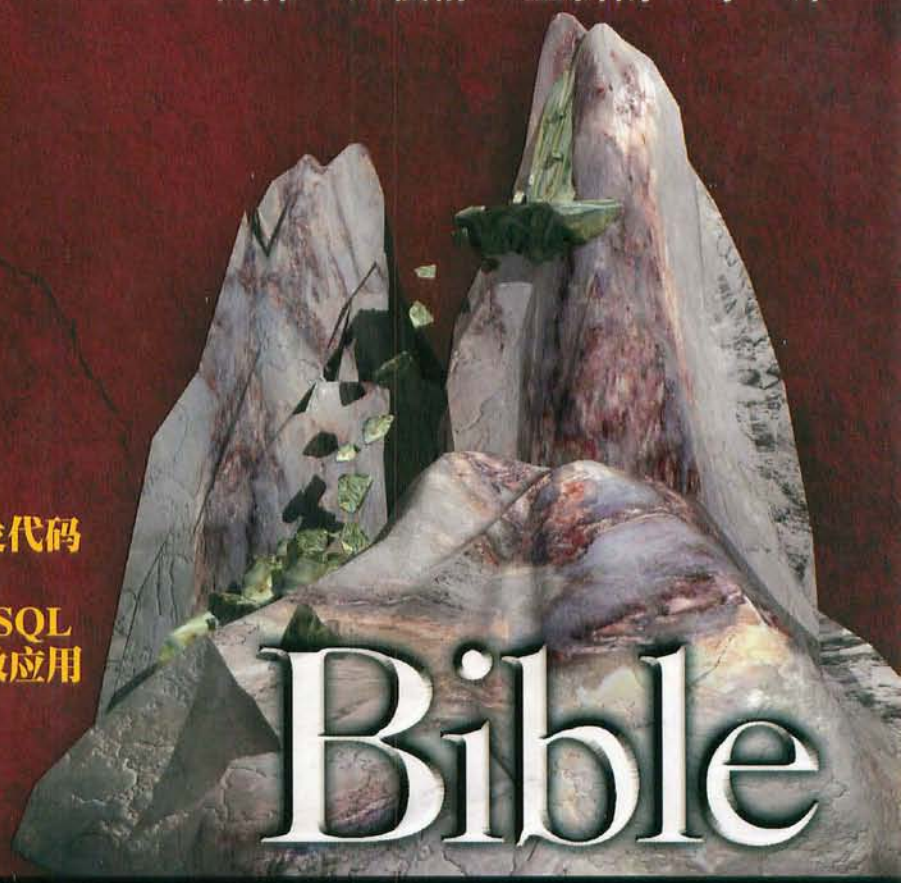
WILEY

丰富 权威 实用

SQL宝典

[美] Alex Kriegel
Boris M. Trukhnov 著
高伟 田俊静 王新南 等 译



探讨新的SQL标准

编写更有效的查询或开发代码

在Oracle、IBM DB2和SQL Server三大数据库中高效应用

Bible

人民邮电出版社
POSTS & TELECOM PRESS

SQL宝典

[美] Alex Kriegel
Boris M. Trukhnov 著
高伟 田俊静 王新雨 等 译

人民邮电出版社

www.TopSage.com

计算机精品学习资料大放送

软考官方指定教材及同步辅导书下载 | 软考历年真是解析与答案

软考视频 | 考试机构 | 考试时间安排

Java 一览无余：Java 视频教程 | Java SE | Java EE

.Net 技术精品资料下载汇总：ASP.NET 篇

.Net 技术精品资料下载汇总：C#语言篇

.Net 技术精品资料下载汇总：VB.NET 篇

撼世出击：C/C++编程语言学习资料尽收眼底 电子书+视频教程

Visual C++(VC/MFC)学习电子书及开发工具下载

Perl/CGI 脚本语言编程学习资源下载地址大全

Python 语言编程学习资料(电子书+视频教程)下载汇总

最新最全 Ruby、Ruby on Rails 精品电子书等学习资料下载

数据库管理系统(DBMS)精品学习资源汇总：MySQL 篇 | SQL Server 篇 | Oracle 篇

平面设计优秀资源学习下载 | Flash 优秀资源学习下载 | 3D 动画优秀资源学习下载

最强 HTML/xHTML、CSS 精品学习资料下载汇总

最新 JavaScript、Ajax 典藏级学习资料下载分类汇总

网络最强 PHP 开发工具+电子书+视频教程等资料下载汇总

UML 学习电子资下载汇总 软件设计与开发人员必备

经典 LinuxCBT 视频教程系列 Linux 快速学习视频教程一帖通

天罗地网：精品 Linux 学习资料大收集(电子书+视频教程) Linux 参考资源大系

Linux 系统管理员必备参考资料下载汇总

Linux shell、内核及系统编程精品资料下载汇总

UNIX 操作系统精品学习资料<电子书+视频>分类总汇

FreeBSD/OpenBSD/NetBSD 精品学习资源索引 含书籍+视频

Solaris/OpenSolaris 电子书、视频等精华资料下载索引

大家网
TopSage.com

图书在版编目（CIP）数据

SQL宝典 / （美）克里格尔（Kriegel, A.），（美）特鲁诺（Trukhnov, B. M.）著；高伟等译. —北京：人民邮电出版社，2009.5
ISBN 978-7-115-20519-3

Ⅰ. S… Ⅱ. ①克…②特…③高… Ⅲ. 关系数据库—数据库管理系统 Ⅳ. TP311.138

中国版本图书馆CIP数据核字（2009）第029699号

版 权 声 明

Alex Kriegel and Boris M.Trukhnov
SQL Bible, Second Edition
Copyright © 2008 by Wiley Publishing, Inc., Indianapolis, Indiana
All rights reserved. This translation published under license.
Authorized translation from the English language edition published by Wiley Publishing, Inc..
本书中文简体字版由 Wiley Publishing 公司授权人民邮电出版社出版，专有版权属于人民邮电出版社。

SQL 宝典

◆ 著　[美] Alex Kriegel　Boris M. Trukhnov
译　高　伟　田俊静　王新南　等
责任编辑　陈　昇
◆ 人民邮电出版社出版发行　北京市崇文区夕照寺街 14 号
邮编　100061　电子函件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
北京顺义振华印刷厂印刷
◆ 开本：787×1092　1/16
印张：30
字数：915 千字　2009 年 5 月第 1 版
印数：1－3 500 册　2009 年 5 月北京第 1 次印刷
著作权合同登记号　图字：01-2009-0379 号
ISBN 978-7-115-20519-3/TP

定价：59.00 元

读者服务热线：(010)67132705　印装质量热线：(010)67129223
反盗版热线：(010)67171154

www.TopSage.com

内容提要

本书全面介绍了结构化查询语言（Structured Query Language，SQL），从供应商特定实现和SQL标准两方面，理论和实际相结合地阐述了SQL的功能。全书分为6部分，首先介绍SQL的基本概念和原理；接着深入探讨数据库对象——表、视图、索引和序列等；然后说明数据处理和事物控制（包括插入、删除和更新记录等），以及数据检索和传送（涉及SELECT语言和SQL函数等）；最后介绍数据库安全和数据库访问等其他内容。本书始终围绕SQL标准和三大主要RDBMS（Oracle、IBM DB2和Microsoft SQL Server 2008）之间不同的实现而展开，详细介绍这三大供应商与标准之间的区别以及它们的特定实现。除了SQL的基本功能和用途外，本书还说明了SQL在商业活动中所扮演的角色，以及SQL未来的发展走向。

本书结构严谨、组织清晰、示例丰富、指导性强，适合所有层次的读者，可作为大专院校相关专业的教材，也是自学和初学者的必备参考书。

www.TopSage.com

www.TopSage.com

大家网
TopSage.com

献给我的父母：Raisa 和 Lazar Kriegel——Alex Kriegel

献给我的孩子们：Anthony、Sasha 和 Max——Boris Trukhnov

www.TopSage.com

关于作者

Alex Kriegel： 拥有 PMP、MCSD/MCTS 证书，是一位工作在俄勒冈州的系统架构师。此前他曾在 Pope and Talbot, Inc.、Psion Teklogix International, Inc.和 Belorussian Academy of Sciences 工作。他于 1988 年在 Belarus State Polytechnic 学院获得金属物理学硕士学位。Alex 是项目管理学院认证的项目管理专家（PMP），拥有多个 Microsoft 证书。他也是《Microsoft SQL Server 2000 Weekend Crash Course》（Wiley，2001）、《SQL Functions》（Wrox，2005）的作者，并与他人合著了《Introduction to Database Management》（Wiley，2007）和本书第一版。

Boris M. Trukhnov： 拥有 OCP 证书，从 1998 年起在俄勒冈州波特兰的 Pope & Talbot, Inc.担任高级技术分析师/Oracle DBA。他以前曾担任过高级程序分析师、高级软件开发人员和高级操作分析师。他从 1994 年开始使用 SQL 和关系数据库。Boris 拥有明尼苏达大学计算机科学学士学位。他是《Introduction to Database Management》（Wiley，2007）的合著者。

www.TopSage.com

致 谢

Alex：我想感谢我的全家对我的耐心、支持和理解。

Boris：我真诚地感谢我的妻子 Kate，她给予了我专业帮助、精神支持和毫无保留的理解。

我感谢我的所有朋友们，在我需要他们的时候，他们总是和我在一起。

我也要感谢 Jaideep Srivastava 教授，他把我带进了关系数据库的世界。

Alex 和 **Boris**：我们要感谢 Jim Melton 进行的一丝不苟的编辑和对本书内容的深刻洞察；他的帮助使本书终于完美地付梓！我们要感谢其他技术编辑——Victor Shulman（也是对本书做出贡献的作者）和 Christopher Moore，他们为准备本书的最终版本付出了努力。

我们要真诚地感谢作出贡献的作者们：Alex Makarov 提供了附录 D 和附录 E；Kate Trukhnov 提供了第 8 章；Victor Shulman 指明了通常的数据库技术要点，特别是 SQL/XML，而且提供了第 15 章。

我们感谢 Wiley Publishing 的执行编辑 Robert Elliott，他帮助开发并提供了该项目，在整个繁复的编辑过程中表现出极大的耐心并提供了专业指导。

我们感谢 Wiley 编辑团队，特别是输出开发编辑 Chirstopher Rivera 和产品编辑 Liz Britten，他们提供了改进本书内容的有价值建议，指出我们的冗长部分、疏忽和不足之处。他们的帮助使得本书好上加好。

www.TopSage.com

大家网
TopSage.com

做出贡献的作者们

Kate Trukhnova：在俄勒冈州波特兰的 Legacy Health System 担任财务系统分析师，她以前是俄勒冈州人力服务部门的高级系统分析师和 Amdocs, Inc.的高级软件工程师。Kate 在信息技术行业有 9 年以上的工作经验。

Alex L. Makarov：拥有 OCP 证书，从 2005 年开始在俄勒冈州波特兰的 Blackwell Book Services 担任 Oracle DBA。他擅长 SQL 和 PL/SQL 编程、关系数据库设计以及 RDBMS 支持。Alex 还拥有 DB2 和其他关系数据库方面的工作经验。

Victor Shulman：是位于 Edmonton，AB 的一家大型公司的软件开发人员。他对最新的架构方向和开发技术，乃至提高客户满意度等所有方面都有浓厚的兴趣。

www.TopSage.com

前 言

本书介绍有关结构化查询语言的内容。众所周知，SQL 是关系数据库的标准语言和数据库世界的混合语。它已历经 20 多年而光彩依旧。它经过多次修订，因数据库供应商频繁引入专有发明（大概每两到三年就一次），这些成果要么被纳入标准，要么随着数据库社区的发展而被废弃。最新的 SQL 标准在 2003 年发布，将在 2008 年再次更新。

不管取得怎样令人瞩目的进展，在特定实现和 ANSI/ISO 的 SQL 标准之间仍然存在显著的差异。这就使得很难找到一本“涵盖所有内容”的 SQL 书籍。一位作者可能对某个特定供应商有偏好，这样读者会得到一本很好的 Oracle 或 Microsoft SQL Server 书籍，但它未必是一本 SQL 方面的好书；一本单独说明全部 SQL ANSI/ISO 标准的书籍可能会满足 RDBMS 开发人员的需要，但它可能对商业从业人员用处不大。我们相信这两种方式的结合是联系理论与实践的绝好方法。

通过分析现代数据库供应商市场后发现，Oracle、IBM DB2 和 Microsoft SQL Server 已经并且可能继续拥有最大的市场份额。这并不意味着其他供应商就无关紧要了。“较小的供应商”提供的某些功能可以满足甚至超过“三大”（我们将如此称呼它们）供应商提供的功能；它们拥有自己的忠诚客户，并且这种市场态势今后还将继续。不能忽视开放源代码社区正在对数据库市场产生的巨大冲击，其中诸如 MySQL（来自 MySQL AB）和 PostgreSQL（来自 PostgreSQL 全球开发组）这样的 RDBMS 正在分享一定的市场份额。但是本书不可能讨论每一种和每一个专用 SQL 扩展插件，所以把讨论的重点集中在“三大”数据库供应商上，并通过强调 Oracle、DB2 和 Microsoft SQL Server 之间的不同以及它们与当前 SQL 标准的差别来说明 SQL 功能。

Note 注意：Sybase Adaptive Server 的 SQL 语法在很多方面类似于 Microsoft SQL Server 的语法，本书中的大多数 Microsoft SQL Server 例子也可以在 Sybase RDBMS 中运行。

本书的读者

本书面对所有层次的读者——从初学者到高级用户。本书的目的是为读者提供全面的参考，帮助与关系数据库打交道的每个人，特别是在异类环境中。程序员和数据库管理员可以找到有关 SQL 标准的最新信息以及大多数主流数据库产品采用的用语。不管使用的是三大 RDBMS 中的哪一个，数据库用户都可以更深刻地理解幕后过程，使得日常工作更轻松。评估数据库产品的管理者能深入了解 RDBMS 技术的内在实现。对于那些必须对其组织的 RDBMS 需求进行策划的管理者来说，本书也说明了 SQL 在现代商业活动中扮演的角色，以及 SQL 未来的发展趋势。

本书的组织结构

本书包含 17 章，分为 6 个部分。还有 11 个附录。

第 1 部分：SQL 基本概念和原理

第一部分中的 3 章介绍了 SQL——关系数据库的标准语言。第 1 章介绍语言和关系数据库系统（RDBMS）的历史，第 2 章和第 3 章从较高层次概述构建 SQL 的主要原理，并深入讨论了 SQL 数据类型。强调了 SQL 标准和三大 RDBMS 实现——Oracle、IBM DB2 和 Microsoft SQL Server 2008 之间的不同。

www.TopSage.com

第 2 部分：创建和修改数据库对象

第 2 部分的 2 章继续深入说明数据库对象——表、视图、索引、序列等，包括创建、修改和删除数据库对象的语法，再次突出说明了标准与它特定实现之间的区别。

第 3 部分：数据操作和事务控制

在第 3 部分中，第 6 章介绍了数据操作语言（Data Manipulation Language，DML），该语言用于在数据库表中插入、更新和删除记录。此外还详细讨论了高级 MERGE 和 TRUNCATE 语句。再次强调了 Oracle、IBM 和 Microsoft 在 RDBMS 实现方面的不同。第 7 章从 SQL 标准的角度说明了多用户环境中的会话、事务和锁定机制，并通过实际的实现进行了比较。

第 4 部分：检索和传送数据

第 4 部分在第 8 章和第 9 章中介绍了最重要的 SQL 语句之一——SELECT。从简单的单表查询到高级的多表 SELECT 语句，说明了供应商特定实现之间的不同。第 10 章主要介绍 SQL 函数。既包括 SQL 标准规定的函数，也包括 RDBMS 供应商提供的函数。交叉参考了所有三大实现的最常用函数。第 11 章讨论了 SQL 运算符以及它们在几个 RDBMS 供应商中的实现和在不同环境中的用法。

第 5 部分：使用系统目录实现安全

不能低估在日益庞大的互联世界中信息安全的重要性。第 12 章介绍数据库安全的重要概念，包括贯穿 SQL 的基本安全性和供应商集成到各自产品中的高级安全性。第 13 章讨论通过 SQL 标准规定的 INFORMATION_SCHEMA 和系统目录访问 RDBMS 中的元数据信息的问题。

第 6 部分：超越 SQL——过程编程和数据库访问机制

第 6 部分中的各章涉及那些不是 SQL 固有领域的主题，如用于创建存储过程、函数和触发器的过程扩展插件（像 Oracle 的 PL/SQL 和 Microsoft 的 Transact SQL）。这一部分全面介绍嵌入式 SQL 和动态 SQL，通过用 C、Visual Basic、Java 和 C#等语言进行编程的例子，深入探讨 RDBMS 访问接口这一复杂主题（像 ODBC、OLEDB、ADO.NET 和 JDBC）。最后一章主要介绍 SQL 世界中当前出现的新开发技术：XML 集成、OLAP 商业智能和 RDBMS 的面向对象功能。

附录

附录提供“如何做”指南以及用于正文的大量参考资料。

- 附录 A 说明配套网站的内容。
- 附录 B 说明本书中每个例子实际使用的 ACME 示例数据库结构。
- 附录 C 讨论关系数据库设计基础。
- 附录 D、E 和 F 详细说明如何安装 RDBMS 软件（可以从供应商网站下载），如何访问选择的 RDBMS 以及如何安装 ACME 示例数据库（配套网站上提供了所有 SQL 脚本）。
- 附录 G 列出了 Oracle 11g、IBM DB2 9.5 和 Microsoft SQL Server 2008 的 500 多个 SQL 函数，包括每个函数功能的简要描述。
- 附录 H 提供 SQL 语法参考，可以快速查找需要的语句。
- 附录 I 是所有 SQL 保留关键字列表，在编程中不能用它们作标识符。
- 附录 J 列出除 Oracle、IBM 和 Microsoft 开发的 RDBMS 产品之外可能会用到的一些 RDBMS 产品。
- 附录 K 简要介绍了集合和离散数学理论，这有助于理解 SQL 管理的一般原理。

www.TopSage.com

大学网 TopSage.com

本书中使用的约定

本书中的所有编程代码（包括 SQL 语句、数据库对象名称、变量声明等）显示为这种字体：`fixed-width`。

按下面的方式显示层次菜单选项：“文件”→“保存”，这个例子中表示选择菜单栏上的“文件”，然后从出现的子菜单中选择“保存”。

全书中还有如下特殊内容：

注意：提供有关主题的额外信息。

提示：显示使工作更快或更高效的方式。

警告：提供应当考虑的额外信息。

交叉参考：显示可以在本书的另一部分中找到相关主题的内容。

什么是侧栏？

侧栏提供相关但有时偏离主要主题的信息。

www.TopSage.com

计算机精品学习资料大放送

软考官方指定教材及同步辅导书下载 | 软考历年真是解析与答案

软考视频 | 考试机构 | 考试时间安排

Java 一览无余：Java 视频教程 | Java SE | Java EE

.Net 技术精品资料下载汇总：ASP.NET 篇

.Net 技术精品资料下载汇总：C#语言篇

.Net 技术精品资料下载汇总：VB.NET 篇

撼世出击：C/C++编程语言学习资料尽收眼底 电子书+视频教程

Visual C++(VC/MFC)学习电子书及开发工具下载

Perl/CGI 脚本语言编程学习资源下载地址大全

Python 语言编程学习资料(电子书+视频教程)下载汇总

最新最全 Ruby、Ruby on Rails 精品电子书等学习资料下载

数据库管理系统(DBMS)精品学习资源汇总：MySQL 篇 | SQL Server 篇 | Oracle 篇

平面设计优秀资源学习下载 | Flash 优秀资源学习下载 | 3D 动画优秀资源学习下载

最强 HTML/xHTML、CSS 精品学习资料下载汇总

最新 JavaScript、Ajax 典藏级学习资料下载分类汇总

网络最强 PHP 开发工具+电子书+视频教程等资料下载汇总

UML 学习电子资下载汇总 软件设计与开发人员必备

经典 LinuxCBT 视频教程系列 Linux 快速学习视频教程一帖通

天罗地网：精品 Linux 学习资料大收集(电子书+视频教程) Linux 参考资源大系

Linux 系统管理员必备参考资料下载汇总

Linux shell、内核及系统编程精品资料下载汇总

UNIX 操作系统精品学习资料<电子书+视频>分类总汇

FreeBSD/OpenBSD/NetBSD 精品学习资源索引 含书籍+视频

Solaris/OpenSolaris 电子书、视频等精华资料下载索引

目　录

第 1 部分　SQL 基本概念和原理

第 1 章　SQL 和关系数据库管理系统 ……3
1.1　理想的数据库特征 ……3
1.1.1　充足的容量 ……3
1.1.2　足够的安全和审核 ……4
1.1.3　多用户环境 ……4
1.1.4　效率和查找能力 ……4
1.1.5　可伸缩性 ……4
1.1.6　用户友好 ……4
1.2　选择数据库软件 ……5
1.2.1　市场占有率 ……5
1.2.2　总拥有成本 ……5
1.2.3　支持和持久性 ……6
1.3　主要 DBMS 实现 ……6
1.4　现实中的数据库例子 ……7
1.4.1　订购管理系统数据库 ……7
1.4.2　健康护理提供者数据库 ……8
1.4.3　视频共享和编辑数据库 ……8
1.4.4　科学数据库 ……8
1.4.5　非营利组织数据库 ……8
1.5　早期数据库 ……9
1.5.1　平面文件数据库 ……9
1.5.2　层次数据库 ……9
1.5.3　网状数据库 ……10
1.6　关系数据库 ……11
1.6.1　表 ……11
1.6.2　关系 ……12
1.6.3　主键 ……12
1.6.4　外键 ……13
1.6.5　RDBMS 的发展 ……13
1.7　其他 DBMS 模型 ……13
1.8　SQL 和 SQL 标准简史 ……14
1.8.1　卑微的开始：RDBMS 和 SQL 的演变 ……14
1.8.2　SQL 标准简史 ……16
1.9　小结 ……18
第 2 章　SQL 基本概念和原理 ……19
2.1　约定和要求 ……19
2.2　SQL 初探 ……22
2.2.1　数据库示例 ……22
2.2.2　获取和导出数据 ……22
2.2.3　从不同角度看数据 ……24
2.2.4　聚合 ……25
2.2.5　数据安全 ……25
2.2.6　从客户端应用程序访问数据 ……26
2.2.7　新的发展 ……26
2.3　平台和时间无关性 ……27
2.4　小结 ……28
第 3 章　SQL 数据类型 ……30
3.1　不附加字符串 ……31
3.1.1　字符串 ……31
3.1.2　二进制字符串 ……35
3.2　数字 ……36
3.2.1　精确数字 ……37
3.2.2　近似数字 ……38
3.3　日期和时间数据类型 ……39
3.3.1　复杂数据类型介绍 ……39
3.3.2　日期和时间实现 ……40
3.4　XML 数据类型 ……44
3.5　结构数据类型和用户定义数据类型 ……44
3.5.1　SQL:2003 ……45
3.5.2　Oracle 11g ……46
3.5.3　DB2 9.5 ……47
3.5.4　Microsoft SQL Server 2008 ……48
3.6　其他数据类型 ……48
3.6.1　BOOLEAN ……48
3.6.2　ROWID ……49
3.6.3　UROWID ……49
3.6.4　BFILE ……49
3.6.5　DATALINK ……49
3.6.6　BIT ……49
3.6.7　TIMESTAMP ……49
3.6.8　SQL_VARIANT ……49
3.7　NULL ……49
3.8　小结 ……50

