

第一部分 数据库管理的原则

第1章 数据库、DBMS原理和关系型模型

本章要点：

了解数据库

了解DBMS

了解RDBMS

1.1 了解数据库

在过去的许多年里，有许多关于“数据库”这个名词的定义。数据库是一个服务于一个核心目标的数据的有组织的集合。数据库中的数据是有组织的，从某种意义上说，数据库中存储的数据采用一种不变的方式被存储、格式化、存取以及显示。因为数据库不含有无关的或冗余的数据，它可以适用于一个核心目标。一本电话簿就是一个很好的数据库例子，它包含有关的数据（名字），让人们能够查找电话号码；它不包含无关的数据，如某人的电话机的颜色；它只贮存那些与它的目标相关的信息。最常见的，一个数据库的目标是商务应用，但是也可能贮存科学、军事或其他数据，这些数据通常不能当作商务数据看待。因此，有商业数据库、科学数据库、军事数据库以及其他的数据库等等。另外，数据不仅能根据它的应用分类，还能根据它的格式分类，现代数据库包括多种类型的数据。例如，现在数据库贮存图像、图表、声音、视频或包括两种或多种类型的复合文档，已经是很普通的事了。

当讨论数据库特别是数据库设计时，通常称数据库所服务的核心目标为它的业务（business），而不管它属于什么特殊的领域，如太空、生物医学或其他。此外，在实际生活中，你会发现数据库通常有它明确的业务应用。

在早些年，编写程序实现自动数据处理（Automatic Data Processing, ADP）要求的程序员发现，在每次运行中，他们需要频繁地贮存数据，这就是通常所说的对永久内存的需要，即从程序的一次运行到下一次运行时，需要将数据存留或贮存。这个基本的需要成为数据库发展的开端。其次的需要——简单的数据存储，也促进了数据库的产生。在线归档和历史数据就是一对特别的例子。尽管文件、目录和文件系统能满足多数普通的数据存储需要（包括索引变化），但数据库可以做文件系统能做的工作并且还可做更多的。

现代数据库通常为上级组织或企业的部门，或它们的小型组织单位实现存贮处理需求。因此，你用术语“企业范围（enterprisewide）”来指代整个组织的商务范围，用“部门范围（departmentwide）”来指代一个部门级的范围，用“工作组（workgroup）”指一个部门内的一些单位。通常，在部门范围和工作组级上查找数据库。

偶尔，你会查找服务于企业范围的数据库，例如工资单和人事数据库，但是它们在数量上远远少于那些较小的数据库。实际上，当几个部门的数据库放在一起或合并成为一个大数

据库时，这就是建立一个数据仓库（Data Warehouse, DW）的本质。那些充当大型数据库的数据源的小型数据库，称为可操作数据库。然而，这不是新东西，一个操作数据库就是产生数据的数据库，多年来一直被叫做成品数据库；只有在建立数据仓库的前提下，你才会发现成品数据库是指可操作数据库或有时是指可操作的数据存贮。随着因特网技术的出现，数据库和数据仓库现在常常作为前端 Web浏览器的后端使用。

当工作组数据库合并起来以满足一个大型部门需要时，结果通常相当于一个数据中心（Data Mart, DW），一个DM就是一个部门规模的数据仓库。和术语“数据库”那样，术语“数据仓库”也会产生多种定义。然而，当你把几个小型数据库集成为一个大型数据库，为一个较广泛的组织服务时，如果该数据库存储历史数据、提供决策支持、提供数据汇总、提供只读数据，并且实质上充当所有向它提供数据的相关成品数据库的数据接收器，那么它通常被看作是一个数据仓库。

另外，如果一个数据库增大的原因是因为该数据库是一个长时期贮存数据的历史数据库（例如一个人口普查数据库），或因为它必须贮存数据的类型（例如一个图像数据库），或者因为它必须采用的贮存数据的频率（例如一个卫星遥测数据库），那么它通常被称为一个非常大的数据库（Very Large Database, VLDB）。

随着时间的发展，由于磁盘容量的增大和价钱的下降、均衡多处理技术机器的出现、冗余廉价磁盘阵列（Redundant Array of Inexpensive Disks, RAID）技术的发展、数据库软件的增多或规模化，判定一个数据库是否为 VLDB的条件不断变化。当前，一个常用的准则是任何 100GB或100GB以上的数据库就可以被看作是一个 VLDB；而就在几年前，10GB就被看作是分隔点了。

1.2 理解DBMS

数据库管理系统（Database Management System, DBMS）就是管理一个数据库的软件，它充当所有数据的知识库，并对它的存储、安全、一致性、并发操作、恢复和访问负责。DBMS有一个数据词典（有时被称为系统目录），其中贮存着它拥有的每个事物的数据，例如名字、结构、位置和类型，这种关于数据的数据也被称为元数据（metadata）。在一条数据的生存周期里（从它的创建到删除），这条数据的逻辑和物理信息都被记录在数据词典中。数据库系统管理员（Database Administrator, DBA）应该熟悉 DBMS的数据词典；在数据库的整个生命周期内，数据词典为他或她服务。

1.2.1 数据的安全

在成品数据库中，安全性一直是一个重点，在开发或测试数据库中也常如此。这通常不是有没有安全性的问题，而是有多大安全性的问题。除了操作系统和网络安全设施外，一个 DBMS通常提供几层安全措施。最常见的情形是，DBMS要求用户登录用户帐户的口令，口令被验证为真，才能访问数据库。

DBMS也提供其他的机制，例如组、角色、权限以及简档，这些都提供更加精良的安全措施。提供这些安全级不仅是为了加强安全性，而且为了建立商业安全策略。例如，只有一个经过验证属于航空组的用户才能存取航空数据；又例如，只有经过验证拥有操作者角色的用户才能备份数据库。

1.2.2 维护和实施完整性

数据的完整性是指它的一致性和正确性。对于一致的数据，在它所有的出现的场合中，必须以相同的方式建模和实现，对于正确的数据，它必须是正确、精确和有意义的。

DBMS维护完整性的一个方法是在一条数据项发生改变的过程中进行锁定。数据库通常是在数据库页级或行级锁定数据。锁定偶尔也允许并发操作，在下面你将涉及到这种情形。

DBMS实施完整性的另一个方法是：如果一条数据贮存在多个地方，那么把变化复制到这条数据上。DBMS实现完整性的最后一个方法是通过监视输入的或改变的数据值，使它们全部符合要求的规格（例如：一个范围检查）。

如果接着的是合适的建模和实现过程（在后面的第2章“逻辑数据库的设计和标准化”中讨论），DBMS会自动地帮助实施这种完整性，例如，通过一个触发器或约束条件，这两个概念在第28章“PL/SQL基础”中被定义和说明。没有完整性，数据是没有价值的。有了完整性，数据就是信息。完整性不仅能增强数据，它还给予数据以价值。

当DBMS提供多用户存取时，它必须管理并发操作。那就是说，当几个人同时访问相同的数据库（特别是同一条数据）时，DBMS必须保证这种并发访问能够以某种方式实现。并发可以广义上定义为同时发生，即两个或多个用户在同一时期访问同一数据。

DBMS实现并发操作的方法倒不太复杂，但背后的实际程序却很复杂。本质上说，当两个或多个人只想简单地浏览一下同一个数据而不改变它时，一切还好。但当至少一个人想改变数据而其他人想浏览或也想改变数据时，DBMS必须贮存多个备份。当每个人都完成改动后，DBMS必须将所有改变的备份保存为一条正确的数据。

前面已提到并发管理的一种方式 is 加锁。通常来讲，锁越精密（越小），并发性就越好（那就是说，更多的用户无需等待即可同时访问）。行通常比最小的数据库页或块都小，因此，行级锁可较好地管理短小、杂乱数据的处理，块级锁可较好地管理长的、连续数据的处理。

这就是并发性和完整性是如何结合的。当一个人想查看或更改一条数据时，这个人就是执行数据库的一个事务。

1.2.3 理解事务

作为DBMS标准的一部分，DBMS有一个事务管理器（transaction manager），它的目的是管理并发操作和确保事务的完整性。事务管理器的工作是艰巨的，因为它必须允许许多人同时访问相同的数据，并且在访问之后要把数据放回到数据库中，就好象在某个时间上只有一个人存取数据，他完成工作后另一个人才工作，这样确保数据的正确性。DBMS解决数据的多个备份的基本方案就在这中间。如果（并且是只有）数据是串行的，那么在保持数据的准确性的同时也进行了事务处理。简单地说，DBMS必须重新整理所有改变，以使得它们的最终结果仿佛发生在一个文件中。

事务是并发或工作的单位。不能产生比一个事务更小或更少的东西，也就是说，没有人能够只完成数据改变工作的一部分。全部事务必须都是原子的，所以每个单独的事务要么完成，要么不完成。直到20世纪现代物理发展起来以前，原子一直被当作物质的最小单位。同样，事务也是并发的最小单位，它要么全有，要么全无。一个被完成的事务可以说是被提交了，没有完成的事务则是被回滚了。

DBMS用事务作为恢复的单位来控制恢复，正常完成、手工要求中止以及意料之外的退

出都要求DBMS重新访问数据的多个备份来提交或回滚数据。为了回滚或前滚，DBMS保持了一个事务日志。回滚是一个撤销操作。前滚是一个重做操作，例如，由于一个硬件或软件错误，一个已提交的事务无法将它所做的操作从内存中存储到磁盘就会发生前滚，DBMS只是简单地重做这个操作。因此，在DBMS中事务恢复的关键是一个事务必须是原子的，而且在必要时可以做、不做或重做。

1.2.4 与数据库通信

如果你不能和一个DBMS对话，那么这个DBMS就不是很好的。你可能会问怎样和DBMS对话。可以通过一种存取或查询语言访问数据库。结构化查询语言（Structured Query Language, SQL）是当今主要的查询语言，它主要用于管理主流类型的DBMS——关系型DBMS（RDBMS）。所有与数据库相关的通信往来都将通过DBMS完成，为了做这件事，你可以使用SQL或其他类似的东西。数据库系统管理员（DBA）使用查询语言来建立并维护数据库，用户使用查询语言来访问数据库并查看或更改数据。

1.3 理解RDBMS

1970年，E. F. Codd创立了关系模式的概念。在RDBMS（例如DB2）产生之前，层次（IMS）和网状（IDMS）模式是常见的。在这些模式之前，使用平面文件（操作系统文件不一定平面）来建立数据库，并且使用第三代语言（3GL）访问例程。实际上，一些专用系统仍然是按这种方式建立的，只是进行了修改或根本没有变化。在大型机和微机中依然存在着许多这样的遗留数据库。CODASYL（数据系统语言协会）是数据库任务组（Database Task Group, DBTG）创建的一种数据库标准，这是一种基于COBOL的网络数据库标准，并且IDMS是一个厂商的实现。但是，从70年代起，RDBMS已经逐渐地控制了市场，如Oracle、Sybase、Informix和Ingres。

最近，面向对象（Object-Oriented, OO）的DBMS已经成为最为突出的数据库管理系统，并找到了许多适当的应用环境，如在CAD/CAM、工程、多媒体等等。面向对象DBMS适于在这些领域中应用，因为在一个几乎非事务性的环境中，它们具有控制复型数据类型实力。由于竞争，RDBMS厂商为了提供包括文本、音频、图像和视频数据类型的面向对象/多媒体性能，已经制造了商业可用的通用服务器。Oracle的Universal Server就是一个例子。另外，用户定义的数据类型或可扩展类型，已经被扩大或增加到核心数据库服务器中，Oracle 8就提供了这样的性能。类似这样的RDBMS产品被认为是混合的，然而它们明显比以前的RDBMS更具有主流性。

此外，多维数据库（Multi-Dimensional Database, MDD）也分享了部分市场份额，这些数据库为带有许多必须被多维存取或列表的变量（例如行为科学数据）的应用提供了高度索引化的数据。在传统的RDBMS中，这几乎是不可能实现的，数据库只允许单独使用。再者，为和MDD竞争，RDBMS供应商提供了一些他们自己的层次产品，这些产品提供超级索引化的数据，并使用了特殊的技术，例如位映射索引。Oracle的Express就是一个多维数据库的例子。

1.3.1 关系模型

你已经了解了DBMS的主要任务，为了进一步了解一个RDBMS是由什么构成的，你必须

先了解关系模型。下列情况出现在一个关系模型中：

- 数据的基础项是关系。
- 在这些表上的操作只产生关系（关系型闭合）。

什么是关系？这是一个描述两个集合的元素如何相互联系或如何一一对应的数学概念。因此，关系模型是建立在数学基础上的。然而，对你来说，关系只是一个带有一些特殊属性的表，一个关系模型把数据组织到表中，而且仅在表中。客户、数据库设计者、数据库系统管理员和用户都以同样的方式——即从表中——查看数据。那么，表就是关系模型的近义词。

一个关系型表有一组命名的属性（attribute）或列，以及一组元组（tuple）或行。有时列被称为域，行被称为记录，列和行的交集通常被叫做单元。列标示位置，有作用域或数据类型，例如字符或整数。行自己就是数据。表 1-1 有三列四行。

表1-1 汽车表

制 造	模型品牌	价 格
Toyota	Camry	\$25K
Honda	Accord	\$23K
Ford	Taurus	\$20K
Volkswagen	Passat	\$20K

一个关系表必须符合某些特定条件，才能成为关系模型的一部分：

- 贮存在单元中的数据必须是原子的。每个单元只能存贮一条数据，这也叫信息原则（Information Principle）。尽管在过去的数年中按某些违反这一条的方式已经建立了许多系统，但违反这一条将不能运用良好的设计原则。当一个单元包含多于一条的信息时，这叫做信息编码（information coding），一个很好的例子是一个车辆识别号码（Vehicle Identification Number, VIN）。如果它被存贮成一行，这将违犯信息原则，因为它包含了多条信息，例如产地、型号、出厂等等。在这样的情况下，是否采用违背理论的方案是一个设计的选择问题，尽管在多数情况下，结果证明这对数据的完整性是不利的。
- 贮存在列下的数据必须具有相同数据类型。
- 每行是唯一的（没有完全相同的行）。
- 列没有顺序。
- 行没有顺序。
- 列有一个唯一性的名称。

除了表和它们的属性，关系模型有它自己特殊的操作。不需要深入研究关系型数学，只需说明这些操作可能包括列的子集、行的子集、表的连接以及其他数学集合操作（如联合）等就足够了。真正要知道的事情是这些操作把表当作输入，而将产生的表作为输出。SQL是当前RDBMS的ANSI标准语言，它包含这些关系型操作。在SQL占主导地位之前，一种具有竞争性的语言是来自Ingres的QUEL或QUEry语言，另一种是UDL（统一数据语言，Unified Data Language）。ANSI（美国国家标准化组织）是一个具有广泛范围的标准实体，其中包括计算机软件语言（如SQL）的标准。

允许数据操作或数据处理的主要语句是SELECT、INSERT、UPDATE和DELETE。因此，这些数据处理操作中任何一个都是一个事务。

允许数据定义或结构化处理的基本语句是CREATE、ALTER和DROP。所有这些语句都可

以使用一组子句来替代，这些子句可以利用多种变化来定义和存取组成数据库关系表的结构和数据。因此，SQL既是一种数据定义语言（Data Definition Language，DDL），也是一种数据操作语言（Data Manipulation Language，DML）。统一的DDL和DML肯定比两种不同的语言和接口更有效、更有用。数据库系统管理员和用户可以通过完全相同的语言访问数据库。

关系模型要求的最后一件事是两个基础的完整性原则。它们是实体完整性原则（entity integrity rule）和引用完整性原则（referential integrity rule）。首先，让我们看看两个定义：

- 主键（primary key）是能唯一标识行的一列或一组列的集合。有时，多个列或多组列可以被当作主键。
- 由多个列构成的主键被称为连接键（concatenated key）、组合键（compound key），或者更常称为复合键（composite key）。

数据库设计者决定哪些列的组合能够最准确和有效地反映业务情形，这并不意味着其他数据未被存贮，只是那一组列被选作主键而已。

剩余有可能被选为主键的列被叫做候选键（candidate key）或替代键（alternate key）。一个外键（foreign key）是一个表中的一列或一组列，它们在其他表中作为主键而存在。一个表中的外键被认为是对另外一个表中主键的引用。实体完整性原则简洁地表明主键不能全部或部分地空缺或为空，引用完整性原则简洁地表明一个外键必须为空或者与它所引用的主键当前存在的值相一致。

一个RDBMS就是一个建立在前面这些关系模型基础上的，一般能满足所提到的全部要求的DBMS。但是，在70年代末到80年代初，RDBMS开始销售的时候，SQL超越了本质为非关系型的系统，受到普遍欢迎，并被称作关系型。这引发了一些修正活动，即 Codd十二条法则（1985）。

1.3.2 Codd十二条法则

DBMS应该遵循Codd提出的十二条法则，才能被分类到完全关系型：

- 1) 信息法则。信息表现为贮存在单元中的数据，正如前面所讨论过的，将 VIN作为一个单独的列使用，违反了这条规则。
- 2) 授权存取法则。每一个数据项必须通过一个“表名 + 行主键 + 列名”的组合形式访问。例如，如果你能用数组或指针访问一个列，就违反这条规则。
- 3) 必须以一致的方式使用空值。如果由于缺少数字值，空值（Null）被当作0来处理，或者由于缺少字符值而被当作一个空格处理，那么它就违反了这条规则。空值仅仅是指缺少数据而且没有任何数值。如果缺少的数据需要值，软件提供商通常提供使用缺省值的能力满足这一目的。
- 4) 一个活跃的、在线数据字典应作为关系型表被存储，并且该字典应该可以通过常规的数据存取语言访问。如果数据字典的任何部分贮存在操作系统文件里，就违反了这条规则。
- 5) 除了可能的低级存取例程外，数据存取语言必须提供所有的存取方式，并且是存取的仅有方式。如果你能通过一个实用程序而不是一个 SQL接口来存取支持一个表的文件，就有可能违反了本规则。参见规则 12。
- 6) 所有能被更新的视图应当是可更新的。例如，如果你能将三个表连结起来，作为一个视图的基础，但却不能更新这个视图，则违反本规则。

7) 必须有集合级的插入、更新和删除。目前, 大多数 RDBMS 提供商都在某种程度上提供了这种能力。

8) 物理数据的独立性。应用不能依赖于物理结构, 如果一个支持某表的文件从一张盘移动到其他盘上或重新命名, 不应该对应用产生影响。

9) 逻辑数据的独立性。应用不应依赖于逻辑结构。如果一个表必须被分成两个部分, 那么应该提供一个视图, 以把两段连接在一起, 以便不会对应用产生影响。

10) 完整性的独立性。完整性规则应该贮存在数据字典中。主键约束、外键约束、检查约束、触发器等等都应该贮存在数据字典中。

11) 分布独立性。一个数据库即使被分布, 也应该能继续工作。这是规则 8 的一个扩展, 一个数据库不仅能在一个系统(本地地)分布, 也能在通过系统的网络(远程地)分布。

12) 非破坏性法则。如果允许低级存取, 一定不能绕过安全性或完整性规则, 这些规则是常规的数据存取语言所遵守的, 例如, 一个备份或载入工具不能绕过验证、约束和锁来备份或载入数据。然而, 软件供应商出于速度的原因, 通常提供这些功能。那么, 数据库系统管理员就有责任确保数据的安全性和完整性, 如果瞬间出现问题, 应该立即恢复。例如当载入 VLDB 时, 可以临时禁止并重新打开约束检查。

如果一个 DBMS 能满足本章中讨论的所有基本原则(两个定义、六个属性、关系型操作以及两个完整性规则)和这十二条法则, 那么它就可以被当作一个 RDBMS。Codd 用他的法则 0 总结了这一切: “对于一个有资格成为 RDBMS 的系统来说, 该系统必须排他地使用它的关系型工具来管理数据库。”

第2章 逻辑数据库的设计和标准化

本章要点：

实体-关系模型

将实体关系图映射为关系模型

理解标准化

2.1 实体-关系模型

对一个数据库管理员来说，所能为他的数据库做的最好的事情就是使之开始于一个合理的逻辑设计。不幸的很，数据库设计常常被匆匆地完成以致于做错，甚至在数据库建立后重新返工。一个见闻广博的和聪明的数据库管理员知道对数据库进行很好的设计，会大大提高数据库的性能，而不是减损数据库的性能，这种思想与流行的思想相反。事实上，直接投入物理设计或更深层的工作，只会带来麻烦，不仅在性能方面，而且在数据完整性方面同样如此。如果一个数据库运行得很快，但收藏的数据却是错误的，这又有什么好处呢？而且，在数据库系统的早期设计阶段，创建一个合理的逻辑设计，可以让它接受以后创建和维护阶段物理设计改变的考验。可是，如果你在逻辑设计阶段走捷径，你将不但可能需要重新设计逻辑模型，而且还可能需要重新构造下面的物理模型。间接的代价（职员的工作时间、停工期等等）可能会是令人吃惊的。在进行和建立数据库之前，需要了解逻辑数据库设计和标准化背后的基本原则。

在70年代中期，关系数据库模型逐渐超越其他的数据模型占据主导地位，关系模型技术的风靡使设计性能得到规范化。这其中最流行的是实体关系图（Entity-Relationship Diagram, ERD），它是P.P.Chen在1976年提出来的。这就是语义数据模型，因为它试图捕获业务要素（业务本质）的语义或正确含义。因为关系模型本身几乎就是一个依据语法的模型，是一种主要处理结构的模型，实体关系图（ERD）通常用于补充它。实际上，ERD建模必然先于关系建模。当一个ERD结束时，它或多或少地被直接映射到关系模型上，而后关系模型再被映射到它的物理模型上。

一个实体是一个业务元素，比如一个雇员或一个项目。一个关系就是两个实体之间的联系，比如工作于不同项目的雇员。属性即组成实体的特征，比如一个雇员的工资或项目的预算。属性被认为是来自定义域中的取值或值的集合，它们所取的值是它们以后在关系模型中所用到的数据。它们是对一个事物全部抽取或部分抽取。ERD有许多画法，只要你选择一种并在整个使用过程中保持含义一致即可。

使用方框代表实体画高级图（那些不带属性的），将实体的名字列于方框的中心。低级图的实体名称列于方框中的上部，后面跟着属性名称。在方框之间画有箭头，代表关系类型，有三种基本类型的关系：一对一、一对多以及多对多。一对一的关系根据一对一关系的类型，在线条的一端或两端使用单箭头。一对多使用双箭头，多对多在两边使用双箭头。当一个实体的每一个值都和另一个实体的一个值并且只有一个值有关时，就存在着一个纯粹的一对一

关系，反之亦然。这种类型的关系是很少见。图 2-1展示一种一对一关系，一个丈夫只能和一个妻子结婚，而且一个妻子也只能和一个丈夫结婚（没有考虑一夫多妻或一妻多夫的情况）。

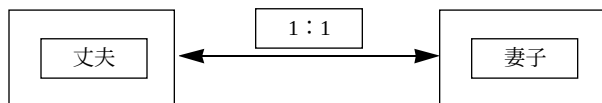


图2-1 一对一 (1:1) 关系

一种更为普遍的一对一关系是子类型关系，这是面向对象分析和设计的基础之一。在面向对象系统中，这被看作是类和子类（或者更简单地说，类的级别）。图2-2显示一个一对一子类型关系是如何被模拟的，该图显示了一个经典的例子：正方形是长方形的子类型。箭头的方向指明了继承的方向，继承是有关类级别的另一个面向对象概念。换句话说，在更为普遍的实体中的属性（长方形）上，将属性（如长和宽）送给更为特定的实体（正方形）。因此，继承的方向是从一般到特殊。

子类型关系比纯类型的一对一关系更为常见，但这两种都不常用。通常，当一个设计者偶然遇到一对一关系时，他必须问下列问题：

- 这两个实体能结合吗？
- 它们对于自己的目标是否是完全相同的？
- 它们是否由于某些业务原因必须保持独立和不同吗？

通常情况下，一对一实体是可以合并的。

在关系模型中，使用得最多的关系是一对多关系，图 2-3显示了一种一对多关系。一个州有许多城市，但所有这些城市只属于一个州（这是正确的。然而，你会发现一个城市的名字被不同的州重复使用，这只代表着一件事：作为一名设计者，你选择的主键一定不能是一个城市的名字，例如：主键可以是州名+城市名。前面的章节包含了主键的定义）。

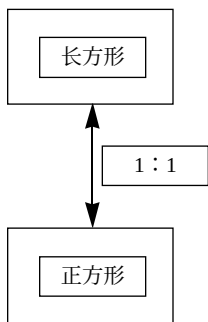


图2-2 一个一对一 (1:1) 的子类型关系

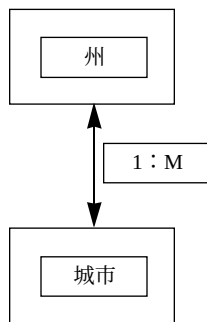


图2-3 一对多 (1:M) 关系

最后，图2-4表示许多雇员正在从事多个项目——这是一个多对多关系。注意带有下列划线的属性是标识符属性，代表着你当前推测它们有可能最后成为关系模型中的主键。

在这种情况下，作为一名设计者，你所能为自己做的最好的事情就是把自己从所有的多对多关系中解脱出来；并不是真正地除去它们，但你可以在它们原先位置上使用两个或多个一对多关系来代替多对多关系。之所以要这样做，是因为关系模型并不能实际处理一个多对多关系的直接实现。仔细想想，如果有许多从事多个项目的雇员，你怎样来贮存外键？你不能在一列中贮存多个值，这样违反了数据必须是原子的关系型要求，这意味着没有一个单元

能够持有一条以上的信息。信息法则（在第1章已讨论过）的这一示例也说明：它是第一范式（First Normal Form）的一个特殊情况。稍后我们将讨论第一范式。因此，为确保数据的原子性，每条多对多关系都被两个或者多个一对多关系所取代。

因此，你所要作的工作就是分割多对多关系，这样多名正从事多个项目的雇员就变成了一个雇员拥有多项职位和一个项目分配给多个职位，而职位成为新的实体。图2-5表示了这种新的关系，注意，标识符属性已经被合并。

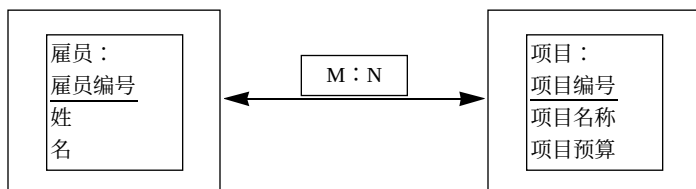


图2-4 多对多（M:N）关系

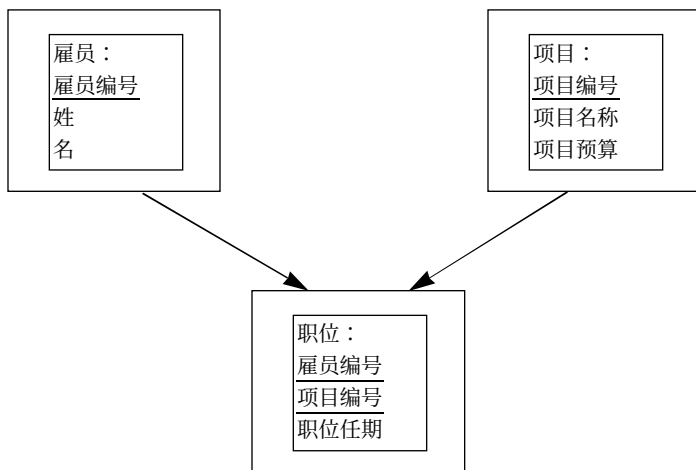


图2-5 使用两个一对多（1:M和1:N）关系修改多对多（M:N）关系

在关系模型中，被称为职位的新实体通常被叫做交叉表，因为它代表着与其相关的两个表中每一对实际值的交集，有时也称为叫纽带表（junction table）或连接表（join table），交叉表是这样的一个实体：它不一定总是一些业务元素的真实抽象，但是它是解决和实现关系模型中多对多关系的基本方法。

2.2 将实体关系图映射为关系模型

一个ERD可以准确地映射到关系模型上，因为它就是为这个目的而创建的。实质上，实体变成了表，属性变成了列，标识符属性变成了主键。如果不通过交叉表，关系并不能真正实现。外键是这样创建的：把一个单方表中的主键放入一个多方表中。例如：一个州对应许多城市的关系要求你把州的主键放到城市表中，在那里创建一个外键，这样在两者之间就建立了关系。

当前市场上有许多自动化计算机辅助软件工程（Computer Assisted Software Engineering, CASE）工具能够帮助你实现这个映射。这样的工具包括 LogicWorks公司的ERwin和Oracle公

公司的Designer/2000, 这些工具不仅能画出实体关系图, 而且能指定主键、外键、索引、约束, 甚至能够产生标准数据定义语言 (Data Definition Language, DDL) SQL代码帮助你建立表和索引。对于Oracle来说, 你可以直接运行这些脚本, 但通常需要修改它们, 例如修改数据类型或增加存储参数。这些工具还能帮你从一个没有文档的现存数据库反推出该数据库的逻辑模型! 当想要集成数据库或承担一个已建数据库的数据库系统管理员的职责时, 这是特别有用的。CASE工具不仅能够帮助你设计和建立数据库系统, 而且同样也能帮助你形成文档。

2.3 理解标准化

标准化是关系模型的改良或扩展, 标准化也是依照第一个关系模型草案并在此基础上采用一定具体方法进行改进的进程, 这些具体的改进方法你不久将会学到。和关系模型一样, 标准化的基础是数学, 它是建立在被称为函数相关性 (functional dependency, FD) 的概念上的。

尽管没有必要深入研究函数相关的数学, 但至少为关系模型定义它是有用的和指导意义的。一列或一组列 Y , 被认为是与另列或一组列 (X) 函数相关; 如果赋予 X 一组给定的值, 那么将决定 Y 的一组唯一值。说 Y 和 X 函数相关, 就等于说 X 决定 Y , 通常写作 $X \rightarrow Y$ 。当然, 最明显的例子就是, 一个关系表中的主键能够唯一决定表中某一行的值。然而, 其他相关也可能存在, 只是它们不是主键的结果。标准化的主要目的就是消除关系模型中不是主键造成的所有函数相关。

以下是标准化的三个主要原因, 在许多数据库分析和设计的书中经常提到它们:

- 保持数据完整性。这个原因, 也许是最重要的原因, 足可以解释为什么要实现标准化。因为数据只被贮存一次, 所以它保持正确和一致性。换句话说, 不必保存数据的多个备份。否则, 相同数据项的各种备份可能不再同步, 并且可能最终要求繁重的应用程序控制, 因为RDBMS的自动完整性机制不能起到调整作用。许多遗留系统正处于这种状态。
- 建立一个尽可能与应用无关的模型。换句话说, 标准化只是增强这样一个概念, 即关系模型应该是数据驱动的, 而不是程序驱动的。对大部分数据库来说, 这意味着改变处理需求时数据库设计能保持稳定和原封不动。应用程序的要求应该和数据库逻辑设计无关 (尽管像你稍后将看到的那样, 它们对物理数据库的设计很有意义)。
- 减少存储需求 (并且也要不断地为提高查询性能打下基础)。除了外键, 全面标准化消除了关系设计中所有冗余的东西。不必要的数据备份同样要求不必要的次级存储需求。另外, 存有及可能被搜索的数据越多, 需要的总系统时间就越多, 因此性能也就越差。

2.3.1 使用一个标准化例子

在前面的章节, 你跳过了原子数据要求 (即信息原则) 怎样成为第一范式 (1NF) 的子集, 这部分将再次强调它, 并更准确地定义 1NF。

1. 第一范式

第一范式 (1NF) 不包含重复组, 这等于说贮存在一个单元里的数据必须是一个单一、简单值, 而且不能保留一条以上的信息。为了清晰, 信息原则不允许在一个列中有重复组, 而 1NF 额外要求在一行中不能有重复组, 不管是重复的列还是列中含有的重复信息都不允许。

表2-1列出了去年人口增长率不低于 5% 的城市所在的州, 因为所有城市的信息存储在一个

重复组中，这个表是不标准的或不满足 1NF 的。首先，你怎么能确切知道城市右边列中的人口和百分比是属于那些城市的？当然，你可以给它们假定一个顺序，但是这违反了列是没有顺序的这一基本关系法则。更糟的是，不得不使用数组，这要求终端用户知道并使用一个物理数据结构，如数组。这肯定产生不出良好的用户界面。

表2-1 拥有人口增长率大于等于 5% 的城市的州

STATE	ABBREV	SPOP	CITY	LPOP	CPOP	PCTINC
North Carolina	NC	5M	Burlington,	40k	44K	10%
			Raleigh	200K	222K	11%
Vermont	VT	4M	Burlington	60K	67.2K	12%
New York	NY	17M	Albany,	500k	540K	8%
			New York City,	14M	14.7M	5%
			White Plains	100K	106K	6%

为让它满足 1NF，把重复组从交叉列移到行中，表 2-2 以 1NF 的形式显示了同样的一个表，该表用 STATE 作为主键。然而，这个表还是有问题。如果要更新或删除州的信息，你必须访问许多行，并要在程序上保证它们的一致性（DBMS 不做这些）。如果要插入城市的信息，你必须同时加入州的信息。如果删除一个州的最后一个城市，那么州的信息也将随之删除等等。问题出在那里呢？过一会儿你就会看到。

表2-2 第一范式形式的州和城市

STATE	ABBREV	SPOP	CITY	LPOP	CPOP	PCTINC
North Carolina	NC	5M	Burlington	40K	44K	10%
North Carolina	NC	5M	Raleigh	200K	222K	11%
Vermont	VT	4M	Burlington	60K	67.2K	12%
New York	NY	17M	Albany	500K	540K	8%
New York	NY	17M	New York City	14M	14.7M	5%
New York	NY	17M	White Plains	100K	106K	6%

为了实现更高的标准化级，需要一个非键列。狭义的非键列定义是：一个不是主键的一部分的列；广义的非键列定义是：不是任何候选键的一部分的列。这种情况下，应选择狭义定义。本质上，一个表的列集合可以看作是由一个主键和剩余项组成，剩余项的任意一部分都是非键列。

2. 第二范式

第二范式（2NF）没有不完全相关，每个非键列都依赖于全主键，如果主键是复合键，则包括它所有的列。表 2-2 当前并没有遵循这个准则。城市信息不依赖于州信息，即：所有的城市列（CITY、LPOP、CPOP 和 PCTINC）不依赖州的名字（STATE），因此，你可以把它们拆分为两个表（表 2-3 和表 2-4）。这意味着州和城市尽管有关，但它们是单独的实体，并因此应该是不同的表。

表2-3 第二范式（2NF）形式的州

STATE	ABBREV	SPOP
North Carolina	NC	5M
Vermont	VT	4M
New York	NY	17M

表2-4 第二范式(2NF)形式的城市

CITY	ABBREV	LPOP	CPOP	PCTINC
Burlington	NC	40K	44K	10%
Raleigh	NC	200K	222K	11%
Burlington	VT	60K	67.2K	12%
New York City	NY	14M	14.7M	5%
Albany	NY	500K	540K	8%
White Plains	NY	100K	106K	6%

3. 第三范式

第三范式(3NF)没有传递的相关性。没有依赖于其他非键列的非键列，如果一个表所用的非键列都依赖于键、全键，并且只依赖于键，那么这个表就是 3NF形式的表。如果消除重复组后，每个非主键列都依赖于键和全键，这是 2NF，只依赖于键则是 3NF。城市表（表 2-4）没通过这种检测，因为 PCTINC（增长率）列依赖于 CPOP（当今人口数）和 LPOP（去年人口数量）。实际上，它是这两个列的函数，这类列叫做衍生列，因为它是从其他已经存在的列中衍生而来的。但是，所有的这些列都是非键列，直接的解决办法是删除 PCTINC列并且在运行时计算它，如果访问它的频率很高，最好使用一个视图。还有，在你的州表（表 2-3）中，SPOP（州人口数）依赖于 ABBREV（简称），因为这是一个候选键，尽管它不是主键。表 2-5、表2-6和表2-7显示了解决方案，现在你有了3个 3NF形式的表。

表2-5 第三范式(3NF)形式的州

ABBREV	SPOP
NC	5M
VT	4M
NY	17M

表2-6 第三范式(3NF)形式的州名

STATE	ABBREV
North Carolina	NC
Vermont	VT
New York	NY

表2-7 第三范式(3NF)形式的城市

CITY	ABBREV	LPOP	CPOP
Burlington	NC	40K	44K
Raleigh	NC	200K	222K
Burlington	VT	60K	67.2K
New York City	NY	14M	14.7M
Albany	NY	500K	540K
White Plains	NY	100K	106K

2.3.2 继续讨论范式

Boyce Codd 范式 (Boyce Codd Normal Form, BCNF)包括不反转的不完全相关（有时不太正规地称之为 3 1/2NF），主键和它的任何部分都不依赖一个非键属性。因为你取的是非键

的严格定义，3NF考虑候选键问题，而且你的表总是 BCNF形式的。

还有第四范式 and 更高级范式。在学术上，标准化理论已超出 BCNF许多级了，通常数据库分析和设计的相关书籍已达到 5NF的高度。4NF处理多值相关（MVD）问题，而5NF处理连接相关（JD）问题。尽管这些范式的理论稍稍超出了本书的范围，但你应该知道如果每个 MVD都是一个FD，那么这个表是4NF形式的表；如果每个JD都是它的关系键的结果，那么这个表是5NF形式的表。

第七、第八范式等在一些论著中有介绍。另外，可供选用的范式例如域键范式（Domain Key Normal Form, DKNF）已经被发展，与当前标准化理论并行，或包含了当今的标准化理论。

建议：至少要为BCNF而努力，然后必要时会得到物理数据库设计补偿，这将引导你进入下一个主题。如果可能，研究第四范式和第五范式，设法通过你的努力达到这些级别的标准。你作为一个数据库系统管理员的目标就是尽可能地提高标准化程度，然而使用尽可能少的实体来实现它们。这是一个挑战，因为，通常标准形式越高，产生的实体越多。

第3章 物理数据库的设计、硬件和相关问题

本章要点：

- 理解应用类型
- 使用定量评估
- 使非标准化
- 理解存储分层结构
- 理解RAID
- 理解DBMS中的瓶颈
- 选择平台
- 操作系统集成和通用内存 /CPU建议
- 物理设计原则和常用硬件设计建议

3.1 理解应用类型

在学习物理数据库设计以及以后进行的性能调整之前，了解主要的应用类型是很重要的。首先，考虑术语“事务”和“查询”。在数据库理论中，广义地讲，事务（transaction）就是一个单一的、原子的SELECT、INSERT、UPDATE或DELETE语句。但是，考虑到应用类型，事务被更为宽松地定义为一个业务处理过程，可能会包含多条INSERT、UPDATE或DELETE语句。另外，DML实际指的就是SELECT、INSERT、UPDATE和DELETE语句。然而，像文 中的事务一样，DML通常只被用于表示INSERT、UPDATE和DELETE操作。总的来说，DML和事务通常意味着只写或只修改。为区分作为只读的SELECT操作，要用到术语“查询（query）”。

看一下下面这些工业约定，尽管它们十分容易引起混淆，并且实际上和它们真实的定义相冲突。这里指出，在数据库系统应用领域有三个主要的应用类型：

- **OLTP（联机事务处理）**——OLTP系统是一个包含繁重DML的应用，其面向事务的活动主要包括更新，但也包括一些插入和删除。经典的例子是预定系统，例如那些用于航空公司和旅馆的系统。OLTP系统可以允许有很高的并发性（在这种情况下，高并发性通常表示许多用户可以同时使用一个数据库系统）。
- **DSS（决策支持系统）**——DSS系统通常是一个大型的、包含历史性内容的只读数据库，通常用于简单的固定查询或特别查询。DSS常常按某种方式变成一个VLDB、DM或DW，这些方式在第1章“数据库、DBMS 原理和关系型模型”中已经讨论过。DSS一个很好的例子是某个组织的局域网后面的数据库。
- **批作业处理**——批作业处理系统是作用于数据库的非交互性的自动应用。它通常含有繁忙DML语句并有较低的并发性（在这种情况下，较低的并发性通常表示少数几个用户能够同时使用一个数据库系统）。事务查询的比率决定了如何物理地设计它，经典的例子是与DW有关的成品数据库和可操作数据库。

一些不太常用的应用类型包括如下：

- **OLAP（联机分析处理）**——OLAP系统可提供分析服务。这意味着数学、统计学、集合以及大量的计算，一个OLAP系统并不永远适合OLTP或DSS模型，有时它是两者之间的交叉。另外，有些人简单地把OLAP看作是在OLTP系统或DSS之上的一个扩展或一个附加的功能层次。ROLAP代表关系型OLAP，尽管这个术语在分类方法上并不能真正增加什么。一个OLAP工具常被严格地和一个MDD（在第1章讨论过）配对使用，有时它只是简单地放在一个修改的RDBMS上。通常，地理信息系统（Geographic Information Systems, GIS）或有关空间的数据库和OLAP数据库相集成，提供图表的映射能力。用于社会统计的人口统计数据库就是一个很好的例子。
- **VCDB（可变基数数据库）**——这种类型的数据库通常被用作一个处理系统的数据库后端，这样就会导致在数据处理期间，数据库中的表显著地增长或收缩，其相反情况是保持不变或在一定范围变化，例如一个10%的变化。基数是指在一个给定时间里一个表中行的数目。有些表可能是静态的查询表，但多数表的记录数目肯定是高度可变的，一个好的例子就是任何记录短期活动的数据库，例如一个安全授权数据库。

3.2 使用定量评估

任何种类的定量评估都是试图量化或测量一些过程或产品，对于数据库来说，定量评估的两种主要类型是事务分析（有时称为量化分析，volume analysis）和筛分分析（sizing analysis）。

1. 事务分析

事务分析是简单地把数字放到数据库系统中。不同的评估方法意味着不同的事情，并应用于特定种类的数据库系统。最典型的度量或尺度，包括下列项目的最小数、平均数和最大数：

- 并发用户数目。
- 响应时间。
- 经过的时间。
- 事务数目。
- 并发程序的数目。
- 读或写的字节数。

还有更多的尺度，例如受一个操作影响的行的数目。

如果这些评估是在一个给定的时间段或动作的背景下表达出来，那么它们通常有着更多的含义。例如，知道每秒钟事务的最大数比知道事务的累积数更有用处。后者只能告诉你很少的关于数据库通常的压力或负载情况。这些数字在数据库为哪种应用服务的背景下也意味着更多的含义。

如果你的应用类型是一个OLTP系统，并发用户的数目、每秒钟的事务数目和响应时间尤为重要，因为并发是OLTP系统的首要问题。

如果你有一个批量处理系统，经过的时间和并发的程序数也许最重要。DSS可能要求你除了知道其他项目外，还应知道每个时间单位内可读的字节数。

作为一名DBA，你需要问这些问题并收集尽可能好的答案，因为这会影响到你的物理设

计和运行性能。另外，这些数字基本上就是用于衡量基准工作上的度量措施，那么，需要你努力去做的就是在系统真正建立之前，收集估计的、原型的或典型试验的数字以帮助建立系统。

2. 筛分分析

筛分分析也许是一种更广为人知的行为，甚至广泛地被所有的 DBA 采用。在事务或量化分析中，考虑到处理和数据流量，你会问“多久一次？”和“有多少？”；对于筛分分析，考虑到数据的存储，你会问“有多大”。

基本情况是：如果一个表有 n 行，每行最多有 b 个字节，那么至少需要 $n \times b$ 个字节的存储空间。当然，这没有考虑系统的开销，而且这种计算随着所选择的商品软件不同而有很明显的差异。例如，像其他软件厂商一样，Oracle 提供一套相当复杂的步骤，帮助测量一个表、索引或其他结构。建议最好是一次把这个公式放入一个电子表格中，那么你永远不必再做这件事了。每次需要测量其他的数据库对象时，你需要做的就是取出它并插入相关的输入数字，如块的大小、块参数、行数、列大小等等。这样，你可以用表、表集合、索引、索引集合为整个数据库实现部分求和和分类汇总。

另外，一个经验丰富的 DBA 会在估算数值上加上一个裕量因子来对付任何低估错误、意外的疏忽以及未预料到的变化。将有些项目（如最后估计值的大小）乘以 120%，不是没有道理的。

也要记住，你所评估的数字通常是单独基于表和索引的。如同所有的 RDBMS 商用软件一样，有许多更具结构性的因素增加了整个系统的大小（在第 5 章“Oracle 数据库体系结构”和第 6 章“Oracle 实例体系结构”中讨论过）。不管你最后得到的字节数是什么，记住这是你需要的可用空间，而不是裸盘的空间。低级的硬件和一个高级的操作系统会消耗最初磁盘未格式化（裸盘）的大小，剩下的可用空间比你认为的要少。

例如，格式化可占用一个 4GB 磁盘中 15% 的空间，仅留下 85%（即 3.4GB）。如果你最终的大小估算是 20GB，并且没有考虑到这一点，于是你买了 $5 \times 4\text{GB}$ 的磁盘，然而实际上你只有 $5 \times 3.4\text{GB}$ （即 17GB）的可用空间，那么你还可能需要其他磁盘。

3.3 使非标准化

使非标准化是指由于物理设计、性能调整或其他原因，使表的标准化形式的级别在一定程度上降低。

提示 除非你有非常好的理由，否则不要这样做。缺少磁盘存储空间而且又没有买新盘的预算或许就是一个很好理由。没有根据地判定将有一个不良性能，而不是实际存在不良性能，并不是一个好理由。实际上，即使是不良性能本身，在你的逻辑设计上也不会立即表明有放弃的必要。你所要做的第一步工作就是性能调整，非标准化一般是最后的办法。

那么，到底要做什么呢？假定在数据库中，有 30 个临时的表组成数据库的逻辑设计，这些表都是 3NF 或更高形式的。然而，当使用的时候，这些表中的三个表实际上总是要联合到一起。联合操作本身是资源密集的，并且只消耗很少的机器时间。出于对性能改善的要求，又没有其他性能调整的技术可用，那么为了长久地使用，你可能希望把这三个表预先联合成一个表，这样就可避免将来访问时的联合。把两个或多个标准化的表放回到一个表，实际上

的确降低了标准化级别或甚至完全变成非标准化的。当然，如果必须这样做，必须写入更多的过程化的和应用级的数据完整性控制，来取代 DBMS 的自动特性。

例如，如果你选择同时保留原始表和新的联合表，那么你将面临一个不麻烦的任务：使它们保持同步。然而，如果更新非常频繁（每小时或更短时间），并且要求联合数据的当前值，就不得不使用其他方法了。

另外一种方法就是在标准化的基表上创建一个非标准化的视图。然而，这个方案的性能与简单地拥有三个原始表相同或更糟。它唯一的好处就是用户接口简单。那么，尽管非标准化可能在一些静态数据事例中有帮助，但常常无法解决许多动态数据问题。在这种情况下，如果非标准化不能工作，Oracle 聚簇这样的方法可能就足够了。这第 5 章“Oracle 数据库体系结构”、第 16 章“性能调整基础”以及第 19 章“调整输入/输出”中将详细讨论。

最后，非常普通的一个非标准化例子出现在 DBA 必须处理时间序列时。时间序列通常必须被非标准化的原因，是因为时间必须总是表主键的一部分。这是正确的，因为大多数存储的数据是与时间相关的。

一个较好的例子是一个贮存卫星反馈数据的数据库，这样的数据带有时间信息，它是数据库中大多数表的主键的一部分。对于任何给定的包括时间信息成分的表，所有组成主键的其他列（如卫星 ID）对于每个不同的时戳来说是重复的。换句话说，如果数据库每小时从一个卫星下载 100 行数据，那么在 8 小时内，你要毫无必要地重复 800 次以存储 800 行数据以及其他的卫星信息，这是对存储的极大浪费，尤其是考虑到可能有多个卫星或更密集的取样频率。

典型的解决办法就是通过把数据从行方式转换为列方式，实行非标准化。现在你有 100 个带有时间信息的列，并且你把行数从 800 减少到 8，或者使用一个取样间隔为 100 的因子。存储的减少，尤其是行的减少，总能有助于提高搜索性能，因为在扫描一个表或一个索引时，扫描的行数减少了。然而，这种类型的非标准化，尽管通常是必要的，结果却导致了一个不标准的表，因为时间信息被作为重复组列贮存。因此，在先前为主键组成部分的列上，你不能用外键约束。相反，你必须依赖于检查约束、过程、触发器以及可能的增补应用完整性来处理有关操作。你应从这个特殊的例子中学到教训是：如果出于性能因素，你必须实行非标准化，那么你必须完整性管理上付出代价。

3.4 理解存储分层结构

数据库系统管理员能够学到的最重要的事情之一就是存储分层，以及与其相关的折衷方案。这比其他任何事情更能帮助你解释许多关于物理设计和性能调整问题。存储分层背后的一个关键思想就是机电设备的不足（electromechanical disadvantage）——即任何有引擎或运动机件的东西，天生就比纯电子设备的速度慢。

图 3-1 表示了存储分层的现代解释。很清楚，内存比磁盘速度快。沿这个金字塔向上，速度提高（存取时间），费用也增加（每单位），存储量降低。

你想贮存的越多，必须付出的也越多，尤其是你希望得到更快速度时更是如此。但是，你可以取一些折衷方案。对于一个历史数据系统，你可以把比较旧的数据存放到联机光盘或至少较慢的磁盘上，而将更老的数据存放到联机或脱机磁带上。如果你特定的存储金字塔能让所有的存储介质自动联机、下载到磁带，并且你有按照要求访问它的软件，那么这就是一个分层存储管理（Hierarchical Storage Management, HSM）系统。有专门研究这类硬件和软

件的厂商。历史数据系统能允许向金字塔底部移动，因为相对于数据流通，速度并不重要。但是，对于实时系统（例如飞机导航等），将存储加快是最优先的考虑因素。许多商务应用的一个合理的方法就是把重要、当前的数据放在 RAID或快速磁盘上，而将其他东西放在较慢的磁盘上。

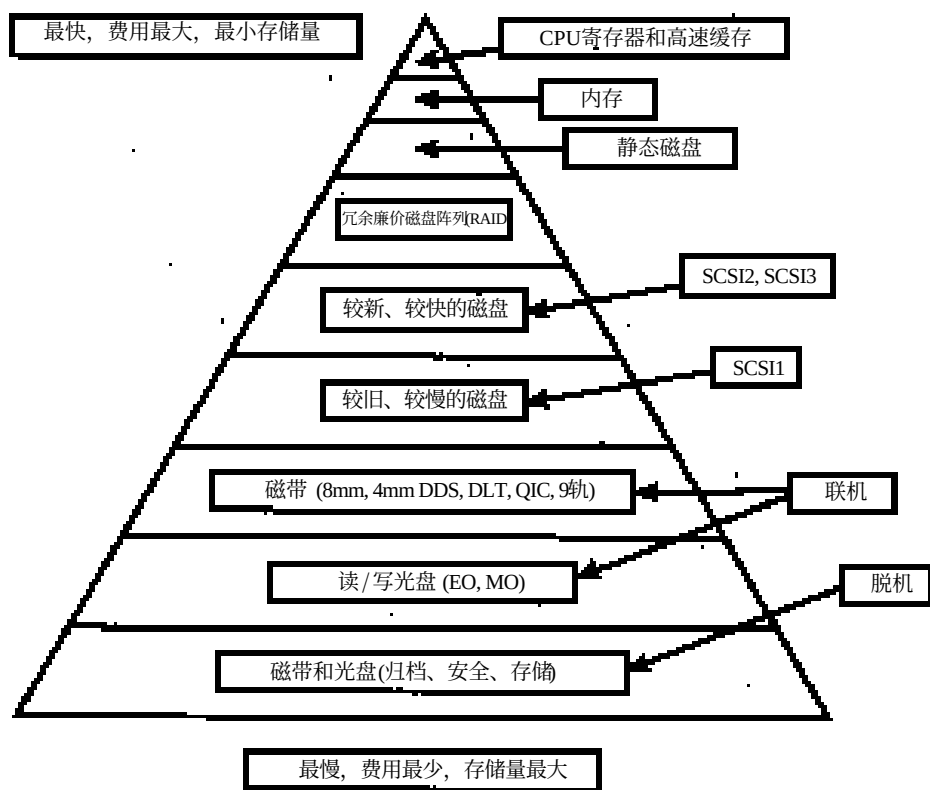


图3-1 存储分层

3.5 理解RAID

RAID（冗余廉价磁盘阵列，Redundant Array of Inexpensive Disks）也许是一个误称。从一开始，组成 RAID的磁盘从未真正地比普通的 SCSI（小型计算机系统接口）盘便宜过。另外，RAID需要特定的硬件、软件或两者皆有来工作，这都增加了成本。所以 RAID中的I（廉价）不十分确切。

RAID是一组能并行工作的磁盘，能够减少 I/O时间，其效果主要取决于组成磁盘组的磁盘数量这一因素，对数据库来说这是件重要的事情。RAID通过人们所熟知数据条（striping）技术来实现并行工作，与普通磁盘存储有所区别，后者是串行工作的。这只是以并行方式跨越多个磁盘使用数据条来写一个文件或文件块。数据条是一个物理数据块的一定倍数，是磁盘I/O操作的最小单位。通常，一个数据块由 512个字节组成，这对于大多数 UNIX、VMS和 DOS/NT系统来说都是正确的。

RAID还提供实时磁盘失效恢复，这是数据库的另外一件重要的事情。RAID可以通过奇偶信息提供这一级别的可用性，这种信息也被写到一个或多个磁盘中。奇偶校验、校验和、

以及错误纠正是通过一定的数学算法实现的，这些算法能重建在丢失磁盘上的丢失数据。RAID既支持大磁盘组也可以支持小磁盘组，可以有助于不同的数据库应用类型。例如，DSS系统使用较大的磁盘通常运行得更好，而OLTP系统使用较小的磁盘运行得较好。RAID分为很多形式，被称为级别。

当今在工业上最常使用的RAID级是0、1、3和5。所有其他的级别都没有被使用或只是这4个级别的一些变种。供应商提供的RAID级达到了7或更高，但在实际工作中很少看到它们。RAID 0是没有校验的基本数据条。就是说，你得到了性能上的优点但是没有校验。当纯粹为了速度而可用性并不太重要时，这是很好的。

RAID 1就是大家所知的镜像（mirroring），或者有时叫双工（duplexing），在镜像中同样没有校验。实际上，对于偶数数目的磁盘组，其中的一半是保存实际的或主要的数据，另一半基于磁盘级保存数据的备份。实际上，两部分磁盘是同时写入的，并且有时这意味着写入时性能的损失。另一方面，在读的时候，如果软件被设计成能读多个盘并且能将数据流合并使用，那么实际上读操作是可以被加速的。

RAID 3是数据条，带有一个单一、专用校验盘。你可以丢失一个数据盘，并且仍然可以用校验盘来恢复它。不过，校验盘是一个单一故障点。如果它丢失，将是致命的，这依赖于软件或硬件以及它是如何写入的来处理这类事件。

RAID 5也是带有校验的数据条，只不过不是用一个单一、专用磁盘，它将校验信息与数据一起保存在所有的磁盘上。校验信息和数据一样受到保护，并且没有单一的故障点。和RAID 3一样，RAID 5能够在单个盘的丢失时恢复数据。然而，3和5都不能恢复两个磁盘的缺失，因为丢失了太多的校验信息。

工业上很少销售或实现RAID级别2、4、6。这些级别的校验信息背后的数学计算量太大而不实用。

关于RAID最后要说明的就是：对于数据条，RDBMS和OS工具及技术手工能做的事，RAID都能做。但是，它通常做得更好，它有精密形式的数据条并提供更多的特性，如分区容量（数据条集合）、可调整大小的数据条等等。另外，尽管RAID的数据条部分可以毫无困难地以一种总体的方式来实现，但你必须管理这些数据条。RAID通常擅长于此，并提供一个虚拟文件系统（Virtual File System, VFS）的单一文件接口，协调OS文件级的命令和工具并且提供了其他特性，如大于物理磁盘尺寸的文件。

最后，尽管你可能可以管理某些形式的数据条，但是除了有标准的备份和恢复机制外，并没有其他保护措施。但是RAID可以容忍丢失一张磁盘并进行恢复，你要试图自己编写这样的软件实在是太复杂了。

提示 当可用性不是一个主要关心的事时，可以使用大致的、手工的RDBMS或OS数据条。否则，当你需要它提供的可用性并希望使用更为精密的数据条以获得更佳性能时，或需要它的其他特性（如巨型文件）的时候，要使用RAID。

3.6 理解DBMS中的瓶颈

过去，DBMS常常被指责为I/O限制或硬盘限制。这就等于说一个DBMS由于它对磁盘的读出和写入，导致它在系统中成了瓶颈。研究一下你刚刚学到的存储分层就会发现这几乎是不相关的。访问磁盘的速度要比访问内存或CPU的速度慢得多，这实际上就是过去的数年里

许多数据库应用的情况。对于 DSS、VLDB和DW来说，这尤为正确。因为，对于一个查询来说必须移动大量的数据（GB）。然而，这并不永远是事实。OLTP和OLAP（在线分析处理）系统常常是内存限制或CPU限制。

如果一个数据库硬件服务器有一个较低的内存对硬盘的比率，意味着应用是内存短缺的，数据库性能降低显然是由于存储数据和缓存代码的内存空间不足。如果一个 OLAP类型的数据库应用（比如一个科学数据库），那么速度是最重要的事。因此，这种类型的应用如果运行在带有相对较弱 CPU的服务器上，按照用户的期望（如响应时间）来讲，性能肯定是低劣的。这里需要知道的就是在任何系统（包括数据库系统）上，瓶颈现象是与应用类型相关的。

最后，在客户/服务器数据库系统中，网络系统是整个系统中最慢的组件，甚至比硬盘还要慢。为了有效地发挥一个客户/服务器系统的功能，必须有合理的应用分段并且网络系统一定不能超载。另外，和其他资源一样，网络系统必须被合理设计，以减少竞争。还有，为利用当前网络的能力，网络硬件和软件应该实行现代化。你将在第 16章“性能调整基础”中再次了解瓶颈问题。

3.7 选择平台

当选择适合于应用的硬件数据库服务器和操作系统的类型时，有许多事情需要考虑，包括如下：

- 应用类型 —— 正如已讨论过的，OLTP、DSS、批任务处理或其他东西。
- 定量估计 —— 正如已讨论过的，关于事务的、量化的和筛分的数值。
- 当前环境 —— 主要指你现在的主要平台是什么。
- 趋势 —— 对于你当前的环境，工业发展的前进方向是什么，以及你正在考虑什么。
- 处理的需要 —— 需要是实时的、对时间要求不太严格的还是周期的？用你的事务和量化数值来说明并发及加载。
- 存储需要 —— 用你的估算数值来确定对裸盘的需要，用事务和量化数值来确定你是否需要RAID。
- 人员能力 —— 你认为哪些人员能力要求与你的人员的基本能力相差甚远。
- 时间限制 —— 在给定时间里，你能移植或开发出系统吗。
- 衔接和合伙 —— RDBMS供应商是否与OS及硬件供应商有融洽的伙伴关系。
- 集成 —— 与前一条相似，但是如果不考虑他们之间的业务关系，那么将所有的工作放在一起会有多好呢？

目前主要的RDBMS供应商是Oracle、Sybase和Informix。CA在Ingres上做的多一些，微软只提供NT环境的SQL服务器。这里不考虑小型、桌面数据库。现在，只考虑 Oracle，UNIX和Windows NT是它的两个主要平台。对UNIX来说，平台主要是Sun Solaris。当然，Oracle也可移植到其他形式的UNIX平台。Oracle在MVS和VMS系统上也有一些大的安装数量。然而，写本书时，Solaris和Windows NT是它目前最优先选用的系统。

至于硬件，不算 Intel公司的机器，Sun公司的Solaris系统是唯一的真正选择。Sun公司的机器已经经历了几代变化，但是它们均基于 RISC的SPARC芯片。Windows NT主要使用在Compaq机器上，但也应用在DEC和HP的机器上。在DEC上，关于Windows NT最重要的事就是DEC支持Alpha和Intel机器。Compaq最近对DEC的购买，简化了软件提供商这种进退两难

的局面。Solaris和Windows NT都可以在多处理器系统上运行。但是，Windows NT只能处理有限数量的处理器（14），并且用于处理Oracle的处理器能力大约为4个。Solaris能非常好地扩增，可以超过20个处理器，即使对于Oracle也是如此。虽然Sun公司比Compaq、HP公司更容易提供更大的选择范围，但是磁盘存储选择几乎是相同的，当然DEC公司也能提供和Sun一样大的存储。

不必更深一步地比较硬件，这里有一个建议：作为一套常规原则，在准备选择一个平台之前，列出它们的各项指标，并充分考虑这些指标。如果你的主要环境是DEC/VMS，并且你有可能保留这种环境达几年时间，那么，你的下一个平台最有可能还是DEC/VMS，特别是如果你只打算增加一台机器时更是如此。如果你准备为整个部门购买机器而且风险也不是关键问题，这是考虑选用Windows NT或者UNIX的最佳时机。

如果你只需一台机器，且服务于相对较小的低于10GB的数据库，考虑选用NT较好。相反，如果你需要一台机器，但是对存储要求较大，或者对性能要求较高，考虑选用UNIX较好。最后，如果你没有一个真正的主导环境，或者你正在启动一个新计划（或这个新计划在某个方面影响你的购买），务必既要考虑Windows NT也要考虑UNIX，还要考虑除了环境因素之外的所有因素。

当前，最后一个可供考虑的是Linux。它起源于UNIX，当前得到一些支持，是Windows NT的一个直接竞争者，因为它既是一个操作系统，又是一个网络操作系统（Network Operating System, NOS）。因为它还有待于去建立硬件、应用和围绕它的技术支持，所以还没达到大多数工业销售的可信级；但是，它的普及程度还没有达到巅峰，决不可小视。

3.8 操作系统集成与通用内存/CPU建议

除了辅助存储、硬盘和RAID外，当讨论存储分层时，需要强调的是：你需要考虑其他的硬件问题和操作系统组件，如内存和CPU。

注意 内存通常是指核心内存、物理内存、主存或随机存取内存（RAM），这些都是同一事物的不同名词，所以这里只说内存。

本质上说，内存是一种非常快的电子存储器，它贮存指令和数据。对于一个DBMS来说，最重要的事情就是操作系统能够让出一些内存给它，那么，DBMS用它来做自己要做的事。因此DBMS有时被称为微型操作系统、操作系统中的操作系统，或一个建立在操作系统上的操作系统。实质上，虽然要服从操作系统并与之协作，但DBMS在管理和获取资源需求方面，能够照顾自己。这些事情通常是通过一个叫做共享内存功能来实现的，尤其是在UNIX环境中（详情参见附录A“在Solaris系统上的Oracle”）。

加锁是DBMS的一个关键组成部分，它也是通过内存结构来控制的。共享的资源没有进程竞争的危险，因为它们被依次使用。DBMS控制它自己的锁，部分工作和操作系统一起来做，或将加锁任务交给操作系统完成。

当操作系统将它的部分内存出让给构成DBMS的一个进程时，DBMS从那里得到内存，并在这个内存空间里贮存它自己的结构（代码缓存）和数据（数据缓冲）。在第5、6章中将讨论Oracle的结构，没有深入了解Oracle结构的人需要看一看在那里究竟讨论了些什么。Oracle的内存存取基于它的系统全局区（System Global Area, SGA）的资源分配。SGA包含一个叫做数据库高速缓冲（高速数据缓存）的结构和共享池。共享池包含库缓存（高速代码缓存）以

及数据字典缓存。撤销（回滚）块存储在数据块缓冲内，重做块存储在它自己的重做日志缓冲（日志缓冲）部分。通过 Oracle 的参数文件 ‘init.ora’，这些组成部分都是可以配置的，后面将会详细介绍。

关于 CPU 的使用，RDBMS 已走过了一段长路。像前面提到的那样，多数过去的数据库系统趋向于 I/O 限制。然而，有了 VLDB 和 OLAP/MDD 系统，越来越多的数据库变为内存限制或 CPU 限制。有了 VLDB，由于内存数量太小难以容纳大量数据，内存成为瓶颈。对于有大量的分析或科学的系统或者 DW 系统，由于庞大的、并行的计算要求，CPU 可能会成为瓶颈。

在近十年中，随着多处理器机器的出现和改进，许多事情发生了变化。现在，非常大的内存（≥几十 GB）已成为可能，而且，CPU 的体系结构和速度也大大提高。在写本书的时候，目前的字长是 32 位和 64 位，时钟速度大约是 500MHz，并且很快将达到 1GHz。这些 CPU 拥有密集的管线体系结构，允许每个指令周期（CPU 步）执行多条指令。最重要的是新 RDBMS 软件也跟着推出。

Oracle 和其他主要的 RDBMS 供应商一样，为充分利用这些先进硬件，有可能重新编写它们的过时代码。除了共享内存和非常大的内存之外，多处理器是近年来主要的提高和改进，目前存在着两种主要的多处理器：

- 对称的多处理器（Symmetric MultiProcessor, SMP）
- 大规模并行处理器（Massively Parallel Processor, MPP）

在 SMP 机器中（例如那些 Sun 公司提供的），CPU 使用共享内存和其他内部硬件项目（如总线）。现在 SMP 已达到 64 个处理器。MPP 机器中有一个完全不共享体系结构，并且类似于微型局域网（micro-LAN）或一个盒子中的局域网。MPP 可以拥有数百个或数千个处理器。

现在，RDBMS 软件或是完全多线程的或是伪多线程的，以便利用多处理器机器的处理能力。对一个操作系统来说，RDBMS 就是一个或多个进程。多线程是软件的一种能力，在相同的父进程环境下，能够运行多个子进程或线程。例如，Sybase 和 Informix 就是完全多线程的。当使用多线程服务器（MultiThreaded Server, MTS）选项时，Oracle 是伪多线程的；否则，它是单线程的。数据库系统管理员只需要知道 CPU 的数目，并且能够配置 Oracle MTS 就可以了。其他的 Oracle 参数受 CPU 数目的影响（例如，被称为闕锁的特定加锁结构）。

3.9 物理设计原则与常规硬件设计建议

物理数据库设计的主要原则是什么？物理数据库设计实际上就是预调整，或者说是调整的第二个阶段（逻辑数据库设计是第一阶段）。因此，主要的物理数据库设计原则基本上与主要的性能调整原则相同，只不过你正在处理的数据库是在数据库创建之前或期间而不是创建之后。有许多设计原则，但主要包括如下几条：

- 分而治之——分区、分段和并行是分而治之算法设计方法的扩展。如果一个处理时间被分成数块并能够同时运行，那么就说它是可并行的。对此的主要要求是处理的每个部分必须是数据无关的；即：不管任何其他程序块是否已结束，某一程序块都可以开始。例如把一个求总数的运行时间很长的查询程序分割成两块，在不同的 CPU 上运行它们，然后把它们的部分和相加，得到最后的结果。实际上，这就是 Oracle 的并行查询功能所提供的东西。
- 预分配和预编译——静态分配和固定分配都表示相同的意思，即预分配。换句话说，

提前分配你的资源，而不是动态地由软件为你做这件事。这通常导致额外的计算和 I/O 开销，而这总是不受欢迎的。预编译程序比解释程序执行起来要节省实际时间。虽然 DBMS 缓存控制许多事情，但是 DBA 应该注意可以做些什么以进行补充，例如编写一些通用、可重用的过程并把它们驻存在内存中。Oracle 的 KEEP 操作可实现后者，KEEP 操作将在后面的第 18 章“调整内存”中讨论。

- 前摄 —— 预测主要的问题。遵循帕拉图规则：解决可能引起 80% 麻烦的 20% 的问题，这也被称为 20/80 规则。用统计的方式来说就是所有的性能问题并不是由各种原因平均引起的。你应尽力预测最主要的问题，并且把它们设计在系统外或至少补偿并围绕它们进行设计。考虑一个大型串行运行的批处理作业系统，只有几个主要程序，每个都可以插入、更新或删除大量数据。这里潜在的问题与事务日志有关，尤其是撤销日志增长得非常快，甚至有可能耗光存储空间。对于 Oracle，撤销日志就是一组可用的回滚段。对于一个 DBA 来说，合理的设计将至少有一个非常大的回滚段，而不是能够控制产生的最大数量回滚数据的 SYSTEM 表空间。
- 批量、块和批处理 —— 使用大量传送。成批的事务在一起时进行批量处理才有意义。这就意味着对硬盘和网络 I/O 这样的事物来说，常用的最好方法就是大量传送：将有着相同的起源和终点的 I/O 操作组合在一起。这项工作通常是 DSS 和批量处理系统而做的。例如，用户可能经常从一个极大的表中选择多个行。因为这些选择返回如此多行，因此他们从来不用任何可用的索引。如果这时没有得益于任何并行硬件或软件，那么表应该连续地存放在一个单独的、大的磁盘上。然后，你可以增加数据库逻辑缓存的尺寸，以同时读许多物理数据块。例如在 Oracle 中，你可以把 DB-BLOCK-SIZE 设到平台所支持的最大值（如在一个 Windows NT 机器上设到 8K），这样一次读请求便读到尽可能多的数据块。这些都源于一个简单的事实，即在几乎所有机电系统中，启动耗费是相当大的。一个类似的网络情形是在每一个包里发送尽可能多的信息，因为这会提高成本效率。
- 合理地分割应用 —— 这可以看作是“分而治之”原则的小主题，它的区别在于所强调的东西是整个的环境和应用程序，而不只是它背后的数据库（后端）。在分布应用前，考虑客户、网络和服务器的相对性能能力。把功能放在逻辑上能被执行的地方，例如：在一个交互式的应用中显示和表现的任务显然是属于客户的；而作为数据库对象的数据库事件应在数据库中处理（通过 DBMS），而不是由一些前端的软件处理。在 Oracle 中，使用了触发器和存贮过程进行这种设计。

物理设计和调整的主要目标是消除或最大限度地减少竞争，竞争（contention）就是两个或多个软件争夺同一个资源。常识告诉你有些东西必须等待。为了解决竞争，实践下列的规划建议：

- 把表和索引分开。
- 把大的表和索引放到它们自己的盘上。
- 把经常联合的表放在单独的盘上，或把它们聚合。
- 必要时，把不常联合的表放在相同的盘上（就是说，如果你的磁盘不多）。
- 把 DBMS 软件与表和索引分开。
- 把数据字典与表和索引分开。

- 如果可能，把撤销（回滚）日志 和重做日志分别放到它们自己的盘上。
- 为撤销日志或重做日志使用 RAID 1（镜像）。
- 为表数据使用 RAID 3或5（带有校验的数据条）。
- 为索引使用 RAID 0（没有校验的数据条）。

正如前面所讨论的，在某些情况下，如果 RAID是不可用的，那么手工数据条可以在某种程度上满足需要。但是，一般来讲，它不够灵活、安全或快速。另外，当前面的列表提到要进行分离时，那是表示把它们放到单独的盘上；但是，这又可能进一步意味着把它们放到单独的磁盘控制器上。控制器越多，性能和安全性就越理想。

第4章 Oracle的解决方案

本章要点：

Oracle历史回顾

Oracle是一个DBMS

Oracle 是一个RDBMS吗

再论物理规划

Oracle的未来

4.1 Oracle历史回顾

Larry Ellison是Oracle公司的董事长兼执行总裁。1977年在加利福尼亚的Redwood，他和Robert Minor一起建立了Oracle公司。在IBM的System/R（关系模型）的基础上，他们推出了第一个关系型数据管理系统，这也是第一个使用IBM的结构化查询语言（SQL）的RDBMS。

今天，Oracle RDBMS可用于几乎所有的操作环境，包括IBM大型机、DEC VAX小型机、基于UNIX系统的小型机、Windows NT以及一些专用硬件操作系统平台；该公司无疑是世界上最大的RDBMS供应商。在写此书时，Oracle 是世界上最主要的信息处理软件供应商，也是仅次于微软公司的世界第二大软件公司，年销售额为75亿美元，Oracle在世界范围内大约聘用了三万六千多名专业技术人员。

Oracle最初的大部分产品都出自IBM DB2以及SQL/DS家族的产品，后者是DB2直接子代产品。事实上，DB2一直是在MVS上的RDBMS。与之相比，Sybase和Informix（它们表面上很相近）都是从Ingres发展而来的。有了Oracle7，Oracle公司在近几年内稳固地发展成最杰出的RDBMS供应商，它的市场从政府部门和个人应用，从国内范围到国际市场。起初，Oracle把精力集中在RDBMS的研制开发上。最近，它成功地开发了许多横向和纵向的产品，例如Oracle应用服务器。另外，在过去的数年中，Oracle提供了许多可用的开发工具，这也许是它的事业稳步上升的原因之一。举例来说，这些工具包括Designer/2000计算机辅助系统工程（CASE）工具和Developer/2000开发包。这些后来的工具现在被简单地称作“设计器”和“开发器”。随着工业进步，这些工具正逐渐变得基于Web。另外，Oracle的应用软件（包括广泛使用Oracle Financial（财务软件））为Oracle带来了大量收益，补充了他们核心的RDBMS生意。

注意 Oracle开发了许多产品，它们被统称为Oracle应用软件。这些产品包括Oracle财务软件（Oracle Financial）、Oracle制造业软件（Oracle Manufacturing）、Oracle人力资源软件（Oracle Human Resources）、Oracle自动控制软件（Oracle Automotive）以及其他的应用软件，例如Oracle可以提供保健系统。Oracle已经拥有基于不同客户环境的Oracle RDBMS服务器工具，这些工具包括Discoverer、Express及其他的软件。

4.2 Oracle是一个DBMS

根据实际的准则，Oracle是一个DBMS，因为它维护了一个数据库、提供多级安全保障、

提供了确保数据完整性的手段、处理并发事件并提供了一种语言接口。Oracle在它的SYS和SYSTEM模式中维护数据字典，SYS和SYSTEM模式贮存在SYSTEM表空间中。Oracle提供一个数据库级的注册管理，如果需要可以进行操作系统级验证，并可以进行角色、权限和简档验证。它还通过权限和角色，提供了使用普通ANSI SQL授权和撤销的能力。当然，SQL视图总是可用的。简档更多地处理资源的使用。完整性是通过对主键和外键的声明完整性来控制的，因此你只需声明什么是键以及什么不是键，而不必编写程序去实现它。

Oracle为每个主键都生成唯一的索引，用户可以随意地为一个外键创立索引。Oracle也提供了常用的SQL约束和触发器来处理其他的完整性问题。在缺省情况下，在行级上加锁，或者可自由选择在校级上加锁。行级锁可带来更好的并发性，尤其是在OLTP应用中。重做日志存储在数据库外部，并被系统缓存起来，存放在系统全局区中（SGA）。撤销（回滚）的数据存储在数据库中，并同样缓存到系统全局区中。

Oracle是与条目级SQL-92兼容的，也就是说它能满足最新的ANSI SQL标准的第一级。Oracle提供了SQL*Plus，这是一个交互的SQL命令解释器，它也能运行非交互的脚本以及称为PL/SQL的过程化程序SQL语言，存储过程和触发器被写进PL/SQL中。

最后，Oracle还提供预编译（对于嵌入SQL）和3GL函数调用接口，称作Oracle调用接口（Oracle Call Interface, OCI）。的确，Oracle是一种具有工业潜能和卓越伸缩性的DBMS。伸缩性以及Oracle为它的RDBMS所提供的优秀工具，也许是它目前广受欢迎的另一个原因。许多关键任务和产品环境都依靠Oracle来存储它们的数据并提供服务。

4.3 Oracle是一个RDBMS吗

现在，你会问Oracle是否真是一个RDBMS？当然，它作为一个RDBMS被销售并被十分有效地使用已有多年了，唯一存在的问题是Oracle怎样符合Codd的十二条法则，这些原则你在第1章“数据库、DBMS原则和关系模型”的1.3.1节“Codd十二条法则”中已经见过。在这点上，Oracle不是非常好，这一点也不奇怪，因为它主要的手们也无法做得更好。

看一下Oracle对于这12条法则可能的得分情况：

0=不符合规则

1/2=符合部分规则

1=符合规则

二进制巨型对象（Binary Large Objects, BLOB）是用于存储和管理多媒体数据类型的RDBMS技术。BLOB通常存储在数据库的外部（以文件的形式），把指针存放在对应列的位置。BLOB面临头两条原则的质疑，数据应该表现为存储在单元中，并且可通过“表名+列名+行主键”访问。BLOB是一种数据类型，它的物理实现对于用户来说可见的。它们的实现应该是透明的。但是，BLOB没有明显地违反头两条规则。Oracle（和其他供应商一样）前两个规则总的评分为1 1/2点，因为BLOB在第一条丢失了1/2点。这被认为是一个差的品行分数。

Oracle对于原则3（对NULL采用前后一致的处理）得分为1/2。对所有的供应商来说，在SQL中NULL的当前实现并不理想，而且已经受到Codd和另一位关系理论的开拓者C. J. Date的攻击。Codd提出一个三值逻辑，而Date也提议全部废除NULL。这场争论持久得令人生厌，而且这里也不是进行争论的合适地方。现在只需说明，在RDBMS中对于NULL的使用存在着一些非常现实的数据问题。

Oracle对于原则4（数据字典通过常规的SQL可以访问）得分为1/2。是的，Oracle只允许SELECT，而不可能使用INSERT、UPDATE和DELETE。这并不是说这种情况应该时常出现，也不是说期望这种事。然而，按照原则4，它应该变为可能。Oracle正为争取原则5和原则7~10的通过标记。

Oracle对于原则6和11得分为1/2。对于原则6（可更新视图），Oracle 7只允许某些（简单的）视图可更新。Oracle 8已对此有所改善，但也不是100%。对于原则11（分布独立），Oracle提供了数据库链、快照、对称复制以及带有两阶段提交（2PC）的分布式的数据库。然而，网络的一些部分对用户可见，两阶段提交能够工作（但不太好），而复制也有它自己的问题。大多数的抱怨集中在这样一个事实：所有这些性能要求用户对于配置的了解要比他们应该知道的多。

Oracle对于原则12得分为0，原则12是非破坏规则。这是因为通过它的一些工具，例如SQL*Loader和导入，Oracle允许越过标准的SQL路由，将数据存放到数据库中，为数据库的潜在破坏打开了大门。

最后得分：12分中丢了8 1/2分，由于每人对于评分的宽大或严厉态度并不相同，应在再增加或扣除一两分。这个分数并不是一个精确的最后得分，但可以说明Oracle不能全部遵循Codd的十二条原则。但是，Sybase和Informix也不满足Codd的十二条原则。这并不意味着你不能将Oracle称为RDBMS，确实Oracle就是这样的。这仅仅意味着Oracle像它的同行们一样，还有许多工作要做，这没什么可奇怪的。Oracle正被稳步地改进，例如：在版本7中，Oracle仅能够完全地通过原则10——存储（及声明）完整性。

4.4 再论物理规划

在物理数据库设计上，采用关于数据字典、重做、撤销、表和索引存储的常规设计建议。再者，你还没有过多地深入到Oracle体系结构中——仅仅达到能做自己规划而已。

表空间（tablespace）是Oracle数据库中的物理结构，用于贮存逻辑对象或段，例如表和索引。表空间通常作为文件贮存在操作系统中，SYSTEM表空间贮存数据字典。在排序和创建期间，临时表空间（TEMP）贮存临时对象，如表和索引。ROLL1储存回滚段的回滚信息。一组重做日志直接被贮存为文件，而且这种数据文件没有表空间。你要提供一个理想的、通用的Oracle硬件规划，而且与应用无关。图4-1表示一个小型、可能的规划设计。

VOL代表一卷或一组磁盘，每个卷的意义描述如下：

- VOL0（操作系统和Oracle安装软件）可能是一个或两个磁盘，这取决于它是在UNIX系统环境还是NT系统环境。
- VOL1（数据字典）应该只需要一个磁盘。
- VOL2（临时空间）应该只需要一个大容量的盘，但有些应用可能需要额外的盘。
- VOL3（回滚数据）应该只需要一个盘。
- VOL4（数据）需要根据数据规模来指定用多少的磁盘——一组使用RAID3或5的盘。
- VOL5（索引）需要索引规模所指定大小的磁盘，一般要小于全部数据的规模（一组使用RAID0的盘）。
- VOL6（重做）可以是一组使用RAID1的盘，这组盘需要三个或三个以上的磁盘。至少控制文件（那些重要的操作系统文件，包含数据库的物理、结构信息）的三份备份应

该分布任何VOL上，特别是在头几个VOL上。

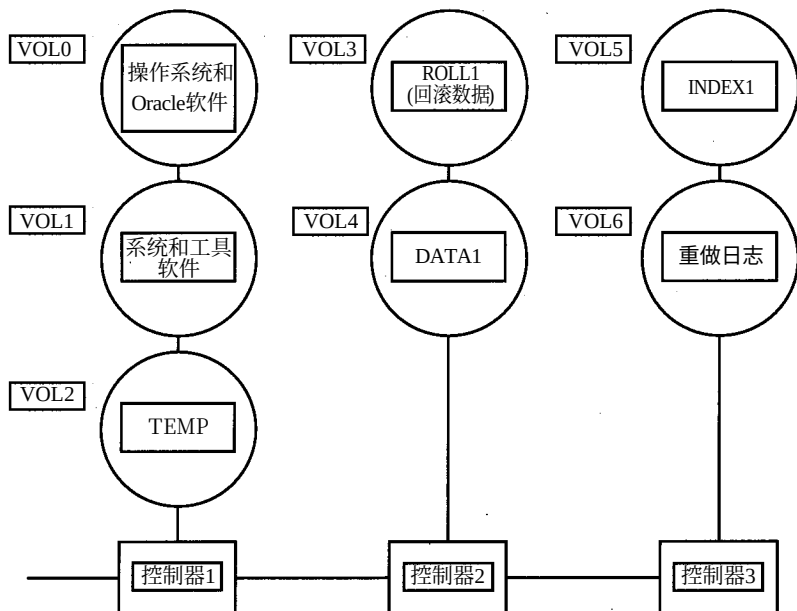


图4-1 一个通用的Oracle物理规划

注意 RAID表示廉价盘的冗余阵列（Redundant Array of Inexpensive Disks），这是通过I/O通道的并行化提高输入/输出性能的一种方法。这是通过一种称为数据条的技术实现的，数据条能以不同的方式实现。RAID和它的各个级别曾在第3章“数据库物理设计、硬件及相关问题”中详细地讨论过。

init.ora参数文件和口令文件通常贮存在 Oracle软件子目录下它们的默认位置。注意，TEMP、DATA1和INDEX1不但在不同的磁盘上，而且与独立的I/O通道或总线上的不同控制器相关。这是性能（和可用性）原因所要求的。

例如，如果DATA1和INDEX1存储在不同的磁盘上，但与相同的控制器相关，那么你消除一个瓶颈（硬盘竞争）却产生了另外一个瓶颈（总线竞争）。这要比在同一磁盘上存放DATA1和INDEX1效果要好一些，但却比图4-1的配置效果要差。在现实生活中，作为一个数据库系统管理员，你的目标就是拥有尽可能多的盘和控制器作为你的硬件配置，只有绝对必要时才妥协。

最后，一个数据库系统管理员可能需要增加辅助的回滚表空间（ROLL2），以适应应用需求的变化，以及更多的数据或索引表空间、更多的临时空间等需求。总的来说，为了正确地设计数据库，你要了解你的应用。在你较多地了解了系统结构与性能调整后，你将在第16章“性能调整基础”中再次看到这些。那时就能看到，能够为已接触过的不同应用类型（例如OLTP和DSS）提供可供选择的设计规划。

评估Oracle

为了帮助你评估Oracle，我已经总结了一张Oracle电子表单，它所基于的Oracle评估准则，你能在《Oracle服务器管理员指南（Oracle Server Administrator's Guides）》中找到。这里包

含的准则牵涉循环引用，在电子表单里不能直接实现。我修改了它的数学算法，并提出一些等同的公式，它们能够产生你所需要的相同结果：在合理的假设的条件下，算上表和索引，你的数据库总共有多大？只需输入你所需要的块输入变量，如下所示：

- block (DB_BLOCK_SIZE, 块) 以字节形式 (2048~65536)。
- pctfree (PCTFREE, 自由分区控制表) 为一个整数 (0~100)。
- initrans (INITRANS) 为一个整数 (0~200+)。

然后，对于表和索引部分，对于表单中每一个表或索引行，你只需输入输入变量即可，这些变量如下所示：

- #rows为一整数。
- row_row以字节为单位，每行的最大字节值。
- #cols_f为一整数，定长列（如NUMBER或CHAR）的数目。
- #cols_v为一整数，变长列（如VARCHAR2）的数目。

其他的事物都用于计算数目，如每一个表或索引的字节、块、KB、MB或GB，然后，计算出它们（数据库）的总数。然而，这并没把回滚段、重做日志、临时表空间和其他外围评估要素考虑在内。如果需要，可以改变 pctfree 变量，分别计算每一个表和索引。注意，不仅要包括主键索引，还包括外键索引和其他索引，尤其是那些唯一索引。你必须包括你希望实现的所有索引，这一点最为重要。这个表单有希望成为对你有用的工具，就像它对我非常有用一样！相信这种半自动方法确实要胜过手工估算。这个文件是一个 Microsoft Excel 7.0 电子表单，文件名为 orasiz.xls。

4.5 Oracle的未来

Oracle像其他的RDBMS 供应商一样，已经把面向对象特性如对象类（类型）、方法、集合（VARRAY和嵌套表）、安全性（视图）和识别符（REF）集成到它的RDBMS中。这在Oracle8中比以往更加明显。在Oracle 7中，初次增加了Universal Server。在Oracle 8中，你可以预料Universal Server的用途将变得更为广泛。另外，使用Oracle 8，你能在服务器上使用用户定义的数据类型（也就是对象类型）。利用网络计算体系结构，你可以得到数据盒开发、带有应用程序接口（API）的可插入数据类型的有关概念。还有，可以预料将来Oracle Web开发会继续发展，尤其是伴随着Oracle应用服务器、Oracle 8i、用于Web的Designer/2000、适用于Web的Developer/2000以及JDeveloper可用的改进版本的推出。随着Oracle 8i的发行，文件系统、DBMS以及Web服务器之间的差别就变得模糊了。事实上，正像某些商务杂志所指出的那样，随着Oracle 8i的出现，Oracle 应用服务器是否能够保持相应的市场都让人怀疑。有了Oracle 8和8i新性能的相互结合，你有理由预想Oracle至少可以继续占据市场上的主导地位。

Oracle发展的里程碑

- 1977 — Oracle公司成立
- 1979 — Commercial（商业）RDBMS
- 1983 — Portable（可移植）RDBMS
- 1987 — Client/Server（客户/服务器）RDBMS
- 1987 — CASE（计算机辅助系统工程）工具软件
- 1988 — Oracle 6发行

1992 — UNIX上运行的Oracle 7发行

1993 — Cooperative Development Environment (协作开发环境, CDE)

1994 — 在PC机上运行的Oracle 7 发行

1995 — Web连通性

1996 — 网络计算体系结构 (NCA)

1996 — Oracle Universal Server发行

1997 — 为Web提供客户/服务器工具软件

1997 — Oracle 8发行

1999 — Oracle 8i发行

与Oracle公司联系方法

主页：<http://www.oracle.com>

通信地址：

Oracle Systems Corp.

500 Oracle Parkway

Redwood Shores, CA 94065

USA

电话号码：

总部：(650) 506-7000

美国用户联络处：(650) 506-1500

美国销售处：(800) ORACLE-1

第二部分 Oracle 7.x RDBMS

第5章 Oracle数据库体系结构

本章要点：

定义数据库

SYS和SYSTEM的模式

数据库组件

数据库段

Oracle数据字典

其他数据库对象

5.1 定义数据库

“数据库”这个术语既可以作为整个数据库管理环境的名字，也可以（在 Oracle中）用来描述建立关系型数据库管理系统（RDBMS）的逻辑的和物理的数据结构。作为一名 Oracle专家，你可以把Oracle数据库定义为共同组成数据处理环境的配置文件、数据文件、控制文件以及重做日志文件；表、索引和其他包含在这些对象中的结构。

5.2 SYS和SYSTEM的模式

SYS和SYSTEM是每个Oracle数据库都缺省安装的两个帐户。SYS模式是所有内部数据库表、结构、供给包、过程等等的拥有者，它还拥有所有的 VS和数据字典视图，并创建所有封装的数据库角色（DBA、CONNECT、RESOURCE等等）。SYS是一个Oracle数据库的根用户或系统管理员；由于它具有全能的性质，你应尽量避免作为 SYS注册到系统中工作。当你以SYS帐户注册到系统中工作，哪怕一个简单的打字错误，就有可能造成毁灭性的灾难。

SYS帐户是唯一能够访问特定内部数据字典表的用户，因为它拥有所有的数据字典结构，为了将数据字典对象明确地授权给其他模式，SYS 也是你必须登录的帐户。当你使用数据字典视图和表编写存储过程或触发器时，也必需使用 SYS帐户。当数据库首次安装时，SYS帐户的缺省口令是CHANGE_ON_INSTALL，并且每个称职的数据库系统管理员（他或她）都会立即更改这个口令。

SYSTEM模式也是在数据库创建时安装的，是用于 DBA 任务的缺省帐户。SYSTEM也对所有的数据库对象拥有完全的权限，而且许多第三方工具软件依赖于 SYSTEM模式的存在及特权。SYSTEM 帐户的缺省口令是MANAGER，并且像SYS帐户的口令一样，在数据库创建后应该立即被更改。许多数据库系统管理员使用 SYSTEM模式执行数据库管理任务，但它更适于创建一个特殊的用户来完成数据库管理员的任务，这就确保了一个特殊的帐户与一个特定的人相连，而那个特定的用户则对全部数据库的修改负有责任。

因为这些模式都被大家所熟知，并且存在于每个 Oracle 数据库中，在安装完数据库后立即更改它们的缺省口令是很重要的，这样可以防止对帐户未经授权的访问。如果安全性是个主要的问题，你或许还应该考虑将这些帐户变为不可登录，只在需要登录它们时，才设置合法的口令。

注意 通过发出 ALTER USER × × × IDENTIFIED BY VALUES '口令' 命令，可以禁止对一个帐户的注册；这里 '口令' 是任何小写字字符串。这就将贮存的口令设置为你给出的实际的值，而不是加密的方式，在加密方式中，Oracle 用一个普通的 ALTER USER × × × IDENTIFIED BY '口令' 命令来存储口令。对 Oracle 来说，产生一个小写加密口令字符串是不可能的，从而使得对该帐户的登录变为不可能。

5.3 数据库组件

你可以把数据库对象划分成两种不同的类型：一类是由 RDBMS 内部使用的对象，被称为系统数据库对象（system database object），另一类是可以通过任何程序访问的对象，被称为用户数据库对象（user database object）。

5.3.1 系统数据库对象

当提到系统数据库对象时，是指 RDBMS 用于支持内部数据库功能的数据库对象。这些对象是由数据库系统管理员或服务器本身配置和创建的，并且不显式地用于用户数据库事务。

系统数据库对象如下所示：

- 初始化参数文件。
- 控制文件。
- 联机 and 归档重做日志文件。
- 追踪文件。
- ROWID（行内部地址）。
- Oracle 块。

下面是对这些对象的逐一解释：

1. 初始化参数文件

初始化参数文件（initialization parameter file）或 init.ora 是 RDBMS 主要的配置点，它是配置键码和数值的集合，每一个配置键码和数值都控制或修改数据库和实例操作的某个方面。它是一个 ASCII 文本文件，可以在 UNIX 服务器上的 \$ORACLE_HOME/dbs 目录下和 NT 服务器上的 \$ORACLE_HOME/database 下找到它。默认条件下，文件的名字为 initSID.ora，在这里，SID 相当于它所控制的数据库的标识符。在一个 UNIX 服务器上，如果在命令行上没有显式地指定一个 init.ora 文件，当启动一个数据库时，这就是 Oracle 服务器所要寻找的文件名（这里 SID 等于 \$ORACLE_SID 环境变量的值）。每个 Oracle 数据库和实例都有它自己唯一的 init.ora 文件。

init.ora 文件可以包含来自其他使用 IFILE 参数的文件的配置值。在 UNIX 环境中，通常将 \$ORACLE_HOME/dbs/init.ora 文件与其他位置的文件相连，以便使数据库环境安装有更好的控制和结构。非正式的参数（主要是由 Oracle Worldwide Customer Support 使用）使用前缀下划线来命名。

当数据库启动时，在创建实例或读取控制文件之前，先读取 init.ora 文件。init.ora 文件中的值决定着数据库和实例的特性，例如共享池、高速缓存、重做日志缓存分配、后台进程的自动启动、控制文件的读取、自动联机回滚段等等。直到数据库被关闭并重新启动，对 init.ora 文件中参数的更改才被承认。

在 Oracle RDBMS 中，缺省的 init.ora 文件位于 \$ORACLE_HOME/dbs 目录下，带有针对小、中、大型数据库所预先设置的基本 init.ora 参数和不同的推荐（任意的）值。当你创建新数据库和实例时，这个文件可被复制和重新命名。

通过查询 V\$PARAMETER 视图，可以从数据库内部观察 init.ora 文件中的配置参数集。V\$PARAMETER 视图把所有的 init.ora 参数和它们的值都列出来，并且每一个值都有一个标记符，用以指明参数值是否为服务器默认值。

在表 5-1 中，解释说明了包含在缺省 init.ora 文件中的参数。要得到一个更为全面的清单，参见服务器文档集中的 Oracle Server Reference manual（Oracle 服务器参考手册）。

表 5-1 常用的 init.ora 参数

参数名称	用 途
audit_trail	允许或禁止写记录到审计追踪文件。注意，这里只是允许审计，审计的动作必须分别配置
background_dump_dest	Oracle 后台进程追踪文件的最终目录，包括 alert.log
compatible	数据库的兼容级。能够防止使用比所使用数据库版本参数值高的数据库特性
control_files	数据库的控制文件
db_block_buffers	包含在高速缓存中的数据库块数目。db_block_buffers × db_block_size = 数据库高速缓存的大小，单位为字节
db_block_size	Oracle 数据库块的大小。在数据库建立起来后，这个值就不能改变了
db_files	能够打开的数据库文件的最大数目
db_name	可选择数据库的名字。如果使用该参数，它必须与用在 CREATE DATABASE 语句中的数据库名称相一致
db_file_multiblock_read_count	一次 I/O 操作所能读取数据库块的最大数。这是用于顺序搜索的，当调整全表搜索时，是非常重要的
dml_locks	由所有数据库用户用于全部表的 DML 锁的最大数目
log_archive_dest	归档重做日志文件的最终位置
log_archive_start	允许或禁止自动归档。如果允许，当实例启动时，ARCH 进程自动地启动
log_buffer	分配给重做日志缓冲区的字节数
log_checkpoint_interval	触发一个检测点需要填充的重做日志文件块数
max_dump_file_size	Oracle 追踪文件操作系统块的最大尺寸
processes	能与数据库连接的操作系统进程的最大数目，其中包括后台进程。当在 UNIX 服务器上调整共享内存时，这是很重要的
remote_login_passwordfile	确定一个口令文件是否用于远程内部验证，并指定有多少数据库可以使用一个口令文件。它可以被设置为 NONE、SHARED 和 EXCLUSIVE
rollback_segments	在数据库启动时，自动联机回滚段的列表清单
sequence_cache_entries	能够缓存在系统全局区中的序列数。它应当被设置成实例中随时能用的最大序列数
shared_pool_size	共享池的大小，用字节表示
snapshot_refresh_processes	在实例启动时，启动的 SNP 进程数目。SNP 进程负责刷新快照，以及执行由 DBMS_JOB 提交的数据库工作
timed_statistics	允许或禁止收集数据库的实时统计信息。虽然把该参数设置成真会导致性能开销，但是在数据库调整时有更大的灵活性
user_dump_dest	用户追踪文件的最终目录，包括通过设置 sql_trace 为真，所产生的那些文件

2. 控制文件

控制文件是数据库的心脏，它包含以下信息：属于数据库的数据文件和重做日志文件信息、数据库中的数据应该以何种字符集存储的信息、数据库中每个数据文件的状态和版本信息、以及其他的重要信息。包含在控制文件中的大部分参数是在数据库创建过程中设定的，并且相对来说是静态的；也就是说，它们不是经常改变的。控制文件采用二进制格式，并且是不可读或手工编辑的。

控制文件是在数据库创建时创建的。大多数数据库可以操作多个控制文件。特定控制文件的创建是在init.ora参数CONTROL_FILES中指定的，在CREATE DATABASE子句中指定的数据库创建参数存贮在这些文件中。

如果没有正确的控制文件，数据库不能被打开。如果由于某种原因，导致控制文件无效或毁坏，数据库将无法启动，并且存储在数据库中的数据信息将无法访问。正是由于这个原因，控制文件的镜像备份功能是被Oracle服务器内部支持的，并且也是大力推荐的。如果要控制文件镜像备份到一个新的数据库中，只需要在发出CREATE DATABASE命令之前，为CONTROL_FILES指定一个以上的参数值即可。要将控制文件镜像备份到一个现有数据库中，你必须关闭数据库，将当前的控制文件拷贝到你想要备份的目录当中，编辑CONTROL_FILES参数，以指定新的控制文件的位置，然后启动数据库。

注意 根据经验得来的一个有用的法则是：在四个不同的物理磁盘上，存储至少四份控制文件的备份。

不幸的是，修改控制文件参数不像改变初始化参数并启动数据库那么容易。为了改变任何控制文件参数，你必须重新创建控制文件。重新创建控制文件的步骤如下：

1) 备份数据库。修改控制文件过程中，出现的任何一个错误都会毁坏你的数据库，使之无法恢复。没有一个有效的数据库备份，千万不要进行这项操作。

2) 发出ALTER DATABASE BACKUP CONTROLFILE TO TRACE；命令，它是来自服务器管理器或SQL*Plus的命令。这就建立了一个用户追踪文件（位于USER_DUMP_DEST），该文件带有重建当前控制文件所必须的命令。

3) 编辑在前面步骤中所产生的追踪文件。除了CREATE CONTROLFILE语句外，删除追踪文件中的所有行，设置新的参数值。

4) 正常关闭（SHUTDOWN NORMAL）数据库。将旧的控制文件移放到备份目录里，确保Oracle在数据库启动时，不能在目录中找到控制文件的任何副本，否则下一步操作将会失败。

5) 执行STARTUP NOMOUNT命令启动数据库，运行你所编辑的CREATE CONTROLFILE追踪文件，这将重建带有新参数值的控制文件。

6) 执行ALTER DATABASE OPEN；命令。

在数据库创建时，把数据库参数值设为比你所需要的值高一些，可以避免重建你的控制文件。将数据库参数值设为比需要的高一些，带来的唯一负面影响是在内存和磁盘空间方面微不足道的浪费。

警告 当你创建数据库时，正确地设定CHARACTERSET参数是很重要的，改变这个参数需要重新创建整个数据库，不能通过重建控制文件改变它。

可配置的系统控制文件参数见表 5-2

表5-2 包含在控制文件中的可修改的配置参数

参数名称	说 明
MAXLOGFILES	联机重做日志文件的最大数
MAXLOGMEMBERS	每个重做日志文件的最大成员数
MAXDATAFILES	数据文件的最大数
MAXINSTANCES	能够安装到这个数据库上的最大实例数（并行服务器）
MAXLOGHISTORY	用于恢复实例的归档重做日志文件组的最大数（并行服务器）

注意 要改变数据库名，要像前面所描述的那样重新创建控制文件。但是，要在追踪文件中把 REUSE DATABASE <老名字> 行变为 SET DATABASE <新名字>。

V\$CONTROLFILE 视图列出了 Oracle 服务器当前正在读写的控制文件。

提示 你应该经常通过执行语句 ALTER DATABASE BACKUP CONTROLFILE TO TRACE，保持控制文件的一个文本备份（在保持通常的二进制副本的同时），当所有其他的手段都失败的时候，它可以帮助你重新创建数据库。

3. 联机重做日志文件

日志写后台进程（LGWR）将重做日志缓冲区的内容写入联机重做日志文件中，重做日志贮存了数据库的所有变化信息，并在数据库的恢复期间被 Oracle 使用。正如图 5-1 所示，联机重做日志文件至少由两组重做日志文件组成，并且按照一种循环的特性写入。

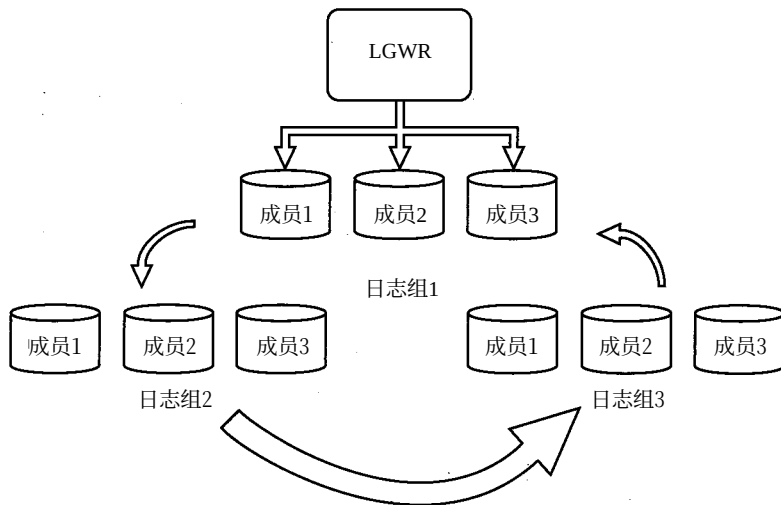


图5-1 带有多个成员的重做日志组

一个重做日志组如果当前正被 LGWR 写入，那么它是当前的。当当前日志组被填满，并且 LGWR 进程停止对它的写入而转到下一个日志组的时候，就发生日志切换。当日志切换发生并且归档日志模式被数据库允许的时候，前面写入的重做日志组被归档 (ARCH) 进程锁定，并被拷贝到磁盘或磁带上，具体拷贝方向取决于系统配置。如果 LGWR 追上 ARCH 进程，并且需要实时写入由 ARCH 正在写入的组，所有的数据库活动将被挂起，直到 ARCH 进程写完日志。如果你在 alert.log 文件中看到错误说明一个自由的日志组不能被 LGWR 进程锁定，那么说

明你需要添加更多的重做日志组或者调整日志组的大小。

每个日志组可由多个成员组成，日志组的每个成员是其他成员的精确镜像，并且重做日志项被并行地写入每个成员中。如果 LGWR 不能写入组里的成员，它也不失败；相反，它在 alert.log 文件中写入一个数据项。通过在每个组中使用多个成员，你能防止数据库由于丢失重做日志而导致失效。只要组里有一个成员是可用的，数据库就能继续工作。

注意 通过使用 RAID1（镜像）卷镜像备份重做日志组，可以获得同样的功能。当 LGWR 进程必须为每个数据库事务更新多个日志组成员时，这将缓解上述情况造成的系统开销。

V\$LOG 和 V\$LOGFILE 视图保存联机重做日志文件的信息，下列的查询检查当前日志的状态：

```
SELECT b.member, a.bytes, a.members, a.status
FROM v$log a, v$logfile b
WHERE a.group# = b.group#
ORDER BY b.member;
```

4. 追踪文件

所有的 Oracle 数据库都至少有一个文件用于记录系统信息、错误及主要事件。这个文件，叫做 sidALRT.log（这里 sid 是数据库系统的标识符），存储在由 init.ora 参数 BACKGROUND_DUMP_DEST 指定的位置。当调查数据库故障时，这是你应该查看的第一个位置。关键的错误总是在这里被记载，如数据库的启动和关闭信息、日志切换信息以及其他的事件。

后台进程和用户进程也建立它们自己的追踪文件，在其中记载出现问题和故障的信息。后台进程追踪文件贮存在 BACKGROUND_DUMP_DEST 所指定的位置，而用户进程追踪文件贮存在由 USER_DUMP_DEST 参数设置所指定的目录里。将 USER_DUMP_DEST 目录设定为不同于由 BACKGROUND_DUMP_DEST 所指定的目录，使得你能够跟踪不同类型的追踪文件。后台进程追踪文件被命名为 sidPROC.trc，这里 sid 是数据库的系统标识符，PROC 是后台进程（DBWR、LGWR、SMON、PMON 等等）的名字。

用户会话追踪文件带有一个 ora 前缀，后面是一系列带有 .trc 文件扩展名的唯一标识数字。当一个用户会话引起无法恢复的问题（例如死锁或一个服务器进程崩溃）或用户会话被显式地告知创建跟踪文件时（例如当 SQL 跟踪被允许，或者发出一个 ALTER DATABASE BACKUP CONTROLFILE TO TRACE 命令），用户会话追踪文件就被生成。如果要允许 SQL 追踪，在 SQL*Plus 提示符状态下发出 ALTER SESSION SET SQL_TRACE=TRUE 命令，或把 init.ora 参数 SQL_TRACE 设置为 TRUE。但是注意，在 init.ora 中将 SQL_TRACE 设为 TRUE，将会导致把所有与数据库不一致的 SQL 语句写进追踪文件中，这将产生大量的追踪信息。

可以在 V\$PARAMETER 视图中获得 BACKGROUND_DUMP_DEST 和 USER_DUMP_DEST 参数的当前设置。

5. ROWID

为了提取信息，Oracle 数据库必须能够唯一识别数据库中的每一行。Oracle RDBMS 中用于这种任务的内部结构叫做 ROWID（行内部地址），这是存储数据库中每条记录行的物理位置的两字节数值。ROWID 的格式如下：

```
BBBBBBBB.RRRR.FFFF
```

BBBBBBBB是数据文件中行所在的块号（采用16进制）。RRRR是数据行所在块中的行号（采用16进制），FFFF是块所在的文件号（采用16进制）。

例如：一个表中的一行可能有如下所示的 ROWID 形式：

0000068C.0000.0001

这个 ROWID 是在第一个数据文件（0001）的 68C（16进制）块中，并且是该块的第一行（0000）。

注意 通过查询 DBA_DATA_FILES 视图，你可以把前面 ROWID 中的文件号和文件名相匹配。在 Oracle7 中，这些文件号是固定的；而在 Oracle8 中，它们是在数据库启动时决定的。

当每行在第一次创建时，都会被分配一个 ROWID。除非该行被删除或该行所在的段被重组（通过导入/导出、第三方厂商重组工具等等），否则这个 ROWID 保持不变。在数据库中使用 ROWID 是找到一行的最快方法。数据库的每个表都有一个名为 ROWID 的伪列（pseudocolumn），通过查询它可以列出表中每一行的 ROWID。

因为 ROWID 的唯一性，它可以被用来创造性地解决许多不同的难题。例如：ROWID 的一个最常用用途是在数据库中识别那些具有相同值的列。下面的 SQL 脚本将显示这是如何实现的：

```
delete from duplicate_table
where ROWID not in (select MIN (ROWID) from duplicate_table
                    group by a1, a2);
```

在查询时，你也可以使用 ROWID 的实际值，下面的 SQL 脚本显示一个在其中含有行的表的数据库文件数目：

```
SELECT COUNT(DISTINCT(SUBSTR(ROWID, 15, 14))) "Files"
FROM test_table;
```

6. Oracle 块

你能操作的最低级的数据库存储就是 Oracle 块，这是服务器能够访问的最小存储单元。不应该把它与操作系统块混淆，一个 Oracle 块是由操作系统块组成的，它们并不相同。

提示 Oracle 块是操作系统块的整数倍。例如：在 UNIX 中，操作系统块通常为 8KB，那么你应该将 db_block_size 设置为 8192、16384 等等，使它成为 8KB 的整数倍。

所有的数据访问都是以 Oracle 块的形式实现的。Oracle 块的大小是指一次 I/O 过程中，RDBMS 从数据文件中读写的字节数、数据库对象大小和高速缓存中的块也是以 Oracle 块的形式设置的（尽管有些视图用字节显示存储的大小，但这只是为了增加可读性）。

注意 Oracle 块的大小是在数据库创建时为数据库设置的，是不能改变的。如果在创建数据库后，你认为需要大一些的（或小些的）块，那么你不得从头开始，重新建立整个数据库。

每个 Oracle 块都包含头信息空间、块中数据将来的更新空间以及块中实际存储行所占空间。块头保存着这样的信息，例如在块中有行存在的数据库段、同时可以有多少事务访问块等等。每个块还分配了一定量的空间，用于块中贮存行的将来更新，如果升级引起原先的行变大，那么这个自由空间就被使用了。

一个块可以用于接受新行的能力是通过 PCTFREE和PCTUSED存储设置来控制的，PCTFREE参数负责分配块空间的百分比，这些空间被保留在一旁，用于数据更新。例如，如果一个块有一个30%的PCTFREE值，那么该块70%的空间是用于存储新行的，当该块70%的块已经被填充时，Oracle把该块从自由列表中取出，利用剩下30%的空间来处理要求更多的空间的块中行的更新。

在块被重新放回自由列表之前，PCTUSED参数指定块中必须有多少自由空间（通过删除块中的行，或更新减少贮存一行所需空间）。

PCTFREE参数和PCTUSED参数一起使用，确保一个块有足够的空间来满足将来的存贮需要，而且确保了块不会反复摆动，换句话说，不会花费相当多的时间从自由列表进进出出。PCTFREE和PCTUSED的值不应该等于100%，否则参数值就不起作用了，参见第21章“管理数据库存储”，可以得到关于调整Oracle块的更多信息。

因为数据库块决定着在每一次I/O操作中能从数据文件中读取的字节数，所以它的大小是非常重要的调整因素。联机事务处理（Online Transaction Processing, OLTP）应用有时可以从较小的块（4KB或8KB）上获益，因为这些应用在每个事务期间，通常要读写少量的数据，因此较小的块在I/O性能方面更为有效。

对于仓库式的或决策支持系统（Decision Support System, DSS）数据库来说，较大的数据库块能够极大地提高性能。这些类型的应用在一个事务过程中，常常需要处理大量的数据，并且通常考虑系统的响应时间以分钟来计时，而不以秒来计时。

通过设置数据库块的大小，你可以调整一次I/O操作所读入内存缓冲中的数据量。当决定数据库块的最佳大小时，你必须记住你的应用将要执行的特定类型的数据存储，以及你希望贮存在每一行中的数据量。一个过大的数据库块将会把一些不必要的数据读进缓冲区，而一个太小的块则会导致行链接。

5.3.2 用户数据库对象

用户数据库对象（User database objects）是指那些不是Oracle RDBMS专用的对象。当然，它们是受Oracle内部管理的，但是它们能够提供一组用户建造块，利用这些块用户可以创建自己的数据库。用户数据库对象包括数据文件、区间、表空间以及数据库段。本章下面部分对这些术语做了一一解释。

1. 数据文件

Oracle 数据文件作为操作系统文件而存在，每个数据文件被分配给一个表空间，并且拥有存储在那个表空间里的实际数据。数据文件是文件系统上的实际文件，它可以像任何其他操作系统文件一样进行监控和操作。存储在数据文件中的数据采用 Oracle二进制格式，这样除了Oracle RDBMS 外，不能被其他任何东西读取。

数据文件是使用SQL命令CREATE TABLESPACE或ALTER TABLESPACE创建的。一个数据文件的大小是根据 CREATE语句中指定的大小来建立的，而不是由它所贮存的数据量决定的，例如：一个以10MB大小创建的数据文件，不管它包含一行或一百万行，它都用满10MB的空间。

从Oracle 7.3开始，数据库管理员具备动态增加和缩减 Oracle数据文件的能力。然而，数据文件不能被收缩得小于数据文件的高水位线。数据文件也能被单个脱机，以进行备份或其

他的数据库操作。

你可以访问 DBA_DATA_FILES和V\$DATAFILE 视图，来获得数据文件中为数据库定义的有关信息。下列查询展示了表空间与数据文件之间的映射：

```
SELECT tablespace_name, file_name, bytes
FROM dba_data_files
ORDER BY tablespace_name, file_name;
```

2. 区间

区间（extent）是一个存储单位，由一个或多个逻辑（就是说，在 Oracle内）连续的 Oracle块组成，每个数据库段都是由一个或多个区间组成，在一个数据库段中的每个区间的大小可以相同或不同。从 Oracle 7.3开始，一个数据库对象所能拥有的区间最大数取决于 Oracle块的大小。

一个数据库段在对象创建时，完全按照 CREATE命令的存储子句中所指定的来分配区间。当一个段再也无法把新数据填充到它当前分配的区间中时，必须为它分配其他的区间。下一个分配区间的大小决定于好几个因素，有些因素并不是显而易见的。新区间的分配管理是数据库空间管理主要涉及的内容。参阅第 21章“管理数据库存储”，可以得到更多信息。

区间信息贮存在DBA_EXTENTS视图中。

3. 表空间

表空间（tablespace）是一个数据结构，用于组合被相似地访问的数据。每个表空间都是由一个或多个数据文件组成的，所有的数据库对象必须被指定一个表空间，它们在那里被创建。于是组成对象的数据就被贮存到分配给指定表空间的数据文件中。

表空间被用于分隔涉及数据访问的 I/O，例如：可以创建一个表空间来存储数据对象，也可以创建另外一个表空间来存储索引对象。通过给驻留在不同物理磁盘上的表空间分配数据文件，你可以确保对索引数据的访问不会影响对索引指向的数据的访问。参阅第 19章“调整输入输出”，可以得到更多信息。

在数据库备份和恢复中，表空间也扮演了一个重要的角色。因为一个表空间直接映射着一个或多个数据文件，备份和恢复数据通常是在表空间（数据文件）级下进行的。当然，备份或恢复操作是应用于整个数据库时是例外的。

你可以在DBA_TABLESPACES查看表空间的信息。

5.4 数据库段

数据库段（database segment）是贮存在数据库中的用户建立的对象。这在很大程度上包括了组成模式的数据（表）和索引。

除了用户创建的数据和索引段外，通常还有两种类型的段被称为系统段或管理员创建段，它们是临时段和回滚段。当然，那些拥有合法权限的用户可以创建这些段。但是，最好由数据库系统管理员来创建，然后让应用用户和程序共享这些段。

5.4.1 表

表（table）是存储数据的数据库段。每个表是由一个或多个列组成的，每个列都被指定一个名字和数据类型。每个列的数据类型为贮存在表中的数据定义了类型和精度，合法的 Oracle列数据类型列于表 5-3中。

表5-3 有效的Oracle 7.x 列数据类型

数据类型	说 明	最大长度
CHAR	固定长度的字符域，尾部使用空格填充	255 字节
VARCHAR	变长字符域	2KB
VARCHAR2	变长字符域	2KB
LONG	变长字符数据	2GB
NUMBER	变长数字数据	$1 \times 10^{+130} \sim 9.99 \times 10^{+125}$
DATE	固定长度的日期和时间域，从公元前到公元后 4712年12月31日	4712年1月1日
RAW	变长原始二进制数据	255字节
LONG RAW	变长原始二进制数据	2GB
ROWID	行内部地址变量类型	6字节

注意 为预防ANSI规定的改变，Oracle建议所有的变长字符域定义为VARCHAR2，而不是VARCHAR。Oracle保证永远不对VARCHAR2做出导致需要对应用程序修改才能使它们向上兼容的功能性变化。因为VARCHAR的功能是由ANSI标准委员会授权管理的，Oracle不能保证这些功能在新版本里不发生巨大的变化。

DBA_TABLES和DBA_TAB_COLUMNS包含数据库中关于表的信息。

5.4.2 索引

索引（indexe）是为了加速对特定表数据的访问而创建的数据段。一个索引拥有表的一列或多列的值以及与这些列值相对应的行内部地址（ROWID）。当Oracle服务器需要在表中查找某一指定行时，它在索引中查找ROWID，然后直接从表中提出数据。

在Oracle RDBMS中有几种可用的索引类型。到目前为止，最常用的索引类型是 B*-Tree（B-树）索引。这是执行标准的 CREATE INDEX 语句时，需要使用的索引类型。B*-Tree索引是标准的搜索树算法的一个变异，其中通过遍历索引树，你保证能在相同数量的树遍历过程中找到所有的叶子结点。每一个叶子结点指向下一个和前一个叶子结点的位置，这样就能够索引扫描范围内进行快速索引搜索及类似操作。B*-Tree索引能确保维持平衡状态，并且每个结点的四分之三都是空的，以便为更新提供可用空间。

对B*-Tree索引结构的深层解释不属于本书范围。但是，你可以在大部分现代数据库教科书中看到有关该主题的大量信息，例如 Elmasri和Navathe的Fundamentals of Database Systems。

簇（cluster）索引是在簇中被表共享的列的索引。在本章稍后会解释簇表。不同于常规的索引，簇索引在索引中只存储一次索引键值，而不管索引键在表中重复多少次。在簇上能够执行任何数据操作语言（DML）前，必须在簇上创建簇索引。

最新的索引类型是位映射（bitmap）索引，在一个位映射索引中，位映射是从索引表的列值中创建而来的，并存储在索引中，而不是由实际的列值创建的。换句话说，在键上索引为每一行保存一个位映射，键对于表中的每一行包含一位信息。如果值包含在行里，位值是 1，否则是 0。在基数较低（不同值数量较小）的列中，位映射索引可以比传统的 B*-Tree索引更小，更有效。表5-4是一个关于汽车表的位映射例子，位映射的键是表中的颜色列。

在这个例子中，表中共有 20 行。汽车表中行与颜色相匹配处，位映射将为 1，即：第 4、7、11、15 和 20 行是红色汽车，而第 2、6、和 12 行是黑色汽车。位映射索引结构创造了比传统

B*-Tree索引更小的索引，但正如讨论的那样，位映射索引只适用于特定类型数据。

DBA_INDEXES 和 DBA_IND_COLUMNS 含有关于数据库中全部索引的信息。

表5-4 汽车表位映射索引

汽车颜色	位 映 射
红色	00010010001000100001
绿色	10000001010001001010
银色	00100000100010010000
白色	00001000000000000100
黑色	01000100000100000000

5.4.3 回滚段

回滚段（rollback segment）是存储在数据库事务中发生改变的原始数据块的数据库对象。它们用于提供数据的已经改变但尚未提交的读一致性视图。当做出数据改变时，原始数据就被拷到回滚段中，而且在内存缓冲区中对数据块做出更改。如果其他用户会话要求同样的数据，那么存储在回滚段中的原始数据就会被返回（这就叫读一致性）。当做出更改的会话被提交时，回滚段项将被标明为无效。

多个用户会话可以共享一个回滚段，每个回滚段至少由两个区间组成。当一个事务开始时，用户会话在一个可用回滚段中得到一个可用区间的专用锁，于是事务信息被写入回滚段中，如果事务填满了第一个区间，它会被分配另一个区间。如果另一个区间不可用，回滚段会自动地给自己分配其他的区间，这正是用户会话所要获取的，这叫回滚段扩展（rollback segment extension）。因为区间的分配影响性能，因此，你的目标应该是在没有分配新区间的情况下，使所有事务能够执行。

如果回滚段不能分配其他的区间（或许是因为已经达到回滚段的最大区间数，或者在回滚段表空间上没有更多的自由区间），那么就会出现错误，而且事务将被回滚。这通常发生在装载大量数据时，在那时联机回滚段无法为事务提供足够的空间来存储全部的回滚信息。

有关创建和管理回滚段的详细信息，参见第 21 章。

DBA_ROLLBACK_SEGS 视图中含有关于回滚段的信息。

5.4.4 表簇

表簇（table cluster）是一个数据库对象，它可以将那些经常在相同数据块中一起使用的表进行物理分组。当你处理那些经常连接在一起进行查询的表时，表簇是特别有效的。一个表簇存储簇键（用于将表连接到一起的列），以及簇表中的列值。因为簇中的表都被贮存在相同的数据库块中，所以使用簇工作时，I/O 操作就减少了。

5.4.5 哈希簇

哈希簇（hash cluster）是数据库存储的最后选项。在一个哈希簇中，表是基于哈希值组织的，在表的主键值上使用哈希函数可以得到这个哈希值。在从哈希簇中提取数据时，哈希函数被用于要求的键值上，结果值给出 Oracle 哈希簇中的存储数据的块。

使用哈希簇能明显减少从表中提取数据行的 I/O 操作，但是使用哈希簇也有一些缺点。有

关创建和管理哈希簇的详细信息参见第 21 章。

5.5 Oracle 数据字典

数据字典 (data dictionary) 是存储在数据库中的所有对象信息的知识库, Oracle RDBMS 使用数据字典获取对象信息和安全信息, 而用户和数据库系统管理员用它来查阅数据库信息。它保存着数据库中数据库对象和段的信息, 例如表、视图、索引、包以及过程, 它还保存着关于如用户、权限、角色、审计和约束等的信息。数据字典是只读的, 你永远不要尝试对任何数据字典表中的任何信息进行手工更新或改动。它由四部分组成: 内部 RDBMS (X\$) 表、数据字典表、动态性能 (V\$) 视图和数据字典视图。

5.5.1 内部 RDBMS (X\$) 表

Oracle 数据库的心脏就是即所谓的内部 RDBMS (X\$) 表, 这些表被 Oracle RDBMS 用于跟踪内部数据库信息。X\$ 表是加密命名的、非文献性的表, 并且几乎是无法解密的。大多数 X\$ 表被设计成不能被数据库系统管理员或用户直接使用。尽管如此, 它们仍包含着有价值的信息。许多非文献性的或内部的统计和配置只能在 X\$ 表中找到。

要解密在某个 X\$ 表中贮存的信息, 最简单的方法是从一个已知的数据字典表倒推寻找。SQL*Plus 自动跟踪特性对这项工作是非常有用的。例如, 要想知道 V\$SGASTAT 中的信息到底存放在哪里, 你可以执行下面的分析步骤:

- 1) 作为 SYS 用户 (或者一个对 X\$ 和 V\$ 表有明确访问权限的帐户) 登录到 SQL*Plus。如果你所登录的模式中不存在 PLAN_TABLE, 你可以通过运行 \$ORACLE_HOME/rdbms/admin/UTLXPLAN.sql 语句来建立一个。

- 2) 执行下列 SQL*Plus 命令: SET AUTOTRACE ON。

- 3) 针对有你感兴趣内容的表执行一条查询, 把 WHERE 子句设为一个永远不会为真的值, 以便没有任何行能够返回: SELECT * FROM V\$sgastat WHERE 0=1;

在其他信息中, 自动追踪返回类似下列的输出:

```
Execution Plan
0  SELECT STATEMENT Optimizer=CHOOSE
1  0      FILTER
2  1      FIXED TABLE (FULL) OF 'X$KSMSS'
```

从 SQL 追踪的输出, 你可以解密数据字典基表, 从基表中抽取视图信息。查询建立在这种形式上的 X\$ 表, 常常可以产生令人吃惊的信息。

5.5.2 数据字典表

数据字典表 (data dictionary table) 存储表、索引、约束以及所有其他数据库结构的信息。它们属于 SYS, 通过运行 SQL.BSQ 脚本来创建 (在数据库创建时自动发生)。通过它们名字后面的美元 (\$) 符号 (tab\$, seg\$, cons\$ 等等), 可以很容易地将它们辨认出来。在数据字典视图中可以找到数据字典表中的大部分信息, 但是一些应用或查询也可以从使用包含在基表中的信息中获益。

数据字典的列和表在 SQL.BSQ 文件中被很好地归档, 这个文件在 \$ORACLE_HOME/dbs 目录中可以找到。当你熟悉了 SQL.BSQ 的内容后, 你就可以更好地理解 Oracle RDBMS 实际

上是怎样贮存数据字典和数据库信息的了。

5.5.3 动态性能视图

动态性能 (V\$) 视图 (dynamic performance (V\$) view) 是 Oracle 数据库系统管理员的主要依靠, 这些视图包含了大量数据库函数运行时的性能和统计信息。它们还具有相当的可读性 (与 X\$ 表相反), 也就是说能够被数据库系统管理员用于诊断和解决问题。关于大多数 V\$ 视图的文档能够在 Oracle Reference Manual 中查到。

注意, V\$ 视图实际上是由 SYS 拥有的 V\$ 视图的公共同义词, 当编写读取 V\$ 表的存储过程或函数时, 这是很值得注意的。引用或授权给基表 V\$ 视图 (而不是 V\$ 公共同义词) 常常是更必要的。

5.5.4 数据字典视图

数据字典视图是在 X\$ 和数据字典表上创建的视图, 也就意味着它们能被终端用户和数据库系统管理员使用和查询, 它们被分成三类: DBA_、ALL_ 和 USER_ 视图。DBA_ 视图包含了数据库所有对象的信息。例如, DBA_TABLES 包含所有已创建表的信息, ALL_ 视图包含了用户查询表时可以访问的所有对象的信息, USER_ 视图包含了用户查询表时表所拥有的全部对象的信息。

5.6 其他数据库对象

在数据库中还贮存着一些其他的对象, 这些对象没有被准确地按段分类, 尽管如此还是应该讨论一下它们。它们包括视图、序列、同义词、触发器、数据库链以及存储包、过程和函数。下面几节对它们进行说明。

5.6.1 视图

视图是存储的能够被查询的 SQL 语句。由于安全原因, 视图用于隐藏一些数据 (例如一个 HR 视图只显示姓、名和地址信息, 而没有显示社会保险号码和工资数据), 并使复杂的查询变得易于理解和使用。通过在远程数据库表上创建视图, 视图也能被用于隐藏分布式数据库对象。任何可以做为 SQL 查询而执行的语句, 都能被创建成为一个视图。

注意 在设计应用程序时, 视图是非常有用的, 因为它们能够以表的形式隐藏复杂的查询逻辑, 使得更易于查询。它们可以和嵌在它们内部的优化器提示一起被创建, 以确保最佳的查询性能。

THE DBA_VIEWS 视图存有在数据库中创建的视图的信息。

5.6.2 序列

序列是用于产生唯一数码的数据库对象。序列创建时带有初始值、增量值以及最大值。每次从序列中返回一个数字, 当前序列值是一个一个递增的, 每个序列产生的数字可达到 38 位长度。

可以通过选择来自序列的 NEXTVAL 或 CURRVAL 伪列使用序列。例如: 你有一个名为

EMP_SEQ的序列，执行“SELECT EMP_SEQ.NEXTVAL FROM DUAL；”语句，返回序列的下一个整数值，并且当前序列值加1；你可以使用“SELECT EMP_SEQ.CURRVAL FROM DUAL；”语句，返回当前序列的整数值。注意，要使用 CURRVAL，你必须通过执行一个序列在NEXTVAL 伪列上的查询，来预先初始化用户会话的序列。

序列最通常的用途是提供唯一的数字作为表主键列的代用品。有关序列的信息贮存在 DBA_SEQUENCES 视图中。

5.6.3 触发器

触发器 (trigger) 是存储过程，当针对一个表发生特定的动作时，就会激活它。触发器可以被编码。当针对一个表进行插入、更新、删除或三种操作的结合时，激活触发器，也可以在某行被影响或某条语句出现时被激活。触发器经常用于加强数据完整性约束和业务规则，这些业务规则对于内建的 Oracle 引用完整性约束来说实在是太复杂了。关于数据库触发器的信息可以在 DBA_TRIGGERS 视图找到。

5.6.4 同义词

同义词 (synonym) 是指向其他数据库表的数据库指针。当你创建一个同义词时，你就指定了一个同义词名字和同义词所引用的对象。当你引用同义词名字时，Oracle 服务器会自动地用同义词定义的对象名字来代替同义词的名字。

同义词有两种类型：私有 (private) 和公共 (public)。私有同义词是在指定的模式中创建的，并且只允许拥有它的模式访问。公共同义词由 PUBLIC 模式所拥有，所有的数据库模式都可以引用它们。

理解在 SQL 语句中对象名字的解析顺序是十分重要的。如果执行 SQL 语句“SELECT * FROM EMP_SALARY；”，Oracle 服务器将按照如下方式决定 EMP_SALARY 对象：

- 1) 首先，服务器查看在发出命令的用户模式中是否有名为 EMP_SALARY 的表或视图。
- 2) 如果这个表或视图不存在，Oracle 检查名为 EMP_SALARY 的私有同义词是否存在。
- 3) 如果这个私有同义词存在，这个同义词所引用的对象将取代 EMP_SALARY。
- 4) 如果这个私有同义词不存在，检查名为 EMP_SALARY 的公共同义词是否存在。
- 5) 如果这个公共同义词不存在，Oracle 返回消息“ORA-00942, table or view does not exist”。

公共同义词应小心使用，因为所有的模式都可以用公共同义词决定对象名字，以致产生无法预料的结果。

关于公共同义词的信息贮存在 DBA_SYNONYMS 中。注意，在这个视图中，公共同义词的拥有者作为 PUBLIC 被列出来。

5.6.5 数据库链

数据库链 (database link) 是与远程数据库连接的存储定义，它们用于查询分布式数据库环境中的远程表。因为它们存贮在 Oracle 数据库中，它们被归入数据库对象类别。关于数据库链的详细信息可以在第 40 章“分布式数据库管理”和 DBA_DB_LINKS 数据字典视图中得到。

注意 如果使用特定的 UserID 和口令与远程数据库连接，DBA_DB_LINKS 是一种能够

以明码存储口令的视图。当允许终端用户访问这个数据库视图时，一定要小心。

5.6.6 包、过程和函数

存储包、过程和函数与它们的源代码一起存在数据字典中。一个存储过程是一个操作的代码单元，可以被传递参数，并能够返回值。一个存储函数是一个代码单元，可以被传递参数，并能够返回一个值。一个包是一个过程、变量和函数的集合，包按照功能被逻辑地分组。参见第26章“管理员使用的SQL * Plus”可得到更多的信息。

你可以通过DBA_OBJECTS和DBA_SOURCE视图查看有关存储包、过程和函数的有关信息。

第6章 Oracle 实例结构

本章要点：

- 介绍
- 定义实例
- 创建实例
- Oracle实例的组成
- 事务剖析
- 监控实例

6.1 介绍

当有人谈到Oracle数据库时，很可能指的是整个 Oracle数据库管理系统（DBMS）；但是作为一个专业的Oracle数据库人员，你必须认识到数据库和实例二者之间的重大区别——一个经常使非Oracle的系统管理人员混淆的区别。本章将探讨 Oracle实例的结构和构造，在下一章将继续从更深层次探讨 Oracle关系型数据库管理系统（RDBMS）。为了避免混淆，RDBMS这个术语用来描述由Oracle数据库和实例组成的整个数据管理服务器。本章还将详细地讨论实例何时和如何产生。

6.2 定义实例

为提供Oracle客户所期望的不同程度的服务、灵活性与性能，数据库的许多工作由实例完成，实例是一系列复杂的内存结构和操作系统进程。除非使用并行 Oracle服务器选项，否则每个Oracle数据库都有一个实例与之相关，一个数据库被唯一的一个实例装载。实例结构允许RDBMS同时对来自多个用户的不同种类事务的请求提供服务，与此同时提供一流的性能、容错性、数据的一致性和安全性。

注意 本章定义的术语“进程”是指在没有用户干预的情况下正在运行的任务。你的操作系统可能将之称作“进程”，或者使用其他术语，例如任务、作业、线程和其他类似的术语。

在UNIX实现多任务操作系统后，实例是松散的结构方式。在一起工作的离散的进程在实现实例的目标的RDBMS中完成指定的任务。每一个进程都有各自的一个内存块，该内存块用于保存私有变量、地址堆栈和其他运行时的信息。进程间使用公共共享区并在公共共享区内完成它们的工作。公共共享区是能够在同一时间内被不同程序和不同进程读写的一块内存区。该内存块称为系统全局区（SGA）。

注意 因为SGA驻留在一个共享内存段中，所以它经常被称作共享全局区。

你可以认为后台进程就像数据库的手，直接处理数据库的组件；你也可以认为 SGA就像大脑，在必要时间间接地调度手处理它们的信息与存储检索。SGA参与发生在数据库中全部的

信息和服务器的处理。

注意 单用户的Oracle配置（例如 Personal Oracle Lite），不使用多进程执行数据库的功能。相反，所有的数据库功能由一个Oracle进程完成。由于这个原因，单用户也称为单进程Oracle。

6.3 创建实例

打开一个Oracle数据库包括以下三步：

- 1) 创建一个Oracle实例（非安装阶段）。
- 2) 由实例安装数据库（安装阶段）。
- 3) 打开数据库（打开阶段）。

Oracle实例在数据库启动的非安装阶段创建。当数据库经过非安装阶段时，读取 init.ora 参数文件，启动后台进程，初始化系统全局区（SGA）。init.ora 文件定义了实例的配置，包括内存结构的大小和启动后台进程的数量和类型等。实例名根据环境变量 Oracle_SID 设置，它不一定要与打开的数据库名称相同（但是习惯上通常如此）。下一阶段称为安装阶段。init.ora 文件中的控制文件参数值决定数据库的安装实例。在安装阶段，读取控制文件并使其成为可访问的，可以对控制文件内存储的数据进行查询和修改。最后的阶段就是打开数据库。在这一阶段，其名字存储在控制文件中的数据库文件以排它使用方式被实例锁定，使数据库能够被普通用户访问。打开是数据库的正常操作状态。在数据库打开之前，只有 DBA 能访问数据库，且只能通过服务器管理器对其进行访问。

为了改变数据库的操作状态，必须作为内部连接到数据库，或拥有 SYSDBA 特权。当数据库从关闭状态到打开状态时，你可以明确地单步调试数据库的每一个操作状态，但当关闭数据库时，只能从当前运行状态转到完全关闭状态。例如，可以在服务器管理器工具中执行 STARTUP NOMOUNT 命令，这将使数据库处在非安装阶段，接下来可以运行 ALTER DATABASE MOUNT 或者运行 ALTER DATABASE OPEN 命令以单步调试到操作阶段。无论是在何种操作状态下，如果执行 SHUTDOWN 命令，将完全关闭数据库。例如，数据库不能从打开状态转到安装状态。

没有安装数据库的实例被称为空闲的——它使用内存，但不做任何工作。一个实例只能唯一地与一个数据库连接，而且除非使用并行服务器，否则对一个数据库也只分配一个实例。实例是数据管理的核心——它做所有的工作，而数据库存储所有的数据。

6.4 Oracle实例的组成

图6-1是Oracle实例的可视化表示。以下是对其各不同部分的解释。

许多参数和现成的技术能够帮你配置实例，以最好地支持你的应用和要求。配置实例对象以获得最高性能在多数情况下是一个试验和错误过程——你可以使用可能的参数值启动，但是只有经过一段时间的监控才会得出所有设置和变量的最好的可能组合。

配置实例参数包括改变必要的 init.ora 参数和在数据库状态之间切换（打开、关闭数据库）。在 init.ora 文件中有大量的参数，这些参数有些是无文档说明的。虽然你不应该改变或者增加你不熟悉的参数，但是可以查询内部表 x\$kspopi 以查看数据库所有可能的初始化参数。ksppinm 和 ksppdesc 列分别给出参数名和对参数的简要描述。

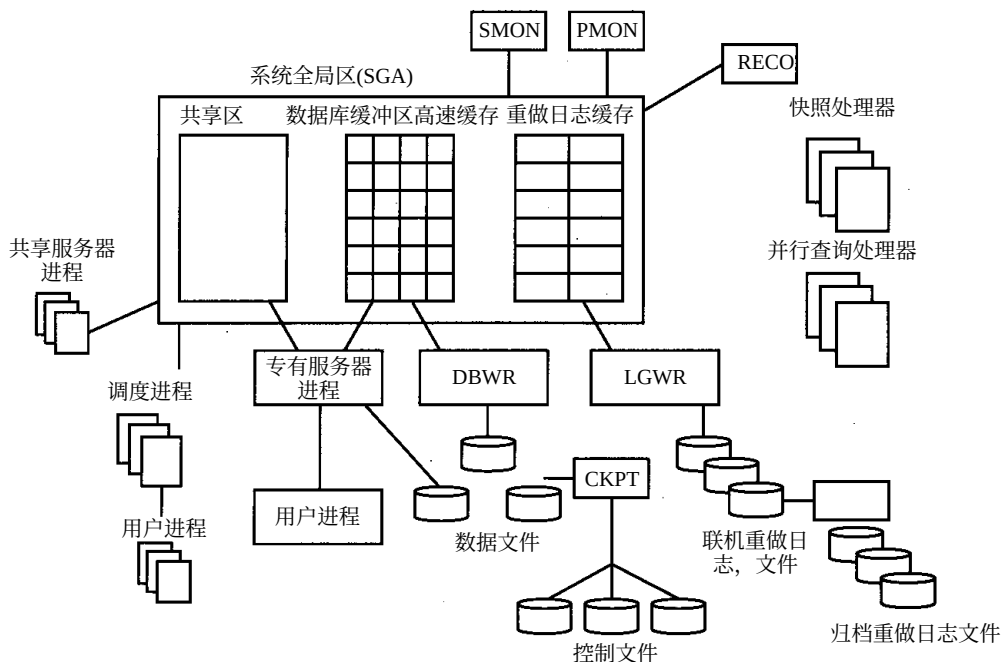


图6-1 Oracle实例是一个复杂的内存与后台进程的交互体

注意 改变初始化参数而没有清楚地理解这种改变可能的后果，是非常危险的。有许多参数纯粹是出于单纯的诊断原因而存在的，改变它能导致你的数据库处于非同步状态或者崩溃状态。无文档说明的参数以下划线开头命名。除非你有足够的把握，否则不要轻易增加或改变init.ora文件的关键字或值。

对于大多数部分，实例的配置主要关心在SGA中的对象，你会发现你的数据库配置与调整时间主要花在这些结构上。但是，与后台进程相关的问题和配置选项也需要进行解释。

6.4.1 系统全局区

系统全局区是实例的主要部分。它含有数据维护、SQL语句分析与重做缓存所必须的的所有内存结构。系统全局区的数据是共享的，也就是说，多个进程可以在同一时间对SGA中的数据进行访问和修改。所有数据库操作都使用包含在SGA中某点上的结构。正如前所述，在数据库非安装阶段，当实例被创建时，分配SGA；当实例关闭时，释放SGA。

SGA组成如下：

- 共享池。
- 数据库缓冲区高速缓存。
- 重做日志缓冲区。
- 多线程服务器（MTS）结构。

在下面几节对它们做出解释。

1. 共享池

共享池（参见图6-2）包括库高速缓存、数据字典高速缓存和服务器控制结构（例如数据库字符集）。库高速缓存存储已提交给RDBMS的SQL语句文本、分析过的格式与执行计划，

以及已被执行的PL/SQL包头与过程等。数据字典高速缓存存储用于分析 SQL 语句的数据字典行。

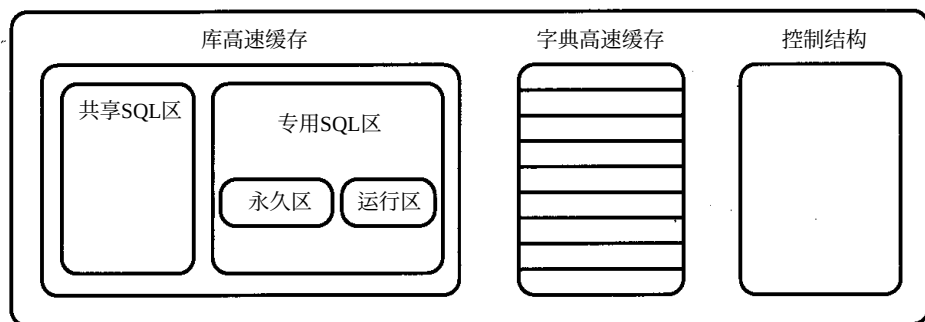


图6-2 当分析和执行SQL语句时使用的共享池高速缓存信息

Oracle服务器用库高速缓存来提高执行 SQL 语句的性能。当一条 SQL 语句提交时，服务器首先查找高速缓存，查看相同的语句是否已被提交或缓存过。如果有，Oracle使用存储的语法分析树和执行路径来执行该语句，而不是从头重建这些结构。尽管使用这个特性不会影响特定的查询的执行，但是使用存储代码可以获得明显的性能提高。

注意 对于使用以前缓存的版本的SQL语句，它必须在所有方面都与缓存版本完全相同，包括标点符号和字母的大小写。Oracle对语句中的文本使用哈希算法识别语句——为了可以使用缓存的版本，语句文本产生的哈希值必须对当前语句和缓存的语句是唯一的。

库高速缓存包括共享和专用 SQL 区。共享 SQL 区包括 SQL 语句语法分析树和执行路径，而专用 SQL 区存储特定的会话信息，例如捆绑变量、环境和会话参数、运行堆栈和缓冲区等。专用 SQL 区在每个事务初始化时创建，在与专用 SQL 区相关的游标关闭时被释放。一个用户会话能够一次打开的专用 SQL 区的数量由 init.ora 参数 OPEN_CURSORS 决定。使用这两个结构，Oracle 服务器能够重用一条 SQL 语句的所有执行的所有共同信息。与此同时，可以从专用 SQL 区中查询执行的特定会话信息。然而值得注意的重要一点是，包含在用户的进程全局区（PGA）中的用户全局区（UGA）的特定会话信息（包括专用 SQL 区），只在使用多线程服务器实例时才由 SGA 拥有，否则，它由专用服务器拥有。

注意 游标使用过程中并不关闭游标的应用会继续为应用分配越来越多的内存，部分原因是为每个打开的游标分配了专用 SQL 区。

库高速缓存中的专用 SQL 区可更进一步分为永久区和运行区。永久区中包含合法的信息，并可应用于 SQL 语句的多个执行中，而运行区中仅包含正在被执行的 SQL 语句的数据。

字典高速缓存含有 RDBMS 引擎分析 SQL 语句所使用的数据字典信息。在这个区域中含有段信息、安全性、访问权限和在可用的自由存储空间等信息。

共享池的大小取决于 init.ora 文件参数 SHARED_POOL_SIZE，它是以字节为单位的。你必须将这个值设得足够大，以确保有足够的可用空间来装载和存储 PL/SQL 块和 SQL 语句。共享区经过长期装卸和卸载数据对象会产生许多碎片，如果在共享池中没有一个足够的连续空间用来装载目标数据，会产生错误。解决这个问题的捷径是运行 SQL 命令 ALTER SYSTEM FLUSH SHARED_POOL。但是如果在数据库操作时，经常遇到共享池错误，你必须增大共享

池。

2. 数据库缓冲区高速缓存

数据库缓冲区高速缓存是影响整个数据库系统运行的重要因素之一。数据库缓冲区高速缓存是由与 Oracle 块相同大小的内存块组成。所有 Oracle 操作的数据在使用前被装入到数据库缓冲区高速缓存中。数据的更新在内存块中完成。由于这个原因，合适大小的缓冲区高速缓存是至关重要的。访问内存的速度要比访问磁盘快几百倍。在 OLTP 环境下，大多数数据操作应该完全在内存中进行，使用早已被装入高速缓存中的数据库块。

Oracle RDBMS 根据最近最少被使用（LRU）列表将数据清出缓冲区高速缓存。最近最少使用列表追踪记录哪些数据块被访问和访问的频繁程度。当一个块被访问或者被查询写了缓冲区时，它被写入最近最多被使用（MRU）列表的末端，当 Oracle 服务器在缓冲区中需要更多空间来从磁盘读入一个数据块时，它去访问 LRU 列表，以确定可以清出哪些块，在 MRU 端最末端的那些块最先被移走。使用这种方法，保证最频繁使用的块保存在内存中。

注意 LRU 装载规则的例外是，通过全表扫描被访问的数据自动地放在 LRU 列表的底端。

可以通过指明表为 CACHE 忽略这个规则。

被修改过的缓冲块称为脏的，它们被记录在脏列表中。脏列表追踪记录所有在内存中被修改而又尚未写入磁盘中的数据。当 Oracle 接收到修改数据的请求时，对高速缓存中的块进行数据修改，同时写入重做日志中，然后该块被放入脏的列表中，对这些数据的随后访问从高速缓存中读取改变的数据的新的值。

Oracle 服务器使用延迟、多个块写以减缓磁盘 I/O 的冲突。这意味着更新一块数据，并不立即更新数据文件中的数据。RDBMS 等到以下时刻才将改变的数据刷新写入数据文件：预定数量的数据块被改变、要求收回缓冲区中的空间以装入新数据、发生一个检查点或者超过数据库后台写的时间。当数据库后台写进程被通知执行一个缓冲区高速缓存写操作时，它将一组块中的数据写入数据文件中。

配置缓冲区高速缓存的关键是保证分配正确总量的内存，以优化缓存数据。将有可能获得的内存资源都分配给缓冲区高速缓存是不必要的；然而，在多数计算机应用中，内存分配持续增长到某一点后性能增长明显降低。这时增加内存以获得一个增长的更好的缓存命中率方面的投入应减少。分配给缓冲区高速缓存的内存能够被更好地利用在其他地方，例如其他的 Oracle 内存结构中。

DB_BLOCK_SIZE 和 DB_BLOCK_BUFFERS 是决定缓冲区高速缓存大小的两个初始化参数。DB_BLOCK_SIZE 参数用于在数据库创建时，设置 Oracle 块大小。这一点将在第 7 章“探讨 Oracle 环境”中进行详细解释。DB_BLOCK_BUFFERS 参数决定分配给缓冲区高速缓存的块的数量。用 DB_BLOCK_SIZE 乘以 DB_BLOCK_BUFFERS 就得出缓冲区高速缓存的内存总数（以字节为单位）。

3. 重做日志缓冲区

重做日志缓冲区用于在内存中存储未被刷新写入联机重做日志文件的重做信息。它是循环使用的缓冲区，这意味着从顶端到底端填充信息，然后又返回到缓冲区的起始点。当重做日志缓冲区填满时，将它的内容写入联机重做日志文件。

重做日志缓冲区的大小是由 LOG_BUFFER 初始化参数决定，以字节为单位，决定在内存中保留多少空间缓存重做日志项。如果这个值设置得过低，进程之间相互竞争，日志写入

(LGWR) 进程读出和写入缓存, 有可能会导性能问题。然而, 这在总数中是很少的, 但却是数据库最活跃的情形, 可以用 V\$SYSSTAT视图进行监控。使域 name等于redo log space requests, 查询V\$SYSSTAT视图value域, 它表明用户进程等待重做日志缓冲区所花费的时间。

为强迫重做日志顺序写入, Oracle服务器使用门控制对缓存的访问。门是一个 Oracle进程对一个内存结构的锁定——在概念上与文件锁定和行锁定类似。一个进程必须持有重做分配门, 才能写入重做日志缓冲区。当一个进程持有分配门时, 其他任何进程都不能使用这个分配门写入重做日志缓冲区。

Oracle服务器使用LOG_SMALL_ENTRY_MAX_SIZE初始化参数限制一次写入的重做的总量。这个参数以字节为单位, 其缺省值随操作系统和硬件不同而不同。对具有多个 CPU的服务器而言, Oracle服务器不允许使用重做分配门所书写的重做日志项所需空间比参数 LOG_SMALL_ENTRY_MAX_SIZE大。相反, 进程必须持有一个重做复制门。可获得的重做复制门的数量等于LOG_SIMULTANEOUS_COPIES初始化参数的值。LOG_SIMULTANEOUS_COPIES的缺省值是系统中CPU的数量。使用重做复制门, 多个进程能同时写入重做日志缓冲区。

可以使用V\$LATCH动态性能视图监控重做分配门与重做复制门(参见第18章“调整内存”, 以获得更多调整重做门的信息)。

6.4.2 Oracle 后台进程

在任意瞬间, Oracle数据库可以处理许多行信息、处理几百个同步用户请求、进行复杂的数据操作, 与此同时提供最高水平的性能和数据的完整性。为了完成这些任务, Oracle数据库将一项大的工作分散到多个程序中, 其中每个程序的大部分操作都是相互独立的, 并扮演一个特定的角色。这些程序称为 Oracle后台进程, 是有效地处理施加在 Oracle数据库的多个操作的有效方法。完全理解后台进程和它们担负的任务, 将有助于分析性能问题、指出瓶颈和诊断数据库中的故障点。

注意 在NT服务器上, 后台进程的执行是作为Oracle Service的多线程实现的, 它允许 Oracle进程更有效地使用共享内存地址空间, 从而减少了NT操作系统处理Oracle操作时的环境改变。

Oracle后台进程如下:

- 系统监控和进程监控进程 (SMON和PMON)。
- 数据库写进程 (DBWR)。
- 日志写进程 (LGWR)。
- 调度进程 (Dnnn)。
- 归档进程 (ARCH)。
- 检查点 (CKPT)。
- 恢复进程 (RECO)。
- 快照进程 (SNPn)。
- 锁进程 (LCKn)。
- 并行查询进程 (Pnnn)。
- 用户和服务器进程 (Snnn)。

它们之中包括处理数据库上用户事务的用户和服务器进程，以及执行对数据库的并行操作的并行查询进程。尽管它们不能归类于 Oracle 后台进程，但是理解它们在 Oracle 环境中所担负的角色非常重要。下面对这些进程分别进行讨论。

1. 系统监控和进程监控

由于种种原因，对 Oracle 数据库的连接可能会发生崩溃、挂起或其他非正常终止。终端用户可能关闭他们的客户机而没有从数据库应用中退出，或者与数据库无关的网络或系统故障可能造成一个数据库自动作业失败。Oracle 服务器必须有能力去处理由于上述原因而引起的失败。

系统监控和进程监控都是自动解决数据库系统问题的后台进程。进程监控（PMON）自动清除中断或失败的进程，包括清除非正常中断的进程留下的孤儿会话、回滚未提交事务、释放被断开连接的进程占有的锁、释放被失败进程占有的系统全局区（SGA）资源，它同时监控服务器和调度进程，如果它们失败则自动重启它们。

系统监控（SMON）担任一个比较小但非常重要的角色。从数据库启动开始，SMON 就作为恢复自动实例的进程。如果最后的数据库关闭不干净，SMON 自动向前回滚正在进行的操作，并回滚尚未提交的事务。SMON 也是管理某些数据库段的进程，收回不再使用的临时段空间，并自动合并和数据文件中相邻的自由空间块。

注意 SMON 仅在当创建表空间或表时的缺省存储参数——pctincrease 不是 0 时，合并表空间中的自由空间。如果想让 SMON 自动地处理这个操作，至少将 pctincrease 设置为 1。

SMON 和 PMON 是两个必需的后台进程。如果它们之中的任意一个在数据库启动时失败，数据库将不能启动。

2. 数据库写

数据库写进程（DBWR）负责将缓冲区中脏的数据块写入到数据文件中。DBWR 不是在每一数据块被修改后立即写入数据文件，而是一直等待，直到满足一定标准后，才成批地读脏列表，并将在脏列表中发现的所有块刷新写入数据文件。这提供了高级别的性能，并最小化数据库输入/输出约束的范围。

当下列情况发生时，数据库刷新脏的块：

- 1) 发生一个检查点。
- 2) 脏列表的长度达到 init.ora 文件中 DB_BLOCK_WRITE_BATCH 参数值的一半。
- 3) 使用的缓冲区数量达到 init.ora 参数 DB_BLOCK_MAX_SCAN。
- 4) DBWR 后台进程发生超时（大约每 3 秒）。

配置 DBWR 后台进程是相当简单的，对于小型或中型数据库而言，它的缺省设置值在许多情况下是足够的。然而，对于较大的、经常活动的或经常有专门需求的专用数据库，必须手工配置一些 DBWR 后台进程参数。

在大多数安装中，有一个 DBWR 进程处理所有数据库的所有写入活动。但是当你发现这个 DBWR 进程不能满足数据库要求时，你可以启动一个以上的 DBWR 进程。在 init.ora 文件中 DB_WRITES 的缺省值是 1，它设置在启动时创建的 DBWR 进程数。在多数情况下，仅当安装的 Oracle 服务器的操作系统不支持同步的 I/O 时，才决定使用多个 DBWR 进程。一旦遇到这种情况，应该创建多个 DBWR 进程。建议你使用与存储数据文件的物理磁盘一样多的 DBWR 进程；另一建议是将这个数量设为与数据库中数据文件的数量相等。增加或减少 DBWR 进程数

进行试验，直到能够最好地满足你的执行。

数据库缓冲区高速缓存也使用门控制对内存结构的访问。LRU门控制在缓冲区高速缓存中缓冲区的替换。有具有多个CPU的极为活动的服务器上，可能会发生对这些门的争用。如果发生这种情况，设置DB_BLOCK_LRU_LATCHES参数值为与为缓冲区高速缓存创建的门的数量相等。这个数值不能超过CPU数量的两倍，并被自动设置为系统中的CPU数量。

另一个影响DBWR行为的init.ora参数是DB_BLOCK_CHECKPOINT_BATCH。这个参数设置在每个检查点DBWR写入的块的最大量（参看下面的检查点进程以得到更详细的信息）。增大这个参数，可以减少DBWR刷新缓冲区的次数。然而将这个数值增加得过大，当DBWR最终刷新缓冲区时，可能会产生不能接受的延迟。

要记住的第三个参数是DB_BLOCK_CHECKSUM。这是一个布尔参数。启用这个参数时，导致每个数据库块被写入时附加一个校验和值。当随后读取该块时，计算校验和值并与存储在数据库中的数值进行比较，如果值不同，将产生错误。当查找数据损坏问题时，这是一个有价值的参数，但是不应该在全部时间都启用这个值，因为对每一个I/O计算并存储校验和会影响性能。

3. 日志写

日志写（LGWR）是第四个也是最后一个必须的后台进程。LGWR是将在系统全局区中重做日志缓冲区重做日志条目写入到联机重做日志文件的进程。LGWR执行写入操作的条件是：发生提交、到达LGWR非活动时限、重做日志缓冲区满度达到三分之一或者DBWR在检查点完成数据缓存块的刷新。如果LGWR在完成代表其他用户的提交刷新缓冲区之前，有一个或多个用户执行提交时，LGWR也能同时处理多用户提交事务。

值得注意的重要一点是：直到Oracle在LGWR完成将重做信息从重做缓冲区刷新到联机重做日志文件之后，Oracle才认为一个事务已完成。在LGWR成功地将重做日志项写入联机重做文件时（并不是改变数据文件中的数据时），将一个成功码返回给服务器进程。

LGWR进程极少能引起数据库操作问题。另外，可以用于配置LGWR进程的选项很少，大多数的配置涉及重做日志缓冲区和支持该缓冲区的内存结构，而不是LGWR进程自身。

然而，有一点例外，LGWR进程处理的次要任务是，执行实施数据库检查点所需要的操作。除非检查点进程被激活，否则LGWR进程完成这一任务。检查点导致LGWR和DBWR都要花费进程和I/O时间。检查点间隔时间越短，发生数据库故障时需要的恢复时间越短，同时减少了必须执行每一检查点所需的工作。当你决定正确的检查点间隔时，你必须权衡所有这些因素。有几个控制发生数据库检查点的参数。

LOG_CHECKPOINT_INTERVAL和LOG_CHECKPOINT_TIMEOUT是能够改变检查点间隔的两个参数，也是触发数据库检查点所必须的时间或条件。当设置LOG_CHECKPOINT_INTERVAL参数时，当一定数量的操作系统块（不是Oracle块）写入重做时，引起数据库中的一个检查点被触发。当设置LOG_CHECKPOINT_TIMEOUT参数时，在这个参数指定的时间间隔（以秒为单位）发生一个检查点。

对这些参数的使用必须要小心。如果使用LOG_CHECKPOINT_INTERVAL参数，应该设置它以使触发检查点的操作系统块的数量与重做日志组的大小相对。记住，当一个重做日志组写满时，一个检查点被触发。要注意的是不要设置不必要的检查点，或者迫使不需要的检查点发生。例如，如果一个重做日志组大小为3MB，而LOG_CHECKPOINT_INTERVAL设置

为2.5MB。当有2.5MB的数据写入重做日志时，参数 LOG_CHECKPOINT_INTERVAL 的值导致发生一个检查点。另外，当重做日志组写满时（仅在又写入 0.5MB的数据后），发生另一个检查点。事实上，这两个检查点将相继发生。

你还可以通过相应地设置重做日志组的大小控制检查点发生的频繁程度。如果你设置你的日志大小使得每小时发生一次日志切换，那么在重做日志组切换后，在一小时内，你将只有一个检查点。但是如果你的日志组大小设置为每 5 分钟发生一个检查点，你将浪费大量的进程活动和I/O次数以执行相关的检查点。

最后一个使用的参数是布尔值 LOG_CHECKPOINTS_TO_ALERT。每当检查点发生时，它为数据库在 alert.log 文件中设置一个标记，并用于试图指出确切的检查点间隔。

4. 调度进程

正如前面所提到的，服务器进程既可以是一个用户进程专有的，也可以在多个用户进程之间共享。使用共享服务器要求配置多线程服务器，在第 37 章“安装和配置 OAS”中将讨论这个问题。当使用共享服务器进程时，至少必须存在一个调度进程（Dnnn），在环境需要时也可能有多个调度进程。调度进程将用户请求传送到系统全局区的请求队列，并将服务器的响应信息返回给正确的用户进程。

使用 init.ora 参数的一个数控制调度进程的数量。参数 MTS_DISPATCHERS 指定调度进程使用的协议及开始使用该协议的调度进程的数量。多协议组可以使用多个 MTS_DISPATCHERS 行进行配置。一个典型的 MTS_DISPATCHERS 行可能像这样：

```
MTS_DISPATCHERS = "tcp, 4"  
MTS_DISPATCHERS = "spx, 2"
```

多协议组也可以在相同的 MTS_DISPATCHERS 参数内中像这样配置：

```
MTS_DISPATCHERS = ("tcp, 4", "spx, 2")
```

参数 MTS_MAX_DISPATCHERS 控制 RDBMS 允许的调度进程的最大数量（参见第 34 章“高级安全选项”以获取关于 MTS 服务配置的详细信息）。

5. 归档进程

归档进程（ARCH）负责将全部联机重做日志复制到归档重做日志文件。这仅在数据库运行在归档模式（ARCHIVELOG）下才发生。归档模式要求时间点恢复，它也允许“热”备份。当 ARCH 正在复制归档重做日志时，没有其他进程能够写入这个重做日志。记住这一点非常重要，因为重做日志是按顺序循环使用的。如果数据库需要转换重做日志，但是 ARCH 还正在按其顺序复制下一个日志，所有数据库的活动将终止，直到 ARCH 完成。还要注意如果归档由于某些原因不能完成复制日志，它将等待直到引起不能写入的错误得到解决为止。

非常值得注意的是在 init.ora 文件中 ARCHIVE_LOG_START 参数必须设置为 TRUE，当数据库启动时，才会自动开始归档。设置数据库处于归档模式并不足以导致 ARCH 自动启动。如果你确实设置了归档模式，但不自动启动 ARCH，当所有联机重做日志写满时，数据库将会挂起，等待你手工归档联机日志。

6. 检查点进程

检查点进程（CKPT）是可选的后台进程，执行 LGWR 进程通常会执行的检查点任务——即用当前版本信息更新数据文件和控制文件头。当有经常性的检查点发生、频繁的日志切换或在数据库中有多个数据文件时，启用这个进程来减少 LGWR 的工作量。

设置CHECKPOINT_PROCESS参数为TRUE可以启用CKPT进程。当CKPT进程运行时，所有与说明检查点有关的其他参数也必须为TRUE。

警告 在Oracle 8.x中，CHECKPOINT_PROCESS参数已被废弃，因为它已被集成入RDBMS中并设置为TRUE。如果在你的Oracle8.x的init.ora文件中包括这个参数，你的实例启动将会失败。

7. 恢复进程

恢复进程（RECO），负责在分布式数据库中恢复失败的事务。当分布式事务配置数据库时（也就是，设置init.ora参数DISTRIBUTED_TRANSACTIONS值大于0），RECO会自动启动。当分布式数据库中有悬而未决的事务时，RECO在很少或没有DBA干预的情况下运行。当数据库连接成功时，RECO试图连接远程数据库并解决悬而未决事务（参见第40章“分布式数据库管理”了解关于RECO和两阶段提交的详细信息）。

8. 快照进程

快照进程（SNPn）处理数据库快照的自动刷新，并通过DBMS_JOB包运行预定的数据库过程。init.ora参数JOB_QUEUE_PROCESS设置启动的快照进程数，参数JOB_QUEUE_INTERVAL决定快照进程在被唤醒以处理挂起的作业或事务之前休眠的时间（以秒为单位）。

9. 锁进程

在并行服务器环境中，多个实例安装在同一个数据库上，锁进程（LCKn）负责管理和协调每个实例占有的锁。可以分配给安装在并行服务器上的每个实例1~10个锁进程，每个实例必须具有相同的数目。在非并行服务器环境中，这个进程不起作用。参见第39章“并行服务器管理”以获得锁进程的详细信息。

10. 并行查询进程

并行查询进程称为Pnnn。根据数据库的活动和你的并行查询选项的配置，Oracle服务器启动和停止查询进程。这些进程涉及并行索引的创建、表的创建及查询。启动的进程的数量与参数PARALLEL_MIN_SERVERS指定的数量相同，但决不能超过参数PARALLEL_MAX_SERVERS指定的进程数。

配置并行查询进程的详细信息参见第38章“并行查询管理”。

11. 用户和服务器进程

应用和工具通过用户进程访问RDBMS。用户进程连接到服务器，既可使用专用方式，也可以在多个用户进程之间使用共享方式（在多线程服务器中）。服务器进程分析和执行用户提交的SQL语句，并将结果集返回给用户进程。这个进程也从数据文件将数据块读入数据库缓冲区。

每个用户进程被分配一部分内存区，称为进程全局区（PGA），PGA的内容根据所连接的数据库模式而有所不同。当一个用户进程通过专用服务器方式连接数据库时，用户的会话数据、堆栈空间和游标状态信息存储在进程全局区中。用户的会话数据包括安全和资源使用信息；堆栈空间含有为用户会话指定的本地变量；游标状态区包括运行时的游标信息、返回的行和游标返回的代码。但是如果用户进程通过共享服务器进程方式进行连接，游标和会话信息被存储在系统全局区（SGA）中。尽管对整个数据库而言，这并不增加对内存空间的要求，但是它需要一个更大的系统全局区来存放这些附加的会话信息。

6.5 事务剖析

为更好地理解前面讨论的实例组件是如何相互交互的，看一个典型的事务在实例结构中的移动。

当一个用户会话使用 SQL*Net 驱动程序连接到服务器会话时，开始一个事务。这个连接可以使用它自己的一个服务进程进行专用连接，或通过调度进程处理的一个共享连接。服务器会话对传递给它的 SQL 语句进行哈希构造，并将该语句的哈希数与已经保存在共享 SQL 区中的语句的哈希数进行比较，如果在共享池中发现有完全一样的语句，使用该语句早已存储的做过语法分析的形式与执行计划。如果在共享池中没有发现匹配语句，服务器进程对这些语句进行语法分析。

接下来，服务器会话查看在数据库缓冲区中是否已经存储了完成该事务所必须的数据块。如果在缓冲区中没有，服务器会话就从数据文件中读取必要的数据，将它们拷贝到缓冲区中。如果事务是一个查询，服务器会话将查询的结果返回给用户进程（执行必要次数的数据块读和拷贝以返回所有数据）。

对一个修改数据的事务，有更多的步骤需要做。例如，假设事务是一个更新。在将必要的数据块读入缓冲区高速缓存中之后，修改内存中的数据块。修改的缓存块标记为脏的，并将它们放入脏列表中。还产生重做信息，并将重做信息存储在重做日志缓存中。

事务继续进行，直到发生以下几个事情中一个。如果事务是相对短期的（例如，对一行销售数据的修改），事务完成，用户提交，发出信号给 LGWR 进程让它将重做日志缓冲区刷新到联机重做日志文件。如果事务相对持续时间很长和很复杂，下列事情有可能发生：

- 产生的重做引起重做日志缓冲区写满三分之一的空间，这会触发 LGWR 进程刷新重做日志缓冲区。
- 放入脏列表中的块数达到门限的长度，这会触发 DBWR 进程将数据库缓冲区所有脏列表项刷新写入到数据文件中，它反过来也使 LGWR 进程向磁盘刷新重做日志缓冲区。
- 遇到一个数据库检查点，这将触发数据库缓冲区高速缓存和重做日志高速缓存刷新。
- 可获得的缓冲区高速缓存中的自由缓存空间下降到规定的门限以下，这也会引起数据库高速缓存缓冲区的刷新。
- 产生一个不可恢复的数据库错误，这迫使该事务中断、回滚并向服务器会话报告错误。

当事务正在处理向重做缓存生成的重做并刷新时，联机重做日志逐渐被填满。当前日志被填满后，LGWR 进程开始写入下一个日志组；与此同时，归档进程将重做日志复制到磁盘或磁带。因为直到所有重做日志信息从重做缓冲区写入到联机重做日志之后，事务才被记录为执行成功的，所以 LGWR 和 ARCH 必须各自有能力在无错误的情况下完成自己的任务。

6.6 监控实例

在大多数情况下，系统全局区和后台进程在没有系统管理员干预的情况下运行。然而，有时会发生问题，这时必须能诊断和修复发生的问题。对 DBA 来说，有几种方法可以监控和追踪实例及其相关的结构。

6.6.1 使用追踪文件

查找关于实例的问题最好的地方是在进程自己的追踪文件中。根据特定的进程和所遇到

的错误，这些追踪文件被写在由参数 `USER_DUMP_DEST` 或者 `BACKGROUND_DUMP_DEST` 所指定的位置。当一个后台进程被终止或者非正常中断一个操作时，通常产生一个追踪文件，包含导致失败的错误信息、当前进程堆栈的转储、当前执行的游标和与问题有关的其他信息。尽管有些信息对 DBA 而言是有用的，但更重要的是收集这些追踪文件，并将它们提供给 Oracle 全球范围客户支持顾问，他们可能会帮助你诊断问题。他们具有可用的工具，能确切指出问题发生的位置。后台进程失败经常在数据库的 `alert.log` 文件中写入一项，或者写到它们各自的追踪文件，这些文件的位置由 `init.ora` 参数 `BACKGROUND_DUMP_DEST` 指定。

6.6.2 通过操作系统追踪

后台进程也能通过使用操作系统命令进行追踪。在 UNIX 环境中，每一个后台进程是一个分立的任务，因此可以分立追踪。通过查看操作系统的进程的内存和 CPU 的使用（通过使用 `sar`、`ps`、`vmstat` 和 `top` 等工具）指明性能问题和失控的查询通常是十分有价值的。有时解决挂起、中断服务器进程或用户进程的唯一方法是在操作系统级中断它们。然而，在试图修改或中断任何其他 Oracle 后台进程时，使用这种方法要非常小心，许多后台进程将会由于非正常中断而使整个数据库崩溃。

在 NT 服务器环境中，追踪后台进程很容易，这是因为全部的 Oracle 实例在 NT 操作系统作为一个称为服务的后台进程实现。各个后台进程作为属于这个服务的线程实现。尽管有大量工具可以用于在 NT 中追踪和监控进程的行为，但是线程管理工具还是相当不常见的。一个解决办法是使用与 NT OS 一起到货的 Performance Monitor 工具进行监控，它监控许多事情，包括属于这个服务的线程的内存的消耗和环境的切换。通过以下查询将 SPID 列从十进制转换到十六进制，可以将 NT 线程 ID 与 Oracle 端的后台进程进行匹配。

```
SELECT spid, name FROM V$process, V$bgprocess WHERE addr = paddr;
```

附录 B “Windows NT 上的 Oracle” 中有更详细的调整和追踪 NT 后台线程的信息。

6.6.3 使用 V\$ 表监控实例结构

DBA 可以使用许多动态性能视图显示实例的信息。当试图发现当前数据库实例的状态和与实例相关的故障问题时，这些视图有很大的价值。

1. 监控数据库的连接

所有连接到实例的用户和后台进程都能够使用 V\$ 视图被监控。V\$process 视图显示所有连接到数据库的进程的信息，包括后台进程和用户进程；V\$bgprocess 含有所有可能的后台进程的列表和一个附加的 PADDR 列，PADDR 列包含后台运行进程的十六进制地址（对于那些没有运行的进程使用 00）。

在 V\$process 表中你可能感兴趣的列如表 6-1 所示。

表6-1 V\$process表的列

列	用 途
ADDR	进程的 Oracle 地址
PID	Oracle 进程的 ID 号
SPID	操作系统进程 ID 号
USERNAME	操作系统进程拥有者

(续)

列	用 途
SERIAL#	Oracle进程序列号#
TERMINAL	操作系统终端标识符
PROGRAM	操作系统程序连接
BACKGROUND	1是后台进程，NULL是用户进程

V\$bkgprocess表中你可能感兴趣的列如表 6-2所示。

表6-2 V\$bkgprocess表中的列

列	用 途
PADDR	Oracle进程地址（与V\$process表的ADDR列相同）
NAME	后台进程名
DESCRIPTION	后台进程的描述
ERROR	错误状态代码（0是无错误）

你可以通过连结V\$process表和V\$bkgprocess表显示所有运行的后台进程的地址和名称，如下所示：

```
SELECT spid, name
FROM V$process, V$bkgprocess
WHERE paddr(+) = addr;
```

连接到数据库的用户会话的信息存储在V\$session视图中，V\$session视图包括许多列和大量能够进行访问的有价值信息。

V\$session视图中你可能感兴趣的列如表 6-3所示。

表6-3 V\$session视图的列

列	用 法
SID	会话标识符
SERIAL#	会话序列号
PADDR	父会话的地址
USER#	Oracle用户标识符（来自SYS.USER\$表）
USERNAME	Oracle用户名
COMMAND	这个会话正在处理的当前命令。要了解命令事务的号，应查看 sys.audit_actions表
STATUS	会话的状态（ACTIVE、INACTIVE、KILLED）
SERVER	会话具有的服务器连接的类型（DEDICATED、SHARED、PSEDUO或NONE）
OSUSER	进行连接的操作系统用户名
PROGRAM	进行到数据库连接的操作系统程序
TERMINAL	进行连接的数据库终端类型
TYPE	会话的类型（BACKGROUND或USER）SQL_HASH_VALUE和SQL_ADDRESS。 用于唯一标识当前正在执行的SQL语句

下面这个查询描述了已连接进程的重要信息。它也说明了进程视图之间相关的方式。

```
col bgproc format a6 heading 'BGProc'
col action format a10 heading 'DB Action'
col program format a10
col username format a8
col terminal format a10

SELECT
  b.name bgproc, p.spid, s.sid, p.serial#, s.osuser,
  s.username, s.terminal,
```

```

DECODE(a.name, 'UNKNOWN', '-----', a.name) action
FROM
V$process p, V$session s, V$bgprocess b,
sys.audit_actions a
WHERE
p.addr=s.paddr(+) AND b.paddr(+) = s.paddr AND
a.action = NVL(s.action, 0)
ORDER BY
sid;

```

通过查询V\$access视图，能显示当前用户正在进行访问的数据库对象的信息，这对断定是第三方应用或无文档说明的过程正在做什么是有用的，对解决安全问题也是有用的。使用DBA帐户运行一个涉及到安全性问题的应用或过程，你能决定安全性应该给予哪个对象。

最后，V\$mtns视图含有共享服务器进程的追踪信息。这个视图包含的列有最大的连接数、服务器启动、服务器中止和服务器的最高水位。

2. 监控共享SQL区

有时能够查看RDBMS引擎内部和正在被执行SQL语句是有用的。视图V\$sqlarea包含共享SQL区中的SQL语句的信息，包括被执行SQL语句的文本、访问该语句的用户数、执行语句时访问的磁盘块和内存块以及其他信息。

注意 在V\$sqlarea中disk_reads和buffer_gets列追踪从磁盘中读入的块数和从缓冲区中读入的块数。这两列是找出使用大量数据库资源的查询的简便方法。

V\$open_cursor视图对研究尚未关闭的游标是有用的。下面的查询显示一个给定的用户SID打开的所有游标：

```

SELECT b.piece, a.sql_text
FROM V$open_cursor a, V$sqltext b
WHERE
a.sid = &SID and
a.address = b.address and
a.hash_value = b.hash_value
ORDER BY
b.address, b.hash_value, b.piece asc;

```

视图V\$sqltext也能用于确定传递给数据库引擎的SQL语句。V\$sqlarea视图仅存储SQL语句的前80个字符，而视图V\$sqltext保存完整的SQL语句。除了在SQL语句中适时加入换行符之外，V\$sqltext_with_newlines视图与V\$sqltext视图是完全相同的。

注意 在V\$sqltext视图中，SQL语句是分块存储的。要获取整个语句，你必须按照PIECE列排序获取SQL语句的所有部分。

3. 监控系统全局区

有两个V\$视图提供了对系统全局区操作的信息。V\$sga视图显示每一个SGA主要组成部分的大小（以字节为单位），包括重做日志缓冲区、数据库缓冲区高速缓存和共享池。V\$sgastat视图包括更多的相关信息，在这个视图中你能查找在系统全局区包含的每个单独的内存结构的大小，包括为堆栈空间、PL/SQL变量和堆栈预留的内存。你也可以查询这个视图，找出SGA区中可获得的自由内存的总数：

```

SELECT bytes FROM V$sgastat WHERE name = 'free memory';

```

4. 监控库和字典缓冲区

有两个视图包含库和数据字典缓冲区的信息。V\$librarycache视图中包含库缓冲区中的每

种类型对象的性能信息；V\$rowcache视图中包含数据字典缓冲区的性能信息（参见第 38章以获得这些视图和它们所包含的信息的更多知识）。

5. 监控并行查询进程

视图V\$pq_sysstat和V\$pq_tqstat包含并行服务器进程及其行为的信息。查询视图 V\$pq_sysstat显示并行查询服务器的当前运行信息，例如忙的和空闲的查询服务器的数量与动态服务器创建和终止的统计数。V\$pq_tqstat视图包含以前使用并行查询服务器运行的查询的信息（参见第38章以获得追踪并行服务器的详细信息）。

6. 监控归档进程

归档进程活动信息存储在 V\$sarchive视图中，可以从这个视图中查询由 ARCH进程写入的归档日志信息（各列的定义参见第 39章）。

7. 监控多线程服务器进程

视图V\$mts、V\$dispatcher和V\$shared_server包含多线程服务器（MTS）和内存结构的状态信息。V\$mts视图包含共享服务器进程的追踪信息，例如启动和终止的服务器数量和运行服务器的最高水位；V\$dispatcher视图包含运行的调度进程的信息，从这个视图中，可以查询调度进程的名字、支持的协议、处理的字节数、处理的消息数、当前的状态和与调度进程相关的其他运行信息；V\$shared_server视图为运行的共享服务器进程提供相同类型的信息（参见第33章“Oracle网络基础”，以获取关于设置和调整共享服务器和调度进程的详细信息）。

第7章 探讨 Oracle 环境

本章要点：

- 创建Oracle环境
- 设计一个优化的弹性体系结构
- 配置Oracle环境
- 理解Oracle软件的环境变量
- 创建你的第一个数据库
- 探索Oracle数据库
- 研究不熟悉的环境

7.1 创建Oracle环境

Oracle数据库是一个强大但复杂的数据存储和检索系统，有能力支持庞大的工作量，同时保持性能、安全性和数据一致性的高水准。一个成功数据库的重要部分是在数据库服务器上软件安装和配置的方法。这一章研究服务器软件和配置在数据库环境中扮演的角色，并澄清在Oracle服务器安装中容易引起混淆的问题。这一章也探讨创建 Oracle环境和创建第一个数据库的策略。最后是如何熟悉你所不熟悉的 Oracle数据库。

当配置一个服务器安装 Oracle软件时，必须考虑到几个关键的因素。这些因素包括：设计的灵活性、管理的方便性与结构的简单性。如果在设计环境时将这些因素谨记在心，你得到的回报是获得有益于达到你的正常运行时间和性能目标的环境，同时避免一个拙劣设计的服务器结构可能产生的问题。

7.2 设计一个优化的弹性体系结构

创建和配置你的 Oracle环境的 Oracle标准叫做优化弹性体系结构，或 OFA标准。这是一套规则和指南，能使你很容易地创建和配置一个产品级的 Oracle服务器。使用OFA标准能够在不牺牲使用性、管理方便性和设计简单性的情况下，创建最灵活的 Oracle环境。

Oracle公司的 Cary Millsap先生在90年代早期编写了 OFA标准。这个结构的权威性的白皮书能够在<http://www.europa.com/~orapub/index.html>中找到。Millsap先生的论文比这里的描述更加详细和深入，他的白皮书中涉及配置整个环境；然而，你所关心的是创建和配置用于支持Oracle数据库的操作系统目录结构和命名规定。任何对实现 Oracle安装的最佳实践感兴趣的 Oracle DBA系统集成者都应该阅读 Millsap先生的 OFA标准论文。

Millsap先生用自己的话概括了 OFA标准建议目的：“好的标准应该有坚实的基础，不要成为空中楼阁。” OFA标准建议正是这样的建议。你特定的环境也许具有一些要求和声明，要满足它们最好使用不包括在（或相反） OFA标准的方法。然而，仔细研究 OFA标准会发现最佳的实践方法，这些方法只有通过艰难的试验、错误和实践才能发现。

7.2.1 创建顶层目录

配置环境的第一步是决定命名和创建操作系统安装点以及你的 Oracle数据库安装的顶层目录。由于具有许多配置项目，该步骤的实现在 UNIX和NT操作系统上有所不同，但是基本的概念仍然是相同的。优化弹性体系结构第一条规则与这个主题相关，如下所示：

“OFA规则1 —— 命名所有包含特定地点数据的安装点与 /pm格式相匹配。这里的p是不代表任何安装点的内容的字符串常量，m是长度固定、用来区分其他安装点的唯一键。”

换言之，创建的顶层安装点各不相同，但它们并不代表任何意思。字符部分（p）应该简短——理想情况是一至两个字符，用数字部分来区分各个不同的安装点。每个安装点应有相同的命名部分和相同的长度（例如u01、u02、u03，或者ora1、ora2、ora3）。

在NT服务器上，不同驱动器和卷标已经按照驱动器盘符分开。这使得使用前面描述的命名规则变得不必要。在每个驱动器上使用命名目录 orant（Oracle缺省命名）指示Oracle应用和文件。

提示 为了减少混淆，UNIX命名惯例完全使用这些解释。但是，除非另外声明，所有给定的结构和命名规定既适用于UNIX环境，也适用于NT环境。

在重新配置你的系统时，如果使用不是特定地点、特定应用或者特定硬件内涵的安装点，能获得很大的灵活性。考虑下面这个例子：Bob是一个初学的DBA，他在UNIX系统上命名安装点，指定每个安装点到物理磁盘卷标。这样，对 Bob来说在驱动器上均衡他的数据文件 I/O是非常容易的。扫视一下路径，他就能说出他的主要数据表空间在 DISK01上，而索引表空间位于DISK05。但是，总会存在这种情况，他的数据库增长了，强迫他增加更多的磁盘和新的硬件。Bob想利用他的新RAID控制器分离和镜像一些卷，这意味着他必须重新组织他的整个驱动子系统以完成这个目标。不幸的是，这也意味着，如果他想保持他的命名惯例，他必须重新命名他的安装点并改变所有他的备份脚本、应用等等，以指向新的路径。Bob可以通过使用唯一的、但是本身没有意义的目录名来避免这种情况。

7.2.2 使用应用目录

实现遵循优化弹性体系结构准则的配置的好处之一是：使用相同的目录结构，能够很容易地安装多个应用和同一应用的多种版本。在 OFA环境中，每一个应用都有它自己的主目录，用于存储二进制、配置文件和类似的运行这个应用所必须的文件。与此相关的另外两个 OFA准则如下：

“OFA准则2 —— 命名应用主目录应匹配的形式是 /pm/h/u，这里的pm是安装点的名字，h从一个小的标准目录名集合中选择。u是应用或者应用的所有者名。”

“OFA准则3 —— 存储每一个版本 Oracle服务器发布软件的目录应匹配的形式是 /h/product/v，这里h是Oracle软件拥有者的应用主目录，v代表软件的版本。”

以下这个例子是执行这个规则的目录模板：

[/安装点]/APP/[应用主目录]/PRODUCT/[版本]/应用名

在这个例子中，你看到应用主目录存储在直接位于安装点之下的 APP子目录中，接着使用分级命名来区分存储在目录下的应用，然后是 PRODUCT目录和版本。使用命名规则的分层命名的例子如下：


```
/u01
  app
    finance
      [...]
    qse
      [...]
    oracle
      admin
      product
        7.1.6
          [...]
        7.3.3
          [...]
        8.0.3
          [...]
```

尽管这种方法可能看上去比需要的复杂得多，但是它带来的灵活性和易于管理性使它是值得采用的。正如你所看见的，在这个安装点之下安装有三个应用——finance、qse和oracle。finance和qse是第三方产品或者是主目录的增长应用，在主目录之下的目录结构与Oracle分层显示的格式相同。在Oracle应用目录中，你能看到product和admin子目录。在product目录里有三种版本的Oracle软件——7.1.6、7.3.3和8.0.3。任何改变或者一个新版本的Oracle软件的安装都不会影响到其他版本。

当用于执行测试和安装新版本的Oracle数据库时，用这种方式分开版本是十分重要的。显而易见的是你需要有能力很容易地安装一个新版本的软件而不影响已安装的产品。在你的新版本被测试后，搬移新版本是一个简单的事情，只需设置正确的环境变量为一个新目录即可。在你空闲时可以删除旧版本。

7.2.3 管理数据库文件

当DBA同时管理多个数据库时面临的困难成为创建OFA指南的必要性之一。需要一个方法将管理信息和数据文件组织成可管理和可预测的格式。你在前面所见的admin目录，被用作存储和管理与数据库相关的文件，并极大地改进了单人持续追踪多个数据库的能力。下一个与这个结构相关的OFA准则是：

“OFA规则4——对每一个db_name=d的数据库，数据库的管理文件存储在/h/admin/d子目录中，这里的h是Oracle软件拥有者的登录主目录。”

下面是admin目录结构的例子：

```
/u01/app/oracle/admin
  PROD
    bdump
    udump
    cdump
    pfile
    sql
    create
```

在admin目录下的子目录解释如表7-1。

init.ora文件、追踪和转储文件、alter.log等等都存储在admin这个集中管理的目录中，这使管理由Oracle数据库产生的大量数据十分容易。你也可以在需要时增加目录来存储其他数据。

注意 在UNIX平台上创建一个从\$ORACLE_HOME/dbs目录到admin/pfile目录的init.ora

文件链接。这保证Oracle缺省配置的完整性，并可以使你从OFA结构中受益。在NT平台上，要么在启动数据库和创建一个符合OFA规则的结构时，必须对init.ora文件指定全部路径，要么必须在缺省的\$ORACLE_HOME/database目录中存储init.ora文件，来保持缺省的Oracle配置的完整性。

表7-1 admin 目录

目录名	用 途
bdump	后台转储文件（BACKGROUND_DUMP_DEST的值）
udump	用户转储文件（USER_DUMP_DEST的值）
cdump	内核文件（仅在UNIX中）
pfile	init.ora文件和任何其他数据库初始化参数
sql	数据库管理的SQL文件
create	用于创建最初的数据库和数据库对象的脚本

7.2.4 命名约定

当在一个数据库服务器上管理多个数据库时，文件命名约定是十分重要的。OFA标准对数据库文件的命名给出如下的规定：

/pm/q/d/control.ctl —— 控制文件。

/pm/q/d/redon.log —— 重做日志文件。

/pm/p/d/tn.dbf —— 数据文件。

这些命名约定定义如下：

- pm 是安装点名。
- q是一个字符串，表明Oracle数据与所有其他文件分开。
- d 是数据库的db_name。
- n是一个对给定文件类型有固定长度的区分键。
- t是Oracle表空间名。

除了控制文件、重做日志文件或与数据库 d相关的数据文件之外，不要在pm/q/d下存储其他的任何文件。

采用这个标准的严格程度取决于个人的偏爱，但是命名的约定这个思想不应该被忽视。用一个有意义的文件扩展名，在数据文件名中包括表空间名，并将所有的数据文件保持在它们各自不同的目录中。

一个供选择的保持最初命名约定的模板可能是 tn_SID_n.ext，这里tn是数据文件的表空间的缩写，SID是数据文件所属的数据库，n是数据文件号。

尽可能地保持表空间的缩写简短，如果可能，保持它为一个字母符号集。这样可以产生一致的报告。而且，重做日志文件的数字部分应该同时表明日志组和日志成员 —— 使用数字和字符的组合，例如01a（第一个日志组，第一个成员）和02c（第二个日志组，第三个成员）等。命名使用的替代符号如表7-2所示。

正如你所见，为你的数据库文件实现一个命名约定可以产生不包含不确定意义的名字；通过查看它们的名字，你可以立即确定它们的功能。

OFA规则也表明数据库的数据文件、重做日志文件和控制文件应该存储在安装点之外的

专用目录中。下面列出的目录结构是遵循这个建议的目录结构，其中含有四个数据库的数据。

```
/u01
  app
  oradata
    PROD
      [...]
    PPRD
      [...]
    DEVL
      [...]
    TEST
      [...]
```

表7-2 数据库文件举例

文 件 名	解 释
syst_PROD_01.dbf	PROD数据库的第一个系统表空间数据文件
ctrl_TEST_02.ctl	TEST数据库的第二个控制文件
redo_PPRD_01a.log	PPRD数据库的第一个重做日志组的第一个成员
redo_PPRD_02c.log	PPRD数据库的第二个重做日志组的第三个成员
data_PROD_02.dbf	PPRD数据库的第二个数据表空间数据文件
initTEST.ora	TEST数据库的初始化参数文件

所有的数据文件、日志文件、控制文件和归档重做日志文件都存储在 oradata子目录下的目录中。每个数据库都有它自己的目录，仅有数据库文件存储在这些目录中。

7.2.5 综合

一个聪明人曾说过关于标准最重要的事情是灵活掌握。当描述 OFA指南时，这是特别适合的。没有特别好的创建 Oracle服务器环境的解决策略，因为不存在两个完全相同的环境。

清单7-1和清单7-2是在UNIX和NT下的OFA实现例子。这主要是给你指明思想，让你开始工作。当你重新工作时，你可能会发现一些特殊方法比那些现存的方法更适合你的特定需要。下面也给出各部分的解释。

清单7-1 实现OFA准则的UNIX例子

/根目录	
u01	u01安装点
app	应用目录
oracle	Oracle应用目录
admin	系统管理目录
PROD	PROD数据库目录
pfile	初始化参数文件
bdump	后台转储文件
udump	用户转储文件
cdump	内核文件
create	数据库创建脚本
sql	SQL脚本
PPRD	PPRD数据库目录
[...]	与PROD有相同的目录结构
TEST	TEST数据库目录
[...]	与PROD有相同的目录结构

product	应用文件目录
7.1.3	7.1.3版本文件
[...]	7.1.3版本的Oracle主目录
8.0.3	8.0.3版本文件
[...]	8.03版本的Oracle主目录
qse	QSE应用目录
admin	系统管理目录
[...]	QSE系统管理目录
product	应用文件目录
[...]	QSE版本和文件
oradata	Oracle数据库文件目录
PROD	PROD数据库文件
PPRD	PPRD数据库文件
TEST	TEST数据库文件
u02	u02安装点
oradata	Oracle数据库文件目录
PROD	PROD数据库文件
PPRD	PPRD数据库文件
TEST	TEST数据库文件
[...]	

清单7-2 实现OFA规则的NT例子

C:	驱动器盘符
oracle	Oracle软件目录
home	Oracle主目录
[...]	Oracle主目录内容
oradata	Oracle数据库文件
PROD	PROD数据库文件目录
[...]	PROD数据库文件
PPRD	PPRD数据库文件目录
[...]	PPRD数据库文件
admin	Oracle系统管理目录
PROD	PROD管理目录
[...]	与UNIX操作系统中的结构相同
PPRD	PPRD管理目录
[...]	与UNIX操作系统中的结构相同
D:	驱动器盘符
oracle	Oracle软件目录
oradata	Oracle数据库文件
PROD	PROD数据库文件目录
[...]	PROD数据库文件
PPRD	PPRD数据库文件目录
[...]	PPRD数据库文件
[...]	

使OFA标准起作用的一个关键因素是将应用与数据库文件分开。创建目录 admin和oradata 目录以保证所有特定的数据库文件与 Oracle软件分开。这对满足前面讨论的原则之一很必要：新软件版本的简单的和透明的分离。

Millsap先生的论文的最后一條OFA准则如下：

“OFA规则6——仅在特定设计用来存储它们的文件中指出显式路径名，例如 UNIX /etc/passwd文件和Oracle oratab文件；仅在/etc/group 中指出组成员。”

如果遵循这个规则，这个规则能够帮助你避免犯每一个人都易犯的错误：快速创建一个应用并使之成为一个每天要用的常用工具，但在任何系统重新配置时都会崩溃。避免在一个壳脚本、批处理文件或应用程序中使用硬编码目录名。当使用 Oracle工具工作时，实现这一条比想象的要容易。任何目录或你可能需要用的值可能早已作为一个环境变量或注册键实现了。

7.3 配置Oracle环境

Oracle服务器是一个复杂的交互式程序和进程的集合。所涉及的软件环境变量很复杂，经常是难以理解。存在各种不同的参数控制 Oracle程序的行为、为配置或共享文件定位、定义使用的语言和字符集及执行其他的大量任务。在 UNIX机器上，这些参数作为环境变量进行存储，这些环境变量通常从称为用户登录脚本的 coraenv或oraenv文件中装入。在 16位的Windows计算机，这些参数存储在 oracle.ini文件中。在 32位Windows计算机（Windows 95、Windows NT等），这些配置参数被存储在注册表中。

环境变量文件或注册表键值在机器上第一次安装 Oracle产品时创建。其后安装的软件增添和更新这些配置。系统程序员或 DBA也能通过手工修改来调整这些配置、查找故障点和自定义Oracle环境。一个完整的软件配置参数的描述，能够在你的操作系统中特定的 Oracle文献中找到。花一小会儿时间熟悉你的操作系统中各种不同参数；许多问题都能通过细心地研究这些参数得以解决。

注意 不要混淆Oracle配置参数和一个数据库的初始化参数。Oracle配置参数应用于安装在一个计算机上的Oracle软件，而初始化参数构造单个数据库。

有几个你应该立即熟悉的环境变量，因为它们在运行的Oracle服务器中扮演着重要的角色。ORACLE_HOME定义 Oracle软件的基本路径，这在本章的后面各节中将详细解释。ORACLE_SID为你的会话设置缺省的数据库 SID（系统识别符），许多工具用ORACLE_SID来决定所要连接的数据库。知道环境变量所设置的值是十分重要的——如果有错误的值，既可能导致在错误的数据库上执行，也可能导致使用错误版本的 Oracle工具。

7.4 理解Oracle软件的环境变量

当在你的服务器或客户机上安装 Oracle软件时，Oracle目录结构被创建用来存储 Oracle程序使用的二进制可执行文件、共享库、配置文件、追踪文件等。这个目录树跨不同服务器平台和Oracle软件版本被半标准化。服务器和客户端软件存储在相同目录中。

提示 在UNIX机器上，在同一 Oracle目录中不能混合不同版本的 Oracle软件。在Windows机器上，不能在同一个Oracle目录中混合16位和32位版本的Oracle软件。

Oracle软件通常使用为该任务特别创建的操作系统帐户安装在服务器上。这个缺省命名为Oracle的用户ID，必须有权力操作所有 Oracle目录（包括那些用于软件安装和 Oracle数据文件存储的目录）和文件。使用 OFA准则的服务器配置，用户主目录是 /mount/app/oracle。Oracle帐户阻止未授权用户访问是十分重要的问题。在许多环境中，访问这个帐户会危害整个数据库环境的安全。

以下部分解释 Oracle 目录结构的各个部分。特别是，你将看到 Oracle 主目录是什么，在 UNIX 和 NT 这两个平台上这些服务器结构有何不同，在 UNIX 和 NT 平台上不同版本的 Oracle 软件如何能够共存。

7.4.1 ORACLE_HOME 主目录

Oracle 目录树的顶层被称之为 Oracle 主目录。它在服务器中被定义为环境变量或注册表的设置被命名为 ORACLE_HOME，它是一个必须的配置参数。许多 Oracle 的可执行文件使用 ORACLE_HOME 变量来定位共享代码、配置文件等。例如，一个 UNIX 服务器上，一个 Oracle 程序可能在 \$ORACLE_HOME/network/admin 中查找配置文件，而不是在 /u01/app/oracle/network/admin 中查找配置文件。

注意 有些 Oracle 环境变量（如 TNS_ADMIN）忽视缺省的用于定位文件的 ORACLE_HOME 目录。

ORACLE_HOME 目录应该在数据库登录时设置，可以通过用户登录脚本、注册表值或 .ini 文件值设置，这取决于你的体系结构。其他环境变量值，如 LD_LIBRARY_PATH（在 UNIX 系统中）和 PATH，应该参照于先前设置的 ORACLE_HOME 变量。这是 OFA 规则的要求之一：使硬编码目录值只被放置在一个系统管理位置上。

7.4.2 在 UNIX 上运行 Oracle 多种版本

当启动一个数据库时，ORACLE_HOME 变量值决定用哪个版本的 Oracle 服务器可执行文件和库来打开一个数据库。通过操作和控制 ORACLE_HOME 值，你能定义使用哪个版本的 Oracle 软件。这就是多版本的 Oracle 数据库如何能够同时在同一服务器上运行的原因。下面显示了 Oracle 目录结构的例子（在 UNIX 服务器上）：

```
/u01/app/oracle/product
  7.1.6
    bin
    rdbms
    [...]
  7.3.3
    bin
    rdbms
    [...]
  8.0.3
    bin
    rdbms
    [...]
```

在这个例子中，三种版本的 Oracle 数据库安装在按照 OFA 准则定义的目录结构中。在任何时候，运行服务器的多个版本可执行文件的多个数据库能够运转。如果 DBA 登录并设置他的 ORACLE_HOME 为 /u01/app/oracle/product/7.1.6，那么他启动的数据库运行 7.1.6 版本的服务器可执行文件；如果 DBA 登录并设置他的 ORACLE_HOME 为 /u01/app/oracle/product/8.0.3，那么他使用 8.0.3 版本的服务器可执行文件。

当在 Oracle 环境中工作时需要记住两个版本：

- 1) 软件自身的版本(服务器可执行文件和库)。
- 2) 数据库版本。

这些版本是分开的和彼此之间是有区分的。Oracle不对服务器可执行文件和数据库执行版本检查。例如，用一个7.3.3版本的软件启动7.1.6版本的数据库时，你将不会接收到错误。当多个版本的Oracle安装在同一服务器上，必须注意做到不要混合软件和Oracle版本。

注意 在UNIX环境下，缺省的ORACLE_HOME和相关的环境变量设置在数据库登录文件中，通过提示用户指出要使用的SID，在oratab文件中查找SID，并用在oratab文件中所查找的SID值设置ORACLE_HOME。在Windows环境中，ORACLE_HOME从注册表或.ini文件中读取值。

7.4.3 在NT环境下运行Oracle的多个版本

因为UNIX和NT服务器结构不同，在NT服务器上运行Oracle服务器多个版本的软件，处理的方法不同于UNIX。在NT服务器上一个实例作为带有后台进程的服务实现，同时服务进程产生大量线程。这个服务从Windows NT注册表获得它的所有结构信息。因为在NT服务器仅有唯一的注册表有一组值，在任何时候，仅有唯一的ORACLE_HOME能够在服务器上定义。这意味着在NT服务器运行多个版本的Oracle软件不能通过操作ORACLE_HOME环境变量来进行。

为了使多个Oracle版本存在单个NT服务器上，所有特定版本的目录、可执行文件和文件都应附加有主要的版本号。例如，如果在NT服务器上数据库版本是7.2.3，Oracle服务器可执行文件被命名为oracle72.exe，而7.3.3版本被命名为oracle73.exe。照此方法，RDBMS目录被命名为rdbms72或rdbms73，Oracle应用如tkprof或Export在7.2版本的数据库上被命名为tkprof72或在7.3版本的数据库为rxo73。

注意 由于这种命名约定，不支持有多个较小版本的Oracle软件同时安装。例如7.3.2版本和7.3.3版本的软件在它们的目录和文件中有相同的后缀73，因此它们不能在同一NT服务器上共存。因为容易引起混淆和存在事故发生的隐患，所以尽可能地避免在同一服务器上运行多个版本NT。

Oracle提供一个应用用来在16位Windows客户和Windows 95、Windows NT机器进行主目录的转换。这不同于用多个ORACLE_HOME目录运行，这个程序所做的所有工作就是将init.ora或注册表中的目录引用从一个位置改变到另一个位置。这使你能在多个ORACLE_HOME目录中安装和测试软件，但是它不能使程序从分开的ORACLE_HOME目录中同时运行。

16位的Window软件使用oracle.ini文件设置Oracle软件配置变量。32位的Windows软件使用/HKEY/LOCAL_MACHINE/SOFTWARE/ORACLE注册表键来设置Oracle环境变量。鉴于此，16位和32位版本的Oracle软件能够共存于32位的Windows操作系统中。建议使用不同的ORACLE_HOME(缺省值是orawin和orawin95或orant)，通过它们各自的.ini文件和注册表设置来进行分开管理，避免在不同版本之间由于共享代码不符而引发的问题。

用于16位Oracle软件的驱动器和程序不能在32位版本上运行，反之亦然。你将经常看到与此相关的使用SQL*Net的程序(如ODBC驱动)和依靠Oracle提供的支持库和DLL库的第三方编程工具。在这种情况下，同时运行16位和32位版本Oracle软件的唯一方法是保持每项工作平稳地运行。

7.4.4 ORACLE_HOME目录

如前所述，ORACLE_HOME变量指向Oracle软件目录树的顶层。ORACLE_HOME包含其他目录，这些目录中又包含其他目录和文件。在 ORACLE_HOME内定位目录是在所有 Oracle 安装中相当标准的。每个目录包含用于特定的程序或功能的文件。

表7-3列出了能够在Oracle安装中找到的一些重要目录和它们的作用。

表7-3 重要的ORACLE_HOME目录和用途

目 录 名	用 途
BIN	Oracle可执行文件和共享DLL库
DBS	Oracle服务器文件和init.ora文件的缺省目录
DATABASE	开始的数据库和缺省的Oracle数据库文件（NT）
RDBMS	数据库文件
RDBMS/ADMIN	创建系统数据库的对象、角色和视图的 SQL脚本
RDBMS/TRACE	缺省的数据库转储目录
NETWORK	Oracle网络文件
NETWORK/ADMIN	SQL*NET配置文件
NETWORK/TRACE	缺省的SQL*NET转储目录
PLSQL	PL/SQL配置文件和SQL脚本
DOC	联机文档
SYSMAN	Oracle企业管理器文件
PLUS	SQL*PLUS文件

通常，admin目录包含与系统管理对象有关的 SQL脚本，这些系统管理对象与它们所在的产品目录相关。对DBA来说最需要关心的目录是\$ORACLE_HOME/rdbms/admin，在这个目录中你能看到用于运行生成数据库对象、包和过程等的 SQL脚本。\$ORACLE_HOME/network/admin包含用于配置SQL*Net通信所有必需的文件。\$ORACLE_HOME/dbs包含sql.bsq文件，这个SQL文件在数据库创建期间用来运行创建数据字典表。

7.4.5 其他重要的配置文件

在你安装Oracle服务器时应该了解几个其他文件。oratab、oraenv和coraenv文件仅能在UNIX服务器中见到。oratab是系统中每个数据库在其中占据一行的文本文件，每行包括数据库SID、特定数据库的Oracle主目录和一个用来表明数据库是否自动启动或者关闭的标志。有几个由Oracle提供的程序和脚本（如dbstart或dbshut）使用这个文件来识别数据库在服务器中的位置和它们的版本号。dbstart和dbshut也使用自动启动标志来选择在哪个数据库上执行它们的活动。

oraenv和corenv是UNIX环境中的脚本文件，用于定义和设置运行 Oracle程序和工具所必需的环境变量。oraenv文件用于sh或ksh环境，而文件 coraenv用于csh环境。oraenv和coraenv也访问oratab文件，以决定正确的ORACLE_HOME，登录数据库键入的SID基于这个值。这些文件是当你开始自定义你的UNIX Oracle环境变量时，最初开始查找的位置。

在NT服务器中，你应该了解口令文件，它是由 oradim工具为每一个数据库创建的。这个文件包括内部用户帐户口令——这个帐户在数据库启动和关闭时使用。这个口令文件被命名为oraSID.pwd或pwdSID.ora，它被设置标志、隐藏并存储在 \$ORACLE_HOME/database目录中。它用一定的安全配置来验证数据库的远端 DBA。另外，在NT上，这个口令文件在内部与

该SID的控制文件保持同步。

7.5 创建你的第一个数据库

由于你已经获得了关于 Oracle环境、Oracle数据库和实例的知识，现在该了解创建 Oracle数据库的过程了。

创建Oracle数据库包括一些离散的步骤：

- 1) 为一个新数据库创建初始化参数文件和系统管理目录。
- 2) 创建实例。
- 3) 创建数据库。
- 4) 运行数据库创建后的步骤。
- 5) 创建支持的数据库对象。
- 6) 安全化缺省帐户。
- 7) 更新系统文件以识别新数据库。

在以下部分对每一步进行解释。

7.5.1 创建初始化参数文件

创建一个新的数据库的第一步是复制缺省 init.ora文件（在UNIX平台上，能在\$ORACLE_HOME/dbs中找到；在NT服务器上，在\$ORACLE_HOME/database目录中）到唯一标识参数文件属于哪个特定数据库的文件。这个文件包括一个新建数据库的特定参数设置。init.ora文件按Oracle缺省的命名惯例是 initSID.ora；这里的SID是你为新数据库选择的系统标识符。如果在启动数据库时，一个参数文件没有显式地被指定，Oracle工具将查找用这个名字命名的文件名（用ORACLE_SID来填写文件名的SID部分）。编辑复制的文件，用有意义的值取代缺省的参数值。至少要确保controlfile参数、db_block_size参数和db_name 参数是正确无误的。

现在该是创建用来存放Oracle数据文件、后台和用户追踪文件和其他你要使用的管理目录结构的时候了。参见7.2节“设计一个优化的弹性体系结构”，其中有这个配置的详细信息。

7.5.2 创建实例

实例的创建是最容易引起误解的与Oracle数据库系统管理有关的任务之一。正如前面所讨论的，实例是由后台进程和包含在服务器内存地址空间中的内存结构所组成的 Oracle RDBMS的特定部分。创建和运行实例在UNIX和NT平台之间有很大差异。以下部分讨论涉及这两个环境的不同步骤。

1. 在UNIX上创建实例

在UNIX平台上，Oracle实例不是一个永久的结构，而是每次数据库启动时使用 init.ora文件中特定的数据库启动参数来创建的。实例名使用环境变量 ORACLE_SID的值。实例创建的确切时间发生在非安装阶段。当你启动数据库到非安装态时，你所做的工作仅只是分配系统全局区和启动服务器后台进程。取决于你的 UNIX平台，实例名（或SID，由ORACLE_SID值定义）也有可能是后台进程任务名的一部分。

2. 在NT上创建实例

对于创建实例和用实例工作，在NT服务器上所涉及到的内容要比 UNIX服务器上多很多。

由于NT的结构，所有UNIX控制台进程都使用NT服务执行。一个服务必须定义和存储在 NT注册表中之后，才能使它激活。因此对每一个你拥有的 NT数据库，你必须为数据库创建一个与主机实例相关的服务。不同于 UNIX服务器，不管实例是否被启动， NT Oracle实例存在于一个永久存在的物理操作系统结构中。

在多数安装中，每一个数据库被分配两个NT服务，只有其中的一个服务(即OracleServiceSID)是数据库操作所需要的。OracleServiceSID服务由嵌入它的定义的初始化参数文件创建，这就是为何当 NT创建实例时知道查找哪个 init.ora文件和该实例是为哪个数据库创建的(使用在 init.ora中的controlfile参数)。另一服务(OracleStartSID)是当NT服务器启动时提供的自动启动你的实例和数据库的一种方便方法。当 OracleStartSID被启动时，它自动启动 OracleServiceSID并打开OracleServiceSID所指向的数据库。创建 OracleStartSID服务在本章后面的各节讨论。

运行Oracle实例所需的服务使用 oradim工具创建。记住 oradim正如所有其他主要的 Oracle NT可执行文件一样，将在其上附加 Oracle版本的前两位数字，例如 oradim80.exe。 oradim有两种操作方法——图形用户界面(GUI)接口和命令行方式。那些习惯使用 NT的用户将可能喜欢用GUI接口；而对UNIX管理员，更喜好用命令行方式。用无参数形式激活 oradim将自动启动GUI接口。GUI接口能够用来创建初始化服务和实例；而命令行选项能够有更多的控制能力和更强大的系统管理选项。

关于oradim值得注意的一件事是，它不能将错误信息输出到标准输出或打印到屏幕上，而是将所有的错误信息写入位于 \$ORACLE_HOME/rdbms目录的 oradim.log文件中。在未来 Oracle版本中将有望改变这种小小的不便。这种错误处理设计的原因已超出理解的范围。

当调用 oradim GUI接口时，显示两个要求填入数据库参数的屏幕。跟随指示， oradim创建Oracle服务、创建数据库并运行 catalog.sql和catproc.sql(在7.5.4节“运行创建数据库后的步骤”中讨论)。确保改变给出的数据库名、实例名、数据文件和参数文件的缺省值。

运行字符命令的 oradim是使事情开始变得有趣。在命令行中，你能创建、编辑和删除已有的Oracle服务。对有经验的Oracle DBA来说，对命令行方式的oradim的喜爱要胜过GUI，因为它有更高级的控制和功能。对这个例子，假设使用 oradim命令行方式。

最基本的oradim命令行使用格式如下：

```
oradim -command -sid SID [flags]
```

在表7-4中描述了oradim命令。

表7-4 oradim 命令行选项

命 令	功 能
NEW	创建一个新的数据库实例服务
EDIT	编辑一个现有的实例服务
STARTUP	启动一个实例和/或数据库
SHUTDOWN	关闭一个实例和/或数据库
DELETE	删除实例服务

需要意识到的一个重要的事情是，实例服务完全独立于与它们相关的数据库。在 NT服务器中一个数据库能够存在，而不需要与之相关的实例一定存在。并且，在创建这个服务时使用的oradim版本指定在启动实例时将运行的 Oracle服务器可执行文件(oracle.exe)的版本。

要创建一个Oracle服务，发出如下命令：

```
oradim -new -sid [sid_name]
-pfile [init.ora directory]
-intpwd [internal password]
-startmode auto
```

这就创建了 OracleServiceSID 和 OracleStartSID，其中 SID 为设定的 sid_name 值。auto 的 startmod 创建 OracleStartSID 文件。pfile 参数给出用于配置由 OracleServiceSID 服务启动的实例的 init.ora 文件的位置。

更详细的使用 oradim 命令的信息和例子，参见特定的 Oracle NT 操作系统文献。

7.5.3 创建数据库

在配置好实例之后，你可以创建数据库，这就是你真正发出 CREATE DATABASE 命令的时候。CREATE DATABASE 命令创建系统数据文件、表空间、控制文件、初始的重做日志组、系统回滚段和运行 Oracle 数据库所需要的内部数据库表。

CREATE DATABASE 命令的语法能够在 Oracle Server Reference Manual 中找到。在 CREATE DATABASE 命令中指定正确大小的数据文件是十分重要的，这个数据文件被分配给系统 SYSTEM 表空间。由于硬盘驱动器空间是相对不太昂贵的，将文件大小设置为比你希望使用的大两到三倍（至少 50~100MB）。如果数据库将大量使用存储包、过程、函数和触发器，那么需要分配更多的空间，因为这些对象的代码存储在 SYSTEM 表空间中，会占用相当多的空间。同时将你的 MAX 参数设置得大一些。如果没有明确声明 MAXDATAFILES、MAXLOGFILES 等的状态值，将使用缺省值，通常缺省值十分低。唯一的缺陷是控制文件将比你所需要的大几个 KB——这几乎算不上是一个缺陷。你还要指定字符集（CHARACTERSET）的正确值。它是在数据库安装之后不能改变的几个参数之一。

在运行 CREATE DATABASE 命令之前，确保在 init.ora 文件中控制文件参数设置正确，你的 ORACLE_SID 设置为正确的数据库名。如果你在 init.ora 参数中设置了 db_name 参数，要确保它与你的 Oracle 服务器的 SID 值相同。将 SQL 语句 CREATE DATABASE 保存在一个文本文件中，可以减少由于打字错误发生错误的概率和能够提供一定级别的自我参考文献。同时保证在 NT 控制面板运行 Services applet 来启动 OracleServiceSID 服务并保证 OracleServiceSID 正在运行。

在 UNIX 上运行 svrmgrl；在 NT 系统中对于 7.3.x 运行 svrmgr23.exe；或对于 8.0.x 运行 svrmgr30.exe。以 internal 连接到数据库。对 7.3.x，你得到一个状态信息，表明你已“连接到一个空闲的实例”。这表明实例正在运行，但是在其上没有依附数据库。但是对于 8.0.x，你仅接收到“已连接”消息，尽管实际上实例是空闲的。发出以下命令来启动非安装态的数据库：

```
startup nomount pfile=[init.ora file path and name];
```

当数据库开始运行你的 CREATE DATABASE 脚本时，完成这个命令所需要的时间主要取决于你所产生的系统数据文件的大小。当创建命令完成后，数据库自动打开。这时，你有一个工作的（虽然是空的）Oracle 数据库。

7.5.4 运行数据库创建后的步骤

在数据库创建完成之后，有两个 Oracle 提供的 SQL 脚本必须立即运行：

■ catalog.sql

■ catproc.sql

在创建数据库之后， oradim GUI工具自动执行这些脚本。这两个脚本能够在 \$ORACLE_HOME/rdbms/admin中找到。catalog.sql创建数据字典视图和 V\$视图同义词，而 catproc.sql运行安装过程化选项或 PL/SQL对象和它支持的数据库结构需要的脚本。这两个脚本之中的任意一个运行失败，将导致奇怪的和不可预测的错误。

另一个你可能想运行的脚本是 pupbld.sql。它创建用户的环境资源表，这些 Oracle提供的表能够用来限制 Oracle程序访问数据库。产品用户环境资源表在今天很少被使用，这是由于许多第三方应用对数据库的访问使用本地或 ODBC驱动，它们不检查这些表对资源的限制。但是如果你没有创建这些表，每当不是 SYSTEM用户的用户通过 SQL*Plus登录数据库时，他在屏幕上会得到一个令人讨厌的错误信息。当创建这些表时，应以 SYSTEM用户登录。

最后，如果这个数据库涉及到复制，你需要运行 \$ORACLE_HOME/rdbms/admin/catrep.sql。这个脚本创建支持复制选项所需要的过程 and 对象。

提示 对于数据库进程的创建，非常有意义的一点提示是在你运行数据库创建脚本之前，启动一个缓冲文件，然后当脚本运行之后，浏览它的错误。这将帮助你迅速查找错误。

7.5.5 创建数据库支持的对象

现在你有了一个数据库框架，你能够开始创建能够使你的数据库支持产品级工作量的支持对象。创建支持对象的关键是灵活性和可管理能力。如果你能将这两个目标记在心中，你的数据库安装将是成功的。

首先你需要做的是创建一个回滚段，在创建其他的你支持的数据库对象时需要用到这个回滚段。在创建时，你的数据库只有一个在系统表空间中创建的系统回滚段。这个回滚段仅只被用作涉及系统表空间的事务，它不能被删除或脱机。使用如下命令创建一个临时回滚段：

```
create rollback segment rbs_temp
storage (initial 10k next 10k minextents 3 maxextents 121)
tablespace system;
```

发出如下命令使回滚段联机：

```
alter rollback segment rbs_temp online;
```

现在创建你的产品表空间。至少需要创建列在表 7-5中的表空间。

表 7-5 数据库表空间的一些例子

表空间名	用 途
TEMP	存储临时数据库段
TOOLS	存储第三方或内建数据库工具对象
RBS	存储回滚段
RBS_02	存储用于装载大块数据的回滚段
USER_DATA	终端用户的缺省表空间
USER_INDEX	终端用户的索引表空间

你创建这些表空间的目标是分离操作性的和应用的数据和功能。 TEMP、RBS和RBS_02表空间包含的对象是 Oracle数据库工作所必须的，而 TOOLS、USER_DATA和USER_INDEX

表空间包含用户数据。参看本书第四部分“性能调整”，其中有更为详细的使用表空间分离数据、I/O和功能的信息。

同时，创建你的产品回滚段。在 RBS 表空间至少创建两个回滚段来支持 OLTP 事务。这些段应该相对较小，但有许多区间。将它们命名为 RBS## —— 例如，RBS01、RBS02 和 RBS03。在 RBS_02 表空间创建一个用于装载大块数据的回滚段，这个段应该较大而有较少的扩展，称这个回滚段为 RBS_LOAD## —— 例如 RBS_LOAD01。使 OLTP 回滚段联机，将它们的名字添加在 init.ora 文件 rollback_segments 列表中。

当你做完这些工作后，删除你早期创建的临时回滚段，使用如下命令：

```
drop rollback segment rbs_temp;
```

7.5.6 缺省帐户的安全性

一个新安装的数据库总有两个缺省的系统帐户—— SYS 和 SYSTEM。SYS 的初始口令是 CHANGE_ON_INSTALL，SYSTEM 帐户的初始口令是 MANAGER。立即改变这些口令。如果你知道用 SYS 或 SYSTEM 登录破坏了多少个数据库，你会吓坏的。

并且，如果你安装了例子表，考虑改变 scott 用户的口令，从大家都知道的口令 tiger 变为其他的口令。

7.5.7 更新系统的配置文件

最后，你需要在你的系统中的所有配置文件中注册你新的数据库。如果这是在系统中安装的第一个数据库，你要设置你的 ORACLE_SID 变量来反映这一点。在 NT 服务器中，在你设置这个值之前，需要先创建它。如果你在 UNIX 服务器上，你需要更新 oratab 以反映新的数据库信息。同时，将数据库放入 tnsnames.ora 和 listener.ora 文件中，以便通过 SQL*Net 能连接它们。

现在（假如前面的所有步骤已成功地完成），你已有一个功能齐全的 Oracle 数据库，可以开始开发、测试和运行了。

7.6 探索 Oracle 数据库

任何使用 Oracle 工作过或长或短时间的人，都面临着管理、故障排除的任务，或者工作在对他说来完全陌生的数据库环境。能够使自己快速地熟悉常规配置并建立一个外部环境是十分重要的。要确保你的时间花在处理关键性工作上，而不在耗时、琐碎的事情上浪费时间，例如，你能用哪个回滚段进行你的数据装载，哪个表空间存放你的用户数据等。

在以下部分，你将看到如何从数据库中提取关于数据库的信息。这些节中的信息与在前面部分包括的信息相结合，这将使你能够完整地勾画出一个你并不熟悉的数据库环境。

7.6.1 查看数据库

第一件事是在数据库和它的对象上，获得一个句柄。这包括查看数据文件、表空间、重做日志、回滚段和对不同缓冲区的内存分配，以及运行的后台进程。存储在工作消息队列中的数据库工作，用户和分配给用户的权限也是相关的信息。

V\$dbase 和 V\$thread 数据视图被分别用来定位数据库信息和实例信息。V\$dbase 显示

数据库名和归档模式（数据库运行在归档模式，还是非归档模式）。V\$thread视图是并行服务器特有的视图，但是当它给出实例名时，它是有用的（在 NT服务器比在 UNIX环境有更多的说明）。你也可以查询v\$instance视图来看实例的启动时间，如果你使用 Oracle 7，以下查询转换v\$instance视图启动信息为可用的格式：

```
SELECT TO_CHAR (TO_DATE (a.value, 'J')
+ b.value/86400, 'HH24:MI:SS DD-MON-RR')
start_time
FROM v$instance a, v$instance b
WHERE a.key = 'STARTUP TIME - JULIAN'
AND b.key = 'STARTUP TIME - SECONDS';
```

这个查询有类似下列的输出结果：

```
START_TIME
-----
22:48:37 06-MAR-98
```

在Oracle 7中，这个相当精心设计的查询是必须的，因为启动时间的 Julian日期部分被存放v\$instance一列中，而日期的时间部分以秒为单位存储在分开的域中。

注意 v\$thread 包含OPEN_TIME域，该域可能不同于v\$instance视图的时间显示。这是由于一个域显示实例被创建的时间，而另一个显示数据库被打开并能够进行访问的时间。

你也能通过查询v\$controlfile视图列出活动的控制文件，回滚段信息存储在dba_rollback_seg、v\$rollstat和v\$rollname中。dba_rollback_segs显示所有在数据库中的回滚段信息，而 V\$视图仅列出当前联机的回滚段。下面的查询列出联机回滚段信息：

```
SELECT a.segment_name, a.tablespace_name,
b.bytes, b.extents,
c.shrinks, c.extends, c.hwmsize
FROM dba_rollback_segs a, dba_segments b, v$rollstat c
WHERE a.segment_id = c.usn
AND a.segment_name = b.segment_name;
```

这个查询产生类似如下的输出：

```
SEGMENT_NAME      TABLESPACE_NAME
BYTES      EXTENTS
SHRINKS      EXTENDS      HWMSIZE
-----
-----
RBS01                ROLLBACK_DATA
102400                2
1                3      100352
RBS02                ROLLBACK_DATA
102400                2
0                0      100352
RBS03                ROLLBACK_DATA
102400                2
1                1      100352
SYSTEM              SYSTEM
204800                4
0                0      202752
```

重做日志信息被存储在 v\$log和v\$logfile视图中，以下查询显示出重做日志中为数据库创建的永久信息。

```
SELECT member, bytes, members, a.status
FROM v$log a, v$logfile b
```

```
WHERE a.group# = b.group#
```

```
ORDER BY member;
```

这个查询的输出与下面类似：

MEMBER	BYTES	MEMBERS	STATUS
C:\ORADATA\PPRD\REDO_PPRD_01A.LOG	204800	1	CURRENT
C:\ORADATA\PPRD\REDO_PPRD_02A.LOG	204800	1	INACTIVE

映射你的数据库文件和表空间是重要的。从 dba_data_files中查询这个信息。你能从 dba_tablespace中查询指定表空间信息和从 V\$datafile中查询指定的数据文件信息。

查询V\$sfgastat视图查看实例中的内存分配信息，以下查询可能对你有帮助：

```
SELECT name, bytes
FROM v$sfgastat
WHERE name in
('free memory', 'fixed_sga', '
db_block_buffers', 'log_buffer',
'dictionary cache', 'library cache','sql area');
```

查询一个非常小的数据库返回的结果如下：

NAME	BYTES
free memory	2287676
fixed_sga	35208
db_block_buffers	409600
log_buffer	8192
dictionary cache	226376
library cache	247636
sql area	466040

v\$bgprocess 视图能够用来快速列出正在运行的后台进程。如下查询会对你有帮助：

```
SELECT name FROM v$bgprocess WHERE paddr <> '00';
```

表dba_jobs含有通过DBMS_JOBS包按时间表未完成的作业的信息。你能使用如下查询列出时间表数据库作业：

```
SELECT log_user, job, what,
to_char(last_date, 'DD-MON-YY') l1, last_sec l2,
to_char(next_date, 'DD-MON-YY') n1, next_sec n2, failures
FROM dba_jobs
ORDER BY next_date DESC, next_sec DESC;
```

用户信息被存储在 dba_users中，授权信息存放在 dba_tab_privs、dba_sys_privs和dba_role_privs表中。你能通过查询role_tab_privs、role_sys_privs和role_role_privs来显示授给角色的特权信息。关于控制安全信息的更详细的信息可在第 23章“安全管理”中获得。

7.6.2 查看数据库段

既然你已知道数据库结构信息存放的位置，那么查看一下用户数据对象信息。你所感兴趣的数据库段包括表、索引、视图、簇和其他数据存储对象。

DBA_TABLES、DBA_INDEXES、DBA_SEGMENTS、DBA_SEQUENCES和DBA_OBJECTS数据字典视图提供了存储在数据库中各个不同数据库段中的信息。

在这些视图中的格式和信息是相当清楚的。DBA_OBJECTS是很有趣的，因为它存储数据库中所有对象创建和最后修改（时间戳）的信息。DBA_SEGMENTS是有用的，因为它保存了被每个数据库段使用的区间的大小和数量。请参看第 21章“管理数据库存储”，其中有更

为详细的说明。

7.6.3 查看各种各样的数据库对象

你还应该查看一些其他的结构，包括存储的 PL/SQL对象、触发器、同义词和数据库链接。

包、过程和函数的源存储在 sys.source\$表中。当你创建对象时，所有的空行被忽略，每行作为一个分开的记录存储在这个表中。查询 dba_source可以很容易地查出存储的包、过程和函数的文本内容。下面的查询产生指定的存储对象的源代码：

```
SELECT text
FROM dba_source
WHERE name = upper('&Object_Name')
ORDER BY line;
```

有关触发器的信息存储在 dba_triggers表中。因为触发器体是作为一个长整型存储的，所以当你用 SQL*Plus 查询这一列时，你需要设置 longsize 为足够长的大小。在这一视图上，TRIGGER_TYPE 列既可以是行，又可以是语句，TRIGGERING_EVENT 存储触发器的类型——before insert、after update 等等。

dba_synonyms 列出数据库中的同义词信息。注意公共同义词，用 PUBLIC 作为拥有者。查询 dba_db_links 获得在数据库中定义的数据库链接的信息。关于 dba_db_links 有一个警告：如果这个链接是用连接的用户名和口令定义的，则口令被存放为明文。

7.7 研究不熟悉的环境

现在把包含在本章中的一些信息放入一个熟悉的参考框架中。假如你是一个签约的 DBA，刚到达一个新场所，前一个 DBA 已离开现场，而且他没有留下任何关于他的系统结构的支持文献，该怎么办？

以下两节探讨可能采取的行动——一个是在 UNIX 服务器上，另外一个在 NT 服务器上。它们应该会给你一些思路，你还要求助于本章前面所描述的信息。

7.7.1 研究UNIX环境

你主要的兴趣是判断这个 Oracle 软件是如何安装和配置的和在系统中安装了什么数据库。首先，获得用于安装 Oracle 软件的操作系统帐户的访问权。接下来，你需要定位可能的 ORACLE_HOME 目录。如果在系统中仅有一个 ORACLE_HOME，它应该已经放置在你的登录序列中。从你的环境变量中检查 ORACLE_HOME 的值，同时查找和检查 oratab 文件以确定其中是否包含数据库和 ORACLE_HOME 中的信息。这不是一个确保完全的清单，有可能 oratab 文件被手动更新，最新的信息不是数据库操作所要求的。另一个选择是搜索 Oracle 服务器可执行文件（oracle）或 rdbms 和 dbs 目录。在所有的 Oracle 服务器安装中，这是标准的。对所有可能的 Oracle 目录作一个详细清单。

接下来，判断服务器中有什么数据库。在每一个你所见的 Oracle 主目录中，检查 dbs 目录。在这个目录中，所有配置的数据库应该有一个 initSID.ora 文件。这个文件可能被链接到不同的管理目录，但是这些文件的列表显示所有已配置的数据库。再次声明，这并不一定意味着所有这些数据库都存在。创建和删除 init.ora 文件，绝大多数是由 DBA 手工进行的。并且不保证

init.ora文件在dbs目录中，由于dbs目录是Oracle程序缺省查找的地方，所以你很可能在那里找到它。

检查你的用户和系统登录的脚本来判断是否 Oracle环境已经进行过自定义修改。查看coraenv或oraenv是否正在被登录脚本调用和是否这些文件已经发生了其他方式的修改。如果coraenv或oraenv没有被调用，判断是否 Oracle环境变量正在被设置，检查系统的启动脚本（rc文件和与此类似的）来看什么程序或后台进程用系统根用户自动启动。不要忘记检查crontab文件或系统的时间表，以便Oracle拥有者和系统管理判断什么是按时间表规划并自动运行的作业。最后，检查UNIX内核参数来查看Oracle安装如何配置内存和内核。设置这些参数的具体位置和方法随UNIX厂商而有所不同，查看你的操作系统文献以获得更详细信息。

提示 使用UNIX的find命令来自动定位文件，否则需要手工查找。查找你的UNIX系统的联机手册（man）获得正确的语法。

7.7.2 研究NT环境

在NT服务器上，理解Oracle环境的特定配置要比UNIX服务器上直接一些。这是因为只有很少方面的配置要让DBA去费心完成。你的第一步是翻译环境变量，查看 Oracle注册表。使用regedit查看HKEY_LOCAL_MACHINE\Software\Oracle键，这个字符串值告诉你ORACLE_HOME的位置和其他的有用信息。接下来，在Control Panel中打开Services applet，查看所有Oracle服务。在系统中每个数据库都有一个Oracle服务，同时每个数据库也可能有一个Oracle启动服务。同时查看其他被安装的服务——TNS监听器、Web服务器，等等。

所有相关的配置值被存储在 Oracle注册表键中。数据库的参数文件在 \$ORACLE_HOME/database目录下，这个目录也是数据库数据文件存储的缺省目录。使用 Windows NT资源管理器帮你定位文件。一定要检查 NT任务管理器和作业时间表来查看可能运行或已经在运行的作业。

第8章 Oracle数据库系统管理概述

本章要点：

什么是数据库的生命周期

DBA是什么

建立一个数据库的系统管理步骤

本章将讨论数据库系统管理员（DBA）是什么和做什么，数据库生命周期由哪些部分组成，创建 Oracle 数据库时 DBA 应遵照执行的步骤和 Oracle DBA 的维护职责。正如能从 RDBMS 映射到 Oracle 的 RDBMS，这一章的许多内容也可以应用到 DBA 支持的几乎任何其他的 RDBMS 中。这是由于包括管理在内的数据库技术和理论能够适用于所有 RDBMS。正如我们在第 4 章“Oracle 的解决方案”中提到的，包括 Oracle 在内的 RDBMS 都有不同方面的缺陷并在很大程度上都忠实于他们对关系模型的特定实现方法。

8.1 什么是数据库的生命周期

软件的生命周期是一种描述如何创建和进化软件的方法。一个经常用来描述一个软件的生命周期的谚语是“从摇篮到坟墓”，其意思是从开始到结束。包括数据库在内，所有软件的生命周期都要经过一系列阶段。这些阶段是分开的，它们使软件开发过程朝着更深层次方向发展，换言之，它们描述了生命周期的发展。这些过程是系列化的和相互依赖的。例如通常说，如果阶段 A 先于 B，那么阶段 B 依赖于 A，更进一步，B 可以对 A 提供反馈，循环进行。要想简单地表示先于和依赖于，你可以使用 A→B。要增加反馈表示，你仅需要画一条从 B 返回到 A 的弧线。

有许多模型提供各种不同的方法来表示（图形化或其他的）软件的生命周期。最典型的软件生命周期模型是瀑布模型。还有其他模型存在，例如螺旋模型。首先，让我们看一个最基本的软件生命周期的描述。它是一个简单的瀑布模型。

分析→设计→实现→维护

这个模型是所有软件开发的基线。现在简要地概括描述每个阶段。

8.1.1 分析阶段

分析阶段代表用户需求的全面收集和文档编制，另外分析这些需求的操作意义是什么，因此通常称分析阶段为需求分析阶段。根据需求的不同，客户提供不同层次的合作。有些提供十分详细的技术工作的描述，而另一些可能只简单地用言辞表述。尽管通常焦点放在略述要创建的系统的功能性要求，它可能包括也可能不包括处理诸如用户接口或性能考虑这样的项目的详细说明。详细说明书简单地说是需求的另一种说法，当一个程序员想开始设计他的程序时，他必须查看“说明书”。

8.1.2 设计阶段

设计阶段主要包括算法设计，也包括输入 / 输出考虑；如果是交互式的程序，还有格式化和显示考虑；或者如果程序是用于科学方面的，还有精度问题。无设计的编码等同于无计划的长途旅行。任何情况下，在设计阶段所做的许多重要选择不仅影响功能和显示，而且还会影响纯粹的执行。毫无疑问，数据结构和算法的正确选择会导致或打断该程序的后续执行。

8.1.3 实现阶段

实现阶段主要是程序或系统的编码。在你有了正确选择的数据结构、算法和其他类似的构建块后，你能够用所选择的语言开始编程。在编程本身，也有许多选择要做。例如，哪种控制流能最好地实现一个特定算法？另外，在这个阶段，你也能够进行隐含或明确选择，如对存储的速度（反之亦然）进行选择。这时，软件开发实际上经常受到理论上的约束。例如，你或许必须使用某种类型的程序接口，不是因为它们在某种程度上更好或者更快，而是由于它们被用一种或另一种方法标准化了。当然，实现还应该说包括其他阶段，通常所指的是测试和展开。

1. 测试阶段

测试阶段正如你能想到的一样。例如，在测试阶段，你可能首先测试系统的所有独立组件（单元测试），然后把它们放在一起进行端到端的测试（开始到完成），这也是众所周知的整体或系统测试。你做一下改变，然后返回（回归），重新测试每一项（或者至少测试可能受影响的每一项），比较输出结果，确保功能没有丢失。其他测试方法是装载测试和执行测试。装载测试一个系统时，开始用所希望的最少用户数运行测试，然后增加用户数接近你希望的最大的用户数。比较每一个运行结果，看发生了什么影响。如果有，在你的系统执行中增强并发性。另一种装载测试方法是增加通过你的系统的数据流量，这正好与增加访问用户数相反。在多种情况下，两种情况你都能见到。

在执行测试时，你要确立你的执行目标是什么。在很多种情况下，如果客户没有明白地表达过执行目标，只需简单地询问客户。通常，你考虑三个执行措施或尺度中的一个：经过时间、反应时间和吞吐量。经过时间，也称为分界时间或时钟时间，它是进程持续了多长时间，用结束时间减去开始的时间，这对批处理系统是有用的；反应时间是一个进程对一个请求做出响应所花的时间，因此，这个措施对交互式系统（例如 DSS 系统）是有益的；吞吐量是一个工作单位时间内通过系统的块数，通常是业务定义的“事务”的度量，例如，每秒钟的事务数是吞吐量的一种度量方式，这种度量方式对 OLTP 系统是有益的。执行测试可以被定义为功能需求的一部分。

2. 展开阶段

展开阶段发生在系统通过它的所有测试阶段之后，并准备在它的所有终端系统、客户和服务器展开或安装时。安装所有支持的硬件和软件过程，当然也包括应用软件自身的运行形式都有关于展开的问题。根据合约声明，一旦每件事情被展开，它可能要进行简单的再测试，然后经过一些事务阶段，有效地“移交”给负责的用户（如果在合同中该项目是外协的，有可能要提供一些现场的应用软件支持）。展开完成后，产品进入维护阶段。

8.1.4 维护阶段

维护阶段是许多软件的生命周期中最长的部分，估计许多软件在维护阶段的时间占 70% 或更多。因此，仅就此而言，这可能是最重要的阶段，因为在软件开发中的错误决定能够削减软件某方面的正确功能。这里的决定包括许多与“开放”系统有关的问题，例如，可移植性和互操作能力，它会可能影响软件长期的生活中的一部分。维护也涉及许多系统集成问题，例如替换过时的现有硬件和软件组件，由于各种因素，随时间删除某些其他组件，并引入一些新的组件。由于各种原因，正如你想象的一样，涉及的部分越多，集成可能变得越复杂，因为涉及到许多部分之间的相互作用。在执行阶段，当预测什么是想要的行为时，标准化的接口变得更加重要。关于系统集成，有条原则“移动的部分越多，就越有可能发生问题（和失败）”。集成问题背后的原因之一是异质性，也就是，如果许多或所有不同组件来自于不同厂商，无缝集成（也就是，没有任何不兼容性）的概率是很小的。

现在你了解了通常软件开发的整个生命周期，考虑一个数据库的特定生命周期。相对于一般的软件模型，数据库特定的活动是什么？以下是一些解答：

- 分析 —— 概念模型（或业务模型）开始成形。
- 设计 —— 概念模型被限定。逻辑设计包括将概念模型初步映射为关系模型和实体关系图（ERD）。执行规范化。然后规划物理设计。同样地，只要逻辑设计一完成，数据库的前端和后端程序设计便能够开始。
- 实现 —— 对一个数据库系统，这个阶段是通常所说的建立，就是用数据库语言（DDL）建立数据库结构；如果初始装载是适当的（与超时装载相对），装载就是用数据库操纵语言（DML）或厂商的应用工具最初将数据装载到数据库中。同时，对一个数据库系统，程序的编码能够从设计阶段开始或连续。
- 测试 —— 在数据库创建和装载之后，程序被写入，全部进行测试，进行单元和整体测试。
- 展开 —— 在许多现代系统中，展开意思是在 PC（客户）和服务器（Web服务器和数据库服务器）上安装软件。
- 维护 —— 这与其他软件系统没有任何不同，除了它是更多地属于以数据为中心的软件而不是以进程为中心的软件。

8.2 DBA是什么

我的学生经常请求我描述当前市场上所有可获得的数据库职业有什么不同。当然，DBA 是数据库人员中处于中心位置的，所有其人员都围绕着他。在考虑 DBA 是什么之前，首先考虑一般能够在市场上找到的所有与数据库相关的职位：

- 设计者。
- 建筑师。
- 工程师。
- 开发者。
- 程序员。
- 操作员。
- 数据库系统管理员（DBA）。

首先考虑设计者。设计者收集初始用户的需求（如果还没有收集）。然后他开始分析、精炼和推断，使其适合标准化数据库的模型输入，例如使它们符合关系模型。最初他与用户讨论，随后分析帮助形成典型的关系模型，这些在第一部分“数据库管理原理”和第2章“逻辑数据库的设计和标准化”中有一些描述。概念模型也可能看作业务模型。它不仅包括已有的业务数据模型，而且还有业务规则，这将在后面作为在任意给定的 RDBMS实现中的约束、过程和触发器来明确它们。正如你猜想的，设计者的责任范围跨越需求分析、逻辑设计和物理设计阶段。

建筑师，有时也称数据库系统设计师或信息系统设计师。他对在设计和工程中的所有硬件、软件和包含在企业中的网络部件承担很大的责任。因此建筑师部分是设计者，部分是工程师，在他的职业中可能有一些或全部与其他所有数据库职位相关的职务。建筑师通常不仅是在数据库技术领域，而且也在信息技术领域有多年的工作经验（例如 10~15年或更多）。

与建筑师相比，工程师明显地有更窄范围的责任，只注重于数据库技术或仅是一个组织中的一个部门或一个组织分枝。他的范围可能在他的技术与组织范围内更窄。但是，工程师一般是数据库系统管理员或有多年工作经验的开发者（例如 5~10年或更长）并有足够的经验，并理解与数据库相关的技术和可能更多的外围知识。

开发者，有时与程序员意思相同，但能够做一个区分。数据库开发者一般用前端程序工作，前端程序提供交互式输入（例如 Oracle Forms）和输出（例如 Oracle Reports）。因此，数据库开发的交互特性使许多开发者用 Microsoft Visual Basic、Microsoft Access、Sybase(PowerSoft) PowerBuilder和Borland Delphi这样的程序工作。所有这些程序都是关于建立图形用户界面（GUI）的，例如，那些典型地遵循 Microsoft或Web浏览器的界面设计。这些基于Microsoft Windows的前端，有时不严格地称为在客户/服务器模式中的客户应用，或者甚至有时不严格地简称为客户/服务器。那些基于 Web浏览器和Web技术的前端被简单称为基于Web。不幸的是，这种不严格的分类是误导的，因为基于 Web的系统是简单的客户/服务器系统的特例。然而，开发者负责建立前端、GUI-数据库、输入/输出程序。但是他们生产的实际程序代码量是很少的，因为几乎所有的代码都由前端工具自己生成。当开发者必须写实际代码时，他们通常也只仅需写相对高级的语言，如第4代语言（4GL），包括宏和脚本语言。许多这些工具既基于 ODBC技术，又可以基于本地厂商的 RDBMS协议，例如Oracle SEL*Net或 Sybase Open Client(DB_Lib和Net_Lib)。

与开发者相比，数据库的程序员实际上编写很多代码，经常使用的是 SQL和基于厂商的、控制流的、基于SQL的第4代语言，如Oracle PL/SQL和SQL server Transact-SQL（T-SQL）。但是你将经常见到数据库的程序员编写嵌入式 SQL或本地应用程序接口（API），如Oracle Call Interface（OCI）或Sybase Open Client。正因为这样，当然你将发现这些数据库程序员经常使用C、C++和Java。因此他们与使用COBOL、FORTRAN访问文件系统而不是数据库的前辈没有太多的不同。实际上，在许多方面，这些程序员是同一个人或者是他们更新了他们的技术。

操作员可能被认为是支持操作（也就是所谓的维护和操作，或 M&O）的技术人员。他的明确职责是负责定期维护确定的硬件和软件。你将发现操作员一天 24小时一周7天轮流值班。操作员典型的职责是负责磁带（和光盘）归档，包括数据库归档。随着现代的磁带（和光盘）自动设备（接卡箱、自动光盘机等）的使用，操作员在这个舞台上一点儿也没有变得不重要，

但可能会减少一些工作量，因为这些设备很少需要人关心和进行输入。操作员除了履行第三位的存储职责（如“悬挂磁带”），还可能履行“悬挂电缆”（物理安装网络电缆）的角色，在计算机房布局电源分布电缆，连接第二存储设备（硬盘），建立支持多计算机机架，甚至于建立一些其他设备。

最后是数据库管理员（DBA）。他是数据库的中心职位。通常看作是级别在程序员之上，工程师之下。DBA经常发展成为工程师，或者甚至是建筑师，而且他经常已是开发者、程序员或操作员DBA和操作系统的系统管理员（SA）负责维护和执行任务。同时，DBA必须与开发者和程序员紧密工作，以建立一个完全的、集成良好的数据库系统；其中，DBA将精力集中在工作在后端，开发者/程序员将精力集中在前端。另外，DBA既可已经是或可能将成为设计者。在任何一种情况下，DBA必须与设计者特别紧密地合作，在设计到建立（实现）之间建立联系的桥梁。

在现实世界里，你将经常发现有智慧的人扮演重要的角色。例如，一个人可能是设计者、开发者/程序员和DBA。在某些情况下，你将发现一个人同时作为DBA和SA。在有些组织中，以上所有的职业是分开的和有明确区分的工作位置。一个人担任多种角色可能由于种种因素，包括一个人技术才能的广度和深度，或者公司不愿意分得这么细或者缺乏对不同角色的需要的远见卓识。另外，有时一个公司既不能负担与公司规模相对应的全部位置，又缺乏对数据库重要性的认识。同时日益增长更多的情况是公司不能雇用和补充全部的多个位置，因为相对于需求而言，仍然缺少与数据库相关的人员。这种倾向不仅使单个位置的潜在薪水提高，而且还有鼓励有能力履行职责的工作人员担任多个角色。

现在看一看与通常的DBA相比，Oracle DBA有什么特殊性。

8.3 建立一个数据库的系统管理步骤

对于Oracle DBA，在创建Oracle数据库时有几个“管理步骤”，它们中的许多只需少量的修改就能应用于任何其他厂商的数据库中。有很长一段时间，我有很多工作在不同背景的数据库（包括通常任何类型的数据库和特定的用Oracle数据库）的同事和学生，询问在创建数据库时DBA应遵循的主要步骤是什么。这根本不是一个不合理的问题，因为事务的正确顺序经常被掩盖了。更常见的是，有许多参考和资料用低层次的说明掩盖了高级顺序。例如你找到了如何编辑和改变在init.ora文件中参数，但是怎么适合“大图形”？要获得全部的顺序和事件，下列清单足够详细，但还不够深入：

- 1) 安装Oracle软件（RDBMS、网络、客户等等）。
- 2) 编辑初始化参数文件（init.ora）。
- 3) 创建Oracle实例。
- 4) 创建Oracle数据库，包括系统表空间和重做日志。
- 5) 创建数据字典（也就是运行cat脚本）。
- 6) 创建其余的表空间和回滚段。
- 7) 创建表。
- 8) 添加主键和唯一键（未使能）。
- 9) 使能主键和唯一键（创建索引）。
- 10) 增加外键（已使能）。

- 11) 创建外键索引和为其他适当的列创建索引。
- 12) 增加检查约束和写适当的触发器。
- 13) 编写适当的过程、函数和其他代码（也就是 Pro*C）。

尽管这些步骤中有些能够更进一步分割（或者合并），以我的工作经验，在创建数据库中，这种DBA（和部分的开发者）职责的分割方法是有很高的灵活性和可伸缩性。

1) 当安装Oracle软件时，不仅必须考虑你当前的需要和数据库的初始状态，而且还要考虑将来的需要和预期的增长。使用优化弹性体系结构（OFA）将是十分有益的。优化弹性体系结构的更进一步信息（包括目录和文件的创建），请参见第7章“探讨Oracle环境”和附录A“在Solaris系统上的Oracle”，它们都包含有实现优化弹性体系结构数据库的概述。

2) 当然，除非你安装和希望使用缺省的 Oracle实例和数据库（ORCL），你需要创建自己的数据库。要做这一工作，你首先必须编辑初始化参数文件（init.ora或init<SID>.ora）。你至少应指定db_name和control_files参数，但可能还有其他一些参数，如 db_block_size尺寸。最后，当你创建了你的回滚段并想在启动时作为不变的基础用 rollback_segs打开它，你还必须重新编辑init.ora文件。另外，还有很多其他你可能调整或使能的事情，如归档日志模式。

3) 然后，创建一个实例，首先创建包含控制文件声明的已编辑的 init.ora文件。在UNIX系统中，当你在svrmgrl中启动为非安装状态时，你已“创建”了你的实例。在UNIX系统中，能够用命名为dba的UNIX组连接internal。在NT系统中，你将使用 Oracle NT实例管理器（orsdim80.exe）的命令行方式或图形用户接口方式，创建你的实例。这也将创建口令文件和相关服务（用启动文件）。当你创建了NT实例时，你为internal指定了一个口令，以使你能在svrmgr30.exe中连接internal。参看第7章“探讨Oracle环境”和附录B“Windows NT上的Oracle”，其中有关于NT中的Oracle的更详细信息；参看第7章和附录A，其中有关于UNIX中的Oracle的更详细信息。

4) 接下来的任务是在非安装状态下创建你的数据库，当你编写创建数据库脚本时，至少应指定创建系统表空间和初始的重做日志，但是你还应该利用这个机会设置诸如MAXDATAFILES和MAXLOGFILES这样的参数为你希望的值。一个初始的系统回滚段被创建，该回滚段以后将被删除。在数据库创建时，读取你的 init.ora参数，在指定位置被创建你的控制文件。

5) 你将运行 Oracle提供的通常为任意 Oracle数据库运行的基本包：catalog.sql、catproc.sql、pupbld.sql；可能还要运行其他的一些包，如 utlexcpt.sql、utlxpln.sql或catrep.sql，这取决于你的应用。这实质上是在 Oracle的数据字典（X\$表）中建立你的系统管理访问和视图（对SYS、SYSTEM和其他用户）。创建V\$视图、DBA_表和USER_表。

6) 然后你能创建余下的用户表空间，它将服务于你的交互用户和应用的需要。在这时，你将指定你的表（和索引）正确的尺寸并应该能够根据你的估计提供合理的表空间大小。参看第4章“Oracle的解决方案”。如果你想使系统监控（SMON）能够自动合并表空间中的碎片，记住设置PCTINCREASE为1。另外，在这里你将需要创建你希望的用户和应用回滚段。等做完这些，要记住返回去重新编辑 init.ora中的rollback_segs参数。

7) 现在再使用你的尺寸输入，准备好创建你的空表。你的考虑应该包括覆盖表空间的存储参数和考虑什么数据类型最适合存放你的数据。例如，使用 VLDB，逻辑压缩可能十分重要。

记住，一定将具有大量的连结和其他有同时大量访问的表分开，不仅分开到不同的表空间，而且最好是分开到在不同的磁盘上。参看第3章“物理数据库的设计、硬件和相关问题”和第4章“Oracle的解决方案”、第16章“性能调整基础”和第19章“调整输入/输出”，以得到更详细的关于I/O分配和优化的指导。

8) 在Oracle中记住增加使能（缺省值）的主键将在拥有者的缺省表空间中自动地创建索引，这通常总不是一个好办法。因此，应定义你的所有全部主键初始无效。主键所保持的为真，对唯一键也同样保持为真。

9) 使用特定的物理参数分别使能所有你的主键，这保持了逻辑定义与物理布局分开。这总是一个好办法，特别是对VLDB和关心可移植性的软件。同样，唯一键也能照同样方法处理。

10) 在Oracle中，使外键使能（缺省值）不会在任何地方自动地为它们创建索引。了解这一点是很重要的，因为当大表被连接时，外键可能会成为性能损害。

11) 在大表中为希望连接的外键创建索引，因为没有它们，父表被行级锁定。使用它们，就可像普通表一样，既可有行级锁定，也可没有锁定要求。另外，索引大型表中那些可能用SELECT的列列表部分或用WHERE语句进行大量访问的列。

12) 附加的约束和触发器使关系数据库实现数据一致性。没有它们，除了主键和外键约束外，你实际只是拥有了另一个数据库。用这些附加的结构，你能增加业务规则，如强迫域完整性（如一系列的有效值范围）和数据流的依赖性（例如，向数据库中插入一个雇员时，首先将他的识别信息插入到一个表中，最后将他的其他信息插入到定义雇员的其他表中）。记住，当约束使能时，对每一行的每一次访问都进行检查，这是要付出很大的代价的。为识别任何对约束的违反（例外），你一般应在任何大块的装入或大的修改之后启用它们，然后禁用它们。定义有触发器的表进行修改时触发器被触发，然后它能够在任何其他表上操作（或简单地做一些计算或文件处理），但是它们不能在定义它们自身的表上操作，与检查约束正好相反，检查约束仅能在定义它们的表上操作（“检查”列）。约束是简单的数据库定义语言（DDL）语句，触发器通过数据库定义语言（DDL）定义，但它从本质上讲是一个过程体（DML）。

