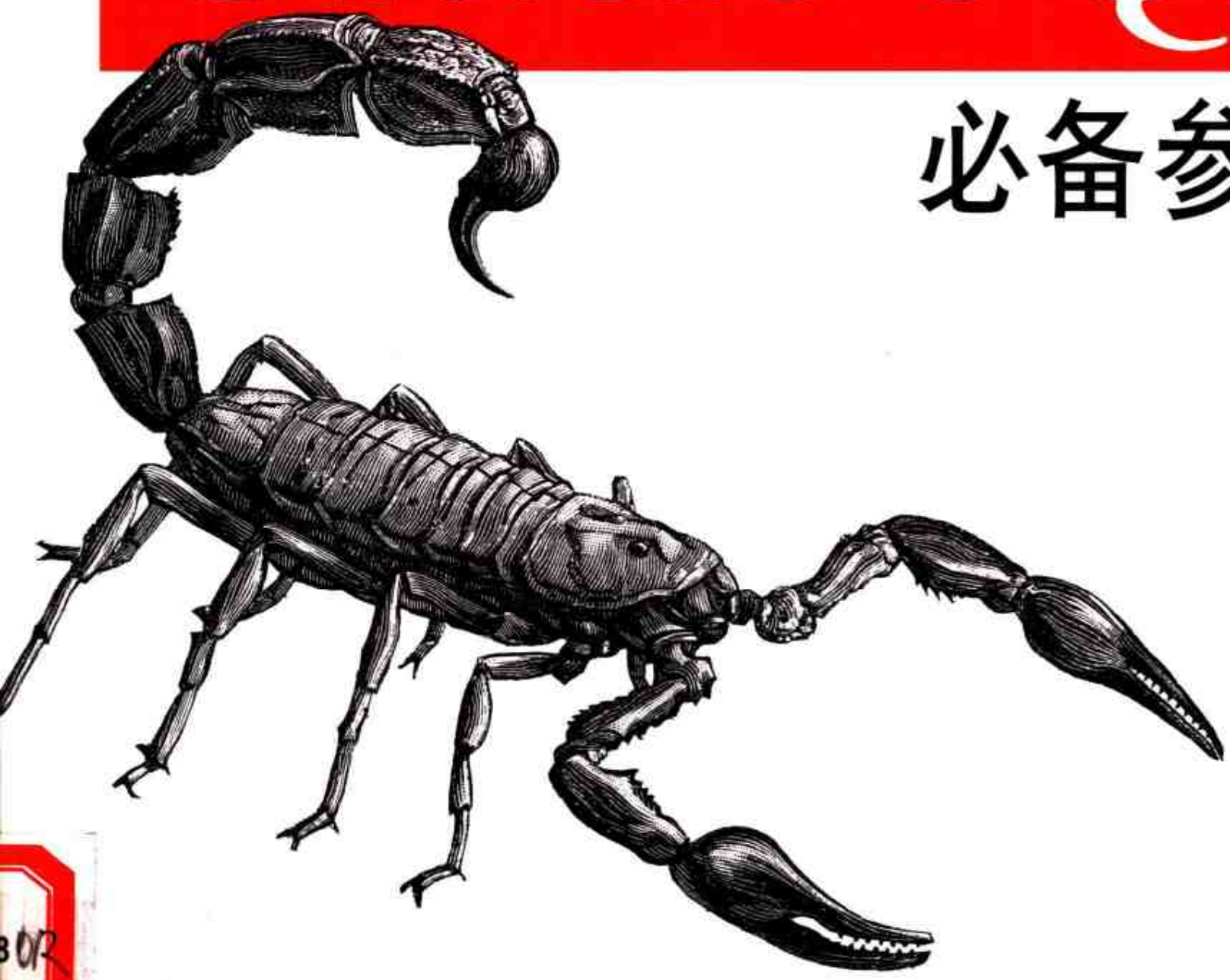
Oracle SQL: The Essential Reference

涵盖 Oracle8 & Oracle8i

Oracle SQL

必备参考



David C. Kreines 著
Ken Jacobs 序
吴安青 薛涛 卫红权 译

O'REILLY®
中国电力出版社

Oracle

O'REILLY®

Oracle SQL 必备参考

SQL（结构化查询语言）是关系数据库管理系统的核心。其语言用于查询数据库、创建新表、更新和删除数据，并设置数据库中的权限。因为第一版的SQL在20世纪70年代产生，所以该语言已经应用于每一个数据库产品中，并被ANSI和ISO采纳为标准。尽管SQL的历史很长，而且关系相对简单，但真正精通该语言的人毕竟是少数，本书刚好是超出人们注意范围的有关SQL的进一步深入。

本书适用于任何一个需要使用SQL访问Oracle数据库的人——包括开发者、DBA、设计者和常用用户。它包含了大家需要掌握的标准SQL的所有信息，而且以单一、简明的参考方式包含了Oracle在该方面的扩展内容。它包含了快速参考表、性能提示和语句语法摘要、数据类型以及其他有用的SQL相关信息（也包括SQL从业者通常需要掌握的SQL*Plus和PL/SQL信息），所有的内容都以可访问的格式组合在一起。本书包含便于参考的内容：

- SQL的基本元素（命名需要、数据原型、数据转换、操作符等等）
- SQL DDL（数据定义语言）、DML（数据操纵语言）和函数
- SQL*Plus和PL/SQL语句及选项

本书也包含了怎样调试SQL语句的扩展内容讨论，而且使读者在使用SQL的时候可以使用Oracle优化器得到数据库的最好性能。本书覆盖了Oracle8、Oracle 8*i*和以上版本。

"Dave Kreines在Oracle团体中无私奉献了多年。如果对本书进行调查，就会发现作者在本书的创作过程中倾注了大量的心血。如果希望手头有可靠的信息，就应该拥有并经常参考本书。"

——Marlene Theriault，《Oracle8i Networking 101》的作者

"本书是大家书架上甚至公文包中的最优秀的参考书。无论大家是刚开始关注Oracle，还是在Oracle运行在640K RAM的时期就已经熟悉了Oracle，都将发现本书是SQL语法和使用信息的一本独立参考手册。指出需要使用什么样的语句是大家成功的一半。本书提供了赢得胜利所需要的内容。"

——Mattew F. Reagan，Centocor公司的资深的技术管理员和DBA

ISBN 7-5083-1102-7

9 787508 311029

O'Reilly & Associates, Inc. 授权中国电力出版社出版

ISBN 7-5083-1102-7

定价：49.00元

TP311.138/R
2K279

Oracle SQL必备参考

David C. Kreines 著
Ken Jacobs 序
吴安青 薛涛 卫红权 译

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

O'Reilly & Associates, Inc. 授权中国电力出版社出版

中国电力出版社

图书在版编目（CIP）数据

Oracle SQL 必备参考 /（美）克雷尼斯（Kreines, D. C.）著；吴安青，薛涛，卫红权译．－北京：中国电力出版社，2002.11

书名原文：Oracle SQL: The Essential Reference

ISBN 7-5083-1102-7

I. O... II. ①克 ... ②吴 ... ③薛 ... ④卫 ... III. 关系数据库－数据库管理系统，SQL IV. TP311.138

中国版本图书馆 CIP 数据核字（2002）第 084654 号

北京市版权局著作权合同登记

图字：01-2002-1198 号

©2000 by O'Reilly & Associates, Inc.

Simplified Chinese Edition, jointly published by O'Reilly & Associates, Inc. and China Electric Power Press, 2003. Authorized translation of the English edition, 2000 O'Reilly & Associates, Inc., the owner of all rights to publish and sell the same.

All rights reserved including the rights of reproduction in whole or in part in any form.

英文原版由 O'Reilly & Associates, Inc. 出版 2000。

简体中文版由中国电力出版社出版 2003。英文原版的翻译得到 O'Reilly & Associates, Inc. 的授权，此简体中文版的出版和销售得到出版权和销售权的所有者 O'Reilly & Associates, Inc. 的许可。

版权所有，未得书面许可，本书的任何部分和全部不得以任何形式重制。

书　　名 / Oracle SQL 必备参考

书　　号 / ISBN 7-5083-1102-7

责任编辑 / 陈维宁

封面设计 / Ellie Volckhausen，张健

出版发行 / 中国电力出版社（www.infopower.com.cn）

地　　址 / 北京三里河路 6 号（邮政编码 100044）

经　　销 / 全国新华书店

印　　刷 / 北京市地矿印刷厂

开　　本 / 787 毫米 × 1092 毫米　16 开本　27 印张　395 千字

版　　次 / 2003 年 4 月第一版　2003 年 4 月第一次印刷

印　　数 / 0001-5000 册

定　　价 / 49.00 元（册）

O'Reilly & Associates 公司介绍

为了满足读者对网络和软件技术知识的迫切需求，世界著名计算机图书出版机构O'Reilly & Associates公司授权中国电力出版社，翻译出版一批该公司久负盛名的英文经典技术专著。

O'Reilly & Associates 公司是世界上在 UNIX、X、Internet 和其他开放系统图书领域具有领导地位的出版公司，同时是联机出版的先锋。

从最畅销的《The Whole Internet User's Guide & Catalog》（被纽约公共图书馆评为二十世纪最重要的50本书之一）到GNN（最早的Internet门户和商业网站），再到 WebSite（第一个桌面PC的Web服务器软件），O'Reilly & Associates 一直处于Internet发展的最前沿。

许多书店的反馈表明，O'Reilly & Associates是最稳定的计算机图书出版商——每一本书都一版再版。与大多数计算机图书出版商相比，O'Reilly & Associates公司具有深厚的计算机专业背景，这使得O'Reilly & Associates形成了一个非常不同于其他出版商的出版方针。O'Reilly & Associates所有的编辑人员以前都是程序员，或者是顶尖级的技术专家。O'Reilly & Associates还有许多固定的作者群体 —— 他们本身是相关领域的技术专家、咨询专家，而现在编写著作，O'Reilly & Associates依靠他们及时地推出图书。因为O'Reilly & Associates紧密地与计算机业界联系着，所以O'Reilly & Associates知道市场上真正需要什么图书。

作者简介

David C. Kreines 是 Rhodia 公司（Rhone-Poulenc S.A 的辅助机构）的数据库业务主管，而且是《Oracle Database Administration:The Essential Reference》（O'Reilly & Associates 于 1999 年出版）一书和《Oracle Scripts》一书（O'Reilly & Associates 于 1998 年出版）的作者之一。从 1985 年开始，Dave 已经开始以开发者和数据库管理员的身份与 Oracle 打交道，从 PC 到大型机的大量不同类型的平台都有所涉及。他通过了 Oracle 的 OCP DBA 认证，同时还经常参与美国和欧洲的 Oracle 会议、用户组织和刊物出版。Dave 已经担任了两届国际 Oracle 用户组织美国组（International Oracle Users Group-Americas，IOUG-A）的主席，并在委员会中呆了十年。

封面介绍

本书封面上的动物是一只蝎子。化石记录显示，蝎子是第一代节肢动物之一，而且在一些化石标本中的有腮动物可以显示蝎子是从海洋中的生物祖先进化而来。当今的蝎子是陆地居住生物，生活在岩石或沙地下，在夜间出动捕食昆虫。科学家掌握的大约有 1300 多种蝎子，大小从 1 英寸到 8 英寸不等，而且在颜色上也各不相同，有棕黄色，也有绿色和黑色。

蝎子通过脚上的感应触发器检测空气中的变动来定位猎物，利用尾部毒针散发出的麻醉毒液注射到昆虫身上，从而进行捕食。蝎子也使用毒针对敌人进行防御，包括黄蜂、蜈蚣和蜘蛛，也包括蜥蜴、鸟类和小的哺乳动物。

蝎子交配的礼节包括精细的求爱舞蹈，此时雌蝎子与雄蝎子通过脚进行交互，雄蝎子将精囊分泌在岩石或树枝上，雌蝎子在经过时吸取精子。对一些种类的蝎子而言，怀孕期可以持续一年半载，然后雌蝎子孵化出很多年轻的生命，这些小蝎子将在母体的背上度过一到两周的时间。

蝎子产生的毒液对人类来说是致命的。不要在夜间在荒凉的地方赤脚行走，或者，你需要在前面点一盏夜灯 —— 蝎子会在紫外线下发出荧光。

目录

序 1

前言 17

第一章 SQL 元素 25

词汇习惯 26

SQL 命名 28

架构对象 28

数据类型 31

数据转换 38

关系操作符 39

SQL 语句的结构 45

SQL 语句 49

第二章 数据定义语句 57

按任务列出的 SQL DLL 语句 57

SQL 语句的语法 64

第三章 数据操纵和控制语句 139
根据任务说明SQL DML和控制语句 139
SQL语句语法 140

第四章 公共SQL元素 167

第五章 SQL函数 179
聚集函数 180
数字函数 187
字符函数 197
日期函数 211
转换函数 220
其他函数 229

第六章 SQL*Plus 244
命令行语法 244
SQL*Plus编辑命令 248
规定SQL*Plus输出的格式 252
各种各样的SQL*Plus命令 263
SQL*Plus变量和相关命令 281
SQL*Plus系统变量 285

第七章 PL/SQL 307
PL/SQL结构 308
块头 310
声明部分 310
执行部分 326
异常部分 346
过程和包 351
触发器 367

第八章 SQL 语句调试 372
使用 EXPLAIN PLAN 373
使用 Oracle 的 SQL 跟踪工具 383
SQL*Plus 调试助手 394
改进查询性能 400

附录 SQL 资源 411

词汇表 415

序

SQL：庄严的历史和重要的将来

SQL语言是数据库管理的事实标准。对开发者或数据库管理员来说，熟练掌握SQL、编程语言知识及应用程序业务需要的知识一样重要。本书可以成为成功发挥Oracle8*i*中SQL实现的必不可少的指南。

SQL具有漫长而古老的历史，它不仅是当今电子商务IT系统中的重要角色，而且具有美好的未来。本序中描述了SQL的起源、演变和未来，在你阅读这本优秀语言参考时，希望它能帮助你加深对SQL优越性的认识。

编程和数据访问语言

第二次世界大战时期，为了军事应用，人们开发了多用途的可编程计算机。1951年，产生了第一台商用多用途计算机UNIVAC I。从那时到现在，人们已经开发了许多代编程语言。每一代编程语言都通过自动机械任务提高了程序员的生产力，允许程序员集中精力研究与应用程序相关的高层概念。

最早的程序通常都是采用机器码编写的，这些机器码数字对应于程序员希望存储在机器内存中的指令。允许程序员使用名字代替指令数字和内存位置的汇编语言开发于20世纪50年代早期。高级编程语言的开发标志着程序员工作所在的语义层有了

重大突破。这样的编程语言相继发明出来，从Fortran（1957）到C（1972）以及到Java（1995），都先后经历了成长期和衰退期。除Fortran、C和Java之外，Algol、COBOL、Ada、C++和Basic也都是我们已用于开发应用程序和系统程序的一些重要编程语言。

与如此众多的过程化编程语言相反，目前只有一种应用十分广泛的数据访问语言：SQL。SQL是非过程化数据访问语言，因为它让数据库管理系统负责决定怎样处理数据的查询问题。应用程序的程序员不必关心产生理想结果的数据访问路径和处理步骤。与底层机器代码和汇编语言相比，因为采用高级过程化编程语言更容易编写应用程序，所以SQL语言使得访问应用程序数据更加容易，尤其是还可以采用交互方式进行。SQL允许应用程序开发者全神贯注于业务逻辑，而不必关心使用索引或指针链提取或更新数据的问题。

虽然SQL在20世纪70年代中后期开发，并仍然在不断进化发展，但似乎不可能被其他语言取代。与过程化编程语言不同（对于过程化编程语言，主要的语言几乎不会被完全废止），当今大部分数据库管理系统都实现了SQL的某种方言。Rosetta石头（译注1）上包含的数据揭示了古埃及语言和象形文字的奥秘，因而可以引导大家更好地理解古埃及的历史和文化。当今，许多人则把SQL当作揭示企业数据库中数据值和信息的语言。

不同厂家的实现产品中有多种方言，但大多数SQL语言在当今市场上大部分商业数据库管理系统中是相同的。可以确信，存在其他数据访问语言，其中一些在大学内得到进一步开发，而另一些则在商业产品中得以实现，但是没有一种数据访问语言有SQL如此成功或者实现如此广泛。加利福尼亚大学伯克莱分校的Michael Stonebreaker教授曾经将SQL称为“银河系的数据语言”。

SQL的起源

那么，SQL是怎样产生、进化并发展的呢？20世纪70年代，SQL的故事开始于美国加利福尼亚州圣何塞旁边的IBM研究实验室。在IBM工作的数学家和研究员Ted Codd创造了数据管理的正式理论，并写了有重大影响的论文，名为“A Relational Model of Data for Large Shared Data Banks”，1970年6月发表在《Communications

译注1：古埃及文物。

of the ACM》上。他定义了关系数据模型，该模型由数据结构（行和列组成的表）、对数据的操作（如选择、投影和连接）、确保数据连续性的完整性规则（例如主键和参照完整性）等组成。

Codd对关系模型的严格数学定义可以定义保持数据完整性和最小化冗余的数据库设计过程。所谓的规范化理论定义了第三范式，这里，数据库中的每张表都有可以惟一定义表中每一行记录的主键，而且行中的每一列都依赖于主键。特别是，用第三范式设计的数据库能支持在数据库设计时不能预期的应用程序和查询。

Codd还定义了数学数据操纵语言——DSL/Alpha。该语言是基于数学集合论建立的，而且可以用于快速查询并操纵组成关系数据库的数据表。Codd证明了可以使用自己定义的操作在任何查询方式下操纵关系数据库，所以可以得到任何组成数据库的结果。他称这种性质为关系完备性（relational completeness）。

与传统上非关系数据库中编写用复杂指针链定位的程序相比，Codd定义的语言功能非常强大。与占用几页的程序相比，Codd的语言可以在少数几行内查询如“找到比经理干得更多的职员”这样的问题。

20世纪70年代早期，基于Codd的思想，以IBM研究分队的形式形成的小组开发了关系数据库管理系统原型。由Frank King领导的名为System R的工程启动了。该工程的目标是开发一个完全支持SQL的关系数据库原型，但仍然传送现存非关系数据库的关键属性，包括多用户支持、事务、安全性和良好的性能。

System R小组认识到，Codd的数学DSL/Alpha语言对没有数学基础的人来说太难理解，所以，他们创建了名为SQUARE的语言，代表以关系表达式形式指定的查询。尽管改进了DSL/Alpha，SQUARE还不适合于键盘输入，因为它需要当时还很难表示的下标。

该小组然后决定将SQUARE的思想改为基于英语关键字的方法，因为这样比较容易录入。他们扩展并改进了这一新语言，并称之为SEQUEL，代表结构化英语查询语言（Structured English Query Language）。因为商标问题，该名字随后被改为SQL。SQL的一般发音为“sequel”，有时也被发音为“ess-queell”，它们都很常见。1974年，Don Chamberlin和Ray Boyce合写了题目为“SEQUEL:A Structured English Query Language”的论文，并发表在1974年5月的《the Proceedings of the May

1974 ACM SIGMOD Workshop on Data Description,Access,and Control》学报上。这是有关这门即将成为SQL的语言的最先广泛流传的论文。

1995年，包括早期开发System R和SQL在内的小组成员为了25周年的纪念而重新团聚。他们回忆起这些人和该工程，而且提供了有关SQL开发的有价值的观点。他们的讨论在WWW上可以找到，URL为：*http://www.mcjones.org/System_R/SQL_Reunion_95/index.html*。

SQL语言

SQL（以及Codd最初的数据操纵语言）的主要功能是，它以非过程化形式加快了数据集合的操作，而不需要程序逐个提取记录并指定处理每条记录的步骤顺序。它不像大部分语言编写的程序那样指定了执行的特定步骤顺序，SQL语句表达了用户理想的结果，而且使数据库管理系统负责产生尽可能高效的结果。SQL语句指定了要在行集合上执行操作（如过滤、分组和排序），而且数据库系统决定了访问数据的精确方式，以及产生理想结果需要的各种处理步骤的顺序。SQL一个非常有用的方面是“闭包（closure）”性质——查询结果以表的形式产生。因此，查询返回的结果行集可以插入到另一张表中，或者用作SQL中查询表达式的一部分，如“子查询”或部分视图定义。

SQL原始定义的另一个重要元素是它包含了定义数据库内容的语法。数据库管理员定义了它的架构（schema）——其他地方叫做表名、列名及数据类型，这是使用所谓的数据定义语言（DDL，Data Definition Language）实现的。实际上它并不是一种独立语言，而是像CREATE、DROP和ALTER一类的SQL命令（动词）集合。SQL在这一方面与大部分SQL语言一样，是数据操纵语言（DML，Data Manipulation Language），该部分的SQL用于查询和更新数据库。DDL由动词SELECT、INSERT、UPDATE和DELETE以及其他SQL动词组成，如GRANT和REVOKE，这些语句都用于指定用户用以访问数据的权利。

很明显，在数据字典（data dictionary）表的行和列中，SQL指定了存储在数据库中用于描述数据库本身内容的元数据（meta-data）。数据字典（或者目录）表也可以使用SQL进行查询，所以可以动态书写应用程序并调整为适合于操作的数据库的外形和内容。

SQL的最初设计意图并非作为一个完整的编程语言。SQL的非过程化面向集合的能力是访问和操纵数据的理想途径，但应用程序的业务逻辑需要更传统的过程化语言。System R开发者创建了嵌入式SQL（Embedded SQL），这是一种允许应用程序员在主编程语言（如COBOL、Fortran和C）中使用SQL语句的“子语言”。SQL语句采用的前缀为“EXEC SQL”，可以嵌入在程序的源代码中，而且可以引用主程序语言中的变量（主变量）。预编译程序代替了嵌入式SQL语句，该预编译程序叫做DBMS特定编程库。

虽然SQL的许多方面遵从Codd关系理论的原始定义，但是为了性能、易用性或易于实现，也做了许多定义上的让步。例如，在Codd语言中，查询结果通常直接由行组成，因为在定义中“投影”操作消除了重复值。在SQL中，除非在查询的SELECT清单中出现了DISTINCT关键字，否则在查询返回的行集合中就可以出现重复值。

此外，作为一种计算机语言，SQL具有自身的怪异之处和缺点。理想的语言应该更直观、更有规则，而且语言元素上的一些限制可以出现在上下文中。SQL的一些批评家发现了SQL在处理过程中存在丢失信息的缺点（即丢失null值），并发现了SQL可以采用多种途径编写不同的语句返回相同查询结果。

Chris Date是致力于普及管理技术和SQL的作家和演说家，他是SQL语言最有声誉的批评家之一。事实上，Date和Codd在处理丢失数据的方法上意见迥然不同。但是，所有的批评和缺陷以及那些已经在用的数据库管理系统，都已证实了SQL无限的价值，而且已经大大成功了，这远远超出了其发明者的期待范围。

SQL在20世纪80年代的商业发展

1977年，Larry Ellison和另外两人建立了关系软件公司（Relational Software Incorporated，RSI），其明确目的是推动世界上第一个商业关系数据库管理系统的市场发展。他们受1970年Codd有关描述关系模型的论文和1974年描述SQL的论文的启发，决定从头开发尽可能与当时IBM研究中心开发的原型相兼容的临时商业产品。Ellison的想法是要在小型计算机上实现SQL系统，而且他正确预见到，除了关系数据库的新颖性之外，与IBM兼容性也会对市场有吸引力。实际上，他们在限制与System R兼容性的承诺是如此彻底，以致于Larry Ellison本人给IBM的Don Chamberlin打电话询问系统使用的错误号。早期ORACLE的演示版通常包括用于

演示IBM System R原型功能的“所得报酬过少的管理者”查询。ORACLE占用空间很小，而且与System R比较，所需资源较少，而System R不仅需要的空间很大，而且需要运行在水冷的大型计算机上。

1979年，RSI推出了第一个商用关系数据库——ORACLE。ORACLE的名字来源于Ellison和他的同事为美国政府工作的一个项目。ORACLE版本1是内部原型，所以商业发行的版本是ORACLE 2。在ORACLE 2中应用SQL是很完善的，因为它包括连接、子查询和视图，也包括处理层的惟一语言扩展，即CONNECT BY从句。在后一个主要的版本中添加了外部连接、日期/时间数据类型和大量嵌入式函数等创新。

系统的首批用户成功地展开了在简单部门中的应用，此时的系统主要是为了决策支持而不是为了满足严格代表事务处理的需求。ORACLE的许多早期用户也对关系模型的功能和使用SQL提供的简易性印象深刻，所以他们通常都忽略了早期ORACLE版本在可靠性上的缺点。RSI在1982年改名为Oracle公司，并开始快速增长，在随后的10年中每年以双倍的速度增长。1989年，Oracle在加利福尼亚的Redwood Shores建立了其总部园区。数据库领域的一个小趣闻是：离Oracle总部最近的机场位于San Carlos，而它竟然也使用了三字母代码——SQL。

自1979年Oracle中引入SQL以来，已经有了许多成功的SQL实现，而且在关系技术上的成功非常明显。令人惊奇的是，IBM从研究关系数据库管理及SQL开发中获益颇费了一些时日。尽管Codd的论文在1970年发表，并且1974年第一次描述了SQL语言，但IBM仍然用了多年时间才开发出其第一个投放市场的SQL产品。直到1981年，IBM才推出了应用于DOS/VSE和VM操作系统上的SQL/DS（使用了原始System R的许多原型代码）。1985年，IBM推出了运行在MVS大型机上的DB2，因为它非常小心地将其定位在需要主要决策支持的部门应用程序上，所以不会与其标志性层次系统IMS竞争。但因为IBM当时在IT行业居于统治地位，而且SQL也日益成为事实上的工业标准，所以这些消息的发布大大加速了SQL和关系系统的发展。

预见到关系数据库巨大潜能的并不只是IBM的研究者，而将该技术引向市场所能获得的巨大商机也并非只有Larry Ellison才知道。从20世纪70年代早期开始，加利福尼亚大学伯克利分校的Michael Stonebreaker教授和他的计算机科学专业的学生就开发了用于当时的Unix操作系统的INGRES关系数据库原型。伯克利小组是根据Codd的思想建立的，但是具有明确的竞争性，至少在INGRES小组和IBM研究所

之间存在学术竞争性。1980年，Stonebreaker建立了一个名为RTI（Relational Technology Incorporated）的公司，将INGRES推向市场。最后，RTI改名为Ingres有限公司。该公司最后被Ask公司收购，随后被Computer Associates收购，该公司目前专攻OpenIngres产品市场。

INGRES实现了称为QUEL的数据访问语言，该语言与SEQUEL相似。一些人认为QUEL是比SQL“更好”的一种语言，因为其武断性限制更少（更具“正交性”），而且具有SQL缺乏的一些功能。但无论其技术优点怎样，QUEL都没有SQL所具有的市场里程碑的意义，因为它被视为一种专有语言。直觉上，SQL可能会成为一种事实上的工业标准，可以可靠地应用于许多产品。结果，为了保留竞争性，大约在1986年，Ingres公司应用了SQL的一个语言子集，处于现存的QUEL接口层之上，但缺少一些主要功能，如空值和子查询等。INGRES的后继版本支持本地的SQL实现。

在关系数据库市场的早期，非关系数据库的坚定维护者视SQL和关系数据库如玩具，从来不会用于重要商业应用。SQL和关系系统的倡导者则赞扬他们的系统产品，而且声称可以克服理论上的性能障碍。

一些人认为，SQL的高层关系接口不能与被应用程序程序员调用的低层操纵接口相竞争。其他人则认为，关系表的物理存储组织和通过索引需要访问的数据值，不会像记录结构中嵌入的指针一样可以直接进行访问。System R开发者宣称SQL语句的自动编译和查询优化可以克服这些问题。当然，经过许多年，SQL系统在关系技术上的改进（随着硬件性能的显著改进）使其甚至适用于对事务处理要求最高的系统。关系数据库系统也能够利用SQL设置的面向对象特性，支持多CPU并行执行SQL语句，提供大数据仓库数据库上复杂查询的高度可升级性能。

在20世纪80年代，许多其他厂家提出了SQL系统。Relational Data Systems，就是后来的Informix公司，于1984年推出了带有SQL接口的同名数据库管理系统。在其他硬件厂家中，数字设备公司（Digital Equipment Corporation，DEC）于1985年发布了Rdb。Rdb应用的不是SQL，而是竞争的关系语言，叫做RDML。RDML在DEC客户中相当流行，但DEC并没有试图让其变得更流行，所以没有使其标准化。1988年，DEC推出了Rdb 5，承认需要遵守工业标准，具有完全内在的SQL应用特性。1994年，数字设备公司将Rdb卖给了Oracle公司，后者目前仍然在销售和支持该产品。

1985年，Teradata并行查询机的引入是SQL演变上的一个重要里程碑。Teradata系统使用了由Intel 8086处理器与私有树网络连接组成的特殊用途硬件平台，这是可以自动并行执行SQL语句的第一个商业数据库产品。但是，Teradata的SQL方言有很大局限，最初缺乏对视图和参照完整性的支持。Teradata系统被定位为面向数据仓库应用程序的查询处理需要，而且通常认为不能应用于事务处理系统。

从Ingres出来的核心人物Britton-Lee也设计并销售了“关系数据库机”，但该产品只取得了有限的市场成功，而且很快被Teradata收购，并最终从市场上消失了。继而NCR又收购了Teradata（随后NCR也被AT&T收购）。目前，NCR/Teradata已经放弃了采用特殊硬件的方法，现在都运行在使用Windows NT和Unix操作系统的多用途平台上。Teradata在数据仓库市场上已经非常成功，尤其在它们的数据仓库上已经具有大量追随者和大量数据。Teradata和Britton-Lee都发现很难与硬件和软件设计变革并驾齐驱，而且很难用需要特殊硬件的私有方法实现常用硬件系统。

另一个著名的里程碑是，1987年Tandem引入了不停止的SQL（NonStop SQL）。为了得到优秀的事务处理性能和高度实用性，NonStop SQL进行了优化。Tandem通过模拟简单银行事务的工作量支持其运行性能。这个基准的派生最终变成了事务处理性能理事会（Transaction Processing Performance Council，TPC）开发的第一个工业标准基准。NonStop SQL的引入打破了关系系统不能提供高端事务处理应用程序所需性能的神话。

Sybase是SQL市场上一个非常重要但相对姗姗来迟的迟到者，Sybase公司于1987年才推出了第一个版本的产品。微软购买了Sybase产品的源代码并随后于1993年推出了Windows NT下的SQL Server。

Sybase被设计为客户/服务器结构，应用程序运行在PC或工作站上，并通过网络访问数据库服务器。在并行操作下，我们看到了SQL语言和接口高层特性出人意料的好处。例如，仅通过调用一些网络消息，单条SQL语句就能迭代大量的行集，或者把表连接在一起。通常，采用低层导航式接口进行这样的操作会带来过多的网络流通量。

Sybase是第一个可编程的SQL数据库系统，而且也具有显著的市场影响。采用Sybase，DBA或应用程序开发者可以实现业务逻辑，并用触发器和Transact-SQL（Sybase公司专有的过程化语言）写的存储过程加强数据完整性规则。DBA和应用

程序开发者可以书写包含嵌入式SQL语句的程序提取，或更新数据库数据来执行完整的业务事务。触发器可以与数据库表相关，并可以在INSERT、UPDATE或DELETE操作之后执行，使事务生效、审核并执行其他转换。这一方法减少了网络流通量，因为整个业务事务可以用存储过程执行，并进行有效的网络调用。利用存储在数据库中并在数据库服务器上执行的存储过程，应用程序不需要与服务器上被访问的每一条记录进行通信，也不需要与业务事务中需要完成的每一条SQL语句进行通信。

可编程特性的另一个重要好处是，数据库服务器可以保护数据库的完整性，从而防止恶意的个别用户和应用程序通过网络对数据库进行攻击。虽然如参照完整性这样的基本关系型完整性的规则通常定义得很好，但是作为数据库架构的一部分，数据库触发器积极加强了需要过程定义的任意业务规则。通过数据库中的中心化业务逻辑，就不需要在访问数据库的每一个应用程序上编码，从而有效地避免了冗余和错误，并给终端用户直接访问数据提供了灵活性。

Oracle公司对自己的一部分产品使用了Ada编程语言作为PL/SQL模型，这是其专有的过程语言。像Ada一样，PL/SQL中包含了如异常处理和可以应用于开发的可用的、大型复杂系统的参数类型定义等语言特性。将过程语言添加到SQL标准中，从而最终组成了PL/SQL。PL/SQL最早出现在1988年的客户端中（在Oracle的SQL*Forms中），随后在1992年的Oracle7中使用了触发器和数据库中执行的存储过程。

SQL的进化：20世纪90年代和以后

如果20世纪70年代是SQL发明创造的十年，而20世纪80年代是SQL商业化的十年，那么20世纪90年代就是SQL进化的十年。在这一时期，SQL产品的各个厂家为了支持新的、要求高的应用程序而竞相在市场上推出新的特性。近年来，商业SQL产品和SQL标准都已被扩展，新特性支持面向对象的编程语言和多媒体数据、集成了Java和XML，而且满足数据仓库应用的需要。SQL已经成为一种充满活力的语言，并随市场的发展而不断发展。

20世纪90年代早期，面向对象的编程变成了商业应用程序开发的流行模式，因为程序员发现他们可以使用对象书写更快、更可靠的复杂应用程序。面向对象的语言

允许程序员定义描述数据结构及其行为的类型（或类）。类型可以是复杂的结构，也可以包括过程（方法）作为定义的一部分。类型可以来源于其他类型，并继承父类型的属性。对象范型的基本概念是每个对象都有截然不同的标识符，而且一个对象可以通过其对象标识符直接引用另一个对象。

尽管直到20世纪90年代后期对象技术才影响了SQL的方向，但是对象编程的方法并非一种新方法，早在20世纪60年代，在Simula和Smalltalk编程语言中就有了起源。目前已经开发了许多面向对象编程语言，但是第一种吸引了广泛商业用途的语言是C++，这是C语言的向上兼容的扩展。C++的成功部分归功于与C程序的互用性和C程序员不需要学习一门全新语言这一事实。

对象模型代表了关系模型完全相反的一面，具有简单的数据结构（表、行和列）和数据访问的非导航方法。从根本上说，关系模型依靠基于值寻址，这里的行用存储在（主）键列上的数据值进行定位。在SQL数据库中，通过比较列的值，连接操作通过比较表的多列值匹配多张表中的行，这一方法与直接从一个对象指向另一对象的对象引用相反。

因为在类型系统中存在强烈的差别，所以在面向对象的编程语言和SQL之间已经形成了所谓的“阻抗失配”。一些人认为SQL和关系数据库模型是绝对的，而且只有数据库系统设计使编程语言对象无缝连续才可以满足现代应用程序的需要。其他人已经开发了与应用程序定义的类型相匹配的简单数据类型、数据库结构的产品。

在最近几年，如Oracle、IBM和Informix这样的关系数据库厂家已经在SQL语言中添加了对象的性能。这些对象关系的产品以及最近的SQL标准，允许类型定义与其他对象语言相似，但又与它们不同。这些扩展的SQL类型在每一列上有多个值，也可以用方法或函数作为定义的一部分，而且可以从更高层的类型上继承属性，并可以包含值是其他一些特定类型的对象实例引用（指针）的属性。这种加强的扩展关系模型给数据库设计者提供了与现实世界更直接的模仿能力，而且使系统能将数据库类型与如C++和Java这样的面向对象的编程语言类型相匹配。

扩展SQL方法的主要目标是保留关系模型的优势，包括在对象集（通常存储在表中）上执行非过程查询的能力。新的面向对象应用程序可以与现存的关系应用程序一同存在，而且数据库系统可以通过名为对象视图（object view）的新特性与传统关系数据中的对象合成。面向对象的SQL扩展以向上兼容的方式添加，就像C++是从C

开发而来一样。尽管人们在定义更纯粹的面向对象新数据库语言上做出了不少努力，但是，具有这些新对象能力的SQL至今仍成功地捍卫了其“通用数据语言”的地位。

面向对象的数据库厂家已经使用了扩展SQL，来支持以前在自己的产品中非常难以管理的关系数据库数据类型，包括文本、图像和声音数据。这些产品的用户也可以使用扩展SQL定义应用程序指定的数据类型和索引类型。

Internet技术的出现也在SQL中产生了新的需求。Java是简洁的面向对象语言，尤其适用于开发为Internet设计的应用程序。近年来，SQL发展迅速，形成了不断增长的Java开发者团体。数据库厂家很快就同意并引入了可以在Java上集成SQL的产品接口。JDBC调用接口允许程序发送SQL语句到数据库服务器上执行。SQLJ规范允许SQL以与其他主要编程语言（如COBOL和Fortran）类似的方式嵌入在Java程序中。例如，Oracle8*i*在数据库服务器上支持执行JDBC和SQLJ程序。因此，SQLJ和JDBC程序可以在客户端、应用服务器或者数据库服务器本身上作为存储过程、数据库触发器或对象类型的方法执行。SQL语言将继续发展以便同Java更好集成，例如，通过支持使用Java类作为列数据类型的定义。

XML代表可扩展标记语言，它是另一种影响了SQL进化的Internet技术。因为XML以自描述的形式组成数据，所以尤其适用于在独立开发的应用程序和企业之间交换信息。例如，电子商务应用程序可以使用XML交换如订单、支付和客户信息一类的数据。自然地，因为大部分业务应用程序使用关系数据库，所以SQL数据和XML数据的共存就变得更重要。

就像SQL已经发展为接纳Java和其对象模型一样，它也已经开始扩展应用XML数据。当今SQL语言的丰富扩展对象非常适合于支持简便地表现或匹配XML数据，将数据库操纵和查询带入静态XML数据结构中。如Oracle这样的厂家已经展开了应用能力的攻势，可以映射XML数据结构为数据库数据，并从SQL查询下产生XML格式的结果。SQL与基于XML查询语言的集成也是厂家、标准和学术团体积极开发的一个区域。

尽管SQL在许多领域中功能非常强大，不管其在数据仓库应用程序上有多重要，但它从来没有提供强有力的分析任务支持。许多基础业务智能计算也需要标准SQL之外的扩展编程，通常具有明显的性能挑战。

虽然一些为解决这些需求而设计的专有SQL扩展编程已经存在于一些特殊的产品中，但只有最近厂家们才同意扩展标准SQL以满足需要。

在SQL标准和许多SQL产品中，人们已经在GROUP BY从句中增加了CUBE和ROLLUP扩展。这些操作符在分组列的值之间填充合计和小计，而且应用于产生交叉报表的聚集。数据仓库和业务智能用户长期坚持需要SQL支持队列和移动平均值，并执行周期间的比较。但是，像“显示每一个区域中最上10个和最下10个销售人员”或“计算股票价格13周移动的平均值”这样的查询，在标准SQL中已经变得非常难以实现或者不可能表达。最近，Oracle和IBM已经联合设计并提交了解决这些需求的新能力标准。Oracle8*i* 第2版介绍了一系列强有力的分析功能，支持队列、移动平均值、在聚集不同层之间比较值、以及周期与周期之间的比较。

SQL语言标准

因为IBM在20世纪80年代的控制，所以SQL注定要成为数据库管理上的重要语言。Oracle紧密遵循IBM有关SQL上的定义，这是首批这样做的许多厂家之一，从而使其成为事实上的标准。但是，如果没有国家和国际标准组织努力开发语言公用规范，SQL也不会成为这样的通用数据访问语言。

如果说SQL语言是揭开了访问世界信息之锁的Rosetta石头，那么SQL标准文档就是Rosetta石头自身的一些内容，而不是厂家的文档，SQL标准提供SQL语言的惟一形式并完整定义了语法和语义。

标准进程的历史非常有趣。20世纪50年代，美国的国防部建立了数据系统语言协会（Conference on Data Systems Languages，CODASYL），开发业务应用程序上的标准化计算机编程语言。CODASYL开发了COBOL语言，而且是数据基本任务组（Data Base Task Group，DBTG）的父组织，并在1971年出版了一系列规范，COBOL程序通过这些规范可以操纵应用了基于指针的“网络模型”数据库。这些起源促进了SQL语言正式标准的产生。

信息技术标准国家委员会（the National Committee for Information Technology Standards，缩写为NCITS）特许广为人知的ANSI SQL委员会（也就是数据库上的H2技术委员会）开发数据库语言的美国国家标准（American National Standards），并代表了美国在相关国际标准上的行为。ANSI委员会始建于1978年，

正式建立了CODASYL的标准化规范。虽然它负责网络数据库的标准，但是，该委员会从1982年也开始致力于关系数据库标准。

尽管ANSI SQL委员会将IBM的SQL正式标准应用于关系数据库上，但其最初的努力则是致力于解决SQL上许多公认的不足。委员会的工作成果令人满意，由此产生了“改进的”语言（他们将其命名为RDL，代表关系数据库语言［Relational Database Language］）。但是，RDL与DB2代表的事实标准SQL差别很大。1984年，因为不得不重新考虑商业上可以使用的新数据库语言应用分支，该委员会只好放弃了以前的努力，重新安排最初的IBM SQL草案作为ANSI和ISO标准的基础。

第一个正式的SQL标准于1986年出版，大约有100页。该标准定义了代表当时最重要的SQL实现通用特性的语言基干，包括许多严格的限制。该文档定义了基本SQL语言，包括创建表和视图的能力，但不包括删除和修改表及视图的能力，也不包括授权或收回访问权限的能力。

从关系数据库倡导者的观点来看，SQL-86缺乏参照完整性能力是一个引人注目的问题。因为受到了激烈的批评，所以SQL标准委员会快速推出了“完整性加强特性（Integrity Enhancement Feature）”规范来弥补这一缺点。这一特性包括定义主键和外部键作为数据库架构一部分的能力，也要求插入、更新和删除不会导致外部键中的行与另一张表中的主键不匹配。这些基本特性意味着，非常基础的数据完整性规则可以通过系统应用标准来自动加强。完整性加强特性被并入到1989年的SQL标准修订版中，该标准也包括在COBOL、Fortran和C中嵌入SQL的规范。

下一个标准在1992年采纳，也称为SQL-92。SQL-92增加了SQL语言上无数的能力，包括外部连接、日期－时间和其他数据类型、标准错误报告、一系列的标准分类表和动态架构操纵（DROP、ALTER、GRANT和REVOKE），而且包括主程序执行编译时未定义的SQL语句（动态SQL语句）的能力。SQL-92的其他新特性包括级联更新和删除引用行为、事务连贯层、滚动游标和延迟约束检查。该标准大约有600页，而且被分成3层：

- 入口层（Entry）SQL-92只包含SQL-89上的特性。
- 中间层（Intermediate）SQL-92增加了大约一半的新特性。
- 完全层（Full）SQL-92代表了完整标准。

当在商业数据库产品中实现时，SQL-86和SQL-89标准都定义了SQL语言子集。与之对照，在定义时，SQL-92标准期待在SQL产品上进行开发，而且仍然充当软件开发的向导。当厂家在应用其定义的新特性时，他们通常采用该规范，但SQL-92也包含了厂家还没有应用到的特性。

当前的标准是SQL-1999，于1999年7月出版，所有文档大约有2000页。实际上该标准的建立工作开始于1990年，因为SQL委员将许多特性延迟到SQL-92的下一标准（在开发中称为SQL3）中。SQL3开发的周期很长，主要是因为其范围广，尤其是在SQL中可包含对象的能力。在得到一致同意之前，大家调解了许多意见和错误。许多争论围绕抽象数据类型（abstract data types，ADT）定义为可引用的“对象ADT”与那些定义为嵌入的“值ADT”之间的精细区别，也有许多建议被考虑、采纳并代替了各种SQL草稿。最后，委员会成员通过组合统一属性的抽象类型单模型解决了它们之间的差别。

在新SQL标准中合并了面向对象的扩展功能集，组成了一个与Java很相似的对象模型，从而简化了将这两种语言放在一起使用的任务。SQL-1999增加了有行为（方法）和封装内部结构（包括数组和命名行类型）的用户定义类型的应用。ADT定义可以追溯到更常用的类型（单个继承）。SQL-1999支持带有编译时检查和动态方法分派（多态）的强类型。对象数据类型的实例可以存储在普通表的列中。但是，存储为行（在类型表中）的这些类型的每个实例，都具有可以从SQL语句中引用的永久对象ID，而且可以永久地存储为另一对象的属性。

SQL的核心功能（或者叫SQL/Foundation）除了包含对象功能、一些预期性商业应用和在大量商业产品中长期出现的其他特性之外，还包含了许多其他特性。SQL-1999中包括的新特性有：

- 用户定义过程和函数，包括扩展性定义
- 在INSERT、UPDATE或DELETE之前或之后触发的行层和语句层数据库触发器
- 布尔数据类型和大对象（二进制和字符LOB）
- 支持字符集、转换和排序规则（排序）
- 新的WHERE谓词（可以用于所有、一些或相似）

- 可更新视图
- 定义安全性轮廓的角色
- 保护性断点，从而可以部分完成事务的回退
- 递归查询，允许处理大量素材

除了SQL/Foundation的核心功能，SQL委员会还开发了SQL规范的其他部分，包括现在作为SQL一部分的应用对象模型。简要来说，这些功能包括SQL-1999的下列部分：

- SQL/PSM（persistent stored modules，持久的存储模块）：循环、分支、过程调用（procedure invocation）和动态异常处理的过程语言能力
- SQL/OLB（对象语言绑定）定义了方法使Java语言与SQL语言和访问SQL数据接口
- SQL/MED（扩展数据管理）指定接口允许SQL访问存储在操作系统或非SQL源上的数据
- SQL/CLI（调用层接口）指定SQL和数据库服务的应用程序编程接口
- SQL/Temporal定义了支持数据库内容随时间改变的视图特性

同时SQL委员会也从标准上出发并立足于新的ADT能力，开发了SQL/MM（多媒体数据）规范，定义了管理文本、声音和图像数据的功能。

显然，SQL不再是定义、访问和管理包含具有单值列行表的单一语言。根据SQL-1999，起源于正式关系模型数学的该语言已经超出并背离了原始的、注重实效的模型。像CODASYL在SQL-1999中定义的网络数据模型，现在支持数组、分组、重复分组和嵌套重复分组的复杂数据记录结构。根据这样的功能和复杂性，数据库设计处理趋向于超出了Ted Codd定义的数据库标准化原理。数据库设计者将需要高度理解应用程序处理的需求，也需要掌握数据从属性的知识，从而在SQL-1999中尽可能多地从数据结构和层次上受益。

SQL语言的标准是庞大，而且完全详细地理解语言的每一方面大概超出了人们的理解范围。目前，没有一种商业产品实现了整个标准并给定标准的大小，它们似乎永

远也不会实现这一点。此外，不同产品实现了不同集合的特性，而且没有遵从该标准的任何认证测试。那么，该规范的好处是什么呢？

一般而言，在应用新特性时，厂家确实以查看标准规范作为设计向导，所以，从一个设备到另一个设备上的应用产品的系统性和应用程序的便捷性，至少必须适用于大部分主流产品。此外，各个厂家在标准化进程中就设计SQL的特性上进行了频繁的协作，以免将来的产品在新性能上出现分歧。而且，通过对比产品的特性与SQL标准，选择SQL产品的公司可以评估厂家的兼容性和技术领先地位。具有SQL实践经验的DBA或开发者都可以利用来自其他产品的知识，因为SQL的各种方言不同于另一种SQL语言，这有点像地区口音与整个语言之间的差别一样。

SQL：成功的历史

关系数据库技术和SQL语言的空前成功，是IT行业取得伟大成功的因素之一。一系列因素促进了这一成功。

作为非过程数据访问语言的SQL有着坚实基础和关系型理论，这已被证实是其强大的起点，尽管SQL的发展已经超越了这一起点。硬件和软件技术的进步已经克服了最初关心的关系数据库的性能问题。已经证实的SQL接口高层特性特别适用于数据仓库并行处理、客户/服务器以及Internet环境需要的可编程数据库扩展性。SQL和关系数据库已经在较广范围的客户应用程序中成为一个重要的成功因素，这些应用程序包括了最简单的部门应用程序到任务关键的事务处理系统，以及各种支持业务智能的数据仓库。为了响应市场的竞争压力，数据库厂商已经对自己的SQL实现进行了持续改革和扩展，同时迫于市场压力而不得不顺从标准并为标准作出贡献。标准处理已经提供了SQL的合法性和将来的发展蓝图。目前，SQL已经证实了其适应市场需要的能力，例如，在数据仓库和新的技术如对象、Java和XML上得到了有意义的提高。

如果有人要查看IT行业的分类广告，就应该对正确理解SQL的工作投以欣赏的目光。了解SQL历史知识和将来的一些意义，可以帮助熟悉语言和技术的人更加欣赏其价值。

——Ken Jacobs，产品策略副总裁，服务器技术部，
Oracle公司

前言

关系数据库系统的根源要追溯到1970年，那时E.F.Codd博士发表了标题为“A Relational Model of Data for Large Shared Data Banks”的论文（注1），当年6月刊登在由美国计算机学会（the Association of Computer Machinery，ACM）主办的《Communciations of the ACM》期刊上。该论文永久地改变了计算世界。Codd建议的关系数据库管理系统模型(relational database management system,RDBMS)最后变成了关系数据库的定义标准，而关系数据库则成了当今使用的主流数据库。

Codd的模型需要访问数据库的新语言，所以IBM开发了结构化的英文查询语言，并在1974年首次命名为SEQUEL。遗憾的是，SEQUEL商标已经存在，所以从名字中删除了“英文（English）”字样，将其重新命名为SQL，或者叫结构化查询语言（Structured Query Language）（但是，发音仍然为“sequel”）。

1979年，Larry Ellison和Bod Miner建立了关系软件公司。RSI开始开发Codd模型和SQL语言首批商业化的实现产品，而且公司推出了世界上最早的关系数据库Oracle V.2。不久，RSI改名Oracle系统，并随后改为Oracle公司。

不久后，SQL就变得非常流行，但每一个厂家都以不同方式和不同功能对其进行实现。最后，ANSI和ISO开发了相应的标准，最新的标准是:

注1：E.F.Codd的“A Relational Model of Data for Large Shared Data Banks”，见1970年6月13日的《Communciations of the ACM》，第377-387页。

- ANSI X3.135-1992，“Database Language SQL”
- ISO/IEC 9075:1992，“Database Language SQL”

目前，实际上所有厂家的数据库系统都声称已经实现了关系模型，而且全都使用SQL作为访问语言。SQL-92定义了4种遵从级别：Entry、Transitional、Intermediate和Full。一个遵守SQL的应用程序必须至少支持Entry SQL。Oracle8完全支持Entry SQL，而且有许多特性符合Transitional、Intermediate和Full SQL。

本书的读者

本书是为使用SQL的Oracle开发者和数据库管理员（DBA）所写的。该书设计为参考手册；提供的材料尽可能准确、可靠。因为我假设读者已经在某种程度上熟悉了关系的概念和SQL语言的基础，所以试图提供给大家一些需要的信息，可以在一定位置和格式下允许大家根据需要对其进行快速查阅。如果你要查找有关Oracle实现SQL的清晰、精确的信息，利用大量的摘要表并快速参考语法和用途，即可到达正确的位置。但是，如果你试图学习SQL，或想要一本从头到尾进行指导的指南，则可能需要从一本介绍性的书开始。但是，坚持使用本书，你以后将发现其非常有用。

我当然不想阻止你购买和使用本书，但是我想让你知道，我在书中使用了很多正在快速发展的材料，只能做到尽可能精确。

Oracle版本

本书是用Oracle 8.1.6写的，所有信息都精确适用于该版本。大部分材料也适用于7.3和8.0版，但8.1版（Oracle 8*i*）中向SQL加入了新的功能。

从Oracle 8.0版本开始，已经可以使用一种叫做对象选项（Object Option）的新选项。该对象在Oracle 8.1.6（Oracle8*i*推出的第2版）中的有效实施包括在所有Oracle服务器产品基本版本中。在我的经验中，这个选项还没有得到广泛应用，但是，它非常复杂，而且在SQL语言上添加了大量有意义的语法。事实上，将需要一整本书描述Oracle的对象数据库。因此，我决定从本书的印刷版本中省略对象的特性。如

果你确实要使用对象选项，可以在O'Reilly网址*www.Oreilly.com/catalog/orsqlter*上和Oracle的《SQL Reference Manual》中找到可用的完整语法。如果你像当前的大部分Oracle用户那样不使用对象特性，那么我认为你会发现将更容易使用本书而不必使用特殊的材料。一旦该选项得到广泛使用，我将在本书的随后版本中包括进来。希望能让我知道你对该决定的想法。

本书内容

本书包含下列章节:

第一章，“SQL元素”，提供了SQL语言的通用元素轮廓及需求，包括：结构、命名标准、各种数据类型字符和关系操作符等等。

第二章，“数据定义语句”，代表了用于定义或修改数据库对象的每条SQL语句。在这一章中我已经根据结构语法、需要的语法元素定义简要解释了每条语句所完成的功能，并用一个简短的例子说明了其用法。

第三章，“数据操纵和控制语句”，与第二章的结构非常相似，但列出了数据库中用于操纵数据和控制会话的SQL语句。

第四章，“公共SQL元素”，代表了可以应用于多种类型SQL语句的SQL语言部分。本章中显示的从句引用了第二章和第三章提供的语句。

第五章，“SQL函数”，摘要提供了Oracle中可以使用的SQL函数的丰富列表。为了方便，对这些函数进行分组，例如，所有字符串操纵函数放在一起提供。

第六章，“SQL*Plus”，代表了SQL语言的Oracle标准接口。SQL是Oracle的一个完整部分，所以没有程序员不使用它。本章提供了使用该产品的摘要信息，我希望你能从中看出SQL*Plus的功能（注2）。

注2: 在这一章中提供的信息只是SQL*Plus的一部分内容。要得到有关SQLPlus的更多信息，请参考Jonathan Gennick的《Oracle SQL*Plus:The Definitive Guide》(O'Reilly & Associate, 1999)和《Oracle SQL*Plus Pocket Reference》(O'Reilly & Associate, 2000)。

第七章，“PL/SQL”，提供了结构的简明定义和PL/SQL基本元素的语法，也包括创建和使用Oracle过程、函数、包和触发器（注3）。

第八章，“SQL语句调试”，解释Oracle内核实际怎样执行本书中提供的SQL语句，并提供怎样更好地执行SQL语句和程序的宝贵信息。

附录一，“SQL资源”，提供了与SQL相关的书、其他出版社、组织和可以使用的网站清单。

排版约定

本书使用下列排版约定：

等宽字体（Constant width）

用于表示语法图表和输出命令。

等宽黑体（**constant width bold**）

用于代表例子中用户交互的用户输入。

首字母大写的斜体

表示第四章中文档的SQL通用元素。这些元素通常用“Clause”结束，如*Storage_Clause*。

小写字母斜体

表示该项目在SQL语句中要用实际值代替。

大写字母

表示SQL语法的关键字或组件。

大括号{}

表示该项目需要在大括号中列表提供。列表中的项目用垂直线(|)分离。注意，该列表可以具有很强的扩展性，而且单个项目可以包括许多关键字。

注3： PL/SQL是一个里程碑。要得到有关PL/SQL的更多信息，参见Steven Feuerstein和Bill Pribye的《Oracle PL/SQL Programming》第二版（O'Reilly & Associate，1998）和Feuerstein系列的其他PL/SQL书。

省略号（...）

表示以前的关键字或关键字组可以根据需要进行重复。

中括号[]

表示封装在[]中的该项目可选。多个选项采用垂直线（|）分离。

下划线

表示带下划线的关键字或值是默认值。

列（column）

表示表中的列名。

列表（column_list）

表示表中的一个或多个列，用逗号分离。

文件名

文件的有效操作系统名，通常放在单引号中。

整数（integer）

整数，如0、1、2等。

架构（schema）

数据库中对象集合名，也称为用户、用户名或所有者。

字符串（string）

一个或多个放在单引号中的字符，如‘this is a string’。

表（table）

表名。

表空间（tablespace）

数据库中的表空间名。

建议与评论

本书的内容都经过测试，尽管我们做了最大的努力，但错误和疏忽仍然是在所难免的。如果你发现有什么错误，或者是对将来的版本有什么建议，请通过下面的地址告诉我们:

美国:

O'Reilly & Associates,Inc.
101 Morris Street
Sebastopol, CA 95472

中国:

100080 北京市海淀区知春路 49 号希格玛公寓 B 座 809 室
奥莱理软件（北京）有限公司

询问技术问题或对本书的评论，请发电子邮件到:

info@mail.oreilly.com.cn

最后，您可以在 WWW 上找到我们:

http://www.oreilly.com
http://www.oreilly.com.cn

致谢

我应该感谢为本书创作作出贡献的许多人。

本书始于与O'Reilly的编辑Deborah Russell的一次交谈，她觉得实际需要一本像这样的书，并让我来完成这件事情。我想感谢她提供了内容、结构和风格上宝贵的意见。在本书出版时，她甚至比我还要高兴。

我碰到的第一个 Oracle 员工是 Ken Jacobs，那时他负责技术销售支持联络。在随后的 15 年之多中，我已经多次与 Ken 接触，他以突出的技术和从容能力而在 Oracle 团体中著称。Ken 是一个技术天才，一个优秀的深水潜水员，而且是一个真正伟大的人。我永远要感谢 Ken!

对 Jonathan Gennick，仅靠言语感谢还远远不够。他不仅提供了详细的技术评论和编辑帮助，而且提供了大量的资料，并把我从灾难的边缘挽救回来！对我而言，足以认识到支持的重要性已达到了新的高度。

特别要感谢John Beresniewicz、Steven Feuerstein和Brian Laskey，也要感谢Oracle公司的Richard Sarwal和Sandy Venning，他们利用自己的业余时间对本书进行审校。

我还要重重感谢我的爱人和支持者——Suzanne，她曾经一度容忍我深夜“盘踞”在地下室中，而未抱怨我把相当部分的假期花在笔记本电脑上。

我还要感谢我的同事Rhodia，包括Brian McMahon、Dave Flood、Bin Pan、Deb Irwin、Joaquin Lucero、Paul Mars、Nick Palmer和Frank Foley，因为他们完成了早期通读“草稿”的工作，而且总是很支持我。

如果没有得到O'Reilly & Associates的大量艰苦工作和支持，本书当然不可能问世：Darren Kelly负责管理作品进程；Ann Schirmer负责本书的版式；Maureen Dempsey负责校对本书；Ellie Volckhausen负责制作封面；Rob Romano负责开发图表；David Chu为助理编辑，帮助准备本书的产品；还有James Carter、Deborah Smith和Nancy William，他们提供了产品支持。

第一章

SQL 元素

进行SQL开发是为了更容易地访问关系数据库，所以，SQL能够执行下面几种操作：

- 从数据库中查询数据
- 向数据库中插入数据
- 从数据库中删除数据
- 创建和操纵数据库对象
- 控制对数据库的访问

严格地说，SQL 根本不是一种语言，而是对Oracle数据库传输指令的一种方式。它与传统的编程语言在以下几个主要方面有所不同：

- SQL 提供自动定位数据的能力
- SQL 在数据集合上操作，而不是对独立的数据元素操作
- SQL 是声明的，而不是过程化的，并且不提供过程控制
- SQL 设计在逻辑层完成；这不需要处理实现的细节

简单地说，当在SQL中设计时，你告诉Oracle自己想要干什么，而不是怎样做。但是，这种方法利弊并存。考虑下面的 SQL 语句：

```
SELECT ename, deptno, sal, comm.
FROM scott.emp
WHERE hiredate > '01-JAN-00';
```

这个简单的SQL语句告诉数据库显示2000年1月1日以后雇佣的职工名字(ename)、部门号（deptno）、薪水（sal）和佣金（comm）。在“老式的”过程化编程语言中，这可能需要几百行代码，但是，在SQL中只需要3行代码。Oracle在检索数据时并不总是如此灵巧。尽管Oracle内部的“查询优化器”有了稳步提高，但仍然有许多方法可以改进SQL的性能，第八章在讨论SQL语句的调试时将展示这一主题。

注意： 有人将缺乏过程化控制视为SQL的缺点，因此，Oracle公司开发了PL/SQL（Procedural Language/SQL，即过程化编程语言/SQL），这一内容将在第七章讨论。

SQL语句也称为SQL命令，由以下几部分组成：

关键字

带有特定Oralce操作意义的保留字。

变量

它是可以被动态文本或数字变量替换的数据元素。SQL有许多对象名，如变量、表或视图。

文字

常量数据，包括文本字符串和数字。

操作符

操作一个或多个变量或文字的符号或单词。

词汇习惯

SQL语句由命令、变量和操作符组成，这些内容在本章和第二章将进行详细描述。SQL语句由以下元素构建：

- 字母A到Z（或者数据库字符集中的相等物组成）
- 数字0到9

- 空格
- 下面的特殊字符：+ - * = ? ! @ () _ . , <> | $ #

注意：Oracle 特别不推荐使用 # 和 $。

其他字符，如 & 也用在 SQL 语句中，但是，如果你使用 SQL*Plus 工具，则可以被 SQL*Plus 中途截取或解释。要了解 SQL*Plus 的更多信息，请看第六章。

在任何可以放置单个空格的地方，SQL 语句都可以包含一个或更多的下列项目：

- 制表符
- 回车
- 多空格
- 注释

注意：SQL 语句的某种组件（如变量名和字符串）可以包含其他字符，只要把它放在双引号中即可。

例如，对 Oracle 而言，下面的两个 SELECT 语句在性能上是完全一样的，而且都返回相同的结果集：

```
SELECT ename,empno,sal FROM scott.emp WHERE sal>500;

SELECT ename,
       empno, sal
FROM   scott.emp
WHERE  sal > 500;
```

SQL 通常不区分大小写，所以，除了将大小写放在引号中具有文字意义以外，大小写没有意义。

警告：要明白放在双引号中的变量名区分大小写。如果你使用微软 Access 数据库访问 Oracle 数据库，则该事实尤其重要，因为它将使用小写名字创建对象。

SQL 命名

SQL中的大部分命名需求实际上是Oracle数据库的需求；Oracle数据库中可接受的架构对象（schema objects，在第 3 节中定义）的命名都可在 SQL 中接受，反之亦然。下列规则应用于 Oracle 中的架构对象：

- 可以由 1-30 个字母字符组成。
- 必需用字母开始。
- 可以包括下划线（_）。
- 虽然 Oracle 不鼓励使用美元（$）或英镑（#）字符，但也可以包括这些字符。
- 不能是保留字。
- 不可使用 SQL 命令名。

注意：如果放在双引号中，则可以在开始时包含任何字符。

架构对象

架构对象是存储在数据库中由用户所有的数据或其他对象的逻辑集合。下面的对象类型可以被认为是架构对象：

- 簇
- 数据库链接
- 数据库触发器
- 维度
- 扩展过程库
- 索引组织表
- 索引
- 索引类型

- 物化的视图/快照
- 嵌套的表类型
- 对象类型
- 操作符
- 包
- 序列
- 存储过程
- 同义词
- 表
- 可变数组类型
- 视图
- 数据库链接

常用语法

通常，大家在SQL语句中引用架构对象时使用下列语法:

```
schema.object_name.object_part@dblink
```

这些语法元素的意义如下:

架构（schema）

拥有对象的架构名。在Oracle中，架构与用户名一一对应；如果在引用中省略架构对象，那么默认使用当前登录用户名。

对象名（object_name）

被引用的对象名，如表。

对象部分（object_part）

对具有部分对象名的架构对象，就是部分对象名，如表的列。

数据库链接（dblink）

一个引用远程数据库的数据库连接的名称。

这里显示的语法，在一定的周期之后紧随架构名，后面的对象名（如scott.emp）通常指点（.）。通常，如果*schema.*部分的名称被省略，那么将默认使用连接到数据库架构的当前用户。

例如，下面从表中查询数据的SQL语句，就是架构scott中的架构对象，被命名为emp。该架构在远端数据库中，而且被数据库链接test引用：

```
SELECT ename, empno, sal
FROM scott.emp@test
WHERE sal > 500;
```

分区语法

当引用一个已分区表中的特定分区或子分区时，使用下列语法：

```
schema.table_name {PARTITION (partition) |
                   SUBPARTITION (subpartition)
                                          }
```

该语法元素具有以下意义：

架构（schema）

拥有对象的架构名。在Oracle中，架构对应于一对一的用户名，而且，如果在引用架构对象中省略架构，那么默认使用当前登录用户名。

表名（table_name）

被引用的表的名字。

分区（partition）

表的分区名。

子分区（subpartition）

表的子分区名。

该结构被称为分区扩展表名（partition-extended table name）。分区扩展表名可以没有与之相关的数据库链接。因此，如果你想访问远端数据库上的该对象，就必须创建可以使用以前描述的常用架构对象语法访问的视图。

数据类型

Oracle 在数据库中以 3 种基本数据类型存储数据：字符、数字和日期。字符和数字具有许多相关的直接数据类型，将在本节中描述。

字符数据

字符数据是指不能成为算术操作的直接目标的单字节或多字节的数据字符串。Oracle（和 SQL）支持许多类型的字符数据，下面列出了其使用语法：

CHAR [(*length*)]

定长字符数据，最长可以到 2000 字节。*length* 指定了被存储字符的最大长度。

VARCHAR2 [(*length*)]

可变长度的字符数据，最长可以到 4000 字节。*length* 指定了被存储的字符的最大长度。

NCHAR [(*length*)]

固定长度的字符数据，由国际字符语言（National Character Language，NLS）字符集字符组成。因为一个字符可能需要不止一个字节，所以最长可以到 2000 字节（允许少于 2000 字节）。*length* 指定了被存储字符的最大长度。

NVARCHAR2 [(*length*)]

NLS 字符集字符组成可变长度字符数据。因为字符可能需要不止一个字节，所以最长可以到 4000 字节（允许少于 4000 字节）。*length* 指定了被存储字符的最大长度。

LONG

可变长度的字符数据，最大长度可以为 2GB。

RAW

原始的二进制数据，最大长度可以为 2000 字节。在不同字符集的系统之间移动数据时，不能用 Oracle 转换 RAW 数据。

LONG RAW

原始的二进制数据，最大长度可以为2GB。在不同字符集的系统之间移动数据时，不能用Oracle转换LONG RAW数据。

可以认为下面的字符数据类型与ANSI SQL兼容：

```
CHARACTER
CHARACTER VARYING
CHAR VARYING
NATIONAL CHARACTER
NATIONAL CHAR
NATIONAL CHARACTER VARYING
NATIONAL CHAR VARYING
NCHAR VARYING
```

大对象

Oracle提供了几种数据类型支持在单列中存储大量数据。这些数据类型通常用于存储图像、声音和其他大对象。

BLOB

大原始二进制对象，最大长度可以为4GB。在不同字符集的系统之间移动数据时，不能用Oracle转换BLOB数据。在引用BLOB列时，返回一个LOB定位符。

CLOB

最大长度可以为4GB的大字符数据。

NCLOB

由NLS字符集字符组成的大字符数据，最大长度可以为4GB。在不同字符集的系统之间移动数据时，不能用Oracle转换NCLOB数据。

BFILE

提供对存储在Oracle数据库之外的操作系统文件中的二进制文件访问。该文件最大可为4GB。

数字数据

数字数据是可以不进行数据转换而直接参与算术运算的数据。Oracle只有一种数字类型：NUMBER 类型。

注意：NUMBER 可以控制的值范围是 10^{-130} 和 $9.99999...x\ 10^{125}$，但数字精度只有 38 位。

数字数据元素可以用以下方式表示：

```
NUMBER [(precision[,scale])]
```

精度（precision）

数字中的数字位数，范围为 1~38。

尾数（Scale）

小数点右边的数字位数，可以在 -84~127 之间变化。

如果省略尾数，则将数字当作整数，而且不存储小数点部分。如果既省略精度也省略尾数，则将数字当作为浮点数字。

警告：经常指定一个数字数据类型的精度是一个好想法。如果你不指定，则一些ODBC驱动将认为其精度为 0，与 Oracle 的默认精度为 38 相反。此时，ODBC 兼容的客户工具将不能允许插入或更新列值不为 0 的行。

尽管 Oracle 存储的所有数字数据都可以当作 NUMBER，但下面的数据类型也可以认为与 ANSI SQL 兼容：

```
FLOAT
NUMERIC
DECIMAL
NUMBER
INTEGER
INT
SMALLINT
DOUBLE PRECISION
REAL
```

日期

Oracle使用DATE数据类型存储日期和时间信息。Oracle DATE数据以包含下列信息的专有格式存储:

- 世纪（century）
- 年（year）
- 月（month）
- 日（day）
- 小时（hour）
- 分钟（minute）
- 秒（second）

要以DATE数据类型存储数据，则日期和/或时间信息必需转换为Oracle内部格式。如果字符代表的日期与默认格式（在初始化文件*INIT.ORA*的NLS_DATE_FORMAT参数中指定）相匹配，那么Oracle就可以自动执行这种转换。例如，如果DD-MON-YY的默认NLS_DATE_FORMAT有效，那么下面的SQL语句将插入日期2076年7月4日到表的T_DATE列中:

```
INSERT INTO sample_table (t_date)
VALUES ('07-JUL-76');
```

如果日期和/或时间信息在另一种格式下可用，你就可以使用Oracle内部的TO_DATE函数执行转换。例如，如果相同的日期以07/04/76提供，那么这里显示的SQL语句将使用Oracle的TO_DATE函数插入下面行的数据:

```
INSERT INTO sample_table (t_date)
VALUES (TO_DATE('07-JUL-76','MM/DD/YY'));
```

要了解有关TO_DATE和其他转换函数的更多信息，请参见第五章。

空值（NULL）

有时，SQL令人混淆的一个方面是NULL的概念。NULL既不是字符也不是数字、

它是缺省数据。字符和数字数据元素都可以被设置为NULL，这显示了该元素不包含任何值。

NULL与0也不相同。事实上，任何包含NULL的数据元素的算术运算将得到NULL值。例如，如果当前值为NULL，那么下面的表达式将等于NULL:

```
(a+10)*20
```

注意：如果执行的算术运算包括一个NULL列，对编程者而言，Oracle的这一特性将使工作变得艰难。幸运的是，Oracle提供了NVL函数来解决这一问题。要了解更多信息，请参见第五章。

相似地，在使用比较操作时，NULL不与任何条件匹配，即使从逻辑上看似乎匹配也不行。如果col1包含了NULL值，那么下面的从句:

```
WHERE col1 <> 1
```

将不会得到TRUE。

你可以测试数据元素中存在或缺少NULL的情况。如果表达式求得的值为NULL，那么条件IS NULL求得的结果将为TRUE，如果值不为NULL，则求得的结果将为FALSE。

定位符

Oracle提供了许多数据类型，其基本目的是提供对存储在数据库内部或外部对象的有效引用。

ROWID

ROWID返回代表行数据的物理位置的字符串，并包含了Oracle定位数据行需要的所有信息，包括:

- 数据文件包含的行
- 包含行的文件块

- 数据块中的行位置
- 对象的对象编号（只有 Oracle8 有）

除了表以簇的形式存储的情况之外，ROWID 将惟一定义数据的单个行。

通常ROWID以18个字符串的形式返回且可用于数据操纵语言（DML）语句，就像Oracle的任何其他列一样。

注意：虽然ROWID可以存储在表列中，但你永远也不能依赖保留ROWID值的行。因为ROWID代表了物理位置，所以，如果行以不同的形式存储，则该值将会发生变化。例如，导入以后再导出，则任何存储的ROWID都可能无效。

UROWID

因为Oracle对象的一些类型可能没有Oracle固定的或产生的物理位置（如通过透明网关访问的对象），所以 Oracle 已经开发了通用 ROWID 数据类型（Universal ROWID,UROWID）。在可用时，它可以包含物理的ROWID，否则，它包含逻辑ROWID。Oracle 强烈推荐你使用UROWID数据类型代替ROWID，所以可以使用任何类型的位置信息。

伪列

虽然不是实际的数据类型，Oracle 支持许多特殊目的的数据元素。这些元素并不实际包含在表中，但可以用在 SQL 语句中，就像它们是表的一部分一样。

ROWNUM

对被SQL查询返回的每一行数据，ROWNUM将包含显示检索行顺序的数字。例如，检索的第一行 ROWNUM 为 1，第 2 行 ROWNUM 为 2 等等。该方法可以用于限制被查询返回的行号。下面的 SQL 语句利用了 ROWNUM 伪列：

```
SELECT *
FROM emp
WHERE ROWNUM < 11;
```

警告：ROWNUM返回的数字显示了从表中提取行的顺序，但这不是行一直显示的顺序。例如，如果SQL语句包括了ORDER BY从句，那么行将不会以ROWNUM的顺序显示，因为在排序操作之前已经分配了ROWNUM。

CURRVAL

在使用Oracle SEQUENCE值时（见第二章的“CREATE/ALTER/DROP SEQUENCE”），伪列CURRVAL返回序列的当前值。要引用CURRVAL，就必须与序列相关。

```
[schema.]sequence_name.CURRVAL
```

schema

序列的所有者。如果省略*schema*，Oracle就假设为当前连接到数据库的用户名。

sequence_name

Oracle序列名。

CURRVAL

序列的当前值。

NEXTVAL

在使用Oracle SEQUENCE值时，伪列NEXTVAL返回序列的下一个值，而且使序列增加1。你可以只在NEXTVAL与序列相关时才引用它：

```
[schema.]sequence_name.NEXTVAL
```

schema

序列的所有者。如果省略*schema*，Oracle就假设为当前连接到数据库的用户名。

sequence_name

Oracle序列名。

NEXTVAL

序列的下一个值。

注意：Oracle 将只在给定的 SQL 语句中增加序列，所以，如果语句中包含 NEXTVAL 的多个引用，那么第二个和后继的引用将返回相同的 CURRVAL 值。

LEVEL

被分层查询（使用CONNECT BY从句）返回的每一行，LEVEL 返回 1 表示根节点，2 表示返回根节点的子节点等等。根节点是转换树中的最高节点、子节点是任何非根节点的节点，父节点是有子节点的节点，而叶节点是没有子节点的节点。

USER

该伪列将一直包含你连接到数据库的 Oracle 用户名。

SYSDATE

该伪列将包含当前日期和时间。该列是一个标准的 Oracle 日期数据类型。

在SYSDATE中包含的日期和时间来源于处理查询的服务器，而不是运行查询的客户端。所以，如果你从伦敦的客户工作站连接到东京的服务器上，那么日期和时间将是东京服务器上的具体日期和时间（而且日期大概会提前 1 天）。

警告：如果你通过数据库链接返回 SYSDATE 列（例如，SELECT SYSDATE FROM *dual@london*），则会返回被连接的服务器的日期和时间，而不是数据库链接引用的远端服务器的日期和时间。

数据转换

在数据库中有多种方式代表数据。例如薪水，正常情况下认为是数字值，如25 000，也可以简单地用如“25000”的字符串来代表。同样地，职工 ID 可以用数字（500）或字符串（“500”）来代表。如果你试图在大部分计算机语言上执行字符值算术运算，就会产生错误，对 SQL 则不是这样。当有必要（而且可能）完成查询操作时，Oracle 自动执行数据转换。在下面的 SQL 语句中，假设数据库中定义的 *sal* 为 CHAR(6) ——固定长度为 6 字节的字符串：

```
SELECT ename, SAL * 1.1
FROM scott.emp;
```

该字符串被常量 1.1 相乘。要执行该操作，Oracle 首先将字符串转换成数字，然后执行相乘。这种类型的自动转换称为隐式数据转换（implicit data conversion）。

警告：虽然很方便，但是隐式数据转换也有一定的CPU应用开销，所以，决定怎样在数据库中存储数据元素时一定要特别小心。

隐式数据转换可能在性能上会有预测不到的负面影响，因为它在很大程度上影响了Oracle查询优化器产生执行计划。例如，可能因为隐式数据转换禁止了使用索引，结果导致整表扫描。

SQL 也提供了许多函数执行显式数据转换:

TO_CHAR

执行从数字到字符和从日期到字符的转换。

TO_NUMBER

执行字符到数字的转换。

TO_DATE

执行从字符到日期的转换。

要得到更多信息，参见第五章。

关系操作符

操作符操纵数据元素并返回结果。被操作的数据元素叫做操作数（operands）或提取变量（arguments），特殊字符或关键字代表了实际操作符。Oracle SQL 支持许多类型的操作符，下面一一列出。

算术操作符

算术操作符在一组数据元素和/或常量之间执行算术运算。表 1-1 列出了 SQL 中可以使用的算术操作符:

表 1-1: SQL 算术操作符

操作符	描述	例子
+	加	sal + comm
-	减	sal - comm
*	乘	sal * 1.1
/	除	sal / 12
-	负号	-sal
+	正号	+sal

连接操作符

连接操作符（||）组合两个字符串。考虑下面的 SQL 语句:

```
SELECT fname || ' ' || lname
FROM employee_master;
```

该语句返回（为每一行）单个字符串，由姓、空格和名组成。

警告: 尽管大部分 Oracle 平台使用实竖线（||）作为连接操作符，但有一些平台，最显著的是使用 EBCDIC 字符集的 IBM 平台，使用分断竖线（¦¦）作为连接操作符。在 ASCII 和 EBCDIC 字符集之间进行转换时，这些字符的转换可能不正确。

如果有一个连接字符串为 NULL，则结果是非 NULL 字符串。NULL 字符串被当作空字符串。如果连接的两个字符串都是 NULL，则结果字符串为 NULL。如果有一个操作数为 VARCHAR2 数据类型，则结果字符串就是 VARCHAR2 数据类型。

警告: 如果操作数是 CHAR 数据类型，那么连接字符串的长度最多为 2000 个字符，否则最多为 4000 个字符。其他字符类型，如 LONG 和 CLOB，则不能被连接。

比较操作符

比较操作符用于比较两个数据元素(或者数据元素和常量)，根据两个元素的关系返回结果 TRUE、FALSE 或 NULL。表 1-2 列出了在 SQL 中可以使用的比较操作符。

表 1-2: SQL 的比较操作符

操作符	使用	描述	例子
=	*a* = *b*	测试两个操作数相等	SELECT * FROM emp WHERE sal = 500
!=	*a* != *b*	测试两个操作数不相等	SELECT * FROM emp WHERE sal != 500
^=	*a* ^= *b*	测试两个操作数不相等	SELECT * FROM emp WHERE sal ^= 500
<>	*a* <> *b*	测试两个操作数不相等	SELECT * FROM emp WHERE sal <> 500
<	*a* < *b*	测试操作数 *a* 比操作数 *b* 小	SELECT * FROM emp WHERE sal < 500
!<	*a* !< *b*	测试操作数 *a* 不小于操作数 *b*。这与 >= 相同	SELECT * FROM emp WHERE sal !< 500
>	*a* > *b*	测试操作数 *a* 比操作数 *b* 大	SELECT * FROM emp WHERE sal > 500
!>	*a* !> *b*	测试操作数 *a* 不大于操作数 *b*。这与 <= 相同	SELECT * FROM emp WHERE sal !> 500
<=	*a* <= *b*	测试操作数 *a* 小于等于操作数 *b*	SELECT * FROM emp WHERE <= 500
>=	*a* >= *b*	测试操作数 *a* 大于等于操作数 *b*	SELECT * FROM emp WHERE >= 500

表 1-2: SQL 的比较操作符（续）

操作符	使用	描述	例子
IN	*a* IN (*b*,*c*...)	测试操作数 *a* 在提供的列表中（操作数 *b*、操作数 *c* 等等）至少有一个匹配值	SELECT * FROM emp WHERE sal IN (500,600,700)
NOT IN	*a* NOT IN (*b*,*c*...)	表示操作数 *a* 与提供的列表中的元素（操作数 *b*、操作数 *c* 等等）没有匹配值	SELECT * FROM emp WHERE sal NOT IN (500,600,700)
ANY	*a* = ANY (*b*,*c*...) *a* < ANY (*b*,c...) *a* > ANY (*b*,*c*...) 等等	测试指定的关系（例如，=、<>、<、> 等等）在提供的元素清单（操作数 *b*、操作数 *c* 等等）中至少有一个是真。当测试相等时，就等于 IN 的功能	SELECT * FROM emp WHERE sal = ANY (500,600,700)
SOME	*a* = SOME (*b*,*c*...) *a* < SOME (*b*,*c*...) *a* > SOME (*b*,*c*...) 等等	测试指定的关系（例如，=，<>，<，> 等等）在提供的元素清单（操作数 *b*、操作数 *c* 等等）中至少有一个是真。当测试相等时，就等于 IN 的功能	SELECT * FROM emp WHERE sal = SOME (500,600,700)
ALL	*a* = ALL(*b*,*c*...) *a* < ALL (*b*,*c*...) *a* > ALL (*b*,*c*...) 等等	测试指定的关系（例如，=、<>，<，> 等等）对提供的所有元素清单（操作数 *b*、操作数 *c* 等等）都为真	SELECT * FROM emp WHERE sal = ALL (500,600,700)
BETWEEN	*a* BETWEEN *b* and *c*	测试操作数 *a* 大于等于操作数 *b* 并小于等于操作数 *c*	SELECT * FROM emp WHERE sal BETWEEN 400 AND 600
NOT BETWEEN	*a* NOT BETWEEN *b* and *c*	测试操作数 *a* 小于等于操作数 *b* 并大于等于操作数 *c*	SELECT * FROM emp WHERE sal NOT BETWEEN 400 AND 600

表 1-2: SQL 的比较操作符（续）

操作符	使用	描述	例子
EXISTS	EXISTS(*query*)	测试该查询至少返回一行	SELECT * FROM emp WHERE EXISTS (SELECT deptno FROM dept d WHERE deptno= e.deptno)
NOT EXISTS	NOT EXISTS (*query*)	测试查询不返回一行	SELECT * FROM emp WHERE NOT EXISTS (SELECT deptno FROM dept d WHERE deptno= e.deptno)
LIKE	*a* LIKE *b*	测试操作数 *a* 匹配操作数 *b* 的模式。模式可以包含 _ 匹配该位置的单个字符，或者用 % 匹配所有的字符	SELECT * FROM emp WHERE ename LIKE 'SMI%'
NOT LIKE	*a* NOT LIKE *b*	测试操作数 *a* 不匹配操作数 *b* 的模式。模式可以包含 _ 匹配该位置的单个字符，或者用 % 匹配所有的字符	SELECT * FROM emp WHERE ename NOT LIKE 'SMI%'
IS NULL	*a* IS NULL	测试操作数 *a* 为 NULL	SELECT * FROM emp WHERE comm. IS NULL
IS NOT NULL	*a* IS NOT NULL	测试操作数 *a* 不为 NULL	SELECT * FROM emp WHERE comm. IS NOT NULL

逻辑操作符

SQL提供的逻辑操作符与大部分其他编程语言中可以使用的逻辑操作符相似。逻辑

操作符AND和OR组合两个布尔值之间的结果并产生基于这两个结果的单结果，而逻辑操作符NOT得到相反结果。布尔值可以是能够求TRUE或FALSE值的任何表达式。通常该值来源于比较表达式。表1-3列出了在SQL中可以使用的逻辑操作符，并提供了每一个操作符可能的结果。

表1-3：SQL的逻辑操作符

操作符	操作数1	操作数2	结果
AND（与）	TRUE（真）	TRUE	TRUE
	FALSE（假）	FALSE	FALSE
	TRUE	FALSE	FALSE
	FALSE	TRUE	FALSE
	TRUE	NULL（空）	NULL
	FALSE	NULL	FALSE
	NULL	TRUE	NULL
	NULL	FALSE	FALSE
	NULL	NULL	NULL
OR（或）	TRUE	TRUE	TRUE
	FALSE	FALSE	FALSE
	TRUE	FALSE	TRUE
	FALSE	TRUE	TRUE
	TRUE	NULL	TRUE
	FALSE	NULL	NULL
	NULL	TRUE	TRUE
	NULL	FALSE	NULL
	NULL	NULL	NULL
NOT（非）	TRUE		FALSE
	FALSE		TRUE
	NULL		NULL

集合操作符

不像其他关系操作符操作一个或多个单个的数据元素，集合操作符工作在两个查询返回数据的整个集合上。表1-4列出了SQL中可以使用的集合操作符：

表 1-4: SQL 的集合操作符

操作符	描述	例子
UNION	组合每个查询返回的所有行，并删除重复行	SELECT * FROM emp WHERE deptno=10 UNION SELECT * FROM emp WHERE sal > 500
UNION ALL	组合每个查询返回的所有行，并包括重复行	SELECT * FROM emp WHERE deptno=10 UNION ALL SELECT * FROM emp WHERE sal > 500
MINUS	采用第一个查询返回的行，减去第二个查询中也同样返回的行，最后返回剩下的行	SELECT * FROM emp MINUS SELECT * FROM emp WHERE sal > 500
INTERSECT	只返回两个查询共同返回的行	SELECT * FROM emp WHERE deptno = 10 INTERSECT SELECT * FROM emp WHERE SAL>500

SQL 语句的结构

SQL 语句可以被分成 3 个主要部分:

- SQL 操作

- 目标
- 条件

只有前两个组成部分是必要的；根据执行的SQL操作决定条件可选或不应用条件。

SQL操作

SQL DML语句执行4种基本操作，将分别在本节讨论。每一种操作也是SQL语句名，有关每一条语句的详细语法在第三章中描述。

SELECT

SELECT语句大概是所有SQL语句中最普通和使用最广泛的语句。SELECT语句的目的是从数据库中提取数据。该语句可以从一个或多个数据库表或视图中返回数据元素，也可以从至少一个数据库表或视图和/或常量、或者从常量中返回，也可以从包括数据元素的表达式中返回。SELECT语句总有一个目标组件，而且通常有条件组件。SELECT语句的目标是FROM从句（从该表或视图中检索数据）列出的表集合或视图集合。条件是WHERE从句中的表达式，而且可能也有HAVING从句，用来限制返回的行。如果没有指定条件，则返回目的表和/或视图的所有行。

INSERT

INSERT语句在目标数据库表或视图中产生了新数据行。该语句提供列清单接收被提供的数据（如果没有提供清单就指表或视图的所有列），并提供每一列中被代替的包含数据元素的相应清单。条件元素不应用于INSERT中。

UPDATE

UPDATE语句修改数据库表或视图中已经存在的数据。UPDATE语句总有相关目标，而且通常有条件。如果没有指定条件，则更新目标表中的所有行。

DELETE

DELETE语句从数据库表或视图中删除所有行。该语句将总有相关目标，而且通常有条件。如果不指定条件，则删除目标表或视图中的所有行。

目标

所有的SQL DML语句都在一个或多个数据库表或视图上进行操作。目标组件的目的是定义那些表或视图。根据使用的语句不同，该组件也不同。例如，SELECT和DELETE语句具有相似的目标结构:

```
SELECT *
FROM emp            --这是目标组件
WHERE depno = 10

DELETE
FROM emp            --这是目标组件
WHERE deptno = 10
```

但是，INSERT和UPDATE语句则使用了不同的目标:

```
INSERT INTO emp     --这是目标组件
    (empno, ename, sal, hiredate)
VALUES ('1234','Dave Kreines',500,'06-01-00')

UPDATE emp          --这是目标组件
SET sal = 600
WHERE empno = '1234'
```

连接

当引用两个或多个表或视图作为SELECT语句的目标时，就叫做连接。关系数据库的基本概念之一就是通过指定怎样将表相关（因此就产生了术语“相关的”），来将两张或多张表组合为单结果集。两张或多张表或视图通常通过共享数据的一列或多列彼此相关，这样的列叫做键列。键列的一个例子是部门编号。图1-1显示了这样的关系。

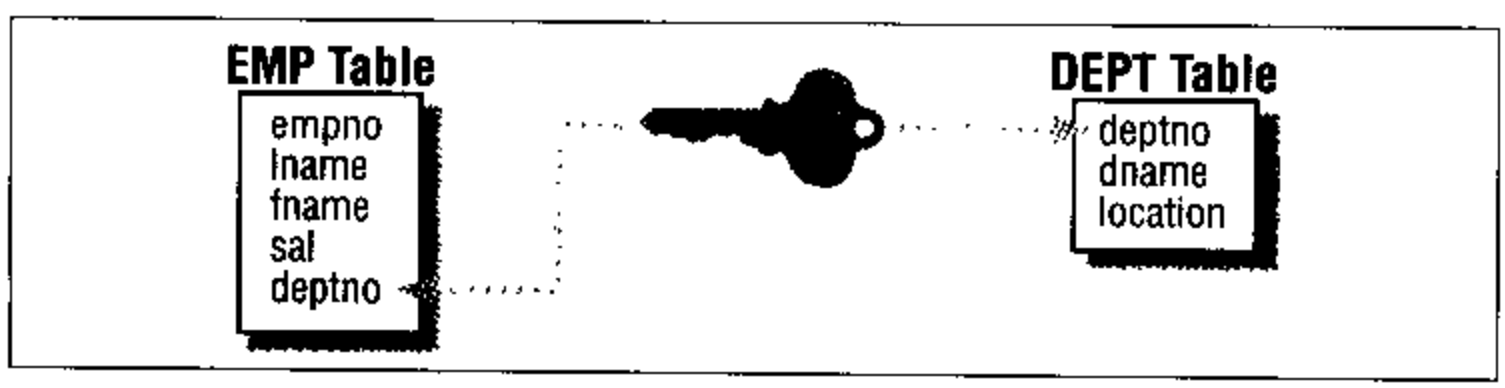


图1-1: 通过一个键列相关的两张表

目标组件说明了连接中包含的表或视图，而且条件组件告诉Oracle怎样将表或视图彼此相关。

下面是连接emp和dept表的例子:

```
SELECT ename, location
FROM emp, dept
WHERE emp.deptno = dept.deptno
```

该语句指示Oracle首先用形成两张表的数据行的所有可能组合从数据库中返回每个职工的名字和地点，然后返回部门编号相匹配的所有行。注意每张表的行数可以不同，这是用作匹配的数据值。dept表的一行可以、而且可能会匹配emp表的多行。

注意：通过连接两张表形成所有行的可能组合的思想是概念上的。Oracle几乎总能找到更好的方式，从而产生连接结果。

如果没有指定连接的条件，则返回两张表的所有可能组合行作为结果。换句话说，在第一张表中的每一行都与第2张表中的每一行相匹配（假定这两张表都被连接）。结果称为笛卡儿产品（Cartesian product），而这通常是你希望避免的一些事情。返回的行号等于第一张表中的行数乘以第二张表中的行数。如果开始时表很大，则笛卡儿产品中的行数就变得非常大。因为不得不产生那些行，所以性能将会大大减弱，而且不太可能有多大用途。笛卡儿产品通常代表书写的查询中有错误。

警告：在FROM从句中存在两张或多张表或视图，而且在WHERE从句中没有对应的连接条件集，就会导致笛卡儿产品。

外部连接

在标准连接中，只有在每一个连接的表或视图中存在对应的行时才会返回行。即使在其中一张表中不存在匹配的行，外部连接（outer join）也允许返回数据。外部连接通过在你想要选择的表的列名后面添加加号（+）进行指定。换句话说，（+）意味着“如果不存在匹配的行，则在该表中包含NULL值的所有列上添加一个幻影行”。例如，如果在dept表中没有匹配的部门编号就显示NULL:

```
SELECT ename, location
FROM emp, dept
WHERE emp.deptno = dept.deptno (+)
```

通常，连接emp和dept表的查询将只返回已经分配了有效部门的职工行。通过在

dept.deptno列名后添加加号（+），就将连接组成了外部连接。现在返回了所有职工的行记录，无论他们是否被分配到有效的部门。

条件

使用WHERE从句进行说明的条件组件定义了被SELECT、UPDATE或DELETE语句操作的特定行。虽然WHERE从句可以很复杂，但它最终是求得每行数据的TRUE或FALSE值，而且该行为控制了是否该操作发生在每一行上。考虑下列查询：

```
SELECT *
FROM emp
WHERE sal > 500
```

Oracle将查找emp表中的每一行数据，而且求得条件sal > 500的值。所有求得该表达式的值为TRUE的行都将被返回，而不会返回那些求得该条件的值为FALSE或未知（那些sal少于或等于500，或者包含NULL的sal值）的行。

WHERE从句的另一个重要用法是定义涉及一张表和另一张表执行连接的列。下列查询包括简单连接：

```
SELECT ename, loc
FROM EMP, DEPT
WHERE emp.deptno = dept.deptno
```

emp和dept表都具有deptno列，包含了部门编号。这两列不需要同名，但它们必须有代表相同意义的数据。在这个例子中，部门编号也存在于dept表的deptno列中。因为我们知道emp表中的部门编号有dept表的deptno列相同的意义，所以将使用这些列指定连接的条件。

警告： 在该例子中，列名（deptno）采用表名（emp或dept）的前缀。使用该前缀是因为联接表中具有同名列。结果，你需要使用表名来限制列名，所以Oracle知道你指哪一列。如果你没有这样处理，Oracle将返回一个错误，因为列名将变得模棱两可。这可能对你显而易见，但对Oracle则不是。

SQL 语句

Oracle支持大量SQL语句，而且在本书的随后章节中都将进行描述。为了方便你查

找特定的SQL语句，表1-5以字母顺序提供了SQL命令，并包括了简要描述及章号和标题，用来找到该语句的完整语法和描述。

表 1-5：本书中包括的 SQL 语句

语句	描述	在第几章中找到	在这个标题下
ALTER CLUSTER	重新定义将来的存储分配或者为簇分配扩展盘区	2	"CREATE/ALTER/DROP CLUSTER"
ALTER DATABASE	改变现存数据库中的一个或多个特性	2	"CREATE/ALTER DATABASE"
ALTER FUNCTION	重新编译存储的 PL/SQL 函数	7	"ALTER FUNCTION"
ALTER INDEX	改变索引特性	2	"CREATE/ALTER/DROP INDEX"
ALTER MATERIALIZED VIEW	改变存储特性或自动刷新物化视图或快照的特性	2	"CREATE/ALTER/DROP MATERIALIZED VIEW"
ALTER MATERIALIZED VIEW LOG	改变物化视图日志的存储特性	2	"CREATE/ALTER/DROP MATERIALIZED VIEW LOG"
ALTER PACKAGE	重新编译 PL/SQL 包	7	"ALTER PACKAGE"
ALTER PROCEDURE	重新编译 PL/SQL 存储过程	7	"ALTER PROCEDURE"
ALTER PROFILE	从现存查询分析器中增加、改变或删除资源限制	2	"CREATE/ALTER/DROP PROFILE"
ALTER RESOURCE COST	修改公式计算会话中耗用的总资源	2	"ALTER RESOURCE COST"
ALTER ROLE	对需要使能的角色改变授权的层次	2	"CREATE/ALTER/DROP ROLE"
ALTER ROLLBACK SEGMENT	改变回退段的在线状态或修改存储特性	2	"CREATE/ALTER/DROP ROLLBACK SEGMENT"

表 1-5: 本书中包括的 SQL 语句（续）

语句	描述	在第几章中找到	在这个标题下
ALTER SEQUENCE	改变 Oracle 序号发生器的特性	2	"CREATE/ALTER/DROP SEQUENCE"
ALTER SESSION	改变当前数据库会话的功能特性	2	"ALTER SESSION"
ALTER SNAPSHOT	改变快照的存储特性或自动刷新特性	2	"CREATE/ALTER/DROP SNAPSHOT"
ALTER SNAPSHOT LOG	改变快照日志的存储特性	2	"CREATE/ALTER/DROP SNAPSHOT LOG"
ALTER SYSTEM	动态改变数据库实例的特性	2	"ALTER SYSTEM"
ALTER TABLE	修改表的特性	2	"CREATE/ALTER/DROP TABLE"
ALTER TABLESPACE	改变现存表空间的特性	2	"CREATE/ALTER/DROPTABLESPACE"
ALTER TRIGGER	重新编译 PL/SQL 触发器	7	"ALTER TRIGGER"
ALTER USER	改变用户的安全性或存储特性	2	"CREATE/ALTER/DROP USER"
ALTER VIEW	重新编译视图	2	"CREATE/ALTER/DROP VIEW"
ANALYZE	收集或删除数据库中对象的统计信息	2	"ANALYZE"
ASSOCIATE STATISTICS	关联数据库对象的统计计算方法	2	"ASSOCIATE STATISTICS"
AUDIT	设置并发用户会话中特定 SQL 语句的审核	2	"AUDIT (SQL Statements)"
AUDIT	指定特定架构对象的审核	2	"AUDIT (Schema Objects)"
CALL	执行存储的 PL/SQL 过程	2	"CALL"

表 1-5：本书中包括的 SQL 语句（续）

语句	描述	在第几章中找到	在这个标题下
COMMENT	增加表、视图、快照或列的注释	2	"COMMENT"
CREATE CLUSTER	创建簇，该簇至少包含一张表的一个或多个公共列	2	"CREATE/ALTER/DROP CLUSTER"
CREATE CONTROLFILE	重新创建控制文件，允许改变一些参数	2	"CREATE CONTROLFILE"
CREATE DATABASE	创建数据库并指定相关参数	2	"CREATE/ALTER DATABASE"
CREATE DATABASE LINK	创建数据库链接，提供对远端数据库对象的访问	2	"CREATE/DROP DATABASE LINK"
CREATE DIMENSION	创建维度定义列集之间的父子关系	2	"CREATE/DROP DIMENSION"
CREATE DIRECTORY	创建目录对象，指定存储 BFILE 对象的操作系统目录	2	"CREATE/DROP DIRECTORY"
CREATE FUNCTION	创建存储的 PL/SQL 函数	7	"CREATE FUNCTION"
CREATE INDEX	创建表或簇的至少一列的索引	2	"CREATE/ALTER/DROP INDEX"
CREATE MATERIALIZED VIEW	创建物化视图，也调用快照	2	"CREATE/ALTER/DROP MATERIALIZED VIEW"
CREATE MATERIALIZED VIEW LOG	创建物化视图日志	2	"CREATE/ALTER/DROP MATERIALIZED VIEW LOG"
CREATE PACKAGE	创建 PL/SQL 包	7	"CREATE PACKAGE"
CREATE PROCEDURE	创建 PL/SQL 存储过程	7	"CREATE PROCEDURE"
CREATE PROFILE	创建 profile，设置数据库资源上的限制	2	"CREATE/ALTER/DROP PROFILE"

表 1-5: 本书中包括的 SQL 语句（续）

语句	描述	在第几章中找到	在这个标题下
CREATE ROLE	创建角色，可以给用户授予一系列的权限	2	"CREATE/ALTER/DROP ROLE"
CREATE ROLLBACK SEGMENT	创建回退段，用于 Oracle 存储数据，必要时用事务回退所做的改变	2	"CREATE/ALTER/DROP ROLLBACK SEGMENT"
CREATE SCHEMA	创建多张表 / 或视图，而且在单个语句中执行授权	2	"CREATE SCHEMA"
CREATE SEQUENCE	创建 Oracle 序列，用于自动产生序列号	2	"CREATE/ALTER/DROP SEQUENCE"
CREATE SNAPSHOT	创建快照（或物化视图）	2	"CREATE/ALTER/DROP SNAPSHOT"
CREATE SNAPSHOT LOG	创建快照日志	2	"CREATE/ALTER/DROP SNAPSHOT LOG"
CREATE TABLE	通过指定现存表的结构或引用创建表	2	"CREATE/ALTER/DROP TABLE"
CREATE TABLESPACE	创建一个新的表空间，选择性地指定随后在表空间中创建的对象的默认存储特性	2	"CREATE/ALTER/DROPTABLESPACE"
CREATE TRIGGER	创建 PL/SQL 触发器	7	"CREATE TRIGGER"
CREATE TEMPORARY TABLESPACE	创建临时表空间	2	"CREATE TEMPORARY TABLESPACE"
CREATE USER	创建新数据库用户并分配安全属性和存储属性	2	"CREATE/ALTER/DROP USER"
CREATE VIEW	创建视图	2	"CREATE/ALTER/DROP VIEW"
CREATE SYNONYM	创建数据库对象的公有或私有同义词	2	"CREATE/DROP SYNONYM"

表 1-5: 本书中包括的 SQL 语句（续）

语句	描述	在第几章中找到	在这个标题下
DROP SYNONYM	删除数据库对象的公有或私有同义词	2	"CREATE/DROP SYNONYM"
DELETE	删除表、视图或快照的一行或多行	3	"DELETE"
DISASSOCIATE STATISTICS	取消统计计算方法与数据库对象之间的关联	2	"DISASSOCIATE STATISTICS"
DROP CLUSTER	从数据库中删除簇	2	"CREATE/ALTER/DROP CLUSTER"
DROP DATABASE LINK	从数据库中删除数据库链接	2	"CREATE/DROP DATABASE LINK"
DROP DIMENSION	从数据库中删除维度	2	"CREATE/DROP DIMENSION"
DROP DIRECTORY	从数据库中删除目录对象	2	"CREATE/DROP DIRECTORY"
DROP FUNCTION	删除存储的 PL/SQL 函数	7	"DROP FUNCTION"
DROP INDEX	从数据库中删除索引	2	"CREATE/ALTER/DROP INDEX"
DROP MATERIALIZED VIEW	从数据库中删除物化视图	2	"CREATE/ALTER/DROP MATERIALIZED VIEW"
DROP MATERIALIZED VIEW LOG	从数据库中删除物化视图日志	2	"CREATE/ALTER/DROP MATERIALIZED VIEW LOG"
DROP PACKAGE	从数据库中删除 PL/SQL 包	7	"DROP PACKAGE"
DROP PROCEDURE	从数据库中删除 PL/SQL 存储过程	7	"DROP PROCEDURE"
DROP PROFILE	从数据库中删除 profile	2	"CREATE/ALTER/DROP PROFILE"

表 1-5: 本书中包括的 SQL 语句（续）

语句	描述	在第几章中找到	在这个标题下
DROP ROLE	从数据库中删除角色	2	"CREATE/ALTER/DROP ROLE"
DROP ROLLBACK SEGMENT	从数据库中删除回退段	2	"CREATE/ALTER/DROP ROLLBACK SEGMENT"
DROP SEQUENCE	从数据库中删除序号发生器	2	"CREATE/ALTER/DROP SEQUENCE"
DROP SNAPSHOT	从数据库中删除快照	2	"CREATE/ALTER/DROP SNAPSHOT"
DROP SNAPSHOT-LOG	从数据库中删除快照日志	2	"CREATE/ALTER/DROP SNAPSHOT LOG"
DROP TABLE	从数据库中删除表	2	"CREATE/ALTER/DROP TABLE"
DROP TABLESPACE	从数据库中删除表空间	2	"CREATE/ALTER/DROP TABLESPA-CE"
DROP TRIGGER	从数据库中删除 PL/SQL 触发器	7	"DROP TRIGGER"
DROP USER	从数据库中删除用户	2	"CREATE/ALTER/DROP USER"
DROP VIEW	从数据库中删除视图	2	"CREATE/ALTER/DROP VIEW"
EXPLAIN PLAN	为 SQL 语句创建执行计划的解释	2	"EXPLAIN PLAN"
GRANT	对一个或多个用户或角色授予系统权限	2	"GRANT (System Privilege or Role)"
GRANT	对一个或多个用户或角色授予数据库对象权限	2	"GRANT (Object Privileges)"

表 1-5: 本书中包括的 SQL 语句（续）

语句	描述	在第几章中找到	在这个标题下
INSERT	在表或视图中插入一行数据	3	"INSERT"
NOAUDIT	停止架构对象以前的 AUDIT 语句定义的审核	2	"NOAUDIT (Schema Objects)"
NOAUDIT	停止 SQL 语句以前的 AUDIT 语句定义的审核	2	"NOAUDIT (SQL Statements)"
RENAME	改变现存表、视图、序号发生器或私有同义词的名字	2	"RENAME"
REVOKE	从一个或多个用户或角色中收回数据库权限或角色	2	"REVOKE (System Privilege or Role)"
REVOKE	从一个或多个用户或角色中收回数据库对象的权限	2	"REVOKE (Object Privileges)"
SAVEPOINT	定义一个事务点，可以使用 ROLLBACK 命令进行回退	3	"SAVEPOINT"
SELECT	从表、视图或快照中检索数据	3	"SELECT"
SET CONSTRAINT	在事务层指定怎样检查约束条件	3	"SET CONSTRAINT"
SET ROLE	使当前会话的角色能或不能使用	3	"SET ROLE"
SET TRANSACTION	建立当前事务为只读或读－写，或指定事务使用的回退段	3	"SET TRANSACTION"
TRUNCATE	从表或簇中删除所有行	3	"TRUNCATE"
UPDATE	改变一张或多张表、视图或快照中存储在列中的数据值	3	"UPDATE"

第二章

数据定义语句

SQL 语句分成两类：数据定义语言（Data Definition Language，DLL）和数据操纵语言（Data Manipulation Language，DML）。在需要操纵 Oracle 数据库结构时，你可以调用 DDL 语句。你可以使用 DLL 语句定义、修改和删除在 Oracle 数据库中存在的每种类型的对象。因为这一原因，所以理解 DLL 语句并知道怎样适当使用 DDL 语句就非常重要。许多 DDL 语句需要特殊的数据库权限，而且通常由 DBA 执行；对 DBA 而言，当然需要掌握 DDL 的知识。但是 Oracle 开发者和设计者也会发现这些语句的知识非常有用。因为 DDL 语句可以执行许多不同的功能，所以有大量的 DDL 语句，而且可以具有较广范围的选项和语法选择。本章提供了这些语句怎样工作和怎样使用这些语句的快速参考。

第二种分类的 SQL 语句 DML 用于操纵 Oracle 数据库中包含的实际数据。第三章将介绍 DML 语句。

按任务列出的 SQL DLL 语句

在使用 Oracle 数据库时，最烦人而又耗时的方面常常是找出完成特定任务的合适语句。表 2-1 列出了最常用的数据定义任务，以及完成每一任务的 SQL 语句。你会在本章找到每一语句的详细语法。

表 2-1：常见的 DDL 任务和对应的 SQL 语句

目的	使用语句
在表中增加列 / 完整性约束	ALTER TABLE
在数据字典中增加表、视图、快照或列的注释	COMMENT
从现存的 profile 中增加资源限制	ALTER PROFILE
在现存的表空间中增加数据文件	ALTER TABLESPACE
增加 / 重命名数据文件	ALTER TABLESPACE
分配簇的扩展盘区	ALTER CLUSTER
分配表的扩展盘区	ALTER TABLE
允许或不允许访问表	GRANT/ REVOKE ROLE
允许或不允许写表	ALTER TABLE
允许或不允许写表空间	ALTER TABLESPACE
将统计计算方法关联数据库对象	ASSOCIATE STATISTICS
审核特定架构对象	AUDIT
审核特定 SQL 语句	AUDIT
备份当前控制文件	ALTER DATABASE
通过将表空间放在备份或正常模式下，开始 / 结束备份	ALTER TABLESPACE
将回退段改变为在线 / 离线	ALTER ROLLBACK SEGMENT
从现存 profile 上改变资源限制	ALTER PROFILE
改变用户默认角色	ALTER USER
改变用户默认表空间	ALTER USER
改变用户口令	ALTER USER
改变用户 profile	ALTER USER
改变用户表空间限额	ALTER USER
改变用户临时表空间	ALTER USER
改变现存表空间	ALTER TABLESPACE
改变数据库控制文件参数	CREATE CONTROLFILE

表 2-1：常见的 DDL 任务和对应的 SQL 语句（续）

目的	使用语句
改变数据库参数	ALTER DATABASE
改变现存表空间的存储参数	ALTER TABLESPACE
改变授权层，要求启动一个角色	ALTER ROLE
改变索引特性	ALTER INDEX
改变 Oracle 序号发生器特性	ALTER SEQUENCE
改变在用数据库实例的动态可修改特性	ALTER SYSTEM
改变用于计算会话中整个耗用资源的计算公式	ALTER RESOURCE COST
改变当前数据库会话的功能特性，包括许多 NLS 特性	ALTER SESSION
改变数据库名	CREATE CONTROLFILE
改变架构对象名	RENAME
改变现存表、视图、序号发生器或私有同义词的名字	RENAME
改变回退段的在线状态	ALTER ROLLBACK SEGMENT
改变用户的安全性和存储特性	ALTER USER
改变物化视图的存储特性	ALTER MATERIALIZED VIEW LOG
改变快照日志的存储特性	ALTER SNAPSHOT LOG
改变物化视图（或快照）的存储特性或自动刷新特性	ALTER MATERIALIZED VIEW
改变快照的存储特性或自动刷新特性	ALTER SNAPSHOT
选择归档日志/非归档日志的模式	ALTER DATABASE
收集数据库中对象的统计信息	ANALYZE
从现存表中复制一张新表	CREATE TABLE AS
创建簇	CREATE CLUSTER
创建数据库	CREATE DATABASE
创建数据库链接，访问远端数据库上的对象	CREATE DATABASE LINK

表 2-1：常见的 DDL 任务和对应的 SQL 语句（续）

目的	使用语句
创建维度，定义列集对之间的父-子关系	CREATE DIMENSION
创建目录对象，该目录对象指定存储 BFILE 对象的操作系统目录	CREATE DIRECTORY
创建物化视图（也称为快照）	CREATE MATERIALIZED VIEW
创建物化视图日志	CREATE MATERIALIZED VIEW LOG
创建新数据库用户	CREATE USER
创建新表空间	CREATE TABLESPACE
创建 PL/SQL 包	CREATE PACKAGE
创建 PL/SQL 包体	CREATE PACKAGE BODY
创建特定 SQL 语句在表上执行时自动执行的 PL/SQL 触发器	CREATE TRIGGER
创建 profile，在数据库资源上进行限制	CREATE PROFILE
创建数据库对象的公有或私有同义词	CREATE SYNONYM
创建角色	CREATE ROLE
创建回退段	CREATE ROLLBACKSEGMENT
创建快照	CREATE SNAPSHOT
创建快照日志	CREATE SNAPSHOT LOG
创建独立函数	CREATE FUNCTION
通过指定结构或引用现存表创建一张表	CREATE TABLE
创建临时表空间	CREATE TEMPORARY TABLESPACE
创建视图	CREATE VIEW
创建 SQL 语句执行计划的解释	EXPLAIN PLAN
创建表或簇的一列或多列索引	CREATE INDEX
创建用于自动创建序列号的 Oracle 序号发生器	CREATE SEQUENCE
从数据库中删除簇	DROP CLUSTER

表 2-1：常见的 DDL 任务和对应的 SQL 语句（续）

目的	使用语句
从数据库中删除数据库链接	DROP DATABASE LINK
从数据库中删除维度	DROP DIMENSION
从数据库中删除目录对象	DROP DIRECTORY
从数据库中删除函数	DROP FUNCTION
从数据库中删除物化视图（或快照）	DROP MATERIALIZED VIEW
从数据库中删除物化视图日志	DROP MATERIALIZED VIEW LOG
从数据库中删除 PL/SQL 包	DROP PACKAGE
从数据库中删除 profile	DROP PROFILE
从数据库中删除公有或私有同义词	DROP SYNONYM
从现存的 profile 中删除资源限制	ALTER PROFILE
从数据库中删除角色	DROP ROLE
从数据库中删除回退段	DROP ROLLBACKSEGMENT
从数据库中删除序号发生器	DROP SEQUENCE
从数据库中删除快照	DROP SNAPSHOT
从数据库中删除快照日志	DROP SNAPSHOT LOG
从一个或多个用户和/或角色中删除系统权限或角色	REVOKE
从数据库中删除表空间	DROP TABLESPACE
从数据库中删除触发器	DROP TRIGGER
从数据库中删除用户	DROP USER
从数据库中删除视图	DROP VIEW
从数据库中删除索引	DROP INDEX
从一个或多个用户或角色中删除数据库对象上的权限	REVOKE
删除数据库中对象的统计信息	ANALYZE
从数据库中删除表	DROP TABLE

表 2-1：常见的 DDL 任务和对应的 SQL 语句（续）

目的	使用语句
停用数据库触发器	ALTER TRIGGER
为当前会话终止该角色	SET ROLE
通过颠倒优先权审核语句的影响终止审核	NOAUDIT
使统计计算方法与数据库对象之间断开关联	DISASSOCIATE STATISTICS
使数据库触发器可以开始工作	ALTER TRIGGER
为当前会话使能使用一个角色	SET ROLE
启用 / 禁止表上的所有触发器	ALTER TABLE
启用 / 禁止进行数据文件大小的扩展	ALTER DATABASE
启用 / 禁止一张表对另一张表加锁	ALTER TABLE
给一个或多个用户或角色授权系统权限或角色	GRANT
给一个或多个用户或角色授予数据库对象上的权限	GRANT
定义表或簇上的移植或链接行	ANALYZE
发出多个 CREATE TABLE、CREATE VIEW 和 GRANT 语句	CREATE SCHEMA
修改表的列特性	ALTER TABLE
修改表的并行度	ALTER TABLE
修改与表和 / 或列相关的完整性约束	ALTER TABLE
修改回退段的存储特性	ALTER ROLLBACK SEGMENT
修改表的存储特性	ALTER TABLE
打开 / 安装数据库	ALTER DATABASE
执行介质恢复	ALTER DATABASE
让表空间在线	ALTER TABLESPACE
重新编译 PL/SQL 包	ALTER PACKAGE
重新编译独立函数	ALTER FUNCTION
重新编译存储函数	ALTER FUNCTION
重新编译存储包	ALTER PACKAGE

表 2-1：常见的 DDL 任务和对应的 SQL 语句（续）

目的	使用语句
重新编译存储过程	ALTER PROCEDURE
重新编译视图	ALTER VIEW
重新创建控制文件	CREATE CONTROLFILE
重新创建 SQL 语句，建立数据库跟踪文件	ALTER DATABASE
从数据库中删除簇	DROP CLUSTER
从数据库中删除数据库链接	DROP DATABASE LINK
从数据库中删除维度	DROP DIMENSION
从数据库中删除目录对象	DROP DIRECTORY
从数据库中删除函数	DROP FUNCTION
从数据库中删除物化视图（或快照）	DROP MATERIALIZED VIEW
从数据库中删除物化视图日志	DROP MATERIALIZED VIEW LOG
从数据库中删除 PL/SQL 包	DROP PACKAGE
从数据库中删除 profile	DROP PROFILE
从数据库中删除公有或私有同义词	DROP SYNONYM
从现存的 profile 中删除资源限制	ALTER PROFILE
从数据库中删除角色	DROP ROLE
从数据库中删除回退段	DROP ROLLBACK SEGMENT
从数据库中删除序号发生器	DROP SEQUENCE
从数据库中删除快照	DROP SNAPSHOT
从数据库中删除快照日志	DROP SNAPSHOT LOG
从一个或多个用户或角色中删除系统权限或角色	REVOKE
从数据库中删除表空间	DROP TABLESPACE
从数据库中删除触发器	DROP TRIGGER
从数据库中删除用户	DROP USER
从数据库中删除视图	DROP VIEW
从数据库中删除索引	DROP INDEX

表 2-1：常见的 DDL 任务和对应的 SQL 语句（续）

目的	使用语句
从一个或多个用户或角色上收回一个数据库对象上的权限	REVOKE
删除数据库中有关一个对象的统计信息	ANALYZE
从数据库中删除表	DROP TABLE
重命名现存表、视图、序号发生器或私有同义词	RENAME
收缩回滚段为可选或给定大小	ALTER ROLLBACK SEGMENT
指定公式计算会话使用资源的整个开销	ALTER RESOURCE COST
通过将表空间放置在备份模式下启动备份	ALTER TABLESPACE
停止以前的 AUDIT 语句定义的审核	NOAUDIT
停止对表空间的备份	ALTER TABLESPACE
让数据文件在线 / 离线	ALTER DATABASE
让表空间在线 / 离线	ALTER TABLESPACE
验证表、索引或簇的索引	ANALYZE
验证数据库中对象的结构	ANALYZE

SQL 语句的语法

本节提供了用于创建和维护数据库对象的每条SQL语句的详细参考信息。每一条语句都至少列出了一种形式，而且简要解释了语句目的、确切语法和相关参数、示例和用途注释。

注意： 正常情况下，这些语句只用于数据库管理，而且通常由 DBA 发出。因为这一原因，所以许多语句都需要特定权限。一些语句也只在安装了特定Oracle选项时应用，如分区选项（Partition Option）或对象选项（Object Option）。

ALTER RESOURCE COST

ALTER RESOURCE COST

```
[CPU_PER_SESSION weight]
[CONNECT_TIME weight]
[LOGICAL_READS_PER_SESSION weight]
[PRIVATE_SGA weight]
```

修改公式用于计算会话的整个资源成本。该成本可以通过用户 profile 中的 COMPOSITE_LIMIT 参数予以限制。

关键字

CPU_PER_SESSION

指定会话中使用的 CPU 时间数量（以百分之一秒计）。

CONNECT_TIME

指定会话中总共消失的时间（以分钟计）。

LOGICAL_READS_PER_SESSION

指定会话中数据库读块的数量，包括从内存和磁盘读。

PRIVATE_SGA

指定会话中可以在系统全局区中（System Global Area，SGA）分配内存的数量（以字节计）。只有在使用多线程服务器和为会话分配 SGA 中的私有空间时可以应用。

weight

每个资源的整数权重。

注意：整个资源成本通过会话中使用的每一资源的数量乘以分配给资源的权重进行计算，而且为所有 4 种资源都添加了产品。结构以服务单元的形式表达。你必须激活 ALTER RESOURCE COST 权限，方可发出该语句。

示例

给 CPU_PER_SESSION 和 CONNECT_TIME 分配权重：

```
ALTER RESOURCE COST
      CPU_PER_SESSION 100
      CONNECT_TIME 2
```

服务单元（service units，SU）中的结果开销计算为:

```
SU = (CPU_TIME * 100) + Connect_Time * 2;
```

ALTER SESSION

要得到该语句的完全描述，请参见第二章。

ALTER SYSTEM

```
ALTER SYSTEM
 {[SET
   {{RESOURCE_LIMIT = TRUE | FALSE] |
    [GLOBAL_NAMES = TRUE | FALSE] |
    [MTS_SERVERS = integer] |
    [MTS_DISPATCHERS = 'protocol, integer'] |
    [LICENSE_MAX_SESSIONS = integer] |
    [LICENSE_SESSIONS_WARNING = integer] |
    [LICENSE_MAX_USERS = integer]
   }] |
  [ENABLE RESTRICTED SESSION] |
  [DISABLE RESTRICTED SESSION] |
  [FLUSH SHARED POOL] |
  [CHECKPOINT [GLOBAL | LOCAL]] |
  [CHECK DATAFILES GLOBAL | LOCAL] |
  [SWITCH LOGFILE] |
  [ENABLE DISTRIBUTED RECOVERY] |
  [DISABLE DISTRIBUTED RECOVERY] |
  [DISCONNECT SESSION 'sid_integer, session_integer' [POST_TRANSACTION] [IMMEDIATE]] |
  [SUSPEND | RESUME]
  [KILL SESSION 'sid_integer, session integer' [IMMEDIATE]] |
  [ARCHIVE_LOG [THREAD integer]
    {[START [TO 'destination']]
     [STOP]
     [SEQUENCE integer [TO 'destination']]
     [CHANGE integer [TO 'destination']]
     [CURRENT [TO 'destination']]
     [GROUP integer [TO 'destination']]
     [LOGFILE 'filename' [TO 'destination']]
     [NEXT [TO 'destination']]
     [ALL [TO 'destination']]
    }
  }
```

对数据库实例进行动态改变。

关键字

RESOURCE_LIMIT

指定是否强制（TRUE）或者不强制（FALSE）资源限制。

GLOBAL_NAMES

指定是否强制（TRUE）或者不强制（FALSE）全局名。

MTS_SERVERS

改变共享多线程服务器（multithreaded server，MTS）进程为最小数量。

MTS_DISPATCHERS

改变命名协议的调度进程数。在发出该语句时，数据库必需是开放的。

LICENSE_MAX_SESSIONS

指定在该实例上允许的最大会话数量。值为0表示没有任何限制。

LICENSE_SESSIONS_WARNING

指定在警告信息被写到警告日志文件中之前该实例允许的最大会话数。值为0表示没有任何限制。

LICENSE_MAX_USERS

指定该数据库中允许的最大用户数。值为0指定没有限制。

ENABLE RESTRICTED SESSION

只允许有RESTRICTED SESSION权限的用户可以登录到该实例中。使用该语句时，数据库可以是没有安装的（dismounted）、安装的（mounted）、开放（opened）或关闭的（closed）。

DISABLE RESTRICTED SESSION

允许有 CREATE SESSION 权限的任何用户登录到该实例中，使用该语句时，数据库可以处于 dismounted、mounted、opened 或 closed 状态。

FLUSH SHARED POOL

从实例共享池中清除所有数据。执行该语句时，该数据库可以处于 dismounted、mounted、open 或 closed 状态。

CHECKPOINT

引起Oracle执行全局（GLOBAL）或局部（LOCAL）检查。GLOBAL为已经

打开数据库的所有实例执行检查；LOCAL 只为你连接的实例执行检查。执行该语句时，数据库可以是开放的或关闭的。

CHECK DATAFILES

验证访问的在线数据文件。GLOBAL 验证已经打开数据库的所有实例可以访问数据文件，而LOCAL验证你连接的实例可以访问数据文件。执行该语句时，数据库可以是开放的或关闭的。

SWITCH LOGFILE

引起 Oracle 切换到恢复日志文件组中。

ENABLE DISTRIBUTED RECOVERY

指定允许分布式恢复，在单进程环境下，用于初始化分布式恢复。

DISABLE DISTRIBUTED RECOVERY

指定不能进行分布式恢复。

DISCONNECT SESSION

通过关闭指定服务器进程或 MTS 虚拟电路从数据库中断开特定连接。

POST_TRANSACTION

指定断开会话之前完成正在进行的事务。

IMMEDIATE

指定立即断开会话，不等待事务完成。如果指定 POST_TRANSACTION，就忽略该关键字。

KILL SESSION

使用 V$SESSION 视图中的 SID 和 SERIAL# 立即终止会话。如果会话在等待一个行为完成，如远端数据库上的操作，则 Oracle 将要等到该行为完成为止，除非指定了 IMMEDIATE。

SUSPEND

指定所有实例的 I/O 行为应该被延缓，直到发出 ALTER SYSTEM RESUME 命令为止。执行该语句之前，所有表空间都应处于热备份模式。

RESUME

指定正常的 I/O 操作应该在 ALTER SYSTEM SUSPEND 语句之后重新再用。

ARCHIVE_LOG

手工归档恢复日志文件组或启用/禁止自动归档模式。

THREAD

指定包含被归档的恢复日志文件组的线程。只有运行在Oracle并行服务器（Oracle Parallel Server）选项的并行模式时才需要该参数。

START

启用恢复日志组的自动归档。

STOP

禁止恢复日志组的自动归档。

SEQUENCE

指定手工归档的恢复日志文件组的日志序列数。执行该语句时，数据库必须是mounted状态，但可以处于open或closed状态。

CHANGE

用整数指定的系统改变号（system change number，SCN）包含的恢复日志条目手工归档在线恢复日志文件组。如果SCN是当前日志组，就执行日志切换。使用这一参数时，数据库必须处于open状态。

CURRENT

手工强制日志切换并归档当前恢复日志文件组。使用该参数时，数据库必须处于open状态。

GROUP

用特定GROUP值手工归档在线恢复日志文件组，这可以在INIT.ORA的DBA_LOG_FILES参数中找到。使用该语句时，数据库必须处于mounted状态，但可以是open或closed状态。

LOGFILE

手工归档包含由*filename*定义的日志文件成员的在线日志文件组。使用该语句时，数据库必须处于mounted状态，但可以是open或closed状态。

NEXT

手工归档下一个满的在线恢复日志文件组，但还没有被归档。使用该语句时，数据库必须处于mounted状态，但可以是open或closed状态。

ALL

手工归档已经满的所有在线恢复日志文件组。使用该语句时，数据库必须处于 mounted 状态，但可以是 open 或 closed 状态。

注意：除了注释的以外，执行这些语句时，数据库必须处于 mounted 或 open 状态。

示例

为 TCP 改变调度进程的编号为 10，而将 DECNet 改变为 8:

```
ALTER SYSTEM
   SET MTS_DISPATCHERS = 'TCP,10'
       MTS_DISPATCHERS = 'DECnet,8';
```

通过设置最大会话数为100，强制执行Oracle许可权，达到门限80时产生告警信息:

```
ALTER SYSTEM
   SET LICENSE_MAX_SESSIONS = 100
       LICENSE_SESSIONS_WARNING = 80;
```

指定位置的归档日志序列号为 123:

```
ALTER SYSTEM
   ARCHIVE LOG SEQ 123 TO '/disk09/oracle/archive';
```

ANALYZE

要得到这一语句的完整描述，请参见第三章。

ASSOCIATE STATISTICS

```
ASSOCIATE STATISTICS WITH
 {COLUMNS [schema.]table.column[, [schema.]table.column...] |
  FUNCTIONS [schema.]function[,[schema.]function...] |
  PACKAGES [schema.]package[,[schema.]package...] |
  INDEXES [schema.]index[,[schema.]index...] |
 }
 {USING [schema.]statistics_type |
  DEFAULT COST (cpu_cost,io_cost,network_cost) |
  DEFAULT SELECTIVITY default_selectivity
 }
```

将统计计算的方法关联数据库对象。

关键字

COLUMNS

指定将要提供的列清单。

schema

包含特定表的架构。

table

包含特定列的表名。

column

关联定义的列名。

FUNCTIONS

指定关联的一个或多个函数。

PACKAGES

指定关联的一个或多个包。

INDEXES

指定关联的一个或多个索引。

USING statistics_type

指定关联的统计类型。

DEFAULT COST

指定为CPU、I/O和网络提供的默认成本。如果使用COLUMNS关键字，则该关键字无效。

cpu_cost

代表单个执行或访问的CPU成本的整数。

io_cost

代表单个执行或访问的I/O成本的整数。

network_cost

代表单个执行或访问的网络成本的整数。

DEFAULT SELECTIVITY default_selectivity

在 1 和 100 之间指定一个整数，代表作为百分比的默认选择值。如果使用 COLUMNS 关键字，则该关键字无效。

注意：要删除或断开对一个对象的统计，使用 DISASSOCIATE STATISTICS 命令。

示例

使用成本函数 cost_funct 为 emp_sal 函数创建关联：

```
ASSOCIATE STATISTICS WITH FUNCTIONS emp_sal USING cost_funct;
```

AUDIT（架构对象）

```
AUDIT {object_option[,object_option ...] | ALL}
ON {[schema.]objectname | DIRECTORY directory_name | DEFAULT}
[BY SESSION [WHENEVER [NOT] SUCCESSFUL]
[BY ACCESS [WHENEVER [NOT] SUCCESSFUL]
```

为特定架构对象设置审核。

关键字

object_option

显示被审核的特定操作。下面的操作有效：ALTER、AUDIT、COMMENT、DELETE、EXECUTE、GRANT、INDEX、INSERT、LOCK、RENAME、SELECT 和 UPDATE。关键字 ALL 指定所有操作。

object_name

指定被审核的架构对象名。

DIRECTORY directory_name

指定被审核的目录名。

DEFAULT

建立特定对象选项作为还没有被审核的对象默认值。

BY SESSION

引起 Oracle 为同一会话中相同类型的所有 SQL 语句写单条记录。

BY ACCESS

引起 Oracle 为每一条被审核的语句写单条记录。

WHENEVER SUCCESSFUL

只为成功完成的 SQL 语句选择审核。

WHENEVER NOT SUCCESSFUL

只为执行失败或导致错误的 SQL 语句选择审核。

示例

为 scott 的 bonus 表发出的 UPDATE 语句进行审核:

```
AUDIT UPDATE ON scott.bonus;
```

在 scott 的 emp 表上审核未完成的操作:

```
AUDIT ALL ON scott.emp WHENEVER NOT SUCCESSFUL;
```

AUDIT (SQL 语句)

```
AUDIT {system_option | sql_option}[,{system_option | sql_option ...}]
{[BY username[,username ...]] |
 [BY PROXY [ON BEHALF OF {ANY|username[,username]]}
}
{[BY SESSION] [WHENEVER [NOT] SUCCESSFUL] |
 [BY ACCESS] [WHENEVER [NOT] SUCCESSFUL]
}
```

为并发用户会话中的特定 SQL 语句设置审核。

关键字

system_option

指定被审核的命名系统权限授权的 SQL 语句。

sql_option

指定被审核的 SQL 语句集合。

BY username

指定审核命名用户发出的 SQL 语句。

BY proxy

指定审核特定代理人发出的SQL语句。

ON BEHALF OF ANY

指定审核代表任何用户发出的SQL语句。

ON BEHALF OF user

指定代表代理人的用户执行该特定语句。

BY SESSION

引起Oracle为相同会话中执行的相同类型的所有SQL语句写单条记录。

BY ACCESS

引起Oracle为每条审核语句写单条记录。

WHENEVER SUCCESSFUL

只为成功完成的SQL语句选择审核。

WHENEVER NOT SUCCESSFUL

只为失败或导致错误的SQL语句选择审核。

注意：审核记录被写到审核轨迹中。审核轨迹是包含了可以通过数据字典视图访问的审核记录数据库表。审核必须使用INIT.ORA文件中的AUDIT_TRAIL参数使能。

示例

为不成功的SELECT语句执行审核（这可以显示试图读一张用户没有被授权访问权限的表）：

```
AUDIT SELECT TABLE
   WHENEVER NOT SUCCESSFUL;
```

CALL

```
CALL [schema.][package.]{function | procedure}[@dblink]
  (expression[,expression...]) [INTO :host_variable [[INDICATOR] :indicator_variable]]
```

执行存储的PL/SQL函数或存储过程。

关键字

schema

指定包含包、函数或过程的架构名。

package

指定包含函数或过程的包名。

function

指定被执行的函数名。

procedure

指定被执行的过程名。

dblink

指定数据库链接名。

expression

指定函数或过程参数。

INTO

指定函数，主变量名将保留返回值。

INDICATOR

指定显示主变量条件的变量名。

COMMENT

```
COMMENT ON {TABLE [schema.]{table | view | snapshot} |
            COLUMN [schema.]{table | view | snapshot}.column
           }
IS 'text'
```

添加表、视图、快照或列的注释到数据字典中。

关键字

schema

指定具有关联注释的包含表、视图或快照的架构名。如果省略*schema*，则假定为当前架构。

TABLE

指定该注释将与表、视图或快照关联。

table

指定注释关联的表名。

view

指定注释关联的视图名。

snapshot

指定注释关联的快照名。

COLUMN

指定注释关联的列。

column

指定注释的列名。

text

注释的实际文本。该文本将记录在数据字典中。

注意：你可以使用COMMENT ON COLUMN命令删除注释。只要设置注释为空字符串（''）就可以。

示例

在scott的emp表上添加ename列的注释：

```
COMMENT ON COLUMN scott.emp.ename IS 'Last name from personnel records';
```

CREATE/ALTER/DROP CLUSTER

```
CREATE CLUSTER [schema.]clustername
  (column  datatype[, column  datatype ...])
  [Physical_Attributes_Clause]
  [SIZE integer [K | M]]
  [TABLESPACE tsname]
  [STORAGE Storage_Clause]
  [INDEX]
  [{SINGLE TABLE} HASHKEYS integer HASH IS expression]
```

```
[{PARALLEL [integer] | NOPARALLEL}]
[CACHE | NOCACHE]
```

创建簇，这是包含具有一个或多个共同列的一个或多个表的架构对象。

```
ALTER CLUSTER [schema.]clustername
[SIZE integer[K | M]]
[Physical_Attributes_Clause]
[STORAGE Storage_Clause]
[{PARALLEL [integer] | NOPARALLEL}]
[ALLOCATE EXTENT
 (EXTSIZE integer[K | M] [DATAFILE 'filename'] [INSTANCE integer])]
```

为簇重新定义将来的存储分配或分配簇的扩展盘区。

```
DROP CLUSTER [schema.]clustername
[INCLUDING TABLES]
[CASCADE CONSTRAINTS]
```

从数据库中删除簇。

关键字

cluster_name

指定簇名。

column

指定定义簇键的列名。

datatype

指定列的数据类型。

PCTUSED

指定Oracle将维护每一块使用空间的最小百分比。该值可以在0~99变化，默认值为40。

PCTFREE

改变为将来更新该簇中包含的行所保留的自由空间百分比。该值可以在0~99之间变化，默认值为10。

INITTRANS

改变簇中分配的每一块事务条目的数量。该值的变化范围是1~255，但通常采用默认值2。

MAXTRANS

改变可以更新簇块的并发事务最大数。该值可以在1~255之间变化，但通常使用默认值，这是Oracle块大小的函数。

SIZE

决定分配多少空间存储行及相同簇键。该值应该是Oracle块大小的一个约数，而且必要时可以向上取舍成下一个大约数。

TABLESPACE

指定存储该簇的表空间名。如果省略，则使用架构所有者的默认表空间。

STORAGE Storage_Clause

指定物理存储特性。请参见第四章得到更详细的信息。

INDEX

指定创建的索引簇。在散列簇中，该关键字无效。

SINGLE TABLE

指定该簇是只包含一张表的特殊类型。

HASHKEYS integer

指定该散列簇用整数散列键创建。

HASH IS expression

指定一个表达式用作散列簇的散列函数。

NOPARALLEL

指定串型创建该簇。这是默认值。

PARALLEL [integer]

指定Oracle选择并行度等于所有参与实例上可用的CPU数乘以PARALLEL_THREADS_PER_CPU初始化参数值。如果指定为整型，则显示并行度。

CACHE

执行整表扫描时，指定检索该表的块放置在最近使用最多的缓存中。

NOCACHE

执行整表扫描时，指定检索该表的块放置在最近使用最少的缓存中。这是默认值。

EXTSIZE integer

以字节、KB 或 MB 的形式指定新扩展盘区的大小。

DATAFILE

在表空间中指定包含该簇的操作系统数据文件名，将保留新的扩展盘区。如果省略，那么 Oracle 将选择数据文件。

INSTANCE

使新的扩展盘区可用于特定实例，由初始化参数INSTANCE_NUMBER指定。该参数只用在并行模式下。

INCLUDING TABLES

指定所有表属于被删除的簇。

CASCADE CONSTRAINTS

指定将要删除指向簇表主键和惟一键的簇外表的所有参照完整性约束。

注意：簇可以改进数据库性能和效率。公共列只存储一次，而且所有表的数据通常都相邻存储。

示例

在 scott 的架构中创建一个簇将为每块簇键分配 512 字节，而且每个簇最大允许 20 个扩展盘区。所有其他值都将是默认值：

```
CREATE CLUSTER demo.employee
  SIZE 512K
  STORAGE (MAXEXTENTS 20);
```

改变现存的簇，为每块簇键分配 512 字节，而且允许簇的最大扩展盘区为 20：

```
ALTER CLUSTER demo.employee
  SIZE 512K
  STORAGE (MAXEXTENTS 20);
```

从数据库中删除簇和它从数据库中包含的所有表：

```
DROP CLUSTER demo.employee INCLUDING TABLES;
```

CREATE CONTROLFILE

```
CREATE CONTROLFILE [REUSE] [SET] DATABASE dbname
LOGFILE [GROUP integer] filespec[,[GROUP integer] filespec ...]
{RESETLOGS | NORESETLOGS}
[MAXLOGFILES integer]
[MAXLOGMEMBERS integer]
[MAXLOGHISTORY integer]
[MAXDATAFILES integer]
[MAXINSTANCES integer]
[ARCHIVELOG | NOARCHIVELOG]
DATAFILE (filespec[,filespec ...])
CHARACTER SET character_set
```

重新创建控制文件，允许改变一些参数。

关键字

REUSE

指定在*INIT.ORA*中存在的一个或多个控制文件可以重复使用和修改。如果省略该关键字，而且如果在*INIT.ORA*中命名的控制文件存在，则会产生错误。

SET

指定提供的*dbname*将是数据库中的新名。有效名字的长度为1~8个字符长度。

DATABASE

指定数据库名。除非你使用SET命令，否则必须使用当前数据库名。

LOGFILE

指定所有恢复日志文件组成员，这些成员必须全部存在。

RESETLOGS

指定应该忽略LOGFILE从句中列出的日志文件内容。在LOGFILE从句中列出的每个文件都必须指定大小。

NORESETLOGS

指定在LOGFILE从句中列出的所有文件（必须是当前恢复日志文件，而且不应该从备份中恢复）应该用其原始大小重新使用。

MAXLOGFILES

指定数据库可以创建的恢复日志文件组的最大数量。默认值和最大值都由操作系统决定。这一值至少应该为2，最好是不小于3。

MAXLOGMEMBERS

指定可以在数据库中存在的恢复日志组备份的最大数量。最小值为1，默认值和最大值由操作系统决定。

MAXLOGHISTORY

为并行服务器的自动介质恢复指定归档恢复日志文件组的最大数量。该默认值和最大值都由操作系统决定。最小值为1，默认值和最大值由操作系统决定。

MAXDATAFILES

指定可以为数据库创建的最大数据文件数量。最小值为1，而且永远不能设置为比数据库中曾经创建的数据文件最大数量要小的数字。

MAXINSTANCES

指定可以安装并打开数据库的最大实例数。该参数只应用于并行服务器。

ARCHIVELOG

指定数据库将运行在归档日志模式下。

NOARCHIVELOG

指定数据库将运行在非归档日志模式下，而且将重复使用在线恢复日志文件。该选项是默认值。

DATAFILE

指定数据库中的所有数据文件名，这些文件必需全部存在。

character_set

如果与默认值不同，则指定用于创建数据库的字符集名

警告：我强烈推荐你在执行该命令之前备份整个数据库，包括控制文件和恢复日志文件。

示例

为数据库重新创建一个TEST控制文件：

```
CREATE CONTROLFILE REUSE DATABASE TEST
   LOGFILE GROUP 1 ('/disk01/oracle/log1.log','/disk02/oracle/log1.log')
           GROUP 2 ('/disk03/oracle/log2.log','/disk04/oracle/log2.log')
   NORESETLOGS
```

```
DATAFILE '/disk10/oradata/db01.dbs'
MAXLOGFILES 6
MAXDATAFILES 128
ARCHIVELOG;
```

CREATE/ALTER DATABASE

```
CREATE DATABASE [dbname]
CONTROLFILE [REUSE]
LOGFILE [GROUP integer] (filespec[,[GROUP integer] filespec ...])
[MAXLOGFILES integer]
[MAXLOGMEMBERS integer]
[MAXLOGHISTORY integer]
[MAXDATAFILES integer]
[MAXINSTANCES integer]
[ARCHIVELOG | NOARCHIVELOG]
[CHARACTER SET charset
[NATIONAL CHARACTER SET charset
DATAFILE (filespec[,filespec ...])[Autoextend_Clause]
```

创建数据库并指定与其关联的参数。

```
ALTER DATABASE [dbname]
  {ARCHIVELOG | NOARCHIVELOG |
   MOUNT [[STANDBY | CLONE] DATABASE] |
   CONVERT |
   OPEN [READ WRITE [RESETLOGS | NORESETLOGS] | READ ONLY] |
   ACTIVATE STANDBY DATABASE |
   RENAME FILE 'filename' [,'filename' ...] TO 'filename' [,'filename' ...]  |
   RENAME GLOBAL_NAME TO database[.domain[.domain ...]] |
   RESET COMPATIBILITY |
   ENABLE [PUBLIC] THREAD integer |
   DISABLE THREAD integer |
   CHARACTER SET character_set |
   NATIONAL CHARACTER SET character_set |
   CREATE DATAFILE 'filename'[,'filename' ...] [AS filespec] |
   DATAFILE 'filename'[,'filename' ...]
     {ONLINE |
      OFFLINE [DROP] |
      RESIZE integer[K | M] |
      END BACKUP |
      Autoextend_Clause
     } |
   TEMPFILE 'filename'[,'filename' ...]
     {ONLINE |
      OFFLINE
      DROP |
      RESIZE integer[K | M] |
      Autoextend_Clause
     } |
```

```
ADD LOGFILE [THREAD integer]
 [GROUP integer] filespec[,[GROUP integer] filespec ...] |
ADD LOGFILE MEMBER 'filename' [RESUSE] [,'filename' [RESUSE] ...]
 TO logfile_descriptor[,logfile_descriptor ...] |
DROP {GROUP integer | 'filename' | ('filename','filename'[,'filename' ...])} |
DROP LOGFILE MEMBER 'filename'[,'filename' ...] |
CREATE STANDBY CONTROLFILE AS 'filename' [REUSE] |
BACKUP CONTROLFILE TO {'filename' [REUSE] | TRACE {RESETLOGS | NORESETLOGS}} |
RECOVER [AUTOMATIC FROM 'location']
RECOVER [STANDBY] DATABASE [UNTIL {CANCEL | TIME date | CHANGE integer}] |
RECOVER [STANDBY] DATABASE USING BACKUP CONTROLFILE |
RECOVER [STANDBY] TABLESPACE {tablespace | DATAFILE 'filename'}
  UNTIL [CONSISTENT WITH] CONTROLFILE |
RECOVER TABLESPACE tablespace[,tablespace ...] |
RECOVER DATAFILE filename[,filename] |
RECOVER LOGFILE filename[,filename] |
RECOVER CONTINUE [DEFAULT] |
RECOVER CANCEL |
RECOVER MANAGED STANDBY DATABASE {TIMEOUT integer | CANCEL [IMMEDIATE]}
```

改变现存数据库的一个或多个特性。

关键字

dbname

指定数据库名。可以是1~8个字符长度，但不能是保留字。

CONTROL FILE REUSE

指定在*INIT.ORA*中指定的一个或多个存在的控制文件可以被重复使用或修改。如果省略该关键字且在*INIT.ORA*中命名的控制文件存在，则会产生错误。如果指定的参数要求控制文件比当前大小要大，则该命令将失败。注意该选项通常不用于创建新数据库。