www.TopSage.com

大家网 TopSage.com

第 2 部分　创建和修改数据库对象

第 4 章　创建 RDBMS 对象 …… 53
4.1　表 …… 53
4.2　索引 …… 70
4.3　视图 …… 74
4.3.1　CREATE VIEW 语句 …… 74
4.3.2　创建复杂视图 …… 77
4.4　别名和同义词 …… 79
4.4.1　SQL:2003 …… 80
4.4.2　Oracle 11g 的 CREATE SYNONYM 语句 …… 80
4.4.3　DB2 9.5 的 CREATE ALIAS 语句 …… 80
4.4.4　Microsoft SQL Server 2008 的 CREATE SYNONYM 语句 …… 81
4.5　架构 …… 81
4.6　序列 …… 83
4.6.1　SQL:2003 中的外部序列生成器 …… 84
4.6.2　Oracle 11g 中的序列 …… 84
4.6.3　DB2 9.5 …… 85
4.7　其他 SQL:2003 以及特定于实现的对象 …… 86
4.7.1　域（SQL:2003） …… 86
4.7.2　字符集（SQL:2003） …… 87
4.7.3　排序规则（SQL:2003） …… 87
4.7.4　表空间和文件组 …… 87
4.7.5　物化视图（Oracle 11g） …… 88
4.7.6　数据库链接（Oracle 11g） …… 89
4.7.7　目录（Oracle 11g） …… 90
4.8　CREATE 语句交叉参考 …… 90
4.9　小结 …… 92
第 5 章　修改和删除 RDBMS 对象 …… 93
5.1　表 …… 93
5.1.1　ALTER TABLE 语句 …… 93
5.1.2　DROP TABLE 语句 …… 99
5.2　索引 …… 100
5.2.1　ALTER INDEX 语句 …… 100
5.2.2　DROP INDEX 语句 …… 101
5.3　视图 …… 102
5.3.1　ALTER VIEW 语句 …… 102
5.3.2　DROP VIEW 语句 …… 103
5.4　别名和同义词 …… 103
5.5　架构 …… 104
5.5.1　SQL:2003 …… 104
5.5.2　DB2 9.5 …… 104
5.5.3　Microsoft SQL Server 2008 …… 104
5.6　序列 …… 105
5.6.1　ALTER SEQUENCE 语句 …… 105
5.6.2　DROP SEQUENCE 语句 …… 105
5.7　其他特定于实现的对象 …… 106
5.7.1　域（SQL:2003） …… 106
5.7.2　字符集（SQL:2003） …… 106
5.7.3　排序规则（SQL:2003） …… 106
5.7.4　表空间 …… 106
5.7.5　物化视图（Oracle 11g） …… 107
5.7.6　数据库链接（Oracle 11g） …… 107
5.7.7　目录（Oracle 11g） …… 107
5.8　ALTER 和 DROP 语句交叉参考 …… 108
5.9　小结 …… 110

第 3 部分　数据操作和事务控制

第 6 章　数据操作语言 …… 113
6.1　INSERT：用数据来填充表 …… 113
6.1.1　一般的 INSERT 语句子句 …… 114
6.1.2　供应商相关的特定 INSERT 语句 …… 118
6.2　UPDATE：修改表数据 …… 123
6.2.1　一般的 UPDATE 语句子句 …… 123
6.2.2　供应商特定的 UPDATE 语句细节 …… 126
6.3　DELETE：从表中删除数据 …… 127
6.3.1　一般的 DELETE 语句子句 …… 128
6.3.2　供应商特定 DELETE 语句子句 …… 129
6.4　MERGE：将 INSERT、UPDATE 以及 DELETE 结合到一个语句中 …… 129
6.4.1　一般的 MERGE 语句子句 …… 129
6.4.2　供应商特定的 MERGE 语句子句 …… 130
6.5　TRUNCATE 语句 …… 132
6.6　小结 …… 133
第 7 章　会话、事务和锁 …… 134
7.1　会话 …… 134
7.2　事务 …… 139
7.2.1　什么是事务 …… 139
7.2.2　事务的 COMMIT 和 ROLLBACK …… 141
7.2.3　事务隔离级别 …… 145

www.TopSage.com

7.3 理解锁……147
7.3.1 锁定模式……147
7.3.2 处理死锁……149
7.4 小结……150

第 4 部分 检索和传送数据

第 8 章 理解 SELECT 语句……153
8.1 单表 SELECT 语句语法……153
8.2 SELECT 子句：选取什么？……154
8.2.1 单列选取……154
8.2.2 多列 SELECT……154
8.2.3 使用常量、函数和计算列……156
8.2.4 在 SELECT 子句中使用子查询……159
8.3 FROM 子句：从哪里选取？……159
8.3.1 从表和视图中选取……159
8.3.2 在 FROM 子句中使用别名……160
8.3.3 在 FROM 子句中使用子查询（内联视图）……160
8.4 WHERE 子句：设置水平限制……161
8.4.1 使用比较运算符……161
8.4.2 复合运算符：使用 AND 和 OR……162
8.4.3 使用 BETWEEN 运算符……162
8.4.4 使用 IN 运算符：集成员身份测试……163
8.4.5 NOT 运算符……164
8.4.6 使用 IS NULL 运算符：NULL 值的专用测试……164
8.4.7 在 WHERE 子句中使用子查询……165
8.5 GROUP BY 和 HAVING 子句：汇总结果……167
8.6 ORDER BY 子句：对查询结果排序……169
8.7 联合多个查询的结果……170
8.7.1 UNION……171
8.7.2 INTERSECT……172
8.7.3 EXCEPT（MINUS）……173
8.8 SQL 分析函数和 TOP N 查询……175
8.8.1 分析函数和 SQL:2003 标准……175
8.8.2 top N 查询设计……176
8.8.3 Limit N 查询……177
8.9 小结……178
第 9 章 多表查询……179
9.1 内部联接……179
9.1.1 内部联接的两个语法……179
9.1.2 同等联接……180
9.1.3 非同等联接……182
9.1.4 自联接……183
9.1.5 交叉联接（笛卡尔积）……184
9.1.6 对两个以上的表进行联接……185
9.2 外部联接：根据包含 NULL 值的列联接表……186
9.2.1 外部联接的两个语法……186
9.2.2 左外部联接……188
9.2.3 右外部联接……189
9.2.4 完全外部联接……190
9.2.5 合并联接……191
9.3 包含内联视图的联接……191
9.4 包含相关查询的多表联接……193
9.5 提高多表查询效率……193
9.6 小结……194
第 10 章 SQL 函数……195
10.1 数字函数……197
10.1.1 CEIL……198
10.1.2 ROUND……198
10.1.3 TRUNC……199
10.1.4 RAND……199
10.1.5 SIGN……200
10.2 字符串函数……200
10.2.1 CONCAT……201
10.2.2 CHARINDEX、INSTR、LOCATE 和 POSSTR……202
10.2.3 SUBSTR 和 SUBSTRING……203
10.2.4 LENGTH……204
10.2.5 LOWER 和 UPPER……205
10.2.6 TO_CHAR、CHAR 和 STR……205
10.2.7 REPLACE……206
10.2.8 REPLICATE 和 REPEAT……206
10.2.9 TRANSLATE……206
10.2.10 TRIM、LTRIM 和 RTTIM……207
10.3 日期和时间函数……208
10.3.1 GETDATE、SYSDATE 和 CURRENT DATE……209
10.3.2 ADD_MONTHS、DATEADD 和日期算法……211
10.3.3 EXTRACT 和 DATEPART……211

www.TopSage.com

10.3.4 DAYNAME、MONTHNAME 和 DATENAME ····· 212
10.3.5 MONTHS_BETWEEN 和 DATEDIFF ····· 213
10.4 聚合函数 ····· 213
10.4.1 SUM ····· 215
10.4.2 COUNT ····· 216
10.4.3 AVG ····· 217
10.4.4 MIN 和 MAX ····· 217
10.5 转换函数 ····· 218
10.5.1 不同数据类型之间的转换 ····· 218
10.5.2 在不同字符集之间转换 ····· 224
10.5.3 Oracle 11g 的 CONVERT、TRANSLATE...USING 和 UNISTR ····· 224
10.5.4 特定于数据类型的转换函数 ····· 225
10.6 系统函数 ····· 226
10.7 杂项函数 ····· 228
10.7.1 DECODE 和 CASE ····· 229
10.7.2 COALESCE 和 NULLIF ····· 230
10.7.3 NVL、NVL2 和 ISNULL ····· 230
10.8 用户定义函数 ····· 232
10.9 小结 ····· 232
第 11 章 SQL 运算符 ····· 233
11.1 算术运算符和字符串串联运算符 ····· 233
11.2 逻辑运算符 ····· 235
11.2.1 ALL ····· 236
11.2.2 ANY|SOME ····· 236
11.2.3 BETWEEN<expression> AND<expression> ····· 236
11.2.4 IN ····· 237
11.2.5 EXISTS ····· 238
11.2.6 LIKE ····· 238
11.2.7 AND ····· 241
11.2.8 NOT ····· 241
11.2.9 OR ····· 241
11.3 运算符优先级 ····· 241
11.4 赋值运算符 ····· 243
11.5 比较运算符 ····· 243
11.6 位运算符 ····· 246
11.7 小结 ····· 247

第 5 部分 使用系统目录实现安全

第 12 章 SQL 和 RDBMS 的安全 ····· 251
12..1 基本安全机制 ····· 251
12.1.1 标识和身份验证 ····· 251
12.1.2 授权和访问控制 ····· 251
12.1.3 加密 ····· 252
12.1.4 完整性和一致性 ····· 252
12.1.5 审核 ····· 252
12.2 定义数据库用户 ····· 252
12.3 使用特权来管理安全 ····· 256
12.3.1 GRANT 语句 ····· 256
12.3.2 REVOKE 特权 ····· 264
12.4 使用角色管理安全 ····· 267
12.5 使用视图实现安全 ····· 271
12.6 利用约束实现安全 ····· 273
12.7 利用存储过程和触发器实现安全 ····· 274
12.8 数据加密 ····· 275
12.9 数据库审核 ····· 278
12.10 安全标准 ····· 280
12.10.1 国际安全标准 ····· 281
12.10.2 规章制度的遵从性 ····· 282
12.11 小结 ····· 283
第 13 章 系统目录和 INFORMATION_SCHEMA ····· 284
13.1 SQL 系统目录 ····· 284
13.2 Oracle 11g 数据字典 ····· 286
13.2.1 Oracle 数据字典结构 ····· 287
13.2.2 Oracle 数据字典和 SQL: 2007 标准 ····· 288
13.2.3 更深一层：有关元数据的数据 ····· 289
13.3 IBM DB2 9.5 系统目录 ····· 290
13.3.1 DB2 中的 INFORMATION_SCHEMA 对象 ····· 290
13.3.2 获得有关 INFORMATION_SCHEMA 对象的信息 ····· 292
13.4 Microsoft SQL Server 2008 系统目录 ····· 293
13.4.1 MS SQL Server 2008 的 INFORMATION_SCHEMA 视图 ····· 293
13.4.2 Microsoft SQL Server 系统存储过程 ····· 295
13.4.3 Microsoft SQL Server 2008 系统函数 ····· 297
13.5 小结 ····· 299

www.TopSage.com

大学网 TopSage.com

第 6 部分 超越 SQL：过程编程和数据库访问机制

第 14 章 存储过程、触发器和用户定义函数……303
14.1 过程化扩展的用途和好处……304
14.1.1 性能和网络通信量……304
14.1.2 数据库安全……304
14.1.3 代码的可重用性……305
14.2 SQL 过程化语言的关键元素……305
14.2.1 变量和赋值……305
14.2.2 模块化、子程序和块结构……306
14.2.3 参数传递……307
14.2.4 有条件地执行……309
14.2.5 重复执行……310
14.2.6 游标……310
14.2.7 错误处理……313
14.3 存储过程……315
14.3.1 CREATE PROCEDURE 语法……315
14.3.2 创建一个简单的存储过程……315
14.3.3 删除存储过程……319
14.4 用户自定义函数……319
14.4.1 CREATE FUNCTION 语法……319
14.4.2 创建一个简单的函数……320
14.4.3 删除用户定义函数……322
14.5 触发器……322
14.5.1 CREATE TRIGGER 语法……322
14.5.2 删除触发器……324
14.6 .Net 存储过程和函数……324
14.7 小结……325
第 15 章 SQL 和 XML……326
15.1 简介……326
15.2 XML 的结构……326
15.2.1 格式化 XML……327
15.2.2 DTD 和架构……327
15.2.3 命名空间……328
15.3 作为数据源的 XML……329
15.3.1 在应用程序里访问 XML 文档……329
15.3.2 XPath……330
15.3.3 XQuery……330
15.4 编码 XML……330
15.4.1 表示 XML 文档……331
15.4.2 XML 和 RDBMS……332
15.4.3 Oracle 11g XML DB……333
15.4.4 IBM DB 9.5 pureXML……337
15.4.5 Microsoft SQL Server 2008……339
15.5 小结……343
第 16 章 SQL 和过程化程序设计……344
16.1 SQL 语句的处理步骤……344
16.2 嵌入式 SQL 和动态 SQL……345
16.2.1 嵌入式 SQL……345
16.2.2 动态 SQL 技术……348
16.3 SQL/CLI 标准……352
16.4 开放式数据库连接以及数据库对象链接和嵌入……355
16.4.1 ODBC……355
16.4.2 OLEDB……356
16.5 SQL 和 Java……356
16.5.1 Java 数据库连接（JDBC）……356
16.5.2 SQLJ……359
16.6 Oracle API 选项……360
16.6.1 Oracle 调用接口……360
16.6.2 OLE 的 Oracle 对象……361
16.6.3 .NET 的 Oracle 数据提供程序……362
16.7 IBM DB2 调用级接口……362
16.8 Microsoft 数据访问接口……362
16.8.1 ActiveX 数据对象……363
16.8.2 ADO.NET……363
16.9 小结……366
第 17 章 SQL 的未来……367
17.1 OLAP 和商业智能……367
17.1.1 Oracle 11g……369
17.1.2 IBM DB2 9.5……369
17.1.3 Microsoft SQL Server 2008……370
17.2 SQL 的 LINQ……370
17.3 对象……371
17.3.1 OOP 范例……371
17.3.2 SQL: 2003 标准中的对象语言绑定（SQL/OLB）……372
17.3.3 Oracle 11g 的支持……372
17.3.4 IBM DB2 9.5 的支持……373
17.3.5 Microsoft SQL Server 2008……374
17.3.6 抽象数据类型……374
17.3.7 面向对象的数据库……375
17.4 小结……376
附录 A 配套网站……377
附录 B ACME 示例数据库……378
B.1 一般信息和业务规则……378

www.TopSage.com

B.2 命名约定 ····· 379
B.3 表间关系 ····· 379
B.4 列约束和默认值 ····· 380
B.5 索引 ····· 380
B.6 视图 ····· 380
B.7 创建 ACME 数据库对象的 SQL 脚本 ····· 380
附录 C 关系数据库设计基础 ····· 387
C.1 标识实体和属性 ····· 388
C.2 规范化 ····· 388
C.2.1 第一范式 ····· 388
C.2.2 第二范式 ····· 389
C.2.3 第三范式 ····· 390
C.3 指定约束 ····· 391
C.4 OLTP 与 OLAP 设计 ····· 391
C.4.1 数据仓库和数据市场 ····· 391
C.4.2 星型和雪花型架构 ····· 391
C.5 关系数据库设计的缺陷 ····· 392
附录 D 安装 RDBMS 软件 ····· 393
D.1 安装 Oracle 11g ····· 393
D.1.1 在 Linux 上安装 Oracle11g release 1（11.1.0.6.0）企业版 ····· 393
D.1.2 在 Windows 上安装 Oracle 11g release 1（11.1.0.6.0）个人版 ····· 399
D.2 安装 IBM DB2 9.5 ····· 401
D.3 安装 Microsoft SQL Server 2008 ····· 404
D.3.1 安装 Microsoft SQL Server 2008（CTP，2007 年 11 月发布）的必备条件 ····· 404
D.3.2 在 Windows Vista Enterprise Edition 上安装 ····· 405
附录 E 访问 RDBMS ····· 409
E.1 使用 ORACLE 11g 实用工具访问 RDBMS ····· 409
E.1.1 SQL*Plus ····· 409
E.1.2 SQL Developer ····· 411
E.1.3 Oracle 企业管理器（SQL Worksheet） ····· 412
E.2 使用 IBM DB2 9.5 实用工具访问 RDBMS ····· 413
E.2.1 命令行处理器（Command-Line Processor，CLP） ····· 413
E.2.2 命令编辑器 ····· 414
E.3 使用 Microsoft SQL Server 2008 实用工具访问 RDBMS ····· 415
E.3.1 使用 SQLCMD ····· 415
E.3.2 使用 SQL Server Management Studio ····· 416
附录 F 安装 ACME 数据库 ····· 417
F.1 在 Oracle 11g 上使用 SQL*Plus 安装 ACME 数据库 ····· 417
F.1.1 Linux/UNIX ····· 417
F.1.2 Microsoft Windows ····· 417
F.2 在 DB2 9.5（Windows）上使用 CLP 安装 ACME 数据库 ····· 418
F.3 在 Microsoft SQL Server 2008 上使用 SQLCMD 实用工具安装 ACME 数据库 ····· 419
附录 G SQL 函数 ····· 420
附录 H SQL 语法参考 ····· 437
H.1 DDL 语句 ····· 437
H.1.1 表 ····· 437
H.1.2 索引 ····· 438
H.1.3 视图 ····· 438
H.1.4 架构 ····· 438
H.1.5 存储过程 ····· 438
H.1.6 用户定义函数 ····· 438
H.1.7 触发器 ····· 438
H.2 DCL 语句 ····· 439
H.3 DML 语句 ····· 439
H.4 DQL 语句 ····· 439
H.4.1 单表 SELECT ····· 439
H.4.2 多表 SELECT ····· 439
H.5 事务控制语句 ····· 440
H.6 谓词 ····· 440
附录 I SQL 保留关键字 ····· 442
I.1 未来的关键字 ····· 446
I.2 ODBC 的保留关键字 ····· 447
附录 J 其他 RDBMS ····· 449
附录 K 数值系统、布尔代数和集合论简介 ····· 452
K.1 数值系统 ····· 452
K.1.1 RDBMS 连接 ····· 452
K.1.2 转换数值 ····· 453
K.2 布尔代数的逻辑元素 ····· 453
K.2.1 NOT（补集或求反） ····· 454
K.2.2 AND（布尔积） ····· 454
K.2.3 OR（布尔和） ····· 454
K.2.4 XOR（异或） ····· 454

www.TopSage.com

K.2.5 NAND（与非）……454
K.2.6 NOR（或非）……454
K.2.7 优先级规则……455
K.3 集合论……456
K.3.1 集合列表……456
K.3.2 子集……456
K.3.3 集合等式……456
K.3.4 集合运算……457
K.3.5 集合代数恒等式……459

www.TopSage.com

计算机精品学习资料大放送

软考官方指定教材及同步辅导书下载 | 软考历年真是解析与答案

软考视频 | 考试机构 | 考试时间安排

Java 一览无余：Java 视频教程 | Java SE | Java EE

.Net 技术精品资料下载汇总：ASP.NET 篇

.Net 技术精品资料下载汇总：C#语言篇

.Net 技术精品资料下载汇总：VB.NET 篇

撼世出击：C/C++编程语言学习资料尽收眼底 电子书+视频教程

Visual C++(VC/MFC)学习电子书及开发工具下载

Perl/CGI 脚本语言编程学习资源下载地址大全

Python 语言编程学习资料(电子书+视频教程)下载汇总

最新最全 Ruby、Ruby on Rails 精品电子书等学习资料下载

数据库管理系统(DBMS)精品学习资源汇总：MySQL 篇 | SQL Server 篇 | Oracle 篇

平面设计优秀资源学习下载 | Flash 优秀资源学习下载 | 3D 动画优秀资源学习下载

最强 HTML/xHTML、CSS 精品学习资料下载汇总

最新 JavaScript、Ajax 典藏级学习资料下载分类汇总

网络最强 PHP 开发工具+电子书+视频教程等资料下载汇总

UML 学习电子资下载汇总 软件设计与开发人员必备

经典 LinuxCBT 视频教程系列 Linux 快速学习视频教程一帖通

天罗地网：精品 Linux 学习资料大收集(电子书+视频教程) Linux 参考资源大系

Linux 系统管理员必备参考资料下载汇总

Linux shell、内核及系统编程精品资料下载汇总

UNIX 操作系统精品学习资料<电子书+视频>分类总汇

FreeBSD/OpenBSD/NetBSD 精品学习资源索引 含书籍+视频

Solaris/OpenSolaris 电子书、视频等精华资料下载索引

SQL 基本概念和原理

PART 1

本部分包括：

第 1 章
SQL 和关系数据库管理系统

第 2 章
SQL 基本概念和原理

第 3 章
SQL 数据类型

www.TopSage.com

www.TopSage.com

大家网
TopSage.com

CHAPTER 1

第 1 章
SQL 和关系数据库管理系统

本章提要：

- 理解数据库
- 一个好数据库的特征
- 数据库市场
- 现实中的数据库例子
- 数据库简史
- SQL 标准及其实现间的差异

信息可能是现代社会中最有价值的商品。它可以以多种不同的形式呈现出来，例如会计账目和薪金信息、有关客户和订购者的信息、科学和统计数据、图形或多媒体。事实上，我们是完全被数据淹没的，而且（至少我们愿意这样认为）这些数据不能够丢失。现在，我们拥有的数据实在是太多了，无法把它们都保存到文件柜或硬纸箱中。对每一个企业而言，安全地存储需要长期保存的海量数据，允许多用户以不同的角度对海量数据进行有效的查询，并在需要的时候能很方便地对数据进行更新，这些都是极为关键和重要的问题。这些工作需要交由现有的数据库去处理，它能够完成上述所有的任务以及其他更多的事情。简单地说，数据库就是一个有组织的信息集合（重点是“有组织的”）。