13) 过程、函数、游标和嵌入式SQL做约束和触发器不能做的工作。例如过程和函数不依赖于单个表的修改事件，它们能够独立于任何给定表的修改运行。当基于集的访问在某种程度不足够（有时是这样）时，游标使过程、函数或嵌入式SQL内部的实时记录访问成为可能。嵌入式SQL提供紧密集成，特别是使用游标与使用第三代语言，如C。借助于所有这些程序工具，一个Oracle DBA或开发者/程序员能够完成几乎任何相关数据库的编程任务。正如你预料的一样，这里正是DBA离开和开发者/程序员进入的地方，尽管在理论上，早在逻辑设计完成时，他们可能已经开始设计和编写程序代码了（他们只是还没有编译它，因为数据库必须被创建）。

对大系统，这些步骤的伸缩性是，除了前三步，每一个剩下的步骤可能符合一个或多个SQL*Plus脚本。例如，你可能具有表8-1中的脚本。

为什么要完全分开帮助？它在多层次上进行帮助。首先，它系统化组织事情。例如，当你创建你的数据库时每次创建一步，如果某事失败，你能精确知道失败发生的地方。经常，你仅需要修复你所做的并简单地重新运行那一步，恰好与完全删除并返回到很大的“创建每一件事”脚本的起始点相反。因此，这些步骤帮助你模块化你的数据库创建。这样做有很多

的原因，至少不仅仅是查找在创建中出现的故障，更进一步，这些步骤倾向于从物理层中分离出逻辑定义。这意味着，当对一个或另一个（通常很可能是物理的）进行改变时，你只需修改一个脚本，不是对每个逻辑 - 物理对中的两个进行修改，如表 - 表空间、addpk-enablepk等等。另外，正如你想要的，如果总的应用是足够大的，当然这些脚本能够更进一步分为不同的区域组。最后一个动机是考虑在另一个平台上使用不同的物理层导入，并重建这个数据库。这些步骤和脚本肯定是会有帮助的（如移植），甚至比导入和导出对象更有帮助。

表8-1 伸缩性建立脚本

步骤	脚 本	描 述
(4)	crdb.sql	创建数据库
(5)	runcat.sql	运行cat脚本（创建 catalogs）
(6)	tspaces.sql	创建表空间
(6)	rollback.sql	创建回滚段
(7)	tables.sql	创建表
(8)	aadpk.sql	增加主键(无效的)
(8)	adduk.sql	增加唯一键（无效的）
(9)	enablepk.sql	使能主键(和创建它们的索引)
(9)	enableuk.sql	使能唯一键(和创建它们的索引)
(10)	addfk.sql	增加外键（使能的）
(11)	crfk.sql	创建外键索引
(12)	addck.sql	增加检查约束
(12)	trig.sql	创建触发器
(13)	procs.sql	创建过程
(13)	funcs.sql	创建函数

最后需要注意的一点：你将在哪里放置一个大块的载入操作（在哪些步骤之间）？用来装入数据的一个正常的地方（如以 SQL*Loader为例），将正好是在创建你的表之后，但在你创建键值、索引和约束之前，也就是在第七步之后，但在第八步之前。为什么不在第八步之后装入？因为有使能键，索引和约束可能使装入数据操作速度相当慢。数据一致性怎么样？在你使能你的键和约束的同时，将发现那时有一些或所有的完整性和约束违背。然后你能修补数据，根据需要增加或全部重新安装。用这种方式，既不牺牲速度又不牺牲完整性。

8.4 小结

你已经了解了一般的软件开发周期和一个数据库特定的软件生命周期怎样映射到常用模型。你看到没有太多的不同；但是，从 DBA的观点看，又有些不同。例如，在设计时，你的焦点更多地数据模型上而不是在进程上，尽管后者也是同等重要的，特别是对开发者和程序员而言。在实现时，你的焦点更多地集中在建立数据库结构上，正好与编写代码相反，后者是开发者的工作，也是程序员工作的难题。

你还要考虑有哪些主要的数据库职业和它们之间的不同，不仅在技能水平和经验上，还有相对于他们的同事，将他们安置在数据库生命周期的哪里更合适。正如你所见，许多这些职业经过一段时间，可能导致从一个向另一个过渡。因此，在数据库领域有相当大的职业增长空间。

接下来，足够详细地理解创建数据库的生命周期阶段，主要集中在灵活性和可伸缩能力来考虑 Oracle DBA 必须做什么。书本和其他资源经常给合适的事件顺序披上了外衣，使你只

见树木，不见森林。本章尽力明确说明这些情况，枚举 Oracle DBA 要遵循的主要创建步骤并强调他们重用能力，如何使它们松散、分离以及将它们写成脚本。对更多深层次 DBA 职责的概括，请参见第五部分“管理 Oracle 数据库”和第六部分“Oracle 接口和应用工具”。在第五部分包含的章节帮你管理存储、资源使用、安全性、备份 / 恢复和一致性。另外，在第六部分的章节中，包括能更有效地管理你的 Oracle 数据库的基本 Oracle 工具，包括 SQL*Plus、PL/SQL 和 Oracle Enterprise Manager (OEM)，最后一个应用工具是一个 GUI 工具，它将为 DBA 提供更多的管理能力，等效于许多 SQL*Plus 脚本或 PL/SQL 程序的工作。

第三部分 Oracle 8.x RDBMS

第9章 从Oracle 7.x到Oracle 8.x的移植

本章要点：

三种主要方法

移植工具

常见问题

移植之后的任务

这一章将讨论从Oracle 7.x数据库到Oracle 8.x数据库移植的三种主要方法。重点将集中在移植工具方法上，讨论伴随任何移植方法所必须的移植前的任务，并考虑常见问题和简要了解一些可能需要的移植后的任务。

9.1 三种主要方法

从Oracle 7数据库移植到Oracle数据库的三种主要方法如下：

- 移植工具 —— Oracle 8提供能够使用GUI或命令行模式的移植工具，用于帮助将 Oracle 7（7.1.6或更高版本）内建的数据库移植到 Oracle 8数据库中。
- 导入/导出 —— 这是与使用所有的 Oracle二进制导出和导入格式完成通常（同种版本到同种版本）整个数据库备份和恢复完全相同的技术。
- COPY或CTAS —— 这是一种用于拷贝整个数据库的方法，通过 SQL*Net（Net 8）的 COPY命令或者经过DDL Create Table As Select（CTAS）命令，从一个Oracle 7数据库到一个Oracle 8数据库拷贝整个数据库，每次仅能拷贝一个表。

表9-1概括列出每种方法的优点和缺点。

表9-1 移植方法的优点和缺点

方 法	优 点	缺 点
移植工具（内建移植）	由Oracle提供和支持。几乎是自动的。 速度相对快。空间相对有效地利用	源Oracle 7数据库必须是7.1.6或更高版本。仅能全部移植。移植是破坏性的（对源Oracle 7数据库）
导出/导入	任何Oracle 7源数据库。可以部分移植。 有利于消除碎片。移植无破坏性	速度相对慢，全库导出经常受操作系统和分区限制。除非直接写到磁带，否则在磁盘上要求有足够空间存放一个数据库完全导出文件
COPY或CTAS	任何Oracle 7源数据库。可以部分移植。 有利于消除碎片。移植无破坏性	速度相对慢，需要明显的手工操作。 源和目标数据库都必须是打开的

现在简要地看一看这些优点和缺点的比较。在移植之前或移植进行过程中，新的 Oracle 8基本表和V\$视图被增加，另外有些 Oracle 7基本表和V\$视图被修改。更新数据文件头的结构，

包括控制文件和重做日志文件的文件头结构。任何方法必须在 Oracle 7数据库转换到 Oracle 8数据库之前或其间完成这项工作。

现在讨论许多 Oracle 7数据库系统管理员应该已经熟悉的两种方法：导出 /导入和 COPY/CTAS。用导出/导入方法，当 Oracle 8导入工具从 Oracle 7导出文件导入时，所需要的 Oracle 7到Oracle 8的转化自动发生。使用导出/导入方法的最大好处可能是使用工具管理的方便性，而最大缺点是对整个数据库导出文件需要大的空间。然而用导出 /导入方法“逐个地”移植也是可能的，尽管每次你仅使用小的空间，但将需要花很长时间来完成移植整个数据库的工作。然而，在多数情况下的讨论几乎不关心移植速度的快慢。相反，在多数安装中最主要关心的是进行成功的移植。

COPY/CTUS方法对于Oracle 7的数据库管理员与同导出/导入一样熟悉。然而这种方法很少作为主要的备份和恢复手段使用，特别是对 VLDB。它经常被用户用作标准备份（如导出/导入或 RMAN）之外的备份方法。这种方法最大的优点是操作简单，而这种方法最大的缺点是移植要花很长的时间。发生后面一种情况是因为表必须被拷贝，不管表有多少，每次只能拷贝一个表。在所有的拷贝完成之前，两个数据库必须保持运行状态。另外，与导出/导入一样，移植也可分阶段进行，但这只是更进一步加长了移植时间。与导出/导入相反，在COPY/CTAS情况下，移植所花的时间是最重要的，因为它与会潜在地影响正常操作的时间。要注意的另一方面是，INSERT INTO.....SELECT能够替代CREATE TABLE AS SELECT，唯一不同是目标表已经被创建。

稍微讨论一下第四种存在的方法：卸载 /装载方法。用这种方法卸出所有的 Oracle 7表数据到一个大的平面（ASCII）文件，然后将它们用 SQL*Loader装入到你新的 Oracle 8数据库中。这种方法的好处是像导出一样，在被重新装入之前，能够卸载脱机数据。因此，它是固有的异步模式，然而，它也是固有的最耗时的手工操作。除了最小的和最简单的数据库之外，我认为对所有其他的数据库而言，这种方法不是真正可行的方法。

9.2 移植工具

因为移植工具（mig和mig80）对Oracle 8是新工具，它特别对大的数据库在空间和时间的有效性方面可能提供全面的解决。本节将讲述它。

9.2.1 高层次的步骤

使用移植工具的高层次步骤概括如下：

- 1) 安装移植工具。
- 2) 对移植的准备（参看9.2.3节“移植前的任务”）。
- 3) 安装Oracle 8软件。
- 4) 运行移植工具。
- 5) 准备将转换的Oracle 7数据库。
- 6) 从Oracle 7数据库到Oracle 8的转换。
- 7) 完成最后的整理工作（参见9.4节“移植之后的任务”）。

9.2.2 低层次的任务

接下来你将花一些时间来了解低层次的任务，在使用移植工具方法时必须遵循这些步骤，

查看每一步骤以获得详细解释。

1) 运行Oracle安装程序 (orainst)。选择 “Install Migration Utility O7->O8”。这个工具被拷贝到Oracle7的\$ORACLE_HOME/bin子目录下。它也将相关的信息文件——一个新版本的migrate.bsq文件安装到\$ORACLE_HOME/dbs子目录，并安装所需要的NLS文件。

2) 按照下一节“移植前的任务”中的提纲，概念化和策略化地准备移植。明智地执行移植所要做的唯一最重要的事情是在继续进行移植之前，确保获得一个成功的、完全的 Oracle 7 数据库备份。如果可能，用两种不同方式获得两个完全备份，例如一个完全冷备份，另一个完全导出。关闭实例。

3) 运行Oracle安装程序 (orainst)，选择Install/Upgrade安装Oracle 8软件。这后一种选择阻止安装程序创建新的实例和数据库。

4) 从任务2往下实例应该始终关闭。运行任务1在Oracle 7的\$ORACLE_HOME/bin子目录下安装的移植工具。这个可执行文件的命名在UNIX下是mig，在NT下是mig80。在运行mig之前，要知道它的主要参数。

- check_only —— 如果是FALSE (缺省值)，计算使用的空间，发生实际移植。如果是TRUE，仅计算使用的空间。它与用主要导出来决定是否有足够的空间用来移植数据库 (源和目标，在这种情况下) 的noexec参数相同。
- no_space_check —— 当它被置为TRUE时，不执行空间检查，check_only被忽略。如果为FALSE (缺省值)，check_only起作用。
- dbname —— 被移植的旧的源数据库名 (在init.ora中的db_name)。
- new_dbname —— 新的目标数据库名 (在新的init.ora中的db_name)，可以和旧的名相同；缺省值是DEFAULT。
- pfile —— 旧的源数据库的init.ora文件名。
- spool —— 日志输出的相对或绝对路径的文件名，以双引号括起。

警告 因为Oracle 8数据字典要比Oracle 7大50%，所以手动增加数据文件到SYSTEM表空间，或至少设置这些数据文件已经支持的自动区间扩展 (假设在操作系统级还有空间)，这是十分重要的。另一个选择是手动重新调整或自动扩展 (AUTOEXTEND) SYSTEM表空间已经存在的数据文件。

在操作系统命令行提示符下运行 mig可执行文件，并在其中配以参数列表，如 <参数1>=<值1> <参数2>=<值2>等等。在NT操作系统在MS-DOS提示符下运行mig80；在UNIX操作系统中，在shell提示符下运行mig命令。假如你的Oracle bin目录是在你系统启动的映象路径下，一个可能的例子如下：

```
C:\>MIG80 DBNAME=MYDB NEW_DBNAME=MYNEWDB SPOOL=MYLOG
```

移植工具将运行ASCII码脚本文件migrate.bsq，它接下来创建一个移植用户，在这个移植用户中建立视图以支持移植过程。当这样做之后，移植工具将在特定位置 (或缺省) 的路径下留下一个日志文件 (从SPOOL选项)，在Oracle 7 ORACLE_HOME/dbs子目录下留下一个二进制转换数据文件 (conv<SID>.dbf)。

5) 准备要转换的Oracle 7数据库，这是在运行移植工具之后独立的关键性的阶段。准备Oracle 7数据库，移走 (移动或重命名) 控制文件，因为新的 Oracle8控制文件也在原来的位置上产生。确保所有其他文件、数据文件和重做日志文件原封不动。

同时, 准备操作系统环境, 重置 ORACLE_HOME 操作系统变量 (在 NT 的注册表或 UNIX 的壳环境变量中) 为新的 Oracle 8 目录位置。然后, 重置 Oracle 二进制可执行文件路径 (在适当的 NT 注册表分枝树中的映象路径, 如果必要, 或 UNIX 的 PATH 变量, 尽管在 UNIX 系统中应该关心 ORACLE_HOME)。如果 ORACLE_SID 发生改变, 重置这个变量。如果需要, 设置 ORA_NLS33 参数。

编辑你的新的 Oracle 8 init.ora 文件以包含参数设置 compatible = 8.0.0 (或更高)。同时移走过时的 Oracle 7 参数 (请参考你的服务器手册, 以找到最新版本)。确保你对 control_files = () 和其他文件位置 (例如 bdump 和 cdump) 的参数设置是正确的 (也就是, 不能包括任何 Oracle 7 的路径引用)。

从 Oracle 7 的 \$ORACLE_HOME/dbs 子目录到 Oracle 8 的 \$ORACLE_HOME/dbs 子目录复制转换数据文件 (conv<SID>.dbf)。

6) 转换你的数据库。这是不可返回的操作, 换句话说, 在你运行转换后, 你的数据库要么转换成功, 要么失败, 没有完全冷备份, 是不能返回到 Oracle 7 状态的。为做到这点, 运行命令 svrmgrl (UNIX) 或 svrmgr30 (NT):

```
svrmgrl> connect internal
svrmgrl> startup nomount
svrmgrl> alter database convert;
svrmgrl> alter database open resetlogs;
```

7) 完成最终的整理工作, 涉及到几件事, 包括删除移植工具运行时创建的移植用户 (用 cascade 选项) 并运行 catalog.sql 和 catproc.sql 脚本。然而, 有一个新的脚本, cat8000sql, 当它被运行时, 它为你完成所有整理工作。

移植工具的图形化界面与命令行版本的 mig 相对应, 它能从 Oracle 安装程序 (orainst) 被访问和运行, 并正好在移植工具被安装之后被执行。选择如下选项 “ Install ” “ Migrate from Oracle7 to Oracle8 ” “ Run Migration Utility ” “ Migration、Utility、Oracle7 to Oracle8.x.x.x ”。然后你将被提示做一些选择, 这些对应于命令行 mig 选项, 如 check_only。如果你做出选择仅检查空间要求, 那么与命令行版本一样, 你必须重新运行工具以执行真正的移植 (不再选择那个选项)。因此, 用 GUI 取代命令行版本, 你应完成任务 1 和 2, 但没有完成任务 3, 因为 GUI 将安装 Oracle 软件和移植数据。

9.2.3 移植前的任务

在你开始运行移植工具之前, 必须计划和做一些重要的事情, 以下所列步骤概括了移植前的检查列表。

- 查看新的 Oracle 8 特性。
- 选择你的移植方法。
- 估计硬件和软件需要的资源。
- 获得需要的资源。
- 开发可接受的准则 (用它来测试)。
- 制定移植规划。
- 备份 Oracle 7 源数据库。

为了帮助你了解新的 Oracle 8 特性, 阅读整个第三部分 “ Oracle 8 RDBMS ”。在这一部分

中，你能决定移植到 Oracle 8 对你现在的安装是否是适当的。另一方面，当你决定移植时，还要考虑除新功能和增强之外的因素。其中最重要的一点是在逐渐进步的基础上 Oracle“不支持” Oracle 旧版本的策略。例如，计划在 2000 年底不支持所有的 Oracle 7，特别是到 1998 年 12 月 31 日不再支持 Oracle 7.3.3 和较早期的版本，而 Oracle 7.3.4 是 Oracle 7 最终的版本，计划到 2000 年 12 月 31 日不再被支持（但是最后的不再支持期限已经被改变了两次）。

当决定选择哪种移植方法时，考虑空间和时间需求是十分重要的。另外，考虑你的数据库的关键性和它所要求的正常运行时间（如果有的话）。如果你的数据库必须是高可用的，并且非常巨大，使用移植工具可能是最好的途径，特别是在实际处理移植产品之前，多次用于测试方式时。另一方面，如果你的数据库相对小或不是很关键的，导出 / 导入与 COPY/CTAS 都是很好的选择。在这种种情况下，使用移植工具的唯一原因，可能是为了熟悉工具自身，如果在移植路线中还需要移植其他的更大更关键的数据库，这也是一个原因。

从你选择的移植方法的当前任务中，回顾你需求的空间和时间，估计你的硬件和软件资源的需要。除了增加 Oracle 8 系统表空间和二进制文件所需的存储空间（这是与移植方法无关的常量）之外，你将发现移植工具（内建）方法将需要最少量的额外量或进程、空间。这是真的，因为不需要中间的过渡媒介。同时，你将发现移植工具是最快速的方法，因为移植的时间与需要移植的数据库对象成正比，而不是与总的数据库大小成正比。最后，用移植工具方法，所有的事情中每件事物只需要一个：一台机器、一个服务器、一个实例、一个数据库等等。这是显然的，因为旧的数据库被“内建”移植成为新的数据库。

用导出/导入方法最明显的资源要求是对完全数据库导出文件需要相当大的总存储空间。但是你能将其直接导出到磁带中，至少在 UNIX 系统中可以这样做，并且这将联机存储需要转换为脱机存储需要。用 COPY/CTAS 方法，明智的存储空间需求是在新的数据库中的数据库对象至少需要与在旧的数据库中一样多的存储空间（实际上，由于新的 ROWID 和其他因素，存储空间要更多一点）。对导出/导入和 COPY/CTAS 这两种方法，它们几乎都需要每种两个事物；两个实例、两个数据库，甚至可能两台机器，例如如果是源数据库已经耗费几乎所有的机器资源。对 VLDB 而言，相对于移植工具，导出/导入和 COPY/CTAS 方法都是代价十分大的方法。

当你确定了你的资源需求，假定你选定了移植方法，获取这些资源。安装和配置第二台机器，镜像主机的环境、资源分配和尽可能接近的利用。如果有两台机器，一台机器明显地比另一台机器具有更强的能力和更多的资源，应使目标 Oracle 8 实例形成在这台机器上，除非有不能不可行的或不能接受的原因。

9.2.4 接受的标准

接受的标准可以包括一系统测试，这些测试在表 9-2 中描述。

假设选择了移植方法、大小、关键性和数据库的正常运行时间以及其他因素，决定实现移植的时间表。如果你的数据库较小或者是非关键的，那么可能在周末或甚至在工作日进行简单移植。但是，如果你的数据库是另一种极端，在你的企业中具有高度“可见性”，那么如果可能，考虑进行脱机移植。例如，如果你的数据库有一天 24 小时一周 7 天不间断的正常运行时间要求，你怎样才能升级？如果你采取完全冷备份，并在另一台机器上重建（那就是，做一个克隆数据库），然后通过某一方式移植克隆体。但是如果你不能丢失从备份时到升级时过程中事务的活动，那么如果可能，考虑创建一个备用数据库或复制数据库，并升级第二个数据库，因

此不会丢失任何事务的活动。如果升级失败，你仅需重新克隆并更新你的第二个数据库。

表9-2 接受的标准

测 试	描 述
移植	移植源数据库的一个子集或者克隆整个数据库。子集能够很容易地用导出 / 导入方法测试，但是克隆数据库将需要测试移植工具，不需要某种类型的备份 / 重建
最少	在移植之后，测试目标数据库应用的最少功能
功能	在移植以后，测试目标数据库应用的全部功能。换句话说，遍历所有可能的功能性路径组合
集成	通过测试所有与目标数据库上核心应用发生交互作用的外部组件（接口），测试整个系统
性能	在移植之前，列出应用的所有性能基准，或者至少是关键的功能性组件的基准。移植发生之后，运行相同的功能性组件（基准），并测量同样的性能方面，将移植之前与之后进行比较，移植之后（Oracle 8）的执行至少应该与移植之前（Oracle 7）相同，可能会更好一些
装载	也称为容量测试或承受力测试。这种类型的测试可以与性能测试联合进行，通过改变用户数和/或数据容量执行测试。然后，比较移植之前和之后的性能，查看移植之后是否执行得与移植之前一样好或更好

在移植到 Oracle 8 之前，备份你的源 Oracle 7 数据库，其意义应该不言而喻的。然而由于许多 DBA 没有足够重视，而没有这样去做。但是如果你选择移植工具方法，结果失败了并且没有好的备份来恢复重建的话，你将处于最糟的境地。

9.3 常见问题

一个常见问题是资源限制：SYSTEM 表空间中空间不够。Oracle 8 需要的空间大约是 Oracle 7 空间总量的 1.5 倍，而且在 Oracle 8 的 \$ORACLE_HOME/bin 下将占有的空间大约是 Oracle 7 下的空间的 3 倍。

部分使用 ROWID 是另一个常见问题。参见第 14 章“Oracle 8i 附加主题”，这一章中说明，在 Oracle 7 应用中 ROWID 的部分使用将不能导入到 Oracle 8，除非使用新的 Oracle8 DBMS_ROWID 包重新写它们以适应新的 Oracle8 ROWID 结构。

移植使所有编码对象无效，所有表数据分布统计（对基于成本的优化器）丢失，有些索引可能被标为不可用。

GUI 移植工具（orainst）仅只使用 US7ASCII 字符集；与之相反，命令行版本可以指定任何字符集。

在 Oracle 7 只读表空间中的表的 ROWID 不能转化为新的 Oracle8 ROWID（因为它们只是只读的）。工作区需要临时地改变表空间到读 - 写（常规）状态，从这些表空间中每一个只读表 SELECT*（强迫 ROWID 转化为使用全表扫描），然后将它们改变回只读状态。不这么做，只读表空间中的表在每次读取时不得不经临时的 ROWID 转换。

9.4 移植之后的任务

除了明显的移植后的活动（备份新移植的 Oracle 8 数据库和测试你在移植前制定的接收标准）外，还有几个其他重要项目。这些包括（性能）调整你的 Oracle 8 数据库和安装非捆绑（可选）产品，如 Object Option。你所依赖的许多系统管理脚本必须进行修改，以帮助你监控和管理存储和增长，以及性能，有效地给出新的 Oracle 8 数据结构、对象和视图。如果没有其他原因，仅只是为利用 oracle 8 的某些新特性，你的 Oracle 7 应用将可能必须被修改。如前面所提到的，那些依靠部分 ROWID 操作的应用必须被重写。最后，如果 SQL*Net 连接字符串不是全部指向 Net 8，依赖它的应用将必须相应地修改。

第10章 分 区

本章要点：

什么是分区

Oracle 8分区：一些例子

分区索引

维护操作

并行能力

附加的考虑

分区（partitioning）可能是Oracle 8 RDBMS中最重要的一個新特性。尽管它的有些形式已经存在于Oracle 7中，但是它在Oracle 8得到增强，并更完全地并入RDBMS引擎。分区这个术语被“重载”。换言之，对于数据库理论和其他厂商（例如 Informix），它有很多层意思。但是，当然这里所处理的是特定的 Oracle分区表和索引能力。

10.1 什么是分区

在Oracle 7中首次出现了用分区视图进行分区的能力。分区视图帮助按照用户定义的业务规则、条件或规范，物理地分开磁盘上的数据存储。从本质上讲，表和索引是能够根据给定的值被分离开的。例如，一个州的列包括两个字符的州名缩写，一个表能够按组成国家不同地区的州（例如 East Coast）存储在分离的磁盘上。

这是通过使用检查约束来完成的，通过互斥来执行。也就是说，对给定的列的所有不同子集的值，当被分配时，不能重叠。如果它们会重叠，你将可能在多个磁盘中存储同一行记录，但是Oracle如何知道使用哪一个记录呢？另外，手工建立相互排斥的分区，你应将设置 partition_view_enabled初始化参数为真。这将告诉优化器去研究对所有表被分区的可能性，以使WHERE语句可能需要访问唯一的或少量的几个分区，而不是访问组成整个表的全部分区。对经历全表扫描的大表，这种有效性更为显著，但在其他环境中也能意识到这种有效性。例如，用大的索引扫描或中等尺寸的表或索引访问能够提高分区的有效性。优化器操作如果知道分区边界，就能够跳过分区，称为分区消除。清单 10-1显示了如何在Oracle 7中建立分区视图：

清单10-1 在Oracle 7中创建分区视图

```
SQL> create table EAST_REGION (STATEID,...) ...;
SQL> create table WEST_REGION (STATEID,...) ...;
SQL> create table NORTH_REGION (STATEID,...) ...;
SQL> create table SOUTH_REGION (STATEID,...) ...;
```

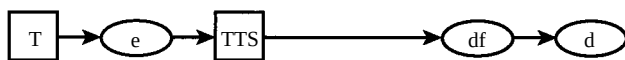
```
SQL> create view NATIONAL as
2 select * from EAST_REGION union all
3 select * from WEST_REGION union all
4 select * from NORTH_REGION union all
5 select * from SOUTH_REGION union all;
```

```
SQL> alter table EAST_REGION
2 add constraint CKSTATE check
3 (STATE_ID between 1 and 13);
...
SQL> alter table SOUTH_REGION
2 add constraint CKSTATE check
3 (STATE_ID between 39 and 50);
```

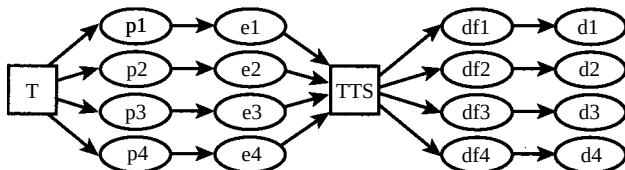
Oracle 7 分区视图的能力被加强和合并到 Oracle 8中，现在没有像partition_view_enabled这样的初始化参数需要告诉优化器。除了在 Oracle 7中的初始化参数以外，分区通过约束和视图来实现。实际上，整个被分区的表是一个由约束创建和存储的所有这些平行的片段的联合体。平行片段自身在实际的表中。

在Oracle 8中情况有所不同，因为在 Oracle 8数据库存储结构中，分区是一个新的物理层，驻留在表和它的区间之间。表被映射为一系列分区，这些分区反过来逐一映射到一系列区间，这些区间中的每个映射到一个或多个表空间，每个表空间具有一个或多个数据文件。图 10-1 比较了分区表和非分区表。

非分区的表



分区的表



T=表，e=分区，TS=表空间，df=数据文件，d=磁盘，p=分区

图10-1 比较分区表和非分区表

更进一步，分区是Oracle扩展SQL数据定义语言（DDL）的一部分。分区作为语言的一部分是十分重要的，这意味着优化器是“知道分区”的，并可以在不需要像初始化参数这种东西的情况下，利用分区消除的优点。表和索引能够被分区。

分区策略

这里存在着几种分区策略。数据库对象（如表或索引）能够使用列值范围被分区，在 Oracle 7或Oracle 8中均可以这么做，这就是通常所说的键值分区，因为列时常是表的索引或键值。Oracle 7能够用BETWEEN...AND或<=或>=不等式限定范围，Oracle 8要求你使用新语言LESS THAN。

有一些Oracle目前没有实现的其他类型的分区，这些分区包括散列分区和表达式分区。正如你所见的，键值分区是表达式分区的特殊情况。为什么有人需要 Oracle 7或Oracle 8不提供的表达式呢？当前使用的Oracle只能采用一列的顺序范围的连续子集对表进行分区。假如纯粹是为性能的原因，你想使用某些MOD或RANDOM函数以某种方式平均分布数据，而以任何特定的列值无关。通常而言，这有可能损坏正常能获得的分区消除的好处；但在有些情况下，

它可能是有益的。例如，分区可有可能作为 Oracle 数据条和 RAID 的替代品。图 10-2 显示这个效果。看一看分割的方法是怎样强烈影响行的分布的。

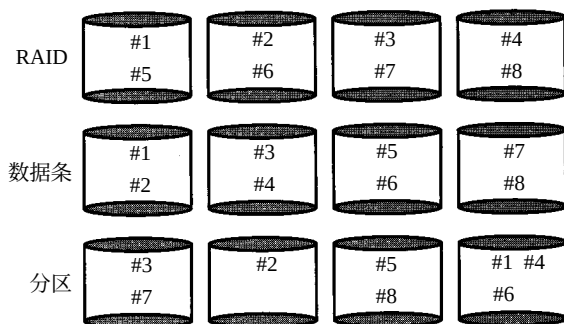


图10-2 RAID、数据条和分区的比较

花费某些代价，你能在 Oracle 中使用键值分区和每个表中的一个附加列模仿表达式分区。例如，假如你想将 5 亿行的 Oracle 表随意分区，你可以在你的表中增加一列，该列含有对每一行唯一的机器产生的随机数，然后使用 LESS THAN 关键字应用 Oracle 8 的键值分区。你能做到这一点，是因为你控制范围或随机数的产生；因为你仅需要一定范围的数字做键值分区。当然，缺点是在很大的表中有一个必须删除的附加列。

在 Oracle 7 或 Oracle 8 中，通常将并行查询与分区视图或分区联合使用。通常设置并行程度等于在版本 7 中组成分区视图的个数的倍数，或等于在版本 8 中组成表的分区数量。你不仅能够获得优化器分区消除的优势，而且你也能获得在其余那些不能被消除的分区中进行并行查询的优势。因此，通过使用并行查询，通常在存储大量数据的应用中使用分区。这些应用倾向于巨型数据库、数据中心或数据仓库。但是，联机事务进程系统也能受益，因为它们自身也可能是很大的数据库。

10.2 Oracle 8 分区的一些例子

在 Oracle 8 中，一个表或索引能够达到 64 000 个分区，分区键是被分区的表或索引上的列。与主键一样，分区键能够组合，最多可以达到 16 列。在分区表中，既不支持簇，也不支持大对象（如 LONG 或 RAW）。非分区表（通常）既能有分区索引，又能有非分区索引；同样，一个分区表既能有分区索引，又能有非分区索引。但是，除简单性外，如可管理性和性能等问题也影响你的物理设计。清单 10-2 是一个 Oracle 8 分区表的例子。

清单10-2 在Oracle 8中创建一个分区表

```
SQL> create table EMPS
2 (EMPID number(5), EMPNAME varchar2(30), EMPADDR varchar2(75))
3 partition by range (EMPID)
4 (partition P1 values less than (10001),
5 partition P2 values less than (20001),
6 partition P3 values less than (30001),
7 partition P4 values less than (40001),
8 partition P5 values less than (50001));
```

在清单 10-2 中，假设你有 50 000 个员工，员工号是 1~50 000，在 EMPS 表创建 5 个分区，每个分区存储 10 000 个员工。

一个表的单个分区能够有不同的表空间和存储参数，如同清单 10-3所示。

清单10-3 分区表的表空间和参数

```
SQL> create table DEPTS
1 (DEPTID number(2), DEPTNAME varchar2(20),
2 DEPTBUDGET number(9)
3 partition by range (DEPTID)
4 (partition P1 values less than (51)
5 tablespace T1 storage (initial 10M next 10M),
6 partition P2 values less than (101)
7 tablespace T2 storage (initial 20M next 20M));
```

VALUES LESS THAN子句指定了那个分区的上限（不含此值）。例如，在清单 10-3中，分区P1能包含DEPTID值最高为50，但不能是51或更高。因此在VALUES LESS THAN子句中的LESS THAN的意思是严格地小于。每个分区自动在分区上创建一个约束。例如，企图插入一行DEPTID=101会引起ORA-14400错误：“插入的分区键值超过最高合法分区键值”，因为最高分区（P2）的最大允许值是100。为了避免发生这种情况，可以使用关键字 MAXVALUE，如清单10-4所示，这是一个对清单 10-3进行的简单修改。

清单10-4 分区表使用MAXVALUE

```
SQL> create table DEPTS
1 (DEPTID number(2), DEPTNAME varchar2(20),
2 DEPTBUDGET number(9)
3 partition by range (DEPTID)
4 (partition P1 values less than (51)
5 tablespace T1 storage (initial 10M next 10M),
6 partition P2 values less than (maxvalue)
7 tablespace T2 storage (initial 20M next 20M));
```

唯一不同的是在最高分区 P2中值101被关键字MAXVALUE替代。另外，NULL不能用于VALUE LESS THAN 语句，当被插入的分区键是NULL值时，它们被分类为大于所有直接量值，但不大于MAXVALUE。

使用分区的例子

正如前面提到的，分区键可以由多列组成（也就是复合）。进行比较的顺序是从左到右；如果MAXVALUE被使用，MAXVALUE使用之后的所有值被忽略。当在一个表上有一个组合键时，或者你想在频繁连结的表上通过主键和外键组合进行分区时，组合分区能够起作用。清单 10-5给出有一个主键的分区表的例子。清单 10-6给出了在一个表上有主键和外键组合的分区例子。

清单10-5 用主键分区

```
SQL> create table EMPS
2 (SSN number(9), SEQNO number(3),
3 EMPNAME varchar2(30), EMPADDR varchar2(75))
4 primary key (SSN, SEQNO)
5 partition by range (SSN, SEQNO)
6 (partition P1 values less than (333333334,334),
7 partition P2 values less than (666666668,668),
8 partition P3 values less than (1000000000,1000));
```

清单10-6 用主键和外键组合分区

```
SQL> create table EMPS
2 (EMPID number(5), EMPNAME varchar2(30),
3 EMPADDR varchar2(75), DEPTID number(2))
4 primary key (EMPID, DEPTID)
5 foreign key (DEPTID) references DEPTS
6 partition by range (EMPID,DEPTID)
7 (partition P1 values less than (10001,21),
8 partition P2 values less than (20001,41),
9 partition P3 values less than (30001,61),
10 partition P4 values less than (40001,81),
11 partition P5 values less than (50001,101));
```

10.3 分区索引

有四种主要类型的索引可以用于分区或非分区表，它们是：

- 1) 非分区（常规的）索引。
- 2) 全局前缀索引。
- 3) 本地前缀索引。
- 4) 本地非前缀索引。

为了理解这些索引类型，首先需要学习一些背景知识。回想被分区的表能够有分区索引或非分区索引；在索引分区上唯一真正的限制是一个聚簇的索引不能再被分区。如果索引的最左列正好与分区键具有相同的顺序和数量，该索引被认为是前缀的。另外，索引列可以是分区键的超集，而不是子集。

全局索引只能被前缀，但是它的分区分布通常与它的相关表不同。如果想得到全局非前缀索引，应使用一个常规的索引替代。Oracle选择不实现这种能力，因为利用它的开销将比使用常规的索引的性能差。全局索引不被 Oracle 维护。也就是说，在被创建之后，Oracle 就不再对在索引分区和表分区之间的特定关系（值范围一致性）进行维护。全局索引最主要的缺点是 Oracle 优化器不能利用全局索引进行分区消除。另一方面，全局索引对分区索引值的范围提供选择，而不是对相关表的值范围提供选择。这在多索引的情况下是有用的，例如查找使用数据仓库和相关联机分析处理应用的多维数据库。同时，当存储非常珍贵时，全局索引是有吸引力的选择，因为它们可能比它们相关的表需要更少的分区。清单 10-7 给出了分区表和相关的全局前缀索引的例子。

清单10-7 分区表的全局前缀索引

```
SQL> create table EMPS
2 (EMPID number(5), SSN number(9), EMPNAME varchar2(30), EMPADDR varchar2(75))
3 partition by range (EMPID)
4 (partition P1 values less than (33334),
5 partition P2 values less than (66668),
6 partition P3 values less than (100000));

SQL> create index EMPS_IDX
2 on EMPS (EMPID, SSN)
3 global
4 partition by range (EMPID)
5 partition P2 values less than (50001),
6 partition P3 values less than (100000));
```

本地索引既可以被前缀，也可以不被前缀。但是，在索引分区和它们相关表分区之间有一一对应的关系。当建立时，这种关系被认为是等分区的，因此由 Oracle 维护。Oracle 优化器能够利用分区消除的优势，这样相对于数据条与 RAID 更能够潜在地提供本质的性能增强。因此，本地索引最主要的优点是自动等分区维护和相关的潜在性能收益。用本地索引唯一不利的是，它严格要求必须与它的相关表具有相同的分区模式，不管是否前缀。对多索引需要，这可能需要太复杂的管理。同时相关的交叉存储可能会成为一个难题。

清单 10-8 给出了分区表和相关的本地前缀索引的例子，清单 10-9 给出了分区表和相关的本地非前缀索引的例子。注意，在这两种情况下都没有 PARTITION BY RANGE 或 VALUES LESS THAN 语句，因为 Oracle 知道这是本地索引，将按照完全相同的表声明对它进行同等分区（LOCAL 关键字告诉 Oracle）。将清单 10-8、清单 10-9 与清单 10-7 相比较可以看出本地和全局的不同，本地前缀与本地非前缀之间的不同。

注意 尽管本地索引分区的区别仅在于选择了不同的索引列，但实际上决定它们是否被前缀，全局索引分区的区别取决于值范围的选择的不同。

清单 10-8 分区表和相关的本地前缀索引

```
SQL> create table EMPS
2 (EMPID number(5), SSN number(9),
3 EMPNAME varchar2(30), EMPADDR varchar2(75))
4 partition by range (EMPID)
5 (partition P1 values less than (33334),
6 partition P2 values less than (66668),
7 partition P3 values less than (100000));

SQL> create index EMPS_IDX
2 on EMPS (EMPID, SSN)
3 local
4 (partition IP1,
5 partition IP2,
6 partition IP3);
```

清单 10-9 分区表和相关的本地非前缀索引

```
SQL> create table EMPS
2 (EMPID number(5), SSN number(9),
3 EMPNAME varchar2(30), EMPADDR varchar2(75))
4 partition by range (EMPID)
5 (partition P1 values less than (33334),
6 partition P2 values less than (66668),
7 partition P3 values less than (100000));

SQL> create index EMPS_IDX
2 on EMPS (SSN)
3 local
4 (partition IP1,
5 partition IP2,
6 partition IP3);
```

10.4 维护操作

有两个基本的语句执行所有的分区操作，它们是 ALTER TABLE 和 ALTER INDEX。对

ALTER TABLE语句的分区扩展语句如下：

```
DROP PARTITION
ADD PARTITION
RENAME PARTITION
MODIFY PARTITION
TRUNCATE PARTITION
SPLIT PARTITION
MOVE PARTITION
EXCHANGE PARTITION
```

对ALTER INDEX语句的分区扩展如下：

```
DROP PARTITION
RENAME PARTITION
REBUILD PARTITION
MODIFY PARTITION
SPLIT PARTITION
PARALLEL
UNUSABLE
```

对有经验的Oracle DBA来说，大多数这些分区操作应该是熟悉的和自我说明的，因为它们通常看上去同平常的表和索引操作（如 DROP TABLE和DROP INDEX）一样，但是你应该仔细观察那些分区独有的操作（SPLIT、MOVE、EXCHANGE和UNUSABLE），清单10-10给出一个SPLIT的例子。

清单10-10 SPLIT PARTITION 维护操作

```
SQL> alter table EMPS split partition P3
1 at (75000)
2 into (partition P3, partition P4);
```

在清单10-10中所发生的是P3分区保持所有EMPID的值小于75000（但大于在P2分区中的值）并产生一个新的P4分区存储EMPID的值等于或大于75000的行。正如你所见的SPLIT总产生一个附加的新分区。

清单10-11给出一个MOVE操作的例子。正如你所见，这只是将一个分区从一个表空间移动到另一个表空间。

清单10-11 移动分区

```
SQL> alter table EMPS move partition P3
1 tablespace T3;
```

清单10-12显示EXCHANGE语句的一个类型。EXCHANGE将一个分区表转换为一个非分区表或将一个非分区表转换为一个分区表。当你想将 Oracle 7分区视图移植到Oracle 8分区表时很可能使用EXCHANG。

清单10-12 将分区表转换为非分区表

```
SQL> alter table EMPS8 exchange partition P1
```

```
1 with table EMPS7
2 including indexes
3 without validation;
```

清单10-12也说明，你可以有选择地包括索引列（用一定的限制）和不需要确认被移动的行范围。缺省时不包括索引并需要确认。

索引分区可以被Oracle标记为索引无效（IU），意思是你如果不重建整个索引，也必须重建至少是那些组成这个索引的IU分区。

标志一个索引为IU的6种维护操作如下：

- 1) SQL*Loader或导入具有忽略索引的选项的分区。
- 2) SQL*Loader或导入具有忽略索引的选项的分区。
- 3) ALTER TABLE TRUNCATE PARTITION
- 4) ALTER TABLE SPLIT PARTITION
- 5) ALTER TABLE MOVE PARTITION
- 6) ALTER INDEX SPLIT PARTITION

10.5 并行能力

有三种主要并行能力能够用于分区以增强性能：

- 1) 并行查询（PQ），也称为并行查询选项（PQO）。
- 2) 并行数据定义语言（PDDL）。
- 3) 并行数据操作语言（PDML）。

并行查询操作在分区表和非分区表中仅有微小的不同。在非分区表，仅当表中的行数大于或等于在编译时被检查的内部门限时，表的全表扫描才被并行化。相反，在分区表中，仅当相关分区总的大小大于或等于那些相同的内部门限时（该门限不仅在编译时被检查，而且如有需要，在运行时也进行检查，这是因为查询自身的分区限制），才将该表的全表扫描并行化。如果在检索与查询需要相应的分区值范围（边界）时，优化器在分区消除时，分区不被跳过，就称为是分区相关的。使用并行查询，并行度由 ROWID 范围确定，与表的分区无关。

并行数据定义语言被限制为 DDL 命令中可以并行化的一个很小的子集：

■ CREATE INDEX

■ CREATE TABLE AS SELECT

■ ALTER TABLE MOVE PARTITION

■ ALTER TABLE SPLIT PARTITION

■ ALTER INDEX REBUILD PARTITION

除了附加的特定分区的操作，这种类型的并行化在 Oracle 8 中与在 Oracle 7 中相同。

并行数据操作语言（DML）可能有点模糊，但基本上，在 Oracle 中通常指的是 INSERT、UPDATE 和 DELETE 操作，不是 SELECT 操作。尽管在 Oracle 之外的数据库的书本和著作中，首字母缩写 DML 代表数据操作语言，但是 Oracle 更愿意让你认为 DML 是数据修改语言。如果你认为 DML 是数据修改语言，而 SELECT 是查询，在 Oracle 术语中是指并行查询处理 SELECT 的并行化，而 PDML 是处理 INSERT、UPDATE 和 DELETE 的并行化。

正如你可能想象的，PDML 在性能方面的影响通常很少与 PQ 一样大，特别是在 OLTP 和 DSS 系统中。但是，有时有一些孤立的处理或在这些应用类型中高度易变的表，这些表的增

长和收缩十分迅速。更进一步，表的这些类型在某些应用中有可能变得十分巨大。当后面这些条件似乎会在你的组织中出现时，PDML应该与PQ一样对你有益，甚至更好。

和使用并行查询一样，PDML的优点依赖于被访问表的行数、这些表所跨的磁盘驱动器数、服务于这些磁盘的控制器数目和类型、CPU的数量和其他软硬件因素。例如 Oracle 使用磁盘到数据文件再到分区的映射来帮助将 PDML 语句的工作分离和分配到它们的伺服器，同时，与并行查询一样，PDML 提供一个失败安全水平。如同所有真正的面向事务的 RDBMS 一样，如果一个 PDML 伺服器的工作失败，所有伺服器都回滚，这样全部 PDML 语句失败，保证了事务的原子性。这种操作很像一个本地两阶段提交操作。用 PDML 时，UPDATE 和 DELETE 操作要求表被分区，但 INSERT...SELECT 不要求这么做，就和用 SELECT 和 PQ 时一样。

注意 没有初始化参数能启用 PDML。如果要启用 PDML，必须使用 ALTER SESSION ENABLE PARALLEL DML。但是优化器可能由于其他限制，选择不并行 DML。在后用的 PDML 会话中的每一条语句必须跟随一条 COMMIT 或 ROLLBACK，以阻止 ORA-12830 错误：“在执行并行 INSERT/UPDATE/DELETE 返回后，必须提交或回滚”。另外，如果在一个事务（PL/SQL 代码块）中的第一条 DML 语句没有以并行方式执行，所有随后的语句被顺序执行，而不管提示或缺省表的并行度。

10.6 附加的考虑

PDML 是在 Oracle 8 中与分区有关的一项主要功能。除此之外，许多其他特性和功能与这些新的分区能力有关。分区扩展表名就是一个伴随分区的增强，这种语言的扩展是非标准的（也就是非 ANSI），因此你不得使用视图来封装代码，来促进标准化。清单 10-13 给出了使用分区扩展表名的例子。

清单 10-13 一个分区扩展表名

```
SQL> select * from EMPS partition (p1);
```

分区锁增加了另一层对 Oracle 7 锁规划的粒度，以支持 Oracle 8 的分区独立的概念。分区独立的简单含意是你能够完全像对待 Oracle 7 中的表一样对待 Oracle 8 中的分区。使用这种方法，一个 DBA 能够维护一个分区而不影响用户访问其他分区。本质上，分区级锁作为在表级和行级锁之间的锁级别存在。总之，分区独立的优点是独立管理恢复和备份这些事，这也能被看作高可用性特性，因为一个分区失败时其他分区仍可被访问。

警告 我建议在使用分区时不要更新分区键的列，那将引起行从一个分区移动到另一个分区。在 Oracle 8 分区中，没有自动的行移植。如果你试图做这样的更新，你将接收到错误“ORA-14402：更新分区键列，将引起分区改变”。用 Oracle 8 时，DBA 不得不手动移植行。

现在，假设你有一个员工表（EMPS），它是使用员工标识符（EMPLID）进行分区的，并有两个分区 P1 和 P2，P1 中的值小于 51，P2 中有 51 以及以上的值。清单 10-14 给出两个更新语句例子，第一条语句成功，但是第二条语句失败，并伴随一个 ORA-14402 错误，如前面的警告中所提到的。解决这个问题的一种方法是做一个插入和删除，但是对成批操作这是无效

的。在这种情况下，你最好用前面讨论的分区操作，或卸载和重新装载你的分区。

清单10-14 分区更新语句

```
SQL> update EMPS set EMPID = 67
2 where EMPID = 66;
/* This works ok, since this particular employee stays in partition P2 */

SQL> update EMPS set EMPID = 67
2 where EMPID = 33;
/* This will not work, since it would cause this particular employee to move
from partition P1 to P2. Hence, it will generate an ORA-14402 error. */

SQL> insert into EMPS partition (P2)
2 values (33, ...);
SQL> delete from EMPS
2 where EMPID = 67;
```

最后一个考虑是新的 Oracle 8 ROWID 格式是与 Oracle 7 相对的。在 Oracle 8 中，ROWID 被扩展，主要用于调节分区，但是也被应用于调节对象选项，这将在第 11 章“大对象的历史”中讨论。这个新的 ROWID 格式在第 13 章“Net 8 网络：新特性和概念”中深入进行讨论。

第11章 大对象的历史

本章要点：

历史

Oracle 7与Oracle 8结构的对比

LOB

BFILE

使用LOB和BFILE的例子

DBMS_LOB包

更多的一些例子

最后的一些问题

11.1 历史

关系型数据库管理系统（RDBMS）一开始对大对象（LOB）的存储和查询的尝试是粗糙的。有趣的是尽管这些开始的尝试是粗糙的，但是他们的确在一些初步的级别上完成了这项工作，这可能解释为什么这些早期的方法在今天仍然对于不同厂商以不同的名字存在。

大部分RDBMS厂商做的早期尝试是外部存储数据类型方法，通常在数据库世界中称之为二进制大对象（BLOB），不要与本章后面将要讨论的内部 Oracle 8二进制大对象数据类型（BLOB）相混淆。这是一种管理大的、未被结构化数据的常规方法。大的意思是什么？这实际上对于不同的RDBMS厂商是不同的，但是通常它的意思是比厂商能支持的最大的变长字符要大。例如在Oracle 7中，最大的变量是varchar2 (2000)。Rdb/VMS、Sybase和Oracle在二进制大对象上都有一些变化。Oracle 7有LONG和LONG RAW，它们在本章后面部分“Oracle 7与Oracle 8结构的对比”中与Oracle 8新的数据类型做比较。未结构化的意思是什么？一般来讲，它的意思是数据可能代表平凡（字母数字）信息，如注释、标记、记事簿、摘要、对调查问题的开放结尾回答等等。那些熟悉dBASE的用户应使用过dBASE术语MEMO域。但它也有多媒体数据的意思，包括图形、工程图、图像、视频、声音等等。

换句话说，BLOB被创建以存储不规则结构的数据，但是能可靠地分解到一张表的每一行的非空列上。实际上，如果数据在某种程度上依附在一个规范化表的右边，由于在每一行中有不同长度的数据，这个表看起来将是不规则的。对于熟悉C/C++的用户，他们将很快认识到这是一个“不规则数组”。从本质上说，对大多数关系数据库设计者而言，将不会这么做：在数据库内部存储数据，依附到一个特定的表并破坏规范化的设计。在今天所有的RDBMS厂商仍然使用的快速补救方法是BLOB，BLOB将存储在数据库外部的操作系统文件中。存储在表中的将是列名和在每一行的那列中指向相关的外部文件的指针。这样，表好像被“正规化”了，因为指针自身具有相同的数据类型（是完全专用的和对特定的RDBMS厂商的实现特定的）。

使用专有实现的缺点

与大多数厂商对关系模型的扩展一样，在 RDBMS 中的专有数据类型不是（现在仍然不是）一个好主意。为什么？因为这将完全或部分地牺牲可移植性。有这种情况：一个公司的应用完全用 Sybase 的 TEXT 列（一种 Sybase 的 BLOB 类型）开发；然后，由于这样或那样的原因必须抛弃全部的 Sybase 环境，公司不得不重新在 Oracle 中实现每一件事。猜一猜怎么了？是的，所有的 Sybase 专有的 BLOB（TEXT 列）不得不用 Oracle 专有的 BLOB（LONG 列）替代。

除了专有数据类型自身之外存在的其他不利因素是什么？第一，正如你可能预期的，专有数据类型意味着非标准，这意味着非 ANSI SQL，反过来意味着没有标准的语言支持数据类型。因此，只要有专有数据类型，你必须伴随着它使用专有语言存储、查询并用不同的方法操作它。换句话说，它需要自己的 DDL 和 DML 操作集，实际上，所有厂商都有这种情况。因此那些必须从 Sybase 转移到 Oracle 的公司不仅必须替换数据类型，而且必须重新编写代码，这通常是在任何转移操作中的最后一种选择手段。

除了数据类型自身和它的相关操作是否还有其他缺点？肯定有。其他特定厂商的环境和操作的行为模式通常要求你或者用“BLOB 使能”模式工作，或者不用这种模式工作。你经常会发现厂商（包括 Oracle 在内）在许多日常操作中不允许 BLOB 操作或访问。在 Oracle 7 中用 LONG 型工作的人员能够很好地理解这一点。

用 BLOB 最根本的问题是它们不是数据库“头等公民”。这意思是，尽管它们被外围存储和查询，但是它们并不与数据库中的其他数据或部分以任何形式发生交互。一种考虑这种情况的方法是 BLOB 实际上仅代表 RDBMS 具有附加的简单操作文件系统的能力，再也没有其他的了。到关系型模型的分支也不能被忽略。一般来说，这些早期 BLOB 是很少有或根本没有并发控制的，这意味着锁完全不是相对于数据库页或行的锁。另外，优化器不“了解”这些 BLOB，不能产生任何形式的包括它们的查询计划。在一个相关点上，如索引和普通的字符集数据子串操作不能用于 BLOB 上。你作为 DBA 或开发者可能想去做的所有其他事情，或者希望 RDBMS 处理的事情能不用 BLOB。因此，使用 Oracle 8，早期的 Oracle 7 的 BLOB 已经转生为另一种形式，但是 Oracle 的确朝着使 BLOB 成为关系模型实现中更加有用的列的方面取得了一些进步。

11.2 Oracle 7 与 Oracle 8 结构的对比

在 Oracle 7 中，LONG 和 LONG RAW 数据类型是内部 BLOB。除了它们是内部的之外，它们也具有 BLOB 的坏特性。在 Oracle 8 中，大对象（LOB）可以是内部的或外部的。表 11-1 概括了 Oracle 7 LONG 与 Oracle 8 LOB 之间的不同。

表 11-1 Oracle 7 与 Oracle 8 大对象的对比

Oracle 版本	7 (LONG)	8 (LOB)
最大尺寸	2GB	4GB
内部或外部	内部	皆可
访问	顺序	随机
SELECT 返回值	数据	指针
每个表的列	一个	多个

将旧的 LONG 与新的 LOB 数据类型进行对比是非常有益的。用 Oracle 7 的 LONG，仅有两个基本 LONG 类型：LONG 字符和 LONG RAW 二进制。Oracle 7 LONG 在它自己的“表中”存

储最多2GB的数据，只能按顺序进行访问，每一个表只能存储一列，SELECT语句直接返回实际的数据。

用Oracle 8 LOB存储数据能够达到4GB，既可以存储在表的内部又可存储在表的外部，能够被随机（直接）访问，每一个表能够存储多于一列的LOB，SELECT语句返回一个指向数据（不管是在存储在表内部还是外部）的指针（定位器）。

11.3 LOB

在Oracle 8中，有四种类型的LOB：BLOB、CLOB、NCLOB和BFILE。它们概括如下：

- BLOB——内部二进制大对象。
- CLOB——内部字符大对象。
- NCLOB——内部定长多字节字符大对象。
- BFILE——外部二进制文件。

正如你可能注意到的，Oracle 8 BFILE是早期RDBMS BLOB（再一次指出，不要将其与Oracle 8 BLOB混淆）的直接继承，它作为数据库指针存储在数据库内部，指向数据库的外部操作系统文件。

LOB由两部分组成：数据（值）和指向数据的指针（定位器）。尽管值与表自身一起存储，但是一个LOB列并不包含值，仅有它的定位指针。更进一步，为了使用大对象，程序必须声明定位器类型的本地变量。你将在本章后面见到PL/SQL接口使用定位器如同操作系统的文件处理一样。当LOB（除了BFILE）被创建时，定位器被存放在列中，值被存放在LOB段中，LOB段是在数据库内部表的一部分。当BFILE被创建时，定位器如同平常一样存储在列中，但是它的值被存储在数据库之外的操作系统文件中。

正如你已学过的，内部LOB（BLOB、CLOB和NCLOB，但不包括BFILE）可以是一个表的列、一个SQL变量或者一个PL/SQL变量。最重要的是，考虑忠实于关系型模型的实现，内部LOB主要是修改并发控制、重做日志和备份/恢复机制，这代表了在早期的RDBMS BLOB基础上的进步。Oracle 8的BLOB在功能上与Oracle 7 LONG RAW相似，CLOB的功能与LONG相似。但是NCLOB是新出现的。它们被创建用来处理国家性字符（NC）集。最后，在这些不同的内部LOB类型之间没有隐式转换。

11.3.1 使用LOB创建表

正如你通常使用的创建表的语句一样，使用CREATE TABLE语句可以创建包含一些LOB列的表。在INSERT操作时，定位器和空值被创建。PL/SQL的DBMS_LOB包或者Oracle Call Interface (OCI) 能够用来装载LOB。删除包含LOB的行，不仅删除定位器，而且还删除相关的值（正如所希望的）。UPDATE语句能更新全部的LOB值，但是不能正好更新它之中的一块（例如在字符类型中子集的操作）。还有，当在INSERT创建LOB时，LOB索引段和LOB（值）段会自动地创建，这个索引段索引LOB值的片断以便随机访问。因此，每一个LOB创建时，创建两个段：一个LOB（值）段和一个LOB索引段。

11.3.2 LOB存储管理

内部LOB存储是如何管理的？存储和并发粒度称为一大块（chunk），它是由一个或多个

Oracle块（由db_block设置）组成的。大块的缺省大小是一个 Oracle块。实际上，一个大块是一串块（链表）。在一个大块中的块在逻辑上是连续的，尽管在给定的链表数据结构下不要求连续。这些知识可以用来协调数据库的其他方面，如设置 db_file_multiblock_read_count等于大块的大小能使性能增强。LOB索引维护大块的链表以使它们可以被直接访问；没有 LOB索引，它们只能被顺序访问。LOB索引是一个大块值到大块的地址的转换列表。版本化是使相同LOB的各种版本能够占有不同比例的 LOB段的机制。存储参数 pctversion表明LOB段空间的多少比例用于存储除了当前版本外的所有（老的）版本。如果当前 LOB的老版本超过 pctversion的值，那么 Oracle尽力从最老版本中申请空间，直到老版本小于或等于 pcversion的值。版本化发生在大块级。

另外两个存储参数 cache/nocache和logging/nologging分别控制I/O是否使用系统全局区数据库缓冲区和重做日志机制。缺省组合 nocache和nologging，对一般的性能是最好的。正如通常一样，当预期重复再使用 LOB时，使用 cache；当对LOB的关键性改变必须不能丢失时，使用 logging。这与通常的对象没有特别不同之处。

什么是内部LOB并发管理？结果是与块一样，大块并不只是存储单元（和 I/O）和版本化，而且也是并发的单元。对读一致性的处理与对其他普通的 Oracle数据类型非常相似。使用 LOB的最主要的区别在于粒度是大块而不是块，读一致性 LOB数据（老的大块代表老版本）被存储在LOB段内，而不是存储在回滚段或数据库缓冲区中。一般来说，如果对 LOB有很多更新活动，在相同的时间框架中还具有高选择性，那么 pctversion应该尽可能相对高一些，以使能更多的非中断并发。什么是中断？就是当使用读一致性时，用户可能遇到 ORA-1555错误“快照太老”。其基本意思是你需要更多的回滚空间（多个单个段存储或多个段）。使用LOB段，你能增加存储总量或者如果 pctversion充足，仅仅简单地增加 pctversion。与此相对，如果低的更新的活动或高的更新活动与高的查询活动不在同一时间发生， pctversion值不是关键的，可以设置得低一些。

11.4 BFILE

文件系统和第三代语言形成了前一代关系型数据库管理系统的基础，甚至是前数据库管理系统、信息存储和检索系统（或者简称为信息系统）的基础。但是，他们的缺点正是朝着 DBMS系统和最终朝着 RDBMS方向发展的驱动力。但是，尽管缺少主要的现代的 RDBMS子系统，基于文件的信息系统仍在某种程度上在一些领域内很广泛采用，特别是，某种基于文件的信息系统可能受益于旧的 BLOB外部存储和检索（在 Oracle 8中的BFILE）方法。最可能的候选者应该不是面向事务的，不是主要依赖于数据的完整性的，而是可能形成于一些松散的结构化的或非结构化的历史应用的基础上，如服务在因特网上的记录本或简历归档。

Oracle 8能够创建、相关和安全化 BFILE对象。然而对 BFILE所做的所有其他事必须通过 RDBMS_LOB包或OCI来做。BFILE是只读的。因此，服务于前述历史应用的 BFILE很适合存储在光盘（CDROM）中。数据必须已经作为操作系统的文件存在， Oracle必须有读该文件（当然还有目录访问、到达文件）的特权。在删除表空间时，数据文件作为操作系统文件仍留在操作系统中；同样，删除 BFILE对象时没有删除相关的操作系统文件。作为 Oracle的DBA要清楚这一点，必须亲自删除它们，或让操作系统管理员来做这项工作。

正如所提到的，当 BFILE被创建时，它的定位器存储在一些表的某一列中，而它的值是

相关的外部操作系统文件自身。BFILE定位器存储的信息不仅仅是文件名，还包括目录和其他有关的状态信息。在“规范化”的表中，每行和列交叉处能包含一个唯一的值（尽管它也能包含在外键中以前被使用的值）。同样，每个BFILE列的每一行能够指向不同的文件。正如同其他内部LOB一样，DBMS_LOB包和Oracle 8 OCI提供访问BFILE的I/O流。不同于发生在普通对象（例如表）和SGA正常块之间的I/O，BFILE I/O（通过DBMS_LOB或OCI）是直接的，类似于用SQL*loader直接的路径装载。换句话说。它绕过SGA。

数据库子系统如并发控制、重做日志和备份/恢复机制在Oracle 8中用BFILE都无法获得。另外安全性是有疑问的。因为BFILE在Oracle 8数据库内的安全性主要依靠定义它的操作系统的安全性。例如，为了BFILE的安全，SA或DBA必须设置目录和文件访问控制（对Oracle读允许），使能Oracle之外的应用访问权限，为初始存储分配足够的存储空间和文件预期的增长空间，并考虑到操作系统或硬件地址的限制，如每一个物理磁盘的最大文件尺寸、单个逻辑卷的容量或绝对的（无条件的）最大值。举一个例子，Solaris 2.2可以访问（通过它的I/O接口）的文件到达2GB，如同许多早期的UNIX实现的一样。但是用Solaris 2.3，“大”文件和文件系统被使能，其尺寸大于2GB（例如4GB或9GB，它和当时可获得的最大单个磁盘的尺寸相一致）。

为了帮助管理员在Oracle 8中访问BFILE，目录数据库对象被创建。这个目录对象是操作系统目录的一个别名或同义词，因此，目录对象是虚拟的。这些内部Oracle目录能被保证安全，通过Oracle内部机制（与操作系统安全相对），提供唯一真正安全的方式对BFILE进行访问。用户能被授予访问这些目录的特权，并且它们能够使用于PL/SQL或OCI。一个DBA或用户在Oracle 8上有CREATE ANY DIRECTORY特权，能够创建一个目录。然后，与任何其他普通数据库对象一样，特权被授予和收回。

在运行检查对BFILE的访问，可能发生的运行错误是以下几种：

- 不充足的用户特权。
- 目录不存在。
- 文件不存在。

当BFILE或目录被创建时，不检查正确的特权、目录名和文件名是否存在（在编译时）。

11.5 使用LOB和BFILE的例子

看一些LOB和BFILE的例子。

清单11-1用一个BLOB(photo)和一个CLOB(resume)创建一个表。CREATE语句产生了五个段：表自身，另两个LOB各有两个段。然后，一个很简单的PL/SQL块声明一个本地变量CLOB(r)和一个本地变量BLOB(p)，用来查询在数据库中相关的LOB对象的定位指针。

清单11-1 用CLOB和BLOB创建一个表

```
SQL> create table candidates (  
2 pk number(9),  
3 name varchar2(40),  
4 address varchar2(80),  
5 resume CLOB,  
6 photo BLOB );  
SQL> declare  
2 r CLOB;  
3 p BLOB;
```

```
4 Begin
5  SELECT resume into r from candidates where pk = 12345;
6  SELECT photo into p from candidates where pk = 12345;
7 End;
```

清单11-2用三个LOB、两个BLOB和一个CLOB创建一个表。大块的尺寸被设置为 32个块(你也可能想在init_ora初始化参数文件中设置db_file_multiblock_read_count为32)。pctversion被设置为30%。这样老的LOB版本能占有总的LOB段的30%。缺省的nocache和nologging在这里进行特别重申。随后是对LOB的索引语句,指定的范围大小比LOB实际使用的尺寸要小些(大约十分之一)。

清单11-2 用LOB、CLOB和BLOB创建一个表

```
SQL> create table landscape_plans (
2  front_yard BLOB,
3  back_yard BLOB,
4  contract CLOB,
5  plan_name varchar2(20),
6  plan_budget number(9)
7  LOB (front_yard, back_yard, contract)
8  store as (
9  storage (initial 1M next 1M pctincrease 0)
10 chunk 32
11 pctversion 30
12 nocache /* default */
13 nologging /* default */
14 index (
15 storage (initial 128K next 128K)
16 ) /* end LOB clause */
17 ); /* end create table */
```

清单11-3创建有两个BFILE列的表,一个用于存储员工的照片(photo),另一个存储指纹(thumbprint)。注意,你实际没有(或需要)任何附加的存储参数,因为BFILE实际上更多是在操作系统域中存储。

清单11-3 用BFILE创建表

```
SQL> create table background_check (
2  employee_id number(5),
3  hire_date date,
4  photo BFILE,
5  thumbprint BFILE );
```

清单11-4说明了怎样在操作系统级和在Oracle 8内为thumbprint和photo创建目录。

清单11-4 为LOB创建目录

```
# mkdir /user/data/bfiles/thumbprints /* UNIX or DOS */
# mkdir /user/data/bfiles/photos

SQL> create directory thumbprints as '/user/data/bfiles/thumbprints';

SQL> create directory photos as
'/user/data/bfiles/photos';
```

11.6 DBMS_LOB包

下面概括了DBMS_LOB过程和函数以及调用形式并对它们的功能进行描述。

1. APPEND (仅用于LOB)

示例语法：APPEND (d,s)

注释：在d后附加s。

2. COMPARE

示例语法：result:=COMPARE(b1, b2, n, o1, o2)

注释：比较b1与b2 (包括BFILE) 的n个字节，从b1偏移o1、b2偏移o2处开始。结果是0表示相等；如果b1小于b2，结果是-1；b1大于b2，结果是1。

3. COPY

示例语法：COPY (d,s,n,do,so)

注释：从d复制n个字节到s，从d偏移do、s偏移so处开始。

4. ERASE (仅用于LOB)

示例语法：ERASE (b,n,o)

注释：从b中擦除几个字节，从偏移量o处开始。

5. FILECLOSE (仅用于BFILE)

示例语法：FILECLOSE (b)

注释：关闭一个打开的BFILE b。

6. FILECLOSEALL (仅用于BFILE)

示例语法：FILECLOSEALL

注释：关闭当前所有打开的BFILE。

7. FILEEXISTS (仅用于BFILE)

示例语法：result:=FILEEXISTS(b)

注释：如果在操作系统中实际文件BFILE b存在返回1，否则返回0。

8. FILEGETNAME (仅用于BFILE)

示例语法：FILEGETNAME (b,d,f)

注释：给定一个BFILE b，返回目录d和文件名f。

9. FILEISOPEN (仅用于BFILE)

示例语法：FILEISOPEN (b)

注释：确定BFILE是否打开。

10. FILEOPEN (仅用于BFILE)

示例语法：FILEOPEN(b,m)

注释：以模式m (例如，file_readonly) 打开文件BFILE b。

11. LOADFROMFILE

示例语法：LOADFROMFILE (b,l,n,bo,lo)

注释：从BFILE b复制 n个字节到LOB l，从b偏移bo、l偏移lo处开始。

12. GETLENGTH

示例语法：length:=GETLENGTH(l)

注释：返回LOB l的长度。如果这个LOB是空的，返回长度将是0。

13. INSTR

示例语法：position:=INSTR(l,p,o,n)

注释：返回LOB l中p的位置，从偏移量o处开始查找p的第n次出现。

14. READ

示例语法：READ (l, n, o, b)

注释：从LOB l中读取n个字节的数据，从偏移量o处开始，返回读入缓冲区b中的字节数。当前返回缓冲区b是RAW或VARCHAR 2类型，仅能处理32K长度。

15. SUBSTR

示例语法：s := SUBSTR (l, n, o)

注释：返回从偏移o处开始LOB l中n个字节的子串s。与READ一样，返回值是 RAW或 VARCHAR2型，仅能达到32K。

16. TRIM (仅用于LOB)

示例语法：TRIM (l, n)

注释：从右边TRIM (减少) LOB l到长度n。

17. WRITE

示例语法：WRITE (l, n, o, b)

注释：从缓冲区b中读n个字节到LOB l中，从偏移量o处开始，与READ和SUBSTR相同，WRITE仅能从RAW或VARCHAR 2缓冲区b中写最多32K。

另外，除了这些，还有“构造器”类型函数创建空的 LOB。正如你想象的那样，EMPTY_CLOB () 和EMPTY_BLOB () 分别建立空的CLOB和空的BLOB。

11.7 更多的一些例子

清单11-5是插入一个空的BLOB和用一个已存在的BLOB值副本更新该BLOB列的例子。

清单11-5 插入和更新一个BLOB列

```
SQL> create table inventory (  
2 ID number(9),  
3 cost number(6),  
4 picture BLOB );  
  
SQL> insert into inventory values (  
2 123456789,  
3 5900, /* $59.00 */  
4 empty_blob() );  
  
SQL> update inventory  
2 set picture = (  
3 select picture from inventory  
4 where ID=999999999 )  
5 where ID=123456789;  
/* This copies the picture belonging  
to item 999999999 to our new item 123456789. */
```

注意这里没有要求任何特定的锁来完成这个复制。这是因为这里做的是全规模复制。但是使用DBMS_LOB例程来移动数据时，你需要用SELECT FOR UPDATE语句锁定你的行，正如清单11-6所示。清单11-7展示如何使用空LOB函数从一行中分离LOB的值。

清单11-6 使用LOB移动数据

```
SQL> create table memo_archive (  
2 ID number(9),
```



```
3 memo CLOB );

SQL> declare
buffer varchar2(80);
3 local_memo CLOB;
4 begin
5 buffer:="For Official Use Only";
6 select memo into local_memo
7 from memo_archive where ID=873561495
8 for update;
9 dbms_lob.write(local_memo,length(buffer),65536,buffer);
10 commit;
11 end;
12 /
```

清单11-7 使用空LOB函数

```
SQL> update memo_archive
2 set memo=empty_clob()
3 where ID=371620603;
```

这使LOB定位器原封不动，但简单地使LOB为空（设置它为“空值字符串”）。

11.8 最后的一些问题

BLOB和BFILE是以字节为单位衡量的，而 CLOB和NCLOB则以字符为单位衡量。仅对 CLOB来说，一个字节等于一个字符，但对于多字节字符集和 NCLOB这就不成立了，对于传给DBMS_LOB过程和函数的偏移量，这一点很重要。缺省的偏移量被设置为 1，意思是BLOB或BFILE的第一个字节和CLON或NCLOB的第一个字符。

有些问题来自于处理导出/导入。导入将CLOB和NCLOB从导出传储文件的字符集转换为目标数据库字符集（或国家字符集）。要确保你的 NLS参数设置正确。BFILE文件不被EXPORT导出，但能导出定位器和目录。导入之后，手动移动操作系统文件并重新将 BFILE文件关联到新的目录。

第12章 面向对象的特性

本章要点：

背景

面向对象技术

Oracle 8对象选项

REF属性

方法

集合——变长数组与嵌套表

对象视图

12.1 背景

面向对象的DBMS（OODBMS），也就是那些非主流的关系型（RDBMS）的DBMS，是面向对象程序语言（如 Actor和SmallTalk）的直接继承。永久存储的需要是促使 DBMS和RDBMS的出现来替代第三代语言（3GL）/基于文件的信息系统的最基本的原因之一，也促使OODBMS的产生来替代面向对象的编程语言信息系统。很有趣的是，喜爱面向对象系统的人通常认为，面向对象系统比关系数据模型的派生物（如实体关系图的变体）提供更有力的语义模型，并具有更高的层次。然而，与此同时，反对面向对象系统的人认为，由于它们关注3GL/基于文件的信息系统，它们的实现和语言要素偶尔会下降到十分低的层次（按 C/C++指针方式）。从本质上讲，它们采取导航的或过程化的查询，这也与前关系型系统（如基于COBOL的分层和网络DBMS）十分相似，而在基于SQL的RDBMS系统中的查询大多是描述性的和声明的。

不能不考虑关系型的厂商（如 Oracle）因此考虑 Oracle 8和它的当前关系-对象型的实现。当用任何RDBMS厂商的产品时，为了估计它的面向对象的能力，必须考虑几件事情，如下所示：

- 对面向对象概念和结构的信任。
- 面向对象到关系的映射的概念和结构。
- 仿真、完全或混合实现。
- 性能的降低或加强。
- 使用现有RDBMS子系统的能力。

希望RDBMS具有面向对象的接口的一个主要的原因是，这样做面向对象应用的程序就可以与RDBMS中的面向对象部分直接进行通信，这与在程序代码中必须动态地处理对象-关系型映射（分解和重组）相反。作为一个例子，考虑嵌入式SQL。这里SQL是基于集合的语言，必须与一个基于记录的主机3GL（如C）交互。这是不希望的情况，称为阻抗失配，可以通过诸如游标的结构和诸如预编译的软件（例如 Pro*C）来进行补救。一个基于面向对象的程序语言的应用与一个RDBMS的交互，情况与此相似。

Oracle 8特别地帮助缓解面向对象的开发和 RDBMS后端的情况，方法是提供一些强有力的内建的面向对象能力，如下所示：

- 关系作为数据类型。
- 继承。
- 集合作为数据类型，包括嵌套（容器）。
- 用户定义（可扩展的）数据类型。
- 改进大对象（LOB）。

和如用任何OODBMS或对象-关系DBMS（ORDBMS）一样，使用Oracle 8时，你能利用面向对象模型的优点。面向对象模型是比关系型模型更高层次的抽象。换句话说，你能抽象出事物的模型作为一个整体来比较接近地反映“真实的”世界。例如，在你的OODBMS中，可以具有一个称为CAR的复杂对象，它很可能是RDBMS中的几个表，如MAKE、MODEL、ENGINE、PARTS等等。面向对象模型与关系模型在“自然性”方面的注释是不同。我们来打个比方：

用面向对象技术，就像你在早上醒来穿上你的衣服，在夜晚你简单地脱掉它们以供下次再穿，十分自然。但是用关系型技术，你必须为你的衣服收集所有的纺织物，将它们缝在一起，然后穿上；在夜晚，必须把它们分开，按次序保存它们以供下次再缝和再穿。

尽管这个比喻不完全准确，却有些道理。基本思想是，用RDBMS进行工作时必须连结在前一次通过规范化分开的部分。关系设计者知道这种情况，这就是这类事情作为非规范化而存在的原因。

面向对象技术有哪些优点，特别是与关系型技术有关的并且是通常的优点呢？有些比较重要的已经讨论过了，而这里有一个列表，列出了大多数面向对象的主要强度：

- 建模：业务级抽象。
- 重用性：生命周期的代码重用可以在很大程度节约成本。
- 复杂性：没有数据类型限制；多媒体数据；嵌套。
- 可扩展性：反映建模的用户定义数据类型。
- 性能。

正如前面概括的，Oracle 8通过提供特定的能力如继承、集合、容器和用户定义数据模型满足了这些需要中的很大一部分。然而，性能是难以决定的，更多地依赖于整个数据库物理的和逻辑的设计。但是适当地应用分割并配合实验、迭代测试（基准程序）通常伴随性能调整效果，能够帮助你判断是Oracle 8使用面向对象还是只用Oracle 8能够更好地满足你的特定应用的性能需要。

12.2 面向对象技术

面向对象技术的一个主要问题是在面向对象团体中总是缺少一个标准定义，也就是什么是对象。相比之下，关系技术实质上对于什么是关系，在关系团体中从没有任何争论。

无论如何，Oracle 8定义对象作为对象类型的实例，一个对象类型作为基本面向对象模型的组成块通常直接指向真实世界的业务项目。因此，类似于表中的行，对象是数据，而对象类型是结构，与其他数据结构（例如记录）很相似。在许多其他面向对象语言和系统中，Oracle对象类型等于对象类。一个Oracle 8对象类型有两部分：

- 属性 —— 基本的数据类型或其他对象类型，有时称为对象类型的结构化部分。
- 方法 —— 组成在对象类型上允许的集合操作的 PL/SQL（或C）子程序；有时称为对象类型的行为部分。

多态性和继承

在面向对象技术中，其他的重要概念包括继承和多态性。继承是一个对象包括来自另一个对象类型的属性或方法的能力。方向是从一般到特殊，正像关系型模型中的子类型。一个例子是，你可能有一个雇员对象类型和一个经理对象类型（雇员类型的子类型）。所有雇员有名字、地址、社会保险号等等，还有雇佣日期、薪水、雇佣比例等等。所有前面这些代表一般的或普通的属性，能从雇员对象类型继承到经理对象类型。这意味着有两件事与效率有关：一、所有雇员（包括经理）的共同属性和方法只需要存储一次；二、经理对象类型仅需要存储它自己特有的属性和方法，如 `signature_authority` (属性) 或 `rate_an_employee` (方法)。因此继承可以基于属性（结构）或方法（行为）。

刚刚被描述的情况（经理对象模型从雇员对象中继承属性和方法）称之为单继承。有另一种类型的继承称之为多继承。单继承是一个子类最多有一个父类的形式；多继承允许一个更一般的模型：网络（或格子）模型，也就是子类型能够有多个父类，像在真实世界中的情况一样。与雇员到经理的单继承对比，考虑也包含官员的多继承方案：雇员→经理←官员。箭头特指继承方向，从一般（父）到特殊（子）。

多态性像继承一样可以是结构的或行为的。考虑一个没有经理对象类型的雇员对象类型。雇员只简单地是一个雇员，而不是经理。你如何从一个雇员类型获得经理类型呢？事实上，这是一个旧的程序结构，在不同的编程语言中有不同的术语，但在这里为了全面将它描述为变量记录。一个变量记录依赖于被输入的记录类型能够包含一些共同（继承）的数据，也能包含一些附加的域。这种情况对非规范化表是模棱两可的，可能同时包含经理和雇员。方法是怎么样的？与属性类似，方法是否可见依赖于被具体说明的对象（雇员或经理）。

刚才描述了常见的结构多态性是什么。那么行为多态性是什么？在面向对象中，当一种方法被调用时，称为向它发送了一条消息（进行执行）。当你向方法发送一条消息时，它可能有不同的执行，取决于它当前属于的对象类型或被影响的实例。例如，如果你调用一个 `hire` 方法，它构造（创建）一个新的职员或经理。尽管你调用的是同一个 `hire` 方法，这里实际上有构造一个实例（实例化一个对象）的两种不同实现：通过为雇员填入共同雇员属性，或通过为经理填入共同雇员属性和特定的经理属性。面向对象系统能够通过简单 IF/THEN 逻辑在单个程序内或通过定向化实现这个变化的程序，定向是 IF/THEN 逻辑加一个对所有可能程序之外的一个必要程序的调用。多态性也称之为重载。稍后捆绑（运行时捆绑）使多态性在现代语言和系统中成为可能。在面向对象中，每一个对象类型总是至少有一种方法——它自身的构造方法。

另外一个重要的概念是对象 ID (OID)，它表示一个在对象创建以后唯一代表该对象的统一标识符。这意味着它在所有的，甚至在 Oracle 之外的面向对象系统和语言中是唯一的。在 Oracle 8 中产生 OID 是 Oracle 公司的一个秘密，但是唯一性好像是以某种方式被链接到 CD-ROM 介质上。对那些熟悉本地网络的以太网的人来说，OID 应该是唯一的，就像以太网卡的以太地址一样，在工厂中被固化。ID 在它自身系统之外也是唯一的概念，与关系模型中

的主键和代理键这些东西是不同的。然而，它在某种程度上像一个分布式数据库中的主键，它们必须在多个系统中是唯一的。

你希望Oracle 8或任何ORDBMS中有的东西是什么？你可能希望一些东西，如组合（集成）多个关系表的对象视图。除了你已经包括的定制的（用户定义）数据外，你可能希望有一些预创建的、综合的、业务的数据类型可以服务于特定的适当范围的需要。客户端对象缓存也可能是需要的，因为它可以提供性能和访问的灵活性。你也不想失去 RDBMS的强项：一致性、并发性和安全性。

12.3 Oracle 8对象选项

那么，通过面向对象技术，Oracle 8给了你什么？Oracle在如下方面扩展它已有的复杂的多用途的RDBMS如下所示：

- 对象类型 —— 本质上是记录或类。
- 对象视图 —— 把你的许多规范化的表放在一起形成一个项目。
- 对象语言 —— 对Oracle SQL和PL/SQL语言的扩展。
- 对象API —— 通过Oracle预编译器（例如Pro*C）支持的对象。
- 对象可移植性 —— 通过对象类型翻译器（OTT），它可以建立Oracle 8对象类型到C++类的接口这样的东西。

但是，即使有了这些进步，Oracle 8 不支持多继承、多态性或在对象属性（如引用一致性）上的约束。

Oracle 8开放式类型系统（OTS）是一个所有Oracle 8对象类型的知识库，也是其他语言或系统中的外部对象类型的知识库。在开放式类型系统中，有一个数据类型层次，作为它的底层的是内建的Oracle 8数据类型。Oracle 8还增加了大对象（参考第11章“大对象的历史”），形式是BLOB、CLOB、NCLOB和BFILE，还增加了VARRAY（变量数组）和嵌套表形式的集合类型和REF（引用）形式的对象ID。另外，用户定义数据类型能够被创建在任何内建的数据类型和已有的用户定义数据类型上，唯一的例外是它们不能被创建在 LONG、LONG RAW、ROWID或%TYPE上。当你在Oracle 8中创建用户定义的对象类型后，可以将它用于多种用途，包括：

- 作为关系型表中的一列。
- 作为另一个对象类型的一个属性。
- 作为关系型表的对象视图的一部分。
- 作为对象表的基础。
- 作为PL/SQL变量的基础。

用于管理对象类型的扩展 Oracle SQL，包括以下这些：

- CREATE TYPE
- ALTER TYPE
- DROP TYPE
- GRANT/REVOKE TYPE

和表的列一样，属性在一个对象类型中必须唯一，但是在其他对象类型中该属性名可以被再用。与表中的列一样，你总能在一个对象中通过它的名字访问其属性。没有能力进行位

置或偏移量访问。任何给定的对象类型可以是简单的、复合的或者自我引用的。简单对象类型就只是它自身；复合对象类型至少包括一个其他对象类型；自我引用对象类型至少包含一个引用对象类型自身的属性。清单 12-1和清单12-2给出了两个创建和使用对象类型的例子。

清单12-1 创建一个雇员对象类型

```
SQL> create type employee_type as object (  
1  ssn number(9),  
2  name varchar2(35),  
3  address varchar2(70),  
4  resume CLOB );
```

清单12-2 使用一个对象类型作为关系型表中的一列

```
SQL> create table manager_candidates (  
1  position varchar2(40),  
2  vacancy CLOB,  
3  employee employee_type );  
  
SQL> insert into manager_candidates values (  
1  'Technical Manager',empty_clob(),  
2  employee_type(123456789,'John Doe','123 Maple St :  
3  Metropolis : NY : 12762',empty_clob() ) );  
  
SQL> select mc.employee.name from manager_candidates mc  
1 where mc.employee.ssn = 123456789;
```

注意你必须使用一个别名和点符号来查询一个表中的对象属性值。同时还要注意当你进行插入时，为命名为employee_type的对象类型调用命名为emplpy_type的构造器方法。所有的构造器具有与它们相关的对象类型相同的名字。清单 12-3给出了另外一个使用对象类型的例子，这次是创建一个对象表。

清单12-3 创建一个对象表

```
SQL> create table employee of employee_type;
```

创建这个表也为一个对象实例（行）创建一个包含对象ID（OID或REF）的列。这与创建一个关系型表，这个表在一个列中具有一个对象类型形成对照。在关系表中的对象类型没有OID。

12.4 REF属性

在Oracle 8对象世界中的关系与在Oracle关系世界中的关系实际没有什么不同：它指明对象之间如何相关。Oracle8仅支持一对一单向的关系，它可能足够也可能不够将你的业务模型化。当你声明一个对象类型中的属性是 REF时，它代表对其他一些对象类型的引用，这是一个最基本的关系。REF代表了一个对象的关系。一个 REF本质上是一个指针，或一个对象的“句柄”。你可以限制一个REF实际指引的范围，在SQL 3中包括范围的概念。SQL 3是对ANSI SQL标准的最新版本，它包含一些面向对象的特性。

像RMT（关系型模型2）中的代理键一样，Oracle 8中的对象引用（REF）是一个系统产生的值，在所有对象中（任何地方）是唯一的，指向一些永久不变的对象。一个 REF实际上引用一个对象——对象表中的一行（实例），而不是真正的对象类型。一个给定对象的 REF是

对象的OID+对象的表ID+对象表的数据库ID的组合。清单 12-4显示如何创建一个包含引用已创建的employee_type的REF属性的对象类型，并创建基于这一个对象类型的对象表，再插入一行（实例化一个对象）。

清单12-4

```
SQL> create type insured_employee_type as object (  
    1 control_number number(9),  
    2 contract CLOB,  
    3 employee_ref ref employee_type );  
  
SQL> create table insured_employee (  
    1 of insured_employee_type  
    2 scope for (employee_ref) is employee );  
  
SQL> insert into insured_employee  
    1 select control_sequence.nextval,  
    2 empty_clob(),  
    3 ref(e) from employee e  
    4 where e.ssn=123456789;
```

12.5 方法

一个对象类型可以有零个或多个成员方法。一个成员方法是一个能够操作任何对象类型（不仅仅是定义它的那个类型）的数据（也就是属性）的子程序。当然，方法简单说就是PL/SQL过程或函数。

正像使用任何PL/SQL封装的子程序一样，应为每一个方法创建一个接口（声明）和一个实现（体）。当访问一个方法时，应使用点表示法（对象.方法）。清单 12-5给出在一个对象类型内创建方法（声明和体）的例子，跟随其后的是调用该方法的一些 PL/SQL代码。

清单12-5

```
SQL> create type auto_type as object (  
    1 vin varchar2(80),  
    2 make varchar2(30),  
    3 model varchar2(40),  
    4 member function getcost  
    5 return number,  
    6 pragma restrict_references  
    7 (getcost, WNPS) );  
  
SQL> create type body auto_type (  
    1 member function getcost  
    2 return number is  
    3 begin  
    4 return (cost);  
    5 end; );  
  
/* create object table and insert some data */  
  
SQL> declare  
    1 a auto_type;  
    2 c number;  
    3 begin  
    4 select value(a)  
    5 into a
```

```
6  from auto a
7  where a.vin=1VWF673FB9093871AF21;
8  c:=a.getcost();
9  /* do something with c */
10 end;
```

注意函数使用pragma指令设置为WNPS。WNPS代表“写无包状态”，它的意思是函数被设置为不能修改自身内的任何参数（否则产生负面影响）。为了比较或排序对象类型，你能定义一个MAP或一个ORDED函数来比较两个对象。当使用这些比较函数时，你指定了对象分类语法（也就是，一个整理的序列），用于条件和分类。请参考 Oracle 8或Oracle 8i CD-ROM文献以获得更详细的实现信息。

12.6 集合——变量数组和嵌套表

集合是排序的或未排序的一组事物，变量数组是排序的，而嵌套表是未排序的。嵌套表是在最新的ANSI标准中，但变量数组是非标准的。任何主从（也就是一对多）关系能帮助对比这两种类型的集合。用某种顺序列出的主表项目能够使用变量数组（VARRAY），对每一个主表行的每行项目细节可以在变量数组中使用嵌套表。

警告 并行查询、并行DML、复制和分布数据库函数都不支持集合。

和许多数组一样，Oracle 8变量数组有一个隐式的顺序。每一个元素在数组中有一个从数组起始位置开始的偏移量，能够通过偏移量直接进行访问（也就是所谓的下标或索引）。变量数组是按行存储的（在表的行中），直到存储单元超过4KB（该点被存储在数据库之外）。和许多数组一样，每一个变量数组有数量和限制。数量是当前有多少元素存储在变量数组中，而限制是能够存储在变量数组中的最大元素数。你能通过在PL/SQL和3GL中使用下标访问单个元素，但是不能在简单的SQL中访问它们。

嵌套表能很好地适合主从（一对多）关系。嵌套表在Oracle 8中实际上是用户定义类型或表类型，能够在一个表作为一系列使用，在对象类型作为属性，或者作为PL/SQL变量。嵌套表被使用STORE AS语句存储在主表之外的存储表中。嵌套表有一些优点和缺点。优点包括能够对它们进行索引（与对象表相反）和不需要连结（与簇类似）。一个缺点是，尽管有级联删除形式（主行删除时将删除所有从属行），但是引用完整性约束是不可能的。清单12-6中给出一个创建和使用变量数组和嵌套表的例子。

清单12-6 创建和使用变量数组和嵌套表（表类型）

```
/* VARRAY */
SQL> create type tax_type as object (
    1 year date,
    2 taxes_paid number );
SQL> create type tax_array_type as VARRAY(3)
    1 of tax_type;
SQL> create type client_type as object (
    1 cid number,
    2 name varchar2(35),
    3 address varchar2(80),
    4 taxes tax_array_type );
```

```
SQL> create table clients of client_type;

/* nested TABLE */

SQL> create type child_type as object (
    1 child_id number,
    2 child_name varchar2(35) );

SQL> create type child_nest_type as TABLE
    1 of child_type;

SQL> create table parent (
    1 parent_id number,
    2 parent_name varchar2(35),
    3 parent_address varchar2(80),
    4 children child_nest_type )
    5 NESTED TABLE children store as child_nest_store;
```

总的来说，当你已在程序中使用数组时使用 VARRAY 集合，当你已在程序中使用一个记录（在一个记录里）时使用嵌套表集合。对这些程序数据结构来说，这些 Oracle 8 集合结构是模糊的。

12.7 对象视图

关系型表的对象视图提供给用户和开发者许多好处，如建立面向对象的前端应用到关系的后端存储之间的联系、协调多个面向对象的前端模式为一个关系型的后端模型和并行开发新的面向对象的应用和当前的关系型应用。后面这一项是对象视图方便地移植到面向对象技术的一种说法。对于用户，整个系统从表面上看似是基于面向对象的，但是实际上数据是关系型的。然而，这是透明的。当然，一般地说因为对象视图是视图的特例，它们有与一般视图相同的优点：为不同用户和不同目的提供相同数据的不同视图。正确地创建对象视图的一般顺序如下：

- 创建表。
- 创建对象类型。
- 创建对象视图。
- 创建 INSTEAD OF 触发器。

INSTEAD OF 触发器使你可以插入、更新或删除对象视图所基于的关系型表，而不是直接试图修改对象视图（它不能够这样做）。

清单12-7中给出一个创建对象视图的例子。

清单12-7 创建对象视图的步骤

```
SQL> create table depts (
    1 deptid number,
    2 deptbudget number );

SQL> create table branches (
    1 branchid number,
    2 branchbudget number,
    3 deptid number );

SQL> create type funding_type as object (
    1 deptid number,
    2 deptbudget number,
```

```
3 branchid number,  
4 branchbudget number );  
  
SQL> create or replace view funding_objv as  
  
1 select funding_type  
2 (deptid,deptbudget,branchid branchbudget) funding,  
3 from depts d, branches b  
4 where d.deptid = b.deptid;  
  
SQL> create to replace trigger funding_trig INSTEAD OF  
  
1 insert on funding_objv for each row  
2 begin  
3 insert into depts values  
4 (:new.funding.deptid, :new.funding.deptbudget);  
5 insert into branches values  
6 (:new.funding.branchid, :new.funding.branchbudget,  
7 :new.funding.deptid);  
8 end;  
9 /
```

第13章 Net 8 网络：新特性和概念

本章要点：

- 概述
- 连接管理器
- 其他新特性和概念
- 配置一些新特性
- 配置文件示例

Oracle 8重新命名SQL*Net 2.x为Net 8.x。伴随这一名字改变的是一组重要功能的加强，如果正确地配置它，数据库能产生更好的可伸缩性和性能。这些加强包括多路复用、连接池、集中、命名服务的改变、连接改变、安全改变和整个性能。管理也得到提高。

13.1 概述

Net 8是服务器反向兼容到SQL*Net 2.1，但不是客户向前兼容。也就是说，Net 8监听器能服务于Net 8和SQL*Net 2.x客户，但SQL*Net 2.x监听器仅能服务于同版本或低版本客户（不包括更高版本，因此不包括Net 8客户）。

可伸缩性通过多路复用、连接池、集中和增强命名服务的可用性得到增强。连接现在支持外来数据源和第二层的开放数据库连接（Open Database Connectivity, ODBC），ODBC支持存储过程调用。安全性是通过高级安全选项（Advanced Security Option, ASO）来增强的，ASO允许基于公共密钥技术X.509数字认证的单方签名，ASO还包括在客户和服务器之间基本的点到点（“通过线路”）加密。在客户和服务器之间用数字证书相互认证，提供单方签名证实，它通常被认为在多数情况下要优于其他外部认证技术。性能的增强是通过TNS Raw接口的使用、客户端名字缓存（类似于UNIX的DNS方式）和优化调度编码路径获得的。

可管理性的提高是通过提供本地原始设备制造商（OEM）和OEM applet集成支持得到的。对分布式数据库可以和本地数据库一样进行集中化配置。Oracle命名（Oracle Names）提供了增强的动态发现。Oracle 8的Net 8客户可以有配置简档，有一个Net 8向导可以用来配置。最后，可以为小规模的系统自动地提供一个缺省配置。

13.2 连接管理器

连接管理器（Connection Manager, CMAN）是一个新的网络特性，它仅能工作在多线程服务器（MTS）选项上。它集中来自多个客户请求，并将它们多路复用到一个物理通讯。除了集中以外，连接管理器能完成几种功能，包括担任防火墙，能够授权传输或在不同协议之间担任转换的网桥。

使用防火墙能力，CMAN能够通过基于的源（主机或客户）、目的主机和数据库（SID）的连接管理器创建客户端授权访问控制列表（ACL）和过滤（允许和拒绝）传输。

使用协议桥能力，CMAN有效地替代以前的多协议交换（MPL），允许主机（客户和服务

器)使用不同网络协议互相通讯。

多路复用包括合并多个逻辑会话到一个物理链接,在本质上,这个物理链接在多个会话中共享,它们当中没有一个会话在时间上会 100%地使用这个物理链接。第一个会话建立物理链接,然后该物理链接被以后的会话分时共享。当所有的会话被关闭时,物理链接就被关闭。

为什么多路复用是有益的?当使用 MTS选项时,多路复用允许比没有使用它时有更多的并发会话。并发会话的数量被操作系统限制,如对文件描述符、信号量、进程和套接字的操作系统最高限度。没有多线程服务器 MTS选择,一般只能有几百个(200~300)并发会话。有了它,几千个并发会话是可能的,也就是增加了单位时间总产量(也就是吞吐量)。正如所提到的,负载的可伸缩性是可能实现的,因为所有会话不总是 100%地忙。换句话说,它们在一些点处于空闲状态。它们在时间上的空闲重叠为并发提供了机会。因此,用长时间运行的 SQL 语句、复杂的 SQL 语句或者那些有高比例计算时间的 SQL 语句的应用将从 MTS和Net 8中多路复用获得最多的利益。因此,MTS选项和Net 8多路复用打破在高并发环境(操作系统和网络)中潜在的主要瓶颈。CMAN负责管理多路复用,用MTS选项提供它工作的基础。

与多路复用一样,连接池是用于有高比例的计算时间的应用有高的并发性的想法。计算时间可能包括查找、分类、格式化或其他 CPU密集型的活动,因此它也被称为“思考时间”。当有意地使用一个有限空闲时隙数少于到来请求数的交换机时,连接池工作得最好。一些端口被建立。基于UNIX网络,一个端口是硬件地址(例如,一主机 IP地址)和软件地址(例如一个UNIX套接字),连接池所做的就是去除空闲会话的开销。

连接和会话这两个术语在本文通常意思相同,尽管有些微妙的区别。不管这么多,连接是在机器间建立的网络连接(客户和服务器的),而会话是建立程序链接。

在任何事件中,连接池暂时断开空闲会话并将它们的连接传递到想要建立连接的新会话或想重新连接的以前建立的空闲会话。连接池确保尽可能多的非空闲会话并行活动。甚至会话数超过DBA指定的上限时,它仍然允许建立新会话。

13.3 其他新特性和概念

Net 8既支持两层(客户/服务器)又支持N层结构的分布式系统。在任何结构中,多个监听器能够被均衡装载。换句话说,多个会话能通过多个监听者被几乎相等地分布访问同一个或多个数据库。

当然,两层客户/服务器结构被旧版本的 SQL/*Net所支持。它由在一个固定的基于连接的协议(如TCP)上的同步远程过程调用(RPC)特性化。客户块进入挂起或等待状态,直到结果从服务器返回,如果服务器没有反应(例如被关闭),RPC失败。因为在传统的客户/服务器布局中,连接是固定的或专用的,如果RPC执行大量的计算,则网络利用率被浪费掉了。

用N层模型,异步消息队列能利用无连接协议,如UDP。客户端将它们的消息放置在队列中,然后既可用中止又可继续其他处理;它们不必等待,当服务器可用时(因此它从不失败),信息便被处理。当实时性需要很关键时,消息机制不是很好的方法,但是当高的吞吐量是重点并面临高的并发性时,这个模型是较好的。

TNS Raw是当可能时通过低层协议不使用Net 8头进行最基本通信的能力。基本的客户/服务器连接透明使用TNS Raw调用。换句话说,使用它的用户或DBA不管是客户端还是服务器端都没有配置设置的要求。当可能时TNS Raw自动地工作,例如当没有安全服务(也就是

ASO)要求时。使用TNS Raw, 有更大的包有效性(或密度), 即每一个包能传输更多的数据, 因为通常传输Net 8头的控制信息空间现在用于搬运数据。更进一步, 在同一会话中发送更少的总数据量。当TNS Raw被透明使用时, 两端Net 8上层的头处理层被绕过, 导致总的网络处理时间更少且总的响应时间更短。

13.3.1 新的安全能力

用高级安全选项(ASO), Net 8传输能够用你选择的加密算法(RSA或DNS)和密钥尺寸(位数)加密。CMAN支持ASO加密, 因此当协议桥存在时, 加密流不打破。ASO也提供通过基于标准协议X.509 数字证书的单个符号安全。数字证书是基于公开密钥基础设施(PKI)技术, 每个证书使用实体的私有密钥“签名”, 因此称之为数字签名。用PKI技术, 执行可以是单向(仅服务器)或双向(服务器和客户)的, 后者也称之为强认证。ASO也能提供外部认证适配器插入到Net 8中, 作为对数字认证的替代或补充, 这些附加的外部验证方法包括基于票卷技术(Kerberos, Sesame .DCE)、基于令牌(SecurID)和生物测定学方法(指纹、视网膜等)。

对于ASO有两个真正的部分。一部分ASO提供传输(Net8)加密替代它的前任——高级网络选项(ANO)和安全网络服务(SNS)。这部分ASO符合加密认证方法, 如数字证书, 替代了捆绑在Oracle 8的早期版本中的安全服务器。ASO与Oracle 8i同时发行, 尽管它如同Oracle 8i一样, 在写此书时仍然不完全可用。

13.3.2 新服务、工具和API

Oracle命名是一个对Net 8服务提供名字到地址解决方案的的目录服务。使用Net 8, Oracle命名得到加强, 或新特性都或多或少基于UNIX域名服务器(DNS)技术。

- 动态解决方案——Net 8服务可以向名字服务器自我注册。
- 客户发现——客户能在安装时查询名字服务器列表。
- 客户缓存——客户端名字的缓存能够被维护, 而不是为每次请求解决名字。

安装程序定位或请求名字服务器, 联系到名字服务器, 所有知道的名字服务器的列表被下载。客户询问在那个列表中的每一个服务器, 按相应时间排序结果(如果需要, 以后DBA可以重新分类)。该排序响应列表成为最初的客户端缓存(或本地名字缓存)存储在SDNS.ORA文件中。其他外部命名服务(甚至非数据库)如NIS、NDS和CDS能够被集成。例如本地主机名适配器能使用一个OS命名服务, 如Sun Microsystem的网络信息服务(NIS, 从前在UNIX中被称之为“黄页”)替代Oracle名字服务器。如果协议是TCP/IP, 每一个数据库有一个监听器, 不要求安全服务(也就是ASO)。

管理已经通过OEM applet被简化。OEM applet能被OEM使用或它们自己使用。尽管网络管理软件是过时的, Net 8助手取代它用于服务器端(监听器)配置。然而, Net 8在TCP/IP环境中自动配置自己。如果接受所有的缺省值, 监听器不要求listener.ora文件。用名字服务器, 客户资源环境文件被创建, 并使用客户环境资源编辑器编辑, 这个编辑器可以在多个用户(也就是客户)间共享。Net 8客户自动从包含环境资源的名字服务器装载客户环境资源文件。

Net 8向导自动初始化客户端配置。在安装时它被调用, 处理客户端初始化创建名字服务器缓存, 提供现有的或缺省的客户资源文件(从名字服务器)的选择, 并使能这些配置文件的编

辑。同时SQL * Net Easy Config被简单地重新命名为Net 8 Easy config，作为客户端配置工具。

Net 8能够通过Oracle Trace使用追踪，这有几个好处。Oracle Trace是Net 8的外部程序，因此比从前的SQL*Net追踪代码，有较小的可执行“足迹”。它能从点到点追踪，实际上从客户到服务器。追踪助手能从追踪文件中抽取或显示错误、提取包类型和客户发送的或服务器接受的信息以及维护包的统计。

最后，Net 8提供开放应用程序接口（API），称为Net 8 OPEN API，它允许可移植性。当从一个协议移动到另一个协议时，建立在Net 8网络上的应用能够很容易地移植，因为Net 8方法是协议独立的。适当的低层调用嵌入在Net 8自身内部，可以用适当的协议适配器库重新链接代替。但是高层Net 8调用嵌入在应用中，不能被改变。

13.4 配置一些新特性

在服务器端，和有一个lsnrctl监听控制程序和一个相关的listener.ora结构文件一样，有一个相似的cmctl连接管理器控制程序和一个相似的cman.ora结构文件。可是，在客户端相同的配置文件tnsnames.ora仍然只有一个将被修改。你可以在cmctl中完成许多相同的用于lsnrctl的命令，例如start、stop、status和version。除了cmctl之外，还有两个cmctl控制的进程：连接管理器网关（cmgv）和连接管理器系统管理器（cmadm）。cmgv与实际的监听进程相似。它是通信中心，主要负责所有连接集中职责以及相关的新特性职责。cmadm与Oracle名字服务器协作，主要负责地址信息和注册。当被用作start和status变元时，cmctl内在引用cmgv作为cm，cmada作为adm。cm是唯一的stop变元，因为当cmgv终止时cmada也自动终止。

通过编辑每一客户的tnsnames.ora文件和编辑init.ora文件使能集中（多路复用和连接共享池）和桥，通过编辑cman.ora文件建立防火墙，通过编辑init.ora文件使能监听器负载平衡，当然也可以通过编辑init.ora文件使能MTS选项。

13.4.1 多路复用和连接共享池

多路复用被建立在每个客户和服务器的上。假设cman.ora文件内容如下：

```
cman=(address=(protocol=tcp)(host=cman)(port=1610))
```

在这种情况下，每个客户的tnsnames.ora被编辑成如下这样：

```
mysid=(description=
  (address_list=
    (address=(protocol=tcp)(host=myserver)(port=1521))
    (address=(protocol=tcp)(host=cman)(port=1610))
    (connect_data=(sid=mysid))
    (source_route=yes)
  ))
```

在服务器上，你的init.ora中MTS节的下一行将被设置成如下这样：

```
MTS_DISPATCHERS="(PROTOCOL=TCP)(POOL=YES)(MULT=YES)"
```

当使能多路复用或连接共享池时，注意这个设置不同于SQL*Net 2.x。当不使用这些Net 8特性时，设置保持不变。注意应为整个协议启用这些集中特性并且不指定任何初始启动调度进程数。

当多路复用在每个客户端被启用和MTS选项被启用（在init.ora文件中）时，那么相关特性的连接共享池也可获得。连接共享池是更特定类型的多路复用。如果MTS没被启用，这两个特性都不能存在，它们都是连接集中的类型。

13.4.2 网桥

多协议支持也主要建立在每个客户端，每个客户端的 tnsnames.ora 将被编辑成如下这样：

```
mysid=(description=
  (address_list=
    (address=(protocol=spx)(service=cman))
    (address=(protocol=tcp)(host=myserver)(port=1521))
    (connect_data=(sid=mysid))
    (source_route=yes)
  ))
```

13.4.3 防火墙

过滤支持建立在服务器上，cman.ora 文件可能被编辑成这样：

```
cman_rules=(rules_list=
  (rule=
    (src=good_host2)(dst=myhost)(srv=mysid)(act=accept))
  (rule=
    (src=bad_host1)(dst=myhost)(srv=mysid)(act=reject))
)
```

实际的有或没有通配符的 IP 地址可以用于 SRC 和 DST 声明，例如（SRC=234.136.*.*）。

13.4.4 监听器负载均衡

要启用监听器负载均衡，在你的 init.ora 文件 MTS 部分，进行如下设置：

```
MTS_MULTIPLE_LISTENERS=TRUE
```

13.4.5 CMAN 环境资源文件

为了设置特定的 CMAN 运行参数，编辑 cman.ora 文件，包括下列 cman_profile 段的一部分或全部：

```
cman_profile=
  (parameter_list=          /* DEFAULTS      */
    (maximum_relays=512)    /* 8          */
    (log_level=1)          /* 0          */
    (tracing=yes)          /* NO         */
    (relay_statistics=yes)  /* NO         */
    (show_tns_info=yes)    /* NO         */
    (use_async_call=yes)   /* YES        */
    (authentication_level=1) /* 0          */
  )
```

表13-1简要概括对每个参数的描述。

表13-1 CMAN 运行参数

参 数	描 述
maximum_relays	并发连接的最大数（最大值是 1024）；与 init.ora 参数 SESSIONS 对照
log_level	0 是无日志，1~4 是递增的日志水平
tracing	是或不是；如果是，Oracle Trace 必须打开
relay_statistics	是或不是；如果是，在其他事件中记录 IN/OUT 字节和包的数目
show_tns_info	是或不是；如果是，并且 log_level 大于 0，TNS 事件被记录在日志文件里
use_async_call	是或不是；如果不是，调用将是同步的
authentication_level	0 或 1；如果是 1，拒绝不使用 ASO 安全服务的客户

13.5 配置文件示例

正如以前提到的，启用高级集中特性的多路复用和连接共享池，首先需要启用 MTS选项，意思是你必须重新配置 init.ora 文件。MTS选项和地址与 tnsnames.ora 和 listener.ora 文件的配置紧密结合。另外，CMAN的配置与 listener.ora 文件和所有客户的 tnsnames.ora 文件配置紧密结合（这些例子没有使用 Oracle 名字服务）。注意黑体部分，这是这些文件中与我们的讨论最相关的部分。MTS、监听器负载平衡、多路复用、连接共享池、网桥、防火墙和 CMAN 环境资源文件被启用。例子如下：

INIT.ORA

```
/* To enable MTS, multiplexing, connection pooling and listener load balancing */
/* This is just the MTS section of an init.ora file */
MTS_SERVICE=MYSID
MTS_SERVERS=10
MTS_MAX_SERVERS=100
MTS_DISPATCHERS="(protocol=tcp)(pool=yes)(mult=yes)"
MTS_DISPATCHERS="(protocol=spx)(pool=yes)(mult=yes)"
MTS_DISPATCHERS="(protocol=ipc)(pool=yes)(mult=yes)"
MTS_MAX_DISPATCHERS=100
MTS_LISTENER_ADDRESS=
"(address=(protocol=tcp)(host=myhost)(port=1521))"
MTS_LISTENER_ADDRESS=
"(address=(protocol=tcp)(host=cmán)(port=1610))"
MTS_LISTENER_ADDRESS=
"(address=(protocol=spx)(service=cmán))"
MTS_LISTENER_ADDRESS=
"(address=(protocol=ipc)(key=mysid))"
MTS_MULTIPLE_LISTENERS=TRUE
```

同时，你可能需要设置 SESSIONS 参数，它告诉实例启用集中特性（多路复用和连接共享池）打开的最大激活的会话数。

TNSNAMES.ORA

```
/* To enable multiplexing, connection pooling, and bridging */
```

```
mysid=(description=
  (address_list=
    (address=(protocol=tcp)(host=myhost)(port=1521))
    (address=(protocol=tcp)(host=cmán)(port=1610))
  )
  (address=(protocol=spx)(service=cmán))
  (address=(protocol=ipc)(key=mysid))
  (connect_data=(sid=mysid))
  (source_route=yes)
))
```

LISTENER.ORA

```
/* The foundation of all the addressing */
```

```
listener=(address_list=
  (address=(protocol=tcp)(host=myhost)(port=1521))
  (address=(protocol=tcp)(host=myhost)(port=1526))
  (address=(protocol=spx)(service=cmán))
  (address=(protocol=ipc)(key=mysid))
  sid_list_listener=
    (sid_list=
      (sid_desc=mysid)
      (sid_desc=another_sid))
    )
)
```

CMAN.ORA

```
/* To enable firewall and set the CMAN profile */
```

```
cman=(address_list=
(address=(protocol=spx)(service=cman))
(address=(protocol=tcp)(host=cman)(port=1610))
)

cman_rules=(rules_list=
(rule=
(src=good_host2)(dst=myhost)(srv=mysid)(act=accept))
(rule=
(src=bad_host1)(dst=myhost)(srv=mysid)(act=reject))
)

cman_profile=
(parameter_list=
(maximum_relays=512)
(log_level=1)
(tracing=yes)
(relay_statistics=yes)
(show_tns_info=yes)
(use_async_call=yes)
(authentication_level=1)
```

第14章 Oracle 8i 附加主题

本章要点：

- 新的ROWID
- 口令管理的增强
- 恢复管理器概念
- 高级队列概念
- 约束、国家语言支持和 sys安全性

在Oracle 8.x中有几个新特性或增强。这些包括所有的增强类型；然而，实现的增强主要包括在第20章中。本章主要包括的有新的 ROWID格式、新的口令管理功能、恢复管理器和高級队列；也简要地包含国家语言支持、约束和 sys安全性的改进。

14.1 新的ROWID

在Oracle 8.x中，新的ROWID格式已被改变，主要是为了适应分区和对象的新特性以及增加数据库限制（参见第18章），提供比Oracle 7.x更多的数据库文件和表空间。ROWID是Oracle 中所有存储寻址的基础，因此这个改进很重要。

ROWID被用做在规则的Oracle 索引（B-树）中唯一识别一行，并在整个Oracle 8的内核中使用。在Oracle 7.x中，ROWID包括文件号、块号和行号。它仅用一个号一致性地识别文件。换句话说，在Oracle 7.x中文件号是绝对的。在Oracle 8.x中，文件号是相对的。一个Oracle 8.x数据文件的文件号由两部分组成：

- 绝对部分在数据库内（和在Oracle 7.x中一样）唯一，由dba_data_files表的FILE_NO列代表。
- 相对部分在表空间中唯一，由dba_files表中的RELATIVE_FNO代表。

因为文件号对表空间是相对的，新的ROWID也必须存储段的一些指示。在Oracle 8.x中，这是用来识别表或分区的数据对象号。通过确定这个段，然后Oracle 8能确定包含的表空间，和使用dba_tablespaces一样。

能够确定数据对象号，例如从dba_objects的DATA_OBJECT_ID列中确定。这个数据对象号不是对象标识符（参见第10章），它被包含在每一块中，并且是唯一的。当一个表被截断或分区被移走时，数据对象号的版本增长。使用这种方法，Oracle 8能验证在块中的号匹配（也就是，是正确版本），也能够用它与回滚段中的块做比较来保证版本的准确性。

新的ROWID格式使用基于64位的编码，它的字符宽度是18。这就意味着在任何给定的Oracle 8数据库中有 64^{18} 个可能的行。新的Oracle 8.x ROWID组成见表14-1。

表14-1 新的ROWID组成

数据对象号	相对文件号	块号	位置号
000000	FFF	BBBBBB	SSS

从左到右，前六个字符代表数据对象号，接下来3个字符代表在表空间中的相对文件号，

再接下来的6个字符代表在一个文件中的块号，最后的3个字符代表块中的位置段（行）号。与Oracle 7.x对比，真正的不同之处是数据对象号和相对文件号。

在表14-2中，列出了Oracle 7.x ROWID的格式。

表14-2 旧的ROWID组成

块 号	行 号	文 件 号
BBBBBBBB.	RRRR.	FFFF

从左到右，前8个字符代表在文件中的块号，接下的4个字符代表块中行号，最后4个字符是绝对文件号。除了在分区表中涉及到全局索引（因为分区能跨多个分区表空间）外，这种类型的ROWID在多数情况下是足够的。

如何能够概括在Oracle 7.x的ROWID和Oracle 8.x新的ROWID之间的不同呢？表14-3会对你有些帮助。

表14-3 ROWID版本比较

版 本	描 述	字 节 数	显 示	文 件 号
7.x	限制	6	2点	绝对
8.x	扩展	10	无点	相对

相对表空间寻址是 Oracle VLDB支持的基础。Oracle 8使用10个字节来存储这个新的ROWID，而Oracle 7.x使用6个字节。当相对ROWID已足够时，限制ROWID被使用。当绝对ROWID被要求（如在分区表中用全局索引）时，扩展（完全）ROWID被使用。对于无分区表上的无分区索引、在分区表上的相等分区索引、跨块的链接移植指针，使用限制ROWID已足够。扩展ROWID用于分区表上的全局索引、内核的使用和存储ROWID的格式。

当从Oracle 8获取Oracle 7 ROWID时，ROWID以限制的格式出现，如同它们在Oracle 7.x中一样。但是，当从Oracle 7.x获取Oracle 8.x ROWID时，你需要使用DBMS_ROWID包来解释扩展格式。Oracle 7的表可以被导出到Oracle 8中，但是包含有新ROWID格式的Oracle 8.x表不能被导出到Oracle 7中。更进一步，如果ROWID被以任何存储格式保持，在它们导入之后，需要重新计算，因为它们的内容将是过时的，需要更新。

只在应用是部分存储或获取ROWID行时，发生应用可移植性问题。另一方面，在整体上，使用ROWID的这些应用将受影响。当Oracle 7表被导出或被移植（使用移植工具）到Oracle 8时，存储ROWID的列的宽度自动地变宽以适应新的比较宽的Oracle 8.x ROWID。前述的DBMS_ROWID包由dbmsutil.sql脚本创建，被catproc.sql脚本调用。表14-4列出了DBMS_ROWID包的函数和返回值。清单14-1显示了从限制ROWID转换到扩展ROWID的例子。

表14-4 DBMS_ROWID 包的函数

功 能	返 回
rowid_create	一个新的ROWID
rowid_info	类型和组成
rowid_type	类型（0=限制，1=扩展）
rowid_object	数据对象号（段）
rowid_relative_fno	相对文件号
rowid_block_number	块号
rowid_row_number	行（位置）号

(续)

功 能	返 回
rowid_to_absolute_fno	绝对文件号
rowid_to_extended	扩展ROWID
rowid_to_restricted	限制ROWID
rowid_verify	0=可以被扩展, 1=不能被扩展

清单14-1 从限制ROWID转换到扩展ROWID的例子

```
SQL> select * from mytable where mykey=12;
mykey myROWID          myvalue
-----
12 00000001A.0011.0009      102
SQL> update mytable set myrowid=
1 dbms_rowid.rowid_to_extended(myrowid, 'JOHN', 'MYTABLE', 0)
2 where mykey=12;
SQL> select * from mytable where mykey=12;
mykey myROWID          myvalue
-----
12 AAAAbCAACAAAAAbAAAA      102
```

rowid_to_extended函数接收旧的（限制的）ROWID、拥有者、表和转换类型（conversion_type）。这个转换类型既可是0用于存储（内部）转换，又可是1用于显示（外部）转换。它返回一个新的（扩展）ROWID。

14.2 口令管理的增强

在Oracle 8中提供的口令管理的增强包括帐户锁定、时效和到期、历史机制和复杂的验证（激活检查）。为了启用口令管理，运行utlpwdmg.sql脚本。你能建立环境资源文件并将用户指定给它们。然而，它不能够像在init.ora文件中那样用resource_limit参数打开和关闭对资源的限制，口令限制总是被强迫执行。

用帐户锁定，在指定数量的登录失败后，企图再次登录时，Oracle 8自动锁定帐户。可以基于用户或组设定这个数。在超过一定的时间间隔或者手动解锁之后，帐户锁定被打开。可以根据需要手动锁定，在这种情况下，只能使用手动解锁（不是通过一个时限解除封锁）。

使用口令时效和到期，你能指定一个截止周期或活动时间，它对应于口令保持多长时间的有效性。你也能指定一个宽限期来使用户有时间在口令到期时改变它的口令。这些特性可以被建立在用户或组上。例如A组可能有90天期限加上5天的宽限，而B组可能有30天期限再加上1天宽限，另外用户C可能有单独的期限和宽限，分别是60天和10天。

使用口令历史机制，一个用户不能再用过去的时间间隔中使用过的任何口令，这个时间间隔被指定在该用户的环境资源文件中。另外，要记住的重要的一点是，和资源限制不一样，它和其他口令限制总是被强迫执行（除非被修改）。

激活口令检查是通过一个缺省的SYS PL/SQL函数提供的。你能用你自己的增加改进的PL/SQL函数替换它，缺省的函数提供复杂的验证来检查每一个口令，如下：

- 1) 至少4个字符长度。
- 2) 不等于用户ID。
- 3) 至少有以下每种字符中的一个：字母、数字、标点符号。

4) 至少有3个字符不同于老口令。

你自己的口令检查函数在 SYS图表中创建，依附在用户或组的环境资源文件中，并且必须附加如下声明：

```
myfunction (
  p_userid IN varchar2(30),
  p_new_password IN varchar2(30),
  p_old_password IN varchar2(30) )
return boolean;
```

CREATE PROFILE 和 ALTER PROFILE 语句处理所有前述的口令限制的设置，对 CREATE PROFILE 语句的缺省口令设置如表 14-5 所示。

表14-5 CREATE PROFILE的缺省口令设置

设 置	缺 省 值	单 位
failed_login_attempts	3	次
password_life_time	60	天
password_reuse_time	180	天
password_reuse_max	无限制	个
password_lock_time	1	个
password_grace_time	10	天
password_verify_function	缺省	函数

这些设置中的大部分已经在前面的段落中解释过，现在来讨论一些可能不太清楚的地方。password_reuse_max 代表在一个口令能够再被使用之前，改变口令的最大个数。缺省值是无限制，但是口令再使用还受 password_reuse_time 控制。这两个参数都能被设置为无限制；然而实际上仅有一个能被设置为数值。在这方面，它们之间的设置是互斥的。

password_lock_time 指定在特定次数的企图登录失败之后，用户帐户被锁定的天数。用缺省的设置，如果用户三次登录失败以后，用户帐户将被锁定一天。最后，缺省的 password_verify_function 的实际名字是简单的 verify_function。

清单 14-2 给出一个例子，这个例子创建一个环境资源文件，设置口令的限制，然后创建一个用户以拥有这个环境资源文件。

清单14-2 为用户创建口令环境资源文件的例子

```
SQL> create profile secure90 limit
  2 password_life_time 90
  3 password_reuse_time 366
  4 password_grace_time 5;
SQL> create user john
  2 identified by all4one?
  3 default tablespace scratch
  4 temporary tablespace temp2
  5 quota 10M on scratch
  6 password expire
  7 profile secure90;
```

注意，CREATE USER 语句在为新用户帐户 john 预定截止日期之前，CREATE USER 和 ALTER USER 语句将你的定制的环境资源文件用于一个特定的用户。你也可以用组来替代这个用户。还有 ALTER USER 语句被用来手动加锁和解除一个帐户的锁，如下所示：

```
SQL> alter user john
  2 account lock | unlock;
```

DBA_USERS和USER_USERS包含反映这些新的口令管理特性的新列，如表 14-6所示。

表14-6 新的用户口令管理特性

新 列	意 义
account_status	锁定、截止或打开
grace_data	改变的日期+password_grace_time
lock_data	锁定的日期
expiry_data	创建的日期+password_life_time

DBA_PROFILES还列出了口令设置，并包含一个新的 resource_type列，指明内核或口令。一个新的user_password_limits视图包含resource_name和limit列。

14.3 恢复管理器概念

恢复管理器（RMAN）是对Oracle 8备份、恢复和重建进行管理的一段软件。RMAN可以是独立的或分布的。RMAN对Oracle 8服务器使用PL/SQL接口，它调用三个主要函数：BACKUP、RECOVER和RESTORE。RMAN使用一个称为恢复目录的知识库。在 8.0.4中，第三方存储子系统如EMC/Epoch、Legato、IBM等等能够被集成到 RMAN解决方案中。例如，磁带驱动器以自动方式使用，能够以并行方式使用诸如那些来自 Exabyte的磁带单元。

RMAN结构的四个基本块是 RMAN接口（一个OEM applet图形接口或命令行解释器，它能在OEM 1.4或更高版本中获得）、BACKUP/RECOVER/RESTORE服务器函数、恢复目录和存储子系统（操作系统的、第三方的或两者都是）。

为什么要使用RMAN管理你的Oracle 8备份？它能自动执行许多管理功能和职责。备份能发生在多级上：数据库、表空间或者数据文件级。数据文件能够被特指为基于用户定义容限的备份。最重要的是，真正的增量备份是可能的；换句话说，在任何级别的备份能够执行只对已改变的块备份；更进一步说，未被使用的块不被备份。备份能够很容易地建立时间表和控制。存储子系统处理更容易管理，它的操作可能被并行化。另外，RMAN能够与已经建立起来的第三方厂商硬件或软件集成。

RMAN有两个主接口：命令行解释（CLI）接口或OEM图形用户接口（GUI）。CLI接口是一个与SQL*Plus相似的应用，能够用交互或非交互的方式执行。GUI在OEM中被集成，提供能够与第三方应用（例如存储子系统）相链接的API库。

14.3.1 恢复目录

恢复目录由catrman.sql脚本创建，通常在专用数据库中对它进行存储，它十分像 OEM知识库。恢复数据库是存放恢复目录的数据库，目标数据库是任何 RMAN要备份的目标数据库的应用或用户数据库。恢复目录必须和你的其他数据库一起备份。为备份恢复数据库，你必须反转它和一个目标数据库的角色，一个目标数据库于是成为恢复数据库的备份数据库，它是暂时的目标数据库，另一些目标数据库是恢复数据库，这意味着它至少有两个恢复目录——一个是为所有用户数据库的，另外至少有一个为恢复目录自身的目录。

提示 你应把主要的RMAN知识库放入分开的（恢复）数据库中，并放在与你的核心应用（目标）数据库分开的机器上。另外你能只为处理重要的 RMAN仓库在一个应用数据库中创建一个RMAN知识库。用这种方法，你能失去任何数据库并能恢复它，包括

恢复数据库自身。

恢复目录维护在它的数据库仓库中的信息，这些信息包括数据文件备份设置、归档日志备份设置、备份块、数据库文件拷贝、归档重做日志、归档重做日志拷贝、目标数据库的控制文件信息（也就是它们的物理结构）和通常可获得的备份命令。

RMAN静态地存储关于所有它的目标数据库的控制文件信息。当备份操作发生时，RMAN与需要备份的服务于目标数据库上的 Oracle 8服务器进行通信，在它的核心里已经有了备份功能。RMAN初始化和管理的备份，但是实际上每一个目标数据库备份它自己。因此，当任何目标数据库结构改变时（例如，一个数据文件被增加或重命名），在下一个备份操作发生之前，除了必须从数据库的数据字典中进行查询（这意味着目标数据库必须打开）的回滚段消息外，几乎所有对更新恢复目录必要的信息都能在目标数据库的控制文件中找到。

然而RMAN能够在没有恢复目录时进行操作，当你仅有一个或少量几个备份数据库，或者表空间按时间点恢复（PTR）不是本质的时，这是适合的。为什么你不能有 PTR？回滚段信息应该存储在恢复目录中，它被从目标数据库中查询，但是没有恢复目录，这儿可以没有 PTR。对无恢复目录操作的其他限制包括不能使用存储脚本（它应被存储在目录中）并且当控制文件被丢失或毁坏时自动恢复不一定可行（只是部分如此）。

14.3.2 RMAN命令和功能

恢复管理器依赖于（目标）服务器备份功能，这个功能驻留在包括所有必须的备份、恢复和复原子程序的数据库包中。RMAN的前期产品，External（或Enterprise）Backup Utility（EBU）仅能调用第三方工具。集成是最高程序层。现在 RMAN能通过API功能控制操作系统（OS）和第三方存储子系统。集成是最低的功能层。

RMAN命令包括注册一个目标数据库、重置目标数据库的状态（甚至对以前备份过的版本）、当目标数据库改变时更新恢复目录（甚至从前备份过的控制文件的状态）的能力。目录同步是周期性更新目标数据库信息：所有控制文件和回滚信息。这是重要的，不仅因为可能有结构性变化，而且因为日志文件现在记录在控制文件里。在 Oracle 8.x中，控制文件改进为追踪日志文件转换和归档重做日志文件拷贝。因此同步变得十分重要。

RMAN提供直接使用 BACKUP、RECOVER和RESTORE命令或在脚本中嵌入的能力。RMAN脚本被RUN命令调用。CREATE SCRIPT命令被用来创建脚本（把它存储在恢复目录中）。为了安全起见，每个你备份的脚本都保存一个操作系统拷贝（例如用于丢失恢复目录的情况）。REPORT和LIST命令提供恢复目录信息和RMAN状态的日志输出，REPORT报告基于用户容量相关的冗余、增量或时间可能需要备份文件的数据库文件；备份文件可能被删除备份文件不能恢复。LIST列出关于在备份集或复制中用户特定数据文件的信息，部分或全部匹配查找准则。

14.3.3 RMAN术语

备份集或者是数据文件（可能还包括控制文件）或者是归档日志的集合。换句话说，多个文件被写入单个备份集中。它与单个文件简单地被复制为镜像是相对地。在备份集中被备份的文件在能够被使用之前必须被恢复（和提取）。而镜像能够直接被使用，只是简单地重新命名为丢失的数据文件。备份集支持完全或增量备份。

并行化能够被指定以匹配可获得磁带驱动器数。备份集能够从磁盘到磁带进行，在恢复

时相反执行。数据文件能够被保存在包括多个磁带的备份（多磁带卷）之中，并被多路复用，在每个数据文件中有用户指定的块数。控制文件块首先被完全写入，如果被包括，它不能在使用数据文件块时交织使用。备份集能够被写入磁带或磁盘中。

备份块是属于备份集的单个文件。这个文件能包含来自多个数据文件的块并能够被指明作为操作系统文件或磁带卷。备份集是它的所有备份块之和。

镜像复制仅只能够写入到磁盘，它与源文件形成一对一的对应。如果所有的镜像文件的总和包含你的整个数据库，那么你有一个完全的冷备份。你也能注册外来备份，那也就是说，这些镜像文件可以不是 RMAN 创建的。

完全备份包含要备份的所有特定的数据块。增量备份仅包含自上次备份以来被修改的块。累加增量备份是可能的。但是它们浪费空间和时间，因此它们的优点被量的积累所消除。你也可以做一个完全备份作为基准，它被所有随后的增量所忽略。多级增量就像 UNIX ufs 备份规划。0级代表完全备份，你能指定最高 7级的增量。例如，2级可能代表周增量，1级代表日增量。用这种规划，增量仅备份自上次同级或更低级备份以来的增量。例如，2级备份仅备份自上次2级、1级或0级备份以来的改变。多级增量增强了恢复，因为你从任何给定的级仅需要恢复最多一个增量。

标识符是能够被指定到备份集或镜像文件的字符串（最多 30个字符）。标识符是被存储子系统用来帮助用户记忆一些存储单元（例如磁带卷或磁盘分区）的符号名。它的用途与 RMAN没有区别。标识符能够跨越备份集或镜像文件。标识符不必对每个备份集或镜像文件唯一。这能有助于分组某些备份集，在感觉上组成一个整体确定的备份集，例如一个表空间。

备份和恢复的并行化操作由 RMAN在内部处理，它建立多个并行用户会话，每个设备一个会话。RMAN在用户会话内多路复用独立命令的发生。因此尽管在 RMAN 内语句执行是串行化的（如SQL*Plus），通过虚拟的多用户会话可以实现并行化。尽管一个用户能够指定分组，当多路复用它们到备份集时，数据文件被 RMAN分组。除SYSTEM表空间外，RMAN支持任何表空间的PTR。为了实现后者，Oracle要求数据库仅用技术支持助手复制和操作。

备份本质上要求数据库被打开（以获得回滚信息），可是恢复仅要求实例被启动。备份和恢复都检查记录和报告块的毁坏。它们也为破碎块探测进行校验和。破碎的块是当它被修改时，正好赶上被备份操作读取，因此它必须被重读来得到一致性版本。

在Oracle 8.x中，控制文件随日志转换和归档文件拷贝增长。可是这个增长能够通过设置 init.ora文件中的参数 control_life_record_keep_time进行控制，这个参数被表示保持最旧的可重用条目或记录的天数，而 Oracle 7.x没有可重用条目。如果你设置这个参数为 0，无论你何时需要，条目都可再用，它们无时间限制。

所有保存RMAN备份和恢复信息的数据字典视图都以 V\$BACKUP_*作为前缀，如下所示：

```
V$BACKUP_CORRUPTION
V$BACKUP_DATAFILE
V$BACKUP_REDOLOG
V$BACKUP_SET
V$BACKUP_PIECE
V$BACKUP_MEMBER
V$BACKUP_RESTORE_STAT
V$BACKUP_PARMS
```

查看这些视图和V\$BACKUP_LOG以及V\$DATAFILE可监控RMAN的活动。

14.4 高级队列概念

在分布式环境中有两种主要类型的执行：立即（同步）或延时（异步）。在大多数情况下，立即执行是在许多分布式系统中被采用的。但是在其他一些情况下，例如基于复制的分布式系统，延时执行是适当的。队列是延时执行机制。队列是先进先出（FIFO）或先来先服务（FCFS）数据结构，首次到来的事情是首先得到服务并退出。可是，存在纯队列的变体，它们全都或多或少地基于优先级这一概念。在队列中的一个任务单元称之为一个工作。一个工作在队列中等待被服务或被处理。即使系统失败，持续队列也能保证在队列中的所有工作将在系统成功重新启动后被处理。工作控制和任务流管理是两个很适合队列化的领域。事务处理（TP）监控器和面向消息的中间软件（MOM）依靠或者模仿消息队列功能。Oracle 8.x提供高级队列（AQ）能力而无需第三方软件。

消息是能被处理的最小工作。和网络包一样，它包括用户数据和控制信息。单个事务能创建（产生）或处理（消耗）多个消息。ENQUEUE创建消息，DEQUEUE使消息被处理。消息队列是消息的队列或消息的集合。由于不同的原因，多消息队列能够被创建，例如从功能上分开一些消息任务、分开独立的消息活动和可能增加进程的吞吐量。消息表是用来存放多消息队列的Oracle表。

Oracle 8.x高级队列的一些比较重要的特性包括如下：

- 消息排列和优先级。
- 对象作为用户数据。
- 响应和异常队列。
- 事务性和非事务性消息。
- 一个消息的多个接收者。
- 消息分组。

Oracle高级队列的许多用户功能管理功能是通过系统包 dbms_aq和dbmsaqadm提供的。dbms_aq包含两个基本过程enqueue和dequeue。用户必须被授予aq_user_role。消息的payload是它的用户数据。看一看enqueue和dequeue声明。

```
enqueue (
queue_name      IN      varchar2,
enqueue_options IN      enqueue_option_t,
message-properties IN    message_properties_t,
payload         IN      <object_type | raw>,
msgid          OUT     raw )

dequeue (
queue_name      IN      varchar2,
dequeue_options IN      dequeue_option_t,
message-properties OUT    message_properties_t,
payload         OUT     <object_type | raw>,
msgid          OUT     raw )
```

因为payload能被指定为一对象类型，它从本质上讲可以是任何对象类型，它的范围很宽（参考第10章）。enqueue_options、dequeue_options和message_properties全部都是预定义的记录类型。queue_name唯一地识别一个队列。msgid唯一地识别一个消息。请参考 Oracle 8.x文献，那里包括所有enqueue和dequeue选项的详细说明。看一个简单enqueue和dequeue调用：


```
dbms_aq.enqueue (
queue_name => 'john.jq1',
payload => employee_type
msgid => msgh );

dbms_aq.dequeue (
queue_name => 'john.jq1',
payload => employee_type
msgid => msgh );
```

用这个enqueue调用，一个消息有msgid、msgsh和指定为employee_type对象的page load，这个消息被放进 john拥有的队列表 jq1中。后续的 dequeue调用处理同一个消息。所有 enqueue_options、dequeue_options和message_properties缺省值被采用。同时，注意赋值操作符=>的使用。

提示 当任何消息有基于时间相关的属性（例如延时、截止或保持间隔）时，要求有一个AQ计时器进程或时间管理器（tm）。设置init.ora参数AQ_TM_PROCESSES为1，这将启动一个时间管理器进程。当前，它仅能被置为1；但是在未来的版本中，它可能增加为使能多个时间管理器。

DBMS_AQADM包含用于管理创建和控制队列和队列表以及管理订阅者和启动 /停止时间管理器进程的子程序，你必须要有 AQ_ADMINISTRATOR_ROLE以执行以下这些过程：

```
CREATE_ | DROP_ QUEUE_TABLE
CREATE_ | DROP_ | ALTER_ | START_ | STOP_ QUEUE
START_ | STOP_ TIME_MANAGER
ADD_ | REMOVE_ | QUEUE_ SUBSCRIBER(S)
```

SYS必须首先执行 grant_type_access过程以给AQ管理者对AQ对象类型的访问权。当需要时AQ管理者能更进一步授权给用户，看一看简单创建和删除队列表的例子：

```
dbms_adqam.create_queue_table (
queue_table => 'jq1',
queue_payload => employee_type );

dbms_adqam.create_queue_table (
queue_table => 'jq1');
```

采用缺省值时，这些语句可以十分简单。但是和创建队列处理队列操作一样，在 DBM_AQADM过程的执行中可以指定许多选项，特别是处理产生的过程。作为一个例子，一个最后关键概念涉及订阅者。一个队列可以允许每个消息有多个接收者（这是队列表创建时的一个选项），可以处理来自这种队列的消息的用户称为订阅者（subscriber）。DBMS_AQ用户和DBMS_AQAD管理包的所有可用选项的细节已超出这本书的范围。但是可以说 Oracle在它的数据库引擎中集成了很强壮、也复杂的队列机制。

DBA_QUEUE和USER_QUEUE表包含表队列信息，DBA_QUEUE和USER_QUEUE包含队列信息。AQ\$<queue_table>视图对于每个队列表可用。

14.5 约束、国家语言支持和SYS安全性

在Oracle 8.x中约束检查能够被拖延直到事务结束前，也就是服务器只在提交时检查约束是否被满足。如果约束在事务期间被违反，这是没有问题的。如果在提交时违反，整个事务被回滚。

与之相对的是立即约束检查，即服务器在每个语句结尾检查约束。如果语句违反约束，

这条语句被回滚，不是整个事务。

还有，替代使约束无效或有效，它们能够被强迫。一个强迫约束强加于所有新修改语句，但是忽略当前已存在的数据（它实际上可能违反约束）。尽管可能不是一个好主意，Oracle 8.x现在可以使用非唯一索引使唯一性约束和主键约束可以被处理。

用于国家语言支持（NLS）的NCHAR列是新的。它是一个定长字符、多字符数据类型，能容纳最多2000个字符。NVARCHAR 2与varchar2的关系就与同NCHAR与char的关系一样。但是NVARCHAR 2能容纳最多4000个字符。最后，为了存储和提取 NCHAR和NVARCHAR2数据类型，实际值用字符N作为前缀，不用引号。

在Oracle 8.x中，除非你属于 OSDBA组或你有 SYS口令/内部口令，否则如果没有 SYSDBA身份，你将不能链接到 SYS图表中。这些有使用 ANY限定（例如 SELECT ANY TABLE）的系统权限的用户，在Oracle 8.x将不能像在Oracle 7.x中那样有权访问SYS图表。在Oracle 8.x中必须有SELECT_CATALOG_ROLE或EXECUTE_CATALOG_ROLE才能选择或执行SYS对象。

第15章 Oracle 8i概述

本章要点：

在Oracle 8i中的 Java能力

因特网文件系统

Oracle interMedia

Oracle WebDB

Oracle 8i的可用性和可恢复性

安全性

其他特性

Oracle RDBMS的最新版本Oracle 8i是在企业内和因特网上开发、布署和管理应用高移动性和可伸缩性的数据库。它被设计成用来支持很大和高容量的事务处理和数据仓库应用。

Oracle 8i最主要的新特性是在数据库中包括 Java和因特网能力。与数据库服务器集成的是称为Oracle JSever的Java虚拟机 (JVM)，因此开发者现在能够用 Java、PL/SQL、C或OCI创建应用。它使开发者能够用Java编写存储过程、数据库函数或触发器。因为Java是平台独立的，因此用它开发的应用能够很容易地部署在任何支持 Java的服务器中，而无需重写和再编译。Oracle 8i的另一个主要特性是因特网文件系统 (iFS)，它使数据库能存储和管理关系和非关系数据文件。Oracle 8i还包括能够管理和访问多媒体（视频、音频、图像、文本、空间）的Oracle interMedia。Oracle 8i还包括Oracle WebDB，这个Web开发环境使开发者能建立使用标准的Web浏览器和Oracle 8i数据库的动态的、数据驱动的 Web网站。

15.1 Oracle 8i 中的Java能力

图15-1说明了Java虚拟机结构。在数据库中有 Java使开发和部署因特网 /intranet应用变得更容易、更快、代价更少和更高效。Java程序语言使开发者能利用容易获得的大量开发工具。与C或C++这样的语言相比，Java代码在数据库服务器中执行是安全的，因为它不能操作指向内存位置的指针。Oracle 8i Java 虚拟机的内存管理器自动分配和释放内存，因此它使在数据库中内存越界所引起的数据损坏问题减至最小。

在Oracle 8i之前，没有企业级的用于开发大规模应用的 Java服务器集成到RDBMS中。许多应用服务器和数据库服务器厂商使用来自 JavaSoft (JDK) 的Java虚拟机来运行服务器端 Java应用。JavaSoft中Java 虚拟机被开发作为客户端虚拟机运行在单用户环境，它不提供在企业范围内应用需要的高可伸缩性。Oracle的Java 虚拟机与数据库有很高的集成度来满足关键任务应用的需要。Oracle 8i Java虚拟机包括集成的Java数据库连接 (JDBC) 驱动器SQL翻译器并提供如下特性：高性能、可伸缩性、高的可用性、管理能力和符合标准以及 Java运行支持。Java虚拟机符合JDK 1.1.6特性并支持所有标准的JDK库。

Java虚拟机有足够的内存管理能力，对于一般的 Java会话减少用户内存需求到 80~150KB

之间，它与数据库内核运行在同一进程和地址空间，共享大量内存堆并为优化性能进行直接访问数据库的缓冲区。因为 Java 虚拟机从数据库的 SGA 中分配共享内存池，所以为适应 Java 虚拟机，应设置较大 SGA。能够扩展到几万个并行的 Oracle 多线程服务器结构（MTS，从 Oracle 7 版本开始提供）是 Java 被集成到服务器的地方。MTS 基于一种结构体系，即数据库监听器路由用户连接到一组与处理数据库连接的服务器进程交互的调度程序。Oracle 8i 的目的是在因特网应用中支持大量并行用户。Java 虚拟机被设计成更快速、更安全和更可靠地运行 Java 应用。由于 Java 虚拟机被集成到数据库，它可跨所有 Oracle 支持的硬件和操作系统平台。因此实现 Java 的承诺：“一旦编写，任何地方皆可运行”。

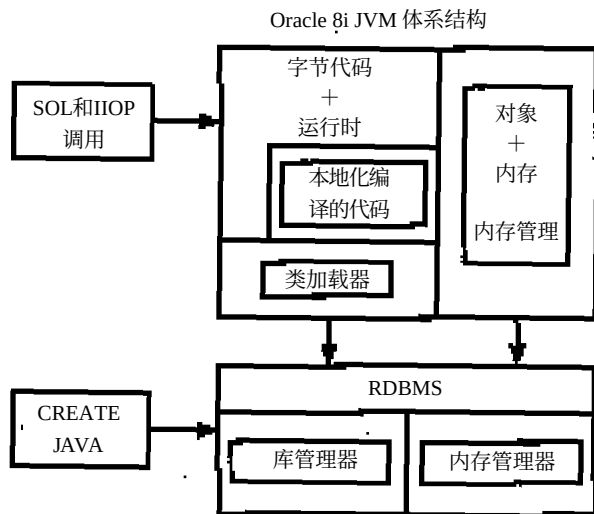


图15-1 Oracle 8i Java虚拟机体系结构

15.1.1 Java虚拟机的主要部分

Java虚拟机的主要部分如下：

- 库管理器提供方便地装载、存储和管理在数据库中的 Java 程序。用户导出或导入 Java 对象能用三种形式：源、二进制和资源（或归档）。
- 类装载机定位、装载和初始化本地 DBMS 存储的 Java 类以响应来自 Java 虚拟机的请求。它读取 .class 文件信息，产生 Java 虚拟机运行 Java 代码所需要的所有数据结构。
- 内存管理器负责自动地管理 Java 虚拟机的内存堆和有效地分配和收集对象内存。内存被分配为称为对象内存的标准大块。每一个对象内存是作为单一单元被分配和收集的一个内存池，在每个单元内所有对象共享同一生命时间和格式。
- Java 至 C 编译器（NCOMP）是一个嵌入 Java 虚拟机的本地编译器。它解释标准 Java .class 二进制文件，产生能被编译成 C 可执行文件的特殊 C 程序。
- Java 存储过程和触发器能够被 Java 本地编译器编译并存储在数据库中作为与 PL/SQL 存储过程类似的数据库对象。用 Java 写的过程能够调用用 PL/SQL 写的过程，反之亦然。因为存储过程在数据库中执行，所以它们使在应用和数据库之间的网络业务量最小化。你现在可以在你传统使用 PL/SQL 的所有地方中使用 Java。整个 Java 存储结构在图 15-2 中说明。

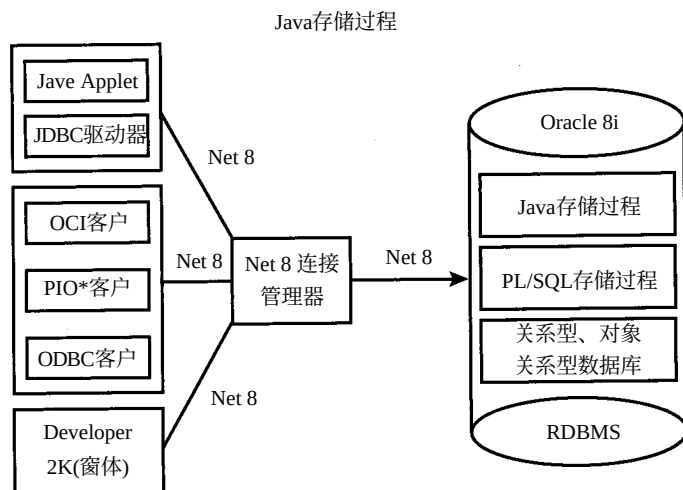


图15-2 在Oracle 8i中Java存储过程

15.1.2 编写Java存储过程

编写Java存储过程要求的步骤如下：

- 1) 编写你的Java代码。
- 2) 装载和解决它。
- 3) 发布代码。
- 4) 运行它。

首先，编写和测试你的Java程序，然后把它装入到Oracle 8i JVM中并解决所有引用。装入到数据库的Java程序被存储在数据库概要中作为库单元，它与被存储的PL/SQL程序单元类似。

下列例子是一个称为 student.java 的简单Java程序，它在计算机屏幕上打印一行“ student List ”：

```
public class student {
    public static void main (string args[]) {
        system.out.print ("Student List");
    }
}
```

一旦你编译并运行这个简单例子，你将知道你的环境已正确地建立了。

你能用两种方法装入Java源代码、二进制或资源文件。一种方法是在SQL*Plus中用新的DDL命令：

```
"CREATE JAVA {SOURCE | CLASS | RESOURCE}"
```

使用这个语句你能从一个二进制文件或通过使用在数据库中的LOB列导入Java代码。

第二种方法是借助Oracle 8i提供的LOADJAVA工具。这个工具自动装入Java到数据库中。它装入Java文件到数据库中名叫 create\$java\$llob\$stable 表中，然后运行SQL“CREATE JAVA ...”命令。还有用于从数据库单元中删除Java程序的DROPJAVA工具。

当你指定解决选择时，LOADJAVA使用SQL ALTER JAVA CLASS ... RESOLVE语句来解决现在被装载的Java类外部引用；否则ALTER JAVA语句在运行时隐含执行。

下列是一个用于解决一个Java类的语法例子。它装入exam.jar文件到用户SCOTT的概要中，然后解决外部引用：

```
loadjava -user scott/tiger -resolve exam.jar
```

loadjava也能采用列出一项或多项解决项目，每项有名字说明和概要说明组成，它的查找按列表顺序。

以下是一个使用LOADJAVA工具用RESOLVE特性来装载student.class文件到在数据库MYSID中的用户PETER概要中。它在PETER概要中查找以home.java开头的类名，然后在PUBLIC概要中查找：

```
loadjava -user peter/page@tysons1:5521:mysid
         -resolver '("/home/java/*" PETER) (* PUBLIC))' student.class
```

如下是在操作系统级使用二进制文件来装载java代码的例子：

```
/* Create a directory on the server's file system. */
SQL> CREATE DIRECTORY b_dir
      2 as '/home/user/oracle8/student';
Statement processed.

/* Then load the Java class file using the "CREATE JAVA CLASS ..." command. */
SQL> CREATE OR REPLACE JAVA CLASS
      2 USING BFILE (b_dir, 'student.class');
Statement processed.
```

在Java程序被装入到数据库之后，代码需要被发布。这用SQL注册程序来实现，但仅有顶层Java项目点需要被注册。顶层Java项目点是Java类或用SQL初始化调用SQL的方法。Java代码通过使用调用说明被发布。一个PL/SQL子程序被说明为由一个Java方法实现。

下面是一个对类employee.emp的app_bonus Java方法的调用说明的例子。

```
SQL> CREATE OR REPLACE PROCEDURE A_BONUS
      (C_SAL IN NUMBER, BONUS NUMBER)
      2> AS LANGUAGE JAVA
      3> NAME 'employee.emp.appr_bonus (float,float,float)';
      4> /
      5> Statement processed.
```

最后一步时通过SQL调用运行你的Java代码。有两种方法做这一步：

一种方法时使用CALL语句：

```
SQL> VARIABLE CSAL NUMBER
SQL> VARIABLE BONUS NUMBER
SQL> EXECUTE :CSAL := 20000;
Statement processed.
SQL> EXECUTE :BONUS := 100;
Statement processed.
SQL> CALL A_BONUS (:CSAL, :BONUS);
Statement processed.
```

另一种方法是从SQL查询中调用Java存储函数：

```
CALL-spec for s_grade Java method of class student.stud -
```

```
SQL> CREATE OR REPLACE FUNCTION SGRADE (GRAD IN NUMBER)
      RETURN VARCHAR2
      2> AS LANGUAGE JAVA
      3> NAME 'student.stud.s_grade (float) return java.lang.String';
```

```
4> /  
Statement processed.
```

以下例子是使用SQL DML语句来调用Java存储函数SGRADE：

```
SQL> SELECT SNAME,GRAD, SGRADE(GRAD) FROM STUD;
```

PL/SQL能够直接通过简单包含PL/SQL 调用说明来调用Java存储过程。

下面是PL/SQL直接调用Java存储过程的例子：

```
DECLARE  
    sal      number;  
    bonus    number;  
BEGIN  
    ...  
    A_BONUS (sal, bonus);  
    ...  
END;
```

提示 在Java，远端方法请求（RMI）调用简单地是特定的远端过程调用（RPC）。Oracle 8i使用RMI作为在因特网内部ORB协议（IIOP）之上的传输协议。正如你预期的，这个已存在的技术“式样翻新”是一个面向对象和网络编程的扩展。然而网络和分布式编程、面向对象编程和Java这些主题已经超出本书讨论范围，所以请参考适当的书籍以获得更进一步的解释和例子。

15.1.3 SQLJ：Oracle 8i Java预处理器

SQLJ是一个新的编程语法，它是用于 JDBC的Oracle预编处理器；它可以被认为是PRO*Java。它使开发者可以在Java程序中嵌入SQL语句，与Oracle的其他预编译器（例如PRO*C）类似。与既支持动态有支持静态的SQL的Pro*C不同，SQLJ只支持静态SQL。Oracle与许多其他软件商共同开发SQLJ，这些合作者包括IBM、Tandem（现在的Compaq）和Sun，在1998年12月SQLJ被采用为ANSI标准。它被设计为独立于目标数据库使用，因此如果其他厂家的数据库支持SQLJ，那么在其中使用Oracle SQLJ存储过程将是可能的。

用嵌入式SQL语句写一个Java程序，然后用SQLJ转化器处理它，用SQLJ的运行时调用替代SQL语句。然后任何Java编译器能够被用来编译产生的Java程序。SQLJ由一个转换器和一个运行组件组成。转换器检查嵌入式SQL的语法和标点符号，产生一个.java源文件以及一个或多个SQLJ环境资源文件。SQLJ环境资源文件包含关于你的SQL操作以及被访问数据的类型和模式的详细信息。然后，SQLJ自动地调用Java编译器，从.java文件中产生.class文件。SQLJ还自动调用定制器以适应连接的数据库的环境资源文件。在缺省的情况下，Oracle定制器被用于增加Oracle特定的扩展特性的支持。SQLJ运行库环境由一个SQLJ运行库，这个库是Java并且调用JDBC驱动器来连接正确数据库。

图15-3说明SQLJ开发过程。

开发SQLJ应用有四个基本步骤：

- 1) 使用SQLJ转换器转换SQLJ代码，用产生带有SQLJ的运行时调用的Java源文件。
- 2) 使用Java编译器编译Java源文件。
- 3) 定制Java.class文件。
- 4) 使用SQLJ运行库运行应用。

这里有一个在SQLJ程序中嵌入一个SQL select语句的例子。符号#sql放在SQL语句前，跟

随内容包含在大括号内部：

```
#sql { select deptno into :deptno from emp where empid = :emp };
```

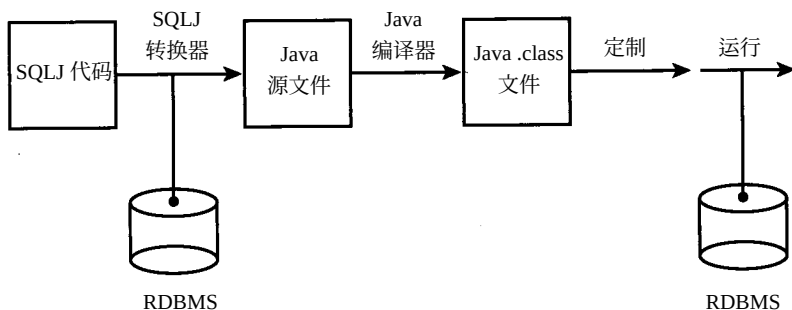


图15-3 SQLJ开发过程

运行SQLJ程序的唯一要求是SQLJ运行库、一个JDBC驱动器和程序将在其上执行的Java虚拟机。SQLJ比JDBC强，因为它在转换时检查SQL的语法和标点符号，而不是在运行时进行。你不必在SQLJ或JSBC之间做出选择，这些API是JDBC的预处理器；它可以被认为是PRO*Java。它使开发者可以把嵌入SQL语句到Java内部。

SQLJ代码与JDBC代码对比示例

如下是SQLJ和JDBC对相同SELECT语句的程序片段。SQLJ代码比较短，因为它使用主变量来将参数传递到SQL语句，而JDBC使用分开的语句绑定每一个参数并获得每一个结果：

```
###SQLJ
java.sql.DATE a; int b; string c;
...
#sql { SELECT col1 INTO :a FROM EMP WHERE col2 = :b and col3 = :c };

###JDBC
java.sql.DATE a; int b; string c;
...
PreparedStatement x = connection.prepareStatement
("SELECT col1 FROM EMP WHERE col2 = ? and col3 = ?");
x.setInt(1, b);
x.setString(2, c);
ResultSet y = x.executeQuery();
y.next();
a = y.getDate(1);
y.close();
x.close();
```

15.1.4 企业级JavaBeans支持

企业级JavaBeans基本上是实现一系列用于事务处理功能的预定义Java接口的分布的Java组件。它们使开发者能够设计并把应用封装在组件内，这个组件可以和其他开发者编写的部件装配在一起。基于组件的程序已经变得更加流行，因为它可再用应用代码，不同厂家的应用容易装配，布置灵活。更多的组织正在为新的应用使用基于组件的开发，这是由于传统的应用不能再用它们的代码。优点包括更好的设计，代码再用，更容易维护和在多层的环境中更容易跨层布置。JavaBeans已经正在被广泛用于装配到基于GUI的应用，但主要作为客户端应用。不同于客户端组件，服务器端的组件（如EJB）一般是事务性的，它们封装业务逻辑并

需要运行在服务器环境中。

Oracle为基于组件的分布计算选择EJB作为它们的结构。当前对于服务器端部件有三种模型：EJB、CORBA和DCOM。由JavaSoft创建的JavaBean特性将Java扩展到支持跨平台对象技术。分布式组件对象模型（DCOM）是用于建立在桌面上跨网络互操作的组件件的Microsoft组件模型。DCOM是COM的扩展。COM作为Windows操作系统的部分已许多年了；有了它作为基础结构，OLE和ActiveX才成为可能。通用对象请求代理结构（CORBA）3.0是最新特性，在1999年12月由对象管理组发布（OMG）。这个模型指定一个传输机制——因特网Inter-ORB协议（IIOP），它允许在不同硬件上运行的不同操作系统互操作。它也指定用于指定接口到对象的接口定义语言（IDL）。有关CORBA如何与Oracle相关更详细的讨论，见15.1.5节。

EJB在一个容器内执行，容器提供这些部件在其中运行的操作系统进程或线程。容器类型有Web服务器、事务处理（TP）监控器或数据库系统等。

开发和配置企业级JavaBeans

开发和布置企业JavaBeans包括如下步骤：一个开发者需要为客户应用定义4个项目以能够请求包含业务逻辑的块（bean）方法：

- 定义远端接口——远端接口列出所有方法或客户能调用的公共接口块。

如下是一个用于Stock块的远端接口的例子：

```
public interface Stock extends javax.ejb.EJBObject
{
    float getPrice () throws java.rmi.RemoteException;
    void buy (float amount) throws java.rmi.RemoteException;
}
```

- 定义主接口——主接口作为使客户能够创建块实例的机制。以下是 Stock块的主接口：

```
public interface StockHome extends javax.ejb.EJBHOME
{
    Stock create () throws java.rmi.RemoteException, javax.ejb.CreateException;
}
```

- 定义块——块自身是标准的实现业务逻辑的Java代码。

下面是块自身的一个例子：

```
public class StockBean implements javax.ejb.SessionBean
{
    public float getPrice ()
    { ADD ANY BUSINESS LOGIC HERE ...
    }
    public void buy (float amount)
    { ADD ANY BUSINESS LOGIC HERE ...
    }
}
```

- 定义接口布置描述器——它将所有块组件连结在一起。其语法很像Java类，用Session Bean语句开始。这里是为Stock块创建一个布置描述器的例子：

```
SessionBean StockBean {
    BeanHomeName = "bn=myStock";
    Attribute
        RemoteInterfaceClassName = Stock;
    Attribute
        HomeInterfaceClassName = StockHome;
    ...}
```

这些步骤完成之后，你需要编译和封装你的Java代码。

用如下语句编译EJB、它的远端接口和对Java类的主接口：

```
% javac StockBean.java Stock.java StockHome.java ...
```

然后，用下列语句在EJB包（一个Java归档或JAR文件）中封装类文件：

```
% jar cf Stock.jar *.class
```

接下来，用Oracle提供的CREATEEJB工具增加块的布置描述器：

```
% createejb Stock.jar -desc Stock.dsc
```

最后，在Oracle 8i服务器上使用Oracle提供的DEPLOYEJB工具布置EJB。

如下是用用户SCOTT概要布置EJB和创建一个jar文件的例子：

```
% deployejb -u scott -p passwd -s <servicename> -gen Stockclient.jar  
Stock.jar
```

在成功地布置EJB之后，可以从一个客户端应用激活它。

15.1.5 Java和CORBA

CORBA（通用对象请求代理结构）是使用不同语言编写的分布式程序可在网络（包括因特网）上进行通信的面向对象的协议。CORBA由对象管理组（OMG）定为一个基于标准的分布式组件模型。它是建立和布置跨平台管理可互操作的分布式对象应用的标准。

CORBA 2.0（和更高版本）对象使用OMG的Internet Inter_ORB（IIOP）协议通信。IIOP是在两个或多个运行在因特网、intranet和企业计算环境中分布式对象之间进行通讯的另一个OMG标准。

用不同语言编写和运行在不同平台上的CORBA组件能够透明地通讯和互操作。因此，客户和服务端代码能够用任何语言编写（不仅仅使用Java）并被编译为本地机器代码。Oracle 8i使用Java作为它的CORBA实现语言。Oracle 8i集成基于Java的，CORBA2.0对象请求代理（ORB），它使用户可以利用IIOP调入调出数据库。这个ORB基于来自Inprise的VisiBroker for Java的代码。Oracle 8i带有一整套用于开发CORBA应用的工具，使用这些工具，你能编译IDL说明，或装载Java源文件或类到数据库中。

15.1.6 Oracle的Java开发工具：JDeveloper

JDeveloper 2.0是一个基于Windows的Java编程工具，它带有集成的项目导航器、一个代码编辑器、一个程序调试器和一个编译器。SQL转换器和调试器也是JDeveloper中的嵌入部分。

JDeveloper被设计供开发者建立多层Java应用。开发者创建、调试并布置Java存储过程、企业级JavaBeans、SQLJ和CORBA对象以及HTML Web应用。项目导航器使开发者能在一个会话的不同层中管理多个项目。有许多代码产生向导来帮助快速开发应用。企业级JavaBeans向导能自动创建新的EJB。在Servlet向导和Java服务器页（JSP）向导帮助下，Servlet和JSP能够很容易产生。数据格式向导指导创建单表或主从格式。开发者能使用PL/SQL到Java向导来从被选择的存储过程产生Java类定义。几个JDBC驱动器随JDeveloper被提供，用于访问其他数据库，尽管它不是最紧密地与Oracle集成。

15.2 因特网文件系统

现在用户能使用新的因特网文件系统将 Web页、文献、扩展页、字处理文件、图像和其他传统文件存储在数据库内。它表面上作为另一个文件系统并能够像任何其他网络目录一样被访问。IFS把它的内容组织到文件夹里，它们呈现给客户任何文件夹或目录应有的方法。存储在IFS中文件可得到Oracle数据库提供的所有可靠性和安全性，这是与其他文件夹或目录不同的。系统管理员的工作很容易，因为他们仅须维护、管理、备份和装载单个系统而不是几个分开的系统。

除了在数据库中提供比较容易的文件存储，IFS还提供关系和非关系数据的完整性。用户能用多种协议访问IFS，对它们的数据提供广泛的访问。被授权用户能查找和查看存储在IFS中他们的任何文件，他们可以从任何计算机通过标准 Web浏览器、Microsoft Windows Explorer、FTP客户或email客户进行上述操作。IFS使用Java编写，作为特殊的解析器和翻译器。IFS使用XML来解析和“翻译”文件，XML是下一代因特网标识语言。它使数据库能使用文件过滤器将已知的文件类型的内容转换为XML文件。XML能够存储在数据库表中。当被客户请求时，翻译转换存储的XML到文件格式（HTML、XML或初始化文件格式）。用户通过Windows Explorer用拖曳方法能够很容易地移动文件进或出IFS。在IFS内双击它们能够直接编辑并启动适当的应用。当增加或更新一个文件时，IFS使用ConText技术来创建一个基于内容和文件的外部属性（作者、题目、日期等等）的关键字索引。随后，IFS能够使用预构造的索引快速完成查找。

IFS的关键特性包括如下：

- 高级查找 —— 文件自动按内容和文件属性进行索引以提供关系查询和基于 ConText的查询。
- 检入、检出（CICO） —— 当检出时文献被锁存，直到返回检入。
- 访问控制列表（ACL）安全 —— 基于ACL模型的安全和特权。
- 版本化 —— 使能多版本文献。
- 改变通知 —— email通知可以在文件的插入、更新或删除时获得。
- 自动过期截止 —— 文件在经过一定期限后被清除。
- XML支持 —— 它通过文献解析和翻译获得。
- 广泛访问 —— 它被通过SMB、HTTP、FTP、SMTP、IMAP和POPS协议提供。
- Java、CORBA、PL/SQL编程API —— 用于编写基于IFS的应用。

15.3 Oracle interMedia

Oracle interMedia包含5个管理模块来处理数据库内的多媒体。

建立在Oracle Context 技术上的Oracle interMedia文本，使用户可以使用高级查询和自然语言查询搜索文本。Oracle interMedia Image管理二维静态、数字图像并支持基本的操作函数，如缩放和剪修。因为它支持多数桌面发布图像文件格式（包括 TIFF、JPG、BMP、TRAGA、PCX、PICT、GIF、CALS Raster、SUN、Flashpix和RPIX），不是图像文件格式专家的用户也能够用interMedia Image存储和获得图像。

Oracle interMedia Video和interMedia Audio能够从不同源（例如，Oracle VideoServer、Oracle 8i和Web站），导入基于浏览器的视频和音频。如下工业标准的视频和音频格式被interMedia支持：AIF、AIFF-C、AUFF、WAV、Quicktime、AVI、MPEG和Real Networks。应用可以像对待传统关系型数据一样查询并获得多媒体数据。Oracle interMedia Locator处理定位查询，例如查找存储、分布点和基于给定空间数据的事件（例如某一地址的位置或距离）。因为interMedia完全集成在Oracle 8i中，开发者通过SQL调用而不是专用接口，具有高级文本查找能力。

Oracle interMedia Text 使用文本索引。文本索引能够在几乎所有数据库的任意列或者对象上创建来完成查找。文本查找利用在数据库内的索引执行并返回一个匹配文档列表。尽管被interMedia查找的文献能够存储在文件系统或任何 URL地址的WEB上，但数据库存储和管理所有文本索引。interMedia能够用单个SQL命令处理混合查询（数据库中的全文或结构化数据）。例如，你能够查找包含“Internet”的文档并结合存储数据中的其他信息（例如，作者、日期或文件类型）。SQL CONTAINS函数使用户能从任何支持SQL或PL/SQL的工具或环境中运行文本查找查询。interMedia提供全文搜索能力，它包括精确单词或短语查找、布尔特性、模糊查询、通配符、大小写敏感和部分查找。它有通过使用大约五十万个单词的主题内容识别文档的能力。还有根据主题查找文档，这可通过使用优化组合全文、基于语言和基于主题技术来进行。

下面的例子将对 bdate列进行查找。然后结果与 CONTRAINS函数合并这个函数搜索包含“Internet”字并和浏览器有关的文档的文本索引：

```
SELECT author, isbn, bdate from books
WHERE bdate > '01-MAR-1999' AND
CONTAINS ( btext, 'Internet AND ABOUT (Browser)' ) > 0;
```

Oracle interMedia使用与Java或C++类似的对象数据类型，称之为ORDAudio、ORDImage和ORDVideo来描述声音、图像和视频数据。这些数据类型的一个实例由诸如元数据、多媒体数据和方法等属性组成。元数据是关于数据的信息；多媒体数据是实际的声音、图像和视频；方法是能够在对象上执行的过程（例如，store、deliver和compress）。多媒体数据能够在数据库中作为BLOB类型保存，或者在外部保存并能够很容易地导入到数据库中。多媒体数据能够存储在其他源中，如BFILE（大对象的操作系统文件）、基于HTTP服务器的URL、特定的媒体服务器（例如Oracle VideoServer）或用户定义的源。这些数据类型的元数据和方法总是在数据库中进行存储和管理。interMedia使媒体数据能通过标准查询或关系数据（例如学生姓名）和对象的元数据（例如学生照片）来定位。当定位后，媒体数据能够通过ORDAudio、ORDImage和ORDVideo数据类型，像其他关系数据一样使用SQL对它进行访问。

Oracle interMedia Locator支持用于定位应用和近似查询的联机地理位置编码功能。使用Oracle 8i空间组件创建它。它在Oracle 8i数据库中支持地理编码、存储和地理编码数据查询。地理编码使用点代表地址和位置信息（例如，邮政编码、经度 /纬度）。和第三方地理编码和绘图工具相结合，interMedia能够完成地理编码数据的距离查询。在数据被地理编码后，interMedia在应用中能计算距离和图形化表述它的位置。例如，你能查询数据库来定位距你工作地点最近的加油站。

15.4 Oracle WebDB

Oracle WebDB提供包括HTML服务器的一个完整的Web开发环境，因此它能处理HTTP请求和服务Web页。不需要在每个用户的机器上安装软件，也不需要使用FTP进行远端文件管理，或者进行复杂的软件升级。开发和布置应用的客户软件时，唯一需要的是 Web浏览器——Netscape Navigator 3.1或更新的，或者Microsoft Internet Explorer 4.0或更新的。WebDB被设计成方便使用、方便访问和方便管理。它是完全包含在 Oracle数据库（版本7.3.3或更新）的PL/SQL过程的集合。它也包括作为 Web服务器和PL/SQL到数据库的接口的轻量监听器。WebDB将全套基于浏览器的用于开发 Web的HTML工具与HTML接口相结合。通过使用指导用户经过创建组件（也就是表格、报告、图表、菜单、日历、分层）的整个过程的简单向导，可以实现快速开发。向导可以替你写 PL/SQL代码，你也可以绕过向导自己编写代码。当开发者创建一个组件时，它被给予一个 URL。用WebDB进行布置与增加 URL到HTML文本一样容易。图15-4给你一个图表向导的例子。

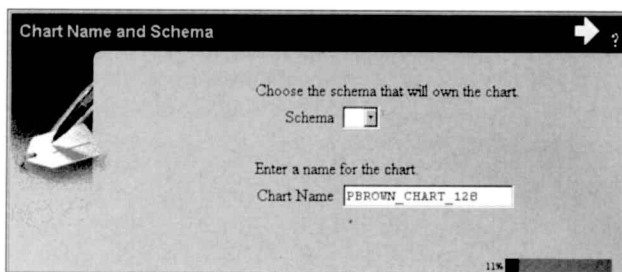


图15-4 WebDB图表向导屏幕

在WebDB中包含所有用于管理和构造数据库的工具。管理员能够从浏览器管理授权和监听器设置或监控终端用户和数据库活动。通过Web浏览器可以创建、查看、编辑和拖动任何数据库对象（表、分区、重做日志等等）。WebDB有高级性能追踪能力，这有助于解决性能问题。你能够监控的用户因素包括：响应时间、按组件名称的请求、按天和小时的请求、按浏览器类型的请求和按IP地址的请求。图15-5 图15-5 一个WebDB图表显示按数据库用户的请求说明按用户的请求数图表。

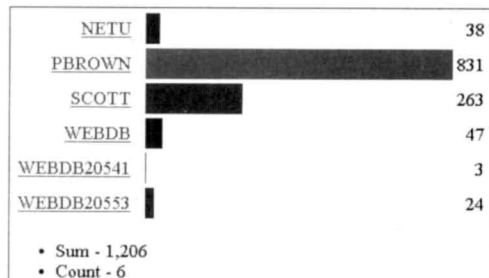


图15-5 一个WebDB图表显示按数据库用户的请求

使用WebDB创建的Web站点被分成叫做角（corner）的几个部分。角包含与特定主题相关的条目。站点管理员为角指定负责管理该区域的拥有者。角的拥有者能将对他们的角的内容进行不同类型访问（例如，增加、编辑和删除）的特权授予其他用户。内容的作者能够更新Web站点的角，只需点击界面而无需任何其他软件。当维护集中控制的 Web站点时，它分散内容的管理。

15.5 Oracle 8i的可用性和可恢复性

Oracle 8i的高级特性和能力提供随时可获得的增强，包括并行服务器的改进、一个新的快

速启动的失败恢复、联机备份、恢复和透明的 Application Failover。

Oracle并行服务器使多个Oracle实例能运行在一个簇的节点上以并行访问单一数据库。这种共享访问帮助删除任何单个失败点，因此一个系统中一个节点毁坏，其他节点能够继续工作。在Oracle 8i中，并行服务器通过使用缓存融合聚簇结构体系和簇负载平衡来支持成千个用户。缓存融合使用专用高速互连直接在节点间为虚拟无限制规模传递数据。这个技术使在簇中一个节点缓冲区可直接向其他节点的缓存发送数据块，因此可减少磁盘 I/O。簇负载平衡自动将用户连接到利用率最低的系统节点，因此减少了用户访问时间并增加了可被服务的用户数。

自动化和只读独立数据库使Oracle可自动维护一个或多个用于灾难性恢复的成品数据库拷贝。独立数据库最初通过复制成品数据库创建。复制能够驻留在同一位置或不同位置。在独立数据库创建之后，成品数据库的更新和改变被传播到独立数据库，因此数据库保持同步。独立数据库使用加强恢复模式。当在成品数据库产生归档重做日志以后，它们通过 Net 8连接传输到独立数据库。在独立数据库接收归档重做日志后，它们被自动运用以使两个数据库保持同步。这样就不必手动维护独立数据库，但也可以这么做。如果网络发生错误或者发生空间溢出问题，阻止日志文件成功创建，那么，在特定的时间间隔之后，自动重试归档操作。以前，如果没有随后创建的成品数据库产品的另一个拷贝，当前的独立数据库不能被打开或使用。在Oracle 8i中，DBA现在可以只读模式打开独立数据库。这提供了正当对成品数据库卸载一些进程时查询和产生独立数据库报告的一种办法。在这一模式下，独立数据库能够连续自动接收来自成品数据库的归档重做日志；然后，当独立数据库被返回到加强或手动恢复模式，排队的日志被应用。

联机数据库重组和碎片消除是另一特点。Oracle 8i现在在数据库重组与索引创建和重建期间，用户可以完全访问数据库。当表和索引完全联机且没有中断任何正在进行的对基础表的改变时，DBA能够执行维护。

当正在创建索引时，用户能够连续访问和更新基础数据表。在创建期间任何对基础表索引的改变被记录在日志表中，并在操作结束后应用到新索引中。索引也能够被联机重建，提供高的数据库可用性。当索引创建时，重建使用日志表来记录在重建操作期间对基础表所做的改变。除了碎片消除不需要附加的存储空间外，联机索引碎片消除和合并很像联机索引重建。索引碎片消除进程将单块合并到相邻的小于一个数据块的索引块中。

Oracle 8i当用户可用索引组织表（IOT）时，IOT可以被移动和重组。对表所做的改变记录在日志表中，在进程结束时合并表。这些联机重组能够在单个 IOT或在被分区的IOT的单个分区中进行。

快速启动的失败恢复是一个旨在使系统失败恢复尽可能快并减少数据库不可用时间的新特性。它是很迅速的，恢复时间能够由管理员设定。在一个系统崩溃后，数据库一般做向前回滚提交过的改变和回滚未提交的改变。Oracle 8i有一个新的快速启动检查点结构。检查点进程不断地写被修改的块到磁盘中。DBA指定多长时间向前进行回滚取代应发生多少次检查点。回滚处理希望通过使用称之为非块回滚的技术进行消除。现在，新的事务能够在向前回滚完成之后马上开始。如果一个新事务访问被死亡事务锁定的一行，只有阻塞它的进程的改变被回滚。同时，数据库有能力并行回滚死亡事务。

扩展备份和恢复用恢复管理器提高性能。恢复管理器是一个管理创建和还原数据库备份

的进程的工具，它现在有一个能够代表数据库引导备份软件完成复制操作的集成的媒体管理 API。它也能在自己目录上针对媒体管理软件目录和过期标注删除备份进行交叉检查。恢复管理器使 DBA 能在恢复/还原操作发生之前预览即将发生的事。另外，恢复管理器现在能多路备份集到多个 I/O 设备。Oracle 使用 Transparent Application Failover 能增强从系统崩溃中恢复的能力。当数据库连接中断时，客户程序能够自动重新连接。同时，任何在崩溃时正在被处理的查询能够自动重启，返回程序的数据将从失败发生点恢复。

15.6 安全性

Oracle 8i 提供一个布置和执行分布式 Java 应用的安全平台。过程、CORBA 服务器和企业级 JavaBeans 组件都能够在 Oracle Net 8 和 CORBA IIOP 上通过因特网标准的安全套接字层 (SSL) 进行访问。Oracle 8i 提供单个符号加密，使用户可在因特网/intranet 环境中认证他们自己，这样可以缓解有太多口令的问题。基于单符号加密的公用密钥结构 (PKI) 使用 X.509 (版本 3) 证书在 SSL 上认证。Oracle Wallet 管理器是一个由 Oracle 提供的用来保护和管理用户证书、密钥和信任点的工具。

Oracle 8 引入了企业角色，这是一个或多个跨越一个或多个数据服务器、被集中管理和维护的全局角色。Oracle 8i 通过在 Internet 目录源中存储和管理用户信息，而不是用专用数据存储来提高这些能力。Internet 目录符合轻量目录访问协议 (LDAP) 版本 3 标准。单站管理是通过 Oracle 企业安全管理工具的增强得到的。现在，管理员能够从单个控制台在多个 Oracle 8i 数据库中创建一个用户，或者跨多个数据库创建企业角色。Oracle 8i 引入单个企业用户，在整个组织为每个用户创建一个帐户，而不是为每个用户创建多个帐户。现在，你要管理的用户帐户更少，安全更容易通过使用 Internet 目录来加强。

Oracle 8i 提供强迫的、灵活的、好的访问控制以及安全的应用环境。通过一个或多个在表或视图上的安全策略使它们可用。用访问控制，在访问数据时有更好的粒度。对不同类型的访问 (也就是 select、update 和 delete) 有不同策略。现在，你能指定谁能访问表中的特定行，区分数据访问到单一级别。访问能够基于用户 (例如，一个用户角色是经理还是职员) 或基于应用 (例如是用户正在访问一个人力资源应用，还是顾客正在创建一个定单)。

下列这个例子中，表 ACCOUNT 有一个策略，银行的顾客仅能看他们自己的帐户，一个顾客通过发出如下命令访问表：

```
SELECT * FROM ACCOUNT;
```

这个 SELECT 语句使数据库调用策略函数，然后透明地和动态地通过附加的 WHERE 语句重写这个查询：

```
SELECT * FROM ACCOUNT WHERE ID = (SELECT ID FROM CUSTTAB WHERE CNAME =  
USERENV('user'));
```

安全应用环境是在 Oracle 8i 中基于访问控制策略提供的。应用环境是由用户作为它们的属性定义；每一个应用能够有不同属性的环境。例如，人力资源应用能有基于 employ number、department、position 属性的安全策略。据此，数据库将通过使在经理位置上的用户只能更新他自己部门里的职员的记录，以执行这个策略。

15.7 其他特性

Oracle 企业管理器 2.0 版是 Oracle 8i 提供的最新版本的系统管理工具。提供的一些特性有：

从任何虚拟环境管理的能力、在企业中管理每件事的能力，和最少化手工管理系统的“设置和忘记”管理。它是基于 Java 的应用并能够从浏览器中访问。管理员能够从一个中心控制台管理多个机器。仓库现在是一个共享的公共知识库，它可供多个 OEM 管理员使用。最初有一个缺省的超级用户 SYSMAN，SYSMAN 能创建其他超级用户和管理员并分配给它们适当的职责和特权。每个新的管理员被给予一个完成管理任务需要的访问控制台口令。因为每个管理员有特定的职责，所以他们能为其他管理员定义对他们创建的组、工作和事件的允许级别。允许级别包括无、视图、修改和完全。

LogMiner 是一个让 DBA 使用 SQL 来读、分析和解析重做日志文件的数据库分析工具。LogMiner 要求访问被分析的数据库的数据字典并用来自 Oracle 8.0 的重做日志文件继续工作。正被分析的日志文件被映射到动态性能视图（V\$logmnr_contents）。在这个表中的每一行代表在数据库中完成的一次逻辑操作的信息。对这些操作的每个，由给定的 REDO 和 UNDO SQL 语句描述在数据库中所做的改变。SQL REDO 语句显示在数据库中所做的原始操作，而 SQL UNDO 语句显示出能回滚这些改变的语句。

这个例子给出的查询要查找 Benlin 对表 SAVINGS 所做的改变。

```
select sql_redo, sql_undo from v$logmnr_contents where username = 'Benlin' and  
seg_name = 'SAVINGS';
```

查询结果如下，它显示出 Benlin 首先删除他的储蓄记录，然后用更高的量重新创建这个记录。Benlin 所做的回滚改变的语句显示在 SQL_UNDO 列。

SQL_REDO

```
delete * from savings where id=999  
and rowid='QSDFJKKJKAA';  
insert into savings(name,id,amt)  
values ('benlin', 999, 500);
```

SQL_UNDO

```
insert into savings (name, id, amt)  
values ('binlin', 999, 10);  
delete * from savings where id=999  
and rowid='QSDFJKKJKAA';
```

数据库资源管理器是在 Oracle 8i 中在数据库用户和应用之中分配和管理 CPU 资源的一个新工具。过去，所有用户和应用对数据库进行同等访问。现在你能对更重要的任务分配更多资源。使用这个工具，用户和组被分配到资源消耗组。管理员建立一个资源计划指定如何在不同资源消耗组中进行资源分配。分计划进一步细分资源，它能在资源消耗组中进行定义。对于分配资源最多能定义八个级别。使用数据库资源管理所做的改变能够动态地在数据库上改变，不需要关闭后再重启系统。管理员能够有许多不同的资源规划和资源消耗组，所有这些都保存在数据字典中。

并行自动化程度（ADOP）特性也被集成在数据库资源管理器中来优化系统使用。这个特性限制并行查询操作的并行化程度。

分区使用户能将表和索引分成更多的可管理单元。Oracle 8i 引入了对表的范围分区，在范围分区里数据是基于分区键的值的范围被分开。在 Oracle 8i 中，现在有哈希分区和合成分区。哈希分区使你能均匀地跨设备分离数据，因为行按一个分区键的哈希值分开。合成分区是范围和哈希值的合并。首先，数据被按照值进行范围分区。然后，每一个分区再细分为几个哈希分区。Oracle 8i 也支持面向关系和 LOB 列的类型。分区能够独立地从表或索引的其他分区中被备份和恢复，不妨碍未受影响分区的访问。

具体化视图是一个新的数据库对象类型。它是用于创建、维护和使用总和表的优化特性。

总和表也称为集合表，在其中求总和和连结操作被预计算，结果被存入数据库中。通过访问总和表而不是每次根据数据求总和，性能得到很大提高。管理员能够指定如何和何时刷新这些表（也就是同步、当新数据被装入时、每周等）。同时，总和查询向导帮助管理员创建和维护总和表。

可传输的表空间使你能在数据库之间移动表空间，而不用做完全的导入 / 导出。DBA能够在相同的系统之间（硬件结构）复制表空间的数据文件，因此转移数据既容易又迅速。要传输的表空间必须被自我包含，也就是没有从表空间内部到表空间之外的引用点。这里有一个称为transport_set_check的PL/SQL过程，用于检查一系列表空间是否是自我包含的。

首先，被移动的表空间被置为只读模式来保证被捕获的数据相一致。然后，导出表空间的元数据，这是迅速的，因为数据量很小。接下来对表空间的数据文件进行复制，使用操作系统应用将其复制到新的系统中。在新的系统中，导入表空间的元数据。这也同样迅速，也因为量很小。还有另一种方法来在表空间上执行按时间点恢复。通过移动一个好的表空间的拷贝到成品数据库中，恢复能够被迅速归档。

本地化管理（位映射）表空间是在数据库中提高空间管理和最小化区间碎片的一种新的表空间类型。一个区间是由一个或多个相连的数据块组成的逻辑存储单元。在 Oracle 8i之前，管理自由区间和使用过的区间是通过数据字典表来完成的。现在，有位映射选项能指明表空间中使用过的块和空闲的块，因此最小化了对数据字典的访问。这些表空间使用带有EXTENT MANAGEMENT LOCAL子句的CREATE TABLESPACE命令创建。

这种类型的表空间将自动调节所有新区间的标准大小，既可用 autoallocate选项又可用uniform选项。在autoallocate选项下，系统将自动创建变化区间尺寸。你能指定初始区间的尺寸，然后系统将决定附加区间尺寸。在 uniform选项下，所有区间创建是相同尺寸的。如果你不指定EXTENT MANAGEMENT子句，缺省是EXTENT MANAGEMENT DICTIONARY，它表明这个表空间将由系统通过数据字典进行管理。

注意 如果你使用LOCAL子句，就不能在CREATE TABLESPACE命令中使用缺省的存储语句、MINIMUM EXTENT或TEMPORARY参数。

下面是创建本地化管理表空间 TS的例子。它的每个区间分配为 64K，在位映射中的每位表明32块（假设每块2K）。

```
SQL> create tablespace TS
      2 datafile '/u01/oracle/mysid/ts.dbf' size 10M
      3 extent management local uniform size 64K;
```

现在能够在用户临时表空间中创建临时表。临时表仅在事务或会话期间保持，只对向它们插入数据的会话可用。临时表不能被分区、索引组织、聚簇或者包含嵌套表或变量数组类型的列。这些表不能有外键约束定义。

下列语句创建临时表 stocks：

```
SQL> CREATE GLOBAL TEMPORARY TABLE stocks
      (stockname varchar2
      stockprice float);
```

Drop column命令提供从一个表中删除一列而不必重新创建该表和它的相关索引和约束的能力。

这个SQL语句将从表census中删除列描述符 col_1和与col_1相关的每一行数据，在该列上

定义的所有索引或约束也都被删除：

```
SQL>ALTER TABLE census DROPcol_1;
```

15.8 小结

有了新特性和增强，Oracle 8i已成为一个因特网计算的理想平台。Oracle 8i将通过服务器端的SQLJ与数据库中的Java虚拟机均衡，从而使上万个Internet用户连接到Oracle 8i数据库的一个单独的实例。因为JVM是集成在Oracle 8i中的，所以它继承了数据库的可靠性、可移植性、可伸缩性、吞吐能力和性能。开发者能用Java过程取代PL/SQL过程，或者使它们共存。提供的工具（例如WebDB）改进使用的简单性和Web应用的布置。iFS选项使多协议跨平台访问相同的信息对所有用户变得更加容易。interMedia产品使标准SQL能访问应用中的文本和多媒体内容。

第四部分 性能调整

第16章 性能调整基础

本章要点：

- 再看物理设计
- 为什么进行调整
- 调整规则
- 调整目标
- 使用投资的回报
- 再看应用类型
- 理解基准测试
- 使用Oracle诊断工具

16.1 再看物理设计

第3章“物理数据库设计、硬件和相关问题”中说明物理（和逻辑）设计是调整数据库的第一步。事实的确如此，在许多有关调整方面书籍中的一个首要步骤便是调整设计。通过采用正确的逻辑和物理设计，你便可以完成这一步骤。如果所有的步骤都已完成的话，你的设计将使你的数据库能够在一个可接受的水平上运行。然而，有两件事情总是伴随着数据库系统：增长和变化的应用需求。

假如你的数据库主要由查询表组成且保持相对不变，或者由于其他原因而使得数据库很少增长或增长得非常缓慢，那么这个因素不是问题。然而，如果你的数据库和大部分其他数据库一样以10%或更高的百分率过度增长的话，那么，不管最初的逻辑与物理设计多么好，你都需要周期性地调整该数据库以便容纳或补偿此增长。此外，所有的数据库系统已经变成不断重定义应用这一现象的牺牲品。这里有一个你或许曾经遇到过的例子：

经理：“应用程序不工作了。它的运行速度太慢了。数据库怎么了？”

程序员：“代码没有改变。”

数据库管理员：“数据库没有改变。”

那么问题出在哪里？在本实例中，使用（usage）可能已经改变了。还有一种流行的说法叫做应用需求已经改变了。例如，假定原先的应用程序用来实现对七个表分别完成只读存取；然而，如果该应用程序目前用来将所有这七个表合起来以收集比原先要求的更多的信息的话，那么或许需要进行一些调整。事实上，或许需要完成某些物理再设计，这些物理再设计也称为物理重构。

你可以在物理上重构一个数据库以提高其性能并仍然可以使应用随意地运行。这对 Oracle 以及其他厂商来说是可能的，因为 Oracle 遵守 Codd 规则8：物理数据独立。因此通过与用户联

系，答案或许可以找到：

用户：“应用程序不工作了吗？我们正在使用一些不同的表。”

除了增长与变化的应用需求外，还有一件需要利用数据库管理员的性能调整技巧的事情：不良设计的数据库。

由于有增长和变化的应用需求，所以你已经假定数据库被很好地设计。不幸的是，正如许多数据库管理员所知道的那样，事实经常并非如此。很多时候，某个数据库管理员要接收另一个数据库管理员留下的数据库系统，或者面对一个由于财产迁移而得到的数据库系统，这被称为修理别人的数据库系统性能。在诸如此类的情况下，在调整性能以前需要做的事情是逻辑和物理分析与重新设计。数据库有可能不需要大的再设计，但也有可能需要大修。

在任何一种情况下，当这个设计已解决时，数据库管理员能够推进到性能调整。然而，在重新设计后，最好首先把该应用程序投入使用并做一些性能监控工作。毕竟，正确的设计（或重新设计）一开始都能带来好的性能。很可能不再需要新的额外的调整（实际上，我发现事实就是如此）。

16.2 为什么进行调整

为什么要调整？你已经回答了这个问题。根据用户定义的准则，当前的数据库系统由于如下的理由而不能令人满意地运行：

- 不良的设计。
- 增长。
- 变化的应用需求（可能包含对可接受性能的重新定义）。

什么时候数据库的调整工作不能完全有效？当对于数据库来说是外部的但对于整个客户 / 服务器应用程序的性能至关重要的组件未能满意地运行时，如果没有相应的其他应用基础结构部分的调整，数据库调整或许就不会产生效果。除了大部分独立的批量数据库应用以外，大部分现代数据库系统是基于客户 / 服务器结构的。

对于后端数据库来说，属于外部的主要组件是后端操作系统（或操作系统），网络和客户操作系统。主要例子如下：

- 非常微弱的客户（PC）。
- 网络饱和。
- 非常弱的、已饱和的或不良调整的操作系统。

从客户（PC）所拥有的较老的 CPU（486或更低级）、相对小的内存（16MB或更小）或较低的存储器容量（1GB或更低）的意义上来说，它们或许是弱的。尤其是在应用对客户来说是重量的并由此需要一个胖的客户的情况下。胖客户（fat client）是指自己完成大部分应用处理的客户。瘦客户（thin client）显然是胖客户的对立面，它的大部分处理发生在服务器端。除非应用程序仅需要哑终端存取或瘦客户，否则弱客户将不足。然而，瘦客户将足以满足因特网计算范型，在该范型中网络浏览器是客户端的唯一要求。

通过改变定义，网络被认为已达到饱和。饱和意味着一个系统组件已经达到了最大的可能吞吐量。一个10Mbps的以太网在达到自身带宽的30%时就被认为是饱和的。网络专家用不同的方法处理饱和问题，包括重新分割本地网络，增加更多的网络段或者改用宽带网络介质类型，诸如100Mbps的光纤分布式数据接口（FDDI）。总之，你已经很好地踏上了数据库调整

之路。

就像客户一样，操作系统可以很弱。它仅相对于它必须支持的应用程序来说是弱的。一般而言，具有4个32位的处理器、256M内存和10G磁盘空间的现代数据库服务器被认为是弱的。就像网络一样，操作系统可以在各种情形下饱和。在重负载的情况下（许多并发用户或大量的数据移动），操作系统的CPU、输入/输出和内存能够达到饱和。

另一种方法是当组件饱和时，系统在那个组件上阻塞。因此，假如一台 UNIX机器的CPU是饱和的，例如达到85%或更高的利用率，那么这个UNIX系统就被认为是在CPU上阻塞，或简单地称为CPU约束。

最后，操作系统或许仅需要一些回报。例如，DSS仅需要把非常大的文件存储在磁盘上，系统管理员（SA）可以调整UNIX文件系统以便它把非常大的文件最佳存放在连续字节块中。

就像你将在第17章、第18章、第19章和第20章中看到的那样，调整数据库系统和调整操作系统缠绕在一起。对于系统管理员和数据库管理员来说这是一件好事情，因为在调整数据库系统或操作系统时不需要请求允许，你只需做它便是。然而，在实际中，数据库管理员经常与系统管理员以及资源分配和调整问题的管理程序发生冲突。我的建议是通过 Oracle的诊断工具收集适当的证据并使用这些图表证明你的资源需求是正当的。Oracle的诊断工具稍后在16.8节和第17~20章中讨论。

16.3 调整规则

现在列举你几乎能在任何一本性能调整参考书中发现的主要性能调整规则。

主要的性能调整规则如下所示：

- 1) 划分与克服。
- 2) 预分配、预取数据和预编译。
- 3) 优先次序。
- 4) 大块、块和批量。
- 5) 对应用进行适当地分段。

这些看起来熟悉的规则便是主要的性能调整规则。它们是在第3章中讨论过的物理设计规则，其中2号和3号颠倒了一下顺序。现在看一下这些规则如何应用到性能调整，而不仅仅是物理数据库设计中。

16.3.1 调整规则1

本基本规则是划分与克服。实际上，你想使用并行技术作为对瓶颈的补救。

看一下以下情况：假如性能监控显示数据库系统有一个较为集中的繁重物理读取和写入某一盘的过程，为了实现并行存取，需要将 Oracle的物理和逻辑结构分开。通常，一个应用有几个用户或者进程正试图存取那张盘上相同的表或表空间。必须被连接在一起的表经常存在于同一张盘上。这导致性能变差。

操作系统必须代表 Oracle为该表的一些数据块存取该磁盘，为第二张表再次执行本循环，然后为两张表的所有块重复整个过程，直到 Oracle已经把两张表连接在一起。硬件要做的事情是磁盘的读/写头必须读取第一张表的申请扇区（物理数据块），重新定位读/写头以读取第二张表的申请扇区直到找到为止，然后重复。磁盘的存取时间（用毫秒表示）包含搜索时间和

等待时间（转动延迟）。搜索时间指已提及的重新定位，读/写头必须沿着磁盘的半径向里或向外移动以寻找包含申请扇区的磁道。搜索时间控制等待时间或者用于磁盘旋转以存取扇区的时间。

重要的事情是连接同一张盘上的两个或多个表需要大量的搜索时间用于磁盘读取。作为一个数据库管理员，你的目标是减少搜索的次数。你想使争用减到最小并消除此瓶颈。要做这项工作，你将连接到不同磁盘上的表分开。这种方法同样适用于那些共存于同一张磁盘上的表，这些表尽管没有连接在一起，但它们能够被同时存取。结果是相同的：瓶颈。解决方法是相同的：分开它们。

通过这种方法，RAID对这个方案就像标准磁盘一样容易敏感。起初，RAID，尤其是1、3和5级（曾在第3章中讨论过）看起来不可能遇到同样类型的问题。明显地，去除几张盘中的一个单表（一个RAID卷）无疑会提高性能。然而，假如该表与一些必须和它连接或至少同时存取的表一起存储在同一个RAID卷中的话，你会遇到相同的问题：瓶颈。解决方法也是相同的：分开它们。

你仍然遇到瓶颈问题的原因起初不是很明显，但要仔细思考这个问题。在RAID卷的所有盘中去除所有的表块（存储在它们各自表空间的数据文件中）。尽管没有一张表在任何一张磁盘中作为一个整体存在，然而所有的表块共存于全部磁盘中。今后，你还会遇到同样的问题。如同标准磁盘设置一样，不是把已连接的或同时存取的表分散到不同的磁盘上，而是把这些表分散到不同的RAID卷中。

16.3.2 调整规则2

本基本规则是预分配、预取数据和预编译。只要可能的话，提前做此工作。

看一下以下情况：假定你有一个特别易变的数据库系统，它明显地且频繁地进行增长和收缩。事实上，作为一种应用类型，可以把它叫做VCDB。当在非增长长期访问数据库时，性能较为合理。然而，在它的增长阶段，性能非常差，尤其是在增长初期。

这里可能发生的事情是动态增长降低了联机系统的速度。在Oracle术语中，这被称为动态扩展（dynamic extension）。因为正在更新或插入行，所以Oracle必须扩展，或者在创建或改变时按照由存储子句所指定的分配表的下一个区域，或者采用表空间缺省值。在这两种情况下，从创建时刻直到最近的增长期，表都有足够的空间，然后必须扩展以适应它。当然，简单地将一张表在Oracle中进行扩展不是世界末日，事实上，它是一个事件的正常过程。

然而，当一张表经常扩展时，特别是为大量数据进行扩展时就会损害联机存取，这就成为一个大问题了。并发用户不应该遭受损失。对于这种需要频繁或大量数据扩展的数据库系统而言，最好使用预分配。你需要创建具有足够存储量的表来解决预期的最大峰值容量。在表（或表空间）的创建语句的存储子句中，设置INITIAL和NEXT扩展容量，然后设置MINEXTENTS，这样 $INITIAL + ((MINEXTENTS - 1) \times NEXT)$ 等于该表的最大预期峰值容量。

16.3.3 调整规则3

本基本规则是优先次序。首先解决最重要的问题以便在投资上获得最大的回报。

看一下以下情况：你有一个运行缓慢的批处理系统。当你执行产品运行时，该系统缓慢运行。在检查了组成应用的SQL代码后，发现有几个要重新编写的效率非常低的程序。要完

成这项工作或许会花费2个月时间。然后你分析性能监控统计数字，发现回滚显示了高的争用，这显然是因为你正在同时运行几个并发程序以帮助加速运行。通过增加更多的段，修改回滚，但是你的运行时间几乎没有改善。要做什么？

不管你如何调整后端，前端（或应用代码）对客户 / 服务器系统的总时间的使用进行支配，这不会引起网络和其他问题。你必须首先调整应用，然后调整数据库。当产品比以前运行得快时，投资便很值得。

假如此调整由于一些原因而不可行，那么其他诸如逻辑或物理重构的数据库努力可以作为临时措施来实施。但老实说，根本不应该这样做；你正在为了适应不良的应用设计而改变好的数据库设计。使用那种方法，你不会收获很大。

16.3.4 调整规则4

本基本规则是大块、块和批量。要处理的一组适当的事情聚集在一起并且彼此不会为资源产生竞争。

看一下以下情况：你的DSS经由因特网给外部分析员提供实时、只读信息。你把一个Web页做为数据库系统的前端。并发有时是中小规模的（最多有100个用户），但是不会引起问题。用户所抱怨的是从大表中选择时的吞吐量。

这有一个典型的例子。当从一张表中选择大部分或全部行时，优化程序必须选择任何索引上的全表扫描作为它的存取路径。假如这是用户所要求的话就不能避免，这种类型的存取是应用需求的一部分。你能做什么来加速它？你一开始可以在你的UNIX系统中将DB_BLOCK_SIZE设置为8K字节，然后可以把该值增加到16K或32K。块越大，读的时候效率越高。换句话说，对块的每一次读取你都可以得到更多的数据行，因此，通过增加大数据查找的块容量使读取的效率更高。

必要时你还可以考虑增加DB_FILE_MULTIBLOCK_READ_COUNT和DB_BLOCK_BUFFERS。同时读取更多的块和增加块缓冲区也会对DSS有所帮助，稍后你将在第18章和第19章中学习相关的更多知识。

16.3.5 调整规则5

本基本规则是对应用进行适当地分段。把各块应用分配到适当的地方（客户、中间层或服务器）。

看一下以下情况：你有一个执行命令输入和控制的客户 / 服务器数据库系统。然而，命令输入屏在PC机上运行缓慢。反应时间很长，应用程序似乎使客户负担过重，因为一个输入要花费几秒钟的时间（就反应时间而论是慢的）。

你首先需要检查应用是如何被分区的。大部分数据验证工作在前端代码中完成，这可能是出现故障的地方，因为它对大部分的代码行进行解释。大部分前端代码算法如下所示：

- 1) 把日志记录到数据库。
- 2) 从一些表中读取数据。
- 3) 把数据键入到屏幕上。
- 4) 等待前端对一些表中键入的数据进行验证。
- 5) 假如完成的话将日志取出，否则重复步骤2~4。

伴随于此的问题是事务在整个时间处于激活状态，它交互式地验证前端表中的数据。这个验证过程（步骤4）应该在服务器上非交互式发生，它属于那儿因为它直接处理数据。处理通常应该尽可能地定位在靠近它们操作的数据的地方，除非有一些强制理由（例如服务器功率不足）。对先前的应用进行分段的标准方式将它划分为 3 个主要的步骤：在事务中读取（输入数据），没有事务的情况下进行本地修改（交互式地）以及在事务中回写变化（输出数据）。应用分段（application segmenting）也称为事务截断（transaction chopping）。如果正确操作的话，它将缩短反应时间。

16.4 调整目标

有不同的方式确定性能调整工作的目标。数据库管理员会全部考虑它们。考虑到你的应用类型，这曾在第 3 章中讨论过，稍后在本章中将再次讨论（参见 16.6 节）。数据库系统可以用各种定量的方法分析，这也在第 3 章中也讨论过，稍后在本章的 16.7 节中继续讨论。这些中最重要的如下：

- 吞吐量 —— 每个单元时间完成的工作，以每秒钟的事务（tps）量表示；越高越好。
- 反应时间 —— 应用做出反应的时间，以毫秒或秒表示；越低越好。
- 背景时间 —— 应用占用的运行时间；越低越好。

在任何一个系统中，吞吐量和反应时间作为调整目标通常是互相对立的。如果反应时间长（坏），吞吐量或许高（好）。如果吞吐量低（坏），反应时间或许低（好）。

常识有助于挑选出这两个冲突的度量标准。多数的并发用户在一定时间内使用一个系统，每个用户很有可能比平时要经历更长的延迟，但是通过系统的事务数量将会更大。相反，假如你减少在某一个时间窗口中访问系统的并发用户数量，以在该时间内完成的全部事务量减少为代价，每个用户将会享受到更快的反应时间。

一般说来，依赖于应用的要求，OLTP 系统需要更低的反应时间或更高的吞吐量。DSS 希望低的反应时间。然而，DSS 也可能希望得到高的吞吐量（每个单位时间读出或写入的块的数量）。这种类型的吞吐量不一定对高并发性和低反应时间起反作用。一个批处理（生产）系统一般需要较低的背景时间。例如，每个人都希望按时完成工资单应用。

一般考虑两条主要的调整目标：

- 使你的投资得到最大回报。把你的时间和精力用于解决很有可能产生最大的利益的问题。
- 使争用减到最小。瓶颈的特点在于延迟和等待；尽可能地消除或减少它们。

最后，考虑如下的通用数据库调整目标：

- 将需要存取的块的数量减到最小；必要时检查并重写代码。
- 只要可能的话，使用高速缓存、缓冲区和队列弥补机电缺点（内存比磁盘更快）；预取数据。
- 将数据传输率（用于读写数据的时间）减到最小；快速磁盘、RAID 和并行操作有助于完成这项工作。
- 调度程序尽可能非竞争性地运行；它们可以并行，但大部分时间仍然是非竞争性的。

16.5 使用投资的回报

投资的回报（ROI）是一个用来查看性能调整过程的组织管理严密的、合算的方法。它帮

助你发现解决特定应用调整问题的最好方法。你已经看到 ROI作为一个主要的性能调整规则（优先次序）和作为两个主要的性能调整目标中的一个出现。现在你把它牢记在心并把它作为高级、逐步的方法用于性能调整。

你想按一定的顺序执行性能调整，希望这个顺序使你所花费的时间和精力获得最大的收益。在随后的 ROI策略中，注意到步骤 1~3相当于逻辑与物理设计，这已在第 1章与第 2章中讨论过。逻辑设计是 DBMS厂商独立进行的。因此，在处理逻辑设计时，没有 Oracle的具体说明（逻辑设计工具不包含在内）。你将再次访问 16.6节的应用类型并运用一些用于物理设计的具体的Oracle建议以及用于这些应用的性能调整建议。在后面的小节中，将更加仔细地研究这些Oracle的性能调整建议，查看调整应用，调整内存和调整输入/输出。

步骤4和5相当于调整应用，你将在第 17章中遇到。步骤 6和7相当于调整内存，它包含在第18章中。步骤8和9相当于调整输入/输出，它出现在第 19章中。步骤 10和11研究对于数据库来说比操作系统更外部的组件：网络与客户。步骤 12提供了一些高级的和不太常用的解决方案，这些方案仅在已经使用了标准方法（步骤 1~11）但想得到的性能收益没有实现的情况下尝试使用。第17、18和19章包含Oracle的具体说明以帮助你完成正确的性能调整。

现在，按照投资回报的逆序详细地回顾这些步骤。

16.5.1 步骤1：进行正确的逻辑设计

像第2章中所讨论的那样，你可以减少存储容量。实际上，这经常意味着更多的表，其中每张表所包含的行数更少。这又意味着快速查找的能力。不考虑查找技术，存储的且必须完全查找的行越少，发现你要寻找的东西就越快。

16.5.2 步骤2：进行正确的物理设计

在第3章中，你看到一个通用的硬件布局。在稍后的 16.5.12节中，你将看到一些对于此通用硬件布局的具体应用方法。

16.5.3 步骤3：如果必要，进行重新设计

假如在数据库生命周期的任何阶段，设计出现问题的话，在资源和管理允许的情况下，考虑重新设计它。有时不必进行大范围的调整纠正一个数据库的最好方法是重新分析与重新设计。例如，在没有初始设计、有一个仓促设计或多个数据库需要合并的情况下考虑重新设计。

16.5.4 步骤4：书写高效率的应用代码

一个程序仅能和形成它的算法运行的一样快。假如一些 SQL代码使用效率低下的查询或排序例程，不管Oracle优化程序尽多大努力，应用程序仍然缓慢地运行着数据库。假如你投入一项工作中，代码已经编写但是写得很差，那么请考虑下一步，而不是试图每次修理这部分代码。

16.5.5 步骤5：如果必要，重写代码

假如应用代码效率低下并且资源和管理允许的话，那么就重新分析并重新编写更加有效

的代码。了解Oracle优化程序和用于更有效代码的 Oracle SQL重写规则。数据库应用与任何一个普通的软件代码段没有区别，它可以被编写得很好也可以很差。在设计结束后，应用是一个获得投资回报的最大的调整机会。毫无疑问，时间和精力应该花费在这里。

16.5.6 步骤6：调整数据库的内存结构

在第6章中，你看到SGA包含了所有的可以调整的、共享的内存结构。当需要时，Oracle通过调整数据库缓冲区缓存提供真正的增长。这个高速缓存在读出或写入时缓存所有由Oracle存取的数据库块。还有，共享池通过库缓存组件保存SQL代码并通过适当命名的数据字典缓存保存数据字典。更多的回滚由上述的数据库缓冲区缓存保存，但它不是独立可调的。重做日志缓冲区是SGA中分别定义的可调区域。所有这些内存结构所具有的充足的容量减少了争用的发生并弥补了机电的缺点。

16.5.7 步骤7：如果必要，调整操作系统内存结构

在Windows NT中，不一定像在诸如VMS和UNIX那样的大型多用户操作系统中那样要通过内存的使用进行调整。在UNIX中，已命名的Oracle用户（通常是“oracle”）一般必须有不受限制的外壳进程限制。这必须通过人工或在一个注册脚本中完成。UNIX交换区域充当一个操作系统临时存储区、用户临时存储区和操作系统虚拟内存后备存储区。因此，在UNIX中，交换区域可以成为一个瓶颈，尤其是在数据库服务器上。

在UNIX系统的Oracle数据库中，系统管理员和数据库管理员必须协同工作以便UNIX提供足够的共享内存和信号量给Oracle进程足够的空间去有效地运行。附录A和附录B进一步讨论这些重要的主题。

16.5.8 步骤8：调整数据库输入/输出

数据库输入/输出受到RDBMS和操作系统内存结构的影响。但是步骤8想要做的事情是通过重新定位数据库逻辑和物理结构调整输入/输出以减少争用。当然，数据库管理员应做到这一步，尤其是在他以正确的顺序执行这些步骤的情况下。

例如，如果你到达这个调整点，你就已经调整了数据库缓冲区缓存。现在你把注意力重新集中在物理设计上。也就是说，你进行更多的物理上的再设计工作，如果必要的话，把数据库输入/输出牢记在心。你的物理再设计比以前完成的更特别。你想要在所希望的方向上提高输入/输出时间评估，例如减少有繁忙插入的应用程序的背景时间。

16.5.9 步骤9：如果必要，调整操作系统输入/输出

操作系统通过所有的进程（包括诸如DBWR和LGWR的Oracle后台进程）完成所有的读写请求。操作系统缓冲这些请求，执行读写操作，然后在完成时进行确认并把数据返回给该进程。文件系统是包含所管理的文件的元数据的数据结构，例如每一个文件起始扇区的地址、扇区长度、目录树位置、属性（例如权限、大小和时间标记等等）以及其他信息。

在UNIX中，文件系统也有它们自己的逻辑块大小，它大于或等于物理块容量（512个字节），缺省值通常是8K字节。Oracle块容量至少是8K字节或8K字节的倍数，例如16K字节。

其他重要的操作系统以及Oracle输入/输出调整问题包括提前读功能、异步输入/输出、多

块读取、RAID数据条大小、磁盘几何结构、控制器和许多其他问题。假如数据库管理员不是系统管理员（事实经常是这样），那么他必须与系统管理员协调工作以适当地使 Oracle和操作系统输入/输出参数相结合。

16.5.10 步骤10：如果必要，调整网络

正如在16.2节中所讨论的那样，一个饱和的网络能够抵消数据库调整带来的改进。数据库管理员必须保证，如果必要的话，通过使用诸如网络监控软件的方法，他的客户 /服务器系统的整体应用性能不会因为网络负载或其他的网络小毛病而过度地受到损害。网络管理员和系统管理员一样必须和数据库管理员密切配合来解决这些问题。

16.5.11 步骤11：如果必要，调整客户端

局域网（LAN）管理员和通用网络管理员（他们有 WAN职责）对网络客户或PC的硬件和软件负责。这些管理员连同数据库管理员必须正确地决定客户的大小和配置客户，如果必要进行升级，在可接受的性能准则下实现客户 /服务器数据库应用程序的运行。例如，当 Oracle已经被调整并且可以在亚秒时间内服务于一条数据库查询（当通过网络从客户端收到时）时，无论从客户到服务器的网络延迟大约是几秒都没有什么关系，反之亦然。

16.5.12 步骤12：如果所有的调整都失败了，考虑更多的特殊解决方案

“特殊解决方案”包括Oracle的多线程服务器（MTS）、事务处理（TP）监控程序、Oracle的并行查询和其他的并行能力、Oracle聚簇、Oracle的位映射索引、MPP机器、固态盘、内存驻留（RAM）盘、硬件加速器和排队系统。

因为这些解决方案不适用后面的小节，所以本节简要地讲述了它们中的一部分。

1. Oracle的多线程服务器（MTS）

正如第3章中所讨论的那样，Oracle的多线程服务器是一个伪多线程解决方案，用以帮助对大量的并发用户或存取单个数据库的程序进行统计。多线程服务器本质上是 TP监控程序的低成本形式，其目标是实现大量的并发用户并对它们进行服务，因此使吞吐量（不是反应时间）达到最大。然而，多线程服务器尽可能地把进程全局区域（PGA）内存移动到SGA中，因此共享更多的内存并且减少维护每个进程的系统开销（它的状态信息）。

2. 事务处理（TP）监控程序

相比之下，事务处理监控程序走得更远，它不维护多线程服务器包含的整个进程状态信息和由普通操作系统或RDBMS所做的永久连接。每一个进程的系统开销进一步减少。操作系统和RDBMS的典型并发用户限制大约是 200个用户。这意味着在大部分操作和数据库系统达到这一数字之后，系统性能急剧下降。这被称为系统失效（thrashing），它是由过度的多程序或用户导致的症状。

事务处理监控程序能够支持大约 1000个并发用户，而在以前可能仅仅只有几百个用户。这就是为什么事务处理监控程序位于 RDBMS的前面的原因。一般认为它们存在于后端（RDBMS）和前端（客户或 PC）之间的中层。在本书编写时两个主要的商业产品是 Novell/BEA公司的Tuxedo和IBM/TransArc公司的Encina。

3. Oracle的并行查询选项（PQO）

像在第3章中提及的那样，Oracle的并行查询（PQ）通过划分与克服规则可以加快查询的速度。然而，Oracle也允许并行索引创建、并行 SQL*Loader装入和并行导入/导出。所有这些功能通过几个因素减少了实际的背景时间，这对数据库管理员的工作来说大有好处。尽管后面的并行操作趋向于主要帮助数据库管理员，因为他完成（或将要完成）这些任务的大部分，但并行查询可以使任何用户受益。

4. Oracle聚簇

借助Oracle，你可以使用索引簇或哈希簇将经常在一起连接或存取的两个或多个表结合在一起。聚簇本质上是在物理层将子表合并到它们的父表中。换句话说，父表的主关键字不仅为某一个主键值存储唯一的行数据（和通常情况一样），而且存储由子表的行组成的重复组，这些子表通过它们的外键对值进行引用。因此，数据在逻辑层保持规范化，但是被认为在物理或存储层不是这样的。这完全可以接受并且没有回避或违背任何关系模型规则或标准化理论。簇可以被索引（B*树）或哈希处理。第19章将进一步讨论这个问题。

5. Oracle的位映射索引

Oracle还提供位映射索引，该索引一般与OLAP系统有关，但是经常与其他系统一起使用，尤其是DSS。位映射索引能很好地代替基于具有不同值的小子集列的常规索引（B*树）。它们比常规索引占用更少的空间并且可以高效地使用复杂的 where子句。因此，它们是DSS和OLAP系统的首选，因为它们有大量的数据和较高的分析要求。假如一个系统适度地或频繁地进行更新，那么位映射不是好的选择。

这些索引的工作方式是一列中的所有不同值由具有0和1的索引列进行转化，用每一行来表示那一列是否等于给定的特殊值。表16-1显示了一个叫做STUDENT的简单表，它有2列，分别是STUDENT_ID和SEX，并有3行。该表告诉你位映射索引如何工作。你将在第17章学习有关位映射使用和实现的知识。

表16-1 STUDENT表的位映射索引

STUDENT_ID	SEX	SEX= 'M'	SEX= 'F'
719250751	M	1	0
298674071	F	0	1
347691027	M	1	0

6. 大规模并行处理机（MPP）

大规模并行处理机和SMP机在第3章中涉及过。这两种机器类型都有多个处理器并且都提供了巨大的计算能力。除了结构上的不同，两种机器之间性能上的不同只是程度上的差别而不是性质上的差别。大规模并行处理机提供了更多的处理器（几百或上千），而SMP只提供了一百个或更少的处理器。中档和低档的SMP已随处可见，而高档的SMP和大规模并行处理机却仍未普及。

现代UNIX和UNIX变体的多处理器可以安全地超过大型机的性能。这对大多数数据库系统意味着增加了并行计算能力。这能够和诸如并行查询这样的操作在一起和谐地工作。不幸的是，大部分数据库系统不能利用所有的这些功能，因为它们可能受到输入/输出的限制而不是处理器的限制。更有可能的是，组成RDBMS内核的实际软件结构（例如多线程可伸缩性）仅能支持20个左右的处理器，超过这个限制后不能发挥巨大的性能作用。目前，在Oracle、

Sybase和Informix中的确是这样。

存在不同的硬件解决方案，有些基于相对过时的技术，而有些基于较新的技术。较老的方案中包括RAM盘。假如一个操作系统允许这种类型的排列，你可拥有一部分充当分离磁盘的核心（和虚拟）内存。当然，假如电源断电，除非内容拷贝到一个更为永久的存储器（即磁盘）中，否则该盘将丢失。有些DOS版本允许此项操作，对于小型的xBASE数据库（例如dBASE）可以工作得很好。UNIX仍然允许这种操作。特别是，Sun Solaris允许实际是UNIX RAM盘的临时文件系统的配置。

总之，这些盘的使用对数据库系统来说是有问题的，不仅仅会遇到硬件失效或电源丧失，你还会遇到一种情况：不管什么理由，无法使用RAM盘并且RDBMS必须对那张盘中的数据做时间标记。结果：数据库需要恢复！这是一个坏消息。然而，假如系统证明是比较可靠的，那么恰当地使用这张盘可以使性能有所提高。在Oracle中，一种可能方案是把TEMP存储在一张RAM盘上。

7. 固态盘

尽管固态盘的想法和原形已经存在了大约15年，但相对来说它还是比较新的。它们一直没有得到商业上的使用，直到最近5年情况才有所改变。硬件SMP厂商（例如Sun公司）和第三方的供应商目前提供这些盘。

固态盘实质是一块专用的永久内存卡。它的明显优点是消除了机电缺陷，因为有了要分解的磁盘存取时间。存取时间接近于内存速度。缺点是价格偏高、所有权/兼容性问题以及这些产品缺乏较长的产品周期。后面这个问题表明该技术仍然相对比较新并且还不可靠。

8. 硬件加速器

硬件加速器是专用的输入/输出卡，它提供了充足的高速缓存空间、专用的固件以及有时是专用的微处理器用来加速它们支持的输入/输出通道。它们已经比固态盘早几年投入商业使用，但是并不很成功。它们的价格较为合理，但也存在所有权/兼容性问题。因为它们不是替换固态盘而仅是扩充它们，所以它们的技术稳定性不如固态盘那么重要。总之，在合理配置后，它们已被证明相当可靠。Database Executor (DBE) 就是一个例子。

9. 排队系统

排队软件，例如IBM公司的消息队列中间件（Message Queuing Middleware, MQM）系列，能够提供异步消息传送能力。这种软件有时也称为面向消息的中间件（Message Oriented Middleware）或MOM。该软件所完成的工作基本上是队列所要做的；以先进先出方式（FIFO）缓冲某些东西。它有时替代或辅助TP监控程序以帮助应付在维护高并发系统中的大量用户时的系统开销。

除了FIFO缓冲以外，这种类型的软件经常可以保证把客户请求发送到服务器。因此，通过既缓冲了网络系统中的客户/服务器请求，又防止了对同一请求的不必要的二次传送，这种软件间接地加速了处理过程。

16.6 再看应用类型

第3章中涉及到3种主要的应用类型（OLTP、DSS和批量），还提及了一些较少使用的类型（OLAP、VCDB）。现在重新考虑Oracle对于这些不同应用类型的调整建议并看一些通用硬件布局上的新的、特殊应用方法。首先，对两种主要的应用类型OLTP和DSS进行对比。

16.6.1 OLTP问题

OLTP系统具有高的并发性（大量的交互式用户）并且是更新密集型的（含有大量的插入、更新和删除语句）。因为OLTP应用可以根据插入和删除而较大地增长或收缩，所以支持大部分易变的表和索引的表空间中的区间应被预先分配到它们最大的期望容量。在这种情况下，动态扩展仅损害性能。基本上，在达到那个最大值以前你就必须计划容量和执行查找。

你必须有足够的重做和回滚来处理事务的要求。假如事务相对短或者修改相对少量的数据，你需要许多较小的回滚段。因为你有许多并发用户，所以你需要许多回滚段来把争用减到最小。假如系统有紧急需要的话，你应有许多重做日志来处理频繁发生的检查点。回滚段和重做日志将被分隔并且可以使用 Oracle 的诊断工具来测定其大小（参见 16.8 节）。

与把这个逻辑放置在应用代码中相比，数据库检查约束和引用完整性约束就计算时间而论花费更少的系统开销。总之，确保应用代码尽可能地重复使用以促进共享。当此代码被多次调用时，Oracle 使用共享池的库缓存来重复使用它。对于 OLTP，通过使用绑定变量而不是直接量可以减少应用进程的系统开销。假如语句中有少量符号要分析，给出少量分析可以更好地共享语句。

此外，在 OLTP 系统中，需要更多的索引，因为你正试图用近乎随机的方式存取表中相对少量的数据。然而，你不想要让索引太多以致于实际上降低了应用的速度。一条规则是，在你的所有主键和外键中建立索引并保守地增加其他东西。记住，更新和重构索引代价非常大，高级 DML（插入、更新和删除）活动触发这种索引重构的排序。

正如 16.5.12 节中提及的那样，当你的数据库系统在极端过载的情况下或需要以每日为基础时，可以使用多线程服务器（MTS）选项。尽管多线程服务器的主要目标是提高高度并发系统的吞吐量并且许多 OLTP 系统的并发度是可变的；然而，最大化吞吐量使反应时间延迟变长。所以使用多线程服务器是一种策略。也就是说，假如高吞吐量给你带来了高的性能，那么就使用它。假如你需要快的反应时间，就不使用它。

16.6.2 DSS问题

使用 DSS 时，数据库经常是大的、历史的和只读的。因此，查询（选择）是 DSS 活动的目标。Oracle 的并行查询能够加速缓慢查询的速度，这些缓慢查询可以通过一些物理（表空间和数据文件）重构并行执行。像以前所讨论的那样，希望 Oracle 块一次读取尽可能多的行。因为 DSS 查询一般触发本质上连续的整个表扫描，所以希望通过读取多个连续的块使你的块读取达到最大值。把 DB_BLOCK_SIZE 和 DB_FILE_MULTIBLOCK_READ_COUNT 参数设置得尽可能高些。这些参数设置将在第 19 章中进一步解释。

然而，这里有一些警告：

- 确保你的区间大小是 DB_FILE_MULTIBLOCK_READ_COUNT 的倍数。
- 确保 DB_FILE_MULTIBLOCK_READ_COUNT 是 DB_BLOCK_SIZE 的倍数。
- 假如你正在使用 RAID，确保你的 RAID 数据条大小是区间大小的倍数并且上述的两条仍然满足。

你有 3 种方法使用 DSS 索引。因为大部分查询返回大表的全表扫描，你也许想知道为什么要使用 DSS 索引（事实上，方法 1 中并不使用它们）。3 种方法如下：

- 方法 1：根本不使用任何索引，尤其是在你没有任何高度选择性查询或买不起用于在每

张表中创建常规索引的辅助存储器的情况下。

■ 方法2：用一个小的DSS查询率保持那些具有选择性的表中的少量索引。

■ 方法3：像16.5.12节中所讨论的那样，使用位映射索引。因为大部分时间里，DSS查询并不趋向于高度的选择性而且即使在常规索引存在的情况下也将触发整个表搜索，所以把位映射索引作为一种可供选择的方法。

当然，在实际中数据库管理员会不时地使用所有这些方法。

不要在DSS应用代码中使用绑定变量。这直接与OLTP的建议形成对照，因为在OLTP下，你希望最小化应用进程开销（语法分析）。然而，随着相对长时间地运行DSS查询，语法分析占用整个查询时间的比例会更小。你想要保证优化程序选择最好的存取计划。假如使用绑定变量，优化程序不能调用它所存储的统计信息（通过ANALYZE命令）以选出存取数据的最好方法。任何本应当通过直接量获得的选择性将不存在。这些直接量有助于减少中间表和返回的最后输出的规模。

在OLTP下，希望有许多相对较小的回滚段。伴随着DSS和长时间运行的查询，类似于长时间运行的批量生产系统，应有一到两个大的回滚段。这仅仅实现一致性读取和大量回滚数据的并发视图。假如DSS是100%只读，这不会成为一个问题。实际中，我发现没有100%的只读系统。

16.6.3 OLTP与DSS都需要考虑的其他问题

OLTP与DSS都需要附加考虑的问题是Oracle的块级参数INITRANS和PCTFREE。对于OLTP，因为每个块必须要有增长和变化的空间，所以把PCTFREE设置得相对低些（或仅仅把它设置为缺省值）。对于大多数的只读DSS，把PCTFREE设置得高些，因为块中的数据通常保持不变。对于OLTP，希望把INITRANS设置得相对高些，因为多数并发用户会触发动态事务头槽扩展，随之而来的是块级重构，这可能逐步影响到区间级。

对于DSS，不太可能出现由于锁定而必须等待的并发用户，因为大部分事务是选择，更不用说DSS存储的大量数据块了。因此，假如你的DSS也支持适当级别的插入、更新或删除的话，那么把INITRANS设置为中等水平。另外，它也可以保持缺省值。

16.7 理解基准测试

基准测试（benchmark）是代表一个应用实际工作的准则。在第3章中提到的作为事务分析结果的数量测量也可以用作基准测试。在基准测试上运行的应用的性能通过一些专门选择的数量测量方法来测定。当基准测试运行时，你通常已经预先设置了条件或配置以便当在运行中的某个时刻改变一个变量时（独立变量），你可以保留常量（初始条件或从属变量）。这使你能够分别研究这些变量的影响。因此，基准测试仅是科学或实验方法的一个变种。假如正确处理的话，你可以对原因和效果，或者最起码是效果的相关性进行评定。假设是改变从属变量（例如，块大小）将影响到独立变量（例如，吞吐量）。

基准测试可以提供各种便利。通常，基准测试与仿真有关。系统仿真是对系统的实际工作进行模拟的功能原型，用来推算系统的性能或测试系统的功能。然而，基准测试也提供运行基线的能力，基线是你在发布一个产品系统之后或开始性能调整以前所采取的相关数量基准。它给数据库管理员一个用于比较的基础，用来测验他或她的性能调整是否已经带来了改进。

最著名的数据库系统基准测试是由事务处理性能委员会（TPC）提供的。这个委员会已经制作了大量的基准测试，它创建的第一个基准测试是具有代表性的银行出纳员操作（TPC-A与TPC-B）。对于这些基准测试，要创建一个小的数据库，包括一张银行表、一张转帐表和一张帐目表。所执行的操作包括对这些表的更新和对历史记录表的插入。性能由每秒钟的事务量（tps）来测定，每秒钟的事务量是吞吐量的一种数量度量。因为TPC-A与TPC-B在几年前就已经发布，因此事务处理性能委员会已经推出较新的基准测试TPC-C与TPC-D，TPC-C与TPC-D针对不同种类的应用进行基准测试。

TPC-A（1989）使用一个相对简单的、多任务的事务对一个具有高并发性和重要的输入/输出的OLTP环境进行测试（执行不止一个的选择、插入、修改或删除的事务被称为是多任务的）。TPC使用事务类型（transaction type）这个术语来指代一个特殊的多任务事务。在TPC-A中，事务有多个更新和一个插入。

TPC-B（1990）除了取消了高并发性需求以外，基本上与TPC-A相同。它被用作数据库耐压测试（stress test），基本上仅测试重要的输入/输出。然而当使用仿真软件中的虚拟终端时，这个差别是模糊不清的，所以这两个基准测试经常被放置在一起或彼此替代。

TPC-C（1992）是一个OLTP基准测试，它是5个不同的、更复杂的事务类型和9个数据库实体的混合。比较起来，TPC-A和TPC-B使用单一的事务类型和4个数据库实体。其业务目前是基于一具有一个定单记录系统的供应商数据库。

TPC-D（1995）是一个DSS基准测试，主要用于繁重的、现实领域的事务。它包括17个针对大量数据的复杂查询。这是目前最新的TPC基准测试。与较轻负载的TPC-A和TPC-B以及相对中等负载的TPC-C完全不同的是，TPC-D适合于长时间运行的重负载查询。

当基准测试在一个无偏的独立环境下执行时，这些基准测试提供了一个在RDBMS软件厂商之间进行比较的基础。数据库系统可以在已调整和未调整的情况下比较。应用是相同的，如果网络 and 平台（硬件和操作系统）是相同的话，那么剩余的唯一可变物是RDBMS软件。通常没有两个厂商会在相同的平台上进行相同的基准测试（至少在同一时期内）。

使用好的仿真软件（例如LoadRunner）可以产生虚拟用户和进程来模拟在基准测试期间数据库系统的负载。并发用户的数量可逐渐增加，同样每个用户的事务数也可以增加。总之，用这两种方法，事务的总数都会增加，你可以有效地创建并观察数据库耐压测试。当确认了运行并且收集了统计数字（tps）后，你可以研究ROI策略并在必要时使用性能调整规则来解决常见的调整问题，即使是在一个仿真系统中也要这样做。

16.8 使用Oracle诊断工具

主要的Oracle诊断工具如下所示：

- SQL_TRACE与TKPROF。
- EXPLAIN PLAN。
- 使用V\$动态性能视图和服务器管理器行方式。
- 服务器管理器监控（GUI）。
- 企业管理器的性能包。
- utlbstat/utlestat与report.txt。
- 第三方产品。

16.8.1 使用SQL_TRACE与TKPROF

要调整应用, 你可以使用 SQL_TRACE与TKPROF。为了得到定时信息, 你必须设置 init.ora参数TIMED_STATISTICS=TRUE。时间可以定到0.01秒。实时数据库系统必须使用代替的定时方法, 例如编写你自己的定时器或者用数据库系统内部的实时操作系统时钟采样。你可以随意地将 USER_DUMP_DEST设置为你所选择的目录, 也可以将 MAX_DUMP_FILE_SIZE设置为你希望跟踪文件增长到的最大字节数。既可以在系统层(对于所有的会话)通过 init.ora参数设置 SQL_TRACE=TRUE, 也可以在会话层通过 ALTER SESSION SET SQL_TRACE=TRUE; 设置SQL_TRACE=TRUE。

执行这个应用, 如果需要, 把 SQL_TRACE设置为FALSE。实际上, 在短期不会为了计时的目的而再次运行这个应用。使用 TKPROF格式化跟踪文件(从命令行运行 TKPROF工具)。收集到的统计数字包括查询执行周期的组成时间(语法分析、执行和获取数据)以及逻辑与物理读取。SQL_TRACE和TKPROF的使用、语法和解释将在第17章中详细讨论。

16.8.2 使用EXPLAIN PLAN

另一个应用调整工具是 EXPLAIN PLAN。你必须运行 utlxpln.sql来创建 PLAN_TABLE, 或者你自己交互式地创建它。不用实际执行 SQL语句, EXPLAIN PLAN FOR <SQL语句>; 就能解释该 SQL语句的执行计划(这类似于其他翻译语言的 NO EXEC选项)。不管有没有 SQL_TRACE和TKPROF, 你都可以使用 EXPLAIN PLAN。我建议你先应该检查一下代码(把它读一遍), 接下来运行SQL_TRACE与TKPROF, 最后假如你仍然对运行较差的应用感到有困难, 那么使用 EXPLAIN PLAN。连同Oracle优化程序是如何工作的, 这些内容可以在第17章中找到, 对Oracle优化程序的介绍有助于解释 EXPLAIN PLAN是如何工作的。

16.8.3 使用V\$动态性能视图

要有助于调整内存(参见第18章)与调整输入/输出(参见第19章), 所有的Oracle产品(例如SQL*DBA, 服务器管理器与企业管理器)都依靠 V\$动态性能视图。这些视图被划分为实例、数据库、内存、磁盘、用户、会话和争用几个方面。它们基于内部 X\$库基础表。DESCRIBE V\$FIXED_TABLE并从V\$FIXED_TABLE表中选取你想要的列以得到 V\$视图的列表。一般不直接查询 X\$表。它们的名称和内容在不同版本之间以及同一版本内是不同的。

V\$视图之所以被称为动态的是因为它们在实例启动时被填充, 在实例关闭时被截取。 V\$视图(和 X\$表)还形成标准Oracle调整脚本utlbstat/utlestat的基础, utlbstat/utlestat使用SQL脚本查询V\$视图并格式化返回的输出值。因此, 假如 utlbstat/utlestat没有为你提供你所需要的东西, 你可以使用服务器管理器与 V\$视图辅助或者取代那些工具。所有的 Oracle平台, 不管是老的还是新的, 它们的共同特性是 SQL脚本、V\$视图以及SQL*DBA或服务器管理器行方式(例如, 在UNIX上是svrmgr1, 在NT上是svrmgr23.exe)。这意味着假如你已经开发了你自己的性能脚本, 你可以在这些命令行工具中运行它们。除此以外, 还有 SQL*Plus。

16.8.4 使用服务器管理器监控器

由于企业管理器的出现, 服务器管理器监控器正变得越来越过时。它提供了与性能有关

的几个屏幕。这些屏幕监控多线程服务器、逻辑与物理输入 / 输出、处理信息、锁与闩、共享池信息和回滚信息。屏幕被实时更新，并且它们基于 V\$视图与X\$表。

16.8.5 使用企业管理器性能包

企业管理器性能包极为有用。它提供了几个组件，这些组件可以节约命令行方法与脚本的很多时间，对有经验的数据库管理员或那些准确知道所要查找的东西的人来说更是如此。企业管理器使你能够很快地接触到那条信息。统计数字基于 V\$视图和X\$表，但是被重新格式化为更容易理解的GUI表格形式。

性能包中的组件有助于分析你的逻辑与物理设计。它们还通过跟踪监控锁、各种性能问题（吞吐量、重做、回滚、输入 / 输出、内存等等）、关于资源消耗的顶端用户会话、表空间存储（数据文件、存储碎片等）以及应用事件。

16.8.6 使用utlbstat/utlestat与reportl.txt

到目前为止最常使用的 Oracle 诊断工具是 utlbstat/utlestat 对。一个数据库管理员在运行他的应用或仿真以前先要运行 utlbstat。utlbstat.sql 脚本构造了必要的开始表用以收集和存储性能数据。然后数据库管理员运行 utlestat.sql，utlestat.sql 构造了结束表和差别表用以计算 utlbstat 运行与 utlestat 运行之间（实际上是应用持续时间）的性能差异，格式化输出数据（包括注释和一些解释）并把它写入缺省文件 report.txt 中。该文件必须由数据库管理员直接或间接地解释（通过采用一些给定的输出值并把它们作为简单公式的输入使用）。

数据的解释意味着把这些最终数值与建立的准则作比较，把 ROI 策略牢记在心，对于那个给定的性能区，把发现的结果分类为可接受的或不可接受的。report.txt 中的输出几乎直接或间接地涉及到所有的基础。所以一个数据库管理员只须应用 ROI 策略、已建立的准则以及他的应用知识并确定是否可以接受内存性能和输入 / 输出性能。

16.8.7 使用第三方产品

本章对主要的 Oracle 诊断工具做了概述。目前还有一些第三方性能监控工具并且在已建立的 Oracle 顾客群中它们占用相当大的市场份额。这些工具包括来自 BMC、Platinum 和 Compuware 公司的产品。你将使用 utlbstat/utlestat 和 Report.txt 作为第 18 章和第 19 章的诊断工具。你将在下一章中使用 EXPLAIN PLAN 和伴随 TKPROF 的 SQL_TRACE。

第17章 应用调整

本章要点：

- 动机
- 理解优化程序
- SQL TRACE与tkprof
- 理解EXPLAIN PLAN
- 鉴别典型问题
- 重写查询
- 介绍ORACLE 8的新的索引特性

17.1 动机

调整应用实际上相当于调整你的应用中的 SQL 语句。假如你的应用有一个图形用户接口 (GUI) 前端，调整应用仍然相当于调整 SQL 语句。换句话说，所有 GUI 层的操作可以被映射到具有不同复杂度的一条或多条 SQL 语句中 (SELECT、INSERT、UPDATE、DELETE 等)。总之，假如可能的话，你的投资返回值 (ROI) 策略告诉你必须首先调整应用，可能的话再调整数据库的所有方面。

主观估计表明应用决定着大约 80% 的总系统 (应用加数据库) 性能，而仅为数据库留下 20% 的系统性能。我同意此观点。然而，我不太同意该观点中的数据。在我的经历中，我见过一些系统中数据库影响大部分性能。

对于数据库管理员来说，这个观点真正的意思是系统性能是相对的、与状况相关的、应用特定的。这听起来很熟悉，尤其是在关于应用类型的讨论中。然而，有一条绝对规则永远有效，我全力提倡数据库管理员应关注第一条调整规则：

“应用应当首先被调整。”

这条规则有如下两条理由：1) 目前存在的大多数数据库系统把它们的大部分性能归于它们的应用。2) 即使这条规则不适用于你的系统，作为一个数据库管理员，你也必须对系统的应用类型进行分类。这意味着你必须检查应用，具体地说，检查事务的类型——访问代码和 SQL 代码。从数据库访问的观点来说，即使你的应用代码尽可能高效地编写，你仍然必须理解这个程序如何对你的数据库产生影响。

你如何完成检查应用的任务？假如从代码的行数或分离的程序单元 (模块) 的数量上来说应用相对较小——那么人工检查每一件事情是可能的。相反，假如应用很大，那么完全人工检查是不可能的。规则 3——优先次序 (参见第 16 章) 给了你方向。你可以把精力集中在那些消耗了大部分资源和时间的模块或代码段上。

首先，你试图维护 20% 的应用代码，这些代码决定了 80% 的系统性能 (假如你有时间和人员，那么你能够在剩余的代码上工作以榨取余下的性能潜力)。你实际如何做呢？17.3 节讨论了你所需要的 Oracle 工具。但是首先，下一节给你了使用那些工具的必要的背景知识——理解

Oracle优化程序以及它是如何工作的。

17.2 理解优化程序

优化程序 (optimizer) 是负责优化的一段软件以及一部分 RDBMS, 或者是对一条给定的用于访问数据的 SQL 语句的最有效表达方式。要做到这一点, 优化程序选择一系列访问路径, 该路径为 Oracle 访问数据提供了最快的路线, 然后建立基于那些访问路径的执行计划。访问路径是一条指向数据的物理路线。执行计划是沿着已选定的访问路径的一系列 Oracle 可执行步骤。

Oracle 优化程序中的技术, 如同许多其他的 RDBMS 厂商一样, 在过去几年中已经取得了显著的进展。具有讽刺意义的是, 优化程序仍然不总是能提供访问数据的最佳方法。在较老的版本中, 程序员必须留意如何编写 SQL 语句。SQL 语句里的排序能够强烈影响 SQL 语句的执行, 所以程序员不得不执行许多人工优化。同样, 数据库管理员必须对这种情况有所了解。优化程序越好, 用户、程序员和数据库管理员的优化责任就越小。即使在当前的 RDBMS 优化程序领域, 程序员仍然需要掌握优化技巧并在可能时运用它们。然而, 程序员不必象以前那样对优化程序很内行。数据库管理员仍然应当知道优化程序技术、优化规则以及正确的优化程序用法。

优化程序有两种风格:

- 基于规则 —— 一个基于规则的优化程序选择的访问路径是基于不变的、RDBMS 厂商特定的排列次序, 其中这些访问路径是到数据的最快方法。尽管这个排列是厂商特定的, 然而在不同厂商的优化程序的输出之间比较和映射类似的访问路径相对比较容易。
- 基于成本 —— 一个基于成本的优化程序选择的访问路径是基于一些内部存储的数据分布统计数字, 这些统计数字通常放在 RDBMS 数据字典中。一般, 数据库管理员必须周期性地运行一些 RDBMS 命令来维护这些统计数字。

Oracle 7.x 和较高级的优化程序可以在同一个优化程序中既提供基于规则的也提供基于成本的优化能力。Oracle 6.x 优化程序提供基于规则的优化, 但是它没有一个全自动的基于成本的优化。所有这些版本为程序员提供了一个称为提示 (hint) 的替换值。从优化程序不总是遵循该提示的意义上来说, 这不是一个真正的替换值。可以认为一个提示就是对优化程序的一个建议。假如不存在语法错误, 优化程序通常会采用它。一个提示被内置存放, 在 SQL 语句代码中有直接意义。清单 17-1 显示一条 SELECT 语句, 建议优化程序使用正在选择的表中的 INDEX。

清单 17-1 一个具有提示的查询的例子

```
SQL> SELECT /*+ INDEX */ EMPLOYEE_ID  
2> FROM EMPLOYEES  
3> WHERE EMPLOYEE_ID = 503748;
```

注意提示会立刻跟踪 SQL 命令 (SELECT)。提示的注释像任何常规的 SQL*Plus 注释一样, 以 /* 开始, 并且马上跟着一个加号 (+), 这样以 /*+ 形成了一个提示注释的开始。通常以 */ 来结束提示注释。还有, 考虑一下, 假如存在不止一条索引的话, 特别是假如由于某些原因多条索引存在于 EMPLOYEE_ID 上, 优化程序如何知道要使用哪一条索引。在以前的例子中, 仅仅规定了 /*+ 索引 */ 并且假定只有一条索引 (主键) 存在于表中。因此, 对于这样的列表不

存在模糊。在存在模糊的情况下，你需要在提示中包含表和索引名以便给优化程序提供具体信息。提示在接下来一节中将详细讨论。

注意 优化程序仅在每条语句或语句块接受一个提示。优化程序忽略任何附加的提示。此外，优化程序忽略任何拼写错误的提示。没有被报告的错误，你必须使用 EXPLAIN PLAN来决定是否采纳了你的提示，EXPLAIN PLAN将在17.4节中讨论。

17.2.1 分级访问路径

Oracle优化程序访问路径按照最快到最慢，顶端到底端的顺序排列，如表 17-1所示。

表17-1 Oracle优化程序访问路径，有序等级（来源：Oracle公司）

等 级	访问路径
1	依据ROWID的单行
2	依据簇连结的单行
3	依据具有唯一键或主键的哈西簇键的单行
4	依据唯一键或主键的单行
5	簇连接
6	哈西簇键
7	索引的簇键
8	组合键
9	单列索引
10	索引列上的限定范围查找
11	索引列上的没有限定范围查找
12	分类合并连结
13	索引列的MAX或MIN
14	索引列上的ORDER BY
15	全表扫描

当Oracle优化程序采用基于规则的策略时，它使用适合给定查询的最快的访问路径。例如，清单17-2显示了和程序清单 17-1一样的查询，不过没有提示它已经用一个 >WHERE子句修改了。

清单17-2 一个没有提示的查询与一个没有范围限制的查找的例子

```
SQL> SELECT EMPLOYEE_ID  
2> FROM EMPLOYEES  
3> WHERE EMPLOYEE_ID > 500000;
```

假如Oracle正在使用基于规则的策略并且一个主键索引存在于EMPLOYEES表的EMPLOYEE_ID上，那么Oracle将对清单17-2的查询使用访问路径#11（索引列上的没有范围限制的查找），因为WHERE子句是不受限制的。换句话说，当一条 WHERE子句有大于（>或>=）运算符并且没有相应的小于（<或<=）运算符时，该子句就是不受限制的。反之亦然。直到执行时才知道范围。

假如WHERE子句是受限制的（实际上，它有两个运算符），限定的范围在语法分析时才知道。要使一个不受限制的 WHERE子句变为受限制的，你可以使用直接量最大值（假如你知道它），或者根据你的业务规则至少使用该列的理论最大值。然而，除了你的初始的不受范围限制的查找外，你或许不想使用 MAX运算符——假如你查看等级排序，正在使用的 MAX是

#13, 它把一个不受限制的查找（#11）提高到一个受限制的查找（#10）。把MAX运算符用于初始的不受范围限制的查找会降低你的速度。但是使用实际的最大值将获得成功。理解这些在清单17-3中说明的原则：

- 提前使用一个已知的实际最大值（直接量）。这在清单17-3的第一条查询中进行了举例说明（678453）。
- 假如不知道实际值的话，提前使用一个已知的理论上的业务最大值（直接量）。这在清单17-3的第二条查询中进行了举例说明（999999）。
- 不要使用最大SQL集合函数，因为它仅会使事情变慢。这在清单17-3的第三条查询中进行了举例说明。

清单17-3 重写一个不受限制的查找以使它受限制

```
SQL> SELECT EMPLOYEE_ID
2> FROM EMPLOYEES
3> WHERE EMPLOYEE_ID > 500000 AND EMPLOYEE_ID <=678453;

SQL> SELECT EMPLOYEE_ID
2> FROM EMPLOYEES
3> WHERE EMPLOYEE_ID > 500000 AND EMPLOYEE_ID <=999999;

SQL> SELECT EMPLOYEE_ID
2> FROM EMPLOYEES
3> WHERE EMPLOYEE_ID > 500000 AND EMPLOYEE_ID <= MAX(EMPLOYEE_ID);
```

17.2.2 分析查询以提高效率

试图再配置或重写查询以提高它们效率的领域被称为查询重写。查询重写将在本章中的17.5节与17.6节中详细讨论。前面的例子很标准并且容易理解。例如，要得到一列中一组数值的最大值（MAX），似乎要花费接近于一个整表扫描所用的时间——不管该列是否被索引。正如你看到的那样，通过最大值访问路径的等级排序，情况大致就是这样。然而，还有不可理解的和更特殊的重写查询方法。

前面已研究了基于规则的优化，现在来看一下基于成本的优先。先来看一下清单17-4和清单17-5中的两个查询。

清单17-4 同一张表上的两个相似查询的第一个

```
SQL> SELECT EMPLOYEE_ID
2> FROM EMPLOYEES
3> WHERE EMPLOYEE_TYPE='VICE PRESIDENT';
```

清单17-5 同一张表上的两个相似查询的第二个

```
SQL> SELECT EMPLOYEE_ID
2> FROM EMPLOYEES
3> WHERE EMPLOYEE_TYPE='PROGRAMMER';
```

现在，假定你拥有一个相对较大的软件公司，你有6个副手和总共大约6000个雇员，这其中有大约2000个程序员。前面的两个查询解释了基于规则和基于成本的查询优化之间的差别以及为什么首选基于成本的优化。假设EMPLOYEE_TYPE列上有一个非唯一索引，那么基于

规则的优化为两个查询选择访问路径 #9（单列索引）。另一方面，假如数据分布是程序员占总行数的 1/3 并且副手仅占总行数的 1/1000——那么为清单 17-4 中的查询选择使用非唯一索引，但是为清单 17-5 中的查询选择整表搜索（最坏的事件访问路径 #15）是明智的。这一决定来自对数据分布方面所具有的知识。

假如优化程序必须对一张表中所有行的有效部分进行访问，那么全表扫描实际上比索引搜索更有效。这是因为搜索行索引，然后再检索那个特殊行的过程至少每行需要两个读操作，有时更多——根据索引中独特数值的数量而不同。然而，全表扫描每行仅需要一个读操作。在有許多行时这个次数增加，和清单 17-5 中的查询一样，同与整个表的刚好读出相比，对一张表进行大量访问时索引更慢的原因相当明显。除了读操作的总数以外，全表扫描优于对表的大部分进行检索的索引的另一个主要原因是大部分必要的读操作是连续的或几乎连续的。显而易见，从一个索引中读取，之后从一张表中读取，然后回到该索引，之后回到该表，这样是不可能连续的，不论表和索引是否在同一张磁盘上或者在好的调整方式下展开。索引由于有类似清单 17-5 的查询和数据而使速度慢了下来。

在类似清单 17-4 的情况下，索引明显优于为查询和数据的全表扫描。这是因为你仅需要检索少量的行（6）并且你的读操作的总数仅是 12（ 2×6 ）。相比之下，一个全表扫描必须完成 6000 个读操作（整个表），因为一个全表扫描不知道单个数据片存储在哪儿，所以它必须检查每一行。实际上，程序清单 17-4 中的查询和数据是一个需要索引的典型例子，特别是在查询经常被执行的情况下。

17.2.3 指定优化程序模式

下一个明显的问题是如何指定 Oracle 中的优化程序模式。迄今为止，你已经学习了有关基于规则和基于成本的优化。可以在实例、会话或语句层指定你希望的优化形式。如果要在实例层指定优化形式，把 init.ora 的参数 OPTIMIZER_MODE 设置为下列值中的一个：

- **CHOOSE** —— 当设置为该值时，如果统计数字可用的话（已经由数据库管理员运行），优化程序选择基于成本的模式。否则，用基于规则的优化。
- **RULE** —— 优化程序使用基于规则的方法。
- **FIRST_ROWS** —— 优化程序选择基于成本的方法（假设统计数字可用的话）来把反应时间减到最小。也就是说，使第一行尽快出现在屏幕上。假如你有一个高度交互式的、基于屏幕的应用，例如许多 OLTP 和较小的 DSS 系统，那么使用这种模式。
- **ALL_ROWS** —— 优化程序选择基于成本的模式（假设统计数字可用的话）来最小化吞吐量，也就是说，最小化每个时间单元内通过系统的总行数（每秒钟的事务）。假如你有一个批处理或大型 DSS 系统的话，那么使用这种模式。

要在会话层指定优化形式，发出如下的 DLL 语句：

```
SQL> ALTER SESSION SET OPTIMIZER_GOAL=<value>;
```

该值是以前提及的优化程序模式的一种（CHOOSE、RULE、FIRST_ROWS、ALL_ROWS）。结果仅适用于本会话，因此，必须在一个未来的会话中重新发出这个命令。

要在语句层指定优化形式，像较早前讨论过的那样使用提示。提示可以采用任何一种优化程序模式值（CHOOSE、RULE、FIRST_ROWS、ALL_ROWS），或者是显示于表 17-2 和表 17-3 中的任何一个访问路径。