LOGFILE

指定被创建的一个或多个恢复日志文件名。

MAXLOGFILES

指定曾经为数据库创建的最大恢复日志文件组数。默认值和最大值由操作系统决定，该值必须至少为2，最好不小于3。

MAXLOGMEMBERS

指定数据库中可以存在的恢复日志组备份的最大数量。最小值为1，默认值和最大值由操作系统决定。

MAXLOGHISTORY

指定并行服务器自动介质恢复的归档恢复日志文件组最大数量。最小值为1，默认值和最大值由操作系统决定。

MAXDATAFILES

指定曾经为数据库创建的数据文件最大数量。最小值为1，而且永远不能设置为小于数据库中曾经创建的数据文件最大数量。

MAXINSTANCES

指定能让数据库安装并打开的最大实例数。该参数只应用于并行服务器。

ARCHIVELOG

指定数据库将运行在归档日志下，这意味着在重新使用恢复归档日志组之前必须对恢复日志组归档。如果该组还没有被归档，则数据库将会中断，直到成功归档为止。该模式要求执行介质恢复。

NOARCHIVELOG

指定恢复日志组将不被归档，而且可以被Oracle立即重复使用。该选项是默认值。

CHARACTER SET

指定数据库用于存储数据的字符集。该字符集不能在数据库创建之后改变。该选择值和默认值都由操作系统决定。

NATIONAL CHARACTER SET

指定用作特定列的国际字符集。如果省略，就使用默认数据库字符集。

DATAFILE

指定数据库的所有数据文件名。如果省略，则默认为系统表空间创建单个数据文件。

Autoextend_Clause

指定数据文件的空间是否自动扩展。请参见第四章得到更详细的信息。

MOUNT STANDBY DATABASE

指定安装备用数据库。

MOUNT CLONE DATABASE

指定安装克隆数据库。

CONVERT

指定数据库的数据字典从 Oracle7 转换为 Oracle8 或 Oracle8*i*。

OPEN READ WRITE

指定数据库以读/写方式打开。该选项是默认值。

RESETLOGS

将日志序列号设置为1，并使所有现存在线和归档日志文件的所有恢复条目都无效。只有执行了不完全介质恢复或用备份控制文件执行介质恢复后打开数据库时，才应该指定该选项；否则就应该使用 NORESETLOGS。如果用 RESETLOGS 关键字打开数据库，就应该立即执行数据库完全备份。

NORESETLOGS

不改变当前日志序列号和恢复日志条目的状态。

OPEN READ ONLY

指定以只读方式打开数据库，这使查询成为可能，但不能执行写操作。

ACTIVATE STANDBY DATABASE

指定数据库从备用状态改变为激活状态。

RENAME FILE filename1 TO filename2

指定改变控制文件中的数据文件名、临时文件名或日志文件名。注意，该关键字不影响操作系统文件名。

RENAME GLOBAL NAME TO

指定改变数据库的全局名为提供的值，最多不超过 8 个字符。

RESET COMPATIBILITY

指定复位数据库兼容性为指定版本。该变化在下次启动数据库时生效。

ENABLE THREAD

使并行服务器环境下能执行恢复日志文件线程。如果指定关键字 PUBLIC，则允许的线程可用于任何实例。否则，该线程只可用于特定要求的实例。

DISABLE THREAD

指定恢复日志文件的线程无效，并使并行服务器环境的所有实例都无效。

CREATE DATAFILE

指定创建一个新的空数据文件代替老的数据文件（可以已经丢失而且没有备份）。在可以使用数据文件之前，必须执行介质恢复。

ONLINE

指定将数据文件带入在线状态。

OFFLINE

指定将数据文件带入离线状态。

RESIZE

指定数据文件的大小增加或减少为显示的大小。

END BACKUP

指定中断热备份后启动数据库时不执行介质恢复。

TEMPFILE

指定在临时数据文件中改动。

DROP

指定从数据库中删除临时数据文件。

ADD LOGFILE

指定添加一个或多个恢复日志文件组。在并行服务器环境下可以显示线程。

ADD LOGFILE MEMBER

指定在现存恢复日志文件组中添加新成员文件名。可以使用REUSE显示已经存在的文件名。

logfile_descriptor

指定现存的恢复日志文件组要么是分组整数，要么是文件名列表。

DROP GROUP

指定执行ALTER SYSTEM SWITCH LOGFILE语句后删除整个恢复日志文件组。删除组可以用分组整数或文件名列表指定。

DROP LOGFILE MEMBER filename

指定删除一个或多个单个的恢复日志文件组成员。

CREATE STANDBY CONTROL FILE

指定创建一个控制文件维护备用数据库。

BACKUP CONTROL FILE TO

指定备份当前控制文件为显示的文件名。如果指定关键字TO TRACE，则SQL语句集合将写入跟踪文件而不是创建控制文件的备份。如果指定

RESETLOGS，则所写的SQL语句将包括ALTER DATABASE OPEN RESETLOGS。如果指定NORESETLOGS，则会写ALTER DATABASE OPEN NORESETLOGS这样的SQL语句。

RECOVER FROM

指定恢复时要读的归档恢复日志文件的位置。如果指定了关键字AUTOMATIC，则会使用LOG_ARCHIVE_DEST和LOG_ARCHIVE_FORMAT初始化参数产生下一个需要恢复的归档日志文件名。

RECOVER DATABASE

指定恢复整个数据库。如果使用STANDBY关键字，则使用主数据库的控制文件和归档恢复日志文件恢复备用数据库。可以使用关键字UNTIL CANCELLED，指定该数据库恢复到使用RECOVER CANCEL从句取消操作为止。TIME关键字指定了基于时间恢复到提供的时间为止。CHANGE关键字显示了在指定系统改变数字之前执行恢复。

RECOVER DATABASE USING BACKUP CONTROL FILE

指定使用备份控制文件（而不是当前控制文件）。

RECOVER STANDBY TABLESPACE UNTIL CONSISTENT WITH CONTROL FILE

指定从当前备用数据库上使用控制文件恢复老的备用数据库表空间。

RECOVER STANDBY DATAFILE UNTIL CONSISTENT WITH CONTROL FILE

指定从当前备用数据库使用控制文件恢复老的备用数据库数据文件。

RECOVER TABLESPACE

指定要被恢复的一个或多个表空间（必须处于离线状态）。

RECOVER DATAFILE

指定要被恢复的一个或多个数据文件（必须处于离线状态）。

RECOVER LOGFILE

指定介质恢复应该继续使用提供的日志文件。

RECOVER CONTINUE DEFAULT

指定恢复将继续使用应该自动产生的恢复日志文件。除了没有提示文件名存在以外，这里的关键字就等于RECOVER AUTOMATIC。

RECOVER CONTINUE

指定中断多实例恢复以使一个线程无效之后，应继续运行多实例恢复。

RECOVER CANCEL

指定基于取消的恢复结束。

RECOVER MANAGED STANDBY DATABASE

指定应该使用持续不变的备用恢复模式发生恢复，这假设备用数据库是一个活跃组件。

TIMEOUT

指定要求的恢复日志文件可以写入备用数据库时所等待的分钟数。

CANCEL

指定已经提供了当前归档恢复日志文件后结束持续恢复，除非使用了IMMEDIATE关键字，它将在应用指定的下一个恢复日志文件读以后终止。

示例

创建称为TEST的一个新数据库：

```
CREATE DATABASE TEST
LOGFILE GROUP 1 ('/disk01/oracle/log1.log','/disk02/oracle/log2.log')
                SIZE 50K,
        GROUP 2 ('/disk03/oracle/log2.log','/disk04/oracle/log2.log')
                SIZE 50K
MAXLOGIFLES 5
DATAFILE '/disk10/oradata/system01.dbf' SIZE 50M
MAXDATAFILES 100
ACHIVELOG;
```

CREATE/DROP DATABASE LINK

```
CREATE [SHARED] [PUBLIC] DATABASE LINK dblink
   [CONNECT TO {CURRENT USER |
                username IDENTIFIED BY password
                  [AUTHENTICATED BY username IDENTIFIED BY password]
               }]
   [USING 'connect string']
```

创建数据库链接，允许访问远端数据库上的对象。

```
DROP [PUBLIC] DATABASE LINK dblink
```

从数据库中删除数据库链接。

关键字

dblink

指定被创建的数据库链接名。必须是有效的 Oracle 对象名。

SHARED

在配置多线程服务器时通过用户共享单网络连接。

PUBLIC

指定数据库链接可用于所有用户。如果省略，则认为该数据库链接是私有的，而且只对使用者可用。

CONNECT TO CURRENT USER

指定创建当前用户数据库链接，这需要在远端数据库上有一个有效账户的全局用户。

CONNECT TO username

指定用于链接远端数据库的用户名和口令。

AUTHENTICATED BY

当使用 SHARED 关键字时，指定远端数据库上的用户名和口令用于鉴定。

USING

为远端数据库指定 Net8 数据库规范。

注意：如果省略CONNECT TO从句，那么在调用数据库链接时将使用当前登录的账户用户名和口令，而不使用创建者用户名和口令。

示例

创建 TEST 数据库上 scott 账户的公用数据库链接：

```
CREATE PUBLIC DATABASE LINK testscott
CONNECT TO scott IDENTIFIED BY tiger
USING 'TEST';
```

具有TEST数据库访问权限的本地数据库用户，现在可以访问scott的任何对象。例如，要从远端数据库上的 emp 表选择，你可以使用如下的 SQL 语句：

```
SELECT * FROM emp@testscott;
```

从数据库中删除公用数据库链接命名的testscott:

```
DROP PUBLIC DATABASE LINK testscott;
```

CREATE/DROP DIMENSION

```
CREATE [FORCE | NOFORCE] DIMENSION [schema.]dimension
LEVEL level IS {level_table.level_column |
               {level_table.level_column,level_table.level_column ...}}
HIERARCHY hierarchy (child level CHILD OF parent_level
  [JOIN KEY {child_key_column | (child_key_column,child_key_column ...)}
   REFERENCES parent_level] |
ATTRIBUTE level DETERMINES {dependent_column | (dependent_column,
                dependent_column ...)}
```

创建维度，这定义了在一对列集合之间的父子关系。

```
DROP DIMENSION [schema.]dimension
```

从数据库中删除维度。

关键字

FORCE

指定被创建的维度，即使被引用的表不存在也创建。

NOFORCE

指定仅在被引用的对象存在（默认情况）时创建维度。

schema

在该架构名中创建维度。

dimension

维度名。

LEVEL

定义带有名称的层次，该名称定义了维度等级和属性。

level_table.level_column

指定列的层次（最大为32）。

HIERARCHY

指定层次名。

child_level

与父层具有 *n* 对 1 的关系的层次名。

CHILD OF

指定父层名。

JOIN KEY

指定与父表连接的 JOIN 条件列名。

REFERENCES

指定父层名。

ATTRIBUTE

指定层次或等级名。

DETERMINES dependent_column

指定依赖于属性层的列名。

示例

创建 city、state 和 country 表上的一个维度:

```
CREATE DIMENSION location
  LEVEL city_code     IS (city.city, city.state)
  LEVEL state_code    IS state.state
  LEVEL country_code  IS country.country
HIERARCHY region (
  City_code     CHILD OF
  State_code    CHILD OF
  Country_code
    JOIN KEY city.state REFERENCES state_code
    JOIN KEY state.country REFERENCES country_code;
```

从数据库中删除命名位置的维度:

```
DROP DIMENSION location;
```

CREATE/DROP DIRECTORY

CREATE [OR REPLACE] DIRECTORY *directory_name* AS *'path_name'*

创建为了存储 BFILE 对象指定操作系统目录的目录对象。

```
DROP DIRECTORY directory_name
```

从数据库中删除目录对象。

关键字

OR REPLACE

指定该目录对象应该代替任何具有相同名字的现存目录对象。

directory_name

目录对象名。

path_name

操作系统目录的完全区分大小写的路径名。

警告：Oracle 不检查主操作系统上是否实际存在目录，所以要仔细检查大家提供的路径。

示例

在主磁盘存储系统上，创建一个目录对象 graphic_home，Oracle 应用程序将用其存储图像：

```
CREATE DIRECTORY graphic_home AS '/disk13/data/graphics';
```

删除目录对象 graphic_home:

```
DROP DIRECTORY graphic_home;
```

CREATE/ALTER/DROP FUNCTION

请参见第七章得到这些语句的完全描述。

CREATE/ALTER/DROP INDEX

```
CREATE [UNIQUE | BITMAP] INDEX [schema.]indexname
ON ([schema.]TABLE [alias] ({column|col_expr} [ASC | DESC]
```

```
                                [,{column|col_expr} [ASC | DESC] ...]) |
    CLUSTER [schema.]cluster
   }
[{{GLOBAL PARTITION BY RANGE (column_list)
     (Global_Partition_Clause[,Global_Partition_Clause...])
   [TABLESPACE {tablespace | DEFAULT}]
   [Physical_Attributes_Clause]
   [STORAGE Storage_Clause]
   [NOSORT | REVERSE]
   [UNRECOVERABLE | LOGGING | NOLOGGING]
   [ONLINE]
   [COMPUTE STATISTICS]
   [COMPRESS integer | NOCOMPRESS]
   [PARALLEL integer | NOPARALLEL] ] |
  [LOCAL (PARTITION partition
   [Physical_Attributes_Clause] [TABLESPACE tablespace] [LOGGING | NOLOGGING]
   [,PARTITION partition
   [Physical_Attributes_Clause] [TABLESPACE tablespace] [LOGGING | NOLOGGING] ...])
   [TABLESPACE {tablespace | DEFAULT}
   [Physical_Attributes_Clause]
  [STORAGE Storage_Clause]
  [NOSORT | REVERSE]
  [UNRECOVERABLE | LOGGING | NOLOGGING]
  [ONLINE]
  [COMPUTE STATISTICS]
  [COMPRESS integer | NOCOMPRESS]
  [PARALLEL integer | NOPARALLEL] ] |
[LOCAL {STORE IN {tablespace[,tablespace...]|DEFAULT} |
                  (PARTITION [partition] [TABLESPACE tablespace]
                   [,PARTITION [partition] [TABLESPACE tablespace] ...])
       }
  [TABLESPACE {tablespace | DEFAULT}
  [Physical_Attributes_Clause]
  [STORAGE Storage_Clause]
  [NOSORT | REVERSE]
  [UNRECOVERABLE | LOGGING | NOLOGGING]
  [ONLINE]
  [COMPUTE STATISTICS]
  [COMPRESS integer | NOCOMPRESS]
  [NOSORT | REVERSE]
  [PARALLEL integer | NOPARALLEL] ] |
[LOCAL {STORE IN {tablespace[,tablespace ...] | DEFAULT} |
        (PARTITION [partition]
           [Physical_Attributes_Clause]
           [TABLESPACE tablespace]
           {STORE IN {tablespace[,tablespace ...]|DEFAULT} |
                      (SUBPARTITION subpartition [TABLESPACE tablespace]
                      [,SUBPARTITION subpartition [TABLESPACE tablespace] ...]))
                     }
       }
    [TABLESPACE {tablespace | DEFAULT}
    [Physical_Attributes_Clause]
    [STORAGE Storage_Clause]
```

```
    [NOSORT | REVERSE]
    [UNRECOVERABLE | LOGGING | NOLOGGING]
    [ONLINE]
    [COMPUTE STATISTICS]
    [COMPRESS integer | NOCOMPRESS]
    [PARALLEL integer | NOPARALLEL]
}]
```

创建一张表或簇上的一列或多列的索引。

```
ALTER INDEX [schema.]indexname
[DEALLOCATE UNUSED [KEEP integer[K | M]]
[ALLOCATE EXTENT (
  [SIZE integer[K | M]]
  [DATAFILE 'filename']
  [INSTANCE integer]])]
[PARALLEL integer | NOPARALLEL]
[Physical_Attributes_Clause]
[STORAGE Storage_Clause]
[UNRECOVERABLE | LOGGING | NOLOGGING]
[{REBUILD [{PARTITION partition | SUBPARTITION subpartition}]
   [PARALLEL integer | NOPARALLEL]
   [TABLESPACE tablespace]
   [ONLINE]
   [Physical_Attributes_Clause]
   [COMPRESS integer | NOCOMPRESS]
   [LOGGING | NOLOGGING] ] |
  REBUILD [REVERSE | NOREVERSE] |
  REBUILD PARAMETERS ('rebuild_parameters')
 }]
[PARAMETERS ('alter_parameters')]
[ENABLE | DISABLE]
[UNUSABLE]
[RENAME TO new_index_name]
[COALESCE]
[Partition_Clause]
```

改变索引特性。

```
DROP INDEX [schema.]indexname
```

从数据库中删除索引。

关键字

UNIQUE

指定基于索引的列值必须惟一。

BITMAP

指定索引被创建为位图，而不是用于正常的B-树结构。

ON CLUSTER

指定簇（不能是散列簇）上的嵌入式索引，而且指定簇名。

ON TABLE

指定索引建立在表上并指定表名。

alias

指定建立索引的表的别名。如果索引引用任何对象类型属性或对象类型方法，则需要该选项。

ASC

指定索引应该基于数据库字符集字符的值以升序方式创建。

DESC

指定索引应该基于数据库字符集字符的值以降序方式创建。

Physical_Attributes_Clause

指定该索引的物理属性。要了解详细信息请参见第四章。

STORAGE

指定物理存储特性。要了解详细信息请参见第四章。

TABLESPACE

指定存储该索引的表空间名。如果省略，就为架构所有者使用默认表空间。

UNRECOVERABLE

指定创建索引时不写恢复日志记录。该选项加速了索引的创建，但在数据库失败的情况下，该记录将不能用日志文件恢复记录，而必须重新创建索引。该关键字是Oracle7的语法，等于Oracle8或Oracle8*i*中指定的NOLOGGING。

LOGGING

指定创建索引时写恢复日志，该选项是默认值。

NOLOGGING

指定创建索引时不写恢复日志记录。在数据库失败的情况下，该记录不能用日志文件恢复记录，而必须重新创建索引。该选项加速了索引的创建。

ONLINE

指定创建索引时可以在索引表上执行DML操作。

COMPUTE STATISTICS

指定创建索引时计算统计信息并插入到数据字典中。

COMPRESS

指定能够进行关键压缩。

NOCOMPRESS

指定不能进行关键压缩。该选项是默认值。

NOSORT

指定以升序的方式加载被索引的行，而且在创建索引时不必排序。

REVERSE

以相反顺序指定索引块存储的字节(ROWID除外)。该关键字不能与NOSORT一起使用。

PARALLEL

指定Oracle以并行方式创建索引，选择并行度等于在所有参与实例上可用的CPU数量乘以PARALLEL_THREADS_PER_CPU初始化参数值。如果指定为整数，那么就指定了并行度。

NOPARALLEL

指定以串行的方式创建索引。该选项是默认值。

GLOBAL PARTITION BY RANGE

指定全局索引将从指定列的范围值中被分区。

column_list

指定分区索引的列名。

Global_Partition_Clause

指定单个分区特性。请参见第四章了解详细信息。

LOCAL

在索引建立的表上指定索引在相同列、相同分区数量和相同分区边界上进行分区。

PARTITION

指定单个分区名，必须等于表分区的数量，而且以相同顺序提供。

LOCAL STORE IN

指定怎样在表空间上分布散列分区或索引子分区。

DEFAULT

在散列或复合分区的表上指定局部索引，将不考虑索引层上指定的表空间，而且将使用表上驻留的相同分区或子分区。

SUBPARTITION

指定子分区名。

DEALLOCATE UNUSED

指定释放索引尾部没有使用的空间，并使其在数据库中可以被其他用户使用。

KEEP

指定在高水平标记之上的字节数量，应该在消除分配后保留为索引的一部分。

ALLOCATE EXTENT

指定该索引定位在新的扩展盘区上。

SIZE

指定分配的扩展盘区大小。

DATAFILE

指定包含新扩展盘区的数据文件名。如果省略，Oracle就为该索引从可用的数据文件中选择一个数据文件。

INSTANCE

指定新扩展盘区只可用于该实例。如果省略，则该扩展盘区可用于所有实例。

REBUILD PARTITION

指定重新创建索引分区。

REBUILD SUBPARTITION

指定重新创建索引子分区。

REBUILD REVERSE

重新建立索引时指定以相反顺序存储索引块字节（不包含ROWID）。

REBUILD NOREVERSE

重新建立索引时指定以正常顺序存储索引块字节（不包含ROWID）。

rebuild_ parameters

指定为重建全局索引传递的索引类型程序的参数字符串。

alter_ parameters

指定更改全局索引时传递的索引类型程序的参数字符串。

ENABLE

指定基于函数的索引可以使用。

DISABLE

指定禁用基于函数的索引。

UNUSABLE

指定索引被标记为不可使用。

RENAME TO

指定索引被重命名。

COALESCE

指定索引块的上下文被合并为可以重新使用的自由块。

注意：如果省略存储选项，则Oracle将为索引分配存储如下：

- 如果索引表没有行，则使用表空间的默认存储值。
- 如果索引表有行，而且结果索引包含的数据块不超过25个，则为该索引分配单个扩展盘区。
- 如果索引表有行，而且结果索引超过25个数据块，则为该索引分配5个相等大小的扩展盘区。

示例

在scott的emp表的empno和ename列上创建索引，带有显示存储参数：

```
CREATE INDEX emp_ndx ON scott.emp(empno,ename)
   STORAGE (INITIAL 50K NEXT 10K PCTINCREASE 0 MAXEXTENTS 10)
```

```
        TABLESPACE users
        PCTFREE 20;
```

更改scott拥有的emp_ndx 索引，所以每一个为该索引添加的新扩展盘区为都是4K，而且不会增加。还指定了在该索引上添加的每个数据块将包含5个初始化事务条目：

```
ALTER INDEX scott.emp_ndx
            INITTRANS 5
            STORAGE (NEXT 4096 PCTINCREASE 0);
```

从数据库的 scott 架构中删除 emp_ndx 索引。在删除索引时，该索引以前占用的所有空间都返回到自由空间池中。

```
DROP INDEX scott.emp_ndx;
```

CREATE/ALTER/DROP MATERIALIZED VIEW

```
CREATE MATERIALIZED VIEW [schema.]materialized_view_name
  [Physical_Attributes_Clause]
  [TABLESPACE tablespace]
  [STORAGE Storage_Clause]
  [REFRESH
     [FAST | COMPLETE | FORCE]
     [START WITH date]
     [NEXTREF date]
AS materialized_view_query
```

创建物质化的视图（也称为快照），这是在一张或多张表或视图上运行查询的结果。

```
ALTER MATERIALIZED VIEW [schema.]materialized_view_name
  [Physical_Attributes_Clause]
  [STORAGE Storage_Clause]
  [REFRESH
     [FAST | COMPLETE | FORCE]
     [START WITH date]
     [NEXTREF date]
```

改变物化视图（或快照）的存储或自动刷新特性。

```
DROP MATERIALIZED VIEW [schema.]materialized_view_name
```

从数据库中删除物化视图（或快照）。

关键字

TABLESPACE

指定创建物化视图的表空间名。架构所有者的默认表空间是默认值。

STORAGE

指定物理存储特性。请参见第四章了解详细信息。

REFRESH

指定自动刷新的时间和模式。FAST 意味着物化视图日志与主表相关联；COMPLETE 意味着通过重新执行物化视图的查询进行刷新；FORCE 是默认值，而且意味着Oracle将决定如果可能，则使用FAST刷新，如果不可能，就进行COMPLETE刷新。

START WITH

使用标准Oracle日期表达式指定下一个自动刷新时间的日期。

NEXTREF

指定新的日期表达式计算自动刷新之间的间隔。

AS materialized_view_query

提供操纵物化视图实际使用的SQL查询，而且服从视图的相同限制。

注意：在试图创建物化视图之前，必须用SYS运行 *dbmssnap.sql* 脚本。

警告：因为在架构中创建物化视图对象时，Oracle添加了7个字符定义了快照名，所以，大家应该限制物化视图名为27个字符或更少。

示例

创建scott的emp表的物化视图，放在名为uk的服务器上。物化视图将在明天移出，它从今天开始7天后刷新一次，此后每7天刷新一次：

```
CREATE MATERIALIZED VIEW uk_emp
   REFRESH COMPLETE
   START WITH SYSDATE + 1
   NEXT SYSDATE + 7
```

```
    AS SELECT * FROM scott.emp@UK;
```

计划 scott 用户的物化视图 dept_snap 明天午夜刷新，然后每周刷新：

```
ALTER MATERIALIZED VIEW scott.dept_snap
   REFRESH COMPLETE
   START WITH TRUNC(SYSDATE + 1)
   NEXT TRUNC(SYSDATE + 7);
```

从数据库中删除物化视图 uk_emp：

```
DROP MATERIALIZED VIEW uk_emp;
```

CREATE/ALTER/DROP MATERIALIZED VIEW LOG

```
CREATE MATERIALIZED VIEW LOG ON [schema.]tablenamc
  [Physical_Attributes_Clause]
  [TABLESPACE tablespace]
  [STORAGE Storage_Clause]
```

创建物化视图日志，这是一张与物化视图的主表关联的用于控制物化视图刷新的表。

```
ALTER MATERIALIZED VIEW LOG ON [schema.]tablename
  [Physical_Attributes_Clause]
  [STORAGE Storage_Clause]
```

改变物化视图日志的存储特性。

```
DROP MATERIALIZED VIEW LOG ON [schema.]table_name
```

从数据库中删除物化视图日志。

关键字

tablename

指定维护的物化视图的表的名称。

Physical_Attributes_Clause

指定该物化视图日志的物理特性。请参见第四章了解详细信息。

TABLESPACE

指定将在哪个表空间名中创建物化视图日志。架构所有者的默认表空间是默认值。

STORAGE

指定物理存储特性。请参见第四章了解详细信息。

示例

在 scott 的 emp 表上创建物化视图日志:

```
CREATE MATERIALIZED VIEW LOG scott.emp
   STORAGE (INITIAL 50K NEXT 50K PCTINCREASE 0)
   TABLESPACE USERS;
```

将 scott 的 emp 表上的物化视图日志的下一个扩展盘区大小改变为 50K:

```
ALTER MATERIALIZED VIEW LOG scott.emp
   STORAGE (NEXT 50K);
```

从数据库中删除与 scott 的 emp 相关的物化视图日志:

```
DROP MATERIALIZED VIEW LOG ON scott.emp;
```

CREATE/ALTER/DROP PACKAGE

请参见第七章了解这些命令的完整描述。

CREATE/ALTER/DROP PROCEDURE

请参见第七章了解这些命令的完整描述。

CREATE/ALTER/DROP PROFILE

```
CREATE PROFILE profile_name LIMIT
  [SESSIONS_PER_USER  integer | UNLIMITED | DEFAULT]
  [CPU_PER_SESSION  integer | UNLIMITED | DEFAULT]
  [CPU_PER_CALL  integer | UNLIMITED | DEFAULT]
  [CONNECT_TIME  integer | UNLIMITED | DEFAULT]
  [IDLE_TIME  integer | UNLIMITED | DEFAULT]
  [LOGICAL_READS_PER_SESSION  integer | UNLIMITED | DEFAULT]
  [LOGICAL_READS_PER_CALL  integer | UNLIMITED | DEFAULT]
  [PRIVATE_SGA  {integer [K | M] | UNLIMITED | DEFAULT}]
  [COMPOSITE_LIMIT  {integer | UNLIMITED | DEFAULT}]
  [FAILED_LOGIN_ATTEMPTS expression | UNLIMITED | DEFAULT]
  [PASSWORD_LIFE_TIME expression | UNLIMITED | DEFAULT]
  [PASSWORD_REUSE_TIME expression | UNLIMITED | DEFAULT]
```

```
  [PASSWORD_LOCK_TIME expression | UNLIMITED | DEFAULT]
  [PASSWORD_GRACE_TIME expression | UNLIMITED | DEFAULT]
  [PASSWORD_VERIFY_FUNCTION function | NULL | DEFAULT]
```

创建 profile，在数据库资源上设置限制。

```
ALTER PROFILE profile_name LIMIT
  [SESSIONS_PER_USER  integer | UNLIMITED | DEFAULT]
  [CPU_PER_SESSION  integer | UNLIMITED | DEFAULT]
  [CPU_PER_CALL  integer | UNLIMITED | DEFAULT]
  [CONNECT_TIME  integer | UNLIMITED | DEFAULT]
  [IDLE_TIME  integer | UNLIMITED | DEFAULT]
  [LOGICAL_READS_PER_SESSION  integer | UNLIMITED | DEFAULT]
  [LOGICAL_READS_PER_CALL  integer | UNLIMITED | DEFAULT]
  [PRIVATE_SGA  integer [K | M] | UNLIMITED | DEFAULT]
  [COMPOSITE_LIMIT  integer | UNLIMITED | DEFAULT]
  [FAILED_LOGIN_ATTEMPTS expression | UNLIMITED | DEFAULT]
  [PASSWORD_LIFE_TIME expression | UNLIMITED | DEFAULT]
  [PASSWORD_REUSE_TIME expression | UNLIMITED | DEFAULT]
  [PASSWORD_LOCK_TIME expression | UNLIMITED | DEFAULT]
  [PASSWORD_GRACE_TIME expression | UNLIMITED | DEFAULT]
  [PASSWORD_VERIFY_FUNCTION function | NULL | DEFAULT]
```

从现有的 profile 中添加、改变或删除资源限制。

```
DROP PROFILE profile_name [CASCADE]
```

从数据库中删除 profile。

关键字

profile_name

要被创建的 profile 名。

SESSIONS_PER_USER

为用户限制并发会话数量。

CPU_PER_SESSION

限制可以用在会话中的 CPU 时间数量（以百分之一秒计）。

CPU_PER_CALL

限制调用（分析、执行或检索）所占的 CPU 时间数量（以百分之一秒计）。

CONNECT_TIME

限制整个会话消耗的时间（以分钟计）。

IDLE_TIME

限制会话中持续不活跃的时间数（以分钟计）。

LOGICAL_READS_PER_SESSION

限制会话中数据库块读的数量，包括内存读和磁盘读。

LOGICAL_READS_PER_CALL

限制调用（分析、执行或提取）的数据库块数量。

PRIVATE_SGA

限制在SGA共享池中一个会话可以分配的内存数（以字节计）。

COMPOSITE_LIMIT

限制会话的整个资源开销（以服务单元计）。要得到额外信息，参见本章前面的ALTER RESOURCE COST。

UNLIMITED

指定该值意味着在该资源上不强加任何限制。

DEFAULT

指定该值意味着默认profile中指定的限制将用于该资源上。

FAILED_LOGIN_ATTEMPTS

在给账户加锁之前，指定允许尝试登录失败的次数。

PASSWORD_LIFE_TIME

指定口令期满而且必须被改变前可以使用的天数。

PASSWORD_REUSE_TIME

指定以前使用的口令可以被重新使用之前的天数。如果设置为整数，PASSWORD_REUSE_MAX就必须设置为UNLIMITED。

PASSWORD_REUSE_MAX

指定当前口令可以重新使用之前需要的口令改变的次数。如果设置为整数，则PASSWORD_REUSE_TIME必须设置为UNLIMITED。

PASSWORD_LOCK_TIME

指定超过了FAILED_LOGIN_ATTEMPTS限制之后，账户仍然保留被加锁的天数。

PASSWORD_GRACE_TIME

指定口令有效期后允许登录时含有警告信息的天数。

PASSWORD_VERIFY_FUNCTION

指定用于校验口令的PL/SQL函数名。设置该参数为NULL表示不执行校验。要给指定用户应用与profile相关的限制，就必须使用CREATE USER或ALTER USER命令给用户分配profile。资源限制也必须通过*INIT.ORA*的RESOURCE_LIMIT参数或使用ALTER SYSTEM处于被允许状态。

示例

为adminprofile定义并发会话数量最大为5，而且不活跃时间为10分钟:

```
CREATE PROFILE admin
   SESSIONS_PER_USER 5
   IDLE_TIME 10;
```

为adminprofile重新定义并发会话数量最大为10，而且不活跃时间为15分钟:

```
ALTER PROFILE admin
   SESSIONS_PER_USER 10
   IDLE_TIME 15;
```

从数据库中删除adminprofile。

```
DROP PROFILE admin;
```

CREATE/ALTER/DROP ROLE

```
CREATE ROLE rolename
  [NOT IDENTIFIED | IDENTIFIED BY password | IDENTIFIED EXTERNALLY]
```

创建角色，该角色是一组可以授给用户的权限集合。

```
ALTER ROLE rolename
  [NOT IDENTIFIED | IDENTIFIED BY password | IDENTIFIED EXTERNALLY]
```

改变授予一个角色权利需要的授权层。

```
DROP ROLE rolename
```

从数据库中删除角色。

关键字

rolename

被创建的角色名。

NOT IDENTIFIED

指定在允许时不需要验证被授予角色权限的用户。

IDENTIFIED BY

在角色被允许时，指定必须提供口令。

IDENTIFIED EXTERNALLY

指定操作系统验证授予角色的用户。

注意：创建角色时，大家自动给角色授予了WITH ADMIN OPTION权限，这允许大家授权或调用角色或使用ALTER ROLE命令修改它。

示例

创建一个称为manager的角色并分配口令为‘dilbert’：

```
CREATE ROLE manager IDENTIFIED BY dilbert;
```

改变现存的MANAGER角色以使用操作系统授权：

```
ALTER ROLE manager IDENTIFIED EXTERNALLY;
```

从数据库中删除manager角色：

```
DROP ROLE manager;
```

CREATE/ALTER/DROP ROLLBACK SEGMENT

```
CREATE [PUBLIC] ROLLBACK SEGMENT segment_name
  TABLESPACE tablespace
  [STORAGE Storage_Clause]
```

创建回退段，Oracle用它存储数据，必要时回退事务所做的变化。

```
ALTER ROLLBACK SEGMENT segment_name
```

```
[STORAGE Storage_Clause]
[ONLINE | OFFLINE]
[SHRINK]
```

改变回退段在线状态或修改存储特性。

```
DROP ROLLBACK SEGMENT segment_name
```

从数据库中删除回退段。

关键字

PUBLIC

指定该回退段可以用于任何事务。如果省略，则只可用于在 *INIT.ORA* 文件的 ROLLBACK_SEGMENTS 参数中命名的实例。

segment_name

被创建的回退段。

TABLESPACE

指定要被创建的回退段的表空间名。

STORAGE

指定物理存储特性。请参见第四章得到更详细的信息。

ONLINE

指定将命名的回退段置于在线状态。

OFFLINE

指定将命名的回退段置于离线状态。

SHRINK

指定命名的回退段应该减为指定大小，或者如果没有指定大小的话，就变为 OPTIMAL 大小

注意：创建了回退段时，回退段将处于离线状态，必须通过 ALTER ROLLBACK SEGMENT 命令将其置为在线状态或用含有回退段名的 *INIT.ORA* 文件重新启动数据库。

示例

创建回退段 rbs02:

```
CREATE ROLLBACK SEGMENT rbs02
   TABLESPACE rollback
   STORAGE (INITIAL 40K NEXT 40K OPTIMAL 80K );
```

将回退段 rbs02 置为离线状态:

```
ALTER ROLLBACK SEGMENT rbs02 OFFLINE;
```

为 rbs02 改变存储分配，所以每个扩展盘区都是 30K，其优化大小为 60K，且最大允许 10 个扩展盘区:

```
ALTER ROLLBACK SEGMENT rbs02
STORAGE (NEXT 30K MAXEXTENTS 10 OPTIMAL 60K);
```

从数据库中删除回退段 rbs02:

```
DROP ROLLBACK SEGMENT rbs02;
```

CREATE SCHEMA

```
CREATE SCHEMA AUTHORIZATION schema
  [CREATE TABLE statement]
  [CREATE VIEW statement]
  [GRANT statement]
```

创建多表和/或视图，并在单个语句中授权。

关键字

schema

指定被创建的架构名，必须与用户名相同。

CREATE TABLE

这是 CREATE TABLE 语句，本章后面将要演示。

CREATE VIEW

这是 CREATE VIEW 语句，本章后面将要演示。

GRANT

这是GRANT语句，本章后面将要演示。

警告：大家必须具有CREATE TABLE、CREATE VIEW和GRANT要求的相同权限来执行该语句。CREATE SCHEMA中的单个命令不必使用SQL终止符进行终止。

示例

为scott创建架构，由两张表和一个视图组成，并在视图上授权给一个角色：

```
CREATE SCHEMA AUTHORIZATION scott
CREATE TABLE dept (
      deptno     NUMBER NOT NULL,
      dname      VARCHAR2(20),
      location   VARCHAR2(15),
      avg_salary NUMBER (9,2))
CREATE TABLE emp (
      ename      VARCHAR2(20),
      deptno     NUMBER,
      sal        NUMBER (7,2),
      comm       NUMBER (7,2))
CREATE VIEW deptview AS SELECT deptno,dname,location FROM dept
GRANT SELECT ON deptview TO non_admin;
```

CREATE/ALTER/DROP SEQUENCE

```
CREATE SEQUENCE [schema.]sequence_name
  [INCREMENT BY integer]
  [START WITH integer]
  [MAXVALUE integer | NOMAXVALUE]
  [MINVALUE integer | NOMINVALUE]
  [CYCLE | NOCYCLE]
  [CACHE integer | NOCACHE]
  [ORDER | NOORDER]
```

创建一个可以在数据库操作期间用于自动产生序列号的Oracle序号发生器。

```
ALTER SEQUENCE [schema.]sequence_name
  [INCREMENT BY integer]
  [MAXVALUE integer | NOMAXVALUE]
  [MINVALUE integer | NOMINVALUE]
  [CYCLE | NOCYCLE]
  [CACHE integer | NOCACHE]
  [ORDER | NOORDER]
```

改变Oracle序号发生器的特性，包括范围、内存缓存中的序列号以及是否保留序列顺序。

```
DROP SEQUENCE [schema.]sequence_name
```

从数据库中删除序列。

注意：DROP SEQUENCE命令和CREATE SEQUENCE命令可以相继运行，从而在一个较小数字上重新启动序列，但也必须在序号发生器上重新创建所有授权。

关键字

INCREMENT BY

指定序列号之间的增量，可以为正数或负数（但不能为0）。默认值为1。

START WITH

指定被产生的第一个序列号。对升序创建的默认值为MINVALUE，对降序产生的默认值为MAXVALUE。

MAXVALUE

指定序列号到达的最大值。默认是NOMAXVALUE，这意味着最大值为10^{27}。

MINVALUE

指定序列号到达的最小值。默认是NOMINVALUE，这意味着最小值为1。

CYCLE

指定序列号达到最大值时，将采用最小值开始，默认值为NOCYCLE。

NOCYCLE

指定达到最大值以后，不产生另外的序列号。

CACHE

指定Oracle将预先产生多少序列号并保留在内存中。注意当数据库关闭时，存储在缓存中未使用的序列号将会丢失。默认值为20。

NOCACHE

指定不预先在内存中产生序列号。

ORDER

指定序列号按要求的次序产生。默认是NOORDER。

NOORDER

指定不保证按要求的顺序产生序列号。

注意：序列号的产生不受事务后来的回退影响，一旦产生，则序列成员将不可再用，所以会产生间隙。序列成员通过使用伪列CURRVAL和NEXTVAL访问。

示例

创建序号发生器ord_seq，使得下一个产生的序列号将是101，而且保证顺序。序列最大值为9999，然后重复循环到1：

```
ALTER SEQUENCE ord_seq
   START WITH 101
   MINVALUE 1
   MAXVALUE 9999
   CYCLE
   ORDER;
```

在scott架构中修改序号发生器ord_seq，使得下一个产生的序列号是10 001，而且保证顺序：

```
ALTER SEQUENCE scott.ord_seq
     MINVALUE 10001
     ORDER;
```

从数据库的scott架构中删除序号发生器ord_seq：

```
DROP SEQUENCE scott.ord_seq;
```

CREATE/ALTER/DROP SNAPSHOT

```
CREATE SNAPSHOT [schema.]snapshot_name
  [Physical_Attributes_Clause]
  [TABLESPACE tablespace]
  [STORAGE Storage_Clause]
  [REFRESH
     [FAST | COMPLETE | FORCE]
     [START WITH date]
```

```
    [NEXTREF date]
AS snapshot_query
```

创建快照，这是在一张或多张表或视图上运行查询的结果。

```
ALTER SNAPSHOT [schema.]snapshot_name
  [Physical_Attributes_Clause]
  [STORAGE Storage_Clause]
  [REFRESH
     [FAST | COMPLETE | FORCE]
     [START WITH date]
     [NEXTREF date]
```

改变快照的存储或自动刷新特性。

```
DROP SNAPSHOT [schema.]snapshot_name
```

从数据库中删除快照。

关键字

Physical_Attributes_Clause

指定该快照的物理属性。请参见第四章了解详细信息。

TABLESPACE

指定存储快照表空间名。如果省略，则使用架构所有者使用的默认表空间。

STORAGE

指定物理存储特性。请参见第四章了解详细信息。

REFRESH

指定自动刷新的模式和时间。FAST意味着快照日志关联主表；COMPLETE 意味着通过重新执行快照的查询进行刷新；FORCE是默认值，意味着Oracle将决定如果可能，则执行FAST刷新，否则将在必要时进行COMPLETE刷新。

START WITH

使用标准Oracle日期表达式指定下一次自动刷新时间的日期。

NEXTREF

指定计算自动刷新时间间隔之间的新日期表达式。

AS

提供用于操纵快照的实际SQL查询，而且服从视图的相同限制。

注意：在试图创建快照之前，必须用SYS运行*dbmssnap.sql*脚本创建嵌入式DBMS_SNAPSHOT包。

因为在架构中创建快照对象时，Oracle在快照名上附加了7个字符标识符，所以应限制快照名为23个字符或更少。

示例

创建scott的emp表的快照，放在称为uk的服务器上。快照将在明天移出，而且从现在开始7天刷新一次，而且此后每7天刷新一次：

```
CREATE SNAPSHOT uk_emp
   REFRESH COMPLETE
   START WITH SYSDATE+1
   NEXT SYSDATE+7
   AS SELECT * FROM scott.emp@UK;
```

计划的scott用户的快照dept_snap在明天午夜刷新，而且此后每周刷新：

```
ALTER SNAPSHOT scott.dept_snap
   REFRESH COMPLETE
   START WITH TRUNC(SYSDATE + 1)
   NEXT TRUNC(SYSDATE + 7);
```

从数据库中删除emp_UK快照：

```
DROP SNAPSHOT emp_UK;
```

CREATE/ALTER/DROP SNAPSHOT LOG

```
CREATE SNAPSHOT LOG ON [schema.]tablename
  [Physical_Attributes_Clause]
  [TABLESPACE tablespace]
  [STORAGE Storage_Clause]
```

创建快照日志（与快照主表相关的表，用于控制快照的刷新）。

```
ALTER SNAPSHOT LOG ON [schema.]tablename
  [Physical_Attributes_Clause]
  [STORAGE Storage_Clause]
```

改变快照日志的存储特性。

```
DROP SNAPSHOT LOG ON [schema.]table_name
```

从数据库中删除快照日志。

关键字

tablename

指定要维护快照日志的表名。

Physical_Attributes_Clause

指定该快照的物理属性。请参见第四章了解详细信息。

TABLESPACE

指定存储该快照的表空间名。如果省略，则使用架构所有者的默认表空间。

STORAGE

指定物理存储特性。请参见第四章了解详细信息。

示例

在 scott 的 emp 表上创建快照日志:

```
CREATE SNAPSHOT LOG ON scott.emp
   STORAGE (INITIAL 50K NEXT 50K PCTINCREASE 0)
   TABLESPACE users;
```

改变 scott 的 emp 表上的快照日志下一个扩展空间大小为 500K:

```
ALTER SNAPSHOT LOG scott.emp
   STORAGE (NEXT 500K);
```

从数据库中删除与 scott 的 emp 表相关的快照日志:

```
DROP SNAPSHOT LOG ON scott.emp;
```

CREATE/DROP SYNONYM

```
CREATE [PUBLIC] SYNONYM synonym_name
  FOR [schema.]object_name[@dblink]
```

为数据库对象创建公有或私有同义词。

```
DROP [PUBLIC] SYNONYM [schema.]synonym_name
```

从数据库中删除公有或私有同义词。

关键字

PUBLIC

指定同义词可以用于所有用户。如果省略，则同义词只可以用于架构所有者。

synonym_name

指定新同义词名。

object_name

指定指向同义词的对象名。它可以通过附加@*dblink*语法来包括对远端数据库的引用。

注意：Oracle首先在当前架构中解决对象名，所以PUBLIC同义词将只用于对象名不用架构名开头的情况，而且不采用*dblink*，不存在于当前架构中。

示例

在UK数据库上创建scott的emp表的同义词：

```
CREATE PUBLIC SYNONYM uk_emps
FOR scott.emp@UK;
```

从数据库中删除公共同义词uk_emps：

```
DROP PUBLIC SYNONYM uk_emps;
```

CREATE/ALTER/DROP TABLE

```
CREATE [GLOBAL] [TEMPORARY] TABLE [schema.]tablename
  (column  datatype [DEFAULT expression] [Column_Contraint]
   [,column  datatype [DEFAULT expression] [Column_Contraint]...]
 [Table_Constraint_Clause])
 [Physical_Attributes_Clause]
 [TABLESPACE tablespace]
 [STORAGE Storage_Clause]
 [LOGGING | NOLOGGING]
```

```
[CLUSTER (column[,column ...])]
[{ORGANIZATION HEAP
   [Physical_Attributes_Clause]
   [TABLESPACE tablespace]
   [STORAGE Storage_Clause]
   [LOGGING | NOLOGGING] |
  ORGANIZATION INDEX
   [PCTTHRESHOLD integer]
   [COMPRESS integer | NOCOMPRESS]
   [Physical_Attributes_Clause]
   [TABLESPACE tablespace]
   [STORAGE Storage_Clause]
   [LOGGING | NOLOGGING]
   [[INCLUDING column] OVERFLOW
     [Physical_Attributes_Clause]
     [TABLESPACE tablespace]
     [STORAGE Storage_Clause]
     [LOGGING | NOLOGGING]
 }]
[LOB {(lob_item[,lob_item ...]) STORE AS
       ([TABLESPACE tablespace]
        [{ENABLE | DISABLE} STORAGE IN ROW]
        [STORAGE Storage_Clause]
        [CHUNK integer]
        [PCTVERSION integer]
        [CACHE | NOCACHE [LOGGING | NOLOGGING]]
       )
      ) |
      (lob_item) STORE AS [(lob_segname)]
       [([TABLESPACE tablespace]
         [{ENABLE | DISABLE} STORAGE IN ROW]
         [STORAGE Storage_Clause]
         [CHUNK integer]
         [PCTVERSION integer]
         [CACHE | NOCACHE [LOGGING | NOLOGGING]]
      )]
     }
[Partition_Clause]
[ENABLE | DISABLE ROW MOVEMENT]
[CACHE | NOCACHE]
[MONITORING | NOMONITORING]
[PARALLEL integer | NOPARALLEL]
[ENABLE | DISABLE [VALIDATE | NOVALIDATE]]
   {UNIQUE (column[, column ...] | PRIMARY KEY | CONSTRAINT constraint_name}
   [USING INDEX
   [TABLESPACE tablespace]
   [Physical_Attributes_Clause]
   [STORAGE Storage_Clause]
   [NOSORT]
   [LOGGING | NOLOGGING]]
   EXCEPTIONS INTO [schema.]table_name]
   CASCADE] ]
 [AS subquery]
```

通过指定结构或通过引用现存的表创建一张表。

```
ALTER TABLE [schema.]tablename
[ADD([column  datatype [DEFAULT expression] [Column_Constraint_Clause]
      [,column datatype [DEFAULT expression] [Column_Constraint_Clause] ...)]
       [Table_Constraint_Clause]
       [LOB {(lob_item[,lob_item ...]) STORE AS
        ([TABLESPACE tablespace]
         [{ENABLE | DISABLE} STORAGE IN ROW]
         [STORAGE Storage_Clause]
         [CHUNK integer]
         [PCTVERSION integer]
         [{CACHE | NOCACHE [LOGGING | NOLOGGING]}]
        } |
        (lob_item) STORE AS [(lob_segname)]
         [([TABLESPACE tablespace]
           [{ENABLE | DISABLE} STORAGE IN ROW]
           [STORAGE Storage_Clause]
           [CHUNK integer]
           [PCTVERSION integer]
           [CACHE | NOCACHE [LOGGING | NOLOGGING]]
        )]
[Partition_Clause]
[MODIFY[(column  datatype [DEFAULT expression] [Column_Constraint_Clause]
        [,column  datatype [DEFAULT expression] [Column_Constraint_Clause] ...)]]
        [{NESTED TABLE | VARRAY} collection_item [RETURN AS {LOCATOR | VALUE}]]
[MOVE [ONLINE] [Physical_Attributes_Clause]
 {[TABLESPACE tablespace] [LOGGING | NOLOGGING] |
  [PCTTHRESHOLD integer]
  [COMPRESS integer | NOCOMPRESS]
  [INCLUDING column] OVERFLOW
    [Physical_Attributes_Clause]
    [TABLESPACE tablespace]
    [STORAGE Storage_Clause]
    [LOGGING | NOLOGGING]
 }
[LOB {(lob_item[,lob_item ...]) STORE AS
      ([TABLESPACE tablespace
       [{ENABLE | DISABLE} STORAGE IN ROW]
       [STORAGE Storage_Clause]
       [CHUNK integer]
       [PCTVERSION integer
       [{CACHE | NOCACHE [LOGGING | NOLOGGING]}]
      ) |
      (lob_item) STORE AS [(lob_segname)]
       [([TABLESPACE tablespace
         [{ENABLE | DISABLE} STORAGE IN ROW]
         [STORAGE Storage_Clause]
         [CHUNK integer]
         [PCTVERSION integer
         [{CACHE | NOCACHE [LOGGING | NOLOGGING]}]
      )]
```

```
[Physical_Attributes_Clause] [STORAGE Storage_Clause]
[LOGGING | NOLOGGING]
[MODIFY CONSTRAINT constraint_name Constraint_State_Clause]
[DROP CONSTRAINT constraint_name] [{PRIMARY | UNIQUE (column[,column ...])} [CASCADE]]
[{DROP COLUMN column | DROP COLUMN (column[,column ...])}
  [CASCADE CONSTRAINTS] [INVALIDATE] [CHECKPOINT integer]
[DROP {UNUSED COLUMNS | COLUMNS CONTINUE} [CHECKPOINT integer]
[{SET UNUSED COLUMN column | SET UNUSED COLUMN (column[,column ...])}
  [CASCADE CONSTRAINTS | INVALIDATE]
[{ALLOCATE EXTENT ([SIZE integer[K | M]]
                   [DATAFILE 'filename']
                   [INSTANCE integer]
                  ) |
  DEALLOCATE UNUSED [KEEP integer[K | M]]
 }]
[CACHE | NOCACHE]
[MONITORING | NOMONITORING]
[RENAME TO new_table_name]
[{MINIMIZE | NOMINIMIZE} RECORDS_PER_BLOCK]
[PCTTHRESHOLD integer | INCLUDING column]
[OVERFLOW [Physical_Attributes_Clause]
          [ALLOCATE EXTENT ([SIZE integer [K | M]]
                            [DATAFILE 'filename']
                            [INSTANCE integer]
                           )
          [DEALLOCATE UNUSED [KEEP integer[K | M]]
          [LOGGING | NOLOGGING]
[Partition_Clause [Partition_Clause] ...]
[PARALLEL integer | NOPARALLEL]
  [{ENABLE | DISABLE}] [{TABLE LOCK | ALL TRIGGERS}]
```

修改表的存储特性或列或与表和/或其列关联的完整性约束。

```
DROP TABLE [schema.]table_name
   [CASCADE CONSTRAINTS]
```

从数据库中删除表。

关键字

GLOBAL TEMPORARY

指定创建的表将是具有所有会话都可见的结构的临时表，但只有创建会话可以看见数据。临时表必须创建在临时表空间之中。

column

指定列名创建为表的一部分。

datatype

指定与列相关的数据类型。

DEFAULT

指定列的默认值，如果在插入到表的行中省略该列的值时将使用该默认值。表达式必须与列的数据类型匹配。

Column_Constraint_Clause

使用第四章列出的语法指定列约束。

Table_Constraint_Clause

使用第四章列出的语法指定表约束。

Physical_Attributes_Clause

指定该表的物理属性。可以从第四章了解详细信息。

TABLESPACE

指定存储表的表空间名。如果省略，就使用架构所有者的默认表空间。

STORAGE

指定物理存储特性 。要了解详细信息，参见第四章。

LOGGING

指定在对象创建时写恢复日志记录。该选项是默认情况。

NOLOGGING

指定创建对象时不写恢复日志记录。在数据库失败的情况下，不能用日志文件恢复该操作，必须重新创建该对象。该选项将加速数据库对象的创建。

ORGANIZATION HEAP

指定该表中没有数据行存储时关联的顺序。该选项是默认情况。

ORGANIZATION INDEX

指定创建的表作为索引组织的表，意味着数据行实际保留在表的主键上定义的索引。

PCTTHRESHOLD

指定每一个索引块保留数据的空间百分比。任何不能放置在该空间的数据行部分将被放置在溢出段。

COMPRESS

指定压缩这些键。

NOCOMPRESS

指定不压缩这些键。这是默认情况。

INCLUDING

指定在这一点上的表行在索引和溢出部分之间分区。*column*后的所有列（主键列除外）将被存储在溢出段中。

OVERFLOW

指定索引组织的表行超过了 PCTTHRESHOLD 值的行将被放置在该从句中描述的段中。

LOB

指定 LOB（大对象，Large OBject）数据的存储属性。

lob_item

指定 LOB 列名。

STORE AS

指定 LOB 数据段名。

ENABLE STORAGE IN ROW

指定该行中存储的 LOB 值。如果指定为索引组织表，那么就必须指定为 OVERFLOW 。该选项是默认值。

DISABLE STORAGE IN ROW

指定该行外部存储的 LOB 值。

CHUNK

指定为 LOB 操纵分配的字节数（与最近数据库块大小接近）。

PCTVERSION

指定在创建新版本的 LOB 中使用的 LOB 存储空间的最大百分比。

ENABLE ROW MOVEMENT

指定如果根据键更新的需要可以把该行移到不同分区或子分区上。

DISABLE ROW MOVEMENT

指定行可以被移动到不同分区或子分区，并在键更新需要这样移动时返回错误。

MONITORING

指定可以为该表收集的修改统计信息。

NOMONITORING

指定不为该表收集修改统计信息。该选项是默认值。

ENABLE

指定将对表的所有新数据应用约束。

DISABLE

指定不对该表应用约束。

VALIDATE

在指定ENABLE时，该关键字引起Oracle验证表中所有存在的数据遵守该约束。

NOVALIDATE

在指定ENABLE时，该关键字阻止Oracle验证表中存在的数据遵守约束，但确保新添加到表中的数据遵守约束。

USING INDEX

指定用于强加约束的索引特性。

EXCEPTIONS INTO

指定表名，Oracle在该表中放置有关违背约束行的信息。该表必须在使用这一关键字之前通过运行 *UTLEXCPT1.SQL* 脚本隐式创建。

AS

指定用于创建时插入行到表中的子查询。如果在CREATE TABLE 语句中省略列定义，则将从子查询中引用的表复制列名和数据类型。

DROP

删除完整性约束。

ALLOCATE EXTENT

使用特定参数为表隐式分配新的扩展盘区。

CASCADE CONSTRAINTS

指定引用表的主键和惟一键被删除时，所有参照完整性约束也将被删除。

示例

在 scott 的架构中创建新表，名为 dept:

```
CREATE TABLE scott.dept (
   deptno              NUMBER(2) NOT NULL,
   dname            VARCHAR2(14),
   loc              VARCHAR(15))
   TABLESPACE USERS
   STORAGE (INITIAL 40K NEXT 4K PCTINCREASE 0)
   PCTFREE 15;
```

创建 scott 的 emp 表的一个备份:

```
CREATE TABLE test_emp
   AS SELECT * FROM scott.emp;
```

在 scott 的 emp 表上添加新列:

```
ALTER TABLE scott.emp
   ADD (bonus NUMBER(7,2));
```

增加 bonus 列的大小为 9 位数字:

```
ALTER TABLE scott.emp
   MODIFY (bonus NUMBER(9,2));
```

在 scott 的 emp 表上添加主键约束:

```
ALTER TABLE scott.emp
   MODIFY (empno  CONSTRAINT pk_emp PRIMARY_KEY);
```

从数据库中删除 scott 的 emp 表:

```
DROP TABLE scott.emp;
```

警告: 删除表时，所有行都被删除。任何保留在表上的索引也自动被删除，无论创建的架构是什么，也无论它们当前属于谁。如果被删除的表是一个视图的基表，或者如果它在任何存储过程中被引用，那么视图或存储过程将被标记为无效（但不会被删除）。如果该表是快照的主表，则快照不会被删除。同样地，如果表具有快照日志，则快照日志不会被删除。

CREATE/ALTER/DROP TABLESPACE

```
CREATE TABLESPACE tablespace_name
  DATAFILE 'filename' [SIZE integer [K | M] [REUSE]] [Autoextend_Clause]
        [,'filename' [SIZE integer [K | M] [REUSE]] [Autoextend_Clause]]
  DEFAULT STORAGE Storage_Clause
  [ONLINE | OFFLINE]
  [PERMANENT | TEMPORARY]
  [LOGGING | NOLOGGING]
  [MINIMUM EXTENT integer]
```

创建新表空间，为表空间中随后创建的对象选择性地指定默认存储特性。

```
ALTER TABLESPACE tablespace_name
  {
   [ADD DATAFILE filename [SIZE integer [K | M]] [REUSE]]
   [Autoextend_Clause]
   [RENAME 'filename1' TO 'filename2']
   [DEFAULT STORAGE Storage_Clause
   [ONLINE]|OFFLINE]
   [PERMANENT | TEMPORARY]
   [BEGIN BACKUP | END BACKUP]
   [LOGGING | NOLOGGING]
   [MAXIMUM EXTENT integer]
  }
```

通过添加或改变数据文件改变存在的表空间，改变存储参数，让表空间离线，让表空间在线，放置为备份模式，或者从备份模式取出。

```
DROP TABLESPACE tablespace_name
   [INCLUDING CONTENTS] [CASCADE CONSTRAINTS]
```

从数据库中删除表空间。

关键字

DATAFILE

指定该表空间的操作系统数据文件名。除非文件已经存在，否则需要SIZE。如果文件确实已经存在，则必须指定REUSE关键字。

DEFAULT STORAGE

指定物理存储特性。请参见第四章了解详细信息。

ONLINE

创建之后，将表空间置为在线状态。该选项是默认值。

OFFLINE

创建之后，将表空间置为离线状态。

PERMANENT

指定表空间可以包含的永久对象。

TEMPORARY

指定该表空间将只创建临时对象。

LOGGING

指定在该表空间中创建对象时写恢复日志记录。该选项是默认值，但可以通过对象层的NOLOGGING忽略它。

NOLOGGING

指定在该表空间中创建对象时不写恢复日志记录。在数据库失败的情况下，这样的操作不能用日志文件恢复，而必须重新创建对象。该选项加速了数据库对象的创建。

MINIMUM EXTENT

指定表空间中每个使用的或自由扩展空间至少与*integer*一样大，而且是*integer*的倍数。该空间控制了自由空间碎片。

BEGIN BACKUP

Oracle认为开始备份表空间的信号，因此改变了日志文件行为为该表空间累计所有的块变化。注意该语句并不实际执行备份；它只是向Oracle发出备份开始的信号。

END BACKUP

表空间备份完成的Oracle信号，因此把日志文件行为恢复为正常情况。

INCLUDING CONTENTS

指定在该表空间中包含的任何对象将被自动删除。如果不包括该关键字而且在表空间中存在任何对象，则该语句将失败。

CASCADE CONSTRAINTS

指定该表空间外部的表的参照完整性约束将被删除，这里的表空间指向该表空间的表中的主键和惟一键。

示例

创建一个名为users的新的25 MB的表空间:

```
CREATE TABLESPACE users
   DATAFILE '/disk09/oracle/oradata/users01.dbf' SIZE 25M
   DEFAULT STORAGE (INITIAL 500K NEXT 50K PCTINCREASE 0);
```

在users表空间上添加一个新25 MB数据文件:

```
ALTER TABLESPACE users
   ADD DATAFILE '/disk09/oracle/oradata/users02.dbf' SIZE 25M;
```

向Oracle发出开始备份users表空间的信号:

```
ALTER TABLESPACE users
   BEGIN BACKUP;
```

从数据库中删除users表空间和它包含的所有对象:

```
DROP TABLESPACE users INCLUDING CONTENTS;
```

CREATE TEMPORARY TABLESPACE

```
CREATE TEMPORARY TABLESPACE
   TEMPFILE 'filename' [SIZE integer [K | M] [REUSE]]
   [Autoextend_clause]
   [EXTENT MANAGEMENT LOCAL]
   [UNIFORM] [SIZE integer [K | M]]
```

创建临时表空间，将用于保留临时对象（只在会话期间保留）。

注意: 像其他的表空间一样，临时表空间可以使用DROP TABLESPACE命令删除。

关键字

TEMPFILE

指定该临时表空间的操作系统数据文件名。

SIZE

以字节、KB或MB的形式指定文件大小。除非文件已经存在，否则需要SIZE。如果文件已经存在，则必须指定使用REUSE关键字。

REUSE

指定操作系统文件必须已经存在，而且可以被临时表空间重新使用。

INITIAL

以字节、KB或MB的形式指定第一个扩展盘区的大小。如果不是数据库块大小的倍数，就必须取余为其倍数。

Autoextend_clause

为数据文件指定是否使自动扩展有效。请参见第四章了解详细信息。

EXTENT MANAGEMENT LOCAL

指定局部管理的临时表空间，意味着该表空间的一些部分被留置为位图。

UNIFORM SIZE

以字节、KB或MB的形式指定该表空间中所有扩展盘区的大小，而且显示所有的扩展盘区的大小相等。如果省略，则默认扩展盘区的大小为1MB。

警告：临时表空间不受介质恢复所影响。

示例

创建名为tempspace的临时表空间，包含100MB的空间，而且每个扩展盘区的大小等于2MB：

```
CREATE TEMPORARY TABLESPACE tempspace
   TEMPFILE '/disk09/oracle/oradata/tempsp01.dbf' SIZE 100M
   EXTENT MANAGEMENT LOCAL UNIFORM SIZE 2M;
```

CREATE/ALTER/DROP TRIGGER

请参见第七章了解这些语句的完整描述。

CREATE/ALTER/DROP USER

```
CREATE USER username
  [IDENTIFIED [BY password | EXTERNALLY]]
  [DEFAULT TABLESPACE tablespace
  [TEMPORARY TABLESPACE tablespace]
  [QUOTA [INTEGER [K | M] | UNLIMITED] ON tablespace]
```

```
  [QUOTA [INTEGER [K | M] | UNLIMITED] ON tablespace]
  ...
  [PROFILE profilename]
```

创建新数据库用户并分配安全性和存储属性。

```
ALTER USER username
  [IDENTIFIED [BY password | EXTERNALLY]]
  [DEFAULT TABLESPACE tablespace]
  [TEMPORARY TABLESPACE tablespace]
  [QUOTA [INTEGER [K | M] | UNLIMITED] ON tablespace]
  [QUOTA [INTEGER [K | M] | UNLIMITED] ON tablespace]
  ...
  [PROFILE profilename]
  [DEFAULT ROLE
    {rolename[,rolename ...] |
     ALL [EXCEPT rolename[,rolename ...]] |
     NONE
    }]
```

改变用户的安全性和存储特性。

```
DROP USER username [CASCADE]
```

从数据库中删除用户，并选择性地删除被用户创建的对象。

关键字

IDENTIFIED BY

为该用户账户指定使用的口令，或者如果指定为EXTERNALLY，那么该用户的授权将由操作系统处理。

注意：你可以使用 ALTER USER 命令改变自己的口令，无论你是否有 ALTER USER 的系统权限。

DEFAULT TABLESPACE

当该用户创建数据库对象时，指定默认使用的表空间名。

TEMPORARY TABLESPACE

当类似于排序这样的操作需要更多可用内存时，指定用于创建临时段使用的表空间名。

QUOTA

指定特定的表空间中该用户允许为对象存储使用的空间数量。UNLIMITED意味着在存储使用上没有限制（受表空间的整体大小支配）。

PROFILE

设置用户 profile 为 *profilename*，这使用户受 profile 中指定的限制支配。

CASCADE

指定在删除用户之前删除用户架构中所有对象。如果用户架构中包含任何对象，则必须指定该关键字；否则，命令失败。

警告： 如果你指定CASCADE选项，则在其他架构的表上指定的参照完整性约束指向该架构中表上的主键和惟一键也将被删除。如果架构中的表或其他数据库对象被另一架构中的视图或同义词、或存储过程、函数或包查阅，那么那些被查阅的对象都将被标记为无效，但不会被删除。

示例

创建新用户 scott:

```
CREATE USER scott IDENTIFIED BY tiger
   DEFAULT TABLESPACE users
   TEMPORARY TABLESPACE temp
   QUOTA 500K ON users;
```

创建相同用户，但要确保该用户被操作系统授权:

```
CREATE USER ops$scott IDENTIFIED EXTERNALLY
   DEFAULT TABLESPACE users
   TEMPORARY TABLESPACE temp
   QUOTA 500K ON users;
```

为 scott 分配新口令:

```
ALTER USER scott IDENTIFIED BY lion;
```

为用户 scott 改变默认和临时表空间:

```
ALTER USER scott
   TEMPORARY TABLESPACE temp
```

```
DEFAULT TABLESPACE users;
```

从数据库中删除用户scott和其架构中的内容:

```
DROP USER scott CASCADE;
```

CREATE/ALTER/DROP VIEW

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW [schema.]viewname
  [(alias[,alias ...])]
AS viewquery
  [WITH CHECK OPTION [CONSTRAINT constraint]]
```

创建视图。

```
ALTER VIEW [schema.]viewname COMPILE
```

重新编译视图。

```
DROP VIEW [schema.]view_name
```

从数据库中删除视图。

关键字

OR REPLACE

如果已经存在，指定替换该视图。

FORCE

指定无论是否视图的基表存在,或者无论架构的所有者是否具有这些表上的权限，都创建视图。

NOFORCE

指定只有在基表存在且架构所有者有权限时才创建视图。该选项是默认情况。

viewname

指定被创建的视图名。

alias

对应于视图查询返回的列或表达式的一个或多个别名。

viewquery

任何 SQL SELECT 语句。要了解 SELECT 的语法，参见第三章。

WITH CHECK OPTION

指定通过视图执行插入和更新一定会产生可以选择视图查询的行。

CONSTRAINT

为CHECK OPTION约束指定名字。该默认值是系统分配的SYS_C*n*的形式名，因为这里的 *n* 是整数，从而产生惟一名。

COMPILE

在更改视图和引起视图重新编译时，需要该关键字。

注意：ALTER VIEW 语句隐式重新编译了视图，在改变了视图的基表之后推荐使用该语句。

示例

创建名为 emploc 的视图：

```
CREATE OR REPLACE VIEW emploc
   (empno,lname,location) AS
SELECT empno,ename,loc
   FROM scott.emp,scott.dept
   WHERE emp.deptno = dept.deptno;
```

重新编译 scott 的视图 emploc：

```
ALTER VIEW scott.emploc RECOMPILE;
```

从数据库中删除 scott 的 emploc 视图：

```
DROP VIEW scott.emploc;
```

DISASSOCIATE STATISTICS

```
DISASSOCIATE STATISTICS FROM
  {COLUMNS [schema.]table.column[, [schema.]table.column ...] |
   FUNCTIONS [schema.]function[,[schema.]function ...] |
   PACKAGES [schema.]package[,[schema.]package ...] |
   INDEXES [schema.]index[,[schema.]index ...]
  }
```

从数据库对象上分离统计信息计算的方法。

关键字

COLUMNS

指定提供的一个或多个列的清单。

schema

包含特定表的架构。

table

包含特定列的表名。

column

关联定义的列名。

FUNCTIONS

指定将被分离的一个或多个函数。

PACKAGES

指定将被分离的一个或多个包。

INDEXES

指定将被分离的一个或多个索引。

示例

从函数 emp_sal 中删除统计关联:

```
DISASSOCIATE STATISTICS FROM FUNCTIONS emp_sal;
```

EXPLAIN PLAN

要得到该语句的完整解释，参见第三章和第八章。

GRANT (对象权限)

```
GRANT {object_priv[,object_priv ...] | ALL [PRIVILEGES]}
  ON {[schema.]object_name |
      DIRECTORY directory_name |
```

```
      JAVA {SOURCE | RESOURCE} [schema.]java_object
     }
  TO {username | role | PUBLIC}
  [WITH GRANT OPTION]
```

这样就对一个或多个用户或角色进行了数据库对象的授权。

关键字

object_priv

指定被授权的对象权限名。有效的权限是：ALTER、DELETE、EXECUTE、INDEX、INSERT、REFERENCES、SELECT 和 UPDATE。

object_name

指定要被授权的对象名。

DIRECTORY

指定要被授权的目录对象。

JAVA SOURCE

指定要被授权的 Java 源。

JAVA RESOURCE

指定要被授权的 Java 资源对象名。

username

指定要被授权的用户名。

role

指定要被授权的角色名。

PUBLIC

指定给当前和将来的用户授权的对象权限。

WITH ADMIN OPTION

指定权限保证可以授权给其他人。

注意：对象必须在架构中，否则就必要采用 GRANT OPTION 给对象授权。

示例

将 scott 的 emp 表的 INSERT 和 UPDATE 权限授给 debby:

```
GRANT insert, update ON scott.emp TO debby;
```

GRANT (系统权限或角色)

```
GRANT {privilege | role}[,{privilege | role} ...]
  TO {username | rolename | PUBLIC}[,{username | rolename | PUBLIC} ...]
  [WITH ADMIN OPTION]
```

给一个或多个用户和/或角色授予系统权限。

关键字

privilege

指定被授权的系统权限名。

role

指定被授权的角色名。

username

指定被授权的用户名或角色。

rolename

指定被授权的角色名或角色。

PUBLIC

指定给所有用户授予权限或角色，包括那些还没有被创建的。

WITH ADMIN OPTION

指定授予的权限或角色可以给其他人授权或添加角色，也可以更改或删除该角色。

示例

将 account_admin 角色授给 scott 和 debby:

```
GRANT account_admin TO scott, debby;
```

将 CREATE USER 和 DROP USER 权限授给 dba_assist 角色:

```
GRANT create user, drop user TO dba_assist;
```

NOAUDIT (架构对象)

