference to learn about restrictions for check constraints

States of Integrity Constraints

滚。

检查约束的主要好处是，具备强制执行非常具体的完整性规则的能力。例如可以使用检查性约束在 hr.employees 表中强制执行下列规则：

- 在 `salary` 列中不能有大于 10000 的值。
- `commission` 列必须有一个值，但不能大于 `salary`。

下面的示例在 `employees` 表上创建了一个最高薪金约束，并演示了当一个语句尝试插入一个其所包含的薪水超过了最大值的行时，会发生什么情况：

```
SQL> ALTER TABLE employees ADD CONSTRAINT
max_emp_sal CHECK (salary < 10001);
SQL> INSERT INTO employees
(employee_id,last_name,email,hire_date,job_id,sala
ry)
  1  VALUES
(999,'Green','BGREEN',SYSDATE,'ST_CLERK',20000);
.
.
.
ERROR at line 1:
ORA-02290: check constraint (HR.MAX_EMP_SAL)
violated
```

单个列可以在其定义中包括多个检查性约束。例如 `salary` 列可以有一个防止其值超过 10000 的约束，和另一个可以防止其值小于 500 的约束。

如果在某列上存在多个检查约束，则他们必须被合理设计，保证他们的目的不会发生冲突。不能假定多个条件计算之间的顺序。数据库不会验证这些检查条件是否相互排斥。

另见：

《*Oracle 数据库 SQL 语言参考*》了解检查约束的限制

完整性约束的状态

As part of constraint definition, you can specify how and when Oracle Database should enforce the constraint, thereby determining the **constraint state**.

Checks for Modified and Existing Data

The database enables you to specify whether a constraint applies to existing data or future data. If a constraint is **enabled**, then the database checks new data as it is entered or updated. Data that does not conform to the constraint cannot enter the database. For example, enabling a `NOT NULL` constraint on `employees.department_id` guarantees that every future row has a department ID. If a constraint is **disabled**, then the table can contain rows that violate the constraint.

You can set constraints to **validate** (`VALIDATE`) or not validate (`NOVALIDATE`) existing data. If `VALIDATE` is specified, then existing data must conform to the constraint. For example, enabling a `NOT NULL` constraint on `employees.department_id` and setting it to `VALIDATE` checks that every existing row has a department ID. If `NOVALIDATE` is specified, then existing data need not conform to the constraint.

The behavior of `VALIDATE` and `NOVALIDATE` always depends on whether the constraint is enabled or disabled. Table 5-4 summarizes the relationships.

Table 5-4 Checks on Modified and Existing Data

Modified Data	Existing Data	Summary
`ENABLE`	`VALIDATE`	Existing and future data must obey the constraint. An attempt to apply a new constraint to a populated table results in an error if existing rows violate the constraint.
`ENABLE`	`NOVALIDATE`	The database checks the constraint, but it need not be true for all rows. Thus, existing rows can violate the constraint, but new or modified rows must conform to the rules.
`DISABLE`	`VALIDATE`	The database disables the constraint, drops its index, and prevents modification of the constrained columns.
`DISABL`	`NOVALID`	The constraint is not checked and is not necessarily

作为约束定义的一部分，您可以指定数据库应如何及何时强制执行该约束条件，从而确定约束状态。

对现有数据和新数据的检查——针对数据

数据库使您可以指定将某个约束应用于现有数据还是应用于新数据。如果约束是**启用**的，则当输入或更新数据时，数据库会检查新的数据。不符合该约束的数据不能输入到数据库。例如，对 employees.department_id 启用 NOT NULL 约束，可以保证新插入的每一行都有一个部门 id。如果约束是**禁用**的，则表可能会包含违反约束的行。

您可以设置约束来**验证** （VALIDATE） 或不验证 (NOVALIDATE) 现有数据。如果指定了 VALIDATE ，则现有数据必须符合该约束。例如，对 employees.department_id 启用 NOT NULL 约束并将其设置为 VALIDATE，则会检查每个现有行都有一个部门 id。如果指定了 NOVALIDATE，则现有数据不需要符合该约束。

VALIDATE 和 NOVALIDATE 的行为都取决于约束是启用的还是禁用的。表 5-4 总结了这些选项组合。

表 5-4 对已修改数据和现有数据的检查

新数据	现有数据	总结
`ENABLE`	`VALIDATE`	现有的和将来的数据都必须遵守约束。如果试图在一个已填充的表上应用一个新的约束，而现有的行违反该约束，则会导致一个错误。
`ENABLE`	`NOVALIDATE`	数据库会检查该约束，但不需要对所有行为真。因此，现有的行可以违反该约束，但新的或修改后的行必须遵守该约束。
`DISABLE`	`VALIDATE`	数据库禁用该约束、删除其上的索引，并防止修改受约束的列。
`DISABL`	`NOVALID`	不检查约束，也不需要为真。

E	ATE	true.

See Also:

Oracle Database SQL Language Reference to learn about constraint states

Deferrable Constraints

Every constraint is either in a **not deferrable** (default) or **deferrable** state. This state determines when Oracle Database checks the constraint for validity. The following graphic depicts the options for deferrable constraints.

Description of the illustration cncpt313.gif

Nondeferrable Constraints

If a constraint is not deferrable, then Oracle Database never defers the validity check of the constraint to the end of the transaction. Instead, the database checks the constraint at the end of each statement. If the constraint is violated, then the statement rolls back.

For example, assume that you create a nondeferrable `NOT NULL` constraint for the `employees.last_name` column. If a user attempts to insert a row with no last name, then the database immediately rolls back the statement because the `NOT NULL` constraint is violated. No row is inserted.

Deferrable Constraints

A deferrable constraint permits a transaction to use the `SET CONSTRAINT` clause to defer checking of this constraint until a `COMMIT` statement is issued. If you make changes to the database that might violate the constraint, then this setting effectively lets you disable the constraint until all the changes are complete.

You can set the default behavior for when the database checks the deferrable constraint. You can specify either of the following attributes:

E	ATE	

另见：

《*Oracle 数据库 SQL 语言参考*》了解约束状态

可延迟约束——针对约束

约束可能处于**不可延迟**（默认值） 或**可延迟**两种状态之一。该状态确定数据库何时检查约束的有效性。下图描述了可延迟约束的选项。

Description of the illustration cncpt313.gif

不可延迟约束

如果一个约束不可延迟，则 Oracle 数据库决不会将约束的有效性检查延迟到事务结束。相反，数据库在每个语句的结尾检查约束。如果违反了约束，则该语句被回滚。

例如，假设您在 employees.last_name 列上创建一个不可延迟的 NULL 约束。如果用户试图插入一个没有姓氏的行，则数据库会立即回滚该语句，因为违反了 NOT NULL 约束。在这种情况下，将不会插入任何行。

可延迟约束

可延迟约束允许事务使用 `SET CONSTRAINT` 子句将约束检查推迟到发出 COMMIT 语句时。如果你对数据库中做了可能违反约束的更改，则此设置让您有效地禁用该约束，直到完成所有更改。

您可以设置数据库检查可延迟约束时的*默认行为*。您可以指定下列属性之一：

- INITIALLY IMMEDIATE

  The database checks the constraint immediately after each statement executes. If the constraint is violated, then the database rolls back the statement.

- INITIALLY DEFERRED

  The database checks the constraint when a COMMIT is issued. If the constraint is violated, then the database rolls back the transaction.

Assume that a deferrable NOT NULL constraint on employees.last_name is set to INITIALLY DEFERRED. A user creates a transaction with 100 INSERT statements, some of which have null values for last_name. When the user attempts to commit, the database rolls back all 100 statements. However, if this constraint were set to INITIALLY IMMEDIATE, then the database would not roll back the transaction.

If a constraint causes an action, then the database considers this action as part of the statement that caused it, whether the constraint is deferred or immediate. For example, deleting a row in departments causes the deletion of all rows in employees that reference the deleted department row. In this case, the deletion from employees is considered part of the DELETE statement executed against departments.

See Also:

Oracle Database SQL Language Reference for information about constraint attributes and their default values

Examples of Constraint Checking

Some examples may help illustrate when Oracle Database performs the checking of constraints. Assume the following:

- The employees table has the structure shown in Figure 5-2.

- INITIALLY IMMEDIATE

  数据库在每个语句执行后，立即检查约束。如果违反了约束，则数据库回滚该语句。→ 与 NOT DEFERRABLE 的区别是什么？？？

not deferrable 与initially immediate是一样的,与deferrable initially immediate则不同。后者允许将constraint再改为initially deferred, 前者不允许更改:

```
>create table test2 (f1 number(3) constraint fk references test(f1) initially immediate);
>alter table test2 modify constraint fk initially deferred;
alter table test2 modify constraint fk initially deferred
*
ERROR 位于第 1 行:
ORA-02447: 无法延迟不可延迟的约束条件

> create table test2 (f1 number(3) constraint fk references test(f1) deferrable initially immediate);
> alter table test2 modify constraint fk initially deferred;

表已更改。
```

- INITIALLY DEFERRED

  发出 COMMIT 时，数据库检查约束。如果违反了约束，则数据库回滚该事务。

假定一个在 employees.last_name 列上的可延迟的 NOT NULL 约束被设置为 INITIALLY DEFERRED。用户创建了一个包含 100 个 INSERT 语句的事务，其中一些在 last_name 列上具有空值。当用户尝试提交时，数据库将回滚所有 100 个语句。但是，如果这种限制被设置为 INITIALLY IMMEDIATE，则数据库将不会回滚整个事务。（只会回滚那些含有 NULL 值的行）

如果一个约束导致一个动作，则数据库认为此动作是引发它的语句的一部分，而无论该约束是可延迟的或立即的。例如，从 departments 表中删除一行，会导致删除 employees 表中引用了相应的被删部门的雇员。在这种情况下，从 employees 表中删除雇员被认为是对 departments 表执行的 DELETE 语句的一部分。

另见：

《*Oracle 数据库 SQL 语言参考*》关于约束属性及其默认值的信息

约束检查的示例

一些例子可以帮助说明数据库何时执行检查约束。假定如下：

- employees 表具有如图 5-2 所示的结构。

- The self-referential constraint makes entries in the `manager_id` column dependent on the values of the `employee_id` column.

- 自引用约束要求在 manager_id 列中的条目依赖 employee_id 列的值。

Insertion of a Value in a Foreign Key Column When No Parent Key Value Exists

Consider the insertion of the first row into the `employees` table. No rows currently exist, so how can a row be entered if the value in the `manager_id` column cannot reference any existing value in the `employee_id` column? Some possibilities are:

- A null can be entered for the `manager_id` column of the first row, if the `manager_id` column does not have a `NOT NULL` constraint defined on it.
  Because nulls are allowed in foreign keys, this row is inserted into the table.
- The same value can be entered in the `employee_id` and `manager_id` columns, specifying that the employee is his or her own manager.
  This case reveals that Oracle Database performs its constraint checking *after* the statement has been completely run. To allow a row to be entered with the same values in the parent key and the foreign key, the database must first run the statement (that is, insert the new row) and then determine whether any row in the table has an `employee_id` that corresponds to the `manager_id` of the new row.
- A multiple row `INSERT` statement, such as an `INSERT` statement with nested `SELECT` statement, can insert rows that reference one another.
  For example, the first row might have 200 for employee ID and 300 for manager ID, while the second row has 300 for employee ID and 200 for manager. Constraint checking is deferred until the complete execution of the statement. All rows are inserted first, and then all rows are checked for constraint violations.

Default values are included as part of an `INSERT` statement before the statement is parsed. Thus, default column values are subject to all integrity constraint checking.

当父键值不存在时，往外键列中插入值

考虑向 `employees` 表中插入第一行的情况。由于当前不存在任何行，那么如果 manager_id 列中的值不能引用在 employee_id 列中的任何现有值，如何才能输入新行呢？一些可能情况是：

- 如果 在 manager_id 列上没有定义 NOT NULL 约束，可以为第一行中的 manager_id 列输入一个空值。

  因为在外键中允许使用空值，此行可以被插入到表中。

- 如果该雇员是他或她自己的经理，可以在 employee_id 和 manager_id 列中输入相同的值。

  这种情况表明数据库是在完全运行该语句之后，执行其约束检查的。若要允许输入在父键和外键中具有相同值的行，数据库必须首先运行该语句 （也就是插入新的行），然后再确定表中是否有任何行其 employee_id 对应新行的 manager_id 列。

- 一个多行的 INSERT 语句，如一个具有嵌套 SELECT 语句的 INSERT 语句，可以插入相互引用的多个行。

  例如，第一行可能具有雇员 ID 200 和经理 ID 300，而第二行的雇员 ID 300 和经理 ID 200。约束检查被推迟到整个语句执行完成。首先，所有行先被插入，然后对所有行进行违反约束检查。

在语句进行解析之前，其默认值作为 INSERT 语句的一部分被包括在内。因此，默认列值都要遵从所有完整性约束检查。

An Update of All Foreign Key and Parent Key Values

Consider the same self-referential integrity constraint in a different

同时更新外键和父键的值

考虑该自引用完整性约束在另一个不同场景中的情况。假如该公司已出售。

scenario. The company has been sold. Because of this sale, all employee numbers must be updated to be the current value plus 5000 to coordinate with the employee numbers of the new company. Because manager numbers are really employee numbers (see Figure 5-3), the manager numbers must also increase by 5000.

Figure 5-3 The employees Table Before Updates

Description of "Figure 5-3 The employees Table Before Updates"

You could execute the following SQL statement to update the values:

```
UPDATE employees SET employee_id = employee_id + 5000,
  manager_id = manager_id + 5000;
```

Although a constraint is defined to verify that each `manager_id` value matches an `employee_id` value, the preceding statement is legal because the database effectively checks constraints after the statement completes. Figure 5-4 shows that the database performs the actions of the entire SQL statement before checking constraints.

Figure 5-4 Constraint Checking

Description of "Figure 5-4 Constraint Checking"

The examples in this section illustrate the constraint checking mechanism during `INSERT` and `UPDATE` statements, but the database uses the same mechanism for all types of DML statements. The same mechanism is used for all types of constraints, not just self-referential constraints.

Note:

Operations on a view or synonym are subject to the integrity constraints defined on the base tables.

由于这次出售，所有雇员编号必须都更新为其当前值加 5000，以适应新公司的员工编号。因为经理编号是真正雇员编号 （请参阅图 5-3)，经理编号也必须增加 5000。

图 5-3 的雇员表前更新

Description of "Figure 5-3 The employees Table Before Updates"

您可以执行下面的 SQL 语句，以更新其值：

```
UPDATE employees SET employee_id = employee_id + 5000,
  manager_id = manager_id + 5000;
```

虽然已定义了一个约束，以验证每个 manager_id 值与 employee_id 值匹配，但前面的语句仍然是合法的，因为数据库在该语句完成后检查约束。图 5-4 显示，数据库在检查约束之前执行整个 SQL 语句的操作。

图 5-4 约束检查

Description of "Figure 5-4 Constraint Checking"

本节中的示例只说明了在 INSERT 和 UPDATE 语句执行过程中的约束检查机制，实际上，数据库在所有类型的 DML 语句中都使用相同的机制。相同的机制用于所有类型的约束，而不只是自引用约束。

注意：

对视图或同义词上的操作，需要遵从定义在基础表上的完整性约束。

6 Data Dictionary and Dynamic Performance Views

This chapter describes the central set of read-only reference tables and views of each Oracle database, known collectively as the **data dictionary**. The chapter also describes the **dynamic performance views**, which are special views that are continuously updated while a database is open and in use.

This chapter contains the following sections:

- Overview of the Data Dictionary
  - Contents of the Data Dictionary
  - Storage of the Data Dictionary
  - How Oracle Database Uses the Data Dictionary
- Overview of the Dynamic Performance Views
  - Contents of the Dynamic Performance Views
  - Storage of the Dynamic Performance Views
- Database Object Metadata

Overview of the Data Dictionary

An important part of an Oracle database is its **data dictionary**, which is a read-only set of tables that provides administrative metadata about the database. A data dictionary contains information such as the following:

- The definitions of every schema object in the database, including default values for columns and integrity constraint information
- The amount of space allocated for and currently used by the schema objects
- The names of Oracle Database users, privileges and roles granted to users, and auditing information related to users (see "User Accounts")

The data dictionary is a central part of data management for every Oracle database. For example, the database performs the following actions:

- Accesses the data dictionary to find information about users, schema objects, and storage structures

第 6 章 数据字典和动态性能视图

本章介绍了每个 Oracle 数据库都具有的只读参考表和视图中最重要的部分，统称为**数据字典**。本章还介绍了**动态性能视图**，它们是一些会在数据库处于打开状态时不断更新的特殊视图。

本章包含以下各节：

- 数据字典概述
  - 数据字典的内容
  - 数据字典存储
  - Oracle 数据库如何使用数据字典
- 动态性能视图概述
  - 动态性能视图的内容
  - 动态性能视图的存储
- 数据库对象元数据

数据字典概述

Oracle 数据库的一个重要部分是它的**数据字典**，它是一组提供有关数据库管理元数据的只读表。数据字典包含如下信息：

- 在数据库中每个模式对象的定义，包括列的默认值和完整性约束信息
- 分配给模式对象的空间量及当前已使用量
- Oracle 数据库用户的名称、 授予用户的权限和角色、和与用户相关的审计信息（请参阅"用户帐户"）

数据字典是每个 Oracle 数据库数据管理的核心部分。例如，数据库执行下列操作：

- 访问数据字典来查找有关用户、 模式对象、和存储结构的信息

- Modifies the data dictionary every time that a DDL statement is issued (see "Data Definition Language (DDL) Statements")

Because Oracle Database stores data dictionary data in tables, just like other data, users can query the data with SQL. For example, users can run SELECT statements to determine their privileges, which tables exist in their schema, which columns are in these tables, whether indexes are built on these columns, and so on.

See Also:

"Introduction to Schema Objects"

Contents of the Data Dictionary

The data dictionary consists of the following types of objects:

- Base tables

  These underlying tables store information about the database. Only Oracle Database should write to and read these tables. Users rarely access the base tables directly because they are normalized and most data is stored in a cryptic format.
- Views

  These views decode the base table data into useful information, such as user or table names, using joins and WHERE clauses to simplify the information. These views contain the names and description of all objects in the data dictionary. Some views are accessible to all database users, whereas others are intended for administrators only.

Typically, data dictionary views are grouped in sets. In many cases, a set consists of three views containing similar information and distinguished from each other by their prefixes, as shown in Table 6-1. By querying the appropriate views, you can access only the information relevant for you.

Table 6-1 Data Dictionary View Sets

Prefix	User Access	Contents	Notes
DBA_	Database administrators	All objects	Some DBA_ views have additional columns containing information useful to the administrator.

- 每次发出 DDL 语句时，修改数据字典 （请参见"数据定义语言 (DDL) 语句"）

就像存储其他数据一样，数据库将数据字典数据存储在表中，用户可以使用 SQL 来查询其中的数据。例如，用户可以运行 SELECT 语句，以确定他们的权限，在他们的模式中有哪些表，这些表中有哪些列，这些列上是否建有索引等。

另见：

"模式对象介绍"

数据字典的内容

数据字典包含下列类型的对象：

- 基表

  这些基础表存储有关数据库的信息。只应该由数据库写入和读取这些表。用户很少直接访问基础表，因为他们已被规范化，且大多数数据存储为一种神秘的格式。

- 视图

  这些视图通过使用联接和 WHERE 子句来简化信息，将基础表的数据解码成有用的信息（如用户或表名等）。这些视图包含数据字典中的所有对象的名称和描述。一些视图可以被所有数据库用户访问，而其他一些则仅供管理员访问。

通常，数据字典视图按集合分组。在许多的情况下，一个集合由三个包含相似信息的视图，并且彼此之间通过不同的前缀来区分，如表 6-1 中所示。通过查询相应的视图，您可以访问只与您相关的信息。

表 6－1 数据字典视图集

前缀	用户访问	内容	注意
DBA_	数据库管理员	所有对象	一些 DBA_ 视图具有更多的列，其中包含对管理员有用的信息。

ALL_	All users	Objects to which user has privileges	Includes objects owned by user. These views obey the current set of enabled roles.
USER_	All users	Objects owned by user	Views with the prefix USER_ usually exclude the column OWNER. This column is implied in the USER_ views to be the user issuing the query.

Not all views sets have three members. For example, the data dictionary contains a DBA_LOCK view but no ALL_LOCK view.

The system-supplied DICTIONARY view contains the names and abbreviated descriptions of all data dictionary views. The following query of this view includes partial sample output:

```
SQL> SELECT * FROM DICTIONARY
  2  ORDER BY TABLE_NAME;

TABLE_NAME                     COMMENTS
------------------------------ ----------------------------------------
ALL_ALL_TABLES                 Description of all object and relational
                               tables accessible to the user

ALL_APPLY                      Details about each apply process that
                               dequeues from the queue visible to the
                               current user
.
.
.
```

See Also:

- *Oracle Database Reference* for a complete list of data dictionary views and their columns
- "Overview of Views"

Views with the Prefix DBA_

Views with the prefix DBA_ show all relevant information in the entire database. DBA_ views are intended only for administrators.

For example, the following query shows information about all objects in the database:

```
SELECT OWNER, OBJECT_NAME, OBJECT_TYPE
FROM   DBA_OBJECTS
ORDER BY OWNER, OBJECT_NAME;
```

ALL_	所有用户	用户具有访问权限的对象	包括用户所拥有的对象。这些视图服从当前已启用的角色集合。
USER_	所有用户	用户所拥有的对象	以 USER_为前缀的视图通常不包括 OWNER 列。此列隐含为对该 USER_ 视图发出查询的用户。

不是所有的视图集都有三个视图成员。例如，数据字典包含 DBA_LOCK 视图，但没有 ALL_LOCK 视图。

系统提供的 DICTIONARY 视图包含所有数据字典视图的名称和简短说明。如下的对此视图的查询包含部分示例输出：

```
SQL> SELECT * FROM DICTIONARY
  2  ORDER BY TABLE_NAME;

TABLE_NAME                     COMMENTS
------------------------------ ----------------------------------------
ALL_ALL_TABLES                 Description of all object and relational
                               tables accessible to the user

ALL_APPLY                      Details about each apply process that
                               dequeues from the queue visible to the
                               current user
.
.
.
```

另见：

- 《*Oracle 数据库参考*》关于数据字典视图及其列的完整列表
- "视图概述"

具有 DBA_前缀的视图

具有 DBA_前缀的视图显示在整个数据库中的所有相关信息。DBA_ 视图仅用于管理员。

例如，下面的查询显示数据库中的所有对象的有关信息：

```
SELECT OWNER, OBJECT_NAME, OBJECT_TYPE
FROM   DBA_OBJECTS
ORDER BY OWNER, OBJECT_NAME;
```

See Also:
Oracle Database Administrator's Guide for detailed information on administrative privileges

Views with the Prefix ALL_

Views with the prefix ALL_ refer to the user's overall perspective of the database. These views return information about schema objects to which the user has access through public or explicit grants of privileges and roles, in addition to schema objects that the user owns.

For example, the following query returns information about all the objects to which you have access:

```
SELECT OWNER, OBJECT_NAME, OBJECT_TYPE
FROM   ALL_OBJECTS
ORDER BY OWNER, OBJECT_NAME;
```

Because the ALL_ views obey the current set of enabled roles, query results depend on which roles are enabled, as shown in the following example:

```
SQL> SET ROLE ALL;

Role set.

SQL> SELECT COUNT(*) FROM ALL_OBJECTS;

COUNT(*)
----------
68295

SQL> SET ROLE NONE;

Role set.

SQL> SELECT COUNT(*) FROM ALL_OBJECTS;

COUNT(*)
----------
53771
```

Application developers should be cognizant of the effect of roles when using ALL_ views in a stored procedure, where roles are not enabled by

另见：
《*Oracle 数据库管理员指南*》关于管理权限的详细信息

具有 ALL_前缀的视图

具有 ALL_前缀的视图，是站在用户角度，从整体上看待数据库。这些视图除了包括用户本身拥有的模式对象之外，还包括该用户通过公共或显式授予的特权及角色所能访问的模式对象信息。

例如，如下的查询将返回有关您有权访问的所有对象的信息，

```
SELECT OWNER, OBJECT_NAME, OBJECT_TYPE
FROM   ALL_OBJECTS
ORDER BY OWNER, OBJECT_NAME;
```

因为 ALL_ 视图遵从当前已启用的角色集合，查询结果取决于已启用了哪些角色，如下例所示：

```
SQL> SET ROLE ALL;

Role set.

SQL> SELECT COUNT(*) FROM ALL_OBJECTS;

COUNT(*)
----------
68295

SQL> SET ROLE NONE;

Role set.

SQL> SELECT COUNT(*) FROM ALL_OBJECTS;

COUNT(*)
----------
53771
```

应用程序开发人员在一个存储过程中使用 ALL_ 视图时应认识到的角色的影响，默认情况下角色是未启用的。

default.

See Also:

"PL/SQL Subprograms"

Views with the Prefix USER_

The views most likely to be of interest to typical database users are those with the prefix USER_. These views:

- Refer to the user's private environment in the database, including metadata about schema objects created by the user, grants made by the user, and so on
- Display only rows pertinent to the user, returning a subset of the information in the ALL_ views
- Has columns identical to the other views, except that the column OWNER is implied
- Can have abbreviated PUBLIC synonyms for convenience

For example, the following query returns all the objects contained in your schema:

```
SELECT OBJECT_NAME, OBJECT_TYPE
FROM   USER_OBJECTS
ORDER BY OBJECT_NAME;
```

The DUAL Table

DUAL is a small table in the data dictionary that Oracle Database and user-written programs can reference to guarantee a known result. The dual table is useful when a value must be returned only once, for example, the current date and time. All database users have access to DUAL.

The DUAL table has one column called DUMMY and one row containing the value X. The following example queries DUAL to perform an arithmetical operation:

```
SQL> SELECT ((3*4)+5)/3 FROM DUAL;

((3*4)+5)/3
-----------
 5.66666667
```

See Also:

Oracle Database SQL Language Reference for more information about the DUAL table

另见：

"PL/SQL 子程序"

具有 USER _前缀的视图

典型的数据库用户最有可能感兴趣的视图可能是那些具有 USER_前缀的视图。这些视图：

- 指的是用户在数据库中的私有环境，包括用户所创建的模式对象的元数据，对该用户的授权等等
- 只显示与该用户相关的行，仅返回 ALL_视图中的信息的一个子集
- 除隐含的 OWNER 列外，与其他视图具有相同的列
- 为了方便，可能具有简洁的 PUBLIC 同义词

例如，以下查询将返回包含在你的模式中的所有对象：

```
SELECT OBJECT_NAME, OBJECT_TYPE
FROM   USER_OBJECTS
ORDER BY OBJECT_NAME;
```

DUAL 表

DUAL 是数据字典中的一个很小的表，数据库和用户编写的程序可以引用它，以保证一个已知的结果。，当一个值必须只能返回一次时，DUAL 表很有用，如当前日期和时间等。所有数据库用户都可以访问 DUAL 表。

DUAL 表只有一个称为 DUMMY 的列， 仅包含一个其值为 X 的行。下面的示例查询 DUAL 来执行一个算术操作：

```
SQL> SELECT ((3*4)+5)/3 FROM DUAL;

((3*4)+5)/3
-----------
 5.66666667
```

另见：

《*Oracle 数据库 SQL 语言参考*》有关 DUAL 表的详细信息

Storage of the Data Dictionary

The data dictionary base tables are the first objects created in any Oracle database. All data dictionary tables and views for a database are stored in the SYSTEM tablespace. Because the SYSTEM tablespace is always online when the database is open, the data dictionary is always available when the database is open.

See Also:

"The SYSTEM Tablespace" for more information about the SYSTEM tablespace

How Oracle Database Uses the Data Dictionary

The Oracle Database user SYS owns all base tables and user-accessible views of the data dictionary. Data in the base tables of the data dictionary *is necessary for Oracle Database to function*. Therefore, only Oracle Database should write or change data dictionary information. No Oracle Database user should *ever* alter rows or schema objects contained in the SYS schema because such activity can compromise data integrity. The security administrator must keep strict control of this central account.

Caution:

Altering or manipulating the data in data dictionary tables can permanently and detrimentally affect database operation.

During database operation, Oracle Database reads the data dictionary to ascertain that schema objects exist and that users have proper access to them. Oracle Database also updates the data dictionary continuously to reflect changes in database structures, auditing, grants, and data.

For example, if user hr creates a table named interns, then new rows are added to the data dictionary that reflect the new table, columns, segment, extents, and the privileges that hr has on the table. This new information is visible the next time the dictionary views are queried.

See Also:

"SYS and SYSTEM Schemas"

Public Synonyms for Data Dictionary Views

Oracle Database creates public synonyms for many data dictionary views so users can access them conveniently. The security administrator can also create additional public synonyms for schema objects that are used systemwide. Users should avoid naming their own schema objects with the same names as those used for public synonyms.

See Also:

数据字典存储

数据字典基表是在任何 Oracle 数据库中创建的第一批对象。所有数据库的数据字典表和视图都存储在 SYSTEM 表空间中。由于当数据库打开时，SYSTEM 表空间始终处于联机状态，所以当数据库打开时，数据字典总是可用的。

另见：

"SYSTEM 表空间"有关 SYSTEM 表空间的详细信息

Oracle 数据库如何使用数据字典

Oracle 数据库的 SYS 用户，拥有数据字典中的所有基表和用户可访问视图。数据字典基表中的数据，对于 Oracle 数据库发挥正常功能是必要的。因此，只应当由数据库写入或更改数据字典信息。无论何时，任何数据库用户都不应改变 SYS 模式中的表行或模式对象，因为这种行为可能会危及数据完整性。安全管理员必须严格控制这个最重要的帐户。

警告：

修改或操作数据字典表中的数据，可能会永久性地对数据库操作造成负面影响。

在数据库运行期间，数据库读取数据字典，以确定存在模式对象，且用户对它们具有适当的访问权限。Oracle 数据库也会不断地更新数据字典，以反映对数据库结构、审计、授权、和数据等所做的更改。

例如，如果 hr 用户创建了一个名为 interns 的表，则新行被添加到数据字典中，以反映新的表、列、 段、 区、和 hr 用户在该表上的权限。这些新的信息在字典视图下一次被查询时即可见。

另见：

"SYS 和 SYSTEM 模式"

数据字典视图的公共同义词

Oracle 数据库为许多数据字典视图创建公共同义词，以便用户可以方便地访问它们。安全管理员还可以为其它在系统范围内使用的模式对象创建公共同义词。用户应避免使用与公共同义词相同的名称来命名他们自己的模式对象。

另见：

"Overview of Synonyms"

Cache the Data Dictionary for Fast Access

Much of the data dictionary information is in the data dictionary cache because the database constantly requires the information to validate user access and verify the state of schema objects. Parsing information is typically kept in the caches. The COMMENTS columns describing the tables and their columns are not cached in the dictionary cache, but may be cached in the database buffer cache.

See Also:

"Data Dictionary Cache"

Other Programs and the Data Dictionary

Other Oracle Database products can reference existing views and create additional data dictionary tables or views of their own. Application developers who write programs that refer to the data dictionary should refer to the public synonyms rather than the underlying tables. Synonyms are less likely to change between releases.

Overview of the Dynamic Performance Views

Throughout its operation, Oracle Database maintains a set of virtual tables that record current database activity. These views are called **dynamic performance views** because they are continuously updated while a database is open and in use. The views, also sometimes called **V$ views**, contain information such as the following:

- System and session parameters
- Memory usage and allocation
- File states (including RMAN backup files)
- Progress of jobs and tasks
- SQL execution
- Statistics and metrics

The dynamic performance views have the following primary uses:

- Oracle Enterprise Manager uses the views to obtain information about the database (see "Oracle Enterprise Manager").

- Administrators can use the views for performance monitoring and debugging.

See Also:

Oracle Database Reference for a complete list of the dynamic performance

"同义词概述"

缓存数据字典以备快速访问

很多数据字典信息存在于数据字典缓存中，因为数据库需要这些信息，来不断验证用户的访问权限，并验证模式对象的状态。解析信息通常保存在缓存中。用于描述表及其列的 COMMENTS 列不会缓存在词典高速缓存中，但可缓存在数据库缓冲区高速缓存中。

另见：

"数据字典缓存"

其他程序与数据字典

其他 Oracle 数据库产品可能会引用现有的视图，并创建他们自己额外的数据字典表或视图。应用程序开发人员在编写需要引用数据字典的程序时，应该引用公共同义词而不是基表。同义词不太可能随版本变化而变化。

动态性能视图概述

在 Oracle 数据库的操作过程中，它会维护一组记录当前数据库活动的虚拟表。这些视图称为**动态性能视图**，因为他们在数据库处于打开状态和在使用过程中会不断更新。这些视图有时也称为 **V$ 视图**，包含如下所示的信息：

- 系统和会话参数
- 内存使用和分配
- 文件状态（包括 RMAN 备份文件）
- 工作和任务的进度
- SQL 运行
- 统计和度量

动态性能视图有以下主要用途：

- Oracle 企业管理器使用这些视图，来获取有关数据库的信息 （请参阅"Oracle 企业管理器"）。

- 管理员可以使用这些视图，用于性能监控和调试。

另见：

《*Oracle 数据库参考*》关于动态性能视图的完整列表

views

Contents of the Dynamic Performance Views

Dynamic performance views are sometimes called **fixed views** because they cannot be altered or removed by a database administrator. However, database administrators can query and create views on the tables and grant access to these views to other users.

SYS owns the dynamic performance tables, whose names begin with V_$. Views are created on these tables, and then public synonyms prefixed with V$. For example, the V$DATAFILE view contains information about data files. The V$FIXED_TABLE view contains information about all of the dynamic performance tables and views.

For almost every V$ view, a corresponding GV$ view exists. In Oracle Real Application Clusters (Oracle RAC), querying a GV$ view retrieves the V$ view information from all qualified database instances (see "Database Server Grid").

When you use the Database Configuration Assistant (DBCA) to create a database, Oracle automatically creates the data dictionary. Oracle Database automatically runs the catalog.sql script, which contains definitions of the views and public synonyms for the dynamic performance views. You must run catalog.sql to create these views and synonyms.

See Also:

- "Tools for Database Installation and Configuration" to learn about DBCA
- *Oracle Database Administrator's Guide* to learn how to run catalog.sql manually
- *Oracle Real Application Clusters Administration and Deployment Guide* to learn about using performance views in Oracle RAC

Storage of the Dynamic Performance Views

Dynamic performance views are based on virtual tables built from database memory structures. Thus, they are not conventional tables stored in the database. Read consistency is not guaranteed for the views because the data is updated dynamically.

Because the dynamic performance views are not true tables, the data is dependent on the state of the database and instance. For example, you can query V$INSTANCE and V$BGPROCESS when the database is

动态性能视图的内容

由于动态性能视图不能被数据库管理员更改或删除，有时它们也称为**固定视图**。但是，数据库管理员可以在表上执行查询和创建视图，并将这些视图的访问权限授予其他用户。

SYS 拥有其名称以 V_$ 开头的动态性能表。动态性能视图基于这些表创建，然后在视图上创建以 V$作为前缀的公共同义词。例如，V$DATAFILE 视图包含有关数据文件的信息。V$FIXED_TABLE 视图包含有关所有动态性能表和视图的信息。

对几乎所有的 V$ 视图，都存在一个相应的 GV$ 视图。在 Oracle 真正应用集群 (Oracle RAC)中，查询 GV$ 视图会从所有合适的数据库实例中检索其 V$ 视图信息（请参见"数据库服务器网格"）。

使用数据库配置助手 （DBCA） 创建数据库时， Oracle 会自动创建数据字典。Oracle 数据库将自动运行 catalog.sql 脚本，其中包含用于动态性能视图的视图和公共同义词的定义。您必须运行 catalog.sql 以创建这些视图和同义词。

另见：

- "数据库安装和配置工具"了解 DBCA
- 《*Oracle 数据库管理员指南*》了解如何手动运行 catalog.sql
- 《*Oracle 真正应用集群管理和部署指南*》了解如何使用在 Oracle RAC 中的性能视图

动态性能视图的存储

动态性能视图基于从数据库内存结构生成的虚拟表。因此，他们不是存储在数据库中的常规表。由于数据是动态更新的，所以不能保证视图的读一致性。

因为动态性能视图并不是真正的表，其数据取决于数据库和实例的状态。例如，当数据库已启动但未装入时，您可以查询 V$INSTANCE 和 V$BGPROCESS。但是，直到数据库已装入时，您才能查询

started but not mounted. However, you cannot query V$DATAFILE until the database has been mounted.

See Also:

Chapter 9, "Data Concurrency and Consistency"

Database Object Metadata

The DBMS_METADATA package provides interfaces for extracting complete definitions of database objects. The definitions can be expressed either as XML or as SQL DDL. Two styles of interface are provided: a flexible, sophisticated interface for programmatic control, and a simplified interface for ad hoc querying.

See Also:

Oracle Database PL/SQL Packages and Types Reference for more information about DBMS_METADATA

V$DATAFILE。

另见：

第 9 章，"数据并发和一致性"

数据库对象的元数据

DBMS_METADATA 软件包提供了用于提取数据库对象的完整定义的接口。定义可以按 XML 或 SQL DDL 来表示。它提供了两种样式的接口： 一个灵活的、 复杂的用于编程控制的接口，和一个简化的用于特定查询的接口。

另见：

《*Oracle 数据库 PL/SQL 包和类型参考*》关于 DBMS_METADATA 的更多信息

7 SQL

This chapter provides an overview of the Structured Query Language (SQL) and how Oracle Database processes SQL statements.

This chapter includes the following topics:

- Introduction to SQL
  - SQL Data Access
  - SQL Standards
- Overview of SQL Statements
  - Data Definition Language (DDL) Statements
  - Data Manipulation Language (DML) Statements
  - Transaction Control Statements
  - Session Control Statements
  - System Control Statement
  - Embedded SQL Statements
- Overview of the Optimizer
  - Use of the Optimizer
  - Optimizer Components
  - Access Paths
  - Optimizer Statistics
  - Optimizer Hints
- Overview of SQL Processing
  - Stages of SQL Processing
  - How Oracle Database Processes DML
  - How Oracle Database Processes DDL

Introduction to SQL

SQL (pronounced *sequel*) is the set-based, high-level declarative computer language with which all programs and users access data in an Oracle database. Although some Oracle tools and applications mask SQL use, all database operations are performed using SQL. Any other data access method circumvents the security built into Oracle Database and potentially compromises data security and integrity.

SQL provides an interface to a relational database such as Oracle

第 7 章 SQL

本章提供了结构化查询语言 (SQL) 的概述，以及 Oracle 数据库如何处理 SQL 语句。

本章包含以下主题：

- SQL 简介
  - SQL 数据访问
  - SQL 标准
- SQL 语句概述
  - 数据定义语言 (DDL) 语句
  - 数据操作语言 (DML) 语句
  - 事务控制语句
  - 会话控制语句
  - 系统控制语句
  - 嵌入式的 SQL 语句
- 优化器概述
  - 使用优化器
  - 优化器组件
  - 访问路径
  - 优化器统计
  - 优化器提示
- SQL 处理的概述
  - SQL 处理的阶段
  - Oracle 数据库如何处理 DML
  - Oracle 数据库如何处理 DDL

SQL 简介

SQL (发音为 *sequel*) 是基于集合的高级别声明性计算机语言，所有程序和用户都使用它来访问 Oracle 数据库中的数据。虽然某些 Oracle 工具和应用程序掩盖了对 SQL 的使用，但实际上所有的数据库操作都是使用 SQL 执行的。任何其他数据访问方法，绕过了内置于数据库的安全机制，并可能危及数据安全和完整性。

SQL 提供一个对关系数据库如 Oracle 数据库的接口。SQL 以一种一致的语

Database. SQL unifies tasks such as the following in one consistent language:

- Creating, replacing, altering, and dropping objects
- Inserting, updating, and deleting table rows
- Querying data
- Controlling access to the database and its objects
- Guaranteeing database consistency and integrity

SQL can be used **interactively**, which means that statements are entered manually into a program. SQL statements can also be **embedded** within a program written in a different language such as C or Java.

See Also:

- *Oracle Database SQL Language Reference* for an introduction to SQL
- "Introduction to Server-Side Programming" and "Client-Side Database Programming"

SQL Data Access

There are two broad families of computer languages: **declarative languages** that are nonprocedural and describe *what* should be done, and **procedural languages** such as C++ and Java that describe *how* things should be done. SQL is declarative in the sense that users specify the result that they want, not how to derive it. The SQL language compiler performs the work of generating a procedure to navigate the database and perform the desired task.

SQL enables you to work with data at the logical level. You need be concerned with implementation details only when you want to manipulate the data. For example, the following statement queries records for employees whose last name begins with `K`:

```
SELECT   last_name, first_name
FROM     hr.employees
WHERE    last_name LIKE 'K%'
ORDER BY last_name, first_name;
```

The database retrieves all rows satisfying the `WHERE` condition, also called the predicate, in a single step. These rows can be passed as a unit to the user, to another SQL statement, or to an application. You do not need to process the rows one by one, nor are you required to know how the rows are physically stored or retrieved.

All SQL statements use the optimizer, a part of Oracle Database that

言统一了以下任务：

- 创建、 替换、 改变、和删除对象
- 插入、 更新、和删除表行
- 查询数据
- 控制对数据库及其对象的访问
- 保证数据库的一致性和完整性

SQL 可以**交互地**使用，即将语句手动地输入到程序。SQL 语句也可以**嵌入**到用不同的语言（如 C 或 Java）编写的程序内。

另见：

- 《*Oracle 数据库 SQL 语言参考*》有关 SQL 的介绍
- "服务器端编程介绍"和"客户端数据库编程"

SQL 数据访问

有两大计算机语言派别： 非过程化的**声明性语言**，它描述应该做*什么*，和**过程化语言**，如 C + + 和 Java，它描述*如何*做。SQL 是声明性的，即是说用户可以指定他们想要的结果，而不必说明如何得到它。SQL 语言编译器的工作是，生成一个过程，以操作数据库，并执行所需的任务。

SQL 使您能够在逻辑层面使用数据。仅当您想要处理该数据时，您才需要关注其实现细节。例如，下面的语句查询姓氏以 K 开头的雇员记录


```
SELECT   last_name, first_name
FROM     hr.employees
WHERE    last_name LIKE 'K%'
ORDER BY last_name, first_name;
```


数据库用一个单一步骤检索满足 WHERE 条件（也称“谓词”）的所有行。这些行可以作为一个整体传递给用户、另一个 SQL 语句、或应用程序。您不需要一行一行地处理，您也不需要知道行是如何被物理存储和检索的。

所有 SQL 语句都使用优化器，作为 Oracle 数据库的一部分，它会确定访问

determines the most efficient means of accessing the specified data. Oracle Database also supports techniques that you can use to make the optimizer perform its job better.

See Also:

Oracle Database SQL Language Reference for detailed information about SQL statements and other parts of SQL (such as operators, functions, and format models)

SQL Standards

Oracle strives to follow industry-accepted standards and participates actively in SQL standards committees. Industry-accepted committees are the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO). Both ANSI and the ISO/IEC have accepted SQL as the standard language for relational databases.

The latest SQL standard was adopted in July 2003 and is often called SQL:2003. One part of the SQL standard, Part 14, SQL/XML (ISO/IEC 9075-14) was revised in 2006 and is often referred to as SQL/XML:2006.

Oracle SQL includes many extensions to the ANSI/ISO standard SQL language, and Oracle Database tools and applications provide additional statements. The tools SQL*Plus, SQL Developer, and Oracle Enterprise Manager enable you to run any ANSI/ISO standard SQL statement against an Oracle database and any additional statements or functions available for those tools.

See Also:

- *Oracle Database SQL Language Reference* for an explanation of the differences between Oracle SQL and standard SQL
- *SQL*Plus User's Guide and Reference* for SQL*Plus commands, including their distinction from SQL statements
- "Tools for Database Administrators" and "Tools for Database Developers"

Overview of SQL Statements

All operations performed on the information in an Oracle database are run using SQL **statements**. A SQL statement is a computer program or instruction that consists of identifiers, parameters, variables, names, data types, and SQL **reserved words**.

Note:

指定数据的最有效方式。Oracle 数据库还支持其它多种技术，可以使优化器更好地执行其工作。

另见：

《*Oracle 数据库 SQL 语言参考*》有关 SQL 语句和其它 SQL （例如，操作符、 函数、和格式模型）的详细信息

SQL 标准

Oracle 致力于遵循行业认可的标准，并积极参与到 SQL 标准委员会。被行业认可的委员会包括美国国家标准协会 (ANSI) 和国际组织的标准化 (ISO)。ANSI 和 ISO/IEC 都已接纳 SQL 作为关系数据库的标准语言。

最新的 SQL 标准在 2003 年 7 月通过，通常称为 SQL:2003。SQL 标准的第 14 部分 SQL/XML (ISO/IEC 9075-14) 在 2006 年被修订，并通常称为 SQL / XML:2006。

Oracle SQL 包含很多针对 ANSI/ISO 标准 SQL 语言的扩展，而且 Oracle 数据库工具和应用程序提供了一些其他语句。工具 SQL * Plus、SQL Developer 、和 Oracle 企业管理器使您能够在 Oracle 数据库上运行任何 ANSI/ISO 标准 SQL 语句，及可用于这些工具的其他语句或函数。

另见：

- 《*Oracle 数据库 SQL 语言参考*》关于标准 SQL 和 Oracle SQL 之间的差异的解释
- 《*SQL * PLUS 用户指南和参考*》关于 SQL*Plus 命令，包括他们与 SQL 语句的区别
- "数据库管理员工具"和"数据库开发人员工具"

SQL 语句概述

在 Oracle 数据库中的信息上执行的所有操作都是使用 SQL **语句**来执行的。SQL 语句是一种计算机程序或指令，它包含标识符、 参数、 变量、 名称、数据类型、和 SQL 的**保留字**。

注意：

SQL reserved words have special meaning in SQL and should not be used for any other purpose. For example, SELECT and UPDATE are reserved words and should not be used as table names.
A SQL statement must be the equivalent of a complete SQL sentence, such as:

```
SELECT last_name, department_id FROM employees
```

Oracle Database only runs complete SQL statements. A fragment such as the following generates an error indicating that more text is required:

```
SELECT last_name;
```

Oracle SQL statements are divided into the following categories:

- Data Definition Language (DDL) Statements
- Data Manipulation Language (DML) Statements
- Transaction Control Statements
- Session Control Statements
- System Control Statement
- Embedded SQL Statements

Data Definition Language (DDL) Statements

Data definition language (DDL) statements define, structurally change, and drop schema objects. For example, DDL statements enable you to:

- Create, alter, and drop schema objects and other database structures, including the database itself and database users. Most DDL statements start with the keywords CREATE, ALTER, or DROP.
- Delete all the data in schema objects without removing the structure of these objects (TRUNCATE).

  Note:
  Unlike DELETE, TRUNCATE generates no undo data, which makes it faster than DELETE. Also, TRUNCATE does not invoke delete triggers.
- Grant and revoke privileges and roles (GRANT, REVOKE).
- Turn auditing options on and off (AUDIT, NOAUDIT).
- Add a comment to the data dictionary (COMMENT).

DDL enables you to alter attributes of an object without altering the applications that access the object. For example, you can add a column to

SQL 保留字在 SQL 中具有特殊的意义，且不用于任何其他目的。例如，SELECT 和 UPDATE 是保留字，不能用作表的名称。

SQL 语句必须等价于一个完整的 SQL 句子，例如：

```
SELECT last_name, department_id FROM employees
```

Oracle 数据库仅运行完整的 SQL 语句。像下面这样的片段将生成一个错误，指示需要更多的文本：

```
SELECT last_name;
```

Oracle SQL 语句可以分为以下几类：

- 数据定义语言 (DDL) 语句
- 数据操作语言 (DML) 语句
- 事务控制语句
- 会话控制语句
- 系统控制语句
- 嵌入式 SQL 语句

数据定义语言 (DDL) 语句

数据定义语言 (DDL) 语句定义、更改、和删除模式对象。例如，DDL 语句使您能够：

- 创建、 更改、和删除模式对象和其他数据库结构，包括数据库本身和数据库用户。大部分的 DDL 语句以关键字 CREATE、 ALTER、或 DROP 开头。

- 删除模式对象中的所有数据，而不删除这些对象的结构（TRUNCATE）。

  注意：
  与 DELETE 不同， TRUNCATE 将不生成回滚数据，这使它比 DELETE 更快。此外，TRUNCATE 不会调用删除触发器。

- 授予和撤消权限和角色 （GRANT, REVOKE）。
- 打开和关闭审核选项 （AUDIT , NOAUDIT） 。
- 将注释添加到数据字典 （COMMENT）。

DDL 使您能够更改对象的属性，而无需更改访问该对象的应用程序。例

a table accessed by a human resources application without rewriting the application. You can also use DDL to alter the structure of objects while database users are performing work in the database.

Example 7-1 uses DDL statements to create the `plants` table and then uses DML to insert two rows in the table. The example then uses DDL to alter the table structure, grant and revoke privileges on this table to a user, and then drop the table.

Example 7-1 DDL Statements

```
CREATE TABLE plants
    ( plant_id    NUMBER PRIMARY KEY,
      common_name VARCHAR2(15) );
INSERT INTO plants VALUES (1, 'African Violet'); # DML statement
INSERT INTO plants VALUES (2, 'Amaryllis'); # DML statement
ALTER TABLE plants ADD
    ( latin_name VARCHAR2(40) );
GRANT SELECT ON plants TO scott;
REVOKE SELECT ON plants FROM scott;
DROP TABLE plants;
```

An implicit `COMMIT` occurs immediately before the database executes a DDL statement and a `COMMIT` or `ROLLBACK` occurs immediately afterward. In Example 7-1, two `INSERT` statements are followed by an `ALTER TABLE` statement, so the database commits the two `INSERT` statements. If the `ALTER TABLE` statement succeeds, then the database commits this statement; otherwise, the database rolls back this statement. In either case the two `INSERT` statements have already been committed.

See Also:

- "Overview of Database Security" to learn about privileges and roles
- *Oracle Database 2 Day Developer's Guide* and *Oracle Database Administrator's Guide* to learn how to create schema objects
- *Oracle Database SQL Language Reference* for a list of DDL statements

如，可以将某个列添加到由人力资源应用程序访问的一个表，而无需重写应用程序。您还可以在当数据库用户正在数据库中执行工作时，使用 DDL 更改对象的结构。

示例 7-1 使用 DDL 语句来创建 `plants` 表，然后使用 DML 在该表中插入两行。然后，该示例使用 DDL 更改表结构、 授予和撤消一个用户在该表上的权限，然后删除表。

示例 7-1 DDL 语句

```
CREATE TABLE plants
    ( plant_id    NUMBER PRIMARY KEY,
      common_name VARCHAR2(15) );
INSERT INTO plants VALUES (1, 'African Violet'); # DML statement
INSERT INTO plants VALUES (2, 'Amaryllis'); # DML statement
ALTER TABLE plants ADD
    ( latin_name VARCHAR2(40) );
GRANT SELECT ON plants TO scott;
REVOKE SELECT ON plants FROM scott;
DROP TABLE plants;
```

在数据库执行 DDL 语句之前会立即执行一个隐式提交，之后立即执行一个提交或回滚。在示例 7-1 中，两个 INSERT 语句后面跟了一个 ALTER TABLE 语句，因此数据库会提交这两个 INSERT 语句。如果 ALTER TABLE 语句成功，则数据库提交此语句 ；否则，数据库回滚此语句。在任一情况下，两个 INSERT 语句都是已经提交的。

另见：

- "数据库安全概述"了解权限和角色
- 《*Oracle 数据库 2 日开发人员指南*》 和 《*Oracle 数据库管理员指南*》了解如何创建模式对象
- 《*Oracle 数据库 SQL 语言参考*》关于 DDL 语句的列表

Data Manipulation Language (DML) Statements

Data manipulation language (DML) statements query or manipulate data in existing schema objects. Whereas DDL statements enable you to change the structure of the database, DML statements enable you to query or change the contents. For example, `ALTER TABLE` changes the structure of a table, whereas `INSERT` adds one or more rows to the

数据操作语言 (DML) 语句

数据操纵语言 (DML) 语句查询或操作现有模式对象中的数据。DDL 语句使您可以更改数据库的结构，而 DML 语句使您能够查询或更改其内容。例如，`ALTER TABLE` 更改一个的表的结构，而 `INSERT` 向表中添加一个或多个行。

table.

DML statements are the most frequently used SQL statements and enable you to:

- Retrieve or fetch data from one or more tables or views (SELECT).
- Add new rows of data into a table or view (INSERT) by specifying a list of column values or using a subquery to select and manipulate existing data.
- Change column values in existing rows of a table or view (UPDATE).
- Update or insert rows conditionally into a table or view (MERGE).
- Remove rows from tables or views (DELETE).
- View the execution plan for a SQL statement (EXPLAIN PLAN). See "How Oracle Database Processes DML".
- Lock a table or view, temporarily limiting access by other users (LOCK TABLE).

The following example uses DML to query the employees table. The example uses DML to insert a row into employees, update this row, and then delete it:

```
SELECT * FROM employees;

INSERT INTO employees (employee_id, last_name,
email, job_id, hire_date, salary)
  VALUES (1234, 'Mascis', 'JMASCIS', 'IT_PROG',
'14-FEB-2008', 9000);

UPDATE employees SET salary=9100 WHERE
employee_id=1234;

DELETE FROM employees WHERE employee_id=1234;
```

A collection of DML statements that forms a logical unit of work is called a transaction. For example, a transaction to transfer money could involve three discrete operations: decreasing the savings account balance, increasing the checking account balance, and recording the transfer in an account history table. Unlike DDL statements, DML statements do not

DML 语句是最经常使用的 SQL 语句，并使您能够：

- 从一个或多个表或视图，检索或获取数据（SELECT）。
- 通过指定一个列值列表，或使用一个子查询，来选择和操作现有数据，以将新的数据行添加到表中（INSERT）。
- 更改表或视图中的现有行中的值（UPDATE）。
- 更新或有条件地将行插入到表或视图 （MERGE）。
- 从表或视图中删除行（DELETE）。
- 查看 SQL 语句的执行计划（EXPLAIN PLAN）。请参阅"Oracle 数据库如何处理 DML"
- 锁定一个表或视图，临时限制其他用户的访问（LOCK TABLE）。

下面的示例使用 DML 来查询 employees 表。该示例使用 DML 往 employees 表中插入一行、 又更新此行、然后将其删除：

```
SELECT * FROM employees;

INSERT INTO employees (employee_id, last_name,
email, job_id, hire_date, salary)
  VALUES (1234, 'Mascis', 'JMASCIS', 'IT_PROG',
'14-FEB-2008', 9000);

UPDATE employees SET salary=9100 WHERE
employee_id=1234;

DELETE FROM employees WHERE employee_id=1234;
```

形成一个逻辑工作单元的 DML 语句的集合，称为一个事务。例如，一个转账事务可能涉及三个独立的操作： 减少储蓄帐户余额、 增加支票帐户余额、和在一个帐户历史记录表中记录转账日志。与 DDL 的语句不同的是，DML 语句不会隐式提交当前事务。

implicitly commit the current transaction.

See Also:

- "Introduction to Transactions "
- *Oracle Database 2 Day Developer's Guide* to learn how to query and manipulate data
- *Oracle Database SQL Language Reference* for a list of DML statements

SELECT Statements

A query is an operation that retrieves data from a table or view. `SELECT` is the only SQL statement that you can use to query data. The set of data retrieved from execution of a `SELECT` statement is known as a **result set**.

Table 7-1 shows two required keywords and two keywords that are commonly found in a `SELECT` statement. The table also associates capabilities of a `SELECT` statement with the keywords.

Table 7-1 Keywords in a SQL Statement

Keyword	Required?	Description	Capability
`SELECT`	Yes	Specifies which columns should be shown in the result. Projection produces a subset of the columns in the table. An expression is a combination of one or more values, operators, and SQL functions that resolves to a value. The list of expressions that appears after the `SELECT` keyword and before the `FROM` clause is called the **select list**.	Projection
`FROM`	Yes	Specifies the tables or views from which the data should be retrieved.	Joining
`WHERE`	No	Specifies a **condition** to filter rows, producing a subset of the rows in the table. A condition specifies a combination of one or	Selection

另见：

- "事务简介"
- 《*Oracle 数据库 2 日开发人员指南*》 了解如何查询和操作数据
- 《*Oracle 数据库 SQL 语言参考*》关于 DML 语句的列表

SELECT 语句

查询是一个从表或视图中检索数据的操作。`SELECT` 是你唯一可用于查询数据的 SQL 语句。从执行 SELECT 语句检索到的数据集被称为**结果集**。

表 7-1 显示了在一个 SELECT 语句中两个必需的关键字，和两个常见的关键字。该表也说明了这些关键字在 SELECT 语句中的功能。

表 7-1 在 SQL 语句中的关键字

关键字	必需?	描述	功能
`SELECT`	是	指定应在结果中显示哪些列。投影会生成表中所包含列的一个子集。 表达式是一个或多个值、 操作符、和 SQL 函数的组合，经过计算后最终得到一个值。出现在 SELECT 关键字之后，和在 FROM 子句之前的表达式列表，叫做**选择列表**。	投影
`FROM`	是	指定应从哪些表或视图中检索数据。	联接
`WHERE`	否	指定一个**条件**来筛选行，以生成表中的行的一个子集。条件指定一个或多个表达式和逻辑（布尔） 运算符的组合，并返回一个值	选择

		more expressions and logical (Boolean) operators and returns a value of TRUE, FALSE, or UNKNOWN.	
ORDER BY	No	Specifies the order in which the rows should be shown.	

See Also:

Oracle Database SQL Language Reference for SELECT syntax and semantics

Joins

A join is a query that combines rows from two or more tables, views, or materialized views. Example 7-2 joins the employees and departments tables (FROM clause), selects only rows that meet specified criteria (WHERE clause), and uses projection to retrieve data from two columns (SELECT). Sample output follows the SQL statement.

Example 7-2 Sample Join

```
SELECT email, department_name
FROM   employees JOIN departments
ON     employees.department_id = departments.department_id
WHERE  employee_id IN (100,103)
ORDER BY email;

EMAIL                     DEPARTMENT_NAME
------------------------- ------------------------------
AHUNOLD                   IT
SKING                     Executive
```

Figure 7-1 graphically represents the operations of projection and selection in the join shown in Example 7-2.

Figure 7-1 Projection and Selection

		（TRUE、 FALSE、或 UNKNOWN）。	
ORDER BY	No	指定行应以何种顺序显示。	

另见：

《*Oracle 数据库 SQL 语言参考*》关于 SELECT 的语法和语义

连接

连接是合并两个或多个表、 视图、或物化视图中的行的查询。示例 7-2 将 employees 表与 departments 表联接(FROM 子句），只选择满足指定条件的行（WHERE 子句），并使用投影从两个列中检索数据（SELECT）。该 SQL 语句的示例输出如下所示。

示例 7-2 连接的例子

```
SELECT email, department_name
FROM   employees JOIN departments
ON     employees.department_id = departments.department_id
WHERE  employee_id IN (100,103)
ORDER BY email;

EMAIL                     DEPARTMENT_NAME
------------------------- ------------------------------
AHUNOLD                   IT
SKING                     Executive
```

图 7-1 以图形方式显示了示例 7-2 中所示连接中的投影和选择操作。

图 7-1 投影和选择

Table DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
60	IT	103	1400
90	Executive	100	1700

Projection
Selection

```
SELECT email,department_name
FROM employees JOIN
departments
ON employees.department_id =
departments.department_id
WHERE employee_id IN (100,103)
ORDER BY email
```

Table EMPLOYEES

EMPLOYEE_ID	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	MANAGER_ID	DEPARTMENT_ID
100	King	SKING		AD_PRES		90
101	Kochhar	NKOCHHAR	21-SEP-89	AD_VP	100	90
102	De Hann	LDEHANN	13-JAN-93	AD_VP	100	90
103	Hunold	AHUNOLD		IT_PROG	102	60

Description of "Figure 7-1 Projection and Selection"

Most joins have at least one join condition, either in the `FROM` clause or in the `WHERE` clause, that compares two columns, each from a different table. The database combines pairs of rows, each containing one row from each table, for which the join condition evaluates to `TRUE`. The optimizer determines the order in which the database joins tables based on the join conditions, indexes, and any available statistics for the tables.
Join types include the following:

- Inner joins

  An **inner join** is a join of two or more tables that returns only rows that satisfy the join condition. For example, if the join condition is `employees.department_id=departments.department_id`, then rows that do not satisfy this condition are not returned.
- Outer joins

  An **outer join** returns all rows that satisfy the join condition and also returns rows from one table for which no rows from the other table satisfy the condition. For example, a **left outer join** of `employees` and `departments` retrieves all rows in the `employees` table even if there is no match in `departments`. A **right outer join** retrieves all rows in `departments` even if there is no match in `employees`.

Table DEPARTMENTS

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
60	IT	103	1400
90	Executive	100	1700

Projection
Selection

```
SELECT email,department_name
FROM employees JOIN
departments
ON employees.department_id =
departments.department_id
WHERE employee_id IN (100,103)
ORDER BY email
```

Table EMPLOYEES

EMPLOYEE_ID	LAST_NAME	EMAIL	HIRE_DATE	JOB_ID	MANAGER_ID	DEPARTMENT_ID
100	King	SKING		AD_PRES		90
101	Kochhar	NKOCHHAR	21-SEP-89	AD_VP	100	90
102	De Hann	LDEHANN	13-JAN-93	AD_VP	100	90
103	Hunold	AHUNOLD		IT_PROG	102	60

Description of "Figure 7-1 Projection and Selection"

大多数连接都至少有一个连接条件，要么在 FROM 子句中，要么在 WHERE 子句中，比较来自不同表的两个列。数据库合并行对，每个行对包含来自不同表、且连接条件计算结果为 TRUE 的一行。基于连接条件、 索引、和任何可用的表统计信息，由优化器确定数据库中联接表的顺序。

联接类型包括：

- 内连接

  内连接是两个或更多表的联接，只返回满足连接条件的行。例如，如果连接条件是 employees.department_id = departments.department_id ，则不满足此条件的行不被返回。

- 外连接

  外连接返回所有满足联接条件的行，也返回一个表中不满足与另一个表的连接条件的行。例如，雇员表和部门表的**左外连接**，会检索雇员表中的所有行，即使在部门表中没有相应匹配的部门。**右外连接**会检索部门表中的所有行，即使在雇员表中没有匹配的雇员。

- Cartesian products

  If two tables in a join query have no join condition, then the database returns their **Cartesian product**. Each row of one table combines with each row of the other. For example, if `employees` has 107 rows and `departments` has 27, then the Cartesian product contains 107*27 rows. A Cartesian product is rarely useful.

See Also:
Oracle Database SQL Language Reference for detailed descriptions and examples of joins

Subqueries and Implicit Queries

A **subquery** is a `SELECT` statement nested within another SQL statement. Subqueries are useful when you must execute multiple queries to solve a single problem.

Each query portion of a statement is called a **query block**. In Example 7-3, the subquery in parentheses is the **inner query block**. The inner `SELECT` statement retrieves the IDs of departments with location ID 1800. These department IDs are needed by the **outer query block**, which retrieves names of employees in the departments whose IDs were supplied by the subquery.

Example 7-3 Subquery

```
SELECT first_name, last_name
FROM   employees
WHERE  department_id
IN     (SELECT department_id FROM departments
WHERE location_id = 1800);
```

The structure of the SQL statement does not force the database to execute the inner query first. For example, the database could rewrite the entire query as a join of `employees` and `departments`, so that the subquery never executes by itself. As another example, the Virtual Private Database (VPD) feature could restrict the query of employees using a `WHERE` clause, so that the database decides to query the employees first and then obtain the department IDs. The optimizer determines the best sequence of steps to retrieve the requested rows.

An **implicit query** is a component of a DML statement that retrieves data without using a subquery. An `UPDATE`, `DELETE`, or `MERGE` statement that does not explicitly include a `SELECT` statement uses an implicit query to retrieve rows to be modified. For example, the following

- 笛卡儿积

  如果两个表联接查询中的没有联接条件，则数据库返回其**笛卡儿积**。一个表中的每一行与另一个表中的每一行联合。例如，如果 `employees` 表有 107 行，而 `departments` 表有 27 行，则其笛卡儿积包含 107 x 27 行。笛卡儿积是很少用到的。

另见：
《*Oracle 数据库 SQL 语言参考*》关于联接的详细的说明和示例

子查询和隐式查询

子查询是嵌套在另一个 SQL 语句中的 SELECT 语句。在您必须执行多个查询以解决一个问题时，子查询非常有用。

一个语句每个查询部分称为一个**查询块**。在示例 7-3 中，括号内的子查询是**内部查询块**。内部的 SELECT 语句检索位置 ID 为 1800 的部门 id。这些部门 id 需要在**外部查询块**中用到，外部查询基于子查询提供的部门 ID 来检索员工姓名。

示例 7-3 子查询

```
SELECT first_name, last_name
FROM   employees
WHERE  department_id
IN     (SELECT department_id FROM departments
WHERE location_id = 1800);
```

SQL 语句的结构不会强制数据库必须首先执行内部查询。例如，数据库可能会重写整个查询，将 `employees` 表和 `departments` 表连接，这样，子查询本身永远不会单独执行。再举一个例子，虚拟专用数据库 (VPD) 功能可能会使用一个 WHERE 子句来限制对员工的查询，这样，数据库可能决定首先查询雇员表，然后获取部门 id。由优化器确定用于检索请求行的最佳步骤序列。

隐式查询是 DML 语句的一部分，它并不使用子查询来检索数据。一个没有显式包含 SELECT 语句的更新、 删除、或合并语句，会使用一个隐式查询来检索要修改的行。例如，下面的语句包括一个对 Baer 记录的隐式查询：

statement includes an implicit query for the Baer record:

```
UPDATE employees
  SET salary = salary*1.1
  WHERE last_name = 'Baer';
```

The only DML statement that does *not* necessarily include a query component is an INSERT statement with a VALUES clause. For example, an INSERT INTO TABLE mytable VALUES (1) statement does not retrieve rows before inserting a row.

See Also:

"Virtual Private Database (VPD)"

Transaction Control Statements

Transaction control statements manage the changes made by DML statements and group DML statements into transactions. These statements enable you to:

- Make changes to a transaction permanent (COMMIT).
- Undo the changes in a transaction, since the transaction started (ROLLBACK) or since a savepoint (ROLLBACK TO SAVEPOINT). A **savepoint** is a user-declared intermediate marker within the context of a transaction.

  Note:

  The ROLLBACK command ends a transaction, but ROLLBACK TO SAVEPOINT does not.
- Set a point to which you can roll back (SAVEPOINT).
- Establish properties for a transaction (SET TRANSACTION).
- Specify whether a deferrable integrity constraint is checked following each DML statement or when the transaction is committed (SET CONSTRAINT).

The following example starts a transaction named Update salaries. The example creates a savepoint, updates an employee salary, and then rolls back the transaction to the savepoint. The example updates the salary to a different value and commits.

```
UPDATE employees
  SET salary = salary*1.1
  WHERE last_name = 'Baer';
```

唯一不需要包括一个查询组件的 DML 语句，是带 VALUES 子句的 INSERT 语句。例如，INSERT INTO TABLE mytable VALUES (1)语句在插入行之前不会检索行。

另见：

"虚拟专用数据库 （VPD）"

事务控制语句

事务控制语句管理 DML 语句所做的更改，和将多个 DML 语句按事务分组。这些语句使您能够：

- 使一个事务的更改持久化（COMMIT）。
- 撤消在一个事务中自事务开始以来的更改（ROLLBACK），或自一个保存点以来的更改 （ROLLBACK TO SAVEPOINT）。**保存点**是在一个事务上下文中由用户声明的中间标记。

  注意：

  ROLLBACK 命令会结束一个事务，但 ROLLBACK TO SAVEPOINT 不会。
- 设置一个你可以回滚到的点（SAVEPOINT）。
- 设定一个事务的属性（SET TRANSACTION）。
- 指定是在每个 DML 语句之后、还是在事务时提交后，执行可延迟完整性约束检查（SET CONSTRAINT）。

下面的示例启动一个名为 Update salaries 的事务。该示例创建一个保存点，更新一个雇员的工资，然后回滚事务到保存点。该示例又更新一个不同的薪金值，然后提交。

```
SET TRANSACTION NAME 'Update salaries';

SAVEPOINT before_salary_update;

UPDATE employees SET salary=9100 WHERE employee_id=1234 # DML

ROLLBACK TO SAVEPOINT before_salary_update;

UPDATE employees SET salary=9200 WHERE employee_id=1234 # DML

COMMIT COMMENT 'Updated salaries';
```

See Also:

- "Introduction to Transactions "
- "Deferrable Constraints"
- *Oracle Database SQL Language Reference*

Session Control Statements

Session control statements dynamically manage the properties of a user session. As explained in "Connections and Sessions", a session is a logical entity in the database instance memory that represents the state of a current user login to a database. A session lasts from the time the user is authenticated by the database until the user disconnects or exits the database application.

Session control statements enable you to:

- Alter the current session by performing a specialized function, such as enabling and disabling SQL tracing (`ALTER SESSION`).
- Enable and disable roles, which are groups of privileges, for the current session (`SET ROLE`).

The following example turns on SQL tracing for the session and then enables all roles granted in the current session except `dw_manager`:

```
ALTER SESSION SET SQL_TRACE = TRUE;

SET ROLE ALL EXCEPT dw_manager;
```

Session control statements do not implicitly commit the current transaction.

See Also:

```
SET TRANSACTION NAME 'Update salaries';

SAVEPOINT before_salary_update;

UPDATE employees SET salary=9100 WHERE employee_id=1234 # DML

ROLLBACK TO SAVEPOINT before_salary_update;

UPDATE employees SET salary=9200 WHERE employee_id=1234 # DML

COMMIT COMMENT 'Updated salaries';
```

- "事务简介"
- "可延迟约束"
- *Oracle 数据库 SQL 语言参考*

会话控制语句

会话控制语句动态地管理用户会话的属性。如"连接和会话"中所述，会话是数据库实例内存中的一个逻辑实体，表示登录到数据库中的一个当前用户的状态。会话从该用户通过数据库验证开始，一直持续到用户断开连接或退出数据库应用程序。

会话控制语句使您能够：

- 通过执行一项专门的功能来改变当前会话，如启用和禁用 SQL 跟踪（`ALTER SESSION`）。
- 启用和禁用角色，它是一组用于当前会话的权限（`SET ROLE`）。

下面的示例为当前会话打开 SQL 跟踪，然后在当前会话中启用除 dw_manager 之外的所有已授予的角色：

```
ALTER SESSION SET SQL_TRACE = TRUE;

SET ROLE ALL EXCEPT dw_manager;
```

会话控制语句不会隐式提交当前事务。

另见：

Oracle Database SQL Language Reference for ALTER SESSION syntax and semantics

System Control Statement

System control statements change the properties of the database instance. The only system control statement is ALTER SYSTEM. It enables you to change settings such as the minimum number of shared servers, terminate a session, and perform other system-level tasks. Following are examples of system control statements:

```
ALTER SYSTEM SWITCH LOGFILE;

ALTER SYSTEM KILL SESSION '39, 23';
```

The ALTER SYSTEM statement does not implicitly commit the current transaction.

See Also:

Oracle Database SQL Language Reference for ALTER SYSTEM syntax and semantics

Embedded SQL Statements

Embedded SQL statements incorporate DDL, DML, and transaction control statements within a procedural language program. They are used with the Oracle precompilers. Embedded SQL is one approach to incorporating SQL in your procedural language applications. Another approach is to use a procedural API such as Open Database Connectivity (ODBC) or Java Database Connectivity (JDBC).

Embedded SQL statements enable you to:

- Define, allocate, and release cursors (DECLARE CURSOR, OPEN, CLOSE).
- Specify a database and connect to it (DECLARE DATABASE, CONNECT).
- Assign variable names (DECLARE STATEMENT).
- Initialize descriptors (DESCRIBE).
- Specify how error and warning conditions are handled (WHENEVER).
- Parse and run SQL statements (PREPARE, EXECUTE, EXECUTE

《*Oracle 数据库 SQL 语言参考*》关于 ALTER SESSION 的语法和语义

系统控制语句

系统控制语句更改数据库实例的属性。唯一的系统控制语句是 ALTER SYSTEM。它使您能够更改系统设置，例如，共享服务器的最小数目、 终止一个会话、和执行其他系统级任务。

下面是系统控制语句的示例：

```
ALTER SYSTEM SWITCH LOGFILE;

ALTER SYSTEM KILL SESSION '39, 23';
```

ALTER SYSTEM 语句不会隐式提交当前事务。

另见：

《*Oracle 数据库 SQL 语言参考*》关于 ALTER SYSTEM 的语法和语义

嵌入式的 SQL 语句

嵌入式的 SQL 语句将 DDL、 DML 、和事务控制语句混入过程化语言程序中。他们和 Oracle 预编译器一起使用。嵌入式的 SQL 是一种在您的过程化语言应用程序中纳入 SQL 的方法。另一种方法是使用一个程序 API，如开放式数据库连接 (ODBC)或 Java 数据库连接 (JDBC) 。

嵌入式的 SQL 语句，使您可以：

- 定义、 分配、和释放游标 （DECLARE CURSOR, OPEN, CLOSE）。
- 指定一个数据库，并连接到该数据库（DECLARE DATABASE, CONNECT）。
- 分配变量名称 （DECLARE STATEMENT）。
- 初始化描述符(DESCRIBE)
- 指定如何处理错误和警告(WHENEVER)。
- 分析并运行 SQL 语句 （PREPARE, EXECUTE, EXECUTE

IMMEDIATE).

- Retrieve data from the database (FETCH).

See Also:
"Introduction to Server-Side Programming" and "Client-Side APIs"

Overview of the Optimizer

To understand how Oracle Database processes SQL statements, it is necessary to understand the part of the database called the **optimizer** (also known as the **query optimizer** or **cost-based optimizer**). All SQL statements use the optimizer to determine the most efficient means of accessing the specified data.

Use of the Optimizer

To execute a DML statement, Oracle Database may have to perform many steps. Each step either retrieves rows of data physically from the database or prepares them for the user issuing the statement.
Many different ways of processing a DML statement are often possible. For example, the order in which tables or indexes are accessed can vary. The steps that the database uses to execute a statement greatly affect how quickly the statement runs. The optimizer generates execution plans describing possible methods of execution.
The optimizer determines which execution plan is most efficient by considering several sources of information, including query conditions, available access paths, statistics gathered for the system, and hints. For any SQL statement processed by Oracle, the optimizer performs the following operations:

- Evaluation of expressions and conditions
- Inspection of integrity constraints to learn more about the data and optimize based on this metadata
- Statement transformation
- Choice of optimizer goals
- Choice of access paths
- Choice of join orders

The optimizer generates most of the possible ways of processing a query and assigns a cost to each step in the generated execution plan. The plan with the lowest cost is chosen as the query plan to be executed.
Note:

IMMEDIATE）。

- 从数据库中检索数据（FETCH）。

另见：
"服务器端编程简介"和"客户端 API"

优化器概述

若要理解数据库如何处理 SQL 语句，有必要了解一下数据库中被称为**优化器**的部件（也称为**查询优化器**或基**于成本的优化器**）。所有 SQL 语句都使用优化器来确定访问指定的数据的最有效手段。

使用优化器

要执行一个 DML 语句，数据库可能需要执行许多步骤。每一步或者是从数据库中物理地检索数据行，或者是在为发出语句的用户准备数据，等等。

许多不同的处理 DML 语句的方式通常都是可能的。例如，访问表或索引的顺序可能会不同。数据库用来执行一条语句的步骤，很大程度上会影响该语句的运行速度有多快。优化器生成执行计划来描述可能的执行方法。

优化器通过考虑几个信息来源来确定哪种执行计划是最有效的，包括查询条件、可用的访问路径、为系统收集的统计信息、以及提示等。对于由 Oracle 处理的任何语句，优化器将执行以下操作：

- 表达式和条件评估
- 检查完整性约束， 以了解数据和基于此元数据的优化的更多信息
- 语句转换
- 优化器目标选择
- 访问路径选择
- 连接顺序选择

优化器生成处理一个查询的几乎所有可能的方法，并给生成的执行计划中的每个步骤分配一个成本。具有最低成本的计划被选择为要执行的查询计划。

注意：

You can obtain an execution plan for a SQL statement without executing the plan. Only an execution plan that the database actually uses to execute a query is correctly termed a query plan.

You can influence optimizer choices by setting the optimizer goal and by gathering representative statistics for the optimizer. For example, you may set the optimizer goal to either of the following:

- Total throughput

  The `ALL_ROWS` hint instructs the optimizer to get the last row of the result to the client application as fast as possible.
- Initial response time

  The `FIRST_ROWS` hint instructs the optimizer to get the first row to the client as fast as possible.

A typical end-user, interactive application would benefit from initial response time optimization, whereas a batch-mode, non-interactive application would benefit from total throughput optimization.

See Also:

- *Oracle Database PL/SQL Packages and Types Reference* for information about using `DBMS_STATS`
- *Oracle Database Performance Tuning Guide* for more information about the optimizer and using hints

Optimizer Components

The optimizer contains three main components, which are shown in Figure 7-2.

Figure 7-2 Optimizer Components

您可以获得一个 SQL 语句的执行计划，而不一定需要执行它。只有数据库实际上用于执行一个查询的执行计划，才称为查询计划。

你可以通过设置优化器目标，并为优化程序收集有代表性的统计数据，来影响优化器的选择。例如，您可以设置优化器目标为以下之一：

- 总吞吐量

  ALL_ROWS 提示指示优化器尽可能快地将所有结果数据返回给客户端应用程序。
- 初始响应时间

  FIRST_ROWS 提示指示优化器尽可能快地获取第一行数据给客户端。

典型的交互式终端用户应用程序将受益于初始响应时间优化，而非交互式批处理模式应用程序将受益于总吞吐量的优化。

另见：

- 《*Oracle 数据库 PL/SQL 软件包和类型参考*》有关使用 DBMS_STATS 的信息
- 《*Oracle 数据库性能优化指南*》有关优化器和使用提示的详细信息

优化器组件

优化器包含三个主要组件，如图 7-2 所示。

图 7-2 优化器组件

Description of "Figure 7-2 Optimizer Components"

The input to the optimizer is a parsed query (see "SQL Parsing"). The optimizer performs the following operations:

- 1 The optimizer receives the parsed query and generates a set of potential plans for the SQL statement based on available access paths and hints.
- 2 The optimizer estimates the cost of each plan based on statistics in the data dictionary. The cost is an estimated value proportional to the expected resource use needed to execute the statement with a particular plan.
- 3 The optimizer compares the costs of plans and chooses the lowest-cost plan, known as the query plan, to pass to the row source generator (see "SQL Row Source Generation").

Query Transformer

The **query transformer** determines whether it is helpful to change the form of the query so that the optimizer can generate a better execution plan. The input to the query transformer is a parsed query, which is represented by a set of query blocks.

See Also:

"Query Rewrite"

Estimator

The **estimator** determines the overall cost of a given execution plan. The

Description of "Figure 7-2 Optimizer Components"

优化器的输入是一个已解析的查询 （请参见"SQL 分析"）。优化器将执行以下操作：

- 1 优化器接收已解析的查询，并基于可用的访问路径和提示，为 SQL 语句生成一组潜在的计划。
- 2 优化器基于数据字典中的统计信息，估计每个计划的成本。成本是一个与特定用于执行该语句的计划所需的预期资源使用成正比的估计值。
- 3 优化器比较各个计划的成本，并选择具有最低成本的计划（也叫查询计划），然后传递给行源生成器（请参见 "SQL 行源生成"）。

查询转换器

查询转换器确定更改查询的形式是否有助于优化器生成一个更好的执行计划。查询转换器的输入是一个由一组查询块表示的已分析的查询。

另见：

"查询重写"

估算器

估算器确定一个给定的执行计划的总体成本。估算器生成三种不同类型的测

estimator generates three different types of measures to achieve this goal:

- Selectivity

  This measure represents a fraction of rows from a row set. The selectivity is tied to a query predicate, such as `last_name='Smith'`, or a combination of predicates.
- Cardinality

  This measure represents the number of rows in a row set.
- Cost

  This measure represents units of work or resource used. The query optimizer uses disk I/O, CPU usage, and memory usage as units of work.

If statistics are available, then the estimator uses them to compute the measures. The statistics improve the degree of accuracy of the measures.

Plan Generator

The **plan generator** tries out different plans for a submitted query and picks the plan with the lowest cost. The optimizer generates subplans for each of the nested subqueries and unmerged views, which is represented by a separate **query block**. The plan generator explores various plans for a query block by trying out different access paths, join methods, and join orders.

The optimizer automatically manages plans and ensures that only verified plans are used. SQL Plan Management (SPM) allows controlled plan evolution by only using a new plan after it has been verified to be perform better than the current plan.

Diagnostic tools such as the `EXPLAIN PLAN` statement enable you to view execution plans chosen by the optimizer. `EXPLAIN PLAN` shows the query plan for the specified SQL query if it were executed now in the current session. Other diagnostic tools are Oracle Enterprise Manager and the SQL*Plus `AUTOTRACE` command. Example 7-6 shows the execution plan of a query when `AUTOTRACE` is enabled.

See Also:

- "Tools for Database Administrators"

量值，以实现这一目标：

- 选择性

  这项测量表示一个行集中的一小部分。受如 last_name 的选择性依赖于查询谓词（或谓词的组合），比如 `last_name='Smith'`。
- 基数

  这项测量表示行集中的行数。
- 成本

  这项测量表示工作量或使用的资源。查询优化器使用磁盘 I/O、 CPU 使用率、和内存使用情况作为工作量。

如果统计数据可用，则估算器使用它们来计算这些测量值。统计信息可以提高测量的精确程度。

计划生成器

计划生成器对提交的查询尝试不同的计划，并选出具有最低成本的计划。优化器为每个由单独**查询块**表示的嵌套子查询和未合并试图生成子计划。计划生成器通过尝试不同的访问路径、 联接方法、和联接顺序，来为查询块探究各种计划。

优化程序自动管理计划，并确保只使用经过验证的计划。SQL 计划管理 (SPM) 通过仅当一个新的计划已经过验证比当前的计划更好时才使用它，来控制计划演进。

像 `EXPLAIN PLAN` 这样的诊断工具使您能够查看优化器所选择的执行计划。`EXPLAIN PLAN` 会显示指定的 SQL 查询的查询计划，就好像它已经在当前会话中执行过一样。其他的诊断工具还有 Oracle 企业管理器和 SQL * Plus AUTOTRACE 命令。示例 7-6 显示一个查询当 AUTOTRACE 处于启用状态时的执行计划。

另见：

- "数据库管理员工具"

- *Oracle Database SQL Language Reference* to learn about `EXPLAIN PLAN`
- *Oracle Database Performance Tuning Guide* to learn about the optimizer components

Access Paths

An **access path** is the way in which data is retrieved from the database. For example, a query that uses an index has a different access path from a query that does not. In general, index access paths are best for statements that retrieve a small subset of table rows. Full scans are more efficient for accessing a large portion of a table.

The database can use several different access paths to retrieve data from a table. The following is a representative list:

- Full table scans

  This type of scan reads all rows from a table and filters out those that do not meet the selection criteria. The database sequentially scans all data blocks in the segment, including those under the high water mark that separates used from unused space (see "Segment Space and the High Water Mark").
- Rowid scans

  The rowid of a row specifies the data file and data block containing the row and the location of the row in that block. The database first obtains the rowids of the selected rows, either from the statement `WHERE` clause or through an index scan, and then locates each selected row based on its rowid.
- Index scans

  This scan searches an index for the indexed column values accessed by the SQL statement (see "Index Scans"). If the statement accesses only columns of the index, then Oracle Database reads the indexed column values directly from the index.
- Cluster scans

  A cluster scan is used to retrieve data from a table stored in an indexed table cluster, where all rows with the same cluster key value are stored in the same data block (see "Overview of Indexed Clusters"). The database first obtains the rowid of a selected row

- 《*Oracle 数据库 SQL 语言参考*》了解 `EXPLAIN PLAN`
- 《*Oracle 数据库性能优化指南*》了解优化器的组件

访问路径

访问路径是从数据库检索数据的方式。例如，使用索引的查询与不使用索引的查询具有不同的访问路径。通常，索引访问路径对于只检索表行中的一个小的子集是最佳的。而完全扫描则对访问表中的一大部分更有效。

数据库可以使用几个不同的访问路径从表中检索数据。以下是一个有代表性的列表：

- 全表扫描

  这种类型的扫描从一个表读取所有行，并滤掉那些不符合选择条件的行。数据库顺序扫描段中的所有数据块，包括那些高水位标记以下的块，高水位标记用来分隔已使用和未使用的空间 （请参阅"段空间和高水位标记"）。
- Rowid 扫描

  行的 rowid 指定包含行的数据文件和数据块，以及行在该块中的位置。数据库通过语句的 WHERE 子句或一个索引扫描，首先获取所选的行的 rowids，然后基于这些 rowid 查找每个选定的行。
- 索引扫描

  此扫描搜索被 SQL 语句访问的索引列的值（请参见"索引扫描"）。如果该语句仅访问已被索引的列，则数据库直接从索引读取索引的列值。
- 簇扫描

  簇扫描用来检索存储在一个索引化的表簇中的一个表中的数据，具有相同的簇键的所有行都存储在同一个数据块中 （请参见"索引化表簇概述"）。Oracle 数据库首先通过扫描簇索引来获取所选行的 rowid。然后基于此 rowid 查找相应行。

by scanning the cluster index. Oracle Database locates the rows based on this rowid.

- Hash scans

  A hash scan is used to locate rows in a hash cluster, where all rows with the same hash value are stored in the same data block (see "Overview of Hash Clusters". The database first obtains the hash value by applying a hash function to a cluster key value specified by the statement. Oracle Database then scans the data blocks containing rows with this hash value.

The optimizer chooses an access path based on the available access paths for the statement and the estimated cost of using each access path or combination of paths.

See Also:

Oracle Database 2 Day + Performance Tuning Guide and *Oracle Database Performance Tuning Guide* to learn about access paths

Optimizer Statistics

Optimizer **statistics** are a collection of data that describe details about the database and the objects in the database. The statistics provide a statistically correct picture of data storage and distribution usable by the optimizer when evaluating access paths.

Optimizer statistics include the following:

- Table statistics

  These include the number of rows, number of blocks, and average row length.

- Column statistics

  These include the number of distinct values and nulls in a column and the distribution of data.

- Index statistics

  These include the number of leaf blocks and index levels.

- System statistics

  These include CPU and I/O performance and utilization.

Oracle Database gathers optimizer statistics on all database objects automatically and maintains these statistics as an automated maintenance

- 哈希扫描

  哈希扫描用于查找哈希群集中的行，其中具有相同哈希值的所有行都存储在同一个数据块中 （请参阅"哈希簇概述"。Oracle 数据库首先通过将哈希函数应用于由该语句指定的簇键值，以获得哈希值。然后扫描包含具有此哈希值的行的数据块。

优化器对访问路径的选择，基于语句的所有可用的访问路径，和使用每个访问路径或其组合的估算成本。

另见：

《*Oracle 数据库 2 日 + 性能调整指南*》和 《*Oracle 数据库性能优化指南*》了解访问路径

优化器统计

优化器**统计信息**是描述有关数据库和数据库中的对象的详细信息的数据集合。统计信息提供数据存储和分布的正确描述，以被优化器用来评估访问路径。

优化器统计信息包括：

- 表统计

  这包括行数、 块数、和平均行长等。

- 列统计

  这包括非重复值数目、空值数目、和数据的分布。

- 索引统计

  这包括叶块数目和索引级别。

- 系统统计

  这包括 CPU 和 I/O 的性能及利用率。

Oracle 数据库自动收集所有数据库对象的优化器统计信息，并作为一项自动维护任务来维护这些统计信息。您还可以使用 DBMS_STATS 包手动收集

task. You can also gather statistics manually using the DBMS_STATS package. This PL/SQL package can modify, view, export, import, and delete statistics.

Optimizer statistics are created for the purposes of query optimization and are stored in the data dictionary. These statistics should not be confused with performance statistics visible through dynamic performance views.

See Also:

- *Oracle Database 2 Day + Performance Tuning Guide* and *Oracle Database Performance Tuning Guide* to learn how to gather and manage statistics
- *Oracle Database PL/SQL Packages and Types Reference* to learn about DBMS_STATS

Optimizer Hints

A **hint** is a comment in a SQL statement that acts as an instruction to the optimizer. Sometimes the application designer, who has more information about a particular application's data than is available to the optimizer, can choose a more effective way to run a SQL statement. The application designer can use hints in SQL statements to specify how the statement should be run.

For example, suppose that your interactive application runs a query that returns 50 rows. This application initially fetches only the first 25 rows of the query to present to the end user. You want the optimizer to generate a plan that gets the first 25 records as quickly as possible so that the user is not forced to wait. You can use a hint to pass this instruction to the optimizer as shown in the SELECT statement and AUTOTRACE output in Example 7-4.

Example 7-4 Execution Plan for SELECT with FIRST_ROWS Hint

```
SELECT /*+ FIRST_ROWS(25) */ employee_id, department_id
FROM   hr.employees
WHERE  department_id > 50;
-----------------------------------------------------------------------
| Id | Operation                    | Name              | Rows | Bytes
-----------------------------------------------------------------------
|  0 | SELECT STATEMENT             |                   | 26   | 182
|  1 |  TABLE ACCESS BY INDEX ROWID | EMPLOYEES         | 26   | 182
|* 2 |   INDEX RANGE SCAN           | EMP_DEPARTMENT_IX |      |
-----------------------------------------------------------------------
```

The execution plan in Example 7-4 shows that the optimizer chooses an index on the employees.department_id column to find the first 25

统计信息。该 PL/SQL 包可以修改、 查看、 导出、 导入、和删除统计信息。

优化器统计信息被创建来用于查询优化的目的，并存储在数据字典中。这些统计数字不应与通过动态性能视图看到的性能统计信息相混淆。

另见：

- 《*Oracle 数据库 2 日 + 性能调整指南*》 和 《*Oracle 数据库性能调整指南*》了解如何收集和管理统计信息
- 《*Oracle 数据库 PL/SQL 包和类型参考*》了解 DBMS_STATS 包

优化器提示

提示是 SQL 语句中的注释，作为优化程序的一个指示。有时应用程序的设计者比优化器更了解一个特定的应用程序的数据的详细信息，他可以选择一个运行 SQL 语句的更有效方法。应用程序设计者可以在 SQL 语句中使用提示，来指定该语句应该如何运行。

例如，假设你的交互式应用程序运行一个查询，返回 50 行。此应用程序最初只读取查询的前 25 行来呈现给终端用户。你想使优化器生成一个计划，尽可能快地获取首批 25 条记录，以使用户不必被迫等待。您可以使用一个提示来将此指令传递给优化器，参阅示例 7-4 中的 SELECT 语句和 AUTOTRACE 输出，如下所示。

示例 7-4 带有 FIRST_ROWS 提示的 SELECT 语句的执行计划

```
SELECT /*+ FIRST_ROWS(25) */ employee_id, department_id
FROM   hr.employees
WHERE  department_id > 50;
-----------------------------------------------------------------------
| Id | Operation                    | Name              | Rows | Bytes
-----------------------------------------------------------------------
|  0 | SELECT STATEMENT             |                   | 26   | 182
|  1 |  TABLE ACCESS BY INDEX ROWID | EMPLOYEES         | 26   | 182
|* 2 |   INDEX RANGE SCAN           | EMP_DEPARTMENT_IX |      |
-----------------------------------------------------------------------
```

示例 7-4 中的执行计划显示，优化器选择 employees.department_id 列上的一个索引，来查找其部门 ID 超过 50 的雇员中的前 25 行。优化器使用从索

rows of employees whose department ID is over 50. The optimizer uses the rowid retrieved from the index to retrieve the record from the employees table and return it to the client. Retrieval of the first record is typically almost instantaneous.

Example 7-5 shows the same statement, but without the optimizer hint.

Example 7-5 Execution Plan for SELECT with No Hint

```
SELECT employee_id, department_id
FROM   hr.employees
WHERE  department_id > 50;
```

```
-----------------------------------------------------------------------
| Id | Operation              | Name              | Rows | Bytes | Cos
-----------------------------------------------------------------------
|  0 | SELECT STATEMENT       |                   | 50   | 350   |
|* 1 |  VIEW                  | index$_join$_001  | 50   | 350   |
|* 2 |   HASH JOIN            |                   |      |       |
|* 3 |    INDEX RANGE SCAN    | EMP_DEPARTMENT_IX | 50   | 350   |
|  4 |    INDEX FAST FULL SCAN| EMP_EMP_ID_PK     | 50   | 350   |
```

The execution plan in Example 7-5 joins two indexes to return the requested records as fast as possible. Rather than repeatedly going from index to table as in Example 7-4, the optimizer chooses a range scan of EMP_DEPARTMENT_IX to find all rows where the department ID is over 50 and place these rows in a hash table. The optimizer then chooses to read the EMP_EMP_ID_PK index. For each row in this index, it probes the hash table to find the department ID.

In this case, the database cannot return the first row to the client until the index range scan of EMP_DEPARTMENT_IX completes. Thus, this generated plan would take longer to return the first record. Unlike the plan in Example 7-4, which accesses the table by index rowid, the plan in Example 7-5 uses multiblock I/O, resulting in large reads. The reads enable the last row of the entire result set to be returned more rapidly.

See Also:

Oracle Database Performance Tuning Guide to learn how to use optimizer hints

Overview of SQL Processing

This section explains how Oracle Database processes SQL statements. Specifically, the section explains the way in which the database processes DDL statements to create objects, DML to modify data, and queries to retrieve data.

引中检索到的 rowid， 从雇员表中检索相应记录，并将其返回给客户端。第一条记录的检索通常几乎是在瞬间即可完成的。

示例 7-5 显示了相同的语句，但未使用优化器提示。

示例 7-5 无提示 SELECT 语句的执行计划

```
SELECT employee_id, department_id
FROM   hr.employees
WHERE  department_id > 50;
```

```
-----------------------------------------------------------------------
| Id | Operation              | Name              | Rows | Bytes | Cos
-----------------------------------------------------------------------
|  0 | SELECT STATEMENT       |                   | 50   | 350   |
|* 1 |  VIEW                  | index$_join$_001  | 50   | 350   |
|* 2 |   HASH JOIN            |                   |      |       |
|* 3 |    INDEX RANGE SCAN    | EMP_DEPARTMENT_IX | 50   | 350   |
|  4 |    INDEX FAST FULL SCAN| EMP_EMP_ID_PK     | 50   | 350   |
```

例 7-5 中的执行计划，将两个索引联接以尽可能快的返回请求的记录。优化器并不像示例 7-4 那样多次在表和索引间倒腾，而是在 EMP_DEPARTMENT_IX 索引上使用范围扫描，找出所有部门 ID 超过 50 的行，并将这些行放在一个哈希表中。然后优化器读取 EMP_EMP_ID_PK 索引。对该索引中的每一行，它探测一次该哈希表，以查找相应的部门 id。

在这种情况下，数据库不能在完成对 EMP_DEPARTMENT_IX 索引的范围扫描之前向客户端返回第一行。因此，此生成的计划将需要更长的时间返回第一条记录。与示例 7-4 中按索引 rowid 访问表的计划不同，示例 7-5 的计划使用多数据块 I/O，导致大量读取操作。这种读取使得整个结果集的最后一行会更快地返回。

另见：

《*Oracle 数据库性能优化指南*》了解如何使用优化器提示

SQL 处理的概述

本部分说明了数据库如何处理 SQL 语句。具体而言，本部分说明了数据库处理创建对象的 DDL 语句、修改数据的 DML 语句、和检索数据的查询语句等的处理方式。

Stages of SQL Processing

Figure 7-3 depicts the general stages of SQL processing: parsing, optimization, row source generation, and execution. Depending on the statement, the database may omit some of these steps.

Figure 7-3 Stages of SQL Processing

Description of "Figure 7-3 Stages of SQL Processing"

SQL Parsing

As shown in Figure 7-3, the first stage of SQL processing is **parsing**. This stage involves separating the pieces of a SQL statement into a data structure that can be processed by other routines. The database parses a statement when instructed by the application, which means that only the application, and not the database itself, can reduce the number of parses. When an application issues a SQL statement, the application makes a **parse call** to the database to prepare the statement for execution. The

SQL 处理的阶段

图 7-3 显示了 SQL 处理的一般阶段： 解析、 优化、 产生行源、和执行。数据库可能会忽略某些步骤，这取决于具体的语句。

图 7-3 阶段的 SQL 处理

Description of "Figure 7-3 Stages of SQL Processing"

SQL 解析

如图 7-3 ， SQL 处理的第一阶段是**解析**。这一阶段涉及将 SQL 语句的各个片断分离到一个可由其他例程处理的数据结构。数据库会在应用程序的指示下，对一个语句进行解析，这意味着只有应用程序可以减少解析数目，而不是数据库本身。

当应用程序发出 SQL 语句时，该应用程序向数据库发出一个**解析调用**，以准备执行该语句。解析调用会打开或创建一个游标，它是一个对特定于会话

parse call opens or creates a cursor, which is a handle for the session-specific private SQL area that holds a parsed SQL statement and other processing information. The cursor and private SQL area are in the PGA. During the parse call, the database performs the following checks:

- Syntax Check
- Semantic Check
- Shared Pool Check

The preceding checks identify the errors that can be found *before statement execution*. Some errors cannot be caught by parsing. For example, the database can encounter deadlocks or errors in data conversion only during statement execution (see "Locks and Deadlocks").

Syntax Check

Oracle Database must check each SQL statement for syntactic validity. A statement that breaks a rule for well-formed SQL syntax fails the check. For example, the following statement fails because the keyword FROM is misspelled as FORM:

```
SQL> SELECT * FORM employees;
SELECT * FORM employees
         *
ERROR at line 1:
ORA-00923: FROM keyword not found where expected
```

Semantic Check

The semantics of a statement are its meaning. Thus, a semantic check determines whether a statement is meaningful, for example, whether the objects and columns in the statement exist. A syntactically correct statement can fail a semantic check, as shown in the following example of a query of a nonexistent table:

```
SQL> SELECT * FROM nonexistent_table;
SELECT * FROM nonexistent_table
              *
ERROR at line 1:
ORA-00942: table or view does not exist
```

Shared Pool Check

During the parse, the database performs a **shared pool check** to determine whether it can skip resource-intensive steps of statement processing. To this end, the database uses a hashing algorithm to generate a hash value for every SQL statement. The statement hash value

的私有 SQL 区的句柄，其中包含了已分析的 SQL 语句和其他处理信息。游标和 私有 SQL 区位于 PGA 中。

在解析调用期间，数据库会执行以下检查：

- 语法检查
- 语义检查
- 共享池检查

前面的检查确定在语句执行之前可以发现的错误。一些错误不能通过解析来捕获。例如，数据库在数据转换过程中可能会遇到死锁或错误，但这仅在语句执行中才会发生（见"死锁和锁"）。

语法检查

Oracle 数据库必须检查每个 SQL 语句的语法有效性。违反了标准格式的 SQL 语法规则的语句无法通过检查。例如，下面的语句会失败，因为 FROM 关键字被错误地拼写为 FORM：

```
SQL> SELECT * FORM employees;
SELECT * FORM employees
         *
ERROR at line 1:
ORA-00923: FROM keyword not found where expected
```

语义检查

语句的语义即是它的含义。因此，语义检查确定一条语句是否是有意义的，例如，该语句中的对象和列是否存在。语法正确的语句可能通不过语义检查，如下例所示，查询一个不存在的表：

```
SQL> SELECT * FROM nonexistent_table;
SELECT * FROM nonexistent_table
              *
ERROR at line 1:
ORA-00942: table or view does not exist
```

共享池检查

在解析期间，数据库执行一个**共享池检查**，以确定是否可以跳过占用大量资源的语句处理步骤。为此，数据库使用一种哈希算法为每个 SQL 语句生成一个哈希值。语句的哈希值即是在 V$SQL.SQL_ID 中显示的 **SQL ID**。

is the **SQL ID** shown in `V$SQL.SQL_ID`.

When a user submits a SQL statement, the database searches the shared SQL area to see if an existing parsed statement has the same hash value. The hash value of a SQL statement is distinct from the following values:

- Memory address for the statement

  Oracle Database uses the SQL ID to perform a keyed read in a lookup table. In this way, the database obtains possible memory addresses of the statement.
- Hash value of an execution plan for the statement

  A SQL statement can have multiple plans in the shared pool. Each plan has a different hash value. If the same SQL ID has multiple plan hash values, then the database knows that multiple plans exist for this SQL ID.

Parse operations fall into the following categories, depending on the type of statement submitted and the result of the hash check:

- Hard parse

  If Oracle Database cannot reuse existing code, then it must build a new executable version of the application code. This operation is known as a **hard parse**, or a **library cache miss**. The database always perform a hard parse of DDL.
  During the hard parse, the database accesses the library cache and data dictionary cache numerous times to check the data dictionary. When the database accesses these areas, it uses a serialization device called a latch on required objects so that their definition does not change (see "Latches"). Latch contention increases statement execution time and decreases concurrency.
- Soft parse

  A **soft parse** is any parse that is not a hard parse. If the submitted statement is the same as a reusable SQL statement in the shared pool, then Oracle Database reuses the existing code. This reuse of code is also called a **library cache hit**.
  Soft parses can vary in the amount of work they perform. For example, configuring the session shared SQL area can sometimes reduce the amount of latching in the soft parses, making them

当用户提交一个 SQL 语句时，数据库搜索共享 SQL 区，以查看是否已经有一个现成的已分析的语句具有相同的哈希值。SQL 语句的哈希值有别于下列值：

- 该语句的内存地址

  Oracle 数据库使用 SQL ID 在一个查找表中执行一个键值读取。通过这种方式，数据库获取该语句的所有可能的内存地址。

- 该语句的执行计划的哈希值

  SQL 语句在共享池中可以有多个计划。每个计划都具有不同的哈希值。如果一个相同的 SQL ID 有多个计划哈希值，则数据库知道针对该 SQL ID 有多个计划存在。

基于所提交语句的类型和哈希检查的结果，解析操作分为以下类别：

- 硬解析

  如果数据库不能重用现有代码，则它必须生成应用程序代码的一个新的可执行版本。此操作称为一个**硬解析**，或**库缓存未命中**。数据库对 DDL 始终执行硬分析。

  在硬解析期间，数据库多次访问库缓存和数据字典缓存以检查数据字典。当数据库访问这些区域时，它在所需对象上使用一个叫做闩锁的串行化设备，以便它们的定义不会被更改 （请参阅"闩锁"） 。闩锁争用会增加语句的执行时间，并降低并发性。

- 软解析

  任何不是硬解析的解析都是**软解析**。如果提交的语句与在共享池中的某个可重用 SQL 语句相同，则数据库将重用该现有代码。重用代码也称为**库缓存命中**。

  软解析要执行的工作量不是完全固定的。例如，配置会话的共享 SQL 区有时可以减少软解析中的闩锁用量，使它们“更软”。

"softer."

In general, a soft parse is preferable to a hard parse because the database skips the optimization and row source generation steps, proceeding straight to execution.

Figure 7-4 is a simplified representation of a shared pool check of an `UPDATE` statement in a dedicated server architecture.

Figure 7-4 Shared Pool Check

Description of "Figure 7-4 Shared Pool Check"

If a check determines that a statement in the shared pool has the same hash value, then the database performs semantic and environment checks to determine whether the statements mean the same. Identical syntax is not sufficient. For example, suppose two different users log in to the database and issue the following SQL statements:

```
CREATE TABLE my_table ( some_col INTEGER );
SELECT * FROM my_table;
```

The `SELECT` statements for the two users are syntactically identical, but two separate schema objects are named `my_table`. This semantic difference means that the second statement cannot reuse the code for the first statement.

Even if two statements are semantically identical, an environmental difference can force a hard parse. In this case, the environment is the

一般地，软解析比硬解析更可取，因为数据库可以跳过优化和行源生成步骤，而直接进入到执行阶段。

图 7-4 是在专用的服务器体系结构中，一个 UPDATE 语句的共享池检查的简化表示。

图 7-4 共享池检查

Description of "Figure 7-4 Shared Pool Check"

如果检查到共享库中有一个语句具有相同的哈希值，则数据库执行语义和环境检查，以确定其含义是否相同。相同的语法是不够的。例如，假设两个不同用户登录到数据库并发出以下 SQL 语句：

```
CREATE TABLE my_table ( some_col INTEGER );
SELECT * FROM my_table;
```

两个用户的 SELECT 语句的语法相同，但这是两个独立的模式对象，名字都是 my_table。这种语义差异意味着第二个语句不能重用第一个语句的代码。

即使两个语句在语义上是相同的，某个环境差异也可能使其强制进行硬解析。在这种情况下，环境是可以影响执行计划生成的全部会话设置，如工作

totality of session settings that can affect execution plan generation, such as the work area size or optimizer settings. Consider the following series of SQL statements executed by a single user:

```
ALTER SYSTEM FLUSH SHARED_POOL;
SELECT * FROM my_table;
ALTER SESSION SET OPTIMIZER_MODE=FIRST_ROWS;
SELECT * FROM my_table;
ALTER SESSION SET SQL_TRACE=TRUE;
SELECT * FROM my_table;
```

In the preceding example, the same `SELECT` statement is executed in three different optimizer environments. Consequently, the database creates three separate shared SQL areas for these statements and forces a hard parse of each statement.

See Also:

- "Private SQL Area" and "Shared SQL Areas"
- *Oracle Database Performance Tuning Guide* to learn how to configure the shared pool

SQL Optimization

As explained in "Overview of the Optimizer", query **optimization** is the process of choosing the most efficient means of executing a SQL statement. The database optimizes queries based on statistics collected about the actual data being accessed. The optimizer uses the number of rows, the size of the data set, and other factors to generate possible execution plans, assigning a numeric cost to each plan. The database uses the plan with the lowest cost.

The database must perform a hard parse at least once for every unique DML statement and performs optimization during this parse. DDL is never optimized unless it includes a DML component such as a subquery that requires optimization.

See Also:

Oracle Database Performance Tuning Guide for detailed information about the query optimizer

SQL Row Source Generation

The **row source generator** is software that receives the optimal execution plan from the optimizer and produces an iterative plan, called the **query plan**, that is usable by the rest of the database. The iterative plan is a binary program that, when executed by the SQL virtual machine, produces the result set.

区大小或优化器设置等。请考虑以下由单个用户执行的一系列 SQL 语句：

```
ALTER SYSTEM FLUSH SHARED_POOL;
SELECT * FROM my_table;
ALTER SESSION SET OPTIMIZER_MODE=FIRST_ROWS;
SELECT * FROM my_table;
ALTER SESSION SET SQL_TRACE=TRUE;
SELECT * FROM my_table;
```

在前面的示例中，相同的 SELECT 语句在三种不同的优化器环境中执行。因此，数据库为这些语句创建三个单独的共享 SQL 区域，并对每个语句强制进行硬解析。

另见：

- "私有 SQL 区"和"共享 SQL 区"
- 《*Oracle 数据库性能优化指南*》了解如何配置共享池

SQL 优化

如"优化器概述"中所述，查询**优化**是选择执行 SQL 语句的最有效手段的过程。数据库对查询的优化基于对正在访问的实际数据收集的统计信息。优化器使用行数、数据集大小、和其他因素，来生成各种可能的执行计划，并为每个计划分配一个成本数值。数据库会使用具有最低成本的计划。

数据库对每个唯一的 DML 语句必须至少执行一次硬解析，并在解析期间执行优化。DDL 永远不会被优化，除非它包括需要优化的 DML 组件，如子查询。

另见：

《*Oracle 数据库性能优化指南*》有关查询优化器的详细信息

SQL 行源生成

行源生成器是一种软件，它从优化器接收经过优化的执行计划，并生成一个称为**查询计划**的迭代计划，可供数据库的其余部分使用。迭代计划是一个二进制程序，由 SQL 虚拟机执行，以生成结果集。

The query plan takes the form of a combination of steps. Each step returns a **row set**. The rows in this set are either used by the next step or, in the last step, are returned to the application issuing the SQL statement.

A **row source** is a row set returned by a step in the execution plan along with a control structure that can iteratively process the rows. The row source can be a table, view, or result of a join or grouping operation. The row source generator produces a **row source tree**, which is a collection of row sources. The row source tree shows the following information:

- An ordering of the tables referenced by the statement
- An access method for each table mentioned in the statement
- A join method for tables affected by join operations in the statement
- Data operations such as filter, sort, or aggregation

Example 7-6 shows the execution plan of a `SELECT` statement when `AUTOTRACE` is enabled. The statement selects the last name, job title, and department name for all employees whose last names begin with the letter `A`. The execution plan for this statement is the output of the row source generator.

Example 7-6 Execution Plan

```
SELECT e.last_name, j.job_title, d.department_name
FROM   hr.employees e, hr.departments d, hr.jobs j
WHERE  e.department_id = d.department_id
AND    e.job_id = j.job_id
AND    e.last_name LIKE 'A%' ;

Execution Plan
----------------------------------------------------------
Plan hash value: 975837011

-----------------------------------------------------------------------------------------
| Id  | Operation                    | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |             |     3 |   189 |     7  (15)| 00:00:01 |
|*  1 |  HASH JOIN                   |             |     3 |   189 |     7  (15)| 00:00:01 |
|*  2 |   HASH JOIN                  |             |     3 |   141 |     5  (20)| 00:00:01 |
|   3 |    TABLE ACCESS BY INDEX ROWID| EMPLOYEES   |     3 |    60 |     2   (0)| 00:00:01 |
|*  4 |     INDEX RANGE SCAN         | EMP_NAME_IX |     3 |       |     1   (0)| 00:00:01 |
|   5 |    TABLE ACCESS FULL         | JOBS        |    19 |   513 |     2   (0)| 00:00:01 |
|   6 |   TABLE ACCESS FULL          | DEPARTMENTS |    27 |   432 |     2   (0)| 00:00:01 |
```

查询计划采用组合多个步骤的形式。每一步返回一个**行集**。该集合中的行可以在下一步被使用，或在最后一步返回给发出 SQL 语句的应用程序。

行源是执行计划中的某一步骤所返回的行集，且带有能够迭代该行集的控制结构。行源可以是表、 视图、或联接操作或分组操作的结果。

行源生成器产生一个**行源树**，它是一个行源的集合。行源树显示以下信息：

- 由语句所引用的多个有次序的表
- 在语句中提及的每个表的访问方法
- 在语句中受联接操作影响的各个表的联接方法
- 进行的数据操作，如筛选、 排序、或聚合等

示例 7-6 显示一个 AUTOTRACE 处于启用状态的 SELECT 语句的执行计划。该语句选择其姓氏以 A 开头的所有雇员的姓氏、 职位名称、和部门名称。此语句的执行计划是行源生成器的输出(入？)。

示例 7-6 执行计划


```
SELECT e.last_name, j.job_title, d.department_name
FROM   hr.employees e, hr.departments d, hr.jobs j
WHERE  e.department_id = d.department_id
AND    e.job_id = j.job_id
AND    e.last_name LIKE 'A%' ;

Execution Plan
----------------------------------------------------------
Plan hash value: 975837011

-----------------------------------------------------------------------------------------
| Id  | Operation                    | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |             |     3 |   189 |     7  (15)| 00:00:01 |
|*  1 |  HASH JOIN                   |             |     3 |   189 |     7  (15)| 00:00:01 |
|*  2 |   HASH JOIN                  |             |     3 |   141 |     5  (20)| 00:00:01 |
|   3 |    TABLE ACCESS BY INDEX ROWID| EMPLOYEES   |     3 |    60 |     2   (0)| 00:00:01 |
|*  4 |     INDEX RANGE SCAN         | EMP_NAME_IX |     3 |       |     1   (0)| 00:00:01 |
|   5 |    TABLE ACCESS FULL         | JOBS        |    19 |   513 |     2   (0)| 00:00:01 |
|   6 |   TABLE ACCESS FULL          | DEPARTMENTS |    27 |   432 |     2   (0)| 00:00:01 |
```

```
---------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
   2 - access("E"."JOB_ID"="J"."JOB_ID")
   4 - access("E"."LAST_NAME" LIKE 'A%')
       filter("E"."LAST_NAME" LIKE 'A%')
```

SQL Execution

During execution, the SQL engine executes each row source in the tree produced by the row source generator. This step is the only mandatory step in DML processing.

Figure 7-5 is an **execution tree**, also called a **parse tree**, that shows the flow of row sources from one step to another. In general, the order of the steps in execution is the *reverse* of the order in the plan, so you read the plan from the bottom up. Initial spaces in the `Operation` column indicate hierarchical relationships. For example, if the name of an operation is preceded by two spaces, then this operation is a child of an operation preceded by one space. Operations preceded by one space are children of the `SELECT` statement itself.

Figure 7-5 Row Source Tree

```
---------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
   2 - access("E"."JOB_ID"="J"."JOB_ID")
   4 - access("E"."LAST_NAME" LIKE 'A%')
       filter("E"."LAST_NAME" LIKE 'A%')
```

SQL 执行

在执行期间， SQL 引擎执行行源生成器所产生的树中的每个行源。这一步是在 DML 处理中唯一的强制性步骤。

图 7-5 是一个**执行树**，也称为**解析树**，显示了行源从一步流向另一个步。通常，执行步骤的顺序与计划中顺序相反，所以你应该从底向上来阅读计划。在 `Operation` 列中的初始空格表示层次结构关系。例如，如果一个操作的名称前面有两个空格，则此操作是前面有一个空格的操作的子操作。前面有一个空格的操作是 SELECT 语句本身的子操作。

图 7-5 行源树

Description of "Figure 7-5 Row Source Tree"

In Figure 7-5, each node of the tree acts as a row source, which means that each step of the execution plan either retrieves rows from the database or accepts rows from one or more row sources as input. The SQL engine executes each row source as follows:

- Steps indicated by the black boxes physically retrieve data from an object in the database. These steps are the **access paths**, or techniques for retrieving data from the database.
  - Step 6 uses a full table scan to retrieve all rows from the `departments` table.
  - Step 5 uses a full table scan to retrieve all rows from the `jobs` table.
  - Step 4 scans the `emp_name_ix` index in order, looking for each key that begins with the letter `A` and retrieving the corresponding rowid (see "Index Range Scan"). For example,

Description of "Figure 7-5 Row Source Tree"

在图 7-5 中，树的每个节点作为一个行源，这意味着每个步骤的执行计划要么从数据库中检索行，要么接受一个或多个行源中的行作为输入。SQL 引擎这样执行每个行源，如下所示：

- 黑框所示的步骤物理地从数据库中检索对象的数据。这些步骤即是访问路径，或某种从数据库中检索数据的技术。
  - 第 6 步使用全表扫描来从 `departments` 表中检索所有行。
  - 第 5 步使用全表扫描来从 `jobs` 表中检索所有行。
  - 第 4 步顺序扫描 emp_name_ix 索引，查找以字母 A 开头的每个键，并检索相应的 rowid （请参见"索引范围扫描"）。例如，与 `Atkinson` 对应的 rowid 是

the rowid corresponding to `Atkinson` is `AAAPzRAAFAAAABSAAe`.

- Step 3 retrieves from the `employees` table the rows whose rowids were returned by Step 4. For example, the database uses rowid `AAAPzRAAFAAAABSAAe` to retrieve the row for `Atkinson`.

- Steps indicated by the clear boxes operate on row sources.

  - Step 2 performs a **hash join**, accepting row sources from Steps 3 and 5, joining each row from the Step 5 row source to its corresponding row in Step 3, and returning the resulting rows to Step 1.

    For example, the row for employee `Atkinson` is associated with the job name `Stock Clerk`.
  - Step 1 performs another hash join, accepting row sources from Steps 2 and 6, joining each row from the Step 6 source to its corresponding row in Step 2, and returning the result to the client.

    For example, the row for employee `Atkinson` is associated with the department named `Shipping`.

In some execution plans the steps are iterative and in others sequential. The plan shown in Example 7-6 is iterative because the SQL engine moves from index to table to client and then repeats the steps.

During execution, the database reads the data from disk into memory if the data is not in memory. The database also takes out any locks and latches necessary to ensure data integrity and logs any changes made during the SQL execution. The final stage of processing a SQL statement is closing the cursor.

See Also:

Oracle Database Performance Tuning Guide for detailed information about execution plans and the `EXPLAIN PLAN` statement

How Oracle Database Processes DML

Most DML statements have a query component. In a query, execution of a cursor places the results of the query into a set of rows called the **result set**.

Result set rows can be fetched either a row at a time or in groups. In the fetch stage, the database selects rows and, if requested by the query,

AAAPzRAAFAAAABSAAe 。

  - 第 3 步，从 `employees` 表中检索由第 4 步所返回的 rowids 所在的行。例如，数据库使用 rowid AAAPzRAAFAAAABSAAe 来检索 `Atkinson` 的行。

- 白框中所示的步骤操作行源。

  - 第 2 步执行一个**哈希联接**，它从第 3 步和第 5 步中接受行源，将第 5 步行源中的每一行与第 3 步中的相应行连接，并将结果行返回给第 1 步。

    例如，雇员 `Atkinson` 所在行与职位 `Stock Clerk` 相关联。
  - 第 1 步执行另一个哈希连接，从第 2 步和第 6 步接受行源，将第 6 步行源中的每一行与第 2 步中的相应行连接，并将结果返回客户端。

    例如，雇员 `Atkinson` 所在行与名为 `Shipping` 的部门相关联。

在某些执行计划中的步骤是迭代的，而其他一些则是顺序的。示例 7-6 中所示的计划是迭代的，因为 SQL 引擎多次在索引、表、客户端重复这些步骤。

在执行过程中，如果数据不在内存中，数据库则从磁盘读取数据到内存。为确保数据的完整性，数据库还取得任何必要的锁和闩锁，并为 SQL 执行过程中所做的任何更改记录日志。处理 SQL 语句的最后一个阶段是关闭游标。

另见：

《*Oracle 数据库性能优化指南*》关于执行计划和 `EXPLAIN PLAN` 语句的的详细信息

Oracle 数据库如何处理 DML

大多数 DML 语句都有一个查询组件。在一个查询中，游标执行后会将查询结果放入一个称为**结果集**的行集。

结果集中的行可以每次读取一行或一组。在读取阶段，数据库选择行，如果该查询要求排序，则将其排序。每次后续读取从结果中检索下一行，直到最

orders the rows. Each successive fetch retrieves another row of the result until the last row has been fetched.

In general, the database cannot determine for certain the number of rows to be retrieved by a query until the last row is fetched. Oracle Database retrieves the data in response to fetch calls, so that the more rows the database reads, the more work it performs. For some queries the database returns the first row as quickly as possible, whereas for others it creates the entire result set before returning the first row.

Read Consistency

In general, a query retrieves data by using the Oracle Database read consistency mechanism. This mechanism, which uses undo data to show past versions of data, guarantees that all data blocks read by a query are consistent to a single point in time.

For an example of read consistency, suppose a query must read 100 data blocks in a full table scan. The query processes the first 10 blocks while DML in a different session modifies block 75. When the first session reaches block 75, it realizes the change and uses undo data to retrieve the old, unmodified version of the data and construct a noncurrent version of block 75 in memory.

See Also:

"Multiversion Read Consistency"

Data Changes

DML statements that must change data use the read consistency mechanism to retrieve only the data that matched the search criteria when the modification began. Afterward, these statements retrieve the data blocks as they exist in their current state and make the required modifications. The database must perform other actions related to the modification of the data such as generating redo and undo data.

See Also:

"Overview of the Online Redo Log"

How Oracle Database Processes DDL

Oracle Database processes DDL differently from DML. For example, when you create a table, the database does not optimize the `CREATE TABLE` statement. Instead, Oracle Database parses the DDL statement and carries out the command.

The database process DDL differently because it is a means of defining an object in the data dictionary. Typically, Oracle Database must parse and execute many **recursive SQL** statements to execute a DDL command.

后一行已被读取。

通常，只有直到读取了最后一行，数据库才知道一个查询到底需要检索多少行数。Oracle 数据库检索数据来响应读取调用，因此数据库读取的行越多，则它执行的工作就越多。对于某些查询，数据库会尽可能快地返回第一行，而其它一些则是先创建整个结果集之后才返回第一行。

读一致性

通常，查询通过使用数据库读取一致性机制来检索数据。这一机制使用撤消数据来显示以前版本的数据，保证查询所读取的所有数据块都是单点时间一致的。

举一个读取一致性的例子，假设一个查询在一次全表扫描中必须读取 100 个数据块。该查询处理前面的 10 块，而在另一个会话中的 DML 修改了第 75 块。当第一个会话读到第 75 块时，发现数据已更改，于是就使用撤消数据来检索旧的、 未经修改的版本，并在内存中构造一个第 75 块的非当前版本。

另见：

"多版本读取一致性"

数据改变

必须更改数据的 DML 语句，使用读取一致性机制来检索，只与修改开始时的搜索条件匹配的数据。然后，这些语句检索数据块，如同他们处于当前状态，并作出必要的修改。数据库必须执行其他与数据修改有关的操作，如生成重做和撤消数据。

另见：

"在线重做日志概述"

Oracle 数据库如何处理 DDL

Oracle 数据库对 DDL 的处理不同于 DML。例如，在创建表时，数据库并不会优化 CREATE TABLE 语句。相反，数据库只是解析该 DDL 语句并执行该命令。

数据库以不同方式处理 DDL 的原因是，它是一种在数据字典中定义对象的方式。通常，为执行 DDL 命令，数据库必须解析和执行许多**递归 SQL** 语句。假设您创建一个表，如下所示：

Suppose you create a table as follows:

```
CREATE TABLE mytable (mycolumn INTEGER);
```

Typically, the database would run dozens of recursive statements to execute the preceding statement. The recursive SQL would perform actions such as the following:

- Issue a `COMMIT` before executing the `CREATE TABLE` statement
- Verify that user privileges are sufficient to create the table
- Determine which tablespace the table should reside in
- Ensure that the tablespace quota has not been exceeded
- Ensure that no object in the schema has the same name
- Insert rows that define the table into the data dictionary
- Issue a `COMMIT` if the DDL statement succeeded or a `ROLLBACK` if it did not

See Also:

Oracle Database Advanced Application Developer's Guide to learn about processing DDL, transaction control, and other types of statements

```
CREATE TABLE mytable (mycolumn INTEGER);
```

通常，数据库将运行数十个递归语句来执行前面的语句。递归 SQL 会执行以下操作：

- 在执行 CREATE TABLE 语句之前，发出 一个 COMMIT 命令
- 验证用户权限足以创建表
- 确定表应位于的表空间
- 确保不超过表空间配额
- 确保在模式中没有具有相同的名称的对象
- 将定义表的行插入到数据字典
- 如果 DDL 语句成功，发出 一个 COMMIT，或者如果未成功，发出一个 ROLLBACK

另见：

《*Oracle 数据库高级应用开发指南*》了解处理 DDL、 事务控制、和其他类型的语句

Part II

Oracle Data Access

This part contains the following chapters:

- Chapter 7, "SQL"
- Chapter 8, "Server-Side Programming: PL/SQL and Java"

第二部分

Oracle 数据访问

本部分包含如下各章:

- 第 7 章, "SQL"
- 第 8 章, "服务器端编程: PL/SQL 和 Java"

8 Server-Side Programming: PL/SQL and Java

Chapter 7, "SQL" explains the Structured Query Language (SQL) language and how the database processes SQL statements. This chapter explains how **Procedural Language/SQL (PL/SQL)** or Java programs stored in the database can use SQL.

This chapter includes the following topics:

- Introduction to Server-Side Programming
- Overview of PL/SQL
  - PL/SQL Subprograms
  - PL/SQL Packages
  - PL/SQL Anonymous Blocks
  - PL/SQL Language Constructs
  - PL/SQL Collections and Records
  - How PL/SQL Runs
- Overview of Java in Oracle Database
  - Overview of the Java Virtual Machine (JVM)
  - Java Programming Environment
- Overview of Triggers
  - Advantages of Triggers
  - Types of Triggers
  - Timing for Triggers
  - Creation of Triggers
  - Execution of Triggers
  - Storage of Triggers

See Also:

Chapter 7, "SQL"

Introduction to Server-Side Programming

In a nonprocedural language such as SQL, the set of data to be operated on is specified, but not the operations to be performed or the manner in which they are to be carried out. In a procedural language program, most

第 8 章 服务器端编程：PL/SQL 和 Java

第 7 章，"SQL"介绍了结构化查询语言 (SQL)，以及数据库如何处理 SQL 语句。本章介绍存储在数据库中**过程化语言（PL/SQL）** 或 Java 程序如何使用 SQL。

本章包含以下主题：

- 服务器端编程简介
- PL/SQL 概述
  - PL/SQL 子程序
  - PL/SQL 包
  - PL/SQL 匿名块
  - PL/SQL 语言结构
  - PL/SQL 集合和记录
  - PL/SQL 如何运行
- Oracle 数据库中的 Java 概述
  - Java 虚拟机 (JVM) 的概述
  - Java 编程环境
- 触发器概述
  - 触发器的优势
  - 触发器类型
  - 触发时间
  - 创建触发器
  - 执行触发器
  - 触发器存储

另见：

第 7 章, "SQL"

服务器端编程介绍

在 SQL 之类的非过程化语言中，指定的是要操作的数据集，而不是要执行什么操作或该操作应如何执行。在过程化语言程序中，大多数语句的执行取决于其前面或后面的语句及控制结构，如循环或条件分支，而这在 SQL 中

statement execution depends on previous or subsequent statements and on control structures, such as loops or conditional branches, that are not available in SQL.