表17-2 提示的基本访问路径

访问路径	描 述
ROWID	使用用于检索的ROWID搜索
CLUSTER	使用簇键搜索
HASH	使用哈希索引搜索
INDEX	使用索引搜索
INDEX_ASC	使用索引搜索并按升序搜索
INDEX_DESC	使用索引搜索并按降序搜索
AND_EQUAL	使用多个索引并且合并它们的结果
ORDERED	使FROM子句中的表命令成为并运算命令
USE_NL	使用用于表连结的嵌套循环方法
USE_MERGE	使用用于表连结的分类合并方法
FULL	使用全表扫描

表17-3 提示的辅助访问路径（版本7.3和更新的版本）

访问路径	描 述
CACHE	通知Oracle把表作为一个缓冲表对待，在一个完全搜索之后将表的数据块保存在SGA中用于后面的快速访问
HASH_AJ	在一个非连结期间指定要使用的连结类型（Oracle 7.3和更新的版本）
MERGE_AJ	在一个非连结期间指定要使用的连结类型
NO_MERGE	通知Oracle不要将视图的SQL语法与使用连结的查询语法合并
NO_CACHE	把块标记为“最近最少使用的”以便它们可以不久后从SGA中移去
NONPARALLEL	禁用查询的并行操作
ROWID	使用TABLE ACCESS BY ROWID操作
STAR	当解决一个连结时使用一个组合键/开始查询执行路径
USE_CONTACT	迫使WHERE子句中的OR条件组合为UNION ALL
USE_HASH	使用哈希连结

还有，要使用任何一个基于成本的优化模式（FIRST_ROWS、ALL_ROWS），数据库管理员必须周期性地运行统计数字以及时反映实际的存储的数据分布。数据库管理员既可以获得一个（简单随机的）样本的所有统计数字也可以获得该样本的某些统计数字。要获得所有的统计数字意味着做一次全表扫描。要获得一个采样意味着要访问少于总行数的某些部分。数据库管理员发出如下的DLL语句以获得所有的统计数字：

```
SQL> ANALYZE TABLE <table_name> COMPUTE STATISTICS;
```

该语句对于相对小的表——少于一百万行来说比较精确。对于大表，数据库管理员或许希望选取一个样本。从统计学上讲，只要样品相对于总行数足够大，那么它就足够精确，不需要获得所有的统计数字。要获得一个样品，数据库管理员应发出如下的DDL语句：

```
SQL> ANALYZE TABLE <table_name> ESTIMATE STATISTICS;
```

不管实际表有多大，这条语句最大可以采样1064行（缺省情况下）。数据库管理员通过发出下面这条语句指定百分率：

```
SQL> ANALYZE TABLE <table_name> ESTIMATE STATISTICS SAMPLE 10 PERCENT;
```

该语句采样了表中总行数的10%，四舍五入到一个整数。例如，该语句从6000行的EMPLOYEES表中采样600行用于估算。数据库管理员也可以通过发出如下的语句指定实际的采样容量：

```
SQL> ANALYZE TABLE <table_name> ESTIMATE STATISTICS SAMPLE 2000 ROWS;
```


很明显，该语句精确地采样了指定表的 2000 行。

警告 假如你指定了一个超过 50% 的百分数（或者超过行数的 50% 的数值），那么 ANALYZE 命令便采用全表扫描计算而不是你所指定的百分数（或行数）。

对于 Oracle 的 7.3 版本和更高的版本，ANALYZE 命令有一条附加的 FOR 子句。利用这条子句，你可以命名具体的 FOR COLUMN 列。例如，假如自上一次 ANALYZE 运行以来只有少数列发生了变化或者你想要减少 ANALYZE 的运行时间，那么指定 FOR INDEXED COLUMNS ONLY，它可以不获得非索引列上的统计数字精确地执行要做的内容。

数据库管理员应定期地运行 ANALYZE。然而，对一个应用敏感的东西对另一个应用或许并不敏感。例如，每月从三个批处理系统进行加载的 DSS 仅需要分析（用计算机）那些已经装入的表，然后立即或在它们下一次使用前的某个合理时间跟踪装入值。相比之下，一个高活性的 OLTP 系统——例如一个处理飞行保留信息的系统——几乎需要所有的表，这些表可能每分钟都进行修改，在一个很短的时间间隔，例如一个小时或更短的时间间隔中被分析（用 ESTIMATE）。这些类型的系统或许被映像并且当某个产品拷贝被脱机分析时，其他的产品拷贝可以继续服务，反之亦然。一般而言，当你的表变化很大时，尽可能频繁地或按一定的间隔分析你的表以便 Oracle 优化程序能够为你的应用需求提供最好的服务。如果必要的话，可以交互式地执行分析，但是这些 ANALYZE 操作经常由 DML 事件预定或触发并由非交互性的脚本或存储过程来编写。

17.2.4 理解优化术语

现在，查看一些数据库管理员可以理解并经常使用的术语。直方图（histogram）无非就是一种特殊类型的条形图。当执行 ANALYZE...COMPUTE 或 ANALYZE...ESTIMATE 时，存储该表的列数据分布。一个数据分布就是映射不同值的频率（数量或小计）或百分率的函数。在 Oracle 中，这些数据分布作为数字存储，但是假如你编写一个程序来显示这些数据分布的话，它们可以作为直方图显示。

选择性（selectivity）有时称为索引的选择性因子，是在一列按总值划分的数据值中总不同值的数量。选择性因子越高越好。除此之外，选择性越高，在那列上创建与使用索引的理由就越充分。通常，不同值越少，意味着选择性越小（这可以讲得通，因为该值是选择性公式的分子）。

通常一个存储性别类型（例如男或女）的列或者只有几种可能值的分类类型（例如婚姻状况，包括独身、已婚、离婚或寡居）的列不具有选择性并且不为索引的使用提供好的基础。一个例外是在这些类型的列中一个或多个不同值几乎不出现或经常被查询的情况下（在 WHERE 子句中频繁使用）。假设婚姻状况用作市长市民表的一列，那么该列数值中独居所占的比例相当小，小于 10%，而独居值经常被查询。因此，索引很适合于这类查询，尽管索引在其他值的查询上不会有太大帮助。此外，在这里位映射索引运行得会很好。决定何时使用索引以及使用何种类型的索引将在本章稍后的 17.5 节中讨论。

与必须从表中取数据相反，优化程序可以从一条可用的索引中读取和返回数据。优化程序优先级被称为索引覆盖（index coverage），尽管偶尔也会使性能变差，但通常会导致好的性能。假定在一张 EMPLOYEE 表的 LAST_NAME、FIRST_NAME 与 MIDDLE_INITIAL 中有一个复合索引，你可以运行如下的查询以找到所有 LAST_NAME 是 SMITH 的雇员：

```
SQL> SELECT LAST_NAME  
2> FROM EMPLOYEES  
3> WHERE LAST_NAME = 'SMITH';
```

在该例中，这个查询被认为是被索引覆盖的。换句话说，SELECT语句的SELECT子句中需要的所有列被存储（不仅在该表中而且）在该索引中。由于显而易见的原因，优化程序通常更喜欢单独使用索引，而不是同时使用索引和表。当一口井便能满足你的所有需要时，为什么要从两口井中取水？然而，索引覆盖有副作用，这将在17.5节中讨论。

最后，数据库管理员应当知道的关于查询调整的最重要的规则是百分之五规则。这条规则应用于Oracle 7.x中，但是在较老的版本中没有使用。假如从ANALYZE操作中得到可用的统计数字，那么当优化程序会发现期望要返回的行数大于5%时，它采用全表扫描。在Oracle 6.x和5.x中，该百分率分别是15%和20%。尽管实际的百分率不再应用于现代Oracle版本中，但该规则经常被称为20%规则。这是数据库管理员必须知道的重要信息，因为优化程序有时会做出错误的（不是最佳的）选择，根据你的应用的知识，你可能需要忽略它（例如，使用一个提示或重写查询）。

17.3 SQL TRACE与tkprof

SQL TRACE工具，连同tkprof格式化程序一起，提供了为单条SQL语句生成和检查性能统计数字的功能。你可以在实例或会话层启用SQL TRACE。然而，你必须在实例或会话层中都启用某些init.ora参数才能使用SQL TRACE。假如还没有设置这些参数，那么请设置如下参数：

```
TIMED_STATISTICS=TRUE  
USER_DUMP_DEST=<directory>  
MAX_DUMP_FILE_SIZE=<size in operating system blocks>
```

很明显，假如你希望SQL TRACE能够收集定时信息的话，那么你应当把TIMED_STATISTICS设置为TRUE。如果你不想把跟踪文件存储在操作系统特定的缺省位置的话，那么请指定USER_DUMP_DEST。此外，MAX_DUMP_FILE_SIZE应被设置得足够大以容纳所有生成的跟踪文件。要在实例层启用SQL TRACE（也就是说，对于所有的会话），设置如下init.ora参数：

```
SQL_TRACE=TRUE
```

所有的会话将产生跟踪文件。除了实例细节外，命名具有会话细节的跟踪文件能够辨别跟踪文件属于哪一个会话并保证在USER_DUMP_DEST中有足够的空间以容纳所有的文件。单独运行跟踪文件，可能的话顺序运行它，所以不会有混乱发生。要在会话层启用SQL TRACE，发出如下的DDL语句：

```
SQL> ALTER SESSION SET SQL_TRACE=TRUE;
```

这个SQL TRACE仅产生会话期的统计数字。现在，要能够读取由SQL TRACE创建的信息，你必须使用tkprof。在操作系统层（命令行）运行tkprof程序。执行这项工作的简化语法如下：

```
tkprof <tracefile> <outputfile>
```

一个UNIX系统中的例子如下所示：

```
hostname% tkprof ora_1776.trc ora_1776.out
```

在该例中，tkprof把跟踪文件（ora_1776.trc）重新格式化到一个可读的、用户定义的输出文件（ora_1776.out）中。一个类似于下述SQL语句的tkprof输出示例显示在清单17-6中：

```
SELECT * FROM EMPLOYEES WHERE DUTY >= 10;
```

清单17-6 一个已格式化的tkprof输出例子

Call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.17	0.35	0	0	0	0
Execute	1	0.00	0.01	0	0	0	0
Fetch	1	0.06	0.10	13	304	0	3679
Misses in library cache during parse: 1							
Misses in library cache during execute: 1							
Optimizer mode: CHOOSE							
Parsing user id: 8							

上述这些信息意味着什么以及它如何帮助你调整你的应用？正如大部分诊断工具一样，信息被给出，但是没有解释，至少没有自动解释。首先，检查你的输出文件中的柱状统计数字以及它们的含义，如表17-4中所讨论的。

表17-4 tkprof统计数字的描述

统 计 量	描 述
count	语法分析、执行或取数据调用的数量
cpu	以秒计的实际CPU时间（假如语法分析的时间为0，则该语句处于库高速缓存中）
elapsed	以秒计的实际时钟（或背景）时间（总是大于或等于 CPU时间）
disk	从数据文件中读取的Oracle数据块；也称为物理读取（physical reads）
query	Oracle缓存读取，用于从回滚缓冲区高速缓存中一致性读取，存储于数据库缓冲区高速缓存中；也称为一致性获取（consistent gets）
current	Oracle缓冲区在当前模式下从数据库缓冲区高速缓存中读取；也称为数据库块获取（db block gets）
rows	由主驱动（或外部）语句访问的行数；对于 SELECT语句针对取数据，否则针对执行（DML）

注意到逻辑读取的数量等于 query+current。你如何解释这一点？一名应用开发者、程序员或数据库管理员的目标是要做如下事情：

1) 尽可能地减少逻辑读取的数量。确保你正好且仅仅访问你需要访问的东西。换句话说，不要访问表格中的不必访问表或行。尽可能地采用简化操作。例如，假设一张 EMPLOYEES表在DEPT_ID上包含一个指向DEPARTMENTS表和一个称为DEPT_NAMES的查询表的外键。当使用如下的语句检索一个 EMPLOYEES行时，在给定DEPT_ID的情况下，现在假设一条通用的查询来检索DEPT_NAME：

```
SQL> SELECT E.LAST_NAME, E.FIRST_NAME, N.DEPT_NAME
2> FROM EMPLOYEES E, DEPARTMENTS D, DEPT_NAMES N
3> WHERE E.DEPT_ID = D.DEPT_ID
4> AND D.DEPT_ID = N.DEPT_ID;
```

上述语句产生了预想的结果，但是没有必要对 DEPARTMENTS表进行访问。假如正确的完整性和有关的约束条件被定义和维护的话，这就是真实的。特别是，DEPT_NAMES中的DEPT_ID对于DEPARTMENTS表来说也是一个外键。因此，如果DEPT_ID不首先存在于DEPARTMENTS表中的话，你决不会有存在于DEPT_NAMES表中的DEPT_ID。为了减少不必要的访问，该查询可以简化为如下所示：

```
SQL> SELECT E.LAST_NAME, E.FIRST_NAME, N.DEPT_NAME
2> FROM EMPLOYEES E, DEPT_NAMES N
3> WHERE E.DEPT_ID = N.DEPT_ID;
```

2) 尽可能地把物理读/逻辑读的比率减小到接近零。首先, 确保实现第一个目标——减少总逻辑读取。其次, 确保用最有效方式访问你需要访问的东西。使用迄今为止你已在本章中学到的知识。其余小节将为你提供更多的例子和方案。第三, 假如你已经完成了前两个工作, 那么在你的SGA中增加数据库缓冲区, 如果需要的话购买更多的内存。这将在第18章中进一步讨论。一般而言, 对于大部分的语句, 你的比率应该小于10%。在SELECT语句的tkprof输出示例中, 比值是13/304, 或大约0.04。这是一个好的比值。事实上, 这相当于96% (100-4) 的数据库高速缓存的命中率。

3) 尽可能地把逻辑读/行减小到接近零。对于每一次逻辑读取, 你希望能够读取尽可能多的行。因此, 比率越小, 你的读取越有效。这是块效率 (block efficiency) 的一个方面, 将在第19章中讨论。很明显, 假如比值太高了, 例如 1:8 (0.125) 或更大, 或许应增加db_block_size的值, 因为它相对于大的行容量来说似乎是太低了。在以前的tkprof示例中, 该比值较好 (13/879), 大约是.015, 或小于1:67。这意味着对于每15个数据块读取, 大约有1000行被检索, 或者说每个数据块大约67行 (假如你正在使用一个16K的Oracle db_block_size的话, 每行大约246个字节)。

除了刚刚提到的简化示例外, tkprof还提供了几个其他的命令行选项。最有用的在表17-5中讨论。解释执行计划将在随后的17.4节中更为详细地讨论。

表17-5 一些tkprof命令行选项

选 项	描 述
explain=user/password	使你作为一个用户 (user) 运行EXPLAIN PLAN; 该选项极为有用, 因为它为跟踪文件中的每一条语句提供了执行计划
record=file	使你能够指定一个文件 (file), 跟踪文件的语句保存在该文件中; 它还用来保存用于单个语句测试的脚本
sys=NO	抑制通过系统用户回忆的循环调用; 这很有用, 因为即使你可以通过缺省值查看这些调用, 它们也不能真正地被修改以有助于性能的提高——不考虑它们, 简化你必须过滤的输出量

要在屏幕上显示所有的选项, 在命令行键入 tkprof help=YES。

警告 explain tkprof命令行选项在tkprof运行时刻提供了计划且并不基于SQL TRACE操作, 因此这两组输出或许不一致。

以下对使用SQL TRACE与tkprof进行了总结:

- 1) 当你有多个并发应用并且希望查明你的最大的资源密集型的应用时, 主要使用这些工具。把那些对于你的优先调整尝试不太有效的应用作为目标。
- 2) 在init.ora中或用ALTER SESSION打开SQL TRACE。
- 3) 运行你的应用。
- 4) 关闭SQL TRACE (假如你不打算不久后重新使用它)。
- 5) 运行tkprof, 把你的跟踪文件格式化到可读的输出文件中。
- 6) 对结果进行解释。
- 7) 按照提示调整你的应用或数据库。这经常意味着要重写查询。
- 8) 返回到步骤2, 然后重复, 直到你对步骤6的结果满意为止。

17.4 理解EXPLAIN PLAN

如果许多语句按一定的应用顺序运行，尽管 SQL TRACE 工具和tkprof程序可以提供有用的统计数字和执行计划信息，然而 EXPLAIN PLAN不必运行语句便可以一次为一条语句给出执行计划信息。这在分析单个问题查询中非常有用，尤其是如果该查询是运行时间很长的查询。当你可以通过使用 EXPLAIN PLAN立即得到它时，为什么要运行一条查询并通过 SQL TRACE或tkprof等待执行计划输出？因此，Oracle也提供这个非常有用的工具。

你如何使用 EXPLAIN PLAN？首先，运行脚本以创建存储 EXPLAIN PLAN输出的 PLAN_TABLE。尽管目录和文件名或许随着操作系统平台的不同而变化，但这在 UNIX操作系统中通常是\$ORACLE_HOME/rdbms<version>/admin/utxlplan.sql。在NT上的位置是类似的。在你运行该脚本后，应当在你的模式下创建了 PLAN_TABLE（假设你有正确的特权）。

其次，在你想要生成执行计划信息的语句上运行 EXPLAIN PLAN，例如用如下的查询：

```
SQL> EXPLAIN PLAN
2> SET STATEMENT_ID='STMT_1'
3> FOR
4> SELECT EMPLOYEE_ID
5> FROM EMPLOYEES
6> WHERE EMPLOYEE_ID=243218;
```

使用如下的 SELECT 语句从 PLAN_TABLE 中得到你想要的输出：

```
SQL> SELECT OPERATION, OPTIONS, OBJECT_NAME, ID, PARENT_ID, POSITION
2> FROM PLAN_TABLE
3> WHERE STATEMENT_ID='STMT_1'
4> ORDER BY ID;
```

输出结果如下所示：

Operation	Options	OBJECT_NAME	ID	PARENT_ID	Position
SELECT STATEMENT			0	2	
TABLE ACCESS	BY ROWID	EMPLOYEES	1	0	1
INDEX	RANGE SCAN	EMPL_IDX	2	1	1

来源：Oracle公司

假如你喜欢的话，你也可以生成一个树型输出。可以使用如下的 SQL 代码得到一个缩进的文本图表示：

```
SQL> SELECT LPAD(' ', 2*LEVEL-1) || OPERATION || ' ' || OPTIONS || ' ' ||
OBJECT_NAME || ' ' || DECODE(ID, 0, 'COST= ' || POSITION) "QUERY PLAN"
2> FROM PLAN_TABLE
3> START WITH ID=0 AND STATEMENT_ID='STMT_1'
4> CONNECT BY PRIOR ID=PARENT_ID;
```

输出结果如下所示：

Query Plan

```
-----
SELECT STATEMENT          COST=2
  TABLE ACCESS BY ROWID EMPLOYEES
    INDEX RANGE SCAN EMPL_IDX
```

这对于产生大的执行计划的大型查询来说常常很有用。就像父子关系一样，缩进表示了执行树中的另一个下属层次。前述的示例包含 3 个层次，成本等于 2。最重要的事情是证实，即你期望优化程序使用索引并且它使用了。假如它没有使用，你需要检查它没有使用的各种

原因,例如返回的行的数量、索引不具有高度选择性等等。努力去改变这种状况;然后返回到EXPLAIN PLAN并且查看是否你已经解决了该问题。在决定是否应继续使用优化程序时,EXPLAIN PLAN的重复使用是EXPLAIN PLAN最值得注意的用法。

下面对使用EXPLAIN PLAN进行了概述:

1) 当在你的应用中仅有一个或几个查询时,当已经把主要的性能问题孤立到一个或几个查询时,当你的查询通常顺序运行时或者当你不能提供重复执行时间用于等待每条 SQL TRACE/tkprof的运行时,使用EXPLAIN PLAN代替SQL TRACE或tkprof。

2) 运行\$ORACLE_HOME/rdbms<version>/admin/utxlplan.sql。

3) 把EXPLAIN PLAN用于你的当前查询。

4) 从结果PLAN_TABLE中选择文本或树输出。

5) 解释结果。验证你对优化程序性能的期望值。

6) 按照提示调整查询。这经常意味着重写查询。

7) 返回到步骤3并且重复,直到你对步骤5的结果满意为止。

17.5 鉴别典型问题

大部分应用调整问题的根源归结于不正确地使用索引。在 17.5.1 节中,你将浏览一下正确使用索引的一些规则。在 17.5.2 节中,你将考虑一些典型的应用问题,这些问题有的直接或间接地与索引有关,有的与索引无关。

17.5.1 正确使用索引

一般,大部分应用调整问题发生在决定是否使用索引的时候。随着时间的过去,索引既可能使用不足又可能使用过度。这是不同专业知识的数据库管理员之间的两个极端情况。你对索引所做的最好的事情就是必须正确地使用它。下面是正确使用索引的提示列表:

- 频繁用于WHERE子句中的索引列。
- 频繁用于连结表的索引列。
- 具有高度选择性的索引列(接近1)。
- 为具有低选择性(接近0)的列使用位映射索引。

除了上述的提示外,通过使用索引,你可以做一些别的事情以使你的数据库更有效。

例如,对具有高度选择性(接近1)并且几乎总是通过点(point)查询来访问的列使用哈希索引,点查询也称为精确匹配(exact match)查询。有具有相等比较关系的查询(例如WHERE x=y或WHERE s='abc')为几乎所有其他列使用普通索引(B*树),这些列趋向于采用实际的受限制的或不受限制的范围(搜索)查询率来访问。还有具有非相等比较关系的查询,例如WHERE x>y或WHERE s LIKE c%。

不要对那些必须经常修改的列做索引。还有,设法抑制对具有高插入与删除操作的表中的任何列做索引。这两种情况中的任何一个都会引起B*树结构的连续重组并最终会由于索引碎片而导致性能变差。主键和外键例外。

对主键和外键做索引。不管你是否创建了一个索引,Oracle在已定义的关键上创建一个唯一索引。然而,缺省情况下Oracle不在外键上创建一个索引,除非你明确地用一条CREATE或ALTER命令执行这项操作。之所以这样做是因为当子表通过DML访问时不锁定父表可以增强系统性能。

在适当的时候，增加辅助列以形成复合索引。这利用了优化程序的索引覆盖功能，索引覆盖在较早前讨论过。Oracle不像Sybase，它使用一个更受限制的索引覆盖形式。Oracle强加约束以便列只有在它们形成那个（复合）索引的主要列的时候才能够从索引中检索。假设一张表在a+b+c列上有一个复合索引。优化程序通过使用a、a+b或a+b+c而不是b、c、b+c或a+c把该索引用于一个WHERE子句。对于Oracle来说，索引覆盖仅能用于复合索引的左侧排序子集（前缀）。

不要对那些总是用于具有函数或操作符（MIN或MAX操作符除外）的WHERE子句中的列做索引。在这种情况下，优化程序不使用索引。换句话说，创建一个导出列或重写该查询，使用提示或技巧绕过这个操作。使用EXPLAIN PLAN验证你的结果。

不要对那些主要用于否定或NULL比较的列做索引。Oracle优化程序不使用WHERE子句中基于如下比较操作的索引：

■ IS NULL

■ IS NOT NULL

■ !=

做为选择，积极地使用=或IN操作符重写查询。如果使用NULL比较，你或许需要考虑使用缺省值。

小心使用视图和复杂子查询的索引。如果给定各种层次的视图和复杂子查询语句，很难预计将要发生的事情。你必须反复使用EXPLAIN PLAN以确定这些查询正在做什么。假如你不能使你的查询按照要求运转，考虑通过把子查询压缩到一个较大的查询中或把它们分裂为多个连续的子查询的方式来重写它们。还有，考虑放弃视图或者实现它们。

最后，不要忘记适当地做索引（参见以前的步骤），不要为了做索引而做索引。我曾经见过表中不含索引的数据库，并且我也见过表中在每件东西上都有索引的数据库。没有索引一般不是好事情，因为你总应该有一个主键索引。在每件东西上都有索引一般也没益处，因为这经常是不良的需求分析、不良的数据库设计或不良的应用设计的症状。

既然已经讨论了主要的索引准则，你可以查看一些在应用调整中发现的更为典型的问题。

17.5.2 应用调整的典型问题

几乎在所有的应用调整中遇到的典型问题都涉及到某些类型的低效率的SQL编码或不适当地使用索引。本节对这些问题中的其中几个进行讨论。

有一个称为索引覆盖异常（index coverage anomaly）的问题。一个关于索引覆盖的有趣的事情可能会发生。当它发生时，那么通过此查询返回的行数比较大，因而全表扫描可能会更加有效。然而，假如ANALYZE没有运行，优化程序借助于基于规则的优化并选择索引访问路径。如果数据库管理员没有保持最新的统计数字，那么优化程序可能选择不良的访问路径，尽管它本身并没有缺陷。

可能发生的另一个问题被称为不必要的多次通过或多回路。请看如下查询：

```
SQL> SELECT A FROM T WHERE A > 9;
```

```
SQL> SELECT B/A FROM T WHERE B < 3 AND A > 9;
```

很明显，这可以通过表T在一个表中完成：

```
SQL> SELECT A , B/A FROM T WHERE B < 3 AND A > 9;
```

使用不必要的函数或NULL比较是另一个可能发生的问题。我们已经讨论过如何正确使用

索引，优化程序不会在这些情形下使用索引，所以不会做如下的事情：

```
SQL> SELECT A FROM T WHERE X < MAX(X);
```

正如较早前讨论过的一样，你希望使用直接量并且以如下形式限制范围搜索：

```
SQL> SELECT A FROM T WHERE X >= 0 AND X < 999999;
```

还有，如果可能的话，把NULL列转化为一个缺省条件：

使用

```
SQL> SELECT A FROM T WHERE X > 0;
```

而不是

```
SQL> SELECT A FROM T WHERE X != 0;
```

或者

```
SQL> SELECT A FROM T WHERE X IS NOT NULL;
```

如果是有意的话，后两条语句都没有在X列上使用索引。

不要忘记对外键做索引。当子表正被插入、修改或删除时，这样做会引起子表锁定父表，反之亦然。假如你对所有的外键做索引，对子表的修改不会锁定父表并且对父表的修改仅得到一个行级锁定。

当表能够以某种方式推导出时，不要从该表中选取信息。涉及到数据库的一个主要应用编程规则如下：

注意 在数据库应用中对存储进行计算是一件好事情。

这是事实，因为大部分数据库应用趋向于受输入/输出的限制。例如，假设你正在计算基于单一固定税率（或者少数平齐的税率）的销售税。税表通常是静态查询表，利率在一个给定的税期（例如，一年）中通常是不变的。下面的示例展示了如何使用一个函数，而不必访问一个已存储的推导列。

不要使用下述语句：

```
SQL> SELECT (P.COST + (P.COST * T.RATE)) "COST WITH TAX"  
2> FROM PRICES P, TAXES T  
3> WHERE P.PART_ID = T.PART_ID
```

使用如下语句：

```
SQL> SELECT (COST * 1.06) "COST WITH TAX"  
2> FROM PRICES;
```

这说明原始设计中有一些问题，你或许不需要 TAXES表。然而，把该问题搁置一边，你当然不必访问税率列（RATE）才够用那个乘数。这是事实，因为你正在依赖一个常量（简单的、硬编码的直接量替换值）。如果开发者在这一业务领域有充足的知识并且如果知识量相对较小且所有开发者都很熟悉的话，那么这可以很好地运行。总之，该例子表明了一个简单的函数如何替代一个表访问。请注意，硬编码技术被认为是不良的程序设计，但是你的目的是取得好的应用性能。这两者经常有矛盾，最后这个例子也不例外。你必须决定你的所有目标中哪一个是最重要的目标。

不要触发不必要的全表扫描，引起累积百分数问题。当你不需要时为什么要对一张表进行扫描？通常，你必须寻找会引发一个全表扫描的查询，换句话说，查询不使用一个会有意或可能以另一种方式引起一个全表扫描的索引。还有，不要忘记 5%规则。假如优化程序希望从查询中返回的值超过 5%（在版本 7.x 或 7.x 以上），就要求助于全表扫描。使用如下语句，在

合并多个WHERE条件总计该数量时可能发生全表扫描：

```
SQL> SELECT A FROM T WHERE X = 1 AND Y = 2 AND Z = 3;
```

如果统计数字显示x=1有3%，Y=2有2%，Z=3有2%，优化程序会发现累积百分数达到了7%并选择全表扫描。你也可以把这两个查询拆开，尤其是对于大表而言，这样就可以在同样的结果下取得较好的性能：

```
SQL> SELECT A FROM T WHERE X = 1;  
SQL> SELECT A FROM T WHERE Y = 2 AND Z = 3;
```

优化程序在X、Y、Z上使用索引，因为第一条查询仅显示3%，第二条查询仅为4%。

小心地使用DISTINCT、ORDER BY与UNION。这些操作会创建临时表并增加排序的附加系统开销。作为一个数据库管理员，除非你有无限的资源，否则你几乎不能容忍这些事情同时发生。许多这些同时发生的操作与同时发出许多CREATE INDEX语句有相同的性能消耗。假如你必须使用DISTINCT的话，那么就频繁地使用它并且存储输出值用于将来的使用。假如你必须使用ORDER BY，设法在那一列上设置一个索引。假如你必须使用UNION，那么就使用UNION ALL（不消除副本）。只要你已定义并启用所有的主键，就可以在大多数情况下使用后一种解决方案。

注意 在任何应用调整中，因为应用前端或许用其他SQL语句编写，所以研究已翻译的SQL语句。由于各种原因，ODBC驱动程序翻译不总是生成优化的查询。注意这些情况并打开ODBC跟踪文件以便你可以看到已翻译的SQL语句，它们或许正阻止索引的使用。

17.6 重写查询

在前几节中已重写了大量查询。现在，把注意力集中在重写查询的几个示例中，在这些示例中你会看到如何能够真正地调整SQL，使之具有高效率。除了已经讨论过的许多技术外，最好的两个技术如下：

- 使用集合操作符。
- 使用布尔转换。

重写查询的领域高度数字化，然而突出了实用。当你听到重写查询时，就意味着重新编写它们以加快速度，而不是调整它们。后者遵循第一次正确得到它的法则，然后对程序设计加以维护。你所感兴趣的是原始速度，经常可以牺牲可读性和可维护性。因此，在投入这一领域的冒险以前就必须做出决定。

17.6.1 使用集合操作符

你已经看到UNION可以降低性能。你应更感兴趣的是使用MINUS或INTERSECT。它们实际上有助于性能的提高。观察如下的查询：

```
SQL> SELECT ACCOUNT_ID  
2> FROM JOBS_COMPLETED  
3> WHERE ACCOUNTS_ID NOT IN (ACCOUNTS_PAID);
```

它可以重写为如下形式：

```
SQL> SELECT ACCOUNT_ID FROM JOBS_COMPLETED  
2> MINUS  
3> SELECT ACCOUNT_ID FROM ACCOUNTS_PAID;
```

这两个查询都给出了相同的操作结果：它们返回那些已经完成了他们的工作但还没有得

到付款的客户。然而，使用 EXPLAIN PLAN 可能会发现第一次的总逻辑读取比第二次的更大。在 Oracle 中，使用 MINUS 运算符是非常有效的。

除了 MINUS 是一个非对称（单向）运算符而 INTERSECT 是对称（双向）运算符外，INTERSECT 像是对 MINUS 的补充。也就是说，对称运算符在两个方向上的工作是相同的。UNION 是对称运算符。如果你交换在以前的示例中使用 MINUS 所选取的表，你会得到一个不同的结果。你将得到所有的已得到付款但还没有完成他们的工作的客户（这或许不是一般的商业惯例）。对于另外一个例子，要查找所有已完成他们的工作并且已得到付款的客户，使用如下语句：

```
SQL> SELECT ACCOUNT_ID FROM JOBS_COMPLETED
2> INTERSECT
3> SELECT ACCOUNT_ID FROM ACCOUNTS_PAID;
```

17.6.2 使用布尔转换

布尔表达式是一个求值结果为 TRUE 或 FALSE 的表达式。一条 SQL 语句的 WHERE 子句便是布尔表达式的一个例子。你可以利用这个事实，提供一个将 WHERE 子句（布尔表达式）转化为数值的函数，例如用 1 表示 TRUE，用 0 表示 FALSE。这样做有何用？如果给出四种类型的婚姻状况（S=单身，M=已婚，D=离婚，W=丧失配偶），考虑如下用于获得不同的税率的查询：

```
SQL> SELECT SINGLE_TAX "TAX" FROM TAXES WHERE STATUS='S';
SQL> SELECT MARRIED_TAX "TAX" FROM TAXES WHERE STATUS='M';
SQL> SELECT DIVORCED_TAX "TAX" FROM TAXES WHERE STATUS='D';
SQL> SELECT WIDOWED_TAX "TAX" FROM TAXES WHERE STATUS='W';
```

还是先把数据库设计问题搁置一边，因为这或许是一个不良设计的表，你可以发现为了得到需要的信息，必须执行 TAXES 表的四个全表扫描。这是一个好的示例，因为你正在处理两个以上的不同值。你可以做得更好吗？首先，你需要一个布尔转换。换句话说，你需要一个函数以便在 STATUS='S' 时返回 SINGLE_TAX，在 STATUS='M' 时返回 MARRIED_TAX，诸如此类。Oracle 恰好提供了这样一个函数——DECODE。假如你仔细地研究了下述查询，你会看到它完全替代了以前的四个查询并提供相同的操作结果，然而它仅需要一个表扫描。

```
SQL> SELECT DECODE(STATUS, 'S', 1, 0)*SINGLE_TAX +
2> DECODE(STATUS, 'M', 1, 0)*MARRIED_TAX +
3> DECODE(STATUS, 'D', 1, 0)*DIVORCED_TAX +
4> DECODE(STATUS, 'W', 1, 0)*WIDOWED_TAX
5> "TAX"
6> FROM TAXES;
```

这明显地提高了效率。然而，正如你所料，可读性与可维护性受到了破坏。除此之外，使用 DECODE 函数不总能发挥作用。你或许要使用其他的转换函数。在本例中，你需要的是一个相关数组，DECODE 仅用于提供那种类型的函数功能。显而易见，困难的工作是检查原始查询以提出一个给你相等结果的函数。

17.7 介绍 Oracle 8 的新的索引特性

几个关于索引的新特性已经添加到 Oracle 8 中，毫无疑问，这些新特性在将来的应用开发中会大有用处。这些新特性将在下面讨论。

17.7.1 使用索引分区

Oracle 8 使你能够按分区键将索引分区（物理划分）。这意味着按组成索引的同一列划分

该索引。分区可以存储在分开的表空间中，这些表空间具有各自的存储参数。因此，分区是“子表”。下面便是一个例子：

```
SQL8> CREATE INDEX EML_IDX ON EMPLOYEES(EMPLOYEE_ID)
2> PARTITION BY RANGE (EMPLOYEE_ID)
3> (PARTITION ip1 VALUES LESS THAN (499999)
4> TABLESPACE empl_idx1,
5> PARTITION ip2 VALUES LESS THAN (1000000)
6> TABLESPACE empl_idx2);
```

17.7.2 使用相等分区的本地索引

在Oracle 8中，假如你创建了一个本地索引，Oracle会自动地对它进行同等分区。也就是说，它使用相同的分区键、相同的分区数以及分区界限，这与它引用的分区表相同。本地索引是一种索引，在该索引中一个分区中的所有索引键指向一个表分区中的所有数据。这是一对一映射。全局索引是另一种索引，在该索引中一对一映射并不适用。这确保了表与它的索引被同等分区。除了类似于数据条的较高的可用性外，本地索引的相等分区能够使优化程序知道分区。下面便是一个例子：

```
SQL8> CREATE INDEX EML_IDX ON EMPLOYEES(EMPLOYEE_ID)
2> LOCAL
3> (PARTITION ip1 TABLESPACE empl_idx1,
4> PARTITION ip2 TABLESPACE empl_idx2);
```

17.7.3 使用知道分区的优化程序

正如刚刚提到的那样，假如你创建相等分区的本地索引，优化程序可以通过使用分区知识生成查询计划。因此，它可以并行执行一些操作。

17.7.4 使用唯一索引表

在Oracle 8中，你现在可以创建一张唯一索引表。它也称为一个在位（in-place）的索引。本质上，该表被物理排序，而不是使用B*树的逻辑排序。把逻辑读取从B*树移动到表中能够给性能带来明显的提高，因为数据和索引是相同的。再有，通常把主键作为索引列。一个例子如下：

```
SQL8> CREATE TABLE EMPLOYEES
2> (EMPLOYEE_ID NUMBER(6) CONSTRAINT empl_pk PRIMARY KEY,
3> <column, column, ...>)
4> ORGANIZATION INDEX TABLESPACE empl_dat1;
```

ORGANIZATION INDEX子句告诉Oracle 8这是一个唯一索引表。

17.7.5 使用反向键值索引

反向键值（reverse key）索引是一种索引类型，在该索引中单个列字节的顺序被反向（不是列序）。已经证明反向键值索引在提高Oracle并行服务器（OPS）性能方面非常有效。在CREATE INDEX中使用REVERSE关键字创建一个用于Oracle并行服务器的索引。

第18章 调整内存

本章要点：

UTLBSTAT/UTLESTAT

调整共享池

调整数据库缓存

调整排序

调整多线程服务器

调整锁

再看操作系统集成

在Oracle中，调整内存通常是指调整共享全局区（SGA）。这包括监控和调整共享池（数据字典与库缓存）与数据库缓存。调整内存与调整应用和输入/输出紧密联系在一起，因为调整的一个主要目标——减少或消除争用——在调整过程的各方面都涉及到。例如，本书在调整输入/输出中讨论了它们都有必须调整的内存成分，例如回滚缓冲区和重做日志缓冲器。然而，因为主要强调的是输入/输出争用，所以在调整输入/输出里讨论调整回滚段和重做日志。你还必须调整其他的诸如锁与闩的基于内存的结构，其中一部分将在本章讨论。分类排序是另外一个内存问题。当你对任何一样东西进行分类排序时，不管它是否是 CREATE INDEX、ORDER BY 或一个连结的结果，理想的情况是尽可能地在内存中执行，仅在必要时求助于磁盘。最终，作为一名数据库管理员，你必须积极地参与到操作系统的集成和调整中。

除非RDBMS由专用的数据库计算机的部分组成，否则包括Oracle在内的任何RDBMS必须从操作系统中申请内存。操作系统在低层处理大部分的内存管理，例如用于RDBMS的共享内存。依赖于RDBMS，它可以处理某些高级操作，例如锁定。作为一个示例，在内存管理方面考察一下Sybase与Oracle。Sybase选择使用一个大型的共享内存段并且通过低级操作系统调用来管理它本身。Oracle选择同样的方法。Sybase处理它自己的锁定和多线程过程中的进程间通信问题。另一方面，Oracle管理部分锁定和内部的进程间通信，还要依赖操作系统做一部分工作。这些处理内存的方法是RDBMS体系结构的直接结果。不论发生哪种情况，一个微型操作系统和其中的RDBMS必须和操作系统紧密结合，这样可以有效地处理它的资源请求。这不仅适用于内存，而且适用于所有的资源，包括输入/输出。但有几个例外。在UNIX操作系统中的原始磁盘空间就是一个明显的例外（参见附录A）。现在，把你的注意力转向实际的数据收集和诊断内存问题上来。

18.1 UTLBSTAT/UTLESTAT

正如你在第16章中所学习的那样，UTLBSTAT和UTLESTAT脚本为你的全部诊断工作打下基础。经常用人工查询或诸如V\$SYSSTAT的V\$动态性能视图的自定义脚本来补充这些脚本。通过在不同的工作中使用这些脚本，你可以用它们来保证同其他Oracle数据库管理员和顾问针对某一系统的性能进行通信。

正如你看到的，这些脚本通常存在于 \$ORACLE_HOME/rdbms<version>/admin 子目录中。

在实际使用它们以前，应在你的 `init.ora` 参数文件中设置 `TIMED_STATISTICS=TRUE` 或在你的会话层设置 `ALTER SESSION SET TIMED_STATISTICS=TRUE`。注册到 `svrmgr1`（服务器管理器行方式，版本 7.1 或更高）。接下来 `CONNECT/AS SYSDBA`（或者是 `CONNECT INTERNAL`）。然后，运行 `utlbstat.sql`。

这样就创建了你的开始集合表和视图（在 `SYS` 模式下）；这些对象已经在它们内部用 `BEGIN` 命名。然后选取开始统计数字并把它们存放在那里。接下来，假如你的应用还没有运行的话，那么运行它。作为一名数据库管理员，你的目标是在系统的峰值活动期间捕获系统统计数字。当峰值过后，或在一些合理的时间周期之后，运行 `utlestat.sql`。此时间间隔不必太长，但是必须充足。例如，假如你的峰值装载活动每天持续 2 小时（120 分钟），那么采样至少应是该时间的 20%，即大约 24 分钟，最好取该间隔的中间值（还有，你的数据库必须在该采样运行的整个期间内保持不变。这不会成为一个问题，然而，因为你正试图从一个活动的、峰值负载的生产数据库中采样，所以希望数据库无论如何要保持不变）。这样就创建了你的结束和差别集合表与视图。结束对象已经在它们内部用 `END` 命名。结束统计数字被选取并存储在那里。开始与结束统计数字之间的差别存储在差别表中。最终，`utlestat.sql` 从对象、格式化数据选取并在 `report.txt` 文件中存储信息。然后，困难的工作——解释开始了。

18.1.1 解释结果

不会有两个顾问在对 Oracle 进行调整的每一个细节上存在相同的意见，但是他们会同意大部分事情。经常听到有人说性能调整一半是科学，一半是艺术。我对这个评价表示赞同。换句话说，解释大部分是艺术，因为那部分工作需要人对系统的实力和弱点做出有价值的判断。

使用高级语法分析和程序的解释可以是并且已经是半自动或全自动进行的。然而，不可否认的事实是不管这个过程如何程序化，它仍然具有部分主观性。尽管它可以被编程，然而相同的两个调整程序或程序员会使用相同的规则或准则来判断“好的”或“差的”性能水平吗？大部分不能。此外，这仅考虑了通用调整。当你面对特殊的应用问题时，这些所谓的完全自动的性能调整程序很快就会失去它们的用处。

现在我将总结一下如何正确地使用 `UTLBSTAT/UTLESTAT` 与调整你的数据库系统：

- 1) 在实例或会话层设置 `TIMED_STATISTICS=TRUE`。
- 2) 登录到 `svrmgr1`（服务器管理器行方式）并 `CONNECT/AS SYSDBA`（或 `CONNECT INTERNAL`）。
- 3) 在监控期开始时运行 `$ORACLE_HOME/rdbms<version>/admin/utlbstat.sql`，其中监控期是正常的峰值活动的持续时间。
- 4) 在监控期结束时运行 `$ORACLE_HOME/rdbms<version>/admin/utlestat.sql`。
- 5) 通过使用合理的准则对 `report.txt` 中的结果进行解释（不久我会涉及到准则问题）。
- 6) 完成推荐的修改或调整（如果有的话）；这可以从修改一个 `init.ora` 参数到重新组织你的物理设计的任何事情。
- 7) 从步骤 3 开始重复此过程，直到对步骤 5 的结果满意为止。

18.1.2 回顾报告文件

`report.txt` 文件中包含了大量用来帮助你调整应用、内存和输入/输出的信息。在高层次上，

它有关于共享池（数据字典与库缓存）、数据库缓冲区缓存、每个事务/登录数据、每个表空间/文件的输入/输出和等待事件的统计数字。你可以使用所有这些统计数字或它们中的一部分。

提示 确信你使用的单个统计数字与你的应用有关。在最后的分析中，你的用户会真正测试你的应用性能是否是“优良的”。

有些统计数字对同一项性能提供了不同的视图而且应一致。当这些视图不一致时，解释成为主观事物和应用特定的。例如，假如你有 3 个统计数字，以不同的方式报告有关库缓存的性能。如果这 3 个中的 2 个根据已接受的准则显示性能是“优良的”，而剩余的那一个显示性能是“不良的”。如果出现这种情况怎么办？这意味着你的应用性能是优良的吗？这就要依赖于你的应用类型以及它与它有关的每一个统计量的含义。例如，假如显示优良性能的两个统计量对于批处理系统更有意义，而你有一个 OLTP 系统，那么那些测量值会令人误解。

18.2 调整共享池

正如你在第 6 章中所学习到的那样，共享池基本上包括两个主要的结构：

- 数据字典缓存。
- 库缓存。

那些存储供稍后重新使用的已经过语法分析的 SQL 语句的区域是共享的 SQL 区域。专用的 SQL 区是一些与应用内游标持续时间有关的区域。在共享池的调整中，调整内存与调整应用明显重叠。本章增强了前面的一章（第 17 章）。

共享池是一个高速缓冲结构。和所有其他的高速缓存结构一样，它是内存驻留数据结构。

高速缓存（cache）是一种特殊类型的缓冲区。尽管缓冲区是一个“无智能的”机制，仅用来为数据在高速内存与慢速磁盘之间的路线上提供临时存储区，但高速缓存是一个“灵巧”的机制，保留关于那条信息的内存或部分信息的内存，因此高速缓存可以尽可能地避免到磁盘的不必要的行程。当一条输入/输出请求发出后，高速缓存查看该请求要求的数据是否已在内存中。如果数据在内存的话，它亲自回答该请求，返回请求的数据。这称为一个命中（hit）。如果数据不在内存的话，就会有一次到磁盘的访问。这被称为一个遗漏（miss）。

对于几乎所有的高速缓存装置来说，保证高效性能的准则是 90%+ 命中率，命中率可以被定义为 $100 \times (1.00 - (\text{遗漏的数量} / \text{请求的数量}))$ ，其中请求的数量 = 遗漏的数量 + 命中的数量。例如，假如你的高速缓存有 4 个遗漏和 46 个命中，那么你的命中率是 $100 \times (1.00 - (4/50)) = 100 \times (1.00 - 0.08) = 92\%$ ，这个数字非常好。高速缓存通常由一个最近最少使用的（LRU）算法来管理，该算法保证在任何给定的时刻，最近使用最多的（MRU）数据被保存在高速缓存中并且最近最少使用的数据被拿走。

当 Oracle 对一条 SQL 语句进行语法分析时，通过把一个数学公式应用到 SQL 语句的字母数字文本并使用该结果在高速缓存中存储（供以后查找和利用）它，Oracle 便可以在库缓存（library cache）中为该 SQL 语句分配一个 SQL 区。换句话说，它使用一个哈希函数（hash function）。为了一条语句可以被重复使用，该语句必须是相同的。例如，当在库缓存中对下述语句进行存储时，在 Oracle 看来这些语句并不是相同的：

```
SELECT * FROM EMPLOYEES;  
SELECT      * FROM EMPLOYEES;  
SELECT * FROM employees;
```

尽管对于你我来说，它们在功能上是相同的，但是它们的同一个哈希值不同。为了使它们成为相同的哈希值，在语句中应没有空白（空格、制表键、缩进或不可打印的控制字符）或其他不同。此外，下面的两条语句不能重复使用相同的哈希 SQL 区语法计划：

```
SELECT * FROM EMPLOYEES WHERE EMPLOYEE_ID=927354;  
SELECT * FROM EMPLOYEES WHERE EMPLOYEE_ID=746293;
```

这是事实，因为在 WHERE 子句中使用了直接量 927354 和 746293。因为在哈希函数中字母数字语句的输入是不同的，所以它们不可能放到相同的库缓存位置。如何才能放到相同的位置？除了较早前讨论过的 DSS 外，你可以使用绑定变量。绑定变量（bind variable）使 SQL 语句足够通用以便重新使用并且具有参数值而不是常量：

```
SELECT * FROM EMPLOYEES WHERE EMPLOYEE_ID=:EMPID;
```

这种类型的语句可以被重复使用并且一般存在于嵌入式 SQL 应用代码中，其中：EMPID 是绑定变量——在该例子中，主 SQL 变量是一个 C 整型。这个 C 变量的值现在可以采用 927354、746293 等等，可以在库缓存中重复使用。

18.2.1 改进库缓存性能的方针

将应用内部的不必要的语法分析调用减到最小。语法分析耗费大量的 CPU 时间。因为高速缓存和缓冲区被用到，作为一个副作用，它也要耗费大量的内存。游标打开和关闭应被仔细地放置在应用中，以方便为多条 SQL 语句重复使用专用 SQL 区。数据库管理员必须增加 init.ora 的参数 OPEN_CURSORS 以便为分配足够的游标空间（专用 SQL 区）做准备。要确定你的应用的效率是否低下，运行 SQL TRACE/tkprof（第 17 章）并检查用于语法分析的计数列是否接近于执行（或取数据）的值。假如这样的话，应用为几乎每一次执行（或取数据）重新进行语法分析。

最大程度地重复使用那些必须被分析的语句。如上所述，SQL 语句要被重复使用必须相同。一种有助于重复使用的方法是采用所有应用开发者必须遵守的标准编码方式，例如“总是用大写字母编写 SQL 语句”。你可以让一个程序通过你的所有开发代码并在开发者的适应性较差的情况下实施它。此外，除了 DSS 应用外，在适当的时候使用绑定变量。这很好理解，与硬编码的直接量相比，绑定变量使你的应用通用化了。如果没有库缓存重用，这在可维护性方面付出了代价。

“钉住”（pin）内存中频繁使用的程序对象。在 Oracle 中，如果使用一个特殊的共享池包 DBMS_SHARED_POOL，那么游标、触发器、过程或包可以保持在内存中。为了创建这个包，运行 \$ORACLE_HOME/rdbms<version>/admin/dbmspool.sql 脚本。你或许还需要运行 \$ORACLE_HOME/rdbms<version>/admin/prvtpool.sql。查看你的操作系统平台版本以判断情况是否如此。要钉住内存中的一个程序对象，使用如下语句：

```
SQL> EXECUTE DBMS_SHARED_POOL.KEEP('<object_name>');
```

要解除这个固定：

```
SQL> EXECUTE DBMS_SHARED_POOL.UNKEEP('<object_name>');
```

要确定对象是否被成功地钉住：

```
SQL> SELECT SUBSTR(NAME,1,25), KEPT FROM V$DB_OBJECT_CACHE;
```

假如对象被钉住，那么 KEPT 列的值为 YES，否则为 NO。

把库缓存中的碎片减到最小。如果你不消除碎片的话，你的应用会遇到 ORA-04031 错误

(没有足够的连续空间)。一种方法是钉住内存中频繁使用的大对象。对于并不频繁使用的大对象, 则为它们保留一些空间。你可以通过设置 init.ora 的参数 SHARED_POOL_RESERVED_SIZE 和 SHARED_POOL_RESERVED_MIN_ALLOC 来完成这项工作。另外设置用于你的大对象的共享池“保留区”。基本上, 要保证你所需的大对象能够找到空间。把 SHARED_POOL_RESERVED_SIZE 设置为同时装载的最大对象的最大字节数。为了使用由 SHARED_POOL_RESERVED_SIZE 指定的保留区, 把 SHARED_POOL_RESERVED_MIN_ALLOC 设置为可以用于一个对象的最小字节数。要确定你希望在保留区中保存的一个特殊对象的大小, 使用如下的语句:

```
SQL> SELECT SUBSTR(NAME,1,25) "NAME", SHARABLE_MEM
2> FROM V$DB_OBJECT_CACHE
2> WHERE NAME='<object_name>';
```

还有, 要确定需要把 SHARED_POOL_RESERVED_SIZE 设置为多大, 使用如下语句:

```
SQL> SELECT SUM(SHARABLE_MEM)
2> FROM V$DB_OBJECT_CACHE
3> WHERE SHARABLE_MEM >= <SHARED_POOL_RESERVED_MIN_ALLOC>;
```

因此, 要执行前面的查询, 你必须有一些关于分类、最小值和“大对象”的粗略想法。所以, 应采取如下步骤:

- 1) 按你的要求设置 SHARED_POOL_RESERVED_MIN_ALLOC。
- 2) 把 SHARED_POOL_RESERVED_SIZE 设置为上一个查询的输出值, 加上一个裕量 (例如 10%)。

再有, 你可以把 CURSOR_SPACE_FOR_TIME 设置为 TRUE 以免与游标有关的 SQL 区在被程序关闭以前从库缓存中删除。

警告 假如下面的任何一条符合你的状况时, 那么不要改变 CURSOR_SPACE_FOR_TIME 的 FALSE 缺省值:

- V\$LIBRARY_CACHE 中的 RELOADS 总是显示 0 值。
- 你正在使用 Oracle Forms 或 SQL *Forms。
- 你使用任何动态的 SQL。

除了刚刚学过的许多基于应用的内存问题与调整方法外, 你可以看一下 utlbstat.sql/utlestat.sql (report.txt) 提供些什么。一个 report.txt 的示例放在本书附带的 CD-ROM 上。特别是, 你可以看一下 report.txt 中的第一部分, 即库缓存统计数字。确保用于 SQL 区库的 GETHITRATIO 在 90s 内, 最好是高 90s。假如你不运行 utlbstat.sql/utlestat.sql, 你还可以使用如下语句:

```
SQL> SELECT GETHITRATIO
2> FROM V$LIBRARYCACHE
3> WHERE NAMESPACE='SQL AREA';
```

然而, 此方法是自上一个实例启动以来并包含了上斜坡 (ramp-up) 时间的平均运行时间。上斜坡是任何系统开始启动时经历的初始化过程。在此期间收集到的统计数字会使人误解并且没有用处, 因为它们基本上反应了必须填充各种 Oracle 缓冲区和高速缓存的物理读取。因此, 有时这也叫做缓存上升 (caching up)。正如你较早前学到的那样, 希望在峰值装载时间里采样统计数字。当一个系统在峰值活动中运行了一段时间时, 该系统被认为是处于平衡状态

(in equilibrium)。假如你在峰值活动到达后运行 utlbstat.sql, 然后在峰值活动刚要停止前运行 utlestat.sql, 你就有了一个有效的样本。因此, 从V\$动态性能视图中选取统计数字会令人误解, 特别是如果你在实例持续期的早期采样它们。实例持续时间越长, V\$视图的统计数字就越好, 因为峰值与正常活动越来越多地超过了初始启动期。要保持准确的话, 对 utlbstat.sql/utlestat.sql也应该这样。

假如GETHITRATIO小于0.90, 运用改进库缓存性能的准则。这一般意味着应用代码可以被更有效地重新编写。

此外, 从report.txt的第一部分(即库缓存统计数字)中计算用于 SQL区库的RELOADS/PINS率。

假如RELOADS/PINS率大于0.01, 通过设置init.ora的参数SHARED_POOL_SIZE来增加总共享池的容量(以字节为单位)。记住, 这设置了整个共享池, 不仅包括库缓存, 而且包括数据字典缓存。

检查report.txt的门统计数字部分, 确保对于 LATCH_NAME='library cache'的HIT_RATIO处于高90s中。其他测量高门争用的方法如下所示, 高门争用是对不良的库缓存(和总共享池)性能的提示。

查看report.txt的System wide wait events部分, 检查Event Name='latch free'的(wait) Count column。

还要运行如下查询:

```
SQL> SELECT COUNT(*) "LIBRARY_LATCH_WAITS"  
2> FROM V$SESSION_WAIT W, V$LATCH L  
3> WHERE W.WAIT_TIME = 0  
4> AND W.EVENT='latch free'  
5> AND W.P2 = L.LATCH#  
6> AND L.NAME LIKE 'library%';
```

如果下面任何一项是正确的话, 运用改进库缓存性能的准则; 如果必要, 增加 SHARED_POOL_SIZE的设置值:

- LATCH_NAME='library cache'的HIT_RATIO小于0.98
- Event Name='latch free'的(wait) Count大于10
- LIBRARY_LATCH_WAITS大于2

正如你所预料的一样, 共享池的数据字典缓存(data dictionary cache)部分保留用于 Oracle数据字典的缓存结构。SHARED_POOL_SIZE参数是间接测量数据字典缓存大小的唯一方法。下面是保存在SYS和SYSTEM模式(SYSTEM表空间)中的对象:

- X\$表。
- V\$视图。
- 数据库管理员视图。
- 用户视图。

度量、诊断与调整库缓存以便它很好地运行, 这样有助于数据字典缓存提高其性能, 因为库缓存和数据字典缓存都共存于共享池中。它们不是分别配置的。

有两种方法来测量数据字典缓存性能。其中一种方法是查询 V\$ROWCACHE视图:

```
SQL> SELECT SUM(GETMISSES)/SUM(GETS) "DC_MISS_RATIO"
```

```
2> FROM V$ROWCACHE;
```

另一种方法是使用 report.txt 的数据字典部分。计算所有的 GET_MISS 的总和，用所有 GET_REQS 的总和除以 GET_MISS 的总和，得到一个类似的 DC_MISS_RATIO。

如果这两种方法中的任何一种产生的 DC_MISS_RATIO 大于 0.15，那么增加 SHARED_POOL_SIZE（然后重新测试）。

18.2.2 多线程服务器问题

你还要简要考虑多线程服务器（MTS）和服务器内存与客户（用户）内存的分配问题。有一个 SGA 就有一个用户全局区（UGA），它包含了用户会话信息、排序区和专用 SQL 区。通常，缺省的 Oracle RDBMS 实例（称为专用的服务器）导致一个用户过程与服务器过程的一对一映射。使用多线程服务器，用户全局区被移动到共享池中。剩余的进程特定的内存被保留在进程全局区（PGA）中并且它保存不能共享的信息。在这种方法中，使用多线程服务器时需要的内存总数不会超过再分配的专用服务器。然而，你必须增加 SHARED_POOL_SIZE。要帮助对重新定位到 SGA 的用户全局区进行度量，运行如下查询：

```
SQL> SELECT SUM(VALUE)
2> FROM V$SESSTAT SE, V$STATNAME SN
3> WHERE SN.NAME = 'max session memory'
4> AND SE.STATISTIC# = SN.STATISTIC#;
```

该查询产生了自实例启动以来已使用的 UGA 会话内存的最大数量。你或许希望多次采样它并得到这些最大值中的最大值，然后把 SHARED_POOL_SIZE 增加到此最大值。

注意 在 Oracle 8 中，对于上一个查询使用 'session uga memory max'。

使用多线程服务器，你可以对服务器和用户内存的分布实施一些控制。影响用户内存的两个 init.ora 参数是：

■ SESSION_CACHED_CURSORS

■ CLOSE_CACHED_OPEN_CURSORS

假如希望的话，把 SESSION_CACHED_CURSORS 设置为缓存在用户内存区中的会话游标的最大期望值。只要重新进行语法分析的工作量较小，这就有助于在以增加用户内存为代价的前提下卸载服务器内存请求。当语句被频繁地重复使用时，可选择是否设置这个参数。

CLOSE_CACHED_OPEN_CURSORS 的缺省设置值是 FALSE，即游标不在 COMMIT 时关闭。假如大部分 SQL 语句很少被重复使用的话，可把该值设置为 TRUE。

警告 确保对 CURSOR_SPACE_FOR_TIME 和 CLOSE_CACHE_OPEN_CURSORS 的设置值不会发生冲突。例如，把两者都设置为 TRUE 或都设置为 FALSE 看来是最好的办法。

下面，把你的注意力转向数据库缓冲区缓存。

18.3 调整数据库缓冲区缓存

或许用来提高 Oracle 系统性能的一个最重要的调整变化是，正确地设置你的数据库缓冲区缓存的大小。数据库缓冲区缓存（database buffer cache）是 SGA 中的高速缓存结构，它可以把最近最多使用（MRU）的 Oracle 数据块的拷贝保存在内存中。度量该区大小的两个参数是：

■ DB_BLOCK_SIZE

■ DB_BLOCK_BUFFER

DB_BLOCK_SIZE是指一个Oracle数据块的大小。在UNIX操作系统平台上该值可以在2KB（2048个字节）和64KB（65536个字节）之间变化。对于性能来说，该值通常是越高越好。假如你的数据库已经用一个相对较小的块大小（例如为2KB的缺省值）创建，假如可行的话，考虑重新建立一个数据库。如果想重建数据库，执行下述步骤：

- 1) 关闭实例。
- 2) 把你的数据库完全导出（如果可行）。
- 3) 在你的init.ora中增加DB_BLOCK_SIZE。
- 4) 启动该实例。
- 5) 把你的数据库作为SYS重新导入。

如果你的数据库太大而不能使用导出/导入的话，你要确保有你的表数据的ASCII文件（如果必要；选取它们或者使用第三方工具）。重新运行你的DLL以创建脚本。可能的话，使用SQL*Loader以并行方式重新装载该表。

DB_BLOCK_BUFFER是保存在内存中的Oracle数据块的数量。每个缓冲区等于一个数据块。该值应足够大以便产生一个有效的高速缓存命中率，但又不会过高以致于引起操作系统分页。最后你希望做的是通过操作系统把你的SGA在内存中或内存外分页。分页是当Oracle达到DB_BLOCK_BUFFER大小时完成的主要工作，你不希望在Oracle下进行操作系统内存分页。在有请求时才对它们进行分页。因此，你的数据库缓冲区缓存以及共享池将装配在实（可用的核心）内存中并且它们的大小不会接近或大于该内存。合理的大小是总系统内存的1/2到3/4。例如，在一个1G的UNIX系统中，你或许要设置数据库缓冲区缓存以便你的SGA占用大约3/4的总量，即750MB。你必须仔细地考虑其他的Oracle与非Oracle应用内存要求、用户内存要求以及操作系统要求。数据库缓冲区缓存的容量是：

DB_BLOCK_BUFFERS x DB_BLOCK_SIZE

数据库缓冲区缓存有点用词不当。回忆一下高速缓存是一种特殊类型的缓冲区。因此，缓冲区缓存这个词实际上是多余的，也有点混乱（或许Oracle本应该叫它数据库块缓存？）。总之，你需要真正理解的是数据库缓冲区缓存保存了Oracle的数据块。它与共享池不同，因为它保存数据，而不保存程序。

在把Oracle数据块传送到用户进程以前，Oracle RDBMS服务器总是把Oracle数据块读取到数据库缓冲区缓存中。用户进程或应用总是从数据库缓冲区缓存中读取（和写入）。下面是在缓冲区管理中的输入/输出请求步骤：

- 1) 用户选择数据（请求数据块）。
- 2) 服务器在数据库缓冲区缓存中查找它。
- 3) 如果通过哈希函数，服务器在LRU列表中发现了用户选取的数据，那么服务器便返回它。
- 4) 如果服务器没有发现它，那么服务器从磁盘上的数据文件块中读取并在适当时把它依附于（使用哈希函数）LRU列表的MRU或LRU端。
- 5) 如要用户不修改它，那么这个输入/输出请求步骤就完成了。
- 6) 如果用户修改它，DBWR把数据块（不干净的缓冲区）写回到它在磁盘上数据文件中

的位置。

一次只能有一个数据块访问索引。全表扫描可以用一条请求完成多个数据块的读取。通过如下的设置来设置块（批量大小）的数量：

```
DB_FILE_MULTIBLOCK_READ_COUNT =<要被读取的块的数量>
```

缓冲区可以是空闲的（干净的）、肮脏的、当前的或一致性读取的（回滚）。一个空闲缓冲区（free buffer）是指自从实例启动以来还没有被使用或曾经被使用过但目前可用的缓冲区。脏缓冲区（dirty buffer）是指已被使用，但还没有被刷新或者在检查点由 DBWR存储的缓冲区。当前缓冲区（current buffer）是指用于INSERT、UPDATE或DELETE服务中的缓冲区。从本质上来说，当前缓冲区更容易变脏。一致性读取（Read-consistent）缓冲区对SELECT语句和回滚进行服务。在全表扫描服务中读取的块被放置在 LRU缓冲区链的最近最少使用（LRU）端。然而，你可以在该链的MRU端缓存整个表。

如何调整数据库缓冲区缓存？因为内存输入/输出比磁盘输入/输出快几个数量级（它们是纳秒与毫秒的关系），所以希望尽可能通过内存满足输入/输出请求。这也就是说，与必须从磁盘中读取数据块相反，希望它们经常（超过总时间的 90%）存放在数据库缓存中。还希望把LRU争用减到最小。LRU缓冲区链或列表通过闕机制被锁定，就像那些用于 Oracle内核的缓冲区链一样，尤其是库缓存中的缓冲区链。如同任何一种闕方法一样，你必须有足够的缓存，因为闕（也称为自旋锁）和信号量一样不包含查询机制。

正如在调整库缓存中所提及的那样，几乎任何一个缓存结构的缓存命中率大于或等于 90%就被认为是好的。还有，缓存命中率是命中与请求之比。换句话说，它是 Oracle数据块满足输入/输出请求的次数除以总输入/输出请求数所得到的结果。当数据块在缓存中的时候叫做命中，当数据块不在缓存中的时候叫做遗漏（必须从磁盘中读取）。至少有两种方法来测量数据库缓冲区缓存的命中率。一种方法是运行如下的查询：

```
SQL> SELECT 1-(P.VALUE/(D.VALUE+C.VALUE)) "CACHE HIT RATIO"  
2> FROM V$SYSSTAT P, V$SYSSTAT C, V$SYSSTAT D  
3> WHERE P.NAME='physical reads'  
5> AND D.NAME='db block gets'  
4> AND C.NAME='consistent gets';
```

其中，'physical reads'是从磁盘中读出的数据块的数量，'db block gets'是从缓存的当前块拷贝中读出的数据块的数量，'consistent gets'是缓存中一致性读取（回滚）的块拷贝的数量。因此，数据库缓存命中率公式如下：

$$1 - (\text{physical reads} / \text{logical reads})$$

逻辑读取的数量等于当前拷贝读取的数量加上一致性拷贝读取的数量。物理读取的数量代表了遗漏的数量。逻辑读取的数量代表了请求的数量。你还可以从 report.txt的Statistics部分中得到物理读取、数据库块读取和一致性读取。使用相同的公式计算缓存命中率。

假如数据库缓存命中率小于 0.90，那么增加 DB_BLOCK_BUFFERS，然后重新运行 utlbstat.sql/utlestat.sql。增加这个参数，就像大部分 init.ora参数一样，需要关闭实例并且再启动。然而，这不会造成重大问题，例如在数据库已经建好后改变 DB_BLOCK_SIZE。

在Oracle中有一种方法用来测试增加更多缓冲区后的效果。为什么你要这样做，而不是简单地增加DB_BLOCK_BUFFERS？情况经常是你需要把更多的缓冲区增加到你的数据库缓冲区缓存中，但是没有足够的实内存支持，因此，这项技术可以判断是否应购买更多内存。要

这样做，关闭实例并设置：

```
DB_BLOCK_LRU_EXTENDED_STATISTICS = <n>
```

其中<n>是你希望增加的缓冲区的数量。

然后再次启动该实例。让你的应用正常运行一段较为合理的时间，就像你运行 utlstat.sql/utlestat.sql的时间一样。X\$KCBRBH表包含了你需要的信息。执行如下的查询：

```
SQL> SELECT SUM(COUNT)
2> FROM X$KCBRBH
3> WHERE INDEX < <n>;
```

该查询返回了通过增加这 <n>个缓冲区而获得的附加缓存命中数。要确定你新假设的数据库缓存命中率是多少，把 <n>添加到原始公式中：

$$1 - (\text{physical reads} - \langle n \rangle) / (\text{logical reads})$$

正如你所看到的那样，它导致了一个较高的缓存命中率。作为一个示例，假设从 report.txt 中有 physical reads=40000, consistent gets=100000 并且 db block gets=30000。那么，在没有增加任何缓冲区以前你的缓存命中率是

$$1 - (40000 / (100000 + 30000)) = 1 - (40000 / 130000) = 1 - (0.31) = 0.69$$

该值低于 0.90。假定你想要增加 10000 个缓冲区。设置你的扩展统计数字并查询 X\$KCBRBH 表，该表表明将获得 30000 个命中。新假设的缓存命中率（伴随着性能的提高）将是

$$1 - ((40000 - 30000) / (130000)) = 1 - (10000 / 130000) = 1 - (0.08) = 0.91$$

该值明显提高。当然，这个例子使用了经过设计的数字，但是它足以强调一点，即当你用完内存并需要判断是否应购买更多内存时，尤其是要支持数据库缓冲区缓存的情况下，缓存命中率是非常有用的。最困难的事情是确定要添加多少缓冲区以使你得到你所需要的附加命中数目。建议你设法增加此百分数；首先增加 10%，然后根据结果再决定是增加还是减小。有一个类似的方法用于测试除去“不必要的”缓冲区后的效果。然而，任何数据库管理员都知道很少有不必要的缓冲区，所以本书不涉及它。

你可以把特殊的系统等待事件看作用来帮助测试数据库缓存性能的准则。像较早前在调整库缓存中所显示的一样，你可以从 report.txt 的整个系统等待事件部分或者 V\$SYSTEM_EVENT 与 V\$SESSION_WAIT 中收集 'buffer busy waits' 和 'latch free'。还有，从 V\$SYSSTAT 或 report.txt 中检查 'free buffer inspected'。特别是，假如下面任何一项大于 0 的话，说明有 contention 问题：

- 缓冲区忙等待。
- 空闲。
- 受检查的空闲缓冲区。

init.ora 的参数 DB_BLOCK_LRU_LATCHES 被缺省设置为一台 SMP 计算机上 CPU 数量的一半。根据以前的经验，如果该值看起来不够的话，那么把它增加到 CPU 数量的 2 倍。

警告 如果你没有增加 DB_BLOCK_LRU_LATCHES 的依据，就不要这样做，你可以一点点地增加它——例如，从 CPU 数量的 1/2 增加到 3/4，等等。如果一个预测的工作量较重，你可以把 DB_BLOCK_LRU_LATCHES 增大一点儿。

如前所述，如果你在 MRU 端缓存整个表的话，你可以减小整个表对放置在 LRU 列表的 LRU 端的数据块进行搜索这种现象。你如何做这项工作呢？要么使用有非缺省值 CACHE 子句

的CREATE TABLE，要么在引用该表的第一个查询中嵌入一个CACHE提示。把init.ora的参数CACHE_SIZE_THRESHOLD设置为每张表允许保存的数据块的最大数量。一般，当你预计要经常重复使用相同的表（例如查询表）时，使用这些方法。当心不要保存太多的表，这样就达不到缓存的目标。情况变化很快，尤其是如果大部分非查询的大表不能得到至少是部分缓存的机会时更是如此。

假如你运行 \$ORACLE_HOME/rdbms<version>/admin/catparr.sql脚本，它便创建V\$CACHE视图以及仅与 Oracle 并行服务器（OPS）有密切关系的其他视图。然而，V\$CACHE视图非常有用，因为它按对象提供了一个数据块到数据文件的映射。因此，你可以确定哪一个对象（例如表）在当前数据库缓冲区缓存中有数据块。它可以告诉你，你保存查询表的努力是否已起作用。该视图像许多V\$视图一样不是动态的，所以不管你是创建了新的对象还是改变了你当前对象的存储参数，你必须重新运行 catparr.sql脚本。

18.4 调整排序

排序（sort）是按某种顺序放置事物的过程，例如按字母顺序或按数字顺序，按升序或降序等等。和任何RDBMS一样，Oracle会由于各种原因而要求进行排序，这些原因中有时隐含地是因为一条SQL语句的性质，有时明显是接受用户的请求。总之，排序消耗大量的CPU时间、内存和磁盘。

首选的最优策略是尽可能地避免不必要的排序。然而，在多数情况下，当排序不能被避免时，则需要对排序进行调整以便它可以最佳地运行。排序可以完全在内存中发生，这是所希望的情形。然而，更有可能溢出到磁盘排序，特别是有大表时，不管数据库有多么好的物理设计，排序也极其消耗时间。

第二条最优策略是尽可能地在内存中排序，仅在不得已时在磁盘上排序。当然，这暗指分配足够的临时磁盘空间（事实上是TEMP表空间），并从Oracle数据文件、回滚段和重做日志的剩余部分中物理地分离该空间。

18.4.1 什么是触发排序

显而易见，CREATE INDEX语句需要在索引键上执行一个排序操作用来建立B*树结构。ALTER INDEX...REBUILD同样需要相同的排序。然而，你可以选择在操作系统层对数据进行排序，然后用NOSORT选项创建该索引。这通常不会给你任何东西，除非你碰巧有该数据的一个已排序拷贝，因为你仅仅用RDBMS排序换取操作系统排序，这根本称不上一个交易。其他的选择包括使用一个快速的第三方排序工具（例如 SyncSort），或利用Oracle的并行查询选项（PQO）来使用SQL*Loader并以并行非排序方式装载数据。

ORDER BY与GROUP BY通常需要排序。然而，在大多数情况下，在索引列上的ORDER BY使用已经排序的索引。要在不执行的情况下验证这一点，按照在第17章中讨论过的那样使用EXPLAIN PLAN。DISTINCT限定符必须使用一个排序方法（除非它在具有唯一索引的列上使用）来消除重复的列值。同样，UNION必须消除重复的行。然而，由定义可知，UNION ALL允许重复行的存在，所以因为它不消除重复，不需要排序。假如在两张UNION表上实施主键，就不会有任何重复，所以UNION ALL是对UNION操作的推荐替代品。INTERSECT与MINUS也需要消除重复，但消耗比UNION操作小。类似，IN或NOT IN需要排序，特别是如

果它们支持嵌套子查询的话，就更需要排序。连结操作需要对连结键上不存在索引的表进行排序。更平常的情况是在主键上连结表（已经有唯一索引），因此不必对任何表排序。下面的列表总结了能够触发排序的SQL命令或操作符：

- CREATE INDEX, ALTER INDEX...REBUILD
- ORDER BY, GROUP BY
- DISTINCT
- UNION, INTERSECT, MINUS
- IN, NOT IN

18.4.2 排序参数

影响排序操作的两个主要的 init.ora 参数是：

- SORT_AREA_RETAINED_SIZE：一个内存中的排序可以使用的最大内存数。
- SORT_AREA_SIZE：外部磁盘排序操作（包括临时段的分配）可以使用的最大内存数。

假如一个排序操作需要比 SORT_AREA_RETAINED_SIZE 还多的内存用于内存中的排序，那么它就试图在 SORT_AREA_SIZE 中执行一个外部磁盘排序，在进程中分配一个临时段。假如排序操作需要更多的内存，那么它把该排序任务分为多个排序操作（sort run）并为此分配多个临时段。服务器进程每次对一个段排序，返回已排序段的合并结果。这些内存分配不存储在 SGA 共享池中，但当使用 MTS 时例外。相反，它们是 UGA 的一部分。如果你正在使用 MTS，它们是 SGA 共享池的一部分，因为 UGA 被重新定位在那里。

使用 EXPLAIN PLAN（参见第17章），你会发现许多 SQL 语句在它们的执行计划中需要多个排序。当前正在执行的排序被称为活动排序（active sort）。连结排序（join sort）是一种支持连结操作的排序。任何一个活动排序需要 SORT_AREA_SIZE。任何一个连结排序需要 SORT_AREA_RETAINED_SIZE。这些设置仅适用于专用服务器。对于 PQO 来说，每一个并行查询服务器需要 SORT_AREA_SIZE。然而，两组并行服务器可以同时工作。所以，对于 PQO 来说，设置如下值：

- $\text{SORT_AREA_SIZE} \times 2 \times (\text{并行度})$
- $\text{SORT_AREA_RETAINED_SIZE} \times (\text{并行度}) \times (\text{排序数} > 2)$

对于 PQO 来说，最佳值是 1MB。更高的值也不会产生较好的性能。一般说来，除了 MTS 以外，设置 $\text{SORT_AREA_SIZE} = \text{SORT_AREA_RETAINED_SIZE}$ ，对于 MTS 需要一些特殊的考虑。

提示 对于 MTS 来说，把 SORT_AREA_RETAINED_SIZE 设置为小于 SORT_AREA_SIZE 的数。作为一条准则，你可以如下设置：

```
SORT_AREA_RETAINED_SIZE =  
(SORT_AREA_SIZE / the number of expected concurrent sorts),  
but not less than 1/10 (SORT_AREA_SIZE).
```

当一个排序不能完全在内存中进行时，必须创建临时（排序）段。也就是说，像已经讨论过的那样，当排序操作对内存的需要超过了 SORT_AREA_RETAINED_SIZE 的设置值时，Oracle 就需要分配一个临时段并设法在 SORT_AREA_SIZE 中工作。一个真正的临时表空间（7.3 版本或更高）段不能包含任何永久对象，只能由单独的排序段组成。临时表空间用

CREATE或ALTER TABLESPACE <tablespace_name> TEMPORARY...语法创建。再有，这些临时表空间由一个段组成，该段最初由需要它的第一个排序创建。随着排序并行性和操作容量的增加，该段在区间内增长。

你如何设置 INITIAL 与 NEXT 区间参数。一个有用的指导是设置 INITIAL=NEXT= (<由数据文件或磁盘规定的最大容量 >/<预期的并行排序的数量 >)。预期的并行排序的数量可以粗略地推算为并行查询数量的两倍。当你不希望一个已测量的大区间小于数据文件的容量时，就会出现上述情况，对于表空间的一般用法（永久）来说，这通常是一个好的建议，稍后你会在第 19 章中看到。还有，因为你不要任何需要多个区间的排序，所以设置 INITIAL 和 NEXT 等于 SORT_AREA_SIZE 的倍数与用于该区间头的系统开销的数据块之和。同时，由于并行存取的随机性，你可以把几个排序存储在相同的区间内。因为你不想出现任何意外的事情，例如越来越大的 NEXT 区间，所以把 PCTINCREASE 设置为 0。此外，还因为并发性再次充当了一个要素。除了实际的度量尺寸技巧外，采用相等尺寸的区间是一个合理的方法。它可以同随机尺寸需求在一起很好地工作（没有一个排序需求远离平均值）。

在 SGA 中，一个被称为排序区间池（SEP）的内存结构包含组成属于临时表空间的单排序段的区域。当一个进程申请排序空间时，该池提供将要重复使用的空闲区间（空闲区间是指那些已经由一个较早前运行的进程分配和使用，目前空闲但没有被解除分配的区间），这与重复使用数据库缓存中的缓冲区的功能很像。此外，V\$SORT_SEGMENT 包含有关用户数量、区间和使用临时排序段的数据块方面的信息。你可以使用此信息来确定效率（命中）以及正确地度量你的区间。

提示 记住要重新定义用户的缺省表空间。使用 ALTER USER 语法或企业管理器完成这项工作。TEMP 传统上是主要临时表空间并且经常用来存储永久（处理）对象。然而，可以创建你需要数量的临时表空间（例如 3 或 4 个）并根据不同的处理要求把它们分配给不同的用户组。这有助于消除用户之间和 Oracle 本身的争用。例如，保存用于 Oracle 的 TEMP，然后创建并指定 TEMP2、TEMP3 等等，使它们属于不同的用户或进程组。

Oracle 提供了绕过数据库缓冲区缓存进行排序的能力。这被叫做排序直接写入（sort direct write）。当然，你仍然需要 SORT_AREA_SIZE 个字节，但是每一个排序操作可以有它自己的存储区并把它们直接写入到磁盘中。缓冲区的数量用 init.ora 参数 SORT_WRITE_BUFFERS（2~8）来设置，缓冲区的大小由 SORT_WRITE_BUFFERS_SIZE（32~64KB）设置。一个普通（串行）排序操作需要大小为（SORT_WRITE_BUFFERS × SORT_WRITE_BUFFERS_SIZE）+ SORT_AREA_SIZE 的排序直接写缓冲区。

对于 PQO 来说，每一个（并行）排序需要：

$$((\text{SORT_WRITE_BUFFERS} \times \text{SORT_WRITE_BUFFERS_SIZE}) + \text{SORT_AREA_SIZE}) \times 2 \times (\text{并行度})$$

init.ora 参数 SORT_DIRECT_WRITES 确定是否使用数据库缓冲区缓存的排序操作。如果设置为 AUTO（即系统的缺省设置），如果 SORT_AREA_SIZE 不小于 10 × 排序直接写缓冲区，则使用排序直接写缓冲区。如果设置为 FALSE，排序写入在被写回磁盘以前在数据库缓冲区缓存中得到缓冲。这是普通的排序缓冲区写。假如设置为 TRUE，排序写就总是排序直接写。VLDB、DSS 和数据仓库通常将该参数设置为 TRUE（或至少设置为缺省值 AUTO）。

18.4.3 其他排序微调参数

如下是影响排序性能的其他微调参数：

■ SORT_READ_FAC

■ SORT_SPACEMAP_SIZE

SORT_READ_FAC是一个比率，它代表了读取一个 Oracle数据块的时间量除以块传送率。该参数必须小于或等于DB_FILE_MULTIBLOCK_READ_COUNT。设置它的公式是：

$(\text{平均寻找时间} + \text{平均延迟时间} + \text{块传送时间}) / (\text{块传送率})$ 。

此参数考虑到操作系统和硬件（磁盘）设备在排序读取上的特性。然而，如果你使用具有混合存取时间（存取时间 = 寻找时间 + 延迟时间）的混合磁盘怎么办？采用最小公分母方法并把它设置为最慢磁盘的特性吗？

警告 除非你有一个非常相似的硬件（磁盘）配置以及足够的硬件经验，否则我建议你不要设置该参数。假如进行了不正确的设置，危害远大于帮助。

SORT_SPACEMAP_SIZE表示排序空间映射（sort space map）的字节大小，排序空间映射是多个排序运行地址中的每一个排序（每一个环境）一个映射。设置该值的公式为（总排序字节/SORT_AREA_SIZE）+64。总排序字节（total sort byte）要求你知道被排序的列乘以行数的字节大小。你可以使用整个表大小，但这是一个过高的估计（不一定是坏事情）。

按如下语句查询V\$SYSSTAT视图：

```
SQL> SELECT M.VALUE / D.VALUE "Memory Sort Ratio"  
2> FROM V$SYSSTAT M, V$SYSSTAT D  
3> WHERE M.NAME = 'sorts (memory)'  
4> AND D.NAME = 'sorts (disk)';
```

从report.txt的Statistics部分中，用'sort (memory)'的总数除以'sort (disk)'的总数来计算Memory Sort Ratio。

假如计算Memory Sort Ratio的两种方法都产生低于0.95的比率，你需要按照不足率来增加SORT_AREA_SIZE（和SORT_AREA_RETAINED_SIZE），其中不足率用0.95减去Memory Sort Ratio来计算。例如，假如你的Memory Sort Ratio是0.78，那么不足率是0.95 - 0.78 = 0.17。这表明你需要把你的排序参数至少增加17%。

18.5 调整多线程服务器

本节不对多线程服务器（MTS）的体系结构进行详细介绍，而是提供一系列简明的解释与准则帮助你调整你的MTS配置，特别是内存性能的优化。回忆一下 MTS把会话用户内存、用户全局区（UGA）重新定位到SGA共享池，该共享池被许多会话共享。通常，当你想要增加一个数据库系统的吞吐量（例如，每秒钟的事务量），尤其是在系统重负载（并行）时，你就要使用多线程服务器。换句话说，多线程服务器提供一个具有至少 200个以上的并行用户的OLTP系统使用。

警告 假如你计划把多线程服务器用于一个负载并不很重的系统的话，使用多线程服务器所花费的系统开销会超过它所带来的好处。也就是说，对于较轻负载的系统来说，多线程服务器实际上会损害系统性能。所以建议把多线程服务器主要用于负载较重的OLTP系统。

提示 将MTS_SERVERS设置为1/100（并行事务的数量），其中并行事务的数量等于并行用户的数量乘上每个用户所拥有的事务的数量。将MTS_MAX_SERVERS设置为1/10（MTS_SERVERS）。逐渐增加它们的设置值并进行测试。类似地，设置 MTS_DISPATCHERS=MTS_SERVERS 以及 MTS_MAX_DISPATCHERS=MTS_MAX_SERVERS。

要对共享服务器进行监控与调整，使用如下语句：

```
SQL> SELECT BUSY / (BUSY+IDLE) "Shared Servers Busy"
2> FROM V$SHARED_SERVER;
```

```
SQL> SELECT SERVERS_HIGHWATER
2> FROM V$MTS;
```

对于调度程序：

```
SQL> SELECT BUSY / (BUSY+IDLE) "Dispatchers Busy"
2> FROM V$DISPATCHER;
```

```
SQL> SELECT SUM(WAIT)/SUM(TOTALQ) "Dispatcher Waits"
2> FROM V$QUEUE
3> WHERE TYPE = 'DISPATCHER';
```

```
SQL> SELECT COUNT(*) "Number of Dispatchers"
2> FROM V$DISPATCHER;
```

假如 Shared Servers Busy 大于 0.50，或者 SERVERS_HIGHWATER 接近于或等于 MTS_MAX_SERVERS 的话，那么增加 MTS_MAX_SERVERS 的值。假如 Dispatchers Busy 大于 0.50，Dispatcher Waits 大于 0，或者 Number of Dispatchers 接近于或等于 MTS_MAX_DISPATCHERS 的话，那么增加 MTS_MAX_DISPATCHERS 的值。通常，如果你增加了一个的话，你或许也要增加另一个。

18.6 调整锁

和上一节一样，本节不会深入地讨论软件工程与 Oracle 锁机制的功能细节问题（参见第 25 章有关 Oracle 锁的背景知识）。你当然会看一些简要的背景知识与有用的调整建议。Oracle 中的锁，就和任何 RDBMS 一样，是基于内存结构的。和门一样，它由两个基本的逻辑结构组成：一个门(gate)和一个队列。假如仅有门的话，那么就有一个门。假如有一个门和一个队列以及一个排队进程的话，就有了一个锁。

调整的主要目标是消除不必要的等待和死锁，以及在关键处理期间不会用完锁。Oracle 锁的粒度是行级。也就是说，对于 INSERT、UPDATE 和 DELETE 语句，缺省情况下，锁被保持在行级。除非在代码中明确地指定，否则缺省情况下 SELECT 语句（查询）不持有锁。DDL 也持有锁，但是其性能调整能力是有限的；所以把注意力转向调整 DML 锁的性能这个主要问题上。

注意 直到事务提交或回滚，事务才释放锁。

对锁定的不良应用编码带来的直接后果是锁会引起不必要的等待。以不必要的高粒度对锁进行编码会引起不必要的锁等待——例如，当你仅需要一行时却锁定整张表。假如这个代码包含了不必要的长事务，会引起过度的锁等待。这方面的例子是在整个事务期内持久地保留客户/服务器连接。这种类型的编码经常可以被分为两个或多个较短持续时间的事务。最后，假如代码没有以正常方式被提交（或回滚），锁仍然被不适当地保持着，也会引起不必要的等

待。当编写代码时，应保证下列各项：

- 仅在必要时持有锁。
- 在尽可能低的层次上持有锁。
- 保持事务尽可能短小。
- 尽可能频繁地提交（或回滚）。
- 可能的话，取消客户/服务器事务。

这是调整锁与调整应用相互重叠的地方。当编写代码时，保证仅在必要时持有锁，在尽可能低的层次上持有锁，保持事务尽可能地短小，尽可能频繁地提交（或回滚）并且在可能的情况下取消客户/服务器事务。

提示 在你的所有外键上创建索引以消除不必要的父子锁定。

要监控锁定，作为SYS运行\$ORACLE_HOME/rdbms<version>/admin/catblock.sql以创建锁定视图。然后你可以从DBA_OBJECTS、DBA_WAITERS、DBA_BLOCKERS、V\$SESSION和V\$LOCK表中收集会话等待信息。你还可以运行 \$ORACLE_HOME/rdbms<version>/admin/utllockt.sql脚本以得到这些信息中的大部分。检查 System wide wait events中的Count for Event Name='enqueue'。假如其中的任何一个信息源提示锁定比必要的高，那么就应用以前的准则。假如必要，使用 ALTER SESSION KILL 语法删除会话。假如发生了死锁，则一个跟踪文件被转储。检查该跟踪文件以确定哪一个会话以及什么类型的锁引起了死锁。然后消除不必要的锁定，必要时对你的处理重新排序。

假如需要增加锁的数量以减少锁等待或者因为你已经到达了缺省值的上限，你可能需要增加ENQUEUE_RESOURCES。ENQUEUE_RESOURCES是DML_LOCKS、DDL_LOCKS、其他参数和平台特性的函数。你可以 ALTER TABLE DISABLE LOCKS以加速某个专用运行，但是我一般不推荐这样做。还有，你可以设置 DML_LOCKS=0来加速一个完整的实例，但是这或许产生不可预料的负面效应。很显然，在不使用重量锁的情况下，Oracle仍然能够以某种方式管理并行性，但是这种机制常发生完整性问题；因此，我不推荐它。

提示 设置DML_LOCKS=（并行用户的最大数）×（表的数量）。你还可以明确地设置ENQUEUE_RESOURCES，但是这样会清除DML_LOCKS。例如，假如你有U个用户和T个表，那么应设置DML_LOCKS=（U×T），还可以加上一些裕量，例如10%。

18.7 再看操作系统集成

不必讨论太具体的操作系统平台，只需简要地看一些与 Oracle中的调整内存有关的常见操作系统内存问题。平台的细节可以在附录 A和附录B中发现。主要的操作系统集成内存问题如下：

- 共享内存。
- 信号量。
- 进程间通信。
- 虚拟内存。
- 内存文件系统。

共享内存（shared memory）是由所有RDBMS厂商，尤其是在UNIX系统上的RDBMS厂

商虚拟使用的一种机制。共享内存支持多线程以及进程之间的内存共享。Oracle把后一种方法用于SGA，在该实例的所有会话之间共享内存。多线程服务器还依赖此资源以便在进程间模拟多线程。

警告 Oracle SGA应很好地适应由操作系统给定的共享内存。否则，会有不必要的内存分页与交换，这可能会削弱一个系统。

信号量（semaphore）是真正的锁定机制。Oracle把它们用作诸如DML锁这样的排队资源的基础。再有，它们由一个（全局内存）门和队列以及一组队列操作组成。Oracle的锁定操作映射到操作系统的低级信号量操作。

进程间通信（interprocess communication）是指允许进程之间互相通信的本地操作系统通信协议。它们能够用网络套接字、流、命名管道或其他机制来实现。因为Oracle不是完全多线程的，所以它大量依赖于操作系统的进程间操作。Oracle的会话间通信被映射到这些低级的操作。当使用SQL*Net时，IPC协议参考并使用缺省的操作系统进程间通信方法。

虚拟内存（virtual memory）是一种特殊类型的高速缓存。它是对磁盘实内存的扩展。正如同所有高速缓存一样，虚拟内存的命中率可以被计算出来。虚拟内存必须很充足，以满足操作系统本身和包括Oracle的所有的应用对总内存空间的需要。当实内存被分页或交换时，它被送给虚拟内存（磁盘）。支持虚拟内存的磁盘叫做备份存储（backing store）。还有，不希望Oracle SGA在那里映射或交换，也不希望任何主要的Oracle后台进程被交换出来。

内存文件系统（memory file system）是完整保存在实内存或虚拟内存中的文件系统。有时它们也叫做RAM盘（在DOS中）或临时文件系统（在UNIX中）。总之，它们的工作性能优于普通的磁盘文件系统一定的数量级。因为它们至少可以部分保存在内存中并利用虚拟内存高速缓存技术，所以大部分操作（至少一部分操作）来自内存或送到内存。Oracle有时把它们作为外来的性能调整解决方法使用。例如，你可以在一个操作系统的内存文件系统中创建一个Oracle临时表空间（或永久表空间）。这些表空间也可以用于存储查询表等用途。

第19章 调整输入/输出

本章要点：

调整表空间与数据文件

调整块和区域

调整回滚段

调整重做日志

Oracle 8的新输入/输出特性

通常，调整输入/输出被认为是物理设计的扩展与改进。当你刚开始做物理设计时，需要完成大量的定量估计工作。如果你不调整输入/输出性能就无法接受的话，那么你就需要在应用程序运行到峰值时刻对它进行实际测量。在 Oracle中调整输入/输出主要包括调整构成数据库的段（表和索引）的基本物理结构。它们包括表空间和数据文件。表空间由区间组成，进而由块组成。数据文件是支持 Oracle物理结构的操作系统（OS）实体。它们和其他的 Oracle结构在第5章中已经讨论过了，因此本章不会过深地涉及它们的定义和函数，只是顺便再强调一些有关的概念。

输入/输出意指读和写。在数据库术语中，特别是对 DML而言，SELECT操作是读，而 INSERT、UPDATE和DELETE操作是写。DDL（CREATE、ALTER或者是DROP）总是写操作。因此，从Oracle结构中读和向Oracle结构中写都被认为是输入/输出问题。例如，发出一条SELECT语句将产生从一个或多个索引/表中读的操作。它也产生一些最小的重做日志信息。发出INSERT、UPDATE或者DELETE就产生对一个或多个索引/表、回滚数据和重做日志信息的读和写操作。这就得出一个非常微妙的一般结论：读就仅仅是读，而写却是读加写。如何把已经读到数据缓冲区的信息写到块中呢？只能使用 SQL* Loader的直接路径功能。

因此，在Oracle中考虑调整输入/输出时，不仅要考虑调整表空间、区域、块和数据文件，同时还要考虑调整回滚段和重做日志（因为用户 DML会产生各种各样的输入/输出）。因此，下面这部分涵盖了各种输入/输出：表空间和数据文件、区域和块、回滚段和重做日志。同前面的调整内存和应用程序的章节类似，你会遇到调整输入/输出与调整内存以及调整输入/输出与调整应用程序相交叉的情况。

记住，任何输入/输出操作都需要在采取进一步的行动前把 Oracle数据块读入数据库高速缓存中。这就是一个调整内存和调整输入/输出相交叉的例子。考虑一个不同的观点：假设你有一个OLTP应用软件，它仅从许多表中的几个表中大量地读和写，而那些表要被放到单独的表空间，甚至是单独的磁盘中。这就是一个调整应用程序和调整输入/输出相交叉的例子。用高层次的眼光看调整表空间和数据文件是开始学习的好起点。

19.1 调整表空间与数据文件

正如你在第5章中所了解的，表空间是 Oracle用于物理存储的结构。表空间存储段的集合：表和索引。一个表空间在操作系统级映射到一个或多个数据文件上去。你在第3章和第16章中

学习了应用程序类型的概念以及它如何影响到你的物理设计的问题。复习一下，为了有一个适当的物理布局，表空间（和它们的数据文件）应尽可能地分散在不同的磁盘上。把表空间分散在不同的盘上能够消除或者至少能够减少磁盘争用。

磁盘争用在多用户或程序试图同时访问同一张磁盘时发生。例如，你有两个必须经常连结在一起的表，例如 DEPARTMENTS 和 EMPLOYEES 表，它们的表空间通常要被分散在两张不同的磁盘上，因为要存取相同磁盘上的表或者索引，所以会导致对同一资源的争用。理想情况就是这四个段放在四张不同磁盘上的四个表空间里。另一种方法是和聚簇那样，把这些段放在相同的磁盘上。你可以研究 Oracle 版的聚簇技术，在第 10 章中被称为外来的解决方案。

19.1.1 分区表空间

和你在第 3 章和第 10 章学习的一样，希望使你的 Oracle 物理布局为如下形式：

- 系统（SYSTEM）放在一张单独的盘上。
- TEMP 至少放在一张单独的盘上。
- 数据 1（DATA1）到数据 N（DATAN）放在 N 张单独的盘上。
- 索引 1（INDEX1）到索引 N（INDEXN）放在 N 张单独的盘上。
- ROLLBACK 至少放在一张单独的盘上。
- redo log 1 到 redo log N 放在 N 张单独的盘上。

记住，在最初的创建中，用户的缺省表空间和缺省的临时表空间指向的是 SYSTEM！如果它们没有被正确创建的话可以通过执行下面的语句加以改变：

```
SQL> ALTER USER <user>  
2> DEFAULT TABLESPACE <tablespace>  
3> TEMPORARY TABLESPACE TEMP;
```

为了要确定 DATA（或者 INDEX）表空间能否共存，你需要按表的活动性级别对它们进行分类，另外还要测定应用程序的类型。

例如，假定你有一个 DSS 应用程序，该应用程序在通常的操作中所有的表都是只读的（它们被大批装载时除外）。那么这是否就意味着你可以把所有的表放在一张盘上呢？不一定！为便于说明，考虑一下从 40 个表中读出 10 个表的情况。这 10 个表中的 7 个是可以并行存取的，其中的 4 个几乎总是连结在一起使用。理想情况是你至少要用 9 张盘使这些数据独立。

我是如何得出这个数字的？需要 7 张单独的盘，因为 7 个表是并行存取的，包括那 4 个经常连接在一起的表。你至少再需要一张盘用于剩下的 3（10-7）个不能并行存取的表，同时你还至少再需要一张盘用于剩下的 30（40-10）个表（如果一张盘能放下所有那 30 个表的话）。总之，跟应用程序分类一样，这是活动性的分类：把表按存取的频率分类。如果一个表非常活跃，那么就说这个表是热的（hot）。同样一个表中的活动的列也被称为是热的（hot）。一个温暖的（warm）表或列是指相对于 hot 成为来讲不太活动的表或列。那么，寒冷（cold）的表就是指那些很少进行存取的。另外，还有低/中/高活动性命名法。总之，可以按下述方法给表分类：

H：高活动性，或者热的。

M：中等活动性，或者温暖的。

L：低活动性，或者寒冷的。

尽管只是一个简单的分类，但它对物理设计和性能调整有很大的帮助。例如，高 DML 活动性表会形成高碎片，如果把它们保存在本身单独的表空间或磁盘上，那么就可以避免较低活动性表的不必要碎片。当分类工作结束后，遵循下面简单的方针给表空间分区以及给表 / 索引定位：

- 把每个热表/索引放在它们本身单独的表空间/磁盘上。
- 把每个温暖表/索引放在它们本身单独的表空间/磁盘上。
- 把多组寒冷表/索引放在它们本身单独的表空间/磁盘上。
- 把已连结的表/索引放在它们本身单独的表空间/磁盘上。
- 保持数据和索引表空间独立。
- 把最热的表空间放在最快的盘上，最寒冷的放在最慢的盘上。
- 由于磁盘空间有限，如果需要的话，可以把温暖的表 / 索引和寒冷的表 / 索引放在相同的表空间/磁盘上，但这就需要考虑并行存取的问题（例如，可能不想把一个温暖的交互式存取的表同一个寒冷的、处理时间长达几个小时的批量存取的表放在同一个表空间上）。
- 如果不并行存取，把“相似”的表 / 索引放在相同的表空间/磁盘上。
- 前面方针的扩展：由于磁盘空间有限，如果需要的话，把热的、温暖的和寒冷的表位置交叉存放。换句话说，就是要了解你的存取模式、使用类型及所有表的并行级别。有了这些知识，如果磁盘空间紧张的话，你就可以把两个不在一起并行存取的热表放在相同的表空间 / 磁盘上，把不同表空间上的热索引和寒冷表放在一个同一张盘上，或者把所有放在六个不同的表空间中的三种类型的表和索引放在同一张盘上。建议：除非绝对需要否则不要这样做。这是对低磁盘空间上的数据库系统所做的最后的努力。

聚簇提供了一种可供选择的存储方法用于存储频繁连结的表。

19.1.2 聚簇

簇（cluster）是一种特殊类型的表空间，在这里能够物理嵌套父 - 子或者主 - 从层次关系。例如，如果一个雇员只能在一个部门工作，这就是一个从 DEPARTMENTS 表到 EMPLOYEES 表的一对多逻辑关系。EMPLOYEES 表有一个外键 DEPTID，对应着 DEPARTMENTS 表的主键 DEPTID。你可以像这样建立一个簇：

```
SQL> CREATE CLUSTER DEPTS_EMPS (DEPTID NUMBER (9))
2> SIZE 256
3> TABLESPACE DATAn
4> STORAGE (...);

SQL> CREATE TABLE DEPARTMENTS
2> (DEPTID NUMBER(9) PRIMARY KEY, ...)
3> CLUSTER DEPTS_EMPS (DEPTID);

SQL> CREATE TABLE EMPLOYEES
2> (EMPID NUMBER (9) PRIMARY KEY, ...
3> DEPTID NUMBER (9) REFERENCES DEPARTMENTS)
4> CLUSTER DEPTS_EMPS (DEPTID);
```

可选择的 SIZE 变量指定簇键（这个例子中是 DEPTID）及所有与它相关的行预计要用的字

节数。表19-1是DEPTS_EMPS簇如何被物理地组织的一个文本表示法。

表19-1 DEPTS_EMPS簇

DEPARTMENTS表（聚簇前）：

DEPTID	DEPTNAME	etc.
1	PERSONNEL	
2	ACCOUNTING	

EMPLOYEES表（聚簇前）：

EMPID	EMPNAME	DEPTID	etc.
1	William Day	1	
2	James Hutch	1	
3	Ely Jones	1	
4	Peter Page	2	
5	Tom Edwards	2	

聚簇后的DEPARTMENTS和EMPLOYEES表：

DEPTID	DEPTNAME	etc.
1	EMPID	EMPNAME
	PERSONNEL	etc.
	1	William Day
2	2	James Hutch
	3	Ely Jones
	ACCOUNTING	
	4	Peter Page
	5	Tom Edwards

“etc.”是DEPARTMENTS表和EMPLOYEES表中的其他非键列。你可以看到，EMPLOYEES表被物理嵌套在DEPARTMENTS表中，这样无论什么时候这两个表都会通过DEPTID的连结一同存取，数据已经用那种方式组织好了，并且比普通情况下没有聚簇的两个表更易使用。在很多情况中，特别是在那些包括实际的范围查找（限制的或不受限制的）的情况下使用常规索引（B树结构）。但是，如果你的应用程序查询几乎总是点查询（精确的匹配、等式的比较）的话，你也许应当使用哈希索引。

哈希索引就是把列值作为输入，然后使用一个专用的内部哈希函数计算该行的物理地址（实际是ROWID）作为输出。如果列是NUMBER数据类型，它均匀分布而且不是组合的，那么你就可以使用它而不是内部哈希函数来创建哈希索引。例如，要用哈希索引重新创建DEPTS-EMPS簇，句法如下所示：

```
SQL> CREATE CLUSTER DEPTS_EMPS (DEPTID NUMBER (9))
2> SIZE 256
3> HASH IS DEPTID HASHKEYS 29
3> TABLESPACE DATAn
4> STORAGE (...);
```

HASH IS 选项告诉Oracle用DEPTID列代替内部哈希函数，HASHKEYS参数说明要创建多少哈希存储段来保存哈希簇键索引（DEPTID）的输出值。这个参数要与不同的簇键值的数量相等，该值调整为下一个最大的质数。在这个例子中，假设你有25个部门（DEPTID），应选择下一个最大的质数29。要了解更多的关于如何管理哈希簇和索引以及测定它们大小的内

容, 请参考 Oracle 服务器管理者指南。

19.1.3 监控

你已经看到了准则和聚簇能帮助减少对热表空间的争用, 但是怎么才能区分出哪个表空间是热的呢? 你在第 3 章中学到, 如果从早期建模说明书中得出好的定量估价, 也就是好的事务分析数字, 那么你就会有一个开头并且能用那些数字指导你最初的物理设计。

比如, 如果 20 个表中的 3 个平均每秒产生 150 个事务 (tps), 而剩下的表产生的事务数每秒还不到 30 个, 那么你就可以说那 3 个表相对于其他表是热表, 因此应该和剩下的表以及它们的索引隔离出来。但在布置阶段之后, 你需要监控那些实际的、低级别输入 / 输出数字, 例如每秒物理读写的次数。

用普通的两个 Oracle 备用工具帮助做监控: V\$ 动态性能视图和 utlbstat.sql/utlestat.sql 适当地运行得到的 report.txt。这些工具在第 18 章中解释了。你也可以用 OEM/PP 检查文件的输入 / 输出。你需要检查 V\$DATAFILE 和 V\$FILESTAT 视图:

```
SQL> SELECT NAME, PHYSRDS, PHYSWRTS  
2> FROM V$DATAFILE DF, V$FILESTAT FS  
3> WHERE DF.FILE# = FS.FILE#;
```

还需要从 report.txt 的输入 / 输出部分检查 PHYS_READS 和 PHYS_WRITES。物理读和物理写的总数就是那个文件或表空间输入 / 输出的总数。考虑一下每张磁盘中所有表空间的文件总数。如果用 V\$ 视图的方法, 选择信息两次, 组成你的统计集合: 在你的应用达到最大容量后进行一次, 在应用的负载刚要减少前再进行一次。这就模仿了 utlbstat.sql/utlestat.sql 的方法。使用任何一种方法, 按磁盘统计开始和结束输入 / 输出总数。从结束时的输入 / 输出中减去开始时的输入 / 输出。从统计-集合运行的结束时间减去开始时间。把时间数字转换成秒。实现的输入 / 输出除以在开始和结束之间占用的时间。

这有个例子: 3 号盘包含文件 6 和文件 7, 开始数字是在上午 11:00 有 1200 个物理读和 400 个物理写。结束数字是在上午 11:20 有 31000 个物理读、17000 个物理写。这等同于:

```
((31000-1200) + (17000-400)) / (11:20-11:00)=  
(29800+16600) / (20m)=  
(46400) / (1200s)=  
38.67 I/O 数/s
```

这表明这张盘上每秒有不到 40 个输入 / 输出, 这比较合理, 因为对大多数现代磁盘来说, 这被普遍认为是一个饱和点。然而, 如果所有其他的磁盘每秒的输入 / 输出都大大地低于 40, 那么仍然希望卸载这张盘以平衡输入 / 输出。不仅希望减少孤立盘的数量, 而且还希望尽可能地平衡输入 / 输出——实际上, 这个原则应延伸到所有的盘。

准则: 如果任何给定的盘每秒的输入 / 输出接近或超过 40, 你可能需要把一些负载卸载到其他盘上去。最好的情况是移动表, 比较坏的情况是移动表空间和它们相应的数据文件。之后, 你需要考虑交叉存取、聚簇、数据条, 或者如果需要的话买更多的盘。你的目标是平均所有盘的每秒输入 / 输出以便它们都是 40 或者更少。如果给定你的应用程序的最大负载需要, 那么尽可能地使所有磁盘的每秒输入 / 输出数相等以便所有磁盘负载相同。

19.2 调整块与区间

Oracle 块由区间进行组织, 区间包含表空间。它们是表空间存储的物理基础。因此, 对它

们的存取和增长管理的效率越高，性能也就越好。

19.2.1 使用预分配值

从第16章中，你学到了动态分配会导致太多的额外开销，并损害输入/输出性能。静态的预分配几乎在所有情况中都是首选的。你可以静态地预分配一个段（表或索引）或者一个表空间。通常，你选择一种方法或者另一种方法。如果你预分配表空间，你就应该有一个关于表大小以及它们如何映射到区间上的好设想。如果你把表空间设为缺省值，之后预分配表，那么你在把不同区间的表混合在同一个表空间时就要有一定的灵活性。但是，记住你不要混合太多不同区间大小的表（通常情况下不超过三个），因为这实际上产生了碎片，而这是所不希望的。

不必回顾第5章和第21章中给出的所有的存储术语和参数，只需简要考虑一下同一个表空间存储两个表的两种不同方法。假设你有两个表，每个表最大需要 100MB空间。创建一个 256MB的表空间以为一些额外的增长留出空间，该表空间只有一个数据文件。第一种方法——预分配表空间，使用如下的语句：

```
SQL> CREATE TABLESPACE TS1
      2> DATAFILE '/data1/file1.dat' SIZE 256M
      3> DEFAULT STORAGE (INITIAL 100M NEXT 100M
      4> MINEXTENTS1);

SQL> CREATE TABLE T1 (a number(9), ..., z number(9))
      2> TABLESPACE TS1;

SQL> CREATE TABLE T2 (a number(9), ..., z number(9))
      2> TABLESPACE TS1;
```

第二种方法——预分配表，使用如下语句：

```
SQL> CREATE TABLESPACE TS1
      2> DATAFILE '/data1/file1.dat' SIZE 256M;

SQL> CREATE TABLE T1 (a number(9), ..., z number(9))
      2> TABLESPACE TS1
      3> STORAGE (INITIAL 100M NEXT 10M
      4> MINEXTENTS 1);

SQL> CREATE TABLE T2 (a number(9), ..., z number(9))
      2> TABLESPACE TS1
      3> STORAGE (INITIAL 100M NEXT 10M
      4> MINEXTENTS 1);
```

预分配表不仅在增长方面上，而且在与这个表空间相关的性能方面都会提供好的粒状控制。如果有一个表在你只分配10MB的时候只是偶尔扩展，那么为什么要分配100MB呢？通常，预分配的存储单元越出色，性能就越好，这其中的许多原因在第10章中已经讨论过了。

19.2.2 使用Oracle条

你在第3章关于物理设计与RAID的讨论以前就已经遇到了条。这一节告诉你如何做手工的条，手工条也称为Oracle条。

Oracle条本质上就是区间预分配的一种形式，每个区间处理（几乎）所有与之相对应的数据文件，这些数据文件都被方便地存放在一张单独的磁盘上。当然，缺点是你应当知道长期的最大增长是多少，或者你至少应该知道中期的最大大小是多少。然而，这有助于并行执行许多高活动性表空间的输入/输出操作。假设你有一个活动性很高的表要加上条。它的最大大

小不到600MB，你有3张磁盘可用于给它加条。语法类似下面这样：

```
SQL> CREATE TABLESPACE TS1
2> DATAFILE '/data1/file1.dat' SIZE 200M,
3> DATAFILE '/data2/file2.dat' SIZE 200M,
4> DATAFILE '/data3/file3.dat' SIZE 200M;

SQL> CREATE TABLE T1 (a varchar2(25), ..., z varchar2(25))
2> TABLESPACE TS1
3> STORAGE (INITIAL 198M NEXT 198M
4> MINEXTENTS 3 PCTINCREASE 0);
```

为了替换缺省设置值（50），把PCTINCREASE设置为0是必要的。现在你已经通过人工预分配必要的（最大）区间大小的方法，在3张不同的盘上存储了一张表，该必要的区间大小略微小于（通常为1%）相应数据文件的长度。这个表已经在盘上被有效地加上条了。条单元是一个大小为198MB的Oracle区间。同RAID条相比，这是一个非常大的条单元，因为RAID条单元通常以KB的倍数或者更小的单位来度量。尽管如此，它仍然有效，并且不需要附加硬件或软件。另一方面，如果你已经错误地度量了表的大小，或者特别是如果表的增长-收缩行为反复无常的话，那么对这种方法的维护很快就会变得很难。不管别的，如果仍然需要条的话，那么这时你就希望把RAID作为Oracle条的替代品。

像我们在第3章和第4章中讨论过的一样，对于支持表或回滚表空间的数据文件来说，最好选择RAID 3或RAID 5。RAID 0对索引是个好选择。RAID 1对重做日志是个好选择。就象刚提到的，RAID可以用来代替Oracle条，但它也可以和Oracle条一起使用。例如，在前面的通过数据文件给表加条的例子中，每一个数据文件都有可能在单个的RAID卷（磁盘集）中加上条，而不是简单地存储在一个单独的（非RAID）磁盘中。这种方法能够提供真正的性能收益，但是需要数据库管理员综合持久地管理，因此要聪明地选择你的策略。记住，尽管RAID 1、3和5能够容忍单个磁盘的损失，但RAID不能替代一个好的备份系统。

19.2.3 避免碎片

调整的一个主要目标是避免不必要的碎片。一种方法是通过估计或测量以及正确的区间预分配进行正确地估价。同时还要避免不同的活动性级别、大小以及已知的区间需求的表在相同表空间中混合。当一个表空间中的区间被分配和解除分配时会引起区间碎片。通过自由空间碎片或者简单的表碎片两种途径形成区间碎片。自由区间是指那些从来没被分配过或者在段被删除后释放的区间。当一些单独的自由区间块遍布整个表空间时，这就被称为瑞士干酪（Swiss cheese）碎片或者冒泡（bubbling）。当在表空间的运行中存在一些连续的自由区间块时，这就是通常所说的蜂窝（honeycomb）碎片。在任何一种情况下，都有自由空间碎片。当一个表动态地伸长超过它原来创建的区域时，这就被称作表碎片。

尽管碎片通常主要被认为是空间问题，但做在一个数据库系统中的所有事情一样，它会对性能有影响。无论直接或非直接地，区间碎片会引起动态扩展。从性能来看，这种扩展是你所不希望发生的。如果在一个表空间的生命期中，段创建和撤消引起了瑞士干酪的效果，就会由于强行从多个区间中输入/输出而损伤性能。一般来说，这通常也不是特别糟。然而，在最坏的情况中，它会严重降低系统的性能。例如，假设你的PQO是基于准确的数据排列而设置，用来加速对一个大的热表的全表（连续的）搜索。过段时间，那个表就分裂成区间碎片。因为PQO结构原来是基于负载-平衡区间而设置的，所以它变得越来越慢。另外，原来对

连续磁盘单元的一组顺序读已经变成了对不同的（随机的）磁盘单元的群体读，这需要更高的活动性！

用下面的方法帮助确定你的表碎片的程度：

```
SQL> SELECT SEGMENT_NAME, EXTENTS
2> FROM DBA_EXTENTS
3> WHERE EXTENTS > 4
4> ORDER BY EXTENTS;
```

如果返回了任何段，那么就认为它们就太零碎了（除非你已经计划把这些区间作为 Oracle 条的一部分或者 VLDB 表的一部分）。要校正这个情况，做下面两件事中的一项：

- 创建第二个带新存储参数的表，从第一张表中选择。使引用的约束条件无效。删除第一张表。把第二张表重新命名为第一张表的名字。重新使引用的约束条件有效。重新建立所有索引和授予必要的特权。
- 带压缩选项导出表。使引用的约束条件无效。删除表。把导出的导入回来。重新建立所有索引和授予必要的特权。

用下面的方法帮助确定你的自由空间碎片的程度：

```
SQL> SELECT TABLESPACE_NAME, COUNT(TABLESPACE_NAME)
2> FROM DBA_FREE_SPACE
3> ORDER BY TABLESPACE_NAME
4> GROUP BY TABLESPACE_NAME
5> HAVING COUNT(TABLESPACE_NAME) > 10;
```

如果返回了任何表空间，就认为这些空间太零碎了。发出 ALTER TABLESPACE <表空间名> COALESCE 语句（对于 Oracle 7.3 和更高的版本）帮助校正它。然后重新运行查询。如果有些表空间仍然被返回并且碎片比较高（大于 5 倍的段数）的话，如果行得通，你也许希望对每个表按前面的准则做。删除所有的表。删除表空间。删除数据文件。重新创建表空间。重新创建所有的表。重新运行查询来进行一个用来与以后的增长做比较的基准测试。

提示 为了使 SMON 能够与一个表空间的自由空间结合，无论是自动的还是通过 ALTER 命令，表空间必须使它的 PCTINCREASE 大于 0。然而，除非这个表忽略 PCTINCREASE 的设置，否则，这会导致表空间内表的不希望的几何增长。建议：对表空间把 PCTINCREASE 设置成 1，对表设置成 0。

块碎片能够通过转移行或者通过链接行发生。转移行是指那些已经在一个块中被修改了，但是长度增加了以至于超过了原来块中剩余的自由空间的行。它们被重新分配了另一个块（不在自由表中），指针在原来的块中维护。链接行是那些被插入或者修改了，而长度不适合任何块的自由空间的行。它们用指针在块间被拆分。链接行可以通过适当的度量和设定 DB_BLOCK_SIZE 来避免。转移行可以通过启用足够的修改空间（100 - (PCTFREE + PCTSUED)）来避免。为了确定链接行或转移行的程度，运行下面的语句：

```
SQL> ANALYZE TABLE T1 COMPUTE STATISTICS;
SQL> SELECT TABLE_NAME, CHAIN_CNT
2> FROM DBA_TABLES
3> WHERE CHAIN_CNT > 0
4> ORDER BY CHAIN_CNT;
```

上述语句给出一个有链接行或转移行的表的排序清单。为了帮助补救转移行的情况，你

可以删除并重建表。如果表非常大，你可以运行 `utlchain.sql`来建立 `CHAINED_ROWS`表。之后运行下面的语句：

```
SQL> ANALYZE TABLE T1 LIST CHAINED ROWS;
```

转移行的 `ROWID`存储在 `CHAINED_ROWS`中。你可以用它从你的表中选择转移行并把它们放到一个临时表中。之后从原来的表中删除它们，再从临时表中把它们插回到原来的表中。

关于碎片性能问题的有关话题包括高水位标志、自由表和表 /索引的重新组织。高水位标志（HWM）是到现在为止那个段中已使用过的块的数目。删除操作不会降低 HWM。在全表扫描期间，Oracle必须读低于高水位标志的未被使用的块。尽管下面的方法没有为当前的段复位高水位标志，你仍然可以用它为其他段回收废弃的空间：

```
SQL> ALTER TABLE T1 DEALLOCATE UNUSED;
```

另外，`TRUNCATE TABLE <表名>`可以为表和所有的索引复位 HWM，这与 `DELETE FROM <表名>`形成对比。

自由列表（freelist）连接在自由块清单上，每个段中包含一个或更多的清单。在 `V$WAITSTAT`中或者使用 `report.txt`的 `System wide wait events`部分检查自由表的等待事件。如果任何自由表中的事件比0或者比你的基准数字大很多，就要考虑往热表中添加自由列表或者自由列表组。它们必须被重新建立。

如果在低于HWM的块中平均有很多自由空间，那么就要考虑重新建立你的表。在分析表并计算或者估计统计数字后，检查下面的语句：

```
SQL> SELECT TABLE_NAME, AVG_SPACE  
2> FROM DBA_TABLES  
3> WHERE AVG_SPACE > (.10 * BLOCKS)  
3> ORDER BY AVG_SPACE;
```

换句话说，如果你的表在低于HWM的块中有超过10%的自由空间，那么用导入/导出或者临时表的选择/重命名方法删除并重建该表。

频繁修改的表的索引应该能承受检查。它们经常需要重新创建。一条通用准则是如果索引的大小接近实际表大小的1/3，那么就删除并重建索引。另一条通用准则是如果索引的级别数超过3，那么该索引可能就太高了。分析你的索引并检查 `INDEX_STATS`表来搜集信息。如果需要的话，删除并重建想要的索引或者使用 `ALTER INDEX <索引名>REBUILD`语句。

在块级别中 `INITRANS`是个影响输入/输出性能的微调参数。`INITRANS`设置每个块中初始事务槽的数目。它应代表并行用户或存取块的程序的数目。这在表级设置。如果你的应用程序有很高的并行需求，把 `INITRANS`设置得相对高一些以避免过多地动态创建事务槽。这个字节系统开销很低，而性能很好。

19.3 调整回滚段

回滚段是或多或少的随机输入/输出单元。它们被并行地（由DBWR）写入，在数据库缓冲区缓存中得到缓冲，并保存在专门的表空间中。回滚段会在它们之间，以及它们与其他数据库输入/输出单元（例如数据表空间）之间产生争用。它们提供取消不提交事务的影响的能力。因此，它们通常被称为取消日志，或者叫做通用事务日志的取消部分。取消数据意指在内存的数据库缓冲区缓存中，或是在磁盘上的回滚段中一致性读的Oracle块。它们用于事务回滚、实例恢复以及共享的读能力。用循环的方式把它们分配给事务。

性能调整的主要目的是为了减少争用。换句话说，你希望等待回滚段的事务数最少，特别是在许多OLTP系统所具有的高并行环境中。回滚段的争用可以通过创建足够数量的适当大小的段来很好地避免。回滚段把事务表放在它们的头部中。争用相同回滚段的并行事务在争用事务表本身时被显示出来。你如何能发现这种争用呢？用下面的方法：

```
SQL> SELECT CLASS, COUNT
2> FROM V$WAITSTAT
3> WHERE CLASS LIKE '%undo%'
3> AND COUNT > 0;
```

你也可以使用这种方法：

```
SQL> SELECT USN, WAITS
2> FROM V$ROLLSTAT
3> WHERE WAITS > 0;
```

你也可以从report.txt的回滚部分检查UNDO_SEGMENT 中是否TRANS_TBL_WAITS>0。你也可以为 '%undo%'事件检查 report.txt的System wide wait events 部分，或者从视图V\$SYSTEM_EVENT中选择这些事件。还有，如果你遇到了 ORA-01555错误：“快照太旧”，表明你已经把回滚用完了。你或者需要更多的段、更大的段或者两者都要。

如果ORA-01555错误频繁地出现或者事务表等待事件比0大得多，这就表明争用。特别是，对非常大的数据库或者非常高的并行查询数据库来说（例如某些 DSS系统），如果来自V\$ROLLSTAT或者TRANS_TBL_WAITS/ TRANS_TBL_GETS二者中任何一个的WAITS/GETS的比值大于1%（0.01），就有可能存在回滚争用的问题。分配回滚段的数目等于并行用户和程序的数目除以4，最大值为50。

其他的回滚段调整建议如下。对于回滚段，设置 NEXT=INITIAL。对于回滚段，PCTINCREASE总是0。把MINEXTENTS设置为大于等于20，这样INITIAL*MINEXTENTS大约比表空间（数据文件）大小低一个百分点，以防止动态扩展在回滚段生命期开始时受并行增加的影响。设置 OPTIMAL=INITIAL*MINEXTENTS以防止回滚段不必要的减少。你可以通过 V\$TRANSACTION 视图计算事务产生的取消数目。在最高负载时观察 SUM(USED_UBLK)和MAX(USED_UBLK)，确定该时间间隔中的取消总数和最大值。几次采样后设置INITIAL为大于等于MAX(USED_UBLK)。为了进行微调，适当地设置 init.ora参数 PROCESSES、TRANSACTIONS和TRANSACTIONS_PER_ROLLBACK_SEGMENT。有关这些参数更详细的内容参考Oracle的服务器管理器指南。但是，这些参数对分配适当数目和大小的回滚段只有很小的影响。

小事务（通常在OLTP系统中出现）是指那些产生少量取消信息的事务。对于这种情况，按以前的准则，用50个回滚段在应用程序中尽可能频繁地进行提交。大事务（通常在批系统中出现）需要单个的大回滚段。因此，你不需要许多回滚段，只要一个大的即可。为了确保一个特别长时间运行的（大的）事务能够使用特别大的回滚段，采用下面的语句：

```
SQL> SET TRANSACTION USE ROLLBACK SEGMENT <rollback_segment>;
```

如果你有许多只能有限改变的小事务，比如每次只改变一个块中的几行或者插入一行，那么就把init.ora参数DISCRETE_TRANSACTIONS_ENABLED设置为TRUE。在用这个参数前，在Oracle的服务器管理器指南中了解使用事务的标准和先决条件。

19.4 调整重做日志

重做日志是连续的输入/输出部分。同时，它们一次只（被 LGWR）写到一处，被缓冲在 SGA（日志缓冲区）的一个单独部分，并作为操作系统文件存储。这样重做日志之间就没有争用了，只在重做日志和其他数据库输入/输出部分（例如数据表空间）之间有争用。与回滚段可用于额外用途相对，重做日志只用于实例恢复（前滚）。当下面的事情之一发生时 LGWR 过程写重做日志（实际上，执行一次日志刷新）：提交、DBWR数据刷新（检查点）、检查点发生超时（到达 LOG_CHECKPOINT_INTERVAL），或者日志缓冲区超过总数的 1/3。它们以循环方式分配给 LGWR，并且日志交换总是触发检查点。

性能调整的主要目的是为了使下面这些最小：

- 给定应用程序的检查点。
- 重做等待事件。
- 重做争用问题。

下面是配置检查点机制的建议：

- 1) 把不小于 2 个组中的不小于 2 个成员放在单独的输入/输出通道（磁盘和控制器）中，每个成员有相同的大小。
- 2) 在 init.ora 中把 CHECKPOINT_PROCESS 设成 TRUE，以便在从 LGWR 过程到 CKPT 的过程中同步卸载文件头。
- 3) 把 init.ora 参数 LOG_CHECKPOINT_TIMEOUT 设成 0。
- 4) 把 LOG_CHECKPOINT_INTERVAL 设为在操作系统块中的一个重做日志大小与一个操作系统块的大小之和。这儿有个例子：所有重做日志成员的大小都被设为 1MB，1MB=1024KB=2048 个操作系统块（每个操作系统块 512 字节）。因此，把 LOG_CHECKPOINT_INTERVAL 设为 2048+1（即 2049），同时已经把 LOG_CHECKPOINT_TIMEOUT 设成 0，就使检查点只发生在日志进行交换时，这通常可以提高性能。

警告 非常繁重的 OLTP 环境需要许多大的重做日志来支持这种设置。换句话说，你可能需要减少检查点的活动性，从几乎每分钟一次减少到每个日志交换事件一次，也许是每 15~30 分钟一次。

警告 如果使用 ARCHIVELOG 模式，确保你的最长运行事务（即产生最多重做信息的那个事务）不会消耗掉你所有可用的重做日志成员。否则，无论何时，只要重做交换超过了归档，你的数据库将死机！

应如何监控和调整重做等待事件以及重做争用问题？检查 report.txt 关于 'redo log space requests' 的 Statistic 部分和 report.txt 中关于 'log file space /switch' 的 System wide wait events 部分。这些事件也可以分别用 V\$SYSSTAT 和 V\$SYSTEM_EVENT 视图检查。

如果这些事件中的任何一个比 0 大得多，这就表明 LOG_BUFFER 长度被设得太小了。把它增加百分之五或者更多，让你的应用程序运行一段时间，再查询这些事件一次。重复这个过程直到这些事件达到或非常接近 0。

使用 report.txt 的 Statistic 部分中的 'redo size' Total，或者 V\$SYSSTAT 的值，确定如何适当地设置重做的大小。这个统计量是采样间隔中产生的重做信息的总数。

提示 减少在直接路径方式下使用 SQL*Loader 产生的不必要的重做信息的数量。如果你正在归档，还要指定 UNRECOVERABLE。如果你有 PQO，就要为你的并行表和索引的创建语句指定 UNRECOVERABLE。

像所有的闩（或者自旋锁）结构一样，重做拷贝闩是基于内存的机制，这种机制提供重做 LOG_BUFFER 的无队列锁定。实际上只有一个重做分配闩，一个进程在它写时可以独自占用它（描述为一个 WAIT 或者 WILLING_TO_WAIT 申请或状态）而其他的进程等待；进程可以释放它（描述为一个 NOWAIT 或 IMMEDIATE 申请或状态）并且在写时保持一个重做拷贝闩，这使另一个过程能够拥有这个重做分配闩。如何度量对这个资源的争用呢？用 V\$LATCH 视图并计算 NAME 类似 'redo%' 的 MISSES/GETS 的比率。同样，计算 report.txt 的两个闩部分中 MISSES/GETS 和 NOWAIT_MISSES/ NOWAIT_GETS 的比率。另外，查看 report.txt 中的 HIT_RATIO 和 NOWAIT_HIT_RATIO 的数字，它们是刚刚计算过的 MISSES/GETS 比率的倒数。

注意 如果任何一个 MISSES/GETS 比率比 1% (0.01) 大得多或者两个命中率中的任何一个比 99% (0.99) 小很多，那么就可能有太多的重做闩争用。要想减少闩争用，增加 init.ora 参数 LOG_SIMULTANEOUS_COPIES 的值，这个参数缺省情况下设置为 CPU 数目与 CPU 数目的两倍之间的一个值。要减少锁定次数以及促进最短工作优先 (SJF) 分配，可以减少 init.ora 参数 LOG_SMALL_ENTRY_MAX_SIZE 的值。它表示一个闩限值。如果一个过程的重做信息比它小——意思是如果它产生了相对小的重做日志项——它就可以独占地保持一个重做分配闩（并迅速写信息，而且比大过程更快地释放闩，从而增加并行操作）。用 SHOW PARAMETER 在 SVRMGR1 中检查它的值。每次把它减少一个百分比 (5% 或 10%)。重复这个过程直到重做争用数可以接受为止。

19.5 Oracle 8 的新输入/输出特性

除了在第 17 章中所讨论过的诸如唯一索引表和等分区对象等 Oracle 8 新索引特性以外，还有一些其他的 Oracle 8 的新特性为输入/输出性能的提高提供了机会，包括分区扩展表名与直接载入 INSERT。

19.5.1 分区扩展表名

与等分区对象（表和索引、相同值集合的相等分区）一样，表分区为分区视图提供了一个代用品。使用 PQO 的分区视图（Oracle 7）的创建方法是：水平地把一个单独的长表放到许多小表中的方法人工建立一个视图把它们联合起来，在每个小表中的值集合上建立约束条件，把 init.ora 的 PARTITION_VIEW_ENABLED 参数设置为 TRUE。分区表（Oracle 8）通过按选中的列分区建立。然而，与其有一个由联合的表组成的视图，还不如有一个由联合的分区组成的表。另外，你可以在一定的限定下直接引用分区：

```
SQL> SELECT * FROM T1 PARTITION (P1);
```

另外，Oracle 8 提供了把以前创建的 Oracle 7 分区视图转换为 Oracle 8 分区表的能力。

19.5.2 直接载入 INSERT

有了直接载入 INSERT，INSERT 就可以利用相同的 SQL*Loader 直接路径模式功能的优势，

这种方法绕过了数据库缓冲区缓存，不产生重做信息，直接写入到数据文件中。此外，在 Oracle8 中，你可以把一个表、索引或者是表空间放置在没有日志模式中。这就额外加强了你的直接载入 INSERT。直接载入 INSERT 由于性能的缘故要交替使用空间，因为它们在段的 HWM 上面插入行。这样就浪费了空间，但如果一个段经常接近于满的话，这不是主要问题，不要体验增长收缩模式，（特别是）要有一个表空间给它自己。对热表来说，这种能力可以用于与已计划的定期重组联合。

第20章 Oracle 8.x调整问题

本章要点：

分区

唯一索引表

并行数据控制语言

大池与非常大的内存

网络增强

数据库限制

第16章适用于任何厂家的 RDBMS 软件或版本，并且第 17~19 章中的大部分适用于 Oracle 7.x 和 Oracle 8.x 的 RDBMS，而本章几乎是专门适用于 Oracle 8.x 的 RDBMS 的。你将了解一下输入/输出、内存和网络关于性能的改进。

20.1 分区

当然，在 Oracle 7.x RDBMS 基础上的 Oracle 8.x RDBMS 的输入/输出性能的主要提高是真正的分区。在第10章中讨论了一些 Oracle 7.x 分区视图与 Oracle 8.x 分区的不同之处。比较明显，Oracle 8.x 有了某些性能上的优点。鉴于 Oracle 7.x 分区视图依赖多重表及那些联结起来的表的视图，因为确实没有新结构，所以没有本地语言支持。因此，所有事情（锁、优化等等）中输入/输出的级别仍然在表级。相反，在 Oracle 8.x 中，分区不仅是新的存储结构，而且被认为是本地语言，并且是优化的。

分区为性能带来的一个主要好处是分区消除。如果一个表在某一行（分区键）上被分区了，任何一条给定的查询（也就是 SELECT 语句）都可以被那些分区中仅仅一个分区中的行满足。一个例子是你已经把雇员在薪水等级列上进行了分区。如果你希望检查某个薪水等级，只需要存取一个分区以获得那些所需要的行即可。Oracle 优化程序知道这个，因为它可以把你需要的值域与那些你以前用来分区表行的值域相匹配。如果这里有 n 个分区要搜索的话，分区消除可以潜在地把性能加速 n 倍。

当然，就像 RAID 和 Oracle “人工”数据条一样，分区是一个分离存放的物理设计策略（参考第3章）。如果有 n 个分区需要存取，并且过程并行与数据并行同时发生（例如，并行查询），那么速度的提高可以接近 n 倍。然而，全部性能提高中还要减去再结合的总开销。另外，还有 Oracle 过程的某些并行方法、并行查询（Oracle 7.x 和 Oracle 8.x）及本章 20.3 节中的并行数据处理语言（Oracle 8.x）。

另一个需要考虑的分区问题是它在表和行之间增加了一个锁的级别。增加锁的级别几乎总是有帮助的，特别是如果它的级别比表低的话。考虑一下接近行总数的 $1/5$ 的多行锁的开销，同一个任务一个单独的分区锁就可以完成。因此，当需要行级锁时，它们被授予。但是当需要一个分区锁而不是一个表锁时，你就不仅节省了许多行锁的系统开销，而且对未锁定分区

提供了更多的并行存取。表锁在语法分析时确定，之后在运行时得到。分区锁都是在运行时确定和得到的，而表锁在任何时间都可以。

在Oracle 8.x中，有一个日志/无日志属性，此属性在表空间、表、索引或者是分区级别的某些SQL语句中是适用的。无日志属性代替 Oracle 7.x中的不可恢复选项。然而，为了以后的兼容性，目前仍然支持不可恢复选项。由于不可恢复选项是与一条语句联系在一起的，因此，在数据库对象中要指定无日志属性。例如，所有的插入（INSERT）操作（串行或并行的）不产生重做日志项，同时不记录实际上要受影响的对象。另一个例子，CREATE TABLE和CREATE INDEX可以使用不记录它们影响的对象，因为它们已经在它们的语句中使用了不可恢复的对象。此外，INSERT...SELECT和CREATE TABLE AS SELECT是可以利用无日志属性的潜在的操作。

直接载入操作作用并行方式支持；然而，它们也可以是串行的。直接载入与 SQL*Loader直接路径模式类似。直接载入只是对插入（INSERT）操作而言。它们使用临时段，它们将临时段直接追加到高于目标对象的高水位标志（HWM）的地方。对象的HWM被调节到新水平，同时新近插入的行被给定了ROWID。这种类型的载入与通常的插入方法不同，在写入目标对象的数据文件前必须经过与SGA相关的自由表。INSERT/*+APPEND*/提供了一个使用直接载入的优化程序提示，这可用于实现对象的直接载入。如果不需要直接载入，那么就不需要做些什么特别的事情，除非由于某些原因你想要完全保护它——那么就用NOAPPEND提示。

20.2 唯一索引表

有了基于B树变体的数据结构的普通Oracle索引，你基本上有该索引和表本身，这个索引是一个表的排序子集。用Oracle 8.x，你可以有唯一索引表，它们实际上就是表，这种表本身也有索引。这就是说表和索引的结构是一个而且是相同的。作为比较，Sybase总是在被物理排序的表中聚簇索引，这与被另一个索引结构逻辑排序的表（在Sybase术语中叫无聚簇索引）形成对比。因此，Oracle唯一索引表模拟Sybase的聚簇索引，普通Oracle索引模拟Sybase的无聚簇索引。

为什么需要唯一索引表？如果使用正确的话，普通Oracle索引（B树）对大多数数据库应用程序是非常有效的。然而，它们确实有一些缺点。首先，它们冗余地存储索引列。因此，性能与键大小成反比。其次，B树总是至少需要两个逻辑读（通常更多）：一个用于读索引，一个用于读表。这种情况如何能得到改善呢？一个很自然的扩充就是唯一索引表（IOT）。

在用这些索引表时有一些限制和需要考虑的事情。一个唯一索引表只能被物理排序一次，它被主键限制。一个IOT必须有一个主键。Oracle唯一索引表（和Sybase的聚簇索引一样）是一个倒排索引或倒排列表的RDBMS例子，它们是在3GL和现代DBMS的文件处理系统中使用的数据结构。而且，IOT既不会被分区也不会被复制。也许考虑使用IOT时最重要的问题是它没有ROWID！例如，因为没有ROWID，在IOT上建立其他索引是不可能的（这与Sybase相反，Sybase在一个聚簇索引表上可以有第二个索引）。IOT不能是索引簇或哈希簇的一部分。另外，除了主键外的唯一性限制不能被实施，因为对于Oracle唯一性限制（例如主键）是通过索引实施的。

清单20-1给你一个建立唯一索引表的例子。

清单20-1 建立一个索引表

```
SQL> create table employee
1 (eid number,
2  ename char(35),
3  eaddr varchar2(80),
4  resume CLOB)
5 primary key (eid)
6 organization index
7 tablespace data1
8 pctthreshold 90
9 overflow tablespace data2;
```

注意CREATE语句的几个事情。在一个给定的表空间（data1）中在主键（eid）上创建唯一索引表（employee），并通过使用organization index子句指定它是一个IOT。更进一步指定如果任何行超出了Oracle块长度（db_block_size）的90%，就要把那些行的非键列存储在溢出表空间（data2）中。这个溢出区域函数非常像链接行的溢出区域，该区域包含指向溢出段的溢出指针。在dba_tables和user_tables视图中的IOT列有一个新的IOT列，这个列被设置为IOT、IOT_OVERFLOW 或者是NULL（对常规的或者非IOT表）。

20.3 并行数据控制语言

我们已经在第10章中简要地讨论了分区所具有的并行特性。本章稍微深入一点，但不特别限定在分区的并行性。有三种主要类型的并行粒度，每一种适用于它自己的一组SQL语句，如表20-1所示。

表20-1 SQL语句的并行粒度

类 型	SQL语句	备 注
ROWID	SELECT	在表中或分区中
分区	UPDATE、DELETE	在分区级
从属	INSERT、SELECT	在表级

并行查询（PQ）结构在Oracle 8.x与Oracle 7.x中是一样的。但是，现在用PQ除了可以并行执行SELECT之外，还能够并行执行一些INSERT、UPDATE和DELETE操作。这个新的DML功能（记住：Oracle用DML指代INSERT、UPDATE和DELETE）被叫做并行数据控制语言（PDML）。PQ可以在一个分区内并行，而PDML只能在分区或表级并行。

ROWID级的并行被并行查询用于处理SELECT语句。表是否被分区了没有关系，因为并行性的粒度级别比分区（ROWID）低。这就意味着多个PQ从属可以从一个单独的表或分区中读。但是，没有PQ从属可以跨过多个分区。

分区或从属过程级的并行被PDML用于处理UPDATE、DELETE和INSERT...SELECT操作。首先，考虑UPDATE和DELETE操作。当从属的表被分区时，并行性就是分区级，用每个分区只允许一个从属分区的方法进行UPDATE和DELETE操作。当从属的表不被分区时，UPDATE和DELETE就不能并行。

不管从属表是否被分区，INSERT...SELECT都将由INSERT的从属过程通过按卷分配新行（没有ROWID）并行。多个从属以直接载入方式插入到目前高水位标志之上，这就象你在本章前面20.1节中直接载入性能的讨论中看到的一样。每个表或分区中都可能多个从属。

你可以在会话级使用PDML，而PQ一般是在实例级使用的，尽管在会话级也是可控的。

SQL代码中的并行提示按对 PQ所做的那样，对 PDML用同样的方法做。同时，优先权规则仍然应用于确定并行度。除了要考虑系统的 CPU数和要被存取的分区分数（如果被分区了）外，PDML跟PQ一样，只需选择下面项目中的一个来确定执行从属 SQL语句的并行度：

- 1) 提示指定的度。
- 2) 表缺省的度。
- 3) 一个连结中有最大度（提示或缺省）的表。

清单20-2给你一个巧妙的有提示的 INSERT...SELECT例子。假定表 T1和T2缺省度都为8。

Oracle执行这个语句时选择的并行度会是什么呢？

清单20-2 在SELECT语句中使用提示

```
SQL> INSERT /*+ parallel(T1,4) */  
INTO T1  
SELECT /*+ parallel (T2,6) */ column1, column2, column3  
FROM T2;
```

总并行度是4，因为首先考虑 INSERT提示，并且不论是表缺省度（8）还是 SELECT提示（6）都被忽略了。与它有关的具有最大度的表（这里，T1和T2都是8）不适用，因为没有涉及连结。

最后，当正常处理 PQ时，如何监控 PDML活动以帮助你调整并行度？PDML_ENABLED列已经被加入到V\$SESSION视图中，并且将详细说明是 YES还是NO，与对那个会话是否使用 PDML相对应。DML Parallelized行已被加入到V\$PQ_SESSTAT视图中，使你能看到有多少用于那个会话的DML操作已经被累积并行放置。当然，V\$PQ_SYSSTAT现在包含一个称为DML Initiated的新行，它表示所有累积的跨所有会话启动的 DML实例。注意，在V\$PQ_SYSSTAT中，所有其他的“老”行针对 PQ和PDML活动的和。例如，“服务器高水位”表示曾经被 PQ或者PDML启动的最多从属进程数。由于本节的讨论主要集中在 PDML上，请参考第38章以对 PQ进行更深入地学习。

为什么 PDML能够增强 Oracle 8.x的性能？n路数据并行能够为将速度提高几倍打下基础，进程并行（PQ和PDML）实际上也一样。因此，伴随着 RAID、条或者特别是 Oracle 8.x的分区分，PQ和PDML可以使速度提高n倍。

20.4 大池与非常大的内存

同在8.0.2中一样，大池是 SGA中一个可选的新内存区。它可以用于通过多线程服务器（MTS）支持UGA（会话特定的）内存再分配，甚至XA支持输入/输出从属的内存分配，例如那些被DBWR和LGWR（db_writer_processes、dbwr_io_slaves和lgwr_io_slaves）使用的内存，以及由备份和恢复操作引起的内存分配。

通过建立一个与共享池分开的大池，MTS不必在共享池中增长，因此要缩小库（SQL）高速缓存的可用空间。这是因为共享池不是动态再分配的，并且MTS和SQL高速缓存（与数据字典高速缓存一起）都竞争相同的内存资源。然而，在静态分配的共享池中，MTS和SQL高速缓存是被动态地分配的内存结构。为帮助减轻这种状况，可以分配一个主要由MTS使用的大池，并让共享池主要处理SQL高速缓存。这就打破了共享池的瓶颈，并防止了它内部的增大-缩小循环的争夺。

同备份和恢复进程一样，输入/输出从属能够分配成百KB的内存。在繁重的竞争下，共享

池可能跟不上这些进程，它们可能因此而缺乏内存。基本上，它们仍然处于长等待状态，没有进展或者每次进展非常少。大池也可以通过从共享池中除去一些竞争负担来缓解这种状况。因为大池没有LRU列表，正如你希望在高速缓存状况中看到的，它们的额外开销很少，仅仅充当一个大的轻量缓冲区。

相关的init.ora参数列在表20-2中。

表20-2 init.ora 参数

参 数	描 述
large_pool_size ^①	大池的大小
large_pool_min_alloc ^①	大池的最小分配
shared_pool_size	共享池的大小
shared_pool_reserved_size	大对象的保留列表的大小
shared_pool_reserved_min_alloc	保留列表的最小分配

① Oracle 8.x新增加的参数

记住，把共享池作为由Oracle分配给SGA的所有内存的一部分。

尽管非常大的内存（VLM）是7.3.x版本的功能，但是它被用户们相对地忽视了并没被充分利用。在Oracle 8.x中它会有更多的用处，这就是它包含在本章而不是包含在第18章中的原因。

Oracle非常大的内存功能由于Windows NT操作系统内存寻址的性能而被过大地想象了。尽管Windows NT可以运行在64位的机器上（例如Alpha服务器），但它仍然是一个32位的操作系统，受到32位地址的限制。例如，可以想象一个32位的机器有2的32次幂（即4GB）的内存要被寻址。然而，Windows NT保留了2GB给它自己（“系统可寻址能力”），2GB给用户线程（“用户可寻址能力”）。请注意，企业版的NT只需要保留1GB内存，因此有效地将最大用户寻址能力增加了1GB内存（达到3GB）。

然而，不是所有的NT都有4GB或者更多的物理内存。那么，Oracle如何在一个只有2GB物理内存（或者更少）的NT机器上运行呢？实际上，NT从最高的内存地址区向下增长，允许诸如Oracle的用户线程向上增长。问题变成了NT和Oracle是否能全部得到它们动态需要的内存而不受到对方的牵制。实际上，这种系统中通常发生的事情是NT或者Oracle开始发生同普通的内存分页形成对比的交换与系统失效，在这些情况下，通常由于内存（缺乏）资源错误引起致命的系统崩溃。

VLM是如何工作的？这个问题的另一种问法是用户过程如何存取已存在的、不直接由操作系统寻址的物理内存？答案是：间接地。Lotus、Intel和Microsoft许多年以前建立了一个规范，这个规范是为了突破所谓的DOS物理内存访问的“640KB或者1MB的屏障”。他们成功地建立了现在熟知的扩充内存的LIM规范。他们所做的是使用DOS的某些高端内存（在第一个640KB上方的384KB），它们是可直接寻址的，存储超过1MB的内存地址。本质上，把可寻址的内存作为间接地址，或者换句话说，作为内存分页表。LIM规范可用于诸如Lotus1-2-3电子表格的应用以便存取几兆字节的内存。尽管随着扩展内存以及后来的Microsoft Windows内存地址的出现，此技术现在已经陈旧了，但这在当时是非常创新的。

LIM如何与VLM相比？Oracle VLM是基于间接地扩展内存存取的完全相同的概念。尽管实际的实现不同，但是概念在功能上是相同的。内存给予者进程是虚拟用户线程，Oracle VBLM使用该进程在4GB的NT机器上扩展对超过2GB的内存的寻址能力。有了Oracle VLM内

存，可寻址能力的极限是安装的物理内存减去 NT所需的1GB或2GB内存。Oracle的检测已经显示性能提高了一两个数量级；换句话说，速度提高 10倍到100倍（这是非常好的，如果没有额外开销，理论上的最大收益大概有三个数量级的提高）。

如何让 VLM工作？可以使用 init.ora文件中的一个参数：use_indirect_data_buffers。把该参数设置为TRUE，Oracle不仅能在通常的SGA分配之外使用内存，也可以在超过由 NT的4GB的可寻址区或者任何其他的用户线程中使用内存。换句话说，这使 Oracle VLM成为可能。另外init.ora参数可能需要调整。请参考 Oracle 8服务器调整指南或者 Oracle 8调整指南，其中包含更具体的实现细节。下面是一些有用的 Oracle VLM信息的小结：

- 允许间接存取超过NT管理范围的物理内存。
- 与Oracle的企业版捆绑在一起。
- 在NT 4.0、NT 4.0 企业版和NT 5.0上透明地运行。
- 应用程序代码不需要改变。
- 已经显示了2~3个数量级的性能增加。

20.5 网络增强

和网络有关的主要增强功能能用积极的方法影响性能。被称为外部过程的结构现在是可用的。外部过程是 PL/SQL或者SQL的外部代码段。它是有用的，因为它总是完成一些非常复杂或者独特的处理，例如某些科学的程序设计。换句话说，许多以前传下来的执行得非常好的例行程序，除了Oracle集成外，不必重写它们。重新使用已有的工作代码是一个比重新编写代码更快的开发方法。

重新使用外部的例行程序而不是重写它们，这不仅是最快的开发方法，而且实际上可能比你用PL/SQL编写的任何程序还要快。例如，它们可以用一些精炼的快速 3GL语言（比如C）编写。此外，C也可以“包含”汇编程序代码，假定该代码被正确编写的话，那么它几乎同任何应用程序代码一样快。

在PL/SQL中调用外部进程的步骤如下所示：

- 1) PL/SQL使用别名库（alias library）调用外部进程。
- 2) PL/SQL把这个需求传送给Net 8监听器。
- 3) 监听器开始extproc专用进程。
- 4) extproc载入外部共享库。
- 5) extproc代表PL/SQL执行外部进程。

别名库是一个内部的Oracle库，它映射到一个外部（操作系统）共享库。要建立（联系）别名库，只需像下面这样使用扩展的DDL：

```
SQL> create or replace library libm  
2 is '/usr/share/libm.so';
```

用户需要什么特权来建立或者使用这些别名库呢？要建立一个别名库，管理员或者用户需要CREATE LIBRARY或者CREATE ANY LIBRARY。外部过程用户需要别名库自身的EXECUTE特权。CREATE LIBRARY语句的准确度只能在运行时检查，而不能在建立时检查。因此，需要确保共享库实际上同指定的绝对文件名（包括路径）正好相同。你可以在USER_LIBRARIES或者ALL_LIBRARIES中检查库的状态。

由监听器启动的 extproc 进程只能在会话限制一个外部进程调出的期间内持续。每个会话使用一个不同的 extproc 进程执行外部进程请求。你必须配置监听器以便与实例进行联系。建议使用独立的监听器，以避免由于存储器泄漏而造成的 SGA 损毁不致于波及外部程序或者它们同 PL/SQL 的交互。TNSNAMES.ORA 中的 extproc 项同任何其他项一样。但是 extproc 别名的名称必须是 extproc_connection_data，这在安装中已经预定义了。简单地将它的注释标志去掉，如果需要的话就修改它，然后使用。下面是一个 extproc 项的例子：

```
extproc_connection_data =
(
DESCRIPTION =
(ADDRESS = (PROTOCOL = ipc) (KEY = mysid) )
(CONNECT_DATA = (SID = mysid) )
)
```

20.6 数据库限制

数据库限制周期性地增加，无论何时，只要一个新的 RDBMS 软件出现，它们就趋向于增加。历史上，许多 RDBMS 厂家都有它们自己的缺点，不同的 RDBMS 有不同的弱点。然而，Oracle 8.x 的一些上限不太可能被大部分应用程序（甚至是大部分 VLDB）很快达到。

表20-3给出了新的数据类型的最大值。

表20-3 数据类型的最大值

数据类型	最大字节
CHAR (n)	n≤2000
VARCHAR(n)	n≤4000
VARCHAR2(n)	n≤4000
NCHAR	n≤4000

表20-4包括了其他数据库对象的最大值。

表20-4 数据库对象的最大值

对 象	最 大 值
数据库容量	512PB（千兆兆字节）
每个数据库的表空间	~2GB
每个表空间的数据文件	1022
每个表的列	1000
每个表索引列	32

20.7 小结

现在复习一下你已经学过的概念以及它们与性能的关系。你已经学习了 Oracle 8.x 的增强概念和结构，在表20-5中列出了与性能有关的细节及说明。

表20-5 Oracle增强

增 强	说 明	关 联 性
分区	数据并行	像RAID那样提高速度
IOT	反向表	逻辑输入/输出减半
PDML	进程并行	速度按因子提高

(续)

增 强	说 明	关 联 性
大池	卸载共享库	更大的可伸缩性
VLM	间接内存存取	在NT上存取超过4GB的内存
extproc	外部进程	调用C
上限	数据类型和对象	主要对于VLDB

所有这些增强都是对 Oracle 7.x 的显著改进。这样，Oracle 使用比以前更高的本地并行（数据和代码）能力向 VLDB 和数据仓库系统存储和检索数据。

第五部分 管理Oracle数据库

第21章 管理数据库存储

本章要点：

- 管理数据库对象
- 理解数据库碎片
- 管理回滚段
- 鉴别存储问题
- 管理增长的数据库
- 故障检查
- 方案：监控数据库增长

21.1 管理数据库对象

一个Oracle数据库有许多消耗磁盘和系统资源的组件。正如你所看到的，每个资源有唯一的存储特性并需要独特的处理和维持。在下面的部分中，你将看到数据库段及它们的唯一存储关系。我要讨论段本身，例如表、簇、索引、回滚段和临时段，以及 Oracle数据库存储它们时所使用的内部方法。

21.1.1 管理Oracle块

Oracle服务器可寻址的最小存储单元是 Oracle块。所有的数据库段都由 Oracle块组成，无论段是一个表、索引、簇或者其他对象，块结构是相同的。设计一个性能最优的数据库要从对Oracle块适当的配置和管理开始。

每个Oracle块都是由下面三部分组成的：

- 块头。
- 数据存储区。
- 自由空间区。

块头包含有关块的信息——什么类型的段数据存储在块中，什么段在块中有数据，块地址以及指向存储在其中的实际行的指针。块头由一个固定部分和一个可变部分组成，在块中块头通常使用85到100字节。

在Oracle块中，管理数据存储区和自由空间区域是彼此直接相关的。数据区域是块中实际存储行的地方。保留的自由空间区是一个区域，被定义为总的可用空间的百分数，被保留用于存储有关在块中的行将来更新的信息。管理数据和保留的块区域是 Oracle块管理所主要关注

的，这将在本章的后面讨论。

21.1.2 理解PCTFREE与PCTUSED

PCTFREE和PCTUSED是应用于段的两个存储参数，它们经常被误解，但是概念实际上非常简单。当Oracle向数据库中写信息时，它必须首先在一个段的分配区中找到一个或更多块来存储信息。Oracle保留了块的一个列表，这些块对每个段来说都是自由的（这也被称为自由表）。Oracle使用PCTFREE和PCTUSED参数的组合确定块何时何时没有足够的空间接受新信息。

提示 在许多情况中，Oracle对PCTFREE和PCTUSED参数的缺省值设置工作得非常好。因此，一直注意并牢记缺省设置的含义就非常重要，特别是要定期地注意行转移。

如果在块中自由空间的百分数比 PCTFREE参数大得多，它就要被添加到段的自由表中以便它可以用来存储新信息（例如，在表中的新行或者索引段的新索引节点）。当自由空间的百分数低于 PCTFREE时，块就被认为是“满的”，Oracle就把它从段的自由表中移出来。当块中使用空间的百分数低于 PCTUSED时，Oracle就把该块加到段的自由表中，这样它就可以再次被用来存储新信息。这种方法允许 Oracle保持足够的额外空间用于行增长，而不需要跨过超过一个块的空间。保持行被限制在一个单独的块中有助于使你的数据库以最高性能运行。

PCTFREE和PCTUSED的值永远都不能等于 100%。如果一个段被这样配置，块很可能将被连续地从自由表中取出，并且在每个数据子处理中又被放置到自由表中。由数据库引擎处理引起的系统开销可以通过在 PCTFREE和PCTUSED之间留至少 20%的裕量而很容易地得到解决。Oracle 为PCTFREE预置了 10，为PCTUSED 预置了 40说明这个裕量。

PCTFREE和PCTUSED在数据库段的存储子句中被指定。你可以用 dba_tables、dba_clusters 或者 dba_indexes数据字典视图查询它们目前的值。用 PCTFREE和PCTUSED，你可以微调单个段的存储特性以满足它们特殊的存储需要。例如，一个永远都不会被修改的表可以安全地把PCTFREE设为 0。这在每个块中都节约了空间——对一个 4KB的块以及设置为 10的 PCTFREE来说，每个块中大约有 800个字节可用于已被保存的数据。对一个 500MB的表，执行这个较小的改变，你会节约大约 50MB的空间。另一方面，如果一个表包含了将来肯定要被改变的列，那么就可以用高一点的 PCTFREE设置以避免行链接或者转移。行链接和转移在本章的21.2节中详细讨论。

图21-1显示了PCTFREE和PCTUSED如何在一起控制块内部的空间使用。在这个例子中假定PCTFREE被设为10%，PCTUSED被设为40%。编号1显示了新数据可以被加到此块中直到剩余的自由空间不到 20%。这是因为在块总开销后空间的 20%是留着为以后借助于 PCTFREE 参数进行修改用的。编号 2显示当达到PCTFREE的限制时，剩余的自由空间只能用于对现存数据的修改——在这个块中不允许新的插入。编号 3显示了由于删除或者修改而被重新放回自由表中的块，并且块中已用的空间下降到 40%以下。编号4显示当已用空间低于 PCTUSED时，在块中新的插入又是可能的了。当在块中进行插入时，已用空间开始增加，再一次接近上限，循环又开始了。

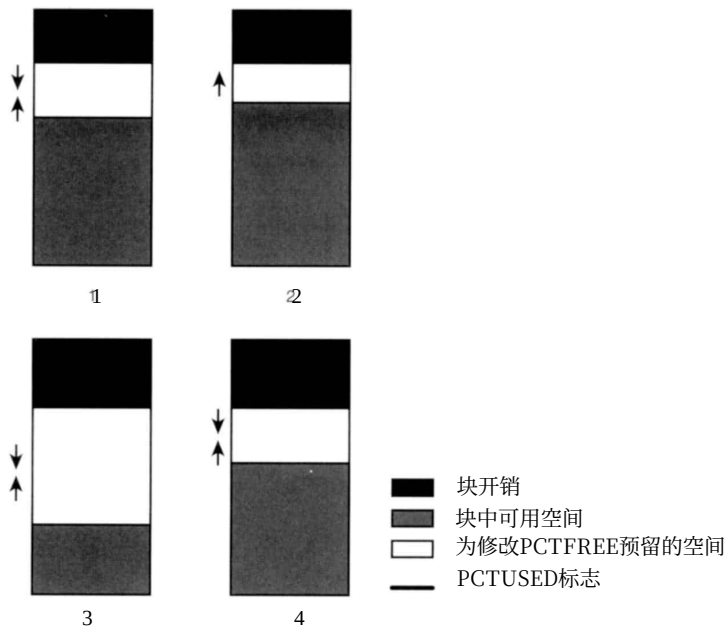


图21-1 PCTFREE和PCTUSED之间的关系

21.1.3 索引与PCTUSED/PCTFREE

索引是值得注意的，因为它们的结构性质，使用 PCTFREE和PCTUSED与表相比稍微有点不同。只有当一个索引最初建立时这些参数适用，在以后的修改期间没有更多的用处。因此，PCTFREE的设置应让索引结构最初在叶块中建立时就有足够的空间，这样就可以增加辅助关键字，而不必重复地拆分已存在的叶块提供给新关键字。由于在索引的建立期间不会删除行，因此索引就不能从任何 PCTUSED设置中获益。

提示 当为一个已存在的表建立新索引时，如果预料到表在近期将大量增长，就要考虑在建立索引时指定一个低的PCTFREE值。

21.1.4 管理表存储区

表容量（是行长×行数的函数）可能是确定存储需求的最好尺度。表的空间需求大致是平均行长与行数的乘积。当数据库管理员讨论他们所关心的数据库的近似容量时，行数经常被用做尺度。尽管它不会为计划长期存储需求提供许多信息，但它确实会使你了解某些有关数据库容量值的知识。

表通常是大多数存储管理的起始点；它们充当着一个排序标准，用来确定其他大多数数据库组件的存储需求。由于这个原因，理解包含在数据库表中的数据性质（数据类型、空列的数目等等）对数据库管理员是有利的。

表的容量由于新行的插入或者存在的行被修改为更大的值而增加。行的平均长度与行的数目是影响表容量的主要因素。

确定表的容量是一个概念上的任务，并且应该在表建立之前的设计阶段完成。问题，诸如预期的增长和存储参数值等问题应该在决定容量参数时处理。

考虑下面CREATE TABLE的例子：

```
create table employee (                                /* PART 1 */
  emp_no    char(4),
  emp_name  varchar2(30),
  age       number,
  dept_id   char(4)
)
  tablespace EMPLOYEE_TS                                /* PART 2 */
  storage ( initial 10M
            next    10M
            pctincrease 0
            minextents 1
            maxextents 50
            freelists 1)
  pctfree 10;
```

这个CREATE TABLE语句由两部分组成：第一部分是必须遵循的表定义，在这里指明关键字的属性，例如列、数据类型和约束条件。第二部分处理涉及到表的容量问题，这是你感兴趣的。让我们看一下对这些容量参数的说明：

- **TABLESPACE**——表空间就是表要在其中创建的地方。如果省略了，表就会建立在拥有者的缺省表空间内。
- **INITIAL**——给表分配的最初的区间容量。
- **NEXT**——如果最初的区间已经完全被表行填满了，那么该参数就是给表分配的下一个区间容量。
- **PCTINCREASE**——超尺寸的表的第三个（如果 **minextents** 被设置为1）和随后的区间的百分数。如果最初建立时以一种方法“懒洋洋地”处理扩展的表，这个参数很快会给粗心的数据库管理员制造使用期限的困难。例如，一个 **PCTINCREASE** 为50（缺省值）的表，**INITIAL** 设置为10M，10M 的**NEXT**有第一个10M容量的区间，第二个10M的区间，第三个15M的区间（10M×150%），第四个22.5M的区间，等等。计算第11个区间的容量，你会看到值0或者1被推荐给这个参数。
- **MINEXTENTS**——在建立时分配的区间数。对表来说，这几乎一直是1——如后面在21.3节中讨论的那样，回滚段有不同的需求。
- **MAXEXTENS**——表能分配的最大区间数。尽管版本7.3和以后的Oracle RDBMS使你能够无限制地分配区间，但是如果只是对一个对象胡乱地扩展进行报警的话，给这个参数赋一个值还是有用的。
- **PCTFREE**——在段的每个块中为现存的行将来发生改变预留的空间总数。
- **PCTUSED**——到了这一点时，块将被放回到自由表中。**PCTFREE**和**PCTUSED**在前面部分中已经讨论了。
- **FREELISTS**——指定要被存储到表、索引或者簇的自由表组中的自由表数。其缺省设置为1，并只在表中的一个块同时可能有许多修改时增加。

21.1.5 管理索引

索引可能是段的类型中最难管理的。许多应用程序员（以及数据库管理员）试图通过申请更多索引的办法解决数据库的性能问题。记住，更多的索引几乎总是损害修改性能，并且可能不会一直提供比较好的SELECT性能。在索引上保持关闭标志并删除任何不用的索引。

提示 用Oracle的EXPLAIN PLAN功能确定增加的索引是否对查询有所希望的影响。

许多不需要的索引会溜进你的数据库并引起性能问题。Oracle几乎总是宁愿要性能而不要节省磁盘空间。把每个没有主键的索引与索引所应用的应用程序以及建立这个索引要解决的特殊功能或者问题做一个记录。对几个为特殊应用程序建立的索引来说，没有什么与众不同的；如果没有记录，你会发现不再需要这个索引了。

21.1.6 监控回滚段

回滚段与诸如索引和表的其他段相比有点不同。因为回滚段实际上是动态的，数据库管理员通常应该集中精力保证每个回滚段及与之联系的表空间有足够的空间用于增长，而不需要使用太多的区间。

21.1.7 监控临时表空间和段

和回滚段一样，Oracle数据库中的临时表空间需要专门的监控。临时段仅在创建该段的事务处于活动状态时被保存。因此，高水位标志是你的度量尺度，而不是在某点有多少空间正被使用。目标是要确保系统要处理的最大事务有足够的空间。

21.2 数据库碎片

数据库管理员面对的一个最常见问题就是处理数据库对象的碎片。碎片浪费空间，导致性能问题，并给数据库对象的管理带来更大的困难。数据库碎片是许多问题，而不是一个问题——包括分裂成碎片的数据库对象，分裂成碎片的表空间、链接行和转移行。

提示 数据库管理员尤其应该注意数据库应用程序是如何执行任务的。开发者通常建立临时表或在程序结束之前便被删除的数据行。这实际上会导致碎片而不留下太多的痕迹。数据库管理员应至少了解一下数据库应用程序是如何设计的。

数据库碎片通常是行被插入、修改和删除以及对象被建立和删除的结果。因此，一个新建立的数据库就不会有碎片。只有当在数据库上运行了一个或更多的活动应用程序后，才开始看到数据库碎片的迹象。

在下面几节中，你会看到四种数据库碎片类型以及调整数据库并阻止碎片发生的简要方法。

21.2.1 分裂成了碎片的表空间

表空间变成碎片是由于错误以及表空间中的数据库对象的无计划撤消和重建造成的。当自由空间气泡在表空间的已用区域中被分离时产生表空间碎片。这在属于表空间的对象被删除时发生。这些被分离的空间气泡可以被重新使用，但是只有在一个对象被建立且能装入以前的对象留下的空间中时发生。

注意 表空间不会由于任何行级动作而变成碎片。这是常见的——但是完全错误的——误解。

例如，假定在一个表空间中有四个表：customer、employee、stock和company。这些对象中的每一个都由一个10MB的区间组成。表空间的结构看起来如图21-2所示。注意表空间末尾10MB的自由区间，这些对象压缩得很紧。这就是目的，是你应该达到的。

当然，你不可能让这种状况持续太长时间。假定由于性能需要，你必须把 customer表移到另一个表空间中。表空间现在看起来就像图 21-3。注意表空间中10MB的空的部分。如果你需要建立一个长度小于或等于 10MB的另一个对象，没有问题。但是假定你需要在这个表空间建立一个policy表，其存储参数为15MB的初始区间、10MB的下一个区间。这个表空间总共有 20MB 的自由空间，但是任何自由区域的最大长度只有 10MB；因此创建policy表的努力就失败了。

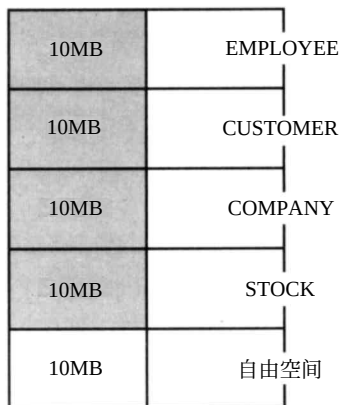


图21-2 分裂为碎片前的表空间结构

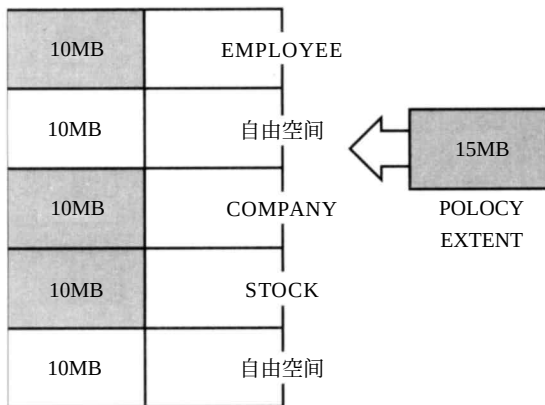


图21-3 在删除一个表之后表空间中的碎片

这个例子显示了一个小规模的问题，在表空间中只有一个对象被删除了。考虑一个繁忙的实际环境，在这里对象被删除以及不同大小的对象被建立几乎是连续不断的。这个动作导致组成表空间的自由空间的非连续气泡——与图21-4中描述的类似。表空间的总自由空间看起来是足够用的，但是在表空间中并没有大的连续自由空间区域，这会严重地限制你使用自由空间的能力。

Oracle RDBMS尽力帮助数据库管理员进行空间分配和管理。考虑一下表空间中四个表的相同情况。现在假定你删除其中的两个表，这两个表都位于表空间的“中间”位置。这就给了你一个如图 21-5所示的表空间结构。如果对一个 15MB表的相同需求被执行了，SMON后台进程自动地把两个 10MB的区间聚合为一个 20MB区间，用它实现对象的建立。如果表空间的PCTINCREASE被设为非零值的话，SMON也可以自动地聚合空间。

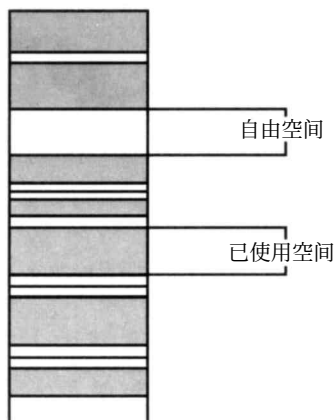


图21-4 一个完全分裂的表空间

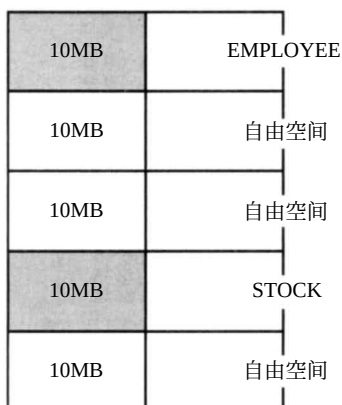


图21-5 在删除两个表后，表空间中的碎片

你也可以用下面的命令在表空间中人工聚合空间：

```
alter tablespace armslivedb01_ts coalesce;
```

下面的脚本建立了一个视图，你可以用它来识别表空间中有问题的段。用 `tablespace_name` 查询这个视图以获得一个空间问题会发生在哪儿的概念，也可以查看一下分裂成碎片的单个对象如何存在于表空间中。

```
CREATE OR REPLACE VIEW ts_blocks_v AS
SELECT tablespace_name, block_id, bytes, blocks, '== free space ==' segment_name
FROM dba_free_space
UNION ALL
SELECT tablespace_name, block_id, bytes, blocks, segment_name
FROM dba_extents;
```

```
Select * from ts_blocks_v;
TABLESPACE_NAME      BLOCK_ID  BYTES      BLOCKS      SEGMENT_NAME
-----
PROG_PLAN_IDX_08      34562    221962240   27095 == free space ==
BMC_SMG_T            339      102088704   12462 == free space ==
DBA_TEST              42372    533700608   65149 == free space ==
BPW_DATA_01          111667    133808128   16334 == free space ==
BPW_IDX_01            155439    299515904   36562 == free space ==
AUDIT_01              84231      40960        5 SQLAB_COLLECTION
AUDIT_01              2         1064960     130 TB_225AUDIT
AUDIT_01             2692      532480      65 TB_237AUDIT
BPW_DATA_01          105737    41943040    5120 TB_G026CAPACITY
BPW_DATA_01          105702    286720      35 TB_G016RESOURCE
BPW_DATA_01          105667    286720      35 TB_G006NSU
BPW_DATA_01          105602    532480      65 TB_7056SUBST_AUDIT
BPW_DATA_01          105567    286720      35 TB_64S6MSG_TYPE_CODESET
BPW_DATA_01          101982    29368320    3585 TB_64R6CAPACITY_CONSTRAINT
BPW_DATA_01          1092      286720      35 TB_5317INV_RI
```

表空间碎片会导致下面的问题：

- 表空间中的空间被分离且不能有效地使用。
- 在需要重建分裂成碎片的对象时会导致管理问题。

注释 在某种程度上，在RDBMS和磁盘技术上的进展已经把分裂成碎片的表空间对整个系统性能的影响减到了最小。然而，仍然需要减小表空间碎片，以便最优化系统性能并使磁盘空间得到最有效使用。

提示 检查数据库碎片时，一定要检查行链接或者转移。这两个问题很容易被忽视，并导致可测的性能问题。

21.2.2 处理分裂成碎片的表空间

处理表空间碎片的最好方法是避免它。这可以做到，但是需要仔细地计划和管理的额外开销。如果你面临着整理分裂成碎片的表空间，最简单的方法是在表空间中导出损坏了的对象，删除该对象，再把它们导入回来。这不仅能够聚合你的自由空间，而且导出操作也可以把你的数据库对象聚合到一个区域中。

要避免碎片，有以下几种方法：

- 把对象用相似的空间和增长特性聚合在一起。
- 如果可能，把它们的空间设成相同的容量，这样所有的对象可以共享释放或者从删除对象中回收的空间。

■ 使每个段的PCTINCREASE保持为0。

按照这些提示做会消除所有用不同大小建立的以及将来的数据库段不能使用的区间。

注意 从Oracle 7.3开始,在表空间上设置缺省的PCTINCREASE比0大会导致SMON自动聚合相邻的自由区域。除非你的数据库系统绝对不会分享CPU时间或者磁盘输入/输出,否则你应该把这个参数至少设为1。

最后,下面的脚本向你展示了在你的数据库的每个表空间中有多少自由空间是可用的和最大自由区间的容量以及每个表空间中自由区间的数目。在查找问题时这是一个好的起始点——如果你在表空间中看到了500MB的自由空间,但是只有一个最大15MB的自由区间,那么就明显有问题。

```
SELECT tablespace_name, sum(bytes) "Free Bytes", max(bytes) "Largest Extent",  
       count(block_id) "# Extents"  
FROM dba_free_space  
GROUP BY tablespace_name;
```

21.2.3 对象碎片

分裂成碎片的表空间的确会使任何数据库管理员对空间分配感到头疼。潜伏的和容易被忽视的碎片问题发生在对象级。在对象级碎片中,对象在表空间级可能看起来很好并很健康,有一个单独的区域,但是在那些区域内部的某些检查可能会揭示出问题。

一个常见的情形是数据库管理员在一个经常改变的表空间中发现空间分配问题,并重建了表空间和它的所有对象。随后,数据库性能提高了。这种在性能上的提高是消除表空间碎片的结果,但也可能是消除对象级碎片带来的性能的提高。

提示 虽然技术上没有碎片,但许多DELETE操作会产生很少被填充的块,这些块还不满足PCTUSED的被放回到自由表的条件。大部分打算处理对象碎片的数据库的操作也会处理稀疏的块问题。

对象碎片会导致下面的问题:

- 对数据库额外的读调用会导致响应时间的增加。
- 由于在表和索引块中的自由空间空洞导致空间的浪费。
- 读性能下降。因为数据不再被紧紧地排在一起,物理磁盘驱动器必需从一个比需要的大磁盘表面区域中搜寻和读数据。

对象碎片是三个特殊碎片类型的“保护伞”:行转移、行链接和过分扩展的段(对象有大量的区间或者几乎接近maxextents参数)。我在下面几节中讨论这些碎片类型。

21.2.4 行转移

行转移在对行的修改引起行的长度比块中可利用空间大时发生。当发生行转移时,行的一部分就转移到一个新的数据块中。这个新单元的地址存储在原始行单元中。当Oracle需要读这个行时,它首先读原始单元并找到一些数据和指向另一个块的指针,再在这个块中找到剩下的行数据,见图21-6。

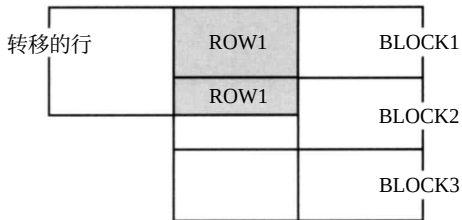


图21-6 块中行转移的例子

行转移存在两个问题：

- Oracle每次必须执行至少一次额外的输入/输出读来获取转移的行。它必须读每一个包括行数据的一部分或者是指向另一个包含着行数据的块的指针的块。
- Oracle也必须和行数据一起存储额外的指针来供给行转移机制。但这通常是不重要的，它浪费了一些磁盘空间。

注意 为什么Oracle执行行转移而不是把整个行移到一个新块中？移动一个行的单元是非常昂贵的，而作为行ROWID的锁定（ROWID与它的物理位置紧密相连）被存储在数据库的索引和其他区域中。对每个ROWID，Oracle必须对ROWID的所有引用有一个反向索引，这样它们就能被有效地修改。无疑这会导致相当大地全系统的额外开销，而这是不必要的。

21.2.5 行链接

行链接在一个行太长以致于不能放入任何一个数据块时发生。这导致该行被存储在一个或多个数据块的链中。行链接经常伴随着包含 LONG、LONG RAW 或者 LOB 数据类型的大行发生。参见图 21-7 了解有关一个表中链接行的更多信息。

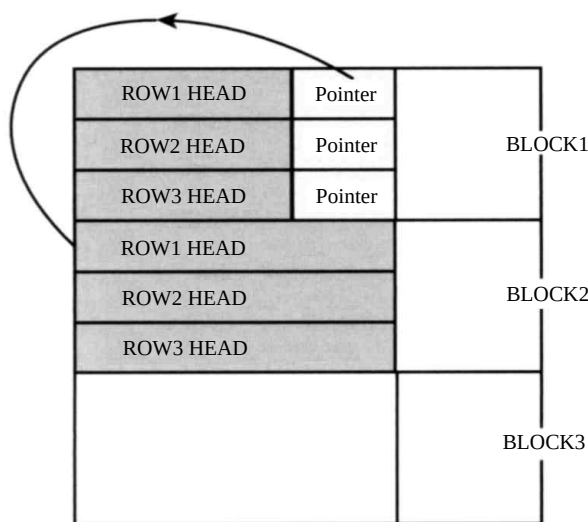


图21-7 行长度比块尺寸大导致行链接

21.2.6 消除链接和转移的行

链接和转移的行会对你的数据库的整体性能造成一系列的问题。你可以通过发出下面的 SQL 语句检查链接和转移的行：

```
analyze table customer list chained rows into chained_rows;
```

要建立 chained_rows 表，执行 \$ORACLE_HOME/rdbms/admin 目录下的 utlchain.sql。当你运行了 utlchain.sql，一个被称为 chained_rows 的表就建立了，它就是该 SQL 语句所要达到的目的。

这个命令把ROWID和数据库中所有链接和转移行的表存储到 chained_rows表中。直接查询chained_rows表，找出行和表的特定信息。

因为转移行仅当一个行被修改为一个新值时发生，你可以通过执行下面的步骤消除所有转移行：

1) 使用如下的SQL语句建立一个临时表来保存转移行：

```
CREATE TABLE temp_emp AS
SELECT * FROM emp WHERE rowid in
(SELECT rowid FROM chained_rows
WHERE table_name = 'EMP');
```

2) 从主表中删除前一语句存储的行：

```
DELETE FROM emp WHERE rowid in
(SELECT head_rowid FROM chained_rows
WHERE table_name = 'EMP');
```

3) 从临时表中插入行：

```
INSERT INTO emp SELECT * FROM temp_emp;
```

这个过程消除了所有转移行。通过删除在 chained_row中引用的行并重新运行 analyze语句，你一定能识别所有链接的行。

这个解决方案的一个问题（特别是对大表而言）就是你浪费了老的数据块中所有的空间，这些链接的行原来就保存在这些块中。插入的行不适合那里，这样所有删除的行都在表中留下了自由空间块，这些块也就是以前的行被放置的地方。这种行压缩类型有两个主要的缺点：

- 表中的许多自由空间可能永远都没用过，这取决于 PCTUSED的设置，因此空间被浪费了。

- 当这个表通过全表扫描被读取时，因为必须读非常多的块，所以可能会损失一些性能（尽管这通常不一定发生）。

可替代它的方法是把数据从表中存储到一个临时表中，截断表，把数据插入回去。你也可以使用导出，删除表，之后使用导入重建该表。后面的解决方法是最简单、最全面的，但也是最消耗时间的。

在尽力避免行转移时，确定 PCTFREE的值很重要。你可以用一个粗略的推算得到 PCTFREE，该推算假定表有一个平均的行长 x字节，y字节在最初插入后将被修改。有了这些必要条件，使用如下公式取得 PCTFREE：

$$PCTFREE=100*y/x$$

例如，如果你有一个行长为 100字节的表，21字节稍后将被修改，PCTFREE的值是 $100 \times 21/100$ ，即21%。

行链接是非常难调整的。唯一的解决办法是缩短表的行长或者增加数据库块的大小。选择前一种方法通常需要对表结构进行重新设计，但是后一种方法需要导出整个数据库，重新创建然后全部导入。这两个方法都不是很吸引人，但是如果一个表中的链接行引起了性能问题，那么就必须使用它们。

提示 在采取任何强有力的步骤消除行链接以前，看一下损坏表中的数据类型。Oracle中的一些较新的数据类型（例如 VARCHAR2）比它们以前的对应类型（CHAR）更有空间意识，并且几乎不需要改变应用程序。

21.3 管理回滚段

数据库回滚段的管理可能比数据库中其他段的管理要复杂。因此，在你为这些有些麻烦的对象考虑策略前理解回滚段操作的复杂性很重要。

回滚段被用来提供读一致性（在本章后面解释）、事务回滚和数据库恢复（在第24章中讨论）。它们存储一个事务中涉及的数据的前像（before image）。如果事务希望取消所做的改变，回滚段中的信息可以把数据库恢复到以前的状态。这被称为事务回滚。

像任何其他段一样，回滚段被定位在表空间中并且包含区间。每个数据库有一个SYSTEM回滚段，它位于系统表空间中并在数据库建立时建立。普通用户不允许使用它并且它不能被脱机使用或者删除。你应该为专用于回滚段的表空间中数据库的正常工作建立附加回滚段。回滚段的扩展和压缩是很普通的，它们不是你在表、索引或者系统表空间中所希望发生的行为。

必须调整回滚段以便处理并行事务以及不同大小的事务。通常，在有許多并行事务的系统上你需要有更多区间的更多回滚段。你的事务越大（例如，修改大量的行），你的回滚段也就需要越大。因为回滚段通常允许增长，然后收缩回存储参数OPTIMUM的值，一定要特别小心以确保你真正知道一个回滚段需要多少空间。

提示 Oracle的“快照太旧”错误信息通常是由太少和/或太小的回滚段引起的。

21.3.1 回滚段操作

事务或者是明确地申请或者是被隐式地指定到一个特殊的回滚段中。多个事务能够并行写入任何给定的回滚段或者该段中的区间。然而，块专用于某个事务。Oracle用循环的方式使用存在的区间和块分配给一个回滚段。根据需要（并且在可用的存储参数允许的范围内），新区间被分配给当前的事务。当事务结束时，它们的回滚信息不被删除（记住，回滚段是循环写的）。这是为了容纳其他已经存在的事务，这些事务可能仍然依赖于已经变化的用于数据库读一致性视图的数据的老版本。

注意图21-8对这个回滚段扩展性的描述。

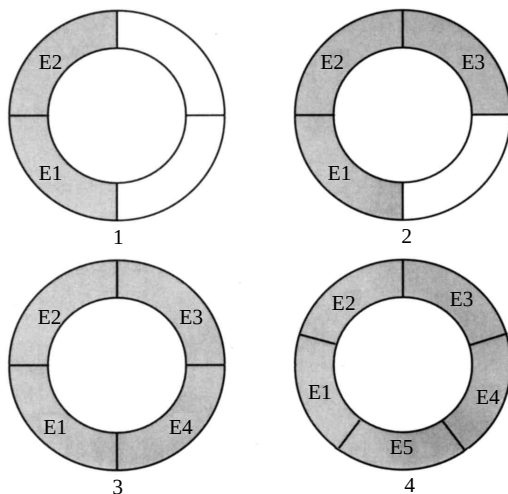


图21-8 回滚段中区域的分配

在图21-8中，图1到3显示Oracle使用已存在区间的顺序；图4显示Oracle增加一个区间到回滚段中。图1显示Oracle活跃地使用区间E1和E2，同时E3和E4被空闲。图2和3显示Oracle使用剩余的区间处理事务回滚信息。图4显示Oracle分配另一个区间E5来容纳活动事务。除非Oracle需要分配另一个区间给活动事务，否则下一个被用到的区间又将是E1。

注意 如果达到了一个回滚段的maxextents，或者表空间中没有更多的空间分配给回滚段的话，扩展就失败了。

回滚段的动态扩展是一个代价很高的操作，并且会导致性能干扰。如果回滚段没有小心地放置在整个数据库的结构中，它也会引起空间问题。应该给回滚段分配为回滚段保留的表空间。包含在任何给定的表空间中的所有回滚段应该被精确地分配成相同的大小——因为回滚段会频繁地扩展和释放区间；如果设置了不匹配的区间大小，那么前面讨论过的碎片问题确实是个问题。

提示 有特殊设计的大小的特殊回滚段是为特殊事务类（例如DSS和OLTP）创建的。因为这些是将被设置为不通用大小的回滚段，它们应该存储在它们自己的表空间中以使碎片减到最小。

这里有一些有关回滚段中区域的扩展和分配的一般概念：

- 回滚段中单独的区间可以被多个事务写入，但是区间中的每个块必须只包含一个事务的信息。
- 不管系统中存在多少回滚段，一个事务只能被分配给一个回滚段。
- 除非Oracle明确地申请某个回滚段，否则它使用循环方式来确定事务将使用的回滚段。尽管它试图使每个回滚段处理相同数目的事务，然而要猜想什么回滚段将要分配给事务变得很困难。
- 分配的区间在回滚段中按照块ID的升序顺序使用。例如，如果回滚段中第五个和第七个区间是自由的，那么第五个区间就总是比第七个区间早分配。
- 每个回滚段（和大多数段一样）必须最少有两个区间。

21.3.2 设定回滚段大小

当设定回滚段大小时，记住它们应该被设置得足够大，这样系统就不必马上重新使用刚被一个完成的事务释放的块。其他的并行事务可能仍然需要使用回滚段信息，这个回滚段信息是由修改/插入/删除事务产生的以完成数据库一致性读视图。

考虑一个长时间运行的报告，这个报告给出关于一个公司中所有雇员薪水的详细情况。这个报告遍历Employee表的每一个记录，并包含与之有关的其他表的信息。假定一名HR雇员在下午3:00发出报告。在下午4:00，薪水册中的另一个用户决定雇员薪水的年增量应该被处理（这是一个部门间很少联络的明显例子）并执行这个维护的工作。这个进程修改了公司中大部分雇员的记录，并马上提交了这个信息，参见图21-9。

提示 回滚段的使用受到事务运行的某种类型修改动作的影响。通常，DELETE消耗大部分的回滚空间，INSERT消耗的回滚空间次于UPDATE

其间，原来的HR报告仍然在变化着。它到了紧急时刻，也就是需要检索的雇员记录的薪水已经被薪水增加进程改变了。Oracle此时应该做什么呢？报告应该发出被提交的以及当前的

薪水详情，还是应该返回增加之前记录的薪水？

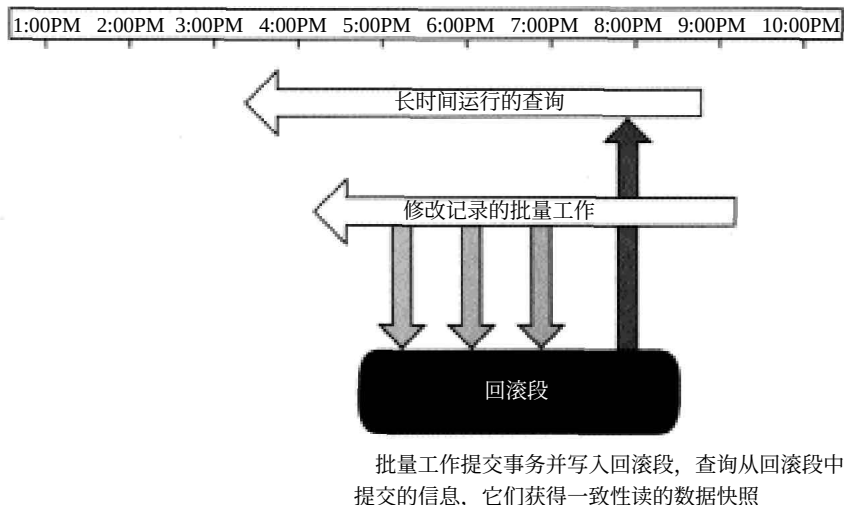


图21-9 长时间运行的查询与批量工作

如果报告得到当前的信息，那将意味着报告中的一些雇员拿到了增加前的薪水，一些拿到了增加后的薪水，这就会使报告前后矛盾并且没有用处。这就是回滚段的重要性显现的地方。通过Oracle的事务处理机制，以前修改的在增加薪水的处理程序中改变的所有行的映像被写进了回滚段。当报告程序到达一条已被修改的记录时，它会用块控制信息（SCN、或者系统改变号）确定需要注意回滚段以便找出报告开始运行时的信息。

注意，对这种情况所做的适当处理在事务有足够的回滚空间时是完全可信的，这样，写入回滚段的信息就不会被随后的事务重写。如果事务需要的信息被重写了，那么你会得到一条“all too common and hateful Snapshot Too Old”的错误。

前面的例子是在设置回滚段容量时要考虑的一个方面。从讨论中看到，很明显在回滚段中应该有足够的空间来存储这些非活动的或者被提交的数据事务，这些数据事务可能是其他长时间运行的查询所需要的。

你可以通过采取下述措施避免长时间运行的查询的问题：

- 分配足够大的回滚段空间以容纳最有可能的事务并运行情况。
- 采用调度法，由此长时间运行的查询与小的 OLTP 事务会在不同时间里运行。
- 建立一个只被长时间运行的批量工作使用的大回滚段。

提示 使用SET TRANSACTION USE ROLLBACK语句在事务开始时明确地使用某个回滚段。

```
set transaction use rollback segment RB1;
```

21.3.3 使用OPTIMAL参数

当建立回滚段时，回滚段的目标容量可以用存储参数 OPTIMAL定义。OPTIMAL参数指定了回滚段要缩小到的尺寸。

让我们看一个create rollback segment 语句的例子：

```
create rollback segment r05
```

```
tablespace rbs
storage ( initial 10M
          next      10M
          optimal 20M
          minextents 2
          maxextents 100);
```

RDBMS尽力把回滚段的尺寸维持在指定的最佳值 20MB上。这个尺寸被解释为上对齐到区间的边界，意思是 RDBMS尽力保持最少的区间数，这样总尺寸就大于或等于 OPTIMAL指定的尺寸。

无论何时，回滚段开始写入另一个区间时，Oracle检查段是否比OPTIMAL的尺寸大，如果大的话，那么就检查下一个区域是否完全空闲的。如果两个检查都是正确的，Oracle就释放该区间并继续进行下一个区间（这个区间如果满足相同的标准也会被释放）。当试图释放非活动区间时，最早的一个首先被释放。

OPTIMAL参数在某些方面被认为是存储参数PCTINCREASE的兄弟——懒惰的数据库管理员的管理方法。尽管它确实使回滚段的管理稍微简单一些，但是证明也会做很多有害的事情。

警告 如果发现你的回滚段由于OPTIMAL子句的缘故持续地收缩，那么很可能你的回滚段被不适当地设置了尺寸。如果这种区间的分配和释放很常见，那么它会影响到性能。

21.3.4 进行装载测试以获得回滚段估算

为了正确地设置回滚段的尺寸，你需要对将由较大事务产生的撤消信息的数量有个了解。通常，每个回滚段都应该足够大以适应最大事务回滚需要，如果不够大的话，它们就会无节制地浪费空间（此情况的唯一例外就是如果你有前面讨论过的专用回滚段）。为了对撤消产生的尺寸有个合理的估计，在样本事务中执行一个装载测试，使用清单 21-1，这个样本事务会产生大量的撤消信息。

提示 如果你不知道某个作业正在对数据库发出什么SQL语句（并且没有可用的源代码）的话，那么考虑使用Oracle的跟踪工具来捕获准确的要进行调整的SQL语句。

在你正在执行测试时，要确保没有其他活动事务产生撤消信息。下面的步骤产生一个装载测试：

- 1) 建立两个表保存你的统计信息：stat\$undo_begin和stat\$undo_end。
- 2) 假定装载测试在回滚段 R01中执行。在与数据库连接后，发出一系列事务处理命令，以确保所有属于此事务的撤消信息被写入回滚段 R01。
- 3) 使用v\$rollstat表把写入回滚段R01的原始数据捕获到表stat\$undo_begin中。
- 4) 发出测试事务的语句。
- 5) 在事务处理结束后，再把写入回滚段 R01的数据放到表stat\$undo_end中。
- 6) 既然你在表中已经有了开始和结束值，只需将这两个值相减便可得到写入回滚段 R01的数据。这些值可以用于确定你的回滚段的尺寸需求。

清单21-1是执行上述步骤的脚本。

清单21-1 执行装载测试以获得回滚段尺寸的估计值

```
REM
REM Create the set up tables
----
```

```
REM
create table stats$undo_begin (writes number);
create table stats$undo_end   (writes number);
REM
REM Capture the initial number of write for rollback seg R01.
REM Replace R01 with the name of your rollback segment.
REM
insert into stats$undo_begin
    select sum(writes) from v$rollstat
    where usn = ( select usn from v$rollname where
                  name = 'R01');

REM
REM Alter the current session so that it uses the test rollback
REM segment only. Replace R01 with the name of your rollback segment.
REM
commit;
set transaction use rollback segment R01;
REM
REM Run the long running batch job which generates largest undo data.
REM Replace the name of your batch job script with 'large_batch_job'
REM
@large_batch_job;
REM
REM Capture the writes generated at the end of the batch job.
REM Replace R01 with the name of your rollback segment.
REM
insert into stats$undo_end
    select sum(writes) from v$rollstat
    where usn = ( select usn from v$rollname where
                  name = 'R01');

REM
REM Get the amount of data that has been generated as redo for the
REM transaction under tesst
REM
select (e.writes - b.writes ) undo_generated
    from stats$undo_begin b, stats$undo_end e;
UNDO_GENERATED
      898998
```

在清单 21-1 的输出中，由批处理工作产生的撤消信息有 898998 个字节。这个脚本可以用作适当地设定回滚段尺寸的起点。

21.4 鉴别存储问题

当数据库启动和运行时，有时你需要再次查看数据库的每个主要组件有多大。最关心的是要确保表空间足够大，能够存储它们所包含的段。表、索引与簇随着时间的推移都会增加，你需要确保 INSERT 和 UPDATE 在生产期间不会因为表空间超出空间之外而失败。

当我讨论为段分配的空间时，我指的是为段预留的空间。这并不表示段实际上正在使用所有的空间（尽管，它几乎总是使用空间的一些部分）。例如，假设表 ABC 在两个分别为 50MB 和 40MB 的区间中被分配了 90MB 的空间。表 ABC 实际上正在使用 50MB 和 90MB 之间的空间。当第一个区间被用尽了，表定义中的 NEXT 参数指示分配另一个 40MB；即使实际上只需要 10KB，也会这么做。ABC 不需要再去寻找更多的空间，直到剩余的 39.99MB 已被填满。尽管这种设计对一些不是从主机数据处理环境来的数据库管理员来说有些古怪，但它在 Oracle 的性能特性中是一个关键的要素。

使用Oracle企业管理器的数据库管理员会发现访问数据库的并行存储状态的工作比它们相对应的命令行方式简单。存储管理器是企业管理器集的一部分，它为大部分普通的有关存储的信息需求提供了GUI接口。图21-10显示了存储管理器的一个示例窗口。

你的数据库不必驻留在 Windows NT机器中来使用存储管理器。Oracle的Net 8网络组件使运行存储管理器的 Windows系统能够和远程机器上的数据库一起工作。远程机器可以驻留在本地或者广域网上，并且可以是任何 Oracle支持的平台。

有经验的数据库管理员会发现存储管理器非常像基于 Windows的Oracle GUI工具，它只是缺少几个用于有效管理存储的重要功能。存储管理器是一个非常有用的工具，它可以简化日常工作，但是它不能替代命令行界面。

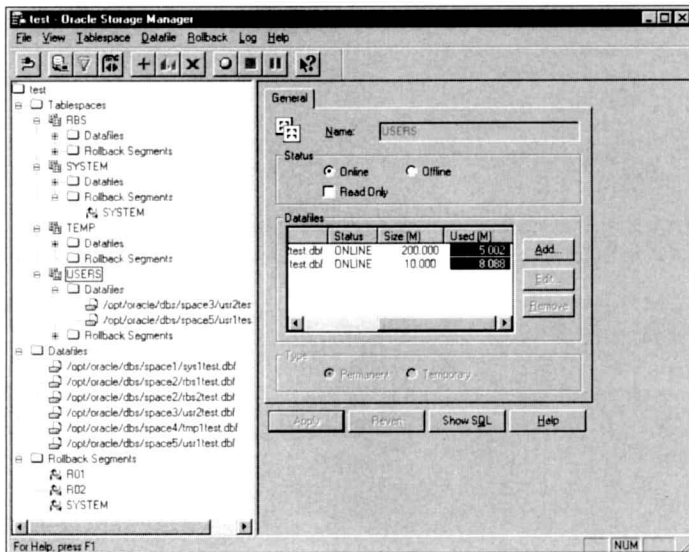


图21-10 Oracle存储管理器的例子

21.4.1 开发表空间

数据库表空间的开发通常包括解释表空间的总尺寸以及表空间使用什么样的数据文件。DBA_DATA_FILES表包含所有关于数据文件的信息。因为表空间的尺寸是它的数据文件的总和，所以你也可以从DBA_DATA_FILES表确定表空间的尺寸。

这里是一个例子：

```
COLUMN TABLESPACE_NAME FORMAT A10
COLUMN FILE_NAME FORMAT A35
SELECT TABLESPACE_NAME, FILE_NAME, BYTES, STATUS FROM DBA_DATA_FILES
ORDER BY TABLESPACE_NAME;
TABLESPACE FILE_NAME                                BYTES STATUS
-----
RBS          /u02/oradata/test/rbs1test.dbf      20971520 AVAILABLE
RBS          /u02/oradata/test/rbs2test.dbf      20971520 AVAILABLE
SYSTEM       /u01/oradata/test/sys1test.dbf      20971520 AVAILABLE
TEMP         /u04/oradata/test/tmp1test.dbf      10485760 AVAILABLE
USERS        /u05/oradata/test/usr1test.dbf      20971520 AVAILABLE
```

你看到数据库中每个数据文件占一行，该行显示了它所属的表空间、它的尺寸（用字节

表示) 以及状态。状态域几乎总是 AVAILABLE。

应经常检查表空间以便确定在段需要增长的情况下有很多可用的自由空间。这个问题不像初看上去那样简单。为了回答这个问题, 你需要知道下面的事情:

- 表空间有多大。
- 总计还有多少自由空间。
- 最大的自由区间是什么。
- 是否任何一个段都可以至少可以通过一个区间增长。
- 表空间如何分裂成碎片。

你需要从知道一个表空间有多大开始。把与数据文件相关的尺寸合计起来可以很容易地为你提供那些信息。例如:

提示 通常, 可以很容易地编写一个自动搜集有关每个数据库存储信息的脚本, 并把该脚本发送给适当的数据库管理人员。

```
SELECT TABLESPACE_NAME,SUM(BYTES) FROM DBA_DATA_FILES GROUP BY TABLESPACE_NAME
```

TABLESPACE	SUM(BYTES)
RBS	41943040
SYSTEM	20971520
TEMP	10485760
USERS	20971520

下一个问题是确定表空间中总计还有多少自由空间。Oracle在DBA_FREE_SPACE表中保存一张数据库中所有自由区间的清单。通过合计表空间中自由区间的尺寸, 你就会看到还剩多少自由空间。下面这个例子显示了数据库中每个表空间的自由空间数。

```
SELECT TABLESPACE_NAME, SUM(BYTES) FROM DBA_FREE_SPACE  
GROUP BY TABLESPACE_NAME;
```

TABLESPACE	SUM(BYTES)
RBS	29343744
SYSTEM	11552768
TEMP	10483712
USERS	9439232

了解每个表空间中的最大自由区域有助于你确定是否任何一个段至少能够通过一个区间增长。尽管表空间USERS可以有9MB的自由空间, 但它可以被分成三份, 每份大约只有 3MB。了解这一点是很重要的, 因为这可能存在一张区间的 NEXT值为5MB的表。如果是这样的话, Oracle不能分配额外的区间, 因为这里没有至少 5MB的可用自由区间。你需要再次求助于DBA_FREE_SPACE表, 只是现在要寻找最大的自由区间, 如在这个例子中显示的:

```
SELECT TABLESPACE_NAME, MAX(BYTES) FROM DBA_FREE_SPACE  
GROUP BY TABLESPACE_NAME;
```

TABLESPACE	MAX(BYTES)
RBS	20969472
SYSTEM	4096000
TEMP	10483712
USERS	9439232

观察最后两条SQL语句的输出, 你可以看到USERS表空间实际上把所有的9MB放在一起。注意, 尽管SYSTEM表空间的最大自由区域只有4MB, 然而它总计有11MB的自由空间。

现在你需要看一下表空间中包含的所有段。回忆一下，存储参数 NEXT 一直指示段申请的下一个区域的尺寸。至少你应该弄明白任何一个段至少可以通过一个区域增长。这就把在空间之外运行的危险降到了最低。你会希望看一下 DBA_SEGMENTS 表，因为它考虑了数据库中的所有段，无论它们是什么类型。这个例子列出了数据库中不能增长至少一个区间的所有段（与附加信息）：

```
COLUMN SEGMENT_NAME FORMAT A30
SELECT SEGMENT_NAME,SEGMENT_TYPE,OWNER,A.TABLESPACE_NAME "TABLESPACE_NAME",
INITIAL_EXTENT,NEXT_EXTENT,PCT_INCREASE, B.BYTES "TABLESPACE MAX FREE SPACE"
FROM DBA_SEGMENTS A,
(SELECT TABLESPACE_NAME,MAX(BYTES) "BYTES" FROM DBA_FREE_SPACE
GROUP BY TABLESPACE_NAME) B
WHERE
A.TABLESPACE_NAME = B.TABLESPACE_NAME AND NEXT_EXTENT > B.BYTES;

SEGMENT_NAME
-----
SEGMENT_TYPE      OWNER                TABLESPACE_NAME
-----
INITIAL_EXTENT NEXT_EXTENT PCT_INCREASE TABLESPACE MAX FREE SPACE
-----
TEST1
TABLE              TGASPER              USERS
512000      41943040           50           9439232
```

在这个例子中你可以看到 TGASPER 模式中的 TEST1 表不能分配另一个区间，因为 NEXT_EXTENT 参数被设置为 40MB，而 USERS 表空间的最大自由区间是 9MB。因为 40MB 也比表空间中的总自由空间大，你需要给表空间增加更多的空间以允许 TEST1 增长。即使已经有了合计起来足够的自由空间，你仍然要增加更多的空间以允许 TEST1 增长到 40MB，或者改变 TEST1 的 NEXT_EXTENT 参数以适合最大的自由区间尺寸。

提示 如果表空间达到了一个点，在这个点上最大的自由区间和总自由区间之间相差很远，这就可能是考虑重新组织表空间的时候了。

警告 在设定 PCTINCREASE 参数时会遇到警告。在前面的例子中，你看到 PCTINCREASE 值被设置为 50。这就意味着每个连续的区间比最后一个大 50%。你的段会非常快地有害地尽量分配大区间。合适的规划通常让你把 PCTINCREASE 设置成一个相当低的值。许多地方的策略要求把所有段的 PCTINCREASE 设置为 0。

不管如何设置段的 PCTINCREASE，几乎总是希望为表空间把 PCTINCREASE 缺省值设置为 1。在 Oracle 7.3 及以后的版本中，当这个参数不是 0 时，SMON 自动地合并相邻的自由段。

表空间碎片会变成一个严重的问题，尤其在开发或测试系统中，因为在这里不同的段被频繁地建立和删除。我以前说过，这种活动会由于造成空间气泡而导致表空间碎片，表空间中的空间气泡太小了不能被大多数段使用。

你应该定期评价你的表空间被分裂成碎片的程度。过多的碎片会阻止段增长，因为所有的自由空间被分裂成小得不能使用的碎片。作为一个临时的解决方法，你可以给表空间增加更多的空间。这就打开了一个大的单个区间，这个区间至少允许段增长并使你的数据库能够运行。建议定期消除碎片以优化性能及磁盘空间。

幸运的是，新的 Oracle 版本为数据库管理员提供了一个相当基本然而非常有用的存储管理工具，称为存储管理器。存储管理器是一个基于图形用户界面的工具，它使数据库管理员能

够快速地检查涉及对象和表空间的关键存储问题。

尽管存储管理器不能替代在这儿讨论的所有信息，但它确实提供了一个快速方法用于检查表空间中已分配的和总的自由空间。

启动存储管理器后，通过双击扩展表空间树的项目。你会在右边的窗口中看到所有的表空间以及每个表空间尺寸和自由空间总数。图 21-11显示了此信息的一个例子。

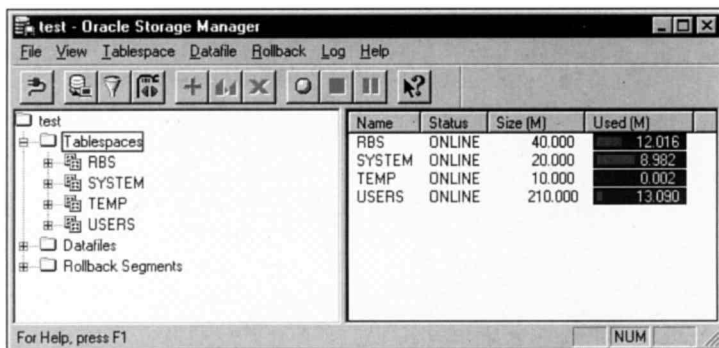


图21-11 存储管理器提供了快速的表空间存储信息

记住，显示的自由空间是表空间中所有自由区间的总数。如果存储管理器指示表空间 USERS 大约有 197MB 的自由空间，并不意味着所有 197MB 的空间是相连的。

21.4.2 表检查

数据库表检查包括估计总的已分配的容量以及实际使用的空间。你也会对表分配的区间数目感兴趣。一两个已用空间与分配空间的比值很低的区间可能暗示着 INITIAL 和 NEXT 区域值可能被设置得太高了。而很多区间可能表明 NEXT 区间参数被设置得太低了。

你可以通过合计所有相关的区域来确定表的总分配空间。你可以在 DBA_EXTENTS 表中找到区间信息。这个例子演示了你如何能够容易地计算出给定的表空间中每个表的分配空间（这个例子中的表空间是 USERS）：

```
SELECT TABLESPACE_NAME, SEGMENT_NAME, SUM(BYTES), COUNT(*) EXT_QUAN
FROM DBA_EXTENTS
```

```
WHERE TABLESPACE_NAME='USERS' AND SEGMENT_TYPE='TABLE'
GROUP BY TABLESPACE_NAME, SEGMENT_NAME;
```

TABLESPACE	SEGMENT_NAME	SUM(BYTES)	EXT_QUAN
USERS	EMP	102400	1
USERS	DEPT	10240	1
USERS	INOUT_LOG	4096000	100
USERS	PHONE_LOG	2048000	1
USERS	HOURS	1024000	2

你可以看到每个表（在 SEGMENT_NAME 列中）分配的空间总数（用字节表示）以及为表分配的区间数量（在 EXT_QUAN 列中）。注意 INOUT_LOG 表已经在 100 个区间中分配了 4MB。这表明存储参数 NEXT 被设置得太低了。

对表的最后评估集中在确定已用空间与分配空间的比值上。回忆一下，Oracle 是根据存储参数 INITIAL 和 NEXT 的值分配空间的。因此，你不能肯定表真正需要分配给它的所有空间。

你所计算的比值仅仅是个估计值，但是它对计算表所使用的实际空间仍然是有用的。

Oracle 通过 ANALYZE TABLE 语句提供了一个计算表统计量的方法。对数据库中的每个

表，你应该定期地计算（或者至少是估计）统计量。使用与分析 PHONE_LOG表的例子相似的命令：

```
ANALYZE TABLE PHONE_LOG COMPUTE STATISTICS;
```

当这条命令运行时，DBA_TABLES视图反映了搜集到的PHONE_LOG表的统计量。在DBA_TABLES中，BLOCKS和EMPTY_BLOCKS列分别指示已经使用的和没被使用的块数目。要知道这些列假定当一个块已经使用了，它就一直被使用。如果一个表包含一百万行，其中700 000行后来被删除了，BLOCKS和EMPTY_BLOCKS值将不会改变。

下面的SQL语句用已分配空间的百分数显示PHONE_LOG表使用的空间：

```
SELECT BLOCKS/(BLOCKS+EMPTY_BLOCKS)*100 USED_PCT FROM DBA_TABLES WHERE
       TABLE_NAME='PHONE_LOG';
USED_PCT
-----
15.397996
```

这里你可以看到尽管PHONE_LOG表已经分到了2MB空间，但只有15%被实际用来存储表。除非你预计这个表在容量上将充分地增长，否则可以考虑删除并重新载入这些表，并只将大约700KB的空间分配给它。

提示 尽管对每个表来说运行一条ANALYZE TABLE命令是必要的，然而生成一个脚本计算整个数据库统计量的工作可以通过使用DBMS_UTILITY.ANALYZE_SCHEMA存储过程而变得更加容易。

21.4.3 优化簇存储

簇检查主要局限于识别过多的区间。为簇优化存储甚至比表更重要，因为它们的效率（或者不足）将不只在表中表现出来。你可以使用下面的SQL语句为每个簇计算区间的数目：

```
COLUMN SEGMENT_NAME FORMAT A20
SELECT OWNER, SEGMENT_NAME, COUNT(*)
       FROM DBA_EXTENTS WHERE SEGMENT_TYPE='CLUSTER'
GROUP BY SEGMENT_NAME, OWNER;
```

COUNT(*)列显示了每个簇的区间数目。理想情况下，你希望每个簇刚好有一个区间。如果超过一打就表明INITIAL和/或者NEXT存储参数值被设置得太低了。

不幸的是，Oracle没有为你提供一种简单的方法确定在一个簇中已使用的块与已分配的块的百分数。尽管这些问题的答案潜藏在x\$表中，仍然建议你不要依赖这些表。

你最好的选择是重新计算最佳的存储参数，就像打算重建一个簇，并把你最近计算的最佳值同该簇当前的设置值相比较。如果这些值不同的话，你最好的选择可能是用正确的存储参数重建簇。要这样做，遵循下面的步骤：

- 1) 执行一个逻辑备份。
 - 2) 删除该簇和相关的表。
 - 3) 用正确的存储参数重新创建已被删除的对象。
 - 4) 从第一步得到的逻辑备份中把行重新载入删除的表中。
- 当这个过程结束后，你就已经准备好允许用户回到数据库系统。

21.4.4 检查索引

同处理簇一样，大部分数据库管理员发现定期的存储完好性检查以验证区间的数目是合

理的。就像对待簇一样，用一条简单的 SQL 语句进行检查：

```
COLUMN SEGMENT_NAME FORMAT A20
SELECT OWNER, SEGMENT_NAME, COUNT(*)
FROM DBA_EXTENTS WHERE SEGMENT_TYPE='INDEX'
GROUP BY SEGMENT_NAME, OWNER;
```

索引可能会对过多的区间数很敏感。对大多数索引而言，应尽力使区间的数目下降到 6 个左右。理想情况当然是仅仅使用一个区域。

21.4.5 观察回滚段的增长

使用回滚段在前面的 21.3 节中已经详细讨论过了。从存储管理的观点出发，你想要了解回滚段增长到多大。回忆一下，通过指定一个存储参数 OPTIMAL 容量，回滚段在事务结束后尺寸会自动缩小。Oracle 保持一个高水位标志，这个标志指示每个回滚段已经增长了多大，自实例启动后它扩充了多少次以及收缩了多少次。使用如下 SQL 语句查看每个回滚段的高水位标志（以字节为单位）：

```
SELECT N.NAME, OPTSIZE, HWMSIZE
FROM V$ROLLNAME N, V$ROLLSTAT S WHERE N.USN=S.USN;
```

对每个回滚段而言，你可以分别在 OPTSIZE 和 HWMSIZE 列中看到优化尺寸和最大尺寸（到目前为止的）。如果 OPTSIZE 是空的，HWMSIZE 就是回滚段目前的容量（没有收缩发生）。

通常，你要为生产时间建立一组回滚段，同时为批量工作建立一组回滚段（可能为特殊工作需要建立一些额外的回滚段）。你会在适当的表空间中一直需要足够的空间以便使用最佳的容量包含所有的回滚段。此外，表空间中必须有足够的自由空间，这样就可以适应每一组回滚段的 HWMSIZE，这些回滚段在任何给定的时间都应该是活动的。

例如，假设你有在生产时间使用的三个回滚段 RP01、RP02 和 RP03，RB01、RB02 在晚上用于批处理。前面 SQL 查询的输出结果是：

NAME	OPTSIZE	HWMSIZE
RP01	10240000	20480000
RP02	10240000	40960000
RP03	10240000	10240000
RB01	102400000	102400000
RB02	51200000	61440000

所有回滚段的总最佳容量大致是 180MB（意思是，这是回滚段的绝对最小尺寸）。假定批量和生产工作从来不同时运行，这样在任何给定的时间，无论是生产还是批量回滚段仍然保持它们的最佳容量。因此，在生产期间，你可以推断回滚段需要的最大空间量是 220MB（合计生产回滚段的 HWMSIZE 列和批量回滚段的 OPTSIZE 列）。在批量时间内，回滚段的最大空间需求是 190MB（从实例最后一次启动开始）。

提示 如果 V\$ROLLSTAT 显示了太多的收缩/扩展，你可能希望调度一个脚本以便提取 V\$ROLLSTAT 的内容来确定回滚段收缩或扩展是否有一个特殊时间。某个工作经常只需要分派给它自己的回滚段就可以被隔离出来。

不要直接采用这些计算。通常应提供大量的裕量空间，这些空间要超过你所计算的迄今为止的最大需求。记住，V\$视图中的值在实例重新启动后总是被复位。

21.4.6 管理临时表空间

监视临时表空间(排序发生的地方)可能是一个复杂的主题。临时表空间是专门满足 Oracle 内部需要的。段在大多数自动安装中按常规方式建立和删除。临时表空间的存储管理问题通常仅限于知道表空间尺寸以及任何时间使用的临时存储的最大值。

临时表空间的尺寸应相对不变,并且不会有太大的起伏——尽管涉及排序的大的查询、大表索引的建立,其他的主要事务会分配异常数量的临时段。

提示 记住, Oracle将多次执行排序作为其他功能的一项必要步骤,例如链接和 GROUP BY子句。不要因为没有明确的排序需求就认为Oracle不用临时表空间。

要监控临时段,可以使用下面的查询检查 DBA_SEGMENTS中类型为TEMPORARY的所有段:

```
SELECT owner, segment_name, tablespace_name, bytes, extents
FROM dba_segments
WHERE segment_type = 'TEMPORARY'
```

总是应寻找在改变日志文件中的错误或者返回到应用程序中的错误,这些错误表明临时表空间太小或者没给事务的完成提供足够的区间。在后面的情况中,解决这个问题通常需要将MAXEXTENTS设置得高一些(或者不受限制的)或者增加区间存储参数 NEXT。

21.5 管理增长的数据库

对数据库的仔细监控通常使你花费一些时间为以后的增长仔细地进行计划。大多数数据库需要性能平衡、容错和易维护。你需要为数据库的增长提供路径,同时维持可接受的性能水平,并确保正常运行时间以及数据库与它的数据的完整性。

数据库增长通常集中在段和表空间的增长。增长的表空间只是给每个需要更多存储的表空间增加一个或者多个数据文件,或者是重新设置已经分配给表空间的数据文件的尺寸的问题。尽管段是自动增长的,但是有一个实际的系统限制,这个限制规定你应该把所有的段数据合并到一个单独的区间中。对合并的需求很大程度上依赖于包含在段中的数据。查找很少增长和很少需要注意的表。相反,包含数据的表可能需要定期地合并,因为它们是定期增长的。

提示 Oracle也可以被配置成允许数据文件自动增长,但是这决不能替代标准的生长监控和管理。除非你确实了解数据库的增长方式,否则自动增长数据文件的使用就会受阻。

在本节中,我谈了围绕增长型数据库环境的维护问题,并且建议执行可能的操作以确保你提供的数据库服务能处理它们的需求。

21.5.1 校正过度的表增长

Oracle RDBMS通过给表段分配额外区间的方法处理常规的表增长。你会不时地发现表增长比预期的要快,这种表可以从完全重组中受益。校正过度扩展的表包括下面的步骤:

- 1) 计算表的总尺寸(把所有区间的尺寸加起来)。
- 2) 对表执行一个逻辑备份。
- 3) 删除该表。

4) 用至少是从步骤1中得出的尺寸的原始区间重建该表。应该使原始区间的尺寸足够大以包含当前的数据并加上额外数量的空间, 以应付 6~12个月的增长。

5) 导入来自第二步的逻辑备份中的表的数据。

这一般称为重组(改组)表。前面的步骤可以用两种方法之一执行——用导出/导入建立逻辑备份, 或者用 CREATE TABLE AS SELECT 语句。对于较小的表, CREATE TABLE AS SELECT 语句执行起来经常又快又容易。

提示 设计定期的数据库改组次数可能对你的组织有帮助。这使你可以重组大大突破设计的存储需求的段。

执行表重组的主要问题是处理依赖于表的外关键字约束条件。Oracle不允许删除那种允许外关键字对表进行存取的表。围绕这个问题有两种方法:

- 删除该表的外关键字, 重组该表, 重建外关键字。

- 禁止所有引用将重组的表的外关键字, 为该表改变区间存储参数 NEXT, 截断该表, 重新载入数据, 重新启用已被禁止的外关键字。

尽管第一种方法工作量较小, 但第二种方法更受欢迎, 因为没有损失完整性定义。它更受欢迎还因为许可及其他的数据库结构没有像表被删除时那样在截断中受损失。即使你禁止了外关键字, Oracle也不允许删除表——这个限制不适用于截断表。确定下一个区间的尺寸, 这样所有的数据库行都填充在这两个区间中并分配足够的空间以满足将来的增长。完成后, 你就有一个含有两个区间的表——一个与它建立时的尺寸相同的原始区间以及另一个包含剩下的表数据和增长空间的区间。

21.5.2 合并簇

因为簇和表在它们的存储性质方面是相似的, 把它们各自的数据合并到一个单独区域中的过程稍微有所不同。按照下面的步骤合并簇:

- 1) 计算簇本身以及相关的表的总分配容量。该簇的总容量应该大致和成员表的累计总容量相匹配。

- 2) 执行该簇的逻辑备份(包含所有相关的表)。

- 3) 删除该簇和相关的表。

- 4) 首先重建该表, 之后以原来的区间尺寸重建该簇, 这个区间要足够包含每个段的所有数据加上足够的能提供6~12个月增长的剩余空间。

- 5) 重新载入来自第二步的逻辑备份中的簇。

簇与表以及前面提到过的适用于簇的原则在性质上是类似的。

21.5.3 合并索引

索引非常容易合并到一个单独的区域中。因为没有实际数据一定要首先存储、之后重新载入, 所以合并索引只有三步:

- 1) 计算索引需要的总数。

- 2) 删除该索引。

- 3) 用最初的区间重建该索引, 这个区间要足够大, 能包含所有索引的数据以及足够的额

外空间容纳6~12个月的增长。

换句话说，就是用 ALTER INDEX REBUILD 命令重建一个有新的存储参数的索引。这对非常大的索引，或者不能被轻易删除的主关键字索引特别有帮助。务必观察 UNRECOVERABLE 和 PARALLEL 子句以加速索引的重建时间。

警告 ALTER INDEX REBUILD 建立一个已存在索引的新拷贝，并且在新索引被完全建立以前不删除原来的索引。尽管它有索引构造时间方面的好处并且把用户冲突减到最小，但是它需要有足够的自由表空间供给原来的索引和新的索引。而且，过多地使用它会严重地分裂索引表空间。

21.5.4 管理表空间的增长

处理表空间的增长，过去常通过使用 ALTER TABLESPACE ADD DATAFILE 命令把一个数据文件增加到表空间中来解决。尽管相当常规，但这是自动或脚本化的方法都不容易实现的过程，因此需要数据库管理员全神贯注地执行。尽管在按计划进行存储升级时，这不是问题；但当生产系统用完临时空间而数据库管理员不在现场时，这就是个问题。

很幸运，Oracle在7.3版本中提出了这个问题并超出了数据库的范围。现在有了两个重要特性，用这两个特性可以帮助你管理表空间和数据文件——可重新调整的数据文件和自动数据文件扩展。尽管它们不是严密的空间监控和计划例程的代替品，但这些新特性在任何 Oracle 数据库管理员的“武器库”中都占有很重要的位置。

要重新调整数据文件，可发出下面的命令：

```
alter database datafile [datafile name] resize [new_size];
```

警告 许多操作系统和相关的工具（例如备份软件）在文件尺寸上有限制。在重新调整数据文件或开启Oracle的数据文件自动扩展特性前，必须熟悉你的操作系统的限制。

例如，下面的语句把 user_data 表空间中数据文件的尺寸重新调整到 20MB：

```
alter database datafile '/u01/oradata/PROD/user_PROD_01.dbf' resize 20M;
```

你可以把数据文件的尺寸重新调整到同存储数据文件的操作系统卷所能处理的一样大。当重新调整表空间的尺寸以收回失去的空间时，表空间只能被调整到大于表空间的高水位标志的尺寸。考虑下面的表空间情况：

Table1	5MB
自由空间	10MB
Table2	15MB
自由空间	125MB
Table3	15MB
自由空间	100MB

在这个例子中，在表空间中有三个表，加起来共 35MB。表空间本身的尺寸是 265MB，但是你能收回的唯一的空间是最后的 100MB。

使用自动增长的数据文件，在数据文件装满容量时，Oracle自动地分配预定数量的空间给该数据文件。命令的语法如下所示：

```
ALTER DATABASE DATAFILE [file_spec]  
AUTOEXTEND ON NEXT [increment_size] MAXSIZE [max_size,UNLIMITED];
```

increment_size 定义了当数据文件装满时增加多少空间给它。MAXSIZE 设置数据文件的最

大尺寸。这条命令的例子如下所示：

```
ALTER DATABASE DATAFILE '/u02/oradata/DEVL/temp_DEVL_01.dbf'  
AUTOEXTEND ON NEXT 10M MAXSIZE 500M;
```

提示 当建立表空间时，也可以通过在数据文件说明后立即放置AUTOEXTEND参数和值的方法使能AUTOEXTEND。

AUTOEXTEND对临时段和回滚段表空间特别有用。因为没有更多的空间供临时段或回滚段扩展，因此它在避免事务失败方面是个强大的工具。但是，要提防 **UNLIMITED MAXSIZE**——因为一个在开始分配你所有的临时空间时就应该被删除的失控的查询，你可能发现你的临时表空间数据文件充满了整个操作系统卷。

OS文件系统碎片

Oracle数据库管理员习惯于处理数据库碎片，但是经常忘记操作系统碎片的含义。大多数Oracle站用数据文件建立表空间，而数据文件存储在属于操作系统的文件系统中。文件系统会快速地并很容易地使大部分自由空间严重地分裂成碎片。当Oracle试图建立一个有几兆字节的文件时，它通常依赖于操作系统实际分配物理设备中需要的空间。如果操作系统不能发现连续的自由空间，它将分配多个自由空间，直到凑出足够用于保存数据文件的集合空间。因此，在这个新数据文件中存取数据需要更多的读/写磁头运动。

从性能的观点出发，如果数据文件被操作系统分散在整个磁盘驱动器上，那么保持低的Oracle段碎片是绝对不好的。整理文件系统碎片的方法在不同的操作系统中是不一样的。与许多UNIX工具不一样，Windows NT可以在数据库和系统启动和运行时整理文件系统的碎片。几个正在出售的第三方产品将执行这个功能。在运行时整理碎片应该在最少的数据库活动期间运行。要有一个最近的备份以对灾难有所准备。

21.6 故障检查

21.6.1 区间限制

Oracle可能不能扩展段，这归因于段的表空间中连续空间不足或者是区间太多。记住，段中的MAXEXTENTS参数可以限制允许的区间数量，你可能已经达到了Oracle的内部区间限制（由数据库块容量确定）。如果问题是由于过多数量的区间引起的，你大概应该集中精力在减少区域数量而不是在段上增大区域限制。

21.6.2 临时表碎片

应用程序的开发者坚持认为，批处理程序建立和删除临时表以执行必要的功能是必要的。但这个行为会导致一些表空间会迅速分裂成严重的碎片。

尽量准确地确定哪个程序将建立临时段并改变它们，使它们在一个或更多的专用于临时段的表空间中建立段（临时段应该把所有缺省的PCTINCREASE设置为1或更高）。在批处理循环的结尾，所有的临时段应该已经被临时表空间清除了，并且SMON将把目前所有的自由区间合并到一个大的自由区间中。

21.7 方案：监控数据库的增长

还记得你的数据库第一次是什么时候建立的么？你恰好已建立每件事情；空间被最优地

分配了，没有明显的数据库碎片的提示，并且你的查询几乎立刻就能运行。几年（也许几个月）以后，由数据文件组成的表空间放在任何你有空间的地方，自由磁盘存储得紧紧的，性能几乎不能接受；你必须整理这些混乱的情况。

如果剩下的未经检查，数据库很容易会失控增长。在这部分中，我将讨论用来处理数据库并监控它增长的技巧。有了这些技巧，你便能够计划磁盘的分配和数据库的增长——在用户打电话让你知道他们“在表空间 PAYROLL_DATA_01 中不能分配 64 字节”之前。我还会给出一些有用的调整信息，这些信息会帮助你保持数据库正常而且能够做出响应。

本书附带的 CD-ROM 包括几个不同的文件，这些文件组成了一个完整的数据库容量跟踪系统。通过定期（理想的情况是每天）运行 GETSTATS.SQL 程序（可以在 CD-ROM 中找到），你就会搜集到有关数据库中的每个段和表空间的详细历史报告以及趋势信息。优化软件系统以使存储这个历史信息的总开销最小。在存储统计数字时它只存储变化。例如，一个段的存储信息只有在它分配了另一个区间时才被记录。当然这是以运行的性能为代价的，几小时后可以由调度查询跟踪系统，把对你的系统有影响的性能减到最小。

在运行 GETSTATS.SQL 搜集统计数字前，你需要建立知识库表并支持视图和存储过程。要做到这一点，应以一个有数据库管理员特权的用户身份在数据库上登录（这个计划将被用于保存知识库对象）并运行下面的 SQL 脚本：

1. CREATE_TABLES.SQL
2. CREATE_FUNCTIONS.SQL
3. CREATE_VIEWS.SQL

跟踪系统中包含的所有信息存储在 SPACEHIST 和 PROCDATES 表中。SPACEHIST 包含了你的数据库中段的历史尺寸，而 PROCDATES 每天都为存在统计数字进行存储。由于 SPACEHIST 只记录变化，你需要用一个视图搞清信息的含义。在你运行 GETSTATS.SQL 的时间里，能够使用视图 V_SPACEHIST 查看你的跟踪表中所包含的数据。

V_SPACEHIST 在 PROCDATES 中的每个日期上为每个段都包含一行。V_SPACEHIST 视图被定义成如下形式：

SAMPLE_DATE	DATE
OWNER	VARCHAR2(20)
SEGMENT_TYPE	VARCHAR2(20)
SEGMENT_NAME	VARCHAR2(20)
TABLESPACE	VARCHAR2(20)
BYTES	INTEGER
EXTENTS	INTEGER
NEXT_EXTENT	INTEGER

下面描述 V_SPACEHIST 的列：

- SAMPLE_DATE 包含段的统计量生效的日期。
- INSTANCE 是包含这个段的数据库的名字。
- OWNER 是段的拥有者。
- SEGMENT_TYPE 定义被检查的段的类型。
- SEGMENT_NAME 指定段的名字。
- TABLESPACE 指示包含这个段的表空间的名字。
- BYTES 是分配给这个段的字节数。这个值是通过合计分配给该段的所有区间的字节计算出来的。

■ EXTENTS是用于段的区间数。

■ NEXT_EXTENT是NEXT_EXTENT参数的值。

如果你不用 WHERE 子句查询 V_SPACEHIST，就会为你跟踪统计数字的每个段返回行，每天这些统计数字都被搜集。看一下这个视图中包含的所有信息，它们可能不太有用。幸运的是，你可以用 SQL 的强大查询功能搜集需要的信息。

提示 还有许多来自诸如 BMC 和 Platinum 的厂家的跟踪数据库增长信息的总控方案。

“一幅画相当于一千个词”精确地说明了数据的图示法比列表更有意义。例如，如果你希望了解在最后 6 个月中你的数据库何时从 2.7GB 增长到 6GB，你可能用 SQL 查询检查几百行，而查看一张显示相同信息的图，立即就能知道最大增长发生的时刻。

ODBC 连接和电子表格软件是用于建立简洁然而含有丰富信息的图的全部工具。如你在图 21-12 看到的，Microsoft Excel 电子表格图总体上显示了数据库增长的一段时期。有了这个信息，你可能希望使用 V_SPACEHIST 视图向下检查段级别数据。通过把数据用图表示，你立即便能确定需要把调查集中在哪段时间上。

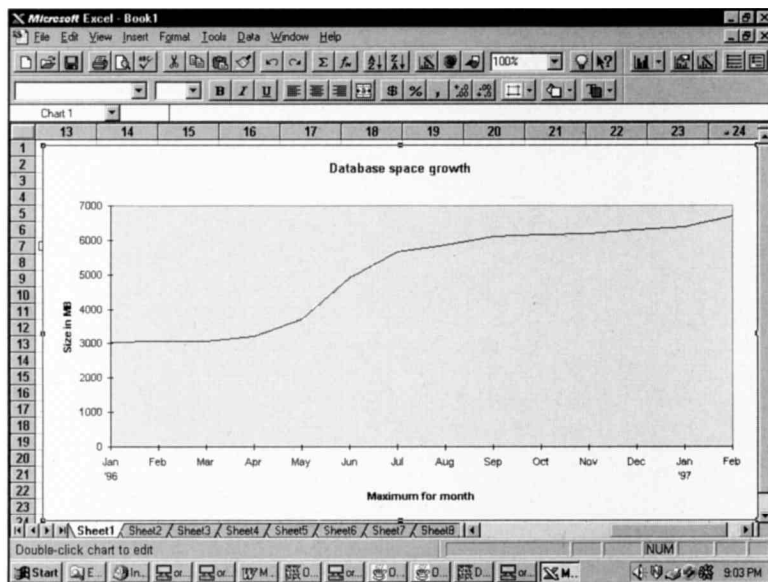


图21-12 用图表示来自V_SPACEHIST视图的信息可以快速确定峰值增长时间

数据库增长的突然改变经常是由于用户数量的增加或应用程序的改变，这会导致大量数据被存储。当一个数据库被建立后，数据库管理员经常会忘记查看信息，你的数据库容量可能在你不知道的情况下迅速增长。通过定期地搜集存储统计量，你可以了解什么时候发生无规律地增长，并有机会在发生严重问题前采取预防措施。

假设，你每天晚上运行 GETSTATS.SQL，并且每星期你都看一次显示你的成品数据库在最后 6 个月中的增长的图。你要用如下的 SQL 语句开始：

```
SELECT sample_date, SUM(bytes)
  FROM v_spacehist WHERE sample_date > (SYSDATE) - 180
GROUP BY sample_date;
```

因为 V_SPACEHIST 视图包含了它所跟踪的每个段的 OWNER 和 TABLESPACE 信息，所以你可以缩小焦点来跟踪特定的用户或表空间的增长。这个例子提供了用户 TFASPER 拥有的所

有段在90天内的增长信息。

```
SELECT sample_date, SUM(bytes) FROM v_spacehist
WHERE owner = 'TGASPER' AND sample_date > (SYSDATE - 90)
GROUP BY sample_date;
```

监控特定的表空间的增长一样容易实现，如同下面这个例子中所显示的那样：

```
SELECT sample_date, SUM(bytes) FROM v_spacehist
WHERE tablespace = 'USERS' AND sample_date > (SYSDATE - 180)
GROUP BY sample_date;
```

除非你处于一个独特的环境中，否则随着时间的推移你的数据库将会增长。当最初设计系统时，你要为将来预期的增长做出计划。对数据库执行定期的监控有助于确保你不会放松对不可预计的增长的警惕。

第22章 鉴别资源密集型用户

本章要点：

不同的资源

CPU

文件输入/输出

内存

每一个数据库管理员都面对过这样一种情形，即数据库开始拖延并由于某种原因速度突然变慢。这种现象能够发生在一个已被很好调整过的数据库中，它会在不知不觉中影响你。这时数据库管理员需要用技巧来辨认资源密集型用户并做一些必要的影响数据库性能的事情。我曾多次查明用光全部系统资源并引起数据库整体性能下降的会话。资源密集型用户可能是一个执行大型查询的用户，其中该查询拥有很多表并在 SELECT子句中遗漏了几个键列连结，或者它可以是一个未优化的语句。本章的目的就是要使数据库管理员具备必要的技巧以便相对容易地鉴别这样的用户。本章提供鉴别资源密集型用户的方法和技术，但没有详细地讨论纠正问题的细节。关于如何解决它们的细节将在本书中其他有关性能调整的章节中讨论。

22.1 不同的资源

你可以采用许多方法鉴别资源密集型用户。下面这个方法首先对环境进行了概述。假如觉察到出错的地方，那么应进一步地在该区域中调查以便对用户进行详细描述。你要查看的资源是CPU、文件输入/输出和内存。假如CPU被严重加载并且完全被某个执行大量的CPU密集型工作的用户占用的话，那么该用户会降低系统速度。类似，频繁地文件输入/输出也能够引起系统性能下降。某些进程对内存的频繁请求会导致内存分页与交换，这将严重阻碍系统性能。你应把注意力集中在这三种资源上。基本上，在服务器上执行的每一个进程试图获取如下资源：

■ CPU —— 每一个进程都需要一些CPU时间片以执行它的工作。有些进程夺取了大量CPU时间，有些进程完成时间较早。可以使用的有限CPU能力必须在系统上的Oracle进程与内部操作系统（OS）进程之间共享。显而易见，CPU密集型用户是指消耗大量CPU时间的用户。

■ 文件输入/输出 —— 每当一个进程需要存取数据时，如果该数据已经由以前的某个进程带入的话，那么该进程就首先查找缓存。如果数据已经存在于该缓存中，则读取很快便完成。如果需要的数据不在缓存中，那么就必须从磁盘上读取该数据。这时必须执行一个物理读取。这种从磁盘中读取数据的操作非常耗时，因为从磁盘中读数据所用的时间几乎是从高速缓存中读数据所用时间的50倍。当系统开始密集地进行输入/输出时，系统性能开始下降，因为所有进程在大部分时间里所做的工作就是等待将从磁盘中返回的数据。

我正在讨论的资源是时间。由于较繁忙的磁盘文件存取而导致大量的时间被消耗，所以

时间显得越来越珍贵。最终，所有资源的使用将归结为一个资源——时间。

■ 内存 —— 由于某些进程对内存的大量使用而导致内存不足也可能造成性能下降。在UNIX操作系统中，利用虚拟内存的原理可以有效地使用内存。在一个虚拟内存系统中，仅仅最常使用的页面才被保留在物理内存中；其余的或当前非活动页面被保存在交换空间中，在需要时才被引入物理内存中。当由于物理内存较少或进程对内存的频繁使用而使得系统中的内存短缺时，为满足对内存的需要，系统首先开始分页。假如内存仍然短缺，系统便求助于交换物理进程，实际上这样做会降低系统性能。分页与交换将在稍后的22.4节中讨论。

当大量使用这些资源时，系统会经历性能的下降。因此，给每一个用户分配多少资源非常重要。要鉴别严重资源消耗，你应每次把注意力集中在一种资源上并设法鉴别由进程消耗的个别资源。

Windows NT性能监控

用于Windows NT的Oracle 8i包含如下三种性能监控工具：

- Oracle 8数据字典视图。
- Windows NT性能监控程序。
- Oracle 8性能监控程序。

1. Oracle 8数据字典视图

有多组含有大量的有关数据库性能信息的数据字典视图。有两个常见的数据字典视图类别：静态（static）与动态（dynamic）。静态视图包含有关数据库整体结构的信息，例如名称、数据库文件的位置、用户信息、存储器使用率、重做日志和控制文件信息等。动态视图（也称为V\$表）反映了诸如物理输入/输出、内存使用率、用户活动等实时性能信息。

2. Windows NT性能监控程序

Windows NT可以调整的参数不多。例如，在UNIX操作系统中，系统管理员可以修改大量的直接影响系统性能的核心参数。相反，Windows NT根本不能接触到核心参数。Windows NT被认为是自调整的，因为它包含了许多自适应算法，这些算法在系统运行时可以动态地调整性能。

Windows NT操作系统的大部分性能信息可以在Windows NT性能监控程序（PerfMon）中找到。Windows NT性能监控程序是一个灵活的图形工具，你可以使用它来收集有关Windows NT的实时性能信息。它也为被选取的每一个计数器产生一个数值。这些值显示在图的低端。你可以使用该信息确定系统中正经历性能瓶颈的部分并采取相应的步骤消除这些问题。

Windows NT几乎把整个操作系统中的每一个事物作为一个对象对待。

3. Oracle 8性能监控程序

Oracle 8性能监控程序是一系列定制的Windows NT性能监控程序计数器，它们能够用来测量Oracle数据库的性能。此性能监控工具采用数据字典视图的信息并以图形方式显示该信息。计数器全部基于依靠数据字典发出的SQL语句。你可以在PERF 81.ORA文件中看到由Oracle 8 PerfMon发出的SQL语句，PERF 81.ORA文件位于ORANT\DBS目录中。当你单击Oracle 8性能监控程序图标时便自动建立一个到数据库的连接。用于此连接的用户名、口令和其他信息位于Windows NT注册表中。要查看此信息，运行注册表编辑程序（REGEDT32.EXE）。打开HKEY_LOCAL_MACHINE键，然后定位到SYSTEM\CurrentControlSet\Service\Oracle80\Performance。

通过双击用户名、口令或主机名来相应地改变它们的值（参见图 22-1）。缺省设置是 system/manager@orcl。



图22-1 用于Oracle 8性能监控工具的注册项

用于Oracle 8的性能监控程序计数器显示于表 22-1中。

表22-1 性能监控程序计数器

对 象	计 数 器
Oracle 8缓存	%物理读/获取的次数
Oracle 8字典缓存	%遗漏/获取的次数
Oracle 8数据文件	物理读/秒的次数；物理写/秒的次数 (每一个实例是一个单独的数据文件)
Oracle 8 DBWR stats 1	已搜索的缓冲区/秒；LRU搜索/秒
Oracle 8 DBWR stats 2	检查点/秒；停工时间/秒
Oracle 8动态空间管理	循环调用次数/每秒
Oracle 8自由表	%自由表等待次数/请求
Oracle 8库缓存	%重新加载/针
Oracle 8重做日志缓冲区	重做日志空间请求
Oracle 8排序	内存中的排序数/秒；磁盘上的排序数/秒

22.2 CPU

UNIX系统是一个多进程操作系统；也就是说，UNIX操作系统能够同时管理多个进程。每一个必须执行一个作业的进程在运行队列中等待得到CPU时间片来执行它的作业。当轮到某一个进程时，该进程便被分配一个时间片，它在该时间片中执行它的作业。一个经过良好调整的CPU在执行进程的运行队列中不会有很多的进程等待，但是它应充分忙碌。22.2.2节将鉴别那些消耗过多CPU资源的进程。当此练习时，记住，在一个较轻负载的系统中，一个进程可以独占整个CPU，这很普通并且不会成为惊慌的理由。当你试图鉴别CPU密集型用户时，你务必保证在任何时间点上系统中有足够数量的作业正在运行并且存在性能问题。仅在

这种情况下你才可以采用这里提到的方法，因为在低负载的系统中，少量的用户消耗了大量的CPU时间,这确实很普通。

22.2.1 CPU预览

在你鉴别正引起CPU资源问题的用户以前，你必须大致了解一下CPU是如何运转的。在UNIX系统中，你可以使用sar -u命令确定CPU的工作情况。sar -u命令有4列输出，见表22-2。

表22-2 sar -u命令输出中的重要列

列名	描 述
usr	花费在为用户请求服务上的CPU时间的百分数
sys	CPU参与系统调用所花费的时间百分数
wio	CPU等待从磁盘中输入/输出的完成所花费的时间百分数。假如该数值较高的话，表明磁盘中存在瓶颈或输入/输出系统的效率低下
Idle	空闲CPU时间的百分数

一个正常的或经过良好调整的CPU通常使用户CPU时间是系统CPU时间的两倍；也就是说，同服务于系统调用相比，CPU花费更多时间服务于用户请求。

sar命令的示例输出稍后显示。基本的命令格式如下：

```
sar -u n t
```

n是执行监控的时间间隔，t是执行监控的次数。下面是一条sar CPU Utilization命令的输出示例：

```
$ sar -uM 2 10
HP-UX arms1 B.10.10 U 9000/819 09/04/97
19:53:57      cpu      %usr      %sys      %wio      %idle
19:53:59          0          38           9          40          12
              1          43          10          37          10
      system          40          10          39          11
19:54:01          0          51          14          29           6
              1          45          17          33           6
      system          48          15          31           6
19:54:03          0          42           8          44           6
              1          44           6          42           8
      system          43           7          43           7
19:54:05          0          52          10          34           4
              1          34          17          42           6
      system          43          14          38           6
19:54:07          0          30          14          45          11
              1          37          10          46           7
      system          34          12          46           9
19:54:09          0          44          10          38           8
              1          44           6          43           6
```

假如CPU含有大量的空闲时间，这可能意味着CPU没有被最充分地利用。假如%wio、%usr和%sys列的数值较低且%idle列的数值较高，这就意味着当前在系统中没有东西正在运行。

假如CPU花费的系统时间高于用户时间，这也需要进行检查。你旨在使用户CPU时间成倍于或超过系统CPU时间；否则，你需要进一步的检查。在前面的输出中，%wio列具有较高的数值；这意味着在当前的方案中，CPU没有出现问题，但是在磁盘上可能存在瓶颈。你需要进一步执行sar -d命令以便发现哪一张磁盘正面临输入/输出负载。

正如较早前所提及的那样，每一个必须在系统中执行的进程要在运行队列中等待。假如有太多的作业在运行队列中等待，这表明系统中有大量的资源密集型作业正在运行，此时CPU不能处理系统的请求。下面是运行队列的一个评定示例，同时也是一个查明CPU是否由于系统中运行的进程而停顿的好起点：

```
$ sar -qu 2 10

HP-UX arms1 B.10.10 U 9000/819    09/06/97

17:35:37 runq-sz %runocc swpq-sz %swpocc
          %usr   %sys   %wio   %idle
17:35:39      1.0    25      0.0      0
          39      11      43      6
17:35:41      0.0     0      0.0      0
          46     10      40      5
17:35:43      1.0    25      0.0      0
          17      5      55     24
17:35:45      1.0    25      0.0      0
          38      9      42     10
17:35:47      1.0    25      0.0      0
          39      9      45      8
17:35:49      0.0     0      0.0      0
          36      6      52      6
17:35:51      1.0    25      0.0      0
          22     10      51     16
17:35:53      1.0    25      0.0      0
          44      7      46      3
17:35:55      2.0    25      0.0      0
          41      8      45      6
17:35:57      0.0     0      0.0      0
          25      8      50     17

Average      1.1     18      0.0      0
Average      35      8      47     10
```

表22-3解释输出中的列的含义。

表22-3 对sar -qu输出中的列的解释

列 名	列 描 述
runq-sz	运行队列的大小。它不包括正在睡眠或等待输入/输出完成的进程；它包括存在于内存中并且正在等待运行的进程
%runocc	等待执行的进程占用的运行队列时间百分率
swpq-sz	在执行监控的间隔期内交换队列的平均长度。准备运行但已经被换出的进程包含在此数量中
%swpocc	能够运行的进程（已换出但准备运行的进程）占用的交换队列的时间百分率

从上面的输出中，你可以看到运行队列的容量仅为1；这意味着在监控间隔期内，队列中仅有一个进程正在等待CPU。在间隔期内，占用的运行队列百分率仅为整个时间的25%。你可以推断出CPU在此时刻负载较轻。一般要注意一个超过5或6的运行队列。假如该队列大于这些值，要么减少运行进程的数量、增加CPU的数量，要么升级已有的CPU。你可以鉴别正等待CPU资源的进程或使用下一节提到的技术加载CPU。

22.2.2 寻找CPU密集型用户

有几种方法用来寻找系统中的CPU密集型用户。你既可以使用操作系统工具，也可以使用

用存储在Oracle系统表中的信息来鉴别CPU密集型用户。在本节中，两种方法你都要检验一下。事实上，这两种方法都可以用来寻找系统中的CPU密集型用户。

1. 使用Oracle途径

Oracle动态系统性能表提供了有关系统中正在发生的事件的大量信息。V\$SYSSTAT与V\$SESSTAT视图是两个动态性能视图，可以使用它们来查找需要的信息。

存储于V\$SYSSTAT视图中的数值的通用列表在下面的清单 22-1中给出。你感兴趣的信息是CPU的使用情况，但是该视图可以用于许多其他的监控目的。

```
SQL> select * from v$sysstat order by class,ststistic#;
```

清单22-1是上述SQL语句的输出结果。

清单22-1 使用V\$SYSSTAT视图查找系统统计信息

STATISTIC#	NAME	CLASS	VALUE
0	logons cumulative	1	2051
1	logons current	1	68
2	opened cursors cumulative	1	72563
3	opened cursors current	1	636
4	user commits	1	289212
5	user rollbacks	1	7299
6	user calls	1	2574300
7	recursive calls	1	4726090
8	recursive cpu usage	1	1334927
9	session logical reads	1	205058382
10	session stored procedure space	1	0
12	CPU used by this session	1	4896925
13	session connect time	1	6.2794E+11
15	session uga memory	1	8599758248
16	session uga memory max	1	169781672
20	session pga memory	1	311833920
21	session pga memory max	1	324695784
101	serializable aborts	1	0
133	bytes sent via SQL*Net to client	1	206511175
134	bytes received via SQL*Net from client	1	174496695
135	SQL*Net roundtrips to/from client	1	2588500
136	bytes sent via SQL*Net to dblink	1	0
137	bytes received via SQL*Net from dblink	1	0
138	SQL*Net roundtrips to/from dblink	1	0
84	redo entries	2	2040536
85	redo size	2	502002531
86	redo entries linearized	2	0
87	redo buffer allocation retries	2	2391
88	redo small copies	2	1807035
89	redo wastage	2	76782505
90	redo writer latching time	2	888
91	redo writes	2	183133
92	redo blocks written	2	576689
93	redo write time	2	502618
94	redo log space requests	2	66
95	redo log space wait time	2	2635
96	redo log switch interrupts	2	0
97	redo ordering marks	2	0
22	enqueue timeouts	4	25
23	enqueue waits	4	57
24	enqueue deadlocks	4	0
25	enqueue requests	4	707487
26	enqueue conversions	4	7259
27	enqueue releases	4	707376

37 db block gets	8	5809539
38 consistent gets	8	202261999
39 physical reads	8	124879028
40 physical writes	8	367214
41 write requests	8	25221
42 summed dirty queue length	8	37479
43 db block changes	8	3994241
44 change write time	8	56399
45 consistent changes	8	822006
46 redo synch writes	8	23407
47 redo synch time	8	103518
48 exchange deadlocks	8	0
49 free buffer requested	8	123453575
50 dirty buffers inspected	8	42044
51 free buffer inspected	8	86462
52 commit cleanout failure: write disabled	8	0
53 commit cleanout failures: hot backup in progress	8	0
54 commit cleanout failures: buffer being written	8	164
55 commit cleanout failures: callback failure	8	2769
56 total number commit cleanout calls	8	547911
57 commit cleanout number successfully completed	8	538392
58 DBWR timeouts	8	10503
59 DBWR make free requests	8	38615
60 DBWR free buffers found	8	18186694
61 DBWR lru scans	8	42222
62 DBWR summed scan depth	8	20800400
63 DBWR buffers scanned	8	20699788
64 DBWR checkpoints	8	998
70 recovery blocks read	8	0
71 recovery array reads	8	0
72 recovery array read time	8	0
73 CR blocks created	8	125716
74 Current blocks converted for CR	8	14791
99 background checkpoints started	8	18
100 background checkpoints completed	8	18
28 global lock gets (non async)	32	1
29 global lock gets (async)	32	0
30 global lock get time	32	0
31 global lock converts (non async)	32	0
32 global lock converts (async)	32	0
33 global lock convert time	32	0
34 global lock releases (non async)	32	0
35 global lock releases (async)	32	0
36 global lock release time	32	0
78 next scns gotten without going to DLM	32	0
79 Unnecessary process cleanup for SCN batching	32	0
80 calls to get snapshot scn: kcmgss	32	2619387
81 kcmgss waited for batching	32	0
82 kcmgss read scn without going to DLM	32	0
83 kcmccs called get current scn	32	0
65 DBWR cross instance writes	40	0
66 remote instance undo block writes	40	0
67 remote instance undo header writes	40	0

68 remote instance undo requests	40	0
69 cross instance CR read	40	0
98 hash latch wait gets	40	0
118 table scans (short tables)	64	366085
119 table scans (long tables)	64	10819
120 table scans (rowid ranges)	64	15
121 table scans (cache partitions)	64	0
122 table scans (direct read)	64	15
123 table scan rows gotten	64	904956876
124 table scan blocks gotten	64	129018277
125 table fetch by rowid	64	31893640
126 table fetch continued row	64	506029
127 cluster key scans	64	135034
128 cluster key scan block gets	64	363979
129 parse time cpu	64	34740
130 parse time elapsed	64	55148
131 parse count	64	532516
132 execute count	64	2390178
139 sorts (memory)	64	40412
140 sorts (disk)	64	268
141 sorts (rows)	64	49141580
142 session cursor cache hits	64	0
143 session cursor cache count	64	0
11 CPU used when call started	128	4896911
14 process last non-idle time	128	6.2794E+11
17 messages sent	128	160593
18 messages received	128	160593
19 background timeouts	128	31883
75 calls to kcmgcs	128	90759
76 calls to kcmgrs	128	4168092
77 calls to kcmgas	128	306090
102 transaction lock foreground requests	128	0
103 transaction lock foreground wait time	128	0
104 transaction lock background gets	128	0
105 transaction lock background get time	128	0
106 transaction tables consistent reads - undo records applied	128	143669
107 transaction tables consistent read rollback	128	131
acks		
108 data blocks consistent reads - undo records applied	128	678275
109 no work - consistent read gets	128	194428691
110 cleanouts only - consistent read gets	128	25194
111 rollbacks only - consistent read gets	128	86452
112 cleanouts and rollbacks - consistent read gets	128	54180
113 rollback changes - undo records applied	128	14117
114 transaction rollbacks	128	1520
115 immediate (CURRENT) block cleanout applications	128	73789
116 immediate (CR) block cleanout applications	128	79374
117 deferred (CURRENT) block cleanout applications	128	264576
144 cursor authentications	128	95248

注意 当你使用该视图时要注意。由于在某些Oracle版本中存在程序错误，所以该视图中的有些统计值非常大，这些统计值或许会显示错误的数值。

你所感兴趣的统计量是CPU used by this session，它的statistic#值为12：

```
12 CPU used by this session          1      4896925
SQL> select * from v$sysstat WHERE name = 'CPU used by this session' order by
class,statistic#
```

现在CPU used by this session的值是在百分之一秒内。通过修改如下的查询把该值转化成分钟：

```
col name format a35
col value format 999.99 heading "Time in | Mins"
select statistic#,name,class ,value/60/100 value
from v$sysstat
where statistic# = 12
/
```

如下的列表是上述SQL语句的输出值。

STATISTIC#	NAME	CLASS	Mins
12	CPU used by this session	1	817.56

该输出表明RDBMS与它所有的后台和前台服务器进程自启动以来已经消耗了大约 817分钟的CPU时间。这是又一条总体水平上的信息。你现在想要查找某一个会话所消耗的 CPU资源的数量。

还有一个叫做 V\$SESSTAT的动态性能视图。基本上，这两个视图之间唯一的区别就是 V\$SYSSTAT存储摘要级的信息，而 V\$SESSTAT存储会话级的信息；V\$SESSTAT可以被称为 V\$SYSSTAT的子表。你可以使用 V\$SESSTAT中的信息来发现哪一个会话消耗 CPU的时间最多。

你通过使用清单22-2中的脚本执行此操作。

清单22-2 产生CPU使用报表的脚本和输出值

```
col prog format a10
col value format 9999.99 heading "Time In| Mins"
```

```
Select a.sid,
        spid,
        status,
        substr(a.program,1,10) prog,
        a.terminal,
        osuser,
        value/60/100 value
From    v$session a,
        v$process b,
        v$sesstat c
Where    c.statistic# = 12
And      c.sid       = a.sid
And      a.paddr     = b.addr
Order by value desc;
```

The following is the output of the above SQL statement.

