Oracle 数据库
概念
11g 版本 2 (11.2)
E25789-01

庞浩然　译
2012 年 1 月

1

Oracle 数据库导论

本章为 Oracle 数据库概述并包含以下部分:

- 关于关系型数据库
- 模式对象
- 数据访问
- 事物管理
- Oracle 数据库架构
- Oracle 数据库文档路线图

关于关系型数据库

每个组织都有他们需要存储和管理的信息。例如，一个企业必须收集并维护其员工的人力资源记录。这些信息必须以可用的方式提供给需要它的人。**信息系统**（**information system**）是一个用于存储和处理信息的正式系统。

信息系统可以是包含各种办公文件夹、文件保存方式说明和文件查找标签的大纸箱子。然而，现在更多的公司使用**数据库**（**database**）来让他们的系统自动化。数据库是一个可以对外提供服务的有组织的信息集合单元。使用数据库的目的是收集、存储和检索信息以提供给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)是控制存储、组织和数据检索的软件。通常情况下，数据库检索系统(DBMS)包含以下元素：

- 内核代码
  内核代码管理数据库管理系统(DBMS)的内存和存储。
- 元数据资料库
  元数据资料库被称作**数据字典**（**data dictionary**）。
- 查询语言
  查询语言允许应用程序访问数据。

数据库应用（**database application**）是一种软件程序可以通过访问数据库并操纵数据来和数据库进行交互。

第一代数据库管理系统包括以下几种类型：

关于关系型数据库

- 分层

  分层数据库（**hierarchical database**）使用 Tree 数据结构组织数据。每一个父记录（父节点）有一个或多个子记录（子节点），类似于文件系统结构。
- 网络

  网络数据库（**network database**）类似于分层数据库，除去有多对多记录而不是一对多关系之外。

过去的数据库管理系统使用很死板（rigid）的预定关系方式存储数据。因为那时还没有一种数据定义语言存在，所以想改变数据结构是件困难的事情。另外，这些系统缺乏一种简单的查询语言，从而阻碍了应用程序的开发。

关系模型

在 1970 年一篇开创性的论文“大型共享数据银行的数据关系模型”中，作者 E.F.Codd 定义了一个基于数学集合论的关系模型。今天，最被广泛接受的数据库模型是关系模型。

关系型数据库（**relational database**）是一种符合关系模型的数据库。关系模型主要有以下方面：

- 结构

  有良好定义的存储对象或访问数据的数据库。
- 操作

  明确定义应用程序可以操作数据库中数据和结构的动作。
- 完整性规则

  完整性规则管理数据库的数据和结构的操作。

关系型数据库存储数据在简单关系集合中。一个**关系**（**relation**）是一个元组集。一个**元组**（**tuple**）是一个无序的属性值。

表（**table**）是一个行（元组）和列（属性）形式关系的二维表示。每一个在表中的行都有相同的列集合。关系数据库即为数据库中使用关系方式存储数据（表）。例如，关系型数据库能够存储关于公司员工的信息在一张员工表中，一张部门表中和一张工资表中。

参见: http://portal.acm.org/citation.cfm?id=362685 Codd 的论文摘要和链接

关系型数据库管理系统 (RDBMS)

关系模型是**关系型数据库管理系统**（**relational database management system**）的基石。从本质上讲，关系型数据库管理系统(RDBMS)将数据移进数据库中、存储数据并检索它，使它能被应用程序操纵。关系型数据库管理系统用以下操作进行分类：

- 逻辑操作

  在这种情况下，应用程序会指定需要什么内容。例如，应用程序请求一个雇员的名字或者增加雇员的记录到表中。
- 物理操作

在这种情况下，关系型数据库管理系统决定如何处理并进行操作。例如，在应用程序查询一张表之后，数据库可能使用一个索引去查找请求的行，读取这些数据到内存中，并且在返回结果给用户之前允许执行许多其他步骤。关系型数据库管理系统存储并检索数据，因此对于数据库应用来说物理操作是透明的。

Oracle 数据库是一种关系型数据库管理系统。关系型数据库管理系统实现诸如用户定义类型、继承和多态等面向对象特性，被称为一个**对象关系型数据库管理系统**（**object-relational database management system** (ORDBMS)）。Oracle 数据库已经从关系模型扩展到对象-关系模型，使它可以在关系数据库中存储复杂的业务模型。

Oracle 数据库简史

当前版本的 Oracle 数据库是 30 多年来发展的结果。Oracle 数据库的里程碑如下：

- Oracle 成立

  1977 年, Larry Ellison（拉里埃里森）, Bob Miner（鲍勃迈纳）和 Ed Oates（艾德奥茨）开始的顾问软件开发实验室，成为了关系软件公司（RSI）。1983 年, RSI 变成了 Oracle 系统公司和后来的 Oracle 公司。

- 第一款商用关系型数据库

  1979 年, RSI 推出了 Oracle V2 (Version 2)作为第一款基于 **SQL** 的商用关系型数据库管理系统，这是在关系型数据库的历史上具有里程碑意义的事件。

- 便携版本的 Oracle 数据库

  Oracle Version 3, 发布于 1983, 是第一个运行于大型机、小型机和 PC 的关系型数据库。数据库使用 C 语言编写，使之被移植到各种平台上。

- 增强并发控制、数据分布和可扩展性

  Version 4 推出了多版本**读一致性(read consistency)** 。Version 5,发布于 1985 年，支持客户/服务计算方式和**分布式数据库(distributed database)**系统。Version 6 带来了增强的磁盘 I/O，行级锁，可扩展性以及备份和恢复。此外，Version 6 推出第一个版本的 **PL/SQL 语言**，一种扩展到 SQL 的专有程序。

- PL/SQL 存储程序单元

  Oracle7, 发布于 1992 年，推出 PL/SQL **存储过程**（**stored procedures**）和**触发器**（**triggers**）。

- 对象和分区

  Oracle8 发布于 1997 年作为对象-关系型数据库，支持许多新的数据类型。此外，在 Oracle8 支持大型表的分区。

- 互联网计算

  Oracle8*i* 数据库，发布于 1999 年，提供本地支持的网络协议和服务端的原生 Java 支持。Oracle8*i* 是专为网络计算设计的，使数据库可以部署在多种环境中(multitier environment)。

- Oracle 真实应用集群(Real Application Clusters (Oracle RAC))

  Oracle9*i* 数据库于 2001 年推出了 Oracle RAC，使多**实例**（**instances**）可以同时访问单一数据库。此外，Oracle XML 数据库(Oracle XML DB)推出能够存储和查询 XML 功能

- 网格计算

Oracle 10g 数据库于 2003 年推出**网格计算**（**grid computing**）。这个发布版本能够通过建立基于低成本商用服务器的**网格基础设施**（**grid infrastructure**）来组织虚拟化计算资源。主要的目的是使数据库能够自我管理和自我调整。**Oracle 自动存储管理(Oacle Automatic Storage Management(Oracle ASM))** 通过虚拟化和简化数据库存储管理来帮助实现这一目标。

- 可管理性、可诊断性和可用性

  Oracle 11g 数据库发布于 2007 年，推出了主机方面的新功能，使管理员和开发人员能够快速适应商业需求的改变。可适应性的关键是通过整合信息并尽可能使用自动化管理来简化信息基础设施。

  参见: http://www.oracle.com/technetwork/issue-archive/2007/07-jul/o4730-090772.html **Oracle® Database Concepts 11g Release 2 (11.2)**

模式对象

关系型数据库管理系统的一种特征是从逻辑数据结构中独立出物理数据存储。在 Oracle 数据库中，一个数据库**模式**（**schema**）是一组逻辑数据结构的集合，或者是**模式对象**（**schema objects**）的集合。数据库用户和有相同名称的**用户名**（**user name**）拥有数据库的模式。

模式对象是用户创建的结构，用来直接将数据提交到数据库中。数据库支持很多不同类型的模式对象，最重要的是表和索引。

参见: "**模式对象简介**"在 2-1 页

表

一张表描述一个实体，例如员工。你可以用表名定义一张表，例如 employees 表和字段（列）的集合。通常情况下，当你创建一张表时，需要给每一个**列**（**column**）写一个名称、一个**数据类型**（**data type**）和一个大小。

一张表是许多行的集合。表中的列标识实体属性的描述，而**行**（**row**）标识了实体的一个实例。例如，员工实体的属性对应 employee ID 和姓。一行标识一位特定的员工。

你可以指定表中每一列的规则。这些规则被称作**完整性约束**（**integrity constraints**）。例如 NOT NULL 完整性约束。这种强制约束的列作用于每一行的值中。

参见:

- "**表的概述**" 在 2-6 页
- **第五章，"数据完整性"**

索引

索引（**index**）是一种可选的数据结构，你可以在一张表上创建一个或多个列的索引。索引可以提高数据检索的性能。当处理请求时，数据库可以使用可用的索引有效定位请求的行。当应用程序经常需要查询特定的行或范围行的时候索引是非常有用的。

索引是逻辑方面和物理方面都独立的数据。因此，在删除和创建索引时不会对表和其它索引产生影响。在删除索引后所有的应用程序将继续工作。

参见: "索引概述" 在 3-1 页

数据访问

对数据库管理系统的一般要求是必须符合数据访问语言的工业化标准。

结构化查询语言 Structured Query Language (SQL)

SQL 是基于集合的声明式语言，提供一个接口访问关系型数据库管理系统，例如 Oracle 这样的数据库。对比过程化语言，如 C 语言，需要声明事情如何去做，而 SQL 是非过程化的并且需要描述做什么的语言。用户可以指定想要的结果（例如，当前员工的名字），而不是如何生成它。SQL 是关系型数据库的 ANSI（American National Standards Institute 美国国家标准学会）标准语言。

在 Oracle 数据库中的所有数据操作都使用 SQL 语句。例如，使用 SQL 去创建表并查询和修改表中数据。SQL 语句可以被看做是简单而强大的计算机程序或指令。如下面的 SQL 语句是 SQL 文本的字符串：

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

SQL 语句使你可以执行以下任务：

- 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
- 控制对数据库的访问和它的对象
- 保证数据库的一致性和完整性

SQL unifies the preceding tasks in one consistent language. **Oracle SQL** 是 ANSI 标准的一种实现。Oracle SQL 支持众多超越标准 SQL 的特性。

参见:第 7 章, "SQL"

PL/SQL 和 Java

PL/SQL 是过程化的扩展至 Oracle SQL（a procedural extension to Oracle SQL）。PL/SQL 在 Oracle 数据库中是综合的，使你能使用所有的 Oracle 数据库 SQL 语句、函数和数据类型。你能使用 PL/SQL 控制 SQL 程序的流程、使用变量并写出错误处理过程。

PL/SQL 主要的好处是能够存储应用的逻辑在数据库本身中。**过程**（**procedure**）或**函数**（**function**）是一种模式对象由一个 SQL 语句集和其他 PL/SQL 构成、组织到一起，存储于数据库中并运行用来解决一个具体问题或者执行相关任务。在服务器端编程的主要好处是内置的功能可以部署到任何地方。

Oracle 数据库也可以存储 Java 程序单元。Java 存储过程是一个发布到 SQL 并且存储

在数据库中常用的 Java 方法。你可以从 Java 程序和 PL/SQL 中的 Java 程序调用已存在的 PL/SQL 程序。

事务管理

参见：第 8 章， "服务器端编程：PL/SQL 和 Java" 和 **"客户端数据库编程"** 在 19-5 页

事务管理

Oracle 是一个多用户数据库。数据库必须确保多用户能同时工作而不破坏彼此的数据。

事务

关系型数据库管理系统必须能分组 SQL 语句使它们都能够被**提交**（**committed**），这意味着它们都被应用到数据库中，或者全都被**回滚**（**rolled back**），也就是它们都被撤销执行。**事务**（**transaction**）是一种逻辑，工作的原子单元包含一个或多个 SQL 语句。