For an illustration of the difference between procedural and nonprocedural languages, suppose that the following SQL statement queries the `employees` table:

```
SELECT employee_id, department_id, last_name,
salary FROM employees;
```

The preceding statement requests data, but does not apply logic to the data. However, suppose you want an application to determine whether each employee in the data set deserves a raise based on salary and department performance. A necessary condition of a raise is that the employee did not receive more than three raises in the last five years. If a raise is called for, then the application must adjust the salary and email the manager; otherwise, the application must update a report.

The problem is how procedural database applications requiring conditional logic and program flow control can use SQL. The basic development approaches are as follows:

- Use **client-side programming** to embed SQL statements in applications written in procedural languages such as C, C++, or Java

  You can place SQL statements in source code and submit it to a precompiler or Java translator before compilation. Alternatively, you can eliminate the precompilation step and use an API such as Java Database Connectivity (JDBC) or Oracle Call Interface (OCI) to enable the application to interact with the database.
- Use **server-side programming** to develop data logic that resides in the database

  An application can explicitly invoke stored subprograms (procedures and functions), written in PL/SQL (pronounced *P L sequel*) or Java. You can also create a trigger, which is named program unit that is stored in the database and invoked in response to a specified event.

This chapter explains the second approach. The principal benefit of server-side programming is that functionality built into the database can be deployed anywhere. The database and not the application determines the best way to perform tasks on a given operating system. Also, subprograms increase scalability by centralizing application processing on

是不可用的。

为演示过程化语言与非过程化语言之间的区别，假设如下的 SQL 语句查询 `employees` 表：

```
SELECT employee_id, department_id, last_name,
salary FROM employees;
```

前面的语句只是请求数据，但并不会在数据上应用逻辑。但是，假设您希望应用程序能够确定在数据集中的每个员工是否应该基于工资和部门绩效来提高待遇。加薪的必要条件是，在过去五年中，加薪没有超过三次以上的雇员。如果要加薪，应用程序必须调整工资并发送电子邮件给经理；否则，该应用程序必须更新一份报告。

问题是，需要条件逻辑和程序流控制的过程化数据库应用程序，如何才能使用 SQL。基本的开发方法如下所示：

- 使用**客户端编程**，将 SQL 语句嵌入到由过程化语言如 C、 C++、 或 Java 等编写的应用程序中

  可以将 SQL 语句置入到源代码中，并在编译之前将其提交给预编译器或 Java 翻译器。或者，您可以消除预编译步骤，而使用如 Java 数据库连接 (JDBC) 或 Oracle 调用接口 （OCI） 这样的 API ，使应用程序能够与数据库进行交互。
- 使用**服务器端编程**，开发驻留在数据库中的数据逻辑

  应用程序可以显式地调用以 PL/SQL (发音为 *P L sequel*) 或 Java 编写的存储子程序（过程和函数）。您还可以创建一个触发器，它是一个存储在数据库中的命名程序单元，在响应指定的事件时调用。

本章介绍了第二种方法。服务器端编程的主要好处，是内置于数据库的功能可以在任意位置部署。由数据库（而不是由应用程序）确定在给定的操作系统上执行任务的最佳方式。此外，通过在服务器上集中应用程序处理，子程序增加可伸缩性，使客户端能够重用代码。由于子程序调用快速而高效，仅仅一个调用就可以启动一个计算密集型的存储子程序，减少了网络流量。

the server, enabling clients to reuse code. Because subprogram calls are quick and efficient, a single call can start a compute-intensive stored subprogram, reducing network traffic.

You can use the following languages to store data logic in Oracle Database:

- PL/SQL

  PL/SQL is the Oracle Database procedural extension to SQL. PL/SQL is integrated with the database, supporting all Oracle SQL statements, functions, and data types. Applications written in database APIs can invoke PL/SQL stored subprograms and send PL/SQL code blocks to the database for execution.
- Java

  Oracle Database also provides support for developing, storing, and deploying Java applications. Java stored subprograms run in the database and are independent of programs that run in the middle tier. Java stored subprograms interface with SQL using a similar execution model to PL/SQL.

See Also:

- "Client-Side Database Programming" to learn about embedding SQL with precompilers and APIs
- *Oracle Database 2 Day Developer's Guide* for an introduction to Oracle Database application development
- *Oracle Database Advanced Application Developer's Guide* to learn how to choose a programming environment

Overview of PL/SQL

PL/SQL provides a server-side, stored procedural language that is easy-to-use, seamless with SQL, robust, portable, and secure. You can access and manipulate database data using procedural schema objects called **PL/SQL program units**.

PL/SQL program units generally are categorized as follows:

- A **subprogram** is a PL/SQL block that is stored in the database and can be called by name from an application. When you create a subprogram, the database parses the subprogram and stores its parsed representation in the database. You can declare a subprogram as a procedure or a function.

在 Oracle 数据库中可以使用以下语言来存储数据逻辑：

- PL/SQL

  PL/SQL 是 Oracle 数据库针对 SQL 的过程化扩展。PL/SQL 与数据库集成，支持所有 Oracle SQL 语句、 函数、和数据类型。由数据库 API 编写的应用程序，可以调用 PL/SQL 存储子程序，或发送 PL/SQL 代码块到数据库以被执行。
- Java

  Oracle 数据库还对开发、 存储、和部署 Java 应用程序提供支持。Java 存储子程序在数据库中运行，并独立于在中间层上运行的程序。Java 存储子程序使用与 PL/SQL 类似的运行模型来与 SQL 接口。

另见：

- "客户端数据库编程"了解如何在预编译器和 API 中嵌入 SQL
- 《*Oracle 数据库 2 日开发人员指南*》有关数据库应用程序开发的简介
- 《*Oracle 数据库高级应用开发指南*》了解如何选择一个编程环境

PL/SQL 概述

PL/SQL 提供了一种服务器端的，存储的过程化语言，它易于使用并与 SQL 无缝集成，而且健壮、可移植、且安全。您可以使用被称为 **PL/SQL 程序单元**的过程化模式对象，来访问和操作数据库数据。

PL/SQL 程序单位一般分为如下几类：

- PL/SQL **子程序**，它存储在数据库中，可以从应用程序按名称调用。当你创建一个子程序时，数据库解析该子程序，并将其已解析的表示形式存储在数据库中。您可以将一个子程序声明为一个过程或函数。

- An **anonymous block** is a PL/SQL block that appears in your application and is not named or stored in the database. In many applications, PL/SQL blocks can appear wherever SQL statements can appear.

The PL/SQL compiler and interpreter are embedded in Oracle SQL Developer, giving developers a consistent and leveraged development model on both client and server. Also, PL/SQL stored procedures can be called from several database clients, such as Pro*C, JDBC, ODBC, or OCI, and from Oracle Reports and Oracle Forms.

See Also:

- "Tools for Database Developers"

- *Oracle Database PL/SQL Language Reference* for complete information about PL/SQL, including packages

PL/SQL Subprograms

A **PL/SQL subprogram** is a named PL/SQL block that permits the caller to supply parameters that can be input only, output only, or input and output values. A subprogram solves a specific problem or performs related tasks and serves as a building block for modular, maintainable database applications.

A subprogram is either a **procedure** or a **function**. Procedures and functions are identical except that functions always return a single value to the caller, whereas procedures do not. The term **procedure** in this chapter means **procedure** *or* **function**.

See Also:

- *Pro*C/C++ Programmer's Guide* and *Pro*COBOL Programmer's Guide* to learn about stored procedures in these languages

- *Oracle Database PL/SQL Language Reference*

Advantages of PL/SQL Subprograms

As explained in "Introduction to Server-Side Programming", server-side programming has many advantages over client-side programming. PL/SQL subprograms provide the following advantages:

- Improved performance

    - The amount of information that an application must send over a network is small compared with issuing individual SQL

- PL/SQL 匿名块，它出现在您的应用程序中，没有名字，也不存储在数据库中。在很多的应用程序中 ，PL/SQL 块可以出现在任何 SQL 语句可以出现的地方。

Oracle SQL Developer 中嵌入了 PL/SQL 编译器和解释器，给与开发人员一个在客户端和服务器上一致和有效的开发模式。此外，PL/SQL 存储过程可从多种数据库客户端调用，比如 Pro*C、 JDBC、 ODBC 、或 OCI，以及 Oracle Reports 和 Oracle Forms 等。

另见：

- "数据库开发人员工具"

- 《*Oracle 数据库 PL/SQL 语言参考*》有关 PL/SQL 及其包的完整信息

PL/SQL 子程序

PL/SQL 子程序是一个命名的 PL/SQL 块，允许调用方提供参数（可以是输入参数、 输出参数、或输入输出参数值）。子程序解决某一具体问题，或执行相关的任务，并作为模块化的、 可维护的数据库应用程序的构造块。

子程序是一个**过程**或**函数**。过程和函数基本相同，区别在于函数始终向调用者返回单个值，而过程不返回任何值。在本章中的**过程**一词，可能代表**过程**或**函数**。

另见：

- 《*Pro*C/C + + 程序员指南*》和《*Pro*COBOL 程序员指南*》了解这些语言中的存储过程

- 《*Oracle 数据库 PL/SQL 语言参考*》

PL/SQL 子程序的优势

如"服务器端编程介绍"所述，服务器端编程相对于客户端编程有很多的好处。PL/SQL 子程序提供了以下优势：

- 提高性能

    - 相比发出一个个 SQL 语句，或将整个 PL/SQL 块的文本发送到数据库中，应用程序必须通过网络发送的信息的量是很小

statements or sending the text of an entire PL/SQL block to Oracle Database, because the information is sent only once and thereafter invoked when it is used.

- The compiled form of a procedure is readily available in the database, so no compilation is required at execution time.

- If the procedure is present in the shared pool of the SGA, then the database need not retrieve it from disk and can begin execution immediately.

- Memory allocation

  Because stored procedures take advantage of the shared memory capabilities of Oracle Database, it must load only a single copy of the procedure into memory for execution by multiple users. Sharing code among users results in a substantial reduction in database memory requirements for applications.

- Improved productivity

  Stored procedures increase development productivity. By designing applications around a common set of procedures, you can avoid redundant coding. For example, you can write procedures to manipulate rows in the `employees` table. Any application can call these procedures without requiring SQL statements to be rewritten. If the methods of data management change, then only the procedures must be modified, not the applications that use the procedures.
  Stored procedures are perhaps the best way to achieve code reuse. Because any client application written in any language that connects to the database can invoke stored procedures, they provide maximum code reuse in all environments.

- Integrity

  Stored procedures improve the integrity and consistency of your applications. By developing applications around a common group of procedures, you reduce the likelihood of coding errors.
  For example, you can test a subprogram to guarantee that it returns an accurate result and, after it is verified, reuse it in any number of applications without retesting. If the data structures

的，因为信息只发送一次，并随后在使用时才调用它。

- 已编译形式的过程在数据库中是随时可用的，所以在执行时不需要编译。

- 如果该过程已在 SGA 的共享池中，则数据库不需要从磁盘检索它，就可以立即开始执行。

- 内存分配

  因为存储过程利用了数据库的共享内存功能，由对多个用户执行的同一过程，数据库只须将该过程加载一次到内存中。在多个用户间共享代码，使应用程序对数据库内存的要求大幅减少。

- 提高生产率

  存储过程增加了开发生产率。通过围绕一组通用的过程来设计应用程序，您可以避免冗余编码。例如，您可以编写过程，以处理 `employees` 表中的行。任何应用程序可以调用这些过程，而无需重写 SQL 语句。如果数据管理方法改变了，则只须修改过程，而不必修改使用过程的应用程序。

  存储过程也许是实现代码重用的最好方式。因为连接到数据库的任何客户端应用程序，无论它是用何种语言编写的，都可以调用存储过程，存储过程提供了在所有环境中最大的代码重用。

- 完整性

  存储过程能改进您的应用程序完整性和一致性。通过围绕一组公用的过程开发应用程序，您可以减少编码错误的可能性。

  例如，您可以测试一个子程序，来保证它将返回一个准确的结果，验证后，可以在任意数量的应用程序中重用它，而无需重新测试。如果该过程所引用的数据结构改变了，则只须重新编译该过程。调用该过

referenced by the procedure are altered, then you must only recompile the procedure. Applications that call the procedure do not necessarily require modifications.

- Security with definer's rights procedures

  Stored procedures can help enforce data security (see "Overview of Database Security"). A **definer's rights procedure** executes with the privileges of its owner, not its current user. Thus, you can restrict the database operations that users perform by allowing them to access data only through procedures and functions that run with the definer's privileges.
  For example, you can grant users access to a procedure that updates a table but not grant access to the table itself. When a user invokes the procedure, it runs with the privileges of its owner. Users who have only the privilege to run the procedure (but not privileges to query, update, or delete from the underlying tables) can invoke the procedure but not manipulate table data in any other way.
- Inherited privileges and schema context with invoker's rights procedures
  An **invoker's rights procedure** executes in the current user's schema with the current user's privileges. In other words, an invoker's rights procedure is not tied to a particular user or schema. Invoker's rights procedures make it easy for application developers to centralize application logic, even when the underlying data is divided among user schemas.
  For example, an `hr_manager` user who runs an update procedure on the `hr.employees` table can update salaries, whereas an `hr_clerk` who runs the same procedure is restricted to updating address data.

See Also:

- *Oracle Database PL/SQL Language Reference* for an overview of PL/SQL subprograms
- *Oracle Database Security Guide* to learn more about definer's and invoker's rights

Creation of PL/SQL Subprograms

A subprogram created at the schema level with the `CREATE PROCEDURE` or `CREATE FUNCTION` statement is a **standalone stored**

程的应用程序不需要修改。

- 定义者权限程序的安全性

  存储过程有助于强制执行数据安全 （请参见"数据库安全性概述"）。**定义者权限过程**按其所有者的权限来执行，而不是按当前用户的权限。因此，您可以通过让用户只通过按定义者权限运行的过程和函数来访问数据，以限制用户执行的数据库操作。

  例如，您可以只授予用户访问更新表的过程的权限，而不授予其访问表本身的权限。当用户调用该过程时，它按其所有者的权限运行。只有权运行该过程（但对其基础表不具有查询、 更新、或删除权限）的用户可以调用该过程，但不能以除此之外的任何其他方式操作表数据。

- 继承权限与调用者权限过程的模式上下文

  调用者权限过程以当前用户的权限在当前用户的模式中执行。换句话说，调用者权限过程不依赖于特定用户或模式。调用者权限过程调用使应用程序开发人员更易于集中处理应用程序逻辑，但是其基础数据可能会分散到多个用户模式。

  例如，一个 hr_manager 用户可以运行更新过程来更新在 hr.employees 表中的薪水，而一个 hr_clerk 用户运行相同的过程却只能更新地址数据。

另见：

- 《*Oracle 数据库 PL/SQL 语言参考*》关于 PL/SQL 子程序的概述
- 《*Oracle 数据库安全指南*》了解更多有关定义者和调用者权限

PL/SQL 子程序的创建

在模式级别使用 `CREATE PROCEDURE` 或 `CREATE FUNCTION` 语句创建的一个子程序叫做**独立存储子程序**。在一个包中定义的子程序称为**包子程**

subprogram. Subprograms defined in a package are called **package subprograms** and are considered a part of the package. The database stores subprograms in the data dictionary as schema objects.
A subprogram has a specification, which includes descriptions of any parameters, and a body. Example 8-1 shows part of a creation statement for the standalone PL/SQL procedure `hire_employees`. The procedure inserts a row into the `employees` table.

Example 8-1 PL/SQL Procedure

```
CREATE PROCEDURE hire_employees IS
  (p_last_name VARCHAR2, p_job_id VARCHAR2,
p_manager_id NUMBER, p_hire_date DATE,
   p_salary NUMBER, p_commission_pct NUMBER,
p_department_id NUMBER)

BEGIN
.
.
.
  INSERT INTO employees (employee_id, last_name,
job_id, manager_id, hire_date,
    salary, commission_pct, department_id)
  VALUES (emp_sequence.NEXTVAL, p_last_name,
p_job_id, p_manager_id, p_hire_date,
    p_salary, p_commission_pct, p_department_id);
.
.
.
END;
```

See Also:

- *Oracle Database 2 Day Developer's Guide* to learn how to create subprograms
- *Oracle Database PL/SQL Language Reference* to learn about the `CREATE PROCEDURE` command

Execution of PL/SQL Subprograms

Users can execute a subprogram interactively by:

- Using an Oracle tool, such as SQL*Plus or SQL Developer (see "Tools for Database Developers")

序，并被认为是包的一部分。数据库将子程序作为模式对象存储在数据字典中。

每个子程序有一个规范，包括一些参数说明和一个程序体。例如，示例 8-1 显示了创建独立 PL/SQL 过程 hire_employees 的语句的一部分。该过程向 `employees` 表中插入一行。

示例 8 1 PL/SQL 过程

```
CREATE PROCEDURE hire_employees IS
  (p_last_name VARCHAR2, p_job_id VARCHAR2,
p_manager_id NUMBER, p_hire_date DATE,
   p_salary NUMBER, p_commission_pct NUMBER,
p_department_id NUMBER)

BEGIN
.
.
.
  INSERT INTO employees (employee_id, last_name,
job_id, manager_id, hire_date,
    salary, commission_pct, department_id)
  VALUES (emp_sequence.NEXTVAL, p_last_name,
p_job_id, p_manager_id, p_hire_date,
    p_salary, p_commission_pct, p_department_id);
.
.
.
END;
```

另见：

- 《*Oracle 数据库 2 日开发人员指南*》 了解如何创建子程序
- 《*Oracle 数据库 PL/SQL 语言参考*》了解 `CREATE PROCEDURE` 命令

PL/SQL 子程序的执行

用户可以通过以下方式交互地执行一个子程序：

- 使用 Oracle 工具，如 SQL * Plus 或 SQL Developer （请参见"数据

- Calling it explicitly in the code of a database application, such as an Oracle Forms or precompiler application (see "Client-Side Database Programming")
- Calling it explicitly in the code of another procedure or trigger

Figure 8-1 shows different database applications calling `hire_employees`.

Figure 8-1 Calling a PL/SQL Stored Procedure

Description of "Figure 8-1 Calling a PL/SQL Stored Procedure"

Alternatively, a privileged user can use Oracle Enterprise Manager or SQL*Plus to run the `hire_employees` procedure using a statement such as the following:

```
EXECUTE hire_employees ('SMITH', 'CLERK', 1037,
SYSDATE, 500, NULL, 20);
```

The preceding statement inserts a new record for `TSMITH` in the `employees` table.

库开发人员的工具"）

- 在一个数据库应用程序（如 Oracle Forms 或预编译器应用程序）代码中显式调用它 （请参见“客户端数据库编程”）
- 在另一个过程或触发器的代码中显式调用它

图 8-1 显示了调用 hire_employees 的不同数据库应用程序。

图 8-1 调用 PL/SQL 存储过程

Description of "Figure 8-1 Calling a PL/SQL Stored Procedure"

另外，有权限的用户可以使用 Oracle 企业管理器或 SQL*Plus 使用如下的语句来运行 hire_employees 过程：

```
EXECUTE hire_employees ('SMITH', 'CLERK', 1037,
SYSDATE, 500, NULL, 20);
```

前面的语句在 `employees` 表中为 TSMITH 插入一条新记录。

A stored procedure depends on the objects referenced in its body. The database automatically tracks and manages these dependencies. For example, if you alter the definition of the `employees` table referenced by the `hire_employees` procedure in a manner that would affect this procedure, then the procedure must be recompiled to validate that it still works as designed. Usually, the database automatically administers such dependency management.

See Also:

- *Oracle Database PL/SQL Language Reference* to learn how to use PL/SQL subprograms
- *SQL*Plus User's Guide and Reference* to learn about the `EXECUTE` command

PL/SQL Packages

A **PL/SQL package** is a group of related subprograms, along with the cursors and variables they use, stored together in the database for continued use as a unit. Packaged subprograms can be called explicitly by applications or users.

Oracle Database includes many **supplied packages** that extend database functionality and provide PL/SQL access to SQL features. For example, the `UTL_HTTP` package enables HTTP callouts from PL/SQL and SQL to access data on the Internet or to call Oracle Web Server Cartridges. You can use the supplied packages when creating applications or as a source of ideas when creating your own stored procedures.

Advantages of PL/SQL Packages

PL/SQL packages provide the following advantages:

- Encapsulation

  Packages enable you to **encapsulate** or group stored procedures, variables, data types, and so on in a named, stored unit. Encapsulation provides better organization during development and also more flexibility. You can create specifications and reference public procedures without actually creating the package body. Encapsulation simplifies privilege management. Granting the privilege for a package makes package constructs accessible to the grantee.
- Data security

  The methods of package definition enable you to specify which

存储过程依赖于在其主体中引用的对象。数据库自动跟踪和管理这些依赖。例如，`hire_employees` 过程引用了 `employees` 表，如果您更改了此表的定义且影响到该过程，则该过程必须重新编译以验证它仍然工作正常。通常，数据库自动管理这种依赖关系。

另见：

- 《*Oracle 数据库 PL/SQL 语言参考*》了解如何使用 PL/SQL 子程序
- 《*SQL*Plus 用户指南和参考*》了解 EXECUTE 命令

PL/SQL 包

PL/SQL 包是一组相关子程序，与其使用的游标和变量一起，作为一个单元存储在数据库中。打包的子程序可以由应用程序或用户显式调用。

Oracle 数据库包括许多原厂商**提供的软件包**，扩展了数据库的功能，并提供对 SQL 功能的 PL/SQL 访问。例如，UTL_HTTP 包使你能够从 PL/SQL 和 SQL 访问发出 HTTP 调用，以访问在因特网上的数据，或调用 Oracle Web Server 模块。你可以在创建应用程序时使用这些原厂商提供的软件包，或当创建您自己的存储过程时将其作为思维来源。

PL/SQL 包的优势

PL/SQL 软件包提供了以下优点：

- 封装

  包使您能够**封装**或分组存储过程、 变量、数据类型等等，并存储为命名单元。封装为开发过程提供了更好地组织形式，也提供了更大的灵活性。您可以创建规范，只引用公共过程而无需实际创建包体。封装简化了权限管理。对包进行授权，使其结构元素可以由被授权者访问。
- 数据安全

  定义包的方法使您能够指定哪些变量、游标过程是公共的或私有的。

variables, cursors, and procedures are public and private. Public means that it is directly accessible to the user of a package. Private means that it is hidden from the user of a package.
For example, a package can contain 10 procedures. You can define the package so that only three procedures are public and therefore available for execution by a user of the package. The remaining procedures are private and can only be accessed by the procedures within the package. Do not confuse public and private package variables with grants to `PUBLIC`.

- Better performance

  An entire package is loaded into memory in small chunks when a procedure in the package is called for the first time. This load is completed in one operation, as opposed to the separate loads required for standalone procedures. When calls to related packaged procedures occur, no disk I/O is needed to run the compiled code in memory.
  A package body can be replaced and recompiled without affecting the specification. As a result, schema objects that reference a package's constructs (always through the specification) need not be recompiled unless the package specification is also replaced. By using packages, unnecessary recompilations can be minimized, resulting in less impact on overall database performance.

Creation of PL/SQL Packages

You create a package in two parts: the specification and the body. The package **specification** declares all public constructs of the package, whereas the **body** defines all constructs (public and private) of the package.
Example 8-1 shows part of a statement that creates the package specification for `employees_management`, which encapsulates several subprograms used to manage an employee database. Each part of the package is created with a different statement.

Example 8-2 PL/SQL Package

公有的意味着包的用户可以直接访问它。私有的意味着它对于包的用户够是隐藏的。

例如，一个包可能包含 10 个过程。您可以定义包，使其只有三个过程是公共的，且可由包的用户执行。其余的过程是私有的，且只可由包中的过程访问。不要将公有或私有的包变量与对 `PUBLIC` 的授权相混淆。

- 更好的性能

  当包中的某个过程第一次被调用时，整个包被加载到内存中。这种加载在一个操作中完成，不同于加载多个独立过程需要分别加载。当调用到包中的相关过程时，只需直接运行内存中的已编译代码，而无需额外磁盘 I/O。

  包体可以被替换并重新编译，而不会影响其规范。这样，（总是通过该规范）引用包的结构元素的模式对象不需要被重新编译，除非包规范也被替换了。通过使用包，可以将不必要的重新编译减小到最少，结果对数据库的总体性能影响就比较小。

PL/SQL 包的创建

分两个部分创建包： 包规范和包体。包**规范**声明包的所有公有结构，而**包体**定义包的所有（公有的和私有的）结构。

示例 8-1 显示了用于创建 `employees_management` 包规范的语句的一部分，封装了用于管理雇员的几个子程序。包的每个部分用不同的语句创建。

示例 8-2 PL/SQL 包

```
CREATE PACKAGE employees_management AS
  FUNCTION hire_employees (last_name VARCHAR2,
job_id VARCHAR2, manager_id NUMBER,
    salary NUMBER, commission_pct NUMBER,
department_id NUMBER) RETURN NUMBER;
  PROCEDURE fire_employees(employee_id NUMBER);
  PROCEDURE salary_raise(employee_id NUMBER,
salary_incr NUMBER);
.
.
.
  no_sal EXCEPTION;
END employees_management;
```

The specification declares the function `hire_employees`, the procedures `fire_employees` and `salary_raise`, and the exception `no_sal`. All of these public program objects are available to users who have access to the package.

The `CREATE PACKAGE BODY` command defines objects declared in the specification. The package body must be created in the same schema as the package. After creating the package, you can develop applications that call any of these public procedures or functions or raise any of the public exceptions of the package.

See Also:

Oracle Database PL/SQL Language Reference to learn about the `CREATE PACKAGE` command

Execution of PL/SQL Package Subprograms

You can reference package contents from database triggers, stored subprograms, 3GL application programs, and Oracle tools. Figure 8-2 shows database applications invoking procedures and functions in the `employees_management` package.

Figure 8-2 Calling Subprograms in a PL/SQL Package

```
CREATE PACKAGE employees_management AS
  FUNCTION hire_employees (last_name VARCHAR2,
job_id VARCHAR2, manager_id NUMBER,
    salary NUMBER, commission_pct NUMBER,
department_id NUMBER) RETURN NUMBER;
  PROCEDURE fire_employees(employee_id NUMBER);
  PROCEDURE salary_raise(employee_id NUMBER,
salary_incr NUMBER);
.
.
.
  no_sal EXCEPTION;
END employees_management;
```

此规范声明了函数 hire_employees、 过程 fire_employees 和 salary_raise、以及异常 no_sal。所有这些公有程序对象，对有权访问该包的用户都是可用的。

`CREATE PACKAGE BODY` 命令定义在规范中声明的对象。包体必须在和包相同的模式中创建。在创建包之后，您就可以开发应用程序，并在其中调用这些公共过程或函数，或抛出包中的任何公共例外。

另见：

《*Oracle 数据库 PL/SQL 语言参考*》了解 `CREATE PACKAGE` 命令

PL/SQL 包子程序的执行

您可以从数据库触发器、 存储子程序、 3GL 应用程序、和 Oracle 工具引用包内容。图 8-2 显示了数据库应用程序调用 employees_management 包中的过程和函数。

图 8-2 在 PL/SQL 包中调用子程序

Description of "Figure 8-2 Calling Subprograms in a PL/SQL Package"

Database applications explicitly call packaged procedures as necessary. After being granted the privileges for the `employees_management` package, a user can explicitly run any of the procedures contained in it. For example, SQL*Plus can issue the following statement to run the `hire_employees` package procedure:

```
EXECUTE employees_management.hire_employees
('TSMITH', 'CLERK', 1037, SYSDATE, 500, NULL, 20);
```

See Also:

- *Oracle Database PL/SQL Language Reference* for an introduction to PL/SQL packages
- *Oracle Database Advanced Application Developer's Guide* to learn how to code PL/SQL packages

PL/SQL Anonymous Blocks

An **anonymous block** is an unnamed, nonpersistent PL/SQL unit. Typical uses for anonymous blocks include:

- Initiating calls to subprograms and package constructs

Description of "Figure 8-2 Calling Subprograms in a PL/SQL Package"

数据库应用程序在必要时显式调用包内的过程。在取得对 employees_management 包的授权后，用户可以显式运行其中的任何过程。例如， SQL * Plus 可以发出以下语句来运行 hire_employees 包过程：

```
EXECUTE employees_management.hire_employees
('TSMITH', 'CLERK', 1037, SYSDATE, 500, NULL, 20);
```

另见：

- 《*Oracle 数据库 PL/SQL 语言参考*》有关 PL/SQL 包的简介
- 《*Oracle 数据库高级应用开发指南*》了解如何编写 PL/SQL 包

PL/SQL 匿名块

匿名块是一个未命名的、 非持久化的 PL/SQL 单元。匿名块的典型用途包括：

- 启动调用子程序和包结构元素

- Isolating exception handling
- Managing control by nesting code within other PL/SQL blocks

Anonymous blocks do not have the code reuse advantages of stored subprograms. Table 8-1 summarizes the differences between the two types of program units.

Table 8-1 Differences Between Anonymous Blocks and Subprograms

Is the PL/SQL Unit ...	Anonymous Blocks	Subprograms
Specified with a name?	No	Yes
Compiled with every reuse?	No	No
Stored in the database?	No	Yes
Invocable by other applications?	No	Yes
Capable of returning bind variable values?	Yes	Yes
Capable of returning function values?	No	Yes
Capable of accepting parameters?	No	Yes

An anonymous block consists of an optional declarative part, an executable part, and one or more optional exception handlers. The following sample anonymous block selects an employee last name into a variable and prints the name:

```
DECLARE
  v_lname VARCHAR2(25);
BEGIN
  SELECT last_name INTO v_lname
  FROM employees
  WHERE employee_id = 101;
  DBMS_OUTPUT.PUT_LINE('Employee last name is
'||v_lname);
END;
```

Oracle Database compiles the PL/SQL block and places it in the shared pool of the SGA, but it does not store the source code or compiled version in the database for reuse beyond the current instance. Unlike triggers, an

- 隔离例外处理
- 通过在其他 PL/SQL 块中嵌套代码来管理控制

匿名块没有存储子程序的代码重用优势。表 8-1 总结了这两类程序单元的之间的差异。

表 8-1 匿名块和子程序的差异

该 PL/SQL 单元是否...	匿名块	子程序
指定了一个名字?	否	是
每次使用时都要重新编译?	否	否
存储在数据库中?	否	是
可由其它应用程序调用?	否	是
能返回绑定变量值?	是	是
能返回函数值?	否	是
能接受参数?	否	是

匿名块包括一个可选的声明性部分、一个可执行部分、和一个或多个可选的异常处理程序。下面的示例匿名块将一个雇员的姓氏选出并存入到一个变量，并输出该姓氏：

```
DECLARE
  v_lname VARCHAR2(25);
BEGIN
  SELECT last_name INTO v_lname
  FROM employees
  WHERE employee_id = 101;
  DBMS_OUTPUT.PUT_LINE('Employee last name is
'||v_lname);
END;
```

Oracle 数据库编译 PL/SQL 块，并将其放在 SGA 的共享池中，但它不会在数据库中存储其源代码或已编译版本，以便在当前运行实例之外能重复使用。与触发器不同的是，一个匿名块在它每次加载到内存中都要被重新编

anonymous block is compiled each time it is loaded into memory. Shared SQL allows anonymous PL/SQL blocks in the shared pool to be reused and shared until they are flushed out of the shared pool.

See Also:

Oracle Database Advanced Application Developer's Guide to learn more about anonymous PL/SQL blocks

PL/SQL Language Constructs

PL/SQL blocks can include a variety of different PL/SQL language constructs. These constructs including the following:

- Variables and constants

  You can declare these constructs within a procedure, function, or package. You can use a variable or constant in a SQL or PL/SQL statement to capture or provide a value when one is needed.
- Cursors

  You can declare a cursor explicitly within a procedure, function, or package to facilitate record-oriented processing of Oracle Database data. The PL/SQL engine can also declare cursors implicitly.
- Exceptions

  PL/SQL lets you explicitly handle internal and user-defined error conditions, called **exceptions**, that arise during processing of PL/SQL code.

PL/SQL can run **dynamic SQL** statements whose complete text is not known until run time. Dynamic SQL statements are stored in character strings that are entered into, or built by, the program at run time. This technique enables you to create general purpose procedures. For example, you can create a procedure that operates on a table whose name is not known until run time.

See Also:

- *Oracle Database PL/SQL Language Reference* for details about dynamic SQL
- *Oracle Database PL/SQL Packages and Types Reference* to learn how to use dynamic SQL in the `DBMS_SQL` package

PL/SQL Collections and Records

Many programming techniques use collection types such as arrays, bags,

译。共享 SQL 允许匿名 PL/SQL 块可以在共享池中被共享和重用，直到它们被从共享池中冲掉。

另见：

《*Oracle 数据库高级应用开发指南*》了解 PL/SQL 匿名块的更多信息

PL/SQL 语言结构

PL/SQL 块可以包括各种不同的 PL/SQL 语言结构。这些结构包括：

- 变量和常量

  您可以在过程、 函数、或包内声明这些结构。在你需要在 SQL 或 PL/SQL 语句中捕获或提供一个值时，您可以使用变量或常量。
- 游标

  您可以在过程、 函数、或包中显式声明一个游标，以便对 Oracle 数据库数据进行面向记录的处理。PL/SQL 引擎也可以隐式声明游标。
- 例外

  PL/SQL 允许您显式处理**例外**，即在 PL/SQL 代码的处理过程中抛出的内部的或用户定义的错误。

PL/SQL 可以运行动态 SQL 语句，其完整的文本直到运行时才知道。动态 SQL 语句被存储为字符串，可以是手工输入的或在运行时由其程序生成。这种技术使您能够创建通用过程。例如，您可以创建一个过程，用于操作一个直到运行时才知道其名称的表。

另见：

- 《*Oracle 数据库 PL/SQL 语言参考*》有关动态 SQL 的详细信息
- 《*Oracle 数据库 PL/SQL 包和类型参考*》学习如何通过 DBMS_SQL 包使用动态 SQL

PL/SQL 集合和记录

许多编程技术都使用集合类型，如数组、 袋、 列表、 嵌套表、 集、和树

lists, nested tables, sets, and trees. To support these techniques in database applications, PL/SQL provides the data types TABLE and VARRAY, which enable you to declare associative arrays, nested tables, and variable-size arrays.

等。为在数据库应用程序中支持这些技术，PL/SQL 提供了 TABLE 和 VARRAY 数据类型，使您能够声明关联数组、 嵌套表、和变长数组。

Collections

A **collection** is an ordered group of elements, all of the same type. Each element has a unique subscript that determines its position in the collection. To create a collection, you first define a collection type, and then declare a variable of that type.

Collections work like the arrays found in most third-generation programming languages. Also, collections can be passed as parameters. So, you can use them to move columns of data into and out of database tables or between client-side applications and stored subprograms.

集合

集合是一组有序的、具有相同类型的元素。每个元素都有一个唯一的下标，以确定其在该集合中的位置。若要创建一个集合，你得首先定义一个集合类型，然后声明一个该类型的变量。

集合的工作原理与第三代编程语言中的数组类似。此外，集合可以作为参数传递。所以，你可以使用他们在数据库中的表、客户端应用程序、和存储子程序等之间，移入或移出数据列

Records

A **record** is a composite variable that can store data values of different types, similar to a struct type in C, C++, or Java. Records are useful for holding data from table rows, or certain columns from table rows.

Suppose you have data about an employee such as name, salary, and hire date. These items are dissimilar in type but logically related. A record containing a field for each item lets you treat the data as a logical unit.

You can use the %ROWTYPE attribute to declare a record that represents a table row or row fetched from a cursor. With user-defined records, you can declare your own fields.

记录

记录是一种复合变量，可以存储不同类型的的数据值，类似于 C、 C++、或 Java 中的结构类型。记录可用于容纳来自多个表行，或多个表行中的某些列的数据。

假设有某个雇员的相关数据，如名字、工资、和雇用日期。这些项的类型不同，但逻辑上相关。记录为每个项都包含了一个字段，让你可以视其为一个逻辑单元。

您可以使用 %ROWTYPE 属性来声明一条记录，以表示一个表行或从游标读取的行。对用户定义的记录，您可以声明自己的域。

See Also:

Oracle Database PL/SQL Language Reference for detailed information on using collections and records

另见：

《*Oracle 数据库 PL/SQL 语言参考*》关于使用集合和记录的详细信息

How PL/SQL Runs

PL/SQL supports both **native execution** and **interpreted execution**. In interpreted execution, PL/SQL source code is compiled into a so-called bytecode representation, which is run by a portable virtual computer implemented as part of Oracle Database. In native execution, which offers the best performance on computationally intensive program units, the source code of PL/SQL program units is compiled directly to object code for the given platform. This object code is linked into Oracle Database.

The **PL/SQL engine** is the tool used to define, compile, and run PL/SQL program units. This engine is a special component of many Oracle products, including Oracle Database. While many Oracle products have PL/SQL components, this section specifically covers the program units that

PL/SQL 如何运行

PL/SQL 支持**本地执行**和**解释执行**。对解释执行，PL/SQL 源代码被编译为所谓的字节表示形式，由实现为 Oracle 数据库一部分的一个可移植虚拟机来运行。而本机执行为计算密集型程序单元提供了最佳性能，PL/SQL 程序源代码被直接编译为给定平台的对象代码。该对象代码被链接到 Oracle 数据库中。

PL/SQL 引擎是用于定义、 编译、和运行 PL/SQL 程序单元的工具。该引擎是包括数据库的许多 Oracle 产品中的一个特殊的组件。虽然许多 Oracle 产品都有 PL/SQL 组件，但本节专门介绍可以存储在数据库中，并使用

can be stored in Oracle Database and processed using Oracle Database PL/SQL engine. The PL/SQL capabilities of each Oracle tool are described in the documentation for this tool.

Figure 8-3 illustrates the PL/SQL engine contained in Oracle Database.

Figure 8-3 The PL/SQL Engine and Oracle Database

Description of "Figure 8-3 The PL/SQL Engine and Oracle Database"

The program unit is stored in a database. When an application calls a stored procedure, the database loads the compiled program unit into the shared pool in the system global area (SGA) (see "Shared Pool"). The PL/SQL and SQL statement executors work together to process the statements in the procedure.

You can call a stored procedure from another PL/SQL block, which can be either an anonymous block or another stored procedure. For example, you can call a stored procedure from Oracle Forms.

A PL/SQL procedure executing on Oracle Database can call an **external procedure** or function written in the C programming language and stored in a shared library. The C routine runs in a separate address space from that of Oracle Database.

See Also:

- *Oracle Database PL/SQL Language Reference* to learn about PL/SQL architecture

Oracle 数据库 PL/SQL 引擎处理的程序单元。每种 Oracle 工具的 PL/SQL 功能在相应工具的文档中描述。

图 8-3 说明了包含在 Oracle 数据库中的 PL/SQL 引擎。

图 8-3 的 PL/SQL 引擎和数据库

Description of "Figure 8-3 The PL/SQL Engine and Oracle Database"

程序单元存储在数据库中。当应用程序调用存储过程时，数据库会将已编译的程序单元加载到系统全局区（SGA）的共享池中 （请参阅"共享池"）。PL/SQL 执行器和 SQL 语句执行器共同处理过程中的语句。

您可以从另一个 PL/SQL 块调用存储过程，这可以是一个匿名块或另一个存储过程。例如，您可以从 Oracle Forms 调用存储过程。

在 Oracle 数据库上执行的 PL/SQL 过程，可以调用一个由 C 编程语言编写的并存储在一个共享库中的外部过程或函数。C 例程在一个不同于数据库的单独的地址空间中运行。

另见：

- 《*Oracle 数据库 PL/SQL 语言参考*》了解 PL/SQL 体系结构

- *Oracle Database Advanced Application Developer's Guide* to learn more about external procedures

Overview of Java in Oracle Database

Java has emerged as the object-oriented programming language of choice. Java includes the following features:

- A Java Virtual Machine (JVM), which provides the basis for platform independence
- Automated storage management techniques, such as garbage collection
- Language syntax that borrows from C and enforces strong typing

Note:
This chapter assumes that you have some familiarity with the Java language.

The database provides Java programs with a dynamic data-processing engine that supports complex queries and multiple views of data. Client requests are assembled as data queries for immediate processing. Query results are generated dynamically.

The combination of Java and Oracle Database helps you create component-based, network-centric applications that can be easily updated as business needs change. In addition, you can move applications and data stores off the desktop and onto intelligent networks and network-centric servers. More importantly, you can access these applications and data stores from any client device.

Figure 8-4 shows a traditional two-tier, client/server configuration in which clients call Java stored procedures in the same way that they call PL/SQL subprograms.

Figure 8-4 Two-Tier Client/Server Configuration

- 《*Oracle 数据库高级应用开发指南*》了解外部过程的详细信息

Oracle 数据库中的 Java 概述

Java 已成为面向对象的编程语言的选择。Java 包括以下特性：

- Java 虚拟机 (JVM)，提供了平台独立性的基础
- 自动化的存储管理技术，例如垃圾回收
- 从 C 语言借鉴的强类型语言语法

注意：
本章假定您已熟悉 Java 语言。

数据库为 Java 程序提供了一个动态的数据处理引擎，支持复杂的查询和数据的多个视图。客户端请求被装配为数据查询，以备立即处理。查询结果是动态生成的。

Java 和 Oracle 数据库的结合，可帮助您创建基于组件的、 以网络为中心的应用程序，并可以轻松地随业务需求的变化而变化。此外，您可以将应用程序和数据存储，从桌面移到智能网络和以网络为中心的服务器上。更重要的是，您可以从任何客户端设备来访问这些应用程序和数据存储。

图 8-4 显示了传统的两层客户端/服务器配置，客户端使用与调用 PL/SQL 子程序相同的方式，来调用 Java 存储过程。

图 8-4 两层客户端/服务器配置

Description of "Figure 8-4 Two-Tier Client/Server Configuration"

See Also:

Oracle Database 2 Day + Java Developer's Guide for an introduction to using Java with Oracle Database

Overview of the Java Virtual Machine (JVM)

A **JVM** is a virtual processor that runs compiled Java code. Java source code compiles to low-level machine instructions, known as **bytecodes**, that are platform independent. The Java bytecodes are interpreted through the JVM into platform-dependent actions.

Overview of Oracle JVM

Oracle JVM is a complete, Java2-compliant environment for running pure Java applications. It is compatible with the JLS and the JVM specifications. It supports the standard Java binary format and APIs. In addition, Oracle Database adheres to standard Java language semantics, including dynamic class loading at run time.

Figure 8-5 illustrates how Oracle Java applications reside on top of the Java core class libraries, which reside on top of the Oracle JVM. Because the Oracle Java support system is located within the database, the JVM interacts with database libraries, instead of directly interacting with the operating system.

Figure 8-5 Java Component Structure

Description of "Figure 8-4 Two-Tier Client/Server Configuration"

另见：

《*Oracle 数据库 2 日 + Java 开发人员指南*》关于在 Oracle 数据库中使用 Java 的简介

Java 虚拟机 (JVM) 的概述

JVM 是一个运行已编译 Java 代码的虚拟处理器。Java 源代码被编译为称为**字节码**的、独立于平台的低级计算机指令。Java 字节码通过 JVM 解释为平台依赖的操作。

Oracle JVM 概述

Oracle JVM 是一个完整的、符合 Java2 标准的环境，用于运行纯 Java 应用程序。它与 JLS 和 JVM 规范兼容。它支持 Java 的标准二进制格式和 API。此外，Oracle 数据库遵从标准的 Java 语言语义，包括在运行时动态加载类。

图 8-5 说明了 Oracle Java 应用程序如何驻留在 Oracle JVM 中的 Java 核心类库的基础上。因为 Oracle Java 支持系统位于数据库内， 所以 JVM 是与数据库的库文件交互，而不是直接与操作系统交互的。

图 8-5 Java 组件结构

Description of "Figure 8-5 Java Component Structure"

Unlike other Java environments, Oracle JVM is embedded within Oracle Database. Some important differences exist between Oracle JVM and typical client JVMs. For example, in a standard Java environment, you run a Java application through the interpreter by issuing the following command on the command line, where *classname* is the name of the class that you want the JVM to interpret first:

```
java classname
```

The preceding command causes the application to run within a process on your operating system. However, if you are not using the command-line interface, then you must load the application into the database, publish the interface, and then run the application within a database data dictionary.

See Also:

See *Oracle Database Java Developer's Guide* for a description of other differences between the Oracle JVM and typical client JVMs

Main Components of Oracle JVM

Oracle JVM runs in the same process space and address space as the database kernel by sharing its memory heaps and directly accessing its relational data. This design optimizes memory use and increases throughput.

Description of "Figure 8-5 Java Component Structure"

与其他的 Java 环境不同， Oracle JVM 是内嵌于数据库中的。Oracle JVM 和典型的客户端 JVM 之间存在一些重要的差异。例如，在一个标准的 Java 环境中，您通过在命令行上向解释器发出下面的命令来运行 Java 应用程序，其中 *classname* 是你希望 JVM 首先解释的类名：

```
java classname
```

前面的命令导致应用程序在您的操作系统上的某个进程内运行。但是，如果您不使用命令行接口，则你必须加载应用程序到数据库、 发布该接口、然后在数据库数据字典中运行这个应用程序。

另见：

请参阅《*Oracle 数据库 Java 开发人员指南*》关于 Oracle JVM 和典型的客户端 JVM 之间的其他差异的描述

Oracle JVM 的主要组件

Oracle JVM 通过共享其内存堆，并直接访问其关联数据，与数据库内核运行在同一个进程空间。这种设计优化了内存使用并提高了吞吐量。

Oracle JVM provides a run-time environment for Java objects. It fully supports Java data structures, method dispatch, exception handling, and language-level threads. It also supports all the core Java class libraries, including `java.lang`, `java.io`, `java.net`, `java.math`, and `java.util`.

Figure 8-6 shows the main components of Oracle JVM.

Figure 8-6 Main Components of Oracle JVM

Description of "Figure 8-6 Main Components of Oracle JVM"

Oracle JVM embeds the standard Java namespace in the database schemas. This feature lets Java programs access Java objects stored in Oracle Database and application servers across the enterprise.

In addition, Oracle JVM is tightly integrated with the scalable, shared memory architecture of the database. Java programs use call, session, and object lifetimes efficiently without user intervention. As a result, Oracle JVM and middle-tier Java business objects can be scaled, even when they have session-long state.

See Also:

Oracle Database Java Developer's Guide for a description of the main components of Oracle JVM

Java Programming Environment

Oracle furnishes enterprise application developers with an end-to-end Java solution for creating, deploying, and managing Java applications. The solution consists of client-side and server-side programmatic interfaces, tools to support Java development, and a Java Virtual Machine integrated with Oracle Database. All these products are compatible with Java standards.

Oracle JVM 提供了一个 Java 对象的运行时环境。它完全支持 Java 数据结构、方法调度、 异常处理、和语言级线程。它还支持所有核心 Java 类库，包括 java.lang、 java.io、 java.net、 java.math、和 java.util。

图 8-6 显示 Oracle JVM 的主要组件。

图 8-6 Oracle JVM 的主要组件

Description of "Figure 8-6 Main Components of Oracle JVM"

Oracle JVM 在数据库模式中嵌入标准 Java 命名空间。此特性使 Java 程序能够访问整个企业范围内的数据库和应用程序服务器中的 Java 对象。

另外，Oracle JVM 与可扩展的、 共享的数据库内存体系结构紧密集成。Java 程序有效地使用调用、 会话、和对象生存期，而无需用户干预。这样，Oracle JVM 和中间层 Java 业务对象，即使处于长会话状态，仍然具有可扩展性。

另见：

《*Oracle 数据库 Java 开发人员指南*》 关于 Oracle JVM 的主要部件的描述

Java 编程环境

Oracle 为企业应用程序开发人员提供了一个端到端的 Java 解决方案，用于创建、 部署、并管理 Java 应用程序。该解决方案包含客户端和服务器端的编程接口、支持 Java 开发的工具、和与 Oracle 数据库集成的 Java 虚拟机。所有这些产品都与 Java 标准兼容。

The Java programming environment consists of the following additional features:

- Java stored procedures as the Java equivalent and companion for PL/SQL. Java stored procedures are tightly integrated with PL/SQL. You can call Java stored procedures from PL/SQL packages and procedures from Java stored procedures.
- The JDBC and SQLJ programming interfaces for accessing SQL data.
- Tools and scripts that assist in developing, loading, and managing classes.

Java Stored Procedures

A **Java stored procedure** is a Java method published to SQL and stored in the database. Like a PL/SQL subprogram, a Java procedure can be invoked directly with products like SQL*Plus or indirectly with a trigger. You can access it from any Oracle Net client—OCI, precompiler, or JDBC.

To publish Java methods, you write **call specifications**, which map Java method names, parameter types, and return types to their SQL counterparts. When called by client applications, a Java stored procedure can accept arguments, reference Java classes, and return Java result values.

Applications calling the Java method by referencing the name of the call specification. The run-time system looks up the call specification definition in the Oracle data dictionary and runs the corresponding Java method.

In addition, you can use Java to develop powerful programs independently of PL/SQL. Oracle Database provides a fully compliant implementation of the Java programming language and JVM.

See Also:

Oracle Database Java Developer's Guide explains how to write stored procedures in Java, how to access them from PL/SQL, and how to access PL/SQL functionality from Java

Java and PL/SQL Integration

You can call existing PL/SQL programs from Java and Java programs from PL/SQL. This solution protects and leverages your PL/SQL and Java code.

Oracle Database offers two different approaches for accessing SQL data from Java, JDBC and SQLJ. Both approaches are available on the client and server. As a result, you can deploy applications on the client and server without modifying the code.

JDBC Drivers

Java 编程环境还包括以下额外功能：

- Java 存储过程类似于 Java，经常和 PL/SQL 配合使用。Java 存储过程与 PL/SQL 紧密集成。您可以从 PL/SQL 包调用 Java 存储过程，或从 Java 存储过程调用 PL/SQL 存储过程。
- 用于访问 SQL 数据的 SQLJ 和 JDBC 编程接口。
- 用于辅助开发、 加载、和管理 Java 类的工具和脚本。

Java 存储过程

Java 存储过程是一个发布到 SQL、 并存储在数据库中的 Java 方法。与 PL/SQL 子程序类似， Java 存储过程可以从 SQL * Plus 之类的产品直接调用，或从一个触发器间接调用。您可以从任何 Oracle Net 客户端访问它——如 OCI、预编译器、或 JDBC 等。

若要发布 Java 方法，你需要编写**调用规范**，以映射 Java 方法名称、参数类型、及其与 SQL 对应的返回类型。当被客户端应用程序调用时， Java 存储过程可以接受参数、 引用 Java 类、并返回 Java 结果值。

应用程序通过引用调用规范的名称来调用 Java 方法。运行时系统在 Oracle 数据字典中查找调用规范的定义，并运行相应的 Java 方法。

此外，您可以使用 Java 来开发功能强大的独立于 PL/SQL 的程序。Oracle 数据库提供了一个与 Java 编程语言和 JVM 完全兼容的实现。

另见：

《*Oracle 数据库 Java 开发人员指南*》 说明了如何在 Java 中编写存储过程、如何从 PL/SQL 访问它们、以及如何从 Java 访问 PL/SQL 功能

Java 和 PL/SQL 集成

您可以从 Java 调用现有 PL/SQL 程序，或从 PL/SQL 调用 Java 程序。此解决方案能保护和利用您的 PL/SQL 和 Java 代码。

Oracle 数据库提供了两种不同的方法从 Java、 JDBC、和 SQLJ 访问 SQL 数据。这两种方法都可用在客户端或服务器上。这样一来，您可以将应用程序部署在客户端或服务器上，而无需修改代码。

JDBC 驱动程序

JDBC is a database access protocol that enables you to connect to a database and run SQL statements and queries to the database. The core Java class libraries provide only one JDBC API, `java.sql`. However, JDBC is designed to enable vendors to supply drivers that offer the necessary specialization for a particular database. Oracle provides the distinct JDBC drivers shown in the following table.

Driver	Description
JDBC Thin driver	You can use the JDBC Thin driver to write pure Java applications and applets that access Oracle SQL data. The JDBC Thin driver is especially well-suited for Web-based applications and applets, because you can dynamically download it from a Web page, similar to any other Java applet.
JDBC OCI driver	The JDBC OCI driver accesses Oracle-specific native code, that is, non-Java code, and libraries on the client or middle tier, providing performance boost compared to the JDBC Thin driver, at the cost of significantly larger size and client-side installation.
JDBC server-side internal driver	Oracle Database uses the server-side internal driver when the Java code runs on the server. It allows Java applications running in Oracle JVM on the server to access locally defined data, that is, data on the same system and in the same process, with JDBC. It provides a performance boost, because of its ability to use the underlying Oracle RDBMS libraries directly, without the overhead of an intervening network connection between the Java code and SQL data. By supporting the same Java-SQL interface on the server, Oracle Database does not require you to rework code when deploying it.

See Also:

- "ODBC and JDBC"
- *Oracle Database 2 Day + Java Developer's Guide* and *Oracle Database JDBC Developer's Guide*

SQLJ

SQLJ is an ANSI standard for embedding SQL statements in Java programs. You can use SQLJ in stored procedures, triggers, and methods

JDBC 是一种数据库访问协议，使您能够连接到数据库，并对数据库运行 SQL 语句和查询。核心 Java 类库仅提供一个 JDBC API， java.sql。但是，JDBC 被设计为允许供应商针对特定数据库提供所需的特殊的驱动程序。Oracle 提供了下表中所示的不同 JDBC 驱动程序。

驱动程序	描述
JDBC 瘦驱动程序	你可以使用 JDBC 瘦驱动程序来编写纯 Java 应用程序和访问 Oracle SQL 数据的小程序。JDBC 瘦驱动程序特别适合基于 Web 的应用程序和小程序，因为您可以从 Web 页动态地下载它，与下载任何其他 Java 小程序类似。
JDBC OCI 驱动程序	JDBC OCI 驱动程序访问特定于 Oracle 的本地代码（即：非 Java 的代码）和客户端或中间层上的库，相比 JDBC 瘦驱动程序，提供了性能的提升，其代价是尺寸明显变大，且需要客户端安装。
JDBC 服务器端内部驱动程序	当 Java 代码在服务器上运行时，Oracle 数据库使用服务器端内部驱动程序。它允许 Java 应用程序在服务器上的 Oracle JVM 中运行，以访问在本地定义的数据，即与 JDBC 在同一系统上，且在同一进程中的数据。由于具有使用底层的 Oracle RDBMS 库的能力，而没有 Java 代码和 SQL 数据之间网络连接的开销，其性能大大提高。通过在服务器上支持相同的 Java-SQL 接口，Oracle 数据库不需要你重新编码代码来部署它。

另见：

- "ODBC 和 JDBC"
- 《*Oracle 数据库 2 日 + Java 开发人员指南*》 和 《*Oracle 数据库 JDBC 开发人员指南*》

SQLJ

SQLJ 是一个用于在 Java 程序中嵌入 SQL 语句的 ANSI 标准。您可以在 Oracle 数据库的存储过程、 触发器、和方法等环境中使用 SQLJ。此外，您

within the Oracle Database environment. In addition, you can combine SQLJ programs with JDBC.

SQLJ provides a simple, but powerful, way to develop client-side and middle-tier applications that access databases from Java (see "SQLJ"). A developer writes a program using SQLJ and then uses the **SQLJ translator** to translate embedded SQL to pure JDBC-based Java code. At run time, the program can communicate with multi-vendor databases using standard JDBC drivers.

The following example shows a simple SQLJ executable statement:

```
String name;
#sql  { SELECT first_name INTO :name FROM
employees WHERE employee_id=112 };
System.out.println("Name is " + name + ", employee
number = " + employee_id);
```

Because Oracle Database provides a complete Java environment, you cannot compile SQLJ programs on a client that will run on the database. Instead, you can compile them directly on the server.

See Also:

Oracle Database SQLJ Developer's Guide

Overview of Triggers

A database trigger is a compiled stored program unit, written in either PL/SQL or Java, that Oracle Database invokes ("fires") automatically whenever one of the following operations occurs:

- 1 DML statements on a particular table or view, issued by any user

  DML statements modify data in schema objects. For example, inserting and deleting rows are DML operations.
- 2 DDL statements issued either by a particular user or any user

  DDL statements define schema objects. For example, creating a table and adding a column are DDL operations.
- 3 Database events

  User login or logoff, errors, and database startup or shutdown are events that can invoke triggers.

Triggers are schema objects that are similar to subprograms but differ in the way they are invoked. A subprogram is explicitly run by a user, application, or trigger. Triggers are implicitly invoked by the database

可以将 SQLJ 程序和 JDBC 组合使用。

SQLJ 提供了一种简单、但功能强大的方法，来开发 Java 客户端和中间层应用程序，以访问数据库（请参阅"SQLJ"）。开发人员将使用 SQLJ 来编写程序，然后使用 **SQLJ 翻译器**将嵌入式的 SQL 翻译为完全基于 JDBC 的 Java 代码。运行时，程序可以使用标准的 JDBC 驱动程序与各种供应商的数据库进行通信。

下面的示例显示了一个简单的 SQLJ 可执行语句：

```
String name;
#sql  { SELECT first_name INTO :name FROM
employees WHERE employee_id=112 };
System.out.println("Name is " + name + ", employee
number = " + employee_id);
```

由于 Oracle 数据库提供了一个完整的 Java 环境，您不能在客户端上编译将在数据库上运行的 SQLJ 程序。相反，您可以直接在服务器上编译它们。

另见：

《*Oracle 数据库 SQLJ 开发人员指南*》

触发器概述

数据库触发器是一个已编译的存储程序单元，使用 PL/SQL 或 Java 编写，在下列操作之一发生时，数据库自动调用（触发）它：

- 1 由任何用户在一个特定的表或视图上，发出 DML 语句

  DML 语句修改模式对象中的数据。例如，插入和删除行即是 DML 操作。
- 2 由任何用户或特定用户发出 DDL 语句

  DDL 语句定义模式对象。例如，创建表并添加一列即是 DDL 操作。
- 3 数据库事件

  用户登录或注销、错误、和数据库启动或关闭，是触发器可以调用的事件。

触发器是模式对象，类似于子程序；但其调用方法不同。子程序由用户、应用程序、或触发器显式运行。而触发器是在触发的事件发生时由数据库隐

when a triggering event occurs.

See Also:

- "Overview of SQL Statements" to learn about DML and DDL
- "Overview of Instance Startup and Shutdown"

Advantages of Triggers

The correct use of triggers enables you to build and deploy applications that are more robust and that use the database more effectively. You can use triggers to:

- Automatically generate derived column values
- Prevent invalid transactions
- Provide auditing and event logging
- Record information about table access

You can use triggers to enforce low-level business rules common for all client applications. For example, several applications may access the `employees` table. If a trigger on this table ensures the format of inserted data, then this business logic does not need to be reproduced in every client. Because the trigger cannot be circumvented by the application, the business logic in the trigger is used automatically.

You can use both triggers and integrity constraints to define and enforce any type of integrity rule. However, Oracle strongly recommends that you only use triggers to enforce complex business rules not definable using an integrity constraint (see "Introduction to Data Integrity").

Excessive use of triggers can result in complex interdependencies that can be difficult to maintain in a large application. For example, when a trigger is invoked, a SQL statement within its trigger action potentially can fire other triggers, resulting in **cascading triggers** that can produce unintended effects.

See Also:

Oracle Database 2 Day Developer's Guide and *Oracle Database PL/SQL Language Reference* for guidelines and restrictions when planning triggers for your application

Types of Triggers

Triggers can be categorized according to their means of invocation and the type of actions they perform. Oracle Database supports the following types of triggers:

式调用的。

另见：

- "SQL 语句概述"了解 DML 和 DDL
- "实例启动与关闭概述"

触发器的优势

正确使用触发器使您能够生成和部署更健壮、 更能有效使用数据库的应用程序。您可以使用触发器：

- 自动生成派生列中的值
- 防止无效事务
- 提供审计和事件日志记录
- 记录有关访问表的信息

您可以使用触发器强制执行对所有客户端应用程序都适用的底层业务规则。例如，几个应用程序都可能要访问 `employees` 表。如果此表上的某个触发器可以确保插入数据的格式，则不需要在每个客户端中重复这种业务逻辑。因为该触发器不能被应用程序绕过，触发器中的业务逻辑会被自动应用。

您可以使用触发器或完整性约束，来定义和执行任何类型的完整性规则。但是，当不能使用一个完整性约束来定义时，Oracle 强烈建议您仅使用触发器来强制执行这种复杂的业务规则（请参见"数据完整性简介"）。

过度使用的触发器，可能导致复杂的相互依赖关系，在一个大的应用程序中可能很难维护。例如，当调用触发器时，在其触发器操作中的 SQL 语句可能会触发其他触发器，导致**级联触发**，且产生意想不到的结果。

另见：

《*Oracle 数据库 2 日开发人员指南*》和 《*Oracle 数据库 PL/SQL 语言参考*》关于为应用程序规划触发器的指导原则和限制

触发器类型

触发器可以根据其调用方式和他们所执行的操作类型进行分类。Oracle 数据库支持以下类型的触发器：

- Row triggers

  A **row trigger** fires each time the table is affected by the triggering statement. For example, if a statement updates multiple rows, then a row trigger fires once for each row affected by the `UPDATE`. If a triggering statement affects no rows, then a row trigger is not run. Row triggers are useful if the code in the trigger action depends on data provided by the triggering statement or rows that are affected.
- Statement triggers

  A **statement trigger** is fired once on behalf of the triggering statement, regardless of the number of rows affected by the triggering statement. For example, if a statement deletes 100 rows from a table, a statement-level `DELETE` trigger is fired only once. Statement triggers are useful if the code in the trigger action does not depend on the data provided by the triggering statement or the rows affected.
- `INSTEAD OF` triggers

  An **`INSTEAD OF` trigger** is fired by Oracle Database instead of executing the triggering statement. These triggers are useful for transparently modifying views that cannot be modified directly through DML statements.
- Event triggers

  You can use triggers to publish information about database events to subscribers. Event triggers are divided into the following categories:
  - A **system event trigger** can be caused by events such as database instance startup and shutdown or error messages.
  - A **user event trigger** is fired because of events related to user logon and logoff, DDL statements, and DML statements.

See Also:

- *Oracle Database 2 Day Developer's Guide*

- 行触发器

  行触发器在表每次受到触发语句影响时触发。例如。如果一个语句更新多个行，则行触发器为受 `UPDATE` 影响的每一行触发一次。如果触发语句未影响任何行，则行触发器不会运行。如果触发器操作中的代码依赖于触发语句提供的数据或受影响的行，则行触发器是有用的。

- 语句触发器

  语句触发器针对触发语句只触发一次，而无论受触发语句影响的行数。例如，如果一个语句从表中删除 100 行，语句级 DELETE 触发器仅触发一次。如果触发器操作中的代码并不依赖触发语句所提供的数据或受影响的行，则语句触发器是有用的。

- `INSTEAD OF` 触发器

  数据库会触发 **INSTEAD OF 触发器**，而不是执行触发语句。这些触发器可用于透明地修改不能直接通过 DML 语句修改的视图。

- 事件触发器

  您可以使用触发器将有关数据库事件的信息发布到订阅服务器。事件触发器可分为以下几类：
  - **系统事件触发器**可以由诸如数据库实例启动和关闭或错误消息之类的事件引起。
  - **用户事件触发器**会因为与用户登录和注销、DDL 语句、和 DML 语句等相关的事件而激发。

另见：

- 《*Oracle 数据库 2 日开发人员指南*》

- *Oracle Database PL/SQL Language Reference*

Timing for Triggers

You can define the **trigger timing**—whether the trigger action is to be run before or after the triggering statement. A **simple trigger** is a single trigger on a table that enables you to specify actions for exactly one of the following timing points:

- Before the firing statement
- Before each row affected by the firing statement
- After each row affected by the firing statement
- After the firing statement

For statement and row triggers, a `BEFORE` trigger can enhance security and enable business rules before making changes to the database. The `AFTER` trigger is ideal for logging actions.

A **compound trigger** can fire at multiple timing points. Compound triggers help program an approach in which the actions that you implement for various timing points share common data.

See Also:

Oracle Database PL/SQL Language Reference to learn about compound triggers

Creation of Triggers

The `CREATE TRIGGER` statement creates or replaces a database trigger. A PL/SQL trigger has the following general syntactic form:

```
CREATE TRIGGER trigger_name
  triggering_statement
  [trigger_restriction]
BEGIN
 triggered_action;
END;
```

A PL/SQL trigger has the following basic components:

- Trigger name

  The name must be unique with respect to other triggers in the same schema. For example, the name may be `part_reorder_trigger`.

- 《*Oracle 数据库 PL/SQL 语言参考*》

触发时间

您可以定义**触发时间** —— 触发器操作是要在触发语句之前运行，还是之后运行。**简单触发器**是一个在表上的触发器，使您能够仅为以下时间点之一指定操作：

- 触发语句之前
- 在受触发语句影响的每行之前
- 在受触发语句影响的每行之后
- 触发语句之后

对语句和行触发器， BEFORE 触发器可以增强安全性，并在对数据库进行更改之前启用业务规则。而 AFTER 触发器则对日志记录操作很理想。

复杂触发器可以在多个时间点触发。复杂触发器提供一种编程方法，使你可以在多个不同的时间点，对共享的公共数据实施操作。

另见：

《*Oracle 数据库 PL/SQL 语言参考*》了解复杂触发器

创建触发器

使用 `CREATE TRIGGER` 语句创建或替换数据库触发器。PL/SQL 触发器具有以下的一般语法形式：

```
CREATE TRIGGER trigger_name
  triggering_statement
  [trigger_restriction]
BEGIN
 triggered_action;
END;
```

PL/SQL 触发器具有以下基本组件：

- 触发器名称

  在相同的模式中，触发器的名称必须是唯一的。例如，名称可能是 part_reorder_trigger。

- The trigger event or statement

  A triggering event or statement is the SQL statement, database event, or user event that causes a trigger to be invoked. For example, a user updates a table.
- Trigger restriction

  A trigger restriction specifies a Boolean expression that must be `true` for the trigger to fire. For example, the trigger is not invoked unless the number of available parts is less than a present reorder amount.
- Triggered action

  A triggered action is the procedure that contains the SQL statements and code to be run when a triggering statement is issued and the trigger restriction evaluates to true. For example, a user inserts a row into a pending orders table.

Suppose that you create the `orders` and `lineitems` tables as follows:

```
CREATE TABLE orders
( order_id NUMBER PRIMARY KEY,
  /* other attributes */
  line_items_count NUMBER DEFAULT 0 );
CREATE TABLE lineitems
( order_id REFERENCES orders,
  seq_no   NUMBER,
  /* other attributes */
  CONSTRAINT lineitems PRIMARY
KEY(order_id,seq_no) );
```

The `orders` table contains a row for each unique order, whereas the `lineitems` table contains a row for each item in an order. Example 8-3 shows a sample trigger that automatically updates the `orders` table with the number of items in an order.

Example 8-3 lineitems_trigger

- 触发事件或语句

  触发事件或语句是导致触发器被调用的 SQL 语句、数据库事件、或用户事件。例如，用户更新一个表。

- 触发器限制

  触发器限制指定一个布尔表达式，其值必须为 true ，触发器才会触发。例如，某触发器仅当可用部件的数量不足订购量时才触发。

- 触发的操作

  触发的操作是包含 SQL 语句和代码的过程，当发出一个触发语句、且触发器限制的计算结果为 true 时将被运行。例如，用户向待办订单表中插入行。

假设您创建 `orders` 表和 `lineitems` 表，如下所示：

```
CREATE TABLE orders
( order_id NUMBER PRIMARY KEY,
  /* other attributes */
  line_items_count NUMBER DEFAULT 0 );
CREATE TABLE lineitems
( order_id REFERENCES orders,
  seq_no   NUMBER,
  /* other attributes */
  CONSTRAINT lineitems PRIMARY
KEY(order_id,seq_no) );
```

`orders` 表为每个唯一订单包含一行，而 `lineitems` 表为所订单中的每个项目包含一行。示例 8-3 显示一个简单触发器，自动更新 `orders` 表的订单中的项目数。

示例 8-3 lineitems_trigger

```
CREATE OR REPLACE TRIGGER lineitems_trigger
  AFTER INSERT OR UPDATE OR DELETE ON lineitems
  FOR EACH ROW
BEGIN
  IF (INSERTING OR UPDATING)
  THEN
    UPDATE orders SET line_items_count =
NVL(line_items_count,0)+1
    WHERE order_id = :new.order_id;
  END IF;
  IF (DELETING OR UPDATING)
  THEN
    UPDATE orders SET line_items_count =
NVL(line_items_count,0)-1
    WHERE order_id = :old.order_id;
  END IF;
END;
/
```

In Example 8-3, the triggering statement is an `INSERT`, `UPDATE`, or `DELETE` on the `lineitems` table. No triggering restriction exists. The trigger is invoked for each row changed. The trigger has access to the old and new column values of the current row affected by the triggering statement. Two correlation names exist for every column of the table being modified: the old value (`:old`), and the new value (`:new`).

If rows in `lineitems` are inserted or updated for an order, then after the action the trigger calculates the number of items in this order and updates the `orders` table with the count. Table 8-2 illustrates a scenario in which a customer initiates two orders and adds and removes line items from the orders.

```
CREATE OR REPLACE TRIGGER lineitems_trigger
  AFTER INSERT OR UPDATE OR DELETE ON lineitems
  FOR EACH ROW
BEGIN
  IF (INSERTING OR UPDATING)
  THEN
    UPDATE orders SET line_items_count =
NVL(line_items_count,0)+1
    WHERE order_id = :new.order_id;
  END IF;
  IF (DELETING OR UPDATING)
  THEN
    UPDATE orders SET line_items_count =
NVL(line_items_count,0)-1
    WHERE order_id = :old.order_id;
  END IF;
END;
/
```

在示例 8-3 中，触发语句是一个在 `lineitems` 表上的 `INSERT`、`UPDATE`、或 `DELETE` 操作。没有触发限制存在。触发器为更改的每一行调用一次。触发器对正在被触发语句影响的当前行，具有访问其旧列值和新列值的权限。正在修改的表的每一列存在两个相关名字： 旧值 (`:old`)和新值 （`:new`）。

如果在 `lineitems` 表中插入或更新了某订单行，则在此操作后触发器会计算该订单中的项目数，并在 `orders` 表中更新计数值。表 8-2 说明了一个场景，其中某个客户启动了两个订单，并在订单中添加或删除订单行项目。

Table 8-2 Row-Level Trigger Scenario

表 8-2 行级触发器场景

SQL Statement	Triggered SQL Statement	Description	描述

`SQL> INSERT INTO orders (order_id) VALUES (78);`
 	
`1 row created.`
		The customer creates an order with ID 78. At this point the customer has no items in the order.
		
		Because no action is performed on the `lineitems` table, the trigger is
			客户创建 ID 为 78 的订单。此时客户在该订单中没有订单项目。
			
			因为未对 lineitems 表执行任何操作，所以触发器不会调用。

		not invoked.	
SQL> INSERT INTO orders (order_id) VALUES (92); 1 row created.		The customer creates a separate order with ID 92. At this point the customer has no items in the order. Because no action is performed on the lineitems table, the trigger is not invoked.	客户创建另一个 ID 为 92 的订单。此时客户在该订单中没有订单项目。 因为未对 lineitems 表执行任何操作，所以触发器不会调用。
SQL> INSERT INTO lineitems (order_id, seq_no) VALUES (78,1); 1 row created.	UPDATE orders SET line_items_count = NVL(NULL,0)+1 WHERE order_id = 78;	The customer adds an item to order 78. The INSERT invokes the trigger. The triggered statement increases the line item count for order 78 from 0 to 1.	客户将某订单项添加到订单 78。 INSERT 操作调用触发器。被触发的语句将订单 78 的项目计数从 0 增加到 1。
SQL> INSERT INTO lineitems (order_id, seq_no) VALUES (78,2); 1 row created.	UPDATE orders SET line_items_count = NVL(1,0)+1 WHERE order_id = 78;	The customer adds an additional item to order 78. The INSERT invokes the trigger. The triggered statement increases the line item count for order 78 from 1 to 2.	客户将另一个项目添加到订单 78。 INSERT 操作调用触发器。被触发的语句将订单 78 的项目计数从 1 增加到 2。
SQL> SELECT * FROM orders; ORDER_ID LINE_ITEMS_COUNT --------- ---------- - 78 2 92 0		The customer queries the status of the two orders. Order 78 contains two items. Order 92 contains no items.	客户查询两个订单的状态。订单 78 包含两个项目。而订单 92 不包含任何项目。
SQL> SELECT * FROM lineitems; ORDER_ID SEQ_NO ---------- ----------		The customer queries the status of the line items. Each item is uniquely identified by the order ID and the sequence number.	客户查询订单行项目的状态。每个项目都由订单 ID 和序列号唯一标识。

78 1 78 2			
SQL> UPDATE lineitems SET order_id = 92; 2 rows updated.	UPDATE orders SET line_items_count = NVL(NULL,0)+1 WHERE order_id = 92; UPDATE orders SET line_items_count = NVL(2,0)-1 WHERE order_id = 78; UPDATE orders SET line_items_count = NVL(1,0)+1 WHERE order_id = 92; UPDATE orders SET line_items_count = NVL(1,0)-1 WHERE order_id = 78;	The customer moves the line items that were in order 78 to order 92. The UPDATE statement changes 2 rows in the lineitems tables, which invokes the trigger once for each row. Each time the trigger is invoked, both IF conditions in the trigger are met. The first condition increments the count for order 92, whereas the second condition decreases the count for order 78. Thus, four total UPDATE statements are run.	客户将订单 78 中的项目移到订单 92 中。 UPDATE 语句更改了 lineitems 表中的 2 行，这将为每个行调用一次触发器。 每次调用触发器时，在触发器中的两个 IF 条件都符合。第一个条件增加订单 92 的计数，而第二个条件减少订单 78 的计数。因此，总共运行四个 UPDATE 语句。
SQL> SELECT * FROM orders; ORDER_ID LINE_ITEMS_COUNT --- --- 78 0 92 2		The customer queries the status of the two orders. The net effect is that the line item count for order 92 has increased from 0 to 2, whereas the count for order 78 has decreased from 2 to 0.	客户查询两个订单的状态。净影响是订单 92 的行项目计数已从 0 增至 2，而订单 78 的计数从 2 下降为 0。
SQL> SELECT * FROM lineitems; ORDER_ID SEQ_NO --- --- 92 1 92 2		The customer queries the status of the line items. Each item is uniquely identified by the order ID and the sequence number.	客户查询订单行项目的状态。唯一标识每个项目的订单 ID 和序列号。
SQL> DELETE FROM lineitems;	UPDATE orders SET line_items_count =	The customer now removes all line items from all orders.	客户现在从所有订单中删除所有行项

2 rows deleted.	NVL(2,0)-1 WHERE order_id = 92; UPDATE orders SET line_items_count = NVL(1,0)-1 WHERE order_id = 92;	The DELETE statement changes 2 rows in the lineitems tables, which invokes the trigger once for each row. For each trigger invocation, only one IF condition in the trigger is met. Each time the condition decreases the count for order 92 by 1. Thus, two total UPDATE statements are run.	目。 DELETE 语句更改 lineitems 的表中的 2 行，每个行会调用一次触发器。对于每个触发器调用，在触发器中只有一个 IF 条件得到满足。每次该条件从订单 92 将计数减 1。因此，总共运行两个 UPDATE 语句。
SQL> SELECT * FROM orders; ORDER_ID LINE_ITEMS_COUNT --------- ---------------- 78 0 92 0 SQL> SELECT * FROM lineitems; no rows selected		The customer queries the status of the two orders. Neither order contains line items. The customer also queries the status of the line items. No items exist.	客户查询两个订单的状态。订单都不包含任何行项目。 客户还查询订单行项目的状态。也没有任何项目。

See Also:

- *Oracle Database 2 Day Developer's Guide* and *Oracle Database PL/SQL Language Reference* to learn how to create triggers
- *Oracle Database PL/SQL Language Reference* to learn about the CREATE TRIGGER command

Execution of Triggers

Oracle Database executes a trigger internally using the same steps as for subprogram execution. The only subtle difference is that a user has the right to fire a trigger if he or she has the privilege to run the triggering statement. With this exception, the database validates and runs triggers the same way as stored subprograms.

See Also:

Oracle Database PL/SQL Language Reference to learn more about trigger execution

Storage of Triggers

Oracle Database stores PL/SQL triggers in compiled form in a database

另见：

- 《*Oracle 数据库 2 日开发人员指南*》和《*Oracle 数据库 PL/SQL 语言参考*》了解如何创建触发器
- 《*Oracle 数据库 PL/SQL 语言参考*》了解 CREATE TRIGGER 命令

执行触发器

Oracle 数据库在内部使用与执行子程序相同的步骤来执行触发器。唯一的细微区别在于，如果用户有权运行触发语句，则他或她有权触发该触发器。除了这个区别之外，数据库使用与存储子程序相同的方式来验证和运行触发器。

另见：

《*Oracle 数据库 PL/SQL 语言参考*》了解触发器执行的更多信息

触发器存储

与 PL/SQL 存储过程类似，Oracle 数据库将触发器以已编译形式存储在数据

schema, just like PL/SQL stored procedures. When a CREATE TRIGGER statement commits, the compiled PL/SQL code is stored in the database and the source code of the PL/SQL trigger is removed from the shared pool.

Figure 8-7 shows a database application with SQL statements that implicitly invoke PL/SQL triggers. The triggers are stored separately from their associated tables.

Figure 8-7 Triggers

Description of "Figure 8-7 Triggers"

Java triggers are stored in the same manner as PL/SQL triggers. However, a Java trigger references Java code that was separately compiled with a CALL statement. Thus, creating a Java trigger involves creating Java code and creating the trigger that references this Java code.

See Also:

Oracle Database PL/SQL Language Reference to learn about compiling and storing triggers

库模式中。当 CREATE TRIGGER 语句提交后，其编译后的 PL/SQL 代码被存储在数据库中，而该 PL/SQL 触发器源代码从共享池中被移除。

图 8-7 显示了一个数据库应用程序，其中的一个 SQL 语句隐式调用了 PL/SQL 触发器。触发器与其相关联的表是分开存储的。

图 8-7 触发器

Description of "Figure 8-7 Triggers"

Java 触发器以与 PL/SQL 触发器相同的方式被存储在数据库中。但是，Java 触发器使用 CALL 语句引用单独编译的 Java 代码。因此，创建一个 Java 触发器涉及到创建 Java 代码，并创建引用此 Java 代码的触发器。

另见：

《*Oracle 数据库 PL/SQL 语言参考*》了解编译和存储触发器

9 Data Concurrency and Consistency

This chapter explains how Oracle Database maintains consistent data in a multiuser database environment.

This chapter contains the following sections:

- Introduction to Data Concurrency and Consistency
  - Multiversion Read Consistency
  - Locking Mechanisms
  - ANSI/ISO Transaction Isolation Levels
- Overview of Oracle Database Transaction Isolation Levels
  - Read Committed Isolation Level
  - Serializable Isolation Level
  - Read-Only Isolation Level
- Overview of the Oracle Database Locking Mechanism
  - Summary of Locking Behavior
  - Use of Locks
  - Lock Modes
  - Lock Conversion and Escalation
  - Lock Duration
  - Locks and Deadlocks
- Overview of Automatic Locks
  - DML Locks
  - DDL Locks
  - System Locks
- Overview of Manual Data Locks
- Overview of User-Defined Locks

Introduction to Data Concurrency and Consistency

In a single-user database, a user can modify data without concern for other users modifying the same data at the same time. However, in a multiuser database, statements within multiple simultaneous transactions can update the same data. Transactions executing simultaneously must produce meaningful and consistent results. Therefore, a multiuser database must provide the following:

Previous Next View PDF

第 9 章 数据并发性和一致性

本章介绍了在多用户数据库环境中，Oracle 数据库如何维护一致的数据。

本章包含以下各节：

- 数据并发和一致性介绍
  - 多版本读一致性
  - 锁定机制
  - ANSI/ISO 事务隔离级别
- Oracle 数据库事务隔离级别概述
  - 读提交隔离级别
  - 可串行化隔离级别
  - 只读隔离级别
- 数据库锁定机制概述
  - 锁定行为总结
  - 使用锁
  - 锁模式
  - 锁转换和锁升级
  - 锁持续时间
  - 锁和死锁
- 自动锁的概述
  - DML 锁
  - DDL 锁
  - 系统锁
- 手动数据锁概述
- 用户定义的锁的概述

数据并发和一致性介绍

在单用户的数据库中，用户可以修改数据，而不用担心其他用户在同一时间修改相同的数据。但是，在一个多用户的数据库中，多个事务内的语句可以同时更新相同的数据。同时执行的多个事务必须产生有意义且一致的结果。因此，多用户数据库必须提供以下功能：

- **Data concurrency**, which ensures that users can access data at the same time
- **Data consistency**, which ensures that each user sees a consistent view of the data, including visible changes made by the user's own transactions and committed transactions of other users

To describe consistent transaction behavior when transactions run concurrently, database researchers have defined a transaction isolation model called **serializability**. A serializable transaction operates in an environment that makes it appear as if no other users were modifying data in the database.

While this degree of isolation between transactions is generally desirable, running many applications in serializable mode can seriously compromise application throughput. Complete isolation of concurrently running transactions could mean that one transaction cannot perform an insertion into a table being queried by another transaction. In short, real-world considerations usually require a compromise between perfect transaction isolation and performance.

Oracle Database maintains data consistency by using a **multiversion consistency model** and various types of locks and transactions. In this way, the database can present a view of data to multiple concurrent users, with each view consistent to a point in time. Because different versions of data blocks can exist simultaneously, transactions can read the version of data committed at the point in time required by a query and return results that are consistent to a single point in time.

See Also:

Chapter 5, "Data Integrity" and Chapter 10, "Transactions"

Multiversion Read Consistency

In Oracle Database, **multiversioning** is the ability to simultaneously materialize multiple versions of data. Oracle Database maintains **multiversion read consistency**, which means that database queries have the following characteristics:

- Read-consistent queries

  The data returned by a query is committed and consistent with respect to a single point in time.

  Important:

  Oracle Database *never* permits **dirty reads**, which occur when a transaction reads uncommitted data in another transaction.

- **数据并发性**，确保多个用户可以同时访问数据
- **数据一致性**，确保每个用户看到数据的一致的视图，包括可以看到用户自己的事务所做的更改，和其他用户已提交的事务所做的更改。

为描述当多个事务同时运行时的事务一致性行为，数据库研究人员定义了一种称为**可串行性**的事务隔离模型。可串行化事务在一种使其看起来好像没有其他用户正在修改数据库中的数据的环境中运作。

虽然事务之间的这种隔离度好像不错，但在可序列化模式下运行许多应用程序可能会严重影响应用程序吞吐量。对并发运行事务的完全隔离可能意味着一个事务无法在某个正在被另一个事务查询的表上执行插入操作。简而言之，现实的考虑通常需要在完美的事务隔离性和性能之间的一个折衷。

Oracle 数据库通过使用**多版本一致性模型**和各种类型的锁和事务，来维护数据的一致性。通过这种方式，数据库可以向多个并发用户呈现一个数据的视图，每个视图都在某个时间点上是一致的。因为不同版本的数据块可以同时存在，事务可以读取在某个查询请求时间点上的已提交版本数据，并返回符合一个单一时间点的结果。

另见：

第 5 章，"数据完整性"和第 10 章，"事务"

多版本读一致性

在 Oracle 数据库中，**多版本**即同时实现数据的多个版本的能力。Oracle 数据库维护**多版本读取一致性**，这意味着数据库查询具有以下特征：

- 读一致查询

  查询所返回的数据已提交的，且关于某个单一时间点一致。

  重要：

  Oracle 数据库绝不允许**脏读**。当一个事务读取了另一个事务中未提交的数据时，会发生脏读。

To illustrate the problem with dirty reads, suppose one transaction updates a column value without committing. A second transaction reads the updated and dirty (uncommitted) value. The first session rolls back the transaction so that the column has its old value, but the second transaction proceeds using the updated value, corrupting the database. Dirty reads compromise data integrity, violate foreign keys, and ignore unique constraints.

- Nonblocking queries

  Readers and writers of data do not block one another (see "Summary of Locking Behavior").

Statement-Level Read Consistency

Oracle Database always enforces **statement-level read consistency**, which guarantees that data returned by a single query is committed and consistent with respect to a single point in time. The point in time to which a single SQL statement is consistent depends on the transaction isolation level and the nature of the query:

- In the read committed isolation level, this point is the time at which the *statement* was opened. For example, if a `SELECT` statement opens at SCN 1000, then this statement is consistent to SCN 1000.
- In a serializable or read-only transaction this point is time the *transaction* began. For example, if a transaction begins at SCN 1000, and if multiple `SELECT` statements occur in this transaction, then each statement is consistent to SCN 1000.
- In a Flashback Query operation (`SELECT ... AS OF`), the `SELECT` statement explicitly specifies the point in time. For example, you can query a table as it appeared last Thursday at 2 p.m.

See Also:

Oracle Database Advanced Application Developer's Guide to learn about Flashback Query

Transaction-Level Read Consistency

Oracle Database can also provide read consistency to all queries in a transaction, known as **transaction-level read consistency**. In this case, each statement in a transaction sees data from the *same* point in time, which is the time at which the transaction began.

Queries made by a serializable transaction see changes made by the transaction itself. For example, a transaction that updates `employees`

为说明脏读的问题，假设一个事务更新某列的值，但不提交。第二个事务读取此已更新的脏 （未提交） 值。第一个会话回滚了事务，使该列仍具有其旧值，但第二个事务继续使用更新的值，这会损坏数据库。脏读会破坏数据的完整性、 破坏外键、和忽略唯一约束。

- 非阻塞查询

  数据的读取者和写入者不会相互阻塞（请参见"锁定行为总结"）。

语句级读取一致性

Oracle 数据库始终强制执行**语句级读取一致性**，保证单个查询所返回的数据是已提交的、且关于某个单一时间点一致。单个 SQL 语句所一致的时间点取决于事务的隔离级别和查询的性质：

- 在读提交隔离级别，该时间点是语句打开的时间。例如，如果一个 SELECT 语句在 SCN 1000 时打开，则此语句一致于 SCN 1000。
- 在可串行化或只读事务隔离级别，该时间点为事务开始的时间。例如，如果一个事务开始于 SCN 1000，且在该事务中有多个 SELECT 语句发生，则每个语句都一致于 SCN 1000。
- 在闪回查询操作（`SELECT ... AS OF`）中，SELECT 语句显式指定时间点。例如，你可以查询某个表在上星期四下午 2 时的数据

另见：

《*Oracle 数据库高级应用开发指南*》了解闪回查询

事务级读取一致性

Oracle 数据库还可以为一个事务中的所有查询提供读取一致性，这称为**事务级读取一致性**。在这种情况下，事务中的每个语句都看到来自同一时间点（即该事务开始的时间）的数据。

在一个可序列化事务中的多个查询，能看到事务本身所做的更改。例如，某个事务更新了 `employees` 表，然后其后续查询将看到对 `employees` 所

and then queries `employees` will see the updates. Transaction-level read consistency produces repeatable reads and does not expose a query to phantom reads.

Read Consistency and Undo Segments

To manage the multiversion read consistency model, the database must create a read-consistent set of data when a table is simultaneously queried and updated. Oracle Database achieves this goal through undo data.

Whenever a user modifies data, Oracle Database creates undo entries, which it writes to undo segments ("Undo Segments"). The undo segments contain the old values of data that have been changed by uncommitted or recently committed transactions. Thus, multiple versions of the same data, all at different points in time, can exist in the database. The database can use snapshots of data at different points in time to provide **read-consistent views** of the data and enable nonblocking queries.

Read consistency is guaranteed in single-instance and Oracle Real Application Clusters (Oracle RAC) environments. Oracle RAC uses a cache-to-cache block transfer mechanism known as Cache Fusion to transfer read-consistent images of data blocks from one database instance to another.

See Also:

- "Internal LOBs" to learn about read consistency mechanisms for LOBs
- *Oracle Database 2 Day + Real Application Clusters Guide* to learn about Cache Fusion

Read Consistency: Example

Figure 9-1 shows a query that uses undo data to provide statement-level read consistency in the read committed isolation level.

Figure 9-1 Read Consistency in the Read Committed Isolation Level

做的更新。事务级读取一致性产生可重复的读取，且不会产生幻读读。

读取一致性及撤消

为管理多版本的读取一致性模型，当表同时被查询和更新时，数据库必须创建一组读取一致的数据。Oracle 数据库通过使用撤销数据实现了这一目标。

每当用户修改了数据，Oracle 数据库会创建撤销条目，并写入到撤销段 ("撤销段"）。撤销段包含由未提交事务或最近提交的事务所更改的数据的旧值。因此，同一数据在各个不同时间点上的多个版本，都可以存在于数据库中。数据库可以使用在不同时间点的数据快照，来提供数据**读取一致视图**，并实现非阻塞查询。

读取一致性在单实例和 Oracle 真正应用集群 （Oracle RAC） 环境中都可以得到保证。Oracle RAC 使用一种称为缓存融合的“缓存到缓存”的数据块传输机制，将一个数据库实例中的数据块读取一致映像传送到另一个实例中。

另见：

- "内部 LOBs"了解 LOB 的读取一致性机制
- 《*Oracle 数据库 2 日 + 实际应用集群指南*》了解缓存融合

读一致性：示例

图 9-1 显示了一个查询，在已提交读隔离级别使用撤销数据以提供语句级的读取一致性。

图 9-1 在已提交读隔离级别的读取一致性

Description of "Figure 9-1 Read Consistency in the Read Committed Isolation Level"

As the database retrieves data blocks on behalf of a query, the database ensures that the data in each block reflects the contents of the block when the query began. The database rolls back changes to the block as needed to reconstruct the block to the point in time the query started processing. The database uses a mechanism called an SCN to guarantee the order of transactions. As the `SELECT` statement enters the execution phase, the database determines the SCN recorded at the time the query began executing. In Figure 9-1, this SCN is 10023. Each query in the transaction

Description of "Figure 9-1 Read Consistency in the Read Committed Isolation Level"

当数据库为某个查询检索数据块时，数据库确保每个块中的数据反映了该查询开始时的内容。数据库根据需要回滚对数据块所做的更改，以将块重建到查询处理开始的状态。

数据库使用一种称为 SCN 的机制，来保证事务的顺序。当 SELECT 语句进入执行阶段时，数据库会确定查询开始执行时所记录的 SCN。在图 9-1 中，该 SCN 为 10023。在事务中的每个查询必须返回在 SCN 10023 时的已提交数据。

must return committed data with respect to SCN 10023.

In Figure 9-1, blocks with SCNs *after* 10023 indicate changed data, as shown by the two blocks with SCN 10024. The `SELECT` statement requires a version of the block that is consistent with committed changes. The database copies current data blocks to a new buffer and applies undo data to reconstruct previous versions of the blocks. These reconstructed data blocks are called **consistent read (CR) clones**.

In Figure 9-1, the database creates two CR clones: one block consistent to SCN 10006 and the other block consistent to SCN 10021. The database returns the reconstructed data for the query. In this way, Oracle Database prevents dirty reads.

See Also:

"Database Buffer Cache" and "System Change Numbers (SCNs)"

Read Consistency and Transaction Tables

The database uses a transaction table, also called an **interested transaction list (ITL)**, to determine if a transaction was uncommitted when the database began modifying the block. The block header of every segment block contains a transaction table.

Entries in the transaction table describe which transactions have rows locked and which rows in the block contain committed and uncommitted changes. The transaction table points to the undo segment, which provides information about the timing of changes made to the database.

In a sense, the block header contains a recent history of transactions that affected each row in the block. The `INITRANS` parameter of the `CREATE TABLE` and `ALTER TABLE` statements controls the amount of transaction history that is kept.

See Also:

Oracle Database SQL Language Reference to learn about the `INITRANS` parameter

Locking Mechanisms

In general, multiuser databases use some form of data locking to solve the problems associated with data concurrency, consistency, and integrity. **Locks** are mechanisms that prevent destructive interaction between transactions accessing the same resource.

See Also:

"Overview of the Oracle Database Locking Mechanism"

ANSI/ISO Transaction Isolation Levels

The SQL standard, which has been adopted by both ANSI and ISO/IEC,

在图 9-1 中，其 SCN 大于 10023 的块具有已更改数据，如图中的两个具有 SCN 10024 的块所示。SELECT 语句需要一个与已提交更改块一致的版本。该数据库将当前数据块复制到新的缓冲区，并应用撤消数据，以重新构造块的早期版本。这些重建的数据块被称为**一致读取 (CR) 克隆**。

在图 9-1 中，数据库创建了两个 CR 克隆： 一个块与 SCN 10006 一致，而另一个块与 SCN 10021 一致。数据库为查询返回重建的数据。通过这种方式，数据库可以防止脏读。

另见：

"数据库缓冲区缓存"和"系统更改号 (SCNs) "

读取一致性和事务表

数据库使用一个称为**感兴趣事务列表 (ITL)** 的事务表，来确定当数据库开始修改块时是否某个事务还未提交。每个段块的块头包含一个事务表。

事务表中的条目描述了哪些事务有被锁定的行，以及块中的哪些行包含提交和未提交的更改。事务表指向撤销段，提供对数据库所做的更改的时间相关信息。

在某种意义上，块头包含影响块中每个行的事务的最近历史记录。CREATE TABLE 和 ALTER TABLE 语句的 `INITRANS` 参数，控制被保留的交易历史记录条数。

另见：

《*Oracle 数据库 SQL 语言参考*》了解 `INITRANS` 参数

锁定机制

通常，多用户数据库使用某种形式的数据锁定，来解决与数据并发性、一致性、和完整性相关的问题。**锁**是防止访问同一资源的事务之间的破坏性相互作用的机制。

另见：

"数据库锁定机制概述"

ANSI/ISO 事务隔离级别

已由 ANSI 和 ISO/IEC 采纳的 SQL 标准，定义了四个**事务隔离**级别。这些

defines four levels of **transaction isolation**. These levels have differing degrees of impact on transaction processing throughput.
These isolation levels are defined in terms of phenomena that must be prevented between concurrently executing transactions. The preventable phenomena are:

- Dirty reads

  A transaction reads data that has been written by another transaction that has not been committed yet.
- Nonrepeatable (fuzzy) reads

  A transaction rereads data it has previously read and finds that another committed transaction has modified or deleted the data. For example, a user queries a row and then later queries the same row, only to discover that the data has changed.
- Phantom reads

  A transaction reruns a query returning a set of rows that satisfies a search condition and finds that another committed transaction has inserted additional rows that satisfy the condition.
  For example, a transaction queries the number of employees. Five minutes later it performs the same query, but now the number has increased by one because another user inserted a record for a new hire. More data satisfies the query criteria than before, but unlike in a fuzzy read the previously read data is unchanged.

The SQL standard defines four levels of isolation in terms of the phenomena that a transaction running at a particular isolation level is permitted to experience. Table 9-1 shows the levels.

Table 9-1 Preventable Read Phenomena by Isolation Level

Isolation Level	**Dirty Read**	**Nonrepeatable Read**	**Phantom Read**
Read uncommitted	Possible	Possible	Possible
Read committed	Not possible	Possible	Possible
Repeatable read	Not possible	Not possible	Possible
Serializable	Not possible	Not possible	Not possible

Oracle Database offers the read committed (default) and serializable

级别对事务处理吞吐量有不同程度的影响。

这些隔离级别根据在同时运行的事务之间必须防止的现象来定义。可预防的现象有：

- 脏读

  一个事务读取了已被另一个事务写入、但尚未提交的数据。

- 不可重复（模糊）读

  一个事务重新读取之前曾经读取过的数据，发现另一个已提交的事务已修改或删除了该数据。例如，用户查询某行，然后稍后又查询相同的行，却发现数据已更改。

- 幻像读

  一个事务重新运行满足某搜索条件的查询，并返回一个行集，发现另一个已提交的事务已插入了满足搜索条件的其他行。

  例如，一个事务查询雇员数目。五分钟后它执行相同的查询，但现在人数却增加了一个，这是因为另一个用户为一名新员工插入了一条记录。满足查询条件的数据比之前更多了，但与不可重复读不同，之前读取的数据不会变化。

根据运行在某个特定的隔离级别的事务所允许发生的现象，SQL 标准定义了四个隔离级别。表 9-1 显示了这些级别。

表 9-1 在不同隔离级别下的可预防的读现象

隔离级别	**脏读**	**不可重复读**	**幻像读**
未提交读	可能	可能	可能
已提交读	不可能	可能	可能
可重复读	不可能	不可能	可能
可串行化	不可能	不可能	不可能

Oracle 数据库提供了已提交读 （默认值）和可串行化隔离级别。另外，数

isolation levels. Also, the database offers a read-only mode.

See Also:

- "Overview of Oracle Database Transaction Isolation Levels" to learn about read committed, serializable, and read-only isolation levels
- *Oracle Database SQL Language Reference* for a discussion of Oracle Database conformance to SQL standards

Overview of Oracle Database Transaction Isolation Levels

Table 9-1 summarizes the ANSI standard for transaction isolation levels. The standard is defined in terms of the phenomena that are either permitted or prevented for each isolation level. Oracle Database provides the transaction isolation levels:

- Read Committed Isolation Level
- Serializable Isolation Level
- Read-Only Isolation Level

See Also:

- *Oracle Database Advanced Application Developer's Guide* to learn more about transaction isolation levels
- *Oracle Database SQL Language Reference* and *Oracle Database PL/SQL Language Reference* to learn about `SET TRANSACTION ISOLATION LEVEL`

Read Committed Isolation Level

In the **read committed isolation level**, which is the default, every query executed by a transaction sees only data committed before the query—not the transaction—began. This level of isolation is appropriate for database environments in which few transactions are likely to conflict. A query in a read committed transaction avoids reading data that commits while the query is in progress. For example, if a query is halfway through a scan of a million-row table, and if a different transaction commits an update to row 950,000, then the query does not see this change when it reads row 950,000. However, because the database does not prevent other transactions from modifying data read by a query, other transactions may change data *between* query executions. Thus, a transaction that runs the same query twice may experience fuzzy reads and phantoms.

Read Consistency in the Read Committed Isolation Level

据库也可以提供一种只读模式。

另见：

- "Oracle 数据库事务隔离级别概述"了解已提交读、可串行化、和只读隔离级别
- 《*Oracle 数据库 SQL 语言参考*》关于 Oracle 数据库与 SQL 标准的一致性的讨论

Oracle 数据库事务隔离级别概述

表 9-1 总结了事务隔离级别的 ANSI 标准。该标准定义了在各个隔离级别所允许或必须防止的现象。Oracle 数据库提供如下事务隔离级别：

- 已提交读隔离级别
- 可串行化隔离级别
- 只读隔离级别

另见：

- 《*Oracle 数据库高级应用开发指南*》了解事务隔离级别的更多信息
- 《*Oracle 数据库 SQL 语言参考*》和 《*Oracle 数据库 PL/SQL 语言参考*》了解 `SET TRANSACTION ISOLATION LEVEL`

读提交隔离级别

在（默认的）**已提交读隔离级别**中，事务中执行的每个查询，仅看到在查询开始之前提交的数据 ——而不是事务开始之前提交的数据。这一隔离级别适合于几乎不可能发生事务冲突的数据库环境。

已提交读事务中的查询可以避免读取在查询过程中所提交的数据。例如，如果一个查询正扫描到一个百万行表的中间，而另一个不同的事务对第 950000 行提交了一个更新，但当查询读到第 950000 行时，它并不能看见这个变化。但是，因为数据库不会阻止其它事务修改一个查询所读取的数据，其他事务可能会在查询执行期间更改数据。因此，两次运行相同查询的事务可能会遇到模糊读取和幻像读取现象。

在已提交读隔离级别中的读取一致性

A consistent result set is provided for every query, guaranteeing data consistency, with no action by the user. An implicit query, such as a query implied by a `WHERE` clause in an `UPDATE` statement, is guaranteed a consistent set of results. However, each statement in an implicit query does not see the changes made by the DML statement itself, but sees the data as it existed before changes were made.

If a `SELECT` list contains a PL/SQL function, then the database applies statement-level read consistency at the statement level for SQL run within the PL/SQL function code, rather than at the parent SQL level. For example, a function could access a table whose data is changed and committed by another user. For each execution of the `SELECT` in the function, a new read-consistent snapshot is established.

See Also:

"Subqueries and Implicit Queries"

Conflicting Writes in Read Committed Transactions

In a read committed transaction, a **conflicting write** occurs when the transaction attempts to change a row updated by an uncommitted concurrent transaction, sometimes called a **blocking transaction**. The read committed transaction waits for the blocking transaction to end and release its row lock. The options are as follows:

- If the blocking transaction rolls back, then the waiting transaction proceeds to change the previously locked row as if the other transaction never existed.
- If the blocking transaction commits and releases its locks, then the waiting transaction proceeds with its intended update to the newly changed row.

Table 9-2 shows how transaction 1, which can be either serializable or read committed, interacts with read committed transaction 2. Table 9-2 shows a classic situation known as a **lost update** (see "Use of Locks"). The update made by transaction 1 is not in the table *even though transaction 1 committed it*. Devising a strategy to handle lost updates is an important part of application development.

Table 9-2 Conflicting Writes and Lost Updates in a READ COMMITTED Transaction

为每个查询提供一个一致的结果集，其目的是为了保证数据一致性，而无需用户采取任何行动。对于隐含的查询（如在一个 UPDATE 语句中的 WHERE 子句），也同样可以保证其一致的结果集。但是，在隐式查询中的每个语句不会看到 DML 语句本身所做的更改，只能看到更改之前所存在的数据。

如果 SELECT 列表中包含一个 PL/SQL 函数，则数据库在该 PL/SQL 函数代码内运行的 SQL 所在语句级别（而不是在父 SQL 级别）上，应用语句级别读取一致性。例如，一个函数可能会访问某个表，其数据被另一个用户更改并提交。在 `SELECT` 语句中的每次函数运行，都会建立一个新的读一致性快照。

另见：

"子查询和隐式查询"

在读提交事务中的写入冲突

在一个读已提交事务中，当事务尝试更改由另一个未提交并发事务（有时称为阻塞事务）所更新的行时，会发生写入冲突。读提交事务将等待阻塞事务结束并释放其行锁。有两个选项如下所示：

- 如果阻塞事务回滚，正在等待的事务将继续并更改之前被锁定的行，就像另一个事务从未存在一样。
- 如果阻塞事务提交并释放了锁，则正在等待的事务将对这个刚被更新的行继续其预定更新。

表 9-2 显示了一个可能是可串行化的或已提交读的事务 1，如何与另一个已提交读的事务 2 进行交互。表 9-2 显示了一个称为**丢失更新**的典型情况（请参阅"使用锁定"）。事务 1 所作的更新不能在表中反映出来，*即使事务 1 已经提交它*。制定一项策略以处理更新丢失是应用程序开发的一个重要部分。

表 9-2 在一个已提交读事务中的写入冲突和丢失更新

Session 1	Session 2	Explanation	解释
`SQL> SELECT last_name, salary FROM employees WHERE`		Session 1 queries the salaries for Banda, Greene, and Hintz. No	会话 1 查询 Banda、Greene、和 Hintz 的薪金。找不到名为 Hintz 的任

last_name IN ('Banda','Greene','Hintz'); LAST_NAME SALARY ---------- ---------- Banda 6200 Greene 950		employee named Hintz is found.	何雇员。
SQL> UPDATE employees SET salary = 7000 WHERE last_name = 'Banda';		Session 1 begins a transaction by updating the Banda salary. The default isolation level for transaction 1 is READ COMMITTED.	会话 1 开始一个事务 1，更新 Banda 的薪金。事务 1 的默认隔离级别是已提交读。
	SQL> SET TRANSACTION ISOLATION LEVEL READ COMMITTED;	Session 2 begins transaction 2 and sets the isolation level explicitly to READ COMMITTED.	会话 2 开始一个事务 2，并将隔离级别显式设置为已提交读。
	SQL> SELECT last_name, salary FROM employees WHERE last_name IN('Banda','Greene','Hintz'); LAST_NAME SALARY ---------- ---------- Banda 6200 Greene 9500	Transaction 2 queries the salaries for Banda, Greene, and Hintz. Oracle Database uses read consistency to show the salary for Banda before the uncommitted update made by transaction 1.	事务 2 查询 Banda、Greene 和 Hintz 的薪金。Oracle 数据库使用读取一致性，显示事务 1 做出未提交更新前 Banda 的薪金。
	SQL> UPDATE employees SET salary =9900 WHERE last_name = 'Greene';	Transaction 2 updates the salary for Greene successfully because transaction 1 locked only the Banda row (see "Row Locks (TX)").	事务 2 成功更新 Greene 的薪金，因为事务 1 只锁定了 Banda 所在的行（请参阅"行锁 (TX)"）。
SQL> INSERT INTO employees (employee_id, last_name, email,hire_date, job_id) VALUES (210,'Hintz', 'JHINTZ', SYSDATE,'SH_CLERK');		Transaction 1 inserts a row for employee Hintz, but does not commit.	事务 1 为雇员 Hintz 插入一行，但不提交。
	SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Banda','Greene','Hintz');	Transaction 2 queries the salaries for employees Banda, Greene, and Hintz.	事务 2 查询 Banda、Greene 和 Hintz 的薪金。 事务 2 看到它自己对 Greene 薪金的

	LAST_NAME SALARY ---------- ---------- Banda 6200 Greene 9900	Transaction 2 sees its own update to the salary for Greene. Transaction 2 does not see the uncommitted update to the salary for Banda or the insertion for Hintz made by transaction 1.	更新。但事务 2 看不到由事务 1 对 Banda 薪金所做的未提交更新，或为 Hintz 插入的新行。
	SQL> UPDATE employees SET salary =6300 WHERE last_name = 'Banda'; -- prompt does not return	Transaction 2 attempts to update the row for Banda, which is currently locked by transaction 1, creating a conflicting write. Transaction 2 waits until transaction 1 ends.	事务 2 试图更新当前被事务 1 锁定的 Banda 行，这会产生一个写入冲突。事务 2 必须等待，直到事务 1 结束。
SQL> COMMIT;		Transaction 1 commits its work, ending the transaction.	事务 1 提交其工作，以结束事务。
	1 row updated. SQL>	The lock on the Banda row is now released, so transaction 2 proceeds with its update to the salary for Banda.	现在 Banda 行上的锁释放了，所以事务 2 得以继续，并完成对 Banda 薪金的更新。
	SQL> SELECT last_name, salary FROM employees WHERE last_name IN('Banda','Greene','Hintz'); LAST_NAME SALARY ---------- ---------- Banda 6300 Greene 9900 Hintz	Transaction 2 queries the salaries for employees Banda, Greene, and Hintz. The Hintz insert committed by transaction 1 is now visible to transaction 2. Transaction 2 sees its own update to the Banda salary.	事务 2 查询雇员 Banda、Greene 和 Hintz 的薪酬。现在事务 1 已提交了所插入的 Hintz 行，并能被事务 2 看到。事务 2 也看到自己对 Banda 薪金的更新。
	COMMIT;	Transaction 2 commits its work, ending the transaction.	事务 2 提交其工作，以结束事务。
SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Banda','Greene','Hintz'); LAST_NAME SALARY ---------- ---------- Banda 6300 Greene 9900		Session 1 queries the rows for Banda, Greene, and Hintz. The salary for Banda is 6300, which is the update made by transaction 2. The update of Banda's salary to 7000 made by transaction 1 is now "lost."	会话 1 查询 Banda、Greene、Hintz 的行。Banda 的薪金是 6300，这是事务 2 所作的更新。事务 1 对 Banda 的薪金改至 7000 的更新现在"丢失"了。

Hintz			

Serializable Isolation Level

In the **serialization isolation level**, a transaction sees only changes committed at the time the transaction—not the query—began and changes made by the transaction itself. A serializable transaction operates in an environment that makes it appear as if no other users were modifying data in the database.

Serializable isolation is suitable for environments:

- With large databases and short transactions that update only a few rows
- Where the chance that two concurrent transactions will modify the same rows is relatively low
- Where relatively long-running transactions are primarily read only

In serializable isolation, the read consistency normally obtained at the statement level extends to the entire transaction. Any row read by the transaction is assured to be the same when reread. Any query is guaranteed to return the same results for the duration of the transaction, so changes made by other transactions are not visible to the query regardless of how long it has been running. Serializable transactions do not experience dirty reads, fuzzy reads, or phantom reads.

Oracle Database permits a serializable transaction to modify a row only if changes to the row made by other transactions were *already* committed when the serializable transaction began. The database generates an error when a serializable transaction tries to update or delete data changed by a different transaction that committed *after* the serializable transaction began:

```
ORA-08177: Cannot serialize access for this
transaction
```

When a serializable transaction fails with the `ORA-08177` error, an application can take several actions, including the following:

- Commit the work executed to that point
- Execute additional (but different) statements, perhaps after rolling back to a savepoint established earlier in the transaction

可串行化隔离级别

在**可串行化隔离级别**，事务只看到自事务开始以来（而不是自查询以来）该事务本身所提交的更改。可串行化事务的运行环境，使其看起来好像没有其他用户在修改数据库中的数据。

可串行化隔离适合如下环境：

- 大型数据库中只更新少数几行的短事务
- 两个并发事务将修改相同的行的可能性相对较低
- 较长时间运行的事务主要为只读事务

在可串行化隔离级别，在语句级别所获得的读取一致性通常延伸到整个事务范围。当重新读取在同一事务中之前读取的任何行时，保证结果相同。可以保证任何查询在该事务的持续期间返回相同的结果，因此其他事务所做的更改是不可见的，无论该查询已运行了多长时间。可串行化事务不会遇到脏读、 模糊读取、或幻读。

Oracle 数据库允许可串行化事务修改行，只要当可序列化事务开始时，由其它事务对行所做更改*已*提交。当一个串行化事务试图更新或删除某数据，而该数据在串行化事务开始后被一个不同的事务更改并提交，则数据库将生成一个错误：

```
ORA-08177: Cannot serialize access for this
transaction
```

当可序列化事务失败，产生 ORA-08177 错误时，应用程序可以采取行动，包括以下几种：

- 将所执行的工作提交到该点
- 也许要先回滚到事务中之前建立的某保存点，然后执行一些其他额外的（不同）语句

- Roll back the entire transaction

Table 9-3 shows how a serializable transaction interacts with other transactions. If the serializable transaction does not try to change a row committed by another transaction after the serializable transaction began, then a **serialized access problem** is avoided.

Table 9-3 Read Consistency and Serialized Access Problems in Serializable Transactions

- 回滚整个事务

表 9-3 显示了一个可串行化事务与其它事务之间的交互。如果可串行化事务不会尝试更改由另一个事务在该可序列化事务开始后所提交的行，就可以避免串行化访问问题。

表 9-3 可串行化事务中的读取一致性和串行化访问问题

Session 1	**Session 2**	**Explanation**	**解释**

`SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Banda','Greene','Hintz');`
 	
`LAST_NAME SALARY`
`---------- ----------`
`Banda 6200`
`Greene 9500`
		Session 1 queries the salaries for Banda, Greene, and Hintz. No employee named Hintz is found.	会话 1 查询 Banda、Greene、和 Hintz 的薪酬。找不到名为 Hintz 的任何雇员。

`SQL> UPDATE employees SET salary = 7000 WHERE last_name = 'Banda';`		Session 1 begins transaction 1 by updating the Banda salary. The default isolation level for is `READ COMMITTED`.	会话 1 开始一个事务 1，更新 Banda 的薪金。事务 1 的默认的隔离级别是已提交读。
	`SQL> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;`	Session 2 begins transaction 2 and sets it to the `SERIALIZABLE` isolation level.	会话 2 开始一个事务 2，并将隔离级别显式设置为可序列化隔离级别。

	`SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Banda','Greene','Hintz');`
	
	`LAST_NAME SALARY`
	`---------- ----------`
	`Banda 6200`
	`Greene 9500`
		Transaction 2 queries the salaries for Banda, Greene, and Hintz. Oracle Database uses read consistency to show the salary for Banda *before* the uncommitted update made by transaction 1.	事务 2 查询 Banda、Greene、和 Hintz 的薪金。Oracle 数据库使用读取一致性，显示事务 1 做出未提交更新前 Banda 的薪金。

	`SQL> UPDATE employees SET salary = 9900 WHERE last_name = 'Greene';`	Transaction 2 updates the Greene salary successfully because only the Banda row is locked.	事务 2 成功更新 Greene 的薪金，因为只有 Banda 行被锁定。

SQL> INSERT INTO employees (employee_id, last_name, email, hire_date, job_id) VALUES (210, 'Hintz', 'JHINTZ', SYSDATE, 'SH_CLERK');		Transaction 1 inserts a row for employee Hintz.	事务 1 为雇员 Hintz 插入一行。
SQL> COMMIT;		Transaction 1 commits its work, ending the transaction.	事务 1 提交其工作，以结束事务。
SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Banda','Greene','Hintz'); LAST_NAME SALARY ---------- ---------- Banda 7000 Greene 9500 Hintz	SQL> SELECT last_name, salary FROM employees WHERE last_name IN('Banda','Greene','Hintz'); LAST_NAME SALARY ---------- ---------- Banda 6200 Greene 9900	Session 1 queries the salaries for employees Banda, Greene, and Hintz and sees changes committed by transaction 1. Session 1 does not see the uncommitted Greene update made by transaction 2. Transaction 2 queries the salaries for employees Banda, Greene, and Hintz. Oracle Database read consistency ensures that the Hintz insert and Banda update committed by transaction 1 are *not* visible to transaction 2. Transaction 2 sees its own update to the Banda salary.	会话 1 查询雇员 Banda、Greene、和 Hintz 的薪金，只能看到由事务 1 提交的更改。会话 1 不能看见会话 2 对 Greene 所做的未提交更新。 事务 2 查询雇员 Banda、Greene、和 Hintz 的薪金。Oracle 数据库读取一致性确保由事务 1 所做的插入 Hintz 行和对 Banda 行的更新，对事务 2 不可见。事务 2 只看到自己对 Banda 薪金的更新。
	COMMIT;	Transaction 2 commits its work, ending the transaction.	事务 2 提交其工作，结束该事务。
SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Banda','Greene','Hintz'); LAST_NAME SALARY ---------- ---------- Banda 7000 Greene 9900 Hintz	SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Banda','Greene','Hintz'); LAST_NAME SALARY ---------- ---------- Banda 7000 Greene 9900 Hintz	Both sessions query the salaries for Banda, Greene, and Hintz. Each session sees all committed changes made by transaction 1 and transaction 2.	两个会话分别查询雇员 Banda, Greene, 和 Hintz 的薪金。每个会话将看到事务 1 和事务 2 作出的所有已提交更改。
SQL> UPDATE employees SET salary = 7100 WHERE last_name =		Session 1 begins transaction 3 by updating the Hintz salary. The default isolation level for transaction	会话 1 更新 Hintz 的薪金，开始事务 3。事务 3 的默认隔离级别是已提交读。

'Hintz';		3 is READ COMMITTED.	
	SQL> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;	Session 2 begins transaction 4 and sets it to the SERIALIZABLE isolation level.	会话 2 开始事务 4，并将其设置为可串行化隔离级别。
	SQL> UPDATE employees SET salary = 7200 WHERE last_name = 'Hintz'; -- prompt does not return	Transaction 4 attempts to update the salary for Hintz, but is blocked because transaction 3 locked the Hintz row (see "Row Locks (TX)"). Transaction 4 queues behind transaction 3.	事务 4 尝试更新 Hintz 的薪金，但被阻塞，因为事务 3 锁定了 Hintz 行（请参阅"行锁定 (TX)"）。在事务队列中，事务 4 排在事务 3 的后面。
SQL> COMMIT;		Transaction 3 commits its update of the Hintz salary, ending the transaction.	事务 3 提交了它对 Hintz 工资的更新，以结束事务：。
	UPDATE employees SET salary = 7200 WHERE last_name = 'Hintz' * ERROR at line 1: ORA-08177: can't serialize access for this transaction	The commit that ends transaction 3 causes the Hintz update in transaction 4 to fail with the ORA-08177 error. The problem error occurs because transaction 3 committed the Hintz update *after* transaction 4 began.	结束事务 3 的提交导致事务 4 对 Hintz 更新失败，导致 ORA-08177 错误。该错误之所以发生，是因为事务 3 是在事务 4 开始后提交的
	SQL> ROLLBACK;	Session 2 rolls back transaction 4, which ends the transaction.	会话 2 回滚事务 4，结束该事务。
	SQL> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;	Session 2 begins transaction 5 and sets it to the SERIALIZABLE isolation level.	会话 2 开始事务 5 ，并将其设置为可串行化隔离级别。
	SQL> SELECT last_name, salary FROM employees WHERE last_name IN ('Banda','Greene','Hintz'); LAST_NAME SALARY ---------- ---------- Banda 7100 Greene 9500 Hintz 7100	Transaction 5 queries the salaries for Banda, Greene, and Hintz. The Hintz salary update committed by transaction 3 is visible.	事务 5 查询 Banda、Greene、和 Hintz 的薪金。事务 3 所提交的对 Hintz 薪酬的更新是可见的。

	`SQL> UPDATE employees SET salary =7200 WHERE last_name = 'Hintz';`
	
	`1 row updated.`
		Transaction 5 updates the Hintz salary to a different value. Because the Hintz update made by transaction 3 committed *before* the start of transaction 5, the serialized access problem is avoided.
		
		Note: If a different transaction updated and committed the Hintz row after transaction transaction 5 began, then the serialized access problem would occur again.
			事务 5 将 Hintz 的薪金更新为一个不同值。因为事务 3 对 Hintz 的更新是在事务 5 开始之前提交的，也就避免了序列化的访问问题。
			
			注意： 如果某个其他事务在事务 5 开始后对 Hintz 行进行了更新并提交，则串行化的访问问题又会出现。

	`SQL> COMMIT;`	Session 2 commits the update without any problems, ending the transaction.	会话 2 提交了更新，未遇到任何问题，并结束事务。

See Also:
"Overview of Transaction Control"

另见：
"事务控制概述"

Read-Only Isolation Level

The **read-only isolation** level is similar to the serializable isolation level, but read-only transactions do not permit data to be modified in the transaction unless the user is `SYS`. Thus, read-only transactions are not susceptible to the `ORA-08177` error. Read-only transactions are useful for generating reports in which the contents must be consistent with respect to the time when the transaction began.

Oracle Database achieves read consistency by reconstructing data as needed from the undo segments. Because undo segments are used in a circular fashion, the database can overwrite undo data. Long-running reports run the risk that undo data required for read consistency may have been reused by a different transaction, raising a `snapshot too old` error. Setting an **undo retention period**, which is the minimum amount of time that the database attempts to retain old undo data before overwriting it, appropriately avoids this problem.

See Also:

- "Undo Segments"
- *Oracle Database Administrator's Guide* to learn how to set the undo retention period

只读隔离级别

只读隔离级别类似于可串行化隔离级别，但只读事务不允许数据在事务中被修改，除非该用户是 SYS。因此，只读事务不会受到 ORA-08177 错误的影响。只读事务可用于生成报表，其内容必须与事务开始时保持一致。

Oracle 数据库通过按需从撤销段重建数据，来实现读取一致性。因为撤消段是以一个循环方式使用的，数据库可以覆盖撤销数据。长时间运行的报表可能有一定的风险，读取一致性所需要的撤销数据，可能已被一个不同的事务重用，并抛出快照太旧（`snapshot too old`）错误。设置一个**撤消保留期**，即在旧数据被覆盖之前，数据库尝试保留撤消数据的最短时间，以期避免这一问题。

另见：

- "回滚段"
- 《*Oracle 数据库管理员指南*》了解如何设置撤消保留期

Overview of the Oracle Database Locking

数据库锁定机制概述

Mechanism

A **lock** is a mechanism that prevents **destructive interactions**, which are interactions that incorrectly update data or incorrectly alter underlying data structures, between transactions accessing shared data. Locks play a crucial row in maintaining database concurrency and consistency.

Summary of Locking Behavior

The database maintains several different types of locks, depending on the operation that acquired the lock. In general, the database uses two types of locks: **exclusive locks** and **share locks**. Only one exclusive lock can be obtained on a resource such as a row or a table, but many share locks can be obtained on a single resource.

Locks affect the interaction of readers and writers. A **reader** is a query of a resource, whereas a **writer** is a statement modifying a resource. The following rules summarize the locking behavior of Oracle Database for readers and writers:

- A row is locked only when modified by a writer.

  When a statement updates one row, the transaction acquires a lock for this row only. By locking table data at the row level, the database minimizes contention for the same data. Under normal circumstances the database does not escalate a row lock to the block or table level.
- A writer of a row blocks a concurrent writer of the same row.

  If one transaction is modifying a row, then a row lock prevents a different transaction from modifying the same row simultaneously.
- A reader never blocks a writer.

  Because a reader of a row does not lock it, a writer can modify this row. The only exception is a `SELECT ... FOR UPDATE` statement, which is a special type of `SELECT` statement that *does* lock the row that it is reading.
- A writer never blocks a reader.

  When a row is being changed by a writer, the database uses undo data data to provide readers with a consistent view of the row.

  Note:

锁是一种机制，用来防止多个共同访问共享数据的事务之间的**破坏性交互**，包括不正确地更新数据或不正确地更改基础数据结构。锁在维护数据库并发性和一致性当中扮演着一个关键的角色。

锁定行为总结

数据库维护几种不同类型的锁，这取决于获取锁的操作。一般地，数据库使用两种类型的锁： **独占锁**和**共享锁**。在一个资源（如行或表）上，只能获得一个独占锁，但在单个资源上可以获得很多共享锁。

锁会影响读取者与写入者的交互。读取者是一个对某种资源的查询语句，而写入者是一个对某个资源的修改语句。以下规则总结了 Oracle 数据库中读取者和写入者的锁定行为：

- 一行只有在被某个写入者修改时，才被锁定。

  当一个语句更新某行时，事务只需要获取在该行上的锁。通过在行级别锁定表数据，数据库最小化对同一数据的争用。在正常情况下，数据库不会将行锁升级到块级或表级。

- 一行的写入者，会阻塞在同一行上的并发写入者。

  如果一个事务正在修改某行，则行锁可防止不同的事务同时修改同一行。
- 一个读取者永远不会阻塞一个写入者。

  由于行的读取者不会将它锁定，所以一个写入者可以修改该行。唯一的例外是 `SELECT ... FOR UPDATE` 语句，它是一种特殊类型的 `SELECT` 语句，*的确*会锁定它正在读取的行。

- 一个写入者绝不会阻塞一个读取者。

  当一个行正在被某个写入者更改时，数据库使用撤销数据向读取者提供一个该行的一致视图。

  注意：

Readers of data may have to wait for writers of the same data blocks in very special cases of pending distributed transactions.

See Also:

- *Oracle Database SQL Language Reference* to learn about `SELECT ... FOR UPDATE`
- *Oracle Database Administrator's Guide* to learn about distributed transactions

Use of Locks

In a single-user database, locks are not necessary because only one user is modifying information. However, when multiple users are accessing and modifying data, the database must provide a way to prevent concurrent modification of the same data. Locks achieve the following important database requirements:

- Consistency

  The data a session is viewing or changing must not be changed by other sessions until the user is finished.
- Integrity

  The data and structures must reflect all changes made to them in the correct sequence.

Oracle Database provides data concurrency, consistency, and integrity among transactions through its locking mechanisms. Locking is performed automatically and requires no user action.

The need for locks can be illustrated by a concurrent update of a single row. In the following example, a simple web-based application presents the end user with an employee email and phone number. The application uses an `UPDATE` statement such as the following to modify the data:

```
UPDATE employees
SET    email = ?, phone_number = ?
WHERE  employee_id = ?
AND    email = ?
AND    phone_number = ?
```

In the preceding `UPDATE` statement, the email and phone number values in the `WHERE` clause are the original, unmodified values for the specified employee. This update ensures that the row that the application modifies was not changed after the application last read and displayed it to the

在某些非常特殊的情况下，在挂起的分布式事务中，数据读取者可能需要等待相同数据块上的写入者。

另见：

- 《*Oracle 数据库 SQL 语言参考*》了解 `SELECT ... FOR UPDATE`
- 《*Oracle 数据库 管理员指南*》了解分布式事务

使用锁

在单用户数据库中，锁不是必需的，因为只有一个用户在修改信息。但是，当多个用户在访问和修改数据时，数据库必须提供一种方法，以防止对同一数据进行并发修改。锁实现了以下重要的数据库需求：

- 一致性

  一个会话正在查看或更改的数据不能被其它会话更改，直到用户会话结束。
- 完整性

  数据和结构必须按正确的顺序反映对他们所做的所有更改。

Oracle 数据库通过其锁定机制，提供在多个事务之间的数据并发性、一致性、和完整性。锁定将自动执行，并且不需要用户操作。

对锁的需求，可以由对某个单一行的并发更新来说明。在下面的示例中，一个简单的基于 web 的应用程序，将某个雇员的电子邮件和电话号码呈现给最终用户。应用程序使用如下所示的 `UPDATE` 语句修改该数据：

```
UPDATE employees
SET    email = ?, phone_number = ?
WHERE  employee_id = ?
AND    email = ?
AND    phone_number = ?
```

在前面的 `UPDATE` 语句中， `WHERE` 子句中电子邮件和电话号码是某指定雇员的原始的、 未经修改的值。此更新可确保应用程序修改的行，在应用程序之前读取并向用户显示后，没有被更改过。以这种方式，应用程序可避免丢失更新的数据库问题，即其中一个用户覆盖了另一个用户所做的修改，

user. In this way, the application avoids the lost update database problem in which one user overwrites changes made by another user, effectively losing the update by the second user (Table 9-2 shows an example of a lost update).