Time In	SID	SPID	STATUS	PROG	TERMINAL	Schema	Mins
	45	11145	INACTIVE	C:\ARMS\BS Windows	PC	TIFFANY	95.87
	95	12370	INACTIVE	C:\WINDOWS Windows	PC	SUSANE	22.99
	47	9778	INACTIVE	C:\ARMS\BS Windows	PC	LIN	10.22

22	9295	ACTIVE	C:\ARMS\DE	Windows	PC	INGEMAR	3.96
37	14427	INACTIVE	C:\ARMS\CS	Windows	PC	SHARON	.50
107	9454	INACTIVE	sqlplus@ar	ttyp2		MARK	.34
65	16959	INACTIVE	C:\ARMS\CS	Windows	PC	APEKSH	.32
35	17077	INACTIVE	C:\ARMS\CS	Windows	PC	TEUTA	.30
34	18995	INACTIVE	C:\ARMS\CS	Windows	PC	KASTURI	.23
23	578	INACTIVE	C:\ARMS\CS	Windows	PC	REEHER	.20
33	16965	INACTIVE	C:\ARMS\CS	Windows	PC	JULIE	.20
71	16973	INACTIVE	C:\ARMS\CS	Windows	PC	AMY	.19
92	16989	INACTIVE	C:\ARMS\CS	Windows	PC	BARRETT	.17
82	8004	INACTIVE	C:\ARMS\CS	Windows	PC	GABRIELLE	.17
96	17090	INACTIVE	C:\ARMS\CS	Windows	PC	MANISH	.16
20	5466	INACTIVE	C:\ARMS\CS	Windows	PC	BURTNER	.16
29	17033	INACTIVE	C:\ARMS\CS	Windows	PC	DAVID	.14
43	16953	INACTIVE	C:\ARMS\CS	Windows	PC	ADITI	.13
77	16947	INACTIVE	C:\ARMS\CS	Windows	PC	GARY	.12
54	8971	INACTIVE	C:\ARMS\CS	Windows	PC	RANDY	.11

从清单 22-2 中，你可以推断出会话 45 与 95 是系统中的顶端 CPU 用户，它们消耗 CPU 的时间分别为 95 分钟和 22 分钟。此输出存在的唯一问题是它反映了自会话开始以来的累积统计值，而不是一个时间间隔内的统计值。

假如会话 A 在 8 小时内已占用了 50 分钟的 CPU 时间，会话 B 在 1 小时中已经占用了 30 分钟的 CPU 时间，此输出将报告会话 A 较会话 B 来说是高 CPU 用户——事实并非如此，因为会话 B 的 CPU 使用率远超过会话 A 的 CPU 使用率。然而，此输出对系统中的顶端 CPU 用户做了一个相当精确的概述。

假如你希望更精确的话，你必须捕获经过一个时间间隔的统计值。选择系统在一天中最忙的一段时期，然后捕获该会话的 CPU 使用信息并把它存储在一个临时表中。在繁忙期过去之后，捕获每一个使用 V\$SESSTAT 视图的会话的 CPU 使用信息并把它存储在一个临时表中。在存储于临时表中的两个快照中捕获的信息能够用来了解间隔期内单个会话的 CPU 使用率以及总 CPU 使用情况。这可以用清单 22-3 中的脚本来实现。

清单 22-3 创建存储快照信息的临时表

```

REM Table to store the start snapshot values
Create table stat$cpu_begin
(sid number,
spid varchar2(9),
schemaname varchar2(30),
cpu_usage number(10),
program varchar2(48),
time_stamp date);
REM Table to store the end snapshot values
Create table stat$cpu_end
(sid number,
spid varchar2(9),
schemaname varchar2(30),
cpu_usage number(10),
program varchar2(48),
time_stamp date);

```

在该表创建之后，使用清单 22-4 中的脚本可以对第一个快照进行处理。

清单 22-4 用于捕获 stat\$cpu_begin 表中的起始 CPU 统计信息的脚本

```

Insert Into stat$cpu_begin

```

```

Select a.sid,
       b.spid,
       a.schemaname,
       value,
       a.program,
       sysdate
From   v$session a,
       v$process b,
       v$sesstat c
Where  c.statistic# = 12
And    c.sid        = a.sid
And    a.paddr      = b.addr;

```

类似地，在繁忙期结束后，你可以再运行清单 22-5中的脚本并生成stat\$cpu_end表。

清单22-5 用于捕获stat\$cpu_end表中的结束CPU统计信息的脚本

```

Insert Into stat$cpu_end
Select a.sid,
       b.spid,
       a.schemaname,
       value,
       a.program,
       sysdate
From   v$session a,
       v$process b,
       v$sesstat c
Where  c.statistic# = 12
And    c.sid        = a.sid
And    a.paddr      = b.addr;

```

现在准备生成CPU密集型用户报告以及CPU使用率。该报表更精确，因为它报告了CPU的使用率，该使用率是确定资源密集型用户的一个重要因素。该报表用每小时使用CPU的分钟数显示了每个会话的CPU消耗率。此快照技术消除了清单 22-5中的差异。清单 22-6中显示的脚本用来产生CPU使用报告。

清单22-6 用于使用快照生成CPU使用报告的脚本

```

Break on sid on spid on tot_usg on mins_per_hr
Select e.sid,
       e.spid,
       e.cpu_usage,
       b.cpu_usage,
       to_char((e.cpu_usage - b.cpu_usage)/(6000),'9999.99') tot_usg_mins
,
       to_char(((e.cpu_usage - b.cpu_usage)/(6000))/((e.time_stamp - b.time_stamp)*24),'99.99') mins_per_hr
From   stat$cpu_end e, stat$cpu_begin b
Where  b.sid = e.sid
And    b.spid = e.spid
order by mins_per_hr;

```

注意 此SQL代码运行在使用V\$SESSTAT视图捕获的统计数字基础之上，此V\$SESSTAT视图是对于统计量为12（CPU used by this session）的视图。如果该值在返回到客户进程前的退出用户调用时被Oracle加1，那么使用该值有时会出问题。因此，假如你正在监控一个具有还没有返回的用户调用的批处理作业，那么CPU使用值或许不能在V\$SESSTAT视图中更新并且你可能会得到不准确的结果。当你正在使用以前的SQL列

表时要注意这一点。假如监控时间间隔较大且所有的会话在此间隔内的某个时间点上更新V\$SESSTAT视图，这样就可以得到准确的结果。

既然你已经识别了大量使用 CPU资源的会话，那么你可以确定正在执行的大量使用 CPU的会话是什么。

清单22-7中的脚本可以用来发现会话当前正在执行的 SQL语句。这清楚地指出了会话正在执行哪一个程序，当前代码段是什么以及是否需要在该 SQL上执行一个查询优化。

清单22-7 用于发现会话当前正在执行的 SQL语句的脚本

```
set head off
set long 9000
set linesize 100
set pagesize 30
set feedback off
set wrap on
set verify off

select sql_text
from v$sqltext , v$session
where sid = 45
and v$sqltext.address = v$session.sql_address
order by piece

The following listing is the output of the above SQL statement.

SQL_TEXT
-----
Select * from employee where emp_id =800
```

注意 这里提到的监控技术假定用户不经常注册或注销。假如用户经常注册或注销，那么由Oracle分配的会话标识会变化，使得在会话级跟踪或报告会话变得困难起来。

注意 因为繁忙的时间间隔已经过去，所以它与执行清单22-7中的脚本无关。然而，通过在间隔期内周期性地查询，你可以使用该脚本在临时表中存储会话正在执行的所有 SQL命令。你能够使用下面的策略完成这项工作。

使用清单22-8中的脚本创建临时表并存储由会话执行的 SQL语句。

清单22-8 用于创建SQL执行历史记录表stat\$session_hash的脚本

```
Create table stat$session_hash
(sid number,
spid varchar2(9),
schemaname varchar2(30),
hash_value number,
time_stamp date,
num_occ number);
```

在创建了该表后，你可以运行清单22-9中的脚本以创建sp_stat\$sample_hash过程，该过程在SQL执行历史记录表stat\$session_hash中存储由监控期内的不同会话执行的所有 SQL语句。

清单22-9 用于创建sp_stat\$sample_hash过程的脚本

```
Cursor c_stat$session is
Select sid,
```

```
        spid,
        schemaname,
        sql_hash_value
    From v$session a,
        v$process b
    Where a.paddr = b.addr;

v_stat$session      c_stat$session%RowType;
v_stat$session_hash stat$session_hash%RowType;
e_next              exception;

Begin

    Open c_stat$session ;

    Loop

    Begin
        Fetch c_stat$session
        Into v_stat$session;

        Begin

            Select *
            Into v_stat$session_hash
            From stat$session_hash
            Where sid = v_stat$session.sid
            And spid = v_stat$session.spid
            And hash_value = v_stat$session.sql_hash_value;

            Insert Into stat$session_hash
            values
            (v_stat$session.sid,
             v_stat$session.spid,
             v_stat$session.schemaname,
             v_stat$session.sql_hash_value,
             sysdate,
             1
            );

        Exception
        When No_Data_Found Then
            Update stat$session_hash
            Set num_occ = num_occ + 1
            Where sid = v_stat$session.sid
            And spid = v_stat$session.spid
            And hash_value = v_stat$session.sql_hash_value;

        When Others Then
            Raise e_next;

    End;

Exception

When e_next Then
    Null;

When Others Then
    Null;

End;

Commit;
```



```

End Loop;

Close c_stat$session;

End;

```

该过程 (sp_stat\$sample_hash) 能够在监控期内间歇地运行。在捕获了开始和结束 CPU 使用信息以及 SQL 执行历史之后, 你可以产生一个 CPU 密集型用户以及每一个会话在快照期内所执行的 SQL 语句的报告。该报告随后便可以用来调整引起大量使用 CPU 的 SQL 语句。清单 22-10 给出了产生大量使用 CPU 报告的脚本以及由会话执行的 SQL 语句的哈希值。

清单22-10 用于产生CPU使用报告以及在监控期内执行的 SQL 语句的哈希值的脚本

```

Break on sid on spid on tot_usg_mins on mins_per_hr

Select e.sid,
       e.spid,
       to_char((e.cpu_usage - b.cpu_usage)/(6000), '9999.99') tot_usg_mins,
       to_char(((e.cpu_usage - b.cpu_usage)/(60100))/((e.time_stamp - b.time_stamp)
)*24), '99.99') mins_per_hr,
       hash_value,
       num_occ
From   stat$cpu_end e, stat$cpu_begin b, stat$session_hash c
Where  b.sid = e.sid
And    b.spid = e.spid
And    e.sid = c.sid
And    e.spid = c.spid
order by mins_per_hr;

```

The following listing is the output of the above SQL statement.

SID	SPID	Tot Usg	Mins	HASH_VALUE	NUM_OCC
	In Mins	Per Hr			
23	9461	29.04	9.68	-1.447E+09	7
24	9776	25.38	8.46	1739700154	1
				-1.306E+09	5
				1014890971	1
34	11525	22.41	7.47	573420945	7
35	9459	17.55	5.85	1739700154	2
				1561770773	1
35	9459	10.32	3.44	2113573249	1
59	9531	.00	.00	1739700154	6
56	10010	.00	.00	573420945	7
53	11114	.00	.00	-2.056E+09	7
52	10014	.00	.00	1739700154	3
105	9533	.00	.00	573420945	7
86	9529	.00	.00	573420945	7
				1907631373	1
73	9527	.00	.00	573420945	7
62	9541	.01	.00	-1.306E+09	5
109	16988	.00	.00	1848020776	7
52	10014	.00	.00	-1.306E+09	1

从此输出中, 你能够很容易地确定会话 23 与 24 是顶端 CPU 用户, 他们每小时使用 CPU 的时间分别是 9.68 分钟与 8.46 分钟。使用 sql_hash_value 与 V\$SQLTEXT 视图, 你可以确定长时间使用 CPU 的 SQL 语句并采取相应的措施。清单 22-10 中的 num_occ 列显示了在监控期内执行 SQL 语句的次数。

注意 大量使用CPU的SQL语句很可能是全表扫描或系统中引起更多物理读取的SQL语句。当所需要的数据已经不在数据块缓冲区时会发生遗漏，数据必须从物理设备中读取。假如没有空闲的空间，数据遗漏会引发缓冲区管理器在缓冲区缓存中为将要读取的数据分配空间。然后，缓冲区管理器把要被写出的数据块移动到脏列表中以便它们可以随后被写出到磁盘中。与直接从高速缓存中读出数据相比，缓冲区管理器的处理要求使物理读取成为一个CPU密集型操作。

2. 使用操作系统途径

你可以使用top命令查找UNIX系统中的CPU密集型用户。top命令按照等待时间由高到低的顺序列举了最大量使用CPU用户。当顶端CPU用户被找到后，你可以发现该用户正在做什么。清单22-11是top命令的一个输出示例。

清单22-11 top命令的输出示例

```
%top
System: arms1                               Sat Aug 30 16:35:57 1997
Load averages: 0.59, 0.61, 0.69
20 processes: 18 sleeping, 2 running
Cpu states:
CPU   LOAD   USER    NICE    SYS   IDLE   BLOCK  SWAIT   INTR   SSYS
0     0.56  34.6%   0.0%    1.4%  64.0%  0.0%   0.0%   0.0%   0.0%
1     0.61  57.6%   0.0%    1.2%  41.3%  0.0%   0.0%   0.0%   0.0%
...
avg   0.59  46.2%   0.0%    1.2%  52.7%  0.0%   0.0%   0.0%   0.0%

Memory: 25956K (10740K) real, 37996K (19316K) virtual, 15364K free Page# 1/14

CPU  TTY  PID USERNAME  PRI NI  SIZE  RES  STATE  TIME %WCPU %CPU COMMAND
1    ?  28586 manish   238 20  9164K 1072K run   157:22 90.13 89.97 oracleor
1    ?  1020 oracle    154 20  8712K  616K sleep  0:10  1.47  1.46 oracleor
1    ?  1900 oracle    154 20  8692K  588K sleep  0:05  1.10  1.09 oracleor
1    ?  1988 oracle    154 20  8620K  528K sleep  0:04  0.55  0.55 oracleor
0    ?   822 oracle    154 20  8764K  656K sleep  0:24  0.51  0.51 oracleor
1    ?  1793 oracle    154 20  8636K  524K sleep  0:01  0.43  0.43 oracleor
1    ?   53 root      100 20    0K    0K sleep 226:02  0.37  0.37 netisr
0    ?   304 root      154 20   24K   24K sleep 258:04  0.33  0.33 syncer
0    ?    0 root      127 20    0K    0K sleep 86:40  0.30  0.30 swapper
0    p2  2204 evett     178 20   720K  312K run    0:00  0.92  0.27 top
1    ?  29494 oracle    154 20  8772K  660K sleep  0:28  0.23  0.23 oracleor
```

从此输出中，很明显可以看出PID为28586并且USERNAME为manish的用户是CPU密集型用户。注意，从USER、SYS与IDLE的值中，你可以看到系统显示USER的值为46%，SYS为1.2%，其余是空闲的。这意味着该系统主要服务于用户请求，这是一个好的迹象。尽管用户是顶端CPU用户，但是这不必惊慌，因为在一个负载较轻的系统中，如果有一个需要大量CPU的进程，那么它会独占CPU。

要查看的关键数据是系统的负载平均值。第一行显示系统的负载平均值为0.59、0.61和0.69。0.59是最后1分钟的负载平均值，0.61是最后5分钟的负载平均值，0.69是最后15分钟的负载平均值。负载平均值清楚地表明负载量系统内以及最后的15分钟内的情况，当CPU负载量突然变化并且系统性能变坏时，负载平均值是非常有用的信息。当负载平均值突然增加时，你可以推断出这是由于系统中某些资源密集型进程而导致的。系统装载量为2或3表明系统轻度负载，装载量为5或6表明系统中度负载。其他要查看的信息有%CPU列，它指示了由特定进程使用的CPU

时间的百分率。如果该数值较高，则很明显表明该进程是一个大量花费CPU时间的进程。

你还可以使用ps命令确定CPU密集型用户。ps命令包括了一个显示CPU在服务于特殊进程时所花费时间的列。你可以按照花费CPU时间由高到低的顺序对ps的输出进行排序，并再次找出顶端CPU资源用户。

清单22-12显示PID为28586和14171是顶端CPU资源。自进程开始以来，28586已经使用了192.21分钟的CPU时间，14171已经使用了152.29分钟的CPU时间。

清单22-12 使用ps命令得到CPU使用情况

```
$ ps -ael | sort -n -r -k 13
```

F S	UID	PID	PPID	C	PRI	NI	ADDR	SZ	WCHAN	TTY	TIME	COMD
1 R	105	28586	28586	89	200	20	3903a00	1741	-	?	192:21	oracle
1 S	104	14171	1	0	154	20	3937a00	1711	36f8b38	?	152:29	oracle
3 S	0	3	0	0	128	20	2529100	0	3c2050	?	122:09	
↳statdaemon												
3 S	0	0	0	0	127	20	40c8b0	0	3c2050	?	86:47	swapper
3 S	0	2	0	0	128	20	2155e80	0	40e990	?	32:19	vhand
1 S	0	459	458	0	127	20	2718180	2	7ffe6000	?	32:05	netfmt
1 S	0	1016	1	0	154	20	2874180	9	40e0a8	?	22:52	spserver
1 S	0	787	1	0	154	20	280e880	180	40e0a8	?	21:01	dced
1 S	0	681	1	0	154	20	27cef80	85	40e0a8	?	15:47	snmpdm
1 S	104	1339	1	0	154	20	2845980	195	40e0a8	?	14:18	tnslsnr
3 S	0	12	0	0	138	20	254de00	0	391cf40	?	12:00	
↳vx_sched_threa												
3 S	0	54	0	0	100	20	254d980	0	-	?	6:24	nvsisr
1 S	125	13379	1	0	156	20	3330200	1535	461a84	?	5:21	oracle
3 S	0	4	0	0	128	20	2529380	0	40e4dc	?	4:09	
↳unhashdaemon												
1 S	125	13383	1	0	156	20	3739d80	1529	461a94	?	4:41	oracle
1 S	104	21303	1	0	154	20	2948d80	1846	3e55438	?	4:02	oracle

注意 这些进程可能是非活动的，所以注意那些已经在一天的开始时消耗了大量CPU资源、目前处于空闲状态的会话。要确定进程是否是空闲的，再次运行相同的命令并检查CPU时间是否增加了。活动会话可以通过查看清单22-12中的第二列来得到识别。数值R表明它正在运行。即便在这里，一个具有S状态的进程当前也可以在睡眠，因为它正在等待某些事情，然后转入运行状态。为了识别一个当前处于活动状态的顶端CPU资源，请记住以上所有的这些东西。换句话说，显示于Oracle途径中的快照技术可以用于识别。最后，管理员可以把来自Oracle途径的值以及来自操作系统途径的值在资源密集型用户中调整归零。

在pid被识别之后，你可以找到sid进程并进一步使用Oracle的V\$SQLTEXT视图来查找当前正在执行的进程。这可以使用清单22-13中的脚本来完成。

清单22-13 用于从sid获得pid并获得会话正在执行的SQL语句的脚本

```
REM Find the sid of the Unix process using the process id
Select sid
From v$session a,
     v$process b
Where a.paddr = b.addr
And b.spid = <pid> ;

REM Find the SQL the user is currently executing
Select sql_text
```

```
From v$sqltext a ,  
      v$session b  
Where   a.hash_value = b.sql_hash_value  
And     b.sid        = <sid>  
Order By piece;
```

在Windows NT操作系统中，你可以监控处理器对象并使用 %处理器时间计数器生成一张一定时期内的CPU使用情况图。你可以创建数据的折线图或条形图。

大量使用磁盘可能引起CPU性能问题。当磁盘正在执行大量的输入/输出时，CPU会因为处理输入/输出中断而陷入停顿。你可以在 PerfMon（Windows NT操作系统）中通过查看处理器：中断数/秒和处理器：%中断时间来对此进行监控。

假如CPU花费过多的时间处理中断，可能会引发严重的性能衰退。还有，你可以检查处理器应用时间（处理器：%处理器时间），如图22-2所示。假如该时间量的持续时间超过90%，就说明有一个CPU瓶颈。

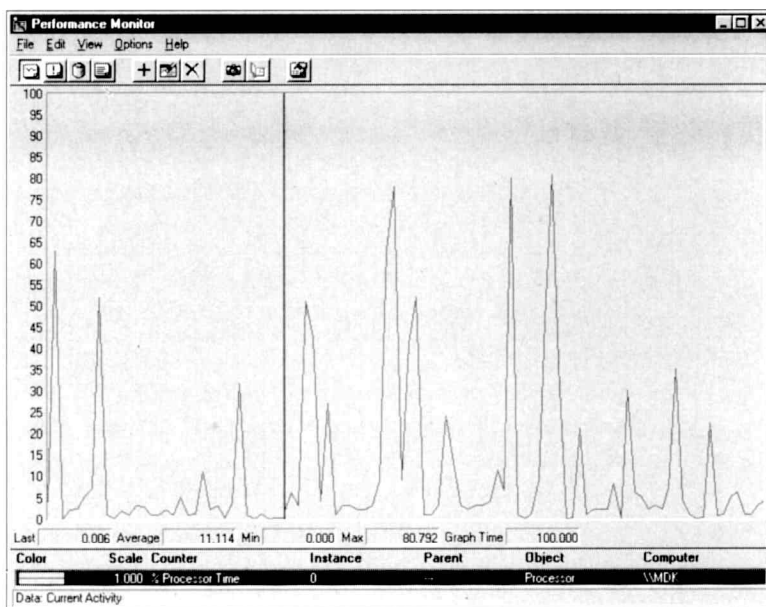


图22-2 使用Oracle 8性能监控程序监控处理器使用时间

22.3 文件输入/输出

当某个用户程序需要对数据进行存取时，操作系统首先查看该数据当前在高速缓存中是否可用，以及如果可用的话，它将如何被使用。假如在高速缓存中没有找到该数据，那么就不得不从磁盘中读取数据。与从高速缓存中读取数据相比，从磁盘中读取数据的过程代价很大，因为内存存取比磁盘存取更快。磁盘操作极大地阻碍了进程的吞吐量。

基于磁盘存取的进程性能依赖于诸如输入/输出控制卡的数量、软件、磁盘速度等因素。

当一个进程请求数据时，下面所列的是检索被请求的数据时发生的短暂事件：

- 1) 输入/输出控制器把数据地址定位在正确的磁盘上；这被称作队列延迟 (queuing delay)。一般还存在其他一些在给定的时刻从磁盘中读取数据的未完成的请求。花费在队列中的时间依赖于操作系统与磁盘上装载的系统。对磁盘中未完成的请求进行排序以便磁盘头不必做太

多的来回移动。这种对请求的排序可能导致请求在队列中花费更多的时间，但搜索时间减少了。

2) 当数据的存放位置找到后，磁盘臂——或者更精确地说，磁盘头必须定位在正确磁盘的正确磁道上。发生在这里的时间延迟叫做搜索时间（seek time）。

3) 当磁盘在磁盘头下旋转时，磁盘头读取该磁盘的数据。在磁盘头被定位在正确的磁道后，它必须等待磁盘旋转到相关的数据扇区刚好位于磁盘头下，这样便能完成读/写操作。这个阶段通常称为旋转等待时间（rotational latency）。

4) 通过磁盘头读出的数据被传送到控制器。这最后一部分的输入/输出被称为数据传送时间（data transfer time）。最终，该数据在输入/输出板上传送并发送到发出数据请求的进程。

用于从磁盘中检索数据的总时间可以粗略地表达为如下形式：

排队时间+搜索时间+旋转等待时间+数据传送时间

可以清楚地看到，磁盘存取包含磁盘的机械和旋转运动，假如有多用户并行地在同一张磁盘上频繁进行输入/输出调用时，磁盘存取或许会成为系统中的瓶颈。最终，磁盘的输入/输出率支配了该磁盘的总吞吐量，在该磁盘中形成用户队列。

22.3.1 输入/输出预览

在你开始查找输入/输出密集型用户以前，必须确定任何一个文件系统当前是否被频繁地存取。如果是这样的话，开始定位这些用户。

像以前一样，得到短时间段的系统快照，然后获得时间间隔输出。通过查看输出值，很容易估价是否正在进行大量的输入/输出。

运行清单 22-14 的脚本可以对系统有一个快速的了解。在该清单中，有 stat\$begin_file 和 stat\$end_file 两个表用于存储开始和结束文件输入/输出的统计信息。利用两个快照之间的差别与时间，你可以计算出文件存取率。运行清单 22-14，在该快照间隔的起始处存储文件输入/输出统计信息。

清单22-14 用于在stat\$begin_file表中创建临时表和插入值的脚本

```
REM
REM Start IO : Script to Capture File I/O Statistics at the beginning of the
interval.
REM
Drop View stats$file_view;
Create View stats$file_view As
Select ts.name      ts,
       i.name       name,
       x.phyrds     pyr,
       x.phywrt     pyw,
       x.readtim     prt,
       x.writetim    pwt,
       x.phyblkrd    pbr,
       x.phyblkwrt   pbw,
       sysdate       st_time
From v$filestat x, ts$ ts, v$datafile i, file$ f
Where i.file#=f.file#
And ts.ts#=f.ts#
And x.file#=f.file#;

Drop Table stats$begin_file;
```



```
Create Table stats$begin_file Tablespace tools
Storage(Initial 10K Next 10K Pctincrease 10 ) As
Select *
From stats$file_view
Where 0 = 1;
```

```
Drop Table stats$end_file;
Create Table stats$end_file Tablespace tools
Storage(Initial 10K Next 10K Pctincrease 10 )As
Select *
From stats$begin_file;
```

```
Insert Into stats$begin_file
Select *
From stats$file_view;
```

在想要的间隔结束后，运行结束输入/输出脚本以便在间隔结束时捕获统计数字以对系统的性能进行分析。结束输入/输出脚本捕获统计数字并存储于 stat\$end_file表中，然后产生输入/输出报告。清单22-15是用于结束输入/输出并获得输入/输出使用报表的脚本。

清单22-15 用于填充stat\$ file_end表并产生输入/输出使用报告的脚本

```
REM
REM End IO : Script to Capture File I/O Statistics at the end of the interval.
REM
Set Term Off
Insert Into stats$end_file
Select *
From stats$file_view;

Drop Table stats$files;
Create Table stats$files Tablespace tools
Storage(Initial 10K Next 10K Pctincrease 0)As
Select b.ts table_space,
b.name file_name,
e.pyr-b.pyr phys_reads,
e.pbr-b.pbr phys_blks_rd,
e.prt-b.prt phys_rd_time,
e.pyw-b.pyw phys_writes,
e.pbw-b.pbw phys_blks_wr,
e.pwt-b.pwt phys_wrt_tim,
(e.st_time -b.st_time)*24*60*60 tot_tim
From stats$begin_file b, stats$end_file e
Where b.name=e.name;

Drop Table stats$begin_file;
Drop Table stats$end_file;
Set Term On
Clear Screen
Set Pause On
Set Pagesize 24
Set Feedback off
Col table_space Format a17
Col file_name Format a55
Col phys_writes Format 9999999 Heading 'PHY_WRS'
Col phys_reads Format 9999999 Heading 'PHY_RDS'
Col phys_rd_time Format 999999 Heading 'PHY_RDT'
Col phys_wrt_tim Format 999999999 Heading 'PHY_WRT'
Col phys_blks_rd Format 999999999 Heading 'BLKS_RD'
Col phys_blks_wr Format 9999999 Heading 'BLKS_WR'
Col ios_sec Format 999 Heading 'IOS_SEC'
```

```

REM
REM Generate IO Overview Report
REM
Select table_space,
       file_name,
       phys_reads,
       phys_writes,
       phys_blks_rd,
       phys_blks_wr,
       phys_rd_time,
       phys_wrt_tim,
       (phys_reads + phys_writes)/tot_tim ios_sec
From stats$files
Order By ios_sec Desc

```

The following listing is the output of the above SQL statement.

TABLE_SPACE	FILE_NAME
PHY_RDS PHY_WRS	BLKS_RD BLKS_WR PHY_RDT PHY_WRT IOS_SEC
ARMSLIVEDB04_TS	/u03/oradata/ora7/armslive/armslivedbu04.dbf
23100 34	23100 34 17644 106 93
ARMSLIVEIDX02_TS	/da11/oradata/ora7/armslive/armsliveidxa07.dbf
16886 0	16886 0 912 0 68
ARMSLIVEIDX11_TS	/u14/oradata/ora7/armslive/armsliveidx19.dbf
3461 542	3461 542 1288 2946 16
ARMSLIVEDB04_TS	/u03/oradata/ora7/armslive/armslivedbu03.dbf
2521 38	2521 38 2131 255 10
RBS2	/db07/oradata/ora7/rbs4ora7.dbf
1 982	1 982 3 7890 4
ARMSLIVEIDX03_TS	/db04/oradata/ora7/armslive/armsliveidxb01.dbf
870 51	870 51 323 124 4
ARMSLIVETMP01_TS	/db06/oradata/ora7/armslive/armslivetmpb01.dbf
866 1	24729 1 1402 2 3
ARMSLIVEDB12_TS	/u03/oradata/ora7/armslive/armslivedbu19.dbf
157 323	157 323 32 1065 2
ARMSLIVEDB11_TS	/u09/oradata/ora7/armslive/armslivedbu16.dbf
87 331	273 331 30 1693 2
ARMSLIVEIDX10_TS	/u16/oradata/ora7/armslive/armsliveidx17.dbf
137 25	137 25 69 66 1
SYSTEM	/u01/oradata/ora7/syst1ora7.dbf
63 23	79 23 31 204 0
SYSTEM	/u01/oradata/ora7/syst2ora7.dbf
34 12	37 12 11 90 0
ARMSLIVEDB02_TS	/db06/oradata/ora7/armslive/armslivedb06.dbf
7 32	7 32 1 203 0

从此输入/输出总报告中可以看出，系统中很明显存在一些输入/输出活动并且受此影响最严重的表空间是 ARMSLIVEDB04_TS。该表空间的输入/输出率大约是每秒完成 93 次读/写。根据磁盘的速度或磁盘能够支持的输入/输出率，可以确定获得的数值是否是高的。在发现系

统中有大量活动发生后，你现在可以鉴别引起此大量输入/输出活动的用户。

你还可以在UNIX操作系统上使用sar命令得到操作系统级的一个抽样的概况。清单 22-16 中的sar命令每5秒钟对磁盘活动进行3次采样。

清单22-16 用于监控磁盘输入/输出统计数字的sar命令

```
$ sar -d 5 3
```

```
HP-UX arms1 B.10.10 U 9000/819 08/30/97
```

	device	%busy	avque	r+w/s	blks/s	avwait	avserv
19:43:00	c0t5d0	76.05	1.94	169	2711	8.76	5.94
19:43:05	c0t4d0	9.58	0.95	12	192	8.86	9.04
	c0t3d0	8.18	0.50	6	102	4.77	12.28
	c3t6d0	7.58	0.50	28	447	5.20	3.12
	c4t6d1	15.77	1.30	28	450	8.44	14.35
	c4t5d0	6.19	0.50	5	77	5.02	12.64
	c3t4d0	7.19	0.50	9	150	5.27	7.68
	c3t5d0	0.40	0.50	0	6	3.80	9.32
19:43:10	c0t6d0	0.20	0.50	0	1	6.42	11.56
	c0t5d0	72.60	1.79	186	2980	8.59	5.09
	c0t4d0	6.20	0.50	7	115	5.48	8.48
	c0t3d0	8.60	0.50	7	112	5.19	13.32
	c3t6d0	6.20	0.50	27	438	5.12	2.42
	c4t6d1	15.40	1.52	25	394	9.70	16.09
	c4t5d0	3.60	0.50	3	42	4.40	14.15
	c3t4d0	5.00	0.50	6	99	5.63	7.49
19:43:15	c0t5d0	73.20	1.86	184	2947	9.44	5.38
	c0t4d0	12.00	0.50	14	221	5.05	9.26
	c0t3d0	8.60	0.50	6	99	5.39	12.99
	c3t6d0	6.40	0.50	27	435	5.31	2.33
	c4t6d1	14.40	1.33	23	365	10.21	14.72
	c4t5d0	4.20	0.50	3	51	5.38	13.08
	c3t4d0	9.60	0.50	13	202	4.76	7.91
Average	c0t5d0	73.95	1.86	180	2879	8.93	5.46
Average	c0t4d0	9.26	0.66	11	176	6.53	9.01
Average	c0t3d0	8.46	0.50	7	104	5.11	12.88
Average	c3t6d0	6.73	0.50	28	440	5.21	2.63
Average	c4t6d1	15.19	1.38	25	403	9.38	15.03
Average	c4t5d0	4.66	0.50	4	56	4.97	13.14
Average	c3t4d0	7.26	0.50	9	150	5.12	7.74
Average	c3t5d0	0.13	0.50	0	2	3.80	9.32
Average	c0t6d0	0.07	0.50	0	0	6.42	11.56

在输入/输出概况快照脚本被执行的同时，这个sar命令被激发。在该命令中，磁盘c0t5d0的读/写率为180，这清楚地表明该磁盘上存在大量的输入/输出。sar命令的输出被详细列出；其中，avwait是花费在队列中的时间，avserv为搜索时间与旋转等待时间、磁盘传送时间三者之和。这条信息相当有用。使用这两项技术，你可以鉴别频繁存取的磁盘和文件。

现在寻找使用大量输入/输出资源的用户。在操作系统级，寻找执行大量输入/输出操作的用户很困难；因此，可使用Oracle系统表取得你所要的信息。

22.3.2 寻找输入/输出密集型用户

你可以使用V\$SESS_IO视图来鉴别资源密集型用户。该视图存储有关会话的输入/输出统计数字的信息。对于每一个会话，在该视图中有一行与之对应。对V\$SESS_IO视图中的列的

解释在表22-4中给出。

表22-4 V\$SESS_IO视图的列描述

列 名	描 述
SID	会话标识符（数据库名）
BLOCK_GETS	会话获得的数据块的数量
CONSISTENT_GETS	使用一致性获取机制的会话的一致性获取的数量
PHYSICAL_READS	此会话的物理读取的数量
BLOCK_CHANGES	此会话改变的数据块的数量
CONSISTENT_CHANGES	此会话的一致性改变的数量

像以前的小节中所讲述的那样，你可以在该视图中使用快照技术，将在一个监控期中捕获的开始和结束快照值存储在一个临时表中，然后利用差别来查找执行大量文件存取的用户。使用在 PHYSICAL READS 列中的差别来查找从磁盘中执行大量数据存取的用户。BLOCK_GETS列中的差别可以用来鉴别执行大量缓冲区读取的用户。可以开发与 22.2节中类似的脚本。

在数据库中引起大量磁盘存取的最常见的事情是磁盘排序（会在临时表空间中引起瓶颈）、全表扫描和频繁地索引读取。

可以使用清单22-17中的脚本鉴别当前正在执行一个排序的会话。

清单22-17 用于鉴别当前正在执行一个排序的会话脚本

```
Select sid
From v$session_wait
Where event = 'db file scattered read'
And p1 = <file id>;
```

在V\$session_wait中的p1列中包含文件标识符（file id），事件将在该标识符上执行。现在，可以把属于临时表空间的文件的 file id用数据库数据文件中的 file#值替换。从用于数据库中的临时表空间的 dba_data_files 中选取 file# 列，然后替换清单22-17中的 file id 变量。

在发现 sid 值以后，可以使用 V\$SQLTEXT 视图和清单22-17中的脚本发现由此会话执行的当前 SQL 语句。如果在数据库中执行了过多的磁盘排序，考虑增加 init.ora 的参数 SORT_AREA_SIZE 的值。

你可以使用清单22-18中的脚本鉴别一个当前正在执行全表扫描且正在等待多块读取调用返回的会话或最后一个等待用于一个多块读取的会话。清单 22-18 立刻便能识别所有执行全表扫描的会话。

清单22-18 用于查找当前正在执行全表扫描的会话脚本

```
col sql_text format a40

select sid,sql_text
From v$session a, v$sqlarea b
Where a.sql_hash_value = b.hash_value
And sid in
  (Select sid
   From v$session_wait
   Where event like 'db file scattered read')
/
```

The following listing is the output of the above SQL statement.

```

SID          SQL_TEXT
-----
100 select ord_id from cust_ord where crea_u
      ser_id like 'BSMJG%'
102 select emp_id,name  from employee where
      salary > 1003;

```

你可以使用V\$SQLAREA视图鉴别正在执行大量物理读取和访问大量缓冲区的 SQL 语句,如清单22-19所示。你可以用一个你认为合适的值替换清单 22-19中的buffer_gets和disk_reads。

清单22-19 用于鉴别正在执行过多物理读取或存取大量缓冲区的会话的脚本

```

set pagesize 100
col sql_text format a50

select substr(sql_text,1,200) sql_text, buffer_gets, disk_reads
from v$sqlarea
where buffer_gets > 10000
or disk_reads > 10000000
order by disk_reads desc;

```

The following listing is the output of the above SQL Script.

SQL_TEXT	BUFFER_GETS	DISK_READS
Select RTrim (cust_ord.brand_id),RTrim (ord_id ,Nvl (cust_ord.pmt _mthd_id ,'12'),Nvl (card _num ,'0'),To_Char (Nvl (card_exp_dt ,sysdate , 'mmyy'),Nvl (pp_amt ,0.00),Nvl (ord_amt ,	20488669	13025204
Select count (*) From cust_ord ,cmpn_pmt_type Whe re auth_stat ='Y' and cust_ord.pmt_mthd_id In (Se lect pmt_mthd_id From pmt_mthd Where pmt_mthd_typ in ('R' ,'D')) and cust_ord.cmpn_id =cmpn_pmt_t	13136493	12952619
SELECT TITLE_ID,CNTRY_ID FROM CUST_ORD_ADR WHER E RTRIM(UPPER(ORD_ID)) = RTRIM(UPPER(:b1)) AND RT RIM(UPPER(ADR_ID)) = RTRIM(UPPER(:b2))	10469659	10453418
update cust_ord set ord_stat = '07', chng_user_id = 'stat19', chng_dt = sysdate where ord_id in (select c.ord_id from cust_ord a, cust _ord_line b, cust_ord_itm c where a.ord_id	24942696	3454225

这些SQL语句中的大部分很有可能正在执行全表扫描,从而引发了大量读操作。通过添加辅助性能索引或使用其他的调整技术对 SQL 语句进行优化。

在Windows NT操作系统中,你必须从Windows NT命令提示符执行如下的DISKPERF命令以使磁盘计数器能够收集到有关磁盘读写的统计数字:

```
DISKPERF -YE
```

这会影响到系统开销;这也就是为什么在缺省情况下不使用该计数器的原因。你必须重新启动该系统以便得到逻辑磁盘计数器的正确读数。

DB_BLOCK_SIZE参数对磁盘输入/输出有影响。DB_BLOCK_SIZE应当等于或成倍于NT簇容量。DB_BLOCK_SIZE决不能小于NT簇容量。簇容量是Windows NT一次所能够读出或写入的最小的输入/输出基本单元。例如,如果DB_BLOCK_SIZE是4K并且NT簇容量是1K的话,那么每当Oracle需要从磁盘中读取数据时,Windows NT就需要执行4次输入/输出操作。

这是不可接受的。

22.3.3 磁盘队列

一个磁盘驱动器一次仅能支持一个请求。当有其他进程等待该磁盘时便会形成磁盘队列。假如该磁盘队列过载的话，你会发现性能出现问题。PerfMon计数器的物理磁盘：磁盘传送量/秒项目让你了解到一定时间中的磁盘传送率。通过把物理磁盘：平均值制成图还能够查看磁盘队列（包括读和写队列）。磁盘队列的长度显示于图 22-3 中。

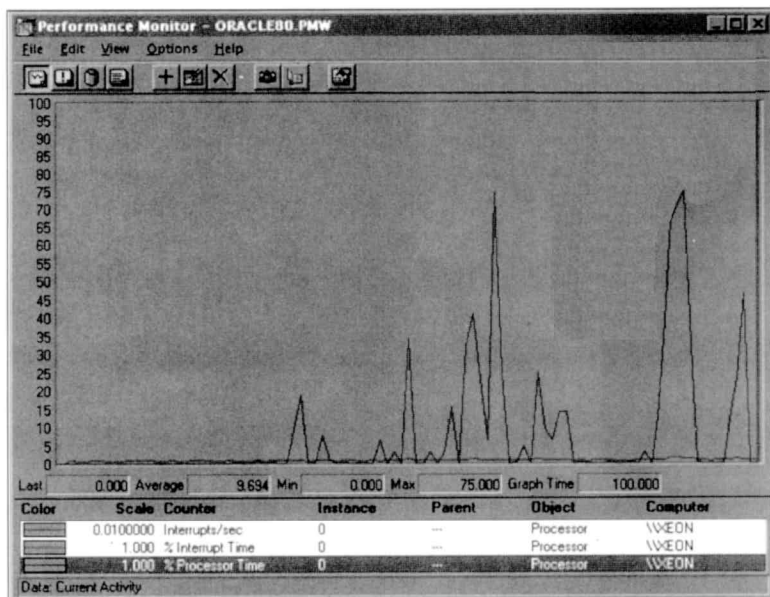


图22-3 监控物理磁盘性能

采用如下预防措施以使输入/输出问题减到最小：

- 通过使用多个磁盘驱动器和磁盘控制器分散输入/输出装载值。
- 将分离的、无数据条的磁盘上的 Oracle 联机重做日志与数据文件分开存放。
- 用更快的磁盘、控制器和输入/输出总线对硬件进行升级，例如，从 IDE 磁盘到 SCSI 磁盘或从 SCSI 磁盘到更快的 SCSI。

22.4 内存

UNIX 系统把虚拟内存用于内存管理。使用虚拟内存进程，系统中所有进程的总容量可能超过系统中可用的物理内存。可以使用分页与交换技术——UNIX 系统中的两种内存管理策略来实现虚拟内存。虚拟内存技术使进程的容量大于可用的物理内存数量成为可能。

主存（RAM）存储执行程序所需要的数据。当数据不再被需要用于执行时，它被存储在磁盘上的辅助内存中以便为其他的活动进程腾出空间。整个进程从主存移动到磁盘上的辅助内存区，这个区域被称为交换区（swap space）。

在内存分页中，单个内存页面被传送到交换区，也就是说，整个进程的一部分被传送到交换区。在交换中，整个进程被传送到交换区。内存分页只释放少量的内存，而交换释放大

量的内存。这些内存管理策略所用到的原理是将进程最需要的内存页面保留在物理内存中，其余的内存页面存放在辅助存储区，即交换区中。

当一个进程需要更多的内存时，物理内存中的非活动页面可以被分页出，物理内存中的空间被释放。当空间在物理内存区中被释放时，驻留在辅助内存中的页面便可以被带入物理内存中（RAM）。当UNIX系统把单个内存页面移动到交换区时发生内存分页。因此，仅有一部分进程移动到交换区。另一方面，交换把整个进程移动到交换区中。这样，交换释放了许多物理内存，但是它比内存分页代价更高。一个理想内存状况的关键指标是在完全负载的情况下，系统轻微分页并且根本不发生交换。

在大多数的系统中可以看到如下的情况：当系统有轻微的负载时，没有分页或交换；当系统适度负载时，开始进行分页；当系统负载较重时，由于繁重的内存请求与物理内存的低可用性，分页停止，系统中的交换开始。交换是最昂贵的内存操作。

在一个虚拟内存系统中，系统中的进程所使用的内存地址不是物理内存地址，而是虚拟地址（virtual address）。此虚拟地址由内存管理单元（MMU）转化为物理内存地址。

当进程需要一张内存页面时，它使用虚拟地址引用那一页。此虚拟地址由内存管理单元转化，用来定位实际物理地址。此转化的地址可以映射到物理内存中的一个区域中或者映射到辅助内存（交换区）地址中。

假如物理页面被定位在主存中，该进程可以立即使用它。如果它被定位在磁盘上，内存管理单元会产生页面出错（page fault），然后操作系统把此页面读到物理内存中的一个空闲页面中；这就叫做调入页面（page in）操作。假如内存严重短缺，此页面的读取会引发物理内存中的其他页面被写入到磁盘。这通常被称为调出页面（page out）。在页面被操作系统读入后，新的页面地址被传送到内存管理单元，然后操作系统重新启动指令。分页仅把物理内存页面移动到磁盘中；交换会把整个进程移动到磁盘中。假如系统中内存严重短缺，那么会非常频繁地发生交换。

无论是调出页面还是调入页面对于系统性能来说都是很昂贵的，因为与可直接在高速缓存中使用的数据相比，磁盘读/写非常昂贵。因此，假如进程由于在物理内存缺乏的系统中频繁申请新的数据页而产生大量的页面错误的话，进程的运行速度会明显慢下来。

在使用虚拟内存的系统中，只有由进程使用的那部分内存驻留在物理内存中。这部分内存被叫做该进程的工作组（working set）。因为在任何给定的时刻上，进程不会使用它的全部页面，因此把当前使用的页面保存在物理内存中是更为经济的做法。在进程运行期间，该进程有可能需要当前不在物理内存中的页面。这会引发页面出错，但是在进程运行期间出现一些页面出错问题是极为普通的，因为把进程需要的所有内存页面都放入主存中明显是不切合实际的。操作系统总是试图把工作组的全部页面保存在主存中。使用内存的这项技术可以使一个系统的内存吞吐量达到最大值，记住，系统中有许多并行进程正在运行，这些并行进程主动生成对内存的并行请求。

进程的实际物理内存使用决不是恒定不变的，由于它的易变性，要确定物理内存的使用非常困难。另一方面，进程对虚拟内存的需求可以很容易地被确定。一个进程所消耗的物理内存的数量基于许多因素，例如系统负载、进程正在做的工作等等。例如，假如一个进程正在一个较重负载的系统中运行，因为有许多需要物理内存页面的其他进程，所以物理内存中实际分配给该进程的页面的数量将非常接近于该进程的工作组的容量。换句话说，如果同一

个进程在一个较轻负载的系统中运行，该系统中正在运行的并行活动进程并不多，那么实际分配给该进程的物理页面的数量会大大超过该进程需要的工作组容量。类似地，在一个进程的运行过程中，如果该进程的工作组改变了，那么分配的物理内存的数量也会随之改变。所有这些使得很难预测消耗大量物理内存的程序，因为内存分配完全是随形势的发展而变化的。要模拟进程对物理内存的需求，你必须还要模拟系统中的负载，因为一个孤立运行在系统中的进程对物理内存的使用要大于运行在峰值负载上的进程对内存的使用。记住如下这些要点：

- 要监控系统中的进程对内存的使用，你必须把系统负载考虑在内。一个进程在较轻负载的系统中独占大量的物理内存而在一个较重负载的系统中占用少量内存是极为普通的事情。

- 要检查物理内存是否足够用于系统中运行的所有进程，最好的方法是做如下测试：

- 1) 将你的系统加载到峰值负载。

- 2) 检查整个内存工作情况（在下一节中讨论）以查看是否存在任何分页或交换的迹象。忽略少量的分页，因为当进程的工作组改变时，进程分页是很自然的事情。假如有过多的分页或交换发生，或许表明系统中可用的物理内存数量较少，你必须重新安排你的某些内存密集型进程的执行时间。

在你对监控内存的使用进行研究以前，分析一下单个进程的内存使用的分类。

22.4.1 进程内存分类

UNIX系统中的每一个进程被分为两个大的领域：文本页与数据页。文本（text）段包含进程执行的机器指令。数据（data）段包含进程在执行期间所需要的所有其他信息。文本页被标记为只读，因为它们包含机器指令并且从文件系统中调出。执行同一个程序的进程将共享相同的文本页，这样便优化了内存使用。例如，假如有 10 个用户工作在 Oracle Form 上，所有这些人共享 Oracle Form 的可执行文件或文本页。

另一方面，数据页被标记为读/写。它们专用于某进程并且从分页区（交换区）中调出；也就是说，每一个进程有它自己专用的数据页，当进程数量增加时，对专用页的需求也随之增加——这与共享文本页的情况不一样。主要的区别是假如一个文本页由于内存短缺而被调出，该文本页当前占用的物理内存页能够直接被新的信息改写；但是如果一个数据页被改写或调出，那么该页必须在物理页开始改写以前写入或拷贝到磁盘中。这其中的理由很简单，因为文本页决不会被修改，文本页的映象总能够从文件系统中获得。

一个UNIX进程实际上被划分为 6 个区域：文本、堆栈、堆、BSS、初始化数据和共享内存。

如前所述的一样，文本在进程之间共享并且从文件系统中调出。它包含机器指令并且被标记为只读。

堆栈（stack）对进程是专用的，它从交换区中调出，包含运行期执行堆栈。除了函数和过程参数以外，运行期堆栈还包括函数和过程活动记录。堆栈页被标记为读/写。堆栈的容量可以增长。

当堆（heap）运行时，它包含了分配给进程的数据页。堆是一个专用的进程区并且它的容量可以在程序运行期间增长得非常大。堆页被标记为读/写，这些页从交换区中调出。

BSS段对进程也是专用的。它被标记为读/写，它的容量在进程执行期间保持不变，BSS

段的页面被标记为读/写。该段的容量在进程执行期间保持固定。该段用于存储静态地分配的未初始化数据，这些页从交换区中调出。

初始化数据（initialized data）段对进程是专用的。它被标记为读/写，从交换区中调出，在进程执行期间其容量保持不变。

共享内存（shared memory）段是在进程之间共享的页面。它们被标记为读/写。使用系统调用 `shmget()` 分配共享页面。进程通过使用系统调用 `shmat()` 共享页面。共享页面从交换区中调出，共享内存段的容量在分配完成后便保持固定。

22.4.2 内存预览

在你深入研究使用内存的个别进程以前，必须首先对系统有一个大概的了解。当对系统中的内存进行预览时，要寻找的关键东西是系统中过多的分页与交换。假如系统中的内存比较缺乏，那么当进程被分配了 CPU 的一个时间片后，该进程把这段 CPU 时间用于交换与分页，不做急剧影响系统性能的其他事情。系统中频繁地分页与交换活动严重地妨碍了系统性能，必须通过增加额外的物理内存或重新安排进程的方式来尽可能地减小。

在 UNIX 系统中，`vmstat` 与 `sar` 是用来对系统内存的使用进行监控的工具。这两个工具的缺陷是它们仅能在总体水平上实现对内存的监控，不能给出每一个进程的内存使用细节。清单 22-20 所显示的是 `vmstat` 命令的输出。

清单 22-20 用于监控内存使用的 `vmstat` 命令的输出

```
vmstat -S 2 10
procs      memory      page      faults      cpu
r  b  w    avm free    si so    pi po    fr de    sr  in  sy    cs us sy id
1 111 0    5152 2885    0  0    0  0    0  0    0 1028 3068 1121 19  8 73
1 111 0    5152 2885    0  0    0  0    0  0    0 1004 2778 1074 17  5 78
2 117 0    4986 2885    0  0    0  0    0  0    0 1022 2666 1097 18 10 72
2 117 0    4986 2885    0  0    0  0    0  0    0 1008 2618 1129 22  7 71
0 111 0    5644 2842    0  0    0  0    0  0    0  994 2742 1079 27 10 64
0 111 0    5644 2841    0  0    0  0    0  0    0 1016 2974 1210 24 14 63
0 111 0    5644 2927    0  0    0  0    0  0    0 1029 4388 1438 21  9 70
0 109 0    5816 2927    0  0    0  0    0  0    0  997 3656 1359 12  8 81
0 109 0    5816 2927    0  0    0  0    0  0    0  953 3135 1255 16  5 79
```

表 22-5 中列举了 `Vmstat` 命令输出中的主要的列。

正如你看到的那样，`vmstat` 命令非常全面，从内存使用、进程状态到 CPU 使用百分数等各方面几乎给了系统一个完整的概述。

在此输出中，就内存使用而言，要查找的关键项目是 `po`、`so`、`b` 与 `w` 列。

假如 `po`（调出页面）与 `so`（换出）值很高，表明系统中内存严重短缺。在清单 22-20 中，系统中没有页面调出。系统中大约有 $2885 \times 4\text{KB}$ （4KB 是内存页面的容量），即 11.26MB 的可用空闲内存。

注意 你可以使用由 `vmstat` 命令产生的 `po` 与 `so` 输出确定分页与交换活动。应在一天中定期运行 `vmstat` 命令以便检查系统是否经常分页或交换。

假如你查看前面的输出，会发现 `b` 列显示了大量的阻塞进程。我通过使用 `sar -d` 命令做了进一步的研究，发现大量的阻塞进程是因为少量磁盘上正在进行繁忙的输入/输出。由于输入/输出正在等待被执行，所以造成进程被阻塞。假如 `vmstat` 输出显示了大量的分页或交换，你必

须鉴别内存密集型用户。

表22-5 vmstat输出中的主要列

列 名	描 述
procs	在如下显示的各种状态中的进程数
r	运行模式下的进程数
b	阻塞模式下的进程数
w	已交换的和正在等待进程资源的进程数
memory	实内存与虚拟内存的报告
avm	平均活动内存
free	页面中自由表的容量
page	在时间间隔中的页面出错报告
si	被换入的页面数
so	被换出的页面数
pi	调进页面的千字节数（检验）
po	调出页面的千字节数
cpu	CPU使用情况的报告
us	服务于用户请求所用的时间百分数
sy	服务于系统调用所用的时间百分数
id	空闲时间

22.4.3 寻找内存密集型用户

需要技巧才能在UNIX系统中监控内存的个别使用。大部分工具（例如 vmstat和sar）报告系统内存使用情况，但不报告单个进程的内存使用情况。实际物理内存的使用不容易获取的另一个原因是每一个进程被划分为文本与数据，文本页在进程之间共享，而数据页专用于进程。假如进程被依附于一个共享内存段，那么此偏差量必须被减去以便获得实际的专用内存（由进程使用的内存）。当文本页被共享时，会在所有进程的虚拟内存使用中报告。因此，假如你增加使用系统中所有进程的虚拟内存，报告中获得的数字将超过所有进程实际使用的虚拟内存量。

尽管如此，你可以假设当对Oracle系统中的内存进行监控时，所有的进程共享相同的文本页（也就是Oracle可执行文件）。假如所有的进程正在使用相同的可执行文件，那么所有进程的文本内存用量是相同的。剩余的内存是由进程使用的专用内存。在UNIX系统中，带有-l选项的ps命令可以用来确定每一个单独进程的内存使用情况。-l选项在ps输出的sz列中报告了内存使用情况。清单22-21中的ps命令可以用来监控系统中Oracle进程（前台和后台）的内存使用情况。sz列报告了进程核心映像的物理页面的容量。它包括文本、数据和堆栈空间。清单22-21中的输出按照sz列值由高到低的顺序显示排序后的ps命令，其中sz列是输出中的第7列。此输出对所有Oracle进程给出了使用的文本+数据+堆栈的容量。

注意 以前提到的sz列代表文本加数据再加堆栈的内存使用。Oracle文本页的容量可以在Oracle可执行文件中使用测量容量的命令来确定。在获得文本页的容量之后，理论上讲，假如你从虚拟内存容量中减去使用ps -ael命令获得的数值，你将得到每个进程的实际虚拟内存使用情况。然而，有时ps -el命令中的sz列不能报告实际的文本页容量；它仅报告进程需要的页面。这是由于有页面请求算法，在该算法中仅仅最需要的文本页才被进程使用，剩余的页面仅在需要时才从文件系统中取出。

你可以使用在top命令中的容量列，因为当top命令在SIZE列中就虚拟内存提出报告时，该命令报告的是总的文本页容量，而不是实际使用的页容量。在清单 22-21中，由进程25623号使用的虚拟内存的容量是8640K，这包括了整个文本页。

清单22-21 使用top命令输出获得虚拟内存容量

```
System: arms1                               Tue Sep  9 19:41:51 1997
Load averages: 4.87, 4.74, 4.44
211 processes: 204 sleeping, 7 running
Cpu states:
CPU  LOAD   USER    NICE     SYS    IDLE   BLOCK  SWAIT   INTR   SSYS
0    4.98  88.1%   0.0%   11.9%   0.0%   0.0%   0.0%   0.0%   0.0%
1    4.75  79.2%   0.0%   20.8%   0.0%   0.0%   0.0%   0.0%   0.0%
-----
avg  4.87  83.2%   0.0%   16.8%   0.0%   0.0%   0.0%   0.0%   0.0%
Memory: 53592K (15892K) real, 72336K (25548K) virtual, 11848K free Page# 1/20
CPU  TTY  PID USERNAME  PRI NI  SIZE  RES  STATE  TIME %WCPU %CPU COMMAND
0    ? 25623 arms      240 20 8640K 548K run   7:34 43.47 43.39 oracleor
1    ? 18128 oracle    241 20 9184K 1024K run  177:07 31.29 31.23 oracleor
0    ? 19523 oracle    240 20 9644K 1552K run   60:48 26.19 26.15
oracleor
```

注意 如果试图使用ps命令寻找容量值，会得到下述结果：

```
$ ps -ael | grep 256,23
1 R    105 25623 25622 254 241 20 3909b00 1524
    9:20 oracle
```

这个ps命令显示虚拟内存的容量是1524×4K=6096K，小于top命令的8640K，这是因为实际使用的文本页容量低于总文本页容量。

当计算单个进程的实际内存使用情况时，请记住上面所说的这一点。

清单22-22中的输出确定了Oracle进程的pid。当在以前的小节中使用SQL语句获得pid时可以发现进程的sid，并且在会话层可以完成进一步的调查。这将在下面的输出中进行说明。

清单22-22 用来查找进程内存使用情况的ps命令

```
$ ps -ael | grep oracle | sort -n -r -k 10
1 R    129 28060 28058 248 240 20 3932f00 2400 - ? 15:24 oracle
1 R    104 28049 1 246 239 20 3941080 2001 - ? 10:55 oracle
1 S    104 25751 1 0 156 20 38e1f80 1939 461b34 ? 19:02 oracle
1 S    104 15018 1 0 154 20 37c2980 1915 3d90e38 ? 198:25 oracle
1 S    104 25861 1 59 148 20 3807d00 1711 2194bac ? 32:42 oracle
1 S    104 25743 1 0 156 20 3e38380 1658 461b1c ? 23:52 oracle
1 S    104 14103 1 0 154 20 2829900 1653 3d4e338 ? 109:12 oracle
1 R    104 25739 1 255 241 20 37c2d80 1623 - ? 142:21 oracle
1 S    104 27772 1 0 156 20 372ba80 1603 461b04 ? 9:11 oracle
1 S    105 22744 22743 0 154 20 3941e80 1578 3d61da2 ? 38:59 oracle
1 S    129 25731 25727 0 154 20 335f700 1538 38c47a2 ? 0:00 oracle
1 S    104 19722 1 0 154 20 2845380 1525 3d6cb38 ? 0:05 oracle
1 S    129 28055 28053 0 154 20 3780f00 1522 3dda0a2 ? 0:00 oracle
1 S    125 25629 1 0 156 20 3738680 1511 461aa4 ? 0:01 oracle
1 S    104 7258 1 0 154 20 38e6100 1498 391e838 ? 0:00 oracle
1 S    125 25623 1 0 156 20 294f600 1493 461a8c ? 0:38 oracle
1 S    125 25621 1 0 156 20 2576080 1490 461a84 ? 1:15 oracle
1 S    125 25627 1 0 156 20 3807300 1483 461a9c ? 0:03 oracle
1 S    125 25625 1 3 156 20 3937000 1483 461a94 ? 2:00 oracle
1 S    125 25649 1 0 156 20 2958980 1478 461af4 ? 0:10 oracle
1 S    125 25647 1 0 156 20 3903b00 1478 461aec ? 0:10 oracle
1 S    125 25645 1 0 156 20 2845a80 1478 461ae4 ? 0:11 oracle
```

1 S	125	25643	1	0	156	20	3354400	1478	461adc ?	0:13	oracle
1 S	125	25641	1	0	156	20	391d700	1478	461ad4 ?	0:13	oracle
1 S	125	25639	1	0	156	20	3739b80	1478	461acc ?	0:13	oracle
1 S	125	25637	1	0	156	20	2852780	1478	461ac4 ?	0:14	oracle
1 S	125	25635	1	0	156	20	2888580	1478	461abc ?	0:15	oracle
1 S	125	25633	1	0	156	20	3663e80	1478	461ab4 ?	0:15	oracle
1 S	125	25631	1	0	156	20	3e40c80	1478	461aac ?	0:15	oracle
1 S	125	25619	1	0	156	20	27fb500	1477	461a7c ?	0:01	oracle

单个进程对内存的使用情况也可以通过使用 V\$SESSTAT 系统视图得到监控。要了解你正在查看的那部分内存，需要研究内存如何在 Oracle SGA 中组织起来。SGA 是一个包括共享池、块缓冲区和重做日志缓冲区的共享段。共享池是 SGA 中的一个区域，它包含了诸如共享 SQL 区与数据字典缓存的构造。

对于正在服务器上执行的每条 SQL 语句来说，有一个共享部分（共享 SQL 区）和一个专用部分（专用 SQL 区）。执行同一段 SQL 代码的两个用户使用相同的共享 SQL 区，但是每个用户有他们各自的专用 SQL 区。

共享 SQL 区包含语法分析树和每条 SQL 语句的执行计划。该区域的大小取决于语句的复杂性。每一个这样的 SQL 区的容量可以用 V\$SQLAREA 视图的 sharable_memory 列来确定，V\$SQLAREA 视图包含了共享池中的所有 SQL 区。每一个执行相同 SQL 语句的会话都共用相同的共享 SQL 区。

专用 SQL 区是一个内存区域，它包含了诸如绑定信息与运行期缓冲区的数据。每一个执行 SQL 语句的会话都有它自己的专用 SQL 区，该区域用于它所执行的每条 SQL 语句。概括地说，对于正在执行同一条 SQL 语句的多个会话来说，有一个共享的 SQL 区并且每个会话各自拥有的多个专用 SQL 区。专用 SQL 区进一步划分为一个持久区和一个运行时间区。

你需要监控内存使用的那部分内存区称为程序全局区（PGA）。程序全局区包含单个进程的数据和控制信息。程序全局区还被称为进程全局区（Process Global Area）。程序全局区的内容取决于连接类型是专用的还是多线程的。

在两种体系结构中，程序全局区总是包含堆栈空间（stack space），它是用来存储会话变量、数组和其他信息的内存。

在专用服务器选项（dedicated server option）中，程序全局区存储与用户的会话有关的辅助信息；也就是说，在这里存储专用 SQL 区和其他的会话信息。在多线程选项（multithreaded option）中，该信息存放在 SGA 中。

你能够监控程序全局区的使用情况并估计系统中每个用户对内存的需求。可以使用 V\$SESSTAT 视图了解对 PGA 内存的使用。statistic# value=20 对应于程序全局区内存使用。使用清单 22-23 中的脚本对内存密集型用户进行监控。

清单 22-23 使用 V\$SESSTAT 查看程序全局区内存的使用情况

```
col value format 99999 heading "Memory |Kb"
```

```
Select sid,
       value/1024 value
  From v$sesstat
 Where statistic# =20
 Order By value desc
```

The following is the output of the above SQL statement.

Memory SID	Kb
31	1165
153	754
142	733
101	686
70	362
73	336
26	306
60	298
82	297
53	297
123	275
44	272
110	270

在确定了消耗大量程序全局区的会话后，你可以继续前面小节中提到的步骤以便发现每个会话正在做的工作。

Windows NT对内存的寻址能力最大可以达到 4GB；其中2GB保留用于操作系统，剩余的2GB用于其他的应用程序。

为了在你的Windows NT服务器中查找可用的内存，打开 Windows NT的资源管理器并选择关于Windows NT的帮助。图22-4所示的窗口出现在你的屏幕上，其中以千字节为单位显示了可用的内存容量。

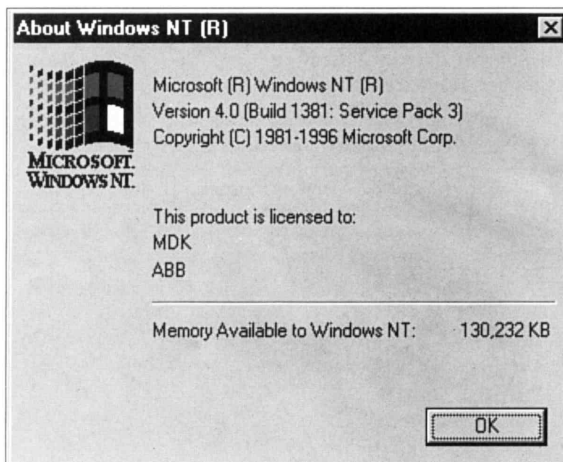


图22-4 本系统中可用的内存容量

22.4.4 缓冲区高速缓存

最重要的内存（SGA）因素是调整数据库缓冲区高速缓存以使用户能够尽可能地从内存中读信息，而不是从磁盘中读信息。假如有效地调整缓冲区高速缓存，那么就可以保证数据库不会受到输入/输出的限制并避免由输入/输出限制所导致的速度变慢。你可以查询V\$SYSSTAT表或使用Oracle 8性能监控程序得到有关缓冲区高速缓存使用的实时性能信息。Oracle 8缓冲区高速缓存的物理读/获取计数器显示了缓存遗漏率，它显示了涉及物理磁盘活动的数据库读取率。

图22-5中显示的当前遗漏率是 1.9%。它不应超过 20%，因为你应总是使缓存命中率保持

在80%以上。

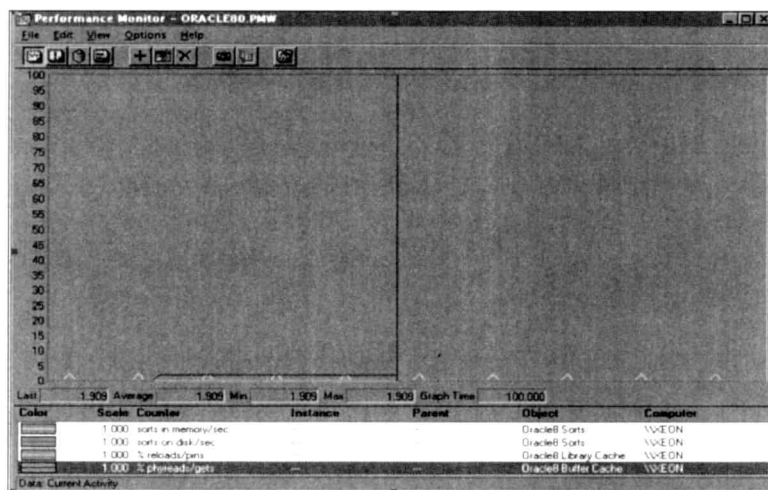


图22-5 监控缓存遗漏率

你可以执行如下查询以得到有关缓冲区高速缓存使用的实时性能信息：

```
select trunc((1-(sum (decode (name, 'physical reads',value,0))/
(sum(decode(name,'db block gets',value,0)) +
(sum(decode(name,'consistent gets',value,0)))))
)*100) "Buffer Hit Ratio"
from v$sysstat;
```

22.4.5 共享池

共享池由库缓存和数据字典缓存组成。

1. 库缓存

库缓存包含所有新近执行过的 SQL 语句，例如存储的过程、函数、包、触发器以及 PL/SQL 块。

执行如下查询，得到命中率的值来确定库缓存的性能：

```
select sum ( pinhits - reloads )/sum ( pins ) "HIT RATIO",
       sum (reloads) / sum (pins) "RELOAD PERCENT"
from v$librarycache;
```

以上 SQL 语句的输出如下：

```
HIT RATIO    RELOAD PERCENT
-----
.99729584    .00005962
```

前面例子中 99.729% 的命中率和 0.0059% 的重新加载率（遗漏）是可以接受的。你应设法使命中率保持在 85% 以上，而重新加载率保持在 2% 或 2% 以下。库缓存重新加载率可以在 Oracle 8 PerfMon 中使用计数器 Oracle 8 库缓存：% 重新加载/针查看。

2. 数据字典缓存

数据字典缓存是有效数据库操作的中心。V\$ROWCACHE 表包含有关数据字典缓存的信息。如果没有数据字典缓存，每次对查询进行语法分析时，Oracle 就必须从磁盘中得到元数据信息。

执行下列查询以确定缓存遗漏率：

```
select sum(gets) "Dict. Gets",  
       sum(getmisses) "Get Misses",  
       sum(getmisses)/sum(gets)*100 "Ratio (Ideal <15%)"  
from v$rowcache;
```

以上SQL语句的输出如下：

```
Dict. Gets Get Misses Ratio (Ideal <15%)  
-----  
204324      960      .46984202
```

对于优化性能而言，字典缓存遗漏率应当低于 15%。你还可以使用 Oracle 8 PerfMon 计数器 Oracle 8 数据字典缓存：%遗漏/获取得到相同信息。

22.4.6 递归调用

只要表、索引或回滚段需要更多的数据库空间时就会出现递归调用。递归调用影响数据字典缓存的性能。当数据库对象没有更多的可用空间并且引发动态扩展时，Oracle 必须暗中执行递归调用以使用新区间的信息更新数据字典。

执行如下查询以确定递归调用的数量：

```
select * from v$sysstat  
where name = 'recursive calls'  
/
```

你还可以使用 Oracle 8 PerfMon 并将计数器 Oracle 8 动态扩展：递归调用/秒用图表示，如图 22-6 所示。

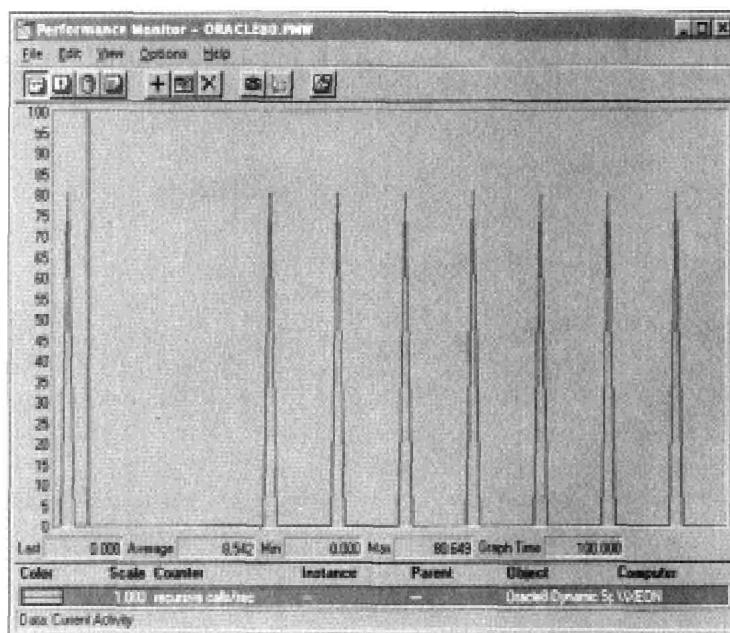


图22-6 监控数据库的递归调用

22.4.7 分页

在内存中，页面是 Windows NT 一次可以使用的输入/输出的最小值。当一个应用程序需要更多的内存时，Windows NT 在内存中寻找当前没有被使用的页面。然后执行页面调出，将非

活动页面移动到磁盘上一个特别指定的文件中，以便腾出空间给更多的活动页面。假如再次需要页面文件中的一个或多个页面，Windows NT执行页面调入，将这些页面从页面文件中移回内存中。基于Intel的NT系统的缺省页面容量是4K，基于Alpha的NT系统的缺省页面容量是8K。

你可以通过在Windows NT PerfMon中查看计数器内存：页面输入 /秒和内存：页面输出 /秒来观察页面调入和页面调出的情况，如图 22-7所示。

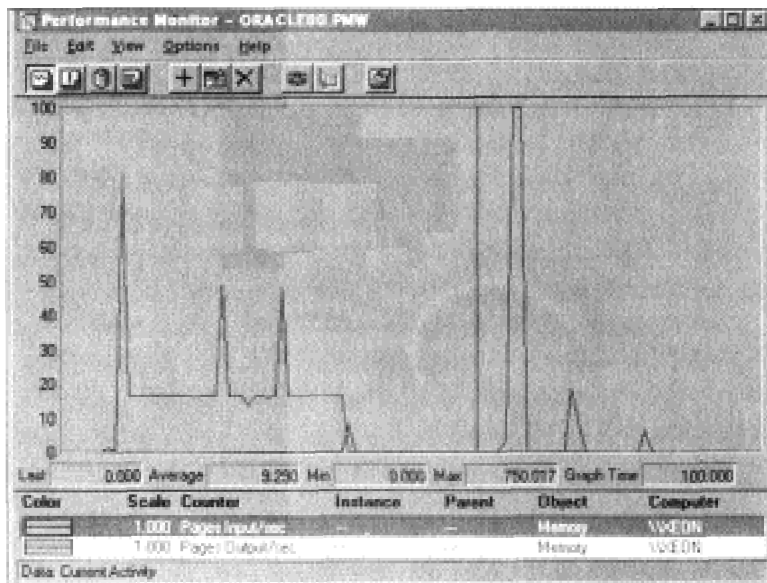


图22-7 监控Windows NT分页活动

Oracle 8初始化参数PRE_PAGE_SGA可以尽可能地把SGA保留在内存中。当PRE_PAGE_SGA被设置为YES，Windows NT尽最大努力把构成SGA的页面保存在物理内存中，并避免把SGA页面写入分页文件中。Windows NT分页文件PAGEFILE.SYS缺省情况下驻留在C：驱动器上。页面文件的总容量至少同物理内存一样大。

22.4.8 分页错误

当一个程序运行时，它所使用的物理内存中的页面总数被称为该进程的工作组（working set）。假如一个进程需要对当前不在它的工作组中的页面进行存取时，就会发生页面错误。这可以是一个软件错误（所需的页面在内存中的其他地方发现）或硬件错误（所需的页面在磁盘上并且必须被读入内存中）。

使用Windows NT的任务管理器可以快速地鉴别产生大量页面错误的进程，如图 22-8所示。

下面的一些措施用于解决过多的分页问题：

- 使Windows NT分页文件更大。
- 停止不必要的Windows NT服务。
- 确认整个SGA能够装入物理内存。
- 删除不再使用的网络协议。
- 增加内存容量。

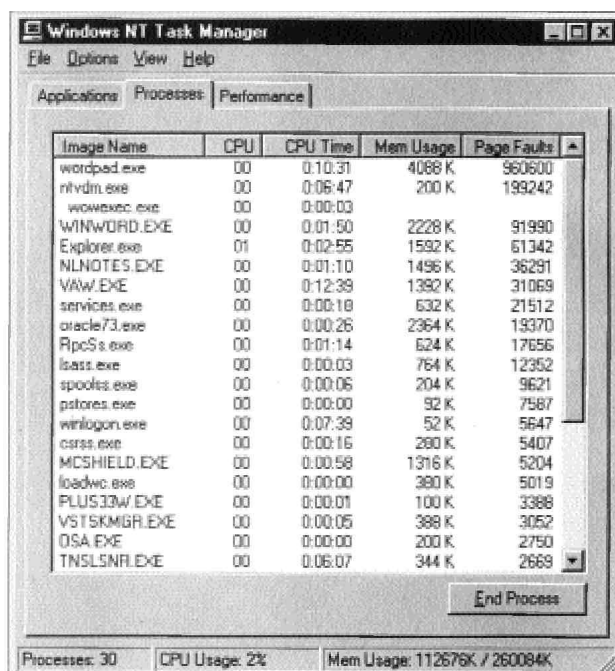


Image Name	CPU	CPU Time	Mem Usage	Page Faults
wordpad.exe	00	0:10:31	4088 K	960600
ntvdm.exe	00	0:06:47	200 K	199242
wwwexec.exe	00	0:00:03		
WINWORD.EXE	00	0:01:50	2228 K	91990
Explorer.exe	01	0:02:55	1592 K	61342
NLNOTES.EXE	00	0:01:10	1496 K	36291
VAVW.EXE	00	0:12:39	1392 K	31069
services.exe	00	0:00:10	632 K	21512
oracle73.exe	00	0:00:26	2364 K	19370
RpcSs.exe	00	0:01:14	624 K	17656
lsass.exe	00	0:00:03	764 K	12352
spoolss.exe	00	0:00:06	204 K	9621
psstores.exe	00	0:00:00	92 K	7587
winlogon.exe	00	0:07:39	52 K	5647
csrss.exe	00	0:00:16	280 K	5407
MCSHIELD.EXE	00	0:00:58	1316 K	5204
loadwc.exe	00	0:00:00	380 K	5019
PLUS32W/EXE	00	0:00:01	100 K	3388
VSTSKMGR.EXE	00	0:00:05	388 K	3052
OSA.EXE	00	0:00:00	200 K	2750
TNSLSNR.EXE	00	0:06:07	344 K	2669

Processes: 30 CPU Usage: 2% Mem Usage: 112676K / 260084K

图22-8 使用任务管理器识别页面错误

第23章 安全管理

本章要点：

- 用户验证
- 数据库权限管理
- Oracle企业安全管理器
- 监控数据库资产
- 保护数据完整性
- Oracle 8i因特网安全特性
- 防火墙支持
- 好的粒状存取控制
- 数据库资源管理器
- 硬件安全
- 恢复丢失的数据

Oracle 8i有一套给人留下深刻印象的全新的安全特性。安全的宗旨是机密性、完整性和可用性。Oracle已经增强了所有这些领域中的安全性。作为 Oracle安全模型基础的安全原理是基于最小特权原则。此原则认为用户应该仅有完成他的任务所必需的特权。这为一个开发安全策略的粒状方法做准备。这些特性可以被分成两大类：数据库安全与用于因特网处理的数据库安全。一个用于因特网的 Oracle数据库包括所有典型的数据库安全外加支持因特网的标准安全选项。因特网数据库需要特别的安全保障。因特网数据库与标准数据库的安全问题将在本章中讨论并详细阐述。

23.1 用户验证

用户在访问你的数据库以前必需被识别与验证。用户将被 Oracle识别，但是他们可以用3种不同的方法得到验证：数据库验证、外部验证或企业验证。另外，数据库和 Web服务器也可以被验证。这些内容将在稍后的 23.6节中讨论。

23.1.1 数据库验证

当创建用户和指定口令时使用数据库验证。这对于一个没有可以利用的辅助安全产品的小型用户团体来说是件好事情。其他类型的验证需要使用 external保留字来代替用户口令。

当创建用户时，必需为该用户选择口令。在用户口令上运用规则被叫做口令管理，一个公司应该有口令准则。强大的口令是不会轻易地被猜到的口令，它应超过 5个字节并不是一个在字典中可以找到的单词。如果口令是字典中的一个单词，那么一个计算机破坏者或许能够通过使用“强力攻击”的方法猜到该口令（强力攻击是一种方法：计算机破坏者使用一个用户标识并编写一个程序用来试验由字典产生的不同的口令）。口令在一定时间后也会过期且不能被同一个用户重复使用。

当Oracle使用数据库验证时便具有了提供口令管理的能力。这可以通过在一个环境资源文件中设置参数并把那个环境资源文件分配给一个用户的方式来实现。环境资源文件（profile）是一个规定了资源限度的数据库实体。当一个环境资源文件被分配给一个用户时，它在该用户上实施这些限制。可以使用企业管理器或 SQL*Plus创建环境资源文件。数据库一定有资源限制，环境资源文件资源限制能够受此影响。要做到这一点，在 init.ora文件中把 RESOURCE_LIMIT参数设置为TRUE。一个环境资源文件可以限制会话的数量、每个会话使用的CPU时间、CPU调用的次数、逻辑读的次数、每次调用的逻辑读的次数、空闲时间以及连接时间。环境资源文件可以防止计算机破坏者利用所有的资源以拒绝服务的攻击方式破坏计算机。

环境资源文件可以实施口令管理规则，你可以把它们作为选项选择使用：

- 用户帐户的锁定 —— 当一个用户多次注册失败后，该帐户能够被锁定一段指定的时间。
- 口令使用期与到期 —— 一个给定的口令在使用一段时间后过期，此时必须修改该口令。在口令过期后，用户会得到一个宽限期；假如用户不修改口令的话，那么该帐户将被锁定。数据库/安全管理员也可以把口令设置为过期状态。
- 口令历史 —— 口令历史选项检查每一个新近设定的口令以确保口令在指定长度的时间内或指定次数的口令修改中未重复使用。数据库管理员可以用 CREATE PROFILE 语句配置口令重复使用的规则。
- 口令复杂性验证 —— 复杂性验证检查口令的长度以使它更难被计算机破坏者所破解。缺省的口令复杂性验证程序要求每个口令符合以下原则：
 - 长度上至少为4个字符。
 - 不要与用户标识相同。
 - 至少包含一个字母字符、一个数字字符和一个标点符号。
 - 不要与简单单词的一个内部列表上的任何单词匹配，例如欢迎、帐户、数据库、用户等等。
 - 与以前的口令至少有3个字符不同。
- 数据库管理员验证 —— 由于数据库管理员任务（例如关闭或启动一个数据库）的特权性，所以他们需要一个更安全的验证模式。可以使用操作系统或口令文件实现附加验证。
- 操作系统 —— 假如操作系统提供一种将用户分割到诸如 UNIX或NT组中的方法，那么 Oracle将推荐把数据库管理员放置在一个特殊的组中。这样就使 Oracle可以通过组ID的辅助验证知道某个用户是数据库管理员。
- 使用一个口令文件验证数据库管理员 —— 用于数据库管理员的口令文件是可选的，可以使用ORAPWD口令工具来设置。口令文件将把管理特权仅限于知道口令且已经被授予了一个特殊角色的用户。这些角色是 SYSOPER和SYSDBA：
 - SYSOPER使你能够执行STARTUP、SHUTDOWN、ALTER DATABASE OPEN/MOUNT、ALTER DATABASE BACKUP、ARCHIVE LOG和RECOVER并且包含了RESTRICTED SESSION特权。
 - SYSDBA包含了具有ADMIN OPTION的所有系统特权以及SYSOPER系统特权；它使你能够执行CREATE DATABASE和基于时间的恢复。

1. 使用ORAPWD

ORAPWD工具在操作系统的命令提示符下执行。使用如下步骤：

1) 使用ORAPWD工具创建口令文件：

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

filename是口令文件的实际文件名，password是以数据库管理员身份注册到数据库的口令，max_users是具有数据库管理员特权的用户的最大值。

2) 把初始化参数REMOTE_LOGIN_PASSWORDFILE设置为一个有效值。该参数有3个有效值：NONE、SHARED、EXCLUSIVE。假如口令文件不存在的话，NONE值使Oracle开始工作；这是缺省值。EXCLUSIVE值完成如下工作：

- 限制一个数据库使用这个口令文件。
- 允许被授予SYSDBA和SYSOPER角色的用户。

SHARED值使一个口令文件能够被多个数据库使用。然而，一个SHARED口令文件能够识别的用户只是SYS和INTERNAL，你不能把用户添加到一个SHARED口令文件中。

3) 只要口令文件处于EXCLUSIVE模式下，通过使用GRANT命令将数据库管理特权分配给合适的用户，用户就能被添加到口令文件中。下面便是一个例子：

```
GRANT SYSDBA TO jefferson
```

```
GRANT SYSOPER TO smith
```

注意 使用REVOKE命令从口令文件中删除用户。

2. 使用SYSDBA

特权SYSDBA使用户能够执行与OSDBA相同的操作。同样，特权SYSOPER使用户能够执行与OSOPER相同的操作。

特权用户可以通过使用如下的命令连接到数据库：

```
CONNECT jefferson/password@prddb.hq.com AS SYSDBA
```

假如被操作系统验证的用户满足了操作系统验证准则，那么使用口令文件不能阻止他们进行连接。

想要列举口令文件的成员，V\$PWFILERS视图显示了所有已被授予数据库的SYSDBA和SYSOPER系统特权的用户。

23.1.2 外部验证

外部验证依赖于操作系统或网络验证服务。尽管Oracle仍然对用户进行识别，但是外部验证等于在Oracle外放置了用于口令管理和用户验证的控制。此类登录不需要数据库口令。要使用该选项，在数据库init.ora文件中设置OS_AUTHENT_PREFIX参数。这告诉Oracle任何同此值有相同前缀的用户将要在外部得到验证。例如，假如该值被设置为ops\$并且你有两个分别名为ops\$jones与smith的用户，那么Oracle不需要来自ops\$jones的口令，但是它需要来自smith的口令。此参数能够被设置为你所希望的任何前缀，甚至能够通过指定一组空的双引号被设置为一个空串。init.ora参数REMOTE_OS_AUTHENT必须被设置为TRUE（缺省值是FALSE）以使Oracle能够使用来自一个非安全连接的用户名。这样可以避免一个潜在的计算机破坏者伪装成一个合法的用户。

网络验证用 Oracle 高级安全（OAS）选项来实现并且能够用如下技术验证用户：

- 网络验证服务（例如 Kerberos 和 SESAME）—— 启用一个中央资源用于口令管理并且能够使用第三方的软件实施单独注册。用户在他将要使用的每个数据库中创建并且数据库特权被分配给该用户，但是口令是保留字 `external`。这告知 Oracle 仅识别用户并启用外部资源对口令进行验证。OAS 使用一个具有用户名、口令和主机名的验证服务器来验证口令。假如口令得到验证，那么用户便能够对 Oracle 数据库进行存取。
- 令牌设备 —— 令牌是用户建立一个到数据库的连接时必须的物理设备。令牌可以是在设备中产生的一次性数字口令，其容量是一个厚信用卡的容量。此数字口令必须用来同一个短的数字型个人识别号码（PIN）相连接。Oracle 服务器有一个添加到配置中的辅助安全服务用来跟踪令牌的口令。另一种方法是询问/应答。给用户发送一个数字（询问），用户在一台设备上键入该数字，该设备给出作为口令使用的另一个数字（应答）。
- 生物统计学设备 —— 使用一个对于个人来说是唯一的物理特征，当前能够用于 OAS 的是指纹扫描设备。用户指纹必须首先记录在系统中，然后该用户指定 Oracle 服务并把他的手指放置在指纹读取器上。在读取器上得到的指纹将与数据库中的指纹进行比较。

23.1.3 企业验证

企业验证启用一个中央资源用于口令管理并能够使用 Oracle 安全服务（OSS）实施单独注册。该用户被称为全局用户（global user）并且必须在每个使用全局口令的数据库中创建。这告诉 Oracle 仅识别这个用户并且启用 Oracle 安全服务验证该口令以及传送用户企业验证。假如口令得到验证，用户便能够访问 Oracle 数据库。Oracle 安全服务与 Oracle 企业管理器的接口集中了安全角色管理和企业验证。这使得当前正在被管理的用户具有全局同一性。

1. 企业角色

企业角色是一个或多个全局角色的容器。企业角色现在存储在一个因特网目录中，它们也可以存储于一个允许使用轻量目录访问协议（LDAP）的目录服务器。

全局角色不同于数据库角色，因为它们使你能够把验证信息跨多个数据库分配给（全局）用户。当一个全局用户登录到一个数据库时，全局角色便被动态地分配给该用户。全局角色必须首先被分配给 Oracle 安全服务器中的一个全局用户，然后全局角色具有与数据库服务器中的每个全局角色相关的特权。与全局角色相关的特权在不同的数据库中是不同的。

2. 表空间分配与使用

当创建用户时，你必须告诉 Oracle 你想把用户在数据库中创建的对象存储在哪里；如果一个存储子句没有指定放置对象的位置，就放置在用户的缺省表空间（default tablespace）。应当指定缺省表空间以防止诸如表或索引的数据库对象在系统表空间中创建；假如用户没有在数据库中创建对象的特权，你可以把缺省值设置为系统表空间。如果用户在系统表空间中创建对象，会形成表空间碎片或空间溢出。临时表空间实际上是一个排序工作区并且由 SQL 语句使用（例如 ORDER BY 与 GROUP BY）。当用户建立一个索引时也使用临时表空间。由于与数据字典的争用不断增加，所以你应当指定一个临时表空间，而不是系统表空间。

表空间定额限制了用户在表空间中创建的数据库对象的大小。缺省情况下没有定额容量限制，但是假如用户已经在一个表空间中分配了数据库对象并且想要限制该表空间的使用，那么就把那个用户的表空间的定额设置为 0。有了此限制，在那个表空间中的当前对象不能被

分配任何更多的空间，但是它们仍然在表空间中。

23.2 数据库权限管理

特权可以是对象特权，也可以是系统特权。有超过 60种不同的系统特权用于用户在数据库中执行管理活动（参见表 23-1）。

表23-1 系统特权

特 权	所能实现的操作
分析	
ANALYZE ANY	分析数据库中的任何表、簇或索引
审计	
AUDIT ANY	审计数据库中的任何模式对象
AUDIT SYSTEM	启用与停用语句和特权的审计选项
簇	
CREATE CLUSTER	在自有的模式中创建一个簇
CREATE ANY CLUSTER	在任何一个模式中创建一个簇；操作类似于 CREATE ANY TABLE
ALTER ANY CLUSTER	改变数据库中的任何一个簇
DROP ANY CLUSTER	删除数据库中的任何一个簇
数据库	
ALTER DATABASE	改变数据库；不管操作系统的特权，经由 Oracle把文件添加到操作系统中
数据库链接	
CREATE DATABASE LINK	在自有模式中创建专用数据库链接
索引	
CREATE ANY INDEX	在任何表的任何模式中创建一条索引
ALTER ANY INDEX	改变数据库中的任何索引
DROP ANY INDEX	删除数据库中的任何索引
库	
CREATE LIBRARY	在自有模式中创建调出库
CREATE ANY LIBRARY	在任何模式中创建调出库
DROP LIBRARY	删除自有模式中的调出库
DROP ANY LIBRARY	删除任何模式中的调出库
特权	
GRANT ANY PRIVILEGE	授予任何系统特权（不包括对象特权）
过程	
CREATE PROCEDURE	在自有模式中创建存储的过程、函数和包
CREATE ANY PROCEDURE	在任何模式中创建存储的过程、函数和包（这要求用户还要有ALTER ANY TABLE、BACKUP ANY TABLE、DROP ANY TABLE、SELECT ANY TABLE、INSERT ANY TABLE、UPDATE ANY TABLE、DELETE ANY TABLE或GRANT ANY TABLE特权
ALTER ANY PROCEDURE	编译任何模式中的任何存储的过程、函数或包
DROP ANY PROCEDURE	删除任何模式中的任何存储的过程、函数或包
EXECUTE ANY PROCEDURE	执行任何过程或函数（独立的或成组的），或在任何模式中引用任何公共包变量
环境资源文件	
CREATE PROFILE	创建环境资源文件
ALTER PROFILE	改变数据库中的任何环境资源文件
DROP PROFILE	删除数据库中的任何环境资源文件
ALTER RESOURCE COST	设置所有的用户会话中使用的资源开销

(续)

特 权	所能实现的操作
公共数据库链接	
CREATE PUBLIC DATABASE LINK	创建公共数据库链接
DROP PUBLIC DATABASE LINK	删除公共数据库链接
公共同义词	
CREATE PUBLIC SYNONYM	创建公共同义词
DROP PUBLIC SYNONYM	删除公共同义词
角色	
CREATE ROLE	创建角色
ALTER ANY ROLE	改变数据库中的任何一个角色
DROP ANY ROLE	删除数据库中的任何一个角色
GRANT ANY ROLE	授权数据库中的任何一个角色
回滚段	
CREATE ROLLBACK SEGMENT	创建回滚段
ALTER ROLLBACK SEGMENT	改变回滚段
DROP ROLLBACK SEGMENT	删除回滚段
会话	
CREATE SESSION	连接到数据库
ALTER SESSION	发出ALTER SESSION语句
RESTRICTED SESSION	当数据库利用 STARTUP RESTRICT启动时进行连接（OSOPER与OSDBA角色包含此特权）
序列	
CREATE SEQUENCE	在自有模式中创建序列
CREATE ANY SEQUENCE	在任何模式中创建任何序列
ALTER ANY SEQUENCE	在任何模式中改变任何序列
DROP ANY SEQUENCE	在任何模式中删除任何序列
SELECT ANY SEQUENCE	在任何模式中引用任何序列
快照	
CREATE SNAPSHOT	在自有模式中创建快照（用户还必须具有 CREATE TABLE特权）
CREATE ANY SNAPSHOT	在任何模式中创建快照（用户还必须具有CREATE ANY TABLE特权）
ALTER SNAPSHOT	改变任何模式中的任何快照
DROP ANY SNAPSHOT	删除任何模式中的任何快照
同义词	
CREATE SYNONYM	在自有模式中创建同义词
CREATE ANY SYNONYM	在任何模式中创建任何同义词
DROP ANY SYNONYM	在任何模式中删除任何同义词
系统	
ALTER SYSTEM	发出ALTER SYSTEM语句
表	
CREATE TABLE	在自有模式中创建表。还使被授权者能在自有模式下的表中创建索引，包括那些用于完整性约束的索引（被授权者必须有表空间的定额或UNLIMITED TABLESPACE特权）
CREATE ANY TABLE	在任何模式中创建表（假如被授权者有 CREATE ANY TABLE特权并在另一个用户模式中创建了一张表，那么拥有者必须在那个表空间上有空间定额。表的拥有者不必具有 CREAT [ANY] TABLE特权）
ALTER ANY TABLE	改变任何模式中的任何表并编译任何模式中的任何视图
BACKUP ANY TABLE	在任何模式中使用表的导出工具执行一个增量导出操作
DROP ANY TABLE	删除或截断任何模式中的任何表
LOCK ANY TABLE	锁定任何模式中的任何表或视图

(续)

特 权	所能实现的操作
COMMENT ANY TABLE	对任何模式中的任何表、视图或列进行注释
SELECT ANY TABLE	对任何模式中的任何表、视图或快照进行查询
INSERT ANY TABLE	把行插入到任何模式中的任何表或视图中
UPDATE ANY TABLE	修改任何模式中的任何表或视图中的行
DELETE ANY TABLE	删除任何模式中的任何表或视图中的行
表空间	
CREATE TABLESPACE	创建表空间；不管用户有何操作系统特权，经由 Oracle把文件添加到操作系统中
ALTER TABLESPACE	改变表空间；不管用户有何操作系统特权，经由 Oracle把文件添加到操作系统中
MANAGE TABLESPACE	使任何表空间脱机，使任何表空间联机，开始和结束对任何表空间的备份
DROP TABLESPACE	删除表空间
UNLIMITED TABLESPACE	使用任何没有数量限制的表空间。此特权忽略了所分配的任何具体定额。假如被取消的话，被授权者的模式对象仍然保留，但是进一步的表空间分配被拒绝，除非这一分配是具体的表空间定额允许的。此系统特权仅可以授予用户，而不授予角色。一般而言，应分配具体的表空间定额，而不授予此系统特权
事务	
FORCE TRANSACTION	强迫提交或回滚本地数据库中悬而未决的自有的分布式事务
FORCE ANY TRANSACTION	强迫提交或回滚本地数据库中悬而未决的任何分布式事务
触发器	
CREATE TRIGGER	在自有模式中创建触发器
CREATE ANY TRIGGER	在任何模式中创建与任何模式的任何表相关的任何触发器
ALTER ANY TRIGGER	启用、停用或编译任何模式中的任何触发器
DROP ANY TRIGGER	删除任何模式中的任何触发器
用户	
CREATE ANY USER	创建用户；分配任意表空间上的定额，设置缺省和临时表空间，指定一个环境资源文件（在 CREATE USER语句中）
BECOME ANY USER	成为另一个用户（这是任何一个执行完全数据库导入的用户所需要的）
ALTER USER	改变其他用户：修改任意用户的口令或验证方法，分配表空间定额，设置缺省或临时表空间，在 ALTER USER语句中指定环境资源文件与缺省角色（不必改变自有口令）
DROP USER	删除另一个用户
视图	
CREATE VIEW	在自有模式中创建视图
CREATE ANY VIEW	在任意模式中创建视图。要在另一个用户模式中创建视图，你必须具有CREATE ANY VIEW特权，拥用者必须在该视图引用的对象上具有所需的特权
DROP ANY VIEW	删除任意模式中的任意视图

你应把这些特权仅授予管理数据库的用户。对象特权允许对数据库对象的存取与维护；此类特权用于终端用户。对象特权（参见表 23-2）能够直接分配给用户，或者该特权被授予一个角色，然后该角色被授予给用户。

表23-2 对象特权

对 象	所能使用的SQL语句
ALTER	ALTER对象（表或序列）
DELETE	DELETE FROM对象（表或视图）

(续)

特 权	所能实现的操作
EXECUTE	EXECUTE对象（过程或函数）；引用公共包变量
INDEX	CREATE INDEX ON对象（仅有表）
INSERT	INSERT INTO对象（表或视图）
REFERENCES	在对象（仅有表）上定义一个 FOREIGN KEY 完整性约束的CREATE或ALTER TABLE语句
SELECT	SELECT...FROM对象（表、视图或快照）；使用序列的 SQL语句
UPDATE	UPDATE对象（表或视图）

23.2.1 理解安全角色

角色（role）是一个数据库实体，该实体是一个已命名的特权组。这样就在特权与角色之间创建了一个多对一关系，所以对于一个角色可有许多特权。角色是数据库中唯一的数据库实体并且不被用户所有。你可以使用 CREATE ROLE语句创建一个角色，该角色不能与用户同名。假如用户有创建角色的特权的话，他们也可以创建角色。当你创建一个角色时，该角色成为你的缺省角色集的一部分。

1. 角色验证

角色的使用可以通过口令得到验证。这增强了角色的安全性并且使不知道口令的用户不能启用该角色。不幸的是，该角色的口令必须在应用程序的某处进行硬编码。当一个角色已经被授予用户但还不是该用户缺省角色集的一部分时，SET ROLE命令能够动态地启用该角色。只要用户在Oracle中创建一个会话，缺省角色集便自动被启用。在 Oracle以前的版本中，这是防止用户从除了应用外的其他任何地方存取数据的最好办法。例如，一个用户或许已经经由 payroll_role角色存取PAYROLL表。假如该角色是用户缺省角色集的一部分，那么该用户就可以使用软件而不是工资单应用程序修改数据。即便使用 SET ROLE命令，用户仍然可以使这个高级的安全特性失效。Oracle现在已经增加了安全应用角色的功能。这些新类型的角色仅能通过一个信任包被启用。信任包证实用户不是直接连接到数据库，而是正在使用应用。创建一个安全应用角色的语法如下：

```
CREATE ROLE payroll_role IDENTIFIED USING payroll_pkg.admin;
```

该角色是payroll_role，它正在使用信任包payroll_pkg.admin。

2. 通过操作系统的角色验证

下面的语句创建一个名为 ACCTS_REC的角色并需要操作系统验证它的使用：

```
CREATE ROLE role IDENTIFIED EXTERNALLY;
```

仅在操作系统可以动态地将操作系统特权（OSP）与应用链接时才能实现操作系统角色验证。当用户启动一个应用时，操作系统把一个OSP授予给该用户。被授予的OSP对应于与该应用有关的角色。此时，应用能够启用应用角色。

如果一个角色被操作系统授权，你必须在操作系统级上配置每个用户的信息。此项操作依赖于操作系统。

数据库有对于用户可以具有的作为缺省角色集一部分的角色数量的软限制。如果用户具有的角色数多于限制值，那么在登录时，他会收到一条错误信息。如果发生了此错误，在 init.ora文件中检查MAX_ENABLED_ROLES；或许需要增加该值。一个角色还可以被授予给

另一个角色，这样被授予的角色的所有特权可以得到继承。

23.2.2 理解管理

管理员仅有他们需要的用于数据库管理的权限。一个数据库环境中的主要管理任务可以被分为两个方面：安全管理与数据库管理。安全管理员（ security administrator）有权创建、改变并删除用户以及维护安全角色和用户环境资源文件，但是他没有数据库管理员的特权，例如打开或关闭一个数据库。

当你把一个角色授予给一个管理角色特权的用户时，你必须用 ADMIN选项来完成这项操作。该选项使用户能够执行管理操作，例如改变或删除角色、使用 ADMIN选项把角色授予给其他用户。除了管理员以外，该选项不应给任何人。

23.3 Oracle企业安全管理器

Oracle企业安全管理器使管理员能够使用一个工具来管理遍及该企业的 Oracle数据库安全的所有方面。该工具允许你创建一个用户、为用户分配缺省表空间和临时表空间以及他们的口令。一旦创建了用户，那么随之为他们分配特权。同全局角色一样，也能够在这里创建和维护非企业角色。单一的控制台使你能够在一个因特网目录中创建用户、在多个 Oracle 8i数据库中创建用户、创建并管理企业角色。要想了解此产品的更多信息，参见第 27章。

23.4 监控数据库资产

Oracle能够审计并记录数据库中发生的活动。你可以使用 AUDIT SQL命令允许审计操作或使用NOAUDIT SQL命令禁止审计操作。有三种类型的审计操作：登录尝试、对象存取（具体对象上的具体语句）和数据库操作（具体的系统特权和语句，不考虑对象）。

任何一条命令，不管是成功的还是不成功的都可以在这些类别中进行审计。要创建审计系统视图，以用户SYS的身份运行CATAUDIT.SQL文件脚本并设置init.ora参数AUDIT_TRAIL。你可以把AUDIT_TRAIL设置为写入数据库或一个操作系统文件：AUDIT_TRAIL=DB写入数据库，AUDIT_TRAIL=OS写入一个操作系统文件。为了使新的 init.ora参数生效，你必须重新启动Oracle实例。假如你把AUDIT_TRAIL参数设置为DB，那么所有已审计的活动将被写入SYS.AUD\$表中。你应将SYS.AUD\$表的表空间存储参数从系统表空间修改为审计目的而创建的表空间。你能够从SQL*WORKSHEET或SQL*Plus中发出所有的审计命令。

23.4.1 审计登录

AUDIT ANY特权是发出审计命令所必须的。使用 AUDIT SESSION命令完成审计登录。此命令可以用来对所有成功和不成功的与数据库建立连接的尝试进行审计。使用 AUDIT SESSION WHENEVER NOT SUCCESSFUL命令仅审计不成功的尝试。使用AUDIT SESSION WHENEVER SUCCESSFUL命令仅审计成功的尝试。如果审计数据存储在 SYS.AUD\$表中的话，你可以生成审计报表。使用 DBA_AUDIT_SESSION视图报告登录尝试：

```
SELECT
os_username, /* O/S user name */
username, /* Oracle user name */
to_char(timestamp,'DD-MON-YY HH24:MI'), /* Login time */
```

```
to_char(logoff_time, 'DD-MON-YY HH24:MI') /* Logoff time */
FROM dba_audit_session
ORDER BY os_username;
```

```
OUTPUT TO QUERY
```

```
OS_USERNAME    USERNAME    TIMESTAMP    LOGOFF
JOHNES         JOHNES     24-MAR-97 18:00    14-FEB-97 18:05
```

23.4.2 审计数据库操作

通过审计数据库操作，可以在语句和系统特权级上进行审计，而不必考虑具体的数据库对象。AUDIT语句的审计功能特性使你能够审计不止一条 SQL语句。例如，使用审计语句ROLE审计SQL语句CREATE ROLE、ALTER ROLE、SET ROLE和DROP ROLE。要审计系统特权，你必须指定该特权。系统特权ALTER DATABASE不包含在语句审计选项中，但是你仍然可以审计它，因为它是一个系统特权。有些语句审计选项使用与系统特权相同的名称。你可以把AUDIT语句用于用户、会话或存取。你可以进一步地定义它以便仅对成功或不成功的语句进行审计。可以被审计的语句见表 23-3。

表23-3 语句审计选项

语 句	选 项
OPTION	SQL STATEMENT
ALTER SYSTEM	ALTER SYSTEM
CLUSTER	CREATE CLUSTER, ALTER CLUSTER, TRUNCATE CLUSTER, DROP
CLUSTER	
DATABASE LINK	CREATE DATABASE LINK, DROP DATABASE LINK
INDEX	CREATE INDEX, ALTER INDEX, DROP INDEX
NOT EXISTS	因为指定的结构或对象不存在而返回错误的所有 SQL语句
PROCEDURE	CREATE [OR REPLACE] FUNCTION, CREATE [OR REPLACE] PACKAGE, CREATE [OR REPLACE] PACKAGE BODY, CREATE [OR REPLACE] PROCEDURE, DROP PACKAGE, DROP PROCEDURE
PUBLIC DATABASE LINK	CREATE PUBLIC DATABASE LINK, DROP PUBLIC DATABASE LINK
PUBLIC SYNONYM	CREATE PUBLIC SYNONYM, DROP PUBLIC SYNONYM
ROLE	CREATE ROLE, ALTER ROLE, SET ROLE, DROP ROLE
ROLLBACK SEGMENT	CREATE ROLLBACK SEGMENT, ALTER ROLLBACK SEGMENT, DROP ROLLBACK SEGMENT
SEQUENCE	CREATE SEQUENCE, DROP SEQUENCE
SESSION	连接和断开连接
SYNONYM	CREATE SYNONYM, DROP SYNONYM
SYSTEM AUDIT	AUDIT, NOAUDIT
SYSTEM GRANT	GRANT系统特权/角色TO用户/角色REVOKE系统特权/角色FROM用户/角色
TABLE	CREATE TABLE, ALTER TABLE, DROP TABLE
TABLESPACE	CREATE TABLESPACE, ALTER, TABLESPACE DROP TABLESPACE
TRIGGER	带有ENABLE、DISABLE和DROP子句的CREATE TRIGGER, ALTER TRIGGER, ENABLE或DISABLE, ALTER TABLE
USER	CREATE USER, ALTER USER, DROP USER
VIEW	CREATE[OR REPLACE] VIEW, DROP VIEW

23.4.3 审计数据库对象上的DML

你可以使用如下语法审计一个具体的模式对象：

```
AUDIT object_opt ON schema.object  
BY SESSION/ACCESS WHENEVER NOT/SUCCESSFUL;
```

你可以指定一个对象选项（例如插入或更新）或者使用关键字ALL来指定所有的对象选项。

```
AUDIT insert,update  
ON scott.emp_table  
WHENEVER NOT SUCCESSFUL;  
  
AUDIT ALL  
ON scott.emp_table  
WHENEVER NOT SUCCESSFUL;
```

23.4.4 管理审计

如果你选择在数据库中存储审计记录，那么最好每天运行审计表上的报告，然后删除审计数据以节省空间。审计表只能进行有限地访问，对于 SYS.AUD\$表的所有活动应通过使用下面的语句被记录：

```
AUDIT INSERT, UPDATE, DELETE, SELECT  
ON sys.aud$  
BY ACCESS;
```

审计可能在数据库中造成过多的系统开销，因此它是可选的。审计语句中的 BY SESSION子句使Oracle为在同一个会话中使用的相同类型的所有 SQL语句只写下一条单独的记录。在审计语句中的BY ACCESS子句使Oracle为每一条已审计的语句写下一条记录。假如你审计数据定义语言（DDL）语句，不管指定哪一条子句，Oracle都会自动地使用BY ACCESS。为了维护一个安全环境，你应当执行一个策略以决定所要审计的操作。

下面的AUDIT语句显示如何使用某些更具限制性的选项：

```
AUDIT statement_opt/system_privileges  
BY user (optional)  
BY session/access WHENEVER NOT/SUCCESSFUL;
```

下面的代码示例显示了如何审计与角色和会话有关的语句：

```
AUDIT role;  
AUDIT session whenever not successful;.
```

23.5 保护数据完整性

你可以通过使用安全用户连接与加密算法来保护数据完整性。当用户注册到一个 Oracle数据库时可以对口令进行加密。使用一个修改的 DES（数据加密标准）算法对口令进行加密。密码通过在客户机上设置一个环境变量以及在数据库服务器上设置 init.ora参数来打开。你必须把客户机的环境变量ORA_ENCRYPT_LOGIN设置为TRUE。

注意 此变量根据操作系统而变化。你还必须在数据库服务器上把 init.ora参数 DBLINK_ENCRYPT_LOGIN设置为TRUE。

当一个数据库会话被初始化时，Oracle对用户标识对应的口令进行加密，但是它不对口令的修改进行加密。

计算机破坏者能够通过修改在网络上传播的数据或 DML损害数据的完整性，这会使数据库的逻辑遭到破坏。当数据库的完整性出现问题时，重新恢复的代价很大并且非常花费时间。

把你的所有数据加密对潜在的安全破坏来说是理想的解决方案，因为如果数据被动地受到计算机破坏者的监控，他们不能使用这些数据，可能会去寻找容易一些的破坏对象。Oracle的高级安全措施（OAS）提供了辅助的保密措施。Oracle的高级安全措施当前提供了两种加密算法：RSA和DES，它们具有不同的密钥长度。对于在美国和加拿大使用的产品，可以使用56位的RSA RC4、56位的DES和128位的RSA；对于美国和加拿大以外的出口产品，可以使用40位的RSA RC4和40位的DES40、3DES。数据完整性通过使用密码校验和得到保护，这个密码校验和使用MD5算法或SHA算法（安全哈希算法）。这将保证数据不会在它离开客户工作站与到达数据库服务器之间的这段时间里被篡改。

23.6 Oracle 8i因特网安全特性

Oracle 8i高级安全选项提供了一组给人深刻印象的安全特性，这些安全特性的设计考虑到了因特网标准。今天的体系结构是多层的并可能通过许多网络连接。这造成了一个令人担心的安全问题。存在连接到应用服务器的瘦客户以及连接到数据库服务器的应用服务器。数据库服务器不仅必须能确认用户就是他自己所说明的身份，而且应能确认应用服务器就是它自己所说明的身份。可以使用两种方法进行这些类型的验证：口令与数字证书。口令已经在前面的23.1节中讨论过了，所以这里要讨论数字证书。

23.6.1 使用数字证书

数字证书能够用来验证用户或机器。此类证书是 X.509版本3，是一个基于因特网标准的公用密钥基础结构（PKI）。证书包含如下内容：

- 可以被辨别的证书拥有者的名字。
- 可以被辨别的证书管理机构的名字。
- 证书拥有者的公用密钥。
- 证书有效的数据范围。
- 发行人的签名。
- 证书的序列号。

这些证书必须以网络安全套接字层协议（SSL）的方式使用。SSL也提供了跨越网络的数据加密和数据完整性检查。加密是保持数据私有的关键。当数据位于到数据库的途中时（参见图 23-1），数据完整性检查使计算机破坏者不能修改该数据。Oracle证书管理机构，或另一个证书管理机构在 Oracle目录服务器或另一个LDAP目录结构中发布“钱包”。Oracle使用一个“Oracle钱包”存储证书和私有密钥。Oracle钱包管理器是管理钱包内容的一个主要工具。

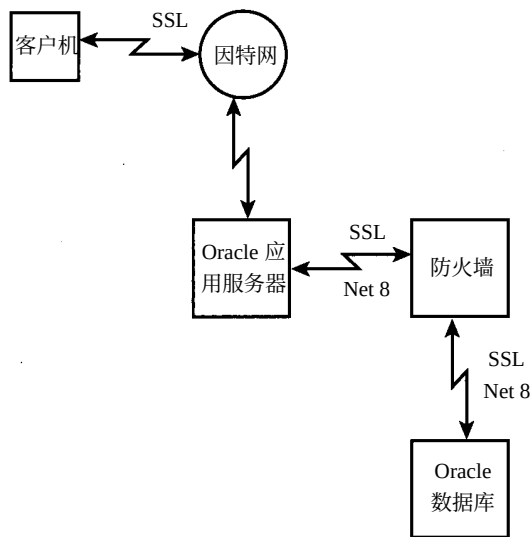


图23-1 使用数字证书进行通信

23.6.2 使用RADIUS协议的高级验证

既然每一台机器或用户可以有一个数字证书，那么 Oracle可以使用一种比口令更强大的验

证方法。RADIUS（远程拨号用户服务器验证）协议已经变成了一种因特网标准，Oracle 8i高级安全选项同任何符合RADIUS的安全服务器一起工作。此协议支持口令、令牌、生物学统计以及智能卡。

23.7 防火墙支持

防火墙用于防止公共因特网通信量进入到一个公司的专用网络中。它还用于保护与连接到因特网的服务器进行通信的服务器（参见图 23-2）。主要的防火墙厂商现在提供代理来使 NET 8 和 SQL*NET通信量能够穿过防火墙，这样为数据库服务器提供了保护，使之免遭计算机破坏者的攻击。计算机破坏者或许试图在数据库服务器操作系统上实施攻击，但是如果防火墙仅允许 NET8 和 SQL*NET通信量通过的话，那么这个攻击是不可能实现的，所以数据库服务器是安全的。

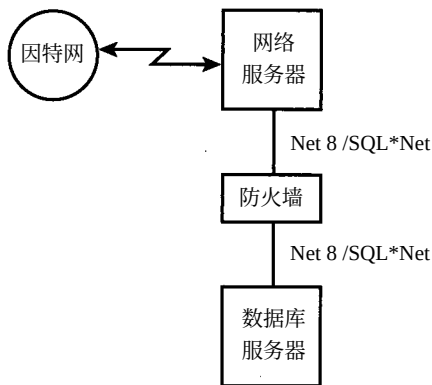


图23-2 防火墙支持

23.8 好的粒状存取控制

虚拟专用数据库保证用户仅能查看他有权使用的数据。这使得一个已存在的成品数据库能够在Oracle的担保下在因特网上被访问，Oracle保证终端用户仅能查看他自己的数据。此安全措施保存在数据库服务器上。这防止了它被诸如报告书写器等客户软件所忽视。一个虚拟专用数据库被相关的表或视图定义为一个安全策略。在此策略执行后，对表或视图的任何存取会引发数据服务器调用一个实施该策略的函数。在 SQL语句被执行以前，该函数返回被添加到WHERE子句中的代码。这个附加的代码包含了已经用于保证数据安全的存取规则。你可以为同一张表的不同 DML定义不同的策略（例如，雇员表的 INSERT可以有与雇员表的 DELETE不同的策略）。这些SQL语句被进行完整的语法分析与优化，然后存在于共享池中，在可用时由其他用户使用。

23.9 数据库资源管理器

数据库资源管理器管理多个 CPU以及多个实例能够从操作系统中获得的并行度。这是一个重要的安全与管理特性。从安全的角度出发，它能够防止对因特网上使用的生产环境进行的拒绝服务攻击。假如那个数据库同其他没有用于因特网的数据库共享系统资源的话，它是一个管理附加物。有四种机制用于管理数据库资源：

- 1) 在用户层分配资源消费者组，所有这些会话是该消费者组的一部分。
- 2) 资源计划包含资源消费者组或其他的资源计划；它们还包含一个如何划分资源的规范。
- 3) 资源分配方法是某些资源采用的策略。资源分配方法由资源计划和资源消费者组来使用。
- 4) 资源计划指令让管理员为计划分配成员，然后管理那些成员使用的资源。

通过创建一个因特网资源组并限制该组资源的数量，管理员能够更好地管理连接到因特网上的产品环境。他可以预防一个资源组首先执行并随之消耗掉操作系统中所有的计算机资源。

23.10 硬件安全

计算机硬件需要存放在一个限定的区域中，在这里，存取受到限制并且有合适的火灾控制机制。电流应当是清洁的，这意味着不会遭受电流起伏或停电的干扰。清洁的电流通常由不间断电源（UPS）提供。

23.11 恢复丢失的数据

在你的安全计划中，你必须指定如何恢复由于违反了安全性而丢失的数据。我认为：“你的系统实际上等于你最后的备份与你的恢复能力”。这句话概括了在数据库恢复方面所采用的基本原理。下面是一些涉及到备份的有益问题：

- 如何保证备份介质物理上的安全？
- 当备份完成时，备份介质是否要脱离站点以防止数据中心丢失的情况？
- 按照制造者的准则，备份介质被替换吗？
- 定期清理备份硬件吗？
- 备份是在恢复时是可读的、可用的？
- 存在一个显示备份介质在某日使用过以及介质已经备份了多少次的备份日志吗？

有两种方法备份 Oracle 数据库。第一种方法是通过使用一个叫做物理备份（physical backup）的操作系统备份；第二种方法是通过使用 Oracle 的导出工具对数据库进行备份，Oracle 导出工具创建的是逻辑备份（logical backup）。

23.11.1 操作系统备份

操作系统备份需要一个专用于操作系统的工具，它能够使数据库管理员通过使用一个操作系统恢复工具来恢复数据库。你可以在数据库关闭（冷备份）或打开（热备份）的情况下执行操作系统备份。热备份用在需要数据库可用性高的场合。热备份使表空间能够在联机时得到备份，或在脱机和不可用时用于事务处理。冷备份必须包括所有的 Oracle 数据文件、控制文件和联机重做日志文件（仅供参考，我把这些文件叫做操作系统备份文件集）。热备份被认为是部分备份，因为它仅备份所需的文件。

你可以在 ARCHIVELOG 模式和 NOARCHIVELOG 模式下运行数据库。ARCHIVELOG 模式指当联机重做日志被填充时，它们的内容被归档，这样便释放了当前的重做日志。归档重做日志用于恢复它们所记录的事务或在数据库恢复时重做它们。要执行一个热备份，数据库必须处于 ARCHIVELOG 模式。NOARCHIVELOG 模式意指联机重做日志在它们充满时并不归档，而是被最新的事务改写。因此，仅仅当前重做日志可用于恢复。假如不归档，你必须做一个操作系统文件集的备份，在数据库被恢复时，你必须恢复操作系统备份文件集中的所有文件。

冷备份的优点是恢复过程中的处理步骤最少。这通常意味着最小的出错几率，冷备份比热备份速度要快。

23.11.2 逻辑备份

使用 Oracle 导出工具的逻辑备份创建了一个文件，该文件含有用于重新创建数据结构和数

据库中所包含的数据的信息。该信息能够被存储在一个磁盘设备或磁带介质中。逻辑备份比物理备份花费的时间要长并且在准备恢复数据库时或许需要完成一些辅助的工作。有三种导出模式：

- 用户模式，备份用户所拥有的所有对象。
- 表模式，备份用户所拥有的指定的表。
- 完全数据库模式，备份数据库的所有对象。

你可以使用一个参数文件对数据库中的数据执行常规备份来控制导出工具。你可以在数据集合的三个层次上执行导出操作：增量、累积和完整。增量导出（incremental export）仅对那些自上次增量导出以来发生变化的数据进行备份。累积导出（cumulative export）仅从那些自上次累积导出以来发生变化的表中导出数据；此导出操作用于压缩增量导出。完全导出（complete export）导出数据库中包含的所有数据。由于所收集到的数据量较大，你只能有限地执行完全导出。

导出可以非常灵活，但是定期地执行导出是很重要的。一个好方案是在每月的第一天执行一次完全导出，每天执行一次增量导出，每周执行一次累积导出。

第24章 备份与恢复

本章要点：

- 备份策略
- 理解物理数据丢失与逻辑数据丢失
- 使用逻辑备份
- 使用冷物理备份
- 使用热物理备份
- 使用恢复管理器进行物理备份
- 从逻辑备份中恢复
- 使用物理恢复
- 测试策略
- 问题解决策略
- 项目：被破坏的归档日志的不同蕴涵

24.1 备份策略

正如大部分人已经知道的那样，数据备份几乎是任何计算机系统绝对必需的组成部分。磁盘头毁损、电路失效甚至数据中心的灾难性丢失都是现实情况，数据库管理员必须对此有所准备。所幸的是，Oracle是一个非常坚固的数据库系统，经过多年实践检验过的备份与恢复技术能够在不可避免的事情发生时保证你的数据安全。作为一名专业的数据库管理员，你必需采取措施保证你的RDBMS中包含的数据免遭各种类型的破坏。

假设在年底处理前的两周，你的数据库系统丢失了一个磁盘驱动器。CEO与CFO都是你办公室中的成员。他们想要知道系统备份需要花费多长时间并且还提醒你不能丢失财务系统中的任何数据。本章将给出你将要告诉他们的信息：“系统备份不会花费很长时间”以及“当然，不会丢失任何数据”。你还将学到一些备份系统的常见非应急用法。

提示 告诉你的用户与管理人员备份能够做什么和不能够做什么以及它一般要花费多长时间恢复系统。你有可能使所有人的希望成为现实。

还要指出的是，备份既可以在应急情况下用于数据恢复，也可以在受人工操纵的情况下用于数据恢复。在受人工操纵的情况下，你可以把数据从一个实例简单地移动到另一个实例、交换机器或重新组织一个数据库。

本章的重点是帮助你确定、设计并测试你的备份策略。恢复你的数据库的难度与实施备份策略直接有关。假如你对恢复环境不满意，你应当利用Oracle技术支持满足你的具体恢复要求，因为具体的细节根据不同的情况而变化。当你设计、实现并使用你的备份策略时，请记住如下这些要点。

- 设计你的备份与恢复策略。设计如何使用你的备份以便恢复你的数据库。
- 定期测试你的备份与恢复策略（对于新的维护成员特别重要）。

- 确保操作系统级的备份仍然脱离数据库服务器。数据库备份不能保护init.ora文件、Oracle设备或操作系统。
- 在重要的修改以前或以后执行适当的数据库备份。假如此修改包括删除段或表空间的话，执行适当的数据库备份就显得特别重要。
- 含有动态数据的表空间比多数静态表空间需要更为频繁地备份。
- 手头上保存较老的备份。有时几个月或几年也不会有一张表或一行已经被删除。
- 定期导出你的数据库进行附加的保护。
- 考虑适用于高利用率应用程序的分布式数据库备份。

24.2 理解物理数据丢失与逻辑数据丢失

数据丢失能够被分为物理丢失和逻辑丢失两类。物理数据丢失是指操作系统级的数据库组件的丢失，例如数据文件、控制文件、重做日志以及存档日志。引起物理数据丢失的原因可能是磁盘驱动器毁损，有人意外删除了一个数据文件或者是改写了任何关键的数据库文件所造成的配置变化。逻辑数据丢失是诸如表、索引或表记录的数据库级组件的丢失。引起逻辑数据丢失的原因可能是有人意外删除了不该删除的表、应用出错或一条 DELETE语句中不适当的WHERE子句。毫无疑问，Oracle能够实现物理数据备份与逻辑数据备份。尽管两种备份方案可以互相替代，但是你会发现有效的备份计划必须把两种方案都包含在内才能使你完全避免数据丢失。

物理数据备份是下面各项的拷贝：

- 数据文件。
- 控制文件。
- 可用的归档重做日志。

假如你丢失了一个磁盘驱动器或者需要来回移动一些数据文件，最直接的方法是恢复。物理备份通常按照预定的时间间隔运行以防止数据库的物理丢失。当然，如果你想保证能够把系统恢复到最后一次提交时的状态，你必须以物理备份为基础，同时你还必须有自上次物理备份以来累积的归档日志与重做日志。

物理备份可以进一步分为冷（cold）备份与热（hot）备份。当你有权关闭数据库足够长的时间以备份系统时可以使用冷备份。在数据库打开并可被用户使用时，热备份提供了一个物理备份。这使一个Oracle数据库能够1周7天、1天24小时地运行，同时仍然可以定时得到备份。如果你不必使数据库不间断地运行，那么最好使用冷备份。正如你将看到的那样，在使用热备份以前，它必须得到充分测试。

请注意术语“冷备份”是“脱机备份（offline backup）”的同义词，就像“热备份”是“联机备份（online backup）”的同义词一样。

备份频率与ARCHIVELOG模式

计划备份的两项主要工作是为你的数据库选择适当的备份频率以及决定是否在ARCHIVELOG模式下运行数据库。

当选择备份间隔时，总是应问自己“我能承担得起多少数据的丢失？”假如你的数据库没有运行在ARCHIVELOG模式下，你会丢失自最后一次备份以来的所有事务。假如你的业务需求使数据库能忍受丢失一周的事务，你需要每周至少备份数据库一次。

在成品OLTP系统中你不能失去任何事务。非常希望数据库能够在 ARCHIVELOG 模式下运行，因为即使你的最后一次物理备份发生在两天以前，它也能够把你的数据库完全恢复到最后一次提交时的状态。

即便在 ARCHIVELOG 模式下，你仍然必须对系统执行定期的物理备份。为了完全恢复一个数据库，你必须有自最后一次物理备份以来的所有归档重做日志。定期物理备份减少了用于存储归档重做日志的空间数量，同时也减少了由于归档重做日志的丢失或损坏而导致丢失事务的风险。

物理备份应用于相同的机器、相同的 Oracle 版本以及引起物理备份的实例上的恢复。因此，物理备份被认为是不可移植的。它们通常仅用于保护同一台机器和实例中的数据免遭数据丢失。

此规则的一个例外是你想要把一个数据库从一个系统完全转移到另一个系统。只要两种机器是相同的体系结构、相同的操作系统版本以及相同的 Oracle 版本，你就能把数据库的物理成分从系统 A 拷贝到系统 B。假如这些条件中的任何一个不满足或者你还不确定，那么就使用逻辑备份来移动你的数据。

逻辑数据备份通常是 SQL 语句的集合，这些 SQL 语句以一种 ASCII 格式重新创建数据库对象（数据库本身、表、索引、授权、角色等）和记录。

当你需要在实例之间移动指定的数据时，或者需要在不同的系统结构、操作系统版本或 Oracle 版本之间拷贝一个实例的全部数据时可以使用 Oracle 的逻辑备份系统。尽管除了物理备份以外确实需要定期的逻辑备份，但是逻辑备份通常必须由数据库管理员执行。

提示 Oracle 传统的备份工具通常不适宜用于归档数据库系统外的陈旧数据（特别是在数据能够被装入一个非 Oracle 系统的情况下）。来自 BMC 和 Platinum 厂商等的第三方工具为这些特殊情况的逻辑备份提供了很大的灵活性和可移植性。

在开发者环境中，应用开发者将与数据库管理员一同工作以设计一个逻辑备份策略。一般存在两个数据库拷贝，一个是成品数据库，另一个由开发者使用，用来调试他们的程序。开发数据库将与成品数据库定期地保持同步。

现在看一些实际问题。你必须保证你的数据库在 ARCHIVELOG 模式下运行并且正在运行归档程序以便把你的系统恢复到最后一次提交时的水平。下面的几节中列举了常见的备份要求和推荐使用的备份方法。本章的余下部分讲述由 Oracle 提供的备份与恢复方法的技术细节。

下面提供了几个例子对应说明了每一种备份系统的正确使用。

1. 物理备份

物理备份是所采用的最主要的备份方式。如果发生物理数据库丢失或崩溃，物理备份用于保证数据库在最小的数据丢失或没有数据丢失的情况下得到恢复。

2. 冷物理备份

冷物理备份提供了最简单和最直接的方法保护数据库免遭物理损坏或丢失。正确使用冷物理备份的示例如下：

- 数据库已经在一台数据库服务器上创建，你想按常规对数据库执行一个备份。数据库在备份过程中不必处于可用状态。
- 系统管理员将对你的磁盘进行升级，每张磁盘的容量从 4GB 升为 9GB。你需要备份系

统并在新驱动器中相同的目录结构下恢复它。

3. 热物理备份

热物理备份提供了相同的保护方法用于免遭物理数据库损坏。备份过程在数据库打开并且用户可以使用的环境下进行。需要执行热物理备份的情形如下：

- 你的业务需求要求你的用户群体可以 1周7天、1天24小时地使用数据库。
- 研究结果表明你的数据库需要一整天的处理以达到一个高缓存命中率。在系统中使用许多 GB内存的特大型 Oracle数据库中，往往需要花费几个小时的处理时间才能使 Oracle缓存整个数据工作组。尽管系统能够在夜晚关闭，但是如果数据库不在夜晚关闭的话，其早晨的性能会更好。
- 尽管你的用户群体一天仅需要系统运行 12小时，然而剩余的 12小时几乎都花费在批处理上；没有足够的停机时间用于批处理与系统备份两方面上。

4. 逻辑备份

逻辑备份用于实现数据库对象的恢复并且它是一个全面备份策略的必要的组成部分，用来保证数据库能够从无意中的修改（例如 DELETE、DROP或UPDATE）中恢复出来。

5. 完全逻辑备份

完全逻辑备份把整个数据库导出到一个 Oracle格式文件中，该文件可以在不同的 Oracle版本、操作系统和硬件平台之间进行移植。如下特殊情况或许需要执行一个完全逻辑备份：

- 你没有另外的作为测试平台的物理系统，但是你有处于同一台机器上的成品与测试实例并且你希望它们不时地进行同步。
- 新服务器刚安装完并且将用来替换你的老服务器。你的老数据库服务器与新数据库服务器使用不同的平台（例如，Sun与HP）。

6. 指定表的逻辑备份

Oracle的逻辑备份工具可完成指定表的逻辑备份。此功能可用于如下情形：

- 你需要把表ABC从JSMITH模式移动到TGASPER模式。
- 尽管你想保存表ABC的备份拷贝，然而你将要删除它。

7. 指定用户的逻辑备份

逻辑备份可由一个或多个指定用户（或模式）组成。当出现如下情形时，你或许想要使用此功能：

- 你刚被告知可以删除用户 JSMITH。该用户拥有几个表，而你认为某些人有可能以后会需要这些表，所以你想要备份 JSMITH模式。
- 有一个应用，该应用使用的表完全包含在一个单独的模式中。你计划对此应用进行升级，必须运行一个脚本对表和索引进行“更新”以便新版本能够正常工作。

提示 这些具体的事例说明了不同的备份方法适合于在你的数据库工作中所遇到的不同情形。不要只依赖一种备份方法。尽管大多数数据库管理员知道需要物理备份的必要性，但仍有许多人不会定期运行逻辑备份。这使他们容易受到一条无意间使用的 DROP命令的损害。记住，DROP命令是立即执行的，没有回滚。在Oracle技术支持的帮助下，使用物理备份恢复一张已被删除的表是可能的。此恢复过程非常耗时间并且代价很大，该过程或许可以通过使用逻辑备份来避免。记住，系统不必为逻辑备份而关闭。

在你运行一个没有经过仔细检查的 SQL脚本以前，执行逻辑备份是非常重要的事

情。因为DROP或DELETE命令或许出人意料地隐藏起来，等待时机对你的数据库“发动攻击”。

假如你把热备份作为你的物理备份的话，那么有机会时你还要运行冷备份。由于热备份的复杂性，使得许多数据库管理员（包括我自己）错误地认为他们已经有了一个相当近期的冷备份，这样的话，他们不能有足够的备份。

24.3 使用逻辑备份

通过EXP工具，Oracle为你提供了逻辑备份（IMP用于逻辑恢复）。你要么在完全批处理模式下完全交互式地使用EXP工具，要么用一些指定的关键字交互式地使用EXP工具。在研究EXP的操作特征以前，首先查看可用的关键字。运行 `exp HELP=Y` 就会显示所有的关键字，如清单24-1中所示。

清单24-1 EXP关键字

oreo:./815/bin\$ exp help=y			
Keyword	Description (Default)	Keyword	Description (Default)
USERID	username/password	FULL	export entire file (N)
BUFFER	size of data buffer	OWNER	list of owner usernames
FILE	output file (EXPDAT.DMP)	TABLES	list of table names
COMPRESS	import into one extent (Y)	RECORDLENGTH	length of IO record
GRANTS	export grants (Y)	INCTYPE	incremental export type
INDEXES	export indexes (Y)	RECORD	track incr. export (Y)
ROWS	export data rows (Y)	PARFILE	parameter filename
CONSTRAINTS	export constraints (Y)	CONSISTENT	cross-table consistency
LOG	log file of screen output	STATISTICS	analyze objects (ESTIMATE)
DIRECT	direct path (N)	TRIGGERS	export triggers (Y)
FEEDBACK	display progress every x rows (0)		
QUERY	select clause used to export a subset of a table		

清单24-1中的关键字在下文进行解释。

USERID是EXP使用的用户标识/口令，用于登录到数据库以便执行导出操作。要使用EXP，你至少必须具有CONNECT SESSION特权。假如你打算导出另一个用户的对象，你必须具有EXP_FULL_DATABASE特权。数据库管理员用户当然可以做任何事情。

提示 你可以把EXP用于使用普通USERID/PASSWORD@REMOTE_NAME约定的远程数据库。

FULL指定是否正在导出整个数据库。它的缺省值是 N。

BUFFER用来指定在提交到导出文件以前内存中将装入多少行。缺省值根据操作系统而定。一般说来，你的缓冲区越大，导出操作运行得也就越快（同时将使用更多的系统内存）。你可以用如下公式计算将要装入到缓冲区中的行数：

行数=缓冲区容量/最大的行容量

假如该表包含一个LONG列，那么每次仅能有一行装入缓冲区。

OWNER列出了你想要导出的模式。你可以使用此关键字导出完整模式。假如 ROWS=N，那么仅仅导出用来重新生成模式的SQL代码——没有表行被导出。

FILE是导出文件的名称。你应当使用一个合理描述所包含的数据的文件名。在有些操作系统（特别是UNIX操作系统）中，你可以把数据直接导出到一个磁带设备中。当你没有足够

的磁盘空间用于导出文件时，此操作对大型数据库的备份来说特别有用。

警告 当你使用FILE关键字指定一个文件名时，根据你正在导出的内容，这些文件可能相当大。在诸如UNIX的操作系统中，你可以把数据直接导出到磁带上，这样就消除了上述的问题。假如不能这样做，你就必须把导出的内容分解为足够小的片，然后要么装入几个不同的文件系统中，要么装入一个地方，以便在随后的EXP运行中每一片能够在下一片创建以前移动到磁带中。你可能想在工作时间之外运行EXP以减轻文件系统的压力。

TABLES列出了你想要导出的单个表。假如ROWS=N，那么仅仅导出用来重新生成指定表的SQL代码，而不导出行。

当导入数据时，COMPRESS有许多含义。COMPRESS=Y将引发IMP构造一个新对象的初始区间，使之足以容纳所有对象的数据。例如，假如一张表有4个区间，每一个区间的容量是20MB，那么新表的初始区间近似为80MB。假如COMPRESS=N，新表将有4个20MB的区间，就像初始时一样。

警告 当你试图重新组织一张表（减少区间的数量以提高性能）时，尽管COMPRESS=Y非常有用，然而在使用此选项以前，你需要做一些准备工作。假设你有一张名叫ORDERS的表，该表有如下这些区间容量：30MB、100MB、1000MB、1500MB、1500MB和1000MB。此表的总容量略微超过5GB。现在假设你的机器仅有2GB和4GB的磁盘驱动器。你的系统能够有的最大的区间是4GB，然而Oracle希望找到一个5GB的区域。那么，导入进程不会工作，因为Oracle不能指定一个5GB的区域。在使用COMPRESS=Y以前，在你打算导出的每张表中使用如下的SQL语句：

```
SELECT SUM(BYTES) FROM DBA_EXTENTS WHERE SEGMENT_NAME=<table name>
AND OWNER=<table owner>
```

你可以使用此语句调节一个区间中的表。

RECORDLENGTH用于把导出文件移动到另一个操作系统上的Oracle数据库中。假如你正在从系统A移动到系统B，你必须知道系统B中的记录容量。此信息保存在由Oracle提供的操作系统文件中。

GRANTS指示EXP是否导出与被导出的对象有关的授权。

INCTYPE指定COMPLETE（完全）、CUMULATIVE（累积）或INCREMENTAL（增量）备份。你必须使用一个具有EXP_FULL_DATABASE的USEID，因为此选项仅在FULL=Y时才有用。在大多数OLTP产品环境中很少使用此能力。

注意 假如一张表已经改变了的话，CUMULATIVE或INCREMENTAL导出操作导出整张表。EXP不能只导出已变化的行。假如你选择此选项的话，确保定期地执行一个COMPLETE导出操作以保持你的当前导出基线。

INDEXES指定要导出正在被导出的表中的索引。

RECORD指定Oracle是否记录COMPLETE、CUMULATIVE或INCREMENTAL导出。这用于下面的情况中：

你通常在星期天执行一次COMPLETE导出，从星期一至星期五执行CUMULATIVE导出。星期二晚上你计划给测试系统重新装入成品系统的COMPLETE导出。当你在星期二晚上导出

成品系统时，如果 RECORD=Y，那么星期三的 CUMULATIVE 导出是基于星期二晚上的 COMPLETE 导出而不是星期天的 COMPLETE 导出。假如你正在按与通常排定的导出顺序不一致的情况下在数据库上运行导出的话，一般使用 RECORD=N。

ROWS 指示是否要导出你正在导出的表中所包含的数据。假如 ROWS=N，仅仅导出用于重新创建该表的 SQL 代码。

PARFILE 是一个参数文件的文件名，该参数文件包含用于 EXP 的任何有效的关键字设置值。下面是一个参数文件的示例：

```
TABLES=CUSTOMER,VEDOR  
OWNER=SUSAN  
COMPRESS=N  
RECORD=N
```

正如你看到的那样，参数文件没什么难理解的。它仅在每行上有一个关键字和数值。假如你有一个需要定期运行的标准导出的话，参数文件就非常有用，因为你不必担心忘记设置关键字。

CONSTRAINTS 表明是否导出与你正在导出的对象有关的约束。

假如你正在导出同时也正在被更新的表时，CONSISTENT 是有用的。CONSISTENT=Y 保证导出中的每张表在任何时刻都是一致的。假如 EXP 在导出一张表时遇到了一个已更新的行，那么它使用回滚信息并导出老版本。假如遇到插入的一个新行，EXP 将忽略它。只有在需要时才使用 CONSISTENT=Y，因为 EXP 在整个导出期间保存了一个回滚段。因为这在你的数据库上加了额外负载，所以该选项最好在业余时间使用。

假如你遇到一个快照太旧错误，表明你或许需要在系统负载较轻期间运行你的导出。你甚至需要在 RESTRICT 模式下打开数据库的情况下运行导出以使用户在导出期间不对数据库进行更新。假如你需要几个表的一致性视图，你或许也要在 RESTRICT 模式下执行导出操作。CONSISTENT=Y 仅在每张表上提供读取一致性。

LOG 是用来保存导出信息和错误信息的文件的文件名。通常，EXP 仅在你的屏幕上显示此信息。然而，因为此信息显示的速度太快以致于你来不及阅读，所以你可能想要把 EXP 日志输出保存到文件中。所有的信息和错误显示在你的屏幕上，甚至当你指定 LOG 文件名时也是这样。

提示 假如你正在使用一个 UNIX 操作系统，如果你不愿看到信息在你的监视器上飞滚，你可以把 EXP 屏幕输出重新定向到 /dev/null。假如 EXP 正在显示大量信息的话，把输出重新定向到 /dev/null 通常会使运行速度更快。这是因为对于 UNIX 操作系统来说，把信息简单地抛弃比把信息发送到屏幕上更快。

STATISTICS 指定当导出文件被导入时如何产生统计值。选项有 COMPUTE、ESTIMATE 或 NONE。假如你的导出文件较大，为了节省时间，把 STATISTICS 设置为 ESTIMATE 或 NONE（NONE 选项最节省时间）。你总是可以以后使用 ANALYZE TABLE 命令产生统计值。

DIRECT 用于增强 EXP。通过直接读取数据块（而不是通过普通的数据库引擎），DIRECT 极大地提高了导出性能。如果你有此选项的话，你通常应该使用它。你的导出操作会运行得更快。

TRIGGERS 指示触发器 PL/SQL 代码是否同表一起导出。缺省值为 Y，该缺省值适用于大多数用途。

FEEDBACK是EXP中新增加的。假如你设置 FEEDBACK=n (n是某个正整数), EXP为它导出的每n行显示一个点 (.)。设置FEEDBACK=0禁用此功能。

QUERY是Oracle 8i中新引入的。它使你能够仅导出一个基于 SELECT子句的表的子集。使用Oracle工具参考手册查阅有关这个新关键字的完整描述。

你可以在命令行上以任何顺序键入你想要的关键字。使用如下语法：

```
exp [<keyword>=<value>[,<value>,...] [<keyword>=<value>[,<value>...] ...]]
```

我一般更喜欢在批处理模式下使用 EXP, 因为我能够在不修改或轻微修改该命令的情况下随意地重复使用它。假如 EXP所需的信息比你在命令行上所提供的信息要多的话, 例如用户标识/口令, 它提示你键入该信息。要完全交互式地运行没有关键字的 EXP, 只要在系统的命令提示符下键入“exp”即可。EXP提示你它所需要的全部信息。

24.3.1 完全逻辑备份

EXP特别适用于完全逻辑数据库备份。唯一的要求是你必须有 EXP_FULL_DATABASE特权以及必须有足够的存储空间用于存储导出文件。这里有一个写入缺省输出文件中的完全逻辑备份的例子：

```
exp USERID=SYSTEM/MANAGER FULL=Y
```

下面的例子显示了一个完全逻辑备份, 该备份直接保存到 UNIX系统中的磁带上：

```
exp USERID=SYSTEM/MANAGER FULL=Y FILE=/dev/rmt0
```

假如一个对象(用户、表、索引等)被无意中删除、破坏或修改的话, 那么新近执行的完全逻辑备份会给数据库管理员一个补救的机会。尽管不能把你所要的对象带回它被删除前的状态, 然而你至少有了得到恢复的对象。

24.3.2 指定用户模式的逻辑备份

EXP很容易地支持指定用户模式的备份。导出一个指定的用户组缺省情况下会导出用户定义、所有的表、索引、约束、授权以及相关的数据。如果必须改变用户的 USERID, 或者一个用户将要被删除, 而你仅希望在手头上保存一个拷贝的话, 这就非常有用。你可以要么指定一个用户, 要么指定一组用户。这里是一个用于把 TGASPER模式导出到标准输出文件中的命令行示例：

```
exp USERID=SYSTEM/MANAGER OWNER=TGASPER
```

现在假定你正在清理老用户帐户, 打算删除 JSMITH、JKLEIN、MJONES和BGEORGE帐户。作为一名谨慎的数据库管理员, 你决不会真正删除任何东西。你需要为任何情况做准备, 例如有人下个月来找你并且说：“我现在需要 JKLEIN帐户的 XYZ表。我知道我们已要求你删除了那个模式, 但是假如我们不能得到那张表的数据, 我们会处于困境中。”下面这条命令用于将多个用户模式导出到一个 UNIX磁带设备中：

```
exp USERID=SYSTEM/MANAGER OWNER=JSMITH,JKLEIN,MJONES,BGEORGE FILE=/dev/rmt0
```

现在你已准备好了。

提示 无论何时, 只要你把将要删除的信息导出到磁带中, 在从数据库中实际删除该信息以前, 你或许希望测试导出磁带。

24.3.3 指定表的逻辑备份

导出一组表的处理方法与用户导出的方法一样。通过使用 TABLES关键字列举表的名字，你可以选择导出一个表或一组表。下面的例子把 ABC表与XYZ表从BGEORGE模式中导出到缺省输出文件中：

```
exp USERID=SYSTEM/NANAGER OWNER=BGEORGE TABLES=ABC,XYZ
```

举例来说，假如你需要在相同的或不同的 Oracle实例中的模式之间移动指定的表，对指定的表进行备份很有用。在从系统中删除一个表以前，你或许还想保存它的逻辑备份。

提示 不管算是经验还是偏执，在没有首先生成一个逻辑备份以前，我总是不愿意真正删除任何东西。你的环境有可能需要也有可能不需要此项措施，但是要注意恢复一个已删除的对象很难且非常花费时间。

注意 EXP缺乏一些主要的功能，许多熟悉其他RDBMS的数据库管理员或许对此感到惊奇。EXP使你不能导出一条基于SQL WHERE子句的信息。它仅使你能够导出全部对象、全部模式或整个数据库。Oracle的导出文件实际上是一种专有格式。你不能存取原始文本数据或用于重新生成数据库、表、授权等的 SQL代码。市场上有许多第三方的工具提供了这些缺少的功能。

24.4 使用冷物理备份

冷物理备份是数据库管理员用来保护数据库系统整体上免遭各种类型数据破坏的工具。定期的冷备份以及一组好的归档重做日志与当前重做日志使你能够把数据库恢复到任何一个时间点上——甚至是恰好在磁盘驱动器失败前的最后一次提交。

在Windows NT桌面环境出现以前，几乎完全使用诸如 tar (UNIX) 的操作系统工具执行冷物理备份。唯一的数据库交互作用是关闭数据库并在以后重新启动它。在 UNIX中仍然是这样。因为使用Windows NT的计算机不是面向命令行的，所以 Oracle创建了一个叫做备份管理器的产品，备份管理器使你能够使用一个标准的 Windows NT图形用户界面程序执行物理备份与恢复。从Oracle 8开始，Oracle引入了一个称为恢复管理器（或简称为 RWAN）的新的备份与恢复系统。该系统尽管有点复杂，但是可以满足所有的备份/恢复要求。

注意 不要把此恢复管理器与Oracle的NT恢复管理器混淆；名字非常相似，但是它们完全是不同的产品。

每种备份系统各有一组不同的特性，尽管最终执行相同的功能，但是方法不同。命令行和桌面备份方法在 24.4节和24.5节中各有一小节。因为RWAN被设计为更奇特的物理备份工具，所以它的使用在稍后的 24.6一节中讨论。

不管你的环境或方法是怎样的，冷物理备份将包含如下数据库元素：

- 控制文件。
- 数据文件。
- 重做日志文件。
- 归档日志文件。

24.4.1 命令行驱动冷物理备份

UNIX系统数据库管理员或许会有被 Oracle抛弃的感觉, 因为 Oracle缺乏用于管理诸如备份或恢复操作的图形用户界面工具。但是假如你做一些完善某些脚本的工作的话, 那么在命令行中执行的备份和恢复将比通过恢复管理器执行的备份和恢复更加灵活且不会出错。

冷物理备份一般涉及生成所有必要的数据库元素(配置文件、数据文件、控制文件、重做日志和归档日志)的一个文件级拷贝。在备份过程开始以前, 你需要确保你的数据库能够足够长时间地停留在脱机状态以便得到完全备份。假如你没有足够的时间, 应考虑使用热备份。

执行一个冷备份所需要的基本步骤如下:

- 1) 建立一个你需要备份的操作系统级文件的列表。
- 2) 选择使用 NORMAL或IMMEDIATE方式关闭数据库。
- 3) 执行步骤1中文件列表的操作系统级备份。
- 4) 正常启动数据库。

警告 在系统突然关闭后不要对它进行备份。仅在使用IMMEDIATE或NORMAL模式关闭数据库后才对它进行备份。

此外, 当你已经备份了你的归档日志文件后, 你可能会从系统中删除它们(但要确保在某个地方长期保存归档日志文件)。我通常喜欢保存系统中最近七天的归档日志。假设你每天晚上备份你的数据库, 即便你的备份已经在最近的这七天失效了, 此项操作仍然可以保证你能够恢复该数据库。设想一下在感恩节的周末, 大部分公司从星期四到星期天关门。假如在星期三你的备份脚本由于某个原因而失效了, 有可能直到下星期一早晨你才发现某些东西失效了。即使你有一组一直轮流值班的操作人员, 他们或许也不能像你一样密切地监控备份。通过保存系统中最近七天的归档日志文件, 即便数据库在星期一下午的某个时间失效了, 你仍然可以恢复它。

提示 对于具有关键使命的关键数据库来说, 至少在整个 ANALYZES或者数据库导出期间保存归档日志是很重要的。其原因是假如出现了一个坏的数据块, 你可以把数据文件恢复到一个较为合适的时刻(当你的最后一次数据库分析或导出表明它是完全有效时)并且使用归档日志把系统前滚到最近的提交。在你已经正确运行了一个针对于数据库中所有表的ANALYZE TABLE VALIDATE STRUCTURE语句后, 你可以认为你的数据库摆脱了任何坏数据块并且较老的归档日志不必从坏数据块中恢复。

1. 建立你的文件列表

不管数据库管理员尽了多大努力, 他们看起来总是不能对组成数据库的全套文件保持一个固定的处理方法。这很不幸, 因为如果冷备份不是数据库文件的完整集合的话, 那么它几乎毫无价值。在你关闭数据库以前, 或许很值得对每样东西多次检查一下以确保你有需要备份的文件的完整列表。

当数据库启动时, 你要么明确地要么隐含地读取一个 init<instance>.ora文件。你必须备份此文件。你还要查看一下此文件是否有一个 ifile行, 在读取init.ora文件时, 该行基本上通知 Oracle包含另一个文件。假如有一个使用 ifile关键字列出的文件名, 你必须备份那个文件。

要生成由数据库使用的控制文件的列表, 你可以查看可用的 init.ora文件或者运行下面的

SQL语句：

```
SELECT VALUE FROM v$PARAMETER WHERE NAME='control_files';
```

Oracle返回控制文件的一个逗号分隔的列表。

可以使用如下SQL语句获得所有数据库数据文件的列表：

```
SELECT FILENAME FROM DBA_DATA_FILES;
```

Oracle返回一个列举了用于数据库中的表空间的所有操作系统级文件名的查询结果。

假如数据库被正常关闭的话，重做日志不包含 Oracle所需的信息。从技术上讲，它们不需要被备份——但是你仍然必须在恢复期间创建新的重做日志，所以无论如何你或许要备份它们。如下的SQL语句生成了重做日志文件的清单：

```
SELECT MEMBER FROM V$LOGFILE;
```

归档日志一般可由归档进程自动写到一个目录中。init.ora文件中的log_archive_dest关键字被设置为目的目录，归档日志将被写入这里。你也可以使用如下SQL语句获得相同的信息：

```
SELECT VALUE FROM V$PARAMETER WHERE NAME='log_archive_dest';
```

提示 在一个大型数据库中，追踪单个的文件非常消耗时间并且容易出错。你对上述两个问题的最好解决方法是采用一个好的文件组织结构。可能的话，使用 Oracle的OFA（光学灵活结构）准则以便容易地维护文件结构。

使用OFA结构，你能够快速地区别需要得到备份的数据库的所有关键组件。OFA还能够帮助数据库管理员确信控制文件、重做日志和其他冗余的数据库组件被映射到不止一个物理设备中。

2. 关闭Oracle

这里你不需要做任何特别的事情，仅仅正常关闭数据库即可。不要使用 SHUTDOWN ABORT，可以使用SHUTDOWN NORMAL，或者SHUTDOWN IMMEDIATE关闭数据库。

3. 执行备份

你准备使用手头现有的或从步骤1中得到的文件列表对数据库进行备份。使用你的操作系统级选择工具，你需要以任何顺序把文件备份到一个备份设备中（通常是磁带）。

对于UNIX操作系统而言，选择工具趋向于使用 tar、cpio或vdump，而tar是最通用的。按照正常步骤启动Oracle数据库。

提示 尽管冷备份消耗时间，然而你可以使用几个选项来减少数据库必须关闭的时间。

假如你有大量的磁盘空间，可以仅在磁盘中的一个临时区中执行配置文件、数据文件、控制文件、重做日志以及归档日志的文件拷贝。然后你可以再次启动你的数据库并开始把临时区备份到磁带上。你至少需要两倍数量的磁盘空间用于该数据库，它会大量减少备份所需的时间。

考虑把多个磁带设备增加到你的系统中。在多操作系统下，你可以同时运行备份程序的多个实例。这使你能够同时备份数据库的几个不同区域。实际上，四个磁带驱动器备份一个数据库所花费的时间通常是一个磁带驱动器备份该数据库所花费时间的三分之一。

24.4.2 桌面驱动冷备份

仅当图形用户界面已经简化了字处理方法、电子表格、电子邮件、甚至是系统管理时，

它才把许多有用的功能提供给数据库管理员。Oracle的备份管理器服务器作为一个工具用于Oracle数据库的备份。原来由脚本和大量基于操作系统的工具完成的工作现在可以由 Oracle提供并支持的一个程序来完成。图 24-1显示了一个通用但很简单的备份管理器界面。

备份管理器的界面相当容易使用。请注意你的备份管理器的窗口或许与你在图 24-1中所看到的有点不同。该窗口或许仅为你提供了可用于数据库状态的选项（正在运行 /没有运行，归档/非归档日志模式）。

你需要在此工具外完成的唯一工作是确定将你的脱机备份存储在哪里。备份管理器使你既可以将脱机备份存储到磁盘上的一个目录中也可以存储到一个磁带驱动器中。注意在图24-1的左下角，备份管理器确切地告诉你备份该数据库所需花费的空间。为了使备份操作正常工作，你需要一个目录或足够容量的磁带设备。

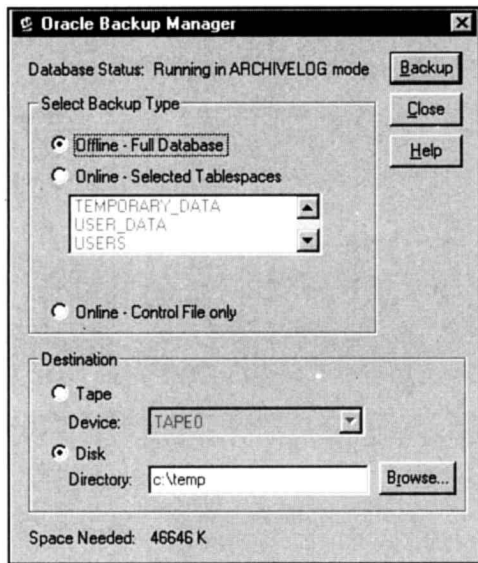


图24-1 用于Windows NT的Oracle备份管理器有一个简单的图形用户界面

执行一个冷物理备份包含如下步骤：

- 1) 确保数据库已经启动并且运行正常（参见随后的警告）。
- 2) 打开备份管理器。
- 3) 从Select Backup Type框中选中Offline-Full Database选项。
- 4) 通过在Destination框中选中Tape或Disk把数据备份到磁盘或磁带中。假如你把数据库备份到磁带中，你还需要指定使用哪一个磁带设备。另外，你需要指定你想把数据备份到哪一个目录中。
- 5) 单击Backup按钮。备份管理器完成剩余的工作。
- 6) 重新启动数据库。

警告 不要把数据备份到你的数据库所驻留的物理设备中。

警告 因为通常在数据库脱机时执行冷备份，所以在步骤1中确保数据库正在运行看起来有点奇怪。这是因为备份管理器需要在由于备份而关闭数据库以前对该数据库进行访问以便它能建立一个需要备份的数据库文件列表。在没有数据库运行的情况下启动备份管理器可能产生一个不完全备份集。

提示 假如存在打开的会话（包括服务器管理器），备份管理器不会执行脱机备份。假如你很难防止会话在Oracle中打开，你可以在启动备份管理器以前在RESTRICT模式下打开数据库。不要忘记关闭你的服务器管理器会话！

备份管理器不备份init.ora、config.ora或归档日志文件。尽管这些文件一般会在普通的操作系统备份过程中得到备份，然而你应与系统管理员一起仔细检查以确保这些非常重要的文件定期得到备份。

24.5 使用热物理备份

数据库管理员受委托维护的数据库经常是公司的命根子。这些数据库有时一周 7天、一天 24小时地用于访问。在这种情况下，只有具有工业强度的 RDBMS才有充分的能力。Oracle就是这样一种数据库，这是已经经过实践证明的。

热备份要求你对 Oracle和支持 Oracle的操作系统具有较高的专业知识以及适应水平。由于热备份的复杂性，你还必须投入更多的时间和精力用来测试你的备份策略。只有当你从一个测试系统中的大量失败中恢复出来时，热备份系统才被认为是可靠的。在开始实施一个热备份方案以前，确信值得为增加可用的时间而增加成本与令人头疼的事。

在继续热备份以前，你应当保证熟悉冷备份。你要依赖在以前的小节中所讨论过的冷备份。特别是，本节利用了你的冷备份知识并且讨论数据库在启动和运行时对它进行备份的附加的复杂性。Oracle提供了两种热备份方法，一种用于命令行 UNIX环境，另一种用于桌面 (Windows) 环境。由于此原因，下面将分别讨论每种热备份方法。

24.5.1 理解复杂性

系统管理员与数据库管理员经常提出的一个问题是“热备份为何如此重要？我们备份有用户的操作系统不会有任何问题，所以我們也可以在系统中有用户时备份 Oracle数据文件。对吗？”回答是，错。要阐述热备份的复杂性需要简要地讨论一下 Oracle如何存储与修改信息。

回忆一下，Oracle的基本存储单元是数据块（可以在创建数据库时配置它的容量）。数据块中存储数据，数据块头存储包含在块中的数据的信息。

进行正常的冷备份时，每一个数据文件被逐位地拷贝到备份设备中。假如数据库处于联机状态时，麻烦的情况是备份程序和 Oracle都工作于同一个块上。假定当 Oracle写新的数据和

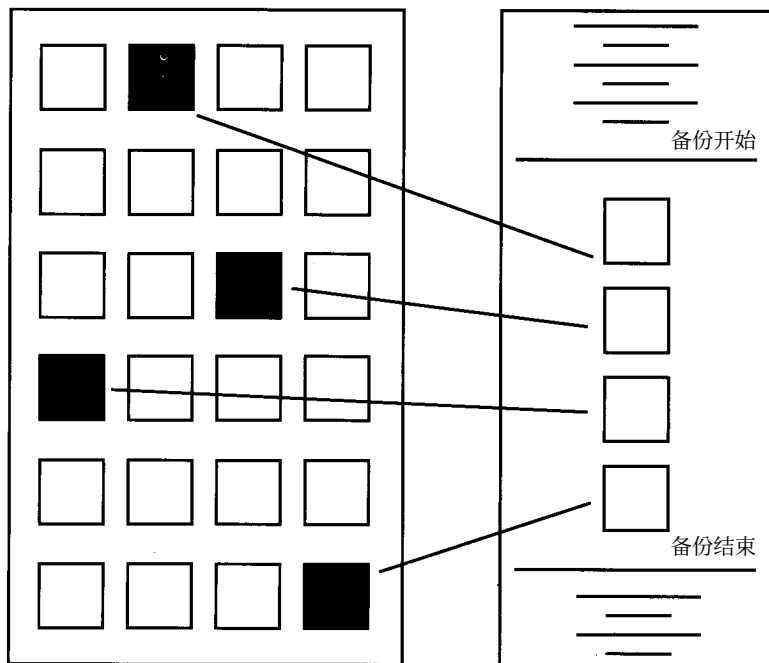


图24-2 Oracle的备份模式在恢复期间对可能受到破坏的数据块进行修补

更新数据块头时，备份程序仅完成了对指定块的块头部分的备份。在备份程序对指定的数据块完成备份之后，该块具有老的块头和新的数据。这个不纯的数据块使 Oracle 不知道如何对它进行处理。

Oracle 的解决方案是实现一种称为备份模式的特别模式，该模式在各个表空间上依次被启用或禁用。当启用该模式并且 Oracle 更新一个数据块时，已修改块的完整拷贝被写入到重做日志中。这与 Oracle 的普通过程仅把修改写入重做日志形成对比。

Oracle 数据文件的联机备份仍然存在不纯的数据块，但有效的拷贝存在于重做日志中（最终会在归档日志中）。在恢复状态下，数据文件得到恢复并且 Oracle 使用重做/归档日志来用有效块“修补”数据文件。这样，最终的结果是一个有效且一致的数据库。在图 24-2 中对此系统进行了举例说明。

24.5.2 命令行驱动热物理备份

通过把 SQL 与操作系统脚本结合使用实现基于命令行的 UNIX 环境下的热备份。热备份的具体细节在不同的操作系统下会有所变化，下面是热备份的操作顺序：

- 1) 建立一张需要备份的表空间的列表。
- 2) 对于该列表中的每一个表空间，执行如下操作：
 - a. 建立一个数据文件列表。
 - b. 让表空间处于备份模式下。
 - c. 把数据文件列表中的数据文件备份到一个备份设备中。
 - d. 使该表空间脱离备份模式。
- 3) 使用 Oracle 生成一个“可备份的”控制文件。
- 4) 把步骤 3 中的控制文件备份到一个备份设备中。
- 5) 强制执行一个日志交换。
- 6) 等待重做日志得到归档。
- 7) 把归档日志目录备份到一个备份设备中。

1. 建立一个表空间列表

如下的示例 SQL 语句在一个 Oracle 数据库中产生表空间列表：

```
SELECT TABLESPACE_NAME FROM DBA_TABLESPACES;
```

2. 为表空间建立一个数据文件列表

你需要得到表空间的数据文件列表以便确切地知道哪一个文件要备份：

```
SELECT FILE_NAME FROM DBA_DATA_FILES WHERE TABLESPACE_NAME='<tablespace name>';
```

3. 让表空间处于备份模式下

在备份表空间的数据文件以前，需要使用如下的 SQL 语句让表空间处于备份模式下：

```
ALTER TABLESPACE <tablespace name> BEGIN BACKUP;
```

4. 把数据文件备份到备份设备中

这是一个较大的操作系统问题。你知道使用 SQL*Plus 为每个表空间生成一张需要备份的文件列表。本书附带的 CD-ROM 包含一个用于 UNIX 的工作脚本，它举例说明了此过程。

5. 从备份模式中取出表空间

就像你使用 SQL 语句让表空间处于备份模式中一样，应使用一条类似的语句使它脱离备

份模式：

```
ALTER TABLESPACE <tablespace name> END BACKUP;
```

6. 生成一个“可备份的控制文件”

因为有一个Oracle实例正在运行，这必然意味着控制文件是打开的并且可能在任何时间更新，所以你需要Oracle产生一个保证是一致的且不会被更新的控制文件拷贝。如下SQL语句将完成此任务：

```
ALTER DATABASE BACKUP CONTROL FILE TO <output filename> REUSE;
```

其中，<output filename>是任何有效的文件名，它指定了你希望写入控制文件拷贝的地方。REUSE意指如果该文件已经存在的话，它将被改写而不是放弃。

7. 强迫执行一个日志交换

当一个表空间上的备份开始和结束时，Oracle在重做日志中做标记。因为你没有备份重做日志，所以你需要归档当前的重做日志（该重做日志内含你刚刚备份的表空间的“备份结束”标记符）。当你备份归档日志目录时，你还要为每个表空间备份带有“开始备份”和“结束备份”的日志。日志交换用如下语句实现：

```
ALTER SYSTEM SWITCH LOGFILE;
```

8. 等待对交换重做日志进行归档

通常当一个日志交换发生时，Oracle的归档进程立刻把当前重做日志拷贝到归档日志目录。这个过程要花费时间并且Oracle并不通知你重做日志何时成功归档。一般来说，在备份脚本中应安排一个两到五分钟的固定延迟时间，在大多数情况下此时间足够了。然而，假如你的重做日志特别大或者日志归档到一个速度较慢的介质中，你应当确保Oracle能够得到足够的延迟时间用于归档一个重做日志。

9. 把归档日志目录备份到一个备份设备中

就像你在冷备份中所做的那样，你需要把包含归档日志的目录备份到一个备份设备中。

提示 因为Oracle对于每个写操作把一个完整的数据块写入到重做日志中，而不是写入改变了的信息中，因此当表空间处于备份模式时，重做日志填充得更快。你应当让你的数据库中的活动最少时进行你的联机备份。尤其要注意已调度的批处理作业。它们趋向于进行非常集中的写操作并且容易被忽略，因为它们经常自动地由调度程序运行。

24.5.3 桌面驱动热物理备份

Windows NT环境下的Oracle数据库管理员会发现热备份比命令行备份更容易。你将使用Oracle的备份管理器执行实际的备份操作。当你在数据库正在运行时启动备份管理器时，一个类似于图24-3的窗口会显示出来。

你需要决定把你的数据库备份存放在哪里。你要么把它备份到磁盘上的一个目录中，要么备份到一个磁带设备中。备份管理器能够指出用你选择的备份方式备份数据库时所需要的空间容量。此信息有助于你决定把数据库备份到哪里。采用如下这些步骤以执行数据库的联机备份：

- 1) 确保你的Oracle数据库运行正常。
- 2) 启动备份管理器。

3) 在 Select Backup Type 框中选中 Online-Selected Tablespaces 选项。

4) 选择你希望备份的表空间。对于一个完全备份来说,你必须选择所有的表空间。在你完成此项操作后,对已选择的表空间进行备份所需要的空间数量显示在窗口的左下角。

5) 在 Destination 框中,选择备份到磁带还是磁盘中。假如你选择备份到磁盘中,还应提供了用于存储数据库备份的目录名称。

6) 单击 Backup 按钮。

7) 当备份结束后,在 Select Backup Type 框中选中 Online-Control File only 选项。

8) 在 Destination 框中,选中 Disk 选项并提供你希望存储控制文件备份的目录名称。尽管你可以备份到磁带中,然而,通常不值得把小容量的控制文件存储到磁带中。

在此过程结束后,你的数据库已经成功地得到备份。备份管理器已经完成了对你的数据库的备份并且已经使所有的表空间脱离备份模式。

警告 备份管理器不能备份用于启动数据库的 init.ora、config.ora 或归档日志文件。尽管这些文件一般在正常的操作系统备份过程中得到备份,然而你应该和系统管理员仔细检查一下以确保这些非常重要的文件定期得到备份。

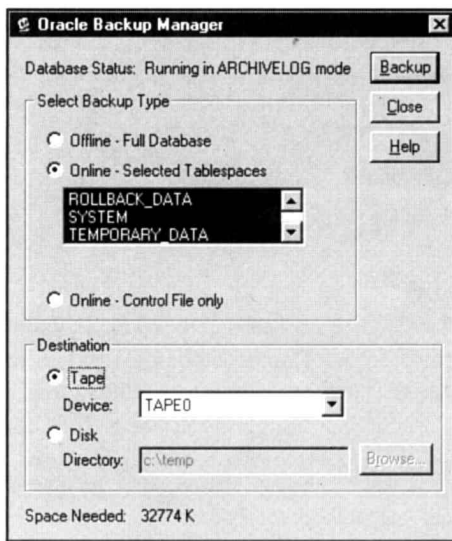


图24-3 备份管理器能够用来执行热备份

24.6 使用恢复管理器进行物理备份

Oracle8中引入的恢复管理器(RMAN)很有可能开创了一个全新的方法用于执行 Oracle 数据库的备份。它提供了从命令行与 GUI用户界面执行的所有热备份和冷备份功能。恢复管理器同磁盘和磁带备份介质一同工作并提供了如下优于传统的基于操作系统的物理备份方法的特点:

- 热备份不会导致与传统的 BEGIN/END 备份方法有关的重做日志产生率。
- 作为备份过程的一个集成部分,检查数据库块是否残缺不全。这可以减少对整个数据库频繁使用 ANALYZE TABLE VALIDATE STRUCTURE 语句的要求。
- 支持增量物理备份。
- 支持多线程备份(仅限企业版)。
- 恢复管理器提供一个集成的分类系统,该分类系统在鉴别所需的备份磁带时能够把混乱减到最小。

恢复管理器明显提供了许多优于传统的备份方法的特点。然而,许多用户由于下述原因而不使用恢复管理器:

- 尽管 Oracle 已经花费很大力气使恢复管理器尽可能地简明易懂,然而在处理简单的备份要求时,恢复管理器必须提供的灵活度仍然使它有点不方便。
- 作为一种较新的产品,它有丢失重要数据的潜在可能性。在完全依赖它以前,有些单

位可能只想仅在更有限和不太关键的系统中使用它。

- 没有经过广泛的测试，管理员不愿改变备份方法；那些拥有经过严格编写的备份脚本的单位或许不愿接受一个新的、相对比较年轻的备份系统；等到该系统的优点超过了初始实现的成本时，这种情况也许会有变化。

对恢复管理器的使用介绍几乎可以占用一本书。在本节中，将向你介绍一个基本的完全系统备份过程。要想得到进一步的信息，参阅 Oracle 8 备份与恢复向导中的“ Recovery Manager Concepts ”一章。所示的例子将使用恢复管理器的命令行版本。图形用户界面版本可通过企业管理器的备份管理器软件得到。

在使用恢复管理器以前，必须在将要得到备份的数据库以外的数据库中建立一个恢复目录 (Catalog)。在不同的数据库 (有可能是一个不同的节点) 中建立恢复目录的原因是，确保当编目的数据库不存在 (这种情况在恢复介质失效时发生) 时目录是可用的。假如你不止有一个数据库节点 (例如 Mars 和 Venus)，你可能使用 Mars 的数据库保存 Venus 的目录，或者使用 Venus 的数据库保存 Mars 的目录。

警告 尽管恢复管理器不要求使用目录，然而建议使用目录。

恢复管理器一般需要发出两个命令行参数 (参见本节中说明参数使用的恢复管理器会话示例)；

- TARGET 指定一个用户标识 / 口令 @ 数据库名，它在将要使用恢复管理器的数据库上具有管理权限。
- RCVCAT 指定一个用于存储目标数据库的恢复目录的用户标识 / 口令 @ 数据库名。这应当是在一个与目标数据库不同的数据库上 (可能也不是同一主机)。

第一次使用恢复管理器需要采用如下步骤：

- 1) 选择一个存放恢复目录的数据库。
- 2) 在步骤1中选取的数据库上用你选好的口令创建一个用户恢复管理器。把连接和资源授予恢复管理器。确保选择合适的缺省表空间与临时表空间。
- 3) 作为恢复管理器注册到目录数据库并运行和 CATALOG.SQL 与 CATPROC.SQL 处于同一目录中的 CATRMAN.SQL 脚本。
- 4) 你必须像如下例子所显示的那样对要在目录中备份的数据库进行注册。

```
[tgasper@oreo tgasper]$ rman target=sys/manager rcvcat=rman/rman@remote_db
```

```
Recovery Manager: Release 8.0.5.0.0 - Production
```

```
RMAN-06005: connected to target database: TST8
```

```
RMAN-06008: connected to recovery catalog database
```

```
RMAN> register database;
```

```
RMAN-03022: compiling command: register
```

```
RMAN-03023: executing command: register
```

```
RMAN-08006: database registered in recovery catalog
```

```
RMAN-03023: executing command: full resync
```

```
RMAN-08029: snapshot controlfile name set to default value: ?/dbs/snapcf_@.f
```

```
RMAN-08002: starting full resync of recovery catalog
```

```
RMAN-08004: full resync complete
```

```
RMAN> exit;
```

你现在准备使用恢复管理器备份你的数据库。下面的示例表明如果数据库没有打开 (然而，该数据库必须被安装) 的话，恢复管理器将执行一个冷数据库备份；如果数据库打开的

话，恢复管理器将执行一个热备份。热备份和传统的备份一样，仅能在运行于 ARCHIVELOG 模式下的数据库系统中执行。

```

RMAN> run {
2> allocate channel c1 type disk;
3> backup full filesper set 3
4> (database format 'rm_%p%d.%s');
5> }

RMAN-03022: compiling command: allocate
RMAN-03023: executing command: allocate
RMAN-08030: allocated RMAN-08500: channel c1: sid=13 devtype=DISK
RMAN-03022: compiling command: backup
RMAN-03023: executing command: backup
RMAN-08008: channel c1: starting datafile backupset
RMAN-08502: set_count=3 set_stamp=362111548
RMAN-08010: channel c1: including datafile 1 in backupset
RMAN-08011: channel c1: including current controlfile in backupset
RMAN-08010: channel c1: including datafile 2 in backupset
RMAN-08013: channel c1: piece 1 created
RMAN-08503: piece handle=rm_1TST8.3 comment=NONE
RMAN-08008: channel c1: starting datafile backupset
RMAN-08502: set_count=4 set_stamp=362111606
RMAN-08010: channel c1: including datafile 4 in backupset
RMAN-08010: channel c1: including datafile 3 in backupset
RMAN-08013: channel c1: piece 1 created
RMAN-08503: piece handle=rm_1TST8.4 comment=NONE
RMAN-03023: executing command: partial resync
RMAN-08003: starting partial resync of recovery catalog
RMAN-08005: partial resync complete
RMAN-08031: released channel: c1

RMAN> exit;

```

24.7 从逻辑备份中恢复

你可以使用由 Oracle 提供的 EXP 工具创建逻辑备份。因为导出存在于一个专有格式中，所以仅有一个免费赠送的工具 IMP。IMP 不会从别的系统中导入数据。适合于此的工具是 SQL*Loader。

IMP 与 EXP 的语法相同并且许多选项是完全相同的。清单 24-2 显示了 IMP HELP=Y 的输出结果。

清单 24-2 IMP HELP=Y 的输出结果

Keyword	Description (Default)	Keyword	Description (Default)
USERID	username/password	FULL	import entire file (N)
BUFFER	size of data buffer	FROMUSER	list of owner usernames
FILE	output file (EXPDAT.DMP)	TOUSER	list of usernames
SHOW	just list file contents (N)	TABLES	list of table names
IGNORE	ignore create errors (N)	RECORDLENGTH	length of IO record
GRANTS	import grants (Y)	INCTYPE	incremental import type
INDEXES	import indexes (Y)	COMMIT	commit array insert (N)
ROWS	import data rows (Y)	PARFILE	parameter filename
LOG	log file of screen output		
DESTROY	overwrite tablespace data file (N)		
INDEXFILE	write table/index info to specified file		
FEEDBACK	display progress every x rows(0)		

用于IMP程序的常用关键字在下面描述。

USERID是用户标识/口令，IMP用它来登录到数据库以执行导入操作。你至少必须有CONNECT SESSION特权才能使用IMP。假如你的导出文件是一个完全导出，你必须具有IMP_FULL_DATABASE特权才能导入它。当然，数据库管理员用户可以做任何事情。

提示 你可以把IMP用于使用普通USERID/PASSWORD@REMOTE_NAME约定的远程数据库。

FULL指定是否导入整个数据库，缺省值是N。

提示 当使用一个用FULL=Y导出的导出文件时，你总能选择FULL=N。

BUFFER用来确定在写入数据库以前内存中将要装入多少行。缺省值视操作系统而定。一般说来，缓冲区越大，导入运行的速度就越快（同时使用更多的系统内存）。你可以用如下公式计算装入缓冲区的行数：

行数=缓冲区容量/最大行容量

假如表包含一个LONG列，那么每次仅能有一行装入缓冲区。

FROMUSER定义了导出文件中资源模式的名称。除非你也使用 TOUSER关键字，否则对象将被导入数据库中相同命名的模式中。假如一个在 FROMUSER中命名的模式不存在（并且没有指定TOUSER），那么对象将被导入运行IMP的USERID模式中。

FILE是导出文件的名称（或者是磁带设备的名称）。

TOUSER是数据库中一个已经存在的用户模式的名称，其中数据库对象可以装入该用户模式中。此关键字支持 FROMUSER关键字。一般说来，TOUSER用于把对象导入到与被导出模式不同名称的模式中。

当把SHOW设置为Y时，显示导出文件的内容；没有东西被实际提交给数据库。

TABLES是将被导入的表的列表。假如省略 TABLES的话，所有的表将被导入。

当把IGNORE设置为Y时，如果不能创建一个对象，仍然能够使导出继续进行。假如你正在把行导入一个已经存在的表中时，此关键字非常有用。

提示 假如你打算导入到一个已经存在的含有约束条件的表中，因为数据可能以任何顺序装入表中，所以你应该暂时停用这些约束条件。当备份结束后，你可以重新启用这些约束条件。

RECORDLENGTH用于把导出文件移动到位于另一个操作系统上的 Oracle数据库中。基本上，假如你正在把文件从系统 A移动到系统B，那么你必须知道系统B上的记录容量。有关于此的信息可以在Oracle提供的操作系统文件中找到。

GRANTS确定权限设置值是否随对象导入。

INCTYPE可以把RESTORE或SYSTEM作为输入。SYSTEM从最后一个增量导出文件中导入该系统的对象（也就是说，仅导入 SYSTEM模式对象）。RESTORE导入自上次导出运行以来发生改变的用户对象。

INDEXES指定用于被导入的表的索引是否也将被导入。

COMMIT指定当导入每个表时IMP是否将生成多个提交。假如 COMMIT=N，那么在装入每张表后仅发生一次提交。COMMIT=Y减少了回滚溢出的可能性。COMMIT=N的优点是如果某个表上的导入失败了，那么将发生一个回滚，它将完全腾空该表。这是很值得做的，因

为它把表返回到导入以前的状态，因此很容易在没有使记录重复的风险下重新开始导入。

ROWS确定是否还装入被导入表的行。假如你想要做的就是表中没有数据的情况下移动一个模式定义的话，那么设置 ROWS=N非常有用。

就像PARFILE用于EXP一样，它同样用于IMP。

LOG用于记录所有正常情况下发送到终端的输出的文件。提供一个你希望在其中记录输出的标准文件名。使用此选项不会阻止输出在屏幕上显示。

当一个完全导出正在被导入时，DESTROY是有效的。当创建一个新的表空间时，DESTROY=Y可以使任何已存在的数据文件被改写。这可用于将一个实例整个替换的情况下。然而，假如你正在相同的机器上创建一个数据库的完全相同的拷贝，应使用 DESTROY=N。因为你不想破坏你的原始数据库！在这种情况下，你应手工创建所有必要的表空间，然后执行导入操作。

INDEXFILE是一个文件名，在该文件中写入用于在导出文件中创建索引的 SQL代码。当使用INDEXFILE时，不执行对数据库的修改。假如你想改变你的新系统中的初始和下一个区间值，INDEXFILE就会发挥作用。假如你要人工创建此输出的索引，应确保在 INDEXES=N时导入以防止它们被自动创建。

就像EXP一样，你可以为IMP提供你所希望的关键字。假如IMP需要的信息多于你所提供的，它会提示你提供信息。要完全交互式地运行，只要在命令行中键入 imp即可。

你可以执行一个完全导入或部分导入。完全导入在一个目标实例中重新创建一个数据库。显而易见，你的导出文件必须包含你需要导入的所有信息。你总可以导入一个导出文件中的数据子集。在随后的小节中将对这两种不同的方式分别进行讨论。

24.7.1 从逻辑备份完全恢复

完全逻辑恢复一般既可以在同一台机器上，也可以在不同的机器上重新创建一个已存在系统的拷贝。因为导出文件不受平台约束，所以它们是可用于此操作的工具。导出文件还不受版本约束。实际上，当你正在更新 Oracle版本时，逻辑备份/恢复经常被推荐使用。

要想完成一次成功的恢复，唯一的要求是存在一个数据库实例。所有的表空间、用户、环境资源文件、表、索引等将用IMP重新创建。

提示 使用IMP创建表空间可以防止一个严重的问题。你可以用相同名称的数据文件（包括它们的路径）重新创建表空间。你可能希望在不同的数据库中有相同的表空间，但是你可能还需要不同的数据文件文件名，把它们存放在不同的目录中，或者使它们具有不同的容量，或一个不同的平台或许根本不能使用相同的文件格式。在这些情况下，你想要在目标系统中人工创建表空间。下面这条 SQL语句帮助你了解每个表空间的名字以及它们的容量（以字节为单位）：

```
SELECT TABLESPACE_NAME,SUM(BYTES) FROM DBA_DATA_FILES GROUP BY  
TABLESPACE_NAME;
```

如下的SQL语句将帮助你确定表空间的容量。它指出了每个表空间中的空间数量（字节），这些表空间还没有被分配给一个段（表、索引等）：

```
SELECT TABLESPACE_NAME,SUM(BYTES) FROM DBA_FREE_SPACE GROUP BY TABLESPACE_NAME;
```

警告 假如你正在同一台机器上把一个实例拷贝到另一个实例，那么你必须在目标实例

中人工创建表空间。假如你不这样做，当IMP试图在目标实例中创建表空间时，它可能会破坏你的原始数据库实例中的数据文件。

一个典型的完全逻辑恢复语法相当简单。下面便是一个例子：

```
imp USERID=SYSTEM/MANAGER FULL=Y IGNORE=Y FILE=<export filename>
```

这里，假设一个实例已经运行并且存在适当的表空间。

24.7.2 使用逻辑备份进行部分恢复

在大多数情况下，你将使用IMP工具把某个对象（或用户模式）集合恢复到你的数据库中。在试图使用IMP恢复部分数据库以前，你应该有合适的表空间、回滚段和用户设置。记住，假如一个表空间不存在或不能容纳被导入的对象的话，那么该对象将被装入用户的缺省表空间中。还有，当导入时，如果当前数据库中没有FROMUSER列举的用户，那么在USERID中指定的用户将用来存储对象。

1. 示例1

回忆一下数据库管理员要使用逻辑备份的情况。在每一个简短的重新叙述后，恢复对象 / 模式需要的IMP命令行将被显示出来。

假定你需要把用户SKLEIN重新命名为SJONES。你已经把原始用户模式导出到一个文件中。新的用户标识SJONES已经被创建，你准备把对象从老模式恢复到新模式中：

```
imp USERID=SYSTEM/MANAGER FILE=SKLEIN2SJONES.DMP FROMUSER=SKLEIN TOUSER=SJONES
```

2. 示例2

上周你被要求从数据库中删除用户TGASPER。在运行了一个FULL=Y导出数据库后（当然在几小时后），你删除了用户TGASPER。然而现在需要查看TGASPER模式中的表。你刚刚重新创建了TGASPER模式并且准备导入该对象：

```
imp USERID=SYSTEM/MANAGER FILE=/dev/rmt0 FROMUSER=TGASPER
```

3. 示例3

一个常见的应用把APAS模式中的表用于它的配置信息。就在上个月，版本2.3被升级到版本3.0。在那次升级中，不再需要表STACONF和USRCONF。作为精明的数据库管理员，你在删除那些表以前导出了它们。版本3.01刚刚推出，你推测它们需要USRCONF表的支持：

```
imp USERID=SYSTEM/MANAGER FILE=table_STACONF.dmp TABLE
```

毫无疑问，你必须处理更复杂的导入 / 导出问题。如果可能的话，在把你的导入用于成品系统以前，总是先在一个测试样本中测试它们——尤其是当你使用IGNORE=Y时。你可以用一个不正确的导入对一个已无法修复的数据库表进行任意地破坏。记住你为Oracle支持程序付了费用。假如你有任何疑问的话，务必使用它解决导入 / 恢复文件问题。

24.8 使用物理恢复

许多数据库管理员认为物理恢复是他们工作中最感到压力和困难的部分。像物理备份一样，大部分的恢复传统上是以前操作系统为中心的。Windows NT恢复管理器与恢复管理器(RMAN)都能够替代恢复进程的大量操作系统部分。不管你使用哪一个平台，大部分相同的问题和序列仍然适用。物理恢复包含的内容远超过本书所涉及到的。通常以下四种情况需要执行物理恢复：

- 你需要在另一个系统上生成当前数据库的一个拷贝（使用相同的操作系统和 Oracle版本）。这也可以通过使用FULL=Y关键字的逻辑备份/恢复来完成。
- 假如磁盘升级或文件系统分区发生变化，你需要恢复数据库或该数据库的任何一部分。
- 一个或多个数据文件已经遭到破坏或丢失。
- 你需要把数据库恢复到以前的状态。

前两种情况实际上是同一个过程：你需要从冷备份中恢复丢失的文件。实际上，你正在物理地重新创建该数据库。此论题将在本章的 24.8.1节中讨论。

当磁盘驱动器失效、数据文件被偶然删除或由于任何原因而使你不能存取数据时，你需要面对恢复丢失的数据文件的问题。因为你不想丢失任何数据，所以你使用完全恢复（complete recovery）。

除了必须采用“前进一步，后退两步”的方法外，把数据恢复到过去的某个时刻也是一种完全恢复形式。这被称为不完全恢复（incomplete recovery）。

提示 假如你正在你的成品系统上工作并且需要执行某种类型的恢复，可能的话，在你考虑恢复数据库以前，你总是应当关闭数据库并且执行一个冷备份。这样，万一某一事物出错，你可以补救。此外，好的经验法则是设法完成一个恢复。假如失败的话，打电话给Oracle公司。数据库管理员通过试用恢复技术很快会掌握它。

24.8.1 物理地重建一个数据库

数据库管理员经常需要在另一台机器上创建数据库的一个副本。作为一般维护的一部分，通常要升级数据库磁盘驱动器或重新建立包含数据库文件的文件系统。在任何一种情况下，你都需要对系统“照相”（换句话说，执行一个冷备份）。当系统管理员完成他们的维护工作后，你可以把所有的文件拷贝到另一台机器上或把他们恢复到同一台机器上。

提示 假如物理恢复正用来创建已存在的系统的另一个拷贝，记住即使SID改变了，数据库名称也不会改变。参考Oracle注释15390.1或Oracle的技术支持以帮助修改一个已存在数据库的数据库名称。

你需要做的全部事情是把冷备份恢复到原来的机器（当维护结束后）或一个新机器中，然后启动数据库。记住，你的所有数据、重做日志、归档日志和控制文件应当刚好放入与它们原来的存放目录相同的目录位置中。新环境下的 Oracle不知道自它被关闭后所发生的任何事情。因此，Oracle希望找到以前的所有组件。

提示 假如你需要改变一个新文件的存放位置，使用24.8.3节的“重新命名并移动数据库文件”部分中所描述的ALTER DATABASE RENAME FILE语句。假如你需要改变大量数据库文件的存放位置，通常最好的办法是创建一个完成此项工作的脚本或使用一个逻辑备份与恢复。在服务器管理器中一次改变许多数据库文件的存放位置非常单调乏味并且容易出错。

在一个命令行系统中，恢复完全是基于操作系统的。你需要恢复如下的数据库文件：

- init<instance>.ora。
- config<instance>.ora。
- 重做日志。

- 控制文件。
- 数据文件。
- 归档重做日志。

你将使用你的机器中可用的任何备份/恢复工具恢复这些文件。

假如你在维护期间仅仅丢失了少数文件，只要你没有在冷备份后重新启动数据库，你就可以恢复丢失的文件。在你安置好所有的文件后，通过服务器管理器启动数据库。

在Windows NT环境下，Oracle使用NT恢复管理器进行数据文件的恢复。在运行 NT恢复管理器以前，确保存在合适的 init<instance>.ora文件。记住把它放置在与原来相同的驱动器和目录路径中。

当你启动NT恢复管理器时，你会看到一个类似于图24-4的窗口。

要恢复数据库，完成如下的步骤：

- 1) 选择Restore from full database backup选项。
- 2) 单击Recover按钮。
- 3) NT恢复管理器现在需要知道到哪里查找备份文件（该文件由Oracle备份管理器产生）。

键入存放备份文件的目录名称，或者选择已经在其中安装了备份磁带的磁带设备。

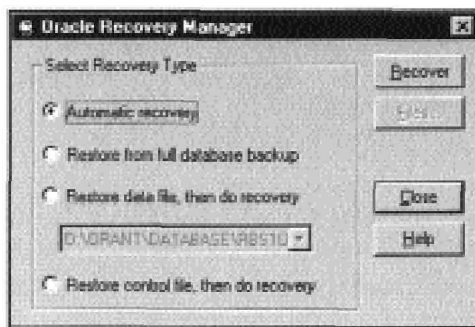


图24-4 Oracle的NT恢复管理器提供了一个方便的图形用户界面用于恢复数据库操作

4) 假如你正在从磁盘中恢复，确保在Backup域中指示了最新的备份。

5) 单击OK。开始进行恢复。

6) 启动数据库。

提示 假如你正在Windows NT环境下使用Oracle并且发现GUI工具没有提供某个恢复选项，记住你仍然有UNIX数据库管理员可以得到用于恢复的几乎所有命令行工具。

24.8.2 完全恢复

假如一个数据文件丢失的话，完全恢复（complete recovery）在不会丢失任何事务的情况下使你的数据库起死回生。假如一个数据文件丢失或遭到破坏的话，完全恢复使用重做日志文件与归档日志文件来阻止数据的丢失。必须强制性地使控制文件、重做日志和归档日志驻留在与数据文件不同的物理磁盘上。归档日志或重做日志如果和数据文件一起丢了，你就不能够在没有数据丢失的情况下进行恢复（这被称为不完全恢复）。

完全恢复的思想是从一个热备份或冷备份中恢复一个已丢失的数据文件的较老的（并且是有效的）拷贝，然后重新运用自恢复的数据文件得到备份以来发生的所有变化。许多RDBMS（包括Oracle）使用前滚操作提供一个完全恢复，意指“重新运行”重做日志与归档日志中包含的操作。Oracle保存一个称为序列号（sequence number）的主里程表，该序列号记录在数据文件与控制文件中。通过查看控制文件，Oracle便可以知道所有的数据文件已经记录的序列号。假如某个数据文件没有包含当前的序列号，Oracle就知道它不是当前的数据文件并且需要在打开数据库以前得到恢复。

通过查看需要恢复的数据文件中的序列号，Oracle知道它需要从哪里开始重新运行事务。

Oracle期望发现至少一组当前重做日志以及所有的归档日志文件，Oracle在归档日志目录（在init.ora文件中设置）中需要这些文件。因此，你应仅删除在最后一个备份开始以前被记录的归档日志目录中的文件。假如你遗漏了任何一个归档日志或已经丢失了重做日志的所有拷贝，你只能执行一个不完全恢复——你不能够恢复到最后一次提交时的状态。

提示 假如你碰巧将归档日志文件放入几个不同的目录中，当Oracle在缺省目录中（假如你正在使用命令行恢复）不能找到一个指定的归档日志文件时，它会提示你。此时，你应能够提供指向该归档日志文件的完整路径。

当丢失了数据文件时，你需要关闭数据库（如果该数据库还没有关闭的话）。必要时使用SHUTDOWN ABORT关闭它。假如你遇到了硬件故障，你需要在继续工作以前修理该硬件问题。

1. 命令行环境

既然操作系统得到备份并正在运行，那么就检查数据库的损坏情况。当你知道丢失了哪些数据文件后，你（或系统管理员）将把它们恢复到它们原先所在的位置中。即使丢失了某些重做日志或控制文件的话，也不要从备份中恢复任何重做日志或控制文件。

假如你丢失了一个控制文件，你可以用当前控制文件的拷贝来替换它。所有用于你的实例的控制文件在init<sid>.ora文件中列出。

Oracle将使用存放在系统中不同位置上的重做日志的拷贝来替换已丢失的重做日志。在修理了数据文件后，再恢复丢失的重做日志文件。

在继续操作以前，仔细检查自上次备份开始以来你所拥有的全部归档日志。它们应全部存放在归档日志目录中（或至少在系统中的一个或多个其他目录中）。现在准备把老数据文件前滚到最后一次提交时的水平：

1) 启动服务器管理器。

2) 像平常一样启动数据库。

3) 当数据库试图打开时，Oracle会发现一个或多个数据文件含有过时的序列号。Oracle将停下来等待你恢复该数据库。你现在应键入如下语句：

```
RECOVER AUTOMATIC DATABASE ;
```

4) 当恢复结束后，过时的数据文件将得到前滚。

5) 正常关闭数据库。假如你丢失了一个重做日志文件，你可以用一个来自相同组号的当前且有效的重做日志文件拷贝替换它。通过查询V\$LOG，你可以确定同一个重做日志组中含有什么文件。

6) 执行数据库的冷备份。

7) 正常启动数据库。

2. Windows NT环境

此处描述了一个与前面所描述的过程完全相同的恢复过程，但是针对Windows NT的。尽管你可以使用命令行工具恢复一个Windows NT环境下的Oracle数据库，然而使用Oracle的图形用户界面工具更容易而且更直观。

你需要鉴别哪一个文件丢失了。当你正在使用NT恢复管理器完全恢复一个数据库时，拥有一张数据库中所有数据文件的列表对你是有帮助的。只要把一个数据文件添加到数据库中就应该启动NT恢复管理器，然后再关闭它。NT恢复管理器在每次启动以及数据库正在运行时

建立一个数据文件、控制文件和重做日志的列表。通过这样做，你可以通过浏览 NT恢复管理器的数据文件列表来确定哪一个文件丢失或遭到破坏。你仍然需要有在你的基于 NT的Oracle实例中的数据文件的纸张拷贝。

在Windows NT环境中，Oracle的NT恢复管理器几乎管理与完全恢复有关的每件事情（假设全部正常的话）。在开始恢复以前，你应仔细检查自上次备份开始以来你所有的归档日志。这些文件应全部位于 init<instance>.ora或config<instance>.ora文件中所指定的归档日志目录中。假如你把你的数据库备份到磁盘上，就需要保证备份文件处于联机状态。否则你需要手头上有备份磁带。

执行如下这些步骤以恢复你的 Windows NT Oracle数据库；

- 1) 设法通过服务器管理器正常启动 Oracle。这会失败，但这是预料之中的，因为存在遗失或受到破坏的文件。
- 2) 启动Oracle NT恢复管理器。
- 3) 选择Restore data file, then do recovery选项。
- 4) 单击Files按钮。一个类似于图 24-5的窗口显示出来。

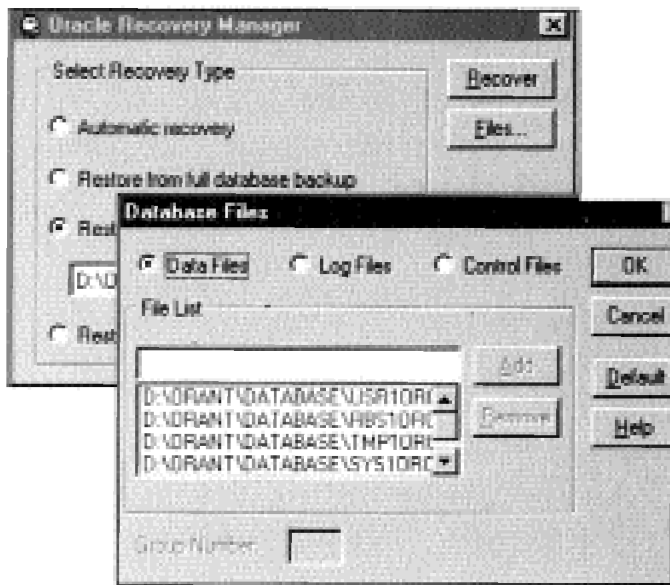


图24-5 选择要使用NT恢复管理器恢复的数据库文件

- 5) 单击Control Files单选按钮。
- 6) 通过单击每一个控制文件的文件名和 Remove按钮删除所列举出的任何控制文件。
- 7) 单击Log Files单选按钮。
- 8) 通过单击每一个重做日志文件的文件名和 Remove按钮删除所列举出的任何重做日志文件。
- 9) 单击Data Files单选按钮。
- 10) 删除所列举出的不需要恢复的全部数据文件。剩下的列举出的文件应是那些遗失或受到破坏的文件。
- 11) 单击OK。

12) 单击Recover。

13) 选择存放你的数据库备份的地方。选择 Restore from tape 或者 Restore from disk。假如你选择了一个磁盘位置, 确保在 Directory 域中所显示的目录就是你的备份文件所在的地方并且确保 <latest backup > 显示在 Backup 域中。

14) 单击OK。

15) 当另一个会话打开时, NT恢复管理器或许会发出一些关于关闭数据库的警告。在弹出的任何对话框中单击OK。

16) 正常启动数据库。

NT恢复管理器会使用重做日志与归档日志自动前滚最后一次备份中的数据文件。假如你需要恢复任何遗失的重做日志文件的话, 完成如下的步骤:

- 1) 关闭数据库。
- 2) 用来自同一组号的一个文件拷贝替换遗失的重做日志文件。
- 3) 重新启动数据库。

假如你需要恢复任何遗失的控制文件的话, 完成如下的步骤:

- 1) 关闭数据库。
- 2) 用一个当前控制文件的拷贝替换遗失的控制日志文件。
- 3) 重新启动数据库。

3. 恢复管理器

恢复管理器能够提供用于执行物理数据库恢复的完整环境。因为恢复管理器维护一个备份目录, 所以它能够精确地跟踪一个给定的恢复方案需要使用什么样的备份集。恢复管理器能够实际消除曾经在恢复数据库时所做的一些操作系统级的工作。恢复管理器提供了许多恢复选项, 其中包含如下选项:

- 整个数据库或表空间的时间点恢复。
- 个别数据文件的恢复。
- 个别表空间的恢复。
- 把数据文件恢复到不同位置。
- 数据库的完全恢复。

就像备份一样, 恢复管理器提供了一大组超出本节范围的恢复选项。请参考 Oracle的备份与恢复向导得到一个对恢复管理器功能的完整描述。

注意 记住, 要使恢复管理器能够恢复数据库, 它必须在 NOMOUNT 模式下启动。这与大多数其他的恢复系统有所不同, 这些恢复系统使或需要数据库处于完全脱机状态。

4. 示例

在下面的例子中, 你有一个刚刚崩溃且破坏了数据文件的 UNIX 系统。数据库现在当然不能启动, 因为 Oracle 不能使所有的数据文件有效。在本例中, 下述内容是客观的:

- BT 的目录已经在另一个 Oracle 8 数据库 TST8 中注册 (TST8 已正确地向 Net 8 注册)。
- BT 与在 initBT.ora 文件所指定的目标目录中, 所有新近的归档日志联机运行在 ARCHIVELOG 模式下。
- 已经使用恢复管理器建立了一个合适的备份周期。

警告 尽管恢复管理器经过非常好地设计以避免进一步地破坏一个已经受到破坏的数据

库，但是在开始任何恢复过程以前，你总是应首先关闭 Oracle，然后执行一个所有 Oracle组件的完全操作系统级备份。

在下面的示例会话中，你可以看到 BT数据库正在服务器管理器中被启动。注意 Oracle已经发现2号文件丢失。Oracle还显示与2号文件有关的文件名，然而，恢复管理器仅需要文件号用来恢复正确的数据文件：

```
[tgasper@oreo BT]$ svrmgrl

Oracle Server Manager Release 3.0.5.0.0 - Production

(c) Copyright 1997, Oracle Corporation. All Rights Reserved.

Oracle8 Release 8.0.5.0.0 - Production
PL/SQL Release 8.0.5.0.0 - Production

SVRMGR> connect internal;
Connected.
SVRMGR> startup
ORACLE instance started.
Total System Global Area                4758800 bytes
Fixed Size                              48400 bytes
Variable Size                           4227072 bytes
Database Buffers                       409600 bytes
Redo Buffers                           73728 bytes
Database mounted.
ORA-01157: cannot identify data file 2 - file not found
ORA-01110: data file 2: '/u02/oradata/BT/BT_data1.dbf'
SVRMGR> exit
```

提示 当一个数据库处于MOUNT模式时，你可以在V\$DATABASE视图中查询FILE#与NAME域，这样就可以看到数据文件的完整列表以及它们相关的文件号。然后，你可以在操作系统级检查以准确地发现已经丢失的文件以及需要在恢复管理器中得到恢复的文件。

你已经知道接下来需要恢复 2号文件，下面的示例会话显示了恢复管理器正从备份中恢复丢失文件并使用 RECOVER DATABASE命令把它完全更新。当从命令行运行恢复管理器时，注意如何使用RCVCAT关键字指定目录模式拥有者和数据库：

```
[tgasper@oreo BT]$ rman

Recovery Manager: Release 8.0.5.0.0 - Production

RMAN>

Recovery Manager complete.
[tgasper@oreo BT]$ rman target=sys/change_on_install rcvcat=rman/rman@tst8

Recovery Manager: Release 8.0.5.0.0 - Production

RMAN-06005: connected to target database: BT
RMAN-06008: connected to recovery catalog database

RMAN> run {
2> allocate channel c1 type disk;
3> restore (datafile 2);
4> recover database;
5> release channel c1;
6> }
RMAN-03022: compiling command: allocate
RMAN-03023: executing command: allocate
RMAN-08030: allocated channel: c1
RMAN-08500: channel c1: sid=10 devtype=DISK
```

```
RMAN-03022: compiling command: restore
RMAN-03022: compiling command: IRESTORE
RMAN-03023: executing command: IRESTORE
RMAN-08016: channel c1: starting datafile backupset restore
RMAN-08502: set_count=1 set_stamp=362157796
RMAN-08019: channel c1: restoring datafile 2
RMAN-08509: destination for restore of datafile 2: /u02/oradata/BT/BT_data1.dbf
RMAN-08023: channel c1: restored backup piece 1
RMAN-08511: piece handle=BT.FULL.BTxxxxxx.1.1L
RMAN-08024: channel c1: restore complete
RMAN-03023: executing command: partial resync
RMAN-08003: starting partial resync of recovery catalog
RMAN-08005: partial resync complete

RMAN-03022: compiling command: recover
RMAN-03022: compiling command: recover(1)
RMAN-03022: compiling command: recover(2)

RMAN-03022: compiling command: recover(3)
RMAN-03023: executing command: recover(3)
RMAN-08054: starting media recovery
RMAN-08055: media recovery complete

RMAN-03022: compiling command: recover(4)

RMAN-03022: compiling command: release
RMAN-03023: executing command: release
RMAN-08031: released channel: c1

RMAN> exit
```

在本例子中，你仅恢复一个数据文件，随后恢复该数据库。然而，你可以按需要增加 RESTORE 命令以恢复所有遗失的数据文件。你还可以省略 RECOVER DATABASE 命令并从另一个服务器管理器会话中自己处理最终前滚。

24.8.3 不完全恢复

当你完全恢复一个数据库时，你的目的是要避免丢失任何已提交的事务。要做到这一点，你必须从最后一次备份开始以来的所有的归档日志以及当前重做日志。假如你没有所有的这些文件，你只能执行一个不完全恢复——数据库要丢失一些事务。

提示 在开始一个不完全恢复以前，通常明智的做法是和Oracle技术支持一起检查以确信你真的没有其他的选择。特殊的环境也许能够启用完全恢复，通常情况下这是不太可能的。

一般说来，应避免此类恢复。然而，有很多情况需要你数据库恢复到过去的某个时刻所具有的状态。

1. 示例

假设一家银行由于掩盖金钱贪污而正受到审计员的调查。审计员希望查看掩盖以前的金钱贪污期间的银行数据库。可以这样说，数据库管理员能够在正确设置热备份或冷备份以及适当的归档日志的情况下恢复数据库，然后审计员便可以检查贪污期间的数据库，使犯罪得到曝光。

2. 命令行环境

假如一个归档日志文件遗失，Oracle将停止恢复过程并提示你把遗失的文件放入归档日志

目录中。因为你没有可用的此日志文件，不管什么原因，你需要告诉 Oracle使用到目前为止已经恢复的日志文件打开数据库。Oracle不能跳过一个归档日志并用随后的日志继续恢复。清单 24-3显示了一个用已丢失的数据文件对数据库进行完全恢复的例子。示例系统遗失了 303号存档日志文件。

清单24-3 Oracle服务器管理器归档日志

```
[tgasper@oreo dbs]$ svrmgrl
Oracle Server Manager Release 3.0.5.0.0 - Production
(c) Copyright 1997, Oracle Corporation. All Rights Reserved.
Oracle8 Release 8.0.5.0.0 - Production
PL/SQL Release 8.0.5.0.0 - Production
SVRMGR> connect internal;
Connected.
SVRMGR> startup
ORACLE instance started.
Total System Global Area                2260240 bytes
Fixed Size                              48400 bytes
Variable Size                          1728512 bytes
Database Buffers                       409600 bytes
Redo Buffers                           73728 bytes
Database mounted.
ORA-01113: file 2 needs media recovery
ORA-01110: data file 2: '/u01/oradata/BT/BT_users1.dbf'
SVRMGR> recover automatic database;
ORA-00279: change 68782 generated at 04/07/99 00:47:34 needed for thread 1
ORA-00289: suggestion : /u01/app/oracle/admin/BT/arch/archT0001S0000000303.ARC
ORA-00280: change 68782 for thread 1 is in sequence #303
ORA-00278: log file '/u01/app/oracle/admin/BT/arch/archT0001S0000000303.ARC'
no longer needed for this recovery
ORA-00308: cannot open archive log
'/u01/app/oracle/admin/BT/arch/archT0001S0000000303.ARC'
ORA-27037: unable to obtain file status
Linux Error: 2: No such file or directory
Additional information: 3
Specify log: {<RET>=suggested | filename | AUTO | CANCEL}
```

Oracle目前正等待你告诉它下一步做什么。因为你的完全恢复失败了，所以你现在必须开始一个不完全恢复。首先，键入 CANCEL取消当前恢复。然后正常关闭数据库。

表空间 USERS使用 BT_users1.dbf，由于遗失的归档日志，Oracle不能完全恢复 BT_users1.dbf。无法把 BT_users1.dbf 文件带回到一致性状态，此一致性状态是相对于数据库剩余部分来说的。你唯一可供选择的安全方法（从数据一致性立场出发）是从你最后一次备份中恢复所有的数据文件并通过 302号日志应用归档日志。你将丢失发生在 302号日志后的所有事务。这里是你需要采取的步骤：

- 1) 从你最后一次备份中恢复所有的数据文件。
- 2) 设法正常启动数据库。
- 3) 当Oracle需要恢复时，键入此 SQL 语句；
RECOVER AUTOMATIC DATABASE；
- 4) 当Oracle不能找到你遗失的归档日志文件时，它将停止恢复过程。
- 5) 通过键入 CANCEL取消恢复过程。
- 6) 使用如下这条 SQL 语句执行一个不完全恢复：
RECOVER DATABASE UNTIL CANCEL；

- 7) 在提示符下用 CANCEL 进行回答。
- 8) 使用如下 SQL 语句强行打开数据库：
`ALTER DATABASE OPEN RESETLOGS;`
- 9) 立刻关闭数据库。
- 10) 执行一个冷备份。
- 11) 重新正常启动数据库。

提示 不必回滚整个数据库，Oracle8的恢复管理器（RMAN）能够对单个表空间执行时间点恢复。就像大多数复杂的恢复操作一样，在使用表空间的时间点恢复以前，你应该同Oracle的技术支持进行协商。

警告 保证只要你用 RESETLOGS 打开数据库，你就执行一个冷备份。假如你不执行一个冷备份，那么任何归档日志文件就会没用，因为 RESETLOGS 使当前的归档日志文件与 RESETLOGS 之前的数据库备份不兼容。

通过大量的工作，你可以把 USERS 表空间中包含的数据恢复到 302 号日志，在没有丢失其他表空间中的数据的情况下把该数据从逻辑上移动到数据库中。但这么做可能引起你的数据的引用完整性问题。

尽管此操作的具体过程超出了本书的范围，然而这里有常用的方案：

- 1) 把你对系统的最后一次备份恢复到一台测试机器上。你需要人工重新创建重做日志。
- 2) 通过 302 号日志把所有的归档日志拷贝到测试机器上。
- 3) 使用 RECOVER DATABASE UNTIL CANCEL 命令恢复数据库。
- 4) 在所有的归档日志上单步执行，直到到达 302 号日志。在 302 号日志之后，取消恢复。
- 5) 使用 RESETLOGS 选项打开数据库。
- 6) 执行 USERS 表空间中所有表的逻辑备份。
- 7) 在你失效的系统中重新创建 USERS 表空间。
- 8) 把在步骤 6 中导出的表从逻辑上恢复到丢失 USERS 表空间的实例中。

在诸如此类的情况下，最好让 Oracle 技术支持参与到该过程中。他们的专业知识将为你提供有关你的特殊恢复状态的所有细节。假如你需要继续进行此操作过程，在尝试任何类型的恢复以前请 Oracle 技术支持出马——不要尝试不完全恢复。

3. 重新命名并移动数据库文件

你经常需要重新命名你的数据库中的文件或把它们从服务器中的一个位置移动到另一个位置。Oracle 使你能够这样做，但是必须遵循指定的步骤。这些步骤中最为重要的是你要在业余时间完成所有的这类工作。你需要关闭数据库以便执行许多这样的操作。在 Oracle 中删除一个重做日志文件、修改控制文件项或执行其他的操作不会物理地删除或重新命名操作系统中的文件。你必须在操作系统层人工执行这些操作。

提示 无论何时，只要超过一个或两个数据库文件被移动，那么使用已经被校正（理想情况是在一个测试系统中检查）过的脚本执行操作系统与 Oracle 任务是明智的。

4. 控制文件

控制文件的位置用 control_files 关键字保存在适当的 init.ora 或 config.ora 文件中。要移动控制文件，应完成如下步骤：

1) 关闭数据库。

2) 把一个控制文件复制或移动到新位置。

3) 编辑适当的init<sid>.ora文件。control_files关键字之后有控制文件列表。你可以改变控制文件的位置或名称、从列表中删除一个控制文件或把一个位置加入一个新的控制文件中。

4) 启动数据库。

5. 重做日志

重做日志必须在SQL*Plus或服务器管理器中改变。从技术上讲，你可以在数据库启动和运行时执行此操作，但是你最好在证实系统中没有活动时做这项操作。采用下面的步骤移动或重新命名重做日志：

1) 关闭数据库。

2) 在EXCLUSIVE模式下启动数据库。如下的示例显示了在服务器管理器中进行此项操作：

```
STARTUP EXCLUSIVE;
```

3) 使用此SQL语句找出你需要改变的文件组号：

```
SELECT * FROM V$LOGFILE;
```

记录下你要处理的组号。

4) 现在，确定哪一个重做日志组是活动的：

```
SELECT * FROM V$LOG;
```

5) 活动的重做日志组用CURRENT标识，显示在步骤4的查询结果的STATUS列中。假如你需要移动的文件当前处于一个活动组中，那么把Oracle转换到下一个重做日志组：

```
ALTER SYSTEM SWITCH LOGFILE;
```

6) 现在删除重做日志组。在本示例中，你正在删除2号日志组。在你的系统中，使用从步骤3中得到的号码。

```
ALTER DATABASE DROP LOGFILE GROUP 2;
```

7) 在你希望的位置用文件创建重做日志文件组。在本示例中，用两个文件重新创建2号组（你在每一个重做日志组中至少有两个文件）。本例中，为每个文件使用1M字节的空间。显而易见，你的组号、实际文件名和文件容量将根据你的要求而变化。

```
ALTER DATABASE ADD LOGFILE GROUP 2
```

```
(' /u03/oradata/test/log2atest.log', '/u04/oradata/test/log2btest.log') size 1M;
```

8) 关闭数据库，然后重新正常启动它。

6. 数据文件

数据文件来回移动相当容易。一般使用服务器管理器和一个操作系统工具实际移动或重新命名数据文件。假如作为物理恢复的一部分重新定位文件，那么只需把数据文件放置在你所希望的地方。下面是改变数据文件位置/名称的步骤：

1) 假如数据库正在运行的话，那么就关闭它。

2) 使用一个操作系统工具把文件移动到一个新的位置或给它重新命名。确保完全了解数据文件曾经在哪儿以及目前在哪儿。

3) 在MOUNT模式下启动数据库。如下的语句显示了服务器管理器中进行此项操作：

STARTUP MOUNT;

4) 对于每一个改动过的文件, 在 Oracle中设置新的文件名。如下的示例在内部修改 Oracle以使它将在正确的位置中寻找数据文件。在这里, 把 rbs1test.dbf文件从/u01/oradata/test 文件系统移动到/u02/oradata/test文件系统中。

```
ALTER DATABASE RENAME FILE  
'/u01/oradata/test/rbs1test.dbf' TO  
'/u02/oradata/test/rbs1test.dbf';
```

5) 现在关闭数据库, 然后重新启动它。假如该数据库由于遗失了一个数据文件而不能启动, 那么你需要找出该文件在哪里并使用 Oracle的RENAME工具以使Oracle发现它所需要的文件。

7. Windows NT环境

不幸的是, Oracle的NT恢复管理器不能完美地处理不完全恢复。由于这个原因, 使用 Oracle的NT恢复管理器执行不完全恢复任务最好在 Oracle技术支持的帮助下完成。不完全恢复是精密的过程, 因此你不能冒险让 NT恢复管理器发出一条使你的数据库永久不能使用的 SQL语句。

警告 尽管有Oracle专家的帮助, 但是在没有对受损数据库进行完全备份以前决不要开始任何恢复过程。只有这样, 在恢复进程失败的情况下, 你才可以退回并从头开始。

24.9 测试策略

在本章中, 已经讨论了大量与备份和恢复 Oracle数据库有关的资料。大部分数据库管理员发现每一个站点都需要独特的备份与恢复操作。因为你严重依赖于你的备份与恢复方案, 所以你必须肯定它们确实可靠。一个未经过测试的备份与恢复方案比没有方案还坏——它会给你一种假的安全感。

理想的情况是你有一台与你想要保护的机器完全一样的机器, 可以用它来测试备份与恢复方案。通过把成品系统复制到你的测试系统 (当然在数据库关闭的情况下), 开始你的备份与恢复方案。通过模拟各种失败 (切断磁盘驱动器的电源、删除文件、破坏重做日志等) 以及从这些失败中进行恢复, 你将获得无价的经验, 这些经验无法通过读书或参加任何讲座得到。假如在执行这些操作时, 能够模拟系统负载, 你会获得最好的经验并且保证你的备份与恢复方案使你免遭任何失败。

假如你像大多数数据库管理员一样没有一个用于测试的完整系统, 那么就在一台最类似于你需要保护的机器上创建一个实例。最好挑选一台没有 Oracle实例的机器。这将把破坏一个工作中的Oracle数据库的可能性减到最小。如果空间允许的话, 尽可能地把成品数据库拷贝到此测试实例中并且尽可能地模仿文件系统 /磁盘格式。当模拟失败并设法从失败中恢复时, 使用那些与你在成品系统中使用的完全一样的备份与恢复产品。

提示 不管你如何测试你的备份与恢复方案, 一定要用文档记录你所采用的确切过程。还要记录你在哪里遇到了困难以及采用了何种解决方案。只要你在成品系统上经历了一次失败, 你就应把当时的情况添加到你的失败 /恢复练习中。只要该失败曾经发生, 它就有可能再次发生。无论何时, 只要一名新的数据库管理员加入你的维护人员行列, 就让他或她接受系统的事故与恢复训练。

24.10 问题解决策略

1. 出故障的热备份

在数据库的热备份期间，系统发生崩溃，必须重新启动。尽管没有丢失任何数据，但 Oracle 服务器指示系统必须执行介质恢复操作。因为数据文件仍然处在备份模式下，所以 Oracle 认为它们需要得到恢复（这可以通过查询 V\$BACKUP 视图得到确认）。实际上不需要进行恢复，只需为每一个仍然处于备份模式下的文件使用 ALTER DATABASE DATAFILE END BACKUP 语句。关闭数据库，然后正常启动它。

2. 表的导入的约束条件

当把数据导入到一个已存在的表集合时，各种约束条件（例如外关键字关系）使数据不能被导入。

为了随意导入数据，有时需要使用 ALTER TABLE DISABLE CONSTRAINT 语句暂时停用诸如外键或唯一键的各种约束条件。注意当发出 ALTER TABLE ENABLE CONSTRAINT 语句以在导入进程结束后重新启用约束条件时，在实际启用这些约束条件以前，Oracle 首先要确认所有的数据满足这些约束条件。这在一张较大的表上会花费一些时间。

24.11 项目：被破坏的归档日志的不同涵意

如果在试图执行一个完全恢复时，Oracle 提示某个归档日志文件遭到破坏或遗失并且不能用了。那么希望你尽快地执行一个恢复操作。

一般来说，你很难做出选择。你可以执行下面的其中一个操作：

- 马上停止恢复过程，执行不完全恢复的指令启动该数据库。这将丢失自坏的归档日志创建以来的所有事务。
- 把数据库恢复到你丢失一个或多个数据文件的初始失败点，然后从数据库中删除丢失或受损的数据文件。你将失去已丢失的数据文件中的任何数据，但是假如你能够从一个逻辑备份的数据文件或另一个数据库系统的拷贝中恢复丢失的信息的话，或许是很有吸引力的。所有剩余的数据就是现存的数据。

假如你选择第二个选项，当数据库处于已安装但没有打开的状态时，你可以使用 ALTER DATABASE DATAFILE OFFLINE 语句将丢失或受损的数据文件脱机，如下所示：

```
SQL> ALTER DATABASE DATAFILE '/u01/oradbs/PRD/PRD_custindx1.dbf' OFFLINE;
```

通过使数据文件脱机，你能够打开数据库并检查受损情况。在打开数据库后，查询 DBA_DATA_FILES 以获得丢失或受损的数据文件的文件号，如下例所示：

```
SQL> select file_id,file_name from dba_data_files where
2 file_name = '/u01/oradbs/PRD/PRD_custindx1.dbf';
```

```
FILE_ID FILE_NAME
```

```
-----
3 /u01/oradbs/PRD/PRD_custindx1.dbf
```

你可以在这里看到丢失文件的文件号是3。现在，在下面的例子中，通过查询 DBA_EXTENTS，可以发现哪一个段已经受到失败的影响：

```
SQL> select owner,segment_name,segment_type from dba_extents
2 where file_id = 3
```

```
OWNER
```

```
SEGMENT_NAME
```

```
SEGMENT_TYPE
```


PRDSYS	MCUST_AKEY1	INDEX
PRDSYS	MCUST_AKEY2	INDEX
DBAUSR	AUDIT_PKEY	INDEX
PRDSYS	INVTRY_IDX3	INDEX
PRDSYS	INVTRY_AKEY1	INDEX
BATCHUSR	JOBRESULT_PKEY	INDEX
BATCHUSR	JOBRESULT_IDX1	INDEX

这里，你可以看到唯一受到丢失数据文件影响的段是索引——它总能够从原始表中得到重建。实际上，尽管你丢失了信息，然而，你并没有丢失不能重新创建的信息。

执行如下这些步骤重新构造这个系统：

- 1) 删除前面的查询所显示的索引。在删除以前，你应当确信你拥有稍后用于重建这些索引的DDL代码。
- 2) 使用ALTER DATABASE DATAFILE OFFLINE DROP语句删除受损的数据文件。
- 3) 重新创建Oracle中的数据文件。
- 4) 重新创建丢失的索引对象。

作为一条通用规则，在尝试一种不寻常的恢复方法以前，你应该同 Oracle技术支持商量一下。一般说来，Oracle是一个非常坚固的数据库系统，当你需要从一个受损数据库中可能有的东西时，它会提供帮助的。

即使是对于一个完全丢失的数据文件，你也可以使用表 24-1作为普通Oracle对象的可恢复性特征的准则。

表24-1 丢失文件的可恢复性特征

Oracle对象	可恢复性
表	假如多个区间保存数据，通常可以对那些没有保存在受到数据文件丢失影响的区间中的行进行查询。如果必要应与 Oracle技术支持协商
索引	像以前一样，索引完全是可恢复的。简单地删除数据库中的索引，重新创建受影响的存储区并重新创建该索引
回滚段	尽管启动 Oracle并删除活动回滚段在技术上是可行的，然而你最好还是同 Oracle技术支持商量以确保在你的特定环境下这意味着什么。删除受损的活动回滚段并继续数据库操作非常冒险，如果可能的话应该避免
临时段	它们一般不会引起严重的问题。一般你能够在不引起任何数据完整性问题的前提下重新创建任何丢失的临时表空间数据文件。任何受到影响的事务将被回滚，然后必须重新提交

第25章 完整性管理

本章要点：

实现锁

分析V\$Lock

监控系统中的锁

避免锁：可能的解决策略

锁定与分布式数据库

使用行实现锁

当数据库的容量与系统中的用户数量增加时，对内部资源的争用也变得越来越复杂，这样最终会导致性能出现问题。内部争用通常表现为有足够的空闲内存并且 CPU不是很忙，但是系统中仍然存在性能问题。上述现象使一个初学的管理员对系统中发生的事情感到惊奇。本章的目的就是帮助管理员识别这样的争用问题。本章要讨论的资源是锁与行。

数据库争用可以被分为两大类，稍后在本章中对这两者分别进行详细讨论。

■ 锁 —— 用于限制其他用户对数据的存取。

■ 行 —— Oracle使用的一个内部机制，用来管理和维护内存锁。

25.1 实现锁

■ 数据一致性 —— Oracle锁机制保证查询返回的数据与查询开始的时刻的数据一致。因此，即使其他用户正在改动数据，执行查询的用户只会看到“静态”的数据。这提供了数据一致性。

■ 数据并行性 —— 锁用于防止对同一对象进行存取的用户之间破坏性的相互作用。Oracle的锁机制使多个用户能够同时对数据进行安全存取。这提供了数据并行性。

■ 数据完整性 —— 事务在执行一个提交或回滚以前保留锁。这提供了数据完整性。

单用户数据库或许从来不需要锁，但是当你工作在一个多用户环境下，有一个合适地自动满足数据并行性、一致性和完整性的机制是重要的。Oracle通过使用一个内部锁定机制维护数据的完整性、并行性与一致性。Oracle如何维护数据的完整性、并行性与一致性将在随后的小节中讨论。

Oracle通过获得不同类型的锁，代表用户允许或阻止其他用户对相同资源的同时存取并确保不破坏数据的完整性，从而自动满足了数据的完整性、并行性与一致性。资源（resource）这个术语在这里指诸如表与行的用户对象。当实现锁时，要记住的一个关键问题是它不能在系统中形成瓶颈，阻止对数据的并行存取。因此Oracle根据所执行的数据库操作自动地要求不同层次上的锁定以确保最大的并行性。例如，当一个用户正在读取某行中的数据时，其他用户能够向同一行中写数据。然而，用户不允许删除表。这些涉及到使用不同锁定级别的事务能够进行什么级别的对象存取或数据存取的问题将在下节中讨论。

25.1.1 锁的需要

在你实际研究锁定以前，了解一下需要锁定的各种情况。下面是一些阐述了数据一致性与并行性为什么如此重要的实例：

- 在一个用户正在修改表中数据的同时，另一个用户正试图删除该表。Oracle通过为第一个用户取得一个表级锁定阻止此类事情的发生。第二个想要删除表的用户等待第一个用户释放了表级锁后才能够删除此表。
- 用户A正试图读取用户B的某个事务中的一些数据，在用户A的事务开始后，该事务由用户B修改和提交。用户A读取用户B提交的数据。这意味着在同一个事务中读取的数据在某个时刻不一致。Oracle保证在一个事务中读取的数据与事务开始时刻保持一致。Oracle使用系统变化号（SCN）实现事务级的读取一致性。

系统变化号（SCN）是一个数据结构，它定义了一个给定时刻提交的数据库版本。SCN可以被认为Oracle的逻辑时钟。每一次提交它们的数值都要增加。例如，如果某个事务执行了一个UPDATE或提交，该事务将被分配一个假定是500的SCN值，那么下一个提交的事务将得到一个501或更大的SCN值。

- 某个用户对数据进行修改，另一个用户在第一个用户提交事务以前对同一行进行修改；因此，第一个用户所做的改变丢失了。Oracle通过在该表的行上取得一个行级专有锁并让第二个用户等待的方式来防止此类情况的发生。
- 一个用户从另一个用户尚未提交的数据中读取数据；也就是说，在用户B的改变提交以前，用户A读取用户B正在修改的行。Oracle通过使用用户A能够读取老数据，也就是说，读取用户B修改以前的数据来防止此类情况的发生。Oracle通过从回滚段中获得数据执行此操作。

25.1.2 锁的概念

Oracle根据前后关系自动决定需要为给定的情况应用什么样的锁以便使用最小的约束提供最大的数据并行性。Oracle有两种锁：共享与专用。

共享锁（share lock）通过数据存取的高并行性来实现。假如获得了一个共享锁，其他用户可以共享相同的资源。许多事务可以获得相同资源上的共享锁，但是对于专用锁来说，情况就不是这样。例如，多个用户可以在相同的时间读取相同的数据。

专用锁（exclusive lock）防止同时共享相同的资源。例如，假如一个事务获得了某一资源上的一个专用锁，那么直到该锁被解除，其他的事务才能够修改那个资源。还有，也允许对资源进行共享。例如，假如一张表被锁定在专用模式下，它并不能阻止其他用户从同一张表中进行选择。

你可以使用锁的级别来实现并行存取并且同时保证数据的完整性。Oracle通过运用最低的约束条件来达到此目的。Oracle会根据需要自动锁定单行或整张表。Oracle使用下述的通用准则决定它需要使用的锁定级别：

- 一个用户正试图从一行中读取数据时，另一个用户应能够把数据写入同一行中。
- 一个用户正在更新一行的数据时，其他的用户应能够同时读取同一行的数据。
- 两个用户正在更新相同的表时，仅当他们试图存取相同的行时才被阻止。

使用锁定机制实现的一个最重要的功能是数据一致性。Oracle在两个层次上保证数据的一

致性：语句层和事务层。

语句层读取一致性（statement-level read consistency）意指任何一条开始执行的语句得到在语句开始执行以前已提交的数据。对数据所做的所有改变，不管是已提交的还是未提交的，在语句执行时对在提交会话以前开始的其他会话来说是不可见的。通过确保其他会话可以得到在语句开始执行时监测到的 SCN 以前提交的数据，Oracle 提供了语句层读取一致性（请参见表 25-1 中的例子）。

表 25-1 其他会话可以得到在语句开始执行时监测到的 SCN 以前提交的数据

时 间	会话-1	会话-2	会话-3
10:00	事务开始		
10:01		事务开始	
10:02	一致性读取 (不能看到会话-2提交的事务)	事务提交与结束事务	
10:03			事务开始
10:04	还是一致性读取		能够看到会话-2提交的数据
10:05			事务提交与结束事务
10:06	一致性读取（不能看到会话-2 与会话-3提交的事务		

事务层读取一致性（transaction-level read consistency）意指在同一个事务中的所有数据对时间点是一致的。Oracle 通过使用 SCN 机制以及获得所需的专用表锁与行锁来确保事务层读取一致性。

Oracle 通过使用回滚段的数据实现读取一致性。当一条查询进入执行状态时，当前的 SCN 就被确定了。当它开始读数据块的数据时，数据块的 SCN 与监测到的 SCN 进行比较。假如数据块 SCN 的数值大于查询开始时的 SCN，就意味着在该查询开始执行以后，那些数据块中的数据已经变化了。假如从数据块中读出此数据，那么它与查询开始时刻不一致。Oracle 使用回滚段重新构造该数据，因为回滚段总是包含老数据——系统中含有大量修改活动的情况除外，在这种情况下回滚段中的老数据可能被改写。

25.1.3 锁的类型

Oracle 根据操作系统和要在其上获得锁的资源的不同自动获得不同类型的锁。

有四种类型的 Oracle 锁：

数据锁（DML Lock）在表中获得并且用于保护数据，更确切地说是保护数据的完整性。

字典锁（DDL Lock）用于保护对象的结构——例如表和视图索引的结构定义。一个字典锁在 DDL 事务需要它时由 Oracle 自动获得。

内部锁与闕（internal lock 和 latch）保护内部数据库结构。闕在“使用闕实现锁”一节中详细讨论。

分布式锁与并行高速缓存管理（PCM）锁用于并行服务器中（参见第九部分“并行环境与分布式环境”）。

1. 数据锁

（1）DML 锁的用途

DML 锁的用途是保护被多个用户并行存取的数据的完整性。DML 锁可以防止同时进行的

DML操作的破坏性干扰。

例如，DML锁保证一张表中的指定行一次仅能被一个事务修改。它们还保证如果一个未提交的事务包含一个对表的插入操作时，该表便不能被删除。

DML语句使用两种类型的锁结构：

- 1) 事务获得表上的一个共享锁。
- 2) 事务获得它正在改变的每一行上的专用锁。

下面的例子显示了共享与专用类型的锁：

- 假如有三个用户视图同时对同一行进行修改，他们全都会得到共享的表锁，但是只有最先请求锁的用户得到行锁。
- 一个正被修改的行总是被专门地锁定以使其他用户不能修改该行，直到包含该锁的事务被提交或回滚为止。

(2) DML锁的模式

下面列举了用于DML的Oracle锁模式。

- 1) 行共享 (RS)：行共享锁是一个共享的表锁。它是一个在被查询行上的专用锁。
- 2) 行专用 (RX)：行专用锁通常表明持有该锁的事务已经对表中的行完成了一次或多次修改。
- 3) 共享锁 (S)：假如一个事务持有一个共享表锁，那么它便防止其他事务在该表中执行DML操作。
- 4) 共享行专用 (SRX)：假如一个事务包含一个SRX表锁，那么它防止其他事务执行DML或SELECT...FOR UPDATE语句。
- 5) 专用 (X)：专用锁是表锁的最具限制性的形式，它允许持有该锁的事务对该表进行独占访问。

注意 你可以通过增加DML_LOCKS与ENQUEUE_RESOURCES参数的值来增加用于一个实例的锁的数量。在V\$LOCK视图中，行锁与表锁分别用Lock Type列中的TX与TM值表示。V\$LOCK视图的Mode列中的SX值针对两种类型的锁。

2. DDL锁 (字典锁)

数据库管理员需要注意数据定义语言 (DDL) 锁以及它们对事务的影响。DDL锁将帮助你诊断并解决锁定问题。当模式对象被一个正在运行的事务引用时，DDL锁能够保护该模式对象的定义。

例如，当用户创建一个过程时，Oracle便自动获得在过程定义中引用的所有对象的DDL锁。

(1) DDL锁的功能

- DDL锁使在过程中引用的对象在过程编译结束以前不能被改变或删除。
- DDL锁仅影响在DDL操作期间被修改或引用的单个模式对象。它们不锁定整个数据字典。
- DDL锁不能被显式地实现。

(2) 不同形式的DDL锁

- 1) 专用DDL锁：当诸如CREATE、ALTER和DROP这样的语句用于一个对象时使用此锁。专用DDL锁的特征：

■ 假如另外一个用户保留了任何级别的锁的话，那么本用户不能得到表中的专用 DDL 锁。例如，假如另一个用户在该表上有一个未提交的事务，那么 ALTER TABLE 语句会失效。

2) 共享 DDL 锁：当诸如 GRANT 与 CREATE PACKAGE 这样的语句用于一个对象时使用此锁。

共享 DDL 锁的特征：

■ 一个共享 DDL 锁不能阻止类似的 DDL 语句或任何 DML 语句用于一个对象上，但是它能防止另一个用户改变或删除已引用的对象。

■ 共享 DDL 锁的另一个特征是在 DDL 语句执行期间它一直维持，直到发生一个隐含的提交。

3) 可破的分析 DDL 锁：库高速缓存中的语句或 PL/SQL 对象持有一个用于它所引用的每一个对象的锁。

可破的分析 DDL 锁的特征：

■ 假如被引用的对象改变了，可破的分析 DDL 锁检查语句是否应失效。

■ 只要相关的 SQL 语句仍然在共享池中，可破的分析 DDL 锁就会持续。假如对象改变了，它检查语句是否应失效。

3. 表锁

事务可以通过发出表 25-1 中的语句获得表锁（TM）。为了使事务能够保护表中的 DML 存取以及防止表中产生冲突的 DDL 操作，Oracle 获得表锁。例如，假如某个事务在一张表上持有一个表锁，那么它会阻止任何其他事务获取该表中用于删除或改变该表的一个专用 DDL 锁。

表 25-2 显示了不同的模式，当执行特定的语句时，由 RDBMS 获得这些模式的表锁（TM）。类型列有诸如 TM 的值，模式列的值分别为 2、3 或 6。TM 表示一个表锁；数值 2 表示一个行共享（RS）锁，3 表示一个行专用锁（RX），6 表示一个专用（X）锁。例如，类型列的 TM 值与模式列的数值 3 将被存储在 V\$lock 表中，当发出相应的语句时，发出该语句的会话要用到它们，所以你需要熟悉这些值。在 V\$lock 表中，Oracle 列举了当前由 Oracle 服务器持有的锁，这些锁将在 25.2 节中伴随例子详细讨论。

表 25-2 采用的语句与获得的表锁

语 句	类 型	模 式
INSERT	TM	行专用（3）（RX）
UPDATE	TM	行专用（3）（RX）
DELETE	TM	行专用（3）（RX）
SELECT FOR UPDATE	TM	行共享（2）（RS）
LOCK TABLE	TM	专用（6）（X）

当执行一条 INSERT 语句时，V\$lock 表中的类型列的值为 TM，模式列数值为 3，数值 3 意指行专用（RX）锁。要想得到一些辅助细节，参考 25.2 节。对于除锁表以外的所有语句，在 V\$lock 表中有两个项目：一个项目对应于表锁（TM），另一个项目对应于事务锁（TX）。对于诸如插入更新的语句来说，删除一个 TM 锁仅用来防止在被锁定对象上的相互冲突的 DDL 操作，这意味着当一个用户正向一张表插入时，他或她会获得一个模式 3 下的 TM 锁（RX）。假如另一个用户试图删除同一张表，就必须获得模式 6 下的 TM 锁（专用）。TM（3）（RX）锁防止 TM（6）（X）会话获得该锁并且第二个会话仍然要等待。表 25-3 说明了会话获得的锁模式

以及允许的锁模式。

表25-3 允许的锁模式和操作

SQL语句	表锁模式	允许的锁模式
select * from tname...	无	RS,RX,S,SRX,X
Insert Into tname...	RX	RS,RX
Update tname...	RX	RS*,RX*
Delete From tname	RX	RS*,RX*
Select...From tname For Update of...	RS	RS*RX*S*,SRX*
Lock Table In ROW SHARE MODE	RS	RS,RX,S,SRX
Lock Table In ROW EXCLUSIVE MODE	RX	RS,RX
Lock Table In SHARE MODE	S	RS,S
Lock Table In SHARE ROW EXCLUSIVE MODE	SRX	RS
Lock Table In EXCLUSIVE MODE	X	无
RS: Row Share	SRX: 共享行专用	
RX: Row Exclusive	X: 专用	
s: Share		

假如另一个事务已经获得了该行上的一个锁，那么将会发生等待。

事务锁（TX）：当一个事务发出表25-4中的语句时将获得此类型的锁。事务锁总是在行级上获得。TX锁独占地锁住该行并阻止其他事务修改该行，直到持有该锁的事务回滚或提交数据为止。

表25-4 事务锁语句

语 句	类 型	模 式
INSERT	TX	专用(6)(X)
UPDATE	TX	专用(6)(X)
DELETE	TX	专用(6)(X)
SELECT FOR UPDATE	TX	专用(6)(X)

要想获得TX锁，事务首先必须获得该表上的一个TM锁。例如，当发出一条INSERT语句时（参考表25-1），必须获得模式3（RX）中的一个TM锁。在获得了RX模式的TM锁后，事务必须获得专用（X）模式的TX锁（参见表25-4）。假如另一个事务在同一行上有一个TX锁，那么就不能获得TX锁；并且，假如在该表中已经有了一个专用（X）模式下的TM锁，那么也不能获得TM锁。

其他的模式是4（共享）与5（共享行专用），但是这些锁定模式一般不发生在数据库中，因此不必过多讨论。

25.1.4 人工锁定

假如你想要执行一张表中列值的全局更新并且希望事务对该表进行单独存取，以便该事务不必等待其他事务完成该表的操作，那么你可以通过人工锁定该表以防止其他事务获得该表中的锁，从而实现这一点。

示例：LOCK TABLE <Table Name> IN <mode>

你可以明确地在不同模式下锁定表：

- 行共享。
- 共享。
- 共享行。
- 共享行专用。
- 行专用。

当用户执行如下的语句时获得共享模式（4）锁：

```
SQL> LOCK TABLE <table name> IN SHARE MODE;  
Table(s) Locked.
```

当用户执行如下的语句时获得共享行专用模式（5）锁：

```
SQL> Lock table <table name> in SHARE ROW EXCLUSIVE MODE;  
Table(s) Locked.
```

现在你可以在不必与其他事务发生争用的情况下修改列值。

实际上，必须使用这些语句的应用很少；因此，同其他每日频繁发生的锁定类型相比，它们被认为是不很重要的。

25.1.5 数据库死锁

死锁（deadlock）是指两个或多个用户都在等待被彼此相互锁定的同一数据而形成的一种局面。

大部分数据库死锁发生在表索引中。来自数据库中一个单行的 SELECT 语句或许会在存储池中放置不止一个锁项目。一个单独的行收到一个锁，但是每一个包含该行数值的索引节点也将得到分配的锁。

Oracle 的共享锁定方案确保维护所有数据库的完整性并且更新不能无意中覆盖以前对数据库的更新。在维护共享锁时有一些不利条件。在 Oracle 中，每一个锁需要 Oracle 实例存储池中的 RAM 存储器的 4 个字节，大型的 SQL SELECT 语句会引起存储器短缺，这可能会损坏整个数据库。例如，假设一条 SELECT 语句把 1000 行检索到缓冲区中，那么就需要 4000 字节的锁定空间。这会引发数据库死锁。

25.2 分析V\$lock

Oracle 在动态状态表 V\$lock 中存储与数据库中的锁有关的所有信息。本节分析了当资源被锁定时数据库中的各种情况并且检查 V\$lock 表以查看 Oracle 如何报告该锁。

下面是对 V\$lock 结构的描述：

```
ADDR RAW(4)  
KADDR RAW(4)  
SID NUMBER  
TYPE VARCHAR2(2)  
ID1 NUMBER  
ID2 NUMBER  
LMODE NUMBER  
REQUEST NUMBER  
CTIME NUMBER  
BLOCK NUMBER
```

用于分析锁定状态的重要的列如下：

■ sid —— 这是会话标识符。

■ type —— 这是所获得的或会话等待的锁类型。示例值如下：

TX事务

TM DML或表锁

MR 介质恢复

ST 磁盘空间事务

本章仅涉及TX与TM锁。

■ lmode/request —— 本列包含锁的模式。可能的取值如下：

0无

1空

2 行共享 (RS)

3 行专用 (RX)

4 共享 (S)

5 共享行专用 (SRX)

6 专用 (X)

假如lmode列含有一个不是0或1的数值，表明进程已经获得了一个锁。假如 request列含有一个不是0或1的数值，表明进程正在等待一个锁。假如 lmode列含有数值0，表明进程正等待获得一个锁。

■ 下面的select语句是一种用于检查是否有会话正在等待获得任何表中的锁的快速方法。

```
Select count (*)  
From v$lock  
where lmode = 0
```

假如这个select语句返回一个大于0的数值，那么就表明系统中当前有锁会话正等待获得锁。

■ id1 —— 根据锁类型的不同，此列中的数值有不同的含义。假如锁类型是 TM，那么此列中的数值是将被锁定或等待被锁定的对象的标识。假如锁类型是 TX，那么此列中的数值是回滚段号码的十进制表示。

■ id2 —— 假如锁类型是TM，那么此列中的数值是0。假如锁类型是TX，那么此列表示交换次数 —— 也就是说，回滚槽重新使用的次数。

使用此信息，你现在可以检查各种情况以及分析不同情况下存储于此表中的数值。下节对数据库中经常发生的锁以及如何从 V\$lock视图中读出此信息进行详细的分析。

25.2.1 案例1：专用锁定的表

为了便于讨论，假设将要被锁定的表是 employee表并且会话标识是28。假设某个用户发出如下语句：

```
Lock table employee in exclusive mode;
```

此语句将在专用模式下锁定该表。假如一张表被一个用户专用锁定，那么另一个用户能够在该表上使用的唯一的SQL语句是select语句。在该锁被解除以前，不允许其他用户在此表上进行插入、修改或任何DDL操作。

使用下面的select语句检查V\$lock表中的记录：

```

Select sid,
       type,
       lmode,
       request,
       id1,
       id2
From v$lock
where sid = 28;

```

当执行此select语句时，得到如下输出结果：

SID	TY	LMODE	REQUEST	ID1	ID2
28	TM	6	0	7590	0

注意如下的监测结果：

- V\$lock表中仅有一项。
- 所获得的锁是TM（表）锁。
- lmode列的值为6，这表明会话已经获得了一个表级专有锁。
- id1列的值为7590，这是employee表的对象标识。正如对TM锁所预料的一样，id2列值为0。你可以使用sys.obj\$表得到对象描述。你可以使用如下的select语句：

```

Select name
      From sys.obj$
     where obj#      =      7590;

      Name
      .....
      Employee

```

- request列的值为0，这表明该锁已经被此会话获得。

25.2.2 案例2：会话更新专用锁定表的行

在案例2中，当另一个会话正试图修改一张表的某一行时，该表被一个会话专用锁定。这里是由试图修改该表的会话所发出的SQL语句：

```

Update employee
Set Name = 'MANISH'
where emp_id      =      '1086';

```

在本例中，在V\$lock表中第一个会话的项目与案例1中的仍然相同，但是第二个会话的项目非常有趣。假设第二个会话的会话标识是29。现在假设你执行如下的select语句：

```

Select sid,
       type,
       lmode,
       request,
       id1,
       id2
From v$lock
where sid in (28,29);

```

输出结果如下：

SID	TY	LMODE	REQUEST	ID1	ID2
28	TM	6	0	7590	0
29	TM	6	0	7590	0

28 TM	6	0	7590	0
29 TM	0	3	7590	0

下面是本例的监测结果：

- 会话28的项目仍然与案例1中的相同。
- 有一个会话29的项目。此项目针对 TM锁。注意lmode列的值为0，request列的值为3。这意味着会话29正等待获得该表中的一个表锁。会话29必须先获得表锁以标记它当前正在使用此表。这将防止其他任何会话在此表上发出 DDL语句。一般，获得此表锁标志着该表被会话使用。这里的id1列再次指出了已经被锁定的表的对象标识。
- 事务（TX）锁仍然没有被会话29获得；如果没有其他的会话正在存取或修改相同的行，仅在获得TM锁以后，会话29才能够获得TX锁。

此时如果会话28回滚或提交了事务，会话29就能够获得它正等待的锁。假如你在会话28已经提交了事务后在V\$lock表中检查会话29的项目，你可以得到两条记录。输出结果如下：

SID	TY	LMODE	REQUEST	ID1	ID2
29 TM		3	0	7590	0
29 TX		6	0	327680	10834

下面是监测结果：

- 在V\$lock中会话28的锁项目不再存在，因为它已经提交了事务。
- 会话29现在能够成功地获得TM锁，request列中的数值3移动到lmode列，表明该锁已被会话获得。
- 会话29还获得表中的另一个锁；它是一个事务锁或TX锁。在Type列中此选项的值为TX，lmode的值为6，表明是一个事务锁。
- id1中的值327680是回滚段的号码。要获得回滚段的名称，你可以使用V\$rollname视图。下面的select语句将给出回滚段的名称，会话29开始在此回滚段中写入回滚信息。

```

Select name
From v$rollname
where usn      =      trunc(327680/65536)

NAME
-----
RB03

```

因此，会话29开始在回滚段RB03中写入回滚信息。

- 不管何时，只要会话必须获得一个TX锁，那么它首先必须获得一个TM表锁。在获得该表锁后，只要没有其他会话已经在同一行上获得TX锁，该会话便可以获得TX锁。总之，该会话必须获得两个锁。

25.2.3 案例3：一个会话试图更新另一会话更新过的行

此锁定案例是每日锁定状况中最常见的情况。当两个会话试图同时修改同一行时，首先修改该行的会话得到锁，第二个会话要想得到锁就必须在第一个会话后等待。仅在第一个会话提交或回滚事务时，它获得的锁才能够被释放。