```
NOAUDIT object_option[,object_option ...]
ON {[schema.]objectname | DIRECTORY directory_name | DEFAULT}
[WHENEVER [NOT] SUCCESSFUL]
```

通过架构对象的优先 AUDIT 语句定义停止审核。

关键字

object_option

显示将停止特定操作上的审核。下列操作是有效的: ALTER、AUDIT、COMMENT、DELETE、EXECUTE、GRANT、INDEX、INSERT、LOCK、RENAME、SELECT 和 UPDATE。关键字 ALL 等于指定所有操作。

object_name

指定将要停止审核的架构对象名。

DIRECTORY

指定将要停止审核的目录名。

DEFAULT

指定不执行审核，因为还没有为特定的对象选项创建对象的默认值。

WHENEVER SUCCESSFUL

只有在 SQL 语句成功完成时才关闭审核。

WHENEVER NOT SUCCESSFUL

在 SQL 语句失败或产生错误时关闭审核。

示例

为 scott 的 bonus 表关闭审核执行的 UPDATE 语句:

```
NOAUDIT UPDATE ON scott.bonus;
```

关闭审核在 scott 的 emp 表上执行的任何未成功的操作:

```
NOAUDIT ALL ON scott.emp WHENEVER NOT SUCCESSFUL;
```

NOAUDIT (SQL 语句)

```
NOAUDIT {statement_opt | system_priv}[,{statement_opt | system_priv} ...]
  [BY  {username[,username ...] | proxy [ON BEHALF OF {ANY | username
       [,username ...]}]}]
  [WHENEVER [NOT] SUCCESSFUL]
```

通过为 SQL 语句定义的优先 AUDIT 语句停止审核。

关键字

statement_opt

指定将要停止审核的语句选项。

system_priv

指定将要停止审核的系统权限。

BY username

只对清单中的用户执行的SQL语句停止审核。该默认选项是停止对多所有用户的审核。

BY proxy

只对代表用户或特定用户列表的代理人发出的 SQL 语句停止审核。

WHENEVER SUCCESSFUL

只对彻底完成的 SQL 语句停止审核。如果指定为NOT，则只有在 SQL 语句引发错误时停止审核。如果省略该从句，则无论成功与否都停止对所有SQL语句的审核。

示例

停止由 scott 执行的 INSERT 和 DELETE 语句的审核:

```
NOAUDIT INSERT,DELETE BY scott;
```

RENAME

RENAME *oldname* TO *newname*

改变现存的表、视图、序列或私有同义词名。

注意：老对象上的完整性约束、索引和授权都被自动转移到新对象上。

关键字

oldname

指定想要分配新名字的现存对象名。

newname

指定数据库对象的新名。

警告：依赖于重命名对象的对象（例如，视图、同义词、存储过程或函数）将被标记为无效。

示例

改变 emp 表的名字为 employees:

```
RENAME emp TO employees;
```

REVOKE (对象权限)

```
REVOKE {object_priv[,object_priv ...] | ALL [PRIVILEGES]}
  ON [schema.]object_name
  FROM {username | role | PUBLIC}
```

从一个或多个用户或角色中收回数据库对象上的权限。

关键字

object_priv

指定被授权的对象权限名。有效的权限是：ALTER、DELETE、EXECUTE、INDEX、INSERT、REFERENCES、SELECT 和 UPDATE。

object_name

指定被授权的对象名。

username

指定被授予对象权限的用户名。

role

指定被授予对象权限的角色名。

PUBLIC

指定为所有当前和将来的用户授予对象权限。

示例

从 debby 上收回对 scott 的 emp 的 INSERT 和 UPDATE 权限:

```
REVOKE insert,update ON scott.emp FROM debby;
```

REVOKE (系统权限或角色)

```
REVOKE {privilege | role}[,{privilege | role} ...]
  FROM {username | rolename | PUBLIC}[,{username | rolename | PUBLIC} ...]
```

从一个或多个用户和/或角色上删除系统权限。

关键字

privilege

指定被收回的系统权限名。

role

指定被收回的角色名。

username

指定被收回权限或角色的用户名。

rolename

指定被收回权限或角色的角色名。

PUBLIC

指定默认情况下不再可以用于所有用户的授权或角色。

示例

从 scott 和 debby 上收回 account_admin 角色:

```
REVOKE account_admin FROM scott,debby;
```

从 dba_assist 角色上收回 CREATE USER 和 DROP USER 权限:

```
REVOKE create user, drop user FROM dba_assist;
```

第三章

数据操纵和控制语句

数据操纵语言（Data Manipulation Language，DML）访问和操纵存储在Oracle数据库中的数据。大家可以使用它们插入、删除和读取数据。控制语句是相关的，因为它们用于控制访问数据库中的数据时Oracle怎样操作。DML和控制语句没有数据定义语言（Data Definition Language，DDL）语句多，但许多DML和控制语句确实有较广范围的选项和语法选择。本章提供这些SQL命令的快速参考，并提供语法指南。

根据任务说明SQL DML和控制语句

表3-1列出了最常用的数据操纵任务，以及完成这些任务的适当SQL语句。每一语句的详细语法在本章后面列出。

表3-1：公共DML任务、控制任务和SQL语句

目 的	使用该命令
改变表、视图或快照中行的数据	UPDATE
改变会话的特性	ALTER SESSION
检查每条DML语句后的约束	SET CONSTRAINT
收集数据库中有关对象的统计信息	ANALYZE
为SQL语句创建执行计划解释	EXPLAIN PLAN

表 3-1：公共 DML 任务、控制任务和 SQL 语句（续）

目 的	使用该命令
延迟约束检查，直到事务结束为止	SET CONSTRAINT
从表、视图或快照中删除行	DELETE
删除数据库中对象的统计信息	ANALYZE
为当前会话停止使用某角色	SET ROLE
让当前会话可以使用某角色	SET ROLE
建立当前会话作为读－写	SET TRANSACTION
建立当前会话为只读	SET TRANSACTION
定义表或簇中移植的行和链接行	ANALYZE
在表或视图中插入数据行	INSERT
在事务中建立一个点，从而可以从该点进行回退	SAVEPOINT
从表或簇中删除所有行	TRUNCATE
从表、视图或快照中提取数据	SELECT
指定被事务使用的回退段	SET TRANSACTION
验证对象的结构	ANALYZE

SQL 语句语法

本节提供了处理Oracle数据库的数据时可能使用的每条SQL语句的详细参考。每条语句显示为一种或多种形式，而且简要解释了语句用途，在可应用时显示了它的确切语法和相关参数及应用例子和使用注解。

ALTER SESSION

```
ALTER SESSION
  {{SET
     [CONSTRAINT[S] = IMMEDIATE | DEFERRED | DEFAULT] |
     [CREATE_STORED_OUTLINES = TRUE | FALSE | category_name] |
     [CURRENT_SCHEMA = schema] |
     [CURSOR_SHARING = FORCE | EXACT] |
     [DB_BLOCK_CHECKING = TRUE | FALSE] |
     [DB_FILE_MULTIBLOCK_READ_COUNT = integer] |
     [FAST_START_IO_TARGET = integer] |
     [FLAGGER = ENTRY | INTERMEDIATE | FULL | OFF] |
     [GLOBAL_NAMES = {TRUE | FALSE}] |
     [HASH_AREA_SIZE = integer] |
```

```
[HASH_JOIN_ENABLED = TRUE | FALSE] |
[HASH_MULTIBLOCK_IO_COUNT = integer] |
[INSTANCE = integer] |
[ISOLATION_LEVEL = SERIALIZABLE | READ COMMITTED] |
[LABEL = 'text' | DBHIGH | DBLOW | OSLABEL] |
[LOG_ARCHIVE_DEST_integer =
  {'' | LOCATION=pathname | SERVICE=tnsnames_service}
  [MANDATORY | OPTIONAL] [REOPEN[=integer]]] |
[LOG_ARCHIVE_DEST_STATE_integer = ENABLE | DEFER] |
[LOG_ARCHIVE_MINIMUM_SUCCEED_DEST = integer] |
[MAX_DUMP_FILE_SIZE = integer | UNLIMITED]
[NLS_CALENDAR = 'text'] |
[NLS_COMP = 'text'] |
[NLS_CURRENCY = 'text'] |
[NLS_DATE_FORMAT = 'date_format'] |
[NLS_DATE_LANGUAGE = language] |
[NLS_DUAL_CURRENCY = 'text'] |
[OPTIMIZER_GOAL = ALL_ROWS | FIRST_ROWS | RULE | CHOOSE] |
[NLS_ISO_CURRENCY = territory] |
[NLS_LABEL_FORMAT = label_format] |
[NLS_NUMERIC_CHARACTERS = 'text'] |
[NLS_LANGUAGE = language] |
[NLS_SORT = sort | BINARY] |
[NLS_TERRITORY = territory] |
[OBJECT_CACHE_MAX_SIZE_PERCENT = integer] |
[OBJECT_CACHE_OPTIMAL_SIZE = integer] |
[OPTIMIZER_INDEX_CACHING = integer] |
[OPTIMIZER_INDEX_COST_ADJ = integer] |
[OPTIMIZER_MAX_PERMUTATIONS = integer] |
[OPTIMIZER_MODE = ALL_ROWS | FIRST_ROWS | RULE | CHOOSE] |
[OPTIMIZER_PERCENT_PARALLEL = integer] |
[PARALLEL_BROADCAST_ENABLED = TRUE | FALSE] |
[PARALLEL_INSTANCE_GROUP = 'text'] |
[PARTITION_VIEW_ENABLED = TRUE | FALSE] |
[PLSQL_V2_COMPATIBILITY = TRUE | FALSE] |
[QUERY_REWRITE_ENABLED = TRUE | FALSE] |
[QUERY_REWRITE_INTEGRITY = ENFORCED | TRUSTED | STALE_TOLERATED] |
[REMOTE_DEPENDENCIES_MODE = TIMESTAMP | SIGNATURE] |
[SESSION_CACHED_CURSORS = integer] |
[SKIP_UNUSABLE_INDEXES = TRUE | FALSE] |
[SORT_AREA_RETAINED_SIZE = integer] |
[SORT_AREA_SIZE = integer] |
[SORT_MULTIBLOCK_READ_COUNT = integer] |
[SQL_TRACE = TRUE | FALSE] |
[STAR_TRANSFORMATION_ENABLED = TRUE | FALSE] |
[TIMED_STATISTICS = TRUE | FALSE] |
[USE_STORED_OUTLINES = TRUE | FALSE | 'category_name'] |
} |
[CLOSE DATABASE LINK dblink] |
[ADVISE COMMIT | ROLLBACK | NOTHING] |
[{ENABLE|DISABLE} COMMIT IN PROCEDURE] |
[{ENABLE|DISABLE|FORCE} PARALLEL {DML | DDL} [PARALLEL integer]
}
```

改变当前数据库会话的功能特性，包括改变许多国际语言支持（National Language Support，NLS）特性。

关键字

CONSTRAINT[S]

指定由延期约束定义的条件何时被强制执行。IMMEDIATE 意味着该条件在每一条 DML 语句之后立即进行检查。DEFERRED 指在事务提交后检查该条件。DEFAULT 恢复所有的约束到初始状态，与创建时定义的相同。

CREATE_STORED_OUTLINES

指定 Oracle 是否为每条查询存储一个轮廓。如果提供 *category_name*，那么就创建轮廓并存储在 *category_name* 分类中。

CURRENT_SCHEMA

改变当前架构为特定 *schema*。尽管改变了架构、但用户没有改变，而且没有额外的权限可以使用。

CURSOR_SHARING

指定可以共享游标的 SQL 语句种类。如果指定 FORCE，那么，如果语句除了一些文字和不影响语句意义的差别之外都相同的话，就可以共享游标。EXACT 意味着只有相同的 SQL 语句可以共享游标。

DB_BLOCK_CHECKING

指定是否执行数据块检查。

DB_FILE_MULTIBLOCK_READ_COUNT

指定执行序列扫描时在单个 I/O 操作中读块的数量。默认值是 8。

FAST_START_IO_TARGET

指定破产崩溃恢复或实例恢复时应该从或到缓存中执行读和写的目标数量。

FLAGGER

指定 FIPS 标志层，当 SQL 语句不遵守 ANSI SQL-92 标准时、会产生一个错误消息。注意当前在 ENTRY、INTERMEDIATE 和 FULL 之间没有差别。

GLOBAL_NAMES

控制是否将全局名解决方案强加于会话之上。

HASH_AREA_SIZE

指定用于散列联接的内存数量（以字节计）。

HASH_JOIN_ENABLED

指定是否在查询中执行散列联接。

HASH_MULTIBLOCK_IO_COUNT

指定散列联接中读或写的数据块的数量。

INSTANCE

指定处于并行服务器环境下，尽管会话连接到特定的实例中，也应该访问数据库文件。

ISOLATION_LEVEL

指定怎样处理数据库修改。SERIALIZABLE 意味着如果试图在一条已经更新而且还没有被另一个会话提交的行上进行更新操作则该语句会失败；该语句与 SQL-92 标准中指定的串行事务隔离模式兼容。READ COMMITTED 意味着 Oracle的默认行为有效，且如果行被另一个未提交的事务加锁，则该语句将等到释放行锁为止才开始执行。

LABEL

改变 DBMS 会话标签为 *text* 指定的标签，即 DBHIGH 或 DBLOW 或操作系统标签（operating system label，OSLABEL）的标签等同物。

LOG_ARCHIVE_DEST_integer

指定归档恢复日志文件组的位置。最多可以定义 5 个位置，如指定为整数，则归档进程将试图为每个位置归档恢复日志文件。

"

指定不该归档日志目的位置定义日的位置。但是，至少需要为一个归档日志日的位置（1~5）定义一个位置。

LOCATION

为归档恢复日志文件指定操作系统位置。

SERVICE

指定运行备用数据库相关的 *tnsnames.ora* 文件中，*tnsnames_service* 条目的 Net8 服务名。

MANDATORY

指定可以重新使用恢复日志文件之前必须成功归档到目的文件中。

OPTIONAL

指定可以重新使用恢复归档文件之前不必成功归档到目的文件中。

REOPEN

指定归档到目的文件和将来可以试图归档到目的文件之间遇到错误后等待的秒数。

LOG_ARCHIVE_DEST_STATE_integer

指定与对应的LOG_ARCHIVE_DEST_*integer*关联的状态。ENABLE意味着可以使用任何有效的LOG_ARCHIVE_DEST。DEFER指不能使用与LOG_ARCHIVE_DEST_*integer*值相同的整数。

LOG_ARCHIVE_MINIMUM_SUCCEED_DEST

指定在可以重新使用恢复日志文件之前，必须成功写的目的文件的最小值。

MAX_DUMP_FILE_SIZE

以块的形式指定追踪倾倒文件的最大值。如果使用UNLIMITED，则表示没有限制。

NLS_CALENDAR

指定新的日历类型。

NLS_COMP

指定使用作为*text*提供的NLS_SORT参数关联的规则执行语言比较。

NLS_CURRENCY

指定被数据格式元素L返回的本地货币符号。该参数不考虑NLS_TERRITORY设置的默认值。

NLS_DATE_FORMAT

指定默认的日期格式。*date_format*必须是有效的Oracle日期格式样式。该参数不考虑NLS_TERRITORY设置的默认值。

NLS_DATE_LANGUAGE

指定用作天和月名的语言，也包括其他特定日期值。该参数不考虑NLS_LANGUAGE设置的默认值。

NLS_DUAL_CURRENCY

指定被数字格式元素U（正常用于欧洲）返回的值。

NLS_ISO_CURRENCY

指定应该使用ISO货币符号的领域。该参数不考虑NLS_TERRITORY设置的默认值。

NLS_LABEL_FORMAT

只用于信任的Oracle；为会话改变默认标签格式。

NLS_LANGUAGE

为Oracle消息、天和月名及排序顺序指定语言。

NLS_NUMERIC_CHARACTERS

指定十进制字符和分组操作符。'*text*'值必须以'*dg*'的形式提供，这里的*d*代表十进制字符、而*g*代表分组字符。该参数不考虑NLS_TERRITORY设置的默认值。

NLS_SORT

指定字符排序的排序规则。BINARY指定了二进制排序，而*sort*指定了特定排序名。

NLS_TERRITORY

指定默认日期格式值、数字十进制、组分割符和本地及ISO货币符号。参数不考虑NLS_LANGUAGE设置的默认值。

OBJECT_CACHE_MAX_SIZE_PERCENT

指定对象缓存增长能超出最佳大小的百分比。默认值是10。

OBJECT_CACHE_OPTIMAL_SIZE

为对象缓存指定最佳大小（以KB计）。默认值是100。

OPTIMIZER_GOAL

对Oracle7，指定该会话的优化目标。下面的目标有效：

ALL_ROWS

优化得到最佳全部吞吐量。

FIRST_ROWS

优化得到最佳响应时间。

RULE

使用基于规则的优化。

CHOOSE

如果可能，则使用基于成本的优化；否则，使用基于规则的优化。

OPTIMIZER_INDEX_CACHING

指定缓存中假设的索引块百分比。

OPTIMIZER_INDEX_COST_ADJ

指定百分比显示优化器放在索引路径上（而不是整表扫描）的可用性的重要性。

OPTIMIZER_MAX_PERMUTATIONS

指定表交换的数量，优化器将在很大的联接操作时用到该数值。

OPTIMIZER_MODE

对 Oracle8，就是指定该会话的优化目标。下面的优化目标有效：

ALL_ROWS

优化得到最佳全部吞吐量。

FIRST_ROWS

优化得到最佳响应时间。

RULE

使用基于规则的优化。

CHOOSE

如果可能，则使用基于成本的优化；否则，使用基于规则的优化。

OPTIMIZER_PERCENT_PARALLEL

指定计算成本时使用优化器的对应数量。默认值为0，表示没有对应的优化器。

PARALLEL_BROADCAST_ENABLED

指定在散列联接和合并联接中，并行处理可以用于加强性能。

PARALLEL_INSTANCE_GROUP

在产生并行查询从属进程时，指定并行实例组。只有并行服务器运行在并行模式时该选项才有效。

PARTITION_VIEW_ENABLED

指定操作分区视图时可以省略不必要的表访问。

PLSQL_V2_COMPATIBILITY

指定是否允许PL/SQL结构在Oracle7(PL/SQL 2.0)下合法，但现在在Oracle8中都是不合法的。值为TRUE表示允许老结构。值为FALSE表示不允许这样的结构。

QUERY_REWRITE_ENABLED

指定是否重写的查询对物化视图有效。如果OPTIMIZER_MODE被设置为RULE，则表示重写查询无效。

QUERY_REWRITE_INTEGRITY

指定重写查询的连贯性。ENFORCED表示依赖于系统强加的关系，所以可以保证数据完整性。TRUSTED意味着创建的物化视图具有ON PREBUILD TABLE从句支持，而且接受未强加的连接关系。STALE_TOLERATED意味着可以使用陈旧但可用的物化视图。

REMOTE_DEPENDENCIES_MODE

指定怎样处理远端存储过程的依赖性。

SESSION_CACHED_CURSORS

为该会话指定缓存中可以保留的游标数量。

SKIP_UNUSABLE_INDEXES

指定是否允许在表上采取不可用的索引或索引分区操作。TRUE意味着允许所有这样的操作，而FALSE则使得这样的操作返回一个错误。

SORT_AREA_RETAINED_SIZE

以字节的形式指定第一次提取后每一个排序操作将要保留的最大内存数。

SORT_AREA_SIZE

以字节的形式指定每一个排序操作将使用的内存最大数量。

SORT_MULTIBLOCK_READ_COUNT

指定每次排序执行从临时段中读的块数量。默认值为2。

SQL_TRACE

控制是否产生性能统计信息。在INIT.ORA文件中设置初始值。

STAR_TRANSFORMATION_ENABLED

指定是否对星型查询应用基于性能的优化。

TIMED_STATISTICS

指定产生基于时间的统计信息时，是（TRUE）否（FALSE）Oracle需要操作系统上的时间信息。

USE_STORED_OUTLINES

产生执行计划时，指定是否优化器将使用存储轮廓。如果提供*category_name*，则表示只使用在目录下存储的轮廓。

CLOSE DATABASE LINK

使用数据库链接*dblink*关闭远端数据库的连接。该命令只有在没有使用数据库链接，而且没有提交链接时能成功。

ADVISE

通过在远端服务器的DBA_2PC_PENDING_ADVICE上放置'C' (COMMIT)、'R' (ROLLBACK)或'' (NOTHING)给远端数据库发送强制执行分布式事务的建议。

ENABLE COMMIT IN PROCEDURE

指定存储过程和存储函数可以执行COMMIT语句和ROLLBACK语句。

DISABLE COMMIT IN PROCEDURE

指定存储过程和存储函数不可以执行COMMIT语句和ROLLBACK语句。

ENABLE PARALLEL

指定在可能的情况下可以并行在会话中执行DML和DDL语句。该选项是DDL语句的默认值。

DISABLE PARALLEL

指定可以在会话中串行执行DML和DDL语句。该选项是DML语句的默认值。

FORCE PARALLEL

指定随后的语句在会话中以并行方式执行。

DML

指定在DML语句中应用ENABLE、DISABLE或FORCE PARALLEL关键字。

DDL

指定在DDL语句中应用ENABLE、DISABLE或FORCE PARALLEL关键字。

PARALLEL

指定并行度。整数将不考虑DDL语句中指定的并行从句，但要考虑随后的DML语句中指定的并行提示。

注意：许多可以使用该命令设置的参数都通过初始化文件(INIT.ORA)中的实例范围基础上的参数定义。尽管适当时在这里显示了默认值，但该初始化参数值将忽略那些默认值，并会成为事实上的默认值。大家试图为会话设置参数之前，要确保理解每个参数的用法。

示例

使会话能进行SQL跟踪的技巧：

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

警告：除了希望进行性能统计时以外，都应该把SQL_TRACE设置为FALSE，因为该选项降低了整个数据库的性能。如果SQL_TRACE被设置为TRUE，则跟踪文件将在udump目录下创建。如果要执行大量的SQL语句，就可以填充该目录。

设置该语言为法语，然后使用4位数字的年份覆盖具有美国风格的日期格式：

```
ALTER SESSION
  SET NLS_LANGUAGE = French;
ALTER SESSION
  SET NLS_DATE_FORMAT - 'mm/dd/yyyy';
```

ANALYZE

```
ANALYZE {TABLE [schema.]tablename |
         INDEX [schema.]indexname |
         CLUSTER [schema.]clustername
        }
       {COMPUTE STATISTICS |
        ESTIMATE [SAMPLE integer {ROWS | PERCENT}] STATISTICS
```

```
DELETE STATISTICS
VALIDATE STRUCTURE [CASCADE]
LIST CHAINED ROWS [INTO [schema.]tablename]
}
```

收集或删除数据库中关于对象的统计信息，验证对象的结构，或定义表或簇中的移植行和链接行。

关键字

COMPUTE STATISTICS

为整个命名对象计算确切的统计数字并存储在数据目录中。

ESTIMATE STATISTICS

为整个命名对象估计统计数字并存储在数据目录中。可选的SAMPLE从句可以用于指定使用的采样大小。SAMPLE从句包含了下列关键字：

ROWS

使表或簇的 *integer* 行或索引的 *integer* 条目被采样。

PERCENT

使表或簇的行的 *integer* 百分比或索引条目的 *integer* 百分比被采样。PERCENT的有效范围是1~99。如果没有指定SAMPLE，则默认采样1050行。

DELETE STATISTICS

使存储在数据目录中的命名对象的所有统计信息都被删除。

VALIDATE STRUCTURE

使命名对象的结构被校验。CASCADE关键字还将使索引相关的命名对象被校验。

LIST CHAINED ROWS

为命名行或簇产生链接的或移植的行清单（不允许在索引上执行该操作）。假设在用户架构中存在该表，则条目都产生在CHAINED_ROWS表中。INTO从句可以用于为目标表指定不同名字。

注意：大家必须拥有被分析的对象或者具有ANALYZE ANY权限。如果你想使用LIST CHAINED ROWS操作列出到一张表中，则该表必须在自己的架构中，或者必须具有该表上的INSERT权限。COMPUTE STATISTICS导致更多的累计统计信息，但可能占用的时间更长。ESTIMATE STATISTICS通常比精确统计要快。在收集统计信息时，被分析的对象将被加锁，所以在繁重的事务环境下宁可采用快速的ESTIMATE STATISTICS统计方法。在ALL_TABLES、USER_TABLES和DBA_TABLES视图中可以访问统计信息。一些列的统计信息可以在ALL_TAB_COLUMNS、USER_TAB_COLUMNS和DBA_TAB_COLUMNS视图中访问。簇统计也出现在USER_CLUSTERS和DBA_CLUSTERS中。

实例

使用50%的采样大小分析scott的emp表：

```
ANALYZE TABLE scott.emp ESTIMATE SAMPLE 50 PERCENT STATISTICS;
```

DELETE

```
DELETE [FROM] {[schema.]table[@dblink] |
               [schema.]table PARTITION (partition) |
               [schema.]table SUBPARTITION (subpartition) |
               [schema.]view[@dblink] |
               [schema.]snapshot[@dblink] |
               (subquery [WITH {READ ONLY}] |
               CHECK OPTION [CONSTRAINT constraint_name])
                }]
              }
[table_alias]
[WHERE condition]
[RETURNING expression[,expression ...] INTO data_item[,data_item ...]]
```

从表、视图或快照中删除行。

关键字

FROM

有助于可读性的可选关键字。

schema

包含要被删除行的表或视图的架构名。如果省略，则假定为当前默认的用户ID。

table

要删除的行所在的表名。

dblink

用于访问要删除行的表或视图的数据库链接名。

PARTITION

指定从特定表的分区（命名分区）中删除行。

SUBPARTITION

指定从特定表的子分区（命名子分区）中删除行。

subquery

指定子查询，该子查询决定被删除的候选行。本章后面的SELECT将介绍有关子查询的更多信息。

WITH READ ONLY

指定子查询不能被更新。

WITH CHECK OPTION

指定如果子查询会产生不包括在该子查询中的行，则该子查询潜在对应的表不能发生改变。

CONSTRAINT

指定与CHECK OPTION关联的约束名。

table_alias

指定表、视图或子查询的别名（或更改名）。

WHERE

指定用于定义被删除的行的条件。指定的*condition*可以是任何有效的WHERE条件。

RETURNING

指定特定表达式的值将是该命令删除的行返回的值。只有在PL/SQL程序中该关键字才有效。

INTO

指定存储该命令删除行返回的值所在的PL/SQL变量。

警告：如果指定table_alias，则在DELETE语句中引用特定表的任何列都必须使用table_alias而不是表名来限制。

示例

从 scott 的 emp 表中删除职工的薪水超过 2000 的所有行：

```
DELETE FROM scott.emp
WHERE sal>2000;
```

从 scott 的 emp 表中删除职工的薪水超过平均薪水的所有行：

```
DELETE FROM scott.emp
WHERE sal>(SELECT AVG(sal) FROM scott.emp);
```

EXPLAIN PLAN

```
EXPLAIN PLAN
   SET STATEMENT_ID = 'text'
   [INTO [schema.]tablename[@dblink]]
   FOR SQL_statement
```

为 SQL 语句执行计划创建解释。

关键字

SET STATEMENT_ID

指定文本字符串，用于标识该EXPLAIN PLAN语句的结果。默认值为NULL。

INTO

指定计划表的名称和位置。默认值是在当前架构中使用 PLAN_TABLE 表。

FOR

指定 SQL 语句，产生计划。

注意：在执行该命令之前，大家必须具有目的表上的INSERT权限（尤其是INTO）。目的表通常称为 PLAN_TABLE，而且可以通过运行脚本 *utlxplan.sql* 产生。在 SET 从句中指定的值出现在目的表的 STATEMENT_ID 列中。

示例

下面的例子为SQL语句产生执行计划。输出结果将被放置在当前架构的plan_table表中：

```
EXPLAIN PLAN
   SET STATEMENT_ID = 'Plan1'
   FOR
   SELECT ename, sal, comm, loc
   FROM emp,dept
   WHERE emp.deptno = dept.deptno;
```

要了解有关计划表的内容，请参见第八章。

INSERT

```
INSERT INTO {schema.{table[{@dblink] |
                        table PARTITION (partition) |
                        table SUBPARTITION (subpartition) |
                        view[@dblink] |
            }[table_alias]
[(column[,column...])
{VALUES (expr[,expr...]) |
 subquery
}
[RETURNING expression[,expression...] INTO data_item[,data_item...]]
```

在表或视图中插入一行数据。

注意：如果指定*table_alias*，则在INSERT语句中引用的任何具有特定表引用的列必须使用*table_alias*限定，而不使用表名。

关键字

schema

包含将要插入行的表或视图的架构名。如果省略，就假定为当前用户ID。

table

被插入行的表名。

dblink

访问被插入行的表或视图的数据库链接名。

PARTITION

指定将行插入到特定表的分区（命名分区）中。

SUBPARTITION

指定将行插入到特定表的子分区（命名子分区）中。

view

被插入的行的视图名。

table_alias

指定表或视图的别名（或选择名）。

column

指定要存储值的表或视图的一列或多列名。如果指定 VALUES 关键字，则要指定每一列，且在 VALUES 从句中必须指定对应的 *expr*。如果省略列名，则认为清单中包含表或视图的所有列。

VALUES

指定被插入的行的每一列值。*expr* 可以是任何有效的 SQL 表达式，而且必须确实有 *exprs* 中列出的指定的表或视图的列。如果不提供列清单，则必须有表达式的相同编号作为表或视图的列。

subquery

指定子查询，返回被插入的行要存储的值。如果指定被插入行的表或视图的列，则 *subquery* 必须确切返回列的相同顺序号。如果没有指定列的清单，则 *subquery* 必须返回表或视图的相同列号。要得到子查询的更多信息，参见本章后面的 SELECT。

RETURNING

指定被该命令插入行返回的表达式的值。该关键字只在 PL/SQL 程序中有效。

INTO

指定被该命令插入行的返回值将被存储在 PL/SQL 变量 *data_item* 中。

示例

插入新行到 scott 的 emp 表中。只有 5 个特定列具有值：

```
INSERT INTO emp (empno, ename, hiredate, mgr, sal)
```

```
VALUES (7999, 'KREINES', '01-JUN-00', 7839, 500)
```

使用子查询导出行插入新行到 newemp 表中。所有列都接受子查询的值:

```
INSERT INTO newemp
   SELECT * FROM scott.emp WHERE comm > 0;
```

SAVEPOINT

SAVEPOINT *savepoint*

在事务中定义一点，在这一点上使用 ROLLBACK 命令可以回退。

savepoint

指定创建的保存点名。

示例

设置保存点，更新 emp 表的 *sal* 列，然后执行 ROLLBACK 命令使表的状态返回到建立保存点的条件下:

```
UPDATE emp
   SET comm = comm * 1.1;

SAVEPOINT updtsal;

UPDATE emp
   SET sal = sal * 1.1;

ROLLBACK TO updtsal;
```

SELECT

```
SELECT {DISTINCT | UNIQUE | ALL}
{[schema.]{table. | view. | snapshot.}* |
 expr [[AS] alias][,expr [[AS] alias] ...] |
 *
FROM {schema.{table[@dblink] |
             table PARTITION (partition) |
             table SUBPARTITION (subpartition) |
             table SAMPLE [BLOCK] sample_percent |
             view[@dblink] |
             snapshot[@dblink] |
             (subquery)
     } [table_alias]
```

```
     [,{schema.{table[@dblink] |
               table PARTITION (partition) |
               table SUBPARTITION (subpartition) |
               table SAMPLE [BLOCK] sample_percent |
               view[@dblink] |
               snapshot[@dblink] |
              (subquery)
     } |
       }[table_alias]
      ...]
[WHERE condition]
[GROUP BY {expr[,expr ...] |
           CUBE (expr[,expr ...]) |
           ROLLUP (expr[,expr ...])
          }
          [HAVING condition]
[[START WITH condition] CONNECT BY condition]
[{UNION [ALL] | INTERSECT | MINUS} (subquery)
 [,{UNION [ALL] | INTERSECT | MINUS} (subquery) ...]]
[ORDER BY {expr | position | alias} [ASC | DESC]
 [, {expr | position | alias} [ASC | DESC] ...]
[FOR UPDATE [OF schema.{table | view].column[,schema.{table | view].column ...]
     [NOWAIT]
```

从表、视图或快照中提取数据。

关键字

DISTINCT

指定只应该返回行的一个备份，即使存在重复行也是这样。重复行是指在SELECT清单中列出的所有列都返回相同值的行。

ALL

指定应该返回所有行，包括重复行。该选项是默认值。

schema

包含表或视图的架构名。如果省略，就假定为当前用户ID。

table

指定表名。

view

指定视图名。

snapshot

指定快照名。

* 指定返回所有列，等价于列出表、视图或快照的每一列。

expr

任何有效的 Oracle 表达式，通常包括表、视图或快照中的一列或多列。

AS 指定列或表达式的别名（或选择名），关键字 AS 可选。

FROM

指定从一张或多张表、视图或快照中提取数据。

PARTITION

指定从由 *partition* 定义的分区 *table* 分区中提取数据。

SUBPARTITION

指定从由 *subpartition* 定义的 *table* 子分区中提取数据。

SAMPLE [BLOCK]

指定选择表中随机采样的行。在语法中用你想要使用的百分比代替 *sample_percent*。如果指定可选关键字BLOCK，则会执行块采样，而不是执行行采样。

subquery

任何有效的 SELECT 语句。注意，子查询中不包含 FOR UPDATE 从句。

table_alias

指定表、视图或快照的别名（或选择名）。

WHERE

指定只有满足 *condition* 的行会被提取。将对 *condition* 求值，而且只有求得的值为 TRUE 的行才会被返回。如果省略该从句，则会返回所有行。

GROUP BY

指定根据提供的表达式对行分组，而且为每一组返回一条摘要信息。

CUBE

指定根据从表达式清单上提供所有可能组合值对行进行分组。

ROLLUP

指定这些行根据提供的表达式清单进行分组，而且返回每一个 *expr* 的摘要行，并带有附加的合计行。

HAVING

指定只返回满足条件的分组数据，这是用与WHERE条件相同的方式求值为TRUE。

START WITH

指定用作层次查询的根的行。如果省略，则将所有行都当作根行。

CONNECT BY

在层次中指定父行和子行之间的关系。

UNION [ALL]

指定在这之前的SELECT语句的结果将与其后SELECT语句的结果组合。

INTERSECT

指定在这之前的SELECT语句的结果将与其后SELECT语句的结果组合，而且只返回在两个SELECT中都出现的行。

MINUS

指定在这之前的SELECT语句的结果将与其后SELECT语句的结果组合。返回的结果是第一个SELECT语句选择的行的集合减去第二个SELECT语句的行的集合。

ORDER BY

指定在返回之前要排序的行。该排序可以在表达式、别名或位置上执行。在语法上，位置是指向SELECT列表中表达式位置的整数。SELECT列表的第一个项目被当作位置1。

ASC

指定值按照升序排序——从低到高。这是默认排序方式。

DESC

指定值按照降序排列——从高到低。

FOR UPDATE

指定锁定被选择的行。如果指定关键字OF，那么将只锁定在表中指定的行。

NOWAIT

指定如果表已经被加锁，那么Oracle不会等到锁释放。如果不指定NOWAIT，则Oracle将等到锁被释放为止。

警告：如果指定表的别名，则在SELECT语句中引用的任何模棱两可定义的列都必须使用表的别名限定，而不是表的名字。

示例

提取 scott 的 emp 表所有列的所有行：

```
SELECT *
FROM scott.emp;
```

从 scott 架构的 emp 和 dept 表中检索列的特定集合：

```
SELECT ename, dname, hiredate, sal, comm
FROM scott.emp, scott.dept
WHERE emp.deptno = dept.deptno;
```

从 scott 的架构的 emp 和 dept 表中检索列的指定集合。为 emp 表使用表的别名。使用子查询限制选择那些比平均薪水高的行。然后以薪水由低到高的顺序排序：

```
SELECT ename, dname, hiredate, sal, comm
FROM scott.emp a, scott.dept
WHERE a.deptno = dept.deptno
AND sal > (SELECT AVG(sal)
           FROM scott.emp)
ORDER BY sal;
```

为每一个部门检索最高和最低的薪水。使用列的别名：

```
SELECT dname Department, MAX(sal) Maximum, MIN(sal) Minimum
FROM scott.emp, scott.dept
WHERE emp.deptno = dept.deptno
GROUP BY dname;
```

SET CONSTRAINT

```
SET {CONSTRAINT | CONSTRAINTS} {ALL | constraint[,constraint...]}
{IMMEDIATE | DEFERRED}
```

指定在事务层是在每条 DML 语句之后检查特定约束，还是延迟到事务结束为止才检查特定约束。

注意：该语句只应用于可延迟的约束。

关键字

ALL

指定事务的所有可延迟约束都被该语句所影响。

constraint

指定可延迟的约束名。

IMMEDIATE

指定约束强加的条件在每条 DML 语句完成后进行检查。

DEFERRED

指定约束强加的条件在整个事务完成和提交后进行检查。

注意：可延迟约束的成功与否可以通过在执行 COMMIT 语句之前执行语句 SET CONSTRAINTS ALL IMMEDIATE 进行检查。

示例

延迟两个约束，直到事务完成为止：

```
SET CONSTRAINTS chk_sal,chk_comm DEFERRED;
```

SET ROLE

```
SET ROLE
   {role [IDENTIFIED BY password][,role [IDENTIFIED BY password ...]] |
    ALL EXCEPT role[,role ...] |
    NONE
   }
```

为当前会话允许或禁止使用该角色。

关键字

role

指定允许使用的角色名。

IDENTIFIED BY

指定角色的口令。如果要用口令保护该角色，则需要该选项。

ALL

指定能够授权所有的角色。如果包括了EXCEPT从句，那么将禁止特定的角色，但可以使用其他授权的角色。

NONE

指定在会话中所有授权角色均被禁止。

示例

允许为当前会话授予开发者角色：

```
SET ROLE developer IDENTIFIED BY manager;
```

SET TRANSACTION

```
SET TRANSACTION {READ ONLY |
                 READ WRITE |
                 ISOLATION LEVEL {SERIALIZABLE | READ COMMITTED} |
                 USE ROLLBACK SEGMENT seg_name
                }
```

建立当前事务为可以被该事务使用的只读、只写或特定的回退段。

关键字

READ ONLY

指定当前事务为只读。

READ WRITE

指定当前事务为只写。

ISOLATION_LEVEL

指定怎样处理数据库修改。SERIALIZABLE意味着如果试图更新另一个会话

已经更新但还没有提交的行，则该语句将失败；这种状态与ANSI SQL-92标准中指定的串行事务隔离层兼容。READ COMMITTED意味着Oracle的默认行为将起作用，而且如果一行被另一个未提交的事务锁定，则该语句将等到行锁被释放为止才执行。

USE ROLLBACK SEGMENT

将该事务分配给*seg_name*指定的回退段。该从句暗含着READ WRITE，而且不能指定为READ ONLY。

警告：如果使用了该语句，则该语句必须放在事务的开头。事务采用COMMIT或COMMIT WORK语句结束。

示例

指定回退段和执行更新事务：

```
SET TRANSACTION
   USE ROLLBACK SEGMENT rbs99;

UPDATE emp
SET SAL = SAL*1.1;

COMMIT;
```

TRUNCATE

```
TRUNCATE {TABLE [schema.]table [{PRESERVE | PURGE} SNAPSHOT LOG] |
          CLUSTER [schema.]cluster
         }
   [{DROP | REUSE} STORAGE]
```

从表或簇中删除所有行。

注意：要执行TRUNCATE命令，就必须要么是被截除的表的所有者，要么具有DROP ANY TABLE的系统权限。仅有对表的授权是不够的。

关键字

table

指定要删除行的表名。

cluster

指定要删除行的簇名。

PRESERVE SNAPSHOT LOG

在该表上指定现存的快照日志应该在表被删除时保留。在执行EXPort/TRUNCATE/IMPort操作重新加载该表时，该选项非常有用，因为不会触发快速刷新。

PURGE SNAPSHOT LOG

在该表上指定现存的快照日志应该在表被删除时清除。

DROP STORAGE

消除被行使用的存储空间并将空间返回到自由空间池中。该选项是默认值。

REUSE STORAGE

收回被删除行使用的空间。在重新加载表或簇中的数据时，该选项非常有用。

注意：TRUNCATE命令不创建回退记录，所以不能被回退。这个特性使得TRUNCATE非常快，所以在不需要回退能力时最好采用该命令而不采用DELETE FROM。在表被删除时，需要指定DROP STORAGE从句，只有表的初始化扩展空间被保留；所有的存储空间都被释放。

示例

从scott的emp表中删除所有行，而且释放为该表分配的空间：

```
TRUNCATE TABLE scott.emp;
```

UPDATE

```
UPDATE {[schema.]table[{@dblink] |
        [schema.]table PARTITION (partition) |
        [schema.]table SUBPARTITION (subpartition) |
        [schema.]table SAMPLE [BLOCK] sample_percent |
```

```
        [schema.]view[@dblink] |
        [schema.]snapshot[@dblink]
       }
       [table_alias]
SET {column={expr | (subquery)}[,column={expr | (subquery)} ... |
     (column[,column ...])=subquery
    }
[WHERE condition]
[RETURNING expression[,expression ...] INTO data_item[,data_item ...]]
```

改变在一张或多张表、视图或快照的一列或多列中存储的数据值。

关键字

schema

包含被更新的表或视图的架构名。如果省略，则假定为当前用户ID。

table

指定被更新的表名。

view

指定被更新的视图名。

snapshot

指定被更新的快照名。

PARTITION

根据分区定义的分区表中被更新的数据。

SUBPARTITION

根据子分区定义的子分区表中被更新的数据。

SAMPLE

指定被更新表中的随机采样行。采样大小用*sample_percent*指定。如果指定关键字BLOCK，则采用块采样，而不是执行行采样。

subquery

任何有效的SELECT语句。注意，子查询不可以包括FOR UPDATE从句。

table_alias

指定表、视图或快照的别名。

column

指定被更新的表、视图或快照的列名。

expr

任何有效的 Oracle 表达式。

WHERE

指定只有满足条件的行会被更新。该条件将被求值，而且只有求得的值为 TRUE 的行被更新，如果省略该从句，则更新所有行。

RETURNING

指定被该语句更新的行的表达式的值将被返回。该关键字只有在 PL/SQL 程序中有效。

INTO

指定被该语句更新的行的返回值被存储在 PL/SQL 变量 data_item 中。

示例

更新 scott 的 emp 表当前薪水少于 2000 的所有职工，将他们的薪水增长 10%：

```
UPDATE scott.emp
SET sal = sal * 1.1
WHERE sal < 2000;
```

更新 scott 的 emp 表当前薪水少于 2000 的所有职工，将他们的薪水和佣金都增长 10%：

```
UPDATE scott.emp
SET sal = sal * 1.1,
    comm=comm*1.1
WHERE sal < 2000;
```

更新 scott 的 emp 表中职工的随机 15% 采样。将当前薪水少于 2000 的职工的佣金和薪水都增长 10%：

```
UPDATE scott.emp SAMPLE 15
SET sal = sal * 1.1,
    comm = comm * 1.1
WHERE sal < 2000;
```

第四章

公共 SQL 元素

SQL 语法的许多组件用在多条 SQL 语句中。这些组件，通常指 SQL 从句，本身就很复杂。在第二章和第三章中，我们简单提供了 SQL 的语法，我在这里收集了这些公共元素。在前面章节的单个 SQL 语法描述中，只是引用了它们的名字和句法。

Autoextend_Clause

```
AUTOEXTEND {OFF |
            ON {NEXT integer[K | M]}
              [MAXSIZE {integer[K | M] | UNLIMITED}]
```

指定文件是否允许以指定大小自动增长。当文件允许以指定大小自动增长时，该从句还要指定增长参数。

关键字

OFF

指定不能执行自动扩展特性，而且该文件不允许以指定大小增长。

ON

指定在文件满而且需要增加额外空间时，该文件采用NEXT参数中指定的数字扩展具体空间，最大不超过 MAXSIZE 参数指定的限制。

NEXT

以字节、KB 或 MB 的形式指定增长发生时给文件添加的数量。

MAXSIZE

以字节、KB 或 MB 的形式指定文件的最大大小。自动扩展特性不能使文件大小超出该范围。

UNLIMITED

指定文件增长的容量可以是物理磁盘的大小，或者是操作系统允许的最大值，取最小值。

Column_Constraint_Clause

```
[CONSTRAINT constraint_name]
{[NULL | NOT NULL] |
 [UNIQUE | PRIMARY KEY (column[,column ...])] |
 [REFERENCES [schema.]table [(column[,column ...])] [ON DELETE {CASCADE | SET NULL}] |
 [CHECK (condition)]
}
[Constraint_State_Clause]
```

定义列约束。

关键字

CONSTRAINT

指定约束名。如果省略，则 Oracle 将以 SYS_C*nnn* 的形式分配名字，这里的 *nnn* 是整数。

NULL

指定列可以包含 NULL 值。

NOT NULL

指定列不可以包含 NULL 值。

UNIQUE

指定表中另一行的相同列的值不能与本行的列的值重复；也就是，对该表的所有行，该列的值必须惟一。但是，要注意，在该列中可以包含不止一个 NULL。

PRIMARY KEY

指定该列或组合列作为定义表的主键。主键列不可以包含NULL，而且同一张表的两条记录不能拥有相同主键值（即键值必须惟一）。

REFERENCES

指定被外部键引用的表或列。引用列的组合必须代表目标表的主键约束或惟一约束。

ON DELETE CASCADE

指定删除包含主键或惟一键的行时，包含依赖外部键的行也将被自动删除。

ON DELETE SET NULL

指定删除包含主键或惟一键的行时，依赖外部键的行也将被自动改变为NULL。

CHECK

指定为了满足约束，条件值必须等于TRUE或NULL。

Constraint_State_Clause

```
[{DEFERRABLE [INITIALLY {IMMEDIATE | DEFERRED} |
   NOT DEFERRABLE [INITIALLY IMMEDIATE ] |
   INITIALLY IMMEDIATE  [[NOT] DEFERRABLE] |
   INITIALLY  DEFERRED
 }]
[RELY | NORELY]
[USING INDEX
  [INITRANS integer]
  [MAXTRANS integer]
  [PCTFREE  integer]
  [TABLESPACE tablespace_name]
  [NOSORT]
  [LOGGING | NOLOGGING]
  [Storage_Clause]
[ENABLE | DISABLE]
[VALIDATE | NOVALIDATE]
[EXCEPTIONS INTO [schema.]table]]
```

允许约束可被选择为允许或禁止。

DEFERRABLE

指定可以延迟约束。延迟约束将等到事务提交以后才开始检查。

INITIALLY IMMEDIATE

指定应该在每一条DML语句执行之后立即开始求约束值。可以在事务中使用SET CONSTRAINTS语句改变这种行为。

INITIALLY DEFERRED

指定应该在事务提交时开始检查约束值。可以在事务中使用SET CONSTRAINTS语句改变这种行为。

NOT DEFERRABLE

指定对该约束不可以延迟检查。

RELY

指定即使相关的约束无效，物化视图（或快照）也适合于查询重写。该关键字只对物化视图有效。

NORELY

指定如果相关的约束无效，物化视图（或快照）将不适合于查询重写。该关键字只对物化视图有效。

USING INDEX

指定使用的索引将验证惟一键或主键约束。

INITRANS

指定在索引的每一个数据块上分配事务条目的初始数量。每个更新块的并发事务需要一个事务条目。

MAXTRANS

为索引的每一个数据块指定事务条目的最大数量。该关键字限制了可以更新块的并发事务的数量限制。默认大小是块的大小。

PCTFREE

指定在每一个索引块中为索引值更新保留的空间百分比。

TABLESPACE

指定存储约束索引的表空间名。

NOSORT

指定以升序的形式将数据行存储在数据库中，所以在创建索引时不需要排序。

LOGGING

指定执行索引创建时将书写恢复日志记录。该选项是默认行为。

NOLOGGING

指定执行索引创建时不书写恢复日志记录。在数据库失败的情况下，不能通过应用日志文件来恢复索引创建操作。相反，必须重新创建索引。该选项加速了使用索引创建约束。

Storage_Clause

指定强加约束的索引存储特性。看一看本章后面部分的“Storage_Clause”。

ENABLE

指定在表或视图的所有新数据上立即应用约束。

DISABLE

指定约束无效，而且不能对表或视图上的数据应用约束。

VALIDATE

指定表或视图上的现存数据必须遵守约束。

NOVALIDATE

指定表或视图上的现存数据不必检查是否遵守约束。

EXCEPTIONS INTO

指定Oracle将要放置违背该约束的行的rowid的表。该表必须存在，而且可以使用 *UTLEXCPT1.SQL* 创建。注意，必须指定 VALIDATE 来使用该关键字。

LOB_Storage_Clause

```
{LOB {(lob_item[,lob_item...]) STORE AS
   ([TABLESPACE tablespace]
        [{ENABLE | DISABLE} STORAGE IN ROW]
        [STORAGE Storage_Clause]
        [CHUNK integer]
        [PCTVERSION integer
        [CACHE | NOCACHE [LOGGING | NOLOGGING]]
       ) |
  LOB (lob_item) STORE AS [(lob_segname)]
   [([TABLESPACE tablespace]
     [{ENABLE | DISABLE} STORAGE IN ROW]
     [STORAGE Storage_Clause]
     [CHUNK integer]
     [PCTVERSION integer
```

```
    [CACHE | NOCACHE [LOGGING | NOLOGGING]]
]
```

指定应用于表、分区或子分区中的LOB数据段的存储参数。

关键字

LOB

指定为列出的LOB项目提供的LOB存储参数。任何未列出的LOB项目将使用表、分区或子分区相同的存储参数。注意，Oracle利用这一关键字为每一条LOB项目自动创建系统管理索引。

STORE AS

在这个关键字之后立即应用的存储参数。

TABLESPACE

指定存储LOB的表空间名。

ENABLE STORAGE IN ROW

指定如果LOB数据的大小大约小于4000字节时，可以将其存储为数据行。该选项是默认行为。

DISABLE STORAGE IN ROW

指定LOB数据一直存储在数据行外部。

STORAGE

为LOB段指定存储参数。要了解详细介绍，参见本章后面的“Storage_Clause”。

CHUNK

指定应该为LOB处理分配整数字节。注意，该整数将是Oracle块大小的倍数。

PCTVERSION

指定用于创建新LOB版本使用的LOB存储空间的最大百分比。默认值为10%。

CACHE

指定为了快速访问而将LOB数据驻留在内存中。

NOCACHE

指定没有LOB数据驻留在内存中。该选项是默认行为。

LOGGING

指定创建 LOB 存储时书写恢复日志记录。该选项是默认行为。

NOLOGGING

指定创建 LOB 存储时不书写恢复日志记录。在数据库失败的情况，不能应用日志文件恢复该操作，而且必须重新创建该对象。该选项加速了创建数据库对象的过程。注意，如果没有指定 CACHE 的话，就不能指定该选项。

STORE AS

指定 LOB 段的名字，只能在列出单个 LOB 项目时使用。

Partition_Clause

```
{PARTITION BY RANGE (column[,column...])
   PARTITION [partition_name] VALUES LESS THAN (value_list)
     [LOB_Storage_Clause]
     [Physical_Attributes_Clause]
     [TABLESPACE tablespace_name] [LOGGING | NOLOGGING]
     [{SUBPARTITIONS integer [STORE IN (tablespace_name[,tablespace_name ...]) |
        (SUBPARTITION [subpartition_name] [TABLESPACE tablespace_name]
           [LOB_Storage_Clause]
     [, SUBPARTITION [subpartition_name] [TABLESPACE tablespace_name]
           [LOB_Storage_Clause] ...]
      }]
  [, PARTITION [partition_name] VALUES LESS THAN (value_list)
     [LOB_Storage_Clause]
     [Physical_Attributes_Clause]
     [TABLESPACE tablespace_name] [LOGGING | NOLOGGING]
     [{SUBPARTITIONS integer [STORE IN (tablespace_name[,tablespace_name ...]) |
        (SUBPARTITION [subpartition name] [TABLESPACE tablespace_name]
           [LOB_Storage_Clause]
     [, SUBPARTITION [subpartition_name] [TABLESPACE tablespace_name]
           [LOB_Storage_Clause] ...]
     }] ...]  |
PARTITION BY RANGE (column[,column ...])
   [SUBPARTITION BY HASH (column[,column]
     [SUBPARTITIONS integer [STORE IN (tablespace_name[,tablespace_name ...]]]]
  {PARTITION [partition_name] VALUES LESS THAN (value_list)
     [LOB_Storage_Clause]
     [Physical_Attributes_Clause]
     [TABLESPACE tablespace_name] [LOGGING | NOLOGGING]
     [{SUBPARTITIONS integer [STORE IN (tablespace_name[,tablespace_name ...]) |
        (SUBPARTITION [subpartition_name] [TABLESPACE tablespace_name]
          [LOB_Storage_Clause [, LOB_storage_clause ...]
          [, SUBPARTITION [subpartition_name] [TABLESPACE tablespace_name]
          [LOB_Storage_Clause [, LOB_storage_clause ...]
      }]
```

```
   [, PARTITION [partition_name] VALUES LESS THAN (value_list)
     [LOB_Storage_Clause]
     [Physical_Attributes_Clause]
     [TABLESPACE tablespace_name] [LOGGING | NOLOGGING]
      [{SUBPARTITIONS integer [STORE IN (tablespace_name[,tablespace_name ...]) |
         (SUBPARTITION [subpartition_name] [TABLESPACE tablespace_name]
           [LOB_Storage_Clause]
          [, SUBPARTITION [subpartition_name] [TABLESPACE tablespace_name]
           [LOB_Storage_Clause] ...]
     }] ...]  |
PARTITION BY HASH (column[,column ...])
 [{PARTITIONS integer [STORE IN (tablespace_name[,tablespace_name ...]) |
    (PARTITION [partition_name] [TABLESPACE tablespace_name] [LOB_Storage_Clause]
     [, PARTITION [partition_name] [TABLESPACE tablespace_name] [LOB_Storage_Clause]
        ...]
}
```

指定分区参数，而且也用于定义返回分区、组合分区或散列分区。

关键字

PARTITION BY RANGE

指定根据被列出的列范围对表进行分区。

PARTITION... VALUES LESS THAN

指定分区名和一个或多个值（与PARTITION BY RANGE关键字的列清单是一一对应的关系）充当分区中包括的最大值。该值列表可以包括MAXVALUE，它代表了给定列的最高可能值。

LOB_Storage_Clause

指定该分区的LOB存储参数。要了解详细信息，参见本章前面的“LOB_Storage_Clause”。

Physical_Attributes_Clause

指定该分区的物理属性。参见本章后面的“Physical_Attributes_Clause”。

TABLESPACE

指定存储分区或子分区的表空间名。

LOGGING

指定在对象创建时要书写恢复日志记录。该选项是默认值。

NOLOGGING

指定在对象创建时不书写恢复日志记录。在数据库失败的情况下，不能应用日

志文件恢复该操作，必须重新创建该对象。该选项加速了创建数据库对象的过程。

SUBPARTITIONS

指定该分区将创建子分区。

STORE IN

指定要创建子分区的一个或多个表空间名。

SUBPARTITION

用名字指定子分区。

PARTITION BY HASH

指定散列分区将在提供的列清单的基础上创建子分区。

PARTITIONS

指定要被创建的 *integer* 个分区。

PARTITION

在 *partition_name* 中指定分区名。如果省略 *partition_name*，则 Oracle 使用 SYS_P*nnn* 的格式分配名字。

注意：分区是一个额外成本选项，而且只可以用于 Oracle 企业服务器产品。

Physical_Attributes_Clause

```
[INITRANS integer]
[MAXTRANS integer]
[PCTFREE integer]
[PCTUSED integer]
```

指定 Oracle 块中影响应用空间的架构对象属性。

关键字

INITTRANS

指定分配给对象的每一块的事务条目数量。允许值范围在 1~255 之间变动，而且正常情况下不应该改变默认值 2。

MAXTRANS

指定可以更新对象块的并发事务的最大数量。该值在1~255之间变动，而且正常情况下不应该改变默认值。默认值是Oracle块大小的函数。

PCTFREE

指定在每个数据块中为将来更新对象保留的自由空间百分比。允许的范围值为0~99。默认值为10。

PCTUSED

指定每个数据块中Oracle试图填充的空间百分比，允许的范围值为0~99，默认值为40。

Storage_Clause

```
STORAGE (
  [INITIAL integer [K | M]]
  [NEXT integer [K | M]]
  [MINEXTENTS integer]
  [MAXEXTENTS [integer | UNLIMITED] ]
  [PCTINCREASE integer ]
  [FREELISTS integer ]
  [FREELIST GROUPS integer ]
  [OPTIMAL [integer [K | M]]]
  [BUFFER_POOL {KEEP | RECYCLE | DEFAULT}]
 )
```

指定Oracle表空间中的存储空间怎样分配给单个对象。

关键字

STORAGE

指定数据库对象的物理存储特性。

INITIAL

以字节、KB或MB的形式指定数据库对象第一次扩展的空间大小。如果不是数据库块的整数倍，则该大小将被向上舍入为数据库块大小的倍数。

NEXT

以字节、KB或MB的形式指定数据库对象下一次扩展空间分配的大小。如果不是数据库块的整数倍，则该大小将被向上舍入为数据库块大小的倍数。

MINEXTENTS

指定创建数据库对象时，分配的扩展空间的大小。除了回退段之外，最小和默认值都是 1，回退段的默认值为 2。

MAXEXTENTS

指定为数据库对象分配的扩展空间的最大数量。默认值随数据库块的大小而变动。如果指定关键字 UNLIMITED，则表示没有上限。

PCTINCREASE

指定每一个扩展在前一个扩展上增长的百分比。默认值为50，意味着每一个扩展空间将比前一个扩展大 1.5 倍。

FREELISTS

为该数据库对象指定每一个自由列表组中包含的自由列表的数量。默认值为1，最大值由数据库块的大小而定。

FREELIST GROUPS

为数据库对象指定自由列表的分组数量。默认值为1。该参数只能与并行服务器选项一起使用。

OPTIMAL

只用于回退段，整数指定了Oracle试图通过取消未使用的回退段扩展空间维护优化空间大小。如果不指定大小，则回退段将不会减小。大家在省略OPTIMAL从句时得到的默认行为是从来不缩减回退段的大小。

BUFFER_POOL

指定怎样将架构对象分配到缓冲池中。KEEP 意味着将对象分配到 KEEP 缓冲池中，而且可能的情况下将永远保留在内存中。RECYCLE 意味着将该对象分配到 RECYCLE 缓冲池中，而且只要不需要时就从内存中删除。DEFAULT 意味着将对象分配到DEFAULT缓冲池中，而且为重新使用的缓存将应用最近最少使用（least-recently-used，LRU）标准算法。注意，必须用 DBA 优先使用来配置 KEEP 和 RECYCLE 池。

Table_Constraint_Clause

```
[CONSTRAINT constraint_name]
{[UNIQUE | PRIMARY KEY] (column[,column ...])] |
 [CHECK (condition) ] |
```

```
 [FOREIGN KEY (column[,column ...])
    REFERENCES [schema.]table [(column[,column ...])]
    [ON DELETE {CASCADE | SET NULL}]]
}
 [Constraint_State_Clause]
```

定义表上的约束。

关键字

CONSTRAINT

指定约束名。如果省略，Oracle以SYS_C*nnn*的形式分配名字，这里的*nnn*代表整数。

UNIQUE

指定列中的值不得与本表中其他行的相同列的值重复。列值或约束中的列必须在表的每一行惟一。但是，要注意不止一列包含NULL值。

PRIMARY KEY

指定由column所标识的列将在其被定义的表中充当主键。主键列不能包含NULL，而且表中其他行的主键不能与这一行的主键值相同。

CHECK

指定为了满足约束必须求值为TRUE或NULL的条件。

FOREIGN KEY

指定本表参与外部键参照完整性关系的一列或多列。

REFERENCES

指定被外部键约束引用的表和列。

ON DELETE CASCADE

指定当包含主键或惟一键的行被删除时，依赖的外部键也将被自动删除。

ON DELETE SET NULL

指定当包含主键或惟一键的行被删除时，依赖的外部键也将被自动改变为NULL。

Constraint_State_Clause

指定在表的数据上怎样应用约束。参见本章前面的“Constraint_State_Clause”。

第五章

SQL 函数

Oracle 实现了大量嵌入式函数，大家可以从 SQL 或 PL/SQL 中调用。如果要使用这些语言，就必须掌握这些函数。如果要使用 SQL，也需要处理大量的聚集函数。所有这些函数都在本章中进行描述，并用例子进行说明。许多例子显示了怎样通过 SQL*PLUS 调用这些函数；要了解更多信息，请参见第六章。

本章描述的 SQL 函数分为以下几类：

- 聚集函数
- 数字函数
- 字符函数
- 日期函数
- 转换函数
- 其他（混合）函数

注意：有关函数参数的注释按顺序进行。每一个函数描述包含了语法块，显示了可以在函数中传递的参数。在许多情况下，参数描述很清楚，我也就对它们不做特别描述。ABS 函数就是这种类型描述的一个例子。如果大家读了函数的描述并看了例子，就会清楚怎样使用该函数。在必要时，我列出了其他函数的详细参数描述。

聚集函数

聚集（aggregate）函数是设计为允许大家汇总表或视图的多行数据的SQL函数。这些聚集函数，其中很多都对数据仓库应用非常有用，它们只在SQL语句中使用时才有效。与其他的嵌入式函数不同，它们不能从PL/SQL表达式中直接调用（第七章介绍了有关PL/SQL编程的更多信息）。表5-1列出了可用的聚集函数。

表5-1：Oracle的聚集函数

函数	描述
AVG	返回行集中一列的平均值
COUNT	返回行集中一列的非空值的数量
GROUPING	允许大家使用Oracle的ROLLUP和CUBE扩展时在查询中插入小计（合计）
MAX	返回行集中一列的最大值
MIN	返回行集中一列的最小值
STDDEV	返回行集中一列的所有值的标准偏差（stardard deviation）
SUM	返回行集中一列的合计值
VARIANCE	返回行集中一列的方差值

GROUP BY

在使用聚集函数时，GROUP BY从句引起Oracle报告一列中每条截然不同值的结果。下面的例子是由数据库用户使用GROUP BY取出对象计数：

```
SQL> SELECT owner, COUNT(object_name)
  2  FROM dba_objects
  3  GROUP BY owner;

OWNER                          COUNT(OBJECT_NAME)
------------------------------ ------------------
CTXSYS                                        164
DBSNMP                                          4
GNIS                                            2
MDSYS                                         137
OEMREP                                        230
...
```

GROUP BY 从句引起结构按照不同的散列表元进行排序 —— 每条记录都代表 GROUP BY 列中一条不同的值。COUNT 函数报告了每一个这些散列表元的 OBJECT_NAME 列的计数。

如果想在特定列中对行计数，而不是求值，大家可以使用星号（*）作为COUNT的参数。例如:

```
SELECT COUNT(*) FROM dba_objects
```

在对列值计数时，列中是否包含 NULL 值是有差别的。通常将 NULL 值忽略。在 COUNT 的情况下，NULL 值不被计数。使用星号则使所有行都参与计数，包括那些 NULL 值。

DISTINCT 和 ALL

大部分聚集函数可以选择性地在参数列表中使用DISTINCT和ALL关键字。这些关键字允许大家控制重复列值处理的方式，它们具有下列意义:

DISTINCT

使聚集函数只查找不同的值，忽略重复值。

ALL

使聚集函数查找所有值，包括重复值。

考虑查询结果如下的表:

```
SQL> SELECT * FROM agg_demo;

         X
----------
         1
         2
         2
```

agg_demo表包含了3行，在X列中存储了两个不同的值。两行值为2，一行值为1。注意，下面的例子中 DISTINCT 和 ALL 关键字的影响:

```
SQL> SELECT COUNT(x), COUNT(DISTINCT x), COUNT(ALL x)
  2  FROM agg_demo;
```

```
  COUNT(X) COUNT(DISTINCTX) COUNT(ALLX)
---------- --------------- ----------
         3               2          3
```

计算所有值是默认行为。使用DISTINCT关键字只计算截然不同的值。在这个例子中，有两个不同的值：1和2。ALL关键字是隐含的。

Aggregate 函数和 NULL 值

大部分情况下，聚集函数忽略NULL值。考虑前一节中例子的扩展例子。这次，agg_demo 表包含一个NULL值：

```
SQL> SELECT * FROM agg_demo;

         X
----------
         1
         2
         2
*null*
```

注意：这一版本的agg_demo表用在本章中给定的大部分聚集函数的例子中。

使用COUNT函数计算列X中值的数量，产生下面的结果：

```
SQL> SELECT COUNT(x), COUNT(DISTINCT x), COUNT(ALL x)
  2  FROM agg_demo;

  COUNT(X) COUNT(DISTINCTX) COUNT(ALLX)
---------- --------------- ----------
         3               2          3
```

注意，没有一种情况计算了NULL值，所有NULL值都被忽略了。

注意：有两种情况聚集函数不忽略NULL值。一种特殊的情况是在使用COUNT(*)时，Oracle计算行，而不是列的值。GROUPING函数则是另一种情况。GROUPING在GROUP BY列中摘要了包含NULL值的一组行的数据。

AVG

AVG([DISTINCT | ALL] *expr*)

计算列、被查询返回的行集合的表达式或者由 GROUP BY 从句指定的行集合的平均值。

示例

```
SQL> SELECT AVG(x), AVG(DISTINCT x), AVG(ALL x)
  2  FROM agg_demo;

    AVG(X) AVG(DISTINCTX)  AVG(ALLX)
---------- -------------- ----------
1.66666667            1.5 1.66666667
```

COUNT

COUNT({*|[DISTINCT | ALL] *expr*})

计算由查询返回的列或表达式的行集合的值的数量，或者计算由 GROUP BY 从句指定的行集合的数量。

示例

```
SQL> SELECT owner, object_type, COUNT(*)
  2  FROM dba_objects
  3  GROUP BY owner, object_type;

OWNER                          OBJECT_TYPE          COUNT(*)
------------------------------ ------------------ ---------
CTXSYS                         INDEX                     35
CTXSYS                         INDEXTYPE                  1
CTXSYS                         LIBRARY                    2
CTXSYS                         OPERATOR                   2
CTXSYS                         PACKAGE                   29
CTXSYS                         PACKAGE BODY              24
```

GROUPING

GROUPING (*expr*)

决定是否特定列或表达式的 NULL 值代表一添加行或者值的合理分组。

GROUPING 函数在 Oracle8*i* 中引入，而且是为 Oracle8*i* 中引入的 CUBE 和 ROLLUP 操作符使用的 SELECT 语句而设计的。CUBE 和 ROLLUP 都使得在查询中插入额外的行到一组摘要记录中。这些额外的行在 GROUP BY 列中包含 NULL 值。如果使

用CUBE或ROLLUP返回的附加行返回了NULL值，则GROUPING返回1；否则，返回0。

示例

本例子中的查询根据所有者和类型返回数据库对象的数量。ROLLUP关键字使得Oracle为每个所有者插入一附加行，该行为该所有者插入提供合计数。该行在OBJECT_TYPE列中具有NULL值。附加行被插入到给定了全部所有者的对象总计数的行集末尾。该行有OWNER和OBJECT_TYPE列的NULL值。

在这个特定的查询中，OWNER和OBJECT_TYPE列通常永远不会为NULL，所以，大家可以安全地将NULL值的出现，当作被插入行的小计行的显示符号。但是，这不代表所有查询都是如此。因此，需要一个可靠机制决定是否查询返回的行，代表了附加的rollup行作为数据库信息的相反信息。录入GROUPING函数。在这一例子中，在OBJECT_TYPE列上应用了GROUPING函数，而且它将返回1显示包含了NULL对象类型的行，因为该行作为使用ROLLUP关键字的结果被添加进来：

```
SQL> SELECT owner, object_type, COUNT(*), GROUPING(object_type)
  2  FROM dba_objects
  3  GROUP BY ROLLUP (owner, object_type);

OWNER        OBJECT_TYPE          COUNT(*) GROUPING(OBJECT_TYPE)
------------ ------------------- --------- --------------------
CTXSYS       INDEX                      35                    0
CTXSYS       INDEXTYPE                   1                    0
CTXSYS       LIBRARY                     2                    0
CTXSYS       OPERATOR                    2                    0
CTXSYS       PACKAGE                    29                    0
CTXSYS       PACKAGE BODY               24                    0
CTXSYS       PROCEDURE                   1                    0
CTXSYS       SEQUENCE                    3                    0
CTXSYS       TABLE                      26                    0
CTXSYS       TYPE                        4                    0
CTXSYS       TYPE BODY                   3                    0

OWNER        OBJECT_TYPE          COUNT(*) GROUPING(OBJECT_TYPE)
------------ ------------------- --------- --------------------
CTXSYS       UNDEFINED                   1                    0
CTXSYS       VIEW                       33                    0
CTXSYS       *null*                    164                    1
DBSNMP       SYNONYM                     4                    0
DBSNMP       *null*                      4                    1
```

```
GNIS          INDEX                 1                 0
GNIS          TABLE                 1                 0
GNIS          *null*                2                 1
*null*        *null*            12185                 1
```

MAX

```
MAX([DISTINCT | ALL] expr)
```

计算由查询或特定 GROUP BY 从句返回的行集表达式或是一列的最大值。

示例

该例子通过几种方式使用了 MAX 函数找到 agg_demo 表的 X 列的最大值:

```
SQL> SELECT MAX(x), MAX(DISTINCT x), MAX(ALL X)
  2  FROM agg_demo;

    MAX(X) MAX(DISTINCTX)  MAX(ALLX)
---------- -------------- ----------
         2              2          2
```

注意: 使用ALL或DISTINCT并不影响MAX函数返回的值。无论MAX查找每个值的一个或多个具体值都无关紧要。无论在哪种情况下，返回的最大值都相同。

MIN

```
MIN([DISTINCT | ALL] expr)
```

计算由查询或特定的 GROUP BY 从句返回的行集表达式或一列的最小值。

示例

该例子通过几种方式使用了 MIN 函数找到 agg_demo 表的 X 列的最小值:

```
SQL> SELECT MIN(x), MIN(DISTINCT x), MIN(ALL X)
  2  FROM agg_demo;

    MIN(X) MIN(DISTINCTX)  MIN(ALLX)
---------- -------------- ----------
         1              1          1
```

注意：使用ALL或DISTINCT并不影响MIN函数返回的值。无论MIN查找每个值的一个或多个具体值都无关紧要。无论在哪种情况下，返回的最小值都相同。

STDDEV

STDDEV([DISTINCT | ALL] *expr*)

计算由查询或特定的GROUP BY从句返回的行集表达式或一列的标准偏差值。

示例

该例子使用STDDEV函数找到agg_demo表的X列中值的许多标准偏差：

```
SQL> SELECT STDDEV(x), STDDEV(DISTINCT x), STDDEV(ALL X)
  2  FROM agg_demo;

 STDDEV(X) STDDEV(DISTINCTX) STDDEV(ALLX)
---------- ---------------- ------------
 .577350269        .707106781   .577350269
```

注意：截然不同值的标准偏差与其他两种方法的值不同；标准偏差由观察数据的数量决定，而且DISTINCT返回的值比其他两种方法返回的值都要少。

SUM

SUM([DISTINCT | ALL] *expr*)

计算由查询或特定的GROUP BY从句返回的行集表达式或一列的合计值。

示例