实际上丢失了第二个用户所做的更新（表 9-2 显示了一个丢失更新的示例）。

Table 9-4 shows the sequence of events when two sessions attempt to modify the same row in the `employees` table at roughly the same time.

表 9-4 显示两个会话在大致相同的时间，试图修改在雇员表中相同行的事件序列。

Table 9-4 Row Locking Example

表 9-4 行锁定实例

Time	Session 1	Session 2	Explanation	解释

t0	`SELECT employee_id, email,`
	`  phone_number`
	`FROM  hr.employees`
	`WHERE  last_name = 'Himuro';`
	
	`EMPLOYEE_ID EMAIL  PHONE_NUMBER`
	`----------- ------- ------------`
	`  118 GHIMURO 515.127.4565`
			In session 1, the `hr1` user queries `hr.employees` for the Himuro record and displays the employee_id (`118`), email (`GHIMURO`), and phone number (`515.127.4565`) attributes.	在会话 1 中，用户 hr1 查询在表 hr.employees 中的 Himuro 的记录，并显示其员工 ID(118)、 电子邮件（GHIMURO）、和电话号码 (`515.127.4565`) 等属性。

t1		`SELECT employee_id, email,`
		`  phone_number`
		`FROM  hr.employees`
		`WHERE  last_name = 'Himuro';`
		
		`EMPLOYEE_ID EMAIL  PHONE_NUMBER`
		`----------- ------- ------------`
		`  118 GHIMURO 515.127.4565`
			In session 2, the `hr2` user queries `hr.employees` for the Himuro record and displays the employee_id (`118`), email (`GHIMURO`), and phone number (`515.127.4565`) attributes.	在会话 2 中，用户 hr2 查询在表 hr.employees 中的 Himuro 的记录，并显示其 员工 ID (118)、（GHIMURO）和电话号码 (`515.127.4565`) 等属性。

t2	`UPDATE hr.employees`
	`SET phone_number='515.555.1234'`
	`WHERE employee_id=118`
	`AND email='GHIMURO'`
	`AND phone_number='515.127.4565';`
	
	`1 row updated.`
			In session 1, the `hr1` user updates the phone number in the row to `515.555.1234`, which acquires a lock on the `GHIMURO` row.	在会话 1 中，用户 hr1 对 `GHIMURO` 所在行，更新其电话号码为 `515.555.1234`，这会在该行上获取一个行锁。

t3		`UPDATE hr.employees`
		`SET phone_number='515.555.1235'`
		`WHERE employee_id=118`
		`AND email='GHIMURO'`
		`AND phone_number='515.127.4565';`
			In session 2, the `hr2` user attempts to update the same row, but is blocked because `hr1` is currently processing the row.	在会话 2 中，用户 hr2 尝试更新相同的行，但被阻塞，因为 hr1 当前正在处理该行。
				
				hr2 的更新尝试几乎与 hr1 的更新

		-- SQL*Plus does not show -- a row updated message or -- return the prompt.	The attempted update by hr2 occurs almost simultaneously with the hr1 update.	同时发生。
t4	COMMIT; Commit complete.		In session 1, the hr1 user commits the transaction. The commit makes the change for Himuro permanent and unblocks session 2, which has been waiting.	在会话 1 中，用户 hr1 提交其事务。 该提交使对 Himuro 的更改持久化，这也就取消了对一直在等待的会话 2 的阻塞。
t5		0 rows updated.	In session 2, the hr2 user discovers that the GHIMURO row was modified in such a way that it no longer matches its predicate. Because the predicates do not match, session 2 updates no r□cords.	在会话 2 中，用户 hr2 发现 GHIMURO 行被修改，且不再匹配其谓词。 由于该谓词不匹配，会话 2 未能更新任何记录。
t6	UPDATE hr.employees SET phone_number='515.555.1235' WHERE employee_id=118 AND email='GHIMURO' AND phone_number='515.555.1234'; 1 row updated.		In session 1, the hr1 user realizes that it updated the GHIMURO row with the wrong phone number. The user starts a new transaction and updates the phone number in the row to 515.555.1235, which locks the GHIMURO row.	在会话 1 中， hr1 用户发现它错误地更新了 GHIMURO 行的电话号码。该用户启动一个新事务，并将该行的电话号码更新为 515.555.1235，这又将锁定 GHIMURO 行。
t7		SELECT employee_id, email, phone_number FROM hr.employees WHERE last_name = 'Himuro'; EMPLOYEE_ID EMAIL PHONE_NUMBER ----------- ------- ------------ 118 GHIMURO 515.555.1234	In session 2, the hr2 user queries hr.employees for the Himuro record. The record shows the phone number update committed by session 1 at t4. Oracle Database read consistency ensures that session 2 does not see the uncommitted change made at t6.	在会话 2 中，用户 hr2 查询表 hr.employees 中的 Himuro 的记录。记录显示由会话 1 在 t4 时刻对电话号码所做的已提交更新。Oracle 数据库读取一致性确保会话 2 不会看到会话 1 在 t6 时刻所做的未提交更改。
t8		UPDATE hr.employees SET phone_number='515.555.1235'	In session 2, the hr2 user attempts to update the same row,	在会话 2 中，用户 hr2 尝试更新相同的行，但被阻塞，因为 hr1 当前

		WHERE employee_id=118 AND email='GHIMURO' AND phone_number='515.555.1234'; -- SQL*Plus does not show -- a row updated message or -- □eturn the prompt.	but is blocked because hr1 is currently processing the row.	正在处理该行。
t9	ROLLBACK; Rollback complete.		In session 1, the hr1 user rolls back th□transaction, which ends it.	在会话 1 中， hr1 用户回滚其事务并结束之。
t10		1 row updated.	In session 2, the update of the phone number succeeds because the session 1 update was rolled back. The GHIMURO row matches its predicate, so the update succeeds.	在会话 2 中，因为会话 1 的更新被回滚，所以这次对电话号码的更新成功了。GHIMURO 行与它的谓词相匹配，因此更新成功了。
t11		COMMIT; Commit complete.	Session 2 commits the update, ending the transaction.	会话 2 提交其更新，以结束事务。

Oracle Database automatically obtains necessary locks when executing SQL statements. For example, before the database permits a session to modify data, the session must first lock the data. The lock gives the session exclusive control over the data so that no other transaction can modify the locked data until the lock is released.

Because the locking mechanisms of Oracle Database are tied closely to transaction control, application designers need only define transactions properly, and Oracle Database automatically manages locking. Users never need to lock any resource explicitly, although Oracle Database also enables users to lock data manually.

The following sections explain concepts that are important for understanding how Oracle Database achieves data concurrency.

See Also:

Oracle Database PL/SQL Packages and Types Reference to learn about the OWA_OPT_LOCK package, which contains subprograms that can help prevent lost updates

Lock Modes

执行 SQL 语句时，Oracle 数据库自动获取所需的锁。例如，在数据库允许某个会话修改数据之前，该会话必须先锁定数据。锁给予该会话对数据的独占控制权，以便在释放该锁之前，任何其他事务都不可以修改被锁定的数据。

因为数据库的锁定机制与事务控制紧密地绑定在一起，应用程序设计人员只需要正确地定义事务，而数据库会自动管理锁定。虽然数据库也支持用户能够手动锁定数据，但用户从来就不需要显式锁定任何资源。

以下各节解释了理解数据库如何实现数据并发很重要的概念。

另见：

《*Oracle 数据库 PL/SQL 软件包和类型参考*》了解 OWA_OPT_LOCK 软件包，其中包含了有助于防止丢失更新的子程序

锁模式

Oracle Database automatically uses the lowest applicable level of **restrictiveness** to provide the highest degree of data concurrency yet also provide fail-safe data integrity. The less restrictive the level, the more available the data is for access by other users. Conversely, the more restrictive the level, the more limited other transactions are in the types of locks that they can acquire.

Oracle Database uses two modes of locking in a multiuser database:

- Exclusive lock mode

  This mode prevents the associated resource from being shared. A transaction obtains an exclusive lock when it modifies data. The first transaction to lock a resource exclusively is the only transaction that can alter the resource until the exclusive lock is released.
- Share lock mode

  This mode allows the associated resource to be shared, depending on the operations involved. Multiple users reading data can share the data, holding share locks to prevent concurrent access by a writer who needs an exclusive lock. Several transactions can acquire share locks on the same resource.

Assume that a transaction uses a `SELECT ... FOR UPDATE` statement to select a single table row. The transaction acquires an exclusive row lock and a row share table lock. The row lock allows other sessions to modify any rows *other than* the locked row, while the table lock prevents sessions from altering the structure of the table. Thus, the database permits as many statements as possible to execute.

Lock Conversion and Escalation

Oracle Database performs **lock conversion** as necessary. In lock conversion, the database automatically converts a table lock of lower restrictiveness to one of higher restrictiveness.

For example, suppose a transaction issues a `SELECT ... FOR UPDATE` for an employee and later updates the locked row. In this case, the database automatically converts the row share table lock to a row exclusive table lock. A transaction holds exclusive row locks for all rows inserted, updated, or deleted within the transaction. Because row locks are acquired at the highest degree of restrictiveness, no lock conversion is required or performed.

Oracle 数据库自动使用最低适用的**限制**级别，来提供最高程度的数据并发，但还能提供非常安全的数据完整性。限制级别越低、 则有更多的可用数据供其他用户访问。相反，限制级别越高，则其他事务为获取其所需的锁类型就将遭受更多的限制。

Oracle 数据库在多用户数据库中使用两种锁定模式：

- 独占锁模式

  此模式可防止相关资源被共享。当一个事务修改数据时，获取一个独占锁。直到独占锁被释放之前，第一个以独占方式锁定资源的事务是唯一可以更改资源的事务，。

- 共享锁模式

  取决于所涉及的操作，此模式允许相关资源被共享。读取数据的多个用户可以共享数据，并持有共享锁，以防止企图获取独占锁的写入者并发访问。多个事务可以同时获取在同一资源上的共享锁。

假定一个事务使用 `SELECT ... FOR UPDATE` 语句选择一个单一表行。该事务获取一个独占行锁和行共享表锁。行锁允许其他会话修改除锁定行之外的任何行，而表锁可防止其它会话更改表结构。这样，数据库允许尽可能多的语句得以执行。

锁转换和锁升级

Oracle 数据库在必要时执行**锁转换**。在锁转换中，数据库自动将较低限制的表锁转换为较高限制的其它锁定。

例如，假设事务对某雇员发出 `SELECT ... FOR UPDATE`，并之后更新了该锁定行。在这种情况下，数据库会自动将行共享表锁转换为行独占表锁。一个事务在该事务中所有执行插入、 更新、或删除的行上持有行独占锁。因为行锁是在最高程度限制下获得的，因此不要求锁转换，也不执行锁转换。

Lock conversion is different from **lock escalation**, which occurs when numerous locks are held at one level of granularity (for example, rows) and a database raises the locks to a higher level of granularity (for example, table). If a user locks many rows in a table, then some databases automatically escalate the row locks to a single table. The number of locks decreases, but the restrictiveness of what is locked increases.

锁转换不同于**锁升级**，锁升级发生在当某个粒度级别持有许多锁（例如行），数据库将其提高到更高粒度级别（例如表）。如果一个用户锁定了一个表中的许多行，则某些数据库自动将行锁升级到单个表锁。锁的数量减少了，但被锁定的东西却增加了。

Oracle Database never escalates locks. Lock escalation greatly increases the likelihood of deadlocks. Assume that a system is trying to escalate locks on behalf of transaction 1 but cannot because of the locks held by transaction 2. A deadlock is created if transaction 2 also requires lock escalation of the same data before it can proceed.

Oracle 数据库永远不会升级锁。锁升级极大地增加了死锁的可能性。假定一个系统尝试升级事务 1 中的锁，但因为事务 2 持有该锁，故不能成功。如果事务 2 在它可以继续操作之前也需要在相同的数据上进行锁升级，则将发生一个死锁。

Lock Duration

Oracle Database automatically releases a lock when some event occurs so that the transaction no longer requires the resource. In most cases, the database holds locks acquired by statements within a transaction for the duration of the transaction. These locks prevent destructive interference such as dirty reads, lost updates, and destructive DDL from concurrent transactions.

锁持续时间

当某些事件发生，使事务不再需要资源时，Oracle 数据库会自动释放锁。在大多数的情况下，数据库持有语句所获取的锁，直至该语句所在事务的整个持续期间结束。这些锁可以防止破坏性干扰，如多个并发事务中的脏读、更新丢失、和破坏性 DDL。

Note:

A table lock taken on a child table because of an unindexed foreign key is held for the duration of the statement, not the transaction. Also, as explained in "Overview of User-Defined Locks", the `DBMS_LOCK` package enables user-defined locks to be released and allocated at will and even held over transaction boundaries.

注意：

因为未索引外键而在子表上发生的表锁，会在语句持续期间持有，而不是在事务持续期间。此外，如"用户定义锁概述"所述， DBMS_LOCK 包使得用户定义的锁可以按你的意志被释放或分配，甚至跨越事务边界而持有。

Oracle Database releases all locks acquired by the statements within a transaction when it commits or rolls back. Oracle Database also releases locks acquired after a savepoint when rolling back to the savepoint. However, only transactions not waiting for the previously locked resources can acquire locks on the now available resources. Waiting transactions continue to wait until after the original transaction commits or rolls back completely (see Table 10-2 for an example).

当 Oracle 数据库提交或回滚事务时，会释放事务内的语句获取的所有锁。当 Oracle 数据库回滚至保存点时，也会释放该保存点之后所获取的锁。但是，只有不在等待以前被锁定的资源的事务，可以获取现有可用资源上的锁。等待中的事务会继续等待，直到原始事务提交或完全回滚（见表 10-2 为例）。

See Also:

"Rollback to Savepoint"

另见：

"回滚到保存点"

Locks and Deadlocks

A **deadlock** is a situation in which two or more users are waiting for data locked by each other. Deadlocks prevent some transactions from continuing to work.

锁和死锁

死锁情况发生在两个或多个用户都在等待被对方锁定的数据时。死锁会阻止某些事务继续工作。

Oracle Database automatically detects deadlocks and resolves them by rolling back one statement involved in the deadlock, releasing one set of the conflicting row locks. The database returns a corresponding message to the transaction that undergoes statement-level rollback. The statement rolled back belongs to the transaction that detects the deadlock. Usually, the signalled transaction should be rolled back explicitly, but it can retry the rolled-back statement after waiting.

Table 9-5 illustrates two transactions in a deadlock.

Oracle 数据库自动检测死锁，并通过回滚死锁中的一个语句以释放其冲突行锁，来解决死锁。数据库向遭遇语句级回滚的事务返回一条相应的消息。被回滚语句属于检测到死锁的事务。通常，收到通知的事务应该明确回滚，但它可以等待一会之后重试被回滚的语句。

表 9-5 说明了在死锁中的两个事务。

Table 9-5 Deadlock Example

表 9-5 死锁示例

Time	Session 1	Session 2	Explanation	解释

t0	`SQL> UPDATE employees`
	`  SET salary = salary*1.1`
	`  WHERE employee_id = 100;`
	
	`1 row updated.`
		`SQL> UPDATE employees`
		`  SET  salary = salary*1.1`
		`  WHERE employee_id = 200;`
		
		`1 row updated.`
			Session 1 starts transaction 1 and updates the salary for employee 100. Session 2 starts transaction 2 and updates the salary for employee 200. No problem exists because each transaction locks only the row that it attempts to update.	会话 1 启动事务 1，并更新雇员 100 的薪酬。会话 2 启动事务 2，并更新雇员 200 的薪酬。没有问题存在，因为每个事务只锁定它会尝试更新的行。

t1	`SQL> UPDATE employees`
	`  SET salary = salary*1.1`
	`  WHERE employee_id = 200;`
	
	`-- prompt does not return`
		`SQL> UPDATE employees`
		`  salary = salary*1.1`
		`  WHERE employee_id = 100;`
		
		`-- prompt does not return`
			Transaction 1 attempts to update the employee 200 row, which is currently locked by transaction 2. Transaction 2 attempts to update the employee 100 row, which is currently locked by transaction 1.
			
			A deadlock results because neither transaction can obtain the resource it needs to proceed or terminate. No matter how long each transaction waits, the conflicting locks are held.
				事务 1 试图更新员工 200 的行，这当前正由事务 2 锁定。事务 2 试图更新雇员 100 的行，这当前正由事务 1 锁定。
				
				死锁发生了，因为任何一个事务都不能获取其得以继续或终止的资源。无论每个事务等待多久，冲突锁依然被对方持有。

t2	`UPDATE employees`
	`  *`
	`ERROR at line 1:`
	`ORA-00060: deadlock detected`
	`while waiting for resource`
			Transaction 1 signals the deadlock and rolls back the `UPDATE` statement issued at t1. However, the update made at t0 is not rolled back. The prompt is returned in	事务 1 发出死锁信号，并回滚在 t1 时刻发出的 UPDATE 语句。但是，在 t0 时刻所做的更新不会回滚。会话 1 返回到提示符。
				
				注意： 死锁中只有一个会话会真正

	SQL>		session 1. **Note:** Only one session in the deadlock actually gets the deadlock error, but either session could get the error.	获取死锁错误，但这两个会话中的任何一个都有可能出现错误。
t3	SQL> COMMIT; Commit complete.		Session 1 commits the update made at t0, ending transaction 1. The update unsuccessfully attempted at t1 is not committed.	会话提交了 t0 时刻所做的更新，以结束事务 1。在 t1 时刻尝试的更新不成功，未能提交。
t4		1 row updated. SQL>	The update at t1 in transaction 2, which was being blocked by transaction 1, is executed. The prompt is returned.	执行事务 2 在 t1 时刻所做的更新，之前被事务 1 阻塞，现在得以执行。并返回提示符。
t5		SQL> COMMIT; Commit complete.	Session 2 commits the updates made at t0 and t1, which ends transaction 2.	会话 2 提交其在 t0 和 t1 时刻所做的更新，结束事务 2。

Deadlocks most often occur when transactions explicitly override the default locking of Oracle Database. Because Oracle Database does not escalate locks and does not use read locks for queries, but does use row-level (rather than page-level) locking, deadlocks occur infrequently.

See Also:

- "Overview of Manual Data Locks"
- *Oracle Database Advanced Application Developer's Guide* to learn how to handle deadlocks when you lock tables explicitly

当事务显式覆盖 Oracle 数据库的默认锁定时，最经常发生死锁。由于数据库不会升级锁，也不对查询使用读锁定，而是使用行级 （而不是页级别）锁定，所以很少会出现死锁。

另见：

- "手动数据锁概述"
- 《*Oracle 数据库高级应用开发指南*》了解当您显式锁定表时，如何处理死锁，

Overview of Automatic Locks

Oracle Database automatically locks a resource on behalf of a transaction to prevent other transactions from doing something that requires exclusive access to the same resource. The database automatically acquires different types of locks at different levels of restrictiveness depending on the resource and the operation being performed.

Note:

The database never locks rows when performing simple reads.

Oracle Database locks are divided into the following categories.

Lock	Description

自动锁的概述

Oracle 数据库会为事务自动锁定资源，以防止其他事务进行某些需要独占访问同一资源的操作。数据库在不同的限制级别自动获取不同类型的锁，这依赖于不同的资源和正在执行的操作。

注意：

执行简单读取时，数据库绝不会锁定行。

Oracle 数据库锁分为以下几类。

锁	描述

DML Locks	Protect data. For example, table locks lock entire tables, while row locks lock selected rows. See "DML Locks".
DDL Locks	Protect the structure of schema objects—for example, the dictionary definitions of tables and views. See "DDL Locks".
System Locks	Protect internal database structures such as data files. Latches, mutexes, and internal locks are entirely automatic. See "System Locks".

DML Locks

A DML lock, also called a **data lock**, guarantees the integrity of data accessed concurrently by multiple users. For example, a DML lock prevents two customers from buying the last copy of a book available from an online bookseller. DML locks prevent destructive interference of simultaneous conflicting DML or DDL operations.

DML statements automatically acquire the following types of locks:

- Row Locks (TX)
- Table Locks (TM)

In the following sections, the acronym in parentheses after each type of lock or lock mode is the abbreviation used in the Locks Monitor of Oracle Enterprise Manager (Enterprise Manager). Enterprise Manager might display TM for any table lock, rather than indicate the mode of table lock (such as RS or SRX).

See Also:

"Oracle Enterprise Manager"

Row Locks (TX)

A **row lock**, also called a **TX lock**, is a lock on a single row of table. A transaction acquires a row lock for each row modified by an `INSERT`, `UPDATE`, `DELETE`, `MERGE`, or `SELECT ... FOR UPDATE` statement. The row lock exists until the transaction commits or rolls back.

Row locks primarily serve as a queuing mechanism to prevent two transactions from modifying the same row. The database always locks a modified row in exclusive mode so that other transactions cannot modify the row until the transaction holding the lock commits or rolls back. Row locking provides the finest grain locking possible and so provides the best possible concurrency and throughput.

Note:

If a transaction terminates because of database instance failure, then

DML 锁	保护数据。例如，表锁锁定整个表，而行锁锁定所选的行。请参阅"DML 锁"。
DDL 锁	保护模式对象的结构 ——例如，表和视图的数据字典定义。请参阅"DDL 锁"。
系统锁	保护内部数据库结构，如数据文件。闩锁、 互斥体、和内部锁是完全自动的。请参阅"系统锁"。

DML 锁

DML 锁，也称为**数据锁**，确保由多个用户并发访问的数据的完整性。例如， DML 锁可防止两个客户从一个在线书店购买某一本书所剩的最后一个拷贝。DML 锁也可以防止多个相互冲突的 DML 或 DDL 操作产生破坏性干扰。

DML 语句自动获取下列类型的锁：

- 行锁 (TX)
- 表锁 (TM)

在下面几节，在每种类型的锁或锁模式后的括号中，首字母缩写词是用于在 Oracle 企业管理器 （企业管理器）的锁监视器中的缩写。企业管理器中可能将任何的表锁显示为 TM，而不显示表锁的模式（如 RS 或 SRX)。

另见：

"Oracle 企业管理器"

行锁(TX)

行锁，也称为 **TX 锁**，是一个表中单个行上的锁。一个事务在被 `INSERT`、`UPDATE`、`DELETE`、`MERGE`、或 `SELECT ... FOR UPDATE` 等语句所修改的每一行上获取一个行锁。行锁一直存在直到事务提交或回滚。

行锁主要作为一种排队的机制，以防止两个事务修改相同的行。数据库始终以独占模式锁定修改的行，以便其他事务不能修改该行，直到持有锁的事务提交或回滚。行锁定提供了近乎最细粒度的锁定，并因此提供了近乎最佳的并发性和吞吐量。

注意：

如果一个事务因为数据库实例失效而终止，会先进行块级恢复以使行可用，

block-level recovery makes a row available before the entire transaction is recovered.

If a transaction obtains a lock for a row, then the transaction also acquires a lock for the table containing the row. The table lock prevents conflicting DDL operations that would override data changes in a current transaction. Figure 9-2 illustrates an update of the third row in a table. Oracle Database automatically places an exclusive lock on the updated row and a subexclusive lock on the table.

Figure 9-2 Row and Table Locks

Description of "Figure 9-2 Row and Table Locks"

Row Locks and Concurrency

Table 9-6 illustrates how Oracle Database uses row locks for concurrency. Three sessions query the same rows simultaneously. Session 2 and 3 proceed to make uncommitted updates to different rows, while session 3 makes no updates. Each session sees its own uncommitted updates but not the uncommitted updates of any other session.

Table 9-6 Data Concurrency Example

之后进行整个事务恢复。

如果一个事务在某行上获取了一个锁，则该事务也将在包含该行的表上获取一个锁。表锁可防止冲突性的 DDL 操作在当前事务中会覆盖数据更改。图 9-2 演示了在一个表中第三行上的更新。Oracle 数据库将自动在更新行上置一个独占锁，且在表上置一个次独占锁。

图 9-2 行锁和表锁

Description of "Figure 9-2 Row and Table Locks"

行锁和并发

表 9-6 说明了 Oracle 数据库如何在并发事务中使用行锁。3 个会话同时查询相同的行。会话 2 和会话 3 对不同的行进行未提交更新，而会话 3 没有做任何更新。每个会话可以看到自己的未提交更新，但看不到其他会话中所做的任何未提交更新。

表 9-6 数据并发示例

Time	Session 1	Session 2	Session 3	Explanation	解释
t0	`SELECT employee_id, salary FROM employees WHERE employee_id`	`SELECT employee_id, salary FROM employees WHERE employee_id`	`SELECT employee_id, salary FROM employees WHERE employee_id`	Three different sessions simultaneously query the ID and salary of employees	三个不同的会话同时查询 ID 为 100 和 101 的雇员的薪金。每个查询所返回的结果

	IN (100, 101); EMPLOYEE_ID SALARY ----------- ------ 100 512 101 600	IN (100, 101); EMPLOYEE_ID SALARY ----------- ------ 100 512 101 600	IN (100, 101); EMPLOYEE_ID SALARY ----------- ------ 100 512 101 600	100 and 101. The results returned by each query are identical.	是相同的。
t1	UPDATE hr.employees SET salary=salary+100 WHERE employee_id=100;			Session 1 updates the salary of employee 100, but does not commit. In the update, the writer acquires a row-level lock for the updated row only, thereby preventing other writers from modifying this row.	会话 1 更新雇员 100 的薪金，但不提交。对此更新，写入者只需要对更新的行获得一个行级锁，从而防止其他写入者修改这一行。
t2	SELECT employee_id, salary FROM employees WHERE employee_id IN (100, 101); EMPLOYEE_ID SALARY ----------- ------ 100 **612** 101 600	SELECT employee_id, salary FROM employees WHERE employee_id IN (100, 101); EMPLOYEE_ID SALARY ----------- ------ 100 512 101 600	SELECT employee_id, salary FROM employees WHERE employee_id IN (100, 101); EMPLOYEE_ID SALARY ----------- ------ 100 512 101 600	Each session simultaneously issues the original query. Session 1 shows the salary of 612 resulting from the t1 update. The readers in session 2 and 3 return rows immediately and do not wait for session 1 to end its transaction. The database uses multiversion read consistency to show the salary as it existed before the update in session 1.	每个会话同时发出原来的查询。会话 1 显示了 t1 时刻的薪金更新结果 612。会话 2 和会话 3 中的读取者立即返回行，并不等待会话 1 结束其事务。数据库使用多版本读取一致性将薪金显示为会话 1 更改它之前的样子。
t3		UPDATE hr.employees SET salary=salary+100 WHERE employee_id=101;		Session 2 updates the salary of employee 101, but does not commit the transaction. In the update, the writer acquires a row-level lock for the updated row only, preventing other writers from modifying this row.	会话 2 更新雇员 101 的工资，但不提交事务。对此更新，写入者只需要对更新的行获得一个行级锁，从而防止其他写入者修改这一行。
t4	SELECT employee id,	SELECT employee id,	SELECT employee id,	Each session simultaneously	每个会话同时发出原来的查

	salary FROM employees WHERE employee_id IN (100, 101); EMPLOYEE_ID SALARY ----------- ------ 100 **612** 101 600	salaryFROM employees WHERE employee_id IN (100, 101); EMPLOYEE_ID SALARY ----------- ------ 100 512 101 **700**	salary FROM employees WHERE employee_id IN (100, 101); EMPLOYEE_ID SALARY ----------- ------ 100 512 101 600	issues the original query. Session 1 shows the salary of 612 resulting from the t1 update, but not the salary update for employee 101 made in session 2. The reader in session 2 shows the salary update made in session 2, but not the salary update made in session 1. The reader in session 3 uses read consistency to show the salaries before modification by session 2 and 3.	询。会话 1 显示了 t1 时刻的薪金更新结果 612，而不显示在会话 2 中为员工 101 所做的的薪金更新。会话 2 中的读取者显示在会话 2 中所做的薪金更新，但不显示会话 1 中所做的薪金更新。在会话 3 中的读取者使用读取一致性来显示会话 1 和 2 的修改之前的薪金。

See Also:

- *Oracle Database SQL Language Reference*
- *Oracle Database Reference* to learn about `V$LOCK`

Storage of Row Locks

Unlike some databases, which use a lock manager to maintain a list of locks in memory, Oracle Database stores lock information in the data block that contains the locked row.

The database uses a queuing mechanism for acquisition of row locks. If a transaction requires a lock for an unlocked row, then the transaction places a lock in the data block. Each row modified by this transaction points to a copy of the transaction ID stored in the block header (see "Overview of Data Blocks").

When a transaction ends, the transaction ID remains in the block header. If a different transaction wants to modify a row, then it uses the transaction ID to determine whether the lock is active. If the lock is active, then the session asks to be notified when the lock is released. Otherwise, the transaction acquires the lock.

See Also:

Oracle Database Reference to learn about `V$TRANSACTION`

Table Locks (TM)

A **table lock**, also called a **TM lock**, is acquired by a transaction when a

另见：

- *《Oracle 数据库 SQL 语言参考》*
- 《*Oracle 数据库参考*》了解 V$LOCK

行锁存储

与某些数据库使用锁管理器在内存中维护一个锁列表不同，Oracle 数据库将锁信息存储在包含锁定行的数据块中。

数据库使用排队机制来获取行锁。如果事务需要为某个未锁定的行获取一个锁，则事务将在数据块上置一个锁。被此事务修改的每个行指向存储在块头中的事务 ID 的一个副本（请参阅"数据块概述"）。

当事务结束时，事务 ID 仍然保留在块头中。如果一个不同的事务想要修改某行，则它使用该事务 ID 来确定锁是否处于活动状态。如果锁是活动的，则会话要求该锁被释放时得到通知。否则，事务获取该锁。

另见：

《*Oracle 数据库参考*》了解 `V$TRANSACTION`

表锁(TM)

表锁，也称为 **TM 锁**，当一个表被 `INSERT`、`UPDATE`、`DELETE`、

table is modified by an INSERT, UPDATE, DELETE, MERGE, SELECT with the FOR UPDATE clause, or LOCK TABLE statement. DML operations require table locks to reserve DML access to the table on behalf of a transaction and to prevent DDL operations that would conflict with the transaction.

A table lock can be held in any of the following modes:

- Row Share (RS)

  This lock, also called a **subshare table lock (SS)**, indicates that the transaction holding the lock on the table has locked rows in the table and intends to update them. A row share lock is the least restrictive mode of table lock, offering the highest degree of concurrency for a table.
- Row Exclusive Table Lock (RX)

  This lock, also called a **subexclusive table lock (SX)**, generally indicates that the transaction holding the lock has updated table rows or issued SELECT ... FOR UPDATE. An SX lock allows other transactions to query, insert, update, delete, or lock rows concurrently in the same table. Therefore, SX locks allow multiple transactions to obtain simultaneous SX and subshare table locks for the same table.
- Share Table Lock (S)

  A share table lock held by a transaction allows other transactions to query the table (without using SELECT ... FOR UPDATE), but updates are allowed only if a single transaction holds the share table lock. Because multiple transactions may hold a share table lock concurrently, holding this lock is not sufficient to ensure that a transaction can modify the table.
- Share Row Exclusive Table Lock (SRX)

  This lock, also called a **share-subexclusive table lock (SSX)**, is more restrictive than a share table lock. Only one transaction at a time can acquire an SSX lock on a given table. An SSX lock held by a transaction allows other transactions to query the table (except for SELECT ... FOR UPDATE) but not to update the table.

MERGE、带 FOR UPDATE 子句的 SELECT 等修改时，由相关事务获取该锁。DML 操作需要表锁来为事务保护 DML 对表的访问，并防止可能与事务冲突的 DDL 操作。

表锁可能以下列模式之一持有：

- 行共享 (RS)

  这种锁也被称为**子共享表锁 (SS)**，表示在表上持有锁的事务在表中有被锁定的行，并打算更新它们。行共享锁是限制最少的表级锁模式，提供在表上最高程度的并发性。

- 行独占表锁 (RX)

  这种锁也被称为**子独占表锁（SX）**，通常表示持有锁的事务已更新了表行或发出了 SELECT ... FOR UPDATE。一个 SX 锁允许其他事务并发地查询、 插入、 更新、 删除、或锁定在同一个表中的（其它？）行。因此，SX 锁允许多个事务对同一个表同时获得 SX 和子共享表锁。

- 共享表锁(S)

  由某个事务拥有的共享表锁允许其他事务查询（而不使用 SELECT ... FOR UPDATE)，但是更新操作只能在仅有单个事务持有共享表锁时才允许。因为可能有多个事务同时持有共享表锁，所以持有此锁不足以确保一个事务可以修改该表。

- 共享行独占表锁 (SRX)

  这种锁也称为**共享子独占表锁（SSX）**， 比共享表锁的限制性更强。一次只能有一个事务可以获取给定的表上的 SSX 锁。由某个事务拥有的 SSX 锁允许其他事务查询该表 （除 SELECT ... FOR UPDATE）但不能更新该表。

- Exclusive Table Lock (X)

  This lock is the most restrictive, prohibiting other transactions from performing any type of DML statement or placing any type of lock on the table.

See Also:

- *Oracle Database SQL Language Reference*
- *Oracle Database Advanced Application Developer's Guide* to learn more about table locks

Locks and Foreign Keys

Oracle Database maximizes the concurrency control of parent keys in relation to dependent foreign keys. Locking behavior depends on whether foreign key columns are indexed. If foreign keys are not indexed, then the child table will probably be locked more frequently, deadlocks will occur, and concurrency will be decreased. For this reason foreign keys should almost always be indexed. The only exception is when the matching unique or primary key is never updated or deleted.

Locks and Unindexed Foreign Keys

When both of the following conditions are true, the database acquires a full table lock on the child table:

- No index exists on the foreign key column of the child table.

- A session modifies a primary key in the parent table (for example, deletes a row or modifies primary key attributes) or merges rows into the parent table. Inserts into the parent table do not acquire table locks on the child table.

Suppose that `hr.departments` table is a parent of `hr.employees`, which contains the unindexed foreign key `department_id`. Figure 9-3 shows a session modifying the primary key attributes of department 60 in the `departments` table.

Figure 9-3 Locking Mechanisms with Unindexed Foreign Key

- 独占表锁(X)

  这种锁是最严格的锁，禁止其他事务执行任何类型的 DML 语句，或在表上放置任何类型的锁。

另见：

- 《*Oracle 数据库 SQL 语言参考*》
- 《*Oracle 数据库高级应用开发指南*》了解表锁的详细信息

锁和外键

Oracle 数据库将与外键相关的父键的并发控制最大化。锁定行为将取决于外键列是否已被索引。如果外键未被索引，则子表将更可能被频繁锁定，或发生死锁，并降低并发性。为此，外键几乎总是应该被索引的。唯一的例外是当匹配的唯一键或主键永远不会被更新或删除。

锁定和未索引外键

当下列两个条件成立时，数据库在子表上获取一个全表锁定：

- 在子表的外键列上没有索引存在。

- 会话修改了父表中的主键 （例如，删除了行或修改了主键属性）或将行合并到父表。在父表中插入行不获取子表表锁。

假设 hr.departments 表是 hr.employees 表的父表，hr.employees 表包含未索引的外键 department_id。图 9-3 显示了一个修改 `departments` 表中的部门 60 的主键属性的会话。

图 9-3 未索引外键时的锁定机制

Description of "Figure 9-3 Locking Mechanisms with Unindexed Foreign Key"

In Figure 9-3, the database acquires a full table lock on `employees` during the primary key modification of department 60. This lock enables other sessions to query but not update the `employees` table. For example, employee phone numbers cannot be updated. The table lock on `employees` releases immediately after the primary key modification on

Description of "Figure 9-3 Locking Mechanisms with Unindexed Foreign Key"

在图 9-3 中，数据库在部门 60 的主键修改过程中，在 `employees` 表上获取了一个全表锁定。此锁使其他会话可以查询 `employees` 表，但不能更新 `employees` 表。例如，雇员的电话号码不能更新。一旦 `departments` 表上的主键修改完成后，`employees` 表上的表锁立即释放。如果在部门中有多个行被修改主键，则在 `departments` 表中每修改一行，就需要在

the departments table completes. If multiple rows in departments undergo primary key modifications, then a table lock on employees is obtained and released once for each row that is modified in departments.

employees 表上获取并释放一次表锁。

Note:

DML on a child table does not acquire a table lock on the parent table.

注意：

在子表上的 DML 不必获取父表中的一个表锁。

Locks and Indexed Foreign Keys

When both of the following conditions are true, the database does *not* acquire a full table lock on the child table:

- A foreign key column in the child table is indexed.
- A session modifies a primary key in the parent table (for example, deletes a row or modifies primary key attributes) or merges rows into the parent table.

A lock on the parent table prevents transactions from acquiring exclusive table locks, but does not prevent DML on the parent *or* child table during the primary key modification. This situation is preferable if primary key modifications occur on the parent table while updates occur on the child table.

Figure 9-4 shows child table employees with an indexed department_id column. A transaction deletes department 280 from departments. This deletion does not cause the database to acquire a full table lock on the employees table as in the scenario described in "Locks and Unindexed Foreign Keys".

Figure 9-4 Locking Mechanisms with Indexed Foreign Key

锁和索引外键

当下面两个条件成立时，数据库不会获取子表上的全表锁定：

- 子表中的外键列已被索引。
- 会话修改了父表中的主键 （例如，删除了行或修改了主键属性）或将行合并到父表。

父表上的锁可以防止其它事务获取独占表锁，但在主键修改过程中不会阻止父表或子表上的 DML。主键修改发生时，我们仍可以修改子表，这种情形正是我们想要的。

图 9-4 显示一个具有索引列 department_id 的子表 employees。事务从 departments 表中删除部门 280。此删除操作不会导致数据库在 employees 表上获取全表锁定。

图 9-4 具有索引外键时的锁定机制

Description of "Figure 9-4 Locking Mechanisms with Indexed Foreign Key"

If the child table specifies `ON DELETE CASCADE`, then deletions from the parent table can result in deletions from the child table. For example, the deletion of department 280 can cause the deletion of records from `employees` for employees in the deleted department. In this case, waiting and locking rules are the same as if you deleted rows from the

Description of "Figure 9-4 Locking Mechanisms with Indexed Foreign Key"

如果在子表指定了 `ON DELETE CASCADE` 选项，则从父表删除记录会导致子表中删除相应记录。例如，删除部门 280 会导致从 `employees` 表中删属于被删除部门的雇员。在这种情况下，等待和锁定规则等同于您先从父表中删除行，然后从子表删除相应行。

child table after deleting rows from the parent table.

See Also:

- "Foreign Key Constraints"
- "Overview of Indexes"

DDL Locks

A **data dictionary (DDL) lock** protects the definition of a schema object while an ongoing DDL operation acts on or refers to the object. Only individual schema objects that are modified or referenced are locked during DDL operations. The database never locks the whole data dictionary.

Oracle Database acquires a DDL lock automatically on behalf of any DDL transaction requiring it. Users cannot explicitly request DDL locks. For example, if a user creates a stored procedure, then Oracle Database automatically acquires DDL locks for all schema objects referenced in the procedure definition. The DDL locks prevent these objects from being altered or dropped before procedure compilation is complete.

Exclusive DDL Locks

An **exclusive DDL lock** prevents other sessions from obtaining a DDL or DML lock. Most DDL operations, except for those described in "Share DDL Locks", require exclusive DDL locks for a resource to prevent destructive interference with other DDL operations that might modify or reference the same schema object. For example, `DROP TABLE` is not allowed to drop a table while `ALTER TABLE` is adding a column to it, and vice versa.

Exclusive DDL locks last for the duration of DDL statement execution and automatic commit. During the acquisition of an exclusive DDL lock, if another DDL lock is held on the schema object by another operation, then the acquisition waits until the older DDL lock is released and then proceeds.

Share DDL Locks

A **share DDL lock** for a resource prevents destructive interference with conflicting DDL operations, but allows data concurrency for similar DDL operations.

For example, when a `CREATE PROCEDURE` statement is run, the containing transaction acquires share DDL locks for all referenced tables. Other transactions can concurrently create procedures that reference the same tables and acquire concurrent share DDL locks on the same tables, but no transaction can acquire an exclusive DDL lock on any referenced

另见：

- "外键约束"
- "索引概述"

DDL 锁

当某个运行中的 DDL 操作正在操作或引用某模式对象时，**数据字典(DDL)锁**保护该模式对象的定义。在 DDL 操作的过程中，只有被修改或引用的单个模式对象被锁定。数据库绝不会锁定整个数据字典。

Oracle 数据库将为任何要求锁的 DDL 事务自动获取 DDL 锁。用户不能显式请求 DDL 锁。例如，如果用户创建一个存储过程，则数据库自动为过程定义中引用的所有模式对象获取 DDL 锁。DDL 锁防止在过程编译完成之前，这些对象被更改或删除。

独占 DDL 锁

独占 DDL 锁可防止其他会话获取 DDL 或 DML 锁。除了那些在"共享 DDL 锁"中所述操作之外，绝大多数 DDL 操作需要对资源获取独占锁，以防止和其他可能会修改或引用相同模式对象的 DDL 之间的破坏性干扰。例如，当 `ALTER TABLE` 正在将一列添加到表时，不允许 `DROP TABLE` 删除表，反之亦然。

独占 DDL 锁在整个 DDL 语句执行期间一直持续，并自动提交。在独占 DDL 锁获取过程中，如果另一个操作在该模式对象上持有另一个 DDL 锁，则这个锁获取将一直等待，直到前一个 DDL 锁被释放，才能继续。

共享 DDL 锁

在资源上的**共享 DDL 锁**可防止与冲突的 DDL 操作发生破坏性干扰，但允许类似的 DDL 操作的数据并发。

例如，当 `CREATE PROCEDURE` 语句运行时，所在事务将为所有被引用的表获取共享 DDL 锁。其他事务可以同时创建引用相同表的过程，并在相同的表上同时获得共享 DDL 锁，但没有任何事务能在任何被引用表上获取独占 DDL 锁。

table.
A share DDL lock lasts for the duration of DDL statement execution and automatic commit. Thus, a transaction holding a share DDL lock is guaranteed that the definition of the referenced schema object remains constant during the transaction.

共享 DDL 锁在整个 DDL 语句执行期间持续存在，并自动提交。因此，持有一个共享 DDL 锁的事务，可保证在事务过程中，被引用模式对象的定义保持不变。

Breakable Parse Locks

A **parse lock** is held by a SQL statement or PL/SQL program unit for each schema object that it references. Parse locks are acquired so that the associated shared SQL area can be invalidated if a referenced object is altered or dropped. A parse lock is called a **breakable parse lock** because it does not disallow any DDL operation and can be broken to allow conflicting DDL operations.

可中断的解析锁

SQL 语句或 PL/SQL 程序单元，为每个被其引用的模式对象持有一个**解析锁**。获取解析锁的目的是，如果被引用的对象被更改或删除，可以使相关联的共享 SQL 区无效。解析锁被称为可中断的解析锁，因为它并不禁止任何 DDL 操作，并可以被打破以允许冲突的 DDL 操作。

A parse lock is acquired in the shared pool during the parse phase of SQL statement execution. The lock is held as long as the shared SQL area for that statement remains in the shared pool.

解析锁是在执行 SQL 语句的分析阶段，在共享池中获取的。只要该语句的共享 SQL 区仍保留在共享池中，该锁就一直被持有。

See Also:

"Shared Pool"

另见：

"共享池"

System Locks

Oracle Database uses various types of system locks to protect internal database and memory structures. These mechanisms are inaccessible to users because users have no control over their occurrence or duration.

系统锁

Oracle 数据库使用各种类型的系统锁，来保护数据库内部和内存结构。由于用户不能控制其何时发生或持续多久，这些机制对于用户几乎是不可访问的。

Latches

Latches are simple, low-level serialization mechanisms that coordinate multiuser access to shared data structures, objects, and files. Latches protect shared memory resources from corruption when accessed by multiple processes. Specifically, latches protect data structures from the following situations:

- Concurrent modification by multiple sessions
- Being read by one session while being modified by another session
- Deallocation (aging out) of memory while being accessed

闩锁

闩锁是简单、 低级别的串行化机制，用于协调对共享数据结构、 对象、和文件的多用户访问。闩锁防止共享内存资源被多个进程访问时遭到破坏。具体而言，闩锁在以下情况下保护数据结构：

- 被多个会话同时修改
- 正在被一个会话读取时，又被另一个会话修改
- 正在被访问时，其内存被释放（换出）

Typically, a single latch protects multiple objects in the SGA. For example, background processes such as DBW*n* and LGWR allocate memory from the shared pool to create data structures. To allocate this memory, these processes use a shared pool latch that serializes access to prevent two processes from trying to inspect or modify the shared pool simultaneously. After the memory is allocated, other processes may need to access shared pool areas such as the library cache, which is required for parsing. In this

通常，一个单一的闩锁保护 SGA 中的多个对象。例如，后台进程（如 DBWn 和 LGWR）从共享池分配内存来创建数据结构。为分配此内存，这些进程使用共享池闩锁来串行化对内存的访问，以防止两个进程同时尝试检查或修改共享池。内存分配后，其他进程可能需要访问共享池区域，如用于解析所需的库高速缓存。在这种情况下，进程只在库缓存获取闩锁，而不是在整个共享池。

case, processes latch only the library cache, not the entire shared pool. Unlike **enqueue latches** such as row locks, latches do not permit sessions to queue. When a latch becomes available, the first session to request the latch obtains exclusive access to it. **Latch spinning** occurs when a process repeatedly requests a latch in a loop, whereas **latch sleeping** occurs when a process releases the CPU before renewing the latch request.

Typically, an Oracle process acquires a latch for an extremely short time while manipulating or looking at a data structure. For example, while processing a salary update of a single employee, the database may obtain and release thousands of latches. The implementation of latches is operating system-dependent, especially in respect to whether and how long a process waits for a latch.

An increase in latching means a decrease in concurrency. For example, excessive hard parse operations create contention for the library cache latch. The `V$LATCH` view contains detailed latch usage statistics for each latch, including the number of times each latch was requested and waited for.

See Also:

- "SQL Parsing"
- *Oracle Database Reference* to learn about `V$LATCH`
- *Oracle Database Performance Tuning Guide* to learn about wait event statistics

Mutexes

A **mutual exclusion object (mutex)** is a low-level mechanism that prevents an object in memory from aging out or from being corrupted when accessed by concurrent processes. A mutex is similar to a latch, but whereas a latch typically protects a group of objects, a mutex protects a single object.

Mutexes provide several benefits:

- A mutex can reduce the possibility of contention.

  Because a latch protects multiple objects, it can become a bottleneck when processes attempt to access any of these objects concurrently. By serializing access to an individual object rather than a group, a mutex increases availability.
- A mutex consumes less memory than a latch.

与行锁之类的**入队闩锁**不同，闩锁不允许会话排队。当闩锁可用时，请求闩锁的第一个会话将获得它的独占访问权限。**闩锁旋转**发生在当一个进程不断地循环来请求一个闩锁时，而**闩锁睡眠**发生在重新发起闩锁请求之前，释放 CPU 时。

通常，一个 Oracle 进程在操作或查看一种数据结构时，只需在一个极短的时间内获得闩锁。例如，仅仅为某一名员工处理工资更新，数据库就可能需要获取并释放成千上万个闩锁。闩锁的实现依赖于操作系统，特别是在一个进程是否会在闩锁上等待以及会在闩锁等待多长时间方面。

闩锁的增加意味着并发的降低。例如，过度硬解析操作会产生库缓存闩锁争用。`V$LATCH` 视图包含每个闩锁的详细使用情况的统计信息，包括每个闩锁被请求和被等待的次数。

另见：

- "SQL 解析"
- 《*Oracle 数据库参考*》了解 `V$LATCH`
- 《*Oracle 数据库性能优化指南*》了解等待事件统计信息

互斥体

互斥对象（mutex）是一种底层机制，用于防止在内存中的对象在被多个并发进程访问时，被换出内存或遭到破坏。互斥体类似于闩锁，但闩锁通常保护一组对象，而互斥体通常保护单个对象。

互斥体提供以下几个优点：

- 互斥体可以减少发生争用的可能性。

  由于闩锁保护多个对象，当多个进程试图同时访问这些对象的任何一个时，它可能成为一个瓶颈。而互斥体仅仅串行化对单个对象的访问，而不是一组对象，因此互斥体提高了可用性。
- 互斥体比闩锁消耗更少的内存。

- When in shared mode, a mutex permits concurrent reference by multiple sessions.

Internal Locks

Internal locks are higher-level, more complex mechanisms than latches and mutexes and serve various purposes. The database uses the following types of internal locks:

- Dictionary cache locks

  These locks are of very short duration and are held on entries in dictionary caches while the entries are being modified or used. They guarantee that statements being parsed do not see inconsistent object definitions. Dictionary cache locks can be shared or exclusive. Shared locks are released when the parse is complete, whereas exclusive locks are released when the DDL operation is complete.
- File and log management locks

  These locks protect various files. For example, an internal lock protects the control file so that only one process at a time can change it. Another lock coordinates the use and archiving of the online redo log files. Data files are locked to ensure that multiple instances mount a database in shared mode or that one instance mounts it in exclusive mode. Because file and log locks indicate the status of files, these locks are necessarily held for a long time.
- Tablespace and undo segment locks

  These locks protect tablespaces and undo segments. For example, all instances accessing a database must agree on whether a tablespace is online or offline. Undo segments are locked so that only one database instance can write to a segment.

See Also:

"Data Dictionary Cache"

Overview of Manual Data Locks

Oracle Database performs locking automatically to ensure data concurrency, data integrity, and statement-level read consistency. However, you can manually override the Oracle Database default locking mechanisms. Overriding the default locking is useful in situations such as the following:

- 在共享模式下，互斥体允许被多个会话并发引用。

内部锁

内部锁是比闩锁和互斥体更高级、 更复杂的机制，并用于各种目的。数据库使用以下类型的内部锁：

- 字典缓存锁

  这些锁的持续时间很短，当字典缓存中的条目正在被修改或使用时被持有。他们保证正在被解析的语句不会看到不一致的对象定义。字典缓存锁可以是共享的或独占的。共享锁在解析完成后被释放，而独占锁在 DDL 操作完成时释放。

- 文件和日志管理锁

  这些锁保护各种文件。例如，一种内部锁保护控制文件，以便一次只有一个进程可以对其进行更改。而另一种锁用于协调联机重做日志文件的使用和归档。数据文件被锁定，确保数据库被多个实例以共享模式装载，或以独占模式被单个实例装载。因为文件和日志锁表示文件的状态，这些锁必要时会被持有较长一段时间。

- 表空间和撤销段锁

  这些锁保护的表空间和撤销段。例如，访问数据库的所有实例对一个表空间是否处于联机或脱机必须保持一致。撤销段被锁定，以便只能有一个数据库实例可以写入该段。

另见：

"数据字典缓存"

手动数据锁概述

Oracle 数据库自动执行锁定，以确保数据并发性、数据完整性、和语句级读取一致性。但是，您可以手动覆盖 Oracle 数据库的默认锁定机制。覆盖默认锁定在以下情况下很有用：

- Applications require transaction-level read consistency or **repeatable reads**.
  In this case, queries must produce consistent data for the duration of the transaction, not reflecting changes by other transactions. You can achieve transaction-level read consistency by using explicit locking, read-only transactions, serializable transactions, or by overriding default locking.
- Applications require that a transaction have exclusive access to a resource so that the transaction does not have to wait for other transactions to complete.

You can override Oracle Database automatic locking at the session or transaction level. At the session level, a session can set the required transaction isolation level with the `ALTER SESSION` statement. At the transaction level, transactions that include the following SQL statements override Oracle Database default locking:

- The `SET TRANSACTION ISOLATION LEVEL` statement
- The `LOCK TABLE` statement (which locks either a table or, when used with views, the base tables)
- The `SELECT ... FOR UPDATE` statement

Locks acquired by the preceding statements are released after the transaction ends or a rollback to savepoint releases them.
If Oracle Database default locking is overridden at any level, then the database administrator or application developer should ensure that the overriding locking procedures operate correctly. The locking procedures must satisfy the following criteria: data integrity is guaranteed, data concurrency is acceptable, and deadlocks are not possible or are appropriately handled.

See Also:

- *Oracle Database SQL Language Reference* for descriptions of `LOCK TABLE` and `SELECT ... FOR UPDATE`
- *Oracle Database Advanced Application Developer's Guide* to learn how to manually lock tables

Overview of User-Defined Locks

With Oracle Database Lock Management services, you can define your own locks for a specific application. For example, you might create a lock to serialize access to a message log on the file system. Because a

- 应用程序需要事务级读取一致性或**可重复读取**。

  在这种情况下，查询在整个事务持续期间必须产生一致的数据，但不反映其他事务所做的更改。通过使用显式锁定、 只读事务、 可串行化事务、或覆盖默认锁定，可以实现事务级别的读取一致性。

- 应用程序需要事务对资源具有独占访问权限，以便该事务不必等待其他事务完成就可以继续。

您可以在会话级或事务级覆盖数据库的自动锁定。在会话级，会话可以使用 `ALTER SESSION` 语句设置需要的事务隔离级别。在该事务级别，包括以下 SQL 语句的事务，会覆盖数据库的默认锁定：

- `SET TRANSACTION ISOLATION LEVEL` 语句
- `LOCK TABLE` 语句 （锁定一张表，或与视图一起使用时则锁定其基表）
- `SELECT ... FOR UPDATE` 语句

由上述语句获取的锁在事务结束后被释放，或回滚到保存点后释放。

如果在任何级别覆盖了 Oracle 数据库的默认锁定，则数据库管理员或应用程序开发人员应确保重写锁定过程能正常运行。锁定过程必须满足以下标准：可以保证数据完整性、可以接受的数据并发性、死锁不会发生（若发生，能被适当地处理）。

另见：

- 《*Oracle 数据库 SQL 语言参考*》关于 `LOCK TABLE` 和 `SELECT ... FOR UPDATE` 的描述
- 《*Oracle 数据库高级应用开发指南*》了解如何手动锁定表

用户定义的锁的概述

使用 Oracle 数据库锁定管理服务，您可以为特定应用程序定义您自己的锁。例如，您可以创建一个锁，来串行化到文件系统上的一个消息日志的访问。因为保留的用户锁与 Oracle 数据库锁一样，它具有包括死锁检测在内

reserved user lock is the same as an Oracle Database lock, it has all the Oracle Database lock functionality including deadlock detection. User locks never conflict with Oracle Database locks, because they are identified with the prefix `UL`.

The Oracle Database Lock Management services are available through procedures in the `DBMS_LOCK` package. You can include statements in PL/SQL blocks that:

- Request a lock of a specific type
- Give the lock a unique name recognizable in another procedure in the same or in another instance
- Change the lock type
- Release the lock

See Also:

- *Oracle Database Advanced Application Developer's Guide* for more information about Oracle Database Lock Management services
- *Oracle Database PL/SQL Packages and Types Reference* for information about `DBMS_LOCK`

的 Oracle 数据库锁的所有功能。用户锁永远不会与数据库锁发生冲突，因为他们用前缀 UL 标识。

可通过 DBMS_LOCK 包中的过程来使用 Oracle 数据库的锁定管理服务。您可以在 PL/SQL 块中包含如下语句：

- 请求一个特定类型的锁
- 给锁起一个唯一的名称，以与（同一实例或不同实例中）另一个过程中的某个锁相区别
- 更改锁类型
- 释放锁

另见：

- 《*Oracle 数据库高级应用开发指南*》关于 Oracle 数据库锁定管理服务的详细信息
- 《*Oracle 数据库 PL/SQL 软件包和类型参考*》有关 DBMS_LOCK 的信息

Part III

Oracle Transaction Management

This part contains the following chapters:

- Chapter 9, "Data Concurrency and Consistency"
- Chapter 10, "Transactions"

第三部分

Oracle 事务管理

本部分包括以下各章:

- 第 9 章, "数据并发性与一致性"
- 第 10 章, "事务"

10 Transactions

This chapter defines a transaction and describes how the database processes transactions.

This chapter contains the following sections:

- Introduction to Transactions
  - Sample Transaction: Account Debit and Credit
  - Structure of a Transaction
  - Statement-Level Atomicity
  - System Change Numbers (SCNs)
- Overview of Transaction Control
  - Transaction Names
  - Active Transactions
  - Savepoints
  - Rollback of Transactions
  - Committing Transactions
- Overview of Autonomous Transactions
- Overview of Distributed Transactions
  - Two-Phase Commit
  - In-Doubt Transactions

Introduction to Transactions

A **transaction** is a logical, atomic unit of work that contains one or more SQL statements. A transaction groups SQL statements so that they are either all **committed**, which means they are applied to the database, or all **rolled back**, which means they are undone from the database. Oracle Database assigns every transaction a unique identifier called a **transaction ID**.

All Oracle transactions comply with the basic properties of a database transaction, known as **ACID properties**. ACID is an acronym for the following:

- Atomicity

  All tasks of a transaction are performed or none of them are. There

第 10 章 事务

本章定义了事务，并描述数据库是如何处理事务的。

本章包含以下各节：

- 事务介绍
  - 示例事务： 帐户借贷
  - 事务的结构
  - 语句级原子性
  - 系统更改号 （SCNs）
- 事务控制概述
  - 事务名
  - 活动事务
  - 保存点
  - 事务回滚
  - 事务提交
- 自治事务概述
- 分布式事务概述
  - 两阶段提交
  - 可疑事务

事务介绍

事务是包含一个或多个 SQL 语句的逻辑的、 原子的工作单元。事务将 SQL 语句分组，以便它们可以一起被**提交**，即将其应用到数据库，或者一起被**回滚**，即将其从数据库中撤消。Oracle 数据库将为每个事务分配一个称为**事务 ID** 的唯一标识符。

所有 Oracle 事务都符合称为 **ACID 属性**的数据库事务的基本属性。ACID 是以下的缩写：

- 原子性

  事务中的所有任务，要么全部执行，要么都不执行。不存在部分完成

are no partial transactions. For example, if a transaction starts updating 100 rows, but the system fails after 20 updates, then the database rolls back the changes to these 20 rows.

- Consistency

  The transaction takes the database from one consistent state to another consistent state. For example, in a banking transaction that debits a savings account and credits a checking account, a failure must not cause the database to credit only one account, which would lead to inconsistent data.
- Isolation

  The effect of a transaction is not visible to other transactions until the transaction is committed. For example, one user updating the `hr.employees` table does not see the uncommitted changes to `employees` made concurrently by another user. Thus, it appears to users as if transactions are executing serially.
- Durability

  Changes made by committed transactions are permanent. After a transaction completes, the database ensures through its recovery mechanisms that changes from the transaction are not lost.

The use of transactions is one of the most important ways that a database management system differs from a file system.

Sample Transaction: Account Debit and Credit

To illustrate the concept of a transaction, consider a banking database. When a customer transfers money from a savings account to a checking account, the transaction must consist of three separate operations:

- Decrement the savings account
- Increment the checking account
- Record the transaction in the transaction journal

Oracle Database must allow for two situations. If all three SQL statements maintain the accounts in proper balance, then the effects of the transaction can be applied to the database. However, if a problem such as insufficient funds, invalid account number, or a hardware failure prevents one or two of the statements in the transaction from completing, then the database must roll back the entire transaction so that the balance of all

的事务。例如，如果事务启动并欲更新 100 行，但在系统在完成 20 行更新后出现故障，则数据库会回滚这 20 行更新。

- 一致性

  事务会将数据库从一个一致状态变为另一个一致状态。例如， 在从一个储蓄账户借记、并从一个支票账户贷记的银行事务中，故障一定不能导致数据库仅仅贷记一个账户，这会导致数据不一致。

- 隔离性

  一个事务必须在被提交之后，其它事务才能看见其效果。例如，正在更新 `hr.employees` 表的一个用户，不会看到由另一个用户在 `employees` 表上未提交的更改。因此，对这些用户来说，这些事务好像是串行执行的。

- 持久性

  已提交的事务所做的更改是永久性的。事务完成后，数据库通过其恢复机制，确保在事务中所做的更改不会丢失。

使用事务是数据库管理系统有别于文件系统的最重要方式之一。

示例事务： 帐户借贷

为了说明事务的概念，考虑一个银行数据库。当一个客户将钱从储蓄账户转移到支票帐户时，事务必须由三个单独的操作组成：

- 减少储蓄账户余额
- 增加支票账户余额
- 在事务日志中记录这笔交易

Oracle 数据库必须考虑两种情况。如果所有三个 SQL 语句都对该账号维持了正确的余额，则事务的效果可以应用到数据库中。但是，如果某个问题，如资金不足、帐号无效、或硬件故障，阻止了事务中某些语句的完成，则数据库必须回滚整个事务，以便所有帐户的余额是正确的。

accounts is correct.

Figure 10-1 illustrates a banking transaction. The first statement subtracts $500 from savings account 3209. The second statement adds $500 to checking account 3208. The third statement inserts a record of the transfer into the journal table. The final statement commits the transaction.

Figure 10-1 A Banking Transaction

Description of "Figure 10-1 A Banking Transaction"

Structure of a Transaction

A database transaction consists of one or more statements. Specifically, a transaction consists of one of the following:

- One or more data manipulation language (DML) statements that together constitute an atomic change to the database
- One data definition language (DDL) statement

A transaction has a beginning and an end.

See Also:

"Overview of SQL Statements"

Beginning of a Transaction

A transaction begins when the first executable SQL statement is encountered. An **executable SQL statement** is a SQL statement that

图 10-1 说明了银行事务。第一条语句从储蓄帐户 3209 中减去 500 美元。第二条语句向支票帐户 3208 增加 500 美元。第三条语句在日志表中插入这笔转账记录。最后一个语句提交事务。

图 10-1 银行事务

Description of "Figure 10-1 A Banking Transaction"

事务的结构

数据库事务由一个或多个语句组成。具体而言，事务包含下列之一：

- 一个或多个数据操作语言 (DML) 语句一起构成的、对数据库的一个原子更改
- 一个数据定义语言 (DDL) 语句

事务有一个开始点和结束点。

另见：

"SQL 语句概述"

事务的开始

事务开始于所遇到的第一个可执行 SQL 语句。**可执行的 SQL 语句**是能产生数据库实例调用的 SQL 语句，包括 DML 和 DDL 语句，及 SET

generates calls to a database instance, including DML and DDL statements and the SET TRANSACTION statement.

When a transaction begins, Oracle Database assigns the transaction to an available undo data segment to record the undo entries for the new transaction. A transaction ID is not allocated until an undo segment and transaction table slot are allocated, which occurs during the first DML statement. A transaction ID is unique to a transaction and represents the undo segment number, slot, and sequence number.

The following example execute an UPDATE statement to begin a transaction and queries V$TRANSACTION for details about the transaction:

```
SQL> UPDATE hr.employees SET salary=salary;
107 rows updated.

SQL> SELECT XID AS "txn id", XIDUSN AS "undo seg", XIDSLOT AS
"slot",
  2  XIDSQN AS "seq", STATUS AS "txn status"
  3  FROM V$TRANSACTION;

txn id             undo seg       slot        seq txn status
---------------- ---------- ---------- ---------- ----------------
0600060037000000          6          6         55 ACTIVE
```

See Also:

"Undo Segments"

End of a Transaction

A transaction ends when any of the following actions occurs:

- A user issues a COMMIT or ROLLBACK statement *without* a SAVEPOINT clause.

  In a **commit**, a user explicitly or implicitly requested that the changes in the transaction be made permanent. Changes made by the transaction are permanent and visible to other users only after a transaction commits. The transaction shown in Figure 10-1 ends with a commit.
- A user runs a DDL command such as CREATE, DROP, RENAME, or ALTER.

  The database issues an implicit COMMIT statement before and

TRANSACTION 语句。

当事务开始时， Oracle 数据库将为事务分配一个可用的撤销数据段，以记录新事务的撤消条目。数据库在第一个 DML 语句过程中，首先会分配撤销段和事务表槽，然后分配事务 ID。事务 ID 对于事务来说是唯一的，并由撤销段号、 槽号、和序列号来表示。

下面的示例执行一个 UPDATE 语句以开始一个事务，并从 V$TRANSACTION 查询有关事务的详细信息：

```
SQL> UPDATE hr.employees SET salary=salary;
107 rows updated.

SQL> SELECT XID AS "txn id", XIDUSN AS "undo seg", XIDSLOT AS
"slot",
  2  XIDSQN AS "seq", STATUS AS "txn status"
  3  FROM V$TRANSACTION;

txn id             undo seg       slot        seq txn status
---------------- ---------- ---------- ---------- ----------------
0600060037000000          6          6         55 ACTIVE
```

另见：

"撤销段"

事务的结束

事务在出现以下操作时结束：

- 用户在没有 SAVEPOINT 子句的语句中发出 COMMIT 或 ROLLBACK。

  提交意味着用户会显式或隐式地请求事务中所做的更改被持久化。只有在提交事务之后，事务所做的更改才是永久性的，并对其他用户可见。图 10-1 所示的事务以一个提交来结束。

- 用户运行一个 DDL 命令，如 CREATE、DROP、RENAME、或 ALTER 等。

  数据库在每个 DDL 语句之前和之后发出一个隐式的 COMMIT 语句。

after every DDL statement. If the current transaction contains DML statements, then Oracle Database first commits the transaction and then runs and commits the DDL statement as a new, single-statement transaction.

- A user exits normally from most Oracle Database utilities and tools, causing the current transaction to be implicitly committed. The commit behavior when a user disconnects is application-dependent and configurable.

  Note:
  Applications should always explicitly commit or undo transactions before program termination.

- A client process terminates abnormally, causing the transaction to be implicitly rolled back using metadata stored in the transaction table and the undo segment.

After one transaction ends, the next executable SQL statement automatically starts the following transaction. The following example executes an `UPDATE` to start a transaction, ends the transaction with a `ROLLBACK` statement, and then executes an `UPDATE` to start a new transaction (note that the transaction IDs are different):

如果当前事务中包含 DML 语句，则 Oracle 数据库首先提交该事务，然后将该 DDL 语句作为一个新的单语句事务来运行并提交。

- 用户从大多数 Oracle 数据库实用程序和工具正常退出，导致当前事务被隐式提交。当用户断开连接时的提交行为依赖于应用程序，并且是可配置的。

  注意：
  在程序终止之前，应用程序应始终显式地提交或回滚其事务。

- 客户端进程异常终止，导致数据库使用存储在事务表和撤销段中的元数据来隐式回滚其事务

一个事务结束后, 下一个可执行的 SQL 语句将自动启动后续事务。下面的示例执行一个更新，来启动一个事务，并以一个 ROLLBACK 语句来结束事务，然后又执行一个更新启动一个新事务 （请注意，事务 Id 变了）：

```
SQL> UPDATE hr.employees SET salary=salary;
107 rows updated.

SQL> SELECT XID, STATUS FROM V$TRANSACTION;

XID              STATUS
---------------- ----------------
0800090033000000 ACTIVE

SQL> ROLLBACK;

Rollback complete.

SQL> SELECT XID FROM V$TRANSACTION;

no rows selected

SQL> UPDATE hr.employees SET last_name=last_name;

107 rows updated.

SQL> SELECT XID, STATUS FROM V$TRANSACTION;

XID              STATUS
---------------- ----------------
0900050033000000 ACTIVE
```

See Also:

- "Tools for Database Administrators" and "Tools for Database Developers"
- *Oracle Database SQL Language Reference* to learn about COMMIT

Statement-Level Atomicity

Oracle Database supports **statement-level atomicity**, which means that a SQL statement is an atomic unit of work and either completely succeeds or completely fails.

A successful statement is different from a committed transaction. A single SQL statement executes successfully if the database parses and runs it

```
SQL> UPDATE hr.employees SET salary=salary;
107 rows updated.

SQL> SELECT XID, STATUS FROM V$TRANSACTION;

XID              STATUS
---------------- ----------------
0800090033000000 ACTIVE

SQL> ROLLBACK;

Rollback complete.

SQL> SELECT XID FROM V$TRANSACTION;

no rows selected

SQL> UPDATE hr.employees SET last_name=last_name;

107 rows updated.

SQL> SELECT XID, STATUS FROM V$TRANSACTION;

XID              STATUS
---------------- ----------------
0900050033000000 ACTIVE
```

另见：

- "数据库管理员工具"和"数据库开发人员工具"
- 《*Oracle 数据库 SQL 语言参考*》了解 COMMIT

语句级原子性

Oracle 数据库支持**语句级原子性**，这意味着 SQL 语句是一个原子工作单元，要么完全成功，要么完全失败。

成功的语句不同于已提交事务。如果一个 SQL 语句能被数据库正确分析，并作为一个原子单位来运行且未产生错误，且所有行都被正确更改，则这个

without error as an atomic unit, as when all rows are changed in a multirow update.

If a SQL statement causes an error during execution, then it is not successful and so all effects of the statement are rolled back. This operation is a statement-level rollback. This operation has the following characteristics:

- A SQL statement that does not succeed causes the loss only of work it would have performed itself.

  The unsuccessful statement does not cause the loss of any work that preceded it in the current transaction. For example, if the execution of the second `UPDATE` statement in Figure 10-1 causes an error and is rolled back, then the work performed by the first `UPDATE` statement is not rolled back. The first `UPDATE` statement can be committed or rolled back explicitly by the user.
- The effect of the rollback is as if the statement had never been run.

  Any side effects of an atomic statement, for example, triggers invoked upon execution of the statement, are considered part of the atomic statement. Either all work generated as part of the atomic statement succeeds or none does.

An example of an error causing a statement-level rollback is an attempt to insert a duplicate primary key. Single SQL statements involved in a deadlock, which is competition for the same data, can also cause a statement-level rollback. However, errors discovered during SQL statement parsing, such as a syntax error, have not yet been run and so do not cause a statement-level rollback.

See Also:

- "SQL Parsing"
- "Locks and Deadlocks"
- "Overview of Triggers"

System Change Numbers (SCNs)

A system change number (SCN) is a logical, internal time stamp used by Oracle Database. SCNs order events that occur within the database, which is necessary to satisfy the ACID properties of a transaction. Oracle Database uses SCNs to mark the SCN before which all changes are known to be on disk so that recovery avoids applying unnecessary redo. The database also uses SCNs to mark the point at which no redo exists for a

SQL 语句是成功执行的。

如果 SQL 语句在执行过程中导致错误，则它是不成功的，该语句中的所有影响都将被回滚。此操作是一个语句级回滚。此操作具有以下特征：

- 未成功的 SQL 语句只会导致它本身执行的工作丢失。

  未成功的语句不会导致丢失当前事务中该语句之前的任何工作。例如，若图 10-1 中第二个 `UPDATE` 语句的执行导致错误并被回滚，但由第一个 `UPDATE` 语句执行的工作是不会被回滚的。第一个 `UPDATE` 语句可以由用户显式提交或回滚。
- 回滚的效果是，好像该语句从来未运行过。

  原子语句的副作用被视为原子语句的一部分，例如，由语句执行所引发的触发器调用。作为原子语句的一部分所产生的所有工作，要么都成功，要么都失败。

由错误导致语句级回滚的一个示例，是尝试插入重复的主键。死锁中的某个为争用相同数据的 SQL 语句，也会导致语句级回滚。然而，在解析中发现的错误，比如语法错误，实际上尚未运行，因此不会导致语句级回滚。

另见：

- "SQL 解析"
- "锁和死锁"
- "触发器概述"

系统更改号 （SCNs）

系统更改号 (SCN) 是一个由 Oracle 数据库使用的逻辑、 内部的时间戳。SCN 按数据库中发生的事件排序，以满足在事务 ACID 属性的需要。Oracle 数据库使用 SCN 来标记某个位置，在其之前的所有更改都被认为已写到磁盘上，以避免在恢复过程中应用不必要的重做。数据库还使用 SCN 来标记一个其数据不存在重做信息的点，以便恢复过程可以在该点停止。

set of data so that recovery can stop.

SCNs occur in a monotonically increasing sequence. Oracle Database can use an SCN like a clock because an observed SCN indicates a logical point in time and repeated observations return equal or greater values. If one event has a lower SCN than another event, then it occurred at an earlier time with respect to the database. Several events may share the same SCN, which means that they occurred at the same time with respect to the database.

Every transaction has an SCN. For example, if a transaction updates a row, then the database records the SCN at which this update occurred. Other modifications in this transaction have the same SCN. When a transaction commits, the database records an SCN for this commit.

Oracle Database increments SCNs in the system global area (SGA). When a transaction modifies data, the database writes a new SCN to the undo data segment assigned to the transaction. The log writer process then writes the commit record of the transaction immediately to the online redo log. The commit record has the unique SCN of the transaction. Oracle Database also uses SCNs as part of its instance recovery and media recovery mechanisms.

See Also:

"Overview of Instance Recovery" and "Backup and Recovery"

Overview of Transaction Control

Transaction control is the management of changes made by DML statements and the grouping of DML statements into transactions. In general, application designers are concerned with transaction control so that work is accomplished in logical units and data is kept consistent. Transaction control involves using the following statements, as described in "Transaction Control Statements":

- The `COMMIT` statement ends the current transaction and makes all changes performed in the transaction permanent. `COMMIT` also erases all savepoints in the transaction and releases transaction locks.
- The `ROLLBACK` statement reverses the work done in the current transaction; it causes all data changes since the last `COMMIT` or `ROLLBACK` to be discarded. The `ROLLBACK TO SAVEPOINT` statement undoes the changes since the last savepoint but does not end the entire transaction.

SCN 是按单调递增的顺序发生的。Oracle 数据库可以像使用时钟一样使用 SCN，因为一个 SCN 观察值指示一个逻辑时间点，而其重复观察值相比之前会相同或更大。若一个事件的 SCN 比另一个事件的 SCN 低，则它在数据库中发生在一个更早的时间。几个事件可以共享相同的 SCN， 这意味着他们在数据库中同时发生。

每个事务都有一个 SCN。例如，如果事务更新了一行，则数据库记录此更新发生时的 SCN。此事务中其他修改具有相同的 SCN。当事务提交时，数据库将为此提交记录一个 SCN。

Oracle 数据库在系统全局区域（SGA）中递增 SCN。事务中修改数据时，数据库会将一个新的 SCN 写入到分配给事务的撤销数据段。然后日志写入器进程立即将事务的提交记录写入在线重做日志。提交记录具有事务的唯一 SCN。Oracle 数据库还使用 SCN 作为其实例恢复和介质恢复机制的一部分。

另见：

"实例恢复概述"和"备份和恢复"

事务控制概述

事务控制即管理 DML 语句所做的更改，和将 DML 语句分组为事务。一般情况下，应用程序设计人员都关注事务控制，以便工作能按逻辑单元来完成，且数据能保持一致。

事务控制涉及使用下面的语句，如"事务控制语句"中所述：

- COMMIT 语句结束当前事务，并使在事务中执行的所有更改都具有持久性。提交还会清除在事务中的所有保存点，并释放事务锁。
- ROLLBACK 语句将取消当前事务中所做的工作；它导致所有自上次提交或回滚以来的数据更改被丢弃。`ROLLBACK TO SAVEPOINT` 语句将撤消自上次保存点以来所做的更改，但不会结束整个事务。

- The SAVEPOINT statement identifies a point in a transaction to which you can later roll back.

The session in Table 10-1 illustrates the basic concepts of transaction control.

- SAVEPOINT 语句标识在事务中您可以稍后回滚到的点。

会话表 10-1 说明了事务控制的基本概念。

Table 10-1 Transaction Control

表 10-1 事务控制

Time	Session	Explanation	解释
t0	`COMMIT;`	This statement ends any existing transaction in the session.	此语句结束会话中的任何现有事务。
t1	`SET TRANSACTION NAME 'sal_update';`	This statement begins a transaction and names it `sal_update`.	此语句开始一个事务，并命名为 sal_update。

t2	`UPDATE employees`
	`SET salary = 7000`
	`WHERE last_name = 'Banda';`
		This statement updates the salary for Banda to 7000.	此语句将 Banda 的薪金更新为 7000。

t3	`SAVEPOINT after_banda_sal;`	This statement creates a savepoint named `after_banda_sal`, enabling changes in this transaction to be rolled back to this point.	此语句创建一个名为 after_banda_sal 的保存点，使该事务中的更改可以回滚到该点。

t4	`UPDATE employees`
	`SET salary = 12000`
	`WHERE last_name = 'Greene';`
		This statement updates the salary for Greene to 12000.	此语句将 Greene 的薪金更新为 12000。

t5	`SAVEPOINT after_greene_sal;`	This statement creates a savepoint named `after_greene_sal`, enabling changes in this transaction to be rolled back to this point.	此语句创建一个名为 after_greene_sal 的保存点，使该事务中的更改可以回滚到该点。
t6	`ROLLBACK TO SAVEPOINT` `after_banda_sal;`	This statement rolls back the transaction to t3, undoing the update to Greene's salary at t4. The `sal_update` transaction has *not* ended.	此语句将事务回滚至 t3，，撤消在 t4 时对 Greene 薪金的更新。Sal_update 事务并未结束。

t7	`UPDATE employees`
	`SET salary = 11000`
	`WHERE last_name = 'Greene';`
		This statement updates the salary for Greene to 11000 in transaction `sal_update`.	此语句在事务 sal_update 中将 Greene 的薪金更新为 11000 。

t8	`ROLLBACK;`	This statement rolls back all changes in transaction `sal_update`, ending the transaction.	此语句回滚事务 sal_update 中的所有更改，以结束事务。

t9	`SET TRANSACTION NAME 'sal_update2';`	This statement begins a new transaction in the session and names it `sal_update2`.	此语句开始新的会话中的事务，并命名为 sal_update2。
t10	`UPDATE employees SET salary = 7050 WHERE last_name = 'Banda';`	This statement updates the salary for Banda to 7050.	此语句将 Banda 的薪金更新为 7050。
t11	`UPDATE employees SET salary = 10950 WHERE last_name = 'Greene';`	This statement updates the salary for Greene to 10950.	此语句将 Greene 的薪金更新为 10950。
t12	`COMMIT;`	This statement commits all changes made in transaction `sal_update2`, ending the transaction. The commit guarantees that the changes are saved in the online redo log files.	此语句提交事务 sal_update2 中的所有更改，以结束事务。提交保证所做的更改被保存在联机重做日志文件中。

See Also:

Oracle Database SQL Language Reference to learn about transaction control statements

Transaction Names

A **transaction name** is an optional, user-specified tag that serves as a reminder of the work that the transaction is performing. You name a transaction with the `SET TRANSACTION ... NAME` statement, which if used must be first statement of the transaction. In Table 10-1, the first transaction was named `sal_update` and the second was named `sal_update2`.

Transaction names provide the following advantages:

- It is easier to monitor long-running transactions and to resolve in-doubt distributed transactions.
- You can view transaction names along with transaction IDs in applications. For example, a database administrator can view transaction names in Oracle Enterprise Manager (Enterprise Manager) when monitoring system activity.
- The database writes transaction names to the transaction auditing redo record, so you can use LogMiner to search for a specific transaction in the redo log.
- You can use transaction names to find a specific transaction in data dictionary views such as `V$TRANSACTION`.

另见：

《*Oracle 数据库 SQL 语言参考*》了解事务控制语句

事务名称

事务名称是一个由用户指定的可选标记，作为工作中正在执行的事务中的一个提示。你使用 `SET TRANSACTION ... NAME` 语句来为事务命名，如果要用它，则它必须是事务的第一个语句。在表 10-1 中的第一个事务被命名为 sal_update，而第二个被命名为 sal_update2。

事务名称提供了以下优点：

- 有助于监测长时间运行的事务，并解决可疑的分布式事务。
- 您可以在应用程序中查看事务名称及其相应事务 ID。例如，数据库管理员可以在监视系统活动时，在 Oracle 企业管理器 （企业管理器）中查看事务名称。
- 数据库将事务名称写入事务的审计重做记录，因此您可以使用 LogMiner 来搜索在重做日志中的特定事务。
- 可以使用事务名称在 `V$TRANSACTION` 之类的数据字典视图中查找特定的事务。

See Also:

- "Oracle Enterprise Manager"
- *Oracle Database Reference* to learn about `V$TRANSACTION`
- *Oracle Database SQL Language Reference* to learn about `SET TRANSACTION`

Active Transactions

An **active transaction** has started but not yet committed or rolled back. In Table 10-1, the first statement to modify data in the `sal_update` transaction is the update to Banda's salary. From the successful execution of this update until the `ROLLBACK` statement ends the transaction, the `sal_update` transaction is active.

Data changes made by a transaction are temporary until the transaction is committed or rolled back. Before the transaction ends, the state of the data is as follows:

- Oracle Database has generated undo data information in the system global area (SGA).
  The undo data contains the old data values changed by the SQL statements of the transaction. See "Read Consistency in the Read Committed Isolation Level".
- Oracle Database has generated redo in the online redo log buffer of the SGA.
  The redo log record contains the change to the data block and the change to the undo block. See "Redo Log Buffer".
- Changes have been made to the database buffers of the SGA.

  The data changes for a committed transaction, stored in the database buffers of the SGA, are not necessarily written immediately to the data files by the database writer (DBW*n*). The disk write can happen before or after the commit. See "Database Buffer Cache".
- The rows affected by the data change are locked.

  Other users cannot change the data in the affected rows, nor can they see the uncommitted changes. See "Summary of Locking Behavior".

Savepoints

A **savepoint** is a user-declared intermediate marker within the context of

另见：

- "Oracle 企业管理器"
- 《*Oracle 数据库参考*》了解 `V$TRANSACTION`
- 《*Oracle 数据库 SQL 语言参考*》了解 `SET TRANSACTION`

活动事务

活动事务即是已开始但尚未提交或回滚的事务。在表 10-1 中， sal_update 事务中第一个修改数据的语句是更新 Banda 的薪金。从该更新成功执行、到 ROLLBACK 语句结束该事务， sal_update 事务处于活动状态。

在事务提交或回滚之前，事务对数据所做的更改是暂时的。在事务结束之前，数据的状态如下所示：

- Oracle 数据库已在系统全局区（SGA）中生成撤消数据信息。

  撤消数据包含由事务中的 SQL 语句所更改的数据旧值。请参阅"在已提交读隔离级别中的读取一致性"。

- Oracle 数据库已在 SGA 的联机重做日志缓冲区中生成了重做信息。

  重做日志记录包含数据块和撤消块的更改。请参阅"重做日志缓冲区"。
- 已经对 SGA 中的数据库缓冲区进行了更改。

  已提交事务所做的数据更改，存储在 SGA 的数据库缓冲区中，但不一定会立即由数据库写入器写入数据文件。磁盘写入可能会在提交之前或之后发生。请参阅"数据库缓冲区高速缓存"。

- 数据更改所影响的行已被锁定。

  其他用户不能更改受影响的行中的数据，也不能查看未提交的更改。请参阅"锁定行为总结"。

保存点

保存点是事务上下文中的一个由用户声明的中间标记。在内部，此标记将被

a transaction. Internally, this marker resolves to an SCN. Savepoints divide a long transaction into smaller parts.

If you use savepoints in a long transaction, then you have the option later of rolling back work performed before the current point in the transaction but after a declared savepoint within the transaction. Thus, if you make an error, you do not need to resubmit every statement. Table 10-1 creates savepoint `after_banda_sal` so that the update to the Greene salary can be rolled back to this savepoint.

Rollback to Savepoint

A **rollback to a savepoint** in an uncommitted transaction means undoing any changes made after the specified savepoint, but it does not mean a rollback of the transaction itself. When a transaction is rolled back to a savepoint, as when the `ROLLBACK TO SAVEPOINT after_banda_sal` is run in Table 10-1, the following occurs:

- 1 Oracle Database rolls back only the statements run after the savepoint.
  In Table 10-1, the `ROLLBACK TO SAVEPOINT` causes the `UPDATE` for Greene to be rolled back, but not the `UPDATE` for Banda.
- 2 Oracle Database preserves the savepoint specified in the `ROLLBACK TO SAVEPOINT` statement, but all subsequent savepoints are lost.
  In Table 10-1, the `ROLLBACK TO SAVEPOINT` causes the `after_greene_sal` savepoint to be lost.
- 3 Oracle Database releases all table and row locks acquired after the specified savepoint but retains all data locks acquired previous to the savepoint.
  The transaction remains active and can be continued.

See Also:

- *Oracle Database SQL Language Reference* to learn about the `ROLLBACK` and `SAVEPOINT` statements
- *Oracle Database PL/SQL Language Reference* to learn about transaction processing and control

Enqueued Transactions

Depending on the scenario, transactions waiting for previously locked resources may still be blocked after a rollback to savepoint. When a transaction is blocked by another transaction it enqueues on the blocking

解析为一个 SCN。保存点将长事务划分为多个更小的部分。

如果你在一个长事务中使用保存点，则您就可以在之后将事务中执行的工作回滚到当前时间之前，却又在事务中声明的某个保存点之后。因此，如果您出了错，您不需要重新提交所有每个语句。表 10-1 创建保存点 after_banda_sal，以便对 Greene 薪金的更新可以回滚到此保存点。

回滚到保存点

在未提交事务中**回滚到某个保存点**，意味着该指定的保存点之后所做的任何更改都将被撤消，但它并不意味着事务本身的回滚。当一个事务回滚到一个保存点时，比如像表 10-1 中运行的 `ROLLBACK TO SAVEPOINT after_banda_sal`，将执行以下操作：

- 1 Oracle 数据库将只回滚在该保存点之后运行的语句。

  在表 10-1 中， `ROLLBACK TO SAVEPOINT` 将导致对 Greene 的更新被回滚，但对 Banda 的更新不会被回滚。

- 2 Oracle 数据库保留在 `ROLLBACK TO SAVEPOINT` 语句中指定的保存点，但随后的所有保存点都将丢失。

  在表 10-1 中， `ROLLBACK TO SAVEPOINT` 子句将导致 after_greene_sal 保存点丢失。
- 3 Oracle 数据库释放在指定保存点之后获取的所有表锁和行锁，但保留在该保存点之前获取的所有数据锁。

  事务仍处于活动状态，并可以继续。

另见：

- 《*Oracle 数据库 SQL 语言参考*》了解 `ROLLBACK` 和 `SAVEPOINT` 语句
- 《*Oracle 数据库 PL/SQL 语言参考*》了解事务处理和控制

排队事务

取决于当时的场景，之前在等待被锁定资源的事务，在回滚到保存点后，可能仍处于阻塞状态。一个事务被另一个事务阻塞时，它会在阻塞事务后面排队，因此整个阻塞事务必须提交或回滚，被阻塞的事务才能得以继续。

transaction itself, so that the entire blocking transaction must commit or roll back for the blocked transaction to continue.

In the scenario shown in Table 10-2, session 1 rolls back to a savepoint created before it executed a DML statement. However, session 2 is still blocked because it is waiting for the session 1 transaction to complete.

在表 10-2 显示的场景中，会话 1 回滚到在它执行一个 DML 语句之前所创建的保存点。然而会话 2 仍被阻塞，因为它正在等待事务 1 完全完成。

Table 10-2 Rollback to Savepoint Example

表 10-2 回滚到保存点示例

Time	Session 1	Session 2	Session 3	Explanation	解释
t0	`UPDATE employees SET salary = 7000 WHERE last_name = 'Banda';`			Session 1 begins a transaction. The session places an exclusive lock on the `Banda` row (TX) and a subexclusive table lock (SX) on the table.	会话 1 开始一个事务。会话将在 `Banda` 行 (TX) 上置一个独占锁，而在表上置一个子独占表锁（SX）。
t1	`SAVEPOINT after_banda_sal;`			Session 1 creates a savepoint named `after_banda_sal`.	会话 1 创建一个名为 after_banda_sal 的保存点。
t2	`UPDATE employees SET salary = 12000 WHERE last_name = 'Greene';`			Session 1 locks the `Greene` row.	会话 1 锁定 `Greene` 行。
t3		`UPDATE employees SET salary = 14000 WHERE last_name = 'Greene';`		Session 2 attempts to update the `Greene` row, but fails to acquire a lock because session 1 has a lock on this row. No transaction has begun in session 2.	会话 2 尝试更新 `Greene` 行，但未能锁定该行，因为会话 1 已在该行上具有一个锁。会话 2 中还没有开始任何事务。
t4	`ROLLBACK TO SAVEPOINT after_banda_sal;`			Session 1 rolls back the update to the salary for `Greene`, which releases the row lock for `Greene`. The table lock acquired at t0 is not released. At this point, session 2 is *still* blocked by session 1 because session 2 enqueues on the session 1 *transaction*, which has not yet completed.	会话 1 回滚对 `Greene` 薪金的更新，并释放了对 `Greene` 行的锁定。在 t0 时刻获取的表锁仍未释放。 此时会话 2 仍被会话 1 阻塞， 因为会话 2 排在会话 1 中的事务之后，而该事务尚未完成。
t5			`UPDATE employees`	The `Greene` row is currently	`Greene` 行当前并未锁定，因此会

			SET salary = 11000 WHERE last_name = 'Greene';	unlocked, so session 3 acquires a lock for an update to the Greene row. This statement begins a transaction in session 3.	话 3 为更新 Greene 行获取了一个该行上的锁。此语句在会话 3 中启动一个事务。
t6	COMMIT;			Session 1 commits, ending its transaction. Session 2 is now enqueued for its update to the Greene row behind the transaction in session 3.	会话 1 提交，结束其事务。在 Greene 行的更新队列中，会话 2 现在排在会话 3 中的事务之后。

See Also:
"Lock Duration" to learn more about when Oracle Database releases locks

Rollback of Transactions

A **rollback** of an uncommitted transaction undoes any changes to data that have been performed by SQL statements within the transaction. After a transaction has been rolled back, the effects of the work done in the transaction no longer exist.

In rolling back an entire transaction, without referencing any savepoints, Oracle Database performs the following actions:

- Undoes all changes made by all the SQL statements in the transaction by using the corresponding undo segments
  The transaction table entry for every active transaction contains a pointer to all the undo data (in reverse order of application) for the transaction. The database reads the data from the undo segment, reverses the operation, and then marks the undo entry as applied. Thus, if a transaction inserts a row, then a rollback deletes it. If a transaction updates a row, then a rollback reverses the update. If a transaction deletes a row, then a rollback reinserts it. In Table 10-1, the ROLLBACK reverses the updates to the salaries of Greene and Banda.
- Releases all the locks of data held by the transaction
- Erases all savepoints in the transaction

  In Table 10-1, the ROLLBACK deletes the savepoint after_banda_sal. The after_greene_sal savepoint was removed by the ROLLBACK TO SAVEPOINT statement.

另见：
"锁定持续时间"了解 Oracle 数据库何时释放锁的更多信息

事务回滚

未提交事务的**回滚**，将撤消已由事务中的 SQL 语句执行过的任何数据更改。事务回滚后，事务中所做工作的影响就不再存在。

对未引用任何保存点的整个事务的回滚， Oracle 数据库执行下列操作：

- 通过使用相应的撤销段，撤消事务中所有 SQL 语句所做的所有更改

  每个活动事务的事务表条目包含一个指向该事务的所有撤消数据（与应用顺序相反）的指针。数据库从撤销段中读取数据，反转其操作，然后将撤消条目标记为已应用。因此，如果一个事务插入行，则其回滚删除行。如果一个事务更新行，则其回滚反转这个更新。如果一个事务删除一个行，则其回滚重新插入该行。在表 10-1 中，ROLLBACK 反转对 Greene 和 Banda 的薪金更新。
- 释放由事务持有的所有数据锁
- 清除在事务中的所有保存点

  在表 10-1 中，ROLLBACK 删除保存点 after_banda_sal。After_greene_sal 保存点被 ROLLBACK TO SAVEPOINT 语句删除。

- Ends the transaction

  In Table 10-1, the ROLLBACK leaves the database in the same state as it was after the initial COMMIT was executed.

The duration of a rollback is a function of the amount of data modified.

See Also:

"Undo Segments"

Committing Transactions

A **commit** ends the current transaction and makes permanent all changes performed in the transaction. In Table 10-1, a second transaction begins with sal_update2 and ends with an explicit COMMIT statement. The changes that resulted from the two UPDATE statements are now made permanent.

When a transaction commits, the following actions occur:

- A system change number (SCN) is generated for the COMMIT.

  The internal transaction table for the associated undo tablespace records that the transaction has committed. The corresponding unique SCN of the transaction is assigned and recorded in the transaction table. See "Serializable Isolation Level".
- The log writer (LGWR) process writes remaining redo log entries in the redo log buffers to the online redo log and writes the transaction SCN to the online redo log. *This atomic event constitutes the commit of the transaction.*
- Oracle Database releases locks held on rows and tables.

  Users who were enqueued waiting on locks held by the uncommitted transaction are allowed to proceed with their work.
- Oracle Database deletes savepoints.

  In Table 10-1, no savepoints existed in the sal_update transaction so no savepoints were erased.
- Oracle Database performs a **commit cleanout**.

  If modified blocks containing data from the committed transaction are still in the SGA, and if no other session is modifying them, then the database removes lock-related transaction information from the

- 结束事务

  在表 10-1 中， ROLLBACK 使数据库处于与刚刚执行了 COMMIT 后相同的状态。

回滚的持续时间与被修改的数据量成正比。

另见：

"撤销段"

事务提交

提交结束当前事务，并使事务中执行的所有更改持久化。在表 10-1 中，第二个事务从 sal_update2 开始，并以显式 COMMIT 语句结束。在两个 UPDATE 语句中所做的更改现在被持久化了。

当一个事务提交时，将发生以下操作：

- 为提交生成系统更改号 (SCN)。

  在关联的回滚表空间的内部事务表中记录已提交的事务。分配相应的唯一事务 SCN，并将其记录在事务表中。请参阅"可串行化隔离级别"。
- 日志写入器 （LGWR）进程将重做日志缓冲区中的剩余重做日志条目写入到在线重做日志中，并将事务 SCN 写入到在线重做日志中。*这个原子事件是提交事务的本质所在。*
- Oracle 数据库释放在行和表上持有的锁。

  还在排队等待由未提交事务持有的锁的用户，现在可以继续他们的工作了。
- Oracle 数据库删除保存点。

  在表 10-1 中，在 sal_update 事务中不存在保存点，所以也没有保存点要清除。
- Oracle 数据库执行一个**提交清理**。

  如果包含已提交的事务的被修改数据块仍处于 SGA 中，也没有其他会话正在对他们进行修改，则数据库从该块中删除与锁相关的事务信息。理想情况下，COMMIT 会清理该数据块，以便后续的 SELECT

blocks. Ideally, the COMMIT cleans out the blocks so that a subsequent SELECT does not have to perform this task.

Note:

Because a block cleanout generates redo, a query may generate redo and thus cause blocks to be written during the next checkpoint.

- Oracle Database marks the transaction complete.

After a transaction commits, users can view the changes.

Typically, a commit is a fast operation, regardless of the transaction size. The speed of a commit does not increase with the size of the data modified in the transaction. The lengthiest part of the commit is the physical disk I/O performed by LGWR. However, the amount of time spent by LGWR is reduced because it has been incrementally writing the contents of the redo log buffer in the background.

The default behavior is for LGWR to write redo to the online redo log synchronously and for transactions to wait for the buffered redo to be on disk before returning a commit to the user. However, for lower transaction commit latency, application developers can specify that redo be written asynchronously so that transactions need not wait for the redo to be on disk and can return from the COMMIT call immediately.

See Also:

- *Oracle Database PL/SQL Language Reference* for more information on asynchronous commit
- "Locking Mechanisms"
- "Overview of Background Processes" for more information about LGWR

Overview of Autonomous Transactions

An **autonomous transaction** is an independent transaction that can be called from another transaction, called the **main transaction**. You can suspend the calling transaction, perform SQL operations and commit or undo them in the autonomous transaction, and then resume the calling transaction.

Autonomous transactions are useful for actions that must be performed independently, regardless of whether the calling transaction commits or rolls back. For example, in a stock purchase transaction, you want to commit customer data regardless of whether the overall stock purchase

操作不必再执行此任务。

注意：

由于数据块清理会生成重做，所以查询可能生成重做，并因此导致在下一个检查点期间该数据块会被写入。

- Oracle 数据库将事务标记为完成

事务提交后，用户可以查看所做的更改。

通常，提交是一个快速的操作，而与事务大小无关。提交的速度不会随着在事务中修改的数据的大小的增大而增加。提交的最长部分是由 LGWR 执行的物理磁盘 I/O。然而，LGWR 所用的时间量却减少了，因为它一直在在后台以增量方式写出重做日志缓冲区中的内容。

默认行为是， LGWR 必须将重做数据同步地写入联机重做日志，而事务必须等到被缓冲的重做数据被写到磁盘上之后，提交操作才能返回到用户。但是，为取得更低的事务提交延迟，应用程序开发人员可以指定重做被异步写入，以便事务不需要等待重做被写入磁盘，就可以立即从 COMMIT 调用返回。

另见：

- 《*Oracle 数据库 PL/SQL 语言参考*》有关异步提交的详细信息
- "锁定机制"
- "后台进程概述"有关 LGWR 的详细信息

自治事务概述

自治事务是一个独立的事务，可以从另一个称为**主事务**的事务中调用。你可以挂起调用事务、 在自治事务中执行 SQL 操作并提交或回滚它们，然后继续执行调用事务。

自治事务对于那些必须独立执行、而不管调用事务是否提交或回滚的操作非常有用。例如，在股票购买事务中，你想要总是提交客户数据，而无论是否股票购买最终成功通过。或者，您想要将错误消息记录到一个调试表，即

goes through. Additionally, you want to log error messages to a debug table even if the overall transaction rolls back.

Autonomous transactions have the following characteristics:

- The autonomous transaction does not see uncommitted changes made by the main transaction and does not share locks or resources with the main transaction.
- Changes in an autonomous transaction are visible to other transactions upon commit of the autonomous transactions. Thus, users can access the updated information without having to wait for the main transaction to commit.
- Autonomous transactions can start other autonomous transactions. There are no limits, other than resource limits, on how many levels of autonomous transactions can be called.

In PL/SQL, an autonomous transaction executes within an **autonomous scope**, which is a routine marked with the pragma `AUTONOMOUS_TRANSACTION`. In this context, routines include top-level anonymous PL/SQL blocks and PL/SQL subprograms and triggers. A **pragma** is a directive that instructs the compiler to perform a compilation option. The pragma `AUTONOMOUS_TRANSACTION` instructs the database that this procedure, when executed, is to be executed as a new autonomous transaction that is independent of its parent transaction.

Figure 10-2 shows how control flows from the main routine (MT) to an autonomous routine and back again. The main routine is `proc1` and the autonomous routine is `proc2`. The autonomous routine can commit multiple transactions (AT1 and AT2) before control returns to the main routine.

Figure 10-2 Transaction Control Flow

使整个事务被回滚。

自治事务具有以下特征：

- 自治事务不会看见在主事务中所做的未提交更改，也不与主事务共享锁或资源。
- 自治事务中的更改在其提交后对其它事务是可见的。因此，用户可以立刻访问更新的信息，而不必等到主事务提交。
- 自治事务可以启动其他自治事务。没有关于自治事务能调用多少级别的任何限制，这只受限于可用的资源。

在 PL/SQL 中， 自治事务在一个标有 AUTONOMOUS_TRANSACTION 的编译指令例程所在的**自治范围**内执行。在这种上下文中，**例程**包括顶层匿名 PL/SQL 块、 PL/SQL 子程序、和触发器。一个编译指令是指示编译器执行某个编译选项的指令。AUTONOMOUS_TRANSACTION 编译指令指示数据库，当此过程执行时，作为一个独立于其父事务的新的自治事务来执行。

图 10-2 显示了控制如何从主程序 （MT）流向一个自治例程，又如何返回。主程序是 proc1 ， 自治例程是 proc2。自治例程在控制返回主例程之前可以提交多个事务 （AT1 和 AT2）。

图 10-2 事务控制流

Description of "Figure 10-2 Transaction Control Flow"

When you enter the executable section of an autonomous routine, the main routine suspends. When you exit the autonomous routine, the main routine resumes.

In Figure 10-2, the `COMMIT` inside `proc1` makes permanent not only its own work but any outstanding work performed in its session. However, a `COMMIT` in `proc2` makes permanent only the work performed in the `proc2` transaction. Thus, the `COMMIT` statements in transactions AT1 and AT2 have no effect on the MT transaction.

See Also:

Oracle Database Advanced Application Developer's Guide and *Oracle Database PL/SQL Language Reference* to learn how to use autonomous transactions

Overview of Distributed Transactions

A distributed database is a set of databases in a distributed system that can appear to applications as a single data source. A **distributed transaction** is a transaction that includes one or more statements that update data on two or more distinct nodes of a distributed database, using a schema object called a database link. A database link describes how one database instance can log in to another database instance. Unlike a transaction on a local database, a distributed transaction alters data on multiple databases. Consequently, distributed transaction processing is more complicated because the database must coordinate the committing or rolling back of the changes in a transaction as an atomic unit. The entire transaction must commit or roll back. Oracle Database must coordinate transaction control over a network and maintain data consistency, even if a network or system failure occurs.

See Also:

Oracle Database Administrator's Guide

Two-Phase Commit

The **two-phase commit** mechanism guarantees that *all* databases participating in a distributed transaction either all commit or all undo the statements in the transaction. The mechanism also protects implicit DML performed by integrity constraints, remote procedure calls, and triggers. In a two-phase commit among multiple databases, one database coordinates the distributed transaction. The initiating node is called the **global coordinator**. The coordinator asks the other databases if they are

Description of "Figure 10-2 Transaction Control Flow"

当您进入一个自治例程的可执行部分时，主例程将挂起。当您退出自治例程时，主例程就恢复了。

在图 10-2 中，proc1 内的提交将持久化自己的工作，及在其会话中执行的任何未结工作。然而，在 proc2 中的提交仅持久化在 proc2 事务中执行的工作。在事务 AT1 和 AT2 中的 `COMMIT` 语句对 MT 事务没有影响。

另见：

《*Oracle 数据库高级应用程序开发者指南*》和《*Oracle 数据库 PL/SQL 语言参考*》学习如何使用自治事务

分布式事务概述

分布式数据库是在分布式系统中，可以对应用程序而言，作为单一数据源出现的一组数据库。**分布式事务**，包括一个或多个使用称为数据库链接的模式对象的语句，更新位于分布式数据库中两个或更多不同节点上的数据。数据库链接描述一个数据库实例如何登录到另一个数据库实例的方式。

与本地数据库上的事务不同，分布式事务变更多个数据库上的数据。因此，分布式事务处理更复杂，因为数据库必须进行协调，以将事务中所做的更改作为一个原子单元提交或回滚。必须提交或回滚整个事务。Oracle 数据库必须通过网络协调事务控制，并保持数据的一致性，即使发生网络或系统故障。

另见：

《*Oracle 数据库管理员指南*》

两阶段提交

两阶段提交机制保证参与分布式事务的所有数据库，要么都提交、或者要么都回滚事务中的语句。该机制还可以保护由完整性约束、 远程过程调用、和触发器执行的隐式 DML。

在涉及多个数据库的两阶段提交中，由某个数据库协调该分布式事务。发起的节点称为**全局协调者**。协调者询问其他数据库是否已准备好提交。如果任何数据库响应说未准备好，则整个事务回滚。如果所有数据库都说已准备

prepared to commit. If any database responds with a no, then the entire transaction is rolled back. If all databases vote yes, then the coordinator broadcasts a message to make the commit permanent on each of the databases.

好，则协调者将广播一个消息，使该提交在每个数据库上持久化。

The two-phase commit mechanism is transparent to users who issue distributed transactions. In fact, users need not even know the transaction is distributed. A COMMIT statement denoting the end of a transaction automatically triggers the two-phase commit mechanism. No coding or complex statement syntax is required to include distributed transactions within the body of a database application.

两阶段提交机制对发出分布式事务的用户是透明的。事实上，用户甚至不需要知道事务是分布式的。表示事务结束的 COMMIT 语句自动触发两阶段提交机制。在数据库应用程序正文中不需要额外编码或复杂的语句语法来包括分布式事务。

See Also:

Oracle Database Administrator's Guide to learn about the two-phase commit mechanism

另见：

《*Oracle 数据库管理员指南*》了解两阶段提交机制

In-Doubt Transactions

可疑事务

An **in-doubt distributed transaction** occurs when a two-phase commit was interrupted by any type of system or network failure. For example, two databases report to the coordinating database that they were prepared to commit, but the coordinating database instance fails immediately after receiving the messages. The two databases who are prepared to commit are now left hanging while they await notification out of the outcome.

可疑分布式事务，发生在两阶段提交被任何类型的系统或网络故障中断时。例如，两个数据库向协调数据库报告他们已准备提交，但协调数据库实例收到消息后却立即失效了。这两个准备提交的数据库现在被挂起，而他们还在等待通知结果。

The recoverer (RECO) background process automatically resolves the outcome of in-doubt distributed transactions. After the failure is repaired and communication is reestablished, the RECO process of each local Oracle database automatically commits or rolls back any in-doubt distributed transactions consistently on all involved nodes.

恢复器 (RECO) 后台进程会自动解决可疑分布式事务的残局。该故障修复和通信重新建立后，在所有涉及到的节点上，每个本地的 Oracle 数据库的 RECO 进程一致地自动提交或回退任何可疑的分布式事务。

In the event of a long-term failure, Oracle Database enables each local administrator to manually commit or undo any distributed transactions that are in doubt because of the failure. This option enables the local database administrator to free any locked resources that are held indefinitely because of the long-term failure.

对于长时间失败的情况， Oracle 数据库使每个本地管理员手动提交或回滚任何由于失败引起的可疑分布式事务。此选项使本地数据库管理员能够释放任何由于长期的失败所引起的无限期持有资源锁定。

If a database must be recovered to a past time, then database recovery facilities enable database administrators at other sites to return their databases to the earlier point in time. This operation ensures that the global database remains consistent.

如果数据库必须恢复到过去的某个时间，则数据库恢复工具使得在其它站点的数据库管理员能将其数据库返回到某个过去的时间点。此操作可确保全局数据库保持一致。

See Also:

另见：

- "Recoverer Process (RECO)"
- *Oracle Database Administrator's Guide* to learn how to manage in-doubt transactions

- "恢复进程(RECO)"
- 《*Oracle 数据库管理员指南*》了解如何管理可疑事务

11 Physical Storage Structures

This chapter describes the primary physical database structures of an Oracle database. Physical structures are viewable at the operating system level.

This chapter contains the following sections:

- Introduction to Physical Storage Structures
  - Mechanisms for Storing Database Files
  - Oracle Automatic Storage Management (Oracle ASM)
  - Oracle Managed Files and User-Managed Files
- Overview of Data Files
  - Use of Data Files
  - Permanent and Temporary Data Files
  - Online and Offline Data Files
  - Data File Structure
- Overview of Control Files
  - Use of Control Files
  - Multiple Control Files
  - Control File Structure
- Overview of the Online Redo Log
  - Use of the Online Redo Log
  - How Oracle Database Writes to the Online Redo Log
  - Structure of the Online Redo Log

Introduction to Physical Storage Structures

One characteristic of an RDBMS is the independence of logical data structures such as tables, views, and indexes from physical storage structures. Because physical and logical structures are separate, you can manage physical storage of data without affecting access to logical structures. For example, renaming a database file does not rename the tables stored in it.

An **Oracle database** is a set of files that store Oracle data in persistent disk storage. This section discusses the database files generated when you issue a `CREATE DATABASE` statement:

第 11 章 物理存储结构

本章介绍了 Oracle 数据库的主要物理数据库结构。物理结构可以在操作系统级别查看。

本章包含以下各节：

- 物理存储结构简介
  - 用于存储数据库文件的机制
  - Oracle 自动存储管理 （Oracle ASM）
  - Oracle 管理的文件和用户管理的文件
- 数据文件的概述
  - 使用数据文件
  - 永久和临时数据文件
  - 联机和脱机数据文件
  - 数据文件结构
- 控制文件的概述
  - 使用控制文件
  - 多个控制文件
  - 控制文件结构
- 联机重做日志概述
  - 使用联机重做日志
  - Oracle 数据库如何写入到联机重做日志
  - 联机重做日志的结构

物理存储结构简介

RDBMS 的特点之一是逻辑数据结构（如表、 视图和索引）与物理存储结构的独立性。由于物理结构和逻辑结构是分开的，您可以管理数据的物理数据存储，而不会影响对逻辑结构的访问。例如，重命名数据库文件并不会重命名存储在其中的表。

Oracle 数据库是在持久性的磁盘存储中存储的一组 Oracle 数据文件。此部分讨论您发出 `CREATE DATABASE` 语句时生成的数据库文件：

- Data files and temp files

  A data file is a physical file on disk that was created by Oracle Database and contains data structures such as tables and indexes. A temp file is a data file that belongs to a temporary tablespace. The data is written to these files in an Oracle proprietary format that cannot be read by other programs.
- Control files

  A control file is a root file that tracks the physical components of the database.
- Online redo log files

  The online redo log is a set of files containing records of changes made to data.

A database instance is a set of memory structures that manage database files. Figure 11-1 shows the relationship between the instance and the files that it manages.

Figure 11-1 Database Instance and Database Files

Description of "Figure 11-1 Database Instance and Database Files"

See Also:

- *Oracle Database Administrator's Guide* to learn how to create a database
- *Oracle Database SQL Language Reference* for `CREATE DATABASE` semantics and syntax

Mechanisms for Storing Database Files

- 数据文件和临时文件

  数据文件是磁盘上的一个物理文件，由 Oracle 数据库创建，并包含如表和索引之类的数据结构。临时文件是一个属于临时表空间的数据文件。数据按 Oracle 专有格式写入这些文件，且不能由其他程序读取。
- 控制文件

  控制文件是一个跟踪数据库的物理组件的根文件。
- 联机重做日志文件

  在线重做日志是一组包含对数据所做的更改记录的文件。

数据库实例是一组管理数据库文件的内存结构。图 11-1 显示实例和它所管理的文件之间的关系。

图 11-1 数据库实例和数据库文件

Description of "Figure 11-1 Database Instance and Database Files"

另见：

- 《*Oracle 数据库管理员指南*》了解如何创建数据库
- 《*Oracle 数据库 SQL 语言参考*》有关 `CREATE DATABASE` 的语义和语法

用于存储数据库文件的机制

Several mechanisms are available for allocating and managing the storage of these files. The most common mechanisms include:

- Oracle Automatic Storage Management (Oracle ASM)

  Oracle ASM includes a file system designed exclusively for use by Oracle Database. "Oracle Automatic Storage Management (Oracle ASM)" describes Oracle ASM.
- Operating system file system

  Most Oracle databases store files in a **file system**, which is a data structure built inside a contiguous disk address space. All operating systems have **file managers** that allocate and deallocate disk space into files within a file system.
  A file system enables disk space to be allocated to many files. Each file has a name and is made to appear as a contiguous address space to applications such as Oracle Database. The database can create, read, write, resize, and delete files.
  A file system is commonly built on top of a **logical volume** constructed by a software package called a **logical volume manager (LVM)**. The LVM enables pieces of multiple physical disks to be combined into a single contiguous address space that appears as one disk to higher layers of software.
- Raw device

  Raw devices are disk partitions or logical volumes not formatted with a file system. The primary benefit of raw devices is the ability to perform **direct I/O** and to write larger buffers. In direct I/O, applications write to and read from the storage device directly, bypassing the operating system buffer cache.
  Note:
  Many file systems now support direct I/O for databases and other applications that manage their own caches. Historically, raw devices were the only means of implementing direct I/O.
- Cluster file system

  A **cluster file system** is software that enables multiple computers to share file storage while maintaining consistent space allocation and file content. In an Oracle RAC environment, a cluster file

有几种机制可用于分配和管理这些文件的存储。最常见的机制包括：

- Oracle 自动存储管理（Oracle ASM）

  Oracle ASM 包括被专门设计用来供 Oracle 数据库使用的文件系统。"Oracle 自动存储管理 (Oracle ASM)"描述了 Oracle ASM。

- 操作系统文件系统

  大多数 Oracle 数据库在**文件系统**中存储文件，这是建立在一个连续的磁盘地址空间内的数据结构。所有操作系统都有**文件管理器**，用于在文件系统中为文件分配和释放磁盘空间。

  文件系统可以将磁盘空间分配给多个文件。每个文件都有一个名称，并对应用程序（如 Oracle 数据库）显示为一个连续地址空间。数据库可以创建、 读取、 写入、 调整大小、和删除文件。

  文件系统通常是由称为**逻辑卷管理器 (LVM)**的软件包创建，并建立在逻辑卷的基础上。LVM 使多个物理磁盘被合并到一个单一的连续地址空间，对上层软件显示为一块磁盘。

- 原始设备

  原始设备是一些磁盘分区，或未使用文件系统格式化的逻辑卷。原始设备的主要好处是能够执行直接 I/O 操作，并按更大的缓冲区写入。对于直接 I/O， 应用程序直接向存储设备写入或从存储设备读取，绕过操作系统缓冲区高速缓存。

  注意：
  现在许多文件系统都支持数据库和其他应用程序的直接 I/O，让它们自己管理自己的缓存。历史上，原始设备曾被作为实施直接 I/O 的唯一手段。
- 集群文件系统

  集群文件系统是使多台计算机共享文件存储，同时保持一致的空间分配和文件内容的软件。在 Oracle RAC 环境中，集群文件系统使共享存储显示为在集群环境中由多台计算机共享的文件系统。在集群文件

system makes shared storage appears as a file system shared by many computers in a clustered environment. With a cluster file system, the failure of a computer in the cluster does not make the file system unavailable. In an operating system file system, however, if a computer sharing files through NFS or other means fails, then the file system is unavailable.

A database employs a combination of the preceding storage mechanisms. For example, a database could store the control files and online redo log files in a traditional file system, some user data files on raw partitions, the remaining data files in Oracle ASM, and archived the redo log files to a cluster file system.

See Also:

- *Oracle Database 2 Day DBA* to learn how to view database storage structures with Oracle Enterprise Manager (Enterprise Manager)
- *Oracle Database Administrator's Guide* to view database storage structures by querying database views

Oracle Automatic Storage Management (Oracle ASM)

Oracle ASM is a high-performance, ease-of-management storage solution for Oracle Database files. Oracle ASM is a volume manager and provides a file system designed exclusively for use by the database.

Oracle ASM provides several advantages over conventional file systems and storage managers, including the following:

- Simplifies storage-related tasks such as creating and laying out databases and managing disk space
- Distributes data across physical disks to eliminate hot spots and to provide uniform performance across the disks
- Rebalances data automatically after storage configuration changes

To use Oracle ASM, you allocate partitioned disks for Oracle Database with preferences for striping and mirroring. Oracle ASM manages the disk space, distributing the I/O load across all available resources to optimize performance while removing the need for manual I/O tuning. For example, you can increase the size of the disk for the database or move parts of the database to new devices without having to shut down the database.

系统中，单个集群中的计算机失败不会使文件系统不可用。而在操作系统文件系统中，倘若某个通过 NFS 或其他手段共享文件的计算机失败，则文件系统会变得不可用。

数据库可以采用前面的某些存储机制的组合。例如，数据库可以将控制文件和联机重做日志文件存储在传统的文件系统中，将某些用户数据文件存储在原始分区上，剩余的数据文件存储在 Oracle ASM 中，而归档重做日志文件存储到集群文件系统。

另见：

- 《*Oracle 数据库 2 日 DBA*》了解如何使用 Oracle 企业管理器 （企业管理器）查看数据库存储结构
- 《*Oracle 数据库管理员指南*》通过查询数据库视图来查看数据库存储结构

Oracle 自动存储管理 （Oracle ASM）

Oracle ASM 是一个针对 Oracle 数据库文件的高性能、 易管理的存储解决方案。Oracle ASM 是一个卷管理器，并提供一个专为数据库使用而设计的文件系统。

Oracle ASM 提供几个超越传统文件系统和存储管理器的优势，包括以下这些：

- 简化了与存储相关的任务，如创建数据库、规划数据库文件布局、和管理磁盘空间
- 跨物理磁盘分布数据，以消除热点，并提供均匀的磁盘性能
- 在存储配置更改后自动平衡数据

若要使用 Oracle ASM，你需要为 Oracle 数据库分配磁盘分区，最好是经过条带化和镜像的。Oracle ASM 会管理磁盘空间，跨所有可用资源分配 I/O 负载，以优化性能，同时消除了手动优化 I/O 的需要。例如，你可以为数据库增加磁盘大小，或将数据库的某些部分移动到新设备，而不必关闭数据库。

Oracle ASM Storage Components

Oracle Database can store a data file as an **Oracle ASM file** in an **Oracle ASM disk group**, which is a collection of disks that Oracle ASM manages as a unit. Within a disk group, Oracle ASM exposes a file system interface for database files.

Figure 11-2 shows the relationships between storage components in a database that uses Oracle ASM. The diagram depicts the relationship between an Oracle ASM file and a data file, although Oracle ASM can store other types of files. The crow's foot notation represents a one-to-many relationship.

Figure 11-2 Oracle ASM Components

Description of "Figure 11-2 Oracle ASM Components"

Figure 11-2 illustrates the following Oracle ASM concepts:

- Oracle ASM Disks

  An **Oracle ASM disk** is a storage device that is provisioned to an Oracle ASM disk group. An Oracle ASM disk can be a physical disk or partition, a Logical Unit Number (LUN) from a storage array, a logical volume, or a network-attached file.
  Oracle ASM disks can be added or dropped from a disk group while the database is running. When you add a disk to a disk group, you either assign a disk name or the disk is given an Oracle ASM disk name automatically.
- Oracle ASM Disk Groups

  An **Oracle ASM disk group** is a collection of Oracle ASM disks

Oracle ASM 存储组件

Oracle 数据库可以将数据文件存储为一个 **Oracle ASM 磁盘组**中的 **Oracle ASM 文件**，Oracle ASM 磁盘组是一个磁盘的集合，被 Oracle ASM 作为一个单元来管理。在一个磁盘组内， Oracle ASM 为数据库文件提供了一个文件系统接口。

图 11-2 显示了一个使用 Oracle ASM 的数据库中的存储组件之间的关系。虽然 Oracle ASM 可以存储其他类型的文件，该图只描述了一个 Oracle ASM 文件和数据文件之间的关系。乌鸦脚表示法表示一对多关系。

图 11-2 Oracle ASM 组件

Description of "Figure 11-2 Oracle ASM Components"

图 11-2 说明了以下 Oracle ASM 概念：

- Oracle ASM 磁盘

  Oracle ASM 磁盘是提供给一个 Oracle ASM 磁盘组的存储设备。Oracle ASM 磁盘可以是某个存储阵列的物理磁盘、分区、逻辑单元号 (LUN)， 或是一个逻辑卷或网络附加文件。

  Oracle ASM 磁盘，可以当数据库正在运行时，被添加到磁盘组，或从磁盘组中删除。当你向磁盘组中添加一个磁盘时，您可以为磁盘指定一个名称，否则会自动被赋予一个 Oracle ASM 磁盘名称。

- Oracle ASM 磁盘组

  Oracle ASM 磁盘组是一组 Oracle ASM 磁盘的集合，被作为一个逻

managed as a logical unit. The data structures in a disk group are self-contained and consume some disk space in a disk group.

Within a disk group, Oracle ASM exposes a file system interface for Oracle database files. The content of files that are stored in a disk group are evenly distributed, or striped, to eliminate hot spots and to provide uniform performance across the disks. The performance is comparable to the performance of raw devices.

- Oracle ASM Files

  An **Oracle ASM file** is a file stored in an Oracle ASM disk group. Oracle Database communicates with Oracle ASM in terms of files. The database can store data files, control files, online redo log files, and other types of files as Oracle ASM files. When requested by the database, Oracle ASM creates an Oracle ASM file and assigns it a fully qualified name beginning with a plus sign (+) followed by a disk group name, as in `+DISK1`.

  Note:

  Oracle ASM files can coexist with other storage management options such as raw disks and third-party file systems. This capability simplifies the integration of Oracle ASM into pre-existing environments.

- Oracle ASM Extents

  An **Oracle ASM extent** is the raw storage used to hold the contents of an Oracle ASM file. An Oracle ASM file consists of one or more file extents. Each Oracle ASM extent consists of one or more allocation units on a specific disk.

  Note:

  An Oracle ASM extent is different from the extent used to store data in a segment.

- Oracle ASM Allocation Units

  An **allocation unit** is the fundamental unit of allocation within a disk group. An allocation unit is the smallest contiguous disk space that Oracle ASM allocates. One or more allocation units form an Oracle ASM extent.

See Also:

- *Oracle Database 2 Day DBA* to learn how to administer Oracle ASM

辑单元来管理。在磁盘组中的数据结构是自包含的，并会在磁盘组中消耗一些磁盘空间。

在一个磁盘组内， Oracle ASM 为数据库文件提供了一个文件系统接口。存储在磁盘组中的文件的内容被均匀分布或条带化，以消除热点，并为所有磁盘提供均匀的性能。其性能与原始设备的性能相差无几。

- Oracle ASM 文件

  Oracle ASM 文件是一个存储在 Oracle ASM 磁盘组中的文件。Oracle 数据库按文件与 Oracle ASM 进行通信。数据库可以将数据文件、 控制文件、 联机重做日志文件、和其他类型的文件存储为 Oracle ASM 文件。当数据库要创建一个文件时， Oracle ASM 创建一个 Oracle ASM 文件，并分配一个以加号 (+)和磁盘组名称开头（如 + DISK1）的完全限定名称。

  注意：

  Oracle ASM 文件可以与原始磁盘和第三方文件系统等其他存储管理选项共存。此功能简化了将 Oracle ASM 集成到现有环境的过程。

- Oracle ASM 扩展区

  Oracle ASM 扩展区是用于容纳 Oracle ASM 文件内容的原始存储。一个 Oracle ASM 文件包含一个或多个文件扩展区。每个 Oracle ASM 扩展区是由特定的磁盘上的一个或多个分配单元组成的。

  注意：

  Oracle ASM 扩展区与用于在段中存储数据的扩展区不同。

- Oracle ASM 分配单元

  分配单元是磁盘组中空间分配的基本单位。分配单元是由 Oracle ASM 分配的最小连续磁盘空间。由一个或多个分配单元形成一个 Oracle ASM 扩展区。

另见：

- 《*Oracle 数据库 2 日 DBA*》学习如何使用 Oracle 企业管理器 （企业

disks with Oracle Enterprise Manager (Enterprise Manager)

- *Oracle Automatic Storage Management Administrator's Guide* to learn more about Oracle ASM

Oracle ASM Instances

An **Oracle ASM instance** is a special Oracle instance that manages Oracle ASM disks. Both the ASM and the database instances require shared access to the disks in an ASM disk group. ASM instances manage the metadata of the disk group and provide file layout information to the database instances. Database instances direct I/O to ASM disks without going through an ASM instance.

An ASM instance is built on the same technology as a database instance. For example, an ASM instance has a system global area (SGA) and background processes that are similar to those of a database instance. However, an ASM instance cannot mount a database and performs fewer tasks than a database instance.

Figure 11-3 shows a single-node configuration with one Oracle ASM instance and two database instances, each associated with a different single-instance database. The ASM instance manages the metadata and provides space allocation for the ASM files storing the data for the two databases. One ASM disk group has four ASM disks and the other has two disks. Both database instances can access the disk groups.

Figure 11-3 Oracle ASM Instance and Database Instances

管理器）管理 Oracle ASM 磁盘

- 《*Oracle 自动存储管理管理员指南*》了解 Oracle ASM

Oracle ASM 实例

Oracle ASM 实例是一个专门用于管理 Oracle ASM 磁盘的 Oracle 实例。ASM 实例和数据库实例都需要对 ASM 磁盘组中的磁盘的共享访问。ASM 实例管理磁盘组的元数据，并提供对数据库实例的文件布局信息。数据库实例直接与 ASM 磁盘进行 I/O ，而不需要通过 ASM 实例。

ASM 实例与数据库实例建立在相同的技术架构之上。例如，与数据库实例类似， ASM 实例也有一个系统全局区域 （SGA） 和多个后台进程。但是，ASM 实例无法装载数据库，并且比数据库实例执行的任务更少。

图 11-3 显示了具有一个 Oracle ASM 实例和两个数据库实例的单节点配置，每个数据库实例与一个不同的单实例数据库相关联。ASM 实例管理元数据，并为用于这两个数据库存储数据的 ASM 文件提供空间分配。其中一个 ASM 磁盘组有四个 ASM 磁盘，而另一个磁盘组有两个磁盘。这两个数据库实例可以访问所有磁盘组。

图 11-3 Oracle ASM 实例和数据库实例

Description of "Figure 11-3 Oracle ASM Instance and Database Instances "

See Also:

- *Oracle Database 2 Day DBA* to learn how to administer Oracle ASM disks with Oracle Enterprise Manager (Enterprise Manager)
- *Oracle Automatic Storage Management Administrator's Guide* to learn more about Oracle ASM

Oracle Managed Files and User-Managed Files

Oracle Managed Files is a file naming strategy that enables you to specify operations in terms of database objects rather than file names. For example, you can create a tablespace without specifying the names of its data files. In this way, Oracle Managed Files eliminates the need for administrators to directly manage the operating system files in a database. Oracle ASM requires Oracle Managed Files.