一个交易需要的例证是从储蓄账户转到支票账户下。转账包括下列隔离操作：

1. 减去储蓄账户金额
2. 增加支票账户金额
3. 在交易日记中记录交易

无论执行成功还是失败，Oracle 数据库确保所有的三个操作是一个单元。例如，当一个硬件故障切断了事务中正在执行的交易语句，这时其它的语句必须回滚。

事务是一种功能可以设置除文件系统外的 Oracle 数据库。（Transactions are one of the features that sets Oracle Database apart from a file system.） 如果你执行一个原子操作时更新了几个文件，并且中途系统出现了故障，那么这些文件就不一致了。相比之下，事务将数据库从一致状态移动到另一个状态。事务的基本原则是“全做或者不做”：一个原子操作的成功和失败，都要作为一个整体的成功和失败。

参见：第 10 章，"事务"

数据并发性

多用户关系型数据库管理系统的要求是当多用户同时访问相同数据时的**并发**（**concurrency**）控制。如果没有并发控制，用户可能不适当的改变了数据，影响了**数据完整性**（**data integrity**）。例如，当不同的用户同时更新时一个用户可以更新一行数据。

如果多用户访问同一个数据，一种管理并发的方式是让用户等待。然而，数据库管理系统的目标是减少等待时间，所以不能存在等待时间或者等待时间可以忽略不计。所有的 SQL 语句必须在尽可能少的干扰下对数据进行修改。不正确地更新数据或改变底层数据结构的相互作用是一种破坏性的相互作用，这种相互作用必须避免。

Oracle 数据库使用锁来控制并发访问数据。**锁**（**lock**）是一种机制来防止多个事务之间破坏性的访问共享资源。锁在允许最大并发访问数据的同时帮助确保数据的完

整性。

参见:"Oracle 数据库锁机制概述" 在 9-11 页

数据一致性（Data Consistency）

在 Oracle 数据库中，每个用户都必须看到数据的一致视图，包括自己改变的事务和其他用户提交的事务。

例如，数据库必须防止脏读，脏读指当一个事物看到另一个并发事务未提交的变化。Oracle 数据库总是强制的**语句级读一致性**（**statement-level read consistency**），保证从单行查询返回的数据是已提交的，并且遵循在同一时间点的一致性。根据事务的隔离级别，这个时间点是在语句被打开或事务开始的时间。闪回查询功能可以使用户明确这个时间点。

数据库还可以提供在一个事务中的所有查询的读一致性，这被称为**事务级读一致性**（**transaction-level read consistency**）。在这种情况下，事务中的每个语句在同一时间点看到数据，这个时间点也是事务开始的时间。

参见:

- **第 9 章,"数据并发性和一致性"**
- ***数据库高级应用开发指南*** 中关于闪回查询

Oracle 数据库架构

数据库服务器（**database server**）是信息管理的关键。通常情况下，在多用户环境中**服务器**（**server**）可靠地管理大量的数据，使用户可以同时访问相同的数据。数据库服务器还可以防止未经授权的访问并提供有效的故障恢复解决方案。

数据库和实例（Database and Instance）

Oracle 数据库服务器由**数据库**（**database**）和至少一个数据库**实例**（**instance**）（俗称单**实例**（**instance**））。因为实例和数据库连接的如此紧密，所以术语 **Oracle 数据库**（**Oracle database**）通常指实例和数据库两者。严格的讲，这些词汇有以下含义：

- 数据库

  数据库是在磁盘中存储数据的文件集合。这些文件可以独立存在于一个数据库实例中。

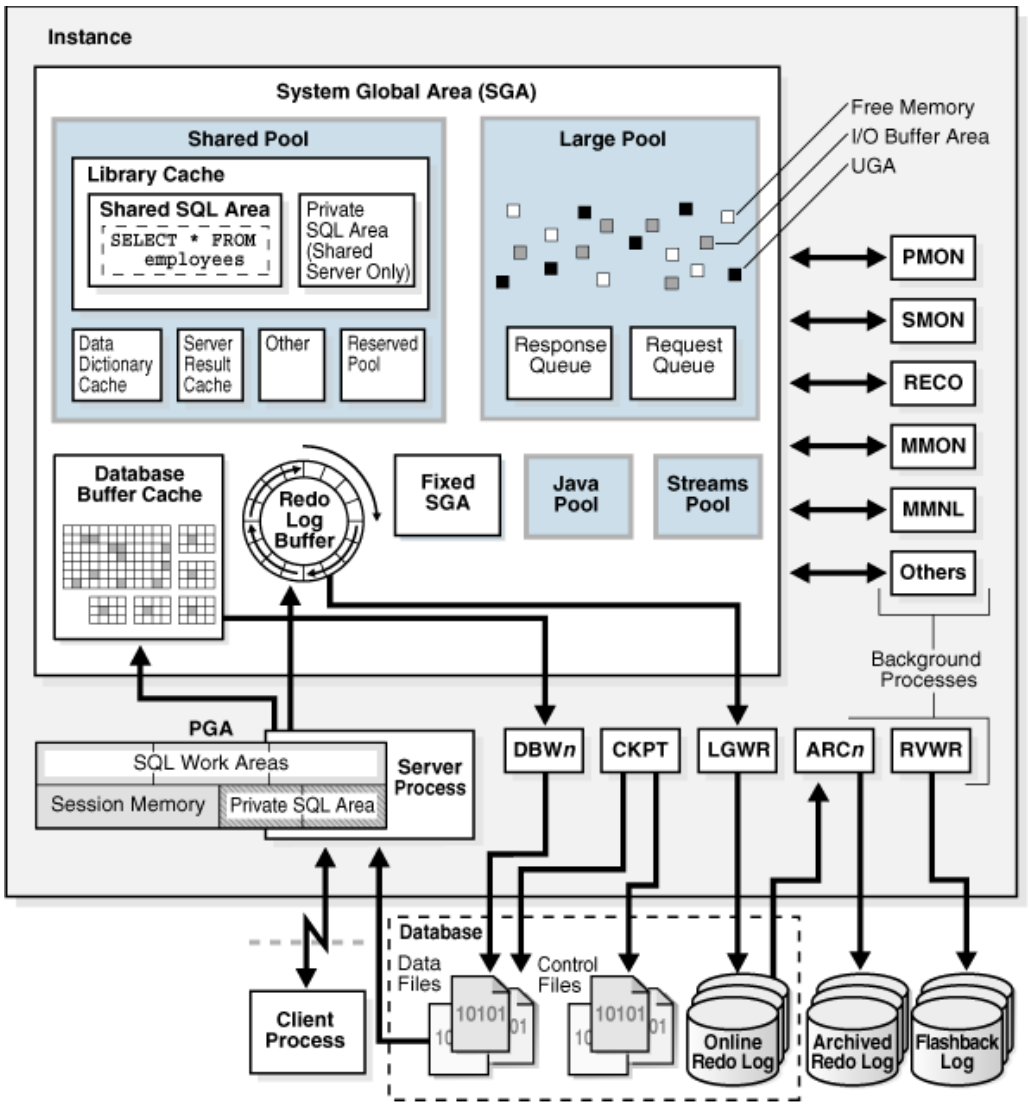
- 数据库实例

  实例是管理数据文件的内存结构的集合。实例由被称作**系统全局区**（**system global area**）**(SGA)**的共享内存区和**后台进程**（**background processes**）集合构成。实例可以独立的存在于数据库文件中。

图 1–1 展示了一个数据库和它的实例。应用在**客户端进程**（**client process**）中运行，可以让每个用户连接到实例。每个客户端进程都与它自己的服务器进程相关。服务器进程都有自己的私有会话内存，称为**程序全局区**（**program global area**）**(PGA)**。

Oracle 数据库架构

图 1–1 Oracle 实例和数据库



数据库可以从物理和逻辑两方面考虑。物理数据是在操作系统级别的数据可视。例如，Linux 操作系统工具 ls 和 ps 命令可以列出数据文件和进程。逻辑数据对于数据库才是仅有意义的，例如表。SQL 语句可以列出数据库中得表，但操作系统工具不行。

数据库包含**物理结构**（**physical structures**）和**逻辑结构**（**logical structures**）。因为物理结构和逻辑结构是分离的，管理物理存储可以不影响访问逻辑存储结构。

例如，重命名一个数据库物理文件不会重命名在这个文件中存储的表。

参见：第 13 章，"Oracle 数据库实例"

数据库存储结构

关系型数据库的基本任务是存储数据。本节简要介绍了 Oracle 数据库所使用的物理和逻辑存储结构。

物理存储结构

物理数据库结构是一些存储数据的文件。当用户执行 SQL 命令 CREATE DATABASE 时，将会建立以下文件：

- 数据文件

  每一个 Oracle 数据库都有一个或多个物理数据文件（**data files**），其中包括了数据库中的所有数据。数据库的逻辑数据结构，诸如表和索引也以物理方式存储在数据文件中。
- 控制文件

  每一个 Oracle 数据库都有控制文件（**control file**）。控制文件包含元数据指定的数据库的物理结构，包括有数据库名称、数据库文件的名称和存储位置。
- 在线重做日志文件

  每一个 Oracle 数据库都有在线重做日志（**online redo log**），包含两个或多个在线重做日志文件（**online redo log files**）的集合。在线重做日志（**redo log**）由重做条目（或者称作重做记录（**redo records**））构成，其中记录了所有数据的改变。

许多其它的文件是 Oracle 数据库服务器功能的重要文件。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（**archived redo log files**）是重要的备份和恢复的离线文件。

参见：第 11 章, "物理存储结构"

逻辑存储结构

本节讨论逻辑存储结构。下面的逻辑存储结构使 Oracle 数据库可以细粒度控制磁盘空间的使用：

- 数据块

  在最佳的粒度级别上，Oracle 数据库的数据被存储在数据块中。一个数据块（**data block**）对应磁盘上特殊的字节数量。
- 区（扩展 Extents）

  区（**extent**）是逻辑上连续的几个数据块，单独进行分配，用来存储特定类型的信息。
- 段

  段（**segment**）是区的集合，用来分配给用户对象（例如，表或索引），撤销数据（**undo data**）或者临时数据。
- 表空间

  数据库分配到的逻辑存储单元叫表空间（**tablespaces**）。表空间是段的逻辑上的容器。每个表空间包含至少一个数据文件。

参见：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。所有内存结构存在于计算机的主内存中，构成了关系型数据库管理系统。

当应用程序连接到 Oracle 数据库时，它们都连接到数据库实例。实例服务应用程序分配其它内存区域增加到 SGA 区，并启动其它进程增加到后台进程中。

Oracle 数据库进程

进程（**process**）是操作系统内可以运行一系列步骤的机制。一些操作系统会使用像工作（*job*）、任务（*task*）或线程（*thread*）这样的专有名词。这里讨论的目的，一个线程相当于一个进程。Oracle 数据库实例有以下进程类型：

- 客户端进程

  这些进程的创建和维护用来运行应用程序或 Oracle 工具的软件代码。大多数环境中有单独的计算机运行这些客户端进程。

- 后台进程

  这些进程的巩固某些功能，否则它们会被每个客户端进程的多个运行的 Oracle 数据库程序处理。（These processes consolidate functions that would otherwise be handled by multiple Oracle Database programs running for each client process.）后台进程异步执行输入输出（I/O）并监控其他 Oracle 数据库进程来提高并行，从而有更好的性能和可靠性。

- 服务进程

  这些进程与客户端进程进行沟通并和 Oracle 数据库交流进行请求。

Oracle 进程（**Oracle processes**）包括服务器进程和后台进程。在大多数环境中，Oracle 进程和客户端进程运行于不同的计算机上。

参见：第 15 章,"进程结构"

实例的内存结构

Oracle 数据库创建并使用内存结构的目的，如程序代码的内存，用户之间共享数据，每个连接用户的私有数据区。以下是一个实例相关的内存结构：

- 系统全局区(SGA)

  SGA 是一组共享内存结构，包括一个数据库实例的数据和控制信息。一个 SGA 组件的例子包括数据块高速缓存和共享 SQL 区。

- 程序全局区(PGA)

  PGA 是一种内存区域，包括服务器或后台进程的数据和控制信息。访问 PGA 是使用独占式进程每个服务器进程和后台进程有它自己的 PGA。

参见：第 14 章,"内存架构"

应用程序和网络架构

要充分利用已有计算机的系统或网络，Oracle 数据库可以分开处理数据库服务器和客户端程序。当运行应用程序的计算机处理解释并显示数据时，运行 RDBMS 的计算机担当处理数据库服务的责任。