在本例中，会话29（第一个会话）已经把employee表的一条记录中的雇员名字更新为TOM并且还没有提交此修改。会话30（第二个会话）目前试图修改同一行。要分析V\$lock，你可以使用与上一个例子中相同的一条select语句，但是要用相关的sid值替换sid列：

SID	TY	LMODE	REQUEST	ID1	ID2
29	TM	3	0	7590	0
29	TX	6	0	327680	10834
30	TM	3	0	7590	0
30	TX	0	6	327680	10834

基于这个select语句的输出，可以得到如下的监测结果：

- 会话29（第一个会话）在V\$lock中有两个记录锁：TM锁和TX事务锁。
- 会话30在此表中有一个对应于TM表锁的项目。在此情况下，会话30能够立刻获得TM锁，因为没有其他会话排它地锁定此表。
- 会话30有另外一个对应于事务锁（TX）的记录。此项目的request列值为6。这意味着会话30正等待获得一个行专用锁。id1与id2列中的值与对应于会话29的id1与id2列中的值相同。
- 一个会话锁定了一行，而另一个会话正试图更新该行是最常见的锁定案例。
- 假如会话29现在回滚或提交与案例2中一样的事务，那么会话30可以获得该锁并且该锁项目的request列中的值将移动到lmode列中，表明已经获得了该锁。

25.3 监控系统中的锁

既然你已熟悉了锁定与相关的系统表的概念，那么很容易用脚本检测锁。在掌握了前一小节中所包含的信息的前提下，用户能够编写他自己的锁检测脚本。尽管如此，这里仍然给出了一些锁监控脚本。本节将包含一些用于检测和监控锁的有用的脚本。

清单25-1中的脚本将帮助你得到有关被锁定在数据库中的对象的数量。它将报告当前锁定的所有对象和它们的锁模式，它还将报告正在等待获得锁的所有会话。

清单25-1 用于检查系统中锁的简单脚本

```

Select s.username,
       s.sid,
       l.type,
       l.id1,
       l.id2,
       l.lmode,
       l.request,
       p.spid PID
From v$lock l,
     v$session s,
     v$process p
Where s.sid = l.sid
And   p.addr = s.paddr
And   s.username is not null
Order By id1, s.sid,request;

```

USERNAME	SID	TY	ID1	ID2	LMODE	REQUEST	PID
MARK	39	TM	4573	0	3	0	19271
MANISH	41	TM	4573	0	2	0	20155
KASTURI	116	TM	4573	0	2	0	19914
ADITI	125	TM	4573	0	2	0	19906
ADITI	95	TM	12547	0	2	0	19906
JYOTI	95	TM	15397	0	3	0	20364
KASTURI	116	TM	15397	0	2	0	19914
ADITI	39	TM	15397	0	2	0	19906
MANISH	41	TX	65548	91271	6	0	20155

ADITI	125 TX	65548	91271	0	6 19906
MARK	39 TX	196626	107701	6	0 19271
JYOTI	95 TX	262156	118264	6	0 20364
KASTURI	116 TX	327699	200758	6	0 19914

所有锁的类型是 TM（2或3）和 TX（6）。如前所述，这些锁是在任何数据库中最常见的锁。

快速浏览输出可以得到如下结果：

■ 有3个表对象——它们是4573、12547和15397——当前正被锁定在数据库中。

■ 会话39正把对象4573锁定在行专用模式中。

■ 会话41正把对象4573锁定在行共享模式中。

■ 会话116正把对象4573锁定在行共享模式中。

■ 会话125正把对象4573锁定在行共享模式中。

■ 会话95正把对象12547锁定在行共享模式中。

■ 会话95正把对象15397锁定在行专用模式中。

■ 会话116正把对象15397锁定在行共享模式中。

■ 会话39正把对象15397锁定在行共享模式中。

■ 会话95、116、125、39和41各有一个事务锁（TX）。最重要的观察结果是会话125在request列中有一个非零值，说明它正在等待一个锁。因为会话41和125的TX选项上的id1与id2值相匹配，你可以推断出事务41阻塞了事务125。因为事务125与事务41之间的公用资源是对象4573，所以你能够由此推断出会话41已经锁定了对象4573中的某一行并且会话125正试图锁定同一行——因此，它正在等待。

此输出结果容易解释，因为在会话125与41之间仅有一个公共资源并且可以很容易地指出正引起锁问题的资源。在大多数情况下，最常见的是在两个会话之间不止一个公共资源，当然它们中的一个将被锁定。在这种情况下，很难使用前面的输出结果解释哪一个资源被锁定。你必须使用V\$sqltext视图，它存储了属于系统全局区（SGA）中的共享SQL指针的SQL语句的文本。使用前面的输出结果，查找正等待获得锁的会话的标识，然后使用下面的select语句解释该会话正试图锁定以及正等待哪一个资源：

```
Select sqltext
from v$sqltext a,v$session b
where a.address = b.sql_address
and a.hash_value = b.sql_hash_value
and b.sid = 125
order by piece;
```

上述SQL语句的输出结果如下：

```
SQL_TEXT
-----
update employee set sname = 'JYOTI' where emp_id = '100';
```

这意味着会话41已经锁定了雇员记录100，会话125正试图锁定同一行。因此，使用锁监控报告以及SQL语句能够很容易地查明被锁定的资源。

通用SQL脚本

本节将提供几个通用脚本以帮助你确定数据库中的锁信息。

清单25-2是一个通用脚本，它显示了SQL文本、SID以及当前在数据库中持有的锁的对象名。

清单25-2 显示数据库锁的信息

```
Set pagesize 60
Set linesize 132
select s.username username,
       a.sid sid,
       a.owner||'|'||a.object object,
       s.lockwait,
       t.sql_text SQL
from v$sqltext t,
     v$session s,
     v$access a
where t.address = s.sql_address
and    t.hash_value = s.sql_hash_value
and    s.sid = a.sid
and    a.owner != 'SYS'
and    upper(substr(a.object,1,2)) != 'V$'
/
```

上述SQL语句的输出结果如下：

USERNAME	SID	OBJECT	LOCKWAIT	SQL
SHARON	8	MARK.CONFIG	E0034C98	update mark.config set irn=12 where ➤irn=4
KASTURI	12	KASTURI.EMP	E0034A5C	update kasturi.emp set tabno=10 ➤where tabno=5
ADITI	99	DAVID.CHANNEL	E0034D99	update david.channel set c_irn=3 ➤where c_irn=1

清单25-3是一个通用脚本，它产生一个持有（或等候）在数据库中的锁的易读报表。

清单25-3 产生在数据库中持有的锁的报表

```
select B.SID,
       C.USERNAME,
       C.OSUSER,
       C.TERMINAL,
       DECODE(B.ID2, 0, A.OBJECT_NAME, 'Trans-'||to_char(B.ID1)) OBJECT_NAME,
       B.TYPE,
       DECODE(B.LMODE, 0, '—Waiting—',
              1, 'Null',
              2, 'Row Share',
              3, 'Row Excl',
              4, 'Share',
              5, 'Sha Row Exc',
              6, 'Exclusive', 'Other') "Lock Mode",
       DECODE(B.REQUEST, 0, ' ',
              1, 'Null',
              2, 'Row Share',
              3, 'Row Excl',
              4, 'Share',
              5, 'Sha Row Exc',
              6, 'Exclusive', 'Other') "Req Mode"
from    DBA_OBJECTS A,
        V$LOCK B,
        V$SESSION C
where A.OBJECT_ID(+) = B.ID1
and B.SID = C.SID
and C.USERNAME is not null
order by B.SID, B.ID2;
```

上述SQL语句的输出结果如下：

Sess ID	Op USERNAME	Sys User ID	OBJ NAME or TERMINAL	TRANS_ID	TY Lock Mode	Req Mode
7	JYOTI	jla	JLA	TRANSMISSION	TA Row Excl	
8	JOHN	jxl	JXL	TABLE_CONFIG	TM Row Excl	
9	BONITA	bog	BOG	DEPT_SCHED	TX -Waiting—	Exclusive
10	MANISH	mdk	MDK	AUDIT	TX Exclusive	
12	MARK	mdb	MDB	INDEX_BLOCKS	TM Row Excl	

另一个有用的视图是 V\$session_wait，它列举了活动会话正在等待的事件。在 V\$session_wait 视图中登记的事件是 enqueue。因此，假如你查询 V\$session_wait 视图寻找一个被锁定的或者你认为被锁定的会话，你会得到清单 25-2 中的输出。

清单 25-4 中的语句是由 SQL*Plus 发出的，用于对 V\$session_wait 的查询输出结果进行格式化。

清单 25-4 使用 V\$session_wait 查询会话中的锁

```
col event format a8
col p2text format a5
col p2 format 999999
col p3text format a5
col p3 format 999999
col wait_time format 999999999
col secs format 99999
col state format a8
col seq# format 99999
set pagesize 24
set verify off

select event, p2text,p2 ,p3text,p3,seq#,wait_time,state,seconds_in_wait secs
from v$session_wait
where sid = 125;
```

上述SQL语句的输出结果如下：

EVENT	P2TEX	P2	P3TEX	P3	SEQ#	WAIT_TIME	STATE	SECS
enqueue	id1	65548	id2	91271	6845	0	WAITING	850

p2与p3列包含id1与id2值，id1与id2是能够从V\$lock中获得的数值。输出结果还报告了会话被锁定的秒数。

注意 当试图解释 V\$session_wait 的输出结果时，应当记住检查 state 列的值，因为它是一个非常重要的参数。此列可能的取值有：

- Waiting—表明会话当前正在等待事件的发生。当 state 列中的值为 Waiting 时，seconds_in_wait 列将包含会话用于等待事件所花费的以毫秒为单位的实际时间。
- Waited unknown time—当 TIMED_STATISTICS 参数被设置为 false 时，此值被显示出来。
- Waited short time—表明会话短期等待。
- Waited known time—表明会话已经获得了它正等待的事件。wait_time 列包含了会话实际等待的时间。

另一个重要的视图——V\$sysstat——包含了整个系统的所有重要的统计信息。你可以使用V\$sysstat对每一个会话等待锁定的次数有一个总的了解。

使用如下的SQL语句：

```
select decode(class,1,'User',
              2,'Redo',
              4,'Enqueue',
              8,'Cache',
              16,'OS',
              32,'Par Ser',
              64,'SQL',
              128,'Debug') class1,
       statistic#,name,value
from v$sysstat
WHERE name = 'enqueue waits';
```

上述SQL语句的输出结果如下：

Class	S#N	Name	Value
Enqueue	23	enqueue waits	19989

通过此SELECT语句，你能够推断出系统中有19989次对锁定的等待。注意当锁被释放时，此计数器被递增；假如会话当前正等待一个锁的话，此值没有被反应在计数器中。假如enqueue waits的值太高，那么应用程序需要重新检查——有可能对同时运行存取相同资源的作业进行重新排定。

清单25-5是一个用于产生对等待锁的用户（会话）的报告通用脚本。

清单25-5 产生等待锁的用户的报告

```
column username format A15
column sid      format 9990   heading SID
column type     format A4
column lmode    format 990    heading 'HELD'
column request  format 990    heading 'REQ'
column id1      format 9999990
column id2      format 9999990
break on id1 skip 1 dup
spool tfslickwt.lst
SELECT sn.username,
       m.sid,
       m.type,
       DECODE(m.lmode, 0, 'None',
              1, 'Null',
              2, 'Row Share',
              3, 'Row Excl.',
              4, 'Share',
              5, 'S/Row Excl.',
              6, 'Exclusive',
              lmode, ltrim(to_char(lmode,'990'))) lmode,
       DECODE(m.request, 0, 'None',
              1, 'Null',
              2, 'Row Share',
              3, 'Row Excl.',
              4, 'Share',
              5, 'S/Row Excl.',
              6, 'Exclusive',
              request, ltrim(to_char(m.request,'990'))) request,
       m.id1,
       m.id2
FROM   v$session sn,
       v$lock m
```

```

WHERE (sn.sid = m.sid AND m.request != 0)
      OR (   sn.sid = m.sid
            AND m.request = 0 AND lmode != 4
            AND (id1, id2) IN (SELECT s.id1, s.id2
                              FROM v$sqllock s
                              WHERE request != 0
                              AND s.id1 = m.id1
                              AND s.id2 = m.id2)
      )
ORDER BY id1, id2, m.request;
spool off
clear breaks

```

上述SQL语句的输出结果如下：

USERNAME	SID	TYPE	HELD	REQ	ID1	ID2
SYSTEM	12	TX	Exclusive	None	131087	2328
SCOTT	7	TX	None	Exclusive	131087	2328
AMY	8	TX	Exclusive	None	131099	2332
MANISH	10	TX	None	Exclusive	131099	2332
BOLDT	12	TX	None	Exclusive	131099	2332

清单25-6是一个通用脚本，它显示了锁并给出了 SID以及要删除的序列号；此脚本产生的报告给出了持有锁的会话的信息，还给出了使用 ALTER SYSTEM KILL SESSION命令进行删除所需要的信息。

清单25-6 显示持有锁的会话的信息

```

set linesize 132 pagesize 66

break on Kill on username on terminal
column Kill heading 'Kill String' format a13
column res heading 'Resource Type' format 999
column id1 format 9999990
column id2 format 9999990
column lmode heading 'Lock Held' format a20
column request heading 'Lock Requested' format a20
column serial# format 99999
column username format a10 heading "Username"
column terminal heading Term format a6
column tab format a35 heading "Table Name"
column owner format a9
column Address format a18
select  nvl(S.USERNAME,'Internal') username,
        nvl(S.TERMINAL,'None') terminal,
        L.SID||'|'||S.SERIAL# Kill,
        U1.NAME||'|'||substr(T1.NAME,1,20) tab,
        decode(L.LMODE,
                1,'No Lock',
                2,'Row Share',
                3,'Row Exclusive',
                4,'Share',
                5,'Share Row Exclusive',
                6,'Exclusive',null) lmode,
        decode(L.REQUEST,1,'No Lock',
                2,'Row Share',
                3,'Row Exclusive',
                4,'Share',
                5,'Share Row Exclusive',
                6,'Exclusive',null) request
from    V$LOCK L,
        V$SESSION S,
        SYS.USER$ U1,

```

```

SYS.OBJ$ T1
where L.SID = S.SID
and T1.OBJ# = decode(L.ID2,0,L.ID1,L.ID2)
and U1.USER# = T1.OWNER#
and S.TYPE != 'BACKGROUND'
order by 1,2,5 ;

```

上述SQL语句的输出结果如下：

Username	Term	Kill String	Table Name	Lock Hld	Lock Req
SAMUEL	ttyr6	7,5159	SAMUEL.TABLE_CONFIG	Row Excl	
YA-MEI	ttyr5	8,941	SCOTT.SCHEDULE	Row Excl	
BARRETT	ttyq3	6,8885	SYS.CLUSTERS	Exclusiv	
			SCOTT.ACCOUNT	Row Excl	

执行下述的命令，演示如何删除一个会话：

```
ALTER SYSTEM KILL SESSION '6,8885';
```

该命令将删除Barrett的已锁定的会话。

可以在\$ORACLE_HOME/rbms/admin目录中发现另外一个有用的脚本。此脚本被称为utllockt.sql，它能够以树格式打印锁信息。这使得信息容易阅读并且容易发现谁正在锁定资源。

25.4 避免锁：可能的解决策略

本节提供了一些避免锁定的通用练习。

有些应用使用控制表产生后续的序列号。一个典型的例子是在一个排序表中的后续序号。将有一个控制表存储所产生的最后一个序号。任何需要产生新序号的程序将从此表中选择所产生的最后一个排序号，然后递增它。在最后一个序号被递增后，它被更新回控制表中。

在一个多用户环境下，有许多会话试图进入数据库中的一个序列中，此时控制表会成为一个瓶颈。考虑一种情况，其中一个会话从控制表中选择了最后一个序号，在更新了递增的序号以后，由于某些原因在提交以前挂起；整个应用会因为在被挂起的会话后面形成的用户排队而停止。当一个会话把递增的序号更新到控制表，然后在提交以前执行一些辅助的处理时会发生由此类序号引起的另一个瓶颈。控制表记录被锁定的时间将是处理持续的时间。从控制表中捕获下一个序列号以后的辅助处理时间将极大地限制系统的吞吐量。

对于这些问题，有几种可能的解决方案：

- 减少把最后一个序号更新到控制表中和提交之间的时间；也就是说，一获得下一个序列号，就立刻把它提交给数据库，这样使其他会话能够对记录进行存取。
- 考虑使用序列。优点有很多。序列的使用极大地简化了应用代码。按照以前的逻辑，需要执行一个选择和一个更新。然而，在从序列中执行了选择以后，序列号自动递增，这样就有了输入/输出的有利条件。锁定瓶颈被完全消除，因为 Oracle在内部自动地为你递增了序列号。缺点是产生的最后一个号码在使用后不能被回滚，因此会引起空洞。

批处理作业固定地从一张正在被处理的表中选择大量的记录，在这些记录上执行一些处理，然后用新值更新它们。在程序执行处理以前锁定这些记录非常重要，因为当另一个会话到来并且更新记录时，批处理作业可能会从该表中选择过时的数值。因此，当批处理作业必须在处理结束时把结果更新回该表中时，它会发现另一个会话已经更新了该记录。要预防此类事情的发生，在选择记录时预先锁定它们。为此编写的 PL/SQL 伪代码如清单 25-7 所示。

清单25-7 批处理伪代码

```
Begin
  Declare c_emp for
    Select * from emp
    <Where Clause >
  For Update;

  v_emp c_emp%RowType;

  Begin
  Open c_emp;
  Loop;
  Begin

    Fetch c_emp into v_emp;

    --Do some processing --
    Update c_emp
    Set sal = X,
        incr = Y
    Where emp_id = v_emp.emp_id;
    End;
  End Loop;
  Commit;
  Close c_emp;
  End ;
End ;
```

在此代码中，要被处理的整个记录集被锁定；在处理以前需要对记录进行锁定以便当处理这些记录时，另一个会话不能更新它们。此代码的缺陷是即使在代码得到处理后，它们仍然被锁定，直到所有的记录被处理完。假如处理时间很长的话，这些记录被锁定的这段时间会阻止别的会话对它们的并行存取。因为某一给定的时刻或许需要这些记录，所以此锁定方案可能会成为性能的瓶颈。

用来克服上述问题的替换代码在清单 25-8 中给出。

清单25-8 用于取得最大并行度的批处理伪代码

```
Begin
  Declare c_emp for
    Select rowid from emp
    <Where Clause >;

  v_emp c_emp%RowType;
  r_emp emp%RowType;

  Begin
  Open c_emp;
  Loop;
  Begin

    Fetch c_emp into v_emp;

    Select *
    Into r_emp
    From emp
    Where rowid = v_emp.rowid;
    <Where Clause >
    For Update;
    --Do some processing --
    Update emp
```

```

        Set sal = X,
            incr = Y
        Where rowid = v_emp.rowid
        <Where Clause >;

    Commit;

End;
End Loop;

Close c_emp;
End ;

End ;

```

注意到在清单 25-8 中，外层游标仅选择该表中所有的行。外层游标把值反馈到内层选择中，内层选择使用 for update 子句每次仅锁定一行。清单 25-8 中的提交现在被移动到循环里面；因此，在修改了每一行后，进行提交，这样其他用户能够实现对表的最大并行存取。你或许想在清单 25-7 中的脚本的循环内部移动提交。不幸的是，假如在 Oracle 6.0 中完成此项工作的话，游标会关闭，造成 PL/SQL 循环的中断。只要游标被 open 子句打开，那么表中所有相关的行被锁定。当进行提交时，游标失去它的环境并且显示如下消息：

```
ORA-01002: fetch out of sequence
```

因此按前面显示的式样编写代码是必要的。

当使用含有行级触发器的表时要注意。例如，一张表可以有一个行级触发器，这样当该表的一条记录被更新时，该表中的触发器会使其他的表也得到更新。这会由于一条 update 语句而在系统中造成大量的锁。

用户培训有助于避免锁。有很多次我注意到那些认为他们的进程由于某些原因而挂起的用户为了复位该进程而关掉他们的终端（尤其是在客户 / 服务器应用中）。另一类用户是在让他们的终端处于注册状态时便去吃午饭或离开工作桌。这会锁住系统中的一些表并造成其他用户因为此用户而等待。用户需要明白的是关闭终端不能复位进程，该进程在服务器端仍然处于活动状态。类似地，用户应被要求将非活动的终端退出登录。这有助于增加并行性以及使用户不被锁住。

25.5 锁定与分布式数据库

当一个事务企图同时更新两个分布式数据库时，分布式数据库必须解决固有的更新问题，这被称为两步提交（2PC），可以用如下的 SQL 语句实现：

```

APPLY UPDATE A (DB1)
APPLY UPDATE B (DB2)
IF A = OK AND B = OK
    COMMIT A
    < = At this point deadly exposure occurs.
    COMMIT B
ELSE
    ROLLBACK A
    ROLLBACK B

```

这里，在成功地执行 UPDATE 语句之后，系统将把 COMMIT 语句发布到 A 与 B。当一个失败发生在 A 的 COMMIT 以后和 B 的 COMMIT 以前时会发生一个曝光点。由于没有自动恢复，所以它会造成完整性的重大损失。解决此问题的唯一办法是中断该事务并把更新人工回滚到 A 与 B。

如下的技术能够用来释放数据库中的锁：

- 使用命令删除持有 ALTER SYSTEM KILL SESSION 'sid ,serial#' 锁的会话;
- 请求锁持有者提交或回滚。
- 使用 kill -9 <UserID> 在 UNIX 提示符下删除用户标识。
- 在一个两步提交事务中, 使用 ROLLBACK FORCE 或 COMMIT FORCE。
- 从服务器管理器菜单模式或企业管理器中删除用户会话。

详细锁定信息脚本

清单25-9是一个详细锁定信息脚本, 该脚本提供了有关当前在数据库中持有的锁的完整译码信息。

清单25-9 检索数据库锁的译码信息

```

set lines 200
set pagesize 66
break on Kill on sid on username on terminal
column Kill heading 'Kill String' format a13
column res heading 'Resource Type' format 999
column id1 format 9999990
column id2 format 9999990
column locking heading 'Lock Held/Lock Requested' format a40
column lmode heading 'Lock Held' format a20
column request heading 'Lock Requested' format a20
column serial# format 99999
column username format a10 heading "Username"
column terminal heading Term format a6
column tab format a30 heading "Table Name"
column owner format a9
column LAddr heading "ID1 - ID2" format a18
column Lockt heading "Lock Type" format a40
column command format a25
column sid format 990
rem      L.SID||', '||S.SERIAL# Kill,
select
nvl(S.USERNAME,'Internal') username,
      L.SID,
      nvl(S.TERMINAL,'None') terminal,
      decode(command,
0,'None',decode(l.id2,0,U1.NAME||'. '||substr(T1.NAME,1,20),'None')) tab,
decode(command,
0,'BACKGROUND',
1,'Create Table',
2,'INSERT',
3,'SELECT',
4,'CREATE CLUSTER',
5,'ALTER CLUSTER',
6,'UPDATE',
7,'DELETE',
8,'DROP',
9,'CREATE INDEX',
10,'DROP INDEX',
11,'ALTER INDEX',
12,'DROP TABLE',
13,'--',
14,'--',
15,'ALTER TABLE',
16,'--',
17,'GRANT',
18,'REVOKE',

```

```
19, 'CREATE SYNONYM',
20, 'DROP SYNONYM',
21, 'CREATE VIEW',
22, 'DROP VIEW',
23, '- ',
24, '- ',
25, '- ',
26, 'LOCK TABLE',
27, 'NO OPERATION',
28, 'RENAME',
29, 'COMMENT',
30, 'AUDIT',
31, 'NOAUDIT',
32, 'CREATE EXTERNAL DATABASE',
33, 'DROP EXTERNAL DATABASE',
34, 'CREATE DATABASE',
35, 'ALTER DATABASE',
36, 'CREATE ROLLBACK SEGMENT',
37, 'ALTER ROLLBACK SEGMENT',
38, 'DROP ROLLBACK SEGMENT',
39, 'CREATE TABLESPACE',
40, 'ALTER TABLESPACE',
41, 'DROP TABLESPACE',
42, 'ALTER SESSION',
43, 'ALTER USER',
44, 'COMMIT',
45, 'ROLLBACK',
46, 'SAVEPOINT',
47, 'PL/SQL EXECUTE',
48, 'SET TRANSACTION',
49, 'ALTER SYSTEM SWITCH LOG',
50, 'EXPLAIN',
51, 'CREATE USER',
52, 'CREATE ROLE',
53, 'DROP USER',
54, 'DROP ROLE',
55, 'SET ROLE',
56, 'CREATE SCHEMA',
57, 'CREATE CONTROL FILE',
58, 'ALTER TRACING',
59, 'CREATE TRIGGER',
60, 'ALTER TRIGGER',
61, 'DROP TRIGGER',
62, 'ANALYZE TABLE',
63, 'ANALYZE INDEX',
64, 'ANALYZE CLUSTER',
65, 'CREATE PROFILE',
66, 'DROP PROFILE',
67, 'ALTER PROFILE',
68, 'DROP PROCEDURE',
70, 'ALTER RESOURCE COST',
71, 'CREATE SNAPSHOT LOG',
72, 'ALTER SNAPSHOT LOG',
73, 'DROP SNAPSHOT LOG',
74, 'CREATE SNAPSHOT',
75, 'ALTER SNAPSHOT',
76, 'DROP SNAPSHOT',
84, '- ',
85, 'TRUNCATE TABLE',
86, 'TRUNCATE CLUSTER',
87, '- ',
88, 'ALTER VIEW',
89, '- ',
90, '- '.
```

```

91,'CREATE FUNCTION',
92,'ALTER FUNCTION',
93,'DROP FUNCTION',
94,'CREATE PACKAGE',
95,'ALTER PACKAGE',
96,'DROP PACKAGE',
97,'CREATE PACKAGE BODY',
98,'ALTER PACKAGE BODY',
99,'DROP PACKAGE BODY',
command||' - ???') COMMAND,
    decode(L.LMODE,1,'No Lock',
        2,'Row Share',
        3,'Row Exclusive',
        4,'Share',
        5,'Share Row Exclusive',
        6,'Exclusive','NONE') lmode,
    decode(L.REQUEST,1,'No Lock',
        2,'Row Share',
        3,'Row Exclusive',
        4,'Share',
        5,'Share Row Exclusive',
        6,'Exclusive','NONE') request,
l.id1||' - '||l.id2 Laddr,
l.type||' - '||
decode(l.type,
'BL','Buffer hash table instance lock',
'CF','Cross-instance function invocation instance lock',
'CI','Control file schema global enqueue lock',
'CS','Control file schema global enqueue lock',
'DF','Data file instance lock',
'DM','Mount/startup db primary/secondary instance lock',
'DR','Distributed recovery process lock',
'DX','Distributed transaction entry lock',
'FI','SGA open-file information lock',
'FS','File set lock',
'IR','Instance recovery serialization global enqueue lock',
'IV','Library cache invalidation instance lock',
'MB','Master buffer hash table instance lock',
'MM','Mount definition global enqueue lock',
'MR','Media recovery lock',
'RE','USE_ROW_ENQUEUE enforcement lock',
'RT','Redo thread global enqueue lock',
'RW','Row wait enqueue lock',
'SC','System commit number instance lock',
'SH','System commit number high water mark enqueue lock',
'SN','Sequence number instance lock',
'SQ','Sequence number enqueue lock',
'ST','Space transaction enqueue lock',
'SV','Sequence number value lock',
'TA','Generic enqueue lock',
'TD','DDL enqueue lock',
'TE','Extend-segment enqueue lock',
'TM','DML enqueue lock',
'TT','Temporary table enqueue lock',
'TX','Transaction enqueue lock',
'UL','User supplied lock',
'UN','User name lock',
'WL','Being-written redo log instance lock',
'WS','Write-atomic-log-switch global enqueue lock',
'TS',decode(l.id2,0,'Temporary segment enqueue lock (ID2=0)',
    'New block allocation enqueue lock (ID2=1)'),
'LA','Library cache lock instance lock (A=namespace)',
'LB','Library cache lock instance lock (B=namespace)',
'LC','Library cache lock instance lock (C=namespace)',

```

```
'LD','Library cache lock instance lock (D=namespace)',
'LE','Library cache lock instance lock (E=namespace)',
'LF','Library cache lock instance lock (F=namespace)',
'LG','Library cache lock instance lock (G=namespace)',
'LH','Library cache lock instance lock (H=namespace)',
'LI','Library cache lock instance lock (I=namespace)',
'LJ','Library cache lock instance lock (J=namespace)',
'LK','Library cache lock instance lock (K=namespace)',
'LL','Library cache lock instance lock (L=namespace)',
'LM','Library cache lock instance lock (M=namespace)',
'LN','Library cache lock instance lock (N=namespace)',
'LO','Library cache lock instance lock (O=namespace)',
'LP','Library cache lock instance lock (P=namespace)',
'LS','Log start/log switch enqueue lock',
'PA','Library cache pin instance lock (A=namespace)',
'PB','Library cache pin instance lock (B=namespace)',
'PC','Library cache pin instance lock (C=namespace)',
'PD','Library cache pin instance lock (D=namespace)',
'PE','Library cache pin instance lock (E=namespace)',
'PF','Library cache pin instance lock (F=namespace)',
'PG','Library cache pin instance lock (G=namespace)',
'PH','Library cache pin instance lock (H=namespace)',
'PI','Library cache pin instance lock (I=namespace)',
'PJ','Library cache pin instance lock (J=namespace)',
'PK','Library cache pin instance lock (K=namespace)',
'PL','Library cache pin instance lock (L=namespace)',
'PM','Library cache pin instance lock (M=namespace)',
'PN','Library cache pin instance lock (N=namespace)',
'PO','Library cache pin instance lock (O=namespace)',
'PP','Library cache pin instance lock (P=namespace)',
'PQ','Library cache pin instance lock (Q=namespace)',
'PR','Library cache pin instance lock (R=namespace)',
'PS','Library cache pin instance lock (S=namespace)',
'PT','Library cache pin instance lock (T=namespace)',
'PU','Library cache pin instance lock (U=namespace)',
'PV','Library cache pin instance lock (V=namespace)',
'PW','Library cache pin instance lock (W=namespace)',
'PX','Library cache pin instance lock (X=namespace)',
'PY','Library cache pin instance lock (Y=namespace)',
'PZ','Library cache pin instance lock (Z=namespace)',
'QA','Row cache instance lock (A=cache)',
'QB','Row cache instance lock (B=cache)',
'QC','Row cache instance lock (C=cache)',
'QD','Row cache instance lock (D=cache)',
'QE','Row cache instance lock (E=cache)',
'QF','Row cache instance lock (F=cache)',
'QG','Row cache instance lock (G=cache)',
'QH','Row cache instance lock (H=cache)',
'QI','Row cache instance lock (I=cache)',
'QJ','Row cache instance lock (J=cache)',
'QK','Row cache instance lock (K=cache)',
'QL','Row cache instance lock (L=cache)',
'QM','Row cache instance lock (M=cache)',
'QN','Row cache instance lock (N=cache)',
'QO','Row cache instance lock (O=cache)',
'QP','Row cache instance lock (P=cache)',
'QQ','Row cache instance lock (Q=cache)',
'QR','Row cache instance lock (R=cache)',
'QS','Row cache instance lock (S=cache)',
'QT','Row cache instance lock (T=cache)',
'QU','Row cache instance lock (U=cache)',
'QV','Row cache instance lock (V=cache)',
'QW','Row cache instance lock (W=cache)',
'QX','Row cache instance lock (X=cache)',
```

```
'QY','Row cache instance lock (Y=cache)',  
'QZ','Row cache instance lock (Z=cache)', '????') Lockt  
from   V$LOCK L,  
        V$SESSION S,  
        SYS.USER$ U1,  
        SYS.OBJ$ T1  
where  L.SID = S.SID  
and    T1.OBJ# = decode(L.ID2,0,L.ID1,1)  
and    U1.USER# = T1.OWNER#  
and    S.TYPE != 'BACKGROUND'  
order by 1,2,5  
/
```

25.6 使用闩实现锁

Oracle使用在以前的小节中提到过的各种技术实现锁。使用闩是其中的一种方法。闩与锁不同，它在内部获得并由Oracle释放。不能通过发布一条语句明确地获得闩。一般，闩用于控制对共享代码路径的存取。闩是一种能够被快速获得或释放的锁。闩用来防止一个以上的进程在一个特定的时间里执行相同的代码段。

25.6.1 闩的功能

闩用于控制对共享结构的存取。可以在操作系统层上使用信号量实现闩。闩是在非常短时间内持有的锁。

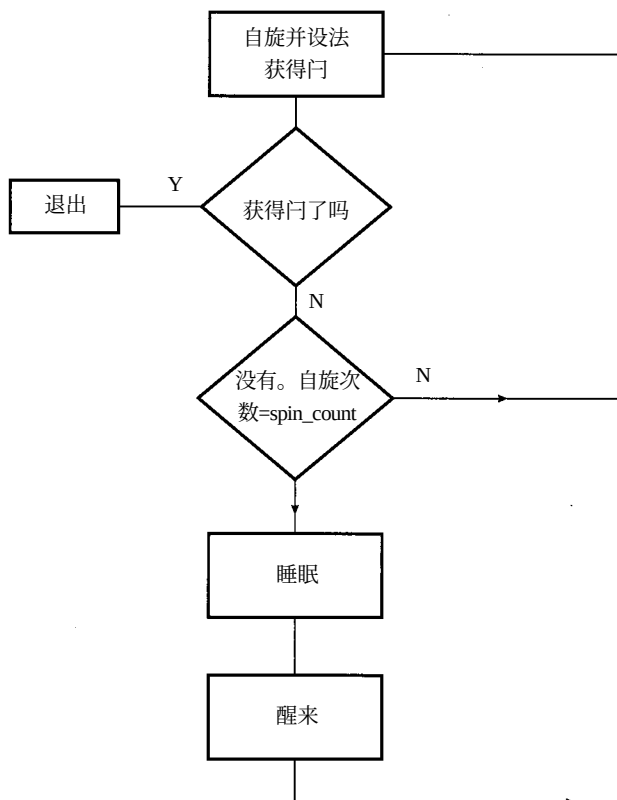


图25-1 在一个闩上的进程自旋

一个进程在能够对受门保护的共享结构存取以前，它首先必须获得该门。此门当前可能是空闲的（也就是说，没有其他的进程正在持有该门），在这种情况下此进程立刻能得到该门。此进程将在需要门的时候持有它，然后在不需要时便释放它。这是最简单和典型的情况。

考虑一种情况，在该情况下一个进程已经获得了某个门。另一个进程根据它正试图获得的门的类型，会获取有两个选项的门。有两种与门有关的类型：

■ 立刻。

■ 可以等待。

假如一个进程试图在立刻模式下获得门，而该门已经被另外一个进程所持有，如果该门不能立刻可用的话，那么该进程就不会为获得该门而等待。它将继续执行另一个操作。

在可以等待模式下，假如一个进程在第一次尝试中没有获得该门，那么它会等待并且再尝试一次。如果系统有多个 CPU，此未成功的进程将开始围绕该门自旋并设法获得它。该进程围绕该门自旋的次数在 init.ora 文件的参数 spin_count 中定义。每一次自旋，它都设法获得该门；如果它不能获得该门，那么它就再次自旋并继续下去，直到自旋数达到 spin_count 所规定的数值。然后该进程转入睡眠状态，持续一段指定长度的时间，然后再次醒来，按顺序重复以前的步骤，如图 25-1 所示。

25.6.2 分析与门相关的视图

本节讨论数据字典视图，该视图能够用来获得系统中有关门的更多信息。在这里仅讨论这些视图中的重要列。V\$latch 与 V\$latchholder 是两个与门有关的非常重要的视图：

■ V\$latch —— 本视图包含与系统中各种门的性能有关的所有重要的统计数字。表 25-5 给出了对视图 V\$latch 的重要列的描述。

■ V\$latchholder —— 假如系统当前存在门争用问题，本视图能够用来确定哪一个会话当前拥有该门。

表25-5 V\$latch

列 名	描 述
immediate_gets	立刻获得的门的成功请求次数（仅限立刻模式）
immediate_misses	没有立刻获得的门的未成功请求次数（仅限立刻模式）
gets	甘愿等待而获得的门的成功请求次数
misses	甘愿等待而获得的门的未成功请求次数
sleeps	一个甘愿等待门的进程请求门的次数并且当该进程没有得到门时，必须转入睡眠状态
spin_gets	不必睡眠，而仅仅自旋就获得门的次数
sleep1~10	每当进程睡眠时，该列中的值就递增——例如，假如一个进程在获得门以前不得不睡眠了3次的话，那么 sleep3 列将增1

25.6.3 门竞争约束

假如系统正面临门竞争问题，那么清单 25-10 和清单 25-11 中的脚本能够用来发现哪一个门受到严重地打击。所有的脚本查询 V\$latch 表以便找到门的细节。下面显示的是对 V\$latch 的描述：

ADDR	RAW(4)
LATCH#	NUMBER
LEVEL#	NUMBER
NAME	VARCHAR2(64)

GETS	NUMBER
MISSES	NUMBER
SLEEPS	NUMBER
IMMEDIATE_GETS	NUMBER
IMMEDIATE_MISSES	NUMBER
WAITERS_WOKEN	NUMBER
WAITS_HOLDING_LATCH	NUMBER
SPIN_GETS	NUMBER
SLEEP1	NUMBER
SLEEP2	NUMBER
SLEEP3	NUMBER
SLEEP4	NUMBER
SLEEP5	NUMBER
SLEEP6	NUMBER
SLEEP7	NUMBER
SLEEP8	NUMBER
SLEEP9	NUMBER
SLEEP10	NUMBER
SLEEP11	NUMBER

下面的脚本将在SQL*Plus中运行。

清单25-10 用于鉴别系统中的问题的脚本

```
col name heading "Name" format a20
col pid heading "HSid" format a3
col gets heading "Gets" format 999999990
col misses heading "Miss" format 99990
col im_gets heading "ImG" format 99999990
col im_misses heading "ImM" format 99990
col sleeps heading "Sleeps" format 99990

select n.name name, h.pid pid , l.gets gets , l.misses misses,
l.immediate_gets im_gets, l.immediate_misses im_misses, l.sleeps sleeps
from v$latchname n , v$latchholder h , v$latch l
where l.latch# = n.latch#
and l.addr = h.laddr(+)
```

上述SQL语句的输出结果如下所示：

Name	HSid	Gets	Miss	ImG	ImM	Sleeps
cached attr list		0	0	0	0	0
modify parameter values		931	0	0	0	0
messages		736117	44	0	0	80
enqueue hash chains		1257100	145	0	0	163
trace latch		0	0	0	0	0
cache buffers lru chains	197351761	2092	66288289	20919	4020	
cache buffer handles		2348	0	0	0	0
multiblock read objects		2139864	700	1	0	900
cache protection latch		0	0	0	0	0
shared pool		1261081	207	0	0	208
library cache		7867103	11803	376	0	11854
redo allocation		1306657	501	0	0	760
redo copy		34	0	463290	71	47

在此输出中，HSID列指示了当前持有问题的会话的标识。假如 misses与 gets之比超过1%，那么该问题需要引起注意。

使用V\$session wait视图是另外一种用于检查一个长时间挂起的会话是否正在等待一个问题的方法，可以使用清单25-11中的查询。

清单25-11 使用V\$session wait视图来鉴别问题竞争

```
select event, p1text,p1 ,p2text,p2 ,seq#,wait_time,state
```

```
from v$session_wait
where   sid = '&&1'
and event = 'latch free';
```

上述SQL语句的输出结果如下所示：

EVENT	P1TEXT	P1	P2TEXT	P2	SEQ#	WAIT_TIME	STATE
latch free	address	3	number	11	1181	0	WAITING

假如wait_time列的值为零，表明该会话当前正等待。

P2列显示了锁的数量。可以使用V\$latchname得到锁的名称：

```
Select name
From v$latchname
Where latch# = 11;
cache buffers chains
```

清单25-12是一个报告生成脚本，该脚本列举了用于确定一个数据库实例是否正经历锁竞争的关键信息。锁竞争率应当小于或等于1%。

清单25-12 列举用于确定锁竞争的信息

```
tttitle -
      center  'Latch Contention Report'  skip 3

col name form A25
col gets form 999,999,999
col misses form 999.99
col spins form 999.99
col igets form 999,999,999
col imisses form 999.99

select name,
       gets,
       misses*100/decode(gets,0,1,gets) misses,
       spin_gets*100/decode(misses,0,1,misses) spins,
       immediate_gets igets,
       immediate_misses*100/decode(immediate_gets,0,1,immediate_gets) imisses
from v$latch
order by gets + immediate_gets
/
```

上述SQL语句的输出结果如下所示：

Latch Contention Report					
NAME	GETS	MISSSES	SPINS	IGETS	IMISSES
cached attr list	0	.00	.00	0	.00
trace latch	0	.00	.00	0	.00
cache protection latch	0	.00	.00	0	.00
KCL freelist latch	0	.00	.00	0	.00
redo copy	0	.00	.00	0	.00
archive control	0	.00	.00	0	.00
KCL name table latch	0	.00	.00	0	.00
lock element parent latch	0	.00	.00	0	.00
loader state object freelist	0	.00	.00	0	.00
process queue	0	.00	.00	0	.00
error message lists	0	.00	.00	0	.00
query server freelists	0	.00	.00	0	.00
query server process	0	.00	.00	0	.00
virtual circuits	0	.00	.00	0	.00

virtual circuit queues	0	.00	.00	0	.00
virtual circuit buffers	0	.00	.00	0	.00
global tx hash mapping	0	.00	.00	0	.00
device information	0	.00	.00	0	.00
parallel query alloc buffer	0	.00	.00	0	.00
parallel query stats	0	.00	.00	0	.00
process queue reference	0	.00	.00	0	.00
global transaction	0	.00	.00	0	.00
global tx free list	0	.00	.00	0	.00
cost function	0	.00	.00	0	.00
instance latch	0	.00	.00	0	.00
NLS data objects	1	.00	.00	0	.00
cache buffer handles	42	.00	.00	0	.00
latch wait list	70	.00	.00	0	.00
multiblock read objects	124	.00	.00	0	.00
library cache load lock	326	.00	.00	0	.00
sort extent pool	867	.00	.00	0	.00
ktm global data	867	.00	.00	0	.00
process allocation	680	.00	.00	680	.00
sequence cache	2,128	.00	.00	0	.00
user lock	2,676	.00	.00	0	.00
session switching	4,120	.00	.00	0	.00
modify parameter values	5,005	.00	.00	0	.00
cache buffers lru chain	41,770	.00	.00	31,613	.00
dml lock allocation	116,700	.00	.00	0	.00
transaction allocation	177,154	.00	.00	0	.00
undo global data	206,727	.00	.00	0	.00
enqueue hash chains	271,775	.00	.00	0	.00
messages	592,041	.00	.00	0	.00
shared pool	860,906	.00	.00	0	.00
enqueues	1,151,391	.00	.00	0	.00
list of block allocation	1,558,054	.00	.00	0	.00
session allocation	1,688,085	.00	.00	0	.00
system commit number	1,859,249	.00	.00	0	.00
redo allocation	2,009,258	.01	.00	0	.00
row cache objects	2,501,694	.00	.00	42	.00
session idle bit	4,167,221	.00	.00	0	.00
library cache	13,734,618	.00	.00	41	.00
cache buffers chains	43,694,786	.00	.00	6,832,121	.00

清单25-13是一个给出门的各种睡眠率的脚本。

清单25-13 检索门睡眠率

```
col name form A18 trunc

col gets form 999,999,990
col miss form 90.9
col cspins form A6 heading 'spin|s106'
col csleep1 form A5 heading 's101|s107'
col csleep2 form A5 heading 's102|s108'
col csleep3 form A5 heading 's103|s109'
col csleep4 form A5 heading 's104|s110'
col csleep5 form A5 heading 's105|s111'
col Interval form A12
set recsep off

select a.name,
       a.gets gets,
       a.misses*100/decode(a.gets,0,1,a.gets) miss,
       to_char(a.spin_gets*100/decode(a.misses,0,1,a.misses),'990.9')||
       to_char(a.sleep6*100/decode(a.misses,0,1,a.misses),'90.9') cspins,
       to_char(a.sleep1*100/decode(a.misses,0,1,a.misses),'90.9')||
       to_char(a.sleep7*100/decode(a.misses,0,1,a.misses),'90.9') csleep1,
```

```

to_char(a.sleep2*100/decode(a.misses,0,1,a.misses),'90.9')||
to_char(a.sleep8*100/decode(a.misses,0,1,a.misses),'90.9') csleep2,
to_char(a.sleep3*100/decode(a.misses,0,1,a.misses),'90.9')||
to_char(a.sleep9*100/decode(a.misses,0,1,a.misses),'90.9') csleep3,
to_char(a.sleep4*100/decode(a.misses,0,1,a.misses),'90.9')||
to_char(a.sleep10*100/decode(a.misses,0,1,a.misses),'90.9') csleep4,
to_char(a.sleep5*100/decode(a.misses,0,1,a.misses),'90.9')||
to_char(a.sleep11*100/decode(a.misses,0,1,a.misses),'90.9') csleep5
from v$latch a
where a.misses <> 0
order by 2 desc
/

```

The output from the SQL statement above is as follows:

```

spin  sl01  sl02  sl03  sl04  sl05
NAME                                GETS  MISS  sl06   sl07  sl08  sl09  sl10  sl11
-----
cache buffers chain  43,696,090  0.0    0.0 ##### 0.0  0.0  0.0  0.0
                        0.0    0.0  0.0  0.0  0.0  0.0
library cache        13,738,029  0.0    0.0  97.1  0.0  2.9  0.0  0.0
                        0.0    0.0  0.0  0.0  0.0  0.0
session idle bit     4,167,650  0.0    0.0 ##### 0.0  0.0  0.0  0.0
                        0.0    0.0  0.0  0.0  0.0  0.0
row cache objects    2,502,963  0.0    0.0 ##### 0.0  0.0  0.0  0.0
                        0.0    0.0  0.0  0.0  0.0  0.0
redo allocation      2,010,038  0.0    0.0 ##### 0.0  0.0  0.0  0.0
                        0.0    0.0  0.0  0.0  0.0  0.0
system commit numb   1,859,423  0.0    0.0 ##### 0.0  0.0  0.0  0.0
                        0.0    0.0  0.0  0.0  0.0  0.0
session allocation    1,688,269  0.0    0.0 ##### 0.0  0.0  0.0  0.0
                        0.0    0.0  0.0  0.0  0.0  0.0
enqueuees            1,152,139  0.0    0.0 ##### 0.0  0.0  0.0  0.0
                        0.0    0.0  0.0  0.0  0.0  0.0
messages              595,209  0.0    0.0 ##### 0.0  0.0  0.0  0.0
0.0    0.0  0.0  0.0  0.0  0.0
enqueue hash chain    272,103  0.0    0.0  0.0  0.0 ##### 0.0  0.0
0.0    0.0  0.0  0.0  0.0  0.0
cache buffers lru     41,778  0.0    0.0 ##### 0.0  0.0  0.0  0.0
                        0.0    0.0  0.0  0.0  0.0  0.0

```

调整一些重要的门

在清单25-14中显示的所有门中，可以采用几个重要的和正确的步骤来调整它们。本节中将要讨论的门有 cache buffers lru chain、redo allocation、redo copy 和 library cache。

清单25-14 列举所有的门

LATCH#	NAME
0	latch wait list
1	process allocation
2	session allocation
3	session switching
4	session idle bit
5	cached attr list
6	modify parameter values
7	messages
8	enqueuees
9	enqueue hash chains
10	trace latch
11	cache buffers chains
12	cache buffer handles
13	multiblock read objects
14	cache protection latch


```
15 cache buffers lru chain
16 system commit number
17 archive control
18 redo allocation
19 redo copy
20 KCL freelist latch
21 KCL name table latch
22 instance latch
23 lock element parent latch
24 loader state object freelist
25 dml lock allocation
26 list of block allocation
27 transaction allocation
28 sort extent pool
29 undo global data
30 ktm global data
31 sequence cache
32 row cache objects
33 cost function
34 user lock
35 global tx free list
36 global transaction
37 global tx hash mapping
38 shared pool
39 library cache
40 library cache load lock
41 virtual circuit buffers
42 virtual circuit queues
43 virtual circuits
44 NLS data objects
45 query server process
46 query server freelists
47 error message lists
48 process queue
49 process queue reference
50 parallel query stats
51 parallel query alloc buffer
52 device information
```

- 高速缓存缓冲区最近最少使用链 高速缓存缓冲区最近最少使用链负责保护指向高速缓存中的数据库块缓冲区的存取路径。由 init.ora文件的db_block_buffers参数定义在SGA中的高速缓存的容量并且包含了已保存的从数据文件中读出的数据拷贝。当进程需要读数据时，由于某些进程读入的数据而使此缓冲区中存在数据，这使得读操作变成非常有效率。

高速缓存分为两个列表：脏列表与LRU列表。脏列表（dirty list）包含那些已经被修改但还没有被写入到磁盘中的缓冲区。LRU列表由被钉住的缓冲区、还没有被移动到磁盘中的脏缓冲区和空闲缓冲区组成。被钉住的缓冲区（pinned buffer）是指那些当前被其他进程存取的缓冲区。脏缓冲区（dirty buffer）包含那些没有被写入到磁盘中的缓冲区，这些缓冲区随后移动到脏列表中。空闲缓冲区（free buffer）是指那些可供使用的缓冲区。

当一个进程需要从磁盘中读取已经不在高速缓存中的数据时，它需要一个空闲缓冲区用于读取新数据。它搜索LRU列表以寻找空闲缓冲区。假如有过多的对高速缓存中的空闲缓冲区的需求，那么对LRU列表的大量存取会引发对高速缓存缓冲区LRU链的争用。

对门的争用可以通过使用 init.ora参数db_block_1ru_latches来减到最小，此参数可以在Oracle 8中使用。通过增加db_block_1ru_latches参数的值，可将对门的争用减到最小。此参数

的最大值是CPU数量的两倍。

对此争用的基本原因是大量地请求空闲缓冲区。你可以优化此 SQL 语句以把对空闲缓冲区的高请求减到最小或增加 db_block_buffer参数的值以增加系统中可用的空闲缓冲区的数量。

注意 注意SGA必须装入实内存的一个连续块中，所以说，如果增加缓冲区高速缓存的话，你必须确保系统中有足够的连续可用内存。

- **重做分配和重做拷贝** 重做分配（redo allocation）和重做拷贝（redo copy）控制对重做日志缓冲区的写访问。当某个进程需要写重做日志缓冲区时，该进程将获得这两种中的一个。假如写入到重做日志缓冲区的重做日志信息的容量小于 log_small_entry_max_size参数的值，那么该进程将使用重做分配。假如信息的容量大于该值，那么使用重做拷贝拷贝该进程。

一种用于检查重做日志缓冲区中是否存在争用的快速方法是检查是否存在与写重做日志缓冲区有关的任何等待。这可以通过使用系统视图 V\$sysstat来实现：

```
Select name, value
From v$sysstat
Where name = 'redo log space requests';
```

上述SQL语句的输出结果如下所示：

Name	Value
redo log space requests	12

假如等待数量太高的话，必须增加重做日志缓冲区的容量。

- **争用重做分配** 在一个多CPU系统中通过强迫进程使用重做拷贝能够减少对重做分配的争用。因为可以有多个重做拷贝，所以拷贝会更有效地被执行。重做拷贝的数量由init.ora参数log_simultaneous_copies定义。系统中可用口的最大数量是 CPU数量的两倍。对于一个单 CPU系统来说，该值为0并且将使用重做分配。假如存在一个对重做分配的争用，那么可以减小 log_small_entry_max_size参数的当前值以便使用重做拷贝。
- **争用重做拷贝** 假如系统正面临对重做拷贝的争用，可以通过增加 log_small_entry_max_size参数的值（以便使用重做分配）或者增加 log_simultaneous_copies参数的值（以便增加可用的重做拷贝的数量）来减少这种争用。

可以增加init.ora文件参数log_entry_prebuild_threshold的值以便被写入重做日志缓冲区中的数据被分组或写出。通过增加该参数，可以对许多写操作进行分组以便它们能够用一个操作写出，这样便减少了对这些口的请求，从而减少了争用。

- **库高速缓存** 该口主要与控制对库高速缓存的存取有关。库高速缓存包含共享 SQL区、专用的SQL区、PL/SQL过程包以及其他的控制结构。共享 SQL区包含在多个会话之间共享的SQL代码。通过增加对这些 SQL代码的共享，可以避免对此口的争用。

当存在大量的对库高速缓存中空间的请求时会发生对此口的争用。系统中大量的语法分析以及由于进程之间的低共享而产生的大量对打开一个新游标的请求是常见的引发对此口争用的原因。

通过使用可以被多个会话共享的代码可以避免对此口的争用。例如， RDBMS以不同的方

式处理下面的两条SQL语句：

```
select name from employee where emp_id = 100;  
Select name from employee where emp_id = 100;
```

尽管两条SQL语句看起来相同，但是这两条语句的哈希值不同。上面那条SQL语句的“select”中的“s”是小写字母，而下面那条SQL语句中用的是大写的S。因此，当发出这两条语句时，RDBMS将把它们作为不同的语句分别进行语法分析，从而引起门的负载。通过应用编码标准可以避免此类情况，在编码标准中实现缩进与格式标准，以便所编写的每条SQL语句尽可能有相同的散列值。注意，就连放入更多的空格都会使一条select语句被认为是另一条而引起增加语法分析。

在SQL语句中使用绑定变量能够增加对同一子句的共享。例如，考虑两种不使用绑定变量的情况：

```
Select sal from employee where emp_id = 100;  
Select sal from employee where emp_id = 200;
```

现在，对它们使用绑定变量：

```
Select sal from employee where emp_id := emp_id;
```

通过使用绑定变量，在第一种情况下的前两条select语句都能够共享存储在库缓存中的相同的select语句，这样减少了不必要的语法分析和门的负载。使用绑定变量使在共享池中不致于形成同一条select语句的多个拷贝。

可以在共享池中减少语法分析或者钉住诸如过程与包等频繁使用的对象来减少语法分析。钉住这些对象的优点是这些对象决不会被从共享池中清除并且缩减了随后的语法分析时间。可以通过使用如下的查询识别那些频繁使用的对象：

```
Select name ,executions  
From v$db_object_cache  
Where executions > <threshold limit>  
order by 2 desc;
```

可以使用如下语句将这些频繁使用的对象钉在共享池中：

```
dbms_share_pool.keep('object_name','p');
```

要在共享池中检查没有被钉住的对象，执行如下的查询：

```
Select name,type,kept,sharable_mem  
From v$db_object_cache  
Where kept = 'NO'  
Order by sharable_mem desc;
```

共享池的碎片也会引发大量对这些门的需求。已碎片化的共享池意指共享池中可用的净空闲内存可能很大，但是可用的总的连续内存并不大。因此，一个具有非常高的内存需求的对象（例如大型PL/SQL对象）被定位在共享池中会使已分配内存区的许多小字节块被刷新掉。当一个用户再次请求相关的SQL语句时，这些语句必须再次从语法上得到分析。引起碎片的主要原因是大型PL/SQL对象。通过使用绑定变量以及使用上述提及的技术钉住大型PL/SQL对象以增加SQL语句间的共享来避免碎片的形成。

第六部分 Oracle接口和应用工具

第26章 管理员使用的SQL*Plus

本章要点：

系统管理的SQL*Plus

使用SQL*Plus的COPY命令

使用SQL创建SQL

在SQL*Plus中对用户权限的限制

追踪SQL语句

SQL*Plus 8.1版的增强

本章讨论SQL*Plus，它是用于访问Oracle数据库的交互式应用程序，它也是数据库系统管理员最好的朋友和不可缺少的工具。

除了SQL*Plus外，数据库系统管理员还可以在另外两种工具中与数据库交互并使用SQL*Plus命令（但仅仅是SQL*Plus命令的一个子集）：

- 服务器管理器。

- 企业管理器中的工作表工具。

注意 从8.1.3版本开始，所有的服务器管理器命令在SQL*Plus中都是可用的。Oracle正计划放弃服务器管理器，并提供一个一体化的工具完成所有的数据库系统管理和SQL命令（参见26.6节）。

因为SQL*Plus是个很大的主题，不是在一个章节中能完成的，我将讨论主要集中在DBA感兴趣的特性方面，包括EXECUTE、AUTOTRACE和各种各样的新特性，还有少为人知且较少用到的特性（COPY命令、禁止命令）。

26.1 系统管理的SQL*Plus

有两个文件glogin.sql和login.sql用于管理SQL*Plus。glogin.sql文件是全局设置文件，login.sql文件是为个人使用准备的。这两个文件包含每次Oracle用户激活SQL*Plus所执行的SQL*Plus命令和/或SQL语句。首先读和执行glogin.sql文件，然后是用户的login.sql文件。

glogin.sql文件位于\$ORACLE_HOME/sqlplus/admin目录下。Oracle数据库系统管理员可以自定义该文件含有SQL*Plus命令、SQL语句和PL/SQL块，它们在每个SQL*Plus用户开始他或她的SQL*Plus会话时被执行。glogin.sql文件也称作节点资源文件。

注意 在Windows 95/NT 4.0环境下，glogin.sql文件位于%ORACLE_HOME%\PLUSnn目录下，这里的nn代表机器上安装的SQL*Plus版本号。例如，如果安装的是SQL*Plus 3.3版，目录名将是%ORACLE_HOME%\PLUS33。

26.1.1 使用SQL*Plus环境变量

SQL*Plus里使用两个环境变量：

- SQLPATH。
- 编辑器。

SQL*Plus使用环境变量SQLPATH指示login.sql文件的所在目录。换句话说，SQL*Plus将查看SQLPATH中定义的每个目录以找到login.sql，使用本地目录（当你启动SQL*Plus时所在的目录）启动SQL*Plus。

例如，如果你希望SQL*Plus首先在本地目录下查找login.sql文件，然后在你的根目录，最后再查找其他目录，那么如下设置SQLPATH：

```
$ SQLPATH=".:$HOME:<other_directory>"; export SQLPATH      — (Bourne/Korn shell)
$ set SQLPATH=(. $HOME <other_directory>)                  — C shell
```

在Windows 95/NT 4.0环境下，SQLPATH在注册表中定义，缺省值是 \$ORACLE_HOME\DBS（在安装期间设置的）。ORACLE_HOME的值在Windows95中为C:\ORAWIN 95，在NT 4.0中为C:\ORANT（如果你没有将Oracle安装在C：驱动器上，使用你安装Oracle的磁盘驱动器符号代替C：）。

注意 在Windows 95/NT 4.0环境下，login.sql文件位于%ORACLE_HOME%\DBS目录下，这是SQLPATH的缺省值。

在Windows 95/NT 4.0环境下要设置或改变SQLPATH的值，遵照以下步骤：

- 1) 在Start菜单中选择Run。
- 2) 输入regedit.exe/regedit32.exe（分别对应于Windows 95/NT 4.0环境）。
- 3) 点击OK。
- 4) 双击HKEY_LOCAL_MACHINE。
- 5) 双击SOFTWARE。
- 6) 双击Oracle。
- 7) 双击SQLPATH。
- 8) Edit String 对话框出现，在Value Data域输入新的SQLPATH值。
- 9) 点击OK。
- 10) 在Registry菜单，选择Exit。
- 11) 重新启动你的机器，以便让新的值起作用（或者注销并再次登录进入Windows NT 4.0）。

注意 SQLPATH还被SQL*Plus用于指示你在SQL*Plus中运行的SQL脚本的位置。

在glogin.sql或login.sql中还可以设置另外一个环境变量，这个环境变量称为_editor，它定义了你可以用于编辑SQL*Plus命令的编辑器。

要设置编辑器为vi文本编辑器，在glogin.sql或login.sql文件中输入下面的代码行：

```
define _editor=vi
```

注意 在Oracle 8.1.x中，可以在工作过程中更改编辑器，进入Edit | Editor | Define editor并输入“Notepad”（笔记本）或“vi”。

如果你使用任何其他的文本编辑器，将“vi”替换为相应的名称。有关使用不同编辑器的

更多信息，在稍后的26.1.5节中讨论。

26.1.2 激活/访问SQL*Plus

要在操作系统提示符状态下激活 SQL*Plus，请使用下面的命令：

```
$ sqlplus [[-S{ILENT}]] [logon] [start]]!-?
```

当从一个壳脚本中运行 SQL*Plus时，使用-S[ILENT]参数，因为它消除了 SQL*Plus被激活时显示的全部信息，例如 SQL*Plus标志、提示信息和命令行提示符。

[logon]部分需要下面的语法结构：

```
username[/password] [@connect_string]!;/Nolog
```

[start]子句让你能够启动 SQL*Plus并运行一个包含 SQL*Plus命令、SQL语句和/或PL/SQL块的任意组合的命令文件。另外，你可以向这个命令文件传递参数。 start子句要求下面的语法：

```
@file_name[.ext] [arg...]
```

如果你没有输入用户名和/或口令，SQL*Plus提示你输入它们。

在你成功地启动 SQL*Plus后，可以在 SQL*Plus提示符（SQL>）状态下输入三种类型的命令：

- 用于处理数据库对象和操作数据库中存储的数据的 SQL命令/语句。
- 用于处理数据库对象和操作数据库中存储的数据的 PL/SQL（过程化语言/SQL）块。
- 用于设置选项、编辑、存储和提取 SQL命令与PL/SQL块以及格式化查询输出结果的 SQL*Plus命令。

要向SQL*Plus提交一条SQL命令，请在命令的结尾处输入一个分号（；）并且按下回车键。SQL*Plus执行该命令，显示查询的结果并且返回到提示符状态。

要结束一个PL/SQL块，在它自己的行上输入一个小数点即可。要提交一个 PL/SQL块用于执行，在PL/SQL的行上用一条斜线（/）结束它并按下回车键。

26.1.3 编辑SQL命令

如果你在输入一条SQL命令时出现一个错误并且希望改正它，或者如果你希望运行仅仅稍加改动的最后一条命令，你非常幸运！SQL*Plus将最近输入的SQL命令存储在一个缓冲区中，该缓冲区被称为 SQL缓冲区。SQL*Plus提供了一系列命令来提取并编辑存储在该缓冲区中的SQL语句和PL/SQL块。

注意SQL*Plus命令没有存储在该缓冲区中，所以，它们不能够被提取和修改。但是，有一种方法可以在该缓冲区中存储 SQL*Plus命令，这种方法在稍后的26.1.4节中讨论。

表26-1列出用于查看、编辑和运行 SQL*Plus缓冲区内容的命令。

除LIST命令外，所有其他的编辑命令仅影响缓冲区中的一行，该行被称为当前行，当列出SQL命令或PL/SQL块时，当前行用星号标记出来。当运行 LIST命令时，当前行总是缓冲区中的最后一行。还要注意终结一条 SQL命令的分号（；）没有存储在缓冲区中，如清单 26-1所示。

如果你得到一条错误信息，含有错误的行成为当前行，这样你就可以立即编辑该行并纠正错误。清单26-2显示了带有错误的程序清单。

表26-1 用于SQL缓冲区的SQL*Plus命令

命 令	缩 写	作 用
APPEND 文本	A 文本	在一行尾部添加文本
CHANGE 旧文本/新文本	C 旧文本/新文本	使用新文本更改一行中的旧文本
CHANGE /文本	C /文本	从一行中删除文本
CLEAR BUFFER	CL BUFF	删除全部行
DEL	(无)	删除缓冲区中的当前行
INPUT	I	在缓冲区中添加一行或多行
INPUT 文本	I 文本	添加一条文本组成的行
LIST	L	列出SQL*Plus缓冲区中的内容
LIST n	L n 或n	列出第n行
LIST *	L *	列出当前行
LIST m n	L m n	列出第m行到第n行
LIST LAST	L LAST	列出缓冲区中的最后一行

清单26-1 存储在缓冲区中的SQL语句

```
SQL> LIST
1 SELECT empno, ename, deptno, job, sal, comm
2 FROM emp
3* WHERE comm IS NOT NULL
```

清单26-2 带有错误的行成为当前行

```
SQL> SELECT empno, empname, deptno, job, sal, comm
2 FROM emp
3 WHERE comm IS NOT NULL;
SELECT empno, empname, deptno, job, sal, comm
*
ERROR at line 1:
ORA-00904: invalid column name
```

26.1.4 输入并编辑SQL*Plus命令

直接在SQL*Plus提示符下输入的SQL*Plus命令（例如DESCRIBE）不保存在SQL缓冲区中。所以，你不能够提取最后输入的SQL*Plus命令来编辑并/或再次运行它。试图从缓冲区中列出一条SQL*Plus命令的结果如清单26-3所示。

清单26-3 SQL*Plus命令没有被保存在缓冲区中

```
SQL> DESCRIBE EMP
Name Null? Type
-----
EMPNO NOT NULL NUMBER(4)
ENAME VARCHAR2(10)
JOB VARCHAR2(9)
MGR NUMBER(4)
HIREDATE DATE
SAL NUMBER(7,2)
COMM NUMBER(7,2)
DEPTNO NOT NULL NUMBER(2)
SQL> LIST
No lines in SQL buffer.
```

要在缓冲区中存储SQL*Plus命令，请输入不带有文本的INPUT命令并且按下回车键。

SQL*Plus使用一个行号提示你可以输入命令。当你完成命令输入时，按下回车键输入一行空行。清单26-4显示了一个示例。

清单26-4 在缓冲区中存储SQL*Plus命令

```
SQL> INPUT
      1 DESCRIBE EMP
      2
SQL> L
      1* DESCRIBE EMP
```

注意 参阅SQL*Plus命令表，缩写“L”代表“List”命令。

不可以从缓冲区中执行该命令，但是可以将它保存在一个文件中，可以以后在SQL*Plus中检索并执行这个文件。清单26-5显示了从缓冲区中执行一个SQL*Plus命令的尝试失败，后来成功地将它保存到文件中。

清单26-5 保存SQL*Plus命令到文件

```
SQL> RUN
      1* DESCRIBE EMP
DESCRIBE EMP
*
ERROR at line 1:
ORA-00900: invalid SQL statement
SQL> SAVE test
Created file test
SQL> GET test
      1* DESCRIBE EMP
```

新创建的文件被保存在由SQLPATH环境变量指定的目录中。

26.1.5 在SQL*Plus中使用你的操作系统编辑器

SQL*Plus提供的编辑能力非常有限，而且相对于其他的文本编辑器也不直观（例如，不能使用箭头键、HOME键或END键）。所以，许多用户喜欢使用他们自己感到满意的编辑器来创建命令文件，然后在SQL*Plus中使用START命令或@命令运行这些文件。

如果你宁愿使用你的操作系统编辑器而不愿使用SQL*Plus的编辑功能，你可以在SQL*Plus中使用EDIT命令实现这种操作。EDIT命令的语法为：

```
EDIT [file_name [.ext]]
```

上述命令将会使用glogin.sql或login.sql文件中变量_editor指定的编辑器打开名为file_name的文件。如果你想在SQL*Plus会话里使用另一种编辑器，你可以使用SQL*Plus命令DEFINE重新定义_editor变量：

```
DEFINE _editor=emacs
```

如果_editor变量没有定义，EDIT命令尝试使用操作系统缺省的编辑器（例如，在Windows 95环境下的记事本（Notepad））。

当你发出EDIT命令时，从SQL*Plus内部激活一个编辑器而不需要退出SQL*Plus，这样是非常方便的。

如果你运行EDIT命令时没有提供一个文件名，SQL*Plus将在一个文件中保存SQL缓冲区中的内容并使用编辑器打开这个文件。在缺省情况下，该文件的名称为afiedt.buf，它创建在

当前目录下或由SQLPATH环境变量定义的目录下。当打开编辑器的时候，你可以使用你希望编辑的文件的全路径名，例如：EDIT C:\MYDIR\MYFILE.SQL。

```
SQL> EDIT
Wrote file afiedt.buf
```

通过在editfile变量中设置合适的值，可以更改SQL*Plus用于保存缓冲区内容的文件的名称，如清单26-6所示。

清单26-6 被编辑的文件有一个缺省名称

```
SQL> SHOW editfile
editfile "afiedt.buf"
SQL> SET editfile "buffer.txt"
SQL> SHOW editfile
editfile "buffer.txt"
```

当运行EDIT命令时，如果你没有输入一个文件名并且SQL缓冲区为空，SQL*Plus返回一个通知消息，如清单26-7所示。

清单26-7 没有保存任何东西的空缓冲区

```
SQL> CLEAR BUFFER
buffer cleared
SQL> EDIT
Nothing to save.
```

缺省的文件名称的扩展名是.sql。所以，如果你没有指定一个文件扩展名，SQL*Plus将查找名为file_name.sql的文件。如果你想编辑一个扩展名不是.sql的文件，你必须明确地指定文件的扩展名。你还可以通过SUFFIX变量更改文件扩展名的缺省值，如清单26-8所示。

清单26-8 更改缺省的扩展名

```
SQL> SHOW SUFFIX
suffix "SQL"
SQL> SET SUFFIX sh
SQL> SHOW SUFFIX
suffix "sh"
```

注意 SUFFIX变量只适用于命令文件，不适用于输出文件。依赖于操作系统，输出文件的缺省扩展名是.lst或.lis。

26.1.6 运行SQL*Plus/SQL命令

你可以采用三种方式运行SQL命令和PL/SQL块：

- 命令行方式。
- SQL缓冲区方式。
- 命令文件方式（非正式地称为SQL脚本）。

为了以缓冲区方式执行SQL命令或PL/SQL块，SQL*Plus提供了RUN命令和/（前斜线）命令。RUN命令的语法是：

```
R[UN]
```

RUN命令列出并执行当前存储在缓冲区中的SQL命令或PL/SQL块。

我们假设缓冲区中包含以下查询：

```
SELECT empno, ename FROM emp
```

如果你使用RUN命令运行这个查询，它看起来如清单 26-9所示。

清单26-9 使用RUN命令运行一个查询

```
SQL> RUN
      1* SELECT empno, ename FROM emp
EMPNO      ENAME
-----
      7369 SMITH
      7499 ALLEN
      7521 WARD
      7566 JONES
      7654 MARTIN
      7698 BLAKE
      7782 CLARK
      7788 SCOTT
      7839 KING
      7844 TURNER
      7876 ADAMS
      7900 JAMES
      7902 FORD
      7934 MILLER
14 rows selected.
```

RUN命令显示缓冲区中的命令并返回查询的结果。另外，RUN命令使得缓冲区中的最后一行成为当前行。

/命令类似于RUN命令，它执行存储在缓冲区中的SQL命令或PL/SQL块，但它不显示缓冲区的内容，如清单 26-10所示。

清单26-10 使用/命令运行一个查询

```
SQL> /
EMPNO      ENAME
-----
      7369 SMITH
      7499 ALLEN
      7521 WARD
      7566 JONES
      7654 MARTIN
      7698 BLAKE
      7782 CLARK
      7788 SCOTT
      7839 KING
      7844 TURNER
      7876 ADAMS
      7900 JAMES
      7902 FORD
      7934 MILLER
14 rows selected.
```

注意 不同于RUN命令，/命令不会使缓冲区中的最后一行成为当前行。

要以命令行形式运行一个SQL命令、一个SQL*Plus命令或一个PL/SQL块，有两种命令：

■ START

■ @ (“at”)

START命令的语法是：


```
START file_name[.ext] [arg1 arg2...]
```

参数file_name[.ext]代表你想运行的命令文件，如果省略扩展名，SQL*Plus使用缺省的命令文件扩展名（通常为.sql）。

SQL*Plus在当前目录下查找具有你在START命令中指定的文件名和扩展名的文件。如果没有找到符合条件的文件，SQL*Plus将在SQLPATH环境变量定义的目录中查找该文件。也可以包括文件的全路径名，例如：C:\MYDYR\MYFILE.SQL。

可以在一个命令文件中包括正常情况下交互地输入的任何SQL命令、SQL*Plus命令或PL/SQL块。命令文件中的EXIT或QUIT命令用于退出SQL*Plus。

参数部分（[arg1 arg2...]）代表你希望传递给命令文件中的参数的值，命令文件中的参数必须使用如下格式声明：&1、&2、...（或&&1、&&2、...）。如果你输入一个或多个参数，SQL*Plus使用这些值替换命令文件中的参数。第一个参数替代每个&1，第二个参数替代每个&2，以此类推。

START命令使用参数的值定义参数。如果你在同一个SQL*Plus会话中再次运行这个命令文件，你可以输入新的参数或省略参数以使用参数的当前值。要运行名称为DELTBL.SQL的命令文件，输入以下代码：

```
SQL> START DELTBL
```

@（“at”）命令的功能与START命令非常类似，唯一的区别就是@命令既可以在SQL*Plus会话内部运行，又可以在启动SQL*Plus时的命令行级别运行，而START命令只能在SQL*Plus会话内部运行。要启动一个SQL*Plus会话并执行一个命令文件中的命令，输入下列代码：

```
$ sqlplus [username/password] @file_name[.ext] [arg1 arg2...]
```

如果START命令被禁止，同样会禁止@命令。本章后面的26.4节含有关于禁止SQL*Plus命令的详细信息。

1. 使用EXECUTE命令

从SQL*Plus 3.2开始，有一个新的命令——EXECUTE，该命令能够直接在SQL*Plus提示符状态下执行单条PL/SQL语句，而不需要从缓冲区或命令文件中执行。EXECUTE的主要用途是运行涉及一个函数或存储过程的PL/SQL语句，如清单26-11所示。

清单26-11 使用EXECUTE命令运行存储过程

```
SQL> VARIABLE id NUMBER          -- Define a bind variable
SQL> EXECUTE :id := ADD_CASES(10);
PL/SQL procedure successfully completed.
SQL> PRINT id
      ID
-----
      10
SQL> EXECUTE :id := ADD_CASES(3);
PL/SQL procedure successfully completed.
SQL> PRINT id
      ID
-----
       0
```

存储过程ADD_CASES返回的值存储在绑定变量:id中。

2. 保存SQL*Plus/SQL命令

使用SAVE命令，你可以将缓冲区中存储的SQL*Plus或SQL命令保存到一个操作系统文件（称为命令文件）中。SAVE命令的语法为：

```
SAV[E] file_name[.ext] [CRE[ATE] | [REP[LACE] | APP[END]]
```

file_name[.ext]是用于存储SQL缓冲区内容的操作系统文件的名称。要命名该文件，应使用SQL*Plus运行的操作系统的文件命名约定。如果你没有提供文件的扩展名，SQL*Plus使用缺省的扩展名.sql。你还可以指定一条路径作为文件名称的一部分。如果没有指明路径，SAVE命令将使用SQLPATH环境变量中指定的目录作为文件的路径。

CRE[ATE]参数创建命令文件。如果这个文件已经存在，你将收到一条错误消息。

REP[LACE]参数使用SQL缓冲区中的内容替换一个已有文件。如果文件不存在，SAVE...REPLACE命令创建它。

APP[END]参数将SQL缓冲区中的内容添加到文件的尾部。如果文件不存在，SAVE...APPEND命令创建该文件。

3. 提取SQL*Plus/SQL命令

要提取SQL*Plus或SQL命令，使用GET命令。该命令的语法是：

```
GET file_name[.ext] [LIS[T] | NOL[IST]]
```

GET命令加载包含SQL*Plus和/或SQL命令的操作系统文件file_name到SQL缓冲区，那么你就可以编辑文件中的命令或运行它们。文件的缺省扩展名为.sql。

LIS[T]参数在SQL*Plus加载文件到缓冲区时列出文件的内容。LIST参数是缺省设置。NOL[IST]参数不列出文件的内容。

如果在GET命令中没有指定文件名的全路径，SQL*Plus首先在当前目录下查找该文件，然后在环境变量SQLPATH中列出的目录中查找文件。

提示 如果使用Windows 95/NT，通过在指定目录下启动SQL*Plus应用，可以让SQL*Plus在该目录下查找命令文件。要实现它，更改Windows快捷方式的Start In属性为包含这些文件的目录。没有指定全路径的文件将在SQL*Plus启动时所在的目录中被打开、创建或保存。

26.2 使用SQL*Plus的COPY命令

虽然由于Oracle在7.2版中推出的CREATE TABLE...UNRECOVERABLE AS SELECT...命令在某种程度上削弱了COPY命令所起的作用，但COPY命令仍然是一条最为有用的SQL*Plus命令；但是人们对它不甚了解，所以不经常使用它。COPY命令可以用来实现几个功能：

- 从一个本地数据库将一个或多个表或整个模式拷贝到一个远程数据库或另外一个本地数据库。这可以用于从一个数据库将整个模式移动到另外一个数据库，而不需要使用导出/导入工具，当导出大于操作系统文件限制的文件时这尤其有用。
- 将一个表中指定的记录（基于查询）拷贝到远程数据库或本地数据库的其他表中。
- 将包含LONG类型数据列的表的内容拷贝到其他表。因为LONG类型的列不能用于SELECT语句中，所以这是解决此问题的唯一方法。
- 从一个Oracle数据库向一个非Oracle数据库拷贝表。

COPY命令的语法为：

```
COPY {FROM username[/password]@database_specification| TO
username[/password]@database_specification| FROM
username[/password]@database_specification TO
username[/password]@database_specification} {APPEND|CREATE|INSERT|REPLACE}
destination_table [(column, column, column ...)] USING query
```

username[/password]代表你希望拷出/拷入的Oracle用户名和口令。在FROM子句中, username/password标识数据的源;在TO子句中username/password标识数据的目的地。如果你既没有在FROM子句中指定口令也没有在TO子句中指定口令, SQL*Plus会提示你输入口令。SQL*Plus不显示对这些提示的用户响应。

database-specification (数据库声明)是数据库链接名、Net 8服务名或SQL*Net V2服务名。在COPY命令中, 必须提供数据库声明子句。在FROM子句中, 数据库声明代表源端数据库;在TO子句中, 数据库声明代表目的端数据库。

destination_table (目标表)是你希望创建的表或向其中添加数据的表, [(column, column, column...)]指示了目标表中的列名。

如果指定列, 列的数目必须与查询选择的列的数目相同。如果没有指定任何列, 如果COPY命令创建目标表, 被拷贝的列在目标表中将具有与源表中的列相同的名字。

USING query 参数指定一个提取COPY命令拷贝的行和列的SELECT语句。

FROM username[/password]@database_specification部分定义用户名、口令和含有需要拷贝的数据的数据库。如果省略FROM子句, 源数据库在缺省情况下为SQL*Plus连接的数据库。你必须包含一个FROM子句以指定缺省数据库以外的源数据库。

TO username[/password]@database_specification部分指明包含目标表的数据库。如果省略TO子句, 目标表在缺省情况下为SQL*Plus连接的数据库。必须包含TO子句以指定一个缺省数据库以外的目标数据库。

有几个参数用于控制COPY命令如何从一个表向另一个表中拷贝数据。这些参数如下所示:

- 如果目标表存在, APPEND将查询出的记录插入到目标表。如果目标表不存在, COPY命令创建该表。即使该表为空(不含数据), APPEND命令也插入记录。
- CREATE首先创建目标表, 然后向目标表中插入取自查询的记录。如果目标表已经存在, COPY命令返回一个错误。
- INSERT将取自查询的记录插入目标表。如果目标表不存在, COPY命令返回一个错误。当使用INSERT命令时, USING查询必须为目标表中的每列选择一行。
- REPLACE使用查询提取的记录替换目标表及其内容。如果目标表存在, COPY删除它并用含有拷贝数据的表替换它。如果目标表不存在, COPY命令创建它。

有三个SQL*Plus SET变量控制COPY命令的行为:

- LONG
- COPYCOMMIT
- ARRAYSIZE

LONG变量决定你所拷贝的LONG类型列的长度。如果LONG类型列包含的数据长度大于LONG变量的值, COPY命令将截断数据。

在缺省情况下, SQL*Plus在每个成功执行的COPY命令后面执行一次提交。如果设置SQL*Plus的SET变量COPYCOMMIT为一个正整型值n, SQL*Plus将在每拷贝n批记录后执行

一次提交。SET变量ARRAYSIZE决定批操作的大小。

注意 当使用COPY命令时，一些操作系统环境（特别是VAX/VMX、OpenVMX）要求以双引号括起数据库声明。

注意 如果在COPY命令的（column column, ...）子句中定义的任一列名含有小写字母或空格，列名必须使用双引号括起来。

注意 当使用COPY命令拷贝含有数字型（NUMBER）列的表时要非常小心！这些列在目标数据库中变为精度为38位的小数类型（DECIMAL）。

26.3 使用SQL创建SQL

在许多情况下数据库系统管理员或一个普通的用户需要对许多数据库对象（表、索引等）执行SQL语句。例如，数据库系统管理员可能会需要周期性地从一个成品数据库加载数据到开发数据库或测试数据库。在加载数据之前，他或她可能需要删除开发数据库或测试数据库中所有表中的数据。基于单个表来做这件工作会非常繁琐，尤其当数据库中有几百个表时。在这种情形下，一种简单快捷的解决方案是使用 SQL*Plus创建所需的SQL语句，将它们保存到一个操作系统文件中，并在 SQL*Plus中运行该文件。

例如，要删除属于用户 SCOTT的所有表中的所有记录，SYSTEM用户可以使用清单 26-12 中所示的SQL语句。

清单26-12 SQL生成SQL

```
SQL> CONNECT SYSTEM
Enter password: *****
Connected.
SQL> SET PAGESIZE 0
SQL> SET HEADING OFF
SQL> SET FEEDBACK OFF
SQL> SET VERIFY OFF
SQL> SET ECHO OFF
SQL> SPOOL DELTBL.SQL
SQL> SELECT 'DELETE '||TABLE_NAME||';'
2 FROM DBA_TABLES
3 WHERE OWNER='SCOTT';
DELETE ACCTS;
DELETE ACCT_ADDRS;
DELETE BONUS;
DELETE CITIES;
DELETE COMPANY_SUMMARY;
DELETE CUSTOMER;
DELETE DEPT;
DELETE DUMMY;
DELETE EMP;
DELETE FUNDS;
DELETE FUND_CONTRIB;
DELETE FUND_XACT;
DELETE F_EMPCOMP;
DELETE F_XACT_TYPE;
DELETE INVINFO;
DELETE INVREQUEST;
DELETE ITEM;
DELETE ORD;
DELETE ORDER_HISTORY;
```

```
DELETE PRICE;  
DELETE PRODUCT;  
DELETE SALES_REVENUE;  
DELETE SALGRADE;  
DELETE STOCK_HISTORY;  
SQL> SPOOL OFF
```

当你运行上述文件时，将得到文件 DELTBL.SQL，如下所示：

```
DELETE ACCTS;  
DELETE ACCT_ADDRS;  
DELETE BONUS;  
DELETE CITIES;  
DELETE COMPANY_SUMMARY;  
DELETE CUSTOMER;  
DELETE DEPT;  
DELETE DUMMY;  
DELETE EMP;  
DELETE FUNDS;  
DELETE FUND_CONTRIB;  
DELETE FUND_XACT;  
DELETE F_EMPCOMP;  
DELETE F_XACT_TYPE;  
DELETE INVINFO;  
DELETE INVREQUEST;  
DELETE ITEM;  
DELETE ORD;  
DELETE ORDER_HISTORY;  
DELETE PRICE;  
DELETE PRODUCT;  
DELETE SALES_REVENUE;  
DELETE SALGRADE;  
DELETE STOCK_HISTORY;
```

发出命令 @DELTBL.SQL 运行 DELTBL.SQL 文件将删除所示表中的记录。其他的命令用于格式化输出和创建命令文件：

- SET PAGESIZE 0 关闭全部页格式化信息，例如列标题、标题、起始空白行、分页符等等。
- SET HEADING OFF 关闭列标题显示（你可以使用 SET HEADING OFF 或 SET PAGESIZE 0 实现）。
- SET FEEDBACK OFF 隐藏查询返回的记录号。
- SET VERIFY OFF 不在 SQL*Plus 使用实际值替换变量的前后显示 SQL 命令的文本。
- SET ECHO OFF 可以在文件执行时，隐藏 DELETE.SQL 文件中的 SQL 命令清单。
- SPOOL DELTBL.SQL 开始将在 SQL*Plus 提示符下输入的命令查询结果保存入名为 DELTBL.SQL 的文件中。命令和查询结果继续在终端上显示。如果没有指明一个文件扩展名，SQL*Plus 为输出文件使用缺省的扩展名（.lst 或 .lis）。
- SPOOL OFF 停止保存并关闭 DELTBL.SQL 文件。

如果 SYSTEM 想删除 SCOTT 用户的全部表中的记录，他所要做的就是 在 SQL*Plus 中执行 DELTBL.SQL 文件，可以按照以下方式实现：

```
SQL>@DELTBL
```

26.4 在 SQL*Plus 中对用户权限的限制

Oracle 为数据管理员提供了一个工具，让他们能够禁止在 SQL*Plus 环境下指定的 SQL 和

SQL*Plus命令的执行，它基于单个用户实现。事实上，这个工具是一个表——PRODUCT_USER_PROFILE，由用户SYSTEM拥有。在表26-2中给出PRODUCT_USER_PROFILE的表说明；表26-3说明PRODUCT_USER_PROFILE表中每列的用途。

表26-2 PRODUCT_USER_PROFILE表定义

名 称	能否为空	类 型
PRODUCT	不能为空	VARCHAR2(30)
USERID		VARCHAR2(30)
ATTRIBUTE		VARCHAR2(240)
SCOPE		VARCHAR2(240)
NUMERIC_VALUE		NUMBER(15, 2)
CHAR_VALUE		VARCHAR2(240)
DATE_VALUE		DATE
LONG_VALUE		LONG

表26-3 PRODUCT_USER_PROFILE表的列的用途

列	用 途
PRODUCT	必须含有产品的名称（在本例中为SQL*Plus）。在该列中不允许使用通配符（%）或空（NULL）值
USERID	必须含有你希望禁止命令操作的用户的名称（大写）。如果针对几个用户禁止某个命令，使用通配符（%）或输入多条记录。如果针对全部用户禁止某个命令，只要在该列中插入一个通配符（%）即可
ATTRIBUTE	必须包含被禁用的SQL或SQL*Plus命令（例如ALTER）的名称（大写）。该列不允许使用通配符值
SCOPE	SQL*Plus忽略该列。应该在该列输入一个空（NULL）值。该列被保留用于非SQL*Plus产品
NUMERIC_VALUE	SQL*Plus忽略该列。应该在该列输入一个空（NULL）值。该列被保留用于非SQL*Plus产品
CHAR_VALUE	必须含有字符串"DISABLED"
DATE_VALUE	SQL*Plus忽略该列。应该在该列输入一个空（NULL）值。该列被保留用于非SQL*Plus产品
LONG_VALUE	SQL*Plus忽略该列。应该在该列输入一个空（NULL）值。该列被保留用于非SQL*Plus产品

作为SYSTEM用户运行脚本PUPBLD.SQL创建PRODUCT_USER_PROFILE表。这个文件的位置根据操作平台的不同而不同。例如，对于Windows 95下的个人Oracle系统，PUPBLD.SQL文件位于目录C:\ORAWIN 95\DBS中（C:\ORAWIN95是Oracle产品在Windows 95操作系统中的缺省ORACLE_HOME目录）。在UNIX平台，这个文件位于\$ORACLE_HOME/sqlplus/admin目录下。

在安装Oracle软件与创建起始数据库期间（对于UNIX平台在crdb脚本中，对于Windows 95/NT平台在buildall脚本中），PUPBLD.SQL命令文件被运行。如果在安装过程中没有创建起始数据库，必须作为SYSTEM用户运行PUPBLD.SQL脚本以创建PRODUCT_USER_PROFILE表。

如果PUPBLD.SQL没有运行，每当用户登录SQL*Plus时，他或她将收到下面的警告消息：

```
Warning: Product user profile information not loaded !
Error in disabling roles in product user profile.
```

（警告：产品用户环境资源文件信息没有加载！在产品用户环境资源文件中禁止角色产生错误。）

26.4.1 禁用一个SQL命令

要禁止某给定用户使用某个 SQL或SQL*Plus命令，SYSTEM用户必须插入与表 26-4所示列和值匹配的一条记录。

表26-4 用于限制SQL*Plus命令的值

列	值
PRODUCT	SQL*Plus
USERID	被限制的用户的用户名
ATTRIBUTE	被限制的命令名称
CHAR_VALUE	DISABLED
SCOPE	NULL (空)
NUMERIC_VALUE	NULL (空)
DATE_VALUE	NULL (空)
LONG_VALUE	NULL (空)

例如，要禁止用户 SCOTT从SQL*Plus访问操作系统，需要作为 SYSTEM用户登录到SQL*Plus并执行清单 26-13中的SQL语句。

清单26-13 禁止SCOTT用户使用HOST命令

```
INSERT INTO
product_user_profile
(product, userid, attribute, scope, numeric_value,
char_value, date_value, long_value)
VALUES
('SQL*Plus', 'SCOTT', 'HOST', NULL, NULL, 'DISABLED', NULL, NULL);
```

当用户访问SQL*Plus时，SQL*Plus从PRODUCT_USER_PROFILE表读取用户限制，并在用户对话期间强制实施这些限制。

注意 如果在PRODUCT_USER_PROFILE表中做出影响用户的任何改动，这些改变将不会影响用户的当前会话。用户可以在下一次登录到SQL*Plus时看到这些改变的效果。

26.4.2 重新允许使用一个SQL命令

要重新允许使用一个 SQL命令，需要删除含有限制的记录。清单 26-14重新允许使用在清单26-13中被禁止的命令。

清单26-14 允许用户SCOTT使用HOST命令

```
DELETE [FROM] product_user_profile
WHERE userid='SCOTT' and attribute='HOST';
```

可以使用PRODUCT_USER_PROFILE表禁止下面的SQL*Plus命令（以字母顺序排列）：

- COPY
- EDIT
- EXECUTE
- EXIT
- GET

- HOST（或者操作系统HOST的别名，例如UNIX上的‘！’或VMS上的‘\$’）
- QUIT
- PASSWORD
- RUN
- SAVE
- SET（参见下面的注释）
- SPOOL
- START（参见下面的注释）

注意 禁止SQL*Plus的SET命令还会禁止SET ROLE和SET TRANSACTION命令。

注意 禁止SQL*Plus的START命令还会禁止SQL*Plus的@和@@命令。

下面的SQL命令可以被禁止：

- ALTER
- ANALYZE
- AUDIT
- CONNECT
- CREATE
- DELETE
- DROP
- GRANT
- INSERT
- LOCK
- NOAUDIT
- RENAME
- REVOKE
- SELECT
- SET ROLE
- SET TRANSACTION
- TRUNCATE
- UPDATE

你还可以禁止下面的PL/SQL命令：

- BEGIN
- DECLARE

注意 禁止BEGIN和DECLARE命令不妨碍SQL*Plus的EXECUTE命令的使用。EXECUTE命令必须被单独禁止。

26.4.3 禁用SET ROLE

SQL命令SET ROLE使应用的用户能够访问获取应用的角色授予的特权，这样做可能会产生安全问题。要阻止应用的用户在SQL*Plus中获取应用的角色，数据库系统管理员可以使用

PRODUCT_USER_PROFILE表禁用SET ROLE命令。当他或她启动SQL*Plus时，该命令限制SQL*Plus用户只能使用角色允许的相关的权限。

26.4.4 禁用角色

要禁止一个指定用户的角色，应在PRODUCT_USER_PROFILE表中插入一条记录，该记录的UserID列中含有用户的用户名，Attribute（属性）列含有ROLES，Char_Value列含有角色名称。

注意 当禁用角色时，如果在UserID列中键入PUBLIC或%，便禁止了全部用户的角色。%或PUBLIC只能用于被授权为PUBLIC的角色。

Oracle用户SYSTEM拥有PRODUCT_USER_PROFILE表。当SYSTEM登录时，SQL*Plus不读取PRODUCT_USER_PROFILE表。因此，没有任何SQL*Plus限制应用到SYSTEM用户上。

注意 PRODUCT_USER_PROFILE表提供的产品级安全策略只由SQL*Plus实施，而不是由Oracle RDBMS实行。

26.5 追踪SQL语句

在SQL*Plus 3.3之前，获得有关在SQL*Plus中SQL语句的执行情况的统计手段非常有限。用于统计目的的命令只有SET TIME和SET TIMING。

SET TIME命令控制SQL*Plus如何显示当前时间。ON参数在每个SQL*Plus提示符前面显示当前时间，OFF参数关闭当前时间的显示。SET TIME命令的语法是：

SET TI[ME] [OFF | ON] （OFF是缺省值）

清单26-15显示了使用SET TIME命令的一个示例。

清单26-15 估算运行时间

```
SQL> SHOW TIME
time OFF
SQL> SET TIME ON
01:17:57 SQL> SELECT t1.dname, t2.ename, t2.sal, t2.job
01:18:11 2 FROM dept t1, emp t2
01:18:23 3 WHERE t1.deptno = t2.deptno;
```

DNAME	ENAME	SAL	JOB
RESEARCH	SMITH	800	CLERK
SALES	ALLEN	1600	SALESMAN
SALES	WARD	1250	SALESMAN
RESEARCH	JONES	2975	MANAGER
SALES	MARTIN	1250	SALESMAN
SALES	BLAKE	2850	MANAGER
ACCOUNTING	CLARK	2450	MANAGER
RESEARCH	SCOTT	3000	ANALYST
ACCOUNTING	KING	5000	PRESIDENT
SALES	TURNER	1500	SALESMAN
RESEARCH	ADAMS	1100	CLERK
SALES	JAMES	950	CLERK
RESEARCH	FORD	3000	ANALYST
ACCOUNTING	MILLER	1300	CLERK

```
14 rows selected.  
01:18:43 SQL>
```

通过用查询执行后第一个提示符显示的时间（01:18:43）减去最后一个输入行处显示的时间（01:18:23），用户可以得到一个执行这个查询所消耗的时间估算值（包含用于输入WHERE子句的时间）。

使用这种方法计算得出的运行时间不能反映 Oracle 执行一个查询的真正速度，因为流逝的时间包含将查询结果显示到 SQL*Plus 界面的时间消耗，运行查询的实际时间要短得多。

SET TIMING 命令控制执行时间统计的显示，SET TIMING 命令的语法为：

SET TIMING [OFF | ON] （OFF 是缺省值）

SET TIMING ON 命令显示 SQL*Plus 中执行的每一条 SQL 语句或 PL/SQL 块的执行时间统计。SET TIMING OFF 关闭时间统计的显示。

清单 26-16 显示了使用 SET TIMING 命令的一个示例。

清单 26-16 执行的计时

```
SQL> SHOW TIMING  
timing OFF  
SQL> SET TIMING ON  
SQL> SELECT t1.dname, t2.ename, t2.sal, t2.job  
2 FROM dept t1, emp t2  
3 WHERE t1.deptno = t2.deptno;  
DNAME          ENAME          SAL      JOB  
-----  
RESEARCH       SMITH          800      CLERK  
SALES           ALLEN         1600     SALESMAN  
SALES           WARD          1250     SALESMAN  
RESEARCH       JONES         2975     MANAGER  
SALES           MARTIN        1250     SALESMAN  
SALES           BLAKE         2850     MANAGER  
ACCOUNTING     CLARK         2450     MANAGER  
RESEARCH       SCOTT         3000     ANALYST  
ACCOUNTING     KING          5000     PRESIDENT  
SALES           TURNER        1500     SALESMAN  
RESEARCH       ADAMS         1100     CLERK  
SALES           JAMES         950      CLERK  
RESEARCH       FORD          3000     ANALYST  
ACCOUNTING     MILLER        1300     CLERK  
14 rows selected.  
real: 2690
```

如清单 26-15 和清单 26-16 所示，SET TIME 和 SET TIMING 命令提供了在 SQL*Plus 中 SQL 语句和（或）PL/SQL 块执行情况的非常基本且不是非常精确的计时统计。在 SQL*Plus 3.3 以前版本的 SQL*Plus 中，不能直接获得关于优化器使用的执行（访问）路径的任何信息。

SQL*Plus 3.3 版以及更高版本中使用了一个称为 AUTOTRACE 的新功能。AUTOTRACE 使 SQL*Plus 用户可以在 SQL*Plus 中查看数据库管理语言（DML）语句（SELECT、INSERT、UPDATE 和 DELETE）的执行计划，而不必使用 Oracle 提供的普通工具（SQL_TRACE、EXPLAIN PLAN 和 TKPROF 工具）来追踪 SQL 语句的执行情况。

要使用该功能，数据库系统管理员必须作为 SYS 用户运行一个称为 PLUSTRCE.SQL 的 SQL 脚本。PLUSTRCE.SQL 文件的位置依赖于操作系统。PLUSTRCE 脚本完成下列任务：

■ 创建一个名为 PLUSTRACE 的角色。

■ 为角色 PLUSTRACE 授予 V\$SESSTAT、V\$STATNAME 和 V\$SESSION 表的 SELECT 权限。

■ 使用 ADMIN 选项将 PLUSTRACE 角色授权给 DBA 角色。

要让用户具有 AUTOTRACE 能力，必须满足三个条件：

■ 用户必须被任何具有 DBA 角色的用户授予 PLUSTRACE 角色。

■ 用户必须在自己的模式中创建一个 PLAN_TABLE 表。用户可以运行 utlxplan.sql 脚本完成这项工作。在 UNIX 操作平台下该脚本文件位于目录 \$ORACLE_HOME/rdbms/admin 中，在 Windows 95/NT 操作系统下该脚本位于目录 %ORACLE_HOME%\Rdbms8x\Admin 下。

■ 必须适当地设置 SET 系统变量 AUTOTRACE。

在这些步骤执行后，在成功地运行任何 DML 语句（SELECT、INSERT、DELETE 和 UPDATE）后，用户能够得到关于优化器使用的执行路径的一个报告和执行统计。报告的输出由系统变量 AUTOTRACE 控制。

在表 26-5 中列出系统变量 AUTOTRACE 允许的設置和它們的使用結果。

表 26-5 AUTOTRACE 值

值	结 果
SET AUTOTRACE OFF	缺省值。不生成任何报告
SET AUTOTRACE ON EXPLAIN	追踪报告只显示执行路径，不显示执行统计
SET AUTOTRACE ON STATISTICS	追踪报告只显示执行统计，不显示执行路径
SET AUTOTRACE ON	追踪报告同时显示执行路径和执行统计
SET AUTOTRACE TRACEONLY	与 SET AUTOTRACE ON 相同，但是不显示查询的结果

26.5.1 理解执行计划

执行计划显示当执行一个查询时优化器使用的访问路径。使用 EXPLAIN PLAN 命令生成执行计划的输出。执行计划中显示的每一行都有一个连续的行号，也显示父操作的行号。执行计划由四列组成（对于标准查询，只显示三列，只有在分布式查询或使用并行查询选项（Parallel Query Option, PQO）运行查询的情况下才显示第四列）。

表 26-6 显示列的名称和它们的说明。

表 26-6 执行计划列说明

列 名	说 明
ID_PLUS_EXP	显示每步执行步骤的行号
PARENT_ID_PLUS_EXP	显示某步骤和它的父步骤之间的关系
PLAN_PLUS_EXP	显示报告的每步骤；例如 TABLE ACCESS (FULL) OF 'DEPT'
OBJECT_NODE_PLUS_EXP	显示使用的数据库链接或并行查询服务器（只有在运行分布式查询或使用并行查询选项（PQO）时）

缺省的列格式通常在站点环境资源文件（glogin.sql 文件）中设置。

列的格式可以使用 SQL*Plus 命令 COLUMN 加以改变。可以更改列的宽度、重新命名列或停止显示列。例如，要禁止显示 ID_PLUS_EXP 列，应键入：

```
SQL> COLUMN ID_PLUS_EXP NOPRINT
```

此语句追踪报告的第二部分显示语句执行的统计。它们显示用于执行查询的系统资源和它们的用法。与执行路径不同，不能改变统计报告的缺省格式。

26.5.2 使用AUTOTRACE功能

本节详细介绍为使SCOTT用户能够在SQL*Plus中使用AUTOTRACE功能所需要的步骤。

首先, 用户SYSTEM登录到SQL*Plus, 并授予用户SCOTT PLUSTRACE角色 (SYS用户已在先前运行过 \$ORACLE_HOME/sqlplus/admin/plustrce.sql脚本, 在数据库中创建了PLUSTRACE角色)。清单26-17显示该项工作已经完成。

清单26-17 授权PLUSTRACE角色

```
SQL>

C:\>sqlplus system

SQL*Plus: Release 8.1.3.0.0 - Beta on Sun Mar 7 09:25:09 1999

(c) Copyright 1998 Oracle Corporation. All rights reserved.

Enter password:

Connected to:
Oracle8 Enterprise Edition Release 8.1.3.0.0 - Beta
With the Partitioning and Objects options
PL/SQL Release 8.1.3.0.0 - Beta
SQL> GRANT plustrace TO scott;
Grant succeeded.
```

接着, 用户SCOTT登录到SQL*Plus, 并通过运行 \$ORACLE_HOME/rdbms80/admin/utlxplan.sql脚本在他的模式中创建PLAN_TABLE (这是EXPLAIN PLAN命令需要的), 如清单26-18所示。

清单26-18 创建PLAN_TABLE

```
SQL> CONNECT scott
Enter password: *****
Connected.
SQL> @$ORACLE_HOME/rdbms80/admin/utlxplan
Table created.
SQL> L
 1 create table PLAN_TABLE (
 2   statement_id    varchar2(30),
 3   timestamp       date,
 4   remarks         varchar2(80),
 5   operation       varchar2(30),
 6   options         varchar2(30),
 7   object_node     varchar2(128),
 8   object_owner    varchar2(30),
 9   object_name     varchar2(30),
10   object_instance numeric,
11   object_type     varchar2(30),
12   optimizer       varchar2(255),
13   search_columns  numeric,
14   id              numeric,
15   parent_id       numeric,
16   position        numeric,
17   cost            numeric,
18   cardinality     numeric,
19   bytes           numeric,
20   other_tag       varchar2(255),
21*  other           long)
```

L[IST]命令列出 SQL 缓冲区中的内容，它含有从 ORACLE_HOME/rdbms80/admin/utlxplan.sql 脚本中执行的最后一条 SQL 命令 —— CREATE TABLE plan_table 语句。

接着，SCOTT 用户输入并执行他希望追踪的查询。清单 26-19 显示了一个示例。

清单 26-19 追踪一个查询

```
SQL> SELECT t1.dname, t2.ename, t2.sal, t2.job
2 FROM dept t1, emp t2
3 WHERE t1.deptno = t2.deptno;
DNAME          ENAME          SAL      JOB
-----
RESEARCH        SMITH           800      CLERK
SALES            ALLEN           1600     SALESMAN
SALES            WARD            1250     SALESMAN
RESEARCH        JONES           2975     MANAGER
SALES            MARTIN          1250     SALESMAN
SALES            BLAKE           2850     MANAGER
ACCOUNTING      CLARK           2450     MANAGER
RESEARCH        SCOTT           3000     ANALYST
ACCOUNTING      KING            5000     PRESIDENT
SALES            TURNER          1500     SALESMAN
RESEARCH        ADAMS           1100     CLERK
SALES            JAMES           950      CLERK
RESEARCH        FORD            3000     ANALYST
ACCOUNTING      MILLER          1300     CLERK
14 rows selected.
```

要同时得到执行路径和执行统计的信息，用户 SCOTT 必须正确地设置 AUTOTRACE 变量。清单 26-20 显示了用于设置 AUTOTRACE 为打开状态并从缓冲区中运行先前的查询的命令。

清单 26-20 追踪缓冲区中的查询

```
SQL> SET AUTOTRACE ON
SQL> /
Execution Plan
-----
0      SELECT STATEMENT Optimizer=CHOOSE
1  0    NESTED LOOPS
2    1    TABLE ACCESS (FULL) OF 'EMP'
3    1    TABLE ACCESS (BY ROWID) OF 'DEPT'
4    3    INDEX (UNIQUE SCAN) OF 'DEPT_PRIMARY_KEY' (UNIQUE)
Statistics
-----
0  recursive calls
2  db block gets
43 consistent gets
0  physical reads
0  redo size
726 bytes sent via SQL*Net to client
376 bytes received via SQL*Net from client
3  SQL*Net roundtrips to/from client
0  sorts (memory)
0  sorts (disk)
14 rows processed
```

不显示查询数据，追踪相同的语句，如下所示：

```
SQL> SET AUTOTRACE TRACEONLY
SQL> /          - slash command: doesn't display query text
14 rows selected.
Execution Plan
-----
```

```

0      SELECT STATEMENT Optimizer=CHOOSE
1      0      NESTED LOOPS
2      1      TABLE ACCESS (FULL) OF 'EMP'
3      1      TABLE ACCESS (BY ROWID) OF 'DEPT'
4      3      INDEX (UNIQUE SCAN) OF 'DEPT_PRIMARY_KEY' (UNIQUE)
Statistics
-----
0 recursive calls
2 db block gets
43 consistent gets
0 physical reads
0 redo size
726 bytes sent via SQL*Net to client
376 bytes received via SQL*Net from client
3 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
14 rows processed

```

提示 当调整一个返回大量记录的查询时，TRACEONLY选项特别有用。

执行统计中涉及的客户是 SQL*Plus。不管 SQL*Net 是否被安装并使用，SQL*Net 是指客户（SQL*Plus）和服务器（Oracle RDBMS）之间通常的进程间的通信。

26.6 SQL*Plus 8.1版的增强

SQL*Plus 8.1版给予用户实现下列功能的能力：

- 访问所有的 Oracle 8i 数据类型和用户定义的数据类型。
- 访问所有 Oracle 8i 引入的 SQL 新特性。
- SQL*Plus 8.1 版拥有 8.0 版具有的全部特性。

而且，服务管理器功能目前在 SQL*Plus 中已可使用。这些功能包括：启动、关闭、显示参数、显示系统全局区（SGA）、归档日志、恢复数据库和内部连接等。

有一个名为 ARCHIVE LOG 的新命令，ARCHIVE LOG 命令使重做日志文件被归档，它还显示重做日志文件的信息。

例如：

```

SQL> connect internal
Connected.
SQL> archive log list
Database log mode           No Archive Mode
Automatic archival          Enabled
Archive destination         D:\ORANT\oradata\orcl2\archive
Oldest online log sequence  2
Current log sequence        5
SQL>

```

你现在可以在 SQL*Plus 中使用 RECOVER 命令恢复一个数据库。RECOVER 命令执行表空间、数据文件或整个数据库的介质恢复。

例如：

```

SQL> recover database
ORA-00283: recovery session canceled due to errors
ORA-01124: cannot recover data file 1 - file is in use or recovery
ORA-01110: data file 1: 'D:\ORANT\ORADATA\ORCL2\SYSTEM01.DBF'

```

SET 命令现在有一个 INSTANCE 子句，INSTANCE 子句将会话的缺省实例改变为指定的实例。SET 命令现在还有一个 LOGSOURCE 子句，LOGSOURCE 子句指定数据库恢复期间检索

归档日志的路径。

现在你可以在SQL*Plus中使用SHOW PARAMETERS命令查看init.ora参数。

例如：

```
SQL> show parameters block
```

NAME	TYPE	VALUE
db_block_buffers	integer	1000
db_block_checking	boolean	TRUE
db_block_checksum	boolean	FALSE
db_block_lru_latches	integer	1
db_block_max_dirty_target	integer	1000
db_block_size	integer	2048
db_file_multiblock_read_count	integer	8
hash_multiblock_io_count	integer	1
sort_multiblock_read_count	integer	2

可以在SQL*Plus中查看系统全局区（SGA）信息。

例如：

```
SQL> show sga
```

```
Total System Global Area  20396276 bytes
Fixed Size                  63732 bytes
Variable Size              18210816 bytes
Database Buffers           2048000 bytes
Redo Buffers                73728 bytes
SQL>
```

可以在SQL*Plus中使用带有安装和打开一个数据库的选项的STARTUP命令来启动数据库。

例如：

```
SQL> startup
ORACLE instance started.
Total System Global Area      20396276 bytes
Fixed Size                    63732 bytes
Variable Size                 18210816 bytes
Database Buffers              2048000 bytes
Redo Buffers                   73728 bytes
Database mounted.
Database opened.
SQL>
```

可以在SQL*Plus中使用SHUTDOWN命令关闭一个数据库，SHUTDOWN命令关闭当前正在运行的一个Oracle实例。该命令也可以关闭和卸载一个数据库。

例如：

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

SQL*Plus是最经常用来和数据库交互并执行系统管理功能如启动、关闭或恢复数据库等的唯一工具。它是一个交互式的工具，允许用户创建、修改、删除或查询数据库对象如表、视图、索引和约束。它还允许具有适当权限的用户创建新的用户，向他们授予特权或修改现有的用户。拥有适当权限的用户能够在SQL*Plus中删除其他用户和他们的权限。在Oracle 8i中，所有的服务器管理器命令都合并入SQL*Plus，所以数据库系统管理员能够使用这个工具有效地管理他们的数据库。

第27章 Oracle企业管理器

本章要点：

- 理解企业管理器体系结构
- 开始学习
- 使用控制台功能
- 使用数据库管理工具
- 使用诊断包
- 使用调整包
- 使用企业增值产品
- 小结

27.1 理解企业管理器体系结构

企业管理器不只是一套数据库管理工具，该产品为企业范围分布式系统管理解决方案提供一个框架。例如，一个数据库系统管理员可能居住在洛杉矶，却在相同的集成控制台上管理着纽约、伦敦和东京的数据库。一系列集中定位的 Oracle表形成一个知识库，在知识库中存储用于支持每一个数据库系统管理员行为所需的信息。这个知识库由新的管理服务器（Management Server）维护，能够驻留在支持企业管理器客户端的任意网络节点或本地 Windows 95/98或Windows NT工作站上。

企业管理器是开放的和可扩展的。工具命令语言（Tool Command Language, Tcl），发音为“tickle”，是一个非私有且广泛使用的脚本编程语言，具有易于使用和可扩展特性。Tcl用于提交命令给远程操作系统和数据库执行。企业管理器使用的 Tcl工具称为OraTcl，包含你用于完全管理一个Oracle数据库环境所需功能的扩展：

- 启动和停止Oracle数据库。
- 执行SQL。
- 访问操作系统资源和函数。
- 访问描述Oracle数据库的简单网络管理协议（Simple Network Management Protocol, SNMP）管理信息基础（Management Information Base, MIB）变量。
- 访问拥有简单网络管理协议的非-Oracle设备和服务。

注意 要熟悉Tcl，查看出现在%ORACLE_HOME%\SYSMAN\SCRIPTS\TCL中的企业管理器Tcl脚本。

简单网络管理协议（SNMP）是远程智能代理与非 Oracle的软件与硬件建立通信所要使用的一个开放标准。最初，简单网络管理协议设计用于与网络设备通信，但它现在也可以用于与应用通信。使用简单网络管理协议标准，能够让大量的第三方厂家的软件和硬件产品与企业管理器集成。应用开发者使用对象链接与嵌入（Object Linking and Embedding, OLE）技术或者调用针对企业管理器的一个发布的应用程序接口（API）在控制台内部集成应用。更进

一步, 当信息在控制台与远程智能代理之间的网络上移动时, 可以使用 Net 8 的高级网络选项 (Net 8 Advanced Networking Option, ANO), 通过 Oracle 安全网络服务保护信息。这一级的安全特性使得管理因特网上的远程数据库变为可能。

所有这些组件和技术通过使用一种体系结构连接到一起, 这种体系结构使用将面向图形的控制台客户链接到与数据库通信的远程智能代理 (见图 27-1)。除了通信后台服务进程和智能代理之间的信息之外, 应用组件例如数据库管理工具通过在 Net 8 上使用 SQL 命令直接与远程数据库通信。由应用组件直接在远程数据库上实时执行的任务不使用通信后台服务进程。通信后台服务进程使用本地的 TCP/IP 协议或 Net 8 透明网络底层 (Transparent Network Substrate, TNS) 连接与远程智能代理通信, 完成核心的控制台功能例如工作时间表安排和事件管理。

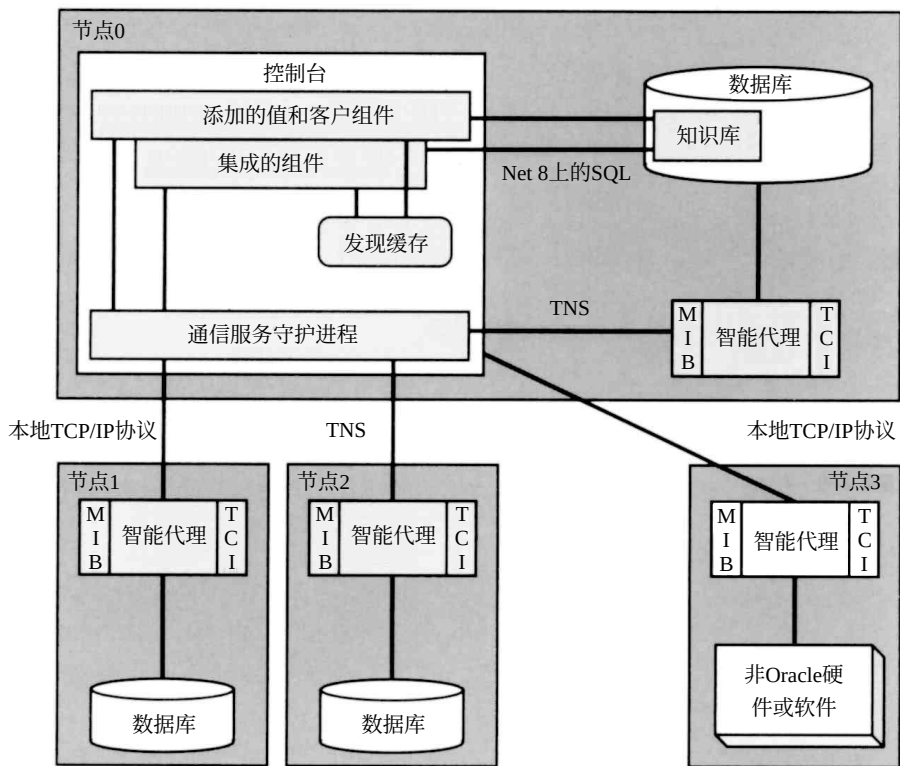


图27-1 企业管理器结构提供集中的数据库管理

发现缓存 (discovery cache) 是一个大的缓存区, 用于存储描述由控制台管理的服务和节点的元数据。对象被第一次访问时元数据被收集, 并且在会话的剩余时间中元数据存储于发现缓存中。当发现缓存中的元数据被访问时, 不需要重复查询以确定远程数据库的结构。因此, 响应时间加快了, 并且避免了不必要的资源使用。在会话结束后, 数据被存储到知识库中, 便于在下一个会话中使用。自定义应用也能够访问发现缓存。

智能代理自动地执行和管理远程动作。在通信服务守护进程指示远程代理在给定时间执行特定的脚本后, 脚本独立于控制台执行。当到了远程任务应该被执行的时间时, 如果控制台不可用, 代理管理任务的执行并缓存最多 500 条返回消息。当控制台变为可用时, 代理将这

些消息传递回控制台，并重新建立通信。与此类似，智能代理独立于控制台处理远程事件的监控和响应。

大多数企业管理器的安装没有完全利用该体系结构。在某些方面，这是对使用诸如模式管理器组件而不依赖于整个体系结构以支持实时数据库系统管理的产品灵活性的一个贡献。早已使用稳固的第三方系统管理工具（例如 BMC Patrol、CA-Unicenter、Compuware EcoTools或 Platinum ProVision）的站点，如果实现整体框架，将重复现有的功能，同时增加系统管理总开销和系统工作量。没有使用类似工具的节点也许会发现，企业管理器与 Oracle 8i捆绑在一起，不需要额外费用，是一个合适的替代品（诊断包等增值产品在付出额外费用后也可以使用）。

由于企业管理器范围的限制和当前可用的扩展企业管理器功能的附加工具有限，它不是一个稳固的能很好地扩展覆盖数据库管理各个方面的系统管理产品的替代品。尤其在大的企业中，操作人员可以从通用用户界面的效率及跨越多个平台、数据库技术和其他的 management 技术完成任务中受益，一个更为全面的解决方案或许会增加开销。在任何情况下，在任意 Oracle 数据库环境中，企业管理器的某些组件都有可能占有一席之地。

27.2 开始学习

尽管企业管理器拥有稳固的体系结构，但是开始使用它却非常容易。在软件安装后，像模式管理器（Schema Manager）这样的工具不需要知识库或远程智能代理就可以立即起作用。知识库可以保证一些功能的实现，例如用于性能分析和容量计划的数据累计。通过存储先前会话之间的验证，知识库使得一些动作更为容易，例如登录到控制台。由于存储的信息随系统管理员不同而不同，每个系统管理员应该拥有一个独立的知识库。在任意给定的时间，只能有一个企业管理器的实例连接到知识库。

开始研究企业管理器并能立即从中得到好处的最佳地方是使用下列数据库系统管理工具：

- 实例管理器（Instance Manager）。
- 模式管理器（Schema Manager）。
- SQL 工作单（SQL Worksheet）。
- 安全管理器（Security Manager）。
- 存储管理器（Storage Manager）。

有四种方式可以使用这五种工具。第一种方式是直接从 Windows 任务栏使用。例如，从任务栏上的 Start 菜单，依次选择 Programs | Oracle-oracle home | DBA Management Pack，然后选择 Instance Manager 以启动实例管理器。

使用数据库管理应用程序的第二种方式是从企业管理器控制台的应用运行面板。在你可以使用控制台之前，建立一个知识库十分必要。不同于企业管理器的早期版本，知识库的创建过程自动执行。第一次控制台使用一个给定的用户识别号（UserID）启动，知识库管理器说明启动控制台所必须的子组件。当你对提示信息响应 OK 时，知识库管理器为它自己、企业管理器和软件管理器创建一个知识库，并且它将启动发现新服务向导（Discover New Services Wizard）。如果网络上已经有拥有 Oracle 智能代理配置的节点，这个向导会与远程节点通信，在控制台上填充导航树。因为这项功能可以在任意时间完成，为了快速开始学习可以略过它。

注意 因为知识库是在首次登录时自动创建的，你必须小心避免为一个系统用户（例如 SYSTEM）在 SYSTEM 表空间创建一个知识库。在 SYSTEM 表空间中的知识库会引起

磁盘碎片和空间约束，反过来会影响性能和可管理性。

在企业管理器控制台打开后，使用 Applications 工具条中的 Database Administration Tools 运行实例管理器、模式管理器和其他的数据库管理工具。其余的工具——备份管理器、数据管理器和软件管理器——只能从控制台这里使用。

使用数据库系统管理工具的第三种方式也是通过在菜单条上选择 Tools | Application 从控制台内部使用数据库管理工具。

企业管理器现在已经准备好使用，但企业管理器不知道任何数据库或监听器，除非发现新服务向导利用远程智能代理自动地发现并定义它们。要添加数据库和节点表示系统，从菜单条上选择 Navigator | Service Discovery | Manually Define Services 以后启动服务定义向导。该向导全过程引导数据库系统管理员完成为企业管理器定义监听器和数据库的过程。数据库系统管理工具现在可以通过第五种方式（即最后一种方式）使用，即在导航器窗口左上方以右键点击任一个新添加的数据库对象，从快捷菜单选择 Related Tools，查看可用工具的上下文相关的列表，以访问选中的数据库（见图 27-2）。

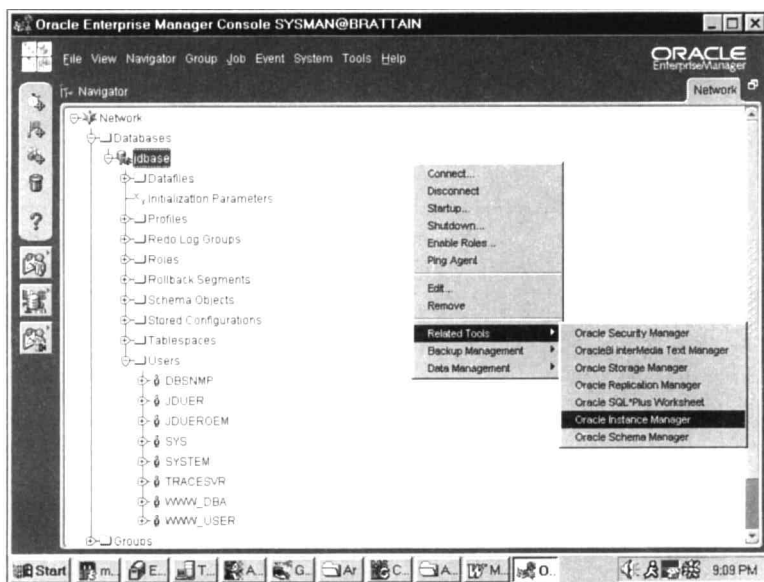


图27-2 上下文相关的菜单使得模式管理器可以通过控制台访问数据库

27.3 使用控制台功能

当数据库系统管理员熟悉企业管理器时，控制台功能通常不再是该产品有吸引力的地方。多数数据库系统管理员因为数据库系统管理工具和诊断与调整工具包，而对企业管理器感兴趣。但是，如果没有事件管理和作业时间表组件与远程智能代理协调工作，这些组件的一些重要部分会被削弱或根本不起作用；

- 没有作业时间表，备份管理器不能执行备份任务或恢复任务。
- 没有作业时间表，数据管理器不能执行导入、导出或加载数据任务。
- 没有作业时间表，软件管理器不能分布软件。
- 没有作业时间表，追踪管理器不能收集用于分析的性能数据。

- 表空间管理器的碎片消除与合并功能要求使用作业时间表。

- 伴随诊断与调整工具包发生的高级事件依赖于事件管理器。

使用本节中的资料信息实现整个体系结构，这样企业管理器的承诺便能实现。本书中有关集成的控制台功能实现和配置的重点在于：让数据库系统管理员充分利用依赖于控制台组件的所有其他的有用功能的优势。

企业管理器集中了数据库环境的控制。不论管理的是单一数据库还是一个全球企业数据库，控制台提供了统一的管理。有四个主要的控制台功能，它们中的每一个都由控制台窗口的一个面板表示：

- 导航器 (Navigator)。

- 组 (Group)。

- 事件管理 (Event Management)。

- 作业时间表 (Job Scheduling)。

导航器提供一个树型结构代表所有的数据库、服务器、Net 8监听器、Web服务器和这些组件的组。对数据库系统管理工具的上下文相关的访问仅仅是一些鼠标点击操作而已。组（正式地称为映射）提供了一种地理地追踪系统的状态和深入系统内部检查监听器和数据库的方法。与事件功能相结合，映射可以一目了然地显示系统状态。

事件管理和作业时间表依赖于远程智能代理的可用性。事件管理组件与远程智能代理通信，追踪远程系统上的活动和情况。企业管理器使用这种能力发送电子邮件、广播找人或运行一项作业以改正某个错误，或捕捉诊断信息便于日后分析。如果运行一项作业，目的在于解决或防止某个错误出现，作业时间表组件处理该项任务。另外，作业时间表可以启动、监控并报告远程系统上一系列动作的完成。

27.3.1 理解集成的控制台功能

一些控制台功能应用于企业管理器的所有其他组件以加强安全性、使应用可用、管理控制台窗口的四个面板和管理通信后台服务进程。

1. 安全性

从控制台管理企业管理器的安全性功能，可以通过在菜单上选择 System | Preference 进行访问。数据库系统管理员可以存储访问由企业管理器管理的数据库所必须的口令。当需要使用许多具有不同口令的用户标识以管理一个大的环境时，通常应将这些口令与用户记到纸条上或笔记本中。要记住 20 个用户标识和相关的不断更换的口令几乎是不可能的。用户更希望至少让数据库系统管理员能将口令存储在一个安全的环境中，在那里它们不会意外地落到坏人手中。

2. 执行面板

执行面板 (Launch palette) 是位于控制台左边的小工具条。通过在菜单条上选择 view，然后选择 Show/Hide Toolbar，可以访问并管理执行面板。没有增值或自定义产品的企业管理器缺省安装只提供应用面板。

通过从菜单条选择 View | Show Toolbar，可以激活企业管理器工具条。这个工具条由针对相关任务的五个按钮部分组成（见图 27-3）。这些应用面板上可见的任务管理提供拖放帮助、管理映射图像、创建并删除事件以及创建并删除作业。

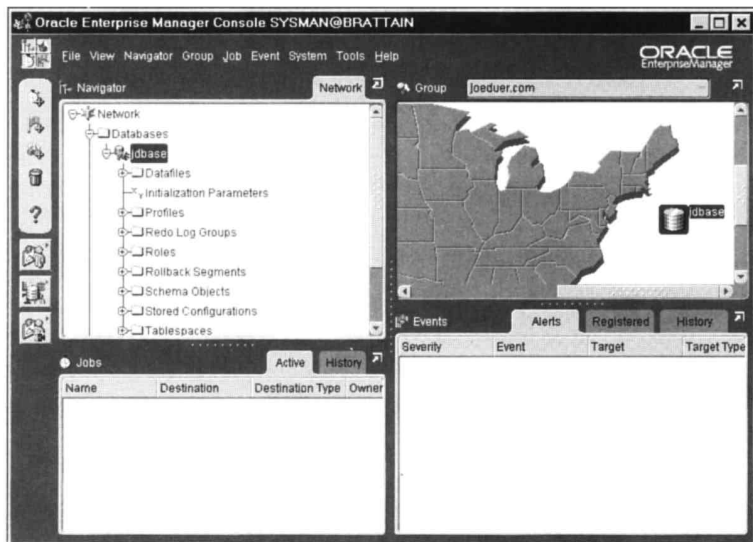


图27-3 企业管理器工具条（最左边）为映射、事件和作业控制台面板提供工具

27.3.2 使用导航器浏览数据库

导航器是企业管理器控制台中最广为使用的部分。它的工作非常类似 Windows 的资源管理器，所以它的界面是直观的并且是立即可用的。右击任意对象会显示一个鼠标菜单创建新的树型视图，访问导航器功能或使用相关的数据库系统管理工具。

从 Navigator 菜单，选择 Split View，创建一个引用点，便于在导航器窗口顶部使用 TAB 键能够容易地访问新建视图中某个树的顶层。在一个复杂的数据库或操作环境中，树型视图使得对感兴趣的环境部分的访问更为高效（见图 27-4）。过滤器提供另外一种对感兴趣对象的树枝进行限制的选择方案。例如，如果全部的通用帐目对象使用“GL_”打头，一个值为 GL_% 的过滤器将限制树型结构中该过滤器下面的对象为那些属于通用帐目系统的对象。



图27-4 导航器提供一种到达并操作任意数据库中任意对象的方法

同样的菜单允许数据库系统管理员创建相同类型的新对象或删除已有的对象。一些对象的修改可以直接在导航器中使用快速编辑（Quick Edit）实现，但是较为复杂的更改需要通过选择相关的工具，使用在选定的对象上进行操作的一个数据库系统管理工具来完成。

提示 因为快捷菜单是上下文相关的，可用的操作在不同对象间的变化很大。

导航器还是组的源组件。要向映射中添加一个对象，只要简单地从导航器中将对象拖动到映射中，并将对象放到映射中你希望它所在的位置即可。

使用导航器，一个数据库系统管理员不需要编写一条 SQL 语句就可以访问任意数据库中的任意对象、创建新对象和删除已有的对象。导航器容易地成为数据库系统管理员用来手工查询并改变数据库和它们对象的主要界面。

27.3.3 使用组使数据库形象化

映象提供了一种地理导向方法来监控数据库、节点、监听器和 Web 服务器的状态和基本信息，还有上述四个对象的组的信息。亚洲、欧洲、日本、美国和世界的映象与产品一起提供，但是任何位图文件（*.BMP）都是有效的映象。该映象甚至不必是地理的。可以考虑使用组织图、生产流程和网络图作为可供选择的映象。控制台工具条上的快速切换映像能力使企业可以有許多视图。

与远程智能代理结合使用，组使用一个小信号旗指明映象上每个对象的状态。绿色的信号旗意味着一切正常。黄旗指示有一些情况需要引起注意，红旗表明一个严重的故障。企业管理器的事件管理组件使数据库系统管理员能够定义必要的条件和门限以触发信号旗指定的状态的改变。双击映象对象启动对象的一个只看快速编辑器。数据库系统管理工具通常从控制台中启动；如果在映象中选定一个对象，可以直接针对这个对象启动数据库系统管理工具。

27.3.4 使用作业使数据库系统管理任务自动化

企业管理器的作业时间表组件——即作业（Job）——为数据库系统管理员在指定时间或在操作环境中基于事件自动执行任务提供了一种方法。时间表在具有多个系统的复杂环境下尤为有用。作业可以执行需要操作系统命令或 SQL 语句的任何任务。远程代理在处理实际的作业执行时独立于企业管理器控制台。

提示 如果 Oracle 智能代理没有安装和配置，作业不可用。

作业伴随着准备配置并计划的大量预定义任务。创建一个作业的过程可以用一个例子很好地解释。要计划一个远程数据库的导出，从企业管理器的菜单条上选择 Job | Create Job 开始。出现的对话框中有五个选项卡：General、Tasks、Parameters、Schedule 和 Permissions。

General 选项卡给作业一个名称和说明。然后，选择目标类型和目标。要执行一个数据库导出，目标类型应该是 Database。接下来，转移到 Tasks 选项卡。有超过一打的数据库和系统构造块任务可以用于配置一个作业。从可用的任务列表中选择 Export 任务并将它移动到任务作业列表。更为复杂的作业由多个按要求排序的任务组成。

接着，选择 Parameters 选项卡，这里管理着作业中每项任务的配置信息。对于你的导出作业，选择 Details 按钮启动导出向导，并决定你是导出整个数据库还是导出指定的用户和表（见图 27-5）。

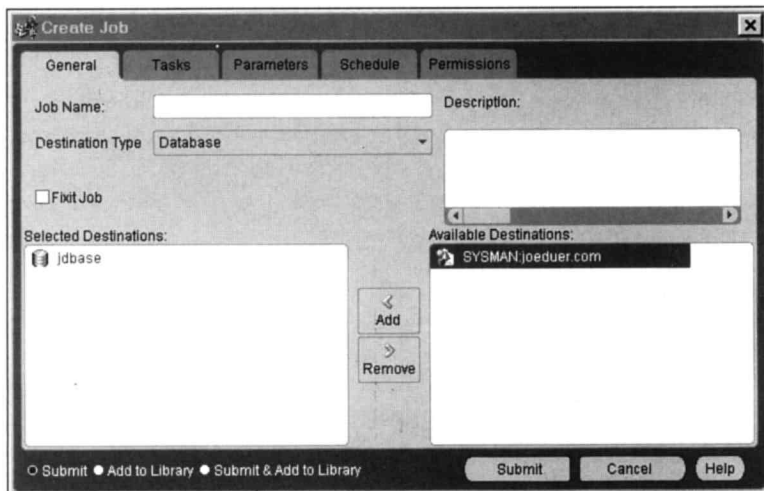


图27-5 许多任务构造块可以用来创建一个作业，每个块都有它自己的参数

提示 如果在可用的对象列表中没有对象，进入导航器并在Database Groups文件夹中创建一个数据库组。数据库组组成可用对象列表。

导出作业现在已准备好按时间表工作。选择 Schedule选项卡并定义作业应该在何时执行。按下Submit按钮保存作业；作业已经做好在指定时间执行的准备。当 Job Library选项卡被选中时，作业的名称和说明在作业控制台窗口中显示。有关活动的作业和已完成的作业的信息可以从其他选项卡中得到。

Job是易于使用且组织良好的作业时间表系统，它给予数据库系统管理员直接从企业管理器控制台自动执行不同系统上的任务的能力。但是，作业并不是功能更为强大的综合系统管理包的作业时间表组件的替代品，这些作业时间表组件提供了基于远程系统上一个文件的存在而具有的诸如作业重启及作业触发器的特性。全方面使用 Job的关键在于理解你的作业时间表的需求。如果你当前正使用 cron命令来计划你的基于 UNIX的全部作业，Job是一个巨大的提高。如果你正在考虑实现一个更为强大的作业时间表过程，将你的全部操作作为一个企业范围系统管理策略的一部分，你或许会发现其他产品更为合适。

27.3.5 使用事件管理响应变化

事件管理系统（Event Management System, EMS）是企业管理器的组件，用于监控远程数据库和服务是否正常且良好，并使它们对应用是可用的。作业按照特定时间表触发任务，而EMS与远程智能代理合作不间断地监控非正常的情况（即事件），然后基于数据库系统管理员定义的说明管理这些事件。事件可以触发一个作业，预防或修正动作以保持系统可用，并收集信息便于日后分析。如果事件足够严重，映象被通知更改图形显示的信号旗。EMS甚至能够发送电子邮件或使用网页通报某个事件。

类似于作业，EMS使用在本地客户端的通信后台服务进程和每个系统上的远程智能代理来监控并响应事件。智能代理实际地监控系统的资源，所以不需要企业管理器控制台的直接参与，事件一天24小时都被管理。如果没有安装和配置 Oracle智能代理，作业不可用。

要对一个特定事件的远程资源开始监控，该事件必须在 EMS中注册。有大量的预定义事

件准备配置和注册。开始监控的过程可以用一个例子很好地说明。要监控一个远程监听器的可用性，首先从企业管理器菜单条上选择 Event | Create Event Set开始，出现的对话框包含4个选项卡：General、Tests、Parameters、Permissions。

首先选择General选项卡，给事件一个名字和说明并选择目标的类型。要监控一个数据库，目标类型应该是Database。接下来，转到Tests选项卡。在可用事件列表中选择 UpDown事件并将它移动到选中的事件列表中。较为复杂的事件集由多个事件组成，如图 27-6所示。

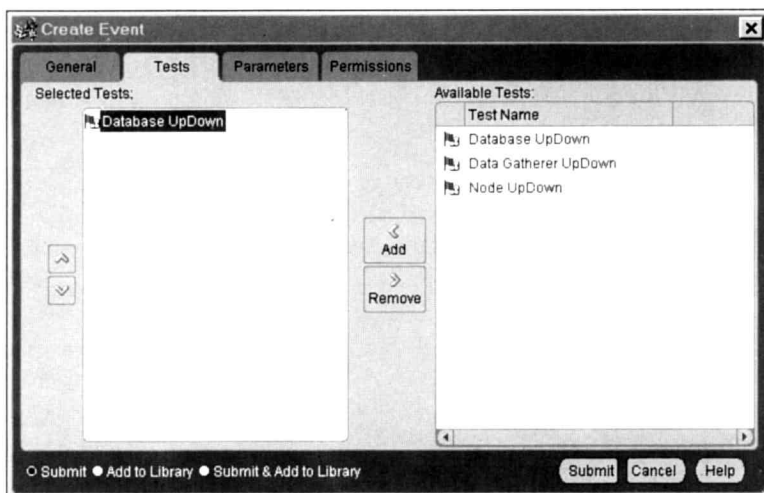


图27-6 许多预定义事件可以用于监控数据库

接着，选择Parameters选项卡，在这里为一个事件集中的每个选定事件管理配置信息。这种类型的测试不需要参数。要监控一个数据库，只需确认你已经在 General选项卡中设置了频率（Frequency）信息，该信息决定远程智能代理应该每隔多长时间检查数据库，然后按下 Submit按钮完成事件集合的设置。

该事件集目前尚未注册。通过进入企业管理器的事件控制面板并点击 Registered选项卡，你可以再次核对事件尚未注册。

注意 如果可用目标列表中没有目标，需要进入导航器并在Database Group文件夹中创建一个数据库组。数据库组构成可用目标列表。当事件集被注册时，基于责任的系统管理员定义的组简化了企业管理器系统中的通知配置。

现在EMS被配置为用于监控基于事件集配置选定的数据库并响应该事件。

类似于作业，EMS是企业管理器的一个固有组件，它打开了企业级系统管理的大门。在你花费精力全面配置企业管理器以监控事件之前，应确认该工具满足你的全部需求。对于一个纯 Oracle数据库焦点，EMS或许可以满足需求，但如果你需要一个覆盖数据库监控以外的领域，或要求对多个数据库技术进行监控的集成解决方案，你也许需要购买一个更加全面的工具。

27.4 使用数据库管理工具

虽然企业管理器导航器控制台组件提供了访问 Oracle数据库环境中各种对象的手段，细节的管理工作需要使用数据库系统管理工具来实现。管理工具的用户界面基于非常熟悉的 Windows资源管理器树型结构，这使全部工具非常直观并易于使用。当在窗口的左边选定对

象时，将在窗口右边的显示中描述对象，对于文件夹，将显示文件夹中的对象。

工具条提供了对每个应用中主要功能的快捷访问手段。当在导航树中选定某个对象时，对应的按钮便被激活，可用于在选定对象上进行操作。一些应用在工具条上有它们自己的向导，针对它们的主题领域执行某些功能。大多数数据库系统管理工具中的工具条功能包括数据库连接请求、导航树刷新、文件夹过滤器、对象创建与删除及拖放帮助。

决定如何使用数据库系统管理工具的主要因素围绕在数据库设计工具在开发过程中如何使用。如果数据库的设计是由诸如 Oracle Designer、Computer Systems Advisors Silverrun、Logic Works Erwin或Powersoft PowerDesigner等工具管理的，使用模式管理器更改产品模式中的对象类似于使用本地 DDL达不到预期目标。模式管理器并不是数据库设计工具的替代品。但是，模式管理器对于创建如产品数据库设计范围之外的临时表的对象还是有用的。在变化被引入数据库设计前，开发者使用这些工具来创建对象用于测试。

27.4.1 管理实例

数据库系统管理员使用实例管理器管理数据库的可用性和配置，还包括管理实例会话和事务。全部配置信息存储在配置文件 INIT<SID>.ORA中，可以直接从实例管理器访问这个文件（见图 27-7）。基于窗口左边的树型导航器选定的对象，该对象的会话和悬而未决的事务在窗口的右边被监控并管理。

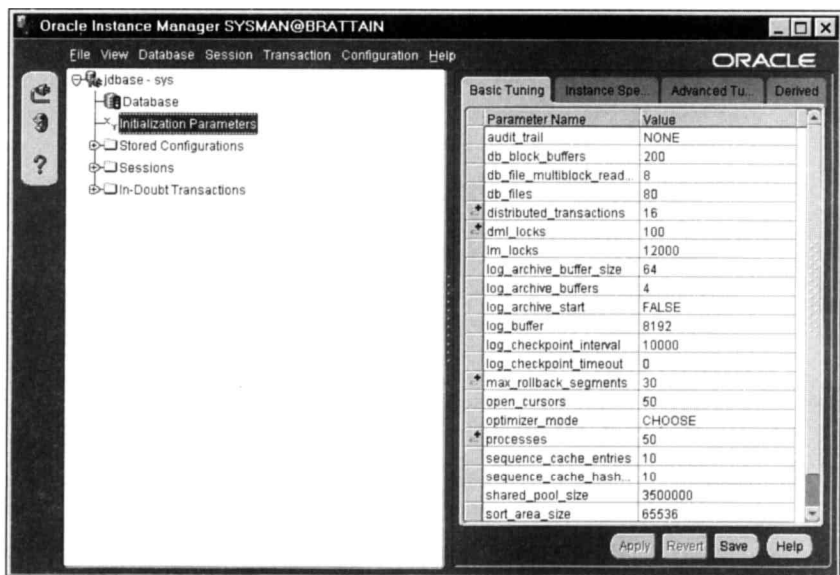


图27-7 实现实例调整参数的改变

实例管理器提供了大量有关数据库可用性管理的功能。数据库系统管理员能够关闭并启动数据库实例、安装并打开数据库和切换归档日志的开与关。可以临时打开自动归档（直到实例完成）或触发几种类型的手工归档。通过在导航窗口选定数据库对象，可以得到数据库状态的信息、它的日志记录状态和 SGA 配置信息。

提示 如果在菜单条上通过选定数据库发现数据库管理功能是变灰的且不可使用的，这是因为连接的用户没有被授权使用这些功能。如果你希望执行这些任务，作为

SYSOPER或SYSDBA用户重新连接数据库。

选择导航窗口中的 Sessions文件夹得到会话信息。另外，使用菜单条上的 Sessions菜单，能够禁止或打开实例对新会话的访问。会话既可以立即断开，也可以在当前事务完成后断开。选择In-Doubt Transactions将得到这些事务的信息，它们可以被强制提交或回滚。

实例配置信息同样由实例管理器管理。在数据库内部动态更新的参数立即起作用。其他的变化在数据库的下一次使用周期起作用。多个配置信息存储在 Windows注册表中以备访问或查看。任意配置信息都可以被输出到一个文件中或自动地传递到远程系统。

提示 存储的配置信息帮助追踪每个数据库的变化。如果所有的配置在实例管理器中改变并存储，你可以经常联机查看你的历史变化记录。变化完成的日期和做出变化的人作为配置信息名称和说明的一部分储存。将来的版本可能将在知识库中存储配置信息而不是在本地工作站的Windows注册表中存放这些信息。

27.4.2 管理模式

模式管理器或许是最为完整且应用最为广泛的数据库系统管理工具。模式管理器管理簇、约束、数据库链接、函数、索引、包、分区、权限、过程、队列、刷新组、序列、快照日志、快照、同义词、表、触发器、视图（见图 27-8）以及这些对象之间所具有的关系。

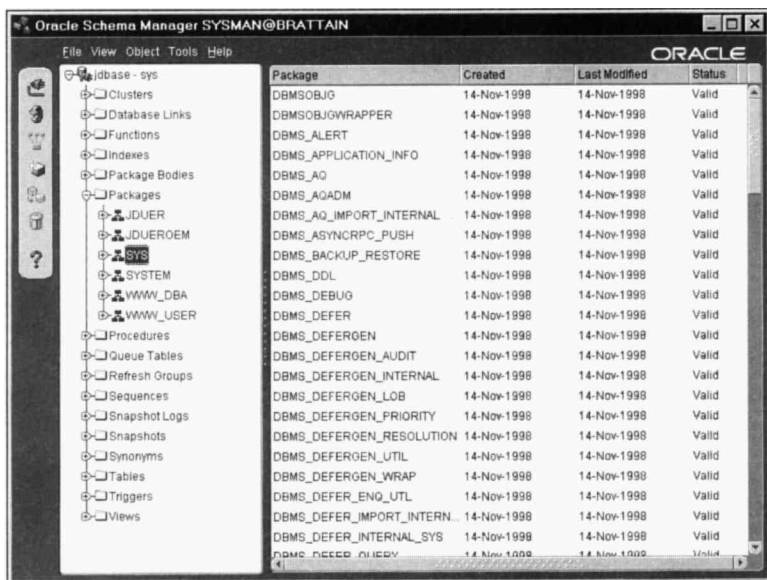


图27-8 模式管理器显示一个依赖关系视图

模式管理器提供了许多有关管理模式对象的能力。数据库系统管理员可以创建任意管理对象，甚至可以使用现有的对象作为模板。当在导航树中选定一个对象时，模式管理器可以创建或指定与选定对象有关的对象权限、同义词和索引。可以从一个选定的表中删除分区。

模式管理器功能非常强大，它是最易于使用的数据库系统管理工具之一。清单 27-1显示复制SYS.OBJ\$表和它的索引由模式管理器记录的结果。

OBJ\$_TBL.SQL的全部内容是在四个现有的对象上使用 create like操作生成的，更改模式的名称和表空间并重新指定索引的列顺序（在索引上的 create like操作将不被保留）。

清单27-1 创建OBJ\$和它的索引的DDL OBJ\$_TBL.SQL

```

REM Schema Manager recorded SQL to create a duplicate copy of SYS.OBJ$
REM and its indexes and constraints.
CREATE TABLE "JGENTRY".TEST
(OBJ# NUMBER NOT NULL, DATAOBJ# NUMBER NULL, OWNER# NUMBER NOT NULL,
NAME VARCHAR2(30) NOT NULL, NAMESPACE NUMBER NOT NULL,
SUBNAME VARCHAR2(30) NULL, TYPE# NUMBER NOT NULL, CTIME DATE NOT NULL,
MTIME DATE NOT NULL, STIME DATE NOT NULL, STATUS NUMBER NOT NULL,
REMOTEOWNER VARCHAR2(30) NULL, LINKNAME VARCHAR2(128) NULL,
FLAGS NUMBER NULL, OID$ RAW(16) NULL, SPARE1 NUMBER NULL,
SPARE2 NUMBER NULL, SPARE3 NUMBER NULL, SPARE4 VARCHAR2(1000) NULL,
SPARE5 VARCHAR2(1000) NULL,
SPARE6 DATE NULL, CHECK (OBJ# IS NOT NULL), CHECK (OWNER# IS NOT NULL),
CHECK (NAME IS NOT NULL), CHECK (NAMESPACE IS NOT NULL),
CHECK (TYPE# IS NOT NULL), CHECK (CTIME IS NOT NULL),
CHECK (MTIME IS NOT NULL), CHECK (STIME IS NOT NULL),
CHECK (STATUS IS NOT NULL)) PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
STORAGE (INITIAL 10K NEXT 100K MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0) TABLESPACE "USER_DATA"
CREATE UNIQUE INDEX "JGENTRY"."I_OBJ1" ON "JGENTRY"."TEST" ("OBJ#")
TABLESPACE "USER_DATA" PCTFREE 10 INITRANS 2 MAXTRANS 255
STORAGE (INITIAL 10K NEXT 24K MINEXTENTS 1 MAXEXTENTS 121 PCTINCREASE 50
FREELISTS 1)
CREATE UNIQUE INDEX "JGENTRY"."I_OBJ2" ON "JGENTRY"."TEST"
("OWNER#", "NAME", "NAMESPACE", "REMOTEOWNER", "LINKNAME", "SUBNAME")
TABLESPACE "USER_DATA" PCTFREE 10 INITRANS 2 MAXTRANS 255
STORAGE (INITIAL 10K NEXT 100K MINEXTENTS 1 MAXEXTENTS 2147483645
PCTINCREASE 0 FREELISTS 1)
CREATE INDEX "JGENTRY"."I_OBJ3" ON "JGENTRY"."TEST" ("OID$")
TABLESPACE "USER_DATA" PCTFREE 10 INITRANS 2 MAXTRANS 255
STORAGE (INITIAL 10K NEXT 10K MINEXTENTS 1 MAXEXTENTS 121 PCTINCREASE 50
FREELISTS 1)
EXIT;

```

27.4.3 管理安全性

安全管理器提供给数据库系统管理员在 Oracle 8i 中管理访问权限和对象权限的一种方法 (见图 27-9)。数据库系统管理员能够容易地为用户和其他角色创建、修改和删除安全对象以

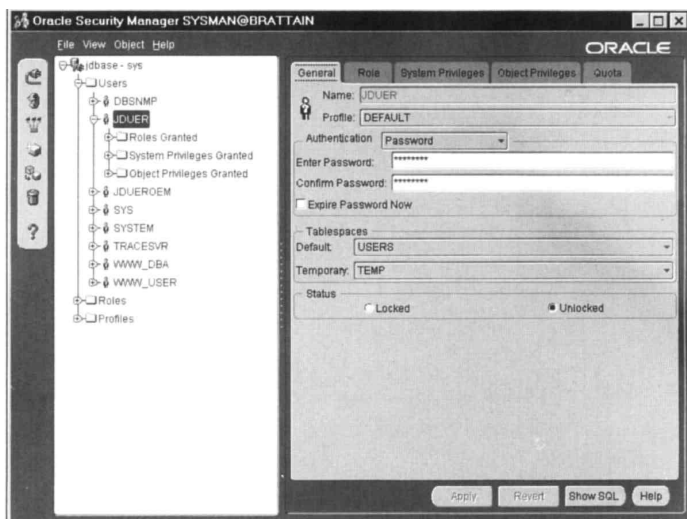


图27-9 注意位于安全管理器屏幕左边的系统管理员工具条

及相关的权限、角色、限额和环境资源文件。

安全管理器非常易于使用，以致于可以将它提供给终端用户安全管理员用于管理应用安全。这些终端用户安全管理员拥有在他们职责范围之内的对角色和对象的 ADMIN OPTION 授权。通过合理地使用角色、进行应用用户训练和使用安全管理器，可以避免使用执行该功能的自定义应用组件。应用用户喜欢直接并及时地管理属于他们自己的数据，而数据库和安全管理员可以在更有效益的工作上花费时间。

27.4.4 管理存储

存储管理器提供给数据库系统管理员在 Oracle 8i 内部管理并监控表空间、数据文件和回滚段的一种方法。数据库系统管理员能够容易地创建、修改和删除存储（见图 27-10）。存储对象的相关性在三个存储对象的任何一个的透视图中得到展示。存储管理器可以向表空间添加回滚段和数据文件，将表空间联机 and 脱机，并将它们指定为只读模式或取消只读模式。单个数据文件和回滚段也可以被联机或脱机。存储管理器还能够收缩回滚段。

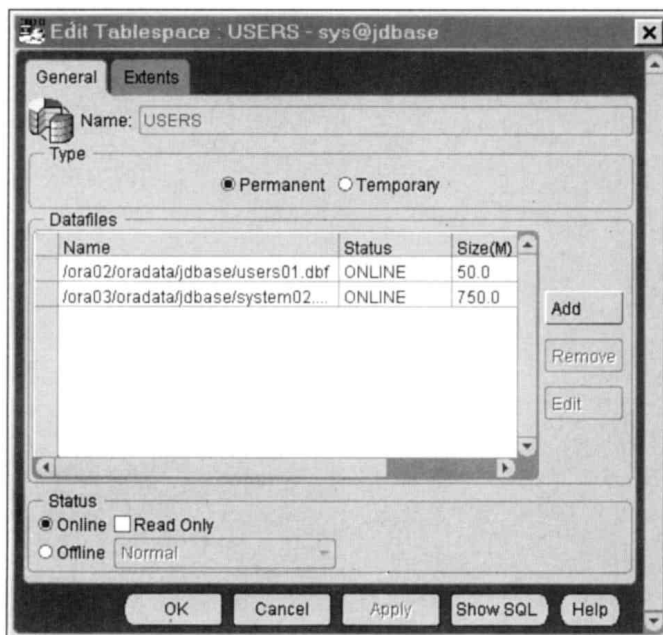


图27-10 在显示存储应用时创建一个表空间

存储应用的图形显示让数据库系统管理员一目了然地快速捕捉到存储故障并在同样的工具中对所有这些故障做出响应。

27.4.5 执行SQL

SQL工作单是适应服务管理器和 SQL*Plus 的数据库管理工具。面向数据库系统管理员，它提供了一种提交 SQL、PL/SQL、DBA 命令和 SQL 脚本的一种手段。它带有强大的命令历史记录和检索能力，为数据库系统管理员提供了一些使现在被废弃的 SQL*DBA 应用受喜爱的能力。SQL 工作单被分割成上部的输出面板和底部的输入面板两部分，两个面板都可以滚动，在输入面板输入命令，结果滚动显示在顶部的输出面板中。

输入面板工具条提供了几个实用方便的功能。工具条上的按钮用于创建新的脚本、打开脚本并保存脚本。另外一个按钮可用于执行当前打开的脚本。剩余的三个按钮显示命令历史记录，让用户可以在以前使用过的命令中滚动。可以从一个独立窗口显示的命令历史记录中将先前执行的命令拖入输入窗口中执行。一个与输出面板相关的按钮使数据库系统管理员可以输出结果保存到一个文件中。

27.4.6 管理可恢复性

备份管理器为 Oracle 8i 数据库的恢复与备份管理提供了两个子系统：操作系统备份和 Oracle 8i 恢复管理器。操作系统备份不能为 Oracle 8i 提供一个全面的解决方案（它是面向 Oracle 7 的）。Oracle 8i 恢复管理器是一个全面的恢复与备份解决方案，包括诸如基于时间点的数据库恢复、脚本和控制文件恢复等功能。一个可选择的恢复目录表提供了说明备份与恢复的元数据的一个知识库。

提示 要使用备份管理器，作业必须与一个运行在目标数据库所在的系统上的远程智能代理共同被配置和使用。

可以使用向导指导数据库系统管理员完成执行数据恢复与备份所需的步骤。作为选择，数据库系统管理员可以手工创建用于映像拷贝的备份集和作业。

27.4.7 管理数据

数据管理器（Data Manager）帮助数据库系统管理员执行导入、导出和表装载。该工具提供了一个易于使用的界面，以生成传递给 SQL*Loader 和导入/导出应用工具执行的参数文件。为了在远程系统上执行这些功能，作业必须与一个运行在目标数据库的系统上的远程智能代理一起被配置和使用。数据管理器可以在本地数据库上操作而不需要使用作业。

数据管理器中的导航树是面向对象的，由表和分区填充用户下面的两级层次。每个导出、导入和加载操作都有自己的向导，用于准备运行这些应用工具所必需的输入参数。为了使用加载功能，要提前使用应用工具准备一个加载控制文件。

数据管理器对于没有经验的新手 DBA 来说，是熟悉可以将数据移入和移出数据库的三种工具的快捷路径。在稍后讨论本地应用工具和它们的参数前，向导是一个很好的练习工具。

27.4.8 管理软件

软件管理器（Software Manager）为数据库系统管理员提供了一种方法，通过一个易于使用的中央客户接口与远程智能代理合作工作，数据库系统管理员可以在计算机网络上分布 Oracle 软件和相关的配置信息。多个软件分布服务器提供了扩展大型安装解决方案的方法。软件管理器提供了安装在目标远程系统上 Oracle Home 目录下的软件产品的清单。

软件分布过程在软件管理器内部建立。特定的系统被指定为软件分布的主机。这些主机是分布到目标系统的软件的源。基于在 Oracle 软件分布介质或在安装级区域内找到的产品安装文件，在软件分布服务器上定义软件版本。然后，基于分布服务器上的一个或多个版本组件的软件包被捆绑起来，以备分布与安装。需要应答的安装使用应答文件处理。然后，软件包被提交，用于使用作业在目标远程系统上安装，这需要使用软件管理器。

27.5 使用诊断包

企业管理器诊断包（Enterprise Manager Diagnostics）是企业管理器的一个增值组件，它提供了用于性能监控与追踪、会话监控、追踪管理和锁管理的工具。良好的数据库性能调整的要求远不只是一个基于时间点的系统信息的快照。性能管理是一种方法，不是一种工具或技术，要求每隔一段时间就对每天、每星期和每月的不同时间的主要性能指标进行访问。理论上，性能管理的结果不是基于一个即时故障的改正行动，而是基于趋向于检测一个正在进行的分析过程的预防动作。诊断包提供了满足这个目标所必需的捕捉、存储和分析性能数据的工具。

提示 注意还要花费一笔额外的费用才能得到使用企业管理器诊断包的许可。除非正当得到许可，否则你将不能安装这个组件。

27.5.1 性能监控与追踪

诊断包的核心部件是性能管理器（Performance Manager），这个工具提供给数据库系统管理员实时性能监控和记录性能特性以供日后分析的能力。一系列图形和表格形式的预定义的度量让数据库系统管理员可以很快地开始使用这个工具，而不需要预先的开发。如果性能管理器提供的标准图表不能满足要求，数据库系统管理员可以添加新的度量以满足特定的需求。

性能管理器拥有内嵌的图表用于收集和分析竞争、全局数据库统计、I/O、工作负载级别、内存和并行服务器信息。使用一条 SQL 语句可以收集的任何其他信息都是特定的分析和调整所需要的新图表的基础。

性能管理器虽然是一个综合的工具，但是易于使用。以灵活的图解细目显示的性能信息集合能够很快创建。

27.5.2 追踪数据库活动

追踪管理器是一个综合的工具，收集用于性能管理、容量计划和工作负载管理的数据。通过使用提供的 API，追踪管理器甚至可以收集来自第三方厂商和自定义应用的信息。企业管理器作业组件按时间表使用远程代理收集系统上的追踪数据。在使用追踪管理器之前，必须先配置作业。这个工具还使用它自己的知识库和一个 Oracle 专家知识库，专家知识库是从未使用过 Expert 或 Trace 的用户首次访问它时建立。

要配置追踪管理器，创建收集和发现追踪的产品非常必要。在配置时，启动收集以采集来自允许追踪的软件的数据。Oracle Trace 是一个灵活的工具，具有来自远程应用代码和追踪管理器给予的许多数据收集的能力。当前版本中的文档包含两卷共 275 页。安装要求已配置作业的企业管理器控制台、客户端的追踪和专家知识库、允许追踪的程序所在节点上的远程智能代理、远程节点上的追踪收集服务（通常与远程应用连接）和远程节点上的追踪格式化表。每个应用也许同样需要配置。例如，在产生追踪数据之前，Oracle 8i 至少必须将 ORACLE_TRACE_ENABLE 参数设定为 TRUE。配置并不是无足轻重的，不可以马马虎虎。但是，Trace 是提供充分利用企业管理器诊断包所需数据的关键。

27.5.3 监控会话

TopSessions（顶层会话）使数据库系统管理员能够监控数据库会话，并在必要的时候终

止它们，例如在会话不活动或过量使用系统资源时。这个工具可以监控所有的会话，或基于对系统特定资源（例如内存或输入/输出）的消耗只监控最活跃的会话。当分析活动时，过滤提供了将焦点缩小到感兴趣的会话的一种手段。

提示 在运行 Oracle 8i 的顶层会话之前，运行 `%ORACLE_HOME%/SYSMAN/SMTPSInn.SQL`，以确保用于执行所有功能所需要的表都在适当的位置并有适当的许可。注意，上面的脚本不能按照所示的运行，需要修改 `ORACLE_HOME` 的驱动器和名称。仍然有必要作为 SYS 用户登录到系统解释存取计划。

TopSessions 能够深入会话内部显示关于重做、排队、缓存、操作系统、并行服务器、SQL 和其他各种各样信息的详尽会话统计。检查活动的或打开的游标，显示由每个游标执行的 SQL 代码和用于执行 SQL 代码的存取计划。还有锁定信息，包括由会话持有的阻塞锁标识。

提示 TopSession 能够为某个会话指明锁，但对于深入地分析全部锁的状况，要使用的工具是锁管理器（Lock Manager）。锁管理器可以显示数据库上持有的全部锁或仅显示阻塞锁或等待锁。造成问题的会话可以直接在锁管理器中被终止。

27.5.4 使用容量计划器与锁管理器

诊断包的容量计划器用于在一个知识库中收集和存储数据库和操作系统的统计信息，供日后分析之用。这是在诊断包 2.0 版中增加的新特性。

通过定义收集视图（collection view），可指明需要收集什么类型的统计信息，并且通过使用分析视图（analysis view），容量计划器可分析并汇总这些采集数据。

有两种方式执行容量计划器：知识库（repository）方式将收集到的系统统计存储到知识库数据库供日后分析；独立（standalone）模式不需要连接到一个知识库，但在这种模式下不是所有的功能都可用。

你需要了解掌握的最后一个诊断包应用工具是锁管理器。锁管理器（也是 2.0 版的新特性）对于监控数据库锁、解决阻塞和等待状态的问题是一个了不起的工具。

使用锁管理器的先决条件是数据库中要有锁视图，这些视图是：

- V\$LOCK
- V\$SESSION
- DBA_OBJECTS

锁管理器窗口按会话标识号显示锁信息，但信息可以用不同的组合方式存储。

27.6 使用调整包

你可以花费额外费用以得到许可使用另外一个 Oracle 工具包——调整包（Tuning pack）。包含在调整包中的应用工具是表空间管理器（Tablespace Manager）、Oracle 专家（Oracle Expert）和 SQL 分析（SQL Analyze）。

通过查看存储的使用和数据库表空间的需求、数据库自身的动态性能、由数据库执行的 SQL 语句结构，这些应用工具帮助数据库系统管理员调整数据库。在本书的下面部分中，将包含每种工具的详细介绍。

27.6.1 管理表空间

表空间管理器提供了关于表空间、数据文件和回滚段存储使用的详尽信息。每块存储的用不同色彩作标记的示意图让数据库系统管理员可以形象化地看到每个数据库对象使用的存储器、使用了多少区间并且在表空间和数据文件中区间被放置在哪里。当追踪输入 / 输出竞争或追踪大型数据库加载与建立索引的过程时，这些信息是无价的。

除了图形显示，在右边的应用面板还有两个选项卡。Space Usage选项卡显示空间使用统计信息，例如行计数和平均行长度。Extent Information选项卡显示诸如选定对象的区间计数和选定区间所在的数据文件等信息。

提示 在Space Usage选项卡下的大多数信息是由ANALYZE SQL命令产生的。对于没有统计信息的对象，使用ANALYZE命令或表空间分析器向导（Tablespace Analyzer Wizard）生成统计信息。如果数据库按基于规则的优化进行调整并使用CHOOSE优化模式，生成统计信息或许会有负面的性能影响。同样，如果采用基于成本的优化并且统计信息不可用，性能也许会严重受损。

使用作业，表空间管理器应用四个向导以改变和分析存储配置。所有四个向导都向企业管理器作业时间表组件提交作业，以执行要求的任务。表空间分析器向导（Tablespace Analyzer Wizard）使用ANALYZE SQL命令分析选定的表、分区、簇和索引。表空间管理器向导（Tablespace Organizer Wizard）使用导出和导入工具整理表、分区和簇的磁盘碎片并重建索引。它还可以检测段尾没有使用的空间。磁盘碎片整理向导（Defragmentation Wizard）具有与表空间管理器同样的磁盘碎片整理能力，但由于它使用了缺省选项，使用它更快更容易。合并向导（Coalesce Wizard）将相邻的自由空间合并为一个块。对于活动的可操作的数据库这一点尤为重要，在那里对象的大小经常改变，并且临时对象经常被创建或删除。除非一个选定的对象含有相邻的自由空间，否则这个向导禁止使用。

27.6.2 使用Oracle专家

与Oracle追踪（Oracle Trace）相同，Oracle专家（Oracle Expert）是一个精心设计的产物，在当前版本中，它自己的用户指南由160页组成。Oracle专家不仅仅是一个产品，还是一种性能管理方法论的实现。其他的诊断包组件提供了信息让数据库系统管理员分析，而Oracle专家对数据采用一系列规则得出结论并为提高性能做出建议。它考虑随时间推移的开发趋势和当前的系统环境。当推出一个新的数据库版本时，要更新规则以对如何利用新的高级特性做出建议。因此，它作为一个导师，帮助数据库系统管理员更为详尽地学习有关性能管理和Oracle 8i的知识。

不幸的是，为满足日常的需求经常忽略性能调整，直到严重的性能问题影响产品数据库环境。Oracle专家提供了一种手段，自动地筛选性能数据，这些数据甚至可能会消耗一个熟练的数据库系统管理员的大部分时间。Oracle专家不能作为一个性能调整器取代数据库系统管理员，不过它确实将数据库系统管理员从筛选堆积如山的数据以发现潜在问题的现实任务中解放出来。有了Oracle专家，性能调整的重点变为如何处理它发现的问题，并使用基于数据库系统管理员的经验和知识的更多规则加强它。性能调整变得更有效率、更加高效和有益。

要启动一个Oracle专家调整会话，从菜单上选择File | New并告诉它需要分析的有关区域。

调整会话的作用范围基于实例调整、应用调整和存储结构调整的需求确定范围。

系统的环境资源文件基于工作负荷的类型（OLTP、DSS或批处理）、停机时间容限、逻辑写入速率峰值和应用信息定义。在作用范围确定以后，选择 Collect选项卡来定义需要收集何种类型与多少数据用于分析。当使用满足调整会话所必须的最小级别定义收集参数时，点击 Collect按钮以获取所需的数据。在将此步骤应用到一个产品数据库之前，请先在一个测试环境中进行练习，直到理解各种选项对系统所产生的影响并了解了执行指定动作所要消耗的时间量为止。

接下来，选择 View/Edit选项卡，获得对调整会话更为详尽的说明。在这里定义特定的调整规则和阈值。有经验的调整员或许希望基于目标数据库的环境资源文件和个人知识与经验改变百个规则中的一部分。对于有经验的调整员，这是判断 Oracle专家是否可信任的一个机会。

在 Analyze选项卡中，点击 Perform Analysis按钮开始分析过程。在分析完成后，选择 Recommendation选项卡以得到建议。如果建议看起来是合理的，使用 Implement选项卡生成实现脚本和新的参数文件。运行脚本并执行新的参数以使专家的建议起作用。如果建议看起来不正确，这通常要返回去根据目标数据库环境的需要修改一个原则。基于规则的改变产生新的建议不需要重新收集数据，只需要简单地改变规则，使之更好地反映目标数据库的环境，并再次运行分析即可。

Oracle专家提供了集中所收集到的 Oracle公司智慧的机会，并加入本地数据库管理小组的经验和知识，使其达到数据库性能管理的结构化的方法。虽然大多数的安装没有使用企业管理器的这个组件，但至少在一个只有有限的数据库调整经验的小型组织中，它或许是最有用的。

27.6.3 使用SQL分析

最后一个需要学习掌握的调整包应用工具是 SQL分析（SQL Analyze）。SQL分析是一个能够在任何有问题的SQL语句上进行性能调整的工具。

在启动SQL分析后，你可以使用 TopSQL查找并分析SQL语句，然后挑选出你认为需要调整的语句。然后，针对语句如何执行生成 EXPLAIN PLAN和统计信息。在审阅统计信息后，你就可以调整语句了。为帮助你调整语句，可以使用提示与调整向导。这些向导可以增加优化提示、设置语句所使用的表的访问路径并在语句做连结时定义连结顺序。

在你优化调整了执行不当的语句后，再次执行该语句并对比执行结果。

27.7 使用企业增值产品

Oracle提供了另外几个与企业管理器集成的增值产品。每种产品满足某些领域的特定需求，例如Web服务器管理和处理生物测定学信息（例如指纹信息）。

复制管理器对于大量地使用 Oracle 8i复制功能的环境是很有用的。Oracle针对NT系统的 Rdb提供了管理 Rdb数据库的工具，这些 Rdb数据库是 Oracle公司几年前从数字设备公司（Digital Equipment Corporation，DEC）获得的。Oracle故障安全（Fail Safe）管理在 Windows NT集群上实现的高可用性环境。Oracle生物测定学与其他硬件协同工作，使用高级网络选项（Advanced Networking Option，ANO）管理指纹鉴定。Oracle还将Web服务器管理器和安全服务器管理器作为增值应用与企业服务器集成。

使用Oracle企业管理器软件开发者工具包（Software Developer's Kit, SDK），开发者可以创建他们自己的与企业管理器集成的应用，以满足他们的环境特定需求。一些第三方厂商的应用与Oracle企业管理器集成，这些应用的功能范围从系统管理功能（例如通过 Oracle XA 连接的事务过程管理）到特定的顶层市场应用，包括计算机化医药记帐系统。还有更多的第三方厂商应用正处于开发过程中。

2.0版企业管理器中包含的新的可用包如下所示：

- 修改管理包（Change Management Pack）—— 该工具包帮助数据库系统管理员管理在多个数据库之间变化的对象定义。
- 数据库应用管理包（Management Pack for Oracle Applications）—— 使用这个工具包可以简化管理多层数据库应用环境的工作。Oracle应用管理包扩展了Oracle企业管理器的功能，提供了针对Oracle应用的监控、容量计划和故障排除功能。
- 标准管理包（Standard Management Pack）—— 该包是一组通用应用工具，用于帮助数据库系统管理员进行故障处理、追踪变化并调整数据库索引。

27.8 小结

在本章，你学习了Oracle企业管理器的所有内容。对组成企业管理器的每个组件都做了介绍，例如模式管理器、安全管理器和表空间管理器。你还学习了可选的软件包提供的附加功能，例如诊断包和调整包。

通过最大限度地使用这些功能，数据库系统管理员能够在夜晚甜美地熟睡，因为他知道数据库此时正由一位合格的软件卫兵严密地监视着。

第28章 PL/SQL基础

本章要点：

- 理解PL/SQL
- 理解PL/SQL引擎
- 在工具箱里增加PL/SQL
- 开始学习PL/SQL
- 语言教程
- Oracle 8i特性

28.1 理解PL/SQL

PL/SQL是ANSI标准SQL的Oracle版本的过程化语言的扩展。SQL是非过程化语言，程序员只需要说明执行什么工作，至于如何执行则留给 Oracle服务器的 SQL优化器。相反，PL/SQL，像第三代（3GL）过程化语言一样，要求一步步指导来确定下一步做什么。

类似其他的工业标准语言，PL/SQL提供变量声明、赋值、条件测试与分支以及迭代的语言元素。正如C或Pascal一样，它侧重于面向块。PL/SQL服从严格的作用域原则，提供参数化子程序结构，并且类似Ada语言，有一个称为包的类似于容器的特性，由程序员随意决定隐藏或显示数据和函数。它是强类(型)语言，在编译和运行时捕捉数据类型不符的错误。可以进行显式的和隐式的数据类型转换，支持复杂的用户自定义数据结构，子程序可以被重载，以创建一个灵活的、模块化的环境。

另外，因为它是过程化封装的 SQL，该语言能够很好地与 SQL结合使用。某些语言特性让它能够与Oracle RDBMS交互操作，执行集合与单条记录操作。你所了解用于编写 SQL程序的知识越多，你所设计的PL/SQL程序就越好。

PL/SQL提供了一种称为异常事件处理（Exception Handling）的特性，同步地处理事务处理期间可能发生的错误和类似事件。你将学习如何在 PL/SQL代码中嵌入意外事件处理程序以漂亮地处理错误状态，例如NO_DATA_FOUND或DUP VAL ON INDEX。

PL/SQL不是面向对象的语言，它具有在其他的语言，例如 Pascal和Ada语言中见到的一些特性。如果熟悉 Pascal语言的语法，学习 PL/SQL不会有任何困难。不像 C和Pascal语言，PL/SQL不支持指针的使用。PL/SQL是主要的后端开发工具，在那里它严格地与数据库表和其他的数据库对象进行交互操作。通过使用所提供的数据库工具包，处理 PL/SQL与操作系统以及外部软件组件的交互操作。

PL/SQL在所有的Oracle平台上是100%可移植的。因为它的数据类型基于数据库服务器上的数据类型，该语言与机器完全无关。不需要学习 UNIX、Windows NT、NetWare等等各种各样的特性。一个PL/SQL程序可以不需要任何修改，在所有的 Oracle服务器上编译并运行。

这种可移植性也扩展到第三代（3GL）编程语言。通过 Oracle提供的预编译器，PL/SQL针对各种语言例如 C和COBOL语言提供了一个标准接口。预编译器支持内嵌 SQL的ANSI标

准。

使用Oracle 8i, Oracle将重点集中到RDBMS与国际互联网 (Internet) 的结合。除了编写PL/SQL应用, 让世界上的所有浏览器能够通过 Oracle应用服务器进行访问的能力外, Oracle8i还增加了直接从PL/SQL过程中发出HTTP请求的能力。Oracle还将大量的工作放到存储在数据库中的PL/SQL和Java语言的互用性上, 以使它们相互之间可以无缝调用。

28.2 理解PL/SQL引擎

在把PL/SQL看作一种语言之前, 需要在执行环境中理解它。

28.2.1 适合客户/服务器环境

在一个客户/服务器配置中, 真正的瓶颈通常是网络。当几百个用户通过编译的 C、C++、Delphi或COBOL程序连接到Oracle服务器时, 将得到一个非常缓慢的网络系统。解决方案是将复杂的程序段, 尤其是那些执行重复的或相关的 SQL语句, 合并成一个PL/SQL块。这些块可以被嵌入到一个OCI (Oracle调用接口, Oracle Call Interface) 程序中, 或通过以下方法更为有效地执行: 将它们转移到数据库内部作为存储的函数、过程或包。图 28-1显示了客户应用与Oracle服务器之间典型的交互作用。

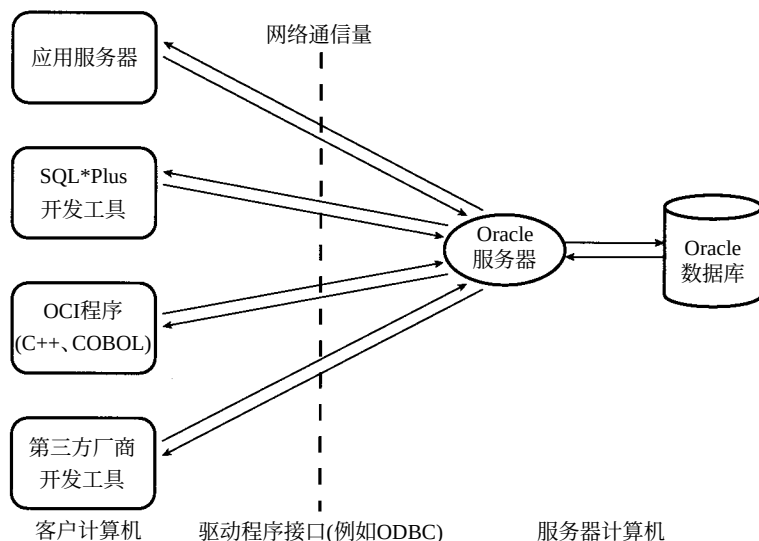


图28-1 一个典型的客户/服务器环境

PL/SQL程序由PL/SQL引擎执行, PL/SQL引擎是数据库服务器的一部分。图 28-2表明一个PL/SQL块在内部是如何处理的。

不论使用哪种工具, 例如 Oracle SQL*Plus, 该工具必须向Oracle服务器提交PL/SQL源代码, PL/SQL引擎扫描、分析并编译代码, 然后编译后的代码预备执行。在执行期间, 将所有的SQL语句传递给SQL语句执行器 (SQL Statement Executor) 组件执行。SQL语句执行器执行SQL或DML语句。PL/SQL引擎可以使用由查询提取的数据集进行更进一步的处理。

相对于逐条发送一组 SQL语句, 使用PL/SQL块执行一组SQL语句的一个好处是降低了网络传输量, 图28-3表明了这种思想。

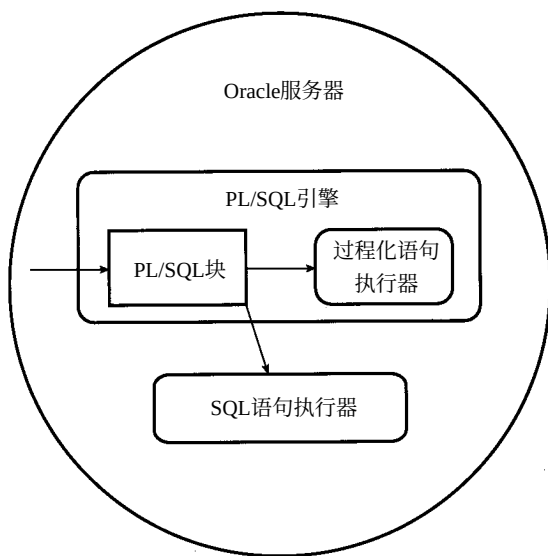


图28-2 PL/SQL引擎是Oracle数据库服务器的一个组件

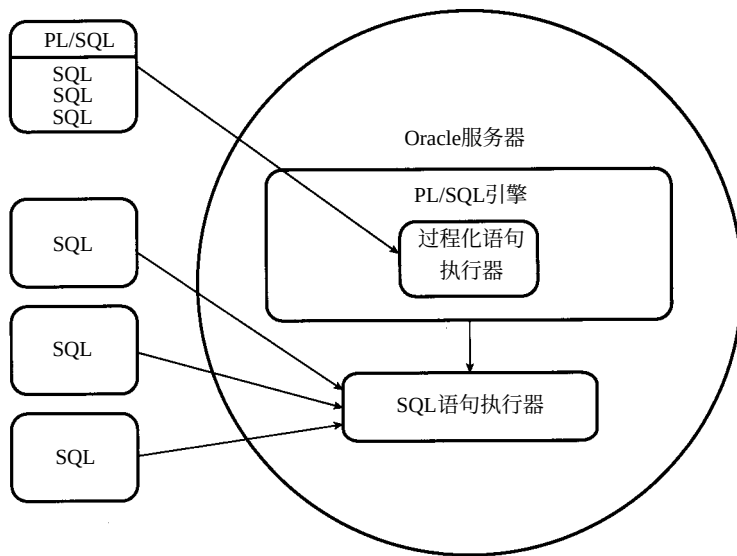


图28-3 将几个SQL语句组合为一个PL/SQL块降低了网络传输量

单独这样做，可以明显地提高应用的性能。另外，SQL/DML语句可以当作一个事务对待。如果整个事务成功，那么对数据库的全部改动会被提交。如果事务的任何部分失败，整个事务会被回滚。因为PL/SQL块中可以包含复杂的逻辑，因此在服务器上执行，客户程序的大小和复杂程度降低了。

1. 执行存储子程序

一个更深入的改进包括在数据库内部存储编译的命名的PL/SQL块。PL/SQL块在本章中统称为存储子程序或称为子程序。“命名”只是意味着子程序包含代码与名称，就像C或Pascal子程序。图28-4显示PL/SQL引擎如何调用存储子程序。

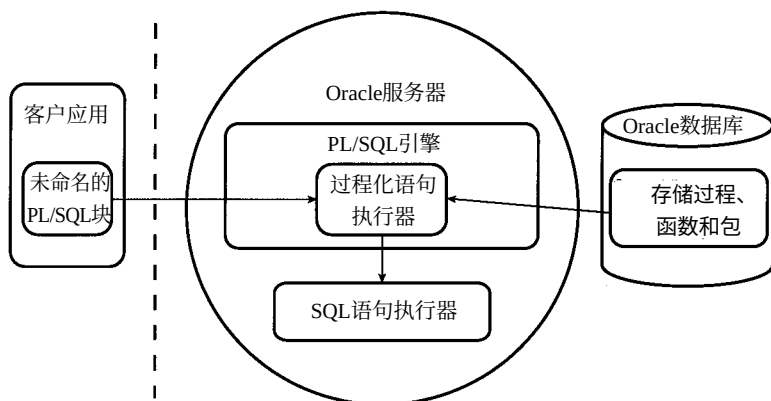


图28-4 PL/SQL引擎运行存储子程序

这些子程序可以执行复杂的逻辑和错误处理。在一个客户应用中嵌入的一个简单的匿名或未命名的块（没有使用名字标记的 PL/SQL 代码块）能够激活这些子程序。这种能力通常称为远程过程调用（Remote Procedure Call, RPC）。子程序还可以调用其他子程序。因为这些子程序已经被编译并被开发者很好地调整过，它们提供了显著的性能改进，并通过为其他应用或模块提供可重复使用的块减少了应用开发。

2. 系统全局区内的共享 SQL 区

系统全局区（System Global Area, SGA）是操作系统为 Oracle 服务器分配的一大块内存区域。在这个内存区域中，服务器维护表数据、游标、用户本地变量和其他各式各样项目的本地拷贝。

当编译任意 PL/SQL 程序时，不管是命名块还是未命名块的代码，源代码和对象代码都被高速缓存在共享 SQL 区中。分配给一个 PL/SQL 块的空间叫做一个游标。服务器使用最近最少使用算法，在共享 SQL 区保存缓存的程序直到它过期失效。在 PL/SQL 块中的任何 SQL 语句都被给予各自的共享 SQL 区。当一个命名子程序被编译时，它的源代码还被保存到数据字典中。

包含在一个子程序中的代码是可重入的，也就是说，它在所有连接的用户间是共享的。当提交一个未命名的 PL/SQL 块给服务器执行时，服务器通过比较源代码文本，决定在高速缓存中是否有该程序块。如果代码文本字符之间（包括大小写）完全相同，那么将执行缓存的被编译的代码。对于 SQL 语句这点也适用，如果查询文本完全相同，能够简单执行缓存的分析代码。否则，必须首先分析新的语句。通过共享可执行代码，一个基于服务器的应用能够真正地实现内存节省，这在拥有数百个连接用户时尤其重要。

3. 私有 SQL 区

如果几个用户正在执行相同的代码块，服务器如何分开保存它们各自的数据呢？每个用户的会话将得到一个私有 SQL 区。这块内存含有子程序中包含的变量数据的私有拷贝。还要为 PL/SQL 块中的任意 SQL 语句分配一个私有 SQL 区。图28-5显示这种模式。

一个存储子程序首次被引用时，必须将它从数据库中加载到 SGA 区。一旦加载后，它对于每个用户都是可用的。只要它继续被任意用户调用（不必是第一个调用它的人），该子程序将继续保留在内存中。包的工作方式与之相同，只是包中可能包含一整套子程序。最初的加载过程可能会花费较长的时间，但是现在所有这些子程序都是可用的，不再需要与磁盘打交

道。当预期激活一套子程序时，最好一次将它们全部加载。包提供了这种机制。

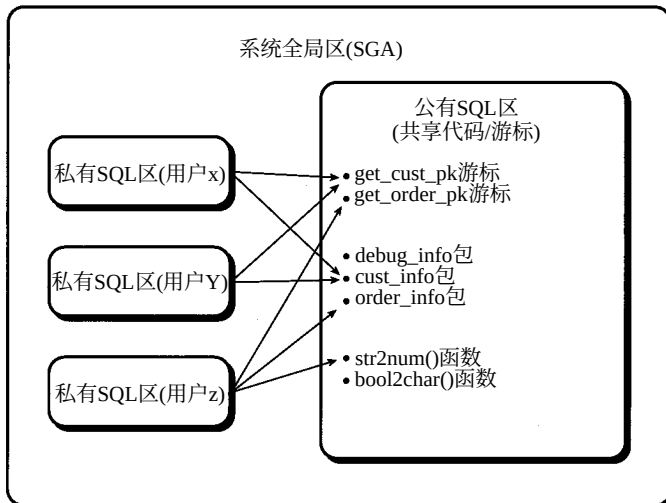


图28-5 在SGA中的共享和私有SQL区

28.2.2 适合客户环境

PL/SQL引擎还可以嵌入到某些 Oracle 工具中，例如 SQL*Forms 4.5和其他的 Oracle 开发者开发工具，主要的好处是使用熟悉的语言编程。客户程序可以执行在本地 PL/SQL 块中包含的计算，并能够向服务器发送 SQL 命令或激活存储的 PL/SQL 部分。另外，通过直接在数据库服务器上处理复杂的逻辑部分，PL/SQL 引擎支持代码重用并简化了客户程序。

28.2.3 对比服务器端与客户端开发

开发者必须有意识地决定在服务器上隐藏多少复杂性，以及在客户程序中保留多少复杂性。经过几年客户/服务器应用的开发，作者建议几条需要遵循的指导原则：

- 减少网络传输量。将一些功能放到服务器上，网络在任何客户/服务器应用中是典型的瓶颈。
- 开发标准代码。标准代码可以在新应用中或当添加到现有应用中时再次使用。这种方法节省了开发努力。
- 力争低偶联与高内聚性。当一个模块依赖于另外一个模块中的特定内容时，例如当使用全局变量而不使用参数传递时，会出现偶联。内聚性意味着放置相似的元素到相同的位置，例如将全部数学函数捆绑为一个数学库。
- 隐藏实现的细节。这样会减少客户程序的复杂性并拆分实现的功能。拆分是指消除或避免模块之间的依赖。例如，当实现自顶向下的设计时，将实现细节推动到一个较低的层次，隐藏实际的“解释作法”。
- 以普通方式编写模块。一个模块越特殊，重用的可能就越小。查找模型和公共特性。但这并不意味着将几个不同的行为塞入一个模块中。
- 以一致的、集中的方式处理商务原则。这使它们在将来的工程项目中更为直观和可重用。如果你的商务原则经常变化并且跨越多个客户程序模块，你必须到处定位和修改

它们——一个既无效率又不便于维护的过程。

提示 克莱美格可维护性原则 (Clamage's Rule of Maintainability)：一个高度可维护程序模块是指当对模块的需求改变时，高度可维护性模块需要少量修改或根本不需要修改。一个不可维护程序是指程序需要做实质的修改以加入新的需求。上述原则会帮助指导朝着建立高度可维护性软件的目标前进。

使用存储子程序实现这些原则。通过提供存储子程序的标准库，多个和将来的客户应用可以利用较早完成的开发。例如，商务原则可以作为存储子程序得以实现，如果一条原则发生改变，例如计算一个销售佣金的原则发生变化，仅仅需要修改并重新编译适当的存储程序。倘若存储子程序的接口保持不变，全部相关的客户程序不需要改动或甚至不需要重新编译。

通过允许子程序的名称被重载，包是提高子程序的通用性的一种好方法。重载简单地说是给一组代码相同的子程序名称，通常在参数表中具有不同的数据类型或不同数目的参数。例如，加运算符在几乎所有的计算机语言中被重载，以处理整数和浮点数运算。

28.3 在工具箱里增加PL/SQL

稍停几分钟，考虑一下在日常任务中如何使用 PL/SQL。你或许没有被卷入到应用开发工作中。作为一个数据库系统管理员，必须对数据库进行持续的维护。可以使用 PL/SQL让日常工作变得较为容易。

28.3.1 加强SQL脚本

事实上，一些 SQL*Plus脚本能够生成其他的脚本，例如列出表空间中每个表的记录数量。而且，这些脚本可以用 PL/SQL重写、编译并保存到数据库中，因此运行速度非常快，而且不必将中间脚本写入文件。

编写这些类型的脚本也许有些困难；一个 PL/SQL脚本更为简明易懂并且易于书写和维护。另外，一个已编译的 PL/SQL程序能够很容易地在数据库系统管理员之间共享（尤其当你在一个“24x7”商店中，24是指每天24小时，7是指每周7天，24x7意即连续不间断交易的商店）。不需要查找硬盘以找出脚本，每个人总能知道它们在哪里及如何运行它们。

存储子程序可以接收参数，这使它们非常灵活。你甚至可以为它们编写 SQL*Plus前端程序，收集用户输入数值并传入这些数值，而不必知道参数的顺序和类型。

28.3.2 简化数据库管理

或许有一套很好的 SQL脚本，为你提供运转着的数据库的信息。但在理解 PL/SQL后，你会希望得到一套全新的程序，为你提供有关数据库性能、存储、用户负载、锁等等附加信息。PL/SQL消除了普通SQL脚本的约束和限制。

通过使用提供的工具包按照设定的间隔运行它们，可以自动执行许多任务。可以将系统或应用数据发送给外部的性能监控器（可能用 C++或Delphi语言编写），并且使用它们强大的报表和图形功能实时显示这些信息。

有些工具，例如 Visual Basic很容易使用。加上 PL/SQL提供的附加能力，可以开发可视化工具以简化任务，例如运行 EXPLAIN PLAN和以图形方式查看表的索引和约束。几个可扩展

的第三方厂商工具就是采用这种方式开发的。

28.3.3 以较少的争论得到较好的信息

可以编写使用数据字典的 PL/SQL 程序去显示任意指定表的详尽索引信息，即使该表在另外一个模式中。如果你不是作为 SYS 或 SYSTEM 用户登录的，你需要一些数据字典的信息，并且不希望进行重连接，这样做非常好。这对于开发者尤为重要，他们经常需要存取这种类型的信息。可以使用 PL/SQL 逆向工程设计任意数据库对象，再建一个语法纠正语句以重建该对象。这样一来，每次改变一个表或授权一个新的权限时，可以解脱手工更新脚本带来的麻烦。

如果编写一条大且复杂的 SQL 语句有困难，可以将它分割成较小的部分，让它们在一个 PL/SQL 脚本中一起运行。这样做胜过尝试连接七个以上的表，连接七个以上的表会让 SQL 执行器陷入困境。通过主表单独查询，可以显著改善多表连结。还有，当知道 PL/SQL 脚本内部的重用代码已工作正常时，验证脚本的输出结果很容易。

28.3.4 设计更好的数据库应用

可以使用存储过程和包中的 PL/SQL 建立更好的应用。设计和创建可重用的 PL/SQL 模块的共同努力可以得到长远的利益。通过深谋远虑地设计和计划，可以做到：

- 当创建下一个应用时，可以借鉴一个应用已取得的成果。
- 在几个相关的应用之间分摊开发费用。
- 提供对底层表与其他对象的一致性接口
- 简化并优化数据存取。
- 降低应用维护和展开费用。
- 实施编码标准化。

使用 PL/SQL 是简单而有趣的。与语言相比，它只需要相当短的学习过程。通过试验可判断哪些能够很好地工作或哪些根本就不能工作。如果按照本节详述的步骤去做，只需要几天就能够工作，在短短的几个月内，你就会完全掌握 PL/SQL，取得一些高级的成果。

28.4 开始学习 PL/SQL

本节详述在操作 PL/SQL 前，数据库系统管理员必须要做的工作。对工作所在的应用环境的体系结构也将做一探讨。

在准备编写 PL/SQL 前，你或你的数据库系统管理员首先必须做下列工作：

- 授予你（也就是你的 Oracle 帐户名）CREATE PROCEDURE 权限，让你可以在自己的模式中创建子程序。对于新的应用，你或许希望创建一个特殊的用户名和模式（一个特别的表空间）。
- 在可能编写 PL/SQL 程序的任意模式中的数据库对象上（例如，表和序列），直接授予你（而不是通过角色）SELECT、INSERT、UPDATE 或 DELETE 权限。一个存储子程序或包只能引用子程序的拥有者（你）有直接存取权限的对象（不是通过角色）。
- 确认已经编译适当的 Oracle 提供的包，并且对它们拥有 EXECUTE 权限。你可以在目录“<Oracle_HOME>/rdbms<version>/admin/”下找到它们，其中 Oracle_HOME 是 Oracle 数据库的安装目录（该名字显示在 UNIX 命令解释程序变量中或在 Windows 的注册表

中), <version>是数据库的版本。包脚本的名字有通用的形式 ‘ dbms*.sql ’。例如, 在笔者的Windows NT机器上, dbmssql.sql位于 ‘ c:\oraNT\rdbs80\admin ’ 下。

- 确认有足够的存储空间可以存放存储子程序和包。这些对象在 SYSTEM表空间中产生额外的数据字典项。数据库系统管理员可以为你创建一个缺省的表空间, 用于存储你自己的表和索引的本地副本, 让你不会给产品对象添乱。
- 确认系统全局区 (System Global Area, SGA) 中的共享和私有 SQL区足够大, 可以处理运行PL/SQL脚本和子程序产生的预期负载。
- 确认对SQL*Plus应用或一个合适的第三方工具拥有使用权限。

一旦这些准备工作各就各位, 就准备开始开发 PL/SQL程序吧!

28.4.1 理解事情的模式

按抽象术语说, 模式是相关数据库对象的逻辑集合, 例如表、索引和视图。模式是开发者眼睛看到的感兴趣的数据库对象的视图, 就像实体关系图表 (Entity-Relationship Diagram, ERD) 上所显示的。

在一个Oracle数据库中, 模式是存储在一个或多个表空间中对象的逻辑集合。因为模式是逻辑集合, 几个应用可以共享模式, 或跨越模式的界限进入其他模式。模式对象的关系型组织严格按照数据库系统管理员和开发者看到的那样, 这正是为什么像实体关系图这样的“路标”对于任何数据库相关的软件开发如此关键。一个表空间是映射磁盘上一个或多个物理文件的逻辑存储区域。图 28-6显示了这种组织。

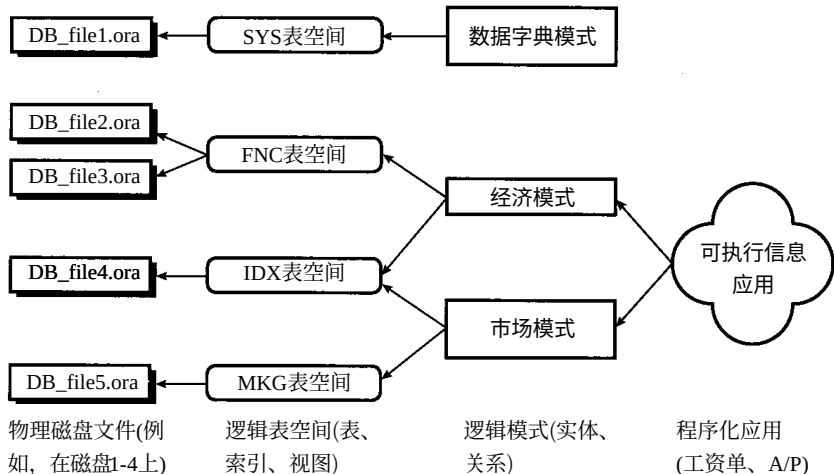


图28-6 应用、模式、表空间和数据库文件被分层组织管理

通常一个模式精确地只有一个拥有者(也就是说, 一个 Oracle用户帐户)负责创建、修改和删除这些对象。这个拥有者可以向其他用户授予存取权限, 既可以直接授权也可以通过使用角色授权。如果你不是这些对象的拥有者, 通过使用圆点符号指明它们在哪一个表空间来限定对这些对象的指引, 就像 `SELECT CUST_NAME FROM MKG.LEADS`。

管理PL/SQL代码

当PL/SQL代码在你自己的开发模式中被彻底测试和验证后, 存储 PL/SQL程序极有可能变为由模式拥有者管理的对象。在一个拥有众多开发者的复杂的开发环境中, 决定谁是源代码

的拥有者和在哪里存放源代码的问题变得十分重要。当模块被开发时，模块处于不断变动的状态。开发者可能会对一个私有副本做出重要改动，一些模块可能在后来被取消。强烈推荐使用代码管理系统（Code Management System, CMS）控制源代码的一个正式版本。当彻底测试一个模块并可用于产品时，将新版本存放在版本管理系统中，模式的拥有者可以编译这个新版本。从头开始的最快的方法是放弃追踪哪个拷贝是最好的版本。在长时间的运行中，只为不同的版本保存不同的子路径是一个很坏的编程管理方法。

在模式和表空间之间没有对应性，一个或多个模式可能存储在一个表空间中，或者也可能被分割存储于不同的表空间。实际上，将模式与文档集合存储在它自己的表空间组内是一个好主意。例如，将表存储在一个表空间中，而将索引存储在另外一个表空间中。概念上。在同一个模式中包括两个表空间。

提示 PL/SQL的开发者应该拥有他们自己的缺省表空间，以便于在上面做试验，最好选择一个非产品数据库作为试验环境。你可以创建你自己的表和索引，并用它来做试验。这样，如果你不小心毁坏一个产品表的一个临时、本地副本，谁会在意呢？否则的话会导致你事业的意外终结（最少你会丢掉现有的工作）。另外，所有编译的存储程序和包被严格地定位在你的模式内部（与属于其他模式的子程序一起存放在数据字典中），因此你可以在一个安全的开发环境中对产品程序代码做试验性的改动。

注意 对模式使用的告诫是要注意对象的引用。如果对象在缺省模式中，不需要使用模式的名称来限定引用。否则，必须要限定它，以便于能够正确地找到它。可以对 PL/SQL代码中没有限定的对象使用同义词，以提供限定。

出于实际考虑，一些应用可以容易地跨越模式的界限，但是核心应用可能集中在一个模式上。例如，或许已经开发出一个市场应用以使该部门自动化。后来上级管理部门要对整个企业有一个高层的观察，需要一个新应用从几个模式中获得数据。你在一个应用中完成的工作可能在其他工作中被再次使用，所以要注意让 PL/SQL程序模块有足够的通用性，使它们可以在多个环境中使用。

28.4.2 基本PL/SQL开发环境

使用你喜欢的文本编辑器和 Oracle提供的 SQL*Plus应用，或几个很好的第三方开发工具包，就可以开发 PL/SQL程序代码。如果在 UNIX环境中，启动两个会话是一个好主意——一个用来运行文本编辑器，例如 vi或emacs，另外一个会话运行 SQL*Plus。如果运行的是 Windows环境，可以使用记事本（或一个真正的程序员编辑器）并启动一个 SQL*Plus会话。

警告 在Windows 3.x环境下，在SQL*Plus中运行任何SQL或PL/SQL程序期间，你的整个Windows环境被有效地锁定。在SQL*Plus中执行程序之前，一定要将改动的文本保存。如果发现需要终止一个长时间执行的查询，知道没有丢失任何工作的危险，会感觉好一些。

Oracle慎重地提供了一些标准的运行库，能够执行某些重要的动作，例如标准输入/输出、计时、活动会话之间的数据交换、创建动态 SQL以及其他复杂的底层操作。当程序需要增加复杂性时，可以经常使用这些包。

28.4.3 存取数据字典

数据字典通过不同的视图呈现给不同的用户。这些视图拥有带有不同前缀的相同名称。这些视图可以被划分为以下几类（没有特定顺序）：

- 常规类和对象信息。
- 对象的存储信息，例如表空间、段和区间。
- 过程、包、触发器和它们的源代码。
- 表、列、簇和索引。
- 视图、同义词和序列。
- 用户、角色、环境资源文件和权限。
- 锁和审计信息。
- 性能信息。
- 分布式数据库信息。
- 导出信息。

对于一个授权的数据库系统管理员用户，最适合使用的视图被前缀了 `DBA_`。这些视图提供了所有模式中每个数据库对象的有关信息，通常由模式的拥有者管理。对于所有的其他用户，可以使用 `ALL_` 视图和 `USER_` 视图。另外，性能视图（以 `V$` 开头）是公共可用的。

1. 自己模式中的对象

`USER_` 视图只显示由当前连接用户所拥有的数据库对象。如果创建了它，可以发现它会在此列出。例如，视图 `USER_SOURCE` 包含存储子程序和包的源代码。

2. 其他模式中的对象

`ALL_` 视图显示所拥有的数据库对象或被授权存取的其他模式中的数据库对象。在后面的案例中，可以直接授予或通过角色授予这些权限。但是，在 `PL/SQL` 代码中只能访问那些使用直接方式授权给你的对象。例如，`ALL_TABLES` 含有全部表，加上被授权存取的其他任何用户的表。

例如，视图 `ALL_SOURCE` 包含全部源代码，还有其他用户向你授予 `EXECUTE` 权限的存储包的头。这个细小差别的原因是：只需要知道如何正确地调用其他用户封装的子程序，不能以任何方式修改它们，所以不需要查看实现细节。

当执行他人拥有的子程序时，程序通常以他人的权限被执行，而不是使用你的权限来执行。但是在 `Oracle8i` 中，开发者拥有这样的选择，他可以指定子程序使用“调用者的权限”而不是“定义者的权限”来执行。

3. 数据库系统管理员模式中的对象

数据库系统管理员是享有特权的，他可以存取数据字典中所有模式里的每个数据库对象。存储子程序是将这个信息展示给全体用户的合适方式，而不用给他们任何数据库系统管理员权限——当然，假设数据库系统管理员编译了这些子程序。例如，视图 `DBA_OBJECTS` 包含所有对象类型（例如表、索引、序列等等）的每个数据库对象（包括那些由 `SYSTEM` 和 `SYS` 所拥有的）的有关信息。通过编写并编译少量简单的 `PL/SQL` 存储过程，数据库系统管理员可以采用易于使用的方式，让那些需要该信息的用户使用它。

28.5 语言教程

学习掌握一门新语言的最佳方式是投身其中！后面的部分以指导方式介绍 `PL/SQL`，从基

础开始，很快地串讲不同的语言特性。如果熟悉任何计算机语言像 Pacal、FORTRAN、C或 COBOL，学习过程中不会有任何困难。我们强烈建议试验这里所介绍的和附带 CD中所提供的程序代码，并照此编程。

马克吐温曾经说过：“吃一堑，长一智”。在本节，听从他的建议，犯许多错误，然后从中吸引教训。要查看常见的编译和运行错误，以及服务器返回的（通常是简明的）错误信息。有意运行一些这里或那里有问题的代码，目的是让你有兴趣去调试它。这样，当独自处理问题并且遇到一个错误消息时，就不会感到无所适从。

28.5.1 编码规定

PL/SQL不区分大小写，所有的文本都被转换为大写形式（除了那些在文字串中的文字）。所以，你不可以使用大小写来区分变量和其他用户定义的元素。出于程序可读性原因，为了区分源代码中的元素，本节中的全部代码都被标准化，使其遵从一些简单的语法规则。

- 所有保留字、Oracle提供的包、内建函数、用户定义常量、Oracle和用户定义的数据类型以及模式名称都采用大写。
- 用户定义的游标、变量、包、Oracle和用户定义的子程序、表和列的名称都采用小写。
- 参数表变量以大写字母开头（例如，Pname中P代表参数）。

可以遵从这些规定或其他你乐于接受的任何规定，只是要一致。本节中的程序代码严格按照以下方式书写：带有大量的内部程序说明；仔细地设定行缩进（通常一次为两个空格，不使用TAB键）；使用空格以更清晰地对齐语句并且使每行的文本在 80个字符以内。

PL/SQL是一种自由形态的文本语言，任意数量的空白，例如空格、新行和跳格，起着分隔符的作用或被忽略。在一行上可以有一条或多条语句，一条语句也可以被拆分为几行。通常，语句以分号（；）结束。对于一行拥有的字符数量和一个模块拥有的语句行数没有限制。但是，源代码模块总的大小依赖于操作系统的限制条件。

警告 Oracle服务器的早期版本依赖于操作系统，对 PL/SQL模块的大小有限制。在 NetWare 3.x上，模块的大小被限制在32KB以内。对于多数UNIX和Windows NT系统，模块的大小被限制在64KB以内。违反这个限制，将导致数据库服务器或服务器机器自身崩溃。这个限制在Oracle8i中被取消了。

28.5.2 特殊字符

一些字符在PL/SQL程序中有特殊的含义。大多数是显而易见的，例如数学符号和关系运算符。其他的字符，例如联合与主变量指示器（Association and Host Variable Indicator），就不那么明显了。在后面的章节中将对这些字符做一探讨。表 28-1列出了这些字符（有时候是成对的）和一个简要的说明。

注意，有几种方式可以用来在 PL/SQL中表示“不等于”。出于一致性原因，本教程中从头至尾使用一种形式（!=）。以防你感到奇怪，模运算由内建函数处理。

注意 注释不能嵌套。单行注释后面的任何语句都被忽略。多行注释标识符之间输入的任何语句行也被忽略。

表28-1 按类型组织的特殊字符

类 型	字 符	说 明
数学运算符	+	加和一元正
	-	减和一元负
	*	乘
	/	除
	**	乘幂
关系运算符（在布尔表达式中使用）	=	相等
	<	小于
	>	大于
	<>	不等于
	!=	不等于（备用）
	~=	不等于（备用）
	^=	不等于（备用）
	<=	小于等于
	>=	大于等于
表达式和列表（在语句、数据类型声明、参数列表声明、变量和表引用中使用）	:=	赋值
	(	开始列表或子表达式
	)	结束列表或子表达式
	,	分隔列表项（在参数列表中时）
	..	范围运算符（在FOR-IN循环中使用）
		字符串连接
	=>	关联（在参数表中使用）
	;	语句结束
	%	游表属性或对象类型
	.	限定引用的成员说明符
	@	远程数据库指示符
	'	开始/结束字符串
	"	开始/结束引用标识符
	:	主变量指示器
	&	捆绑变量指示器
注意和标签	—	单行注意
	/*	开始多行注意
	*/	结束多行注意
	<<	开始标签
	>>	结束标签

28.5.3 PL/SQL的块结构

PL/SQL块是基本的编程结构。在块中编程有助于自顶向下、结构化的模块性和简明合理的组织。

一个未命名PL/SQL块含有三个部分：块声明、块体部分以及可选的意外处理部分：

```
DECLARE
  -- declarations
BEGIN
  -- executable code
EXCEPTION
```



```
-- exception handlers
END;
```

事实上，块声明部分也是可选的，但是，如果没有声明变量，几乎什么也做不成！所有用户定义的变量、常量、数据类型、游标、函数和过程都在声明部分定义。如果不需要进行任何定义，可以省略这个部分。

例如，考虑许多编程语言教材中常见的“世界，你好！”示例（见程序清单 28-1）。

注意 如果希望详细了解SET语句的有关信息，参阅第26章。

清单28-1 hello.sql —— “世界，你好！”程序

```
SET SERVEROUTPUT ON
BEGIN
  DBMS_OUTPUT.enable;
  DBMS_OUTPUT.put_line('Hello, World!');
END;
/
```

第一行告诉SQL*Plus将服务器所返回的写出来。

第二行和第五行提供当前块的范围。

第三行打开输出机制（这在7.3版和更高版本中是可选的）。

第四行打印字符串“Hello, World！（世界，你好！）”。

第六行执行这个未命名的基础PL/SQL块。

服务器响应如下：

```
Hello, World!
PL/SQL procedure successfully completed.
SQL>
```

在过程结束前，屏幕上不会有任何返回信息，这是因为服务器正在处理 PL/SQL 块（很有可能通过网络），只有当块操作完全结束时，才会返回信息。

直接量字符串通常使用单引号括起来，在继续学习之前，简单地谈谈直接量。

1. 直接量(Literals)

直接量是固定的字符串、数字和布尔值。在运行时，它们不能够被修改，它们被限制为只读值。单字符和字符串直接量使用单引号括起指定。这里有一些字符串直接量示例：

- ‘ ’ —— 一个空格字符。
- ‘Hello, World!’（世界，你好！）。
- ‘This string contains “embedded” single quotes’（这个字符串包含一个内嵌的单引号）。
- ‘Double quotes are “Okay” too’（双引号也可以内嵌）。
- ‘12345’ —— 这是一个字符串，不是一个字面上的整数。
- ‘01/01/1900’ —— 看起来像个日期。
- ‘~!@#\$\$%^&*()_+=\|{}[];<>.,?/’ —— 直接量可以包含特殊字符。
- ‘’’’ —— 指定了一个包含三个字符的字符串‘’（单引号、空格、单引号）

字符和字符串直接量被当作CHAR（定长字符串）类型处理。它可以被赋值给任何CHAR或VARCHAR2变量。PL/SQL只有在字符串直接量中才大小写敏感。

数字直接量可以有任意整型值或浮点值，例如：

12 345	整型直接量
-12 345.0	浮点直接量
12345.678 90	浮点直接量可以有任意精度
100.	这是一个浮点直接量，精度为 0
1.2 345E2	可以使用科学计数法
1.2 345E-2	
0.12345或.12345	开头的0是可选的

声明基本常数包含程序中用到的直接量值是一个良好的编程习惯。可以给它们有意义的名字。另外，它们更具有可维护性，如果必须修改一个直接量数值，只需要在一个地方做改动即可。这项技术避免了不可维护的“幻数”综合症，这种症状的表现是在某个地方使用了一个特定数值，但没有任何人记得为什么用它。程序清单 28-2显示了一个例子。

清单28-2 circle.sql —— 使用常数使代码更易于维护

```
DECLARE
  PI CONSTANT REAL := 3.14159265359;
  circumference REAL;
  area REAL;
  radius REAL := &Radius;
BEGIN
  circumference := PI * radius * 2.0;
  area := PI * radius**2;
  DBMS_OUTPUT.put_line('Radius = ' || TO_CHAR(radius) ||
    ', Circumference = ' || TO_CHAR(circumference) ||
    ', Area = ' || TO_CHAR(area));
END;
```

当运行这个程序时，SQL*Plus首先提示你输入使用“与”符号（&，第五行）指定的捆绑变量的数值。这里是输入数值后所发生的结果：

```
Enter value for radius: 3.5
old 5: radius REAL := &Radius;
new 5: radius REAL := 3.5;
Radius = 3.5, Circumference = 21.99114857513, Area = 38.4845100064775
PL/SQL procedure successfully completed.
```

注意捆绑变量不是一个PL/SQL变量，它只是你必须提供数值的占位符。并且，如果你从第一个例子开始就遵照指示去做的话，不需要再次启动服务器的输出机制或再次激活输出机制。一旦这些事物被启动，它们会在会话期间始终存在（除非有意地关闭它们或重新进行连接）。

注意常数变量PI在它的声明期间是如何被指定和接受数值的。你还可以使用关键词DEFAULT代替赋值操作符“:=”，它在声明的环境中有相同的基本功能。

还支持布尔直接量TRUE和FALSE（注意它们没有被括在单引号中），它们可用于任意布尔表达式中和赋值给任意布尔型变量。

表28-2中的真值表显示了关系运算符AND、OR和XOR（异或）的所有二元布尔组合，以及一元否定（NOT或~）的TRUE、FALSE和NULL值。PL/SQL不支持关系运算符XOR，但是，PL/SQL的内建函数XOR（）执行这个操作。

程序清单28-3是一个相关的例子。

表28-2 AND、OR、XOR、NOT三逻辑真值表

p	q	p AND q	p OR q	P XOR q	NOT P
TRUE	TRUE	TRUE	TRUE	FALSE	FALSE
TRUE	FALSE	FALSE	TRUE	TRUE	FALSE
TRUE	NULL	NULL	TRUE	NULL	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE	TRUE
FALSE	FALSE	FALSE	FALSE	FALSE	TRUE
FALSE	NULL	FALSE	NULL	NULL	TRUE
NULL	TRUE	NULL	TRUE	NULL	NULL
NULL	FALSE	FALSE	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL

清单28-3 booly.sql：使用布尔型数据和运算

```

DECLARE
  T CONSTANT BOOLEAN NOT NULL := TRUE;  -- must have a value
  x BOOLEAN;
  y BOOLEAN;
BEGIN  -- illustrate Booleans
  x := 1 = 2;  -- this expression evaluates to false
  y := XOR(T, x);  -- and this evaluates to true
  IF (x) THEN  -- test for true fundamental
    DBMS_OUTPUT.put_line('x = TRUE');
  ELSIF (NOT x) THEN  -- test for false
    DBMS_OUTPUT.put_line('x = FALSE');
  ELSE  -- x must be null
    DBMS_OUTPUT.put_line('x is NULL');
  END IF;
  IF (y) THEN  -- test for true
    DBMS_OUTPUT.put_line('y = TRUE');
  ELSIF (NOT y) THEN  -- test for false
    DBMS_OUTPUT.put_line('y = FALSE');
  ELSE  -- y must be null
    DBMS_OUTPUT.put_line('y is NULL');
    DBMS_OUTPUT.put_line('Booly for you!');
  END IF;
END;
/

```

服务器响应下列信息：

```

x = FALSE
y = TRUE
PL/SQL procedure successfully completed.

```

没有函数可以将一个布尔数值转换为字符串或将字符串转换为布尔值，你可在稍后编写这样的函数。注意程序的第二行，如何指定一个变量不能为空。这限制了程序中的取值，但PL/SQL支持该语法。注意，对于声明不能为空的变量，系统处理时会有一些额外开销。

使用值T和x进行试验，返回该块查看得到了什么（SQL*Plus 的EDIT命令可以编辑文本并返回该文本）。如果忽略缺省赋值，将得到最基本的结果，如下所示：

```

ERROR at line 1:
ORA-06550: line 2, column 3:
PLS-00322: declaration of a constant 'T' must contain an initialization
assignment
ORA-06550: line 2, column 5:
PL/SQL: Item ignored

```

注意 参阅第26章了解EDIT命令的详细信息。

你遇到了第一个编译错误。在第一行上的错误告诉你整个未命名块有一个错误。这是因为整个块是作为一个事务提交给服务器的。下一个信息非常有用，告诉你忘记了初始化常量值。事实上，所有的常量必须被初始化，即使它为空，只要没有 NOT NULL（不为空）子句限制，它也肯定是合法的。ORA-06550错误代码表明Oracle服务器遇到了某种类型的PL/SQL编译错误（留给PL/SQL编译器解释实际的错误）。

如果删除CONSTANT（常量）和初始化值，但留下 NOT NULL（不为空）子句，将得到下列信息：

```
ERROR at line 1:
ORA-06550: line 2, column 5:
PLS-00218: a variable declared NOT NULL must have an initialization assignment
```

在运行时，当块被输入时，一个变量被初始化为指定的值。如果没有规定的赋值，那么缺省情况下该变量被赋予一个 NULL（空）值。试一试除去 CONSTANT、NOT NULL 和初始化，将得到下面的结果：

```
x = FALSE
y is NULL
Booly for you!
```

变量在运行时接受数值，并不是在编译时接受数值，因为像其他语言中的自动（或堆栈）变量一样，除非块在作用域中，否则它们并不真正存在。这意味着当块在作用域外时，它们被释放，并且在块再次进入时被重新分配。要注意多次调用过程和函数所造成的系统额外开销。事实上，要充分利用它。因为变量缺省地被初始化为 NULL（空），如果它们确实需要一个初始非空值，这是一个做这件工作的好地方。

当一个变量被声明为 CONSTANT（常量）时，它的值在运行时不能改变。试图改变常量的值将会产生一个例外。还有，初始化直接量的类型和长度必须符合变量的类型和长度。例如，如果一个变量 x 的声明为 CHAR(4) := “NO GOOD!”，在运行时（不是编译时）将会产生一个例外。如前所见，常量必须在它们的声明中赋值，即使需要的值是 NULL，否则，会发生编译错误。

一个变量可以被声明为 NOT NULL（非空），以指明它不能接受一个 NULL（空）值。这样的变量在声明时必须被初始化为某个非空值。注意 CONSTANT（常量）和 NOT NULL（不为空）关键词的正确位置，它们必须被分别地放在数据类型的前后。

这里是一些合法的和非法的初始化语句示例：

```
DECLARE
  price          NUMBER(5,2) := 19.92;    -- valid initialization
  discount       NUMBER(3,3) := .0625;    -- gets rounded to .063
  max_quantity   INTEGER(4) := 50000;     -- runtime error!
  max_discount   CONSTANT REAL := 0.75;   -- valid
  min_discount   CONSTANT REAL;           -- compile error! not initialized
  disc_type      VARCHAR2(1) := NULL;     -- valid and redundant
  disc_name      CONSTANT CHAR(20) := NULL; -- valid but of dubious value
  quantity       INTEGER NOT NULL := 0;   -- correct usage
  item_name      VARCHAR2(30) DEFAULT 'Hammer'; -- alternate assignment
```

2. 分支

在考虑程序分支时，首先让我们列举 IF/THEN 语句支持的全部语法结构类型：

1) 使用条件（IF/THEN/ELSE）逻辑 基本的条件测试与分支逻辑具有以下形式：

```

IF (某些条件为真) THEN      —— 测试条件
.....                     —— 条件为真, 做这项工作
ELSE                       —— 条件为假
.....                     —— 做这项工作
END IF;                    —— 测试结束

```

就记录来说, IF...THEN后面的第一个语句块称为前项, 跟在 ELSE后面的块称为后项。可以在 THEN、ELSE和ELSIF块中拥有多条语句。如果愿意, 可以在前项或后项内部放置 BEGIN...END语句。当然, 后项是可选的。注意, 块以 END IF结束。在条件语句的开始和结尾处进行注意是一个好习惯, 尤其对于一个长而重要的语句, 可以知道它是用来做什么的。

```

IF (某些条件为真) THEN      —— 检验某些条件
  BEGIN
  ...
  END;
ELSE                       —— 条件为假
  DECLARE                  —— 定义一些局部变量
    X NUMBER;
  BEGIN
  ...
  END;
END IF;                    —— 检验某些条件

```

还会创建前提中的变量 X, 除非执行了结果。可以使用这种方式来限制局部变量只用于需要它们的块中, 这可以节省小量内存。

如何决定语句的缩进取决于你, 但要注意将一个块缩进到另外一个块内部。

可以嵌套语句到任意深度, 可以使用这种方式执行一些非常复杂的逻辑操作。

使用简化求值计算条件表达式。这意味着, 如果一个复杂 OR表达式的第一部分值为 TRUE (真), 或者一个 AND表达式的第一部分值为 FALSE (假), 求值立即停止, 没有再进一步求值的必要了。大多数其他语言同样支持简化求值。这种表达式的计算步骤总是从左到右, 圆括号内的表达式有较高优先级。

2) 测试NULL (为空) 条件 特殊的直接量值NULL可以赋值给任何数据类型的变量。它代表一个未知数值。你必须使用特殊的语法 IS NULL和IS NOT NULL分别检测它的存在或不存在。下面的测试

```

IF (x=NULL) THEN ...和
IF (x= ! NULL) THEN ...

总会出错, 正确的方式是
IF (x IS NULL) THEN ...和
IF (x IS NOT NULL) THEN ...

```

3) 使用连续的IF逻辑 有时需要一个接着一个地测试一系列的数值。如果前面的一条语句已经完全成功, 后面的语句预期会失败, 那么编写一组 IF/THEN语句效率会很差, 如下所示:

```

IF (val = '0') THEN -- is value a digit?
...
END IF; -- zero?
IF (val = '1') THEN -- maybe a 1?
...
END IF; -- 1?

```

```

...
IF (val = '9') THEN  -- maybe a 9?
...
END IF;  -- end of test

```

可以使用嵌套IF/ELSE语句，但这样做会使语句很难看，它使全部语句都缩进。可以使用ELSIF语句代替上面的语句，让代码更有效率。只要其中的一个条件为真，它的前项被执行并且退出整个块。

```

IF (val = '0') THEN  -- is value a digit?
...
ELSIF (val = '1') THEN
...
ELSIF (val = '2') THEN
...
ELSIF (val = '9') THEN
...
ELSE  -- not a number
...
END IF;  -- end of test

```

提示 PL/SQL语句没有SWITCH语句（像在C语言中一样），没有CASE语句（像在PASCAL语言一样），也没有可计算的GOTO语句（像FORTRAN语言）。PL/SQL使用连续的IF模拟这些语句结构。

清单28-4显示了一个简单的例子，编写该程序用于估算作者 1996年的基础联邦税款债务（来自美国国税局公告15，通知E，雇主税款指南，修订版本，1997年1月）。

清单28-4 fedtax.sql —— 用来检测取值范围的连续的IF逻辑

```

SET SERVEROUTPUT ON;
-- assumes annual payroll, married
DECLARE
num_wh NUMBER := &num_withholding;
wh_amount NUMBER;
gross NUMBER := &annual_gross_salary;
liab NUMBER;
adj_gross NUMBER;
BEGIN
  DBMS_OUTPUT.ENABLE;
  wh_amount := num_wh * 2550;  -- annual allowance
  adj_gross := gross - wh_amount;
  IF adj_gross <= 6425 THEN
    liab := 0;
  ELSIF adj_gross <= 44250 THEN
    liab := (adj_gross - 6425) * 0.15;
  ELSIF adj_gross <= 89675 THEN
    liab := 5673.75 + (adj_gross - 44250) * 0.28;
  ELSIF adj_gross <= 151850 THEN
    liab := 18392.75 + (adj_gross - 89675) * 0.31;
  ELSIF adj_gross <= 267900 THEN
    liab := 37667 + (adj_gross - 151850) * 0.36;
  ELSE  -- over max
    liab := 79445 + (adj_gross - 267900) * 0.396;
  END IF;  -- test adjusted gross
  DBMS_OUTPUT.PUT_LINE('FEDERAL TAX LIABILITY: ' || TO_CHAR(liab));
END;
/

```

当运行这个函数时，你被提示输入捆绑变量值，并且服务器返回下列主要信息：

```
Enter value for num_withholding: 4
```



```
old 2: num_wh NUMBER := &num_withholding;
new 2: num_wh NUMBER := 4;
Enter value for annual_gross_salary: 60000
old 4: gross NUMBER := &annual_gross_salary;
new 4: gross NUMBER := 60000;
FEDERAL TAX LIABILITY: 7227.75
PL/SQL procedure successfully completed.
```

被测试表达式的括号是可选的，但是推荐使用它们，对于一个复杂的表达式更要使用括号。

4) 非条件分支 PL/SQL使用GOTO语句支持非条件程序分支。可以使用一个标签指定一个跳转的目标点，就像 BASIC和汇编语言中的一样。可以跳转到当前块的任意地方或跳转到封装块中。通常，不赞成使用 GOTO语句，因为它导致非结构化的程序代码。这样的程序通常很快会变得难以维护。GOTO语句的语法如下面的程序清单 28-5所示。

清单28-5 okgoto.sql —— 使用GOTO

```
BEGIN
  BEGIN
    GOTO MID;  -- forward reference OK because it's in scope
  END;
  <<MID>>
  NULL;
END;
/
```

标签MID使用双角括号括起来表示。注意标签没有使用分号结束。

这个SQL脚本可以成功地编译。但是，程序清单 28-6显示了一种不同的主要情形。

清单28-6 badgoto.sql —— 不合理地使用GOTO语句

```
BEGIN
  GOTO MID;  -- invalid!
  BEGIN
    <<MID>>
    NULL;
  END;
END;
/
```

因为标签在范围之外，将得到一个编译错误：

```
ERROR at line 1:
ORA-06550: line 2, column 8:
PLS-00201: identifier 'MID' must be declared
ORA-06550: line 2, column 3:
PL/SQL: Statement ignored
```

外部的块不知道内部块中存在的任何东西。

注意NULL（空）语句。这确实是一个合法的语句，仅仅作为占位符使用。不能够在任意形式的END语句前面（像在END IF中）立即拥有一个标签，否则将得到一个编译错误。可以使用一条NULL语句解决这个问题。

不可以从一个块中跳转到封装块中同一层次上的其他块中，如程序清单 28-7所示。

这些限制还适用于条件语句。原则上，不能从前项跳转到后项（见程序清单 28-8）。

这时Oracle检测出你做了某些超乎常理之外的事情，并且给出下列信息：

```
ERROR at line 3:
ORA-06550: line 3, column 5:
PLS-00375: illegal GOTO statement; this GOTO cannot branch to label 'CONSEQUENT'
ORA-06550: line 4, column 3:
PL/SQL: Statement ignored
```

清单28-7 bad2goto.sql —— 不能在一个封闭块内部同一层次的块之间使用 GOTO语句

```
BEGIN
  BEGIN
    GOTO OTHER; -- invalid!
  END;
  BEGIN
    <<OTHER>> -- out of scope
    NULL;
  END;
END;
/
```

清单28-8 bad3goto.sql —— 不可以在前项与后项间跳转

```
BEGIN
  IF (TRUE) THEN -- always do the antecedent
    GOTO CONSEQUENT;
  ELSE -- consequent
    <<CONSEQUENT>>
    NULL;
  END IF; -- of jump example
END;
/
```

还有，Oracle不检查无限循环，如下所示：

```
BEGIN
  <<INFINITE_LOOP>>
  GOTO INFINITE_LOOP;
END;
```

警告 少量程序代码就可以使SQL*Plus会话陷于死循环的混乱状态！如果愚蠢到做出这样的事情，将不得不请求数据库系统管理员终止该会话，尤其在Windows 3.1环境下，系统将会被锁起，直到该会话被终止。如果正在运行Windows 95或Windows NT，可以放弃这个SQL*Plus会话，丢失机器上用于基本处理的一些资源（例如为该会话分配的内存和它使用的可用句柄）。

5) 标签和作用域规则 变量声明的作用域对于封装块是局部的。变量通常只能在块的范围内被引用，或在定义它的子块中被引用。子程序参数表中的变量声明只能够在那个子程序中被引用。另外，子程序体中的任何变量声明都被限制为子程序的局部变量。一个内部块可以重新命名一个变量，对于外部区域中的变量，它仍然是一个新的、不同的变量。

在运行时，当执行到块作用范围以外时，它的变量、游标以及相似的局部结构都被释放。当该块再次进入作用范围，变量被再次创建并被初始化为在声明中指定的任意值。你将在后面存储子程序部分对此进行仔细观察。

当决定哪里是放置声明的最佳位置时，通常使用的经验规则是缩小变量的声明范围，只作用到那些需要引用它的块。因此，只有当实际需要时，该对象才会存在，节省了内存。为配合节省内存，分配对象所要求的对象保留需要和节约时间的探索成为第一位的。

通过使用标签，一个内部块可以引用包含块中的对象。这是一种坏的编程习惯，应该使用一个不同的变量名或使用向子程序传递堆栈中的变量来替代它。但是，PL/SQL 语言确实支持标签，所以如果感觉必须使用这个特性，遵照以下步骤：

- 在外部块的开始处插入一个标签。
- 在内部块中使用点符号引用由外部块定义的变量。
- 标签是一个限定词，需要用来引用与内部变量同名的外部变量。

程序清单 28-9 显示了正确的和错误的限定基本标签的用法。

清单 28-9 badlabel.sql —— 使用与误用标签来区分块

```
<<OUTER_BLOCK>>    -- this label names the outer block
DECLARE
  x NUMBER; -- available to both inner and next blocks
  y NUMBER;
BEGIN
  <<INNER_BLOCK>>    -- might as well label the inner one too
  DECLARE
    x NUMBER; -- only available to this block
  BEGIN -- inner block
    x := OUTER_BLOCK.x; -- qualified reference to outer block variable
    y := 0; -- an unqualified reference to outer block variable

  END INNER_BLOCK;
  <<NEXT_BLOCK>>
  DECLARE
    x NUMBER;
  BEGIN -- next block
    x := INNER_BLOCK.x; -- THIS IS ILLEGAL!!
    x := OUTER_BLOCK.x; -- this is OK
  END NEXT_BLOCK;
  x := INNER_BLOCK.x; -- THIS IS ALSO ILLEGAL!!
END OUTER_BLOCK;
```

将得到下面的错误信息：

```
ERROR at line 1:
ORA-06550: line 17, column 22:
PLS-00219: label 'INNER_BLOCK' reference is out of scope
ORA-06550: line 17, column 5:
PL/SQL: Statement ignored
ORA-06550: line 20, column 20:
PLS-00219: label 'INNER_BLOCK' reference is out of scope
ORA-06550: line 20, column 3:
PL/SQL: Statement ignored
```

注意，不能够反向操作，即，不可以在外部块中引用内部块的变量。为什么不能？因为它不存在。内部块中的变量没有被创建，直到你真正地进入内部块时，它们才被创建，并且在程序退出内部块时，它们立即被释放，所以外部块没有机会引用内部块的变量。

同样，一个子程序中嵌套的子程序使用带有点符号的子程序的名字来引用封装子程序层定义的一个变量。通过参数表传递需要的值，或简单地在内部块中不再使用相同的变量名是一个更好的编程习惯。非正规的或特殊的东西涉及得越少，代码维护起来就越容易。

可以拥有嵌套块而不标注它们。甚至可以在子块中创建新的变量。当执行一个可能会产生例外的操作，并且需要在封装块内部定位例外处理程序时嵌套块非常有用。

你已经看到了一些标识符 —— 变量、数据类型、包和过程。现在注意有关它们命名的一些常用规则。

3. 标识符

标识符是代表编程项目的词素，例如变量、常量、用户定义数据类型、游标、例外、数据库表、列、过程、函数和包。一个标识符必须以字母开头，最多可以有 30 个字母、数字、下划线、美元符号 (\$) 或磅字符 (#)。

合法标识符	非法标识符
PI	3D-array
length_of_string	function_header
char\$	var name
RQ\$Get\$Segment	area/height
local##time	\$HOME

标识符不能含有内嵌的空格、连字符或斜线，但是可以使用双引号括起标识符，让它们区别于其他的词素，例如当一个表的列名与 PL/SQL 的保留字相同时。EXCEPTION 是一个 PL/SQL 的保留字，然而它不是 SQL 的保留字，所以可以被用作列名。正确引用这样一个列的基本命令是使用双引号，如下方式表示：

```
SELECT "EXCEPTION" INTO exc FROM prog_errors;
```

因为列以大写形式存储，带引号的标识符必须是大写的。还可以使用双引号来包括空格或其他分隔符，例如：

```
SELECT cr_msg "*** Credit Message ***" FROM messages;
```

在 SQL*Plus 中这更典型地用作列标题。

4. 注释代码

好的程序员要为他们的程序代码添加大量的注意。当编写代码时，他们拥有的时间越多，他们的注意就会越好。当首次设计一个复杂的 PL/SQL 程序时，可以使用注释来指明不同的部分。事实上，可以经常从程序的说明中直接获取注意，然后可以回到每个部分并使用真正的代码充实它。例如，使用含有某些基本信息的模块头注释，然后根据作用范围和类型组织声明，如下所示：

```
/*
  Program Name: c2f
  Module Name : c2f.sql
  Written By  : Daniel J.. Clamage
  Description : .This module converts Celsius to Fahrenheit.
  Modification: V.001 04-OCT-1997 - djc - Initial release.
*/
...
-- public global constant variables (shared between modules)
prog_version CONSTANT VARCHAR2(25) := 'V.001 04-OCT-1997 - djc';
prog_name CONSTANT VARCHAR2(30) := 'c2f';
...
-- public global record types needed
...
-- public global variables
...
-- public global cursors needed
...
```

当编译一个保存的带有多行注释的基本脚本时，SQL*Plus 将它们作为一个 DOC（文档）块回显：

```
SQL> @d:\plsqli\library\c2f
DOC> Program Name: c2f
...
DOC> Modification: V.001 04-OCT-1997 - djc - Initial release.
DOC>*/
```

当编译一个完整的模块集，并且将结果脱机存储到一个文件时，这非常方便。我们建议在一个模块的注释头块中限制使用多行注释，因为在编译的时候它们会被回显，引起一些混乱。如果第一个多行注释分隔符没有放在它自己的行上，那么第一条注释行就不显示。原则上，可以在任意地方使用单行注释。

28.5.4 声明变量

通常有两种类型的变量：标量的和复合的。标量变量只允许一个值。复合变量可以包含多个相同的或单一的数据类型的值，取决于复合变量的种类。一条语句只能声明一个变量。

表28-3显示了PL/SQL支持的标量数据类型和它们对应的数据库类型（如果有的话）的列表。

表28-3 标量数据类型（+代表7.2版；^代表7.3版；#代表8.0.3版）

类 型	子 类	说 明	取值限制	数据库限制
CHAR(s)	CHARACTER STRING ROWID* NCHAR***	定长字符串，** 接受NLS数据	0~32 767字节， 长度可选，缺省为1	255字节，*接受数 据库ROWID 字符串，**2 000字节
VARCHAR2(s)	VARCHAR STRING± NVARCHAR2#**	变长字符串， *接受NLS数据	0~32 767字节， 长度可选，缺省为1	2 000 字节 #4 000 字节
NUMBER (p, s)	NUMERIC DEC DECIMAL INT* INTEGER * FLOAT * REAL DOUBLE PRECISION SMALLINT-*	压缩十进制值p = 精度(共#位数字)s=范围 (这里进行四舍五入)， *不能指定范围	取值范围是 1.0E ⁻¹²⁹ ~9.99E ¹²⁵ 。 精度为1~38(缺省长度 是系统支持的最大值)。 比例是-84~127（缺省 为0）	与PL/SQLNUMBER 长度相同
BINARY INTEGER	NATURAL * POSITIVE** NATURALN^ - POSITIVEN^ - SIGNTYPE#***	有符号二进制数，*从 0开始，**从1开始，预 定义非空；与上面具有 相同的范围	取值范围是-2 ³¹ -1到2 ³¹ -1； 或±2 147 483 647，***限 制为(-1, 0, 1)用于实现 三态逻辑	不存在相应的数据库 类型(使用NUMBER)
PLS INTEGER^		快速机器运算的有符号 二进制数字,比NUMBER 占用的存储少	取值范围是-2 ³¹ -1到2 ³¹ -1； 或±2 147 483 647，	不存在相应的数据 库类型(使用NUMBER)
DATE		一个内部日期值	公元前4712年1月1日 —公元后4712年12月31 日；以秒表示的时间从 午夜开始计时（缺省的 时间是午夜12：00）	与PL/SQL日期相同
BOOLEAN		布尔（逻辑）值	TRUE、FALSE、NULL (未知值)	不存在数据库类型
LONG		变长字符串	0~32 767字节（长度未 规定）	0~2 147 483 647字 节

(续)

类 型	子 类	说 明	取值限制	数据库限制
RAW(s)	LONG RAW	非解释二进制数据 (与字符集无关)	RAW : 0~32 767字节 ; LONG RAW : 0~32 767 字节 (与LONG相同) (要求长度至少为1)	RAW是2000字节, LONG RAW是2 147 483 647字节
REF CURSOR±		游标变量(游标的指针)		不存在数据库类型
LOB	BFILE *	存储在数据库内部或外	2 ³² -1字节或0~4GB, *只	与PL/SQL LOBs相同
(巨型对象)	BLOB**	部操作系统文件中的巨	有外部文件 (只读), **内	
	CLOB***	型、无结构的或二进制	部/外部二进制数据(读/写),	
	NCLOB****	数据 (图像、声音、视	***内部/外部单字节数据	
		频、文档)	****内部/外部固定宽度多 字节数据(读/写)	

子类型通常与它们的对应主体有相同的取值限制。Oracle在PL/SQL中提供它们是为了Oracle与其他数据库中的列数据类型的兼容性。例如,当你希望一些事物更具描述性时,可以使用可选的NUMBER子类。如果一个类型使用了一个可选的长度参数,它被如上所示放在括号内。只能用整型直接量指定长度参数(例如 CHAR (3))。对于字符串,一个长度为0的串是NULL(空)值(未知值)。

字符串被当作标量对待。为了能够存取单个字符,可以使用Oracle提供的字符函数SUBSTR()、INSTR()等等。一个CHAR类型数据的长度使用字节表示,而不是使用字符表示。这意味着多字节字符集将需要更多的存储空间。注意从7.1版到7.2版STRING子类处理上的变化。

为了完成数学运算,Oracle执行从一个NUMBER(压缩十进制)数据类型到一个内部浮点表示的转换。相反,BINARY_INTEGER和PLS_INTEGER数据类型已经具有内部格式。使用BINARY_INTEGER数据类型的索引编写的循环程序,当值限制在一定范围内时,执行速度较快,因为不需要转换过程。PLS_INTEGER执行速度甚至比BINARY_INTEGER更快,因为它使用本地机器运算,而BINARY_INTEGER和NUMBER依赖于数学库程序。但是,BINARY_INTEGER值可以被赋值给NUMBER变量以避免溢出例外,而使用PLS_INTEGER则不会消除溢出例外。不需要过多关心这个细小的语义差别。

负数四舍五入到小数点的左边;例如,-3四舍五入到最近的千位(1000, 2000, 3000……),而+3四舍五入到最近的千分位(0.001, 0.002, 0.003……)。

观察这个表,也许会意识到当与数据库中的列交互时,LONG和RAW的PL/SQL类型拥有有限的作用。PL/SQL的LONG RAW数据类型实际上比RAW长度要短!他们应该称它为RAW LONG。但是,有一个Oracle提供的程序可以将一个数据库的LONG值转换为PL/SQL的LONG变量。

这个表中其他要注意的事情是数据库数据类型的限制和对应的PL/SQL数据类型可能会有明显的不同(特别是字符类型)。在编程时请记住这点。可能遇到的取值错误与这个差异有关(当不小心试图将一个较长的字符串赋值给一个较短的变量时,会出现错误)。

1. 使用PL/SQL数据类型

一些标量变量声明的例子如下所示:

■ item_name CHAR(32); —— 也许尾部使用空格填补。

■ item_category VARCHAR2(32); —— 这样的字符串不需要填以空格。

- price NUMBER (5, 2) ; —— 可以接受± 999.99之间的任意数字。
- quantity INTEGER(3) ; —— 可以接受± 999之间的任意数字。
- discountable BOOLEAN ; —— 可以是TRUE、FALSE或NULL。
- discount REAL (3, 3) ; —— 在± 0.999之间的任意实数。
- run_date DATE ; —— 日期域占用7个字节。

当然，任意数据类型的变量可以接受一个 NULL（空）值（而不仅仅是布尔类型变量才可以）。

一个真正灵活的特性称作“配置”变量。不需要确实地编写一个数据类型，可以使用其他变量、数据库列或表的数据类型。可以使用 %TYPE和%ROWTYPE属性实现这个目的。%TYPE属性提供了所需要的数据库列或变量的类型及长度。%ROWTYPE属性允许人们定义一个记录变量，它的成员变量拥有表或游标中每一列正确的类型及长度。使用点符号引用记录中的每个成员变量。我自己的惯例是使用带有‘_rec’后缀的%ROWTYPE变量以指明该变量是一个记录变量。例如：

- quantity orders.qty%TYPE; —— 基于表订单中的qty列。
- discount orders.discount%TYPE; —— 基于表订单中的discount列。
- orders_rec orders %ROWTYPE; —— 基于表订单中的一条记录。

这里是更多的使用类型的示例：

```
DECLARE -- define user variables
MAX_INT CONSTANT INTEGER := +2147483647;
MAX_STR VARCHAR2(32767); -- let's use this for typing only
notes MAX_STR%TYPE; -- based on another variable's type
items_rec items%ROWTYPE; -- based on a table in the database
iname items.item_name%TYPE; -- based on a column in a table
last_id MAX_INT%TYPE; -- this must be a constant too
BEGIN -- executable code
iname := 'HAMMER';
items_rec.item_name := iname; -- compatible data types
last_id := 0; -- Wait! Aaggh! Runtime error!
```

这提供了非常好的方法，控制解决了当数据库列或表改变时所产生的数据类型问题。因为变量的类型是基本类型，不是显式声明的，正确的数据类型和长度已经就绪。如果表发生改变，只需要重新编译代码即可。使用这种方法，基本变量提供了一种数据独立和降低代码维护的手段。想像一下，当一些列的长度或类型改变时，如果代码中使用显式的数据类型，需要做多少工作来修改多个模块中的代码。当然，如果表列的数据类型从 CHAR (5) 改变为 NUMBER (5)，你还要对你的 PL/SQL 代码重新加工（我已经遇到过这样的事情了），例如，要插入 TO_CHAR () 函数转换。

然而，当使用 %TYPE 时，会卸去所有的负担。在前面的例子中，可以看到程序员错误地试图给一个基于常量的变量赋予一个新值，这样也会使基本变量变为一个常量。

尤其应该在子程序参数表和函数返回值声明中使用基本变量，那里变量的类型被声明，但长度未被声明。

2. 定义复合数据类型

PL/SQL 支持两种形式的复合（一个矢量或数值集）数据类型： RECORD（记录）和 TABLE（表）。复合变量可以基于一个表或游标的行类型创建，如上所示。这和 RECORD 变量有相似的语义。

要声明一个新的复合数据类型，要使用关键字 `TYPE`。`TYPE`声明只声明了一个新的数据类型，它们并没有定义存储空间。为了能够使用它们，必须声明变量为哪种类型。在这里，使用新数据类型名后缀 `_TYPE`的约定以指明该变量被用作一个数据类型：

```
DECLARE
  -- user-defined data types
  TYPE MY_STRING_TYPE IS RECORD OF (
    str_len INTEGER := 0, -- initialized when variable is declared
    str VARCHAR2(32767)); -- defaults to NULL
  TYPE MY_ARRAY_TYPE IS TABLE OF CHAR(8) INDEX BY BINARY_INTEGER;
  -- variables
  str MY_STRING_TYPE; -- a new composite variable
  arr MY_ARRAY_TYPE; -- a new 1-D array
```

3. 创建自己的记录类型

`RECORD`类型定义了一个结构，该结构可以包含任意数目的任意数据类型的成员变量，包括前面定义的`RECORD`或`TABLE`类型。与一些表`%ROWTYPE`的记录变量一样，使用圆点符号引用每个成员。在运行时对基本变量的成员进行初始化定义。

可以嵌套`RECORD`类型：

```
DECLARE
  TYPE ZIP_TYPE IS RECORD (
    zip5 VARCHAR(5),
    DASH CONSTANT VARCHAR2(1) := '-', -- for display purposes perhaps
    plus4 VARCHAR2(4) := '0000'); -- initialized at runtime
  TYPE ADDR_TYPE IS RECORD (
    line1 VARCHAR2(30),
    line2 VARCHAR2(30),
    city VARCHAR2(20),
    state VARCHAR2(2),
    zip_code ZIP_TYPE);
  TYPE EMPLOYEE_TYPE IS RECORD (
    ssn VARCHAR2(9);
    dob DATE;
    address ADDR_TYPE);
  employee_rec EMPLOYEE_TYPE; -- actual storage defined
BEGIN
  employee_rec.ssn := '123456789'; -- someone's SSN#
  employee_rec.address.city := 'Pittsburgh';
  employee_rec.address.zip_code.zip5 := '15210'; -- Pittsburgh's zip
  employee_rec.address.zip_code.plus4 := '3702'; -- Mt. Oliver in Pgh
```

注意引用该嵌套记录任一子组件的圆点符号的用法。这种形式的主要优点是可以更为容易地管理大量不同数据类型的相关数据。使用一个参数，就能将一个数据集传递给子程序或将数据集从子程序返回。它比其他方式更加整齐和紧凑。记录使开发者按照紧密相关的数据块来考虑问题。

只要声明为具有相同的数据类型，就可以将一个记录变量赋值给其他的记录变量。基于数据库表的`ROWTYPE`变量和那些定义为记录类型的变量通常是不能兼容的，即使它们的成员完全匹配：

```
DECLARE
  TYPE ORDERS_TYPE IS RECORD ( -- looks like the database table
    ord_num orders.ord_num%TYPE,
    quantity orders.quantity%TYPE,
    ...
    discount orders.discount%TYPE);
  orders_rec orders%ROWTYPE; -- based on a database table
  new_ords_rec ORDERS_TYPE;
```

```

old_orcs_rec ORDERS_TYPE;
BEGIN
... -- do some work
old_orcs_rec := new_orcs_rec; -- this is correct
new_orcs_rec := orders_rec;   -- this is incorrect!

```

当向数据库表中插入记录时，VALUES子句必须分别指明每个成员变量：

```

INSERT INTO orders orders_rec;    -- WRONG!
INSERT INTO orders old_orcs_rec;  -- WRONG!
INSERT INTO orders
(ord_num, qty, ..., discount) VALUES
(new_orcs_rec.ord_num, new_orcs_rec.quantity, ...,
 new_orcs_rec.discount);          -- CORRECT!

```

4. 创建自己的数组

PL/SQL表是任何单独的标量类型数据的一维数组。对于 7.3版之前的 Oracle服务器，TABLE可以含有一个 RECORD或另外一个 TABLE。对于 Oracle 7.3 版和更高版本，一个 TABLE可以包含一个用户定义的记录，而不能包含其他的 TABLE。在 Oracle8i中，TABLE就是几种集合的类型。

PL/SQL的TABLE不同于Oracle数据库的表。它是一个无界的数组，在被赋值以前，它的元素是不存在的。它总是使用BINARY_INTEGER索引，给予它的元素的索引范围是±2 147 483 647（还好，不是完全无界）。程序员可以选择数组索引的开始偏移量为 0、1或其他任何有意义的位置。因为一个未赋值的元素并不存在，数组可以稀疏地包含定位的数值，不需要额外内存的使用。每个元素使用圆括号中的偏移量来引用。首先声明一个表类型，然后基于该类型创建一个变量，就可以创建一个 PL/SQL表变量了。例如，下面的代码只使用足够的内存来存储这些值：

```

DECLARE
TYPE STR_TYPE IS TABLE OF VARCHAR2(8)
INDEX BY BINARY_INTEGER;
arr STR_TYPE;
i BINARY_INTEGER := 17; -- Universal Constant of Uncertainty
BEGIN
arr(-2,147,483,647) := 'smallest';
arr(0) := 'zero';
arr(+2,147,483,647) := 'biggest';
arr(i) := 'UCU'; -- an index variable makes for good loops

```

警告 在7.3以前的版本中，一个TABLE元素被分配一个缓存区，大小依据数据类型和它被声明的长度而定，这增添了系统的一点开销。这意味着，如果表为 VARCHAR2 (32 767)，存储在表中的每条字符串需要 32KB缓存，即使这个字符串只有一个字节。注意不要耗光系统的内存。

当声明许多长字符串变量时，考虑是否 PL/SQL总是为 VARCHAR2变量分配所声明大小的缓存是非常明智的。

通过以如下方式声明表，可以指定一个存储的值不能为 NULL（空）：

```

DECLARE
TYPE STR_TYPE IS TABLE OF VARCHAR2(8) NOT NULL
INDEX BY BINARY_INTEGER;
arr STR_TYPE;
BEGIN
arr(0) := NULL; -- raises an exception!