Description of "Figure 11-3 Oracle ASM Instance and Database Instances "

- 《*Oracle 数据库 2 日 DBA*》学习如何使用 Oracle 企业管理器 （企业管理器）管理 Oracle ASM 磁盘
- 《*Oracle 自动存储管理管理员指南*》了解 Oracle ASM

Oracle 管理的文件和用户管理的文件

Oracle 管理的文件是一种命名策略，使您能够按数据库对象而不是按文件名称来指定操作。例如，您可以创建一个表空间，而无需指定其数据文件的名称。这样一来，Oracle 管理的文件不需要管理员直接管理数据库中的操作系统文件。Oracle ASM 需要使用 Oracle 管理的文件。

Note:
This feature does not affect the creation or naming of administrative files such as trace files, audit files, and alert logs (see "Overview of Diagnostic Files").
With **user-managed files**, you directly manage the operating system files in the database. You make the decisions regarding file structure and naming. For example, when you create a tablespace you set the name and path of the tablespace data files.
Through initialization parameters, you specify the file system directory for a specific type of file. The Oracle Managed Files feature ensures that the database creates a unique file and deletes it when no longer needed. The database internally uses standard file system interfaces to create and delete files for data files and temp files, control files, and recovery-related files stored in the fast recovery area.
Oracle Managed Files does not eliminate existing functionality. You can create new files while manually administering old files. Thus, a database can have a mixture of Oracle Managed Files and user-managed files.

See Also:
Oracle Database Administrator's Guide to learn how to use Oracle Managed Files

Overview of Data Files

At the operating system level, Oracle Database stores database data in data files. Every database must have at least one data file.

Use of Data Files

Part I, "Oracle Relational Data Structures" explains the logical structures in which users store data, the most important of which are tables. Each nonpartitioned schema object and each partition of an object is stored in its own segment.
For ease of administration, Oracle Database allocates space for user data in tablespaces, which like segments are logical storage structures. Each segment belongs to only one tablespace. For example, the data for a nonpartitioned table is stored in a single segment, which is turn is stored in one tablespace.
Oracle Database physically stores tablespace data in data files. Tablespaces and data files are closely related, but have important differences:

- Each tablespace consists of one or more data files, which conform to the operating system in which Oracle Database is running.

注意：
此功能不会影响诸如跟踪文件、审计文件、和警报日志等管理性文件的创建或命名（参见"诊断文件概述）。

你可以使用**用户管理的文件**来直接管理数据库中的操作系统文件。由你来决定关于文件的结构和命名。例如，当创建表空间时，由您设置表空间数据文件的名称和路径。

通过初始化参数，您可以指定一个特定类型文件的文件系统目录。Oracle 管理的文件特性可确保数据库能创建具有唯一名称的文件，并在不再需要时删除它。数据库在内部使用标准文件系统接口来创建和删除文件，如数据文件、临时文件、 控制文件、和存储在快速恢复区中的与恢复相关的文件等。

Oracle 管理的文件并不消除现有的功能。您可以创建新的文件，并手动管理旧文件。因此，数据库可以混合使用 Oracle 管理的文件和用户管理的文件。

另见：
《*Oracle 数据库管理员指南*》了解如何使用 Oracle 管理的文件

数据文件的概述

在操作系统层面， Oracle 数据库将数据库数据存储在数据文件中。每个数据库必须至少有一个数据文件。

使用数据文件

在第一部分，Oracle 关系数据结构中讲述了用于存储用户数据的逻辑结构，其中最重要的是表。每个非分区模式对象和每个对象的分区都存储在其自己的段中。

为便于管理， Oracle 数据库在表空间中为用户数据分配空间。与段类似，表空间也是逻辑存储结构。每个段只属于一个表空间。例如，非分区表的数据存储在单个段中，最终也存储在一个表空间中。

Oracle 数据库将表空间中的数据在物理上存储为数据文件。表空间和数据文件密切相关，但又有重要区别：

- 每个表空间包含一个或多个数据文件，这需要遵从运行 Oracle 数据

- The data for a database is collectively stored in the data files located in each tablespace of the database.
- A segment can span one or more data files, but it cannot span multiple tablespaces.
- A database must have the SYSTEM and SYSAUX tablespaces. Oracle Database automatically allocates the first data files of any database for the SYSTEM tablespace during database creation.

The SYSTEM tablespace contains the data dictionary, a set of tables that contains database metadata. Typically, a database also has an undo tablespace and a temporary tablespace (usually named TEMP). Figure 11-4 shows the relationship between tablespaces, data files, and segments.

Figure 11-4 Data Files and Tablespaces

库的操作系统。

- 数据库数据被集中地存储在位于每个数据库表空间的数据文件中。
- 段可以跨越一个或多个数据文件，但它不能跨多个表空间。
- 数据库必须有 SYSTEM 表空间和 SYSAUX 表空间。Oracle 数据库自动在数据库创建过程中为 SYSTEM 表空间分配数据库的第一个数据文件。

SYSTEM 表空间包含数据字典，它是包含数据库元数据的一组表。通常，数据库也有一个撤销表空间和临时表空间 （通常称为 TEMP）。

图 11-4 显示了表空间、 数据文件、和段之间的关系。

图 11-4 数据文件和表空间

Description of "Figure 11-4 Data Files and Tablespaces"

See Also:

- "Overview of Tablespaces"
- *Oracle Database Administrator's Guide* and *Oracle Database 2 Day DBA* to learn how to manage data files

Permanent and Temporary Data Files

A **permanent tablespace** contains persistent schema objects. Objects in permanent tablespaces are stored in data files.

A **temporary tablespace** contains schema objects only for the duration of a session. Locally managed temporary tablespaces have temporary files (temp files), which are special files designed to store data in hash, sort, and other operations. Temp files also store result set data when insufficient space exists in memory.

Temp files are similar to permanent data files, with the following exceptions:

- Permanent database objects such as tables are never stored in temp files.
- Temp files are always set to `NOLOGGING` mode, which means that they never have redo generated for them. Media recovery does not recognize temp files.
- You cannot make a temp file read-only.
- You cannot create a temp file with the `ALTER DATABASE` statement.
- When you create or resize temp files, they are not always guaranteed allocation of disk space for the file size specified. On file systems such as Linux and UNIX, temp files are created as **sparse files**. In this case, disk blocks are allocated not at file creation or resizing, but as the blocks are accessed for the first time.

  Caution:

- Sparse files enable fast temp file creation and resizing; however, the disk could run out of space later when the temp files are accessed.
- Temp file information is shown in the data dictionary view `DBA_TEMP_FILES` and the dynamic performance view

Description of "Figure 11-4 Data Files and Tablespaces"

另见：

- 表空间概述
- 《*Oracle 数据库管理员指南*》和《*Oracle 数据库 2 日 DBA*》了解如何管理数据文件

永久和临时数据文件

永久表空间中包含持久性的模式对象。永久表空间中的对象存储在数据文件中。

临时表空间中仅包含用于会话期间的模式对象。本地管理的临时表空间具有临时文件，它是被设计用于存储在哈希、 排序、和其他操作使用的数据的特殊文件。临时文件还用于存储在内存中的空间不足时的结果集数据。

临时文件类似于永久数据文件，但有以下例外情况：

- 永久性数据库对象（如表）绝不会存储在临时文件中。
- 临时文件始终被设置为 `NOLOGGING` 模式，这意味着永远不会为其生成重做。介质恢复不识别临时文件。
- 你不能将临时文件置为只读模式。
- 你无法使用 ALTER DATABASE 语句来创建一个临时文件。
- 当你创建临时文件或调整其大小时，并不能始终保证会为他们分配指定文件大小的磁盘空间。在 Linux 和 UNIX 等的文件系统上，临时文件被创建为**稀疏文件**。在这种情况下，磁盘块并不是在文件的创建或调整大小时分配的，而是在块第一次被访问时分配的。

  警告：

- 稀疏文件使你能够快速地创建临时文件或调整其大小 ；但是，当之后访问临时文件时，可能会超出磁盘空间。
- 临时文件信息显示在数据字典视图 `DBA_TEMP_FILES` 和动态性能视图 `V$TEMPFILE` 中，但不在 `DBA_DATA_FILES` 或

V$TEMPFILE, but not in DBA_DATA_FILES or the V$DATAFILE view.

See Also:

- "Temporary Tablespaces"
- *Oracle Database Administrator's Guide* to learn how to manage temp files

Online and Offline Data Files

Every data file is either **online** (available) or **offline** (unavailable). You can alter the availability of individual data files or temp files by taking them offline or bringing them online. Offline data files cannot be accessed until they are brought back online.

Administrators may take data files offline for many reasons, including performing offline backups, renaming a data file, or block corruption. The database takes a data file offline automatically if the database cannot write to it.

Like a data file, a tablespace itself is offline or online. When you take a data file offline in an online tablespace, the tablespace itself remains online. You can make all data files of a tablespace temporarily unavailable by taking the tablespace itself offline

See Also:

- "Online and Offline Tablespaces"
- *Oracle Database Administrator's Guide* to learn how to alter data file availability

Data File Structure

Oracle Database creates a data file for a tablespace by allocating the specified amount of disk space plus the overhead for the **data file header**. The operating system under which Oracle Database runs is responsible for clearing old information and authorizations from a file before allocating it to the database.

The data file header contains metadata about the data file such as its size and checkpoint SCN. Each header contains an **absolute file numbe**r and a **relative file number**. The absolute file number uniquely identifies the data file within the database. The relative file number uniquely identifies a data file within a tablespace.

When Oracle Database first creates a data file, the allocated disk space is formatted but contains no user data. However, the database reserves the space to hold the data for future segments of the associated tablespace. As the data grows in a tablespace, Oracle Database uses the free space in

V$DATAFILE 视图中。

另见：

- 临时表空间
- 《*Oracle 数据库管理员指南*》了解如何管理临时文件

联机和脱机数据文件

每个数据文件可能是**联机的**（可用） 或**脱机的**（不可用）。可以通过让单个数据文件或临时文件脱机或联机，来改变其可用性。脱机的数据文件不能被访问，直至他们恢复联机状态。

管理员因为很多原因将数据文件脱机，包括执行脱机备份、重命名数据文件、或数据块破坏。如果数据库无法写入数据文件，则数据库自动将其脱机。

与数据文件类似，表空间本身也可以是脱机的或联机的。当你将一个联机表空间中的数据文件脱机时，表空间本身将仍保持联机状态。您可以将整个表空间脱机，来使该表空间中的所有数据文件暂时不可用

另见：

- "联机和脱机表空间"
- 《*Oracle 数据库管理员指南*》了解如何更改数据文件的可用性

数据文件结构

Oracle 数据库通过分配指定数量的磁盘空间，加上**数据文件头**开销，来为表空间创建数据文件。在 Oracle 数据库之下运行的操作系统，负责在将文件分配给数据库之前，从文件中清除旧信息和授权。

数据文件头包含数据文件的元数据，例如数据文件大小和检查点 SCN。每个文件头包含一个**绝对文件号**和一个**相对文件号**。绝对文件号唯一地标识数据库内的数据文件。相对文件号唯一地标识表空间内的数据文件。

当 Oracle 数据库刚创建一个数据文件时，分配的磁盘空间已格式化，但不包含任何用户数据。但是，数据库会保留此空间，将来用于容纳其相关联的表空间中的数据段。随着表空间中的数据增长，Oracle 数据库将用数据文

the data files to allocate extents for the segment.

Figure 11-5 illustrates the different types of space in a data file. Extents are either used, which means they contain segment data, or free, which means they are available for reuse. Over time, updates and deletions of objects within a tablespace can create pockets of empty space that individually are not large enough to be reused for new data. This type of empty space is referred to as **fragmented free space**.

Figure 11-5 Space in a Data File

Description of "Figure 11-5 Space in a Data File"

See Also:

Oracle Database 2 Day DBA and *Oracle Database Administrator's Guide* to learn how to view data file information

Overview of Control Files

The database control file is a small binary file associated with only one database. Each database has one unique control file, although it may maintain identical copies of it.

Use of Control Files

The control file is the root file that Oracle Database uses to find database files and to manage the state of the database generally. A control file contains information such as the following:

- The database name and database unique identifier (DBID)
- The time stamp of database creation
- Information about data files, online redo log files, and archived redo log files
- Tablespace information
- RMAN backups

件中的可用空间来为段分配扩展区。

图 11-5 说明了数据文件中不同类型的空间。扩展区要么是已使用的，意味着他们包含段数据，或是空闲的，意味着它们是可重用的。随时间的流逝，表空间中的对象不断更新和删除，会产生很多小的空白空间，但太小不足以被重用以容纳新数据。这种类型的空白空间称为**碎片化的可用空间**。

图 11-5 在数据文件中的空间

Description of "Figure 11-5 Space in a Data File"

另见：

《*Oracle 数据库 2 日 DBA*》 和《*Oracle 数据库管理员指南*》了解如何查看数据文件的信息

控制文件的概述

数据库控制文件是一个小的二进制文件，仅与一个数据库相关联。每个数据库有一个唯一的控制文件，尽管它可能维护多个相同的拷贝。

使用控制文件

控制文件是 Oracle 数据库用来查找数据库文件，并从总体上管理数据库状态的根文件。控制文件包含以下信息：

- 数据库名称和数据库唯一标识符（DBID)
- 创建数据库的时间戳
- 有关数据文件、 联机重做日志文件、和归档重做日志文件的信息
- 表空间信息
- RMAN 备份

The control file serves the following purposes:

- It contains information about data files, online redo log files, and so on that are required to open the database.

  The control file tracks structural changes to the database. For example, when an administrator adds, renames, or drops a data file or online redo log file, the database updates the control file to reflect this change.
- It contains metadata that must be accessible when the database is not open.
  For example, the control file contains information required to recover the database, including checkpoints. A checkpoint indicates the SCN in the redo stream where instance recovery would be required to begin (see "Overview of Instance Recovery"). Every committed change before a checkpoint SCN is guaranteed to be saved on disk in the data files. At least every three seconds the checkpoint process records information in the control file about the checkpoint position in the online redo log.

Oracle Database reads and writes to the control file continuously during database use and must be available for writing whenever the database is open. For example, recovering a database involves reading from the control file the names of all the data files contained in the database. Other operations, such as adding a data file, update the information stored in the control file.

See Also:

- "Checkpoint Process (CKPT)"
- *Oracle Database Administrator's Guide* to learn how to manage the control file

Multiple Control Files

Oracle Database enables multiple, identical control files to be open concurrently and written for the same database. By multiplexing a control file on different disks, the database can achieve redundancy and thereby avoid a single point of failure.

Note:

Oracle recommends that you maintain multiple control file copies, each on a different disk.

If a control file becomes unusable, then the database instance fails when it attempts to access the damaged control file. When other current control

控制文件用于以下目的：

- 它包含有关数据文件、 联机重做日志文件等为打开数据库所需的信息。

  控制文件跟踪数据库的结构变化。例如，当管理员添加、 重命名、或删除数据文件或联机重做日志文件时，数据库将更新控制文件以反映此更改。

- 它包含未打开数据库时必须可访问的元数据。

  例如，控制文件中包含，包括检查点在内的用于恢复数据库所需的信息。检查点指示在重做流中进行实例恢复所需的起始 SCN（见"实例恢复概述"）。每一个检查点 SCN 之前提交的更改被确保已保存到磁盘上的数据文件中。至少每隔三秒，检查点进程会在控制文件中，记录有关联机重做日志中的检查点位置。

在数据库使用期间， Oracle 数据库不断读取和写入控制文件，并且只要数据库处于打开状态，控制文件就必须是可用的，以便可以写入。例如，恢复数据库涉及从控制文件中读取数据库中包含的所有数据文件名称。其他的操作，如添加数据文件，会更新存储在控制文件中的信息。

另见：

- 检查点进程(CKPT)
- 《*Oracle 数据库管理员指南*》了解如何管理控制文件

多个控制文件

Oracle 数据库对同一数据库启用多个完全相同的控制文件，并同时打开和写入。通过在不同的磁盘上多路复用控制文件，数据库可以实现冗余，从而避免单点故障。

注意：

Oracle 建议您维护多个控制文件副本，每个都在不同的磁盘上。

如果控制文件变得不可用，则当数据库实例尝试访问已损坏的控制文件时，会遭遇失败。当其他的当前控制文件副本存在时，可以重新装入数据库并打

file copies exist, the database can be remounted and opened without media recovery. If *all* control files of a database are lost, however, then the instance fails and media recovery is required. Media recovery is not straightforward if an older backup of a control file must be used because a current copy is not available.

See Also:

- *Oracle Database Administrator's Guide* to learn how to maintain multiple control files
- *Oracle Database Backup and Recovery User's Guide* to learn how to back up and restore control files

Control File Structure

Information about the database is stored in different **sections** of the control file. Each section is a set of **records** about an aspect of the database. For example, one section in the control file tracks data files and contains a set of records, one for each data file. Each section is stored in multiple logical **control file blocks**. Records can span blocks within a section.

The control file contains the following types of records:

- Circular reuse records

  These records contain noncritical information that is eligible to be overwritten if needed. When all available record slots are full, the database either expands the control file to make room for a new record or overwrites the oldest record. Examples include records about archived redo log files and RMAN backups.
- Noncircular reuse records

  These records contain critical information that does not change often and cannot be overwritten. Examples of information include tablespaces, data files, online redo log files, and redo threads. Oracle Database never reuses these records unless the corresponding object is dropped from the tablespace.

As explained in "Overview of the Dynamic Performance Views", you can query the dynamic performance views, also known as `V$` views, to view the information stored in the control file. For example, you can query `V$DATABASE` to obtain the database name and DBID. However, only the database can modify the information in the control file.

Reading and writing the control file blocks is different from reading and

开，而不需进行介质恢复。如果数据库的所有控制文件都丢失了，那么，实例会失败，且需要进行介质恢复。如果控制文件的当前副本不可用，而必须使用一个旧备份控制文件，则介质恢复就没那么直截了当了。

另见：

- 《*Oracle 数据库管理员指南*》了解如何维护多个控制文件
- 《*Oracle 数据库备份和恢复用户指南*》了解如何备份和还原控制文件

控制文件结构

有关数据库的信息存储在控制文件的各个不同**部分**中。每个部分是有关数据库的某个方面的一组**记录**。例如，控制文件中有一个部分跟踪数据文件，并包含一个记录集合，每一个数据文件有一条记录。每个部分存储在多个逻辑控制文件块中。同一部分的记录可以跨越块。

控制文件中包含以下类型的记录：

- 循环重用记录

  这些记录包含必要时可以被覆盖的非关键信息。当所有可用的记录槽用完时，数据库要么会扩展控制文件，以便为新记录腾出空间，要么会覆盖最旧的记录。其示例包括有关归档重做日志文件和 RMAN 备份的记录。
- 非循环重用记录

  这些记录包含不经常更改且不能被覆盖的关键信息。其示例包括表空间、 数据文件、 联机重做日志文件、和重做线程。Oracle 数据库绝不会重用这些记录，除非从表空间中删除相应的对象。

如"动态性能视图概述"所述，您可以查询动态性能视图（也称为 V$ 视图）来查看存储在控制文件中的信息。例如，您可以查询 `V$DATABASE` 获取数据库名称和 DBID。但是，只有数据库可以修改控制文件中的信息。

控制文件块的读取和写入不同于数据块的读取和写入。对控制文件，Oracle

writing data blocks. For the control file, Oracle Database reads and writes directly from the disk to the program global area (PGA). Each process allocates a certain amount of its PGA memory for control file blocks.

See Also:

- *Oracle Database Reference* to learn about the `V$CONTROLFILE_RECORD_SECTION` view
- *Oracle Database Reference* to learn about the `CONTROL_FILE_RECORD_KEEP_TIME` initialization parameter

数据库直接从磁盘读取并写入到程序全局区（PGA）。每个进程为控制文件块分配一定的 PGA 内存。

另见：

- 《*Oracle 数据库参考*》了解 V$CONTROLFILE_RECORD_SECTION 视图
- 《*Oracle 数据库参考*》了解 CONTROL_FILE_RECORD_KEEP_TIME 初始化参数

Overview of the Online Redo Log

The most crucial structure for recovery is the online redo log, which consists of two or more preallocated files that store changes to the database as they occur. The online redo log records changes to the data files.

Use of the Online Redo Log

The database maintains online redo log files to protect against data loss. Specifically, after an instance failure the online redo log files enable Oracle Database to recover committed data not yet written to the data files.

Oracle Database writes every transaction synchronously to the redo log buffer, which is then written to the online redo logs. The contents of the log include uncommitted transactions, undo data, and schema and object management statements.

Oracle Database uses the online redo log only for recovery. However, administrators can query online redo log files through a SQL interface in the Oracle LogMiner utility (see "Oracle LogMiner"). Redo log files are a useful source of historical information about database activity.

See Also:

"Overview of Instance Recovery"

How Oracle Database Writes to the Online Redo Log

The online redo log for a database instance is called a redo thread. In single-instance configurations, only one instance accesses a database, so only one redo thread is present. In an Oracle Real Application Clusters (Oracle RAC) configuration, however, two or more instances concurrently access a database, with each instance having its own redo thread. A separate redo thread for each instance avoids contention for a single set of online redo log files.

An online redo log consists of two or more online redo log files. Oracle

联机重做日志概述

用于恢复的最关键结构是联机重做日志，它由两个或多个预分配文件组成，用于存储数据库发生的更改。联机重做日志用于记录对数据文件的更改。

使用联机重做日志

数据库维护联机重做日志文件，以防止数据丢失。具体来说，在一个实例失败后，联机重做日志文件使 Oracle 数据库可以恢复已提交但尚未写入数据文件的数据。

Oracle 数据库将每个事务同步地写入重做日志缓冲区，然后将其写入到联机重做日志。日志的内容包括未提交的事务、撤消数据、和模式和对象管理语句。

Oracle 数据库使用联机重做日志的唯一目的是为了恢复。然而，管理员可以通过 Oracle LogMiner 实用程序中的 SQL 接口，来查询联机重做日志文件 （请参阅"Oracle LogMiner"）。重做日志文件是一种有关数据库活动的有用的历史信息来源。

另见：

"实例恢复概述"

Oracle 数据库如何写入到联机重做日志

数据库实例的联机重做日志称为重做线程。在单实例配置中，只有一个实例访问数据库，所以只存在一个重做线程。不过在一个 Oracle 真正应用集群（Oracle RAC） 配置中，有两个或更多实例同时访问数据库，且每个实例都有它自己的重做线程。每个实例具有其单独的重做线程可以避免对单一联机重做日志文件组的争用。

一个联机重做日志包含两个或更多联机重做日志文件。Oracle 数据库要求

Database requires a minimum of two files to guarantee that one is always available for writing while the other is being archived (if the database is in ARCHIVELOG mode).

See Also:

Oracle Database 2 Day + Real Application Clusters Guide and *Oracle Real Application Clusters Administration and Deployment Guide* to learn about online redo log groups in Oracle RAC

Online Redo Log Switches

Oracle Database uses only one online redo log file at a time to store records written from the redo log buffer. The online redo log file to which the log writer (LGWR) process is actively writing is called the **current** online redo log file.

A log switch occurs when the database stops writing to one online redo log file and begins writing to another. Normally, a switch occurs when the current online redo log file is full and writing must continue. However, you can configure log switches to occur at regular intervals, regardless of whether the current online redo log file is filled, and force log switches manually.

Log writer writes to online redo log files circularly. When log writer fills the last available online redo log file, the process writes to the first log file, restarting the cycle. Figure 11-6 illustrates the circular writing of the redo log.

Figure 11-6 Reuse of Online Redo Log Files

最少两个文件，以保证当另一个正在被归档时，总有一个始终可供写入（如果数据库是在归档模式下）。

另见：

《*Oracle 数据库 2 日 + 真正应用集群指南*》和 《*Oracle 真正应用集群管理和部署指南*》 了解在 Oracle RAC 中的联机重做日志组

联机重做日志切换

Oracle 数据库一次只使用一个联机重做日志文件来存储从重做日志缓冲区写入的记录。正在由日志写入器（LGWR） 进程写入的联机重做日志文件称为**当前**联机重做日志文件。

当数据库停止向一个联机重做日志文件写入，并开始向另一个写入时，发生日志切换。通常，切换发生在当前联机重做日志文件已满，且必须继续写入时。但是，您可以配置定期日志切换，而不论当前联机重做日志文件是否已满，或者也可以手动强制日志切换。

日志写入器以循环方式写入联机重做日志文件。当日志写入器填满最后一个可用的联机重做日志文件时，该进程又向第一个日志文件写入，如此等等，周而复始。图 11-6 说明了重做日志的循环写入。

图 11-6 重用联机重做日志文件

Description of "Figure 11-6 Reuse of Online Redo Log Files"

The numbers in Figure 11-6 shows the sequence in which LGWR writes to each online redo log file. The database assigns each file a new log sequence number when a log switches and log writers begins writing to it. When the database reuses an online redo log file, this file receives the next available log sequence number.
Filled online redo log files are available for reuse depending on the archiving mode:

- If archiving is disabled, which means that the database is in `NOARCHIVELOG` mode, then a filled online redo log file is available after the changes recorded in it have been **checkpointed** (written) to disk by database writer (DBW*n*).

- If archiving is enabled, which means that the database is in ARCHIVELOG mode, then a filled online redo log file is available to log writer after the changes have been written to the data files *and*

Description of "Figure 11-6 Reuse of Online Redo Log Files"

图 11-6 中的数字显示了 LGWR 写入到每个联机重做日志文件的顺序。当发生日志切换时，数据库为每个文件分配一个新的日志序列号，然后日志写入器便开始向新的日志写入。当数据库重复使用一个联机重做日志文件时，此文件接收下一个可用的日志序列号。

已填充的联机重做日志文件的可重用性取决于归档模式：

- 如果禁用了归档，这意味着数据库处于 `NOARCHIVELOG` 模式下，则填充的联机重做日志文件，在数据库写入器 (DBWn) 已对该日志中的更改记录执行过检查点操作并写入到磁盘后，该日志即是可重新使用的。

- 如果启用了归档，这意味着数据库是在 ARCHIVELOG 模式下，则填充的联机重做日志文件，只有在该日志中所包含的更改记录被写入数

the file has been archived.

In some circumstances, log writer may be prevented from reusing an existing online redo log file. For example, an online redo log file may be **active** (required for instance recovery) rather than **inactive** (not required for instance recovery). Also, an online redo log file may be in the process of being cleared.

See Also:

- "Overview of Background Processes"
- *Oracle Database 2 Day DBA* and *Oracle Database Administrator's Guide* to learn how to manage the online redo log

Multiple Copies of Online Redo Log Files

Oracle Database can automatically maintain two or more identical copies of the online redo log in separate locations. An **online redo log group** consists of an online redo log file and its redundant copies. Each identical copy is a **member** of the online redo log group. Each group is defined by a number, such as group 1, group 2, and so on.

Maintaining multiple members of an online redo log group protects against the loss of the redo log. Ideally, the locations of the members should be on separate disks so that the failure of one disk does not cause the loss of the entire online redo log.

In Figure 11-7, `A_LOG1` and `B_LOG1` are identical members of group 1, while `A_LOG2` and `B_LOG2` are identical members of group 2. Each member in a group must be the same size. LGWR writes concurrently to group 1 (members `A_LOG1` and `B_LOG1`), then writes concurrently to group 2 (members `A_LOG2` and `B_LOG2`), then writes to group 1, and so on. LGWR never writes concurrently to members of different groups.

Figure 11-7 Multiple Copies of Online Redo Log Files

据文件，且该日志已被归档后，该日志才是可重新使用的。

在某些情况下，日志写入器可能无法重新使用某个现有的联机重做日志文件。例如，联机重做日志文件可能处于活动状态（实例恢复需要它），而不是非活动状态 （实例恢复不需要它）。另外，一个联机重做日志文件可能正处于被清除过程中。

另见：

- "后台进程概述"
- 《*Oracle 数据库 2 日 DBA*》 和 《*Oracle 数据库管理员指南*》了解如何管理联机重做日志

联机重做日志文件的多个副本

Oracle 数据库可以在不同的位置自动维护两个或更多联机重做日志的相同副本。一个**联机重做日志组**包括一个联机重做日志文件及其冗余副本。每个完全相同的副本是联机重做日志组的一个**成员**。每个组由一个数字来定义，如组 1、 组 2 等等。

维护联机重做日志组的多个成员，可以防止重做日志的丢失。理想情况下，各个重做日志成员的位置应该在单独的磁盘上，这样单个磁盘故障不会导致整个联机重做日志的丢失。

在图 11-7 中， A_LOG1 和 B_LOG1 是组 1 的两个相同的成员，A_LOG2 和 B_LOG2 是组 2 的两个相同的成员。每个组中的所有成员必须具有相同的大小。LGWR 同时写入组 1 （成员 A_LOG1 和 B_LOG1），然后同时写入组 2 （成员 A_LOG2 和 B_LOG2），然后又重新写入到组 1，如此等等。LGWR 永远不会同时写入不同组的成员。

图 11-7 联机重做日志文件的多个副本

Description of "Figure 11-7 Multiple Copies of Online Redo Log Files"

Note:
Oracle recommends that you multiplex the online redo log. The loss of log files can be catastrophic if recovery is required. When you multiplex the online redo log, the database must increase the amount of I/O it performs. Depending on your system, this additional I/O may impact overall database performance.

See Also:
Oracle Database Administrator's Guide to learn how to maintain multiple copies of the online redo log files

Archived Redo Log Files

An **archived redo log file** is a copy of a filled member of an online redo log group. This file is not considered part of the database, but is an offline copy of an online redo log file created by the database and written to a user-specified location.

Archived redo log files are a crucial part of a backup and recovery strategy. You can use archived redo log files to:

Description of "Figure 11-7 Multiple Copies of Online Redo Log Files"

注意：
Oracle 建议你多路复用联机重做日志。如果需要进行恢复，则日志文件丢失可能是灾难性的。当你多路复用联机重做日志文件时，数据库所执行的 I/O 肯定会增加。取决于您的系统，这种额外的 I/O 可能会影响数据库的总体性能。

另见：
《*Oracle 数据库管理员指南*》了解如何维护联机重做日志文件的多个副本

归档重做日志文件

归档重做日志文件是联机重做日志组的已填充成员的副本。此文件不是该数据库的一部分，而是由该数据库生成、并被写到用户指定位置的联机重做日志文件脱机副本。

归档重做日志文件是备份和恢复策略的关键部分。您可以使用归档重做日志文件：

- Recover a database backup
- Update a standby database (see "Computer Failures")
- Obtain information about the history of a database using the LogMiner utility (see "Oracle LogMiner")

Archiving is the operation of generating an archived redo log file. Archiving is either automatic or manual and is only possible when the database is in ARCHIVELOG mode.
An archived redo log file includes the redo entries and the log sequence number of the identical member of the online redo log group. In Figure 11-7, files A_LOG1 and B_LOG1 are identical members of Group 1. If the database is in ARCHIVELOG mode, and if automatic archiving is enabled, then the **archiver process (ARCn)** will archive one of these files. If A_LOG1 is corrupted, then the process can archive B_LOG1. The archived redo log contains a copy of every group created since you enabled archiving.

See Also:

- "Data File Recovery"
- *Oracle Database Administrator's Guide* to learn how to manage the archived redo log

Structure of the Online Redo Log

Online redo log files contain **redo records**. A redo record is made up of a group of **change vectors**, each of which describes a change to a data block. For example, an update to a salary in the employees table generates a redo record that describes changes to the data segment block for the table, the undo segment data block, and the transaction table of the undo segments.
The redo records have all relevant metadata for the change, including the following:

- SCN and time stamp of the change
- Transaction ID of the transaction that generated the change
- SCN and time stamp when the transaction committed (if it committed)
- Type of operation that made the change
- Name and type of the modified data segment

See Also:
"Overview of Data Blocks"

- 恢复数据库备份
- 更新备用数据库 （请参见"计算机故障"）
- 使用 LogMiner 实用程序获取有关数据库的历史信息 （请参阅 "Oracle LogMiner"）

归档即是生成一个归档重做日志文件操作。归档可以是自动的或手动的，只有在数据库处于归档模式下时才会归档。

一个归档重做日志文件包括重做条目和联机重做日志组的相同成员的日志序列号。在图 11-7 中，文件 A_LOG1 和 B_LOG1 是组 1 的相同成员。如果数据库处于 ARCHIVELOG 模式下，并启用了自动归档，则归档器进程（ARCn） 将存档这些文件之一。如果 A_LOG1 损坏了，则该进程可以归档 B_LOG1。归档重做日志包含自您启用存档以来创建的每个日志组的副本。

另见：

- "数据文件恢复"
- 《*Oracle 数据库管理员指南*》了解如何管理归档重做日志

联机重做日志的结构

联机重做日志文件包含重做记录。重做记录由一组**更改向量**组成，每个向量描述对某数据块的一个更改。例如，对 employees 表中的薪水的更新，将生成一个重做记录，描述对该表的数据段块、 回滚段数据块、和回滚段的事务表的更改。

重做记录记录了与更改相关的所有元数据，包括如下：

- 更改的 SCN 和时间戳
- 变更事务的事务 ID
- 事务提交时的 SCN 和时间戳（如果它已提交）
- 所做更改的操作类型
- 被修改的数据段的名称和类型

另见：
"数据块概述"

Part IV

Oracle Database Storage Structures

This part describes the basic structural architecture of the Oracle database, including logical and physical storage structures.

This part contains the following chapters:

- Chapter 11, "Physical Storage Structures"
- Chapter 12, "Logical Storage Structures"

第四部分

Oracle 数据库存储结构

本部分描述 Oracle 数据库的基本结构性架构，包括逻辑存储结构和物理存储结构

本部分包括如下各章：

- 第 11 章, "物理存储结构"
- 第 12 章, "逻辑存储结构"

12 Logical Storage Structures

This chapter describes the nature of and relationships among logical storage structures. These structures are created and recognized by Oracle Database and are not known to the operating system.

This chapter contains the following sections:

- Introduction to Logical Storage Structures
  - Logical Storage Hierarchy
  - Logical Space Management
- Overview of Data Blocks
  - Data Blocks and Operating System Blocks
  - Data Block Format
  - Data Block Compression
  - Space Management in Data Blocks
- Overview of Extents
  - Allocation of Extents
  - Deallocation of Extents
  - Storage Parameters for Extents
- Overview of Segments
  - User Segments
  - Temporary Segments
  - Undo Segments
  - Segment Space and the High Water Mark
- Overview of Tablespaces
  - Permanent Tablespaces
  - Temporary Tablespaces
  - Tablespace Modes
  - Tablespace File Size

Introduction to Logical Storage Structures

Oracle Database allocates logical space for all data in the database. The logical units of database space allocation are data blocks, extents, segments, and tablespaces. At a physical level, the data is stored in data files on disk (see Chapter 11, "Physical Storage Structures"). The data in

第 12 章 逻辑存储结构

本章介绍逻辑存储结构的特征及其之间的关系。这些结构由 Oracle 数据库创建和识别，而对操作系统是透明的。

本章包含以下各节：

- 逻辑存储结构简介
  - 逻辑存储层次结构
  - 逻辑空间管理
- 数据块概述
  - 数据块和操作系统块
  - 数据块格式
  - 数据块压缩
  - 数据块的空间管理
- 扩展区概述
  - 分配扩展区
  - 释放扩展区
  - 扩展区的存储参数
- 段概述
  - 用户段
  - 临时段
  - 撤消段
  - 段空间和高水位标记
- 表空间概述
  - 永久表空间
  - 临时表空间
  - 表空间模式
  - 表空间文件大小

逻辑存储结构简介

Oracle 数据库为数据库中的所有数据分配逻辑空间。数据库空间分配的逻辑单元是数据块、 扩展区、 段、和表空间。而在物理级，数据被存储在磁盘上的数据文件中 （请参阅第 11 章"物理存储结构"）。数据文件中的数据

the data files is stored in operating system blocks.

Figure 12-1 is an entity-relationship diagram for physical and logical storage. The crow's foot notation represents a one-to-many relationship.

Figure 12-1 Logical and Physical Storage

Description of "Figure 12-1 Logical and Physical Storage"

Logical Storage Hierarchy

Figure 12-2 shows the relationships among data blocks, extents, and segments within a tablespace. In this example, a segment has two extents stored in different data files.

Figure 12-2 Segments, Extents, and Data Blocks Within a Tablespace

Description of "Figure 12-2 Segments, Extents, and Data Blocks Within a

存储在操作系统块中。

图 12-1 是一个物理和逻辑存储的实体关系图。乌鸦脚表示法表示一对多关系。

图 12-1 逻辑和物理存储

Description of "Figure 12-1 Logical and Physical Storage"

逻辑存储层次结构

图 12-2 显示了在表空间中的数据块、 扩展区、和段之间的关系。在此示例中，一个段具有分别存储在不同数据文件中的两个扩展区。

图 12-2 表空间中的段、 扩展盘区、和数据块

Description of "Figure 12-2 Segments, Extents, and Data Blocks Within a

Tablespace"

At the finest level of granularity, Oracle Database stores data in data blocks. One logical data block corresponds to a specific number of bytes of physical disk space, for example, 2 KB. Data blocks are the smallest units of storage that Oracle Database can use or allocate.
An extent is a set of logically contiguous data blocks allocated for storing a specific type of information. In Figure 12-2, the 24 KB extent has 12 data blocks, while the 72 KB extent has 36 data blocks.

A segment is a set of extents allocated for a specific database object, such as a table. For example, the data for the `employees` table is stored in its own data segment, whereas each index for `employees` is stored in its own index segment. Every database object that consumes storage consists of a single segment.
Each segment belongs to one and only one tablespace. Thus, all extents for a segment are stored in the same tablespace. Within a tablespace, a segment can include extents from multiple data files, as shown in Figure 12-2. For example, one extent for a segment may be stored in `users01.dbf`, while another is stored in `users02.dbf`. A single extent can never span data files.
See Also:
"Overview of Data Files"

Logical Space Management

Oracle Database must use **logical space management** to track and allocate the extents in a tablespace. When a database object requires an extent, the database must have a method of finding and providing it. Similarly, when an object no longer requires an extent, the database must have a method of making the free extent available.
Oracle Database manages space within a tablespace based on the type that you create. You can create either of the following types of tablespaces:

- Locally managed tablespaces (default)

  The database uses bitmaps in the tablespaces themselves to manage extents. Thus, locally managed tablespaces have a part of the tablespace set aside for a bitmap. Within a tablespace, the database can manage segments with **automatic segment space management (ASSM)** or **manual segment space**

Tablespace"

在最细的粒度级别， Oracle 数据库将数据存储为数据块。一个逻辑数据块对应于特定字节数的物理磁盘空间，比如 2 KB。数据块是 Oracle 数据库可以使用或分配的最小存储单位。

扩展区是一组逻辑上连续的数据块，被分配来用于存储特定类型的信息。在图 12-2 中，这个 24 KB 的扩展区有 12 个数据块，而这个 72 KB 的扩展区有 36 个数据块。
段是为一个特定数据库对象（如一个表）分配的一组扩展区。例如，`employees` 表的数据存储在其自己的数据段中，而该表的每个索引存储在其自己的索引段中。会消耗存储空间的每个数据库对象都由单个段组成。

每个段属于且仅属于一个表空间。因此，一个段的所有扩展区存储在相同的表空间中。在一个表空间中，一个段可以包括多个数据文件中的扩展区，如图 12-2 所示。例如，段的一个扩展区可能存储在 users01.dbf 中，而另一个存储在 users02.dbf 中。单个扩展区绝不会跨越多个数据文件。

另见：
数据文件概述

逻辑空间管理

Oracle 数据库必须使用**逻辑空间管理**来在表空间中跟踪并分配扩展区。当数据库对象需要扩展区时，该数据库必须有查找和分配扩展区的方法。同样，当对象不再需要扩展区时，数据库必须提供一种方法来重用可用空间。

Oracle 数据库基于您创建的表空间的类型来管理其中的空间。您可以创建下列类型的表空间之一：

- 本地管理表空间 （默认值）

  数据库使用表空间本身中的位图来管理扩展区。因此，本地管理表空间需要预留表空间的一部分用于位图。在一个表空间中，数据库可以使用**自动段空间管理 （ASSM）** 或**手动段空间管理 （MSSM）**来管理段 。

management (MSSM).

- Dictionary-managed tablespaces

  The database uses the data dictionary to manage extents (see "Overview of the Data Dictionary").

Figure 12-3 shows the alternatives for logical space management in a tablespace.

Figure 12-3 Logical Space Management

Description of "Figure 12-3 Logical Space Management"

Locally Managed Tablespaces

A locally managed tablespace maintains a bitmap in the data file header to track free and used space in the data file body. Each bit corresponds to a group of blocks. When space is allocated or freed, Oracle Database changes the bitmap values to reflect the new status of the blocks.

The following graphic is a conceptual representation of bitmap-managed storage. A `1` in the header refers to used space, whereas a `0` refers to free space.

- 字典管理表空间

  数据库使用数据字典来管理扩展区 （请参阅"数据字典概述）。

图 12-3 显示了一个表空间中的逻辑空间管理方法的可选方案。

图 12-3 逻辑空间管理

Description of "Figure 12-3 Logical Space Management"

本地管理表空间

本地管理表空间在数据文件头中维护一个位图，以跟踪数据文件体中的可用空间和已用空间。每一位对应一组块。当空间被分配或释放时，Oracle 数据库更改位图值，以反映数据块的新状态。

下面的图形是位图管理存储的概念表示。标头中的 1 是指已用空间，而 0 指可用空间。

Description of the illustration cncpt332.gif

A locally managed tablespace has the following advantages:

- Avoids using the data dictionary to manage extents

  Recursive operations can occur in dictionary-managed tablespaces if consuming or releasing space in an extent results in another operation that consumes or releases space in a data dictionary table or undo segment.
- Tracks adjacent free space automatically

  In this way, the database eliminates the need to coalesce free extents.
- Determines the size of locally managed extents automatically

  Alternatively, all extents can have the same size in a locally managed tablespace and override object storage options.

Note:
Oracle strongly recommends the use of locally managed tablespaces with Automatic Segment Space Management.
Segment space management is an attribute inherited from the tablespace that contains the segment. Within a locally managed tablespace, the database can manage segments automatically or manually. For example, segments in tablespace `users` can be managed

Description of the illustration cncpt332.gif

本地管理表空间具有如下优势：

- 避免使用数据字典来管理扩展区

  如果消耗或释放一个扩展区会导致在数据字典表或撤消段中消耗或释放空间，则在字典管理的表空间中会发生递归操作。

- 自动跟踪相邻的可用空间

  通过这种方式，数据库消除了合并空闲扩展区的需要。

- 自动确定本地管理扩展区的大小

  或者，在本地管理表空间中所有的扩展区可以具有相同的大小，并覆盖对象存储选项。

注意：
Oracle 强烈建议使用自动段空间管理的本地管理表空间。

段空间管理是一个从包含该段的表空间继承来的属性。在一个本地管理表空间中，数据库可以自动地或手动地管理段。例如，`users` 表空间中的段使用自动管理，而 `tools` 表空间中的段使用手动管理。

automatically while segments in tablespace `tools` are managed manually.

Automatic Segment Space Management

The ASSM method uses bitmaps to manage space. Bitmaps provide the following advantages:

- Simplified administration

  ASSM avoids the need to manually determine correct settings for many storage parameters. Only one crucial SQL parameter controls space allocation: `PCTFREE`. This parameter specifies the percentage of space to be reserved in a block for future updates (see "Percentage of Free Space in Data Blocks").
- Increased concurrency

  Multiple transactions can search separate lists of free data blocks, thereby reducing contention and waits. For many standard workloads, application performance with ASSM is better than the performance of a well-tuned application that uses MSSM.
- Dynamic affinity of space to instances in an Oracle Real Application Clusters (Oracle RAC) environment

ASSM is more efficient and is the default for permanent, locally managed tablespaces.

Note:

This chapter assumes the use of ASSM in all of its discussions of logical storage space.

Manual Segment Space Management

The legacy MSSM method uses a linked list called a **free list** to manage free space in the segment. For a database object that has free space, a free list keeps track of blocks under the high water mark (HWM), which is the dividing line between segment space that is used and not yet used. As blocks are used, the database puts blocks on or removes blocks from the free list as needed.

In addition to `PCTFREE`, MSSM requires you to control space allocation with SQL parameters such as `PCTUSED`, `FREELISTS`, and `FREELIST GROUPS`. `PCTUSED` sets the percentage of free space that must exist in a currently used block for the database to put it on the free list. For example, if you set `PCTUSED` to `40` in a `CREATE TABLE` statement,

自动段空间管理(ASSM)

ASSM 方法使用位图管理空间。位图提供了以下优点：

- 简化管理

  ASSM 可以避免手动确定许多存储参数的正确设置的需要。只有一个很关键的控制空间分配的 SQL 参数： PCTFREE。此参数指定要为块中保留用于将来的更新的空间百分比（请参阅"数据块中的可用空间百分比"）。
- 增强并发性

  多个事务可以搜索多个相互独立的空闲数据块列表，从而减少争用和等待。对很多标准工作负载，使用 ASSM 的应用程序性能比使用优化得很好的使用 MSSM 的应用程序性能更好。
- 对 Oracle 真正应用集群 （Oracle RAC） 环境中实例的空间动态亲合性

ASSM 更有效，并且是永久性本地管理表空间的默认值。

注意：

本章假定在逻辑存储空间的所有讨论中使用 ASSM。

手动段空间管理（MSSM）

旧式的 MSSM 方法使用称为**空闲列表**的链表来管理段中的可用空间。对一个具有可用空间的数据库对象，有一个空闲列表会跟踪位于高水位线 (HWM) 之下的数据块，所谓高水位，即已使用段空间和未使用段空间之间的分界线。当块被使用时，数据库根据需要将块放入空闲列表，或将块从空闲列表中删除。

除了 PCTFREE，MSSM 需要您使用额外的几个 SQL 参数（如 `PCTUSED`、`FREELISTS`、和 `FREELIST GROUPS`）来控制空间分配。PCTUSED 设置在当前使用块中必须存在的可用空间百分比，当使用率低于该百分比时，数据库会将其放入空闲列表中。例如，如果 CREATE TABLE 语句中设置 PCTUSED 为 40，则只有当块空间使用少于 40%的情况下，您才能往段中

then you cannot insert rows into a block in the segment until less than 40% of the block space is used.

As an illustration, suppose you insert a row into a table. The database checks a free list of the table for the first available block. If the row cannot fit in the block, and if the used space in the block is greater than or equal to PCTUSED, then the database takes the block off the list and searches for another block. If you delete rows from the block, then the database checks whether used space in the block is now less than PCTUSED. If so, then the database places the block at the beginning of the free list.

An object may have multiple free lists. In this way, multiple sessions performing DML on a table can use different lists, which can reduce contention. Each database session uses only one free list for the duration of its session.

As shown in Figure 12-4, you can also create an object with one or more **free list groups**, which are collections of free lists. Each group has a **master free list** that manages the individual **process free lists** in the group. Space overhead for free lists, especially for free list groups, can be significant.

Figure 12-4 Free List Groups

Description of "Figure 12-4 Free List Groups"

Managing segment space manually can be complex. You must adjust PCTFREE and PCTUSED to reduce row migration (see "Chained and Migrated Rows") and avoid wasting space. For example, if every used block in a segment is half full, and if PCTUSED is 40, then the database does not permit inserts into any of these blocks. Because of the difficulty of fine-tuning space allocation parameters, Oracle strongly recommends

的块插入新行。

作为一个说明，假设向一个表中插入行。数据库检查该表的空闲列表，以查找第一个可用的块。如果行无法容纳进该块中，并在块中已使用空间大于或等于 PCTUSED，则数据库将该块从空闲列表中移除，并搜索另一个块。如果从块中删除行，则数据库检查块中的已使用空间现在是否小于 PCTUSED。如果是，则数据库将该块置于空闲列表的开头。

一个对象可能有多个空闲列表。通过这种方式，在表上执行 DML 的多个会话可以使用不同列表，以减少争用。每个数据库会话在其会话持续时间只使用一个空闲列表。

如图 12-4 所示，你也可以创建具有一个或多个**空闲列表组**的对象，空闲列表组是空闲列表的集合。每个组有一个**主空闲列表**，用于管理组中的各个的**处理空闲列表**。空闲列表、尤其是空闲列表组的空间开销，可能非常显著。

图 12-4 空闲列表组

Description of "Figure 12-4 Free List Groups"

手动管理段空间可能会很复杂。您必须调整 PCTFREE 和 PCTUSED，以减少行迁移 （请参阅"链接行和迁移行"） 和避免空间浪费。例如，如果段中的每个使用的块是半满，并且 PCTUSED 是 40， 则数据库不允许向这些块插入新行。由于微调空间分配参数很困难， Oracle 强烈建议使用 ASSM。在 ASSM 中，由 PCTFREE 确定是否可以将新行插入一个块中，但它不使用空闲列表，并忽略 PCTUSED。

ASSM. In ASSM, PCTFREE determines whether a new row can be inserted into a block, but it does not use free lists and ignores PCTUSED.

See Also:

- *Oracle Database Administrator's Guide* to learn about locally managed tablespaces
- *Oracle Database 2 Day DBA* and *Oracle Database Administrator's Guide* to learn more about automatic segment space management
- *Oracle Database SQL Language Reference* to learn about storage parameters such as PCTFREE and PCTUSED

Dictionary-Managed Tablespaces

A dictionary-managed tablespace uses the data dictionary to manage its extents. Oracle Database updates tables in the data dictionary whenever an extent is allocated or freed for reuse. For example, when a table needs an extent, the database queries the data dictionary tables, and searches for free extents. If the database finds space, then it modifies one data dictionary table and inserts a row into another. In this way, the database manages space by modifying and moving data.

The SQL that the database executes in the background to obtain space for database objects is **recursive SQL**. Frequent use of recursive SQL can have a negative impact on performance because updates to the data dictionary must be serialized. Locally managed tablespaces, which are the default, avoid this performance problem.

See Also:

Oracle Database Administrator's Guide to learn how to migrate tablespaces from dictionary-managed to locally managed

Overview of Data Blocks

Oracle Database manages the logical storage space in the data files of a database in units called **data blocks**, also called **Oracle blocks** or **pages**. A data block is the minimum unit of database I/O.

Data Blocks and Operating System Blocks

At the physical level, database data is stored in disk files made up of operating system blocks. An **operating system block** is the minimum unit of data that the operating system can read or write. In contrast, an Oracle block is a logical storage structure whose size and structure are not known to the operating system.

另见：

- 《*Oracle 数据库管理员指南*》了解本地管理表空间
- 《*Oracle 数据库 2 日 DBA*》 和 《*Oracle 数据库管理员指南*》了解自动段空间管理
- 《*Oracle 数据库 SQL 语言参考*》了解存储参数，如 PCTFREE 和 PCTUSED

字典管理表空间

字典管理表空间使用数据字典来管理其扩展区。每当分配或释放了一个扩展区时， Oracle 数据库更新数据字典中的表。例如，当表需要扩展区时，数据库查询数据字典表，并搜索空闲扩展区。如果数据库找到了空间，则修改一个数据字典表，并插入一行。按这种方式，数据库通过修改和移动数据来管理空间。

数据库在后台为数据库对象获取空间而执行的 SQL 是**递归 SQL**。频繁使用递归 SQL 可能会对性能有负面影响，因为必须串行化对数据字典的更新。而默认的本地管理表空间避免了这种性能问题。

另见：

《*Oracle 数据库管理员指南*》了解如何从字典管理表空间迁移到本地管理表空间

数据块概述

Oracle 数据库以**数据块**（也称为 **Oracle 块**或**页**）为单位，来管理数据库数据文件中的逻辑存储空间。数据块是数据库 I/O 的最小单位。

数据块和操作系统块

在物理级别，存储在磁盘文件中的数据库数据由操作系统块组成。**操作系统块**是操作系统可以读取或写入的最小数据单位。相比之下， Oracle 块是一个逻辑存储结构，其大小和结构对操作系统是透明的。

Figure 12-5 shows that operating system blocks may differ in size from data blocks. The database requests data in multiples of data blocks, not operating system blocks.

Figure 12-5 Data Blocks and Operating System Blocks

Description of "Figure 12-5 Data Blocks and Operating System Blocks"

When the database requests a data block, the operating system translates this operation into a requests for data in permanent storage. The logical separation of data blocks from operating system blocks has the following implications:

- Applications do not need to determine the physical addresses of data on disk.
- Database data can be striped or mirrored on multiple physical disks.

Database Block Size

Every database has a **database block size**. The `DB_BLOCK_SIZE` initialization parameter sets the data block size for a database when it is created. The size is set for the `SYSTEM` and `SYSAUX` tablespaces and is the default for all other tablespaces. The database block size cannot be changed except by re-creating the database.

If `DB_BLOCK_SIZE` is not set, then the default data block size is operating system-specific. The standard data block size for a database is 4 KB or 8 KB. If the size differs for data blocks and operating system blocks, then the data block size must be a multiple of the operating system block size.

See Also:

- *Oracle Database Reference* to learn about the `DB_BLOCK_SIZE` initialization parameter

图 12-5 显示操作系统块与数据块的大小可能有所不同。数据库按数据块（而不是按操作系统块）的倍数来请求数据。

图 12-5 数据块和操作系统块

Description of "Figure 12-5 Data Blocks and Operating System Blocks"

当数据库请求一个数据块时，操作系统将此操作转换为对永久存储数据的多个请求。数据块与操作系统块的逻辑分离具有以下含义：

- 应用程序不需要确定磁盘上的数据的物理地址。
- 数据库数据可以在多个物理磁盘上进行条带化或镜像。

数据库块大小

每个数据库都有一个**数据库块大小**。DB_BLOCK_SIZE 初始化参数在数据库被创建时设置其数据块大小。此大小是 `SYSTEM` 和 `SYSAUX` 表空间的大小，并且是其它表空间的默认大小。不能更改数据库的块大小，除非重新创建数据库。

如果尚未设置 DB_BLOCK_SIZE，则默认数据块大小特定于操作系统。数据库的标准数据块大小为 4 KB 或 8 KB。如果数据块和操作系统块的大小不同，则数据块大小必须是操作系统块大小的整数倍。

另见：

- 《*Oracle 数据库参考*》了解 DB_BLOCK_SIZE 初始化参数

- *Oracle Database Administrator's Guide* and *Oracle Database Performance Tuning Guide* to learn how to choose block sizes

Tablespace Block Size

You can create individual tablespaces whose block size differs from the `DB_BLOCK_SIZE` setting. A nonstandard block size can be useful when moving a transportable tablespace to a different platform.

See Also:

Oracle Database Administrator's Guide to learn how to specify a nonstandard block size for a tablespace

Data Block Format

Every data block has a **format** or internal structure that enables the database to track the data and free space in the block. This format is similar whether the data block contains table, index, or table cluster data. Figure 12-6 shows the format of an uncompressed data block (see "Data Block Compression" to learn about compressed blocks).

Figure 12-6 Data Block Format

Description of "Figure 12-6 Data Block Format"

Data Block Overhead

Oracle Database uses the block overhead to manage the block itself. The block overhead is not available to store user data. As shown in Figure 12-6, the block overhead includes the following parts:

- Block header

- 《*Oracle 数据库管理员指南*》和《*Oracle 数据库性能调整指南*》了解如何选择块大小

表空间块大小

你可以创建其块大小不同于 DB_BLOCK_SIZE 设定值的表空间。当你需要将一个可移动表空间移动到一个不同的平台时，非标准的块大小非常有用。

另见：

《*Oracle 数据库管理员指南*》了解如何为表空间指定非标准的块大小

数据块格式

每个数据块有一个**格式**或内部结构，使得数据库能够跟踪块中的数据和可用空间。各种数据块的格式是类似的，无论其包含的是表、 索引、或表簇数据。图 12-6 显示了一个未压缩的数据块的格式 （请参阅"数据块压缩"，以了解压缩块）。

图 12-6 数据块格式

Description of "Figure 12-6 Data Block Format"

数据块开销

Oracle 数据库使用块开销来管理块本身。块开销不能用来存储用户数据。如图 12-6， 块开销将包括以下部分：

- 块头

This part contains general information about the block, including disk address and segment type. For blocks that are transaction-managed, the block header contains active and historical transaction information.
A **transaction entry** is required for every transaction that updates the block. Oracle Database initially reserves space in the block header for transaction entries. In data blocks allocated to segments that support transactional changes, free space can also hold transaction entries when the header space is depleted. The space required for transaction entries is operating system dependent. However, transaction entries in most operating systems require approximately 23 bytes.

- Table directory

  For a heap-organized table, this directory contains metadata about tables whose rows are stored in this block. Multiple tables can store rows in the same block.
- Row directory

  For a heap-organized table, this directory describes the location of rows in the data portion of the block.
  After space has been allocated in the row directory, the database does not reclaim this space after row deletion. Thus, a block that is currently empty but formerly had up to 50 rows continues to have 100 bytes allocated for the row directory. The database reuses this space only when new rows are inserted in the block.

Some parts of the block overhead are fixed in size, but the total size is variable. On average, the block overhead totals 84 to 107 bytes.

Row Format

The row data part of the block contains the actual data, such as table rows or index key entries. Just as every data block has an internal format, every row has a **row format** that enables the database to track the data in the row.
Oracle Database stores rows as variable-length records. A row is contained in one or more **row pieces**. Each row piece has a **row header** and **column data**.
Figure 12-7 shows the format of a row.

Figure 12-7 The Format of a Row Piece

此部分包含关于块的一般信息，包括磁盘地址和段类型。对于事务管理块，其块头包含活动的和历史的事务信息。

每个更新块的事务都需要一个**事务条目**。Oracle 数据库预先在块头中为事务条目保留空间。在分配给段用于支持事务性更改的数据块中，当块头空间耗尽时，可用空间也可以容纳事务条目。事务条目所需的的空间取决于操作系统。但是，绝大多数操作系统中的事务条目需要大约 23 个字节。

- 表目录

  对于堆组织表，此目录包含有关其行存储在该块中的表的元数据。多个表可以将行存储在相同的块中。

- 行目录

  对于堆组织表，此目录描述该块的数据部分中的行的位置。

  当已在行目录中分配空间后，即使在行被删除后，数据库也不会回收此空间。因此，就算某块现在是空的，但若之前曾经达到 50 行，则在行目录仍会保留已分配的 100 字节。仅在块中插入新行时，数据库才会重用此空间。

块开销的某些部分是大小固定的，但总的大小是可变的。平均起来，块开销总计在 84 到 107 字节左右。

行格式

块的行数据部分包含实际数据，如表行或索引键条目等。正如每个数据块具有一个内部的格式，每一行也有一个**行格式**，使得数据库能够跟踪行中的数据。

Oracle 数据库以可变长度记录形式来存储行。行包含在一个或多个**行片断**中。每个行片断有一个**行头**和**列数据**。

图 12-7 显示了行格式。

图 12-7 行片断格式

Description of "Figure 12-7 The Format of a Row Piece"

Row Header

Oracle Database uses the row header to manage the row piece stored in the block. The row header contains information such as the following:

- Columns in the row piece
- Pieces of the row located in other data blocks

  If an entire row can be inserted into a single data block, then Oracle Database stores the row as one row piece. However, if all of the row data cannot be inserted into a single block or an update causes an existing row to outgrow its block, then the database stores the row in multiple row pieces (see "Chained and Migrated Rows"). A data block usually contains only one row piece per row.
- Cluster keys for table clusters (see "Overview of Table Clusters")

A row fully contained in one block has at least 3 bytes of row header.

Column Data

After the row header, the column data section stores the actual data in the row. The row piece usually stores columns in the order listed in the `CREATE TABLE` statement, but this order is not guaranteed. For

Description of "Figure 12-7 The Format of a Row Piece"

行头

Oracle 数据库使用行头来管理存储在块中的行片断。行头包含以下信息：

- 行片断中的各列
- 位于其他数据块中的各个行片断

  如果整个行可以插入到一个数据块中，则 Oracle 数据库将该行存储为一个行片断。但是，如果所有行数据不能插入一个单一的块，或者一个更新导致现有的行不能容纳在原来的块中，则数据库将该行存储为多个行片断（请参阅"链接行和迁移行"）。数据块中通常每行只包含一个行片断。
- 表簇的簇键（见"表簇概述"）

包含在一个块中的完全行至少有 3 个字节的行头。

列数据

在行头之后的列数据部分存储行中的实际数据。行片断通常按 `CREATE TABLE` 语句中列出的顺序来存储列，但这个顺序并不总是能保证的。例如，`LONG` 类型列总是在最后。

example, columns of type LONG are created last.

As shown in Figure 12-7, for each column in a row piece, Oracle Database stores the column length and data separately. The space required depends on the data type. If the data type of a column is variable length, then the space required to hold a value can grow and shrink with updates to the data.

Each row has a slot in the row directory of the data block header. The slot points to the beginning of the row.

See Also:

"Table Storage" and "Index Storage"

Rowid Format

Oracle Database uses a rowid to uniquely identify a row. Internally, the rowid is a structure that holds information that the database needs to access a row. A rowid is not physically stored in the database, but is inferred from the file and block on which the data is stored.

An **extended rowid** includes a data object number. This rowid type uses a base 64 encoding of the physical address for each row. The encoding characters are A-Z, a-z, 0-9, +, and /.

Example 12-1 queries the ROWID pseudocolumn to show the extended rowid of the row in the employees table for employee 100.

Example 12-1 ROWID Pseudocolumn

```
SQL> SELECT ROWID FROM employees WHERE employee_id = 100;

ROWID
------------------
AAAPecAAFAAAABSAAA
```

Figure 12-8 illustrates the format of an extended rowid.

Figure 12-8 ROWID Format

Description of "Figure 12-8 ROWID Format"

An extended rowid is displayed in a four-piece format, OOOOOOFFFBBBBBBRRR, with the format divided into the following components:

如图 12-7 所示，对行片断中的每一列， Oracle 数据库独立地存储列长度和列数据。所需的空间取决于数据类型。如果列的数据类型是可变长度的，则用于容纳一个值所需的空间可能在其数据被更新时会增长和收缩。

每一行都在数据块标头的行目录中有一个槽位。槽位指向行的开始部分。

另见：

"表存储"和"索引存储"

Rowid 格式

Oracle 数据库使用一个 rowid 唯一地标识一行。在内部，rowid 是一个结构，用于保存数据库访问行所需要的信息。一个 rowid 并不物理地存储在数据库中，而是从存储数据的文件和块推导而来的。

扩展的 rowid 包括数据对象号。这种 rowid 类型使用每个行的物理地址的 64 进位编码。编码的字符为 A-Z、a-z、0-9、+、和/。

示例 12-1 查询 ROWID 伪列来显示 employees 表中雇员 100 的所在行的扩展 rowid。

示例 12-1 ROWID 伪列

```
SQL> SELECT ROWID FROM employees WHERE employee_id = 100;

ROWID
------------------
AAAPecAAFAAAABSAAA
```

图 12-8 说明了一个扩展的 rowid 的格式。

图 12-8 ROWID 格式

Description of "Figure 12-8 ROWID Format"

一个扩展 rowid 以一个四段式格式显示，OOOOOOFFFBBBBBBRRR，此格式分为以下几个组件：

- OOOOOO

  The **data object number** identifies the segment (data object AAAPec in Example 12-1). A data object number is assigned to every database segment. Schema objects in the same segment, such as a table cluster, have the same data object number.
- FFF

  The tablespace-relative **data file number** identifies the data file that contains the row (file AAF in Example 12-1).
- BBBBBB

  The **data block number** identifies the block that contains the row (block AAAABS in Example 12-1). Block numbers are relative to their data file, not their tablespace. Thus, two rows with identical block numbers could reside in different data files of the same tablespace.
- RRR

  The **row number** identifies the row in the block (row AAA in Example 12-1).

After a rowid is assigned to a row piece, the rowid can change in special circumstances. For example, if **row movement** is enabled, then the rowid can change because of partition key updates, Flashback Table operations, shrink table operations, and so on. If row movement is disabled, then a rowid can change if the row is exported and imported using Oracle Database utilities.

Note:

Internally, the database performs row movement as if the row were physically deleted and reinserted. However, row movement is considered an update, which has implications for triggers.

See Also:

- "Rowid Data Types"
- *Oracle Database SQL Language Reference* to learn about rowids

Data Block Compression

The database can use table compression to eliminate duplicate values in a data block (see "Table Compression"). This section describes the format of

- OOOOOO

  数据对象号标识段 （如示例中 12-1 中的数据对象 AAAPec)。数据库中的每个段都被分配了一个数据对象号。同一段中的模式对象（如一个表簇）具有相同的数据对象号。

- FFF

  表空间相对**数据文件号**，标识包含行的数据文件 (如示例 12-1 中的文件 AAF)。
- BBBBBB

  数据块号标识包含行的块 （如示例 12-1 中的块 AAAABS）。块号是相对于他们的数据文件的，而不是其表空间。因此，具有相同块号的两行，可以驻留在同一表空间的不同数据文件中。

- RRR

  行号标识块中的行（如示例 12-1 中的 AAA）。

在一个 rowid 被分配给一个行片断后，该 rowid 在特殊情况下可以更改。例如，如果启用了**行移动**，则 rowid 可能会因为分区键更新、 闪回表操作、收缩表操作等而发生变化。如果禁用了行移动，则如果使用 Oracle 数据库实用程序导出和导入了行，其 rowid 可能会发生变化。

注意：

在内部，数据库执行行移动，就像行是被物理地删除、然后又重新插入。不过，行移动被认为是更新，会隐含触发触发器。

另见：

- "Rowid 数据类型"
- 《*Oracle 数据库 SQL 语言参考*》了解 rowid

数据块压缩

数据库可以使用表压缩来消除在数据块中的重复值（请参阅"表压缩"）。本节介绍使用压缩的数据块格式。

data blocks that use compression.

The format of a data block that uses basic and OLTP table compression is essentially the same as an uncompressed block. The difference is that a **symbol table** at the beginning of the block stores duplicate values for the rows and columns. The database replaces occurrences of these values with a short reference to the symbol table.

Assume that the rows in Example 12-2 are stored in a data block for the seven-column `sales` table.

Example 12-2 Rows in sales Table

```
2190,13770,25-NOV-00,S,9999,23,161
2225,15720,28-NOV-00,S,9999,25,1450
34005,120760,29-NOV-00,P,9999,44,2376
9425,4750,29-NOV-00,I,9999,11,979
1675,46750,29-NOV-00,S,9999,19,1121
```

When basic or OLTP table compression is applied to this table, the database replaces duplicate values with a symbol reference. Example 12-3 is a conceptual representation of the compression in which the symbol `*` replaces `29-NOV-00` and `%` replaces `9999`.

Example 12-3 OLTP Compressed Rows in sales Table

```
2190,13770,25-NOV-00,S,%,23,161
2225,15720,28-NOV-00,S,%,25,1450
34005,120760,*,P,%,44,2376
9425,4750,*,I,%,11,979
1675,46750,*,S,%,19,1121
```

Table 12-1 conceptually represents the symbol table that maps symbols to values.

Table 12-1 Symbol Table

Symbol	Value	Column	Rows
*	29-NOV-00	3	958-960
%	9999	5	956-960

Space Management in Data Blocks

As the database fills a data block from the bottom up, the amount of **free space** between the row data and the block header decreases. This free space can also shrink during updates, as when changing a trailing null to a nonnull value. The database manages free space in the data block to optimize performance and avoid wasted space.

使用基本表压缩和 OLTP 表压缩的数据块，其格式与一个未压缩的块实质上是相同的。区别是使用位于块开头的**符号表**来存储行和列的重复值。数据库使用一个到符号表的短引用来替换这些重复出现的值。

假定示例 12-2 中的行，存储在具有七个列的 `sales` 表的某个数据块中。

*示例 **12-2 sales** 表中的行*

```
2190,13770,25-NOV-00,S,9999,23,161
2225,15720,28-NOV-00,S,9999,25,1450
34005,120760,29-NOV-00,P,9999,44,2376
9425,4750,29-NOV-00,I,9999,11,979
1675,46750,29-NOV-00,S,9999,19,1121
```

当在此表上应用基本或 OLTP 表压缩时，数据库会将重复值替换为一个符号引用。示例 12-3 是压缩的一个概念性展示，使用符号*替换了 `29-NOV-00`，而使用%替换了 `9999`。

*示例 **12-3 sales** 表中的 **OLTP** 压缩行*

```
2190,13770,25-NOV-00,S,%,23,161
2225,15720,28-NOV-00,S,%,25,1450
34005,120760,*,P,%,44,2376
9425,4750,*,I,%,11,979
1675,46750,*,S,%,19,1121
```

表 12-1 从概念上展示了从符号映射到值的符号表。

*表 **12-1** 符号表*

称号	值	列	行
*	29-NOV-00	3	958-960
%	9999	5	956-960

数据块的空间管理

随着数据库从底向上不断填充数据块，在行数据和块头之间的**可用空间**就会逐渐减少。在更新期间，当将一个尾部空值更改为非空值时，可用空间也会变少。数据库会管理数据块中的可用空间，以优化性能并避免空间浪费。

Note:
This section assumes the use of automatic segment space management.

Percentage of Free Space in Data Blocks

The `PCTFREE` storage parameter is essential to how the database manages free space. This SQL parameter sets the minimum percentage of a data reserved as free space for updates to existing rows. Thus, `PCTFREE` is important for preventing row migration and avoiding wasted space.

For example, assume that you create a table that will require only occasional updates, most of which will not increase the size of the existing data. You specify the `PCTFREE` parameter within a `CREATE TABLE` statement as follows:

```
CREATE TABLE test_table (n NUMBER) PCTFREE 20;
```

Figure 12-9 shows how a `PCTFREE` setting of `20` affects space management. The database adds rows to the block over time, causing the row data to grow upwards toward the block header, which is itself expanding downward toward the row data. The `PCTFREE` setting ensures that *at least* 20% of the data block is free. For example, the database prevents an `INSERT` statement from filling the block so that the row data and header occupy a combined 90% of the total block space, leaving only 10% free.

Figure 12-9 PCTFREE

Description of "Figure 12-9 PCTFREE"

注意：
本节假定使用自动段空间管理。

数据块中的可用空间百分比

PCTFREE 存储参数对于数据库如何管理其可用空间非常重要。此 SQL 参数设置为更新现有的行而保留的可用空间最小百分比。因此，PCTFREE 对于防止行迁移并避免空间浪费非常重要。

例如，假设您创建了一个只是偶尔需要更新的表，其中大多数更新都不会增加现有数据的大小。你可以在 CREATE TABLE 语句中指定 PCTFREE 参数，如下所示：

```
CREATE TABLE test_table (n NUMBER) PCTFREE 20;
```

图 12-9 显示了将 PCTFREE 设置为 20 是如何影响空间管理的。数据库不断将行添加到块，这将导致行数据朝着块头方向向上不断增长，而块头本身又朝行数据方向向下不断扩大。该 PCTFREE 设置确保至少 20%的数据块是空闲的。例如，数据库可以防止一个 `INSERT` 语句填充该块，使行数据和标头一共占据块总空间的 90%，而只剩 10%可用空间。

图 12-9 PCTFREE

Description of "Figure 12-9 PCTFREE"

Note:
This discussion does not apply to LOB data types, which do not use the PCTFREE storage parameter or free lists. See "Overview of LOBs".

See Also:
Oracle Database SQL Language Reference for the syntax and semantics of the PCTFREE parameter

Optimization of Free Space in Data Blocks

While the percentage of free space cannot be *less* than PCTFREE, the amount of free space can be *greater*. For example, a PCTFREE setting of 20% prevents the total amount of free space from dropping to 5% of the block, but permits 50% of the block to be free space. The following SQL statements can increase free space:

- DELETE statements
- UPDATE statements that either update existing values to smaller values or increase existing values and force a row to migrate
- INSERT statements on a table that uses OLTP compression

  If inserts fill a block with data, then the database invokes block compression, which may result in the block having more free space.

The space released is available for INSERT statements under the following conditions:

- If the INSERT statement is in the same transaction and after the statement that frees space, then the statement can use the space.
- If the INSERT statement is in a separate transaction from the statement that frees space (perhaps run by another user), then the statement can use the space made available only after the other transaction commits and only if the space is needed.

See Also:
Oracle Database Administrator's Guide to learn about OLTP compression

Coalescing Fragmented Space

Released space may or may not be contiguous with the main area of free space in a data block, as shown in Figure 12-10. Noncontiguous free space is called **fragmented space**.

Figure 12-10 Data Block with Fragmented Space

注意：
本讨论并不适用于 LOB 数据类型，它不使用 PCTFREE 存储参数或空闲列表。请参阅 "LOB 概述"。

另见：
《*Oracle 数据库 SQL 语言参考*》关于 PCTFREE 参数的语法和语义

优化数据块中的可用空间

虽然可用空间的百分比不能小于 PCTFREE，但其可用空间量可以更大。例如， 20%的 PCTFREE 设置可以防止可用空间总量少于 5%，但它允许该块具有 50%的可用空间。下面的 SQL 语句可能会增加可用空间：

- DELETE 语句
- UPDATE 语句，要么将现有值更新为更小的值，或者增大现有值并被强制迁移到另一个行
- 在使用了 OLTP 压缩的表上的 INSERT 语句

  如果插入操作往块中填充了数据，则数据库会调用块压缩，结果可能会导致在块中有更多的可用空间。

在下列条件下，释放的空间可供 INSERT 语句使用：

- 如果 INSERT 语句是位于释放空间的语句之后，且处于相同的事务中，则该语句可以使用已释放的空间。
- 如果 INSERT 语句与释放空间的语句位于各自单独的事务中（也许是由另一个用户运行的），只要另一个事务已提交，则该语句可以在需要此空间时使用它。

另见：
《*Oracle 数据库管理员指南*》了解 OLTP 压缩

合并碎片空间

被释放的空间与数据块可用空间的主要区域可能是连续的，也可能是不连续的，如图 12-10 所示。不连续的可用空间称为**碎片空间**。

图 12-10 具有碎片空间的数据块

Description of "Figure 12-10 Data Block with Fragmented Space"

Oracle Database automatically and transparently coalesces the free space of a data block *only* when the following conditions are true:

- An `INSERT` or `UPDATE` statement attempts to use a block that contains sufficient free space to contain a new row piece.
- The free space is fragmented so that the row piece cannot be inserted in a contiguous section of the block.

After coalescing, the amount of free space is identical to the amount before the operation, but the space is now contiguous. Figure 12-11 shows a data block after space has been coalesced.

Figure 12-11 Data Block After Coalescing Free Space

Description of "Figure 12-10 Data Block with Fragmented Space"

只有在满足以下条件时，Oracle 数据库才会自动且透明地合并数据块中的可用空间，

- `INSERT` 或 `UPDATE` 语句试图使用的块，包含足够的可用空间来容纳新的行片断。
- 可用空间已被碎片化，以至于该行片断不能插入到块中的某一个连续区域。

合并后，可用空间量与操作之前是相同的，只不过该空间现在已成为连续的。图 12-11 显示一个数据块的空间被合并了之后的情况。

图 12-11 数据块合并后的可用空间

Description of "Figure 12-11 Data Block After Coalescing Free Space"

Oracle Database performs coalescing only in the preceding situations because otherwise performance would decrease because of the continuous coalescing of the free space in data blocks.

Reuse of Index Space

The database can reuse space within an index block. For example, if you insert a value into a column and delete it, and if an index exists on this column, then the database can reuse the index slot when a row requires it.

The database can reuse an index block itself. Unlike a table block, an index block only becomes free when it is empty. The database places the empty block on the free list of the index structure and makes it eligible for reuse. However, Oracle Database does not automatically compact the index: an `ALTER INDEX REBUILD` or `COALESCE` statement is required.

Figure 12-12 represents an index of the `employees.department_id` column before the index is coalesced. The first three leaf blocks are only partially full, as indicated by the gray fill lines.

Figure 12-12 Index Before Coalescing

Description of "Figure 12-11 Data Block After Coalescing Free Space"

Oracle 数据库仅在上述情况下执行合并，否则会因为连续地合并数据块中的可用空间反而降低性能。

重用索引空间

数据库可以重复使用一个索引块内的空间。例如，如果您往列中插入了一个值，又将其删除，而对此列的索引仍然存在，则数据库可以在某行需要时，重新使用该索引槽位。

数据库可以重复使用一个索引块本身。与表块不同，一个索引块仅当其完全为空时，它才是空闲的。数据库将空块放入索引结构的空闲列表中并使其可重新使用。然而，Oracle 数据库不会自动压缩索引： 这需要一个 `ALTER INDEX REBUILD` 或 `COALESCE` 语句。

图 12-12 代表在 employees.department_id 列上的一个索引在其合并之前的情况。前面三个叶块只是部分填满的，如灰色填充线所示。

图 12-12 索引合并之前

Description of "Figure 12-12 Index Before Coalescing"

Figure 12-13 shows the index in Figure 12-12 after the index has been coalesced. The first two leaf blocks are now full, as indicated by the gray fill lines, and the third leaf block has been freed.

Figure 12-13 Index After Coalescing

Description of "Figure 12-12 Index Before Coalescing"

图 12-13 显示图 12-12 中的索引在合并后的情况。前两个叶块现在已充满，如灰色填充线所示，而第三个叶块已被释放了。

*图 **12-13** 索引合并后*

Description of "Figure 12-13 Index After Coalescing"

See Also:

- *Oracle Database Administrator's Guide* to learn how to coalesce and rebuild indexes
- *Oracle Database SQL Language Reference* to learn about the `COALESCE` statement

Chained and Migrated Rows

Oracle Database must manage rows that are too large to fit into a single block. The following situations are possible:

- The row is too large to fit into one data block when it is first inserted.

  In row chaining, Oracle Database stores the data for the row in a **chain** of one or more data blocks reserved for the segment. Row chaining most often occurs with large rows. Examples include rows that contain a column of data type `LONG` or `LONG RAW`, a `VARCHAR2(4000)` column in a 2 KB block, or a row with a huge

Description of "Figure 12-13 Index After Coalescing"

另见：

- 《*Oracle 数据库管理员指南*》了解如何合并和重新生成索引
- 《*Oracle 数据库 SQL 语言参考*》了解 `COALESCE` 语句

链接和已迁移的行

Oracle 数据库必须管理太大而不能放入一个单一块的行。可能存在下列情况：

- 该行太大，在它第一次插入时，无法放入一个数据块。

  在行链接中，Oracle 数据库将数据存储在为段保留的一个或多个被**链接**的数据块中。行链接最经常出现在大行中。示例包括包含 `LONG` 或 `LONG RAW` 数据类型列的行、或在 2 KB 块中的 VARCHAR2(4000) 列、或具有大量列的行。在这些情况下链接行是不可避免的。

number of columns. Row chaining in these cases is unavoidable.

- A row that originally fit into one data block is updated so that the overall row length increases, but insufficient free space exists to hold the updated row.

  In row migration, Oracle Database moves the entire row to a new data block, assuming the row can fit in a new block. The original row piece of a migrated row contains a pointer or "forwarding address" to the new block containing the migrated row. The rowid of a migrated row does not change.
- A row has more than 255 columns.

  Oracle Database can only store 255 columns in a row piece. Thus, if you insert a row into a table that has 1000 columns, then the database creates 4 row pieces, typically chained over multiple blocks.

Figure 12-14 depicts shows the insertion of a large row in a data block. The row is too large for the left block, so the database chains the row by placing the first row piece in the left block and the second row piece in the right block.

Figure 12-14 Row Chaining

Description of "Figure 12-14 Row Chaining"

Figure 12-15, the left block contains a row that is updated so that the row is now too large for the block. The database moves the entire row to the right block and leaves a pointer to the migrated row in the left block.

- 本来可以放入一个数据块的行，在更新后整体行长增加了，但没有足够的可用空间来容纳更新的行。

  在行迁移中，假设行可以容纳在一个新块中，Oracle 数据库将整个行移动到一个新的数据块。已迁移行的原始行片断，包含一个指针或“转发地址”到包含已迁移行的新块。已迁移行的 rowid 不会更改。

- 超过 255 个列的行。

  Oracle 数据库在一个行片断中只能存储 255 的列。因此，如果你在具有 1000 个列的表中插入行，则数据库将创建 4 个行片断，通常会链接多个块。

图 12-14 描述了向数据块中插入大行的情形。行对于左边的块太大，所以数据库通过将第一个行片断放入在左边的块，将第二个行片断放入右边的块中，形成行链接。

图 12-14 行链接

Description of "Figure 12-14 Row Chaining"

在图 12-15 中，左边的块包含的行被更新，致使该行现在太大而不能放入块中。数据库将整个行移动到右边的块中，并在左边的块中保留一个指向被迁移行的指针。

Figure 12-15 Row Migration

Description of "Figure 12-15 Row Migration"

When a row is chained or migrated, the I/O needed to retrieve the data increases. This situation results because Oracle Database must scan multiple blocks to retrieve the information for the row. For example, if the database performs one I/O to read an index and one I/O to read a nonmigrated table row, then an additional I/O is required to obtain the data for a migrated row.

The Segment Advisor, which can be run both manually and automatically, is an Oracle Database component that identifies segments that have space available for reclamation. The advisor can offer advice about objects that have significant free space or too many chained rows.

See Also:

- "Row Storage" and "Rowids of Row Pieces"
- *Oracle Database 2 Day DBA* and *Oracle Database Administrator's Guide* to learn how to reclaim wasted space
- *Oracle Database Performance Tuning Guide* to learn about reducing chained and migrated rows

Overview of Extents

An **extent** is a logical unit of database storage space allocation made up of contiguous data blocks. Data blocks in an extent are logically contiguous but can be physically spread out on disk because of RAID striping and file system implementations.

图 12-15 行迁移

Description of "Figure 12-15 Row Migration"

当行被链接或迁移时，检索数据所需的 I/O 会增加。会导致这种情况是因为 Oracle 数据库必须扫描多个块以检索行信息。例如，如果数据库执行一个 I/O 读取索引和一个 I/O 读取一个未迁移的表行，则对于一个迁移行需要一个额外的 I/O 才能获得数据。

段顾问是一个 Oracle 数据库组件，可以手动或自动执行，使用此工具可以标识有空间可回收的段。这个顾问工具可以对具有很多可用空间的对象，或有太多行链接的对象，提供建议。

另见：

- "行存储"和"行片断的 Rowids"
- 《*Oracle 数据库 2 日 DBA*》 和 《*Oracle 数据库管理员指南*》了解如何回收浪费的空间
- 《*Oracle 数据库性能调整指南*》 了解如何减少行链接和行迁移

扩展区概述

扩展区是数据库存储空间分配的逻辑单位，由多个连续数据块组成。在一个扩展区中的数据块在逻辑上是连续的，但在物理上可能由于 RAID 条带化和文件系统的实施方式而在磁盘上分散分布。

Allocation of Extents

By default, the database allocates an **initial extent** for a data segment when the segment is created. An extent is always contained in one data file.

Although no data has been added to the segment, the data blocks in the initial extent are reserved for this segment exclusively. The first data block of every segment contains a directory of the extents in the segment. Figure 12-16 shows the initial extent in a segment in a data file that previously contained no data.

Figure 12-16 Initial Extent of a Segment

Description of "Figure 12-16 Initial Extent of a Segment"

If the initial extent become full, and if more space is required, then the database automatically allocates an **incremental extent** for this segment. An incremental extent is a subsequent extent created for the segment.

The allocation algorithm depends on whether the tablespace is locally managed or dictionary-managed. In the locally managed case, the database searches the bitmap of a data file for adjacent free blocks. If the data file has insufficient space, then the database looks in another data file. Extents for a segment are always in the same tablespace but may be in different data files.

Figure 12-17 shows that the database can allocate extents for a segment in any data file in the tablespace. For example, the segment can allocate the initial extent in `users01.dbf`, allocate the first incremental extent in `users02.dbf`, and allocate the next extent in `users01.dbf`.

Figure 12-17 Incremental Extent of a Segment

分配扩展区

在默认情况下，在数据段被创建时，数据库会为数据段分配一个**初始扩展区**。一个扩展区总是包含在一个数据文件中。

虽然还没有数据被添加到段中，在初始扩展区中的数据块被专门为段保留起来。每个段的第一个数据块包含该段中的扩展区目录。图 12-16 显示了一个数据文件中某个段中的初始扩展区，它之前还没有包含任何数据

图 12-16 段的初始扩展区

Description of "Figure 12-16 Initial Extent of a Segment"

如果初始扩展区满了，并需要更多空间时，则数据库自动为该段分配一个**增量扩展区**。一个增量扩展区是为该段创建的后续扩展区。

分配算法取决于表空间是本地管理的还是字典管理的。对于本地管理表空间，数据库搜索一个数据文件的位图，以查找相邻的可用数据块。如果该数据文件中没有足够的空间，则数据库到另一个数据文件中查找。一个段的扩展区始终位于同一表空间中，但可能在不同的数据文件中。

图 12-17 显示，数据库可能为一个段，在表空间中的任意数据文件中分配扩展区。例如，段可能在 users01.dbf 中分配初始扩展区， 在 users02.dbf 中分配第一个增量扩展区，并在 users01.dbf 中分配下一个扩展区。

图 12-17 段的增量扩展区

Description of "Figure 12-17 Incremental Extent of a Segment"

The blocks of a newly allocated extent, although they were free, may not be empty of old data. In ASSM, Oracle Database formats the blocks of a newly allocated extent when it starts using the extent, but only as needed (see "Segment Space and the High Water Mark").

Note:

This section applies to serial operations, in which one server process parses and runs a statement. Extents are allocated differently in parallel SQL statements, which entail multiple server processes.

See Also:

Oracle Database Administrator's Guide to learn how to manually allocate extents

Deallocation of Extents

In general, the extents of a user segment do not return to the tablespace unless you drop the object using a `DROP` command. In Oracle Database 11*g* Release 2 (11.2.0.2), you can also drop the segment using the `DBMS_SPACE_ADMIN` package. For example, if you delete all rows in a table, then the database does not reclaim the data blocks for use by other objects in the tablespace.

Note:

In an undo segment, Oracle Database periodically deallocates one or more extents if it has the `OPTIMAL` size specified or if the database is in automatic undo management mode (see "Undo Tablespaces").

Description of "Figure 12-17 Incremental Extent of a Segment"

在新分配扩展区中的块，虽然他们是可用的，但可能并未清空旧数据。在 ASSM 中，对新分配扩展区中的块，只有当 Oracle 数据库开始使用它们时，才会根据需要将其格式化（请参阅"段空间和高水位标记"）。

注意：

本节仅适用于串行操作，即一个进程解析并运行一条语句。但在并行 SQL 语句中的扩展区具有不同的分配方式，它需要多个服务器进程。

另见：

《*Oracle 数据库管理员指南*》了解如何手动分配扩展区

释放扩展区

通常，用户段的扩展区不会返回给表空间，除非您使用 DROP 命令删除该对象。在 Oracle 数据库 11g 第 2 版中 (11.2.0.2)也可以使用 DBMS_SPACE_ADMIN 软件包删除段。例如，如果你删除了表中的所有行，但数据库不会回收数据块以供表空间中的其他对象使用。

注意：

在一个撤消段中， 如果指定了 `OPTIMAL` 大小参数，或数据库处于自动撤消管理模式，则 Oracle 数据库会周期性地释放一个或多个扩展区（请参见"撤消表空间"）。

In some circumstances, you can manually deallocate space. The Oracle Segment Advisor helps determine whether an object has space available for reclamation based on the level of fragmentation in the object. The following techniques can free extents:

- You can use an **online segment shrink** to reclaim fragmented space in a segment. Segment shrink is an online, in-place operation. In general, data compaction leads to better cache utilization and requires fewer blocks to be read in a full table scan.
- You can move the data of a nonpartitioned table or table partition into a new segment, and optionally into a different tablespace for which you have quota.
- You can rebuild or coalesce the index (see "Reuse of Index Space").
- You can **truncate** a table or table cluster, which removes all rows. By default, Oracle Database deallocates all space used by the removed rows except that specified by the `MINEXTENTS` storage parameter. In Oracle Database 11*g* Release 2 (11.2.0.2), you can also use `TRUNCATE` with the `DROP ALL STORAGE` option to drop entire segments.
- You can deallocate unused space, which frees the unused space at the high water mark end of the database segment and makes the space available for other segments in the tablespace (see "Segment Space and the High Water Mark").

When extents are freed, Oracle Database modifies the bitmap in the data file for locally managed tablespaces to reflect the regained extents as available space. Any data in the blocks of freed extents becomes inaccessible.

See Also:

Oracle Database Administrator's Guide to learn how to reclaim segment space

Storage Parameters for Extents

Every segment is defined by **storage parameters** expressed in terms of extents. These parameters control how Oracle Database allocates free space for a segment.

The storage settings are determined in the following order of precedence, with setting higher on the list overriding settings lower on the list:

在某些情况下，可以手动释放空间。基于对象中的碎片级别，Oracle 数据段顾问工具可帮助确定一个对象是否有空间可回收。以下技术可以释放扩展区：

- 你可以使用**在线段收缩**来回收在段中的零碎空间。段收缩是一种在线、 就地的操作。通常，紧凑的数据会获得更好的缓存利用率，且在全表扫描中只需读取更少的块。
- 您可以将非分区表或表分区的数据移动到一个新的段，或可选地移入另一个你在其中具有配额的表空间。
- 您可以重建或合并索引 （请参见"重用索引空间"）。
- 您可以**截断**表或表簇，这将删除所有行。默认情况下，除了留下由 `MINEXTENTS` 存储参数所指定的扩展区数之外，Oracle 数据库将释放已删除行所使用的所有空间。在数据库 11g 第 2 版 (11.2.0.2) 中，您还可以用带有 `DROP ALL STORAGE` 选项的 `TRUNCATE` 语句来删除整个段。
- 您可以释放未使用的空间，即释放数据库段的高水位端的未使用空间，以便供表空间中的其他段使用（请参阅"段空间和高水位标记"）。

当扩展区被释放时，Oracle 数据库将修改本地管理表空间的数据文件中的位图，以反映重新获得作为可用空间的扩展区。被释放扩展区的数据块中的任何数据变得不可访问。

另见：

《*Oracle 数据库管理员指南*》了解如何回收段空间

扩展区的存储参数

每一个段是由扩展区表示的**存储参数**来定义的。这些参数控制 Oracle 数据库如何为段分配可用空间。

存储设置按如下的优先级顺序确定，列表中较高的设置会覆盖较低的设置：

- ✧ 1. Segment storage clause
- ✧ 2. Tablespace storage clause
- ✧ 3. Oracle Database default

A locally managed tablespace can have either uniform extent sizes or variable extent sizes determined automatically by the system:

- For **uniform extents**, you can specify an extent size or use the default size of 1 MB. All extents in the tablespace are of this size. Locally managed temporary tablespaces can only use this type of allocation.
- For **automatically allocated extents**, Oracle Database determines the optimal size of additional extents.

For locally managed tablespaces, some storage parameters cannot be specified at the tablespace level. However, you can specify these parameters at the segment level. In this case, the databases uses all parameters together to compute the initial size of the segment. Internal algorithms determine the subsequent size of each extent.

See Also:

- *Oracle Database Administrator's Guide* to learn about extent management considerations when creating a locally managed tablespace
- *Oracle Database SQL Language Reference* to learn about options in the storage clause

Overview of Segments

A segment is a set of extents that contains all the data for a logical storage structure within a tablespace. For example, Oracle Database allocates one or more extents to form the data segment for a table. The database also allocates one or more extents to form the index segment for a table.

As explained in "Logical Space Management", Oracle Database manages segment space automatically or manually. This section assumes the use of ASSM.

User Segments

A single data segment in a database stores the data for one user object. There are different types of segments. Examples of **user segments** include:

- ✧ 1. 段存储子句
- ✧ 2. 表空间存储子句
- ✧ 3. Oracle 数据库缺省值

本地管理表空间可以具有统一的扩展区大小，或由系统自动确定可变的扩展区大小：

- 对**统一的扩展区**，可以指定扩展区的大小，或使用默认的大小 1 MB。表空间中的所有扩展区都是一样大小。本地管理的临时表空间只能使用这种分配类型。
- 对于**自动分配的扩展区**由 Oracle 数据库确定新分配扩展区的最佳大小。

对本地管理表空间，其某些存储参数不能在表空间级指定。但是，您可以在段级指定这些参数。在这种情况下，数据库使用所有参数一起计算段的初始大小。由内部算法确定每个后续扩展区的大小。

另见：

- 《*Oracle 数据库管理员指南*》了解当创建本地管理表空间时的扩展区管理注意事项
- 《*Oracle 数据库 SQL 语言参考*》了解存储子句中的选项

段概述

段是一个表空间内包含某个逻辑存储结构的所有数据的扩展区集合。例如，Oracle 数据库分配一个或多个扩展区，形成一个表的数据段。数据库还分配了一个或多个扩展区，形成一个表的索引段。

如"逻辑空间管理"所述， Oracle 数据库可以自动或手动地管理段空间。本节假定使用 ASSM 。

用户段

在数据库中的单个数据段用于存储一个用户对象的数据。段有不同的类型。**用户段**的示例包括：

- Table, table partition, or table cluster
- LOB or LOB partition
- Index or index partition

Each nonpartitioned object and object partition is stored in its own segment. For example, if an index has five partitions, then five segments contain the index data.

User Segment Creation

By default, the database uses **deferred segment creation** to update only database metadata when creating tables and indexes. Starting in Oracle Database 11*g* Release 2 (11.2.0.2), the database also defers segment creation when creating partitions. When a user inserts the first row into a table or partition, the database creates segments for the table or partition, its LOB columns, and its indexes.

Deferred segment creation enables you to avoid using database resources unnecessarily. For example, installation of an application can create thousands of objects, consuming significant disk space. Many of these objects may never be used.

You can use the `DBMS_SPACE_ADMIN` package to manage segments for empty objects. Starting with Oracle Database 11*g* Release 2 (11.2.0.2), you can use this PL/SQL package to do the following:

- Manually materialize segments for empty tables or partitions that do not have segments created
- Remove segments from empty tables or partitions that currently have an empty segment allocated

To best illustrate the relationship between object creation and segment creation, assume that deferred segment creation is disabled. You create a table as follows:

```
CREATE TABLE test_table (my_column NUMBER);
```

As shown in Figure 12-18, the database creates one segment for the table.

Figure 12-18 Creation of a User Segment

- 表、 表分区、或表簇
- 大对象或大对象分区
- 索引或索引分区

每个未分区对象和对象分区存储在其自己的段中。例如，如果索引有五个分区，则有五个段包含该索引的数据。

创建用户段

默认情况下，数据库使用**延迟的段创建**，当创建表和索引时，仅更新数据库元数据。在 Oracle 数据库 11g 第 2 版 (11.2.0.2) 中，数据库在创建分区时，也延迟段创建。在用户向表或分区中插入第一行时，数据库才创建表或分区、 及其 LOB 列、和索引段等。

延迟的段创建使您能够避免不必要地使用数据库资源。例如，某个应用程序在安装过程中可能会创建成千上万个占用大量磁盘空间的对象。许多这些对象可能从未使用。

可以使用 DBMS_SPACE_ADMIN 包管理空对象的段。从 Oracle 数据库 11g 第 2 版 (11.2.0.2) 开始，你可以使用该 PL/SQL 包执行下列操作：

- 为没有创建段的空表或分区手动实例化段
- 从空表或当前分配了空段的分区中删除段

为了更好地说明对象创建和段创建之间的关系，假设延迟段创建已被禁用。你创建一个表，如下所示：

```
CREATE TABLE test_table (my_column NUMBER);
```

如图 12-18 所示，数据库为该表创建一个段。

图 12-18 创建用户段

Description of "Figure 12-18 Creation of a User Segment"

When you create a table with a primary key or unique key, Oracle Database automatically creates an index for this key. Again assume that deferred segment creation is disabled. You create a table as follows:

```
CREATE TABLE lob_table (my_column NUMBER PRIMARY
KEY, clob_column CLOB);
```

Figure 12-19 shows that the data for `lob_table` is stored in one segment, while the implicitly created index is in a different segment. Also, the CLOB data is stored in its own segment, as is its associated CLOB index (see "Internal LOBs"). Thus, the `CREATE TABLE` statement results in the creation of *four* different segments.

Figure 12-19 Multiple Segments

Description of "Figure 12-19 Multiple Segments"

Note:
The segments of a table and the index for this table do not have to occupy the same tablespace.
The database allocates one or more extents when a segment is created. Storage parameters for the object determine how the extents for each segment are allocated (see "Storage Parameters for Extents"). The parameters affect the efficiency of data retrieval and storage for the data segment associated with the object.

See Also:

Description of "Figure 12-18 Creation of a User Segment"

当创建具有主键或唯一键的表时， Oracle 数据库将自动为此键创建一个索引。再假设延迟的段创建被禁用。你创建一个表，如下所示：

```
CREATE TABLE lob_table (my_column NUMBER PRIMARY
KEY, clob_column CLOB);
```

图 12-19 显示 lob_table 表的数据存储在一个段中，而隐式创建的索引在另一个不同的段中。此外，CLOB 数据及其关联的 CLOB 索引也分别存储在其自己的段中，（请参见"内部 LOBs"）。因此，这个 CREATE TABLE 语句会创建四个不同的段。

图 12-19 多个网段

Description of "Figure 12-19 Multiple Segments"

注意：
表段与其索引段不必位于相同的表空间中。

当创建一个段时，数据库将分配一个或多个扩展区。由对象的存储参数确定每个段的扩展区如何分配（请参见"扩展区的存储参数"）。这些参数影响数据检索的效率，和与该对象关联的数据段的存储。

另见：

- *Oracle Database Administrator's Guide* to learn how to manage deferred segment creation
- *Oracle Database Advanced Replication* for information on materialized views and materialized view logs
- *Oracle Database SQL Language Reference* for `CREATE TABLE` syntax

Temporary Segments

When processing a query, Oracle Database often requires temporary workspace for intermediate stages of SQL statement execution. Typical operations that may require a temporary segment include sorting, hashing, and merging bitmaps. While creating an index, Oracle Database also places index segments into temporary segments and then converts them into permanent segments when the index is complete.

Oracle Database does not create a temporary segment if an operation can be performed in memory. However, if memory use is not possible, then the database automatically allocates a temporary segment on disk.

Allocation of Temporary Segments for Queries

Oracle Database allocates temporary segments for queries as needed during a user session and drops them when the query completes. Changes to temporary segments are not recorded in the online redo log, except for space management operations on the temporary segment (see "Overview of the Online Redo Log").

The database creates temporary segments in the temporary tablespace assigned to the user. The default storage characteristics of the tablespace determine the characteristics of the extents in the temporary segment. Because allocation and deallocation of temporary segments occurs frequently, the best practice is to create at least one special tablespace for temporary segments. The database distributes I/O across disks and avoids fragmenting `SYSTEM` and other tablespaces with temporary segments.

Note:

When `SYSTEM` is locally managed, you must define a default temporary tablespace at database creation. A locally managed `SYSTEM` tablespace cannot be used for default temporary storage.

See Also:

- *Oracle Database Administrator's Guide* to learn how to create temporary tablespaces
- *Oracle Database SQL Language Reference* for `CREATE TEMPORARY TABLESPACE` syntax and semantics

- 《*Oracle 数据库管理员指南*》了解如何管理延迟段创建
- 《*Oracle 数据库高级复制*》关于物化视图和物化视图日志的信息
- 《*Oracle 数据库 SQL 语言参考*》关于 `CREATE TABLE` 的语法

临时段

当处理查询时，Oracle 数据库通常在 SQL 语句执行的中间阶段需要临时工作区。可能需要临时段的典型操作包括排序、 哈希、和合并位图。当创建索引时，Oracle 数据库将索引段置于临时段，然后当索引创建完成时将其转换成永久段。

如果一个操作可以在内存中执行，则 Oracle 数据库不会创建一个临时段。不过，如果不能使用内存，则数据库自动在磁盘上为其分配临时段。

为查询分配临时段

Oracle 数据库在用户会话期间，根据需要为查询分配临时段，并在查询完成后将其删除。除了会记录在临时段上的空间管理操作，临时段的更改不会记录在联机重做日志中（请参阅"联机重做日志概述"）。

数据库在分配给用户的临时表空间中创建临时段。表空间的默认存储特性决定了临时段中的扩展区特征。由于分配和释放临时段会频繁发生，最佳做法就是为临时段创建至少一个特别的表空间。数据库将 I/O 分布到多个磁盘，并避免在 `SYSTEM` 和其他表空间使用临时段，以防碎片化。

注意：

当 `SYSTEM` 为本地管理表空间时，您必须在数据库创建时定义默认临时表空间。本地管理的 `SYSTEM` 表空间不能用于默认临时存储。

另见：

- 《*Oracle 数据库管理员指南*》了解如何创建临时表空间
- 《*Oracle 数据库 SQL 语言参考*》关于 `CREATE TEMPORARY`

TABLESPACE 的语法和语义

Allocation of Temporary Segments for Temporary Tables and Indexes

Oracle Database can also allocate temporary segments for temporary tables and their indexes. Temporary tables hold data that exists only for the duration of a transaction or session. Each session accesses only the extents allocated for the session and cannot access extents allocated for other sessions.

Oracle Database allocates segments for a temporary table when the first INSERT into that table occurs. The insertion can occur explicitly or because of CREATE TABLE AS SELECT. The first INSERT into a temporary table allocates the segments for the table and its indexes, creates the root page for the indexes, and allocates any LOB segments. Segments for a temporary table are allocated in a temporary tablespace of the current user. Assume that the temporary tablespace assigned to user1 is temp1 and the temporary tablespace assigned to user2 is temp2. In this case, user1 stores temporary data in the temp1 segments, while user2 stores temporary data in the temp2 segments.

See Also:

- "Temporary Tables"
- *Oracle Database Administrator's Guide* to learn how to create temporary tables

Undo Segments

Oracle Database maintains records of the actions of transactions, collectively known as undo data. Oracle Database uses undo to do the following:

- Roll back an active transaction
- Recover a terminated transaction
- Provide read consistency
- Perform some logical flashback operations

Oracle Database stores undo data inside the database rather than in external logs. Undo data is stored in blocks that are updated just like data blocks, with changes to these blocks generating redo. In this way, Oracle Database can efficiently access undo data without needing to read external logs.

为临时表及其索引分配临时段

Oracle 数据库也可以为临时表及其索引分配临时段。临时表仅保存一个事务或会话的持续期间的数据。每个会话只能访问分配给该会话的扩展区，而不能访问分配给其他会话扩展区。

Oracle 数据库在第一次插入表时，会为临时表分配段。插入操作可能会显式发生，或者在运行 CREATE TABLE AS SELECT 时发生。在向一个临时表第一次插入数据时，会为表及其索引分配临时段、为索引创建根页、并分配 LOB 段。

临时表的段在当前用户的临时表空间中分配。假定为 user1 指派的临时表空间是 temp1， 分配给 user2 的临时表空间是 temp2。在这种情况下，user1 将临时数据存储在 temp1 内的段中，而 user2 将其临时数据存储在 temp2 内的段中。

另见：

- "临时表"
- 《*Oracle 数据库管理员指南*》了解如何创建临时表

撤消段

Oracle 数据库维护事务的操作记录，统称为撤消数据。Oracle 数据库使用撤消来执行下列操作：

- 回滚一个活动事务
- 恢复一个已终止的事务
- 提供读取一致性
- 执行一些逻辑闪回操作

Oracle 数据库将撤消数据存储在数据库内，而不是在外部日志中。存储撤消数据的块与数据块类似，对这些块的更改将生成重做数据。通过这种方式， Oracle 数据库可以高效地访问撤消数据，而无需读取外部日志。

Undo data is stored in an undo tablespace. Oracle Database provides a fully automated mechanism, known as automatic undo management mode, for managing undo segments and space in an undo tablespace.

Undo Segments and Transactions

When a transaction starts, the database binds (assigns) the transaction to an undo segment, and therefore to a transaction table, in the current undo tablespace. In rare circumstances, if the database instance does not have a designated undo tablespace, then the transaction binds to the system undo segment.

Multiple active transactions can write concurrently to the same undo segment or to different segments. For example, transactions T1 and T2 can both write to undo segment U1, or T1 can write to U1 while T2 writes to undo segment U2.

Conceptually, the extents in an undo segment form a ring. Transactions write to one undo extent, and then to the next extent in the ring, and so on in cyclical fashion. Figure 12-20 shows two transactions, T1 and T2, which begin writing in the third extent (E3) of an undo segment and continue writing to the fourth extent (E4).

Figure 12-20 Ring of Allocated Extents in an Undo Segment

Description of "Figure 12-20 Ring of Allocated Extents in an Undo Segment"

At any given time, a transaction writes sequentially to only one extent in an undo segment, known as the **current extent** for the transaction. Multiple active transactions can write simultaneously to the same current

撤消数据存储在一个撤消表空间中。Oracle 数据库提供了一个称为自动撤消管理模式的完全自动化机制，来管理撤消表空间中的撤消段和空间。

撤消段和事务

当事务启动时，数据库将此事务绑定到 （分配给）一个撤消段，并因此也绑定到当前撤消表空间中的一个事务表。在极少数的情况下，如果数据库实例没有一个指定的撤消表空间，则该事务被绑定到系统撤消段。

多个活动事务可以同时写入同一个撤消段或不同的撤消段。例如，事务 T1 和 T2 可以同时写入撤消段 U1，或 T1 写入 U1 而 T2 写入 U2。

从概念上讲，一个撤消段中的扩展区形成一个环。事务先写到一个撤消扩展区，然后写到环中的下一个扩展区，如此等等，循环往复。图 12-20 显示两个事务 T1 和 T2，从一个撤消段的第三个扩展区开始写入，然后继续往第四个扩展区 (E4）写入。

图 12-20 撤消段中的扩展区分配环

Description of "Figure 12-20 Ring of Allocated Extents in an Undo Segment"

在任何给定时间，一个事务仅在一个撤消段上的某个扩展区上顺序写入，该扩展区称为事务的**当前扩展区**。多个活动事务可以同时写入同一个当前扩展

extent or to different current extents. Figure 12-20 shows transactions T1 and T2 writing simultaneously to extent E3. Within an undo extent, a data block contains data for only one transaction.

As the current undo extent fills, the first transaction needing space checks the availability of the next allocated extent in the ring. If the next extent does *not* contain data from an active transaction, then this extent becomes the current extent. Now all transactions that need space can write to the new current extent. In Figure 12-21, when E4 is full, T1 and T2 continue writing to E1, overwriting the nonactive undo data in E1.

Figure 12-21 Cyclical Use of Allocated Extents in an Undo Segment

Description of "Figure 12-21 Cyclical Use of Allocated Extents in an Undo Segment"

If the next extent *does* contain data from an active transaction, then the database must allocate a new extent. Figure 12-22 shows a scenario in which T1 and T2 are writing to E4. When E4 fills up, the transactions cannot continue writing to E1 because E1 contains active undo entries.

区或不同的当前扩展区。图 12-20 显示事务 T1 和 T2 同时写入扩展区 E3。在一个撤消扩展区内某个数据块仅包含一个事务的数据。

当前撤消扩展区被填充后，第一个需要空间的事务将检查环中分配的下一个扩展区的可用性。如果下一个扩展区*不*包含活动事务数据，则该扩展区将成为当前扩展区。现在，需要空间的所有事务都可以写入新的当前扩展区内。在图 12-21 中， 当 E4 填满时，T1 和 T2 继续写入 E1，覆盖在 E1 中的非活动撤消数据。

图 12-21 撤消段中已分配扩展区的周期性使用

Description of "Figure 12-21 Cyclical Use of Allocated Extents in an Undo Segment"

如果下一个扩展区确实包含某个活动事务的数据，则数据库必须分配一个新的扩展区。图 12-22 显示 T1 和 T2 同时写入 E4 的场景。当 E4 填满时，这两个事务无法继续写入 E1，因为 E1 包含活动的撤消条目。因此，数据库为该撤消段分配新的扩展区 (E5)。事务继续写入 E5。

Therefore, the database allocates a new extent (E5) for this undo segment. The transactions continue writing to E5.

Figure 12-22 Allocation of a New Extent for an Undo Segment

Description of "Figure 12-22 Allocation of a New Extent for an Undo Segment"

See Also:

Oracle Database 2 Day DBA and *Oracle Database Administrator's Guide* to learn how to manage undo segments

Transaction Rollback

When a ROLLBACK statement is issued, the database uses undo records to roll back changes made to the database by the uncommitted transaction. During recovery, the database rolls back any uncommitted changes applied from the online redo log to the data files. Undo records provide read consistency by maintaining the before image of the data for users accessing data at the same time that another user is changing it.

Segment Space and the High Water Mark

To manage space, Oracle Database tracks the state of blocks in the segment. The **high water mark (HWM)** is the point in a segment beyond which data blocks are unformatted and have never been used.

图 12-22 为撤销段分配新的扩展区

Description of "Figure 12-22 Allocation of a New Extent for an Undo Segment"

另见：

《*Oracle 数据库 2 日 DBA*》 和 《*Oracle 数据库管理员指南*》了解如何管理撤消段

事务回滚

当发出了一个 ROLLBACK 语句时，数据库将使用撤消记录，来回滚未提交事务对数据库所做的更改。在恢复期间，数据库将回滚从联机重做日志应用到数据文件的任何未提交更改。当数据正在被其它用户更改时，撤消记录通过维护数据前像，为在同一时间访问数据的用户提供读取一致性。

段空间和高水位标记

Oracle 数据库通过跟踪段中的块状态来管理空间。**高水位标记 (HWM)** 是段中的一个点，超过该点的数据块是未格式化和未使用过的。

MSSM uses free lists to manage segment space. At table creation, no blocks in the segment are formatted. When a session first inserts rows into the table, the database searches the free list for usable blocks. If the database finds no usable blocks, then it preformats a group of blocks, places them on the free list, and begins inserting data into the blocks. In MSSM, a full table scan reads *all* blocks below the HWM.

ASSM does not use free lists and so must manage space differently. When a session first inserts data into a table, the database formats a single bitmap block instead of preformatting a group of blocks as in MSSM. The bitmap tracks the state of blocks in the segment, taking the place of the free list. The database uses the bitmap to find free blocks and then formats each block before filling it with data. ASSM spread out inserts among blocks to avoid concurrency issues.

Every data block in an ASSM segment is in one of the following states:

- Above the HWM

  These blocks are unformatted and have never been used.

- Below the HWM

  These blocks are in one of the following states:

  - Allocated, but currently unformatted and unused
  - Formatted and contain data
  - Formatted and empty because the data was deleted

Figure 12-23 depicts an ASSM segment as a horizontal series of blocks. At table creation, the HWM is at the beginning of the segment on the left. Because no data has been inserted yet, all blocks in the segment are unformatted and never used.

Figure 12-23 HWM at Table Creation

MSSM 使用空闲列表来管理段空间。在创建表时，段中的块并未被格式化。当一个会话初次向表中插入行时，数据库将搜索空闲列表来查找可用的块。如果数据库未找到可用的块，则它会预格式化一组块，并将它们放置在空闲列表中，并开始将数据插入到块中。在 MSSM 中，全表扫描会读取 HWM 之下的*所有*块。

ASSM 不使用空闲列表，所以必须以不同的方式管理空间。当会话初次向表中插入数据时，数据库只格式化一个单一位图块，而不像在 MSSM 中那样预格式化一组块。位图取代了空闲列表，用于跟踪在段中的块的状态。数据库使用位图来查找可用的块，然后在往块写入数据之前将其格式化。ASSM 将插入操作分散到多个块，以避免并发问题。

在一个 ASSM 段中的每个数据块处于以下状态之一：

- 在 HWM 之上

  这些块是未格式化的，且从未使用过。

- 在 HWM 之下

  这些块处于以下状态之一：

  - 已分配，但当前未格式化且未使用
  - 已格式化且包含数据
  - 已格式化且为空，因为数据已被删除

图 12-23 将一个 ASSM 段描述为一系列水平的块。在创建表时，HWM 在左侧段的开头。因为还未插入数据，段中的所有块都还是未格式化且从未使用过。

图将 12-23 在创建表时的 HWM

Description of "Figure 12-23 HWM at Table Creation"

Suppose that a transaction inserts rows into the segment. The database must allocate a group of blocks to hold the rows. The allocated blocks fall below the HWM. The database formats a bitmap block in this group to hold the metadata, but does not preformat the remaining blocks in the group.

In Figure 12-24, the blocks below the HWM are allocated, whereas blocks above the HWM are neither allocated or formatted. As inserts occur, the database can write to any block with available space. The **low high water mark (low HWM)** marks the point below which all blocks are known to be formatted because they either contain data or formerly contained data.

Figure 12-24 HWM and Low HWM

Description of "Figure 12-24 HWM and Low HWM"

Description of "Figure 12-23 HWM at Table Creation"

假设一个事务将行插入到段中。数据库必须分配一组块来容纳这些行。已分配的块在 HWM 之下。数据库格式化该组中的一个位图块来容纳元数据，但不会预格式化组中其余的块。

在图 12-24 中， HWM 之下的块是已分配的，而 HWM 之上的块是既未分配也未格式化的。插入发生时，数据库可以写入到任何具有可用空间的块。由**低位高水位标记** (低 HWM) 标记一个点，该点之下的所有块都已知是已格式化的，要么包含数据，或以前曾包含数据。

图 12-24 HWM 和低 HWM

Description of "Figure 12-24 HWM and Low HWM"

In Figure 12-25, the database chooses a block between the HWM and low HWM and writes to it. The database could have just as easily chosen any other block between the HWM and low HWM, or any block below the low HWM that had available space. In Figure 12-25, the blocks to either side of the newly filled block are unformatted.

Figure 12-25 HWM and Low HWM

Description of "Figure 12-25 HWM and Low HWM"

The low HWM is important in a full table scan. Because blocks below the HWM are formatted only when used, some blocks could be unformatted, as in Figure 12-25. For this reason, the database reads the bitmap block to obtain the location of the low HWM. The database reads all blocks up to the low HWM because they are known to be formatted, and then carefully reads only the formatted blocks between the low HWM and the HWM. Assume that a new transaction inserts rows into the table, but the bitmap indicates that insufficient free space exists under the HWM. In Figure 12-26, the database advances the HWM to the right, allocating a new group of unformatted blocks.

Figure 12-26 Advancing HWM and Low HWM

在图 12-25 中，数据库选定了 HWM 和低 HWM 之间的一个块，并往其中写入数据。数据库也可能会随意选择 HWM 和低 HWM 之间的任何其他块，或低 HWM 之下任何有可用空间的块。图 12-25 中，在新填充块两边的块都还是未格式化的。

图 12-25 HWM 和低 HWM

Description of "Figure 12-25 HWM and Low HWM"

低 HWM 在全表扫描中非常重要。因为 HWM 之下的块只在被使用时才格式化，所以可能还有一些块是未被格式化的，如图 12-25 所示。鉴于此，数据库读取位图块，以获取低 HWM 的位置。数据库读取低 HWM 之下的所有块，因为已知它们是已格式化的，然后仅仔细读取位于低 HWM 和 HWM 之间的已格式化块。

假定一个新事务将行插入到该表，但位图指示在 HWM 之下没有足够的可用空间。在图 12-26 中， 数据库向右推进 HWM， 分配一组新的未格式化块。

图 12-26 推进 HWM 和低 HWM

Description of "Figure 12-26 Advancing HWM and Low HWM"

When the blocks between the HWM and low HWM are full, the HWM advances to the right and the low HWM advances to the location of the old HWM. As the database inserts data over time, the HWM continues to advance to the right, with the low HWM always trailing behind it. Unless you manually rebuild, truncate, or shrink the object, the HWM never retreats.

See Also:

- *Oracle Database Administrator's Guide* to learn how to shrink segments online
- *Oracle Database SQL Language Reference* for `TRUNCATE TABLE` syntax and semantics

Overview of Tablespaces

A **tablespace** is a logical storage container for segments. Segments are database objects, such as tables and indexes, that consume storage space. At the physical level, a tablespace stores data in one or more data files or temp files.

A database must have the `SYSTEM` and `SYSAUX` tablespaces. Figure 12-27 shows the tablespaces in a typical database. The following sections describe the tablespace types.

Figure 12-27 Tablespaces

Description of "Figure 12-26 Advancing HWM and Low HWM"

当 HWM 与低 HWM 之间的块填满时，HWM 向右推进，而低 HWM 相应推进到旧的 HWM 的位置。数据库不断插入数据，随着时间的推移，HWM 继续向右推进，而低 HWM 总尾随其后。除非您手动重建、 截断、或缩小该对象，否则 HWM 从不倒退。

另见：

- 《*Oracle 数据库管理员指南*》了解如何在线收缩段
- 《*Oracle 数据库 SQL 语言参考*》关于 `TRUNCATE TABLE` 的语法和语义

表空间概述

表空间是段的逻辑存储容器。段是占用存储空间的数据库对象，如表和索引。在物理级别，表空间将数据存储在一个或多个数据文件或临时文件中。

数据库必须有 `SYSTEM` 和 `SYSAUX` 表空间。图 12-27 显示了一个典型的数据库中的表空间。以下各节描述表空间类型。

图 12-27 表空间

Description of "Figure 12-27 Tablespaces"

Permanent Tablespaces

A **permanent tablespace** groups persistent schema objects. The segments for objects in the tablespace are stored physically in data files.

Each database user is assigned a default permanent tablespace. A very small database may need only the default SYSTEM and SYSAUX tablespaces. However, Oracle recommends that you create at least one tablespace to store user and application data. You can use tablespaces to achieve the following goals:

- Control disk space allocation for database data
- Assign a **quota** (space allowance or limit) to a database user
- Take individual tablespaces online or offline without affecting the availability of the whole database
- Perform backup and recovery of individual tablespaces
- Import or export application data by using the Oracle Data Pump utility (see "Oracle Data Pump Export and Import")
- Create a **transportable tablespace** that you can copy or move from one database to another, even across platforms

Description of "Figure 12-27 Tablespaces"

永久表空间

永久表空间对持久性模式对象进行分组。表空间中的对象的段物理上存储在数据文件中。

每个数据库用户被分配了一个默认的永久表空间。一个非常小的数据库可能只需要默认的 SYSTEM 和 SYSAUX 表空间。Oracle 建议您创建至少一个表空间来存储用户和应用程序数据。可以使用表空间实现以下目标：

- 控制数据库数据的磁盘空间分配
- 为数据库用户分配**配额** （空间限额或限制）
- 将单个表空间联机或脱机，而不影响整个数据库的可用性
- 执行单个表空间的备份和恢复
- 使用 Oracle 数据泵实用程序导入或导出应用程序数据（请参见 "Oracle 数据泵导入和导出"）
- 通过创建一个**可传输表空间**，可以将其从一个数据库复制或移动到另一个数据库，甚至跨平台

Moving data by transporting tablespaces can be orders of magnitude faster than either export/import or unload/load of the same data, because transporting a tablespace involves only copying data files and integrating the tablespace metadata. When you transport tablespaces you can also move index data.

See Also:

- *Oracle Database Administrator's Guide* to learn how to transport tablespaces
- *Oracle Database Utilities* to learn about Oracle Data Pump
- *Oracle Streams Concepts and Administration* for more information on ways to copy or transport files

The SYSTEM Tablespace

The SYSTEM tablespace is a necessary administrative tablespace included with the database when it is created. Oracle Database uses SYSTEM to manage the database.

The SYSTEM tablespace includes the following information, all owned by the SYS user:

- The data dictionary
- Tables and views that contain administrative information about the database
- Compiled stored objects such as triggers, procedures, and packages

The SYSTEM tablespace is managed as any other tablespace, but requires a higher level of privilege and is restricted in some ways. For example, you cannot rename or drop the SYSTEM tablespace.

By default, Oracle Database sets all newly created user tablespaces to be locally managed. In a database with a locally managed SYSTEM tablespace, you cannot create dictionary-managed tablespaces (which are deprecated). However, if you execute the CREATE DATABASE statement manually and accept the defaults, then the SYSTEM tablespace is dictionary managed. You can migrate an existing dictionary-managed SYSTEM tablespace to a locally managed format.

Note:

Oracle strongly recommends that you use Database Configuration Assistant (DBCA) to create new databases so that all tablespaces,

通过传输表空间来移动数据，可能比导出/导入或卸载/装载相同的数据要快几个数量级，因为传输表空间只涉及复制数据文件并集成表空间元数据。当您传输表空间时，还可以移动索引数据。

另见：

- 《*Oracle 数据库管理员指南*》了解如何传输表空间
- 《*Oracle 数据库实用程序*》了解 Oracle 数据泵
- 《*Oracle 流概念和管理*》关于复制或传输文件的方法的详细信息

SYSTEM 表空间

SYSTEM 表空间是在数据库被创建时就包括的一个必要的管理性表空间。Oracle 数据库使用 SYSTEM 来管理数据库。

SYSTEM 表空间包括以下信息，全部都由 SYS 用户所拥有

- 数据字典
- 包含数据库管理信息的表和视图
- 已编译的存储对象，如触发器、 过程、和包等

SYSTEM 表空间的管理与其他表空间几乎一样，但需要较高的特权，且在某些方面有所限制。例如，您不能重命名或删除 SYSTEM 表空间。

默认情况下，Oracle 数据库将所有新创建的用户表空间设置为本地管理表空间。在 SYSTEM 为本地管理表空间的数据库中，不能创建字典管理表空间 （这已经过时）。但是，如果手动执行 CREATE DATABASE 语句并接受默认值，则 SYSTEM 表空间是字典管理的。您可以将一个现有字典管理 SYSTEM 表空间迁移到本地管理格式。

注意：

Oracle 强烈建议您使用数据库配置助手 (DBCA)创建新数据库，以便使包括 SYSTEM 在内的所有表空间都缺省为本地管理表空间。

including SYSTEM, are locally managed by default.

See Also:

- "Online and Offline Tablespaces" for information about the permanent online condition of the SYSTEM tablespace
- "Tools for Database Installation and Configuration" to learn about DBCA
- *Oracle Database Administrator's Guide* to learn how to create or migrate to a locally managed SYSTEM tablespace
- *Oracle Database SQL Language Reference* for CREATE DATABASE syntax and semantics

The SYSAUX Tablespace

The SYSAUX tablespace is an auxiliary tablespace to the SYSTEM tablespace. The SYSAUX tablespace provides a centralized location for database metadata that does not reside in the SYSTEM tablespace. It reduces the number of tablespaces created by default, both in the seed database and in user-defined databases.

Several database components, including Oracle Enterprise Manager and Oracle Streams, use the SYSAUX tablespace as their default storage location. Therefore, the SYSAUX tablespace is created automatically during database creation or upgrade.

During normal database operation, the database does not allow the SYSAUX tablespace to be dropped or renamed. If the SYSAUX tablespace becomes unavailable, then core database functionality remains operational. The database features that use the SYSAUX tablespace could fail, or function with limited capability.

See Also:

Oracle Database Administrator's Guide to learn about the SYSAUX tablespace

Undo Tablespaces

An **undo tablespace** is a locally managed tablespace reserved for system-managed undo data (see "Undo Segments"). Like other permanent tablespaces, undo tablespaces contain data files. Undo blocks in these files are grouped in extents.

Automatic Undo Management Mode

Undo tablespaces require the database to be in the default automatic

另见：

- "联机和脱机表空间"有关 SYSTEM 表空间永久在线状态的信息
- "数据库安装和配置"了解 DBCA
- 《*Oracle 数据库管理员指南*》了解如何创建或迁移到本地管理的 SYSTEM 表空间
- 《*Oracle 数据库 SQL 语言参考*》有关 CREATE DATABASE 的语法和语义

SYSAUX 表空间

SYSAUX 表空间是 SYSTEM 表空间的一个辅助表空间。SYSAUX 表空间为未驻留在 SYSTEM 表空间的数据库元数据提供了一个集中的位置。它减少了在种子数据库和用户定义的数据库中缺省创建的表空间数量。

包括 Oracle 企业管理器和 Oracle 流在内的几个数据库组件将 SYSAUX 表空间作为其默认存储位置。因此，SYSAUX 表空间在数据库创建或升级过程中自动创建。

在正常的数据库操作中，数据库不允许删除或重命名 SYSAUX 表空间。如果 SYSAUX 表空间变得不可用，则数据库的核心功能仍是可用的。使用 SYSAUX 表空间的数据库特性可能会失败，或只能发挥有限功能。

另见：

《*Oracle 数据库管理员指南*》了解 SYSAUX 表空间

表空间

撤消表空间是为系统管理的撤消数据（见"撤消段"）而保留的一个本地管理表空间。与其他永久表空间类似，撤消表空间也包含数据文件。在这些文件中的撤消块被分组为一些扩展区。

自动撤消管理模式

撤消表空间要求数据库处于默认的自动撤消管理模式。此模式消除了手动管

undo management mode. This mode eliminates the complexities of manually administering undo segments. The database automatically tunes itself to provide the best possible retention of undo data to satisfy long-running queries that may require this data.