应用程序架构

应用程序架构（**application architecture**）指一个数据库应用连接到 Oracle 数据库的计算环境。两种最常见的数据库架构是客户端/服务器架构和多层架构。

在**客户端/服务器架构**（**client/server architecture**）中，客户端应用程序启动一个请求，在服务器端执行这个操作。服务器运行 Oracle 数据库软件并处理功能请求的并发和共享数据访问。服务器接收并处理从客户端发来的请求。

在传统的**多层体系结构**（**multitier architecture**）中，一个或多个应用服务器执行部分操作。**应用程序服务器**（**application server**）包含大部分应用逻辑，为客户端提供数据访问，并执行一些查询处理，从而减少数据库负荷量。应用服务器可以在客户端和多层数据库之间像接口一样提供服务，并且提供另外的安全级别。

面向服务架构（**Service-oriented architecture (SOA)**）是在**服务**（**services**）中封装了有某些功能的应用程序的多层架构。SOA 服务通常是网页服务（Web Service）实现的。Web Service 可以通过 HTTP（超文本传送协议）访问并基于像 Web Service 描述语言（Web Services Description Language (WSDL)）和 SOAP（简单对象访问协议）这样的 XML 标准。

Oracle 数据库可以在一个传统的多层或 SOA 环境中作为 Web Service 的提供者。

参见:

- **"多层体系结构概述"** 在 16-3 页
- *Oracle XML DB 开发者手册* 关于使用带有数据库的 Web services 的更多信息

网络架构

Oracle 网络服务（**Oracle Net Services**）是数据库和网络通信协议的接口，可以促进**分布式处理**（**distributed processing**）和分布式数据库。通信协议定义了数据传输和网络中的接收方式。Oracle 网络服务支持所有主要的网络协议通信，包括 TCP/IP, HTTP, FTP 和 WebDAV。

Oracle 网络（**Oracle Net**），是 Oracle 网络服务的一个组成部分，从客户端应用程序到数据库服务器之间建立和维护网络会话。在网络会话建立后，Oracle 网络为客户端应用程序和数据库服务器扮演着数据快递者的角色，交换它们之间的信息。Oracle 网络可以执行这些工作，因为它位于网络中的每个计算机中。

网络服务的一个重要的组件是 **Oracle 网络监听器**（**Oracle Net Listener**）(称为**监听**（**listener**）)，这是一个单独运行于数据库服务器或其他网络的进程。客户端应用程序可以发送连接请求到监听，监听管理这些请求到数据库的流量。当建立连接时，客户端和数据库将直接通信。

配置 Oracle 数据库服务于客户端请求的常见方法是：

- 专用服务器架构

  每个客户端连接到一个专用服务器进程。服务器进程不与其他任何持续的客户端会话共享。每个新的会话会分配一个专用服务器进程。

- 共享服务器架构
  数据库为多会话使用共享进程池。客户端进程和**调度器**（**dispatcher**）进行通信，调度器允许许多的客户端连接到相同的数据库实例而不需要专有服务进程为每个客户端分配。

参见:

- **"Oracle 网络架构概述"** 在 16-5 页
- *Oracle 数据库网络服务管理员指南* 了解更多关于 Oracle 网络架构
- *Oracle XML DB 开发者指南* 有更多关于和数据库一起使用的 WebDAV 的信息

Oracle 数据库文档路线图

本节解释应该如何阅读这本手册并且该手册作为 Oracle 数据库文档的全局概览。
作为一位新用户，Oracle 数据库文档库会令人望而生畏。不仅是它拥有超过 175 本手册，而且许多的手册都超过了几百页的长度。但是这些文档被设计成了可以使用特殊访问路径，来确保用户尽可能的找到他们需要的信息。
这个文档集分为三层或者部分：基础、中级和高级。用户开始使用基础部分的手册(*Oracle 数据库 2 日速成 DBA*，*Oracle 数据库 2 日速成开发指南*，或本手册)，接着使用中级部分（2 日速成+系列），最终到高级手册，其中包括文档集的其余部分。

基础部分

一位新的Oracle数据库技术人员从头到尾阅读一个或多个基础部分的手册作为开始。在这个部分中的每本手册被设计成可以在 2 天内阅读完。除本手册外，基础部分包括：

- *Oracle 数据库 2 日速成 DBA*
  这本手册是让基础任务的DBA可以快速开始，教你如何执行日常数据库管理任务。它会教你如何执行常见保持数据库运行的管理任务，包括如何执行基本故障排除和性能监测活动。
- *Oracle 数据库 2 日速成开发者指南*
  这本手册是基于任务的数据库开发人员的快速开始指南，介绍了如何通过 SQL 和 PL/SQL 使用 Oracle 数据库的基本特征。

在基础部分中的手册都是紧密联系的，这主要体现在一些交叉引用方面。例如，*Oracle 数据库概念* 多次指引用户到 *2 日速成* 手册来了解在概念的基础上如何执行一个任务。*2 日速成* 手册也经常引用 *Oracle 数据库概念* 关于一个任务的概念背景。

中级部分

从基础部分开始，下一步是中级部分。这部分手册的前缀字是 *2 日+* 因为它们的范

围已扩大并包含 *2 日速成* 手册中的信息。这些手册相比基础部分涵盖了更深入的专题，或者涵盖了特别有益的专题。像表 1-1 所示， *2 日+* 手册分为 DBA 手册和开发人员手册。

表 1–1 中级部分: 2 日+ 指南

数据库管理员	数据库开发者
Oracle 数据库 2 日+ 性能调优指南	*Oracle 数据库 2 日+ 应用快速开发者指南*
Oracle 数据库 2 日+ 真实应用集群指南	*Oracle 数据库 2 日+ Java 开发者指南*
Oracle 数据库 2 日+ 数据仓库指南	*Oracle 数据库 2 日+ .NET 开发者指南（Microsoft Windows）*
Oracle 数据库 2 日+ 数据复制和集成指南	*Oracle 数据库 2 日+ PHP 开发者指南*
Oracle 数据库 2 日+ 安全指南	

高级部分

从中级部分到下一步是高级部分。这些手册目的是为需要关于特定主题、比 *2 日+* 手册多更多细节信息的专家级用户编写的。在高级部分中必要的参考手册包括：

- *Oracle 数据库 SQL 语言参考*

  这本手册是有关 Oracle SQL 的权威信息来源。

- *Oracle 数据库参考*

  该手册是有关初始化参数、数据字典和动态性能视图的权威信息来源。

高级指南有太多数量不能在这里一一列举。表 1-2 列出了主要的专家 DBA 和开发人员在同一时间或其他时间使用的指南。

表 1–2 高级部分

数据库管理员	数据库开发者
Oracle 数据库管理员指南	*Oracle 数据库高级应用开发者指南*
Oracle 数据库性能调优指南	*Oracle 数据库 PL/SQL 语言参考*
Oracle 数据库备份和恢复用户指南	*Oracle 数据库 PL/SQL 包和类型参考*
Oracle 真实应用集群管理和部署指南	

其它高级手册提供给一些特别依赖于某种职责领域的用户。例如，安全专员自然会参考 *Oracle 数据库安全指南*

Oracle 数据库文档路线图

部分 1

Oracle 关系型数据结构

这部分介绍了 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。

这部分包含以下章节：

- 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，其他模式对象"
- 第 5 章,"数据完整性"
- 第 6 章,"数据字典和动态性能视图"

2

表和表簇

本章提供了模式对象的简介，并讨论最常见的模式对象——表。

本章包含以下部分：

- 模式对象介绍
- 表的概述
- 表簇的概述

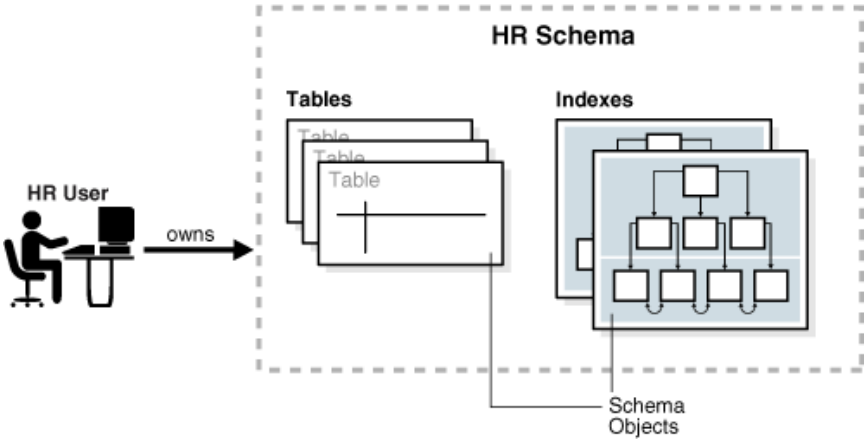
模式对象介绍

数据库的**模式**（**schema**）是一种数据结构的逻辑容器，也被称为**模式对象**（**schema objects**）。表和索引就是模式对象的例子。模式对象的创建和操纵需要使用 **SQL**。

数据库用户（**database user**）拥有一个密码和各种数据库的**权限**（**privileges**）。每个用户拥有一个模式，它和用户名具有相同的名称。这个模式包含属于用户自己模式的数据。例如，hr 用户拥有 hr 模式，其中包含模式对象，像 employees 表这样。在生产数据库中，模式所有者通常代表一个数据库应用程序而不是一个人。

在一个模式中，每个特别类型的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 描述了一个名叫 hr 的模式所有者和 hr 模式的模式对象。

图 2–1 HR 模式



Oracle 数据库文档路线图

参见:"数据库安全概述" 在 17-1 页了解更多的用户和权限

模式对象类型

关系型数据库中最重要的模式对象就是表。一张**表**（**table**）将数据存储于行中。Oracle SQL 使您能够创建和操纵许多其他类型的模式对象，包括以下：

- 索引

  索引是一种模式对象，它包括已经被索引的表或**表簇**（**table cluster**）的每一行并提供直接、快速的访问行。Oracle Database 支持多种类型的索引。索引组织表示是在索引结构中存储数据的表。见**第 3 章，“索引和索引组织表”**。

- 分区

  分区是大块的表和索引。每个分区都拥有自己的名称并可以选择它自己的存储特性。见**“分区概述”** 在 4-1 页。

- 视图

  视图是在一张或多张表或者其他视图中自定义的显示其中的数据。你可以把它们视为被存储好的查询。视图并不真正的包含这些数据。见**“视图概述”**在 4-12 页。

- 序列

  序列是用户创建的对象，可以由多个用户共享来生成整数。通常情况下，序列用来生成**主键**（**primary key**）。见**“序列概述”** 在 4-20 页。

- 维度

  维度在双列集之间定义父子关系，这个双列集的所有列必须来自同一张表。维度经常被用于分类数据，像客户、产品和时间。见**“维度概述”** 在 2-22 页。

- 同义词

  同义词是另一个模式对象的别名。因为同义词是一个简单的别名，它要求定义在没有存储其它定义的**数据字典**（**data dictionary**）中。见**“同义词概述”**在 4-22 页。

- PL/SQL 子程序和包

PL/SQL 是 Oracle SQL 过程扩展。**PL/SQL 子程序**（**PL/SQL subprogram**）被称为 PL/SQL 块，可以使用一组参数调用。**PL/SQL 包**（**PL/SQL package**）组织逻辑方面相关的 PL/SQL 类型、变量和子程序。见**“PL/SQL 子程序”**在 8-3 页和**“PL/SQL 包”** 在 8-6 页。

其它类型的对象也存储在数据库中，并且可以用 SQL 语句进行创建和操纵，但 SQL 语句并不包含在模式中。这些对象包含数据库用户、**角色**（**roles**）、**上下文**（**contexts**）和**字典对象**（**directory objects**）。

参见：

- *Oracle 数据库 2 日速成 DBA* 和 *Oracle 数据库管理员指南* 来学习如何管理模式对象
- *Oracle 数据库 SQL 语言手册* 来了解更多关于模式对象和数据库对象