```

还可以使用基本引用，例如对一个数据库列的引用：

```
DECLARE
  TYPE QTY_TYPE IS TABLE OF items.quantity%TYPE NOT NULL
    INDEX BY BINARY_INTEGER;
  qty QTY_TYPE;
BEGIN
  qty(0) := 0; -- quantity of zero
```

使用PL/SQL表的好处包括以下几个方面：

- 它们让你能够将大量数据传递进、出一个子程序。
- 它们能够很快地在一组巨型数据上执行操作。
- 它们可以和来自数据库表的大量数据集一起加载和处理，而不需要额外的数据库访问开销。
- 在Oracle 8i中，它们可以在一个带有 new Table()运算符的选择语句中使用。
- 它们简化了执行复杂的操作，这些操作使用 SQL实现将非常困难，尤其是那些在数据集中含有多个行的操作。

另外，可以将这些 PL/SQL表捆绑到OCI中的数组和预编译程序中，这会允许非常快速的数据传输。但是，当在 PL/SQL表和这些程序之间转移大量的数据时，请考虑网络负载的增加。

注意 只可以将基于标量数据类型的一维宿主数组捆绑到PL/SQL表（不支持基于记录的宿主数组）。

如果表变量有相同的基本类型，可以相互赋值。赋值将表的整个内容拷贝给其他的表变量，这项技术还用于清除一个表。

```
DECLARE
  TYPE ORDITEM_TYPE IS TABLE OF orders.item_no%TYPE
    INDEX BY BINARY_INTEGER;
  clr_orditem ORDITEM_TYPE; -- use ONLY to clear array
  new_orditem ORDITEM_TYPE; -- new list of order items
  old_orditem ORDITEM_TYPE; -- old list of order items
BEGIN
  ... -- fill up new order items array
  old_orditem := new_orditem; -- copy for safekeeping
  ... -- modify new order items array and save
  new_orditem := clr_orditem; -- erase contents for next operation
```

如果试图读取一个还没有被赋值的元素，PL/SQL出现NO_DATA_FOUND例外。程序清单28-10展示了这种现象。

清单28-10 badref.sql —— 在给数组元素赋值前引用它会产生一个例外

```
DECLARE
  SCHAR VARCHAR2(1); -- we'll use this for typing only
  TYPE SCHAR_TYPE IS TABLE OF SCHAR%TYPE INDEX BY BINARY_INTEGER;
  schar_arr SCHAR_TYPE;
  local_schar SCHAR%TYPE;
BEGIN
  local_schar := schar_arr(0); -- no value stored!
END;
/
ERROR at line 1:
ORA-01403: no data found
ORA-06512: at line 7
```

在PL/SQL 2.3版（Oracle 7.3）和以后的版本中，表还可以基于所声明的数据库表和记录类型。每个记录成员都必须是一个标量类型。这样可以让你能够加载并处理一组数据库记录。

5. 引用行类型的数组

可以使用点符号引用拥有数组元素的记录的一个特定成员：

```
DECLARE
  TYPE ORDERS_TYPE IS TABLE OF orders%ROWTYPE
    INDEX BY BINARY_INTEGER;
  qty orders.quantity%TYPE; -- local copy of quantity
  orders_tab_rec ORDERS_TYPE; -- array of table rows
BEGIN
  ... -- fill the array with database table rows
  qty := orders_tab_rec(i).quantity;
```

6. PL/SQL表属性

在PL/SQL 2.3版中，表拥有可以帮助你更为容易地操作它们的属性。COUNT、FIRST和LAST可以在表自身上操作，EXISTS、PRIOR、NEXT和DELETE可以在一个特定的表元素上操作。表28-4列出了一些这样的属性。

表28-4 使用PL/SQL表属性

属 性	说 明	返 回 值	用 法
EXISTS	测试一个表元素的值	TRUE/FALSE	Tablename.EXISTS (偏移量)
COUNT	返回表中的项目数	BINARY_INTEGER	Tablename.COUNT
FIRST	返回第一个表项的偏移量	BINARY_INTEGER	Tablename.FIRST
LAST	返回最后一个表项的偏移量	BINARY_INTEGER	Tablename.LAST
PRIOR	返回上一个表项的偏移量	BINARY_INTEGER	Tablename.PRIOR (偏移量)
NEXT	返回下一个表项的偏移量	BINARY_INTEGER	Tablename.NEXT (偏移量)
DELETE	删除指定的元素，部分或全部	无	Tablename.DELETE ([off_m, [off_n]])

当已经位于FIRST或LAST元素时，PRIOR和NEXT属性分别返回NULL（空）。DELETE可以删除指定的元素。使用下列任意方式，可以删除一定范围内的元素或全部元素：

- Tablename.DELETE (m) ； —— 仅删除一个元素项。
- Tablename.DELETE (m, n) ； —— 删除范围[m..n]内的全部元素。
- Tablename.DELETE ； —— 删除表中的全部元素。

如果在一个指定范围内的偏移量不含有元素项，PL/SQL简单地跳过它（不会发生例外）。范围必须是升序的，如果m>n，DELETE属性不做任何操作。

7. 变量数组和嵌套表

Oracle 8i定义了两类新类型的类数组结构或集合：变量数组或变长数组，以及嵌套表。它们不同于PL/SQL表，PL/SQL表现在称为“被索引的表”，区别在于如何使用它们以及它们主要的目的不同。被索引的表是PL/SQL结构，而数组和嵌套表则集中在数据库本身。例如，它们可以存储在数据库表中，就像它们是任意其他的数据列，但是它们不能够被捆绑到宿主变量中。

8. 创建变量数组

创建一个数组类似于创建一个PL/SQL表，因为首先需要声明类型，然后创建那种类型的一个变量。但是又不同于PL/SQL表，一个数组是“紧凑的”，不是“松散的”——在数组中没有“空洞”，空洞是指没有数据，这点与表完全不同。还有，PL/SQL表事实上是没有界限的，而数组有一个明确的限制。当创建一个数组时，数组为空，并且可以被测试为无效。为了能够使用它，必须首先使用一个构造器初始化它，如下列代码所示：

```
TYPE Performers IS VARRAY(250) OF varchar2(100);
TYPE Contacts is VARRAY(200) OF prospects;
showLineup Performers('Juggler', 'Singer', 'Fire Swallower');
myContacts Contacts := Contacts('John', 'Mary', 'Tito');
```

9. 嵌套表

还可以在SQL*Plus中创建类型，并将类型用做数据库中的嵌套表。这时，由数据库管理存储。

```
CREATE TYPE act AS OBJECT (
    act_no number(5),
    act_name varchar2(35),
    act_desc varchar2(500));

CREATE TYPE performers IS TABLE OF act;

CREATE TABLE schedules (evt_id number(5) PRIMARY KEY,
    evt_name varchar2(100),
    evt_date date,
    evt_location varchar2(100),
    evt_lineup performers)
NESTED TABLE evt_lineup STORE AS evt_lineup_tab;
INSERT INTO schedules VALUES (1138, 'Not Quite the Greatest Show On Earth',
sysdate+180, 'Colorado Springs', performers(act('1', 'John', 'Trick Riding'),
    act('2', 'Mary', 'High Wire'),
    act('3', 'Albert the Dog-faced Boy', 'Contortionist')));
```

处理每个元素涉及 Oracle8i 中一个新运算符的用法——TABLE()。TABLE() 返回一个变量数组，可以允许个别地访问数据，虽然它的语法看起来并不直观。要为上面显示的科罗拉多州春天表演添加一个节目，参阅下列的代码片段：

```
BEGIN
    INSERT INTO
        TABLE(SELECT evt_lineup FROM schedules WHERE evt_id = 1138)
        VALUES('12', 'The Amazing Rando', 'Magician');
END;
```

同样，可以使用这段代码修改信息：

```
BEGIN
    UPDATE TABLE(SELECT evt_lineup FROM schedules WHERE evt_id = 1138)
        SET act_desc = 'Ruler of the Known Universe'
        WHERE act_no = 12;
END;
```

使用这条语句，可以删除 Mary 的走钢丝表演：

```
BEGIN
    DELETE TABLE(SELECT evt_lineup FROM schedules WHERE evt_id = 1138)
        WHERE act_name = 'Mary';
END;
```

当然，使用下面的代码可以取出信息来查看：

```
DECLARE
    MyAct varchar2(35);
    MyDesc varchar2(500);
BEGIN
    SELECT act_name, act_desc INTO MyAct, MyDesc
        FROM TABLE(SELECT evt_lineup FROM schedules WHERE evt_id = 1138)
        WHERE act_no = 3;
    ...
END;
```

10. 使用子类型

在 Oracle 7.2 版 PL/SQL 2.2 中引入用户定义子类型。子类型让你能够为一个类型提供另外一种更有意义的名字，不受任何约束。它们不定义存储，仅仅是一个更通用数据类型的另外一个名称。例如，预定义子类型 NUMBER 和 CHAR 就使用了这种方式。不能直接使用一个固定长度约束一个子类型；相反，必须使用包含 %TYPE 和 %ROWTYPE 的两步方法。这里是一些例子（正确的和错误的）：


```

SUBTYPE SSN_TYPE IS VARCHAR2;           -- legal (method #1)
emp_ssn SSN_TYPE(9);                     -- legal
SUBTYPE PHONE_TYPE IS VARCHAR2(10);      -- illegal!
big_string VARCHAR2(32767);              -- will be used below
TYPE MAX_STRING_TYPE IS big_string%TYPE; -- legal (method #2)
dynam_str MAX_STRING_TYPE;               -- legal

```

注意方法#2，这是定义长度约束子类型的一种方法。另外一种方法是基于一个约束的数据库列类型定义变量。使用这种方法能够帮助你基于子类型捆绑检查变量。当合理地使用时，这种方式可以产生良好的自书写程序代码。

提示 通过基于数据库列进行变量定义（变量将包含数据库列的信息），可以帮助防止取值错误。如果更改表中某一列的大小或类型，代码将仍然是正确的。

```

small_str VARCHAR2(20) NOT NULL := 'blick'; -- #1
subtype SMALL_TYPE is small_str%TYPE; -- fails NOT NULL constraint!
tiny_str VARCHAR2(1) := 'T'; -- #2
subtype TINY_TYPE is tiny_str%TYPE;
tiny TINY_TYPE; -- does not inherit assignment!

```

只有子类按顺序基于相同的类型时，基于不同的非约束子类的变量出于赋值目的才是兼容的。例如，DECIMAL和NUMERIC作为NUMBER的子类，可以进行内部转换。如果父类不同，必须进行某种类型的转换。

11. 将一种类型转换为其他类型

在使用强类型语言时，有时需要将一种数据类型转换为其他数据类型。这通常发生在给变量赋值时或在计算一个表达式时。有两种方法可以实现类型转换：显式的和隐式的。

12. 隐式的类型转换

隐式的类型转换由PL/SQL编译器执行，你不需要做任何事情。例如，给出下面的例子：

```

DECLARE
  flt_x FLOAT(3,2) := 2.25;
  int_y INTEGER(6) := 100;
  flt_z DOUBLE(10,2);
BEGIN
  flt_z := flt_x + int_y; -- implicit numeric conversion
END;
/

```

为了正确地进行运算，整型值被隐式地转换为一个浮点值。这在编译器中很常见，为了能够执行计算，一个值被转换为最通用类型的值。在PL/SQL中，VARCHAR2数据类型是最通用的数据类型。

表28-5显示了全部合法的隐式类型转换。如果在列表中没有看到一个转换，你将不得不对它进行显式转换，需要使用内嵌的转换函数。

表28-5 隐式类型转换

To FROM	VARCHAR 2	CHAR	NUMBER	DATE	RAW	ROWID		
BINARY_		√		√	√	√		
INTEGER								
CHAR	√		√	√	√	√	√	√
DATE		√		√			√	
LONG		√			√		√	
NUMBER	√	√		√			√	
RAW		√		√			√	
ROWID		√					√	
VARCHAR2	√	√	√	√	√	√	√	√

PLS_INTEGER具有与BINARY_INTEGER相同的隐式转换特性，可以隐式地在它们之间进行类型转换。下面的例子都是合法的声明和赋值：

```
DECLARE
    date_from DATE := '20-AUG-85'; -- char to date (match NLS format)
    cnt INTEGER(3) := '0';          -- char to integer (subtype of number)
    loop_control BINARY_INTEGER;
    bin_val RAW(2) := '1';           -- char to raw
    str VARCHAR2(9) := '26-AUG-60'; -- char to varchar2
    short_num VARCHAR2(3);
BEGIN
    loop_control := cnt + '1';       -- char to number to binary integer
    str := date_from;                -- date to varchar2
    short_num := bin_val;             -- raw to varchar2
    cnt := short_num;                -- varchar2 to integer
```

任何值都可以被隐式地转换为CHAR和VARCHAR2类型，也可以从CHAR和VARCHAR2类型的值隐式地转换为任何值。如果需要将一个RAW值转换为NUMBER值，可以首先将RAW值转换为VARCHAR2值，然后再转换为NUMBER值。隐式转换同样支持一个数据类型的子类。

对于DATE（日期）类型，只有当字符串是为数据库定义的缺省NLS格式时，一个从CHAR或VARCHAR2类型的隐式转换才可能正确发生。如果字符串是不同格式，必须执行一个显式转换并要提供一个转换字符串。

13. 显式类型转换 转换表28-6列出了所有能够执行显式数据类型转换的PL/SQL函数。

表28-6 显式数据类型转换

从……到	VARCHAR2	CHAR	NUMBER	DATE	RAW	ROWID
VARCHAR2			TO_NUMBER (vc, [fmt (,lang)])	TO_DATE (vc, [fmt [, lang]])	HEXTORAW (vc)	CHARTOROWID (vc)
CHAR			TO_NUMBER (vc, [fmt [, (,lang)])	TO_DATE (vc, [fmt [, lang]])	HEXTORAW (c)	CHARTOROWID (c)
NUMBER	TO_CHAR (n, [format [,lang]])	TO_CHAR (n,[format [,lang]])		TO_DATE (n, [fmt [, lang]])		
DATE	TO_CHAR (dt, [format [,lang]])	TO_CHAR (dt, [format [,lang]])				
RAW	RAWTOHEX (raw)	RAWTOHEX (raw)				
ROWID	ROWIDTOCHAR (rowid)	ROWIDTOCHAR (rowid)				

其中：

■ fmt是一个格式字符串。

■ lang是NLS语言格式字符串。

实际上，TO_CHAR()返回VARCHAR2，但是由于CHAR和VARCHAR2两种类型可以隐式转换，所以返回值可以容易地用做CHAR。但是要记住，没有LONG类型的转换程序。如果要使用LONG，必须首先隐式地将它转换为VARCHAR2，然后显式地转换为其他类型的数据。

28.5.5 赋值

PL/SQL不像某些语言，例如C或Pascal，赋值必须是一条单独的语句。只能在一条语句里给一个变量赋值。不能在另一条语句内部给变量赋值，例如，在一个 IF条件测试语句中赋值。

只要可以决定表达式的类型，在编译时就可发现大多数数据类型不相符的错误。取值错误在运行时被发现。

在处理NULL（空）值时要特别注意。任意含有一个 NULL值的数学运算总要返回一个 NULL（空）值；根据定义，结果是不确定的。这点不适用于字符串连接。例如：

```
DECLARE
  x NUMBER;
  y NUMBER;
  z NUMBER := 10;
  a VARCHAR2(20) := 'Hello, ';
BEGIN
  x := 1/NULL;          -- result is NULL
  y := x * (z - 1);     -- result is NULL
  z := (y - 1)**10;     -- result is NULL
  a := a || NULL || 'World!'; -- result is 'Hello, World!'
```

28.5.6 循环

PL/SQL支持三种方式的循环：

- 非条件循环（Do-Forever循环）。
- 迭代循环（FOR循环）。
- 条件循环（WHILE循环）。

最简单的循环看起来就像这样：

```
LOOP
  NULL; -- infinite loop!
END LOOP;
```

注意LOOP..END LOOP的语法。Oracle不检查无限循环。必须进行测试以跳出循环。

有几种方式能够退出循环：

- EXIT —— 无条件跳出循环。在一个IF测试中使用它。
- EXIT WHEN —— 当给定的条件存在时，有条件退出循环。
- GOTO —— 跳出循环，到循环外部的语句。

显然，推荐使用头两种涉及EXIT的方法。这里有一些示例：

```
DECLARE
  i NUMBER := 0;
BEGIN
  LOOP -- example #1
    i := i + 1;
    IF (i >= 100) THEN -- enough iterations
      i := 0; -- reset
      EXIT; -- unconditional termination
    END IF; -- enough iterations
  END LOOP; -- example #1 done

  LOOP -- example #2
    i := i + 1;
    EXIT WHEN (i >= 100); -- conditional termination
  END LOOP; -- example #2 done
```

当然，当条件满足，你要做的就是终止循环时，EXIT WHEN语法更加清晰。在退出循环

前，当必须复位数值或做一些其他工作时使用条件逻辑。

1. 控制循环

可以使用 WHILE 循环在一个循环的开始处测试一个条件。只要条件为真，循环继续重复。相反，如果需要在循环的尾部测试一个条件，使用 LOOP-EXIT WHEN 结构：

```
WHILE (x < 10) LOOP  -- While loop
...
    x := x + 1;
END LOOP;  -- done
LOOP  -- simulated Repeat-Until (or Do-While) loop
...
    EXIT WHEN ...
END LOOP;  -- done
```

在循环的头部和尾部添加注意是一个良好的编程习惯，尤其当程序非常长时，就会知道为什么要使用注意了。

PL/SQL 不支持类 FORTRAN 的 CONTINUE 语句转到循环的结束，而跳过它们之间的所有代码。一些人会告诉你在这种情况下，使用条件逻辑模块化程序代码，会使程序代码变得令人讨厌，犬牙交错且难于维护。但是，这里有一个解决方案，可以使用 GOTO 语句模拟 CONTINUE 语句（参阅清单 28-11）。CONTINUE 是 SQL 中的一个保留字，所以不可以在一条语句中或标签中使用它。

清单 28-11 contine.sql——模拟

```
DECLARE
j NUMBER := 0;
BEGIN
    DBMS_OUTPUT.enable;
    LOOP  -- print even numbers between 0 and 20
        IF (MOD(j, 2) = 1) THEN  -- skip odd numbers
            GOTO CONTINE;  -- the misspelling is an inside joke
        END IF;  -- skipping odd numbers
        DBMS_OUTPUT.put_line(TO_CHAR(j) || ' is even');
        <<CONTINE>>
        EXIT WHEN j = 20;  -- done
        j := j + 1;  -- don't forget to increment the loop counter!
    END LOOP;  -- print even numbers between 0 and 20
END;
/
```

服务器响应如下信息：

```
0 is even
2 is even
4 is even
6 is even
8 is even
10 is even
12 is even
14 is even
16 is even
18 is even
20 is even
PL/SQL procedure successfully completed.
```

以这种方式使用 GOTO 语句，简化了一个循环中的逻辑关系。

2. 使用 FOR 循环进行迭代

另外一种循环执行指定数量的迭代：

```
BEGIN
  FOR i IN 1..100 LOOP -- do nothing for exactly 100 iterations
    NULL;
  END LOOP; -- done doing nothing
END;
```

注意FOR循环的语法，IN子句必须指定一个范围。在这个案例中，不需要声明循环控制变量（在本例中是i），因为它在FOR循环的作用域内自动创建。这意味着不能在循环的外部引用该变量，因为当循环完成时，它不再存在。如果出于某种原因需要循环变量的值，必须将它拷贝到其他的变量中（参见清单 28-12）。

清单28-12 lastodd.sql —— 获取循环控制变量的值

```
DECLARE
  j NUMBER;
BEGIN
  DBMS_OUTPUT.enable; -- enable output
  FOR i IN 1..100 LOOP -- do nothing for exactly 100 iterations
    IF (MOD(i, 2) = 1) THEN -- must be odd
      j := i;
    END IF; -- capture odd numbers
  END LOOP; -- done doing nothing
  DBMS_OUTPUT.put_line('last odd number was ' || TO_CHAR(j));
END;
/
```

服务器显示下列信息：

```
last odd number was 99
PL/SQL procedure successfully completed.
```

PL/SQL不支持不为1的任意其他步进值。但是可以模拟步进值不为1的情形！有两种方法：一种方法是使用步长乘以循环计数器，以得到需要用于处理的数值。不能真正地修改循环控制变量，因为这样做是非法的（它是严格的只读值）。另外一种方法是创建一个变量，并且使用希望的数值增长它，这样做或许会让一个简单的循环更加清楚，但是它在FOR循环里仍然是合法的。也许必须要调整循环的范围，以得到所希望的行为。当然，可以使用EXIT或EXIT WHEN提早退出一个循环。

可以采用倒数方式，反向执行重复（参见清单 28-13）。

清单28-13 countdwn.sql —— 在一个循环中使用

```
BEGIN
  FOR j IN REVERSE 1..10 LOOP -- countdown
    DBMS_OUTPUT.put(TO_CHAR(j) || '-');
  END LOOP; -- countdown
  DBMS_OUTPUT.put_line('Blastoff!');
END;
/
```

将得到下列信息：

```
10-9-8-7-6-5-4-3-2-1-Blastoff!
PL/SQL procedure successfully completed.
```

28.5.7 使用游标

游标是一个对象，能够提供行级的 SQL 语句控制。游标声明不同于变量，而是用于实现游标的一个内存区域的句柄。游标声明仅仅定义了将提交给 SQL 语句执行器（SQL Statement Executor）哪些查询，在可执行代码中的程序控制下进行查询的管理。游标可以表示任意合法的 SQL SELECT 语句。游标通常是任何 PL/SQL 应用的基本构建块，它们为数据库中存储的数据集合上的操作提供了循环机制。如果还需要进行更新，使用 FOR UPDATE 子句。

警告 当心，使用 FOR UPDATE 子句将锁定查询发现的所有记录。所有这些记录在游标被关闭前将保持被锁定状态。

游标可以返回一条或多条记录，或一条也不返回。通常的操作顺序是：

- 1) 声明游标，以及一个检索记录的数据结构。
- 2) 打开游标。
- 3) 从游标中重复提取每条记录到数据结构中，直到数据集合被提空。
- 4) 关闭游标。

1. 定义一个游标

在定义游标时，要注意几个主题上的变化：

```
DECLARE
  -- gets all orders in database
  CURSOR get_orders IS
    SELECT * FROM orders;
  -- gets a few columns for a specified order number
  CURSOR get_orditem(Pord_num orders.ord_num%TYPE) IS
    SELECT seq_num, quantity, unit_price, extended_price
    FROM orders
    WHERE ord_num = Pord_num;
  -- gets the whole row for a particular item#
  CURSOR get_items(Pitem_no items.item_no%TYPE) RETURN items%ROWTYPE IS
    SELECT * FROM items WHERE item = Pitem;

  -- gets the item name for a particular item#
  CURSOR get_item_name(Pitem_no items.item_no%TYPE)
  RETURN items.item_name%TYPE IS
    SELECT item_name FROM items WHERE item_no = Pitem_no;
```

虽然有不少的差别，但仅有两个基本模式。一个游标可以接受参数或不接受参数。当打开游标的时候，为参数提供数值。可以定义返回类型，或不定义返回类型。返回类型可以是用户定义记录、数据库表记录类型或独立的变量。不论哪种方式，在 SELECT 子句中定义的列必须与用于接受返回值的参数一一对应。

一旦游标被声明，就可以打开它、提取记录、检查它的状态并且在不再需要它的时候关闭它。

一个游标可以拥有多个名字，上面例子中的约定是在游标的名字后添加 _cur 或 _loop 后缀，或在游标的名字前使用 get_ 前缀（完全取决于个人的风格）。在游标名字中使用表的名字，或试图让它有意义（不要太罗嗦），例如 get_addresses 或 employee_by_dept_cur 这样的名字。

要清楚参数表只能输入数值，不能输出数值。这就是为什么既不需要也不允许参数流的原因。而且输入参数必须是标量数值。

当打开一个游标时，执行 SQL 程序并计算相应的数据集合。但是，没有记录被真正地返

回给程序。使用FETCH语句一次提取一条记录。在执行另外一条提取语句之前，被提取的记录保持为当前行。只能正向提取记录，在数据集中没有可以回退的控制。

2. 游标属性

游标具有表28-7所描述的属性。

表28-7 游标属性

属 性	返 回 值	说 明
ISOPEN	TRUE/FALSE (真假)	指出一个游标是打开的还是关闭的
FOUND	TRUE/FALSE (真假)	指出是否发现一条记录
NOTFOUND	TRUE/FALSE (真假)	指出是否没有发现一条记录
ROWCOUNT	NUMBER	每条提取记录的序数值 (第一、第二、第三...)

这里是它们使用的一些示例：

```

IF (orders_cur%FOUND) THEN  -- got an order
  OPEN items_cur(orders_cur.orderno);  -- open a related cursor
  LOOP
    FETCH items_cur into order_item;
    EXIT WHEN items_cur%NOTFOUND;  -- break out when done
    -- show how many rows were processed so far
    DBMS_OUTPUT.put_line('On Row #' || TO_CHAR(items_cur%ROWCOUNT));
  END LOOP;
END IF;  -- got an order
...
IF (items_cur%ISOPEN) THEN  -- close the cursor
  CLOSE items_cur;
END IF;  -- close an open cursor
    
```

3. 游标的FOR循环

一个游标的FOR循环是使用游标最简单的方式。在FOR循环范围内自动处理游标的打开、提取和关闭。也隐式定义返回记录变量，不能在循环范围以外引用返回记录变量。游标可以接受参数。既可以在声明部分声明游标，也可以在游标FOR循环本身的程序体中声明游标 (参见清单28-14)。

清单28-14 cfor1.sql —— 简单使用游标循环

```

DECLARE
  CURSOR get_tables IS
    SELECT * FROM user_tables;
BEGIN
  FOR get_tables_cur IN get_tables LOOP
    DBMS_OUTPUT.put_line(get_tables_cur.table_name);
  END LOOP;
END;
/
    
```

如果作为scott/tiger用户运行这段程序，将得到下列信息：

```

BONUS
DEPT
EMP
SALGRADE
PL/SQL procedure successfully completed.
    
```

行2~3定义游标。

行5定义了游标记录变量，它与游标一起使用。

行6使用点符号引用游标中指定的列。

如清单28-15所示，可以使用更为简明的方法得到相同的结果：

清单28-15 cfor2.sql —— 一个更为简单的循环

```
BEGIN
  FOR get_tables_cur IN (SELECT * FROM user_tables) LOOP
    DBMS_OUTPUT.put_line(get_tables_cur.table_name);
  END LOOP;
END;
/
```

区别在于游标也在循环内部定义。对于简单的程序，这样做非常好。

还可以接受一个参数并且查询一个特定的表（参见清单 28-16）。

清单28-16 cfor3.sql —— 接收参数的简单循环游标

```
DECLARE
  CURSOR get_tables(Powner all_tables.owner%TYPE) IS
    SELECT * FROM all_tables
      WHERE owner = Powner;
  local_owner all_tables.owner%TYPE := 'DEMO'; -- search criteria
BEGIN
  FOR get_tables_cur IN get_tables(local_owner) LOOP
    DBMS_OUTPUT.put_line(get_tables_cur.table_name);
  END LOOP;
END;
/
```

注意有效地使用基本变量，尤其在游标定义中，还要注意参数是怎样传递给游标的。这一次你得到下面的信息（因为目前只能查看模式 DEMO中的表，这些表对于用户 scott是可见的）：

```
CUSTOMER
DEPARTMENT
EMPLOYEE
JOB
LOCATION
SALARY_GRADE
SALES_ORDER
PL/SQL procedure successfully completed.
```

对于较大的、复杂的程序，在封装块范围内部定义游标（正如前面进行的），并且自己控制游标。

4. 打开、提取与关闭游标

从上一个例子开始，比较罗嗦的方法，看起来类似清单 28-17。

清单28-17 ofc.sql —— 自己来控制游标

```
DECLARE
  -- cursor definitions
  CURSOR get_tables(Powner all_tables.owner%TYPE) IS
    SELECT * FROM all_tables
      WHERE owner = Powner;
  -- record variable definitions
  get_tables_rec get_tables%ROWTYPE;
```

```

-- local variables
local_owner all_tables.owner%TYPE := 'DEMO'; -- search criteria
BEGIN
OPEN get_tables(local_owner);           -- compute rows to return
LOOP -- find all DEMO tables available to SCOTT
  FETCH get_tables INTO get_tables_rec; -- try to get a row
  EXIT WHEN get_tables%NOTFOUND;       -- no more rows
  DBMS_OUTPUT.put_line(get_tables_rec.table_name);
END LOOP;
CLOSE get_tables; -- done with this cursor
END;
/

```

你得到与先前完全相同的数据集。这次定义记录变量来接受基于游标的记录。或许你已经自己定义了一个游标（使用 TYPE RECORD IS…语句）。事实上，或许你已经定义了一个不同的变量来提取记录，但是基于游标的返回类型定义变量非常简单而且易于维护。现在，如果更改游标的列列表，除了需要修改涉及记录变量的东西之外，不需要修改任何东西。

在打开游标之后，游标将计算游标返回的记录数（虽然游标还没有真正返回任何东西），循环提取全部记录，每次提取一条记录，需要检测数据集的结尾并跳出循环，如第 14 行所示。当循环结束后，第 17 行代码关闭循环。为了能够再次使用这个游标，必须重新打开它，这一次打开游标的时候或许带有几个参数。

假设只关心一条记录，程序清单 28-18 显示了最简单的解决方案。

清单 28-18 onerow.sql —— 显式提取一条记录

```

DECLARE
-- cursor definitions
CURSOR get_tables(Powner all_tables.owner%TYPE) IS
  SELECT * FROM all_tables
  WHERE owner = Powner;
-- record variable definitions
get_tables_rec get_tables%ROWTYPE;
-- local variables
local_owner all_tables.owner%TYPE := 'DEMO'; -- search criteria
BEGIN
OPEN get_tables(local_owner);           -- compute rows to return
FETCH get_tables INTO get_tables_rec; -- try to get a row
IF (get_tables%FOUND) THEN             -- got a row
  DBMS_OUTPUT.put_line(get_tables_rec.table_name);
END IF;
CLOSE get_tables; -- done with this cursor
END;
/

```

这次得到下面信息：

```

CUSTOMER
PL/SQL procedure successfully completed.

```

注意游标属性 %FOUND 用于检查是否真正地提取到记录。如果没有返回任何记录，记录变量的值将与 FETCH 语句执行前的值相同。实际上，让我们来证明这点（参见清单 28-19）。

清单 28-19 badfetch.sql —— 仅仅成功地重写一个记录变量

```

DECLARE
-- cursor definitions

```

```
CURSOR get_tables(Powner all_tables.owner%TYPE) IS
  SELECT * FROM all_tables
  WHERE owner = Powner;
-- record variable definitions
get_tables_rec get_tables%ROWTYPE;
-- local variables
local_owner all_tables.owner%TYPE := 'BLICK'; -- unknown owner
BEGIN
  get_tables_rec.table_name := 'GARBAGE'; -- initialize to something
  OPEN get_tables(local_owner);           -- compute rows to return
  FETCH get_tables INTO get_tables_rec; -- try to get a row
  IF (get_tables%NOTFOUND) THEN           -- got a row
    DBMS_OUTPUT.put_line(get_tables_rec.table_name);
  END IF;
  CLOSE get_tables; -- done with this cursor
END;
```

你得到下面信息：

```
GARBAGE
PL/SQL procedure successfully completed.
```

这个结果证明了一点：如果FETCH语句失败，它不会重写它提取的任何东西。

5. 使用隐式游标

隐式游标相对来说需要较少的维护。只有当你预料将得到一条记录时，才会使用隐式游标。如果找到0条或多于1条的记录，会出现NO_DATA_FOUND或TOO_MANY_ROWS例外，它并不会给你带来任何麻烦。程序清单 28-20展示了一个活动的隐式游标。

清单28-20 一个简单的隐式游标

```
DECLARE
  -- record variable definitions
  get_tables_rec all_tables%ROWTYPE; -- based on the table
  -- local variables
  local_owner all_tables.owner%TYPE := 'DEMO'; -- search criteria
  local_table all_tables.table_name%TYPE := 'CUSTOMER';
BEGIN
  SELECT * INTO get_tables_rec -- looking for a single row
  FROM ALL_TABLES
  WHERE owner = local_owner AND table_name = local_table;
  DBMS_OUTPUT.put_line(get_tables_rec.tablespace_name);
END;
```

你得到如下信息：

```
USER_DATA
PL/SQL procedure successfully completed.
```

这就是你没有构造查询，以便能够准确地返回一条记录所发生的事情（参见程序清单 28-21）。

清单28-21 toomany.sql —— 当发现多条记录时，隐式游标失效

```
DECLARE
  -- record variable definitions
  get_tables_rec all_tables%ROWTYPE; -- based on table
  -- local variables
  local_owner all_tables.owner%TYPE := 'DEMO';
```

```
BEGIN
  SELECT * INTO get_tables_rec FROM ALL_TABLES
  WHERE owner = local_owner;
  DBMS_OUTPUT.put_line(get_tables_rec.tablespace_name);
END;
/
```

服务器反馈下面一段有意义的错误消息：

```
ERROR at line 1:
ORA-01422: exact fetch returns more than requested number of rows
ORA-06512: at line 7
```

这是实际发生的事情：

- 1) Oracle打开游标并提取一条记录。
- 2) 没有满足条件，Oracle试图进行另外一次提取操作并得到另外一条记录。
- 3) 因为你不能将10磅的记录放入5磅的变量中，Oracle关闭游标（游标会以任何手段关闭自己）并且产生例外。

提示 Oracle通常在隐式游标上进行两次提取，只是为了查看游标是否返回另外一条记录（并且因此导致请求失败，因为一个隐式游标确切地只返回一条记录）。这是ANSI标准的一个要求。

当全部需求就是一条记录，并且不希望被例外弄乱时，需要使用一个显式游标并且使用一条FETCH语句。如果希望知道是否有多于1条的记录（或根本没有记录），并且不在乎处理例外，那么使用隐式游标。事实上，可以在一个单行查询程序中使用隐式游标，就是出于这个原因。需要知道单行查询是否确实就是一条记录，或者察觉到它有错误。如果只关心得到一条记录，使用显式游标处理起来效率更高。

隐式游标不能使用数组处理额外的记录。根据定义，所有的隐式游标都被假设为只返回一条记录。

28.5.8 例外处理

例外是一个非致命事件，它立即中断程序的正常执行并引起一个非条件转移，跳转到当前程序块的例外处理部分。一些例外，像NO_DATA_FOUND或TOO_MANY_ROWS，可以被认为是正常的处理部分。像VALUE_ERROR这样的例外表明一个程序错误或一些意料之外的事件。还有一些例外表明一个严重的问题，例如耗光内存。

如果没有给一个程序块定义例外处理程序，例外将被返回给下一个更高层的块（如果有的话）。例外从封装的块中“冒出”，直到为该块找到一个例外处理程序，或将控制返回给调用的环境（在我们这里的案例中是SQL*Plus）。

你的程序将始终经历例外，你所需要的实际就是一个例外处理程序，用它来处理这些例外。再次参见上次出现问题的隐式游标（参见程序清单28-22）。

清单28-22 gracefule.sql —— 使用例外处理程序

```
DECLARE
  -- record variable definitions
  get_tables_rec all_tables%ROWTYPE; -- based on table
  -- local variables
  local_owner all_tables.owner%TYPE := 'DEMO';
  status NUMERIC := 0; -- capture error code (initialize to OK)
```

```
BEGIN
  SELECT * INTO get_tables_rec FROM ALL_TABLES
  WHERE owner = local_owner;
  DBMS_OUTPUT.put_line(get_tables_rec.tablespace_name);
EXCEPTION
  WHEN TOO_MANY_ROWS THEN
    status := SQLCODE;
    DBMS_OUTPUT.put_line('get_tables: ' || SQLERRM(status));
    DBMS_OUTPUT.put_line('get_tables: Exiting gracefully.');
```

这一次，服务器给出下列信息：

```
get_tables: ORA-01422: exact fetch returns more than requested number of rows
get_tables: Exiting gracefully.
PL/SQL procedure successfully completed.
```

在第6行上，定义了一个变量用于存储潜在的错误代码。

在第12行上，在例外部分中使用一个 WHEN 子句指定例外。

在第13行上，捕获到错误代码。

在第14行上，显示与错误代码相关的错误消息。

服务器现在告诉你程序已经成功地完成。如果没有出现错误，SQLCODE 将被设置为零，零意味着一切正常。SQLCODE 实际上就是一个返回 SQL 状态的内建函数。

表28-8是一个预定义例外的列表。

表28-8 可以在例外块中被显式地测试的例外

例外名称	Oracle错误代码	值
CURSOR_ALREADY_OPEN	ORA-06511	-6511
DUP_VAL_ON_INDEX	ORA-00001	-1
INVALID_CURSOR	ORA-01001	-1001
INVALID_NUMBER	ORA-01722	-1722
LOGIN_DENIED	ORA-01017	-1017
NO_DATA_FOUND	ORA-01403	+100
NOT_LOGGED_ON	ORA-01012	-1012
PROGRAM_ERROR	ORA-06501	-6501
STORAGE_ERROR	ORA-06500	-6500
TIMEOUT_ON_RESOURCE	ORA-00051	-51
TOO_MANY_ROWS	ORA-01422	-1422
TRANSACTION_BACKED_OUT	ORA-00061	-61
VALUE_ERROR	ORA-06502	-6502
ZERO_DIVIDE	ORA-01476	-1476

注意大多数例外如何拥有负的 SQLCODE 数值，而 NO_DATA_FOUND 是最明显的例外（没有故意使用双关语）。这是因为 ANSI 标准要求当没有记录被发现时，SQLCODE 的数值应该是 +100，即使 Oracle 错误代码都是负的。Oracle 是一个 ANSI 兼容数据库。你或许会说它使得它们 ANSI 化了（不好的双关语）。

如果在上面的列表中不包括预期的例外，或者不能确定将会发生什么例外，或者除了处理它们并不关心它们，可以使用 WHEN OTHERS 子句，这是一个捕捉器，允许例外处理器处理可能会出现的任意错误。

还可以使用任意例外名的组合，并且使用 WHEN OTHERS THEN 子句结束名称组合列表。

它们在一起的作用类似一组 IF ...ELSIF...ELSE 语句。如果它们中的任何一个为真，相关的程序代码将被执行，并且程序块将被退出：

```
EXCEPTION
  WHEN NO_DATA_FOUND THEN -- row not found!
    -- do this and skip to end of block
    ...
  WHEN ZERO_DIVIDE THEN -- divide by zero!
    -- do that and skip to end of block
    ...
  WHEN VALUE_ERROR THEN -- 10 lbs of data in a 51b var
    -- do these and skip to end of block
    ...
  WHEN OTHERS THEN -- dunno, don't care
    -- do the other and skip to end of block
    ...
END;
```

提示 一般要在例外处理器尾部编写一个 WHEN OTHERS 子句。有时候，它还是唯一的处理程序部分。

1. 避免在一个例外处理器中的无限循环

有时，执行一个例外处理程序中的动作，动作本身就有可能产生例外。这会导致一个无限循环！避免出现这种现象的窍门是将例外处理程序封装在一个块中，并且给它自己的例外处理器。这种内嵌的例外处理器不需要（也不应该）做任何事情。参考下面代码：

```
EXCEPTION -- main block handler
  WHEN OTHERS THEN
    BEGIN -- embedded block
      IF (get_cursor%ISOPEN) THEN -- cursor left open
        CLOSE get_cursor; -- close it
      END IF; -- cursor left open
    EXCEPTION
      WHEN OTHERS THEN
        NULL; -- don't care
    END; -- embedded block
END; -- main block
```

例外处理程序应该执行全部必要的清除工作，例如关闭可能被打开的游标。

提示 一般将一个例外处理器放置在任何应用 PL/SQL 代码的最顶层块中。例外应该以良好的方式处理。用户需要查看的最后一件事情就是一个简洁的、技术性的错误消息。

2. 定义你自己的例外

如果预定义例外列表看起来有些简略，通过为任意特定的 Oracle 例外序号关联一个名称，可以更多地定义它们。然后可以在例外程序块中通过名字引用这些例外。要实现这项工作，定义一个例外和一个注记，传递给编译器，告诉你想使用给定的错误代码关联例外。一个注记就是一个编译指示，在编译时处理而不是在运行时处理。它指示编译器应该如何处理特定语言信息或条件。所有的优秀编译器都具有注记（或者类似的东西）。程序清单 28-23 是使用注记的一个示例。

清单 28-23 excpinit.sql —— 使用 EXCEPTION_INIT

```
SET SERVEROUTPUT ON
DECLARE
```

```

-- exceptions and pragmas (compiler directives)
INVALID_NUM_FORMAT EXCEPTION; -- first we define an exception object
PRAGMA EXCEPTION_INIT (INVALID_NUM_FORMAT, -1481); -- then associate it with
--an exception
-- constants
NUM_FMT CONSTANT VARCHAR2(3) := 'aaa'; -- an invalid number format
x NUMBER(10);
BEGIN
  DBMS_OUTPUT.enable;
  SELECT TO_NUMBER('999', NUM_FMT) INTO x FROM DUAL; -- try to convert
EXCEPTION
WHEN INVALID_NUM_FORMAT THEN
  DBMS_OUTPUT.put_line('Trapped an illegal Number Conversion');
WHEN OTHERS THEN
  DBMS_OUTPUT.put_line('Some other error');
END;
/

```

这次，你得到下列信息：

```

Trapped an illegal Number Conversion
PL/SQL procedure successfully completed.

```

这里看到一个两步过程：首先定义了一个例外，然后将它与一个 Oracle 错误代码关联。下一个问题是，为什么 Oracle 不预先定义全部的错误代码？全部 24 000 个代码吗？Richard Bach（理查德·巴克）曾经写过，“如果你正确地提问问题，问题本身就回答了它自己。”Oracle 仅仅定义了带来问题的 80% 的错误。毕竟，这些东西确实占用内存空间。

另外一种方式是定义应用的特定例外。这与 Oracle 错误代码无关，它们是为应用定义的错误。必须检测违犯一个特定商务规则的情形，并且产生合理的用户定义例外。例如，假设非免税雇员，他一小时挣钱少于 \$10，只能允许加薪百分之 1 至 5。你的朋友，Joe（乔），是一个计时雇员，一小时挣 8 美元，他的老板（也是他的叔叔）希望给他大幅度加薪。程序清单 28-24 显示了这个商务规则的一个实现。

清单 28-24 busrule.sql —— 使用应用特定的例外捕获违反商务规则的操作

```

DECLARE
-- exceptions
INVALID_ANNUAL_RAISE EXCEPTION;
-- constants
HRLY_STATUS CONSTANT VARCHAR2(1) := 'H'; -- hourly status
HRLY_WAGE_LIMIT CONSTANT REAL(4,2) := 10.00; -- upper hourly wage limit
MIN_HRLY_RAISE CONSTANT REAL(5,4) := 1.0100; -- lower hourly raise limit
MAX_HRLY_RAISE CONSTANT REAL(5,4) := 1.0500; -- upper hourly raise limit
-- info entered by user
emp_name VARCHAR2(20) := '&emp_name'; -- employee name
emp_status VARCHAR2(1) := '&emp_status'; -- employee status
hrly_wage REAL := &wage; -- hourly wage
ann_raise REAL := &annual_raise; -- annual raise
BEGIN
  IF (emp_status = HRLY_STATUS AND -- this is business rule #532
      hrly_wage < HRLY_WAGE_LIMIT AND
      ann_raise NOT BETWEEN MIN_HRLY_RAISE AND MAX_HRLY_RAISE) THEN
    RAISE INVALID_ANNUAL_RAISE;
  ELSIF (FALSE) THEN -- code other rules here (pertaining to raises)
    NULL;
  ELSE -- passed the gauntlet
    DBMS_OUTPUT.put_line(emp_name || ' now makes ' ||
                          TO_CHAR(hrly_wage * ann_raise, '$99.99'));
  END IF;
END;

```

```

END IF;
EXCEPTION
WHEN INVALID_ANNUAL_RAISE THEN
  DBMS_OUTPUT.put_line('Don't give ' || emp_name || ' the raise!');
WHEN OTHERS THEN
  DBMS_OUTPUT.put_line('Some other problem computing ' || emp_name || ' 's
increase');
END;
/

```

服务器以下列方式处理乔老板的需求：

```

Enter value for emp_name: Joe
old 10:  emp_name  VARCHAR2(20) := '&emp_name'; -- employee name
new 10:  emp_name  VARCHAR2(20) := 'Joe';      -- employee name
Enter value for emp_status: H
old 11:  emp_status VARCHAR2(1)  := '&emp_status'; -- employee status
new 11:  emp_status VARCHAR2(1)  := 'H'; -- employment status
Enter value for wage: 8.00
old 12:  hrly_wage REAL(4,2)      := &wage;          -- hourly wage
new 12:  hrly_wage REAL(4,2)      := 8.00;          -- hourly wage
Enter value for annual_raise: 1.06
old 13:  ann_raise REAL(5,4)      := &annual_raise;  -- annual raise
new 13:  ann_raise REAL(5,4)      := 1.06;          -- annual raise
Don't give Joe the raise!
PL/SQL procedure successfully completed.

```

使用应用特定的例外并在你的逻辑程序中引发它们是一个基本的商务规则编码技术。通过建立一个每个人都可以遵循的模板，将代码设计得更具有扩展性与易维护性。

如果希望同一个代码执行两个或多个例外，只需要在一个布尔 OR 条件中将它们列出，如下所示：

```

EXCEPTION
WHEN INVALID_NUMBER OR VALUE_ERROR THEN
  ...
WHEN OTHERS THEN
  ...
END;

```

3. 处理在线例外

需要养成的好习惯是使用例外处理程序为所有的 SQL 语句编写程序块。如果一个例外发生在 SQL 语句上，通常希望继续处理在线程序（继续处理下一个概念上的程序块）。这点对于循环尤其正确。如果没有为一个循环内部 SQL 语句编写带有例外处理程序的代码块，任何例外将导致循环立即终止，程序跳转到封装的块例外处理程序（“冒泡”带来的问题）。为了表明这点，请看下面的代码：

```

LOOP -- get some rows from a master table
  FETCH master_cursor INTO master_rec;
  EXIT WHEN master_cursor%NOTFOUND;
  BEGIN -- delete some child table rows
    -- master primary key is foreign key in child
    DELETE FROM child_table
    WHERE master_fkey = master_rec.master_pkey;
  EXCEPTION -- something untoward occurred
    WHEN OTHERS THEN -- so output a message, say
      status := SQLCODE; -- always capture error code!
      DBMS_OUTPUT.put_line('during delete: ' || SQLERRM(status));
  END; -- delete some child table rows
END LOOP;

```

如果带有例外处理程序的封装块不存在，一个例外（例如没有发现记录）将立即跳出循环。所希望的事情是程序跳过这一行，继续执行下一条程序。处理在线例外满足了这种需求。

28.5.9 使用子程序

现在继续考虑子程序。子程序使你能够模块化程序代码，并让代码更为有效。

1. 什么是子程序？

子程序是可以被调用一次或多次的子例程。我通常使用术语“子程序”，交替使用子例程。实际上，子程序是Oracle使用的术语，我只是更熟悉它而已。

PL/SQL拥有两种类型的子程序（或子例程）：过程和函数。在一个未命名的PL/SQL块中，在声明部分声明（给予名称和可选参数及类型）并定义（给出可执行代码）函数和过程。可以认为它们是可调用静态代码段。为了能够执行子程序包含的代码，过程和函数必须在执行时被引用。这些子程序，像大多数程序块一样，包含声明（除了用子程序的名字替换关键字DECLARE之外），程序体和可选的例外处理部分，并以END语句结束，有时使用子例程的名字作为标签。

2. 函数

函数是一个子程序，它的名字返回一个某一特定数据类型的数值。可以将这个返回值赋值给相同类型的一个变量，或者在一个表达式中使用该值。函数通常接受参数。当没有参数传递时，不需要使用圆括号。一些简单并且非常熟悉的例子如下所示：

```
status := SQLCODE; -- note no parentheses
NL CONSTANT VARCHAR2(1) := CHR(10); -- ASCII character (newline)
IF (employee_exists('Scaboda')) THEN ... -- use them in expressions
```

在一个未命名PL/SQL块中，在块的声明部分，跟随其他的声明之后，同时声明和实现子程序。程序清单28-25显示了一些函数例子。

清单28-25 boolsub.sql —— 函数使重复性工作变得简单

```
DECLARE
  -- local variables
  x NUMBER(3);
  -- local subprograms (must follow all other declarations!)
  FUNCTION is_even(Pnum IN x%TYPE) -- returns TRUE if number is even
  RETURN BOOLEAN IS
  BEGIN
    RETURN(MOD(Pnum, 2) = 0); -- test for evenness
  EXCEPTION
  WHEN OTHERS THEN
    RETURN (NULL); -- indeterminate
  END is_even;
  FUNCTION bool_to_char(Pbool IN BOOLEAN)
  RETURN VARCHAR2 IS
    str VARCHAR2(5); -- capture string to return
  BEGIN
    IF (Pbool) THEN -- test Boolean value for TRUE
      str := 'TRUE';
    ELSIF (NOT Pbool) THEN -- FALSE
      str := 'FALSE';
    ELSE -- must be NULL
      str := 'NULL';
    END IF; -- test Boolean value
    RETURN (str);
  END bool_to_char;
```

```
BEGIN -- executable code
  x := 0;
  DBMS_OUTPUT.put_line('It is ' || bool_to_char(is_even(x)) ||
    ' that ' || TO_CHAR(x) || ' is even');

  x := 1;
  DBMS_OUTPUT.put_line('It is ' || bool_to_char(is_even(x)) ||
    ' that ' || TO_CHAR(x) || ' is even');

  x := 2;
  DBMS_OUTPUT.put_line('It is ' || bool_to_char(is_even(x)) ||
    ' that ' || TO_CHAR(x) || ' is even');

  x := 3;
  DBMS_OUTPUT.put_line('It is ' || bool_to_char(is_even(x)) ||
    ' that ' || TO_CHAR(x) || ' is even');

  x := 4;
  DBMS_OUTPUT.put_line('It is ' || bool_to_char(is_even(x)) ||
    ' that ' || TO_CHAR(x) || ' is even');

  DBMS_OUTPUT.put_line(bool_to_char(NULL) || ' is neither TRUE nor FALSE');
END;
/
```

服务器输出下列信息：

```
It is TRUE that 0 is even
It is FALSE that 1 is even
It is TRUE that 2 is even
It is FALSE that 3 is even
It is TRUE that 4 is even
NULL is neither TRUE nor FALSE
PL/SQL procedure successfully completed.
```

这个相当长的例子说明了函数的一些属性和限制：

- 函数（实际上全部子程序）必须在全部类型、常量和变量之后声明。
- 函数的局部变量在 IS...BEGIN 之间声明。
- 参数可以是基本量，当表示数据库列值时，这是推荐使用的方法。
- RETURN 子句必须用于返回一个值，即使在 EXCEPTION 子句中也不例外。
- 实际上，函数调用可以嵌套，并且可以在一个任意复杂的表达式中调用函数。
- 给予函数的 END 语句与函数名称一个相同的标签（虽然它是可选的，我却总是这样做，而且必须一致）。
- 每个参数拥有一个模式，一个关键字指定参数是否被当作只读（IN）、只写(OUT)或读写（IN OUT）变量。如果不指定模式，函数缺省模式为 IN。

注意函数冗长的声明。参数变量的作用范围被严格地限定为函数或函数内嵌套的任意子程序的局部变量。通过在 IS...BEGIN 部分定义子程序，可以嵌套其他的子程序，只有当程序执行进入封装的子程序，嵌套的子程序才存在。限制作用范围到最小程序单元的技术帮助减少程序例程之间的连接，这是一件好事情。

虽然从技术角度来说，在参数表中返回一个值是合法的（通过指定模式为 OUT 或 IN OUT），但通常不赞成这种不好的编程风格。如果要这样做，必须说明原因并且必须注明函数的准确用法，将它提升到一个标准的层次。这会防止滥用和不标准的编程风格（有人可能争论函数从参数表中返回值就已经是一种滥用风格了）。

提示 大多数 Oracle 开发者避免从函数的参数表中返回值。为了防止令人讨厌的、不可维护的代码增加，你也许希望彻底禁止这种习惯。

在一个函数中封装小块的代码，甚至只有一行，是非常正常且可以接受的。这种技术减小了代码尺寸并且简化了代码的维护。例如，如果以后需要为函数添加一些额外的数据确认，只需要在一个地方添加它即可。

3. 过程

过程是一个子程序，可以执行一定量的重复工作，能够严格地通过参数表传入和传出值。一个过程总是一条语句，不能在一个表达式中插入过程。函数作用范围规则也适用于过程。语法几乎和函数完全相同，只是少了返回语句。

如果没有指定模式，过程的模式缺省为 IN。作为一种风格，我总是指定模式。编译器将捕获一个参数的使用是否与它的模式相一致。IN参数必须出现在一个赋值语句的右边，或在任何能被求值的表达式中。OUT参数只能出现在一条赋值语句的左边。IN OUT参数可以出现在任何地方。

通常应使用子程序所需的最具限制性的模式。这会提供最大限度的程序错误保护，例如在一个子程序执行期间，不小心删除了变量。

当使用模式的时候，服务器复制一份变量用于处理，所以如果产生一个例外的话，变量的原始值被保留下来。但是有些时候，这并不是你所希望的，因为无论如何这会造成非常显著的开销。Oracle8i提出NOCOPY修改量，它让服务器按引用直接指出原始值。如果坚持使用OUT变量并且不在乎保留原始值，通过走这条路可以得到显著的性能改善。

尝试将全部的单行查找程序编写为过程。依照相同的编程风格来处理单行查询，传入键值，一个返回记录类型，以及一个状态指示器。这种风格具有以下优点：为了能够处理在线例外，封装了SQL语句或游标，还可以让程序代码段能够被再利用。它还是一个易于维护的风格。程序清单28-26显示了这种风格的一个例子。

清单28-26 table.sql —— 标准的编程风格使其成为可维护代码

```
DECLARE
  -- constants
  TB CONSTANT VARCHAR2(1) := CHR(9); -- TAB
  -- variables
  status NUMERIC;
  table_rec all_tables%TYPE;
  -- routines
  PROCEDURE get_table(Powner IN      all_tables.owner%TYPE,
                     Ptable IN      all_tables.table_name%TYPE,
                     Prec          OUT all_tables%TYPE,
                     Pstatus IN OUT NUMERIC) IS
    -- local cursors
    CURSOR table_cur(Cowner all_tables.owner%TYPE,
                    Ctable all_tables.table_name%TYPE) IS
      SELECT *
      FROM all_tables
      WHERE owner = Cowner AND table_name = Ctable;
    -- local variables
    Lowner all_tables.owner%TYPE;
    Ltable all_tables.table_name%TYPE;
BEGIN
  Pstatus := 0; -- OK
  Lowner := UPPER(Powner);
  Ltable := UPPER(Ptable);
  OPEN table_cur(Lowner, Ltable);
  FETCH table_cur INTO Prec;
  IF (table_cur%NOTFOUND) THEN
```



```

        RAISE NO_DATA_FOUND;
    END IF;
    CLOSE table_cur;
EXCEPTION
WHEN OTHERS THEN
    BEGIN
        Pstatus := SQLCODE; -- capture error code
        IF (table_cur%ISOPEN) THEN -- close the open cursor
            CLOSE table_cur;
        END IF;
        Prec := NULL; -- clear return values and display input values
        DBMS_OUTPUT.put_line('get_table: ' || SQLERRM(Pstatus));
        DBMS_OUTPUT.put_line('OWNER = ' || '<' || Lower || '>');
        DBMS_OUTPUT.put_line('TABLE = ' || '<' || Ltable || '>');
    EXCEPTION
    WHEN OTHERS THEN
        NULL; -- don't care (avoid infinite loop)
    END;
END get_table;
BEGIN -- display storage parameters for a given table
    DBMS_OUTPUT.enable;
    DBMS_OUTPUT.put_line('TABLE' || TB || 'TABLESPACE' || TB ||
        'INITIAL' || TB || 'NEXT' || TB || 'MAX');
    DBMS_OUTPUT.put_line(RPAD('-', 43, '-')); -- just an underline
    get_table('scott', 'dept', table_rec, status);
    IF (status = 0) THEN
        DBMS_OUTPUT.put_line(
            table_rec.table_name || TB ||
            table_rec.tablespace_name || TB ||
            table_rec.initial_extent || TB ||
            table_rec.next_extent || TB ||
            table_rec.max_extents);
    END IF;
    get_table('scott', 'garbage', table_rec, status);
    IF (status = 0) THEN
        DBMS_OUTPUT.put_line(
            table_rec.table_name || TB ||
            table_rec.tablespace_name || TB ||
            table_rec.initial_extent || TB ||
            table_rec.next_extent || TB ||
            table_rec.max_extents);
    END IF;
END;
/

```

服务器返回下面的信息：

```

TABLE    TABLESPACE    INITIAL NEXT    MAX
-----
DEPT     USER_DATA        10240  10240  121
get_table: ORA-01403: no data found
OWNER = <SCOTT>
TABLE = <GARBAGE>
PL/SQL procedure successfully completed.

```

如果希望使用一个唯一键的精确匹配，需要自己来管理游标并精确地执行依次提取。当发现没有记录时，在例外处理程序中关闭游标，而不是在条件语句块内部关闭游标（无论如何都要在例外程序块中关闭游标，为什么要编写三遍程序呢？）。注意，必须引发预先定义的例外NO_DATA_FOUND，因为提取动作不会自动地生成一个例外。输入值被转换为大写形式，使用局部变量，因为被转换后的值将在几个地方用到。

还要注意例外处理程序显示的额外信息。为什么不抓住机会显示例外发生时的键值呢？当处理大量的记录时，这样做尤其有价值。这条信息还可以被转储到一个错误表，便于事后深入分析。

你也许认为，“我可以使用一条简单的 SELECT 语句得到相同的信息。这样做能给我带来什么？”。在较大的事物模式中，固定的查询更有效率，因为可以在公共 SQL 区找到它们，并且能够再次使用它们。手工控制游标的单行提取确实更有效率，尤其当游标一次又一次运行了几千遍时。记住，目标是编写效率高的应用。在得到记录后，可以程序化地做任何希望的事来处理它。你拥有全部的灵活性和控制权，还有底层曾经编写且可重用的过程。

清单28-27显示了一个含有数字值的 PL/SQL 表上实现二进制查找的例子。

清单28-27 bintest.sql —— 过程使一个二进制查找程序易于使用

```
SET SERVEROUTPUT ON
DECLARE
  -- constants
  FIXED_TOP CONSTANT NUMBER := 12; -- fixed # of elements
  -- data types
  TYPE NUMARR_TYPE IS TABLE OF NUMBER INDEX BY BINARY_INTEGER;
  -- global variables
  numarr NUMARR_TYPE;
  isfound BOOLEAN;
  rowout NUMBER;

  -- routines
  PROCEDURE binary_search( -- binary search on sorted array
    Parr IN NUMARR_TYPE,
    Pnum IN NUMBER,
    Pfound OUT BOOLEAN,
    Prow OUT NUMBER) IS
    local_found BOOLEAN := NULL;
    top BINARY_INTEGER := FIXED_TOP;
    bottom BINARY_INTEGER := 1;
    middle BINARY_INTEGER := NULL;
  BEGIN
    local_found := FALSE;
    LOOP -- binary search
      middle := ROUND((top + bottom) / 2); -- find middle
      IF (Parr(middle) = Pnum) THEN -- exact match
        local_found := TRUE; -- match succeeded
        EXIT; -- break
      ELSIF (Parr(middle) < Pnum) THEN -- GO UP
        bottom := middle + 1;
      ELSE -- GO DOWN
        top := middle - 1;
      END IF; -- test for match
      IF (bottom > top) THEN -- search failed
        IF (Pnum > Parr(middle)) THEN
          middle := middle + 1; -- MAY BE OUTSIDE ARRAY!
        END IF; -- insert after
        EXIT;
      END IF; -- failed
    END LOOP; -- search
    Pfound := local_found;
    Prow := middle;
  EXCEPTION
    WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE(SQLERRM(SQLCODE));
      DBMS_OUTPUT.PUT_LINE(TO_CHAR(middle));
  END binary_search;
```

```

FUNCTION bool_to_char(Pbool IN BOOLEAN) -- convert Boolean to char
RETURN VARCHAR2 IS
  str VARCHAR2(5); -- capture string to return
BEGIN
  IF (Pbool) THEN -- test Boolean value for TRUE
    str := 'TRUE';
  ELSIF (NOT Pbool) THEN -- FALSE
    str := 'FALSE';
  ELSE -- must be NULL
    str := 'NULL';
  END IF; -- test Boolean value
  RETURN (str);
END bool_to_char;
BEGIN -- bintest executable code
  DBMS_OUTPUT.enable;
  numarr(1) := 100; -- fill array with numbers in order
  numarr(2) := 103;
  numarr(3) := 104;
  numarr(4) := 108;
  numarr(5) := 110;
  numarr(6) := 120;
  numarr(7) := 121;
  numarr(8) := 122;
  numarr(9) := 130;
  numarr(10) := 140;
  numarr(11) := 145;
  numarr(12) := 149;
  binary_search(numarr, 90, isfound, rowout);
  DBMS_OUTPUT.put_line('FOUND=' || bool_to_char(isfound) ||
    ', ROW=' || TO_CHAR(rowout) || ' SB=1');
  binary_search(numarr, 150, isfound, rowout);
  DBMS_OUTPUT.put_line('FOUND=' || bool_to_char(isfound) ||
    ', ROW=' || TO_CHAR(rowout) || ' SB=13');
  binary_search(numarr, 100, isfound, rowout);
  DBMS_OUTPUT.put_line('FOUND=' || bool_to_char(isfound) ||
    ', ROW=' || TO_CHAR(rowout) || ' SB=1');
  binary_search(numarr, 145, isfound, rowout);
  DBMS_OUTPUT.put_line('FOUND=' || bool_to_char(isfound) ||
    ', ROW=' || TO_CHAR(rowout) || ' SB=11');
  binary_search(numarr, 108, isfound, rowout);
  DBMS_OUTPUT.put_line('FOUND=' || bool_to_char(isfound) ||
    ', ROW=' || TO_CHAR(rowout) || ' SB=4');
  binary_search(numarr, 105, isfound, rowout);
  DBMS_OUTPUT.put_line('FOUND=' || bool_to_char(isfound) ||
    ', ROW=' || TO_CHAR(rowout) || ' SB=4');
END; -- bintest
/

```

服务器的输出如下所示：

```

FOUND=FALSE, ROW=1 SB=1
FOUND=FALSE, ROW=13 SB=13
FOUND=TRUE, ROW=1 SB=1
FOUND=TRUE, ROW=11 SB=11
FOUND=TRUE, ROW=4 SB=4
FOUND=FALSE, ROW=4 SB=4
PL/SQL procedure successfully completed.

```

注意OUT参数模式，它意味着只输出。这表示在过程内部，只能写入这个变量。如果需要对一个参数变量进行读写操作，使用IN OUT模式声明变量。

二进制到字符转换程序看起来有些面熟是不是？它是不是非常好，可以被复制到每个需要它的PL/SQL程序中？

4. 缺省参数值

参数可以接收缺省值，当在实际调用子程序而没有提供参数值时，就要使用缺省值。这会让子程序看起来好像它具有参数变量表一样：

```
DECLARE
... -- types, constants, variables
FUNCTION get_data (Pkey IN CHAR,
                  Pflag IN BOOLEAN DEFAULT FALSE,
                  Psort IN CHAR DEFAULT ' ')
RETURN VARCHAR2 IS
... -- function implementation
BEGIN -- executable code
IF get_data(key1) THEN -- valid call (Pflag, Psort defaulted)
...
ELSIF get_data(key2, TRUE) -- valid call (Psort defaulted)
...
ELSIF get_data(key3, , 'ASCENDING') THEN -- invalid!
```

注意关键字DEFAULT的使用。还可以使用赋值运算符（:=）。作为一个编程约定，只有DEFAULT是这样使用的，以使这个语义上与众不同的构造和声明中较为直截了当的赋值相区别。

可以有意地省略两种缺省参数，以便在第一个调用 get_data 中，标志参数被默认为FALSE，并且排序参数被默认为空间。这样做是一种非常清楚的编程风格，可以在参数表中只定义你感兴趣的参数。但是要注意，不能跳过一个默认参数而直接提供下一个参数，因为定义参数的标记符是有位置的。参数的位置有重要意义，不能尝试使用一个占位符，例如额外的逗号。

5. 位置和命名表示法

可以使用一个称为命名表示法的替换表示法以任意顺序定义参数。与值一起提供参数的名称。

```
ELSIF get_data(key3,
              Psort => 'ASCENDING') THEN -- valid (Pflag defaulted)
```

可以从左到右开始使用位置表示法，然后切换到命名表示法，这被称为混合表示法。一旦使用命名表示法，必须在后面的参数中保持它的名字。命名表示法可以被用于任意参数，而不仅仅是默认的参数。

```
ELSIF get_data(key3, Psort => 'ASCENDING',
              Pflag => TRUE) THEN -- right
...
ELSIF get_data(Pkey => key3, 'ASCENDING',
              Pflag => TRUE) THEN -- wrong!
```

虽然这看起来非常方便而且在编程语言中不同寻常，我可是从来不使用它。但是，如果有一组参数，并且几乎每个参数都是缺省参数，而且希望调用子程序具有最大限度的灵活性，这实际上是不可缺少的。一个例子是使用 Oracle 的 Web 应用服务器开发者工具箱，其中的过程具有许多参数，大部分在多数时间都不会用到。

6. 内建函数

几乎全部在 SQL 中使用的内建函数和运算符都可以用于 PL/SQL 的表达式和过程化语句中，只有极个别例外不符合这个规则。

在一个表达式中不能使用 ‘=ANY (…)’，而要以下列形式使用 IN 运算符：

```
IF (key IN ('A', 'B', 'C')) THEN -- acts like an OR conditional
```

还可以使用其他运算符，例如 BETWEEN、IS NULL、IS NOT NULL、LIKE 等等。

不能在过程化语句中使用 DECODE，而且，在过程化语句中不允许使用任何 SQL 组函数，它们在这种环境下没有使用的意义。当然，对于 PL/SQL 中嵌套的 SQL 语句，没有更多的限制。

已经看到如何使用 SQLCODE 捕获数字形式的例外错误值。SQLERRM (sqlcode) 可以将与例外相关的 SQLCODE 值转换为错误消息字符串。所有其他的内建函数都差不多。

28.6 Oracle 8i 专有的功能

Oracle 的每个版本都致力于提高系统的性能和可用性，Oracle 8i 也不例外。除了其他方面的改进提高（例如 PL/SQL 表更易于使用）外，Oracle 还使 PL/SQL 的一些功能更加强大的特性易于使用或清楚快捷。

28.6.1 本地动态 SQL

有些时候不知道要查找些什么信息，直到实际操作时才知道需要的东西。例如，当工作时，也许不知道需要查看哪些表或需要使用什么查找条件。可以这么说，开发者已经拥有能力生成一个忙碌的查询，但是它是一个神秘的进程，要用到 RDBMS（关系型数据库管理系统）提供的 DBMS_SQL 包。但是使用 Oracle 8i，处理过程被大大地简化了，如下所示：

```
PROCEDURE INSERT_ITEMS (Ptable varchar2, Pprod varchar2, Pdesc varchar2) is
  sql_statement varchar2(500);
BEGIN
  sql_statement := 'insert into '||p_table||' values (:prod_id, :desc)';
  EXECUTE IMMEDIATE sql_statement USING Pprod, Pdesc;
END;
```

如果使用 DBMS_SQL 完成同样的任务，则需要几十行的程序代码！本地动态 SQL 将给你一个显著的性能改善。

28.6.2 成批捆绑

使用 PL/SQL 的一个好处是可以将发送到服务器的一系列语句包含到一个组中这一事实。虽然如此，当服务器在 PL/SQL 引擎与 SQL 引擎之间移动时，嵌套在 PL/SQL 代码中的 SQL 语句还是需要环境的切换。为了利用这个新特性，语句：

```
FOR i in 1..10000
LOOP
  INSERT INTO orders VALUES (hold_orders(i).cust_id, hold_orders(i).prod_id);
END LOOP;
```

可以被写作：

```
FORALL I in 1..10000
  INSERT INTO orders VALUES (hold_orders(i).cust_id, hold_orders(i).prod_id);
```

第29章 使用存储子过程、包与提供的包

本章要点：

- 定义存储子过程与包
- 建立与使用存储程序
- 使用SHOW ERRORS调试
- 检查存储程序或包的状态
- 建立与使用包
- 关于Oracle 8i数据库提供的包
- 描述提供的包
- 开始学习Oracle 8i提供的包
- 使用Oracle 8i提供的包

29.1 定义存储子过程与包

当你着手创建存储子程序和包的时候，真正的 Oracle8i应用开发开始了，存储子程序和包是经过编译并存储在数据库中的永久程序代码模块。它们是你设计的可共享、可重入并且可重用的软件对象。可以使用支持远程过程调用的语言从其他的 PL/SQL模块、SQL语句以及客户端应用中调用它们。

当你编译了一个存储子程序或包时，程序的源代码、编译代码、编译状态以及全部编译错误存储到数据字典中。各种不同的数据字典视图可以帮助你查看这些项。使用 USER_视图，能够得到关于所编译模块的信息；使用 ALL_视图，能够得到其他人编译并授权你访问的相关模块限制信息；假如你拥有数据库系统管理员权限，使用 DBA_视图，可以得到任何人编译的全部信息。这些视图在表 29-1中列出（为了简便，作为 DBA_视图列出）。

表29-1 存储子程序和包的数据字典视图

视图名称	说明
DBA_SOURCE	所有编译模块的文本源代码
DBA_ERRORS	全部模块编译错误的文本列表清单
DBA_OBJECT_SIZE	编译模块的状态，例如有效性、源代码以及对象大小
DBA_OBJECTS	编译模块种类（存储过程、函数、包、包程序体）
DBA_DEPENDENCIES	对象相关性列表，例如在包中引用的表

不存在暴露对象代码的视图，因为根本不需要看到它。你要知道的一切就是它是不是在这里以及它是否是合法的。当一个相关对象被修改或删除时，例如向一个表添加一列或删除一个表时，一个已编译模块变为非法。如果一个存储模块变为非法，必须由服务器自动地重新编译它或由它的拥有者手工编译它。

29.2 建立与使用存储程序

创建存储子程序的语法与在匿名 PL/SQL 块中定义子程序的语法非常类似。除含一些额外增加的特性外，存储子程序拥有子程序的全部相同特性，这些特性你已经在前面学习编写过。让我们使用你先前看到的 `bool_to_char` 函数编写一个存储函数（参见清单 29-1）。

清单29-1 BOOL2CHR.SQL —— 编写重用代码存储子程序

```
CREATE OR REPLACE FUNCTION bool_to_char(Pbool IN BOOLEAN)
RETURN VARCHAR2 IS
    str VARCHAR2(5); -- capture string to return
BEGIN
    IF (Pbool) THEN -- test Boolean value for TRUE
        str := 'TRUE';
    ELSIF (NOT Pbool) THEN -- FALSE
        str := 'FALSE';
    ELSE -- must be NULL
        str := 'NULL';
    END IF; -- test Boolean value
    RETURN (str);
END bool_to_char;
/
```

服务器回答：

Function created.

这就是你所得到的。服务器并不执行程序，它仅编译程序，以便日后当你从其他的 PL/SQL 块中调用它时，你可以执行它。

提示 `CREATE OR REPLACE` 语法创建一个新函数或替换一个已有函数。这意味着不需要给存储代码增加重编译或改变的源代码，而是使用新版本完全代替它们。

警告 当使用 `CREATE OR REPLACE` 时，需要良好的源代码管理手段。当替换子程序时，老代码将永远从数据字典中消失。它还意味着在模式中只能含有一个拥有这个名字的对象。

现在，使用一个未命名 PL/SQL 块运行这个新创建的存储函数，如程序清单 29-2 中所示。

清单29-2 TESTBOOL.SQL —— 测试执行存储子程序

```
SET SERVEROUTPUT ON
BEGIN
    DBMS_OUTPUT.enable;
    DBMS_OUTPUT.put_line(bool_to_char(TRUE));
    DBMS_OUTPUT.put_line(bool_to_char(FALSE));
    DBMS_OUTPUT.put_line(bool_to_char(NULL));
END;
/
```

这个例子差不多测试了存储函数可能返回值的全部可能性。它被称为一个单元测试。对于每项输入，验证它的输出结果。输入值应该测试所有的临界状态（输入定义限制内或接近输入限制的值，包括限制之间的一些随机值）。应该给你的存储子程序准备单元测试文件，就像这个文件，那么你就可以检验并验证你的代码是否工作正确。将单元测试程序与存储子程序一起保留，以便当你修改子程序的时候，可以再次测试它。

在运行程序清单29-2后，服务器送回下列输出结果信息：

```
TRUE
FALSE
NULL
```

PL/SQL procedure successfully completed.

如果尝试传入一个不是布尔值的其他值，将会发生什么？试试看：

```
BEGIN
    DBMS_OUTPUT.put_line(bool_to_char(0));
END;
/
```

服务器响应下列信息：

```
ERROR at line 1:
ORA-06550: line 2, column 24:
PLS-00306: wrong number or types of arguments in call to 'BOOL_TO_CHAR'
ORA-06550: line 2, column 3:
PL/SQL: Statement ignored
```

错误消息表明这是一个编译错误。PL/SQL引擎的强类型检查捕获这个错误并返回详尽的错误消息。

看一下另外一个略微长一些的存储过程例子，如程序清单 29-3所示：

清单29-3 SHOWINDX.SQL —— 显示表的索引信息的存储过程

```
CREATE OR REPLACE PROCEDURE show_index(Ptable IN all_indexes.table_name%TYPE
DEFAULT NULL) IS
    -- local cursors
    CURSOR show_index_cur(Ctable all_indexes.table_name%TYPE) IS
        SELECT
            table_owner, table_name, tablespace_name, index_name, uniqueness, status
        FROM all_indexes
        WHERE
            (Ctable IS NULL OR table_name = Ctable) -- one table or all
        ORDER BY
            table_owner, table_name, index_name;
    -- local constants
    TB CONSTANT VARCHAR2(1) := CHR(9); -- tab character
    -- local record variables
    show_index_rec show_index_cur%ROWTYPE; -- based on cursor
    old_index_info show_index_cur%ROWTYPE; -- used to detect control break
    -- local variables
    status NUMERIC;
    local_table all_indexes.table_name%TYPE;
BEGIN
    status := 0;
    local_table := UPPER(Ptable); -- make upper case
    old_index_info.table_owner := 'GARBAGE_OWNER'; -- initialize
    old_index_info.table_name := 'GARBAGE_TABLE';
    IF (local_table IS NULL) THEN -- one table or all?
        DBMS_OUTPUT.put_line('User ' || USER || ': Index Information for All
Tables');
    ELSE
        DBMS_OUTPUT.put_line('User ' || USER || ': Index Information for Table ' ||
local_table);
    END IF; -- one table or all?
    OPEN show_index_cur(local_table);
    LOOP -- get index information
        FETCH show_index_cur INTO show_index_rec;
        EXIT WHEN show_index_rec%NOTFOUND;
        IF (old_index_info.table_owner != show_index_rec.table_owner OR
            old_index_info.table_name != show_index_rec.table_name) THEN -- control
```

```

break
    DBMS_OUTPUT.put_line(TB); -- double spacing between tables
END IF;
DBMS_OUTPUT.put_line('Table Owner: ' || show_index_rec.table_owner || TB ||
    'Table: ' || show_index_rec.table_name);
DBMS_OUTPUT.put_line('Index: ' || show_index_rec.index_name || TB || ' in '
||
    show_index_rec.tablespace_name || TB ||
    show_index_rec.uniqueness || TB ||
show_index_rec.status);
old_index_info := show_index_rec; -- copy new values to old
END LOOP; -- get index information
CLOSE show_index_cur;
EXCEPTION
WHEN OTHERS THEN
BEGIN
    status := SQLCODE;
    DBMS_OUTPUT.put_line('show_index: ' || SQLERRM(status)); -- display error
message
    IF (show_index_cur%ISOPEN) THEN -- close any open cursors
        CLOSE show_index_cur;
    END IF;
EXCEPTION
WHEN OTHERS THEN
    NULL; -- don't care
END;
END show_index;
/

```

这一次，服务器返回下面信息：

Procedure created.

要执行这个过程，可以从一个匿名 PL/SQL 块中调用它，或者简单地使用 EXECUTE 命令单行调用它：

```
EXECUTE show_index('DEPT');
```

提示 EXECUTE 命令只能用于运行单行语句。如果语句包含两个或多个语句行，必须使用匿名 PL/SQL 块（使用 BEGIN...END）。如果将两条或多条语句塞入一行，仍旧可以使用 EXECUTE 命令。实际上，EXECUTE 命令将一行扩展为一个 PL/SQL 块，它仅仅是简写形式。

下面是作者的测试系统生成的输出结果信息：

```

User SYSTEM: Index Information for Table DEPT

Table Owner: SYSTEM      Table: DEPT
Index: DEPT_PRIMARY_KEY in USER_DATA  UNIQUE  VALID

PL/SQL procedure successfully completed.

```

第一次运行这个存储过程时，当等待服务器将过程加载到内存中的时候，你或许注意到了个略微的停顿。在继续进行的调用中，因为过程已经被加载到数据库的高速缓存，运行它的速度明显加快了。

这个特定的例子展示了一些良好的特性，它们是在你自己的存储过程中所需要的。你应该与局部变量声明的结构保持一致，并且以相同的顺序放置它们。还有，注意例外处理程序中的块，一定要关闭游标，以免一个错误把它设为打开状态。如果没有这样做而且遇到一个例外，下一次当运行这个存储过程的时候，将由于 CURSOR_ALREADY_OPEN 错误立即产生

故障。因为游标在会话内保持打开状态直到它被关闭或会话结束。为了清除打开的游标，你的用户将不得不重新连接数据库。

如果用户需要显示所有的表和它们的索引，只需要简单地删除单独输入参数（和括号），就像下面所示：

```
EXECUTE show_index;
```

必须将自己放到用户的角度来设想用户希望采用哪种方式使用这个工具。更好的方法是询问用户。

29.2.1 从SQL调用存储过程

假设不满意Oracle8i提供的内建函数TO_NUMBER（）。你的抱怨可能是一个字符到数字的转换失败了，原因是字符串不能表示一个合法的数字，SQL失败并突然终止。你所希望的是，至少错误能够被准确地处理，那么就可以继续处理剩余的数据集了。尝试使用存储函数解决这个问题，如程序清单29-4中所示：

清单29-4 CHAR2NUM.SQL —— 字符到数字转换

```
CREATE OR REPLACE FUNCTION char_to_number(Pstr IN VARCHAR2, Pformat IN VARCHAR2
DEFAULT NULL)
RETURN NUMBER IS
BEGIN
    IF Pformat IS NULL THEN -- optional format not supplied
        RETURN (TO_NUMBER(Pstr));
    ELSE
        RETURN (TO_NUMBER(Pstr, Pformat)); -- format supplied
    END IF; -- test for optional format
EXCEPTION
WHEN OTHERS THEN -- unknown value
    RETURN (NULL);
END char_to_number;
/
```

可以采用两种方式运行这个存储函数：

- 从一个PL/SQL块。
- 从一条SQL语句。

首先，尝试从PL/SQL块中运行它，如程序清单29-5所示。

清单29-5 TESTC2N1.SQL —— 从PL/SQL块测试char_to_number（）函数

```
DECLARE
    v VARCHAR2(1) := 0;
    w VARCHAR2(10) := '999.999'; -- try a floating point number
    x VARCHAR2(11) := '+4294967295'; -- try a big positive number
    y CHAR(11) := '-4294967296'; -- try a big negative number
    z VARCHAR2(10) := 'garbage'; -- this is NOT a number!
BEGIN
    -- stored function returns NULL on error, so convert NULL to error message
    DBMS_OUTPUT.put_line(v || ' is ' || NVL(TO_CHAR(char_to_number(v)), 'NOT A
NUMBER!'));
    DBMS_OUTPUT.put_line(w || ' is ' || NVL(TO_CHAR(char_to_number(w)), 'NOT A
NUMBER!'));
    DBMS_OUTPUT.put_line(x || ' is ' || NVL(TO_CHAR(char_to_number(x)), 'NOT A
NUMBER!'));
    DBMS_OUTPUT.put_line(y || ' is ' || NVL(TO_CHAR(char_to_number(y)), 'NOT A
```

```
NUMBER!'));
  DBMS_OUTPUT.put_line(z || ' is ' || NVL(TO_CHAR(char_to_number(z)), 'NOT A
NUMBER!'));
END;
/
```

服务器返回下列信息：

```
0 is 0
999.999 is 999.999
+4294967295 is 4294967295
-4294967296 is -4294967296
garbage is NOT A NUMBER!

PL/SQL procedure successfully completed.
```

现在让我们在一条SQL语句中测试它，如程序清单 29-6所示。

清单29-6 TESTC2N2.SQL —— 从SQL运行char_to_number () 函数

```
SELECT '0' str,
       NVL(TO_CHAR(char_to_number('0')), ' IS NOT A NUMBER!') num FROM DUAL;
SELECT '999.999' str,
       NVL(TO_CHAR(char_to_number('999.999')), ' IS NOT A NUMBER!') num FROM DUAL;
SELECT '+4294967295' str,
       NVL(TO_CHAR(char_to_number('+4294967295')), ' IS NOT A NUMBER!') num FROM DUAL;
SELECT '-4294967296' str,
       NVL(TO_CHAR(char_to_number('-4294967296')), ' IS NOT A NUMBER!') num FROM DUAL;
SELECT 'garbage' str,
       NVL(TO_CHAR(char_to_number('garbage')), ' IS NOT A NUMBER!') num FROM DUAL;
```

得到与先前相同的结果。当你刚刚完成从一个字符串到数字的转换，就将它转换回字符串，看起来有些愚蠢，但是这样验证了存储函数的操作，尤其在上一个查询中。

29.2.2 从PL/SQL调用存储子过程

已经见到过如何从一个 PL/SQL块中激活一个存储子程序。它们有什么更高的使用价值呢？已经尽力构造一个有 14个表连结而成的查询了吗？我们可以使用其他手段取而代之，为每个主表创建一组单行查找程序，或许在连结中用到过主表的键值。然后，在一个 PL/SQL块中，只要写入用于驱动一个游标循环的最少量的表。在循环内部，执行单行查找以找出附带的数据值。使用这项技术，可以看到速度提高了 100%或更多。先前需要执行 20分钟的查询，使用这种技术可以在 2分钟内或更短的时间内完成。如果使用了某种类型的报表编写器来格式化输出结果，可以将数据写入临时表，然后临时表可以使用报表工具快速扫描。显然，这样做在某种程度上减慢了程序执行速度，但是它还是比使用大量、复杂的查询让 Oracle8i服务器陷入停顿要快得多。另外，它还易于验证输出结果的正确性。如果严格地测试单行查找程序，必须验证的唯一事实是足够小的查询。

现在考虑一下有多少次不得不编写一个复杂查询的外部连结代码。在 PL/SQL块中，测试单行查找没有发现数据并且跳转到循环的结束或继续循环是非常容易的事情。例如，一个申请处理程序也许含有一个循环，它看起来有点像这样：

```
LOOP -- process all claims for the period selected
  FETCH claims_cur INTO claims_rec;
  EXIT WHEN claims_cur%NOTFOUND;
```

```
-- get related table info
get_claimant_info(claims_rec.claimant_ssn, status);
get_provider_info(claims_rec.provider_id, status);
get_approval_info(claims_rec.approvedby, status);
IF (status != 0) THEN -- no approval on file!
    GOTO SKIPIT; -- skip processing claim
END IF;
... -- more single row lookups, claims processing
<<SKIPIT>> -- continue with next claim
END LOOP; -- process all claims for the period selected
```

本例中，单行查找的SQLCODE结果值返回变量status（状态）。为了让这个程序真正有用，需要将有关为什么申请没有被处理（没有被批准）的信息储存到某种类型的应用错误表中。更好的方法是，在查找程序的内部，可以将查找键值和错误代码存储到你设计的一个错误表中，那样，你可以在日后判定引起一个申请未被处理的子程序和键值。调用程序可以存储额外的数据（例如环境信息），可以更加容易地排除故障。

存储过程的另外一个用途就是实现商务规则。可以从触发子、客户程序或其他的 PL/SQL 程序中调用这些规则，如下例所示：

```
CREATE OR REPLACE TRIGGER check_approval_status
BEFORE INSERT ON claim_disbursal
DECLARE
    status NUMERIC;
BEGIN -- check for approval
    get_approval_info(:new.approvedby, status);
    IF (status != 0) THEN -- no approval on file!
        RAISE_APPLICATION_ERROR(-20100, 'No approval on file!');
    END IF;
END; -- check for approval
END check_approval_status;
```

这里，应用定义例外引起一个 INSERT 操作被回滚。一个存储过程定义商务规则，可以在许多地方使用该商务规则。如果需要修改规则，可以在存储过程的程序体中更改它。与它相关的 PL/SQL 代码仅仅需要被重新编译即可。

29.3 使用 SHOW ERRORS 调试

到目前为止，存储过程都是没有任何问题地编译通过。不幸的是，可以预料到会犯错误从而导致编译错误。幸好，调试错误的信息唾手可得。但是，Oracle8i 提供的用于观察它们的功能有一些使用限制，原因是它相对比较简单，但是我们有更好的解决方案。

当未命名 PL/SQL 块不能成功编译时，服务器将错误信息直接转储回 SQL*Plus。对于存储子过程，这个过程略有区别。在编译失败后，立即输入下列命令：

```
SHOW ERRORS
```

每个错误以及错误出现的行号都详尽地显示出来。

提示 SHOW ERRORS 命令仅显示最后一个存储子程序或包提交的编译错误。如果在一行中提交了两个子程序，SHOW ERRORS 命令只显示第二个子程序的错误信息（如果有的话）。但是，在 USER_ERRORS 中仍然可以同时获得两个子程序的错误信息。

请考虑程序清单 29-7 中的源代码，在过程之中隐藏了一个缺陷：

清单29-7 SHOWERR1.SQL —— 具有缺陷的错误格式化过程

```

CREATE OR REPLACE PROCEDURE showerr1(
  Pname  IN user_errors.name%TYPE,
  Ptype  IN user_errors.type%TYPE) IS
  CURSOR get_errors(Cname IN user_errors.name%TYPE,
                    CType  IN user_errors.type%TYPE) IS
    SELECT * FROM USER_ERRORS
    WHERE name = Cname AND type = CType
    ORDER BY SEQUENCE;
  get_errors_rec get_errors%TYPE;
  status NUMERIC := 0;
  Lname  user_errors.name%TYPE;
  Ltype  user_errors.type%TYPE;
BEGIN
  Lname := UPPER(Pname);
  Ltype := UPPER(Ptype);
  DBMS_OUTPUT.put_line('Compilation errors for ' || Lname);
  OPEN get_errors(Lname, Ltype);
  LOOP -- display all errors for this object
    FETCH get_errors INTO get_errors_rec;
    EXIT WHEN get_errors%NOTFOUND;
    DBMS_OUTPUT.put_line('At Line/Col: ' || get_errors_rec.line ||
                        ' ' || get_errors_rec.position);
    DBMS_OUTPUT.put_line(get_errors_rec.text);
  END LOOP; -- display all errors
  CLOSE get_errors;
  DBMS_OUTPUT.put_line('Errors Found: ' || TO_CHAR(status));
EXCEPTION
  WHEN OTHERS THEN
    BEGIN
      status := SQLCODE;
      IF (get_errors%ISOPEN) THEN -- cursor still open
        CLOSE get_errors;
      END IF;
    END;
END showerr1;
/

```

服务器返回下列错误消息：

Warning: Procedure created with compilation errors.

要得到更多信息，需要输入下列命令：

SHOW ERRORS

SHOW ERRORS 命令的输出结果如下所示：

Errors for PROCEDURE SHOWERR1:

LINE/COL ERROR

```

-----
9/18      PLS-00206: %TYPE must be applied to a variable or column, not
          'GET_ERRORS'

9/18      PL/SQL: Item ignored
19/5      PL/SQL: SQL Statement ignored
19/27     PLS-00320: the declaration of the type of this expression is
          incomplete or malformed

21/5      PL/SQL: Statement ignored
21/45     PLS-00320: the declaration of the type of this expression is
          incomplete or malformed

23/5      PL/SQL: Statement ignored

```

```

23/26      PLS-00320: the declaration of the type of this expression is
           incomplete or malformed

26/3       PL/SQL: Statement ignored

26/52      PLS-00201: identifier 'ERRS' must be declared

```

注意 PL/SQL引擎清除存储源代码的空白行。如果为了程序的可读性，在程序中留有两倍行距，会发现存储在USER_SOURCE中的源代码变得很快与源代码不一样了，这使在最初源代码文件上进行的调试更为困难。

某些错误消息相当清楚，另外一些却非常含糊。还提供了错误出现的行号和列号。不幸的是，出现问题的源代码行没有显示出来。可以使用下面的 SQL 语句显示源代码：

```

SELECT line, text
FROM user_source
WHERE name='SHOWERR1' AND line IN (9, 19, 21, 23, 26)
ORDER BY line;

```

源代码显示如下：

```

LINE TEXT
-----
 9  get_errors_rec get_errors%TYPE;
19  FETCH get_errors INTO get_errors_rec;
21  DBMS_OUTPUT.put_line('At Line/Col: ' || get_errors_rec.line ||
23  DBMS_OUTPUT.put_line(get_errors_rec.text);
26  DBMS_OUTPUT.put_line('Errors Found: ' || TO_CHAR(errs));

```

配合上面的错误消息，将看到下列信息：

在第9行上，必须基于get_errors%ROWTYPE定义记录变量，而不是%TYPE。

第19行错误的原因是get_errors_rec没有正确地定义。

第21行错误的原因是get_errors_rec没有正确地定义。

第23行错误的原因是get_errors_rec没有正确地定义。

在第26行上，你忘记了声明变量errs。

可以看到一个错误是怎样扩展到整个代码的。在声明中修改错误后，相关的错误全部消失了。

还有，要注意在这次编译中没有暴露的一些语法错误。例如，列命令 type 应该大写并且应该用双引号括起来，因为 TYPE 实际上是一个保留字。在上面的五个错误清除后，这个错误出现了（试一试）。

修改并提高的存储过程在程序清单 29-8 中可以找到。

清单29-8 SHOWERR.SQL —— 最终版本，PL/SQL 错误格式化过程

```

CREATE OR REPLACE PROCEDURE showerr(
  Pname IN user_errors.name%TYPE,
  Ptype IN user_errors."TYPE"%TYPE DEFAULT NULL) IS
  CURSOR get_errors(Cname IN user_errors.name%TYPE,
                    Ctype IN user_errors."TYPE"%TYPE) IS
    SELECT
      E.sequence, E.line, E.position, E.text err_text, S.text src_text
    FROM USER_ERRORS E, USER_SOURCE S
    WHERE
      E.name = Cname AND E."TYPE" = Ctype AND
      S.name = E.name AND S."TYPE" = E."TYPE" AND S.line = E.line
    ORDER BY E.sequence;

```

```

get_errors_rec get_errors%ROWTYPE;
status NUMERIC := 0;
Lname user_errors.name%TYPE;
Ltype user_errors.type%TYPE;
errs NUMERIC := 0; -- number of errors
cols NUMERIC; -- extra column padding needed for error position
wspc VARCHAR2(2) := CHR(20) || CHR(10); -- trailing whitespace
dspc VARCHAR2(1) := CHR(9); -- double spacing
BEGIN
  Lname := UPPER(Pname);
  IF (Ptype IS NOT NULL) THEN -- user supplied type
    Ltype := UPPER(Ptype);
  ELSE -- look for 1 object and get the type
    BEGIN
      SELECT object_type INTO Ltype
      FROM user_objects
      WHERE object_name = Lname;
    EXCEPTION
      WHEN OTHERS THEN
        RAISE_APPLICATION_ERROR(-20100, Lname ||
          ' type ambiguously defined');
    END;
  END IF; -- user supplied type
  DBMS_OUTPUT.put_line('Compilation errors for ' ||
    Ltype || ' ' || Lname);
  OPEN get_errors(Lname, Ltype);
  LOOP -- display all errors for this object
    FETCH get_errors INTO get_errors_rec;
    EXIT WHEN get_errors%NOTFOUND;
    IF (SUBSTR(get_errors_rec.err_text, 1, 4) = 'PL/S') THEN
      GOTO SKIPIT; -- ignore the 'PL/SQL: Statement ignored' messages
    END IF;
    cols := LENGTH(TO_CHAR(get_errors_rec.line)) + 3;
    DBMS_OUTPUT.put_line([' ' || TO_CHAR(get_errors_rec.line) ||
      ' ' || RTRIM(get_errors_rec.src_text, wspc)];
    DBMS_OUTPUT.put_line(LPAD('^', get_errors_rec.position + cols, '-'));
    DBMS_OUTPUT.put_line([' ' || TO_CHAR(get_errors_rec.line) ||
      ' ' || get_errors_rec.err_text]);
    DBMS_OUTPUT.put_line(dspc); -- double space
    errs := errs + 1;
    <<SKIPIT>>
    NULL;
  END LOOP; -- display all errors
  CLOSE get_errors;
  DBMS_OUTPUT.put_line('Errors Found: ' || TO_CHAR(errs));
EXCEPTION
  WHEN OTHERS THEN
    BEGIN
      status := SQLCODE;
      IF (get_errors%ISOPEN) THEN -- cursor still open
        CLOSE get_errors;
      END IF;
      DBMS_OUTPUT.put_line('showerrs: ' || SQLERRM(status));
    EXCEPTION
      WHEN OTHERS THEN
        NULL; -- don't care
    END;
END showerr;
/

```

通过执行新的过程，得到的结果更为清晰、更为详细并且更易于阅读：

```

execute showerr('showerr1');
Compilation errors for PROCEDURE SHOWERR1
-----

```

```
[9]  get_errors_rec get_errors%TYPE;
-----^
[9] PLS-00206: %TYPE must be applied to a variable or column,
not 'GET_ERRORS'

[19]  FETCH get_errors INTO get_errors_rec;
-----^
[19] PLS-00320: the declaration of the type of this expression is
incomplete or malformed

[21]  DBMS_OUTPUT.put_line('At Line/Col: ' || get_errors_rec.line ||
-----^
[21] PLS-00320: the declaration of the type of this expression is
incomplete or malformed

[23]  DBMS_OUTPUT.put_line(get_errors_rec.text);
-----^
[23] PLS-00320: the declaration of the type of this expression is
incomplete or malformed

[26]  DBMS_OUTPUT.put_line('Errors Found: ' || TO_CHAR(errs));
-----^
[26] PLS-00201: identifier 'ERRS' must be declared

Errors Found: 5

PL/SQL procedure successfully completed.
```

你会更喜欢这样。或许你想控制出现问题行前后的源代码打印行数，以便能够得到相关的上下文。我把这个问题作为一个练习留给你来解决。

29.4 检查存储程序或包的状态

执行下面的SQL语句查看你的全部存储子程序的状态：

```
COLUMN object_name FORMAT A30
COLUMN timestamp FORMAT A20
SELECT object_name, object_type, timestamp, status
FROM user_objects
WHERE
    object_type IN ('FUNCTION', 'PROCEDURE', 'PACKAGE', 'PACKAGE BODY')
ORDER BY object_name, object_type;
```

这条命令的输出结果，在我的系统上如下所示：

OBJECT_NAME	OBJECT_TYPE	TIMESTAMP	STATUS
CHAR_TO_BOOL	FUNCTION	1997-11-01:16:06:43	VALID
CHAR_TO_NUMBER	FUNCTION	1997-11-01:13:34:26	VALID
SHOWERR	PROCEDURE	1997-11-05:06:01:31	VALID
SHOWERRS	PROCEDURE	1997-11-05:05:47:10	VALID
SHOWERR1	PROCEDURE	1997-11-04:19:51:31	INVALID
SHOW_INDEX	PROCEDURE	1997-11-01:13:17:37	VALID

6 rows selected.

注意过程SHOWERR1存在，即使它不能成功地编译。它确实在那里，源代码和它的所有内容。但是如果试图运行它的话，会得到一个错误。

提示 表结构的更改有时候会使存储过程和对象无效。可以使用ALTER...COMPILE命令重新编译指定的数据库对象，如下面的例子所示：

```
ALTER PROCEDURE SHOWERRS COMPILE;
```

另外一个要查看的有趣事情是代码统计。要查看这些信息，运行下面的语句：

```
SELECT name, "TYPE", source_size, parsed_size, code_size, error_size
FROM user_object_size
WHERE
  "TYPE" IN ('FUNCTION', 'PROCEDURE', 'PACKAGE', 'PACKAGE BODY')
ORDER BY name, "TYPE";
```

在我的系统上，这条语句的输出结果看起来如下所示：

NAME	TYPE	SOURCE_SIZE	PARSED_SIZE	CODE_SIZE	ERROR_SIZE
CHAR_TO_BOOL	FUNCTION	391	841	546	0
CHAR_TO_NUMBER	FUNCTION	375	575	384	0
SHOWERR	PROCEDURE	2587	5725	3598	0
SHOWERRS	PROCEDURE	2227	5035	3073	0
SHOWERR1	PROCEDURE	1149	0	0	495
SHOW_INDEX	PROCEDURE	2408	4221	2629	0

6 rows selected.

注意，因为没有能够成功编译，SHOWERR1没有语法分析或代码大小。有趣的是，代码和语法分析的大小大于源代码。在源代码大小与语法分析及代码大小之间的确没有太多的关联。代码分析的大小取决于游标、隐式 SQL、子程序等对象的数目——你可能会获得更多存储的事物。代码大小使你具有了每个对象在 SGA 中占用多少内存的概念。

提示 使用 DESCRIBE 命令，能够得到特定存储子程序参数列表的方便快捷的视图。

```
DESC SHOWERR
PROCEDURE SHOWERR
Argument Name          Type                      In/Out Default?
-----
PNAME                  VARCHAR2(30)             IN
PTYPE                  VARCHAR2(12)             IN          DEFAULT
```

29.5 建立与使用包

包是程序对象的集合，可以提供数据、游标和其他语言结构的连续性。当引用包里的任意对象时，整个包便被加载到内存中。虽然这看起来像是一个大的磁盘需求，但要记住当包被加载后，代码对所有的用户可用。只要有任意一个用户引用这个包，它就保留在 SGA 区中，当前在 SGA 区里面的全部程序代码都是可用的。另外一种做法是像加载存储子程序那样分别加载每段程序代码。因为被加载后，代码是可以重入的，所以对于全体用户都可用。代码重入弥补了第一次磁盘命中所造成的系统开销。从服务器视图的观点来说，这样非常有效，既节省了内存的使用又减少了磁盘存取。

包还允许子程序重载，这个特性对于纯粹的存储子程序不可用。使用重载，同一个子程序名称可以在不同组合及数目的参数类型中被重复使用。如果观察 STANDARD.SQL，可以看到 TO_CHAR、TO_DATE 和 TO_NUMBER 的几个声明，每种声明都带有不同数据类型的参数。

29.5.1 包声明与包体

PL/SQL 包由两个部分组成：

■ 包声明（也被称为头）。

■ 包体。

包体包含包的声明部分中提到的全部子程序的实际代码。

包声明含有希望展示给其他人的全部语言结构，它们可以是变量、用户定义数据类型、游标以及子程序，放置在声明中的任何对象都是可全局访问的，这里没有可执行代码，只有子程序的声明。程序清单 29-9 展示了一个包头的例子，其中使用了几个你已经见过的函数。

清单 29-9 LIBHDR.SQL —— 一个提议的库包的声明

```
CREATE OR REPLACE PACKAGE lib IS
-- public global user defined types
-- public global constants
MAXINT CONSTANT BINARY_INTEGER := +2147483647; -- +(2^31 - 1)
MININT CONSTANT BINARY_INTEGER := -2147483647; -- -(2^31 - 1)
MAXDATE CONSTANT DATE := TO_DATE('31-DEC-4712 AD', 'DD-MON-YYYY AD');
MINDATE CONSTANT DATE := TO_DATE('01-JAN-4712 BC', 'DD-MON-YYYY BC');
-- public global cursors
-- public global variables
-- public global subprograms
-- convert a BOOLEAN to a STRING ('TRUE', 'FALSE', 'NULL')
FUNCTION bool_to_char(Pbool IN BOOLEAN)
RETURN VARCHAR2;
-- convert a STRING to BOOLEAN (TRUE, FALSE, NULL)
FUNCTION char_to_bool(Pstr IN VARCHAR2) RETURN BOOLEAN;
-- safe STRING to NUMBER conversion; returns NUMBER or NULL
FUNCTION char_to_number(Pstr IN VARCHAR2, Pformat IN VARCHAR2 DEFAULT NULL)
RETURN NUMBER;
-- debug flag interface routines
PROCEDURE debug_on;      -- turns debug on
PROCEDURE debug_off;     -- turns debug off
PROCEDURE debug_toggle;  -- toggles debug on/off
FUNCTION debug_status    -- is debug on/off?
RETURN BOOLEAN;
END lib;
/
```

注意在其他的存储子程序对象中见到的标准的 CREATE OR REPLACE 语法。现在，不必像创建单个子程序那样显式地创建子程序，而使用以下方法代替，当子程序封装在块的内部时，就像原先那样声明它们，除了在包头中。只要定义一个声明，包括子程序的名称、参数（如果有的话）以及返回类型（如果有的话）。还有，注意子程序是如何遵循全部其他声明的。

那些熟悉 C 编程语言的人将认出这种函数原形。既然声明存在，不必开发这些程序的实际代码，只需要修改它们——不违背定义。只要声明可用，开发者可以立即着手应用的开发，即使程序体还没有编写。由于子程序声明和实现是分开的，如果仅仅代码升级了，使用这些例程的程序就不需要重新编译。只有当声明部分改变的时候，相关模块才需要重新编译。

注意 包编程的标准惯例是将包头与包体分别存放在不同的文件中。使用这种方式，如果仅仅是代码发生变化（最可能的情况是当包还比较新但仍然处于不断变化的状态时），仅需要重新编译包体。

使用所希望的任意标准化命名约定。有些人喜欢给包头和包体使用不同的扩展名 (*.hdr 和 *.pkg)，还有些人喜欢总是给包头和包体使用 *.sql 扩展名，但还是要指出包的内容 (*.hdr.sql、*.pkg.sql)。

包体含有声明中显示的子程序的实际代码，以及私有定义子程序、变量、用户定义数据

类型以及游标。这些对象隐藏在最下面。包体还可以包含初始化代码，它们只在包被首次引用时执行一次，虽然这样做看起来限制了包的使用，但你可以使用它来激活输出、设置各种不同的应用标志（类似于调试开/关）、以及收集全局用户相关数据。虽然包体与包头是有所区别的对象，但在包头被成功地编译之后，包体才能够进行成功地编译。

这个对象在声明中被称为包体，程序被声明得好像存在于一个匿名块中。还要注意包体的名字必须与包头名字相同。头中声明的每个程序在包体中有相应的实现过程。

另外需要注意的事情是在它们的声明中初始化变量。当包第一次被调用的时候，这样的初始化只进行一次。这是因为这些变量只被分配并初始化一次，然后在整个会话间保持不变。如果我在包的声明中重新设置调试标志而节省了几毫秒，在包的初始化中，就需要更明显的并且更多的维护来做这件事情。

29.5.2 比较公有声明与私有声明

在包声明中声明的任何对象在使用范围上都是全局的。也就是说，在当前的会话内，任何PL/SQL程序可以调用并使用声明中定义的变量、游标和子程序。相反，只在包体中声明的变量、游标以及子程序，虽然对于包体自身来说是全局的，但对于外部世界是隐藏的。在两种情况下，变量和游标都是不变的，它们保存当前值，直到从内存中消失或会话终止。

我对使用全局变量的建议是尽量少用全局变量，或根本不用。还有，如果将他们隐藏在包体中，在控制程序访问它们时，将得到全部变量保持不变的好处。通过创建全局可用的子程序来管理它们，可以处理这个访问。

调试接口程序表明这样一个标准惯例，不要将一个变量本身暴露给外部世界，而是提供操作这个变量的子程序。例如，你不希望开发者直接修改和检查变量 `debug_flag` 的值，如果这个调试标志是公共和全局的，任何程序就能够使用带有潜在危害或者至少不希望的结果更改它。如果将来需要添加一些东西或更改调试功能实现的手段，对开发者隐藏具体的实现细节。这是面向对象编程的常用方法。

除了常量和各种表的记录变量，我很少使用全局变量。我还喜欢将声明以一个标准的顺序放置，如清单29-10中所示。这样一来，我总是能够精确地知道到哪里去查找需要的东西。

清单29-10 LIBPKG.SQL —— 前面声明中的包体

```
CREATE OR REPLACE PACKAGE BODY lib IS
  -- private global user defined types
  -- private global constants
  -- private global cursors
  -- private global variables
  debug_flag BOOLEAN;
  user_name VARCHAR(30); -- application user's login ID
  appl_name VARCHAR2(30); -- application currently running
  modl_name VARCHAR2(30); -- module (within app) currently running
  context_pt VARCHAR2(30); -- check-in point within module
  run_date DATE; -- when an application was launched
  -- private global subprograms
  -- implementation of exposed subprograms
  -- convert a BOOLEAN to a STRING ('TRUE', 'FALSE', 'NULL')
  FUNCTION bool_to_char(Pbool IN BOOLEAN)
  RETURN VARCHAR2 IS
    str VARCHAR2(5); -- capture string to return
  BEGIN
    IF (Pbool) THEN -- test Boolean value for TRUE
```

```

        str := 'TRUE';
    ELSIF (NOT Pbool) THEN -- FALSE
        str := 'FALSE';
    ELSE -- must be NULL
        str := 'NULL';
    END IF; -- test Boolean value
    RETURN (str);
END bool_to_char;
-- convert a STRING to BOOLEAN (TRUE, FALSE, NULL)
FUNCTION char_to_bool(Pstr IN VARCHAR2) RETURN BOOLEAN IS
    Lstr VARCHAR2(32767); -- max string length
    Lbool BOOLEAN := NULL; -- local Boolean value (default)
BEGIN
    Lstr := UPPER(LTRIM(RTRIM(Pstr))); -- remove leading/trailing spaces,
uppercase
    IF (Lstr = 'TRUE') THEN
        Lbool := TRUE;
    ELSIF (Lstr = 'FALSE') THEN
        Lbool := FALSE;
    END IF;
    RETURN(Lbool);
END char_to_bool;
-- safe STRING to NUMBER conversion; returns NUMBER or NULL
FUNCTION char_to_number(Pstr IN VARCHAR2, Pformat IN VARCHAR2 DEFAULT NULL)
RETURN NUMBER IS
BEGIN
    IF Pformat IS NULL THEN -- optional format not supplied
        RETURN (TO_NUMBER(Pstr));
    ELSE
        RETURN (TO_NUMBER(Pstr, Pformat)); -- format supplied
    END IF; -- test for optional format
EXCEPTION
WHEN OTHERS THEN -- unknown value
    RETURN (NULL);
END char_to_number;
-- debug flag interface routines
PROCEDURE debug_on IS -- turns debug on
BEGIN
    debug_flag := TRUE;
END debug_on;
PROCEDURE debug_off IS -- turns debug off
BEGIN
    debug_flag := FALSE;
END debug_off;
PROCEDURE debug_toggle IS -- toggles debug on/off
BEGIN
    debug_flag := XOR(debug_flag, TRUE);
END debug_toggle;
FUNCTION debug_status -- is debug on/off?
RETURN BOOLEAN IS
BEGIN
    RETURN(debug_flag);
END debug_status;
BEGIN -- optional package initialization
    DBMS_OUTPUT.enable(1000000);
    debug_flag := FALSE; -- reset
END lib;
/

```

29.5.3 引用包元素

使用点符号，包外部的PL/SQL代码可以引用包的内容，如清单 29-11中所示。记住，点符

号被用来消除一个对象使用范围上所有含义模糊的地方。为了引用一个包装的对象，还必须提供包的名称，采用以下格式‘包名字.包对象’。如果引用对象的代码在同一个范围内，例如在同一个包内，则不需要使用限定符。

清单29-11 TESTLIB.SQL —— 测试提议的库包的执行情况

```
BEGIN
  DBMS_OUTPUT.put_line('DEBUG status is initially: ' ||
                        LIB.bool_to_char(LIB.debug_status));
  LIB.debug_on;
  DBMS_OUTPUT.put_line('DEBUG status is now: ' ||
                        LIB.bool_to_char(LIB.debug_status));
  LIB.debug_off;
  DBMS_OUTPUT.put_line('DEBUG status is now: ' ||
                        LIB.bool_to_char(LIB.debug_status));
  LIB.debug_toggle;
  DBMS_OUTPUT.put_line('DEBUG status is finally: ' ||
                        LIB.bool_to_char(LIB.debug_status));
END;
/
```

要使其成为一个真正的测试，首先彻底退出你的 SQL*Plus 会话，然后重新进入，这样可以确保会话的确是新的。然后，输入命令 SET SERVER OUTPUT ON，并运行程序清单 29.11 中所示的未命名块。注意该块没有打开输出，这点小事在包的初始化代码中执行。调试标志正常情况下默认值为 NULL，也被初始复位了。服务器返回下列信息：

```
DEBUG status is initially: FALSE
DEBUG status is now: TRUE
DEBUG status is now: FALSE
DEBUG status is finally: TRUE

PL/SQL procedure successfully completed.
```

现在再次运行它。

```
DEBUG status is initially: TRUE
DEBUG status is now: TRUE
DEBUG status is now: FALSE
DEBUG status is finally: TRUE

PL/SQL procedure successfully completed.
```

注意，现在调试标志的初始值为 TRUE，这个值从一个程序传递到下一个程序。如果重新连接，一定会拥有一个新的会话，并且再次执行包的初始化。记住，代码静态地存在内存中，但是每位用户得到一个全新的局部私有变量集。运行第二个会话并验证它的行为。

29.6 Oracle 8i 数据库提供的包

除了给予 Oracle 8i 服务器基本功能的 STANDARD 包，例如基本子类声明和数据类型转换程序外，还有一组专供数据库系统管理员和开发者使用的包。这些包根据它们的名字有所区别，它们使用 DBMS_ 或 UTL_ 开头，表示它们与数据库进行交互或提供常规目的的应用。首先，快速浏览这些包并在稍后更为严密地检查它们。

29.6.1 在服务器内交互

大多数 Oracle 8i 提供的包在服务器环境内部处理数据：数据字典项、用户数据库对象或在

系统全局区（System Global Area）中发现的单独的对象，像共享内存缓冲池等。使用它们，可以管理快照、重新编译程序单元、当数据库项目改变时产生异步警报、运行作业等等。

大部分这样的数据库包与内建在数据库应用中的功能交互，或者通过调用可加载外部模块例如DLL来扩展数据库环境。既不希望也不需要理解或使用这些模块的入口点。我宁愿明确地告诉你不要尝试直接使用它们，一定要通过 PL/SQL编程接口提供的手段来使用它们。

29.6.2 在服务器以外交互

一些包对调用环境给出反馈信息，或者通过其他手段给外部处理进程提供一个接口。例如Oracle8i的管道特性，是专门用于会话之间通信的一种方法，它被严格地限定为只在内存中——没有数据库对象用作中间存储介质。另外，当在 SQL*Plus中运行 PL/SQL程序时，DBMS_OUTPUT包使打印语句能够显示。

29.6.3 从服务器获取更多的信息

有几个包装例程可以令你获得你的数据库的附加调整信息，例如共享缓冲池的使用、段空间信息、运行追踪、获得常规数据库信息等等。在习惯使用它们后，它们都会变成工具箱中的标准部件。

29.7 描述提供的包

表29-2是一些重要的Oracle 8i提供的包的一个方便、快捷参考，它还是关于提供的包的内容的一个简明描述说明。

表29-2 提供的包小结

包 名 称	包头文件	说 明
DBMS_ALERT	dbmsalrt.sql	异步处理数据库事件
DBMS_APPLICATION_INFO	dbmsutil.sql	注册当前运行的应用的名称（用于性能监控）
DBMS_AQADM	dbmsaqad.sql	与高级队列选项一起使用
DBMS_DDL	dbmsutil.sql	重新编译存储过程程序和包，分析数据库对象
DBMS_DEBUG	dbmspb.sql	PL/SQL调试器接口
DBMS_DEFER	dbmsdefr.sql	远程过程调用应用的用户接口
DBMS_DESCRIBE	dbmsdesc.sql	说明存储过程的参数
DBMS_JOB	dbmsjob.sql	按指定的时间或间隔执行用户定义的作业
DBMS_LOCK	dbmslock.sql	管理数据库块
DBMS_OUTPUT	dbmsotpt.sql	将文本行写入内存，供以后提取和显示
DBMS_PIPE	dbmspipe.sql	通过内存“管道”在会话之间发送并接收数据
DBMS_PROFILER	dbmspbp.sql	用于配置PL/SQL脚本以鉴别瓶颈问题
DBMS_REFRESH	dbmssnap.sql	管理能够被同步刷新的快照组
DBMS_SESSION	dbmsutil.sql	程序地执行 Alter Session（改变会话）语句
DBMS_SHARED_POOL	dbmspool.sql	查看并管理共享池内容
DBMS_SNAPSHOT	dbmssnap.sql	刷新、管理快照，并清除快照日志
DBMS_SPACE	dbmsutil.sql	获取段空间信息
DBMS_SQL	dbmssql.sql	执行动态SQL和PL/SQL
DBMS_SYSTEM	dbmsutil.sql	开/关给定会话的SQL追踪
DBMS_TRANSACTION	dbmsutil.sql	管理SQL事务

(续)

包 名 称	包头文件	说 明
DBMS_UTILITY	dbmsutil.sql	多种实用工具：对于一个给定的模式，重新编译存储子程序和包、分析数据库对象、格式化错误信息并调用堆栈用于显示、显示实例是否以并行服务器模式运行、以10毫秒间隔获取当前时间、决定数据库对象的全名、将一个PL/SQL表转换为一个使用逗号分割的字符串或卷、获取数据库版本/操作系统字符串
UTL_RAW	utlraw.sql	RAW数据类型的字符串函数
UTL_FILE	utlfile.sql	读/写基于ASCII字符的操作系统文件
UTL_HTTP	utlhttp.sql	从给定的URL得到HTML格式的主页
DBMS_LOB	dbmslob.sql	管理巨型对象

29.8 开始学习Oracle 8i提供的包

除了与Oracle 8i服务器一起安装的缺省包之外，还有许多可以与选择软件一起安装到服务器的其他包。高级队列选项，即Oracle 8i的一个新建包，就是一个这样的例子。

在能够使用包含在Oracle 8i提供的包中的程序前，应该首先检查它们已被安装，并且是合法的。数据库系统管理员应该运行下面的查询：

```
SELECT object_name, object_type, status
FROM dba_objects
WHERE owner='SYS' AND object_type LIKE 'PACKAGE%'
ORDER BY object_name, object_type;
```

上面的查询会给出一个类似下面信息的列表：

OBJECT_NAME	OBJECT_TYPE	STATUS
DBMS_ALERT	PACKAGE	VALID
DBMS_ALERT	PACKAGE BODY	VALID
DBMS_APPLICATION_INFO	PACKAGE	VALID
DBMS_APPLICATION_INFO	PACKAGE BODY	VALID
DBMS_DDL	PACKAGE	VALID
DBMS_DDL	PACKAGE BODY	VALID
DBMS_DESCRIBE	PACKAGE	VALID
DBMS_DESCRIBE	PACKAGE BODY	VALID
DBMS_JOB	PACKAGE	VALID
DBMS_JOB	PACKAGE BODY	VALID
DBMS_LOCK	PACKAGE	VALID
DBMS_LOCK	PACKAGE BODY	VALID
DBMS_OUTPUT	PACKAGE	VALID
DBMS_OUTPUT	PACKAGE BODY	VALID
DBMS_PIPE	PACKAGE	VALID
DBMS_PIPE	PACKAGE BODY	VALID
DBMS_REFRESH	PACKAGE	VALID
DBMS_REFRESH	PACKAGE BODY	VALID
DBMS_SESSION	PACKAGE	VALID
DBMS_SESSION	PACKAGE BODY	VALID
DBMS_SHARED_POOL	PACKAGE	VALID
DBMS_SHARED_POOL	PACKAGE BODY	VALID
DBMS_SNAPSHOT	PACKAGE	VALID
DBMS_SNAPSHOT	PACKAGE BODY	VALID
DBMS_SPACE	PACKAGE	VALID
DBMS_SPACE	PACKAGE BODY	VALID
DBMS_SQL	PACKAGE	VALID
DBMS_SQL	PACKAGE BODY	VALID

DBMS_SYSTEM	PACKAGE	VALID
DBMS_SYSTEM	PACKAGE BODY	VALID
DBMS_TRANSACTION	PACKAGE	VALID
DBMS_TRANSACTION	PACKAGE BODY	VALID
DBMS_UTILITY	PACKAGE	VALID
DBMS_UTILITY	PACKAGE BODY	VALID

还有一些其他包没有显示在上面，但是这里不必关心它们。这些包中的多数不是为用户应用调用而准备的，仅仅在内部使用而已。

29.8.1 定位DBMS包

Oracle 8i提供的包位于%Oracle_HOME%\RDBMS\ADMIN下，其中Oracle_HOME是指向Oracle主目录的路径（可以检查 UNIX系统中名字相同的系统变量或检查 Windows NT环境下的注册项）。可以检查表29-2中列出的每个包的包头文件，查看有哪些程序和全局变量可用。

或许还要注意具有prvt*.sql和prvt*.plb形式的文件（例如，prvtpipe.sql和prvtpipe.plb）的存在，前面一种是提供的包的包体，采用 ASCII格式；后面一种是包体的二进制编译版本。PLB文件对于PL/SQL引擎是可识别的，当提交给 Oracle8i服务器时，产生一个合法并且可执行的包体。它们代表包体的“发布”形式。

警告 不要试图修改任何Oracle 8i提供的包的包体并重新编译，这样会破坏一些东西。

29.8.2 确定包已正确安装

需要验证两件事：

- 包存在并且合法（就像前面的查询显示的一样）。
- 用户对它们拥有EXECUTE权限或者对用户打算使用的包拥有EXECUTE权限。

使用下面的查询，一个用户可以查看他们是否对 Oracle8i提供的包拥有执行权限：

```
SELECT table_name, grantee
FROM all_tab_privs
WHERE grantor='SYS' and privilege='EXECUTE'
ORDER BY table_name;
```

一个典型的回答应该如下所示：

TABLE_NAME	GRANTEE
DBMS_APPLICATION_INFO	PUBLIC
DBMS_DDL	PUBLIC
DBMS_DESCRIBE	PUBLIC
DBMS_JOB	PUBLIC
DBMS_OUTPUT	PUBLIC
DBMS_PIPE	PUBLIC
DBMS_SESSION	PUBLIC
DBMS_SNAPSHOT	PUBLIC
DBMS_SPACE	PUBLIC
DBMS_SQL	PUBLIC
DBMS_STANDARD	PUBLIC
DBMS_TRANSACTION	PUBLIC
DBMS_UTILITY	PUBLIC
DBMS_REFRESH	PUBLIC
UTL_FILE	PUBLIC

所有的Oracle 8i提供的包应该被授予 PUBLIC上的EXECUTE特权。我使用数据字典视图 all_flavor验证这点。如果打算使用的任何包丢失，数据库系统管理员可以运行适当的脚本重

新生成它们。如果包存在但却是非法的，数据库系统管理员可以使用下面的命令重新编译它：

```
ALTER PACKAGE SYS.<name> COMPILE PACKAGE;
```

其中name是非法包的名字。子句 COMPILE PACKAGE告诉Oracle8i重新编译包声明和包体。如果仅仅包体需要重新编译，运行下面的命令：

```
ALTER PACKAGE SYS.<name> COMPILE BODY;
```

提示 一些提供的包可能与其他包有相关性。如果你必须重新编译一个包头，会让所有相关的包非法。再次检查在重新编译后其他的包是否合法。如果只有包体被重新编译，相关的包不会变为非法。这种方式也有可能使任何用户写入的存储子程序和包受到影响。

29.9 使用Oracle 8i提供的包

下面将使用一些难易适度并且非常有用的例子表明每个提供的包中的一些有趣的内容。

29.9.1 使用DBMS_APPLICATION_INFO监控

这个包让开发者能够在视图 V\$sqlarea和V\$session中添加追踪信息，可以追踪的数据种类如下所示：

- 存储在V\$sqlarea.module和V\$session.module中的模块名称（例如当前运行着的应用的名称），最长可以是48字节的VARCHAR2字符串。
- 存储在V\$sqlarea.action和V\$session.action中的当前动作（例如，“更新客户信息”、“验证信用卡限额”），最长可以达到32个字节的VARCHAR2字符串。
- 存储在V\$session.client_info中的任意客户定义信息，最长可以达到 64个字节的 VARCHAR2字符串。

Oracle 8i不对这些信息做任何处理，这些信息提供给数据库系统管理员使用，这样她就可以针对每个应用和部件操作执行统计。超过所能支持长度的字符串将被截断。程序清单 29-12显示了一个简单的例子。

清单29-12 APPINFO.SQL —— 设置并读取应用信息

```
DECLARE
  module VARCHAR2(48); -- application info
  action VARCHAR2(32);
  client VARCHAR2(64);
  ldate VARCHAR2(30); -- to capture system date
BEGIN
  DBMS_OUTPUT.enable;
  module := 'CLAIM TRACKING';
  action := 'VERIFY ELIGIBILITY';
  DBMS_APPLICATION_INFO.set_module(module, action);
  DBMS_APPLICATION_INFO.set_client_info(USER);
  DBMS_APPLICATION_INFO.read_module(module, action);
  DBMS_APPLICATION_INFO.read_client_info(client);
  DBMS_OUTPUT.put_line(client || ' is running ' || module || ': ' || action);
  SELECT TO_CHAR(SYSDATE, 'YYYY-MON-DD HH:MI:SS') INTO ldate
  FROM DUAL;
END;
```

得到的响应信息如下所示：

```
SCOTT is running CLAIM TRACKING: VERIFY ELIGIBILITY
```

在会话启动期间，数据库系统管理员可以使用下面的程序检查谁在做什么：

```
COLUMN module FORMAT A20
COLUMN action FORMAT A20
COLUMN client_info FORMAT A20
SELECT client_info, module, action
FROM V$session
WHERE client_info IS NOT NULL;
```

这次得到下列信息：

CLIENT_INFO	MODULE	ACTION
SCOTT	CLAIM TRACKING	VERIFY ELIGIBILITY

数据库系统管理员可以使用下面的语句查看每个人运行这个模块和动作的效果：

```
SELECT
  sql_text, SUM(sharable_mem) smem, SUM(persistent_mem) pmem,
  SUM(runtime_mem) rmem, SUM(sorts) sorts, SUM(loads) loads,
  SUM(disk_reads) rdisk, SUM(buffer_gets) bget,
  SUM(rows_processed) prows
FROM V$sqlarea
WHERE module = 'CLAIM TRACKING' AND action = 'VERIFY ELIGIBILITY'
GROUP BY sql_text;
```

结果非常有趣：

SQL_TEXT	SMEM	PMEM	RMEM	SORTS	LOADS	RDISK	BGET	PROWS
SELECT TO_CHAR(SYSDATE, 'YYYY-MON-DD HH:MI:SS') FROM DUAL	4267	508	792	0	1	0	4	2
SELECT USER FROM SYS.DUAL	3596	508	688	0	1	0	4	1
begin dbms_output.get_lines(:lines, :numlines); end;	4317	592	420	0	1	0	0	1

我明确地做出的唯一查询是第一个查询，但是很明显 SQL*Plus在这之后做了一些工作。

因为一个模块执行不同的动作，开发者应该对 set_action () 发出正确的调用以反映这点。采用这种方式，可以收集到一些有趣的统计信息，这些信息反映了一个应用是如何通过查询、动作、模块、用户或用户组恶劣地冲击服务器的。

注意 甚至当用户断开连接之后，项仍然存在于 V\$sqlarea中直到相关的SQL过时，从SGA中消失为止。

29.9.2 使用DBMS_DDL重新编译包

可以对这个包做两件事：

- 使用alter_compile () 重新编译存储过程程序和包。
- 使用analyze_object () 分析一个表、索引或簇。

一个alter_compile () 的简单使用会发现哪些存储程序对象是合法的，然后重新编译它们。程序清单29-13表明这种情形。

清单29-13 RECOMPIL.SQL —— 只重新编译非法的程序对象

```
-- recompile invalid stored program objects
-- CAVEAT: does not take package dependencies
--      into account!
DECLARE
  CURSOR invalid_prog_obj IS
    SELECT object_name, object_type
      FROM user_objects
     WHERE status = 'INVALID';
  rec invalid_prog_obj%ROWTYPE;
  status NUMERIC;
BEGIN
  DBMS_OUTPUT.enable;
  OPEN invalid_prog_obj;
  LOOP -- recompile each stored program object
    FETCH invalid_prog_obj INTO rec;
    EXIT WHEN invalid_prog_obj%NOTFOUND;
    DBMS_OUTPUT.put('Recompile ' || rec.object_type ||
                   ' ' || rec.object_name);

    DBMS_DDL.alter_compile(rec.object_type, NULL, rec.object_name);
    DBMS_OUTPUT.put_line(' SUCCESSFUL'); -- recompile succeeded
  END LOOP; -- invalid program objects
  CLOSE invalid_prog_obj;
EXCEPTION
  WHEN OTHERS THEN
    BEGIN
      status := SQLCODE;
      DBMS_OUTPUT.put_line(' FAILED with ' || SQLERRM(status));
      IF (invalid_prog_obj%ISOPEN) THEN
        CLOSE invalid_prog_obj;
      END IF;
    EXCEPTION WHEN OTHERS THEN
      NULL; -- do nothing
    END;
END;
/
```

程序会返回如下所示的信息：

```
Recompile FUNCTION TABLE_EXISTS SUCCESSFUL
1 Program Objects Recompiled
PL/SQL procedure successfully completed.
```

警告 如果一个程序对象不能够成功地重新编译，alter_compile将不会通知你！在运行alter_compile后，应该检查包的状态，确认它被成功地编译。

如果提供了一个并不存在的程序对象名，或者输错了程序对象名称，会得到下列信息：

```
ORA-20000: Unable to compile PACKAGE "BLICK", insufficient privileges or does not exist
```

可以程序地分析（生成统计数据）表、索引和簇。例如可以分析模式中全部或选定的对象，如程序清单29-14所示。

清单29-14 runstats.sql —— 程序地运行统计

```
-- analyze all tables, indexes and clusters in your own schema
-- computes exact statistics
SET ECHO OFF
ACCEPT method PROMPT 'ANALYZE Method ([COMPUTE]|ESTIMATE|DELETE): '
ACCEPT estrow PROMPT ' IF ANALYZE Method is ESTIMATE, #Rows (0-n) [100]: '
```

```

ACCEPT estpct PROMPT ' IF ANALYZE Method is ESTIMATE, %Rows (0-99) [20]: '
DECLARE
  -- application-defined exceptions
  bad_method EXCEPTION; -- user entered an invalid method
  bad_estrow EXCEPTION; -- user entered an invalid est. #rows
  bad_estpct EXCEPTION; -- user entered an invalid est. %rows
  -- cursors
  CURSOR analyze_obj IS
    SELECT object_name, object_type
    FROM user_objects
    WHERE object_type = ANY ('TABLE', 'INDEX', 'CLUSTER');
  -- constants
  METHOD CONSTANT VARCHAR2(30) := NVL(UPPER('&method'), 'COMPUTE');
  -- variables
  estrow NUMBER := NVL('&estrow', '100'); -- user input est. #rows
  estpct NUMBER := NVL('&estpct', '20'); -- user input est. pct
  rec analyze_obj%ROWTYPE;
  status NUMERIC := 0;
  cnt NUMERIC := 0;
BEGIN
  DBMS_OUTPUT.enable;
  -- validate user input
  IF (METHOD NOT IN ('COMPUTE', 'ESTIMATE', 'DELETE')) THEN
    RAISE bad_method;
  ELSIF (METHOD IN ('COMPUTE', 'DELETE')) THEN -- ignore est. #/row
    estrow := NULL;
    estpct := NULL;
  ELSE -- picked ESTIMATE; must provide either est. #rows or %rows
    IF (estrow < 1 AND estpct = 0) THEN
      RAISE bad_estrow;
    ELSIF (estpct NOT BETWEEN 1 AND 99) THEN
      RAISE bad_estpct;
    END IF;
  END IF; -- validate input
  OPEN analyze_obj;
  LOOP -- analyze schema objects
    FETCH analyze_obj INTO rec;
    EXIT WHEN analyze_obj%NOTFOUND;
    -- COMPUTE STATISTICS for this schema only
    DBMS_OUTPUT.put('Analyze ' || METHOD || ' ' ||
      rec.object_type || ' ' || rec.object_name);
    DBMS_DDL.analyze_object(rec.object_type, NULL, rec.object_name, 'COMPUTE');
    DBMS_OUTPUT.put_line(' SUCCESSFUL');
    cnt := cnt + 1;
  END LOOP; -- analyze schema objects
  CLOSE analyze_obj;
  DBMS_OUTPUT.put_line(TO_CHAR(cnt) || ' objects analyzed');
EXCEPTION
  WHEN bad_method THEN
    DBMS_OUTPUT.put_line('Invalid Method! Must be COMPUTE, ESTIMATE or DELETE
  only');
  WHEN bad_estrow THEN
    DBMS_OUTPUT.put_line('Invalid Est. #Rows! Must be >= 1');
  WHEN bad_estpct THEN
    DBMS_OUTPUT.put_line('Invalid Est. %Rows! Must be between 1 and 99');
  WHEN OTHERS THEN
    BEGIN
      status := SQLCODE;
      DBMS_OUTPUT.put_line(' FAILED with ' || SQLERRM(status));
      IF (analyze_obj%ISOPEN) THEN
        CLOSE analyze_obj;
      END IF;
    EXCEPTION WHEN OTHERS THEN
      NULL;

```

```
END;
END;
/
```

注意必须做全部输入验证。部分原因是因为 analyze_object 函数自己对此做得很少。例如，如果提供了一条使用 COMPUTE 方法估算的纪录行，会得到下列信息：

```
Analyze TABLE <table> FAILED with ORA-01490: invalid ANALYZE command.
```

当选择全部默认值计算统计信息时，运行结果如下所示：

```
ANALYZE Method ([COMPUTE]|ESTIMATE|DELETE):
  IF ANALYZE Method is ESTIMATE; #Rows (1-n) [1]:
  IF ANALYZE Method is ESTIMATE, %Rows (1-99) [20]:
old 12: METHOD CONSTANT VARCHAR2(30) := NVL(UPPER('&method'), 'COMPUTE');
new 12: METHOD CONSTANT VARCHAR2(30) := NVL(UPPER(''), 'COMPUTE');
old 14: estrow NUMBER := NVL('&estrow', '1'); -- user input est.
#rows
new 14: estrow NUMBER := NVL('1', '1'); -- user input est. #rows
old 15: estpct NUMBER := NVL(TRUNC('&estpct'), '20'); -- user input est. pct
new 15: estpct NUMBER := NVL(TRUNC(''), '20'); -- user input est. pct
Analyze COMPUTE TABLE BONUS SUCCESSFUL
Analyze COMPUTE TABLE DEPT SUCCESSFUL
Analyze COMPUTE TABLE EMP SUCCESSFUL
Analyze COMPUTE INDEX PK_DEPT SUCCESSFUL
Analyze COMPUTE INDEX PK_EMP SUCCESSFUL
Analyze COMPUTE TABLE SALGRADE SUCCESSFUL
6 objects analyzed
PL/SQL procedure successfully completed.
```

如果输入了一个非法的方法，服务器返回下列信息：

```
Invalid Method! Must be COMPUTE, ESTIMATE or DELETE only
```

这正是所希望的，反复使用它你会看到你是如何喜欢它。

ANALYZE 命令的这种实现方式没有做 VALID STRUCTURE 和 LIST CHAINED ROWS。因此，它是一种不完全的实现过程，但是仍然相当有用。

29.9.3 使用 DBMS_OUTPUT 格式化输出

如果跟着教材学习，你已经非常熟悉包中的 put_line 过程，这里是关于这个包实现的一些细节。

- 每行使用一个换行符结束。
- 每行最多允许 255 个字节，包括换行符。
- 每行存储在私有的 PL/SQL 表中。
- 除非首先调用 DBMS_OUTPUT.ENABLE，否则不存储任何东西。
- 缓存大小范围规定为 2 000 至 1 000 000 之间。

使用过程 enable 打开输出特性。如果没有首先打开输出，put_line 调用将被忽略。当激活输出时，还要指定一个缓存大小，如下所示：

```
DBMS_OUTPUT.enable(1000000); -- 1 million bytes is the max
```

反之，可以使用 disable 关闭输出，如下所示：

```
DBMS_OUTPUT.disable; -- turn off output
```

可以使用 put_line 存储一行文本。该函数被重载，可以接受一个 DATE、NUMBER 或 VARCHAR2 数值。通过使用 put 过程（它也以同样方式被重载），还可以存储一行没有结束的

文本。如果创建的文本行需要一些逻辑，则这个过程非常有用，如下面的例子所示：

```
-- excerpt taken from Package rev_eng
IF (Ltable IS NULL) THEN -- parameter comments
  DBMS_OUTPUT.put('-- ALL TABLES');
ELSE
  DBMS_OUTPUT.put('-- TABLE ' || Ltable);
END IF;
DBMS_OUTPUT.put_line(' FOR OWNER ' || Lowner ||
                      ', TABLESPACE ' || Ltospace);
```

通过使用new_line过程，可以结束使用put过程提交的文本。该过程使用一个标志表示当前文本行结束，并将文本行的长度与文本行一起存储（这对于用户来说是完全透明的）。注意，如果试图不首先使用put过程就发送新文本行，以两倍或三倍行距打印输出行，该过程将不工作：

```
BEGIN
  DBMS_OUTPUT.put('Hey, ');
  DBMS_OUTPUT.put('Dan!');
  DBMS_OUTPUT.new_line;
  DBMS_OUTPUT.new_line;
  DBMS_OUTPUT.put_line('Time to go to the library!');
end;
/
```

这次给出下列信息：

```
Hey, Dan!
Time to go to the library!
PL/SQL procedure successfully completed.
```

请注意没有打印两倍行距。

如果试图在使用new_line过程结束一行前，输出一个超过255个字节可接受长度的字符串，会得到一个例外：

```
ORA-20000: ORU-10028: line length overflow, limit of 255 bytes per line.
```

为什么Oracle公司的人将字符串的长度限制为255？回顾前面的例子，在那里一个VARCHAR2（32767）类型的PL/SQL表很快地耗光了系统的可用内存（第28章“PL/SQL基本原理”，程序清单28-10）。换句话说，他们选择了一个他们认为合理的限制。

使用get_line返回字符串，这正是SQL*Plus返回使用put_line写入的字符串所做的事情。它以你看不见的方式调用get_line过程，直到再没有任何行可用为止。如果正在编写一个3GL程序接受使用put_line写入的文本行，确实只需要将注意力放到get_line（还有它的多行版本get_lines）过程的使用上。如果愿意，可以在一个PL/SQL程序中使用它，或许将它们插入一个表中，例如缓存文本行然后以FIFO方式访问它们。

缓存大小有什么用？毕竟，一个PL/SQL表可以含有超过四十亿的元素！设想你编写了一个C程序通过get_line过程接收缓存的字符串，分配一块4.29千兆×255字节的内存的确是一个浪费（而且也不是一个好主意）。你将需要上万亿字节的内存！虽然内存今天便宜了许多，但还没有便宜到那种地步。从而，你要求用户规定缓存的大小，比方说，在2K与1MB之间，并且那正是你所分配的。包追踪高速缓存空间的使用，因为你的C程序预期的正是用户指定的数据。然后将返回到该内存区的文本行压缩，浪费很少的空间或基本不浪费空间。最后，所有的文本行都可以显示给用户。这或许看起来不像是一个特别完善的模式，但它简单而且实用。

如果扩展缓存空间为所指定的大小，将得到下面的例外：

```
ORA-20000: ORU-10027: buffer overflow, limit of <buf_limit> bytes.
```

29.10 小结

本章学习了关于如何编写存储过程和包的丰富信息，还学习了如何使用与数据库服务器一起提供的包。学习了如何使用 SHOW ERRORS 命令调试存储程序，甚至能够编写一个脚本提高那些命令的输出。最后，学习了如何检查存储过程的合法性，以及如果存储过程变为非法时如何重新编译它们。现在应该能够理解本章中的例子并能够使用你自己的用户化要求来提高它们，或创建你自己设计的新过程。

第30章 使用导入与导出

本章要点：

- 理解导出与导入的目的与能力
- 理解行为
- 控制与配置导出与导入
- 导出与导入会话预排
- 使用SHOW与INDEXFILE选项
- 便携式表空间

30.1 理解导出与导入的目的与能力

Oracle8.x服务器提供了两个重要的应用工具：导入（Import）与导出（Export）。这两个应用工具对于许多重要的数据库功能非常有用，例如备份数据、在不同模式和数据库之间拷贝对象以及产生对象创建脚本。它们还可以用于将一个版本的 Oracle 移植到另一版本，并可用于将数据库升级到 Oracle8.x。

导出与导入工具提供了范围很广的能力。主要目的是备份并恢复数据及对象。对象和数据以二进制形式存储，只能够被 Oracle 数据库读取。导出与导入可以执行下列任务：

- 备份与恢复数据库 —— 导出与导入经常作为备份计划的一部分使用。通常，完全备份或热备份用于备份磁盘或计算机故障事件中的整个数据库。但是，如果在整个数据库上仅需要恢复一个表，将不得不在另外一台机器上恢复整个数据库，然后将这表拷贝过来，这会带来难以置信的时间与资源的消耗。导出与导入手段能够帮助你解决这一难题，它为用户提供了导出整个数据库并仅仅导入需要恢复的表的能力。它们还可以作为整个数据库的备份与恢复机制，同基于时间点的恢复（Point-in-time Recovery）一样是一种选择。
- 在模式之间转移数据和对象 —— 可以使用导出与导入将表、索引、授权、过程、视图，在所有其他的对象类型之间，从一个模式拷贝到另外一个模式中。因为能够指定需要转移的对象，所以这会帮助节省时间与精力。还有，转移数据可以被用作不同 Oracle 数据库之间数据复制的一种形式。
- 从一个版本将数据库移植到另一个版本 —— 通过使用导出与导入工具，可以升级（或降级）Oracle 数据库的版本。例如，可以导出一个完整的 Oracle7 数据库，然后假设你已经安装了 Oracle8 服务器，那么数据库可以被导入，令数据与应用在 Oracle8 环境中工作。这个过程称为移植。
- 整理一个表空间或整个数据库的碎片 —— 当对象例如表和索引随着时间的流逝被创建、删除、扩大或缩小时，就会出现磁盘碎片。当拙劣地定义对象存储参数时，也会出现磁盘碎片。通过导出一个表空间，合并空间并重新导入对象，可以整理表空间的磁盘碎片。

- 生成CREATE脚本 —— 还可以使用导出与导入工具在数据库中全部其他对象之间生成表、分区、视图、授权、索引、约束与表空间的 CREATE脚本。如果一个对象被破坏或删除，对于修改对象并保护对象的结构来说，导出与导入被证明是非常有用的。
- 在导出中写入多个文件 —— 导出支持写入多个导出文件，导入能够读取多个导出文件。如果在导出时，为导出文件 FILESIZE参数指定了一个值（字节限制），导出将只写入指定数量的字节数据到每个转储文件。在导入时，必须使用导入参数 FILESIZE告诉导入工具你在导出时指定的最大转储文件的大小。

FILESIZE参数有一个最大值，等于64位数据所能存储的最大值。

注意 64位数据存储的最大值依赖于所使用的操作系统（OS），在指定FILESIZE参数之前，应该在操作系统说明文档中确认这个最大值。

可以使用一个后缀K的数字（千字节）指定 FILESIZE参数。例如，FILESIZE=2K等同于FILESIZE=2048。同样，M表示兆字节（ 1024×1024 ），而G代表十亿字节（ $1024 \times 1024 \times 1024$ ）。B保留作为字节的缩写，最终文件大小数量并没有增加（FILESIZE=20 48b等同于FILESIZE=2048）。

- Oracle8i能够支持精密存取 —— 可以采用精密存取策略导出数据库表。但要注意，要恢复该策略，从含有这些表的导出文件中导入的用户必须拥有一定的权限（尤其是DBMS_RLS包上的执行权限，以便于恢复表的安全策略）。如果一个没有正确权限的用户试图从一个包含精密访问策略的表的导出文件中导入数据，将会产生一条警告消息。所以，建议出于安全原因，导出/导入这样表的人员应该是数据库系统管理员。

- Oracle 8i允许分区级导入 —— 按照下面的方法，可以导入表、分区和子分区：

- 表级导入 —— 从一个导出文件指定的表导入全部数据。
- 分区级导入 —— 只导入来自指定源分区或子分区中的数据。当向一个现有表中加载数据时，必须设置参数IGNORE=Y。有关参数IGNORE的信息，参阅本章后面“控制与配置导出与导入”部分参数表中的IGNORE项。
- Oracle 8i能够导出一部分表 —— 当进行表模式导出时，从一个表中选择记录的子集。查询参数的值是一个串，它在 SQL SELECT语句中包含WHERE子句，该子句将应用于TABLE参数中列出的全部表（或表分区）。

例如，如果SCOTT用户想要导出在部门10的雇员，他可以按照下面所示来做（注意这个例子是基于UNIX的）：

```
exp scott/tiger tables=emp query='"where deptno=10"'
```

注意QUERY参数的值是一个引号括起的串。在字符串的开头和结束使用单引号和双引号是UNIX特定的。请查阅操作系统说明文档了解如何在命令行上传递引号括起字符串的信息。当执行这条命令时，导出建立了一个类似下面的 SELECT语句：

```
SELECT * FROM EMP where deptno=10;
```

QUERY应用于在TABLE参数中列出的全部表（或表分区）。例如，

```
exp scott/tiger tables=emp,dept query='"where deptno=10"'
```

将卸载EMP表和DEPT表中符合查询条件的全部记录。还有，导出执行的 SQL语句类似下列的信息：

```
SELECT * FROM EMP where deptno=10;
SELECT * FROM DEPT where deptno=10;
```

注意 导出文件是二进制格式的文件，只能够被Oracle数据库使用。不能从Oracle数据库导出数据并将数据导入一个非Oracle数据库。同样，不能从一个非Oracle数据库中导入数据。如果想要从其他数据库产品例如微软的ACCESS复制数据到Oracle数据库，应该对一个分隔符文件格式的数据例如 CVS（逗号分割值）使用 SQL*Loader。要使用 Oracle 工具，将数据从 Oracle 数据库转移到一个非 Oracle 数据库，必须手工制作一个分隔符文件，从 PL/SQL 或 SQL*Plus 内部输出。

30.2 理解行为

有三种类型的导出：

- FULL（完全）导出 —— 导出数据库中的全部对象、结构和数据。
- OWNER（所有者）导出 —— 只导出那些属于特定用户帐户的对象。
- TABLE（表）导出 —— 只导出指定的表和分区。

因为许多结构可以被导出，区分使用这三种导出类别中每种不同的导出所得到的结果非常重要。表 30-1 显示了使用每种导出选项导出的结构（按照 Export 导出的顺序）。下面部分说明使用 Oracle8 进行导出与导入，运行 exp 和 imp 可执行程序语法。

注意 在 Windows NT 和 UNIX 环境中，exp 和 imp 是导出与导入工具的命令。

表30-1 三种导出选项和它们的对象

对象/模式	带有 FULL=Y 选项	带有 OWNER 选项	带有 TABLE 选项
表空间	全部对象定义		
环境资源文件	全部对象		
用户	全部对象定义		
角色	全部对象		
资源开销	全部对象		
回滚段	全部对象定义		
数据库链	全部对象	只有所有者	
序列	全部对象数目	只有所有者	
路径	全部对象别名		
外部函数	全部对象	只有所有者库名称	
对象类型	全部对象定义	只有所有者	
簇	全部对象定义	只有所有者	只有表和所有者
表	全部对象	只有所有者	只有表和所有者
索引	全部对象	只有所有者	只有表和所有者
引用完整性约束	全部对象	只有所有者	只有表和所有者
可传送的动作	全部对象	只有所有者	
同义词	全部对象	只有所有者	只有表和所有者
视图	全部对象	只有所有者	
存储过程	全部对象	只有所有者	
触发器	全部对象	只有所有者	只有表和所有者
快照	全部对象	只有所有者	
快照日志	全部对象	只有所有者	
作业队列	全部对象	只有所有者	

(续)

对象/模式	带有FULL=Y选项	带有OWNER选项	带有TABLE选项
刷新组和孩子	全部对象	只有拥有者	
用户历史表	全部对象		
缺省和系统审计选项	全部对象		

注意 在一个导入过程中，ROWIDS被重新赋值。如果一个表具有带有ROWID信息的列，数据将变为废弃过时的。还有，ROWID在一次导入后变为废弃过时的，在它们能够自动地再次刷新之前，必须使用COMPLETE选项刷新。另外，一些REF属性可能含有ROWID信息，因此在一个导入会话期间，该REF或许会变为过时的。要在导入数据后依据正确的ROWID重新创建REF信息，需要使用ANALYZE TABLE owner.table_name VALIDATE REF UPDATE；命令。

30.3 控制与配置导出与导入

导出与导入工具——带有许多能够被传递的可用参数——是非常灵活的。还有，有两种方法可以运行导出与导入：交互模式和非交互模式。使用交互模式，可以单步执行一系列的提示，为一个导出与导入会话输入基本信息。这种模式缺少灵活性，仅仅提示众多可用参数中的一小部分。

要使用交互模式导出，只需要在命令行输入exp命令即可。接着被提示输入用户名和口令。

警告 要导出整个数据库，导出所用的用户帐户必须被授予EXP_FULL_DATABASE系统特权。对于拥有数据库系统管理员权限的帐户，例如SYSTEM，这个特权被隐含地授权给该用户。否则导出操作将会失败。

接下来，收到一个提示要求指定数组取出缓存的大小。这是内存缓冲的大小，通过该缓存，记录被导出。这个数字应该大于最大的记录乘以希望在缓存中放置的记录数。这个大小与操作系统有关。

然后收到一个提示信息要求指定导出文件的名称。在缺省情况下，文件名为expdat.dmp。接下来，一个带有三种选项的小菜单出现了：

- 如果选择选项1，E（整个数据库），你会被提示为授权（Y/N）、表数据（Y/N）和压缩区间（Y/N）输入一个值。
- 如果选择选项2，U（用户），你会被提示为授权（Y/N）、表数据（Y/N）和压缩区间（Y/N）以及被导出的用户输入一个值（连续输入，然后在行上输入一个句号结束）。
- 如果选择选项3，T（表），你会被提示为授权（Y/N）、表数据（Y/N）和压缩区间（Y/N）以及要被导出的表（T）或分区（T：P）输入一个值（连续输入，然后在行上输入一个句号结束）。

要以非交互模式导出，可以在参数行传递所有的命令，或通过一个参数文件传递命令。对所有导出命令可用的参数，在Windows NT中输入exp HELP=Y，如图30-1所示。

在导出会话中可以使用23个参数，可以在命令行或指定的参数文件中指明它们。表30-2描述了所有的导出参数。



图30-1 在Windows NT环境, “exp HELP=Y”的简单结果

表30-2 导出实用工具的参数描述

参 数	缺 省 值	描 述
BUFFER	与操作系统有关	BUFFER的大小(字节)决定行导出时的内存缓存的大小。应该比最大的记录与想要在缓存中装入的行的数量的乘积大
COMPRESS	Y	如果COMPRESS=Y, INITIAL存储参数将被设置为对象分配的所有区间的总和大小, 只有在导入对象时这个改变才发生作用
CONSISTENT	N	选择CONSISTENT=Y以一致状态导出所有的表与指引。这将使导出变慢, 因为要使用回滚空间。如果 CONSISTENT=N, 这是缺省状态, 如果在导出过程中, 有一条记录发生了改变, 数据会成为不一致的
CONSTRAINTS	N	指明是否导出表约束
DIRECT	N	如果DIRECT=Y, Oracle忽略SQL命令处理层, 提高导出速度。不幸的是, Oracle8的新的常用对象类型, 如LOB, 不能被导出
FEEDBACK	0	Oracle为每组记录的插入显示一个句号。组的大小由FEEDBACK定义, 例如, 如果设置 FEEDBACK=1000, 每导入 1000条记录, 显示一个句号。这个参数对跟踪大量导入的进展非常有用
FILE	expdat.dmp	缺省情况下, expdat.dmp(代表导出data.dump)将是文件名, 要使用更有意义的文件名, 改变 FILE参数
FULL	N	如果FULL=Y, 将导出整个数据库, 包括表空间定义
FILESIZE		每个卸载文件的最大长度
GRANTS	Y	指明是否导出正被导出对象的全部授权定义
HELP	N	如果指明HELP=Y, 不需要其他的参数。将显示一个基本的帮助屏幕
INCTYPE		INCTYPE参数的合法选项是 COMPLETE、CUMULATIVE与 INCREMENTAL。COMPLETE导出建造一个另两个选项依赖的完全导出, 用以重建数据库。 CUMULATIVE导出自上次 CUMULATIVE或COMPLETE导出发生以来所有改变的表与其他的对象。如果表中有一条记录发生了改变, 导出整个表。 INCREMENTAL导出上次 INCREMENTAL、CUMULATIVE或 COMPLETE导出以来发生改变的所有的表与对象

(续)

参 数	缺 省 值	描 述
INDEXES	Y	指明是否导出用户定义索引。系统索引创建时具有约束（主键、唯一键），OID索引自动导出，不管INDEXES参数的值
LOG		LOG参数指明放置导出会话的反馈的文件的名称。除非另外声明，Oracle对文件添加一个.LOG扩展名
PARFILE		代替在命令行输入所有的参数，一些参数或所有的参数可以保留在参数文件中。PARFILE参数指明如果需要，使用哪个文件。这个参数对非交互导入会话非常有用
POINT_IN_	N	对TABLESPACES参数列出的表空间基于时间点的恢复
QUERY		WHERE子句用于导出表的一个子集
RECORD	Y	如果使用具有RECORD=Y的INCTYPE参数，使用导出数据如拥有者、导出类型与导出时间填充 SYS数据字典表 INCEXP、INCFILE与INCVID
RECORD_LENGTH	与操作系统有关	RECORDLENGTH参数只用于当导出到一个与导出发生时具有不同位数机器的时候。在大多数导入会话中，应该使用缺省值
RECOVERY_		RECOVERY_TABLESPACES，与POINT_IN_TIME_RECOVER
TABLESPACES		参数联合使用，指明使用基于时间点的恢复将恢复哪个表空间。这很重要，因为否则导入会恢复导出之后的事务
ROWS	Y	指明是否导出表与对象的数据。如果ROWS=N，只导出对象定义
STATISTICS	ESTIMATE	指明在导入时是否使用 COMPUTE或ESTIMATE分析表与索引统计。注意在导入时只能分析那些早已具有统计信息的对象。如果没有对象需要分析，指明NONE
TABLES		指明一个以逗号分隔的要被导出的表的清单。这个参数应该与FROMUSER参数联合使用，在非UNIX环境中，如Windows NT中，必须使用括号括起表清单
TABLESPACES		使用POINT_IN_TIME_RECOVER参数导出表空间清单
USERID		指明执行导入的用户的用户名与口令。这条命令的格式为：用户名/口令。如果需要，也可以使用NET8的@连接串格式
VOLSIZE		写入每个磁带卷的字节数。VOLSIZE指明每个磁带卷上导出文件的最大字节数，VOLSIZE参数具有的最大值等于以64位存储的最大值

使用交互模式导入，只需在命令行输入 imp，然后会收到用户名与口令的提示。

警告 为使用DBA调用的导出进行导入，导入的用户帐号必须已被赋予 IMP_FULL_DATABASE系统特权。对具有DBA特权的帐户，如SYSTEM，这个特权被隐式地授予该用户，否则，导入会失败。

然后收到一个提示，要求输入要导入的导出文件名。缺省情况下是 expdat.dmp。下面，要求输入数组提取缓冲区大小，这是行将被导出的内存缓存的大小。

接着将被提示是否想列出导出文件的内容清单（是/否）。如果选择是，结果在本章后面的名为“使用SHOW与INDEXFILE选项”一节描述。如果选择否，提问你是否忽略所有的由于对象已存在而产生的创建错误（是/否）。

然后，提问你是否导入授权、表数据与整个导出文件。如果选择否，导入整个导出文件，接收到提示要求输入与对象所有者对应的用户名，随后会有重复的提示要求输入所有的表与分区。如果保留该行为空，则假设是用户的所有对象。要停止重复提示，在提示符后输入一

个句号。

为在非交互模式中使用导入，可以在命令行传递所有参数，或者通过一个参数文件传递参数。输入“imp HELP=Y”以获得所有导入命令可使用的参数，如图 30-2所示。



图30-2 Windows NT平台上“imp HELP=Y”导出示例

在导入会话中可以使用 27 个参数。可以在命令行指明它们，或在任何指定的参数文件中进行设定。表30-3描述了所有的导入参数。

表30-3 导入实用工具的参数描述

参 数	缺 省 值	描 述
ANALYZE	Y	如果ANALYZE设为Y，导入的表将具有它们的统计分析。注意只有那些在导出时早已具有统计的表才会被计算。缺省情况下表将被ESTIMATED，除非导出执行的是STATISTICS=COMPUTE参数配置
BUFFER	与操作系统有关	BUFFER的大小（字节）决定行导入时的内存缓存的大小。应该比最长的记录与想要在缓存中装入的行的数量的乘积大
CHARSET		CHARSET是一个过时的 Oracle6参数，指明是以 ASCII还是 EBCDIC导出。在 Oracle7与Oracle8中，这个信息是自动处理的
COMMIT	N	缺省情况下，在每个表、嵌套表与分区之后进行一次提交。如果正在导入一个巨型表，回滚段会变得很大。在装载巨型表时，为提高性能，可以设置COMMIT=Y
DESTROY	N	如果设置DESTROY=Y并且进行一个完全导入，Oracle会覆盖所有现存的数据文件。如果使用裸盘设备作为数据文件，在一个完全导入时，裸盘会被重写，因为 DESTROY=Y并不会阻止重写数据文件！在进行这种导入之前最好进行一次数据库的备份。不要使用这个选项，除非确切知道自己正在做什么
FEEDBACK	0	Oracle为每组记录的插入显示一个句号。组的大小由 FEEDBACK定义，例如，如果设置 FEEDBACK=1000，每导入 1000条记录，显示一个句号。这个参数对跟踪大量导入的进展非常有用

(续)

参 数	缺 省 值	描 述
FILE	expdat.dmp	缺省情况下, expdat.dmp (代表导出data.dump) 是要被导入的导入文件名, 如果文件名不是 expdat.dmp, 在FILE参数中指定文件名
FILESIZE		每个转储文件的最大长度 —— 如果从多个文件导入
FROMUSER		指明这个参数将只导入那些由 FROMUSER用户帐户拥有的对象。
FULL	N	如果FULL=Y, 将导出整个数据库
GRANTS	Y	指明是否对导出对象创建所有的授权
HELP	N	如果指明HELP=Y, 不需要其他的参数, 将显示一个基本的帮助屏幕
IGNORE	N	如果IGNORE=Y, 忽略对象创建错误, 并将记录插入表中。要清楚如果在表中没有唯一性约束, 有可能产生重复的记录。注意仍将报告非对象创建错误, 如操作系统问题
INCTYPE		如果导入一个增长性导出, 表被删除并重建。必须首先从最新的SYSTEM导出进行重建 (指明 INCTYPE=SYSTEM)。然后, 导入每个增长性导出 (指明 INCTYPE=RESTORE), 直至需要的改变已应用到数据库为止
INDEXES	Y	指明是否导入用户定义索引。系统索引创建时具有约束 (主键、唯一键), OID索引不管INDEXES参数的值, 自动导入
INDEXFILE		INDEXFILE参数指明生成CREAT INDEX语句的文件名。除非另外指明, Oracle为该文件附加 .SQL 扩展名。可以在本章稍后找到INDEXFILE参数的详细描述
LOG		LOG参数指明存放导入会话的反馈的文件的名字。除非另外声明, Oracle对文件附加一个.LOG扩展名
PARFILE		代替在命令行输入所有的参数, 一些参数或所有的参数可以保留在参数文件中。PARFILE参数指明如果需要, 使用哪个文件。这个参数对非交互导入会话非常有用
POINT_IN_TIME	N	对使用TABLESPACES参数导出的表空间RECOVER执行基于时间点的恢复
RECORDLENGTH	与操作系统有关	RECORDLENGTH参数只用于当你导出一个与导出发生时具有不同文件位数机器的时候。在大多数导入会话中, 应该使用缺省值
RECALCULATE_STATISTICS	N	重新计算统计值, 用于基于成本的优化
SHOW	N	SHOW参数在屏幕上显示每条SQL语句, 并且不修改数据库。当与FILE参数联合使用时, 可以看到并可以修改 SQL语句。在本章后面的“使用 SHOW与INDEXFILE选项”一节可以找到 SHOW参数的详细描述
SKIP_UNUSABLE_INDEX	N	SKIP_UNUSABLE_INDEXES可以延迟索引INDEXES的创建直至记录的数据被导入后, 只有那些设置为不使用状态的索引受到影响, 如果INDEXES=Y (这是缺省值), 其他的索引将被创建
TABLES		指明一个以逗号分隔的要被导入的所有表的清单。这个参数应该与FROMUSER参数联合使用, 在非UNIX环境中, 如Windows NT中, 必须使用括号括起表清单
TOUSER		如果预期要导入到与表的最初拥有者不同的用户帐户中, TOUSER参数指明将表导入到哪个用户帐户。这个参数需要与FROMUSER参数联合使用
TOID_NOVALIDATE		当导入引用一个类型的表, 而在数据库中早已存在这个名字的一个类型时, 导入试图验证事先存在的类型的确是表所使用的类型 (而不是恰巧具有相同名字的不同类型)。要这样做, 导入将类型的唯一标识 (TOID) 与导出文件中存储的标识比较, 如果 TOID

(续)

参 数	缺 省 值	描 述
USERID		不匹配, 将不会导入表记录。在有些情况下, 在指明类型时, 不希望发生这种校验 (例如, 如果类型是由插件安装创建的), 可以使用 TOID_NOVALIDATE 参数指明 TOID 比较排除的类型 指明执行导入的用户的用户名与口令。这条命令的格式为: 用户名/口令。如果需要, 也可以使用 NET8 的 @ 连接串格式

30.4 导出与导入会话预排

这一节单步调试一些导入与导出会话的预排。这些预排演示了这两个工具的一些有用的功能, 并指出一些应该避免的缺陷。

30.4.1 当表存在时识别行为

如果试图导入一个数据库中早已存在的表, 会产生一个错误。导入跳过这个表 (伴有表上的外键与索引), 继续进行导入处理, 导入所有指定的其他的表与对象, 如图 30-3 所示。

如果想要将数据导入一个早已存在的表中, 使用 IGNORE=Y 参数声明。这会导致数据追加到表中。如果违反约束, 如具有相同主键的重复记录, 不装载那些违反约束的记录。约束的存在对于防止向表中导入重复记录是有帮助的。

例如, 假设 DEMO 用户帐户拥有 EMPLOYEE 表, 在表中拥有记录。尝试启动一个向表中装载重复记录的导入, 图 30-4 显示这种行动的结果。

```

MS-DOS Prompt
C:\>imp scott/tiger
Import: Release 8.1.3.0.0 - Beta on Sun Mar 7 12:12:44 1999
(c) Copyright 1998 Oracle Corporation. All rights reserved.

Connected to: Oracle8 Enterprise Edition Release 8.1.3.0.0 - Beta
With the Partitioning and Objects options
PL/SQL Release 8.1.3.0.0 - Beta
Import file: EXPDMP.DMP >
Enter insert buffer size (minimum is 8192) 30720
Export file created by EXPORT:V08.01.03 via conventional path
Import done in UEBIS08857PI character set and UEBIS08857PI NCHAR character set
List contents of import file only (yes/no): no >
Ignore create error due to object existence (yes/no): no >
Import grants (yes/no): yes >
Import table data (yes/no): yes >
Import entire export file (yes/no): no >
Username: scott
Enter table(s) or partition(s): names. Null list means: all tables for user
Enter table(s) or partition(s): name or . if done: emp
Enter table(s) or partition(s): name or . if done:
Importing SCOTT's objects into SCOTT
IMP-00015: following statement failed because the object already exists:
"CREATE TABLE 'EMP' ('EMPNO' NUMBER(4, 0), 'ENAME' VARCHAR2(10), 'JOB' VARCHAR2(9), 'MGR' NUMBER(4, 0), 'HIREDATE' DATE, 'SAL' NUMBER(7, 2), 'COMM' NUMBER(2, 2), 'DEPTNO' NUMBER(2, 0)) PCTFREE 10 PCTUSED 40 INITIALLY 1 MAXIMUM 255 LOGGING STORAGE(INITIAL 10240 NEXT 10240 MINEXTENTS 1 MAXEXTENTS 121) NO INCREASE 50 FREELISTS 1 FREELIST GROUPS 1 BUFFER POOL DEFAULT TABLESPACE 'SYSTEM'"
Import terminated successfully with warnings.
C:\>

```

图30-3 导入一个数据库中早已存在的表时产生的错误

```

MS-DOS Prompt
C:\>imp scott/tiger
Import: Release 8.1.3.0.0 - Beta on Sun Mar 7 12:12:44 1999
(c) Copyright 1998 Oracle Corporation. All rights reserved.

Connected to: Oracle8 Enterprise Edition Release 8.1.3.0.0 - Beta
With the Partitioning and Objects options
PL/SQL Release 8.1.3.0.0 - Beta
Import file: EXPDMP.DMP >
Enter insert buffer size (minimum is 8192) 30720
Export file created by EXPORT:V08.01.03 via conventional path
Import done in UEBIS08857PI character set and UEBIS08857PI NCHAR character set
List contents of import file only (yes/no): no >
Ignore create error due to object existence (yes/no): no >
Import grants (yes/no): yes >
Import table data (yes/no): yes >
Import entire export file (yes/no): no >
Username: scott
Enter table(s) or partition(s): names. Null list means: all tables for user
Enter table(s) or partition(s): name or . if done: emp
Enter table(s) or partition(s): name or . if done:
Importing SCOTT's objects into SCOTT
IMP-00015: following statement failed because the object already exists:
"CREATE TABLE 'EMP' ('EMPNO' NUMBER(4, 0), 'ENAME' VARCHAR2(10), 'JOB' VARCHAR2(9), 'MGR' NUMBER(4, 0), 'HIREDATE' DATE, 'SAL' NUMBER(7, 2), 'COMM' NUMBER(2, 2), 'DEPTNO' NUMBER(2, 0)) PCTFREE 10 PCTUSED 40 INITIALLY 1 MAXIMUM 255 LOGGING STORAGE(INITIAL 10240 NEXT 10240 MINEXTENTS 1 MAXEXTENTS 121) NO INCREASE 50 FREELISTS 1 FREELIST GROUPS 1 BUFFER POOL DEFAULT TABLESPACE 'SYSTEM'"
Import terminated successfully with warnings.
C:\>

```

图30-4 试图向具有唯一性约束的表中导入重复记录示例

30.4.2 重新组织一个碎片表空间

一个碎片表空间具有许多不连续的自由空间块。碎片会导致性能与空间问题。性能受到影响是因为 Oracle 不得不扫描多个对象区间，并有可能跨越多个物理磁盘。当数据进行磁盘碎片整理时，对象可以从多个区间压缩为一个区间，这会减少扫描数据时的内部 Oracle 开销。

相反，一个碎片表空间影响对象存储。具有许多小的自由空间块延伸跨越多个表空间，一些对象可能不会被创建，而如果所有的自由空间是连续的，空间将足够用于创建该对象。通过表空间磁盘碎片整理，重新组织数据以使所有小的自由块形成一个自由块，如图 30-5 所示。



图30-5 碎片空间与非碎片空间示例

当对象如表和索引随时间创建、删除、增大或减小尺寸时，会发生碎片。由于 Oracle 只能在表空间中的一个连续的自由空间内创建一个区间，会开始出现一些小的自由空间。当删除一个对象时，它的自由空间很有可能处于表空间中分散的地点，只能在那个自由空间中创建另一个相等的或小一些的对象。

为检查表空间的碎片，可以检查在指定的表空间中有多少自由空间段，以及它们的大小。这可以通过运行 CHP30_1.SQL 脚本并传递表空间名实现，如程序清单 30-1 所示。可以在 SQL*PLUS、服务器管理器或任何可以连接到 Oracle8 数据库的第三方产品中运行这个脚本。

清单30-1 CHP_30.SQL —— 决定表空间内自由空间的数量与大小的脚本

```
SQL> START CHP30_1.SQL
SQL> COLUMN FILE_NAME FORMAT A40
SQL> SELECT A.TABLESPACE_NAME, B.FILE_NAME, A.BYTES
2 FROM DBA_FREE_SPACE A, DBA_DATA_FILES B
3 WHERE A.TABLESPACE_NAME = '&tablespace_name' AND
4 A.FILE_ID = B.FILE_ID
5 ORDER BY BYTES DESC
```

当运行这个脚本时，服务器会提示输入表空间名：

Enter value for tablespace_name: USERS

服务器将返回自由空间信息：

TABSPACE_NAME	FILE_NAME	BYTES
USERS	/u02/oradata/TEST/users01.dbf	32768
USERS	/u02/oradata/TEST/users01.dbf	114688
USERS	/u04/oradata/TEST/users02.dbf	122400
USERS	/u04/oradata/TEST/users02.dbf	101908480SQL>

对每一个不同的FILE_NAME, 应该各有一条记录。如果没有, 表空间就是碎片的。在表空间中含有的表在导出与导入过程中会成为用户不可用的。在依赖导入与导入表空间进行表空间碎片整理时, 应该输入命令 ALTER TABLESPACE 表空间名 COALESCE并再次运行 CHP30_1.SQL。如果自由空间的两个大块彼此毗邻, 这条命令使它们组成一个更大的块。如果这条命令没有作用, 必须使用导出与导入以整理表空间碎片。

另一个整理表空间碎片的原因是是否表空间内的一个对象含有多个区间的情况。在大多数情况下, 如果对象具有多于五个区间的情况, 应该加以关注, 因为在这点之后性能开始受到显著影响。通过指明 COMPRESS=Y 导出表, Oracle 计算初始区间的尺寸以将所有的数据放入一个区间, 这也会帮助清除数据库碎片。

运行CHP30_2.SQL以确定哪个对象具有多于五个区间, 如程序清单 30-2所示。

清单30-2 CHP30_2.SQL —— 列出数据库中具有多于五个区间的所有对象的脚本

```
SQL> START CHP30_2.SQL
SQL> COLUMN SEGMENT_NAME FORMAT A25
SQL> SELECT OWNER, SEGMENT_NAME, EXTENTS
2 FROM DBA_SEGMENTS
3 WHERE EXTENTS > 5 AND
4 OWNER NOT IN ('SYS','SYSTEM')
5 ORDER BY EXTENTS
```

在运行这个脚本之后, 服务器返回具有多于五个区间的所有对象。

OWNER	SEGMENT_NAME	EXTENTS
DEMO	EMPLOYEE	6
DEMO	JOB	9
DEMO	DEPARTMENT	12

在开始进行碎片整理之前, 确信已通知到所有将受影响的人, 因为在这个处理过程中, 表空间中的表将成为不可用的。如果可能, 计划在没有人访问这些表的时刻进行这个处理。

为使用导入与导出以整理一个表空间的碎片, 应遵循以下步骤:

- 1) 导出表空间包含的所有表, 确保已设置导出的 COMPRESS=Y 选项。如果必要, 它会改变表的 INITIAL 存储参数, 以使每个表适合一个区间。

- 2) 手工从表空间中删除所有的表。

- 3) 合并表空间中的自由空间, 这由 ALTER TABLESPACE 表空间名 COALESCE 命令完成。所有的自由空间将被合并为一个大块, 或与表空间的数据文件一样多的块。这是由于表空间中不含有对象。

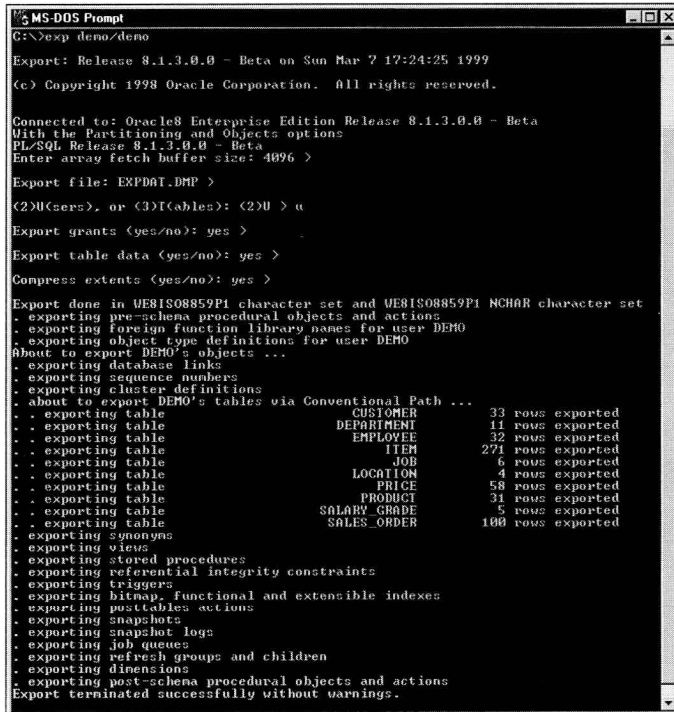
- 4) 导入表空间包含的所有表。Oracle 为每个对象分配合适的空间, 因为在导出时已指明 COMPRESS=Y。完成时, 便具有了一个干净的、没有碎片的表空间了。

警告 如果使用 COMPRESS=Y 进行导出, 任何被导出的 LOB 数据将不会被压缩, 它的初始 INITIAL 与 NEXT 存储参数不会改变。

前一种方法需要了解表空间中含有的所有对象。在一个更消耗时间的方法中, 可以清除整个数据库的碎片。通过以下方法实现: 导出整个数据库, 使用 CREATE DATABASE 命令删除并重建数据库, 然后导入整个导出文件。

30.4.3 在模式间移动数据库对象

许多时候，在模式之间移动对象非常重要。下面就是这种情况，一个开发者想要一套表



```
MS-DOS Prompt
C:\>exp demo/demo

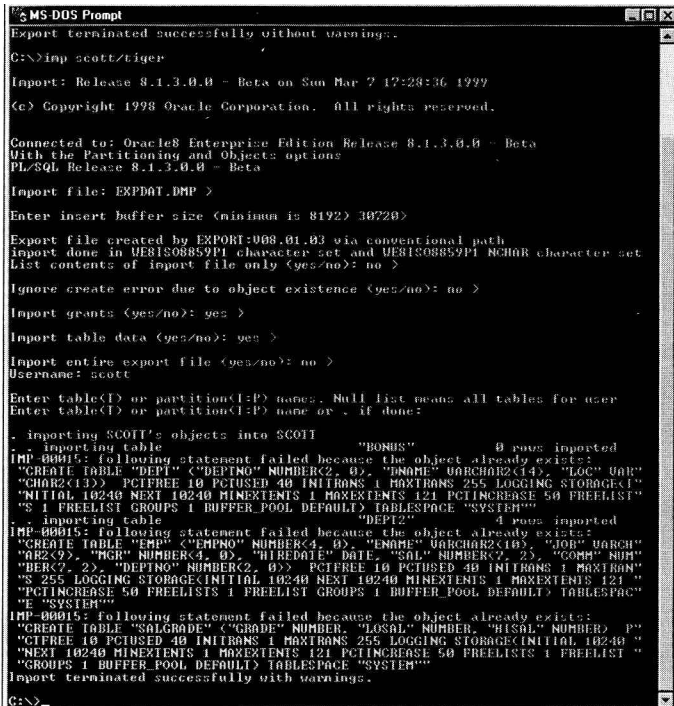
Export: Release 8.1.3.0.0 - Beta on Sun Mar 7 17:24:25 1999
(c) Copyright 1998 Oracle Corporation. All rights reserved.

Connected to: Oracle8 Enterprise Edition Release 8.1.3.0.0 - Beta
With the Partitioning and Objects options
PL/SQL Release 8.1.3.0.0 - Beta
Enter array fetch buffer size: 4096 >

Export file: EXPDAT.DMP >
(2)U(cons), or (3)T(ables): (2)U > u
Export grants (yes/no): yes >
Export table data (yes/no): yes >
Compress extents (yes/no): yes >

Export done in VERIS08859P1 character set and VERIS08859P1 NCHAR character set
. exporting pre-schema procedural objects and actions
. exporting foreign function library names for user DEMO
. exporting object type definitions for user DEMO
About to export DEMO's objects ...
. exporting database links
. exporting sequence numbers
. exporting cluster definitions
. about to export DEMO's tables via Conventional Path ...
. . exporting table
. . . DEPARTMENT 33 rows exported
. . exporting table
. . . EMPLOYEE 32 rows exported
. . exporting table
. . . ITEM 271 rows exported
. . exporting table
. . . JOB 6 rows exported
. . exporting table
. . . LOCATION 4 rows exported
. . exporting table
. . . PRICE 58 rows exported
. . exporting table
. . . PRODUCT 31 rows exported
. . exporting table
. . . SALARY_GRADE 5 rows exported
. . exporting table
. . . SALES_ORDER 100 rows exported
. exporting synonyms
. exporting views
. exporting stored procedures
. exporting referential integrity constraints
. exporting triggers
. exporting bitmap, functional and extensible indexes
. exporting posttables actions
. exporting snapshots
. exporting snapshot logs
. exporting job queues
. exporting refresh groups and children
. exporting dimensions
. exporting post-schema procedural objects and actions
Export terminated successfully without warnings.
```

图30-6 拷贝DEMO模式到QUE模式



```
MS-DOS Prompt
Export terminated successfully without warnings.
C:\>imp scott/tiger

Import: Release 8.1.3.0.0 - Beta on Sun Mar 7 17:28:36 1999
(c) Copyright 1998 Oracle Corporation. All rights reserved.

Connected to: Oracle8 Enterprise Edition Release 8.1.3.0.0 - Beta
With the Partitioning and Objects options
PL/SQL Release 8.1.3.0.0 - Beta
Import file: EXPDAT.DMP >
Enter insert buffer size (minimum is 8192): 30720 >

Export file created by EXPORT=0008.01.03 via conventional path
Import done in VERIS08859P1 character set and VERIS08859P1 NCHAR character set
List contents of import file only (yes/no): no >
Ignore create error due to object existence (yes/no): no >
Import grants (yes/no): yes >
Import table data (yes/no): yes >
Import entire export file (yes/no): no >
Username: scott
Enter table(s) or partition(s) names. Null list means all tables for user
Enter table(s) or partition(s) name or * if done:
. importing SCOTT's objects into SCOTT
. importing table "BONUS" 0 rows imported
IMP-00015: following statement failed because the object already exists:
"CREATE TABLE "DEPT" ("DEPTNO" NUMBER(2, 0), "DNAME" VARCHAR2(14), "LOC" VARCHAR2(13)) PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 LOGGING STORAGE(INITIAL 10240 NEXT 10240 MINEXTENTS 1 MAXEXTENTS 121 PCTINCREASE 50 FREELIST "S" 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT) TABLESPACE "SYSTEM""
. importing table "DEPT2" 4 rows imported
IMP-00015: following statement failed because the object already exists:
"CREATE TABLE "EMP" ("EMPNO" NUMBER(4, 0), "ENAME" VARCHAR2(16), "JOB" VARCHAR2(9), "MGR" NUMBER(4, 0), "HIREDATE" DATE, "SAL" NUMBER(7, 2), "COMM" NUMBER(7, 2), "DEPTNO" NUMBER(2, 0)) PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 LOGGING STORAGE(INITIAL 10240 NEXT 10240 MINEXTENTS 1 MAXEXTENTS 121 PCTINCREASE 50 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT) TABLESPACE "SYSTEM""
IMP-00015: following statement failed because the object already exists:
"CREATE TABLE "SALGRADE" ("GRADE" NUMBER, "LOCAL" NUMBER, "RISAL" NUMBER, "CTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 LOGGING STORAGE(INITIAL 10240 NEXT 10240 MINEXTENTS 1 MAXEXTENTS 121 PCTINCREASE 50 FREELISTS 1 FREELIST GROUPS 1 BUFFER_POOL DEFAULT) TABLESPACE "SYSTEM""
Import terminated successfully with warnings.
```

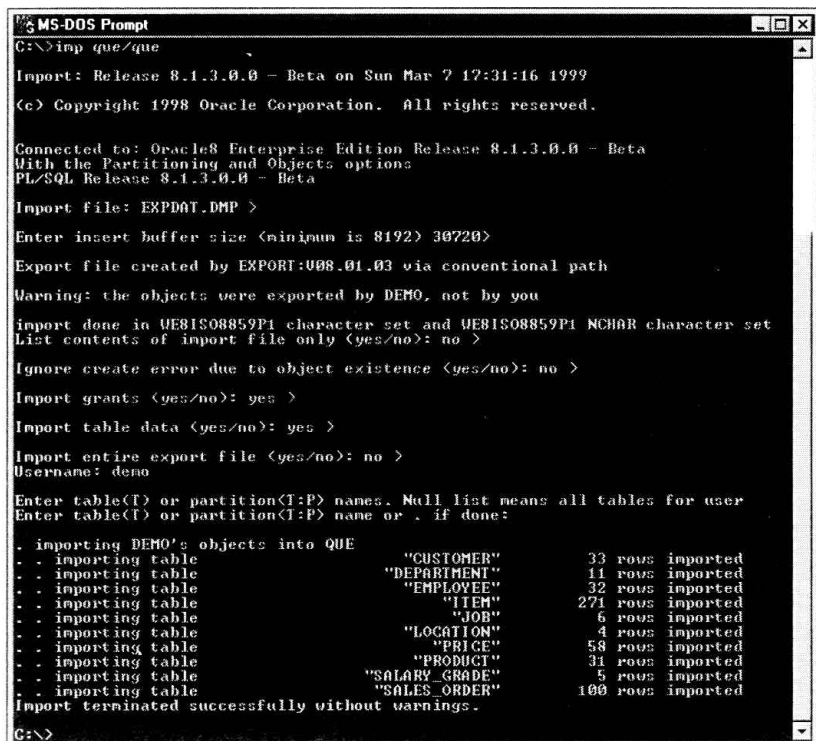
图30-7 导出DEMO用户帐户中的所有对象

用于测试，但不想影响在最初的模式中的数据。这对在实例之间拷贝表也很有用。

首先，导出一个给定用户帐户的所有对象。通过指明导出工具中的 OWNER 参数完成这步。在这个例子中，将拷贝 DEMO 模式到 QUE 模式中，如图 30-6 所示。

如果导入的任何对象与现有的对象冲突，只跳过那些冲突的对象。

接着如果帐户不存在，创建成为新的拥有者的用户帐户。在这一点，可以通过指明导入工具中的 FROMUSER 与 TOUSER 参数导入到新模式中，如图 30-7 所示。



```
MS-DOS Prompt
C:\>imp que/que

Import: Release 8.1.3.0.0 - Beta on Sun Mar 2 17:31:16 1999
(c) Copyright 1998 Oracle Corporation. All rights reserved.

Connected to: Oracle8 Enterprise Edition Release 8.1.3.0.0 - Beta
With the Partitioning and Objects options
PL/SQL Release 8.1.3.0.0 - Beta

Import file: EXPDAT.DMP >
Enter insert buffer size (minimum is 8192) 30720
Export file created by EXPORT:V08.01.03 via conventional path
Warning: the objects were exported by DEMO, not by you
Import done in WE8ISO8859P1 character set and WE8ISO8859P1 NCHAR character set
List contents of import file only (yes/no): no >
Ignore create error due to object existence (yes/no): no >
Import grants (yes/no): yes >
Import table data (yes/no): yes >
Import entire export file (yes/no): no >
Username: demo
Enter table(T) or partition(T:P) names. Null list means all tables for user
Enter table(T) or partition(T:P) name or . if done:
. importing DEMO's objects into QUE
. importing table "CUSTOMER" 33 rows imported
. importing table "DEPARTMENT" 11 rows imported
. importing table "EMPLOYEE" 32 rows imported
. importing table "ITEM" 271 rows imported
. importing table "JOB" 6 rows imported
. importing table "LOCATION" 4 rows imported
. importing table "PRICE" 58 rows imported
. importing table "PRODUCT" 31 rows imported
. importing table "SALARY_GRADE" 5 rows imported
. importing table "SALES_ORDER" 100 rows imported
Import terminated successfully without warnings.
C:\>
```

图30-8 将所有的DEMO用户帐户的对象导入到QUE帐户

提示 如果具有BFILE数据类型，Oracle只存储指向文件自身的指针，真正的数据存储于数据库外。如果从一个数据库导出并导入到不同服务器上的另一个数据库中，确信已拷贝所有的外部文件并已将它们放置在相同的路径下，否则，Oracle不能访问有关的BFILE文件。

30.4.4 多个对象与多个对象类型

如果想要将表的一个子集从一个模式移动到另一个模式中，或出于备份目的，在导出时需要在TABLES参数声明中指明对象清单。这必须与导入的 FROMUSER 与 TOUSER 联合使用。

要导出一个表，指明拥有者，随后是一个点与表名。如拥有者 .表名。为导出一个分区，指明拥有者，随后是点与表名，然后是冒号，紧接着是分区名，如拥有者 .表名：分区名。

图30-9显示了导出多个对象与多个对象类型的一个示例：PRICE表、EMP表五个分区中的两个（LOW_SALARY与MEDIUM_SALARY）。

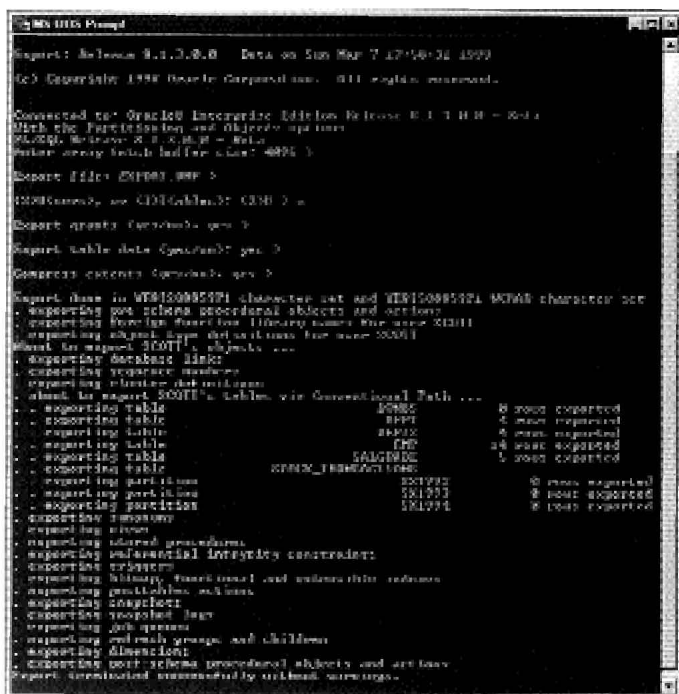


图30-9 导出多个对象与多个对象类型示例

为将多个对象与多个对象类型导入到不同的用户帐户中，导入指明FROMUSER与TOUSER子句。如果想导入到相同的用户中，使用FULL=Y或OWNER参数声明。

30.4.5 分区级导入

分区级导入将源表的一套分区或子分区导入到目标表中。

注意以下要点：

导入总存储与目标表分区模式对应的记录。分区级导入使你可以选择从导出文件的特定分区或子分区检索数据。分区级导入只插入特定的源分区或子分区的记录。如果目标表是分区的，分区级导入拒绝任何比目标表最大分区大的记录。分区级导入可以只以表的模式声明。分区级导出与导入不提供分割一个分区。

30.4.6 当表空间不匹配时识别行为

当导入一个对象时，Oracle试图在与导出相同的表空间内创建这个对象。有时，当在一个数据库上执行一个导出并导入到另一个数据库时，表空间并不总是匹配的。例如，导出数据库中的一个对象存储在DBA_TOOLS表空间中，而在将导入的数据库中不存在DBA_TOOLS表空间。

在这种情况下，在导入过程中，导入工具试图在DBA_TOOLS中创建对象，但失败了，因为在目标数据库中不存在这样的表空间。然后导入试图在该对象拥有者的缺省表空间中创建这个表。如果有足够的空间，并且拥有者具有表空间上适当的配额，对象被导入，否则，产生一个错误，不导入对象。

30.4.7 在表空间之间移动数据库对象

缺省情况下，导入试图在与导出相同的表空间中创建对象。如果用户不具有那个表空间的权限，或者那个表空间不再存在时，Oracle在用户帐户的缺省表空间中创建数据库对象。这些特性可以用于使用导出与导入在表空间之间移动数据库对象。要为 USER_A 将 TABLESPACE_A 的所有对象移动到 TABLESPACE_B，应遵循以下步骤：

- 1) 为 USER_A 导出 TABLESPACE_A 中的所有对象。
- 2) 执行 REVOKE UNLIMITED TABLESPACE ON TABLESPACE_A FROM USER_A; 以回收任何授予用户帐户的无限制表空间权限。
- 3) 执行 ALTER USER USER_A QUOTA 0 ON TABLESPACE_A; 以使 USER_A 帐户不能在 TABLESPACE_A 上创建任何对象。
- 4) 删除 TABLESPACE_A 中 USER_A 拥有的所有对象。
- 5) 执行 ALTER USER USER_A DEFAULT TABLESPACE TABLESPACE_B; 以使 TABLESPACE_B 成为 USER_A 用户帐户的缺省表空间。Oracle 试图将对象导入 TABLESPACE_A，因为这些对象是从 TABLESPACE_A 导出的。注意用户不具有 TABLESPACE_A 上的配额。然后将查看用户的缺省表空间。在 Oracle 可以将数据导入 TABLESPACE_B 之前，必须给予 USER_A 用户该表空间上足够大的配额，如以下步骤所示。
- 6) 执行 ALTER USER USER_A QUOTA UNLIMITED ON TABLESPACE_B;。通过给予 USER_A 一个无限的配额，只要 TABLESPACE_B 足够大，能处理所有被导入的数据库对象，导入就会成功。
- 7) 导入被导出的数据库对象。缺省情况下，导入工具试图将它们导入到 TABLESPACE_A 中，然而，因为用户不具有这个表空间的配额，所以所有的对象将被创建在 USER_A 的缺省表空间，TABLESPACE_B 中。

前面的步骤显示了怎样使用导入与导出工具，同时还有 SQL 的知识，以对数据进行相对容易的强有力的操作。导入与导出工具最有用的能力之一是 SHOW 与 INDEXFILE 选项的使用，这将在下一节介绍。

30.5 使用 SHOW 与 INDEXFILE 选项

可以使用 SHOW 参数生成用于创建数据库结构与所有对象的全部 SQL 语句。包括创建注释、表空间、用户、权限、授权、角色与它们的分配、限额定义、回滚段、序列、表、约束、索引、包、过程、分区、用户定义数据类型等等。

SHOW 参数的一个最有力的使用是创建可以重建全部或部分数据库的脚本文件。语句按正确的相关性顺序列出——也就是说，在创建索引之前创建表、指向主键的外键在主键创建后创建等等。程序清单 30-3 显示了指明 SHOW=Y 的输出示例的一部分。

清单 30-3 CHP30_3.lst 具有 SHOW=Y 导入的部分示例

```
"ALTER SCHEMA = "QUE" "  
"CREATE UNIQUE INDEX "I_PRICE" ON "PRICE" ("PRODUCT_ID" , "START_DATE" ) PC "  
"TFREE 10 INITRANS 2 MAXTRANS 255 STORAGE (INITIAL 10240 NEXT 10240 MINEXTEN "  
"TS 1 MAXEXTENTS 121 PCTINCREASE 50 FREELISTS 1) TABLESPACE "USER_DATA" LOGG "  
"ING "  
"ALTER TABLE "PRICE" ADD CHECK (PRODUCT_ID IS NOT NULL) ENABLE"
```



```

"ALTER TABLE "PRICE" ADD CHECK (START_DATE IS NOT NULL) ENABLE"
"ALTER TABLE "PRICE" ADD CHECK (LIST_PRICE IS NULL OR MIN_PRICE IS NULL OR "
"MIN_PRICE <= LIST_PRICE) ENABLE"
"ALTER TABLE "PRICE" ADD CHECK (END_DATE IS NULL OR START_DATE <= END_DATE)"
" ENABLE"
"ALTER TABLE "PRICE" ADD PRIMARY KEY ("PRODUCT_ID","START_DATE") ENABLE"
"GRANT SELECT ON "PRICE" TO PUBLIC"
"ALTER SCHEMA = "QUE"
"COMMENT ON TABLE "PRICE" IS 'Prices (both standard and minimum) of product"
"s. Database tracks both effective dates and expiration dates for prices.'"
"COMMENT ON COLUMN "PRICE"."PRODUCT_ID" IS 'Product number to which price a"
"pplies. Product name found in table PRICE.'"
"COMMENT ON COLUMN "PRICE"."LIST_PRICE" IS 'Undiscounted price (in U.S.dol"
"lars).'"
"ALTER TABLE "PRICE" ADD FOREIGN KEY ("PRODUCT_ID") REFERENCES "PRODUCT" ("P"
"RODUCT_ID") ENABLE"

```

为从结果产生文件，指明 LOG=文件名参数声明。可以修改这个文件以改变数据库的几乎任一部分。每一行以引号标志开始和结束。确信已在每行的开始和结束串起这些引号。另外，Oracle在输出时不会自动换行，结果是语句的句子和数字有可能被切成两半。为补救它，必须手工连接每条语句的行。前面程序清单中显示的示例程序清单将被整理如程序清单 30-4 所示。

清单30-4 CHP30_4.lst整理SHOW=Y导入文件之后的SQL语句

```

ALTER SCHEMA = "QUE";
CREATE UNIQUE INDEX "I_PRICE" ON "PRICE" ("PRODUCT_ID" , "START_DATE" )
  PCTFREE 10 INITRANS 2 MAXTRANS 255
  STORAGE (INITIAL 10240 NEXT 10240 MINEXTENTS 1 MAXEXTENTS 121
  PCTINCREASE 50 FREELISTS 1)
  TABLESPACE "USER_DATA" LOGGING;
ALTER TABLE "PRICE" ADD CHECK (PRODUCT_ID IS NOT NULL) ENABLE;
ALTER TABLE "PRICE" ADD CHECK (START_DATE IS NOT NULL) ENABLE;
ALTER TABLE "PRICE" ADD CHECK (LIST_PRICE IS NULL OR MIN_PRICE IS NULL
OR MIN_PRICE <= LIST_PRICE) ENABLE;
ALTER TABLE "PRICE" ADD CHECK (END_DATE IS NULL OR START_DATE <= END_DATE)
  ENABLE;
ALTER TABLE "PRICE" ADD PRIMARY KEY (PRODUCT_ID,START_DATE) ENABLE;
GRANT SELECT ON "PRICE" TO PUBLIC;
COMMENT ON TABLE "PRICE" IS
  'Prices (both standard and minimum) of products. Database tracks both
  effective dates and expiration dates for prices.';
COMMENT ON COLUMN "PRICE"."PRODUCT_ID" IS 'Product number to which price applies.
  Product name found in table PRICE.';
COMMENT ON COLUMN "PRICE"."LIST_PRICE" IS 'Undiscounted price (in U.S.dollars).';
ALTER TABLE "PRICE" ADD FOREIGN KEY (PRODUCT_ID) REFERENCES PRODUCT (PRODUCT_ID) ENABLE;

```

可以使用INDEXFILE参数生成CREATE INDEX语句。INDEXFILE参数的值指明要被创建的文件的名字。除非另外声明，缺省情况下，Oracle附加一个.SQL扩展名。显示常用的表创建语句，并被注释起来，以便在脚本运行时不被执行。INDEXFILE参数不生成CREATE主键或唯一键子句。程序清单 30-5是具有INDEXFILE=X.LOG声明的输出文件X.LOG的一部分。注意Oracle怎样相应地自动换行，并且不在每行之前和之后增加引号标志。这使索引文件可以不需要进一步修改就能立即使用。

清单30-5 X.LOG 具有INDEXFILE=X.LOG导入创建的X.LOG文件的部分示例

```
REM CREATE TABLE "QUE"."PRICE" ("PRODUCT_ID" NUMBER(6, 0), "LIST_PRICE"
REM NUMBER(8, 2), "MIN_PRICE" NUMBER(8, 2), "START_DATE" DATE, "END_DATE"
REM DATE) PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255 LOGGING
REM STORAGE(INITIAL 10240 NEXT 10240 MINEXTENTS 1 MAXEXTENTS 121
REM PCTINCREASE 50 FREELISTS 1 FREELIST GROUPS 1) TABLESPACE "USER_DATA" ;
REM ... 58 rows
CONNECT QUE;
CREATE UNIQUE INDEX "QUE"."I_PRICE" ON "PRICE" ("PRODUCT_ID" ,
"START_DATE" ) PCTFREE 10 INITRANS 2 MAXTRANS 255 STORAGE (INITIAL 10240
NEXT 10240 MINEXTENTS 1 MAXEXTENTS 121 PCTINCREASE 50 FREELISTS 1)
TABLESPACE "USER_DATA" LOGGING ;
```

30.6 便携式表空间

这个功能使用户可以将 Oracle 数据库的一个子集移动到另一个 Oracle 数据库中。子集看起来像是从初始数据库中拔出来并插到另一个数据库中。也可以克隆一个数据库中的一个表空间并将其插入另一个数据库中，从而，在数据库之间拷贝表空间。使用便携式表空间功能移动数据潜在地比使用导出/导入或卸载/装载要快许多，因为便携式表空间只包括拷贝数据库文件与集成的元数据。在数据仓库环境中，大量数据从最初的 OLTP 数据库流入企业数据仓库并流向数据商业中心，这个功能显示了数据移动的一个更快与更创新的方法。另外，它也可以扩展用于归档数据。

第31章 SQL*Loader

本章要点：

- 运行SQL*Loader
- SQL*Loader组件
- SQL*Loader示例一览
- 常规路径装载与直接路径装载

31.1 运行SQL*Loader

今天，数据库在复杂程度和大小上都不断增加，G字节大小的数据库已经非常普通，并且数据仓库通常达到上T字节的存储容量。随着这些数据库的增长，使用外部数据快速有效地填充数据库的需求变得非常重要。为迎接这个挑战，Oracle提供了一个称为SQL*Loader的工具，它可以将数据从外部数据文件装载进一个Oracle数据库。

SQL*Loader具有许多功能，包括以下能力：

- 可以从不同文件类型的多个输入数据文件中装载数据。
- 输入记录可以是定长的或变长的记录。
- 可以在同一次运行中加载多个表，还可以逻辑地将选定的记录载入到每个不同的表中。
- 在输入数据载入表之前，可以对其使用SQL函数。
- 多个物理记录可以被编译成一个逻辑记录，同样，SQL可以提取一条物理记录并把它作为多个逻辑记录加载。
- 支持嵌套列、嵌套表、VARRAYS和LOBs（巨型对象，包括BLOB、CLOB、NCLOB和BFILES）。

可以通过在命令行输入sqlload、sqlldr或sqlldr80激活SQL*Loader。确切的命令根据操作系统（OS）的不同可能会有所区别。参阅Oracle操作系统说明手册获取精确的语法使用说明。请注意，本章中的全部程序清单和服务响应可能会不同于你的结果，这完全取决于正在使用的操作系统。Sqlldr命令接受数个命令行参数。不使用任何参数激活SQL*Loader，将会显示全部可用合法参数的帮助信息（参见清单31-1）。

清单31-1 SQL*Loader帮助信息

```
Invoking SQL*Loader without parameters:
$ sqlldr

The server responds with help information because SQL*Loader was invoked without
parameters:
SQL*Loader: Release 8.1.5.0.0 - Production on Wed Mar 10 7:20:11 1999
(c) Copyright 1999 Oracle Corporation. All rights reserved.

Usage: SQLLOAD keyword=value [,keyword=value,...]
```

Valid Keywords:

```
userid -- ORACLE username/password
control -- Control file name
log -- Log file name
bad -- Bad file name
data -- Data file name
discard -- Discard file name
discardmax -- Number of discards to allow (Default all)
skip -- Number of logical records to skip (Default 0)
load -- Number of logical records to load (Default all)
errors -- Number of errors to allow (Default 50)
rows -- Number of rows in conventional path bind array or between
       direct path data saves
       (Default: Conventional path 64, Direct path all)
bindsize -- Size of conventional path bind array in bytes (Default
65536)
silent -- Suppress messages during run
         (header, feedback, errors, discards, partitions)
direct -- use direct path (Default FALSE)
parfile -- parameter file: name of file that contains parameter
         specifications
parallel -- do parallel load (Default FALSE)
readsize -- Size (in bytes) of the read buffer
file -- File to allocate extents from
skip_unusable_indexes -- disallow/allow unusable indexes or index
partitions (Default FALSE)
skip_index_maintenance -- do not maintain indexes, mark affected indexes
as unusable (Default FALSE)
commit_discontinued -- commit loaded rows when load is discontinued
(Default FALSE)
```

PLEASE NOTE: Command-line parameters may be specified either by position or by keywords. An example of the former case is 'sqlload scott/tiger foo'; an example of the latter is 'sqlload control=foo userid=scott/tiger'. One may specify parameters by position before but not after parameters specified by keywords. For example,

'sqlload scott/tiger control=foo logfile=log' is allowed, but 'sqlload scott/tiger control=foo log' is not, even though the position of the parameter 'log' is correct.

31.2 SQL*Loader组件

SQL*Loader是一个Oracle工具，能够将数据从外部数据文件装载到数据库中。图 31-1显示了SQL*Loader的不同组件。

控制文件

控制文件是SQL*Loader的中枢核心，控制文件能够控制外部数据文件中的数据如何映射到Oracle的表和列。注意，SQL*Loader输入数据类型完全独立于它们将被载入的数据库列的数据类型。如果需要，将进行隐式的数据类型转换，如果转换失败，将捕捉错误消息。在控制文件中使用的语言是SQL*Loader数据定义语言（Data Definition Language, DDL）。控制文件由带有许多参数的多个部分组成，在本章中无法深入地涉及到每个部分。参阅《Oracle服务器工具手册》（Oracle Server Utilities manual）了解控制文件全部合法语法及参数的有关文档。基本的语法包含在31.3节“SQL*Loader示例一览”部分，用来说明SQL*Loader的各种不同应用。

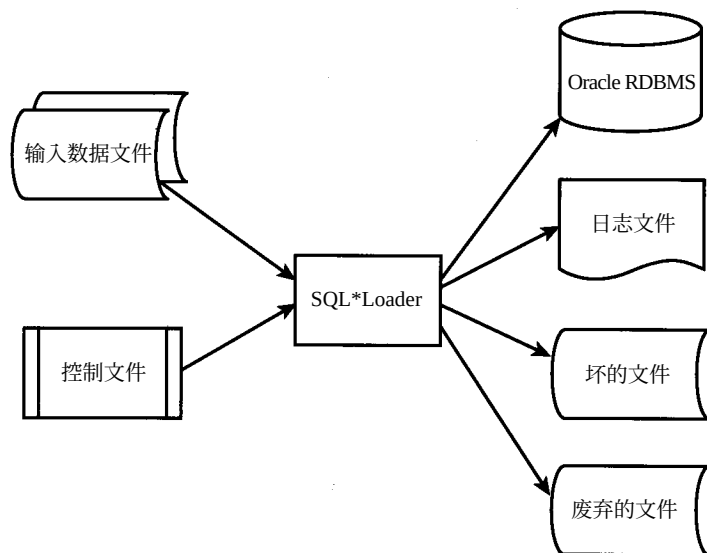


图31-1 SQL*Loader组件

31.2.1 SQL*Loader输入数据

SQL*Loader能够接收多种不同格式的数据文件。文件可以存储在磁盘或磁带上，或记录本身可以被嵌套到控制文件中。记录格式可以是定长的或变长的，定长记录是指这样的记录：每条记录具有相同的固定长度，并且每条记录中的数据域也具有相同的固定长度、数据类型和位置。例如，一条记录中的 PART_NBR数据项总是占据第10到19列，而不管数据的实际长度。如果部件号为12345，在15列至19列中剩余的部分将为空白。对于变长记录，数据项只占据每条记录实际需要的空间。

在PART_NBR例子中，数据项只需要5个字节，没有尾随的空格。在变长格式文件中的每条记录可以拥有PART_NBR项不同的空间，这完全取决于该数据项的实际长度。虽然变长记录可能在数据文件中使用较少的空间，但是每条记录上的数据项必须有一个间隔符以分隔每个数据项，意识到这点非常重要。

31.2.2 SQL*Loader输出

下面几节讨论由一个SQL*Loader会话创建的各种文件。

1. Oracle表和索引

SQL*Loader可以在同一个装载会话中向Oracle数据库装载多个表和索引，SQL*Loader插入数据和创建索引的行为将在31.4.1节“使用常规路径装载”与31.4.2节“使用直接路径装载”部分讨论。

2. 坏文件

SQL*Loader验证插入数据库中的数据要经历两阶段过程。第一阶段是按照控制文件中的说明验证数据的格式。如果数据的格式或长度与说明不一致，SQL*Loader将该记录写入坏文件。当记录通过第一阶段的验证后，它们被传递给数据库用于插入。

第二阶段的验证工作发生在数据库内部，数据库出于各种原因可能会拒绝记录，某些原

因可能是数据库检查约束错误和数据类型转换错误。第二阶段验证拒绝也被写入坏文件。如果拒绝的数目达到一定阈值（默认值为 50），SQL*Loader会话将终止退出，这个阈值可以在命令行上使用 errors 参数设置，采用与初始输入数据文件相同的格式写入坏文件，这样一来，在做出了必要的改正后，可以使用同一个控制文件装载坏文件记录。

3. 废弃文件

如果控制文件中含有条件并且记录不能满足全部的条件，SQL*Loader将这些记录写入废弃文件。例如，控制文件中的一项条件说明在列 1 必须拥有一个 X，没有 X 的记录将不会被插入数据库，而且要将它们写入废弃文件。与坏文件不同，废弃记录的缺省阈值为允许全部的废弃记录，可以在命令行使用 discardmax 参数将该阈值调低。

4. 日志文件

在SQL*Loader执行时，它创建一个日志文件。妨碍该日志文件成功地创建的任意情形都将终止装载会话。日志文件的缺省名称为带有 .log 扩展名的控制文件名。如果在命令行没有使用 log 参数给出一个新的文件名，装载会话自动地使用相同名称重写上一个日志文件，记住这一点很重要。日志文件含有多个部分，使用结果和小结统计信息显示装载会话运行的环境。在31.3节“SQL*Loader示例一览”部分中的程序清单 31-8显示了一个日志文件中的内容。

31.2.3 控制文件语法

大部分控制文件使用下列的关键字开头：

LOAD DATA

可能在它们之前出现的其他关键字是 ‘ - ’，它是注释，是可选的，它使得先前讨论的命令行选项能够被包含进控制文件。

后面紧跟着定义，以指明用于装载的源外部数据文件：

INFILE 'mydata.dat'

通过指定多个 INFILE 语句，可以在同一个会话中装载多个数据文件：

INFILE 'mydata1.dat'

INFILE 'mydata2.dat'

如果没有指定文件的扩展名，SQL*Loader默认文件的扩展名为 .dat。虽然没有要求使用单引号将数据文件括起来，但当指定完全数据路径时，我们强烈建议使用单引号将数据文件括起来，以避免错误的特殊字符转换。在装载会话中所有表使用的装载方法（参见表 31-1）也被指定。

表31-1 表装载方法

方 法	说 明
INSERT	这是缺省方法。该方法假设在数据装载前表是空的。如果在表中仍然有记录，SQL*Loader退出
APPEND 行	这种方法允许记录被添加到数据库表中，而且不影响已经存在的记录
REPLACE	这种方法首先删除表中已经存在的记录，然后开始装载新的记录。注意，当老记录被删除时，表上的任意删除触发器将被触发
TRUNCATE	这种方法在装载数据前，使用 SQL 命令 TRUNCATE 删除老的记录，因为去除了触发器的触发并且没有创建回滚，所以这种方法要比 REPLACE 快得多。TRUNCATE 不是一个可恢复命令。为了使用这种方法，表引用完整性约束必须被禁止，并且要授予特定的权限

后面跟着表定义：

INTO TABLE 表名称 方法

在这个例子中，方法与上面表 31-1 中说明的相同，但是这种方法只适用于在 INTO TABLE 行上指定的表。当从同一个 SQL*Loader 会话中装载多个表时，这给了用户较好的控制。

跟在 INTO TABLE 关键字后面的是列与数据类型说明。不需要回顾全部不同的选项，你可以较为容易地浏览本章后面部分中的不同例子。

31.3 SQL*Loader 示例一览

所有的示例使用下面四个表（清单 31-2）组成的模式，这个模式使用客户、帐户和事务表模拟了一个银行模式。为了表明向一个分区表装载数据，partition_xact 表是事务表的副本，具有基于事务发生时刻的数据分区。

清单 31-2 LIST1.1 —— 模式示例

```
create table customer (
    cust_nbr      number(7)      not null,
    cust_name     varchar2(100)   not null,
    cust_addr1    varchar2(50),
    cust_addr2    varchar2(50),
    cust_city     varchar2(30),
    cust_state    varchar2(2),
    cust_zip      varchar2(10),
    cust_phone    varchar2(20),
    cust_birthday date)
/

create table account (
    cust_nbr      number(7)      not null,
    acct_nbr      number(10)     not null,
    acct_name     varchar2(40)   not null)
/

create table transaction (
    acct_nbr      number(10)     not null,
    xact_amt      number(10,2)   not null,
    xact_flag     char           not null,
    xact_date     date           not null)
/

create table partition_xact (
    acct_nbr      number(10)     not null,
    xact_amt      number(10,2)   not null,
    xact_flag     char           not null,
    xact_date     date           not null)

PARTITION BY RANGE (xact_date)
(PARTITION P1 VALUES LESS THAN (to_date('01-APR-1999','DD-MON-YYYY')),
PARTITION P2 VALUES LESS THAN (to_date('01-JUL-1999','DD-MON-YYYY')),
PARTITION P3 VALUES LESS THAN (to_date('01-OCT-1999','DD-MON-YYYY')),
PARTITION P4 VALUES LESS THAN (MAXVALUE))
/
```

所有的示例使用下面的数据文件（参见清单 31-3、清单31-4与清单31-5）。

清单31-3 cust.dat —— 清单说明

0000001BOB MARIN	123 MAIN ST.	TOPEKA	KS12345
999-555-1234	20-APR-55		
0000002MARY JOHNSON	18 HOPE LANE	SAN FRANCISCO	CA94054
415-555-1299	32-JAN-69		
0000003RICHARD WILLIAMS	1225 DAFFODIL LANE	BOSTON	MA98377
0000004WALTER SIMS	1888 PROSPECT AVE.	BROOKLYN	NY11218
718-555-3420			
0000005LARRY HATFIELD	TWO FIELDS CT.	SOMERSET	NJ07689
732-555-2454	25-DEC-60		
0000006LAURA LAU	25 CHRISTOPHER LN	SAN BRUNO	CA90234
510-555-4834			
0000123PRISCILLA WONG	888 FORTUNE COURT	PHILADELPHIA	PA35545
01-JAN-65			
0000068SONNY BALRUP	27 KAMA ST.	JACKSON HEIGHTS	NY10199
718-555-9876	07-MAY-61		
0023494RUPAL PARIKH	2 FORCE BLVD	NEW YORK	NY10105
212-555-5887	31-DEC-72		
0000324CRAIG SILVEIRA	1674 ISLAND ST	SMITHTOWN	NY12467
516-555-5534	27-OCT-74		
0000010DANIEL SMITH	35 DIRECT DRIVE	BERGEN	NJ07899
201-555-3734			
0011102STEPHEN LEUNG	16 STANFORD CT	STANFORD	CA96688
650-555-1248	05-SEP-76		
0011102ALICIA LOWRY	5678 TIMOTHY DR	ATLANTA	GA47730
0002340JENNIFER LEUNG	1 MURRAY HILL	GREENWICH	CT78835
203-555-7564			
1003423HORACE MICHAEL	90 MINISTER ST	MINNEAPOLIS	MN77788
18-MAR-65			
0000223CHRISTOPHER YEE	9077 MUSIC AVE	DETROIT	MI45345
777-555-7785	22-JUL-75		
0009032JAMES BORIOTTI	65 FIREMENS LANE	COLUMBUS	OH37485
904-555-5674			
0000088HIREN PATEL	69 CLUB ST.	NEW YORK	NY12445
212-555-7822	12-APR-70		
0000100RICHARD JI	1225 STEER ST	KOBE	KS12009
999-555-5824	10-OCT-74		
0000046DAVID CHOW	49 HUGO DRIVE	FLUSHING	NY10199
718-555-4367			
0000758HENRY WALKER	12 SIGMUND ST.	CHICAGO	IL33890
312-555-5567	09-APR-45		
0002993GEORGE BLOOM	28 BRIDGEWATER ST	SAN MATEO	CA90475
650-555-2838	25-MAY-63		
0009488LISA JONES	30 MISSION ST	UNITY	FL23899

清单31-4 acct.dat —— 清单说明

0000001,459023,SAVINGS
0000001,459024,CHECKING
0000003,211108,SAVINGS
0000003,211123,CHECKING
0000006,23388,SAVINGS
0000123,43992,CHECKING
0000123,50699390,LINE OF CREDIT
0000068,23330,SAVINGS
0023494,433020,SAVINGS
0000010,4566,SAVINGS
0000010,4599,CHECKING
0000223,8887544,SAVINGS

清单31-5 xact.dat —— 清单说明

0000459023	123.45D01-FEB-99
0000459023	1233.86C01-MAR-99
0000459023	987.00P01-DEC-99
0000459024	1000C03-JUN-99
0000211108	875.27D23-JUL-99
0000211123	20987.88C30-DEC-99
0000211123	12500.16D10-JAN-99
0000023388	1.75C19-MAY-99
0000043992	350.00C12-MAR-99
0050699390	2899.09D01-SEP-99
0000023330	100D26-JAN-99
0000433020	60.99C20-NOV-99
0000004566	230.23C20-AUG-99
0000004599	14.96D05-JUN-99
0000004599	14.AAD07-JUN-99
0008887544	9999.99D11-JUL-99

31.3.1 例子1：装载定长数据

这个示例将 cust.dat 数据文件中的数据装载到客户表，因为数据是定长格式，所以控制文件（参见清单31-6）按列的位置将数据映射到数据库。

清单31-6 load1.ctl —— 控制文件

```

LOAD DATA
INFILE 'cust.dat'
INTO TABLE customer
(cust_nbr    POSITION(01:07)    INTEGER EXTERNAL,
 cust_name   POSITION(08:27)    CHAR,
 cust_addr1  POSITION(28:47)    CHAR,
 cust_city   POSITION(48:67)    CHAR,
 cust_state  POSITION(68:69)    CHAR,
 cust_zip    POSITION(70:79)    CHAR,
 cust_phone  POSITION(80:91)    CHAR,
 cust_birthdate POSITION(100:108) DATE "DD-MON-YY" NULLIF cust_birthdate=BLANKS)

```

使用 example/express 作为用户名和口令激活 SQL*Loader，正如前面在“SQL*Loader 输出”部分讨论的，控制文件、日志文件、坏文件和废弃文件作为命令行参数传递给 SQL*Loader。（见清单31-7、清单31-8）

清单31-7 load1.ctl —— 激活 SQL*Loader

```

$ sqlldr example/express control=load1.ctl log=load1.log bad=load1.bad
discards=load1.dis

```

下面是服务器的响应信息：

```

SQL*Loader: Release 8.0.3.0.0 - Production on Wed Mar 10 8:10:23 1999
(c) Copyright 1997 Oracle Corporation. All rights reserved.

Commit point reached - logical record count 23

```

清单31-8 load1.ctl —— 例子1日志文件内容

```

SQL*Loader: Release 8.0.3.0.0 - Production on Wed Mar 10 8:11:03 1999

(c) Copyright 1997 Oracle Corporation. All rights reserved.

```

```
Control File:  load1.ctl
Data File:     cust.dat
Bad File:      load1.bad
Discard File:  load1.dis
(Allow all discards)
```

```
Number to load: ALL
Number to skip: 0
Errors allowed: 50
Bind array:     64 rows, maximum of 65536 bytes
Continuation:   none specified
Path used:      Conventional
```

Table CUSTOMER, loaded from every logical record.
Insert option in effect for this table: INSERT

Column Name	Position	Len	Term	Encl	Datatype
CUST_NBR	1:7	7			
CHARACTER					
CUST_NAME	8:27	20			CHARACTER
CUST_ADDR1	28:47	20			CHARACTER
CUST_CITY	48:67	20			CHARACTER
CUST_STATE	68:69	2			CHARACTER
CUST_ZIP	70:79	10			CHARACTER
CUST_PHONE	80:91	12			CHARACTER
CUST_BIRTHDAY	100:108	9			DATE DD-MON-YY

Column CUST_NAME is NULL if CUST_NAME = BLANKS
 Column CUST_ADDR1 is NULL if CUST_ADDR1 = BLANKS
 Column CUST_CITY is NULL if CUST_CITY = BLANKS
 Column CUST_STATE is NULL if CUST_STATE = BLANKS
 Column CUST_ZIP is NULL if CUST_ZIP = BLANKS
 Column CUST_PHONE is NULL if CUST_PHONE = BLANKS
 Column CUST_BIRTHDAY is NULL if CUST_BIRTHDAY = BLANKS

Record 2: Rejected - Error on table CUSTOMER, column CUST_BIRTHDAY.
 ORA-01847: day of month must be between 1 and last day of month

Table CUSTOMER:

```
22 Rows successfully loaded.
1 Row not loaded due to data errors.
0 Rows not loaded because all WHEN clauses were failed.
0 Rows not loaded because all fields were null.
Space allocated for bind array:          9216 bytes(64 rows)
Space allocated for memory besides bind array: 0 bytes
```

```
Total logical records skipped: 0
Total logical records read:    23
Total logical records rejected: 1
Total logical records discarded: 0
```

```
Run began on Wed Mar 10 8:11:03 1999
Run ended on Wed Mar 10 8:11:03 1999
```

```
Elapsed time was:    00:00:00.21
CPU time was:       00:00:00.04
```

例子1注释：由于非法的日期，记录2被拒绝并写入坏文件（load1.bad）。这条记录可以在坏文件中被改正并通过下列方法装载到数据库：对同一个控制文件做出更改，使用坏文件作为输入文件并在关键字 INTO TABLE前添加关键字 APPEND。还有，请注意在控制文件中

NULLIF子句的使用，没有这个子句，日期中含有空格的记录将不能进行数据库日期检查。

31.3.2 例子2：装载变长数据

这个示例将数据文件 acct.dat中的数据装载到客户表。因为日期采用变长格式，控制文件定义了分隔符用于区分不同的数据项（见清单 31-9、清单31-10）。

清单31-9 LOAD2.CTL — 例子2控制文件

```
LOAD DATA

INFILE 'acct.dat'

INTO TABLE account

FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'

(cust_nbr, acct_nbr, acct_name)
```

为例子2在命令行激活SQL*Loader的方法如下所示：

```
$ sqlldr example/epass control=load2.ctl log=load2.log bad=load2.bad
discard=load2.dis
```

下面是服务器的响应信息：

```
SQL*Loader: Release 8.0.3.0.0 - Production on Wed Mar 10 8:15:51 1999
(c) Copyright 1997 Oracle Corporation. All rights reserved.
Commit point reached - logical record count 12
```

清单31-10 例子2日志文件内容

```
SQL*Loader: Release 8.0.3.0.0 - Production on Wed Mar 10 8:16:03 1999
(c) Copyright 1997 Oracle Corporation. All rights reserved.

Control File:   load2.ctl
Data File:      acct.dat
Bad File:       load2.bad
Discard File:   load2.dis
(Allow all discards)

Number to load: ALL
Number to skip: 0
Errors allowed: 50
Bind array:     64 rows, maximum of 65536 bytes
Continuation:   none specified
Path used:      Conventional

Table ACCOUNT, loaded from every logical record.
Insert option in effect for this table: INSERT
```

Column Name	Position	Len	Term	Encl	Datatype
CUST_NBR	FIRST	*	,	O(")	CHARACTER
ACCT_NBR	NEXT	*	,	O(")	CHARACTER
ACCT_NAME	NEXT	*	,	O(")	CHARACTER

Table ACCOUNT:

```

12 Rows successfully loaded.
0 Rows not loaded due to data errors.
0 Rows not loaded because all WHEN clauses were failed.
0 Rows not loaded because all fields were null.

Space allocated for bind array:          35328 bytes(64 rows)
Space allocated for memory besides bind array:    0 bytes

Total logical records skipped:          0
Total logical records read:             12
Total logical records rejected:         0
Total logical records discarded:        0

Run began on Wed Mar 10 8:16:03 1999
Run ended on Wed Mar 10 8:16:28 1999

Elapsed time was:      00:00:00.23
CPU time was:         00:00:00.04

```

例子2注释：全部记录都被成功地装载。记住，变长数据必须含有分隔符以分隔输入数据文件内部的数据项。当装载分隔的字符数据时，定义每个字符数据域的最大长度更有效率。在这个例子中，应该在控制文件中使用 `acct_name char(20)`。如果没有定义最大长度，SQL*Loader使用255字节作为长度的缺省值，这会影响在每个赋值数组执行中有多少记录被插入。

31.3.3 例子3：装载嵌套数据

这个示例表明数据不一定必须在一个数据文件中，数据可以直接安置在控制文件中。BEGINDATA关键字指示在它之后的全部信息行都是数据记录，可以被用作该控制文件的输入源（见清单31-11、清单31-12）。

清单31-11 LOAD3.CTL — 例子3控制文件

```

LOAD DATA
INFILE *
APPEND
INTO TABLE account
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
(cust_nbr, acct_nbr, acct_name)

BEGINDATA

0000324,89073,SAVINGS
0000324,89074,CHECKING
0000075,111,SAVINGS
0011102,800,CHECKING
0000068,23338,CHECKING

```

从命令行为例子3激活SQL*Loader，如下所示：

```
$ sqlldr example/epass control=load3.ctl log=load3.log bad=load3.bad
discard=load3.dis
```

下面是服务器的响应信息：

```

SQL*Loader: Release 8.0.3.0.0 - Production on Wed Mar 10 8:20:11 1999

(c) Copyright 1997 Oracle Corporation. All rights reserved.

```

Commit point reached - logical record count 5

清单31-12 例子3日志文件内容

SQL*Loader: Release 8.0.3.0.0 - Production on Wed Mar 10 8:21:21 1999

(c) Copyright 1997 Oracle Corporation. All rights reserved.

Control File: load3.ctl
Data File: load3.ctl
Bad File: load3.bad
Discard File: load3.dis
(Allow all discards)

Number to load: ALL
Number to skip: 0
Errors allowed: 50
Bind array: 64 rows, maximum of 65536 bytes
Continuation: none specified
Path used: Conventional
Table ACCOUNT, loaded from every logical record.
Insert option in effect for this table: APPEND

Column Name	Position	Len	Term	Encl	Datatype
CUST_NBR	FIRST	*	,	0(")	CHARACTER
ACCT_NBR	NEXT	*	,	0(")	CHARACTER
ACCT_NAME	NEXT	*	,	0(")	CHARACTER

Table ACCOUNT:

5 Rows successfully loaded.
0 Rows not loaded due to data errors.
0 Rows not loaded because all WHEN clauses were failed.
0 Rows not loaded because all fields were null.

Space allocated for bind array: 35328 bytes(64 rows)
Space allocated for memory besides bind array: 0 bytes

Total logical records skipped: 0
Total logical records read: 5
Total logical records rejected: 0
Total logical records discarded: 0

Run began on Wed Mar 10 8:20:11 1999
Run ended on Wed Mar 10 8:20:35 1999

Elapsed time was: 00:00:00.24
CPU time was: 00:00:00.03

例子3注释：要装载放置在控制文件中的数据，在关键字 INFILE后面使用*代替一个输入文件名。这个例子还显示了 APPEND装载方法的使用，使用该方法添加帐户表。如果在控制文件中省略 APPEND关键字，因为 SQL*Loader缺省的装载方法是 INSERT，这要求在装载前表必须为空，所以装载过程终止退出。

31.3.4 例子4：条件装载

这个示例从文件 xact.dat装载数据。控制文件检查标志，查看总额是借方还是贷方，并据此加载数据（见清单31-13、清单31-14）。

清单31-13 例子4控制文件

LOAD DATA

```

INFILE 'xact.dat'
INTO TABLE transaction
WHEN xact_flag = 'D'

```

```

(acct_nbr    POSITION(01:10)    INTEGER EXTERNAL,
 xact_amt    POSITION(11:20)    INTEGER EXTERNAL ":xact_amt * -1",
 xact_flag    POSITION(21:21)    CHAR,
 xact_date    POSITION(22:31)    DATE "DD-MON-YY" NULLIF xact_date=BLANKS)

```

```

INTO TABLE transaction
WHEN xact_flag = 'C'

```

```

(acct_nbr    POSITION(01:10)    INTEGER EXTERNAL,
 xact_amt    POSITION(11:20)    INTEGER EXTERNAL,
 xact_flag    POSITION(21:21)    CHAR,
 xact_date    POSITION(22:31)    DATE "DD-MON-YY" NULLIF xact_date=BLANKS)

```

从命令行为例子4激活SQL*Loader, 如下所示:

```

$ sqlldr example/expass control=load4.ctl log=load4.log bad=load4.bad
discards=load4.dis

```

下面是服务器的响应信息:

```

SQL*Loader: Release 8.0.3.0.0 - Production on Wed Mar 10 9:20:11 1999
(c) Copyright 1997 Oracle Corporation. All rights reserved.
Commit point reached - logical record count 16

```

清单31-14 例子4日志文件内容

```

SQL*Loader: Release 8.0.3.0.0 - Production on Wed Mar 10 9:20:11 1999
(c) Copyright 1997 Oracle Corporation. All rights reserved.

```

```

Control File:  load4.ctl
Data File:     xact.dat
Bad File:      load4.bad
Discard File:  load4.dis
(Allow all discards)

```

```

Number to load: ALL
Number to skip: 0
Errors allowed: 50
Bind array:    64 rows, maximum of 65536 bytes
Continuation:  none specified
Path used:     Conventional

```

```

Table TRANSACTION, loaded when XACT_FLAG = 0X44(character 'D')
Insert option in effect for this table: INSERT

```

Column Name	Position	Len	Term Encl	Datatype
ACCT_NBR	1:10	10		CHARACTER
XACT_AMT	11:20	10		CHARACTER
XACT_FLAG	21:21	1		CHARACTER
XACT_DATE	22:31	10		DATE DD-MON-YY

Column XACT_AMT had SQL string


```

":xact_amt * -1"
  applied to it.
Column XACT_DATE is NULL if XACT_DATE = BLANKS
Table TRANSACTION, loaded when XACT_FLAG = 0X43(character 'C')
Insert option in effect for this table: INSERT

```

Column Name	Position	Len	Term	Encl	Datatype
ACCT_NBR	1:10	10			CHARACTER
XACT_AMT	11:20	10			CHARACTER
XACT_FLAG	21:21	1			CHARACTER
XACT_DATE	22:31	10			DATE DD-MON-YY

Column XACT_DATE is NULL if XACT_DATE = BLANKS

Record 3: Discarded - failed all WHEN clauses.

Record 15: Rejected - Error on table TRANSACTION, column XACT_AMT.

ORA-01722: invalid number

Table TRANSACTION:

```

7 Rows successfully loaded.
1 Row not loaded due to data errors.
8 Rows not loaded because all WHEN clauses were failed.
0 Rows not loaded because all fields were null.

```

Table TRANSACTION:

```

7 Rows successfully loaded.
0 Rows not loaded due to data errors.
9 Rows not loaded because all WHEN clauses were failed.
0 Rows not loaded because all fields were null.

```

```

Space allocated for bind array:          7168 bytes(64 rows)
Space allocated for memory besides bind array: 0 bytes

```

```

Total logical records skipped:          0
Total logical records read:             16
Total logical records rejected:          1
Total logical records discarded:         1

```

Run began on Wed Mar 10 9:20:11 1999

Run ended on Wed Mar 10 9:20:46 1999

```

Elapsed time was:      00:00:00.35
CPU time was:          00:00:00.02

```

例子4注释：这个例子显示了如何在控制文件中使用 WHEN子句进行条件装载。在这个例子中，多个 WHEN子句装载同一个表，但是它们可以是不同的表。当事务标志等于 D时，在插入表前，美元数量被乘以 -1。这是一个说明当数据被装载时，能够对数据使用运算符或 SQL函数的例子。

因为标志既不是 C也不是 D，不满足任何一个条件，所以记录 3被废弃并被放入废弃文件中。因为美元数目不是数字型数据，所以记录 15被拒绝。这两条记录显示了前面提到的两阶段检查。记录3被装载过程废弃，而记录 15直到数据库试图将它插入到表中时才被拒绝。

31.3.5 例子5：装载到分区表

这个示例是例子4的一个变种，要从文件xact.dat加载数据。在此要将数据装载到partition_xact

表，该表有四个分区，按照日历季度存储数据，表的一个分区或全部分区可以在一个装载会话中装载。在本例中，只为第一个季度装载了分区p1（见清单31-15、清单31-16）。

清单31-15 LOAD5.CTL — 例子5控制文件

LOAD DATA

```

INFILE 'xact.dat'
INTO TABLE partition_xact PARTITION (P1)
WHEN xact_flag = 'D'

  (acct_nbr    POSITION(01:10)    INTEGER EXTERNAL,
   xact_amt    POSITION(11:20)    INTEGER EXTERNAL ":xact_amt * -1",
   xact_flag   POSITION(21:21)    CHAR,
   xact_date   POSITION(22:31)    DATE "DD-MON-YY" NULLIF xact_date=BLANKS)

INTO TABLE partition_xact PARTITION (P1)
WHEN xact_flag = 'C'
  (acct_nbr    POSITION(01:10)    INTEGER EXTERNAL,
   xact_amt    POSITION(11:20)    INTEGER EXTERNAL,
   xact_flag   POSITION(21:21)    CHAR,
   xact_date   POSITION(22:31)    DATE "DD-MON-YY" NULLIF xact_date=BLANKS)

```

从命令行为例子5激活SQL*Loader的命令如下所示：

```
$ sqlldr example/expass control=load5.ctl log=load5.log bad=load5.bad
discards=load5.dis
```

下面是服务器的响应信息：

```

SQL*Loader: Release 8.0.3.0.0 - Production on Wed Mar 10 10:20:11 1999
(c) Copyright 1997 Oracle Corporation. All rights reserved.
Commit point reached - logical record count 16

```

清单31-16 例子5日志文件内容

```

SQL*Loader: Release 8.0.3.0.0 - Production on Wed Mar 10 10:20:11 1999
(c) Copyright 1997 Oracle Corporation. All rights reserved.

Control File:   load5.ctl
Data File:      xact.dat
Bad File:       load5.bad
Discard File:   load5.dis
(Allow all discards)

Number to load: ALL
Number to skip: 0
Errors allowed: 50
Bind array:     64 rows, maximum of 65536 bytes
Continuation:   none specified
Path used:      Conventional

Table PARTITION_XACT, partition P1, loaded when XACT_FLAG = 0X44(character 'D')
Insert option in effect for this partition: INSERT

```

Column Name	Position	Len	Term	Encl	Datatype
ACCT_NBR	1:10	10			CHARACTER
XACT_AMT	11:20	10			CHARACTER
XACT_FLAG	21:21	1			CHARACTER

XACT_DATE 22:31 10 DATE DD-MON-YY

Column XACT_AMT had SQL string

":xact_amt * -1"

applied to it.

Column XACT_DATE is NULL if XACT_DATE = BLANKS

Table PARTITION_XACT, partition P1, loaded when XACT_FLAG = 0X43(character 'C')

Insert option in effect for this partition: INSERT

Column Name	Position	Len	Term Encl	Datatype
ACCT_NBR	1:10	10		CHARACTER
XACT_AMT	11:20	10		CHARACTER
XACT_FLAG	21:21	1		CHARACTER
XACT_DATE	22:31	10		DATE DD-MON-YY

Column XACT_DATE is NULL if XACT_DATE = BLANKS

Record 3: Discarded - failed all WHEN clauses.

Record 5: Rejected - Error on table PARTITION_XACT, partition P1.

ORA-14401: inserted partition key is outside specified partition

Record 10: Rejected - Error on table PARTITION_XACT, partition P1.

ORA-14401: inserted partition key is outside specified partition

Record 14: Rejected - Error on table PARTITION_XACT, partition P1.

ORA-14401: inserted partition key is outside specified partition

Record 15: Rejected - Error on table PARTITION_XACT, column XACT_AMT.

ORA-01722: invalid number

Record 16: Rejected - Error on table PARTITION_XACT, partition P1.

ORA-14401: inserted partition key is outside specified partition

Record 4: Rejected - Error on table PARTITION_XACT, partition P1.

ORA-14401: inserted partition key is outside specified partition

Record 6: Rejected - Error on table PARTITION_XACT, partition P1.

ORA-14401: inserted partition key is outside specified partition

Record 8: Rejected - Error on table PARTITION_XACT, partition P1.

ORA-14401: inserted partition key is outside specified partition

Record 12: Rejected - Error on table PARTITION_XACT, partition P1.

ORA-14401: inserted partition key is outside specified partition

Record 13: Rejected - Error on table PARTITION_XACT, partition P1.