An undo tablespace is automatically created with a new installation of Oracle Database. Earlier versions of Oracle Database may not include an undo tablespace and use legacy rollback segments instead, known as **manual undo management mode**. When upgrading to Oracle Database 11*g*, you can enable automatic undo management mode and create an undo tablespace. Oracle Database contains an Undo Advisor that provides advice on and helps automate your undo environment.

A database can contain multiple undo tablespaces, but only one can be in use at a time. When an instance attempts to open a database, Oracle Database automatically selects the first available undo tablespace. If no undo tablespace is available, then the instance starts without an undo tablespace and stores undo data in the `SYSTEM` tablespace. Storing undo data in `SYSTEM` is not recommended.

See Also:

- *Oracle Database Administrator's Guide* to learn about automatic undo management
- *Oracle Database Upgrade Guide* to learn how to migrate to automatic undo management mode
- *Oracle Database 2 Day DBA* for information on the Undo Advisor and on how to use advisors

Automatic Undo Retention

The **undo retention period** is the minimum amount of time that Oracle Database attempts to retain old undo data before overwriting it. Undo retention is important because long-running queries may require older block images to supply read consistency. Also, some Oracle Flashback features can depend on undo availability.

In general, it is desirable to retain old undo data as long as possible. After a transaction commits, undo data is no longer needed for rollback or transaction recovery. The database can retain old undo data if the undo tablespace has space for new transactions. When available space is low, the database begins to overwrite old undo data for committed transactions.

Oracle Database automatically provides the best possible undo retention

理撤消段的复杂性。数据库自动对其进行优化，提供撤消数据最佳可能的保留期，以满足可能需要此数据的长时间查询。

新安装的 Oracle 数据库会自动创建一个撤消表空间。早期版本的 Oracle 数据库可能不包含撤消表空间，而是使用在**手动撤消管理模式**下的传统的回滚段。在升级到 Oracle 数据库 11 g 时，您可以启用自动撤消管理模式，并创建一个撤消表空间。Oracle 数据库包含一个撤消顾问，可提供有助于你自动化您的撤消环境的建议。

数据库可以包含多个撤消表空间，但一次只能使用其中的某一个。当实例尝试打开数据库时，Oracle 数据库将自动选择第一个可用的撤消表空间。若没有撤消表空间可用，则实例以不带撤消表空间的方式启动，并将撤消数据存储在 `SYSTEM` 表空间中。建议不要在 `SYSTEM` 表空间中存储撤消数据。

另见：

- 《*Oracle 数据库管理员指南*》了解自动撤销管理
- 《*Oracle 数据库升级指南*》了解如何迁移到自动撤消管理模式
- 《*Oracle 数据库 2 日 DBA*》 关于撤消顾问，以及如何使用顾问的信息

自动撤销保留

撤消保留期是 Oracle 数据库在覆盖旧的撤消数据之前尝试对其保留的最小时间量。撤消保留很重要，因为长时间运行的查询可能需要旧数据块前像以提供读取一致性。此外，某些 Oracle 闪回功能也可能会依赖于撤消数据的可用性。

通常，应该尽可能保留旧的撤消数据。事务提交后，撤消数据对于回滚或恢复事务不再是必需的。如果撤消表空间还有足够空间用于新事务，数据库可能会保留旧的撤消数据。当可用空间偏低时，数据库开始覆盖已提交事务的旧撤消数据。

Oracle 数据库自动为当前的撤消表空间提供最佳可能的撤消保留。数据库

for the current undo tablespace. The database collects usage statistics and tunes the retention period based on these statistics and the undo tablespace size. If the undo tablespace is configured with the AUTOEXTEND option, and if the maximum size is not specified, then undo retention tuning is different. In this case, the database tunes the undo retention period to be slightly longer than the longest-running query, if space allows.

See Also:

Oracle Database Administrator's Guide for more details on automatic tuning of undo retention

Temporary Tablespaces

A **temporary tablespace** contains transient data that persists only for the duration of a session. No permanent schema objects can reside in a temporary tablespace. The database stores temporary tablespace data in **temp files**.

Temporary tablespaces can improve the concurrency of multiple sort operations that do not fit in memory. These tablespaces also improve the efficiency of space management operations during sorts.

When the SYSTEM tablespace is locally managed, a default temporary tablespace is included in the database by default during database creation. A locally managed SYSTEM tablespace cannot serve as default temporary storage.

Note:

You cannot make a default temporary tablespace permanent.

You can specify a user-named default temporary tablespace when you create a database by using the DEFAULT TEMPORARY TABLESPACE extension to the CREATE DATABASE statement. If SYSTEM is dictionary managed, and if a default temporary tablespace is not defined at database creation, then SYSTEM is the default temporary storage. However, the database writes a warning in the alert log saying that a default temporary tablespace is recommended.

See Also:

- "Permanent and Temporary Data Files"
- *Oracle Database Administrator's Guide* to learn how to create a default temporary tablespace
- *Oracle Database SQL Language Reference* for the syntax of the

会收集使用情况统计信息，并基于这些统计信息和撤消表空间大小来调整保留期。如果撤消表空间配置了 AUTOEXTEND 选项，而且也未指定最大大小，则撤消保留调整会有所不同。在这种情况下，如果空间允许，数据库将撤消保留期调整为比最长运行的查询稍长。

另见：

《*Oracle 数据库管理员指南*》有关自动调整的撤消保留的更多详细信息

临时表空间

临时表空间仅包含在会话期间存在的瞬态数据。永久模式对象不能驻留在临时表空间中。数据库临时表空间数据存储在**临时文件**中。

临时表空间可以提高不适合在内存中进行的多个排序操作的并发性。这些表空间也提高了在排序过程中空间管理的效率。

当 SYSTEM 表空间为本地管理时，在数据库创建期间，该数据库中会缺省包含一个默认的临时表空间。本地管理的 SYSTEM 表空间不能作为默认临时存储。

注意：

你不能使一个默认临时表空间永久化。

当你创建一个数据库时，你可以通过在 CREATE DATABASE 语句中使用 DEFAULT TEMPORARY TABLESPACE 扩展选项，来指定一个用户命名的默认临时表空间。如果 SYSTEM 是字典管理的，且在创建数据库时未定义一个默认临时表空间，则 SYSTEM 为其默认临时存储。不过，数据库会在警报日志中写入一条警告，说建议创建一个默认的临时表空间。

另见：

- "永久和临时数据文件"
- 《*Oracle 数据库管理员指南*》了解如何创建一个默认临时表空间
- 《*Oracle 数据库 SQL 语言参考*》关于 CREATE DATABASE 和

DEFAULT TEMPORARY TABLESPACE clause of CREATE DATABASE and ALTER DATABASE

Tablespace Modes

The **tablespace mode** determines the accessibility of the tablespace.

Read/Write and Read-Only Tablespaces

Every tablespace is in a **write mode** that specifies whether it can be written to. The mutually exclusive modes are as follows:

- Read/write mode

  Users can read and write to the tablespace. All tablespaces are initially created as read/write. The SYSTEM and SYSAUX tablespaces and temporary tablespaces are permanently read/write, which means that they cannot be made read-only.
- Read-only mode

  Write operations to the data files in the tablespace are prevented. A read-only tablespace can reside on read-only media such as DVDs or WORM drives.
  Read-only tablespaces eliminate the need to perform backup and recovery of large, static portions of a database. Read-only tablespaces do not change and thus do not require repeated backup. If you recover a database after a media failure, then you do not need to recover read-only tablespaces.

See Also:

- *Oracle Database Administrator's Guide* to learn how to change a tablespace to read only or read/write mode
- *Oracle Database SQL Language Reference* for ALTER TABLESPACE syntax and semantics
- *Oracle Database Backup and Recovery User's Guide* for more information about recovery

Online and Offline Tablespaces

A tablespace can be **online** (accessible) or **offline** (not accessible) whenever the database is open. A tablespace is usually online so that its data is available to users. The SYSTEM tablespace and temporary tablespaces cannot be taken offline.
A tablespace can go offline automatically or manually. For example, you

ALTER DATABASE 的 DEFAULT TEMPORARY TABLESPACE 子句的语法

表空间模式

由**表空间模式**确定表空间的可访问性。

读/写和只读表空间

每个表空间都处于某种**写入模式**，以指定是否可以对其进行写入操作。有两种互斥的模式，如下所示：

- 读/写模式

  用户可以读取和写入表空间。所有表空间最初都被创建为可读/写的。SYSTEM、 SYSAUX 表空间、和临时表空间总是可读/写的，也就是不能将它们置为只读。

- 只读模式

  对表空间中数据文件的写操作被禁止。只读表空间可以驻留在只读介质上，如 DVD 或 WORM 驱动器。

  只读表空间消除了对数据库中大型的静态部分执行备份和恢复的需要。只读表空间不会变更，因此不需要重复的备份。如果在介质故障后恢复一个数据库，则不需要恢复只读表空间。

另见：

- 《*Oracle 数据库管理员指南*》了解如何将表空间更改为只读或读/写模式
- 《*Oracle 数据库 SQL 语言参考*》关于 ALTER TABLESPACE 的语法和语义
- 《*Oracle 数据库备份和恢复用户指南*》关于恢复的详细信息

联机和脱机表空间

当数据库处于打开状态时，表空间可以处于**联机状态** （可访问） 或**脱机状态** （不能访问）。表空间通常是联机的，以便其数据可供用户使用。SYSTEM 表空间和临时表空间不能脱机。

表空间可以自动或手动脱机。例如，您可以将表空间脱机，以进行维护或备

can take a tablespace offline for maintenance or backup and recovery. The database automatically takes a tablespace offline when certain errors are encountered, as when the database writer (DBW*n*) process fails in several attempts to write to a data file. Users trying to access tables in an offline tablespace receive an error.

When a tablespace goes offline, the database does the following:

- The database does not permit subsequent DML statements to reference objects in the offline tablespace. An offline tablespace cannot be read or edited by any utility other than Oracle Database.
- Active transactions with completed statements that refer to data in that tablespace are not affected at the transaction level.
- The database saves undo data corresponding to those completed statements in a deferred undo segment in the SYSTEM tablespace. When the tablespace is brought online, the database applies the undo data to the tablespace, if needed.

See Also:

- "Online and Offline Data Files"
- "Database Writer Process (DBW***n***)"
- *Oracle Database Administrator's Guide* to learn how to alter tablespace availability

Tablespace File Size

A tablespace is either a **bigfile tablespace** or a **smallfile tablespace**. These tablespaces are indistinguishable in terms of execution of SQL statements that do not explicitly refer to data files or temp files. The difference is as follows:

- A smallfile tablespace can contain multiple data files or temp files, but the files cannot be as large as in a bigfile tablespace. This is the default tablespace type.
- A bigfile tablespace contains one very large data file or temp file. This type of tablespaces can do the following:
    - Increase the storage capacity of a database

      The maximum number of data files in a database is limited (usually to 64 KB files), so increasing the size of each data file increases the overall storage.

份与恢复。数据库在遇到某些错误时会自动将一个表空间脱机，比如数据库写进程 (DBWn)多次试图写入数据文件但遭遇失败时。当用户试图访问一个脱机表空间中的表时，将收到一个错误。

当一个表空间进入脱机状态时，数据库执行以下任务：

- 数据库不允许后续 DML 语句引用脱机表空间中的对象。脱机的表空间无法被除 Oracle 数据库之外的任何其它实用程序读取或编辑。
- 活动事务中的某些已完成的语句，曾引用了脱机表空间中的数据，这些事务在事务级别不受影响。
- 数据库将那些已完成语句相应的撤消数据，保存在 SYSTEM 表空间中的延迟撤消段中。当表空间被联机时，数据库在必要时将撤消数据应用到该表空间。

另见：

- "联机和脱机数据文件"
- "数据库写入进程（DBWn）"
- 《*Oracle 数据库管理员指南*》了解如何更改表空间的可用性

表空间文件大小

表空间可以是一个**大文件表空间**或**小文件表空间**。这些表空间的区别在于 SQL 语句的执行是否需要显式引用数据文件或临时文件。区别如下：

- 小文件表空间可以包含多个数据文件或临时文件，但其文件不能像大文件表空间中的文件那么大。这是默认的表空间类型。
- 大文件表空间仅包含一个非常大的数据文件或临时文件。这种类型的表空间可以：
    - 提高数据库的存储容量

      数据库中的数据文件的最大数量有限 （通常为 64 KB 个文件），所以增加每个数据文件的大小也就增加了总的存储量。

- Reduce the burden of managing many data files and temp files
  Bigfile tablespaces simplify file management with Oracle Managed Files and Automatic Storage Management (Oracle ASM) by eliminating the need for adding new files and dealing with multiple files.
- Perform operations on tablespaces rather than individual files
  Bigfile tablespaces make the tablespace the main unit of the disk space administration, backup and recovery, and so on.

- Bigfile tablespaces are supported only for locally managed tablespaces with ASSM. However, locally managed undo and temporary tablespaces can be bigfile tablespaces even when segments are manually managed.

See Also:

- "Backup and Recovery"
- *Oracle Database Administrator's Guide* to learn how to manage bigfile tablespaces

- 减轻管理过多数据文件和临时文件的负担

  大文件表空间使用 Oracle 管理的文件和自动存储管理 (Oracle ASM)，通过消除添加新文件和处理多个文件的需要，简化了文件管理。

- 在表空间上（而不是在单个文件上）执行操作

  大文件表空间将表空间作为磁盘空间管理、 备份和恢复等的主要单位。

- 大文件表空间只支持使用 ASSM 的本地管理表空间。然而，本地管理的撤消和临时表空间，即便在手动段管理模式下，仍然可以是大文件表空间。

另见：

- "备份和恢复"
- 《*Oracle 数据库管理员指南*》了解如何管理大文件表空间

13 Oracle Database Instance

This chapter explains the nature of an Oracle database instance, the parameter and diagnostic files associated with an instance, and what occurs during instance creation and the opening and closing of a database.

This chapter contains the following sections:

- Introduction to the Oracle Database Instance
  - Database Instance Structure
  - Database Instance Configurations
- Overview of Instance Startup and Shutdown
  - Overview of Instance and Database Startup
  - Overview of Database and Instance Shutdown
- Overview of Checkpoints
  - Purpose of Checkpoints
  - When Oracle Database Initiates Checkpoints
- Overview of Instance Recovery
  - Purpose of Instance Recovery
  - When Oracle Database Performs Instance Recovery
  - Importance of Checkpoints for Instance Recovery
  - Instance Recovery Phases
- Overview of Parameter Files
  - Initialization Parameters
  - Server Parameter Files
  - Text Initialization Parameter Files
  - Modification of Initialization Parameter Values
- Overview of Diagnostic Files
  - Automatic Diagnostic Repository
  - Alert Log
  - Trace Files

Introduction to the Oracle Database Instance

A database instance is a set of memory structures that manage database files. A database is a set of physical files on disk created by the `CREATE`

第 13 章 Oracle 数据库实例

本章说明了 Oracle 数据库实例的特征、与实例关联的参数和诊断文件，以及在实例创建，和数据库打开、关闭期间会发生什么。

本章包含以下各节：

- Oracle 数据库实例简介
  - Oracle 实例结构
  - 数据库实例配置
- 实例启动和关闭概述
  - 实例和数据库启动概述
  - 数据库和实例关闭概述
- 检查点概述
  - 检查点的目的
  - Oracle 数据库何时启动检查点
- 实例恢复概述
  - 实例恢复的目的
  - Oracle 数据库何时执行实例恢复
  - 实例恢复检查点的重要性
  - 实例恢复阶段
- 参数文件的概述
  - 初始化参数
  - 服务器参数文件
  - 文本初始化参数文件
  - 修改初始化参数值的值
- 诊断文件概述
  - 自动诊断存储库
  - 警报日志
  - 跟踪文件

Oracle 数据库实例简介

数据库实例是一组用于管理数据库文件的内存结构。数据库是一组由 `CREATE DATABASE` 语句在磁盘上创建的物理文件。由实例管理其关联

DATABASE statement. The instance manages its associated data and serves the users of the database.

Every running Oracle database is associated with at least one Oracle database instance. Because an instance exists in memory and a database exists on disk, an instance can exist without a database and a database can exist without an instance.

Database Instance Structure

When an instance is started, Oracle Database allocates a memory area called the system global area (SGA) and starts one or more background processes. The SGA serves various purposes, including the following:

- Maintaining internal data structures that are accessed by many processes and threads concurrently
- Caching data blocks read from disk
- Buffering redo data before writing it to the online redo log files
- Storing SQL execution plans

The SGA is shared by the Oracle processes, which include server processes and background processes, running on a single computer. The way in which Oracle processes are associated with the SGA varies according to operating system.

A database instance includes background processes. Server processes, and the process memory allocated in these processes, also exist in the instance. The instance continues to function when server processes terminate.

Figure 13-1 shows the main components of an Oracle database instance.

Figure 13-1 Database Instance

的数据，并为数据库用户提供服务。

每个正在运行的 Oracle 数据库至少与一个 Oracle 数据库实例相关联。因为实例存在于内存中，而数据库存在于磁盘上，所以实例可以在没有数据库时而存在，数据库也可以在没有实例时存在。

Oracle 实例结构

当实例启动时， Oracle 数据库分配一个叫做系统全局区（SGA） 的内存区域，并启动一个或多个后台进程。SGA 用于以下几个目的：

- 维护由很多进程和线程同时访问的内部数据结构
- 缓存从磁盘中读取的数据块
- 缓冲重做数据，然后再将其写入联机重做日志文件
- 存储 SQL 执行计划

SGA 由运行在一台单一计算机上的多个 Oracle 进程共享，包括服务器进程和后台进程。Oracle 进程与 SGA 相关联的方式会因不同的操作系统而有所不同。

数据库实例包括多个后台进程。服务器进程和在这些的进程中分配的进程内存也存在于实例中。当服务器进程终止时，实例仍会继续运行。

图 13-1 显示 Oracle 数据库实例的主要组件。

图 13-1 数据库实例

Description of "Figure 13-1 Database Instance"

See Also:

- "Overview of the System Global Area"
- "Overview of Background Processes"

Database Instance Configurations

You can run Oracle Database in either of the following mutually exclusive configurations:

- Single-instance configuration

  A one-to-one relationship exists between the database and an instance.
- Oracle Real Application Clusters (Oracle RAC) configuration

  A one-to-many relationship exists between the database and instances.

Figure 13-2 shows possible database instance configurations.

Description of "Figure 13-1 Database Instance"

另见：

- "系统全局区概述"
- "后台进程概述"

数据库实例配置

您可以在以下两种互斥配置中的一种来运行 Oracle 数据库：

- 单实例配置

  数据库和实例之间存在一对一关系。
- Oracle 真正应用集群 （Oracle RAC）配置

  数据库与实例之间存在一对多关系。

图 13-2 显示了这两种可能的数据库实例配置。

Figure 13-2 Database Instance Configurations

Description of "Figure 13-2 Database Instance Configurations"

Whether in a single-instance or Oracle RAC configuration, a database instance is associated with only one database at a time. You can start a database instance and **mount** (associate the instance with) one database, but not mount two databases simultaneously with the same instance.

Note:

This chapter discusses a single-instance database configuration unless otherwise noted.

Multiple instances can run concurrently on the same computer, each accessing its own database. For example, a computer can host two distinct databases: `prod1` and `prod2`. One database instance manages `prod1`, while a separate instance manages `prod2`.

See Also:

Oracle Real Application Clusters Administration and Deployment Guide for information specific to Oracle RAC

Duration of an Instance

An instance begins when it is created with the `STARTUP` command and ends when it is terminated. During this period, an instance can associate itself with one and only one database. Furthermore, the instance can mount a database only once, close it only once, and open it only once. After a database has been closed or shut down, you must start a *different* instance to mount and open this database.

图 13-2 数据库实例的配置

Description of "Figure 13-2 Database Instance Configurations"

无论是在单实例还是在 Oracle RAC 配置中，一个数据库实例在同一时刻只与一个数据库相关联。您可以启动一个实例，并装载（与实例相关联）一个数据库，但不能同时将两个数据库装载到同一实例。

注意：

除非另有说明，本章仅讨论单实例数据库配置。

在同一台计算机上可以同时运行多个实例，每个实例访问其自己的数据库。例如，一台计算机可以承载两个完全不同的数据库： prod1 和 prod2。一个数据库实例管理 prod1，而另一个实例管理 prod2。

另见：

《*Oracle 真正应用集群管理和部署指南*》关于 Oracle RAC 的特定信息

实例的持续时间

实例在使用 `STARTUP` 命令创建时开始，在终止时结束。这此期间，实例能且只能与一个数据库相关联。此外，该实例只能装载数据库一次，关闭数据库一次、打开数据库一次。在数据库已关闭后，您必须启动一个不同实例来装载并打开此数据库。

Table 13-1 illustrates a database instance attempting to reopen a database that it previously closed.

表 13-1 说明了一个数据库实例试图重新打开一个之前已关闭的数据库。

Table 13-1 Duration of an Instance

表 13-1 持续时间的实例

Statement	**Explanation**	**解释**
SQL> STARTUP ORACLE instance started. Total System Global Area 468729856 bytes Fixed Size 1333556 bytes Variable Size 440403660 bytes Database Buffers 16777216 bytes Redo Buffers 10215424 bytes Database mounted. Database opened.	The STARTUP command creates an instance, which mounts and opens the database.	STARTUP 命令创建一个实例，它会装载并打开数据库。
SQL> SELECT TO_CHAR(STARTUP_TIME,'MON-DD-RR HH24:MI:SS') AS "Inst Start Time" FROM V$INSTANCE; Inst Start Time ---------- JUN-18-09 **13:14:48**	This query shows the time that the current instance was started.	此查询显示当前实例的启动时间。
SQL> ALTER DATABASE CLOSE; Database altered.	The instance closes the database, leaving it in a mounted state. The instance can read and write to the control file but not the data files.	实例关闭数据库，将其置于已装载状态。实例可以读写控制文件，但不能读写数据文件。
SQL> ALTER DATABASE OPEN; ALTER DATABASE OPEN * ERROR at line 1: ORA-16196: database has been previously opened and closed	The instance attempts to reopen the database that it previously closed. Oracle Database issues an error because the same instance cannot open a database twice.	该实例尝试重新打开之前已关闭的数据库。Oracle 数据库发出一个错误，因为同一实例不能打开数据库两次
SQL> SHUTDOWN IMMEDIATE	At this stage, the only option for the instance is to shut down, ending the life of this instance.	在这一阶段，唯一选择的是关闭实例，结束此实例的生命周期。
SQL> STARTUP Oracle instance started.	The STARTUP command creates a new	STARTUP 命令创建一个新的实例，装载

. . .	instance and mounts and open the database.	并打开数据库。
SQL> SELECT TO_CHAR(STARTUP_TIME,'MON-DD-RR HH24:MI:SS') AS "Inst Start Time" FROM V$INSTANCE; Inst Start Time ---------- JUN-18-09 **13:16:40**	This query shows the time that the current instance was started. The different start time shows that this instance is different from the one that shut down the database.	此查询显示当前实例的启动时间。不同的开始时间显示此实例不同于之前已关闭数据库的那个实例。

Oracle System Identifier (SID)

The **system identifier (SID)** is a unique name for an Oracle database instance on a specific host. On UNIX and Linux, Oracle Database uses the SID and Oracle home values to create a key to shared memory. Also, the SID is used by default to locate the parameter file, which is used to locate relevant files such as the database control files.

On most platforms, the `ORACLE_SID` environment variable sets the SID, whereas the `ORACLE_HOME` variable sets the Oracle home. When connecting to an instance, clients can specify the SID in an Oracle Net connection or use a net service name. Oracle Database converts a service name into an `ORACLE_HOME` and `ORACLE_SID`.

See Also:

- "Service Names"
- *Oracle Database Administrator's Guide* to learn how to specify an Oracle SID

Overview of Instance Startup and Shutdown

A database instance provides user access to a database. This section explains the possible states of the instance and the database.

Overview of Instance and Database Startup

In a typical use case, you manually start an instance and then mount and open the database, making it available for users. You can use the SQL*Plus `STARTUP` command, Oracle Enterprise Manager (Enterprise Manager), or the SRVCTL utility to perform these steps. Figure 13-3 shows how a database progresses from a shutdown state to an open state.

Figure 13-3 Instance and Database Startup Sequence

Oracle 系统标识符 (SID)

系统标识符 (SID) 是特定主机上的某个 Oracle 数据库实例的唯一名称。在 UNIX 及 Linux 上，Oracle 数据库使用 SID 和 Oracle Home 的值来创建一个指向共享内存的键。此外，SID 在默认情况下用于定位参数文件，并使用参数文件来进一步定位其它相关文件，如数据库控制文件等。

在大多数的平台上， 由 ORACLE_SID 环境变量设置 SID，而 ORACLE_HOME 变量设置 Oracle Home。当客户端要连接到实例时，可以在 Oracle Net 连接中指定 SID，或使用网络服务名称。Oracle 数据库将服务名称转换为一个 ORACLE_HOME 和 ORACLE_SID 。

另见：

- "服务名"
- 《*Oracle 数据库管理员指南*》了解如何指定 Oracle SID

实例启动和关闭概述

数据库实例提供了用户对数据库的访问。此部分说明了实例和数据库可能存在的状态。

实例和数据库启动概述

举一个典型的使用案例，您手动启动一个实例，然后装载并打开数据库，使其对用户可用。您可以使用 SQL*Plus 的 `STARTUP` 命令、 Oracle 企业管理器 （企业管理器） 、或 SRVCTL 实用程序来执行这些步骤。图 13-3 显示了数据库如何从关闭状态一步步推进到打开状态。

图 13-3 实例和数据库启动序列

Description of "Figure 13-3 Instance and Database Startup Sequence"

A database goes through the following phases when it proceeds from a shutdown state to an open database state:

- 1 Instance started without mounting database

  The instance is started, but is not yet associated with a database. "How an Instance Is Started" explains this stage.
- 2 Database mounted

  The instance is started and is associated with a database by reading its control file (see "Overview of Control Files"). The database is closed to users.
  "How a Database Is Mounted" explains this stage.
- 3 Database open

  The instance is started and is associated with an open database. The data contained in the data files is accessible to authorized users.
  "How a Database Is Opened" explains this stage.

See Also:

- "Oracle Enterprise Manager"
- *Oracle Database 2 Day DBA* and *Oracle Database Administrator's Guide* to learn how to start an instance
- *Oracle Database Administrator's Guide* to learn how to use SRVCTL

Connection with Administrator Privileges

Description of "Figure 13-3 Instance and Database Startup Sequence"

当数据库从关闭状态推进到打开状态时，会经历以下几个阶段：

- 1 启动实例，但未装载数据库

  实例已启动，但尚未与某个数据库相关联。
  "实例是如何启动的"解释了这一阶段。
- 2 装载数据库

  实例已启动，并且通过读取控制文件，来与数据库相关联 （请参阅"控制文件概述"）。数据库对用户是关闭的。

  "数据库是如何装载的"解释了这一阶段。
- 3 打开数据库

  实例已启动，并与一个打开的数据库相关联。授权的用户可以访问数据文件中包含的数据。

  "数据库是如何打开的"解释了这一阶段。

另见：

- "Oracle 企业管理器"
- 《*Oracle 数据库 2 日 DBA*》 和 《*Oracle 数据库管理员指南*》了解如何启动实例
- 《*Oracle 数据库管理员指南*》了解如何使用 SRVCTL

具有管理员权限的连接

Database startup and shutdown are powerful administrative options that are restricted to users who connect to Oracle Database with administrator privileges. Normal users do not have control over the current status of an Oracle database.

Depending on the operating system, one of the following conditions establishes administrator privileges for a user:

- The operating system privileges of the user enable him or her to connect using administrator privileges.
- The user is granted the SYSDBA or SYSOPER system privileges and the database uses password files to authenticate database administrators over the network.

SYSDBA and SYSOPER are special **system privileges** that enable access to a database instance even when the database is not open. Control of these privileges is outside of the database itself.When you connect with the SYSDBA system privilege, you are in the schema owned by SYS. When you connect as SYSOPER, you are in the public schema. SYSOPER privileges are a subset of SYSDBA privileges.

See Also:

- "SYS and SYSTEM Schemas"
- "Overview of Database Security" to learn about password files and authentication for database administrators
- *Oracle Database Administrator's Guide* to learn about SYSDBA and SYSOPER

How an Instance Is Started

When Oracle Database starts an instance, it performs the following basic steps:

- ✧ 1. Searches for a server parameter file in a platform-specific default location and, if not found, for a text initialization parameter file (specifying STARTUP with the SPFILE or PFILE parameters overrides the default behavior)
- ✧ 2. Reads the parameter file to determine the values of initialization parameters
- ✧ 3. Allocates the SGA based on the initialization parameter settings
- ✧ 4. Starts the Oracle background processes

数据库启动和关闭是功能强大的管理选项，仅限于能以管理员权限连接到 Oracle 数据库的用户来使用。一般用户对 Oracle 数据库的当前状态不具有控制权。

取决于不同的操作系统，使用下列条件之一来建立用户的管理员权限：

- 用户的操作系统权限使其能够使用管理员权限来连接。
- 用户被授予了 SYSDBA 或 SYSOPER 的系统权限，数据库通过网络使用密码文件对数据库管理员进行身份验证。

SYSDBA 和 SYSOPER 是特殊的系统权限，使你甚至能够在数据库未打开时仍能访问数据库实例。这些权限控制处于数据库本身之外。当你使用 SYSDBA 系统权限连接时，你将处于 SYS 模式中。当你使用 SYSOPER 连接时，你将处于 PUBLIC 模式中。SYSOPER 权限是 SYSDBA 权限的子集。

另见：

- "SYS 和 SYSTEM 模式"
- "数据库安全性概述"了解密码文件和数据库管理员的身份验证
- 《*Oracle 数据库管理员指南*》了解 SYSDBA 和 SYSOPER

实例是如何启动的

当 Oracle 数据库启动一个实例时，它执行以下基本步骤：

- ✧ 1. 在特定于平台的默认位置搜索服务器参数文件，如果未找到，则搜索一个初始化参数文本文件（为 STARTUP 指定 SPFILE 或 PFILE 参数将覆盖该默认行为）
- ✧ 2. 读取参数文件，以确定初始化参数值
- ✧ 3. 基于初始化参数设置，分配 SGA
- ✧ 4. 启动 Oracle 后台进程

- 5. Opens the alert log and trace files and writes all explicit parameter settings to the alert log in valid parameter syntax

At this stage, no database is associated with the instance. Scenarios that require a NOMOUNT state include database creation and certain backup and recovery operations.

See Also:

Oracle Database Administrator's Guide to learn how to manage initialization parameters using a server parameter file

How a Database Is Mounted

The instance **mounts** a database to associate the database with this instance. To mount the database, the instance obtains the names of the database control files specified in the CONTROL_FILES initialization parameter and opens the files. Oracle Database reads the control files to find the names of the data files and the online redo log files that it will attempt to access when opening the database.

In a mounted database, the database is closed and accessible only to database administrators. Administrators can keep the database closed while completing specific maintenance operations. However, the database is not available for normal operations.

If Oracle Database allows multiple instances to mount the same database concurrently, then the CLUSTER_DATABASE initialization parameter setting can make the database available to multiple instances. Database behavior depends on the setting:

- If CLUSTER_DATABASE is false (default) for the first instance that mounts a database, then only this instance can mount the database.
- If CLUSTER_DATABASE is true for the first instance, then other instances can mount the database if their CLUSTER_DATABASE parameter settings are set to true. The number of instances that can mount the database is subject to a predetermined maximum specified when creating the database.

See Also:

- *Oracle Database Administrator's Guide* to learn how to mount a database
- *Oracle Real Application Clusters Administration and Deployment Guide* for more information about the use of multiple instances with

- 5. 打开警报日志和跟踪文件，并以有效的参数语法将所有显式参数设置写入警报日志中

这一阶段还没有数据库与该实例相关联。需要 NOMOUNT 状态的场景包括创建数据库和某些备份与恢复操作。

另见：

《*Oracle 数据库管理员指南*》了解如何使用服务器参数文件来管理初始化参数

数据库是如何装载的

由实例装载数据库，以将数据库与该实例相关联。为装载数据库，该实例获取由 CONTROL_FILES 初始化参数指定的数据库控制文件名称，并打开文件。Oracle 数据库读取控制文件，以查找数据文件和联机重做日志文件的名称，当打开数据库时，它会尝试访问这些文件。

在一个已装载的数据库中，该数据库是关闭的，且只有数据库管理员可以访问。管理员可以在完成某些特定的维护操作时保持数据库关闭。但是，数据库此时还不可用于常规操作。

如果 Oracle 数据库允许多个实例同时装载同一数据库，则 CLUSTER_DATABASE 初始化参数设置可以使数据库可用于多个实例。数据库的行为取决于其设置：

- 如果装入数据库的第一个实例的 CLUSTER_DATABASE 为 false （默认），则仅此实例可以装入数据库。
- 如果第一个实例的 CLUSTER_DATABASE 为 true，则其他实例在其 CLUSTER_DATABASE 参数也设置为 true 时可以装载数据库。可以装载数据库的实例数量决定于在创建数据库时指定的预定最大值。

另见：

- 《*Oracle 数据库管理员指南*》了解如何装载数据库
- 《*Oracle 真正应用集群管理和部署指南*》关于如何将单个数据库和多

a single database

How a Database Is Opened

Opening a mounted database makes it available for normal database operations. Any valid user can connect to an open database and access its information. Usually, a database administrator opens the database to make it available for general use.

When you open the database, Oracle Database performs the following actions:

- Opens the online data files in tablespaces other than undo tablespaces

  If a tablespace was offline when the database was previously shut down (see "Online and Offline Tablespaces"), then the tablespace and its corresponding data files will be offline when the database reopens.
- Acquires an undo tablespace

  If multiple undo tablespaces exists, then the `UNDO_TABLESPACE` initialization parameter designates the undo tablespace to use. If this parameter is not set, then the first available undo tablespace is chosen.
- Opens the online redo log files

See Also:

"Data Repair"

Read-Only Mode

By default, the database opens in **read/write mode**. In this mode, users can make changes to the data, generating redo in the online redo log. Alternatively, you can open in **read-only mode** to prevent data modification by user transactions.

Note:

By default, a physical standby database opens in read-only mode. See *Oracle Data Guard Concepts and Administration*.

Read-only mode restricts database access to read-only transactions, which cannot write to data files or to online redo log files. However, the database can perform recovery or operations that change the database state without generating redo. For example, in read-only mode:

个实例一起使用的详细信息

数据库是如何打开的

打开一个已装载的数据库，使其可用于常规的数据库操作。任何有效的用户可以连接到打开的数据库，并访问其信息。通常，由数据库管理员打开数据库，使其可用于一般用途。

在打开数据库时， Oracle 数据库执行下列操作：

- 打开除撤消表空间之外的其他的表空间中的联机数据文件

  如果在之前数据库关闭时，某个表空间是脱机的（见"联机和脱机表空间"），则该表空间及其相应的数据文件，在重新打开数据库时，仍将处于脱机状态。
- 获取一个撤消表空间

  如果存在多个撤消表空间，则由 UNDO_TABLESPACE 初始化参数指定要使用的撤消表空间。如果尚未设置此参数，则会选择第一个可用的撤消表空间。
- 打开联机重做日志文件

另见：

"数据修复"

只读模式

默认情况下，数据库打开为**读/写模式**。在这种模式下，用户可以对数据作出更改，并在联机重做日志中生成重做。或者，也可以打开为只读模式，以防止数据被用户事务修改。

注意：

默认情况下，物理备用数据库将以只读模式打开。请参见 《*Oracle 数据卫士概念和管理*》 。

只读模式限制数据库访问只能为只读事务，也就无法写入到数据文件或联机重做日志文件。但是，数据库能执行恢复，或更改数据库状态但不生成重做的操作。例如，在只读模式：

- Data files can be taken offline and online. However, you cannot take permanent tablespaces offline.
- Offline data files and tablespaces can be recovered.
- The control file remains available for updates about the state of the database.
- Temporary tablespaces created with the `CREATE TEMPORARY TABLESPACE` statement are read/write.
- Writes to operating system audit trails, trace files, and alert logs can continue.

See Also:

Oracle Database Administrator's Guide to learn how to open a database in read-only mode

Database File Checks

If any of the data files or redo log files are not present when the instance attempts to open the database, or if the files are present but fail consistency tests, then the database returns an error. Media recovery may be required.

See Also:

"Backup and Recovery"

Overview of Database and Instance Shutdown

In a typical use case, you manually shut down the database, making it unavailable for users while you perform maintenance or other administrative tasks. You can use the SQL*Plus `SHUTDOWN` command or Enterprise Manager to perform these steps. Figure 13-4 shows the progression from an open state to a consistent shutdown.

Figure 13-4 Instance and Database Shutdown Sequence

- 数据文件可以被脱机和联机。但是，你不能将永久表空间脱机。
- 脱机数据文件和表空间是可以恢复的。
- 控制文件仍然可用，以更新有关数据库的状态。
- 使用 `CREATE TEMPORARY TABLESPACE` 语句创建的临时表空间是可读/写的。
- 可以继续往操作系统写入审计文件、跟踪文件、和警报日志。

另见：

《*Oracle 数据库管理员指南*》了解如何在只读模式下打开数据库

数据文件检查

当实例尝试打开数据库时，如果任何数据文件或重做日志文件不存在，或者虽然文件存在，但一致性测试失败，则数据库将返回一个错误。此时可能需要介质恢复。

另见：

"备份和恢复"

数据库和实例关闭概述

举一个典型的使用案例，您手动关闭数据库以执行维护或其他管理任务，这将使数据库对用户不可用。您可以使用 SQL * Plus 的 `SHUTDOWN` 命令或企业管理器来执行这些步骤。图 13-4 显示从打开状态推进到一致关闭的过程。

图 13-4 实例和数据库关闭顺序

Description of "Figure 13-4 Instance and Database Shutdown Sequence"

Oracle Database automatically performs the following steps whenever an open database is shut down consistently:

1. Database closed

   The database is mounted, but online data files and redo log files are closed.
   "How a Database Is Closed" explains this stage.
3. Database unmounted

   The instance is started, but is no longer associated with the control file of the database.
   "How a Database Is Unmounted" explains this stage.
5. Database instance shut down

   The database instance is no longer started.
   "How an Instance Is Shut Down" explains this stage.

Oracle Database does not go through all of the preceding steps in an instance failure or `SHUTDOWN ABORT`, which immediately terminates the instance.

See Also:

Oracle Database 2 Day DBA and *Oracle Database Administrator's Guide* to learn how to shut down a database

Shutdown Modes

A database administrator with `SYSDBA` or `SYSOPER` privileges can shut

Description of "Figure 13-4 Instance and Database Shutdown Sequence"

当一致地关闭打开的数据库时，Oracle 数据库自动执行以下步骤：

2. 数据库关闭

   数据库仍处于装载状态，但数据文件和联机重做日志文件已被关闭。

   "数据库是如何关闭的"解释了这一阶段。
4. 数据库卸载

   实例仍处于启动状态，但已不再与数据库的控制文件相关联。

   "数据库是如何卸载的"解释了这一阶段。
6. 数据库实例关闭

   数据库实例不再处于启动状态。
   "实例是如何关闭的"解释了这一阶段。

当实例失败或遭遇 `SHUTDOWN ABORT` 时，Oracle 数据库不会经历之前这些所有步骤，而只是立即终止该实例。

另见：

《*Oracle 数据库 2 日 DBA*》 和 《*Oracle 数据库管理员指南*》了解如何关闭数据库

关闭模式

具有 SYSDBA 或 SYSOPER 权限的数据库管理员，可以使用 SQL*Plus 的 `SHUTDOWN` 命令或企业管理器来关闭数据库。`SHUTDOWN` 命令有几个决定

down the database using the SQL*Plus SHUTDOWN command or Enterprise Manager. The SHUTDOWN command has options that determine shutdown behavior. Table 13-2 summarizes the behavior of the different shutdown modes.

Table 13-2 Shutdown Modes

Database Behavior	ABORT	IMMEDIATE	TRANSACTIONAL	NORMAL
Permits new user connections	No	No	No	No
Waits until current sessions end	No	No	No	Yes
Waits until current transactions end	No	No	Yes	Yes
Performs a checkpoint and closes open files	No	Yes	Yes	Yes

The possible SHUTDOWN statements are:

- SHUTDOWN ABORT

  This mode is intended for emergency situations, such as when no other form of shutdown is successful. This mode of shutdown is the fastest. However, a subsequent open of this database may take substantially longer because instance recovery must be performed to make the data files consistent.

  Note:

  Because SHUTDOWN ABORT does not checkpoint the open data files, instance recovery is necessary before the database can reopen. The other shutdown modes do not require instance recovery before the database can reopen.

- SHUTDOWN IMMEDIATE

  This mode is typically the fastest next to SHUTDOWN ABORT. Oracle Database terminates any executing SQL statements and disconnects users. Active transactions are terminated and uncommitted changes are rolled back.

- SHUTDOWN TRANSACTIONAL

  This mode prevents users from starting new transactions, but waits for all current transactions to complete before shutting down. This

关闭行为的选项。表 13-2 总结了在各种不同关闭模式下的行为。

表 13-2 关闭模式

数据库行为	ABORT	IMMEDIATE	TRANSACTIONAL	NORMAL
允许新用户连接	否	否	否	否
将等待，直到当前会话结束	否	否	否	是
将等待，直到当前事务结束	否	否	是	是
执行一个检查点，并关闭打开的文件	否	是	是	是

可能的几种 SHUTDOWN 语句选项是：

- SHUTDOWN ABORT

  此模式用于紧急情况，如其他形式的关闭方式都未能成功时。这种模式的关闭是最快的。但是，随后打开该数据库可能时间会显著增加，因为必须执行实例恢复以使数据文件一致。

  注意：

  因为 SHUTDOWN ABORT 不对打开的数据文件执行检查点操作，所以在重新打开数据库之前，必须进行实例恢复。对于其他的关闭模式，在数据库重新打开之前不需要进行实例恢复。

- SHUTDOWN IMMEDIATE

  这种模式通常是除 SHUTDOWN ABORT 之外最快的关闭模式了。Oracle 数据库终止任何正在执行的 SQL 语句，并断开用户连接。所有活动事务都将终止，并回滚所有未提交的更改。

- SHUTDOWN TRANSACTIONAL

  这种模式可以防止用户启动新事务，但在关闭之前会等待所有的当前事务完成。这种模式可能会花费相当长的时间，这取决于当前事务的

mode can take a significant amount of time depending on the nature of the current transactions.

- SHUTDOWN NORMAL

  This is the default mode of shutdown. The database waits for all connected users to disconnect before shutting down.

See Also:

- *Oracle Database 2 Day DBA* and *Oracle Database Administrator's Guide* to learn about the different shutdown modes
- *SQL*Plus User's Guide and Reference* to learn about the SHUTDOWN command

How a Database Is Closed

The database close operation is implicit in a database shutdown. The nature of the operation depends on whether the database shutdown is normal or abnormal.

How a Database Is Closed During Normal Shutdown

When a database is closed as part of a SHUTDOWN with any option other than ABORT, Oracle Database writes data in the SGA to the data files and online redo log files. Next, the database closes online data files and online redo log files. Any offline data files of offline tablespaces have been closed already. When the database reopens, any tablespace that was offline remains offline.

At this stage, the database is closed and inaccessible for normal operations. The control files remain open after a database is closed.

How a Database Is Closed During Abnormal Shutdown

If a SHUTDOWN ABORT or abnormal termination occurs, then the instance of an open database closes and shuts down the database instantaneously. Oracle Database does not write data in the buffers of the SGA to the data files and redo log files. The subsequent reopening of the database requires instance recovery, which Oracle Database performs automatically.

How a Database Is Unmounted

After the database is closed, Oracle Database unmounts the database to disassociate it from the instance. After a database is unmounted, Oracle Database closes the control files of the database. At this point, the instance remains in memory.

How an Instance Is Shut Down

The final step in database shutdown is shutting down the instance. When

特征。

- SHUTDOWN NORMAL

  这是默认的关闭模式。在关闭之前数据库会等待所有连接的用户断开连接。

另见：

- 《*Oracle 数据库 2 日 DBA*》 和 《*Oracle 数据库管理员指南*》了解不同的关闭模式
- 《SQL*Plus 用户指南和参考》了解 SHUTDOWN 命令

数据库是如何关闭的

数据库的关闭操作是在数据库停机中的一个隐含步骤。该操作的性质取决于数据库停机是正常还是非正常的。

正常停机过程中是如何关闭数据库的

当数据库作为 SHUTDOWN 的一部分（不使用 ABORT 选项）而关闭时，Oracle 数据库将 SGA 中的数据写入数据文件和联机重做日志文件。然后数据库关闭联机数据文件和联机重做日志文件。脱机表空间中的任何脱机数据文件是本来就是已关闭的。当重新打开数据库时，任何本来脱机的空间仍然保持脱机状态。

这一阶段中，该数据库是关闭的，且无法进行正常操作访问。在数据库关闭后，控制文件仍保持打开状态。

非正常关机过程中是如何关闭数据库

如果执行了 SHUTDOWN ABORT，或出现异常终止，则打开数据库的实例会关闭，并在瞬间将数据库停机。Oracle 数据库不会将 SGA 缓冲区中的数据写入数据文件和重做日志文件。随后重新打开数据库需要实例恢复，这将由 Oracle 数据库自动执行。

数据库是如何卸载的

数据库关闭之后，Oracle 将数据库卸载，并解除与实例的关联。数据库卸载后，Oracle 数据库关闭数据库的控制文件。这个时候，实例仍将保留在内存中。

实例是如何关闭的

关闭数据库的最后一步是关闭实例。当数据库实例关闭时，SGA 被从内存中

the database instance is shut down, the SGA is removed from memory and the background processes are terminated.
In unusual circumstances, shutdown of an instance may not occur cleanly. Memory structures may not be removed from memory or one of the background processes may not be terminated. When remnants of a previous instance exist, a subsequent instance startup may fail. In such situations, you can force the new instance to start by removing the remnants of the previous instance and then starting a new instance, or by issuing a `SHUTDOWN ABORT` statement in SQL*Plus or using Enterprise Manager.

See Also:

Oracle Database Administrator's Guide for more detailed information about database shutdown

Overview of Checkpoints

A checkpoint is a crucial mechanism in consistent database shutdowns, instance recovery, and Oracle Database operation generally. The term **checkpoint** has the following related meanings:

- A data structure that indicates the **checkpoint position**, which is the SCN in the redo stream where instance recovery must begin

  The checkpoint position is determined by the oldest dirty buffer in the database buffer cache. The checkpoint position acts as a pointer to the redo stream and is stored in the control file and in each data file header.
- The writing of modified database buffers in the database buffer cache to disk

See Also:

"System Change Numbers (SCNs)"

Purpose of Checkpoints

Oracle Database uses checkpoints to achieve the following goals:

- Reduce the time required for recovery in case of an instance or media failure
- Ensure that dirty buffers in the buffer cache are written to disk regularly
- Ensure that all committed data is written to disk during a consistent shutdown

When Oracle Database Initiates Checkpoints

移除，且后台进程都将被终止。

在异常情况下，实例可能并未干净地关闭。内存结构可能未从内存中删除，或某个后台进程可能未被终止。当之前的实例存在残留物时，后续的实例启动可能会失败。在这样的情况下，为强制启动一个新实例，您可以通过删除之前实例的残留物并启动一个新实例、或通过在 SQL*Plus 中发出 `SHUTDOWN ABORT` 语句、或使用企业管理器。

另见：

《*Oracle 数据库管理员指南*》关于关闭数据库的更详细的信息

检查点概述

检查点是进行一致的数据库关闭、 实例恢复、和 Oracle 数据库通用操作的关键机制。术语**检查点**具有如下相关含义：

- 指示**检查点位置**的一个数据结构，该位置是在重做流中实例恢复必须开始处的 SCN

  检查点位置是由数据库缓冲区高速缓存中最旧的脏缓冲区来确定的。检查点位置作为一个指向重做流的指针，并存储在控制文件中，和在每个数据文件头中。
- 将数据库缓存中已修改的数据库缓冲区写入到磁盘中

另见：

"系统更改号（SCNs）"

检查点的目的

Oracle 数据库使用检查点，来实现以下目标：

- 减少实例失败或介质故障情况下恢复所需的时间
- 确保在缓冲区高速缓存中的脏缓冲区被定期写入磁盘
- 确保在一致的关闭过程中所有已提交的数据都被写入磁盘

Oracle 数据库何时启动检查点

The **checkpoint process (CKPT)** is responsible for writing checkpoints to the data file headers and control file. Checkpoints occur in a variety of situations. For example, Oracle Database uses the following types of checkpoints:

- Thread checkpoints

  The database writes to disk all buffers modified by redo in a specific thread before a certain target. The set of thread checkpoints on all instances in a database is a **database checkpoint**. Thread checkpoints occur in the following situations:
  - Consistent database shutdown
  - `ALTER SYSTEM CHECKPOINT` statement
  - Online redo log switch
  - `ALTER DATABASE BEGIN BACKUP` statement

- Tablespace and data file checkpoints

  The database writes to disk all buffers modified by redo before a specific target. A tablespace checkpoint is a set of data file checkpoints, one for each data file in the tablespace. These checkpoints occur in a variety of situations, including making a tablespace read-only or taking it offline normal, shrinking a data file, or executing `ALTER TABLESPACE BEGIN BACKUP`.
- Incremental checkpoints

  An incremental checkpoint is a type of thread checkpoint partly intended to avoid writing large numbers of blocks at online redo log switches. DBW*n* checks at least every three seconds to determine whether it has work to do. When DBW*n* writes dirty buffers, it advances the checkpoint position, causing CKPT to write the checkpoint position to the control file, but not to the data file headers.

Other types of checkpoints include instance and media recovery checkpoints and checkpoints when schema objects are dropped or truncated.

See Also:

- "Checkpoint Process (CKPT)"

检查点进程 （CKPT） 负责将检查点写入数据文件头和控制文件。检查点会在多种情况下发生。例如， Oracle 数据库使用以下类型的检查点：

- 线程检查点

  数据库将某个确定目标之前、被某个特定的重做线程所修改的所有缓冲区写入磁盘。数据库中所有实例的线程检查点的集合即为数据库检查点。线程检查点在下列情况下发生：
  - 一致的数据库关闭
  - `ALTER SYSTEM CHECKPOINT` 语句
  - 联机重做日志切换
  - `ALTER DATABASE BEGIN BACKUP` 语句

- 表空间和数据文件的检查点

  数据库将某个确定目标之前、被重做线程所修改的所有缓冲区写入磁盘。表空间检查点是一组数据文件检查点，每个数据文件检查点对表空间中的某个数据文件做检查点操作。这些检查点发生在很多情况下，包括将一个表空间变为只读、将表空间脱机、收缩数据文件、或执行 `ALTER TABLESPACE BEGIN BACKUP` 等。

- 增量检查点

  增量检查点是一种线程检查点，部分原因是为了避免在联机重做日志切换时写入大量的块。DBWn 至少每隔三秒会进行检查以确定是否有工作要做。当 DBWn 将脏缓冲区写入磁盘时，它会向前推进检查点位置，导致 CKPT 将检查点位置写入控制文件，而不是数据文件头。

其他类型的检查点包括实例和介质恢复检查点，和删除或截断模式对象时的检查点。

另见：

- "检查点进程(CKPT)"

- *Oracle Real Application Clusters Administration and Deployment Guide* for information about global checkpoints in Oracle RAC

Overview of Instance Recovery

Instance recovery is the process of applying records in the online redo log to data files to reconstruct changes made after the most recent checkpoint. Instance recovery occurs automatically when an administrator attempts to open a database that was previously shut down inconsistently.

Purpose of Instance Recovery

Instance recovery ensures that the database is in a consistent state after an instance failure. The files of a database can be left in an inconsistent state because of how Oracle Database manages database changes.

A redo thread is a record of all of the changes generated by an instance. A single-instance database has one thread of redo, whereas an Oracle RAC database has multiple redo threads, one for each database instance.

When a transaction is committed, log writer (LGWR) writes both the remaining redo entries in memory and the transaction SCN to the online redo log. However, the database writer (DBW*n*) process writes modified data blocks to the data files whenever it is most efficient. For this reason, uncommitted changes may temporarily exist in the data files while committed changes do not yet exist in the data files.

If an instance of an open database fails, either because of a `SHUTDOWN ABORT` statement or abnormal termination, then the following situations can result:

- Data blocks committed by a transaction are not written to the data files and appear only in the online redo log. These changes must be reapplied to the database.
- The data files contains changes that had not been committed when the instance failed. These changes must be rolled back to ensure transactional consistency.

Instance recovery uses only online redo log files and current online data files to synchronize the data files and ensure that they are consistent.

See Also:

- "Database Writer Process (DBW***n***)" and "Database Buffer Cache"
- "Introduction to Data Concurrency and Consistency"

When Oracle Database Performs Instance Recovery

- 《*Oracle 真正应用集群管理和部署指南*》了解在 Oracle RAC 中的全局检查点信息

实例恢复概述

实例恢复是将联机重做日志中的记录应用到数据文件，以重建最近检查点之后所做更改的过程。当管理员尝试打开一个之前以不一致方式关闭的数据库时，会自动执行实例恢复。

实例恢复的目的

实例恢复可确保数据库在一个实例失败后仍能回到一个一致的状态。由于 Oracle 数据库对数据文件更改的管理方式所致，数据库的文件可以处于不一致的状态。

重做线程是对实例生成的所有更改的记录。单实例数据库拥有一个重做线程，而一个 Oracle RAC 数据库拥有多个重做线程——每个数据库实例有一个。

当事务提交时，日志写入器 (LGWR) 将内存中的重做条目和事务 SCN 同时写入联机重做日志。但是，数据库写入器 (DBWn) 进程只在最有利的时机将已修改的数据块写入数据文件。由于这个原因，未提交的更改可能会暂时存在于数据文件中，而已提交的更改也可能还不在数据文件中。

如果某个打开的数据库的实例失败，或者由于 `SHUTDOWN ABORT` 语句或异常终止，则可能会导致下列情况：

- 由某事务已提交的数据块更新还未写入数据文件，而仅写入了联机重做日志中。这些更改必须重新应用到数据库。
- 数据文件包含实例失败时尚未提交的更改。这些更改必须回滚，以确保事务一致性。

实例恢复只使用联机重做日志文件和当前在线的数据文件，以同步数据文件，并确保它们一致。

另见：

- "数据库写入过程 （DBWn）"和"数据库缓冲区高速缓存"
- "数据并发性和一致性简介"

Oracle 数据库何时执行实例恢复

Whether instance recovery is required depends on the state of the redo threads. A redo thread is marked open in the control file when a database instance opens in read/write mode, and is marked closed when the instance is shut down consistently. If redo threads are marked open in the control file, but no live instances hold the thread enqueues corresponding to these threads, then the database requires instance recovery.

Oracle Database performs instance recovery automatically in the following situations:

- The database opens for the first time after the failure of a single-instance database or all instances of an Oracle RAC database. This form of instance recovery is also called **crash recovery**. Oracle Database recovers the online redo threads of the terminated instances together.
- Some but not all instances of an Oracle RAC database fail. Instance recovery is performed automatically by a surviving instance in the configuration.

The SMON background process performs instance recovery, applying online redo automatically. No user intervention is required.

See Also:

- "System Monitor Process (SMON)"
- *Oracle Real Application Clusters Administration and Deployment Guide* to learn about instance recovery in an Oracle RAC database

Importance of Checkpoints for Instance Recovery

Instance recovery uses checkpoints to determine which changes must be applied to the data files. The checkpoint position guarantees that every committed change with an SCN *lower than* the checkpoint SCN is saved to the data files.

Figure 13-5 depicts the redo thread in the online redo log.

Figure 13-5 Checkpoint Position in Online Redo Log

是否需要实例恢复取决于重做线程的状态。在数据库实例被打开为读/写模式时，重做线程在控制文件中被标记为打开，而当实例被一致关闭时，重做线程被标记为关闭。如果重做线程在控制文件中被标记为打开，但没有活动的实例持有对应于这些线程的线程队列，则数据库将需要实例恢复。

Oracle 数据库在以下情况下自动执行实例恢复：

- 单实例数据库或 Oracle RAC 数据库的所有实例失败后第一次打开数据库。这种形式的实例恢复也称为**崩溃恢复**。Oracle 数据库一起恢复所有已终止实例的联机重做线程。
- 只是 Oracle RAC 数据库中的某些、但不是所有实例失败。实例恢复将由配置中的某个存活实例自动进行。

SMON 后台进程自动执行实例恢复并应用联机重做记录。而不需要任何用户干预。

另见：

- " 系统监控进程（SMON）"
- 《*Oracle 真正应用集群管理和部署指南*》了解在 Oracle RAC 数据库中的实例恢复

实例恢复检查点的重要性

实例恢复使用检查点来确定必须将哪些更改应用到数据文件。检查点位置始终保证所有比其 SCN 低的检查点所对应的已提交更改都已保存到数据文件。

图 13-5 描述了联机重做日志中的重做线程。

图 13-5 联机重做日志中的检查点位置

Description of "Figure 13-5 Checkpoint Position in Online Redo Log"

During instance recovery, the database must apply the changes that occur between the checkpoint position and the end of the redo thread. As shown in Figure 13-5, some changes may already have been written to the data files. However, only changes with SCNs lower than the checkpoint position are *guaranteed* to be on disk.

See Also:

Oracle Database Performance Tuning Guide to learn how to limit instance recovery time

Instance Recovery Phases

The first phase of instance recovery is called cache recovery or **rolling forward**, and involves reapplying all of the changes recorded in the online redo log to the data files. Because rollback data is recorded in the online redo log, rolling forward also regenerates the corresponding undo segments.

Rolling forward proceeds through as many online redo log files as necessary to bring the database forward in time. After rolling forward, the data blocks contain all committed changes recorded in the online redo log files. These files could also contain uncommitted changes that were either saved to the data files before the failure, or were recorded in the online redo log and introduced during cache recovery.

After the roll forward, any changes that were not committed must be undone. Oracle Database uses the checkpoint position, which guarantees that every committed change with an SCN lower than the checkpoint SCN is saved on disk. Oracle Database applies undo blocks to roll back uncommitted changes in data blocks that were written before the failure or introduced during cache recovery. This phase is called **rolling back** or

Description of "Figure 13-5 Checkpoint Position in Online Redo Log"

实例恢复期间，数据库必须应用检查点位置和重做线程结尾之间发生的更改。如图 13-5 所示， 某些更改可能已经写入数据文件。但是，只有其 SCN 低于检查点位置的更改，才保证已被写到了磁盘上。

另见：

《*Oracle 数据库性能优化指南*》了解如何限制实例恢复时间

实例恢复阶段

实例恢复的第一阶段称为缓存恢复或**前滚**，这涉及将联机重做日志中记录的所有更改重新应用到数据文件。因为回滚数据记录在联机重做日志中，前滚也会重新生成相应的撤消段。

前滚会遍历各个必要的联机重做日志，以将数据库推进到一个更前的一致时间点。前滚之后，数据块包含记录在联机重做日志文件中的所有已提交更改。这些文件可能还包含未提交的更改，要么是在实例失败前保存到数据文件中的，或者是在缓存恢复过程中引入的。

前滚之后，任何未提交的更改必须被撤消。Oracle 数据库使用检查点位置，保证每个低于其 SCN 的已提交更改都已保存到磁盘。Oracle 数据库应用撤消块，以回滚数据块中在实例失败前写入的或缓存恢复过程中引入的未提交更改。这一阶段称为**回滚**或事务恢复。

transaction recovery.
Figure 13-6 illustrates rolling forward and rolling back, the two steps necessary to recover from database instance failure.

Figure 13-6 Basic Instance Recovery Steps: Rolling Forward and Rolling Back

Description of "Figure 13-6 Basic Instance Recovery Steps: Rolling Forward and Rolling Back"

Oracle Database can roll back multiple transactions simultaneously as needed. All transactions that were active at the time of failure are marked as terminated. Instead of waiting for the SMON process to roll back terminated transactions, new transactions can roll back individual blocks themselves to obtain the required data.

See Also:

- "Undo Segments" to learn more about undo data
- *Oracle Database Performance Tuning Guide* for a discussion of instance recovery mechanics and tuning

Overview of Parameter Files

To start a database instance, Oracle Database must read either a **server parameter file**, which is recommended, or a **text initialization parameter file**, which is a legacy implementation. These files contain a list of configuration parameters.

图 13-6 说明了前滚和回滚，这是恢复数据库实例失败的两个必要步骤。

图 13-6 基本的实例恢复步骤： 前滚和回滚

Description of "Figure 13-6 Basic Instance Recovery Steps: Rolling Forward and Rolling Back"

Oracle 数据库可以根据需要同时回滚多个事务。实例失败时的所有活动事务被标记为终止。新事务可以自己回滚个别块以获取所需的数据，而不必等待 SMON 进程来回滚这些已终止的事务。

另见：

- "撤消段"了解撤消数据的更多信息
- 《*Oracle 数据库性能调整指南*》 关于实例恢复机制和调整的讨论

参数文件的概述

要启动数据库实例，Oracle 数据库必须读取一个（推荐的）**服务器参数文件**或一个（传统的）**文本初始化参数文件**。这些文件包含配置参数的列表。

To create a database manually, you must start an instance with a parameter file and then issue a `CREATE DATABASE` command. Thus, the instance and parameter file can exist even when the database itself does not exist.

要手动创建一个数据库，必须用一个参数文件启动实例，然后发出 `CREATE DATABASE` 命令。因此，即使数据库本身还不存在，实例和参数文件即可以存在。

Initialization Parameters

Initialization parameters are configuration parameters that affect the basic operation of an instance. The instance reads initialization parameters from a file at startup.

Oracle Database provides many **initialization parameters** to optimize its operation in diverse environments. Only a few of these parameters must be explicitly set because the default values are adequate in most cases.

初始化参数

初始化参数是会影响实例基本操作的配置参数。实例在启动时从一个文件读取初始化参数。

Oracle 数据库提供了许多**初始化参数**，以优化其在不同环境中的操作。只需要显式设置几个参数，因为其默认值对大多数情况已经足够。

Functional Groups of Initialization Parameters

Most initialization parameters belong to one of the following functional groups:

- Parameters that name entities such as files or directories
- Parameters that set limits for a process, database resource, or the database itself
- Parameters that affect capacity, such as the size of the SGA (these parameters are called **variable parameters**)

Variable parameters are of particular interest to database administrators because they can use these parameters to improve database performance.

初始化参数的功能分组

绝大多数的初始化参数属于以下功能分组之一：

- 名字条目参数，如文件或目录
- 限制设置参数，如进程、 数据库资源、或数据库本身等
- 影响容量的参数，如 SGA 的大小（这些参数也称为**可变参数**）

数据库管理员可能会对可变参数特别感兴趣，因为他们可以使用这些参数来提高数据库性能。

Basic and Advanced Initialization Parameters

Initialization parameters are divided into two groups: basic and advanced. In most cases, you must set and tune only the approximately 30 basic parameters to obtain reasonable performance. The basic parameters set characteristics such as the database name, locations of the control files, database block size, and undo tablespace.

In rare situations, modification to the advanced parameters may be required for optimal performance. The advanced parameters enable expert DBAs to adapt the behavior of the Oracle Database to meet unique requirements.

Oracle Database provides values in the starter initialization parameter file provided with your database software, or as created for you by the Database Configuration Assistant (see "Tools for Database Installation and Configuration"). You can edit these Oracle-supplied initialization

基本和高级的初始化参数

初始化参数可以分为两组： 基本的和高级的。在大多数的情况下，你只需设置和调整大约 30 个基本参数，就可以获得还不错的性能。基本参数设置的特性包括数据库名称、控制文件的位置、数据库的块大小、和撤消表空间等。

在某些情况下，为获得最佳性能，可能需要修改高级参数。启用高级参数可以使专家级 DBA 能够调整 Oracle 数据库以满足某些特殊需求。

Oracle 数据库软件自带的初始化参数文件提供了一些初级参数值，或者你也可以通过数据库配置助手来产生这些值（请参见"数据库安装和配置工具"）。您可以编辑这些 Oracle 提供的初始化参数并添加其他参数，这取决于您的配置，以及你计划如何优化数据库。对于初始化参数文件中不包含的有

parameters and add others, depending on your configuration and how you plan to tune the database. For relevant initialization parameters not included in the parameter file, Oracle Database supplies defaults.

See Also:

- *Oracle Database 2 Day DBA* and *Oracle Database Administrator's Guide* to learn how to specify initialization parameters
- *Oracle Database Reference* for an explanation of the types of initialization parameters
- *Oracle Database Reference* for a description of `V$PARAMETER` and *SQL*Plus User's Guide and Reference* for `SHOW PARAMETER` syntax

Server Parameter Files

A **server parameter file** is a repository for initialization parameters that is managed by Oracle Database. A server parameter file has the following key characteristics:

- Only one server parameter file exists for a database. This file must reside on the database host.
- The server parameter file is written to and read by only by Oracle Database, not by client applications.
- The server parameter file is binary and cannot be modified by a text editor.
- Initialization parameters stored in the server parameter file are persistent. Any changes made to the parameters while a database instance is running can persist across instance shutdown and startup.

A server parameter file eliminates the need to maintain multiple text initialization parameter files for client applications. A server parameter file is initially built from a text initialization parameter file using the `CREATE SPFILE` statement. It can also be created directly by the Database Configuration Assistant.

See Also:

- *Oracle Database Administrator's Guide* to learn more about server parameter files

关参数，Oracle 数据库提供默认值。

另见：

- 《*Oracle 数据库 2 日 DBA*》和《*Oracle 数据库管理员指南*》了解如何指定初始化参数
- 《*Oracle 数据库参考*》关于初始化参数类型的说明
- 《*Oracle 数据库参考*》关于的 `V$PARAMETER` 和 《*SQL*Plus 用户指南和参考*》关于 `SHOW PARAMETER` 的语法

服务器参数文件

服务器参数文件是 Oracle 数据库用来管理初始化参数的一个存储库。服务器参数文件具有以下主要特征：

- 对于一个数据库，只存在一个服务器参数文件。此文件必须驻留在数据库主机上。
- 服务器参数文件只由 Oracle 数据库读取和写入，而不能由客户端应用程序读取和写入。
- 服务器参数文件是二进制的，且不能使用文本编辑器修改。
- 存储在服务器参数文件中的初始化参数是永久性的。数据库实例正在运行时对参数所做的任何更改，可以跨实例关闭和启动而存在。

服务器参数文件消除了为多个客户端应用程序维护多个文本初始化参数文件的需要。服务器参数文件最初可以从一个文本初始化参数文件，使用 `CREATE SPFILE` 语句来生成。也可以直接由数据库配置助手创建。

另见：

- 《*Oracle 数据库管理员指南*》了解服务器参数文件的更多信息

- *Oracle Database SQL Language Reference* to learn about `CREATE SPFILE`

Text Initialization Parameter Files

A **text initialization parameter file** is a text file that contains a list of initialization parameters. This type of parameter file, which is a legacy implementation of the parameter file, has the following key characteristics:

- When starting up or shutting down a database, the text initialization parameter file must reside on the same host as the client application that connects to the database.
- A text initialization parameter file is text-based, not binary.
- Oracle Database can read but not write to the text initialization parameter file. To change the parameter values you must manually alter the file with a text editor.
- Changes to initialization parameter values by `ALTER SYSTEM` are only in effect for the current instance. You must manually update the text initialization parameter file and restart the instance for the changes to be known.

The text initialization parameter file contains a series of `key=value` pairs, one per line. For example, a portion of an initialization parameter file could look as follows:

```
db_name=sample
control_files=/disk1/oradata/sample_cf.dbf
db_block_size=8192
open_cursors=52
undo_management=auto
shared_pool_size=280M
pga_aggregate_target=29M
.
.
.
```

To illustrate the manageability problems that text parameter files can create, assume that you use computers `clienta` and `clientb` and must be able to start the database with SQL*Plus on either computer. In this case, two separate text initialization parameter files must exist, one on each computer, as shown in Figure 13-7. A server parameter file solves

- 《*Oracle 数据库 SQL 语言参考*》了解 `CREATE SPFILE` 命令

文本初始化参数文件

文本初始化参数文件是一个文本文件，其中包含初始化参数的列表。这是一种旧式参数文件，具有以下关键特征：

- 当启动或关闭数据库时，文本初始化参数文件必须驻留在连接到该数据库的客户端应用程序相同的主机上。
- 文本初始化参数文件是基于文本的，而非二进制。
- Oracle 数据库可以读取，但不能写入文本初始化参数文件。若要更改参数值，必须使用文本编辑器手动更改文件。
- 通过 `ALTER SYSTEM` 对初始化参数值所做的更改，仅在当前实例中有效。您必须手动更新文本初始化参数文件，并重新启动实例以使更改生效。

文本初始化参数文件包含一系列 `key=value` 对，每行一个。例如，某个初始化参数文件中的一部分看起来如下所示：

```
db_name=sample
control_files=/disk1/oradata/sample_cf.dbf
db_block_size=8192
open_cursors=52
undo_management=auto
shared_pool_size=280M
pga_aggregate_target=29M
.
.
.
```

为了说明文本参数文件可能产生的可管理性问题，假定您使用计算机 clienta 和 clientb，并且必须能够从其中任何一台计算机上使用 SQL*Plus 启动数据库。在这种情况下，必须存在两个独立的文本初始化参数文件，每台计算机上一个，如图 13-7 所示。服务器参数文件解决了分散的参数文件问题。

the problem of the proliferation of parameter files.

Figure 13-7 Multiple Initialization Parameter Files

Description of "Figure 13-7 Multiple Initialization Parameter Files"

See Also:

- *Oracle Database Administrator's Guide* to learn more about text initialization parameter files
- *Oracle Database SQL Language Reference* to learn about `CREATE PFILE`

Modification of Initialization Parameter Values

You can adjust initialization parameters to modify the behavior of a database. The classification of parameters as **static** or **dynamic** determines how they can be modified. Table 13-3 summarizes the differences.

Table 13-3 Static and Dynamic Initialization Parameters

Characteristic	Static	Dynamic
Requires modification of the parameter file (text or server)	Yes	No
Requires database instance restart before setting takes affect	Yes	No
Described as "Modifiable" in *Oracle Database Reference* initialization parameter entry	No	Yes
Modifiable only for the database or instance	Yes	No

Static parameters include `DB_BLOCK_SIZE`, `DB_NAME`, and

图 13-7 多个初始化参数文件

Description of "Figure 13-7 Multiple Initialization Parameter Files"

另见：

- 《*Oracle 数据库管理员指南*》了解有关文本初始化参数文件的更多信息
- 《*Oracle 数据库 SQL 语言参考*》了解 `CREATE PFILE` 命令

修改初始化参数的值

您可以修改初始化参数，以调整数据库行为。将其按静态或动态参数的分类，确定了应如何修改它们。表 13-3 总结了的其中的差异。

表 13-3 静态和动态初始化参数

特征	静态	动态
需要修改（文本或服务器）参数文件	是	否
在设置生效之前数据库实例需要重启	是	否
在 《*Oracle 数据库参考*》初始化参数条目中被描述为"可修改"	否	是
仅对数据库或实例可修改	是	否

静态参数包括 `DB_BLOCK_SIZE`、`DB_NAME` 、和 `COMPATIBLE` 等。动

COMPATIBLE. Dynamic parameters are grouped into **session-level parameters**, which affect only the current user session, and **system-level parameters**, which affect the database and all sessions. For example, MEMORY_TARGET is a system-level parameter, while NLS_DATE_FORMAT is a session-level parameter (see "Locale-Specific Settings").

The **scope** of a parameter change depends on when the change takes effect. When an instance has been started with a server parameter file, you can use the ALTER SYSTEM SET statement to change values for system-level parameters as follows:

- SCOPE=MEMORY

  Changes apply to the database instance only. The change will not persist if the database is shut down and restarted.
- SCOPE=SPFILE

  Changes are written to the server parameter file but do not affect the current instance. Thus, the changes do not take effect until the instance is restarted.

  Note:

  You must specify SPFILE when changing the value of a parameter described as not modifiable in *Oracle Database Reference*.
- SCOPE=BOTH

  Changes are written both to memory and to the server parameter file. This is the default scope when the database is using a server parameter file.

  The database prints the new value and the old value of an initialization parameter to the alert log. As a preventative measure, the database validates changes of basic parameter to prevent illegal values from being written to the server parameter file.

See Also:

- *Oracle Database Administrator's Guide* to learn how to change initialization parameter settings
- *Oracle Database Reference* for descriptions of all initialization parameters
- *Oracle Database SQL Language Reference* for ALTER SYSTEM syntax and semantics

态参数可分为**会话级参数**和**系统级参数**，会话级参数只影响当前的用户会话，而系统级参数影响数据库及所有会话。例如，MEMORY_TARGET 是一个系统级参数，而 NLS_DATE_FORMAT 是一个会话级参数 （请参见"特定于区域的设置"）。

参数更改的**范围**取决于更改何时生效。若实例是用服务器参数文件启动的，则可以使用 ALTER SYSTEM SET 语句更改系统级参数的值，如下所示：

- SCOPE=MEMORY

  更改只应用于数据库实例。如果数据库关闭并重新启动，更改将不会保留。
- SCOPE=SPFILE

  更改被写入服务器参数文件，但不会影响当前实例。因此，所做的更改不会生效，直到重新启动该实例。

  注意：

  当你修改某个在《*Oracle 数据库参考*》中描述为不可修改的参数的值时，必须指定 SPFILE。
- SCOPE=BOTH

  将更改同时写入内存和服务器参数文件。当数据库使用服务器参数文件时，这是默认的范围。

  数据库会在警报日志中输出初始化参数的旧值和新值。作为一项预防措施，数据库会验证对基本参数的更改，以防止非法值被写入到服务器参数文件中。

另见：

- 《*Oracle 数据库管理员指南*》了解如何更改初始化参数设置
- 《*Oracle 数据库参考*》有关所有初始化参数的说明
- 《*Oracle 数据库 SQL 语言参考*》关于 ALTER SYSTEM 的语法和语

义

Overview of Diagnostic Files

Oracle Database includes a **fault diagnosability infrastructure** for preventing, detecting, diagnosing, and resolving database problems. Problems include critical errors such as code bugs, metadata corruption, and customer data corruption.

The goals of the advanced fault diagnosability infrastructure are the following:

- Detecting problems proactively
- Limiting damage and interruptions after a problem is detected
- Reducing problem diagnostic and resolution time
- Simplifying customer interaction with Oracle Support

诊断文件概述

Oracle 数据库包括一个**故障可诊断性基础设施**，以预防、 检测、 诊断、并解决各种数据库问题。这些问题包括如代码错误、 元数据损坏、和客户数据损坏等严重错误。

这个先进的故障可诊断性基础设施的目标如下：

- 主动侦测问题
- 在检测到问题后，限制破坏和中断
- 减少问题诊断和解决时间
- 简化客户与 Oracle 支持的交互

Automatic Diagnostic Repository

Automatic Diagnostic Repository (ADR) is a file-based repository that stores database diagnostic data such as trace files, the alert log, and Health Monitor reports. Key characteristics of ADR include:

- Unified directory structure
- Consistent diagnostic data formats
- Unified tool set

The preceding characteristics enable customers and Oracle Support to correlate and analyze diagnostic data across multiple Oracle instances, components, and products.

ADR is located *outside* the database, which enables Oracle Database to access and manage ADR when the physical database is unavailable. An instance can create ADR before a database has been created.

自动诊断存储库

自动诊断存储库 （ADR） 是一个基于文件的存储库，存储了如跟踪文件、警报日志、和健康监测报告等数据库诊断数据。ADR 的关键特征包括：

- 统一的路径结构
- 一致的诊断数据格式
- 统一的工具集

上述特性使得客户和 Oracle 支持部门能跨多个 Oracle 实例、 组件、和产品对诊断数据进行关联和分析。

ADR 处于数据库之外，这使得即便物理数据库不可用时，也可以访问和管理 ADR。在创建数据库之前，其实例就可以创建 ADR。

Problems and Incidents

ADR proactively tracks **problems**, which are critical errors in the database. Critical errors manifest as internal errors, such as `ORA-600`, or other severe errors. Each problem has a **problem key**, which is a text string that describes the problem.

When a problem occurs multiple times, ADR creates a time-stamped **incident** for each occurrence. An incident is uniquely identified by a numeric **incident ID**. When an incident occurs, ADR sends an **incident alert** to Enterprise Manager. Diagnosis and resolution of a critical error

问题和事件

ADR 主动跟踪数据库中的严重错误**问题**。严重错误通常显示为内部错误，如 `ORA-600` 或其他严重错误。每个问题有一个**问题键**，它是一个描述此问题的文本字符串。

当一个问题出现多次时， ADR 为每次发生的错误创建一个带时间戳的**事件**。事件由一个数字**事件 id** 唯一标识。当事件发生时， ADR 发送一个**事件警报**到企业管理器中。对严重错误的诊断和解决，通常从一个事件警报开始。

usually starts with an incident alert.

Because a problem could generate many incidents in a short time, ADR applies flood control to incident generation after certain thresholds are reached. A **flood-controlled incident** generates an alert log entry, but does not generate incident dumps. In this way, ADR informs you that a critical error is ongoing without overloading the system with diagnostic data.

See Also:

Oracle Database Administrator's Guide for detailed information about the fault diagnosability infrastructure

ADR Structure

The **ADR base** is the ADR root directory. The ADR base can contain multiple ADR homes, where each **ADR home** is the root directory for all diagnostic data—traces, dumps, the alert log, and so on—for an instance of an Oracle product or component. For example, in an Oracle RAC environment with shared storage and ASM, each database instance and each ASM instance has its own ADR home.

Figure 13-8 illustrates the ADR directory hierarchy for a database instance. Other ADR homes for other Oracle products or components, such as ASM or Oracle Net Services, can exist within this hierarchy, under the same ADR base.

Figure 13-8 ADR Directory Structure for an Oracle Database Instance

Description of "Figure 13-8 ADR Directory Structure for an Oracle

因为一个问题可以在短时间内生成许多事件，ADR 在达到某些阈值时，将对事件生成应用防洪控制措施。**防洪受控事件**将生成一个警报日志条目，但不会生成事件转储。这样一来，ADR 会通知您正在发生一个严重错误，而不会在系统中产生过量诊断数据。

另见：

《*Oracle 数据库管理员指南*》有关故障可诊断性基础设施的详细信息

ADR 结构

ADR base 是 ADR 的根目录。ADR base 可能包含多个 ADR home，每个 **ADR home** 是一个 Oracle 产品或组件实例的所有的诊断数据的根目录，包括跟踪、 转储、 和警报日志等等。例如，在有共享存储和 ASM 的 Oracle RAC 环境中，每个数据库实例和每个 ASM 实例都有自己的 ADR Home。

图 13-8 说明了数据库实例的 ADR 目录层次结构。其他 Oracle 产品或组件（如 ASM 或 Oracle 网络服务）的 ADR Home 也可以存在于此层次结构中，在相同的 ADR base 之下。

图 13-8 Oracle 数据库实例的 ADR 目录结构

Description of "Figure 13-8 ADR Directory Structure for an Oracle

Database Instance"

As the following Linux example shows, when you start an instance with a unique SID and database name *before* creating a database, Oracle Database creates ADR by default as a directory structure in the host file system. The SID and database name form part of the path name for files in the ADR Home.

Example 13-1 Creation of ADR

```
% setenv ORACLE_SID osi
% echo "DB_NAME=dbn" > init.ora
% sqlplus / as sysdba
.
.
.
Connected to an idle instance.

SQL> STARTUP NOMOUNT PFILE="./init.ora"
ORACLE instance started.

Total System Global Area  146472960 bytes
Fixed Size                  1317424 bytes
Variable Size              92276176 bytes
Database Buffers           50331648 bytes
Redo Buffers                2547712 bytes

SQL> SELECT NAME, VALUE FROM V$DIAG_INFO;

NAME                 VALUE
-------------------- ---------------------------------------------
Diag Enabled         TRUE
ADR Base             /u01/oracle/log
ADR Home             /u01/oracle/log/diag/rdbms/dbn/osi
Diag Trace           /u01/oracle/log/diag/rdbms/dbn/osi/trace
Diag Alert           /u01/oracle/log/diag/rdbms/dbn/osi/alert
Diag Incident        /u01/oracle/log/diag/rdbms/dbn/osi/incident
Diag Cdump           /u01/oracle/log/diag/rdbms/dbn/osi/cdump
Health Monitor       /u01/oracle/log/diag/rdbms/dbn/osi/hm
Default Trace File
```

Database Instance"

如下的 Linux 示例显示， 当你在创建一个数据库之前，使用一个唯一的 SID 和数据库名称启动一个实例时，Oracle 数据库在主机文件系统中会将 ADR 默认创建为一个目录结构。SID 和数据库名称形成 ADR Home 中的文件路径名称的一部分。

例 13-1 创建 ADR

```
% setenv ORACLE_SID osi
% echo "DB_NAME=dbn" > init.ora
% sqlplus / as sysdba
.
.
.
Connected to an idle instance.

SQL> STARTUP NOMOUNT PFILE="./init.ora"
ORACLE instance started.

Total System Global Area  146472960 bytes
Fixed Size                  1317424 bytes
Variable Size              92276176 bytes
Database Buffers           50331648 bytes
Redo Buffers                2547712 bytes

SQL> SELECT NAME, VALUE FROM V$DIAG_INFO;

NAME                 VALUE
-------------------- ---------------------------------------------
Diag Enabled         TRUE
ADR Base             /u01/oracle/log
ADR Home             /u01/oracle/log/diag/rdbms/dbn/osi
Diag Trace           /u01/oracle/log/diag/rdbms/dbn/osi/trace
Diag Alert           /u01/oracle/log/diag/rdbms/dbn/osi/alert
Diag Incident        /u01/oracle/log/diag/rdbms/dbn/osi/incident
Diag Cdump           /u01/oracle/log/diag/rdbms/dbn/osi/cdump
Health Monitor       /u01/oracle/log/diag/rdbms/dbn/osi/hm
Default Trace File
```

```
/u01/oracle/log/diag/rdbms/dbn/osi/trace/osi_ora_10533.trc
Active Problem Count  0
Active Incident Count 0
```

The following sections describe the contents of ADR.

Alert Log

Each database has an **alert log**, which is an XML file containing a chronological log of database messages and errors. The alert log contents include the following:

- All internal errors (`ORA-600`), block corruption errors (`ORA-1578`), and deadlock errors (`ORA-60`)
- Administrative operations such as DDL statements and the SQL*Plus commands `STARTUP`, `SHUTDOWN`, `ARCHIVE LOG`, and `RECOVER`
- Several messages and errors relating to the functions of shared server and dispatcher processes
- Errors during the automatic refresh of a materialized view

Oracle Database uses the alert log as an alternative to displaying information in the Enterprise Manager GUI. If an administrative operation is successful, then Oracle Database writes a message to the alert log as "completed" along with a time stamp.

Oracle Database creates an alert log in the `alert` subdirectory shown in Figure 13-8 when you first start a database instance, even if no database has been created yet. The following example shows a portion of a text-only alert log:

```
Fri Jun 19 17:05:34 2009
Starting ORACLE instance (normal)
LICENSE_MAX_SESSION = 0
LICENSE_SESSIONS_WARNING = 0
Shared memory segment for instance monitoring created
Picked latch-free SCN scheme 2
Autotune of undo retention is turned on.
IMODE=BR
ILAT =12
LICENSE_MAX_USERS = 0
SYS auditing is disabled
Starting up ORACLE RDBMS Version: 11.2.0.0.0.
```


```
/u01/oracle/log/diag/rdbms/dbn/osi/trace/osi_ora_10533.trc
Active Problem Count  0
Active Incident Count 0
```


以下各节描述了 ADR 的内容。

警报日志

每个数据库都有一个警报日志，它是一个 XML 文件，其中包含按时间排序的数据库信息和错误的日志。警报日志的内容包括如下：

- 所有的内部错误 (`ORA-600`)、 块损坏错误 （`ORA-1578`）、和死锁错误 (`ORA-60`)
- 管理性操作，如 DDL 语句和 `STARTUP`、`SHUTDOWN`、`ARCHIVE LOG`、和 `RECOVER` 等 SQL*Plus 命令
- 几个与共享服务器和调度器进程的功能有关的消息和错误
- 物化视图自动刷新过程中的错误

Oracle 数据库使用警报日志作为在企业管理器 GUI 中显示信息的一种替代。如果管理操作成功，则 Oracle 数据库向警报日志写入一条带有时间戳的“已完成”消息。

如图 13-8 所示， 当您首次启动一个数据库实例，即使其数据库尚未创建，Oracle 数据库会在 `alert` 子目录中创建一个警报日志。下面的示例演示一个纯文本警报日志的一部分：


```
Fri Jun 19 17:05:34 2009
Starting ORACLE instance (normal)
LICENSE_MAX_SESSION = 0
LICENSE_SESSIONS_WARNING = 0
Shared memory segment for instance monitoring created
Picked latch-free SCN scheme 2
Autotune of undo retention is turned on.
IMODE=BR
ILAT =12
LICENSE_MAX_USERS = 0
SYS auditing is disabled
Starting up ORACLE RDBMS Version: 11.2.0.0.0.
```

```
Using parameter settings in client-side pfile
.
.
.
System parameters with nondefault values:
  db_name                  = "my_test"
Fri Jun 19 17:05:37 2009
PMON started with pid=2, OS id=10329
Fri Jun 19 17:05:37 2009
VKTM started with pid=3, OS id=10331 at elevated priority
VKTM running at (20)ms precision
Fri Jun 19 17:05:37 2009
DIAG started with pid=4, OS id=10335
```

As shown in Example 13-1, query V$DIAG_INFO to locate the alert log.

Trace Files

A **trace file** is an administrative file that contain diagnostic data used to investigate problems. Also, trace files can provide guidance for tuning applications or an instance, as explained in "Performance Diagnostics and Tuning".

Types of Trace Files

Each server and background process can periodically write to an associated trace file. The files information on the process environment, status, activities, and errors.

The SQL trace facility also creates trace files, which provide performance information on individual SQL statements. To enable tracing for a client identifier, service, module, action, session, instance, or database, you must execute the appropriate procedures in the DBMS_MONITOR package or use Oracle Enterprise Manager.

A **dump** is a special type of trace file. Whereas a trace tends to be continuous output of diagnostic data, a dump is typically a one-time output of diagnostic data in response to an event (such as an incident). When an incident occurs, the database writes one or more dumps to the incident directory created for the incident. Incident dumps also contain the incident number in the file name.

See Also:

- "Session Control Statements"

```
Using parameter settings in client-side pfile
.
.
.
System parameters with nondefault values:
  db_name                  = "my_test"
Fri Jun 19 17:05:37 2009
PMON started with pid=2, OS id=10329
Fri Jun 19 17:05:37 2009
VKTM started with pid=3, OS id=10331 at elevated priority
VKTM running at (20)ms precision
Fri Jun 19 17:05:37 2009
DIAG started with pid=4, OS id=10335
```

如例 13-1 所示，通过查询 V$DIAG_INFO 来定位警报日志。

跟踪文件

跟踪文件是一个管理性文件，其中包含用于调查问题的诊断数据。此外，跟踪文件也可以为优化应用程序或实例提供指导，如"性能诊断与调整"所述。

跟踪文件的类型

每个服务器和后台进程可以定期写入一个关联的跟踪文件。这些文件包括进程环境、 状态、 活动、和错误等信息。

SQL 跟踪设施还会为单个 SQL 语句创建跟踪文件，以提供性能信息。若要为客户端标识、 服务、 模块、 动作、 会话、 实例、或数据库等启用跟踪，必须执行 DBMS_MONITOR 包中适当的过程，或使用 Oracle 企业管理器。

转储是一种特殊类型的跟踪文件。跟踪往往是诊断数据的连续输出，而转储通常是响应某个事件的一次性诊断数据输出。当事件发生时，数据库会将一个或多个转储写入为事件创建的目录。事件转储还在文件名称中包含了事件号。

另见：

- "会话控制语句"

- *Oracle Database Administrator's Guide* to learn about trace files, dumps, and core files
- *Oracle Database Performance Tuning Guide* to learn about application tracing

Locations of Trace Files

ADR stores trace files in the `trace` subdirectory, as shown in Figure 13-8. Trace file names are platform-dependent and use the extension `.trc`. Typically, database background process trace file names contain the Oracle SID, the background process name, and the operating system process number. An example of a trace file for the `RECO` process is `mytest_reco_10355.trc`.

Server process trace file names contain the Oracle SID, the string `ora`, and the operating system process number. An example of a server process trace file name is `mytest_ora_10304.trc`.

Sometimes trace files have corresponding trace map (`.trm`) files. These files contain structural information about trace files and are used for searching and navigation.

See Also:

Oracle Database Administrator's Guide to learn how to find trace files

- 《*Oracle 数据库管理员指南*》了解跟踪文件、 转储和核心文件
- 《*Oracle 数据库性能优化指南*》了解应用程序跟踪

跟踪文件的位置

ADR 将跟踪文件存储在 `trace` 子目录中，如图 13-8 所示。跟踪文件的名称取决于其运行平台，并使用扩展名.trc。

通常，数据库后台进程的跟踪文件名称中包含 Oracle SID、 后台进程名称、和操作系统进程号。`RECO` 进程的跟踪文件的一个示例是 mytest_reco_10355.trc。

服务器进程跟踪文件名称包含 Oracle SID、 字符串 `ora` 和操作系统进程号。服务器进程跟踪文件名称的一个示例是 mytest_ora_10304.trc。

有时跟踪文件有相应的跟踪位图 (.trm) 文件。这些文件包含跟踪文件的相关结构信息，并用于搜索和导航。

另见：

《*Oracle 数据库管理员指南*》了解如何查找跟踪文件

Part V

Oracle Instance Architecture

This part describes the basic structural architecture of the Oracle database instance. This part contains the following chapters:

- Chapter 13, "Oracle Database Instance"
- Chapter 14, "Memory Architecture"
- Chapter 15, "Process Architecture"
- Chapter 16, "Application and Networking Architecture"

Previous Next View PDF

第五部分

Oracle 实例体系结构

本部分介绍了 Oracle 数据库实例的基本体系结构。本部分包含以下各章：

- 第 13 章，"Oracle 数据库实例"
- 第 14 章，"内存体系结构"
- 第 15 章，"进程体系结构"
- 第 16 章，"应用程序和网络体系结构"

14 Memory Architecture

This chapter discusses the memory architecture of an Oracle Database instance.

This chapter contains the following sections:

- Introduction to Oracle Database Memory Structures
  - Basic Memory Structures
  - Oracle Database Memory Management
- Overview of the User Global Area
- Overview of the Program Global Area
  - Contents of the PGA
  - PGA Usage in Dedicated and Shared Server Modes
- Overview of the System Global Area
  - Database Buffer Cache
  - Redo Log Buffer
  - Shared Pool
  - Large Pool
  - Java Pool
  - Streams Pool
  - Fixed SGA
- Overview of Software Code Areas

See Also:

Oracle Database Administrator's Guide for instructions for configuring and managing memory

Introduction to Oracle Database Memory Structures

When an instance is started, Oracle Database allocates a memory area and starts background processes. The memory area stores information such as the following:

- Program code

第 14 章 内存架构

本章讨论 Oracle 数据库实例的内存架构。

本章包含以下各节：

- Oracle 数据库内存结构简介
  - 基本内存结构
  - Oracle 数据库内存管理
- 用户全局区概述
- 程序全局区概述
  - PGA 的内容
  - 在专用和共享服务器模式中使用 PGA
- 系统全局区概述
  - 数据库缓冲区高速缓存
  - 重做日志缓冲区
  - 共享池
  - 大池
  - Java 池
  - 流池
  - 固定 SGA
- 软件代码区概述

另见：

《*Oracle 数据库管理员指南*》关于配置和管理内存的说明

Oracle 数据库内存结构简介

当实例启动时， Oracle 数据库分配内存区，并启动后台进程。内存区存储如下信息：

- 程序代码

- Information about each connected session, even if it is not currently active
- Information needed during program execution, for example, the current state of a query from which rows are being fetched
- Information such as lock data that is shared and communicated among processes
- Cached data, such as data blocks and redo records, that also exists on disk

See Also:

Chapter 15, "Process Architecture"

Basic Memory Structures

The basic memory structures associated with Oracle Database include:

- System global area (SGA)

  The SGA is a group of shared memory structures, known as **SGA components**, that contain data and control information for one Oracle Database instance. The SGA is shared by all server and background processes. Examples of data stored in the SGA include cached data blocks and shared SQL areas.
- Program global area (PGA)

  A PGA is a nonshared memory region that contains data and control information exclusively for use by an Oracle process. The PGA is created by Oracle Database when an Oracle process is started.

  One PGA exists for each server process and background process. The collection of individual PGAs is the **total instance PGA**, or **instance PGA**. Database initialization parameters set the size of the instance PGA, not individual PGAs.
- User Global Area (UGA)

  The UGA is memory associated with a user session.
- Software code areas

  Software code areas are portions of memory used to store code that is being run or can be run. Oracle Database code is stored in a software area that is typically at a different location from user programs—a more exclusive or protected location.

Figure 14-1 illustrates the relationships among these memory structures.

- 有关每个已连接会话的信息，无论其当前是否处于活动状态
- 在程序执行期间所需的信息，例如，正在索取数据的查询的当前状态
- 在多个进程之间被共享和传递的信息，如锁信息
- 缓存数据，如数据块和重做记录等，它们也存在于磁盘上

另见：

第 15 章， "进程体系结构"

基本内存结构

与 Oracle 数据库相关联的基本内存结构包括：

- 系统全局区 （SGA）

  SGA 是一组称为 **SGA 组件**的共享内存结构，其中包含一个 Oracle 数据库实例的数据和控制信息。SGA 由所有服务器进程和后台进程共享。例如，SGA 中存储的数据包括数据块缓存和共享 SQL 区。
- 程序全局区 (PGA)

  PGA 是一个非共享的内存区域，其中包含专门供某个 Oracle 进程使用的数据和控制信息。Oracle 进程启动时， Oracle 数据库会为其创建 PGA。
  每个服务器进程和后台进程都存在一个 PGA。所有单个 PGA 的集合即是**总实例 PGA**，或**实例 PGA**。数据库初始化参数设置实例 PGA 的大小，而不是单个 PGA 的大小。
- 用户全局区 (UGA)

  UGA 是与某个用户会话相关联的内存区。
- 软件代码区

  软件代码区是用来存储正在运行或可能要运行的代码的那部分内存。Oracle 数据库代码通常存储在与用户程序不同的位置， ——一个更专门或更受保护的位置。

图 14-1 说明了这些内存结构之间的关系。

Figure 14-1 Oracle Database Memory Structures

Description of "Figure 14-1 Oracle Database Memory Structures"

Oracle Database Memory Management

Memory management involves maintaining optimal sizes for the Oracle instance memory structures as demands on the database change. Oracle Database manages memory based on the settings of memory-related initialization parameters. The basic options for memory management are as follows:

- Automatic memory management

  You specify the target size for instance memory. The database instance automatically tunes to the target memory size, redistributing memory as needed between the SGA and the

图 14-1 Oracle 数据库内存结构

Description of "Figure 14-1 Oracle Database Memory Structures"

Oracle 数据库内存管理

内存管理涉及维护 Oracle 实例内存结构的最优大小，以满足数据库的变更需求。Oracle 数据库基于与内存相关的初始化参数设置来管理内存。内存管理的基本选项如下所示：

- 自动内存管理

  您指定实例内存的目标大小。数据库实例自动优化到这个目标内存大小，根据需要在 SGA 和 PGA 实例之间重新分配内存。

instance PGA.

- Automatic shared memory management

  This management mode is partially automated. You set a target size for the SGA and then have the option of setting an aggregate target size for the PGA or managing PGA work areas individually.
- Manual memory management

  Instead of setting the total memory size, you set many initialization parameters to manage components of the SGA and instance PGA individually.

If you create a database with Database Configuration Assistant (DBCA) and choose the basic installation option, then automatic memory management is the default.

See Also:

- "Memory Management" for more information about memory management options for DBAs
- "Tools for Database Installation and Configuration" to learn about DBCA
- *Oracle Database 2 Day DBA* and *Oracle Database Administrator's Guide* to learn about memory management options

Overview of the User Global Area

The UGA is **session memory**, which is memory allocated for session variables, such as logon information, and other information required by a database session. Essentially, the UGA stores the session state. Figure 14-2 depicts the UGA.

Figure 14-2 User Global Area (UGA)

Description of "Figure 14-2 User Global Area (UGA)"

If a session loads a PL/SQL **package** into memory, then the UGA contains the **package state**, which is the set of values stored in all the package

- 自动共享内存管理

  这种管理模式是部分自动化的。您设置一个 SGA 的目标大小，然后设置 PGA 总目标大小，或单独管理 PGA 的各个工作区。

- 手工内存管理

  你不必设置总的内存大小，但您需要设置许多初始化参数，以单独管理 SGA 和 PGA 实例中的各个组件。

如果你使用数据库配置助手 （DBCA）来创建数据库，并选择了基本安装选项，则将缺省使用自动内存管理。

另见：

- "内存管理" 有关供数据库管理员使用的内存管理选项的详细信息
- 数据库安装和配置工具了解 DBCA
- 《*Oracle 数据库 2 日 DBA*》 和 《*Oracle 数据库管理员指南*》有关内存管理的选项

用户全局区概述

UGA 是为会话变量分配的**会话内存**，如登录信息和数据库会话所需的其他信息。实际上，UGA 存储了会话状态。图 14-2 描述了 UGA。

图 14-2 用户全局区 (UGA)

Description of "Figure 14-2 User Global Area (UGA)"

如果某个会话将 PL/SQL **包**加载到内存，则在 UGA 中包含**包状态**，即是所有包变量在某个特定的时刻所存储的值集（见"PL/SQL 包"）。当包的子程

variables at a specific time (see "PL/SQL Packages"). The package state changes when a package subprogram changes the variables. By default, the package variables are unique to and persist for the life of the session.
The **OLAP page pool** is also stored in the UGA. This pool manages OLAP data pages, which are equivalent to data blocks. The page pool is allocated at the start of an OLAP session and released at the end of the session. An OLAP session opens automatically whenever a user queries a dimensional object such as a cube.
The UGA must be available to a database session for the life of the session. For this reason, the UGA cannot be stored in the PGA when using a shared server connection because the PGA is specific to a single process. Therefore, the UGA is stored in the SGA when using shared server connections, enabling any shared server process access to it. When using a dedicated server connection, the UGA is stored in the PGA.

See Also:

- "Connections and Sessions"
- *Oracle Database Net Services Administrator's Guide* to learn about shared server connections
- *Oracle OLAP User's Guide* for an overview of Oracle OLAP

Overview of the Program Global Area

The PGA is memory specific to an operating process or thread that is not shared by other processes or threads on the system. Because the PGA is process-specific, it is never allocated in the SGA.
The PGA is a memory heap that contains session-dependent variables required by a dedicated or shared server process. The server process allocates memory structures that it requires in the PGA.
An analogy for a PGA is a temporary countertop workspace used by a file clerk. In this analogy, the file clerk is the server process doing work on behalf of the customer (client process). The clerk clears a section of the countertop, uses the workspace to store details about the customer request and to sort the folders requested by the customer, and then gives up the space when the work is done.
Figure 14-3 shows an instance PGA (collection of all PGAs) for an instance that is not configured for shared servers. You can use an initialization parameter to set a target maximum size of the instance PGA (see "Summary of Memory Management Methods"). Individual PGAs can grow

序更改变量的值时，包状态也将更改。默认情况下，包变量在会话的存活期间是唯一且持久的。

OLAP 页面缓冲池也存储在 UGA 中。该池管理相当于数据块的 OLAP 数据页面。页面缓冲池在启动一个 OLAP 会话时分配，并在该会话结束时释放。每当用户查询一个多维对象（如立方体）时，就会自动打开一个 OLAP 会话。

该 UGA 必须在数据库会话的整个存活期间是可用的。由于这个原因，当使用共享服务器的连接时，UGA 不能存储在 PGA 中，因为 PGA 是特定于单个进程的。因此，当使用共享服务器的连接时， UGA 被存储在 SGA 中，以使任何共享服务器进程都能访问它。在使用专用服务器的连接时，UGA 存储在 PGA 中。

另见：

- "连接和会话"
- 《*Oracle 数据库网络服务管理员指南*》了解共享服务器连接
- 《*Oracle OLAP 用户指南*》关于 Oracle OLAP 的概述

程序全局区概述

PGA 是特定于一个操作系统进程或线程的内存区，且不和系统上的其他进程或线程共享。由于 PGA 是特定于进程的，所以它决不会在 SGA 中分配。

PGA 是包含某个专用或共享服务器进程所需的会话变量的内存堆。服务器进程在需要时会在 PGA 中分配内存结构。

PGA 好比是文员所使用的临时工作台面。在这个比喻中，文员是为客户（客户端进程）服务的服务器进程。文员清理出台面的一部分，使用这个工作空间来存储有关客户要求的详细信息，并对顾客请求的文件夹排序，然后在完成工作时让出工作空间。

图 14-3 显示某个未配置为共享服务器的实例的 PGA（所有 PGA 的集合）。您可以使用一个初始化参数设置实例 PGA 的目标最大大小 （请参见"内存管理方法总结"）。根据需要，各个 PGA 可以按需增大到这个目标大小。

as needed up to this target size.

Figure 14-3 Instance PGA

Description of "Figure 14-3 Instance PGA"

Note:

Background processes also allocate their own PGAs. This discussion focuses on server process PGAs only.

Contents of the PGA

The PGA is subdivided into different areas, each with a different purpose. Figure 14-4 shows the possible contents of the PGA for a dedicated server session. Not all of the PGA areas will exist in every case.

Figure 14-4 PGA Contents

Description of "Figure 14-4 PGA Contents"

Private SQL Area

A **private SQL area** holds information about a parsed SQL statement and other session-specific information for processing. When a server process executes SQL or PL/SQL code, the process uses the private SQL area to store bind variable values, query execution state information, and query execution work areas.

Do not confuse a *private* SQL area, which is in the UGA, with the *shared*

图 14-3 实例 PGA

Description of "Figure 14-3 Instance PGA"

注意：

后台进程也分配它们自己的 PGA。本讨论的重点仅限于服务器进程 PGA。

PGA 的内容

PGA 被进一步细分为多个不同区域，每一个都有不同的目的。图 14-4 显示一个专用服务器会话的 PGA 中可能包含的内容。不是所有的 PGA 区域在任何情况下都存在。

图 14-4 PGA 内容

Description of "Figure 14-4 PGA Contents"

私有 SQL 区

私有 SQL 区保存了有关某个已解析的 SQL 语句的信息，和其他特定于会话的信息。当服务器进程执行 SQL 或 PL/SQL 代码时，该过程使用其私有 SQL 区域，来存储绑定变量值、 查询执行状态信息、和查询执行工作区。

不要混淆在 UGA 中的*私有* SQL 区，和在 SGA 中存储执行计划的*共享* SQL

SQL area, which stores execution plans in the SGA. Multiple private SQL areas in the same or different sessions can point to a single execution plan in the SGA. For example, 20 executions of `SELECT * FROM employees` in one session and 10 executions of the same query in a different session can share the same plan. The private SQL areas for each execution are not shared and may contain different values and data.
A **cursor** is a name or handle to a specific private SQL area. As shown in Figure 14-5, you can think of a cursor as a pointer on the client side and as a state on the server side. Because cursors are closely associated with private SQL areas, the terms are sometimes used interchangeably.

Figure 14-5 Cursor

Description of "Figure 14-5 Cursor"

A private SQL area is divided into the following areas:

- The run-time area

  This area contains query execution state information. For example, the run-time area tracks the number of rows retrieved so far in a full table scan.
  Oracle Database creates the run-time area as the first step of an execute request. For DML statements, the run-time area is freed when the SQL statement is closed.
- The persistent area

  This area contains **bind variable** values. A bind variable value is supplied to a SQL statement at run time when the statement is executed. The persistent area is freed only when the cursor is closed.

The client process is responsible for managing private SQL areas. The allocation and deallocation of private SQL areas depends largely on the application, although the number of private SQL areas that a client

区。在相同或不同的会话中的多个私有 SQL 区，可能指向 SGA 中的一个单一执行计划。例如，在某个会话中运行“`SELECT * FROM employees`” 20 次，而在另一个不同的会话中运行同一查询 10 次，但它们可以共享相同的执行计划。每次运行的私有 SQL 区并不共享，因此可能包含不同的值和数据。

游标是指向某个特定的私有 SQL 区的一个名称或句柄。如图 14-5 所示，你可以将游标看成是一个从客户端指向服务器端状态信息的指针）。游标与私有 SQL 区密切相关，这两个术语有时可以互换使用。

图 14-5 光标

Description of "Figure 14-5 Cursor"

私有 SQL 区又分为以下几个区域：

- 运行时区域

  此区域包含查询执行状态信息。例如，运行时区域会跟踪到目前为止在全表扫描中检索到的行数。

  Oracle 数据库将创建运行时区域，作为一个执行请求的第一步。对于 DML 语句，其运行时区域将在 SQL 语句关闭时被释放。

- 持久区域

  此区域包含**绑定变量**的值。绑定变量是执行 SQL 语句时，在运行时提供给 SQL 语句的值。仅当关闭该游标时，持久区域才被释放。

客户端进程负责管理私有 SQL 区。虽然客户端进程可以分配的私有 SQL 区数量由初始化参数 OPEN_CURSORS 限制，但私有 SQL 区的分配和释放主

process can allocate is limited by the initialization parameter `OPEN_CURSORS`.

Although most users rely on the automatic cursor handling of database utilities, the Oracle Database programmatic interfaces offer developers more control over cursors. In general, applications should close all open cursors that will not be used again to free the persistent area and to minimize the memory required for application users.

See Also:

- "Shared SQL Areas"
- *Oracle Database Advanced Application Developer's Guide* and *Oracle Database PL/SQL Language Reference* to learn how to use cursors

SQL Work Areas

A **work area** is a private allocation of PGA memory used for memory-intensive operations. For example, a sort operator uses the **sort area** to sort a set of rows. Similarly, a hash join operator uses a **hash area** to build a hash table from its left input, whereas a bitmap merge uses the **bitmap merge area** to merge data retrieves from scans of multiple bitmap indexes.

Example 14-1 shows a join of `employees` and `departments` with its query plan.

Example 14-1 Query Plan for Table Join

```
SQL> SELECT *
  2  FROM   employees e JOIN departments d
  3  ON     e.department_id=d.department_id
  4  ORDER BY last_name;
.
.
.
-----------------------------------------------------------------------------------
| Id  | Operation           | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |             |   106 |  9328 |    7  (29)| 00:00:01 |
|   1 |  SORT ORDER BY      |             |   106 |  9328 |    7  (29)| 00:00:01 |
|*  2 |   HASH JOIN         |             |   106 |  9328 |    6  (17)| 00:00:01 |
|   3 |    TABLE ACCESS FULL| DEPARTMENTS |    27 |   540 |    2   (0)| 00:00:01 |
|   4 |    TABLE ACCESS FULL| EMPLOYEES   |   107 |  7276 |    3   (0)| 00:00:01 |
-----------------------------------------------------------------------------------
```

要取决于应用程序。

虽然大多数用户依靠数据库实用程序的游标自动处理机制，但 Oracle 数据库编程接口仍为开发人员提供了对游标更多的控制。一般情况下，应用程序应关闭所有打开的且不再使用的游标，以释放持久区域，并最小化应用程序用户的内存需求。

另见：

- "共享 SQL 区"
- *《Oracle 数据库高级应用程序开发者指南》*和 *《Oracle 数据库 PL/SQL 语言参考》*了解如何使用游标

SQL 工作区

工作区是在 PGA 中为内存密集型操作分配的私有内存区。例如，排序操作使用**排序区**来对一组行进行排序。同样，哈希联接操作将其左侧数据为输入，并使用**哈希区**来创建一个哈希表，而位图合并操作则使用**位图合并区**来合并从扫描多个位图索引检索到的数据。

示例 14-1 显示了 `employees` 表和 `departments` 表相联接的查询计划。

示例 14-1 表联接的查询计划


```
SQL> SELECT *
  2  FROM   employees e JOIN departments d
  3  ON     e.department_id=d.department_id
  4  ORDER BY last_name;
.
.
.
-----------------------------------------------------------------------------------
| Id  | Operation           | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |             |   106 |  9328 |    7  (29)| 00:00:01 |
|   1 |  SORT ORDER BY      |             |   106 |  9328 |    7  (29)| 00:00:01 |
|*  2 |   HASH JOIN         |             |   106 |  9328 |    6  (17)| 00:00:01 |
|   3 |    TABLE ACCESS FULL| DEPARTMENTS |    27 |   540 |    2   (0)| 00:00:01 |
|   4 |    TABLE ACCESS FULL| EMPLOYEES   |   107 |  7276 |    3   (0)| 00:00:01 |
-----------------------------------------------------------------------------------
```

In Example 14-1, the run-time area tracks the progress of the full table scans. The session performs a hash join in the hash area to match rows from the two tables. The ORDER BY sort occurs in the sort area.

If the amount of data to be processed by the operators does not fit into a work area, then Oracle Database divides the input data into smaller pieces. In this way, the database processes some data pieces in memory while writing the rest to temporary disk storage for processing later.

The database automatically tunes work area sizes when automatic PGA memory management is enabled. You can also manually control and tune the size of a work area. See "Memory Management" for more information.

Generally, larger work areas can significantly improve performance of an operator at the cost of higher memory consumption. Optimally, the size of a work area is sufficient to accommodate the input data and auxiliary memory structures allocated by its associated SQL operator. If not, response time increases because part of the input data must be cached on disk. In the extreme case, if the size of a work area is too small compared to input data size, then the database must perform multiple passes over the data pieces, dramatically increasing response time.

See Also:

- *Oracle Database Administrator's Guide* to learn how to use automatic PGA management
- *Oracle Database Performance Tuning Guide* to learn how to tune PGA memory

PGA Usage in Dedicated and Shared Server Modes

PGA memory allocation depends on whether the database uses dedicated or shared server connections. Table 14-1 shows the differences.

Table 14-1 Differences in Memory Allocation Between Dedicated and Shared Servers

Memory Area	Dedicated Server	Shared Server
Nature of session memory	Private	Shared
Location of the persistent area	PGA	SGA
Location of the run-time area for DML/DDL statements	PGA	PGA

See Also:

Oracle Database Net Services Administrator's Guide to learn how to

在示例 14-1 中，运行时区域跟踪全表扫描的进度。此会话在哈希区中执行一个哈希联接，以匹配两个表中的行。而 ORDER BY 排序操作在排序区中进行。

如果要由该运算符处理的数据量太大，不能在工作区中完成，则 Oracle 数据库会将输入数据分成更小的片断。这样一来，数据库先在内存中处理一些数据片断，而将其余数据写入到临时磁盘存储，以待后续处理。

当启用了自动 PGA 内存管理时，数据库自动优化工作区大小。你也可以手动控制和优化工作区大小。更多的信息，请参阅"内存管理"。

通常，更大的工作区可以显著提高操作性能，但这是以更多的内存消耗为代价的。理想的情况是，工作区的大小足以容纳输入的数据，而由其关联的 SQL 运算符分配辅助内存结构。否则，响应时间会增加，因为输入数据的一部分必须被缓存到磁盘上。在极端情况下，如果工作区的大小相比输入数据的大小显得过小，则数据库必须来回多次倒腾这些数据片断，极大地增大了响应时间。

另见：

- 《*Oracle 数据库管理员指南*》了解如何使用自动 PGA 管理
- 《*Oracle 数据库性能优化指南*》了解如何调整 PGA 内存

在专用和共享服务器模式中使用 PGA

PGA 内存分配取决于数据库是使用专用的还是共享的服务器连接。表 14-1 显示了差异之处。

表 14-1 专用和共享服务器之间的内存分配差异

内存区	专用服务器	共享服务器
会话内存的性质	私有的	共享的
持久区域的位置	PGA	SGA
DML/DDL 语句的运行时区域的位置	PGA	PGA

另见：

《*Oracle 数据库网络服务管理员指南*》了解如何将数据库配置为共享服务器

configure a database for shared server

Overview of the System Global Area

The **SGA** is a read/write memory area that, along with the Oracle background processes, make up a database instance. All server processes that execute on behalf of users can read information in the instance SGA. Several processes write to the SGA during database operation.

Note:

The server and background processes do not reside *within* the SGA, but exist in a separate memory space.

Each database instance has its own SGA. Oracle Database automatically allocates memory for an SGA at instance startup and reclaims the memory at instance shutdown. When you start an instance with SQL*Plus or Oracle Enterprise Manager, the size of the SGA is shown as in the following example:

```
SQL> STARTUP
ORACLE instance started.

Total System Global Area  368283648 bytes
Fixed Size                  1300440 bytes
Variable Size             343935016 bytes
Database Buffers           16777216 bytes
Redo Buffers                6270976 bytes
Database mounted.
Database opened.
```

As shown in Figure 14-1, the SGA consists of several **memory components**, which are pools of memory used to satisfy a particular class of memory allocation requests. All SGA components except the redo log buffer allocate and deallocate space in units of contiguous memory called **granules**. Granule size is platform-specific and is determined by total SGA size.

You can query the `V$SGASTAT` view for information about SGA components.

The most important SGA components are the following:

系统全局区概述

SGA 是一个可读写的内存区，与 Oracle 后台进程一起组成数据库实例。所有以用户名义执行的服务器进程，都可以读取 SGA 实例中的信息。在数据库操作过程中，有几个进程会对 SGA 进行写入操作。

注意：

服务器和后台进程并不驻留在 SGA 中，而是存在于单独的内存空间。

每个数据库实例都有其自己的 SGA。当实例启动时 Oracle 数据库自动为其 SGA 分配内存，并在该实例关闭时回收内存。当您使用 SQL *Plus 或 Oracle 企业管理器启动实例时，会显示 SGA 的大小，如下示例所示：

```
SQL> STARTUP
ORACLE instance started.

Total System Global Area  368283648 bytes
Fixed Size                  1300440 bytes
Variable Size             343935016 bytes
Database Buffers           16777216 bytes
Redo Buffers                6270976 bytes
Database mounted.
Database opened.
```

如图 14-1 所示， SGA 由几个**内存组件**组成，它们是用来满足特定类别的内存分配请求的内存池。除重做日志缓冲区外，所有 SGA 组件按称为**颗粒**的连续内存单元来分配和释放空间。颗粒大小是特定于平台的，由 SGA 的总大小决定。

您可以查询 V$ SGASTAT 视图，来查看有关 SGA 组件的信息。

最重要的 SGA 组件如下所示：

- Database Buffer Cache
- Redo Log Buffer
- Shared Pool
- Large Pool
- Java Pool
- Streams Pool
- Fixed SGA

See Also:

- "Introduction to the Oracle Database Instance"
- *Oracle Database Performance Tuning Guide* to learn more about granule sizing

Database Buffer Cache

The **database buffer cache**, also called the **buffer cache**, is the memory area that stores copies of data blocks read from data files. A **buffer** is a main memory address in which the buffer manager temporarily caches a currently or recently used data block. All users concurrently connected to a database instance share access to the buffer cache.

Oracle Database uses the buffer cache to achieve the following goals:

- Optimize physical I/O

  The database updates data blocks in the cache and stores metadata about the changes in the redo log buffer. After a `COMMIT`, the database writes the redo buffers to disk but does not immediately write data blocks to disk. Instead, database writer (DBW*n*) performs **lazy writes** in the background.
- Keep frequently accessed blocks in the buffer cache and write infrequently accessed blocks to disk

  When **Database Smart Flash Cache (flash cache)** is enabled, part of the buffer cache can reside in the flash cache. This buffer cache extension is stored on a **flash disk device**, which is a solid state storage device that uses flash memory. The database can improve performance by caching buffers in flash memory instead of reading from magnetic disk.

  Note:

- 数据库缓冲区高速缓存
- 重做日志缓冲区
- 共享池
- 大池
- Java 池
- 流池
- 固定 SGA

另见：

- "Oracle 数据库实例简介"
- 《*Oracle 数据库性能优化指南*》了解如何确定颗粒的尺寸

数据库缓冲区高速缓存

数据库缓冲区高速缓存，也称为**缓冲区高速缓存**，是用于存储从数据文件读取的数据块副本的内存区域。缓冲区是缓冲区管理器用来暂时缓存当前或最近使用的数据块的主内存地址。所有同时都连接到一个数据库实例的用户，以共享方式访问缓冲区高速缓存。

Oracle 数据库使用缓冲区高速缓存，来实现以下目标：

- 优化物理 I/O

  数据库更新缓存中的数据块，并将有关更改的元数据存储在重做日志缓冲区。提交之后，数据库将重做缓冲区写入磁盘，但不一定会立即将数据块写入磁盘。相反，数据库写入器 (DBWn) 在后台执行**惰性写入**操作。

- 将频繁访问的块保持在缓冲区高速缓存中，而将不常存取的块写到磁盘

  当启用了**数据库智能闪存高速缓存** （**flash 高速缓存**）时， 缓冲区高速缓存的一部分可能驻留在闪存缓存中。此缓冲区高速缓存扩展存储在**闪存磁盘设备**上，这是一种使用闪存的固态存储设备。通过将缓冲区缓存到闪存中，而不是从磁盘读取，数据库可以提高性能。

  注意：

Database Smart Flash Cache is available only in Solaris and Oracle Enterprise Linux.

Buffer States

The database uses internal algorithms to manage buffers in the cache. A buffer can be in any of the following mutually exclusive states:

- Unused

  The buffer is available for use because it has never been used or is currently unused. This type of buffer is the easiest for the database to use.
- Clean

  This buffer was used earlier and now contains a read-consistent version of a block as of a point in time. The block contains data but is "clean" so it does not need to be checkpointed. The database can pin the block and reuse it.
- Dirty

  The buffer contain modified data that has not yet been written to disk. The database must checkpoint the block before reusing it.

Every buffer has an access mode: **pinned** or **free** (unpinned). A buffer is "pinned" in the cache so that it does not age out of memory while a user session accesses it. Multiple sessions cannot modify a pinned buffer at the same time.

The database uses a sophisticated algorithm to make buffer access efficient. Pointers to dirty and nondirty buffers exist on the same **least recently used (LRU) list**, which has a hot end and cold end. A **cold buffer** is one that has not been recently used. A **hot buffer** is frequently accessed and has been recently used.

Note:

Conceptually, there is only one LRU, but for concurrency the database actually uses several LRUs.

Buffer Modes

When a client requests data, Oracle Database retrieves buffers from the database buffer cache in either of the following modes:

- Current mode

  A **current mode get**, also called a **db block get**, is a retrieval of

数据库智能闪存高速缓存仅在 Solaris 和 Oracle 企业 Linux 中可用。

缓冲区状态

数据库使用内部算法来管理高速缓存中的缓冲区。缓冲区可以处于下列互斥状态之一：

- 未使用的

  缓冲区可供使用，因为它从未使用过，或者当前未使用。这种类型的缓冲区是数据库最容易使用的。

- 干净的

  此缓冲区在之前曾被使用过，而现在包含某个数据块在某个时间点的读取一致版本。块包含数据但是干净的，因此它不需要将执行检查点操作。数据库可以订住该块并重用它。

- 脏的

  缓冲区包含已修改、但尚未写入到磁盘的数据。数据库在重用该数据块之前必须对其执行检查点操作。

每个缓冲区具有两种访问模式之一： **订住的**或**空闲的** (未订住)。缓冲区被"**订住**"在缓存中，以便当其被某个用户会话访问时，它不会因为内存不足被换出内存。多个会话不能在同一时间修改某个已被订住的缓冲区。

数据库使用复杂的算法以使缓冲区访问高效。**最近最少使用 （LRU）** 列表有一个热端和冷端，同时存在指向同一 LRU 上的脏缓冲区和非脏缓冲区的指针。冷缓冲区是最近未被使用的。热缓冲区是被频繁访问并在最近已使用的。

注意：

从概念上讲，只有一个 LRU， 但为了并发，数据库实际上使用多个 LRU。

缓冲区模式

当一个客户端请求数据时， Oracle 数据库以下列两种模式之一从数据库缓冲区高速缓存中检索缓冲区：

- 当前模式

  当前模式获取，也称为**数据库块获取**，这是一种对当前已出现在缓冲

a block as it currently appears in the buffer cache. For example, if an uncommitted transaction has updated two rows in a block, then a current mode get retrieves the block with these uncommitted rows. The database uses db block gets most frequently during modification statements, which must update only the current version of the block.

- Consistent mode

  A **consistent read get** is a retrieval of a read-consistent version of a block. This retrieval may use undo data. For example, if an uncommitted transaction has updated two rows in a block, and if a query in a separate session requests the block, then the database uses undo data to create a read-consistent version of this block (called a **consistent read clone**) that does not include the uncommitted updates. Typically, a query retrieves blocks in consistent mode.

See Also:

- "Read Consistency and Undo Segments"
- *Oracle Database Reference* for descriptions of database statistics such as db block get and consistent read get

区高速缓存中的块的检索。例如，如果一个未提交事务已更新某个块中的两行，则当前模式获取会检索这个具有未提交行的块。数据库最常使用数据库块获取的情况是在修改语句期间，它只需更新块的当前版本。

- 一致模式

  一致读取获取是对某个块的一致读取版本的检索。此检索可能会使用撤消数据。例如，如果一个未提交事务已更新某个块中的两行，而在另一个独立会话中的查询请求该块，则数据库使用撤消数据来创建该块的一个读取一致版本（称为**一致读取克隆**），它不包括未提交的更新。通常，查询以一致模式检索块。

另见：

- "读取一致性和撤消段"
- 《*Oracle 数据库参考*》关于数据库统计信息的描述，例如，数据库块获取和一致读取获取

Buffer I/O

A **logical I/O**, also known as a **buffer I/O**, refers to reads and writes of buffers in the buffer cache. When a requested buffer is not found in memory, the database performs a **physical I/O** to copy the buffer from either the flash cache or disk into memory, and then a logical I/O to read the cached buffer.

Buffer Writes

The database writer (DBW*n*) process periodically writes cold, dirty buffers to disk. DBW*n* writes buffers in the following circumstances:

- A server process cannot find clean buffers for reading new blocks into the database buffer cache.

  As buffers are dirtied, the number of free buffers decreases. If the number drops below an internal threshold, and if clean buffers are required, then server processes signal DBW*n* to write.

  The database uses the LRU to determine which dirty buffers to write. When dirty buffers reach the cold end of the LRU, the

缓冲区 I/O

逻辑 I/O，也称为**缓冲区 I/O**，指的是读取和写入缓冲区高速缓存中的缓冲区。当在内存中找不到请求的缓冲区时，数据库将执行一个物理 I/O，将缓冲区从闪存缓存或磁盘复制到内存，然后再执行一个逻辑 I/O，以读取缓存的缓冲区。

缓冲区写出

数据库写入器 (DBWn) 进程定期将冷的、 脏的缓冲区写入磁盘。DBWn 在以下情况下会将缓冲区写出：

- 服务器进程找不到干净的缓冲区，以将新块读入数据库缓冲区高速缓存。

  随着缓冲区变脏，可用缓冲区的数量就会减少。如果该数值低于一个内部阈值，而又需要干净的缓冲区，则服务器进程将通知 DBWn 执行写出操作。

  数据库使用 LRU 来确定哪些脏缓冲区被写出。当脏缓冲区到达 LRU 的冷端时，数据库将其从 LRU 上移出到**写出队列**。DBWn 将队列中

database moves them off the LRU to a **write queue**. DBW*n* writes buffers in the queue to disk, using multiblock writes if possible. This mechanism prevents the end of the LRU from becoming clogged with dirty buffers and allows clean buffers to be found for reuse.

- The database must advance the checkpoint, which is the position in the redo thread from which instance recovery must begin.
- Tablespaces are changed to read-only status or taken offline.

See Also:

- "Database Writer Process (DBW***n***)"
- *Oracle Database Performance Tuning Guide* to learn how to diagnose and tune buffer write issues

Buffer Reads

When the number of clean or unused buffers is low, the database must remove buffers from the buffer cache. The algorithm depends on whether the flash cache is enabled:

- Flash cache disabled

  The database re-uses each clean buffer as needed, overwriting it. If the overwritten buffer is needed later, then the database must read it from magnetic disk.
- Flash cache enabled

  DBW*n* can write the body of a clean buffer to the flash cache, enabling reuse of its in-memory buffer. The database keeps the buffer header in an LRU list in main memory to track the state and location of the buffer body in the flash cache. If this buffer is needed later, then the database can read it from the flash cache instead of from magnetic disk.

When a client process requests a buffer, the server process searches the buffer cache for the buffer. A **cache hit** occurs if the database finds the buffer in memory. The search order is as follows:

✧ 1. The server process searches for the whole buffer in the buffer cache.
If the process finds the whole buffer, then the database performs a **logical read** of this buffer.