一个更明确的定义是**数据库管理系统**（DBMS），它常被用做“数据库”的同义词。这个术语的含义更为宽泛，除了指存储信息外，还包含了一些使用数据的方法和维护数据的工具。

Note 注意：DBMS 可以被定义为一个相关数据的集合及一套用于访问、修改和维护数据的程序。有关 DBMS 的更多信息将在本章的后面介绍。

1.1 理想的数据库特征

关于数据库是什么和它应该做什么，存在着许多不同的意见。但不管怎么说，所有现代数据库都应该至少具有下列特征。

1.1.1 充足的容量

数据库的主要功能是存储海量信息。例如，一个中型规模企业的订购管理系统，其数据量动辄就是千兆字节甚至兆兆字节；并且公司越大，需要存储和依赖的数据就越多。一个需要保存历史（档案）数据的公司则需要更为庞大的存储空间。对存储容量的需求在迅速增长，一个很好的例子就是近几年对音频和视频文件的使用。仅在五六年以前，在数据库中存储影片文件、使用在线工具编辑和重新混合文件以及互联网上共享，几乎都是不切实际的，而现在，我们天天都在面对这些事情。看看 YouTube 网站和类似服务网站（例如 Eyespot、Grouper、Jumpcut、Video 等）的受欢迎程度就知道了。

www.TopSage.com

1.1.2 足够的安全和审核

就像前面提到的，企业数据是有价值的，并且必须被安全地保存。这意味着不仅要保护存储的数据，以免被恶意或无意地改动，比如未经授权的登录、意外的信息删除或修改等，还要保护其免受硬件故障和自然灾害的损坏。对于许多企业来说，数据库安全并不是“随时拥有”的特性，而是由大量不同级别的标准所管理的强制性要求，包括美国的联邦法律，例如，萨班斯奥克斯利法案（Sarbanes-Oxley Act of 2002，Sarbox）和健康保险责任法案（Health Insurance Portability and Accountability Act of 1996，HIPAA）。

1.1.3 多用户环境

为了更具实用性，存储在一个数据库中的信息必须对于大量同时处于不同安全级别的用户而言是可访问的，并且，数据必须保持一致。认识到这一点也是非常重要的。例如，如果两个用户试图同时改变相同的信息片，结果将是不可预知的（例如，数据讹误），因此，像这样的情形必须由内部的数据库机制进行适当的处理。同样，某些用户组可能被允许修改几个信息片、浏览信息的其他几个部分，但被禁止查看信息的某一部分（一些公司数据，例如雇员和顾客的私人信息，可以通过非常有限的访问权来进行严格的保密）。

1.1.4 效率和查找能力

用户需要快速地访问到他们需要的数据。因此，这就要求数据库不仅能够存储数据，而且还能以高效的算法来处理数据。例如，要在存储着数百万条记录的数据库中找出一个订购者，就不得不滚过每一条记录，这对于用户而言是不可接受的。所以，对数据库查询的响应必须是快速的，最好能在瞬间完成。

Note 注意：类似地，假定要找出一本书中所有“对象”这个词，可以手动地一页一页浏览全书，直到最后一页，也可以使用索引来确定这个词在哪些页面使用过。这种情形就类似于使用了坏的或好的编程算法。

1.1.5 可伸缩性

数据库必须具有灵活性，并能够轻松地适应不断变化的商业要求。这首先意味着在几乎不对其他对象和程序造成影响的情况下，可以对数据库对象的内部结构进行轻松的修改。例如，要在一个旧有数据库中添加一个字段，将不得不让整个数据集离线，也就是说，使用户无法访问数据库，然后修改数据库、改变并重新编译相关的程序，等等。我们将在本章的“早期数据库”部分中进行更详细的讨论。

另一个可伸缩性方面的问题是数据通常要比硬件和用来访问和操作它的软件寿命更长，因此，为了适应当前“短暂流行”的开发环境而不得不重新设计整个数据库将不会是非常方便的事情。例如，公司的决策权易手，或公司的管理层突然决定将产品环境从Java切换到C#。

另外，一个只有少数用户的小型私人企业可能会逐渐成长为一个拥有几十万甚至几百万用户的大公司。在这种情况下，能够将一开始小型服务器上简单廉价（甚至是免费）的数据库版本轻松地转换为强大的多处理器服务器上同一供应商的企业级版本，是非常有用的。

Note 注意：对于强大的多处理器服务器来说，最近非常流行的一种选择是“网格”计算概念。即，由多个节点集成一个虚拟的巨型计算机，每个节点是一个相对小型的服务器。这些节点可以从系统中轻松地添加或删除，显著地提高了系统的可伸缩性。

1.1.6 用户友好

数据库不仅仅只是为程序员和技术人员准备的（有些数据库可以说就不是为程序员准备的）。非技术用户构成了当今所有数据库用户的主体。会计师、经理、销售人员、医生和护士、图书管理员、科学家、技术员、客户服务代表等，对于所有这些以及更多的人而言，与数据库进行交互已经成为了他

www.TopSage.com

们工作中不可或缺的组成部分。这意味着数据必须易于操作。当然，大多数用户将通过图形用户界面来访问数据库，这个图形用户界面有着一套预定义的屏幕和有限的功能。但特别的数据库查询和报表正变得越来越流行，尤其是在那些高级计算机用户当中更是如此。

Note

注意：思考这样一种情况。一个订购管理应用程序有一个用来查看所有订购的屏幕以及另一个用来浏览客户的窗口。它也可以生成大量的报表，包括一个用来分析按客户分组的订单分析报表。会计师 Jerry 正在为他的老板制作一个报表，他需要找出 10 位债务最高的客户。他可以从 IT 部门请求一个新的报表，但由于繁琐的例行公事、程序员繁忙或其他一些原因，这将花费几天（甚至几周）的时间。SQL 知识可以帮助 Jerry 创建自己的特别查询，获得数据，并完成报表。GUI 工具（例如 TOAD 或 Win-SQL）的使用会让 Jerry 的工作更加轻松——在这种情况下，他根本不需要知道准确的 SQL 语法。只要 Jerry 能够指定需要检索的表和列以及这些表之间的关系，这种工具就能自动构建并执行需要的 SQL 语句。

1.2 选择数据库软件

市场上每一种 DBMS 都遵循本质上相同的基本原理，目前市场上数据库产品的种类非常多，对于没有坚实的数据库背景知识的人来说，要想确定应该学习和使用哪种数据库将是一件非常困难的事情。数据库市场充斥着各式各样的 RDBMS：IBM DB2、Oracle、Microsoft SQL Server、Sybase、MySQL 和 PostgreSQL，这些只是其中很小的一部分。

没有完全相同的两个系统。有的相对容易使用，有些则需要掌握严格的专业技术知识才能安装和操作；有些产品是免费的，而其他一些则相当昂贵，所有这些产品都附带着各式各样的其他一些小零碎儿，比如许可、专业技能的可用性等。在选择 DBMS 的过程中，没有单一的程式，但在做出选择时有几方面的因素可以考虑。下面是最常见的一些值得考虑的因素。

1.2.1 市场占有率

国际数据集团（International Data Group，IDG）旗下子公司互联网数据中心（Internet Data Center，IDC）的一份调查表明，在 2006 年，“三大”主要的 DBMS 供应商占领了超过 84%的数据库市场。Oracle 占 44.3%，IBM 大约为 12.2%，Microsoft SQL Server 为 18.6%。Sybase 以 3.2%的市场占有率位居第四，紧跟其后的是 Teradata（2.8%）。剩下的市场（小于 10%）则由多个小型供应商或非关系型“恐龙级 DBMS”共同分享。

同样值得关注的是，“三大”市场的份额一直在持续增长（结果当然只能是进一步挫伤其他更小的竞争者）。在 1997 年，“三大”数据库合在一起的市场份额还不到 70%。在 2001 年，其市场占有率增至大约 80%，而现在这个数字竟超过了 85%。

从 2005 年到 2006 年，整个 RDBMS 软件市场增长了 14.3%。Microsoft 以 25%的增长率处于领先地位，紧跟其后的是 Oracle（其增长率为 14.7%）。IBM 为 12%，稍稍低于平均值。

所有开源 RDBMS 供应商所占的份额是微不足道的。根据市场研究公司 Gartner Dataquest 的调查，在 2005 年，开源 RDBMS 供应商所占的份额不到数据库市场的 1%；然而，RDBMS 供应商的增长率为 47%，这多亏几种流行的数据库产品，如 MySQL 和 PostgreSQL。

1.2.2 总拥有成本

“三大”主要的实现在价位上可以说是同级别的，但根据包含的功能、用户数以及计算机处理器的不同，其价格会有很大的变化，变化范围从一千美元以下的许可较少的标准版本，直至允许无限用户访问的价值数十至数百万美元的企业级版本。许多小型数据库开发商的实现是免费的。此外，在过去的几年中，“三大”供应商都发布了自己的免费 RDBMS 版本。Oracle 首先在 2005 年提供了 XE 数据库，Microsoft 则免费推出了 SQL Server Express，而 IBM 最近发布了 DB2 的 Express-C 版本。

www.TopSage.com

而专业技术方面却是另外一回事。数据库专业技能是昂贵的，并且通常情况下是短缺的。平均而论，Oracle专业技能比同级别的Microsoft SQL Server或DB2的专业技能的价格要更高些。由开发商自己给出的总拥有成本（TCO）分析不可避免地会倾向有利于自己的那一边，因此，在把公司托付给一个特定的开发商之前，应该运用最佳的判断并做好该做的准备工作。不要在这件事情上犯错，这是一个长期的投入，如果在生产的中途更换数据库开发商的话，那将毫无疑问是一个极端痛苦且非常昂贵的事情。

1.2.3 支持和持久性

你可能会问为什么把数千美元花费在可以用一个免费的产品替代的东西上？答案非常简单，对于大多数的企业而言，最重要的事情是支持。他们付钱来购买公司的安全和股东们内心的宁静，外带所有那些伴随着一个有着如雷贯耳般名号的企业级产品而来的各式各样的锦上添花的小玩意儿（就像格言中说过的："从没有人为购买了IBM而被解雇"）。首先，当某些事情出错时，企业可以得到有资质的专家们提供的迅速而有针对性的支持。其次，管理层可以做一个合理的假设，像IBM，Microsoft或Oracle这样的开发商在未来的10年里依然会存在（当然，没有人能够保证这一点，但较之选择那些更小的竞争者，这样的选择毫无疑问看上去会更好些）。另外，"免费"的产品很少有较昂贵产品那样的规模，在一定数据量的大量订单上优化大量查询方面，"免费"的产品也很少像较昂贵产品那样易于管理、那样健壮和灵活。因此，由小型数据库开发商提供的价格便宜（而且有时是免费的）的产品可能为小型的企业、非营利的组织或非关键的工程所接受，但大型的公司绝不会考虑在他们的薪金或会计系统中使用这些产品。

Note 注意：应该注意到，由于产品越来越具依赖性和功能性，越来越多的企业在使用开源RDBMS以完成某项任务。以下是很好的例子：Sony Online Entertainment转向了EnterpriseDB（一个构建于PostgreSQL之上的企业级关系数据库管理系统），Google、Yahoo和Ticketmaster为它们的关键项目使用了MySQL。而且，RDBMS工具供应商已经开始为某些流行开源数据库产品提供支持。例如，Quest Software开发的TOAD（一种Oracle开发人员中流行的数据库工具）现在可用于MySQL。另外，一些专门从事软件支持的大型企业现在已开始为某些开源产品提供服务。

1.3 主要DBMS实现

一本书无法涵盖所有现存的数据库实现，因此，我们决定把关注的焦点集中在"三大"数据库供应商上：Oracle Database，用于Linux、UNIX和Windows的IBM BD2，以及Microsoft SQL Server。这些实现有很多共同的特征：它们都是颇具业界实力的企业级关系数据库（关系型数据库的模型和SQL标准会在本章的后面讨论），它们都使用由美国国家标准协会（American National standardization，ANSI）和国际标准化组织（International Organization for Standardization，ISO）规定的结构化查询语言（Structured Query Language，SQL），并且所有这三者都能够运行于Windows操作系统。Oracle在实际中的UNIX、Linux、Apple Mac OS X Server、IBM z/OS以及OpenVMS上也都是可用的；为Linux、UNIX和Windows开发的DB2可以运行于AIX、HP-UX、Solaris、Linux和Microsoft Windows上。

Note 注意：ANSI是一个私人的非营利性组织，这个组织管理和调整美国自发的标准化和一致性评估系统。ANSI的任务是通过促进和推动自发形成的标准化和一致性评估系统并保护其完整性来增强美国企业的全球竞争力和提高生活质量。ANSI（www.ansi.org）创立于1918年10月18日，它是国际标准化组织（ISO）和其他一些国际协会的美国官方代表。

问题是，以上提及的数据库中没有一个是百分之百遵从ANSI SQL的。这些数据库使用相同的SQL基本语法（尽管存在着一些差异），但语言运算符、命名限制、内部函数、数据类型（尤其是与日期和时间相关的那些）以及程序化语言扩展的实现是不同的。

www.TopSage.com

交叉参考：参阅第 14 章获得有关 SQL 程序扩展的更多信息。

表 1-1 对不同的数据库实现所支持的最大名称长度给出了一些数据对比。

表 1-1 一些数据库对象的最大名称长度限制

	IBM DB2 9.5	Microsoft SQL Server 2008	Oracle 11G
表名长度（字符）	128	128	30
列名长度（字符）	128	128	30
约束名长度（字符）	128	128	30
索引名长度（字符）	128	128	30
表列数	1012	1024	1000

在理想的世界里，标准将拥有至高的统治权，而为了人类的利益，SQL 可以自由地在不同的实现间共享。但遗憾的是，真实的世界看上去有些不同。尽管从所有的数据库开发商的产品中提取出一个标准的 SQL 是可能的，但使用特定实现所包含的功能会更好和更快地完成一些并不琐碎的任务。

1.4 现实中的数据库例子

说数据库无处不在只是一种保守的描述。实际上它们已经渗透到我们生活中的各个角落：在线商店、健康护理提供者、俱乐部、图书馆、音像店、美容院、旅行社、电话公司， 以及如 FBI，INS，IRS 和 NASA 这些政府机构，它们全部使用数据库。依据所属的类型和种类，这些数据库可以有很大的不同，并且，通常情况下，为了满足某些特定客户的需求，这些数据库都需要进行特别的设计。本节介绍如下一些例子。

注意：依照它们的主要功能，大多数关系数据库都可以分为两种主要类别——**在线事务处理**（OLTP）和**数据仓库系统**。OLTP 的典型表现就是大量用户同时创建和更新个人记录，换句话说，它是易变和计算密集型的。数据仓库是为进行信息处理和分析而设计的数据库，它专注于未来的计划而不是日复一日的操作。这些信息的改变不会非常频繁，从而为用户确保了信息一致性（可重复的结果）。真实世界中的大多数系统，如果不是刻意设计为数据仓库的话，都是这两种数据库的混合体。

1.4.1 订购管理系统数据库

一个典型的用于某个销售建材公司的数据库可能会进行如下的计划：公司一定有至少一个客户。数据库中的每一个客户都分配一个或多个地址、一个或多个联系电话以及一个默认的销售员，这个销售员负责客户与公司之间的联络。公司销售多种产品。每一种产品都有定价、描述以及其他一些特征。在某一时刻，针对一个或多个产品的订单可以被发出。每一个产品在逻辑上构成一个订单行。当一个订单完成后，可以发货，然后是开发票。发票号码和发货号码被自动地填充到数据库中，且用户无法改变这些号码。每一个订单都被分配有一个状态：COMPLETE、SHIPPED、INVOICED 等。数据库也包含特别的发货信息（提货单号码、发货的装箱号码、日期等）。通常一个发货包含一个订单，但数据库是以这样一种方式被设计的：一个订单可以被分布到一个以上的发货中去，同样，一个发货也可以包含一个以上的订单。数据库中也存在一些约束。例如，某些字段不能是空的，而其他一些字段只能包含某些特定类型的信息。

通过前面的定义，我们已经知道了数据库是一个多用户环境。通常需要依照用户所执行的功能以及他们被授予的安全级别对用户进行分组。在这里描述的订购管理系统可以有三个不同的用户组：销售部门的职员登录或修改订购及客户信息；发货部门的雇员创建和更新发货数据；仓库管理员处理产品。另外，通过使用基于 Web 的报表和特别的查询，所有三个用户组都可以以不同的视角查看不同的数据库信息。

我们把这个数据库称为 ACME，在整本书中我们将使用这个数据库作为例子和练习。ACME 数据

www.TopSage.com

库是一个真实的生产数据库的简化版本。它只有13个表，而实际生产数据库通常拥有几百个表。

交叉参考：参见附录B和附录F以获得ACME示例数据库的更详细的信息以及安装指导。

1.4.2 健康护理提供者数据库

一个健康护理提供公司在许多不同的州都有多个办事处。有许多医生为这个公司工作，而每一位医生都要照顾多个病人。有些医生只在一个办事处工作，而其他一些医生则在不同的日子里工作于不同的办事处。数据库保存着每一位医生的信息，比如姓名、地址、联系电话、专业领域等。每一位病人可以被指派给一位或多位医生。具体的病人信息（姓名、地址、电话、健康记录、出生日期、预约历史记录、处方、验血、诊断等）也被保存在数据库中。客户可以预约和取消预约，可以通过电话或利用公司网站来订购处方药品。也可以应用一些限制，例如，要看专家门诊的话，患者需要从他的主治医师那里获得一个批准；要订购处方，病人应当保留至少一个有效替换物；诸如此类的内容。

既然如此，那么主要的数据库用户组是什么呢？病人应该能够通过Web浏览器来访问数据库，以便订购处方和进行预约。这就是患者可以对数据库所做的一切了。医生和护士可以浏览有关他们的病人的信息、撰写和更新处方、安排验血和拍X光的时间等。行政人员（比如接待员、药房助手）可以为病人安排预约、填写处方以及给出详细的报告。

同样，在现实生活中，这个数据库将会复杂得多，并且会有更多的商业规则，但目前我们的主要目的是对数据库应该包含怎样的信息给出一个大致的概念。

健康提供者和订购管理系统数据库都是典型的**混合数据库**的例子(尽管前者可能更新近于OLTP)。

1.4.3 视频共享和编辑数据库

在线的视频共享服务，比如YouTube、Eyespot和Jumpcat，都使用数据库来存储、检索和编辑视频文件。用户能够以不同的格式保存自己的视频，能够共享这些视频或将其设为私有的，以及当他们不再需要时能够删除这些视频剪辑。

这种数据库的两个主要的要求是容量（媒体文件会相当大，而且视频共享服务的会员人数每个月都在增加）和性能（仅仅存储大量的二进制文件是不够的，用户应该能够在找到他们需要的文件并在Web浏览器中进行播放）。