ORA-14401: inserted partition key is outside specified partition

Table PARTITION_XACT, partition P1:

3 Rows successfully loaded.

5 Rows not loaded due to data errors.

8 Rows not loaded because all WHEN clauses were failed.

0 Rows not loaded because all fields were null.

Table PARTITION_XACT, partition P1:

2 Rows successfully loaded.

5 Rows not loaded due to data errors.

9 Rows not loaded because all WHEN clauses were failed.

0 Rows not loaded because all fields were null.

Space allocated for bind array:

7168 bytes(64 rows)

Space allocated for memory besides bind array:

0 bytes

Total logical records skipped: 0

Total logical records read: 16

Total logical records rejected: 10

Total logical records discarded: 1

Run began on Wed Mar 10 10:21:39 1999

Run ended on Wed Mar 10 8:22:01 1999

Elapsed time was: 00:00:00.31

CPU time was: 00:00:00.04

例子5注释：要装载分区表的一个分区，需要在控制文件中这个表名的后面添加关键字 PARTITION。请注意分区名称（在本例中为 P1），必须被圆括号括起来。如果省略关键字 PARTITION，全分区都将被装载。没有在 P1 的分区范围中的全部记录都被写入坏文件中。

31.4 常规路径装载与直接路径装载

SQL*Loader提供了两种方法装载数据——常规路径装载与直接路径装载。常规路径装载是SQL*Loader的默认装载方法。为了能够进行直接路径装载，在激活 SQL*Loader时，必须将 DIRECT=TRUE 添加到命令行参数中。正如在图 31-2 中所看到的，常规路径装载具有直接路径装载不需要的一些额外步骤。这些额外步骤增加了系统处理的额外开销，使得常规路径装载的速度慢于直接路径装载。格式化 SQL INSERT 语句以及搜索 SGA 内存高速缓冲区的额外步骤与其他同时并发运行在数据库上的进程产生竞争。虽然出于速度原因，我们倾向于使用直接路径装载，但是当使用常规路径装载时，还有一些限制和情况。这将在下面的部分中涉及到。

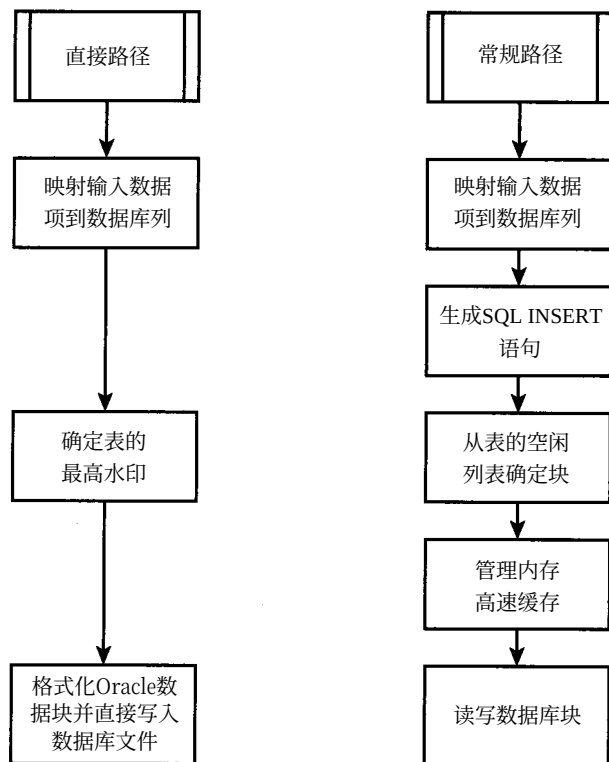


图31-2 常规路径装载与直接路径装载方法

31.4.1 使用常规路径装载

常规路径装载使用 SQL INSERT 语句和内存中的键数组缓存（bond array buffers）将数据

装载到 Oracle 数据库的表中。这个过程与其他进程竞争 SGA 内部的内存资源。如果数据库已经有支持多个并发处理进程的开销，常规路径装载会降低装载的性能。

使用常规路径装载的另外一个开销是装载进程必须搜索数据库，以查找被装载表的部分填充块，并试图填充这些块。这对日常的事务处理是非常有效的，但是它是常规路径装载的一个额外开销。

下面是一些情形与案例，要求最好或有时必须使用常规路径装载方法，而不能使用直接路径装载：

- 如果被装载的表是被索引的并且被并发访问的，或者如果要对表进行插入或删除，必须使用常规路径装载。
- 当在控制文件中使用 SQL 函数时，必须使用常规路径装载。当使用直接路径装载时，SQL 函数将不适用。
- 当装载的表是一个簇表时。
- 当装载少量记录到一个大型索引表，或当表具有引用完整性或检查约束时。将在下面的“使用直接路径装载”部分解释为什么在这种情形下使用常规路径装载要好一些。
- 当装载工作是通过 SQL*Net 或 Net8 在不同的平台上进行时，为使用直接路径装载，两个节点必须属于同一个计算机家族并且使用同样的字符集。

31.4.2 使用直接路径装载

不需要使用 SQL INSERT 语句和键数组缓存，直接路径装载格式化输入数据到 Oracle 数据库并将它们直接写入数据库中。注意直接路径装载总是在表的最高水位之上插入数据，这种方式消除了用于搜索部分填充块的时间。如果使用直接路径装载且装载工作没有完成，并且如果装载从开始处重新开始，那么被装载的数据将被截去，或者表会被删除并被重新创建。仅仅简单地删除数据并不能复位最高水印，而且会使用更多的存储空间。

理解使用直接路径装载方法对于表的索引会发生什么非常重要，这可以最大限度地提高直接路径装载的效率。在 Oracle8 中，当装载数据变为可见时（移动最高水印并提交），索引被标志为无用的，因为在这点上索引相对于它们索引的数据来说是过时的。当每个索引被更新为最新时，索引的无用状态便被消除。这使得当直接路径装载进行时，针对表的查询可以发生。但是，在装载结束阶段（移动最高水印并更新索引），如果查询要求一个处于无用状态的索引，这个查询可能会失败。

在装载期间，新的索引值写入临时段，在装载结束时，它们被存储并与旧的索引值合并。这时，索引再次变为可用。这样可以显著地提高对临时空间的需要。为了避免对临时空间的额外需求以进行排序，可以按照索引的序列对数据进行预排序，并且在控制文件中使用关键字 SORTED INDEXES。

在前面有关常规路径装载的部分中，列出阻止使用直接路径装载的限制，装载少量的记录到一个带有索引和 / 或引用及检查约束的大型表不是使用直接路径装载的一个限制因素，但是使用常规路径装载更为有效。在存在索引的情况下，或许常规路径装载在装载数据时更新索引要快一些，而且不需要进行一个大的排序 / 组合来创建新的索引。对于引用和检查约束，直接路径装载要求在装载前禁止这些约束。在全部数据装载后，重新允许检查约束，整个数据表都将根据这些约束进行检查，而不仅仅是被装载的数据。

31.4.3 装载嵌套列、嵌套表与变长数组

嵌套列对象是这样的一种对象，在嵌套列中，一个列对象在另外一个列对象之中。

嵌套表是另外一个表内部的表。这个嵌套表作为另外一个表中的列出现。这个表可以当作表被引用，也可以当作其他表中的列被引用。

变长数组是一个称为元素的有序对象的集合。这些元素可以依据它们在数组中的位置被索引。

程序清单31-17显示了两个嵌套列对象，em_contact嵌套在dept_mgr中，它是表dept的一个嵌套列。

清单31-17 LOAD6.CTL —— 例子6控制文件

```
LOAD DATA

INFILE 'example6.dat'

INTO TABLE dept

    (deptno          CHAR(5),
     dept_name       CHAR(30),
     dept_mgr        COLUMN OBJECT
       (name         CHAR(30),
        age          INTEGER EXTERNAL(3),
        emp_id       INTEGER EXTERNAL(7),
        em_contact   COLUMN OBJECT
          (name       CHAR(30),
           PHONE      CHAR(25))
       )
    )
```

例子6注释：装载数据到一个嵌套表中类似于装载数据到一个普通表。注意在前面例子中语法上的差别，在那里嵌套表作为一个列对象出现。

程序清单31-18显示了VARRAY和嵌套表在控制文件中的使用。

清单31-18 LOAD7.CTL —— 例子7控制文件

```
LOAD DATA

INFILE 'example7.dat'

INTO TABLE dept
(deptno          CHAR(4),
dept_name       CHAR(30),
emp_count       FILLER INTEGER EXTERNAL(5),
emp_str         VARRAY COUNT(emp_count)
  (name         FILLER CHAR(30),
   emp_str      COLUMN OBJECT
     (name       CHAR(30),
      age        INTEGER EXTERNAL(3),
      emp_id     INTEGER EXTERNAL(7)
     )
  ),
proj_sid       FILLER CHAR(30),
emp_projects   NEXTED TABLE SDF(CONSTANT 'project.txt', "proj1"
  SID(proj_sid) TERMINATED BY ";")
  (project_id   POSITION(1:6) INTEGET EXTERNAL(6),
   project_name POSITION(7:31) CHAR
  )
)
```

例子7注释：装载数据到一个变长数组要求 SQL*Loader确切地知道有多少记录要被装载，这可以使用 COUNT语法来获得，COUNT含有一个参数，包含变长数组的元素数目。还要注意COUNT的参数域直接出现在 COUNT语法之前。

注意 变长数组的每个元素在装载到数据库前，需要四个字节的客户内存。

SQL*Loader可能为每个变长数组需要至少两倍这样的空间。例如，如果有100个元素的变长数组，将需要400个字节的客户内存，并且SQL*Loader可能需要超过800个字节的客户内存来处理这个变长数组。如果在装载过程中用光了内存，试一下给 BINDSIZE或 ROWS一个小一些的值，然后重新启动装载。

31.4.4 使用SQL*Loader装载LOBS

LOBS或巨型对象，属于两个类别，一类存储在数据库的内部，而另一类存储在数据库的外部。BLOBS、CLOBS和NCLOBS存储在表中，外部存储的LOB存储在BFILE中。程序清单 31-19说明了存储一个内部定义LOB的控制文件，程序清单 31-20说明了存储一个内部定义LOB但是从LOBFILE文件中装载该LOB的控制文件，程序清单 31-21说明了一个使用BFILE的控制文件。

清单31-19 RESUME1.CTL — 例子8控制文件

```
LOAD DATA

INFILE 'resume1.dat'

INTO TABLE resumes

  (resume_name    POSITION(1:21) CHAR,
   resume         POSITION(22:1000) CHAR
  )
```

例子8注释：这个装载控制文件非常直观，因为 resume_name和LOB数据出现在同一个输入数据文件中。

清单31-20 RESUME2.CTL — 例子9控制文件

```
LOAD DATA

INFILE 'resume2.dat'

INTO TABLE resumes

  (resume_name    POSITION(1:21) CHAR,
   "RESUME2"      LOBFILE(CONSTANT 'Resume2' VARCHAR(4,2000))
  )
```

例子9注释：这个控制文件含有一个次级的数据文件，或称为 SDF，注意CONSTANT是SDF的名称，还要注意这个文件的头四个字节位置必须含有LOB对象的长度，这由VARCHAR(4, 2000)指定，这里4表示文件RESUME2的头四个字节的位位置将含有LOB对象的长度。

警告 当从LOBFILES文件中装载LOB数据时，如果LOB不能成功地装载，含有LOB数据的记录将被装载，但是含有一个空的LOB列。

清单31-21 RESUME3.CTL —— 例子10控制文件

```
LOAD DATA

INFILE 'resume3.dat'

INTO TABLE resumes
FIELDS TERMINATED BY ','

    (resume_name    CHAR(3),
     file_name      CHAR(50),
     resume         BFILE(CONSTANT "Resume", file_name)
    )
```

例子10注释：BFILE在操作系统文件中存储LOB数据，BFILE列存储真正含有LOB数据的外部文件的定位器。这对于巨型的 LOBS数据，例如视频数据很方便。

31.4.5 SQL*Loader性能提示

下面是增加SQL*Loader性能的一些补充技巧：

- 1) 使用定位域而不要使用分隔域，分隔域要求装载机搜索数据以查找分隔符。定位域比较快，因为装载机只需要做简单的指针运算。
- 2) 为终止域指定最大长度，使每个捆绑数组更为有效地插入。
- 3) 预分配足够的存储空间。当数据被装载时，表中需要更多的空间，Oracle分配更多的区间以容纳数据，如果在数据装载期间频繁地做这项操作，处理的开销将非常大。在装载之前计算或估算存储空间需求能够让你预先创建必要的存储空间。
- 4) 如果可能，在控制文件中尽量避免使用 NULLIF和DEFAULTIF子句。这两个子句对于被装载的每条记录都会引起列运算。
- 5) 分割数据文件，并行运行常规路径装载。
- 6) 通过使用命令行参数ROWS，减少提交次数。
- 7) 避免不必要的字符集转换，确保客户端的 NLS_LANG环境与服务器端的相同。
- 8) 只要可能，尽量使用直接路径装载方法。
- 9) 当使用直接路径装载方法时，为表的最大索引预先排序并使用 SORTED INDEXES子句。
- 10) 当使用直接路径装载方法时，尽量使用并行直接路径选项。
- 11) 在直接路径装载期间，尽可能少使用重做日志。有三种不同级别的控制实现这点：
 - 禁止数据库归档。
 - 在控制文件中使用关键字 UNRECOVERABLE。
 - 使用NOLOG属性修改表和/或索引。

31.5 小结

SQL*Loader是由Oracle公司提供的功能强大的应用工具。使用 Oracle核心（常规路径装载）或使用特别快速的数据装载（直接路径装载），SQL*Loader能够从几乎全部的数据格式中装载几乎全部的数据类型。SQL*Loader消除了编写装载特殊数据类型程序的需要。

第32章 系统管理员设计器

本章要点：

设计器：Oracle 8i的常用计算机辅助工程解决方案
设计器一览
设计器管理
提高设计器性能
应用编程接口
故障检测设计器

32.1 设计器：Oracle 8i的常用计算机辅助工程解决方案

首字母缩写词CASE代表计算机辅助软件/系统工程（Computer-Aided Software/Systems Engineering）。传统上，系统开发过程几乎不包括在系统开始运行之前的计划，而且信息技术（IT）组也没有持续地与系统的用户进行交流，这导致系统不能满足用户的期望。今天，许多组织依靠一种正式的方法论去指导他们的系统开发任务，方法论中有特定的关于在项目的特定阶段完成什么工作的准则，这会产生更好的系统。购买CASE工具或正在考虑购买CASE工具以帮助进行系统开发任务的企业数量正在不断增长。Oracle 8i流行的CASE解决方案是设计者工具集。

32.1.1 系统开发生命周期

生命周期的概念指的是将一个大型的项目分割为一些较小的称为阶段的单元。每个阶段由许多特定的任务组成，这些任务必须在下一阶段开始之前结束。生命周期从最初的方案概念开始，随后是计划和设计、编程、测试、实现，并最后使用系统。通常，生命周期在初始阶段花费较多的时间，此时从用户中收集关键性的数据，并且形成适当的文档。企业必须理解这一阶段的重要性，如果他们匆匆完成这一阶段，结果的系统也许不是用户所期望的。

传统的系统开发生命周期（SDLC）至少由五个阶段组成：

- 计划 —— 包括获得项目范围与项目如何进行的一个全面想法。
- 分析 —— 包括决定用户对被提议的系统与项目计划的要求。
- 设计 —— 包括数据库设计与程序模块化设计。
- 实现 —— 编程与测试系统。
- 支持 —— 系统投入使用，训练用户使用系统，并提供对系统的支持。

Oracle 8i公司有一个专用的方法论，称为Oracle 8i CASE方法。客户可以购买CASE*方法任务与交付使用（CASE*Method Tasks and Deliverables），其中有每一阶段的详细任务与结果。

在此有一些需要知道的术语：

- 应用 —— 软件工程要素的逻辑组，它可以代表从高级概念模型到特定系统模型的任何

一种模型。

- DES2K知识库/实例 —— 一个拥有一套数据库对象的 Oracle 8i帐户。每当你在DES2K中创建、编辑或删除事物时，底层表的记录由 DES2K通过特定的包进行修改。
- 下属用户 —— 可以通过同义词访问知识库的 Oracle 8i帐户。
- 知识库 —— 由用于与数据交互的表与视图以及管理它们的过程组成。

32.1.2 上游CASE与下游CASE

当今常用的关于 CASE的术语是指给定的 CASE工具是上游 CASE还是下游 CASE。上游 CASE工具用于决定系统的需求并产生图表和文档，用户可以在项目的早期阶段查看这些表与文档。上游 CASE工具不用于生成真正的数据库或程序模块。下游 CASE工具主要考虑 SDLC 的设计阶段与建立阶段。它们通过生成创建数据库与程序的语法提高了生产力。就其本身来说，两种工具都不充分，因为上游 CASE工具没有增加生产力，而下游 CASE工具不产生用户用于验证的文档。

32.2 设计器一览

设计器提供了一个完全的上游 CASE与下游CASE解决方案并支持整个SDLC。

在分析阶段创建的逻辑模型成为物理模型的基础，物理模型在设计阶段创建。应用可以使用这个模型生成，因而任何商务变化可以立即反映到应用中。用户也可以使用非 Oracle 的 CASE方法。

安装设计器的最小要求如下列所示（客户端，然后是服务器端）：

- 奔腾90MHz处理器或更高的处理器。
- CD-ROM驱动器。
- 300MB-600MB空闲硬盘空间，依赖于选择的选项。
- 最少40MB交换空间以运行应用。
- 32M可用RAM。
- 1GB硬盘。
- NT4.0、Windows 95或Windows 98。
- SQL*NET 2.3.4版本（如果在远程服务器上运行时，需要使用 SQL*NET）。
- SQL*Plus 3.3.4或SQL*Plus 8.0.4（对Oracle8数据库）。
- NET8客户8.0.4（对Oracle8数据库）。
- 显示分辨率设置至少为 640×480。

服务器端要求如下：

- Oracle7 7.3.4版本或更高版本
- 至少18MB共享池大小（Shared_pool_size）。
- 建议使用基于代价的优化模式。
- 至少30MB可用的RAM。
- SYSTEM表空间中有60M的空间，用于存储PL/SQL包、过程与视图。
- 回滚（ROLLBACK）表空间中有70MB的空间用于知识库的安装。
- SQL*Plus 3.3.4或SQL*Plus 8.0.4（对Oracle8i8数据库）。

- Net8客户8.0.4（对Oracle8i8数据库）。
- 对基于PC的服务器使用Windows NT4.0、Windows 95或Windows 98。
- 对基于UNIX的服务器使用SUN Solaris 2.5或相当的UNIX系统。

注意 参考安装手册以决定存储要求。

关于产品可用性的最新信息，请与 Oracle8i世界范围客户支持（Oracle Worldwide Customer Support）联系。在写本书的时候，可用的最新版本是设计器 R2.1.2，并且已在下列软件上做过测试：

- Windows NT4.0、Windows 95与Windows 98。
- Oracle 7.3.4或更高版本。
- 设计器对象扩展（对象数据库设计器 R2.1.1）。
- 具有1998年10月应用补丁的开发器 2.1产品。
- WEB 服务器3.0.1、3.0.1与4.0。
- Netscape 3.04金版、Netscape 4.0.6、Internet Explorer 3.0.2与Internet Explorer 4.01。
- Visual Basic 4.0或更高版本。

注意 应该确信所有的客户工具与知识库具有相同的版本级别，以避免兼容性问题。

32.2.1 设计器/2000 R1.3.2的改变

设计器2.x版本提供了多种新的功能，这些功能总结如下：

扩展服务器生成包括如下：

- 对Oracle7、Oracle Rdb与Oracle Web服务器的支持。
- 对非Oracle8i数据库的支持：MS SQL Server、Sybase、DB2/2与Informix。
- 对ODBC数据库的支持。
- DML的API代与对表的锁过程。

客户生成包括如下：

- 设计重用：在定义模型之后，组件可以与其他模型共享。
- 应用逻辑：存储PL/SQL、Visual Basic与Java代码。
- 生成器集成。
- 基于知识库中的公布设计信息生成数据库触发。
- 应用逻辑的100%逆向工程设计与100%的生成。

设计编辑器包括如下：

- 一个单独的、完全集成的设计与生成环境。
- 通过使用向导、拖放、就地编辑、鼠标右键菜单与同步更新的方法最大化使用的方便性。

升级到设计器R2.x的方法如下：

- 设计器R2.x可以执行在设计器2000 R1.x知识库上的更新。

应用升级包括：

- 在RON（Repository Object Navigator，知识库对象导航器）中的恢复工具，在“设计器系统管理”中对RON有详细讨论，它可以用于将一个早已存在的1版本应用或一组

应用到升级设计器 R2.x。一个两步过程将归档的应用升级为 1.3.2 可兼容的应用，然后再将其升级为 R2.x。这种方法的缺点是如果升级过程由于错误被中断，不得不使用最初的归档重新开始。为阻止这种事件的发生，在归档之前分析数据。

版本 1.x 与版本 2.0 中可以共存的，如下所示：

- 16 位版本 1.x —— 是，只要 16 位 1.x 版本与 32 位 2.x 版本在不同的 Oracle8i HOME 下。
- 32 位版本 1.x —— 是，将 2.x 版本安装在已有的 1.x 版本之上将不会产生问题，因为在 2.x 版本中缺省的路径名是不同的。

注意事项如下：

- 注册表中的 NLS_LANG 参数必须与服务器的 NLS_LANG 参数匹配，否则在安装与升级过程中装载包定义时会产生问题。
- 典型的升级将花费 2~3 小时用以删除现存的 API，还需要 2~3 小时装载新的 API 并执行实例转换与初始化。
- 如果 RAU (Repository Administration Utility, 知识库系统管理工具 —— 在“设计器系统管理”一节中详细讨论) 调用 SQL*PLUS，不要手工调用 SQL*PLUS，这会导致交出处理，甚至会毁坏知识库。

32.2.2 设计器组件

设计器工具集包括下列组件。

知识库服务包括如下工具：

- 知识库对象导航器 (RON)。
- 矩阵表。
- 知识库管理工具。
- 进程造模。

系统造模包括如下工具：

- 实体关系图表。
- 功能层次图表。
- 数据流图表。

系统设计器包括如下工具：

- 设计编辑器。
- 生成器。

设计器组件如图 32-1 所示。

设计器接口由不同部分组成，描述如下：

- 建模系统要求 —— 这个区域中的工具可以用于建模商务处理过程、创建 E-R 图，并创建系统中的实体模型、功能模型与数据流模型。
- 生成初步的设计 —— 这个区域中的数据仓库变换器与应用变换器可以用于从早期创建的模型中创建设计。

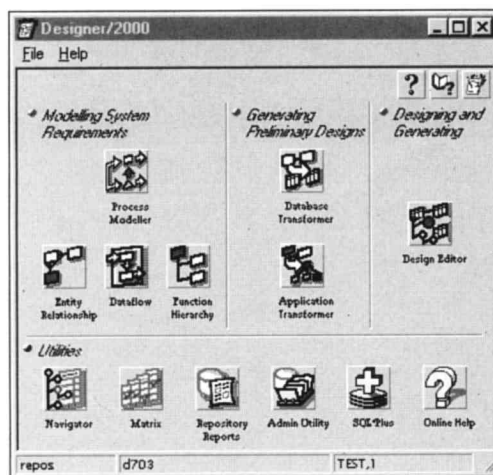


图32-1 设计器接口含有基于功能分组的不同组件

- 设计与生成 —— 这个区域中的工具可以用于设计一个代表系统功能的系统。可以使用知识库中的信息创建服务器端组件与客户端应用。
- 工具 —— 这个区域中的工具可以用于管理知识库，交互式地书写基于 SQL 的查询，并获得联机帮助。

32.2.3 设计编辑器

设计编辑器代替并强化了数据模型图（Data Schema Diagrammer）、模型数据图（Module

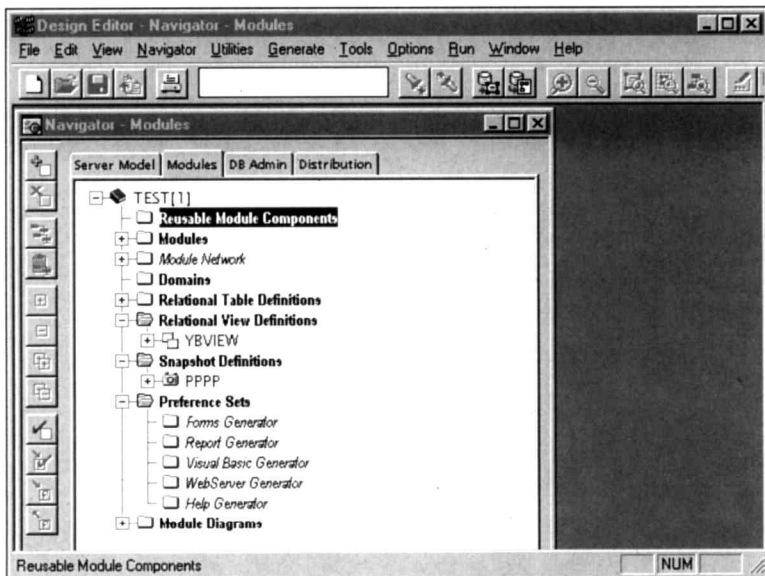


图32-2 设计编辑器导航器模型

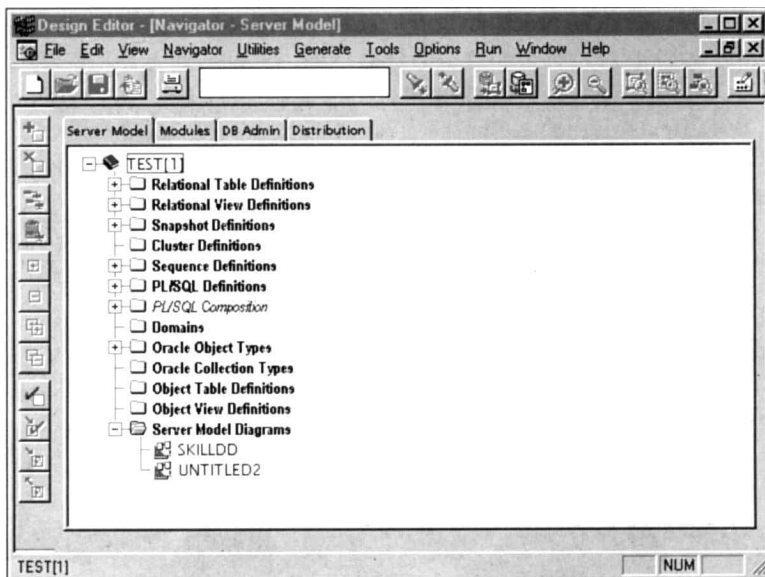


图32-3 设计编辑器导航器服务器模型

Data Diagrammer)、模型网络图 (Module Network Diagrammer)、模型逻辑导航器 (Module Logic Navigator) 与优选导航器 (Preferences Navigator) 的功能。设计编辑器是一个集成的环境, 使你可以在设计时执行不同的任务 (参见图 32-2与图32-3), 包括如下:

- 模型图的创建。
- 数据图的创建。
- 拖放功能。
- 导航器, 可以在图表范围之外对数据模型进行操作。

32.2.4 理解知识库

设计器的核心是知识库, 它可以记录所有输入设计器的信息, 它是 SDLC实现阶段的基础阶段, 包括以下功能特性:

- 知识库对象导航器 (RON) —— 知识库服务的一部分, 使用一种高度直接的方法查看与操作知识库对象, 如图 32-4所示。

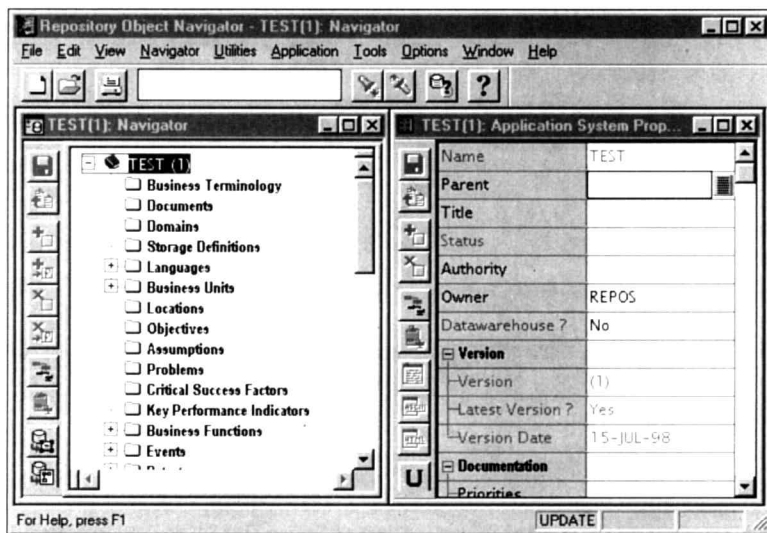


图32-4 知识库对象导航器组件

- 知识库基础 —— 这允许多个用户并发访问, 登记和注销工具, 也使得可以从其他产品如SQL*PLUS访问数据。
- 版本控制 —— 这使得应用可以从冻结的当前状态继续开发, 它可以维护系统开发时的历史状态, 也可以使用一个或多个版本进行应用的并行开发。例如, 可以开发一个同时在美国与英国使用的存货系统 —— 一个版本含有特定的英国细节, 另一个含有特定的美国信息。
- 质量检测 —— 通过相互参照与检测一致性规则, 提供了在初始阶段的多种质量检测, 可以在最初阶段修正错误。
- 文档 —— 提供了一种报表选项的扩展集, 以生成系统文档。在项目进行的过程中, 所有的信息输入相应的表单与图表中。可以容易地获取报表。可以运行一百多种标准报表。

■ 国际语言支持 —— 这使设计器可以运行在以任何字符集建立的 Oracle8i数据库上，包括多字节字符集。

还可以从知识库获得一些报表，如图 32-5所示。

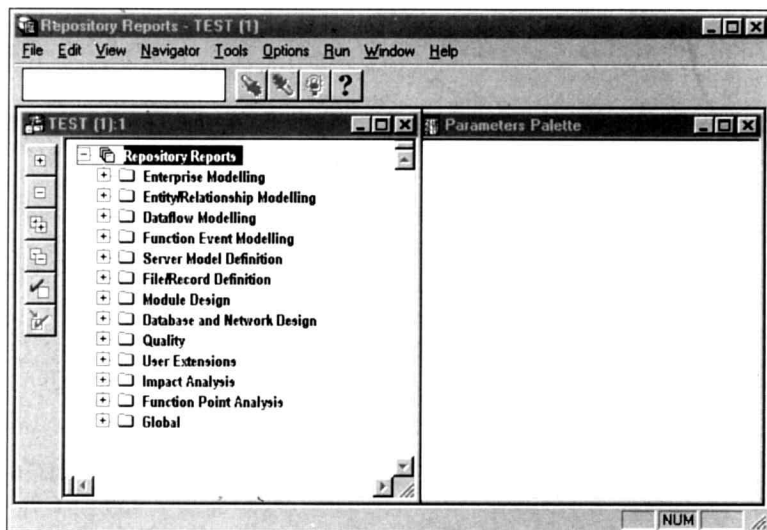


图32-5 知识库中可用的报表

32.2.5 使用图表

设计器工具集的所有图表的共同特性如下：

- 图形用户界面
- 点击能力。
- 下拉菜单。
- 上下文相关的帮助。
- 多窗口。
- 与知识库集成。
- OLE 2.0集成。
- 分层视图。

与DES2K一起提供的所有的图表与工具具有一些共同特性，可以使用相似的过程与它们交互。在安装设计器之后，可以在设计器组中看到一个 Oracle8i设计器图标，双击这个图标，并注册到知识库所在的 Oracle8i数据库。如果正打开一个现有的应用系统，在该点选择它，否则，可以在该点创建一个新的应用系统。

注意 可以使用以下语法从命令行启动设计器窗口：

DES2KXX 用户名/口令@数据库 /A:apps_system, 版本/S。用户名/口令@数据库是连接串，/A指明应用系统与版本，/S用于关闭一闪而过的屏幕。

要创建一个新的应用系统，需要完成以下步骤：

- 1) 从设计编辑器屏幕选择知识库对象导航器（RON），依次选择File | New Application。
- 2) 为应用系统创建一个名字。

可以继续使用这个应用系统工作，或通过从文件菜单中选择一个不同的应用系统，改变当前应用系统。

不同工具与应用的状态行含有关于当前会话的有价值的信息，如应用系统、版本与用户注册。所有的图表具有相似的外观与感觉，它们有一个多文档界面（MDI），使用户可以同时从不同的视图、通过不同的图表看到相同的对象。

图表大量使用鼠标来，操作对象与响应对话框。支持所有的标准 Windows 鼠标行为，如拖放、选择、双击改变属性、移动、重新设定大小等等。有一个完全功能的菜单系统，可以用于对对象与应用系统的简易维护，还提供了工具栏作为菜单系统的扩充，使开发者能够容易地访问最常使用的功能。DES2K 实用工具也有一个公共外观与感觉。使用两种类型的工具：

- 全窗口工具，如知识库对象导航器、知识库系统管理工具、知识库报表与矩阵图，提供与图表相似的 MDI 界面。这些工具提供了一个对象导航器与属性窗口以便能够容易地维护对象。
- 弹出窗口工具，如生成器与逆向工程工具，通常从另一个图表或工具中启动。

32.2.6 设计器使用的图表技术

在系统的逻辑设计与物理设计过程中，图表帮助你精确地形成计划系统的文档，并与用户和其他组成员交互。下面的设计器图表通常用于数据库的物理设计与逻辑设计过程：

- 实体关系图（Entity Relationship Diagrammer, ERD）—— ERD 模拟一个组织所需要的信息。应该包括项目组与最终用户之间的交流与一致意见。实体关系建模包括确定组织中的重要事物（实体），这些事物的性质（属性），以及它们怎样相互关联（关系）。一个简单的两实体 ERD 如图 32-6 所示。

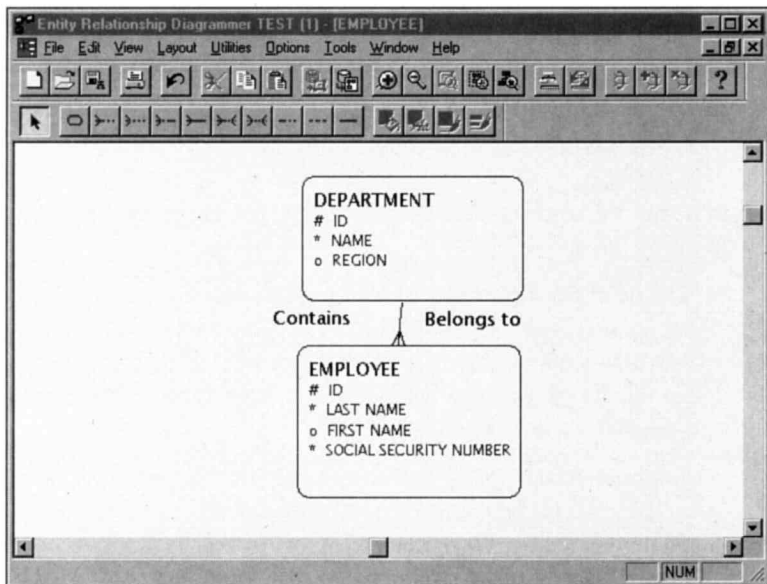


图32-6 两实体关系图示例

- 功能层次图（Function Hierarchy Diagrammer）—— 确定企业需要做的每一件事，并显示企业执行的功能与将来想要实现的功能。功能层次以全面的任务描述开始，然后分

解为一些主要的功能，并继续进行分解直至不能再分为止。

- 数据流程图（Dataflow Diagrammer, DFD）—— DFD描述商务活动中的信息流程及它怎样在外部环境和系统接口之间流动。
- 矩阵图（Matrix Diagrammer）—— 设计器的知识库服务可以为系统生成多种矩阵图。矩阵的目的在于标识在逻辑设计中可能被忽略的区域。
- 过程流程图（Process Flow Diagrammer）—— 设计器的过程建模组件，支持过程建模技术，可以用于支持业务处理重组技术（Business Process Re-engineering, BPR）或原型。也可以作为一个有关系统要求分析的理解给予用户的可视化反馈。

32.2.7 生成器

设计器提供了一些代码生成器，这些知识库工具可以用于在系统开发生命周期的建立阶段生成完整的、没有故障的代码。生成器的集合可以被分为两种类型：服务器代码生成器（Server Code Generator）与前端代码生成器（Front-end Code Generator）。

1. 表单生成器（前端代码生成器）

这是一个DES2K组件，可以基于知识库信息快速建立 Developer/2000应用。表单生成器具有许多功能，包括如下：

- 生成完全特性与菜单的表单应用。
- 生成具有图形用户界面项的应用，如按钮、选择框、选择按钮组等等。
- 提供外键验证。
- 提供商务规则的实现。
- 启用Oracle 表单应用的逆向工程设计并将它们存入知识库。
- 重新生成表单，而不会丢失由开发者增加的代码。
- 提供对16位版本的VBX控件与32位版本的OCX控件的支持。
- 提供OLE 2.0支持，并与基于Windows的桌面应用集成。

2. 报表生成器（前端代码生成器）

报表生成器是一个DES2K的组件，可以基于知识库信息快速地生成 Oracle8报表应用。它的主要功能包括如下：

- 支持多重连接报表。
- 提供许多报表风格与布局，包括控制中断、主从与矩阵报表。
- 增加开发者的生产力。
- 提供标准化模板。
- 可以定制生成器操作。
- 提供多种总计函数。
- 可以启用现存报表进行逆向工程设计，并创建存储在知识库中的模型定义。
- 使用改变的定义重新生成报表。

3. 服务器生成器（服务器代码生成器）

这是一个设计器的组件，提供基于知识库信息创建数据库对象与 PL/SQL模型的手段。它的功能包括如下：

- 为数据库对象创建生成ANSI标准的SQL。

- 生成检查约束。
- 生成并校验PK-FK关系。
- 支持Oracle 7基于角色的安全性。
- 生成PL/SQL模块，包括触发器、过程、函数与包。
- 支持快照。
- 逆向工程设计数据库对象并在知识库中存储它们的定义。

本质上，需要在知识库中定义对象，然后使用这个生成器指明希望脚本创建哪个对象。

4. Visual Basic生成器（前端代码生成器）

这个生成器与表单生成器相似，该生成器生成的代码可以被载入 Visual Basic6.0中，并可被用于创建一个可执行文件。

5. WebServer 生成器（前端代码生成器）

WebServer生成器可以用于创建Web应用。为创建一个WEB应用，应遵循以下步骤：

- 1) 在模型数据图中定义模型。
- 2) 使用WebServer从模型中创建PL/SQL包。
- 3) 在Oracle WebServer上运行生成的PL/SQL脚本以创建Web应用。

6. MS帮助生成器（前端代码生成器）

MS帮助生成器生成微软（MS）帮助文件格式的帮助文件，可以与表单和 Visual Basic应用一起使用。微软帮助生成器（MS Help Generator）将帮助文本放入一个文件中，并创建必要的链接与需要的跳转。还需要微软帮助编译器（MS Help Compiler），这是一个单独的产品，用以生成WinHelp格式的文件。

7. C++对象层生成器（服务器端代码生成器）

这个生成器使你能够通过创建 C++类定义并使用系统建模工具代码，访问对象关系型数据库。它可以用于为组成对象的数据与函数产生类。需要将生成的类与其他的 C++代码一起编译并将它与C++库链接。

32.2.8 模型重建机制

所有的生成后期工作应该以这种方式实现：模型的重新生成应该或多或少以一种简便的方式完成。下面的机制可以完成这项任务：

- 将所有信息放入知识库。这个方法确保改变自动地被应用到重新生成模型中。然而，这个方法导致对知识库系统管理的增加。
- 使用多个模板：一个基模板与其他特定表单模板。基模板可以用于系统中的所有表单，也可以将表单特定信息放入其他的模板，从而使对代码的管理变得容易。

32.2.9 Oracle CASE交换机

Oracle CASE交换机是一个可以在设计器与其他厂商的 CASE工具间进行信息双向交换的工具。这对于原先使用其他厂商的 CASE工具，而现在对使用设计器有兴趣的用户来说是非常有用的。如果一个组织使用其他的 CASE工具进行分析，如LBMS Automate Plus，现在想要使用设计器完成这个项目，那么 Oracle CASE交换机更为有用。DES2K的一个限制是它不能生成3GL代码。使用CASE交换机，可以将知识库中的信息转换到另一个 CASE工具中，并从另

一个CASE工具生成3GL代码。

Oracle CASE 交换机支持下列CASE工具。

上游CASE支持包括如下：

- Knowledgeware（智能软件）公司的ADW/IEW。
- Intersolv公司的Excelerator。
- Anderson Consulting公司的Design/1。
- LBMS Automate Plus。

下游CASE支持包括如下：

- ASK Ingres 6.0。
- Pansophic Telon。

CASE交换机加载前提供的校验检测如下：

- 没有属性的实体。
- 没有实体的属性。
- 没有关系的实体。
- 没有唯一标识的实体。
- 与目标工具的可兼容性。

32.2.10 使用设计器的瀑布定向方法

瀑布方法论对于CASE学习很流行，它利用以下阶段实现：策略、分析、设计与实现。在这个过程中可以使用的不同的设计器组件如下所示：

1) 创建一个APPLICATION，它将是你的工作区域。使用知识库对象导航器，如图 32-7 所示（选择File | New Application）。

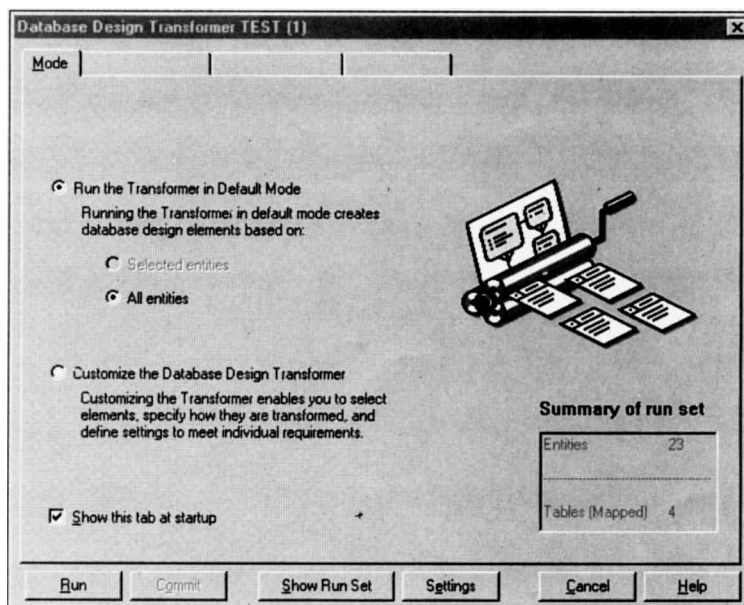


图32-7 数据库设计转换器组件

2) 使用实体关系图, 创建实体与它们的关系。使用 Edit | Domains 输入它们的有效值、它们的缺省值。双击一个实体以输入属性并指明它的作用域与关系, 同时增加缺省属性与有效值 (见图32-8)。

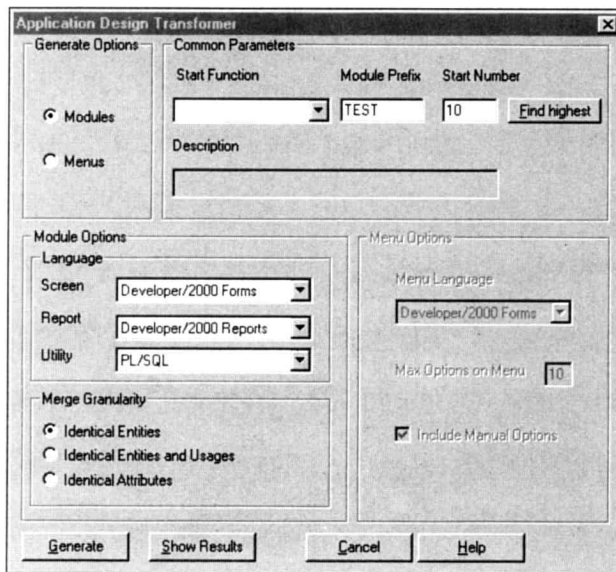


图32-8 应用设计转换器组件

3) 使用功能层次图创建这个图表。双击每个功能, 增加附加的信息以决定它是一个模型还是一个表单。

- 4) 使用矩阵图, 指明功能与实体属性的相关性。
- 5) 使用不同的报表以观察分析的质量。
- 6) 使用属性窗口定义一个数据库对象所在的区域。
- 7) 使用数据库设计转换器, 指明将被转化为表的所有实体。
- 8) 使用RON编辑表定义。
- 9) 使用RON增加违反规则的错误信息。
- 10) 使用RON检察约束以实现数据完整性约束。
- 11) 创建序列类型列以描绘主表的细节表 (detail table)。
- 12) 使用数据图编辑列定义。
- 13) 使用服务器生成器生成 DDL。
- 14) 使用RON的应用向导, 创建应用的一个图形用户界面 (GUI) 模型。
- 15) 使用模型数据图编辑模型定义。
- 16) 使用相应的生成器 (表单、报表、菜单等等) 生成应用。

表32-1显示了最常用的任务, 以及可以实现它们目的的工具。

表32-1 用于特定任务的设计器工具

任 务	工 具
建立过程流程模型	过程模型或数据流图
创建一个实体关系图	ER图
列出实体与过程	知识库报表

(续)

任 务	工 具
将实体使用映射为过程	矩阵图
功能分解	功能层次图
从一个阶段移动到另一个阶段	RON（应用版本）原型/生成器
实现GUI标准	设计编辑器
设计物理数据库	数据库设计转换器（参见图 32-11）
精炼物理设计	数据图与 RON
应用设计	应用设计 —— 转换器（参见图 32-12）
精炼应用设计	模型数据图、模型结构图、矩阵图、RON
开发应用模型	生成器
数据库对象创建脚本	数据图、服务器生成器
设计的明智检查	矩阵图
文档	知识库报表
修正错误	依赖于应用
用户与知识库管理	知识库对象导航器、知识库系统管理工具

32.3 设计器管理

设计器管理由用于监控与管理一个项目数据的技术集组成。DES2K提供的执行这些功能的工具可以通过文件、程序、知识库对象导航器（RON）中的应用下拉菜单与知识库系统管理实用工具（Repository Administration Utility, RAU）获得。系统管理任务包括知识库建立、对知识库用户授予权限、升级、版本控制以及数据元素共享。图32-9显示了知识库系统管理工具的界面。



图32-9 知识库系统管理工具组件

32.3.1 理解知识库

知识库作为标准Oracle数据库而被实现，它由存储关于系统被分析、设计与生成信息的表

组成。具有足够权限的知识库用户可以创建一个或多个应用系统，并将应用系统级的访问权限授予其他知识库用户。只要另一个应用系统的拥有者给予了他们共享访问该系统的权限，应用系统的创建者/拥有者可以从另一个应用系统将对象共享到他们的应用系统中。然而，这些共享对象只能在它们被创建的应用系统中进行修改。

为有效地使用知识库，理解它的设计与组成它的元素很重要。有一些由 Oracle 作为设计器软件一部分发货的图表对于理解知识库非常有帮助，它们可以帮助理解为什么设计器以这种方式工作。知识库由元素、这些元素之间的关联、以及元素或关联的属性组成。在知识库中可以存在许多类型的元素与关联，如实体、列、功能实体等等。

有两组知识库元素：主访问控制（Primary Access Controlled, PAC）与次要访问控制（Secondary Access Controlled, SAC）。主访问控制元素由应用系统拥有，删除一个主访问控制元素将删除它的所有次要访问控制元素。PAC 的例子包括表、功能与模型。次要访问控制元素由 PAC 拥有，一些 SAC 可以具有它们自己的细节 SAC。SAC 的例子包括属性、ID 等等。关联也需要一个双亲，并且不能独立存在，这一点与 SAC 相似。事实上，关联具有两个双亲——它连结起来的两个 PAC，当任何一个双亲 PAC 被删除时，关联也被删除。关联的例子包括功能-实体使用与功能-商务单元使用。知识库模型不允许使用关联的关联。

32.3.2 知识库大小

知识库系统管理工具具有一个功能，可以计算出知识库的大小为小型、中型或大型。安装手册提供了一张表，可以使用这张表决定某个大小的知识库所需要的表空间。表空间与回滚段分配的灵活性最小化了安装与升级过程中用完资源的可能性。

32.3.3 保护设计器知识库

有两个主要的方法可以用来控制访问设计器知识库：

- 知识库级 —— 可以控制一个用户是否具有管理员或用户权限。
- 应用系统级 —— 可以控制一个用户对独立的应用是否具有只读、插入、更新、删除、共享或系统管理权限。如果需要更好的控制，可以使用设计器的加锁能力。

知识库系统管理工具（RAU）的 Maintain Users tab（维护用户标签）可以用于创建新的知识库用户。RAU 只能由知识库的拥有者访问。这个标签含有如下信息：

- Oracle 用户名 —— 需要访问知识库的用户的名字（在数据库中）。
- 用户名 —— Oracle 用户的描述。
- 类型 —— 管理者（可以在知识库中创建新的应用系统）或用户。
- 描述 —— 用户注释。

点击 RECONCILE（调整）按钮以保存修改。

使用知识库对象导航器（RON）授予知识库用户访问一个特定应用系统的权限。在用户中只有对该应用系统具有系统管理权限的应用系统拥有者可以执行这种授权功能。通过选择应用，授予访问授予一定权限的结果描述如下表所示：

- 选择 —— 给予用户对应用系统的只读访问权限。
- 插入 —— 使用户可以在应用系统中创建新的主访问元素。
- 更新 —— 使用户可以修改主访问元素，创建或删除次访问元素。

- 删除 —— 使用户可以删除主访问元素。
- 共享 —— 使用户可以在另一个应用系统中共享这个应用系统的定义。
- 系统管理 —— 使用户可以应用系统级控制访问。具有这个权限的用户是应用系统管理员，系统管理员可以做应用系统创建者能做的几乎所有工作。应用系统管理员可以将对应用系统的访问权限授予其他用户、从其他用户回收应用系统访问权限、设置应用系统级权限、冻结与解冻参数选择、移去元素集上的锁以及执行许多其他功能。唯一的一个应用系统拥有者可以做，而其他系统管理员不能做的工作是改变应用系统的拥有权。

32.3.4 共享对象与传送对象

这些实用程序使多个应用系统可以使用一个元素的定义。表 32-2描述了这些不同的命令以及它们的使用。例如，如果有在一个应用系统中定义的 Codes表，而另一个应用系统中需要这个Codes表，那么可以共享Codes表，并通过使用设计器的设计重用特性（Designer's Reuse feature）以确保Codes表在所有的应用系统中的结构相同。共享定义的应用系统应该在相同的知识库中（参见表32-3）。

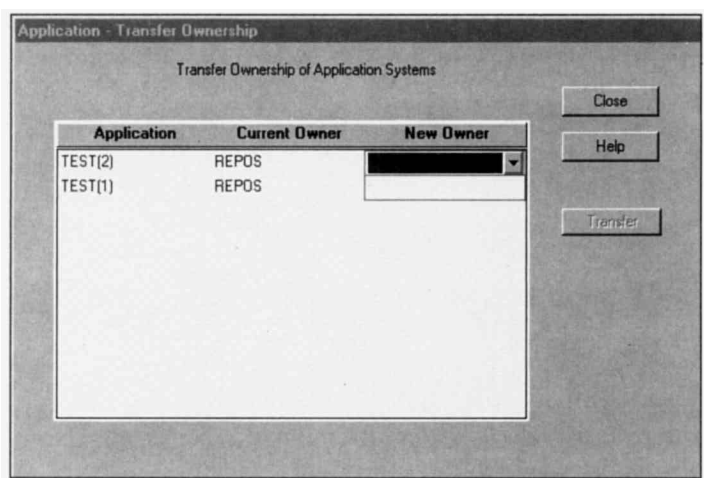


图32-10 转换应用拥有者

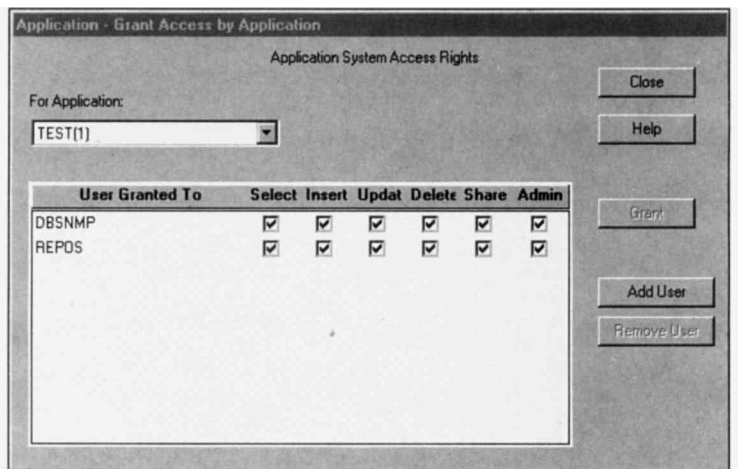


图32-11 授予访问权限

表32-2 在RON中共享与传输对象

命 令	描 述
应用、共享	应用系统声明允许另外一个应用系统使用该系统的对象。参见图 32-11
应用、不共享	一个应用系统声明不再使用另一个应用系统的元素定义
应用、转换拥有者	将一个主访问元素移动到另一个应用系统中，参见图 32-10

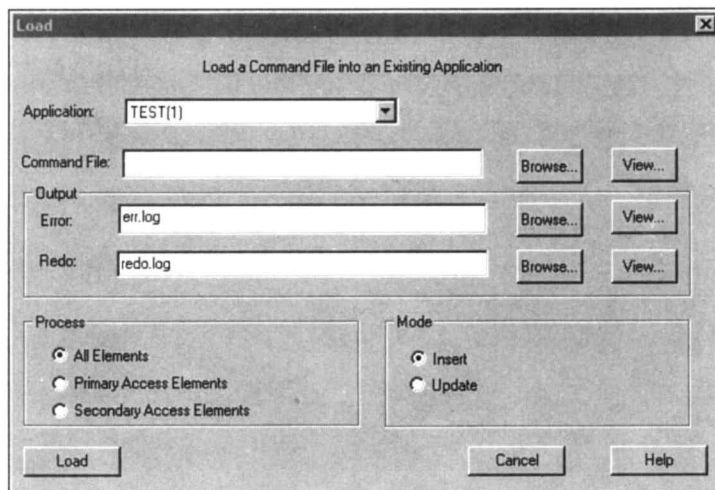


图32-12 知识库对象导航器装载工具

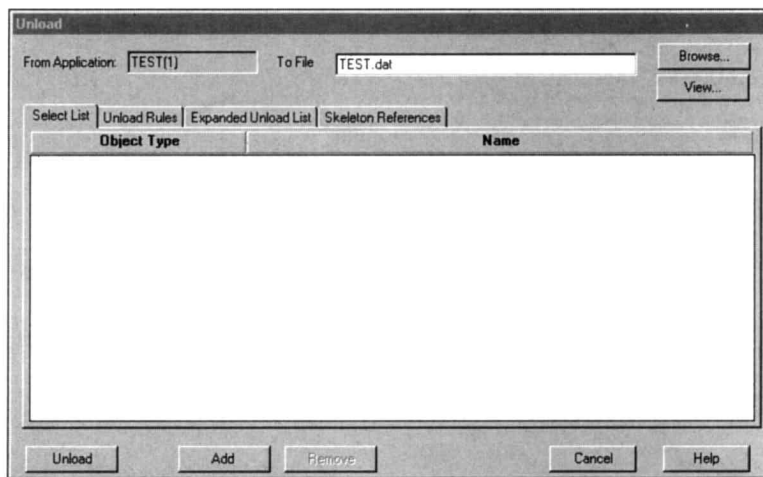


图32-13 知识库对象导航器卸载工具

表32-3 控制知识库中持有的声明

RON中的命令	描 述
实用程序、不装载	可以创建一个ASCII文件，该文件以专有形式存储以下定义：选中的主访问控制（PAC）对象、PAC的次要访问控制对象、与选中的对象相关的其余对象。参见图 32-12
实用程序、装载	与文件卸载相对。使装载的信息可以从另一个知识库或应用系统中卸载。参见图32-13

(续)

RON中的命令	描 述
实用程序、检入、检出、 锁设置、解除锁设置	与装载和卸载相似，检入与检出允许在应用系统之间的相同的或不同的知识库移动选中的元素。检出还锁定源应用系统中的对象，因而阻止在检出期间对该对象的意外修改。检入解开对象上的锁。参见图 32-14与图32-15
应用、归档	提取知识库应用版本到一个特定的临时归档表集合
应用、导出	将临时归档表中的内容输出到一个操作系统文件中
应用、重建	用于将.DMP文件装载到临时归档表，然后载入知识库中
应用、复位	刷新临时归档表。知识库中应用的版本不发生改变

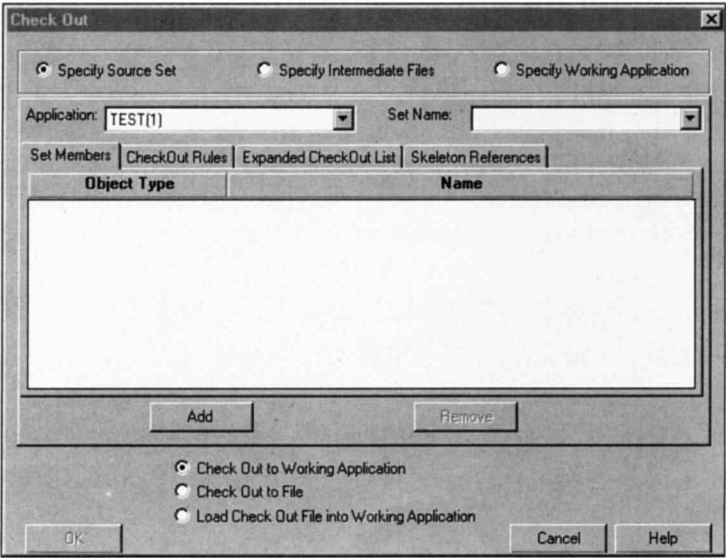


图32-14 知识库对象导航器检出

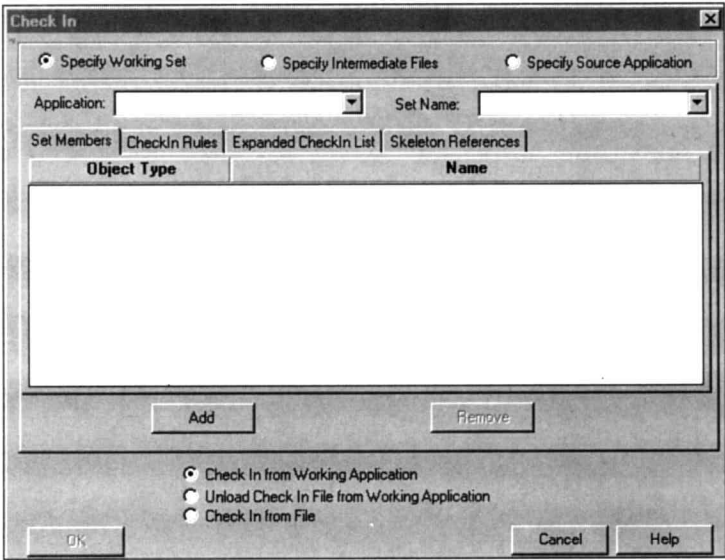


图32-15 知识库对象导航器检入

32.3.5 使用知识库的引用完整性

设计器 R2.x 具有一个工具，可用于检查知识库中含有数据的引用完整性。RAU（实用程序菜单）中的知识库分析器删除了冗余的或不合法的数据，确保了数据的完整性。为运行知识库分析器，知识库所有者必须被授予创建表与创建索引的权限。

执行以下 SQL 语句以检查知识库的状态。如果没有返回行，知识库的引用完整性是正确的。

```
SQL> select count (*)  
       from ckaz_actions  
       where check_no is not null  
       group by check_no, elmt_type, reld_elmt_type, action;
```

为使用基于成本的优化器，使用以下语句计算知识库表的统计信息：

```
SQL> analyze table <repository table> compute statistics;
```

Oracle 服务器可以使用 SQL 语句统计的最有效的路径访问被分析的对象。

32.3.6 版本控制与改变控制

对应用与管理类型具有管理访问权限的用户可以创建应用的一个新版本。为创建一个新版本的应用，应该连接到 DES2K，从 Design Editor 屏幕选择 RON 工具，然后依次选择 Application | New Version。

这会产生两个版本，一个版本是被冻结的，不能被修改，但可以被查看，而另外一个版本可以被修改。

如果一个应用系统与其他的应用系统共享元素，将为其他的应用创建占位程序的应用。

例如，如果有两个应用系统如存货（IV）与项目（ITM）共享对象，要创建存货应用系统的一个新版本，将创建 ITM 的占位程序应用（01ITM）。INV 的版本 1 与 01ITM 一致，INV 的版本 2 将与 ITM 一致。01ITM 被冻结。

要删除一个应用的某个版本，必须确保它的元素没有被其他应用使用。如果有正在被使用的元素，可利用如下步骤来删除应用：

- 不共享所有的共享元素。
- 同时删除所有的应用。
- 删除依赖的应用（假设没有内部依赖）。

需要考虑的问题如下：

- 版本控制是一个单用户工具，它将其他的用户锁在系统外。不要在普通商务时间或有人访问系统时运行它。
- 当运行时，它需要一个大的回滚空间，该空间与应用的大小成正比。
- 当创建一个新的版本后，应用在数据库中大约占据两倍的空间。
- 运行版本控制工具的时间与知识库的大小成正比，而不是与应用的大小成正比。
- 应用的版本可以在任意时刻被冻结或解冻，然而，解冻一个老的版本可能导致不一致性。

32.3.7 移植应用

下面几节描述在不同的应用之间移植软件设计的方法。

1. 导出/导入整个知识库

这个方法包括从源数据库将整个知识库转移到目标数据库中。转移的单元是 Oracle 帐户。必须在目标数据库中创建所需要的包。

需要完成下列步骤：

- 1) 使用RON，从备份组中选择Export Repository User（导出知识库用户）。
- 2) 将导出文件以二进制模式传送到目标机上。
- 3) 创建一个Oracle用户（参考安装指南获取详细信息）。
- 4) 使用RON工具，将用户导入到目标数据库中。
- 5) 在目标数据库中创建包。
- 6) 重新创建同义词。

当执行以下任务时，建议不要使用这种方法：

- 移动单个应用系统。
- 向一个早已具有其他应用的CASE实例中移入应用。

2. 卸载/装载一个应用系统

这个方法包括在知识库之间移动一个或多个应用系统，移植的单元是应用系统，通常用于在相同版本的DES2K之间移动应用。

需要完成如下步骤：

- 1) 作为对应用具有管理级与系统权限的用户访问 RON。
- 2) 使用Application | Reset清除提取表。
- 3) 使用Application | Archive, 将应用载入精炼表中。
- 4) 使用Application | Export将提取表导出到一个*.DMP文件中。
- 5) 将导出文件以二进制方式传送到目标机。
- 6) 在目标数据库中，使用Application | Import, 将*DMP文件装载入提取表中。
- 7) 使用Application | Restore将提取表载入基本表。

想要执行以下任务时，不要使用这种方法：

- 移动一个实例中的所有应用系统。
- 作CASE实例的备份。

32.3.8 移动主存取控制元素

通常有两种方法用于控制在CASE知识库之间移动主存取控制（Primary Access Controlled PAC）元素：卸载/装载与检入/检出。

1. 卸载/装载处理

这个方法被设计用于不需要任何限制的快速传输。元素被卸载装入一个扩展名为 .DAT的ASCII文件中，在保留元素之间的关联时，也卸载次要访问控制（SAC）元素。如果关联中的一个元素不是卸载集的一部分，它创建一个具有该元素名字的框架以完成关联。然后将元素从.DAT文件载入一个应用系统。

有两种装载模式用于解决冲突方案：

- 插入 —— 导致冲突的元素将不被装载，在日志文件中记录下冲突之后，装载过程继续进行。

■ 更新 —— 使用.DAT文件中的信息解决这个冲突。

2. 检出/检入处理

用户定义集（User-defined set, UDS）是一个非共享的PAC元素，它在应用系统中具有与其他PAC元素的关联。一个元素可以与多个UDS关联，一个UDS可以按照需要被锁定与解锁。可以在一个被锁定的UDS上执行的操作有：拷贝元素、共享元素或更新在该元素与非锁定元素之间的关联。不可以在一个被锁定的UDS上执行的操作有：改变元素的所有权、更新元素的属性或锁定含有该元素的其他UDS。

检出与检入的处理在UDS上执行。检出时，一个UDS自动地在源应用系统中上锁，并在检入时自动地解锁。如果一个源应用系统中检出一个UDS，则创建了一个.RCO文件。如果在工作的应用系统中检出一个UDS，则产生一个.RCI文件，这两种文件格式含有UDS以及它的关联。

表32-4比较与对比了哪些装载/卸载可以使用，而检出/检入不能使用的不同情形。

表32-4 使用装载/卸载与检出/检入的情况

情 况	卸载/装载	检出/检入
单用户环境	是，不需要追踪从一个应用系统中卸载的元素	否
多用户环境	否	是，需要持续追踪移植的元素。检出元素会被上锁，防止意外更新
使用其他应用系统的元素建立应用系统	是，这很简单，只进行一次处理，所以不需要定义一个UDS	否
备份一个元素	是，这很简单	否，不需要定义一个UDS或备份整个应用系统
需要对元素严密控制	否，它非常灵活	是，对检出元素的严密控制可以被归档
分布式开发	否	是
需要对源进行控制	否	是，从源应用系统中检出的元素要被上锁，以阻止更新

32.3.9 在文档中放置设计器图表

设计器启用许多输出方法，并提供了将不同的图表输出捕捉到诸如字符处理器和基于HTML工具的目标的能力，这些目标包括 Netscape Navigator（Netscape导航器）与 Microsoft Internet Explorer（微软互联网浏览器）。

1. 装载图表到MICROSOFT WORD

可以使用三种不同的方法完成这项任务：对象链接与嵌入（Object Linking and Embedding, OLE）、屏幕捕捉与打印文件。这些方法将在下面详细描述：

■ 使用对象链接与嵌入（OLE）—— 设计器是V2.0 OLE兼容的，可以用作OLE包容器或OLE服务器。为将一个ERD加载到WORD，需要完成以下步骤：

- 启动MS WORD，然后打开一个新的文档。
- 选择Insert，从菜单中选择Object，选择Entity Relationship Diagrammer（ERD）。

为将一个WORD文档装载到ERD，需要完成以下步骤：

- 启动设计器。
- 从ERD中，选择Edit | Insert New Object 并包括一个WORD文档。

- 使用屏幕捕捉 —— 有许多技术用于屏幕捕捉，包括使用 Alt+Print打印屏幕，使用画图编辑并粘贴图象，或使用第三方工具如 PaintShop Pro来捕捉屏幕。
- 使用打印文件 —— 除非选择另一个打印机，否则设计器工具打印到缺省打印机。使用打印管理器，可以修改打印机设置以便使打印的目的地是文件。然后这个文件可以被加载到文档中。如果缺省打印机是一个传真机，可以直接从设计器发送传真。

2. 将设计器知识库数据放入万维网

使用一个第三方驱动器，如 Adobe Exchange，它支持 .PDF 文件格式，可以将设计器知识库数据放入万维网（World Wide Web）。使用这个驱动器作为缺省驱动器，将文件保存为 .PDF 格式。这个文件然后可以在 WebServer 上被下载，并且建立了一个 HTML 链接以能够使用任意 Web 浏览器，如 Netscape 或 Microsoft Internet Explorer，在 Web 上浏览该文件。

32.3.10 使用设计器的逆向工程

大多数组织拥有许多应用，他们想将这些应用载入设计器。完成这项任务有许多方法，下面是最常用的方法：

- 逆向工程 DDL —— 这个方法可以用于从现存的 Oracle 7 数据库中提取数据对象，并将其编写成知识库中的元素资料。然而，它不能演绎通过程序代码实现的约束。它可以从本地或远程 Oracle 8i 数据库逆向工程设计数据对象。使用这种方法要求知识库用户具有表、视图与快照上的选择权限，并且具有对函数、过程与包的执行权限。
- 逆向工程表单与报表 —— 表单逆向工程工具提取 FORMS 版本 5.0 程序的信息，并将其放入 DES2K 模型定义与次要访问元素中。报表逆向工程工具与表单逆向工程工具非常相似，能够用多种语言逆向工程设计报表，包括 SQL*PLUS 报表、Oracle 预编译报表、SQL*ReportWriter 1.1 报表等等。

遵循以下步骤实现逆向工程设计表单和报表：

- 1) 在知识库中定义数据库。
- 2) 在知识库中定义主键约束与外键约束。
- 3) 将表单加载到 Oracle 数据库中，并通过一个批量作业或每次一个的形式逆向工程设计它们。
- 4) 手工插入任何没有得到逆向工程设计的逻辑。

注意 触发器、库附件、过程、过程调用不能被逆向工程设计。

- 从 RON、工具、表到实体更新改进 —— 这个工具将物理数据模型转换为逻辑数据模型。这个方法的效果和使用依赖于物理模型与逻辑模型相互之间的匹配程度。当使用这个工具时最需要警惕的是，确保已在物理数据库设计中定义了所有的主键与外键约束。应该注意这种方法不能将一个不好的物理设计转换为一种好的逻辑设计。事实上，它只是按照原样进行转换，因此，如果想要节省转换完成后需要进行的大量工作，必须将物理模型尽可能地与所需要的逻辑模型相匹配。这种方法具有以下限制：

- 应该存在约束定义，否则所有的列会成为属性而不是关系。
- 不改进共享表定义。
- 不应该事先将表映射到实体，如果已经映射为实体，必须删除这个关联或实体的定义。

为运行改进工具，可以从 RON 或实体关系图（ERD）中选择 Utilities | Table Entity Retrofit。

通常执行完所有的改进之后，需要清除所产生的逻辑设计。要这么做，需要完成以下步骤：

- 1) 将关系名改为更具有意义的名字，因为由改进生成的名字直接取自外键限制名，通常并不直观。
- 2) 检查关系的类型，确保它们的确是所需要的关系。例如，强制外键成为选择性的 / 强制的关系，而选择性的外键成为选择性的 / 选择性的关系。
- 3) 重新命名属性，使其对用户更具有意义，因为在物理设计中，属性名可以缩写。
- 4) 对结果逻辑设计进行一次明智的检查，以确保实体与关系正是所假定的，因为物理设计可能会以不同的方式实现它们。

32.3.11 使用设计器进行数据管理配置

数据系统管理包括监控与控制一个项目的数据库。为实现这个目的可以使用不同的配置，每一种配置都各有优缺点。设计器在 RON 与 RAU 中提供特性以简化这个功能。

数据管理配置由以下内容构成：清楚最适合你的组织结构的应用配置，同时知道这种结构最佳使用的知识库的数量以及需要支持的用户的数量。

1. 决定使用 CASE 应用的数量

在决定使用 CASE 应用的数量时，需要回答一些问题，包括如下：

- 组织的结构是什么？
- 团体之间的交流程度如何？
- 应用的大小是多少？
- 设计者对 CASE 技术熟悉吗？
- 控制与灵活性哪个更重要？

有一些机制可以使用，每一种有它自己的正面与负面效果。

2. 集中化配置

在集中化配置（centralized configuration）中，所有的数据由一个应用拥有，其他应用共享这些数据。

这种机制的优点包括如下：

- 易于管理，因为只有一组访问权限需要维护。
- 不需要元素传输，因为所有的元素都由一个应用拥有。
- 版本控制简单。

这种机制的缺点包括如下：

- 需要所有的用户访问一个数据库（单故障点）。
- 需要组之间更多的交流，因为他们共享相同的数据，因此不得不在许多事情上保持意见的一致性。
- 导致一个大的系统，因而工具的运行需要较长的时间。

3. 分散配置

分散配置（Decentralized configuration）意味着每个应用拥有它自己的数据，可以按需要将其元素的权限授予其他的应用。

这种机制的优点包括如下：

- 比集中方法具有更大的灵活性。
- 可以独立地进行应用系统的移植和版本控制。
- 导致较小的应用，所以工具运行速度较快。
- 可以访问的应用开发组数量的增加并不会增加相互锁定的可能性。

这种机制的缺点包括如下：

- 如果应用没有被很好地分区，会导致许多元素的共享与传输。
- 必须小心以确保没有冗余。
- 每个独立应用的界限和目标必须预先建立好，以使这个方式成功。

4. 联合配置

在联合配置（combined configuration）中，有一个主应用拥有多个应用通用的元素，并与需要它们的应用共享这些元素。也有一些应用含有应用特定的数据，这些数据不被其他应用共享。

这种机制的优点包括如下：

- 既可以共享元素，也可以进行独立应用系统的开发，同时具有两种方法的优势。
- 导致可管理的应用。

这种机制的缺点包括如下：

- 如果计划不当，会导致后期阶段复杂的版本控制与移植任务。
- 由于它所提供的灵活性，需要大量的系统管理精力。

5. 决定知识库的数量

当决定知识库的数量时，需要考虑几件事情，这些事情包括如下：

- 应用的数量以及它们之间的交互关系
 - 如果有多个应用，它们共享元素，它们应该驻留在相同的知识库中，因为不支持知识库之间的共享。
 - 一个应用只能存在于一个实例中，否则，会在后期阶段导致大量的合并。
 - 没有共同点的应用应该放在不同的实例中，以便对它们进行更好的控制。
- 预期的用户的数量
 - 某些工具，例如 Versioning（版本管理），运行在单用户模式下，如果这种工具被不同的应用系统并发运行，有必要将该应用放置在不同的实例中。
 - 如果达到硬件的物理限制，需要考虑多个实例。
 - 每个CASE用户需要一定数量的 RAM、交换空间等等。有些情况下，考虑一个客户 / 服务器解决方案更有益处。

表32-5 使用不同配置的正面与负面效果

需要考虑的因素	集中化配置	分散化配置	混合配置
管理访问权限的时间	低	低	中
版本控制的容易程度	高	低	中
对并行系统开发的支持	低	高	中
版本经常变化的系统的性能	低	高	中
需要使用多实例	低	高	中
应用的相互独立性	高	低	中

使用多实例具有它的优点与缺点：

- 会产生一个容错系统。
- 不需要所有的用户访问相同的数据库。

使用不同配置的正面与负面效果在表 32-5中总结。

32.4 提高设计器性能

设计器的安装指南上有许多需要满足的硬件与软件要求。下面的讨论提供了改进设计器性能的建议（小心使用它们，因为这会影响其他的 Windows应用。）

32.4.1 优化客户机

客户服务器配置的设计器工作得很好。要求至少386机器，使用不同厂家的硬件没有显著区别，内存要求至少32MB，然而建议使用64MB内存。在系统中使用以下技巧优化内存的使用：

- 使用硬盘优化器定期清除硬盘上的碎片。
- 不要直接将工作目录设到LAN上。
- 定期清除Temp（临时）目录。
- 设计器最小需要40MB交换文件。
- 重新安排装载程序的顺序，驱动器可以帮助最小化碎片。
- 禁用屏幕保护程序。
- 卸载不用的字体。

32.4.2 优化网络

网络性能会影响整个系统的性能，应该细心监控。

得到SWL*NET追踪文件或置入探测器以确定网络性能。在哪里存在性能降低。通过在sqlnet.ora文件中设置sqlnet.expire_time=<time_in_minutes>使用死亡连接检测。这将使监听器可以每隔先前设定的一段时间就进行一次检查，以确保没有已死亡的连接。

32.4.3 优化设计器

要优化设计器，有许多事情可做：

- 重建CASE索引。
- 重新安排知识库数据以使用当前环境的正确配置。
- 决定使用的知识库的数量。
- 决定在应用系统之间是否正确地共享元素。
- 得到诊断信息以确保SQL语句被优化地使用。
- 知识库存储在数据库内自己的表中，因此，数据库优化技术，如消除磁盘碎片，也适用于设计器。

32.4.4 优化数据库服务器

数据库优化是一门艺术，由于数据库上条件的改变，必须持续不断地在数据库上执行优化操作。Oracle出版社的《调整Oracle》（Tuning Oracle）是一本非常好的书，含有可以用于

优化数据库性能的多种不同技巧。下面的技巧会有所帮助（参考本书的第 16至第20章，以获取调整数据库服务器的更多信息）。

- 使用正确的优化器模式（推荐对 R2.x使用基于COST的优化）
- 截断RM\$HASH_ELMS。
- 通过合理地固定对象并且不使用一个大的 sort_area_size减少磁盘碎片。
- 减少硬盘I/O。
- 减少竞争，保持一个高命中率。
- 设置LOG_CHECKPOINT_INTERVAL的值比重做日志的大小稍大一些，以便只在日志切换时发生一个检查点。
- 将DB_BLOCK_BUFFERS设为正确的值，以防止知识库因为 API回到硬盘。
- 增加重做日志的大小与数量。
- 确保数据文件与交换文件在不同的硬盘上。
- 在一个与含有数据文件磁盘不同的硬盘上，创建知识库索引。
- 将重做日志与回滚段放在不含有交换文件的硬盘上。
- 将API固定在共享池中。

32.5 应用编程界面

当使用设计器开发应用时，你会意识到有一些需要实现的活动甚至使用设计器所提供的最完善的工具也无法轻易实现。DES2K的用户扩展功能可以帮助你把你所创建的扩展放入知识库中。API通过使你从知识库输入、输出信息以及创建用户扩展功能打开知识库。API基本上能够满足你使用任何方式维护知识库，因而越过了可能遇到的 DES2K功能性的限制。API允许使你可以根据项目需求，定制特定的功能，而不是被强迫地使用 DES2K功能性。这导致生产力的直接增长与生成的高比例。理解使用 API工作需要创建程序代码，这需要一个学习的曲折过程很重要。因此确保 DES2K没有真正地为你提供所寻找的功能是很重要的。从长远的观点来看，它将帮助你识别完成任务的最好方法。

32.5.1 使用API

API是一组在知识库拥有者模式中的数据库视图与 PL/SQL包，可以对知识库进行安全操作。知识库由一组表组成，并且这些表之间的关系没有形成文档。API由许多这些表的重要视图与PL/SQL包组成，允许你在这些表上不需要使用 DES2K界面就执行DML。

建议对 SQL与PL/SQL应该有一个较好的了解，以便有效地使用 API。对你试图操纵的元素和属性有一个透彻的理解也很有用。在知道需要使用哪个属性与元素之后，查看 API文档，包括联机帮助以及与软件产品一同发货的内部模型图，以便对所能使用的 API有一个更好的理解。与设计器一起发货的内部模型图非常有用，显示了 API视图与它们的关系。这些图表分组为以下子集：

- 商务计划模型。
- 商务需求模型。
- 数据库管理模型。
- 数据库设计模型。

■ 模块设计模型。

基于想达到的目标使用适当的模型。在理解了有关元素的细节以及 API的使用后，下一步工作是创建PL/SQL过程和包以进行必要的操作。

32.5.2 API视图与包

知识库基表的名字具有 SDD_与CDI_的前缀。API视图具有前缀 CI_。所有的知识库视图具有一些重要的列，包括如下：

■ CREATED_BY与CHANGED_BY，可以查询这些列找出元素的定义是何时被插入或更新的。

■ ID列，唯一识别知识库中的元素。

■ 列，对应它们所代表元素的属性。

■ NAME列，保存在RON中所看到的名字。

联机帮助系统在决定视图描述时功能非常强大。

API包以CIO为前缀，有300多个PL/SQL包组成API，每一种知识库元素具有一个包，这些包含有具有下列前缀的过程：

■ INS（插入）。

■ UPD（更新）。

■ DEL（删除）。

■ SEL（选择）。

联机帮助系统还提供了有关包的有用信息。除了前面提到的 API调用之外，还有执行一组语句而不是单独的语句，因而作为事务模型的 API调用。事务模型在事务完成前临时禁止约束检查，结果是，可以建立复杂的关系，并且约束检查被延迟到事务提交的最后一刻。 CDAPI包的主要责任是处理事务模型。

可以使用下面的步骤实现事务模型：

- 1) 声明将使用的应用系统与版本。
- 2) 打开事务。
- 3) 装载记录变量值。
- 4) 装载记录变量指示器。
- 5) 执行DML并进行校验。
- 6) 如果前面的步骤成功，则提交事务，否则，回滚事务。

32.5.3 API限制

虽然你已看到了API的强大而灵活的功能，但是它也有一些限制。例如， API不能操纵文本类型，如注释、描述、 PL/SQL块以及WHERE子句。可以从 CDI_TEXT表查出文本，但建议不要在这些文本上执行DML命令。 CDI_TEXT表的结构如表32-6中所描述。

表32-6 CDI_TEXT表的结构

列 名	是否可为空	类 型	注 释
TXT_REF	Not Null	NUMBER(38)	元素ID
TXT_SEQ	Not Null	NUMBER(6)	记录行号

(续)

列 名	是否可为空	类 型	注 释
TXT_TYPE	Not Null	VARCHAR2(6)	文本类型
TXT_TEXT	Null	VARCHAR2(240)	文本

API的另一个限制是不能直接从 Oracle Form中调用。

32.6 故障检定设计器

虽然设计器是“完美的”，也有发生非用户错误的时候，你需要判定导致错误的原因并解决它。这一节讨论一些常见的错误，以及用于诊断设计器错误的不同策略。

32.6.1 检查常见错误

在安装过程中，知识库系统管理工具检查它的工作，如果发生问题，则停止安装。常见的错误包括如下：

■ ORA-4030: out of process memory when trying to allocate <x> bytes

这是一个操作系统问题，原因是服务器已用完分配给进程全局区的内存。
解决方案：增加可用的交换内存或真正的内存，或减少服务器上运行的进程的数量。

■ ORA-4031: unable to allocate <x> bytes of shared memory

服务器上的SGA用光了内存。
解决方案：执行下面的命令：ALTER SYSTEM FLUSH SHARED_POOL;如果该命令不生效，增加shared_pool_size（查看建议的值）。

■ ORA-3113 or ORA-3114: no longer connected to the database

可能是一个网络错误，查看服务器的警告日志以获取详细信息。

■ warning: Package Body created with compilation errors

输入以下文本：SQL>show errors package body <package>。

32.6.2 使用诊断和追踪

有一些事物可以被用来诊断设计器组件，以便找出产生问题的底层 SQL，或生成一个 Oracle支持需要的追踪文件。表 32-7列出了可以使用的不同级别的追踪，级别的选择依赖于所需要的信息。

表32-7 诊断错误使用的追踪级别

级别	说明与使用
0	不追踪，没有诊断输出
1	生成SQL追踪
2	将SQL语句作为一个数字显示，缩小有关的代码
3	显示相关的SQL语句的约束变量
4	执行线程
5	使用函数自变量执行线程

注意 不是所有的DES2K组件都支持上述的级别，换句话说，一些级别在某些组件中是无效的。

表32-8含有oracle.ini变量，每个设计器组件都放置在 [designer (设计器)] 部分中。

表32-8 每个设计器组件的oracle.ini变量

组 件	变 量
过程模型	DES2_BPDMOD_DIAG_LEVEL
实体关系	DES2_SYSMOD20_ERD_DIAG_LEVEL
功能层次	DES2_SYSMOD20_FHD_DIAG_LEVEL
数据流图	DES2_SYSMOD20_DFD_DIAG_LEVEL
矩阵图	DES2_REPADM10_MD_DIAG_LEVEL
数据图	DES2_SYSDS10_DD_DIAG_LEVEL
表单生成器	DES2_CGENF45_DIAG_LEVEL
报表生成器	DES2_CGENR25_DIAG_LEVEL
RON	DES2_REPADM10_RON_DIAG_LEVEL
知识库系统管理	DES2_REPADM10_RAU_DIAG_LEVEL
知识库报表	DES2_REPADM10_REP_DIAG_LEVEL
知识库工具	DES2_REPADM10_UTL_DIAG_LEVEL
Visual Basic生成器	DES2_VBGENI0_DIAG_LEVEL
C++生成器	DES2_CPPGENI0_DIAG_LEVEL
DB设计向导	DES2_DATWIZ55_DIAG_LEVEL

例如，使用数据流图（Data Flow Diagrammer）时，如果接收到信息 ora-1422 exact fetch returns more than requested number of rows，应该编辑你的oracle.ini文件——将级别设为4——以便再次产生这个错误。检查追踪文件，找到与错误相关的额外线索。

下面只适用于Win95/NT：

- 追踪RON会话——可以修改注册表，使RON会话能够将追踪信息放入一个文件。当对注册表进行修改时必须非常小心，一定要在修改之前备份注册表。使用 REGEDIT或 REGEDT32，在\HKEY_LOCAL_MACHINE\SOFTWARE\Oracle\REPADM10结点下创建下面的键值。
- DES2_REPADM10_RON_TRACE_FILE——提供了追踪文件的位置与文件名。
- DES2_REPADM10_RON_DIAG_LEVEL——依据想要追踪的事务，将该值设定为某个追踪级别。表32-9列出了一个完全的清单。

表32-9 RON追踪级别

追踪类型	追踪值与意义
SQL区	SQL追踪 SQL语句定位 SQL语句
信息	致命信息 非致命信息 所有信息
调度	调度调用
外部调用	winExec命令 实用程序参数
通常	功能追踪 版本信息

这些追踪级别可以被累积使用。例如，如果想要追踪所有致命错误信息的 SQL追踪和SQL语句位置，使用设置 19 (1+2+16)。图32-16显示了可以被设置的一些诊断事件。

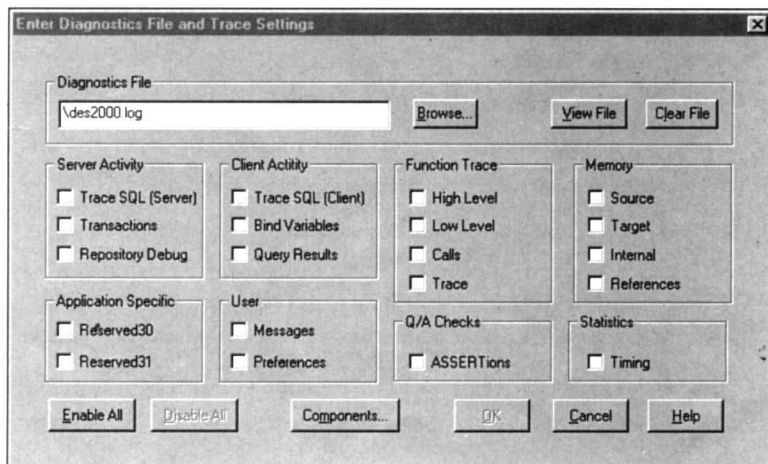


图32-16 诊断事件设置

32.6.3 从设计器有效地生成开发器应用的技巧

开发器是一族产品，可以用于开发客户服务器应用。开发器由以下主要组件组成：

- Oracle表单 —— 用于建立基于表单的应用。
- Oracle报表 —— 用于设计报表。
- Oracle图表 —— 用于建立生成商务图表的查询。
- 过程建立器 —— 用于设计并管理PL/SQL代码。

表单生成器以文档中不明显的方式使用知识库。一些能够消除从设计器生成应用的困难的技巧如下面列表所示，这些技巧可以帮助系统管理员 / 开发者使用设计器有效地建立应用：

- 在正确的时间进行培训 —— 如果人们在开始使用系统前的很长一段时间就经过培训，那么在开始使用系统时，他们会忘掉大部分所学过的东西。另一方面，如果他们在开发系统之后进行培训，他们会通过他们的自学与其他书领会事物，有可能不是使用最佳途径去完成工作。
- 建立用户界面原型，并征得用户的同意 —— 这非常重要，因为在界面生成后再进行用户界面的修改会花费很大精力。原型会传达给用户这样的信息，设计者真正理解的用户需求是什么，以及用户预期从最终产品得到什么。用户界面设计可以对所有的系统设计产生影响，甚至在数据的物理布局设计上产生影响。在用户界面设计时需要考虑的事情包括但不限于以下内容：工具栏设计、颜色、字体、单窗口还是多窗口、菜单系统、屏幕布局等等。
- 域的使用使得对列的维护非常方便 —— 域应该用于以下情况：
 - 在多个实体中出现的属性。
 - PK-FK属性。
 - 可以被标准化的列定义。
 - 具有一些相异可能值集的列。
 - 域的使用可以简单地将变化一步传播到全部有影响的列。
- 理解设计的限制 —— 不要向用户许诺一个使用当前的设计不能实现的界面，如果设计

不能反映用户所持有的最终想像，将花费大量人力与财力去解决表单后期生成。

- 规范化数据设计 —— 规范化数据设计更易于维护，也更能响应要求的改变。然而，一个标准化的数据设计对用户是不直观的。例如，一个标准化的数据设计将消除数据冗余并将数据放在不同的表中，但用户希望在同一个表中看到一切事物。

有三种不同的替代方法可以解决这个问题：

- 规范化物理设计与非规范化用户显示 —— 一个“标准”的数据显示对用户来说并不总是直观的，因此，用户需要争取一个非标准的数据显示形式。这个方法基本上总是会争取到一个标准的表单，但会导致一个高维护的系统。每当采取这个方法时，会导致生成后期的修改，如创建许多视图（非标准的显示），以协调两者之间的差别。
- 教育用户认识到标准化的使用 —— 如果用户可以认识到数据标准化的方法，这个途径可能是最佳的。还有，用户需要更加清楚数据库特定的事物，如主-外键。他们可能会乐于理解这样的事情，也可能不愿意理解这样的事情。
- 非标准化物理设计，并将显示与用户所期望的相匹配 —— 这一方法具有更高的生成（较低的生成后修改）的优势。然而，它对用户的需求具有较低的修改灵活性。在项目开始时，理解用户的需求非常重要，而且希望在设计的过程中这些要求不会改变太多。

记住，创建项目是为用户谋利的，因此，用户对项目最终结果的信心与满意对项目的成功非常重要。第一种方法通常涉及许多生成后的维护工作，因为设计没有满足用户的期望，因此，需要对表单进行修改或创建视图以调和项目设计与用户需求之间的差别。假设用户愿意接受对这种处理方式的学习，第二种方法非常有用，而且这种方法在不同的组织中是不同的。第三种方法涉及很小的生成后的管理工作，但很难将变化插入最初的需求中。这不是一个简单的选择，但如果开发组与用户不断地相互交流，这个过程会变得更具可管理性。

- 使用视图协调生成后的差别 —— 通常开发者害怕使用视图，因为视图是附加的需要维护的对象，并且比直接查询表的效率要低。然而，记住视图只是存储的 SELECT 语句，可以用于生成后阶段以满足用户所需求的最小程度的改变，如新的报表或应用安全性的改变。
- 在创建模块之前，确保已输入“缺省”数据 —— 在创建模块之前，应该在表定义中输入缺省数据。可以在每个表中查证下面的缺省值：显示数据类型、显示顺序、格式、提示、示意、缺省排序顺序、显示长度和显示宽度。在模块需要被重新开发时，这会节省很多时间。
- 显示有意义的错误信息 —— 当用户在域中输入不合法的信息时，最好为用户提供有意义的错误信息，显示合法的值是什么，而不是给予他们毫无意义的系统生成的错误信息。
- 加强标准化 —— 开发中的标准化会导致增加开发效率、减少维护工作以及减少新开发者学习时所走过的弯路。设计器提供了许多实现标准的报表，它还提供了一个 API，这个 API 由一套知识库表上的视图与一套允许知识库更新的 PL/SQL 包组成，它也可以用于产生报表，该报表列表/修复标准违反的情况。
- 学习表单与报表的生成优选项 —— 在项目生成阶段，花费在理解优选项上的时间最终会通过降低生成后的维护工作得到回报。
- 为应用设置缺省优选项 —— 熟悉可以在应用级设置，并且在生成的模块中保持不变的

优选项。对每个生成的模块，还有其他的优选项，该优选项根据需要可以一致于设计标准。

- 学习与使用表单和报表生成器 —— 许多开发者喜欢使用设计器生成器生成一个表单或报表的框架，而使用 Developer/2000 做大部分的开发工作。然而，DES2K 生成器非常有用，如果花一些时间理解它们的能力，会产生一个高比率的生成。
- 使用户参加一定范围的项目设计 —— 这对于得到用户同意原型设计有明显的帮助，与用户的不断交流也是必要的。然而，在某个特定点之后，不要允许用户再做出任何建议，否则，系统将会不断地改变，用户的期望也会越来越高。
- 应该在所有的模块生成之后开始生成后工作 —— 如果生成后工作在早期阶段开始，数据设计的主要改变会难于与最终模块合并。
- 文档后生成步骤 —— 使用 DES2K 知识库存储有关生成后步骤的详细描述，这样，如果需要重新应用这些步骤（通常会有这样的需要），开发者可以容易地再次运用这些步骤。可以使用模块文本域，如“注释”，或创建自己定制的文本域来存放这些信息。
- 定制模板 —— DES2K 提供了可以实现特定需要的一些定制的模板。
- 应该将 PL/SQL 放入库函数、过程与包中 —— 这会简化模块的代码维护。模板代码可以访问这些库，如果在库中代码有任何修改，模板会立即反映这个修改，而不需要重新编译。

32.6.4 设计器与 Oracle 8i

Oracle8i 的引入使数据库以及在这些数据库上建立的应用达到一个全新的级别，包括在存储的数据量与支持的用户量方面都达到一个全新的级别。设计器为 Oracle8i 的关系型与对象构造提供了一套完整的建模与生成工具。

设计器 2.x 提供对所有 Oracle8i 可伸缩性功能的支持，包括如下：

- 分区表。
- BLOB（二进制巨型对象）与 CLOB（字符型巨型对象）。
- 索引组织表（IOT）。
- 延迟约束检查。
- 类型表。
- 集合。
- 对象视图（虽然它是一个类型，但代表的却是一种关系型结构）。
- 嵌套类型。
- VARRAYS（多值列）。
- 嵌套表（嵌套在另一个表中的表）。
- 引用（直接从另一个对象引用一个对象）。

设计器的模式建模已扩展到使用 Oracle8i 的可伸缩性与对象功能，而不需要损害使用的简易性。它允许你设计数据库模式或将现有的模式定义载入到知识库中，然后服务器生成器将图形表示转换为相应的 SQL DDL 语言来实现该模式。设计器使用统一的建模语言（Unified Modeling Language, UML），一个对象管理组（Object Management Group, OMG）的开放标准来显示它的类型模型。

表32-10详细列出了设计器支持的版本清单（请与 Oracle 全球支持联系以获得设计器 R2.x 的更新的清单）。

表32-10 设计器支持的版本

设计器2.1.2	组件版本
平台	Windows NT 4.0或Windows 95或Windows 98
Oracle 服务器版本	7.3.4 or higher
设计器知识库系统管理	2.0.20.5.0
设计器设计编辑器	2.0.20.5.0
设计器服务器生成器	7.0.20.5.0
设计器表单生成器	5.0.20.11.0
设计器报表生成器	3.0.20.6.0
设计器库生成器	2.0.20.5.0
设计器MS帮助生成器	2.0.20.5.0
设计器Visual Basic生成器	2.0.20.5.0
设计器Web服务器生成器	2.0.20.6.0
Oracle安装器	3.3.0.1.3
SQL*Plus	8.0.4.0.0
Oracle Net8产品	8.0.4.0.0
报表运行	3.0.5.8.0

第七部分 Oracle网络

第33章 Oracle网络基础

本章要点：

- 理解Oracle网络产品特性
- SQL*Net与Net8体系结构
- 网络协议堆栈
- 使用开放系统互连参考模型
- 理解SQL*Net操作
- 安装与配置 SQL*Net

33.1 理解Oracle网络产品特性

Oracle的网络产品——Oracle7中为SQL*Net，Oracle8中为Net8，是透明的客户到客户与服务器到服务器应用连接的Oracle网络解决方案，这些产品含有丰富的网络服务核心，包括数据传输、网络命名、安全性与事务监控。这些Oracle的网络产品形成了一个抽象层，使用户和用户应用与物理网络隔离，并允许计算机间的异类、分布式计算，而忽略生产厂家、操作系统、硬件结构或网络拓扑。应用可以在如下环境工作：在令牌环网络上应用LU6.2网络协议的AS-400上，或在以太网上应用TCP/IP网络协议的HP-9000上。Oracle的SQL*net事实上在所有的Oracle支持的平台上都可以工作，从PC机到大型机，并且支持几乎所有的网络传输协议，包括TCP/IP、Novell SPX/IPX、IBM LU6.2、Net BIOS、DEC Net与AppleTalk。

注意 Net8现在允许应用从使用IIOP或HTTP协议的浏览器直接连接到Oracle数据库，而且还支持因特网技术如iFS（因特网文件系统）、企业级JAVA BEANS与因特网标准SSL（安全套接字层）协议以提供网络连接的附加安全性。

Oracle的网络体系结构为它的工作提供了良好的性能。在Oracle7中，SQL*Net支持网络负载均衡与容错。当存在到达数据库服务器的多条路径时，SQL*Net决定哪条路径是最有效的并建立相应的连接，而且在建立连接的过程中，SQL*Net使用第二个网络接口或网络软件协议堆栈检测主路径上的任何组件错误并自动切换到可能的替换路径。SQL*Net也检测断开的连接并释放所有的服务器资源，清除失败的连接。

Oracle 8与Net 8在性能与可扩展性方面胜过了SQL*Net家族，主要表现为两个新特性：连接池（Connection Pooling）与会话多路（Session multiplexing）技术。这些技术减少了服务器系统支持网络连接所需要的资源总量，允许一个单独的数据库服务器支持增长的用户数量。在Oracle8企业版中，一个新的称为Oracle连接管理器的中间件利用了多路技术的优势，担当了

数据库服务器的高速代理。作为代理，Oracle连接管理器作为防火墙控制网络级对数据库服务器的存取，从而提高了网络安全性。

除了连接池与会话多路技术，Oracle8i还提供了连接负载平衡特性。当有多个调度器与监听器时，Net8具有平衡这些调试器与监听器负载的能力。它由两个部分组成：

- 客户端负载平衡以在监听器之间分布客户端连接
- 连接负载平衡以在多个实例与它们的处理器间分布客户端连接。

其他Oracle网络产品具有的标准特性包括：

- 支持同一个机器上同时运行多个协议。
- 支持事务进程监控器如 Encina 与 Tuxedo。
- 对第三方应用的开放式接口，通过开放式接口与 Oracle 的 SQL*Net/Net8 OPEN 应用编程接口实现。
- 开放的数据库接口，对 Microsoft Windows 使用 Oracle 的 SQL*Net/Net8 开放式数据库互连 (ODBC) 驱动器，对第三方数据库存取使用 Oracle 的透明网关技术，Net8 为 Java 应用提供了 JavaSoft 的 JDBC 规范接口。
- 使用 Oracle 多协议转换时，在同一个连接中的多个协议。Oracle 的多协议转换运行在载有两个或多个协议的计算机上，这个计算机提供了转换协议桥以使运行一种协议的客户端可以连接到运行不同协议的服务器上。
- 因为每个数据库实例都使用监听器进行注册，实例现在可以进行连接负载平衡，因为它拥有所有远程监听器的信息。连接负载平衡均衡同一个服务器上不同类型实例与调度器间的活动连接数量。这使监听器可以根据每个调度器上有多少个连接，以及怎样装载实例运行的节点，进行它们的路由选择判断。

33.1.1 理解系统管理与管理组件

Oracle 的网络产品包括一套全面的系统管理与管理组件。SQL*Net 的图形化管理工具，Oracle 7 中的 Oracle 网络管理器及 Oracle 8 中的 Net 8 助理，是创建与管理 Oracle 网络的强有力工具。Oracle Net 8 助理与 Net 8 简易配置，通过提供由向导界面驱动的基于 Java 的工具，代替了 Oracle 的网络管理器。系统可以使用这些工具以集中方式或以分布方式配置所有的 Oracle SQL*Net 家族组件，包括数据库服务器、网关、客户及 Oracle 多协议转换。Oracle Net 8 助理可以运行在多数桌面及 UNIX 操作系统上，但 Oracle 网络管理器只适用于桌面系统。

Oracle 名字，Oracle 的全局命名服务器，也可以完全由前面的工具管理和维护，Oracle 名字允许系统管理员定义网络实体（如服务器地址、数据库别名等等）。

SQL*Net 与 Net 8 包括对简单网络管理协议 (SNMP) 的支持，可以使用基于 SNMP 的网络管理控制台监控 Oracle 7 与 Oracle 8 服务器（以及网络监听器、多协议转换与 Oracle 名字），这使得公司中现有的网络操作中心可以在他们早已存在的监控中包括进 Oracle 系统。

其他的系统管理组件与特性包括：

- 客户注册服务给了 SQL*Net 2.3 与 Net 8 的系统管理员谁正在存取数据库的存取注册信息，包括用户名与资源使用情况。
- 在客户端与服务端端的全面诊断错误与追踪日志。在 Oracle 7 版本 7.3 中的 Oracle 追踪与 Net8 中的 Oracle 追踪助理是诊断与性能分析工具，用于帮助系统管理员翻译生成的追

踪信息。

- 本命名服务器允许现在企业级路径服务器的杠杆作用，如 Sun NIS/ 黄页、Novell NetWare路径服务（NDS）、Banyan StreetTalk及OSF DCE Cell路径服务，这个高级网络选项将Oracle客户端与服务器集成为一个本地命名环境。
- 网络配置与路由测试工具，包括网络测试（NetTest）（用于测试通过网络的远程数据库连接），TNSPing（用于测试到服务器的连接），不需要服务器名、用户名与口令，TNSPing还允许完全诊断追踪写入追踪文件，并衡量客户到服务器连接的来回程时间。
- 在Oracle8.1.x中，数据库实例在启动时使用监听器进行自身注册，在以前的版本中，数据库实例的信息必须在LISTENER.ORA文件中进行手工配置。现在数据库实例注册通过服务器注册完成，它向监听器提供实例的信息，如数据库服务器的名字与实例名。

33.1.2 网络命名规定

在一个大型组织中的挑战之一是为所有的网络资源产生一个唯一的名字，常用的单层模型命名系统中，给对象一个单独唯一的名字，这在企业中经常是不够的，因而，Oracle支持使用域名分层命名网络资源。在一个域名模型中，网络资源命名取决于它们所在的组或它们的服务目标，例如，在Oracle中的市场部门应该属于marketing.Oracle.com域名。属于市场部门的TNS资源也许会使用这种域名命名——例如，产品数据库使用prod01.marketing.Oracle.com，监听器使用listener01.marketing.Oracle.com。

在Oracle数据库版本7.3中，如果没有为TNS资源创建域名，将使用world作为域名，这保证了需要域名的网络应用总会有一个域名，即使组织实现的是单层命名层次。

提示 Oracle工具需要所有的网络资源名字，包括域名，除非在sqlnet.ora配置文件中定义了一个缺省域名，names.default_域名参数指定了当需要解决资源名时，自动为网络资源名增加的域名。

在多数安装中，使用单层命名模型（使用缺省域名.world实现）已经足够，在你的网络拓扑上增加域名是多余的。不单单因为可以这样做，管理多域名的Oracle网络增加了安装与配置的复杂性。

33.1.3 理解可选择的安全性扩展名

最后一套Oracle网络产品包括一些安全选项扩展，它们组成了Oracle高级网络选项，包括：

- 使用RSA RC4或DES加密运算法则进行高速全局数据加密，将所有的网络负载加密为56位或可输出的40位密码键。
- 逐一注册，使用Kerberos与SESAME验证服务器、安全动态ACE/服务器令牌系统及Identix TouchNet II指纹验证系统。
- 集成OSF/DCE与DCE安全系统。
- RADIUS（远程拨号用户验证服务器）验证方法。

RADIUS是一个轻便的开放式协议，用于在网络的客户与验证服务器间进行用户验证、验证与记帐。RADIUS验证方法使用半径适应装置如令牌卡实现校验，而且，RADIUS支持使用

挑战响应机制与记帐。Net8提供了RADIUS服务的集成。

33.2 SQL*Net与Net8体系结构

通过引入Oracle 8, Oracle把当前的SQL*Net网络产品组重新命名为Net 8。如前面所指出的那样, Oracle在更新的Net 8产品中引入了一些新的网络性能与可伸缩特性, 所有Net 8中的其他组件与特性并没有改变, 与SQL*Net 2.3.3, Oracle7版本7.3的最后一个SQL*Net版本一样。在这个Oracle网络基础的讨论中将描述SQL*Net的函数。大多数例子将使用TCP/IP与以太网, 因为这是主流的网络体系结构。为使事情尽量简化, 将描述所有的行话, 并在必要时使用类比。

在大多数现有SQL*Net体系结构表中, SQL*Net表现为中心建设的块, 紧密地与网络层的网络传输 (Oracle网络适配器)、网络命名 (命名适配器) 及安全服务 (Oracle的高级网络选项) 集成。客户端应用位于这个SQL*Net层的顶部以存取Oracle7服务器、到第三方数据库服务器的透明网关、到其他应用的过程化网关或Oracle Web服务器。

33.3 网络协议堆栈

本节将详细讨论网络传输服务, 组成网络核心的局域网 (LAN) 和广域网 (WAN) 协议堆栈。当本地网络命名服务应用于本地网络协议时, 你也将接触到它们。本地网络命名服务用于映射带有网络地址的计算机的“英语”名字与别名。Oracle名字用于在Oracle数据库中映射英语名字与别名, Oracle名字经常以如下方式利用本地网络命名服务器的优势, 即允许系统管理员在Oracle名字数据库中指定服务器主机名, 而不是服务器主机的网络地址。

如前所述, 行话在出现时将被定义, 从网络体系结构与协议堆栈开始。网络体系结构是模块化组件的集合, 这些模块化组件连接到一起形成一个有效的协同解决方案, 每个组件为整体提供一个“服务”, 在选择与连接可用的组件技术子集的过程中, 建立了一个网络体系结构。正是这个能力混合与匹配了组件, 成为开放系统规则与协同工作能力的基础。

如果没有某些类型的参考蓝图指导组件装配及一套控制构造的标准, 这些部分无法即插即用, 这种环境存在于本地网络传输中, 这些标准通常称为协议, 参考蓝图称为堆栈。参考蓝图将网络框架描述为服务器层的堆栈, 更像一个层蛋糕。堆栈内的每一层都有一个具体的清楚定义的职责, 提供了对上一层的服务, 直到所有的堆栈组件一起工作。

在本地网络传输中, 参考模型是国际标准组织 (ISO) 的开放系统互连参考模型。标准由美国国家标准协会 (ANSI) 及电子与电力工程师协会 (IEEE) 802委员会开发及修订, 在本地网络传输上的Oracle的网络层及协议遵循Oracle的SQL*Net体系结构。

当前的SQL*Net体系结构蓝图遗漏一些重要的用于理解SQL*Net如何工作的协议层细节, 它遗漏了SQL*Net的核心网络协议, Oracle的透明网络底层 (TNS) 对等网络应用。正是TNS提供了主要的抽象层, 或常用的应用接口给许多网络产品, 这使得Oracle应用的硬件与网络透明——透明网络底层由此得名。TNS提供了服务器层获取本地网络传输、Oracle名字及各种安全性及命名服务。Oracle的网络监听器是一个运行在Oracle服务器上的TNS应用, 建立了客户端与服务器的连接, TNS层负责大多数错误信息, 当发现客户错误信息时前缀TNS-。

SQL*Net体系结构有五个基本层组成 (参见图 33-1) :

■ 特定网络协议堆栈——基础是计算机支持的本地网络传输及LAN或WAN, 最常使用的

LAN协议（例子中也使用）是TCP/IP协议堆栈，还支持SPX/IPX、NetBIOS、DECNet、LU6.2、AppleTalk、Banyan Vines、MaxSix、Named Pipes、OSI及X.25协议。

- Oracle协议适配器 —— 在TNS协议与本地网络传输协议堆栈之间功能调用的连接器，这个适配器是特定用于确切的本地堆栈的。
- 透明网络底层（TNS） —— 对等网络协议，与NetBIOS极其相似，组成Oracle网络产品的通用传输。
- SQL*Net —— TNS上部的层，提供一套用于校验与连接分布式数据库的功能。
- 用户编程接口（UPI） —— 在最顶部，是SQL*Net与客户应用之间的应用编程接口连接器。

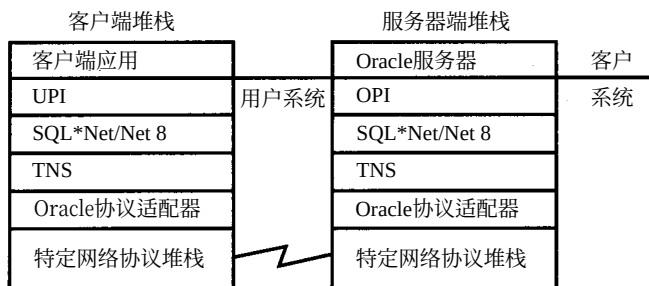


图33-1 客户端与服务端SQL*Net与Net8协议堆栈

33.3.1 Oracle协议适配器

Oracle协议适配器是特定本地协议堆栈实现与透明网络底层（TNS）之间的接口。Oracle协议适配器执行的功能类似于本地网络传输的接口，使用本地堆栈中找到的函数映射TNS的标准功能性。Oracle协议适配器特定用于本地协议堆栈的特定厂家实现。

33.3.2 透明网络底层

如前所述，透明网络底层（TNS）是Oracle开发的一个对等网络协议，它提供了建立网络连接的方法，并给予一个“管道”，通过这个管道传送SQL*Net、应用信息及数据。最关键的TNS应用是网络监听器。

网络监听器是一个与协议无关的应用监听器，或端口监督程序，接收所有Oracle服务器上运行的TNS应用的TNS连接。SQL*Net连接基于SQL*Net指定的端口或套接字，通过监听器进行区分，在TNSNAMES.ORA文件或Oracle名字数据库中的连接描述器的ADDRESS部分指明这些套接字，当使用监听器建立会话时，建立（或分配）了一个专有服务器进程处理连接会话。

33.4 使用开放系统互连参考模型

作为TNS与SQL*Net的基础，局域网（LAN）协议必须在任何顶部层工作之前正常工作。理解适当的LAN操作是理解Oracle网络的先决条件，正如在本地网络协议堆栈中描述的，将重点放在问题解决策略及正确的设计所必要的概念上。

如前所述，本地网络传输的参考蓝图是 ISO的七层开放系统互连（OSI）参考模型。在 70 年代后期，ISO组织了一个计算机专家小组建立标准，鼓励计算机与数据传输厂家之间协同工作的能力，最终的OSI蓝图将网络传输组织为以下七层：

- 第一层——物理层
- 第二层——数据链路层
- 第三层——网络层
- 第四层——传输层
- 第五层——会话层
- 第六层——表示层
- 第七层——应用层

这个参考蓝图对实现目标而言太过详细了，在此将描述一个简单的参考蓝图（参见图33-2），以基层、接口及协议堆栈开始。

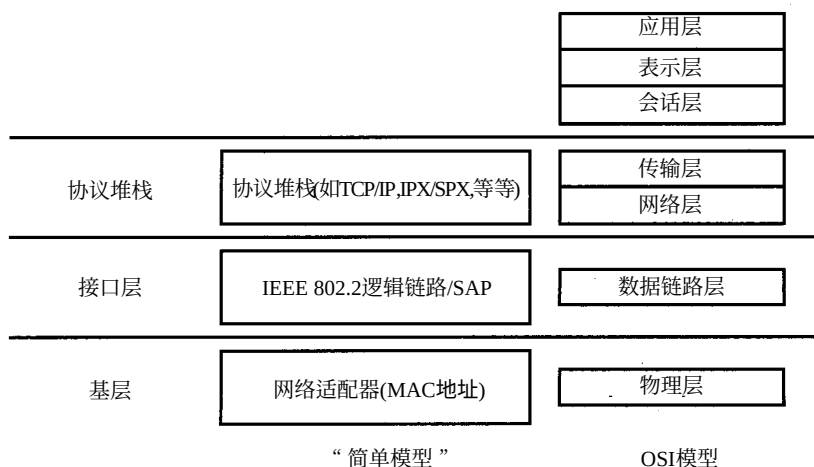


图33-2 与OSI对照的“简化”的三层参考模型

33.4.1 基层

任何建筑都需要一个坚固的地基，网络亦是如此，需要由物理电缆与电线组成的层来在计算机之间电力传输数据，这个世界由电子工业联合会（EIA）、电气工业联合会、TIA、电子与电气工程师协会（IEEE）统治，联合EIA/TIA标准委员会正在开发电缆标准与标准实践，这些标准协调电话与数据所必要的配线。在配线工业中采纳 EIA/TIA-568A标准驱使局域网成为星状设计，使用称为非屏蔽双绞线（UTP）的电话类型电缆。在 568A标准中还包括屏蔽双绞线（IBM类型1令牌环）与光纤电缆。

IEEE标准存在于大多数的流行网络技术中，包括以太网、令牌环、FDDI、100BASE-T、ATM及千兆以太网。

为什么要关心这一层呢？有两个主要的原因：首先，MAC地址（有时称为DLC地址）是一个几乎所有协议堆栈的主要的编址组件，单个工作站可以基于这个号码被识别并被追踪；第二在于这一层是网络中最不可靠的层，80%的网络故障发生在线路，这些故障不只是由于错误的或松的线缆连接引起的，错误的配线也会导致线缆故障。

提示 在以太网中，主要的故障资源来自过量的包冲突，原因在于在站点间有过多的中继器（多于三个），连接到带有SFE的中继器的以太网收发器开着（在这个实例中SFE决不会打开），而失效的以太网适配器在发送之前没有收听。

33.4.2 接口

在基层的顶层是接口，LAN协议的主要接口是IEEE802.2逻辑链路控制协议，它创建了一个简单而标准的方法，称为服务访问点（SAP）以获取线缆。对故障检定而言，这并不是非常重要的一层，但是应该记住SAP是用于访问基层的方法，因为这个术语偶然出现。

33.4.3 协议堆栈

在接口之上是协议堆栈，协议堆栈包围着OSI模型中的网络与传输层，多协议堆栈可以共享相同的线缆，而相同的协议堆栈使用不同的线缆。堆栈的网络层处理信息在网络传输的路由，或通过多条连接线缆传输的路由。堆栈的传输层提供了数据、数据流控制、错误检测与接收确认的无错传输。TCP/IP将它自己建立为标准的协议堆栈，其他的协议堆栈包括IPX/SPX、NetBIOS、DECnet、LU6.2、AppleTalk、Named Pipes、Banyan Vines等等。

协议堆栈是非常复杂的，许多东西进入堆栈，每个堆栈互相之间又极不一样，协议堆栈通常具有与MAC地址不同的另一个编址模式，这些编址模式通常将网络分解为子网或子网，在有些堆栈中也称为带（zone）或区（area）。与拨打长途电话需要区号一样，特定的子网编址用于与另一个子网通信，特定的称为路由器的设备，被放在临近子网的交叉点上，以检查信息并将它们传送到正确的子网。

33.4.4 TCP/IP协议堆栈

TCP/IP是用于SQL*Net与客户/服务器连接的最流行的协议堆栈。60年代由美国军方及西海岸美丽坚大学的一个小组研制，TCP/IP研制用于早期称为ARPAnet的因特网，TCP/IP设计的核心是当丢失网络的一部分时的“自体愈合能力”，考虑到核打击时的命令与控制。这对冷战时期的军队非常重要。它也是没有中心控制的巨型网络——因特网的很好的设计。

TCP/IP堆栈可以运行在几乎每一种基层上，包括以太网、令牌环、帧中继与ATM，这种灵活性在较大程度上说明了TCP/IP的成功。TCP/IP协议实现了两种协议，IP与ICMP，用以处理网络上信息的路由。

1. 因特网控制信息协议（Internet Control Message Protocol, ICMP）

ICMP是一个用于故障检定的非常重要的协议，因为它提供了测试通过网络的路由的错误及其他控制信息。一些最有用的ICMP信息包括：

- 回应请求/回应答复——TCP/IP工具，称为PING，使用回应请求和答复决定是否可以达到一个特定的地址，PING命令向远方结点发送回应请求。如果协议堆栈正在工作，它使用ICMP回应答复响应这个请求。当PING接收到答复时，它计算往返所需要的消逝时间，它还指出答复是否丢失，指示一个繁忙的网络，这个工具是绝对必要的。
- 不能到达终点——如果网络与子网之间的路由不能到达，ICMP报告这个错误，同时，PING工具报告一个不能到达的终点，指示一些类型的路由错误配置或网络失效。
- 超时——当信息的存活时间（TTL）计数期满时，ICMP向发送者报告一个错误。当信

息第一次在网络上广播时，TTL计数通常设置为20。每次当一条网上的信息从一个网络或子网经过另一个网络或子网时，TTL计数减1。当计数减为0时，ICMP将信息返回给发送者，同时提供最后一个路由器或跳的名字，这阻止信息进入无限循环并提供不好的路由器的名字。

另一个TCP/IP必不可少的工具使用这一特性追踪信息器到达终点的路由。TRACEROUTE工具以TTL为1开始传送信息，第一个路由器期满过计数并送回信息，这个路由器的地址就被TRACEROUTE记录下来，TTL为2的信息现在开始传送以发现第二个路由器，如此往复直至发现终点，在Windows95中，这个标准的微软工具称为TRACERT.EXE。

提示 在Windows95中可用的一个顶端的、自由的连接PING与TRACEROUTE工具称为Asub-O-Tronic，可以从<http://subspace.vie.com>的Virgin Interactive公司的下载网页获得。它是一个诊断工具，使用它们的SubSPace因特网游戏确定影响它们的客户/服务器游戏性能的网络瓶颈，最好拥有它！

另一个系统管理员必须关心的领域是正确的IP编址与配置。为使IP协议能正确在网上发送信息，必须正确设置一些参数。

2. IP编址

IP地址表示为4字节的十进制点符号，在十进制符号中，一个字节的取值范围为0至255，所以，4字节、十进制点符号的地址看起来应该在0.0.0.0与255.255.255.255之间。如前所述，IP地址指示的是一个特定的网络或子网以及一个特定的主机，这些地址分为以下几类：

- A类——A类地址的第一个字节是一个在1~126之间的数，A类地址使用第一个字节指派网络地址，最后三个字节指派主机地址，A类地址是一个拥有巨大主机数的保留网络。
- B类——B类地址的第一个字节是一个在128~191之间的数，B类地址使用头两个字节指派网络地址，最后两个字节指派主机地址，B类地址通常保留用于因特网服务供应商。
- C类——C类地址的第一个字节是一个在192~223之间的数，C类地址使用头三个字节指派网络地址，最后一个字节指派主机地址，在任何C类网络中只能有254个主机。
- D类——D类地址的第一个字节是一个在224~247之间的数，这些地址保留用于多点传送，不应用于IP编址。

一些IP地址是保留地址或有特殊意义的地址，一些重要的地址包括：

- 一个地址的所有字节都是0或都是255是不允许的，如0.0.0.0或255.255.255.255。
- IP地址的主机部分不允许都是0，因为主机部分都是0的地址是用于指明网络自身的，是一个保留地址。
- IP地址的主机部分不允许都是255，因为主机部分都是255的地址用于广播通信，可被网络上的所有节点接收。
- 地址127.0.0.1是一个自回环的测试地址，当ping地址127.0.0.1时，TCP/IP堆栈进行响应，表明堆栈处于工作状态。
- 第一个字节在248至254之间的IP地址是有特定用途的，不能用于IP编址。

提示 10.x.x.x网络被另外保留，用于想要使用一个未注册IP地址模式的结点，因特网

内的路由器忽略10.x.x.x网络地址的信息，强烈建议在大多数节点使用这种模式。

3. IP子网掩码

IP网络编址附加的另一个复杂级别是子网掩码，子网掩码用于扩展网络的数量，因为 IP编址域有4位字节的限制。可以从主机号中提出位以允许定义附加的子网，例如，可以选择使用10.x.x.x A类网络号用于你的非注册网络，可以使用这个地址再用于子网掩码以创建附加的网络。子网掩码定义了 IP地址的哪些位指派网络地址，哪些位指派主机号。指派网络地址的位值为1，指派主机地址的位值为0。因而，A类地址的头八位应该都是1，因为第一个字节是网络地址，所有字节都是1的十进制数是255，因而A类地址的标准掩码是255.0.0.0。

现在想要使用10.x.x.x地址的第二个字节用于子网地址，既然头两个字节已用于网络地址，所以掩码应该为255.255.0.0。这将导致254个子网，每个子网65 534个主机。

4. 缺省路由器

组成TCP/IP路由正确传过网络的 IP部分的另一个配置是指明一个缺省路由。这是与其他网络相连的你的网络的路由器的 IP地址。如果你正向另一个网络的一个地址发送信息，而你的节点不知道在到达目标路途上的下一跳地址，IP将你的信息发送到缺省路由主机，假设这个路由知道如何处理你的信息。如果失败，“网络不能到达”的信息返回。

5. 域名服务器（DNS）协议

TCP/IP协议的另一个重要的部分是 DNS协议，在静态网络中，主机名与它们的相应 IP地址都可以被输入到一个主机表中，并存储在每一台客户计算机上。当网络增长时，这变得难以维护。DNS服务，与Oracle名字服务相似，从一个集中的名称数据库中将正确的 IP地址发送到客户端。DNS还具有通过查询其他的 DBS服务器了解名字，并缓存结果的能力。如果连接到因特网对你的企业很重要，那么实现 DNS是非常重要的，DNS是所有因特网编址与地址分析的基础。DNS设计与配置是一个复杂的论题，超出了本章的范围。

6. 动态主机配置协议

动态主机配置协议（DHCP）提供了对 IP地址的简单自动的管理，任何 NT服务器、InnerNetWare服务器及大多数UNIX服务器可以配置为DHCP服务器。DHCP服务器使用网络的网络地址配置，并为它提供 IP地址。这种配置称为DHCP范围，将被“租用”的地址的范围也在范围定义中声明。DHCP也可以为客户端提供正确的子网掩码、缺省路由、在提供 IP地址时提供DBS服务器地址。DHCP是一个强有力的协议，应该在任何 Windows95或Windows NT客户端的网络上实现。

33.5 理解SQL*Net操作

对使用SQL*Net 进行分布式处理的 Oracle产品与数据库，SQL*Net提供了三种常用的操作，这些SQL*Net提供的功能在后台操作，对用户是不可见的：

连接操作 一个客户端应用通过提供要求的终点数据库的数据库别名，指明一个对远程数据库的连接的请求。这个别名，称为服务名，映射到存储在 TNSNAMES.ORA网络配置文件中连接描述串含有的网络地址，或由 Oracle名字服务的数据库含有的网络地址。

数据操作 客户端与服务器可以互相之间传送与接收数据，Oracle支持同步（单线程，ping/pong）请求与异步（多线程，窗口）请求。

例外操作 SQL*Net支持一些例外函数，如断开、重新同步及测试 SQL*Net连接。

33.6 安装与配置SQL*Net

所有安装信息适用于 SQL*Net版本2.3.3, 安装Net 8时参考操作系统特定的《Net8安装手册》。

33.6.1 计划网络设计

在安装SQL*Net之前, 需要先做一些计划, 包括如下内容:

- 网络层及LAN协议的使用。
- 集中化还是分散化Oracle网络管理。
- 网络与数据库组件命名。
- Oracle网络管理器工具的使用。

最核心的网络层决定是否需要使用多个LAN协议, 使用多协议极大地增加了Oracle网络的复杂性, 并增加了故障点的数量。如果可能, 只使用一个LAN协议。

第二个问题是集中化Oracle网络管理。与大多数事情一样, 插手的人越多, 出现的问题也越多, 有些结点是如此的巨大以致于需要进行分散管理。如果必要, 集中化网络管理。

Oracle网络中的许多事物需要名字, Oracle网络中使用的每个协议类型称为一个团体, 并给出一个团体名。一个企业的范围可以被打破以适于分散管理。这些范围称为域。缺省情况下, Oracle网络管理器将核心域命名为.WORLD。而且, 所有的网络监听节点、数据库服务名字、多协议交换及Oracle名字服务器都需要名字。为一个模式做文献并保留它, 一致性计数! 参见第34章以获取更详细的命名信息。

正如现在可以断定的, Oracle网络管理器是命名与配置所有这些网络组件的工具。

33.6.2 配置文件一览

客户机与服务器需要配置文件以进行正确的操作。客户机通常有三个由Oracle网络管理器程序创建的配置文件, 一个必须手工创建的选择文件。这些文件提供了关于网络终点、网络导航及如追踪与记录诊断的配置信息,

TNSNAMES.ORA —— 这个文件含有客户端使用的网络终点信息。这个文件含有的信息包括数据库“服务器名”与连接描述。在Net8中, 在这个文件中增加了另一节, 用以处理多个实例可以存取同一个数据库的Oracle并行服务器选项模式下的故障。在这节中可以使用不同的参数: BASIC、PRECONNECT、SELECT、SESSION及NONE。这些参数的详细解释参见Oracle并行服务器手册。没有Oracle名字, 客户端需要这个文件将短的服务名字扩展为完全定义连接。这个文件可以由Oracle网络管理器或Oracle Net8助手生成。虽然Oracle不建议这样, 但有经验的、熟悉相应语法的用户可以成功地手工编辑这个文件。如果想要这样做, 最好先备份这个文件。

```
# D:\ORANT\NETWORK\ADMIN\TNSNAMES.ORA Configuration
File:D:\ORANT\NETWORK\ADMIN\tnsnames.ora
# Generated by Oracle Net8 Assistant
EXTPROC_CONNECTION_DATA.US.ORACLE.COM =
(DESCRIPTION =
  (ADDRESS_LIST =
    (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC0))
  )
)
```

```

(CONNECT_DATA =
  (SID = PLSExtProc)
  (PRESENTATION = RO)
)
)
ORCL.US.ORACLE.COM =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = pxsharma-lap)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = orcl.world)
    )
  )
)

```

提示 ORCL是数据库实例的名字。

SQLNET.ORA —— 这个文件中含有 Oracle 名字的可选择的诊断参数及客户端信息。这个文件由网络管理器创建，但也可以含有需要进行手工编辑的特定节点参数。

```

# D:\ORANT\NETWORK\ADMIN\SQLNET.ORA Configuration
File:D:\ORANT\network\admin\sqlnet.ora
# Generated by Oracle Net8 Assistant
  NAMES.DEFAULT_DOMAIN = us.oracle.com
SQLNET.AUTHENTICATION_SERVICES= (NTS)

```

TNSNAV.ORA —— 这个文件用于使用一个或多个 Oracle 多协议转换器的多协议（多组）网络中。它列出了网络导航信息，如组及名字与转换机的地址，以到达这些其他的团体。这个文件由网络管理器创建，永远也不能进行手工编辑。

PROTOCOL.ORA —— 这个文件含有 LAN 协议需要的特定协议选项（如，Async 与 APPC/LU6.2）。如果需要这个文件，必须手工创建。数据库服务器也可以含有一个配置文件用于 TNS 监听，这个文件通常叫做 LISTENER.ORA。

LISTENER.ORA —— 这个文件含有一台机器上的所有监听器的名字与地址、数据库的系统标识（SIDs）、监听器控制工具使用的控制参数。

例如：

```

# D:\ORANT\NETWORK\ADMIN\LISTENER.ORA Configuration
File:D:\ORANT\network\admin\listener.ora
# Generated by Oracle Net8 Assistant
LISTENER =
  (ADDRESS_LIST =
    (ADDRESS = (PROTOCOL = TCP)(HOST = pxsharma-lap)(PORT = 1521))
    (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC0))
  )
SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = PLSExtProc)
      (ORACLE_HOME = D:\ORANT)
      (PROGRAM = extproc)
    )
    (SID_DESC =
      (SID_NAME = orcl3)
    )
  )
)

```

33.6.3 准备安装SQL*Net

Oracle 安装工具用于安装所有的客户端 Oracle 产品，包括所有的网络产品。Oracle 安装器

也包括Oracle客户软件管理器（OCSM），允许在任何Windows客户平台使用的安装共享文件服务器上安装16位（Windows 3.x）的应用。这并不意味着不能从共享服务器路径上运行Oracle产品的标准安装——可以这样做！OCSM在共享路径下运行，支持直接对客户端硬件驱动器上发送文件更新。

不是所有的SQL*Net配置都是等同创建的。有两个不同的Oracle网络产品与Oracle应用版本，16位版本支持Windows 3.1客户，32位版本适用于Windows 95与Windows NT，16位版本与32位版本的DLL装置极为不同，16位SQL*Net上的应用与工具在32位版本上无法工作。

然而，有一个好消息，Oracle网络产品的16位与32位版本可以安装在同一台PC机上，可以存在两个不同的Oracle HOME并且互不冲突，稍后将进行详细的描述。

这里还需强调的一点是Oracle安装与Oracle产品有两个不同的32位版本：一个32位版本用于Windows 95安装，另一个用于Windows NT安装，必须在使用的操作系统上运行正确的安装，Windows 95安装在Oracle产品媒体\WIN95路径下，Windows NT安装在\WIN32路径下。

另一个16位版本与32位版本的显著区别在于完备性，16位版本具有以下工具，而在32位版本中没有。

- GUI TNSPing应用，用于测试TNS协议堆栈与网络监听器，32位SQL*Net只提供了命令行的TNSPing。
- 网络测试工具用于测试通过SQL*Net协议的数据库存取。
- 客户状态监控器，一个简单的GUI工具，以总结性和打印表格的形式显示客户端软件与配置参数设置。
- SQLNET.ORA编辑器，允许设置重要的追踪与日志参数。
- Oracle客户端软件管理器（OCSM），OCSM目前不支持Windows 95与Windows NT的共享安装。
- Oracle HOME转换工具用于在多个Oracle HOME之间进行转换。

16位版本与32位版本之间的主要区别在于客户端配置信息的路径，16位版本的SQL*Net将配置信息存储在名为Oracle.INI的文件中，在Windows的WIN.INI文件中[Oracle]节内的ORA_CONFIG参数指明了这个文件的路径。32位版本将所有的配置信息存储在Windows注册中。

1. 决定Oracle HOME路径

当在客户端PC机上安装SQL*Net时，所要做的最主要的决定就是选择Oracle HOME，它是所有Oracle网络与应用产品的路径，它也含有重要的客户端与网络配置文件。对非OCSM安装，两个占优势的模式是将Oracle HOME安装在客户端PC的主驱动器上，或将Oracle HOME安装在一个常用的共享文件服务器路径上，前一个模式提供了当损失了集中配置管理时，可能最高的性能与最简单的方法，后一种方法则正好相反。

因为进行集中化配置管理必须放弃许多性能与简单性，为什么要这样呢？除能够对一个集中的共享的Oracle HOME进行快速更新与打补丁之外，还有四个重要的配置文件组存储在Oracle HOME下的\ORAWIN\NETWORK\ADMIN路径下。

```
TNSNAMES.ORA
SQLNET.ORA
TNSNAV.ORA
PROTOCOL.ORA
```

因为这些文件是使用Oracle网络管理器维护的，每次当增加或改变一个新的数据库服务名或服务器名时，这些文件也发生改变。将这些文件放在一个集中的路径下是非常有利的。

2. 选择独立的非OCSM Oracle HOME

如果选择将 Oracle安装在本地硬盘上，记住需要从网络配置文件的集中储存库中拷贝 *.ORA文件，这些文件应该拷贝到\ORAWIN\NETWORK\ADMIN路径下。

3. 选择共享的非OCSM Oracle HOME

如果想要将文件加载到一个共享网络驱动器上，一种称为共享非 OCSM Oracle HOME的配置，有许多步骤：

1) 在Oracle产品的安装过程中，一些 DLL文件写入\WINDOWS与\WINDOWS\SYSTEM路径下，文件必须被指明并拷贝给每个用户，一个指明这些文件的快捷方法是使用 ATTRIB命令关闭Windows与系统路径下所有文件的归档位：

```
ATTRIB C:\WINDOWS\*. * -A /S
```

2) 运行安装之后，搜索所有文件并打开它们的归档位，这些文件在安装时已被写入。

3) 将Oracle.INI文件拷贝到C:驱动器，并将其放在Windows路径下。

4) 编辑 C:\WINDOWS路径的 WIN.INI文件，令其中存在 [Oracle]节及关键字ORA_CONFIG=C:\WINDOWS\ORACLE.INI，改变这个关键字使其指向ORACLE.INI文件的路径。

5) 在搜索路径中增加\ORAWIN\BIN路径，这通常在网络注册脚本中完成。

前面的步骤帮助将配置文件平滑地安装在共享网络驱动器中。

4. 选择本地DLL/EXE,使用共享的非OCSM Oracle HOME

这是在以前的安装基础上的一个改进。在这个配置中，在本地 C：驱动器上做了一个共享\ORAWIN\BIN目录的拷贝。将这个共享路径改为在 AUTOEXEC.BAT文件或网络注册脚本中的本地路径，每次当共享bin路径更新后，这个本地bin路径也需要进行更新。

33.6.4 安装32位SQL*Net

这个描述假设你在本地硬盘上安装一个 32位SQL*Net的本地版本，如果运行的是Windows 95/98或Windows NT，你的PC将自动为你的操作系统运行相应的SQL*Net 32位版本安装。如果Oracle安装自动运行，跳到步骤2，否则，从步骤1开始安装32位SQL*Net。

1) 开始Oracle安装，在CD-ROM上32位版本的安装路径Windows 95为\WIN95，Windows NT为\WIN32。确定选择正确的路径。改变这个路径并运行SETUP.EXE程序。在Windows 95及Windows NT 4.0下进行Oracle安装，点击开始、任务栏中的运行，然后输入如下（将D替代为你的CD-ROM驱动器的字母）：Windows 95为D:\WIN95\SETUP.EXE，Windows NT为D:\WIN32\SETUP.EXE，点击OK确认这个选择。

在Windows NT 3.51（或更高版本中）进行Oracle安装，点击“文件”、程序管理器中的“运行”，然后输入如下（还要用CD-ROM驱动器字母代替D）：D:\WIN32\SETUP.EXE，通过点击OK确认。

2) 选择Oracle安装配置。

3) 下面被提示使用安装的语言，在选择一种语言并点击OK后，显示出一个Oracle安装设置对话框。

4) Oracle HOME的选择很重要, 在选择缺省之前一定要考虑到这点。在本例中, 你再次将Oracle安装到本地硬盘驱动器上, 缺省的 Oracle HOME 是: 32位Windows 95版本为C:\ORAWIN95, Windows NT的32位版本为C:\ORANT。

5) 下一屏是自动安装产品如Oracle网络、SQL*Plus、文献、等等的向导。

注意 如果以前安装有SQL*Net, Oracle建议在安装更新版本之前, 首先移走所有的组件。这可以采用如下方式完成: 选择自定义安装并移走所有屏幕右边的组件。

6) 对一个新的安装, 选择 Oracle网络以快速安装所有必要的 SQL*Net与TNS组件, 选择正确的底层网络协议及正确的堆栈厂商, 让 Oracle安装其余的。对Windows 95, 将在Winsock堆栈下。如果你正使用本地 TCP/IP堆栈, 选择微软Windows 95的配置。如果安装在相应的路径下发现重复的DLL文件, 安装要求你将它们改名, 以便不会与老版本发生冲突, 如果回答不, 安装创建一个文件列出发现的重复文件。

7) 当安装完成后, 显示为两个部分的Oracle安装屏幕。

已安装的核心SQL*Net文件显示在Oracle安装屏幕的右边, 而可获得的应用显示在左边。如果使用的是Oracle数据库的老版本, 需要加载面向这些版本的需要的支持文件 (RSF), 7.3 RSF早已安装。如果使用的是 Oracle 7.2, 也要安装这些RSF。出于支持与测试的原因, 最好安装SQL*Plus。如果需要支持Oracle 7.1或7.0, 需要找出一个较老的Oracle CD-ROM, 以找到这些老的RSF。

要安装特定的产品组件, 双击可用产品中的目标产品, 然后选择特定的组件进行安装, 点击“安装”开始产品的安装。

提示 更新Windows注册, 在HKEY_Local_Machine/Software/Oracle节及HKEY_Local_Machine/Software/Oracle/Oracle_Homes/Oracle1节下增加关键字USERNAME。它的值应该为用户的名字。当进行一个对Oracle服务器的SQL*Net连接时, 使用这个值, 这个值为你的连接显示在屏幕上, 帮助释放连接以关闭数据库。

提示 对注册的另两个改变也需要考虑: 在 Windows注册中的 HKEY_Local_Machine/Software/Oracle节及HKEY_Local_Machine/Software/Oracle/ Oracle_Homes/Oracle1节增加关键字CNTL_BREAK=ON。这个值允许使用Ctrl+C键停止一个长时间的查询。另外, 改变以上两节中的NLS_LANG的值可以改进使用ORDER BY 子句排序的排序数据。运行在 Oracle上的PeopleSoft软件需要使用 US7ASCII字符集设置。在Oracle.NII文件中使用如下语法: NLS_LANG=AMERICAN_AMERICA.US7ASCII。

33.6.5 使用Oracle客户软件管理器组件

Oracle客户软件管理器 (OCSM) 是Oracle Windows安装的一个新特性。它允许系统管理员在任何Windows客户平台上使用的共享服务器路径上安装 16位应用, 包括 Windows95与WindowsNT。安装是集中进行的, 对客户工作站的软件更新是自动的, 使用 Oracle安装中的智能。

Oracle客户软件管理器可以用于传送 Oracle 7工作组与企业服务器捆绑的16位客户端软件, 包括SQL*Net、Oracle Developer/2000的开发与运行版、Oracle Office。Oracle应用套件也可以使用OCSM安装, 它也是Oracle财务软件GUI版的一个关键性传输工具。

注意 两个著名的Oracle 7 OCSM的传递的应用省略是Oracle ODBC驱动器及7.1与7.0必需的支持文件，它们不再是Net 8的事件。

Oracle Windows安装创建了Oracle客户软件管理路径，并将选中的软件加载到这个共享网络路径上。Oracle Windows安装配置系统管理员的PC以运行Oracle客户软件系统管理，它是允许系统管理员设定客户端配置的工具。一组用户需要的确切的产品套件与配置有关。然后用户就可以分配相应的配置以进行工作了。系统管理员可以给用户一套产品由他们自己选择并进行安装。

客户软件系统管理还有其他的选项，如安装冲突解决、为系统管理员决定执行路径的能力。可以以三种方式分配客户使用Oracle应用：

- 在共享模式，直接取自文件服务器。
- 在EXE/DLL下载模式，为性能提高，只有可执行程序与DLL加载到客户端PC上。
- 在完全下载模式，在客户端机器上进行完全安装。

OCSM的用户部分是Oracle客户软件代理。这个小的可执行程序加载到客户端PC上，运行在Windows启动组中。代理监控服务器的改变并使用Oracle Window安装引擎自动更新客户端。

33.6.6 使用Oracle客户软件管理器安装SQL*Net

如果正在运行Windows 95或Windows NT，你的PC将为操作系统自动运行相应的32位版本安装，因为正在安装，OCSM只在Oracle网络产品的16位版本下运行。如果你的PC机已启用自动运行，取消这个安装，使用以下步骤利用Oracle客户软件管理器安装SQL*Net。

提示 可以在加载CD-ROM时按住shift键，禁用自动运行。

1) 开始Oracle安装，CD上Oracle网络产品16位版本的安装路径是\WINDOWS。到这个路径下，找到并以/SHARED选项运行程序SETUP.EXE，在Windows 3.1及Windows NT 3.51中使用“文件”|“运行”，在Windows 95与Windows NT 4.0中使用“启动”|“运行”对话框，输入如下命令（使用CD-ROM驱动器字母代替D）：D:\WINDOWS\SETUP.EXE/SHARED。

2) 选择Oracle安装设置

3) 下面将被提示用于安装的语言，选择一个语言并点击OK之后，将显示一个Oracle安装设置对话框，在公司名域输入你的公司名，然后在Oracle HOME域指定共享Oracle HOME的路径。

注意 在Oracle 8.1.x中，安装运行d:\setup.bat，其中d:是你的CD-ROM驱动器。

在Oracle 8.1.x中，网络组件安装与Oracle 8.1.x数据库安装合并了。

对Windows NT，网络文件如tnsnames.ora在%ORAHOME%\net80\admin路径下。

警告 在共享安装时，必须小心Oracle HOME。这个路径应该在一个所有的Oracle用户都可以获取的网络文件服务器驱动上，它不应该在本地硬盘驱动器上。

4) 然后被提示在Autoexec.bat文件中的/BIN路径设置，你看到Oracle客户软件管理器系统管理信息对话框，输入系统管理员的用户名，点击OK，现在显示标准的软件资产管理对话框。

5) 在共享Oracle HOME下安装Oracle安装、Oracle客户软件管理系统管理员及Oracle客户软件管理代理, 首先在左边的面板中使它们变亮, 然后点击安装。完成后, 安装所有其他必要的应用, 如SQL*Net及相应的Oracle协议适配器, 选择其他共享Oracle HOME需要的应用。

6) 退出安装。

与软件管理系统管理员相似, 系统管理程序安装在你的PC上, 需要使用PC进行共享配置的改变。

警告 不能同时是软件管理系统管理员和运行管理代理的用户, 必须选择一台不用作开发或生产的机器作为系统管理员机器。应该选择一个别名作为系统管理员的名字, 那么当你在办公桌上的计算机上安装Oracle产品时, 系统管理员的名字不会与用户名发生冲突。

33.7 小结

Net8 (以前称为SQL*Net), 是Oracle的标准网络软件层, 用于允许从客户到服务器的通信。在客户与服务器端都需要安装Net8, 在客户端可以连接到服务器之前, 服务器必须先启动监听进程。Net8可以正常活动所需要的主要文件是SQLNET.ORA、TNSNAMES.ORA与LISTENER.ORA。

第34章 高级安全选项

本章要点：

- 强力安全方针基础
- 因特网使用三层应用设计
- 防火墙：它们是什么？不是什么
- 使用高级安全选项
- 启用数据加密与校验
- ASO支持的Radius协议设备
- 安全套接字层（SSL）协议
- 对Oracle工具包的支持
- 理解多线程服务器
- 使用Oracle连接管理器
- 实例学习

随着因特网与基于Web应用的到来，数据库世界发生了改变，Oracle提供了一个新的选择以适应这个新社会的挑战。

高级安全选项（ASO）提供了创建一个高度安全环境所需要的所有工具，安全环境定义为秘密的，具有完整性与可用性，这是基础。ASO将帮助实现这些安全原则。

34.1 强力安全方针基础

ASO具有许多选项提供了强有力的安全数据库环境。这意味着使用这个产品可以加强计算体系结构中的脆弱点。安全性通常以一个计划开始，安全性计划通常基于企业安全方针，这个方针的基础基于预期的对计算资源及不保护这些资源的相关风险的处理。计算机安全策略的基础在于机密性、完整性与可获取性。事务的机密性包括用户的验证，与/或发起这个事务的服务器确认这是一个授权的事务。验证通过ASO支持的完整验证建立，这个验证可扩展到用户、数据库及Web服务器。

用户使用口令进行验证，Web服务器与数据库使用数字符号进行验证。

数据加密保证了数据处于保密状态，ASO支持所有加密运算法则的不同级别。

为什么我们需要一个高安全性环境？

数据库必须是安全的，可阻止计算机破坏者的侵入。在过去，数据库被限制在安全的内部网络中，由于数据库对外部世界是不可获取的，所以违背安全性是罕有的。数据库管理员应该勤奋地执行相应的安全，包括：

- 确定以相应的操作系统文件级安全保护所有的外部数据库结构。
- 通过减少操作系统超级用户口令的人员与有权注册到数据库服务器的人员，限制对数据库服务器的存取。

■ 物理地将服务器限制在一个安全的房间中，并以适当的防火墙进行保护。

提示 像所有其他的产品服务器一样，用于因特网处理的数据库服务器必须被保护（参见第23章）。使用ASO可以实现辅助安全，满足因特网数据库的需要。

由于基于Web的计算与电子商务的出现，今天数据库管理员面对的安全性挑战是一个令人畏缩的问题。网络不再是内部的安全地带，而是因特网，一个虚拟的世界，现在允许任何注册到因特网的用户存取公司数据的王冠。

34.2 因特网使用三层应用设计

用户使用瘦客户机连接到Web服务器，然后Web服务器连接到数据库（参见图34-1）。Web服务器成为数据库的客户端。而非因特网的用户使用典型的客户-数据库服务器两层配置（参见图34-2），本章后面将讨论客户端的配置，客户端可以是Web服务器或是传统的客户PC。

三层应用设计

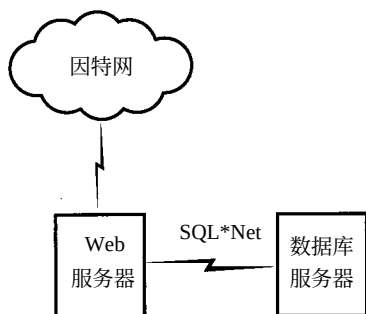


图34-1 三层应用设计

两层应用设计

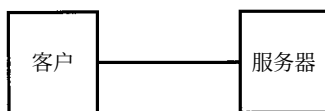
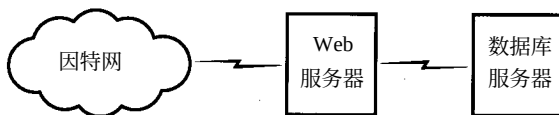


图34-2 两层应用设计

因特网会对你的公司造成威胁

因特网会对你的公司产生一个真实的威胁。计算机入侵者是活生生的，并且并不是按典型的九点至五点工作日进行工作。计算机入侵者有他们自己的协会，他们可以加入并学习怎

计算机入侵者仿效Web服务器



在计算机破坏者之后

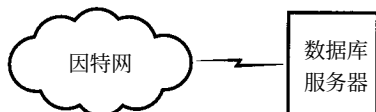


图34-3 计算机入侵者仿效一个Web服务器

样才能做得更好，做得更像一个标准的计算机专家，他们有 WEB 站点共享信息，甚至具有一本印刷出来的杂志，可以在任何稍大一点的图书链上获得。这个威胁是非常真实的，报纸上只报导了真正发生的计算机入侵中的极少部分。黑客具有探测你的网络的工具以发现你正使用的数据库服务器的类型与版本号。他们还具有工具用以猜测口令、读网络数据包及仿效 Web 服务器的 IP 地址以使你的数据库认为它正在与你的网络上的一个正式计算机进行交谈（参见图 34-3）。通过因特网存取一个公司的产品数据库需要特殊的安全性考虑，即使如此，计算机入侵者仍可进行侵袭。典型的攻击是试图击败一个或多个安全性原则，网络探测者可以查看数据包，这也许会损害你的数据的机密性。网络探测者可以使用 Windows NT 4.0 操作系统，所以即使是最可依赖的雇员也可能发现新的开发网络的方法，这些数据可能是信用卡号码或正准备与你进行交易的客户信息。

34.3 防火墙：它们是什么？不是什么

防火墙用于将安全环境与潜在的敌对环境隔离。防火墙的典型结构是通过因特网连接到一个 Web 服务器，然后由 Web 服务器通过防火墙将数据传送到数据库服务器（参见图 34-4）。防火墙担当从 Web 服务器到数据库器的入口。

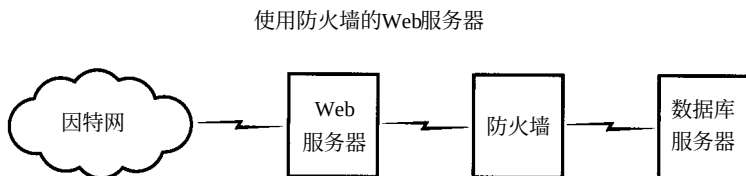


图34-4 使用防火墙的Web服务器

防火墙的目标如下：

- 锁定计算机资源的端口。
- 计算机资源间网络通信的约束。
- 约束通信的方向，即只允许网络通信沿一个方向传送。
- 限制Web服务器与数据库的交互类型。

34.3.1 在计算机资源中锁定端口

计算机入侵者寻找进入网络的方法，一个计算机系统上开放的端口是第一个对他们的邀请。SQL*Net需要使用1521端口进行SQL*Net通信，不同的操作系统及 Oracle 网关可能需要使用其他的端口。如果端口是打开的，所有类型的网络通信都可以使用这个端口，而不单只是 SQL*Net。

这允许计算机入侵者使用网络协议堆栈的第七层。挑战在于以安全的方式打开 SQL*Net 使用的端口，这正是防火墙要做的事情。

34.3.2 计算机资源间网络通信的约束

因特网数据库可以被成千上万的用户访问，这些用户有些想要破坏你的公司，有些不想。防火墙允许孤立数据库所在的数据库服务器或网络。

34.3.3 约束通信方向

防火墙可以打开一个端口，允许网络通信只能入站或只能出站。图 34-5显示了工作在典型的因特网配置中的两个防火墙。防火墙 A 允许在一个指定端口的入站与出站通信，防火墙 B 只允许在一个指定端口的出站通信。这阻止了获取任何由防火墙 B 保护的网上的事务，这个产品网络可以将数据项放入因特网，而不允许任何进站通信。这阻止了计算机入侵者试图探测这个产品网络，并阻止他们损害公司的核心。

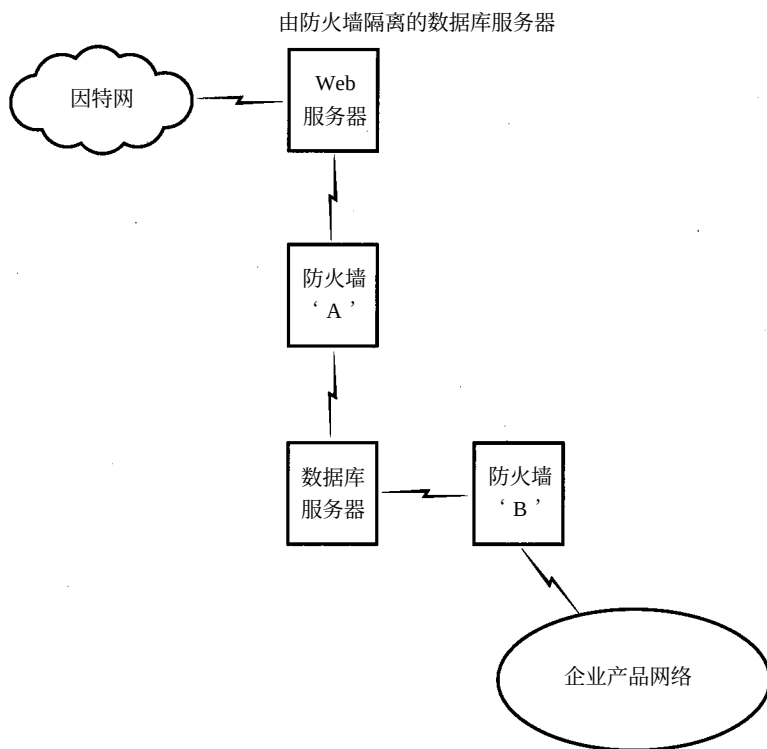


图34-5 典型的因特网配置

34.3.4 限制Web服务器与数据库交互

Web激活的数据库与公司的产品计算资源之间的事务应该被限制在仅有计算服务的必要点。例如应该关闭网络文件系统服务（NFS），这个服务已被计算机入侵者圈子开发了许多年了。

提示 当测试因特网数据库时，关闭所有的选项服务，如邮件服务、FTP、NFS。然后运行数据库，你也许会惊奇地发现你的数据库运行只需要这么少的服务。每次关闭一个服务都减少了一个计算机入侵者可以使用的区域。

防火墙怎样实现是基于公司的安全方针的，这个方针决定了防火墙设想作什么及使用何种类型的防火墙。这个方针基于一个风险分析，该风险分析显示公司的计算资源处于风险的何种级别及认为什么是可接受的风险。

提示 保护内部网络的最安全的方法是具有一个完全分开的网络用于因特网服务器。

34.3.5 防火墙类型

有三种主要的防火墙类型：筛选路由器、代理网关或防护 (guard)。

1. 筛选路由器

一个宿主/服务器使用一个路由器指引网络通信流向一个指定的网络，筛选路由器允许规则在数据包级别执行。规则必须不能太复杂，否则路由器会成为网络上的明显瓶颈。筛选路由器可以只查看包头信息并检查产生包的宿主的地址，尽管这可以进行伪造。筛选路由器可以配置为阻止来自一个指定网络的通信。它也可以用于在应用级控制通信，方法是限制端口的使用。例如，文件传输协议 (FTP) 使用端口 21。如果 FTP 应用被关闭，筛选路由器会关闭端口 21 或只允许 FTP 通信沿一条路流动。

2. 代理网关

筛选路由器只查看数据包的头，并允许有危害的传输发生，只要它们通过筛选路由器。代理网关担当因特网应用与因特网试图交互的内部网之间的真正的中间人。电子邮件，当用于因特网时，连接到公司的邮件服务器（参见图 34-6），代理网关允许因特网连接到网关，并且因特网认为它已连接到邮件应用，这允许代理网关筛选所有到来的命令并只执行允许的那些命令，这个行为对邮件服务器与 Web 服务器都是透明的。

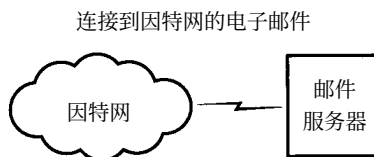


图34-6 连接到代理网关的电子邮件

3. 防火墙的防护类型

防护防火墙合并了所有代理防火墙的特性，同时增加了一些给定行为的可计算的规则。这些规则可以是合成的，并且必须经过错误检测。如果引入了错误，会危及公司计算资源的安全。

提示 更高级的防火墙现在具有 SQL*Net 的代理，这意味着当打开一个端口用于 SQL*Net 通信时，只有 SQL*Net 通信被允许通过这个端口。过去当这个端口打开用于 SQL*Net 通信时，计算机入侵者会开发这个端口作为攻击点并使用这个端口探查网络并使其曝光。

34.4 使用高级安全选项

Oracle 高级安全选项包含了一套工具，提供了在标准 Oracle 网络工具集中找不到的数据安全性与完整性级别。虽然一些特性在 Oracle 软件中可以具有，但主要的功能来自于 Oracle 网络与第三方安全性与验证产品的集成。

高级安全选项提供的功能为：

- 数据加密或校验和 —— 使用高级安全选项，可以保证传过网络数据的安全性，方法是当数据流在客户端与服务器间传递时，对数据流加密。也可以保护数据在客户端与服务器之间传递时不被修改，方法是使数据包与校验和包同时传递。
- 校验与单独注册 —— 高级安全选项允许将 Oracle 环境与你的结点上可以放置的其他校验或单独注册解决方案集成，Net8 支持 Kerberos、CyberSAFE、Identix TouchNetII、SecurID 的适配器，而且新的 Oracle 安全服务器将为 Oracle 资源提供验证服务，对安全

套接字层（SSL）使用数字验证 X.509V3。

- 与DCE环境集成 —— 使用高级安全选项，可以将 Oracle环境与资源同 OSF的分布式计算环境（DCE）集成。
- 远程拨号用户校验服务（RADIUS）协议 —— 这允许ASO使用所有遵守RADIUS的设备。
- 安全套接字层（SSL） —— SSL是因特网的标准之一，现在极易与 ASO集成。
- Oracle工具包 —— 帮助管理公共键基础结构（PKI）。

配置第三方验证适配器与 DCE适配器已超出了本书的范围。如果要获取这些项目的信息，参考《Oracle Advanced Security Option Administrator's Guide》与你试图安装的适配器的特定文档。

注意 有两个Oracle高级安全选项的版本：一个版本用于本土使用（美国与加拿大），可以使用现在能用的加密的最高级别。另一个版本适用于其他国家，称为出口使用版本。出口版本必须使用加密的最低级别，这是法律规定的。询问Oracle销售商以决定你的结点可以使用哪种相应的软件。

34.5 启用数据加密与校验

ASO使用RSA数据安全的RC4或数据加密标准（DES）加密数据。一个为Net8会话随机产生的键值提供了所有网络通信的安全性，这种加密适用于所有的 Net8网络，包括网关。

为启用数据流加密或数据检验和，必须在客户端与服务器的 sqlnet.ora文件中设置一些参数。在客户端与服务器端各有一套参数。注意如果数据库服务器也作为客户端使用，必须同时设置客户端与服务器端参数。可以按如下方式配置 sqlnet.ora文件：使用文本编辑器编辑这个文件，或使用网络管理或NET助手编辑缺省的资源文件。

SQLNET.ENCRYPTION_SERVER与SQLNET.ENCRYPTION_CLIENT参数指明一个连接是否需要加密，服务器与客户端的值一起进行评估以确定会话的配置，合法的值为：

- ACCEPTED —— 这台机器不加密这个会话，除非其他的机器要求对它加密。
- REJECTED —— 这个机器不加密，即使其他端点要求加密。如果其他的机器 REQUIRES一个加密会话，这两台机器将不能连接。
- REQUESTED —— 这台机器试图加密这个会话，但如果其他的机器不加密这台机器仍可连接。
- REQUIRED —— 这台机器只接受一个加密连接。

SQLNET.ENCRYPTION_TYPES_SERVER与SQLNET.ENCRYPTION_TYPES_CLIENT参数指明客户端与服务器机器可以使用的加密运算法则。如果指定了多于一条运算法则，机器将逐一测试，从第一个开始，到最后一个。在会话中使用的真正的运算法则是由客户端与服务器协商决定的。如果由于服务器与客户端没有一个共同的算法不能协商一条运算法则，连接失败。

合法的加密类型是：

- RC4_40~RSA RC4（40位键值大小） 本土的与国际的
- RC4_56~RSA RC4（56位键值大小） 只有本土的
- RC4_128~RSA RC4（128位键值大小） 只有本土的
- DES~标准DES（56位键值大小） 只有本土的
- DES40~DES40（40位键值大小） 本土的与国际的

■ 3DES (使用SSL的三倍DES)

为指定校验和行为, 使用SQLNET.CRYPTO_CHECKSUM_SERVER与SQLNET.CRYPTO_CHECKSUM_CLIENT参数。与加密参数相似, 这些参数接受 ACCEPTED、REJECTED、REQUESTED与REQUIRED作为合法值, 当协商连接时以相同的方式运转。

最后一套参数是: SQLNET.CRYPTO_CHECKSUM_TYPES_SERVER与SQLNET.CRYPTO_CHECKSUM_TYPES_CLIENT, 这些参数指明了产生校验和值的运算法则的类型。当前这些参数只支持MD5作为合法值。

最后, SQLNET.CRYPTO_SEED参数在客户端计算机上配置以种下用密码写的键值, 这是一个从10个到70个字符长的数字值。这个系列数越长越随机, 校验和键值越坚固。当使用加密或校验和时, 必须为这个参数指定一个值。

34.6 ASO支持的RADIUS协议设备

支持RADIUS协议意味着更多的授权厂商可以与 Oracle共同工作, 给予了实现安全计划更多的机会。Oracle服务器作为一个 RADIUS 客户并有权连接到RADIUS服务器 (参见图34-7)。客户使用 ASO注册到Oracle8i服务器, 数据库请求 RADIUS服务器对该请求的验证, RADIUS服务器接受或者拒绝这个请求, 该回答送给Oracle8i服务器, 然后它执行适当的行为。Oracle8i服务器真正执行了透明的代理验证。ASO可以提供客户化的Java类, 所以可以定制你的安全政策。RADIUS选项可以在因特网机制中实现, 方法是使用一个作为RADIUS客

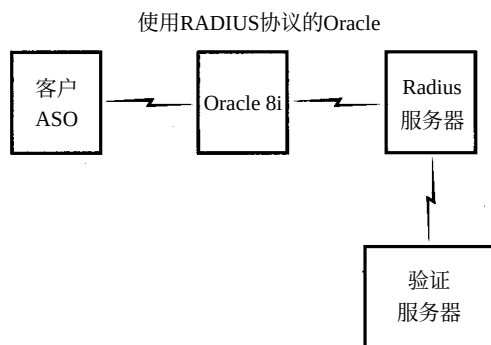


图34-7 使用RADIUS协议的Oracle

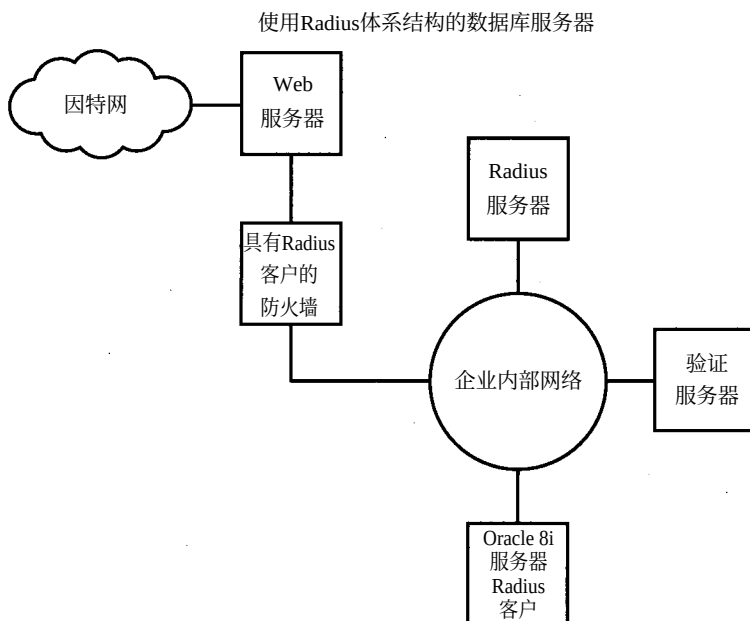


图34-8 使用RADIUS体系结构的数据库服务器

客户端的防火墙（参见图 34-8）。验证服务器可以是任何RADIUS协议意见服务器，如ActivCard、SecureId、kerberos、Biometrics等等。

ASO对RADIUS的高级特性包括质询—响应验证与记帐，并包含以下步骤：

- 1) RADIUS服务器向应用服务器质询。
 - 2) 质询传送到客户端。
 - 3) 质询呈现给最终用户。
 - 4) 用户然后对质询提供一个响应。
 - 5) 响应传送到RADIUS服务器。
 - 6) RADIUS服务器然后证实这个响应并发出拒绝或接受最终用户的消息。
- RADIUS记帐提供了创建审计报告的能力，审计报告可以用于安全性及资源的使用情况。

34.7 安全套接字层协议

安全套接字层（SSL）支持已合并入Oracle 8i数据库与ASO。SSL是因特网标准之一，现在更易于ASO集成。SSL提供了使用X.509数字证明的验证并加密网络通信。当一个因特网会话的所有组件都使用SSL时，就拥有了一个产业强度的安全解决方案。必要条件是因特网会话的所有层都支持SSL，包括客户层、Web服务器层与数据库层。SSL与Net8连接共同工作，并使用IIOP与企业级JavaBeans（EJB）。确保数据完整性由密码组处理。客户与服务器都有一套密码组，它们互相商议传输时使用哪个组，参见显示SSL体系结构的图34-9。

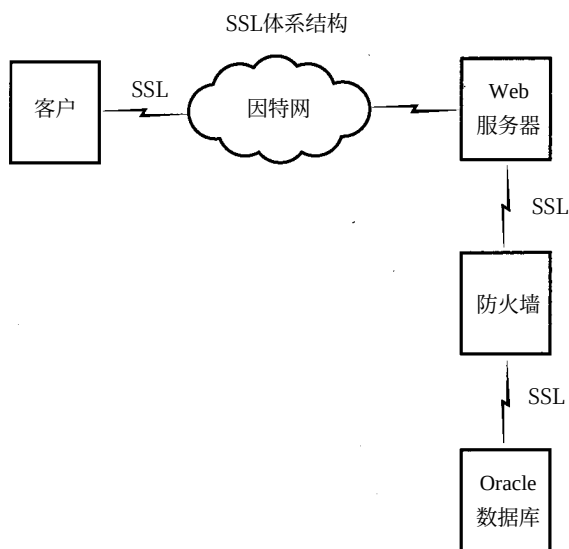


图34-9 SSL体系结构

34.8 Oracle工具包的支持

支持Oracle工具包帮助管理公共键基础结构（PKI）。Oracle工具包管理器允许用户与系统管理员管理包的内容。一个包可以含有用户证明及信任点数（证明权限的可信度）。当配置时，包可以放置在一个由口令保护的因特网路径下，这个口令可以不使用计算资源的方式与用户通信，这完成了只允许用户知道口令的安全方法。

一个工具包可以配置为具有对公司的普通信任点数，并且可以使用集中管理计划进行管理，使用工具包的过程如下：

- 1) 安装Oracle工具包。
- 2) 要求工具包的口令。
- 3) 口令用于创建一个公有键与一个私有键，私有键存储在工具包中。
- 4) Oracle包管理器请求证明（可以从Oracle或非Oracle CA产生）。
- 5) 证明下载并存储在工具包中。
- 6) SSL连接初始化。
- 7) 使用SSL检索用户的身份，服务器验证这个身份。

34.9 理解多线程服务器

缺省情况下，用户通过使用专有服务器进程连接到 Oracle数据库服务器。这意味着对于每个用户连接，一个关联进程替这个用户进程处理工作，如从数据文件将需要的数据载入到数据块缓冲区，并将数据库的查询结果返回给用户。这是提供连接到数据库的最快与最简单的方法。然而，在上百甚至上千个用户同时连接的情况下，维护这些专有服务器进程的开销代价过高，而且专有服务器进程消耗数据库服务器上相同量的资源，不管它们是活动的还是空闲的。在多数情况下，如果有大量用户连接到数据库但事实上很少从数据库存取数据，维护这些专有服务器进程占用的服务器资源很浪费。多线程服务器（MTS）可以节约这些开销。

以最简单的术语说，多线程服务器允许许多用户会话共享一组服务器进程，因而减少了支持一个大量用户所必要的资源开销。这个结构也允许降低这些服务器会话的全部空闲时间。例如，如果有100个同步用户连接，但平均在每一时刻每10个用户连接只有一个是活动的，可以通过分配10个服务器进程给用户使用来使资源极大化，这使10个服务器进程在任何时间都是活动的，而不是当使用专有服务器进程时，10个进程是活动的，而90个进程处于空闲状态。

34.9.1 多线程服务器结构

当使用MTS时，需要了解在体系结构中有一些不同之处。回想第6章当使用专有服务器进程连接到Oracle数据库时，监听器使用专有服务器进程连接到用户会话，管理用户对Oracle数据库的连接。在MTS环境下，监听器的行为稍有不同。不是分散并连接用户会话到一个专有服务器进程，它将用户连接传送给一个或多个调度器进程，这些调度器进程负责将用户进程命令放入请求队列，并从请求队列检索用户进程命令的结果，请示和响应队列都保留在SGA中。

共享服务器进程不直接与调度器或服务器进程交互，而是监控请求队列。当一个新的命令放入队列后，他们读这个命令，以下面的方式处理它：读数据块缓冲区中的适当的数据块并向数据库提交命令，然后将结果放入调度器响应队列中。所有的调度器将它们请求放入同一个请求队列，每个调度器也有它自己的响应队列。共享服务器进程确保用户命令的结果为执行命令的调度器放入正确的响应队列中。通过这种方法，共享服务器进程可以非常有效地一起工作，处理所有的用户会话请求，调度器只需要检索并操作它们所维护的用户会话的数据。

多线程服务器的使用也改变了对SGA内存区的分配。因为没有专用服务器进程，用户的

会话数据与游标状态信息存储在 SGA中, 而不是PGA中。由于这个原因, SGA应该相应地进行调整。注意这并不是运行 MTS的附加代价, 而是调整在内存的什么地方存放数据。

34.9.2 配置多线程服务器

多线程服务器主要通过每个数据库的 init.ora文件中的参数配置, 以下是配置使能 MTS的参数。

MTS_SERVICE 与调度器进程相关的调度器进程服务的名字。监听器将对一个服务的请求传送到相应的基于这个参数值的调度器。通常将这个值设为数据库的名字 (db_name init.ora参数的值)。

MTS_SERVERS 指定当实例启动时共享服务器进程的数量。

MTS_MAX_SERVERS 指定一次运行的共享服务器进程的最大数量。共享服务器进程按需要被分配和消灭, 但它们的数量不能超过这个数, 或低于 MTS_SERVERS的值。

MTS_DISPATCHERS 在实例启动时定义协议与分配的调度器的数量。指定使用不同协议的多个调度器, 需要分别使用 MTS_DISPATCHERS指明每个协议。

MTS_MAX_DISPATCHERS 指定一次运行的调度器最大数量。调度器进程基于系统负载被分配和消灭。

MTS_LISTENER_ADDRESS 调度器进程监听的地址。与监听器地址相同。

一个示例init.ora文件的MTS参数如清单34-1所示。

清单34-1 INIT.ORA

```
MTS_SERVICE = PROD01          # Database name is PROD01.
MTS_SERVERS = 3                # Start 3 shared server processes at
instance start.
MTS_MAX_SERVERS = 10           # Never start more than 10 shared server processes.
MTS_DISPATCHERS = "tcp,3"      # Start 3 dispatchers that use the TCP/IP protocol.
MTS_DISPATCHERS = "spx,1"      # Start 1 dispatcher that uses SPX.
MTS_MAX_DISPATCHERS = 10       # Never start more than 10 dispatcher processes.
MTS_LISTENER_ADDRESS = "(address=(protocol=tcp)(address=dbserver)(port=1521))"
MTS_LISTENER_ADDRESS = "(address=(protocol=spx)(service=novellserver))"
```

除配置调度器与共享服务器进程之外, 也可以控制客户端的 MTS行为。因为某些工作不能使用共享服务器进程运行 (如直接加载卸出与 SQL*Loader执行), 有些工具使用专有服务器进程运行得更好 (如批量或过程化处理密集工作), 需要强制使用专有服务器。可以通过将 sqlnet.ora参数USE_DEDICATED_SERVER设为TRUE, 在客户端全局使用专有服务器进程。为一个独立的 TNS别名指定使用一个专有服务器, 将 tnsnames.ora文件的 SERVER参数设为 DEDICATED。例如, 清单34-2指定了两个TNS别名连接到同一个数据库。然而, 第二个别名使用专有服务器, 而第一个使用共享服务器。

清单34-2 连接到相同数据库的两个TNS别名

```
PROD_MTS =
(DESCRIPTION =
  (ADDRESS_LIST =
    (ADDRESS =
      (COMMUNITY = tcp.world)
      (PROTOCOL = tcp)
      (HOST = dbprod)
      (PORT = 1521)
```



```

    )
  )
  (CONNECT_DATA = (SID = PROD01)
)
)

PROD_BATCH =
(DESCRIPTION =
  (ADDRESS_LIST =
    (ADDRESS =
      (COMMUNITY = tcp.world)
      (PROTOCOL = tcp)
      (HOST = dbprod)
      (PORT = 1521)
      (SERVER = DEDICATED)
    )
  )
  (CONNECT_DATA = (SID = PROD01)
)
)
)

```

34.9.3 管理多线程服务器

在通常的操作中，Oracle服务器按需要启动和关闭共享服务器与调度器进程。这基于进程上所加的负载。然而，可能因为性能问题有必要手工管理 MTS操作或监控。由于这个原因，有一些可用的数据字典视图展示与MTS工作有关的信息。V\$DISPATCHERS与V\$SHARED_SERVERS显示了调度器与共享服务器进程运行时的信息。V\$MTS显示了所有的统计数字，如MTS连接的最大数量、服务器启动的数量、服务器杀死的数量及服务器的最高水位标记；V\$QUEUE显示请求队列与响应队列的信息；最后 V\$CIRCUIT显示使用调度器或服务器的用户连接信息。这些性能视图允许检查你的 MTS配置，必要时进行调整或改进。

可以使用适当的 ALTER SYSTEM命令在实例运行时调整共享服务器或调度器进程的数量。可以使用适当的命令增加或减少进程的数量。注意如果试图中断服务器进程或调度器，Oracle服务器只有在使用它们的用户会话关闭后才会中断它们。

下例显示这些命令是怎样使用的：

```
ALTER SYSTEM SET MTS_DISPATCHERS = 'spx,10';
```

它在运行10个SPX调度器时启动或停止SPX调度器

```
ALTER SYSTEM SET MTS_SERVERS = 5;
```

它在运行5个共享服务器进程时启动或停止共享服务器进程。

34.10 使用Oracle连接管理器

Oracle连接管理器是一个新的 Net8选项，由Oracle8的企业版提供。它提供了以下方面的增强服务：处理连接池、存取控制及多协议支持。在具有成百上千同时连接的环境中它是完美的，并提供了与多线程服务器相同的益处。

使用网络管理器或Net8助手配置连接管理器。还可以通过直接编辑 cman.ora文件配置连接管理器。

34.10.1 配置连接多路技术

多路技术连接（或集中化连接，如 Oracle文档中所提到的）是这样一种进程：在一个单一

的网络连接上路由多个离散的连接。当连接是多路复用的时，服务器只消耗很少的资源，因为多路复用连接只使用一个单一的连接，可以在配置使用 MTS 的环境中使用集中化管理。

在使用动态发现配置的 Oracle 名字环境中，当连接管理器在线时，集中化自动发生。要手工使能集中化，必须将连接管理器正监听的地址与端口放入 tnsnames.ora 文件中。

为了设定连接管理器监听什么，在 cman.ora 中加入下面一行：

```
cman=(address=(protocol=tcp)(host=HOSTNAME)(port=PORT))
```

其中 HOSTNAME 是连接管理器正在运行的计算机的主机名，PORT 是连接管理器应该监听的端口。缺省情况下，连接管理器在端口 1600 监听。

为配置客户端使用连接管理器池，在地址列表中指定连接管理器信息并将 SOURCE_ROUTE 参数设为 yes，如：

```
(description =  
  (address_list =  
    (address = (protocol=tcp)(host=conman)(port=1600))  
    (address = (protocol=tcp)(host=dbserver)(port=1580))  
  )  
  (connect_data=(sid=db1))  
  (source_route = yes))
```

注意 SOURCE_ROUTE 参数指出客户端必须通过多个目的地以到达最终目的地，使用连接管理器表明客户端在游历到真正的数据库之前，必须首先到达连接管理器所在的机器。

34.10.2 配置多协议支持

连接管理器的多协议支持代替了 Oracle7 的多协议交换。为使连接管理器可以处理多个协议，在连接管理器运行的机器上安装所有的协议。连接管理器依赖客户端 tnsnames.ora 文件的配置自动路由查询。源路由地址指明了在 tnsnames.ora 文件中遍历的协议是所有需要使能多协议连接的协议。例如：

```
(description =  
  (address_list =  
    (address = (protocol = spx)(service=conmansrv))  
    (address = (protocol = tcp)(host = dbserver)(port = 1580))  
  )  
  (connect_data = (sid = db))  
  (source_route = yes))
```

34.11 实例学习

一个公司刚刚开始使用因特网，所以他们安装了一台连接到因特网的 Web 服务器，并允许 Web 服务器通过防火墙与数据库服务器通信。这是一个独立的网络，没有连接到企业内部网（参见图 34-10）。这为管理层提出了一些真正的问题，因为数据库没有连接到内部网络，对数据库的远程管理变得十分困难。备份与恢复数据库并不适合他们当前的公司模型，不得不开发附加的硬件与过程。安全小组与数据库小组举行了一个会议，决定将数据库连接到内部网络是否是可行的。安全小组迅速指出如果计算机入侵者危及了内部网络的安全，他们会对公司造成极大的破坏，所以认为风险是相当高的。在进一步审阅图表后，讨论了一系列替代意见。替代意见选择在数据库服务器服务的因特网应用与公司的内部网络之间，需要一个

附加的防火墙，这个防火墙只允许 1521 端口存取内部网络，并且只允许 SQL*Net 通信（参见图 34-11）。数字证明也将使用以验证 SQL*Net 通信是从公司内部的数据库服务器产生的，而不是计算机入侵者产生的。这个解决方案允许 DBA 以有效的方法管理数据库，而不会危及公司的安全。

实例学习

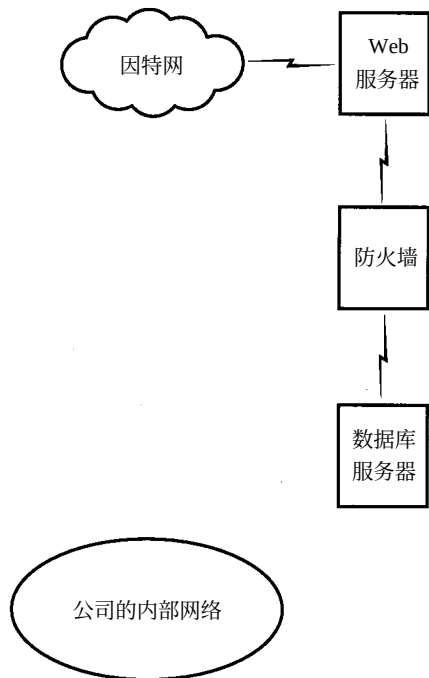


图34-10 实例学习

实例学习 —— 替代实现

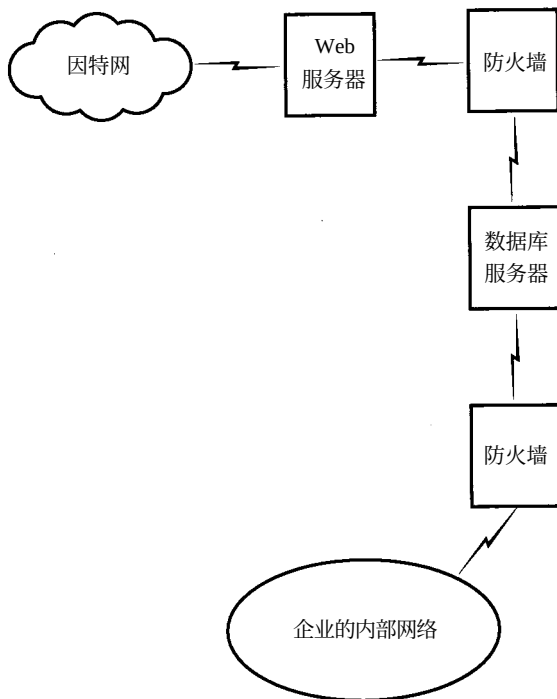


图34-11 实例学习 —— 替代实现

34.12 小结

高级安全选项（ASO）提供了一套非常完整的工具保护你的数据库。安全性总是以一个计划开始，并包括达到这个计划结果的许多检验与平衡。ASO 在与安全性工业标准集成方面的能力是非常强的。因特网改变了每一件事；现在具有计算机入侵者打击公司的计算资源每个核心的可能性，你给了他们一个大门进入你的电子堡垒。商业需求要求你允许因特网存取以简化商务，但基于任何给定因特网系统的风险分析必须成功。ASO 给了我们工具，但现在必须使用它们建立一个安全金字塔保护公司的基础结构。

第八部分 Oracle应用服务器

第35章 Oracle应用服务器概述

本章要点：

- Oracle应用服务器介绍
- 理解因特网计算
- 理解Oracle应用服务器
- 使用Oracle应用服务器提供基本服务

35.1 Oracle应用服务器介绍

在1995~1996年间因特网闯入了许多人的意识之中，它作为新的、与以往任何计算革新不同的事务被媒体与计算机用户捕捉到。

因特网也为计算环境增加了一些新的标准，通过因特网与 TCP/IP提供的近乎普遍的地址模式，因特网服务提供了标准邮件及其他的协议，通过在 HTML中的一个简易的、跨平台的超文本显示标准，因特网提供了一种方法集成化遍布世界的不同的计算机。

Oracle在提供数据库技术方面处于领导地位已超过 15年，因为因特网是一个用于交换数据的庞大的网络，Oracle产品应该在通过因特网及通过基于因特网的应用检索与分发数据方面扮演重要角色。

Oracle应用服务器（Oracle Application Server, OAS）是目前Oracle使用因特网的基础。然而，不只有一种方法使用因特网，OAS是Oracle“因特网计算”机制的关键部分，反过来，因特网是实现完全分布式应用系统的草图。基于因特网的应用是一种分布式应用的类型，所以OAS在理论上也适于基于WEB的应用。本章的其余部分将在一个高级别上介绍 Oracle应用服务器，帮助创建一个在因特网或一个内部网上实现的系统，并使用 Oracle应用服务器的服务及Oracle8 RDBMS。

35.2 理解因特网计算

因特网计算是一个框架，可以使用它创建一个开放的分布式应用系统。为理解因特网计算，查看一下过去几年中计算基础结构的演变很有帮助。

十年前，事实上所有的应用都在一个单独的单块的机器上开发，所有的用户使用哑终端连接到这台机器上。运行在这台机器上的不同的软件组件用于实现系统，如数据库管理系统及应用程序，但所有的组件都通过共享资源相互通信，如通过硬盘或内存，最坏的情况甚至是通过内部通信总线。

在个人计算机开始进入计算领域之后，客户/服务器模型开始用于应用系统。客户/服务器计算将PC作为客户端，具有足够的能力支持用户需要的图形用户界面及应用逻辑的执行，通

过局域网连接到数据库服务器。开发者花费了许多年，他们使用的工具相应地支持客户/服务器体系结构并减少了由网络带宽限制产生的瓶颈冲突。

客户/服务器计算体系结构被用户广泛地喜爱，但将计算资源分布在多台客户机上，对应用开发者与系统员产生了它自己的一套问题。每当应用改变后，新的客户端组件不得不分布到所有的客户机上，不断增长的应用逻辑的复杂化开始拉紧 PC 客户的计算能力的界限。为解决这些问题，发展了一个三层计算模型，其中应用逻辑运行在一个服务器机上，这个服务器机作为客户端与数据库服务器的中间物。

因特网计算将三层模型扩展为多层模型，它提供了一个标准接口，许多组件都可以使用这个接口，它是功能性自包含单位，可以作为客户组件、应用逻辑组件或数据组件。因特网计算还包括一个标准化的通信协议，因而一个应用中不同的插件可以相互通信。基本的如 Oracle 应用服务器所实现的因特网计算结构如图 35-1 所示。因特网计算也适用于需要存取不同位置的应用，一个 Web 客户，以浏览器的形式可以使用这个框架与其他应用组件交互，相同的组件可以在一个更传统的客户/服务器体系结构中作为插件使用，也可以通过哑终端连接到一个单块主机。通过清楚地定义客户、逻辑与数据插件交互的方法，因特网计算在创建与开发应用系统中允许真正的无限制的灵活性。

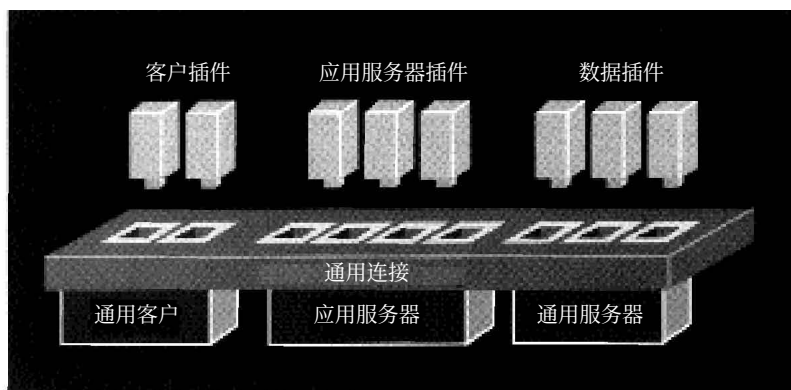


图35-1 因特网体系结构为分布式计算提供了一个框架

35.3 理解Oracle应用服务器

Oracle应用服务器在因特网计算中扮演了一个关键性的角色，并作为组件服务管理与组件或插件之间通信的焦点。

应用是商务逻辑的集合，可以由多个插件服务器组成。插件服务器是一个代码模型，通过标准接口与 OAS 交互。OAS 的基本功能是管理插件的交互，或更确切地说，是插件实例之间的交互。为完成这项任务，OAS 管理应用逻辑插件的创建，客户插件与应用逻辑插件之间的通信，及应用逻辑插件之间的插件中通信。OAS 还通过 CORBA 2.0 兼容的对象请求代理 (Object Request Broker, ORB) 提供了强壮应用所必需的组件的基本服务。

Oracle应用服务器由以下三层组成：

- HTTP 监听层，它通过标准因特网协议处理客户端与应用服务器层之间的通信。
- Oracle 应用服务器层，管理插件服务器进程与插件实例的创建；在独立插件的多个实

例之间均衡负载；插件服务，如传输服务、插件之间通信服务、永久存储服务及验证服务。

■ 应用层，在此使用特定的插件实现特定的应用功能。

这种层体系结构给予 OAS 两个基本的优点。首先，它允许每个组件被设计为最好将重点放在特定功能的需要上，而不是试图处理一台服务器上的所有任务。对实例而言，HTTP 监听器必须准备好接收所有的 HTTP 信息，所以它应该是一个很小的应用，在必要时作出响应。另一方面，ORB 将三层连接到一起，潜在地必须处理不同类型的请求，所以它必须有能力管理多重任务，同时需要一个更坚固的应用。

OAS 可以支持独立插件的多个实例，这使使用这些插件的应用具有高度可伸缩性。通过在需要时增加附加资源而不影响组件的其余部分的能力实现这个伸缩性。对实例而言，必须具有多个监听器处理高容量的通信，或在一个特定插件功能被沉重使用时产生附加的插件服务器与实例。

第二个优点来自于具有一个在不同组件之间定义明确的应用程序接口（API），这个接口使 OAS 成为一个开放系统，可以在其上增加你自己的客户组件以创建你的系统。例如，可以使用其他的 HTTP 监听器代替 Oracle Web 监听器，并仍使用 Oracle 应用服务器的其他组件。更为重要的是，OAS 的开放体系结构允许以常用编程语言书写自己的插件以支持任何开发环境，并传递在 Web 应用中需要的任何类型的功能（见图 35-2）。

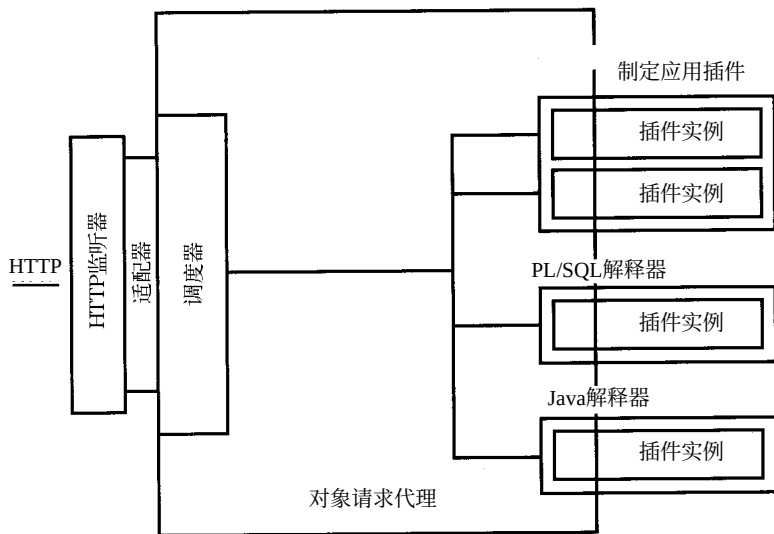


图35-2 Oracle应用服务器体系结构将任务处理分配给Web请求引擎

35.3.1 HTTP监听层

HTTP 监听层，正如名字所暗示的，监听到达服务器主机 IP 地址的 HTTP 请求。当 HTTP 监听器得到一个请求时，它试图将 URL 翻译为物理实体，如 HTML 文件或 CGI 脚本。如果它不能翻译，它将请求传送给调度器，调度器与 ORB 一起将监听器捆绑入 Oracle 应用服务器层。监听器的一部分是虚拟路径管理器。如果虚拟路径管理器接收到这个请求，它很可能对应到一个应用插件。应用逻辑被执行并将结果返回给监听器以供浏览器取回。

35.3.2 Oracle应用服务器层

Oracle应用服务器层是将所有的事务及基础服务，如验证、注册、故障恢复、事务控制及负载平衡的位置连接到一起的胶。与 ORB一起，它允许用于分布式系统，在分布式系统中，应用、监听器与数据可以被放置在不同的物理机器上，但仍可以从一个中央接口配置与管理。

35.3.3 应用层

在Oracle应用服务器层下面是什么？当有一个特定应用的请求到来时，调度器将请求传送给适当的插件服务器。每个插件服务器具有一个多线程，每个插件实例都有它自己的线程。这节省了为每一个查询启动一个进程的开销。一个应用可以拥有不止一个插件服务器（只要它们是相同类型的），并指定这个服务的最大与最小线程数量，所以调度器在多个服务之间进行动态负载均衡。

每个插件服务器有它自己的执行机并使用一个共享库。对象请求代理通过三个基本的 API 调用与组件通信，这三个基本应用用于初始化一个服务，关闭这个服务及将请求传递给这个服务。

35.3.4 插件

Oracle Web应用服务器具有以下六个预定义插件：

- PL/SQL插件，能调用存储PL/SQL包与过程。
- Jweb插件，提供了一个运行时的、在服务器上的 Java执行环境。
- Perl插件，调用Perl脚本，消除了为每个请求都启动 Perl翻译器的需要。
- ODBC插件，提供使用开放式数据库互连接口存取各类数据。
- C Web插件（以前是WRB API，在Windows NT中简称为C插件），允许用C语言书写自己的服务器端应用。
- LiveHTML插件，与服务器端的插件共同工作

PL/SQL、JWeb、C Web插件含有附加的功能，扩展了插件执行外部进程的能力，如将HTML写回浏览器。

Oracle也给出了Oracle开发服务器，一个使用 Oracle开发器开发的应用的 Web插件接口。开发服务器允许任何 Oracle开发器或Oracle设计器创建的应用在 Web上使用Java应用程序重新编译与开发，并作为客户端与运行在 Web服务器的应用自身的用户接口。

OAS使用一个开放式 API，所以可以用C语言设计自己的插件。可以创建系统插件，系统插件执行一个预定义的功能，或程序插件，程序插件可以用于解释运行时的应用。第三方开发商也可以为特定的目标开发插件，任何新的插件可以轻易地集成到开放式 OAS环境中，方法是将插件注册到OAS配置文件中。

35.3.5 JCORBA与企业级Java Beans

在Oracle应用服务器 4.0中，你不仅具有书写基于插件应用的能力，而且可以书写 JCORBA对象——用Java书写的CORBA对象。JCORBA对象可以通过 IIOP存取。与任何存取 CORBA对象的方法相同，并可以被任何适当的客户存取，如浏览器甚至是基于插件的应用。

也支持企业级Java Beans, 允许更容易地重新使用自己的Java代码或从其他厂商购买Java代码以节省自己的开发时间。

35.4 使用Oracle应用服务器提供基本服务

OAS提供了一些任何插件都可以使用的基本服务。有四个 Oracle应用服务器提供的基本服务类型：

- 事务服务。
- 插件内交换服务。
- 永久存储服务。
- 验证服务。

此外, Oracle Web应用服务器还提供了日志服务用于记录对 Oracle应用服务器的请求及日志分析, 帮助理解服务器日志。

35.4.1 事务服务

Oracle应用服务器提供了对所有插件的事务服务。事务是明确定义的工作单位。如果事务提交, 所有事务实现的修改应用于有关的数据存储区。如果事务回滚, 这意味着数据返回到事务开始之前的状态, 数据存储区保持与事务开始之前相同的状态。因为 Web是一个天生没有国家的媒体, 每个网页内部或它自己通常都是一个事务。通过提供事务服务, Oracle应用服务器允许克服无连接HTTP通信面临的困难, 方法是使事务能够生成多个请求。

35.4.2 插件之间交换服务

Oracle应用服务器提供了插件之间交互服务, 允许不同的插件之间相互通信。

在Oracle应用服务器3.0版本中, 插件之间的内部通信是由内部通信协议处理的, 这个内部通信协议遵守通用对象请求代理体系结构(Common Object Request Broker Architecture), 也称为CORBA。CORBA标准是一个开放式协议, 有许多各种类型的硬件与软件厂商都支持CORBA。

Oracle应用服务器4.0版本为插件之间交互扩展了CORBA标准的使用, 所以通过使用OAS的插件, 单独开发的CORBA组件可以透明地与插件交互。

35.4.3 永久存储服务

OAS包括一套应用编程接口, 允许开发者读写数据对象、创建数据对象并删除数据对象及他们的数据。这些API可以用于将数据写入Oracle8数据库或写入服务器平台上的本地数据文件中。

这些API建立在一个模式上, 这个模式含有的属性包括内容类型、作者及创建日期。永久存储服务允许所有的插件通过一个普通接口进行操作, 无论何时, 数据必须存储在硬盘上或为将来的引用存储在一个应用中。

35.4.4 验证服务

Oracle应用服务器4.0版本给开发者多种扩展验证模式, 可以用于他们的应用, 包括基本的、摘要的、域的及数据库验证。而且, 4.0版本还允许通过LDAP服务器进行验证。

验证服务给了开发者以最适合他们的特殊应用需要的方式实现安全性的灵活性。

第36章 Oracle应用服务器组件

本章要点：

分析HTTP监听层
分析应用服务器层
分析应用层

36.1 分析HTTP监听层

Oracle应用服务器（OAS）使用一个Web监听器等待服务请求的到来。通常，一个监听器会设定一个用作电子路标的地址，告诉所有数据包旅客“我正在营业”。当然，监听器在营业之前必须先告诉 Oracle应用服务器它的地址。当 OAS将这个地址注册后，可以建立一条从监听器来的路。通过这种方法，当一个客户发出一个请求时，监控器可以将信息路由到应用服务器，应用服务器会执行适当的服务（参见图 36-1）。

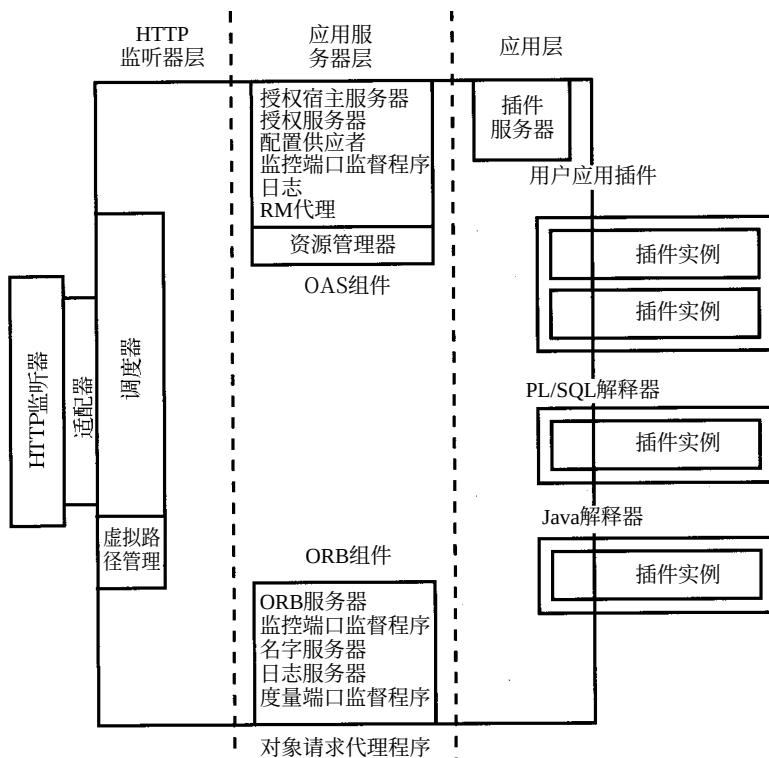


图36-1 Oracle应用服务器是由三个一起工作的概念上的层组成的

36.1.1 获得更详细的细节

HTTP监听器处理一个或多个同步请求，在这种情况下，远程服务器或浏览器作为客户。

为在远程服务器上执行一个请求，必须确保对象请示代理（Object Request Broker, ORB）首先已被实例化，否则远程监听器将不存在，客户请求不能被远程执行。

幸好ORB在监控器运行期间总是处于运行状态，因为监听器不能启动，除非 ORB已运行，并且如果监听器正在运行，ORB也不能被关闭，这是因为监听器必须将它自己注册到调度器，调度器是Oracle应用服务器层的一个组件，它通过与 ORB挂钩，将监听器连接到应用服务器。

调度器的一个重要部分是虚拟路径管理器 (Virtual Path Manager, VPM)。当监听器接收到一个请求时，它的形式是“虚拟路径”，即URL的后一部分。VPM负责将这个虚拟路径翻译为物理实体。如果这个物理实体是一个基于插件的应用，请求通过 ORB路由，ORB协同它的执行并将结果返回给监听器。在静态文件与 CGI脚本的情况下，请求路由回监听器，监听器创建请求自身并将结果返回给客户端。

通过结点管理器 (Node Manager) 创建并管理监听器，结点管理器是一个基于 Web的接口，允许管理Oracle应用服务器的所有部分。结点管理器由安装器自动启动，但也可以使用命令行工具：`owsctl start -nodemgr`稍后再启动它：

要获取结点管理器的详细信息，参见第 37章。

36.1.2 理解HTTP监听器的体系结构

虽然在当今的关键任务客户 /服务器环境下多线程处理是必须的，但一些 IS技术人员过度使用并滥用了这项技术。他们疏忽理解的是——也很容易做到的是每个进程都需要分配内存与资源。没有适当的内存与资源分配，当打开更多的线程时，系统性能会降低，特别是当进程同步处理请求时。因而，需要在实现多线程体系结构时仔细考虑一下。

Oracle设计HTTP监听器时考虑了这个问题，监听器的任务只是接收到来的客户请求，将它们路由到应用服务器层，并将结果返回给浏览器。因而，只需要两个进程——一个监听、一个谈话。HTTP监听器也异步处理请求，意思是当它接收到一个请求时，在以后的处理中监听器不再需要这个客户。

36.1.3 文件的内存映射

如果文件不能永久地缓存在内存中，HTTP监听器使用一个动态内存映射模式，其中多个客户连接可以存取共同的文件。结果是，由于避免了重复硬盘读，文件存取速度加快。并且因为文件映射到内存中，操作系统可以搜索这个文件的下一个内存块，而同时将另一个文件传送到客户端。这种性能增强机制在只有一个连接的情况下仍工作得很好。

36.1.4 路径映射

面向对象圈中的一个重要的概念是封装或数据捆绑。在这里，不需要关心对象做了什么，只需要它按预期的方式执行。与此相似，HTTP监听器在某种意义上封装了它是怎样存储 URL路径名的。也就是说，当到达一个 Web站点时，每次都看到相同的 URL，但随着时间的推移，它可能对应不同的物理文件。OAS管理员负责正确地映射 URL路径，使用户可以看到机器文件系统上的真正的物理路径。

36.1.5 解析域名

域名服务器解析 (Domain name server resolution)保证了客户端用户的请求发送到正确的网

络地址/URL。当这发生时，HTTP服务器记录下客户机的IP地址，通过它路由对HTML格式的请求。

域名服务（DNS）服务器用作客户的IP地址与ASCII主机名转化的参考。在这个解决方案处理中，可以包括多个主机，即在等候地址解析之前，可能要花费一段时间。

HTTP监听器提供给Oracle应用服务器系统管理员四种处理DNS任务的方法：

- 连接时解析IP地址。
- 需要时解析IP地址。
- 当CGI脚本或应用插件需要时解析IP地址。
- 从不解析IP地址。

使用DNS解析方案，这个进程的结果缓存在内存中，虽然多个连接可能在同一时间窗发生，也有可能来自同一客户机。由于性能的原因，DNS解析方案的缺省配置是从不解析IP地址。

36.1.6 HTTP监听器配置参数

HTTP监听器的配置参数存储在配置文件中，当监听器初始启动时，读这个文件。可以使用命令行工具启动监听器，这个工具在Oracle应用服务器的不同版本中是不同的，4.0版本也不例外。使用如下命令从命令行启动监听器：

```
owsctl start -l myListener
```

结点管理器，可以被任何窗体式Web浏览器存取，消除了手工编辑监听器配置文件的需要，在多数情况下，为每一个参数提供了解释性帮助文本。这一节文本完整地记录下配置文件中的参数，用于系统管理员想手工编辑文件的情况。

Web监听器配置文件分为多节，以括号内的节名开始，例如 [NetInfo]。使用名字=值对逐一配置参数，配置参数在等号左边（lvalue），值在右边（rvalue）。例如，下面就是一个典型的配置文件：

```

;
[NetInfo]
MaxConnectCount    = 338
;   The maximum number of connections. 338 is an OS limitation
;   based on the number of filehandles a single process can handle.
DNSResolution = NEVER
ServerPID = /u01/app/oracle/admin/ows/website40/httpd_csac07/myweb/myweb.pid
;
[LogParams]
TimeStyle = GMT
AdminFile = /u01/app/oracle/admin/ows/website40/httpd_csac07/myweb/myweb.err
;
[Server]
InitialFile        = wwwIndex.html
;   The "default" file accessed when no file is specified,
;   ie "http://www.csac.com/" would return
;   http://www.csac.com/wwwIndex.html. Note that while
;   wwwIndex.html is the default for OAS,
;   most sites use "index.html" as their default.
DefaultMIMETYPE    = application/octet-stream
DefaultCharset     = iso-8859-1
PreferredLanguage  = en
ImageMap           = map
UseDirIndexing     = FALSE
CGITimeout         = 900
;

```

```
[SecureInfo]
UserID          = root
GroupID         = dba
; Only applicable on UNIX systems. Note that if you are
; listening on a port under 1024, the process must run as root.
;
[DirMaps]
;Physical Directory      Flag      Virtual Path shown in URL
; The flag specifies whether content is Non-CGI, CGI, or Win-CGI,
; and whether the designation is Recursive or Non-Recursive.
/u01/app/oracle/product/8.0.4/ows/4.0/doc/      NR      /
/u01/app/oracle/product/8.0.4/ows/4.0/admin/doc/ NR      /ows-adoc/
/u01/app/oracle/product/8.0.4/ows/4.0/admin/img/ NR      /ows-aimg/
/u01/app/oracle/product/8.0.4/ows/4.0/bin/       CN      /ows-bin/
/u01/app/oracle/product/8.0.4/ows/4.0/doc/       NR      /ows-doc/
/u01/app/oracle/product/8.0.4/ows/4.0/admin/img/ NR      /ows-img/
/u01/app/oracle/product/8.0.4/ows/4.0/classes/   NR      /ows-vbclasses/
/u08/htdocs/                                     NR      /soaig/
;
en          iso-8859-1          eng
en          unicode-1-1        engU          uc
fr-CA       iso-8859-1          frc
fr-FR       iso-8859-1          fr
jp-JP       iso-2022-jp        jp
jp-JP       unicode-1-1-utf-8  jpU
; Used by NLS functions.
;
[MIMETypes]
text/html    htm              html
image/jpeg   jpg              jpeg
image/gif    gif
appl/text    doc
text/plain   txt              ksh          lst
application/pdf pdf
application/postscript ps
model/vrml   wrl              vrml
video/mpeg   mpeg             mpg          mpe
;
[Encodings]
compress     Z
gzip         gz
;
[DynApps]
/u01/app/oracle/product/8.0.4/ows/4.0/lib/ndwfss.so oracle_adp_init
; Directs the listener to the Adapter.
;
[MultiPort]
; IP address      TCP Port      Encryption      Host Name      Base Directory
; LogInfo Directory Client Authentication
ANY              80              NORM            csac07.csac.com /
➔/u01/app/oracle/admin/ows/website40/httpd_csac07/myListener/ NONE
ANY              443              SSL             csac07.csac.com /
➔/u01/app/oracle/admin/ows/website40/httpd_csac07/myListener/ NONE
; A single listener can listen on multiple ports and multiple IP addresses,
; and can direct logging for each IP/port combination to a different
➔directory.
```

36.1.7 第三方HTTP监听器

因为HTTP监听器与ORB对话使用调度器作为中介，所以在 Oracle HTTP监听器中没有限制。只要有适当的适配器，就可以使用任何 HTTP监听器（参见图36-2）。

在Oracle应用服务器中使用第三方 HTTP监听器，需要“注册”它。从 OAS欢迎页选择“OAS Utilities”，并展开左边的主菜单，从那里再展开“External Listener”结点及“Register”结点以找到合适的适配器。点击相应的适配器并依照下面的菜单注册你的外部监听器。

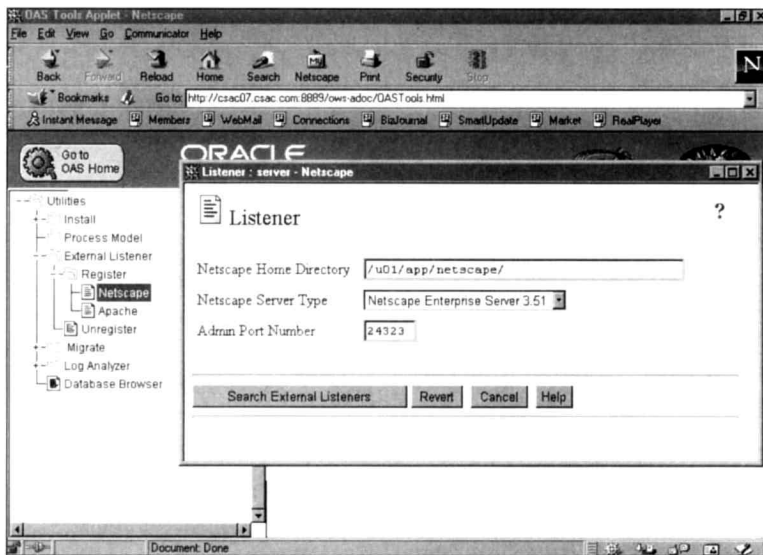


图36-2 注册一个外部监听器

注意 4.0.7版本具有由于Netscape与Apache（UNIX上）与微软的因特网信息服务器（在Windows NT上）的适配卡。

36.2 分析应用服务器层

HTTP成功地接收到客户端送来的信息后，将权力传递给应用服务器层。在所有的 Oracle应用服务器组件中，应用服务器层（以前称为 Web请求代理或WRB）是最关键的，因为监听器与应用插件通过它代理他们的信息。

WRB合并了多线程与多进程体系结构，导致机器生产力的增长，几乎可以同步处理多个请求。WRB也支持异步独立进程，所以客户与服务器资源不必被迫等待返回结果集或其他信息。

当调度器通知ORB它需要一个新的插件执行实例时，资源管理器组件（Resource Manager, RM）检查所有可用的插件实例。如果没有可用的，它指示一个现有的插件服务器创建一个实例。如果所有现有的插件服务器都具有了最大数量的实例，RM指示插件制造厂创建一个新的插件服务器。如果插件服务器已到最大数，请求被排队。

注意 对象间信息通信遵循常用对象请求代理体系结构（CORBA）。在核心中，支持CORBA模型的系统具有担负以下角色的对象：客户、代理或服务器，每个都帮助从终端用户机将信息传递到本地或远程服务器。

36.2.1 WRB调度器

已经介绍了这么多信息，现在是更深一点钻研的时间了。在 Web监听器向WRB发送一条

信息后，WRB的调度器决定客户端请求的对象类型。这个调度器详细地查阅 WRB配置文件，这个文件帮助调度器决定虚拟路径与 WRB服务的关系。如果调度器找不到适合客户请求的关系，它将请求返回给Web监听器，这使信息生命循环进行下去。

当调度器收到基于插件应用的请求时，请求需要被路由到一个插件实例，插件实例作为插件用品的一个线程运行。如果调度器知道一个可用的插件实例，它使用 ORB向这个实例发送请求。如果调度器不知道可用的实例，它与应用服务器层联系并要求一个新的实例。参考物返回给调度器，并存储在它的对象缓存中以便当前与将来之用。

使用这种体系结构可以很自然地在服务对象实例之间动态分布进程负载，这有助于最小化网络连接与最优化 Oracle应用服务器性能。作为 Oracle应用服务器的系统管理员，当使用多进程时（或对 NT用户是服务实例），必须时刻关注潜在的资源消耗与性能退化。可以为每个应用指定最小与最大的插件服务（进程）数量，也可以指明这些内部的最小与最大插件实例（线程）数量。

注意 并不是所有的应用都可以是多线程的。例如，认为 PL/SQL 插件是有“疆界的”，而不是“无疆界的”，所以每个插件服务器最大只能具有一个插件实例。

在内核中，每个对象都有一个接口提供了将“产品”传递给客户的基本方法（函数）。在面向对象的术语中，这种服务职责的接受与由此引起的服务传递称为“捆绑契约”（binding contract），而且，正如一些客户/服务器专家所指出的，“对象的位置应该对客户与对象实现是透明的。”

在这种情况下，OAS与ORB模型一同定义，事实上使用因特网通信的 CORBA。通过使用 CORBA，ORB背后的主要思想是：一个对象，作为代表一些客户对象的代理，处理远程服务的请求，然后 ORB通过网络或在你的机器上提供给客户透明地调用服务对象上的对象方法的能力。这意味着不必关心提供你所需要服务的对象的操作系统或平台。这将它自己带入数据捆绑或封装及分布式安装，在分布式安装中，功能可以被扩展到许多机器或结点上。

36.2.2 IPC支持

与传输无关的 ORB 协议处理所有的进程间通信（IPC）。Oracle 应用服务器支持标准 IPC 机制，IPC 是 UNIX 平台上客户/服务器体系结构的核心与灵魂。然而，IPC 并不是最简单的客户/服务器工具套，特别是当它用于开发多层、多进程应用时。我的直觉是 Oracle 是由于在基于 UNIX 平台上过多的后台进程才提供这种支持的。

Oracle 应用服务器 4.0 版本支持 IPC 及其他的机制，如 OMX、Oracle 的遵守 CORBA 的 ORB。此处的意见是服务应该同工业标准分布式对象一起实现，因而实现跨越多结点的分布与可伸缩性。

36.2.3 应用服务器层组件

这个层的组件主要分为两大类：ORB 组件与应用服务器组件。

1. ORB 组件

我们称 ORB 为一个单独的对象，但它实际上是由共同工作的五个进程组成的（在 Windows NT 中进程名以“40”结尾）。

- 监控端口监督程序 (mnmon40) —— 启动并监控其他的ORB组件。
- 日志服务器 (mnlogsrv) —— 日志的状态及错误信息。
- 公制端口监督程序 (mnorbmet) —— 由WRB监控进程使用, 收集应用的统计信息。
- 名字服务器 (mncnmsrv) —— ORB服务器、所有的应用服务器组件、所有的监听器 (除了节点服务器), 及所有使用名字服务器注册的组件。
- ORB服务器 (mnorbsrv) —— 处理从分布式对象来或返回的请求。

2. 应用服务器组件

ORB只是方便连接。请求实际是由应用服务器组件处理的。这些进程以一至两种模式运行: 压缩模式与非压缩模式。在非压缩模式下, 所有以后的进程都是可见的, 并且可以控制。在压缩模式下, 下面的一些进程显示为“oassrv”进程。

- 验证服务器 (wrbasrv) 与验证主机服务器 (wrbahsrv) —— 使用一个代理与服务器验证提供者, 这些进程处理监听器与插件的验证。
- 配置提供者 (wrbcfg) —— 所有的配置信息都包含在一个单独的文件, wrb.app文件中, 即使对远方结点也是这样。配置提供者读这个文件并提供适当的信息。
- 日志 (wrblog) —— 在一个集中的位置记录应用服务器信息、错误、警告等等, 这个集中的位置可以是一个文件或一个数据库。
- 监控端口监督程序 (wrbmon) —— 监控应用服务器的进程与对象。
- 资源管理器 (wrbroker) —— 只在主结点存在, 这个进程关注负载管理, 追踪插件服务器与实例的数量。
- RM代理 (wrbrmpxy) —— 维护指向JCORBA与EJB对象的对象。

36.3 分析应用层

在Oracle应用服务器上建立应用时, 有一个金字塔形的选择。可以建立 PL/SQL过程、移植Perl CGI脚本、建立Java伺服器或JCORBA对象, 甚至使用企业级Java Beans。

36.3.1 使用PL/SQL插件

有了PL/SQL插件, 可以使用Oracle存储过程开发自己的Web应用。Oracle向它的用户担保使用PL/SQL与数据库对象交互将比使用其他语言容易得多。显而易见的是, 由于 Oracle应用服务器使用Oracle数据库更为成熟, 对最适于移植与最具有伸缩性的问题的最可行的方法是并入PL/SQL。

在速度方面, PL/SQL插件自从第一版以来有了长足改进。这个改进来自于这样一种事实: 每个PL/SQL代理在请求到来之间的空闲中, 仍然连接到 Oracle数据库而不是在它将结果返回后就与数据库断开。此处的思想是没有必要为每一个存储过程的执行建立一个新的数据库, 在孤立测试以确保不会产生不良影响之后, 应该使用 PL/SQL的最新版本。

PL/SQL代理执行两项任务, 允许存储过程以创建动态 HTML页:

- 它调用一个参数化的过程并传入 HTTP请求中装载的真实参数, Web应用服务器必须将一个HTTP请求翻译为PL/SQL。
- 它提供了一种存储过程的方法, 将数据返回给调用者, 此处低级别的细节对你无缝的。

对于非骨干程序员，这里有比所需要的更多的信息。PL/SQL插件的主要优点在于它隐藏了与数据库交互的复杂性。基本上，当它接收到一个请求时，插件服务器将其记录到数据库日志中，执行一个存储过程并向浏览器返回任何结果。

除了执行这些工作以外，OAS还包括PL/SQL工具箱，它是封装的过程、函数与工具的集合，用以实现生产力与确保合法的HTML输出。这个工具箱包括的PL/SQL版本有全部HTML3.2的标签（在http包中），及帮助执行更高级任务的工具。

要记住的最重要的事情是从浏览器调用一个存储过程与从如SQL*Plus工具调用该过程是一样的。任何插入、更新、删除等等，都以相同的方式执行，只有一个重要的区别：除非正使用事务，否则只要请求一结束，插件实例将退出数据库，修改将被提交（见清单 36-1）。

清单36-1 示例PL/SQL过程

```
procedure demo_http (the_user varchar2) is
  user_host      varchar2(150);
  cnt            integer;
begin
  http.htmlOpen;
  http.headOpen;
  http.title(the_user||'s Page');
  -- Takes the value of "the_user" from the input parameters
  http.headClose;
  http.bodyOpen;

  user_host := owa_util.get_cgi_env('REMOTE_HOST');
  -- Gets the value of the "remote host" CGI environment variable. You
  -- can use any environment variable

  insert into user_history (web_user, web_host, web_time)
    values (the_user, user_host, sysdate);
  -- This will be inserted as if the procedure was executed from SQL*Plus

  http.p('Good morning, '||the_user||'!<P>');
  -- http.p() can be used to out put any arbitrary text to the browser,

  -- including HTML tags

  select count(*) into cnt from user_history;
  http.p('There have been '||cnt||' visits to this page so far.');
```

-- You can easily integrate information from the database into your pages.
-- This could just as easily have been names, times, anything stored in the
-- database.

```
  http.bodyClose;
  http.htmlClose;

end;
```

从浏览器执行这个过程，只需将浏览器设为：

http://www.yourhost.com/myApp/plsql/demo_http?the_user=Nick

假设已将虚拟路径“myApp/plsql”指派为指向PL/SQL插件。

为帮助了解这是如何工作的，可以如下方式进行测验：注册进入SQL*Plus，在调用这个网页之前与之后查看表中的内容，然后向表中手工插入一行，提交这个修改，再次调用这个网页。

不能对参数使用定位符号，名字必须确切匹配，每个进入的参数都是 `varchar2` 类型，因为它是 URL 文本的一部分。聪明的方法是，如果在包中重载过程，他们可以具有相同的参数类型，虽然参数的名字是不同的。

提示 可以在 SQL*Plus 中使用下述语句检查过程的输出：

```
set serveroutput on
exec myprocedure;
exec owa_util.showpage
```

Oracle 包括了成打的应用工具以简化工作，在开始工作之前仔细查看工具箱的文档。

注意 Oracle 实现了随时支取的 JAVA 存储过程，就像它们是 PL/SQL 存储过程一样，所以它们可以同样的方式从浏览器随时支取。

对于核心程序员，Oracle 继续支持 C 编程语言，将它合并入一个真正的插件，称为 UNIX 上的 C Web 插件，并简化了 Windows NT 上的 C 插件。

36.3.2 WRB SDK 的孩子：C Web 插件

在 2.0 版本中，Oracle 引入了 Web 请示代理软件开发者的工具，或 WRB SDK，这个开放式的 API 鼓励用户与解决方案提供者扩展 Oracle 预想的服务器，他们建立的应用称为“插件”，但现在简称为“应用”。这些应用是用 C 写的，可以做任何 C 程序可以做的工作，如与付款系统集成。他们作为共享库运行，在运行时动态加载并放入缓存。

如果为以前的版本书写 C 插件，需要使用结点管理器中的 OAS 工具将其升级，但它们仍可工作。老的 API 仍被支持，但在将来的版本中可能被撤走。

36.3.3 WRB 应用程序界面

可以使用 WRB API 与 ORB 注册三个基本回调函数：

- 初始化。
- 请求处理。
- 关闭。

这些方法依据典型的类结构集合到一起，可以初始化对象、在对象初始化后发送与接收信息、使用完成后破坏这个对象。这个结构使它自己非常易于面向对象方法的系统开发。

WRB API 含有的辅助方法与函数极多。这些方法中的一些将在本章稍后讨论。这些方法将从先前提到的三个基本回调直接调用。

Oracle 对产品开发的开放途径指的是用户与方案解决提供者 / 合作者联合开发 WRB API。这个合作努力意味着 API 将在 Oracle 应用服务器与第三方应用之间提供最大的开放与合作能力。

36.3.4 理解插件与 ICX

ICX，代表插件之间的交换，是一个 Oracle 应用服务器支持的机制，推动插件之间的通信，它允许不同的机器之间进行通信。特别是使用基于 Java 的插件。

1. WRB 插件

每个插件都遵循一个体系结构 / 模型类型，插件开发者只需要知道如下三种设计模型：

- 请求-响应模型。
- 会话。
- 事务。

2. 请求-响应模型

与膝跳反射几乎相似，WRB插件实例处理单独的客户请求并对每一个预期的响应执行一个请求。预期的请求被分派后，插件实例具有了“健忘症”，此时没有数据与客户端相关，请求也不是永久地到处存储。如果熟悉缺少规则HTTP请求追踪的状态信息，这将是非常常见的。所以，插件的Exec方法应该将请求作为变元接收，并返回相关信息而不需追踪任何状态数据。

3. 会话模型

在客户与一个特定的执行中心插件实例之间建立会话。然后插件使用会话机制维护这两个之间的持久关联。可以通过设置休息时间确定连接坚持多长时间。然后插件的Exec方法将声明与客户相关的信息并初始化请求。Exec方法将这个声明信息存储在拥有实例插件的应用的上下文结构中，即，应用负责创建插件的一个实例（初始化插件实例，设计时插件的运行状态对象拷贝）。

注意 插件运行时的体系结构与设计时体系结构的区别的详细信息，参见Grady Booch、Jim Rumbaugh或Lvar Jacobson在面向对象方法学方面的著作。

会话机制确保在客户端与可执行插件实例之间有一个关联，这个关联提供给插件在它自己与客户端之间的一个特殊的焦点。这个实例特殊化正是运行时与设计时体系结构不同的意义所在。在任何给定的时间，可以执行一个插件的许多实例，每个实例含有特定用户的特定状态与数据。也就是说，如果你有一个为特定的银行分支管理许多经常帐户的插件，插件的一个实例具有\$5 000的结余，另一个实例具有\$230.98的结余，另一个具有\$1 209的结余，等等，所有的都应在本日末协调他们的批量执行。数据完整性不会被破坏，因为每个插件实例只关心与一个帐户相关的信息，然后可以保存与应用中每个帐户有关的状态信息，将其放入文本结构，以便此信息在当前用户会话期间是存活的。

注意 在3.0版本中，每个插件实例在它自己的进程空间运行。在4.0版本中，实例可以共享进程空间（在插件服务器进程中）。

4. 事务模型

如果在分析与设计中，意识到插件在管理数据库事务时需要集中，事务模型正合适，Exec方法将承担事务处理的角色，在这种模型的实现中包括一些常用状态：

- 插件接收一个请求。
- 插件开始一个事务。
- 插件在事务范围内执行增加的更新。
- 如果每件事都顺利，插件提交一个事务。
- 如果不是每件事都顺利，插件进行事务上的回滚（事务的状态：结束）。
- 插件向客户端发送一个响应（认可或错误）。

在一个调用中不必产生事务（调用Exec方法）。一些事务可以横跨许多对插件的Exec方法的调用。在此是插件的Exec方法必须决定对当前场景应用哪种前述的状态。如果设计将来调用这个特性，这个模型允许支持与任何事务无关的查询（更详细的信息，参见插件设计方面

的WRB事务服务API指南)。

如果你已充分理解, 并购买 Oracle应用服务器的企业版, 你可以从事务服务库中得到更多的API帮助 (在Oracle应用服务器的基本版中不包括这些服务)。通过这个库, 你或你的开发小组可以从你的插件执行任何数量的数据库事务。Oracle将WRB事务服务建立在TX接口之上, 与它被x/Open公司定义了一样。

注意 如果需要关于这个有帮助的特性的更多的信息, 查看 X/开放的分布式事务处理: TX (事务分界) 规范。

通过高级事务服务, 插件可以执行跨越多个 HTTP请求的数据库事务。对数据库的真正存取通过基于数据库存取的 API被建立, 如 OCI或ProC, 作为事务服务的补充。事务函数如 commit与rollback并不通过这些服务完成。为使插件可以使用事务服务, 必须启用这个插件的事务服务。查阅事务服务文档以获取更详细的细节。

5. 看看ICX

WRB插件之间交换服务API允许一个WRB插件将HTTP请求发给另一个WRB插件。

使用ICX帮助系统体系结构设计提供一个应用或企业领域特殊行为的插件。 ICX允许插件旋转在不同的机器上, 因而改进了 Oracle应用服务器的性能, 并对网络提供了有价值的负载均衡。这也意味着增长的存储空间 (所有应用的组件并不位于一台机器上), 及更灵活的体系结构。与好的类与对象模型结合, ICX使维护许多交互式的插件相对容易一些。

表36-1显示了WRB插件之间交换服务API中的所有方法。

表36-1 ICX APE方法

方 法	描 述
WRB_ICXcreateRequest()	创建一个请求对象
WRB_ICXdestroyRequest()	摧毁一个请求对象
WRB_ICXfetchMoreData()	当WRB_ICXmakeRequest()返回WRB_MOREDATA结构时, 得到更多的数据
WRB_ICXgetHeaderVal()	得到特定响应头的值
WRB_ICXgetInfo()	得到有关一个请求的信息
WRB_ICXgetParsedHeader()	得到响应头
WRB_ICXmakeRequest()	执行一个请求
WRB_ICXsetAuthInfo()	为一个特定请求设置授权头
WRB_ICXsetContent()	为一个特定请求设置内容数据
WRB_ICXsetHeader()	为一个特定请求设置头
WRB_ICXsetMethod()	设置一个特定请求中使用的 HTTP方法
WRB_ICXsetNoProxy()	指明代理服务器不能使用的域
WRB_ICXsetProxy()	指明一个代理服务器
WRB_ICXsetWalletInfo()	设置ICX请求去使用 SSL证明用于授权

ICX API还支持两个枚举的类型:

■ WRBInfoType

■ WRBMethod

下面的部分描述每一种方法。

WRB_ICXcreateRequest() 这个方法分配并返回对一个不透明请求对象的处理, 对一个由给定URL指明的请求编码。程序清单 36-2显示了创建一个请求对象所需要的代码:

清单36-2 用WRB_ICXcreateRequest()创建一个请求对象

```
dvoid *
WRB_ICXcreateRequest(void *WRBCTX,
text *url);
Parameters
Return Values
WRB_ICXcreateRequest() returns a handle to the newly created request object.
WRB_ICXcreateRequest() returns NULL on failure.
```

除了创建请求对象，或许还想发出请求，发出请求的过程如下：

- 1) 调用WRB_ICXcreateRequest ()。
- 2) 调用WRB_ICXmakeRequest ()。
- 3) 发出请求。
- 4) 如果出现了错误，通过调用 WRB_ICXdestroyRequest ()方法终止请求，这个调用释放WRB为请求分配的资源。

* WRBCTX是指向WRB应用引擎传递给你的插件方法的 WRB不透明内容对象的指针，text*url是URL请求的指针。

WRB_ICXdestroyRequest () 这个方法释放WRB为特定请求分配的资源。释放不再使用的对象占用的资源是一个很好的编程习惯。程序清单 36-3展示了在使用完请求对象后，如何摧毁它。

清单36-3 使用WRB_ICXdestroyRequest ()摧毁一个请求对象

```
WAPIReturnCode
WRB_ICXdestroyRequest(void *WRBCTX,
dvoid *hRequest);
Parameters
Return Values
WRB_ICXdestroyRequest() returns a value of type WAPIReturnCode.
```

请求完成后，必须调用WRB_ICXdestroyRequest ()，只要不使用WRB_ICXmake Request ()发出一个请求，就可以使用WRB_ICXdestroyRequest ()取消一个由WRB_ICXcreateRequest ()创建的请求。

* WRBCTX是指向WRB应用引擎传递给你的插件方法的 WRB不透明内容对象的指针，*hRequest指明将被摧毁的请求，WRB_ICXcreateRequest ()将返回这个控制。

WRB_ICXfetchMoreData () 当一个先前对 WRB_ICXmakeRequest ()的调用返回WRB_MOREDATA时，WRB_ICXfetchMoreData ()重新得到请求数量的字节。如果这个请求的数量比可以获得的大，这个方法返回可以获得的字节数量。程序清单 36-4详细展示了这个方法。

清单36-4 使用WRB_ICXfetchMoreData ()获取更多数据

```
WRBAPIReturnCode
WRB_ICXfetchMoreData(dvoid *WRBCTX,
dvoid *hRequest,
dvoid **response,
ub4 *responseLength,
ub4 chunkSize);
Parameters
```

Return Values

WRB_ICXfetchMoreData() returns a value of type **WAPIReturnCode**.

*WRBCTX是指向 WRB 应用引擎传递给你的插件方法的 WRB 不透明内容对象的指针, **dvoid *hRequest**指明将被摧毁的请求, **dvoid **response**是一个指针, 指向的位置是方法存储响应数据的指针, **ub4 * responseLength**是指向函数存储响应数据长度 (以字节为单位) 的指针。如果 **ub4 chunkSize** 具有一个非零值, 需要重复调用 **WRB_ICXfetchMoreData ()** 方法直至接收到整个响应。如果 **ub4 chunkSize** 的值为 0, 不返回数据。

如果对 **WRB_ICXmakeRequest ()** 的调用返回 **WRB_MOREDATA**, 可以调用这个方法必要多的次数以获取请求响应 **chunkSize** 额外的字节, 需要这样做直至接收到所有的响应数据。

WRB_ICXgetHeaderVal () 在使用这个方法之前, 必须调用 **WRB_ICXmakeRequest ()** 方法, 它发出一个请求。**WRB_ICXgetHeaderVal ()** 使用这个请求的响应返回一个指定 HTTP 头的值 (参见清单 36-5)。

清单 36-5 使用 WRB_ICXgetHeaderVal () 获取一个指定响应头的值

```
text *
WRB_ICXgetHeaderVal(void *WRBCTX,
dvoid *hRequest,
text *name);
Parameters
Return Values
WRB_ICXgetHeaderVal() returns the value of the specified header.
WRB_ICXgetHeaderVal() returns NULL on failure.
```

WRB_ICXgetHeaderVal () 返回的指针指向由 **WRB_ICXmakeRequest ()** 发出请求的响应的特定 HTTP 头的值, 这个指针对于检索响应头含有的响应数据是非常有价值的。

*WRBCTX是指向 WRB 应用引擎传递给你的插件方法的 WRB 不透明内容对象的指针, *hRequest 指明 WRB 用以提取响应头值的响应, **text * name** 是指向你想要的头的名字的指针。

WRB_ICXgetInfo () 这个方法返回含有有关特定请求信息的字符串, 可以指定想要返回的信息的种类, 程序清单 36-6 详细展示这个方法。

清单 36-6 使用 WRB_ICXgetInfo () 获取有关一个请求的信息。

```
text *
WRB_ICXgetInfo(void *WRBCTX,
dvoid *hRequest,
WRBInfoType infoType);
Parameters
Return Values
WRB_ICXgetInfo() returns a pointer to the requested information as a character
string. WRB_ICXgetInfo() returns NULL on failure.
```

当一个 ICX 请求完成时, 调用 **WRB_ICXgetInfo ()** 以获取有关这个请求的信息。触发这个完成状态的事件是 **WRB_ICXmakeRequest ()** 的完成与返回。此处无用的是万一 WRB 重新以适当的授权信息执行请求时, 你会获得界名 (realm name)。

*WRBCTX是指向 WRB 应用引擎传递给你的插件方法的 WRB 不透明内容对象的指针, **dvoid *hRequest** 指明你需要的是哪个请求的信息, **infoType** 与 **WRBInfoType** 代表指明你想要的信息类型的代码, **infoType** 参数指明返回信息的种类。

WRB_ICXgetInfo ()接受将由WRBInfoType指明的请求信息类型的变元返回给调用者，程序清单36-7显示了这种类型的结构。

清单36-7 WRBInfoType枚举类型

```
typedef enum _WRBInfoType
{
    STATUSCODE,
    HTTPVERSION,
    REASONPHRASE,
    REALM
} WRBInfoType;
WRBInfoType Values
```

STATUSCODE代表HTTP响应代码，HTTPVERSION代表响应使用的HTTP协议的版本，REASONPHRASE代表与HTTP响应代码一致的原因文本串，REALM代表在响应中指明的授权界名。

WRB_ICXgetParsedHeader () 在使用这个方法之前，必须调用WRB_ICXmakeRequest ()发出一个请求，WRB_ICXgetParsedHeader ()使用这个请求的响应返回指定头的值，程序清单36-8展示了这个方法。

清单36-8 使用WRB_ICXgetParsedHeader ()获取响应头

```
WAPIReturnCode
WRB_ICXgetParsedHeader(void *WRBCTX,
    dvoid *hRequest,
    WRBpBlock *hPblock);
Parameters
Return Values
WRB_ICXgetParsedHeader() returns a value of type WAPIReturnCode.
```

这个方法返回由hPblock指向的地址，这个地址是一个含有由WRB_ICXmakeRequest ()发出的ICX请求响应的参数块。此处无用的是你可以检索响应头包含的响应数据。

*WRBCTX是指向WRB应用引擎传递给你的插件方法的不透明的WRB内容对象的指针，dvoid *hRequest指明请求，WRBpBlock * hPblock是指向函数用以存储含有解析头数据的参数块的位置的指针。

WRB_ICXmakeRequest () 这个方法发出指定的请示，程序清单36-9展示了这个方法。

清单36-9 使用WRB_ICXmakeRequest ()发出一个请求

```
WAPIReturnCode
WRB_ICXmakeRequest(void *WRBCTX,
    dvoid *hRequest,
    void **response,
    ub4 *responseLength,
    ub4 chunkSize,
    ub1 sendToBrowser);
Return Values
WRB_ICXmakeRequest() returns a value of type WAPIReturnCode.
```

在调用WRB_ICXcreateRequest ()创建一个请求及其他的ICX API函数如WRB_ICXsetHeader ()与WRB_ICXsetContent ()准备请求后，可以调用WRB_ICXmakeRequest ()发出请求。

*WRBCTX是指向WRB应用引擎传递给你的插件方法的不透明的WRB内容对象的指针，

dvoid *hRequest指明要发出的请求。

Void ** response是用以指向函数存储指向响应数据的位置的指针， ub4 * responseLength是指向函数用以存储响应数据字节长度的地址的指针。

ChunkSize ub4有一个非零值，请求响应的字节长度必须与这个值匹配。在这种情况下，需要反复调用WRB_ICXfetch MoreData ()方法直至接收到全部响应。如果ub4 ChunkSize的值为0，不返回数据。如果ub1 sendToBrowser是非零值，WRB将直接将响应发送给起始浏览器。在这种情况下，响应参数返回时将为空。

WRB_ICXsetAuthInfo () 这个方法设置伴随指定请求的授权头数据（参见清单 36-10）。

清单36-10 使用WRB_ICXsetAuthInfo ()设定指定请求的授权头

```
WAPIReturnCode
WRB_ICXsetAuthInfo(void *WRBCTX,
dvoid *hRequest,
text *username,
text *password,
text *realm);
Parameters
Return Values
WRB_ICXsetAuthMethod() returns a value of type WAPIReturnCode.
```

如果你的插件向另一个插件发出请求，反过来需要你的插件为它自己授权，你需要调用WRB_ICXsetAuthInfo ()方法，这样做为其他插件的每一个请求设置授权头。

*WRBCTX是指向WRB应用引擎传递给你的插件方法不透明的 WRB内容对象的指针，dvoid *hRequest指明为哪个请求建立授权。WRB_ICXcreateRequest ()返回这个句柄， text * username是指向请求授权的用户名的指针。用户名必须在指定的界中定义。 text * password指向用户名口令， text * realm指向定义用户名的授权界的名字。

WRB_ICXsetContent () WRB_ICXsetContent ()设置指定请求的请求内容（参见清单 36-11）。

清单36-11 使用WRB_ICXsetContent ()为一个指定请求设置内容数据

```
WAPIReturnCode
WRB_ICXsetContent(void *WRBCTX,
dvoid *hRequest,
WRBpBlock hPBlock);
Parameters
Return Values
WRB_ICXsetContent() returns a value of type WAPIReturnCode.
```

为一个特定查询建立内容数据，执行以下步骤：

- 1) 调用WRB_createPBlock ()分配一个含有内容的参数块。
- 2) 将参数块传递给WRB_ICXsetContent ()，通过传递WRB_ICXcreateRequest ()返回的请求句柄指定请求。
- 3) 设置内容数据。

*WRBCTX是指向WRB应用引擎传递给你的插件方法的不透明的 WRB内容对象的指针，dvoid *hRequest指明是指定哪个请求的内容，这应该是由 WRB_ICXcreateRequest ()返回的句柄，WRBpBlock hPBlock描述含有请求内容的参数块。

WRB_ICXsetHeader () WRB_ICXsetHeader ()负责设置指定请求的HTTP头数据（参见清单36-12）。

清单36-12 使用WRB_ICXsetHeader ()为指定请求设置头

```
WAPIReturnCode
WRB_ICXsetHeader(void *WRBCTX,
dvoid *hRequest,
WRBpBlock hPBlock,
boolean useOldHdr);
Parameters
Return Values
WRB_ICXsetHeader() returns a value of type WAPIReturnCode.
```

调用这个方法需要首先进行一些特定的调用，调用顺序如下：

- 1) 调用WRB_createPBlock ()分配一个参数块及包含头数据。
- 2) 将参数块传递给WRB_ICXsetHeader ()，通过传递WRB_ICXcreateRequest ()返回的请求句柄指定请求。
- 3) 为请求设置头数据。

*WRBCTX是指向WRB应用引擎传递给你的插件方法的不透明的 WRB内容对象的指针，dvoid *hRequest指明设定哪个请求的头数据。WRB_ICXcreateRequest ()返回dvoid *hRequest句柄，WRBpBlock *hPBlock描述含有头信息的参数块。如果 boolean useOldHdr设为TRUE，ICX除参数块中定义的数据外，还从初始请求查询相应的头数据。如果 boolean useOldHdr设为FALSE，只使用参数块中的头数据。

WRB_ICXsetMethod () WRB_ICXsetMethod ()负责为指定的请求设置请求方法，如GET或POST（参见清单36-13）。

清单36-13 使用WRB_ICXsetMethod ()设置指定请求中使用的HTTP方法

```
WAPIReturnCode
WRB_ICXsetMethod(void *WRBCTX,
dvoid *hRequest,
WRBMethod method);
Parameters
Return Values
WRB_ICXsetMethod() returns a value of type WAPIReturnCode.
```

此处假设已调用 WRB_ICXcreateRequest ()方法创建了一个请求。调用 WRB_ICXsetMethod ()指定这个请求的请求方法，缺省请求方法是GET。

* WRBCTX 是指向WRB 应用引擎传递给你的插件方法的不透明的 WRB 内容对象的指针，dvoid *hRequest 指明请求的设置方法。WRB_ICXcreate Requestt ()返回这个句柄。

WRB_ICXsetMethod ()具有一个WRBMethod类型的变元，描述请求中将使用的请求方法（参见清单36-14）。

清单36-14 WRBMethod枚举类型

```
typedef enum _WRBMethod
{
OPTIONS,
GET,
HEAD,
```



```
POST,  
PUT,  
DELETE,  
TRACE  
} WRBMethod;
```

要获取这个枚举类型的详细信息，参考 Web 请求代理的有关文档。

WRB_ICXsetNoProxy () 这个方法建立了代理服务器（由 WRB_ICXsetProxy() 指定）不能使用的 DNS 域的列表，这确保了拒绝任何给定代理服务器最初试图请求的 URL，程序清单 36-15 展示了这个方法是怎样实现的。

清单 36-15 使用 WRB_ICXsetNoProxy () 指定代理服务器不能使用的域

```
WAPIReturnCode  
WRB_ICXsetNoProxy(void *WRBCTX,  
text *noProxy);  
Parameters  
Return Values  
WRB_ICXsetNoProxy() returns a value of type WAPIReturnCode.
```

如果插件调用 WRB_ICXsetProxy () 启动代理服务器请求翻译，但你不希望所有 DNS 域都使用代理服务器到达，使用 WRB_ICXsetNoProxy () 指定一个以逗号分隔的域的列表，请求直接发送到这些域，而不需要代理服务器的插入。

*WRBCTX 是指向 WRB 应用引擎传递给你的插件方法的不透明的 WRB 内容对象的指针，*noProxy 指向以逗号分隔的直接发送的请求的 DNS 域的列表。

WRB_ICXsetProxy() 这个方法告诉插件在建立将来的 ICX 请求时使用哪个必须在防火墙外路由的代理服务器。程序清单 36-16 展示了怎样指定一个代理服务器。

清单 36-16 使用 WRB_ICXsetProxy 指定代理服务器

```
WAPIReturnCode  
WRB_ICXsetProxy(void *WRBCTX,  
text *proxyAddress);  
Parameters  
Return Values  
WRB_ICXsetProxy() returns a value of type WAPIReturnCode.
```

当基于内部网的插件需要将 ICX 请求分派到防火墙外的服务器时，调用这个方法很有用。插件将指向这个方法设定的地址。

*WRBCTX 是指向 WRB 应用引擎传递给你的插件方法的不透明的 WRB 内容对象的指针，*proxyAddress 以字符串形式表示代理地址。

36.3.5 WRB 记录器应用程序界面

在 Oracle 应用服务器与 Oracle 数据库紧密结合的前提下，应该会更熟悉与数据库连接相关的 WRB API，WRB 记录器 API 函数提供了这种例程：

- WRB_LOGopen () —— 打开一个文件或建立一个数据库连接。
- WRB_LOGwriteMessage () —— 将系统信息写入永久存储。永久存储可以是数据库或一个普通的文件，其中日志数据在服务器应用生命期外仍可存活。

注意 在 Oracle 应用服务器的早期版本中支持 WRBLogMessage ()。在 Oracle 应用服务器

4.0版本中，应该使用WRB_LOGwriteMessage ()代替它。

■ WRB_LOGwriteAttribute () —— 将客户端定义的属性写入永久存储。

■ WRB_LOGclose () —— 关闭一个文件或断开数据库连接。

程序清单36-17展示了WRB_LOGopen ()方法的实现。

清单36-17 WRB_LOGopen ()的语法

```
WAPIReturnCode WRB_LOGopen( dvoid *WRBctx,
ub4 *logHdl,
WRBLogType type,
WRBLogDestType dest,
text *MyFileName );
Parameters
Return Values
[Returns WAPIReturnCode]
```

从插件组件调用这个方法用以打开一个文件或初始化一个数据库连接。 WRB应用引擎将WRBctx指针传递给你的插件中的调用方法； WRBctx指向WRB内容对象； LogHdl是指向连接对象中含有的对象类型的处理：文件或数据； type指出日志条目的类型； dest指明将事务日志写入一个平面文件还是数据库； MyFileName是文件的字符串名，当将日志信息写入数据库时，它也可以为空。

程序清单36-18中的方法将系统信息写入由 LogHdl指明的永久存储中。

清单36-18 WRB_LOGwriteMessage ()的语法

```
WAPIReturnCode WRB_LOGwriteMessage( dvoid *WRBctx,
ub4 logHdl,
text *component,
text *msg,
sb4 severity);
Parameters
Return Values
Returns WAPIReturnCode.
```

从插件组件调用这个方法用以将系统信息写入由 LogHdl中的值指明的永久存储中。 WRB应用引擎将 WRBctx指针传递给你的插件中的调用方法； WRBctx指向WRB内容对象； LogHdl是指向连接对象中含有的对象类型的处理：文件或数据库； type指出日志条目的类型，它可以是客户端属性或信息组件，指向文本描述确定插件的类型，如“ java”。msg就是你想要记录的文本，信息不能超过 2K长度；反引号（'）用于分隔，如果在你的信息中需要含有这个字符，必须以反斜线为标记（即\'）； severity只是消息的严谨级。

程序清单36-19中的代码将客户端定义的属性写入由 LogHdl指明的存储中。

清单36-19 WRB_LOGwriteAttribute ()的语法

```
WAPIReturnCode WRB_LOGwriteAttribute( dvoid *WRBctx,
ub4 logHdl,
text *component,
text *name,
text *value);
Parameters
Return Values
Returns WAPIReturnCode.
```

从插件组件调用这个方法用以将客户端定义的属性写入由 LogHd1中的值指明的永久存储中。存储的信息对查看你的客户插件是怎样响应系统的是非常有用的。当开发组件时，追踪错误与例外处理是非常关键的， WRB应用引擎将 WRBCtx指针传递给你的插件中的调用方法；WRBCtx指向WRB内容对象， LogHd1是指向连接对象中含有的对象类型的处理：文件或数据库；type指出日志条目的类型，它可以是客户端属性或信息组件，指向文本描述确定插件的类型，如“ ODBC ”。msg就是你想要记录的文本； name是一个文本项，指明你想要用日志记录下来的特定属性； value提供附加的文本以限定你命名的属性。参见程序清单 36-20，查看WRB_LOGclose ()方法的语法。

清单36-20 WRB_LOGclose ()的语法

```
WAPIReturnCode WRB_LOGclose( dvoid *WRBCtx, ub4 logHd1);  
Parameters  
Return Values  
Returns WAPIReturnCode.
```

依据LogHd1中指明的值，从插件组件调用这个方法用以关闭一个文件或断开一个数据库连接。

WRB应用引擎将 WRBCtx指针传递给你的插件中的调用方法， WRBCtx指向WRB内容对象； LogHd1是指向连接对象中含有的对象类型的处理：文件或数据库，这个处理在 WRB_LOGopen ()方法中创建。

36.3.6 使用JWeb插件

JWeb插件，以前称为Java插件，也许会有一些混淆。JWeb插件与Java applets毫无关系，Java applets是一个下载的小程序，运行在浏览器中。本质上是一个Java虚拟机，JWeb插件在服务器上运行一个Java应用，并直接将HTML返回给浏览器。浏览器不需要特殊的能力，因为它除了解释HTML，不需要做任何事情。

Oracle还提供了pl2java，一个允许从Java应用调用PL/SQL的工具，及Java Toolkit，允许将PL/SQL与WRB函数写入Java应用。

1. 增加一个java应用

在运行一个Java应用之前，需要让OAS知道它在哪里，它是什么。

- 1) 访问结点管理器。
- 2) 点击OAS管理器。
- 3) 扩展左边的主菜单并点击应用。
- 4) 在右边，通过点击+，增加一个新的应用。
- 5) 在应用类型下面，选择JWeb，确认单选按钮被手工选中，点击应用按钮（参见图36-3）。
- 6) 填入应用名（即，Hello World）与显示名（helloworld），这些是你的应用将分别在应用列表与菜单中显示的。选取合适的版本号，如1.0，为这个应用按钮点击增加插件。
- 7) 输入插件名与显示名，这些将被用于在结点管理器中显示信息。
- 8) 输入将要存放已编译的类文件的路径位置及真实路径，如 /java/myTest，并点击应用，这个路径必须是CLASSPATH环境变量的一部分。
- 9) 重新装载应用服务器，点击左边的 Oracle应用服务器，然后选取“所有”与“重载”。

它看起来像一个VCR反复播放按钮。

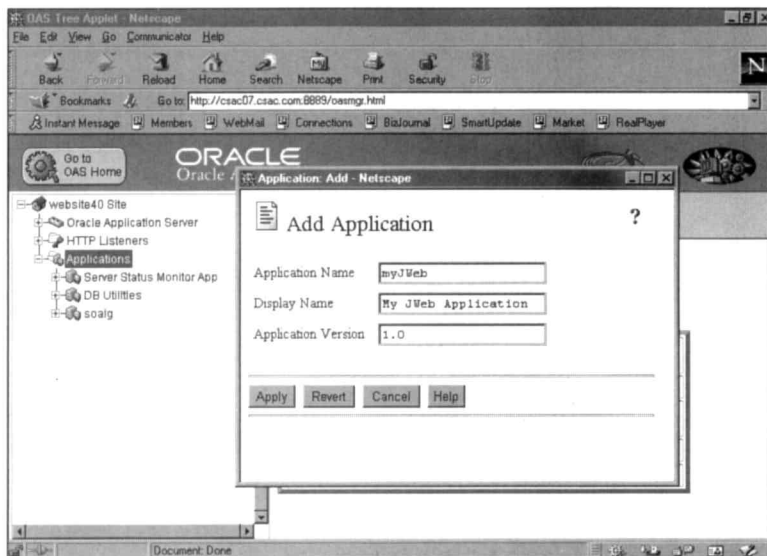


图36-3 增加一个JWeb应用

现在已使用一个插件创建了“Hello World”应用。

提示 如果已有一个用PL/SQL书写的的应用，不用感觉有责任将它们转换为Java。Oracle已经构建对Java的支持，所以可以从一个程序调用另一个，不用重写PL/SQL，只需要从新的Java代码中调用它即可。

2. 在Java中建立一个网页

现在准备创建真正的Java应用。在此与常用的Java编程有微小的区别：你需要有一个或多个类，但用户基本上使用main()方法启动应用，并一起使用它直至完成。在这个时间内，用户看到的東西可能被许多不同的方法控制着。当为OAS创建Java应用时，记住每个单独的网页都有它自己的类，OAS将总是被main方法调用以显示它。让我们看一个样例应用（见清单3-21）：

清单36-21 Jweb应用示例

```
import oracle.html.*; // This is the Java Toolkit
class showJavaPage {
    public static void main (String args[]) {
        HtmlHead h = new HtmlHead("My Java App");
        // Just creates the HTML for the <HEAD> and <TITLE> tags
        HtmlBody b = new HtmlBody();
        b.addItem("This text was generated by my Java Application.");
        // Add text to the body
        HtmlPage p = new HtmlPage(h, b);
        // Creates the page object and adds the head and body objects
        p.printHeader();
        // All pages must have HTTP headers.
        p.print();
        // The page prints itself.
    } // main
} // class
```

在应用服务器可以获取它之前，必须将它编译为 *.class 文件。JWeb 插件使用 Java 1.1.4，而不是 1.2 或 1.1.6，所以仔细察看你增加了什么特性。Oracle 建议选择一个使用相同版本的编译器。Oracle 也有它自己的 Java 工具，JDeveloper。

在类被编译并测试后，在创建应用时将它拷贝到你指向的路径。为在你的浏览器中察看它，到下述 URL：

<http://www.yourmachine.com/java/myTest/showJavaPage>

注意 www.yourmachine.com 是你的机器，/java/myTest 是你创建应用时进入的虚拟路径，注意即使你的应用称为 “Hello World”，但你实际并不能看到这个文本。应用的名字是一个任意的选择，只有应用的系统管理员使用结点管理器时需要它。

使用 JWeb 插件，你将获取与使用 CWeb 插件相同的 WRB 服务，它们包含在 Oracle.* 包中。

提示 在版本 4.0 中，不赞成使用 Oracle.owas.wrb.services.logger 包，使用 Oracle.OAS.Services.Logger 代替。

对拿不定主意的人，LiveHTML 只是服务器端包括的 Oracle 的名字，或 SSI。SSI 是 Web 在网页上提供动态信息的最早的方法之一。它允许开发者创建一个静态 HTML 页，但在内容中“内嵌”动态信息，如日期或另一个 HTML 页。当服务器接收到对这些页的请求时，它分析这个页，增加附加的信息，并将整个事件送回给浏览器。查看 HTML 源文件没有给出这个发生的任何指示。LiveHTML 可以分为两类，第一类是 SSI。

在 SSI 中，命令以如下语法被内嵌：

```
<!--#command var=values-->
config: Determines how subsequent information will be displayed
<!--#config cmdecho=on-->
include: Includes the contents of a file.
        Can be an absolute or virtual path, so you can include dynamic information
<!--#include file="/u08/webdocs/motd.txt"-->
<!--#include virtual="/soaig/plsql/register.reg_page"-->
echo: Prints an environment variable
<!--#echo var="DATE_LOCAL"-->
fsize: Prints the size of a file -- not necessarily the file that
        includes this command
<!--#fsize file="pic3.jpg">
flastmod: Same as fsize, but prints the last time the file was modified
<!--#flastmod file="index.html">
exec: Executes a script. This could be a CGI script or a shell script.
        Administrators can disable this tag, since it does constitute a security
        risk. Note that to see the results of a cmd execution, you must use
        config to set cmd=on.
<!--#exec cgi="/cgi-bin/counter.pl">
<!--#exec cmd="/home/nick/scripts/myscript.sh">
request: Allows you to access the ICX by calling on another cartridge. Most
        commonly used to get the results of a PL/SQL, Perl or Java request, but
        can be used to call on another machine altogether, and even allows access
        to protected URL's by allowing for a username and password to be included.
<!--#request url="http://user:pass@www.mymachine.com/java/user_info?day=Sunday-->
```

提示 可以使用 LiveHTML 的 SSI 能力以允许用户改变内容，在 INCLUDEed 中以文本文件的形式，不用接触接口自身。

在 OAS 上的 LiveHTML，扩展了规模，不仅包括日期或另一个文件的内容。事实上，Oracle 含有与微软的 Active Server Pages 相似的功能，然而服务器端的脚本不是基于 VB 的，而

是基于Perl。这有几个优点，Oracle提供了许多标准的“Web应用对象”允许高级功能甚至存取CORBA对象。

内嵌脚本有三种语法形式，与ASP相似：

```
<% %> : Encapsulates a script
<%= %> : Prints the value of an expression
<SCRIPT RUNAT=SERVER> </SCRIPT> : Also encapsulates a script,
but if "RUNAT=SERVER" is left off, the script text will be provided to
and run by the browser.
```

也可以选择存取该页提供的变化，例如，假设调用网页

<http://www.myserver.com/scriptpages/hellopage.asp?user=Nick&numstuff=3>

该网页的文本包括

```
Let me guess. Your name is
<% $Response->write($user) %>
, right?
<% $Response->write(" Let's see. You own ");
    for ($countem = 1; $countem >= $numstuff; $countem++) { %>
    <%= $countem %>,
    <% } ;
    $Response->write(" computers.")%>
```

输出为：

Let me guess. Your name is Nick, right? Let's see. You own 1, 2, 3, computers.