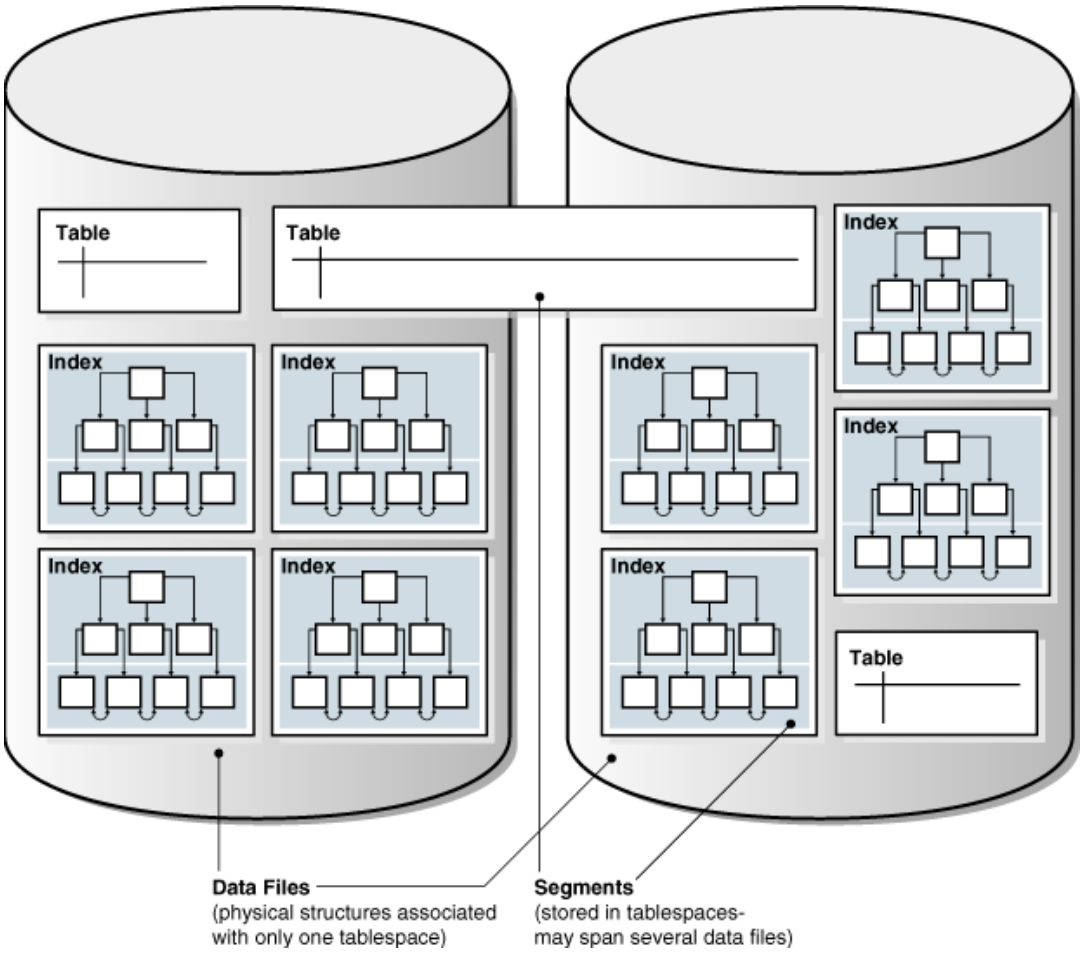
模式对象存储

有些模式对象在被称作**段**（**segments**）的逻辑存储结构中保存数据。例如，一个未分配的**堆组织表**（**heap-organized table**）或者一个索引建立的段。其他模式对象，像视图和序列，包括元数据。本节仅描述带有段的模式对象。

Oracle 数据库使用**表空间**（**tablespace**）来以逻辑方式存储模式对象。在模式和表空间之间没有任何关系：一个表空间可以包含不同模式的对象，并且一个模式的所有对象可以存在于不同的表空间中。每个对象的数据被使用物理方式存储于一个或多个数据文件中。

图 2–2 展示了一种可以配置表和索引的段、表空间和数据文件。一个表的数据段跨越两个数据文件，这两个文件都是相同表空间中的一部分。一个段不能跨越多个表空间。

图 2–2 段、表空间和数据文件



Oracle 数据库文档路线图

参见:

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

模式对象依赖关系

一些模式对象会引用其它对象，这就建立了**模式对象依赖**（**schema object dependencies**）。例如，一个视图的**查询**（**query**）引用了表或其他视图，或者当一个 **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 是遵循于 B 的一个**依赖对象**（**dependent object**）并且 B 是遵循于 A 的一个**引用对象**（**referenced object**）。

Oracle 数据库提供了一种自动机制，以确保依赖对象引用的对象总是最新。当创建完一个依赖对象时，数据库将跟踪依赖对象和引用对象的依赖关系。当一个被引用对象发生改变会影响依赖对象时，依赖对象会被标记成无效。例如，用户删除一张表，任何基于这张表的视图都是不可用的。

无效的依赖对象在它是可用状态之前，它必须根据新定义的引用对象被重新编译。当无效的依赖对象被引用时，重编译会自动发生。

作为一个模式对象如何创建依赖关系的实际例子，下面的示例脚本创建了一张表

test_table，然后用一个存储过程查询这张表：

```
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;
/
```

接着查询存储过程 test_proc 的状态显示是有效的：

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

在 test_table 增加 col3 列后，存储过程仍然有效，因为存储过程在此列上没有依赖关系：

```
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
```

然而，改变 test_proc 存储过程依赖的列 col1 的数据类型，会使存储过程无效：

```
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
```

运行或重编译这个存储过程会使它重新变为有效，如下面的例子所示：

```
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
```

***参见:** Oracle 数据库管理员指南* 和 *Oracle 数据库高级应用开发者指南* 来了解如何管理模式对象依赖关系

SYS 和 SYSTEM 模式

所有 Oracle 数据库包括默认管理账户。管理账户是有高度特权并且仅适于给 DBA 授权来执行像启动和停止数据库，管理内存和存储，建立和管理数据库用户等等这样的任务。

当数据库建立时，管理账户 SYS 将被自动创建。这个账户可以执行所有数据库管理的功能。SYS 模式存储**数据字典**（**data dictionary**）的基表和视图。这些基表和视图是操作 Oracle 数据库的关键。SYS 模式中的表只能由数据库操纵，绝不能被其他用户修改。

当创建数据库时，SYSTEM 账户也自动被创建。SYSTEM 模式存储额外的表和视图可以显示管理信息，并且各种 Oracle 数据库选项和工具会使用内部的表和视图。不要使用 SYSTEM 模式存储其它非管理用户的表。

参见:

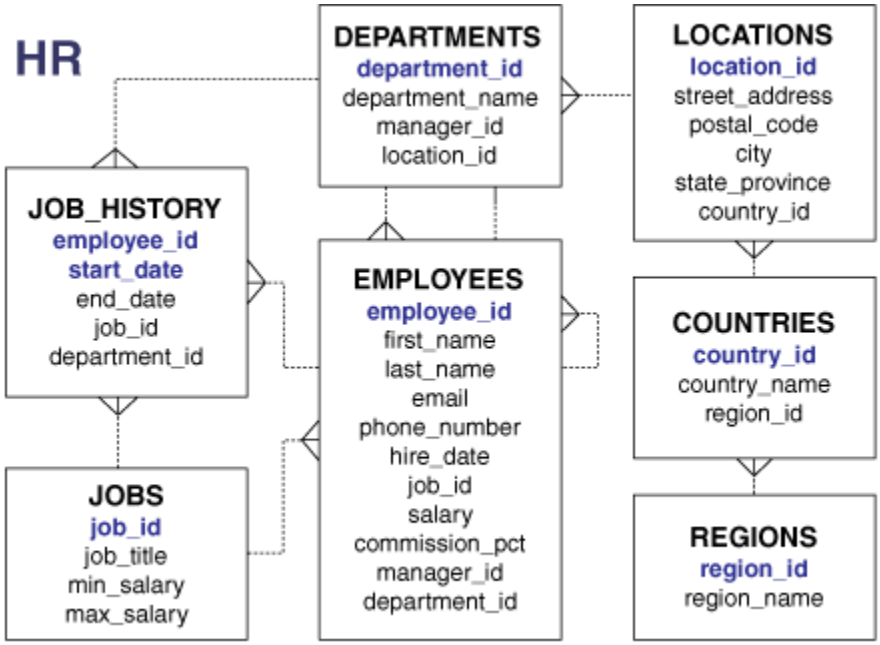
- **"用户账户"** 在 17-1 页 和 **"使用管理员权限连接"** 在 13-6 页
- *Oracle 数据库 2 日速成 DBA* 和 *Oracle 数据库管理员指南* 来了解更多关于 SYS, SYSTEM, 和其他管理账户

示例模式

Oracle 数据库可能包含**示例模式**（**sample schemas**），示例模式是一组相互关联的模式使 Oracle 文档和 Oracle 教材展示常见的数据库任务。hr 模式是一种简单模式包含员工、部门、地点和工作经历等等。

图 2–3 是在模式中表的实体关系图。这本手册的大多数例子使用这个模式中的对象。

图 2–3HR 模式



参见: Oracle 数据库示例模式

表的概述

在 Oracle 数据库中，**表**（**table**）是数据组织的基本单元。表描述**实体**（**entity**），这是关于记录信息最重要的东西。例如，一位职工可以是一个实体。

Oracle 数据库的表下降到以下基本级别：

- 关系表

  关系表有简单列并且是最普通的表类型。在 2-8 页的**例子 2-1** 展示了关系表的 CREATE TABLE 语句。

- 对象表

  列对应**对象类型**（**object type**）的顶级属性。见**“对象表”**在 2-15 页。

你可以使用以下组织的特点创建关系表：

- **堆组织表**（**heap-organized table**）不在任何特定的顺序中存储行。CREATE TABLE 语句默认创建对组织表。
- **索引组织表**（**index-organized table**）根据主键值来指定行。对于某些应用，索引组织表提高性能并有效的使用磁盘空间。见**“索引组织表概述”**在 3-20 页。

- **外部表**（**external table**）是只读表，它的元数据存储在数据库中，但数据存储在数据库外部。见**“外部表”**在 2-16 页。

表可以是**永久的**（**permanent**）或者**临时的**（**temporary**）。永久表的定义和数据支持通过会话。**临时表**（**temporary table**）定义支持与永久表相同的定义方式，但数据仅用户**事务**（**transaction**）和**会话**（**session**）存在期间。在应用程序中，临时表十分有用。可能因为结果是由多个操作构成的，因此结果集必须临时性被持有。

本节包含以下主题：

- 列和行
- 例子：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
- 表压缩

> **参见:** *Oracle 数据库 2 日速成 DBA* 和 *Oracle 数据库管理员指南* 来了解如何管理表

列和行

表的定义包括**表名**（**table name**）和列集。通过表中的**列**（**column**）来标识实体描述的属性。例如，在 employees 表中的 employee_id 列是指雇员实体的雇员 ID 属性。

一般情况下，当你创建一张表时你会给每个列一个**列名**（**column name**）、一个**数据类型**（**data type**）和一个**长度**（**width**）。例如，employee_id 的数据类型是 NUMBER(6)，表明这个列只能包含数值长度最大是 6 个数字的数据。通过数据类型可以定义长度，如同 DATE 类型。

表可以包含**虚拟列**（**virtual column**），这不像非虚拟列不占用磁盘空间。在需要的虚拟列上数据库通过计算一组用户指定的**表达式**（**expressions**）或函数来派生值。例如，虚拟列 income 可以是 salary 列和 commission_pct 列的函数。

在创建表之后，你可以使用 SQL 来插入、查询、删除和更新行。**行**（**row**）是表中列信息对应记录的集合。例如，在 employees 表中行描述一个特定员工的属性。

> **参见:** *Oracle 数据库管理员指南* 来了解如何管理虚拟列

例子：CREATE TABLE 和 ALTER TABLE 语句

Oracle SQL 的建表命令是 CREATE TABLE。**例 2-1** 展示了 CREATE TABLE 语句在 hr 示例模式中创建 employees 表。该语句指定了列，如 employee_id, first_name, 等等，为每一列指定了数据类型，像 NUMBER 或 DATE。

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) ) ;
```

例 2–2 展示了 ALTER TABLE 语句添加**完整性约束**（**integrity constraints**）到 employees 表中。完整性约束强制业务规则并防止无效信息进入到表中。

例 2–2ALTER 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
     ) ;
```