这种数据库也是一个混合数据库。但在查找和检索数据的能力比快速更新和插入新记录的能力更重要的情况下，它又接近于数据仓库。

1.4.4 科学数据库

用于基因组研究和分子与细胞生物学相关研究领域的数据库是一个很好的科学数据库的例子。它包含了用于完全基因排序和局部基因排序、基因图谱以及生物体信息的基因目录，同时，还包含了数据库中所有生物体已知基因中序列相似性的有关数据以及有关细胞中的分子网络和化合物及反应的信息。

这个数据库只有一个用户组，所有的研究人员对所有信息都拥有同样的访问权，该数据库可以作为数据仓库的例子。

1.4.5 非营利组织数据库

一个老爷车俱乐部的数据库是一个不错的例子。毫无疑问，那样的一个组织通常不会有太多的会员。因此，数据库不会很大。需要保存会员的个人信息，如地址、电话号码、兴趣爱好等。数据库也可能会包含有关汽车的信息（品牌、年代、颜色、车况等）。汽车被链接到它们的拥有者（俱乐部的会员）。每一位会员可以拥有一部或多部汽车，而每一部汽车只能被一位会员所拥有。

这个数据库将只有几位用户，可能包括俱乐部的会长、一位助手以及一位秘书。

最后的两个例子不是关键的商业数据库，因此不必用昂贵的企业级软件实现。但这些数据仍然需要被安全地保存，不能丢失，但万一（让我们假设一下）硬件出了故障，在数据库从备份恢复前，用

www.TopSage.com

户可能要等待一两天的时间。由此看来，使用像 MySQL、PostgreSQL 这样的免费数据库，甚至非关系的 Posgres 数据库都是能够满足需要的。另一个不错的选择是 Microsoft Access，它是 Microsoft Office Tools（专业版或更高级的版本）的一个组成部分。Microsoft Access 在不超过 15 个用户的情况下会工作得很好。

1.5 早期数据库

平面文件、层次以及网状数据库通常被称为**早期**数据库。它们描绘了早期——大约 40 年前人们组织信息的方法。

1.5.1 平面文件数据库

平面文件数据库可能是最早的数据库管理系统之一。平面文件背后的想法是非常简单的：一个单一的、通常是非结构化的数据文件。它反映了“远古时期”计算机出现之前的数据存储系统，如笔记本、食品杂货店的商品名录等。可以把平面文件数据库比做一个办公桌的抽屉，可以用这个小抽屉存储各种物品，账单存根、信件、硬币等。尽管把信息放到里面只需要付出非常小的努力，但这样一个“设计”使**提取**信息变成一场噩梦，因为为了搜索所需的一条记录，将不得不依次查看所有的记录。把相关的数据放置到分离的文件中，甚至将其组织成不同的表（想像一个文件柜）也只能稍微减轻痛苦，因为这些做法都没有移除主要的障碍：数据冗余（相同的信息，或更糟糕的，**认为相同的**信息可能在不同的文件中被保存了一次以上）、缓慢的处理速度（“我知道它在那里的某个地方……”），以及在存储和检索数据时容易出错。此外，从根本上说使用这个数据库，需要有对数据库结构的深入了解，假如在费用文件中搜索订购信息，搜索将完全失效。

例如，我们可以为订购目录系统设计一个平面数据库系统，用以收集有关客户、客户订单以及客户订购产品的信息。如果数据按顺序存储的话，文件将包含有关客户信息，然后是订单和产品的信息，之后是一些新客户的信息等。订单中的所有数据都已输入（参见表 1-2）。且不说大量的单元格将会空置（将用什么来填充“Ace Hardware”的“数量”列或“Nails”的“地址”列？），只需想像一下，要从这堆杂乱无章的数据中提取出任何有意义的信息将有多么困难。

表 1-2　　平面文件记录示例

名称	类型	地址	价格	数量
Nails	Product	n/a	100	2000
Ace Hardware	Customer	1234Willow Ct Seattle，Washington	n/a	n/a
Cedar planks	Product	n/a	2000	5000

对这些缺点的不满意刺激了数据库存储和检索系统领域的发展。

注意：Excel 经常被用于创建平面文件数据库。

1.5.2 层次数据库

层次数据库的概念大约出现于 20 世纪 60 年代，而且令人难以置信的是直至今天它仍在使用。层次模型是相当直观的；正如它的名字所暗示的那样，它按层次结构来存储数据，类似于一个族谱、组织系统图或金字塔。可以想像一个计算机文件系统，这个系统是经由一些图形界面来呈现的。

最流行的层次数据产品是 IBM 的信息管理系统（Information Management System，IMS），这套系统运行于大型计算机上。自从 1968 年第一次被引入，这套系统就一直存在（期间经过了多次改进），这主要是由于层次数据库为某些类型的查询提供了惊人的速度性能。

层次数据库是基于“父/子”样式的，在层次数据库中，每个父表可以拥有多个子表，但每一个子表有且只有一个父表。可以把这一结构想象成一棵上下颠倒的树，从根（树干）开始，在许多的层次上长出枝条（参见图 1-1）。

www.TopSage.com

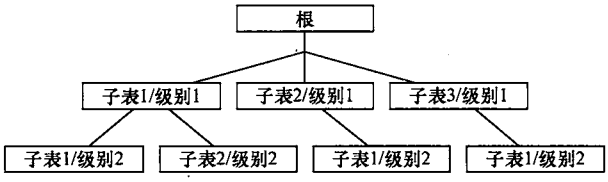


图 1-1 层次结构

因为子表中的记录是经由层级体系进行访问的，因此，如果父表中没有相应的指针记录，子表中就不会与之对应的记录，这个层级体系一直上溯至根。这个层级体系可以比做文件管理系统（就像是在 Windows 资源管理器中看到的那种目录结构视图），要想访问目录中的文件，必须首先打开包含那个文件的文件夹。

Note 注意："表"不是层次数据库的术语。它首先出现在关系数据库模型中，这一模型将在后面的章节中进行讨论。尽管如此，本书仍然使用"表"来指代层次数据库和网状数据库的"存储区域"（结构化文件）。毕竟，表可以被简单地定义为以行和列组织的一系列元素的集合。

让我们对先前讨论过的平面文件模型做一些改进。这里不是把所有的信息倾倒进一个单一的文件，而是把这些信息分离到三个表中，每一个表包含了相关信息：企业名称和地址保存在 CUSTOMER 表中，产品描述、品牌名称和价格保存在 PRODUCT 表中，而 ORDER_HEADER 表用来保存订单的详细信息。

层次数据库模型极大地减少了冗余（与平面文件数据库模型相比）：有关客户、产品等信息仅存储一次。ORDER_HEADER 表（参见图 1-2）将包含指向客户和这个客户已经订购的产品的指针。无论何时，如果需要查看某位客户购买了什么产品，都可以从 ORDER_HEADER 表开始，找到所有发出订单的客户的识别号列表以及每一位客户的产品识别号列表；然后，利用 CUSTOMER 表，可以找到客户的名称，随后，可以从 PRODUCT 表中利用已经获得的产品识别号来获得有关产品的描述。

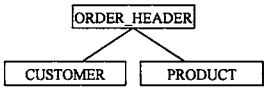


图 1-2 层次数据库例子

唯一的问题是这种检索信息的方式不够直观（无论请求的是什么信息，都必须从根开始，这里是从 ORDER_HEADER 表开始）。如果只需要客户的名称，那么这个层次数据检索速度会很快，从一个父表直接到达一个子表。要从一个层次数据库获得任何信息，用户必须对这个数据库的结构有深入的了解；并且这个结构本身是毫无灵活性可言的。举个例子，如果决定客户必须通过第三方发出订单，将必须重新调整所有的关系，因为 CUSTOMER 表将不再与 ORDER_HEADER 表有任何关系，并且，为了纳入这多出来的一步，即找到销售这个产品的代理商，然后再找到购买了这个产品的客户，必须重写所有的查询。这也促成了一个显而易见的事实——还是没有摆脱冗余的问题：如果一位客户通过一个以上的销售代理商进行订购，您将不得不为大量客户表中的每一个代理商复制所有的信息。

但如果需要添加从未发过订单的客户，或是要添加从未被任何客户订购过的产品时会发生什么事情呢？答案是做不到这一点，因为层次数据库不具备这种能力，如果父表中没有指向某个子表的指针，则层次数据库就无法在这个子表中存储信息。按照层次定义，层次数据库中既不能存在没有被订购过的产品，也不能存在没有发出过订单的客户，这显然不符合现实。

层次数据库在处理一对多关系（如第 2 章中所述）时表现得很好。但不管怎么说，在很多情况下，我们都希望子表能够与一个以上的父表建立关系，即一个产品不仅能出现在多个订单中，而且一个订单中也可以包含多个产品。这个问题在层次数据库的领域里没有答案（至少不是一个容易的答案）。

1.5.3 网状数据库

试图解决与层次数据库相关的问题促成了网状数据库模型的出现。网状数据库模型起源于名为数据系统语言联合会（Conference on Data System Languages，CODASYL）的组织，这个组织是 1957 年由美国国防部成立的。CODASYL 负责推动 COBOL（它是第一批得到广泛流行的编程语言中的一种），并于 1971 年发布了网状数据库标准。当时，最流行的网状数据库商业实现是 Adabas（早已被转换为

www.TopSage.com

关系模型）。

网状模型类似于层次模型，也是基于父/子关系概念的，但它解除了一个子表有且只有一个父表的限制。在网状数据库模型中，一个父表可以有多个子表，而一个子表也可以有多个父表。可以把这种结构想象成几棵树共享一些分支。在网状数据库的术语里，这些关系被称作“集合”。

除了能够处理一对多关系外，网状数据库还可以处理多对多关系。

Cross-Reference 交叉参考：一对一、一对多、多对多关系在第 2 章中进行阐述。

同样，数据访问也不必从根开始。可以从任何数据元素开始遍历该数据库结构，并可以在任何方向上导航到一个关联的数据元素——假设在两个方向上都有显式的链接（参见图 1-3）。

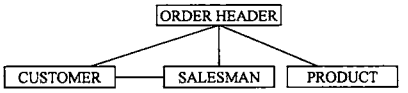


图 1-3 网状数据库例子

在这个例子中，要找出哪些产品被卖给了哪些客户仍然必须从 ORDER_HEADER 表开始，然后继续在 CUSTOMER 表和 PRODUCT 表查找，这一点与层次模型相同。但当客户通过一个以上的代理商发出订单时，情况将大为改善：不必再遍历代理商就能列出特定产品的客户，搜索记录时也无需从根开始。

尽管具备一些优势，但网状数据库还是存在几个与层次数据库同样的问题。二者都不具备灵活性，任何对结构的改变（例如，添加一个新表以反映已改变的商业逻辑）都需要重建整个数据库；同样，集合关系和记录结构都必须预先定义。

网状和层次数据库的主要劣势是它们都是程序员领域的。要回应一个最简单的查询，也不得不创建一个程序，这个程序定位数据库结构并最终输出结果；不同于 SQL，这个程序是用程序化语言写成的，这些程序化语言常常都是一些归各个公司所有的专利语言，同时，使用者还需要具备数据库结构和当前操作系统两方面的深厚知识。毫无疑问，这样的程序不具有可移植性，而且编写这些程序将花费大量的时间（以今天的标准来看）。

1.6 关系数据库

由网状和层次数据库的能力匮乏最终导致了**关系数据模型**的出现。关系数据模型在网状数据库的概念上又前进了几步。与层次和网状模型相同，关系型模型是基于二维存储空间（表）的，这些表基于共有的一个字段（或一系列字段）而互相关联。然而，现在这种关系是由列值来实现的，而不像以往是由低级的物理指针来定义。

Note 注意：人们头脑中普遍有一个错误概念，那就是术语“关系”来源于表之间的关系。实际上，“关系”这个词是一个数学术语，在一定条件下，可被认为是“表”。

1.6.1 表

表是关系数据库的基本的组成单元。它组织数据的方法相当直观，并且已经存在了几个世纪。一个表由行和列组成（用非关系型数据库的术语来说是**记录**和**字段**）。数据库中的每一个表都有一个**唯一**的名称（它必须是唯一的**完全限定名称**，以模式或数据库名称作为前缀）。

Note 注意：一个完全限定名称里的点（.）符号通常在编程中用于描述对象和它的属性之间的层次结构。这个点符号不仅可以用于数据库对象，也可以用于结构、用户定义类型等。例如，某个 Microsoft SQL Server 数据库中的表字段可以以 ACME.DBO.CUSTOMER.CUST_ID_N 的形式来引用，在这里，ACME 是数据库名称，DBO 是表所有者（Microsoft 标准），CUSTOMER 是表名，而 CUST_ID_N 是 CUSTOMER 表中的列名。

Cross-Reference 交叉参考：参阅第 4 章获得有关表和其他数据库对象名称的更多信息。

表中的每一个字段都有一个唯一的名称，任何表都至少拥有一个字段。一个表的字段数通常是有

www.TopSage.com

大家网 TopSage.com

限的，实际的限制决定特定的实现。不同于早期数据库结构，表中的记录并不按特定的次序来存储和检索，尽管可以使用**聚集索引**（聚集索引将在第 4 章中进行讨论），对记录排序。在关系数据库系统（RDBMS）中排列记录的任务是由 SQL 来完成的。

一条记录是由许多单元组成的，每个单元都有一个唯一的名称并可以包含一些数据。没有记录的表被称为空表。

字段中的数据必须是同一类型的，例如，字段 AMOUNT 只包含数字，而字段 DESCRIPTION 只包含词。一个字段中数据的集合被称为列的**域**。

Note 注意：早期的关系或其他类型的数据库被只能包含文本数据；现代数据库可以存储能够被转换成二进制格式的一切事物：图片、电影、音频记录等。

好的关系设计能确保一条记录能够描述一个**实体**（entity）（另一个新的关系数据库的术语，将在本书的后面部分进行讨论，但在这里有必要提及它）。换句话说，记录不应该包含不相干的信息：例如，CUSTOMER 表只处理客户信息，它的记录不应该包含这个客户订购的产品等信息。

Note 注意：把相关的数据收集到一起的过程消除了冗余，这个过程被称为规格化，将在第 2 章进行讨论。规格化并不是 SQL 本身的一部分，但的确影响了 SQL 的查询效率。

一个表可以拥有的行数在理论上没有限制，尽管某些实现强加了一些限制。同样，考虑到实际情况（这些情况是应该被考虑的），还是需要有一些限制：数据检索的速度、存储的总量等。

1.6.2 关系

RDBMS 中的表可能是关联的也可能不关联。像前面提到的，RDBMS 是建立在表之间关系上的，但与早期数据库（层次和网状数据库）不同，关系数据库中的关系完全基于表列中的值；这些关系具有逻辑意义，而不是低级的计算机特定指针。让我们用一个假想的订购项目数据库（一个我们将贯穿全书设计、建立和使用的数据库）作为例子。ORDER_HEADER 表被关联到 CUSTOMER 表，因为这两个表都有一个**共同的值集合**：ORDER_HEADER（和它的值）中的字段 ORDHDR_CUSTID_FN（客户 ID）对应于 CUSTOMER 中的 CUST_ID_N。字段 CUST_ID_N 是 CUSTOMER 表的**主键**，而它对于 ORDER_HEADER 表而言是**外键**（以字段 ORDHDR_CUSTID_FN 的形式）。

1.6.3 主键

主键在 RDBMS 中身兼数职。如前面所提到的，这是关系的重要组成部分。它还可以存储二进制数据，比如部门号、零件编码或员工识别号；但它主要的任务还是在表中唯一地标识每一条记录。

在早期数据库时代，记录总是按预定义的次序存储；如果这样的次序必须被打破的话（可能是因为有人以错误的次序插入了记录或是商业规则被改变），那么，整个表（并且很可能是整个数据库）必须被重建。RDBMS 废除了用于记录的固定次序，但仍然需要一些机制对记录进行唯一性标识，而主键正是为这一目的服务的。它基于一个字段（多个字段）思想，而这些字段包含唯一值。

Note 注意：在一个 RDBMS 中，主键只是候选键的一个特殊情况。候选键是指表中其每一行都拥有唯一值的列的集合。每个表可以有任意个候选键，但只有其中的一个可以被声明为主键。

主键的性质决定其不能为空，这意味着在定义了主键的表中，主键字段的每一个记录都必须包含非空数据。

Note 注意：尽管主键不是必需的，但最好为每个表设置主键。事实上，许多 RDBMS 实现都会在创建了一个表却没有定义主键时发出警告。一些纯化论者甚至指出主键应该是无意义的，应该使用生成的唯一值（比如 EMPLOYEE_ID），而不是像社会安全号码这样的数据（尽管这些也是唯一的）。

主键可以由一个或多个列组成。也就是说，尽管某些字段可能包含相同的值，但这些字段的联合体（集合）在整个表中仍然是唯一的。一个由几个列组成的键被称为**组合键**。

www.TopSage.com

Note

注意：尽管主键是在 RDBMS 中定义关系的基石，但现实中的实现（尤其是那些早期的）并不总是为这一逻辑概念提供内置支持。实际上，强制选定的主键具有唯一性是程序员的责任（比如，要求程序员在插入新的记录前检查已存在的值）。今天，所有主要的关系数据库产品都有对主键的内置支持，在一个非常基础的层次上，这意味着数据库会对违反唯一性约束的行为进行检查，任何时候如果试图插入一个重复的记录，数据库都会给出一个错误。第 4 章详细地讲述了这一问题。

1.6.4 外键

让我们回到 CUSTOMER 和 ORDER_HEADER 表。至此，已经明白了为什么 CUST_ID_N 被指定为主键——它具有唯一的值，任何客户只能拥有一个 ID，而且每个 ID 只能分配给一个客户。要跟踪哪位客户发出了哪些订单，需要在客户和他们的订单间提供连接。

表 ORDER_HEADER 拥有自己的主键 ORDHDR_ID_N，这个主键可以唯一地标识订单；此外，它将拥有一个外键——ORDHDR_CUSTID_FN 字段。这个字段中的值对应 CUSTOMER 表中 CUST_ID_N 主键字段中的值。注意，不同于主键，外键不必是唯一的——一个客户可以发出几个订单。

现在，通过查找 ORDER_HEADER 表，可以找到发出特定订单的客户。ORDER_HEADER 表通过外键的值关联到 CUSTOMER 表。这样，很容易基于订单找到一个客户或为客户找到他的订单。不需要了解数据的布局、理解表中记录的次序，或是精通某些低级的专门编程语言，就可以查询数据。现在，可以用类似英语的标准语言——结构化查询语言（SQL）来执行即席查询。

1.6.5 RDBMS 的发展

虽然关系数据库模型具有明显的优势，真正投入使用却需要花费一些时间。主要的原因之一是硬件。逻辑上清晰而纯粹的模型实现起来却并不是一件轻松的事情，与早期数据库相比，它需要更多的内存和更强大的处理能力。

大中型企业对收集、保存和分析数据的需求推动了关系数据库的发展。1965 年，Intel 公司的共同创始人之一 Gordon Moore，给出了他著名的观察报告，在这份报告中他指出 IC 上每平方英寸的晶体管数每年翻一番。令人吃惊的是，直到今天这个法则依然适用。更强大的机器使得实现和销售 RDBMS 成为可能；廉价的内存和强大的处理器让它们运行得飞快；对信息的不断增长的欲望让 RDBMS 产品成为了一件日常用品，价格得到了大幅度的下降。根据某些评估，只有不到 8%的市场还把持在“恐龙级的”早期数据库中——主要是由于这些数据库的拥有者是在 25 年前做出这一重大投资的，他们的市场份额正在逐渐减少。

不管怎样，关系数据库系统已经在这一领域处于统治地位。

1.7 其他 DBMS 模型

20 世纪 80 年代末期时髦的口号是**面向对象编程**（object-oriented programming，OOP）。与 RDBMS 并未得到广泛采用的原因（内存需求和处理能力）类似，面向对象编程当时并未得到广泛应用，直到进入 20 世纪 90 年代。OOP 语言基于这样的概念：可以基于对象对编程问题进行建模。

在程序的编码几乎相同的情况下，组织编码的方法明显地改变了。OOP 还改变了构建、编码和执行程序的方式。对于与数据库进行交互的应用程序来说，这是很自然的事情，即在现有基础上存储对象，而不是将对象分解到文本或文本的部分字段中，然后在需要的时候将它们一起放回原处。

现代 RDBMS 具有存储二进制对象（例如，图像、声音等）的能力。面向对象（OO）数据库需要储存概念对象：客户、订单，等等。20 世纪 90 年代末期，所有情况都表明面向对象数据库正在成为继 OOP 以后的另一件大事。然而，令人吃惊的是，从那以后，面向对象编程的热度就开始逐渐降低。虽然市场上有几种纯粹的面向对象数据库系统（object-oriented database system，OODBS）的产品，但没有一种产品得到广泛采用，看起来重点逐渐转移到了面向对象的关系数据库管理系

www.TopSage.com

统（object-oriented relational database system，OORDBMS），该系统将面向对象特性与传统的RDBMS属性结合了起来。

Note 注意：大约在1999年，面向对象特性成为了SQL标准的一部分。“三大”RDBMS供应商都至少部分地遵从了面向对象标准。

值得关注的另一个发展是可扩展标记语言（eXtensible Markup Language，XML）的广泛应用。XML发展为标准规范化标记语言（Standard Generalized Markup Language，SGML）的一种逻辑简化，一定程度上是由于简单静态超文本标记语言（HyperText Markup Language，HTML）几乎把精力全部放在了表现上，这种情况下，就需要一种能够提供结构和语义标记的标记语言。XML文档包含平台独立的、行业标准格式的自述性数据，这使得它转换为其他类型的文档、在不同网络中搜索或传输变得很容易。

Cross-Reference 交叉参考：第15章将对XML进行讨论。

XML首先在2003年成为SQL标准的一部分。SQL：2003包含一个描述SQL XML标准的独立文档（第14部分：XML相关规范）。该文档2006年被主修订版代替（主要是对XQuery的支持），另一个修订版本正在筹划当中，计划在2008年发布。

Note 注意：每个主要的RDBMS版本都有自己产品的最新版本或现有版本的插件，用于对XML进行处理并在某种程度上遵从SQL：2003标准。

虽然我们不可能预知未来什么模型将以胜利者的身份出现，但现在看起来，关系数据库至今已经历了长时间的发展，并且还没有充分发挥其潜能。因此，作为RDBMS语言的SQL将在数据库世界中继续保持其举足轻重的地位。

1.8 SQL和SQL标准简史

就像我们前面所讨论的，预定义关系的数据库没有一套用来操作数据的标准命令。每一个数据库或者拥有它自己的专利语言，或者使用COBOL、C等语言编写的程序来操作记录。这些数据库灵活性较差，而且，如果不使数据库脱机并重写大量代码就不允许任何内部结构更改。在20世纪60年代以前，这样的工作方式还算有效，因为那时绝大多数的计算机应用程序都是基于严格的批处理过程的（从开始运行到结束而没有用户的交互）。

1.8.1 卑微的开始：RDBMS和SQL的演变

在20世纪70年代初，联机应用程序（需要用户交互的程序）的发展触发了对更具灵活性的数据库的需求。某条记录需要额外字段或子字段数量超出了文件设计最大数目的情形变得越来越普遍。

例如，想像CUSTOMER记录集有两个固定长度的字段，ADDRESS1（开账单地址）和ADDRESS2（发货地址），它被所有的客户使用了一段时间。但如果有一位客户已经拥有一家硬件商店，现在又买下了另一家商店，情况将会怎么样呢？至此这个记录必须拥有一个以上的发货地址。再比如，如果有了一位新客户，WILE ELECTRONICS INC.，这位客户拥有十家商店，该怎么办？现在，有两种选择。第一种选择，把整个的数据集脱机，修改设计，并更改或重新编译处理这些数据的所有程序。这会满足那些拥有十家商店的客户的需求，但另一方面，其他只拥有一家商店的客户都将在他们的记录中拥有九个不必要的字段（参见图1-4）。此外，没有人能够保证明天其他的客户不会买下比如说25家商店，如果那样的话，将不得不再次重复这个过程。另一个选择是为WILE ELECTRONICS INC.添加十个同样的记录，除了发货地址外，其余的部分都是完全多余的（参见图1-5）。仍然必须修改程序否则可能返回错误的结果。

NAME	BILLADDR	SHIPADDR_1	SHIPADDR_2	SHIPADDR_N	SHIPADDR_10
MAGNETICS USA INC.	123 LAVACA ST.	444 PINE ST.			
WILE ELECTRONICS INC.	411 LONDON AVE.	232 EEL ST.	454 OAK ST.	...	999 ELK AVE.