注意 也可以选择使用<SCRIPT Language="language">指明一种语言，但现在只支持Perl语言，Oracle计划将来增加对其他语言的支持。

36.3.7 使用Perl插件

在应用服务器出现之前，多数CGI脚本都是用C或Perl写的，我们一直在使用CWeb与C应用打交道，现在我们准备与Perl打交道。Perl脚本比书写C应用容易得多，但将它们作为CGI运行有它的缺点。首先，像任何CGI脚本一样，每次一个用户产生一个请求，都需要启动翻译器，因为Perl是一种解释性语言，这将是严重的资源与性能消耗。

OAS使用Perl插件解决了这个问题，Perl插件本质上是Perl翻译器的一个版本，保留在内存中等待请求，然后执行它们。依赖于操作系统，插件作为共享对象（UNIX、libperlctx.so）或动态链接库（NT、perlnt40.dll）实现。将Perl脚本升级为OAS是一个简单的过程——基本上将虚拟路径重新映射指向插件服务器而不是CGI。

创建一个Perl应用，遵循与JWeb应用相似的步骤，选择Perl替代JWeb，文件位置不需要是任何特定环境变量的一部分。

必须紧记Perl插件的一些事项：

- 翻译器是基于Perl 5.003的。
- 标准I/O与STDERR分别重新指向浏览器与WRB记录日志。
- 不能使用“fork”，使用“system”，但记住输出已被重新指向浏览器。

36.3.8 企业级Java Beans与JCORBA

企业级Java Beans（EJB）与JCORBA包括用Java书写CORBA对象。EJB规范由JavaSoft创

建，JCORBA规范是一个Oracle拥有的规范，先于EJB由Oracle书写，最后由JavaSoft完成。它们都设计用于封装对象并允许从其他的CORBA对象发送或接收信息。选择使用哪个依赖于你已做了什么开发及想要完成什么，紧记以下事项（见表36-2）。

表36-2 选择一个对象模型

对 象	EJB	JCORBA
现存对象	扩展java.rmi.Remote	扩展org.omg.CORBA.Object
传递对象用	值	指引
客户端的书写用	只有Java	Java或其他语言

如果使用第三方BEANS，也需要使用EJB。

36.4 小结

Web的早期结点只由静态信息组成，甚至没有图表。从那时以来事务已改变了许多！今天，不只Web有了巨大的进步，而且Web类型的应用也在商务的所有方面迅速增长。幸运的是，技术能跟上这个步伐，你可以使用Oracle应用服务器不只将现存的PL/SQL应用在Web上使用，而且跟上Java的步伐，甚至创建了使用C插件通常不用考虑Web内容的插件。通过这些应用服务器体系结构的优势，不论以后出现什么，都可以使用Oracle应用服务器或它的后继产品建立应用。

第37章 安装与配置OAS

本章要点：

- 在Sun Solaris上安装Oracle应用服务器
- 在Windows NT上安装Oracle应用服务器
- 理解Oracle应用服务器的最新安装特性
- 安装Oracle应用服务器
- 使用Oracle应用服务器的结点管理器
- 创建HTTP监听器
- 安装PL/SQL工具包
- 创建一个新的PL/SQL插件
- 疑难解答

37.1 在Sun Solaris上安装Oracle应用服务器

Oracle应用服务器自从它只是紧密集成的 Oracle7.x服务器中包括的一个简单的超文本协议（HTTP）服务器以来，有了很大发展。这个数据库服务器组件允许在静态的、或没有变化的数据之外，从Oracle数据库中存储的数据创建动态超文本标记语言（HTML）文档。

HTTP监听器，与一个新的“应用服务器层”集成，决定从客户浏览器发出的请求是一个静态文档还是动态文档。HTTP监听器直接提供静态文档请求服务，以及传统的通用网关接口（CGI）请求服务，并将每一件其他的事务传递给应用服务器层，然后应用服务器处理这个请求并将结果返回给监听器。

37.1.1 硬件与软件需求

要想在Solaris上安装OAS，需要满足一些硬件需求（参见表 37-1、表37-2、表37-3）。

表37-1 硬件需求

硬 件	需 求
CPU	一个SPARC处理器
内存	128M
硬盘空间	300M
交换空间	256M
CD-ROM设备	RockRidge格式

表37-2 软件需求

软 件	需 求
操作系统	Solaris2.5.1及以下的修补程序包 103640-08（或更新版） 103566-08（或更新版） 103582-08（或更新版）

(续)

软 件	需 求
浏览器	Netscape连接器： 4.04（JDK1.1补丁包） 4.05（只有因特网版本） 4.06或更高

表37-3 授权软件

软件项目	版 本
>HTTP监听器	40位Oracle(v2.14) 128位Oracle(v2.14) Netscape企业服务器 2.01、3.0、3.51 Netscape Fastrack服务器 2.01、3.0、3.51 Apache 1.2.5
Oracle	7.3.2x 7.3.4 8.0.3 8.0.4 8.0.5
Java开发工具包（JDK）	1.1.4

来源：Oracle公司

注意 Oracle的文档并没有要求在应用服务器上独占使用Oracle数据库。然而Oracle应用服务器对Oracle数据库来说已经很成熟。

37.1.2 安装完成后重新链接可执行文件

除了软件需求与硬件需求之外，重新链接可执行文件也很必要。链接是将编译与库连接为可执行程序的过程。重新链接可执行文件非常必要，因为 Oracle应用服务器处理了一些通常是操作系统处理的职责。此外 Oracle应用服务器有它自己的头文件，需要将它并入任何需要这个应用服务器的可执行文件中。一般情况下，任何必要的重新链接由安装自行处理，但如果想要在安装之后选择重新链接可执行文件， Oracle应用服务器要求安装以下文件、库与工具：

- Oracle库文件，具有.a扩展名。
- Oracle生成文件，具有.mk扩展名。
- Oracle目标文件，具有.o扩展名。
- 网络系统库。
- ar库控制工具。
- ld工具。
- 生成工具。
- X库。
- Motif库。

如果时间允许，最好对可执行文件进行一次完全重新编译。这样，可以在任何升级 /迁移异常来打扰你之前，就消除它们。

37.1.3 进行预安装工作

在Solaris上安装Oracle应用服务器之前，必须进行一些工作以确保一个平稳的安装过程，这些工作包括：

- 为结点管理监听器（以前称为 admin监听器）选择一个网络端口，这个端口使 Web监听器接收到的客户请求成为可能。这个号最小是 1024，但不能大于 65 535。缺省情况下，在UNIX上安装时是8889。
- 在UNIX下，必须设置以下环境变量： ORACLE_TERM、ORACLE_HOME、ORAWEB_HOME与ORACLE_SID。还需要知道你的机器的名字，通常是使用这个机器名存取文件与服务。例如，对 HTTP WEB机器，你的机器名可能是 www.csac.com或相似的名字。如果不能确定，询问你的公司或小组中创建域名与分配 TCP/IP地址的人。
- 创建admin口令，这是当连接到基于Web的配置网页时浏览器需要的口令。
- 需要具有一个操作系统帐号以真正拥有 Web监听器的UNIX帐号。由于安全原因，使用Oracle用户帐号作为这些进程的帐号不是一个好主意，因为它会严重违背数据库的安全性。使用根用户或超级用户帐号会有同样的问题，但在多数情况下，这是不可避免的。例如，缺省的HTTP服务器端口是80，但只有根用户才能启动端口号小于1024的进程。然而因为可以从命令行启动监听器，为增加安全性，也可以选择以另一个用户运行结点管理器监听器，如较低权限的 nobody。有一个useradd命令帮助创建这种帐号。

下面的安装工作需要使用固定的 Oracle用户帐号进行注册。因为它假设你是一个具有授权安装这种系统，不需要口令。

37.1.4 设置预备环境变量

为服务器环境变量设置初始值有助于操作系统与 Oracle应用服务器交互。相应环境变量需要在安装前进行设定，其中一些需要在启动或管理应用服务器时设定。需要将必要的环境变量值放入固定的Oracle帐户的启动文件中，在可能的情况下，建议这样做。当然，如果没有在启动文件中设置它们，在Oracle帐户的.PROFILE或.LOGIN文件中指定它们。也可以在壳会话提示符显示时，设置特定于当前的壳会话的变量（参见表 37-4）。

表37-4 初始化环境变量值

Environment Variable	Sample Value
ORACLE_HOME	/u01/app/oracle/product/8.0.3
ORACLE_BASE	/u01/app/oracle
ORACLE_SID	MyID
ORACLE_TERM	xsun5
ORAWEB_HOME	\$ORACLE_HOME/ows/4.0
ORAWEB_SITE	website40
TNS_ADMIN	\$ORACLE_HOME/network/admin
LD_LIBRARY_PATH	\$ORACLE_HOME/orb/4.0. lib:\$ORAWEB_HOME/lib
CLASSPATH	\$ORAWEB_HOME/classes: \$ORAWEB_HOME/admin
PATH	..\$ORAWEB_HOME/bin: \$ORACLE_HOME

(续)

Environment Variable	Sample Value
	/bin:/opt/bin:/bin:/usr/bin
	/usr/ccs/bin:/GNU/bin/make
	\$ORACLE_HOME/orb/4.0/bin
	\$ORACLE_HOME/orb/4.0/admin/cgi
	:\$ORAWEB_HOME/admin/cgi
TMPDIR	/var/tmp
TWO_TASK	在安装软件时必须不能定义它

例如，使用C命令解释程序设置 Oracle_SID值，如下所示：

```
setenv ORACLE_SID MyID
```

在Bourne中，如下所示：

```
ORACLE_SID=MyID; export ORACLE_SID
```

当设置环境变量时，紧记以下几点：

- 3.0版本使你可以使用与数据库相同的 Oracle_HOME，但Oracle应用服务器4.0版本必须安装在不同的Oracle_HOME下（如果愿意稍后重建服务管理器与 SQL*Plus，唯一例外是Oracle8.0.4与Oracle8.0.5）。
- 如果设置了 Oracle_BASE，执行一个OFA（Oracle灵活体系结构）安装，如果不想进行OFA安装，确认 Oracle_BASE在安装之前没有设置。

37.1.5 设计创建文件的权限代码

在启动文件中设置权限代码， umask变量含有所需要的权限代码。在改变 umask值之前，在提示状态下输入下行命令查看它的内容：

```
$ umask
```

如果umask的值不是022，将它改为这个值。这个值告诉服务器哪个组或用户具有读与执行权限；WRITE权限不受影响。执行以下操作把umask设为022：

- 对于Bourne或Korn shell，在.profile中输入umask 022
- 对于C shell，在.login中输入umask 022。

最后，需要检察不同的用户启动文件以确认 umask变量已被设为022。

37.1.6 从起始文件更新环境

当环境状态改变后（即安装了新的节点等），需要升级环境信息。要升级环境变量，将启动文件载入内存或永久媒体，如下所示：

```
Bourne/Korn shell: $ . .profile
C shell: % source .login
```

注意如果在非永久介质，如内存中（在提示状态下）更新这些变量，当退出当前会话后，它们的值将不再被存储。如果你将它们存储在一个永久对象如文件中，必须执行启动文件以使这些值生效。

37.2 在Windows NT上安装Oracle应用服务器

在Windows NT上安装Oracle产品通常比在 UNIX上安装要简单得多，但基本的步骤是相

同的（参见表37-5、表37-6、表37-7）。

硬件和软件需求

表37-5 硬件需求

硬 件	需 求
CPU	一个与486 Intel相当或更高级的处理器
内存	128M
硬盘空间	200M
交换空间	256M

表37-6 软件需求

软 件	需 求
操作系统	Windows NT 4.0 服务Pack 3
浏览器	Netscape连接器： 4.04（JDK1.1补丁包） 4.05（只有国际版本） 4.06或更高的微软IE4.01，具有服务Pack 1

来源：Oracle公司

表37-7 授权软件

软件项目	版 本
>HTTP监听器	40位Oracle(v2.14) 128位Oracle(v2.14) Netscape企业服务器2.01、3.0、3.51 Netscape Fastrack服务器2.01、3.0、3.51 微软因特网信息服务器3.0、4.0
Oracle	7.3.2x 7.3.4 8.0.3 8.0.4 8.0.5
Java开发工具包（JDK）	1.1.4

来源：Oracle公司

在开始安装Oracle之前，确认已停止所有的Oracle服务，如数据库自身与MSDTC。

开始安装，将CD-Rom放入驱动器。如果支持自动运行特性，将开始自动安装，如果不支持这个特性，运行G:\NT_X86\Install\SETUP.EXE，其中G:是你的CD-ROM。

37.3 理解Oracle应用服务器的最新安装特性

为使安装容易一些，Oracle在Oracle应用服务器4.0版本中加入了一些新的安装改进特性，这些特性包括如下：

- 安装组件 —— Oracle提供了一些安装工具，包括 HTTP监听器，Web请示代理，基于CORBA的对象请求代理、日志工具及一些以插件服务器形式显示的不同的编程语言环境。
- owsctl工具 —— 这个工具提供了对每个应用服务器的监听器与相关对象请示代理（ORB）

进程的集中控制。在 3.0 版本上有所改进，原来由系统管理页使用的 HTTP 监听器现在从 Web 应用服务器层分离出来。以前在进行任何系统管理之前必须先启动 Web 请示代理，现在可以单独启动结点管理器监听器。

- 灵活的升级与迁移选项 —— 如果你以前的因特网服务器是 Oracle WebServer 2.x 或 Oracle Web 应用服务器 3.x，Oracle 应用服务器现在可以帮助你减少迁移的麻烦。如果是 Netscape 服务器，Oracle 应用服务器也可以进行帮助。然而，不幸的是，它似乎没有提供任何从微软的因特网信息服务器进行迁移的帮助。
- 单结点或多结点安装选项 —— 如果你的域只需要一个网络结点，可以选择单结点选项，如果需要在多个结点安装，具有这种灵活性：定制安装使它认出多于一个的结点。
- 灵活的 Oracle 应用服务器位置变量（ORAWEB_HOME） —— 回到 2.x 版本，当 ORACLE 应用服务器仍称为 WebServer 时，系统的位置依赖于在 ORACLE_HOME 环境变量中指定的原目录位置。在 3.0 与 4.0 版中，可以具有一个不依靠这个环境变量的位置。每个安装将有它自己的称为 ORAWEB_HOME 的环境变量域。

37.3.1 鉴别产品依赖性

需要考虑在使 Oracle 应用服务器可用的情况下必须具有哪些适当的产品。如果计划将 Oracle 应用服务器作为一个单机因特网服务器，不需要阅读本节。然而，如果计划在所有 Oracle 环境内将 Oracle 应用服务器作为一台服务器实现，必须阅读本节。为确保环境具有所有必要的组件，应该直接查看下面的列表，它包括软件组件及各自的版本。

- Oracle 7 服务器, 7.1.6。
- PL/SQL, 2.1.6。
- SQL*Net, 2.1.6。
- TCP/IP 协议, 2.1.6。
- 服务器管理器 2.1.3。

37.3.2 设计路径结构

指明对一个路径结构的需要在某些方面与面向对象设计的原则类似，必须知道基对象及任何从这些基对象下来的导出对象。实现错误的层次会产生以后的混淆并导致冗余的工作。这对路径更是真实的，特别是当服务器软件的老版本已过时，必须进行升级时。许多域都有创建与管理复杂路径结构的方针。Oracle 提供了最佳的灵活体系结构（OFA）以减轻与路径相关的管理与维护工作。清单 37-1 显示了推荐的主结点路径结构。

清单 37-1 Oracle 推荐的目录结构

```
ORACLE_BASE
  product
    oracle
      8.0.4 ($ORACLE_HOME)
        rdbms
          ows
            cartx
              plsql
                java
```

```
4.0
  bin
  lib
admin
  MyDBName
  ows
    MySite1
      httpd_MyMachine1
        owl.cfg
        admin
          svadmin.cfg
          svadmin.pid
log
  wrb.log
  plsql.log
wrb
  wrb.app
  site.app
```

这个路径结构符合最佳灵活体系结构路径结构。OFA将数据文件与配置文件和可执行文件分开，所以可以运行任何一个 Oracle 产品的多个版本，包括 Oracle 应用服务器。

37.4 安装Oracle应用服务器

安装Oracle应用服务器，遵循以下步骤：

- 1) 启动Oracle安装器。
- 2) 在回答完有关数据库系统（最好是 Oracle 8x数据库）的问题后，选择安装 Oracle 应用服务器。
- 3) 为Oracle应用服务器结点指定主机名与网络端口。正是监听器供给你将来用于配置应用服务器的页。在UNIX下，缺省的是8889，在Windows NT下，缺省的是8888。
- 4) 为admin监听器指明一个端口。与以前的版本不同，不是监听器管理你的服务器，而是监听器供给例子。在UNIX下，缺省的是10 000，在Windows NT下，缺省的是8889。
- 5) 指明Oracle应用服务器系统管理员帐户的口令，Oracle应用服务器系统管理员的用户名是admin。
- 6) 在UNIX系统中，作为根注册，然后运行 \$Oracle_HOME/orainst/oasroot.sh壳脚本。这一步非常重要，因为它调整了重要文件的所有者与权限，并启动结点管理器。在 Windows NT 系统中，重新启动你的机器。
- 7) 启动Oracle应用服务器。要这样做，打开你的浏览器，到 <http://www.yourdomain.com:8889>（或任何你选择的主机名与端口），点击OAS管理器。在右边的面板中，看到你的系统上的所有结点的清单。选择紧挨主结点的单选按钮，然后点击“开始”按钮，它是在结点清单上方的三角形的图标，看起来像VCR的“播放”按钮。在分离的窗口中列出单独的进程，可以在其中检错。

提示 OAS在安装时对DNS解决方案非常敏感，确信在启动前DNS处于运行状态。

37.5 使用Oracle应用服务器的结点管理器

在Oracle应用服务器3.0版本中，所有的HTTP监听器在启动时都需要应用服务器层——后

来称为Web请求代理处于运行状态——包括系统管理工具。当然，这意味着如果WRB出了问题，没有可以修复它的简单方法。当Web请示代理仍存在时——虽然形式改变了，Oracle使用结点管理器在4.0版本中解决了这个问题，一个简单的HTTP监听器不需要应用服务器运行。

结点管理器能够对整个Oracle应用服务器进行管理，这些页可能的系统管理任务包括如下：

- 启动与关闭Oracle数据库。
- 启动与关闭Web监听器。
- 启动与关闭所有的系统组件。
- 创建一个新的Oracle Web监听器并修改现有的Oracle Web监听器。
- 创建与修改PL/SQL或其他应用。

在安装完成后运行oasroot.sh脚本时，结点管理器自动启动。也可以稍后从命令行启动或停止它，使用owsctl工具或在Windows NT下使用OracleStartOAS4.0（Oracle_HOME、website_name）服务。要启动结点管理器，输入：owsctl start -nodemgr。为获取系统管理工具，打开浏览器并定位到如下URL（除非在安装过程中选择了一个不同的端口）：

UNIX: <http://www.yourdomain.com:8889>

NT: <http://www.yourdomain.com:8888>

注意与以前的版本不同，这不是admin监听器。虽然这只是一个名字的改变，它仍执行admin监听器所执行的功能。

37.6 创建一个HTTP监听器

任何Web页请求的第一步是联络HTTP监听器。因为HTTP监听器在应用服务器操作中扮演了如此重要的角色，应该理解Web监听器需要的不同的配置参数。

要创建一个新的HTTP监听器，遵循以下步骤：

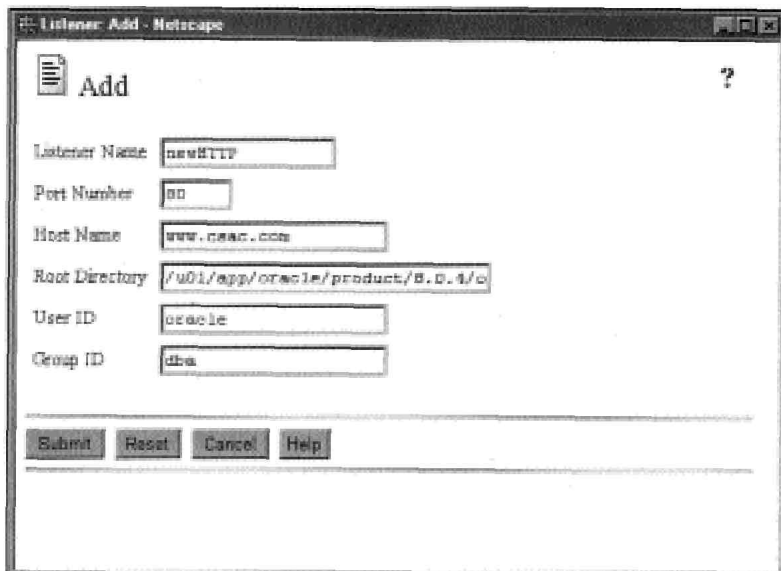


图37-1 创建一个HTTP监听器

1) 到OAS欢迎页, 通常在 <http://www.yourdomain.com:8889> (UNIX) 或 <http://www.yourdomain.com:8888> (NT)。

2) 选择OAS管理器。

3) 当窗口左边的小程序出现后, 双击以展开主菜单。

4) 选择HTTP监听器, 它在窗口右边显示出所有的现存的 HTTP监听器。点击右边部分的+号, 增加一个新的监听器。选择适当的结点, 点击“Apply”(参见图37-1)。

5) 填入适当的值, 并点击“Submit”(见表37-5)。

表37-8 Web监听器配置参数

参 数 名	缺 省 值	描 述
监听器名	无	此监听器的唯一标识符
端口号	80	它监听TCP端口。如果这个端口号小于1024, 需要根权限启动这个监听器
主机名	来自安装	此监听器应该知道的“因特网名”, 如www.csac.com
根路径	\$ORAWEB_HOME/doc	HTTP监听器试图找到文件的缺省路径。例如, 使用前例, 在这个路径下将找到 http://www.csac.com/index.html
用户ID	www	拥有监听器进程的操作系统用户 (仅 UNIX)
组ID	dba	在其下运行监听器的操作系统组 (仅 UNIX)

37.7 安装PL/SQL工具包

从Web浏览器获得Oracle数据库信息的最简单的方法是使用 Oracle应用服务器的服务器组件PL/SQL插件。要这样做, 只需简单书写调用 PL/SQL工具包的存储PL/SQL过程与函数。在以前的版本中, 除非作出计划, 否则这些包对每个由 Web获取的模式都会有一份拷贝。然而, 在4.0版本中, 可以在一个单独的模式中安装包并使其成为其他模式可共用的包。安装 PL/SQL 工具包, 遵循以下步骤:

1) 连接到Oracle应用服务器系统管理器, 点击 OAS 工具链接。

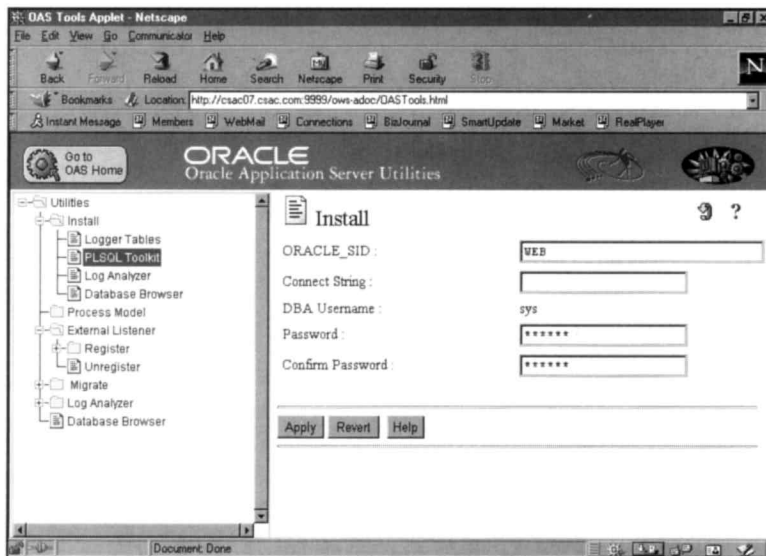


图37-2 安装PL/SQL工具包

- 2) 展开左边的“utilities”菜单与“Install”菜单。
- 3) 点击“PL/SQL工具包”。
- 4) 如果将工具包装在本地数据库中, 使用 ORACLE_SID, 否则, 使用 Connect String。
- 5) 输入 sys 用户口令。应用使用 sys 数据库用户, 因为它创建 OAS_PUBLIC 用户, 授予它权限, 并创建公有同义词。如果没有这个口令, 必须找到知道这个口令的人。幸运的是, 对每个数据库, 只需这样做一次。
- 6) 点击 Apply 按钮 (参见图 37-2)。

37.8 创建一个新的 PL/SQL 插件

PL/SQL 插件是直接由以前的“Oracle Web 代理”派生的, 是一个注册到数据库的 CGI 脚本, 该脚本将存储过程的输出返回给浏览器。然而, 在 4.0 版本中, 它更类似 OWA, 就像现代人类穴居人一样。现在, 它是一个发育完全的多线程应用, 甚至不需要与 HTTP 监听器位于相同的结点上。

当然, 功能越多也就越复杂。创建一个新的 PL/SQL 也就不像应该的那样简单, 必须进行以下操作:

- 1) 获取结点管理器。
- 2) 点击 OAS 管理器。
- 3) 展开左边的主菜单, 并点击 Application。
- 4) 在右边, 通过点击“+”增加一个新的应用。
- 5) 在 Application Type 下, 选择“PL/SQL”, 确信单选按钮已被手工选中。点击 Apply 按钮。

6) 填入应用的名字与显示的名字, 这些项目是你的应用将分别在应用列表与菜单中的显示。选择一个适当的版本号, 如“1.0” (参见图 37-3), 点击 Apply, 然后 Add 这个应用按钮的插件。

7) 在插件可以存取数据库之前, 需要知道它正在存取什么数据库。这个信息存储在数据库存取描述器 (Database Access Descriptor, DAD) 中, 一个大型的概念上的组件, 它存取如用户名、口令、服务器名或 ORACLE_SID 之类的信息。点击 Create New Dad 按钮 (参见图 37-4)。

8) 填入适当的识别信息。用户名与口令是应用在存取数据库时要用到的。注意底部标志为“Store the username and password in the DAD”框, 可以选择不将它存储在 DAD 中。如果这样做, 当用户试图获取一个数据库中存储的页时, 它们的浏览器弹出一个窗口询问用户名与口令, 并启动应用要用到的那个。点击 Apply 按钮。

9) 既然已创建了你的应用, 再给它增加一个插件。插件名与显示名再次决定插件将在系统管理工具中如何显示。虚拟路径决定应用在浏览器中看起来怎样。例如, 在 3.0 版本中,

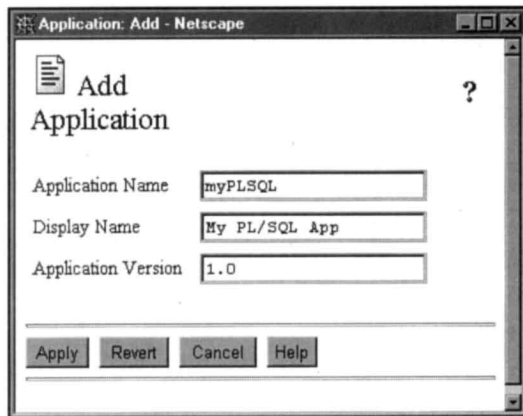


图37-3 增加一个PL/SQL插件

可能是mycart/plsql, 在1.0与2.0版本中, 可能是mycart/owa, 但只需简化为mycart, 而将物理路径保留为缺省的\$ORAWEB_HOME/bin。点击Apply (参见图37-5)。

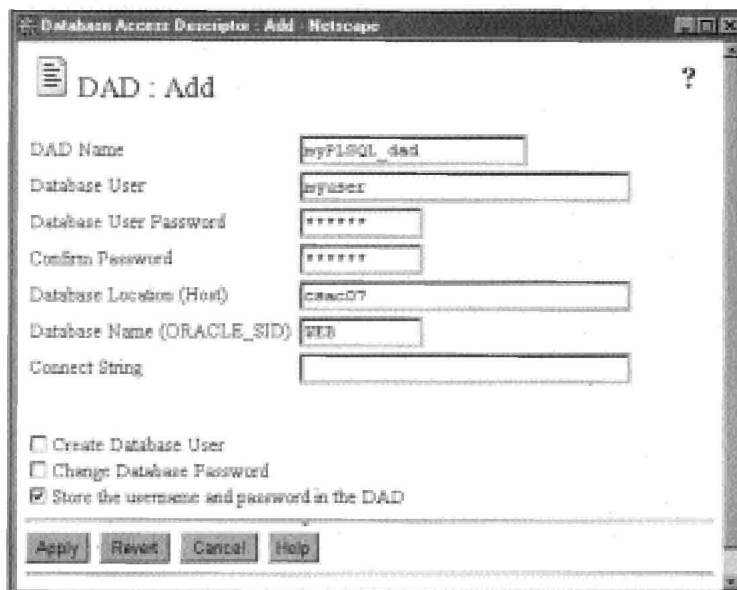


图37-4 创建一个新的数据库存取描述器

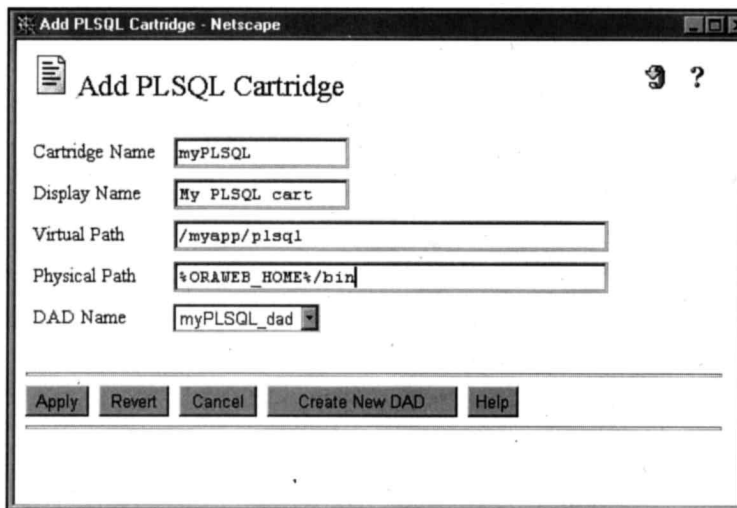


图37-5 应用一个PL/SQL插件

10) 重新装载应用服务器。点击左边的 Oracle Application Server, 然后选择all及reload按钮, 它看起来像一个VCR回放按钮。

要测试新插件, 创建如下的存储过程:

```
create or replace procedure hello_world is
begin
    http.p('Hello World!');
end;
```


通过将浏览器设置如下以获取它：

http://www.yourmachine.com/yourvirtualpath/hello_world.

37.9 疑难解答

因为Oracle应用服务器4.0是全新上市的，所以在运行中可能会有许多问题。其他的用户遇到了一些问题，因特网与 Oracle用户也遇到了不少问题。下面的段落包括了从个人经验、Oracle联机手册及其他因特网上资源中精选的信息。

1. 问题

结点管理器没有运行

解决策略

确定\$ORACLE_HOME/orainst/root.sh壳脚本文件已由根用户运行，可以使用 \$ORAWEB_HOME/bin/owsctl作为启动结点管理器的替代方法，使用下述命令：

```
<C1>owsctl start -nodemgr
```

2. 问题

HTTP监听器没有启动

解决策略

确认配置了从结点管理器进行监听的不同的端口。如果它小于 1024，确认具有根存取。

确认正使用新的4.0语法。例如，使用如下命令启动监听器 mylistener：

```
owsctl start -l mylistener
```

3. 问题

当试图获取一个新的插件时，收到“ URL Not Found ”消息。

解决策略

确认已重新装载Oracle应用服务器。

4. 问题

接收到错误消息OWS-05526:Service OWA_DEFAULT_SERVICE submission failed due to error 1034。

解决策略

在试图创建服务之前，启动 Oracle 8x数据库。

37.10 安装的其他有用提示

当准备打开电视并放松一下时，还有许多事情需要学。下面展示了一些在 Web上有用的建议与警告。它们处理 Oracle应用服务器的安装部分，Oracle应用服务器提供了也许需要经常修改的简化的、非常基本的缺省安装过程，查看以下的建议：

- 要得到最新的Solaris补丁程序包，到URL

<ftp://sunsolve1.sun.com/pub/patches/patches.html>.

- 也许需要使用第二个HTTP监听器。在这种情况下，对每个监听器，也许都想将 URL别名映射到Oracle应用服务器根目录。也想将这些别名合并入服务之中。

第九部分 并行环境与分布式环境

第38章 并行查询管理

本章要点：

- 介绍
- 并行加载
- 并行恢复
- 并行传播（复制）
- 并行SQL执行
- 可以并行的SQL操作
- 理解并行度
- 理解查询服务器进程
- 分析对象以更新统计
- 理解9,3,1算法
- 理解并行DML
- OPS环境下的并行执行
- Oracle8i中的改变
- 调整并行查询
- 疑难解答

38.1 介绍

具有大内存与CPU资源的巨型系统已经出现十几年了，然而，这些系统只能使用专有的操作系统，并且并不是非常有效的。使用UNIX开放系统体系结构的多处理器机器在90年代初出现，这些机器具有众多的硬件资源并相对较便宜。Oracle在Oracle7.1版本中引入了并行查询选项（PQO）以充分使用这些系统中可用的硬件资源。Oracle并行查询选项允许长时间运行的SQL操作（主要是查询），以协同方式在多个CPU间运行，这使系统减少了资源密集型SQL操作的运行时间。

并行查询选项使多个服务器进程可以并行执行一定的操作。进程，称为查询协调器，将一条语句的执行调配到多个服务器执行，协同所有服务器的结果，并将结果返回给用户。

虽然这个特性通常称为PQO（并行查询选项），它也包括如下内容：

- 并行加载。
- 并行恢复。
- 并行复制。
- 并行SQL。

38.2 并行加载

SQL*Loader直接路径使你可以使用多个SQL*Loader会话将数据同步加载到相同的表或分区中，如下例所示：

```
sqlload saledba/saledba control=sales1.ctl parallel=TRUE direct=TRUE  
sqlload saledba/saledba control=sales2.ctl parallel=TRUE direct=TRUE  
sqlload saledba/saledba control=sales3.ctl parallel=TRUE direct=TRUE
```

提示 在SQL*Loader命令行使用关键字parallel与direct以执行并行数据加载操作。

注意命令中关键字parallel与direct的使用，还要注意每个会话中三种不同的输入数据文件的使用。并行加载的重要特性是：

- 每个SQL*Loader会话分派了一个新的区间并将数据载入新区间中，为优化系统的 I/O 性能，强烈建议在 OPTIONS子句中使用FILE与STORAGE关键字控制新区间的位置与大小。FILE关键字可以在命令行或控制文件中定义，然而，存储子句只能在控制文件中定义。
- 并行加载要求没有本地或全局索引。如果存在任何索引，会产生一个错误信息并退出加载操作。在加载之前需要手工删除索引，在加载完成后，再重建索引。
- 每个加载会话需要一个表上的共享锁。
- 每个加载会话都是独立的，在加载会话之间没有任何联系。
- 当一个会话完成加载后，Oracle将加载的区间与已有的区间连接到一起，上次加载区间中没有使用的块作为空区间返回，这导致加载之后非标准大小的区间。即使通过加载控制文件中的存储子句选项指定了区间大小，截断也会发生。

如以前所指出的，每个加载会话向新的区间中载入数据，并不需要使用任何现存的区间，即使现存区间中不含有任何数据，因而，一个表的初始区间在并行载入时是从不使用的。

提示 为优化硬盘空间的使用，可以创建一个非常小的初始区间或不使用并行加载选项，将数据加载入初始区间中。

38.3 并行恢复

Oracle的基本读-写单位是数据块。每当对块进行修改时，Oracle将这些改变以重做日志的形式记录下来，如果需要时，可以使用这些日志重建这些块。如果由于介质失效或任何其他的原因而造成当前数据文件的内容丢失，那么这些文件可以从适当的备份拷贝中重建，然后进行恢复。恢复过程包括以下步骤：

- 1) 读日志文件并获得数据块的修改序列号。
- 2) 决定哪一个数据块需要改变。
- 3) 在高速缓存中读这些数据块。
- 4) 从重做日志中将相应的改变应用到这些数据块。
- 5) 将修改完成的数据块写回硬盘。

为在并行状态下进行恢复，需要设置初始化参数 RECOVERY_PARALLELISM。也可以使用RECOVER命令的PARALLEL子句代替。在并行恢复过程中，服务器管理器或SQLDBA会话读重做日志文件并将改变传递给并行服务器进程。然后这些进程读相应的数据文件并应用

改变。

提示 当在崩溃后重新启动一个Oracle实例时，在实例可以被用户使用之前执行崩溃恢复。如果需要以并行状态执行实例崩溃恢复，设置初始化参数 `RECOVER PARALLELISM`。

38.4 并行传播

复制使你可以维护多个数据库中的一个或多个数据库对象映象。Oracle服务器在这些数据库之间通过数据库链传送数据，以传播对复制对象的改变。SNP后台进程执行数据复制。然而，如果要复制的数据卷非常大，同步这个对象会花费很长时间。Oracle8使用并行复制，其中可以使用多个并行服务器进程传播事务。

38.5 并行SQL执行

Oracle数据库是一个由不同的进程维护的物理数据文件的集合。这样一套后台进程与共享内存段，统称为Oracle实例，使并发用户可以与数据库交互。当一个用户需要使用（`select`、`insert`、`update`或`delete`）数据库中的数据时，他需要连接到数据库。在大多数UNIX系统中，Oracle使用两任务体系结构。在这个模式中，当一个用户连接到数据库时，派生一个附加的进程，通常称为影子进程、前台等等。影子进程代表用户存取共享内存段与数据文件。

注意 在多线程服务器（MTS）环境中，当用户注册到数据库时，一个共享服务器进程用于代替影子进程。在并行查询操作过程中，这个共享服务器进程用作查询协调器。

Oracle在数据文件的块中存储数据。数据块是Oracle的基本数据输入/输出单元。当用户需要数据时，影子进程读取Oracle的内存高速缓存（共享内存段的一部分）中相应的数据块。然后在高速缓存中的块可以由用户的影子进程进行修改。修改过的数据块（脏缓存器）由DBWR进程写回硬盘。这样，一个影子进程做了用户需要做的主要工作，包括分析SQL语句、排序数据等等操作也由影子进程完成。

这在OLTP系统中工作得很好，在OLTP系统中用户维护的数据总数相对较小，然而在DDS环境中，用户通常需要处理大量数据，影子进程完成必需的工作需要花费一段时间。由于影子进程操纵数据工作时间很长，很辛苦，系统很可能有空闲的CPU与内存资源。这正是Oracle并行查询选项发挥作用的地方。通过使用这个选项，Oracle可以将一个用户的大量数据处理请求分解为多个、相对较小的工作单元，这些工作单元可以由不同的进程并发执行。它使用一套专有后台进程，称为并行查询伺服（服务器），完成这项工作。影子进程被提升为管理的角色，并称为查询协调器，查询协调器的职责如下：

- 1) 将功能分解为并行的小片。
- 2) 确保可以得到足够数量的并行查询伺服器。
- 3) 为给定的工作初始化伺服器进程，并在PQO服务器的进程之间分派工作。
- 4) 从伺服器进程收集输出并将输出返回。
- 5) 当完成预期的工作后，查询伺服器进程被释放，并可由其他的工作获得。

在最有利的情况下，并行执行减少了执行时间，执行时间的因子是使用的查询伺服器的个数。然而，由SQL语句消耗的总时间并没有减少。并行执行比顺序执行使用更多的内存，

所以如果机器没有足够的CPU与内存资源，不会得到预期的伸缩性。

38.6 可以并行的SQL操作

当一条SQL语句被执行时，它首先被分析，然后才被执行。在分析完成后与执行开始之前，优化器建立了执行计划。查询协调器进程检查执行计划中的操作以决定单独的操作是否可以被并行化。

在版本8之前，Oracle只能并行具有查询或DDL操作的SQL语句，如create index、create table、as select等等，它的执行计划包括一个现存表的全表扫描。在版本8中，Oracle为插入、更新与删除操作引入了并行执行。

Oracle8服务器概念手册指出Oracle可以并行执行以下的操作：

- 表扫描。
- 嵌套循环连结。
- 排序合并连结。
- 哈希连结。
- “Not in”。
- Group by。
- Select distinct。
- Union与union all。
- 集合。
- 从SQL调用的PL/SQL函数。
- Order by。
- Create table as select。
- Create index。
- Rebuild index。
- Move partition。
- Split partition。
- Update。
- Delete。
- Insert...select。
- Enable约束（表扫描是并行化的）。
- 星形变换。

除这些操作之外，Oracle8i还能并行执行cube与rollup操作。

作为一条规则，可以说Oracle可以并行执行任何含有全表（分区）扫描的操作。考虑如下的SQL语句：

```
select cust_id, sales_amt from sales_97 order by sales_amt;
```

执行计划由一个sales_97表的扫描组成，假设这些列上没有索引，随后是在sales_amt列上的排序。如图38-1所示，扫描与排序操作被分开并由多个进程执行。扫描与排序都由不同的进程并行完成，因而，PQO不只使一个操作能被多个进程完成，而且使在相同SQL语句中的多个操作可以并行执行而不是顺序执行。这是可能的，原因在于这些操作的创建者与消费者

的本质。在这个例子中，表扫描操作取得的行给予排序操作，不必等待扫描结束。

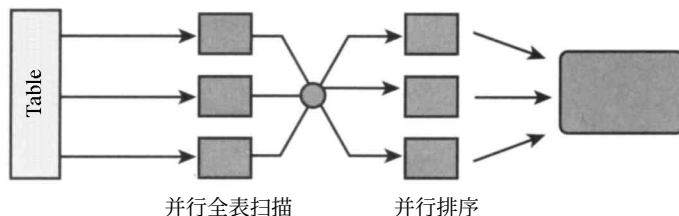


图38-1 并行执行包括一个全表扫描及排序

注意 将一个SQL操作在多个服务器进程之间分隔通常称为内部操作并行，而同步执行多个操作（从相同的SQL语句）称为交互操作并行。

在前例中，扫描与排序操作都有他们自己的服务器进程。因而，交互操作并行为每个操作分配了不同套的服务器。Oracle可以在并行状态下同时执行两个操作。如果SQL语句有三个或更多的可以并行的操作，第三个操作处于等待状态直至第一个操作完成。第一个操作使用的查询伺服器然后被循环由第三个操作使用。

38.7 理解并行度

一个SQL操作可以使用的并行查询伺服器的数量称为并行度。Oracle决定并行度依赖于不同的用户输入、实例初始化参数、表所在的数据文件的数量、系统CPU的数量等等。

38.7.1 决定并行度

一条给定SQL语句并行度的数量由以下方面按给定的顺序决定：

1) SQL语句级 —— SQL语句可以通过使用并行提示定义并行度，如下所示：

```
select /*+ PARALLEL (sales_97, 3) */
cust_id, sales_amt from sales_97 order by sales_amt;
```

在这个例子中，Oracle使用三个查询伺服器执行全表扫描操作，排序操作需要另外三个伺服器。因而，SQL语句总共使用七个进程 —— 六个并行查询伺服器，一个影子进程。

2) 对象级 —— 可以通过create或alter命令设置与表、簇、分区等相关的并行度。可以从数据字典的USER_TABLES视图的degree列查询现有对象的并行度，如下所示：

```
Alter table sales_97 parallel(degree,3);
select degree from user_tables where table_name='SALES_97';
```

第一条语句将sales_97表的并行度设为3，第二条SQL语句查询sales_97表的并行度。如果在SQL操作中包括多个表，每个表都有一个不同的并行度，最大的并行度成为这条SQL操作的并行度。在下例中，SQL操作使用并行度5进行操作。

```
alter table sales_97 parallel (degree 3);
alter table warehouse parallel (degree 5);
select /*+ parallel(s) parallel(w) */
w.location, s.sales_date, s.sales_value
from warehouse w, sales_97 s
where c.w_id = s.w_id
order by s.sales_value, w.location ;
```


3) 实例级 —— 如果Oracle没有在前两步获得并行度（即，如果用户没有定义并行度），使用缺省的并行度。

决定缺省并行度的过程与 7.3版本不同。在 7.3版本及更高版本中，Oracle使用可以获得的CPU与硬盘数量（文件，如果不能决定硬盘的数量）决定缺省的并行度，如下所示：

```
Default degree of parallelism = Min (#CPU in the system, #disk drives the tables  
is spread on)
```

如果使用RAID硬盘与OS条带化，那么一个数据文件会跨接多个硬盘条带化，Oracle无法知道硬盘底层的硬盘条带，并假设文件是在一个硬盘上。

在7.1版与7.2版中，Oracle使用缺省的最小并行度，如下所示：

```
Table size in number of blocks / PARALLEL_DEFAULT_SCANSIZE  
PARALLEL_DEFAULT_MAX_SCANS (init.ora parameter)
```

应该使用的并行度依赖于机器资源（包括 CPU、内存及系统的I/O带宽），数据如何散布、SQL语句并发执行的数量及其他系统上的负载。在决定并行度时需要进行充分考虑。具有给定并行度的足够的系统资源非常重要，否则会产生问题，如过多的页面调度、I/O瓶颈等等，那样反而会达不到目标。

38.7.2 当没有足够的查询伺服器时

当决定了一条SQL操作的并行度时，查询协调器试图征募可用（已创建但处于空闲状态）的服务器。如果可用的服务器不够用，查询协调器再创建服务器。如果由于MAX_PARALLEL_SERVERS的限制不能创建所需数量的查询伺服器，它为并行执行创建尽可能多的伺服器，并使用可用的伺服器。

注意 在7.3以前的版本中，Oracle用尽可能多的查询伺服器执行并行操作。如果得不到查询伺服器，查询由影子进程自己执行，即顺序执行。

在7.3版中，可以使用 init.ora参数PARALLEL_MIN_PERCENT指定执行查询所需的伺服器的最小数量。这个参数的缺省值是 0，它模拟 7.3以前的版本行为。可以为这个参数指定 0到100之间的任何数。当这个参数具有非零值时，Oracle以如下方式决定一个操作所需的查询伺服器的最小数量：

```
Minimum slaves required = Parallel_min_percent * Degree of parallelism / 100
```

如果Oracle得不到上面的公式所决定的数量的查询伺服器，它向用户发出一个错误信号，退出SQL操作。

38.8 理解查询服务器进程

当一个Oracle实例启动时，也启动初始化参数 PARALLEL_MIN_SERVERS确定的数量的并行查询服务器。如果在一个系统上运行多个 Oracle实例，每个实例都有它自己的一套独立的并行查询伺服器。与此相似，在 Oracle并行服务器环境下，每个实例有它自己的一套并行查询伺服器。如果在任何时间实例需要更多的查询服务器，就创建它们。然而，任何时间创建的并行查询伺服器的最大数量都不能超过 PARALLEL_MAX_SERVERS初始化参数。空闲服务器在空闲了 PARALLEL_SERVER_IDLE_TIME所规定的时间（以分为单位）后被关闭。然而，任何时间服务器的最小数量不能低于 PARALLEL_MIN_SERVERS。

在Oracle并行服务器环境下，一个SQL操作可以在多个实例中执行。在一条并行SQL执行中可以参与的实例的最大数量，由实例启动时的PARALLEL_MAX_INSTANCE init.ora参数决定。查询协调器也驻留在SQL语句开始时的实例上。

38.9 分析对象以更新统计

在并行执行过程中Oracle使用基于成本的优化。如果在数据字典中找到与表相关的统计，就使用它们。如果没有统计，就立即收集它们。如果统计的对象已过时，Oracle也许会使用错误的执行计划。需要经常分析对象以便保持最新的数据字典统计。而且，在表上进行任何主要的数据维护操作之后，都需要重新分析这个表。

38.10 理解9,3,1算法

Oracle对并行执行使用不同的算法。在版本8之前，它主要基于全表扫描进行并行操作。并行一个表的扫描的最简单的模式包括如下步骤：

- 1) 如前面一节中所描述的那样决定并行度。
- 2) 找到表的最高水印，因而找出表所拥有的数据块的总数。
- 3) 将要扫描的块分为与总的块数除以可用的查询服务器数相等的部分，现在已准备好为每个查询伺服器分配等量的工作了。然而，如果数据不能被平分或对部分数据的存取速度很慢，一些伺服器会先于另一些伺服器完成工作。为指出这个问题并优化性能，Oracle使用9,3,1算法。

4) Oracle进一步将每项工作分区分成三大块，第一部分是整个的9/13，第二部分与第三部分是整个的3/13与1/13。因而整个表被分成并行度为*3的分区。

5) 查询协调器现在将所有的9/13部分分给每个查询伺服器，当查询伺服器完成它们的早期分配工作后，接着分配3/13与1/13部分。这样确保几乎相等地使用所有的伺服器。

请注意9,3,1是Oracle用于并行执行的一个基本算法。Oracle并行执行还考虑了其他的因素，如下所示：

- 1) 在Oracle并行服务器环境中，Oracle工作划分算法还考虑了硬盘相似性等。
- 2) 当并行分区表与索引时，Oracle8由分区进行并行化。

38.11 理解并行DML

如前所述，Oracle也可以并行插入、更新与删除操作。为并行执行插入、更新与删除操作，需要执行以下命令：

```
alter session enable parallel dml ;
```

如果不执行这个命令，DML操作顺序执行，即使SQL语句含有清楚的提示或并行子句使其并行化，也要如此。

更新与删除操作只能跨越分区被并行化。然而，插入操作可以在分区内部被并行化。对更新与删除操作来说，最大的并行度还能超过分区的数量。下面的SQL语句从sales_97表中删除所有小于\$100的销售事务，它有三个分区：

```
delete /** parallel (s,3) */  
from sales_97 s where sales_value < 100 ;
```

下面是与并行查询和并行 DML 操作有关的提示：

- **PARALLEL** 可以在 SQL 操作中使用 PARALLEL 提示指明并行度。Oracle7 使你可以使用 PARALLEL 提示控制全表扫描操作的并行度。在 Oracle8 中，可以使用提示控制除表扫描操作之外的 INSERT、UPDATE、DALETE 操作的并行度。PARALLEL 提示的语法为：

PARALLEL (<table_name>, m,n)

m 等于用于 SQL 操作的并行服务器的数目，n 等于执行 SQL 操作的实例数目。仅在 OPS 环境下有用。

- **NOPARALLEL** 如果想要忽略与一个对象相关的并行度，并想让 SQL 操作以顺序方式执行，使用 NOPARALLEL 提示。
- **APPEND** 可以使用 APPEND 提示在 INSERT 操作过程中控制并行执行。这是并行插入操作的缺省模式，它对插入的数据使用新的空闲块，不使用块中现有的空闲空间。
- **NOAPPEND** 这个提示使你可以在为插入的数据分配新块之前，使用块中现有的空闲空间。
- **PARALLEL_INDEX** Oracle8 允许对分区索引进行索引范围内扫描的并行执行。可以使用 PARALLEL_INDEX 提示实现这个效果，它的语法与 PARALLEL 提示相似。

38.12 OPS 环境下的并行执行

Oracle 允许多个 Oracle 并行服务器（OPS）实例参与一条 SQL 操作的并行执行。如果有四个 Oracle 实例，而你执行并行操作的并行度为 2，每个实例使用两个并行查询伺服器。

可以使用以下方式控制参与的实例数。

- **Parallel 提示。**
- **Parallel 实例组。**

INSTANCE_GROUPS 允许将实例按预期的方式分组。PARALLEL_INSTANCE_GROUP 参数指定了执行实例并行操作的缺省组。PARALLEL_INSTANCE_GROUP 是一个动态参数，可以在会话级或系统级使用 ALTER SESSION/SYSTEM... 命令改变 PARALLEL_INSTANCE_GROUP 参数的值。当一个并行操作开始时，在执行中包括所有属于 PARALLEL_INSTANCE_GROUP 参数定义的 INSTANCE_GROUP 的实例。

考虑一个使用如下初始化参数的四实例 OPS 配置：

```
Instance 1 -> INSTANCE_GROUPS = g_12, g_13, g_14, g_123
PARALLEL_INSTANCE_GROUP = g_12

Instance 2 -> INSTANCE_GROUPS = g_12, g_23, g_24, g_123
PARALLEL_INSTANCE_GROUP = g_123

Instance 3 -> INSTANCE_GROUPS = g_13, g_23, g_34
PARALLEL_INSTANCE_GROUP = g_34

Instance 4 -> INSTANCE_GROUPS = g_14, g_24, g_34
PARALLEL_INSTANCE_GROUP = g_12
```

当一个用户注册到第一个实例并执行一个并行 SQL 操作时（没有执行 alter session set parallel_instance_group 命令），这个操作由实例 1 与实例 2 的并行服务器进程执行。然而如果用户在执行 SQL 语句之前，执行了

```
alter session set parallel_instance_group g_123 ;
```

操作由实例1, 2, 3执行。

38.13 Oracle 8i 中的改变

Oracle在Oracle8i中的并行查询选项方面做了一些改变, 但是这些改变不会影响并行执行的技术。多数改变用以自动确定并行度及调整进程。Oracle这么做的目的是使这个特性尽可能地对用户透明。将数据库管理员从调整并行操作中解放出来, 并让 Oracle在考虑到查询开始时可用的系统资源的情况下迅速完成这项工作。

提示 当你希望 Oracle决定并行执行的行为时, 设置 `PARALLEL_AUTOMATIC_TUNING=TRUE`, 同时设置 `PARALLEL_ADAPTIVE_MULTI_USER=TRUE`, 以便 Oracle根据空闲的系统资源决定并行操作开始时的并行度。

38.14 调整并行查询

并行查询操作是一个强有力的工具, 如果有效使用, 可以减少一些命令的处理时间。然而, 它很消耗资源, 并且所获得的性能提升高度依赖于数据分布。需要有足够的 CPU与内存资源以支持活动的并行查询伺服器。而且, 适当地计划数据以使在并行处理过程中不会有 I/O 瓶颈。

38.15 疑难解答

1. Oracle怎样决定并行操作的并行度?

通常, Oracle用如下方式决定并行度:

- 1) 首先考虑一个并行子句或 SQL 语句中的提示。
- 2) 下一个标准是一个对象 (表/索引) 在数据字典中定义的并行度。
- 3) Oracle使用可用的处理器与对象扩展到的硬盘驱动器数量决定缺省并行度 (参见 38.7.1 节以获取详细信息)。

4) 在Oracle 8i中, 如果`PARALLEL_ADAPTIVE_MULTI_USER`设为`TRUE`, Oracle可以根据查询开始时可用的CPU与硬盘资源减少并行度。

2. 在执行表上的并行操作之前, 是否必须分析这些表?

Oracle并行查询操作总是使用基于成本的优化。如果没有表统计, Oracle迅速收集统计。然而, 如果有统计, Oracle就使用它们。必须确保在进行主要的 DML操作之后已重新分析了表并更新了数据字典中的统计, 过时的统计会导致无效的执行计划。

3. 现在的表与它过去相比, 只含有很少的行, 使用并行查询的操作是否仍然需要更多的 (或相等的) 时间?

并行查询操作总是扫描表的当前 HWM、高水位 (含有数据的最高格式化块)。从表中删除行并不会降低高水位, 因而, 不会改进表中所包括的并行 SQL操作的性能。建议在大量删除操作之后重新建表以降低 HWM。

第39章 并行服务器管理

本章要点：

- 理解并行服务器的优点
- 单实例与并行服务器数据库的比较
- 决定何时并行服务器能够解决一个商务需要
- 为失效设计一个并行数据库
- 为可伸缩性设计一个并行数据库
- 并行服务器创建的特殊考虑
- 怎样监控与调整并行服务器
- 附加提示与注意

39.1 理解并行服务器的优点

Oracle并行服务器用于两种基本的应用类型：那些需要高可用性与那些可以与底层硬件一起移植的应用。为了让并行实例运行，需要比单实例数据库更多的组件。本章将介绍这些附加的组件都是什么，及怎样使用它们管理你的并行服务器数据库。

并行服务器是一个选项，允许两个或更多的实例同时装配在相同的数据库上。使用多实例，可以增加数据库资源的数量或净值数量。这些资源包括：

- 系统全局区，包括共享池空间与数据库调整缓存。
- 后台进程如数据库写（DBWR）、日志写（LGWR）与并行处理伺服器。
- 用户进程。
- SQL*net或Net8监听器。

图39-1显示了这些资源在并行实例中是怎样成倍增长的。如你可以想像的，这些增长允许数据库为更多的用户处理更多的工作，这也是多数人归因于并行服务器的最主要的好处。只要你使用了长时期伸缩性机制，当你增加更多的硬件结点时，你也可以启动更多的实例。

提示 并行服务器可以帮助增加你正在做的任务数，而对相同数量的用户，不会增加每项工作的持续时间或减少同步工作的总时间。如果工作可以在可用的实例之间分散以避免对可用资源的竞争，就可以实现这个目标。

可以使用不同的初始化参数配置在并行服务器数据库上运行的各种实例。这使你可以根据你的用户需求，为进行繁重事务处理的用户调整一些实例，以最好地支持批报表程序并优化联机决策支持活动。如果实例运行在具有不同能力的结点上，如内存大小、CPU数量或CPU能力，也需要分别配置它们。

另外，附加的实例可以为用户与数据库管理员提供安全覆盖。有许多事件（有计划的与无计划的），会导致Oracle实例停止运行。在单实例数据库中，一个停止的实例对用户及他们的应用关闭了所有对数据库的存取。在并行实例中，运行在一个停止实例上的用户可以重新连接到一个仍在运行的实例上。对 Oracle 8的用户来说，如果应用使用了特定的 Oracle调用接

口 (OCI) 调用, Net8 配置文件具有正确的条目, 这个重新连接可以自动实现。可以设计你的系统使用具有一个或多个富余实例的并行服务器, 以便在主实例不可获取时支持你的所有用户。

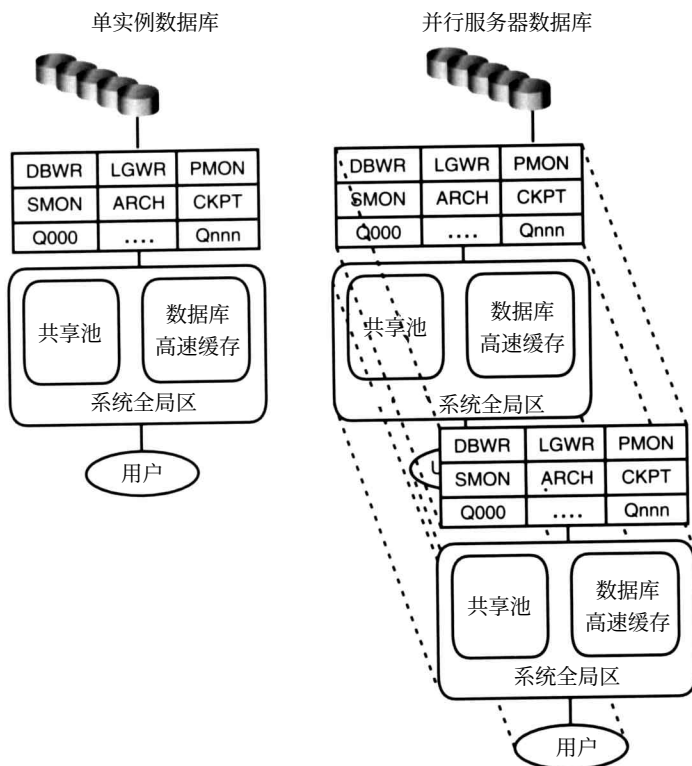


图39-1 单实例与并行服务器实例和资源

如果硬件可以支持足够的结点, 可以在相同的数据库上使用并行服务器的伸缩特性及故障恢复能力。只需简单地在—个或多个结点上保留实例, 用于故障恢复的情况, 而不用为性能的提升而同时运行它们。在设计多实例数据库时, 在这两个功能之间进行明确的区分是非常重要的。将工作转移到因为故障恢复的原因才保留的实例上, 在失败实例的用户需要使用它时, 将会导致竞争与性能问题。

39.2 单实例与并行服务器数据库的比较

为完全理解并行数据库的组织结构, 需要很好掌握数据库与实例的区别。数据库是由存储在硬盘上的一套文件组成的, 包括数据文件、重做日志文件及控制文件; 实例包括一套内存结构与进程, 主要的结构称为系统、或共享全局区 (SGA), 需要的后台进程是数据库写 (DBWR) 进程、重做日志写 (LGWR) 进程、系统监控 (SMON) 进程与进程监控 (PMON) 进程。如果运行 Oracle 8 (版本 8.0 或 8.1), CKPT 进程也是一个强制性的后台进程, 还有许多可以通过设置初始化文件中的参数启动的选择性进程。

Oracle 并行服务器需要附加的后台进程。其中一些由初始化参数启动, 另外一些自动启动, 与非并行实例的强制性进程一样。需要哪个进程, 可以使用初始化参数控制哪个进程, 依赖

于正运行的 Oracle 的版本，但至少总是需要一个负责管理实例之间封锁的锁（LCK）进程。在本章稍后，将看到 Oracle 并行服务器版本与控制特定后台进程参数之间不同的细节。

提示 需要做好准备修改单实例数据库使用的 init.ora 参数，以使并行服务器使用的多个实例可以成功地相互作用。一些参数在每个实例中都是相同的，另一些需要映射每个单独的实例的使用。

还需要知道如何在并行方式下启动一个实例。一个通常的排它的 Oracle 实例在启动时锁定控制文件，禁止任何其他实例使用。为设置这个锁，实例必须以并行方式或共享方式打开。在 7.3 版本中，通过在命令行状态的 STARTUP 语句中包括 PARALLEL 或 SHARED 关键字，或在 GUI 数据库管理工具的实例管理屏上使用 SHARED 单选按钮达到这个目的。在 Oracle 8 中，需要将初始化文件中的 PARALLEL_SERVER 设为 TRUE。

注意 实例管理 GUI 屏幕是 Oracle 的企业管理器产品的一部分，只能运行在 NT 平台上。也存在专门的并行服务器管理 GUI 工具，但它们也被限制在 NT 平台上使用。然而，使用其他操作系统的数据库也可以使用这些工具，只要为数据库服务器将 NT 平台配置为客户端。

39.2.1 厂商接口

并行服务器使用两个或更多的并发实例运行一个单独的数据库。硬件必须支持这种配置：允许独立的结点共享存储有数据库的硬盘。许多厂商在 UNIX 簇中支持多个结点，有一些早已提供了 NT 下的两个或四个结点簇。一些厂商支持使用专有操作系统的簇。在所有情况下，厂商必须提供软件去管理簇及对共享硬盘的存取。这可能是专有操作系统的一部分，或是一个或多个另外的产品。Oracle 保证并行服务器可以运行在可以成功与这些软件交互的平台上。有些情况下，Oracle 会限制并行服务器可以运行的结点的数量，即使厂商可以配置更多的结点。

为达到不同的实例可以相互交谈的目的，必须使用硬件的结点之间的连接与磁盘管理软件。Oracle 代码与厂商代码之间的接口随时间改变。在 7.3 版本中，Oracle 需要硬件厂商提供分布式锁管理器（Distributed Lock Manager, DLM），提供 Oracle 在不同结点的实例之间传递消息的机制，这种体系结构如图 39-2 所示。有些情况下，Oracle 与厂商紧密配合工作为 DLM 编写所需要的代码，其他早已具有簇管理软件的厂商可以与 Oracle 交谈而不需要进行任何修改。在 NT 平台上，没有发布 7.3 版本的并行服务器选项，直至 Oracle 8 产品在它的开发中加入了这个选项。因此，为避免为新的版本重写代码，并行服务器的 NT 版本是基于 Oracle 8 体系结构的。

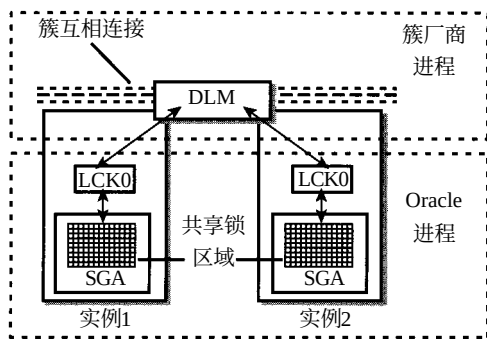


图39-2 7.3版并行服务器与DLM的接口

在 Oracle 8, 8.0 版本中，Oracle 将 DLM 集成为它自己的代码。集成的 DLM（IDLM）通过统称为组成员服务（Group Membership Services GMS）的一系列服务与厂商软件对话。GMS 负责管理簇、处理共享硬盘存取、代表 IDLM 在不同的结点之间传递消息。一些厂商甚至支持启动与关闭接口以便 Oracle 可以随之在每个结点上启动与关闭 GMS。8.0 版本配置的一个图表

如图39-3所示。

为改进Oracle的性能与能力以指明并诊断问题，设计师们在开发 Oracle8i 8.1版本的并行服务器中决定将大多数GMS函数合并入内核中。这需要对后台进程、LMON、对IDLM的监控进程增加代码。没有合并入IDLM的GMS执行的功能仍作为簇组服务（Cluster Group Services, CGS），由簇厂商提供。这个改进减少了 Oracle对厂商代码的依靠，簇实现只暴露极小部分，同时提供更无缝的产品用于管理。图 39-4显示了Oracle 8i 8.1版本的一张图表，簇存取的体系组件。

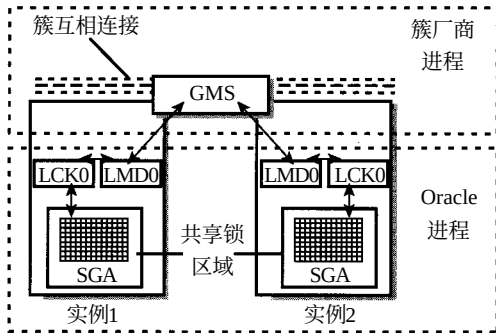


图39-3 8.0并行服务器与GMS接口

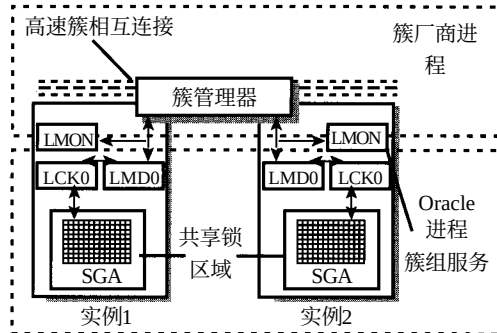


图39-4 8.1版本并行服务器与簇组服务的接口

不管正使用哪个版本，Oracle/厂商接口的详细资源依赖于厂商的体系结构。需要阅读与特定硬件、软件及Oracle版本相关的Oracle并行服务器安装手册，以找出这个配置的详细内容。也许必须运行一些特殊的厂商提供的代码，并在初始化文件中设置参数以配置特定版本所需要的后台工作。

如果运行在UNIX或NT操作系统上，当计划并行服务器时，必须考虑的最后一点是在这些平台上，必须将数据库文件旋转在裸盘分区上以便它们可以被多个实例成功地共享，包括数据文件、控制文件、联机重做日志文件。有些情况下，这非常有利，因为在硬盘写方面有一些性能改进，但对系统管理员管理而言，裸盘通常更难于管理。除了用于系统的操作系统文档之外，还可以在Oracle安装与配置指南中为特定的数据库版本找到裸盘的细节。

39.2.2 使用并行缓存管理锁进程

并行服务器使用并行缓存管理（Parallel Cache Management, PCM）控制不同实例对数据库块的存取。实例只有在以正确的状态拥有必要的PCM锁时才能存取一个块。为了使用INSERT、UPDATE或DELETE操作改变一个块的内容，实例必须以排它方式保持这个块的PCM锁。要查询一个块的内容，实例以共享方式保存必需的PCM锁。PCM锁根据基本需要在实例之间传递，从模式名中你可能会猜出，排它模式一次只能由一个实例拥有，而共享锁可以由多个实例同时拥有。PCM锁与行级锁是完全独立的，Oracle使用与单实例数据库相同的方式继续使用行级锁以确保在某一时间只有一个用户可以改变一行的内容。

每个并行实例控制一个LCK进程以追踪该实例PCM锁的状态，并在实例与DLM之间传达锁请求。就每个实例来说，每个PCM锁都有一个状态：

- NULL 当锁没有使用时。
- SHARED 当锁以共享模式持有时。

■ EXCLUSIVE 当实例当前拥有这个锁只为它自己单独使用时。

每个锁的状态存储在 SGA 区中的一个特定区域，称为共享锁区域。当用户需要一个锁，而他当前没有拥有这个锁，或是要求以排它模式拥有而当前以共享模式拥有这个锁时，LCK 进程负责将锁状态改变的请求送给 DLM。与此相似，当 DLM 接收到锁状态改变的请求时，它发出一个请求放弃实例上的 LCK 进程的锁，那里当前持有 PCM 锁。

DLM 负责所有实例的锁的全局分配，并管理不同实例发出的锁的管理请示。DLM 可以分配没有使用的锁或从另一个实例重新调用锁，并将它们传递给发出要求的实例。DLM 还维护所有锁的当前状态，不管它们是否已被分配给一个实例。在 Oracle8 以前，DLM 是由操作系统厂商或 Oracle 提供的，是与平台有关的。因而，你用于启动或停止 DLM 的命令以及需要配置它的任何文件或参数在不同的平台互不相同。一个 Oracle DLM 配置文件示例如清单 39-1 所示。

清单39-1 DLM配置文件示例

					SERVICE
#NODE NAME	IP ADDRESS	NODE ID	DOMAIN	PORT NUMBER	
apple	123.45.678.001	1	0	1544	
orange	123.45.678.002	2	0	1544	
pear	123.45.678.003	3	0	1544	

无论你正以分离的 DLM 使用 Oracle 7 或以 IDLM 使用 Oracle 8，都需要查询与平台有关的 Oracle 安装/配置指南获得特定的 DLM 配置信息。在 Oracle 7 中，这个选项趋向于依赖平台，因为 DLM 是由厂商提供的，而 IDLM 完全由初始文件中的参数配置。由 DLM、LMD0 与 LMON 使用的两个后台进程接大小排列时需要这两个参数。

向数据库数据文件分配 PCM 锁是多实例数据库系统管理员的主要任务。由于共享锁区域与 DLM 结构需要内存，过多的锁会导致性能退化。如果没有分配足够的锁，会导致大量的块碰撞询问（pinging）。

确定并阻止过量的块碰撞询问

当一个实例需要一个 PCM 锁，而这个 PCM 锁当前正由另一个实例以排它模式持有时，就会发生块碰撞询问。在释放这个锁之前，拥有这个锁的实例唤醒 DBWR 进程，检查由这个锁覆盖的块。DBWR 查看所有含有改变但还没有向硬盘刷新的块，并将它们写回到数据库文件。这个写脏块的进程作为 PCM 锁向下变化的部分称作碰撞询问。当增加由一个单独的锁覆盖的块的数量时，也增加了另一个实例可能需要这个锁而导致更多块碰撞询问的潜在可能性。

在碰撞询问的另一端是需要这个锁的实例。当接收到这个锁时，这个实例必须将它需要的数据块都读入缓冲器缓存，不能依赖于内存中早已存在的这些块的映射，因为其他的实例在它持有排它锁时可能已修改了这个块。如果接收实例需要在硬盘询问所有的块，这些询问被认为是真实的碰撞询问，但是，如果接收的实例不需要或只需要询问到硬盘的块，不必要的块写操作被认为是假的碰撞询问。参见图 39-5 查看这些不同类型的碰撞询问。

当实例需要定期存取确实相同的块时，真的碰撞询问可能会是无法避免的。假的碰撞询问组成不必要的额外开销，需要被检测出来并减少它们。你所经历的碰撞询问与假碰撞询问的总数依赖于数据库设计及初始化文件中 PCM 锁参数的设置。

注意 碰撞询问并不是坏事，而是并行服务器体系结构的一个必要的组件。过量的碰撞询问是一个问题，因为它降低了性能，本章讨论怎样设计一个维护可接受级别碰撞询问的数据库。

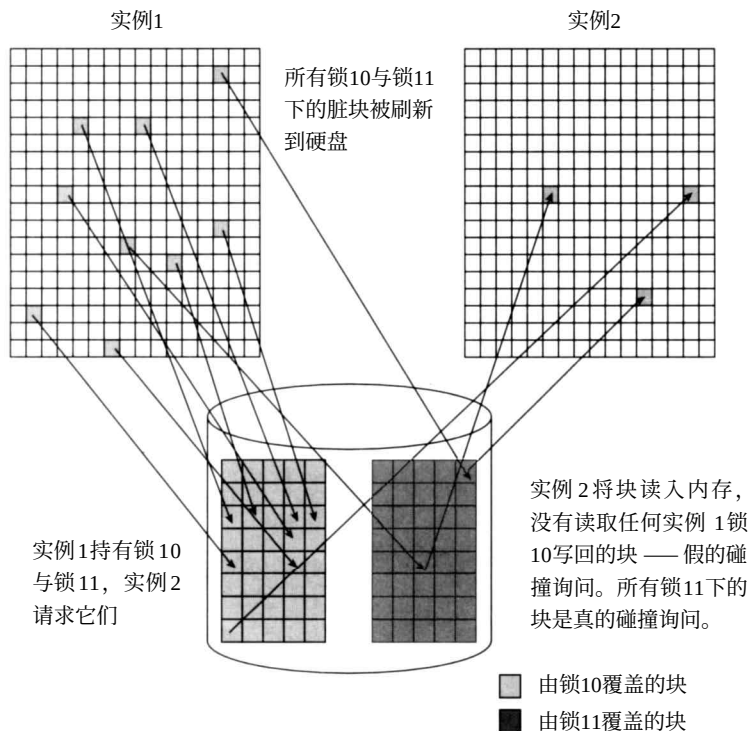
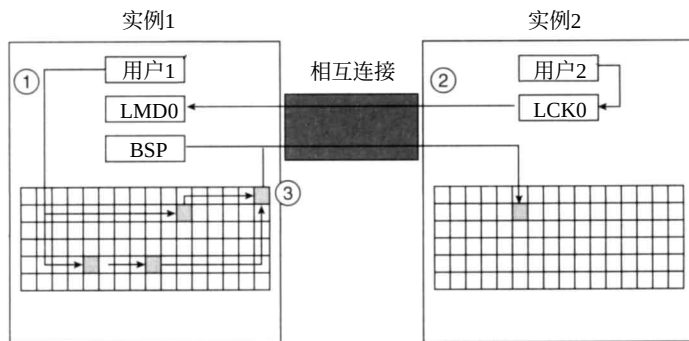


图39-5 碰撞询问在实例之间是怎样发生的

对那些正使用或计划使用 Oracle 8i, 8.1版本的人而言，一个新的特性——高速缓存融合 (Cache Fusion)，阶段I将帮助减少pinging。在Oracle 8i, 8.1版本中，DLM接受PCM共享锁的请求，并可以容易地构建请求的实例需要的读一致性块映象，它可以避免碰撞询问。与其说是将另一个实例要读的块写回硬盘，倒不如说是拥有排它 PCM锁的实例将使用它的块服务进程 (BSP) 后台进程建立所需要的块映象。它将使用任何所需的信息建立与请求实例与 IDLM 联系时刻一致的块映象。BSP会将这个块映象通过调整结点相互连接直接传送给请求的实例。这不只避免了资源仪器的硬盘读写，而且块映象拷贝可以不需要 IDLM改变原有实例上锁的排



- ①实例1上的用户1在一个排它PCM锁下改变一个块及相关的回滚块
- ②用户2请求块用于实例2的查询，实例2上的LCK0进程联系LMD0进程将PCM降级为共享状态。
- ③代替改变锁，BSP进程在实例1的内存调整缓存中建立了一个重做块，使用所有必要的回滚信息，并通过相互连接直接将整个块发送给实例1。

图39-6 高速缓存融合，阶段I怎样避免碰撞询问。

它状态就被传递。图 39-6 显示了高速缓存融合怎样在 Oracle 8i, 8.1 版本上工作。

39.2.3 使用并行缓存管理锁参数

使用一套全局高速缓存管理参数规定实例的 PCM 锁的环境。全局高速缓存管理参数名以字符串 GC 为前缀。反映先前讨论过的每个 Oracle 并行服务器版本中体系结构的改变, 这些参数在不同版本之间略有不同。表 39-1 列举了参数名与描述信息并指明它们在哪个版本中是合法的。

表 39-1 PCM 锁管理参数

参 数 名	描 述	7.3	8.0	8.1
GC_DB_LOCKS	哈希 PCM 锁的数量	X(3)		
GC_DEFER_TIME	在响应 ping 请求之前的等待时间 (1/100 秒)	X(3)	X(4)	
GC_FILES_TO_LOCKS	对数据文件分配 PCM 锁	X(3)	X(4)	X(5)
GC_FREELIST GROUPS	空闲列表组标题块的哈希锁的数量	X(3)		
GC_LATCHES	为门管理保留的锁的数量 (只在 8.0.4 版本)	X(3)		
GC_LCK_PROCS	锁进程的数量 (LCK0, LCK1, 等等)	X(3)	X(4)	
	Start			
GC_RELEASABLE LOCKS	好的精细 PCM 锁的数量	X(3)	X(4)	X(5)
GC_ROLLBACK_LOCKS	用于回滚段的锁的数量 (只用于版本 8.0 以前的标题块的哈希锁)	X(3)	X(4)	X(5)
GC_ROLLBACK_SEGMENTS	回滚段重做块的哈希锁的数量	X(3)		
GC_SAVE_ROLLBACK LOCKS	延迟回滚段块的哈希锁的数量	X(3)		
GC_SEGMENTS	用于非回滚段标题块的哈希锁的数量	X(3)		
GC_TABLESPACES	用于 ALTER ONLINE/OFFLINE 表空间命令的哈希锁的数量	X(3)		

可以将这些参数中的大部分设为它们的缺省值, 如许多 DBA 所做的那样。当然, 需要监控数据库性能以确保没有创建不必要的开销, 这些开销可以通过改变这些值避免。并行服务器最严重的问题趋向于由于选择不合适的 GC_RELEASABLE_LOCKS 与 GC_FILES_TO_LOCKS 参数值导致的过量的 ping 与锁变换。而且, 每个锁在 DLM 中消耗空间, 同时也在每个实例的 SGA 区内的共享锁区域消耗空间。如果定义了过多的锁, 可以使用用于 PCM 锁管理的内存, 这是其他数据库操作需要竞争的区域。

如在表 39-1 中所见的, 有两种类型的 PCM 锁: 哈希锁与精细锁, 也称为数据块地址 (dba) 锁。这两种类型的锁有一个重要的不同之处, 当一个哈希 PCM 锁是一个实例必需的时, 它并不将它释放回 DLM, 除非它是另一个实例必需的。因此, 一个实例很有可能含有一个哈希锁并直至实例关闭之前从不释放它。然而, 一个精细锁只在实例使用由该锁管理的块时才由该实例持有, 在此之后, 锁返回给 DLM 供另一个实例甚至是同一个实例使用 (如果需要)。

通常, 对查询集中的应用应该使用哈希锁, 因为实例可以共享模式获得相同的锁。只要块没有被任何实例更新, 这些锁便被每个执行查询的实例不确定地持有。可以定义这些哈希锁跨越许多块 (这减少了锁的总数), 因而减少了这个实例需要的内存。如果也定义了锁覆盖连续的块, 将通过读相邻的一套块改进全表扫描的性能。在事务密集型数据库中, 应该考虑使用精细锁, 这些锁的每一个覆盖一个单独的块或可能是很少的一些相关块。因而 ping 的可能性, 特别是假的 ping 的可能性, 明显地降低。

警告 不应该自动使用精细锁, 除非预期在实例之间会有严重的 PCM 锁竞争。精细锁产

生附加的消耗，因为它们在实例完成相关块后自动释放，即使另一个实例不需要它们。

当然，如果为了故障转移的目的在两结点系统中使用并行服务器，其中在任何时间只有一个实例会真正含有活动的用户，可以使用最小数量的哈希锁，而不管数据库活动的类型。锁在用于故障转移的目的而单独使用的并行环境下，不会提供它们通常的实例之间的功能。

怎样设置GC_RELEASABLE_LOCKS与GC_FILES_TO_LOCKS参数的值以达到要求的PCM锁分布呢？首先，应该清楚在 Oracle 7中，缺省的状态与 Oracle 8和Oracle 8i略有不同，在Oracle 7，7.3版本中，缺省的机制是哈希锁，参数 GC_DB_LOCKS决定文件可以获得多少个哈希锁，这些文件上没有使用 GC_FILES_TO_LOCKS参数分配的PCM锁。这个参数中没有列出的文件将使未分配的锁在它们之间平均分配，图 39-7显示了一个只有10个哈希锁的数据库配置——在该例之外似乎没有这样的数据库存在，将这10个锁由缺省的状态进行分配。图39-7已简化用于这个示例。数据文件中的第一个块没有必要由分配给它的最小号的PCM覆盖，而是使用了一个哈希算法。

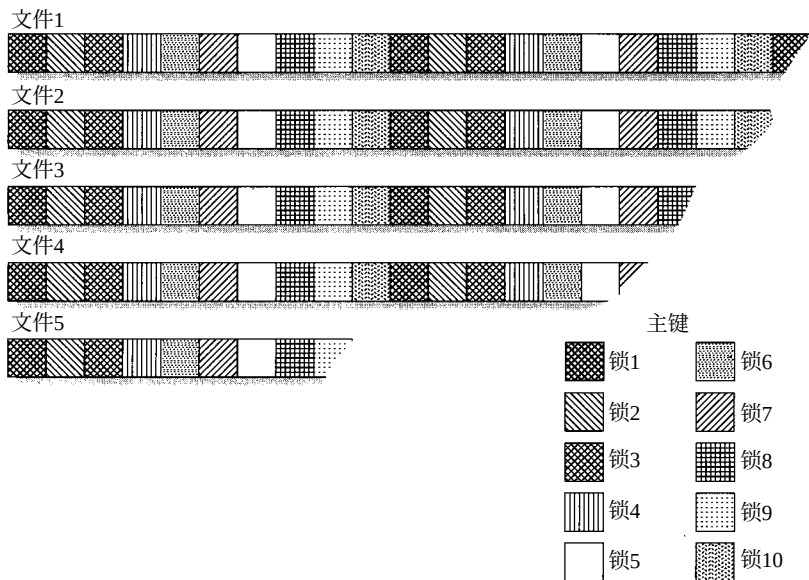


图39-7 哈希PCM锁的缺省分配

在Oracle8 8.0版本中，缺省的锁机制是精细锁，具有可释放的锁的数量与 DB_BLOCK_BUFFERS值相同。缺省情况下，Oracle8与Oracle8i的数据文件上没有预分配的锁。每当实例需要一个块时，可释放池中的一个精细锁分配给并覆盖这个块，当不再需要这个块时，锁被释放并可被下一个需要它的块获得。

当分配PCM锁时，不需要给以下种类的表空间分配任何锁：

- 只读模式。
- 临时模式。
- 只含有一个回滚段。
- 只含有一个临时段。

将这些表空间的文件ID号排除在GC_FILES_TO_LOCKS参数之外，以避免分配不必要的锁。

提示 通过将所有的只读段放入它们自己的表空间,可以将这些表空间改变为只读的,因此减少所需要的 PCM 锁的数量。与此相似,可以在为临时段定义表空间时,通过使用 TEMPORARY 关键字减少必须分配的锁的数量。这可以阻止在这些表空间中创建任何其他类型的段,以便安全地对它们的数据文件分配 0 个 PCM 锁。

另外,应该预定表空间只含有回滚段,因为你不需要对与其相关的数据文件分配 PCM 锁,只需要对回滚段自身分配 PCM 锁。然而,没有特殊的关键字确保这些表空间摆脱其他类型的段。

1. 在 7.3 版本中的 PCM 锁

如果在 7.3 版本中使用 GC_DB_LOCKS 参数定义哈希锁,它们将在数据文件之间平均分配(参见图 39-7)。你也许非常确定想要忽略这种缺省的组织,并按最适合你的应用与数据库设计的比例,对你的数据文件分配这些锁。使用初始化文件中的 GC_FILES_TO_LOCKS 参数完成这项任务。必须留有一些未分配的锁,以便 Oracle 具有一池备用锁以分配给没在 GC_FILES_TO_LOCKS 中指定的数据文件,或分配给也需要增加的数据文件。GC_FILES_TO_LOCKS 参数的语法如下:

```
GC_FILES_TO_LOCKS =  
➤ "file_list=locks:file_list=locks:file_list=locks"
```

这里

```
file_list    is a file id number, or multiple file id numbers separated by  
commas for individual files, or dashes for inclusive sets  
locks       is the number of locks.
```

其中, file_list 是一个文件 ID 号,或多个以逗号分隔的独立文件的文件 ID 号,或破折号,用于包括的集; locks 是锁的数量。

可以包括任何需要的不同的文件列表,以冒号分隔每个文件列表及锁计数值。注意条目的完整集必须括在双引号之内,并且在条目之内还能有空格。在此是一个合法的分配了总数为 1 005 个锁的参数示例:

```
GC_FILES_TO_LOCKS = "1=100:2-5=500:6,8,12=400:7,9-11=5"
```

文件 1 接受 100 个锁,文件 2~5 共享另外的 500 个锁,文件 6、8、12 分享另 400 个锁,文件 7、9、10、11 共享 5 个锁。图 39-8 显示了文件 7、9、10、11 上的五个锁是怎样分配给这些文件的,每个文件的第一个锁依赖于哈希算法。这个参数中没有列出的任何文件将由备用锁覆盖。这将是 by gc_db_locks 指派的但不是由 GC_FILES_TO_LOCKS 分配的锁平衡。在前面的例子中,将有 GC_DB_LOCKS 减去 1 0005 的备用锁,这些备用锁可以如图 39-7 所示以缺省循环方式分配给文件。

如果想让这些锁覆盖连续批的块,可以使用组运算符,惊叹号 (!),指出每组块的数量。例如,为使前例中的文件 7、9、10、11 上的每个哈希锁覆盖 10 个块,重新将参数写为:

```
GC_FILES_TO_LOCKS = "1=100:2-5=500:6,8,12=400:7,9-11=5!10"
```

图 39-9 显示这些块是如何分配给这些文件中的头几个块的。

另一个可以减少输入的参数串长度的选项是使用关键字 EACH,它对文件列表中的每个文件分配相同数量的锁。将前例重写为:

```
GC_FILES_TO_LOCKS = "1=100:2-5=500:6,8,12=400EACH:7,9-11=5!10"
```

你将分配总数为 1 805 个锁,文件 6、8、12 每个含有 400 个锁。

如果喜欢在 7.3 版中的数据库中使用可释放的锁,需要稍微修改一下全局高速缓冲区的参数缺省值。首先,需要设置 GC_RELEASABLE_LOCKS 参数需要的精细锁的数量。它的设置

值不能小于DB_BLOCK_BUFFERS参数的值。如果希望，可以将GC_RELEASABLE_LOCKS的值设得比DB_BLOCK_BUFFERS参数的值稍高一些。下面决定你是否想要使用所有的精细锁或是想要将精细锁与哈希锁混合使用。前者通过将GC_DB_LOCKS设置为0即可实现，第二个选项需要在将GC_DB_LOCKS与GC_RELEASABLE_LOCKS设置为非0值后，再分别设置每一类锁的各自的参数。

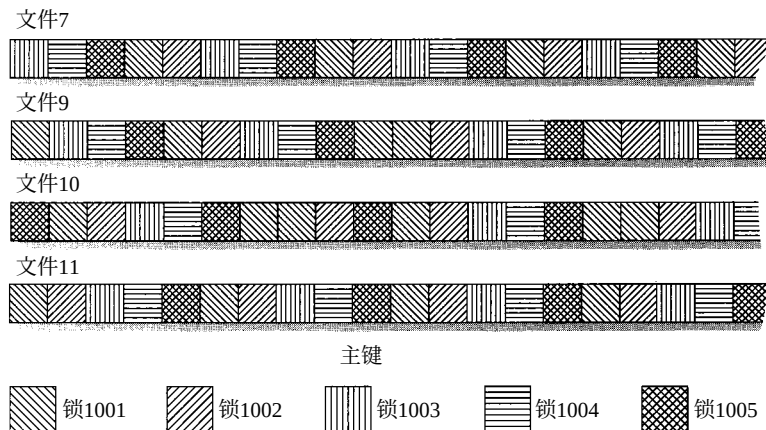


图39-8 GC_FILES_TO_LOCKS="...7, 9-11=5"的哈希PCM锁分配

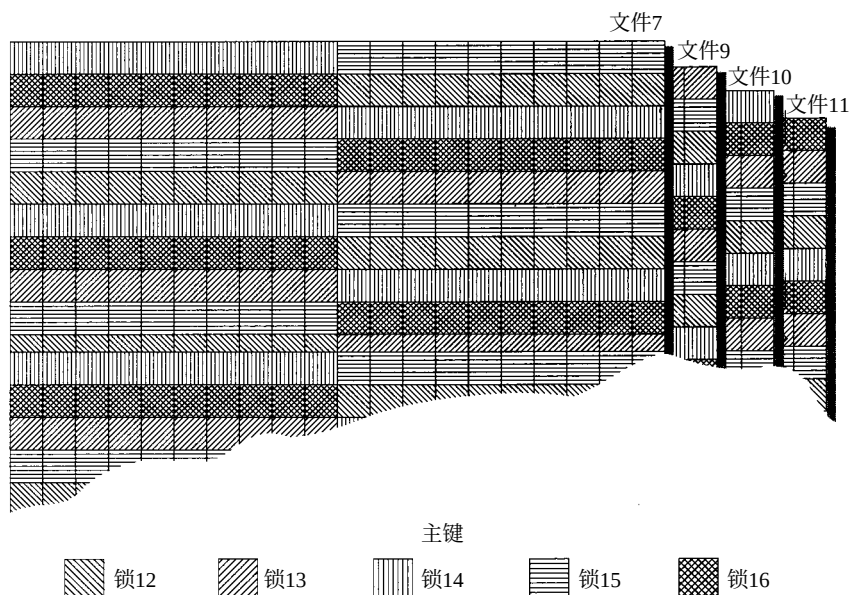


图39-9 GC_FILES_TO_LOCKS="...7, 9-11=5!10"的哈希PCM锁分配

要在一个特定类的锁上使用精细锁代替哈希锁，将相关的参数设为 0。表39-1列出了参数及可以控制的锁的类型。可以通过将GC_FILES_TO_LOCKS参数中的数据库段上的锁分配为0，为数据库段分配精细锁。清单 39-2是将所有段标题块上与文件 13~17一起设置为精细锁的参数文件摘录。

清单39-2 一个Oracle 7并行实例参数文件的示例条目

```

DB_BLOCK_BUFFERS = 1000      # Total block buffers
GC_DB_LOCKS = 500            # Total hash locks
GC_RELEASABLE_LOCKS = 1000    # Total fine grain locks
                                # Assign fine grain locks to:
GG_SEGMENTS = 0              # (a) segment header blocks
GC_FREELIST_GROUPS = 0        # (b) freelist header blocks
GC_ROLLBACK_SEGMENTS = 0      # (c) rollback segment header blocks
GC_FILES_TO_LOCKS = "13-17=0" # (d) datafiles 13 through 17
GC_ROLLBACK_BLOCKS = 200      # Assign hash locks to rollback blocks

```

清单39-2中没有包括的数据文件将分享哈希锁，有 500多个哈希锁，因为 Oracle使用内部算法增加了GC_DB_LOCKS中设置的值后面的数量。在这些锁中，200个分配用于回滚段重做块，只留下300多一点的锁以普通的循环方式分配给未列出的文件。

精细锁也许是对段标题块的最好的选择，如清单 39-2所使用的，除非具有一个非常繁忙的数据库，拥有大量的活动实例，例如一个运行联机事务处理（OLTP）系统的大量并行处理（MPP）。然而，分配给特定锁类型的哈希锁的缺省数量可能不够。可能改变的两个参数是GC_SEGMENTS与GC_ROLLBACK_SEGMENTS。前一个应该设定为数据库中的表与索引的总数，后一个设为回滚段的数量。

在7.3版本中，可以如先前关于哈希锁的讨论一样，分配好的精细 PCM锁覆盖连续批的块。为在GC_FILES_TO_LOCKS中设置精细锁的组特性，必须对锁的数量使用 0值，否则文件会回复为哈希锁定。下面一行是从一个参数文件中提出的，其中同时使用了哈希锁与精细锁，一些有组选项，一些没有组选项中。等号右边的 0值强制文件9~14与18上用精细锁，“！”导致锁在文件9~14中的20个连续块集及在文件 19~22上的10个块集上被分组。

```
GC_FILES_TO_LOCKS = "1=50:5-8=100EACH:9-14=0:20:15,18=0:16-17,19-22=1000:10"
```

2. 在Oracle8中的PCM锁

如在表39-1中所看到的，在 Oracle8数据库中可处理的全局高速缓存参数比 Oracle7数据库要少得多。然而，在怎样分配锁方面确实具有更多的选项。首先，应该注意到从版本 8.0开始，缺省的锁机制是精细锁而不是哈希锁，Oracle承担将锁分配给许多锁种类的能力。第二，哈希锁可以如在Oracle 7.3中一样运转，或者它们也可以承担精细锁的一些特性。第三，在回滚段重做块上的锁使用与 GC_FILES_TO_LOCKS参数相似的语法进行分配。在一个特定的锁类型分配给一个锁或一组锁之后，它的运转将与在 7.3版本中一样。一个非释放的哈希锁（缺省类型）不会被一个实例放弃，除非由代表另一个实例的 DLM要求这样做；一个精细锁或可释放的哈希锁在实例使用它完成工作后，立即释放。

在Oracle 8 8.0版本中引入的新的参数GC_DEFER_TIMER，如果设置正确可以帮助减少pinging。在Oracle 7中，或者将GC_DEFER_TIMER设置为它的缺省值0，一个实例对PCM锁的需求以他们可以处理的速度被很快满足。在有些情况下，两个实例试图同时使用相同的块，或相同的一套少数块时，将一遍又一遍地请求必需的 PCM锁，以完成一个单独的事务。结果的锁变换与块pings阻止了这两个事务以快速方式完成工作。通过将 GC_DEFER_TIME设为一个正数，数量单位是1/100秒，可以允许实例在满足DML请求释放一个PCM锁之前等待这个数量的时间，这使实例在请求实例释放锁与任何修改的块之前，可以有时间在锁下完成它的工作。

表39-1中列出的Oracle 8参数的其他改变在表中是不明显的。首先，有一个新的选项，允许指明想要释放哪个哈希锁，这需要在 GC_FILES_TO_LOCKS参数中包括字母R以指明这些锁（或者由它自己，或者作为 EACH关键字的前缀）。下例中的条目显示了这个新特性的合法的使用。

```
GC_FILES_TO_LOCKS = "1=100:4-6=500R:7,9=200EACH"
```

提示 如果想要减少一个实例在启动过程中，或在计划内或计划外的实例关闭之后的重新启动过程中的时间，对哈希锁使用R（可释放的）选项，这可以帮助你满足服务级的协议。然而，应该认识到，一个用户进程的每个可释放哈希锁的初始获得需要花费更长的时间，所以与在缺省方式下获得哈希锁相比，会有轻微的性能下降。

可以看到文件4~6分离的500个哈希锁是可释放的，与文件7~9分配的400个哈希锁一样，文件7~9中每个文件具有200个锁。

对哈希锁使用R（可释放的）选项不会导致这些锁与好的精细可释放锁表现得完全一样。一旦获得，这些锁是不可释放的，但在实例的生命期中被保留，并在实例之间传递，就像任何其他哈希PCM锁一样。这个选项的目的在于允许锁在第一次需要时获得，而不是在实例启动时获得。

警告 如果同时需要R与EACH选项，它们必须按所示的正确的顺序连接，表达式EACHR是非法的。

在GC_FILES_TO_LOCKS参数中的其他选项在所有当前的 Oracle版本7.3、8.0与8.1中没有改变。

第二个在Oracle 8中以新的形式显示的参数是GC_ROLLBACK_LOCKS。代替为这个值提供一个单一的数，需要使用一个与 GC_FILES_TO_LOCKS相似的格式为每个回滚段分配不同数量的锁。此处有一条约定：回滚段不能共享锁。当使用这个参数时，如果指明了回滚段的范围，必须使用EACH选项，如下所示：

```
GC_ROLLBACK_LOCKS = "1=50:2=100:3-10=200EACH:11-15=30EACH"
```

在这个参数中，等号左边的值是回滚段的ID号，不是如GC_FILES_TO_LOCKS中的文件ID号。可以通过查询DBA_ROLLBACK_SEGMENTS或GV\$ROLLNAME将ID号改为数据库中的回滚段的名字。这两个查询结果将显示重做段号（USN）或回滚段ID号及回滚段名。

最后一个在Oracle 8 8.0版本中引入的对现存全局高速缓存参数的改变是GC_RELEASABLE_LOCKS。这个微小的改变允许将值设置为比DB_BLOCK_BUFFERS小。现在可以将这个参数值设为在各个实例中是相等的，即使为了减少锁的开销，这些实例中的DB_BLOCK_BUFFERS的值是不同的。

警告 建议不要将GC_RELEASABLE_LOCKS设置为比DB_BLOCK_BUFFERS小的值，除非确定不需要所有的锁。只有两种可能情况：首先，如果已使用 GC_FILES_TO_LOCKS定义了所有可释放的锁同时覆盖内存中的多个块，第二，如果调整缓存中的块平衡总是被哈希锁覆盖的。如果没有足够可用的可释放的锁，所有的锁都在使用，应用将被迫处于等待状态，直至一个锁成为可用的，即使调整缓存中具有需要这个锁的块。

39.2.4 并行服务器初始参数

与前面几节讨论过的全局高速缓存（GC_）参数一样，为成功地管理并行数据库，需要在init.ora文件中使用一些附加的参数。表 39-2显示了可以在哪个版本中使用这些参数。

表39-2 PCM锁管理参数

参 数 名	版本		
	7.3	8.0	8.1
ALLOW_PARTIAL_SL_RESULTS			X(4)
CACHE_SIZE_THRESHOLD		X(3)	X(4)
DELAYED_LOGGING_BLOCK_CLEANOUTS	X(2)	X(3)	X(4)
DML_LOCKS	X(2)	X(3)	X(4)
FREEZE_DB_FOR_FAST_INSTANCE_RECOVERY		X(3)	X(4)
IFILE	X(2)	X(3)	X(4)
INSTANCE_GROUPS		X(3)	X(4)
INSTANCE_NAME			X(4)
INSTANCE_NUMBER	X(2)	X(3)	X(4)
LM_LOCKS		X(3)	X(4)
LM_PROCS		X(3)	X(4)
LM_RESS		X(3)	X(4)
MAX_COMMIT_PROPAGATION_DELAY	X(2)	X(3)	X(4)
OPS_ADMIN_GROUP			X(4)
PARALLEL_DEFAULT_MAX_INSTANCES		X(3)	X(4)
PARALLEL_INSTANCE_GROUP		X(3)	X(4)
PARALLEL_SERVER		X(3)	X(4)
PARALLEL_SERVER_INSTANCES			X(4)
PARALLEL_TRANSACTION_RESOURCE_TIMEOUT		X(3)	
SERVICE_NAMES			X(4)
THREAD	X(2)	X(3)	X(4)

1. ALLOW_PARTIAL_SN_RESULTS

如果这个值设为TRUE，将允许一个全局事务表的查询结束，即使不是所有组的实例都在联机。如果设为FALSE，不能获得查询组的一个或多个实例的查询将失败。应该将这个值设为TRUE，以在批处理程序中获取全局视图（GV\$_）以便它们不会失败。将它设为FALSE以使用这些视图来接收任何实例失败的警告。

2. CACHE_SIZE_THRESHOLD

任何表可以被缓存入一个单独实例数据库的缓存中的块的最大数量。缺省将它设置为DB_BLOCK_BUFFERS值的1/10，通常是一个推荐值。

3. DELAYED_LOGGING_BLOCK_CLEANOUTS

当事务提交时，该事务改变的块将被更新以指明在一个进程中的事务的完成，这通常称为块消除。有些情况下，清除与生成相关的重做信息被延迟直至块再次存取。当这种延迟清除发生时，块是脏的，如果另一个实例需要这些块，需要一个排它 PCM与潜在的ping到硬盘。通过将DELAYED_LOGGING_BLOCK_CLEANOUTS设为TRUE，可以阻止查询执行延迟块清除。因而，只有需要排它 PCM锁的DML命令将执行延迟的清除。如果将这个参数设为FALSE，可以执行延迟块清除，因而产生潜在的 pings。

4. DML_LOCKS

这个参数控制因并发事务修改的表中的排队。将所有实例中的这个值设为 0 将提高性能，但下面的语句将被禁用：DROP TABLE, CREAT INDEX 及显式锁声明（LOCK TABLE）。当不需要使用前述命令时，可以将这个值分配为 0。

5. FREEZE_DB_FOR_FAST_INSTANCE_RECOVERY

这个参数控制在实例进行恢复的过程中是否冻结所有的实例。如果设为 TRUE，Oracle 禁止所有的硬盘读写，除了那些与恢复过程相关的硬盘读写。这允许恢复更快地完成，但在恢复完成前将有效地冻结所有的用户活动。当这个值设为 FALSE 时，用户可以继续在实例失效没有影响到的数据库部分上工作，但他们将与恢复过程发生硬盘存取竞争。Oracle 选择这个参数的缺省值的方法基于哈希锁与精细锁所覆盖的文件的比例，如果所有的联机数据文件都使用哈希锁，这个值就是 FALSE。

6. IFILE

IFILE 参数指明当前文件中包括的另一个参数文件。使用这个参数包括一个常用的含有所有参数的文件，这些参数在所有的实例上是相同的。你的特定版本的文档将指明哪些参数必须在所有的结点上具有相同的值。理想情况下，将这个文件放在一个共享文件系统上以使所有的实例都可以读取一个唯一的拷贝。

警告 所包括的参数文件将被逻辑地插入到当前 init.ora 文件中包括 IFILE 参数的地方，如果在这个包括文件中的一个参数在当前文件的后面又重复定义了一遍，实例将使用当前文件的值，而不是所包括的文件的值，因为它是最后一个读到的值。

最好将 IFILE 参数放在实例的 init.ora 文件的最后部分。如果使用了一些逻辑的参数安排组织参数文件，如按功能、按字母或按日期的改变顺序排列，需要包括 IFILE 参数两次；一次在它的逻辑位置带有注释，该注释在文件末尾被重复，另一次是文件的最后一项。

7. INSTANCE_GROUPS

INSTANCE_GROUPS 命名实例将分配的一个或多个实例组，组名可以是在 PARALLEL_INSTANCE_GROUP 参数中显式或隐式的标识。只有在 PARALLEL_INSTANCE_GROUP 中命名的实例，不管是在 init.ora 文件中还是使用 ALTER SYSTEM 或 ALTER SESSION 命令命名的，将被选中用于执行并行操作。

8. INSTANCE_NAME

当想要指明需要连接哪一个实例时，这个参数设置在 Net8 连接中使用的实例名。它的缺省值是在操作系统中设置的实例标识符（SID）。参见 SERVICE_NAMES 参数的描述以获取更多的连接选项的信息。

9. INSTANCE_NUMBER

INSTANCE_NUMBER 与这样的实例有关：这个实例具有手工分配的段中的空闲列表及在分配给表和索引的特定的空闲列表组的实例。如果表或索引具有比 INSTANCE_NUMBER 值分配的少的空间列表组，模数算法将用于指明使用的空闲列表组。缺省值是 0，在这种情况下，在启动时分配的实例号基于这个实例相对于其他实例的启动顺序。

10. LM_LOCKS

这个参数指明了 IDLM 使用的锁的数量，这个值不能小于：

资源 + (资源 * (结点数 - 1)) / 结点数

如果应用在试图执行 DML 时冻结了，需要通过在上述建议的最小值上增加这个参数的值，

来为DML增加更多的锁。

提示 Oracle8与Oracle 8i并行服务器概念与管理手册含有用于计算一个合理的LM_LOCKS与其他的IDLM参数（LM_）值的计算的详细解释。它们也提供了可以用于执行这些计算的示例工作表。

11. LM_PROCS

LM_PROCS控制可以使用锁管理服务的 Oracle进程的数量，这个值至少为（进程数+活动实例的数量）。

12. LM_RESS

LM_RESS定义了DLM层使用的资源的数量，这个值应该为

$$2 * (\text{ENQUEUE_RESOURCES} + \text{GC_FILES_TO_LOCKS} \\ \text{GC_RELEASABLE_LOCKS} + \text{GC_ROLLBACK_LOCKS})$$

建议不要将这个值减小到比上述表达式的结果小。如果不知道这个计算中的一个或多个参数，应该启动一个实例并查询 V\$PARAMETER表以找到所需要的值。

13. MAX_COMMIT_PROPAGATION_DELAY

这个参数设置了任何对的实例在同步它们的系统修改序列号（SCN）之前需要的最长时间。可以选择降低这种同步的频率而减少实例之间的消耗，但如果这样做，将会冒这样的风险：在一个实例中已提交的修改在另一个实例中却未被提交的，这是因为 SCN用于确定在修改发生之前还是之后开始查询。通常，缺省值已经足够了，应该使用缺省值，除非查询看不到应用所要求的足够的当前的修改。更快的传播比例会伤害性能。

14. OPS_ADMIN_GROUP

这个参数用于向实例分配一个实例组名，以允许为系统管理与报表的目的指明实例的子集。使用GV\$_VIEW的查询仅返回与当前实例具有相同组名的实例的结果。

15. PARALLEL_DEFAULT_MAX_INSTANCES

这个参数指明了当具有 INSTANCES的DEFAULT值，在并行方式下处理的查询时，查询协调器执行一行查询所使用的实例的缺省号。这个参数的缺省值出自不同的操作系统的值，对大多数数据库来说，这个缺省值是最佳的。

16. PARALLEL_INSTANCE_GROUP

PARALLEL_INSTANCE_GROUP指明了以并行方式跨越多个实例执行的操作所使用的实例组。只有伺服器进程属于请求组，如由 INSTANCE_GROUPS参数所设置的，将用于执行并行操作。当数据库被分成相关实例的子集时，这个参数是有用的。但如果需要，可以使用 ALTER SYSTEM或ALTER SESSION命令覆盖它，以指明一个替代的实例组。

17. PARALLEL_SERVER

这个参数指明实例可以作为并行方式与其他实例共同打开，它替代了 7.3版本中STARTUP命令的关键字 PARALLEL，或选项 SHARED。如果设为FALSE，这个实例必须是排它的，是在数据库上运行的唯一的一个实例。

18. PARALLEL_SERVER_INSTANCES

把这个参数设置为希望在数据库上以并行方式运行的实例的总数。 Oracle使用这个值将SGA调整为这个数的大小。在多数情况下，比起使用缺省值来，这将会对内存进行更好的使用。

19. PARALLEL_TRANSACTION_RESOURCE_TIMEOUT

这个参数设置了在并行方式下执行的操作，在等待另一个实例的 PCM 锁或其他共享资源被释放或降级为一个兼容的级别时所需要的秒数。如果在这个指定时间内，不能获得指定资源，这个操作将成为过时的，这种过时将由实例之间的竞争表示，包括潜在的死锁。可以为这个参数分配 0 值，它将把过时时间设置为无限大，阻止了这种过时的发生。

20. SERVICE_NAMES

Net 8 使用服务名帮助在多个实例之间均衡用户的负载。这个参数指明了与一个实例相关的一个或多个服务名，用户请求 Net 8 连接到这些服务名中的一个，而不是连接到一个指定的实例。Net 8 将这个用户的连接分配给与命名服务相关的一个实例，使用一种算法决定哪个可用的实例是负载最小的。

21. THREAD

THREAD 指明实例使用的重做日志线程的数量。如果数量为 0，实例将试图获得一个空闲的公有重做线程。如果这个数是非零值，实例将试图获得这个数量的重做线程，不论它作为私有线程还是公有线程启用。如果需要的线程是可用的，实例需要获得它供其专用；否则实例将不能打开。关于重做线程以及私有和公有选项的讨论将在随后章节出现。

39.2.5 并行服务器的回滚段考虑

并行实例将与数据库上的其他实例操作共享系统回滚段。至少需要为每个实例增加一个附加的回滚段用于专用，否则实例将不能启动。在创建数据库之后，需要在一个专有实例中所做的是：必须使用这个实例增加所需的回滚段。当然，可以为每个实例增加一个以上的回滚段，遵循与在单实例数据库中相同的方针。

当为并行服务器创建回滚段时，可以使用私有段或公有段。虽然私有回滚段需要更多的初始化工作，但它们给予更多的控制并在长时间的运行中更易于管理。如果使用私有回滚段，这是缺省类型，需要在与它们相关的初始化文件的 ROLLBACK_SEGMENTS 参数中列出它们的名字。而对公有回滚段，需要在 CREATE ROLLBACK SEGMENT 命令中使用 PUBLIC 关键字，还需要把 TRANSACIONS 与 TRANSACTIONS_PER_ROLLBACK_SEGMENT 两个参数设置为要求的值。如果每个实例都需要相同数量的回滚段，可以对所有的实例使用一个包括参数文件，简单地将所有实例的这些参数设为相同的值。也可以控制每个独立的实例试图获得的公有回滚段的数量。Oracle 计算两个不同的参数 —— TRANSACIONS 与 TRANSACTIONS_PER_ROLLBACK_SEGMENT 的比例，通过将结果值上舍入到与其相近的最大整数（CEILING 函数），决定实例应该获得的专有回滚段的数量。

```
Number of public rollback segments required =  
CEILING ( TRANSACTIONS / TRANSACTIONS_PER_ROLLBACK_SEGMENT )
```

当使用公有回滚段时要非常小心，要为每个实例创建它需要数量的足够的回滚段。只要有一个回滚段是可用的，实例就可以启动。这会导致实例需要比预期的要少的回滚段，并产生一个性能较差的实例。如果由于一些实例需要比预期更多的回滚段而没有可用的足够的空闲回滚段，一个或多个实例会不能启动。

这种实例会以比需要的活动回滚段少的数量启动的倾向是推荐私有回滚段的原因。因为必须命名私有回滚段，你的实例只有在获得他们设计的回滚段时才会启动。如果不小心在两

个参数文件中命名了一个相同的回滚段，只有第一个启动的实例可以获得它，第二个实例在没有被分配到回滚段的情况下将不能启动。

推荐私有回滚段的第二个原因是它们的位置。当创建回滚段时，可以将它们分配给一个特定的表空间，将每个实例的回滚段放在它们自己的表空间中或一套表空间中是很有用的。当底层硬盘驱动器不可用时，这很容易识别哪个实例将受到影响。而且，当使用非共享体系结构时，应该将回滚段放置在实例的本地硬盘上。如果使用公有回滚段，你不能控制一个实例需要本地硬盘的回滚段还是非本地硬盘的回滚段。

警告 只有很少的Oracle并行服务器工具只支持公有回滚段。如果你正使用这样的系统，你的特定平台的文档将会指明。

39.2.6 重做日志与并行服务器实例

在并行服务器环境下的每个实例运行它自己的 LGWR进程。在大多数平台上，这些实例写它们自己的一套联机重做日志文件。有两个体系结构，其中日志将它们的输出写向一个集中的进程，这个进程交错将它们插入到一套单独的日志文件中。这种平台很少，可以从安装指南中轻易地决定你的平台是否属于这种平台。本节的其余部分重点介绍写它们自己的重做日志文件的实例。

一个实例使用的一套日志文件称为 thread。通过使用回滚段，可以将线程设为公有的或私有的。与回滚段相似，重做的公有线程由实例在启动时从可获得的资源池中获得，还能控制哪个线程由哪个实例获得。与公有回滚段应用相关的相同的考虑也适用于重做的公有线程。

当使用公有重做线程时，不需要知道哪个线程正由哪个实例写，因而，当经受硬盘失效时，也很难确定哪个用户将受到影响。在数据库扩展时，也不能为所有实例启用足够的公有线程，因而，在启动时会有比预期的要少的实例是可用的。如果正在一个什么也不共享的平台上使用并行服务器，应该使用私有重做线程将相关的日志文件放在每个实例结点的本地硬盘上。

必须在一个活动的实例中创建并启用重做线程。达到这个目标的命令是选择使用 ALTER DATABASE命令，如程序清单 39-3所示的那样。也应该注意到重做的一个线程必须在它的相关实例可以启动之前创建和启用。为每个实例的重做日志选择一个唯一的线程号，注意必须在整个数据库中分配唯一的重做日志组号，而不只是在每个线程中。

清单 39-3 定义一个重做的线程并使其可用

```
SQL> ALTER DATABASE ADD LOGFILE THREAD 2
2      GROUP 3 ('/ora/testdb/redo1/redo3a.log', '/ora/testdb/redo2/redo3b.log')
3          SIZE 1M
4      GROUP 4 ('/ora/testdb/redo1/redo4a.log', '/ora/testdb/redo2/redo4b.log')
          SIZE 1M;

Database altered.

SQL> ALTER DATABASE ENABLE PUBLIC THREAD 2;

Database altered.
```

与回滚段不同，使一个线程成为公有的或私有的选项是 ALTER命令的一部分，而不是

CREATE命令的一部分。为使一个线程成为私有的，在 ALTER DATABASE 命令中省略 PUBLIC关键字。如果需要改变一个已启用进程的状态，例如将它从公有线程转变为私有线程，首先必须关闭任何当前正在使用这个线程的任何实例，并从另一个不同的实例中进行这个改变。改变自身需执行 ALTER DATABASE 命令两次，首先使用 DISABLE THREAD 选项，然后使用所需的 ENABLE THREAD 选项。

如果实例不是做相似的工作，不必使所有的线程都是相同的。例如，可以在一个处理许多事务的实例的一个线程中具有更大的或更多的重做日志，面对读密集型实例使用更少的或更小的重做日志。如果决定使用这样一种变化的重做结构，必须启用私有实例，以确保每个实例获得适当的线程。如果计划你的实例相互之间作为故障转移实例，应该考虑为每个实例使用相同的重做线程。

39.2.7 使用空闲列表组以避免竞争

Oracle使用空闲列表识别在一个 INSERT 操作中，段中的哪个块含有用于新行插入的空间。空闲列表中的第一个可用的块被段标题块的条目识别，其他的块在空闲列表中由指针将每一个块链接到下一个块。缺省情况下，一个段的所有空闲列表开始于相同的标题块。当一个段含有多个空闲列表组时，为这个段创建一个附加的、特定的标题块。这些标题块的每一个含有起始指针，指向属于一个特定空闲列表组的空间列表上的块。

警告 虽然大多数关于空闲列表的讨论指明了它们用于 INSERT 操作，还应该清楚 UPDATE 也需要获取一个空闲列表。如果由于一行或多行长度的增长，行不再适合它的起始块，它将迁移到一个不同的块。对这种迁移行来说，如果没有给段分配块，将不得不从空闲列表中找出一个空的块。

并行服务器使用多个空闲列表组有两个主要的原因：第一是避免对空闲列表访问的竞争。在多用户数据库中，在某一时间只有一个用户可以控制一个空闲列表，所以需要多个空闲列表以允许对相同段的并发插入。基于数据库块的大小，有一个可以分配给段的空闲列表数量的上限。对每个活动的数据库与段，也许需要通过包括多个空闲列表组，每个具有多个空闲列表来增加更多的空闲列表。

由于一个常用段的空闲列表的信息存储在段的标题块中，所以也会发生空闲列表的竞争。当块从空闲列表中增加或删除时，标题块中的这条信息必须被更新。当不同实例上的空闲列表经常改变时，这个活动产生消耗，并由于标题块的 pinging 而导致严重的性能下降。

第二个使用空闲列表组的原因是控制实例插入时使用哪个块。如果两个或更多的实例共享空闲列表，它们很有可能试图将新的行放入相同的块中，由于每个空闲列表头上的块在实例之间传递，这将再一次导致过量的 pinging。通过使用空闲列表组，可以显式地或隐式地将一批块分配给指定的一空闲列表并确保那些块是由不同的 PCM 锁覆盖。然后每个实例只执行在它自己的空闲列表组的空间列表中的块的插入，避免使用分配给另一个实例的块。

第一种方法使用一个表，在这个表中，区间被手工地划分并分配给一个实例号，它隐式地将它们与一个空闲列表组相关联。当查找可用的块时，每个实例将使用与它在启动时获得的实例号相关的空闲列表组。对任何特定实例的插入将指向一个特定区间的块。通过这些区间放置在分离的数据文件中，使用 GC_FILES_TO_LOCKS 参数，可以确保为每个实例区间使用不同的 PCM 锁。结果是，PCM 锁与块在实例之间不再是必须共享的，相关的块也不需要

被ping。用这种方法建立的有四个空闲列表组的表如图 39-10所示。

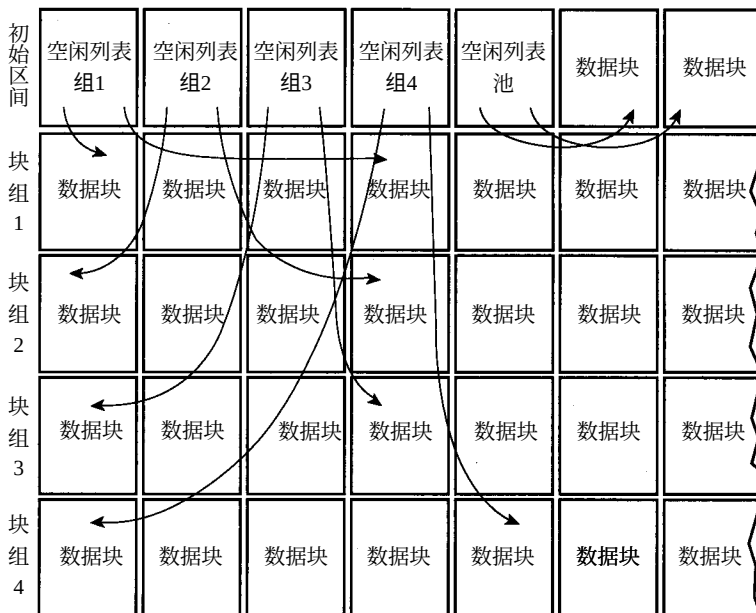


图39-10 具有四个空闲列表组的表

为了使用这种空闲列表组分配的方法，应该使用 STORAGE子句的FREELIST GROUPS选项建立这个表，并使用ALTER TABLE ... ALLOCATE EXTENT命令增加实例指定的区间。这个命令给予你将每个划分的区间分配给一个特定的数据文件与实例号的能力，需要监控这种方案中的空间使用情况，确保 Oracle没有动态地划分新的区间，因为你不能控制它们在哪个数据文件中创建，它们有可能会被分配给存储在主表标题块的普通空闲列表。也应该使用一个最小的初始区间，因为你不想记录插入到表的这个常用部分，应该将这个区间与实例的特定区间分开。图39-11显示了一个已建立的表仅支持两个实例在两个空闲列表组上的插入。注意如果两个实例需要访问这个区间，初始区间不只在它自己的数据文件中，而且使用精细锁以减少竞争。

为避免你没有指明分配给一个空闲列表组的块的竞争，应该考虑以下方针：

- 1) 创建只足够用于保存段标题块与空闲列表标题块的初始段。
- 2) 使用STORAGE子句为每个实例/空闲列表限制初始区间的+ 1的区间数。
- 3) 只为每个实例在它自己的数据文件中分配一个区间。
- 4) 监控空间的使用，当一个实例需要更多的空间时，只增加 MAXEXTENTS存储子句，在这种情况下，将MAXEXTENTS加1，并立即分配一个实例特定的区间。

在程序清单39-4中，可以看到怎样初始建立表，并为两个实例的每一个增加一个区间。

清单39-4 创建一个表用于两个实例共享

```
CREATE TABLE my_table (id NUMBER(10), . . . )
  TABLESPACE user_data
  STORAGE (INITIAL 10K MAXEXTENTS 3 FREELIST GROUPS 2);

ALTER TABLE my_table
  ALLOCATE EXTENT (SIZE 1M DATAFILE 'user_data_instance1_01.dbf' INSTANCE 1);

ALTER TABLE my_table
  ALLOCATE EXTENT (SIZE 1M DATAFILE 'user_data_instance2_01.dbf' INSTANCE 2);
```

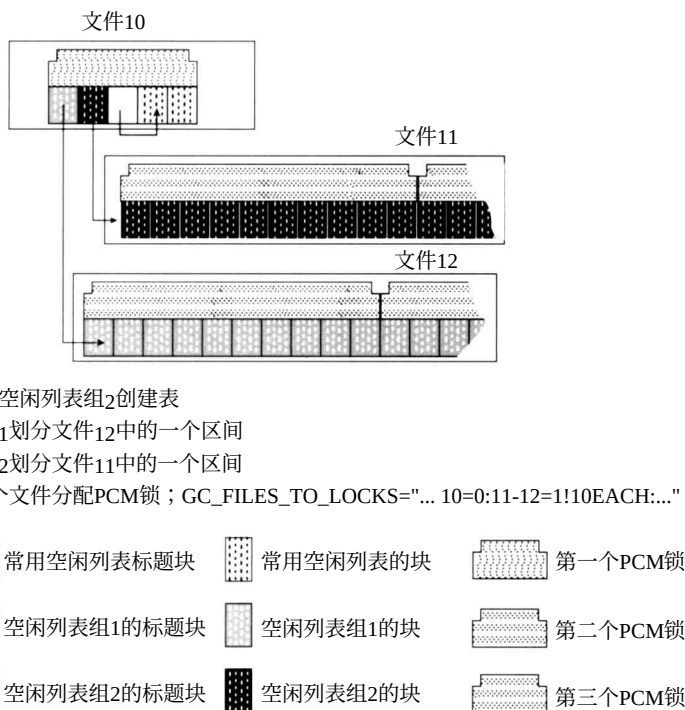



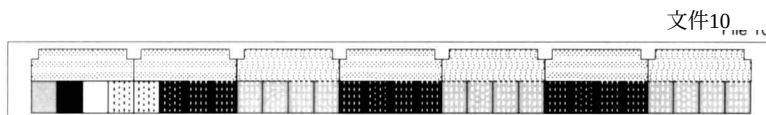
图39-11 空闲列表组的手工分配

第二种可以用于为不同的实例划分插入所用的块与 PCM锁的方法是使用精细锁上的组选项。为理解这点，需要一些背景信息。当 Oracle向段中插入项时，它将移动最高水位，一个段标题块中的指针到段中的最后使用的块。做这个工作以降低空闲列表必须跟踪的块的数量；任何最高水位之后的块逻辑上是不能使用的，所以不必包括在空闲列表中。通常，Oracle一次移动最高水位五个块。

然而，如果给数据文件分配了精细锁，段被划分到多个空闲列表组，Oracle改变了它的缺省状态。当达到当前的最高水位并需要移动这个水位时，Oracle通过为这个数据文件在 GC_FILES_TO_LOCKS参数中定义的数量移动它。另外，它为实例的空闲列表组分配了一个单独的可释放的PCM锁，覆盖所有最高水印之后增加的块。这保证了这些块可以被看到并只被它们分配的实例使用。

另一个需要表中空间的实例将再次移动最高水位，一个进程将为它的空闲列表组获得相同数量的附加块，并且一个单独的可释放锁覆盖住它们。这种每个实例最高水位的移动，及伴随的空闲列表与 PCM锁分配，保证了插入将只发生在与特定实例的空闲列表组标题块相关的块上及早已分配的PCM锁上。这个行为如图39-12所示，再次限制为两个实例的数据库。

当使用空闲列表组时，假设块总是只属于一个实例是不安全的。在块分配给一个空闲列表组之后，它还呆在它的相关的空闲列表中直到它已被填满到 PCTUSED的值，此时块将返回给进行这个修改的实例的空闲列表组。这个实例不必是开始在它的空闲列表中含有这个块的实例。与此相似，如果卸出然后载入一个具有多个空闲列表组的段中的行，行将被放入属于执行这个载入的实例的空闲列表组中。因而，它们不必要，也不会返回到它们的初始空闲列表中。



1. 使用空闲列表组2创建表
2. 给文件分配PCM锁；GC_FILES_TO_LOCKS=" 10=100!4"

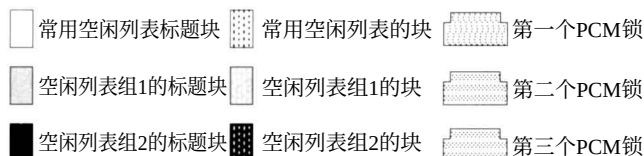


图39-12 自动划分空闲列表组与PCM锁

39.3 决定何时并行服务器可以解决一个商务需要

为决定并行服务器是否很适合你的超越故障的需要，需要将它的成本和复杂性与其他的选项进行比较。如果需要数据库每天 24 小时，每年 365 天都处于可用状态，将适合使用并行服务器。当至少有一个实例是活动的时，用户可以继续存取数据库。应该考虑使用并行服务器作为增长应用、数据库大小或用户团体的手段。

下面两节讨论在并行服务器上建造可用的与灵活性的应用所需要的工作。通常按下面各节中的描述进行工作，你应该可以决定你的环境是否适合使用并行服务器。一些问题很适合并行服务器的体系结构，而一些问题不适合使用它。为了做出最好的决定，应该完成这种替换策略的相似性分析，并比较它们的正面与负面结果。

39.4 为超越故障设计一个并行数据库

最简单的超越故障设置的组成是：一个两结点的簇，Oracle 实例运行在每个结点上，而数据库存储在一个共享磁盘系统上。缺省情况下所有的用户连接到相同的实例，同时具有连接到替代数据库的第二条路径。在客户 / 服务器环境下，SQL*Net 或 Net8，（依赖于 Oracle 版本）可以配置提供替代的、超越故障地址。在三层或 n 层体系结构中，一个 TP 类型的监控器可以路由用户到活动的实例。

这个模型可以被扩展，包括多个活动的结点与实例同时具有一个超越故障的实例，在此需要指出的是一个单独的结点与实例不能提供充足的故障保护。如果同时有两个实例失效——当结点增长时可能出现的增长的事件——一个单独的超越故障实例将不能处理所有失效实例的工作量。决定将哪个用户移动到哪个超越故障结点，它是衡量你的应用需要采取的分区分活动的子集。

不管有一个还是多个超越故障的实例，仍然需要建立 TNSNAMES.ORA 文件以指明主实例与超越故障实例，或在中间层组件中路由模式的其他代码，如 TP 监控器。为达到前一个选项，需要使用 Oracle 版本所允许的最好的选择。在最简单的所有版本中都具有的级别上，需要在 TNSNAMES.ORA 文件中 DESCRIPTION 子句使用一个 ADDRESS_LIST 及具有相同名字的实例。程序清单 39-5 列出了 TNSNAMES.ORA 文件的一个条目示例。

清单39-5 在TNSNAMES.ORA条目中指明主结点与超越故障结点

```
PROD.world=
  (DESCRIPTION=
    (ADDRESS_LIST=
      (ADDRESS=
        (COMMUNITY=tcp.world)
        (PROTOCOL=TCP)
        (HOST=apple)
        (PORT=1521)
      )
      (ADDRESS=
        (COMMUNITY=tcp.world)
        (PROTOCOL=TCP)
        (HOST=pear)
        (PORT=1521)
      )
    )
    (CONNECT_DATA=
      (SID=PROD)
    )
  )
```

Oracle 8增加了并行服务器的超越故障的能力。首先，具有启动一个为每个主数据库的用户连接到超越故障实例的并行、哑连接的选项。如果主实例失效，重新连接到备用实例将比如果没有创建预先产生的连接要快得多。如果不希望允许在主实例运行时，其他任何到备用实例的连接，应该考虑使用这个选项。

第二，Oracle调用接口（OCI）语言被扩展以允许断开与主实例的连接的用户——因为主实例失效的原因——可以自动连接到超越故障实例。另外，如果用户正在执行一条查询的处理，查询将在这种重新自动连接之下对它们重新启动。TNSNAMES.ORA文件含有一个FAILOVER_MODE子句以指明需要哪种类型的自动重新连接。新的关键字如表 39-3所示。

表39-3 为TNSNAMES.ORA提供重新连接选项的关键字

关键字名	功 能
BASIC	不使用在替代实例上的预先创建的连接
PRECONNECT	当切换到替代实例时，使用一已存在的连接
SELECT	在重新连接到替代实例之后，继续任何被打断的查询
SESSION	在实例失效之后，重新连接到替代实例
NONE	在任何主实例失效后，不试图进行自动重新连接

清单39-6展示怎样为Net 8建立一个TNSNAMES.ORA文件以提供一个从失效实例到替代实例的自动故障切换。

清单39-6 在TNSNAMES.ORA文件中定义超越故障结点

```
(DESCRIPTION=
  (ADDRESS=
    (PROTOCOL=TCP)
    (HOST=apple)
    (PORT=1521)
  )
  (CONNECT_DATA=
    (SIDGRP=fruit_group)
    (SERVER=DEDICATED)
  )
  (FAILOVER_MODE=
```

```
(TYPE=select)
(METHOD=basic)
```

如果决定利用OCI提供的超越故障选项的优势，应该清楚它们也可以用于实现一个软件超越故障。如果需要关闭一个实例，可以对用户透明地将活动的连接转移到它们的超越故障实例，使用如下命令

SHUTDOWN FAILOVER

停止当前的事务并用一个错误信息通知应用关闭告警。对没有计划的超越故障，这个技术依赖于OCI程序捕捉一个错误信息并调用超越故障连接。也可以使用如下命令将故障切换到一个单独的连接。

```
ALTER SYSTEM DISCONNECT SESSION 'sid,serial#' POST-TRANSACTION;
```

这个命令将再次使用一个错误通知应用，这个错误可以被 OCI程序捕捉并管理，在当前事务完成后，导致用户重新连接到超越故障实例。

不同于设置重新指向到替代实例的连接所必需的组件，为支持一个简单的故障切换选项不需要作任何数据库设计的修改。由于在某一时刻只有一个结点是活动的，所以不需要这样做的块与PCM锁共享。理论上，一个单独的锁对整个数据库已经足够了。然而，由于内部算法，将永远也不可能将锁的数量减少到这么少。通过将参数设置如下：

```
GC_DB_LOCKS = 1
GC_RELEASABLE_LOCKS = 0
```

将GC_FILES_TO_LOCKS留作它的一个空串的缺省值，将允许 Oracle分配可能的最小数量的锁。

39.5 为可伸缩性设计一个并行数据库

可伸缩性指的是系统具有按需要增长的能力，不论是支持更多的用户、处理更多的数据还是执行更复杂的任务。可伸缩性系统设计阶段的主要目标在于确保在实例之间具有最小的竞争。这对查询密集型 Oracle数据库来说是一个很容易的问题，因为所有的 PCM锁都可以由所有需要它们的实例以共享模式同时获得。对于事务密集型数据库，特别是 OLTP系统，这个挑战比较艰巨。不得不将数据库分区以便每个实例使用块的一套子集，而另外的实例使用它们自己不同的子集。第二个目标是将工作量在实例之间尽可能地平均分布，从而不会使一台服务器的资源工作过度而在另外的服务器上却没有充分使用它们。

有一些怎样分区数据库的选项，包括应用、功能、部门 /线型商务、物理表与事务分区。以下几节解释这些选项。

39.5.1 应用分区与功能分区

应用分区与功能分区具有相似的步骤，但在开始分区的商务级别方面有明显的区别。

应用分区包括将不同的应用运行在不同的实例上，如金融与制造业。通常，应用使用完全不同的表集，所以很少需要将它们连接到同一个数据库中。如果在应用之间的确具有一些相同的数据，需要在相同的数据库中同时运行这些应用。为并行服务器设计具有这种特性的数据库，需要使用与功能分区相同的步骤。

功能分区包括将一个应用的不同功能分区。例如，需要将人力资源应用分为四个功能——招募新人、雇佣历史、薪水册与奖金。如果公司很大，需要它的雇员和征募的新人可以通过电话线与 Web 存取本公司的数据库，这将导致一个具有高度事务比率的巨型数据库，这种应用需要运行在一个并行服务器上。

对应用分区与功能分区，需要考虑数据的实现而不是功能的定义。这需要分析数据库中的每个表将如何被每个应用或功能使用。为简化这项工作，可以在表中记录下你的发现。以人力资源为例，可以按如下进行：首先，创建一个与表 39-4 相似的表，定义每个功能使用哪个表。

表39-4 分区表——功能使用的表

招募新人	雇佣历史	薪水册	奖 金
Table 1	Table 1	Table 1	Table 1
Table 2	Table 2	Table 3	Table 2
Table 3	Table 4	Table 5	Table 4
	Table 6	Table 7	Table 5

下面记录下公用表——那些由多个实例使用的表。为简化这个例子，只考虑表的薪水册与奖金子集。下一步表的迭代如表 39-5 所示。

表39-5 分区表——功能使用的公用表

薪水册	表 公用表	奖 金
Table 3	Table 1	Table 2
Table 7	Table 5	Table 4

如果在应用中发现了一些公用表，需要重新考察这些表的定义以确定它们是否需要重新定义。目标应该是最小化公用表列中表的数量。如果没有公用表，就具有一个完美的分区应用。在这种情况下，将不同功能的表放到分离的表空间中，并为所包括的每个文件分配一个单独的哈希锁，甚至是与每个功能性表空间或表空间相关的文件集分配一个单独的哈希锁。每个功能的用户将连接到它们的相应实例。

对仍处于公用列中的表，下一步是调查并记录下每个实例在表上的活动类型及存取的频率。在表 39-6 中，插入了人力资源示例的值。

表39-6 分区表——公用表活动

公用表名	薪水册存取与容量	奖金存取与容量		
Table 1	SELECT	100/seconds	SELECT	300/seconds
Table 5	SELECT	200/seconds	SELECT	10/seconds
	INSERT	10/seconds	INSERT	0s
	UPDATE	20/seconds	UPDATE	200/seconds

现在可以准备设计表空间以减少 PCM 锁竞争与块 pinging 了。不是多个实例公用的数据库表可以被放置在它们自己的表空间中，只需要最小数量的 PCM 锁。这些表空间中的文件可以使用哈希锁覆盖，而不是使用精细锁，因为它们的竞争不会被其他实例共享。

在公用表列中的只读表——即，所有实例对它们只执行查询——也可以放置在它们自己的表空间中。应该在将数据加载之后就这些表空间改变为只读的，因为 Oracle 对不能被更新的数据不需要使用任何 PCM 锁。在 GC_FILES_TO_LOCKS 参数中略去只读表空间的数据文件，以避免分配不必要的锁。

现在可以处理由一个或多个功能使用，并至少有一个功能修改的表了。首先，需要将它们分类为低活动、中活动及高度活动的表。基于事务率的范围，应该可以定义哪个表属于哪个类。例如，如果最繁忙的表每秒有 100个插入与 600个更新，它的事务率为每秒 700个操作。如果最不稳定的表每秒有 40个操作，事务率范围为 660个操作（700-40）。将它分为三等分，即220个操作，把它们如下分类：

- 具有最多每秒260个事务的表是低活动率表。
- 具有每秒261个事务至480个事务的表是中等活动率表。
- 具有每秒481个事务至700个事务的表是高活动率表。

对高活动率表，应该使用分配了哈西锁或精细锁的表空间。如果唯一的活动是插入新的行，应该使用表及索引段上的多个空闲列表组，然后分配或者使用与手工分配区间联合的哈西锁，或者使用相关数据文件中 GC_FILES_TO_LOCKS参数的组因素分配精细锁。应该将中等活动率与低活动率的表分离到它们自己的表空间中，这些表空间相关的数据文件上具有哈西锁。

虽然以后会需要进行一些更好的调整，但最初应该选择使用三个简单的值，计算你应用到想要使用哈西锁的文件上的锁的数量。这些值将代表哈西锁的粒度，可以基于一些简单的比例选出。例如，对高活动率到中等活动率的数据文件，可以使用 1到10的比率，对中等活动率到低活动率的数据文件使用相同的比率。这些粒度数被数据文件中块的数量除，以计算覆盖这个文件所需要的哈西锁的数量。使用所提到的比率，可以选择高活动率文件的粒度为 10，中等活动率文件的粒度为 100，低活动率文件的粒度为 1000。

对任何给定的文件，使用文件中的块数除以文件内容中活动级别的相应粒度数。表 39-7 显示了使用前表中的值，对粒度的每个级别计算不同大小示例文件的锁的数量。

表39-7 基于活动级别与数据文件大小的哈西锁分配

块的数量	哈西锁的数量		
	高活动率 粒度=10	中等活动率 粒度=100	低活动率 粒度=1 000
1 000 000	100 000	10 000	1 000
5 000	500	50	5
1 000	100	10	1

如果使用前述步骤，仍需要一个大数量的锁以覆盖数据文件，就具有其他的选项以试图减少它们。可以对属于中等活动率与高活动率表或所有公有表的表空间的数据文件使用可释放的锁。也可以试用一种不同的分区机制或在早已具有的功能分区上应用一个附加的分区类型。如果具有太多的锁，系统无法处理它们，或者由于太多的锁而使实例无法在合理的时间内启动，你无疑需要使用上述这些技巧之一。

注意 如果增加了PCM与相关的DLM锁的数量，也许想要测试实例恢复的时间。锁的数量越多，DLM重新配置的时间越长，特别是如果一个整个的结点失效，停止了所有的实例和DLM在这个结点上的处理时，更是如此。

如果用户具有可以在故障之后空闲时间的最大数，需要减少所分配的锁的数量以满足他们的恢复时间的需要。

39.5.2 部门与线型商务分区

如果具有一些用户，他们试图获取他们自己的数据子集，即使这些数据处于同一个表中，也需要使用部门与线型商务（LOB）分区。为扩展上节中的人力资源示例，公司需要有一定数量的部门，其中人员由一个本地人力资源部门管理，或需要一个 HR组管理本地雇员，另一个HR组管理国际雇员。

如果存在这种固有的分区，可以对每个部门或 LOB使用不同的实例。由于这个分区模式的本质，很有可能所有节点上的用户都几乎执行所有的 DML，及对它们自己的数据的查询。继而，对每个共享表，只需要极少数的 PCM锁。达到这个目的的最好的方法是使用多个空间列表组，并为每个实例分配它自己的数据文件或数据文件集中的区间。有些情况下，你会发现在有些文件上只需要一个单独的锁，虽然有些行只是偶而被其他实例共享，但这不会提供足够好的性能。

39.5.3 物理表分区

如果不具有用户的固有子集，或有一个或多个所有用户都要修改的关键表，需要求助于表的物理分区。这不像是部门或 LOB分区那样一流的解决策略，因为还必须控制用户怎样连接到数据库。使用部门或 LOB分区，用户早已工作在不同位置或不同系统管理单元上，一个新的用户将属于这些组之一，并且可以被分配给相应的实例。为实现物理分区，需要一种机制以确保用户连接到正确的实例，依赖于他们需要存取数据的什么部分。

物理分区的一个例子是创建多个雇员表代替一个单独的表。对四个实例的数据库而言，需要拥有一个姓在字母 A~E范围内的表，一个姓在字母 F~J范围内的表，一个姓在字母 K~Q范围内的表，一个姓在字母 R~Z范围内的表。可以将这些表放在不同的数据文件中，甚至放在不同的表空间中。在 Oracle 7中，需要使用分区视图及完全分离的表以实现物理分区。在 Oracle 8中，可以利用分区选项建立一个具有多个分区的单独的表，尽管分区视图与物理分区表仍是可用的。

如果用户可以使用如表一样的分区范围进行分区，可以轻易地实现这个机制，这与部门或LOB分区极为相似。如果不同的人力资源部门雇员按照雇员的姓负责雇员，正是这种案例。一个分区模式与刚刚讨论过的一样进行工作，这个案例与 LOB分区的最主要区别在于当查询需要报告所有的或至少超过一个分区的雇员情况时。如果使用 Oracle 8的分区选项，Oracle 8解决了这个问题，因为只需要查询一个单独的表。在 Oracle 8之前，将需要使用 UNION 或 UNION ALL操作创建的分区的视图检索所有的记录。

如果配置了并行服务器数据库在运行这种查询时在多个实例上使用伺服器进程，在物理分区之上的查询可以是非常有效的。分区选项也允许使用多个实例上的伺服器越过所有的分区执行并行DML命令。在Oracle 8 8.0版本之前，不能执行并行DML。

如果不能按表的标准分区用户，需要应用基于正在处理的数据，将用户发送到正确的实例上，这甚至需要用户为每个事务进行一次新的连接以便它们总是基于有趣的记录工作在要求的实例上。然而这在一个标准的客户/服务器体系结构中是不切实际的，使用三层或更多层的体系结构建立这种应用的模型是可能的。在多层体系结构中，事务处理（Transaction Processing, TP）监控器位于用户与数据库服务器之间，控制客户的 SQL语句的路由，并将信息返回给用户。当一个请求没有指定实例时，TP监控器也可以帮助负载均衡，如果个别的实

例发生故障，TP监控器可以将请求路由到替代结点。

如果可以实现一个物理表分区，也可以应用一个哈西 PCM锁模式。行的初始存储与后继操作将由相同的实例执行，所以可以在含有这个实例分区的数据文件上使用一个单独的锁。另外，不需要多个实例组，因为只有一个实例将创建新的行。然而，使用部门或 LOB分区，将需要更多的锁，即使另一个实例很少需要使用这些行。

39.5.4 事务分区

如果不能如前面讨论的那样，将应用或表手工进行分区，也许需要试一试事务分区。事务分区需要非常复杂的应用，也许需要与 TP监控器一同使用。这一步需要每个事务发送到最好的实例处理它，基于事务的本质与所包括的表。如果所有的表都被分配给一个实例的锁覆盖，编写程序非常简单，但当一个事务包括的表分配了不同实例上的锁时，决策树会非常复杂，这个要求使这个途径成为一个艰难而昂贵的选择。

事务分区的一个好处是如果需要增加更多的实例支持增长，不需要对表结构进行修改。使用其他的分区模式，当增加一个新的实例时，通常需要对表或用户进行重新分区。使用事务分区，你修改程序而不是数据库。这听起来像是一项很繁重的工作，移动大量数据所需要的时间足够达到可用性的服务级协定了。然而，使用事务分区，可以在一个测试环境中预见到一个新事务时，进行程序代码的改变，并可用最小的停工时间实现。

39.5.5 索引与可伸缩性考虑

当为数据库表决定最佳的分区选择时，也需要设计可能的索引。为减少硬盘竞争，强烈建议将索引放在与父表分离的表空间中，使用不同的硬盘存放它们的数据文件。为在并行服务器环境下达到这个目的，使用早已存在的允许表分区的表空间，会使系统管理员非常头痛。因而，在某些情况下，将表与它的索引放在相同的表空间中是非常有益的。Oracle分区选项也为在相同的表空间中，存储表的分区与它们的本地分区索引提供了很好的原因。使用这种方法，相关的表与索引分区可以同时联机或脱机。

需要预见的另一个问题是在多实例数据库中索引的处理与线性增长的主键值的使用相关。主键通常是一个数字值键，由 Oracle序列生成器或其他使用一个增长整数值的机制提供值，如果两个或多个实例使用这种主键机制向一个表中插入记录，它们也需要向相同的索引叶子块中插入条目，这会导致这个块的连续的 pinging。为避免这个问题，有一些可用的机制。

可以创建实例值的指定范围，将值乘以实例的数量，使用如下的方程式：

$$\text{instance_number} * 10000000 + \text{sequence_number}$$

这个选项需要修改代码语句以使用索引获取行。当不得不增加行时，如果想要检索它，需要对数据做相同的改变操作。如果不是创建这个条目的实例想要检索它，这并不像听起来的那样简单。

第二个选项是为每个实例使用一个不同的序列。有许多方法定义这种序列生成器，虽然它们都有自己特定的缺点。最简单的两种方法包括对不同的实例使用不同的下限与上限的生成器，或使用不同的开始值与非单元步进值的生成器。例如，考虑一个四实例数据库。

在第一个例子中，实例1将分配的范围是1~1 000 000，实例2的范围是1 000 001~2 000 000，实例3的范围是下一个1 000 000个数，实例4的范围是4 000 001至以上。实例1可以使用一个

序列生成器定义起始为1，增长步长为4，因而会被分配数1、5、9、13等等，与此相似，实例2的生成器起始为2，步长为4，值为2、6、10、14等等。虽然这种方法不会超出现有实例的允许数，但如果在应用增长需要一个附加的实例时，这很难与增长的实例协调。

提示 如果想要使用具有不同起始值的序列，还想要为附加的实例作计划，可以使步长比当前的实例数大一些。例如，在我们的四实例例子中，可以将四个序列的步长设为10，实例1将产生值1、11、21等等，与此相似，其他的实例将产生它们之间的值，2、3、4、12、13、14、22、23、24等等。这留下了5至10，增长为10的数用于新的实例使用。

最后一个选项是使用 Oracle8 的保留键索引特性，它将保留索引中每列的字节。这个方法的好处是在 SQL 语句的断言中，仍可以使用列的初始值，因为优化知道索引的结构。创建一个保留键值索引使用如下所示的 REVERSE 关键字。

```
CREATE INDEX busy_table_ix ON busy_table (id) REVERSE;
```

警告 虽然索引列中字节的保留将索引条目分配到整个索引叶子块中，并避免了在实例之间最有可能的叶子块竞争，但这个值的分配不是完全随机的。如果每个实例每次缓存100个数，两个不同实例从它们的缓存值中检索出的可用值的最后一个字节数可能是相同的。例如，数4022的最后一位字节的内部表示是27，与数4122的最后一位字节完全相同。如果两个实例增加它们的值，分别使用4022与4122同时分配给表，如果它们使用一个保留键值索引的话，它们也需要同时更新相同的叶子块。

应该使用保留键值索引帮助降低索引重建的频率——它们设计的目的，并且不希望它们会避免并行服务器数据库中的竞争。可以不恰当地分配 PCM 锁，那样你将会对数据库的性能感到失望。

39.5.6 序列生成器与多实例

如果决定在并行服务器数据库中使用序列生成器，需要清楚一些限制。序列生成器可以用于加速处理的一个方法是通过它们将一些数缓存入一个实例的 SGA 区。这允许大多数需要序列数的事务从内存中找到它，而不必执行一次硬盘读。当下一个数生成并保存时，缓存也可以避免锁定资源的需要。

序列生成器的 ORDER 选项用于确保数按顺序被分配。这意味着将给第一个要求数的进程一个最小的没有使用的数。在并行服务器数据库中需要使用这个特性，但 Oracle 只能提供从硬盘上的数据字典分配每个数的能力，它不能使用缓存的数，因为实例没有能力在它们自己之间传送缓存的数。一个失效的实例不能将它的缓存的数传送给另一个实例。因此，如果同时使用 CACHE 与 ORDER 选项定义了一个序列生成器，当以并行方式启动一个实例时，Oracle 将忽略 CACHE 选项，这很可能降低序列生成器的效率。

39.6 并行服务器创建的特殊考虑

不论是否正在建立并行数据库用以支持超越故障或可伸缩性能力，需要注意与基本结构有关的不同的选项。当准备建立数据库时，应该早已决定需要多少个实例，以及如果需要，准备怎样分区数据。也需要考虑作为 Oracle 数据库设计一部分的数据库结构的其他选项，如

ARCHIVELOG或NOARCHIVELOG状态，以及需要什么样的字符集支持所有需要的字符集。当已决定了这些设计时，就可以使用 CREAT DATABASE命令建立数据库了。

在表39-8中，CREAT DATABASE命令的选项与在建立数据库时应该考虑的问题一同列出。

表39-8 CREATE DATABASE命令选项

选 项	描述与使用
CONTROLFILE REUSE	只在计划重写控制文件时需要。建议不要使用这个选项，因为如果使用了错误的init.ora参数文件启动实例，它会破坏一个好的控制文件
LOGFILE	在单实例数据库中，这个参数允许创建需要的多个重做日志文件组，每个含有要求的多重拷贝的数量。使用这个选项创建的组将属于第一个重做线程，并将自动作为公有线程启动，如果想要，可以在启动另一个不同的实例并关闭这个实例之后，禁止这个线程，并重新允许它作为私有线程启动
MAXLOGFILES	设置可以为数据库创建的重做日志组的最大数量。这个数应该是所有想要创建的实例的重做日志组的数量之和。也许想要将这个值设得足够大，以支持在附加的日志组以免你需要对一个或多个实例增加重做日志组及支持附加的可用的实例
MAXLOGMEMBERS	设置可以分配给任何一个重做日志组的多重重做日志文件的最大数量。通常，日志组成员的数量在所有的实例重做上是相同的，但如果不相同，应该将这个设为一个日志组需要的最大数
MAXLOGHISTORY	决定将存储在控制文件中的归档日志文件条目的数量。当达到这个数时，当新的条目增加时，老的项目被取代。这些条目含有与归档日志名与内容有关的信息，用于自动介质恢复选项以指明它需要使用哪个归档日志文件。如果想要使用自动的介质恢复，应该将这个参数值至少设为在一个单独的恢复循环中所实例将创建的归档日志文件的数量
MAXDATAFILES	设置为数据库创建的数据文件的最大数量。它是一个典型地比单实例数据库要高的数，由于避免实例之间块竞争的分区分表与索引的需要。与使用MAXLOGFILES一样，应该考虑将这个值设为足够大，以容纳将来分区需要的增长
MAXINSTANCES	指明可以同时存取数据库的实例的最大数量。如果为超越故障使用一个或多个实例，需要使用这个总数，即使在通常情况下不需要用户连接它们。这是另一个需要考虑到将来增长设置的参数
[NO]ARCHIVELOG	允许决定数据库在创建时是否处于 ARCHIVELOG模式。虽然建议对大多数产品数据库使用 ARCHIVELOG模式，通常不必要立即将它置为这种模式。缺省值是NOARCHIVELOG
EXCLUSIVE	这是一个选择性关键字，并将在 Oracle8中废除。它加强了第一个实例不能以共享或并行方式打开的事实。建议忽略它
CHARACTER SET	指明了数据库用于存储数据的字符集。这个参数没有特定于并行服务器的特殊的特性
DATAFILE	命名最初组成SYSTEM表空间的数据文件或数据文件集。这个参数没有用于并行服务器的特殊的特性。与此相似，对已命名数据文件的 AUTOEXTEND选项在单实例与多实例数据库中是完全相同的

需要建立一个SQL脚本文件以包含这条命令，及其他为并行服务器使用用户化数据库的命令。这些包括执行下述步骤的命令。

- 至少为每个实例增加一个回滚段。
- 为每个附加的实例增加一个重做线程。
- 启用重做线程。
- 运行catparr.sql脚本。

39.7 怎样监控与调整并行服务器

需要知道一些特定的动态性能表，以监控并行数据库的性能。在用于任何 Oracle数据库的标准的调整步骤之外，还需要使用这些调整工具与调整步骤。

对调整Oracle并行服务器非常有用的一些动态性能表使用称为 catparr.sql的脚本建立。作为SYS用户运行这个脚本，创建表并当增加新的想要追踪的段或区间时更新它们，在 Oracle动态增加想要监控用于数据库映射视图的区间位置之后，也需要运行这个脚本。

提示 应该养成在开始任何监控或调整工作之前，将运行这个脚本当作理所当然的事的习惯。这样，可以确保当检查需要的各种各样的动态性能表时，看到的是最当前的信息。

在Oracle 8中，下面几节中讨论的表，与大多数其他动态性能表一样，具有一个全局版本显示所有活动实例的各自的值。一个实例数列指明了数据正在报告的是哪个实例。为使用这些全局表，需要在表名前包括一个g。程序清单39-7显示了在全局视图V\$VERSION上进行的查询。

清单39-7 查询一个全局动态性能表

```
SQL> SELECT * FROM gv$version;
```

```
INSTANCE_ID BANNER
```

```
-----
1 CORE Version 4.0.2.0.0
1 NLSRTL Version 3.3.0.0.0
1 Oracle8 Server Release 8.0.3.0.0
1 PL/SQL Release 3.0.2.0.0
1 TNS for Solaris: Version 3.0.2.0..0
2 CORE Version 4.0.2.0.0
2 NLSRTL Version 3.3.0.0.0
2 Oracle8 Server Release 8.0.3.0.0
2 PL/SQL Release 3.0.2.0.0
2 TNS for Solaris: Version 3.0.2.0.0
```

39.7.1 V\$LOCK_ACTIVITY

需要监控的第一个表是V\$LOCK_ACTIVITY。这个表显示了在一个单独的实例中，自它启动以来所发生的锁变化的数量和类型。虽然真实的数量自身并没有任何意义，因为每个系统都将以不同的速度处理锁与相关的活动，但可以使用它们比较不同实例的性能，并比较一个特定实例不同时期的行为。在V\$LOCK_ACTIVITY表上的查询将产生与清单39-8相似的输出结果。

清单39-8 查询V\$LOCK_ACTIVITY

FROM	TO	ACTION	COUNTER
NULL	S	Lock buffers for read	3595
NULL	X	Lock buffers for write	8111
S	NULL	Make buffers CR (no write)	2763
S	X	Upgrade read lock to write	1046
X	NULL	Make buffers CR (write dirty buffers)	92
X	S	Downgrade write lock to read (write dirty buffers)	1220
X	SSX	Write transaction table/undo blocks	907
SSX	NULL	Transaction table/undo blocks (write dirty buffers)	0
SSX	S	Make transaction table/undo block available share	0
SSX	X	Rearm transaction table write mechanism	907

清单39-8显示了一个实例中发生的PCM锁变换的不同类型，同时还有变换发生的次数。FROM与TO列中的项目显示了锁的状态：

- NULL —— 不是当前分配给实例的。
- S —— 共享模式。
- X —— 排它模式。
- SSX —— 子共享排它模式。

共享锁用于查询表与索引块，当需要更改块竞争时使用排它锁。回滚段使用子共享排它锁，因为每个回滚段都由一个实例在启动时获得，只有这个实例可以向它写，因而不需要实例放弃写向它的回滚段块的独占能力。然而，其他的实例也许需要读回滚段信息，并当这些读在处理时，使用子共享排它锁在拥有的实例上覆盖回滚段块。应该看到从 X到SSX与从SSX到X的转变的两个计数含有相同的值，因为任何借给另一个实例用于读目的的回滚段都不得不返回它拥有的实例。可以忽略从 SSX到空与从SSX到S的转变的项目，因为它们来自于在系统启动时或表空间管理活动时所执行的操作。

如果能察觉到 PCM锁变换与inging正导致性能问题，或如果想要预先活动并定期监控PCM锁的状态，V\$LOCK_ACTIVITY表正是你应该开始的地方。你将总会看到一些块变换活动，因为一个实例在实例启动时拥有所有的 PCM锁在NULL状态，在实例可以做任何工作之前改变实例需要的锁的状态。应该将注意力放在锁返回为 NULL状态的锁转变的值，及回滚段锁活动的值上。转变为NULL状态只发生在不是你监控的实例需要使用这个锁时。如果这些转变的计数高并持续增长，应该猜想到 pining是导致任何数据库性能下降的一个原因。你将需要察看其他的并行服务器动态性能表以确定包括哪个块与段。

在从X到SSX与从SSX到X的成对的变化表明回滚段数据正被 ping。如果这些数值较高，应该考虑使用GC_ROLLBACK_LOCKS增长回滚段锁的数量。如果问题仍然存在，可能需要转变为精细锁。

39.7.2 V\$BH

V\$BSH表显示实例的数据库缓冲缓存中的当前内容。它提供了每个缓存中的文件与块的标识数，及该块上的PCM锁的状态。一个查询，连同部分列表一起展示，如清单 39-9所示。

清单39-9 查询V\$BSH表

```
SQL> SELECT file#, block#, class#, status, xnc FROM v$bh;
```

FILE#	BLOCK#	CLASS#	STATUS	XNC
8	438	1	XCUR	0
8	467	1	SCUR	0
8	468	1	SCUR	0
8	468	1	SCUR	0
9	192	1	CR	0
9	198	1	CR	2
9	201	1	XCUR	10

列的状态是如表39-9所示的七个值之一。通常在实例运行一段以后，只看到行具有 SCUR、SCUR或CR值。此时，所有的缓存至少已使用一次，任何介质或实例恢复应该已经完成。偶而会发生一个缓存正在使用从一个数据文件读出的数据装填。

表39-9 在V\$BH中显示的缓存状态

状 态	解 释
FREE	这个缓存当前没有使用
XCUR	这个缓存中的块由一个排它 PCM 锁覆盖
SCUR	这个缓存中的块由一个共享 PCM 锁覆盖
CR	这个缓存中的块由降级的 XCUR 或 SCUR 锁覆盖
READ	块正在从硬盘读入缓存
MREC	缓存中的块处于介质恢复状态
IREC	缓存中的块处于实例恢复状态

提示 如果发现缓存处于 MREC 或 IREC 状态，应该立即退出数据库。如果缓存处于任何一种恢复状态，将得不到数据库性能的任何有用的信息。而且会话几乎正在竞争资源，这些资源是由恢复进程以实时方式完成工作的最好利用的资源。

经常会发现在内存中有许多缓存处于 CR 状态，同时在 V\$LOCK_ACTIVITY 中有过多的 X 到 NULL 或 X 到 S 的转变。XNC 列显示了特定块上 X 到 NULL 的转变，该列上的非零值代表这个块当前正被 ping，可以从 FILE# 与 BLOCK# 列找出包含了哪些块。通过使用另一个实例的 V\$BH 表或 Oracle 8 的 GV\$BH 表，与另一个实例的缓存中的块相比较，可以找出哪些块正在哪些实例之间被 ping。有些情况下，可能在其他的实例中找不到相同的块，这表明发生的是假 ping，因为锁已被释放，但并不需要这些块本身。

为解决真正的或假的 pinging 的问题，知道包括哪个表甚至是这些表中的哪些值很有用处。可以使用 ROWID 伪列查询表，以查看哪些块被包括。清单 39-10 是 7.0 版数据库中这种查询的一个例子。使用这种方法找不到索引项，最多能找出索引的名字。

清单39-10 检查特定块的行内容

```
SQL> SELECT * FROM DEPT
2  WHERE SUBSTR(rowid,1,8) = '00000201'      -- the block id number
3  AND SUBSTR(rowid,15,4) = '0009';          -- the file id number
```

DEPTNO	NAME	LOCATION	MGR_ID
14	Human Resources	Fairview	198
22	International Sales	New York	1356
...			

为找出与 V\$BH 中指明的特定块相关的段，可以查询 DBA_EXTENTS 视图，但使用 V\$CACHE 与 V\$PING 视图会更简单。

39.7.3 V\$CACHE与V\$PING

V\$CACHE 与 V\$PING 这两个动态性能表与 V\$BH 视图基于相同的缓存信息，但它们包括三个附加的列。在清单 39-11 中，可以在 V\$CACHE 上所进行的查询输出中看到这些列。列指明了块的名字、类型及块在缓存中属于哪个段的段 ID 号。这些视图在创建之后就不会含有新增加的段的信息，除非运行 catparr.sql。如果 NAME、KIND 与 OWNER# 含有空值，再次运行脚本，使用新的信息填充这个视图。

V\$CACHE 与 V\$PING 的不同之处在于：前者为每个高速缓存中的内存含有一项，而

V\$PING中为有可能被ping的内存含有一行。这由在XNC列拥有一个非零值的缓存决定，指明它含有的块至少有一个X到NULL的转变。

清单39-11 查询V\$CACHE

```
SQL> SELECT name, kind, owner#, file#, block#, status, xnc FROM v$cache;
```

NAME	KIND	OWNER#	FILE#	BLOCK#	CLASS#	STATUS	XNC
EMP	TABLE	14	8	438	1	XCUR	0
EMP	TABLE	14	8	467	1	SCUR	0
EMP	TABLE	14	8	468	1	SCUR	0
EMP	TABLE	14	8	468	1	SCUR	0
DEPT	TABLE	14	9	192	1	CR	0
DEPT	TABLE	14	9	198	1	CR	2
DEPT	TABLE	14	9	201	1	XCUR	10
...							

也可以使用V\$FALSE_PING动态性能表以指明潜在的假ping。这个表含有与V\$PING相同的列，但只含有极有可能多次被假ping的块的行。然而，Oracle不保证在这个表中列出的行即将被假pinging，或这个表含有被假ping的每一个块。仍然应该检查其他的缓存以找出上述的假pinging的确切证据，可以使用讨论过的视图的全局版本轻易实现这个目标。

39.7.4 并行服务器的调整策略

多实例数据库并行实例组件的基本调整策略含有下面四个步骤：

- 1) 决定是否有过多的碰撞询问。
- 2) 指出什么正在被碰撞询问。
- 3) 解析这个碰撞询问是真还是假。
- 4) 解析这个碰撞询问问题。

通过使用前面所讨论过的并行服务器动态性能表，可以找出pinging的总数，正被ping的对象，是否一个或多个实例正在ping相同的块。有了这些信息应该怎么做呢？

对所有的调整活动，如果用户不需要进行数据库性能的调整，只需要进行调整。通过不时地监控V\$LOCK_ACTIVITY的值，可以发现任何与事务响应时间成反比例增长的值。如果发现这样的值，需要从V\$BH, V\$CACHE或V\$PING中查出对此负责的块。

警告 当响应时间或相关事务处于峰值时，使用动态性能表要小心。如果现在查看这些值，会发现它们比一小时前要大多得多，因为Oracle在实例的生命期中累积统计值。在任何数据库做有用工作的时候，它的一些性能统计会增长以反映这个变化。对多个实例需要使用一个与单实例数据库相同的监控机制，这包括固定周期地收集统计数据，以便监控特定时期的变化率而不是依赖于不变的值。

提示 如果使用的是Oracle 8，使用全局动态性能表收集与实例有关的统计会很容易。如果从Oracle 7数据库升级为Oracle 8，需要将用于查询V\$_表的任何脚本替换为查询GV\$_表。

如果在动态性能表中发现真正的pinging，有一些解决选项。首先，如果只有两个实例，可以考虑将它们的工作负载与用户合并为一个实例。如果由于资源的限制或由于包括了不止

一个的实例，不可能进行这种合并，需要考虑使用前面讨论过的分区选项。目的是将多数对一个给定块的存取限制为只有一个实例存取。这需要仔细检查 ping 块的行以确定分区选项。

如果发现正在发生一个假的 ping，最简单的解决策略是对涉及的数据文件增加更多的哈希锁，或将哈希锁转换为精细锁。前一种将涉及创建过多的锁，以至于当前内存无法处理，在这种情况下，需要使用精细锁。如果这些附加的精细锁与现存的哈希锁合在一起仍然组成了过多的锁，以至于系统无法处理，需要试试将一些由哈希锁覆盖的数据文件释放为精细锁。

如果使用这些方法仍不能解决 ping 问题，这可能意味着数据库设计不合理，需要返回来重新进行规划，并使用前面讨论过的技术找出一个更好的设计方案。Oracle 7.3 以前的版本没有提供精细锁与相关益处，会使一些数据库不适宜使用并行服务器选项。现在，有了精细锁，使用 TP 监控器与 Oracle 8 附加的功能，如全局视图与反向键值索引，应该有能力为超越故障或可升级目的（如果不是两者兼备）建立或监控一个多实例数据库。

39.8 附加的提示与注意事项

除非正使用一种非常罕见的操作系统，诸如 nCUBE 与 DEC 的 VAXCluster 的派生系统，必须将 Oracle 并行服务器数据库放置在裸盘上。在 Oracle 8i 版本中，控制文件不再是固定长度的，其中存储了归档重做日志项的状态，用于 Oracle 的恢复管理器使用，即使不使用恢复管理器工具用于备份及恢复。可以通过设置初始化参数 CONTROL_FILE_RECORD_KEEP_TIME，将其设为一个较小的数，减少存储的项的数量。

也可以通过自己的操作，改变控制文件的大小。例如，可以设置实例的参数文件包括一个 DB_FILES 的值，它大于在 CREATE DATABASE 命令中设置的 MAXDATAFILES 的值。如果为数据库增加了比 MAXDATAFILES 要求的数量更多的文件，控制文件将增长以容纳它们。应该确保将控制文件放在足够大的裸盘上，因而可以含有一个比 CREATE DATABASE 命令产生的更大的控制文件。如果硬盘管理系统允许裸盘分区的内容比分区容量大，这更为重要。

虽然第一阶段的高速缓存融合减少了查询中使用的块在实例之间传递的开销，但并不能总结出可以避免按实例对数据进行分区的进程，在将块发送给请求的实例时，仍然会有消耗。集成的分布锁管理仍然必须在实例之间传递锁请求，簇交互必须在实例之间传递块映象，而不只是短信息。如果设计需要通过簇交互传送过多的块，会超过簇的带宽，因而降低系统的效率。

也有一些厂商不提供 Oracle 用于传送块映象的交互。在使用高速缓存融合之前，应该检查安装与配置手册。

如果具有一个不需要用户注册到一个指定实例的数据库设计，可以使用新的在 Net 8 文件中定义服务的选项。服务由两个或多个实例组成，在 Net 8 连接串中使用服务的名字而不是一个单独的实例的名字。Net 8 将使用由监听器进程维护的与服务相关的每个实例的统计，指出最不繁忙的实例，用户的注册将路由到这个实例。这允许在实例之间维持动态工作负载平衡。如果运行了几个实例，当然可以创建多个服务，每个服务覆盖实例的一个子集。这允许提供不同用户与实例集的负载平衡。

第40章 分布式数据库管理

本章要点：

- 理解分布式数据库
- 使用一个分布数据库
- 使用分布式事务
- 理解只读快照
- 在列级创建一个只读快照
- 快速快照刷新

40.1 理解分布式数据库

分布式数据库是作为一个单独的数据库但位于不同场所的系统。这些场所可以在任何地方，从紧邻的办公室到世界的另一端。在一个具有不同结点连接起来的网络环境中，分布式数据库担当一个单独的系统。为了完全理解一个分布式系统是如何工作的，数据库管理员必须具备一定的知识，理解多硬件系统、网络、客户 / 服务器技术与数据库管理。

涉及到分布式数据库系统管理的信息很少。随着 Oracle 8i 版本与未来开发的数据仓库的出现，分布式数据库的使用有了极大的改变。许多 DBA 可能没有机会管理这种分布式系统，所以并不熟悉这种系统是什么。

本章描述各种类型的分布式系统，它们是怎样使用的，怎样管理它们。Oracle 8i 提供了一种创建多个分布式系统的简单的、实际的方法。然而，不是所有的公司都有能力支持一个 GUI 环境，或如果进行远程支持，只能使用服务器端的行命令模式。由于这个原因，本章描述的处理不提及分布式选项的 GUI 部分，只提及当鼠标点下之后发生什么，以及在没有 GUI 环境的情况下如何管理一个系统。Oracle 8 与 Oracle 8i 中出现一个名为 Oracle 企业管理器的 GUI 工具，用于管理分布式数据库。在 Oracle 8 中，企业管理器只在 Windows NT 上可用。在 Oracle 8i 中，企业管理器是一个基于 Java 的工具，可以从任何主要的平台上运行。

40.1.1 描述每种类型的数据库

分布式数据库实际有两种类型：

- 具有远程查询、数据维护与两阶段提交的分布式数据库。
- 通过数据管理方法，诸如快照与触发器或其他非数据库管理方法（如 SQL*PLUS 中的 COPY）的复制数据库。

最简单的分布式数据库是一系列独立的数据库在逻辑上通过网络链接到一起以形成一个单独的视图。复制数据库含有通过网络连接从其他远程数据库拷贝的信息。

复制数据库可简单按在它们之间传递信息所使用的方法进行分类。下面是这种拷贝处理的两种主要的方法（通常称为传播）：

- 分布式事务。

■ 快照刷新。

分布式事务是这样一种处理：在其中用户对一个结点进行更新，通过触发器或过程将这个改变传送给另一个结点。快照是表的副本（或子集），可以从主结点传播到每个远程结点。

在分布式数据库系统中，有必要将数据跨系统进行连接，糟糕的数据存取技巧会导致在数据库服务器之间的用户数据流成为全部性能的主要因素。为了给一个最优化的执行计划决定最好的存取技巧，优化器首先必须能够决定足够数量的替代路径。在 Oracle 8i 中加强的远程连接表现出更多的选项，允许产生一个伴有相应性能增长的更好的执行计划。

40.1.2 数据库命名规定

数据库命名、存取与管理的方式开始对存取因特网与基于因特网的数据库有较大的影响。Oracle 建议数据库与它们的链的命名规定遵循标准因特网域名命名规定，这个规定可以有許多部分，每一部分如 IP 地址一样，由点进行分隔。

数据库的名字从右向左读，这个命名规定可以有几个部分，第一部分（最右边的部分）是基本的或根域。缺省情况下，域名为 world。这在 Net 8 中并不要求，但为了一致性原因，最好还是包括它以支持 SQL*Net 的老版本。

域名可以基于公司或所处位置的结构。例如，在德国的数据库的命名规定可以有几种方式，最简单的形式是，名字可以是 GERMANY.WORLD。如果数据库名字基于位置，名字将是 GERMANY.BWC，其中 BWC 是域。另一种方法将是使用大陆扩展位置，名字将是 GERMANY.EUROPE.BWC。不论是哪种命名规定，在将来的维护中，名字对程序员与用户必须是透明的而且易于理解。

警告 在命名一个域与数据库时还有一些限制。在常用数据库命名规定之外（没有空间、没有特殊的字符等等），操作系统或网络会限制名字的长度。参考 Oracle 指定的文档以查找名字的长度限制。

40.1.3 归档透明性

这点很重要：当为远程系统上的表或对象提供名字时，命名规定允许系统的程序员与用户使用名字存取远程表，就像它们在本地图一样。透明性的概念是可以获取所有对象，这些对象对数据库管理员与用户是相同的。

Oracle 通过使用 SQL*Net 与透明网关，使系统的开发看起来是相同的，而不管数据库的厂商、类型与位置。为获取更多的与数据透明性有关的信息，参考第 34 章。

透明性的目的是提供对所有数据库的无缝访问。例如，用户可以以相同的方式访问任何一个表（前提是它们具有安全性），可以使用与访问本地表相同的语法访问在 SYBASE 数据库上的表。可以使用与在底特律的本地表相同的语法访问在德国一个 Oracle 数据库中的表。要获取更多的透明网关信息，阅读 Oracle 8i 在相应的网关上的文档。例如，通过使用一个透明网关，SYBASE 系统现在看起来就像一个 Oracle 数据库，可以使用数据库链进行指引。

异构服务代理在 Oracle 8i 中处理为多线程。当系统中有大量用户会话并行访问相同的非 Oracle 系统时，这种新的功能将减少系统消耗的资源总数。这种更有效地使用系统资源的方式允许大量的并发用户会话。

下面是一个异构型环境的示例图：

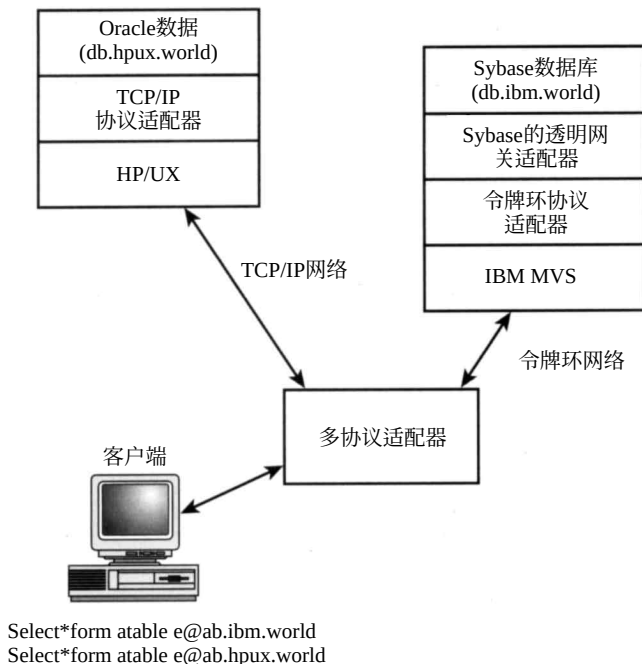


图40-1 分布式异构环境

必须有一个方法论以将任何数据库地址的改变传播到其他的分布式结点。这最终是数据库管理员的职责。许多结点将 TNSNAMES.ORA与SQLNET.ORA放在一个集中的应用服务器上，这会以一种较方便的方法管理 TNSNAMES。然而，这并不总是最优的解决策略。在一个分布式环境中，每个结点上可能具有其他的独立的数据库（如测试数据库、该结点上特定的其他数据库等等），并且无法拥有一个本地化的 TNSNAMES。如果情况是这样，公司应该决定使用 Oracle NAMES而不是Net8作为管理连接的方法。在创建一个分布式环境时，模式名、用户ID与口令也必须是可被管理的。例如，如果在创建数据库链时含有口令，口令就不能改变，如果口令改变了，必须告诉对受该模式影响的数据库链进行维护的数据库管理员。

注意 如果使用Net 8，结点之间的通信很重要。必须有一个形式化的方法用于移动分布式数据库的IP地址或重命名数据库。

在Oracle 8.0中引入了代理自注册，它减少或消除了配置异构型服务时需要数据库管理员的介入。在Oracle 8i中，重写了代码使自注册处理更加有效。

Oracle 8i中的新特性是代理指定，HS对象文件的共享库而不是共享磁盘，它是在连接代理执行文件时的替代品。这个改变的益处是与平台有关的，可以通过为所有类型的代理使用一个单独的代理库（extproc、hsalloci、hssqlpss、hsdepaxa及hsots）增强它们的可伸缩性。而且对内存的需要将减少，因为代理执行程序会变得非常小。

40.1.4 使用Oracle安全服务器与全局用户

Oracle安全服务器是使数据库管理员或安全管理员管理全局环境中的安全的工具。安全服

务器使系统管理员可以创建一个用户，利用个人证明而不是口令进行验证。这种证明类似于请求访问支持电子商务的结点，这种证明代理之一是 VeriSign。

当注册到某一结点时，产生一个错误，警告用户在进入结点之前必须具备个人证明，他们的证明通过名字、地址与其他唯一鉴别所需要的信息鉴别他们。Oracle安全服务器第1版只对Web服务器执行证明。在第2版中，改进了这个能力，包括对Net 8客户与服务器的证明。依照Oracle文档，这种由Oracle安全服务器执行的证明遵守电子商务的国际标准（X.509）。

为确保只有具有相应特权的那些人才能访问数据库，Oracle创建了一个两层的安全级别。数据库管理员必须使用Oracle安全服务器创建用户，并且必须作为全局用户存在于数据库中。如果没有用户ID与证明，访问被拒绝。在创建全局用户时必须小心，用户ID不能既是全局的又是本地的。

40.1.5 SQL*Net

数据库管理员必须对怎样有效地设置与使用Net8管理分布式系统有深刻的理解。Net8及它的管理在第33章中讨论。

通过系统管理员和网络管理员协力，数据库管理员将必须建立有能力支持分布式系统的服务器。控制邮件或企业内部网系统的服务器不适于用作分布式数据库。网络负载将对任何数据库的性能有直接的影响，特别是分布式系统。

40.2 使用一个分布数据库

分布式数据库由通过数据库链与网络连接链接起来的多个数据库组成。对这些系统的管理可以大不相同。在大型、高度集中的企业中，每个位置必须有它自己的数据库管理员；在一个高度集中的企业中，一个数据库管理员将负责位于不同州，有些情况下，不同国家中的数据库。对这些数据库的管理与调整与自治的数据库系统相同，只是指向它们的链接与视图有所不同。

有许多原因需要使用一个分布式数据库系统，下面就是一些：

- 应用设计 —— 一定的软件设计深化为一个分布式系统。具有多个位置，每个位置上有分离的IT/IS部门或数据库的公司将使用相同的或相似的软件。每个地点将维护它自己环境中唯一的数据。当公司发展时，每个地点将会要求访问其他结点位置上的数据，这最终会发展成为一个分布式系统。
- 改进的性能 —— 本地数据库将比一个大型集中的数据库小。结果是，该数据库上的查询与其他的事务将会运行得更快。网络活动将显著降低，改进整个系统的性能。
- 较少的、不太昂贵的硬件需求 —— 通过减少每个系统上用户的总数，支持这种系统的硬件可以相当地小（依赖于每个结点上的实际用户数）。例如，每个本地数据库可能会运行在一个单CPU NT服务器上，而不是运行在一个大型主机上。
- 改进的可靠性与可用性 —— 通过将数据分布到多个结点，系统更有可能处于运行状态。当一个结点停机时，系统的其他结点仍处于运行状态并且是可访问的。这种软件的分布增加了另一维可靠性。通过具有拥有软件副本的结点，每个结点可以是一个镜像结点以支持一个灾难恢复计划。
- 改进的大量开发支持与前台应用。

在8.1 (8i) 版本中, Oracle给它的后台应用和前台应用提供了高级复制。后台应用要求实时复制数据, 前台应用区域是一个正在增长的市场, 特别对大量开发而言, 此时高级复制功能具有许多优势。

这些功能对企业是有用的。例如, 假设大型装饰品公司 (Big Widget Company, BWC) 是一个大的国际化的装饰品制造企业, 在密执安州的底特律、意大利的米兰、德国的法兰克福都有办事处。BWC开始一个进取性的研究与发展计划, 开发一种新产品用于国际使用。因为每个国家的标准不同, 每个国家的装饰品定义略有差异。这些区别保存在一个名为 SITE_SPECIFICATION的分离的表中。因为这是一个研究与发展项目, 定义的改变是高度动态的, 每个国家需要临时访问其他国家的定义, 所以需要使用分布式数据库。

40.2.1 设置一个分布式系统

分布式系统的设置必须独立于应用。如果软件使用分布式数据库, 许多由数据库管理员执行的工作, 诸如创建数据库链, 将在软件安装的过程中自动实现。然而, 在 BWC案例中, 由数据库管理员负责设置分布式数据库。因为这是一个研究与发展计划, BWC已决定在项目的开发中进行投资。结果是, 维护数据库的责任依赖于一个数据库管理员。

每个数据库将作为一个自治的数据库被安装。安装将基于每个国家的数据库上的并发用户的数量。其他在安装数据库时要考虑的因素如下所示:

- 数据库的所有使用。
- 使用数据库的其他应用。
- 标准DBA调整技术。
- 用于连接的网络协议的类型。
- 数据是如何动态变化的, 它将如何影响网络传输量。
- 当前网络传输量的级别以及新的数据库将对它产生什么影响。

注意 如果系统被大量地访问, 在决定并发用户时应该包括远程用户。

1. 使用数据库链

数据库链组成了Oracle用于访问远程数据库对象的方法, 有三种类型的数据库链:

- 公有。
- 私有。
- 全局。

公有数据库链与公共同义词相似, 所有的用户都可以引用它。私有数据库链由链的拥有者(模式)访问, 公有数据库链在使用Oracle NAMES时自动创建。第33章详细解释Oracle Names。

在Oracle 8数据库链的语法中有一些显著的增加。创建一个公有数据库链或私有数据库链的主要部分是相同的。

```
CREATE
[SHARED]
[PUBLIC]
DATABASE LINK dblink
[authenticated clause]||[CONNECT TO [CURRENT_USER|user
IDENTIFIED BY password] [authenticated clause] USING '{connect string}';
```

这个语法在Oracle 8与Oracle 7中有一些不同, 在Oracle 8中增加了一个新的进程, 另一个

进程也被改变。

SHARED是一个全新的进程。为了使用这个进程，数据库必须使用一个多线程服务器。这使创建数据库链时可以使用现存的可用的连接。除非数据库管理员已完全理解不能共享数据库链与多线程服务器的隐含概念，否则不能这样做。如果设置不正确，将严重影响系统的性能。为获得有关共享数据库链及怎样有效地使用它们的详细信息，阅读《Oracle8 Server Distributed Database Systems》中的第2章。

验证子句与数据库链的 SHARED部分有关。当使用 SHARED时，必须使用它。下面是验证子句的语法：

```
AUTHENTICATED BY username IDENTIFIED BY PASSWORD
```

在验证子句中用户并不执行任何操作，只要求用户名是远程服务器上的一个合法用户与口令。数据库管理员可以专为这个目的创建一个哑帐号。

CURRENT_USER使用新的Oracle8的全局用户类型，这个强有力的新功能能够创建一个用户，该用户可以使用一个单独的注册访问结点上的任何数据库。这并不会与本地用户产生混淆。这个用户必须使用Oracle8的安全服务器创建。

使用BWC示例，下面的代码将从底特律创建一个到德国的简单的公有数据库链。

```
CREATE PUBLIC DATABASE LINK germany.bwc.com  
USING 'GERMANY';
```

注意数据库名字对大小写是敏感的。如果不是由 Oracle Names进行的定义，必须以别名的形式或者以 TNSNAMES.ORA中的真实名字的形式显示 GERMANY名字。TNSNAMES.ORA必须在客户和服务上。使用本语法，用户当前必须注册到在德国和底特律的数据库中。假设模式名在德国与底特律是相同的，访问 SITE_SPECIFICATION的语法如下：

```
SELECT * FROM SITE_SPECIFICATION@GERMANY.BWC.COM;
```

注意 SQL语句内的链接不是大小写敏感的。

提示 为确保透明性，创建一个同义词或视图，对用户隐藏数据库链。

```
CREATE SYNONYM germany_specification  
FOR site_specification@germany.bwc.com;
```

可以创建简单的数据库链，也可以创建复杂的数据库链，这由数据库管理员决定。例如，可以创建一个连接到欧洲的链，如果 BWC扩展到欧洲的其他国家，这将非常有利。米兰和法兰克福的数据库链为：

```
CREATE DATABASE LINK europe using 'EUROPE.BWC';
```

然而，在创建一个同义词时，这个初始化连接将不起作用，数据库管理员通过增加国家扩展这个名字。

```
CREATE SYNONYM germany_specification FOR site_specification@europe@germany;
```

注意 与Oracle以前的版本一样，Oracle8不支持从远程数据库选择LONG数据类型。

2. 延迟事务

延迟事务是稍后执行的事务。例如，在主结点上进行的改变存储在一个队列中，并且被延迟直到复制机制取出改变并将它们应用到远程结点（参见表 40-1）。

3. 对分布式系统使用初始化参数

除在分布式数据库一节指明的参数以外，更多的参数如表 40-1所示。

表40-1 被延迟事务数据库影响的参数

参 数	描 述
COMMIT_POINT_STRENGTH(0-255)	这个参数用于设置在两阶段提交中的提交点场地。具有最高提交点强度的结点是提交点场地。每个使用这个结点作为提交点场地的结点必须处于相同的结点上。决定哪个数据库将成为提交点场地的因素是数据的拥有者（驱动器）、数据的关键程度与系统的可用性。有关提交的信息存在提交点的系统上。建议所有的结点还要具有相同的值或只有一个较大的值，这确保了万一失效，其他的结点可以记录这个数据。这个参数的缺省值与操作系统有关，参考 Oracle8特定操作系统文档以找出这些值
DISTRIBUTED_TRANSACTIONS (0-TRANSACTIONS)	并发分布式事务的数量的限制。如果设为 0，不激活 RECO进程（Oracle的恢复进程），禁用数据库的分布式能力
GLOBAL_NAMES	如果设为 TRUE，链中所引用的名字必须与数据库名匹配，而不是与别名匹配。为使用 Oracle 8的高级复制功能，必须使用这个值
DML_LOCKS(20-无限)	在一个单独的事务中限制 DML锁数量
ENQUEUE_RESOURCES(10-65535)	允许几个并发进程共享资源。为确定这个值是否设置得当，查看 v\$sysstat中的 enqueue_waits值。如果它是非零值，增大 enqueue资源。
MAX_TRANSACTION_BRANCHES(1-32)	这个参数的最大值可以从 8~32。分支是在一个单独的分布式事务中可以访问的不同服务器（或服务器组）的数量。减小这个数可能会降低共享值使用的总数
OPEN_LINKS(0-255)	一个会话中打开的到其他数据库的并发连接的数量。在分布式环境中，必须小心确保这个值不小于在一个单独的 SQL语句中可以引用的远程表的数量。如果将它设为 0，将不再有分布式事务
OPEN_LINKS_PER_INSTANCE (0-UB4MAXVAL)	这是Oracle 8的新参数，它限制了由外部事务管理器创建的打开链接的数量，要获得详细信息，参见 Oracle 8i 文档

40.2.2 识别分布式系统中的潜在问题

在管理一个简单的分布式系统中，系统改变是最大的问题，这与管理一个单独的、自治的数据库相似，但区别在于每个数据库可以看到其他数据库中的对象，这样就创建了一个依赖于连续性的系统。下面是必须在分布式系统内部进行调整的变化。

- 结构的改变 —— 当远程表作了结构上的修改或被删除后，使用过程与触发器更新或复制特定的远程表将会失败。指向一个被删除的表的数据库链将会失败。

注意 如果用户删除了另一个系统链接引用的表，将不会有引用完整性警告，只有当表中含有被引用的元素时才发生这些警告。

- 模式的改变 —— 除表被移走之外，数据库链中的引用用户必须具有正确的权限。对资源环境文件与角色的改变必须进行调整以确保它们不会影响分布式系统的其余部分。这包括口令！
- 对SQL*Net对象的改变 —— 文件诸如 TNSNAMES、PROTOCOL与 TNSNAV将影响分布式数据库系统。
- 系统停机 —— 在停机之间进行调整至关重要，特别是对依赖于分布式更新、快照或复制的系统。有些情况下，如果这个处理被打破，必须进行手工干涉以修复这个问题。

除非数据库管理员被告知数据库或网络停机，否则分布式系统的完整性将受到危害。

组织之间的交流至关重要，特别是在开发者与数据库管理员之间的交流。一个数据库或 IP 地址结构的改变会影响整个企业及分布式系统的完整性。这种调整通常由 DBA 管理。

如果在数据库之间的连接丢失了，可能发生另一个问题。首先 DBA 负有决定如果连接丢失会发生什么的职责。DBA 必须能够容易地识别错误信息，特别是与 SQL*net 或 Net8 相关的错误信息。更多的有关连接问题解决策略的信息，参见本书第 33 章。

40.2.3 调整分布式系统

分布式数据库的调整方法与自治数据库相同，请参阅第四部分并记住以下规则：

- 如果分布执行，书写拙劣的 SQL 仍会执行较差。
- 本地基于成本的优化不会将分区的远程表看作是分区的。
- 避免在一个单独的 SQL 语句中使用多个数据库链。

40.3 使用分布式事务

当提及分布式事务与两阶段提交时，复杂性似乎是压倒一切的。记住一个主要的思想：Oracle 7 与 Oracle 8i 自动管理分布式事务。大多数 DBA 没有机会真正看到一个两阶段提交，因为它们是基于触发器的进程。下面的例子用图解说明了一个两阶段提交。

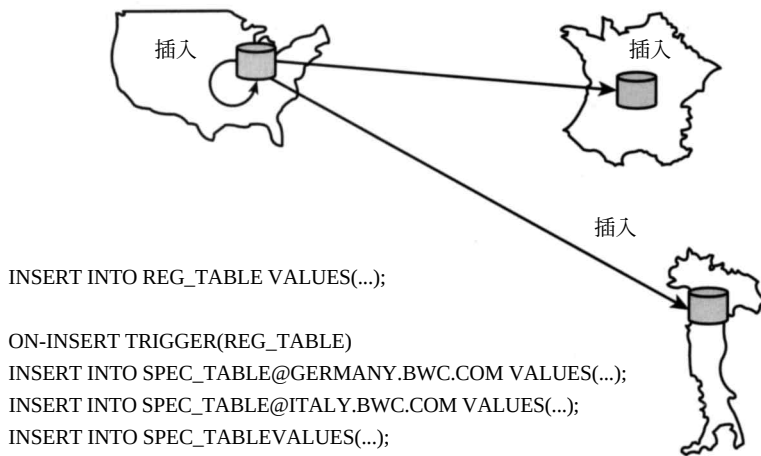


图40-2 两阶段提交

BWC 有一个规范表 (SPEC_TABLE)，依赖于美国 (US_REG) 中的规则表。当 US_REG 表中插入新的数据或进行更新时，应用更新并向在德国、意大利与底特律的 SPEC_TABLE 表中插入数据。

40.3.1 理解两阶段提交

所有的数据操作语言 (DML) 伴随发生两阶段提交，并且如名字所意味的，它是由两个截然不同的阶段组成的。为了确保解释的一致性，本段使用的 DML 是更新。

两阶段提交的第一阶段称为准备阶段。起始的数据库称为全局协调器，它向所有其他的数据库发送信息，通知它们将要发生一个更新。这个信息并不具有真正的更新，只是一个很

小的总量以使其他数据库知道将更新什么。如果更新发生，接收的数据库将响应全局协调器更新。接收的数据库可以三种方式进行响应：

- 准备好 —— 已准备好进行更新。
- 只读 —— 没有准备的必要。
- 退出 —— 子处理不能执行更新。

注意 全局协调器将决定数据库被更新的顺序。

接收的数据库也必须进行一些系统检查，包括发送一个准备语句给它自己的所有协同的数据库。这包括在将被更新的行放置锁（如果不是只读的）。在所有的接收数据库进行响应之后，全局协调器将事务置于悬而未决状态。如果所有的子响应都是准备好或只读的，全局协调器决定提交点。

DBA根据数据库的稳定性决定哪个数据库将作为提交点。例如，在 UNIX服务器上的数据库将比在PC上的数据库更加稳定。主要的目标是选出一个不会由于悬而未决事务而创建一个锁的数据库。被提交或回滚的事务称为悬而未决事务。

DBA由参数文件中的COMMIT_POINT_STRENGTH的值分配提交点。分布树中的较高的值决定了提交点强度，如果系统是高度分布的，可以有一个以上的数据库被分配作为提交点。如果事务影响不同的数据库，这也适用。提交点是第一个将被提交的数据库。它保持事务的状态直到所有的事务完成。

第二个阶段，或提交阶段，是提交数据的过程。在所有接收数据库被成功地提交并将它们成功传给全局协调器后，全局协调器向提交点发出一个信息，提交点移走悬而未决状态。每个子处理将提交或释放记录上的行级锁。

当子响应发出一个退出信息时，整个事务被回滚。事务是一个要么全有要么全无的进程。为确保数据的一致性，所有的子响应必须提交或回滚主事务。DBA必须可以指出并改正一个不成功的提交以有效地管理这种类型的分布式系统。

注意 当提交不成功时，必须通报执行这个提交的程序或人发生一个错误，应该通过设计产生这个通报。

注意 如果事务在分布式系统的一个结点提交失败，事务在所有的结点回滚。当创建回滚段时，分布式系统中的所有的主机（结点）必须有能力供给分布式系统上预期的最大的事务。

40.3.2 悬而未决事务的处理

网络问题是最有可能导致悬而未决事务的问题。与管理两阶段提交相同，Oracle 7与Oracle8i自动管理在网络故障中检测出的问题。与允许进行灾难恢复处理的自动恢复一样简单，Oracle建议DBA允许悬而未决事务的自动恢复。这个建议有一些例外。由于在网络故障过程中放置在悬而未决事务上的行级锁有可能产生问题，在故障过程中，其他用户试图访问这些行就成为问题，这些数据成为其他用户不可访问的数据。DBA必须要手工管理这个事务以清除这些锁。管理事务的过程称为强制一个事务执行。

如果网络丢失，它不只会显现给DBA，还会显现给所有受那个网络影响的用户集团。其他没有受到网络停机影响的数据库可以继续工作。当用户试图在一个不可访问的数据库上执

行一个DDL时，他们将会发现一个问题，这个DBA也注意到的问题就是锁。有早期SQL*Net版本经验的DBA熟悉当删除一个连接时，会发生一个端口监督程序锁。这与在分布式事务的过程中网络掉了的情况下发生的锁的类型相同。主要的区别在于如果不管它，分布式事务将在网络恢复时修正它自己。如果网络要关闭很长一段时间，DBA应该准备强制任何产生锁的显著的事务执行。

强制事务执行为回滚或提交的过程很简单。问题是DBA必须在本地强迫事务执行。这意味着如果一个DBA负责管理多个数据库，每个事务将不得不进行手工调查。例如，在底特律的一个用户更新了表中的一行，这需要同时更新米兰与法兰克福中相同的表。如果事务在写任何系统之前失败，DBA应该登录到每个计算机并在每个场所强迫事务执行。这个强制执行可以是提交或回滚的形式。虽然这很麻烦，但幸好这不是经常要做的工作。

当手工强制事务执行时要非常小心。它应该是要么全部要么没有的解决策略。当执行时，DBA应该与其他的DBA协同以确保这个修改与其他的环境匹配或手工验证所有数据库上的一致性。这意味着查询每个受影响的数据库，并确保数据在每个结点是完全相同的。如果没有这样做，会发生数据的不一致性。手工强制事务执行的步骤如下：

- 1) 跟随事务到它失败或开始失败的地方。
- 2) 如果可能，找出一个已完成的事务，或者回滚这个事务，或者提交这个事务。
- 3) 使用这个事务作为强制其他事务执行的模板，开始执行的逻辑位置是提交点。
- 4) 查询DBA_2PC_PENDING表，在TRAN_COMMENT列将有事务从何处开始与做了什么的信息，例如：

```
COMMIT COMMENT 'transaction name';
```

注意 需要使用单引号，事务名是大小写敏感的。

在ADVICE列也有信息，它指示DBA事务在哪里，例如：

```
ALTER SESSION ADVISE COMMIT;
```

DBA_2PC_PENDING表的GLOBAL_TRAN_ID列提供给DBA事务的ID，这个ID在分布式系统包括的所有数据库中是相同的。如果这个数据库是全局协调器，GLOBAL_TRAN_ID的最后一部分将与DBA_2PC_PENDING表的LOCAL_ID相匹配。

- 5) 查看DBA_2PC_NEIGHBORS，这个表看起来像这样：

LOCAL_TRAN_ID —— 这个分布式事务的本地事务ID。

IN_OUT —— 指出事务是IN还是OUT。

DATABASE —— 如果事务是IN，这是传送事务的数据库的名字。如果事务是OUT，这是事务正发送到的数据库的数据库链的名字。

DBUSER_OWNER —— 如果事务是IN，这是本地用户的名字。如果事务是OUT，这是数据库链拥有者的名字。

INTERFACE —— C代表提交请求，N代表准备好或只读。

DBID —— Oracle对连接的数据库的唯一的标识符。

SESS# —— 连接的会话ID。

BRANCH —— 这个分支的事务ID。

- 6) 使用这个表（数据库与LOCAL_TRAN_ID）中的信息，遵循这个线程到另一个数据库。

7) 查询DBA_2PC_PENDING表, 如果LOCAL_TRAN_ID与GLOBAL_TRAN_ID相匹配, 这是一个全局协调器。是 DBA应该开始事务线程的地方。如果在这个追踪过程中, DBA遇到了不能解决的问题, 使用它作为显著事务的模板。

8) 要强制执行一个事务, 使用 DBA_2PC_PENDING表中的TRANSACTION_ID, 如下所示:

```
Commit force 'transaction id';  
Rollback force 'transaction_id';
```

注意 需要使用单引号。

9) 如果有问题, 将事务回滚, 然后将这个回滚通知用户集团。作为预防, 检查受影响的表的状态以确保数据的一致性。虽然这很麻烦, 但它不是经常发生的, 并且总会保证数据库的完整性。

40.4 理解只读快照

快照是主表的一个副本(或子集), 以特定的间隔复制到其他子结点。只读快照是比分布式查询(一个指向远程表的SELECT语句)快的解决方案。除不占网络开销之外, 在远程表上的命中明显降低。如果信息不需要是实时的, 那么快照可以是一个为其他远程结点提供信息的最好的方法。快照通常在一个预定义的时间表内进行刷新。每个快照或快照集可以有它自己的更新计划表。快照可以按照每个结点上的需求复制到不同的位置, 除预定的快照刷新之外, DBA还可以手工刷新一个快照, 使用 DBMS_REFRESH过程完成, 这种刷新的语法为:

```
EXECUTE DBMS_REFRESH. REFRESH ('Snapshotname');
```

注意 使用这种形式, 快照将使用缺省的快速刷新。如果需要完全刷新, 必须将它加到过程调用中。

提供快照的目的与简单的分布式系统的目的相同, 下面是使用快照而不是动态数据库链的一些特殊原因:

- 改进的性能 —— 快照实际上是一个本地对象。用户可以查询快照而没有与网络相关的开销。
- 更高的可靠性 —— 如果主结点出了故障, 并不会限制对快照对象的访问。

例如, 由于对美国装饰品定义的依赖, BWC决定分布式数据库不再支持这个业务。装饰器规定的快照被建立。出于商务原因, BWC决定意大利与德国不需要知道其他国家的装饰品定义是什么, 但美国需要知道这两个国家的装饰品定义。所有的装饰品设计被不同的表持有, 所以不需要连接这些表。为减少混淆, 表的拥有者在全世界范围是相同的。

40.5 在列级创建一个只读快照

可更新快照现在可以是水平的(选中的行)子集或垂直的(选中的列)的子集。以前的版本只允许垂直子集。

垂直分区允许远程结点需要最小数量的数据, 因而降低连接时间。它还保护快照结点不修改与它们相关的主结点。可以在不影响快照结点的情况下向主结点增加列, 也可以在不影响快照结点的情况下在主结点删除快照没有引用的列。

40.5.1 设置一个快照

创建快照与创建表一样简单。事实上，这正是 DBA所做的——创建一个表的副本，不同之处在于副本成为一个自动的过程而不是人工的过程。在表被拷贝后，快照按规定的时间表进行更新以反映表的改变。快照的规划将基于与创建表所使用的一样的规则。

创建一个快照的语法如下：

```
CREATE SNAPSHOT [schema.]SNAPSHOT [Physical attributes *]  
[TABLESPACE tablespace]  
[LOB_storage_clause]  
[CACHE|NOCACHE]  
[CLUSTER (cluster column{,})]  
[table partition clause]  
[parallel clause]  
[USING INDEX physical attributes clause [LOGGING|NOLOGGING]TABLESPACE tablespace]  
[FAST|COMPLETE|FORCE]  
[START WITH date]  
[NEXT date]  
[WITH PRIMARY KEY|ROWID]  
[USING DEFAULT {MASTER|LOCAL}] ROLLBACK SEGMENT]  
[LOCAL|MASTER ROLLBACK SEGMENT rollback_segment]  
FOR UPDATE AS select_command;
```

例如，如下语句创建一个简单快照：

```
CREATE SNAPSHOT milan_specification  
REFRESH FORCE START WITH SYSDATE next SYSDATE + 7 ;
```

CREATE SNAPSHOT命令的START WITH与NEXT基于SYSDATE。整个数代表一个单独的日期。在前例中，下一次刷新日期是第一次刷新之后的第七天。如果不使用 NEXT，快照将永远不会刷新。如果刷新发生在一个时间段，如每小时或每半小时，它会成为片断的。例如，每小时的刷新率为 $\text{SYSDATE} + 1/24$ ，每半小时的刷新率为 $1/48$ ，每15分钟的刷新率为 $1/64$ 等等。如果总是在星期一进行刷新，语法稍微复杂一些，如下所示：

```
NEXT_DAY(TRUNC(SYSDATE), 'MONDAY')+3/24
```

指定NEXT_DAY代替SYSDATE，24上面的3代表早晨3点。

使用索引允许创建一个索引，当从快照进行查询时，索引可以改进性能。

CREATE SNAPSHOT命令的FORCE/FAST/COMPLETE是复制的方法。COMPLETE是时间最集中的，它重建这个快照，并通过执行一个 INSERT INTO ... AS SELECT *完成，.COMPLETE也是创建最初的快照的命令。如果表比较大，在初始创建快照之后，应该避免使用这个命令。

FAST是缺省值。它使用日志比较两个快照，并只增加在日志中的改变。

注意 在7版本数据库中，如果快速刷新失败，快照刷新被中断，如果没有手工干涉就无法恢复。在8版本中，如果快速刷新失败，将执行完全刷新。只在快照具有快照日志时才能进行快速刷新。

如果不能执行一个快速刷新的话，就强制执行它（例如，一个被中断的刷新）。这个选项在8版本中被取消。

快照的管理在7版本与8版本中并没有显著的改变。最大的改变是在创建子句中增加了选项，大多数的改变与快照处理并不直接有关，而是由于新增加的对象类型，新的创建表命令与新的创建索引命令。不同之处包括：

LOB_STORAGE_CLAUSE与大对象定义有关，大对象的详细信息参见《Oracle8i A Server Application Developer's Guide》。

LOGGING/NOLOGGING指明是否在创建快照时记录重做日志信息。这与 CREATE INDEX NO RECOVER命令相似。它也影响直接装载与直接装载插入。在恢复过程中也会显著影响索引。有关CREATE INDEX的详细信息参阅Oracle8i SQL文档。

CACHE/NOCACHE指明在一个全表扫描之后，快照是否被保存在内存中。更详细的信息参阅《Oracle8i Concepts Guide》中有关内存结构、高速缓存与LRU算法部分。

WITH PRIMARY KEY是在Oracle 8i中的缺省值。通过指明创建一个主键索引快照，主表可以重新组织而不影响快照刷新，涉及使用PRIMARY KEY快照的限制如下：

- 在被引用的表中必须有一个主键。必须是在 CREATE TABLE命令中使用PRIMARY KEY约束。
- 必须使用主键中的所有列。

警告 如果没有在创建快照命令中使用主键，将使用记录的行ID。如果由于数据库恢复而使主表得到恢复，这会产生潜在的问题。恢复将改变行ID的值，可能会使快照无效。为确保不产生这样的问题，在任何数据库恢复之后使用完全选项刷新快照。

USING DEFAULT MASTER | LOCAL指明回滚段的类型。DEFAULT将使Oracle进行选择；MASTER将使用远程结点上的回滚段；LOCAL将使用本地数据库的回滚段。

40.5.2 使用快照刷新组

许多快照，虽然与不同的主表相关，可能具有相同的刷新频率，使它们可以组成一个组。虽然Oracle8i为创建这些快照使用了一个点击的方法，但有必要知道在鼠标后面发生什么。

Oracle将这个能力称为API调用。复制对象的操作不是由标准SQL命令执行的。这些调用通过使用过程执行。有关过程、它们的语法与怎样使用它们的详细信息，参见《Oracle 8i™ Application Developer's Guide》。

40.6 快速快照刷新

快照刷新已被优化以支持大的刷新组。有改进的对子查询快照及空刷新（自从上次刷新以来主表没有改变）的支持。一个单独的刷新组现在可以含有400个快照，在快照组中刷新快照所需要的往返次数也被减少（在8.0.5版本中第一次增加这个功能）。

40.6.1 确定快照中的潜在问题

在数据库管理过程中，活动前的管理是最正确的方法。要为活动前的管理快照过程，DBA必须能够指出潜在的问题，并设计系统防止这些问题的发生，本节指出了这样的一些问题：

1. 大小

在创建快照的过程中，应该避免一种缺陷。虽然快照只用于查询，但它不是一个静态表。大小应基于表副本的更新率，而不是快照自身。例如，如果GERMANY_SPECS表是高度动态的，应该具有一个较大的使用百分比（PCTUSED）以适应更新。表应该调整尺寸以确保不会

产生碎片。

2. 不小心删除了对象

一些快照在快照组中。为找与它们相关的工作号，查询 DBA_REFRESH_CHILDREN表。这些组可以被删除，然而，必须小心以确保没有同时删除对象。因此，删除任何类型的复制组的方法应该由SQL代码完成，而不是由企业管理器完成。企业管理器使 DBA可以通过点击对象删除它们。

删除刷新组的方法使用这个过程：

```
execute DBMS_REPCAT.DROP_SNAPSHOT_REPGROUP(gname=>'[name of group]'
```

drop_contents缺省为FALSE。当使用过程时，必须作为一行输入，否则它们不会被调用。

警告 如果设置drop_contents为TRUE，与该组相关的表也被删除。

3. 丢失快照刷新

当一个主数据库不能完成快照的刷新时，在刷新试图停止之前将再试 16次，这16次并不意味着16次刷新间隔。第一次刷新失败后，快照处理将等待一分钟，然后再试。如果失败，时间成倍增长为两分钟，然后四分钟，时间成指数增长直至到达或超过真正的刷新间隔。当发生这个时，它将按刷新率进行。在 16次之后，Oracle将更新USER_REFRESH表与USER_REFRESH_CHILDREN表中的BROKEN列。BROKEN列表明快照刷新发生了错误。不只在这个表中反映这个问题，Oracle SNP进程也在追踪文件与alert.log中指出刷新问题。

减少一个中断快照的可能性的方法是使用 FORCE选项而不是 FAST选项创建快照。FORCE将选择使用FAST，除非发生了一些事情阻止成功的快速刷新；然后它将执行一个完全刷新。使用这个选项的一个缺点是万一是一个非常大的表，刷新将进行很长时间。如果本次刷新仍在进行，又发生了按时间表规定的下次刷新，这将产生问题。

在DBA确定刷新被打断并矫正了中断快照的问题后，快照可以被手工刷新。Oracle8i提供了一些手工刷新快照的方法，一个方法是执行快照的工作。要这样做，在子结点记录日志并执行以下查询：

```
SELECT RNAME,JOB FROM DBA_REFRESH;
```

结果与下面相似：

RNAME	JOB
GERMANY SPECS	142

运行中断快照的工作，工作名与快照名相同（本地/子）。要这样做，执行下面的过程：

```
Execute DBMS_JOB.RUN(142);
```

响应如下：

```
PL/SQL procedure successfully completed.
```

当这个完成时，验证 USER_REFRESH与USER_REFRESH_CHILDREN表中的BROKEN状态不再是Y。如果BROKEN状态仍然是Y，重复上述过程。如果仍不能修复问题，确信没有其他的问题（如丢失连接），它会阻止刷新进行。

为预活动地监控快照处理并阻止另一个中断的快照，比较快照时间与快照日志中的时间。它们应该相匹配，如果不匹配，可能表明导致快照丢失的问题仍没有解决。

在主结点，使用下面的查询：

```
SELECT MASTER,SUBSTR  
(TO_CHAR  
(CURRENT_SNAPSHOTS,'MM-DD-YYYY HH:MI:SS'),  
.....
```

```
1,20)  
TIME FROM DBA_SNAPSHOT_LOGS;
```

从本地/子结点, 使用下面的查询:

```
SELECT NAME,SUBSTR  
(TO_CHAR  
(LAST_REFRESH,'MM-DD-YYYY HH:MISS'),  
1,20)  
TIME FROM DBA_SNAPSHOTS;
```

这些时间应该相同。如果不同, 需要对快照进行手工管理直至完全解决问题。

4. 控制快照日志增长

快照日志含有主快照的 DML 改变。这决定了向远程结点发送什么。在使用这个日志的所有快照完成后, 日志被净化。然而, 如果快照有问题, 或者其中一个快照不定期地刷新, 快照日志会不受控制地增长, 这会在两个方面产生问题。DBA 应该设置日志大小以使它可以容纳一定的增长总数。如果快照的稳定性是首要任务, 建议使用无限的区间, 这会消除达到最大区间数, 并填满日志所在表空间的可能性。DBA 应该定期检查这个日志以确保不会发生这种问题。如果无法管理日志的大小, DBA 应该净化日志, 这可以由企业管理器工具完成, 或者 DBA 可以使用存储过程或 API 调用 DBMS_SNAPSHOT.PURGE_LOG 执行, 语法如下:

```
Execute dbms_snapshot.purge_log('[master table name]',number,'flag');
```

number 是要清除的历史快照的数量。如果 DBA 想要清除所有的快照日志, 这个数会非常大。当这样使用时要小心, 如果清除了所有的日志, 快照将不得不进行完全刷新。如果表非常大, 这会是一个问题。

注意 标志是一个可覆盖的值。如果设为 DELETE, 或 number 设为 0, 将从最近最少刷新的快照删除日志。

40.6.2 理解快照的限制

一个大表快照的初始创建将可能超出系统的能力。例如, 如果一个有几千万条记录的大表的快照不得不通过不稳定的连线进行复制的话, 最好执行一个脱机实例化。要完成这项工作, 需要执行一些步骤。必须小心进行这些处理, 如果可能, 在一个有很大存储空间的测试环境中进行。步骤如下:

- 1) 在产品数据库中, 为每个主表(将在远程结点具有快照的表)创建一个快照日志。
- 2) 使用一个新的模式, 并且在测试数据库中, 创建一个指向含有主表的产品数据库的快照, 名字必须是唯一的, 理想上应该是远程结点上将使用的名字。
- 3) 如果这个处理必须在产品数据库中完成(不建议这样做), 在 CREATE SNAPSHOT 语句中的链接将指向当前数据库, Oracle 将这个能力称为回环链接。
- 4) 导出这个新的模式, 与新的快照拥有者相同的模式将执行这个导出。
- 5) 删除这个新建的快照, 确保这是手工完成的(使用 API 调用)。如果不是手工完成的, 确保只创建快照, 而没有创建相应的对象。

在远程结点创建一个快照:

- 1) 创建一个空的快照组, 要支持下一步的过程调用, 这一步必须进行。
- 2) 使用下面的过程:


```
DBMS_OFFLINE_SNAPSHOT.BEGIN_LOAD  
(gname=' [groupname]' .sname=>'snapshotname',)
```

这将为将被导入的数据创建一个快照“壳”。

3) 导入这个快照的基表，由前缀为 SNAP\$ 的表标识。

4) 导入完成后，使用过程 DBMS_OFFLINE_SNAPSHOT.END_LOAD 暗示系统导入完成。实现这个处理的最简单的方法是热备份处理，其中执行命令 ALTER TABLESPACE BEGIN-BACKUP。

注意 与不能在 LONG 数据类型上使用 SELECT 创建表一样，快照也不能支持这个 SELECT 语句。

40.6.3 调整快照

由多个表连接组成的复杂快照会严重降低性能。完全测试与调整用于创建快照的查询是必不可少的。这包括在创建快照之前使用 explain plan 与 TKPROF。如果需要，在子结点创建一些单独的表快照，然后创建一个基于物理快照的本地视图。性能将会显著增强，网络负载将降低。

为在库缓存中保留用于刷新数据库的包，最好将这些包钉在内存中。这将帮助阻止从内存中移走包。为了钉住一个包，调用这个包，然后使用 DBMS_SHARED_POOL 包。在钉一个包之前，必须先引用它。最简单的引用包的方法是重新编译它，语法如下：

```
ALTER PACKAGE DBMS_SNAPSHOT.I_AM_A_REFRESH COMPILE;
```

响应为：

```
Package altered.
```

现在，使用下面的语句打包：

```
Execute DBMS_SHARED_POOL.KEEP('DBMS_SNAPSHOT');
```

响应为：

```
PL/SQL procedure successfully completed.
```

这将改进性能，但可能不会显著增强性能，然而，这确实有帮助——特别是对那些经常刷新的快照。

可以因性能被钉的其他包为：

- DBMS_REFRESH
- DBMS_REPUTIL
- DBMS_REPCAT
- DBMS_DEFER

这些包的详细信息，参阅《Oracle 8i Server Replication — Replication Manager API Reference》。

初始化复制包

通过将 PL/SQL 复制包内在化与优化快照刷新，会实现显著的性能改进。

从版本 8.0 开始持续地趋势是，将更多的复制代码移入数据库引擎内。PL/SQL 生成的用于在远程结点复制事务的包已被内在化。这允许复制事务更为有效地在远程结点应用，因为包没有生成，结点可以被更快地实例化。内部的包也更安全，因为它们是不会被乱动的。

40.6.4 使用快照的初始化参数

init.ora表含有一些直接影响快照处理的参数，这些参数在表 40-2中定义。

表40-2 快照的初始化参数

参 数	描 述
JOB_QUEUE_INTERVAL(1-3600)	快照处理被唤醒的间隔。必须小心以确保这个值没有设得过高，不会妨碍快照自己的时间间隔
JOB_QUEUE_PROCESSES (0-36)	快照处理进程数的限制。通常，1就足够了，除非有几个快照，或可能会影响快照进程的大的快照

第41章 复制管理

本章要点：

- 什么是复制
- 多重主机复制
- 传播的类型
- 复制的类型
- 冲突解决
- 快照
- 一些有用的工具
- Oracle 8和Oracle 8i的新特性

41.1 什么是复制

复制是在分布式数据库环境中维护表副本的过程。插入、更新、删除存储在本地并应用于分布式环境。复制对象包括在一台以上机器上存在的表、索引、数据库触发器、包与视图。Oracle支持复制对象的完整拷贝与复制对象的部分拷贝。Oracle使用复制组来轻松复制过程的管理。Oracle也实现了许多复制方法。当一个数据库环境诸如 Oracle支持复制结点的可更新能力时，可能在复制对象的数据之间存在冲突。这种多结点更新能力称为高级复制环境。Oracle提供了一些解决复制冲突的方法。

提示 Oracle 8支持分区表与索引。

Oracle复制使用主结点/快照结点方案。主结点含有所有要被复制到其他结点的对象（表、索引、视图等等）。主组含有一组要被复制到多个结点的对象，快照结点直接与一个并且只与一个主结点相关，它可以含有主结点的所有对象，但通常只含有对象的一个子集。

工作进程是复制过程的重要组成部分。每个主结点需要一个工作进程。快照结点需要一至三个工作进程。这些工作进程在用户定义的工作、净化与按时间表链接时是必须的。需要周期性地初始化这些工作。job_queue_intervals机制控制这些工作按多长时间间隔开始，按秒记录。job_queue_intervals控制工作的时间。

注意 确保在带有五个其他主结点的一个主结点的环境下，开始复制设置之前先设置下面的INIT.ORA参数，每分钟复制一次。

```
global_names= True
job_queue_processes = 7
job_queue_intervals = 60
```

41.2 多重主机复制

复制也可以由多个主机执行，下面通过主群组与快照组的概念进行讨论。

41.2.1 主群组

主群组是复制对象的一个集合。主结点存储所有的对象与所有的数据。通常有一个主定义结点，或一个含有正被复制对象的定义的主结点，作为其他主结点的控制点。这个主定义

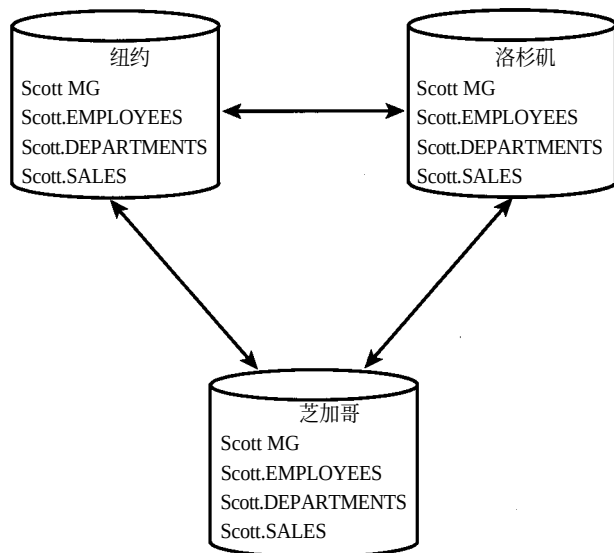


图41-1 多重主结点复制示例

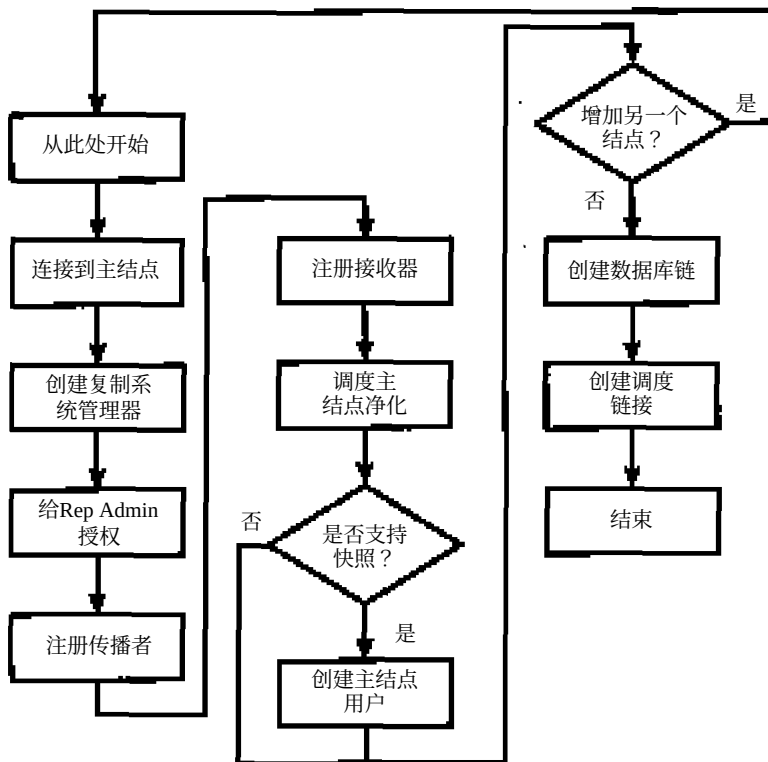


图41-2 设置一个主结点的步骤

结点是DDL改变被应用与复制开始和结束的地方。对数据的改变将从一个主结点推动到另一个主结点，这也称为n种方法复制。

当复制组放在多个机器上时，称他们为多重主群组，多重主群组必须在每个位置含有相同数量的复制对象。这些多重主群组执行真正的对等复制，每个组含有相同的对象与相同的数据。图41-1显示了一个多重主机复制模式。注意复制组与对象在每个结点是相同的。还要注意主组只能与相邻的结点共享数据。这给予了Oracle一种双主机更新路径。如果所有的主结点可以在同一时刻更新所有其他的主结点，潜在的冲突将无法预测，更不用说管理了。

为设置一个主结点进行复制，遵循图41-2中列出的步骤。遵循图41-3中的步骤以设置主群组。

注意 只在Oracle 8i企业版中支持多主机复制。

提示 在尝试创建一个主结点之前，确定你已作为SYSTEM注册。

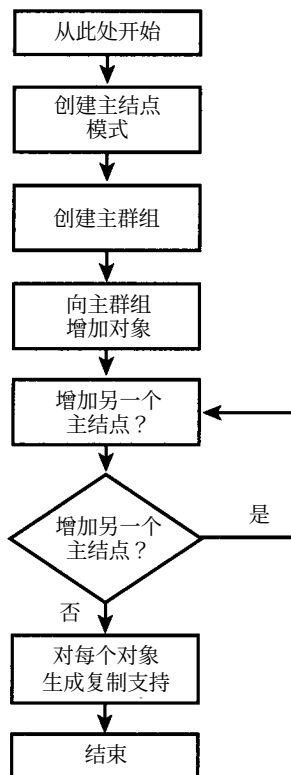


图41-3 设置主群组的步骤

41.2.2 快照组

快照是一个主机对象的子集。快照可以在主群组中含有对象的完全拷贝或只是一个子集。

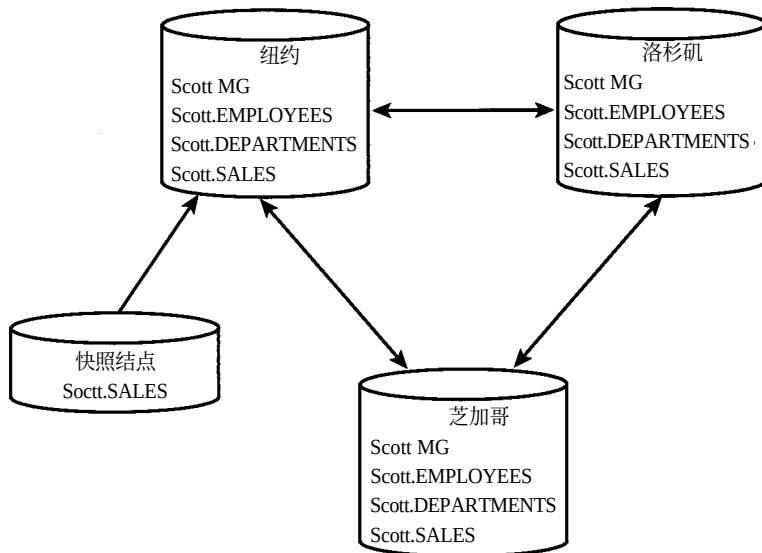


图41-4 多重主结点/快照复制

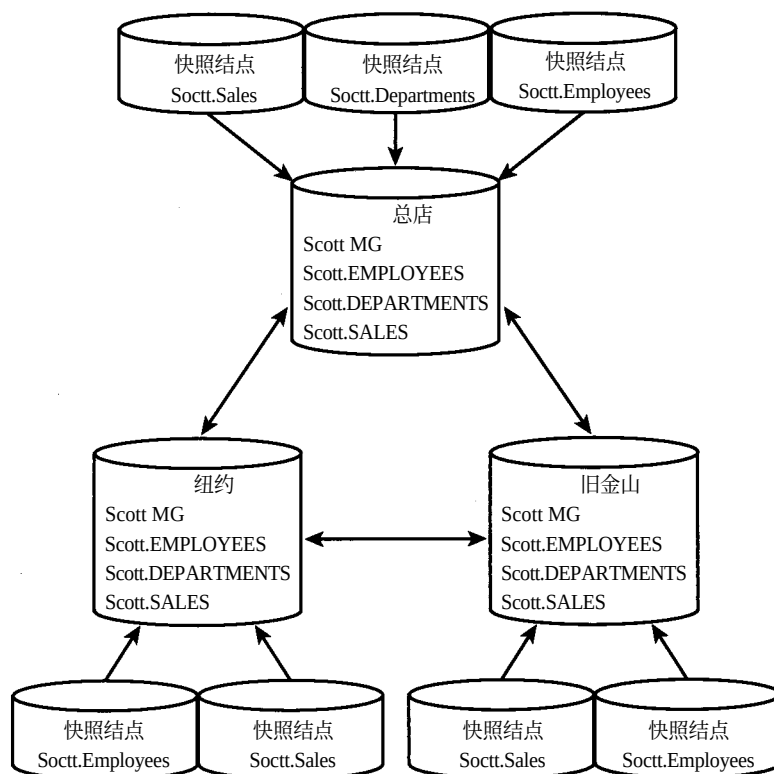


图41-5 复杂主群组/快照组示例

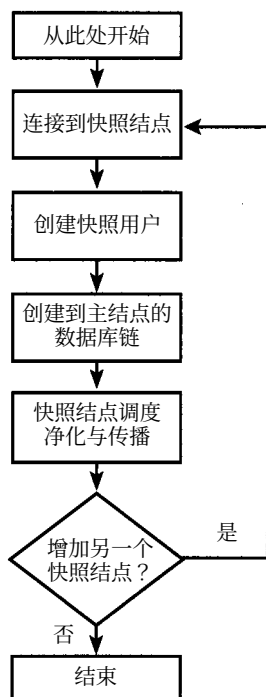


图41-6 设置一个快照结点的步骤

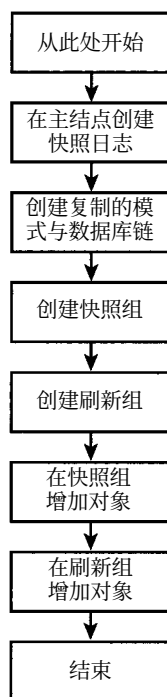


图41-7 设置一个快照组的步骤

快照组与主群组相似，它们都是维护复制的一种便利的方法。快照组只与一个单独的主群组相关，可以被更新。图 41-4显示了一个多重主/快照复制组配置，注意快照组只能与一个单独的主群组共享数据。

现在看一个在图 41-5中含有的主群组与快照组的更复杂的模型。注意总店组也是主定义结点。总店组含有三个分离的快照组，每个部门有一个可更新的快照，负责控制内部的数据。总店组含有所有的对象，旧金山与纽约主群组结点是营业所，它们也含有所有的复制对象，但它们的快照只含有与特定店有关的销售数据与雇员数据。

快照结点也可以很容易地设置，遵循图 41-6中列出的步骤以设置快照结点，在这些步骤完成后，遵循图41-7中的步骤以设置快照组。

41.3 传播的类型

对主群组的改变从一个主结点被推动到另一个主结点。改变从主结点被拉到它的快照结点。对快照结点的改变被推回到它们的相应主结点。

41.3.1 异步传播

异步传递，也称为延迟事务，是当改变被延迟或在它们发生的结点排队然后隔一定时间推动到其他结点。这种推动也可以手工启动，这种方法称为保存并转发方法。图 41-8显示了在高级复制模式下在两个主结点之间的 Oracle 触发器与队列的使用。这种相同的配置也可以被用于以异步方式在快照结点与主结点之间传播改变。

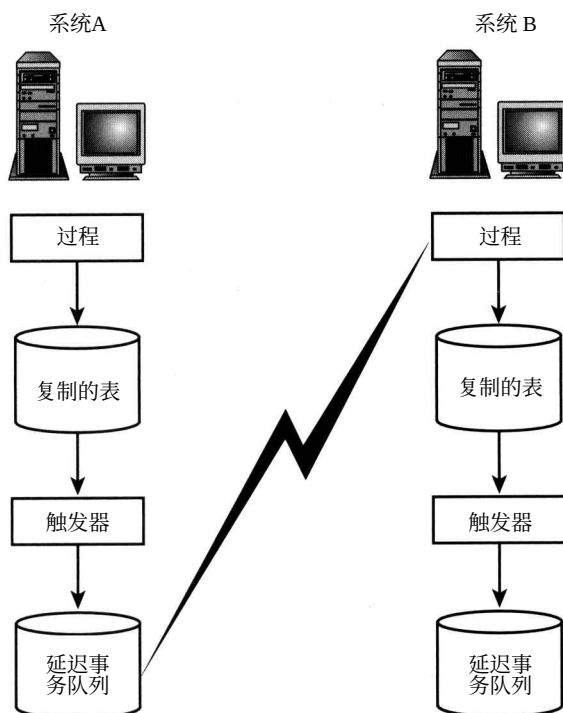


图41-8 在两个主结点之间的Oracle触发器与队列

在高级复制模式中触发器的使用很关键。Oracle使用Oracle的工作队列机制与延迟事务管理这种传播模式。内部的Oracle触发器建立RPC，用于在主结点或快照结点上复制数据的改变。这些RPC是真正为以后的传播进行的存储。工作队列中的工作是一种典型的高级复制模式，包括推动事务的工作、净化整个事务的工作等等。

异步传播也具有高度可用性，即使在快照结点不可获取时，用户仍然可以继续工作。这种传播的方法也执行得更好，因为用户在继续他们的工作之前，不需要等待改变应用到其他结点。然而可能发生数据冲突，正在被复制的数据不立即应用到其他结点，异步传播环境更难于管理。

41.3.2 同步传播

同步传播或实时传播，是在一个单独的事务中，将改变应用到本地复制与所有其他的主复制结点与快照复制结点。这种传播的方法更容易配置，但非常依赖网络与系统资源的随时可用性。图41-9显示了同步复制中包括的少数步骤。

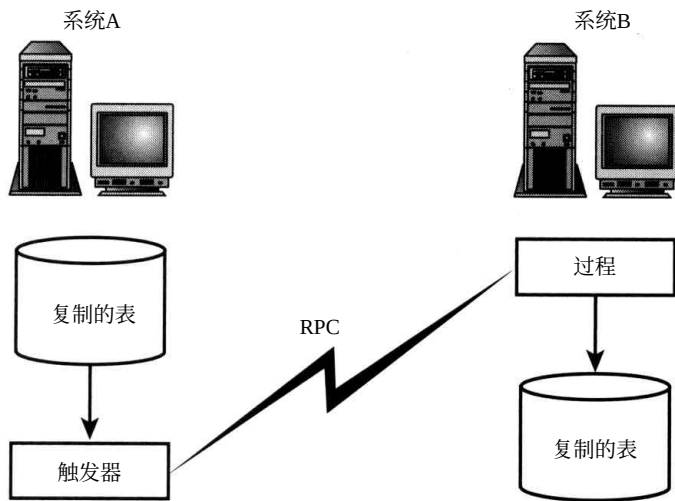


图41-9 同步传播过程

同步复制的优点包括没有数据冲突，使数据可以立即被所有的复制结点使用，并且更易于管理。但通常同步复制环境效率更低，因为用户必须等待改变应用到整个复制模式。

注意 如果在同步传播环境中，任何复制结点是不可获取的，改变不会应用到任何结点。

41.4 复制的类型

在Oracle 8与Oracle 8i复制环境中有四种主要的复制类型：行级（或直接接口）、串行、并行与过程化。

41.4.1 行级复制

行级复制，或直接接口复制，使应用可以使用标准的DML语句更新本地副本。内嵌自动

冲突检测，可以使用内嵌的冲突解决例程。

行级复制更易于实现，因为 Oracle 为真正的传播与冲突检测创建了必要的代码。这种方法也更有灵活性，因为不论使用什么工具，改变都被传播并进行数据的修改。

行级复制的常用对象包括如下：

- <表名>\$RT —— 在每条 DML 语句触发这个触发器，并将改变放入延迟日志队列（只用于 Oracle7）。
- <表名>\$TP —— 当正应用同步传播时，这个触发器与 \$RT 触发器共同使用。
- <表名>\$RP —— 这个包用于在目的地环境中执行 DML 语句，这个包也含有冲突检测（Oracle8）。
- <表名>\$RR —— 这个包与 \$RP 包一起工作，执行必要的冲突解决方案（Oracle8）。

图41-10含有实现行级复制的必要步骤。

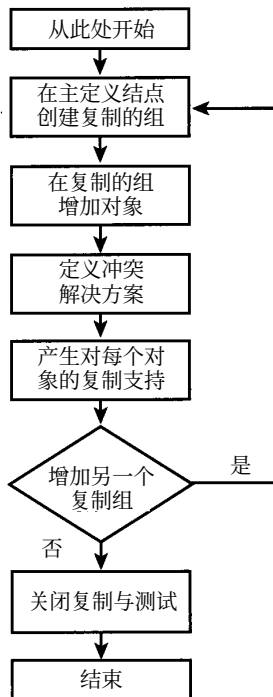


图41-10 行级复制的步骤

41.4.2 串行传播

Oracle 按与初始结点上改变的提交完全一致的次序将改变传播到复制结点。

41.4.3 并行传播

Oracle 使用与 Oracle 用于并行查询、载入、恢复等相同的并行机制同步更新复制结点。

41.4.4 过程化复制

过程化复制也称为封装接口，开发者只书写数据操作的必要过程，Oracle 书写一个封装处理传播，在此没有提供自动冲突检测，没有可用的内嵌冲突例程。这种方法比行级复制更快，因为开发者可以控制被复制的数据。

提示 过程化复制用于批处理最理想，因为应用可以在一个单独的事务中改变大量的数据。数据并没有真正地复制，更新代码被传播到每个结点并在每个结点执行。

图41-11含有实现过程化复制的必要步骤。

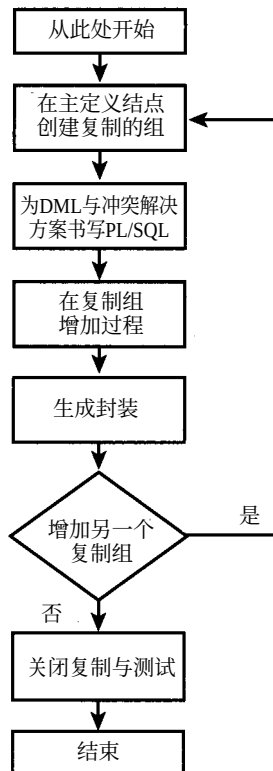


图41-11 过程化复制的步骤

41.5 冲突解决

每当在多环境中更新相同的行时，就有可能产生数据冲突。DBA或开发者必须考虑如果不同结点的事务在几乎相同的时间试图更新相同的记录时，应该做什么。

大多数冲突可以通过预先的应用设计被避免。诸如航空订票的应用不能具有冲突。其他类型的应用，如销售导向应用，可以通过数据的拥有者进行本地更新。有一些应用确实需要数据可以在多个结点更新，这种情况时，需要以这样一种方法设计应用：在冲突发生之前就处理数据冲突。

41.5.1 冲突的类型

有三种类型的冲突：

- 更新冲突，发生在当两个或多个结点在几乎相同的时间试图更新相同的记录时。
- 唯一性冲突，发生在当一个更新产生了一个违反主键或唯一性约束时。
- 删除冲突，发生在当一个事务删除了另一个事务已更新的一行时。

图41-12显示了多结点更新怎样导致数据完整性冲突。在这个例子中，纽约结点更新了一条部门代码，几乎与此同时，洛杉矶结点正在更新相同的部门名称。

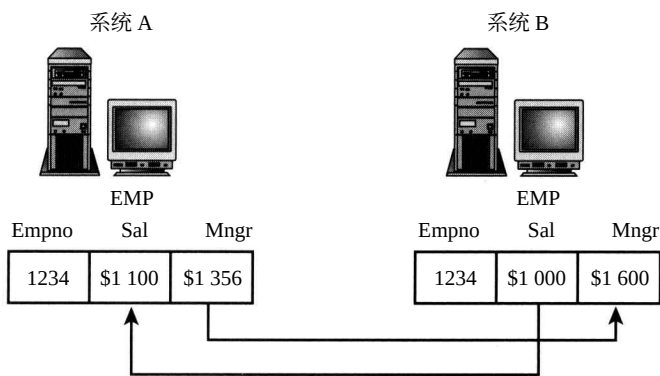


图41-12 多结点更新冲突

41.5.2 避免冲突

避免冲突的第一条建议是限制具有相同数据更新能力的结点的数量，这可以由主结点拥有者达到这个目的。在这种情形下，只有一个结点可以更新数据，所有其他的结点只有特定数据的读权限。更粒状的控制形式能够通过仅运行某个结点更新某些数据列来实现。动态拥有，或令牌环传递，与主拥有不同，具有更新从一个结点移动到另一个结点的数据的能力，这对那些流水作业应用特别有用。在这种情形下，应该设计应用按需要自动改变拥有者。

每当一行将被更新时，需要做以下工作：

- 定位行的当前拥有者。
- 锁定行，当拥有者进行更新时阻止其他更新。
- 改变行的拥有者（更新行域）。
- 执行更新。

图41-13显示了这在应用中是如何实现的。

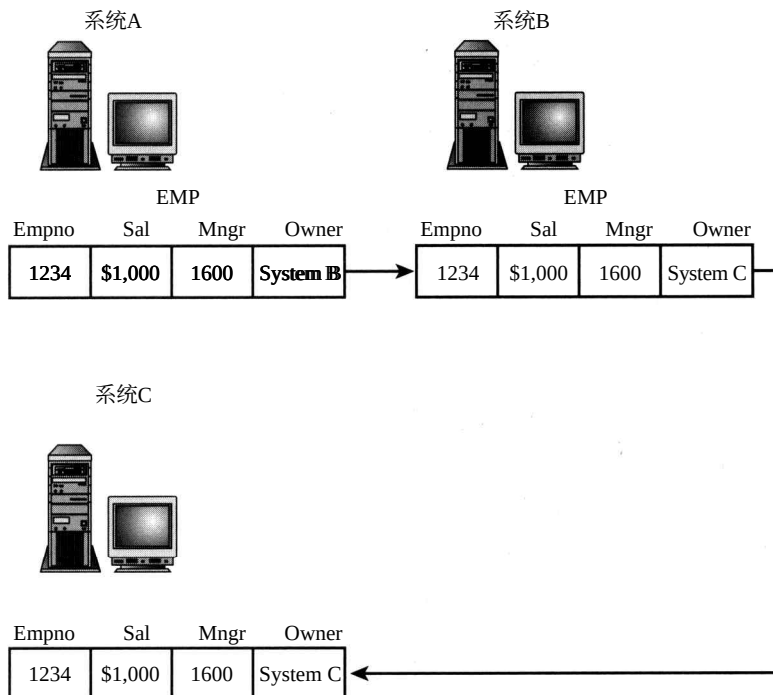


图41-13 动态拥有示例

在每个复制结点创建一系列唯一序列号可以帮助避免整个复制结点的唯一性冲突。另一种观点是将本地序列号作为组合键值的一部分。

可以通过不使用 DELETE DML 语句避免删除冲突。需要删除记录能力的應用可以标志要删除的行，并使用过程化复制将它们定期地清除。

在高级复制环境中的更新冲突很难避免。试着理解所有冲突情形的可能，并配置复制环境以自动处理冲突。最省事的方法，提供一个警告处理就可以查找并修正冲突。

41.5.3 鉴别冲突

Oracle需要能够鉴别与匹配不同复制结点上的相关联的行。Oracle使用主键实现这个功能，如果需要更新主键，需要指明一个替代键。

提示 不允许应用修改复制对象的主键值。

41.5.4 解决冲突

如果不能避免数据冲突，使用 Oracle 的自动冲突解决方案在冲突发生时修正它们。如果没有解决数据冲突，不一致性会产生更多的不一致性。如果 Oracle 的自动冲突解决策略没有实现，数据冲突被简单地记录到日志，不得不由手工进行处理。这种手工方法，在没有及时执行时，将导致数据不一致性的增长。

可以使用列组以检测并解决更新类型的冲突。列组是一个或多个列的集合。这些列在主

结点被指明，方法是首先创建列组，然后向这些列组分配列。图41-14说明了创建列组的必要步骤。列组使你可以配置不同数据冲突的不同解决方案，要注意解决方法不能相互冲突。

如果不能避免唯一性冲突，可以为主键或唯一键约束指派冲突解决方法以在这些类型的冲突发生时解决它们。为解决这些类型的冲突，需要书写你自己的冲突解决方法，因为 Oracle 没有提供这种冲突解决方法。

列组提供了五种Oracle预建的更新类型数据冲突的冲突解决方法：覆盖与放弃值、最小值与最大值、最早与最晚时间戳值、求和与平均值、优先组与结点优先。

覆盖与放弃预建了可以忽略初始结点或目标结点值的解决方法。这种方法意图成为具有多个快照结点的一个单独的主结点。可以设置覆盖与放弃来强迫主结点的改变应用到快照结点。

最小与最大值预建了比较初始结点与目标结点值，并使用两个值中小小的值或者是大的值创建数据收敛的解决方法。

最早与最晚时间戳值预建的解决方法与前面提到的最小与最大值解决方法相同，只是它应用在日期数据域。

求和与平均值预建的解决方法用于只有一个数字域的列组。这种方法的求和部分将目标结点上的原始列值与当前列值的旧值与新值的不同求和，平均值部分平均原始列结点的新值与目标结点上的当前值，这种平均值方法只能用于单主机结点配置中。

优先组与结点优先可以为优先组指派列值的优先级，或者为结点优先选择分配优先级。Oracle使用较低的优先级解决冲突。图 41-15说明了五个销售状态列值及相关优先级。当解决一个冲突时，Oracle使用具有较高优先级的列值更新具有较低优先级列值的结点。结点优先以相同的方法工作，只是给予不同的结点一个优先值。

	优先组	优先级		值
...	结点优先级	1	...	系统A
...	结点优先级	2	...	系统B
...	Sales	1	...	合格
...	Sales	2	...	测试
...	Sales	3	...	合同

图41-15 优先级组示例

还有三个Oracle预建的唯一性冲突解决方法。下面的方法可以分配给主键或唯一键约束：追加结点名为双倍值、追加序列号为双倍值，或废除双倍值。

注意 Oracle没有任何用于删除类型的冲突或主键列更新的预建的解决方法。

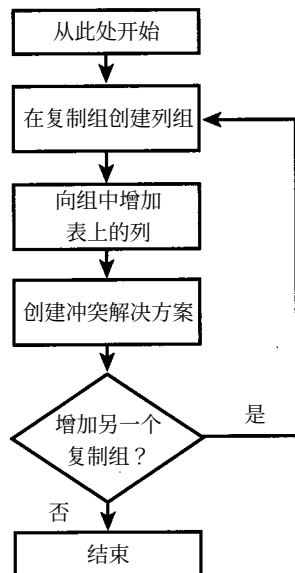


图41-14 创建列组的步骤

除了前面提到的预建的解决方法之外，Oracle还提供了列组的多个冲突解决方法，当一个冲突解决失败后修正这个冲突时，这特别有用。图 41-16说明了多重冲突复制方法是怎样实现的。

可以为解决一个特定的冲突分配多个方法。例如，如果时间信息不能解决一个冲突，如果配置了结点优先级，Oracle仍然能够解决冲突。

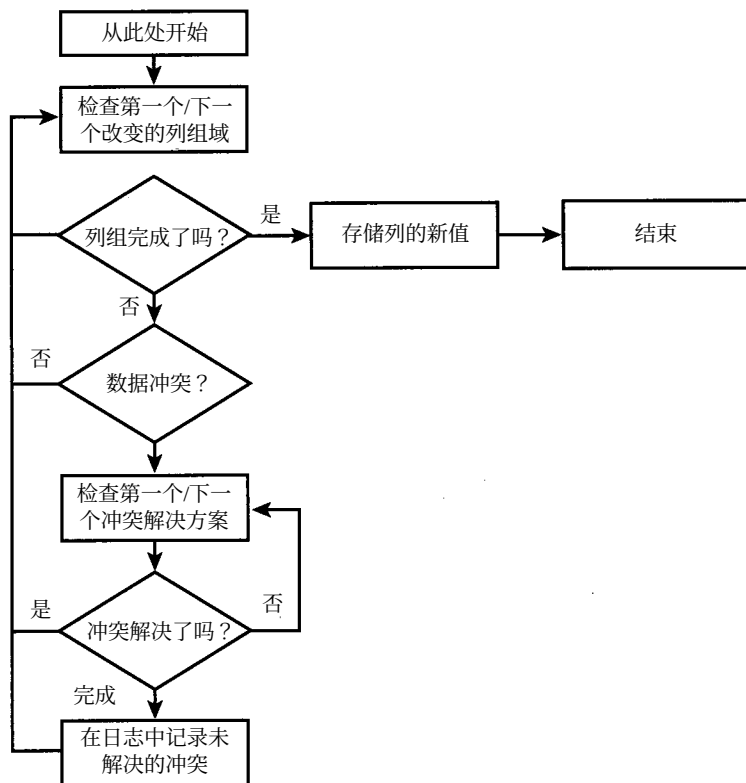


图41-16 多重冲突解决示例

41.6 快照

快照是主结点表的一个副本。多个主结点表可以从其他的主结点进行更新，其中快照从一个单独的主结点使用刷新进行更新（参见图 41-17）。将刷新考虑为一个批处理工作。快照可以使用 WHERE 子句含有主结点数据的子集，当不需要在每个远程结点维护整个数据集时，这特别有用。快照的改变被推进一个单独的主结点，然后这些改变被传播到其他主结点，如前面讨论过的那样。主结点的改变被拉进快照结点，这些改变需要一个专有网络连接，使销售人员连接到网络时，从销售人员的便携式电脑刷新数据成为可能。

提示 快照可以用于减少网络流量。当使用异步传播方法时，可以通过传播的定时控制网络流量。

快照体系结构是非常直接的，取决于快照是否是可更新的（参见图 41-18），或者快照是否是只读的（参见图 41-19）。如果快照是可更新的，如图 41-18所示，有一个快照日志含有受

更新影响的行的主键与/或ROWID。刷新可以是快速刷新，其中只应用快照的日志内容，或者是一个完全刷新，其中整个快照表被重新拷贝或重新刷新。快照日志也可以含有快照上主结点子集的筛选信息。

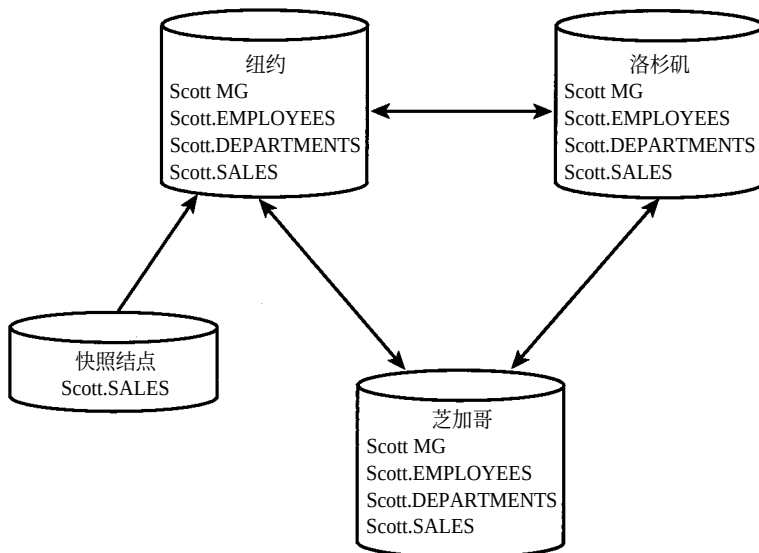


图41-17 一个单独的主结点的快照更新

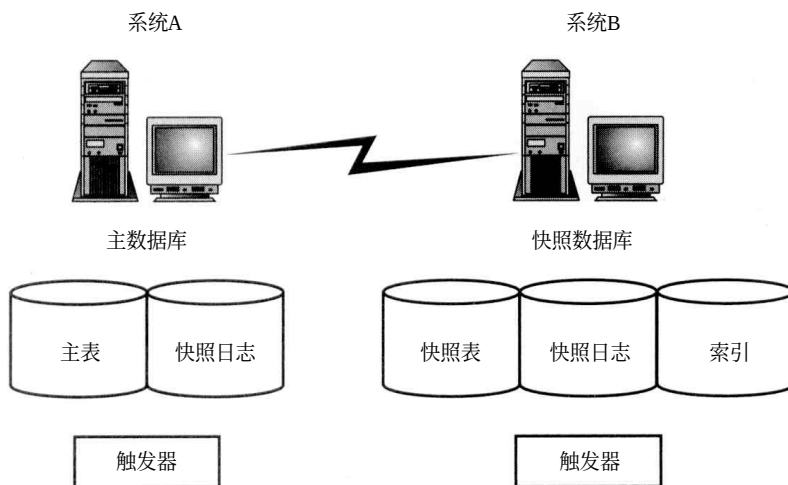


图41-18 可更新快照示例

注意 如果快照不依赖于ROWID并且主表被截断或重新构造，不能使用快照日志的快照刷新功能。

有三种可用的快照类型：主键快照、ROWID快照与复合快照。

主键快照是Oracle 8环境中的常用快照类型，改变只基于主键被记录并传播。

ROWID快照的维护与Oracle 7具有兼容性。

复合快照含有SELECT语句，其中有DISTINCT或其他集合函数、CONNECT BY 子句、

一个子集操作（UNION、INTERSECT或MINUS）。复合快照不能使用快照刷新技术，并且如果大量数据被传播，应该避免使用复合快照。

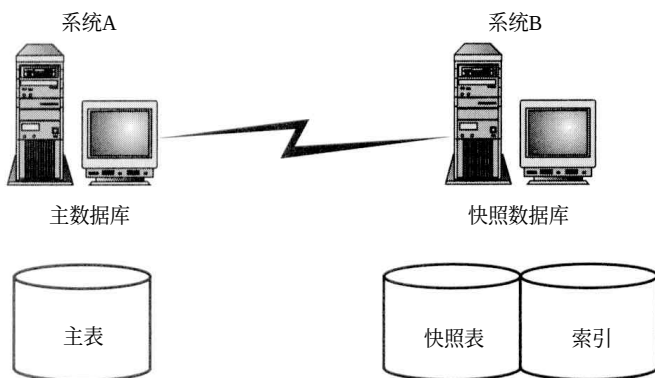


图41-19 只读快照示例

图41-20显示了简单快照与复合快照的不同。简单快照查询的性能是可接受的，这个快照方案可以使用快速刷新。复合快照早已将数据进行了连接，所以查询性能将更好，但代价是必须进行完全刷新。如果刷新是较频繁的，使用复合快照。如果需要刷新的速度与效率，使用简单快照。快照索引总是基于主结点的主键创建的，索引名为 I_SNAP\$_snapshot_name。只为可更新快照创建快照日志。

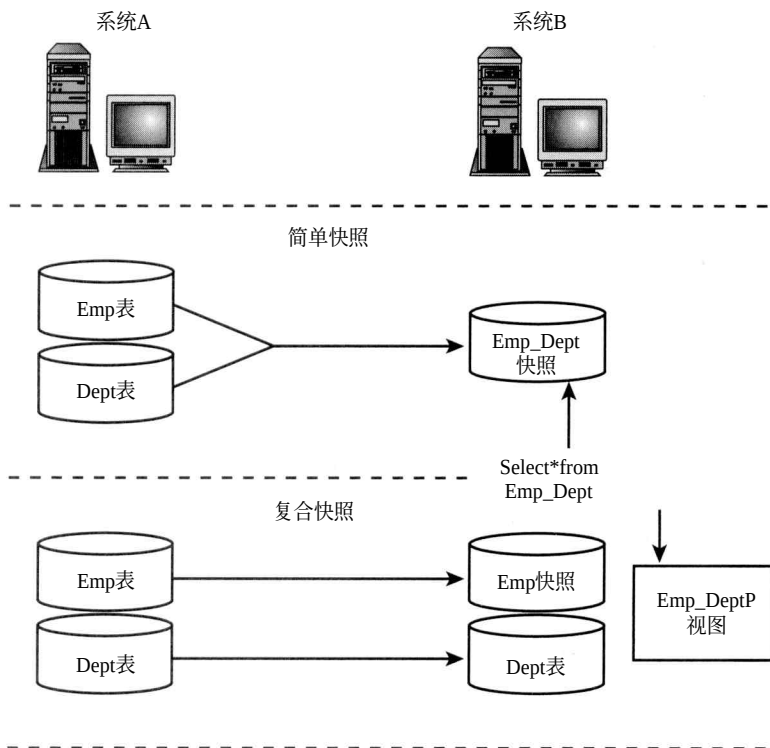


图41-20 简单快照与复合快照示例

只读快照使用许多相同的维护机制，但它们不需要属于一个快照组。一个只读快照是只读的，因为在CREATE SNAPSHOT语法中省略了FOR UPDATE子句。

使用WHERE子句创建具有主结点数据子集的快照减少了刷新时的网络通信量（意为更佳的刷新性能），减少快照数据存储需求，并且可以用于只为用户提供他们需要的数据。后一种方法可以用作一种安全手段，因为只有需要在一个结点存取的敏感数据才真正存储在那个结点上。

Oracle支持主结点列含有以下数据类型的快照：NUMBER、DATE、VARCHAR2、CHAR、NVARCHAR2、NCHAR、RAW、ROWID、LOB、BLOB及CLOB。

注意 在Oracle 8i与Oracle 7版本7.3的混合环境中不支持LOB数据类型。

展开模板

展开模板使在分布式环境下快照的安装对 DBA而言既简单又安全。展开模板可以像简单快照一样简单，如果需要，也可以很复杂。展开模板给予 DBA对整个安装的控制，在不同结点重复运行安装的能力，并根据授权用户列表帮助进行安全管理。

图41-21说明了建立一个展开模板的必要步骤。

在建立一个展开模板之后，需要创建一个脚本，它执行展开模板安装的必要步骤。这称为实例化。实例化可以是联机的，也可以是脱机的。联机与脱机实例化的区别在于脱机脚本同时含有创建快照对象的DDL与填充快照的DML。联机实例化只含有DDL，像常用刷新机制一样填充快照。

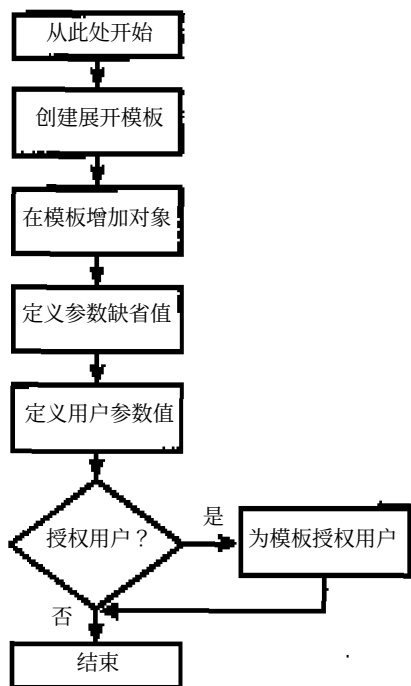


图41-21 建立展开模板的步骤

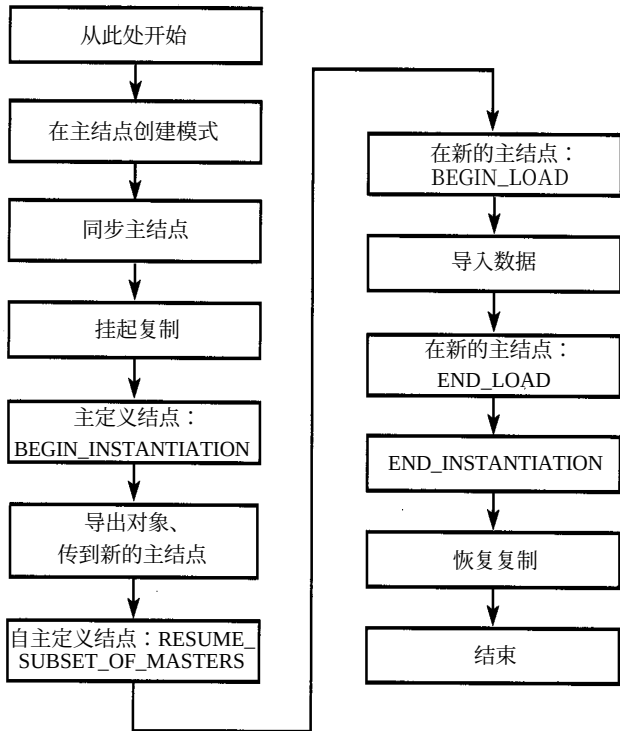


图41-22 主结点脱机实例化示例

图41-22展示了主结点上脱机实例化的必要步骤。图 41-23展示了快照结点上脱机实例化的必要步骤。

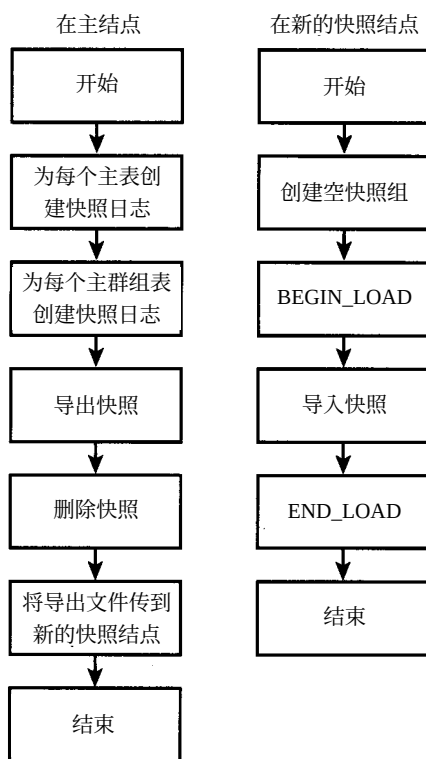


图41-23 快照结点脱机实例化示例

41.7 一些有用的工具

下面是用于创建与维护高级复制选项的包清单：

DBMS_REPCAT_AUTH	这个包用于从系统管理员帐户授予并回收 SYS 权限。
DBMS_REPCAT_ADMIN	这个包创建复制系统管理员帐户、授予并回收角色等等。
DBMS_REPCAT	这个包是高级复制模型的主力，执行许多类型的操作。这个包向复制组增加主结点，创建主组与快照结点组，启用并禁止传播，生成包、触发器与过程等等。
DBMS_REPUTIL	这个工具在会话级使能与禁止复制。
DBMS_OFFLINE_OG	这个工具使用户可以实例化结点，从主结点向其他复制结点拷贝数据等等。
DBMS_RECTIFIER_DIFF	这个工具比较两个复制表，如果发现不同，可以使它们同步。
DBMS_DEFER_SYS	这个工具使 DBA 可以计划时间表、执行与删除排队的事务。
DBMS_DEFER	这个工具用于建立延迟调用。
DBMS_DEFER_QUERY	这个工具提供获取传递给延迟调用的参数，并也可以

用于诊断报告。

DBMS_SNAPSHOT

这个工具使DBA可以维护快照与快照日志。

DBMS_OFFLINE_SNAPSHOT

这个包用于实例化快照，特别是大的快照。

DBMS_REFRESH

这个工具在快照结点初始化快照组的一次刷新。当远程没连接的PC连接到网络并且想要访问他们的数据时，它是非常有用的。

41.8 Oracle 8与 Oracle 8i的新功能

Oracle 8引入了许多性能改进与新的复制功能，这将在本章的剩余部分讨论。

41.8.1 Oracle 8复制的新功能

Oracle 8.0版本增强了性能并增加了一些功能，下面逐条列举。Oracle也在Oracle 8复制模型中增加了对LOB复制的支持。

功 能	描 述
延迟事务的并行传播	这个功能通过并行化数据的同步与异步传播极大地改进了性能
内在化复制触发器	这个改进减小了开销，需要更少的系统管理，并改进了响应时间性能
减少的数据传播	这个功能减少了通过网络传播的数据总量
基于子查询的快照子集	子查询的常用类型现在可以被快速刷新
LOB支持	Oracle8现在支持BLOB、CLOB与NCLOB的复制
主键快照	Oracle8继续支持ROWID快照，但这个新功能将成为缺省的快照模式。这个功能也允许重新组织主表并维护快速刷新功能及主机快照日志的一致性
分区表与索引	Oracle8支持分区表与索引的复制
增强安全模型	Oracle8改进了同步与异步环境下的一致性，事务极少有可能由于缺少权限而失败

41.8.2 Oracle 8i复制的新功能

下面包括了一些Oracle 8i的新复制功能。

特 性	描 述
内部应用包	为继续Oracle8内在化触发器与所需要的函数，Oracle8i内在化用于将改变应用到远程结点的PL/SQL生成的包
刷新组	这项技术导致快速快照刷新，一个单独的刷新组现在可以含有400个快照
参数化快照展开模板	这项功能将推动非常巨大的应用的展开，例如自动销售、国际化组织等等
列级快照子集	Oracle通常可以支持快照结点上的记录数量的限制，但Oracle8i现在也支持快照结点上的记录数量的限制

41.9 小结

复制无疑是Oracle的强项之一，Oracle继续支持并增强它的复制模型。本章介绍了Oracle复制模型，以及所有的术语与不同组件的体系结构。对每个复制选项与潜在的竞争问题的理解将促进对Oracle的高级复制功能的有效管理。