例 2–3 展示了 hr.employees 表的 8 行 6 列

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	

在**例 2-3** 中的输出说明了以下表、列和行的重要特性：

- 表的行描述一个员工的属性：名字、工资、部门等等。例如，输出中第一行显示的名叫 Steven King 雇员的记录。
- 列描述员工的一个属性。在这个例子中，employee_id 列是**主键**（**primary key**），这意味着每一名员工的员工 ID 都是唯一标识的。保证任意两名员工不会有相同的

员工 ID。

- 非键列可以包含相同值得行。在这个例子中，员工 101 和员工 102 的薪资同样是：17000。
- **外键**（**foreign key**）列是指主键或唯一键在相同表或不同表中。在这个例子中，在 department_id □ 值是 90 对应的 departments 表的 department_id 列。
- **字段**（**field**）是行和列的交叉点。它只能包含一个值。例如，104 员工的部门 ID 字段包含值是 60。
- 字段中可以缺少值。在这种情况下，可以说这个字段包含一个**空**（**null**）值。员工 100 的 commission_pct 列值是空，而员工 149 的列值是. 2。允许一个列为空，除非 NOT NULL 或者主键完整性约束定义在这个列上，这种情况下没有该列的值将不能插入这一行。

参见: *Oracle 数据库 SQL 语言手册* 见 CREATE TABLE 语法和语义

Oracle 数据类型

每个列都有一个**数据类型**（**data type**），它和具体的存储格式、约束还有有效值范围都有关系。值的数据类型关联着一组带有值的固定属性。这些属性会导致 Oracle 数据库处理一种数据类型值不同于其它值。例如，你可以用 NUMBER 数据类型的值相乘，而不是 RAW 数据类型的值。

当你创建表时，你必须指定它每一列的数据类型。随后在列中插入的每个值采用列的数据类型。

Oracle 数据库提供了一些内置数据类型。最常用的数据类型分为以下几类：

- **字符数据类型**
- **数值数据类型**
- **日期时间数据类型**
- **Rowid 数据类型**
- **格式模型和数据类型**

其它重要的内置类型分类包括 raw、大对象（large objects (LOBs)）和集合。PL/SQL 中有常量和变量的数据类型，其中包括布尔（BOOLEAN）、引用类型复合类型（记录）和用户自定义类型。

参见:

- **"LOBs 概述"** 在 19-12 页
- *Oracle 数据库 SQL 语言手册* 来了解关于内置 SQL 数据类型
- *Oracle 数据库 PL/SQL 语言手册* 来了解关于 PL/SQL 数据类型

- *Oracle 数据库高级应用开发者指南* 关于如何使用内置数据类型

字符数据类型

字符数据类型存储字符（字母）数据在字符串中。最常见的字符数据类型是 VARCHAR2，这是最有效的用于存储字符数据的选项。

字节值对应字符编码模式，一般被称为**字符集**（**character set**）或**代码页**（**code page**）。在创建数据库时数据库就建立字符集。字符集如 7 位 ASCII、EBCDIC 和 Unicode UTF-8 等等这些例子。

字符数据类型的语义长度能使用字节或字符度量。**字节语义**（**Byte semantics**）把字符串作为字节序列处理。默认对字符数据类型做这种处理。**字符语义**（**Character semantics**）把字符串作为字符序列处理。一个字符就是数据库字符集技术方面的一个代码点。

参见:

- **"字符集"** 在 19-9 页
- *Oracle 数据库 2 日速成开发者指南* 和 *Oracle 数据库高级应用开发者指南* 来了解如何选择（select）字符数据类型。

VARCHAR2 和 CHAR 数据类型 VARCHAR2 数据类型存储可变长度字符。术语**常量**（**literal**）和**常数值**（**constant value**）是同义词并且指固定的数据值。例如，'LILA', 'St. George Island' 和 ' 101' 全部是字符常量；5001 是一个数值常量。字符常量被括在单引号标记中，以便数据库可以用模式对象名称区分它们。

注意: 本手册使用**文本常量**（**text literal**），**字符常量**（**character literal**）和**字符串**（**string**）这几个术语时会互换。

当你使用 VARCHAR2 列创建表时，你可以指定最大字符串长度。在**例子 2-1** 中，last_name 列有一个 VARCHAR2(25) 的数据类型，这意味着存储在这列中的任何名字可以最多有 25 个最大字符。

对于每一行，Oracle 数据库存储在一个可变长度字段中存储每个值，除非值超过最大长度，这种情况下数据库会返回一个错误。例如，在单字节字符集中，如果你在一行中的 last_name 列输入 10 个字符，那么这一行中的该列只存储 10 个字符（10 字节），而不是 25。使用 VARCHAR2 可以减少空间的消耗。

相比于 VARCHAR2，CHAR 存储固定长度的字符串。当你创建带有 CHAR 列的表时，该列需要字符串长度。默认值是 1 字节。数据库使用空白值来填充到指定长度。

Oracle 数据库比较 VARCHAR2 的值使用非填充式比较语义，而比较 CHAR 值使用填充式比较语义。

参见: *Oracle 数据库 SQL 语言手册* 了解详细的关于填充和非填充比较语义

NCHAR 和 NVARCHAR2 数据类型 NCHAR 和 NVARCHAR2 数据类型存储 Unicode 字符数据。**Unicode** 是一种通用编码字符集，可以在任何语言中使用一种字符集存储信息。NCHAR 存储固定长度字符串来对应国际字符集，而 NVARCHAR2 存储可变长度字符串。

当创建数据库时，你需要指定一个国家字符集。NCHAR 和 NVARCHAR2 数据类型的字符集必须都是 AL16UTF16 或者 UTF8。这两个字符集都使用 Unicode 编码。

当你使用 NCHAR 或者 NVARCHAR2 列创建表时，最大的大小总是在字符长度语义中。字符长度语义是默认的，并且仅有 NCHAR 或者 NVARCHAR2 的长度语义。

> **参见:** *Oracle 数据库全球化支持指南* 有关 Oracle 全球化支持功能的信息

Numeric 数据类型

Oracle 数据库数字数据类型存储固定和浮点数、零和无穷大。还有些数值类型存储的值是操作中的未定义结果，这被称为“不是一个数字”或者 NAN(not a number)。

Oracle 数据库用可变长度格式存储数值数据。每个值使用科学计数法存储，并使用 1 个字节存储指数。数据库使用至 20 字节来存储尾数（**mantissa**），它是浮点数的一部分包含了有效数字。Oracle 数据库不存储起始和尾随零。

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大小的数字。可以保证运行于 Oracle 数据库中的这些数据能在不同的操作系统之间进行迁移。当你必须存储数字数据时，多数情况下建议使用 NUMBER 数据类型。

你可以使用 NUMBER(p, s) 格式指定定点数字，这里的 p 和 s 是指以下特点：

- 精度

  精度（precision） 指数字的总个数。如果没有指定精度，那么该列存储的值和应用程序提供的一样，不带有任何舍入。

- 规模

  规模（scale）指从小数点到最少有效数字的数字个数。**正规模（Positive scale）**从小数点的右侧起计数并包含最少的有效数字。**负规模（Negative scale）**从小数点的左侧起计数但不包含最少的有效数字。如果你指定的精度没有规模，如 NUMBER(6)，那么它的规模是 0。

在例 2–1 中，salary 列的类型是 NUMBER(8, 2)，这样的精度是 8 且规模是 2。因此，数据库中存储年薪 100,000 为 100000.00。

浮点数（Floating-Point Numbers）Oracle 数据库为浮点数单独提供两种数字的数据类型：BINARY_FLOAT 和 BINARY_DOUBLE。NUMBER 数据类型提供这些数据类型支持的所有的基本功能。然而，NUMBER 使用十进制精度，而 BINARY_FLOAT 和 BINARY_DOUBLE 使用二进制精度，这可以实现更快的运算并降低存储需求。

BINARY_FLOAT 和 BINARY_DOUBLE 是近似数值数据类型。它们存储近似表示的十进制值，而不是确切的表示。例如，值 0.1 不能由 BINARY_DOUBLE 或 BINARY_FLOAT 准确的表示。它们通常用于科学计算。它们的行为类似于 Java 和 XML 模式中的数据

类型 FLOAT 和 DOUBLE。

参见：*Oracle 数据库 SQL 语言参考* 来学习关于精度、规模和其它数值类型的特点。

日期时间数据类型（Datetime Data Types）

日期时间（datetime）数据类型是 DATE 和 TIMESTAMP。Oracle 数据库为时间戳提供了全时区支持。

日期数据类型（DATE Data Type）DATE 数据类型存储日期和时间。虽然日期时间可以用字符或数字数据类型表示，但是 DATE 有特殊的关联性。例 2–1 中的 hire_date 列是 DATE 数据类型。

数据库内部将日期存储为数字。日期都存储在每个 7 个字节固定长度的字段中，对应世纪、年、月、日、小时、分钟和秒。

注意：日期完全支持算术运算，所以从日期中可以用数字相加减。

参见 *Oracle 数据库高级应用开发者指南*

数据库根据指定的**格式模型（format model）**显示日期。格式模型是一串字符文字，它用字符串描述日期时间的格式。标准日期格式是 DD-MON-RR，用格式 01-JAN-09 显示日期。

虽然 RR 类似于 YY（年的最后两位数字），但是世纪的返回值根据指定的两位数年和当前年的最后两位数变化。假设在 1999 年数据库显示 01-JAN-09。如果日期格式使用 RR，那么 09 指 2009，而如果格式使用 YY，那么 09 指 1909。你可以在**实例（instance）**和**会话（session）**级别两者里更改默认日期格式。

Oracle 数据库用 24 小时格式—HH:MI:SS 存储时间。如果没有时间部分输入，那么在日期字段中使用默认时间 00:00:00 AM。在只有时间输入情况下，日期部分默认为当前月的第一天。

参见：

- *Oracle 数据库高级应用开发者指南* 获得更多关于世纪和日期格式掩码的信息
- *Oracle 数据库 SQL 语言参考手册* 获得关于日期时间格式编码的信息

时间戳（TIMESTAMP）数据类型 TIMESTAMP 数据类型是 DATE 数据类型的扩展。它在存储 DATE 数据类型基础上增加了秒的小数部分。TIMESTAMP 数据类型适用于存储精确的时间值，如应用必须跟踪事件的顺序。

了解 DATETIME 数据类型 TIMESTAMP WITH TIME ZONE 和 TIMESTAMP WITH LOCAL TIME ZONE 包含时区。当用户选择这种数据时，该值被调整到用户会话所在的时区。这种数据类型用于收集和评估跨地理区域的日期信息。

参见：*Oracle 数据库 SQL 语言手册* 来获得关于在时间戳列创建和输入数据语法的详细信息

Rowid 数据类型

在数据库中存储的每一行都有一个地址。Oracle 数据库用一个 ROWID 数据类型

来存储在数据库中每行的地址(**rowid**)。Rowid 分为以下类别：

- **物理 rowid（Physical rowids）**在堆组织表、表簇和表与索引分区中存储行地址。
- **逻辑 rowid（Logical rowids）**在索引组织表中存储行地址。
- **外部 rowid（Foreign rowids）**在外表中是标识符，如通过网关访问的 DB2 数据库表。它们不是标准的 Oracle 数据库 rowid。

有一种数据类型被称作**通用 rowid（universal rowid）**或者 UROWID，支持所有种类的 rowid。

Rowid 的使用（Use of Rowids）Oracle 数据库内部为建立的索引使用 rowid。**B 树索引（B-tree index）**，是最常见的类型，包含按范围分配的键的有序列表。为了快速访问，每个键和 rowid 关联且 rowid 指向关联行的地址。最终用户和应用开发者也可以为了几个重要功能使用 rowid：

- Rowid 是访问特定行最快的方法。
- Rowid 提供能了解表是如何组织的。
- Rowid 是在给定的表中，行的唯一标识符。

你也可以用 ROWID 数据类型来定义创建表的列。例如，你可以定义带有数据类型 ROWID 列的异常表来存储违反完整性约束行的 rowid。使用 ROWID 数据类型定义的列行为像其它表的列：值可以更新等等。