图1-4 用多列来解决CUSTOMER的多地址问题

可以看到，大部分问题的症结在于数据库的结构，这种数据库通常仅包含一个文件，该文件由固

www.TopSage.com

定长度的记录组成。解决方案是把数据分散到几个文件并在需要时重新装载请求的数据。正如本章前面所提到的，层次和网状数据库模型都试图朝这个方向努力，但它们仍然有太多的缺陷，因此，关系模型就成为了最受欢迎的技术。我们刚刚讨论的问题在关系数据库中得到了解决。在关系数据库中，CUSTOMER 和 ADDRESS 是分离的实体（表），通过主/外键关系联系在一起（参见图 1-6）。唯一要做的就是添加需要的任意多条 ADDRESS 记录，这些记录有一个引用其父表的外键（参见图 1-7）。

NAME	BILLADDR	SHIPADDR
MAGNETICS USA INC.	123 LAVACA ST.	444 PINE ST.
WILE ELECTRONICS INC.	411 LONDON AVE.	232 EEL ST.
WILE ELECTRONICS INC.	411 LONDON AVE.	454 OAK ST.
WILE ELECTRONICS INC.	411 LONDON AVE.	456 WILLOW ST.
WILE ELECTRONICS INC.	411 LONDON AVE.	678 MAPLE AVE.
WILE ELECTRONICS INC.	411 LONDON AVE.	332 WALNUT ST.
WILE ELECTRONICS INC.	411 LONDON AVE.	531 DEER ST.
WILE ELECTRONICS INC.	411 LONDON AVE.	865 CEDAR AVE.
WILE ELECTRONICS INC.	411 LONDON AVE.	911 MYRTLE ST.
WILE ELECTRONICS INC.	411 LONDON AVE.	777 SITKA AVE.
WILE ELECTRONICS INC.	411 LONDON AVE.	999 ELK AVE.

图 1-5　用多记录来解决 CUSTOMER 的多地址问题

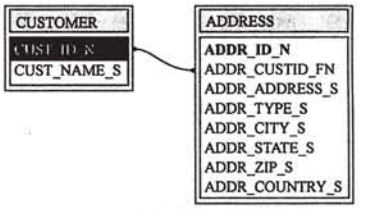


图 1-6　表间的主/外键关系

CUST_ID_N	CUST_NAME_S
7	WILE ELECTRONICS INC.

ADDR_CUSTID_FN	ADDR_ADDRESS_S	ADDR_TYPE_S
7	411 S LONDON AVE.	BILLING
7	454 OAK ST.	SHIPPING
7	678 MAPLE AVE.	SHIPPING
7	999 ELK AVE.	SHIPPING
7	777 SITKA AVE.	SHIPPING
7	911 MYRTLE ST.	SHIPPING
7	865 CEDAR AVE.	SHIPPING
7	531 DEER ST.	SHIPPING
7	332 WALNUT ST.	SHIPPING
7	456 WILLOW ST.	SHIPPING
7	232 EEL ST.	SHIPPING

图 1-7　在关系模型中解决多客户地址问题

Note

注意：这个例子看起来可能不是很令人信服，几乎就像把相同的记录添加到独立的 ADDRESS 表中而不是把新的发货地址添加到 CUSTOMER。事实上，两者之间的差异是巨大的。在一个现实中的早期数据库中，CUSTOMER 文件将不仅只有 NAME 和 ADDRESS 字段，而是包含大量的其他信息：订单、产品、发票、发货等。每当访问数据库时，所有这些内容都将被重复加载，即便是添加一个新的发货地址这样简单的操作也是如此。

关系架构的另一个优势是简化了数据处理应用程序的逻辑。例如，假定非关系型 CUSTOMER 数据集的字段最多拥有 5 个客户订单（顺便说一下，经由订购，它很可能变成几个字段）。如果想要显示某个特定客户的所有订单，程序将不得不滚过所有这些字段，确定哪些是非空的，并显示它们的内容。在关系数据库中，只需显示 ORDER-HEADER 表中所有拥有这个客户编号的记录。这种关系架构使得即席查询语言相对容易编写，并最终导致了 SQL 的出现。

关系数据库的概念以及由此引出的 SQL，最初是由 IBM 的一名研究人员 Edward Frank Codd 博士于 1970 年在他的题为“用于大型共享数据库集的关系数据模型”的论文中提出的。简单来说，他有关关系数据库模型的想法的基础是数据独立于硬件和存储实现，以及一种用于访问数据的非程序化高级计算机语言。问题是，IBM 已经发布了它自己的产品，这一被称为 IMS 的产品是 IBM 唯一的战略性数据库产品；公司的管理层不值得花费金钱和精力去开发基于关系架构的新的商业软件。并且，新的数据库产品也存在着对 IMS 销售的潜在冲击。

尽管如此，一个被称为 System R 的关系数据库原型还是在 20 世纪 70 年代末最终由 IBM 提出了，但它一直就没能成为商业产品，而更多的是作为一个科学兴趣，相反，其数据库语言 SQL（最初被称为 SEQUEL）最终成为了所有关系数据库的标准（经过多年的发展之后）。同一时期的另一个关系产

www.TopSage.com

品是由科学家们在加州大学伯克利分校一个由政府资助的项目中开发的，这个产品被称为 Ingres，它也拥有自己的非程序化语言——QUEL，类似于 IBM 的 SQL。

但第一个商业关系数据库既不是 System R 也不是 Ingres。Oracle 公司于 1979 年发布了它的第一个产品，紧跟其后的是 IBM 的 SQL/DS（1980 年 1 月 8 日）和 DB2（1982 年 3 月 8 日）。Ingres 的商业版本也在 20 世纪 80 年代早期出现。Sybase 公司于 1986 年发布了其产品的第一个版本，而在 1988 年 Microsoft 发布了 SQL Server。其他公司开发的许多产品也陆续发布，但它们在现在市场上的份额是很小的。

1.8.2 SQL 标准简史

20 世纪 80 年代末，关系数据库模型缓慢但稳健地成为了行业标准。问题是，尽管 SQL 成为了一种得到普遍公认的数据库语言，但主要供应商的实现间的差异却在不断增大，因此，一些标准还是必需的。

大约在 1978 年，数据系统和语言委员会（Committee on Data Systems and Language，CODASYL）受托开发一种网络数据模型，以作为未来数据库实现原型。这项连续的工作由数据定义语言委员会（Data Definition Language Committee，DDLC）于 20 世纪 70 年代初开始。在 1982 年，这些努力以数据定义语言（DDL）和数据操作语言（DML）标准的提出达到了顶峰。四年后它们成为了标准——是由一个组织签字认可的，该组织即美国国家标准协会的 X3H2 技术委员会（American National Standards Institute's Technical Committee X3H2）。

Note 注意：该组织的名称被更改过几次。随着 Internet 的出现，字符"X3"（它没有任何实际意义，仅仅是一个标识符）成为人们查找色情（"XXX"）电影的部分途径，因此"X3"被更改为 NCITS（美国信息技术标准国家委员会，National Committee for Information Technology Standards）。为了确保人们能领会其全部含义，几年之后，该名称被更改为 INCITS，其中用 "International" 代替了 "National"。

1. SQL-86/87（SQL1）

1982 年，X3H2 被授权用来标准化关系数据模型。这个工程最初是基于 IBM SQL/DS 规范的，并在一段时间内紧跟 IBM DB2 发展的脚步。考虑到数据库产品供应商之间的诸多差异，在 1984 年，这个标准被重新设计以更具一般性。顺利通过所有的官方环节之后，1986 年，它终于被批准为美国国家标准协会的标准。1987 年，国际标准化组织采用了该标准。两年之后，发布了其修订标准，通常被称为 SQL-89。

2. SQL-89（SQL1.1）

SQL-89（或称 SQL1）是一个最低限度标准，在 1989 年，它在当时所有已存在的 RDBMS 的包围之中诞生了。它仅包括有限数量的功能，这些功能是关于在关系数据库中查找和检索信息的。各大主要的商业供应商没能在实现的细节上达成共识，而且直到今天在一定程度上仍没有统一，因此，SQL-89 标准中有很多都有意被保留为未完成，并且大量的功能被标记为由实现者定义。

3. SQL-92（SQL2）

由于上述原因，前一个标准被修订了。1992 年，发布了第一个统一的 SQL 标准，即 SQL-92 或称为 SQL2。ANSI 把 SQL-89 作为基础，但纠正了它里面存在的几个缺陷，填补了老标准中的许多缺口，并提出了概念性 SQL 特征，在当时，这些特征超越了任何已有的 RDBMS 实现的能力。同样，SQL-92 标准的文本也比前一个标准长了五倍多（足有 500 多页），并具有三个级别的一致性。

入门级一致性基本上就是改良过的 SQL-89。两者的差异甚微，例如，在 SQL92 中，对视图实施 `WITH CHECH OPTION` 的规范已明确规定。

中间级一致性是一套重要的改进，它包括但不限于：对用户命名的约束，对不同长度字符和国家字符集的支持，case 和 cast 表达式，内置连接运算符，动态 SQL；能够修改表，设置事务，在可更新的视图中使用子查询，以及使用集合运算符（`UNION`、`EXCEPT`、`INTERSECT`）来联合多个查询的结果。

完全级一致性包括了一些真正的高级功能，包括可延迟约束、断言、临时的本地表、字符集和域

www.TopSage.com

的特权。

一致性测试是由美国商业部的国家标准和技术协会（National Institute of Standards and Technology，NIST）的政府部门来执行的。供应商们在这件事情上非常配合，因为一个于 20 世纪 90 年代初通过的公开法案要求 RDBMS 产品必须通过这个测试，才能被联邦机构认可。

Note 注意：1996 年，NIST 废除了该一致性测试程序（这项决定背后隐藏的原因是测试所带来的“高成本”）。从此，对 SQL 标准一致性的唯一验证只能来自于 RDBMS 供应商本身。可以理解，这就使供应商特定功能和标准功能的非标准实现越来越多。

4. SQL:1999（SQL3）

SQL:1999 代表了 SQL 标准发展的另一个阶段。对这个标准的定义实际上在其前任 SQL 标准 SQL-92（SQL2）被采用一年之前就开始进行了。这个新的标准是在 ANSI 和 ISO 委员会的共同指导下开发的，由 SQL3 引入到数据库世界中的改变就如同从非关系数据库变为关系数据库模型那样惊人，其复杂性反映在描述这个标准的文本的页数上（2000 页，而 SQL-89 大约 120 页，SQL-92 大约为 600 页）。某些已定义的标准（例如，存储过程）是作为供应商特定扩展而存在的，某些标准（比如 OOP）对于 SQL 本身而言是全新的。SQL3 作为一个 ANSI/ISO 的草案标准于 1999 年发布，同年晚些时候它的状态被改变成标准级。

SQL3 扩展了传统的关系数据模型，将对象和复杂数据类型，以及所有支持的机制结合到关系表中。它为 SQL 引入了所有主要的 OOP 规则，即**继承**（inheritance）、**封装**（encapsulation）和**多态**（polymorphism）（所有这些内容都超出了本书的范围），以及在 SQL-92 中定义的“标准”SQL 功能。它提供了与数据用户应用程序的无缝集成，其中这些应用程序是以 OO 语言（如 SmallTalk、Eiffel 等）设计和实现的。

Note 注意：SQL 标准变得越来越灵活。例如，在 1992 年和 1999 年之间，即主要 SQL 标准发布时期，SQL/CLI 和 SQL/PSM 分别于 1995 年和 1996 年被添加到了 SQL 标准中。

5. SQL:2003

SQL:2003 中的主要话题是 XML。另外，还有对 SQL:1999 标准一些细微的修改和“错误修正”，还添加了几项新的功能，比如表函数、序列生成器、自动生成值、标识列，以及对 DML（MERGE）和 DLL（CREATE TABLE LIKE 和 CREATE TABLE AS）语法的一些修改。同样，删除了几种过时的数据类型（BIT 和 BIT VARYING），添加了几种新的数据类型（BIGINT、MULTISET 和 XML）。

SQL:2003 由 9 部分组成：SQL/框架（该标准所有部分的通用信息）、SQL/基础（数据定义和数据操作的语法和语义）、SQL/CLI（调用级接口/ODBC）、SQL/PSM（永久性存储模块——过程语言扩展）、SQL/MED（外部数据管理——外部 OS 文件访问）、SQL/OLB（对象语言绑定——在 Java 中嵌入 SQL 的语法）、SQL/Schemata（元数据）、SQL/JRT（Java 例程和类型）、SQL/XML（SQL 中关于 XML 的规范）。这个标准的文本共 3606 页，几乎是上一个标准的两倍长。

6. SQL:2008

预计 SQL 标准的另一个修订版将在 2008 年发布。已在 2006 年做过重大修改和扩展的 XML 部分，将被再次修订。同样，对不同部分的其他小的更改也正在进行之中。新的 SQL:2008 特性很可能包括一种新的 BINARY 数据类型、正则表达式支持，可能还包括具体化的视图和 FIRST n/TOP n 查询。

Note 注意：编写这本书时，所有的数据库供应商（Oracle、DB2 和 Microsoft SQL Server）都至少部分地遵从 SQL:2003 标准，但市场上没有任何一个供应商能够完全地符合这个新的标准。每个供应商都有自己专有的特性，以便遵从更高级别的一致性。

Note 注意：要了解 SQL ANSI/ISO 标准很容易，它们可以帮助人们学习实际的 SQL 实现，但是，要编写一个真正的百分之百遵从 ANSI SQL 的产品脚本是不可能的。这就好比了解拉丁语可以帮助人们学习西班牙语、意大利语或葡萄牙语，但是如果在马德里、罗马或里斯本的大街上讲拉丁语，那里的人们几乎不会知道你在说什么。两者主要的区别是其大体的方向——拉丁语是所有以上提到的（而且还有更多）语言的祖先，而 ANSI/ISO SQL 标准是所有专有 SQL 产品的目标。

www.TopSage.com

大家网 TopSage.com

1.9 小结

数据库渗透到了人类活动的方方面面，而关系数据库已经成为了实际的标准。一些早期数据库模型（层次和网状数据库）依然在使用，但关系数据库模型（RDBMS）占据了这块市场的最大份额。

RDBMS 解决了早期数据库中一些固有的问题，并且随着速度更快的硬件的出现以及行业中重量级人物的大力支持，RDBMS 已成为每一个企业的首选。新的面向对象的数据库系统（OODMS）正在发展之中，虽然没有任何一种 OODMS 能达到与 RDBMS 同样的接收级别。面向对象的特性已成为 SQL 标准的一部分，所有主要的供应商在某种程度上都在实现着这些特性，它们逐渐演变成了面向对象的关系数据库系统（OORDBMS）。

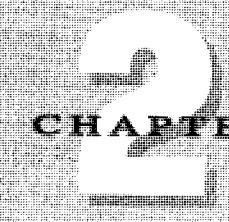
XML 在 2003 年成为了 SQL 标准的一部分，并且越来越流行。Oracle、Microsoft 和 IBM，以及许多其他较小的供应商都将 XML 特性添加到了自己的 RDBMS 产品中。

数据库市场上的大多数现有应用程序都使用 SQL 作为标准的数据库语言。到目前为止，SQL 标准经历了四个阶段：SQL-86/87 和 SQL-89、SQL-92、SQL:1999 以及 SQL:2003（SQL:2008 即将问世），其中每种 RDBMS 产品都至少部分地遵从 SQL:2003 标准。

SQL 作为 RDBMS 的语言已经在市场上存在了将近 30 年，而且每件事都表明在不久的将来它将继续存在。事实证明，SQL 有的用户友好性和灵活性足以支持新的特性，同 SQL 标准一样，它还在不断向前发展。

www.TopSage.com

第2章
SQL基本概念和原理



本章提要：

- SQL 标准回顾
- SQL 初探
- 四个基本的 SQL 语句：入门
- 高级 SQL 功能
- 平台独立性

自从 SQL-98 作为第一个 SQL 标准被采用以来，SQL 就一直致力于成为标准化的、通用的、非过程的以及不依赖于供应商的关系数据库语言。从某种程度上来说，它成功了，最新版本的 SQL-2003 标准使得现实离这一目标更近了一步。

2.1 约定和要求

和许多流行的编程语言（例如，C++、Java、Visual Basic 和 C#）不同，SQL 用于处理数据集，而且是非过程语言，也就是说，在其他编程语言中被认为理所当然的特性，如流控制语句（IF……THEN）、循环结构（FOR……NEXT）等，在 SQL 中完全没有。

SQL 实现了数据存储、数据检索及数据操作功能，它与数据库管理系统(DBMS)紧紧地联系在一起，既不能存在于 DBMS 之外，也不能在 DBMS 外执行。我们要做的就是将一个查询提交到 DBMS，然后在某些客户端程序中得到结果，这些结果可以是从数据库得到的真实数据，也可以是一项任务的状态结果（例如，插入或删除记录）。

和编程风格的**变量操作**不同，插入数据、更新数据和检索数据都是**基于集的过程**。SQL 语句操作于数据集，虽然在 SQL 程序员看来，完成一项操作可能要花费很长的时间，但这些操作确实不包含任何流控制。当然，在某些过程语言（通常是 C 语言）中会实现执行 SQL 查询的引擎。大多数情况下，不管它多短多简单或者多长多复杂，一个 SQL 程序就是作为一个整体来执行的一条或多条语句。

注意：为了解决 SQL 程序不足所引发的问题，数据库供应商提供了许多自己的 SQL 函数和程序扩展插件，例如，用于 Oracle 的 PL/SQL、用于 Microsoft SQL Server 的 Transact-SQL，以及用于 IBM DB2 的 SQL PL。而现在，我们可以在 RDBMS 中使用高级语言了，如 Java 或 Visual Basic。这些都逐渐成为了 SQL 标准（SQL/JRT 和 ISO/IEC 9075-13:2003）。

SQL 的使用经常与客户端/服务器计算模型（很久以前，这是个很时髦的说法）结合在一起。**客户端**只需要知道如何连接

www.TopSage.com

到一个服务器、如何请求数据并为用户以某种格式（通常是可读的）表示它。服务器用来理解客户端的请求并返回数据，以及在内部管理此数据以达到最佳性能，并提供可靠存储和安全服务。

SQL 语句是如何被翻译为机器语言并执行的呢？这种低级别实现的复杂性隐藏在简明 SQL 语句（如 SELECT、INSERT 或 UPDATA）背后，而将它们转换为实际机器命令的任务是由 RDBMS 完成的。