的缓冲区写到磁盘，如果可能使用多块写，则用之。此机制可以防止 LRU 尾端被脏缓冲区塞满，并保证有干净的缓冲区可供重用。

- 数据库必须推进检查点，即重做线程中进行实例恢复的起点。
- 表空间被更改为只读状态，或脱机。

另见：

- 数据库写入器进程（DBWn）"
- 《*Oracle 数据库性能优化指南*》了解如何诊断和调整缓冲区写出的问题

缓冲区读取

当干净的或未使用的缓冲区的数量变得很少时，数据库就必须将某些缓冲区从缓冲区高速缓存删除。其算法取决于是否启用了闪存缓存：

- 禁用了闪存高速缓存

  数据库中根据需要覆盖并重新利用每个干净的缓冲区。如果以后需要重用被覆盖的缓冲区，则数据库必须从磁盘重新读取。
- 启用了闪存高速缓存

  DBWn 将干净缓冲区的正文写入到闪存缓存，使其内存中的缓冲区可以被重用。数据库在主内存中保留 LRU 列表的缓冲区头，以跟踪在闪存缓存中的缓冲区体的状态和位置。如果以后需要该缓冲区，则数据库可以从闪存缓存中读取它，而不用从磁盘读取。

当客户端进程请求一个缓冲区时，服务器进程在缓冲区高速缓存中搜索缓冲区。如果数据库在内存中找到缓冲区，则发生**缓存命中**。搜索顺序如下所示：

✧ 1. 服务器进程在缓冲区高速缓存中搜索整个缓冲区。

如果该进程发现整个缓冲区，则数据库对此缓冲区执行一个**逻辑读取**。

- 2. The server process searches for the buffer header in the flash cache LRU list.
  If the process finds the buffer header, then the database performs an **optimized physical read** of the buffer body from the flash cache into the in-memory cache.
- 3. If the process does *not* find the buffer in memory (a **cache miss**), then the server process performs the following steps:
  - a. Copies the block from a data file into memory (a **physical read**)
  - b. Performs a logical read of the buffer that was read into memory

Figure 14-6 illustrates the buffer search order. The extended buffer cache includes both the in-memory buffer cache, which contains whole buffers, and the flash cache, which contains buffer bodies. In the figure, the database searches for a buffer in the buffer cache and, not finding the buffer, reads it into memory from magnetic disk.

Figure 14-6 Buffer Search

Description of "Figure 14-6 Buffer Search"

- 2. 服务器进程在闪存缓存 LRU 列表中搜索缓冲区头。

  如果该进程找到了缓冲区头，则数据库执行一个**优化的物理读取**，将缓冲区正文从闪存缓存读入到内存内缓存。

- 3. 如果该进程没有在内存中找到该缓冲区（**缓存未命中**），则服务器进程将执行以下步骤：
  - a. 将该块从数据文件复制到内存中 （**物理读取**）
  - b. 对已读入到内存中的缓冲区执行一个逻辑读取

图 14-6 说明了缓冲区的搜索顺序。扩展的缓冲区高速缓存包括包含整个缓冲区的内存内缓冲区缓存，和包含缓冲区正文的闪存缓存。在图中，数据库在缓冲区高速缓存中搜索某个缓冲区，但未能找到，因此将其从磁盘中读入到内存。

图 14-6 缓冲区搜索

Description of "Figure 14-6 Buffer Search"

In general, accessing data through a cache hit is faster than through a cache miss. The **buffer cache hit ratio** measures how often the database found a requested block in the buffer cache without needing to read it from disk.

The database can perform physical reads from either a data file or a temp file. Reads from a data file are followed by logical I/Os. Reads from a temp file occur when insufficient memory forces the database write data to a temporary table and read it back later. These physical reads bypass the buffer cache and do not incur a logical I/O.

See Also:

Oracle Database Performance Tuning Guide to learn how to calculate the buffer cache hit ratio

Buffer Touch Counts

The database measures the frequency of access of buffers on the LRU list using a **touch count**. This mechanism enables the database to increment a counter when a buffer is pinned instead of constantly shuffling buffers on the LRU list.

Note:

The database does not physically move blocks in memory. The movement is the change in location of a pointer on a list.

When a buffer is pinned, the database determines when its touch count was last incremented. If the count was incremented over three seconds ago, then the count is incremented; otherwise, the count stays the same. The three-second rule prevents a burst of pins on a buffer counting as many touches. For example, a session may insert several rows in a data block, but the database considers these inserts as one touch.

If a buffer is on the cold end of the LRU, but its touch count is high, then the buffer moves to the hot end. If the touch count is low, then the buffer ages out of the cache.

Buffers and Full Table Scans

When buffers must be read from disk, the database inserts the buffers into the middle of the LRU list. In this way, hot blocks can remain in the cache so that they do not need to be read from disk again.

A problem is posed by a full table scan, which sequentially reads all rows under the table high water mark (see "Segment Space and the High Water Mark"). Suppose that the total size of the blocks in a table segment is greater than the size of the buffer cache. A full scan of this table could clean out the buffer cache, preventing the database from maintaining a cache of frequently accessed blocks.

一般情况下，通过缓存命中访问数据比通过缓存未命中要快。**缓冲区高速缓存命中率**用于测量数据库在缓冲区高速缓存中找到请求的块，而不需要从磁盘中读取的比率。

数据库可能从数据文件或临时文件执行物理读取。在数据文件读取之后，会紧跟着有逻辑 I/O。临时文件读取发生在当内存不足时，迫使数据库将数据写入一个临时表，稍后又将其读回。这种物理读取会绕过缓冲区高速缓存，并不会导致逻辑的 I/O。

另见：

《*Oracle 数据库性能优化指南*》了解如何计算缓冲区高速缓存命中率

缓冲区触摸计数

数据库使用**触摸计数**来测量对 LRU 列表上的缓冲区进行访问的频率。这种机制使得当某个缓冲区被订住时数据库可以增加一次计数，而不用不断地移动 LRU 列表上的缓冲区。

注意：

数据库不会物理地移动内存中的块。此移动只是改变列表上指针的位置。

当缓冲区被订住时，数据库将确定其触摸计数的最后一次递增是何时发生的。如果计数递增发生在超过三秒钟前，则计数将增加 1；否则，计数保持不变。三秒钟规则防止对该缓冲区的一系列突发订住会产生过多计数。例如，一个会话可能会在一个数据块中插入几行，但数据库将这些插入作为一个触摸看待。

如果某个缓冲区是在 LRU 的冷端上，但其触摸计数很高，则该缓冲区将移动到热端。如果触摸计数很低，则缓冲区将从缓存中老化移出。

缓冲区和全表扫描

当缓冲区必须从磁盘读入时，数据库会将缓冲区插入到 LRU 列表的中部。通过这种方式，热块可以保留在缓存中，以使他们不需要再次从磁盘读取。

全表扫描顺序读取表高水位（请参阅"段空间和高水位"）下的所有行，这可能引发一个问题。假设表段中块的总大小大于缓冲区高速缓存的大小。在此表上的全表扫描可能会清除高速缓存，致使数据库不能将频繁访问的块维持在高速缓存中。

Blocks read into the database cache as the result of a full scan of a large table are treated differently from other types of reads. The blocks are immediately available for reuse to prevent the scan from effectively cleaning out the buffer cache.

In the rare case where the default behavior is not desired, you can change the CACHE attribute of the table. In this case, the database does not force or pin the blocks in the buffer cache, but ages them out of the cache in the same way as any other block. Use care when exercising this option because a full scan of a large table may clean most of the other blocks out of the cache.

See Also:

- *Oracle Database SQL Language Reference* for information about the CACHE clause
- *Oracle Database Performance Tuning Guide* to learn how to interpret buffer cache advisory statistics

Buffer Pools

A **buffer pool** is a collection of buffers. The database buffer cache is divided into one or more buffer pools.

You can manually configure separate buffer pools that either keep data in the buffer cache or make the buffers available for new data immediately after using the data blocks. You can then assign specific schema objects to the appropriate buffer pool to control how blocks age out of the cache. The possible buffer pools are as follows:

- Default pool

  This pool is the location where blocks are normally cached. Unless you manually configure separate pools, the default pool is the only buffer pool.
- Keep pool

  This pool is intended for blocks that were accessed frequently, but which aged out of the default pool because of lack of space. The goal of the keep buffer pool is to retain objects in memory, thus avoiding I/O operations.
- Recycle pool

  This pool is intended for blocks that are used infrequently. A recycle

大型表的完全扫描导致将块读入数据库高速缓存，与其他类型读取的处理方式有所不同。（在完成扫描之后，这些缓冲区中的）块可以被立即重用，以防止该扫描真正地清理缓冲区高速缓存。

在某些少有的情况下，若默认行为不是你所希望的，您可以更改表的 CACHE 属性。在这种情况下，数据库不会在缓存中强制或订住块，而是与任何其他块相同的方式，让其从高速缓存中老化移出。使用此选项时要小心，因为大表的全表扫描可能会将大部分的其他块从缓存中清理出去。

另见：

- 《*Oracle 数据库 SQL 语言参考*》有关 CACHE 子句的信息
- 《*Oracle 数据库性能优化指南*》了解如何解释缓冲区高速缓存顾问的统计信息

缓冲区池

缓冲池是缓冲区的集合。数据库缓冲区高速缓存被划分为一个或多个的缓冲池。

您可以手动配置多个单独的缓冲池，要么将数据保留在缓冲区高速缓存中，或在使用了数据块之后、使缓冲区立即可用于新的数据。然后，你可以将模式对象分配给适当的缓冲池，以控制数据块如何从缓存中老化移出。

可能的缓冲池如下所示：

- 缺省池

  该池是块通常被缓存的地方。除非您手动配置单独的池，默认池将是唯一的缓冲池。
- 保留池

  该池用于被频繁访问的块，使其不会由于缺省池的空间不足而被移出。保留缓冲池的目标是将对象保留在内存中，从而避免 I/O 操作。
- 循环池

  该池用于不被频繁使用的块。循环池防止对象在缓存中占用不必要的

pool prevent objects from consuming unnecessary space in the cache.

A database has a standard block size (see "Database Block Size"). You can create a tablespace with a block size that differs from the standard size. Each nondefault block size has its own pool. Oracle Database manages the blocks in these pools in the same way as in the default pool.

Figure 14-7 shows the structure of the buffer cache when multiple pools are used. The cache contains default, keep, and recycle pools. The default block size is 8 KB. The cache contains separate pools for tablespaces that use the nonstandard block sizes of 2 KB, 4 KB, and 16 KB.

Figure 14-7 Database Buffer Cache

Description of "Figure 14-7 Database Buffer Cache"

See Also:

- *Oracle Database 2 Day DBA* and *Oracle Database Administrator's Guide* to learn more about buffer pools
- *Oracle Database Performance Tuning Guide* to learn how to use multiple buffer pools

Redo Log Buffer

The **redo log buffer** is a circular buffer in the SGA that stores redo entries describing changes made to the database. **Redo entries** contain the information necessary to reconstruct, or redo, changes made to the database by DML or DDL operations. Database recovery applies redo entries to data files to reconstruct lost changes.

Oracle Database processes copy redo entries from the user memory space

空间。

数据库有一个标准的块大小 （请参阅"数据库块大小"）。您可以创建一个块大小不同于标准大小的表空间。每个非默认的块大小都有自己的缓冲池。Oracle 数据库使用与默认池中相同的方式，来管理中这些不同块大小的缓冲池中的块。

图 14-7 显示了使用多种缓冲池时的缓冲区高速缓存结构。包含缺省池、 保留池、和循环池。缺省块大小为 8 KB。缓存为各种使用 2 KB、 4 KB 、和 16 KB 的非标准块大小的表空间包含单独的缓冲池。

图 14-7 数据库缓冲区高速缓存

Description of "Figure 14-7 Database Buffer Cache"

另见：

- 《*Oracle 数据库 2 日 DBA*》 和 《*Oracle 数据库管理员指南*》了解缓冲池
- 《*Oracle 数据库性能优化指南*》了解如何使用多个缓冲池

重做日志缓冲区

重做日志缓冲区是 SGA 中的一个循环式缓冲区，用来存储对数据库所做更改的重做条目。**重做条目**包含用于重建（或重做）由 DML 或 DDL 操作对数据库所做更改所需的信息。数据库恢复将重做条目应用到数据文件，以重建丢失的更改。

Oracle 数据库进程将重做条目从用户内存空间复制到 SGA 的重做日志缓冲

to the redo log buffer in the SGA. The redo entries take up continuous, sequential space in the buffer. The background process log writer (LGWR) writes the redo log buffer to the active online redo log group on disk. Figure 14-8 shows this redo buffer activity.

Figure 14-8 Redo Log Buffer

Description of "Figure 14-8 Redo Log Buffer"

LGWR writes redo sequentially to disk while DBW*n* performs scattered writes of data blocks to disk. Scattered writes tend to be much slower than sequential writes. Because LGWR enable users to avoid waiting for DBW*n* to complete its slow writes, the database delivers better performance.

The `LOG_BUFFER` initialization parameter specifies the amount of memory that Oracle Database uses when buffering redo entries. Unlike other SGA components, the redo log buffer and fixed SGA buffer do not divide memory into granules.

See Also:

- "Log Writer Process (LGWR)" and "Importance of Checkpoints for Instance Recovery"
- *Oracle Database Administrator's Guide* for information about the online redo log

Shared Pool

The **shared pool** caches various types of program data. For example, the shared pool stores parsed SQL, PL/SQL code, system parameters, and data dictionary information. The shared pool is involved in almost every operation that occurs in the database. For example, if a user executes a

区中。重做条目占用缓冲区中连续、顺序的空间。日志写入器(LGWR)后台进程将重做日志缓冲区写入磁盘上的活动联机重做日志组。图 14-8 显示了这种重做缓冲区活动。

图 14-8 重做日志缓冲区

Description of "Figure 14-8 Redo Log Buffer"

LGWR 进程将重做信息顺序写入磁盘，而 DBWn 进程将数据块分散写入磁盘。分散写入往往要比顺序写入慢得多。LGWR 使用户能够避免等待 DBWn 完成其缓慢的写入，为数据库提供了更好的性能。

LOG_BUFFER 初始化参数指定了 Oracle 数据库在缓冲重做条目时所能使用的内存量。与其他的 SGA 组件不同，重做日志缓冲区和固定的 SGA 缓冲区不按颗粒划分内存。

另见：

- "日志写入进程 （LGWR）"和"实例恢复检查点的重要性"
- 《*Oracle 数据库管理员指南*》有关联机重做日志的信息

共享池

共享池缓存各种类型的程序数据。例如，共享池存储已解析的 SQL、PL/SQL 代码、 系统参数、和数据字典信息。几乎数据库中发生的每个操作都涉及到共享池。例如，如果用户执行一个 SQL 语句，则 Oracle 数据库会访问共享池。

SQL statement, then Oracle Database accesses the shared pool. The shared pool is divided into several subcomponents, the most important of which are shown in Figure 14-9.

Figure 14-9 Shared Pool

Description of "Figure 14-9 Shared Pool"

This section includes the following topics:

- Library Cache
- Data Dictionary Cache
- Server Result Cache
- Reserved Pool

Library Cache

The **library cache** is a shared pool memory structure that stores executable SQL and PL/SQL code. This cache contains the shared SQL and PL/SQL areas and control structures such as locks and library cache handles. In a shared server architecture, the library cache also contains private SQL areas.

When a SQL statement is executed, the database attempts to reuse previously executed code. If a parsed representation of a SQL statement exists in the library cache and can be shared, then the database reuses the code, known as a **soft parse** or a **library cache hit**. Otherwise, the database must build a new executable version of the application code, known as a **hard parse** or a **library cache miss**.

Shared SQL Areas

共享池分为几个子组件，其中最重要的几个显示在图 14-9 中。

图 14-9 共享池

Description of "Figure 14-9 Shared Pool"

本节包括以下主题：

- 库缓存
- 数据字典缓存
- 服务器结果缓存
- 保留池

库缓存

库缓存是存储可执行 SQL 和 PL/SQL 代码的共享池内存结构。此缓存包含共享 SQL 和 PL/SQL 区，以及锁和库缓存句柄之类的控制结构。在共享服务器体系结构中，库缓存还包含私有 SQL 区。

在执行 SQL 语句时，数据库将尝试重用以前执行过的代码。如果在库缓存中存在该 SQL 语句的已解析表示形式，并且是可以共享的，则数据库会重用该代码，这称为**软解析**或**库缓存命中**。否则，数据库必须为应用程序代码建立一个新的可执行版本，这称为**硬解析**或**库缓存未命中**。

共享 SQL 区

The database represents each SQL statement that it runs in the following SQL areas:

- Shared SQL area

  The database uses the shared SQL area to process the first occurrence of a SQL statement. This area is accessible to all users and contains the statement parse tree and execution plan. Only one shared SQL area exists for a unique statement.
- Private SQL area

  Each session issuing a SQL statement has a private SQL area in its PGA (see "Private SQL Area"). Each user that submits the same statement has a private SQL area pointing to the same shared SQL area. Thus, many private SQL areas in separate PGAs can be associated with the same shared SQL area.

The database automatically determines when applications submit similar SQL statements. The database considers both SQL statements issued directly by users and applications and recursive SQL statements issued internally by other statements.

The database performs the following steps:

- 1. Checks the shared pool to see if a shared SQL area exists for a syntactically and semantically identical statement:

  - If an identical statement exists, then the database uses the shared SQL area for the execution of the subsequent new instances of the statement, thereby reducing memory consumption.
  - If an identical statement does not exist, then the database allocates a new shared SQL area in the shared pool. A statement with the same syntax but different semantics uses a **child cursor**.

  In either case, the private SQL area for the user points to the shared SQL area that contains the statement and execution plan.
- 2. Allocates a private SQL area on behalf of the session

  The location of the private SQL area depends on the connection established for the session. If a session is connected through a shared server, then part of the private SQL area is kept in the SGA.

数据库所运行的每个 SQL 语句，会存在于下列两种 SQL 区之一：

- 共享 SQL 区

  数据库使用共享 SQL 区来处理 SQL 语句第一次发生时的情况。该区域对所有用户可访问，并包含语句的解析树和执行计划。对于每个唯一的语句，只存在一个共享 SQL 区。

- 私有 SQL 区

  每个发出 SQL 语句的会话在其 PGA 中有一个私有 SQL 区（见"私有 SQL 区"）。提交同一语句中的每个用户都分别有一个私有 SQL 区，但都指向同一共享 SQL 区。因此，在多个单独 PGA 中的私有 SQL 区可能与同一共享 SQL 区相关联。

当应用程序提交类似的 SQL 语句时，数据库会自动决定怎么做。数据库会同时考虑从用户和应用程序直接发出的语句，及其它语句从内部发出的递归 SQL 语句。

数据库执行以下步骤：

- 1. 检查共享池，看是否存在某个语法和语义上都相同的语句的共享 SQL 区：

  - 如果存在这样一个相同的语句，则数据库为该语句的后续新实例使用这个共享 SQL 区，从而减少内存消耗。

  - 如果不存在这样一个相同的语句，则数据库在共享池中分配一个新的共享 SQL 区。语法相同、但语义不同的语句会使用一个**子游标**。

  在这两种情况下，用户的私有 SQL 区都指向包含语句和执行计划的共享 SQL 区。
- 2. 为该会话分配一个私有 SQL 区

  私有 SQL 区的位置取决于该会话所建立的连接。如果会话是通过共享服务器连接的，则该私有 SQL 区中的一部分被保存在 SGA 中。

Figure 14-10 shows a dedicated server architecture in which two sessions keep a copy of the same SQL statement in their own PGAs. In a shared server, this copy is in the UGA, which in the large pool or in the shared pool when no large pool exists.

Figure 14-10 Private SQL Areas and Shared SQL Area

Description of "Figure 14-10 Private SQL Areas and Shared SQL Area"

See Also:

- *Oracle Database Performance Tuning Guide* to learn more about managing the library cache
- *Oracle Database Advanced Application Developer's Guide* for more information about shared SQL

Program Units and the Library Cache

The library cache holds executable forms of PL/SQL programs and Java classes. These items are collectively referred to as **program units**.

图 14-10 显示了一个专用服务器架构，其中两个会话在其 PGA 中保留了同一 SQL 语句的副本。若在共享服务器中，此副本会位于大池的 UGA 中，或当大池不存在时位于共享池中。

图 14-10 私有 SQL 区和共享 SQL 区

Description of "Figure 14-10 Private SQL Areas and Shared SQL Area"

另见：

- 《*Oracle 数据库性能优化指南*》了解库缓存管理
- 《*Oracle 数据库高级应用程序开发者指南*》有关共享 SQL 的详细信息

程序单元和库缓存

库缓存包含 PL/SQL 程序和 Java 类的可执行形式。这些项目统称为**程序单元**。

The database processes program units similarly to SQL statements. For example, the database allocates a shared area to hold the parsed, compiled form of a PL/SQL program. The database allocates a private area to hold values specific to the session that runs the program, including local, global, and package variables, and buffers for executing SQL. If multiple users run the same program, then each user maintains a separate copy of his or her private SQL area, which holds session-specific values, and accesses a single shared SQL area.

数据库程序单元的处理与 SQL 语句类似。例如，数据库分配一个共享区域来保存 PL/SQL 程序的已解析和已编译形式。数据库分配一个私有区域来保存运行该程序特定于会话的值，包括为执行 SQL 的本地、全局、包变量、和缓冲区等。如果有多个用户运行同一程序，则每个用户将维护其私有 SQL 区的一个单独副本，其中包含特定于会话的值，以访问某个单一的共享 SQL 区。

The database processes individual SQL statements within a PL/SQL program unit as previously described. Despite their origins within a PL/SQL program unit, these SQL statements use a shared area to hold their parsed representations and a private area for each session that runs the statement.

数据库按前面所述的那样处理 PL/SQL 程序单元内的单个 SQL 语句。尽管它们源于同一 PL/SQL 程序单元，这些 SQL 语句将使用一个共享区域来保存其已分析的表示形式，并为运行该语句的每个会话使用一个私有区域来保存私有数据。

Allocation and Reuse of Memory in the Shared Pool

共享池中的内存分配和重用

The database allocates shared pool memory when a new SQL statement is parsed. The memory size depends on the complexity of the statement.

当解析新的 SQL 语句时，数据库将分配共享池内存。内存大小取决于语句的复杂性。

In general, an item in the shared pool stays until it is removed according to an LRU algorithm. The database allows shared pool items used by many sessions to remain in memory as long as they are useful, even if the process that created the item terminates. This mechanism minimizes the overhead and processing of SQL statements.

通常，共享池中的项目会一直保留，直到按 LRU 算法将其移除。数据库允许共享池中用于多个会话的项目被保留在内存中，只要他们还有用，即使创建该项目的进程已经终止。这一机制能最小化开销和对 SQL 语句的处理。

If space is needed for new items, then the database frees memory for infrequently used items. A shared SQL area can be removed from the shared pool even if the shared SQL area corresponds to an open cursor that has not been used for some time. If the open cursor is subsequently used to run its statement, then Oracle Database reparses the statement and allocates a new shared SQL area.

如果需要为新项目腾出空间，则数据库会释放内存中不常使用的项目。某个共享的 SQL 区可能会被从共享池中删除，即使该共享 SQL 区对应于一个已打开但已有一段时间未使用的游标。如果随后又需要运行这个已打开的游标，则 Oracle 数据库重新解析该语句并分配新的共享 SQL 区。

The database also removes a shared SQL area from the shared pool in the following circumstances:

数据库还在以下情况下从共享池中删除共享 SQL 区：

- If statistics are gathered for a table, table cluster, or index, then by default the database gradually removes all shared SQL areas that contain statements referencing the analyzed object after a period of time. The next time a removed statement is run, the database parses it in a new shared SQL area to reflect the new statistics for the schema object.

- 如果为表、 表簇、或索引收集了统计信息，则默认情况下，数据库在一段时间后，会逐步删除所有包含引用了已分析对象的语句的共享 SQL 区。下次运行某个已删除的语句时，数据库将在一个新的共享 SQL 区中对其进行解析，以反映模式对象的新的统计信息。

- If a schema object is referenced in a SQL statement, and if this object is later modified by a DDL statement, then the database invalidates the shared SQL area. The optimizer must reparse the statement the next time it is run.

- 如果一个模式对象在某个 SQL 语句中被引用，而之后此对象被某个 DDL 语句修改，则数据库将使这个共享 SQL 区无效。当下一次运行

- If you change the global database name, then the database removes all information from the shared pool.

You can use the `ALTER SYSTEM FLUSH SHARED_POOL` statement to manually remove all information in the shared pool to assess the performance that can be expected after an instance restart.

See Also:

- *Oracle Database SQL Language Reference* for information about using `ALTER SYSTEM FLUSH SHARED_POOL`
- *Oracle Database Reference* for information about `V$SQL` and `V$SQLAREA` dynamic views

该语句时，优化程序必须重新解析该语句。

- 如果更改了全局数据库名称，则数据库将删除所有共享池信息。

你可以使用 `ALTER SYSTEM FLUSH SHARED_POOL` 语句，手动删除共享池中的所有信息，以评估实例重新启动后预期的性能。

另见：

- 《*Oracle 数据库 SQL 语言参考*》有关使用 `ALTER SYSTEM FLUSH SHARED_POOL` 的信息
- 《*Oracle 数据库参考*》有关 `V$SQL` 和 `V$SQLAREA` 动态视图的信息

Data Dictionary Cache

The data dictionary is a collection of database tables and views containing reference information about the database, its structures, and its users. Oracle Database accesses the data dictionary frequently during SQL statement parsing.

The data dictionary is accessed so often by Oracle Database that the following special memory locations are designated to hold dictionary data:

- Data dictionary cache

  This cache holds information about database objects. The cache is also known as the **row cache** because it holds data as rows instead of buffers.
- Library cache

All server processes share these caches for access to data dictionary information.

See Also:

- Chapter 6, "Data Dictionary and Dynamic Performance Views"
- *Oracle Database Performance Tuning Guide* to learn how to allocate additional memory to the data dictionary cache

数据字典缓存

数据字典是数据库表和视图的集合，其中包含有关数据库及其结构、用户等参考信息。Oracle 数据库在解析 SQL 语句期间，会频繁访问数据字典。

Oracle 数据库如此频繁地访问数据字典，所以指定了以下这些特别的内存位置来保存字典数据：

- 数据字典缓存

  此缓存保存有关数据库对象的信息。此缓存也称为**行缓存**，因为它按行、而不是按缓冲区保存数据。
- 库缓存

所有服务器进程都共享这些缓存来对数据字典信息进行访问。

另见：

- 第 6 章，"数据字典和动态性能视图"
- 《*Oracle 数据库性能优化指南*》了解如何为数据字典缓存分配额外的内存

Server Result Cache

Unlike the buffer pools, the **server result cache** holds result sets and not data blocks. The server result cache contains the **SQL query result**

服务器结果缓存

与缓冲池保存数据块不同，**服务器结果缓存**保存的是结果集。服务器结果缓存包含 **SQL 查询结果缓存**和 **PL/SQL 函数结果缓存**，它们共享相同的基础

cache and **PL/SQL function result cache**, which share the same infrastructure.

A **client result cache** differs from the server result cache. A client cache is configured at the application level and is located in client memory, not in database memory.

See Also:

- *Oracle Database Administrator's Guide* for information about sizing the result cache
- *Oracle Database PL/SQL Packages and Types Reference* for information about the `DBMS_RESULT_CACHE` package
- *Oracle Database Performance Tuning Guide* for more information about the client result cache

SQL Query Result Cache

The database can store the results of queries and query fragments in the **SQL query result cache**, using the cached results for future queries and query fragments. Most applications benefit from this performance improvement.

For example, suppose an application runs the same `SELECT` statement repeatedly. If the results are cached, then the database returns them immediately. In this way, the database avoids the expensive operation of rereading blocks and recomputing results. The database automatically invalidates a cached result whenever a transaction modifies the data or metadata of database objects used to construct that cached result.

Users can annotate a query or query fragment with a `RESULT_CACHE` hint to indicate that the database should store results in the SQL query result cache. The `RESULT_CACHE_MODE` initialization parameter determines whether the SQL query result cache is used for all queries (when possible) or only for annotated queries.

See Also:

- *Oracle Database Reference* to learn more about the `RESULT_CACHE_MODE` initialization parameter
- *Oracle Database SQL Language Reference* to learn about the `RESULT_CACHE` hint

PL/SQL Function Result Cache

The **PL/SQL function result cache** stores function result sets. Without caching, 1000 calls of a function at 1 second per call would take 1000 seconds. With caching, 1000 function calls with the same inputs could

结构。

客户端结果缓存不同于服务器结果缓存。客户端缓存在应用程序级别配置，并位于客户端内存中，而不是位于数据库内存中。

另见：

- 《*Oracle 数据库管理员指南*》有关设置结果缓存大小的信息
- 《*Oracle 数据库 PL/SQL 包和类型参考*》关于 DBMS_RESULT_CACHE 包的信息
- 《*Oracle 数据库性能优化指南*》有关客户端结果缓存的详细信息

SQL 查询结果缓存

数据库能在 **SQL 查询结果高速缓存**中存储查询和查询片段的结果，将此缓存结果用于将来的查询和查询片段。大多数应用程序受益于这种性能改善。

例如，假设应用程序将重复执行相同的 SELECT 语句。如果结果会被缓存，则数据库会将其立即返回。通过这种方式，数据库避免了重读块和重计算结果等昂贵操作。每当事务修改了数据或用于构造该缓存结果的数据库对象元数据时，数据库自动使其高速缓存结果无效。

用户可以为查询或查询片断添加 RESULT_CACHE 提示，以指示数据库应将其结果存储在 SQL 查询结果缓存中。由 RESULT_CACHE_MODE 初始化参数确定 SQL 查询结果缓存将用于所有查询（如果可能） 还是仅用于添加了注释的查询。

另见：

- 《*Oracle 数据库参考*》了解 RESULT_CACHE_MODE 初始化参数
- 《*Oracle 数据库 SQL 语言参考*》了解 RESULT_CACHE 提示

PL/SQL 函数结果缓存

PL/SQL 函数结果缓存存储函数的结果集。若不使用缓存功能，每次调用花 1 秒钟的函数，调用该函数 1000 次将花时间 1000 秒。而缓存后，以相同的输入调用函数 1000 次总共仅需要时间 1 秒。对于在相对静态的数据上

take 1 second *total*. Good candidates for result caching are frequently invoked functions that depend on relatively static data.
PL/SQL function code can include a request to cache its results. Upon invocation of this function, the system checks the cache. If the cache contains the result from a previous function call with the same parameter values, then the system returns the cached result to the invoker and does not reexecute the function body. If the cache does not contain the result, then the system executes the function body and adds the result (for these parameter values) to the cache before returning control to the invoker.

Note:

You can specify the database objects that are used to compute a cached result so that if any of them are updated, the cached result becomes invalid and must be recomputed.
The cache can accumulate many results—one result for every unique combination of parameter values with which each result-cached function was invoked. If the database needs more memory, then it ages out one or more cached results.

See Also:

- *Oracle Database Advanced Application Developer's Guide* to learn more about the PL/SQL function result cache
- *Oracle Database PL/SQL Language Reference* to learn more about the PL/SQL function result cache

被频繁调用的函数，结果缓存是一个很好的选择。

PL/SQL 函数代码可以包括一个缓存其结果的请求。调用此函数时，系统将检查缓存。如果高速缓存中包含之前使用相同参数值调用该函数的结果缓存，则系统将缓存结果返回给调用程序，而不会重新执行函数体。如果缓存中不包含其结果，则系统执行函数体，并在将控制返回给调用程序前，将其（针对这些参数值的）结果添加到缓存中。

注意：

您可以指定用于计算缓存结果的数据库对象，如果其中任何对象被更新，则缓存结果变为无效且必须重新计算。

缓存中可能会积累很多的结果缓存 ——当调用每个结果缓存函数时，每个不同的参数组合都形成一个结果缓存。如果数据库需要更多内存，则它会移除一个或多个缓存结果。

另见：

- 《*Oracle 数据库高级应用程序开发者指南*》了解 PL/SQL 函数结果缓存
- 《*Oracle 数据库 PL/SQL 语言参考*》了解 PL/SQL 函数结果缓存

Reserved Pool

The **reserved pool** is a memory area in the shared pool that Oracle Database can use to allocate large contiguous chunks of memory.
Allocation of memory from the shared pool is performed in chunks. Chunking allows large objects (over 5 KB) to be loaded into the cache without requiring a single contiguous area. In this way, the database reduces the possibility of running out of contiguous memory because of fragmentation.
Infrequently, Java, PL/SQL, or SQL cursors may make allocations out of the shared pool that are larger than 5 KB. To allow these allocations to occur most efficiently, the database segregates a small amount of the shared pool for the reserved pool.

See Also:

Oracle Database Performance Tuning Guide to learn how to configure the reserved pool

保留池

保留池是共享池中的一个内存区， Oracle 数据库使用它来分配大的连续内存块。
从共享池中分配内存是按大块执行的。大块使得大型对象(超过 5 KB)可以被加载到缓存中，而不需要一个单一的连续区域。这样一来，数据库中由于碎片引起的连续内存不足的可能性降低了。

有时，Java、 PL/SQL、或 SQL 游标也可能会从共享池中分配大于 5 KB 的内存块。为使这种分配更有效，数据库从共享池中为保留池隔离出少量内存。

另见：

《*Oracle 数据库性能优化指南*》了解如何配置保留池

Large Pool

The **large pool** is an optional memory area intended for memory allocations that are larger than is appropriate for the shared pool. The large pool can provide large memory allocations for the following:

- UGA for the shared server and the Oracle XA interface (used where transactions interact with multiple databases)
- Message buffers used in the parallel execution of statements
- Buffers for Recovery Manager (RMAN) I/O slaves

By allocating session memory from the large pool for shared SQL, the database avoids performance overhead caused by shrinking the shared SQL cache. By allocating memory in large buffers for RMAN operations, I/O server processes, and parallel buffers, the large pool can satisfy large memory requests better than the shared pool.

Figure 14-11 is a graphical depiction of the large pool.

Figure 14-11 Large Pool

Description of "Figure 14-11 Large Pool"

The large pool is different from reserved space in the shared pool, which uses the same LRU list as other memory allocated from the shared pool. The large pool does not have an LRU list. Pieces of memory are allocated and cannot be freed until they are done being used. As soon as a chunk of memory is freed, other processes can use it.

See Also:

- "Dispatcher Request and Response Queues" to learn about allocating session memory for shared server

大池

大池是一个可选的内存区域，它所提供的内存分配往往比共享池分配的内存更大。大池可以提供以下大内存分配：

- 共享服务器的 UGA，和 Oracle XA 接口 （用于与多个数据库进行交互的事务）
- 并行执行语句中使用的消息缓冲区
- 恢复管理器 （RMAN) I/O 从属进程的缓冲区

通过从大池中为共享 SQL 分配会话内存，数据库可以避免由收缩共享 SQL 缓存引起的性能开销。通过为 RMAN 操作、I/O 服务器进程、和并行缓冲区等分配大的缓冲区内存，大池可以比共享池更好地满足大型内存请求。

图 14-11 是大池的图形显示方式。

图 14-11 大池

Description of "Figure 14-11 Large Pool"

共享池中的保留空间与其它内存分配都使用同一 LRU 列表，而大池与此不同，它根本就没有 LRU 列表。这些被分配的内存片断直到操作完成后才会被释放。一旦释放了一个大块内存，其他进程就可以使用它。

另见：

- "调度器请求和响应队列"了解为共享服务器分配会话内存

- *Oracle Database Advanced Application Developer's Guide* to learn about Oracle XA
- *Oracle Database Performance Tuning Guide* for more information about the large pool
- "Parallel Execution" for information about allocating memory for parallel execution

Java Pool

The **Java pool** is an area of memory that stores all session-specific Java code and data within the Java Virtual Machine (JVM). This memory includes Java objects that are migrated to the Java session space at end-of-call.

For dedicated server connections, the Java pool includes the shared part of each Java class, including methods and read-only memory such as code vectors, but not the per-session Java state of each session. For shared server, the pool includes the shared part of each class and some UGA used for the state of each session. Each UGA grows and shrinks as necessary, but the total UGA size must fit in the Java pool space.

The Java Pool Advisor statistics provide information about library cache memory used for Java and predict how changes in the size of the Java pool can affect the parse rate. The Java Pool Advisor is internally turned on when `statistics_level` is set to `TYPICAL` or higher. These statistics reset when the advisor is turned off.

See Also:

- *Oracle Database Java Developer's Guide*
- *Oracle Database Performance Tuning Guide* to learn about views containing Java pool advisory statistics

Streams Pool

The **Streams pool** stores buffered queue messages and provides memory for Oracle Streams capture processes and apply processes. The Streams pool is used exclusively by Oracle Streams.

Unless you specifically configure it, the size of the Streams pool starts at zero. The pool size grows dynamically as required by Oracle Streams.

See Also:

Oracle Database 2 Day + Data Replication and Integration Guide and *Oracle Streams Replication Administrator's Guide*

Fixed SGA

- 《*Oracle 数据库高级应用开发指南*》了解 Oracle XA
- 《*Oracle 数据库性能调整指南*》 有关大池的详细信息
- “并行执行”有关为并行执行分配内存的信息

Java 池

Java 池是一个存储 Java 虚拟机 (JVM) 内所有会话特定的 Java 代码和数据的内存区。此内存包括在**调用端**迁移到 Java 会话空间的 Java 对象。

对于专用服务器连接，Java 池包括每个 Java 类的共享部分，包括方法和只读内存，如代码向量，但不包括每个会话的 Java 状态。对于共享服务器，Java 池包括每个 Java 类的共享部分，和用于每个会话状态的一些 UGA 。每个 UGA 根据需要增长和缩小，但总的 UGA 大小必须适应 Java 池空间。

Java 池顾问统计提供有关用于 Java 的库缓存内存信息，并预测 Java 池大小的更改如何影响解析率。当 `statistics_level` 设置为 `TYPICAL` 或更高时，Java 池顾问在内部会被打开。这些统计信息在顾问被关闭时会被重置。

另见：

- 《*Oracle 数据库 Java 开发人员指南*》
- 《*Oracle 数据库性能优化指南*》了解包含 Java 池顾问统计信息的有关视图

流池

流池用于存储缓冲的队列消息，并为 Oracle 流的捕获进程和应用进程提供内存。流池专门由 Oracle 流使用。

如果您没有专门配置流池，则其大小从零开始。池池的大小由 Oracle 流按需动态增长。

另见：

《*Oracle 数据库 2 日 + 数据复制和集成指南*》和 《*Oracle 数据流复制管理员指南*》

固定 SGA

The **fixed SGA** is an internal housekeeping area. For example, the fixed SGA contains:

- General information about the state of the database and the instance, which the background processes need to access
- Information communicated between processes, such as information about **locks** (see "Overview of Automatic Locks")

The size of the fixed SGA is set by Oracle Database and cannot be altered manually. The fixed SGA size can change from release to release.

Overview of Software Code Areas

Software code areas are portions of memory that store code that is being run or can be run. Oracle Database code is stored in a software area that is typically more exclusive and protected than the location of user programs.

Software areas are usually static in size, changing only when software is updated or reinstalled. The required size of these areas varies by operating system.

Software areas are read-only and can be installed shared or nonshared. Some database tools and utilities, such as Oracle Forms and SQL*Plus, can be installed shared, but some cannot. When possible, database code is shared so that all users can access it without having multiple copies in memory, resulting in reduced main memory and overall improvement in performance. Multiple instances of a database can use the same database code area with different databases if running on the same computer.

Note:

The option of installing software shared is not available for all operating systems, for example, on PCs operating Windows. See your operating system-specific documentation for more information.

固定 SGA 是内部的内务管理区域。例如，固定 SGA 包含：

- 有关数据库及其实例状态的一般信息，后台进程需要访问这些信息
- 进程间通讯的信息，如有关**锁**的信息 （请参阅"自动锁概述"）

固定 SGA 的大小由 Oracle 数据库设置，且不能手动更改。固定 SGA 大小可能会因为版本不同而不同。

软件代码区概述

软件代码区是用于存储正在运行或可以运行的代码的那部分内存。Oracle 数据库代码的存储位置通常比用户程序更专用、更受保护。

软件区的大小通常是静态的，其大小只在软件更新或重新安装时才会改变。这些区域所需的大小因操作系统而异。

软件区是只读的，并且可以按共享或非共享方式安装。某些数据库工具，如 Oracle Forms 和 SQL*Plus，可以共享安装，但有些却不能。只要可能，这些数据库代码会被共享，以便所有用户都可以访问它，而无需在内存中存在多个副本，从而减少主内存并改进整体性能。数据库的多个实例可以与同一台计算机上运行的不同的数据库使用相同的数据库代码区域。

注意：

以共享方式安装软件的选项并不是对所有操作系统都可用的，例如，在运行 Windows 的 PC 上是不可用的。请参阅您的特定操作系统文档的详细信息。

15 Process Architecture

This chapter discusses the processes in an Oracle database.
This chapter contains the following sections:

- Introduction to Processes
  - Multiple-Process Oracle Database Systems
  - Types of Processes
- Overview of Client Processes
  - Client and Server Processes
  - Connections and Sessions
- Overview of Server Processes
  - Dedicated Server Processes
  - Shared Server Processes
- Overview of Background Processes
  - Mandatory Background Processes
  - Optional Background Processes
  - Slave Processes

Introduction to Processes

A process is a mechanism in an operating system that can run a series of steps. The mechanism depends on the operating system. For example, on Linux an Oracle background process is a Linux process. On Windows, an Oracle background process is a thread of execution within a process.
Code modules are run by processes. All connected Oracle Database users must run the following modules to access a database instance:

- Application or Oracle Database utility

  A database user runs a database application, such as a precompiler program or a database tool such as SQL*Plus, that issues SQL statements to a database.
- Oracle database code

  Each user has Oracle database code executing on his or her behalf

第 15 章 进程体系结构

本章讨论 Oracle 数据库中的进程。
本章包括如下各节：

- 进程简介
  - 多进程 Oracle 数据库系统
  - 进程类型
- 客户端进程概述
  - 客户端和服务器进程
  - 连接和会话
- 服务器进程概述
  - 专用服务器进程
  - 共享服务器进程
- 后台进程概述
  - 强制性后台进程
  - 可选后台进程
  - 从属进程

进程简介

进程是在操作系统中可以运行一系列步骤的机制。该机制取决于操作系统。例如，在 Linux 上的一个 Oracle 后台进程是一个 Linux 进程。而在 Windows 上的一个 Oracle 后台进程是在某个进程内执行的一个线程。

代码模块由进程来运行。所有已连接的 Oracle 数据库用户必须运行以下模块来访问数据库实例：

- 应用程序或 Oracle 数据库实用程序

  数据库用户运行一个数据库应用程序，如预编译器程序或数据库工具（如 SQL*Plus），来向数据库发出 SQL 语句。

- Oracle 数据库代码

  每个用户都具有为其运行的 Oracle 数据库代码，以解释和处理应用

that interprets and processes the application's SQL statements.
A process normally runs in its own private memory area. Most processes can periodically write to an associated **trace file** (see "Trace Files").

See Also:

"Tools for Database Administrators" and "Tools for Database Developers"

Multiple-Process Oracle Database Systems

Multiple-process Oracle (also called **multiuser Oracle**) uses several processes to run different parts of the Oracle Database code and additional processes for the users—either one process for each connected user or one or more processes shared by multiple users. Most databases are multiuser because a primary advantages of a database is managing data needed by multiple users simultaneously.

Each process in a database instance performs a specific job. By dividing the work of the database and applications into several processes, multiple users and applications can connect to an instance simultaneously while the system gives good performance.

Types of Processes

A database instance contains or interacts with the following types of processes:

- **Client processes** run the application or Oracle tool code.
- **Oracle processes** run the Oracle database code. Oracle processes including the following subtypes:
  - **Background processes** start with the database instance and perform maintenance tasks such as performing instance recovery, cleaning up processes, writing redo buffers to disk, and so on.
  - **Server processes** perform work based on a client request.

    For example, these processes parse SQL queries, place them in the shared pool, create and execute a query plan for each query, and read buffers from the database buffer cache or from disk.

    Note:

    Server processes, and the process memory allocated in these processes, run in the instance. The instance continues to function when server processes terminate.

程序的 SQL 语句。

进程通常在其自己的私有内存区内运行。大多数进程可以定期写入一个关联的**跟踪文件** （请参阅"跟踪文件"）。

另见：

"数据库管理员工具"和"数据库开发人员工具"

多进程 Oracle 数据库系统

多进程 Oracle （也称为**多用户 Oracle**） 使用几个进程来运行 Oracle 数据库代码的不同部分，并为用户运行一些其他进程——要么每个连接的用户一个进程，或由多个用户共享一个或多个进程。大多数数据库都是多用户的，因为数据库的主要优势是管理同时由多个用户所需的数据。

数据库实例中的每个进程执行某个特定的作业。通过将数据库和应用程序的工作划分成几个进程，多个用户和应用程序可以同时连接到实例，而系统仍可提供较好的性能。

进程类型

数据库实例包含如下类型的进程，或与以下类型的进程进行交互：

- **客户端进程**，运行应用程序或 Oracle 工具代码。
- **Oracle 进程**，运行 Oracle 数据库代码。Oracle 进程包括下列子类型：
  - **后台进程**，与数据库实例同时启动，并执行一些维护任务，如执行实例恢复、清理进程、将重做缓冲区写入磁盘等。
  - **服务器进程**，基于客户端请求，执行相应的工作。

    例如，这些进程解析 SQL 查询、将查询放入共享池、为查询创建并执行查询计划、和将数据从数据库缓冲区高速缓存（或从磁盘）读入缓冲区。

    注意：

    服务器进程，和为这些进程分配的进程内存，是在该实例中运行的。当服务器进程终止时，实例仍将继续运行。

- **Slave processes** perform additional tasks for a background or server process.

The process structure varies depending on the operating system and the choice of Oracle Database options. For example, the code for connected users can be configured for dedicated server or shared server connections. In a shared server architecture, each server process that runs database code can serve multiple client processes.

Figure 15-1 shows a system global area (SGA) and background processes using dedicated server connections. For each user connection, the application is run by a client process that is different from the dedicated server process that runs the database code. Each client process is associated with its own server process, which has its own program global area (PGA).

Figure 15-1 Oracle Processes and the SGA

- **从属进程**，为后台进程或服务器进程执行额外任务。

进程的结构取决于操作系统，以及选择了哪些 Oracle 数据库选项。例如，已连接用户的代码可能会配置为专用服务器或共享服务器的连接。在一个共享服务器的体系结构中，每个运行数据库代码的服务器进程可以为多个客户端进程服务。

图 15-1 显示了使用专用服务器连接时的系统全局区 （SGA） 和后台进程。对每个用户连接，应用程序由一个客户端进程来运行，而由其专用服务器进程来运行数据库代码。每个客户端进程都与其自己的服务器进程相关联，而每个服务器进程都有其自己的程序全局区（PGA）。

图 15-1 Oracle 进程和 SGA

Description of "Figure 15-1 Oracle Processes and the SGA"

See Also:

- "Dedicated Server Architecture" and "Shared Server Architecture"
- Your Oracle Database operating system-specific documentation for more details on configuration choices
- *Oracle Database Reference* to learn about the `V$PROCESS` view

Overview of Client Processes

When a user runs an application such as a Pro*C program or SQL*Plus, the operating system creates a client process (sometimes called a **user**

Description of "Figure 15-1 Oracle Processes and the SGA"

另见：

- "专用服务器体系结构"和"共享服务器体系结构"
- 您的特定于操作系统的 Oracle 数据库的文档，关于配置选项的更多详细信息
- 《*Oracle 数据库参考*》了解 `V$PROCESS` 视图

客户端进程概述

当用户运行一个应用程序（如 Pro*C 程序或 SQL*Plus）时，操作系统会创建一个客户端进程 （有时称为**用户进程**）来运行用户应用程序。客户端应

process) to run the user application. The client application has Oracle Database libraries linked into it that provide the APIs required to communicate with the database.

用程序具有已链接到进程的 Oracle 数据库库文件，以提供与数据库进行通信所需的 API。

Client and Server Processes

Client processes differ in important ways from the Oracle processes interacting directly with the instance. The Oracle processes servicing the client process can read from and write to the SGA, whereas the client process cannot. A client process can run on a host other than the database host, whereas Oracle processes cannot.

For example, assume that a user on a client host starts SQL*Plus and connects over the network to database `sample` on a different host (the database instance is not started):

```
SQL> CONNECT SYS@inst1 AS SYSDBA
Enter password: *********
Connected to an idle instance.
```

On the client host, a search of the processes for either `sqlplus` or `sample` shows only the `sqlplus` client process:

```
% ps -ef | grep -e sample -e sqlplus | grep -v grep
clientuser 29437 29436  0 15:40 pts/1    00:00:00 sqlplus as sysdba
```

On the database host, a search of the processes for either `sqlplus` or `sample` shows a server process with a nonlocal connection, but no client process:

```
% ps -ef | grep -e sample -e sqlplus | grep -v grep
serveruser 29441     1  0 15:40 ?        00:00:00 oraclesample (LOCAL=NO)
```

客户端进程和服务器进程

客户端进程和与实例直接交互的 Oracle 进程在某些重要方面不同。为客户端进程服务的 Oracle 进程可能会读取或写入 SGA，而客户端进程不能。客户端进程可以在数据库主机之外另一台主机上运行，而 Oracle 进程不能。

例如，假设在客户端主机上的某个用户启动 SQL*Plus，并通过网络连接到另一台主机上的 `sample` 数据库（该数据库实例还未启动）：

```
SQL> CONNECT SYS@inst1 AS SYSDBA
Enter password: *********
Connected to an idle instance.
```

在客户端主机上，搜索 sqlplus 或 `sample` 相关进程，显示只有 sqlplus 客户端进程：

```
% ps -ef | grep -e sample -e sqlplus | grep -v grep
clientuser 29437 29436  0 15:40 pts/1    00:00:00 sqlplus as sysdba
```

在数据库主机上，搜索 sqlplus 或 `sample` 相关进程，显示一个非本地连接的服务器进程，但没有客户端进程：

```
% ps -ef | grep -e sample -e sqlplus | grep -v grep
serveruser 29441     1  0 15:40 ?        00:00:00 oraclesample (LOCAL=NO)
```

Connections and Sessions

A connection is a physical communication pathway between a client process and a database instance. A communication pathway is established using available interprocess communication mechanisms or network software. Typically, a connection occurs between a client process and a server process or dispatcher, but it can also occur between a client process and Oracle Connection Manager (CMAN).

A session is a logical entity in the database instance memory that represents the state of a current user login to a database. For example, when a user is authenticated by the database with a password, a session is established for this user. A session lasts from the time the user is authenticated by the database until the time the user disconnects or exits the database application.

A single connection can have 0, 1, or more sessions established on it. The

连接和会话

连接是在客户端进程和数据库实例之间的一条物理通信路径。通信路径使用可用的进程间通信机制或网络软件来建立。通常，连接在客户端进程和服务器进程（或调度器进程）之间产生，但它也可以在客户端进程与 Oracle 连接管理器 （CMAN）之间产生。

会话是数据库实例内存中的一个逻辑实体，表示登录到数据库的某个当前用户的状态。例如，当用户使用密码通过了数据库身份验证时，会为此用户建立一个会话。会话的持续时间从用户通过数据库验证开始，直到用户断开连接，或退出数据库应用程序。

在单个连接之上，可以建立 0 个、 1 个、或多个会话。这些会话相互独

sessions are independent: a commit in one session does not affect transactions in other sessions.

Note:

If Oracle Net connection pooling is configured, then it is possible for a connection to drop but leave the sessions intact.

Multiple sessions can exist concurrently for a single database user. As shown in Figure 15-2, user `hr` can have multiple connections to a database. In dedicated server connections, the database creates a server process on behalf of each connection. Only the client process that causes the dedicated server to be created uses it. In a shared server connection, many client processes access a single shared server process.

Figure 15-2 One Session for Each Connection

Description of "Figure 15-2 One Session for Each Connection"

Figure 15-3 illustrates a case in which user `hr` has a single connection to a database, but this connection has two sessions.

Figure 15-3 Two Sessions in One Connection

立：一个会话中的事务提交并不会影响其他会话中的事务。

注意：

如果配置了 Oracle 网络连接池，则可能出现连接已断掉、而会话却保持不变的情况。

单个数据库用户可能会同时运行多个会话。如图 15-2 所示，`hr` 用户可能具有到数据库的多个连接。对于专用服务器连接，数据库为每个连接创建一个服务器进程。只有导致创建该专用服务器进程的客户端进程才能使用它（该会话）。对于共享服务器连接，由多个客户端进程访问单个共享服务器进程。

图 15-2 每个连接一个会话

Description of "Figure 15-2 One Session for Each Connection"

如图 15-3 所示，hr 用户仅有一个单一数据库连接，但此连接有两个会话。

图 15-3 单个连接中的两个会话

Description of "Figure 15-3 Two Sessions in One Connection"

Generating an autotrace report of SQL statement execution statistics re-creates the scenario in Figure 15-3. Example 15-2 connects SQL*Plus to the database as user SYSTEM and enables tracing, thus creating a new session (sample output included).

Example 15-1 One Connection with Two Sessions

```
SQL> SELECT SID, SERIAL#, PADDR FROM V$SESSION
WHERE USERNAME = USER;
SID SERIAL# PADDR
--- ------- --------
 90      91 3BE2E41C

SQL> SET AUTOTRACE ON STATISTICS;
SQL> SELECT SID, SERIAL#, PADDR FROM V$SESSION
WHERE USERNAME = USER;

SID SERIAL# PADDR
--- ------- --------
 88      93 3BE2E41C
 90      91 3BE2E41C
...
SQL> DISCONNECT
```

The DISCONNECT command in Example 15-1 actually ends the *sessions*, not the connection. Opening a new terminal and connecting to the instance as a different user, the query in Example 15-2 shows that the connection from Example 15-1 is still active.

Example 15-2 Connection with No Sessions

```
SQL> CONNECT dba1@inst1
Password: ********
Connected.
SQL> SELECT PROGRAM FROM V$PROCESS WHERE ADDR =
HEXTORAW('3BE2E41C');
PROGRAM
------------------------------------------------
oracle@stbcs09-1 (TNS V1-V3)
```

See Also:

"Shared Server Architecture"

Description of "Figure 15-3 Two Sessions in One Connection"

生成一个 SQL 语句执行统计信息的自动跟踪报告，将再现图 15-3 中的场景。示例 15-2 使用 SYSTEM 用户从 SQL*Plus 连接到数据库，并启用跟踪，从而创建一个新的会话 （包括示例输出）。

示例 15-1 有两个会话的单个连接

```
SQL> SELECT SID, SERIAL#, PADDR FROM V$SESSION
WHERE USERNAME = USER;
SID SERIAL# PADDR
--- ------- --------
 90      91 3BE2E41C

SQL> SET AUTOTRACE ON STATISTICS;
SQL> SELECT SID, SERIAL#, PADDR FROM V$SESSION
WHERE USERNAME = USER;

SID SERIAL# PADDR
--- ------- --------
 88      93 3BE2E41C
 90      91 3BE2E41C
...
SQL> DISCONNECT
```

在示例 15-1 中的 DISCONNECT 命令实际上只是结束了会话，而未中断连接。打开一个新的终端并以一个不同的用户连接到实例，示例 15-2 中的查询显示，示例 15-1 中的连接仍处于活动状态。

示例 15-2 没有会话的连接

```
SQL> CONNECT dba1@inst1
Password: ********
Connected.
SQL> SELECT PROGRAM FROM V$PROCESS WHERE ADDR =
HEXTORAW('3BE2E41C');
PROGRAM
------------------------------------------------
oracle@stbcs09-1 (TNS V1-V3)
```

另见：

"共享服务器体系结构"

Overview of Server Processes

Oracle Database creates **server processes** to handle the requests of client processes connected to the instance. A client process always communicates with a database through a separate server process.
Server processes created on behalf of a database application can perform one or more of the following tasks:

- Parse and run SQL statements issued through the application, including creating and executing the query plan (see "Stages of SQL Processing")
- Execute PL/SQL code
- Read data blocks from data files into the database buffer cache (the DBW*n* background process has the task of writing modified blocks back to disk)
- Return results in such a way that the application can process the information

Dedicated Server Processes

In dedicated server connections, the client connection is associated with one and only one server process (see "Dedicated Server Architecture"). On Linux, 20 client processes connected to a database instance are serviced by 20 server processes.
Each client process communicates directly with its server process. This server process is dedicated to its client process for the duration of the session. The server process stores process-specific information and the UGA in its PGA (see "PGA Usage in Dedicated and Shared Server Modes").

Shared Server Processes

In shared server connections, client applications connect over a network to a **dispatcher process**, not a server process (see "Shared Server Architecture"). For example, 20 client processes can connect to a single dispatcher process.
The dispatcher process receives requests from connected clients and puts them into a request queue in the large pool (see "Large Pool"). The first available shared server process takes the request from the queue and processes it. Afterward, the shared server place the result into the dispatcher response queue. The dispatcher process is monitors this queue and transmits the result to the client.
Like a dedicated server process, a shared server process has its own PGA. However, the UGA for a session is in the SGA so that any shared server

服务器进程概述

Oracle 数据库创建**服务器进程**，以处理连接到实例的客户端进程的请求。客户端进程总是通过单独的服务器进程与数据库进行通信。

为某个数据库应用程序创建的服务器进程，可以执行一个或多个下列任务：

- 解析并运行通过应用程序发出的 SQL 语句，包括创建和执行查询计划 （参见" SQL 处理的几个阶段"）
- 执行 PL/SQL 代码
- 将数据块从数据文件读入数据库缓冲区高速缓存 （将已修改的块写回磁盘是 DBWn 后台进程的任务）
- 返回结果信息，以便应用程序可以进一步处理

专用服务器进程

对于专用服务器连接，客户端连接与一个且仅与一个服务器进程（参见"专用服务器体系结构"）相关联。在 Linux 上，连接到一个数据库实例的 20 个客户端进程，有相应的 20 个服务器进程为其提供服务。

每个客户端进程直接与其服务器进程进行通信。在会话期间，此服务器进程专用于其客户端进程。服务器进程将特定于进程的信息和 UGA 存储在其 PGA 中 （参见"在专用和共享服务器模式中使用 PGA"）。

共享服务器进程

对于共享服务器连接，客户端应用程序通过网络连接到一个**调度器进程**，而不是一个服务器进程（请参见"共享服务器体系结构"）。例如，20 个客户端进程可能会仅连接到一个单一的调度器进程。

调度器进程接收来自已连接客户端的请求，并将它们放入大池中的请求队列（参见"大池"）。第一个可用的共享服务器进程从队列承接一个请求并处理之。之后，共享服务器进程将结果放回调度器响应队列。调度器进程会监视此队列，并将结果传回客户端。

与专用服务器进程类似，共享服务器进程也有其自己的 PGA。但是，会话

can access session data.

Overview of Background Processes

A multiprocess Oracle database uses some additional processes called **background processes**. The background processes perform maintenance tasks required to operate the database and to maximize performance for multiple users.

Each background process has a separate task, but works with the other processes. For example, the LGWR process writes data from the redo log buffer to the online redo log. When a filled log file is ready to be archived, `LGWR` signals another process to archive the file.

Oracle Database creates background processes automatically when a database instance starts. An instance can have many background processes, not all of which always exist in every database configuration. The following query lists the background processes running on your database:

```
SELECT PNAME
FROM   V$PROCESS
WHERE  PNAME IS NOT NULL
ORDER BY PNAME;
```

This section includes the following topics:

- Mandatory Background Processes
- Optional Background Processes
- Slave Processes

See Also:

Oracle Database Reference for descriptions of all the background processes

Mandatory Background Processes

The **mandatory background processes** are present in all typical database configurations. These processes run by default in a database instance started with a minimally configured initialization parameter file (see Example 13-1).

This section describes the following mandatory background processes:

的 UGA 位于 SGA 中，以便任何共享服务器进程可以访问其中的会话数据。

后台进程概述

多进程 Oracle 数据库会使用某些称为**后台进程**的额外进程。后台进程执行操作数据库所需的维护任务，并最大化多用户性能。

每个后台进程都有其单独的任务，但又与其他进程相互协作。例如， LGWR 进程将重做日志缓冲区中的数据写入到联机重做日志。当已填充的日志文件将被存档时， LGWR 通知另一个进程来归档该文件。

数据库实例启动时，Oracle 数据库将自动创建后台进程。实例可能会有许多后台进程，但并不是所有这些进程在每种数据库配置中都存在。以下查询列出了在您的数据库上运行的后台进程：

```
SELECT PNAME
FROM   V$PROCESS
WHERE  PNAME IS NOT NULL
ORDER BY PNAME;
```

本节包括以下主题：

- 强制性后台进程
- 可选后台进程
- 从属进程

另见：

《*Oracle 数据库参考*》有关所有后台进程的说明

强制性后台进程

强制性后台进程存在于所有典型的数据库配置。默认情况下，这些进程运行在以一个最小化配置的初始化参数文件启动的数据库实例中 （请参见示例 13-1）。

本节介绍以下强制性后台进程：

- Process Monitor Process (PMON)
- System Monitor Process (SMON)
- Database Writer Process (DBW***n***)
- Log Writer Process (LGWR)
- Checkpoint Process (CKPT)
- Manageability Monitor Processes (MMON and MMNL)
- Recoverer Process (RECO)

See Also:

- *Oracle Database Reference* for descriptions of other mandatory processes, including MMAN, DIAG, VKTM, DBRM, and PSP0

- *Oracle Real Application Clusters Administration and Deployment Guide* and *Oracle Clusterware Administration and Deployment Guide* for more information about background processes specific to Oracle RAC and Oracle Clusterware

Process Monitor Process (PMON)

The **process monitor (PMON)** monitors the other background processes and performs process recovery when a server or dispatcher process terminates abnormally. PMON is responsible for cleaning up the database buffer cache and freeing resources that the client process was using. For example, PMON resets the status of the active transaction table, releases locks that are no longer required, and removes the process ID from the list of active processes.

PMON also registers information about the instance and dispatcher processes with the **Oracle Net listener** (see "The Oracle Net Listener"). When an instance starts, PMON polls the listener to determine whether it is running. If the listener is running, then PMON passes it relevant parameters. If it is not running, then PMON periodically attempts to contact it.

System Monitor Process (SMON)

The **system monitor process (SMON)** is in charge of a variety of system-level cleanup duties. The duties assigned to SMON include:

- Performing instance recovery, if necessary, at instance startup. In an Oracle RAC database, the SMON process of one database instance can perform instance recovery for a failed instance.

- 进程监视器进程 (PMON)
- 系统监视器进程(SMON)
- 数据库写入器进程(DBWn)
- 日志写入器进程(LGWR)
- 检查点进程(CKPT)
- 可管理性监视器进程(MMON and MMNL)
- 恢复器进程(RECO)

另见：

- 《*Oracle 数据库参考*》关于其他强制性进程，包括 MMAN、 DIAG、VKTM、 DBRM、和 PSP0 等的说明

- 《*Oracle 真正应用集群管理和部署指南*》和 《*Oracle 集群件管理和部署指南*》关于 Oracle RAC 和 Oracle 集群件特定的后台进程的详细信息

进程监视器进程 （PMON）

进程监视器 （PMON） 监视其他的后台进程，并在某个服务器进程或调度器进程异常终止时执行进程恢复。PMON 负责清理数据库缓冲区高速缓存，并释放客户端进程以前还在使用的资源。例如， PMON 重置活动事务表的状态、 释放不再需要的锁、并从活动进程列表中删除其进程 ID。

PMON 还将实例和调度器进程的信息注册到 **Oracle 网络监听器**（见 "Oracle 网络监听器"）。当实例启动时， PMON 轮询监听器以确定它是否正在运行。如果监听器正在运行，则 PMON 将有关参数传递给它。如果监听器未运行，则 PMON 定期尝试联系它。

系统监视器进程 （SMON）

系统监视器进程 （SMON） 负责各种系统级清理职责。分配给 SMON 的职责包括：

- 如有必要，在实例启动时执行实例恢复。在 Oracle RAC 数据库中，一个数据库实例的 SMON 进程可以为另一个失败的实例执行实例恢复。

- Recovering terminated transactions that were skipped during instance recovery because of file-read or tablespace offline errors. SMON recovers the transactions when the tablespace or file is brought back online.
- Cleaning up unused temporary segments. For example, Oracle Database allocates extents when creating an index. If the operation fails, then SMON cleans up the temporary space.
- Coalescing contiguous free extents within dictionary-managed tablespaces.

SMON checks regularly to see whether it is needed. Other processes can call SMON if they detect a need for it.

Database Writer Process (DBW*n*)

The **database writer process (DBWn)** writes the contents of database buffers to data files. DBW*n* processes write modified buffers in the database buffer cache to disk (see "Database Buffer Cache").

Although one database writer process (DBW0) is adequate for most systems, you can configure additional processes—DBW1 through DBW9 and DBWa through DBWj—to improve write performance if your system modifies data heavily. These additional DBW*n* processes are not useful on uniprocessor systems.

The DBW*n* process writes dirty buffers to disk under the following conditions:

- When a server process cannot find a clean reusable buffer after scanning a threshold number of buffers, it signals DBW*n* to write. DBW*n* writes dirty buffers to disk asynchronously if possible while performing other processing.
- DBW*n* periodically writes buffers to advance the checkpoint, which is the position in the redo thread from which instance recovery begins (see "Overview of Checkpoints"). The log position of the checkpoint is determined by the oldest dirty buffer in the buffer cache.

In many cases the blocks that DBW*n* writes are scattered throughout the disk. Thus, the writes tend to be slower than the sequential writes performed by LGWR. DBW*n* performs multiblock writes when possible to improve efficiency. The number of blocks written in a multiblock write varies by operating system.

See Also:

Oracle Database Performance Tuning Guide for advice on configuring,

- 在实例恢复期间，由于读文件或表空间脱机错误而跳过的已终止事务，由 SMON 进行恢复。当表空间或文件重新联机时，SMON 将恢复该事务。
- 清理未使用的临时段。例如， Oracle 数据库在创建索引时会分配扩展区。如果操作失败，则 SMON 会清理临时空间。
- 合并在字典管理的表空间中的多个连续空闲扩展区。

SMON 定期检查是否需要做什么。其他进程在发现需要做什么时，也可以调用 SMON 来做。

数据库写入器进程 （DBWn）

数据库写入器进程 （DBWn） 将数据库缓冲区的内容写入到数据文件中。DBWn 进程将数据库缓存中被修改的缓冲区写入到磁盘中（请参见"数据库缓冲区高速缓存"）。

虽然一个数据库写入进程 （DBW0） 对于大多数系统来说已经足够，但如果您的系统大量修改数据，您也可以配置额外的写入器进程——从 DBW1 到 DBW9，和从 DBWa 到 DBWj ——来提高写入性能。这些额外的 DBWn 进程在单处理器系统上用处不大。

DBWn 进程在下列条件下会将脏缓冲区写入到磁盘：

- 当服务器进程扫描了额定数目的缓冲区后，仍未找到干净的可重复使用的缓冲区时，它会通知 DBWn 执行写入操作。DBWn 尽可能以异步方式将脏缓冲区写入到磁盘，以便同时能执行其他处理。
- DBWn 周期性地写出缓冲区，以推进检查点，该点是重做线程中实例恢复开始的位置（请参阅"检查点概述"）。检查点的日志位置由在缓冲区高速缓存中最老的脏缓冲区确定。

在许多情况下，DBWn 写入的块将遍布整个磁盘。因此，该写操作往往要比由 LGWR 执行的顺序写入速度慢。DBWn 进程尽可能进行多块写入，以提高效率。多块写入的数据块数量因操作系统而异。

另见：

《*Oracle 数据库性能优化指南*》有关配置、 监测、和调整 DBWn 的建议

monitoring, and tuning DBW*n*

Log Writer Process (LGWR)

The **log writer process (LGWR)** manages the redo log buffer. LGWR writes one contiguous portion of the buffer to the online redo log. By separating the tasks of modifying database buffers, performing scattered writes of dirty buffers to disk, and performing fast sequential writes of redo to disk, the database improves performance.

In the following circumstances, LGWR writes all redo entries that have been copied into the buffer since the last time it wrote:

- A user commits a transaction (see "Committing Transactions").
- An online redo log switch occurs.
- Three seconds have passed since LGWR last wrote.
- The redo log buffer is one-third full or contains 1 MB of buffered data.
- DBW*n* must write modified buffers to disk.

Before DBW*n* can write a dirty buffer, redo records associated with changes to the buffer must be written to disk (the **write-ahead protocol**). If DBW*n* finds that some redo records have not been written, it signals LGWR to write the records to disk and waits for LGWR to complete before writing the data buffers to disk.

LGWR and Commits

Oracle Database uses a **fast commit** mechanism to improve performance for committed transactions. When a user issues a `COMMIT` statement, the transaction is assigned a system change number (SCN). LGWR puts a commit record in the redo log buffer and writes it to disk immediately, along with the commit SCN and transaction's redo entries.

The redo log buffer is circular. When LGWR writes redo entries from the redo log buffer to an online redo log file, server processes can copy new entries over the entries in the redo log buffer that have been written to disk. LGWR normally writes fast enough to ensure that space is always available in the buffer for new entries, even when access to the online redo log is heavy.

The atomic write of the redo entry containing the transaction's commit record is the single event that determines the transaction has committed. Oracle Database returns a success code to the committing transaction although the data buffers have not yet been written to disk. The corresponding changes to data blocks are deferred until it is efficient for

日志写入器进程 （LGWR）

日志写入器进程 （LGWR） 管理重做日志缓冲区。LGWR 将缓冲区中的一个连续部分写入联机重做日志。通过分离各种修改数据库缓冲区任务，将脏缓冲区分散写入到磁盘，而将重做缓冲区顺序写入到磁盘，提高了数据库性能。

在以下的情况下， LGWR 将自上次最后写入以来，复制到缓冲区的所有重做条目写入联机重做日志：

- 用户提交了事务 （见"提交事务"）。
- 发生了在线重做日志切换。
- 自 LGWR 最后一次写入到现在超过了三秒。
- 重做日志缓冲区已达到三分之一满，或包含 1 MB 以上被缓冲的数据。
- DBWn 必须将修改的缓冲区写入到磁盘。

在 DBWn 可以将脏缓冲区写到磁盘之前，与该缓冲区更改相关联的重做记录必须先被写入磁盘 （**预写协议**）。如果 DBWn 发现一些重做记录尚未写入，则它通知 LGWR 将记录写入磁盘，并等待 LGWR 完成此工作，然后 DBWn 才将数据缓冲区写入磁盘。

LGWR 与提交

Oracle 数据库使用**快速提交**机制来提高已提交事务的性能。当用户发出 COMMIT 语句时，事务分配到一个系统更改号 (SCN)。LGWR 将一个提交记录记入重做日志缓冲区，连同提交 SCN 和事务的重做条目，并立即写入到磁盘。

重做日志缓冲区是循环的。当 LGWR 将重做条目从重做日志缓冲区写入到联机重做日志文件时，服务器进程可以复制新条目并覆盖已写入到磁盘的重做日志缓冲区中的条目。通常 LGWR 的写入速度足够快，以确保在缓冲区中总会有可用空间供新条目使用，即使对联机重做日志的访问很繁重时也是如此。

包含事务提交记录的重做条目的原子写入，是确定该事务已提交的唯一事件。Oracle 数据库向已提交事务返回一个成功代码，虽然数据缓冲区尚未写入到磁盘。对数据块的相应更改被延迟，直到 DBWn 在某个有利的时机将它们写到数据文件。

DBW*n* to write them to the data files.

Note:

LGWR can write redo log entries to disk before a transaction commits. The redo entries become permanent only if the transaction later commits.

When activity is high, LGWR can use **group commits**. For example, a user commits, causing LGWR to write the transaction's redo entries to disk. During this write other users commit. LGWR cannot write to disk to commit these transactions until its previous write completes. Upon completion, LGWR can write the list of redo entries of waiting transactions (not yet committed) in one operation. In this way, the database minimizes disk I/O and maximizes performance. If commits requests continue at a high rate, then every write by LGWR can contain multiple commit records.

LGWR and Inaccessible Files

LGWR writes synchronously to the active mirrored group of online redo log files. If a log file is inaccessible, then LGWR continues writing to other files in the group and writes an error to the LGWR trace file and the alert log. If all files in a group are damaged, or if the group is unavailable because it has not been archived, then LGWR cannot continue to function.

See Also:

- "How Oracle Database Writes to the Online Redo Log" and "Redo Log Buffer"
- *Oracle Database Performance Tuning Guide* for information about how to monitor and tune the performance of LGWR

Checkpoint Process (CKPT)

The **checkpoint process (CKPT)** updates the control file and data file headers with checkpoint information and signals DBW*n* to write blocks to disk. Checkpoint information includes the checkpoint position, SCN, location in online redo log to begin recovery, and so on. As shown in Figure 15-4, CKPT does not write data blocks to data files or redo blocks to online redo log files.

Figure 15-4 Checkpoint Process

注意：

LGWR 可能会在提交事务之前，将重做日志条目写入到磁盘。只有之后提交了事务，这些重做条目才会成为永久性的。

当事务活动很高时， LGWR 可能会使用**组提交**。例如，某个用户提交其事务，导致 LGWR 将事务的重做条目写入到磁盘。在此写操作的过程中，其他用户也试图提交。但 LGWR 无法写入磁盘以提交这些事务，直到前面的写入完成为止。完成后, LGWR 可以将（尚未提交的)等待事务中的重做条目列表在一个操作中全部写入。通过这种方式，数据库最小化了磁盘 I/O，而最大化了性能。如果提交请求继续维持在一个高的水平，则每个 LGWR 写入操作都可能包含多个提交记录。

LGWR 和无法访问的文件

LGWR 同步写入联机重做日志文件组的所有活动镜像。如果某个日志文件不可访问，则 LGWR 继续写入组中的其他文件，并在 LGWR 跟踪文件和警报日志中记录一个错误。如果组中的所有文件都损坏，或者如果因为尚未存档而不可用，则 LGWR 不能继续工作。

另见：

- "Oracle 数据库如何写入联机重做日志"和"重做日志缓冲区"
- 《*Oracle 数据库性能优化指南*》有关如何监视和调整的 LGWR 性能的信息

检查点进程（CKPT）

检查点进程 (CKPT) 使用检查点信息更新控制文件和数据文件头，并通知 DBWn 将块写入磁盘。检查点信息包括检查点位置、SCN、联机重做日志中的起始恢复位置等等。如图 15-4 所示， CKPT 不会将数据块写入数据文件，也不会将重做块写入联机重做日志文件。

图 15-4 检查点进程

Description of "Figure 15-4 Checkpoint Process"

See Also:

"Overview of Checkpoints"

Manageability Monitor Processes (MMON and MMNL)

The **manageability monitor process (MMON)** performs many tasks related to the Automatic Workload Repository (AWR). For example, MMON writes when a **metric** violates its threshold value, taking snapshots, and capturing statistics value for recently modified SQL objects.

The **manageability monitor lite process (MMNL)** writes statistics from the Active Session History (ASH) buffer in the SGA to disk. MMNL writes to disk when the ASH buffer is full.

See Also:

"Automatic Workload Repository (AWR)" and "Active Session History (ASH)"

Recoverer Process (RECO)

In a distributed database, the **recoverer process (RECO)** automatically resolves failures in distributed transactions. The RECO process of a node automatically connects to other databases involved in an in-doubt distributed transaction. When RECO reestablishes a connection between the databases, it automatically resolves all in-doubt transactions, removing from each database's pending transaction table any rows that correspond to the resolved transactions.

Description of "Figure 15-4 Checkpoint Process"

另见：

"检查点概述"

可管理性监视器进程 （MMON 和 MMNL）

可管理性监视器进程 （MMON） 执行许多与自动工作负载存储库 （AWR) 相关的任务。例如， 当某个度量超出其阈值时，MMON 会写入信息，并拍摄快照，捕获最近修改的 SQL 对象的统计信息。

可管理性监视器轻量进程 （MMNL） 将 SGA 中的活动会话历史 (ASH) 缓冲区统计信息写入磁盘。当 ASH 缓冲区已满时，MMNL 将其写入到磁盘。

另见：

"自动工作负载存储库(AWR)"和"活动会话历史 (ASH)"

恢复器进程（RECO）

在一个分布式数据库中，**恢复器进程 （RECO）** 会自动解决在分布式事务中的故障。一个节点的 RECO 进程会自动连接到可疑分布式事务中涉及的其他数据库。当 RECO 数据库之间重新建立了连接后，它会自动解决所有可疑事务，从每个数据库事务表中删除任何与可疑事务对应的未决事务行。

See Also:

Oracle Database Administrator's Guide for more information about transaction recovery in distributed systems

Optional Background Processes

An **optional background process** is any background process not defined as mandatory. Most optional background processes are specific to tasks or features. For example, background processes that support Oracle Streams Advanced Queuing (AQ) or Oracle Automatic Storage Management (Oracle ASM) are only available when these features are enabled.

This section describes some common optional processes:

- Archiver Processes (ARC*n*)
- Job Queue Processes (CJQ0 and Jnnn)
- Flashback Data Archiver Process (FBDA)
- Space Management Coordinator Process (SMCO)

See Also:

- "Oracle Streams Advanced Queuing (AQ)"
- *Oracle Database Reference* for descriptions of background processes specific to AQ and ASM

Archiver Processes (ARC*n*)

The **archiver processes (ARCn)** copy online redo log files to offline storage after a redo log switch occurs. These processes can also collect transaction redo data and transmit it to standby database destinations. ARC*n* processes exist *only* when the database is in ARCHIVELOG mode and automatic archiving is enabled.

See Also:

- "Archived Redo Log Files"

- *Oracle Database Administrator's Guide* to learn how to adjust the number of archiver processes
- *Oracle Database Performance Tuning Guide* to learn how to tune archiver performance

Job Queue Processes (CJQ0 and Jnnn)

Oracle Database uses **job queue processes** to run user jobs, often in batch mode. A **job** is a user-defined task scheduled to run one or more times. For example, you can use a job queue to schedule a long-running update in the background. Given a start date and a time interval, the job

另见：

《*Oracle 数据库管理员指南*》有关分布式系统中的事务恢复的详细信息

可选后台进程

可选后台进程即是未定义为强制性的后台进程。大多数可选后台进程是特定于任务或功能特性的。例如，支持 Oracle 流高级队列 (AQ) 或 Oracle 自动存储管理 (Oracle ASM) 的后台进程，只有在启用这些功能时才可用。

本节将介绍一些常见的可选流程：

- Archiver Processes (ARC*n*)
- Job Queue Processes (CJQ0 and Jnnn)
- Flashback Data Archiver Process (FBDA)
- Space Management Coordinator Process (SMCO)

另见：

- "Oracle 数据流高级排队 (AQ) "
- 《*Oracle 数据库参考*》有关特定于 AQ 和 ASM 的后台进程的说明

归档器进程（ARCn）

重做日志切换发生后，**归档器进程（ARCn）** 将联机重做日志文件复制到脱机存储。这些进程也可能会收集事务重做数据，并将其传送到备用数据库目标位置。仅当数据库处于归档模式下且启用了自动归档时，才存在 ARCn 进程。

另见：

- "归档重做日志文件"

- 《*Oracle 数据库管理员指南*》了解如何调整归档器进程的个数

- 《*Oracle 数据库性能优化指南*》了解如何调优归档器性能

作业队列进程 （CJQ0 和 Jnnn）

Oracle 数据库使用**作业队列进程**来运行用户作业，通常使用批处理模式。**作业**是一种用户定义的被计划运行一次或多次的任务。例如，您可以使用作业队列来计划一个在后台长时间运行的更新任务。给定一个开始日期和时间间隔，作业队列进程尝试在下一个间隔重新运行该作业。

queue processes attempt to run the job at the next occurrence of the interval.

Oracle Database manages job queue processes dynamically, thereby enabling job queue clients to use more job queue processes when required. The database releases resources used by the new processes when they are idle.

Dynamic job queue processes can run a large number of jobs concurrently at a given interval. The sequence of events is as follows:

- 1. The **job coordinator process (CJQ0)** is automatically started and stopped as needed by Oracle Scheduler (see "Oracle Scheduler"). The coordinator process periodically selects jobs that need to be run from the system `JOB$` table. New jobs selected are ordered by time.
- 2. The coordinator process dynamically spawns **job queue slave processes (Jnnn)** to run the jobs.
- 3. The job queue process runs one of the jobs that was selected by the CJQ0 process for execution. Each job queue process runs one job at a time to completion.
- 4. After the process finishes execution of a single job, it polls for more jobs. If no jobs are scheduled for execution, then it enters a sleep state, from which it wakes up at periodic intervals and polls for more jobs. If the process does not find any new jobs, then it terminates after a preset interval.

The initialization parameter `JOB_QUEUE_PROCESSES` represents the maximum number of job queue processes that can concurrently run on an instance. However, clients should not assume that all job queue processes are available for job execution.

Note:

The coordinator process is not started if the initialization parameter `JOB_QUEUE_PROCESSES` is set to 0.

See Also:

- *Oracle Database Administrator's Guide* to learn about Oracle Scheduler jobs
- *Oracle Streams Advanced Queuing User's Guide* to learn about AQ background processes

Flashback Data Archiver Process (FBDA)

The **flashback data archiver process (FBDA)** archives historical rows

Oracle 数据库动态管理作业队列进程，以使作业队列客户端在需要时能使用更多作业队列进程。当这些新启动的进程处于空闲状态时，数据库会释放其资源。

动态作业队列进程可能会在某个给定的时间间隔同时运行大量的作业。事件的顺序如下所示：

- 1. Oracle 调度器（请参见"Oracle 调度器"）根据需要会自动启动和停止**作业协调器进程 (CJQ0)**。协调器进程定期从系统表 `JOB$`中选择需要运行的作业。被选出的新作业是按时间排序的。
- 2. 协调器进程动态派生出**作业队列从属进程 (Jnnn)** 来运行作业。
- 3. 作业队列进程运行由 CJQ0 进程选出以备执行的某个作业。每个作业队列进程一次运行一个作业直至完成。
- 4. 在进程完成单个作业的执行后，它会轮询以运行更多工作。如果没有计划的作业可供执行，则它进入睡眠状态，且周期性地醒来再次轮询以运行更多工作。如果该进程未找到任何新的作业，则它在一个预设的时间间隔后会终止。

初始化参数 JOB_QUEUE_PROCESSES 表示在实例上可以同时运行的作业队列进程的最大数目。然而，客户端不应假定所有作业队列进程都可用于执行作业。

注意：

如果初始化参数 JOB_QUEUE_PROCESSES 被设置为 0，则不会启动协调器进程。

另见：

- 《*Oracle 数据库管理员指南*》了解 Oracle 调度器作业
- 《*Oracle 数据流高级队列用户指南*》了解 AQ 后台进程

闪回数据存档器进程 （FBDA）

闪回数据归档器进程 （FBDA） 将跟踪表的历史行归档到闪回数据归档

of tracked tables into Flashback Data Archives. When a transaction containing DML on a tracked table commits, this process stores the pre-image of the rows into the Flashback Data Archive. It also keeps metadata on the current rows.

区。当跟踪表中包含 DML 的事务提交时，该进程将行的前像存储到闪回数据存档区。它也会保存当前行上的元数据。

FBDA automatically manages the flashback data archive for space, organization, and retention. Additionally, the process keeps track of how far the archiving of tracked transactions has occurred.

FBDA 自动管理闪回数据存档的空间、 组织、和保留期等。此外，该进程会跟踪被跟踪的事务归档已发生了多久。

Space Management Coordinator Process (SMCO)

The SMCO process coordinates the execution of various space management related tasks, such as proactive space allocation and space reclamation. SMCO dynamically spawns slave processes (W*nnn*) to implement the task.

空间管理协调器进程 (SMCO)

SMCO 进程协调各种空间管理相关的任务，比如主动预防性的空间分配和空间回收。SMCO 动态生成从属进程 （Wnnn） 来执行此任务。

See Also:

Oracle Database Advanced Application Developer's Guide to learn about Flashback Data Archive

另见：

《*Oracle 数据库高级应用开发指南*》了解闪回数据归档

Slave Processes

Slave processes are background processes that perform work on behalf of other processes. This section describes some slave processes used by Oracle Database.

See Also:

Oracle Database Reference for descriptions of Oracle Database slave processes

从属进程

从属进程是为其他进程执行工作的后台进程。本部分介绍一些 Oracle 数据库使用的从属进程。

另见：

《*Oracle 数据库参考*》关于 Oracle 数据库从属进程的描述

I/O Slave Processes

I/O slave processes (Innn) simulate asynchronous I/O for systems and devices that do not support it. In **asynchronous I/O**, there is no timing requirement for transmission, enabling other processes to start before the transmission has finished.

I/O 从属进程

I/O 从属进程 （Innn） 为不支持异步 I/O 的系统和设备模拟异步 I/O。异步的 I/O 没有传输时间要求，使其他进程可以在完成传输之前就可以开始。

For example, assume that an application writes 1000 blocks to a disk on an operating system that does not support asynchronous I/O. Each write occurs sequentially and waits for a confirmation that the write was successful. With asynchronous disk, the application can write the blocks in bulk and perform other work while waiting for a response from the operating system that all blocks were written.

例如，假定应用程序在一个不支持异步 I/O 的操作系统上写入 1000 个块到磁盘。每次写入都是顺序发生的，并等待一个写操作已成功的确认。而对于异步磁盘，应用程序可以大容量写入数据块，并在等待操作系统确认写入完的同时执行其他工作。

To simulate asynchronous I/O, one process oversees several slave processes. The **invoker process** assigns work to each of the slave processes, who wait for each write to complete and report back to the invoker when done. In true asynchronous I/O the operating system waits for the I/O to complete and reports back to the process, while in

为模拟异步 I/O，由一个进程监督几个从属进程。**调用者进程**将工作分配给每个从属进程，从属进程会等待每个写操作完成，并在完成后报告调用者。在真正的异步 I/O 中，操作系统会等待 I/O 完成，并在完成时报告主进程，而在模拟异步 I/O 中，从属进程会等待，并在完成时报告调用者。

simulated asynchronous I/O the slaves wait and report back to the invoker.

The database supports different types of I/O slaves, including the following:

- I/O slaves for Recovery Manager (RMAN)

  When using RMAN to back up or restore data, you can make use of I/O slaves for both disk and tape devices.
- Database writer slaves

  If it is not practical to use multiple database writer processes, such as when the computer has one CPU, then the database can distribute I/O over multiple slave processes. DBWR is the only process that scans the buffer cache LRU list for blocks to be written to disk. However, I/O slaves perform the I/O for these blocks.

See Also:

- *Oracle Database Backup and Recovery User's Guide* to learn more about I/O slaves for backup and restore operations
- *Oracle Database Performance Tuning Guide* to learn more about database writer slaves

Parallel Query Slaves

In **parallel execution** or **parallel processing**, multiple processes work together simultaneously to run a single SQL statement. By dividing the work among multiple processes, Oracle Database can run the statement more quickly. For example, four processes handle four different quarters in a year instead of one process handling all four quarters by itself.

Parallel execution reduces response time for data-intensive operations on large databases such as data warehouses. Symmetric multiprocessing (SMP) and clustered system gain the largest performance benefits from parallel execution because statement processing can be split up among multiple CPUs. Parallel execution can also benefit certain types of OLTP and hybrid systems.

In Oracle RAC systems, the service placement of a particular service controls parallel execution. Specifically, parallel processes run on the nodes on which you have configured the service. By default, Oracle Database runs the parallel process only on the instance that offers the service used to connect to the database. This does not affect other parallel operations such as parallel recovery or the processing of

数据库支持包括以下不同类型的 I/O 从属进程：

- 恢复管理器 (RMAN)的 I/O 从属进程

  当使用 RMAN 备份或还原数据时，您可以为磁盘和磁带设备使用 I/O 从属进程。
- 数据库写入器从属进程

  如果计算机只有一个 CPU，则数据库使用多个写入器进程并不现实，此时数据库可以通过多个从属进程来分布 I/O。DBWR 是扫描缓冲区高速缓存 LRU 列表中的块以写入到磁盘的唯一进程。而 I/O 从属进程为这些块执行 I/O。

另见：

- 《*Oracle 数据库备份和恢复用户指南*》了解备份和还原操作中的 I/O 从属进程
- 《*Oracle 数据库性能优化指南*》了解数据库写入器从属进程

并行查询从属进程

在**并行执行**或**并行处理**中，多个进程同时协作，来运行一个单一 SQL 语句。通过在多个进程之间分配工作，Oracle 数据库可以更快地运行语句。例如，四个进程分别处理一年中四个不同的季度，而不是由一个进程处理所有四个季度。

并行执行减少了在大型数据库（如数据仓库）上的数据密集型操作的响应时间。对称多处理 (SMP) 和集群系统能从并行执行获得最大的性能优势，因为语句处理可以在多个 CPU 之间划分。并行执行也可以使某些类型的 OLTP 和混合系统受益。

在 Oracle RAC 系统中，由特定服务的服务配置控制并行执行。具体而言，并行进程在已配置服务的多个节点上运行。默认情况下，Oracle 数据库只在提供数据库连接服务的实例上运行并行进程。这并不影响其他并行操作，如并行恢复或对 GV$查询的处理。

GV$ queries.

See Also:

- *Oracle Database Data Warehousing Guide* and *Oracle Database VLDB and Partitioning Guide* to learn more about parallel execution
- *Oracle Real Application Clusters Administration and Deployment Guide* for considerations regarding parallel execution in Oracle RAC environments

Serial Execution

In **serial execution**, a single server process performs all necessary processing for the sequential execution of a SQL statement. For example, to perform a full table scan such as `SELECT * FROM employees`, one server process performs all of the work, as shown in Figure 15-5.

Figure 15-5 Serial Full Table Scan

Description of "Figure 15-5 Serial Full Table Scan "

Parallel Execution

In **parallel execution**, the server process acts as the **parallel execution coordinator** responsible for parsing the query, allocating and controlling the slave processes, and sending output to the user. Given a query plan for a SQL query, the coordinator breaks down each operator in a SQL query into parallel pieces, runs them in the order specified in the query, and integrates the partial results produced by the slave processes executing the operators.

Figure 15-6 shows a parallel scan of the `employees` table. The table is divided dynamically (**dynamic partitioning**) into load units called **granules**. Each granule is a range of data blocks of the table read by a single slave process, called a **parallel execution server**, which uses `Pnnn` as a name format.

Figure 15-6 Parallel Full Table Scan

另见：

- 《*Oracle 数据库数据仓库指南*》和 《*Oracle 数据库 VLDB 及分区指南*》了解并行执行的更多信息
- 《*Oracle 真正应用集群管理和部署指南*》关于在 Oracle RAC 环境中的并行执行的注意事项

串行执行

在**串行执行**中，由单个服务器进程，执行顺序执行的某个 SQL 语句的所有必要处理。例如，执行一个全表扫描（如 `SELECT * FROM employees`），由一个服务器进程执行所有的工作，如图 15-5 所示。

图 15-5 串行全表扫描

Description of "Figure 15-5 Serial Full Table Scan "

并行执行

在**并行执行**中，服务器进程作为**并行执行协调器**，负责解析查询、分配和控制从属进程、并将输出发送给用户。给定一个 SQL 查询的查询计划，协调器将 SQL 查询中的每个运算符分解成一些平行的片断、 并按查询中指定的顺序运行它们，最后又将这些执行各个操作的从属进程所产生的部分结果集成起来。

图 15-6 显示了在 `employees` 表上的并行扫描。表被动态划分 （**动态分区**）成多个称为**颗粒**的加载单元。每个颗粒是一个由单个从属进程读取的数据块范围，这些从属进程叫做**并行执行服务器**，其名称格式为 Pnnn。

图 15-6 并行全表扫描

Description of "Figure 15-6 Parallel Full Table Scan"

The database maps granules to execution servers at execution time. When an execution server finishes reading the rows corresponding to a granule, and when granules remain, it obtains another granule from the coordinator. This operation continues until the table has been read. The execution servers send results back to the coordinator, which assembles the pieces into the desired full table scan.

The number of parallel execution servers assigned to a single operation is the **degree of parallelism** for an operation. Multiple operations within the same SQL statement all have the same degree of parallelism.

See Also:

- *Oracle Database VLDB and Partitioning Guide* to learn how to use parallel execution
- *Oracle Database Data Warehousing Guide* to learn about recommended initialization parameters for parallelism

Description of "Figure 15-6 Parallel Full Table Scan"

数据库在执行时将颗粒映射到执行服务器。当一个执行服务器完成对某个颗粒对应行的读取，而此时还有其它颗粒要读取时，该执行服务器从协调器获取另一个颗粒。此操作将继续，直到读取完整个表。执行服务器向协调器发送执行结果，协调器再将这些片断合并成最终所需的全表扫描结果。

分配给单个操作的并行执行服务器的数量是操作的**并行度**。在同一 SQL 语句中的多个操作都具有相同的并行度。

另见：

- 《*Oracle 数据库 VLDB 和分区指南*》学习如何使用并行执行
- 《*Oracle 数据库数据仓库指南*》了解并行度初始化参数的建议

16 Application and Networking Architecture

This chapter defines application architecture and describes how an Oracle database and database applications work in a distributed processing environment. This material applies to almost every type of Oracle Database environment.

This chapter contains the following sections:

- Overview of Oracle Application Architecture
  - Overview of Client/Server Architecture
  - Overview of Multitier Architecture
  - Overview of Grid Architecture
- Overview of Oracle Networking Architecture
  - How Oracle Net Services Works
  - The Oracle Net Listener
  - Dedicated Server Architecture
  - Shared Server Architecture
  - Database Resident Connection Pooling
- Overview of the Program Interface
  - Program Interface Structure
  - Program Interface Drivers
  - Communications Software for the Operating System

Overview of Oracle Application Architecture

In the context of this chapter, **application architecture** refers to the computing environment in which a database application connects to an Oracle database.

Overview of Client/Server Architecture

In the Oracle Database environment, the database application and the database are separated into a **client/server architecture**:

- The **client** runs the database application, for example, SQL*Plus or a Visual Basic data entry program, that accesses database information and interacts with a user.

第 16 章 应用程序和网络体系结构

本章定义应用程序体系结构，并描述 Oracle 数据库和数据库应用程序在分布式处理环境中如何工作。这些内容几乎适用于每种类型的 Oracle 数据库环境。

本章包含以下各节：

- Oracle 应用程序体系结构概述
  - 客户端/服务器体系结构概述
  - 多层体系结构概述
  - 网格体系结构概述
- Oracle 网络体系结构概述
  - Oracle 网络服务的工作原理
  - Oracle 网络监听器
  - 专用服务器体系结构
  - 共享服务器体系结构
  - 数据库驻留连接池
- 程序接口概述
  - 程序接口结构
  - 程序接口驱动程序
  - 操作系统的通信软件

Oracle 应用程序体系结构概述

在本章中，应用程序体系结构指的是连接到 Oracle 数据库的某个数据库应用程序的计算环境。

客户端/服务器体系结构概述

在 Oracle 数据库环境中，数据库应用程序和数据库被划分为**客户端/服务器体系结构**：

- **客户端**运行数据库应用程序，例如 SQL*Plus 或 Visual Basic 数据输入程序，它访问数据库中的信息，并与某个用户进行交互。

- The **server** runs the Oracle Database software and handles the functions required for concurrent, shared data access to an Oracle database.

Although the client application and database can run on the same computer, greater efficiency is often achieved when the client portions and server portion are run by different computers connected through a network. The following sections discuss variations in the Oracle Database client/server architecture.

Distributed Processing

Using multiple hosts to process an individual task is known as distributed processing. Front-end and back-end processing occurs on different computers. In Figure 16-1, the client and server are located on different hosts connected through Oracle Net Services.

Figure 16-1 Client/Server Architecture and Distributed Processing

Description of "Figure 16-1 Client/Server Architecture and Distributed Processing"

Figure 16-2 is a variation that depicts a distributed database. In this example, a database on one host accesses data on a separate database located on a different host.

Figure 16-2 Client/Server Architecture and Distributed Database

- **服务器**运行 Oracle 数据库软件，并处理对 Oracle 数据库进行并发、共享的数据访问所需的功能。

虽然客户端应用程序和数据库可以在同一台计算机上运行，但当客户端部分和服务器部分在通过网络连接的不同计算机上运行时，往往会取得更高的效能。以下各节讨论了各种 Oracle 数据库客户端/服务器体系结构的变体。

分布式处理

使用多个主机来处理单个任务被称为分布式处理。前端处理和后端处理发生在不同的计算机上。在图 16-1 中，客户端和服务器位于通过 Oracle 网络服务连接的不同主机上。

图 16-1 客户端/服务器体系结构和分布式处理

Description of "Figure 16-1 Client/Server Architecture and Distributed Processing"

图 16-2 描述了分布式数据库的一种变体。在本示例中，在一个主机上的数据库，访问位于另一台主机上的一个不同数据库中的数据。

图 16-2 客户端/服务器体系结构和分布式数据库

Description of "Figure 16-2 Client/Server Architecture and Distributed Database"

Note:
This rest of this chapter applies to environments with one database on one server.

Advantages of a Client/Server Architecture

Oracle Database client/server architecture in a distributed processing environment provides the following benefits:

- Client applications are not responsible for performing data processing. Rather, they request input from users, request data from the server, and then analyze and present this data using the display capabilities of the client workstation or the terminal (for example, using graphics or spreadsheets).
- Client applications are not dependent on the physical location of the data. Even if the data is moved or distributed to other database servers, the application continues to function with little or no modification.
- Oracle Database exploits the multitasking and shared-memory facilities of its underlying operating system. As a result, it delivers the highest possible degree of concurrency, data integrity, and performance to its client applications.
- Client workstations or terminals can be optimized for the presentation of data (for example, by providing graphics and mouse support), while the server can be optimized for the processing and

Description of "Figure 16-2 Client/Server Architecture and Distributed Database"

注意：
本章的余下部分适用于在一台服务器上的一个数据库的环境。

客户端/服务器体系结构的优点

分布式处理环境中的 Oracle 数据库客户端/服务器体系结构，提供了以下好处：

- 客户端应用程序不负责执行数据处理。他们从用户请求输入、 从服务器请求数据、然后使用客户端工作站或终端（例如，使用图形或电子表格）的显示功能，分析并显示这些数据。
- 客户端应用程序不依赖于数据的物理位置。即使数据被移动或分发到其他数据库服务器，应用程序不用修改或只需很少修改，就可以继续正常工作。
- Oracle 数据库利用其底层操作系统的多任务和共享内存设施。其结果是，它为客户端应用程序提供了尽可能高的并发性、 数据完整性、和性能。
- 客户端工作站或终端可以对数据显示进行优化（例如，通过提供图形和鼠标支持)，而服务器可以对数据处理和存储进行优化（例如，通

storage of data (for example, by having large amounts of memory and disk space).

- In networked environments, you can use inexpensive client workstations to access the remote data of the server effectively.

- The database can be **scaled** as your system grows. You can add multiple servers to distribute the database processing load throughout the network (**horizontally scaled**), or you can move the database to a minicomputer or mainframe to take advantage of a larger system's performance (**vertically scaled**). In either case, data and applications are maintained with little or no modification because Oracle Database is portable between systems.
- In networked environments, shared data is stored on the servers rather than on all computers, making it easier and more efficient to manage concurrent access.
- In networked environments, client applications submit database requests to the server using SQL statements. After it is received, each SQL statement is processed by the server, which returns results to the client. Network traffic is minimized because only the requests and the results are shipped over the network.

See Also:

Oracle Database Administrator's Guide to learn more about distributed databases

Overview of Multitier Architecture

In a traditional multitier architecture environment, an application server provides data for clients and serves as an interface between clients and database servers. This architecture enables use of an application server to:

- Validate the credentials of a client, such as a Web browser
- Connect to a database server
- Perform the requested operation

An example of a multitier architecture appears in Figure 16-3.

Figure 16-3 A Multitier Architecture Environment

过拥有大量内存和磁盘空间）。

- 在网络化的环境中，您可以使用廉价的客户端工作站来有效地访问服务器上的远程数据。

- 随着您的系统规模增大，数据库可以**扩展**。您可以添加多个服务器，以便能够通过网络分散数据库处理负载（**水平扩展**），或将数据库移到小型机或大型机，以利用更大型系统的性能 （**垂直扩展**） 。在任何一种情况下，对数据和应用程序的维护基本不需要进行修改或只需要很少的修改，因为 Oracle 数据库在多个系统之间是可移植的。

- 在网络化的环境中，共享数据存储在服务器上，而不是在所有计算机上，这使管理并发访问变得更容易、 更高效。

- 在网络化的环境中，客户端应用程序使用 SQL 语句向服务器提交数据库请求。服务器收到请求后，会处理每个 SQL 语句并将结果返回给客户端。由于只有请求及其结果在网络上传输，网络通信量被最小化了。

另见：

《*Oracle 数据库管理员指南*》了解分布式数据库的更多信息

多层体系结构概述

在传统的多层体系结构环境中，应用程序服务器为客户端提供数据，并作为客户端和数据库服务器之间的接口。这种体系结构使用应用程序服务器来：

- 验证客户端（比如 Web 浏览器）凭据
- 连接到数据库服务器
- 执行请求的操作

图 16-3 显示了一个多层体系结构的示例。

图 16-3 多层体系结构环境

Description of "Figure 16-3 A Multitier Architecture Environment"

Clients

A client initiates a request for an operation to be performed on the database server. The client can be a Web browser or other end-user program. In a multitier architecture, the client connects to the database server through one or more application servers.

Application Servers

An application server provides access to the data for the client. It serves as an interface between the client and one or more database servers, and hosts the applications.

An application server permits **thin clients**, which are clients equipped with minimal software configurations, to access applications without requiring ongoing maintenance of the client computers. The application server can also perform some data reformatting for the client, reducing the load on the client workstation.

The application server assumes the identity of the client when it is performing operations on the database server for that client. The privileges of the application server should be restricted to prevent it from performing unneeded and unwanted operations during a client operation.

Database Servers

A database server provides the data requested by an application server on

Description of "Figure 16-3 A Multitier Architecture Environment"

客户端

客户端向数据库服务器发起一个请求，以在其上执行某个操作。客户端可能是一个 Web 浏览器或其他终端用户程序。在一个多层体系结构中，客户端通过一个或多个应用程序服务器连接到数据库服务器。

应用程序服务器

应用程序服务器为客户端提供了对数据的访问。它作为客户端和一个或多个数据库服务器之间的接口，并承载应用程序。

应用程序服务器允许多个**瘦客户端**访问，它们只具有最少的软件配置，可以访问应用程序而无需在客户端计算机上进行持续维护。应用程序服务器也可以为客户端执行一些数据格式化，以减少客户端工作站上的负荷。

当应用服务器为客户机在数据库服务器上执行操作时，应用程序服务器将假定以客户端的身份来执行。应用程序服务器的权限应该有所限制，以防止在客户端操作期间执行不需要和不必要的操作。

数据库服务器

数据库服务器为代表客户端的应用程序服务器提供其请求的数据。数据库执

behalf of a client. The database performs all of the query processing. The database server can audit operations performed by the application server on behalf of clients and operations performed by the application server on its own behalf (see "Monitoring"). For example, a client operation can request information to display on the client, while an application server operation can request a connection to the database server.

行所有的查询处理。
数据库服务器可以审计代表客户端的应用程序服务器所执行的操作，和代表其自身（请参阅"监视"）的应用程序服务器所执行的操作。例如，客户端可能会请求信息并在客户端上显示，而应用程序服务器可能会请求连接到数据库服务器。

Service Oriented Architecture (SOA)

The database can serve as a Web service provider in traditional multitier or **service-oriented architecture (SOA)** environments. SOA is a multitier architecture relying on **services** that support computer-to-computer interaction over a network. The services can be dynamically discovered and queried on available functions and calling sequences. SOA services are usually implemented as Web services accessible through the HTTP protocol. They are based on XML standards such as WSDL and SOAP.

The Oracle Database Web service capability, which is implemented as part of XML DB, must be specifically enabled by the DBA. Applications can then accomplish the following through database Web services:

- Submit SQL or XQuery queries and receive results as XML
- Invoke standalone PL/SQL functions and receive results (see "PL/SQL Subprograms")
- Invoke PL/SQL package functions and receive results

Database Web services provide a simple way to add Web services to an application environment without the need for an application server. However, invoking Web services through application servers such as Oracle Fusion Middleware offers security, scalability, UDDI registration, and reliable messaging in an SOA environment. However, because database Web services integrate easily with Oracle Fusion Middleware, they may be appropriate for optimizing SOA solutions.

See Also:

- *Oracle XML DB Developer's Guide* for information on enabling and using database Web services
- Oracle Fusion Middleware documentation for more information on SOA and Web services

面向服务的体系结构 (SOA)

在传统的多层环境或**面向服务的体系结构 (SOA)** 环境中，数据库可以作为 Web 服务提供者来工作。SOA 是一个依赖于**服务**的多层体系结构，以支持在网络上计算机到计算机的交互。这些服务的可用功能及其调用顺序可以被动态发现和查询。

SOA 服务通常是作为可通过 HTTP 协议访问的 Web 服务来实现的。他们基于 XML 标准，如 WSDL 和 SOAP 等。

Oracle 数据库的 Web 服务能力是作为 XML DB 的一部分来实现的，这必须专门由 DBA 来启用。然后应用程序就可以通过数据库的 Web 服务来完成以下工作：

- 提交 SQL 或 XQuery 查询，并按 XML 接收其结果
- 调用独立 PL/SQL 函数，并接收其结果 (见"PL/SQL 子程序")
- 调用 PL/SQL 包函数，并接收其结果

数据库的 Web 服务提供了一个简单的方法，以将 Web 服务添加到一个应用程序环境，而不需要应用程序服务器。然而，通过应用程序服务器（如 Oracle 融合中间件）调用 Web 服务，可在 SOA 环境中提供安全性、 可扩展性、 UDDI 注册、和可靠消息等。但是，由于数据库的 Web 服务能够方便地与 Oracle 融合中间件集成，他们可能有助于优化你的 SOA 解决方案。

另见：

- 《*Oracle XML 数据库开发人员指南*》关于如何启用和使用数据库的 Web 服务的信息
- Oracle 融合中间件文档关于 SOA 和 Web 服务的详细信息

Overview of Grid Architecture

In an Oracle Database environment, **grid computing** is a computing architecture that effectively pools large numbers of servers and storage into a flexible, on-demand computing resource. Modular hardware and software components can be connected and rejoined on demand to meet the changing needs of businesses.

See Also:

"Overview of Grid Computing" for more detailed information about server and storage grids

Overview of Oracle Networking Architecture

Oracle Net Services is a suite of networking components that provides enterprise-wide connectivity solutions in distributed, heterogeneous computing environments. Oracle Net Services enables a network session from an application to a database instance and a database instance to another database instance.

Oracle Net Services provides location transparency, centralized configuration and management, and quick installation and configuration. It also lets you maximize system resources and improve performance. The Oracle Database shared server architecture increases the scalability of applications and the number of clients simultaneously connected to the database. The **Virtual Interface (VI)** protocol places most of the messaging burden on high-speed network hardware, freeing the CPU.

Oracle Net Services uses the communication protocols or application programmatic interfaces (APIs) supported by a wide range of networks to provide distributed database and distributed processing. After a network session is established, Oracle Net Services acts as a data courier for the client application and the database server, establishing and maintaining a connection and exchanging messages. Oracle Net Services can perform these tasks because it exists on each computer in the network.

See Also:

Oracle Database Net Services Administrator's Guide for an overview of Oracle Net architecture

How Oracle Net Services Works

Oracle Database protocols take SQL statements from the interface of the Oracle applications and package them for transmission to Oracle Database through a supported industry-standard higher level protocol or API. Replies from Oracle Database are packaged through the same higher level communications mechanism. This work occurs independently of the

网格体系结构概述

在 Oracle 数据库环境中，**网格计算**体系结构将大量的服务器和存储有效地集中到一个灵活的、 随需而变的计算资源。模块化的硬件和软件组件，可以根据需要进行连接或重组，以满足不断变化的业务需求。

另见：

"网格计算概述"关于服务器和存储网格的更详细的信息

Oracle 网络体系结构概述

Oracle 网络服务是一套网络组件，在分布式的异构计算环境中，提供企业范围的连接解决方案。Oracle 网络服务使你能够从一个应用程序会话通过网络连接到一个数据库实例，或从一个数据库实例连接到另一个数据库实例。

Oracle 网络服务提供位置透明性、 集中式的配置和管理、和快速安装和配置。它还允许您最大化系统资源利用率并提高性能。Oracle 数据库共享服务器体系结构增加了应用程序的可扩展性，和可以同时连接到数据库的客户端数量。**虚拟接口 (VI)** 协议让消息传递的大部分负担由高速网络硬件来承担，减轻了 CPU 负载。

Oracle 网络服务使用支持广泛范围的网络的通信协议或应用编程接口 (API)，以提供分布式数据库和分布式处理。建立网络会话后，Oracle 网络服务将作为客户端应用程序和数据库之间的数据传递者，建立和维持一个连接，并为其交换消息。Oracle 网络服务之所以可以执行这些任务，是因为它存在于网络中的每台计算机上。

另见：

《*Oracle 数据库网络服务管理员指南*》关于 Oracle 网络体系结构的概述

Oracle 网络服务是如何工作的

Oracle 数据库协议从 Oracle 应用程序接口接受 SQL 语句，并将其打包，以便通过一个受行业标准支持的更高级别协议或 API 传输到 Oracle 数据库。从 Oracle 数据库返回的结果会使用同样高度级别的通信机制打包。此工作的发生独立于网络操作系统。

network operating system.
Depending on the operating system that runs Oracle Database, the Oracle Net Services software of the database server could include the driver software and start an additional background process.

See Also:

Oracle Database Net Services Administrator's Guide for more information about how Oracle Net Services works

The Oracle Net Listener

The **Oracle Net Listener**, also called the **listener**, is a server-side process that listens for incoming client connection requests and manages traffic to the database. When a database instance starts, and at various times during its life, the instance contacts a listener and establishes a communication pathway to this instance.

Service registration enables the listener to determine whether a database service and its service handlers are available. A **service handler** is a dedicated server process or dispatcher that acts as a connection point to a database. During registration, the PMON process provides the listener with the instance name, database service names, and the type and addresses of service handlers. This information enables the listener to start a service handler when a client request arrives.

Figure 16-4 shows two databases, each on a separate host. The database environment is serviced by two listeners, each on a separate host. The PMON process running in each database instance communicates with both listeners to register the database.

取决于运行 Oracle 数据库的操作系统，数据库服务器的 Oracle 网络服务软件可能包含相应的驱动程序软件，并会启动一个额外的后台进程。

另见：

《*Oracle 数据库网络服务管理员指南*》有关 Oracle 网络服务如何工作的详细信息

Oracle 网络监听器

Oracle 网络监听器，也称为**监听器**，是一个服务器端进程，用于侦听传入的客户端连接请求，并管理到数据库的通信量。数据库实例启动时，及其存活期间的某些时候，实例会联系一个侦听器，并建立一个到此实例的通信路径。

服务注册使监听器可以确定一个数据库服务及其服务处理程序是否是可用的。**服务处理程序**是一个专用服务器进程，或作为到数据库的一个连接点的分派器。在注册期间， PMON 过程为监听器提供实例名称、 数据库服务名称、和服务处理程序的类型和地址。此信息使监听器能够在客户端请求到达时启动一个服务处理程序。

图 16-4 显示了分别位于两台独立主机上的两个数据库。为该数据库环境服务的是分别位于另外两台独立主机上的两个监听器。在每个数据库实例上运行的 PMON 进程都与这两个监听器进行通信，以注册其数据库。

Description of the illustration cncpt315.gif

Figure 16-4 shows a browser making an HTTP connection and a client making a database connection through a listener. The listener does not need to reside on the database host.

Figure 16-4 Listener Architecture

Description of "Figure 16-4 Listener Architecture"

The basic steps by which a client establishes a connection through a listener are:

- 1. A client process or another database requests a connection.
- 2. The listener selects an appropriate service handler to service the client request and forwards the request to the handler.
- 3. The client process connects directly to the service handler. The listener is no longer involved in the communication.

See Also:

"Overview of Client Processes" and "Overview of Server Processes"

Service Names

In the context of net services, a **service** is a set of one or more database instances. A **service name** is a logical representation of a service used for client connections.

Description of the illustration cncpt315.gif

图 16-4 显示一个浏览器发出一个 HTTP 连接，而一个客户端发出一个数据库连接，它们都通过监听器连接到数据库。监听器不必驻留在数据库主机上。

图 16-4 监听器的体系结构

Description of "Figure 16-4 Listener Architecture"

客户端通过一个监听器建立连接的基本步骤如下：

- 1. 客户端进程或另一个数据库请求一个连接。
- 2. 监听器选择一个适当的服务处理程序，来为客户端请求提供服务，并将请求转发到处理程序。
- 3. 客户端进程直接连接到服务处理程序。之后侦听器就不再参与其间的通信了。

另见：

"客户端进程概述"和"服务器进程概述"

服务名

在网络服务的环境中，**服务**是一组一个或多个数据库实例。**服务名称**是用于客户端连接的服务的逻辑表示形式。

When a client connects to a listener, it requests a connection to a service. When a database instance starts, it registers itself with a listener as providing one or more services by name. Thus, the listener acts as a mediator between the client and instances and routes the connection request to the right place.

A single service, as known by a listener, can identify one or more database instances. Also, a single database instance can register one or more services with a listener. Clients connecting to a service need not specify which instance they require.

Figure 16-5 shows one single-instance database associated with two services, `book.example.com` and `soft.example.com`. The services enable the same database to be identified differently by different clients. A database administrator can limit or reserve system resources, permitting better resource allocation to clients requesting one of these services.

Figure 16-5 Multiple Services Associated with One Database

Description of "Figure 16-5 Multiple Services Associated with One Database"

See Also:

Oracle Database Net Services Administrator's Guide to learn more about naming methods

当客户端连接到一个监听器时，它请求连接到某个服务。当一个数据库实例启动时，它提供一个或多个服务名称，以将自己注册到某个侦听器。因此，监听器充当客户端和实例之间的中间人，并将连接请求路由到正确的位置。

通过监听器识别到的一个单一的服务，可以标识一个或多个数据库实例。单个数据库实例也可以向一个侦听器注册为一个或多个服务。连接到服务的客户端不需要指明它们需要哪个实例。

图 16-5 显示了一个单实例数据库与两个服务(book.example.com 和 soft.example.com)相关联。服务使得同一数据库以被不同的客户端以不同的方式来标识。数据库管理员可以限制或预留系统资源，以便为请求这些服务的客户机提供更好的资源分配。

图 16-5 与一个数据库关联的多个服务

Description of "Figure 16-5 Multiple Services Associated with One Database"

另见：

《*Oracle 数据库网络服务管理员指南*》了解命名方法的更多信息

Service Registration

Service registration is a feature by which the PMON process dynamically registers instance information with a listener, which enables the listener to forward client connection requests to the appropriate service handler. PMON provides the listener with information about the following:

- Names of the database services provided by the database
- Name of the database instance associated with the services and its current and maximum load
- Service handlers (dispatchers and dedicated servers) available for the instance, including their type, protocol addresses, and current and maximum load

Service registration is dynamic and does not require configuration in the `listener.ora` file. Dynamic registration reduces administrative overhead for multiple databases or instances.

The initialization parameter `SERVICE_NAMES` lists the services an instance belongs to. On startup, each instance registers with the listeners of other instances belonging to the same services. During database operations, the instances of each service pass information about CPU use and current connection counts to all listeners in the same services. This communication enables dynamic load balancing and connection failover.

See Also:

- "Process Monitor Process (PMON)"
- *Oracle Database Net Services Administrator's Guide* to learn more about service registration
- *Oracle Real Application Clusters Administration and Deployment Guide* to learn about instance registration and client/service connections in Oracle RAC

Dedicated Server Architecture

In a **dedicated server architecture**, the server process created on behalf of each client process is called a **dedicated server process** (or **shadow** process). This server process is separate from the client process and acts only on its behalf, as shown in Figure 16-6.

Figure 16-6 Oracle Database Using Dedicated Server Processes

服务注册

PMON 进程通过**服务注册**功能，动态地将实例信息注册到监听器，以使监听器能将客户端的连接请求转发到适当的服务处理程序。PMON 为监听器提供了以下信息：

- 由数据库提供的数据库服务名称
- 与服务相关联的数据库实例名称，及其当前负载和最大负载
- 对实例可用的服务处理程序 （分派器和专用服务器），包括其类型、 协议地址，当前负载和最大负载

服务注册是动态的，并且不需要在 listener.ora 文件中配置。动态注册减少了管理多个数据库或实例的开销。

初始化参数 SERVICE_NAMES 列出实例所属的服务。在启动时，每个实例及其属于相同服务的其他实例，被注册到所有监听器。在数据库操作期间，每个服务的实例将 CPU 使用和当前的连接计数等有关信息，传递给监听相同服务的所有监听器。这种通信实现了动态负载平衡和连接的故障转移。

另见：

- 进程监视器进程（PMON）"
- 《*Oracle 数据库网络服务管理员指南*》了解服务注册的更多信息
- 《*Oracle 真正应用集群管理和部署指南*》了解在 Oracle RAC 中的实例注册和客户端/服务连接

专用服务器体系结构

在**专用服务器体系结构**中，为每个客户端进程创建的服务器进程称为**专用服务器进程** （或**影子**进程）。此服务器进程与客户端进程是相互分离的，只是为其服务，如图 16-6 所示。

图 16-6 Oracle 数据库使用的专用服务器进程

Description of "Figure 16-6 Oracle Database Using Dedicated Server Processes"

A one-to-one ratio exists between the client processes and server processes. Even when the user is not actively making a database request, the dedicated server process remains—although it is inactive and can be paged out on some operating systems.

Figure 16-6 shows user and server processes running on networked computers. However, the dedicated server architecture is also used if the same computer runs both the client application and the database code but the host operating system could not maintain the separation of the two programs if they were run in a single process. Linux is an example of such an operating system.

In the dedicated server architecture, the user and server processes communicate using different mechanisms:

- If the client process and the dedicated server process run on the same computer, then the program interface uses the host operating

Description of "Figure 16-6 Oracle Database Using Dedicated Server Processes"

客户端进程与服务器进程之间存在一对一的比例关系。即使当用户并未正在向数据库发起请求，专用服务器进程仍然是存在的——尽管它处于非活动状态，并在某些操作系统上可能会被换出到磁盘。

图 16-6 显示了在联网的多台计算机上运行的用户进程和服务器进程。不过，如果在同一台计算机运行客户端应用程序和数据库的代码，但如果他们在一个单一进程中运行，主机操作系统可能无法在其中维持两个程序的适度分离，则还是会使用专用服务器体系结构。Linux 就是这种操作系统的一个例子。

在专用服务器体系结构中，用户进程和服务器进程使用不同的机制进行通信：

- 如果客户端进程和专用服务器进程在同一台计算机上运行，则程序接

system's interprocess communication mechanism to perform its job.

- If the client process and the dedicated server process run on different computers, then the program interface provides the communication mechanisms (such as the network software and Oracle Net Services) between the programs.

Underutilized dedicated servers sometimes result in inefficient use of operating system resources. Consider an order entry system with dedicated server processes. A customer places an order as a clerk enters the order into the database. For most of the transaction, the clerk is talking to the customer while the server process dedicated to the clerk's client process is idle. The server process is not needed during most of the transaction, and the system may be slower for other clerks entering orders if the system is managing too many processes. For applications of this type, the shared server architecture may be preferable.

See Also:

Oracle Database Net Services Administrator's Guide to learn more about dedicated server processes

Shared Server Architecture

In a **shared server architecture**, a dispatcher directs multiple incoming network session requests to a pool of shared server processes, eliminating the need for a dedicated server process for each connection. An idle shared server process from the pool picks up a request from a common queue.

The potential benefits of shared server are as follows:

- Reduces the number of processes on the operating system

  A small number of shared servers can perform the same amount of processing as many dedicated servers.
- Reduces instance PGA memory

  Every dedicated or shared server has a PGA. Fewer server processes means fewer PGAs and less process management.
- Increases application scalability and the number of clients that can simultaneously connect to the database
- May be faster than dedicated server when the rate of client connections and disconnections is high

口使用主机操作系统的进程间通信机制来执行其工作。

- 如果客户端进程和专用服务器进程在不同的计算机上运行，则由程序接口提供程序之间的通信机制（例如，网络软件和 Oracle 网络服务）。

未充分利用的专用服务器有时会导致操作系统资源的使用效率低下。考虑一个使用专用服务器进程的订单录入系统。当某个客户提交了一个订单时，工作人员将订单输入到数据库中。在事务的大部分时间里，工作人员都在与客户交谈，而专用于工作人员客户端进程的服务器进程一直处于空闲状态。该服务器进程在事务的大部分时间里并不是需要的，为输入命令，如果系统维持了太多进程，则当别的工作人员在输入订单时，系统速度可能会变得更慢。对这种类型的应用程序，共享服务器体系结构可能更为可取。

另见：

《*Oracle 数据库网络服务管理员指南*》了解专用服务器进程

共享服务器体系结构

在**共享服务器体系结构**中，分派器将传入网络的多个会话请求定向到一个共享服务器进程池，消除了为每个连接分配一个专用服务器进程的需要。池中的一个空闲的共享服务器进程从公共请求队列取出一个请求并处理之。

共享服务器的潜在好处如下所示：

- 减少了操作系统上的进程数

  相比许多的专用服务器，只需少量的共享服务器就可以执行同样多的处理量。
- 减少了实例 PGA 内存

  每个专用服务器或共享服务器都有一个 PGA。更少的服务器进程意味着更少的 PGA 和更少的进程管理。
- 增加了应用程序的可扩展性，和可同时连接到数据库的客户端数量
- 当客户端频繁地连接和断开连接时，共享服务器的速度可能比专用服务器更快

Shared server has several disadvantages, including slower response time in some cases, incomplete feature support, and increased complexity for setup and tuning. As a general guideline, only use shared server when you have more concurrent connections to the database than the operating system can handle.

The following processes are needed in a shared server architecture:

- A network listener that connects the client processes to dispatchers or dedicated servers (the listener is part of Oracle Net Services, not Oracle Database)

  Note:

  To use shared servers, a client process must connect through Oracle Net Services, even if the process runs on the same computer as the Oracle Database instance.
- One or more dispatcher process (D*nnn*)
- One or more shared server processes

Note that a database can support both shared server and dedicated server connections simultaneously. For example, one client can connect using a dedicated server while a different client connects to the same database using a shared server.

See Also:

- *Oracle Database Net Services Administrator's Guide* for more information about the shared server architecture
- *Oracle Database Administrator's Guide* to learn how to configure a database for shared server

Dispatcher Request and Response Queues

A request from a user is a single API call that is part of the user's SQL statement. When a user makes a call, the following actions occur:

- 1. The dispatcher places the request on the **request queue**, where it is picked up by the next available shared server process.

  The request queue is in the SGA and is common to all dispatcher processes of an instance (see "Large Pool").
- 2. The shared server processes check the common request queue for new requests, picking up new requests on a first-in-first-out basis.

共享服务器有几个缺点，包括在某些情况下响应速度较慢、功能支持不完整、增加了安装和调整的复杂程度。作为一个一般的指导原则，仅当您需要并发连接到数据库的数量超过操作系统的处理能力时，才使用共享服务器。

在共享服务器体系结构中需要以下进程：

- 将客户端进程连接到分派器进程或专用服务器进程的网络监听器（监听器是 Oracle 网络服务的一部分，而不是 Oracle 数据库的一部分）

  注意：

  要使用共享服务器，客户端进程必须通过 Oracle 网络服务来连接，即使该进程与 Oracle 数据库实例在同一台计算机上运行。
- 一个或多个分派器进程 （Dnnn）
- 一个或多个共享服务器进程

请注意，数据库可以同时支持共享服务器连接和专用服务器连接。例如，一个客户端可以使用专用服务器连接，而另一个不同的客户端使用共享服务器连接到同一数据库。

另见：

- 《*Oracle 数据库网络服务管理员指南*》有关共享服务器体系结构的详细信息
- 《*Oracle 数据库管理员指南*》了解如何将数据库配置为共享服务器

分派器请求和响应队列

用户的请求是一个 API 调用，它是用户的 SQL 语句的一部分。当用户发出一个调用时，会发生以下操作：

- 1. 分派器将请求放入**请求队列**，下一个可用的共享服务器进程会从请求队列取出该请求。

  请求队列位于 SGA 中，能被实例的所有分派器进程访问 （见"大池"）。
- 2. 共享服务器进程检查公共请求队列，按先进先出的原则取出一个新请求。

- 3. One shared server process picks up one request in the queue and makes all necessary calls to the database to complete this request.

  A different server process can handle each database call. Therefore, requests to parse a query, fetch the first row, fetch the next row, and close the result set may each be processed by a different shared server.
- 4. When the server process completes the request, it places the response on the calling dispatcher's **response queue**. Each dispatcher has its own response queue.
- 5. The dispatcher returns the completed request to the appropriate client process.

For example, in an order entry system, each clerk's client process connects to a dispatcher. Each request made by the clerk is sent to this dispatcher, which places the request in the queue. The next available shared server picks up the request, services it, and puts the response in the response queue. When a request is completed, the clerk remains connected to the dispatcher, but the shared server that processed the request is released and available for other requests. While one clerk talks to a customer, another clerk can use the same shared server process. Figure 16-7 shows how client processes communicate with the dispatcher across the API and how the dispatcher communicates user requests to shared server processes.