```
SQL> SELECT SUM(x), SUM(DISTINCT x), SUM(ALL x)
  2  FROM agg_demo;

    SUM(X) SUM(DISTINCTX)  SUM(ALLX)
---------- -------------- ----------
         5              3          5
```

VARIANCE

VARIANCE([DISTINCT | ALL] *expr*)

计算由查询或特定的 GROUP BY 从句返回的行集表达式或一列的值的方差。

示例

```
SQL> SELECT VARIANCE(x), VARIANCE(DISTINCT x), VARIANCE(ALL x)
  2  FROM agg_demo;

VARIANCE(X) VARIANCE(DISTINCTX) VARIANCE(ALLX)
----------- ------------------- --------------
 .333333333                  .5     .333333333
```

注意：像标准偏差一样，方差由观察数据的数量而定，所以DISTINCT返回的行较少，而且因此导致了方差不同。

数字函数

数字函数包括那些采用数字作为参数，而且返回数字值的函数。许多这些函数与三角有关，也有算术函数和许多常用函数。

ABS

ABS(*n*)

返回数字的绝对值：任何值的负号都被删除。

示例

```
SQL> SELECT ABS(-1), ABS(1)
  2  FROM dual;

   ABS(-1)     ABS(1)
---------- ----------
         1          1
```

ACOS

ACOS(*n*)

返回在-1~1之间的值的反余弦。该函数与余弦函数COS的输出相反。结果以角度的弧度表示，范围在0~π之间。

示例

```
SQL> SELECT ACOS(-1), ACOS(0), ACOS(1)
  2  FROM dual;

  ACOS(-1)    ACOS(0)    ACOS(1)
---------- ---------- ----------
3.14159265 1.57079633          0
```

ASIN

ASIN(*n*)

返回在-1~1之间的值的反正弦。该函数与正弦SIN函数的输出相反。结果以角度的弧度表示，范围在-π/2 ~π/2之间。

示例

```
SQL> SELECT ASIN(-1), ASIN(0), ASIN(1)
  2  FROM dual;

  ASIN(-1)    ASIN(0)    ASIN(1)
---------- ---------- ----------
-1.5707963          0 1.57079633
```

ATAN

ATAN(*n*)

返回一个值的反正切。该函数与正切函数TAN的输出相反。结果以角度的弧度表示，范围在-π/2 ~π/2之间。

示例

```
SQL> SELECT ATAN(.781285627), TAN(.663225116)
```

```
  2  FROM dual;

ATAN(.781285627) TAN(.663225116)
---------------- ---------------
      .663225116       .781285627
```

ATAN2

ATAN2(*n*,*m*)

返回 *n/m* 的反正切。换句话说，ATAN2(*n*,*m*)与 ATAN(*n/m*)相同。结果以角度的弧度表示，范围在 -π/2 ~π/2 之间。

示例

```
SQL> SELECT ATAN2(.4,2), ATAN(.2)
  2  FROM dual;

ATAN2(.4,2)   ATAN(.2)
----------- ----------
  .19739556  .19739556
```

CEIL

CEIL(*n*)

返回大于或等于输入的最小整数值。如果 *n* 已经是整数，就返回 *n*。

示例

```
SQL> SELECT CEIL(10), CEIL(10.5), CEIL(-10.5)
  2  FROM dual;

  CEIL(10) CEIL(10.5) CEIL(-10.5)
---------- ---------- -----------
        10         11         -10
```

注意负数，符号与大小相关：-10 比 -10.5 大。

COS

COS(*n*)

返回余弦值，这里的角度用弧度表示。

示例

```
SQL> SELECT COS(0)
  2  FROM dual;

    COS(0)
----------
         1
```

COSH

COSH(n)

返回角度的双曲余弦，这里的角度用弧度表示。

示例

```
SQL> SELECT COSH(0)
  2  FROM dual;

   COSH(0)
----------
         1
```

EXP

EXP(*n*)

返回 *e* 的 *n* 次方。

示例

常量 *e* 约等于2.71828183。下面的例子显示了计算 *e* 的3次方的两种方式。结果稍微有点不同，因为EXP函数采用的 *e* 的近似值比另一表达式中使用的要更精确。

```
SQL> SELECT EXP(3), 2.71828183*2.71828183*2.71828183
  2  FROM dual;

    EXP(3) 2.71828183*2.71828183*2.71828183
---------- -------------------------------
20.0855369                        20.085537
```

FLOOR

FLOOR(*n*)

返回小于等于输入值的最大整数。

示例

```
SQL> SELECT FLOOR(9.9), FLOOR(-9.9)
  2  FROM dual;

FLOOR(9.9) FLOOR(-9.9)
---------- -----------
         9         -10
```

与CEIL一样，与符号有关，-10 比 -9.9要小。

GREATEST

GREATEST(*expr* [,*expr*...])

返回所提供的参数列表的最大值。GREATTEST 函数可以用于字符串和日期，也可以用于数字。

示例

返回数字清单的最大值:

```
SQL> SELECT GREATEST(1,2,3) FROM dual;

GREATEST(1,2,3)
---------------
              3
```

返回字符串列表的最大值:

```
SQL> SELECT GREATEST('One','Two') FROM dual;

GRE
---
Two
```

返回日期列表的最大值（最近的日期):

```
SQL> SELECT GREATEST(TO_DATE('11/15/1961','MM/DD/YYYY'),
  2                  TO_DATE('12/29/1988','MM/DD/YYYY'))
  3  FROM dual;

GREATEST(
---------
29-DEC-88
```

注意：如果混合使用数据类型，则Oracle在选择最大值之前使用第一个参数的数据类型作为基准类型，并将所有其他参数转换为（如果可能的话）该类型。

LEAST

LEAST(*expr* [,*expr*...])

返回提供参数列表的最小值。GREATTEST函数可以用于字符串和日期，也可以用于数字。

示例

```
SQL> SELECT LEAST(1,2,3) FROM dual;

LEAST(1,2,3)
------------
           1
```

LN

LN(*n*)

返回一个数字的自然对数。LN函数和EXP函数是互补的。

示例

```
SQL>SELECT LN(100), EXP(4.60517019) FROM dual;

   LN(100) EXP(4.60517019)
---------- ---------------
4.60517019             100
```

LOG

LOG(*m*,*n*)

返回以 *m* 为底的 *n* 的对数。

参数

m 指定不为 0 或 1 的正数代表对数的底。

n 指定数字，必须为正数，将计算该数以 *m* 为底的对数值。

示例

```
SQL> SELECT LOG(10,10000) FROM dual;

LOG(10,10000)
-------------
            4
```

MOD

MOD (*m*,*n*)

返回当 *m* 被 *n* 除时的余数。

示例

```
SQL> SELECT MOD(18,12), MOD(30,12), MOD(30,30)
  2  FROM dual;

MOD(18,12) MOD(30,12) MOD(30,30)
---------- ---------- ----------
         6          6          0
```

在第一种情况下，18 除以 12 等于 1 余 6。第 2 种情况，30 除以 12 等于 2 余 6。第三种情况，30 除以 30 等于 1 余 0，所以没有余数。

POWER

POWER(*m*,*n*)

产生 m 的 n 次方。换句话说，该函数返回 m^n 。

参数

m 指定非0数字。

n 指定幂。如果 m 为正数，则 n 可以为正数或负数。如果 m 为负数，则 n 必须为整数。

示例

```
SQL> SELECT POWER(10,3), POWER(-10,-3), POWER(-10,3)
  2  FROM dual;

POWER(10,3) POWER(-10,-3) POWER(-10,3)
----------- ------------- ------------
       1000          -.001        -1000
```

注意： m^{-n} 等于 $1/m^n$。

ROUND

ROUND(*n*,*m*)

对指定数字的十进制位数进行四舍五入。

参数

n 指定要四舍五入的值。

m 指定结果中保留的十进制位数。该数字必须为整数。m 的负数值将根据该值对小数点左边进行四舍五入。

示例

```
SQL> SELECT ROUND(123.45), ROUND(123.45,1), ROUND(123.45,-1)
  2  FROM dual;

ROUND(123.45) ROUND(123.45,1) ROUND(123.45,-1)
```

```
------------ -------------- ---------------
         123          123.5             120
```

SIGN

SIGN(*n*)

显示 *n* 的符号值。SIGN 返回的数字将是下列数字之一:

-1 该数字为负数

0 该数字为 0

1 该数字为正数

示例

```
SQL> SELECT SIGN(76), SIGN(0), SIGN(-76.17)
  2  FROM dual;

  SIGN(76)    SIGN(0)  SIGN(-76)
---------- ---------- ----------
         1          0         -1
```

SIN

SIN(*n*)

返回角度 *n* 的正弦，*n* 必须用弧度表示。

示例

```
SQL> SELECT SIN(3.14), SIN(0)
  2  FROM dual;

 SIN(3.14)     SIN(0)
---------- ----------
.001592653          0
```

SQRT

SQRT(*n*)

返回 *n* 的平方根，*n* 必须为正数。

示例

```
SQL> SELECT SQRT(100), 10*10, SQRT(81), 9*9 FROM dual;

 SQRT(100)      10*10   SQRT(81)        9*9
---------- ---------- ---------- ----------
        10        100          9         81
```

TAN

TAN(*n*)

返回角度*n*的正切，*n*必须用弧度表示。

示例

```
SQL> SELECT TAN(.291456794), ATAN(.3) FROM dual;

TAN(.291456794)   ATAN(.3)
--------------- ----------
      .299999999 .291456794
```

TANH

TANH(*n*)

返回角度*n*的双曲正切，*n*必须用弧度表示。

示例

```
SQL> SELECT TANH(0), TANH(3.14) FROM dual;

   TANH(0) TANH(3.14)
---------- ----------
         0 .996260205
```

TRUNC

TRUNC (*n* [,*m*])

按照指定的小数位数截去特定的数字。

参数

n 指定被截去的数字。

m 指定在结果中保留的小数位数。截去所有的小数位数是默认值。负数值m导致从数字的小数点左边开始截取。

示例

```
SQL> SELECT TRUNC(99.99), TRUNC(99.99,1), TRUNC(99.99,-1)
  2  FROM dual;

TRUNC(99.99) TRUNC(99.99,1) TRUNC(99.99,-1)
------------ -------------- ---------------
          99           99.9              90
```

注意：该函数在小数点位置进行截取，而且不进行四舍五入。

字符函数

在手册上，Oracle将字符函数分成两类——返回字符值和返回数字值的函数。SUBSTR函数返回大字符串的子字符串，是前一种类型的例子。LENGTH函数返回字符串的长度，是后面一种类型的例子。为了方便查阅，本书将两种类型组合成一节进行讨论。

ASCII

ASCII(*char*)

基于数据库的字符集有效返回字符的十进制表示法。真实的ASCII值只有在数据库使用7位ASCII字符集时能够被返回。注意，ASCII函数的参数可以是字符串，也可以是单个字符。但是，如果提供字符串，则ASCII函数返回字符串基于第一个字符的值。

示例

```
SQL> SELECT ASCII('D'), ASCII('Dave') FROM dual;
```

```
ASCII('D') ASCII('DAVE')
---------- -------------
        68            68
```

CHR

CHR(*n* [USING NCHAR_CS])

返回与特定数字相关的数据库字符集的字符。如果指定USING NCHAR_CS，则数据库使用国际字符集。

示例

```
SQL> SELECT CHR(68), CHR(68 USING NCHAR_CS) FROM dual;

C C
- -
D D
```

注意：该例子使用ASCII字符集产生。

CONCAT

CONCAT(*string1, string2*)

连接两个输入字符串并返回结果。CONCAT函数等于连接操作符（||）；要了解该操作符的更详细的信息，请参见第一章。

示例

```
SQL> SELECT CONCAT('Good ','Morning'), 'Good ' || 'Morning'
  2  FROM dual;

CONCAT('GOOD 'GOOD'||'MOR
------------ ------------
Good Morning Good Morning
```

GREATEST

参见描述数字函数的部分对GREATEST的定义。

INITCAP

INITCAP(*string*)

将字符串中每个单词的第一个字母改为大写，而所有其他字母改为小写。

示例

```
SQL> SELECT INITCAP('good MORNING, dave') FROM dual;

INITCAP('GOODMORNI
------------------
Good Morning, Dave
```

INSTR

INSTR(*string1*, *string2*[, *n*[, *m*]])

在 *string1* 中搜索 *string2* 并返回 *string1* 中 *string2* 开始的位置。

参数

string1

指定想要搜索的字符串。

string2

指定想要找到的字符串。

n 指定从该字符位置开始查找。默认值是从第一个字符开始查找，就是位置1。使用负数指定起始位置从字符串 1 的右边开始，而不是从左边。

m 指定想要找到该字符串第几次出现的场合（如果出现多于 1 次），默认情况是找到第一次出现的场合。

示例

```
SQL> SELECT INSTR('easy come, easy go','easy') FROM dual;

INSTR('EASYCOME,EASYGO','EASY')
-------------------------------
                              1
```

```
SQL> SELECT INSTR('easy come, easy go','easy',1,2) FROM dual;

INSTR('EASYCOME,EASYGO','EASY',1,2)
-----------------------------------
                                 12
```

INSTRB

INSTRB(*string1*, *string2*[, *n*[, *m*]])

搜索 *string1*、找到 *string2*、并返回 *string1* 中 *string2* 开始的位置。

INSTRB 与 INSTR 相似，但返回值代表 *string1* 的字节索引，而不是字符索引。当使用单字节字符集时，INSTR 和 INSTRB 返回的结果相同；只有使用多字节字符集时存在一些差别。

LEAST

参见描述数字函数的部分中对 LEAST 的定义。

LENGTH

LENGTH(*string*)

返回字符串中的字符数。

示例

```
SQL> SELECT LENGTH('Dave'), LENGTH('Brighten the corner where you are.')
  2  FROM dual;

LENGTH('DAVE') LENGTH('BRIGHTENTHECORNERWHEREYOUARE.')
-------------- ------------------------------------
             4                                   34
```

LENGTHB

LENGTHB(*string*)

以字节的形式返回字符串长度。一般情况LENGTHB 返回的结果与LENGTH 相同，但当使用多字节字符集时有所不同。

LOWER

LOWER(*string*)

将所有字符串转换成小写字符。

示例

```
SQL> SELECT LOWER('LIFE IS a Wonderful tHINg.') FROM dual;

LOWER('LIFEISAWONDERFULTHI
---------------- --------
life is a wonderful thing.
```

LPAD

LPAD(*string1*, *n*[, *string2*])

在 *string1* 的左边添加空格，或者重复填充由 *string2* 指定的字符串，直到结果字符串达到 *n* 个字符为止。

参数

string1

指定想要填充的字符串。

n 指定想在结果中填充的字符数。

string2

指定用于填充到输入字符串中以达到理想长度的字符序列。该参数默认为单个空格。

示例

```
SQL> SELECT LPAD('X',3), LPAD('X',3,'Y'), LPAD('RIGHT',12,'LEFT')
  2  FROM dual;

LPA LPA LPAD('RIGHT'
--- --- ------------
  X YYX LEFTLEFTRIGHT
```

LTRIM

LTRIM(*string1* [,*string2*])

从字符串左边删除指定字符。

参数

string1

指定要删除字符的字符串。

string2

指定包含要被删除的字符集的字符串。该参数默认为单个空格。当函数遇到字符串中没有提供的字符时，删除字符过程结束。

示例

```
SQL> SELECT LTRIM('   The default is to trim leading spaces.')
  2  FROM dual;

LTRIM('THEDEFAULTISTOTRIMLEADINGSPACES
--------------------------------------
The default is to trim leading spaces.

SQL> SELECT LTRIM('***You can trim other characters as well.','*')
  2  FROM dual;

LTRIM('***YOUCANTRIMOTHERCHARACTERSASW
--------------------------------------
You can trim other characters as well.

SQL> SELECT LTRIM('**@@*@You can even trim multiple characters.','*@')
  2  FROM dual;

LTRIM('**@@*@YOUCANEVENTRIMMULTIPLECHA
--------------------------------------
You can even trim multiple characters.
```

NLS_INITCAP

NLS_INITCAP(*string*[, 'NLS_SORT=*sort*'])

类似于INITCAP，但使用了国际字符集。

参数

string

指定输入的字符串。

sort

指定为所用语言指定大写规则的语言类别序列名。

示例

```
SQL> SELECT NLS_INITCAP('money','NLS_SORT=XFRENCH')
  2  FROM dual;

NLS_I
-----
Money
```

NLS_LOWER

NLS_LOWER(*string*[, 'NLS_SORT=*sort*'])

类似于LOWER，但使用国际字符集。

参数

sting

指定输入的字符串。

sort

指定为所用语言指定小写规则的语言类别序列名。

示例

```
SQL> SELECT NLS_LOWER('MONEY','NLS_SORT=XFRENCH')
  2  FROM dual;

NLS_L
-----
money
```

NLS_UPPER

NLS_UPPER(*string*[, 'NLS_SORT=*sort*'])

类似于UPPER，但使用国际字符集。

参数

string

指定输入的字符串。

sort

指定为所用语言指定大写规则的语言类别序列名。

示例

```
SQL> SELECT NLS_UPPER('money','NLS_SORT=XFRENCH')
  2  FROM dual;

NLS_L
-----
MONEY
```

NLSSORT

NLSSORT(*string*[, 'NLS_SORT=*sort*'])

返回用于代表使用语言类别序列分类值的字节字符串。

参数

string

指定字符串。

sort

指定语言类别序列名或关键字BINARY。如果使用BINARY，则NLSSORT返回输入字符串的确切字节值。

示例

```
SQL> SELECT NLSSORT('MONEY','NLS_SORT=XFRENCH') FROM dual;
```

```
NLSSORT('MONEY','NLS_SORT=XFRENCH')
-------------------------------------------
505A55288200010101010100

SQL> SELECT NLSSORT('money','NLS_SORT=XFRENCH') FROM dual;

NLSSORT('MONEY','NLS_SORT=XFRENCH')
-------------------------------------------
505A55288200020202020200
```

REPLACE

REPLACE(*string*, *search_string* [,*replacement_string*])

搜索字符串并用一个子字符串代替另一个子字符串。该函数也可以用于删除出现的子字符串。

参数

string

指定输入字符串和被搜索的字符串。

search_string

指定要搜索的子字符串。

replacement_string

指定想要用来代替*search_string*的字符串。如果省略该可选参数，则出现的所有 *search_string* 都将被删除。

示例

```
SQL> SELECT REPLACE('The sky is blue.','blue','red') FROM dual;

REPLACE('THESKY
---------------
The sky is red.

SQL> SELECT REPLACE('The sky is blue red.',' red') FROM dual;

REPLACE('THESKYI
----------------
The sky is blue.
```

RPAD

RPAD(*string1*, *n*[, *string2*])

在 *string1* 的右边填充空格或重复填充由 *string2* 指定的字符串，直到结果字符串达到 *n* 个字符为止。

参数

string1

指定想要填充的字符串。

n 指定结果中需要的字符数量。

string2

指定用于填充到输入字符串中的字符序列，从而达到理想长度。该参数默认为单个空格。

示例

```
SQL> SELECT RPAD('X',3), RPAD('X',3,'Y'), RPAD('RIGHT',12,'LEFT')
  2  FROM dual;

RPA RPA RPAD('RIGHT'
--- --- ------------
X   XYY RIGHTLEFTLEF
```

RTRIM

RTRIM(*string1* [,*string2*])

从字符串右边删除指定的字符。

参数

string1

指定想要从中删除字符的字符串。

string2

指定包含要被删除的字符集的字符串。默认值为单个空格。删除字符过程从右向左进行，当该函数遇到了不在该集合中的一个字符时就停止。

示例

```
SQL> SELECT RTRIM('The default is to trim trailing spaces.   ')
  2  FROM dual;

RTRIM('THEDEFAULTISTOTRIMLEADINGSPACES
--------------------------------------
The default is to trim trailing spaces.

SQL> SELECT RTRIM('You can trim other characters as well.***','*')
  2  FROM dual;

RTRIM('***YOUCANTRIMOTHERCHARACTERSASW
--------------------------------------
You can trim other characters as well.

SQL> SELECT RTRIM('You can even trim multiple characters. **@@*@','*@')
  2  FROM dual;

RTRIM('**@@*@YOUCANEVENTRIMMULTIPLECHA
--------------------------------------
You can even trim multiple characters.
```

SOUNDEX

SOUNDEX(*string*)

返回代表输入字符串数字语音发音的数字字符串。下面的步骤用于得到输出字符串:

1. 保留字符串的第一个字母。
2. 删除下列字母: a、e、h、I、o、u、w和 y。
3. 剩下的字母用表 5-2 中列出的数字代替。
4. 结果字符串被修整为 4 个字符。

使用SOUNDEX可以轻松地检索给定的字符串，因为大家不是一定要用完全正确的拼写方式才能得到匹配。

表 5-2: SOUNDEX 字母与数字的对应

字母	SOUNDEX 数字
b, f, p, v	1
c, g, j, k, q, s, x, z	2

表 5-2: SOUNDEX 字母与数字的对应（续）

字母	SOUNDEX 数字
d, t	3
l	4
m, n	5
r	6

示例

```
SQL> SELECT SOUNDEX('O''Reilly'), SOUNDEX('ORiley'), SOUNDEX('OH Riley')
  2  FROM dual;

SOUN SOUN SOUN
---- ---- ----
O640 O640 O640
```

SUBSTR

SUBSTR(*string*, *m* [,*n*])

返回字符串的一部分。

参数

string

指定输入字符串。

m 指定从*char*开始提取的第一个字符的索引。索引从1开始，是字符串左端的偏移量。大家可以使用负数来指定从字符串的右边开始偏移。

n 指定提取的字符数量。该参数可选。默认情况下，大家得到的是从位置*m*到字符串的末尾的所有字符。

示例

```
SQL> SELECT SUBSTR('OneTwoThree',7) FROM dual;

SUBST
-----
Three
```

```
SQL> SELECT SUBSTR('OneTwoThree',4,3) FROM dual;

SUB
---
Two

SQL> SELECT SUBSTR('OneTwoThree',-5) FROM dual;

SUBST
-----
Three
```

SUBSTRB

SUBSTRB(*string*, *m* [,*n*])

返回字符串的一部分。SUBSTRB 与 SUBSTR 相同，但 *n* 和 *m* 都指字节而不是字符。只有在字符串是基于多字节字符集的时候它们才有所区别。

TRANSLATE

TRANSLATE(*string*, *from_string*, *to_string*)

通过转换一种字符集为另一种字符集修改字符串。

string

输入字符串。

from_string

想要转换的字符集。

to_string

将代替 *from_string* 的新字符集。输入字符串中的每一个 *from_string* 字符都被 *to_string* 中的对应字符所代替。*from_string* 的字符如果在 *to_string* 中没有对应的元素，则该字符被删除。

示例

下面的例子将数字 0~9 转换成字母 a~j。在第 2 个例子中，*from_string* 包含了一个句点，而 *to_string* 则不包含。该状况使得句点被删除：

```
SQL> SELECT TRANSLATE('123.45','0123456789','abcdefghij') FROM dual;

TRANSL
------
bcd.ef

SQL> SELECT TRANSLATE('123.45','0123456789.','abcdefghij') FROM dual;

TRANS
-----
bcdef
```

TRIM

```
TRIM([LEADING | TRAILING | BOTH]
     [trim_character FROM] string)
```

从字符串中删除前面和后面的空格（或其他字符）。

参数

LEADING

指定只去掉前面的空格。

TRAILING

指定只去掉后面的空格。

BOTH

指定既去掉前面的空格也去掉后面的空格。

trim_character

指定去掉一些其他字符而不是空格。

string

指定输入字符串。

示例

```
SQL> SELECT TRIM('   TEENA   ') FROM dual;

TRIM(
-----
TEENA
```

```
SQL> SELECT TRIM(LEADING '$' FROM '$123.45') FROM dual;

TRIM(L
------
123.45
```

UPPER

UPPER(*string*)

将所有字符转换为大写。

示例

```
SQL> SELECT UPPER('LIFE IS a Wonderful tHINg.') FROM dual;

UPPER('LIFEISAWONDERFULTHI
--------------------------
LIFE IS A WONDERFUL THING.
```

日期函数

日期函数在日期上操作或者返回日期值。一些函数返回当前日期和时间，将日期/时间值截取为特定单元，或者在两个日期之间计算月的个数。

本节的许多例子使用了TO_CHAR和TO_DATE函数。这些日期转换函数接受日期格式字符串作为参数。为了使例子尽可能简单易懂，我们假定默认的日期格式为'dd-Mon-yyyy'。在一些例子中，我们使用了不同的格式（例如，包括了时间），这些例子将用ALTER SESSION命令开始。

ADD_MONTHS

ADD_MONTHS(*d*, *n*)

在日期*d*上添加*n*个月。负数可以用于从*d*上减去*n*个月。下面的规则控制是否影响了月中的天数：

- 如果初始日期代表了月的最后一天，结果日期将被调整，从而也保证是该月的最后一天。

- 如果保持月的相同天为有效的日期，因为新的月份具有的天数比原始的要少（例如，一月有31天，而二月只有28天），则将调整天，从而满足新的月份。

如果这里显示的任何一条规则都没有应用，则月份的天将不会被改变。

示例

```
SQL> SELECT ADD_MONTHS(TO_DATE('1-Jan-2000'), 3) FROM dual;

ADD_MONTHS(
-----------
01-Apr-2000

SQL> SELECT ADD_MONTHS(TO_DATE('31-Jan-2000'), 1) FROM dual;

ADD_MONTHS(
-----------
29-Feb-2000

SQL> SELECT ADD_MONTHS(TO_DATE('29-Feb-2000'), -1) FROM dual;

ADD_MONTHS(
-----------
31-Jan-2000
```

GREATEST

参见数字函数一节中描述的GREATEST定义。

LAST_DAY

LAST_DAY(*d*)

返回日期*d*所在的月份中对应最后一天的日期。

示例

```
SQL> SELECT LAST_DAY(TO_DATE('13-FEB-2000')) FROM dual;

LAST_DAY(TO
-----------
28-Feb-2000
```

LEAST

参见数字函数一节中有关LEAST的定义。

MONTHS_BETWEEN

MONTHS_BETWEEN(*d1,d2*)

返回两个日期之间的月数。

如果两个日期代表月的同一天，或者代表他们各自的月的最后一天，则返回一个整数，否则，MONTHS_BETWEEN将返回一个分数值。

示例

```
SQL> SELECT MONTHS_BETWEEN(TO_DATE('29-Dec-1999'), TO_DATE('29-Dec-1988'))
  2  FROM dual;

MONTHS_BETWEEN(TO_DATE('29-DEC-1999'),TO_DATE('29-DEC-1988'))
-------------------------------------------------------------
                                                         132

SQL> SELECT MONTHS_BETWEEN(TO_DATE('12-May-2000'), TO_DATE('29-Dec-1988'))
  2  FROM dual;

MONTHS_BETWEEN(TO_DATE('12-MAY-2000'),TO_DATE('29-DEC-1988'))
-------------------------------------------------------------
                                                  136.451613
```

注意：MONTHS_BETWEEN返回的分数值是基于每月31天形成的。当大家跨月时，会得到一些古怪的值。在第2个例子中，大家知道从1988年12月29日到2000年4月29日得到136个月。分数部分0.451613乘以31，将得到14天。这似乎非常奇怪，因为在4月29日到5月12日只有13天，但因为计算分数结果的目的，MONTHS_BETWEEN函数使用了31天作为4月的天数。

NEW_TIME

NEW_TIME(*d, z1, z2*)

将日期/时间值从一个时区转换成另一时区。

参数

d 指定要转换的日期/时间值。

z1 指定源时区。该参数必须是表 5-3 显示的时区标识符之一。

z2 指定目的时区。该参数必须是表 5-3 显示的时区标识符之一。

表 5-3: 时区标识符

时区标识符	时区
AST	大西洋标准时间
ADT	大西洋白天时间
BST	白令海标准时间
BDT	白令海白天时间
CST	地中海标准时间
CDT	地中海白天时间
EST	东部标准时间
EDT	东部白天时间
GMT	格林威治标准时间
HST	阿拉斯加－夏威夷标准时间
HDT	阿拉斯加－夏威夷白天时间
MST	山区标准时间
MDT	山区白天时间
NST	纽芬兰标准时间
PST	太平洋标准时间
PDT	太平洋白天时间
YST	育空河标准时间
YDT	育空河白天时间

示例

下面的例子演示了太平洋标准时间上午8:00等于东部标准时间上午11:00。ALTER SESSION 命令改变了默认日期格式，使得一同显示时间和日期:

```
SQL> ALTER SESSION SET NLS_DATE_FORMAT = 'dd-Mon-yyyy hh:mi am';
```

```
Session altered.

SQL> SELECT NEW_TIME(TO_DATE('8-May-2000 8:00 am','dd-mon-yyyy hh:mi am'),
  2                  'PST','EST')
  3  FROM dual;

NEW_TIME(TO_DATE('8-
--------------------
08-May-2000 11:00 am
```

NEXT_DAY

NEXT_DAY(*d*, *string*)

计算下一次出现的特定一周中的某一天。

参数

d 指定日期值。日期中的任何时间组件都被保留，而且返回在结果中。

string
指定一周中某一天的名字。该参数可以是完整名，如Wednesday，或者缩写，如Wed。天名必须是当前NLS_DATE_LANGUAGE设置的有效日期。

示例

```
SQL> SELECT NEXT_DAY('20-May-2000', 'SATURDAY'),
  2         NEXT_DAY('20-May-2000', 'Sun')
  3  FROM dual;

NEXT_DAY('2 NEXT_DAY('2
----------- -----------
27-May-2000 21-May-2000
```

NEXT_DAY总是向前查找。如果大家传递的参数刚好就是大家要查找的日期，则NEXT_DAY返回这一天的随后出现的天。例如，2002年5月20日刚好是星期六，而该函数返回了2000年5月27日作为结果，因为它是“下一个”星期六。如果想让NEXT_DAY返回输入的日期作为结果，就应该首先给输入减1。例如：

```
SQL> SELECT NEXT_DAY(TO_DATE('20-May-2000')-1, 'SATURDAY')
  2  FROM dual;

NEXT_DAY(TO
```

```
-----------
20-May-2000
```

TO_DATE 函数被用于第 2 个例子中，因为 Oracle 不允许大家从字符串上减 1。从日期上减 1 导致 NEXT_DAY 返回 2000 年 5 月 20 日，这刚好就是星期六。

ROUND

ROUND(*d* [,*fmt*])

将日期/时间值取舍为最接近的日期/时间单位。

参数

d 指定的日期/时间值。

fmt

指定日期格式元素。看一看表 5-4，可以得到完整的列表。通过该日期元素指定的单位，将对输入值进行取舍。将日期/时间值取舍为最近一天是默认值。

注意：取舍与截取不相同。取舍日期可以导致新日期比开始的日期要大。如果不想取入，则使用 TRUNC 函数予以代替。

示例

```
SQL> ALTER SESSION SET NLS_DATE_FORMAT = 'DD-Mon-YYYY hh24:mi';

Session altered.

SQL> SELECT ROUND(TO_DATE('21-Jul-2000 15:20')) day,
  2         ROUND(TO_DATE('21-Jul-2000 15:20'),'HH') hour,
  3         ROUND(TO_DATE('21-Jul-2000 15:20'),'YYYY') year
  4  FROM dual;

DAY                 HOUR                YEAR
----------------- ----------------- -----------------
22-Jul-2000 00:00 21-Jul-2000 15:00 01-Jan-2001 00:00
```

表 5-4: 日期格式元素

格式元素	功能
- / , . ; :	标点符号可以包括在日期格式字符串的任何地方，而且将被包括在输出中
'text'	被引用的文本也可以包括在日期格式字符串中，而且将在输出中重复产生
AD 或 A.D. BC 或 B.C.	在日期显示中包括公元和公元前
AM 或 A.M. PM 或 P.M.	无论在任何时候应用时间都显示 AM 或 PM
CC	世纪数字。该数字用 20 代表 1900 年到 1999 年
SCC	与 CC 相同，但公元前的日期是负数
D	一周中一天的数字（1 到 7）
DAY	天的名称（星期六、星期天和星期一等等）
DD	月的天数
DDD	年的天数
DY	天的缩写名（Sat、Sun 和 Mon 等等）
HH	天中的小时，这里是 1~12。
HH12	天中的小时，这里是 1~12，与 HH 相同
HH24	24 小时时钟下天的小时数（0~23）
IW	一年中的周（1~53）
IYYY	4 位数字年份
IYY	年的最后 3 位数
IY	年的最后两位数
I	年的最后一位数字
J	儒略历天。第 1 天等于公元前 4712 年 1 月 1 日
MI	分钟
MM	月数
MON	3 个字母的月份缩写

表 5-4: 日期格式元素（续）

格式元素	功能
MONTH	月名，完全拼写出来
Q	年的季度。1 季度是 1~3 月，2 季度是 4~6 月等等
RM	罗马数字的月份数
RR	在使用 TO_CHAR 时，返回年的最后 2 位数字
RRRR	在使用 TO_CHAR 时，返回年的 4 位数字
SS	秒
SSSSS	从午夜开始的秒数
WW	一年中的周
W	月中的周。第一周从月的第一天开始算起，第 2 周从月的第 8 天开始算起等等
Y,YYY	4 位数的年份，在第一位数字后有逗号
YEAR	用单词的形式拼写出的年
SYEAR	用单词的形式拼写出的年，在表示公元前的年份时，前面有一个负号
YYYY	4 位数字年份
SYYYY	4 位数字年份，当表示公元前的年份时，前面有一个负号
YYY	年数的最后 3 位数字
YY	年数的最后 2 位数字
Y	年数的最后 1 位数字

SYSDATE

SYSDATE

返回当前日期和时间。该时间组件的结果包括小时、分钟和秒。SYSDATE 也称为伪列。要了解更多信息，参见第一章。

示例

```
SQL> SELECT SYSDATE FROM dual;
```

```
SYSDATE
-----------
21-May-2000

SQL> ALTER SESSION SET NLS_DATE_FORMAT = 'DD-Mon-YYYY hh24:mi';

Session altered.

SQL> SELECT SYSDATE FROM dual;

SYSDATE
--------------------
21-May-2000 15:16:40
```

TRUNC

TRUNC(*d* [,*fmt*])

返回被截取为指定单位的日期/时间值。TRUNC与ROUND类似，但它总是向下舍去，从来不会取入。

注意：当大家想确保只需要日期，而不是组合时间元件的日期时，使用TRUNC函数。

参数

d 指定日期/时间值。

fmt

指定日期格式元素。要得到这些元素的清单，参见表5-4。输入值将被截取（向下舍去）为该格式元素指定的单位。将一天的日期/时间截取到天是默认行为。

示例

```
SQL> ALTER SESSION SET NLS_DATE_FORMAT = 'DD-Mon-YYYY hh24:mi';

Session altered.

SQL> SELECT TRUNC(TO_DATE('21-Jul-2000 15:20')) day,
  2         TRUNC(TO_DATE('21-Jul-2000 15:20'),'HH') hour,
  3         TRUNC(TO_DATE('21-Jul-2000 15:20'),'YYYY') year
  4  FROM dual;
```

```
DAY                 HOUR                YEAR
-----
21-Jul-2000 00:00 21-Jul-2000 15:00 01-Jan-2000 00:00
```

转换函数

转换函数允许将一种数据类型转换为另一种数据类型。最常用的转换函数有:

TO_CHAR

转换日期或数字值为字符串。

TO_NUMBER

转换字符串为数字。

TO_DATE

转换字符串为日期值。

还有一组可用的其他转换函数，所有这些转换函数都包含在本节中。

CHARTOROWID

CHARTOROWID(*char*)

将CHAR型或VARCHAR2型值转换成ROWID值。这里的ROWID类型是Oracle7的类型。

示例

```
SQL> SELECT *
  2 FROM dual
  3 WHERE ROWID = CHARTOROWID('AAAADCAABAAAAVUAAA');

D
-
X
```

CONVERT

CONVERT (*string, dest_char_set*[, *source_char_set*])

将字符串从一个字符集转换成另一个字符集。

参数

string

指定要被转换的字符串。

dest_char_set

指定目的字符集名。

source_char_set

指定源字符集名。该参数默认为数据库字符集。

示例

```
SQL> SELECT CONVERT(CHR(194) || CHR(133) || CHR(64) || CHR(213)
  2                 || CHR(137) || CHR(131) || CHR(133),
  3                 'US7ASCII','WE8EBCDIC37C')
  4  FROM dual;

CONVERT
-------
Be Nice
```

HEXTORAW

HEXTORAW(*string*)

转换字符串中包含的十六进制数字为对应于那些数字组成的RAW值。

示例

```
SQL> SELECT DUMP(HEXTORAW('C28540D5898385'))
  2  FROM dual;

DUMP(HEXTORAW('C28540D5898385'))
--------------------------------------
Typ=23 Len=7: 194,133,64,213,137,131,133
```

RAWTOHEX

RAWTOHEX(*raw*)

转换RAW值为十六进制数字的字符串，其中每个2字符十六进制数字对应于一个字节的原始值。

示例

```
SQL> SELECT RAWTOHEX(HEXTORAW('C28540D5898385'))
  2  FROM dual;

RAWTOHEX(HEXTO
--------------
C28540D5898385
```

ROWIDTOCHAR

ROWIDTOCHAR(*rowid*)

转换ROWID为字符串。

示例

下面的例子使用了ROWIDTOCHAR函数显式转换ROWID值为字符串：

```
SQL> SELECT ROWID, ROWIDTOCHAR(ROWID) FROM dual;

ROWID              ROWIDTOCHAR(ROWID)
------------------ ------------------
AAAADCAABAAAAVUAAA AAAADCAABAAAAVUAAA
```

这一例子显示，只要从用SQL*PLUS的表中选择了ROWID，该值就被隐式转换为字符串。因为这一原因，大家通常看见SELECT语句提取ROWID，但并不可以显式转换它们。

TO_CHAR（将日期转换为字符串）

TO_CHAR(*d* [,*fmt* [,'*nlsparams*']])

转换日期/时间值为用字符表示的值。

参数

d 指定日期/时间值。

fmt

指定由表 5-4 中的元素组成的日期格式字符串。

nlsparams

以NLS_DATE_LANGUAGE=*langname*的格式指定字符串，这里的*langname*代表有效的NLS语言名。NLS语言名决定了天和月名使用的拼写。

示例

```
SQL> SELECT TO_CHAR(SYSDATE),TO_CHAR(SYSDATE,'dd-Mon-yyyy hh:mi:ss PM')
  2  FROM dual;

TO_CHAR(SYSDATE)  TO_CHAR(SYSDATE,'DD-MON
----------------- ----------------------
21-May-2000 17:40 21-May-2000 05:40:43 PM

SQL> SELECT TO_CHAR(SYSDATE,'dd-Mon-yyyy','NLS_DATE_LANGUAGE=FRENCH')
  2  FROM dual;

TO_CHAR(SYS
-----------
21-Mai-2000
```

TO_CHAR (转换数字为字符串)

TO_CHAR(*n* [,*fmt* [,'*nlsparams*']])

将数字值转换成用字符表示的值。

参数

n 指定要转换的数字值。

fmt

指定由表 5-5 中的元素组成的数字格式规范。

nlsparams

指定由下列一个或多个元素组成的字符串，允许大家为结果指定各种NLS字符。

NLS_NUMERIC_CHARACTERS=''*dg*''

允许大家指定为小数点和分组分割符指定字符。*d*字符变成了小数点，而

g字符变成了分组分割符。要使该参数生效，大家必须在自己的格式规范中使用D和G，以分别标记小数点和分组分割符的位置。

NLS_CURRENCY=''*text*''

允许指定任何货币符号（最多10个字符）代替格式规范中包含的L字符。

NLS_ISO_CURRENCY=''*territory*''

允许大家指定NLS区域，可用其中的货币符号代替格式规范中包含的C字符。

注意： *nlsparams*的设置是嵌入在字符串中的字符串。在周围看见的引号是两个单引号，而不是双引号。

示例

```
SQL> SELECT TO_CHAR(75917.63,'$99,999.99') a,
  2         TO_CHAR(75917.63,'L99G999D99',
  3                 'NLS_NUMERIC_CHARACTERS='',*'' NLS_CURRENCY=''$$$''') b,
  4         TO_CHAR(75917.63,'C99G999D99',
  5                 'NLS_ISO_CURRENCY=''JAPAN''') c
  6  FROM dual;

A           B                    C
----------- -------------------- -------------
 $75,917.63            $$$759*18     JPY759,18
```

表5-5：数字格式元素

格式元素	功能
9	用于控制显示的有意义的数字的编号
0	用于在结果中标记希望开始显示前面的0的位置。它代替了一个9。最左边的格式字符串是0最通用的位置，但也可以放置在任何地方
$	在数字前显示$符号
,	在输出中放置逗号
.	标记小数点的位置
B	强制将0值显示为空白
MI	在格式字符串的结尾使用，使后面的负号显示负数值

表 5-5: 数字格式元素（续）

格式元素	功能
S	可以用在格式字符串的开始或末尾，并显示出符号；“+”用于标记正数，“-”用于标记负数；当使用 S 时，就显示符号
PR	使负数值显示在角括号中。例如，-123.99 将被显示为<123.99>。正数值将在前面和后面放置空格代替角括号
D	标记小数点的位置
G	在输出中放置分组分割符（通常是逗号）
C	标记想显示 ISO 货币指示符的地方。对美元，将使用 USD
L	标记想要出现本地货币显示符的位置。对美元，将使用“$”字符。不能在相同格式中使用 L 和 C
V	用于显示缩放值。在 V 右边显示的数字表示显示数字之前小数点的右边将有多少位被移动
EEEE	使得 SQL*Plus 使用科学表示法来显示值。大家必须使用确切的 4 个 E，而且必须出现在格式字符串的最右边
RN	允许大家使用罗马数字显示数字。这是大小写会引起差别的惟一格式元素。大写字母“RN”产生大写的罗马数字，而小写的字母“rn”产生小写的罗马数字。用罗马数字显示数字，必须是整数，而且必须在 1~3 999 之间
DATE	使得 SQL*Plus 假定数字代表儒略历日期，而且显示为 mm/dd/yy 格式

TO_DATE

```
TO_DATE(string [,fmt [,'nlsparams']])
```

把字符串代表的日期/时间值转换为 DATE 类型值。

参数

string

指定要被转换的日期/时间的字符串值。

fmt

指定由表 5-4 中的元素组成的日期格式字符串。

nlsparams

以NLS_DATE_LANGUAGE=*langname*的形式指定字符串，这里*langname*代表了有效的NLS语言名。NLS语言名影响了天数和月份的拼写。

示例

```
DECLARE
  x DATE;
BEGIN
  x := TO_DATE('21-May-2000','dd-Mon-yyyy');

  x := TO_DATE('5/21/2000 7:15 PM','mm/dd/yyyy hh:mi pm');
END;
```

TO_LOB

TO_LOB(*long_column*)

转换LONG或LONG RAW值为CLOB、BLOB或NCLOB。当使用该语句导出LOB列时，该函数只可以用作INSERT...SELECT FROM语句的子查询中。LONG值被转换为CLOB或NCLOB值，这由目的列的数据类型决定。LONG RAW值被转换为BLOB值。

示例

```
SQL> CREATE TABLE lob_table(x CLOB);

Table created.

SQL> CREATE TABLE long_table(x LONG);

Table created.

SQL> INSERT INTO long_table (x) VALUES ('test');

1 row created.

SQL> INSERT INTO lob_table (x)
  2      SELECT TO_LOB(x)
  3      FROM long_table;

1 row created.
```

TO_MULTI_BYTE

TO_MULTI_BYTE(*char*)

转换单字节字符为多字节等同物。如果单字节字符没有多字节等同物，就不作改变。该函数是为使用基于多字节字符集的字符串而设计的。

示例

```
SQL> SELECT TO_MULTI_BYTE('Be Nice') FROM dual;

TO_MULT
-------
Be Nice
```

TO_NUMBER

TO_NUMBER(*string* [,*fmt*[, '*nlsparams*']])

转换用字符表示的数字值为 NUMBER 值。

参数

string

指定想要转换的包含基于字符表示的字符串。

fmt

指定由表 5-5 中显示的元素组成的数字格式规范。

nlsparams

指定由下列元素中的一个或多个元素字符串，允许大家为结果指定各种NLS字符：

NLS_NUMERIC_CHARACTERS=''*dg*''

允许为小数点和分组分割符指定字符。*d* 字符变成了小数点，而 *g* 字符变成了分组分割符。为使其生效，大家必须在自己的格式规范中使用D和G，分别标记小数点和分组分割符的位置。

NLS_CURRENCY=''*text*''

允许指定任何货币符号（最多10个字符）代替格式规范中包含的L字符。

NLS_ISO_CURRENCY=''*territory*''

允许大家指定NLS区域，其中的货币符号可代替格式规范中包含的C字符。

示例

```
SQL> VARIABLE x NUMBER
SQL> SELECT TO_NUMBER('123.45') INTO :x FROM dual;

TO_NUMBER('123.45')
-------------------
             123.45

SQL> SELECT TO_NUMBER('$123,456.78','$999,999.99') INTO :x FROM dual;

TO_NUMBER('$123,456.78','$999,999.99')
--------------------------------------
                             123456.78
```

TO_SINGLE_BYTE

TO_SINGLE_BYTE(*char*)

转换多字节字符为单字节的等同物。如果多字节没有单字节等同物，则不进行转换。本函数是为使用基于多字节字符集的字符串而设计的。

示例

```
SQL> SELECT TO_SINGLE_BYTE('Be Nice') FROM dual;

TO_SING
-------
Be Nice
```

TRANSLATE USING

TRANSLATE(*text* USING {CHAR_CS | NCHAR_CS})

转换文本为数据库字符集或国际字符集。

参数

text

指定大家想要转换的字符串。

CHAR_CS

将字符串从国际标准字符集转换为数据库字符集。返回的结果为VARCHAR2值。

NCHAR_CS

使字符串从数据库字符集转换为国际字符集。返回的结果为NVARCHAR2值。

示例

```
SQL> SELECT TRANSLATE('Be Nice' USING NCHAR_CS)
  2  FROM dual;

TRANSLA
-------
Be Nice
```

其他函数

许多函数不能归入任何其他分类，所以在这里列出。DECODE、NVL和USER函数是这一种类中最重要的3个函数。其他的还包括SYS_CONTEXT、DUMP和VSIZE。

BFILENAME

BFILENAME(*directory, filename*)

返回指向指定文件的BFILE定位符。

参数

directory

指定以前使用CREATE DIRECTORY语句创建的包含该文件的目录。

filename

指定想要该定位符指向的文件名。注意，大家可以创建定位符指向并不存在的位置。

示例

```
DECLARE
```

```
   admin_photo BFILE;
BEGIN
   admin_photo := BFILENAME('/home/oracle','administrator_photo.jpg');
   ...
END;
```

DECODE

DECODE(*expr*, *search*,*result* [,*search*,*result*...] [,*default*])

提供嵌入IF语句的能力。DECODE是Oracle的嵌入式函数中最有用的可以列出参数的函数之一。给DECODE传入一个输入函数并提供一组结果/值对的列表，DECODE然后查找这一组值，看哪一个对与输入相匹配，当找到了这一对以后，DECODE就从该对中返回结果作为函数的结果值。如果没有找到，DECODE就返回默认结果。

注意：DECODE是可以正确操作NULL值的少数函数之一。

DECODE不可以从PL/SQL表达式中调用，它只可以从SQL语句中调用。

参数

expr

指定输入值。DECODE将该值与随后的检索值比较，看能否找到匹配值/结果对。

search

指定值/结果对的值部分。

result

指定值/结果对的结果部分。

default

如果没有一个检索值与输入表达式匹配，则指定DECODE返回可选的默认结果。

注意：DECODE限制最大有255个参数。

数据类型由第一个检索、结果对控制。输入表达式和所有检索值被在转换为第一个检索值的数据类型。返回值被转换为第一个结果值的数据类型。

示例

DECODE可以用于许多创造性目的。3种常用用途是：将编码值扩展为更易读的形式，将行转换为列，以及基于其他值返回一列。下面的例子显示了DECODE用于解释V$DATA-FILE视图中的编码字段。永远也不应该返回默认的返回值（Invalid plugged_in "value"）。

```
SQL> COLUMN name FORMAT A50
SQL> SELECT name,
  2         DECODE(plugged_in,
  3                0,'Not Plugged In',
  4                1,'Plugged In',
  5                'Invalid plugged_in value') plugged_in
  6  FROM v$datafile;

NAME                                               PLUGGED_IN
-------------------------------------------------- ------------------
/s01/app/oracle/oradata/donna/system01.dbf         Not Plugged In
/s01/app/oracle/oradata/donna/oemrep01.dbf         Not Plugged In
/s01/app/oracle/oradata/donna/rbs01.dbf            Not Plugged In
/s01/app/oracle/oradata/donna/temp01.dbf           Not Plugged In
/s01/app/oracle/oradata/donna/users01.dbf          Not Plugged In
/s01/app/oracle/oradata/donna/indx01.dbf           Not Plugged In
/s01/app/oracle/oradata/donna/drsys01.dbf          Not Plugged In
/s01/app/oracle/oradata/donna/oemrep02.dbf         Not Plugged In
/s01/app/oracle/oradata/donna/gnis_data_01.dbf     Plugged In
/s01/app/oracle/oradata/donna/gnis_index_01.dbf    Plugged In
```

DECODE的另一个创造性使用是，将通常应该作为多行返回的查询结果转换为许多列组成的一行结果。考虑下面的查询例子，计算SYS和SYSTEM拥有的对象数量：

```
SQL> SELECT owner, COUNT(*)
  2  FROM dba_objects
  3  WHERE OWNER IN ('SYS','SYSTEM')
  4  GROUP BY owner;

OWNER                          COUNT(*)
------------------------------ ---------
SYS                               6174
SYSTEM                             161
```

如果想为每一个用户只返回一行，该怎么办？可以使用DECODE。技巧是书写两个DECODE表达式，每一个表达式检查不同的用户名：

```
SQL> SELECT SUM(DECODE(owner,'SYS',1,0)), SUM(DECODE(owner,'SYSTEM',1,0))
  2  FROM dba_objects;

SUM(DECODE(OWNER,'SYS',1,0)) SUM(DECODE(OWNER,'SYSTEM',1,0))
---------------------------- ------------------------------
                        6174                            161
```

这里，第一个DECODE为SYS拥有的所有对象返回值1，而为所有其他的返回0。第二个DECODE表达式完成相同的事情，但它是针对SYSTEM。将这两列加在一起，允许大家得到每一个用户的总数，只为查询返回一行结果。该技术只在大家已经固定了要查找结果的数量时能起作用。这里没有方法扩展该技术来水平列出可变的用户数量。

前面提到的DECODE的第三种用法是用于返回一列值基础上的许多不同列（或表达式）的。下面的对DBA_TAB_COLUMNS的查询使用DECODE返回基于DATA_TYPE列的合适数据类型定义。

```
SELECT owner, table_name, column_name,
       DECODE(data_type,
              'VARCHAR2','VARCHAR2 (' || TO_CHAR(data_length) || ')',
              'NUMBER', DECODE(data_precision,
                               NULL, 'NUMBER',
                               'NUMBER (' ||
                               TO_CHAR(data_precision) || ',' ||
                               TO_CHAR(data_scale) || ')'))
FROM dba_tab_columns
WHERE data_type IN ('VARCHAR2','NUMBER');
```

为了简单起见，该例子只处理了VARCHAR2和NUMBER数据类型，但大家已经可以看到怎样扩展逻辑来合理显示其他数据类型。注意，使用嵌套DECODE函数来处理NUMBER类型。浮点数字定义为没有精度和尾数。只有在最初指定一个时，嵌套DECODE才会使NUMBER定义包括精度和尾数。

DUMP

DUMP(*expr* [,*return_format* [,*start_position* [,*length*]]])

返回一个VARCHAR2字符串显示的数据类型，而且内部代表的数据存储在表达式返回的列或数据中。

参数

expr

指定要转储的数据。该参数可以是列名或有效的 SQL 表达式。

return_format

指定格式控制转储数据的方式，这些格式是：

8 使用八进制符号。

10 使用十进制符号。这是默认值。

16 使用十六进制符号。

17 显示结果采用字符显示。

注意：如果用 1000 来指定格式，也会返回字符集名。只有在大家转储字符列或字符表达式结果时使用。

---

start_position

指定数据转储的起始字节。默认情况是从数据的第一个字节开始。

length

指定转储的字节数。默认情况是转储所有的数据。

示例

```
SQL> SELECT DUMP(SYSDATE) FROM dual;

DUMP(SYSDATE)
-------------------------------------
Typ=13 Len=8: 208,7,5,21,16,59,57,0

SQL> SELECT DUMP('Hi There',16) FROM dual;

DUMP('HITHERE',16)
-------------------------------------
Typ=96 Len=8: 48,69,20,54,68,65,72,65

SQL> SELECT DUMP('Hi There',1016) FROM dual;

DUMP('HITHERE',1016)
-------------------------------------------------------------
Typ=96 Len=8 CharacterSet=US7ASCII: 48,69,20,54,68,65,72,65
```

EMPTY_BLOB

EMPTY_BLOB()

返回空BLOB的位置，可以用于初始化BLOB列。

示例

```
SQL> CREATE TABLE x(b blob);

Table created.

SQL> INSERT INTO x (b) VALUES (EMPTY_BLOB());

1 row created.
```

警告：不像其他函数，EMPTY_BLOB必须带有括号，即使没有传递参数也是如此。

EMPTY_CLOB

EMPTY_CLOB()

返回空CLOB的位置，可以用于初始化CLOB列。

示例

```
SQL> CREATE TABLE x(c clob);

Table created.

SQL> INSERT INTO x (c) VALUES (EMPTY_CLOB());

1 row created.
```

注意：不像其他函数，EMPTY_CLOB必须带有括号，即使没有传递参数也是如此。

NLS_CHARSET_DECL_LEN

NLS_CHARSET_DECL_LEN(*bytecnt*, *csid*)

在提供的字节计数的基础上，返回NCHAR列的定义宽度（根据字符的数量）。

参数

bytecnt

以字节的方式，指定NCHAR列的大小。

csid

指定数字定义为该列使用的NLS字符集。可以使用NLS_CHARSET_ID函数得到与字符集名对应的字符集数字。

示例

```
SQL> SELECT NLS_CHARSET_DECL_LEN(100,NLS_CHARSET_ID('US7ASCII'))
  2  FROM dual;

NLS_CHARSET_DECL_LEN(100,NLS_CHARSET_ID('US7ASCII'))
----------------------------------------------------
                                                 100
```

NLS_CHARSET_ID

NLS_CHARSET_ID(*text*)

返回对应于给定的NLS字符集名的ID号。

示例

```
SQL> SELECT NLS_CHARSET_ID('US7ASCII'), NLS_CHARSET_ID('WE8EBCDIC37C')
  2  FROM dual;

NLS_CHARSET_ID('US7ASCII') NLS_CHARSET_ID('WE8EBCDIC37C')
---------------------------- -----------------------------
                           1                            90
```

NLS_CHARSET_NAME

NLS_CHARSET_NAME(*n*)

返回给定NLS字符集ID号的对应名。

示例

```
SQL> SELECT NLS_CHARSET_NAME(1), NLS_CHARSET_NAME(90)
  2  FROM dual;

NLS_CHAR NLS_CHARSET_
-------- ------------
US7ASCII WE8EBCDIC37C
```

NVL

NVL(*expr1*, *expr2*)

如果给定的输入值为NULL，则返回一个备用值。若*expr1*为NULL，则NVL返回*expr2*；否则，返回*expr1*。

参数

expr1

指定测试是否为NULL的表达式。如果该值不是NULL，则它是函数返回的值。

expr2

指定当*expr1*为NULL时的返回值。

示例

```
SQL> SELECT username, NVL(TO_CHAR(lock_date),'Not Locked')
  2  FROM dba_users;

USERNAME                       NVL(TO_CHAR(LOCK_
------------------------------ -----------------
SYS                            Not Locked
SYSTEM                         Not Locked
...
GNIS                           21-May-2000 17:18
```

SYS_CONTEXT

SYS_CONTEXT(*namespace*, *attribute_name* [,*length*])

返回应用程序上下文命名空间中的属性值。

参数

namespace

指定以前使用 CREATE CONTEXT 语句创建的命名空间的名字。大家也可以指定默认命名空间 USERENV。

attribute

指定命名空间中的属性名。然后由该函数返回该属性值。许多预定义属性都可以与默认的 USERENV 命名空间一同使用。这些属性在表 5-6 中列出。

length

指定属性的返回值的允许长度。该可选参数只从Oracle8.1.6开始使用，而且只用于 AUTHENTICATION_DATA 属性。

表 5-6: USERENV 命名空间中的预定义属性

属性名	描述	最大长度	版本介绍
AUTHENTICATION_DATA	用于鉴别用户的数据。如果鉴别是通过 X.503 认证进行，那么认证的内容将以十六进制的格式返回。这是惟一可以使用 SYS_CONTEXT 函数的 LENGTH 参数的选项。可以指定最大长度为 4000 字节	2564000	8.1.6
AUTHENTICATION_TYPE	显示用户怎样被鉴别。将返回下列值之一: DATABASE: 使用用户名和口令 OS: 使用鉴别的操作系统 NETWORK: 通过网络协议或先进的网络工作选项（ANO）鉴别 PROXY: 由 OCI（Oracle 调用接口）代理鉴别	30	8.1.6
BG_JOB_ID	如果当前会话是由后台的 Oracle 进程所创建，该属性就是会话的工作 ID; 否则，就是空值	30	8.1.6

表 5-6: USERENV 命名空间中的预定义属性（续）

属性名	描述	最大长度	版本介绍
CLIENT_INFO	使用 DBMS_APPLICATION_INFO 包返回存储的不超过 64 字节用户会话信息	64	8.1.6
CURRENT_SCHEMA	当前架构名。通常与当前用户相匹配。但是，如果以 INTERNAL 用户或者 SYSDBA 或 SYSOPER 登录，架构名就是 SYS	30	8.1.5
CURRENT_SCHEMAID	与当前架构关联的 ID 号	30	8.1.5
CURRENT_USER	当前用户名。如果已经调用了存储过程，这就不会与 SESSION_USER 返回的登录用户名相同	30	8.1.5
CURRENT_USERID	与当前用户相关的 ID 号	30	8.1.5
DB_DOMAIN	由 DB_DOMAIN 初始化参数指定的数据库域	256	8.1.6
DB_NAME	由 DB_NAME 初始化参数指定的数据库名	30	8.1.6
ENTRYID	审核条目标识符。在分布式 SQL 语句中，该属性无效。另外，必须将 AUDIT_TRAIL 初始化参数设置为 TRUE	30	8.1.6
EXTERNAL_NAME	数据库用户的扩展名。对由 SSL 鉴别的用户，这是 V.503 认证的区别名（distinguished name，DN）	256	8.1.6
FG_JOB_ID	如果会话使由客户的前台进程创建的话，就是会话的工作 ID；否则，就为空	30	8.1.6
HOST	客户端连接的机器名	54	8.1.6

表 5-6: USERENV 命名空间中的预定义属性（续）

属性名	描述	最大长度	版本介绍
INSTANCE	该数字标识了当前连接的实例	30	8.1.6
IP_ADDRESS	用户的 IP 地址。该地址只出现在 TCP/IP 连接中	30	8.1.5
ISDBA	根据是否允许 ISDBA 角色，返回 TRUE 或 FALSE	30	8.1.6
LANG	当前语言名的 ISO 缩写	62	8.1.6
LANGUAGE	当前语言设置、区域设置和数据库字符集名	52	8.1.6
NETWORK_PROTOCOL	用于连接的网络协议名。该名字来源于 Net8 连接字符串的 PROTOCOL 属性	256	8.1.6
NLS_CALENDAR	当前 NLS 日历名	62	8.1.5
NLS_CURRENCY	当前 NLS 货币指示符	62	8.1.5
NLS_DATE_FORMAT	当前 NLS 日期格式	62	8.1.5
NLS_DATE_LANGUAGE	当前 NLS 日期语言	62	8.1.5
NLS_SORT	当前排序基础	62	8.1.5
NLS_TERRITORY	当前 NLS 区域名	62	8.1.5
OS_USER	初始化数据库连接的客户操作系统用户名进程	30	8.1.6
PROXY_USER	打开当前会话的用户，代表当前会话的用户名	30	8.1.6
PROXY_USERID	打开当前会话的用户，代表当前会话的用户 ID	30	8.1.6
SESSION_USER	当前用户登录的名字。注意即使存储过程被另一个用户调用时，也不会改变该名字	30	8.1.5
SESSION_USERID	与会话用户关联的 ID 号	30	8.1.5

表 5-6: USERENV 命名空间中的预定义属性（续）

属性名	描述	最大长度	版本介绍
SESSIONID	大家的审核会话标识符。在分布式 SQL 语句中，该属性无效	30	8.1.6
TERMINAL	当前会话的客户的操作系统标识符。在分布式环境下，该标识符只用于远端 SELECT 语句，并返回本地会话的标识符	10	8.1.6

示例

```
SQL> SELECT SYS_CONTEXT('USERENV', 'SESSION_USER')
  2  FROM dual;

SYS_CONTEXT('USERENV','SESSION_USER')
-------------------------------------------------------------------
SYSTEM

SQL> SELECT SYS_CONTEXT('USERENV', 'NLS_SORT')
  2  FROM dual;

SYS_CONTEXT('USERENV','NLS_SORT')
-----------------------------------------------------
BINARY
```

SYS_GUID

```
SYS_GUID()
```

返回可以用作全局惟一标识符的16字节的RAW值。在大部分平台下，该值是主机ID、进程（或线程）ID和序列号的组合体。

当被SQL*PLUS显示时，RAW值被转换成十六进制，所以在例子中，结果显示了32，而不是16个字符，因为已经被显示的每一个字符使用了2个十六进制数字。

示例

```
SQL> SELECT SYS_GUID() FROM dual;
```

```
SYS_GUID()
--------------------------------
6827BA0C1CF2D067E0300B0A100C0246
```

注意：不像其他函数，SYSGUID 后必须跟随括号，即使不传递参数也应如此。

UID

UID

返回惟一定义当前数据库用户的整数值。该值来源于 V$SESSION 视图的 USER# 列。

示例

```
SQL> SELECT UID FROM dual;

       UID
---------
        10
```

USER

USER

返回当前用户。正常情况下，USER 返回用于登录到数据库的用户名。但是，当在存储过程或函数中调用时，USER 返回存储过程或函数的所有者的名字。USER 也称为伪列。从第一章可以了解更多信息。

注意：当在触发器中调用时，USER 返回登录用户名。

示例

```
SQL> SELECT USER FROM dual;

USER
------------------------------
SYSTEM
```

USERENV

USERENV(*option*)

返回有关当前用户的信息。

参数

option

指定被返回的确切信息，而且必须是表5-7中列出的值之一。被该函数返回的值总是VARCHAR2字符串。

注意：从Oracle8.1.6开始，SYS_CONTEXT函数也可以用于提取这些用户的环境信息。

表5-7：USERENV的可选值

选项	描述
ENTRYID	返回审核条目标识符。要使选项有效，必须使该AUDIT_TRAIL初始化参数为TRUE，而且不能在分布式环境下使用该选项
INSTANCE	返回实例标识符
ISDBA	根据是否允许ISDBA角色，返回TRUE或FALSE
LANG	返回当前ISO语言的缩写
LANGUAGE	返回用户的当前语言和区域设置
SESSIONID	返回审核会话标识符。要使选项有效，必须使该AUDIT_TRAIL初始化参数为TRUE，而且大家不可以在分布式环境下使用该选项
TERMINAL	返回当前会话的操作系统终端标识符

示例

```
SQL> SELECT USERENV('INSTANCE'),USERENV('ISDBA') FROM dual;

USERENV('INSTANCE') USEREN
------------------- ------
                  1 FALSE
```

VSIZE

VSIZE(*expr*)

以字节的形式返回内部代表值的大小。

示例

```
SQL> SELECT VSIZE(SYSDATE) FROM dual;

VSIZE(SYSDATE)
--------------
             8

SQL> SELECT VSIZE(username) FROM dba_users;

VSIZE(USERNAME)
---------------
              3
              6
              5
```

第六章

SQL*Plus

SQL*Plus 是与数据库的 Oracle 基本命令行接口。它允许大家执行 SQL 语句和 PL/SQL 块及安排输出的格式，也给你提供一定数量的操作控制。从最早版本的 Oracle 开始，就可以使用 SQL*Plus（那时叫做用户友好接口，UFI）。SQL*Plus 可以执行下列类型的语句：

- SQL 语句（DDL 和 DML），对应于 ANSI SQL 标准和 Oracle 扩展
- PL/SQL 语句，从 Oracle 扩展的私有 SQL 语言
- SQL*Plus 语句，从 Oracle 扩展的私有格式和操作语句

用简单的术语来说，这意味着SQL*Plus直接与源语句交往，不需要分开编译。SQL 语句被发送到 Oracle 服务器上（本地的或通过 NET8 或 SQL*Net），在这里分析这些语句并执行。PL/SQL 块也被发送到 Oracle 服务器上，在这里被编译并被执行。SQL*Plus 语句直接通过运行 SQL*Plus 来执行。

命令行语法

SQL*Plus是一个命令行应用程序，而且通常从命令提示符下调用。从命令提示符下调用 SQL*Plus 的语法如下：

```
sqlplus [option] [username[/password][@hostname]] [@script] [parm1] [parm2] . . .
```

关键字

option

指定控制SQL*Plus操作的选项，具有下列语法：

```
[-]
{-?]
[-M[ARKUP] HTML [ON | OFF]
  [HEAD text]
  [BODY text]
  [ENTMAP {ON | OFF}]
  [SPOOL {ON | OFF}]
  [PRE[FORMAT] {ON | OFF}]]
[-R[ESTRICT] {1 | 2 | 3}]
[-S[ILENT]] ]
```

- 显示SQL*Plus命令行的使用和语法，然后退出操作系统。

-? 显示SQL*Plus的版本，然后退出操作系统。

MARKUP HTML

指定该SQL*Plus产生的HTML输出。

ON

指定使用指定选项产生HTML输出。

OFF

指定不产生HTML输出。该选项为默认值。

HEAD

指定*text*被用作<HEAD>标签的内容。标题文本必须闭包在引号中。默认情况下，*text*是<TITLE>SQL*Plus Report</TITLE>。

BODY

指定*text*被用作<BODY>标签的内容。文本体必须闭包在引号中。

ENTMAP ON

指定SQL*Plus显示特定的字符“<”、“>”和“&”，作为HTML各自的实体`<`、`>`和`&`。

ENTMAP OFF

指定该SQL*Plus不改变显示的特殊字符“<”、“>”和“&”。

SPOOL ON

指定 SQL*Plus 书写 HTML 标签到每一个由 SQL*Plus SPOOL 命令创建的文件的开始和结尾。

SPOOL OFF

指定 SQL*Plus 不书写 HTML 标签到每一个由 SQL*Plus SPOOL 命令创建的文件的开始和结尾。

PREFORMAT ON

指定 SQL*Plus 在 HTML <PRE> 标签内部书写输出。

REFORMAT OFF

指定 SQL*Plus 不在 HTML <PRE> 标签内部书写输出。

RESTRICT

指定在该会话中有效的限制层，用 1、2 和 3 显示。限制层禁止了特定的 SQL*Plus 命令，如表 6-1 所示。

SILENT

指定被压缩的所有的 SQL*Plus 信息和提示信息，包括命令提示、命令的回应和启动 SQL*Plus 时的正常标语显示。

username

大家正在连接的账户的 Oracle 用户名。

password

用户名的口令。如果省略，SQL*Plus 将提示需要口令。

hostname

给正在连接的数据库分配的主机名。通常，主机名使用 *tnsnames.ora* 或 Oracle 名来解决。如果省略主机名，则 SQL*Plus 将试图从环境变量或注册命名为 LOCAL 的条目中得到。如果没有 LOCAL 环境变量或注册条目，则 SQL*Plus 将试图通过当前 ORACLE_SID 设置连接到本地数据库上。

script

成功执行连接到数据库上的 SQL*Plus 脚本名。

parm1, parm2 ...

将被传递到 SQL*Plus 的选择参数，而且可以使用 SQL*Plus 脚本中的代替变量引用。字符串参数包含的空格或其他特殊字符必须封装在引号中。

表 6-1：被限制层限制的命令

命令	第一层	第二层	第三层
EDIT	Yes	Yes	Yes
GET	No	No	Yes
HOST 或!	Yes	Yes	Yes
SAVE	No	Yes	Yes
SPOOL	No	Yes	Yes
START 或 @ 或 @@	No	No	Yes
STORE	No	Yes	Yes

示例

如果连接到本地服务器，而且适当地设置 ORACLE_SID 环境变量，就可以使用下列简单的命令调用 SQL*Plus:

```
sqlplus
```

SQL*Plus 将被启动，而且将被提示输入用户名和口令。大家然后将连接到默认 Oracle 示例中。

如果大家喜欢的话，可以在命令行传递用户名和口令。下列的命令将使大家连接到默认实例并使用 scott 用户登录:

```
sqlplus scott/tiger
```

下一个例子的命令启动 SQL*Plus，运行名为 *report1.sql* 的 SQL 脚本。将提示大家输入用户名和口令:

```
sqlplus @report1
```

在最后一个例子中的命令行启动了 SQL*Plus，而且通过 Net8 服务命名测试作为用户 scott 连接到 Oracle 实例上，然后执行 SQL 脚本 *report2.sql*。命令行提供的参数值 1000 和 5000 被传递到脚本中:

```
sqlplus scott/tiger@test @report2 1000 5000
```

注意：实际SQL*Plus的可执行名可以因平台而变动，也可以因版本而变动。例如，SQL*Plus版本8.0命令行下的可执行名在Windows95/98下是*PLUS80.EXE*；而在该平台下的GUI版本的可执行名则是*PLUS80W.EXE*。

SQL*Plus 编辑命令

在启动SQL*Plus会话时，会提供一个SQL*Plus命令提示，显示SQL*Plus已经准备接受输入。默认的提示是SQL>。输入的格式很自由，可以由以下内容组成：SQL语句(DML或DDL)、PL/SQL代码和SQL*Plus命令。

SQL*Plus命令立即被执行，而且不被保存。SQL语句和PL/SQL块被存储在内存缓存中，而且称为SQL*Plus缓存。缓存的内容以行的形式组织。当大家录入SQL命令或PL/SQL块时，每次在大家按下回车键时，就创建新的缓存行。本节中描述的命令允许编辑和关联SQL*Plus缓存中的内容操作，而且可以直接从SQL>提示下录入。它们也可以被包含在SQL*Plus脚本中。

注意：大部分编辑命令在当前行上操作。当前行总是最近录入或显示的行，而且SQL*Plus总是将其标记为星号。如果想要编辑的行不是当前行，就可以使用LIST命令让其变成当前行。然后大家就可以执行其他编辑命令，输入想要进行的变化。

APPEND

A[PPEND] *text*

添加文本到当前行的末尾。

CHANGE

C[HANGE] */old/[new/]*

改变老的行为新的当前行。要删除文本，可以提供老的值而不提供对应的新值。

示例

改变文本“abc”为“xyz”：