ROWID 伪列（ROWID Pseudocolumn）在 Oracle 数据库中每个表都有一个名叫 ROWID 的**伪列（pseudocolumn）**。虽然伪列的行为像一张表中的列，但实际上它并没有存储在表中。你可以从伪列中查询，但你不能插入、更新或删除它们的值。伪列类似于不带参数的 SQL **函数（function）**。在结果集中不带参数的函数通常会在每一行返回相同值，而伪列通常会在每行返回不同值。ROWID 伪列的值是表示每一行地址的字符串。这些字符串的数据类型是 ROWID。当执行 SELECT 或 DESCRIBE 列出表结构时，这个伪列不显示，伪列也不占用空间。但是，每行的 rowid 可以用带有像列名一样的保留字 ROWID 的 SQL 查询来检索。

例 2–4 查询 ROWID 伪列显示 employees 表中员工号为 100 的行的 rowid。

例 2–4ROWID 伪列

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

AAAPecAAFAAAABSAAA

参见：

- "Rowid 格式" 在第 12-10 页
- *Oracle 数据库高级应用开发人员指导手册* 来了解如何用地址确定行
- *Oracle 数据库 SQL 语言参考手册* 来了解关于 rowid 类型

格式模型和数据类型

格式模型（format model）是一种存在于字符串中描述日期时间或数字数据格式字符文字。在数据库中格式模型不改变数值的内部表示。

当你将字符转换到日期或者数字，格式模型将决定数据库如何解释这个字符串。在 SQL

中，你可以使用格式模型作为 TO_CHAR 和 TO_DATE 函数的参数来规定从数据库中返回的格式或者规定存储到数据库中的格式。

下面的语句选择了在 80 部门员工的薪水并使用 TO_CHAR 函数将这些薪水转换成了带有指定数字格式模型'$99,990.99'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
```

下面的例子使用带有格式掩码'YYYY MM DD'的 TO_DATE 函数来转换字符串'1998 05 20'到 DATE 类型值更新雇佣日期：

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

参见: *Oracle 数据库 SQL 语言参考手册* 来了解更多关于格式模型

完整性约束

完整性约束（Integrity constraints） 是表中限制一列或多列值的命名规则。这些规则防止无效数据输入到表中。此外，约束可以防止在表的删除时，存在一定的依赖关系。如果启用约束，那么数据库会检查进入和修改的数据。不符合约束的数据会被防止进入。如果约束被禁用，那么不符合约束的数据可以允许进入数据库。

在 2-8 页的例 2–1 中，CREATE TABLE 语句指定了 NOT NULL 约束的 last_name，email，hire_date 和 job_id 列。约束条款识别列和约束表达式。这些约束确保指定的列包含非空值。例如，尝试插入一个不带工作 ID 的新员工产生一个错误。

你可以创建一个约束在创建表的时候或者之后。如果需要，约束可以被临时禁用。数据库在**数据字典（data dictionary）**中存储约束。

参见：

- 第 5 章，"数据完整性" 来了解关于完整性约束
- *Oracle 数据库 SQL 语言参考手册* 来了解关于 SQL 约束条款

对象表

Oracle **对象类型(object type)**是带有名称、属性和方法的用户定义的类型。对象类型使它可以模拟真实世界的实体，如客户和采购订单作为对象存在数据库中。

对象类型定义逻辑结构，但不创建存储。例 2-5 创建一个叫 department_typ 的

对象类型。

例 2–5 对象类型

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

对象表(object table)是一种特定类型的表，在该表中每一行代表一个对象。在例 2-6 中语句创建一个 department_typ 对象类型的名称为 departments_ obj 的对象表。此表的属性（列）是来自于该对象类型的定义。INSERT 语句向表中插入行。

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');
```

像关系列，对象表可以包含只是一种类型的东西，即如表一样的相同声明类型的对象实例。默认情况下，对象表中每行对象都有一个相关的逻辑**对象标识符（object identifier (OID))**，在对象表中唯一识别。对象表的 OID 列是一个隐藏列。

参见：

- *Oracle 数据库对象-关系开发者指南* 来了解关于 Oracle 数据库中对象-关系的功能
- *Oracle 数据库 SQL 语言参考手册* 对于 CREATE TYPE 的语法和语义

临时表

Oracle 数据库**临时表 (temporary table)** 持有数据仅存在于一个事务或会话之间。在临时表中数据对于会话是私有的，这意味着每个会话仅能看见并修改它自己的数据。在应用中当结果集必须被缓冲时，临时表会起到了作用。例如，一个调度应用允许学院学生去创建选学

课程安排。临时表中的每一行代表每一个调度。在会话期间，调度数据是私有的。当学生决定时间表，应用移动选中时间表的行到永久表中。在会话最后，在临时数据中的调度数据被自动丢弃。

临时表的创建

CREATE GLOBAL TEMPORARY TABLE 语句创建临时表。ON COMMIT 子句指定表中数据是否指定为特定事务或者特定会话。

不像一些其他关系型数据库的临时表，当你在 Oracle 数据库中创建临时表，你可以创建一个静态表定义。临时表在数据字典中被描述成一种持久性对象，但显示为空，直到你的会话插入数据到表中。你为数据库创建它自身的临时表，而不是为每一个 PL/SQL **存储过程 (stored procedure)**。

因为临时表是静态定义的，你可以使用 CREATE INDEX 语句为它们创建索引。被创建

在临时表上的索引也是临时的。索引中的数据像临时表中的数据一样有相同的会话和事务范围。你也可以在临时表上创建**视图（view）**或者**触发器（trigger）**。

参见：

- *Oracle 数据库管理员指南* 来了解如何创建和管理临时表
- *Oracle 数据库 SQL 语言参考手册* 对于 CREATE GLOBAL TEMPORARY TABLE 的语义和语法
- "视图概览" 在 4-12 页中和 "触发器概览" 在 8-16 页中

临时表中的段分配

像永久表一样，临时表的定义在数据字典中。临时段是第一次插入数据时分配的。直到会话中加载表的数据显示为空。回收临时段是在有特定事务临时表的事务末端和特定会话临时表的会话末端。

参见： "临时段" 在 12-23 页

外部表

外部表（external table） 访问在外部源中的数据，犹如这个数据在数据库中的一个表中一样。你可以使用 SQL、PL/SQL 和 Java 来查询外部数据。

外部表用于查询平面文件。例如，一个基于 SQL 的应用程序可能需要访问在文本文件中的记录。记录的格式如下：

```
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
```

你可以创建一张外部表，复制文件到外部表定义中的指定位置，并使用 SQL 查询在文本文件中的记录。

在**数据仓库（data warehouse）**环境中外部表对执行 **ETL** 常见任务也是有价值的。例如，外部表允许数据加载阶段流水线带有转换阶段，省去了准备时数据库里处理较长的需要展示的数据库中的数据。参见“数据仓库和商业智能概述”在 17-14 页。

外部表的创建

在内部，创建外部表意味着在数据字典中创建元数据。不像一张普通表，在数据库中一张外部表既不描述存储的数据，也不描述数据是如何存储在外部的。相反的外部表的元数据描述了外部表层如何必须*呈现（present）*数据到数据库中。

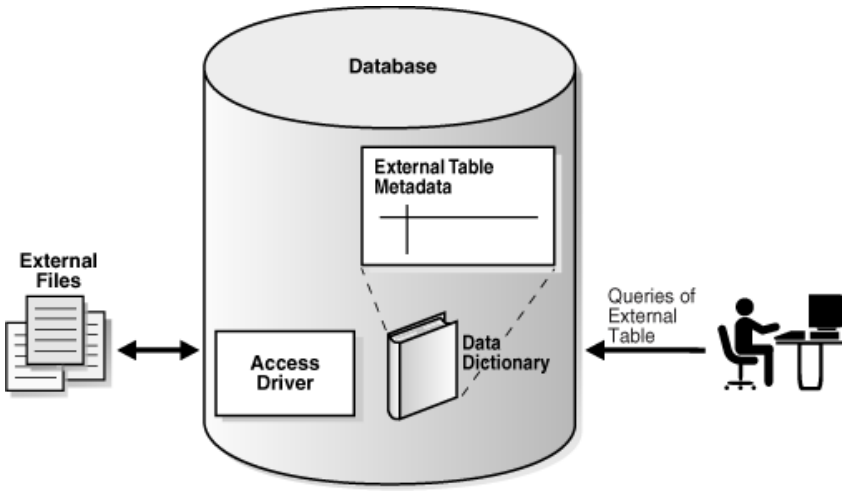
CREATE TABLE ... ORGANIZATION EXTERNAL 语句有两部分。**外部表定义（external table definition）**描述列的类型。这个定义像一个视图，允许 SQL 查询外部数据而不加载到数据库中。语句的第二部分将外部数据映射到列中。

外部表是只读的，除非用带有 ORACLE_DATAPUMP 访问驱动的 CREATE TABLE AS SELECT 语句创建。外部表上的限制包括不支持索引列，虚拟列和列对象。

外部表访问驱动

访问驱动（access driver）是一个为数据库解释外部数据的 API。访问驱动在数据库内部运行，其中用到驱动读取在外部表中的数据。访问驱动和外部表层负责执行数据文件中数据的转换需求，以便它匹配外部表的定义。图 2–4 表示如何访问外部数据。

图 2–4 外部表



Oracle 为外部表提供了 ORACLE_LOADER（默认）和 ORACLE_DATAPUMP 访问驱动。对于这两个驱动，外部文件不是 Oracle 数据文件。

ORACLE_LOADER 能够使用 SQL*Loader 只读访问外部文件。你不能使用 ORACLE_LOADER 驱动创建、更新或追加到一个外部文件。

ORACLE_DATAPUMP 驱动允许你**卸载（unload）**外部数据。这个操作涉及到从数据库中读取数据和插入数据到外部表中，由一个或多个外部文件表示。外部文件被创建之后，数据库不能向它们中更新或追加数据。驱动也允许你**加载（load）**外部数据，其中包括读取外部表和加载它的数据到数据库中。

参见：

- *Oracle 数据库管理员指南* 来了解关于管理外部表、外部连接和**目录对象（directory objects）**
- *Oracle 数据库工具* 来了解关于外部表
- *Oracle 数据库 SQL 语言参考手册* 信息关于创建和查询外部表

表的存储

Oracle 数据库在表空间中使用**数据段（data segment）**来保存表中数据。诚如在 12-21 页中的“用户段”，一个段包含由**数据块（data blocks）**组成的**扩展区（extents）**。

对于一张表的数据段（或者簇的数据段，当处理**表簇（table cluster）**时）是位于表所有者的默认表空间或者在 CREATE TABLE 语句中的表空间名称。

表的组织

默认情况下，一张表被组织为一个**堆（heap）**，这意味着数据库将行放置在比用户

指定位置更适合的地方。因此，堆组织表是一个无序的行集合。如用户添加行，数据库将行放置在数据段中的第一个可用空间。行不能保证按照它们被插入的顺序来检索。

注意：索引组织表使用不同的组织原则。参见“索引组织表概览”在 3-20 页。

---

hr.departments 表是一个堆组织表。包含 department ID、name、manager ID 和 location ID 列。如被插入的行，数据库把它们存储到适合的地方。在表段中的数据块也许包含无序行在例 2-7 中所示。

例 2–7 Departments 表中的行

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

在一张表中列的顺序是相同的。通常数据库按照在 CREATE TABLE 语句中的顺序存储列，但是不保证一定是这个顺序。例如，如果一张表有 LONG类型的列，那么 Oracle 数据库总是在行中最后存储这一列。另外，如果你在表中增加新列，那么这个新列就成为存储的最后一列。

一张表可能包含**虚拟列（virtual column）**，不像正常列那样不消耗磁盘空间。在虚拟列中数据库按照需求通过计算用户定义的表达式或函数派生值。你可以在虚拟列上索引虚拟列、收集统计信息和创建完整性约束。因此，虚拟列非常像非虚拟列。

参见：Oracle 数据库 SQL 语言参考手册 来了解关于虚拟列

行存储

数据库在数据块中存储行。一张表中的每一行都包含小于 256 列的数据于一个或多个**行片（row pieces）**。

如果可能，Oracle 数据库存储每一行为一个行片。然而，如果所有的行数据不能被插入到单个数据块中，或者如果更新一个存在的行导致增长大于数据块，那么数据库使用多块存储这种行。（参见“数据块格式”在 12-7 页）。