Figure 16-7 The Shared Server Configuration and Processes

- 3. 一个共享服务器进程从队列中取出一个请求，并对数据库发出所有必要的调用以完成该请求。

  一个不同的服务器进程可能会处理数据库中的每个调用。因此，解析查询、 读取第一行、 读取下一行、和关闭结果集等每个请求都可能会由不同的共享服务器来处理。
- 4. 当服务器进程完成了请求时，它将响应结果置于分派器的**响应队列**。每个分派器都有它自己的响应队列。
- 5. 分派器将已完成的请求结果返回给适当客户端进程。

例如，一个订单录入系统，每个工作人员的客户端进程连接到一个分派器。工作人员所发出的每个请求被发送到该分派器，分派器则将此请求放入队列中。下一个可用的共享服务器取出该请求并为其服务，并将响应结果置于响应队列。当请求完成时，工作人员还与分派器保持着连接状态，但处理请求的共享服务器已经被释放并去为其他请求服务了。当工作人员与客户交谈时，另一个工作人员也可以使用同一共享服务器进程。

图 16-7 显示客户端进程如何与分派器进行跨 API 通信，及分派器如何就用户请求与共享服务器进程进行通信。

图 16-7 共享服务器配置和进程

Description of "Figure 16-7 The Shared Server Configuration and Processes"

Dispatcher Processes (D*nnn*)

The **dispatcher processes** enable client processes to share a limited number of server processes. You can create multiple dispatcher processes for a single database instance. The optimum number of dispatcher processes depending on the operating system limitation and the number of connections for each process.

Note:

Each client process that connects to a dispatcher must use Oracle Net Services, even if both processes run on the same host.

Dispatcher processes establish communication as follows:

- 1. When an instance starts, the network listener process opens and establishes a communication pathway through which users connect to Oracle Database.
- 2. Each dispatcher process gives the listener process an address at which the dispatcher listens for connection requests.

  At least one dispatcher process must be configured and started for

Description of "Figure 16-7 The Shared Server Configuration and Processes"

分派器进程(Dnnn)

分派器进程使得客户端进程可以共享数量有限的服务器进程。您可以为单个数据库实例创建多个分派器进程。分派器进程的最佳个数取决于操作系统的限制和每个进程的连接数。

注意：

每个连接到分派器的客户端进程必须使用 Oracle 网络服务，即使这两个进程运行在同一主机上。

分派器进程建立通信的过程如下所示：

- 1. 当实例启动时，网络监听器进程被打开，并建立一条通过用户连接到 Oracle 数据库的通信路径。
- 2. 每一个分派器进程将其监听连接请求的地址提供给监听器进程。

  必须至少为每个数据库客户端将使用的网络协议配置并启动一个分派

each network protocol that the database clients will use.

- 3. When a client process makes a connection request, the listener determines whether the client process should use a shared server process:
  - If the listener determines that a shared server process is required, then the listener returns the address of the dispatcher process that has the lightest load, and the client process connects to the dispatcher directly.
  - If the process cannot communicate with the dispatcher, or if the client process requests a dedicated server, then the listener creates a dedicated server and establishes an appropriate connection.

See Also:

Oracle Database Net Services Administrator's Guide to learn how to configure dispatchers

Shared Server Processes (S*nnn*)

Each **shared server process** serves multiple client requests in the shared server configuration. Shared and dedicated server processes provide the same functionality, except shared server processes are not associated with a specific client process. Instead, a shared server process serves any client request in the shared server configuration.

The PGA of a shared server process does not contain UGA data, which must be accessible to all shared server processes (see "Overview of the Program Global Area"). The shared server PGA contains only process-specific data.

All session-related information is contained in the SGA. Each shared server process must be able to access all sessions' data spaces so that any server can handle requests from any session. Space is allocated in the SGA for each session's data space.

Restricted Operations of the Shared Server

Certain administrative activities cannot be performed while connected to a dispatcher process, including shutting down or starting an instance and media recovery. These activities are typically performed when connected with administrator privileges. To connect with administrator privileges in a system configured with shared servers, you must specify that you want to use a dedicated server process.

See Also:

Oracle Database Net Services Administrator's Guide for the proper connect

器进程。

- 3. 当客户端进程发出连接请求时，监听器将确定客户端进程是否应使用共享服务器进程：
  - 如果监听器确定需要使用一个共享服务器进程，则监听器返回负载最轻的分派器进程的地址，客户端进程则直接连接到该分派器。
  - 如果客户端进程不能与分派器进行通信，或者客户端进程请求的是一个专用服务器，则监听器将创建专用服务器，并建立一个适当的连接。

另见：

《*Oracle 数据库网络服务管理员指南*》了解如何配置分派器

共享服务器进程 （Snnn）

在共享服务器配置中，每个**共享服务器进程**为多个客户端请求服务。除了共享服务器进程不与特定客户端进程相关联之外，共享服务器进程和专用服务器进程提供相同的功能。实际上，共享服务器配置中的共享服务器进程可以为任何客户端请求服务。

共享服务器进程的 PGA 不包含 UGA 数据，UGA 数据必须对所有共享服务器进程可访问 （请参阅"程序全局区概述"） 。共享服务器 PGA 仅包含特定于进程的数据。

所有与会话相关的信息存储在 SGA 中。每个共享服务器进程必须能够访问所有会话的数据空间，以便任何服务器都可以处理来自任何会话的请求。每个会话的数据空间是在 SGA 中分配的。

受限制的共享服务器操作

当连接到一个分派器进程时，无法执行某些管理性活动，包括关闭或启动实例和执行介质恢复等。这些活动通常需要在具有管理员权限的连接时执行。若要在使用共享服务器配置的系统中以管理员权限来连接，则必须指定您要使用专用服务器进程。

另见：

《*Oracle 数据库网络服务管理员指南*》关于正确的连接字符串语法

string syntax

Database Resident Connection Pooling

Database Resident Connection Pooling (DRCP) provides a connection pool of dedicated servers for typical Web application scenarios. A Web application typically makes a database connection, uses the connection briefly, and then releases it. Through DRCP, the database can scale to tens of thousands of simultaneous connections.

DRCP provides the following advantages:

- Complements middle-tier connection pools that share connections between threads in a middle-tier process.
- Enables database connections to be shared across multiple middle-tier processes. These middle-tier processes may belong to the same or different middle-tier host.
- Enables a significant reduction in key database resources required to support many client connections. For example, DRCP reduces the memory required for the database and boosts the scalability of the database and middle tier. The pool of available servers also reduces the cost of re-creating client connections.
- Provides pooling for architectures with multi-process, single-threaded application servers, such as PHP and Apache, that cannot do middle-tier connection pooling.

DRCP uses a **pooled server**, which is the equivalent of a dedicated server process (not a shared server process) and a database session combined. The pooled server model avoids the overhead of dedicating a server for every connection that requires the server for a short period.

Clients obtaining connections from the database resident connection pool connect to an Oracle background process known as the **connection broker**. The connection broker implements the pool functionality and multiplexes pooled servers among inbound connections from client processes.

As shown in Figure 16-8, when a client requires database access, the connection broker picks up a server process from the pool and hands it off to the client. The client is directly connected to the server process until the request is served. After the server has finished, the server process is released into the pool. The connection from the client is restored to the broker.

Figure 16-8 DRCP

数据库驻留连接池

数据库驻留连接池 （DRCP） 为典型的 Web 应用程序场景提供专用服务器的连接池。Web 应用程序通常发起一个数据库连接、 短暂地使用该连接、然后将其释放。通过 DRCP，数据库可以扩展到能够同时接受数以万计的连接。

DRCP 提供了以下优点：

- 提供一个中间层连接池，以在中间层进程的线程之间共享连接。
- 使得数据库连接能在多个中间层进程之间共享。这些中间层进程可能在相同或不同的中间层主机上。
- 显著减少为支持众多客户端连接所需的关键数据库资源。例如，DRCP 可减少数据库的内存需求，并大大提高数据库和中间层的可扩展性。可用的服务器池还可以减少重新创建客户端连接的成本。
- 为多进程、 单线程的应用程序服务器体系结构提供连接池，如不能做中间层连接池的 PHP 和 Apache 等。

DRCP 使用**池化服务器**，这相当于一个专用服务器进程 （而不是共享服务器进程） 和数据库会话的联合体。池化服务器模型避免了为每个连接分配一个专用服务器但只需短时间使用该服务器的开销。

客户端从数据库驻留连接池获取连接，连接到一个称为**连接代理**的 Oracle 后台进程。该连接代理实现了池功能，并在来自客户端进程的多个入站连接之间多路复用池化服务器。

如图 16-8 所示，当客户端要求数据库访问时，连接代理将从池中取出一个服务器进程，并交给客户端。客户端直接连接到服务器进程，直到请求的服务被完成。服务器完成后，服务器进程将被释放到池中。与客户端的连接将被归还给代理。

图 16-8 DRCP

Description of "Figure 16-8 DRCP"

In DRCP, releasing resources leaves the session intact, but no longer associated with a connection (server process). Unlike in shared server, this session stores its UGA in the PGA, not in the SGA. A client can reestablish a connection transparently upon detecting activity.

See Also:

- "Connections and Sessions"
- *Oracle Database Administrator's Guide* and *Oracle Call Interface Programmer's Guide* to learn more about DRCP

Overview of the Program Interface

The **program interface** is the software layer between a database application and Oracle Database. The program interface performs the following functions:

- Provides a security barrier, preventing destructive access to the SGA by client client processes

Description of "Figure 16-8 DRCP"

在 DRCP 中，释放资源后使会话保持如初，但会话不再与连接（服务器进程）相关联。与共享服务器不同，此会话将其 UGA 存储在 PGA 中，而不是在 SGA 中。客户端可以通过检测后透明地重新建立连接。

另见：

- "连接和会话"
- 《*Oracle 数据库管理员指南*》和《*Oracle 调用接口程序员指南*》了解 DRCP 的更多信息

程序接口概述

程序接口是一个数据库应用程序和 Oracle 数据库之间的软件层。程序接口执行以下功能：

- 提供一个安全屏障，以防止客户端进程对 SGA 的破坏性访问

- Acts as a communication mechanism, formatting information requests, passing data, and trapping and returning errors

- Converts and translates data, particularly between different types of computers or to external user program data types

The **Oracle code** acts as a server, performing database tasks on behalf of an **application** (a client), such as fetching rows from data blocks. The program interface consists of several parts, provided by both Oracle Database software and operating system-specific software.

Program Interface Structure

The program interface consists of the following pieces:

- Oracle call interface (OCI) or the Oracle run-time library (SQLLIB)

- The client or user side of the program interface

- Various **Oracle Net Services drivers** (protocol-specific communications software)
- Operating system communications software

- The server or Oracle Database side of the program interface (also called the OPI)

The user and Oracle Database sides of the program interface run Oracle software, as do the drivers.

Program Interface Drivers

Drivers are pieces of software that transport data, usually across a network. They perform operations such as connect, disconnect, signal errors, and test for errors. Drivers are specific to a communications protocol.

There is always a default driver. You can install multiple drivers, such as the asynchronous or DECnet drivers, and select one as the default driver, but allow an user to use other drivers by specifying a driver when connecting.

Different processes can use different drivers. A process can have concurrent connections to a single database or to multiple databases using different Oracle Net Services drivers.

See Also:

- Your system installation and configuration guide for details about

- 作为一种通信机制，格式化请求的信息、传递数据、和捕获并返回错误

- 在不同类型的计算机之间转换数据，或将其翻译为外部用户程序数据类型

Oracle 代码作为一个服务器程序，为应用程序 （客户）执行数据库任务，如从数据块读取行。程序接口由 Oracle 数据库软件和特定于操作系统的软件等几个部分组成。

程序接口结构

程序接口由以下几部分组成：

- Oracle 调用接口 （OCI） 或 Oracle 运行时库 (SQLLIB)

- 程序接口的客户端或用户端

- 各种 **Oracle 网络服务驱动程序** （特定协议通信软件）

- 操作系统通信软件

- 程序接口的服务器端或 Oracle 数据库端（也称为 OPI）

用户端和程序接口的 Oracle 数据库端，都运行 Oracle 软件的驱动程序。

程序接口驱动程序

驱动程序通常是跨越网络传输数据的软件。他们执行如连接、断开连接、发出错误信号、并测试错误等操作。驱动程序是特定于通信协议的。

一般都有默认的驱动程序。您可以安装多个驱动程序，如异步驱动程序或 DECnet 驱动程序，并选择一个作为默认的驱动程序，但允许用户通过在连接时指定一个驱动程序来使用其他驱动程序。

不同的进程可以使用不同的驱动程序。一个进程可以使用不同的 Oracle 网络服务驱动程序，并发地连接到一个或多个数据库。

另见：

- 您系统的安装和配置指南，关于选择、 安装、及添加驱动程序的详

choosing, installing, and adding drivers

- *Oracle Database Net Services Administrator's Guide* to learn about JDBC drivers

Communications Software for the Operating System

The lowest-level software connecting the user side to the Oracle Database side of the program interface is the communications software, which is provided by the host operating system. DECnet, TCP/IP, LU6.2, and ASYNC are examples. The communication software can be supplied by Oracle, but it is usually purchased separately from the hardware vendor or a third-party software supplier.

细信息

- 《*Oracle 数据库网络服务管理员指南*》了解 JDBC 驱动程序

操作系统的通信软件

程序接口中将用户端连接到 Oracle 数据库端的最底层软件，是由主机操作系统提供的通信软件。例如 DECnet、 TCP/IP、 LU6.2、ASYNC 等。通讯软件可能由 Oracle 提供，但通常是从硬件供应商或第三方软件供应商单独购买的。

The script content on this page is for navigation purposes only and does not alter the content in any way.

17 Topics for Database Administrators and Developers

The previous parts of this manual described the basic architecture of Oracle Database. This chapter summarizes common database topics that are important for both database administrators and developers, and provides pointers to other manuals, not an exhaustive account of database features.

This chapter contains the following sections:

- Overview of Database Security
  - User Accounts
  - Authentication
  - Encryption
  - Access Control
  - Monitoring
- Overview of High Availability
  - High Availability and Unplanned Downtime
  - High Availability and Planned Downtime
- Overview of Grid Computing
  - Database Server Grid
  - Database Storage Grid
- Overview of Data Warehousing and Business Intelligence
  - Data Warehousing and OLTP
  - Data Warehouse Architecture
  - Overview of Extraction, Transformation, and Loading (ETL)
  - Business Intelligence
- Overview of Oracle Information Integration
  - Federated Access
  - Information Sharing

See Also:

Chapter 18, "Concepts for Database Administrators" discusses topics

此页上的脚本内容仅用于导航，并不会以任何方式改变其中的内容。

第 17 章 数据库管理员和开发人员的主题

本手册的前面部分阐述了 Oracle 数据库的基本体系结构。本章概述了对数据库管理员和开发人员都非常重要的数据库主题，并提供指向其他手册的链接。

本章包含以下各节：

- 数据库安全性概述
  - 用户账户
  - 身份验证
  - 加密
  - 访问控制
  - 监视
- 高可用性的概述
  - 高可用性和计划外停机时间
  - 高可用性和计划内停机时间
- 网格计算的概述
  - 数据库服务器网格
  - 数据库存储网格
- 数据仓库和商业智能概述
  - 数据仓库和 OLTP
  - 数据仓库体系结构
  - 提取、 转换，和加载 (ETL) 概述
  - 商务智能
- Oracle 信息集成概述
  - 联邦访问
  - 信息共享

另见：

第 18 章，"数据库管理员相关概念"讨论了特定于数据库管理员的主题。第

specific to DBAs. Chapter 19, "Concepts for Database Developers" discusses topics for developers.

Overview of Database Security

In general, **database security** involves user authentication, encryption, access control, and monitoring.

User Accounts

Each Oracle database has a list of valid database **users**. The database contains several default accounts, including the default administrative account `SYSTEM` (see "SYS and SYSTEM Schemas"). You can create user accounts as needed.

To access a database, a user must provide a valid user name and authentication credential. The credential may be a password, Kerberos ticket, or public key infrastructure (PKI) certificate. You can configure database security to lock accounts based on failed login attempts.

Privilege and Role Authorization

In general, **database access control** involves restricting data access and database activities. For example, you can restrict users from querying specified tables or executing specified database commands.

A user **privilege** is the right to run specific SQL statements. Privileges can be divided into the following categories:

- System privilege

  This is the right to perform a specific action in the database, or perform an action on any objects of a specific type. For example, `CREATE USER` and `CREATE SESSION` are system privileges.
- Object privilege

  This is the right to perform a specified action on an object, for example, query the `employees` table. Privilege types are defined by the database.

Privileges are granted to users at the discretion of other users. Administrators should grant privileges to users so they can accomplish tasks required for their jobs. Good security practice involves granting a privilege only to a user who requires that privilege to accomplish the necessary work.

A **role** is a named group of related privileges that you grant to users or other roles. A role helps manage privileges for a database application or

19 章，"数据库开发人员相关概念"讨论了与开发人员相关的主题。

数据库安全性概述

通常，**数据库安全**涉及用户身份验证、 加密、 访问控制、和监控等。

用户账户

每个 Oracle 数据库都有一个有效数据库**用户**的列表。数据库包含几个默认账户，其中包括缺省的管理帐户 `SYSTEM`（请参阅"SYS 和 SYSTEM 模式"）。您可以根据需要创建用户帐户。

若要访问数据库，用户必须提供有效的用户名称和身份验证凭据。该凭据可能是一个密码、 Kerberos 票证、或公钥基础结构 (PKI) 证书。您可以配置数据库安全，以在失败登录尝试次数超限时锁定帐户。

权限和角色授权

通常，**数据库访问控制**涉及限制数据访问和数据库活动。例如，您可以限制用户查询指定的表或执行指定的数据库命令。

用户**权限**是指运行特定 SQL 语句的权利。权限可以分为以下几类：

- 系统权限

  这是在数据库中执行特定操作或在任何特定类型的对象上执行操作的权限。例如，`CREATE USER` 和 `CREATE SESSION` 是系统权限。
- 对象权限

  这是在一个对象上执行指定的操作的权限，例如，查询 `employees` 表。权限类型是由数据库定义的。

对用户的授权是由其他用户来授予的。管理员应该向用户授予权限，使他们可以完成其工作所需的任务。良好的安全实践包括，仅将权限授予给需要该权限的用户，使其能完成必要的工作。

角色是您授予用户或其他角色的一组命名的相关权限。角色有助于管理数据库应用程序或用户组的权限。

user group.

Figure 17-1 depicts a common use for roles. The roles PAY_CLERK, MANAGER, and REC_CLERK are assigned to different users. The application role ACCTS_PAY, which includes the privilege to execute the ACCTS_PAY application, is assigned to users with the PAY_CLERK and MANAGER role. The application role ACCTS_REC, which includes the privilege to execute the ACCTS_REC application, is assigned to users with the REC_CLERK and MANAGER role.

Figure 17-1 Common Uses for Roles

Description of "Figure 17-1 Common Uses for Roles"

See Also:

- *Oracle Database 2 Day + Security Guide* and *Oracle Database Security Guide* to learn how to manage privileges
- *Oracle Database Security Guide* to learn about using roles for security
- *Oracle Database 2 Day DBA* and *Oracle Database Administrator's Guide* to learn how to administer roles
- *Oracle Database Reference* to learn about the SESSION_PRIVS view

Profiles

In the context of system resources, a **profile** is a named set of resource limits and password parameters that restrict database usage and instance resources for a user. Profiles can limit the number of concurrent sessions

图 17-1 描述了角色的常见用途。角色 PAY_CLERK、 MANAGER 和 REC_CLERK 被分配给不同的用户。应用程序角色 ACCTS_PAY 包括执行 ACCTS_PAY 应用程序的权限，该角色被分配给具有 PAY_CLERK 和 MANAGER 角色的用户。应用程序角色 ACCTS_REC 包括执行 ACCTS_REC 应用程序的权限，该角色被分配给具有 REC_CLERK 和 MANAGER 角色的用户。

图 17-1 角色的常见用途

Description of "Figure 17-1 Common Uses for Roles"

另见：

- 《*Oracle 数据库 2 日+ 安全指南*》和《*Oracle 数据库安全指南*》了解如何管理权限
- 《*Oracle 数据库安全指南*》了解将角色用于安全
- 《*Oracle 数据库 2 日 DBA*》和《*Oracle 数据库管理员指南*》学习如何管理角色
- 《*Oracle 数据库参考*》了解 SESSION_PRIVS 视图

概要文件

在系统资源相关上下文中，**概要文件**是一个资源限制和密码参数的命名集合，用于限制某个用户的数据库和实例资源的使用情况。概要文件可以限制

for a user, CPU processing time available for each session, and amount of logical I/O available (see "Buffer I/O"). For example, the `clerk` profile could limit a user to system resources required for clerical tasks.

Note:

It is preferable to use Database Resource Manager to limit resources and to use profiles to manage passwords.

Profiles provide a single point of reference for users that share a set of attributes. You can assign a profile to one set of users, and a default profile to all others. Each user has at most one profile assigned at any point in time.

See Also:

- *Oracle Database Security Guide* to learn how to manage resources with profiles
- *Oracle Database SQL Language Reference* for `CREATE PROFILE` syntax and semantics

Authentication

Authentication is the process by which a user presents credentials to the database, which verifies the credentials and allows access to the database. Validating the identity establishes a trust relationship for further interactions. Authentication also enables accountability by making it possible to link access and actions to specific identities.

Oracle Database provides different authentication methods, including the following:

- Authentication by the database

  Oracle database can authenticate users using a password, Kerberos ticket, or PKI certificate. Oracle also supports RADIUS-compliant devices for other forms of authentication, including biometrics. The type of authentication must be specified when a user is create in the Oracle database.
- Authentication by the operating system

  Some operating systems permit Oracle Database to use information they maintain to authenticate users. After being authenticated by the operating system, users can connect to a database without specifying a user name or password.

Database operations such as shutting down or starting up the database

用户的并发会话数、每个会话可用的 CPU 处理时间、和可用的逻辑 I/O 量（请参阅"缓冲区 I/O"）。例如，`clerk` 概要文件可能会限制用户对文书工作所需的系统资源。

注意：

最好使用数据库资源管理器来限制资源，并使用概要文件来管理密码。

概要文件为共享一组属性的多个用户提供了一个单一参照点。您可以将一个概要文件分配给一组用户，而将一个默认概要文件分配给所有其它用户。每个用户在任何一个时间点最多被分配一个概要文件。

另见：

- 《*Oracle 数据库安全指南*》了解如何使用概要文件来管理资源
- 《*Oracle 数据库 SQL 语言参考*》关于 `CREATE PROFILE` 的语法和语义

身份验证

身份验证是用户提供对数据库的验证凭据，而数据库验证该凭据并决定是否允许访问的过程。身份验证建立了进行进一步互操作的信任关系。身份验证通过将访问和操作与特定的身份链接，以达到问责跟踪的能力。

Oracle 数据库提供了以下不同的身份验证方法：

- 通过数据库进行身份验证

  Oracle 数据库可以使用密码、 Kerberos 票证、或 PKI 证书对用户进行身份验证。Oracle 还支持 RADIUS 兼容设备用于其他形式的身份验证，包括生物特征识别等。身份验证的类型必须在 Oracle 数据库中创建用户时指定。
- 通过操作系统进行身份验证

  某些操作系统允许 Oracle 数据库使用他们所维护的信息来对用户进行身份验证。在经过操作系统认证后，用户可以在不指定用户名称或密码的情况下连接到数据库。

如关闭或启动数据库之类的数据库操作不应由非管理性数据库用户执行。这

should not be performed by non-administrative database users. These operations require SYSDBA or SYSOPER privileges (see "Connection with Administrator Privileges").

See Also:

- *Oracle Database Security Guide* and *Oracle Database Advanced Security Administrator's Guide* for more information about authentication methods
- *Oracle Database Administrator's Guide* to learn about administrative authentication

Encryption

Encryption is the process of transforming data into an unreadable format using a secret key and an encryption algorithm. Encryption is often used to meet regulatory compliance requirements, such as those associated with the Payment Card Industry Data Security Standard (PCI-DSS) or breach notification laws. For example, credit card numbers, social security numbers, or patient health information must be encrypted.

Network Encryption

Network encryption refers to encrypting data as it travels across the network between a client and server. An intruder can use a network packet sniffer to capture information as it travels on the network, and then spool it to a file for malicious use. Encrypting data on the network prevents this sort of activity.

Transparent Data Encryption

Oracle Advanced Security **transparent data encryption** enables you to encrypt individual table columns or a tablespace. When a user inserts data into an encrypted column, the database automatically encrypts the data. When users select the column, the data is decrypted. This form of encryption is transparent, provides high performance, and is easy to implement.

Transparent data encryption includes industry-standard encryption algorithms such as the Advanced Encryption Standard (AES) and built-in key management.

See Also:

Oracle Database 2 Day + Security Guide and *Oracle Database Advanced Security Administrator's Guide*

Access Control

Oracle Database provides many techniques to control access to data. This section summarizes some of these techniques.

些操作需要 SYSDBA 或 SYSOPER 权限（请参见"使用管理员权限进行连接"）。

另见：

- 《*Oracle 数据库安全指南*》和 《*Oracle 数据库高级安全管理员指南*》有关身份验证方法的详细信息
- 《*Oracle 数据库管理员指南*》了解管理性身份验证

加密

加密是使用一个安全密钥和加密算法，将数据转换为不可读格式的过程。加密通常用于满足如那些与支付卡行业数据安全标准 (PCI DSS) 或违反通知法相关的法规遵从性要求。例如，信用卡号码、 社会安全号码、或病人的健康信息等必须被加密。

网络加密

网络加密是指当数据通过网络在客户端和服务器之间进行传输时对其进行加密。当信息在网络上传输时，入侵者可以使用网络数据包嗅探器捕获信息，然后将其假脱机到一个文件以便作案。对网络上的数据进行加密能防止这类活动。

透明数据加密

Oracle 高级安全的**透明数据加密**，可以加密个别表列或表空间。当用户将数据插入到加密列时，数据库自动加密列数据。当用户选择加密列时，其数据被解密。这种形式的加密是透明的， 提供高的性能，也很容易实现。

透明数据加密包括工业标准的加密算法，如高级加密标准 (AES) 和内置的密钥管理。

另见：

《*Oracle 数据库 2 日+ 安全指南*》和 《*Oracle 数据库高级安全管理员指南*》

访问控制

Oracle 数据库提供了很多技术来控制对数据的访问。本节对这些技术中的

Oracle Database Vault

Oracle Database Vault is a security option that restricts privileged user access to application data. You can use Oracle Database Vault to control when, where, and how the databases, data, and applications are accessed. Thus, you can address common security problems such as protecting against insider threats, complying with regulatory requirements, and enforcing separation of duty.

See Also:

Oracle Database 2 Day + Security Guide and *Oracle Database Vault Administrator's Guide*

Virtual Private Database (VPD)

Virtual Private Database (VPD) enables you to enforce security at the row and column level. A **security policy** establishes methods for protecting a database from accidental or malicious destruction of data or damage to the database infrastructure.

VPD is useful when security protections such as privileges and roles are not sufficiently fine-grained. For example, you can allow all users to access the `employees` table, but create security policies to restrict access to employees in the same department as the user.

Essentially, the database adds a dynamic `WHERE` clause to a SQL statement issued against the table, view, or synonym to which an Oracle VPD security policy was applied. The `WHERE` clause allows only users whose credentials pass the security policy to access the protected data.

See Also:

Oracle Database 2 Day + Security Guide and *Oracle Database Security Guide*

Oracle Label Security (OLS)

Oracle Label Security (OLS) is a security option that enables you to assign data classification and control access using **security labels**. You can assign a label to both data and users.

When assigned to data, the label can be attached as a hidden column to existing tables, providing transparency to existing SQL. For example, rows that contain highly sensitive data can be labeled `HIGHLY SENSITIVE`, while rows that are less sensitive can be labeled `SENSITIVE`, and so on. When a user attempts to access data, OLS compares the user label with the data label and determines whether access should be granted. Unlike VPD, OLS provides an out-of-the-box security policy and the metadata

一些进行了总结。

Oracle 数据库保险库

Oracle 数据库保险库是一种安全选项，用于限制特权用户访问应用程序数据。你可以使用 Oracle 数据库保险库来控制在何时、何处、以及如何访问数据库、 数据、和应用程序。这样，您可以解决常见的安全问题，如防护内部威胁、 遵从法规要求、和强制执行职责分离。

另见：

《*Oracle 数据库 2 日+ 安全指南*》和 《*Oracle 数据库保险库管理员指南*》

虚拟私有数据库 (VPD)

虚拟私有数据库(VPD) 使您能够执行行级和列级安全性。**安全策略**建立了保护数据库遭受意外或恶意的数据破坏或损坏数据库基础结构的方法。

当基于权限和角色的安全保护仍显得过于粗放时，VPD 是很有用的。例如，您可以允许所有用户访问 `employees` 表，但可以创建安全策略，以限制用户仅能访问同一所在部门中的雇员。

实际上，当在 SQL 语句所涉及的表、 视图、或同义词上应用了一个 Oracle VPD 安全策略时，数据库会将一个动态的 WHERE 子句添加到 SQL 语句。WHERE 子句只允许其凭据通过了安全策略的用户访问受保护的数据。

另见：

《*Oracle 数据库 2 日+ 安全指南*》和 《*Oracle 数据库安全指南*》

Oracle 标签安全 (OLS)

Oracle 标签安全 (OLS) 是一种安全选项，使您能够使用**安全标签**来指定数据分类和控制访问。您可以为数据和用户分配一个标签。

当标签被分配给数据时，可以作为一个隐藏列附加到现有表，对现有 SQL 提供了透明性。例如，包含高度敏感数据的行可以被标记为 `HIGHLY SENSITIVE`，而较不敏感的行可以标记为 `SENSITIVE` 等等。当用户尝试访问数据时，OLS 比较数据标签与用户标签，并确定是否可以访问。与 VPD 不同，OLS 提供了一个立即可用的安全策略，和用于定义、存储标签的元数据存储库。

repository for defining and storing labels.

See Also:

Oracle Database 2 Day + Security Guide and *Oracle Label Security Administrator's Guide*

Monitoring

Oracle Database provides multiple tools and techniques for monitoring user activity.

Database Auditing

Database auditing is the monitoring and recording of selected user database actions. You can use **standard auditing** to audit SQL statements, privileges, schemas, objects, and network and multitier activity. Alternatively, you can use **fine-grained auditing** to monitor specific database activities, such as actions on a database table or times that activities occur. For example, you can audit a table accessed after 9:00 p.m.

Reasons for using auditing include:

- Enabling future accountability for current actions
- Deterring users (or others, such as intruders) from inappropriate actions based on their accountability
- Investigating, monitoring, and recording suspicious activity
- Addressing auditing requirements for compliance

See Also:

- *Oracle Database 2 Day + Security Guide* and *Oracle Database Security Guide* to learn how to enable and disable auditing
- *Oracle Label Security Administrator's Guide* to learn about Oracle Label Security auditing, which supplements standard auditing

Oracle Audit Vault

Oracle Audit Vault enables you to consolidate, report, and configure alerts for audited data. You can consolidate audit data generated by Oracle Database and other relational databases. You can also use Oracle Audit Vault to monitor audit settings on target databases.

See Also:

Oracle Audit Vault Administrator's Guide

另见：

《*Oracle 数据库 2 日+ 安全指南*》和《*Oracle 标签安全管理员指南*》

监视

Oracle 数据库提供了多种工具和技术，用于监视用户活动。

数据库审计

数据库审计，是对选定用户的数据库操作进行监视和记录。可以使用**标准审计**来审核 SQL 语句、 特权、 模式、 对象、网络、和多层活动。或者，您可以使用**细粒度审计**来监视特定的数据库活动，如对某个数据库表的操作及其发生的时间。例如，您可以审计晚上 9: 00 后对表的访问

使用审计的原因包括：

- 使您在将来能够对当前的操作进行问责
- 基于用户（或其他人，如入侵者）的问责记录，阻止其进行不当操作
- 调查、 监视、并记录可疑活动
- 满足法规遵从性的审计要求

另见：

- 《*Oracle 数据库 2 日+ 安全指南*》和《*Oracle 数据库安全指南*》了解如何启用和禁用审计
- 《*Oracle 标签安全管理员指南*》了解 Oracle 标签安全审计，它是标准审计的补充

Oracle 审计保险库

Oracle 审计保险库使您能够对审计数据进行整合、报告、并配置警报。您可以将由 Oracle 数据库和其他关系数据库生成的审计数据进行整合。Oracle 审计保险库还可用于监视目标数据库上的审计设置。

另见：

《*Oracle 审计保险库管理员指南*》

Enterprise Manager Auditing Support

Oracle Enterprise Manager (Enterprise Manager) enables you to view and configure audit-related initialization parameters. Also, you can administer objects when auditing statements and schema objects. For example, Enterprise Manager enables you to display and search for the properties of current audited statements, privileges, and objects. You can enable and disable auditing as needed.

Overview of High Availability

Availability is the degree to which an application, service, or functionality is available on demand. For example, an OLTP database used by an online bookseller is available to the extent that it is accessible by customers making purchases. Reliability, recoverability, timely error detection, and continuous operations are the primary characteristics of **high availability**.

The importance of high availability in a database environment is tied to the cost of **downtime**, which is the time that a resource is unavailable. Downtime can be categorized as either planned or unplanned. The main challenge when designing a highly available environment is examining all possible causes of downtime and developing a plan to deal with them.

See Also:

Oracle Database High Availability Overview for an introduction to high availability

High Availability and Unplanned Downtime

Oracle Database provides high availability solutions to prevent, tolerate, and reduce downtime for all types of unplanned failures. Unplanned downtime can be categorized by its causes:

- Site Failures
- Computer Failures
- Storage Failures
- Data Corruption
- Human Errors

See Also:

Oracle Database High Availability Overview to learn about protecting against unplanned downtime

Site Failures

A **site failure** occurs when an event causes all or a significant portion of

企业管理器的审计支持

Oracle 企业管理器 （企业管理器） 使您能够查看和配置与审计有关的初始化参数。此外，当审计语句和模式对象时，您可以管理对象。例如，企业管理器使您能够显示和搜索当前被审计的语句、特权、和对象等属性。您可以根据需要启用和禁用审计。

高可用性的概述

可用性是指应用程序、 服务、或功能按用户要求的可用程度。例如，一个在线图书销售商使用的 OLTP 数据库是可用的，客户可以访问它并作出购买决定。可靠性、 可恢复性、 及时检错、和连续操作是**高可用性**的主要特征。

在数据库环境中的高可用性的重要性，取决于**停机时间**的成本。停机时间是某种资源不可用的时间。停机时间可被归类为计划内的或计划外的。设计高度可用环境时面临的主要挑战，是检查造成停机时间的所有可能的原因，并制定一个计划来应对他们。

另见：

《*Oracle 数据库高可用性概述*》关于高可用性的简介

高可用性和计划外停机时间

Oracle 数据库提供了高可用性解决方案，以防止、 容忍、并减少所有类型的计划外故障所造成的停机时间。计划外停机时间可按其原因分为以下几类：

- 站点故障
- 计算机故障
- 存储故障
- 数据损坏
- 人为错误

另见：

《*Oracle 数据库高可用性概述*》了解针对计划外停机的保护

站点故障

站点故障，即当某个事件发生，使得整个应用程序或其大部分停止处理或慢

an application to stop processing or slow to an unusable service level. A site failure may affect all processing at a data center, or a subset of applications supported by a data center. Examples include an extended site-wide power or network failure, a natural disaster making a data center inoperable, or a malicious attack on operations or the site.

The simplest form of protection against site failures is to create database backups using RMAN and store them offsite. You can restore the database to another host. However, this technique can be time-consuming, and the backup may not be current. Maintaining one or more standby databases in a Data Guard environment enables you to provide continuous database service if the production site fails.

See Also:

- *Oracle Database High Availability Overview* to learn about site failures
- *Oracle Database Backup and Recovery User's Guide* for information on RMAN and backup and recovery solutions
- *Oracle Data Guard Concepts and Administration* for an introduction to standby databases

Computer Failures

A **computer failure** outage occurs when the system running the database becomes unavailable because it has shut down or is no longer accessible. Examples of computers failures include hardware and operating system failures.

The following Oracle features protect against or help respond to computer failures:

- Enterprise Grids

  In an **Oracle Real Applications Cluster (Oracle RAC)** environment, Oracle Database runs on two or more systems in a cluster while concurrently accessing a single shared database. A single database system spans multiple hardware systems yet appears to the application as a single database. See "Overview of Grid Computing".
- Oracle Data Guard

  Data Guard enables you to maintain a copy of a production database, called a **standby database**, that can reside on a

到一个不可接受的服务级别。站点故障可能会影响一个数据中心的所有处理，或一个数据中心所支持的应用程序的一个子集。示例包括持续的整个站点电源或网络故障、 使数据中心无法操作的自然灾难、或对操作或站点的恶意攻击。

针对站点故障保护的最简单形式是使用 RMAN 创建数据库备份，并将它们存储到异地。您可以将数据库还原到另一台主机。然而，这种技术可能非常耗时，并且备份可能不是最新的。在 Data Guard 环境中维护一个或多个备用数据库，使您可以在生产站点出现故障时，仍能提供连续的数据库服务。

另见：

- 《*Oracle 数据库高可用性概述*》了解站点故障
- 《*Oracle 数据库备份和恢复用户指南*》关于 RMAN 备份和恢复解决方案的信息
- 《*Oracle 数据卫士概念和管理*》有关备用数据库的简介

计算机故障

运行数据库的系统由于被关闭或不再可访问而变得不可用时，会发生**计算机故障**停机。计算机故障的例子包括硬件和操作系统故障。

以下的 Oracle 特性能防止计算机故障，或有助于对其采取适当的响应：

- 企业网格

  在 **Oracle 真正应用程序群集 （Oracle RAC）** 环境中， Oracle 数据库运行在集群中的两个或多个系统上，并同时访问一个共享的数据库。单个数据库系统跨越多个硬件系统，但对于数据库应用程序来说，仍然只是一个单一数据库。请参阅"网格计算概述"。
- Oracle 数据卫士

  数据卫士使您可以维护一个生产数据库的副本，称为**备用数据库**，它可以驻留在不同的大洲，也可以位于相同的数据中心。如果主数据库

different continent or in the same data center. If the primary database is unavailable because of an outage, then Data Guard can switch any standby database to the primary role, minimizing downtime. See *Oracle Data Guard Concepts and Administration*.

- Oracle Restart

  Components in the Oracle Database software stack, including the database instance, listener, and Oracle ASM instance, can restart automatically after a component failure or whenever the database host computer restarts. Oracle Restart ensures that Oracle components are started in the proper order, in accordance with component dependencies. See *Oracle Database Administrator's Guide* to learn how to configure Oracle Restart.

- Fast Start Fault Recovery

  A common cause of unplanned downtime is a system fault or crash. The **fast start fault recovery technology** in Oracle Database automatically bounds database instance recovery time. See *Oracle Database Performance Tuning Guide* for information on fast start fault recovery.

See Also:

Oracle Database High Availability Best Practices to learn how to use High Availability for processes and applications that run in a single-instance database

Storage Failures

A **storage failure** outage occurs when the storage holding some or all of the database contents becomes unavailable because it has shut down or is no longer accessible. Examples of storage failures include the failure of a disk drive or storage array.

In addition to Oracle Data Guard, solutions for storage failures include the following:

- Oracle Automatic Storage Management (Oracle ASM)

  Oracle ASM is a vertically integrated file system and volume manager in the database kernel (see "Oracle Automatic Storage Management (Oracle ASM)"). Oracle ASM eliminates the complexity associated with managing data and disks, and simplifies mirroring and the process of adding and removing disks.

由于停机而不可用时，则数据卫士可以将任何备用数据库切换为主数据库角色，将停机时间降至最低。请参见 《*Oracle 数据卫士概念和管理*》。

- Oracle Restart

  Oracle 数据库软件堆栈中的组件，包括数据库实例、 监听器、和 Oracle ASM 实例等，当某个组件发生故障或当数据库主机计算机重新启动时，可以自动重新启动。Oracle Restart 确保 Oracle 组件按其依赖关系以正确的顺序启动。请参阅《*Oracle 数据库管理员指南*》了解如何配置 Oracle Restart。

- 快速启动故障恢复

  计划外停机时间的一个常见原因是系统故障或崩溃。Oracle 数据库的**快速启动故障恢复技术**自动限定数据库实例恢复时间。请参阅《*Oracle 数据库性能优化指南*》关于快速启动故障恢复的信息。

另见：

《*Oracle 数据库高可用性最佳实践*》，学习如何为在单实例数据库中运行的进程和应用程序使用高可用性

存储故障

当包含一些或所有数据库内容的存储，由于已关闭或不再可访问而变得不可用时，会发生**存储故障**停机。存储故障的例子包括磁盘驱动器或存储阵列故障。

除了 Oracle 数据卫士，存储故障的解决方案包括以下这些：

- Oracle 自动存储管理 （Oracle ASM）

  Oracle ASM 是在数据库内核中垂直整合的文件系统和卷管理器（请参阅"Oracle 自动存储管理 （Oracle ASM）"）。Oracle ASM 消除了与数据和磁盘管理相关的复杂性，并简化了镜像和添加、删除磁盘的过程。

- Backup and recovery

  The Recovery Manager (RMAN) utility can back up data, restore data from a previous backup, and recover changes to that data up to the time before the failure occurred (see "Backup and Recovery").

See Also:

- *Oracle Database 2 Day DBA* to learn how to administer Oracle ASM disks with Oracle Enterprise Manager (Enterprise Manager)
- *Oracle Automatic Storage Management Administrator's Guide* to learn more about Oracle ASM

Data Corruption

A **data corruption** occurs when a hardware, software or network component causes corrupt data to be read or written. For example, a volume manager error causes bad disk read or writes. Data corruptions are rare but can have a catastrophic effect on a database, and therefore a business.

In addition to Data Guard and Recovery Manager, Oracle Database supports the following forms of protection against data corruption:

- Oracle Hardware Assisted Resilient Data (HARD)

  The HARD Initiative is a joint initiative between Oracle and hardware vendors to prevent data corruptions from being written to disk. Under the HARD Initiative, Oracle works with selected system and storage vendors to build operating system and storage components that can detect corruption early and prevent corrupted data from being written to disk.
- Lost write protection

  A data block lost write occurs when an I/O subsystem acknowledges the completion of the block write when the write did not occur. You can configure the database so that is records buffer cache block reads in the redo log. Lost write detection is most effective when used with Data Guard.
- Data block corruption detection

  A **block corruption** is a data block that is not in a recognized

- 备份和恢复

  恢复管理器 （RMAN) 实用程序可以备份数据、将数据还原到之前的一个备份、将数据更改恢复到故障发生前的时间（见"备份和恢复"）。

另见：

- 《*Oracle 数据库 2 日 DBA*》 学习如何使用 Oracle 企业管理器 （企业管理器）来管理 Oracle ASM 磁盘
- 《*Oracle 自动存储管理管理员指南*》了解 Oracle ASM 的更多信息

数据损坏

当硬件、 软件、或网络组件导致读写已损坏的数据时，就会发生**数据损坏**。例如，卷管理器错误导致错误的磁盘读写。数据损坏很少见，但可以对数据库、甚至对业务造成灾难性影响。

除了数据卫士和恢复管理器，Oracle 数据库还支持针对下列形式数据损坏的保护：

- Oracle 硬件辅助数据恢复 （HARD）

  HARD 是 Oracle 和硬件供应商之间的一个联合倡议，以防止错误数据被写入到磁盘。在 HARD 倡议下， Oracle 与选定的几家系统和存储供应商，以建立可以检测早期损坏并防止将错误数据写入到磁盘的操作系统和存储组件。

- 丢失写入保护

  当一个 I/O 子系统确认了块写入完成，而该写入实际上未发生，就会发生数据块写丢失。您可以配置数据库，以使其能够在重做日志中记录对缓冲缓存块的读取。丢失写入检测与数据卫士一起使用时是最有效的。

- 数据块损坏检测

  块损坏即是一个数据块不是一种可识别的 Oracle 格式，或其内容在

Oracle format, or whose contents are not internally consistent. Several database components and utilities, including RMAN, can detect a corrupt block and record it in V$DATABASE_BLOCK_CORRUPTION. If the environment uses a real-time standby database, then RMAN can automatically repair corrupt blocks.

- Data Recovery Advisor

  Data Recovery Advisor is an Oracle tool that automatically diagnoses data failures, determines and presents appropriate repair options, and executes repairs at the user's request.

See Also:

- *Oracle Database High Availability Best Practices* to learn how to protect against data corruptions
- *Oracle Database Backup and Recovery User's Guide* for information on RMAN and backup and recovery solutions
- http://www.oracle.com/technology/deploy/availability/htdocs/HARD.html for more information about HARD

Human Errors

A **human error** outage occurs when unintentional or malicious actions are committed that cause data in the database to become logically corrupt or unusable. The service level impact of a human error outage can vary significantly depending on the amount and critical nature of the affected data.

Much research cites human error as the largest cause of downtime. Oracle Database provides powerful tools to help administrators quickly diagnose and recover from these errors. It also includes features that enable end users to recover from problems without administrator involvement.

Oracle Database recommends the following forms of protection against human error:

- Restriction of user access

  The best way to prevent errors is to restrict user access to data and services. Oracle Database provides a wide range of security tools to control user access to application data by authenticating users and then allowing administrators to grant users only those privileges

内部不一致。包括 RMAN 的几个数据库组件和工具，可以将检测损坏的块，并将其记录到 V$DATABASE_BLOCK_CORRUPTION 中。如果环境使用实时的备用数据库，则 RMAN 可以自动修复损坏的块。

- 数据恢复顾问

  数据恢复顾问是一个 Oracle 工具，它可以自动诊断数据故障、 确定和提供适当的修复选项，并在用户的请求下执行修复。

另见：

- 《*Oracle 数据库高可用性最佳实践*》了解如何防止数据损坏
- 《*Oracle 数据库备份和恢复用户指南*》关于 RMAN 备份和恢复解决方案的信息
- http://www.oracle.com/technology/deploy/availability/htdocs/HARD.html 有关 HARD 的详细信息

人为错误

当某些意外的或恶意的操作，导致数据库中的数据产生逻辑损坏或变得不可用时，会发生**人为错误**停机。基于受影响的数据的数量及其重要性，人为错误停机的服务级别影响可会显著不同。

很多研究表明，人为错误是造成停机时间的最大原因。Oracle 数据库提供了功能强大的工具，以帮助管理员快速诊断，并从这些错误中恢复。它还包括了使最终用户能够从问题中恢复而不需要管理员介入的功能。

针对人为错误，Oracle 数据库建议下列形式的保护：

- 限制用户访问：

  防止错误的最佳方法是限制用户对数据和服务的访问。Oracle 数据库提供了广泛的安全工具，通过进行用户身份验证，然后让管理员仅授予用户为执行其任务所需的权限（请参阅"数据库安全概述"），来控制用户对应用程序数据的访问。

required to perform their duties (see "Overview of Database Security").

- Oracle Flashback Technology

  Oracle Flashback Technology is a family of human error correction features in Oracle Database. Oracle Flashback provides a SQL interface to quickly analyze and repair human errors. For example, you can perform:
  - Fine-grained surgical analysis and repair for localized damage
  - Rapid correction of more widespread damage
  - Recovery at the row, transaction, table, tablespace, and database level
- Oracle LogMiner

  Oracle LogMiner is a relational tool that enables online files to be read, analyzed, and interpreted using SQL (see "Oracle LogMiner").

See Also:

- *Oracle Database High Availability Best Practices* to learn how to recover from human errors
- *Oracle Database Backup and Recovery User's Guide* and *Oracle Database Advanced Application Developer's Guide* to learn more about Oracle Flashback features
- *Oracle Database Utilities* to learn more about Oracle LogMiner

High Availability and Planned Downtime

Planned downtime can be just as disruptive to operations, especially in global enterprises that support users in multiple time zones. In this case, it is important to design a system to minimize planned interruptions such as routine operations, periodic maintenance, and new deployments. Planned downtime can be categorized by its causes:

- System and Database Changes
- Data Changes
- Application Changes

See Also:

Oracle Database High Availability Overview to learn about features and

- Oracle 闪回技术

  Oracle 闪回技术是 Oracle 数据库中的一系列纠正人为错误的功能。Oracle 闪回提供了快速分析和修复人为错误的 SQL 接口。例如，您可以执行：
  - 细粒度的精确分析，并修复局部损坏
  - 快速纠正更大范围的破坏
  - 在行、 事务、 表、 表空间、和数据库级别进行恢复
- Oracle LogMiner

  Oracle LogMiner 是一种关系型工具，它能使用 SQL 对日志文件进行读取、分析、和解释（请参阅"Oracle LogMiner"）。

另见：

- 《*Oracle 数据库高可用性的最佳实践*》了解如何从人为错误中恢复
- 《*Oracle 数据库备份和恢复用户指南*》和《*Oracle 数据库高级应用开发指南*》了解 Oracle 闪回功能的更多信息
- 《*Oracle 数据库实用程序*》了解 Oracle LogMiner

高可用性和计划内停机时间

计划内的停机时间对于公司运营可能同样是具有破坏性的，特别是在支持多个时区用户的全球性企业中。在这种情况下，设计一个系统以尽量减少例行操作、 定期维护、和新的部署等造成的计划内停机是很重要的。

计划内的停机时间，可以按其原因分类：

- 系统和数据库变更
- 数据更改
- 应用程序变更

另见：

《*Oracle 数据库高可用性概述*》了解用于计划内停机的功能和解决方案

solutions for planned downtime

System and Database Changes

Planned system changes occur when you perform routine and periodic maintenance operations and new deployments, including scheduled changes to the operating environment that occur outside of the organizational data structure in the database. Examples include adding or removing CPUs and cluster nodes (a **node** is a computer on which a database instance resides), upgrading system hardware or software, and migrating the system platform.

Oracle Database provides **dynamic resource provisioning** as a solution to planned system and database changes:

- Dynamic reconfiguration of the database

  Oracle Database dynamically accommodates various changes to hardware and database configurations, including adding and remove processors from an SMP server and adding and remove storage arrays using Oracle ASM. For example, Oracle Database monitors the operating system to detect changes in the number of CPUs. If the `CPU_COUNT` initialization parameter is set to the default, then the database workload can dynamically take advantage of newly added processors.
- Autotuning memory management

  Oracle Database uses a noncentralized policy to free and acquire memory in each subcomponent of the SGA and the PGA. Oracle Database autotunes memory by prompting the operating system to transfer granules of memory to components that require it. See "Memory Management".
- Automated distributions of data files, control files, and online redo log files

  Oracle ASM automates and simplifies the layout of data files, control files, and log files by automatically distributing them across all available disks. See *Oracle Automatic Storage Management Administrator's Guide* to learn more about Oracle ASM.

Data Changes

Planned data changes occur when there are changes to the logical structure or physical organization of Oracle Database objects. The primary objective of these changes is to improve performance or manageability.

系统和数据库变更

计划内系统变更发生在执行例行和定期的维护操作和新的部署时，如在数据库的组织数据结构之外，执行预定的操作环境变更。示例包括添加或删除 CPU 和集群节点 （**节点**是一个数据库实例所驻留的计算机）、 升级系统硬件或软件、以及迁移系统平台等。

Oracle 数据库提供了**动态资源调配**作为计划内系统和数据库变更的解决方案：

- 数据库动态重配置

  Oracle 数据库动态适应各种硬件和数据库配置变更，包括从 SMP 服务器添加和删除处理器，和使用 Oracle ASM 添加和删除存储阵列。例如， Oracle 数据库监视操作系统，以侦测 CPU 数量的变化。如果 CPU_COUNT 初始化参数设置为默认值，则数据库工作负荷可以动态地利用新添加的处理器。

- 自动优化内存管理

  Oracle 数据库使用非集中化的策略，在每个 SGA 和 PGA 子组件中释放或获取内存。Oracle 数据库通过提示操作系统将内存颗粒传输到需要它的组件来自动优化内存。请参见"内存管理"。

- 自动分布数据文件、 控制文件、和在线重做日志文件

  Oracle ASM 通过自动将数据文件、 控制文件、和日志文件分布到所有可用的磁盘，从而自动简化其布局。请参阅《*Oracle 自动存储管理管理员指南*》了解 Oracle ASM 的更多信息。

数据变更

当需要更改 Oracle 数据库对象的逻辑结构或物理组织时，将发生**计划内数据变更**。这些变更的主要目的，是提高性能或可管理性。示例包括重定义表、 添加表分区、和创建或重新生成索引。

Examples include table redefinition, adding table partitions, and creating or rebuilding indexes.

Oracle Database minimizes downtime for data changes through online reorganization and redefinition. This architecture enables you to perform the following tasks when the database is open:

- Perform **online table redefinition**, which enables you to make table structure modifications without significantly affecting the availability of the table
- Create, analyze, and reorganize indexes (see Chapter 3, "Indexes and Index-Organized Tables")
- Move table partitions (see "Overview of Partitions")

See Also:

Oracle Database Administrator's Guide to learn how to change data structures online

Application Changes

Planned application changes may include changes to data, schemas, and programs. The primary objective of these changes is to improve performance, manageability, and functionality. An example is an application upgrade.

Oracle Database supports the following solutions for minimizing application downtime required to make changes to an application's database objects:

- Rolling patch updates

  Oracle Database supports the application of patches to the nodes of an Oracle RAC system in a rolling fashion. See *Oracle Database High Availability Best Practices*.
- Rolling release upgrades

  Oracle Database supports the installation of database software upgrades, and the application of patchsets, in a rolling fashion—with near zero database downtime—by using Data Guard SQL Apply and logical standby databases. See *Oracle Database Upgrade Guide*.
- Edition-based redefinition

Oracle 数据库通过在线重组织和重定义来进行数据变更，将停机时间降至最低。这种体系结构使您可以在打开数据库时执行以下任务：

- 执行**表在线重定义**，使您能够进行表结构修改，而不会显著影响表的可用性
- 创建、 分析、并重新组织索引 （见第 3 章"索引和 Index-Organized 表"）
- 移动表分区 （见"分区概述"）

另见：

《*Oracle 数据库管理员指南*》了解如何在线更改数据结构

应用程序变更

计划内应用程序变更可能包括更改数据、 模式、和程序。这些更改的主要目的是提高性能、 可管理性、和功能。例如，应用程序升级。

Oracle 数据库支持以下解决方案，以尽量减少对应用程序的数据库对象进行变更所需的应用程序停机时间：

- 滚动补丁更新

  Oracle 数据库支持将补丁以滚动方式应用到 Oracle RAC 系统中的多个节点。请参阅 《*Oracle 数据库高可用性的最佳实践*》。
- 滚动发布更新

  Oracle 数据库支持以滚动的方式安装数据库软件升级和应用补丁集——几乎没有数据库停机时间 ——通过使用数据卫士 SQL 应用和逻辑备用数据库。请参阅 《*Oracle 数据库升级指南*》。
- 基于版本的重定义

Edition-based redefinition enables you to upgrade the database objects of an application while the application is in use, thus minimizing or eliminating down time. Oracle Database accomplishes this task by changing (redefining) database objects in a private environment known as an edition. See *Oracle Database Advanced Application Developer's Guide*.

- DDL with the default `WAIT` option

  DDL commands require exclusive locks on internal structures (see "DDL Locks"). In previous releases, DDL commands would fail if they could not obtain the locks. DDL specified with the `WAIT` option resolves this issue. See *Oracle Database High Availability Overview*.

- Creation of triggers in a disabled state

  You can create a trigger in the disabled state so that you can ensure that your code compiles successfully before you enable the trigger. See *Oracle Database PL/SQL Language Reference*.

Overview of Grid Computing

Grid computing is a computing architecture that effectively pools large numbers of servers and storage into a flexible, on-demand resource for all enterprise computing needs. A **Database Server Grid** is a collection of commodity servers connected together to run on one or more databases. A **Database Storage Grid** is a collection of low-cost modular storage arrays combined together and accessed by the computers in the Database Server Grid.

With the Database Server and Storage Grid, you can build a pool of system resources. You can dynamically allocate and deallocate these resources based on business priorities.

Figure 17-2 illustrates the Database Server Grid and Database Storage Grid in a Grid enterprise computing environment.

Figure 17-2 Grid Computing Environment

基于版本的重定义，可以在应用程序正在使用时，使您能够升级应用程序的数据库对象，从而最小化或消除停机时间。Oracle 数据库通过在一个称为版本的私有环境中更改（重定义）数据库对象来完成这项任务。请参阅《*Oracle 数据库高级应用程序开发人员指南*》

- 具有默认 `WAIT` 选项的 DDL

  DDL 命令需要在内部结构上获取独占锁 （见"DDL 锁"）。在以前的版本中，如果 DDL 命令无法获取锁，则它们将会失败。使用指定了 `WAIT` 选项的 DDL 可以解决此问题。请参阅 《*Oracle 数据库高可用性概述*》。

- 创建处于禁用状态的触发器

  您可以创建一个处于禁用状态的触发器，使您能够在启用触发器之前确保您的代码编译成功。请参阅 《*Oracle 数据库 PL/SQL 语言参考*》。

网格计算概述

网格计算是一种计算体系结构，能有效地将大量服务器和存储集中到一个灵活的、随需而变的资源池，以满足整个企业的计算需要。**数据库服务器网格**将一批低端服务器互连，以运行一个或多个数据库。**数据库存储网格**将一批低成本模块化存储阵列合并到一起，供数据库服务器网格中的计算机访问。

使用数据库服务器网格和存储网格，您可以构建一个系统资源池。您可以根据业务优先级，动态分配和释放这些资源。

图 17-2 说明了企业网格计算环境中的数据库服务器网格和数据库存储网格。

图 17-2 网格计算环境

Description of "Figure 17-2 Grid Computing Environment"

See Also:

- *Oracle Database High Availability Overview* for an overview of Grid Computing
- http://www.gridforum.org/ to learn about the standards organization Global Grid Forum (GGF)

Database Server Grid

Oracle Real Application Clusters (Oracle RAC) enables multiple instances that are linked by an interconnect to share access to an Oracle database. In an Oracle RAC environment, Oracle Database runs on two or more systems in a cluster while concurrently accessing a single shared database. Oracle RAC enables a Database Server Grid by providing a single database that spans multiple low-cost servers yet appears to the application as a single, unified database system.

Oracle Clusterware is software that enables servers to operate together as if they are one server. Each server looks like any standalone server.

Description of "Figure 17-2 Grid Computing Environment"

另见：

- 《*Oracle 数据库高可用性概述*》关于网格计算的概述
- http://www.gridforum.org/了解全球网格论坛(GGF)标准组织

数据库服务器网格

Oracle 真正应用集群 （Oracle RAC） 使相互连接的多个实例能对一个 Oracle 数据库进行共享访问。在 Oracle RAC 环境中， Oracle 数据库运行在集群中的两个或多个系统上，并发访问一个共享的数据库。Oracle RAC 通过提供跨越多个低成本服务器的单个数据库，而对应用程序显示为一个单一、 统一的数据库系统，形成一个数据库服务器网格。

Oracle 集群件是一个软件，它使多个服务器能够共同协作，如同是一台服

However, each server has additional processes that communicate with each other so that separate servers work together as if they were one server. Oracle Clusterware provides all of the features required to run the cluster, including node membership and messaging services.

See Also:

- *Oracle Database 2 Day + Real Application Clusters Guide* for an introduction to Oracle Clusterware and Oracle RAC
- *Oracle Real Application Clusters Administration and Deployment Guide* to learn how to manage an Oracle RAC database
- *Oracle Clusterware Administration and Deployment Guide* to learn how to administer and deploy Oracle Clusterware

Scalability

In a Database Server Grid, Oracle RAC enables you to add nodes to the cluster as the demand for capacity increases. The Cache Fusion technology implemented in Oracle RAC enables you to scale capacity without changing your applications. Thus, you can scale the system incrementally to save costs and eliminate the need to replace smaller single-node systems with larger ones.

You can incrementally add nodes to a cluster instead of replacing existing systems with larger nodes. Grid Plug and Play simplifies addition and removal of nodes from a cluster, making it easier to deploy clusters in a dynamically provisioned environment. Grid Plug and Play also enables databases and services to be managed in a location-independent manner. SCAN enables clients to connect to the database service without regard for its location within the grid.

See Also:

- *Oracle Real Application Clusters Administration and Deployment Guide* to learn more about Cache Fusion
- *Oracle Database Installation Guide* to learn how to install Grid Plug and Play

Fault Tolerance

Fault tolerance is the protection provided by a high availability architecture against the failure of a component in the architecture. A key advantage of the Oracle RAC architecture is the inherent fault tolerance provided by multiple nodes. Because the physical nodes run independently, the failure of one or more nodes does not affect other

务器一样。每个服务器看起来像一台独立服务器。然而，每个服务器都有额外的进程与其它服务器互相通信，使得各个独立的服务器可以一起工作，犹如它们是一台服务器一样。Oracle 集群件提供了所有运行该集群所需的功能，包括节点成员资格和消息服务。

另见：

- 《*Oracle 数据库 2 日+真正应用集群指南*》关于 Oracle 集群件与 Oracle RAC 的介绍
- 《*Oracle 真正应用集群管理和部署指南*》了解如何管理 Oracle RAC 数据库
- 《*Oracle 集群件管理和部署指南*》学习如何管理和部署 Oracle 集群件

可扩展性

在数据库服务器网格中，随着容量需求的增加，Oracle RAC 使您能够将节点添加到集群。在 Oracle RAC 中实施缓存融合技术，使您能够扩展容量，而不更改您的应用程序。因此，您可以以增量方式扩展系统，以节省成本，和消除将较小的单节点系统替换为较大的系统的需要。

你可以以增量方式将节点添加到集群中，而不是用更大的节点替换现有系统。网格即插即用简化了从群集中添加和删除节点，使得在动态资源调配环境中部署集群变得更加容易。网格即插即用还使数据库和服务能以一种独立于位置的方式来管理。SCAN 使客户端能够连接到数据库服务，而不用考虑其在网格中的位置。

另见：

- 《*Oracle 真正应用集群管理和部署指南*》了解高速缓存融合的更多信息
- 《*Oracle 数据库安装指南*》了解如何安装网格即插即用

容错能力

容错能力是高可用性体系结构针对其中的组件故障所提供的保护。Oracle RAC 体系结构的关键优势是由多个节点提供的固有的容错能力。因为物理节点独立运行，一个或多个节点的故障不会影响集群中的其他节点。

nodes in the cluster.
Failover can happen to any node on the Grid. In the extreme case, an Oracle RAC system provides database service even when all but one node is down. This architecture allows a group of nodes to be transparently put online or taken offline, for maintenance, while the rest of the cluster continues to provide database service.
Oracle RAC provides built-in integration with Oracle Clients and connection pools. With this capability, an application is immediately notified of any failure through the pool that terminates the connection. The application avoids waiting for a TCP timeout and can immediately take the appropriate recovery action. Oracle RAC integrates the listener with Oracle Clients and the connection pools to create optimal application throughput. Oracle RAC can balance cluster workload based on the load at the time of the transaction.

See Also:

- "Database Resident Connection Pooling"
- *Oracle Real Application Clusters Administration and Deployment Guide* to learn more about automatic workload management
- *Oracle Database High Availability Best Practices* for an overview of fault tolerance in Oracle RAC

Services

Oracle RAC supports **services** that can group database workloads and route work to the optimal instances assigned to offer the services. A service represents the workload of applications with common attributes, performance thresholds, and priorities.
You define and apply business policies to these services to perform tasks such as to allocate nodes for times of peak processing or to automatically handle a server failure. Using services ensures the application of system resources where and when they are needed to achieve business goals. Services are integrated with the Database Resource Manager, which enables you to restrict the resources that are used by a service within an instance. In addition, Oracle Scheduler jobs can run using a service, as opposed to using a specific instance.

See Also:

- *Oracle Database 2 Day + Real Application Clusters Guide* to learn about Oracle services

故障切换可能在网格中的任何节点上发生。在极端情况下，即使除了一个节点之外的所有结点都处于关闭状态， Oracle RAC 系统仍能提供数据库服务。这种体系结构允许将一组节点透明地联机或脱机以进行维护，而集群中的其余部分仍然继续提供数据库服务。

Oracle RAC 提供与 Oracle 客户端和连接池的内置集成。有了这一功能，应用程序可以立即从终止该连接的池得到任何故障通知。应用程序可以避免等待 TCP 超时，并可以立即采取适当的恢复操作。Oracle RAC 将监听器与 Oracle 客户端和连接池集成，以创建优化的应用程序吞吐量。Oracle RAC 可以基于在事务中的负载，平衡集群工作量。

另见：

- "数据库驻留连接池"
- 《*Oracle 真正应用集群管理和部署指南*》了解自动工作负载管理
- 《*Oracle 数据库高可用性最佳实践*》关于在 Oracle RAC 中的容错能力的概述

服务

Oracle RAC 支持**服务**，它可以将数据库工作负荷分组，并将工作路由到被指定来提供服务的最佳实例。一个服务代表具有共同属性、 性能阈值、和优先级的应用程序工作负载。

你定义服务来执行任务，并将业务策略应用于这些服务，如为高峰处理时间分配节点，或自动处理服务器故障。使用服务可以确保系统资源在需要的地方和需要的时候被使用，以实现业务目标。

服务与数据库资源管理器集成，使您能够限制实例内由某个服务使用的资源。此外，Oracle 调度器作业可以使用一个服务来运行，而不是使用特定的实例来运行。

另见：

- 《*Oracle 数据库 2 日+ 真正应用集群指南*》了解 Oracle 服务

- *Oracle Database Administrator's Guide* to learn about the Database Resource Manager and Oracle Scheduler

Database Storage Grid

A DBA or storage administrator can use the Oracle ASM interface to specify the disks within the Database Storage Grid that ASM should manage across all server and storage platforms. ASM partitions the disk space and evenly distributes the data across the disks provided to ASM. Additionally, ASM automatically redistributes data as disks from storage arrays are added or removed from the Database Storage Grid.

See Also:

- "Oracle Automatic Storage Management (Oracle ASM)"
- *Oracle Database High Availability Overview* for an overview of the Database Storage Grid
- *Oracle Automatic Storage Management Administrator's Guide* for more information about clustered Oracle ASM

Overview of Data Warehousing and Business Intelligence

A **data warehouse** is a relational database designed for query and analysis rather than for transaction processing. For example, a data warehouse could track historical stock prices or income tax records. A warehouse usually contains data derived from historical transaction data, but it can include data from other sources.

A data warehouse environment includes several tools in addition to a relational database. A typical environment includes an ETL solution, an OLAP engine, Oracle Warehouse Builder, client analysis tools, and other applications that gather data and deliver it to users.

Data Warehousing and OLTP

A common way of introducing data warehousing is to refer to the characteristics of a data warehouse as set forth by William Inmon[Foot 1]:

- Subject-Oriented

  Data warehouses enable you to define a database by subject matter, such as sales.

- 《*Oracle 数据库管理员指南*》了解数据库资源管理器和 Oracle 调度器

数据库存储网格

DBA 或存储管理员可以使用 Oracle ASM 接口，来指定数据库存储网格内 ASM 应跨所有服务器和存储平台来管理的磁盘。ASM 对磁盘空间进行分区，并将数据均匀分布在为 ASM 提供的多个磁盘上。此外， 当从数据库存储网格的存储阵列中添加或删除磁盘时，ASM 会自动重新分布数据。

另见：

- "Oracle 自动存储管理 （Oracle ASM）"
- 《*Oracle 数据库高可用性概述*》关于数据库存储网格的概述
- 《*Oracle 自动存储管理管理员指南*》关于集群化的 Oracle ASM 的详细信息

数据仓库和商业智能概述

数据仓库是一个关系数据库，它是为查询和分析设计的，而不是为事务处理设计的。例如，数据仓库可能会跟踪历史股票价格或所得税记录。一个数据仓库通常包含来自历史事务的数据，但它也可以包含来自其他源的数据。

数据仓库环境包括一个关系数据库和几个工具。一个典型的环境包括一个 ETL 解决方案、 OLAP 引擎、 Oracle 仓库生成器、 客户端分析工具、和其他收集数据并将其提交给用户的应用程序。

数据仓库和 OLTP

介绍数据仓库的常见方式是引用威廉 • 伊曼提出的数据仓库特征 [Foot 1]

- 面向主题

  数据仓库使您能够按照主题（如销售）定义数据库。

- Integrated

  Data warehouses must put data from disparate sources into a consistent format. They must resolve such problems as naming conflicts and inconsistencies among units of measure. When they achieve this goal, they are said to be integrated.
- Nonvolatile

  The purpose of a warehouse is to enable you to analyze what has occurred. Thus, after data has entered into the warehouse, data should not change.
- Time-Variant

  The focus of a data warehouse is on change over time.

Data warehouses and OLTP database have different requirements. For example, to discover trends in business, data warehouses must maintain large amounts of data. In contrast, good performance requires historical data to be moved regularly from OLTP systems to an archive. Table 17-1 lists differences between data warehouses and OLTP.

Table 17-1 Data Warehouses and OLTP Systems

- 集成的

  数据仓库必须将来自不同来源的数据，以一致的格式存放。他们必须解决命名冲突和度量单位之间不一致等问题。当它们达成此目标时，就可以说它们是集成的。

- 非易失的

  数据仓库的目的是使您能够分析发生了什么。因此，数据已进入仓库后，数据不应更改。

- 随时间变化的

  数据仓库的重点是随时间的变化。

数据仓库和 OLTP 数据库有不同的要求。例如，为发现业务中的趋势，数据仓库必须保持大量数据。而另一方面是，需要定期将历史数据从 OLTP 系统移动到存档，以维持良好的性能。表 17-1 列出了数据仓库和 OLTP 之间的差异。

表 17-1 数据仓库和 OLTP 系统

Characteristics	**Data Warehouse**	**OLTP**	**特性**	**数据仓库**	**OLTP**
Workload	Designed to accommodate ad hoc queries. You may not know the workload of your data warehouse in advance, so it should be optimized to perform well for a wide variety of possible queries.	Supports only predefined operations. Your applications might be specifically tuned or designed to support only these operations.	工作负载	设计为适应即席查询。你可能不能预先知道您的数据仓库的工作量，因此它应被优化，以能很好地执行各种可能的查询。	只支持预定义的操作。您的应用程序可能需要被专门调整或设计，以支持这些操作。
Data modifications	Updated on a regular basis by the ETL process using bulk data modification techniques. End users of a data warehouse do not directly update the database.	Subject to individual DML statements routinely issued by end users. The OLTP database is always up to date and reflects the current state of each business transaction.	数据修改	通过使用大容量数据修改技术的 ETL 过程来定期更新。数据仓库的最终用户不会直接更新数据库。	单个 DML 语句通常是由最终用户发出的。OLTP 数据库始终处于最新状态，并反映每个业务事务的当前状态。
Schema design	Uses denormalized or	Uses fully normalized	模式设计	使用反规范化，或部分反规	使用完全规范化模式，以优

	partially denormalized schemas (such as a star schema) to optimize query performance.	schemas to optimize DML performance and to guarantee data consistency.		范化模式（如星型模式），以优化查询性能。	化 DML 性能，并保证数据一致性。
Typical operations	A typical query scans thousands or millions of rows. For example, a user may request the total sales for all customers last month.	A typical operation accesses only a handful of records. For example, a user may retrieve the current order for a single customer.	典型操作	典型的查询扫描数以千计或数以百万计的行。例如，用户可能要求上个月所有客户的总销售额。	典型的操作仅访问个别记录。例如，用户可能检索某个单一客户的当前订单。
Historical data	Stores many months or years of data to support historical analysis.	Stores data from only a few weeks or months. Historical data retained as needed to meet the requirements of the current transaction.	历史数据	存储数月或数年的数据，以支持历史分析。	只存储几周或几个月的数据。根据需要保留历史数据以满足当前的事务要求。

See Also:

- *Oracle Database Data Warehousing Guide* for a more detailed description of a database warehouse
- *Oracle Database VLDB and Partitioning Guide* for a more detailed description of an OLTP system

另见：

- 《*Oracle 数据库数据仓库指南*》有关数据库仓库的更详细说明
- 《*Oracle 数据库 VLDB 和分区指南*》关于 OLTP 系统更详细的说明

Data Warehouse Architecture

Data warehouses and their architectures vary depending on the business requirements. This section describes common data warehouse architectures.

Data Warehouse Architecture (Basic)

Figure 17-3 shows a simple architecture for a data warehouse. End users directly access data that was transported from several source systems to the data warehouse.

Figure 17-3 Architecture of a Data Warehouse

数据仓库体系结构

数据仓库及其体系结构基于不同的业务需求而变化。本节介绍了常见的数据仓库体系结构。

数据仓库体系结构 (基本)

图 17-3 显示了数据仓库的一个简单体系结构。最终用户直接访问从多个源系统传输到数据仓库中的数据。

图 17-3 数据仓库的体系结构

Description of "Figure 17-3 Architecture of a Data Warehouse"

Figure 17-3 shows both the metadata and raw data of a traditional OLTP system and summary data. A **summary** is an aggregate view that improves query performance by precalculating expensive joins and aggregation operations and storing the results in a table. For example, a summary table can contain the sums of sales by region and by product. Summaries are also called materialized views.

See Also:

Oracle Database Data Warehousing Guide to learn about basic materialized views

Data Warehouse Architecture (with a Staging Area)

In the architecture shown in Figure 17-3, operational data must be cleaned and processed before being put into the warehouse. Figure 17-4 shows a data warehouse with a **staging area**, which is a place where data is preprocessed before entering the warehouse. A staging area simplifies the tasks of building summaries and managing the warehouse.

Figure 17-4 Architecture of a Data Warehouse with a Staging Area

Description of "Figure 17-3 Architecture of a Data Warehouse"

图 17-3 显示了元数据、传统 OLTP 系统的原始数据、和汇总数据。**汇总**是一个聚合视图，通过预先计算昂贵的联接和聚合操作，并将结果存储在表中，以提高查询性能。例如，汇总表可能包含按区域和产品的销售总和。汇总也称为物化视图。

另见：

《*Oracle 数据库数据仓库指南*》了解基本的物化视图

数据仓库体系结构（使用临时区域）

图 17-3 所示的体系结构中，业务数据必须经过清理和处理才能被放入数据仓库。图 17-4 显示与一个使用临时区域的数据仓库，临时区域是数据进入仓库前进行预处理的地方。临时区域简化了构建汇总和管理仓库的任务。

图 17-4 使用临时区域的数据仓库体系结构

Description of "Figure 17-4 Architecture of a Data Warehouse with a Staging Area"

See Also:

Oracle Database Data Warehousing Guide to learn about different transportation mechanisms

Data Warehouse Architecture (with a Staging Area and Data Marts)

You may want to customize your warehouse architecture for different groups within your organization. You can achieve this goal by transporting data in the warehouse to **data marts**, which are independent databases designed for a specific business or project. Typically, data marts include many summary tables.

Figure 17-5 separates purchasing, sales, and inventory information into independent data marts. A financial analyst can query the data marts for historical information about purchases and sales.

Figure 17-5 Architecture of a Data Warehouse with a Staging Area and Data Marts

Description of "Figure 17-4 Architecture of a Data Warehouse with a Staging Area"

另见：

《*Oracle 数据库数据仓库指南*》了解不同的传输机制

数据仓库体系结构 （使用临时区域和数据集市）

您可能希望为您组织内的不同工作组定制仓库体系结构。你可以通过将仓库中的数据传输到**数据集市**来实现这个目标，数据集市是一些为特定业务或项目设计的独立数据库。通常，数据集市包括许多汇总表。

图 17-5 将购买、 销售、和库存信息分隔为独立的数据集市。金融分析师可以在这些数据集市中查询有关采购和销售的历史信息。

图 17-5 使用临时区域及数据集市的数据仓库体系结构

Description of "Figure 17-5 Architecture of a Data Warehouse with a Staging Area and Data Marts"

See Also:

Oracle Database Data Warehousing Guide to learn about transformation mechanisms

Overview of Extraction, Transformation, and Loading (ETL)

The process of extracting data from source systems and bringing it into the warehouse is commonly called **ETL**: extraction, transformation, and loading. ETL refers to a broad process rather than three well-defined steps.

In a typical scenario, data from one or more operational systems is extracted and then physically transported to the target system or an intermediate system for processing. Depending on the method of transportation, some transformations can occur during this process. For example, a SQL statement that directly accesses a remote target through a gateway can concatenate two columns as part of the `SELECT` statement.

Oracle Database is not itself an ETL tool. However, Oracle Database provides a rich set of capabilities usable by ETL tools such as Oracle Warehouse Builder and customized ETL solutions. ETL capabilities provided by Oracle Database include:

- Transportable tablespaces

  You can transport tablespaces between different computer

Description of "Figure 17-5 Architecture of a Data Warehouse with a Staging Area and Data Marts"

另见：

《*Oracle 数据库数据仓库指南*》了解转换机制

提取、 转换，和加载 (ETL) 概述

从源系统中提取数据并将其添加到仓库的过程通常称为 ETL： 提取、 转换、和加载。ETL 指的是一个广泛的过程，而不是三个明确的步骤。

在一个典型场景中，从一个或多个业务系统中提取数据，然后物理地传送到目标系统或中间系统进行处理。依赖于传输的方法，在此过程中可能发生某些转换。例如，一个通过网关直接访问远程目标的 SQL 语句，可以连接两个列，来作为 SELECT 语句的一部分。

Oracle 数据库本身不是一个 ETL 工具。然而，Oracle 数据库提供了一套丰富的功能可供 ETL 工具（如 Oracle 仓库生成器和定制的 ETL 解决方案）来使用。Oracle 数据库所提供的 ETL 功能包括：

- 可传输表空间

  你可以在不同体系结构的计算机和操作系统之间传输表空间。可传输

architectures and operating systems. Transportable tablespaces are the fastest way for moving large volumes of data between two Oracle databases. See *Oracle Database Administrator's Guide* to learn about transportable tablespaces.

- Table functions

  A **table function** can produce a set of rows as output and can accept a set of rows as input. Table functions provide support for pipelined and parallel execution of transformations implemented in PL/SQL, C, or Java without requiring the use of intermediate staging tables. See *Oracle Database Data Warehousing Guide* to learn about table functions.
- External tables

  External tables enable external data to be joined directly and in parallel without requiring it to be first loaded in the database (see "External Tables"). Thus, external tables enable the pipelining of the loading phase with the transformation phase.
- Table compression

  To reduce disk use and memory use, you can store tables and partitioned tables in a compressed format (see "Table Compression"). The use of table compression often leads to a better scaleup for read-only operations and faster query execution.
- Change Data Capture

  This feature efficiently identifies and captures data that has been added to, updated in, or removed from, relational tables and makes this change data available for use by applications or individuals.

See Also:

- *Oracle Database Data Warehousing Guide* to learn about Change Data Capture
- *Oracle Warehouse Builder Data Modeling, ETL, and Data Quality Guide* for an overview of ETL

Business Intelligence

Business intelligence is the analysis of an organization's information as an aid to making business decisions. Business intelligence and analytical

表空间是在两个 Oracle 数据库之间移动大量数据的最快方法。请参阅 《*Oracle 数据库管理员指南*》了解可传输表空间。

- 表函数

  表函数可以生成一个行集作为输出，也可以接受一个行集作为输入。表函数提供执行流水线和并行转换的支持，它使用 PL/SQL、C、Java 实现，而无需使用中间临时表。请参阅 《*Oracle 数据库数据仓库指南*》了解表函数。

- 外部表

  外部表使你可以直接以并行方式对外部数据进行连接操作，而不需要先将它加载到数据库中（请参见"外部表"）。因此，外部表使得加载阶段和转换阶段可以平滑对接。

- 表压缩

  为减少磁盘和内存的使用，您可以使用一种压缩的格式来存储表和分区表（请参阅"表压缩"）。使用表压缩功能通常可以改进只读操作和执行更快的查询。

- 更改数据捕获

  此功能有效地标识并捕获在关系表中被插入、更新、删除的数据，并使这些更改的数据可供应用程序或个人使用。

另见：

- 《*Oracle 数据库数据仓库指南*》了解更改数据捕获
- 《*Oracle 仓库生成器数据建模、ETL 、和数据质量指南*》关于 ETL 的概述

商业智能

商业智能即分析组织信息以作为做出业务决策的辅助手段。商业智能和分析应用程序的操作，主要是对层次结构进行上卷和下钻，以及比较聚合值等操

applications are dominated by actions such as drilling up and down hierarchies and comparing aggregate values. Oracle Database provides several technologies to support business intelligence operations.

Analytic SQL

Oracle Database has introduced many SQL operations for performing analytic operations. These operations include ranking, moving averages, cumulative sums, ratio-to-reports, and period-over-period comparisons. For example, Oracle Database supports the following forms of analytic SQL:

- SQL for aggregation

  Aggregate functions such as `COUNT` return a single result row based on groups of rows rather than on single rows. Aggregation is fundamental to data warehousing. To improve aggregation performance in a warehouse, the database provides extensions to the `GROUP BY` clause to make querying and reporting easier and faster. See *Oracle Database Data Warehousing Guide* to learn about aggregation.
- SQL for analysis

  Analytic functions compute an aggregate value based on a group of rows. They differ from aggregate functions in that they return multiple rows for each group. Oracle has advanced SQL analytical processing capabilities using a family of analytic SQL functions. For example, these analytic functions enable you to calculate rankings and percentiles and moving windows. See *Oracle Database Data Warehousing Guide* to learn about SQL for analysis and reporting.
- SQL for modeling

  With the `MODEL` clause, you can create a multidimensional array from query results and apply rules to this array to calculate new values. For example, you can partition data in a sales view by country and perform a model computation, as defined by multiple rules, on each country. One rule could calculate the sales of a product in 2008 as the sum of sales in 2006 and 2007. See *Oracle Database Data Warehousing Guide* to learn about SQL modeling.

See Also:

Oracle Database SQL Language Reference to learn about SQL functions

作。Oracle 数据库提供了支持商业智能操作的几种技术。

分析型 SQL

Oracle 数据库为执行分析型操作引入了许多 SQL 操作。这些操作包括分级、浮动平均值、 累积求和、 比对报告、同期比较。例如 Oracle 数据库支持以下形式的分析型 SQL：

- 聚合 SQL

  聚合函数为一组行（而非单个行）返回单个结果行，比如 `COUNT`。聚合是数据仓库的重要功能。为了提高在仓库中的聚合性能，数据库提供对 GROUP BY 子句进行扩展，使得查询和报表更容易和更快。请参阅《*Oracle 数据库数据仓库指南*》了解聚合。

- 分析 SQL

  分析函数基于一组行计算聚合值。他们与聚合函数的不同在于他们会为每个组返回多个行。Oracle 使用一系列分析函数，具有高级的 SQL 分析处理能力。例如，这些分析函数使您能够计算分级、分位点、和移动窗口。请参阅《*Oracle 数据库数据仓库指南*》了解用于分析和报表的 SQL。

- 建模函数

  使用 `MODEL` 子句，你可以从查询结果中创建一个多维数组，并将规则应用于此数组以计算新值。例如，你可以按国家对销售视图中的数据进行分区，为每个国家定义一个规则，并按这些规则执行一个模型计算。一个规则可能会计算在 2008 年某种产品的销售额，以及在 2006 年和 2007 年的销售总额。请参阅《*Oracle 数据库数据仓库指南*》了解 SQL 建模。

另见：

《*Oracle 数据库 SQL 语言参考*》了解 SQL 函数

OLAP

Oracle **online analytical processing (OLAP)** provides native multidimensional storage and rapid response times when analyzing data across multiple dimensions. OLAP enables analysts to quickly obtain answers to complex, iterative queries during interactive sessions.
Oracle OLAP has the following primary characteristics:

- Oracle OLAP is integrated in the database so that you can use standard SQL administrative, querying, and reporting tools.
- The OLAP engine runs within the kernel of Oracle Database.
- Dimensional objects are stored in Oracle Database in their native multidimensional format.
- Cubes and other dimensional objects are first class data objects represented in the Oracle data dictionary.
- Data security is administered in the standard way, by granting and revoking privileges to Oracle Database users and roles.

Oracle OLAP offers the power of simplicity: one database, standard administration and security, and standard interfaces and development tools.

See Also:

- "Overview of Dimensions"
- *Oracle OLAP User's Guide* for an overview of Oracle OLAP

Data Mining

Data mining involves automatically searching large stores of data for patterns and trends that go beyond simple analysis. Data mining uses sophisticated mathematical algorithms to segment data and evaluate the probability of future events. Typical applications of data mining include call centers, ATMs, E-business relational management (ERM), and business planning.
With Oracle Data Mining, the data, data preparation, model building, and model scoring results all remain in the database. Oracle Data Mining supports a PL/SQL API, a Java API, SQL functions for model scoring, and a GUI called Oracle Data Miner. Thus, Oracle Database provides an infrastructure for application developers to integrate data mining

OLAP

Oracle **联机分析处理 (OLAP)** 提供原生的多维存储，和跨多个维度分析数据时的快速响应。OLAP 使分析师在交互式会话期间进行复杂、 迭代的查询时，能快速获得答案。

Oracle OLAP 具有以下主要特点：

- Oracle OLAP 与数据库集成，使您可以使用标准的 SQL 管理、查询、和报表工具。
- OLAP 引擎在 Oracle 数据库内核中运行。
- 维度对象以其原生的多维格式存储在 Oracle 数据库中。
- 数据立方体和其他维度对象，在 Oracle 数据字典中是以一级数据对象表示的。
- 通过对 Oracle 数据库用户和角色授予和撤消权限，数据安全以标准的方式进行管理。

Oracle OLAP 简单而强大： 一个数据库、 标准的管理和安全、和标准接口和开发工具。

另见：

- "维度概述"
- 《*Oracle OLAP 用户指南*》关于 Oracle OLAP 的概述

数据挖掘

数据挖掘从大量数据存储中自动搜索超越简单分析的某种模式和趋势。数据挖掘使用复杂的数学算法来划分数据并估计未来事件的概率。数据挖掘的典型应用包括呼叫中心、自动柜员机、电子商务关系管理 (ERM) 、和业务规划。

使用 Oracle 数据挖掘时，其数据、 数据准备、模型建立、和模型评分结果全都被保留在数据库中。Oracle 数据挖掘支持一个 PL/SQL API、一个 Java API、多个模型评分的 SQL 函数、和一个称为 Oracle 数据挖掘机的 GUI。Oracle 数据库为应用程序开发人员提供了一个将数据挖掘与数据库应用程

seamlessly with database applications.

See Also:

Oracle Data Mining Concepts

Overview of Oracle Information Integration

As an organization evolves, it becomes increasingly important for it to be able to share information among multiple databases and applications. The basic approaches to sharing information are as follows:

- Consolidation

  You can consolidate the information into a single database, which eliminates the need for further integration. Oracle RAC, Grid computing, and Oracle VPD can enable you to consolidate information into a single database.
- Federation

  You can leave information distributed, and provide tools to **federate** this information, making it appear to be in a single virtual database.
- Sharing

  You can share information, which lets you maintain the information in multiple data stores and applications.

This section focuses on Oracle solutions for federating and sharing information.

See Also:

Oracle Database 2 Day + Data Replication and Integration Guide for an introduction to data replication and integration

Federated Access

The foundation of federated access is a **distributed environment**, which is a network of disparate systems that seamlessly communicate with each other. Each system in the environment is called a **node**. The system to which a user is directly connected is called the **local system**. Additional systems accessed by this user are **remote systems**.

A distributed environment enables applications to access and exchange data from the local and remote systems. All the data can be simultaneously accessed and modified.

Distributed SQL

Distributed SQL synchronously accesses and updates data distributed

序无缝集成的基础架构。

另见：

《*Oracle 数据挖掘概念*》

Oracle 信息集成概述

随着组织的发展，在多个数据库和应用程序之间共享信息变得越来越重要。共享信息的基本方法如下所示：

- 整合

  您可以将信息整合到一个单一的数据库，这消除了进一步集成的需要。Oracle RAC、 网格计算、和 Oracle VPD 可以使您能够将信息整合到一个单一的数据库。
- 联邦

  可以保持信息分散化，并提供将信息结成联邦的工具，这使其看起来像一个单一的虚拟数据库。
- 共享

  您可以共享信息，允许您将信息维护在多个数据存储和应用程序中。

本节着重于联邦和共享信息的 Oracle 解决方案。

另见：

《*Oracle 数据库 2 日+ 数据复制和集成指南*》有关数据复制和集成的简介

联邦访问

联邦访问的基础是**分布式环境**，这是一个完全不同的系统无缝地相互通信的网络。环境中的每个系统称为一个**节点**。用户直接连接到的系统称为**本地系统**。由该用户访问的额外系统是**远程系统**。

分布式环境使应用程序能够在本地和远程系统之间访问和交换数据。所有数据都可以同时访问和修改。

分布式 SQL

分布式 SQL 同步访问和更新分布在多个数据库中的数据。分布式的 Oracle

among multiple databases. An Oracle distributed database system can be transparent to users, making it appear as a single Oracle database.

Distributed SQL includes distributed queries and distributed transactions. The Oracle distributed database architecture provides query and transaction transparency. For example, standard DML statements work just as they do in a non-distributed database environment. Additionally, applications control transactions using the standard SQL statements `COMMIT`, `SAVEPOINT`, and `ROLLBACK`.

See Also:

- "Overview of Distributed Transactions"
- *Oracle Database 2 Day + Data Replication and Integration Guide* to learn about distributed SQL
- *Oracle Database Administrator's Guide* to learn how to manage distributed transactions

Database Links

A **database link** is a connection between two physical databases that enables a client to access them as one logical database. Oracle Database uses database links to enable users on one database to access objects in a remote database. A local user can access a link to a remote database without being a user on the remote database.

Figure 17-6 shows an example of user `hr` accessing the `employees` table on the remote database with the global name `hq.example.com`. The `employees` synonym hides the identity and location of the remote schema object.

Figure 17-6 Database Link

Description of "Figure 17-6 Database Link"

See Also:

Oracle Database Administrator's Guide to learn about database links

Information Sharing

At the heart of any integration is the sharing of data among applications in the enterprise. **Oracle Streams** is the asynchronous information sharing

数据库系统对用户可能是透明的，使其看起来像是一个单一的 Oracle 数据库。

分布式 SQL 包括分布式查询和分布式事务。Oracle 分布式数据库体系结构提供了查询和事务的透明性。例如，标准 DML 语句就像他们在非分布式数据库环境中一样正常工作。此外，应用程序使用标准的 SQL 语句 `COMMIT`、 `SAVEPOINT`、和 `ROLLBACK` 来控制事务。

另见：

- "分布式事务概述"
- 《*Oracle 数据库 2 日+ 数据复制和集成指南*》了解分布式 SQL
- 《*Oracle 数据库管理员指南*》了解如何管理分布式事务

数据库链接

数据库链接是两个物理数据库之间的一个连接，它使客户机能将其作为一个逻辑数据库来访问。Oracle 数据库使用数据库链接来使一个数据库上的用户能够访问另一个远程数据库中的对象。本地用户可以访问到远程数据库的一个链接，而不用成为远程数据库上的一个用户。

如图 17-6 中的例子所示，用户 hr 访问在远程数据库上具有全局名称 hq.example.com 的 `employees` 表。`employees` 同义词隐藏了远程模式对象的标识和位置。

图 17-6 数据库链接

Description of "Figure 17-6 Database Link"

另见：

《*Oracle 数据库管理员指南*》了解数据库链接

信息共享

整合的核心是在企业中的应用程序之间共享数据。**Oracle 流**是 Oracle 数据

infrastructure in Oracle Database. This infrastructure enables the propagation and management of data, transactions, and events in a data stream either within a database, or from one database to another.

Oracle Streams includes replication and messaging. **Replication** is the process of sharing database objects and data at multiple databases. **Messaging** is the sharing of information between applications and users.

See Also:

- *Oracle Streams Concepts and Administration*
- *Oracle Streams Replication Administrator's Guide*

Oracle Streams Replication

In **Oracle Streams replication**, a change to a database object at one database can be shared with other databases in the replication environment. For example, Oracle Streams propagates an update to an `employees` table to an identical `employees` table in a different database. In this way, the database objects and data are kept synchronized at all databases in the replication environment.

Typical uses for Oracle Streams replication include:

- Creating a reporting site to offload processing from a primary OLTP site.
- Providing load balancing and improved scalability and availability for a call center or similar application.
- Providing site autonomy between locations to satisfy certain common business requirements.
- Transforming and consolidating data from multiple locations.
- Replicating data between different platforms and Oracle Database releases, and across a wide area network (WAN).

Oracle Streams Information Flow

The architecture of Oracle Streams is very flexible. Figure 17-7 depicts the basic information flow in a replication environment.

Figure 17-7 Oracle Streams Information Flow

库中的异步信息共享基础设施。此基础结构使你能够在一个数据库中，或从一个数据库到另一个数据库，以数据流的形式传播和管理数据、 事务、和事件。

Oracle 流包括复制和消息传递。**复制**是在多个数据库中共享数据库对象和数据的过程。消息传递是在应用程序和用户之间共享信息。

另见：

- 《*Oracle 流概念和管理*》
- 《*Oracle 流复制管理员指南*》

Oracle 流复制

在 Oracle 流复制中，在一个数据库上对某个数据库对象的更改可以在复制环境中与其他数据库共享。例如， Oracle 流将 `employees` 表的更新传播到不同的数据库中相同的 `employees` 表。这样一来，数据库对象及其数据能在复制环境中的所有数据库上保持同步。

Oracle 数据流复制的典型用途包括：

- 创建一个报表站点，以减轻 OLTP 主站点上的处理负担。
- 为呼叫中心或类似应用程序提供负载平衡、改进的可扩展性、和可用性。
- 提供站点位置之间的自治，以满足某些常见的业务要求。
- 转换和整合来自多个位置的数据。
- 在不同的平台、不同的 Oracle 数据库版本之间、或跨广域网 (WAN) 复制数据。

Oracle 流的信息流动

Oracle 流的体系结构是非常灵活的。图 17-7 描述了复制环境中的基本的信息流动。

图 17-7 Oracle 流的信息流动

Description of "Figure 17-7 Oracle Streams Information Flow"

As shown in Figure 17-7, Oracle Streams contains the following basic elements:

- Capture

  Oracle Streams can implicitly capture DML and DDL changes. Rules determine which changes are captured. Changes are formatted into **logical change records (LCRs)**, which are messages with a specific format describing a database change.
- Staging

  LCRs are placed in a staging area, which is a queue that stores and manages captured messages. Message staging provides a holding area with security, as well as auditing and tracking of message data. Propagations can send messages from one queue to another. The queues can reside in the same or different databases.
- Consumption

  LCRs remain in a staging area until subscribers consume them implicitly or explicitly. An apply process implicitly applies changes encapsulated in LCRs.

Note:

Oracle Streams is fully inter-operational with materialized views, which you can use to maintain updatable or read-only copies of data (see "Overview of Materialized Views").

See Also:

Oracle Database 2 Day + Data Replication and Integration Guide to learn how to replicate data using Oracle Streams

Oracle Streams Replication Environments

Oracle Streams enables you to configure many different types of custom replication environments. However, the following types of replication environments are the most common:

- Two-Database

  Only two databases share the replicated database objects. The changes made to replicated database objects at one database are captured and sent directly to the other database, where they are

Description of "Figure 17-7 Oracle Streams Information Flow"

如图 17-7 所示， Oracle 流包含下列的基本元素：

- 捕获

  Oracle 流可以隐式地捕获 DML 和 DDL 更改。由规则确定将捕获哪些更改。更改被格式化为**逻辑更改记录 (LCR)**，这是一些描述数据库更改的具有特定格式的消息。

- 中转

  LCR 被放入一个临时区域，这是一个存储和管理已捕获消息的队列。消息中转提供了具有安全性的存储区，并提供对消息数据的审计和跟踪。可以发送消息并将其从一个队列传播到另一个队列。队列可以驻留在相同或不同的数据库中。

- 消费

  LCR 驻留在临时区域，直到订阅者隐式或显式地将其消费。应用程序将隐式应用封装在 LCR 中的更改。

注意：

Oracle 流是完全使用物化视图在内部进行操作的，您可以使用物化视图来维护数据的可更新的或只读的副本（请参阅"物化视图概述"）。

另见：

《*Oracle 数据库 2 日+ 数据复制和集成指南*》了解如何使用 Oracle 流来复制数据

Oracle 流复制环境

Oracle 流使你可以配置许多不同类型的自定义复制环境。而以下类型的复制环境是最常见的：

- 双数据库

  只有两个数据库共享被复制的数据库对象。在一个数据库上对被复制的数据库对象所做的更改，被捕获并直接发送到另一个数据库，然后在那儿被应用。

applied.
In a **one-way replication environment**, only one database allows changes to the replicated database objects, with the other database containing read-only replicas of these objects. In a **bi-directional replication environment**, both databases can allow changes to the replicated objects. In this case, both databases capture changes to these database objects and send the changes to the other database, where they are applied.

- Hub-and-Spoke

  A central database, or **hub**, communicates with secondary databases, or **spokes**. The spokes do not communicate directly with each other. In a hub-and-spoke replication environment, the spokes might or might not allow changes to the replicated database objects.

- N-Way

  Each database communicates directly with every other database in the environment. The changes made to replicated database objects at one database are captured and sent directly to each of the other databases in the environment, where they are applied.

See Also:
Oracle Database 2 Day + Data Replication and Integration Guide to learn more about common replication environments

Oracle Streams Advanced Queuing (AQ)

Oracle Streams Advanced Queuing (AQ) is a robust and feature-rich message queuing system integrated with Oracle Database. When an organization has different systems that must communicate with each other, a messaging environment can provide a standard, reliable way to transport critical information between these systems.

A sample use case is a business that enters orders in an Oracle database at headquarters. When an order is entered, the business uses AQ to send the order ID and order date to a database in a warehouse. These messages alert employees at the warehouse about the orders so that they can fill and ship them.

Message Queuing and Dequeuing

Advanced Queuing stores user messages in abstract storage units called **queues**. **Enqueuing** is the process by which **producers** place messages into queues. **Dequeuing** is the process by which **consumers** retrieve

在**单向复制环境**中，只能有一个数据库允许对被复制的数据库对象进行更改，而另一个数据库仅能包含这些对象的只读副本。**在双向复制环境**中，两个数据库都可以更改被复制的对象。在这种情况下，这两个数据库都会捕获这些数据库对象的更改，并将更改发送到对方数据库然后在那儿被应用。

- 中心辐射

  一个中央数据库（或**中心**），与多个次级的数据库（或**分支**）进行通信。各个分支之间不直接与对方进行通信。在中心辐射复制的环境中，分支可能或可能不允许对复制的数据库对象进行更改。

- N－路

  每个数据库直接与环境中的每个其他数据库进行通信。在一个数据库上对复制的数据库对象所做的更改，被捕获并直接发送到每个在该环境中的其他数据库，并在那儿被应用。

另见：
《*Oracle 数据库 2 日+ 数据复制和集成指南*》了解常见复制环境的更多信息

Oracle 流高级队列 (AQ)

Oracle 流高级队列 (AQ) 是一个与 Oracle 数据库集成的、健壮且功能丰富的消息队列系统。当一个组织具有多个必须相互通信的不同系统时，消息环境可以提供一种在这些系统之间传输关键信息的标准的、 可靠的方法。

一个示例用例是在总部 Oracle 数据库中录入订单的业务。当输入一个订单时，业务使用 AQ 将订单 ID 和订单日期发送到仓库中的数据库。这些消息提醒仓库中的雇员注意有关的订单，使他们可以填写订单并发运货物。

消息入队和出队

高级队列将用户消息存储为称为**队列**的抽象存储单元。**入队**是**生产者**将消息放入队列的过程。**出队**是**消费者**从队列检索消息的过程。

messages from queues.
Support for explicit dequeue allows developers to use Oracle Streams to reliably exchange messages. They can also notify applications of changes by leveraging the change capture and propagation features of Oracle Streams.
Figure 17-8 shows a sample application that explicitly enqueues and dequeues messages through Advanced Queuing, enabling it to share information with partners using different messaging systems. After being enqueued, messages can be transformed and propagated before being dequeued to the partner's application.

Figure 17-8 Oracle Streams Message Queuing

Description of "Figure 17-8 Oracle Streams Message Queuing"

Advanced Queuing Features

Oracle Streams Advanced Queuing supports all the standard features of message queuing systems. These features include:

- Asynchronous application integration

  Oracle Streams Advanced Queuing offers several ways to enqueue messages. A capture process or synchronous capture can capture the messages implicitly, or applications and users can capture messages explicitly.
- Extensible integration architecture

  Many applications are integrated with a distributed hub and spoke model with Oracle Database as the hub. The distributed applications on an Oracle database communicate with queues in the same hub. Multiple applications share the same queue, eliminating the need to add queues to support additional applications.
- Heterogeneous application integration

  Advanced Queuing provides applications with the full power of the Oracle type system. It includes support for scalar data types, Oracle

对显式出队的支持使得开发人员能使用 Oracle 流来可靠地交换消息。它们也可以通过利用 Oracle 流的变化捕获和传播特征，将更改通知到应用程序。

图 17-8 显示了一个示例应用程序，它通过高级队列明确地将消息入队和出队，使其能够与使用不同消息系统的合作伙伴共享信息。入队后，消息能够被转换，且在出队之前传播到伙伴应用程序。

图 17-8 Oracle 数据流消息队列

Description of "Figure 17-8 Oracle Streams Message Queuing"

高级队列特性

Oracle 流高级队列支持消息队列系统的所有标准功能。这些功能包括：

- 异步应用程序集成

  Oracle 流高级队列提供了几种方法可以将消息入队。捕获进程或同步捕获可以隐式捕获消息，或者应用程序和用户也可以显式捕获消息。
- 可扩展的集成体系结构

  许多应用程序都与一个以 Oracle 数据库为中心的分布式中心辐射模型相集成。Oracle 数据库上的分布式应用程序与同一中心的队列进行通信。多个应用程序共享同一队列，消除了添加队列以支持其他应用程序的需要。
- 异构应用程序集成

  高级队列为应用程序提供全面的 Oracle 类型系统的能力。它包括对标量数据类型、可继承的 Oracle 数据库对象类型、针对 XML 数据的

Database object types with inheritance, `XMLType` with additional operators for XML data, and `ANYDATA`.

- Legacy application integration

  The Oracle Messaging Gateway integrates Oracle Database applications with other message queuing systems, such as Websphere MQ and Tibco.

- Standards-Based API support

  Oracle Streams Advanced Queuing supports industry-standard APIs: SQL, JMS, and SOAP. Changes made using SQL are captured automatically as messages.

See Also:

- *Oracle Database 2 Day + Data Replication and Integration Guide* to learn how to send messages using Advanced Queuing
- *Oracle Streams Advanced Queuing User's Guide*

Oracle Streams Use Case

For a sample use case, assume that a company uses Oracle Streams to maintain multiple copies of a corporate Web site. The business requirements include:

- A reporting database must contain the most current data for analysts in a New York office to perform ad hoc querying.
- Updatable materialized views must support the field sales staff.
- Data must be shared with applications hosted on a Sybase database.

Figure 17-9 illustrates this Streams configuration.

Figure 17-9 Streams Configuration

XMLType 及其他运算符、以及 ANYDATA 的支持。

- 遗留应用程序集成

  Oracle 消息网关将 Oracle 数据库应用程序与其他消息队列系统集成，诸如 Websphere MQ 和 Tibco 等。

- 基于标准的 API 支持

  Oracle 流高级队列支持行业标准的 API： SQL、JMS、和 SOAP。使用 SQL 所做更改会以消息的形式自动被捕获。

另见：

- 《*Oracle 数据库 2 日+ 数据复制和集成指南*》了解如何使用高级队列发送消息
- 《*Oracle 流高级队列用户指南*》

Oracle 流使用案例

举一个示例使用案例，假定一个公司使用 Oracle 流来维护一个企业 Web 站点的多个副本。业务需求包括：

- 在纽约办公室的报表数据库必须包含分析师执行特设查询所需的最新数据。
- 可更新的物化视图必须能支持现场销售人员。
- 数据必须能与 Sybase 数据库上的应用程序共享，。

图 17-9 说明了这个流配置。

图 17-9 流配置

Description of "Figure 17-9 Streams Configuration"

Oracle Streams is used to replicate data in an n-way configuration consisting of sites in New York, London, and Tokyo. At each site, Streams implicit capture collects any changes that occur for subscribed tables in each local region, and stages them locally in the queue. Changes captured in each region are then forwarded to each of the other region's databases. Changes made at each database can be reflected at every other database, providing complete data for the subscribed objects throughout the world. At each regional database, an Oracle Streams apply process applies the changes automatically. As changes are applied, Oracle Streams checks for and resolves any conflicts. Streams can also be used to exchange data for particular tables with non-Oracle databases. Using the Oracle Database Gateway for Sybase, a Streams apply process applies the changes to a Sybase database using the same mechanisms as it does for Oracle databases.

The reporting database is hosted in New York. This database is a fully functional Oracle database that has a read-only copy of the relevant application tables. The reporting site is not configured to capture changes on these application tables. Oracle Streams imposes no restrictions on the configuration or use of this reporting database.

The London site also serves as the master site for several updatable materialized view sites. Each salesperson receives an updatable copy of the required portion of data. These sites typically only connect once a day

Description of "Figure 17-9 Streams Configuration"

Oracle 流被用于在一个由在纽约、 伦敦、和东京等站点组成的 n-路配置中复制数据。在每个站点，*流隐式捕获*收集每个本地区域的订阅表中发生的任何更改，并将其在本地临时放入队列。在每个区域中捕获的更改然后被转发到其它区域中的每个数据库。在每个数据库中所做的更改都能反映到其它每一个数据库，为世界各地的订阅对象提供完整的数据。

在每个区域数据库，一个 Oracle 流应用进程将自动应用所做的更改。当更改被应用时， Oracle 流检查并解决任何冲突。流也可以用于与非 Oracle 数据库交换特定表的数据。使用针对 Sybase 的 Oracle 数据库网关、 一个流应用进程使用与 Oracle 数据库相同的机制将更改应用于 Sybase 数据库。

报表数据库位于纽约。此数据库是全功能的 Oracle 数据库，其中包含有关应用程序表的只读副本。报表站点未配置为捕获这些应用程序表的更改。Oracle 流对该报表数据库的配置或使用未作任何限制。

伦敦站点还担任了几个可更新物化视图站点的主站点。每个销售人员接收所需部分数据的一个可更新副本。这些站点通常只能每天只连接一次，以上传他们的订单或下载自上次刷新后的任何更改。

to upload their orders and download any changes made after their last refresh.

See Also:

Oracle Database 2 Day + Data Replication and Integration Guide for examples of configuring Oracle Streams

---

Footnote Legend

Footnote 1: *Building the Data Warehouse*, John Wiley and Sons, 1996.

另见：

《*Oracle 数据库 2 日+ 数据复制和集成指南*》关于配置 Oracle 数据流的示例

Footnote Legend

Footnote 1: *Building the Data Warehouse*, John Wiley and Sons, 1996.

Part VI

Oracle Database Administration and Development

This part describes summarizes topics that are essential for database administrators and developers.

This part contains the following chapters:

- Chapter 17, "Topics for Database Administrators and Developers"
- Chapter 18, "Concepts for Database Administrators"
- Chapter 19, "Concepts for Database Developers"

Previous Next View PDF

第六部分

Oracle 数据库管理与开发

本部分概述了对数据库管理员和开发人员都很重要的主题。

本部分包含以下各章：

- 第 17 章，"数据库管理员和开发人员的主题"
- 第 18 章，"数据库管理员相关概念"
- 第 19 章，"数据库开发人员相关概念"

18 Concepts for Database Administrators

This chapter contains the following sections:

- Duties of Database Administrators
- Tools for Database Administrators
  - Oracle Enterprise Manager
  - SQL*Plus
  - Tools for Database Installation and Configuration
  - Tools for Oracle Net Configuration and Administration
  - Tools for Data Movement and Analysis
- Topics for Database Administrators
  - Backup and Recovery
  - Memory Management
  - Resource Management and Task Scheduling
  - Performance Diagnostics and Tuning

Duties of Database Administrators

The principal responsibility of a database administrator (DBA) is to make enterprise data available to its users. DBAs must work closely with the developers to ensure that their applications make efficient use of the database, and with system administrators to ensure that physical resources are adequate and used efficiently.

Oracle DBAs are responsible for understanding the Oracle Database architecture and how the database works. DBAs can expect to perform the following tasks:

- Installing, upgrading, and patching Oracle Database software
- Designing databases, including identifying requirements, creating the logical design (conceptual model), and physical database design
- Creating Oracle databases
- Developing and testing a backup and recovery strategy, backing up Oracle databases regularly, and recovering them in case of failures
- Configuring the network environment to enable clients to connect to databases
- Starting up and shutting down the database

第 18 章 数据库管理员相关概念

本章包含以下各节：

- 数据库管理员的职责
- 数据库管理员的工具
  - Oracle 企业管理器
  - SQL*Plus
  - 数据库安装和配置工具
  - Oracle 网络配置和管理工具
  - 数据移动和分析工具
- 数据库管理员的主题
  - 备份和恢复
  - 内存管理
  - 资源管理与任务调度
  - 性能诊断和调优

数据库管理员的职责

数据库管理员 (DBA) 的主要责任是向用户提供企业数据。DBA 必须与开发人员紧密合作，确保他们的应用程序有效利用了的数据库，并与系统管理员紧密合作，以确保物理资源足够且被高效地使用。

Oracle 数据库管理员负责了解 Oracle 数据库体系结构和数据库的工作方式。数据库管理员可能需要执行下列任务：

- 安装、 升级、和修补 Oracle 数据库软件
- 设计数据库，包括确定需求、创建逻辑设计 （概念模型） 、和物理数据库设计
- 创建 Oracle 数据库
- 开发和测试备份和恢复策略，定期备份 Oracle 数据库，并在发生故障的情况下恢复它们
- 配置网络环境，使客户端能连接到数据库
- 启动和关闭数据库

- Managing storage for the database
- Managing users and security
- Managing database objects such as tables, indexes, and views
- Monitoring and tuning database performance
- Investigating, gathering diagnostic data for, and reporting to Oracle Support Services any critical database errors
- Evaluating and testing new database features

The preceding tasks, and many others, are described in *Oracle Database 2 Day DBA* and *Oracle Database Administrator's Guide*.

The types of users and their roles and responsibilities depend on the database environment. A small database may have one DBA. A very large database may divide the DBA duties among several specialists, for example, security officers, backup operators, and application administrators.

Tools for Database Administrators

Oracle provides several tools for use in administering a database. This section describes some commonly used tools:

- Oracle Enterprise Manager
- SQL*Plus
- Tools for Database Installation and Configuration
- Tools for Oracle Net Configuration and Administration
- Tools for Data Movement and Analysis

Oracle Enterprise Manager

Oracle Enterprise Manager (Enterprise Manager) is a system management tool that provides centralized management of a database environment. Combining a graphical console, Oracle Management Servers, Oracle Intelligent Agents, common services, and administrative tools, Enterprise Manager provides a comprehensive systems management platform for Oracle products.

The Web-based **Enterprise Manager Database Control (Database Control)** is the primary tool for managing an Oracle database. It is installed with Oracle Database. You can use Database Control to perform administrative tasks such as:

- Diagnosing, modifying, and tuning the database
- Grouping related targets together to facilitate administration tasks,

- 管理数据库存储
- 管理用户和安全
- 管理数据库对象，如表、 索引、和视图
- 监视和优化数据库性能
- 针对任何严重的数据库错误，进行调查、 收集诊断数据、并报告给 Oracle 支持服务
- 评价和测试新的数据库功能

《*Oracle 数据库 2 日 DBA*》和《*Oracle 数据库管理员指南*》描述了上述任务和许多其他任务。

用户的类型及其角色和责任取决于数据库环境。一个小型数据库可能会只有一个 DBA。而一个非常大的数据库可能会在几个专家之间划分 DBA 职责，例如安全人员、 备份操作员、和应用程序管理员等。

数据库管理员的工具

Oracle 提供了几种工具，用于管理数据库。本部分介绍了一些常用的工具：

- Oracle 企业管理器
- SQL*Plus
- 数据库安装和配置工具
- Oracle 网络配置和管理工具
- 数据移动和分析工具

Oracle 企业管理器

Oracle 企业管理器 （企业管理器） 是一个提供数据库环境集中化管理的系统管理工具。企业管理器将图形控制台、 Oracle 管理服务器、 Oracle 智能代理、 公共服务、和管理工具结合在一起，为 Oracle 产品提供了一个综合的系统管理平台。

基于 Web 的**企业管理器数据库控制** （数据库控制） 是用于管理 Oracle 数据库的主要工具。它与 Oracle 数据库一起安装。可以使用数据库控制执行管理任务，如：

- 诊断、 修改、和优化数据库
- 将相关的目标组合到一起以便于执行管理任务、与其他管理员共享任

sharing tasks with other administrators, and scheduling tasks at varying time intervals

- Configuring and managing Oracle Net Services for an Oracle home (see "Overview of Oracle Networking Architecture")
- Launching integrated Oracle and third-party tools

The following figure shows the Database Home page of Database Control. The subpage links across the top of the page enable you to access performance, availability, and other database administration pages. The subsections of the Database Home page provide information about the environment and status of the database.

Description of the illustration homepage_small.gif

The following figure shows the basic architecture of Enterprise Manager. The management repository is stored inside the database. Both the agent and the management service run on the database host. You can run the Database Control Console from any Web browser that can connect securely to the management service.

务、和在不同的时间间隔调度任务

- 为某个 Oracle home 配置和管理 Oracle 网络服务（请参阅"Oracle 网络架构概述"）
- 启动集成的 Oracle 工具和第三方工具

下图显示了数据库控制的数据库主页。在页面顶部的子页面链接使您能够访问性能、 可用性、和其他数据库管理页面。数据库主页的各个子节提供了有关环境和数据库状态的信息。

Description of the illustration homepage_small.gif

下图显示了企业管理器的基本体系结构。管理存储库存储在数据库内。代理和管理服务都在数据库主机上运行。您可以从任何能够安全连接到管理服务的 Web 浏览器，运行数据库控制台。

Description of the illustration cncpt305.gif

See Also:

Oracle Database 2 Day DBA to learn how to administer the database with Enterprise Manager

SQL*Plus

SQL*Plus is an interactive and batch query tool included in every Oracle Database installation. It has a command-line user interface that acts as the client when connecting to the database.

SQL*Plus has its own commands and environment. It enables you to enter and execute SQL, PL/SQL, SQL*Plus and operating system commands to perform tasks such as:

- Formatting, performing calculations on, storing, and printing from query results
- Examining table and object definitions
- Developing and running batch scripts
- Administering a database

You can use SQL*Plus to generate reports interactively, to generate reports as batch processes, and to output the results to text file, to screen, or to HTML file for browsing on the Internet. You can generate reports dynamically using the HTML output facility.

See Also:

Oracle Database 2 Day DBA and *SQL*Plus User's Guide and Reference* to

Description of the illustration cncpt305.gif

另见：

《*Oracle 数据库 2 日 DBA*》了解如何使用企业管理器来管理数据库

SQL*Plus

SQL*Plus 是一个包含在每个 Oracle 数据库安装中的交互式和批处理查询工具。它具有一个当连接到数据库时作为客户端的命令行用户接口。

SQL*Plus 有其自己的命令和环境。它使您能够输入和执行 SQL、PL/SQL、SQL*Plus、和操作系统命令来执行任务，如：

- 对查询结果进行格式化设置、 执行计算、 存储、并打印
- 检查表和对象定义
- 开发和运行批处理脚本
- 管理数据库

您可以使用 SQL*Plus 以交互方式或以批处理方式生成报表，并将结果输出到文本文件、屏幕、或在互联网上浏览的 HTML 文件。您还可以使用动态 HTML 输出工具生成报表。

另见：

《*Oracle 数据库 2 日 DBA*》和《*SQL*Plus 用户指南和参考*》了解 SQL*Plus

learn more about SQL*Plus

Tools for Database Installation and Configuration

Oracle provides several tools to simplify the task of installing and configuring Oracle Database software. The tools include:

- Oracle Universal Installer (OUI)

  OUI is a GUI utility that enables you to view, install, and deinstall Oracle Database software. Online Help is available to guide you through the installation. See *Oracle Database Installation Guide* to learn how to install Oracle Database software.
- Database Upgrade Assistant (DBUA)

  DBUA interactively guides you through a database upgrade and configures the database for the new release. DBUA automates the upgrade by performing all tasks normally performed manually. DBUA makes recommendations for configuration options such as tablespaces and the online redo log. See *Oracle Database 2 Day DBA* to learn how to upgrade a database with DBUA.
- Database Configuration Assistant (DBCA)

  DBCA provides a graphical interface and guided workflow for creating and configuring a database. This tool enables you to create a database from Oracle-supplied templates or create your own database and templates. See *Oracle Database Administrator's Guide* to learn how to create a database with DBCA.

Tools for Oracle Net Configuration and Administration

Oracle Net Services provides enterprise wide connectivity solutions in distributed, heterogeneous computing environments. Oracle Net, a component of Oracle Net Services, enables a network session from a client application to an database. You can use the following tools to configure and administer Oracle Net Services:

- Oracle Net Manager

  This tool enables you to configure Oracle Net Services for an Oracle home on a local client or server host. You can use Oracle Net Manager to configure naming, naming methods, profiles, and listeners. You can start Oracle Net Manager using the Oracle Enterprise Manager Console or as an independent application.

的更多信息

数据库安装和配置工具

Oracle 提供了一些工具来简化安装和配置 Oracle 数据库软件的任务。这些工具包括：

- Oracle 通用安装程序 (OUI)

  OUI 是一个图形用户界面实用程序，使您能够查看、安装、和卸载 Oracle 数据库软件。联机帮助可引导您完成安装。请参阅 《*Oracle 数据库安装指南*》了解如何安装 Oracle 数据库软件。

- 数据库升级助手 (DBUA)

  DBUA 以交互方式引导您将数据库升级和配置为新的发行版。DBUA 自动执行与升级相关的所有任务，而在以前这通常是手动执行的。DBUA 会为表空间和联机重做日志等配置选项提出建议。请参阅 《*Oracle 数据库 2 日 DBA*》学习如何使用 DBUA 来升级数据库。

- 数据库配置助手 (DBCA)

  DBCA 提供了一个图形界面和引导工作流以创建和配置数据库。此工具使您能够从 Oracle 提供的模板创建数据库，或创建您自己的数据库和模板。请参阅 《*Oracle 数据库管理员指南*》了解如何使用 DBCA 创建一个数据库。

Oracle 网络配置和管理工具

Oracle 网络服务提供企业范围的分布式异构计算环境中的连接解决方案。Oracle 网络是 Oracle 网络服务的一个组件，使一个网络会话能从客户端应用程序连接到数据库。可以使用以下工具来配置和管理 Oracle 网络服务：

- Oracle 网络管理器

  此工具使您可以为本地客户端或服务器主机上的一个 Oracle Home 配置 Oracle 网络服务。您可以使用 Oracle 网络管理器来配置命名、命名方法、 概要文件、和监听器。您可以使用 Oracle 企业管理器控制台来启动 Oracle 网络管理器，或作为一个独立的应用程序来启动。

- Oracle Net Configuration Assistant

  This tools runs automatically during software installation. The Assistant enables you to configure basic network components during installation, including listener names and protocol addresses, naming methods, net service names in a `tnsnames.ora` file, and directory server usage.
- Listener Control Utility

  The Listener Control utility enables you to configure listeners to receive client connections (see "The Oracle Net Listener"). You can access the utility through Enterprise Manager or as a standalone command-line application.
- Oracle Connection Manager Control Utility

  This command-line utility enables you to administer an **Oracle Connection Manager**, which is a router through which a client connection request may be sent either to its next hop or directly to the database. You can use utility commands to perform basic management functions on one or more Oracle Connection Managers. Additionally, you can view and change parameter settings.

See Also:

- "Overview of Oracle Networking Architecture"
- *Oracle Database Net Services Administrator's Guide* and *Oracle Database Net Services Reference* to learn more about Oracle Net Services tools

Tools for Data Movement and Analysis

Oracle Database includes several utilities to assist in database movement and analysis. For example, you can use database utilities to:

- Load data into Oracle Database tables from operating system files, as explained in "SQL*Loader"
- Move data and metadata from one database to another database, as explained in "Oracle Data Pump Export and Import"
- Query redo log files through a SQL interface, as explained in "Oracle LogMiner"
- Manage Oracle Database diagnostic data, as explained in "ADR

- Oracle 网络配置助理

  此工具在软件安装过程中自动运行。该助手使您能够在安装过程中配置基本网络组件，包括监听器名称及协议地址、命名方法、tnsnames.ora 文件中的网络服务名称、和目录服务器的使用。

- 监听器控制实用程序

  监听器控制实用程序使您可以配置监听器，以接收客户端连接 （请参见"Oracle 网络监听器"）。您可以通过企业管理器或作为一个独立的命令行应用程序来访问该实用程序。

- Oracle 连接管理器控制实用程序

  此命令行实用程序使您能够管理一个 **Oracle 连接管理器**，它是一个路由器，通过它，一个客户端的连接请求能被中转到下一个跃点，或直接连到数据库。可以使用实用程序命令在一个或多个 Oracle 连接管理器上执行基本的管理功能。另外，您可以查看和更改参数设置。

另见：

- "Oracle 网络体系结构概述"
- 《*Oracle 数据库网络服务管理员指南*》和 《*Oracle 数据库网络服务参考*》了解 Oracle 网络服务工具的更多信息

数据移动和分析工具

Oracle 数据库包括了几个实用程序来辅助数据移动和分析。例如，您可以使用数据库实用程序：

- 将数据从操作系统文件加载到 Oracle 数据库表，如"SQL*Loader"中所述
- 将数据和元数据从一个数据库移动到另一个数据库，如"Oracle 数据泵导入和导出"中所述
- 通过 SQL 接口查询重做日志文件，如"Oracle LogMiner"中所述
- 管理 Oracle 数据库诊断数据，如"ADR 命令解释器 （ADRCI）"中所

Command Interpreter (ADRCI)"

Other tasks include performing physical data structure integrity checks on an offline database or data file with DBVERIFY, or changing the database identifier (DBID) or database name for an operational database using the DBNEWID utility.

Note:

Tools related to backup and recovery are covered in "Backup and Recovery".

See Also:

Oracle Database Utilities to learn about DBVERIFY and DBNEWID

SQL*Loader

SQL*Loader loads data from external files, called **data files**, into database tables. It has a powerful data parsing engine that puts little limitation on the format of the data in the data file. You can use SQL*Loader to perform tasks such as:

- Loading data from multiple data files into multiple tables

  You store the data to be loaded in SQL*Loader data files. The SQL*Loader control file is a text file that contains DDL instructions that SQL*Loader uses to determine where to find the data, how to parse and interpret it, where to insert it, and more.

  Note:

  The SQL*Loader data files and control file are unrelated to the Oracle Database data files and control file.

- Control various aspects of the load operation

  For example, you can selectively load data, specify the data character set (see "Character Sets"), manipulate the data with SQL functions, generate unique sequential key values in specified columns, and so on. You can also generate sophisticated error reports.

- Use either conventional or direct path loading

  A **conventional path load** executes SQL `INSERT` statements to populate tables. In contrast, a **direct path load** eliminates much of the database overhead by formatting data blocks and writing them directly to the database files. Direct writes operate on blocks

述

其他任务，包括使用 DBVERIFY 对脱机数据库或数据文件执行物理数据结构完整性检查，或使用 DBNEWID 实用程序更改一个操作数据库的数据库标识符 (DBID） 或数据库名称。

注意：

与备份和恢复相关的工具在"备份和恢复"中介绍。

另见：

《*Oracle 数据库实用程序*》了解 DBVERIFY 和 DBNEWID

SQL*加载器

SQL*加载器将数据从称为**数据文件**的外部文件加载到数据库表中。它有一个功能强大的数据分析引擎，对数据文件中的数据格式几乎没什么限制。您可以使用 SQL*加载器执行如下任务：

- 将数据从多个数据文件加载到多个表

  您将要加载的数据存储在 SQL*加载器的数据文件中。SQL*加载器控制文件是一个文本文件，包含一些 DDL 指令，被 SQL * 加载器用来确定在哪里可以找到数据、如何对数据进行分析和解释、将数据插入到何处等。

  注意：

  在 SQL*加载器的数据文件和控制文件，与 Oracle 数据库数据文件和控制文件无关。

- 控制加载操作的各个方面

  例如，您可以有选择地加载数据、指定数据字符集 （请参见"字符集"）、使用 SQL 函数操作数据、在指定列中生成唯一的连续键值等等。您还可以生成复杂的错误报告。

- 使用常规或直接路径加载

  常规路径加载通过执行 `INSERT` SQL 语句来填充表。相比之下，**直接路径加载**通过格式化数据块和将他们直接写入数据库文件，消除了很多数据库开销。直接写入对高水位以上的块进行操作，直接写入磁

above the high water mark and write directly to disk, bypassing the database buffer cache. Direct reads read directly from disk into the PGA, again bypassing the buffer cache.