当 SQL 标准指定 SQL 构件块时，它们并不会指定将如何实现这些构件块。这就会引起很多麻烦：不仅供应商要选择实现 SQL 标准的不同部分，而且，大多数情况下，这些部分的实现方法也是不同的。供应商所指定的为 SQL 标准的下次迭代提供输入的扩展使得情形更为糟糕。

正如第一章中所提到的，ANSI SQL 标准（SQL-89 和 SQL-92）创建了三种一致性级别：最低、核心和扩展，以使每种数据库产品都遵从于 ANSI 标准。而 SQL3（SQL:1999）只有两种级别的一致性：核心 SQL 和扩展 SQL。除新的 SQL:2003 标准自身提出的标准之外，其定义的核心一致性级别还包括许多以前标准的扩展特性。每种主要的 RDBMS 都遵从第一种（核心）一致性级别，由于现在还没有 SQL 标准的独立的标准化一致性，我们要听取供应商的建议。

ANSI/ISO 标准文档

下列文档构成了 ANSI/ISO 标准的主体。

- **ISO/IEC 9075-1:2003/Cor 2:2007**——信息技术、数据库语言、SQL、第一部分：框架（SQL/框架）
- **ISO/IEC 13249-1:2007**——信息技术、数据库语言、SQL 多媒体和应用程序包、第一部分：框架
- **ISO/IEC 9075-2:2003/Cor 2:2007**——信息技术、数据库语言、SQL、第二部分：基础（SQL/基础）
- **ISO/IEC 13249-2:2003**——信息技术、数据库语言、SQL 多媒体和应用程序包、第二部分：全文
- **ISO/IEC 13249-3:2006**——信息技术、数据库语言、SQL 多媒体和应用程序包、第三部分：空间
- **ISO/IEC 9075-3:2003/Cor 1:2005**——信息技术、数据库语言、SQL、第三部分：呼叫级接口（SQL/CLI）
- **ISO/IEC 9075-4:2003/Cor 2:2007**——信息技术、数据库语言、SQL、第四部分：永久性存储模块（SQL/PSM）
- **ISO/IEC 13249-5:2003**——信息技术、数据库语言、SQL 多媒体和应用程序包、第五部分：静态图像
- **ISO/IEC 13249-6:2006**——信息技术、数据库语言、SQL 多媒体和应用程序包、第六部分：数据挖掘
- **ISO/IEC CD 13249-7**——信息技术、数据库语言、SQL 多媒体和应用程序包、第七部分：历史
- **ISO/IEC 9075-9:2003/Cor 1:2005**——信息技术、数据库语言、SQL、第九部分：外部数据管理（SQL/MED）
- **ISO/IEC 9075-10:2003/Cor 2:2007**——信息技术、数据库语言、SQL、第十部分：对象语言绑定（SQL/OLB）
- **ISO/IEC 9075-11:2003/Cor 2:2007**——信息技术、数据库语言、SQL、第十一部分：信息和定义模式（SQL/大纲）
- **ISO/IEC 9075-13:2003/Cor 1:2005**——信息技术、数据库语言、SQL、第十三部分：使用 Java TM 编程语言的 SQL 例程和类型
- **ISO/IEC 9075-14:2006**——信息技术、数据库语言、SQL、第十四部分：XML 相关规范（SQL/XML）

www.TopSage.com

TopSage.com

✦ **ISO/IEC 9579:2000**——信息技术、带有安全增强的 SQL 远程数据库访问

这些文档在网站 www.iso.org 和 www.ansi.org 上可付费使用。

SQL 适应性很强，以下内容是留给供应商实现的：

- ✦ **语义和语法差异**
- ✦ **为处理技术打开一个数据库**：ODBC、OLEDB、JDBC 和其他接口并不是 SQL 标准的一部分，尽管 SQL 标准已将 CLI（呼叫级接口）定义为 ISO/IEC 9075-3：2003/Cor 1：2005 的一部分。
- ✦ **动态和嵌入的 SQL 实现会因供应商的不同而有所差异**
- ✦ **排序顺序**：已排序的查询结果是如何呈现的呢？这依赖于是否使用了 ASCII 或 EBCDIC 字符。（虽然 UNICODE 标准解决了这一问题，仍然存在着大量的旧有数据和应用程序。）
- ✦ **不同的数据类型扩展**：自定义结构、混合数据类型、对象，等等。
- ✦ **数据库目录表中的差异**：由于这仅在完全一致性级别标准下会提到，在核心一致性下工作的供应商没有理由放弃自己的专有结构。

表 2-1 介绍了从上一个 SQL 标准中添加或修改的主要的 SQL:2003 功能区。

表 2-1　　主要的 SQL:2003 区

功能	描述
调用级接口（CLI）	这个规范定义了通过一套例程对数据库进行访问，这套例程可以由被客户端应用（通过 ODBC，JDBC 等）调用
信息模式	一套数据库视图，用于访问特定数据库的元数据
角色安全增强	一种安全范例，定义了在把安全特权分成逻辑上相关的组时调整它们的能力
递归	引用需要的嵌套关系，以建立层级结构的模型
事务性支持和保存点	在“全部更改，否则不改”的基础上进行数据更改的能力。保存点功能是指把粒度添加到事务操作的能力，其中一个事务可以回滚到某个命名的步骤，而不是回滚到开始
SQL 数据类型：BLOB，CLOB，BOOLEAN，REF，ARRAY，ROW，用户定义类型	能够适应复杂的现代计算的新的数据类型（参阅第 3 章获得详细信息）
SQL 多媒体数据类型：全文（Full Text）、静态图像（Still Image）、空间（Spatial）	为多媒体开发的新的数据格式
SQL/XML	当前数据交换的实际标准。这成为了 OpenOffice、Microsoft Office 和许多其他字处理应用程序的一个标准。
SQL/MED	为数据库语言 SQL 定义了扩展，通过使用外表和数据链数据类型为外部数据管理提供支持
SQL/PSM（永久性存储模块）	由像 Oracle 的 PL/SQL 或 Microsoft Transact-SQL 这样的专有过程化扩展统治的领域；定义了标准的编程流控制结构（IF...THEN...ELSE）、循环，等等
触发器	定义了当某些预定义的事件发生时自动执行的动作；新的标准调整了基本的触发器功能
用于连接、会话、事务以及诊断的管理工具	用于集中或分布式计算的基础框架支持

SQL 是一种充满生机的语言——它一直在发展，能适应不断变化的需求。尽管市场迫切需要标准化数据库之间的功能和数据交换，但供应商们还是更愿意通过让客户迷上某些便利的非标准功能来把他们锁定在某种特定 RDBMS 包中，虽然这些非标准功能有时能显著地改善性能，但也使把 SQL 例程迁移到一个不同的 RDBMS 上变得很难、很昂贵或者根本不可能。

每一个供应商都被鼓励提交具有新意的文件，而 ANSI/ISO 委员会总是在不断对这些新概念进行评估，这最终将导致另一个 SQL 标准的出现。SQL 的明天可能超出我们现在的想象。某些正在涌现的标准包括 XML（eXtensible Markup Language，扩展标记语言）和 OLAP（Online Analytical Processing，联机分析处理）。

交叉参考：XML 将在第 15 章中讨论。

注意：每一个专有 RDBMS 的内置专有功能最终都会提交给标准委员会，标准委员会决定是否将其加入 SQL 标准中。

www.TopSage.com

2.2 SQL初探

在本书中，我们将使用一个虚构硬件商店的ACME订购管理数据库。关于这个数据库的完整描述，以及如何在选择的RDBMS（只要选择Oracle 11g、SQL Server 2008或IBM DB2 9.5）上安装，请分别参阅附录B和附录F。本章在介绍SQL的过程中，只用到ACME表。我们尝试创建了一种适用于本书中讨论的每个RDBMS的通用语法；因此，对特定RDBMS而言，该语法可能并不是最简明和最有效的语法。

2.2.1 数据库示例

就SQL而言，数据库中的每个对象都会以CREATE语句开始。这个语句用于创建组成数据库的所有对象：表、索引、约束等。我们将在第4章和第5章探讨创建、改变和删除数据库对象。

假设已经创建了数据库和架构，从一个CREATE TABLE语句开始。这个语法实际上在所有三个数据库中是完全一样的，即指定列的名称和它们的数据类型，数据类型定义表将要存储的信息类型：

```
CREATE TABLE status
(
    status_id_n INT,
    status_code_s CHAR(2),
    status_desc_s VARCHAR(30)
)
```

在RDBMS上执行这条语句会创建一个由空表组成的结构，其中为列指定了数据类型：status_id_n、status_code_s和status_desc_s。

交叉参考：数据类型在第3章讨论。

上面的语句可能需要重复多次（同一架构内的表名各不相同），以创建多个需要添加到数据库中的表。表可以作为单独的对象存在，与数据库中的其他对象没有任何关系，不过请充分利用RDBMS的**关系**本性，可能需要为此RDBMS定义父子关系。表之间的关系通过外键**约束**定义。为简单起见，这里没有指定任何约束（或默认值）。

交叉参考：约束在第4章讨论。

要删除一个表（或RDBMS中的许多其他对象），需要发出一个DROP语句：

```
DROP  TABLE status
```

注意：在现实生活中，引用完整性约束可以避免彻底删除表。第1章讨论了在数据库完整性环境中，使一个数据库成为关系型的要素。可能需要禁用约束或层叠它们。第4章和第5章会更详细地讨论引用完整性和约束。

数据库不仅仅是一个表的集合，它还拥有多个对象以及关联的进程和结构，所有这些对象协同工作来保存和处理数据，但现在我们只从简单的角度进行讨论。

2.2.2 获取和导出数据

表被创建后，就需要用某些数据填充它们，数据库的用途就是如此。SQL定义了4条基本语句，它们的含义均不言而喻，用于查询和操作数据库表中的数据（参见表2-2）。第6章到第9章将深入讨论这些语句的详细使用方法。

表2-2 4条基本的SQL语句

SQL语句	作用
INSERT	把新的行添加到表中
UPDATE	更新数据/更改数据库表中的现有值
SELECT	在数据库表中检索数据
DELETE	从表中删除数据

www.TopSage.com

交叉参考：在本书后面，将学习数据定义语言（DDL）（第 4 章和第 5 章）、数据操作语言（DML）（第 6 章）、数据查询语言（DQL）（第 8 章和第 9 章）和数据控制语言（DCL）（第 14 章），它们都是 SQL 的一部分。

例如，要为前面示例中创建的表添加一条新的状态记录，可以使用下面的语句：

```
INSERT  INTO  status
        (STATUS_ID_N,  STATUS_CODE_S,  STATUS_DESC_S)
VALUES  (8, '70', 'INVOICED')
```

这条语句包括 SQL 关键字 INSERT，后面是表名，然后是表的列组成的列表和一个与这些列匹配的数据列表。

注意：可以列出所有列或列的子集。在后面这种情况中，忽略的列必须分配默认值（RDBMS 自动用一些预定义值填充丢失的数据）。如果匹配数据列表每个都有值，而且每列的顺序正确，则可以忽略列列表本身。

这条语句可以通过 RDBMS 访问实用工具（在附录 E 中讨论）直接输入。数据库通常用一条消息来确认这个插入，否则生成一条错误消息（如果由于某种原因插入失败）。

注意：STATUS_CODE_S 和 STATUS_DESC_S 的值包含在单引号中，因为这些列是字符数据类型。STATUS_ID_N 是 numeric 数据类型，所以不需要引号。

如果需要更改某些现有数据（例如，当位于别处的其他相关数据保持不变时，某个获取状态代码可能会发生变化），则使用 UPDATE 语句。同样，下面的语法在本书中使用的所有三大 RDBMS 产品（Oracle 11g Database Server、IBM DB2 9.5 和 Microsoft SQL Server 2008）中都是通用的。

```
UPDATE  status
SET     status_desc_s = 'APPROVED'
WHERE   status_id_n = 8
```

基于列进行更新，列出所有要更新的列，用相应代替值进行更新。如果记录中的每一列都需要更新，则它们都必须用相应的值列出。

```
UPDATE status
SET     status_desc_s = 'APPROVED',
        status_code_s = '90'
WHERE  status_id_n = 8
```

上面例子引入了一个很重要的 WHERE 子句，将更新行的编号精确限制为 ID 为 8 的客户。如果查询中省略了此 WHERE 子句，结果将是表中每一个客户的现有值都将被一个新值替换。

上面的方法同样适用于删除数据。如果需要从数据库中完全删除某一特定客户，可以使用下面的命令：

```
DELETE  status
WHERE   status_id_n = 8
```

同样，在上面查询中忽略 WHERE 子句的后果可能是灾难性的，因为表中的所有记录都将消失。通常，数据库有一些内置机制可以恢复被删除的数据，但这并不意味着可以随意操作数据库。

基本 SELECT 语句从表或视图中检索数据。需要指定要把包含在结果集中的列，或者使用星号（*）列出表中的所有列。

注意：视图是一个虚拟表，它是被查询时进行填充。第 4 章会详细讨论视图。

下面的脚本从 V_CUSTOMER_TOTALS 视图中选择 CUSTOMER_NAME、ORDER_NUMBER 和 TOTAL_PRICE 列：

```
SELECT    customer_name,
          order_number,
          total price
FROM      v_customer_totals
customer_name            order_number       total_price
-----------------------  -----------------  ------------
WILE BESS COMPANY            523720              7511.00
WILE BESS COMPANY            523721              8390.00
WILE BESS COMPANY            523722              6608.00
WILE BESS COMPANY            523723             11144.00
WILE ELECTROMUSICAL INC.     523726              6608.00
WILE ELECTROMUSICAL INC.     523727              6608.00
WILE ELECTROMUSICAL INC.     523728              6608.00
```

SELECT 语句可以非常复杂，而且会涉及多个表；此外，可以使用许多修饰符优化该语句或更改

www.TopSage.com

大家网 TopSage.com

数据返回的方式。有关 SELECT 语句的详细讨论，请参考第 8 章和第 9 章。

2.2.3 从不同角度看数据

查看数据总是有很多种方法。SQL 提供了在将结果集返回给客户端应用程序之前操作数据的方法。实际上可以根据表中的任何列按升序或降序排列数据；可以在检索数据时执行计算；限制应该显示哪些数据。下面给出体现这些功能的例子。

基本的 SELECT 返回针对选择条件的结果集，选择条件完全是基于现有数据指定的。例如，如何查看减去了州和联邦税后的净销售利润呢？可以在查询本身中执行相当复杂的计算。本例是基于视图 V_CUSTOMER_TOTALS 的查询，该视图包含列 CUSTOMER_NAME、ORDER_NUMBER 和 TOTAL_PRICE。

假定税率是 8.5%，则这个查询如下：

```
SELECT    customer_name,
          order_number,
          (total_price-(total_price * 0.085)) net_sale
FROM      v_customer_totals
customer_name              order_number       net_sale
-------------------------- ------------------ --------
WILE BESS COMPANY          523720              6872.56
WILE BESS COMPANY          523721              7676.85
WILE BESS COMPANY          523722              6046.32
WILE BESS COMPANY          523723              10196.76
WILE ELECTROMUSICAL INC.   523726              6046.32
WILE ELECTROMUSICAL INC.   523727              6046.32
WILE ELECTROMUSICAL INC.   523728              6046.32
```

注意：默认情况下，结果中的每一列都有它自己的名称，对于类似上面例子中计算列的情况，RDBMS 将使用整个字符串（total_prise-(total_price *0.085)）作为名称。为了更加易读，可以使用别名替换这个繁琐的字符串，使其更具描述性，本例中使用 NET_SALE 作为别名。

查询返回的结果包含计算后的值，换言之，来自表的原始数据已经被转换了。SQL 也提供了几个函数，可以用在查询中，在数据被提取时对它进行操作，大多数供应商（包括本书中提到的那些）都允许创建自定义函数。

交叉参考：第 10 章深入介绍了三大 RDBMS 供应商（IBM、Microsoft 和 Oracle）之间所有流行的函数及其用法。附录 G 几乎列出了至今为止，RDBMS 供应商实现的所有 SQL 函数。

使用 SQL 可以完全控制数据的显示方式。可以根据任意列对数据进行排序（按字母顺序或数字顺序）。假设需要公司和销售列表根据每个客户的订单销售量来排序：

```
SELECT    customer_name,
          order_number,
          (total_price-(total_price * 0.085)) net_sale
FROM      v_customer_totals
ORDER BY net_sale
customer_name              order_number       net_sale
-------------------------- ------------------ -----------
WILE ELECTROMUSICAL INC.   523726              6046.32
WILE ELECTROMUSICAL INC.   523727              6046.32
WILE ELECTROMUSICAL INC.   523728              6046.32
WILE BESS COMPANY          523722              6046.32
WILE BESS COMPANY          523720              6872.56
WILE BESS COMPANY          523721              7676.85
WILE BESS COMPANY          523723             10196.76
```

现在，希望销售份额最多的客户排在列表顶端（与默认顺序**升序**相反）。使用 DESC（降序）修饰符：

```
SELECT    customer_name,
          order_number,
          (total_price-(total_price * 0.085)) net_sale
FROM      v_customer_totals
ORDER BY net_sale DESC
customer_name              order_number       net_sale
-------------------------- ------------------ --------------
WILE BESS COMPANY          523723             10196.76
WILE BESS COMPANY          523721              7676.85
WILE BESS COMPANY          523720              6872.56
WILE BESS COMPANY          523722              6046.32
```


www.TopSage.com

```
WILE ELECTROMUSICAL INC.    523726          6046.32
WILE ELECTROMUSICAL INC.    523727          6046.32
WILE ELECTROMUSICAL INC.    523728          6046.32
```

2.2.4 聚合

使用 SQL，在检索数据时，不仅可以对数据进行转换，还可以聚合数据，即由多个字段中的数据得到一个值，可以根据某一条件基于查询中使用的表列对这个值进行分组。例如，需要知道销售总额。假定数据库包含 V_CUSTOMER_TOTALS 视图，该视图拥有到目前为止的所有销售信息，现在需要求和。

要得出所有产品的所有订购总额，使用 SQL 内置函数 SUM，它将简单地累计在视图的 TOTAL_PRICE 列中找到的所有数额。

```
SELECT    SUM(total_price) net_sale_total
FROM      v_customer_totals
net_sale_total
--------------
     457000.40
```

要得出每次订购的平均数额，则运行以下查询，在 TOTAL_PRICE 列上使用 AVG 聚合函数：

```
SELECT    AVG(total_price) net_sale_average
FROM      v_customer_totals
net_sale_average
----------------
     8960.792156
```

在现实生活中，甚至可能需要进一步限制查询，即查询某一特定客户或日期范围内的平均销售额。

使用其他如 GROUP_BY 和 HAVING 这样的谓词，可以按照客户、日期、产品等对 NET_SALE 求和。分组可以使聚合在一个组内进行。第 8 章将详细讨论这些谓词（及示例）。

前面的例子只是让读者初步了解使用 SQL 和 RDBMS 可以完成哪些事情。

交叉参考：有关 SQL 函数的更多信息（包括聚合函数），请参阅第 10 章。

2.2.5 数据安全

虽然数据库基础结构中内置有大量安全功能，但 SQL 还是提供了大量内置机制以保证数据安全。这些机制是细粒度的，不过粒度级别在很大程序上依赖于特定的实现。从本质上讲，这些机制相当于访问各种数据库对象或执行某些操作的特权。例如，用户可以被授予访问某一特定表或执行某一特定语句（如 SELECT）的权限，但可能未授予向表添加数据（使用 INSERT 命令）的权限。

1. 特权

假定数据库中定义了一个用户 JOHN_DOE，可以使用下面的 SQL 语句，将 SELECT 权限授予此用户：

```
GRANT     SELECT
ON        v_customer_totals
TO        john_doe
```

要同时授予 SELECT 和 UPDATE，可以使用下面的语法：

```
GRANT     SELECT, UPDATE
ON        v_customer_totals
TO        john_doe
```

要撤消 SELECT 特权，使用下面的语法：

```
REVOKE    SELECT
ON        v_customer_totals
FROM      john_doe
```

下面的语法撤消 JOHN_DON 的所有特权：

```
REVOKE    ALL
ON        v_customer_totals
FROM      john_doe
```

注意：供应商特定实现有其他访问控制语句。例如，Microsoft 引入了 DENY 语句，它允许自定义 GRANT 语句。第 12 章提供了更多有关安全的信息。

2. 视图

实现安全的一个常用机制是使用视图。视图是一种限制用户访问数据的方式。可以把视图想象成一个虚拟表：它可以联接来自数据库中几个表的列，限制可以查看的列数等。视图不包含任何数据，而是在 SELECT 语句根据它执行时按需获取数据。对大多数实用目的而言，从视图中选择与等同于从

www.TopSage.com

表中选择。

例如，按照这种方式合计和分组某些数据时，视图 V_CUSTOMER_TOTALS 从 CUSTOMER、ORDER_HEADER、ORDER_LINE 和 PRODUCT 表中收集信息。

交叉参考：要查看创建视图 V_CUSTOMER_TOTALS 的完整 SQL 语法，请参考附录 B。

交叉参考：一些视图会限制对基础数据的访问（例如，不允许执行任何 UPDATE 或 INSERT 语句）。视图在第 4 章讨论。

有关安全的内容还有很多；例如，本书中讨论的所有三大 RDBMS 都实现了基于角色的安全机制，其中，单个用户被分配某一特定角色（如会计师），而所有的特权都授予这个角色。