在表簇中的行包含像非簇表中的行一样。此外，表簇中的行包含它们属于哪一个簇键的信息。

行片的 Rowid

rowid 一个行有效的 10 字节物理地址。诚如 2-13 页中的“Rowid 数据类型”，在堆组织表中的每一行都有一个唯一对应这张表的 rowid 对应一个行片的物理地址。对于表簇，在不同表中的行在相同的数据块中可以有相同的 rowid。

Oracle 数据库内部使用 rowid 来建立索引。例如，在 B-树中的每一个键都和指向相关联行地址的 rowid 相关联来快速访问。（参见“B-树索引“在 3-5 页）。物理 rowid 提供最快可能访问表中的行，使数据库在短短的单个 I/O 中检索到一行。

参见：“Rowid 格式”在 12-10 页

空值的存储

在列中**空（null）**是值得一种情况。空表示缺少、未知或者不适用的数据。空值存储在数据库中，如果它们介于列之间的数据值。在这些情况下，它们需要 1 字节来存储列的长度（零）。在一行中尾随的空值无需存储，因为新的行头标记了剩下在之前行为空的列。例如，如果一张表的最后三列为空，那么这些列没有存储数据。

参见：*Oracle 数据库 SQL 语言参考手册* 来了解更多关于空值

表压缩

数据库可以使用**表压缩（table compression）**来减少表需要的存储容量。压缩保留磁盘空间，减少在**数据库高速缓冲区（database buffer cache）**中的内存使用，并且在某些情况下加速查询执行。表压缩对数据库应用是透明的。

基础和 OLTP 表压缩

基于字典表压缩为堆组织表提供良好的压缩比率。Oracle 数据库支持下列基于字典表压缩的类型：

- 基础表压缩

这个类型的压缩适用于大容量加载操作。数据库不使用 DML 压缩修改的数据。你必须使用直接路径加载、ALTER TABLE . . . MOVE 操作或者在线表重定义来实现基本压缩。

- OLTP 表压缩

这种类型的压缩适用于 OLTP 应用和通过任意 SQL 操作压缩操作数据。

对于基本和 OLTP 表压缩，数据库在**行-主（row-major format）**中存储压缩行。一行的所有列都存放在一起，后面跟着下一行的所有列，等等（参见 12-9 页图 12-7）。重复值被替换为一张符号表中的简短参照在块的起始位置。因此，信息需要在数据块中重新创建已存储的非压缩数据。

压缩数据块看起来很像正常数据块。大多数数据库的特点和功能是工作在正常数据块也工作在压缩块。

你可以在表空间、表、分区或者子分区级别中声明压缩。如果指定在表空间级别中，

那么在表空间中创建的所有表默认都是被压缩的。

下面的语句适用于 OLTP 压缩的 orders 表：

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

下面的部分 CREATE TABLE 语句示例指定一个分区的 OLTP 压缩和其他分区的基本压缩：

```
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 );
```

参见：

- "数据块压缩" 在 12-11 页来了解关于压缩数据块格式
- *Oracle 数据库管理员指南* 和 *Oracle 数据库性能调优指南* 来了解关于表压缩
- "SQL*Loader" 在 18-5 页来了解使用 SQL*Loader 于直接路径加载

混合列压缩

带有混合列压缩，数据库存储相同的列在一起于一组行中。数据块不存储数据在行-主格式中，但使用行和柱状联合的方法。

列数据存储在一起，具有一样的数据类型和相似的特点，极大地增加了压缩实现节省存储。数据库通过任意 SQL 操作压缩操纵的数据，尽管是对于直接路径加载更高的压缩级别。数据库对压缩对象操作工作透明，因此无需更改应用程序。

混合列压缩的类型 如果你的底层存储支持混合列压缩，那么你可以指定下列压缩类型，依赖于你的需求：

- 数据仓库压缩

  这种压缩类型优化用于节省存储空间，目的是为了数据仓库应用。

- 在线归档压缩

  这种类型的压缩优化了最大压缩级别，并且适用于历史数据和不改变的数据。

为了实现仓库或在线归档压缩，你必须使用直接路径加载、ALTER TABLE . . . MOVE 操作或者在线表重定义。

混合列压缩优化数据仓库和在 Exadata 存储上的决策支持。Exadata 查询的性能最大化就是在表中使用混合列压缩技术，利用处理能力、内存和 Infiniband 网络带宽都是 Exadata 存储服务器不可或缺的。

其他的 Oracle 存储系统支持混合列压缩，提供同样的节省空间在 Exadata 存储

上，但不提供相同级别的查询性能。对于这些存储系统，混合列压缩是归档数据库中不经常访问的旧数据的理想选择。

压缩单元 混合列压缩使用一个叫做**压缩单元（compression unit）**的逻辑结构来存储行的集合。当你加载数据到表中，数据库在柱状格式中存储每一组行，并带有存储和压缩的每一列的值在一起。在数据库对行的集合进行列数据压缩后，数据库将数据融入到压缩单元。

例如，你对表 daily_sales 采用混合列压缩。在每天结束时，你将项目和出售数量及项目 ID 和日期形成的复合主键填入表中。表 2-1 显示 daily_sales 中的行的一个子集。

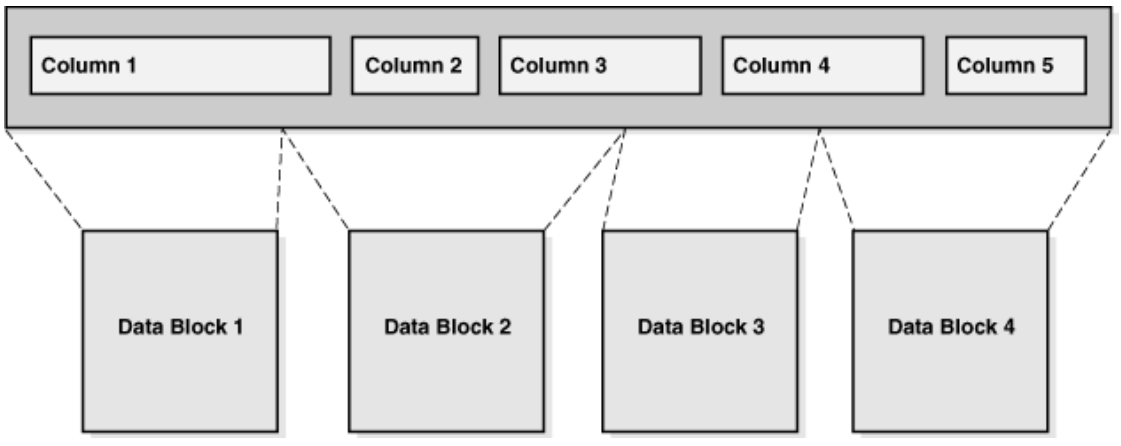
表 2–1 样表 daily_sales

Item_ID	Date	Num_Sold	Shipped_From	Restock
1000	01-JUN-11	2	WAREHOUSE1	Y
1001	01-JUN-11	0	WAREHOUSE3	N
1002	01-JUN-11	1	WAREHOUSE3	N
1003	01-JUN-11	0	WAREHOUSE2	N
1004	01-JUN-11	2	WAREHOUSE1	N
1005	01-JUN-11	1	WAREHOUSE2	N

假设表 2-1 中的行存储在一个压缩单元中。混合列压缩一起存储每一列的值，然后使用多个算法压缩每一列。数据库基于各种因素选择算法，包括列的数据类型、列中实际值的基数和用户选择的压缩级别。

正如图 2-5 所示，每个压缩单元可以跨越多个数据块。特定列的值可能会或可能不会跨越多个块。

图 2–5 压缩单位



混合列压缩会影响行锁（见“行锁（事务）(Row Locks (TX))”在 9-18 页）。为未压缩数据块中的行更新时，只有更新的行被锁住。相比之下，如果更新单元中的任一行，数据库必须锁住压缩单元中的所有行。更新使用混合列压缩的行会导致 rowid 的改变。

注意： 当表使用混合列压缩时，Oracle DML 锁住更大的数据块（压缩单元），这可能会降低并发。

参见：

- *Oracle 数据库许可信息* 来了解关于混合列压缩的许可需求
- *Oracle 数据库管理员指南* 来了解如何使用混合列压缩

表簇(Table Clusters)的概述

表簇（table cluster） 是一组有着共同的列和存储相关的数据在相同块中的表。当表为簇时，单个**数据块(data block)**可以包含来自多个表中的行。例如，块可以存储来自雇员和部门表的行，而不止来自于单一表的行。

簇键（cluster key）是簇表共有的一列或多列。例如，雇员和部门表共享了 department_id 列。当表簇时并在创建每一张表添加表簇时，你指定簇键。

簇键值（cluster key value）是对于一组特定行的簇键列的值。所有的数据都包含相同的簇键值，如 department_id=20，是物理的存储在一起。每一个簇键值都在簇和簇索引中仅存储一次，无论有多少不同表中的行包含这个值。

打个比方，假设人力资源经理有两本案例：一个雇员的文件箱和另一个部门的文件箱。用户经常需要在特定部门的所有雇员文件夹。为了使检索更方便，经理重新排列了所有的箱子在一个案例里。她用部门 ID 来分开箱子。

因此，所有在部门 20 的雇员和部门 20 的文件夹本身都在一个箱子中；在部门 100 的所有雇员的文件夹和部门 100 文件夹是另一个箱子，以此类推。

你可以考虑簇表当它们主要是查询（但不是修改）和经常一起查询或连接的表中记录。因为表簇存储不同表的相关行在相同的数据块中，正确的使用表簇会提供下面对于非表簇更多的好处：

- 磁盘 I/O 在**连接（join）**簇表时减少。
- 连接簇表时访问时间有所改善。
- 需要更少的存储来存储相关的表和索引数据，因为簇键值不重复存储每一行。

通常情况下，簇表在下列情况下是不适用的：

- 表更新频繁。
- 表频繁需要**全表扫描（full table scan）**。
- 表需要截断。

参见：Oracle 数据库性能调优指南 了解何时使用表簇的指引

索引簇的概述

索引簇(indexed cluster)是一个使用索引定位数据的表簇。**簇索引(cluster index)** 是一个在簇键上的 B-树索引。簇索引必须在所有行被插入到簇表之前被创建。

假设你创建的带有簇键 department_id 的簇 employees_departments_cluster，如例 2-8 所示。因为 HASHKEYS 条件并没有指定，这个簇是索引簇。之后，你在这个簇键上创建索引名叫 idx_ emp_dept _cluster。