```
CHANGE /abc/xyz/
```

下一个例子中，因为没有提供新的文本，所以指定的文本就从被编辑的行中简单地删除了：

```
CHANGE /abc/
```

CLEAR

CL[EAR] BUFF[ER]

从缓存中删除所有行。

DEL

DEL {*line* | *}

删除指定行，或者从缓存中删除当前行。

DEL {*begin end* | *line* * | * *line*}

从缓存中删除指定范围的行。

DEL {*line* LAST | * LAST}

从缓存中删除从指定行开始的所有行或当前行。

DEL LAST

从缓存中删除最后一行。

关键字

line

指定行号。

*　指定当前行。

begin

指定多行中被删除的开始行。所有从开始到结尾的行都被删除。

end

指定多行中被删除的最后行。所有从开始到结尾的行都被删除。

LAST

指定缓存中的最后一行。

GET

GET *filename* [LIS[T] | NOL[IST]]

放置文件内容到缓存中。

关键字

filename

指定操作系统文件的名称，其内容将被放置在缓存中，代替当前缓存的内容。

LIST

指定加载缓存时，列出文件内容。该选项是默认值。

NOLIST

指定不列出文件的内容。

注意：正常情况下，GET命令仅用于加载包含单个SQL语句的文件，因为SQL*Plus不能从缓存中执行多条语句。

INPUT

I[NPUT] [*text*]

添加一行或多行到缓存中。在当前行之后插入新行。提供的文本将被输入到缓存中。如果省略文本，SQL*Plus将对每一新行提示输入文本，直到输入了一个空行进行终止为止。

LIST

L[IST]

显示缓存中的所有行。

L[IST] *line*

显示缓存中的单行。

L[IST] *

显示缓存中的当前行。

L[IST] LAST

显示缓存中的最后一行。

L[IST] {*begin end* | *begin* LAST | *line* LAST | * LAST}

显示缓存中的行的范围。

关键字

line

指定行号。

*　指定当前行。

LAST

指定缓存中的最后一行。

begin

指定列表中的第一行。从开始到结尾的所有行都将被列出。

end

指定列表中的最后行。从开始到结尾的所有行都将被列出。

SAVE

SAV[E] *filename* {CREATE | REPLACE | APPEND}

保存缓存内容到特定文件中。保留缓存内容。

关键字

CREATE

创建新文件。如果指定该选项，则该文件一定不能已经存在。该选项是默认值。

REPLACE

代替现存文件的内容。如果文件不存在，则 SQL*Plus 将创建该文件。

APPEND

将缓存中的内容添加到指定文件的结尾。

注意：只有单个SQL语句或PL/SQL块会包含在缓存中将被保存到磁盘上。不会保存SQL*Plus命令，即使它们以前被执行了也不行，因为它们没有保存在缓存中。要创建包含SQL*Plus 命令的脚本，大家必须使用文本编辑器。

一旦缓存中的内容被保存在文件中，则可以使用GET命令提取该文件。START命令可以用于执行该文件中的命令。

规定 SQL*Plus 输出的格式

虽然 SQL*Plus 提供了一些基本形式的默认输出格式，大家也可以以各种形式产生“最后”报表来定制输出。下面的命令实现了 SQL*Plus 规定格式的能力。大家可以从 SQL*Plus 提示符下录入这些命令，或者将它们包括在 SQL*Plus 脚本中。

BREAK（有参数）

```
BRE[AK]  [ON column | expression | ROW | REPORT [action [action]]]. . .
```

指定在哪里和怎样改变报表的格式。BREAK 通常和COMPUTE一起使用来产生合计和小计。

关键字

column

当指定的列的值发生改变时，导致发生特定的行为。

expression

当指定的表达式的值发生改变时(必须与SELECT语句中的表达式正确匹配)，导致发生特定的行为。

ROW

无论什么时候 SQL*Plus 返回一行都导致行为发生。

REPORT

使对应的计算命令在报表的结尾执行。

action

可以是下列行为之一:

SKI[P]

在打印发生断点的一行之前跳过 n 行。

SKI[P] PAGE

跳到当前的新页中的需要的行号。如果指定 ISET NEWPAGE 0，则输出反馈形式的字符。

NODUP[LICATES]

当该值与前面一行的值相同时，使输出打印空白而不是断点列的值。

DUP[LICATES]

无论该列的值是否发生改变都为每一行输出断点列的值。

BREAK (没有参数)

```
BREAK
```

列出所有当前 BREAK 的定义。

BTITLE (控制)

```
BTI[TLE] ON | OFF
```

在不影响定义的情况下打开或关闭打印底部标题。

BTITLE (定义)

BTI[TLE] [*printspec* [*text* | *variable*] . . .]

创建并指定每一页底部出现的标题的格式。

关键字

printspec

用于在标题中放置文本或变量，并规定文本或变量格式的下列从句之一：

COL

启动列 *n* 中的文本或变量。

S[KIP]

在打印文本或变量之前跳过 *n* 行。默认数字是跳过 1 行。

TAB

在打印文本或变量之前向后跳过 *n* 个打印位置（如果 *n* 为负数，表示回跳）。

LE[FT]

按左对齐打印文本或变量。

CE[NTER]

居中打印文本或变量。LINESIZE 设置值用于计算页的宽度。

RI[GHT]

按右对齐打印文本或变量。

BOLD

以黑体打印文本或变量。

FORMAT

根据 *formatstring* 中指定的格式模型规定文本或变量的格式。要得到可用的数字格式，参见表 5-5。

text

在页上打印的字符串。多个单词必须放在单引号中。

variable

用户变量或系统维护变量。

BTITLE (显示)

BTI[TLE]

显示当前 BTITLE 的定义。

CLEAR

CL[EAR] *option*

复位或消除 SQL*Plus 选项的当前值。

关键字

option

可以是下列值之一:

BRE[AKS]

通过 BREAK 命令删除定义。

BUFF[ER]

清除缓存中的所有文本。

COL[UMNS]

使用 COLUMN 命令复位列定义集为默认值。

COMP[UTES]

删除计算命令设置的所有定义。

SCR[EEN]

清屏。

SQL

清除缓存中的所有文本。

TIMI[NG]

删除由 TIMING 命令创建的所有定时区域。

COLUMN

COL[UMN] {*column* | *expression*} [*option* . . .]

使用一个或多个选项指定列或表达式的显示特性。

```
COL[UMN] {column | expression}
```

显示命名列或表达式的当前显示属性。

关键字

column

SQL 语句中使用的列名。

expression

SQL 语句中使用的表达式。

option

下列的一个或多个:

ALI[AS]

分配列或表达式的别名，然后可以用在随后的 BREAK、COMPUTE 或其他 COLUMN 命令中。

CLE[AR]

复位该列的属性或将表达式设为默认值。

FOLD_A[FTER]

在 SQL*Plus 显示的每一行的标题和列之前插入回车。

FOLD_B[EFORE]

在 SQL*Plus 显示的每一行输出的标题和列之前插入回车。

FOR[MAT]

指定包含具体格式的字符串，用于控制怎样显示列的内容。可以从表 5-4 看到可用的日期格式，并从表 5-5 看到可用的数字格式。对字符列，使用指定的格式为 A*xx*，这里的 *xx* 是代表列字符宽度的数量。SQL*Plus 不能规定DATE 列的格式；日期格式必须在SELECT语句中使用TO_CHAR函数指定。

HEA[DING]

定义列字符串。如果字符串包括空格或标点，必须封装在单引号或双引号中。HEADSEP 字符默认是垂直线 (|)，它可以用于放置标题的行分隔线。

JUST[IFY]

按指定格式对齐标题。NUMBER列的标题默认采用右对齐，而其他列的标题的对齐方式为左对齐。

LIKE

复制命名列、表达式或别名的显示属性，并将它们应用到定义格式的列上。

NEWL[INE]

在显示列或表达式的值之前开始新的一行。

NEW_V[ALUE]

指定在特定变量中保存的列值，该列值然后可以用在TTITLE和BTITLE命令中。

NOPRI[NT]

阻止列或表达式的打印。

NUL[L] text

当列值或表达式的值为NULL时，指定显示的字符串。

OLD_V[ALUE]

指定列或表达式的以前值将被保存在指定变量中，这些值然后可以用在TTITLE和BTITLE命令中。

OFF

使列或表达式的显示属性关闭而不影响属性的定义。

ON

当以前的列或表达式的显示属性已经被关闭以后，使大家能够设置其属性。

TRU[NCATED]

指定截取对一列而言太长的值。

WOR[D_WRAPPED]

指定把对一列太长的值放置到下一行中，用不满足列宽度的字符串的第一个完全字符开始。

WRA[PPED]

指定把对一列太长的值放置到下一行中，采用不满足列宽度的字符串的第一个字符开始。

COMPUTE

```
COMP[UTE] [function [LABEL text][ function [LABEL text]] . . . ]
 OF [{expr | column | alias} ...]
 ON {expr | column | alias | REPORT | ROW}
```

使 SQL*Plus 用被选择的行的子集计算并显示摘要行。

关键字

function ...

是下列的一个或多个标准数学函数；如果指定不止一个函数名，则至少使用一个空格来分隔该名字：

AVG
COU[NT]
MAX[IMUM]
MIN[IMUM]
NUM[BER]
STD
SUM
VAR[IANCE]

LABEL

指定该文本被用作计算值的打印标签。打印标签左对齐，而且被截取为列的宽度或行的大小，取最小值。最大的标签长度是500个字符。如果省略该关键字，就默认 text 为未缩写的函数的关键字。

注意：计算值的标签出现在特定的分隔列中。要压缩标签，可以在分隔列上使用COLUMN命令的 NOPRINT 选项。

OFF

指定要在计算中使用的表达式、列或别名。这些选项还必须列在应用 COMPUTE 的 SELECT 语句中。

ON

指定当实体值发生改变时，触发COMPUTE显示计算值的实体。ON必须与对应的 BREAK 语句相匹配。

REPFOOTER (控制)

REPF[OOTER] ON | OFF

在不影响定义的情况下，打开或关闭报表页脚。

关键字

ON

指定打印报表页脚。

OFF

指定不打印报表页脚。

REPFOOTER (定义)

REPF[OOTER] [PAGE] [*printspec* [*text* | *variable*] [*printspec* [*text* | *variable*] . . .]]

在每一张报表的底部放置报表页脚。

关键字

PAGE

指定打印报表页脚之后开始新的一页。

printspec

指定将使用下列的一个或多个从句放置和规定文本或变量。

COL

在列 *n* 中启动文本或变量。

S[KIP]

在打印文本或变量之前跳过 *n* 行。默认数字是跳过 1 行。

TAB

在打印文本或变量之前向后跳过 *n* 个打印位置（如果n为负数，表示回跳）。

LE[FT]

按左对齐打印文本或变量。

CE[NTER]

居中打印文本或变量。LINESIZE 设置值用于计算页的宽度。

RI[GHT]

按右对齐打印文本或变量。

BOLD

以黑体打印文本或变量。

FORMAT

根据 *formatstring* 中指定的格式模型规定文本或变量的格式。要得到可用的数字格式，参见表 5-5。

text

作为页脚部分被打印的字符串。文本包括的空格或标点符号必须包含在单引号中。

variable

用户变量或系统维护变量（如SQL.PNO代表页码）。变量的内容将显示为页脚的一部分。

REPFOOTER (显示)

REPF[OOTER]

显示当前报表页脚的定义。

REPHEADER (控制)

REPH[EADER] ON | OFF

在不影响定义的情况下打开或关闭报表标题。

REPHEADER (定义)

REPH[EADER] [PAGE] [*printspec* [*text* | *variable*] [*printspec* [*text* | *variable*]] . . .]

在每一张报表的上部放置报表标题。

关键字

PAGE

指定打印标题之后开始新的一页。

printspec

指定将使用下列的一个或多个从句放置和格式化文本或变量。

COL

在列 *n* 中启动文本或变量。

S[KIP]

在打印文本或变量之前跳过 *n* 行。默认数字是跳过 1 行。

TAB

在打印文本或变量之前向后跳过n个打印位置（如果n为负数，表示回跳）。

LE[FT]

按左对齐打印文本或变量。

CE[NTER]

居中打印文本或变量。LINESIZE 设置值用于计算页的宽度。

RI[GHT]

按右对齐打印文本或变量。

BOLD

以黑体打印文本或变量。

FORMAT

根据 *formatstring* 中指定的格式模型规定文本或变量的格式。要得到可用的数字格式，参见表 5-5。

text

作为报表标题一部分被打印的字符串。文本包含的空格或标点符号必须放在引号中。

variable

用户变量或系统保留变量（如SQL.PNO代表页码）。变量的内容将作为报表标题的一部分显示。

REPHEADER (显示)

REPH[EADER]

显示当前报表标题的定义。

TTITLE (控制)

TTI[TLE] ON | OFF

在不影响顶部标题定义的情况下打开或关闭顶部标题。

TTITLE (定义)

TTI[TLE] [*printspec* [*text* | *variable*] . . .]

创建和规定在每一页顶部出现的标题及格式。

关键字

printspec

指定将使用下列从句的一个或多个放置文本或变量和规定文本或变量的格式。

COL

在列 *n* 中启动文本或变量。

S[KIP]

在打印文本或变量之前跳过 *n* 行。默认数字是跳过 1 行。

TAB

在打印文本或变量之前向后跳过 *n* 个打印位置（如果 *n* 为负数，表示回跳）。

LE[FT]

按左对齐打印文本或变量。

CE[NTER]

居中打印文本或变量。LINESIZE 设置值用于计算页的宽度。

RI[GHT]

按右对齐打印文本或变量。

BOLD

以黑体打印文本或变量。

FORMAT

根据*formatstring*中指定的格式模型规定文本或变量的格式。要得到可用的数字格式，参见表5-5。

text

作为页标题的一部分被打印的字符串。文本包含的空格或标点符号必须包括在引号中。

variable

用户变量或系统保留变量（如SQL.PNO代表页码）。变量的内容将作为页头的一部分显示。

TTITLE（显示）

TTI[TLE]

显示当前TTITLE的定义。

各种各样的SQL*Plus命令

本节中描述的SQL*Plus命令用于执行各种各样的任务。这些命令可以从SQL*Plus提示符下直接输入，也可以包含在SQL*Plus脚本中。

@

@filename

运行一系列的SQL*Plus命令和/或包含在文件中的SQL和/或PL/SQL语句。*filename*指定了包含SQL和/或PL/SQL语句的操作系统文件名。

注意：该命令等于指定START文件名。

可以使用命令文件嵌套；换句话说，一个命令文件可以包含另一个*@filename*命令。

@@

```
@@filename
```

运行一系列的SQL*Plus命令，和/或包含在文件中的SQL和/或PL/SQL语句，作为嵌套命令。*filename*指定了将被SQL*Plus执行的包含SQL和/或PL/SQL语句的操作系统文件名。当该脚本完成时，将返回调用该脚本的控制。如果没有指定路径，则认为被调用的文件与其父路径相同。

ARCHIVE LOG

```
ARCHIVE LOG {LIST |
             STOP |
             START     [TO destination] |
             NEXT      [TO destination] |
             ALL       [TO destination] |
             log_sequence [TO destination]
            }
```

控制或显示归档日志的信息。该命令是DBA使用。

关键字

LIST

指定该SQL*Plus将显示有关当前归档状态的信息。

STOP

指定停止日志文件的自动归档。

START

指定启动日志文件的自动归档。如果要指定目的地，则将所有的归档日志文件书写到目的地。

NEXT

指定下一个恢复日志文件组（如果被填充的话）使用手工归档。如果指定了到目的文件中，则恢复日志文件组将只采用该命令书写到该目的文件中。

ALL

指定手工归档所有恢复日志文件组。如果指定了到目的文件中，则恢复日志文件组将只采用该命令书写到该目的文件中。

log_sequence

指定手工归档由*log_sequence*定义的日志文件组。如果指定了到目的文件中，则恢复日志文件组将只采用该命令书写到该目的文件中。

CONNECT

```
CONN[ECT] username[/password][@hostname] [AS {SYSOPER | SYSDBA}]
```

使用特定的用户名和口令连接到数据库中。

```
CONN[ECT] /[@hostname] [AS {SYSOPER | SYSDBA}]
```

使用操作系统授权连接到数据库中。

```
CONN[ECT] INTERNAL[/password]
```

连接到内核数据库中。

关键字

username

连接到数据库时使用的Oracle用户名。

password

与提供的用户名相关的口令。如果省略，则SQL*Plus将提示录入口令。

hostname

分配给数据库的连接主机名。该*hostname*通常使用*tnsnames.ora*文件或Oracle名来解决。

AS SYSOPER

指定该用户将作为操作员进行连接。

AS SYSDBA

指定该用户将作为数据库管理员进行连接。

/

指定已经登录的操作系统用户名将与*INIT.ORA*参数OS_AUTHENT_PREFIX（默认为OPS$）连接。最终用户名必须作为一个数据库用户存在（用CREATE USER *username* IDENTIFIED EXTERNALLY 创建）。例如，如果OS_

AUTHENT_PREFIX 被设置为默认值，而且作为SCOTT登录到操作系统或网络中，则必须存在 Oracle 用户名 OPS$SCOTT，而且具有 CONNECT 权限。

INTERNAL

指定进行内核连接。

COPY

```
COPY {FROM username[/password][@hostname] |
      TO username[/password]@hostname] |
      FROM username[/password][@hostname] TO username[/password][@hostname]}
{APPEND | CREATE | INSERT | REPLACE} destination_table [(column[,column . . .])]
USING query)
```

复制查询返回的数据到本地或远端数据库的另一张表中。

关键字

username

从该用户账户（架构）名复制数据。

password

与特定用户账户相关的口令。

Hostname

连接数据库的 SQL*Net 或 Net8 连接字符串。

FROM

指定要从该数据库复制数据的用户名、口令和主机名。如果省略，则使用当前登录的 SQL*Plus 账户。

TO

指定要复制数据的数据库的用户名、口令和主机名。如果省略，则使用当前登录的 SQL*Plus 账户。

APPEND

如果目的表存在，则插入行到目的表中；否则，创建目的表。

CREATE

在插入行之前创建目的表。如果目的表已经存在，则产生错误。

INSERT

向目的表中插入行。如果目的表不存在，则产生错误。

REPLACE

如果存在，则删除目的表，而且在复制行之前创建一个新版本。

destination_table

接收被复制的行的表名。

column

被插入数据的目的表的列名。如果指定列，则它们的号码必须与查询返回的列号相匹配。

query

任何有效的 SQL SELECT 语句。该语句用于返回被复制的行和列。

DESCRIBE

DESC[RIBE] {[*schema*.]*object*[@*hostname*] [*column*] | [*schema*.]*object*[.*subobject*]}

列出指定数据库对象的定义。

关键字

schema

指定包含被描述的对象的架构。

object

被描述的对象名。该名字可以是表、视图、同义词、函数、过程或包。

hostname

包含被描述的对象的数据库的 Net8 连接串。

column

指定表的列名（只针对 Oracle7）。

subobject

包中函数或过程名。

DISCONNECT

DISC[ONNECT]

提交所有挂起的事务，且在不退出 SQL*Plus 的情况下断开当前用户与数据库的连接。

EDIT

ED[IT] [*filename*]

调用系统编辑器（通过用户变量 _EDITOR 值指定）。*filename* 指定用编辑器打开文件名。如果 *filename* 不存在，则缓冲中当前的内容被传递到编辑器中，而且在编辑完成时返回到缓存中。

EXECUTE

EXE[CUTE] *statement*

执行单个 PL/SQL 语句，通常用于执行存储过程。

示例

```
EXECUTE scott.funclib.do_calc;
```

EXIT

EXIT [SUCCESS | FAILURE | WARNING | *n* | *var* | :*bind_variable*]
[COMMIT | ROLLBACK]

提交挂起的事务（除非指定了 ROLLBACK），传递返回码到操作系统中，并退出 SQL*Plus。

关键字

SUCCESS

正常退出，并用一个返回码表示成功。该选项是默认行为。

FAILURE

用一个显示失败的返回码退出。

WARNING

用一个显示警告的返回码退出。

n

用一个特定的返回码退出。

var

使用特定的用户定义的或系统变量的值作为返回码。

bind_variable

使用特定的绑定变量值，采用VARIABLE命令创建，作为返回码。

COMMIT

退出之前提交对数据库挂起的改变。

ROLLBACK

退出之前回退挂起的改变。

HELP

```
HELP [topic]
```

如果指定的话，就调用SQL*Plus帮助系统并显示主题帮助信息。主题来源于SQL*Plus主题的列表；如果不指定主题，则显示主题列表。

警告：有时也无法使用HELP。例如，HELP命令不能在Windows平台上执行，而其他平台可能需要DBA特别安装帮助功能。

HOST

```
HO[ST] [command]
```

在不离开SQL*Plus的情况下执行操作系统命令；*command*可以是任何有效的操作系统命令。如果不指定，将出现系统命令提示符，可以在这里执行，而且将在录入EXIT命令之前均在系统提示符下。

PAUSE

```
PAU[SE] [text]
```

显示空行、后面是包含文本的行（如果指定的话）、并等待用户按下回车键；文本是出现在输出设备中的字符串（不需要用引号）。

RECOVER

```
RECOVER {[AUTOMATIC] [FROM location]
         {[STANDBY] DATABASE
          [UNTIL {CANCEL | CHANGE change_number | TIME time} |
             USING BACKUP CONTROLFILE |
          [STANDBY] DATAFILE filename[,filename ...]
            [UNTIL [CONSISTENT [WITH] CONTROLFILE |
          [STANDBY] TABLESPACE tablespace[,tablespace ...]
            [UNTIL [CONSISTENT [WITH] CONTROLFILE |
          TABLESPACE tablespace[,tablespace ...] |
          DATAFILE filename[,filename ...] |
          LOGFILE filename |
          CONTINUE [DEFAULT] |
          CANCEL
         } |
          MANAGED STANDBY DATABASE {TIMEOUT integer | CANCEL [IMMEDIATE]}
        }
[PARALLEL integer | NOPARALLEL]
```

在数据库、表空间、数据文件或日志文件上执行介质恢复。该命令适用于DBA。

关键字

AUTOMATIC

指定在恢复操作中，下一个需要的归档日志文件将使用*INIT.ORA* 文件中的LOG_ARCHIVE_DEST (或 LOG_ARCHIVE_DEST_1)和LOG_ARCHIVE_FORMAT参数自动产生。如果省略该关键字，而且没有指定LOGFILE，则系统将提示用户输入需要的每一个归档日志文件名。

FROM

指定被读的归档日志文件的位置。*location*必须采用取决于操作系统的格式提供，而且如果指定该关键字，则*location*的值将不考虑*INIT.ORA*文件的 LOG_ARCHIVE_DEST 或 LOG_ARCHIVE_DEST_1 参数值。

STANDBY

指定备用数据库（必须是安装但没有被打开的），将用控制文件进行恢复，而且将从主数据库中复制归档恢复日志文件。

DATABASE

指定被恢复的数据库。

UNTIL CANCEL

指定要执行一个不完整的、基于取消的恢复。通过用归档恢复日志文件的建议名提示进行恢复处理，而且当录入“CANCEL”而不是文件名时，完成恢复操作。

UNTIL CHANGE

指定执行不完整的、基于改变的恢复操作。*Integer*是紧随恢复的最近一次变化的系统改变号（System Change Number，SCN）。

UNTIL TIME

指定执行不完整的、基于时间的恢复，恢复到提供的时间为止，时间必须使用‘YYYY-MM-DD:HH24:MI:SS’格式封装在单引号中。

USING BACKUP CONTROLFILE

指定使用备份控制文件而不是当前控制文件执行恢复。

DATAFILE

指定恢复丢失的或损坏的数据文件。如果指定STANDBY，则使用从主数据库上复制的控制文件和归档恢复日志文件，恢复备用数据库中的数据文件。

UNTIL CONSISTENT WITH CONTROLFILE

指定备用数据文件（或表空间）的恢复使用当前备用数据库控制文件。

TABLESPACE

指定名为 *tablespace* 的表空间被恢复。在恢复中最多可以列出 16 个表空间。

LOGFILE

指定应用 *logfile* 定义的日志文件继续恢复。

CONTINUE

在被中断以禁止一个线程以后，指定继续进行多实例恢复。

DEFAULT

如果不指定其他日志文件，则指定使用将要自动产生的恢复日志文件继续恢复。该关键字等于指定 AUTOMATIC，但 Oracle 没有提示文件名。

CANCEL

指定基于取消的恢复被停止（使用 UNTIL CANCEL 关键字）。

MANAGED STANDBY DATABASE

假设备用数据库是整个备用数据库结果中的一个活跃组件，在此前提下执行持续的备用恢复。主数据库活跃的恢复日志文件被归档到备用位置，在这里它们将用于管理备用恢复操作。

TIMEOUT

以整数分钟的形式，指定持续恢复操作等待需要的归档日志恢复可以写到备用数据库上的等待时间。如果在该时间内恢复日志文件不可用，则恢复进程被终止，并提供错误信息。该语句可以在返回到持续备用恢复模式下再次执行。如果没有指定该关键字，则数据库保留在持续备用恢复模式，直到重新执行了 RECOVER CANCEL 从句或关闭了实例或发生错误为止。

CANCEL IMMEDIATE

指定应用了当前归档恢复文件中的所有恢复记录，或者在下一个恢复日志文件被读之后（取先出现者），终止持续恢复操作。

PARALLEL integer

指定使用并行度（并行操作中使用的并行线程的数量）执行恢复，并行度等于 CPU 在所有期待的实例上可用的数量乘以 INIT.ORA 文件中的 PARALLEL_THREADS_PER_CPU 参数值。如果提供了一个 *integer*，则将使用并行度代替自动取消。PARALLEL 关键字不考虑 *INIT.ORA* 中的 RECOVERY_PARALLELISM 初始化参数。

NOPARALLEL

指定执行串行恢复，而不是并行。该选项是默认值。

注意：该命令等于服务器管理中的 RECOVER 命令，从 SQL*Plus 8.1 版本启用。恢复操作很复杂，最好是有经验的DBA进行尝试。要了解更多信息，参见Oracle的《Oracle8*i* Backup and Recovery Guide》。

REMARK

REM[ARK] [*text*]

表示将同一行上紧随的所有字符当作注释。

SET

SET *system_variable value*

设置 SQL*Plus 系统变量为指定值。

关键字

system_variable

有效的 SQL*Plus 系统变量名（见本章后面的“SQL*Plus 系统变量”）。

value

系统变量的有效值。

SHOW

SHO[W] [*var*]

列出指定的 SQL*Plus 设置的值，或者所有设置的值。

SHO[W] ERR[ORS] [*type*] [[*schema*.]*name*]

显示尝试创建或代替已存储的 PL/SQL 程序单元的编译错误。

关键字

var

可以是下列值之一：

ALL

BTI[TLE]

LABEL (Oracle8*i* 后废止)

LNO

PARAMETERS
PNO
REL[EASE]
REPF[OOTER]
REPH[EADER]
SGA
SPOO[L]
SQLCODE
TTI[TLE]
USER

除了这些指定的项目之外，大家也可以使用任何可以与 SET 命令一起使用的 SQL*Plus 系统变量。

ERRORS

在没有附加关键字时，显示最近编译的PL/SQL过程的错误。当包括类型和名字时，将显示该对象的最近编译的错误。

type

可以是下列值之一：

FUNCTION
PROCEDURE
PACKAGE
PACKAGE BODY
TRIGGER
VIEW
TYPE
TYPE BODY

schema

指定包含命名对象的架构名。如果省略，则假定为当前架构。

name

指定显示错误的存储对象名。

SHUTDOWN

SHUTDOWN [NORMAL | IMMEDIATE | TRANSACTIONAL | ABORT]

关闭数据库实例。该命令适用于DBA。

关键字

NORMAL

指定执行正常关闭。正常关闭将在实例关闭之前阻止新用户登录，并等待所有用户断开连接。

IMMEDIATE

指定执行立即关闭。立即关闭将在实例关闭之前阻止新用户登录，并在当前执行的SQL语句完成以后断开所有用户的连接。当所有用户断开以后，该实例将被关闭。

TRANSACTIONAL

指定执行事务关闭。事务关闭阻止新用户登录并断开事务已经完成的任何用户（即没有挂起的COMMIT）。因为每个剩下的用户通过COMMIT或ROLLBACK语句完成了当前事务，所以断开会话。当断开所有用户时实例将被关闭。

ABORT

指定该实例将不考虑挂起的事务而关闭实例。在关闭之前不写回滚信息且不将“脏”缓冲写入磁盘中。当启动数据库时，系统将执行紧急（crash）恢复。

注意：该命令等于服务器管理器SHUTDOWN命令，从Oracle8.1版启用。

SPOOL

SPOOL

显示当前假脱机的状态。

SPO[OL] [*filename*[.*extension*]] | [OFF | OUT]

将所有随后的 SQL*Plus 输出到文件中。

关键字

filename[.extension]

文件的文件名和可选扩展名，该文件将被写入输出内容。如果不提供扩展名，则使用 *.LIS* 或 *.LST* 作为扩展名(取决于操作系统)。

OFF

停止假脱机。

OUT

停止假脱机，并将输出文件发送到默认打印机上。并不是所有平台上都支持该选项。例如，在 Windows 平台上就不支持。

警告：SPOOL OFF 命令必须在假脱机完成时提供，否则不保存假脱机文件。

START

STA[RT] *filename* [*arg1* [*arg2*...]]

运行在文件中包含的一系列的 SQL*Plus 命令和/或 SQL 和/或 PL/SQL 语句。

关键字

filename

指定包含 SQL 和/或 PL/SQL 语句的操作系统文件名。

arg

要传递到脚本中的一个或多个参数。这些参数可以在SQL或PL/SQL脚本中作为 &1、&2 等等被引用。参数是定位的，而且中间用空格分开。

注意：该命令等于指定 @*filename*。

命令文件可以被嵌套，换句话说，命令行文件可以包含另一个 START *filename* 命令。

STARTUP

```
STARTUP [FORCE] [RESTRICT]
[PFILE=filename]
{MOUNT | OPEN [database_name] | NOMOUNT}
[EXCLUSIVE | {PARALLEL | SHARED} [RETRY]]
```

启动 Oracle 实例。该命令适用于 DBA。

关键字

FORCE

指定将强制启动实例。如果实例已经在运行，则该关键字在指定的STARTUP之后强制执行 SHUTDOWN ABORT。

RESTRICT

指定以强制模式打开数据库，该操作只允许具有RESTRICTED SESSION权限的用户进行连接。

PFILE

指定应该使用提供的 *pfilename* 文件中的参数启动实例。该文件必须在运行 SQL*Plus 的机器上可用。

MOUNT

指定要启动该实例，而且将安装数据库，但不打开。

OPEN

指定要启动该实例，而且安装并打开数据库。该选项是默认值。

database_name

指定数据库名，而且不考虑初始化文件中的 DB_NAME 参数。

NOMOUNT

指定启动数据库，但既不安装也不打开。

EXCLUSIVE

指定将只用该实例安装和/或打开数据库，且不能被其他实例共享。

PARALLEL

指定将只用该实例安装和/或打开数据库，从而其他实例可以同时访问。该关键字是 SHARED 的同义词。

SHARED

指定将只用该实例安装和/或打开数据库，从而其他实例可以同时访问。该关键字是PARALLEL的同义词。

RETRY

指定如果另一实例在数据库上执行恢复，则Oracle将每5分钟重试一次打开操作，直到数据库恢复完成而且操作成功为止。

注意：该命令等于服务器管理器的STARTUP命令，从Oracle 8.1版启用。

STORE

```
STORE SET filename [CRE[ATE] | REP[LACE] | APP[END]]
```

将当前SQL*Plus的环境属性保存在文件中，作为SET命令的列表。

关键字

filename

指定将包含SET命令的操作系统文件名。

CREATE

创建新文件。如果指定该选项，则该文件名一定不能已经存在。

REPLACE

代替现存文件的内容。如果文件名不存在，则创建该文件。

APPEND

添加缓冲中的内容到指定文件的结尾。

TIMING

```
TIMING
```

列出活动的定时器的数量。

```
TIMI[NG] [START text | SHOW | STOP]
```

启动、停止或列出共用的定时器。

关键字

START

设置定时器并将其命名为文本。

SHOW

列出当前的定时器名和定时数据。

STOP

列出当前的定时器名和定时数据，然后删除定时器。

注意：不要把 SET TIMING ON 和 SET TIMING OFF 命令同 TIMING 命令相混淆。TIMING 命令控制了活跃的定时器，而 SET TIMING 命令控制了单个 SQL 语句占用时间的自动报表。

WHENEVER OSERROR

```
WHENEVER OSERROR {EXIT {SUCCESS | FAILURE | n | var | :bind_variable}
                  [COMMIT | ROLLBACK] |
                 CONTINUE [COMMIT | ROLLBACK | NONE]
                }
```

指定操作系统发生错误时由 SQL*Plus 执行的行为。

关键字

EXIT

只要操作系统检测到了错误，就直接退出 SQL*Plus。

COMMIT

在操作系统错误发生时，退出或继续之前，指示SQL*Plus执行COMMIT命令。

CONTINUE

关闭 EXIT 选项。

FAILURE

退出时返回一个显示错误的代码。

n

退出并返回特定数字的返回码。

NONE

发生操作系统错误之后，指示 SQL*Plus 不采取任何行为。

ROLLBACK

发生操作系统错误之后，在退出或继续之前，指示SQL*Plus执行ROLLBACK。

SUCCESS

正常退出，采用返回码显示成功。这是默认行为。

var

用户定义的或系统变量，该值将用作返回码。

bind_variable

绑定变量，采用 VARIABLE 命令在 SQL*Plus 中创建，该值将用作返回码。

WARNING

退出时采用返回码显示警告状态。

WHENEVER SQLERROR

```
WHENEVER SQLERROR {EXIT {SUCCESS | FAILURE | n | var | :bind_variable}
                    [COMMIT | ROLLBACK] |
                   CONTINUE [COMMIT | ROLLBACK | NONE]
                  }
```

指定 SQL 语句或 PL/SQL 块产生错误时由 SQL*Plus 采取的行为。

关键字

EXIT

只要检测到 SQL 或 PL/SQL 错误，就直接退出 SQL*Plus。

COMMIT

检测到操作系统错误时，在退出或继续之前，指示SQL*Plus执行COMMIT命令。

CONTINUE

关闭 EXIT 选项。

FAILURE

退出时返回一个显示错误的代码。

n

退出并返回特定数字的返回码。

NONE

发生 SQL 或 PL/SQL 错误之后，指示 SQL*Plus 不采取任何行为。

ROLLBACK

检测到 SQL 或 PL/SQL 错误时，在退出或继续之前，指示 SQL*Plus 执行 ROLLBACK。

SUCCESS

正常退出，采用返回码显示成功。这是默认行为。

var

用户定义的或系统变量，该值将用作返回码。

bind_variable

绑定变量，采用 VARIABLE 命令在 SQL*Plus 中创建，该值将用作返回码。

WARNING

退出时采用返回码显示警告状态。

SQL*Plus 变量和相关命令

SQL*Plus提供了创建和使用用户变量的机制。在本节中描述的命令和结构用于操纵SQL*Plus用户变量。大家可以直接从SQL*Plus提示符下输入这些命令或构建这些命令，或者将它们包含在 SQL*Plus 脚本中。

&

&n

插入命令行参数。

&var

创建 SQL*Plus 变量 *var*，而且每次遇到 *&var* 时，提示输入 *&var* 的值。

关键字

n

指定将被插入的参数的命令行位置。每一次发生的 *&n* 将采用命令行的相应参数来代替。例如，&1 将用命令行中指定的第一个参数值来代替。

var

指定在执行语句或命令时提供值代替 SQL*Plus 变量名。运行 SQL*Plus 的终止会话将提示该值。

&&

&&var

创建一个赋予某值的SQL*Plus变量，方便以后使用。*var* 指定了在语句或命令被执行时提供的值代替 SQL*Plus 变量名。第一次出现 *&&var* 时，运行 SQL*Plus 的终止会话提示输入该值，而且一直保留该值，直到 SQL*Plus 会话终止为止，或者直到 *var* 解除定义为止。

ACCEPT

ACC[EPT] *var* [*type*] [FORMAT *formatstring*] [DEFAULT *default*] [PROMPT *text* | NOPROMPT] [HIDE]

读取输入的一行并将其存储在变量中。

关键字

var

指定将要保留所提供值的变量名。

type

限制允许的输入类型，可以是 NUMBER、CHAR 或 DATE。

FORMAT

为输入指定将要用做输入格式的*formatstring*。参见表 5-5 得到可用的数字格式。

DEFAULT

指定如果没有提供输入则采用默认值。

PROMPT

指定接受输入之前将文本显示为提示符。

HIDE

指定不将输入值显示在屏幕上。

DEFINE

```
DEF[INE] var = text
```

创建变量并分配文本字符串。

```
DEF[INE] [var]
```

列出指定用户变量的当前值或者所有用户变量的当前值。

关键字

var

指定 SQL*Plus 变量名。

text

指定 CHAR 文本字符串被存储在变量中。

PRINT

```
PRI[NT] bind_variable
```

显示变量的当前值。*bind_variable* 指定将被显示的绑定变量名（利用 VARIABLE 命令创建）。

UNDEFINE

```
UNDEF[INE] var
```

删除用户变量；*var* 指定被删除的用户变量名。

VARIABLE

```
VAR[IABLE] bind_variable
  {NUMBER | CHAR | VARCHAR2 | NVARCHAR2 | CLOB | NCLOB | REFCURSOR}
```

定义可以在 PL/SQL 中引用的绑定变量。

```
VAR[IABLE]
```

显示所有定义的被绑定变量的清单。

关键字

bind_variable

指定要被创建的绑定变量名。

NUMBER

指定绑定变量是 NUMBER 变量。

CHAR

指定 *bind_variable* 是 CHAR 变量。

VARCHAR2

指定 *bind_variable* 是 VARCHAR2 变量。

NVARCHAR2

指定 *bind_variable* 是 NVARCHAR2 变量。

CLOB

指定 *bind_variable* 是 CLOB 变量。

NCLOB

指定 *bind_variable* 是 NCLOB 变量。

REFCURSOR

指定 *bind_variable* 是 REFCURSOR 变量。

注意：要了解有关这些PL/SQL变量类型的更多信息，参见第七章。

SQL*Plus 系统变量

本节中描述的SQL*Plus系统层变量控制SQL*Plus操作的各个方面，而且可以在SQL*Plus SET和SHOW命令中引用。

APPINFO

APPI[NFO]{ON | OFF | *text*}

通过嵌入的DBMS_APPLICATION_INFO包自动注册命令文件，从而使每一个命令文件的性能和资源使用能够被DBA监控。

关键字

ON

指定能够自动注册。

OFF

指定不能进行自动注册。

text

指定APPINFO被设置为OFF时，如果没有运行命令文件，则注册该文本。

ARRAYSIZE

ARRAY[SIZE] {*n*}

设置SQL*Plus将同时从数据库中提取的行数；*n*指定了同时提取的行数。有效值是1到5000，默认值是15。

AUTOCOMMIT

AUTO[COMMIT] {ON | OFF | IMM[EDIATE] | *n*}

在 Oracle 给数据库提交挂起的改变时进行控制。

关键字

ON

在成功执行每一条 SQL 语句或 PL/SQL 块之后，向数据库提交挂起的改变。

OFF

禁止自动提交，所以必须手工提交改变。该选项是默认值。

IMMEDIATE

该关键字的功能与 ON 相同。

n

在每 *n* 条 SQL 语句或 PL/SQL 块成功以后提交。

AUTOPRINT

AUTOP[RINT] {ON | OFF}

控制绑定变量的自动打印。

关键字

ON

指定在每一个 PL/SQL 块或 SQL 语句被引用后自动打印绑定变量。

OFF

指定不自动打印绑定变量。该选项是默认值。

AUTORECOVERY

AUTORECOVERY {ON | OFF}

控制在没有用户介入的情况下是否运行 RECOVER 命令。该变量适用于 DBA。

关键字

ON

指定使用自动恢复。自动恢复允许RECOVER命令自动运行，只要归档日志文件定位在INIT.ORA 文件中的LOG_ARCHIVE_DEST参数，指向的位置就可以了。

OFF

指定不能进行自动恢复，而且使用RECOVER命令时，需要用户的介入。该选项是默认值。

AUTOTRACE

```
AUTOT[RACE] {ON | OFF | TRACE[ONLY]} [EXP[LAIN]] [STAT[ISTICS]]
```

使 SQL*Plus 显示 SQL DML 语句执行成功的报告。

关键字

ON

引起每一条 DML 语句执行之后显示跟踪报表。

OFF

不显示跟踪报表。该选项是默认值。

TRACEONLY

显示跟踪报表，但禁止输出查询数据。

EXPLAIN

使SQL*Plus显示每一条SQL DML语句执行的查询执行计划。使用EXPLAIN PLAN 语句得到执行计划，而且为了让该选项起作用，计划表必须具有架构。

STATISTICS

使 SQL*Plus 为每一条执行的语句显示统计信息。

BLOCKTERMINATOR

```
BLO[CKTERMINATOR] [c]
```

设置非字母字符用来结束PL/SQL块；*c*指定终止PL/SQL块条目的字符，但不会使块被执行。要执行块，必须随后运行RUN或/（斜线）命令。默认值是一个周期。

CLOSECURSOR

CLOSECUR[SOR] ON | OFF

控制执行SQL语句之后SQL*Plus是否关闭游标。

警告：该变量从Oracle8*i*开始废止。

CMDSEP

CMDS[EP] {*c* | ON | OFF}

设置字符分离在一行录入的多条SQL*Plus命令。

关键字

c

指定用于分离SQL*Plus命令的字符（必须为非字母）。默认值为分号（;）。

ON

打开在一行上输入多行命令的能力，而且自动设置命令分隔符为分号（;）。

OFF

关闭在一行输入多条命令的能力。

COLSEP

COLSEP *text*

设置查询返回的列之间被输出的文本。默认的*text*是空格。

COMPATIBILITY

COM[PATIBILITY] V7 | V8 | NATIVE

指定当前连接的会话的 Oracle 版本。

关键字

V7

指定 SQL*Plus 会话被连接到 Oracle7 数据库上，或者，如果该 SQL*Plus 会话连接到 Oracle8 数据库上，则其行为应该效仿 Oracle7。

V8

指定 SQL*Plus 会话被连接到 Oracle8 数据库上。

NATIVE

显示用户想让数据库确定该设置。例如，如果连接到 Oracle8，则默认的兼容能力就是 V8。

CONCAT

CON[CAT] *c* | ON | OFF

设置用于终止代替变量引用的字符。

关键字

c

指定终止代替变量引用的字符，如果想立即在变量引用之后采用字符，那么 SQL*Plus 将用别的方式解释为代替变量名的一部分。默认值是一个句点。

ON

复位句点的 CONCAT 值。

OFF

指定不设置 CONCAT 字符。

COPYCOMMIT

COPYC[OMMIT] *n*

控制批处理的数量，在这之后 COPY 命令将变化提交到数据库中；*n* 指定执行提交

之前，由COPY命令复制的批处理数。因为批处理的大小由ARRAYSIZE控制，所以每次提交之前复制的行数将是ARRAYSIZE * *n*。如果COPYCOMMIT被设置为0（默认值），则COPY只在复制操作的结尾执行提交。

COPYTYPECHECK

COPYTYPECHECK ON | OFF

在使用COPY命令从一张表中复制命令到另一张表中时，控制SQL*Plus是否通过检查，以确保源数据类型是否与目的数据类型相匹配。

DEFINE

DEF[INE] *c* | ON | OFF

设置用作前缀替代变量的字符。

关键字

c

改变定义字符值为该字符。

ON

改变定义字符值返回到默认的“&”，而且能够定义该特性。

OFF

关闭定义特性。

ECHO

ECHO ON | OFF

在执行START命令时，控制SQL*Plus是否列出被执行的命令文件的每条语句。

EDITFILE

EDITF[ILE] *filename*

设置EDIT命令的默认文件名。*filename*指定了当执行EDIT命令时，将被执行的文件名。要了解有关EDIT命令的更多信息，参见本章的EDIT。

EMBEDDED

EMB[EDDED] ON | OFF

控制在一页的什么地方开始每一个报表。

关键字

ON

允许报表从一页的任何地方开始。页号将从一张报表到另一张报表继续进行。

OFF

强制每一张报表从新页的顶部开始。每张报表的页号将是从1开始。该选项是默认值。

ESCAPE

ESC[APE] *c* | ON | OFF

定义用作转义字符的字符，SQL*Plus将把代替字符当作普通字符，而不是需要的变量代替符。

关键字

c

改变转义字符的值为该字符。

ON

改变转义字符值返回到默认的反斜线（"\"），而且允许转义功能。

OFF

关闭转义功能。该选项是默认值。

FEEDBACK

FEED[BACK] {6 | n | OFF | ON}

指定显示被查询返回的记录数的阈值。

关键字

n

指定在显示反馈数字之前需要被返回的记录数量。默认值为6。

ON

允许显示反馈，设置 *n* 为1。

OFF

关闭反馈显示。

FLAGGER

FLAGGER OFF | ENTRY | INTERMED[IATE] | FULL

检查以确保SQL语句遵守ANSI/ISO SQL-92标准。

关键字

OFF

关闭FIPS标志特性。该选项是默认值。

ENTRY

指定SQL按照SQL-92 Entry标准检查。

INTERMED[IATE]

指定SQL按照SQL-92 Intermediate标准检查。

FULL

指定SQL按照SQL-92 Full标准检查。

注意：即使没有连接到数据库上，也可以执行SET FLAGGER。FIPS标志将在SQL*Plus中保留有效，直到SET FLAGGER OFF (或者 ALTER SESSION SET FLAGGER = OFF) 命令成功执行或退出SQL*Plus为止。当允许FIPS标志时，SQL*Plus显示CONNECT、DISCONNECT和ALTER SESSION SET FLAGGER命令的警告信息，即使它们执行成功也发出警告信息。

FLUSH

FLU[SH] ON | OFF

控制输出被发送到用户显示设备的时间。

关键字

ON

指定在发送到显示设备之前可以缓存输出。该选项是默认值。

OFF

指定立即显示输出。

HEADING

HEA[DING] ON | OFF

在报表中控制列的输出。

关键字

ON

在报表中输出列的标题。该选项是默认值。

OFF

取消列的标题。

HEADSEP

HEADS[EP] *c* | ON | OFF

定义标题分隔符。

关键字

c

改变标题分隔符值为该字符。

ON

改变标题分隔符返回到默认的“|”，而且允许题分离功能。

OFF

关闭标题分离功能。

注意：标题在定义时被解释，所以随后对HEADSEP的变化不会影响到现存的标题定义。

INSTANCE

```
INSTANCE instance_name | LOCAL
```

指定使用CONNECT命令时连接到默认数据库上。

关键字

instance_name

为实例指定Net8服务名。

LOCAL

指定默认实例为本地数据库。该选项是默认值。

注意：本地数据库由操作系统指定设置决定。例如，在Windows中，它使用注册文件中的LOCAL参数设置。

LINESIZE

```
LIN[ESIZE] n
```

设置一行上显示的SQL*Plus的字符的总数；*n*指定环绕到新的一行之前显示的字符总数。默认值为80。

注意：当假脱机输出到一个文件中时，LINESIZE决定了产生的输出记录的长度。但是，要注意可能会有额外字符（例如，回车和换行符），这由具体的操作系统决定。

LOBOFFSET

```
LOBOF[FSET] n
```

设置从什么起始位置提取并显示CLOB和NCLOB数据；*n*指定从字符位置*n*开始提取CLOB和NCLOB数据。默认值为1。

LONG

```
LONG n
```

设置显示LONG、CLOB和NCLOB值和复制LONG值的最大宽度；*n*以字节的方式指定最大宽度。默认值为80。

LONGCHUNKSIZE

```
LONGC[HUNKSIZE] n
```

设置SQL*Plus中提取LONG、CLOB或NCLOB值的增量大小；*n*以字节的形式指定长块的大小。默认值为80。

MARKUP HTML

```
MARK[UP] HTML [ON | OFF]
  [HEAD text]
  [BODY text]
  [ENTMAP {ON | OFF}]
  [SPOOL {ON | OFF}]
  [PRE[FORMAT] {ON | OFF}]
```

控制是否将产生HTML输出。该命令从SQL*Plus 8.1.6开始启用。

关键字

ON

指定使用特定的选项产生HTML输出。

OFF

指定不产生HTML输出。该选项是默认值。

HEAD

指定使用文本作为<HEAD>标签的内容。标题文本必须包括在引号中。默认情况下，文本是‘<TITLE>SQL*Plus Report</TITLE>’的形式。

BODY

指定文本作为<BODY>标签的内容。主体文本必须放在引号中。

ENTMAP ON

指定SQL*Plus将特殊字符“<”、“>”和“&”分别显示为HTML的各自实体<、>和&。

ENTMAP OFF

指定SQL*Plus不改变特殊字符“<”、“>”和“&”的显示。

SPOOL ON

指定SQL*Plus书写HTML标签到SQL*Plus SPOOL命令创建的每个文件的开头和结尾中。

SPOOL OFF

指定SQL*Plus不书写HTML标签到SQL*Plus SPOOL命令创建的每个文件的开头和结尾中。

PREFORMAT ON

指定SQL*Plus在HTML <PRE>标签内部书写输出。

PREFORMAT OFF

指定SQL*Plus不在HTML <PRE> tag内部书写输出。

NEWPAGE

NEWP[AGE] *n* | NONE

在每一页的顶部和顶部标题之间设置打印的空白行的数量。

关键字

n

指定在新页的顶部打印空白行号。如果该值为0，则进纸符将用于标记每一页的开始（包括第一页）。进纸字符将清除最底部的屏幕。默认值为1。

NONE

SQL*Plus 在报表页之间既不打印空白行也不打印进纸符。

NULL

NULL *text*

设置在显示 SQL SELECT 语句结果时用于代表空值的文本。

关键字

text

设置显示 SQL SELECT 语句结果时用于代表 NULL 的文本。

NUMFORMAT

NUMF[ORMAT] *formatstring*

设置显示数字的默认格式。*formatstring* 指定显示数字数据时默认使用的数字格式。要得到可用的数字格式，参见表 5-5。

NUMWIDTH

NUM[WIDTH] *n*

设置显示数字的默认宽度；*n* 指定显示数字数据的数字长度。默认值为 10。

注意：NUMFORMAT 不考虑 NUMWIDTH，所以 SET NUMWIDTH 5 后再用 SET NUMFORMAT -999 999.99，将产生 11 个字符宽度列。

PAGESIZE

PAGES[IZE] *n*

指定每一页中的行数量；*n* 指定每一页上要被打印的行数。

注意：PAGESIZE 可以被设置为0，从而禁止所有的标题、分页、题目、初始空白行和其他格式信息。该选项在假脱机输出到文件时可以使用。

PAUSE

PAU[SE] ON | OFF | *text*

控制显示数据的滚动。

关键字

OFF

指定在显示时不暂停输出。该选项是默认值。

ON

指定在显示时暂停输出。用户必须按下 <ENTER> 键再继续输出。

text

指定在每一个输出页的开始时暂停输出。将显示*text*指定的字符串，而且用户必须按下回车键来继续输出的内容。*text*可以包括终端控制序列来控制像颜色或反显等功能。

RECSEP

RECSEP WR[APPED] | EA[CH] | OFF

控制怎样分离记录。

关键字

WRAPPED

只有在封装的行中，才使SQL*Plus打印记录分隔符（由单行的RECSEPCHAR字符重复 LINESIZE 次数组成）。

EACH

使 SQL*Plus 在每一行之后都打印记录分隔符（由单行的 RECSEPCHAR 字符重复 LINESIZE 次数组成）。

OFF

阻止 SQL*Plus 打印任何记录分隔符。

RECSEPCHAR

```
RECSEPCHAR {' ' | c}
```

定义记录分隔符；*c* 指定使用该字符作为分隔符。

SERVEROUTPUT (Oracle7 语法)

```
SERVEROUT[PUT] ON | OFF} [SIZE n]
```

控制是否 SQL*Plus 显示存储过程或 PL/SQL 块从 SQL*Plus 提示符下执行。这样的输出最常见地由 DBMS_OUTPUT.PUT_LINE 过程产生。

关键字

ON

显示输出。

OFF

禁止 DBMS_OUTPUT.PUT_LINE 输出。该选项是默认值。

SIZE

设置在服务器中可以缓存的输出字节数。*n* 的默认值为 2000，而且 *n* 必须在 2000 和 1 000 000 之间。

SERVEROUTPUT (Oracle8 语法)

```
SERVEROUT[PUT] {ON | OFF}
 [SIZE n]
 [FOR[MAT] {WRA[PPED] | WOR[D_WRAPPED] | TRU[NCATED]}]
```

控制是否 SQL*Plus 显示存储过程，或 PL/SQL 块是否从 SQL*Plus 提示符下执行。这样的输出最常见地由 DBMS_OUTPUT.PUT_LINE 过程产生。

关键字

ON

显示输出。

OFF

禁止 DBMS_OUTPUT.PUT_LINE 输出。该选项是默认值。

SIZE

设置可以在服务器上缓存的输出字节数。*n*的默认值为2000，而且*n*必须在2000和 1 000 000 之间。

WRAPPED

指定输出被封装在当前行大小之内。可以在单词的中间发生行分隔。

WORD WRAPPED

指定在当前行大小之间进行输出的字封装。可以在单词的中间发生行分隔。

TRUNCATED

指定将截取超过行大小的任何行。

SHIFT

```
SHIFT[INOUT] {VIS[IBLE]|INV[ISIBLE]}
```

允许正确对齐显示切换（shift）字符的终止符。

关键字

VIS[IBLE]

显示切换字符。

INV[ISIBLE]

不显示切换字符。该选项是默认值。

注意：SET SHIFTINOUT 命令用于终端，如 IBM 3270 终端，与数据一起显示切换字符。

SHOWMODE

SHOW[MODE] ON | OFF

控制使用 SET 命令时 SQL*Plus 是否列出 SQL*Plus 系统变量的老的和新的设置。

关键字

ON

使 SQL*Plus 在变化发生时列出设置。(该选项将会产生与现在废止的 SHOWMODE BOTH 命令一样的结果)。

OFF

进行变化时禁止列出老的和新的设置值。该选项是默认值。

SQLBLANKLINES

SQLBL[ANKLINES] ON | OFF

控制 SQL*Plus 是否允许在 SQL 命令中嵌入空白行。

SQLCASE

SQLC[ASE] MIX[ED] | LO[WER] | UP[PER]

控制优先执行 SQL 语句和 PL/SQL 块的大小写转换(包括任何文本文字)。

关键字

MIX[ED]

指定不改变大小写。

LO[WER]

指定所有的字符被转换成小写字符。

UP[PER]

指定所有的字符被转换成大写字符。

SQLCONTINUE

SQLCO[NTINUE] *text*

在SQL*Plus命令继续超出一行时，设置SQL*Plus显示字符序列为提示符。默认文本是“>”。

SQLNUMBER

SQLN[UMBER] ON | OFF

控制录入多行SQL语句时SQL*Plus是否使用行号作为提示的一部分。

关键字

ON

该行号是提示的一部分。这是默认设置。

OFF

行号不是提示的一部分。

SQLPREFIX

SQLPRE[FIX] *c*

设置SQL*Plus前缀字符；*c*指定用作SQL*Plus前缀的字符。当大家录入SQL命令或PL/SQL块时，可以在单独行上录入SQL*Plus命令，采用该字符前缀，而且将在不干扰录入的SQL语句的情况下执行。SQL*Plus前缀字符必须采用非字母组成，而且默认为“#”。

SQLPROMPT

SQLP[ROMPT] {SQL> | *text*}

设置SQL*Plus命令提示符。

SQLTERMINATOR

SQLT[ERMINATOR] *c* | ON | OFF

设置用于结束和执行 SQL 命令的字符。

关键字

c

改变终止符值为该字符。

ON

改变终止符的值返回到默认的“;”并允许 SQL 终止符功能。

OFF

关闭 SQL 终止符功能。

SUFFIX

SUF[FIX] *text*

设置 SQL*Plus 在命令中指向命令文件的默认扩展文件名；*text* 指定指向 SQL 命令和操作系统支持文件扩展名时的扩展名。默认值是 *.SQL*。

注意：SUFFIX 不控制用作导出文件的扩展名，但用于连接 START、GET 等等。

TAB

TAB ON | OFF

控制 SQL*Plus 怎样在终端输出中规定 whitespace 的格式。

关键字

ON

指定插入大量 whitespace 到终端输出中时，SQL*Plus 使用 TAB 字符。

OFF

指定 SQL*Plus 使用空格作为输出中的空格的格式。

TERMOUT

TERM[OUT] ON | OFF

控制SQL*Plus是否显示从命令文件或终端屏幕上执行的命令产生的输出。

TIME

TI[ME] ON | OFF

控制是否将当前提示符下的一部分显示为时间。

注意： 显示的时间将从当前运行SQL*Plus的机器时钟上得到，不一定是被连接的服务器会话上的时钟。例如，如果在伦敦（设置为本地时间）的工作站上运行SQL*Plus，并连接到莫斯科的Oracle服务器上，则显示伦敦时间。

TIMING

TIMI[NG] ON | OFF

控制定时统计显示。

关键字

ON

指定定时信息（可用时的共用时间和CPU时间）将在每一条SQL语句完成以后显示出来。

OFF

指定不显示定时信息。该选项是默认值。

TRIMOUT

TRIM[OUT] ON | OFF

控制是否SQL*Plus允许在终端显示的每行末尾留有空白。

关键字

ON

使 SQL*Plus 从每一行的末尾删除空白。该选项改进了性能，尤其在以低速拨号连接访问 SQL*Plus 时。该选项是默认值。

OFF

使 SQL*Plus 显示可以包含在一行中的末尾空白。

TRIMSPOOL

TRIMS[POOL] ON | OFF

控制 SQL*Plus 是否允许每一行末尾的空白被写入到假脱机文件中。

关键字

ON

使 SQL*Plus 从每一行的末尾删除空白并写入假脱机文件中。

OFF

使 SQL*Plus 书写假脱机文件的输出时包括末尾空白。该选项是默认值。

UNDERLINE

UND[ERLINE] *c* | ON | OFF

设置用于给列标题加下划线的字符。

关键字

c

改变下划线字符值为该字符。

ON

改变下划线字符值为默认的“-”并允许下划线特性。

OFF

关闭下划线。

VERIFY

VER[IFY] ON | OFF

控制在SQL*Plus用值代替替代变量之前或之后SQL*Plus是否列出SQL或PL/SQL块被修改行的文本。

关键字

ON

指定显示该行包含的被修改的文本。这是默认值。

OFF

指定不显示包含被修改的文本的行。注意，该选项只影响显示；替代仍然会发生。

WRAP

WRA[P] ON | OFF

当前行太长时，控制SQL*Plus怎样显示数据输出。

关键字

ON

指定显示的行继续（包装）在随后的一行或多行中。这是默认选项。

OFF

指定如果一行的长度超过当前行宽时，则该行在显示中将被截取。

第七章

PL/SQL

SQL语言的功能、灵活性和用途是不可否定的。但是，因为Oracle及其开发变得更复杂，SQL内在的限制，尤其是其缺乏过程控制，已经变得很明显。SQL是第四代语言，这意味着编程者指定要做什么，而不是怎样做。其他语言（像C）称为第三代语言。对这些语言，编程者可以将控制保持在编程操作上。每一种方法都很有用，而且各有优缺点。

为了响应第三代类型过程化的能力（尤其在Oracle开发工具中），Oracle创建了PL/SQL语言。代表“SQL的过程化语言扩展”的PL/SQL首先出现在版本6中，该语言已经被更新，并在随后的每一个Oracle版本中都得到了改进。

PL/SQL语言看起来有点像ADA，ADA是美国国防部使用的第三代开发语言。PL/SQL也是基于SQL，而且允许使用标准SQL语句。事实上，PL/SQL可以用于书写像3GL这样的过程代码，也可以包含SQL语句及引用数据库对象。

PL/SQL实际上被用于当今可用的每一个Oracle工具中，而且给开发者提供了丰富的过程化和非过程化阵列的能力。PL/SQL代码可以被开发为脱离于SQL*Plus的“独立产品”，作为Oracle表格或报表的一部分，作为web应用程序的组件，甚至被Oracle数据库存储并执行。本章重点讨论PL/SQL的基础（注1）。

注1：全书均用PL/SQL编写，这里不再重复。要更为详细地了解PL/SQL，我强烈推荐参考Steven Feuerstein和Bill Pribyl编写的《Oracle PL/SQL Programming,Second Edition》（O'Reilly & Associates）。

PL/SQL 结构

PL/SQL 代码嵌入在块中，块是工作的逻辑单元。多个块可以被依次安排以便按顺序执行任务，也可以嵌套块，例如，PL/SQL 块可以包含 PL/SQL 代码的另一块。

PL/SQL 块由 4 个不同的部分组成：

块头

该可选部分定义了块名和类型（如过程、函数和匿名）。如果省略，可以认为该块是一个没有名字的匿名块。

声明部分

该可选部分定义（而且随意地初始化）块中使用的变量和游标。如果不使用变量或游标，则不需要该部分。

执行部分

该可选部分包含了由PL/SQL运行引擎执行的可执行PL/SQL（和SQL）语句。

异常部分

该部分包含了执行块时遇到的定制错误处理。

图 7-1 显示了 PL/SQL 块的结构。

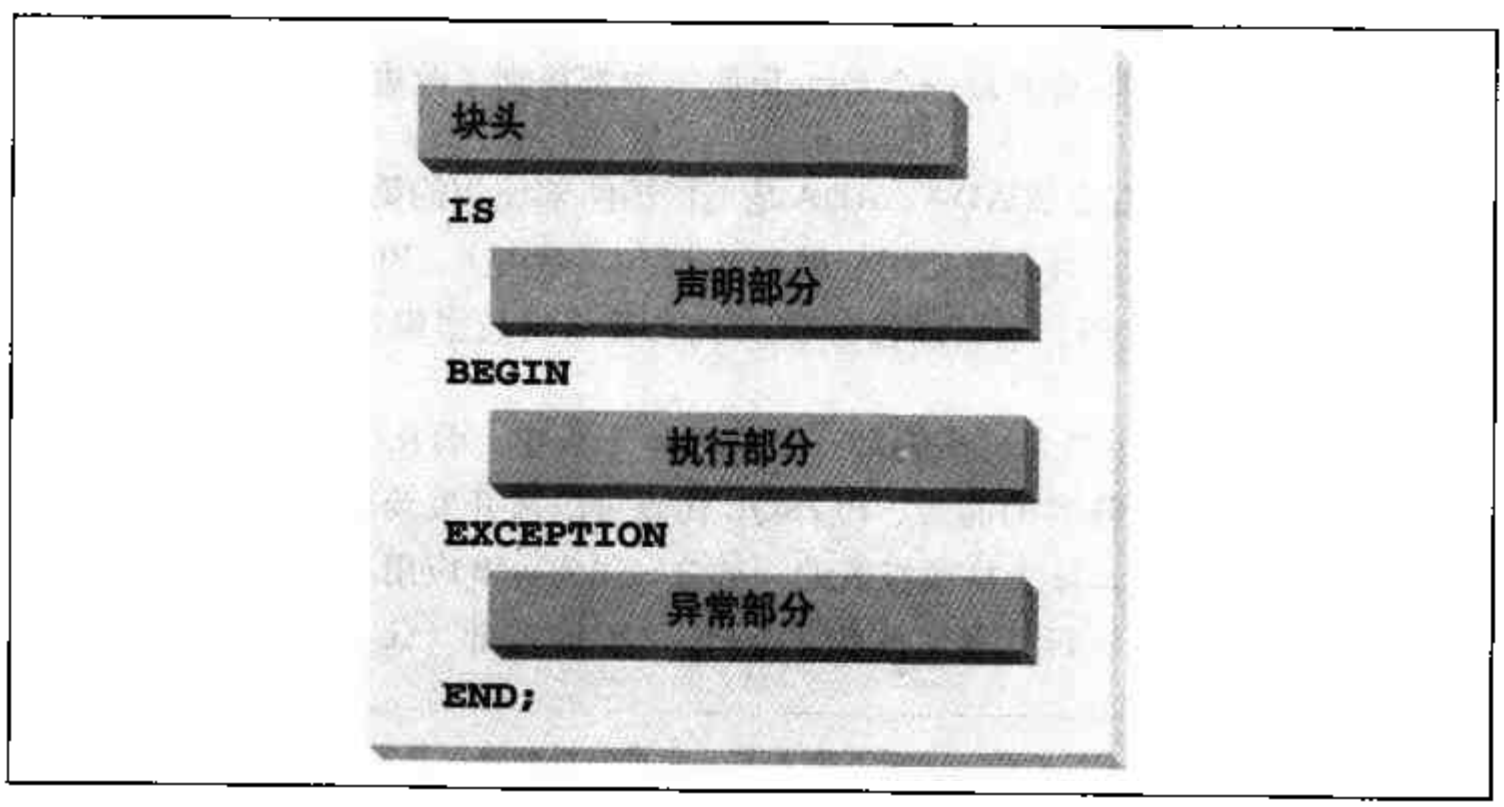


图 7-1：PL/SQL 块的结构

图 7-2 显示了给 90 天前雇佣的所有员工增加 10% 工资的简单 PL/SQL 块的代码。

```
PROCEDURE give_raise IS                    ← 块头

DECLARE
   cutoff_date   DATE;                     ← 声明部分

BEGIN
   cutoff_date := SYSDATE-90               ← 执行部分
   UPDATE emp
   SET sal = sal * 1.1
   WHERE hiredate < cutoff_date;

EXCEPTION                                  ← 异常部分
   WHEN NO_DATA_FOUND THEN NULL;
END;
```

图 7-2: PL/SQL 块示例

大家可以使用 PL/SQL 创建许多不同类型的模板结构，每一个都有特定的目的。

匿名块

可以执行一个或多个 PL/SQL 行为的未命名的 PL/SQL 块。匿名块最常用于强加给大集合的 PL/SQL 块上。匿名块不包含块头。

命名块

已经提供了名字的匿名块，否则就与匿名块相同。

函数

执行一些行为并返回单个值的命名 PL/SQL 块。函数可以接受一个或多个参数。

过程

执行一个或多个行为的命名 PL/SQL 块。参数列表可以用于传递值进和 / 或出过程。

包

PL/SQL 函数和 / 或过程的命名集合。包也可以包含变量。

块头

除了匿名块，所有的PL/SQL块都由一个块头（block header）开始。PL/SQL块头的常用语法如下：

```
{{PROCEDURE | FUNCTION} name IS |  <<name>> }
```

关键字

PROCEDURE

显示该块是PL/SQL过程。

FUNCTION

显示该块是PL/SQL函数。

name

指定分配给块的名字。如果用于创建命名块，则*name*出现在一对双尖括号中（<< >>），而且省略关键字IS。

声明部分

如果在PL/SQL块中使用了变量，则需要声明部分。声明部分还定义使用在块中的游标、类型、本地过程和函数。如果不需要声明变量或其他元素，那么就可以省略该部分。

注意：PL/SQL变量与列名不完全相同。例如，列名可以出现在SELECT语句或WHERE从句中，不是变量且不需要被声明。但是，SELECT...INTO...语句的目标是变量，而且必须被声明。

声明变量

可以在PL/SQL块中声明许多不同类型的变量。大家可以使用简单的、标量变量来保存数字、字符串和日期值。大家可以在PL/SQL块中声明允许操纵多个相关值的记录变量，甚至可以声明内存中的表和数组。

声明标量变量的语法显示如下:

variable_name type [CONSTANT] [NOT NULL] [:=*initial_value*]

声明了一个显式变量。

variable_name [*schema*.] *table_name.column_name*%TYPE

声明了一个被已有数据库列引用的变量。

variable_name pl/sql_variable%TYPE

声明了被以前定义的PL/SQL变量引用的变量。

关键字

variable_name

指定变量名。有效名可以最长为30个字符，而且必须用字母开始，可以包含字母A~Z、数字0~9、下划线、$和#等。变量名通常不区分大小写，但可以通过将它们放在双引号中使其区分大小写。除了放在引号中之外，不可以使用保留字作为变量名。

type

指定变量数据类型。表7-1列出了可用的数据类型。

CONSTANT

指定一个初始化值用于该变量，而且不可以改变该值。

NOT NULL

为该变量指定一个初始化值，而且该变量不可以设置为NULL。

initial_value

指定变量的初始化值。如果省略，则变量的初始值可以为NULL。注意，如果没有指定CONSTANT或NOT NULL，则必须指定初始值。

schema

定义数据库中的架构名。如果省略，则使用当前架构名。

table_name

指定数据库特定架构中的表名。

column_name

指定特定表中的列名。

pl/sql_variable

指定以前定义的 PL/SQL 变量名。

变量数据类型

表 7-1 摘要显示了有效 PL/SQL 数据类型。

表 7-1：PL/SQL 数据类型

数据类型	描述
数字型	
BINARY_INTEGER	整数值的两位补码的带符号的二进制表示
DEC	十进制数字；等于 NUMBER
DECIMAL	十进制数字；等于 NUMBER
DOUBLE PRECISION	双精度十进制数字；等于 NUMBER
FLOAT	浮点型十进制数字；等于 NUMBER
INT	整数数字；等于 NUMBER，但没有指定尾数
INTEGER	整数数字；等于 NUMBER，但没有指定尾数
NATURAL	等于 BINARY INTEGER，但限制值在 0~2 147 483 647 的范围之内
NUMBER	Oracle 内部代表的十进制或整数值，在精度和存储效率上进行了优化
NUMERIC	等于 NUMBER
PLS_INTEGER	存储有符号的整数，范围在 -2 147 483 647 ~ 2 147 483 647 之间。用于得到比 NUMBER 更快速的计算结果
POSITIVE	等于 BINARY INTEGER，但限制值的范围为 1~2 147 483 647 之间
REAL	实数；等于 NUMBER
SMALLINT	小整数；等于 NUMBER

表 7-1：PL/SQL 数据类型（续）

数据类型	描述
字符型	
CHAR	固定长度的字符串
CHARACTER	固定长度的字符串
LONG	PL/SQL 指定的字符串，本质上像 VARCHAR2，最大可以到 32 760 字节。注意，该长度比 LONG 数据库列要短
LONG RAW	与 LONG 相似，但不能在字符集之间转换
NCHAR	固定长度的 NLS 字符数据，最大长度为 32 767 字节
NVARCHAR2	可变长度的 NLS 字符数据，最大长度为 32 767 字节
RAW	与 CHAR 相似，但不能在字符集之间转换
ROWID	代表数据行的物理位置的二进制内部值。这只应该用在 Oracle7 和 Oracle7 兼容的版本中
UROWID	通用行 ID，代表数据行的物理、逻辑或非 Oracle 位置的二进制内部值。这只能用在 Oracle8 以后的版本
STRING	等于 VARCHAR2
VARCHAR	等于 VARCHAR2
VARCHAR2	可变长度字符串，最大长度为 32 767 字节。注意，该长度比 VARCHAR 数据列类型的长度要长，最大长度可以为 4000（在 Oracle7 中是 2000）
布尔型	
BOOLEAN	布尔型数据，值为 TRUE、FALSE 或 NULL
日期 / 时间（Date/Time）	
DATE	Oracle 内部日期格式
大对象（Large Object）	
BFILE	存储保留在操作系统文件中的大二进制对象的位置
BLOB	存储大二进制对象的位置
CLOB	存储大块单字符数据的位置
NCLOB	存储大块 NLS 字符数据的位置

注意：除了表7-1中列出的数据类型，PL/SQL允许变量具有被已有的数据库列或以前声明的PL/SQL变量引用所定义的类型，如前面的声明语法一样。

变量范围

变量的范围指整个PL/SQL程序中可以使用该变量的部分。PL/SQL变量可以在声明它们的声明部分使用，而且保留到整个块结束为止，包括任何嵌套块。图7-3演示了PL/SQL程序中组成两个块的两个变量。

在图7-3中，变量“aaa”可用于整个程序中，包括内部块。但是“bbb”在内部块中定义，所以只可以在该内部块中使用。它不可以用于外部块中。