交叉参考：有关 SQL 安全功能的全面讨论，请参阅第 12 章。

2.2.6　从客户端应用程序访问数据

许多客户端应用程序都可用于访问 RDBMS 数据。它们都以两种完全不同的方式来使用 SQL 完成这件事。

嵌入式 SQL 允许用户创建可以通过**嵌入**在 ANSI/ISO 标准中的 SQL 语句访问 RDBMS 的程序，ANSI/ISO 标准宿主 C 或 COBOL 之类的编程语言。这意味着用标准的编程语言来编写一个应用程序，然后只在需要使用数据库时切换到 SQL。通常供应商们会提供专门的开发工具来创建这类应用程序。

动态 SQL 除了具有嵌入式 SQL 的所有功能外，还有一些其他功能。主要区别在于动态 SQL 不是嵌入在某些编程语言中，而是**动态**生成的，作为一个简单文本传递给 RDBMS。这样就赋予了嵌入式 SQL 所没有的灵活性：不需要硬编码表名、列名甚至数据库名——因为它们都是可变的。

交叉参考：嵌入式 SQL 和动态 SQL 在第 16 章讨论。

2.2.7　新的发展

SQL:2003 标准也反映了一个已经发展了很长时间的概念——**联机分析处理**（OLAP）。它既不是一个新的也不是一个模糊的概念。OLAP 理事会成立于 1995 年，之前，大多数分析都是手动完成的。随着数据仓库（另一个与数据库有关的概念）的出现，OLAP 迅速得到普及。

OLAP 是关于数据的操作：分析复杂的趋势，从各种不同的角度查看数据；从用户角度出发，把原始数据转换为多维对象。OLAP 查询能够回答比普通 SQL 更智能的问题，例如，“如果牛肉的价格每磅涨了 10 美分，会对汉堡包的销售有怎样的影响？”这类问题通常由定义好的专用应用程序来解决，直到 SQL:1999 引入对 OLAP 的内置支持。OLAP 产品补充了 RDBMS，所有三大供应商（IBM、Oracle 和 Microsoft）都使用一些专用扩展插件来支持 OLAP。市场上 30 多家供应商支持基于 RDBMS 的 OLAP 解决方案。

另一个受 SQL 支持的较新的功能是可扩展标记语言（eXtensible Markup Language，XML）。XML 涉及的都是数据交换。一般称 XML 为**自描述格式**，因为它以层次结构来表示数据，XML 与可扩展样式表语言（eXtensible Stylesheet Language，XSL）配合使用，提供通过浏览器的可视化表示，或者作为一种在多方之间使用的数据交换格式。

XML 是开放的标准，没有被某一特定开发商锁定，因此它现在被所有供应商使用，实现了真正的全球数据互操作。XML 的一个主要优势是它可以使用 HTTP 协议（非常重要的 Internet 协议）进行传输，这使得任何专用网络都过时了；XML 可以被加密来获得更好的安全性，或者在安全套接层（Secure Socket Layer，SSL）上进行发送，XML 是 Web 服务的核心，新的编程范例 SOA（面向服务的体系结构）就是基于 Web 服务的。这种多功能性是有代价的，例如，详细的 XML 本身就比编译过的代码更慢，因为它是一个每次都需要被解析和翻译的文本。尽管如此，XPath 和 XQuery 等优势能克服许多缺点，因为本书中讨论的所有供应商都在其 RDBMS 产品中支持 XML，所以 XML 迅速成为存储半结构化信息的首选方式。

www.TopSage.com

TopSage.com

2.3 平台和时间无关性

SQL 与 C/C#、Pascal、VB.NET 和 Java 之类的标准编程语言不同，因为使用 SQL 不能创建独立的应用程序。SQL 不能在数据库引擎之外存在，数据库引擎用于把语句转换为机器语言来执行。SQL 不具备属于其他语言主要部分的所有编程结构，如条件逻辑、循环和变量的使用等。

尽管这些不足正在弥补，首先是引入 SQL 函数，然后是过程化扩展插件，最后还为 SQL 添加了面向对象的功能，但 SQL 永远不会是另一种多用途编程语言。这个“缺点”正是 SQL 无处不在的关键，因为它是依靠 RDBMS 来执行的（非常类似于 Java 依赖特定于平台的 Java 虚拟机，即 JVM），它是第一个真正与平台无关的语言。

每一个程序都是通过输入一些纯 ASCII 形式的命令创建的。对于大多数编程语言而言，这些命令接着被编译成二进制机器码和可执行文件。这种方法带来的一个问题是，对于每一个平台，必须使用特定于平台的编译器重新编译程序——为 Microsoft Windows 编译的程序不会在 UNIX 上工作，反之亦然。Java 提出的解决方案仍然要求它的与平台无关的字节代码在特定于平台的 JVM 上执行。

SQL 查询的创建方法与程序的其他部分非常类似——通过输入 SQL 关键字，但相似性也仅限于此。SQL 程序可以作为一个简单的 ASCII 文件存储，该文件可以复制到 UNIX、Windows、MacOS、Linux 等，无需做任何改变。此外，它可以在任何这些具有标准编辑工具的操作系统中打开、修改和保存，无需考虑任何特定于平台的功能。只要 SQL 仍然是一个脚本，就可以在不同平台（不是在不同 RDBMS 之间）之间轻松转移。在某种意义上，RDBMS 的作用就像一个 JVM。即使 SQL 成为某种宿主语言的一部分（例如，嵌入到 C 程序中），它仍然只是文本。它的行为与 HTML 完全一样，在 HTML 被送入 Web 浏览器进行执行前，它只是一个 ASCII 字符集合。对于 SQL，它的语句需要被发送给一个 RDBMS，以便转换为可执行的机器码，它所需要的只是一个特定于平台的数据库引擎实现。

SQL 真正与平台无关的功能之一是基本数据类型：无论是在 UNIX 还是在 Windows（32 位或 64 位操作系统）上执行 SQL，保留存储块的尺寸和结构都是完全一样的。无论怎样，`INTEGER` 数据类型总是 4 个字节，`DOUBLE` 数据类型总是占用 8 个字节（根据实现，专用数据类型可能会有所不同）。似乎这点是显而易见的，但所有其他编程语言并不都是如此。例如，C 编程语言 `INTEGER` 在 32 位操作系统上是 4 个字节，而在 64 位操作系统上则是 8 个字节（当然，这也取决于特定的 C 编译器）。有关数据类型的更多内容将在第 3 章讨论。

在某种意义上，SQL 使用 Java 编程语言共享与平台无关的思想：就像 Java 虚拟机把 Java 字节代码转换为特定于平台的机器码一样，RDBMS 执行与平台无关的 SQL 代码，把它转换为特定于操作系统的机器码。因此，一个在 Windows 上用记事本编写的 SQL 程序可以由安装在 UNIX 或 Linux 上的 Oracle RDBMS 来运行。

Note

注意：Oracle 11g 数据库（不同的版本）对以下平台是可用的：

IBM AIX 5L 5.2 版和 5.3 版（64 位）；

HP OpenVMS Alpha 8.2 和 Itanium 8.2-1；

HP True64 UNIX V5.1B；

HP-UX Itanium 11i（11.11）和 11iV2（11.23），以及 PA RISC（64 位）；

IBM z/OS（OS/390）V1.4 和 z/OS V1.4 或更高版本；

基于 IBM z 系列的 Linux：Red Hat Enterprise Linux（RHEL）4.0（Update 2 或更高版本）和 SUSE Linux Enterprise Server（SLES）9（Service Pack 2 或更高版本）；

Linux（Itanium、POWER、x86 和 x86-64）：Red Hat Enterprise Linux AS/ES 3.0（Update 4 或更高版本）；Red Hat Linux 4.0（Update 1 或更高版本）；带有 SP4 或更高版本的 SUSE Linux Enterprise Server 8.0；带有 SP2 或更高版本的 SUSE Linux Enterprise Server 9.0；以及 Asianux 1.0/2.0；

带有 Service Pack 1 或更高版本的 Microsoft Windows 2000，所有版本，包括 Terminal

www.TopSage.com

Services 和 Microsoft Windows 2000 多语言版（MLE）；Windows Server 2003，所有版本；Windows XP Professional；以及 Windows Vista——Business 版、企业版和最终版；

Sun Solaris 操作系统：SPARC 64 位（Solaris 8 Update 7 或更高版本；Solaris 9 Update 6 或更高版本；以及 Solaris 10），以及 x86 和 x86-64（Solaris 10）。

IBM DB2 9.5 软件（不同的版本）可以安装在以下平台上：

Microsoft Windows：Windows XP Professional 和 Windows XP Professional x64；Windows 2003 标准版、企业版和数据中心版（32 位和 64 位）；Windows Vista；VMWare 和 Microsoft Virtual Server 2005；

Linux：Novell Open Enterprise Server 9、Red Hat Enterprise Linux（RHEL）4 以及 SUSE Linux Enterprise Server（SLES）9 和 10；

IBM AIX：5.2 版和 5.3 版；

HP-UX：11iv2（11.23.0505）操作系统，带有 2005 年 5 月基本品质（QPKBASE）随附软体、2005 年 5 月应用程序品质（QPAPPS）随附软体和 PHNE_32606 修补包（需要一个 64 位内核；只限于服务器）；

Sun Solaris：Solaris 10（UltraSPARC 和 x86-64 EM64 或 AMD64）或 Solaris 9（只限于 UltraSPARC，需要 64 位内核及修补包 111711-12 和 111712-12）。

Microsoft SQL Server 2008（不同的版本）适用于 Microsoft Windows。它可以在以下任意版本上安装：

Windows Server 2003 64 位 x64 SP1；

Windows Server 2003 64 位 x64 企业版 SP1；

Windows Server 2003 64 位 Itanium SP1；

Windows Server 2003 64 位 Itanium 企业版 SP1；

Windows XP Professional 64 位 x64 SP2；

Windows Vista 64 位 x64 Business 版；

Windows Vista 64 位 x64 企业版。

如前面所述，特定 SQL 语言之间存在许多差别，例如，为 SQL Server 2008 编写的查询可能无法在 IBM DB2 UDB 上执行，反之亦然。但一个在 Windows 上安装的 Oracle 中运行的查询无需更改即可安装在 Linux 或 UNIX 上的 Oracle 中执行。这意味着只要使用所有三个数据库通用的 SQL 功能和关键字，即可在所有这些 RDBMS 上运行完全相同的查询，无需任何更改。

实际上许多 RDBMS 产品都可以在我们能想到的每一种平台（操作系统）上运行，如 Teradata、Sybase、Ingres、Informix、Empress、MySQL、mSQL、PostgreSQL、LEAP RDBMS、FirstBase、Ocelot、Progress、Typhoon、SQL/DS、Daffodil DB、Cloudscape、Compaq Non-Stop SQL/MX&SQL/MP、Linter RDBMS SQL、Interbase、UniVerse 和 GNU SQL Server，这里只列举了一小部分，不断会有新的产品推出。

注意：尽管绝大多数 RDBMS 供应商都选择让他们的产品符合 ANSI，但仍有一小部分市场运行的数据库使用专用的非 SQL 语言。

当然，这些 RDBMS 中的每一个都喜欢炫耀它自己的 SQL 语言，但幸运的是它们中的大部分都是符合 SQL 标准的。在每一种实现中，基本 SQL 语句不需要或只需要很少的改变即可在这些 RDBMS 中的任何一个上执行，在任意这些系统中都可以轻松使用掌握的 SQL 知识。

2.4 小结

SQL 不是一种编程语言，它是特定数据库的语言，用于检索和操作关系数据集。

有一个被所有大型 RDBMS 供应商普遍支持的 SQL 标准。尽管需要密切关注任意特定 RDBMS 所使用的具体 SQL 语言，但 SQL 仍然是当今数据库世界中的通用语言。

www.TopSage.com

使用 SQL 可以创建（和删除）许多数据库对象，还以插入、选择、更新和删除数据。

有两种不同类型的 SQL 用于客户端应用程序：嵌入式 SQL 和动态 SQL。每一种都有它自己的用途，不过语法基本上是相同的。

OLAP 相对 SQL 而言是比较新的概念。作为商业智能领域中迅速发展的一部分，OLAP 为针对商业支持信息的数据分析提供了方便。

XML 是一种标准的数据交换格式，基本上，该领域中的每一个供应商都承诺对其进行支持。一些 RDBMS 产品（包括本书中选择的三大 RDBMS 供应商）本身就支持 XML（一种用于结构化和描述数据的开放标准）。XML 正在成为事实上的数据交换行业标准，特别是在 Internet 上。

www.TopSage.com

大家网
TopSage.com

CHAPTER 3

第3章
SQL 数据类型

本章提要：

- 实现差异
- 字符串
- 二进制字符串
- 精确数字
- 近似数字
- 日期和时间
- 用户定义数据类型
- 对象数据类型
- 文字（常量）
- Null——任何数据类型的替代品

以前“数据库”被定义为一个有组织的信息集合。这不仅意味着数据必须按照公司的商业规则进行组织，而且数据库组织还应反映出信息特征。数据库可以存储美元数额、数量、名称、日期和时间值、二进制文件等许多信息。这些信息可以按类型进一步分类，类型反映数据的“特性”：数字、字符、日期等。

Note 注意：“数据类型”是一个数据库表列的特性，决定该列可以保存哪种类型的数据。SQL:2003 将“数据类型”定义为一组表示值，其中每个值至少属于一个数据类型。

但为什么需要数据类型呢？使用统一的数据类型（如字符串形式）来存储所有信息不是更简单吗？

原因有很多，其中一些是历史原因。关系数据库诞生于 20 世纪后期，那时的磁盘空间和内存存储都非常宝贵，因此，当时的想法就是尽可能有效地存储所有内容。但是，现有编程语言对如何存储不同类型的数据有一些内置规则。例如，任何英文字符（以及一些特殊字符和数字）可以用它的 ASCII 等价形式来表示，而且必须使用一个字节（有关 ASCII 的更多信息将在本章后面部分讨论）进行存储。传统上数字以二进制字符串（由于计算机体系结构的缘故）的形式存储。要表示一个–32 768 到+32 767 之间的数字，需要用两个字节（或 16 位）来存储（2^{16}）。如果用 ASCII 字符表示数字，任何大于 9 999 的整数都将需要用 6 个字节来存储（5 个字节用于数字，1 个字节用于正负号），任何大于 999 的整数需要用 5 个字节来存储（4 个字节用于数字，1 个字节用于正负号），依此类推。现在假设一列中有一百万条记录，如果不用 ASCII 表示就可以省去四百万字节。这在如今看起来似乎没什么，但在 20 世纪 70 年代，那是一个大得惊人的存储空间。当然，有效性原则仍然适用，但现在的级别已经不同了。

Note 注意：一个字节由 8 位组成。每一个位是一个二进制数字（0 或 1）。所有信息都以 1 和 0 的形式存储在内存或硬盘上，计算机只能理解 0 和 1 这两种状态：0 表示没有信号，1 表示有信号。

引入不同数据类型的另一重要原因是数据库性能。要处理的信息量几乎在呈指数增长，所以将信息划分为不同的数据类型就显得非常重要，让不同类型的信息使用对它们这一特定数据类型最有效的方法、函数和运算符。例如，您可能会将 XML

www.TopSage.com

数据（一个字符序列）存储在一个字符串列中，并使用字符串函数和运算符检索和修改它，但这样会大大降低运算速度，所以现在所有大型 RDBMS 供应商都为使用 XML 数据专门设计了一种 XML 数据类型（在本章后面讨论），由 SQL:2003 标准指定。

另一个原因是逻辑一致性。每一种数据类型都有它自己的规则、排序顺序、与其他数据类型的关系、隐式转换规则等。处理相似值（如日期）集合无疑要比处理混合有日期、数字和字符串的值容易得多。试着对比两种图书馆的书架，一种书架上的所有资料都经过了排序和分类（小说在一个位置、儿童读物在另一个位置、音频书在它们的专有区域、录像带在其他地方）；而另一种书架则是一个大拼盘，各种图书、磁带、白皮书和 CD 等杂乱放在一起。您更愿意在哪种书架上寻找信息呢？

最后，一些现代数据类型（特别是电影文件）非常大，也非常复杂，无法按传统方式存储。

Note 注意：SQL:2003 标准支持三种数据类型：**预定义、结构化和用户定义**。不同供应商之间实现的用户定义数据类型大不相同，本章主要介绍前两种数据类型。

3.1 不附加字符串

一般而言，所有字符串都可以被划分为字符字符串（存储纯文本）和二进制字符串，二进制字符串用于存储非传统数据，如图像、声音，甚至计算机程序。

3.1.1 字符串

字符串可以被定义为属于某个预定义**字符**集的字符序列。字符串的长度就是该序列中的字符数。每个字符可以用一个或多个字节表示，具体取决于字符所属的字符集。长度为 0 的字符串称为**空字符串**。根据实现，空字符串可以是 NULL（本章末尾将介绍这一特殊概念）的等价形式，也可以不是。SQL:2003 特别区分了空字符串和 null，但所有大型 RDBMS 供应商均不支持这一标准。

Note 注意："字符集"可以想象成给每个属于字符集的字符分配了一个唯一二进制数字（计算机硬件唯一能理解的格式）的表。排序规则可以被定义为一组规则，此规则指定字符串之间相互比较的方式，通常用于对大量字符串排序。

SQL 中所有字符串的长度可以固定，也可以变化。两者之间的区别很明显，但对于没有任何技术背景的人而言并不容易理解，因此，下面小节对此进行详细介绍。

1. 固定长度字符串

如果把字符串定义为固定长度，那么系统会在内存或硬盘上分配需要的字节数。无论存储在这个字符串中的实际值长度是刚好为分配的字节数、只是一半，还是只有一个字符，数据都将始终占有整个分配空间（未使用的字节用空字符填充），因此，所有字符串的长度将完全相同。例如，假设定义了一个字符字段长度为 13 的 DATABASE 列，现在要在此列中存储三个字符串：ORACLE、DB2 和 MS SQL SERVER。图 3-1 演示了该操作的结果，注意所有字符串的长度都是 13 个字节。

O	R	A	C	L	E	blank	blank	blank	blank	blank	blank	blank
D	B	2	blank	blank	blank	blank	blank	blank	blank	blank	blank	blank
M	S	blank	S	Q	L	blank	S	E	R	V	E	R

图 3-1 固定长度字符串存储

2. 可变长度字符串

如果把 DATABASE 列定义为最多可存储 13 个字符的可变长度字符串，那么存储图将发生变化。保存值所需的实际内存或磁盘空间是动态分配的。因此，所有三个字符串将具有不同的长度，而且需要不同的字节数来存储它们。字符串 ORACLE 使用 6 个字节，DB2 使用 4 个字节，而 MS SQL SERVER 则占用允许的最大值——13 个字节（参见图 3-2）。

O	R	A	C	L	E							
D	B	2										
M	S	blank	S	Q	L	blank	S	E	R	V	E	R

图 3-2 可变长度字符串存储

www.TopSage.com

下面给出一些与平台无关的常规建议：当期望值具有近似大小时，使用固定长度数据类型；当期望值的大小不相同时，则使用可变长度数据类型。在上面的例子中，使用可变长度字符串似乎更合理，但如果需要一个列来存储具体字符值（如性别值M或F，或者标志指示符Y或N），那么固定长度字符串会更合适。

3. 区域字符串

尽管英语非常流行，但它并不是地球上唯一的语言。而且，虽然实际上几乎所有大型软件公司都在美国，但市场有它自己的规则。例如，Oracle是一家全球很流行的数据库供应商，它的客户遍及中国、印度、韩国、德国、法国、以色列、俄罗斯、沙特阿拉伯和许多其他国家。

Note 注意：Oracle在145个国家/地区都有客户，但如果客户只能选择英语会怎样？答案很可能是“拒绝”。客户必须能够用他们本国的语言来存储和使用信息，否则，他们宁可使用提供了这一选择的某些效率更低或更昂贵的DBMS供应商。

现在有一个问题需要解决。到目前为止，我们一直在把术语“字符”和“字节”作为同义词使用。字节是一个计算机术语，表示信息存储单位，由8位组成，每一位是1或0，8位的组合可以存储256（2^8）个不同的值（或者说是由0到255之间的数字表示的256个不同的字符）。这似乎很多，但实际上并不是。您需要单独存储大写字母、小写字母、标点符号、数字、数学符号等。剩下的空间仅仅能够存储那些使用拉丁字母的语言中所使用的不同字符，更不用说没有使用拉丁语的语言了。世界上大约有3 000种不同的语言，包括废弃的和在用的，除了J.R.R. Tolkien的Quenya、Sindanin和Entish这样的结构化语言，以及来自虚构**Star Trek**星球的Klingon外，绝大多数语言都有它们自己独特的字母表。

Note 注意：ASCII（American Standard Code for Information Interchange，美国标准信息交换代码）作为一项ANSI标准于1968年发布。它使用上面提到的256个存储单元来存储不同的字符，直到今天，它仍然是一项有用的标准。

解决方案非常简单——每个字符使用两个字节而不是一个。于是便有65 535（$2^{16}-1$）种不同的组合，足够存储地球上几大语言中的大多数字符。

Note 注意：事实上，即使65 535种也不足以存储世界上的所有现有字符，所以除了双字节Unicode表示形式，还有4字节表示形式。4字节有2 147 483 647（$2^{31}-1$）种不同的组合。