例 2–8 索引簇

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

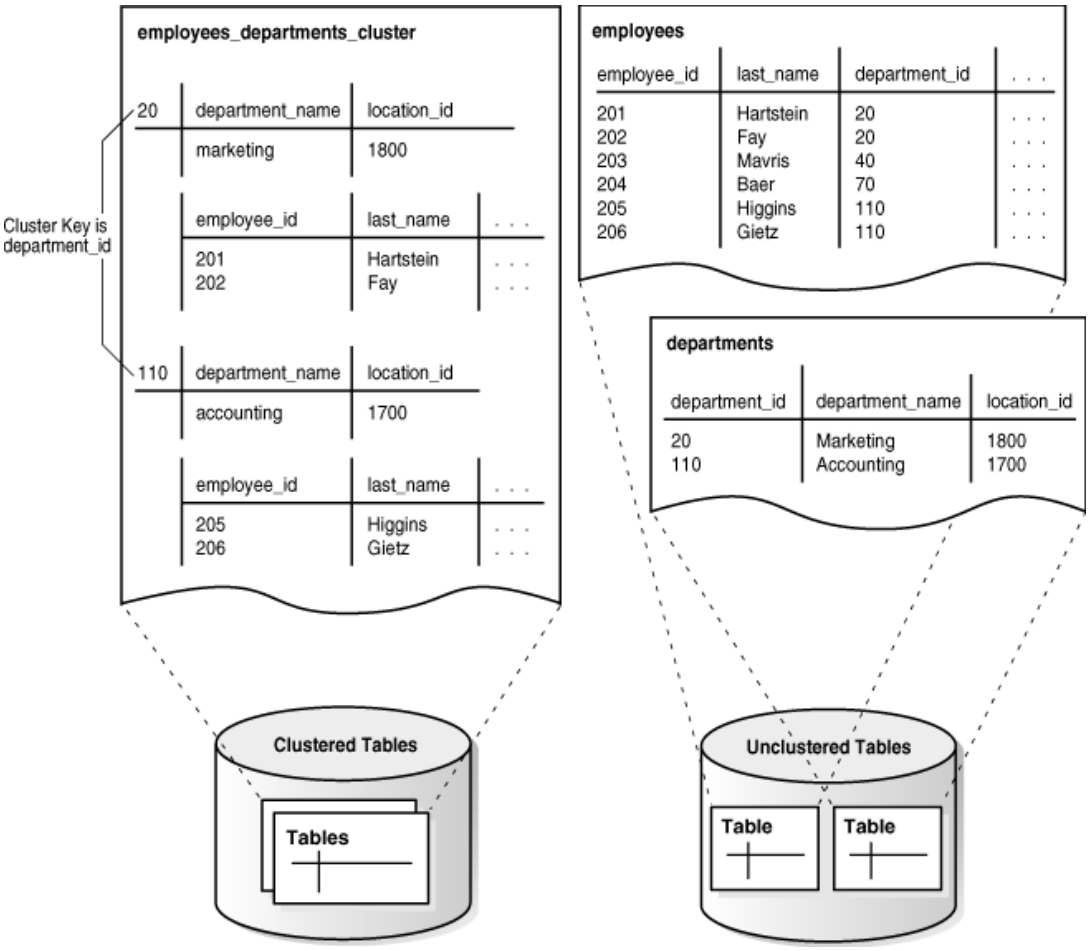
然后你可以在簇中创建 employees 和 departments 表，指定 department_id 列为簇键，如下（省略号标记为指定列的地方）：

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

最后，你添加行到 employees 和 departments 表中。数据库在 employees 和 departments 表中相同数据块以物理方式存储每一个部门的所有行。数据库在堆中存储这些行并且用索引定位它们。

图 2–6 显示了包含 employees 和 departments 的 employees_departments_cluster 表簇，数据库将雇员的行一起存储在部门 20、部门 110 等等。如果表没有簇，那么数据库不确保相关行存储在一起。

图 2–6 簇表数据



B-树簇索引通过包含数据的块的**数据块地址（database block address (DBA)）**和簇键关联。例如，索引项键 20 显示在部门 20 中包含雇员数据的块地址：

20,AADAAAA9d

簇索引是分开管理的，就像非簇表上的索引，可以存在与表簇分离的表空间中。

参见：

- "索引概述" 在 3-1 页
- *Oracle 数据库管理员指南* 来了解如何创建和管理索引簇
- *Oracle 数据库 SQL 语言参考手册* 查看 CREATE CLUSTER 的语法和语义

散列簇概述

散列簇（hash cluster）像索引簇一样，除了索引键被**散列函数（hash function）**替换外。没有分割的簇索引存在。在散列簇中，数据是索引。

有索引表或索引簇的存在，Oracle 数据库在一个单独索引中通过存储的键值定位表中的行。要找到或存储在索引表或者表簇中的行，数据库必须执行至少两个 I/O：

- 一个或多个 I/O 去寻找或存储在索引中的键值
- 另一个 I/O 去读或写在表或表簇中的行

要找到或存储在散列簇中的行，Oracle 数据库要对行的簇键值套用散列函数。产生的散列值对应到在簇中的数据块，由数据库代表发表声明来读取或写入。散列是一个可选的储存表中数据来提高检索数据性能的方式。当符合下列条件时散列簇可能是有益的：

- 查询某表往往多于修改该表。
- 经常用等于条件查询散列键列，例如，WHERE department_id=20。对于这样的查询，簇键值是被散列的。散列键值直接指向存储行的磁盘区域。
- 你能合理的猜出散列键的数量和存储的每一个键值数据的大小。

散列簇的创建

簇键（cluster key），像索引簇的键一样，是单一的列或者在簇中通过表共享的组合键。**散列键值（hash key values）**是实际或可能的值被插入到簇键列中。例如，如果簇键是 department_id，那么散列键值可能是 10、20、30 等等。

Oracle 数据库使用散列函数，它可以接受无限量的散列键值作为输入并把它们排序到一个有限量的桶中。每个桶都有唯一的数字 ID 称为**散列值（hash value）**。每个散列值映射到数据库中存储行对应的散列键值（部门 10、20、30 等等）块的块地址。

要创建一个散列簇，你要对另外的散列键的索引簇使用相同的 CREATE CLUSTER 语句。对于簇的散列值的数量依赖于散列键。在例 2-9 中，部门的数量很可能是 100，所以 HASHKEYS 被设置为 100。

例 2–9 散列簇

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

在你创建 employees_departments_cluster 之后，你可以在簇中创建 employees 和 departments 表。然后你可以加载数据到散列簇中，正如在例 2-8 中索引簇描述的。

参见：*Oracle 数据库管理员指南* 来了解如何创建和管理散列簇

散列簇查询

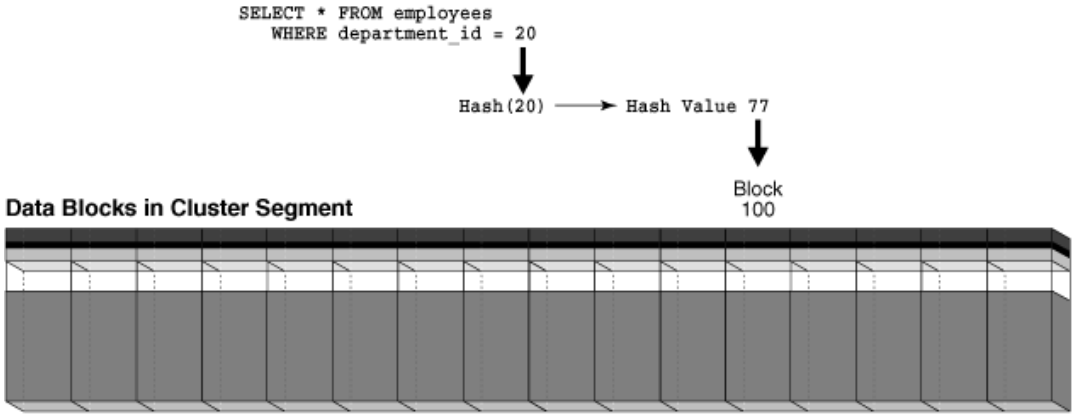
数据库而非用户来确定如何散列用户输入的键值。例如，假设经常执行如下的查询，进入不同的部门 ID 号 p_id：

```
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;
```

如果用户在 department_id=20 中查询员工，那么数据库可能散列该值到 77 桶中。如果用户在 department_id=10 中查询员工，那么数据库可能散列这个值到 15 桶中。数据库使用内部生成的散列值来定位请求的部门中包含雇员行的块。

图 2–7 描述了一个水平排列块样子的散列簇段。正如图中所示，查询可以在单个 I/O 中检索数据。

图 2–7 从散列簇中检索数据



散列簇的一个限制是在非索引簇键上不可用的范围进行扫描（参见“索引范围扫描”在 3-7 页）。假设在例 2-9 中不存在散列簇创建的单独索引。查询部门在 20 和 100 之间的 ID 不能使用散列算法，因为它不能散列在 20 和 100 之间的每个可能值。因为不存在索引，数据库必须执行完整扫描。

散列簇变量

单表散列簇（single-table hash cluster）是在同一时间只支持一张表的优化版本散列簇。一对一映射存在于散列键和行中。当用户需要通过主键快速访问表时，单表散列簇可能起作用。例如，用户经常在 employees 表中通过 employee_id 查看员工记录。

排序散列簇（sorted hash cluster）存储行对应的每个散列函数值，这种方式数据库可以按照排序顺序高效的返回这些值。数据库内部执行优化排序。对于总是用排序顺序消耗数据的应用来说，这项技术可以意味着更快的数据检

索。例如，一个应用可能总是在 orders 表的 order_date 列中排序。

参见：*Oracle 数据库管理员指南* 来了解如何创建单表和排序散列簇

散列簇存储

Oracle 数据库从一个索引簇中给散列簇分配不同的空间。在例 2-9 中，HASHKEYS 特指部门可能存在的数目，而 SIZE 特指每个部门相关的数据大小。数据库根据下面的公式计算存储空间的值：

```
HASHKEYS * SIZE / database_block_size
```

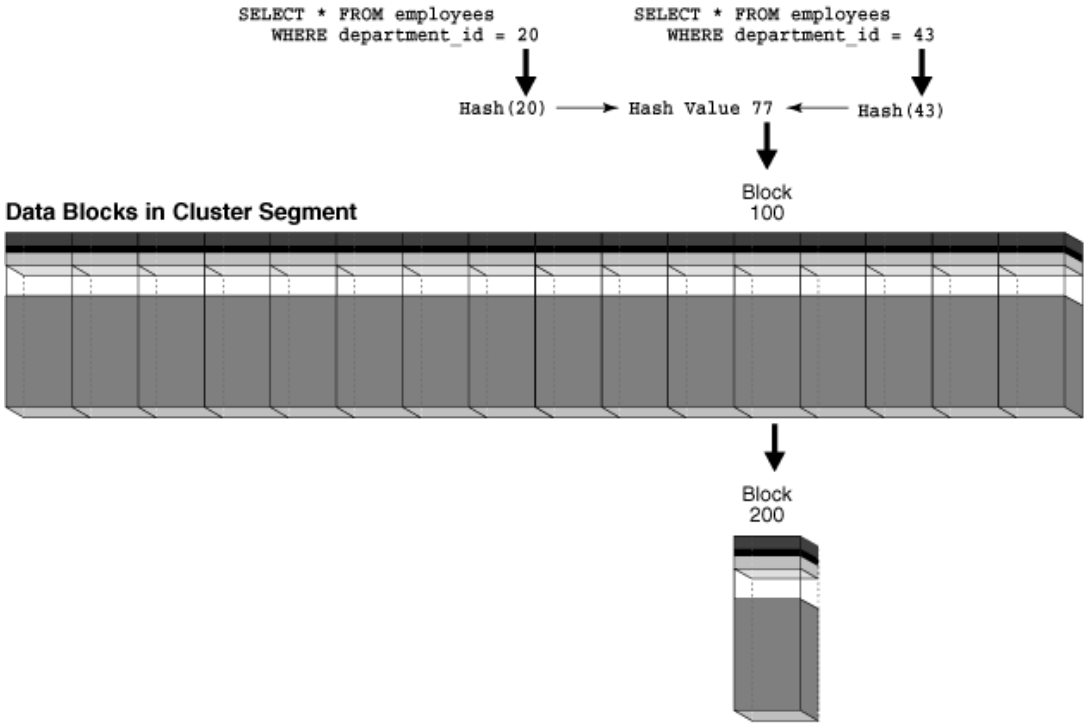
因此，在例 2-9 中如果块大小是 4096 字节，那么数据库需要分配至少 200 个块给散列簇。

Oracle 数据库不限制你插入到簇中散列键值的个数。例如，即使 HASHKEYS 是 100，也没有什么能阻止你插入 200 个唯一的部门在 departments 表中。然而，当散列值的数量超过散列键的数量时，散列簇检索的效果将减弱。

要说明检索的问题，假设在图 2-7 中的块 100 是完满的部门 20 的行。用户用 department_id 43 插入一个新的部门到 departments 表中。部门的数目超过了 HASHKEYS 值，所以数据库散列 department_id 43 为散列值 77，该值与 department_id 20 使用的散列值相同。散列多个输入值到相同输出值被称为**散列冲突（hash collision）**。

当用户插入部门 43 的行到簇中，数据库不能在已满的块 100 中存储这些行。数据库链接块 100 到一个新的溢出块，例如块 200，并在新块中存储插入的行。块 100 和 200 现在都有资格存储部门的数据。正如在图 2-8 中所示，现在同时查询部门 20 或 43 需要*两个* I/O 来检索数据：块 100 和它相关的块 200。你能通过重新创建不同 HASHKEYS 值的簇来解决这个问题。

图 2–8 当散列冲突发生时从散列簇中检索数据



参见：*Oracle 数据库管理员指南* 来了解如何在散列簇中管理空间