```
DECLARE                          ← 外部块始于此处
   aaa     NUMBER;
BEGIN
   DECLARE                       ← 内部块始于此处
      bbb     NUMBER;
   BEGIN
   ...
   END;
...
END;
```

图7-3：PL/SQL变量范围

声明记录

在PL/SQL中，在想要将一定数量的相关变量作为单元操纵时，可以声明记录类型。例如，想将列empno、ename、sal和comm作为PL/SQL程序中的单个单元（或记录）。该PL/SQL类型与C中的结构相似。要使用记录，必须首先定义记录类型。然后基于该类型声明记录变量。

声明记录类型

```
TYPE type_name IS RECORD (
   variable_name type [NOT NULL] [:=initial_value]
  [, variable_name type [NOT NULL] [:=initial_value] ...])
```

声明记录类型。

关键字

type_name

指定记录类型名。

variable_name

指定 PL/SQL 变量名。

type

指定变量数据类型。要得到有效的 PL/SQL 数据类型的列表，参见表 7-1。

NOT NULL

指定变量的初始值，而且该变量不可以设置为 NULL。

initial_value

指定变量的初始值。如果省略，则该变量的初始值可以为 NULL。注意，如果既没有指定 CONSTANT 也没有指定 NOT NULL，就必须指定该初始值。

基于类型声明记录

record_name type_name

一旦声明了记录类型，就可以使用该语法声明该类型的一个或多个记录变量。

关键字

record_name

分配给记录类型的新实例名。

type_name

记录类型名。可以使用 TYPE 语句定义该类型，也可以是一个 %ROWTYPE。

示例

记录类型可以声明如下：

```
DECLARE
   TYPE Emp_Rec_Type IS RECORD  (
      Empno NUMBER(4) NOT NULL,
      Ename VARCHAR2(10),
```

```
        Hiredate DATE,
        Sal NUMBER(7,2),
        Comm NUMBER(7,2));
```

记录类型 Emp_Rec_Type 的实例可以声明如下:

```
old_emp Emp_Rec_Type;
new_emp Emp_Rec_Type;
```

注意: 通过使用 INTO 语句，可以使记录成为 SQL SELECT 语句的目标。

在记录中引用字段

record_name.variable_name

记录的独立字段可以使用这里显示的点符号（.）语法引用。

关键字

record_name

指定记录名。

variable_name

指定记录中的变量名。

示例

下面的语句把当前日期和时间放入 hiredate 字段中:

```
new_emp.hiredate := SYSDATE;
```

%ROWTYPE

record_name [*schema.*]*table_name*%ROWTYPE

通常，需要将记录声明为数据库表的相同结构。PL/SQL 提供了简单语法定义这样的记录。

关键字

record_name

指定记录名。

schema

指定数据库中的架构名。如果省略，则使用当前架构。

table_name

指定特定架构中的表名。

%ROWTYPE

指定创建一条记录，该记录的字段名和数据类型，与表中行的名称和数据类型匹配。

索引表

在需要一个变量多次出现时，可以定义索引表。这些表与C中的数组相似，但并不相同。索引表类型必须在可以声明基于该类型的索引表之前声明。

索引表类型

```
TYPE tabletype_name IS TABLE OF
   {type | variable%TYPE |[schema.]table_name%ROWTYPE}
   INDEX BY BINARY_INTEGER;
```

声明索引表的类型。

关键字

tabletype_name

指定索引表类型名。

type

指定有效的PL/SQL数据类型。结果表中的每一个元素代表指定的类型出现一次。

variable

指定以前定义的PL/SQL变量。*variable* %TYPE语法允许在已有变量声明的基础上创建一张表。

schema

包含 *table_name* 的架构名。如果省略，则使用当前架构。

table_name

定义数据库表。*table_name*%ROWTYPE语法使大家可以很容易地创建一张索引表，在该索引表中每一个元素代表与指定表的结构匹配的记录出现一次。

INDEX BY BINARY_INTEGER

索引表类型声明需要的字符串。

表类型声明的结果是具有两列的内存表：

KEY（总是 BINARY_INTEGER）
VALUE（在声明语句中定义的类型）

声明索引表

```
table_name tabletype_name;
```

一旦已经定义了索引表类型，就可以声明并使用这里列出的标准 PL/SQL 语法来使用该类型的变量，以便声明已有类型基础上的索引表。

```
table_name(key)
```

声明一张表以后，就可以使用该语法引用单个元素。

关键字

table_name

指定表名。

key

指定表的特定行的索引。该索引必须是整数值。

示例

该代码声明了可以用于保存员工姓名的 VARCHAR2 的表：

```
DECLARE
   TYPE ename_table_type IS TABLE OF varchar2(10)
      INDEX BY BINARY_INTEGER;

   ename_table ename_table_type;
```

一旦声明了ename_table，则下面的语句就分配一个“KING”值给表中的一个元素：

```
ename_table(10) := 'KING';
```

注意：索引表可以稀疏分布。大家可以让表中的元素1和元素1000存在，但元素2到元素999不存在。如果大家引用索引表中不存在的元素，则会产生一个NO_DATA_FOUND的异常信息。在大家的代码中，大家应该要么书写异常处理来捕捉该异常信息，或者在引用之前使用EXISTS方法证实元素存在。后面的部分描述了这些语言元素。

索引表方法

PL/SQL索引表类型具有可以在PL/SQL语句中引用的相关方法。表7-2列出了可用的方法，并带有一定类型的返回值（如果有的话）。

表 7-2：PL/SQL 索引表中可用的方法

方法	返回值	描述
COUNT	NUMBER	返回表中的行数
DELETE		从表中删除所有行
DELETE(*x*)		从表中删除具有键 *x* 的行
DELETE(*x*,*y*)		删除表中有 *x*~*y* 之间的键的所有行
EXISTS(*x*)	BOOLEAN	如果表中存在一行具有键 *x* 就返回 TRUE；否则，返回 FALSE
FIRST	BINARY_INTEGER	返回表中具有最小键的行的键
LAST	BINARY_INTEGER	返回表中具有最大键的行的键
NEXT(*x*)	BINARY_INTEGER	返回表中比提供的键要大的最小的行键
PRIOR(*x*)	BINARY_INTEGER	返回表中比提供的键要小的最大的行键

注意：DELETE方法只可用在Oracle8和更高版本中。以前的版本中没有机制可以用于从索引表中删除行。

使用下列的语法引用具体的方法：

```
table_name.method[(x[,y...])]
```

关键字

table_name

指定表名。

method

指定方法，如表7-2所示。

(x[,y...])

如果需要，则为方法指定一个或多个参数（参见表7-2）。

示例

该例子检查存在具有键10的表元素，而且如果它在表中存在，则将其改成初始化大小写（initial capital）。如果不存在，则插入“Missing”：

```
IF ename_table.EXISTS(10) THEN
   ename_table(10) := INITCAPS(ename_table(10));
ELSE
   ename_table(10) := 'Missing';
END IF;
```

在该例子中，FIRST、NEXT和LAST方法用于反复调用表中的所有已有行：

```
t_key := ename_table.FIRST;

LOOP
   ename_table(t_key) := INITCAPS(ename_table(t_key));
   EXIT WHEN t_key = ename_table.LAST;

   t_key := ename_table.NEXT(t_key);
END LOOP;
```

在循环中，表的每个元素被转换成初始化大小写。在最高层上检查该键，当两个值匹配时退出循环。否则，使用NEXT方法得到下一个键。注意，为了作为起点使用，

当前键被传递到方法中。NEXT 启动当前键指定的元素，而且检索下一个非空的表元素。

可变数组

可变数组是具有特定数量的集合类型。与索引表不同的是，在声明数组时，数组中元素的最大数量是固定的。可变数组也叫做VARRAY，而且它们可以用于保存数据库表中的VARRAY列的内容。与其他集合类型一样，大家必须首先声明数组类型，然后才可以声明该类型的变量。

注意：可变数组只可用于 Oracle8*i* 企业版，不能用在标准版中。

TYPE (VARRAY)

```
TYPE vartype_name IS {VARRAY | VARYING ARRAY} (maxsize)
OF data_type [NOT NULL]
```

声明 VARRAY 数组类型。

关键字

vartype_name

希望给类型提供的名字。

maxsize

指定要可变数组保留的实体的最大数量。

data_type

指定数组元素的数据类型。

一旦声明了可变数组的类型，就可以使用下列语法声明变量的类型：

```
varray_name vartype_name;
```

在可以给数组元素分配值之前，大家必须通过使用其构造器函数对数组进行初始化，其名字与基础类型名相同。

示例

下面的例子显示了被创建的两个VARRAY。一个使用空数组初始化，没有向其构造器方法中传递参数。另一个初始化为3个元素的数组。

```
DECLARE
   TYPE num_array IS VARRAY (100) OF NUMBER;

   num1 num_array;
   num2 num_array;
BEGIN
   -- 调用构造器方法
   num1 := num_array(  );
   num2 := num_array(1,2,3);

   -- 为了增加一个值，扩展每一个数组
   num1.extend;
   num1(1) := 1;

   num2.extend;
   num2(4) := 4;
END;
```

注意：不能将VARRAY扩展为超出它们的最大声明大小。

VARRAY数组方法

表7-3显示了可以与PL/SQL可变数组一同使用的方法。

表7-3：可以与PL/SQL可变数组一同使用的方法

方法	返回值	描述
COUNT	INTEGER	返回表中的行数
DELETE		从表中删除所有行
DELETE(*x*)		从表中删除具有键*x*的行
DELETE(*x*,*y*)		删除表中*x*~*y*之间包含的所有行
EXISTS(*x*)	BOOLEAN	如果*x*代表了表中有效的条目，就返回TRUE；否则，返回FALSE
FIRST	INTEGER	返回表中使用的最小索引值

表 7-3: 可以与 PL/SQL 可变数组一同使用的方法（续）

方法	返回值	描述
LAST	INTEGER	返回表中使用的最大索引值
NEXT(*x*)	INTEGER	返回作为输入参数指定的索引的下一个有效的索引
PRIOR(*x*)	INTEGER	返回作为输入参数的索引的前一个有效的索引
TRIM	无	删除具有最大索引值的条目
TRIM(*x*)	无	从表末尾开始删除指定数量的条目。这些条目将是最高的 *x* 个条目
EXTEND	无	在嵌套表中增加一个条目
EXTEND(*x*)	无	在嵌套表中增加 *x* 个条目
EXTEND(*x*,*y*)	无	在嵌套表中增加 *x* 个条目，而且让它们全部是索引 *y* 条目的完全备份
LIMIT	INTEGER	返回数组的最大声明大小

声明游标

PL/SQL 游标允许 SELECT 语句返回被执行的多行记录，然后可以使用循环反复调用那些行的每一条。游标的SQL语句不需要每次打开游标都被分析。要使用大家声明的游标，就必须使用 OPEN、FETCH 和 CLOSE 语句。这些语句在本章的“执行部分”一节中描述。

CURSOR

```
CURSOR cursor_name IS
select_statement
[FOR UPDATE [OF column[,column...]] [NOWAIT]];
```

声明了一个游标。

关键字

CURSOR

显示声明 PL/SQL 游标需要的关键字。

cursor_name

用于声明游标的有效 PL/SQL 名。

select_statement

有效的SQL SELECT语句。该语句可以在WHERE从句中引用PL/SQL变量，只要这些变量已经被定义而且在块中可用就可以了。

FOR UPDATE

指定选择一行作为使用该游标的FETCH操作的结果，在该行上需要独占锁，所以可以在随后更新它。如果该行被加锁，则PL/SQL将等待释放该锁（除非使用了NOWAIT关键字）。UPDATE或DELETE语句的WHERE CURRENT OF可以用于更新或删除游标提取操作返回的最近的行。

OF column[, column...]

指定FETCH之后将要更新的一列或多列。

NOWAIT

指定如果提取的行被另一个进程加锁，则Oracle将不等待释放锁而将直接返回一个错误："ORA-00054: resource busy and acquire with NOWAIT specified."

注意：与游标相关的SELECT语句可以不包括在INTO从句中。相反，使用FETCH语句将数据提取到PL/SQL的变量中。参见"执行游标"一节。

示例

下面的游标声明包括了从scott.emp表中提取薪水大于PL/SQL的变量t_sal当前值的所有行，必须已经声明了t_sal:

```
CURSOR emp_cursor IS
   SELECT sal, comm
   FROM scott.emp
   WHERE sal > t_sal;
```

声明异常

PL/SQL使用异常处理执行PL/SQL程序中可以发生的错误和其他条件。Oracle已经预定义了普通错误的异常处理。大家也可以自己定义。

EXCEPTION

exception_name EXCEPTION

声明自己的异常处理。

关键字

exception_name

异常处理的有效 PL/SQL 名。

EXCEPTION

需要使用该关键字显示这是为用户定义的异常而声明的。

注意：如果定义了命名的异常而没有使用 PRAGMA EXCEPTION_INIT 语句将其指定在特定的错误号上（下面将看到），那么惟一可以产生该异常信息的方法是在代码中手工提供。

PRAGMA EXCEPTION_INIT

PRAGMA EXCEPTION_INIT (*exception_name*, *ora_number*);

将异常处理与指定的 Oracle 错误号关联。

关键字

PRAGMA EXCEPTION_INIT

调用编译器指向与特定 Oracle 错误号相关的命名异常。

exception_name

为异常定义的有效 PL/SQL 名。

ora_number

想要连接到命名的异常上的 Oracle 错误号。

注意：Oracle 的错误号通常是负数。如果想跟踪错误 ORA-02292，可以使用 -2292 作为 *ora_number* 值。

在声明异常之后，大家可以使异常与特定的Oracle错误号相关，这样当特定的错误发生以后，Oracle将自动产生异常信息。

注意：如果不使用PRAGMA EXCEPTION_INIT而要声明异常来连接特定的错误号，则产生该异常的惟一方式就是在代码中手工提出。

执行部分

PL/SQL块的执行部分是惟一必需的部分。它包含PL/SQL代码，实际执行一些行为；异常部分也执行一些行为，但只在执行部分的行为被触发之后才发生。执行部分的常用语法是：

```
BEGIN
   {assignment_statement     |
     proc_call     |
     control_statement     |
     PL/SQL_block     |
     SQL
   };
   [{assignment_statemanet     |
     proc_call     |
     control_statement     |
     PL/SQL_block     |
     SQL
    }; . . . ]
END;
```

关键字

BEGIN

标志执行开始的必要关键字。

assignment_statement

任何有效的PL/SQL赋值语句。该语句通常由变量和表达式组成，采用:=操作符分离。

proc_call

调用PL/SQL过程或函数。

control_statement

任何有效的 PL/SQL 控制语句。这些语句在本章后面描述。

PL/SQL_block

任何完整的 PL/SQL 块。包含执行部分的 PL/SQL 块叫做嵌套块。

SQL

任何有效的 SQL DML 语句，如 SELECT、INSERT、UPDATE、DELETE、ROLLBACK 或 COMMIT。

END

标志执行部分结束的必要关键字。

赋值语句

赋值语句给 PL/SQL 变量提供值。PL/SQL 赋值语句的常用语法如下:

```
variable := expression;
```

关键字

variable

任何以前声明的 PL/SQL 变量。

:=

PL/SQL 赋值符号。注意 := 和在许多其他语言中通常使用 = 之间的差别。

expression

任何有效的 PL/SQL 表达式。

表达式

PL/SQL表达式由一个或多个变量选择性地组合一个或多个运算符组成。在PL/SQL中，表达式可以出现在下列位置:

- 赋值语句的右边
- 作为控制语句的一部分

- 作为SQL语句的一部分

PL/SQL运算符可以使用在表达式中，其相关优先权（1的优先权最高）如表7-4所示。

表7-4：PL/SQL运算符

运算符	描述	类型	优先权
||	连接两个字符串	字符	4
+	相加	数字	4
+	正标识（当用做正号时）	数字	2
-	减	数字	4
-	负标识（当用做负号时）	数字	2
*	乘	数字	3
**	求幂	数字	1
/	除	数字	3
=	等于	数字	5
!=	不等于	布尔	5
<>	不等于	布尔	5
~=	不等于	布尔	5
<	小于	布尔	5
>	大于	布尔	5
<=	小于等于	布尔	5
>=	大于等于	布尔	5
IS NULL	测试空值	布尔	5
LIKE	使用掩码测试等于字符串	布尔	5
BETWEEN	测试范围中包括的内容	布尔	5
IN	测试列表中包含的内容	布尔	5
AN	逻辑“与”求值	布尔	6
OR	逻辑“或”求值	布尔	7
NOT	逻辑“非”求值	布尔	1

控制语句

PL/SQL控制语句用于控制PL/SQL块的执行行为。已有的控制语句给PL/SQL提供了对过程化语言或第三代语言的控制能力。这些能力与SQL截然不同。

IF-THEN-ELSE

```
IF expression THEN
    statement;[statement; ...]
[ELSIF expression THEN
   statement;[statement; ...]
{ELSIF expression THEN
   statement;[statement; ...] ...]
}
[ELSE
   statement;[statement; ...]]
END IF;
```

在布尔表达式的逻辑求值基础上执行一条或多条语句。

关键字

IF

指示条件处理开始需要的关键字。

expression

任何有效的返回布尔结果的PL/SQL表达式。

THEN

指示如果表达式求值结果为TRUE要执行后面语句的必要关键字。

statement

任何有效的PL/SQL赋值语句、SQL语句、过程调用或函数调用。

ELSIF

可选关键字，指示如果基本表达式为FALSE时需要对可选表达式求值。如果求得ELSIF表达式为TRUE，则执行ELSIF后面的语句。可以根据需要指定可选择的条件（ELSIF）。

ELSE

指示没有一个IF或ELSIF表达式求得的值为TRUE时，将优先执行该语句的可选关键字。

END IF

指示IF语句结束的必要关键字。每一个IF必须采用惟一一个END IF关键字结束。

注意：只有一组语句将在IF-THEN-ELSE中执行。一旦表达式的值为TRUE，则执行相关的语句并控制转移到END IF关键字后面的语句上。

示例

在下面的例子中，每个周期根据初始值调整员工的薪水和奖金。如果薪水少于$1000，则增长10%（薪水 * 1.10），但不增加奖金。在$1001和$1500之间的薪水将增长8%而且得到$500奖金。在$1501和$2000之间的薪水，将增长5%，而且得到$750奖金。其他人（薪水超过$2000）增长2%并得到$1000元奖金：

```
IF salary < 1000 THEN
   salary := salary * 1.10;
   bonus := 0;
ELSIF salary BETWEEN 1001 AND 1500 THEN
   salary := salary * 1.08;
   bonus := 500;
ELSIF salary BETWEEN 1501 AND 2000 THEN
   salary := salary * 1.05;
   bonus := 750;
ELSE
   salary := salary * 1.02;
   bonus := 1000;
END IF;
```

LOOP

```
[<<label>>]
LOOP
[EXIT;]
[EXIT WHEN condition;]
statement;[statement;...]
END LOOP [<<label>>];
```

重复执行一条或多条语句，当条件满足或者显式执行了一条EXIT命令时，终止执行。在LOOP和END LOOP之间的每一条语句按顺序执行。如果没有碰到EXIT命令，则返回控制到循环的第一条语句中。

关键字

<<label>>

指定循环的可选标签。循环采用在LOOP语句前使用*<<label>>*提供标签，对应的END LOOP语句然后可以选择性地引用相同的标签。

LOOP

指示重复执行随后语句的必要关键字，直到到达END LOOP关键字为止。

EXIT

可选关键字，使控制被传递到END LOOP关键字后面的下一条语句中。EXIT实际是通常出现在IF语句嵌套循环后的可执行语句。该关键字可以出现在LOOP和END LOOP关键字之间的任何语句中，而且可以多次出现。注意，EXIT不限制LOOP语法的组件，但为清楚起见包括在这里，因为它通常用在LOOP编程中。

EXIT WHEN

选择性地指定条件，当该条件求值为TRUE时，将控制传递到END LOOP关键字的下一条语句中。该关键字可以出现在LOOP和END LOOP关键字之间的任何语句中，而且可以多次出现。指定的退出条件在碰到EXIT WHEN语句时进行求值。注意，EXIT WHEN不限制LOOP语法的组件，但在这里包括是为了简明的缘故，因为它通常用在LOOP编程中。

statement

任何有效的PL/SQL赋值语句、SQL语句、过程调用或函数调用。

END LOOP

显示结束LOOP结构的必要关键字。

警告：在一个循环结构中没有EXIT或EXIT WHEN关键字是合法的。但是，这样的循环将是一个死循环，大概大家并不想这样做！

示例

在这个例子中，我们使用了一张PL/SQL表，循环一直进行，直到碰到两个条件之一发生为止：如果表中有一行为NULL，则立即跳出循环。当循环计数器达到100时，也结束循环。

```
loop_counter := 0;
LOOP
   EXIT WHEN loop_counter = 100;
   loop_counter := loop_counter+1;
   IF salary_table(loop_counter) IS NULL THEN EXIT;
   salary_table(loop_counter) := salary_table(loop_counter) * 1.1;
END LOOP;
```

WHILE 循环

```
[<<label>>]
WHILE condition LOOP
statement;[statement;-]
END LOOP [<<label>>];
```

重复执行一条或多条语句，直到条件不满足为止。当最后一条语句被执行时，将控制返回到第一条语句中继续执行。该循环反复执行，直到不满足条件值为止。

关键字

<<label>>

指定循环的可选标签。循环在WHILE语句前使用*<<label>>*提供标签，而且对应END LOOP语句，然后可以选择性地引用相同的标签。

WHILE ...LOOP

指定这是一个WHILE循环，只要条件满足就一直执行。当条件不满足时，该关键字将把控制转移到END LOOP关键字紧随的下一条语句中。在执行每一个循环体之前对条件进行测试。

statement

任何有效的PL/SQL赋值语句、SQL语句、过程调用或函数调用。

END LOOP

显示结束LOOP结构的必要关键字。

注意：该条件在每一个重复循环之前对条件求值。如果条件为TRUE，则循环中的所有语句继续执行。如果条件为FALSE，则将控制传递到END LOOP关键字后面的一条语句上。

示例

该例子与前面显示的LOOP命令相似，而且演示了相同结果怎样使用不同方法来完成。这里，重复执行两条语句（一条增加循环计数器 loop_counter，而另一条更新薪水表中的对应元素），直到 loop_counter 达到 100。此时，它不再少于 100，而且循环终止：

```
loop_counter := 0;
WHILE loop_counter < 100 LOOP
   loop_counter := loop_counter+1;
   salary(loop_counter) := salary(loop_counter) * 1.1;
END LOOP;
```

FOR 循环

```
[<<label>>]
FOR counter IN [REVERSE] start .. end LOOP
statement; [statement;-]
END LOOP [<<label>>];
```

按指定次数执行一系列的语句。

关键字

<<label>>

指定循环的可选标签。循环在 FOR 语句前使用 *<<label>>* 提供标签，而且对应 END LOOP 语句，然后可以选择性地引用相同的标签。

counter

指定用于保存循环重复次数的变量名。在每一次重复循环后，计数器（counter）增长 1，除非指定了 REVERSE，在这种情况下计数器减 1。该变量自动定义为 BINARY_INTEGER，而且不需要在块中定义。如果定义相同名的变量而且可以在块中使用，则在 FOR 循环激活时临时不考虑其定义，除非使用了 *blockname.variable* 表示法进行显式引用。

REVERSE

指定计数器在执行 FOR 循环时将从末尾（end）减到开始（start）。

start

指定计数器的开始值，除非指定了 REVERSE。如果指定 REVERSE，则 *start* 代表了计数器的结尾值。

end

指定计数器的最后值；也就是，当计数器达到该值时，终止FOR循环而且将控制立即传递到END LOOP循环后面的可执行语句上。如果指定REVERSE，则*end*指定了计数器的初始值，然后逐渐减1，直到达到*start*为止。

statement

任何有效的PL/SQL赋值语句、SQL语句、过程调用或函数调用。

END LOOP

显示结束LOOP结构的必要关键字。

示例

该例子执行了WHILE循环中显示的例子的相同操作，但使用了FOR循环。这里薪水表的前99个元素被更新：

```
FOR loop_counter IN 1..99 LOOP
   salary(loop_counter) := salary(loop_counter) * 1.1;
END LOOP;
```

GOTO

```
GOTO label;
```

立即转移控制到后面的*label*标识的可执行语句上。使用GOTO语句时，必须遵循下面的规则：

- 不能从块外跳转到块中。
- 不能跳转到循环的中间。
- 不能跳转到IF语句中。
- 不能跳转出异常处理器并返回到代码中。

示例

在这个例子中，当薪水超过1000时，就将控制传递到标签no_raise上，而且指定下一条语句，给变量up_date分配当前日期：

```
IF salary > 1000 THEN
   GOTO no_raise;
END IF;
Salary := salary * 1.1;

<<no_raise>>
up_date := SYSDATE;
```

NULL

```
NULL;
```

NULL语句不执行任何操作。在开发新代码或代替不想再执行的IF语句的深层嵌套部分的内容时，可以使用它作占位符。

示例

下面的例子中，NULL已经被用作占位符，因为以后还要书写这一部分代码。

```
IF emp_sal < 0 THEN
   --Write this later when the specs are finished
   NULL;
ELSE
   print_paycheck(emp_no, emp_sal);
END IF;
```

SQL语句

PL/SQL程序中使用SQL的能力是PL/SQL的主要优点之一。通过在SQL代码中增加使用变量的能力，PL/SQL成为了可以充分利用复杂的Oracle数据库的强有力的语言。

但是，在PL/SQL中使用SQL有一些限制。最值得注意是，在PL/SQL中只允许有数据操纵语言和控制语句。在PL/SQL块中允许下面的SQL语句:

SELECT
INSERT
UPDATE
DELETE
COMMIT

ROLLBACK
SET CONSTRAINTS
SET ROLE
SET TRANSACTION
SAVEPOINT
LOCK TABLE

PL/SQL中不允许数据定义语言，包括会话和系统控制语句。EXPLAIN PLAN语句也不被允许。

注意： 有一种方法可以从PL/SQL中执行DDL语句，就是使用DBMS_SQL嵌入包。使用这个包（以及其他的嵌入包）超出了本书的范围。要得到有关使用DBMS_SQL的更多信息，可以参考Steven Feuerstein, Charles Dy和John Beresniewicz编写的《Oracle Built-in Packages》（O'Reilly & Associates）一书。

PL/SQL中使用SQL与“标准”SQL不同主要体现在以下几个方面：

- INTO关键字允许从数据库表中读数据而且放置到PL/SQL变量或记录中。
- PL/SQL变量可以用在允许常量或表达式的任何地方，包括WHERE从句中。所以，尽管书写代码时必须知道表名和列名，但不需要知道列或其他表达式的值。
- 在PL/SQL中执行SQL语句不将数据返回到屏幕或其他输出设备中。

下面提供了被允许的SQL语句的特定语法，只有两点异常。COMMIT和ROLLBACK语句与标准SQL语句相同，所以不在这里重复该语法。

SELECT

```
SELECT select_list
INTO {pl/sql_record | variable [, variable...]}
FROM table_list
[WHERE where_clause];
```

从Oracle表（或视图）中选择数据，并将PL/SQL记录中的结果放置到一个或多个PL/SQL变量中。

关键字

select_list

指定语句提取的列的清单。详细信息参见第三章。

pl/sql_record

指定PL/SQL记录名接收SELECT语句返回的数据。记录必须包含与选择清单中列出数量相同的变量，而且它们的类型必须兼容。如果记录中字段的数据类型与选择清单中的对应列不同，则这两个类型必须兼容，能够在Oracle中隐式地将一种形式转换另一种形式。

table_list

指定一个或多个表名，这些表将被选择数据。

variable

指定以前定义的PL/SQL变量名。必须与选择列的清单中变量数量相同，而且每一个必须具有相同（或兼容）的类型。

where_clause

指定这个SELECT的WHERE条件。WHERE从句中的条件可以包括以前定义的PL/SQL变量。

注意：该语法描述适用于SELECT语句的基本形式。记住，还可以采用其他从句。这些从句包括GROUP BY和ORDER BY从句。要得到SELECT语句的完整语法，参见第三章，包含了所有从句的描述。

示例

这个例子从emp表中选择KING的行，并将hiredate、sal和comm值放到相应的PL/SQL变量中：

```
DECLARE
   t_hiredate DATE;
   t_salary NUMBER;
   t_commission NUMBER;
BEGIN
   SELECT hiredate, sal, comm
   INTO t_hiredate, t_salary, t_commission
   FROM scott.emp
```

```
    WHERE ename = 'KING';
END;
```

INSERT

```
INSERT INTO [schema.]table (column[,column-])
VALUES ({expression[,expression...]  |  select_statement});
```

插入新数据行到特定的表中，并为每一个特定的列提供值。另外，可以从查询中提取结果，并将它们插入到特定表中。

关键字

schema

包含表的架构名。如果省略，则使用当前架构。

table

指定将插入新行的表名。

column

指定 VALUES 从句中接收数据的表中的列名。

expression

指定有效的 PL/SQL 表达式，可以包含以前定义的 PL/SQL 变量或记录类型。

select_statement

指定被插入的数据是这个 SELECT 语句的结果。SELECT 语句必须为 INSERT INTO 从句中列出的每一列返回一列，而且类型必须兼容。

示例

下面的 PL/SQL 块中的代码插入新行到 emp 表中：

```
BEGIN
   INSERT INTO emp (empno, ename, sal)
   VALUES (1111, 'Gennick', 2500);
END;
```

UPDATE

```
UPDATE [schema.]table
```

```
SET {column = expression[,column = expression...] |
         column[,column...] = (select_statement)
        }
WHERE {where_clause | CURRENT OF cursor_name};
```

更新表中的一行或多行。

关键字

schema

包含表的架构名。如果省略，则使用当前架构。

table

指定被更新的行的表名。

column

指定被更新的表中的列名。

expression

任何有效的 PL/SQL 表达式。

select_statement

任何有效的SELECT语句，而且必须返回数据元素的合适数字和类型。在SET从句中使用的 SELECT 语句必须放在括号中。

where_clause

为该更新指定WHERE条件。WHERE从句中的表达式可以包括以前定义的PL/SQL 变量。

CURRENT OF

指定应用该更新到最近提取的并被命名的游标加锁的行上。

cursor_name

指定用于从被更新表中提取数据的 PL/SQL 游标名。

示例

下面的例子更新为员工编号 7788 工作的所有员工记录，并给他们双倍加薪：

```
BEGIN
```

```
    UPDATE emp
       SET sal = sal * 2
    WHERE mgr = 7788;
END;
```

下面采用一个更复杂的UPDATE语句，设置每一个员工的薪水是他们主管薪水的90%。没有主管的员工不作调整：

```
BEGIN
   UPDATE emp e
      SET sal = (SELECT sal * 0.90
                 FROM emp m
                 WHERE e.mgr = m.empno)
   WHERE mgr IS NOT NULL;
END;
```

DELETE

```
DELETE FROM [schema.]table
WHERE {where_clause | CURRENT OF cursor_name};
```

删除表中的一行或多行。

关键字

schema

包含表的架构名。如果省略，则使用当前架构。

table

指定被删除行的表名。

where_clause

指定该删除的WHERE条件。WHERE从句中的表达式可以包括以前定义的PL/SQL变量。

CURRENT OF

指定应用该删除到最近提取的并被命名游标加锁的行上。

Cursor_name

指定用于从指定表中提取数据的PL/SQL游标名。

示例

下面的例子删除了薪水太高的主管:

```
BEGIN
   DELETE FROM emp m
   WHERE empno IN (SELECT mgr FROM emp)
     AND sal * 0.90 > (SELECT MAX(sal)
                         FROM emp e
                        WHERE e.mgr = m.empno);
END;
```

在这种情况下,“薪水太高”定义为乘以90%后仍比下级员工最高薪水还要高的薪水。那些主管是原来的主管。

执行游标

PL/SQL游标提供了从Oracle数据库中访问多行数据,并在每一行上执行程序的完全控制的方法。使用PL/SQL时,大家必须遵循下面的4个步骤:

1. 定义游标(参见本章前面的“声明游标”一节)。
2. 打开游标。
3. 执行FETCH语句提取数据。
4. 关闭游标。

下面的部分显示了游标执行时需要的PL/SQL语句的常用语法。

OPEN

```
OPEN cursor_name;
```

为了在当前块中使用,打开PL/SQL游标。该游标不能已经处于打开状态。

FETCH

```
FETCH cursor_name INTO {variable_list | record};
```

从游标结果集中读取一行数据到指定的PL/SQL变量中。每一个成功的FETCH将从结果集中读取一行新数据,直到不再有行数据要读为止。

关键字

FETCH

指定从SQL语句相关的游标名中返回一行。

cursor_name

指定以前已经定义并打开的游标名。

variable_list

以前定义的一个或多个PL/SQL变量，将放置与游标相关的SELECT语句返回的列值。

record

以前定义的PL/SQL记录，将放置与游标相关的SELECT语句返回的列值。记录中的字段必须与该SQL语句选择列表中的字段匹配。

CLOSE

```
CLOSE cursor_name;
```

关闭游标并释放相关资源。该游标必须是以前定义并打开的。

游标FOR循环（Cursor FOR Loops）

```
FOR {variable_list | record}
IN {cursor_name | (select_statement)} LOOP
   [pl/sql_statement; ...]
END LOOP;
```

虽然通常有必要显式打开游标并从中提取数据，然后关闭游标，PL/SQL提供了一种快捷方式执行该处理。这种快捷方式叫做游标FOR循环，当大家想简单循环被SQL的SELECT语句下的游标返回的所有行时非常有用。

关键字

variable_list

以前定义的一个或多个变量，将放置与游标名相关的SELECT语句返回的列值。

record

以前定义的PL/SQL记录，将放置与游标相关的SELECT语句返回的列值。

cursor_name

以前定义的游标名。游标在执行语句时隐式打开。

select_statement

指定游标名的选择语句，大家可以提供SELECT语句作为FOR循环语法的一部分。如果这样做了，就必须将SELECT语句封装在括号中。

END LOOP

显式结束游标FOR循环。当所有的数据被提取以后，循环结束并显式关闭游标。

注意：当执行游标的FOR循环时，相关的游标被打开并提取数据。在提取每一行之后，在FOR...IN语句之间的PL/SQL语句和END LOOP语句被执行。当所有行都被提取完毕时，游标关闭并继续执行END LOOP下面的语句。

示例

下面的例子显示了游标FOR循环用户重复名为emp_cursor的游标返回的记录：

```
BEGIN
  FOR v_emp_data IN emp_cursor LOOP /*循环开始并打开游标*/
     -- 隐式提取一行，所以现在对其进行处理
     bonus := v_emp_data.sal * 1.12;      /*分配12%的奖金*/
     INSERT INTO bonus (empno, bonus)
     VALUES (v_emp_data.empno, bonus);

     -- 现在循环继续检查其他行，以在可能时取得另一行
  END LOOP;
  COMMIT;
                                          /*继续其他处理*/
END;
```

第二个例子完成第一个例子中相同的工作，但不同的是被执行的SELECT语句是FOR循环的一部分：

```
BEGIN
  FOR v_emp_data IN (SELECT * FROM emp) LOOP
     --隐式提取一行，所以现在对其进行处理
     bonus := v_emp_data.sal * 1.12;
     INSERT INTO bonus (empno, bonus)
```

```
        VALUES (v_emp_data.empno, bonus);

        --现在循环继续检查其他行，以在可能时取得另一行
    END LOOP;
    COMMIT;

END;
```

在这两个例子中使用的v_emp_data记录定义必须与每个游标下的SQL语句返回的列的数量和类型相匹配。在第二个例子中，因为使用了SELECT *，可以声明记录为v_emp_data emp%ROWTYPE。

引用游标属性

一旦定义游标，就可以使用4种属性得到游标的信息。这些属性，包含它们可能返回的值，列出在表7-5中。

表7-5：PL/SQL游标属性

属性	描述	值	含义
%FOUND	显示是否已经返回数据行	TRUE	以前的FETCH返回了数据行
		FALSE	以前的FETCH没有返回数据行；所有行已经被提取
		NULL	游标已经被打开，但没有执行FETCH
%NOTFOUND	显示数据行未被返回。该属性与%FOUND相反	TRUE	以前的FETCH没有返回数据行
		FALSE	以前的FETCH返回了数据行；
		NULL	游标已经被打开，但没有执行FETCH
%ISOPEN	显示游标是否被打开	TRUE	游标已经被打开
		FALSE	游标还没有被打开或已被关闭
%ROWCOUNT	返回该游标迄今为止返回的行数	0	游标已经被打开，但没有执行FETCH操作
		正整数	该游标已经用FETCH提取的行数

游标属性

cursor_name.attribute

引用游标属性。

关键字

cursor_name

以前声明的PL/SQL游标名。该游标可以是关闭的或打开的。如果引用了与隐式游标相关的属性（即单行SELECT、INSERT、DELETE或UPDATE），则隐式游标名是SQL。

attribute

被引用的属性名。

示例

下面的例子声明了从scott.emp表中选择sal列的值大于100的行的游标。执行块将每一行读入到薪水变量salary和增长变量名sal_total中（在LOOP中使用FETCH），直到所有的行都被读完为止。%NOTFOUND属性用于检测所有行都被处理时的情况：

```
DECLARE
   sal_total NUMBER :=0;    /*薪水总计变量 */
   salary emp.sal%TYPE      /*被游标取得的薪水变量*/
   CURSOR emp_cursor IS     /*声明游标 */
          SELECT sal FROM scott.emp WHERE sal > 100;
BEGIN
   OPEN emp_cursor;       /*打开游标*/

   LOOP
      FETCH emp_cursor INTO salary;  /*读入一行到salary中 */
      EXIT WHEN emp_cursor%NOTFOUND; /*检查是否所有行都被读取*/
      sal_total := sal_total + salary;  /*计算总计薪水 */
   END LOOP;              /*回到循环顶端 */

    CLOSE emp_cursor;      /*关闭游标*/
   END;
```

异常部分

即使是写得最好的PL/SQL程序也会遇到错误或未预料到的事件。PL/SQL通过使用异常和异常处理器提供了有力、灵活的方法处理预料到和未预料到的事件。任何Oracle错误（报告为ORA-*xxxxx*形式的Oracle错误号）、PL/SQL运行错误或用户定义条件（不一定是错误），都可以被处理。

注意：PL/SQL编译错误不能通过PL/SQL异常处理来处理，因为这些错误发生在PL/SQL程序执行之前。

PL/SQL错误处理非常简单。当在运行中的PL/SQL程序发生错误时，就提供异常并用适当的异常处理器进行控制。

异常的类型

PL/SQL处理了两种类型的异常：预定义的和用户定义的。用户定义的异常必须在PL/SQL的DECLARE部分声明（参见本章前面的“声明异常”部分）。提供预定义异常是为了处理最常用的错误，而且摘要信息如表7-6所示。

表7-6：预定义的PL/SQL异常

异常	Oracle错误	描述
CURSOR_ALREADY_OPEN	ORA-06511	当游标已经处于打开状态时，试图再次打开游标
DUP_VAL_ON_INDEX	ORA-00001	违反了惟一性限制
INVALID_CURSOR	ORA-01001	非法的操作，如试图关闭已经关闭的游标
INVALID_NUMBER	ORA-01722	在SQL语句中将字符进行显式或隐式转换为数字时失败
LOGIN_DENIED	ORA-01017	因为无效的用户名或口令，所以登录被拒绝
NO_DATA_FOUND	ORA-01404	查询没有返回行
NOT_LOGGED_ON	ORA-01012	会话没有连接到Oracle上

表 7-6: 预定义的 PL/SQL 异常（续）

异常	Oracle 错误	描述
OTHERS		没有预定义的错误发生了
PROGRAM_ERROR	ORA-06501	内部的 PL/SQL 错误发生
ROWTYPE_MISMATCH	ORA-06504	主机游标变量与 PL/SQL 游标变量的类型不匹配
STORAGE_ERROR	ORA-06500	进程超出了内存范围
TIMEOUT_ON_RESOURCE	ORA-00051	需要资源，但操作超时。这个事件通常在等待锁时发生
TOO_MANY_ROWS	ORA-01422	SELECT 语句返回不止一行
TRANSACTION_BACKED_OUT	ORA-00061	死锁条件强制 ROLLBACK
VALUE_ERROR	ORA-06502	在过程语句中发生了转换错误
ZERO_DIVIDE	ORA-01476	试图被 0 除

注意：当异常发生时，也就是说“提供”。预定义异常在异常事件发生时提供。用户定义异常通过显式执行 RAISE 命令提供（在随后的章节“异常处理语句”中描述）。无论在哪种情况下，当发生异常时，则立即将执行转移到合适的异常处理器上。该转移是单向的；没有方法可以用于返回到执行部分。

大家可以通过使用嵌套 PL/SQL 块轻易地在这种限制上工作；一旦处理了异常，则可以将控制传递到外部块中继续执行。

异常处理

异常部分的 PL/SQL 块具有下列语法：

```
EXCEPTION
   [WHEN exception_name THEN
      pl/sql_statements
   [WHEN exception_name THEN
      pl/sql_statements...]]
   [WHEN OTHERS THEN
      pl/sql_statements]
END;
```

关键字

exception_name

预定义异常名或以前定义的用户定义异常。

pl/sql_statements

指定产生异常时的一个或多个有效 SQL 或 PL/SQL 语句。

WHEN OTHERS

定义没有应用其他处理器时，执行捕获所有异常的处理器。

END

标识异常部分结束和块结束。

示例

下面例子中的代码试图为特定的员工提取薪水和姓名（如果没有指定员工的记录，则插入一行到错误日志表中）：

```
DECLARE
   emp_name emp.ename%type;
   emp_sal emp.sal%type;
   empno_to_find NUMBER := 9999;
BEGIN
   SELECT ename, sal
   INTO emp_name, emp_sal
   FROM emp
   WHERE EMPNO = empno_to_find;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
      INSERT INTO error_log (empno, error_message)
      VALUES (empno_to_find, 'Missing employee record');
END;
```

异常处理函数

表 7-7 列出了两个 PL/SQL 嵌入函数，可以更方便地进行错误处理，特别是如果执行 WHEN OTHERS 异常处理器时更为方便。

表 7-7: 异常处理函数

函数	返回值
SQLCODE	遇到的 Oracle 错误号
SQLERRM	遇到的 Oracle 错误文本

抛出异常

在 PL/SQL 块中下列 3 种情况会抛出异常:

- 可以发生一个 Oracle 错误，这时自动触发异常。
- 可以使用 RAISE 语句显式提供异常。
- 可以调用 RAISE_APPLICATION_ERROR 过程。

异常作为Oracle错误是自动进行的，而且不需要讨论它们。其他两种方法必须用编写的代码实现。

异常处理语句

大家可以使用PL/SQL 块中的RAISE 语句和RAISE_APPLICATION_ERROR过程产生异常。也可以将它们包括在异常处理器中，从而重新提供异常或提供新的异常。

RAISE

RAISE *exception_name*

提供异常。通常RAISE语句用于提供以前声明的用户定义异常，但也可以提供预定义异常之一。

注意: 当大家从PL/SQL块内部的执行部分执行RTAISE时，就必须总是指定想要提供的异常名。大家也可以使用异常处理器内部的RAISE，重新提供前一级PL/SQL块的异常。当大家这样做时，就可以省略退出名，使当前异常再次出现。

示例

下面的例子演示了RAISE语句的一种使用方法（当试图使用create_emp过程创建新职员却忘记指定薪水大于0时，过程将失败并提供错误）：

```
CREATE OR REPLACE
PROCEDURE create_emp (emp_no IN NUMBER,
                      emp_name IN VARCHAR2,
                      emp_sal IN NUMBER)
IS
   --声明一个用户定义异常，并使其与错误码-20000相关
   no_salary EXCEPTION;
   PRAGMA EXCEPTION_INIT(no_salary,-20000);
BEGIN
   --如果薪水为0或为null，则提供异常
   IF NVL(emp_sal,0) = 0 THEN
      RAISE no_salary;
   END IF;

   INSERT INTO emp (empno, ename, sal)
   VALUES (emp_no, emp_name, emp_sal);
EXCEPTION
   WHEN no_salary THEN
      --重新提供异常
      RAISE;
END;
```

当试图使用create_emp插入一个带有0薪水的记录时，就提供no_salary异常。该异常使控制跳转到相关的异常处理器中。在异常处理器中，RAISE语句再次产生了相同的异常，并将其传递到调用过程的代码中。

RAISE_APPLICATION_ERROR

RAISE_APPLICATION_ERROR (*error_number,error_text[,keep_errors]*);

提供一种方法定义没有被标准Oracle错误处理的条件的错误号和消息。

关键字

error_number

指定想要用于提供的错误号。该错误号应该在-20 000和-20 999之间提供，以免与Oracle错误号冲突。该范围是Oracle为用户定义的错误号设置的。

error_text

长度为1~2048字节的字符串，提供与提供的错误相关的文本。

keep_errors

布尔值，为TRUE或FALSE。如果指定为TRUE，则将错误添加到已经提供的错误清单中。如果为FALSE，则使用以前的错误代替堆栈中的错误。默认的行为指定为FALSE。

注意： RAISE_APPLICATION_ERROR通常在触发器中很有用。如果决定在触发器中想让触发SQL语句失败，所有需要完成的事情就是调用RAISE_APPLICATION_ERROR。要了解有关触发器的更多信息，请参见本章后面的“触发器”一节。

示例

下面的代码显示在触发器中使用RAISE_APPLICATION_ERROR阻止没有员工姓名的新员工记录插入：

```
CREATE OR REPLACE TRIGGER emp_insert
BEFORE INSERT ON emp
FOR EACH ROW
BEGIN
   IF :new.ename IS NULL THEN
      RAISE_APPLICATION_ERROR(-20000,'Employees must have a name.');
   END IF;
END;
```

该触发器中的代码检查是否新记录的员工名为NULL。如果新名字为NULL，则调用RAISE_APPLICATION_ERROR。在BEFORE触发器中产生的错误总是引起触发语句失败。现在，有更容易的方式强制NOT NULL约束，但该代码清楚地显示了使用RAISE_APPLICATION_ERROR的潜力。

过程和包

大部分第三代语言都包括创建离散单元的能力，叫做子程序（subprogram），可以被其他程序执行。PL/SQL也包括该能力。使用PL/SQL，大家可以应用两种形式的子程序：过程和函数。每个子程序的特性如表7-8所示。

表 7-8: PL/SQL 过程和函数特性

特性	过程	函数
可以接受输入参数	是	是
可以返回参数值	是	是
返回单一值	否	是
可以在表达式中引用	否	是
以编译的形式存储在数据库中	是	是

注意：大家可以从表7-8看到，过程和函数之间的主要差别是函数返回单个值，而且用在可以使用PL/SQL变量或常量的PL/SQL表达式中。另一方面，PL/SQL过程则不返回值，而且不能形成表达式部分。

过程

PL/SQL过程就像其他语言中的子程序。它可以接受一个或多个参数作为输入，可以在这些参数上操作，而且可以使用新值代替这些参数的一个或多个值。调用过程的程序（调用程序），然后可以访问这些值。使用SQL DDL语句CREATE PROCEDURE、ALTER PROCEDURE和DROP PROCEDURE可以创建和维护PL/SQL过程。

CREATE PROCEDURE

```
CREATE [OR REPLACE] PROCEDURE [schema.]procedure_name
[(argument [IN | OUT | IN OUT] [NOCOPY] datatype
[,argument [IN | OUT | IN OUT] [NOCOPY] datatype ...])]
[AUTHID {CURRENT_USER | DEFINER}]
{IS | AS} {plsql_code |
          LANGUAGE {JAVA NAME 'string' |
          C [NAME name] LIBRARY libname [WITH CONTEXT] [PARAMETERS parms]}
```

创建过程或使用新版过程代替旧版过程。

关键字

OR REPLACE

指定代替已有的存储过程。

schema

包含过程的架构名。默认将过程存储在当前架构中。

procedure_name

提供给该过程的名字。

argument

过程的参数名。

IN

指定过程调用时该参数是输入参数。这是默认说明。在过程中，不允许给输入参数赋值。

OUT

指定该参数是一个输出参数，而且在将控制返回到调用程序之前通过过程进行设置。在过程中，允许将值存储在输出变量中，但不允许查阅该值。

IN OUT

指定该参数既是输入又是输出。调用过程时必须提供该值，而且可以在返回之前通过过程设置为另一值。

NOCOPY

指定该参数应该被引用传递，而不是通过值传递。通过值传递参数需要进行赋值。通过引用传递参数只需要传递内存地址，因此提供了明显加强的性能。该关键字影响 OUT 和 IN OUT 参数，默认为 IN 参数。

datatype

参数的数据类型，可以是 PL/SQL 支持的任何数据类型。

AUTHID

指定在谁的权限下执行函数。

CURRENT_USER

指定执行该函数时将使用当前用户的架构和权限。

DEFINER

指定执行该函数时将使用函数所有者的架构和权限，这是默认行为。

plsql_code

实现该过程的 PL/SQL 代码。

LANGUAGE

指定该PL/SQL过程映射为Java或C方法。

JAVA NAME

指定将函数映射为Java方法。

string

指定方法的Java实现。

C [NAME]

指定将函数影射为C方法。

name

指定C程序名。

libname

指定包含C程序的库名。

WITH CONTEXT

指定包含C程序的内容指针。

parms

指定传递到C程序的参数。

注意：如果大家从SQL*Plus和未正确编译的过程中创建过程，就可以使用SQL*Plus命令SHOW ERRORS查看产生的特定错误信息。

示例

下面的例子采用CREATE PROCEDURE语句创建过程，从emp表中读取行并给员工加薪。第一个参数p_ename定义了员工的名字。该过程计算新的薪水并返回到第二个参数newsal中：

```
CREATE OR REPLACE PROCEDURE give_raise
(p_ename IN VARCHAR2,
 newsal OUT NUMBER)
AS
   t_sal NUMBER;
   t_deptno NUMBER;
```

```
BEGIN
   SELECT sal, deptno
   INTO t_sal, t_deptno
   FROM emp
   WHERE ename = p_ename;

   IF t_deptno = 10 THEN
      newsal := t_sal*1.1;
   ELSIF t_deptno =20 THEN
      newsal := t_sal*1.08;
   ELSE
      newsal := t_sal*1.05;
   END IF;
END;
```

注意：关键字 CREATE [OR REPLACE] PROCEDURE... [AS|IS] 代替通常在 PL/SQL 代码块开头找到的 DECLARE 标签。此时，隐式使用了 DECLARE，在过程后进行声明。

下面的例子显示了从 PL/SQL 块的输出，从 SQL*Plus 下执行，演示可以调用 give_raise 过程。在第一次调用 give_raise 时，给员工姓名分配变量，然后将该变量作为一个参数传递给 give_raise:

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2     emp_name emp.ename%type;
  3     new_salary NUMBER;
  4  BEGIN
  5     give_raise('KING', new_salary);
  6     DBMS_OUTPUT.PUT_LINE('King''s new salary is ' ||
  7                          TO_CHAR(new_salary,'$999,999.99'));
  8
  9     emp_name:='SCOTT';
 10     give_raise(emp_name, new_salary);
 11     DBMS_OUTPUT.PUT_LINE('Scott''s new salary is ' ||
 12                          TO_CHAR(new_salary,'$999,999.99'));
 13  END;
 14  /
King's new salary is    $5,500.00
Scott's new salary is    $3,240.00

PL/SQL procedure successfully completed.
```

ALTER PROCEDURE

ALTER PROCEDURE [*schema.*]*procedure_name* COMPILE [DEBUG]

重新编译 Oracle 数据库中已经存在的过程。

关键字

schema

包含过程的架构名。如果省略，则假定为当前架构。

procedure_name

被重新编译的已有过程名。

COMPILE

导致重新编译过程。

DEBUG

使 PL/SQL 编译器为 PL/SQL 调试器产生调试代码。

注意：该命令首先导致重新编译过程上的所有对象。如果有任何对象无效，则重新编译的过程也被标记为无效。

示例

下面的例子显式地重新编译架构 scott 中名为 give_raise 的过程：

```
ALTER PROCEDURE scott.give_raise COMPILE
```

DROP PROCEDURE

DROP PROCEDURE [*schema.*]*procedure_name*

从数据库中删除过程。

关键字

schema

包含被删除的过程的架构名。如果省略，则假定为当前架构。

procedure_name

指定被删除的过程名。

示例

下面的例子从数据库中删除过程 give_raise:

```
DROP PROCEDURE give_raise
```

在这一例子中，因为在 DROP PROCEDURE 命令中不是显式指定的，所以大家需要以被删除过程的过程所有者登录。

函数

PL/SQL 函数与其他许多语言的函数相似。像 PL/SQL 过程一样，PL/SQL 函数采用 SQL DDL 语句创建并维护，这在本节中已经列出。如前所述，函数非常像过程：可以给函数传递参数，而且也可以从函数中输出参数（而且既可以是输入又可以是输出）；它作为其他 PL/SQL 块的相同结构采用一块或多块 PL/SQL 代码创建，而且可以存储在数据库中。但是，它们之间有一个严格的区别：PL/SQL 函数返回单个值，所以用作 PL/SQL 表达式的一部分。

PL/SQL 函数使用 SQL DDL 语句 CREATE FUNCTION、ALTER FUNCTION 和 DROP FUNCTION 创建并维护。

CREATE FUNCTION

```
CREATE [OR REPLACE] FUNCTION [schema.]function_name
[(argument [IN | OUT | IN OUT] [NOCOPY] datatype
[,argument [IN | OUT | IN OUT] [NOCOPY] datatype...])]
RETURN datatype
[AUTHID {CURRENT_USER | DEFINER}]
[DETERMINISTIC]
[PARALLEL_ENABLE]
{IS|AS} {plsql_code |
  LANGUAGE {JAVA NAME 'string' |
                C [NAME name] LIBRARY libname [WITH CONTEXT] [PARAMETERS parms]}
```

创建存储函数。

关键字

OR REPLACE

指定代替已有的函数。

schema

包含函数的架构名。默认为当前用户的架构。

function_name

提供给要被创建的函数的名字。

argument

函数的参数名。

IN

指定函数调用时该参数是输入参数，而且必须给该参数提供值；这是默认参数类型。

OUT

指定该参数是一个输出参数，而且在返回之前通过函数设置。

IN OUT

指定该参数既是输入又是输出。调用过程时可以提供该值，而且可以在返回之前改变该值。

NOCOPY

指定该参数应该被引用传递，而不是通过值传递。通过值传递参数需要进行赋值。通过引用传递参数只需要传递内存地址，因此提供了明显加强的性能。该关键字影响OUT和IN OUT参数，默认为IN参数。

datatype

参数的数据类型，可以是PL/SQL支持的任何数据类型。

AUTHID

指定在谁的权限下执行函数。

CURRENT_USER

指定执行该函数时将使用当前用户的架构和权限。

DEFINER

指定执行该函数时将采用函数所有者的架构和权限，这是默认行为。

DETERMINISTIC

告诉Oracle优化器该函数将一直为给定的输入返回相同结果。查询优化器然后使用该信息避免函数的冗余调用。如果除了输入参数外还有其他因素能影响函数的结果，就不要使用DETERMINISTIC。

PARALLEL_ENABLE

指定查询优化器可以以并行方式执行该函数，作为并行查询操作的一部分。如果函数不使用包变量或者其他会话指定信息，则只能指定PARALLEL_ENABLE。

plsql_code

实现该函数的PL/SQL代码。

LANGUAGE

指定该PL/SQL函数映射为Java或C方法。

JAVA NAME

指定将函数映射为Java方法。

string

指定该方法的Java实现。

C [NAME]

指定将函数映射为C方法。

name

指定C程序名。

libname

指定包含C程序的库名。

WITH CONTEXT

指定包含C程序的内容指针。

parms

指定传递到C程序的参数。

注意： 与存储过程一样，如果从SQL*Plus中创建函数，而且没有纠正函数中的编译错误，就可以使用SQL*Plus命令SHOW ERRORS查看产生的特定错误信息。

示例

下面例子中的CREATE FUNCTION语句从SQL*Plus下创建函数，目的是从scott用户的emp表中返回员工的薪水。员工号用于定义员工，并作为参数传递到函数中：

```
CREATE FUNCTION get_sal
(emp_num IN NUMBER)
RETURN NUMBER
IS
  emp_sal NUMBER (8,2);
BEGIN
  SELECT sal
  INTO emp_sal
  FROM scott.emp
  WHERE emp_num = empno;
  RETURN emp_sal;
END;
```

创建了get_sal函数之后，可以使用下面的代码得到指定员工的薪水：

```
SQL> select get_sal(7788) from dual;

GET_SAL(7788)
-------------
         3000
```

在scott的仿真架构中，员工号7788叫做SCOTT。大家可以从该例子中看到，他的薪水是$3000。

ALTER FUNCTION

ALTER FUNCTION [*schema*.]*function_name* COMPILE [DEBUG]

重新编译存储的函数。

关键字

schema

包含函数的架构名。如果省略，则假定为当前架构。

function_name

被重新编译的函数名。

COMPILE

导致重新编译函数。

DEBUG

使PL/SQL编译器为PL/SQL调试器产生调试代码。

示例

下面的例子显示怎样使用 ALTER FUNCTION 重新编译 get_sal 函数:

```
ALTER FUNCTION get_sal COMPILE
```

DROP FUNCTION

DROP FUNCTION [*schema*.]*function_name*

从数据库中删除函数。

关键字

schema

包含被删除的函数架构名。如果省略，则假定为当前架构。

function_name

被删除的函数名。

示例

使用下面的语句删除函数 get_sal:

```
DROP FUNCTION get_sal
```

包

这通常是创建一定数量的相关PL/SQL过程和函数，并一同维护和存储的理想方法。PL/SQL包就是为这个目的设计的。PL/SQL包是采用单组命令创建的相关函数、过程和变量的集合，并一起存储在数据库中。

包可以包含函数、过程、变量类型和游标。通常，在PL/SQL程序中的DECLARE部分出现的任何内容都可以包含在包中。

注意: 使用包的一个好处是提供了实现全局PL/SQL变量的机制。包的状态通过数据库会话来维护，所以，一旦给包变量设置了值，那么在任何PL/SQL块中的代码都可以访问该值，而且作为相同会话的一相同部分。更重要的是，包变量可以充当全局变量。

包不像函数和过程那样，可以使用单个CREATE语句创建，创建包的步骤分为两步：

1. 创建包说明，有时指包标题（package header）。
2. 创建包体（pachage body）。

包说明可以认为是包的“内容表”。它包含包的信息，如过程和函数定义，但不包含实际的PL/SQL代码。

包体包含用于实现包说明中声明的函数和过程的所有代码。注意，该代码包含函数和/或过程的说明（例如，参数名和参数模式），即使这些说明已经包括在包说明中。此外，这些说明必须与包说明和包体中的一样。

使用SQL DDL语句CREATE PACKAGE、CREATE PACKAGE BODY、ALTER PACKAGE和DROP PACKAGE可以创建和维护PL/SQL包。

CREATE PACKAGE

```
CREATE [OR REPLACE] PACKAGE [schema.]package_name
[AUTHID {CURRENT_USER | DEFINER}]
{IS | AS} plsql_package_spec
```

创建PL/SQL包说明。

关键字

OR REPLACE

指定代替已有的包。

schema

包含包的架构名。如果省略，则默认为当前用户的架构。

package_name

创建的包名字。

AUTHID

指定在谁的函数权限下执行包中的函数和过程。

CURRENT_USER

指定在执行包中的函数和过程时，将使用当前用户的架构和权限。

DEFINER

指定执行包中的函数和过程时，将使用当前包所有者的架构和权限，这是默认行为。

plsql_packae_spec

PL/SQL 包说明，可以包括包和函数说明、变量声明、游标声明和类型声明。

注意：可以使用 SQL*Plus 命令 SHOW ERRORS 显示创建包说明时产生的特定错误信息。

示例

下面的例子为 scott 用户的架构中名为 emp_pkg 的包创建包说明：

```
CREATE OR REPLACE PACKAGE scott.emp_ pkg AS
  FUNCTION emp_sal (empnum IN NUMBER) RETURN NUMBER;
  PROCEDURE update_sal (empnum IN NUMBER, pct_increase IN NUMBER,
                        update_count OUT NUMBER);
  PROCEDURE add_bonus (empnum IN NUMBER, bonus_amt IN NUMBER);
END emp_ pkg;
```

CREATE PACKAGE BODY

```
CREATE [OR REPLACE] PACKAGE BODY [schema.]package_name
{IS | AS}
plsql_declaration [plsql_declaration ...]
[BEGIN
  pl/sql_initialization_code]
END [package_name];
```

创建 PL/SQL 包体。

关键字

OR REPLACE

指定代替已有的包体。

schema

包含包体的架构名。如果省略，则默认为当前用户的架构。

package_name

创建的包体所在的包名。

plsql_declaration

PL/SQL 定义，可以是过程、函数、变量、类型或游标。

Plsql_initialization_code

通过 Oracle 会话第一次将包加载到内存中时执行的 PL/SQL 代码块。

在创建包体之前，包说明应该已经存在；否则，Oracle 将产生告警信息。

示例

下面的例子显示了用 CREATE PACKAGE BODY 语句创建 scott 用户的架构中的 emp_pkg 包的包体：

```
CREATE OR REPLACE PACKAGE BODY scott.emp_ pkg AS
--声明包层变量
update_count NUMBER;

FUNCTION emp_sal (empnum IN NUMBER)
RETURN NUMBER IS
   emp_sal NUMBER (8,2);
BEGIN
  SELECT sal
  INTO emp_sal
  FROM scott.emp
  WHERE empnum = empno;
  RETURN emp_sal;
END;

PROCEDURE update_sal
(empnum IN NUMBER, pct_increase IN NUMBER, update_count OUT NUMBER)
IS
BEGIN
  UPDATE scott.emp
  SET sal = sal+ sal*pct_increase
  WHERE empnum = empno;
  update_count := update_count + 1;
END;

PROCEDURE add_bonus (empnum IN NUMBER, bonus_amt IN NUMBER) is
BEGIN
  UPDATE scott.emp
  SET comm = comm + bonus_amt
  WHERE empnum = empno;
  update_count := update_count + 1;
END;
BEGIN
  --这是包初始化代码。
```

```
  update_count := 0;
END emp_ pkg;
```

注意：这个CREATE PACKAGE BODY语句对应于前面一节显示的CREATE PACKAGE语句。在CREATE PACKAGE语句中，定义的每一个函数或过程也在CREATE PACKAGE BODY语句中定义。也要注意，那些声明下至函数参数名和类型都是相同的。那些项目必须相同。如果在说明中和函数体中使用了不同的名字，则PL/SQL不会认为这两者匹配。

ALTER PACKAGE

ALTER PACKAGE [*schema*.]*package_name*
COMPILE [DEBUG] [PACKAGE | SPECIFICATION | BODY]

重新编译存储包说明、包体，或者两者都重新编译。

关键字

schema

包含包的架构名。如果省略，则默认为当前用户的架构。

package_name

创建的包名字。

COMPILE

重新编译包。

DEBUG

使PL/SQL编译器创建与PL/SQL一同使用的调试信息。

PACKAGE

指定重新编译包说明和包体；这是默认行为。

SPECIFICATION

指定只重新编译包说明。

BODY

指定只重新编译包体。

示例

下面的例子显式地重新编译了 scott 用户的架构和 emp_pkg 包体和包说明:

```
ALTER PACKAGE scott.emp_pkg COMPILE PACKAGE
```

下面的语句只重新编译包体:

```
ALTER PACKAGE emp_pkg COMPILE BODY
```

DROP PACKAGE

DROP PACKAGE [BODY][*schema*.]*package_name*

从数据库中删除包。

关键字

schema

包含被删除的包的架构名。如果省略，则使用当前架构。

package_name

被删除的包名。

BODY

指定只删除包体，而且不删除包说明。

注意: 如果使用 DROP PACKAGE 而不使用 BODY 限定词，则包说明和包体都被删除。

示例

下面的语句删除命名包 emp_pkg 中的包说明和包体:

```
DROP PACKAGE emp_ pkg
```

下面的语句只删除 emp_pkg 包体:

```
DROP PACKAGE BODY emp_ pkg
```

触发器

触发器代表了依赖于事件的特殊类型的PL/SQL块。当通过Oracle数据库执行触发器时，也就是说“激活（fire）”。最常用的触发器类型是DML触发器，它触发对INSERT、UPDATE和DELETE语句的响应。有15种不同类型的DML触发器，在表7-9中列出。

表7-9：Oracle的DML触发器类型

触发器事件	触发对象	描述
BEFORE INSERT	语句	在目标表上执行INSERT语句之前执行代码
BEFORE INSERT	行	在每一行被插入到在目标表上之前执行代码
AFTER INSERT	语句	在目标表上执行INSERT语句之后执行代码
AFTER INSERT	行	在每一行被插入到在目标表上之后执行代码
INSTEAD OF INSERT	行	在试图插入的每一行上执行代码，而不是执行INSERT操作
BEFORE UPDATE	语句	在目标表上执行UPDATE语句之前执行代码
BEFORE UPDATE	行	在目标表上每一行被更新之前执行代码
AFTER UPDATE	语句	在目标表上执行UPDATE语句之后执行代码
AFTER UPDATE	行	在目标表上每一行被更新之后执行代码
INSTEAD OF UPDATE	行	在试图更新的每一行上执行代码，而不是执行UPDATE语句
BEFORE DELETE	语句	在目标表上执行DELETE语句之前执行代码
BEFORE DELETE	行	在目标表上每一行被删除之前执行代码
AFTER DELETE	语句	在目标表上执行DELETE语句之后执行代码
AFTER DELETE	行	在目标表上每一行被删除之后执行代码
INSTEAD OF DELETE	行	在试图删除的每一行上执行代码，而不是执行DELETE语句

注意 结合表7-9中描述的DML触发器一起，Oracle8*i*介绍的数据库事件上的触发器允许在实例启动、实例关闭和用户连接时自动运行代码。这些新的触发器的每一种类代表了自己权限下的很大方面。在本书中，我只关注DML触发器，这是大家最可能遇到的数据库触发器。要了解数据库触发器的更多信息，参见Steven Feuerstein写的《Oracle PL/SQL Programming Guide to Oracle8*i* Features》(O'Reilly & Associates, 1999)一书。

触发器可以包括任何合法的PL/SQL语句，但有以下例外：

- 触发器不可以执行COMMIT、ROLLBACK或SAVEPOINT语句，而且不可以调用执行这些语句之一的函数或过程。
- 触发器不可以声明LONG或LONG RAW变量。
- 触发器不可以在定义它的表上执行DML操作。

下面的部分显示了与触发器一同使用的DDL语句的SQL语法。

使用SQL DDL语句CREATE TRIGGER、ALTER TRIGGER和DROP TRIGGER可以创建和维护触发器。

CREATE TRIGGER

```
CREATE [OR REPLACE]  TRIGGER [triggerschema.]triggername
{BEFORE | AFTER | INSTEAD OF}
{INSERT | DELETE | UPDATE [OF column[,column ...]]}
[OR {INSERT | DELETE | UPDATE [OF column[,column ...]]}...]
ON [tableschema.]{tablename | viewname}
   [REFERENCING [OLD [AS] old ] [NEW [AS] new]]
   [FOR EACH ROW [WHEN condition]]
   plsql_block
```

创建与表或视图相关的触发器（存储的PL/SQL块），将在响应特定的SQL语句或语句集时自动执行。

关键字

triggerschema

包含触发器的架构名。如果省略，就假定为当前架构。

triggername

被创建的触发器名。

BEFORE

指定执行触发语句之前激活触发器。

AFTER

指定执行触发语句之后激活触发器。

INSTEAD OF

指定执行与该触发器相关的代码，而不是事件最初激活的触发器。

INSERT

指定当 INSERT 语句添加行到表中时激活触发器。

DELETE

指定当 DELETE 语句从表中删除行时激活触发器。

UPDATE OF

指定当 UPDATE 语句改变 OF 从句中的一个列值时激活触发器。如果省略 OF 从句，则对任何列的 UPDATE 都会激活触发器。

ON

提出在创建的触发器上指定表名的从句。

tableschema

包含了触发器定义的表的架构名。默认该名字为当前用户的架构。

tablename

定义触发器的表名。该表必须在特定的架构中存在。

viewname

定义触发器的视图名。该视图必须存在于特定的架构中。

REFERENCING

指定相关性名称。该说明允许 PL/SQL 块中的代码指向激活触发器的 DML 语句影响的行的新值或老值。

FOR EACH ROW

指定该触发器是行触发器，被触发机制影响的每一行及满足 WHEN 从句中指定的条件时，激活该触发器一次。如果省略 FOR EACH ROW，则在每个执行的触发语句中都激活该触发器。

WHEN

指定为了激活触发器，SQL的条件必须为真。

plsql_block

触发器激活时要执行的PL/SQL块。该块不可以包含COMMIT、ROLLBACK或SAVEPOINT语句。

示例

下面例子中的语句创建了名为track_sal_changes的审核表和在scott用户的emp表上的触发器。在响应INSERT语句和响应UPDATE语句导致薪水改变时激活该触发器。然后该触发器记录老的薪水到审核表中：

```
CREATE TABLE track_sal_changes (
   empno NUMBER,
   sal NUMBER)
/
CREATE OR REPLACE TRIGGER scott.empaud
   BEFORE INSERT OR UPDATE OF sal
   ON scott.emp
   FOR EACH ROW WHEN (
      (new.sal <> old.sal)
      OR ((new.sal IS NOT NULL) AND (old.sal IS NULL)))

   BEGIN
   /* 写记录 */
      INSERT INTO track_sal_changes
      VALUES (:new.empno, :old.sal);
     END;
```