A typical SQL*Loader session takes as input a SQL*Loader control file and one or more data files. The output is an Oracle database, a log file, a bad file, and potentially, a discard file. Figure 18-1 illustrates the flow of a typical SQL*Loader session.

Figure 18-1 SQL*Loader Session

Description of "Figure 18-1 SQL*Loader Session"

See Also:

Oracle Database 2 Day DBA and *Oracle Database Utilities* to learn about SQL*Loader

Oracle Data Pump Export and Import

Oracle Data Pump enables high-speed movement of data and metadata from one database to another. This technology is the basis for the following Oracle Database data movement utilities:

- Data Pump Export (Export)

  Export is a utility for unloading data and metadata into a set of operating system files called a **dump file set**. The dump file set is made up of one or more binary files that contain table data, database object metadata, and control information.
- Data Pump Import (Import)

盘，从而绕过了数据库缓冲区高速缓存。直接读取操作直接将数据从磁盘读入 PGA，同样绕过了缓冲区高速缓存。

一个典型的 SQL*加载器会话，以一个 SQL*加载器控制文件和一个或多个数据文件作为输入。其输出是一个 Oracle 数据库、 日志文件、 一个不正确的文件，和一个可能会有的丢弃文件。图 18-1 阐释了一个典型的 SQL*加载器会话。

图 18-1 SQL * 加载程序会话

Description of "Figure 18-1 SQL*Loader Session"

另见：

《*Oracle 数据库 2 日 DBA*》 和 《*Oracle 数据库实用程序*》了解 SQL*加载器

Oracle 数据泵导出和导入

Oracle 数据泵能够将数据和元数据，从一个数据库快速移动到另一个数据库。这项技术是以下 Oracle 数据库数据移动工具的基础：

- 数据泵导出 （导出）

  导出是一个实用程序，用于将数据和元数据卸载到一组称为**转储文件集**的操作系统文件。转储文件集是一个或多个二进制文件，其中包含表数据、 数据库对象的元数据、和控制信息等。

- 数据泵导入 （导入）

Import is a utility for loading an export dump file set into a database. You can also use Import to load a destination database directly from a source database with no intervening files, which allows export and import operations to run concurrently, minimizing total elapsed time.

Oracle Data Pump is made up of the following distinct parts:

- The command-line clients `expdp` and `impdp`

  These client make calls to the `DBMS_DATAPUMP` package to perform Oracle Data Pump operations (see "PL/SQL Packages").
- The `DBMS_DATAPUMP` PL/SQL package, also known as the **Data Pump API**
  This API provides high-speed import and export functionality.
- The `DBMS_METADATA` PL/SQL package, also known as the **Metadata API**
  This API, which stores object definitions in XML, is used by all processes that load and unload metadata.

Figure 18-2 shows how Oracle Data Pump integrates with SQL*Loader and external tables. As shown, SQL*Loader is integrated with the External Table API and the Data Pump API to load data into external tables (see "External Tables"). Clients such as Database Control and transportable tablespaces can use the Oracle Data Pump infrastructure.

Figure 18-2 Oracle Data Pump Architecture

导入是一个实用程序，用于将一个导出的转储文件集加载到数据库中。你也可以使用导入直接将数据从一个源数据库加载到目标数据库，而不需要中间文件，这使得导出和导入操作可以同时运行，最小化了总运行时间。

Oracle 数据泵由下列几个不同部分组成：

- 命令行客户端 expdp 和 impdp

  这些客户端调用 DBMS_DATAPUMP 包，以执行 Oracle 数据泵操作（请参阅"PL/SQL 包"）。
- DBMS_DATAPUMP PL/SQL 包，也称为**数据泵 API**

  此 API 提供了高速的导入和导出功能。
- DBMS_METADATA PL/SQL 包，也称为**元数据 API**

  此 API 以 XML 形式存储对象的定义，所有加载和卸载元数据的进程都会使用此 API。

图 18-2 显示了 SQL Oracle 数据泵如何与 SQL*加载器和外部表集成。如图所示， SQL*加载器与外部表 API 和数据泵 API 集成，以将数据加载到外部表 （请参见外部表）。如数据库控制之类的客户端和移动式表空间可以使用 Oracle 数据泵基础设施。

图 18-2 Oracle 数据泵体系结构

Description of "Figure 18-2 Oracle Data Pump Architecture"

See Also:

- *Oracle Database Utilities* for an overview of Oracle Data Pump
- *Oracle Database PL/SQL Packages and Types Reference* for a description of `DBMS_DATAPUMP` and `DBMS_METADATA`

Oracle LogMiner

Oracle LogMiner enables you to query redo log files through a SQL interface. Potential uses for data contained in redo log files include:

- Pinpointing when a logical corruption to a database, such as errors made at the application level, may have begun
- Detecting user error
- Determining what actions you would have to take to perform fine-grained recovery at the transaction level
- Using trend analysis to determine which tables get the most updates and inserts
- Analyzing system behavior and auditing database use through the LogMiner comprehensive relational interface to redo log files

LogMiner is accessible through a command-line interface or through the Oracle LogMiner Viewer GUI, which is a part of Enterprise Manager.

Description of "Figure 18-2 Oracle Data Pump Architecture"

另见：

- 《*Oracle 数据库实用程序*》关于 Oracle 数据泵概述
- 《*Oracle 数据库 PL/SQL 包和类型参考*》关于 DBMS_DATAPUMP 和 DBMS_METADATA 的说明

Oracle LogMiner

Oracle LogMiner 使您能够通过 SQL 接口查询重做日志文件。包含在重做日志文件中的数据的潜在用途有：

- 查明某个数据库逻辑损坏是何时开始发生的，如一些应用程序级别的错误
- 检测用户错误
- 确定你必须要执行哪些操作，以进行事务级别的细粒度恢复
- 使用趋势分析，以确定哪些表上的更新和插入最多
- 通过 LogMiner 针对重做日志文件全面的关系接口，分析系统行为和审核数据库使用

LogMiner 可通过命令行界面或属于企业管理器的 Oracle LogMiner 查看器 GUI 来访问。

See Also:
Oracle Database Utilities to learn more about LogMiner

ADR Command Interpreter (ADRCI)

ADRCI is a command-line utility that enables you to investigate problems, view health check reports, and package and upload first-failure diagnostic data to Oracle Support. You can also use the utility to view the names of the trace files in the Automatic Diagnostic Repository (ADR) (ADR) and to view the alert log. ADRCI has a rich command set that you can use interactively or in scripts.

See Also:

- "Automatic Diagnostic Repository"
- *Oracle Database Utilities* and *Oracle Database Administrator's Guide* for more information on ADR and ADRCI

Topics for Database Administrators

Chapter 17 describes topics important for both developers and DBAs. This section covers topics that are most essential to DBAs and that have not been discussed elsewhere in the manual.

This section contains the following topics:

- Backup and Recovery
- Memory Management
- Resource Management and Task Scheduling
- Performance Diagnostics and Tuning

Backup and Recovery

Backup and recovery is the set of concepts, procedures, and strategies involved in protecting the database against data loss caused by media failure or users errors. In general, the purpose of a backup and recovery strategy is to protect the database against data loss and reconstruct lost data.

A **backup** is a copy of data. A backup can include crucial parts of the database such as data files, the server parameter file, and control file. A sample backup and recovery scenario is a failed disk drive that causes the loss of a data file. If a backup of the lost file exists, then you can restore and recover it. **Media recovery** refers to the operations involved in restoring data to its state before the loss occurred.

See Also:

另见：
《*Oracle 数据库实用程序*》了解 LogMiner 的更多信息

ADR 命令解释器 (ADRCI)

ADRCI 是一个命令行实用程序，使您可以调查问题、 查看健康检查报告、和将首次故障诊断数据打包并上传到 Oracle 支持。你也可以使用此实用程序查看自动诊断存储库 (ADR)中的跟踪文件的名称，并查看警报日志。ADRCI 有一个丰富的命令集，您可以交互方式或在脚本中使用。

另见：

- 自动诊断存储库
- 《*Oracle 数据库实用程序*》和《*Oracle 数据库管理员指南*》关于 ADR 和 ADRCI 的详细信息

数据库管理员的主题

第 17 章描述了对开发人员和数据库管理员都很重要的主题。而本节将介绍之前从未讨论过的、对数据库管理员最重要的主题。

本节包含以下主题：

- 备份与恢复
- 内存管理
- 资源管理与任务调度
- 性能诊断和调优

备份和恢复

备份和恢复是为保护数据库，以防止由于介质故障或用户错误造成数据丢失，所涉及到的一组概念、 过程、和策略。通常，备份和恢复策略的目的是保护数据库，以防数据丢失或重建丢失的数据。

备份是数据的副本。备份可能包含数据库的关键部分，如数据文件、 服务器参数文件、和控制文件等。一个备份和恢复的示例场景是，一个磁盘驱动器发生故障导致数据文件丢失。如果丢失文件的备份存在，则您可以还原并恢复该文件。**介质恢复**指的是将数据还原到其丢失之前的状态所涉及的操作。

另见：

Oracle Database 2 Day DBA and *Oracle Database Backup and Recovery User's Guide* for backup and recovery concepts and tasks

Backup and Recovery Techniques

You can use the following means to back up and recover an Oracle database:

- Recovery Manager (RMAN)

  RMAN is an Oracle Database utility that integrates with an Oracle database to perform backup and recovery activities, including maintaining a repository of historical backup metadata in the control file of every database that it backs up. RMAN can also maintain a centralized backup repository called a **recovery catalog** in a different database. RMAN is an Oracle Database feature and does not require separate installation.
  RMAN is integrated with **Oracle Secure Backup**, which provides reliable, centralized tape backup management, protecting file system data and Oracle Database files. The Oracle Secure Backup SBT interface enables you to use RMAN to back up and restore database files to and from tape and internet-based Web Services such as Amazon S3. Oracle Secure Backup supports almost every tape drive and tape library in SAN and SCSI environments.
  RMAN and Oracle Secure Backup are accessible both from the command line and from Enterprise Manager.
- User-Managed techniques

  As an alternative to RMAN, you can use operating system commands such as the Linux `dd` for backing up and restoring files and the SQL*Plus `RECOVER` command for media recovery. User-managed backup and recovery is fully supported by Oracle, although RMAN is recommended because it is integrated with Oracle Database and simplifies administration.

Figure 18-3 shows basic RMAN architecture. The RMAN client, accessible through Enterprise Manager, uses server sessions on a target database to back up data to disk or tape. RMAN can update an external recovery catalog with backup metadata.

Figure 18-3 RMAN Architecture

《*Oracle 数据库 2 日 DBA*》 和 《*Oracle 数据库备份和恢复用户指南*》关于备份和恢复的概念和任务

备份和恢复技术

可以使用以下方式来备份和恢复 Oracle 数据库：

- 恢复管理器(RMAN)

  RMAN 是一个与 Oracle 数据库集成的 Oracle 数据库实用程序，用于执行备份和恢复操作，包括在每个被备份的数据库的控制文件中维护一个历史备份元数据的存储库。RMAN 还可以在一个不同的数据库中维护一个称为**恢复目录**的集中化的备份存储库。RMAN 是一个 Oracle 数据库功能，不需要单独安装。

  RMAN 与 **Oracle 安全备份**集成，提供可靠的、 集中的磁带备份管理，以保护文件系统数据和 Oracle 数据库文件。Oracle 安全备份 SBT 接口使您能够使用 RMAN 将数据库文件备份到磁带或基于互联网的网络服务（如亚马逊 S3），或反之从其中还原数据文件。Oracle 安全备份支持 SAN 和 SCSI 环境中的几乎每种磁带驱动器和磁带库。

  RMAN 备份与 Oracle 安全备份都可从命令行和从企业管理器中访问。
- 用户管理技术

  作为 RMAN 的替代方法，您可以使用操作系统命令如 Linux dd 来备份和还原文件，和用 SQL*Plus 命令进行介质恢复。尽管由于 RMAN 与 Oracle 数据库集成在一起并简化了管理而被作为推荐的方式，但 Oracle 仍完全支持用户管理的备份和恢复。

图 18-3 显示了 RMAN 的基本体系结构。RMAN 客户可通过企业管理器，使用在目标数据库上的服务器会话将数据备份到磁盘或磁带。RMAN 可以将备份元数据更新到一个外部恢复目录。

图 18-3 RMAN 体系结构

Description of "Figure 18-3 RMAN Architecture"

Whichever backup and recovery technique you use, Oracle recommends that you configure a fast recovery area. This database-managed directory, file system, or Oracle ASM disk group centralizes backup and recovery files, including active control files, online and archived redo log files, and backups. Oracle Database recovery components interact with the fast recovery area to ensure database recoverability.

See Also:

- *Oracle Database 2 Day DBA* to learn how to perform backup and recovery with Enterprise Manager
- *Oracle Database Backup and Recovery User's Guide* for an overview of backup and recovery solutions
- *Oracle Database Administrator's Guide* for information about how to set up and administer the fast recovery area
- *Oracle Secure Backup Administrator's Guide* for an overview of Oracle Secure Backup

Database Backups

Database backups can be either physical or logical. **Physical backups**, which are the primary concern in a backup and recovery strategy, are copies of physical database files. You can make physical backups with RMAN or operating system utilities.

Description of "Figure 18-3 RMAN Architecture"

无论你使用哪种备份和恢复技术， Oracle 都建议您配置一个快速恢复区。它可能是一个数据库管理的目录、 文件系统、或 Oracle ASM 磁盘组，用于集中化管理备份和恢复相关文件，包括活动控制文件、联机和归档重做日志文件、和备份。Oracle 数据库恢复组件与快速恢复区进行交互，以确保数据库的可恢复性。

另见：

- 《*Oracle 数据库 2 日 DBA*》了解如何使用企业管理器执行备份和恢复
- 《*Oracle 数据库备份和恢复用户指南*》关于备份和恢复解决方案概述
- 《*Oracle 数据库管理员指南*》有关如何设置和管理快速恢复区的信息
- 《*Oracle 安全备份管理员指南*》 关于 Oracle 安全备份的概述

数据库备份

数据库备份可以是物理的或逻辑的。**物理备份**是备份和恢复策略的主要关注目标，它是物理数据库文件的副本。您可以使用 RMAN 或操作系统实用程序进行物理备份。

In contrast, **logical backups** contain logical data such as tables and stored procedures. You can extract logical data with an Oracle Database utility such as Data Pump Export and store it in a binary file. Logical backups can supplement physical backups.

Physical backups have large granularity and limited transportability, but are very fast. Logical backups have fine granularity and complete transportability, but are slower than physical backups.

See Also:

Oracle Database Backup and Recovery User's Guide to learn about physical and logical backups

Whole and Partial Database Backups

A **whole database backup** is a backup of every data file in the database, plus the control file. Whole database backups are the most common type of backup.

A **partial database backup** includes a subset of the database: individual tablespaces or data files. A **tablespace backup** is a backup of all the data files in a tablespace or in multiple tablespaces. Tablespace backups, whether consistent or inconsistent, are valid only if the database is operating in `ARCHIVELOG` mode because redo is required to make the restored tablespace consistent with the rest of the database.

Consistent and Inconsistent Backups

A whole database backup is either consistent or inconsistent. In a **consistent backup**, all read/write data files and control files have the same checkpoint SCN, guaranteeing that these files contain all changes up to this SCN. This type of backup does not require recovery after it is restored.

A consistent backup of the database is only possible after a consistent shutdown (see "Shutdown Modes") and is the only valid backup option for a database operating in `NOARCHIVELOG` mode. Other backup options require media recovery for consistency, which is not possible without applying archived redo log files.

Note:

If you restore a consistent whole database backup without applying redo, then you lose all transactions made after the backup.

In an **inconsistent backup**, read/write data files and control files are not guaranteed to have the same checkpoint SCN, so changes can be missing. All online backups are necessarily inconsistent because data files can be modified while backups occur.

相比之下，逻辑备份则包含逻辑数据（如表和存储过程）。可以使用 Oracle 数据库实用程序（如数据泵）提取逻辑数据，并将其存储在一个二进制文件中。逻辑备份可以作为物理备份的补充。

物理备份的粒度更大，且可传输性有限，但速度非常快。逻辑备份粒度更细且具有完全的可传输性，但比物理备份速度更慢。

另见：

《*Oracle 数据库备份和恢复用户指南*》了解物理和逻辑备份

整体和部分数据库备份

整体数据库备份，即是在数据库中的每个数据文件的备份，再加上控制文件备份。整体数据库备份是最常见的备份。

部分数据库备份包含数据库的一个子集： 个别表空间或数据文件。**表空间备份**是一个或多个表空间中所有数据文件的备份。无论表空间备份是一致的或是不一致的，都只仅当数据库运行在归档模式下才是有效的，因为需要重做来使表空间恢复到与数据库的其余部分一致。

一致和不一致的备份

整体数据库备份可能是一致的或不一致的。对一个**一致的备份**，所有的读/写数据文件和控制文件都具有相同的检查点 SCN，以保证这些文件包含的到该 SCN 为止的所有更改。这种类型的备份在还原后不需要恢复。

一致备份仅在数据库一致关闭后才有可能进行（请参见"关闭模式"），并且这也是运行在*非归档*模式下数据库的唯一有效的备份选项。其他备份选项需要介质恢复以保证其一致性，而这在不应用归档重做日志文件的情况下是不可能的。

注意：

如果你还原一个一致的整体数据库备份而不应用重做，则您将丢失备份之后的所有事务。

在一个**不一致的备份**中，读/写数据文件和控制文件不保证有相同的检查点 SCN，因此可能会丢失更改。由于进行联机备份时可以修改数据文件，因此联机备份不一定是一致的。

Inconsistent backups offer superior availability because you do not have to shut down the database to make backups that fully protect the database. If the database runs in ARCHIVELOG mode, and if you back up the archived redo logs and data files, then inconsistent backups can be the foundation for a sound backup and recovery strategy.

See Also:

Oracle Database Backup and Recovery User's Guide to learn more about inconsistent backups

不一致的备份提供了更优异的可用性，因为你不必关闭数据库就可以进行能充分保护数据库的备份。如果数据库在归档模式下运行，并且您备份了归档重做日志和数据文件，则这种不一致的备份完全可以作为备份和恢复策略的基础。

另见：

《*Oracle 数据库备份和恢复用户指南*》 了解不一致备份的更多信息

Backup Sets and Image Copies

The RMAN BACKUP command generates either **image copies** or **backup sets**. An image copy is a bit-for-bit, on-disk duplicate of a data file, control file, or archived redo log file. You can create image copies of physical files with operating system utilities or RMAN and use either tool to restore them.

Note:

Unlike operating system copies, RMAN validates the blocks in the file and records the image copy in the RMAN repository.

RMAN can also create backups in a proprietary format called a **backup set**. A backup set contains the data from one or more data files, archived redo log files, or control files or server parameter file. The smallest unit of a backup set is a binary file called a **backup piece**. Backup sets are the only form in which RMAN can write backups to sequential devices such as tape drives.

Backup sets enable tape devices to stream continuously. For example, RMAN can mingle blocks from slow, medium, and fast disks into one backup set so that the tape device has a constant input of blocks. Image copies are useful for disk because you can update them incrementally, and also recover them in place.

See Also:

Oracle Database Backup and Recovery User's Guide to learn more about backup sets and image copies

备份集和映像拷贝

RMAN BACKUP 命令生成**映像拷贝**或**备份集**。映像拷贝是对磁盘上的数据文件、 控制文件、或归档重做日志文件按位进行复制的副本。可以使用操作系统实用程序或 RMAN 创建物理文件的映像拷贝，并且可以使用任一工具来还原它们。

注意：

与操作系统拷贝不同， RMAN 会验证文件中的块，并将映像拷贝记录在 RMAN 存储库中。

RMAN 也可以按称为**备份集**的专有格式来创建备份。备份集包含一个或多个数据文件、归档重做日志文件、 控制文件、或服务器参数文件中的数据。备份集的最小单位是一个称为**备份片**的二进制文件。备份集是 RMAN 可以将备份写入顺序设备（如磁带驱动器）的唯一形式。

备份集使磁带设备能够使数据流连续不断。例如，RMAN 可以将多个来自慢速、中速、快速的磁盘的数据块混合到一个备份集，使得磁带设备有一个稳定的数据块输入流。映像拷贝比较适合用于磁盘，因为您可以进行增量更新，还可以对其进行就地恢复。

另见：

《*Oracle 数据库备份和恢复用户指南*》了解备份集和映像拷贝的更多信息

Data Repair

While several problems can halt the normal operation of a database or affect I/O operations, only the following typically require DBA intervention and data repair:

- Media failures

  A **media failure** occurs when a problem external to the database

数据修复

虽然有好多个问题都可以导致数据库停止正常运行或影响其 I/O 操作，但通常只有在以下情况才需要 DBA 干预和数据修复：

- 介质故障

  当数据库之外的某个问题导致数据库无法读取或写入文件时，将发生

prevents it from reading from or writing to a file. Typical media failures include physical failures, such as head crashes, and the overwriting, deletion, or corruption of a database file. Media failures are less common than user or application errors, but a sound recovery strategy must prepare for them.

- User errors

  A user or application may make unwanted changes to your database, such as erroneous updates, deleting the contents of a table, or dropping database objects (see "Human Errors"). A good backup and recovery strategy enables you to return your database to the desired state, with the minimum possible impact upon database availability, and minimal DBA effort.

Typically, you have multiple ways to solve the preceding problems. This section summarizes some of these solutions.

See Also:

Oracle Database 2 Day DBA and *Oracle Database Backup and Recovery User's Guide* for data repair concepts

Data Recovery Advisor

The **Data Recovery Advisor** tool automatically diagnoses persistent data failures, presents appropriate repair options, and executes repairs at the user's request. By providing a centralized tool for automated data repair, Data Recovery Advisor improves the manageability and reliability of an Oracle database and thus helps reduce recovery time.

The database includes a framework called **Health Monitor** for running diagnostic checks. A **checker** is a diagnostic operation or procedure registered with Health Monitor to assess the health of the database or its components. The health assessment is known as a **data integrity check** and can be invoked reactively or proactively.

A **failure** is a persistent data corruption detected by a data integrity check. Failures are normally detected reactively. A database operation involving corrupted data results in an error, which automatically invokes a data integrity check that searches the database for failures related to the error. If failures are diagnosed, then the database records them in the Automatic Diagnostic Repository (ADR).

After failures have been detected by the database and stored in ADR, Data Recovery Advisor automatically determines the best repair options and their impact on the database. Typically, Data Recovery Advisor generates

介质故障。典型的介质故障包括数据文件头崩溃、数据文件被覆盖、删除、或损坏等物理故障。介质故障不像用户错误或应用程序错误那么常见，但一个完备的恢复策略对此必须要有所准备。

- 用户错误

  一个用户或应用程序可能会对数据库做出不正确的更改，如错误的更新、删除了某个表的内容、或删除了某些数据库对象 （见"人为错误"）。一个好的备份和恢复策略，使你仅用最小的 DBA 工作量，且仅对数据库可用性造成最小的影响，就可以将您的数据库返回到所需的状态。

通常，你可能有多种方法来解决上述问题。本节总结了其中的部分解决方案。

另见：

《*Oracle 数据库 2 日 DBA*》和《*Oracle 数据库备份和恢复用户指南*》 关于数据修复的概念

数据恢复顾问

数据恢复顾问工具自动诊断持久性数据故障、提供适当的修复选项、并按用户的要求执行修复。通过为自动化数据修复提供集中式的工具，数据恢复顾问提高了 Oracle 数据库的可管理性和可靠性，并有助于减少恢复时间。

数据库包括一个称为**健康监视器**的框架，用于运行诊断检查。**检查器**是注册到健康监视器的诊断操作或过程，用于评估数据库或其组件的运行状况。健康评估也被称为**数据完整性检查**，可按被动或主动方式调用。

故障是由数据完整性检查检测到的持久性数据损坏。故障通常是被动检测到的。涉及损坏数据的数据库操作将导致一个错误，这将自动调用一个数据完整性检查，以在数据库中搜索与此错误相关的故障。如果故障被诊断出来，则数据库将它们记录到自动诊断存储库 （ADR）中。

当数据库检测到故障并将其存储在 ADR 中后，数据恢复顾问会自动确定最佳的修复选项和他们对数据库的影响。通常，数据恢复顾问为每个故障或每组故障同时生成手动和自动修复选项。

both manual and automated repair options for each failure or group of failures.

Before presenting an automated repair option, Data Recovery Advisor validates it for the specific environment and for the availability of media components required to complete the proposed repair. If you choose an automatic repair, then Oracle Database executes it for you. The Data Recovery Advisor tool verifies the repair success and closes the appropriate failures.

See Also:

Oracle Database 2 Day DBA and *Oracle Database Backup and Recovery User's Guide* to learn how to use Data Recovery Advisor

Oracle Flashback Technology

Oracle Database provides a group of features known as **Oracle Flashback Technology** that support viewing past states of data, and winding data back and forth in time, without needing to restore backups. Depending on the database changes, flashback features can often reverse unwanted changes more quickly and with less impact on availability than media recovery.

The following flashback features are most relevant for backup and recovery:

- Flashback Database

  You can rewind an Oracle database to a previous time to correct problems caused by logical data corruptions or user errors. Flashback Database can also be used to complement Data Guard, Data Recovery Advisor, and for synchronizing clone databases. Flashback Database does not restore or perform media recovery on files, so you cannot use it to correct media failures such as disk crashes.
- Flashback Table

  You can rewind tables to a specified point in time with a single SQL statement. You can restore table data along with associated indexes, triggers, and constraints, while the database is online, undoing changes to only the specified tables. Flashback Table does not address physical corruption such as bad disks or data segment and index inconsistencies.

在给出自动修复选项之前，数据恢复顾问会验证针对该特定的环境和为完成建议的修复所需的介质组件的可用性。如果选择自动修复，则 Oracle 数据库将为你执行它。若数据恢复顾问工具验证修复成功，并关闭相应的故障。

另见：

《*Oracle 数据库 2 日 DBA*》 和《*Oracle 数据库备份和恢复用户指南*》学习如何使用数据恢复顾问

Oracle 闪回技术

Oracle 数据库提供了一组称为 **Oracle 闪回技术**的功能，以支持查看数据过去的状态，将数据倒回到某个时间，而无需恢复备份数据。取决于对数据库所作的更改，闪回功能通常可以比介质恢复更快，且影响更小地撤消错误的更改。

以下的闪回功能与备份和恢复密切相关：

- 闪回数据库

  您可以将 Oracle 数据库倒回到一个之前的时间，以纠正逻辑数据损坏或用户错误引起的问题。闪回数据库也可以用于辅助数据卫士、数据恢复顾问、和同步克隆数据库。闪回数据库不还原文件或执行介质恢复，因此您不能使用它来纠正磁盘崩溃之类的介质故障。

- 闪回表

  您可以使用单个 SQL 语句将表倒回到一个指定的时间点。您可以在数据库处于联机状态时，将表数据及其关联的索引、 触发器、和约束一起还原，以撤消对指定表的更改。闪回表不能解决磁盘破坏之类的物理损坏，或数据段和索引不一致等问题。

- Flashback Drop

  You can reverse the effects of a DROP TABLE operation. Flashback Drop is substantially faster than recovery mechanisms such as point-in-time recovery and does not lead to loss of recent transactions or downtime.

See Also:

- *Oracle Database 2 Day DBA* and *Oracle Database Backup and Recovery User's Guide* to learn more about flashback features

- *Oracle Database SQL Language Reference* and *Oracle Database Backup and Recovery Reference* to learn about the FLASHBACK DATABASE statement

Block Media Recovery

A **block corruption** is a data block that is not in a recognized Oracle format, or whose contents are not internally consistent (see "Data Corruption"). Block media recovery is a technique for restoring and recovering corrupt data blocks while data files are online. If only a few blocks are corrupt, then block recovery may be preferable to data file recovery.

See Also:

Oracle Database Backup and Recovery User's Guide to learn how to perform block media recovery

Data File Recovery

Data file recovery repairs a lost or damaged current data file or control file. It can also recover changes lost when a tablespace went offline without the OFFLINE NORMAL option.

Media recovery is necessary if you restore a backup of a data file or control file or a data file is taken offline without the OFFLINE NORMAL option. The database cannot be opened if online data files needs media recovery, nor can a data file that needs media recovery be brought online until media recovery completes.

To **restore** a physical backup of a data file or control file is to reconstruct it and make it available to Oracle Database. To **recover** a backup is to apply archived redo log files, thereby reconstructing lost changes. RMAN can also recover data files with **incremental backups**, which contain only blocks modified after a previous backup.

Unlike instance recovery, which automatically applies changes to online

- 闪回删除

  您可以撤消 DROP TABLE 操作的影响。闪回删除比基于时间点的恢复之类的机制快得多，而且也不会导致最近事务丢失或停机时间。

另见：

- 《*Oracle 数据库 2 日 DBA*》 和 《*Oracle 数据库备份和恢复用户指南*》了解闪回功能的更多信息

- 《*Oracle 数据库 SQL 语言参考*》和 《*Oracle 数据库备份和恢复参考*》了解 FLASHBACK DATABASE 语句

块介质恢复

块损坏即一个数据块的格式不能被 Oracle 辨识，或其内容在内部不一致（见"数据损坏"）。数据块介质恢复是一种当数据文件处于联机状态时，对损坏的数据块进行还原和恢复的技术。如果仅几个块已损坏，则块恢复可能比数据文件恢复更为可取。

另见：

《*Oracle 数据库备份和恢复用户指南*》了解如何执行数据块介质恢复

数据文件恢复

数据文件恢复用于修复已丢失或损坏的当前数据文件或控制文件。它也可以恢复表空间由于未使用 OFFLINE NORMAL 选项时，在离线期间丢失的更改。

当您还原了数据文件或控制文件的备份或由于未使用 OFFLINE NORMAL 选项而脱机的数据文件时，有必要进行介质恢复。如果处于联机状态的数据文件需要介质恢复，则数据库不能处于打开状态；需要介质恢复的数据文件必须在介质恢复完成之后才能将其联机。

还原数据文件或控制文件的物理备份，是为了重建它并使其可用于 Oracle 数据库。**恢复**备份就是应用归档重做日志文件，从而重建丢失的更改。RMAN 也可以使用**增量备份**来恢复数据文件，它仅包含自前一次备份后被修改的块。

与自动将更改应用于在线文件的实例恢复不同，介质恢复必须由用户调用，

files, media recovery must be invoked by a user and applies archived redo log files to restored backups. Data file media recovery can only operate on offline data files or data files in a database that is not opened by any instance.

Data file media recovery differs depending on whether all changes are applied:

- Complete recovery

  Complete recovery applies *all* redo changes contained in the archived and online logs to a backup. Typically, you perform complete media recovery after a media failure damages data files or the control file. You can perform complete recovery on a database, tablespace, or data file.
- Incomplete recovery

  Incomplete recovery, also called **database point-in-time recovery**, results in a noncurrent version of the database. In this case, you do not apply all of the redo generated after the restored backup. Typically, you perform point-in-time database recovery to undo a user error when Flashback Database is not possible.

  To perform incomplete recovery, you must restore all data files from backups created before the time to which you want to recover and then open the database with the `RESETLOGS` option when recovery completes. Resetting the logs creates a new stream of log sequence numbers starting with log sequence 1.

  Note:

  If current data files are available, then Flashback Database is an alternative to DBPITR.

  The **tablespace point-in-time recovery (TSPITR)** feature lets you recover one or more tablespaces to a point in time older than the rest of the database.

See Also:

- "Overview of Instance Recovery"
- *Oracle Database 2 Day DBA* and *Oracle Database Backup and Recovery User's Guide* for media recovery concepts

Memory Management

Memory management involves maintaining optimal sizes for the Oracle

并且将归档重做日志文件应用于已还原的备份。数据文件介质恢复只可以对脱机数据文件、或未被任何实例打开的数据库中的数据文件进行操作。

基于是否应用了所有的更改，数据文件的介质恢复有不同的情况：

- 完全恢复

  完全恢复将归档和联机日志中所包含的所有重做更改应用于备份。通常，您在介质故障损坏了数据文件或控制文件之后，执行完整的介质恢复。您可以对数据库、 表空间、或数据文件执行完全恢复。

- 不完全恢复

  不完全恢复，也称为**基于时间点的数据库恢复**，会导致一个非当前的数据库版本。在这种情况下，您没有应用已还原备份之后生成的所有重做。通常，在发生了用户错误，而又不能使用闪回数据库来解决时，你可以执行基于时间点的数据库恢复。

  要执行不完全恢复，必须从某个你所要恢复到的目标时间之前创建的备份中还原所有数据文件，并在完成恢复后，使用 `RESETLOGS` 选项打开数据库。重置日志会创建一个新的从 1 开始的日志序列流。

  注意：

  如果当前数据文件处于可用状态，则闪回数据库可以作为 DBPITR 的替代方法。

  表空间时间点恢复 (TSPITR) 功能允许您将一个或多个表空间恢复到早于数据库的其余部分中的某个时间点。

另见：

- "实例恢复概述"
- 《*Oracle 数据库 2 日 DBA*》 和 《*Oracle 数据库备份和恢复用户指南*》关于介质恢复的概念

内存管理

内存管理涉及当数据库的需求变更时，为 Oracle 实例内存结构维护最佳大

instance memory structures as demands on the database change. Initialization parameter settings determine how SGA and instance PGA memory is managed.

Figure 18-4 shows a decision tree for memory management options. The following sections explain the options in detail.

Figure 18-4 Memory Management Methods

Description of "Figure 18-4 Memory Management Methods"

See Also:

Chapter 14, "Memory Architecture" to learn more about the SGA and PGA

Automatic Memory Management

In **automatic memory management**, Oracle Database manages the SGA and instance PGA memory completely automatically. This method is the simplest and is strongly recommended by Oracle.

The only user-specified controls are the **target memory size** initialization parameter (`MEMORY_TARGET`) and optional **maximum memory size** initialization parameter (`MEMORY_MAX_TARGET`). Oracle Database tunes to the target memory size, redistributing memory as needed between the SGA and the instance PGA.

Figure 18-5 shows a database that sometimes processes jobs submitted by online users and sometimes batch jobs. Using automatic memory management, the database automatically adjusts the size of the large pool and database buffer cache depending on which type of jobs are running.

小。由初始化参数设置确定如何管理实例的 SGA 和 PGA 内存。

图 18-4 显示内存管理选项的决策树。以下各节说明详细信息的选项。

图 18-4 内存管理方法

Description of "Figure 18-4 Memory Management Methods"

另见：

第 14 章， "内存体系结构"了解 SGA 和 PGA 的更多信息

自动内存管理

在**自动内存管理**中， Oracle 数据库完全自动地管理实例的 SGA 和 PGA 内存。此方法最简单，并被 Oracle 强烈建议。

唯一需要用户指定控制项是**目标内存大小**初始化参数 (MEMORY_TARGET) 和可选的**最大内存大小**初始化参数 (MEMORY_MAX_TARGET)。Oracle 数据库优化目标内存大小，根据需要在实例的 SGA 和 PGA 之间重新分配内存。

图 18-5 显示了一个数据库，它有时处理由在线用户提交的作业，有时执行批处理作业。使用自动内存管理，数据库将基于正在运行哪种作业类型自动调整大池和数据库缓冲区高速缓存的大小。

Figure 18-5 Automatic Memory Management

Description of "Figure 18-5 Automatic Memory Management"

If you create your database with DBCA and choose the basic installation option, then automatic memory management is enabled by default.

See Also:

Oracle Database 2 Day DBA and *Oracle Database Administrator's Guide* to learn about automatic memory management

Shared Memory Management of the SGA

If automatic memory management is not enabled, then the system must use **shared memory management** of the SGA. Shared memory management is possible in either of the following forms:

- Automatic shared memory management

  This mode enables you to exercise more direct control over the size of the SGA and is the default when automatic memory management is disabled. The database tunes the total SGA to the target size and dynamically tunes the sizes of SGA components. Oracle Database remembers the sizes of the automatically tuned components across instance shutdowns if you are using a server parameter file.
- Manual shared memory management

  In this mode, you set the sizes of several individual SGA components and manually tune individual SGA components on an

图 18-5 自动内存管理

Description of "Figure 18-5 Automatic Memory Management"

如果使用 DBCA 创建您的数据库，并选择基本安装选项，则默认情况下将启用自动内存管理。

另见：

《*Oracle 数据库 2 日 DBA*》和《*Oracle 数据库管理员指南*》了解自动内存管理

SGA 共享内存管理

如果未启用自动内存管理，则系统必须使用 SGA **共享内存管理**。共享内存管理是可能是下列形式之一：

- 自动共享内存管理

  此模式使您能够对 SGA 的大小执行更直接的控制，并且当自动内存管理处于禁用状态时它是缺省的。数据库将总 SGA 优化到这个目标大小，并动态调整 SGA 组件的大小。如果您使用的是服务器参数文件，那么即使经历实例关闭，Oracle 数据库也能记住自动调整的组件大小。

- 手动共享内存管理

  在此模式下，由你设置各个 SGA 组件的大小，并在使用过程中不断手动地调整个别 SGA 组件。您可以完全控制个别 SGA 组件大小。当

ongoing basis. You have complete control of individual SGA component sizes. The database defaults to this mode when both automatic memory management and automatic shared memory management are disabled.

See Also:

Oracle Database 2 Day DBA and *Oracle Database Administrator's Guide* to learn about shared memory management

Memory Management of the Instance PGA

If automatic memory management is not enabled, then the following modes are possible for management of PGA memory:

- Automatic PGA memory management

  When automatic memory management is disabled and `PGA_AGGREGATE_TARGET` is set to a nonzero value, the database uses **automatic PGA memory management**. In this mode, the `PGA_AGGREGATE_TARGET` specifies a target size for the instance PGA. The database then tunes the size of the instance PGA to this target and dynamically tunes the sizes of individual PGAs. If you do not explicitly set a target size, then the database automatically configures a reasonable default.
- Manual PGA memory management

  When automatic memory management is disabled and `PGA_AGGREGATE_TARGET` is set to `0`, the database defaults to **manual PGA management**. Previous releases of Oracle Database required the DBA to manually specify the maximum work area size for each type of SQL operator (such as a sort or hash join). This technique proved to be very difficult because the workload is always changing. Although Oracle Database supports the manual PGA memory management method, Oracle strongly recommends automatic memory management.

See Also:

Oracle Database Performance Tuning Guide to learn about PGA memory management

Summary of Memory Management Methods

Table 18-1 summarizes the various memory management methods. If you do not enable automatic memory management, then you must separately configure one memory management method for the SGA and one for the

自动内存管理和自动共享内存管理都被禁用时，数据库会缺省使用这种模式。

另见：

《*Oracle 数据库 2 日 DBA*》 和 《*Oracle 数据库管理员指南*》了解共享内存管理

实例的 PGA 内存管理

如果未启用自动内存管理，则可以下列模式之一管理 PGA 内存：

- 自动 PGA 内存管理

  当自动内存管理被禁用，且 PGA_AGGREGATE_TARGET 设置为非零值时，数据库将使用**自动 PGA 内存管理**。在此模式下，由 PGA_AGGREGATE_TARGET 指定实例 PGA 的目标大小。数据库则将实例 PGA 优化到这一目标大小，并动态调整各个 PGA 的大小。如果您未显式设置一个目标大小，则数据库将自动配置一个合理的默认值。
- 手动 PGA 内存管理

  当自动内存管理被禁用，且 PGA_AGGREGATE_TARGET 设置为 0 时，数据库将默认为**手动 PGA 管理**。以前版本的 Oracle 数据库要求 DBA 为每种类型的 SQL 操作（例如，排序或哈希联接）手动指定最大工作区大小。这种技术被证明是非常困难的，因为工作负荷总是在变化。虽然 Oracle 数据库支持手动 PGA 内存管理方法，但 Oracle 强烈建议使用自动内存管理。

另见：

《*Oracle 数据库性能优化指南*》了解 PGA 内存管理

内存管理方法总结

表 18-1 总结了各种内存管理方法。如果您不启用自动内存管理，则你必须为 SGA 和 PGA 分别配置一种内存管理方法。

PGA.

Note:

When automatic memory management is not enabled, the default method for the instance PGA is automatic PGA memory management.

注意：

当自动内存管理未启用时，则实例 PGA 的默认方法是自动 PGA 内存管理。

Table 18-1 Memory Management Methods

表 18-1 内存管理方法

Instance	SGA	PGA	Description	Initialization Parameters	描述	初始化参数
Auto	n/a	n/a	The database tunes the size of the instance based on a single instance target size.	You set: • Total memory target size for the database instance (MEMORY_TARGET) • Optional maximum memory size for the database instance (MEMORY_MAX_TARGET)	数据库基于单个实例目标大小，来优化实例的大小。	你设置： • 数据库实例的总内存目标大小（MEMORY_TARGET） • 可选的数据库实例最大内存大小（MEMORY_MAX_TARGET）
n/a	Auto	Auto	The database automatically tunes the SGA based on an SGA target. The database automatically tunes the PGA based on a PGA target.	You set: • SGA target size (SGA_TARGET) • Optional SGA maximum size (SGA_MAX_SIZE) • Instance PGA target size (PGA_AGGREGATE_TARGET)	数据库基于一个 SGA 目标大小，来自动优化 SGA 的大小。 数据库基于一个 PGA 目标大小，来自动优化 PGA 的大小。	你设置： • SGA 目标大小（SGA_TARGET） • 可选的 SGA 最大大小（SGA_MAX_SIZE） • 实例 PGA 目标大小（PGA_AGGREGATE_TARGET）

n/a	Auto	Manual	The database automatically tunes the SGA based on an SGA target.
			
			You control the PGA manually, setting the maximum work area size for each type of SQL operator.
				You set:
				• SGA target size (SGA_TARGET)
				• Optional SGA maximum size (SGA_MAX_SIZE)
				• PGA work area parameters such as SORT_AREA_SIZE, HASH_AREA_SIZE, and BITMAP_MERGE_AREA_SIZE
					数据库基于一个 SGA 目标大小，来自动优化 SGA 的大小。
					
					您可以手动控制 PGA，设置每种类型的 SQL 操作的最大工作区大小。
						你设置：
						• SGA 目标大小（SGA_TARGET）
						• 可选的 SGA 最大大小（SGA_MAX_SIZE）
						• PGA 工作区参数，如
						  ○ SORT_AREA_SIZE、
						  ○ HASH_AREA_SIZE、
						  ○ BITMAP_MERGE_AREA_SIZE

n/a	Manual	Auto	You control the SGA manually by setting individual component	You set: • Shared pool size (SHARED POOL SIZE)	你可以通过设置单个组件大小，手动控制	你设置： • 共享池大小（SHARED_POOL_SIZE）

			sizes. The database automatically tunes the PGA based on a PGA target.	• Buffer cache size (DB_CACHE_SIZE) • Large pool size (LARGE_POOL_SIZE) • Java pool size (JAVA_POOL_SIZE) • Streams pool size (STREAMS_POOL_SIZE) • Instance PGA target size (PGA_AGGREGATE_TARGET)	SGA。 数据库基于一个 PGA 目标大小，来自动优化 PGA 的大小。	• 缓冲区高速缓存大小（DB_CACHE_SIZE） • 大池大小（LARGE_POOL_SIZE） • Java 池大小（JAVA_POOL_SIZE） • 流池大小（STREAMS_POOL_SIZE） • 实例 PGA 目标大小（PGA_AGGREGATE_TARGET）

n/a	Manual	Manual	You must manually configure SGA component sizes.
			
			You control the PGA manually, setting the maximum work area size for each type of SQL operator.
				You must manually configure SGA component sizes. You set:
				
				• Shared pool size (SHARED_POOL_SIZE)
				• Buffer cache size (DB_CACHE_SIZE)
				• Large pool size (LARGE_POOL_SIZE)
				• Java pool size (JAVA_POOL_SIZE)
				• Streams pool size (STREAMS_POOL_SIZE)
				• PGA work area parameters such as SORT_AREA_SIZE, HASH_AREA_SIZE, and BITMAP_MERGE_AREA_SIZE
					您必须手动配置各个 SGA 组件的大小。
					
					您可以手动控制 PGA，设置每种类型的 SQL 操作的最大工作区大小。
						您必须手动配置 SGA 组件大小。您设置：
						
						• 共享池大小（SHARED_POOL_SIZE）
						• 缓冲区高速缓存大小（DB_CACHE_SIZE）
						• 大池大小（LARGE_POOL_SIZE）
						• Java 池大小（JAVA_POOL_SIZE）
						• 流池大小（STREAMS_POOL_SIZE）
						• PGA 工作区参数如
						  ○ SORT_AREA_SIZE、
						  ○ HASH_AREA_SIZE、
						  ○ BITMAP_MERGE_AREA_SIZE

See Also:

Oracle Database Administrator's Guide because automatic memory management is not available on all platforms

另见：

《*Oracle 数据库管理员指南*》，因为自动内存管理不是在所有平台上都可用

Resource Management and Task Scheduling

In a database with many active users, resource management is an important part of database administration. Sessions that consume excessive resources can prevent other sessions from doing their work. A related problem is how to schedule tasks so that they run at the best time. Oracle Database provides tools to help solve these problems.

资源管理与任务调度

在一个有许多活动用户的数据库中，资源管理是数据库管理的一个重要组成部分。占用过多资源的会话可能会阻止其他会话进行工作。一个相关的问题是如何调度这些任务，以使他们在最合理的时间运行。Oracle 数据库提供了帮助解决这些问题的工具。

Database Resource Manager

Oracle Database Resource Manager (the Resource Manager) is an infrastructure that provides granular control of database resources allocated to users, applications, and services. The Resource Manager solves many resource allocation problems that an operating system does not manage well, including:

数据库资源管理器

Oracle 数据库资源管理器 （资源管理器） 是一个为用户、应用程序、及服务提供数据库资源分配精确控制的基础结构。资源管理器解决了很多操作系统无法管理好的资源分配问题，包括：

- Excessive overhead
- Inefficient scheduling
- Inappropriate allocation of resources
- Inability to manage database-specific resources

The Resource Manager helps overcome these problems by giving the database more control over allocation of hardware resources and enabling you to prioritize work within the database. You can classify sessions into groups based on session attributes, and then allocate resources to these groups to optimize hardware utilization.
Resources are allocated to users according to a **resource plan** specified by the database administrator. The plan specifies how the resources are to be distributed among **resource consumer groups**, which are user sessions grouped by resource requirements. A **resource plan directive** associates a resource consumer group with a plan and specifies how resources are to be allocated to the group.
Figure 18-6 shows a simple resource plan for an organization that runs OLTP applications and reporting applications simultaneously during the daytime. The currently active plan, `DAYTIME`, allocates CPU resources among three resource consumer groups. Specifically, `OLTP` is allotted 75% of the CPU time, `REPORTS` is allotted 15%, and `OTHER_GROUPS` receives the remaining 10%.

Figure 18-6 Simple Resource Plan

Description of "Figure 18-6 Simple Resource Plan"

See Also:
Oracle Database Administrator's Guide for information about using the Resource Manager

Oracle Scheduler

- 开销过多
- 调度效率低
- 资源分配不合理
- 无法管理特定于数据库的资源

资源管理器通过给予数据库在硬件资源分配方面更多的控制，并使您能够排定数据库内各项工作的优先顺序，来帮助克服这些问题。你可以基于会话属性将会话分组，然后将资源分配到这些组，以优化硬件利用率。

根据由数据库管理员指定的**资源计划**，资源被分配给各个用户。此计划指定如何在多个**资源消费者组**之间分配资源，资源消费者组是按所需资源对多个用户会话的分组。**资源计划指令**将资源消费者组与一项计划相关联，并指定如何向组分配资源。

图 18-6 显示了某组织的一个简单的资源计划，该组织在白天期间同时运行 OLTP 应用程序和报表应用程序。当前的活动计划 `DAYTIME`，在三个资源消费者组之间分配 CPU 资源。具体而言，`OLTP` 分配了 75%的 CPU 时间、`REPORTS` 分配了 15%、而 `OTHER_GROUPS` 得到剩下的 10%。

图 18-6 简单的资源计划

Description of "Figure 18-6 Simple Resource Plan"

另见：
《*Oracle 数据库管理员指南*》有关使用资源管理器的信息

Oracle 调度器

Oracle Scheduler (the Scheduler) enables database administrators and application developers to control when and where various tasks take place in the database environment. The Scheduler provides complex enterprise scheduling functionality, which you can use to:

- Schedule job execution based on time or events
- Schedule job processing in a way that models your business requirements
- Manage and monitor jobs
- Execute and manage jobs in a clustered environment

Program objects (**programs**) contain metadata about the command that the Scheduler will run, including default values for any arguments. Schedule objects (**schedules**) contain information about run date and time and recurrence patterns. Job objects (**jobs**) associate a program with a schedule. To define what is executed and when, you assign relationships among programs, schedules, and jobs.

The Scheduler is implemented as a set of functions and procedures in the `DBMS_SCHEDULER` PL/SQL package. You create and manipulate Scheduler objects with this package or with Enterprise Manager. Because Scheduler objects are standard database objects, you can control access to them with system and object privileges.

Figure 18-7 shows the basic architecture of the Scheduler. The job table is a container for all the jobs, with one table per database. The **job coordinator** background process is automatically started and stopped as needed. Job slaves are awakened by the coordinator when a job must be run (see "Job Queue Processes (CJQ0 and Jnnn)"). The slaves gather metadata from the job table and run the job.

Figure 18-7 Scheduler Components

Oracle 调度器 （调度器） 使数据库管理员和应用程序开发人员能控制在数据库环境中的各项任务将在何时、何地被运行。调度器提供了复杂的企业调度功能，您可以使用它来：

- 基于时间或事件安排作业执行
- 以一种符合您的业务需求的方式来安排作业处理
- 管理和监控作业
- 在集群环境中执行和管理作业

程序对象 （**程序**） 包含调度器将要运行的命令的有关元数据，包括任何参数的默认值。计划对象 （**计划**） 包含有关运行的日期、时间、和重复模式的信息。作业对象 （**作业**） 将程序与计划相关联。通过指定程序、计划、和作业之间的关系，来定义在什么时候执行什么事情。

调度器是作为 DBMS_SCHEDULER PL/SQL 包中的一组函数和过程来实现的。可以使用此包或使用企业管理器来创建和操作各种计划器对象。因为计划器对象是标准的数据库对象，您可以通过系统和对象权限来控制对它们的访问。

图 18-7 显示了调度器的基本体系结构。作业表是所有作业的一个容器，每个数据库只有一个作业表。工作协调器后台进程根据需要自动启动和停止。当作业必须运行时，协调器会唤醒作业从属进程来执行作业（请参阅" 作业队列进程（CJQ0 和 Jnnn）"）。这些从属进程从作业表中收集元数据，并运行该作业。

图 18-7 调度器组件

Description of "Figure 18-7 Scheduler Components"

See Also:

Oracle Database Administrator's Guide to learn about the Scheduler

Performance Diagnostics and Tuning

As a DBA, you are responsible for the performance of your Oracle database. Typically, performance problems result from unacceptable **response time**, which is the time to complete a specified workload, or **throughput**, which is the amount of work that can be completed in a specified time. Common problems include:

- CPU bottlenecks
- Undersized memory structures
- I/O capacity issues
- Inefficient or high-load SQL statements
- Unexpected performance regression after tuning SQL statements
- Concurrency and contention issues
- Database configuration issues

The general goal of tuning is usually to improve response time, increase throughput, or both. A specific and measurable goal might be "Reduce the response time of the specified `SELECT` statement to under 5 seconds." Whether this goal is achievable depends on factors that may or may not be under the control of the DBA. In general, tuning is the effort to achieve specific, measurable, and achievable tuning goals by using database resources in the most efficient way possible.

The **Oracle performance method** is based on identifying and

Description of "Figure 18-7 Scheduler Components"

另见：

《*Oracle 数据库管理员指南*》了解调度器

性能诊断和调优

作为一个数据库管理员，您有责任对您的 Oracle 数据库的性能负责。通常，性能问题源自不可接受的**响应时间**（即完成指定的工作负荷的时间）或**吞吐量**（即在指定的时间内可以完成的工作量）。常见问题包括：

- CPU 瓶颈
- 内存结构大小不足
- I/O 容量问题
- 效率低下或高负载的 SQL 语句
- 优化 SQL 语句之后，意外的性能退化
- 并发和争用问题
- 数据库配置问题

总的优化目标通常是缩短响应时间、提高吞吐量、或二者兼备。一个具体的、 可测量的目标可能是："将指定的 SELECT 语句的响应时间减少到 5 秒以下"。这一目标是否可以实现取决于一些 DBA 可能或不能控制的因素。一般情况下，调整是通过尽可能以最有效的方式使用数据库资源，以实现具体、 可测量的、和可实现的优化目标而所作的努力。

Oracle 性能方法基于查明和消除数据库中的瓶颈，并开发有效的 SQL 语

eliminating bottlenecks in the database, and developing efficient SQL statements. Applying the Oracle performance method involves the following tasks:

- Performing pre-tuning preparations
- Tuning the database proactively on a regular basis
- Tuning the database reactively when users report performance problems
- Identifying, tuning, and optimizing high-load SQL statements

This section describes essential aspects of Oracle Database performance tuning, including the use of **advisors**. Oracle Database advisors provide specific advice on how to address key database management challenges, covering a wide range of areas including space, performance, and undo management.

See Also:

Oracle Database 2 Day + Performance Tuning Guide and *Oracle Database Performance Tuning Guide* provide to learn how to implement the Oracle performance method

Database Self-Monitoring

Self-monitoring take place as the database performs its regular operation, ensuring that the database is aware of problems as they arise. Oracle Database can send a **server-generated alert** to notify you of an impending problem.

Alerts are automatically generated when a problem occurs or when data does not match expected values for metrics such as physical reads per second or SQL response time. A **metric** is the rate of change in a cumulative statistic. Server-generated alerts can be based on user-specified threshold levels or because an event has occurred.

Server-generated alerts not only identify the problem, but sometimes recommend how the reported problem can be resolved. An example is an alert that the fast recovery area is running out of space with the recommendation that obsolete backups should be deleted or additional disk space added.

See Also:

Oracle Database Administrator's Guide

Automatic Workload Repository (AWR)

Automatic Workload Repository (AWR) is a repository of historical performance data that includes cumulative statistics for the system, sessions, individual SQL statements, segments, and services. These

句。应用 Oracle 性能方法涉及以下任务：

- 执行优化前准备
- 定期地、主动地调整数据库
- 当用户报告性能问题时，反应性地优化数据库，
- 确定、 调整、并优化高负载的 SQL 语句

本节介绍了包括各种**顾问工具**在内的 Oracle 数据库性能调优的各个重要方面。Oracle 数据库顾问就如何解决关键数据库管理难题提供了具体的建议，并涉及广泛的领域，包括空间、性能、撤消管理等。

另见：

《Oracle 数据库 2 日+ 性能调整指南》和 《Oracle 数据库性能优化指南》了解如何实现 Oracle 性能方法

数据库自我监控

在数据库执行其常规操作期间会进行自我监控，以确保数据库能意识到发生的问题。Oracle 数据库可以发出一个**服务器生成的警报**，以通知您即将发生的问题。

当问题发生时，或数据与预期的度量值（如每秒物理读或 SQL 的响应时间）不匹配时，将自动生成警报。**度量**是累积统计的变化率。服务器生成的警报可以基于用户指定的阈值级别，或由于发生了某个事件。

服务器生成的警报不只是确定了问题，而且有时也提供如何解决所报告问题的建议。例如，快速恢复区快要空间不足时发出的警报也包括应删除过时备份或添加额外磁盘空间的建议

另见：

《*Oracle 数据库管理员指南*》

自动工作负载存储库 （AWR）

自动工作负载存储库 （AWR) 是一个历史性能数据的存储库，其中包括针对系统、 会话、 单个 SQL 语句、 段、和服务的累积统计信息。这些统计信息是性能优化的基础。通过自动化收集用于问题检测和调整的数据库统计

statistics are the foundation of performance tuning. By automating the gathering of database statistics for problem detection and tuning, AWR serves as the foundation for database self-management.

As shown in Figure 18-8, the database stores recent AWR statistics in the SGA. By default, the MMON process gathers statistics every hour and creates an **AWR snapshot** (see "Manageability Monitor Processes (MMON and MMNL)"). A snapshot is a set of performance statistics captured at a specific time. The database writes snapshots to the `SYSAUX` tablespace. AWR manages snapshot space, purging older snapshots according to a configurable **snapshot retention policy**.

Figure 18-8 Automatic Workload Repository (AWR)

Description of "Figure 18-8 Automatic Workload Repository (AWR)"

An **AWR baseline** is collection of statistic rates usually taken over a period when the system is performing well at peak load. You can specify a pair or range of AWR snapshots as a baseline. By using an AWR report to compare statistics captured during a period of bad performance to a baseline, you can diagnose problems.

An automated maintenance infrastructure known as **AutoTask** illustrates how Oracle Database uses AWR for self-management. By analyzing AWR data, AutoTask can determine the need for maintenance tasks and

信息，AWR 可作为数据库自我管理的基础。

如图 18-8，数据库将最近的 AWR 统计信息存储在 SGA 中。默认情况下，MMON 进程每个小时收集一次统计信息，并创建一个 **AWR 快照** （请参阅"可管理性监控进程 （MMON 和 MMNL）"）。快照是一组在特定时间捕获的性能统计信息。数据库将快照写入 SYSAUX 表空间。AWR 会管理快照空间，根据一个可配置的**快照保留策略**来清除较旧的快照。

图 18-8 自动工作负载存储库 （AWR）

Description of "Figure 18-8 Automatic Workload Repository (AWR)"

AWR 基线是一组统计率的集合，通常当系统在某个运行良好的峰值负载期间时取得。您可以指定一对或某个范围的 AWR 快照作为基准。通过使用 AWR 报告来比较性能低下期间捕获的统计信息与基线之间的差异，可以诊断问题。

一个名为 **AutoTask** 的自动维护基础设施说明了 Oracle 数据库如何使用 AWR 来进行自我管理。通过分析 AWR， AutoTask 可以确定需要的维护任务，并在 Oracle 调度器维护窗口中安排运行他们。任务示例包括为优化器

schedule them to run in Oracle Scheduler **maintenance windows**. Examples of tasks include gathering statistics for the optimizer and running the Automatic Segment Advisor.

See Also:

- "The SYSAUX Tablespace"
- *Oracle Database Performance Tuning Guide* to learn about AWR
- *Oracle Database Administrator's Guide* and *Oracle Database 2 Day DBA* to learn how to manage automatic maintenance tasks

Automatic Database Diagnostic Monitor (ADDM)

Automatic Database Diagnostic Monitor (ADDM) is a self-diagnostic advisor built into Oracle Database. Using statistics captured in AWR, ADDM automatically and proactively diagnoses database performance and determines how identified problems can be resolved. You can also run ADDM manually.

ADDM takes a holistic approach to system performance, using time as a common currency between components. ADDM identifies areas of Oracle Database consuming the most time. For example, the database may be spending an excessive amount of time waiting for free database buffers. ADDM drills down to identify the root cause of problems, rather than just the symptoms, and reports the effect of the problem on Oracle Database overall. Minimal overhead occurs during the diagnostic process.

In many cases, ADDM recommends solutions and quantifies expected performance benefits. For example, ADDM may recommend changes to hardware, database configuration, database schema, or applications. If a recommendation is made, then ADDM reports the time benefit. The use of time as a measure enables comparisons of problems or recommendations.

Besides reporting potential performance issues, ADDM documents areas of the database that are not problems. Subcomponents such as I/O and memory that are not significantly impacting database performance are pruned from the classification tree at an early stage. ADDM lists these subcomponents so that you can quickly see that there is little benefit to performing actions in those areas.

See Also:

Oracle Database 2 Day + Performance Tuning Guide and *Oracle Database Performance Tuning Guide*

Active Session History (ASH)

Active Session History (ASH) samples active database sessions each

收集统计信息，和运行自动段顾问工具。

另见：

- SYSAUX 表空间
- 《*Oracle 数据库性能优化指南*》了解 AWR
- 《*Oracle 数据库管理员指南*》和 《*Oracle 数据库 2 日 DBA*》了解如何管理自动维护任务

自动数据库诊断监视器 （ADDM）

自动数据库诊断监视器 （ADDM） 是 Oracle 数据库中内置的自我诊断顾问。通过使用 AWR 中捕获的统计信息，ADDM 自动、 主动地诊断数据库的性能并确定如何解决已发现的问题。您也可以手动运行 ADDM。

ADDM 对系统性能进行整体分析，将时间作为组件之间的共同要素。ADDM 标识 Oracle 数据库占用时间最长的区域。例如，数据库可能花费过多的时间等待可用数据库缓冲区。ADDM 向下钻取以发现问题的根本原因，而不是浮于表面现象，并报告此问题在总体上对 Oracle 数据库的影响。诊断过程中会出现很小的开销。

在许多情况下，ADDM 会提出推荐的解决方案，并量化预期的性能优势。例如，ADDM 可能会建议更改硬件、 数据库配置、 数据库模式、或应用程序。如果采纳了某项建议，则 ADDM 将报告其时间收益。利用时间作为度量，使你能对问题或建议进行比较。

除了报告潜在的性能问题，ADDM 会记录数据库中不会造成问题的方面。若 I/O 和内存等子组件未对数据库性能造成显著影响，则它们在早期阶段就会从分类树中被剪除。ADDM 列出这些子组件，以便您可以快速发现对这些方面执行操作不会有什么效果。

另见：

《*Oracle 数据库 2 日+ 性能调整指南*》和《*Oracle 数据库性能优化指南*》

活动会话历史 （ASH）

活动会话历史 (ASH) 每秒对活动的数据库会话进行采样，并将样本数据写

second, writing the data to memory and persistent storage. ASH is an integral part of the database self-management framework and is useful for diagnosing performance problems.

入内存和持久性存储。ASH 是数据库自我管理框架的一个组成部分，对于诊断性能问题很有用。

Unlike instance-level statistics gathered by AWR, ASH statistics are gathered at the session level. An **active session** is a session that is using CPU and is not waiting for an event in the idle wait class.

与 AWR 所收集的实例级统计信息不同，ASH 统计信息是在会话级别收集的。**活动会话**是正在使用 CPU，并且不在等待某个空闲等待类事件的会话。

You can use Enterprise Manager or SQL scripts to generate **ASH reports** that gather session statistics gathered over a specified duration. You can use ASH reports for:

- Analysis of short-lived performance problems not identified by ADDM
- Scoped or targeted performance analysis by various dimensions or their combinations, such as time, session, module, action, or SQL ID

可以使用企业管理器或 SQL 脚本来生成 ASH 报告，以收集该会话在指定的持续时间段内的统计信息。您可以使用 ASH 报告：

- 分析未能被 ADDM 发现的短暂性能问题
- 对特定范围和特定目标，按各种维度或其组合进行性能分析，如时间、 会话、 模块、动作、或 SQL ID 等

For example, a user notifies you that the database was slow between 10:00 p.m. and 10:02 p.m. However, the 2-minute performance degradation represents a small portion of the AWR snapshot interval from 10:00 p.m. and 11:00 p.m. and does not appear in ADDM findings. ASH reports can help identify the source of the transient problem.

例如，用户通知您数据库在下午 10: 00 至下午 10: 02 很慢，然而，2 分钟的性能下降只占 10: 00 至 11: 00 之间 AWR 快照时间间隔的一小部分，并不会出现 ADDM 查找结果中。ASH 报告可以帮助识别这种瞬态问题的根源。

See Also:

Oracle Database 2 Day + Performance Tuning Guide and *Oracle Database Performance Tuning Guide*

另见：

《*Oracle 数据库 2 日+ 性能调整指南*》和 《*Oracle 数据库性能优化指南*》

Application and SQL Tuning

应用程序与 SQL 优化

Oracle Database completely automates the SQL tuning process. ADDM identifies SQL statements consuming unusually high system resources and therefore causing performance problems. In addition, AWR automatically captures the top SQL statements in terms of CPU and shared memory consumption. The identification of high-load SQL statements happens automatically and requires no intervention.

Oracle 数据库完全自动化 SQL 调整过程。ADDM 能标识消耗异常高系统资源从而导致性能问题的 SQL 语句。此外，AWR 自动捕获在 CPU 和共享内存消耗方面最高的 SQL 语句。高负荷 SQL 语句会被自动发现，并且需要不干预。

SQL Tuning Advisor

SQL 调优顾问

Automatic SQL tuning is exposed through **SQL Tuning Advisor**. SQL Tuning Advisor runs automatically during system maintenance windows as a maintenance task. During each automatic run, the advisor selects high-load SQL queries in the database and generates recommendations for tuning these queries.

自动 SQL 调优是通过 SQL 优化顾问来使用的。**SQL 优化顾问**在系统维护窗口期间作为一个维护任务自动运行。每个自动运行期间，顾问在数据库中选择高负载的 SQL 查询，并生成优化这些查询的建议。

SQL Tuning Advisor recommendations fall into the following categories:

SQL 优化顾问建议分为以下几类：

- Statistics analysis
- SQL profiling
- Access path analysis
- SQL structure analysis

A **SQL profile** contains additional statistics specific to a SQL statement and enables the optimizer to generate a better execution plan. Essentially, a SQL profile is a method for analyzing a query. Both access path and SQL structure analysis are useful for tuning an application under development or a homegrown production application.

A principal benefit of SQL Tuning Advisor is that solutions come from the optimizer rather than external tools (see "Overview of the Optimizer"). Thus, tuning is performed by the database component that is responsible for the execution plans and SQL performance. The tuning process can consider past execution statistics of a SQL statement and customizes the optimizer settings for this statement.

See Also:

Oracle Database 2 Day + Performance Tuning Guide and *Oracle Database Performance Tuning Guide*

SQL Access Advisor

SQL Access Advisor offers advice on how to optimize data access paths. Specifically, it recommends how database performance can be improved through partitioning, materialized views, indexes, and materialized view logs.

Schema objects such as partitions and indexes are essential for optimizing complex, data-intensive queries. However, creation and maintenance of these objects can be time-consuming, and space requirements can be significant. SQL Access Advisor helps meet performance goals by recommending data structures for a specified workload.

The SQL Access Advisor can be run from Enterprise Manager using the SQL Access Advisor Wizard or by invoking the `DBMS_ADVISOR` package. The `DBMS_ADVISOR` package consists of a collection of analysis and advisory functions and procedures callable from any PL/SQL program.

See Also:

Oracle Database 2 Day + Performance Tuning Guide and *Oracle Database Performance Tuning Guide*

- 统计分析
- SQL 概要
- 访问路径分析
- SQL 结构分析

SQL 概要文件包含特定于 SQL 语句的更多统计信息，并使优化器生成一个更好的执行计划。实质上，SQL 概要文件是一种分析查询方法。访问路径和 SQL 结构分析对于优化正处于开发中的应用程序或自主开发的应用程序很有用。

SQL 优化顾问的主要好处是，其解决方案来自优化器而不是外部工具 （请参阅"优化程序概述"）。因此，优化是由负责执行计划和 SQL 性能的数据库组件执行的。优化过程可以考虑 SQL 语句过去的执行统计信息，并自定义此语句的优化设置。

另见：

《*Oracle 数据库 2 日+ 性能调整指南*》和 《*Oracle 数据库性能优化指南*》

SQL 访问顾问

SQL 访问顾问对如何优化数据访问路径提供建议。具体来说，它建议如何通过分区、 物化视图、索引、和物化视图日志改进数据库性能。

分区和索引等模式对象对于优化复杂的数据密集型查询非常重要。但是，这些对象的创建和维护会非常耗时，且空间要求可能很大。SQL 访问顾问可以通过为指定的工作负荷建议数据结构，来帮助实现性能目标。

SQL 访问顾问可以从企业管理器中使用 SQL 访问顾问向导，或通过调用 DBMS_ADVISOR 包来运行。DBMS_ADVISOR 包包含了一组可从任何 PL/SQL 程序调用的分析和建议函数和过程。

另见：

《*Oracle 数据库 2 日+ 性能调整指南*》和 《*Oracle 数据库性能优化指南*》

19 Concepts for Database Developers

The Oracle Database developer creates and maintains a database application. This section presents a brief overview of what a database developer does and the development tools available.

This section contains the following topics:

- Duties of Database Developers
- Tools for Database Developers
  - SQL Developer
  - Oracle Application Express
  - Oracle JDeveloper
  - Oracle JPublisher
  - Oracle Developer Tools for Visual Studio .NET
- Topics for Database Developers
  - Principles of Application Design and Tuning
  - Client-Side Database Programming
  - Globalization Support
  - Unstructured Data

Duties of Database Developers

An Oracle developer is responsible for creating or maintaining the database components of an application that uses the Oracle technology stack. Oracle developers either develop new applications or convert existing applications to run in an Oracle Database environment. For this reason, developers work closely with the database administrators, sharing knowledge and information.

Oracle database developers can expect to be involved in the following tasks:

- Implementing the data model required by the application
- Creating schema objects and implementing rules for data integrity
- Choosing a programming environment for a new development project
- Writing server-side PL/SQL or Java subprograms and client-side procedural code that use SQL statements

第 19 章 数据库开发人员相关概念

Oracle 数据库开发人员创建并维护数据库应用程序。本章简单概述了数据库开发人员所做的事情及其可用的开发工具。

本部分包含以下主题：

- 数据库开发人员的职责
- 数据库开发人员的工具
  - SQL Developer
  - Oracle Application Express
  - Oracle JDeveloper
  - Oracle JPublisher
  - Oracle 开发工具 for Visual Studio.NET
- 数据库开发人员的主题
  - 应用程序设计和优化的原则
  - 客户端数据库编程
  - 全球化支持
  - 非结构化数据

数据库开发人员的职责

Oracle 开发人员负责创建和维护使用 Oracle 技术堆栈的应用程序的数据库组件。Oracle 开发人员要么在 Oracle 数据库环境中开发新的应用程序，或者转换一个现有的应用程序以便在其中运行。因此，开发人员会与数据库管理员紧密合作，并分享知识和信息。

Oracle 数据库开发人员可能涉及以下任务：

- 实现应用程序所需的数据模型
- 创建模式对象并实施数据完整性规则
- 为一个新的开发项目选择编程环境
- 编写服务器端 PL/SQL 或 Java 子程序，和客户端使用的 SQL 语句的程序代码
- 使用所选择的开发工具创建应用程序接口

- Creating the application interface with the chosen development tool
- Establishing a Globalization Support environment for developing globalized applications
- Instantiating applications in different databases for development, testing, education, and deployment in a production environment

The preceding tasks, and many others, are described in *Oracle Database 2 Day Developer's Guide* and *Oracle Database Advanced Application Developer's Guide*.

See Also:

"Introduction to Server-Side Programming"

Tools for Database Developers

Oracle provides several tools for use in developing database applications. This section describes some commonly used development tools.

SQL Developer

SQL Developer is a graphical version of SQL*Plus, written in Java, that supports development in SQL and PL/SQL. You can connect to any Oracle database schema using standard database authentication. SQL Developer enables you to:

- Browse, create, edit, and delete schema objects
- Execute SQL statements
- Edit and debug PL/SQL program units
- Manipulate and export data
- Create and display reports

SQL Developer is available in the default Oracle Database installation and by free download.

See Also:

Oracle Database 2 Day Developer's Guide and *Oracle Database SQL Developer User's Guide* to learn how to use SQL Developer

Oracle Application Express

Oracle Application Express (APEX) is a Web application development tool for Oracle Database. The tool uses built-in features such as user interface themes, navigational controls, form handlers, and flexible reports to accelerate application development.

Oracle Application Express installs with the database and consists of data in tables and PL/SQL code. When you run an application, your browser sends a URL request that is translated into an Oracle Application Express

- 为开发全球化应用程序建立全球化支持环境
- 为开发、 测试、 培训等，在不同的数据库中实例化应用程序，并在生产环境中部署

在《*Oracle 数据库 2 日开发人员指南*》和《*Oracle 数据库高级应用程序开发人员指南*》中描述了上述任务和一些其他任务。

另见：

"服务器端编程简介"

数据库开发人员的工具

Oracle 提供了用于开发数据库应用程序的几个工具。本节介绍了一些常用的开发工具。

SQL Developer

SQL Developer 是 SQL*Plus 的一个图形化的版本，是用 Java 写的，支持 SQL 和 PL/SQL 开发。您可以使用标准的数据库身份验证连接到任何 Oracle 数据库模式。SQL Developer 使您能够：

- 浏览、 创建、 编辑、和删除模式对象
- 运行 SQL 语句
- 编辑和调试 PL/SQL 程序单元
- 操作和导出数据
- 创建和显示报表

SQL Developer 在默认的 Oracle 数据库安装中就可用，并提供免费下载。

另见：

《*Oracle 数据库 2 日开发人员指南*》 和《*Oracle 数据库 SQL Developer 用户指南*》了解如何使用 SQL Developer

Oracle 应用程序快速开发工具

Oracle 应用程序快速开发工具 (APEX) 是针对 Oracle 数据库的 Web 应用程序开发工具。该工具使用一些内置的功能，如用户界面主题、 导航控件、 表单处理程序和灵活的报表，来加快应用程序的开发。

Oracle 应用程序快速开发工具与数据库安装在一起，并由一些数据表和 PL/SQL 代码组成。当你运行一个应用程序时，您的浏览器发送的 URL 请求被翻译成一个 Oracle APEX 的 PL/SQL 调用。当数据库处理了 PL/SQL 后，

PL/SQL call. After the database processes the PL/SQL, the results are relayed back to the browser as HTML. This cycle happens each time you request or submit a page.

You can use Oracle Application Express with the embedded PL/SQL gateway. The gateway runs in the XML DB HTTP server in the database and provides the necessary infrastructure to create dynamic applications. As shown in Figure 19-1, the embedded PL/SQL gateway simplifies the application architecture by eliminating the middle tier.

Figure 19-1 Application Express with Embedded PL/SQL Gateway

Description of "Figure 19-1 Application Express with Embedded PL/SQL Gateway"

See Also:

Oracle Database 2 Day + Application Express Developer's Guide to learn how to use APEX

Oracle JDeveloper

Oracle JDeveloper is an integrated development environment (IDE) for building service-oriented applications using the latest industry standards for Java, XML, Web services, and SQL. Oracle JDeveloper supports the complete software development life cycle, with integrated features for modeling, coding, debugging, testing, profiling, tuning, and deploying applications.

JDeveloper uses windows for various application development tools. For example, when creating a Java application, you can use tools such as the Java Visual Editor and Component Palette. In addition to these tools, JDeveloper provides a range of navigators to help you organize and view the contents of your projects.

See Also:

- *Oracle Database 2 Day + Java Developer's Guide* to learn how to use JDeveloper
- You can download JDeveloper from the following URL: http://www.oracle.com/technology/software/products/jdev

结果将以 html 格式返回浏览器。每发一次请求或提交一个页面时，都会像这样反复发生。

您可以将 Oracle APEX 与嵌入式 PL/SQL 网关一起使用。网关在数据库中的 XML DB 的 HTTP 服务器上运行，并提供必要的基础结构，以创建动态的应用程序。如图 19-1 中的嵌入式 PL/SQL 网关通过消除中间层，从而简化了应用程序体系结构。

图 19-1 中应用程序快速开发工具与嵌入式 PL/SQL 网关

Description of "Figure 19-1 Application Express with Embedded PL/SQL Gateway"

另见：

《*Oracle 数据库 2 日+ 应用程序快速开发人员指南*》了解如何使用 APEX

Oracle JDeveloper

Oracle JDeveloper 是一个集成开发环境 (IDE)，使用 Java、XML、 Web 服务、和 SQL 等最新行业标准，生成面向服务的应用程序。Oracle JDeveloper 支持完整的软件开发生命周期，具备建模、 编码、 调试、 测试、 分析、 优化，和部署应用程序等集成的功能。

JDeveloper 为各种应用程序开发工具大量使用视窗。例如，在创建一个 Java 应用程序时，您可以使用如 Java 可视编辑器和组件面板之类的工具。除了这些工具之外， JDeveloper 还提供了一系列的导航器，以帮助您组织和查看您的项目中的内容。

另见：

- 《*Oracle 数据库 2 日+ Java 开发人员指南*》了解如何使用 JDeveloper
- 您可以从下面的 URL 下载 JDeveloper: http://www.oracle.com/technology/software/products/jdev

Oracle JPublisher

Java Publisher (JPublisher) is a simple and convenient tool to create Java programs that access database tables. Java code stubs generated by JDeveloper present object-relational structures in the database as Java classes. These classes can represent the following user-defined database entities in a Java program:

- SQL object types
- Object reference types
- SQL collection types
- PL/SQL packages

You can specify and customize the mapping of these entities to Java classes in a **strongly typed** paradigm, so that a specific Java type is associated with a specific user-defined SQL type. JPublisher can also generate classes for PL/SQL packages. These classes have wrapper methods to call the stored procedure in the package.

See Also:

Oracle Database JPublisher User's Guide

Oracle Developer Tools for Visual Studio .NET

Oracle Developer Tools for Visual Studio .NET is a set of application tools integrated with the Visual Studio .NET environment. These tools provide GUI access to Oracle functionality, enable the user to perform a wide range of application development tasks, and improve development productivity and ease of use.

Oracle Developer Tools support the programming and implementation of .NET stored procedures using Visual Basic, C#, and other .NET languages. These procedures are written in a .NET language and contain SQL or PL/SQL statements.

See Also:

Oracle Database 2 Day + .NET Developer's Guide for Microsoft Windows

Topics for Database Developers

Chapter 17 describes topics important for both developers and administrators. This section covers topics that are most essential to database developers and that have not been discussed elsewhere in the manual.

ucts/jdev

Oracle JPublisher

Java 发布服务器 (JPublisher) 是一个简单、方便的工具，用于创建访问数据库表的 Java 程序。由 JDeveloper 生成的 Java 桩代码代表数据库中以 Java 类存在的对象-关系结构。这些类可以表示以下 Java 程序中的用户定义的数据库实体：

- SQL 对象类型
- 对象引用类型
- SQL 集合类型
- PL/SQL 包

你可以按**强类型**范式，指定或自定义这些实体与 Java 类之间的映射，以便将特定的 Java 类型与特定的用户定义 SQL 类型相关联。JPublisher 还可以为 PL/SQL 包生成 Java 类。这些类具有包装器方法，以调用包中的存储过程。

另见：

《*Oracle 数据库 JPublisher 用户指南*》

Oracle 开发工具 for Visual Studio.NET

Oracle 开发工具 for Visual Studio .NET 是一组与 Visual Studio.NET 集成在一起的应用程序开发工具。这些工具提供图形用户界面来访问 Oracle 功能，使用户能够执行广泛的应用程序开发任务，提高开发效率和易用性。

Oracle 开发工具支持使用 Visual Basic、 C#、和其他.NET 语言来编写和实现.NET 存储过程。这些过程是用某种.NET 语言编写的，并且其中包含 SQL 或 PL/SQL 语句。

另见：

《*Oracle 数据库 2 日+.NET 开发人员指南 for Microsoft Windows*》

数据库开发人员的主题

第 17 章介绍了与开发人员和管理员都相关的重要主题。本节介绍之前从未讨论过的、对数据库开发人员最重要的主题。

This section contains the following topics:

- Principles of Application Design and Tuning
- Client-Side Database Programming
- Globalization Support
- Unstructured Data

Principles of Application Design and Tuning

Oracle developers must design, create, and tune database applications so that they achieve security and performance goals. The following principles of application design and tuning are useful guidelines:

- Understand how Oracle Database works

  As a developer, you want to develop applications in the least amount of time against an Oracle database, which requires exploiting the database architecture and features. For example, not understanding Oracle Database concurrency controls and multiversioning read consistency may make an application corrupt the integrity of the data, run slowly, and decrease scalability (see "Introduction to Data Concurrency and Consistency").
- Use bind variables

  When a query uses bind variables, the database can compile it once and store the query plan in the shared pool. If the same statement is executed again, then the database can perform a soft parse and reuse the plan. In contrast, a hard parse takes longer and uses more resources (see "SQL Parsing"). Using bind variables to allow soft parsing is very efficient and is the way the database intends developers to work.
- Implement integrity constraints in the server rather than in the client
  Using primary and foreign keys enables data to be reused in multiple applications. Coding the rules in a client means that other clients do not have access to these rules when running against the databases (see "Advantages of Integrity Constraints").
- Build a test environment with representative data and session activity
  A test environment that simulates your live production environment provides multiple benefits. For example, you can benchmark the

本节包含以下主题：

- 应用程序设计和优化的原则
- 客户端数据库编程
- 全球化支持
- 非结构化数据

应用程序设计和优化的原则

Oracle 开发人员必须设计、 创建、并优化数据库应用程序，使其实现安全性和性能目标。以下的应用程序设计和优化原则是有用的指导方针：

- 了解 Oracle 数据库的工作原理

  作为一名开发人员，您希望在最少的时间内，完成对跑在某个 Oracle 数据库上的应用程序的开发工作，而这需要充分利用数据库的体系结构和功能特性。例如，假如不了解 Oracle 数据库的并发性控制和多版本读取一致性，就可能造成应用程序破坏数据完整性、 运行速度很慢、和降低可扩展性 （请参见"数据并发性和一致性的简介"）。

- 使用绑定变量

  当查询使用了绑定变量时，数据库可以仅对其编译一次，并将其查询计划存储在共享池中。当再次执行同一语句时，数据库可以执行一次软解析，并重新使用该计划。相比之下，硬解析需要花更长时间，并使用更多的资源 （请参见"SQL 解析"）。使用绑定变量以允许软解析非常有效，也是 Oracle 数据库的推荐做法。

- 在服务器端（而不是在客户端）实现完整性约束

  使用主键和外键使得数据可以在多个应用程序中被重用。将业务规则编写在客户端，意味着跑在该数据库上的其他客户端并不能访问这些规则（请参阅"完整性约束的优势"）。

- 使用有代表性的数据和会话活动，来建立测试环境

  一个模拟您的生产环境的测试环境能提供很多好处。例如，你可以对该应用程序做基准测试，以确保它具有良好的可扩展性和性能。此

application to ensure that it scales and performs well. Also, you can use a test environment to measure the performance impact of changes to the database, and ensure that upgrades and patches work correctly.

- Design the data model with the goal of good performance

  Typically, attempts to use generic data models result in poor performance. A well-designed data model answer the most common queries as efficiently as possible. For example, the data model should use the type of indexes that provide the best performance. Tuning after deployment is undesirable because changes to logic and physical structures may be difficult or impossible.
- Define clear performance goals and keep historical records of metrics
  An important facet of development is determining exactly how the application is expected to perform and scale. For example, you should use metrics that include expected user load, transactions per second, acceptable response times, and so on. Good practice dictates that you maintain historical records of performance metrics. In this way, you can monitor performance proactively and reactively (see "Performance Diagnostics and Tuning").
- Instrument the application code

  Good development practice involves adding debugging code to your application. The ability to generate trace files is useful for debugging and diagnosing performance problems.

See Also:

Oracle Database 2 Day Developer's Guide for considerations when designing and deploying database applications

Client-Side Database Programming

As explained in Chapter 8, "Server-Side Programming: PL/SQL and Java", two basic techniques enable procedural database applications to use SQL: server-side programming with PL/SQL and Java, and client-side programming with precompilers and APIs. This section provides a brief overview of client-side database programming.

See Also:

Oracle Database Advanced Application Developer's Guide to learn how to

外，您可以使用测试环境测量数据库变更对性能的影响，并确保升级和补丁程序能正常工作。

- 设计具有良好性能目标的数据模型

  通常，尝试使用通用的数据模型会导致较差的性能。设计完善的数据模型能尽可能高效响应最常见的查询。例如，数据模型应该使用能提供最佳性能的索引类型。在部署后再来优化是很不可取的，因为逻辑和物理结构的更改可能会很困难或根本不可能。

- 定义明确的性能目标，并保存历史度量记录

  开发的一个重要方面是，确定应用程序的预期性能和可扩展性要达到什么水准。例如，您应该使用的度量包括预期的用户负载、 每秒事务量、可接受的响应时间等。好的实践证明，您应该维护性能指标的历史记录。这样，你可以主动地或被动地监视性能（请参阅"性能诊断与优化"）。

- 在应用程序代码中加入调试、跟踪功能

  良好的开发习惯包括将调试代码添加到您的应用程序中。生成跟踪文件的能力对调试和诊断性能问题很有用。

另见：

《*Oracle 数据库 2 日开发人员指南*》 关于设计和部署数据库应用程序时的注意事项

客户端数据库编程

如第 8 章，"服务器端编程： PL/SQL 和 Java"中所述，有两种基本技术使你能在过程化数据库应用程序中使用 SQL：在服务器端使用 PL/SQL 和 Java 编程，或在客户端使用预编译器和 API 编程。本节提供了客户端数据库编程的简要概述。

另见：

《*Oracle 数据库高级应用程序开发者指南*》了解如何选择编程环境

choose a programming environment

Embedded SQL

Historically, client/server programs have used **embedded SQL** to interact with the database. This section explains options for using embedded SQL.

Oracle Precompilers

Client/server programs are typically written using **precompilers**, which are programming tools that enable you to embed SQL statements in high-level programs. For example, the Oracle Pro*C/C++ precompiler enables you to embed SQL statements in a C or C++ source file. Oracle precompilers are also available for COBOL and FORTRAN.

A precompiler provides several benefits, including the following:

- Increases productivity because you typically write less code than equivalent OCI applications
- Enables you to create highly customized applications
- Allows close monitoring of resource use, SQL statement execution, and various run-time indicators
- Saves time because the precompiler, not you, translates each embedded SQL statement into calls to the Oracle Database run-time library
- Uses the Object Type Translator to map Oracle Database object types and collections into C data types to be used in the Pro*C/C++ application
- Provides compile time type checking of object types and collections and automatic type conversion from database types to C data types

The client application containing the SQL statements is the **host program**. This program is written is the **host language**. In the host program, you can mix complete SQL statements with complete C statements and use C variables or structures in SQL statements. When embedding SQL statements you must begin them with the keywords `EXEC SQL` and end them with a semicolon. Pro*C/C++ translates `EXEC SQL` statements into calls to the run-time library SQLLIB.

Many embedded SQL statements differ from their interactive counterparts only through the addition of a new clause or the use of program variables. The following example compares interactive and embedded `ROLLBACK` statements:

```
ROLLBACK;            -- interactive
EXEC SQL ROLLBACK;   -- embedded
```

The statements have the same effect, but you would use the first in an

嵌入式 SQL

由于历史原因，客户端/服务器程序使用**嵌入式 SQL** 与数据库进行交互。本节介绍了使用嵌入式 SQL 的选项。

Oracle 预编译器

客户端/服务器程序通常是使用**预编译器**，使用这种编程工具，使您可以将 SQL 语句嵌入在高级语言程序中。例如 Oracle Pro*C/C++ 的预编译器，使你可以在 C 或 C++ 源文件中嵌入 SQL 语句。此外，Oracle 预编译器也可用于 COBOL 和 FORTRAN。

预编译器提供了以下几个好处：

- 提高生产效率，因为相比等效的 OCI 应用程序，您通常只需要写更少的代码
- 使您能够创建高度自定义的应用程序
- 允许对资源的使用、SQL 语句的执行、和各项运行时指标进行密切监控
- 因为预编译器替你将每个嵌入式 SQL 语句转换为对 Oracle 数据库运行时库的调用，节省了时间
- 使用对象类型转换器将 Oracle 数据库对象类型和集合映射到将在 Pro*C/C++应用程序中用到的数据类型
- 提供了对象类型和集合的编译时类型检查，和从数据库类型到 C 数据类型的自动类型转换

包含 SQL 语句的客户端应用程序是**宿主程序**。此程序是用**宿主语言**写的。在宿主程序中，你可以混合使用完整的 C 语句和完整的 SQL 语句，或在 SQL 语句中使用 C 变量或结构。嵌入 SQL 语句时，你必须以 `EXEC SQL` 关键字开始，并以分号结束。Pro*C/C++将 `EXEC SQL` 语句转换为对运行时库 SQLLIB 的调用。

很多嵌入式 SQL 语句与其交互形式的不同，仅在于增加了新的子句，或使用了程序变量。下面的示例比较了交互式和嵌入式的 `ROLLBACK` 语句：

```
ROLLBACK;            -- interactive
EXEC SQL ROLLBACK;   -- embedded
```

这两个语句有相同的效果，但您将在交互式 SQL 环境（例如 SQL

interactive SQL environment (such as SQL Developer), and the second in a Pro*C/C++ program.

A precompiler accepts the host program as input, translates the embedded SQL statements into standard database run-time library calls, and generates a source program that you can compile, link, and run in the usual way. Figure 19-2 illustrates typical steps of developing programs that use precompilers.

Figure 19-2 Program Development with Precompilers

Description of "Figure 19-2 Program Development with Precompilers"

See Also:

- *Pro*C/C++ Programmer's Guide* for a complete description of the Pro*C/C++ precompiler
- *Pro*FORTRAN Supplement to the Oracle Precompilers Guide*

SQLJ

SQLJ is an ANSI SQL-1999 standard for embedding SQL statements in Java source code. SQLJ provides a simpler alternative to the Java

Developer）中使用第一个，而在某个 Pro*C/C++ 程序中使用第二个。

预编译器接受宿主程序作为输入，将嵌入式的 SQL 语句转换为标准的数据库运行时库调用，并生成源代码程序，然后您可以对其编译、 链接，并以通常的方式运行。图 19-2 说明了使用预编译器开发程序的典型步骤。

图 19-2 使用预编译器的程序开发

Description of "Figure 19-2 Program Development with Precompilers"

另见：

- 《*Pro*C/C++程序员指南*》关于 Pro*C/C++预编译器的完整说明
- 《*Pro*FORTRAN 对 Oracle 预编译器指南的增补*》

SQLJ

SQLJ 是一个针对在 Java 源代码中嵌入 SQL 语句的 ANSI SQL-1999 标准。SQLJ 提供了一个对在 Java 客户端使用 Java 数据库连接 (JDBC) API 进

Database Connectivity (JDBC) API for client-side SQL data access from Java.

The SQLJ interface is the Java equivalent of the Pro* interfaces. You insert SQL statements in your Java source code. Afterward, you submit the Java source files to the SQLJ translator, which translates the embedded SQL to pure JDBC-based Java code.

See Also:

"SQLJ"

Client-Side APIs

Most developers today use an API to embed SQL in their database applications. For example, two popular APIs for enabling programs to communicate with Oracle Database are Open Database Connectivity (ODBC) and JDBC. The Oracle Call Interface (OCI) and Oracle C++ Call Interface (OCCI) are two other common APIs for client-side programming.

OCI and OCCI

As an alternative to precompilers, Oracle provides the OCI and OCCI APIs. OCI lets you manipulate data and schemas in a database using a host programming language such as C. OCCI is an object-oriented interface suitable for use with C++. Both APIs enable developers to use native subprogram invocations to access Oracle Database and control SQL execution.

In some cases, OCI provides better performance or more features than higher-level interfaces. OCI and OCCI provide many features, including the following:

- Support for all SQL DDL, DML, query, and transaction control facilities available through Oracle Database
- Instant client, a way to deploy applications when disk space is an issue
- Thread management, connection pooling, globalization functions, and direct path loading of data from a C application

OCI and OCCI provide a library of standard database access and retrieval functions in the form of a dynamic run-time library (OCILIB). This library can be linked in an application at run time. Thus, you can compile and link an OCI or OCCI program in the same way as a nondatabase application, avoiding a separate preprocessing or precompilation step. Figure 19-3 illustrates the development process.

Figure 19-3 Development Process Using OCI or OCCI

行 SQL 数据访问的简单替代方案。

SQLJ 接口是 Pro*接口的 Java 等效接口。您在 Java 源代码中插入 SQL 语句。之后，您将 Java 源代码文件提交给 SQLJ 翻译器，它将嵌入式 SQL 转换为基于 JDBC 的纯 Java 代码。

另见：

"SQLJ"

客户端 API

绝大多数的开发人员如今在其数据库应用程序中使用 API 以嵌入 SQL。例如，两个流行的使程序能与 Oracle 数据库进行通信的 API，是开放式数据库连接 (ODBC) 和 JDBC。另外两个常见的客户端编程 API 是 Oracle 调用接口 (OCI) 和 Oracle C++调用接口（OCCI）。

OCI 和 OCCI

作为预编译器的替代方法， Oracle 提供了 OCI 和 OCCI API。OCI 使你可以使用一种宿主语言（如 C）来操作数据库中的数据和模式。OCCI 是一种面向对象的、适合与 C++一起使用的接口。这两个 API 使开发人员能够使用本机子程序调用，来访问 Oracle 数据库和控制 SQL 执行。

在某些情况下，OCI 比更高级的接口提供了更好的性能或更多的功能。OCI 和 OCCI 提供了许多功能，包括以下这些：

- 支持所有通过 Oracle 数据库可用的 SQL DDL、 DML、 查询、和事务控制设施
- 即时客户端，它是磁盘空间有限时部署应用程序的一种方法
- 线程管理、 连接池、 全球化功能、以及从 C 应用程序对数据进行直接路径加载

OCI 和 OCCI 以一种动态运行时库 (OCILIB)形式，提供了一个具备标准数据库访问和检索功能的库。这个库可以在运行时链入应用程序。因此，你可以按非数据库应用程序相同的方式，来编译和链接 OCI 或 OCCI 程序，避免单独的预处理或预编译步骤。图 19-3 说明了这种开发过程。

图 19-3 使用 OCI 或 OCCI 时的开发过程

Description of "Figure 19-3 Development Process Using OCI or OCCI"

See Also:

- *Oracle Call Interface Programmer's Guide*
- *Oracle C++ Call Interface Programmer's Guide*

ODBC and JDBC

ODBC is a standard API that enables applications to connect to a database and then prepare and run SQL statements. ODBC is independent of programming language, database, and operating system. The goal of ODBC is to enable any application to access data contained in any database.

A **database driver** is software that sits between an application and the database. The driver translates the API calls made by the application into commands that the database can process. By using an ODBC driver, an application can access any data source, including data stored in spreadsheets. The ODBC driver performs all mappings between the ODBC standard and the database.

The Oracle ODBC driver provided by Oracle enables ODBC-compliant applications to access Oracle Database. For example, an application written in Visual Basic can use ODBC to query and update tables in an Oracle database.

JDBC is a low-level Java interface that enables Java applications to interact with Oracle database. Like ODBC, JDBC is a vendor-independent API. The JDBC standard is defined by Sun Microsystems and implemented

Description of "Figure 19-3 Development Process Using OCI or OCCI"

另见：

- 《*Oracle 调用接口程序员指南*》
- 《*Oracle C++ 调用接口程序员指南*》

ODBC and JDBC

ODBC 是一个标准的 API，使应用程序能够连接到数据库，然后准备和运行 SQL 语句。ODBC 独立于编程语言、 数据库、和操作系统。ODBC 的目标是使任何应用程序可以访问任何数据库中包含的数据。

数据库驱动程序是位于应用程序和数据库之间的软件。驱动程序将应用程序发出的 API 调用转换为可以由数据库处理的命令。通过使用 ODBC 驱动程序，应用程序可以访问任何数据源，包括存储在电子表格中的数据。ODBC 驱动程序执行 ODBC 标准和数据库之间的所有映射。

由 Oracle 提供的 Oracle ODBC 驱动程序，使符合 ODBC 的应用程序能访问 Oracle 数据库。例如，在一个用 Visual Basic 编写的应用程序可以使用 ODBC 查询和更新 Oracle 数据库中的一个表。

JDBC 是一个低级的 Java 接口，使 Java 应用程序能与数据库进行交互。与 ODBC 类似， JDBC 是独立于供应商的 API。JDBC 标准是由太阳微系统公

through the `java.sql` interfaces.
The JDBC standard enables individual providers to implement and extend the standard with their own JDBC drivers. Oracle provides the following JDBC drivers for client-side programming:

- JDBC thin driver

  This pure Java driver resides on the client side without an Oracle client installation. It is platform-independent and usable with both applets and applications.
- JDBC OCI driver

  This driver resides on the client-side with an Oracle client installation. It is usable only with applications. The JDBC OCI driver, which is written in both C and Java, converts JDBC calls to OCI calls.

The following snippets are from a Java program that uses the JDBC OCI driver to create a `Statement` object and query the `dual` table:

```
// Create a statement
Statement stmt = conn.createStatement();

// Query dual table
ResultSet rset = stmt.executeQuery("SELECT 'Hello
World' FROM DUAL");
```

See Also:

- *Oracle Database Advanced Application Developer's Guide* and *Oracle Database 2 Day + Java Developer's Guide* to learn more about JDBC
- *Oracle Database Gateway for ODBC User's Guide*

Globalization Support

Oracle Database **globalization support** enables you to store, process, and retrieve data in native languages. Thus, you can develop multilingual applications and software that can be accessed and run from anywhere in the world simultaneously.
Developers who write globalized database application must do the following:

- Understand the Oracle Database globalization support architecture, including the properties of the different character sets, territories,

司定义的，并通过 java.sql 接口实现。
JDBC 标准使各个提供商来实现和扩展他们自己的 JDBC 驱动程序。Oracle 为客户端编程提供以下的 JDBC 驱动程序：

- JDBC 瘦驱动程序

  这个纯 Java 驱动程序驻留在没有 Oracle 安装客户端软件的客户端主机上。它是独立于平台的，并能与小应用程序和应用程序一起使用。

- JDBC OCI 驱动程序

  此驱动程序驻留在安装了 Oracle 客户端软件的客户端主机上。它只能用于应用程序。JDBC OCI 驱动程序是用 C 和 Java 编写的，它将 JDBC 调用转换为 OCI 调用。

以下代码片断来自某个 Java 程序，该程序使用 JDBC OCI 驱动程序创建 `Statement` 对象，并查询 `dual` 表：

```
// Create a statement
Statement stmt = conn.createStatement();

// Query dual table
ResultSet rset = stmt.executeQuery("SELECT 'Hello
World' FROM DUAL");
```

另见：

- 《*Oracle 数据库高级应用程序开发者指南*》和《*Oracle 数据库 2 日+ Java 开发人员指南*》了解 JDBC 的更多信息

- 《*Oracle 数据库网关 ODBC 用户指南*》

全球化支持

Oracle 数据库的**全球化支持**，使您能够按本地语言来存储、 处理、并检索数据。因此，您可以开发多语言应用程序和软件，同时可以从世界上的任何位置访问和运行。

开发人员编写全球化的数据库应用程序必须执行以下操作：

- 了解 Oracle 数据库全球化支持体系结构，包括不同的字符集、 地

languages, and linguistic sort definitions

- Understand the globalization functionality of their middle-tier programming environment, including how it can interact and synchronize with the locale model of the database
- Design and write code capable of simultaneously supporting multiple clients running on different operating systems, with different character sets and locale requirements

For example, an application may be required to render content of the user interface and process data in languages and locale preferences of native users. For example, the application must process multibyte Kanji data, display messages and dates in the proper regional format, and process 7-bit ASCII data without requiring users to change settings.

See Also:

Oracle Database Globalization Support Guide for more information about globalization

Globalization Support Environment

The **globalization support environment** includes the client application and the database. You can control language-dependent operations by setting parameters and environment variables on the client and server, which may exist in separate locations.

Note:

In previous releases, Oracle referred to globalization support capabilities as National Language Support (NLS) features. NLS is actually a subset of globalization support and provides the ability to choose a national language and store data in a specific character set.

Oracle Database provides globalization support for features such as:

- Native languages and territories
- Local formats for date, time, numbers, and currency
- Calendar systems (Gregorian, Japanese, Imperial, Thai Buddha, and so on)
- Multiple character sets, including Unicode
- Character semantics

Character Sets

A key component of globalization support is a **character set**, which is an encoding scheme used to display characters on your computer screen. The following distinction is important in application development:

- A **database character set** determines which languages can be

域、 语言、和语言的排序定义等属性

- 了解他们的中间层编程环境的全球化功能，包括如何与数据库的区域设置模式进行交互并保持同步

- 设计和编写代码，能够同时支持运行在不同操作系统上，具有不同的字符集和语言环境要求的多个客户端，

例如，应用程序可能需要按本机用户的语言和区域设置首选项，在用户界面显示其内容和处理数据。例如，应用程序必须能够处理多字节汉字数据，按适当的区域格式显示消息和日期，能够处理 7 位 ASCII 数据而无需用户更改设置。

另见：

《*Oracle 数据库全球化支持指南*》有关全球化的详细信息

全球化支持环境

全球化支持环境包括客户端应用程序和数据库。您可以通过在可能存在于不同位置的客户端上和服务器上设置参数和环境变量，来控制与语言相关的操作。

注意：

在以前的发行版中，Oracle 全球化支持功能称为国家语言支持 (NLS) 功能。NLS 实际上是全球化支持的一个子集，并提供了选择某种国家语言并将数据存储为特定字符集的能力。

Oracle 数据库提供全球化支持的功能，例如：

- 本机语言和地区
- 日期、 时间、 数字、和货币的本机格式
- 日历系统 (公历、 日本日历、 皇历、 泰历等等）
- 包括 Unicode 在内的多种字符集
- 字符语义

字符集

全球化支持的一个关键组件是字符集，它是一个用来在您的计算机屏幕上显示字符的编码方案。以下区别对应用程序开发很重要：

- **数据库字符集**确定了在一个数据库中可以表示哪些语言。字符集是在

represented in a database. The character set is specified at database creation.

Note:

After a database is created, changing its character set is usually very expensive in terms of time and resources. This operation may require converting all character data by exporting the whole database and importing it back.

- A **client character set** is the character set for data entered or displayed by a client application. The character set for the client and database can be different.

A group of characters (for example, alphabetic characters, ideographs, symbols, punctuation marks, and control characters) can be encoded as a character set. An **encoded character set** assigns a unique numeric code, called a **code point** or **encoded value**, to each character in the set. Code points are important in a global environment because of the potential need to convert data between different character sets.

The computer industry uses many encoded character sets. These sets differ in the number of characters available, the characters available for use, code points assigned to each character, and so on. Oracle Database supports most national, international, and vendor-specific encoded character set standards.

Oracle Database supports the following classes of encoded character sets:

- Single-Byte character sets

  Each character occupies one byte. An example of a 7-bit character set is US7ASCII. An example of an 8-bit character set is WE8DEC.
- Multibyte character sets

  Each character occupies multiple bytes. Multibyte sets are commonly used for Asian languages.
- Unicode

  The universal encoded character set enables you to store information in any language by using a single character set. Unicode provides a unique code value for every character, regardless of the platform, program, or language.

See Also:

Oracle Database Globalization Support Guide to learn about character set

创建数据库时指定的。

注意：

数据库被创建后，要更改其字符集通常是非常耗费时间和资源的。此操作可能需要导出整个数据库，然后再将其导入，来转换所有字符数据。

- **客户端字符集**是由某个客户端应用程序输入或显示的数据的字符集。客户端和数据库的字符集可以是不同的。

一组字符（例如字母字符、 表意文字、 符号、 标点符号、和控制字符）可以被编码为一个字符集。一个**编码字符集**为集合中的每个字符分配一个唯一的数字代码，称为**代码点**或**编码值**。代码点在全球环境中很重要，因为可能需要在不同的字符集之间转换数据。

计算机行业使用很多编码字符集。这些集合在可用字符及其数目、分配给每个字符的代码点等方面都有所不同。Oracle 数据库支持大多数的国家字符集、 国际字符集、和特定于供应商的编码字符集等标准。

Oracle 数据库支持如下类别的编码字符集：

- 单字节字符集

  每个字符占一个字节。7 位字符集的一个例子是 US7ASCII。8 位字符集的一个例子是 WE8DEC。
- 多字节字符集

  每个字符占用多个字节。多字节字符集通常用于亚洲语言。

- Unicode

  通用编码字符集，使你能够通过使用一种单一字符集，来存储任何一种语言的信息。Unicode 为每个字符提供了一个唯一的编码值，而无论其平台、 程序、或语言。

另见：

《*Oracle 数据库全球化支持指南*》了解字符集迁移

migration

Locale-Specific Settings

A **locale** is a linguistic and cultural environment in which a system or program is running. **NLS parameters** determine locale-specific behavior on both the client and database. A database session uses NLS settings when executing statements on behalf of a client. For example, the database makes the correct territory usage of the thousands separator for a client.

Typically, the `NLS_LANG` environment variable on the client host specifies the locale for both the server session and client application. The process is as follows:

- 1. When a client application starts, it initializes the client NLS environment from the environment settings.

  All NLS operations performed locally, such as displaying formatting in Oracle Developer applications, use these settings.
- 2. The client communicates the information defined by `NLS_LANG` to the database when it connects.
- 3. The database session initializes its NLS environment based on the settings communicated by the client.
  If the client did not specify settings, then the session uses the settings in the initialization parameter file. The database uses the initialization parameter settings only if the client did not specify any NLS settings. If the client specified some NLS settings, then the remaining NLS settings default.

Each session started on behalf of a client application may run in the same or a different locale as other sessions. For example, one session may use the German locale while another uses the French locale. Also, each session may have the same or different language requirements specified. Table 19-1 shows two clients using different `NLS_LANG` settings. A user starts SQL*Plus on each host, logs on to the same database as `hr`, and runs the same query simultaneously. The result for each session differs because of the locale-specific NLS setting for floating-point numbers.

Table 19-1 Locale-Specific NLS Settings

t	Client Host 1	Client Host 2
t0	`$ export`	`$ export`

特定于区域的设置

区域设置是某个系统或程序正在其中运行的语言和文化环境。由 **NLS 参数**确定客户端和数据库所在的特定区域的行为。执行客户端语句时，数据库会话使用 NLS 设置。例如，数据库为客户端使用正确的千位分隔符区域用法。

通常，由客户机主机上的 NLS_LANG 环境变量，指定服务器会话和客户端应用程序的区域设置。该过程如下所示：

- 1. 当客户端应用程序启动时，它根据其环境设置初始化客户端 NLS 环境。

  所有 NLS 操作都在本地执行，例如，在 Oracle Developer 应用程序中使用这些设置的显示格式。
- 2. 当客户端与数据库连接并进行通信时，使用由 NLS_LANG 定义的信息格式。
- 3. 数据库会话基于客户端的通信设置，初始化其 NLS 环境。

  如果客户端没有指定设置，则会话使用初始化参数文件中的设置。仅当客户端未指定任何 NLS 设置时，数据库才会使用初始化参数设置。如果客户端指定了某些 NLS 设置，则其余的 NLS 设置将保持默认值。

每个为客户端应用程序启动的会话，可能与其他会话运行在相同或不同的区域设置。例如，一个会话可能会使用德语区域设置，而另一个可能会使用法语区域设置。每个会话还可能指定了相同或不同的语言要求。

表 19-1 显示了使用不同 NLS_LANG 设置的两个客户端。用户在每台主机上启动 SQL*Plus 以 `hr` 登录到同一数据库，并同时运行相同的查询。由于对于浮点数的特定于区域设置的 NLS 设置不同，每个会话的结果有所不同。

表 19-1 特定于区域设置的 NLS 设置

t	客户端主机 1	客户端主机 2
t0	`$ export`	`$ export`

	NLS_LANG=American_America.US7ASCII	NLS_LANG=German_Germany.US7ASCII
t1	$ sqlplus /nolog SQL> CONNECT hr@proddb Enter password: ******* SQL> SELECT 999/10 FROM DUAL; 999/10 ---------- 99.9	$ sqlplus /nolog SQL> CONNECT hr@proddb Enter password: ******* SQL> SELECT 999/10 FROM DUAL; 999/10 ---------- 99,9

See Also:

Oracle Database 2 Day + Java Developer's Guide and *Oracle Database Globalization Support Guide* to learn about NLS settings

Oracle Globalization Development Kit

The **Oracle Globalization Development Kit (GDK)** simplifies the development process and reduces the cost of developing Internet applications used to support a global environment. The GDK includes comprehensive programming APIs for both Java and PL/SQL, code samples, and documentation that address many of the design, development, and deployment issues encountered while creating global applications.

The GDK mainly consists of two parts: GDK for Java and GDK for PL/SQL. GDK for Java provides globalization support to Java applications. GDK for PL/SQL provides globalization support to the PL/SQL programming environment. The features offered in the two parts are not identical.

See Also:

Oracle Database Globalization Support Guide

Unstructured Data

The traditional relational model deals with simple structured data that fits into simple tables. Oracle Database also provides support for **unstructured data**, which cannot be decomposed into standard components. Unstructured data includes text, graphic images, video clips, and sound waveforms.

Oracle Database includes data types to handle unstructured content. These data types appear as native types in the database and can be queried using SQL.

Overview of XML in Oracle Database

Oracle XML DB is a set of Oracle Database technologies related to high-

	NLS_LANG=American_America.US7ASCII	NLS_LANG=German_Germany.US7ASCII
t1	$ sqlplus /nolog SQL> CONNECT hr@proddb Enter password: ******* SQL> SELECT 999/10 FROM DUAL; 999/10 ---------- 99.9	$ sqlplus /nolog SQL> CONNECT hr@proddb Enter password: ******* SQL> SELECT 999/10 FROM DUAL; 999/10 ---------- 99,9

另见：

《*Oracle 数据库 2 日+ Java 开发人员指南*》和《*Oracle 数据库全球化支持指南*》了解 NLS 设置

Oracle 全球化开发工具包

Oracle 全球化开发工具包 （GDK） 简化了开发过程，并降低了开发用来支持全球环境的互联网应用程序的成本。 GDK 包括丰富的 Java 和 PL/SQL 编程 API、 代码示例、以及用于解决在创建全球化应用程序时遇到的设计、 开发、部署问题的文档资料。

GDK 主要由两部分组成：GDK for Java 和 GDK for PL/SQL。为 GDK for Java 提供了对 Java 应用程序的全球化支持。GDK for PL/SQL 提供了对 PL/SQL 编程环境的全球化支持。两个部分所提供的功能也不相同。

另见：

《*Oracle 数据库全球化支持指南*》

非结构化数据

传统关系模型处理适用于简单表的简单结构化数据。Oracle 数据库也支持**非结构化数据**，这种数据不能被分解成标准的组件。非结构化的数据包括文本、 图像、 视频剪辑、和声音波形。

Oracle 数据库包含用于处理非结构化内容的数据类型。这些数据类型在数据库中显示为本机类型，并且可以使用 SQL 查询。

Oracle 数据库中的 XML 的概述

Oracle XML DB 是一组高性能 XML 存储和检索相关的 Oracle 数据库技

performance XML storage and retrieval. XML DB provides native XML support by encompassing both SQL and XML data models in an interoperable manner.

Oracle XML DB is suited for any Java or PL/SQL application where some or all of the data processed by the application is represented using XML. For example, the application may have large numbers of XML documents that must be ingested, generated, validated, and searched.

Oracle XML DB provides many features, including the following:

- The native `XMLType` data type, which can represent an XML document in the database so that it is accessible by SQL
- Support for XML standards such as XML Schema, XPath, XQuery, XSLT, and DOM
- `XMLIndex`, which supports all forms of XML data, from highly structured to completely unstructured

Example 19-1 creates a table `orders` of type `XMLType`. The example also creates a SQL directory object, which is a logical name in the database for a physical directory on the host computer. This directory contains XML files. The example inserts XML content from the `purchaseOrder.xml` file into the `orders` table.

Example 19-1 XMLType

```
CREATE TABLE orders OF XMLType;
CREATE DIRECTORY xmldir AS path_to_folder_containing_XML_file;
INSERT INTO orders VALUES (XMLType(bfilename('XMLDIR', 'purchaseOrder.xml'),
nls_charset_id('AL32UTF8')));
```

The **Oracle XML developer's kits (XDK)** contain the basic building blocks for reading, manipulating, transforming, and viewing XML documents, whether on a file system or in a database. APIs and tools are available for Java, C, and C++. The production Oracle XDKs are fully supported and come with a commercial redistribution license.

See Also:

- *Oracle XML DB Developer's Guide*
- *Oracle XML Developer's Kit Programmer's Guide*
- *Oracle Database XML Java API Reference*

Overview of LOBs

The large object (LOB) data types enable you to store and manipulate

术。XML 数据库提供本机 XML 支持的可互操作的方式，包括 SQL 和 XML 数据模型。

Oracle XML DB 适合于任何 Java 或 PL/SQL 应用程序，由该应用程序处理的部分或全部数据使用 XML 来表示。例如，应用程序可能需要接收、 生成、 验证、和搜索大量的 XML 文档。

Oracle XML DB 提供了许多功能，包括以下这些：

- 本机 XMLType 数据类型，可以表示数据库中的 XML 文档，以便能通过 SQL 访问
- 对 XML 标准如 XML 模式、 XPath、 Xquery、XSLT 、和 DOM 的支持
- `XMLIndex` 支持从高度结构化到完全非结构化的所有形式的 `XML` 数据

示例 19-1 创建了一个 XMLType 类型的 `orders` 表。该示例还创建一个 SQL 目录对象，它是数据库中针对主机计算机上某个物理目录的逻辑名称。此目录下包含 XML 文件。该示例将来自 purchaseOrder.xml 文件的 XML 内容插入 `orders` 表。

示例 19 1 XMLType

```
CREATE TABLE orders OF XMLType;
CREATE DIRECTORY xmldir AS path_to_folder_containing_XML_file;
INSERT INTO orders VALUES (XMLType(bfilename('XMLDIR', 'purchaseOrder.xml'),
nls_charset_id('AL32UTF8')));
```

Oracle XML 开发人员工具包 (XDK) 中包含用于在文件系统或数据库中读取、 操作、 转换、和查看 XML 文档的基本构造块。其 API 和工具可用于 Java、 C 、和 C++。Oracle XDK 产品完全受支持，并需要一个商用重发布许可证。

另见：

- *《Oracle XML 数据库开发人员指南》*
- *《Oracle XML 开发人员工具包程序员指南》*
- *Oracle 数据库 XML Java API 参考*

LOB 概述

大对象 (LOB) 数据类型使您能够按二进制或字符格式存储和操作大块的非

large blocks of unstructured data in binary or character format. LOBs provide efficient, random, piece-wise access to the data.

Internal LOBs

An **internal LOB** stores data in the database itself rather than in external files. Internal LOBS include the following:

- CLOB (character LOB), which stores large amounts of text, such as text or XML files, in the database character set
- NCLOB (national character set LOB), which stores Unicode data
- BLOB (binary LOB), which stores large amounts of binary information as a bit stream and is not subject to character set translation

The database stores LOBs differently from other data types. Creating a LOB column implicitly creates a **LOB segment** and a **LOB index** (see "User Segment Creation"). The tablespace containing the LOB segment and LOB index, which are always stored together, may be different from the tablespace containing the table.

Note:

Sometimes the database can store small amounts of LOB data in the table itself rather than in a separate LOB segment.

The LOB segment stores data in pieces called **chunks**. A chunk is a logically contiguous set of data blocks and is the smallest unit of allocation for a LOB. A row in the table stores a pointer called a **LOB locator**, which points to the LOB index. When the table is queried, the database uses the LOB index to quickly locate the LOB chunks.

The database manages read consistency for LOB segments differently from other data (see "Read Consistency and Undo Segments"). Instead of using undo data to record changes, the database stores the before images in the segment itself. When a transaction changes a LOB, the database allocates a new chunk and leaves the old data in place. If the transaction rolls back, then the database rolls back the changes to the index, which points to the old chunk.

External LOBs

A BFILE (binary file LOB) is an **external LOB** because the database stores a pointer to a file in the operating system. The external data is read-only.

A BFILE uses a directory object to locate data. The amount of space consumed depends on the length of the directory object name and the

结构化数据。LOB 提供高效的、 随机的、 分段访问数据。

内部 LOB

内部 LOB 将数据存储在数据库本身之中，而不是在外部文件中。内部 LOB 包括：

- CLOB（字符 LOB）按数据库字符集存储大量的文本，如文本或 XML 文件，
- NCLOB （国家字符集 LOB)存储 Unicode 数据
- BLOB （二进制 LOB)，将大量的二进制信息存储为位流，并不受字符集转换的限制

该数据库以与其他数据类型不同的方式存储 LOB。创建一个 LOB 列会隐式创建一个 **LOB 段**和一个 **LOB 索引** （请参见"创建用户段"）。 LOB 段及其 LOB 索引总是一起存储在同一表空间中，但它可能不同于包含表的表空间。

注意：

有时数据库可以将 LOB 数据中的一小部分存储在表本身之中，而不是在一个单独的 LOB 段中。

LOB 段将数据存储为称作**大块**的片断。大块是一组逻辑上连续的数据块，是 LOB 的最小分配单位。表中的一个行存储一个叫做 **LOB 定位器**的指针，指向 LOB 索引。当查询表时，数据库将使用 LOB 索引可以快速找到各个 LOB 块。

数据库管理 LOB 段的读取一致性的的方式不同于其他数据 （请参阅"读取一致性和撤消段"）。该数据库不是使用撤消数据来记录更改，而是将前像存储在段本身之中。当一个事务更改了 LOB 时，数据库分配一个新的大块，并原地保留旧数据。如果事务回滚，则数据库回滚对索引所做的更改，以指向旧的大块。

外部 LOB

BFILE （二进制文件 LOB）是一个**外部 LOB**，因为数据库存储了指向操作系统中的某个文件的指针。外部数据是只读的。

BFILE 使用目录对象来查找数据。使用的空间消耗量取决于目录对象名称的长度和文件名称的长度。

length of the file name.
A BFILE does not use the same read consistency mechanism as internal LOBS because the binary file is external to the database. If the data in the file changes, then repeated reads from the same binary file may produce different results.

因为二进制文件是在数据库外部的，所以 BFILE 不使用与内部 LOB 相同的读取一致性机制。如果数据文件更改了，则重复读取同一个二进制文件可能会产生不同的结果。

SecureFiles

SecureFiles is a LOB data type specifically engineered to deliver high performance for file data comparable to that of traditional file systems, while retaining the advantages of Oracle Database. The SECUREFILE LOB parameter enables advanced features typically found in high-end file systems, such as deduplication, compression, encryption, and journaling.

安全文件

SecureFiles 是一种专门设计的 LOB 数据类型，能提供相当于传统文件系统的文件数据的高性能，同时又保留 Oracle 数据库的优点。SECUREFILE LOB 参数使其具有通常见于高端文件系统中的高级功能，如重复数据删除、 压缩、 加密、和日志。

See Also:

- "Oracle Data Types"
- *Oracle Database SecureFiles and Large Objects Developer's Guide* to learn more about LOB data types

另见：

- "Oracle 数据类型"
- 《*Oracle 数据库 SecureFiles 和大对象开发人员指南*》了解有关 LOB 数据类型的更多信息

Overview of Oracle Text

Oracle Text (Text) is a fast and accurate full-text retrieval technology integrated with Oracle Database. Oracle Text indexes any document or textual content stored in file systems, databases, or on the Web. These documents can be searched based on their textual content, metadata, or attributes.

Oracle Text provides the following advantages:

- Oracle Text allows text searches to be combined with regular database searches in a single SQL statement. The Text index is in the database, and Text queries are run in the Oracle Database process. The optimizer can choose the best execution plan for any query, giving the best performance for ad hoc queries involving Text and structured criteria.
- You can use Oracle Text with XML data. In particular, you can combine XMLIndex with Oracle Text indexing, taking advantage of both XML and a full-text index.
- The Oracle Text SQL API makes it simple and intuitive to create and maintain Oracle Text indexes and run searches.

For a use case, suppose you must create a catalog index for an auction site that sells electronic equipment. New inventory is added every day.

Oracle 文本的概述

Oracle 文本 （文本） 是一种与 Oracle 数据库集成的快速、 准确的全文检索技术。Oracle 文本能索引任何文档，或存储在文件系统、数据库、或 Web 上的文字内容。您可以基于其文本内容、 元数据、或属性来搜索这些文档。

Oracle 文本提供了以下优势：

- Oracle 文本可在一个单独的 SQL 语句中结合使用文本搜索与常规数据库搜索。文本索引存储在数据库中，文本查询是在 Oracle 数据库进程中运行的。优化程序可以为任何查询选择最佳的执行计划，为涉及文本和结构化条件的特殊查询提供最佳性能。
- 您可以在 Oracle 文本中使用 XML 数据。特别是，您可以组合使用 XMLIndex 与 Oracle 文本索引、 同时利用 XML 和全文本索引的优势。
- Oracle 文本 SQL API 使创建和维护 Oracle 文本索引并运行搜索，变得简单和直观。

举一个使用案例，假设您必须为一个销售电子设备的拍卖网站创建目录索引。每一天都有新库存清单加入。项目说明、 投标日期、和价格必须一起

Item descriptions, bid dates, and prices must be stored together. The application requires good response time for mixed queries. First, you create and populate a `catalog` table. You then use Oracle Text to create a `CTXCAT` index that you can query with the `CATSEARCH` operator in a `SELECT ... WHERE CATSEARCH` statement.

Figure 19-4 illustrates the relation of the catalog table, its `CTXCAT` index, and the catalog application that uses the `CATSEARCH` operator to query the index.

Figure 19-4 Catalog Query Application

Description of "Figure 19-4 Catalog Query Application"

See Also:

- *Oracle Text Application Developer's Guide* and *Oracle Text Reference*
- *Oracle XML DB Developer's Guide* to learn how to perform full-text search over XML data

Overview of Oracle Multimedia

Oracle Multimedia enables Oracle Database to store, manage, and retrieve images, medical images that follow the Digital Imaging and Communications in Medicine (DICOM) standard, audio, and video data in an integrated fashion with other enterprise information. Oracle Multimedia provides object types and methods for:

- Extracting metadata and attributes from multimedia data
- Embedding metadata created by applications into image and DICOM data
- Obtaining and managing multimedia data from Oracle Database, Web servers, file systems, and other servers
- Performing operations such as thumbnail generation on image and DICOM data
- Making DICOM data anonymous
- Checking DICOM data for conformity to user-defined validation rules

存储。应用程序要求对各种查询都要有好的响应时间。首先，您创建并填充一个 `catalog` 表。然后您使用 Oracle 文本创建一个 CTXCAT 的索引，您可以在 `SELECT ... WHERE CATSEARCH` 语句中进行带有 CATSEARCH 运算符的查询。

图 19-4 说明了目录表、它的 CTXCAT 索引、和使用 CATSEARCH 运算符来查询索引的目录应用程序之间的关系。

图 19-4 目录查询应用程序

Description of "Figure 19-4 Catalog Query Application"

另见：

- 《*Oracle 文本应用程序开发人员指南*》和《*Oracle 文本参考*》
- 《*Oracle XML DB 开发人员指南*》了解如何在 XML 数据中执行全文搜索

Oracle 多媒体概述

Oracle 多媒体使 Oracle 数据库能以一种与其他企业信息集成的方式，来存储、 管理、和检索图像、合乎医学数字成像与通信 (DICOM) 标准的医学图像、 音频、视频数据。Oracle 多媒体提供的对象类型和方法用于：

- 从多媒体数据中提取元数据和属性
- 将应用程序产生的元数据嵌入图像和 DICOM 数据
- 从 Oracle 数据库、 Web 服务器、 文件系统、和其他服务器获取和管理多媒体数据
- 执行对图像和 DICOM 数据生成缩略图之类的操作
- 制作匿名的 DICOM 数据
- 检查 DICOM 数据，以符合用户定义的验证规则

See Also:

- *Oracle Multimedia User's Guide* and *Oracle Multimedia Reference*
- *Oracle Multimedia DICOM Developer's Guide* and *Oracle Multimedia DICOM Java API Reference*
- *Oracle Multimedia Java API Reference* and *Oracle Multimedia Servlets and JSP Java API Reference*

Overview of Oracle Spatial

Oracle Spatial (Spatial) provides a SQL schema and functions that facilitate the storage, retrieval, update, and query of collections of spatial features in an Oracle database. Oracle Spatial makes spatial data management easier to users of location-enabled applications and geographic information system (GIS) applications.

An example of spatial data is a road map. The spatial data indicates the Earth location (such as longitude and latitude) of objects on the map. When the map is rendered, this spatial data is used to project the locations of the objects on a two-dimensional piece of paper. A GIS is often used to store, retrieve, and render this Earth-relative spatial data. When spatial data is stored in an Oracle database, the data can be easily manipulated, retrieved, and related to other data.

See Also:

Oracle Spatial Developer's Guide

另见：

- 《*Oracle 多媒体用户指南*》和 《*Oracle 多媒体参考*》
- 《*Oracle 多媒体 DICOM 开发人员指南*》和《*Oracle 多媒体 DICOM Java API 参考*》
- 《*Oracle 多媒体 Java API 参考*》和 《*Oracle 多媒体服务小程序和 JSP Java API 参考*》

Oracle 空间概述

Oracle 空间（空间） 提供了一个 SQL 模式和函数集，以便存储、 检索、更新、和查询 Oracle 数据库中一组空间特征。Oracle 空间使位置使能的应用程序和地理信息系统 (GIS) 应用程序的用户，对空间数据的管理变得更容易。

路线图是空间数据的一个示例。空间数据表明对象在地图上的地球位置（如经度和纬度） 。当地图被着色时，此空间数据用于将对象位置投影到一张二维的纸上。GIS 通常用于存储、 检索、和着色这个与地球相关的空间数据。当空间数据被存储在一个 Oracle 数据库中时，数据可以轻松地操纵、 检索、并与其他数据相关联。

另见：

《*Oracle 空间开发人员指南*》