ALTER TRIGGER

```
ALTER TRIGGER [schema.]triggername {ENABLE | DISABLE}
```

允许或禁止数据库上的触发器生效。

关键字

schema

包含触发器的架构名。如果省略，就默认为当前架构。

triggername

指定要改变的触发器名。

ENABLE

指定执行触发语句时激活该触发器。

DISABLE

指定执行触发语句时不激活该触发器。

注意：不像其他的ALTER命令，ALTER TRIGGER语句不改变触发器的结构和定义：这些变化必须与CREATE OR REPLACE TRIGGER语句一同完成。创建触发器时自动允许触发器生效。ALTER TABLE语句的ENABLE ALL TRIGGERS和DISABLE ALL TRIGGERS从句也执行这一功能。

示例

下面的语句禁止scott用户的cmpaud触发器生效：

```
ALTER TRIGGER scott.empaud DISABLE
```

DROP TRIGGER

DROP TRIGGER [*schema*.]*trigger_name*

从数据库中删除触发器。

关键字

schema

包含要被删除的触发器的架构名。如果省略，就默认为当前的架构。

triggername

指定被删除的触发器。

示例

下面的例子从数据库中删除empaud触发器：

```
DROP TRIGGER empaud
```

第八章

SQL 语句调试

一旦大家工作在Oracle下，不久就会面临调试SQL语句的问题。大家可以编写一个“简单”的查询，并计划运行大约一周结束。通过编写该程序，大家可以修正其中的问题，如果不修正这些问题，就有可能会编写出执行时间更长的查询。大家将会看到，利用Oracle的所有知识源（随后大家将看到该运行一周的查询可以在一分钟之内运行!），可以使该查询运行更快。调试SQL语句与编写SQL语句是紧密相关的工作。

在有效调试SQL语句之前，大家需要理解Oracle执行SQL语句的方法，然后决定这是否是最好的方法。最后，假定有更好的方法，就需要使Oracle采用该方法。为了帮助大家完成该项工作，Oracle已经在其软件中嵌入了下列功能:

EXPLAIN PLAN 语句

大家可以使用该语句确定Oracle一条SQL语句的执行计划。

SQL TRACE 工具

不仅可以得到执行计划，而且可以收集相关语句执行的重要统计信息。

*SQL*Plus SET AUTOTRACE 命令*

使SQL*Plus在执行查询时自动显示查询计划和语句执行的统计资料。

*SQL*Plus TIMING 命令*

允许大家测量执行过程所消耗的时间。

Optimizer hints（优化器提示）

允许大家告诉Oracle将怎样执行特定查询。

调试过程通常集中在SELECT语句上进行，而且这些语句是本章中所有例子的基础。注意，尽管INSERT、DELETE和UPDATE语句可以采用与SELECT语句相同的方式进行调试，但在这里，大部分还是采用SELECT语句进行举例说明。

使用 EXPLAIN PLAN

在希望使一条执行较慢的SQL语句运行更快时，大家首先需要找出Oracle执行当前语句的过程。无论什么时候执行一条语句，Oracle中称为优化器（optimizer）的部分构建了一系列习惯于执行该语句的步骤清单，该步骤列表就称为执行计划。例如，如果大家从表中选择资料并打印输出报表，一个可能的执行计划就是读取表中的所有行。大部分执行计划比这更复杂，而且有可能包括了如读索引、连接表和排序数据的任务。

注意：Oracle包括两个优化器：基于规则的优化器和基于成本的优化器。规则优化器有点老，而且Oracle不鼓励使用它。成本优化器基于ANALYZE语句收集的统计信息作出决定。OPTIMIZER_MODE初始化参数控制了实例选择的默认优化器。大家可以考虑不在会话层使用ALTER SESSION语句，而在语句层上使用提示。

大家会发现Oracle执行计划将使用Oracle的EXPLAIN PLAN语句用于SQL语句。该语句是大家在调试中感兴趣的预先计划的语句。例如：

```
EXPLAIN PLAN
  SET STATEMENT_ID = 'emp_report'
  FOR
  SELECT empno, ename
  FROM emp
  ORDER BY hiredate;
```

在使用EXPLAIN PLAN时，Oracle不会将执行策略显示在屏幕上，相反，它在表中插入行，该表称为计划表，而且大家必须使用合适的查询才能看到结果。当然，计划表也必须存在，因此，如果大家以前从来没有使用过EXPLAIN PLAN，就需要先创建计划表，然后再执行EXPLAIN PLAN语句。

创建计划表

Oracle 提供了创建计划表的相关脚本。该脚本叫做 *utlxplan.sql*，在 Linux 或 Unix 系统上，它放置在 *$ORACLE_HOME/rdbms/admin* 目录下。在 Windows 系统上，将放置在像 *E:\ORACLE\ORA81\RDBMS\ADMIN* 或 *C:\ORANT\RDBMS80\ADMIN* 之类的目录下。大家可以在 SQL*Plus 下运行该脚本，如本例子中所示：

```
SQL> @$ORACLE_HOME/rdbms/admin/utlxplan.sql

Table created.
```

最后的计划表看起来有点像：

```
SQL> DESCRIBE plan_table

 Name                                      Null?    Type
 ----------------------------------------- -------- ----------  -------
 STATEMENT_ID                                       VARCHAR2(30)
 TIMESTAMP                                          DATE
 REMARKS                                            VARCHAR2(80)
 OPERATION                                          VARCHAR2(30)
 OPTIONS                                            VARCHAR2(30)
 OBJECT_NODE                                        VARCHAR2(128)
 OBJECT_OWNER                                       VARCHAR2(30)
 OBJECT_NAME                                        VARCHAR2(30)
 OBJECT_INSTANCE                                    NUMBER(38)
 OBJECT_TYPE                                        VARCHAR2(30)
 OPTIMIZER                                          VARCHAR2(255)
 SEARCH_COLUMNS                                     NUMBER
 ID                                                 NUMBER(38)
 PARENT_ID                                          NUMBER(38)
 POSITION                                           NUMBER(38)
 COST                                               NUMBER(38)
 CARDINALITY                                        NUMBER(38)
 BYTES                                              NUMBER(38)
 OTHER_TAG                                          VARCHAR2(255)
 PARTITION_START                                    VARCHAR2(255)
 PARTITION_STOP                                     VARCHAR2(255)
 PARTITION_ID                                       NUMBER(38)
 OTHER                                              LONG
 DISTRIBUTION                                       VARCHAR2(30)
```

表名并不一定要是 plan_table，但这是默认的表名，而且通常是最容易维持原样的方法。如果大家不想访问 *utlxplan.sql* 脚本，就可以手工创建该表。确保这里的列名与显示的数据类型相匹配就可以了。

有时需要考虑在给定数据库中应该出现的计划表的数量。理论上，大家可以只创建一张计划表，并授权所有的用户对该表进行访问，而且让每个人都可以共享。实际上，最简单的就是让每个用户创建自己的计划表。

注意： 在不同版本的Oracle中，计划表包含的列有所不同。通常，这一变化用于支持已经添加的新功能。例如，在Oracle8中介绍分区时添加了与3个分区相关的列。如果原表已经存在了一段时间，大家也可以删除原表并重新创建新表，从而确保自己具有最近版本的计划表。

解释查询

一旦拥有了计划表，就可以准备使用EXPLAIN PLAN语句。为了解释一个计划，大家需要完成3件事情：

- 为正在解释的语句分配一个语句ID。该ID可以是任何字符串，最长可以为30个字符。
- 删除已有计划表中语句ID相同的记录。
- 执行EXPLAIN PLAN语句。

在执行EXPLAIN PLAN语句时，Oracle在计划表中插入多条记录，描述该语句的执行计划的解释。计划表可以一次保存多条语句的执行计划。分配的语句ID不仅定义了给定语句的记录，而且定义了它与其他语句的记录之间的不同点。使用语句ID不是必须的，但鼓励大家多使用语句ID。

EXPLAIN PLAN语句不清空计划表中已有的记录。如果在第二次解释一条语句，则第一次计划表记录将保留在表中。如果两次都使用相同的语句ID，则会得到不可使用的结果。这些结果通常多次出现在重复相同的步骤或步骤序列中。为了安全起见，最好先删除计划表中有关将要使用的语句ID已有的记录。

EXPLAIN PLAN的语法非常简单。只要在感兴趣的SQL语句前面放置关键字EXPLAIN PLAN，并加上一些从句就可以了。完整的语法如下：

```
EXPLAIN PLAN
   SET STATEMENT_ID = 'text'
```

```
[INTO [schema.]tablename[@dblink]]
FOR SQL_statement
```

注意：要了解EXPLAIN PLAN语法，请参考第三章。

下面的例子演示了怎样使用EXPLAIN PLAN。被解释的SELECT语句运行在scott架构的表上，并产生员工清单、部门和薪水等级。该结果首先按照最高薪水进行排序。注意，首先使用了DELETE语句清除了将要使用的语句ID的老记录：

```
SQL> DELETE
  2  FROM PLAN_TABLE
  3  WHERE statement_id = 'emp_report';

8 rows deleted.

SQL>
SQL> EXPLAIN PLAN
  2  SET STATEMENT_ID = 'emp_report'
  3  FOR
  4  SELECT e.empno, e.ename, d.dname, e.sal, sg.grade
  5  FROM emp e, dept d, salgrade sg
  6  WHERE e.deptno = d.deptno
  7  AND e.sal > sg.losal
  8  AND e.sal < sg.hisal
  9  ORDER BY sal DESC;

Explained.
```

在执行该EXPLAIN PLAN语句时，除了能看到单词“Explained”以外，将不会看到任何其他输出，因为Oralce将查询计划存储在计划表中。提取并解释该结果是大家下一步的任务。

查看执行计划

执行计划本身自然分层，也就是说，执行查询的任务是一个大步骤，可以分成一个或多个小步骤进行。这些分开的步骤将一直继续进行，直到结尾为止。要看一看这样的执行计划，需要执行类似于这里显示的层次查询：

```
SELECT LPAD(' ', 2*(level-1)) ||
       operation || ' ' || options
       || ' ' || object_name || ' ' ||
       DECODE(id, 0, 'Cost = ' || position)
```

```
        "Query Plan"
FROM plan_table
START WITH id = 0 AND statement_id
           = 'statement_id'
CONNECT BY PRIOR id = parent_id
       AND statement_id = 'statement_id';
```

注意： 查询中的START WITH和CONNECT BY从句用于组织层次中的结果。计划表中的ID字段和PARENT_ID字段决定该层次；每个步骤都在父步骤下面缩进。查询选择清单中的LPAD表达式用于缩进结果，从而反映相应的层次。每个计划表记录都包含一个ID和一个PARENT_ID列。每个步骤的结果都反馈到父步骤的输入中，该父步骤用PARENT_ID标识身份。

如果该查询用于查看以前章节中解释的emp_report语句的计划，就可以显示下面的结果：

```
Query Plan
----------------------------------
SELECT STATEMENT   Cost = 9
  SORT ORDER BY
    HASH JOIN
      MERGE JOIN CARTESIAN
        TABLE ACCESS FULL DEPT
        SORT JOIN
          TABLE ACCESS FULL SALGRADE
      TABLE ACCESS FULL EMP
```

缩进最多的步骤首先被执行。在本例子中，第一个步骤是salgrade表的完全表扫描。该步骤的结果就是父步骤的反馈内容。表8-1详细解释了该计划的每一个步骤。在表8-2中解释了多种计划的操作。

表8-1：emp_report执行计划解释

执行计划步骤	解释
SORT JOIN TABLE ACCESS FULL SALGRADE	SORT JOIN操作进行行集排序，准备合并连接。在这种情况下，行集由SALGRADE表中的所有行组成
MERGE JOIN CARTESIAN TABLE ACCESS FULL DEPT SORT JOIN TABLE ACCESS FULL SALGRADE	DEPT和SALGRADE表的笛卡尔产品是这一步骤的结果。Oracle读取整个DEPT表，用以前SORT JOIN操作的结果组合行集，而且用DEPT和SALGRADE行的所有可能组合产生了一个行集

表 8-1: emp_report 执行计划解释（续）

执行计划步骤	解释
HASH JOIN MERGE JOIN CARTESIAN ... TABLE ACCESS FULL EMP	合并连接的结果然后与EMP表连接在一起。这次，Oracle使用了散列法连接两个行集。从笛卡尔产品所得的许多行将在该操作中被删除，因为只有一些记录与EMP记录相关
SORT ORDER BY HASH JOIN ...	散列连接的结果被反馈到排序操作中。该排序按照查询语句中ORDER BY从句指定的方式对结果排序
SELECT STATEMENT Cost = 9 SORT ORDER BY ...	作为SELECT语句自身的结果，排序操作的结果被反馈回用户或应用程序

注意：当执行计划中的两个步骤缩进程度相同时，就不可能知道哪个先执行。实际并没有关系，因为从用于执行语句的资源角度来说，最终结果都一样。

执行计划的成本来源于具有0 ID值的记录的POSITION字段。在本例子中，成本为9。查询成本从来不会是一个绝对值。在相同数据库上，查询成本通常与其他查询相关。对成本为9的查询而言，在只有一半的查询I/O和CPU资源时，需要的成本就会为18，但大家并不知道到底需要半分钟还是半小时，只有在查询语句实际被执行时才知道。

查询成本只有在使用成本优化器才进行估计。必须至少在查询中包括的一张表上进行统计，才能计算成本，如果存储该统计值的表，Oracle将返回到规则优化器上，而且不计算成本。

注意：保留最近的统计值。过时的统计值将导致不精确的成本，也可以导致Oracle选择较差的优化执行计划。

解释结果

上一节显示的查询返回的执行计划包括了每个步骤的3列信息：

OPERATION 列

定义执行的主要操作。

OPTIONS 列

进一步限制怎样执行操作。例如，许多排序类型。

OBJECT_NAME 列

在可应用的地方，定义被执行的操作对象。该对象通常是表或索引。如果操作的OBJECT_NAME是空白，则该对象由被操作对象的子对象返回的一个或多个行集组成。

表 8-2 描述了各种操作、而且可以在执行计划中看见这些选项。

表 8-2: 执行计划操作

操作	选项	描述
AND-EQUAL		该步骤将具有两个或更多个子步骤，每一个子步骤返回一系列的ROWID。AND-EQUAL 操作选择的只是所有子操作返回的ROWID
BITMAP	CONVERSION TO ROWIDS	将位图从位图索引转换成可以用于提取实际资料的一系列的ROWID
	CONVERSION FROM ROWIDS	将一系列的ROWID 转换为位图表示
	CONVERSION COUNT	对位图表示的行数进行计数
	INDEX SINGLE VALUE	为单个关键值提取位图。例如，如果字段是一个 YES/NO 字段，而大家的查询只想返回“YES”值的行，就可以执行该操作
	INDEX RANGE SCAN	与BITMAP INDEX SINGLE VALUE 相似，但返回的位图是一定范围的关键值
	INDEX FULL SCAN	扫描整个位图索引

表 8-2: 执行计划操作（续）

操作	选项	描述
BITMAP	MERGE	合并两张位图，并用一个位图作为结果返回。这是两个位图之间的“或（OR）”操作。结果位图将返回第一个位图的所有行，加上第二个位图的所有行
	MINUS	该操作是 MERGE 的相反操作，而且可以具有两个或三个子操作并返回位图。第一个子操作返回的位图用作起点。第一个位图减去第二个位图中提供的所有行。如果列是可以为空的，则所有具有 NULL 值的行也将被减去
	OR	将两个位图作为输入，将它们进行逻辑“或”，并返回一个位图作为结果。返回的位图将从第一个位图选择所有行，加上第二个位图中的所有行
CONNECT BY		分层提取行，因为查询采用了 CONNECT BY 从句
CONCATENATION		组合多个行集为一行，本质上就是 UNION ALL
COUNT		计算从表中已经选择的行数
	STOPKEY	计算的行数被查询语句中的 WHERE 从句中使用的 ROWNUM 所限制
FILTER		采用一系列的行作为输入，而且根据条件过滤 WHERE 从句查询得到的结果
FIRST ROW		只提取查询结果集的第一行
FOR UPDATE		对提取的行加锁。该选项将是原始查询中指定的 FOR UPDATE 的结果

表 8-2: 执行计划操作（续）

操作	选项	描述
HASH JOIN		使用散列联接方法联接两张表
INDEX	UNIQUE	从索引中查找惟一值。大家将只有在索引惟一（例如，用于执行主键或惟一键的索引）时才可看见这一值
	RANGE SCAN	被扫描的行在范围之内。以升序方式扫描索引
	RANGE SCAN DESCENDING	与 RANGE SCAN 相同，但以降序方式扫描索引
INLIST ITERATOR		要在 IN 谓词中为每个值执行一个或多个操作一次
INTERSECTION		将两个行集作为一个输入，并且只返回在两个集合中都出现的行
MERGE JOIN		根据共同值连接两个行集。所有的行集将首先根据这一值进行排序。该操作是一个内部联接
	OUTER	与MERGE JOIN相似，但执行外部联接
	ANTI	显示将执行反连接
	SEMI	显示将执行半连接
MINUS		该操作就是MINUS操作的结果。将两个行集作为输入，行集包含了第一个输入在第二个行集中不出现的所有行
NESTED LOOPS		该操作有两个子操作，每一个返回一个行集。对第一个子操作返回的每一行，将执行第二个子操作
	OUTER	代表用于执行外部联接的嵌套循环

表 8-2: 执行计划操作（续）

操作	选项	描述
PARTITION		执行一个或多个分区操作。PRATITION_START 和 PARTITION_STOP列提供了执行操作的分区范围
	SINGLE	显示该操作将在单个分区上执行
	ITERATOR	显示该操作将在许多分区上执行
	ALL	显示该操作将在所有分区上执行
	INLIST	显示该操作将在分区上执行，而且用 IN 谓词驱动
PROJECTION		采用多个查询作为输入，并返回单个记录集。该操作与 INTERSECTION、MINUS和UNION操作一同使用
REMOTE		显示行集将从远程数据库上返回
SEQUENCE		显示访问 Oracle 序号发生器
SORT	AGGREGATE	在行集上应用分组函数，如 COUNT，并只返回一行作为结果
	UNIQUE	对行集排序，并过滤重复值
	GROUP BY	对行集分组排序。该操作就是 GROUP BY 从句的结果
	JOIN	为准备联接的行集排序。参见 MERGE JOIN
	ORDER BY	根据查询中 ORDER BY 从句对行集进行排序
TABLE ACCESS	FULL	显示Oracle将读取指定表中的所有行
	CLUSTER	显示Oracle将读取表中与特定的索引簇键匹配的所有的行

表 8-2: 执行计划操作（续）

操作	选项	描述
TABLE ACCESS	HASH	显示Oracle将读取表中与特定的散列簇键匹配的所有的行。
	BY ROWID	显示了 Oracle 将从基于 ROWID 的表提取行
UNION		采用两个行集，过滤重复值，并返回一个结果集
VIEW		产生视图后执行查询并返回结果行集

一旦大家理解了Oracle用于查询的执行计划，大家就可以寻找多种方法改进结果。参见本章后面的“改进查询功能”一节。

使用 Oracle 的 SQL 跟踪工具

EXPLAIN PLAN 产生的信息非常有用，但这些信息只说明了一半的内容。知道语句的执行计划是一件事情，但合理地调试查询需要关联具有资源用途的执行计划，而且 Oracle 嵌入式跟踪工具允许该行为。使用 SQL 跟踪（SQL Trace），可以显示被执行的 SQL 语句所消耗的资源统计信息。SQL Trace 记录了执行每条 SQL 语句时的下列信息:

- 被执行语句的数量
- 语句被分析的次数
- 返回的行数
- 物理读的行数
- 读的逻辑号
- 执行语句所消耗的时间
- 执行语句时占用 CPU 的时间

这些信息的价值在于两点原因:

1. 允许大家标识出消耗很多资源的语句，这些语句就是要调试的语句。
2. 提供了具体资料测量调试的效果。可以看到这些效果在执行计划的物理I/O计数和CPU时间上立即得到了改变。

从更高层看，使用SQL Trace的处理看起来有点像：

1. 设置初始化文件关键参数，最显著参数的是TIMED_STATISTICS。
2. 跟踪数据库会话。
3. 运行被调试的SQL语句。
4. 停止跟踪数据库会话。
5. 找到跟踪文件。
6. 使用tkprof应用规定跟踪文件的格式。
7. 回顾tkprof输出。

大家可以根据需要的频率重复执行这些步骤，继续产生变化，直到得到可以接受的性能为止。

注意：SQL TRACE自身并不是一个程序。它是建立在Oracle服务器组件上的容器。被SQL Trace产生的文件并不可读，所以Oracle提供了tkprof应用程序规定它们的格式。tkprof总结了跟踪信息，并将它用可用的格式表达出来。

设置参数

在运行SQL Trace之前，需要检查3个初始化参数：

TIMED_STATISTICS
MAX_DUMP_FILE_SIZE
USER_DUMP_DEST

TIMED_STATISTICS

为了让SQL Trace能收集正在执行的语句的任何定时信息，需要将TIMED_STATI-

STICS参数设置为TRUE。大家可以在实例参数*INIT.ORA*文件中设置这一参数，也可以使用ALTER SYSTEM或ALTER SESSION语句进行设置。例如:

```
ALTER SYSTEM SET TIMED_STATISTICS = TRUE;
ALTER SESSION SET TIMED_STATISTICS = TRUE;
```

如果试图跟踪连接到数据库的程序创建的会话，那么在会话层设置定时统计功能就有点困难。用户需要修改程序以解决必要的ALTER SESSION语句的问题。如果做不到，就只能用ALTER SYSTEM语句在实例层打开定时统计。因为收集定时统计时的过量开销相当小，所以不需要考虑该问题。如果需要的话，可以使实例能进行定时统计。

MAX_DUMP_FILE_SIZE

MAX_DUMP_FILE_SIZE参数限制了Oracle产生的跟踪文件的大小。跟踪文件可以在很短的时间内变得很大，在短时间内运行一个执行大量SQL语句的程序时，该问题尤其突出。通常情况下，大小限制由操作系统块指定，而默认限制则由操作系统指定。但是，大家也可以使用MAX_DUMP_FILE_SIZE参数指定。例如:

```
ALTER SYSTEM SET MAX_DUMP_FILE_SIZE = 100;
ALTER SESSION SET MAX_DUMP_FILE_SIZE = 100;
MAX_DUMP_FILE_SIZE = 100;
```

开始的两个语句分别设置实例和会话层的MAX_DUMP_FILE_SIZE值。第3行显示大家应该在实例参数文件中怎样指定限制。此时，设置大小限制在100个操作系统块上。

USER_DUMP_DEST

USER_DUMP_DEST参数控制在哪里编写跟踪文件，应该指出系统上的目录。该参数只能在实例层而不能在会话层使用ALTER SYSTEM语句进行修改。

设置能否进行SQL Trace

可以为整个实例或单个会话定义是否能够进行SQL Trace。为实例打开这一功能是一种快捷方法:指定每个人连接的实例而产生很多的跟踪文件通常不是一个好方法。

对指定会话打开SQL Trace的最常用的方法如下:

1. 运行想要测试的程序，并使用它建立会话。
2. 使用SQL*Plus登录，并使用DBMS_SYSTEM嵌入式包控制被测试程序执行的语句的跟踪信息。

使实例能进行SQL Trace

要使实例能进行SQL Trace、在实例参数文件中需要放置如下条目，然后关闭并重新启动实例:

```
SQL_TRACE = TRUE
```

当能在实例层进行跟踪时，每一个会话连接的实例将产生一个跟踪文件，该跟踪文件为每一个会话收集跟踪信息，直到会话跟踪文件的大小超过MAX_DUMP_FILE_SIZE的限制为止。当会话的跟踪文件达到最大时，就停止对该会话的跟踪。

使会话能进行SQL Trace

不能用ALTER SYSTEM命令改变整个实例的SQL_TRACE设置，但可以使用ALTER SESSION语句改变单个会话的SQL_TRACE设置。例如:

```
ALTER SESSION SET SQL_TRACE=TRUE;
ALTER SESSION SET SQL_TRACE=FALSE;
```

如果想在当前会话中收集被执行SQL语句的跟踪信息，可以使用该方法。当SQL_TRACE等于TRUE时，将执行所有的SQL语句，而且会在会话的跟踪文件中编写跟踪信息。结束跟踪时，只要简单地执行ALTER SESSION语句设置SQL_TRACE为FALSE就可以了。如果碰到跟踪文件的大小限制时，系统也会自动停止跟踪。

使另一个会话能进行SQL Trace

DBMS_SYSTEM嵌入式包包含名为SET_SQL_TRACE_IN_SESSION的过程，该过程可以用于开启和关闭SQL TRACE，从而有效控制连接到数据库上的会话。使

用该方法可以为另一个会话收集跟踪信息。DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION 过程采用 3 个参数，正式定义形式如下：

```
DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION (sid IN NUMBER,
                                      serial# IN NUMBER,
                                      sql_trace IN BOOLEAN);
```

V$SESSION 视图可以查询用户列表，包括其 sid 和 serial# 号，并可以在 SQL_TRACE_IN_SESSION 中使用这些值确定能否对特定用户进行跟踪。下面的例子决定能否对用户 jeff 进行跟踪：

```
SQL> SELECT username, sid, serial#
  2  FROM v$session;

USERNAME                            SID   SERIAL#
------------------------------ --------- ---------
...
JEFF                                 8         31
SYS                                 11         10

9 rows selected.

SQL> EXECUTE DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION(8,31,TRUE);

PL/SQL procedure successfully completed.

SQL> EXECUTE DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION(8,31,FALSE);

PL/SQL procedure successfully completed.
```

在这个例子中，我们查询 V$SESSION 得到用户清单。找到大家感兴趣的 jeff 用户，具有 sid 8 和 serial# 31。这些值随后用在 DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION 的两个调用中。第一个调用确定开始跟踪，第二个调用确定停止跟踪。如果达到了跟踪文件的大小，系统将自动停止跟踪。

注意：有时，会有多个会话使用相同的用户名。如果遇到这种情况，要具体指出想要跟踪哪个会话就很困难了。可以采用 V$SESSION 查询技术，然后启动想要跟踪的会话，就可以了。这些步骤通常包括运行连接到数据库的程序，然后再次查询 V$SESSION。如果在第二个查询中包括了 LOGON_TIME 列，那么通常就非常容易挑选出刚才启动的会话。

找到跟踪文件

产生跟踪文件之后，要想找到跟踪文件，尚有一些难度。虽然跟踪文件被写入到USER_DUMP_DEST目录，但它们的名字则由系统产生（例如*ora_815.trc*）。跟踪特定用户会话时，要找到相关跟踪文件，其较好方法是列出USER_DUMP_DEST目录下的所有文件并查看它们的时间戳。时间戳将反映该文件上一次修改的时间，这与跟踪停止的时间相关。记住，达到MAX_DUMP_FILE_SIZE指定的值时，可以停止跟踪。如果达到最大文件大小，则时间戳将反映什么时候该事件发生，否则，时间戳将隐含反映不能进行跟踪的时间点，所以设置能否进行跟踪时，要关注时间。

注意：在Linux和Unix系统上，文件名号代表了创建文件的操作系统进程的ID号。有时，在查找正确的文件时该ID号非常有用。另一个方法是使用如grep这样的应用，从而为特定SQL语句查找对应的跟踪文件。

使用tkprof规定跟踪输出的格式

一旦确定了跟踪文件的位置，就可以应用tkprof将跟踪资料的摘要信息写成有用的格式中。tkprof能对结果排序，从而可以基于使用的CPU或I/O数量对查询进行排序，而且能对跟踪文件中的每条SQL语句自动执行EXPLAIN PLAN语句。

tkprof工具采用命令驱动，这意味着该信息使用命令行参数予以传递。语法如下：

```
tkprof tracefile outputfile
       [EXPLAIN=username/password]
       [TABLE=[schema.]tablename]
       [PRINT=integer]
       [AGGREGATE={YES | NO}]
       [INSERT=filename]
       [SYS={YES | NO}]
       [RECORD={YES | NO}]
       [SORT=sort_options
```

关键字

tracefile

该关键字决定跟踪文件的名字。

outputfile

该关键字决定 tkprof 编写格式化的跟踪输出的文件名。

EXPLAIN

该关键字使tkprof为跟踪文件中的每条SQL语句产生执行计划，这是通过作为指定用户连接并发出EXPLAIN PLAN语句来实现的。通常，在这里使用的用户名，应该与第一个会话创建跟踪文件时使用的用户名相同。tkprof工具将使用PROF$PLAN_TABLE表，保存EXPLAIN PLAN语句的输出，并在必要时创建该表。

TABLE

该关键字允许指定执行EXPLAIN PLAN语句时使用的计划表名，默认为plan_table，而且默认的架构是EXPLAIN参数指定的一个用户、当有多个用户同时执行tkprof而且使用EXPLAIN参数的用户名相同时使用该参数。如果所有用户都试图使用相同计划表，那么多个用户使用tkprof将会产生冲突。

PRINT

告诉tkprof只给跟踪文件中的第一个整型的SQL语句产生输出。

AGGREGATE

该关键字控制tkprof是否汇总相同SQL语句多个执行的结果，默认是YES。因此，如果大家执行了相同语句1000次，tkprof将从1000次执行中汇总统计信息。如果设置AGGREGATE等于NO，则会单独报告每个语句执行的结果信息。通常，采用默认值并查看汇总格式的结果可以带来很多方便。

INSERT

该关键字使tkprof产生一个INSERT语句文件，该文件随后用来保存数据库表的跟踪信息。该脚本文件也包含CREATE TABLE语句，创建被INSERT语句引用的表。

SYS

该关键字显示是否想在输出文件中包括递归SQL语句。递归SQL语句是由Oracle分析和执行用户发出的语句之后产生的语句。例如，要分析SELECT语句，Oracle可能需要查询资料目录，那些查询将被当作递归SQL语句。默认情况包括递归SQL语句。

RECORD

该关键字使tkprof产生由跟踪文件中所有用户发出的SQL语句组成的SQL脚

本。使用这里指定的文件名对脚本文件命名，可以在随后使用该脚本重现该跟踪会话，也可以选择性地包括路径和扩展名。

SORT

该关键字使 tkprof 根据指定的 *sort_options* 对输出文件进行排序。表 8-3 描述了可用的选项。将它们放置在逗号分隔的列表中可以指定多个选项。

当对 tkprof 输出排序时，一些选项是可用的。因为排序允许大家将执行最糟糕的语句移到顶部，而且不需要读取整个文件就能查找到最需要调试的语句，所以这种排序非常有用。表 8-3 列出的排序选项允许大家根据性能均衡选择排序。

表 8-3: tkprof 排序选项

描述 / 阶段	分析	执行	提取	其他
语句分析、执行或提取的时间	prscnt	execnt	fchcnt	N/A
分析、执行或提取所花费的 CPU 时间	prscpu	execpu	fchcpu	N/A
分析、执行或提取所消耗的时间	prsela	exeela	fchela	N/A
分析、执行或提取时在磁盘上进行物理读	prsdsk	exedsk	fchdsk	N/A
分析、执行或提取时访问相容读时的缓存数	prsqry	exeqry	fchqry	N/A
分析、执行或提取时访问当前读时的缓存数	prscu	execu	fchcu	N/A
分析或执行时丢失的高速缓存数	prsmis	exemis	N/A	N/A
提取的总行数	N/A	N/A	fchrow	N/A
用户分析该语句的用户标识	N/A	N/A	N/A	userid

tkprof 示例

在 Unix 环境下执行下面的命令:

```
$ tkprof ora_718.trc ora_718.lst SORT=execpu EXPLAIN=scott/tiger

TKPROF: Release 8.1.5.0.2 - Production on Tue Mar 14 17:46:22 2000

(c) Copyright 1999 Oracle Corporation.  All rights reserved.
$
```

该命令从跟踪文件 *ora_718.trc* 中读取，并将规定格式的输出编写到 *ora_718.lst* 文件中。跟踪文件结果通过执行语句的CPU时间进行排序；首先列出执行中消耗CPU

时间最多的语句。EXPLAIN参数使tkprof在遇到的每一条语句上都执行EXPLAIN PLAN。要完成这一任务，需要将tkprof作为用户scott登录到数据库中。

警告：在使用tkprof EXPLAIN参数产生跟踪文件中语句的执行计划时，有一个重要的注意事项。当语句正在被执行时，系统将收集跟踪文件中执行过程的统计信息，但tkprof产生的执行计划只有在运行tkprof应用时才能产生。在跟踪和规定结果的格式之间的时间内，事情都可以发生改变，而且tkprof产生的执行计划，并不是实际执行语句时使用的计划。要防止这一问题，需要在产生跟踪文件之后尽可能运行tkprof，同时要避免创建索引、分析表或做其他影响跟踪语句的执行计划的事情。

解释 tkprof Output

tkprof output文件是一个文本文件，包含在跟踪文件中找到的每一条SQL语句的信息。对每一个SQL语句，该文件将提供下面的信息：

- SQL语句的自身文本
- 有关执行该SQL语句的统计信息
- 库缓存上丢失的信息和优化器目标
- 语句的执行计划

如果对本章开始提供的SQL语句进行跟踪（就是在scott架构中查询3张表并产生员工薪水层次的报表的那个例子），并想让tkprop指定结果的格式，就要重新调整该语句的输出：

```
SELECT e.empno, e.ename, d.dname, e.sal, sg.grade
FROM emp e, dept d, salgrade sg
WHERE e.deptno = d.deptno
AND e.sal > sg.losal
AND e.sal < sg.hisal
ORDER BY sal DESC

call     count       cpu    elapsed       disk      query    current        rows
------ ------  -------  ---------  ---------  ---------  ----------  -------
Parse        1      0.03       0.04          9        149           0        0
Execute      1      0.00       0.00          0          0           0        0
Fetch        2      0.02       0.03          6          3          12       12
------ ------  -------  ---------  ---------  ---------  ----------  -------
total        4      0.05       0.07         15        152          12       12
```

```
Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 20  (SCOTT)

Rows     Row Source Operation
-------  ---------------------------------------------------
     12  SORT ORDER BY
     12   HASH JOIN
     20    MERGE JOIN CARTESIAN
      5     TABLE ACCESS FULL DEPT
     20     SORT JOIN
      5      TABLE ACCESS FULL SALGRADE
     14    TABLE ACCESS FULL EMP


Rows     Execution Plan
-------  ---------------------------------------------------
      0  SELECT STATEMENT   GOAL: CHOOSE
     12   SORT (ORDER BY)
     12    HASH JOIN
     20     MERGE JOIN (CARTESIAN)
      5      TABLE ACCESS   GOAL: ANALYZED (FULL) OF 'DEPT'
     20      SORT (JOIN)
      5       TABLE ACCESS   GOAL: ANALYZED (FULL) OF 'SALGRADE'
     14     TABLE ACCESS   GOAL: ANALYZED (FULL) OF 'EMP'
```

时间统计信息对确定有性能问题的语句非常有用。一旦确定了有问题的语句，就可以回顾执行计划，决定是否有更好的方法可以完成该语句执行的功能。

语句执行的分析

SQL 语句可以在下列 3 个阶段执行：

分析阶段

在这一阶段，Oracle采用原始的SQL语句检查语法，决定包括哪些对象，并检查这些对象上的安全性，然后产生该语句的执行计划。Oracle 将执行计划和 SQL 语句的分析内容存储在共享池中。理想情况下，每个 SQL 语句应该只需要分析一次，随后可以执行多次。

执行阶段

在该阶段，Oracle完成执行语句的实际工作。对INSERT、UPDATE和DELETE语句，执行语句代表了工作量。对 SELECT 语句，执行阶段是 Oracle 找到查询结果返回的行的时间长度。任何排序和分组都在这一阶段完成。

提取阶段

该阶段只应用于 SELECT 语句。在这一阶段，Oracle 给执行 SELECT 语句的应用程序返回行。

SQL Trace 捕获这 3 个阶段的每一阶段的统计信息。tkprof 工具报告了这些统计信息并报告了所有这 3 个阶段的整个组合信息。每个阶段的统计信息包含的项目显示在表 8-4 中。

表 8-4: 统计描述

统计	描述
COUNT	报告语句分析或执行的时间数。在SELECT语句的情况下，也反映了应用程序提取数据的提取数
CPU	报告了在 3 个阶段中每一阶段使用的 CPU 时间数
ELAPSED	报告了在 3 个阶段中每一阶段消耗的时间数
DISK	报告了在 3 个阶段中每个阶段从磁盘上进行物理读的数据库块数量
QUERY	报告了当前模式下重新得到的缓存数。当问题中的 SQL 语句是一个 SELECT 语句时，通常当前缓存采用兼容模式提取数据
CURRENT	报告当前模式下重新得到的缓存数。在执行 INSERT、UPDATE 或 DELETE 语句时，通常在当前模式下重新得到缓存

tkprof 报告中查找的关键元素

在跟踪文件中查找的内容依赖于为什么要在首要位置运行跟踪而决定，但这里有一些建议，需要大家注意：

- 高 CPU 或花费的时间
- 分析明显大于 1 的记数
- 大量的库缓存丢失
- 物理磁盘的高比率读与当前缓存提取兼容

如果运行跟踪是为了响应用户的意见，那么这些意见通常是针对运行某种工作或执行某种任务需要的时间。因此，在运行工作或执行任务时进行跟踪就很合理了，而且这些查询通常代表了可以改进性能的最大机会。

注意：要清楚高CPU时间或开销时间并不一直是一件坏事，大家需要将语句执行过程与执行时间量一并考虑，同时考虑单个执行的时间是否合理完成了语句需要完成的任务。

理想情况下，只应该对语句分析一次。随后，语句的分析内容和执行计划都应该保留在共享池中，以备下一次执行该语句时使用。分析的代价非常昂贵，所以如果分析计数与执行计数密切相关，就需要扩展共享池的大小。

整个库缓存的丢失量大（不一定是单个语句）也显示了共享池可能太小了。Oracle在共享池中缓存了数据目录信息，库缓存丢失代表了Oracle不能在共享池中找到需要的数据目录信息，而不得不花费时间从磁盘上进行读取。

大量的物理读与兼容读或当前读相关，显示正在进行完全表扫描。检查执行计划，可以证实这一点。

注意：整表扫描并不总是很糟糕。确信这些扫描刚好是有意安排的，而不是因为大家忘记创建索引而造成的。

SQL*Plus 调试助手

测试和调试SQL语句时，SQL*Plus有两个功能非常有用。其一是，SET AUTOTRACE命令可用于将SQL*Plus放入到一种模式下，在该模式下，系统将显示每一条被执行的SQL语句的执行计划和统计信息；其二是，SQL*Plus命令也可跟踪执行一条或多条SQL语句时必需占用的时间。

SET AUTOTRACE

SET AUTOTRACE命令引起SQL*Plus显示每一条被执行的语句的执行计划和下列统计信息：

- 被执行的递归SQL语句数量
- 当前模式中提取的块数量（得到db块）
- 读兼容模式中提取块的数量

- 从磁盘上物理读的数量
- 使用归档日志的数量
- 通过 Net8 传输到客户端的执行语句数量
- 在客户和服务器之间交换的 Net8 信息数量（来回）
- 内存排序数量
- 使用磁盘执行排序的数量
- 处理行的整个数量

这些统计信息有一些与提供的SQL Trace不同，而另一些则相同。与SQL Trace产生的统计信息不同的是，从SET AUTOTRACE产生的统计信息不分为分析、执行和提取阶段。

使用自动跟踪的先决条件

在使用SET AUTOTRACE之前需要满足一些先决条件。如果需要执行某个计划，那么在架构中必须存在计划表，而且计划表的名字必须是plan_table。在允许进行自动跟踪的情况下，SQL*Plus将自动使用该表。创建计划表的处理过程在本章中的“创建计划表”一节中已经描述过。

如果想产生自动跟踪统计信息，那么产生统计的用户一定要访问SQL*Plus用于得到统计信息的动态性能视图。这种访问通常应该是使用角色PLUSTRACE进行授权。Oracle提供了名为*plustrce.sql*的脚本创建该角色。脚本放置在Oracle主目录下的*sqlplus/admin*目录中。

注意：在Windows NT的早期Oralce版本中查找叫做plus80或plus73的目录。

要创建PLUSTRACE角色，需要采用用户SYS登录（该登录通常由DBA完成），并执行*plustrce.sql*脚本，如下面的例子所示：

```
SQL> CONNECT sys/change_on_install
Connected.
SQL> @$ORACLE_HOME/sqlplus/admin/plustrce
```

```
SQL>
SQL> DROP ROLE PLUSTRACE;
drop role plustrace
         *
ERROR at line 1:
ORA-01919: role 'PLUSTRACE' does not exist

SQL> CREATE ROLE PLUSTRACE;

Role created.

SQL>
SQL> GRANT SELECT ON v_$sesstat TO PLUSTRACE;

Grant succeeded.

SQL> GRANT SELECT ON v_$statname TO PLUSTRACE;

Grant succeeded.

SQL> GRANT SELECT ON v_$session TO PLUSTRACE;

Grant succeeded.

SQL> GRANT PLUSTRACE TO DBA WITH ADMIN OPTION;

Grant succeeded.

SQL>
SQL> SET ECHO OFF
```

注意：不要担心在前面的例子中显示的错误（如果以前还没有创建该角色），常常会出现这样的错误。在创建角色之前，Oracle试图删除该角色（以免已经存在该角色），当没有要被删除的角色时，就产生该错误结果。

一旦执行该脚本，就可以在数据库中看到一个名为PLUSTRACE的角色，而且可以将该角色授权给需要使用SET AUTOTRACE的用户。

显示统计和计划

要看一看被执行语句的统计信息和执行计划，必须使用SET AUTOTRACE ON命令，然后大家可以执行任何SQL语句进行理想测试。

示例

```
SQL> SET AUTOTRACE ON
SQL> SELECT e.empno, e.ename, d.dname, e.sal, sg.grade
  2  FROM emp e, dept d, salgrade sg
  3  WHERE e.deptno = d.deptno
  4  AND e.sal > sg.losal
  5  AND e.sal < sg.hisal
  6  ORDER BY sal DESC;

     EMPNO ENAME      DNAME                 SAL      GRADE
---------- ---------- -------------- --------- ---------
      7839 KING       ACCOUNTING          5000         5
      7566 JONES      RESEARCH            2975         4
      7698 BLAKE      SALES               2850         4
      7782 CLARK      ACCOUNTING          2450         4
      7499 ALLEN      SALES               1600         3
      7844 TURNER     SALES               1500         3
      7934 MILLER     ACCOUNTING          1300         2
      7521 WARD       SALES               1250         2
      7654 MARTIN     SALES               1250         2
      7876 ADAMS      RESEARCH            1100         1
      7900 JAMES      SALES                950         1
      7369 SMITH      RESEARCH             800         1

12 rows selected.

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=9 Card=1 Bytes=75)
   1    0   SORT (ORDER BY) (Cost=9 Card=1 Bytes=75)
   2    1     HASH JOIN (Cost=7 Card=1 Bytes=75)
   3    2       MERGE JOIN (CARTESIAN) (Cost=5 Card=20 Bytes=700)
   4    3         TABLE ACCESS (FULL) OF 'DEPT' (Cost=1 Card=4 Bytes=8
          8)
   5    3         SORT (JOIN) (Cost=4 Card=5 Bytes=65)
   6    5           TABLE ACCESS (FULL) OF 'SALGRADE' (Cost=1 Card=5 B
          ytes=65)
   7    2       TABLE ACCESS (FULL) OF 'EMP' (Cost=1 Card=14 Bytes=560
          )

Statistics
----------------------------------------------------------
        922  recursive calls
         13  db block gets
        152  consistent gets
         16  physical reads
          0  redo size
       1689  bytes sent via SQL*Net to client
        809  bytes received via SQL*Net from client
          4  SQL*Net roundtrips to/from client
         17  sorts (memory)
          0  sorts (disk)
```

```
          12  rows processed
```

测试完成时，执行 SET AUTOTRACE OFF 命令关闭自动跟踪功能。

只显示计划或统计信息

自动跟踪的输出可以限制为只显示执行计划或只显示统计信息。要使自动跟踪只显示执行计划，可以使用下列命令允许自动跟踪：

```
SET AUTOTRACE ON EXPLAIN
```

要使自动跟踪只显示统计信息，可以使用下列命令：

```
SET AUTOTRACE ON STATISTICS
```

注意：当大家以限制方式运行自动跟踪时，不一定要满足所有先决条件。如果大家不显示执行计划则不需要计划表，而且如果不显示统计信息则不需要 PLUSTRACE 角色。

禁止查询输出

如果大家想看一看查询的自动跟踪信息但不查看查询结果，可以使用：

```
SET AUTOTRACE TRACEONLY
```

TRACEONLY 关键字代替了 ON，所以下面的命令也是有效的：

```
SET AUTOTRACE TRACEONLY EXPLAIN
SET AUTOTRACE TRACEONLY STATISTICS
```

TRACEONLY选项可以节省滚动屏幕查看很多资料的时间，但要认识到，SQL语句仍然被执行了。事实上，查询的结果甚至被传递回客户端。TRACEONLY的惟一差别是 SQL*Plus 不显示查询结果。

定时器

SQL*Plus具有两个嵌入式工具可以测量执行SQL语句时占用的时间。SET TIMING 命令函数与 SET AUTOTRACE 相似，但报告了每一条语句之后消耗的时间。TIMING 命令允许在任何时候启动和停止定时器。

SET TIMING

SET TIMING ON命令引起SQL*Plus为每一条被执行的SQL语句报告消耗的时间，如下面的例子所示。

SET TIMING OFF命令用于关闭定时功能：

```
SQL> SET TIMING ON
SQL> SELECT e.empno, e.ename, d.dname, e.sal, sg.grade
  2  FROM emp e, dept d, salgrade sg
  3  WHERE e.deptno = d.deptno
  4  AND e.sal > sg.losal
  5  AND e.sal < sg.hisal
  6  ORDER BY sal DESC;

     EMPNO ENAME      DNAME                SAL      GRADE
---------- ---------- -------------- --------- ---------
      7839 KING       ACCOUNTING          5000         5
      7566 JONES      RESEARCH            2975         4
      7698 BLAKE      SALES               2850         4
      7782 CLARK      ACCOUNTING          2450         4
      7499 ALLEN      SALES               1600         3
      7844 TURNER     SALES               1500         3
      7934 MILLER     ACCOUNTING          1300         2
      7521 WARD       SALES               1250         2
      7654 MARTIN     SALES               1250         2
      7876 ADAMS      RESEARCH            1100         1
      7900 JAMES      SALES                950         1
      7369 SMITH      RESEARCH             800         1

12 rows selected.

Elapsed: 00:00:00.01
```

注意：该例子运行在Linux系统上。在Linux和Unix上，SQL*Plus以小时、分钟、秒和百分之一秒的形式报告了占用的时间。在Windows环境下，以毫秒作为单位。

TIMING

SQL*Plus的TIMING命令可以用于任意启动和停止定时器。使用TIMING START命令启动定时器并使用TIMING STOP命令停止定时器。当停止定时器时，显示占用的时间。例如：

```
SQL> TIMING START
```

```
SQL>
SQL> TIMING STOP
Elapsed: 00:00:05.15
```

正常的SQL语句将运行在两个TIMING命令之间，所以这样定时就非常有意义。

要显示时间而不停止定时器，可以使用TIMING SHOW命令：

```
SQL> TIMING START
SQL> TIMING SHOW
Elapsed: 00:00:03.40
SQL> TIMING SHOW
Elapsed: 00:00:06.93
```

通过执行多条TIMING命令，可以同时运行多个定时器，如下所示：

```
SQL> TIMING START
SQL> TIMING START
SQL> TIMING STOP
Elapsed: 00:00:05.46
SQL> TIMING STOP
Elapsed: 00:00:15.66
```

当使用多个定时器时，它们以启动时的相反顺序停止。认为每个定时器嵌套在另一个定时器之中。要停止最外部的定时器，必须停止所有内部的定时器。

改进查询性能

如果查询性能不能令人满意，记住可以使用下面的方法予以改进。大家可以：

- 修改数据库结构
- 收集最新的统计信息
- 重写查询
- 使用优化器提示控制执行计划

修改数据库结构

数据库结构修改通常包括索引的创建。提供合适的索引来支持查询是非常重要的，但索引并不完全是性能差的原因。其他结构变化，包括将表数据和索引数据分离到

不同的独立磁盘中、将数据扩散到多个磁盘上并形成分区数据和簇数据。但是，如果在开始查询时情况就不太理想，那么这些变化也不可能引起性能发生太大的变化。

收集统计信息

如果使用成本优化器，那么表和索引将充当主要角色，决定Oracle为给定语句使用相应的执行计划。一系列的统计信息将产生性能较好的执行计划，这一点很重要。理论上，及时统计使数据精确会得到最好的执行计划。实际上，对表分析完成之后，偶尔的性能下降已经是经验方面的问题。Oracle8*i* 实际包括一个功能，该功能允许导入任意统计信息充当优化器产生执行计划的基础。

重写查询

重写查询有时在性能上有重要影响。考虑下面的查询，提取不在波士顿、纽约和芝加哥居住的员工列表：

```
SELECT empno, ename
FROM emp, dept
WHERE emp.deptno = dept.deptno
AND dept.loc NOT IN ('NEW YORK', 'BOSTON', 'CHICAGO');
```

该执行计划看起来有点像：

```
SELECT STATEMENT   Cost = 3
  NESTED LOOPS
    TABLE ACCESS FULL DEPT
    TABLE ACCESS FULL EMP
```

注意，估计查询的成本是3。可以使用NOT EXIST谓词重写查询，代替连接emp和dept表。这里是查询的语句：

```
SELECT empno, ename
FROM emp
WHERE NOT EXISTS (
  SELECT *
  FROM dept
  WHERE emp.deptno = dept.deptno
  AND dept.loc IN ('NEW YORK', 'BOSTON', 'CHICAGO'));
```

该版本的查询执行计划如下：

```
SELECT STATEMENT   Cost = 1
  FILTER
    TABLE ACCESS FULL EMP
    TABLE ACCESS BY INDEX ROWID DEPT
      INDEX UNIQUE SCAN PK_DEPT
```

第二版查询的成本为1，与第一个查询的成本3相反，第二个查询可能是更好的选择。但是大家不能完全依赖优化器的成本估计，因为这只是估计！要确信哪一个查询执行最好，要进行试验，从而得到真实的性能。大家可以为该目的使用SQL Trace，这样就能比较这两种方式下的CPU时间、占用的时间和I/O的占用比。

使用优化器提示（hint）

使用优化器提示也可以明显改善性能，因为它们允许修改执行计划，并在数据库实际数据的基础上进行调试。

这里不是允许Oracle整个控制怎样执行查询，而且使用提示给优化器提供特定的方向。在Oracle中，hint作为组件优化器直接嵌入在SQL语句中。例如：

```
SELECT /*+ FULL(dept) */
       empno, ename, dname
FROM emp, dept
WHERE emp.deptno = dept.deptno;
```

这种情况下的提示是FULL(dept)、它告诉Oracle要完成对dept表的整表扫描。Oracle将遵从该提示并在连接表时执行dept的整表扫描，即使这样做会降低性能也会这样做。

注意：只有在物理或逻辑上不可能这样实现的惟一场合，Oracle才会不遵从提示的意见。

hint语法

提示应用于单个SQL语句，而且只可以为SELECT、INSERT、UPDATE和DELETE语句指定hint。hint采用特殊格式的注释形式，而且必须紧随在该语句关键字开始之后：

```
keyword /*+ [hint|comment...] */
```

keyword

SQL 命令，必须是 SELECT、UPDATE 或 DELETE。提示：必须立即跟随在该语句的关键字开始之后。

hint

提示本身，有时是一个或多个放在括号中的参数。表8-5到表8-10提供了可能的提示的完整列表。提示不区分大小写。单个注释可以包含不止一个提示，只要提示之间至少采用空格分离就可以。

comment

用户定义的注释。Oracle 允许在提示中间分散注释。

注意，在下面的第二个例子中的表名是emp，但是已经指定了一个别名e。表的hint使用相同的别名，而且指定为FULL(e)。记住，无论什么时候使用别名，任何hint中都必须使用该表的别名：

```
SELECT /*+ FULL(emp) */ empno, ename
  FROM emp
 WHERE sal < 4000;

SELECT /*+ FULL(e) do a full tablescan on the emp table, because
                   most employees do have billing rates < 4000. */
       empno, ename
  FROM emp e
 WHERE sal < 4000;
```

如果为一条语句提供了多条hint，则它们必须出现在相同的注释中，如下所示：

```
SELECT /*+ FULL(emp) FIRST_ROWS */ empno, ename
  FROM emp
 WHERE sal < 4000;
```

在使用子查询时，允许它们具有自己的提示。子查询的提示紧随在启动查询的关键字之后，如下所示：

```
SELECT /*+ FIRST_ROWS */ empno, ename
FROM emp
WHERE NOT EXISTS (
  SELECT /*+ FULL(dept) */ *
  FROM dept
  WHERE emp.deptno = dept.deptno
  AND dept.loc IN ('NEW YORK', 'BOSTON', 'CHICAGO'));
```

注意：当使用提示时，要特别注意使用正确的语法。因为提示作为注释嵌入在语句中，所以Oracle不可能进行任何语法检查。Oracle将任何不正确的提示当作注释。另外，大家应该始终在编写提示的代码之后执行EXPLAIN PLAN，确保优化器实际完成大家想要完成的任务。

在尽可能的情况下，Oracle总是遵从提示

只要物理上和逻辑上可行，则系统总会遵从提示。如果一个提示与另一个提示冲突，或者不可能应用提示，则Oralce将简单地忽略这些提示。在下面的例子中，USE_CONCAT提示没有意义，因为查询不包含OR条件，该查询不能区分成两个查询（具有随后联结在一起的结果），所以Oracle将忽略该提示。但是，只要可能实现，都会遵从一个坏的提示：

```
SELECT /*+ USE_CONCAT */
       empno, ename
  FROM emp
 WHERE ename = 'SCOTT';
```

下一个查询中包含了一个hint，该hint完成emp表的主键索引上的索引扫描。emp的主键是empno列，而且随之在该列上有一个索引。该查询在员工姓名的基础上查找一条记录：

```
SELECT /*+ INDEX(emp pk_emp) */
       ename
  FROM emp
 WHERE ename = 'SCOTT';
```

尽管使用empno上的索引提取员工的姓名没有意义，但是Oracle将遵从该查询使用的主键索引并采用下面的执行计划：

```
SELECT STATEMENT   Cost = 2
  TABLE ACCESS BY INDEX ROWID EMP
    INDEX FULL SCAN PK_EMP
```

Oracle将读取主键索引上的每一条目，从员工表（employee）上提取相关行，而且检查姓名是否匹配。虽然这样比进行整表扫描要差，但是Oracle偏偏应用该计划，因为提示需要它，而且以为它是物理上可行的。

提示的类型

Oracle 提示大致可以分成以下几类:

- 优化器目标提示（Optimizer goal hints）
- 访问方法提示（Access method hints）
- 连接顺序提示（Join order hints）
- 联接操作提示（Join operation hints）
- 并行执行提示（Parallel execution hints）
- 其他提示（Other hints）

下面的部分描述了在每一类中可用的提示。

优化器目标提示

在阐明执行计划时，优化器目标提示可以影响优化器的整个目标。例如，提示可以指定该计划应该优化为尽可能快地返回第一条记录。表 8-5 列出了这些提示。

表 8-5: 优化器目标提示

Hint	描述
ALL_ROWS	告诉优化器产生消耗资源最小的执行计划
FIRST_ROWS	告诉优化器产生尽可能快地得到第一行的目标执行计划
CHOOSE	允许优化器在规则模式和成本模式间选择优化器。如果查询中的统计信息为表提供，则使用基于成本的方法
RULE	强制优化器为该语句使用基于规则的方法

警告: 要尽量避免使用RULE提示。该提示会导致使用规则优化器。一方面，在决定语句的执行计划时，规则优化器使用了固定集合的规则，且不影响执行计划的最后成本。另一方面，成本优化器的结论基于I/O和CPU的过量开销，这些过量开销是各种选择计划所需要的估计值。虽然Oracle仍然支持规则优化器，但在近几年中没有进一步的改进，而且在将来也不会改进，从某种观点说，以后可能不支持规则优化器，Oracle会将其开发精力放在成本优化器上。

访问方法提示

访问方法提示控制怎样访问数据。例如，在访问表时，Oracle 可以直接进行整表扫描或使用索引进行扫描，而且可以指定索引名。所有的访问方法提示至少采用一个表名作为参数，可以为查询不同表指定不同的访问方法。

一些访问方法提示与索引相关，而且允许使用索引定义。在许多情况下（如INDEX提示），可以指定或不指定索引名。下面的提示告诉Oracle在emp表上执行索引扫描，但由Oracle提取索引：

```
/*+ INDEX(emp) */
```

在期待Oracle作出正确选择或硬性将索引名编码到不理想的hint中时，该hint非常有用。例如：

```
/*+ INDEX(emp pk_emp) */
```

在这一例子中，指定了确切的索引名。也可以指定索引列表，Oracle会在该列表中进行选择。例如：

```
/*+ INDEX(emp pk emp, emp_dept_indx) */
```

在这一例子中，假定emp表上存在7个索引，但查询中只使用两个。该提示告诉Oracle使用索引扫描访问emp表，使用索引名pk_emp或emp_dept_indx。

注意：AND_EQUAL提示非常特别，因为它需要指定至少两个索引。这是因为提示使Oracle出现在两个索引扫描结果中。除非大家要扫描两个索引，否则不能这样做。

表8-6列出了访问方法提示。

表8-6：访问方法提示

提示	描述
FULL(*table_name*)	无论是否存在索引，都需要对指定的表进行整表扫描
ROWID(*table_name*)	告诉Oracle在ROWID的基础上执行指定表的扫描

表 8-6: 访问方法提示

提示	描述
CLUSTER(*table_name*)	告诉 Oracle 在指定表中进行簇扫描。如果表不是簇的话，该提示就被忽略
HASH(*table_name*)	告诉 Oracle 进行指定表的散列扫描。如果表不是簇的话，该提示就被忽略
HASH_AJ(*table_name*)	告诉Oracle进行指定表的散列反连接
INDEX(*table_name* [*index_name*...])	告诉 Oracle 通过索引扫描访问指定表。大家也可以指定使用哪一个索引；否则就由 Oracle 选择索引。你也可以指定从哪里选择索引列表，而且 Oracle 将从该列表中选择
INDEX_ASC(*table_name* [*index_name*...])	与INDEX相似，但告诉Oracle以升序扫描索引
INDEX_COMBINE(*table_name* [*index_name*...])	告诉 Oracle 使用两个索引的一些组合。大家可以指定或让 Oracle 选择使用哪个索引
INDEX_DESC(*table_name* [*index_name*...])	与 INDEX_ASC 相似，强制 Oracle 以降序扫描索引
INDEX_FFS(*table_name* [*index_name*...])	告诉Oracle进行快速完整的索引扫描
MERGE_AJ(*table_name*)	将 NOT IN 子查询转换成合并反连接
AND_EQUAL(*table_name index_name index name*...)	告诉Oracle扫描两个或多个索引并合并结果。必须至少指定两个索引名
USE_CONCAT	将具有 OR 条件的查询转换成两个或多个用 UNION ALL 连接在一起的查询

连接顺序提示

连接顺序提示允许控制 Oracle 连接表的顺序。这里只有 3 个这样的提示，如表 8-7 所示。

表 8-7: 连接顺序 Hints

Hint	描述
ORDERED	告诉 Oracle 从左到右连接这张表，以 FROM 从句中列出的顺序进行
STAR	告诉Oracle如果可能就使用星型查询执行计划。该步骤只有在至少连接3张表，而且最大的表在两张小表引用列时，具有连接索引时才能工作。两张较小的表首先被连接，然后采用嵌套循环连接从最大的表上提取需要的行
STAR_TRANSFORMATION	告诉Oracle如有可能就将查询转换成星型查询，然后为该查询使用最好的计划

连接操作提示

连接操作提示允许控制两张表的连接顺序。Oracle 在任何可能连接两张表时使用 3 个基本的方法:

合并连接（merge join）

这种类型的连接通过每张表上连接的列对行进行排序。一旦已经对两个行集完成排序，那么Oracle就读取这两个行集并连接任何匹配的行。合并连接通常比其他连接使用的资源要少，但必须在第一行返回之前对所有记录排序。为了处理排序，也需要充足的内存和临时磁盘空间。

嵌套循环连接（nested loops join）

这一类型的连接方法是许多人在思考连接表时所想到的，Oracle挑选一张表作为驱动表，并通过该表逐行读取记录。Oracle在第二张表中查找与从驱动表中读取的每一行对应的行。因为不包括排序，所以嵌套循环连接通常会以比合并连接更快的速度取回第一条记录。因为相同的原因，嵌套循环连接也不需要大量的磁盘空间和内存。但是，嵌套循环连接可能会产生比合并连接大许多的磁盘读。

散列连接（hash join）

这一类型的连接与合并连接相似，但不需要排序。散列表建立在内存中，允许从连接表之一访问具体行，然后从其他表中读取行。当每一行都从第二张表中读时，可以应用散列函数来连接列，结果用于从第一张表中找到相应的行。

用于指定连接方法的提示和其他的提示都集中为连接操作分类。表 8-8 列出了所有的连接操作提示。

表 8-8: 连接操作提示

提示	描述
USE_NL(*table_name*)	告诉Oracle在连接该表时使用嵌套循环。由该提示指定的表是最内层循环访问的一张表。其他表是驱动表
USE_MERGE(*table_name*)	告诉 Oracle 连接该表时使用排序合并方法
USE_HASH(*table_name*)	告诉 Oracle 为指定的表使用散列连接
NO_MERGE	应用于包含一个或多个视图的连接查询。它阻止Oracle在主查询中合并视图查询
DRIVING_SITE(*table_name*)	执行分布式连接时应用该提示，这是从两个或多个数据库中连接表的一个提示。没有该提示，Oracle将选择实际从哪个数据库中收集表并进行连接。通过使用提示，就告诉Oracle，到底想在哪个数据库包含的特定表上执行连接

并行执行提示

并行执行提示影响Oracle并行处理环境中的查询方法。在单CPU上运行时，或者在不能进行并行处理时，就不可能进行并行处理，而将忽略这些提示。并行执行提示如表 8-9 所示。

表 8-9: 并行执行提示

提示	描述
PARALLEL(*table_name* [, *degree* [, *num_instances*]])	告诉 Oracle 以并行处理模式从显示的表中访问数据。大家可以选择性地指定并行度和将被包括的使用实例数量。关键字 DEFAULT 可以用于两个参数，在这一情况下，Oracle 根据 INIT.ORA 文件和表定义决定这些参数值 使用 INSERT 语句中的 PARALLELhint 自动打开 APPEND 模式。参见 APPEND 和 NO_APPEND hint
NO_PARALLEL(*table_name*)	告诉 Oracle 不以并行方式访问指定表
APPEND	只应用于 INSERT 语句。它告诉 Oracle 不要试图重新使用当前分配给表的任何扩展盘区中的可用自由空间

表 8-9: 并行执行提示（续）

提示	描述
NOAPPEND	该提示与APPEND相反，它告诉Oracle使用当前分配给表的自由空间。该提示的存在是因为，当在INSERT语句中使用PARALLEL提示时，APPEND变成了默认行为
PARALLEL_INDEX (*table_name*, *index_name* [,*degree* [,*num_instances*]])	告诉Oracle以并行处理模式扫描指定索引的显示表中的数据。该索引必须是分区索引。可以选择性地指定使用的并行度和所包括的实例数量。 关键字DEFAULT可以使用两个参数，在这种情况下，Oracle根据*INIT.ORA*文件和表的定义决定参数值

其他提示

一些提示并不完全可以归入其他分类中。表 8-10 中列出了这些提示。

表 8-10: 其他提示

提示	描述
CACHE (*table_name*)	只有在特定表上执行整表扫描时应用。它告诉Oracle在最近使用最多的高速缓存上放置该表的块，所以，只要可能，它们就保留在内存中，如果想进行小的重复访问，该提示非常有用
NOCACHE (*table_name*)	该提示与CACHE相反，让Oracle在最近使用最少的高速缓存中放置块，而且尽可能清除这些块
PUSH_SUBQ	告诉Oracle在查询执行中尽可能对没有被合并的子查询进行评估。如果想让子查询消除大量的行，那么使用该提示可以改进性能

附录

SQL 资源

在这个附录中，列出了一些有用的资源，包括在线和离线的，可以使大家了解SQL、SQL*Plus 和 PL/SQL。希望这些内容对大家能有所帮助。

书

当今市场上有许多优秀的介绍Oracle方面的书，这一部分列出了作者找到的对大家特别有帮助的一些书：

Aronoff、Eyal、Kevin Loney和Noorali Sonawalla编著的《Advanced Oracle Tuning and Administration》。Osborne McGraw-Hill, 1997 年发行。本书包括了有关 EXPLAIN PLAN 工具最全面的讨论和解释。

Feuerstein、Steven 和 Bill Pribyl 编著的《Oracle PL/SQL Programming, Second Edition》。O'Reilly & Associates, 1997 年发行。本书是关于 PL/SQL 最方便、最具权威的指南。如果要了解PL/SQL 方面的内容（贯穿 Oracle8），就在本书中。

Feuerstein、Steven 编著的《Oracle PL/SQL Programming: Guide to Oracle8*i* Features》。O'Reilly & Associates，1999 年发行。该书包含了在 Oracle8*i* 中用得非常好的 PL/SQL 新特性的摘要信息。它是《Oracle PL/SQL Programming, Second Edition》的强有力补充。

Feuerstein、Steven编著的《Advanced Oracle PL/SQL Programming with Packages》。O'Reilly & Associates，1996年发行。该书提供了需要开发自己的PL/SQL包时需要的所有信息。

Feuerstein、Steven、Charles Dye和John Beresniewicz编著的《Oracle Built-in Packages》。O'Reilly & Associates，1998年发行。一旦你已经掌握了PL/SQL程序设计，那么该书就通过解释怎样用Oracle在数据库中提供的嵌入式包的杠杆作用，将你带到更高的一个层次。

Gennick、Jonathan编著的《Oracle SQL*Plus: The Definitive Guide》。O'Reilly & Associates，1999年发行。SQL*Plus所有细节的完整指南。该书对初学者和有经验的SQL程序员提供了很有价值的信息。

Kreines、David和Brian Laskey编著的《Oracle Database Administration: The Essential Reference》。O'Reilly & Associates，1999年发行。作者和我的看法有偏差，但该书具有大家需要有效管理Oracle数据库的所有信息。

Lomasky、Brian和David Kreines编著的《Oracle Scripts》。O'Reilly & Associates，1998年发行。该书中有许多实用工具和例程，可以作为日常开发和管理的优秀工具，而且可以作为学习复杂的SQL编程技术的优秀示例。

Niemiec、Richard编著的《Oracle Performance Tuning Tips & Techniques》。Osborne McGraw-Hill，1999发行。如果大家想访问一些真实的SQL专家收集的调试经验，该书就有。

其他出版物

Select

该出版物是由国际Oracle用户美国组（International Oracle Users Group-Americas，IOUG-A）产生的一个有益成员，包含了大量文章和Oracle编程列，并包含SQL相关技巧、技术和经验的文章。

Oracle Magazine

该杂志由Oracle公司出版，是一个基本市场工具，但也包含了SQL和PL/SQL开发方面的文章（通常由Oracle技术成员或有经验的开发者提供）。

许多 Oracle 用户组和特别兴趣组也出版了包含 SQL 程序员有用信息的时事杂志。

组织

国际 Oracle 用户美国组（International Oracle Users Group-Americas，IOUG-A）

401 North Michigan Avenue
Chicago, IL 60611 USA
Voice: (312) 245-1579
fax: (312) 527-6785
email: *ioug@ioug.org*

欧洲 Oracle 用户组（European Oracle Users Group，EOUG）

Brigittenauer Lande 50-54, A-1203
Vienna, Austria
Voice: +43 1 33777 870
fax: +43 1 33777 873
email:*eoug@at.oracle.com*

亚太 Oracle 用户组（Asia-Pacific Oracle Users Group，APOUG）

PO Box 3046
The Pines, Doncaster East
VIC 3109, Australia
Voice: +61 3 9842 3246
fax: +61 3 9842 3050
email: *100242.1746@compuserve.com*

澳大利亚 Oracle 用户组（Australian Oracle User Group，AUSOUG）

PO Box 16
Wilston QLD 4051 Australia
Voice: +61 7 3352 7985
fax: +61 7 3352 7135

网站

http://www.oracle.com

Oracle公司的网站。包含几乎所有的Oracle相关主题的大量信息，并链接到其他地址，包括Oracle支持网站。

http://www.ioug.org

由IOUG-A操作。包括从IOUG-A会议、技术仓库和技术论坛的杂志、论文上选择的技术文章。

http://www.eoug.com

EOUG的网站。包含Oracle开发者通常感兴趣的信息，包括欧洲、中东和非洲的会议及教育事件。

http://www.ausoug.org

AUSOUG网站。包含Oracle开发者的通常感兴趣的信息，包括澳大利亚的会议及教育事件。

http://www.revealnet.com

该网站控制PL/SQL“Pipeline”，这是一个PL/SQL主题自由在线讨论团体，吸引了世界上一批最好的和最有经验的Oracle天才，也欢迎初学者。那里人气很好，参与者可以讨论各种层次的问题。

http://technet.oracle.com

Oracle技术网络（Oracle Technology Network's，OTN）网站，由Oracle公司操作，包含了Oracle开发者的大量技术资源，可以访问所有Oracle文档集、样例代码和白皮书。

http://www.ixora.com.au

该网站明确不是为初学者或胆怯者而设计的，但它确实包含了与Oracle调试和内核的知识深度不相匹配的信息。

http://www.primarykey.com

如果大家想查找Oracle方面的书，这里就可以找到。该网站由实际懂得Oracle的人操作，而且可以帮助大家找到需要的资源。

词汇表

bind
绑定

cluster
簇

constraint
约束

cursor
游标

declaration
声明

DML(Data Manipulation Language)
数据操纵语言

exception
异常

field
字段

hash
散列

header
标题

hint
提示

identifier
标识符

index
索引

instance
实例

keyword
关键字

literal
文字

operator
操作符

optimizer
优化器

outer join
外部连接

partition
分区

primary key
主键

profile
轮廓

referential integrity
参照完整性

roll back
回退

rowset
行集

schema
架构

spool
假脱机

stored procedure
已存储过程

subpartition
子分区

tag
标签

trace
跟踪

[General Information]
书名=Oracle SQL必备参考
作者=
页数=416
SS号=11066338
出版日期=
www.chinaz.com

封面
书名
版权
前言
目录
正文