Unicode的介绍

ASCII的最终接班人是Unicode。它是一个标准的双字节（或4字节）字符集，为每个字符分配一个唯一的数字，因此，所有这些字符可以用一个字符集来表示。按数据库术语讲，Unicode字符数字就像是一个索引世界上几乎所有字符的主键（另一个使用数据库的优秀示例）。Unicode标准是名为Unicode协会的非营利组织联合开发的成果。Unicode的第一个版本Unicode 1.0于1991年发布，从那以后，Unicode开始被Apple、HP、IBM、Microsoft、Oracle、SAP、Sun、Sybase、Unisys这样的行业领导者和许多其他公司采用（目前的版本是Unicode 5.0，于2006年发布）。

Unicode是XML、Java、JavaScript、CORBA、WML和HTML这类现代标准所必需的，大致对应于ISO标准10646的官方途径。大多数操作系统、所有现代的浏览器、主要的RDBMS供应商以及许多其他产品都支持Unicode。Unicode标准的出现以及支持它的工具的可用性是最重要的全球最新软件技术趋势之一。

SQL有两种用于排列区域字符串顺序的数据类型：固定长度区域字符串和可变长度区域字符串。这两种数据类型的行为与前面描述的完全一样——分别对应于固定长度字符串和可变长度的字符串，只是前者的目的是支持各种区域字符集。大多数数据库供应商会倾向于使NATIOAL CHARACTER数据类型与Unicode一样，不过这不符合SQL:2003标准。因此，如果把DATABASE列声明为一个长度为13的区域字符字段，它仍然能容纳13个字符，但会为每一个字母保留2个字节，总共是26个字节。不同的是，现在实际上可以用任何语言来存储前面示例中拼写的名称了，例如，用俄语，如图3-3所示。

www.TopSage.com

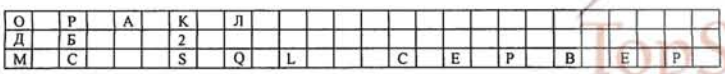


图 3-3 固定长度的 Unicode 字符存储

图 3-4 显示了可变长度区域字符的同一概念。

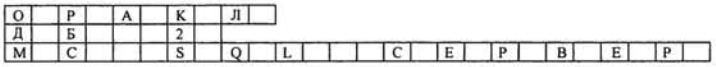


图 3-4 可变长度的 Unicode 字符存储

Note 注意：在俄语中，Oracle 仅用 5 个字符拼写，而不是英语中的 6 个，因此，该单词只需占用 5 个内存（或硬盘）存储单元，不过现有每个存储单元是两个字节。

表 3-1 中总结了各种字符串类型的 SQL:2003 标准和实现规范。

表 3-1 各主要供应商的字符串数据类型的实现

SQL-2003	Oracle 11g	DB2 9.5	Microsoft SQL SERVER 2008
CHAR[ACTER][(n)]	CHAR[ACTER][(n)]	CHAR[ACTER][(n)]	CHAR[ACTER][(n)]
CHAR[ACTER] VARYING(n)或 VARCHAR(n)	CHAR[ACTER] VARYING(n) VARCHAR(n) VARCHAR2(n) LONG[VARCHAR]	CHAR[ACTER] VARYING(n) VARCHAR(n) LONG VARCHAR	CHAR[ACTER] VARYING[(n)] VARCHAR[(n)]TEXT
CLOB 或 CHARACTER LARGE OBJECT	CLOB	CLOB[(n)] [K|M|G]	VARCHAR(MAX)
NATIONAL CHAR[ACTER][(n)] 或 NCHAR[(n)]或 CHARACTER SET <char_set_name>	NATIONAL CHAR[ACTER] [(n)] NCHAR[(n)]	GRAPHIC[(n)]	NATIONAL CHAR[ACTER] [(n)]NCHAR[(n)] NTEXT
NATIONAL CHAR[ACTER] VARYING(n)或 NCHAR VARYING(n) 或CHARACTER VARYING(n) CHARACTER SET <char_set_name>	NATIONAL CHAR[ACTER] VARYING(n) NCHAR VARYING(n) NVARCHAR2(n)	VARGRAPHIC(n) LONG VARGRAPHIC(n)	NATIONAL CHAR[ACTER] VARYING[(n)] NCHAR VARYING[(n)] NVARCHAR[(n)]
NATIONAL CHARACTER LARGE OBJECT	NCLOB	DBCLOB[(n)] [K|M|G]	NVARCHAR(MAX)

4. SQL:2003

SQL:2003 将字符串定义为来自某个字符集的字符序列。字符串的长度可以固定，也可以变化。最小字符长度是 0（空字符串），最大字符串长度由实现定义。

SQL:2003 主要有 3 种字符串类型：CHARACTER、CHARACTER VARYING 和 CHARACTER LARGE OBJECT。此外，还有 NATIONAL CHARACTER、NATIONAL CHARACTER VARYING 和 NATIONAL CHARACTE LARGE OBJECT。

CHARACTER 也可简写为 CHAR，用 CHARACTER(n)这样的形式随意指定大小。例如，CHARACTER (15) 最多可以存储 15 个字符长的字符串。如果忽略大小，则默认为 1。如果用户试图存储一个比声明尺寸大的字符串，就会显示错误消息。

CHARACTER VARYING 可以简写为 CHAR VARYING 或 VARCHAR。必须为要存储的字符串指定最大尺寸，例如，CHARACTER VARYING（15）存储的字符串最多只能包含 15 个字符。

NATIONAL CHARACTER（NATIONAL CHAR、NCHAR、CHARACTER CHARACTER SET<char_set_name>）为实现的国家/地区指定默认数据类型。这是一个固定长度字符串数据类型。

NATIONAL CHARACTER VARYING（NATIONAL CHAR VARYING、NCHAR VARYING、CHARACTER VARYING CHARACTER SET<char_set_name>、CHAR VARYING CHARACTER SET<char_set_name>）是一个可变长度的特定于国家/地区的字符串数据类型。

www.TopSage.com

CHARACTER LARGE OBJECT（CLOB）和 NATIONAL CHARACTE LARGE OBJECT（NCLOB）用于存储大型的（比其他字符串类型所允许的长度更长）可变大小和复杂文本对象，如员工的简历、论文、书籍以及其他类似数据。

5. Oracle 11g

Oracle 在很大程度上符合 SQL:2003 字符串标准。

- CHAR 用于固定长度字符串。一个 CHAR 列的默认长度是 1 个字节，最大长度是 2 000 个字节。
- VARCHAR2 是一个 Oracle 数据类型，用于存储可变长度字符串。它没有默认长度，因此，必须指定一个从 1 到 4 000（VARCHAR2 的最大字节数）之间的值。
- NCHAR 和 NVARCHAR2 用于存储固定长度和可变长度的区域字符串。从 Oracle 9i 开始，它们被重新定义为只能是 Unicode 数据类型，可以分别存储最多 2 000 和 4 000 个字符。这意味着如果将一列声明为 CHAR(100)，则每列将分配 100 个字节，但基于 Unicode 的 NCHAR(100) 列需要 200 个字节，实际数据要受到最多 2 000 字节（对于 NCHAR）和最多 4 000 字节（对于 NVARCHAR2）限制；因此，如果每个字符使用两个字节，NCHAR 列的实际最大限制为 1 000 个字符（NVARCHAR2 则是 2 000 个字符）。

Note 注意：目前，Oracle 中的 VARCHAR 数据类型是 VARCHAR2 的同义词。如果把一列声明为 VARCHAR（30），那么它将被自动转换为 VARCHAR2（30）。Oracle 不建议将 VARCHAR 用作数据类型，而是建议改用 VARCHAR2，因为关键字 VARCHAR 可能会在以后有其他用途。

- 在 Oracle 中，CLOB 和 NCLOB 最多可以存储 8TB 数据。固定长度（例如，每个字符 2 个字节）和可变长度（例如，每个字符 1 个、2 个、3 个、4 个或 5 个字节）字符集均被支持。CLOB 使用 CHAR 数据库字符集，NCLOB 使用区域字符集来存储 Unicode 数据。Oracle 提供了有效的对 CLOB 和 NCLOB 基于字符的随意访问。
- LONG 是一个旧 Oracle 数据类型，用于存储最多包含 2GB 数据的可变长度字符串。它类似于 VARCHAR2，但有许多限制。例如，不能在一个 SELECT 语句（在第 8 章中讨论）的 WHERE 子句中使用 LONG，表不能有多个 LONG 列，它不能被索引等。Oracle 强烈建议停止使用 LONG 数据类型而改用 CLOB。

Note 注意：Oracle 有与 SQL:2003 兼容的同义词。例如，可以使用 CHARACTER（100）而不是 CHAR（100），或者用 CHARACTER VARYING 而不是 VARCHAR2 来获得相同的结果（参见表 3-1）。

6. DB2 9.5

DB2 有一下字符串数据类型：

- CHARACTER 符合 SQL:2003 标准。最大长度是 254 个字符。默认长度是 1。
- VARCHAR 用于可变长度字符串，最大长度是 32 672 个字符。
- LONG VARCHAR 实际上与 VARCHAR 一样，但能存储更大的值（最多 32 700），并且没有一个有用户提供的可选长度限制。
- CLOB 类型是最多 2GB 数据的可变长度字符串。可选最大长度可以用 KB（K 或 k）、MB（M 或 m）或 GB（G 或 g）来表示。例如，CLOB（10M）最多允许 10 048 576 个字符。
- GRAPHIC 大致等价于 NATIONAL CHARACTER。它是一个双字节数据类型，范围从 1 到 127 个字符。如果没有指定长度，则长度默认为 1。
- VARGRAPHIC 是一个可变长度的双字节字符串数据类型，对应于 SQL:2003 的 NATIONAL CHARACTER VARYING。范围从 1 到 16 366。
- LONG VARGRAPHIC 类似于最大长度为 16 350 的 VARGRAPHIC。它没有由用户提供的可选长度限制。
- DBCLOB 是一个等价于 CLOB 的双字节数据类型。最大存储量是一个“GB 字符”（1 073 741 823 双字节字符）。DBCLOB 接受以 KB（K 或 k）、MB（M 或 m）或 GB（G 或 g）表示的最大长

www.TopSage.com

度，与 CLOB 一样。

Note 注意：LONG VARCHAR 和 CLOB 数据类型有特殊限制。例如，SELECT DISTINCT、GROUP BY 和 ORDER BY 等子句中不允许使用这两种数据类型。

请参见表 3-1 了解更多信息。

7. Microsoft SQL Server 2008

下列字符串数据类型受 MS SQL Server 支持：

- CHAR 和 VARCHAR 分别用于固定长度和可变长度字符串数据。最大长度是 8 000 个字符。与 Oracle 不同，也不需要按照 SQL 标准，为 VARCHAR 指定长度——与 CHAR 一样，它默认为 1。
- TEXT 是可变长度字符串数据类型，能保存较大值。它的最大长度是 2 GB，即 $2^{31}-1$（2 147 483 647）个字符。TEXT 在 SQL 查询中使用时会受到一定的限制，已经被 Microsoft 标记为过时。该类型可能在以后的发行版中不再受支持，因此 Microsoft 强烈建议将它转换为 VARCHAR(MAX)。
- NCHAR、NVARCHAR 和 NTEXT 是 CHAR、VARCHAR 和 TEXT 的 Unicode 等价形式。NCHAR 和 NVARCHAR 最多可存储 4 000 个字符；NTEXT 则更大，达到 1 “GB 字符”或 $2^{30}-1$（1 073 741 823）个字符，前面段落中描述的有关 TEXT 的内容在此也有效。Microsoft 建议将 NTEXT 列转换为 NVARCHAR(MAX)。
- SQL Server 2005 引入了 MAX 数据类型，它包含两个新字符串数据类型：VARCHAR(MAX) 和 NVARCHAR(MAX)。最多可以存储 2GB 数据，但与 TEXT 不同，这两个新字符串数据类型与所有 SQL Server 字符串函数完全兼容。

要了解 SQL:2003 兼容同义词，请参见表 3-1。如果一个数据类型有多个名称（或同义词），则最常使用的名称会用斜体给出。

8. 字符串文字

术语“文字”和“常量”指一个固定数据值，例如：

```
'Frozen Margarita'
'ALEX'
'2003/08/07'
'10101101'
```

上面都是字符文字。字符文字包含在单引号中。要表示文字中的单引号，可以输入两个单引号：

```
'0' 'Neil'
```

字符文字在所有三大供应商之间保持惊人的一致，只有细微的变化。例如，如果因连接而关闭 QUOTED_IDENTIFIER 选项，那么 Microsoft SQL Server 允许字符文字使用双引号而不是单引号。要表示一个区域字符集文字，必须在它前面加上字母 N（DB2 除了能理解 N，还能理解 G）：

```
N'Jack Smith'
N'Boris M. Trukhnov'
N'123 OAK ST.'
```

使用此符号输入的文本会转换为区域字符集。

3.1.2 二进制字符串

与字符串一样，二进制字符串也是一个字节序列。但与通常包含文本格式信息的字符串不同，二进制串用于存储非传统数据，如图像、音频和视频文件、程序可执行文件等。二进制字符串可以用于与字符串类似的目的（例如，以 Microsoft Word 格式存储文档），但两种数据类型不兼容；两者的差别如同文本和同一文本的照片。二进制字符串数据类型的总结如表 3-2 所示。

表 3-2 二进制字符串数据类型

SQL:2003	Oracle 11g	DB2 9.5	Microsoft SQL SERVER 2008
BLOB	BLOB	BLOB[(n)]	VARBINARY(MAX)
or	LONG RAW	[K|M|G]	VARBINARY[(n)]
BINARY LARGE	RAW(n)		IMAGE
OBTECT			

大家网

字符与特殊文件

可能会有人感到迷惑：为什么纯文本文档可以被存储为字符串，而 Word 文档必须视为二进制字符串来处理。原因如下：从用户角度看，Word 文件是一个文本文档，但从计算存储角度看却不是这样的。除了纯文本字符外，Word 文件还包含许多只有 Microsoft Word 软件才能解释的特殊符号和指令。任何其他的特殊文件（位图、电子表格、音频和视频文件等）也是如此。可以这样理解：特殊文件（如 DOC、XLS、BMP 或 AVI 类型）就像一盘 VCR 的磁带，而程序（如 Microsoft Word、Excel、Paint 或 QuickTime Player）就像是 VCR。播放磁带必须要有 VCR，而且必须是正确的 VCR——试图在一台欧洲视频录像机（PAL 格式）上播放一盘以 NTSC 格式录制的标准美国 VHS 磁带是不可能实现的。可能会在屏幕看到一些闪烁，也可能会听到一些噪音，但肯定不会看到电影。试着用记事本打开 Word 文件，就会明白这一问题。

1. SQL:2003

SQL:2003 只有一种数据类型 BLOB，该数据类型用来存储二进制字符串。对其他两种数据类型 BIT 和 BIT VARYING 的支持已经停止了。BLOB 被定义为一个字节序列，与字符串不同的是，它没有关联的字符集或排序规则。SQL:2008 还将拥有两种新数据类型：BINARY 和 BINARY VARYING (VARBINARY)。

2. Oracle 11g

Oracle 支持 BLOB 数据类型。它最多能存储 8TB 的二进制数据。此外，Oracle 还有两种其他数据类型来存储二进制字符串 RAW（最多 4 000 个字节）和 LONG RAW（最多能存储 2GB 数据）。LONG RAW 是一个过时的数据类型，提供它只是为了向后兼容。Oracle 建议将 LONG RAW 列转换为 BLOB。

3. DB2 9.5

DB2 中唯一用于二进制字符串的数据类型是 BLOB，它最多可以存储 2GB 数据。

4. Microsoft SQL Server 2008

Microsoft SQL Server 有 4 种用于二进制字符串的数据类型：BINARY、VARBINARY、VARBINARY (MAX) 和 IMAGE。

- BINARY 是一个固定长度数据类型，用于存储二进制数据。可以在 1 到 8 000 之间指定大小；实际存储量是大小加上 4 个字节。
- VARBINARY 可以存储可变长度二进制数据。大小从 1 到 8 000。存储大小是输入数据的实际长度加上 4 个字节。输入数据的长度可以是 0 字节。
- IMAGE 是一个可变长度二进制数据类型，可以存储从 0 到 2 147 483 647 字节（2GB）的数据。
- VARBINARY (MAX) 行为与 VARCHAR (MAX) 和 NVRCHAR (MAX) 的（如本章前面所述）非常类似。该类型最多可以存储 2GB 数据。

5. 二进制字符串文字

Microsoft SQL Server 允许将文字用于二进制字符串字段（BINARY、VARBINARY 和 IMAGE），使用形式可以是带前缀 0x 的十六进制数字，或者是二进制字符串。值必须不带引号：

```
0XAE
0101010010100110
```

Microsoft SQL Server 自动将这些文字转换为合适的二进制格式。Oracle 使用单引号包围十六进制数字来表示将被插入定义为 RAW 数据类型的列的文字值。DB2 没有二进制字符串文字；必须使用特殊的函数将值转换为适当的格式（请参阅第 10 章）。

Note 注意：那些可以存储数个 GB 数据的大型对象很少需要使用文字。在大多数情况下，LOB 不是由传统 SQL 语句操纵的，而是通过特殊程序和接口进行访问，这些程序和接口知道如何处理这类对象，而无需把它们直接读取到内存中。

3.2 数字

通常可以将所有数字数据分成两类：**精确数字**和**近似数字**。

www.TopSage.com

3.2.1 精确数字

精确数字可以是整数（例如，数字主键、订购项目数等之类的数量或年龄），也可以带有小数点（如价格、重量或百分比中的数字）。数字可以正也可以负，有**精度**和**小数位数**。精度决定可以被存储的最大小数位数（包括小数点左边位数和右边位数）。小数位数指定允许的最大小数位。精确数字数据类型的总结如表 3-3 所示。

表 3-3　　精确数字数据类型

SQL:2003	Oracle 11g	DB2 9.5	Microsoft SQL SERVER 2008
INT[EGER]	NUMBER(38)	INT[EGER] BIGINT	INT[EGER] BIGINT
SMALLINT	SMALLINT NUMBER(38)	SMALLINT	SMALLINT TINYINT
NUMERIC[(p[,s])] 或 DEC[IMAL] [(p[,s])]	NUMERIC[(p[,s])] DEC[IMAL][(p[,s])] NUMER [(p[,s])]	NUMERIC[(p[,s])] DEC[IMAL] [(p[,s])]	NUMERIC[(p[,s])] DEC[IMAL][(p[,s])] MONEY SMALLMONEY

1. SQL:2003

SQL:2003 指定了下列用于精确数字的数据类型：NUMERIC、DECIMAL、INTEGER、SMALLINT 和 BIGINT（参见表 3-3）。

- NUMERIC 数据类型可以存储带有特殊小数部分的数字以及整数。可选小数位数指定受支持的小数点位置。
- DECIMAL 与 NUMERIC 非常类似。唯一的区别是，某个供应商特定实现使用的精度（不是小数位数）可以大于声明中使用的精度。
- INTEGER 表示可数数字；它的精度特定于实现。
- SMALLINT 实际上与 INTEGER 一样，只是其最大精度可以比 INTEGER 的精度小。
- BIGINT 实际上与 INTEGER 一样，只是其最大精度可以比 INTEGER 的精度大。

NUMERIC 和 DECIMAL 数据类型的精度和小数位数

NUMERIC 和 DECIMAL 值的小数位数和精度常会引起混淆。请记住，精度指定一个值所允许的所有数字的最大数目。例如，假设一个表有以下列：

```
column1  NUMERIC(10,4)
column2  NUMERIC(10,2)
column3  NUMERIC(10,0)
```

那么，column1 中可以存储的最大数字是 999 999.9999；column2 可以存储的最大值为 99 999 999.99；column3 则能存储 9 999 999 999。要确定小数点之前的最大位数，用精度减去小数位数即可。如果试图插入的值的小数点之前的位数大于列所允许的位数，则会得到一条错误消息，但小数位数比指定位数多的值只会简单地四舍五入到所有“三大”数据库（SQL:2003 标准允许选择是四舍五入，还是静默截断）。例如，把 999 999.9999 插入 column2（或 column3），该值将被四舍五入为 1 000 000，但试图将 column1 设置为 99 999 999.99 则会失败。

2. Oracle 11g

Oracle 用一个数据类型 NUMBER 表示所有数字数据，而且有大量符合 SQL:2003 的同义词（参见表 3-3）。INTEGER 和 SMALLINT 将被转换为 NUMBER（38）；NUMERIC 和 DECIMAL 将被 NUMBER 替代。NUMBER 数据类型存储 0、正的和负的定点和浮点数字，范围从$-1.0*10^{-103}$到$9.9\cdots9*10^{125}$，精度为 38 位。空间是动态分配的，因此，Oracle 宣称用一个数字数据类型来表示所有数字数据不会损害性能。

3. DB2 9.5

DB2 有 4 种用于精确数字的数据类型：INTEGER、SMALLINT、BIGINT 和 DECIMAL(NUMERIC)。

- INTEGER 是一个具有 10 位精度的 4 字节整数。它可以存储-2^{31}（-2 147 483 648）到$2^{31}-1$（2 147 483 647）之间的值。
- SMALLINT 用于保存更小的整数。存储大小是两个字节，范围从-2^{15}（-32 768）到$2^{15}-1$（32 767）。
- BIGINT 是一个具有 19 位精度的 8 字节整数。它的范围从-2^{63}（-9 223 372 036 854 755 808）

www.TopSage.com

到2^{63}–1（9 223 372 036 854 775 807）。

- DECIMAL 数据类型（相当于 NUMERIC）被指定为一个隐含小数点的十进制数。最大精度是31位，范围从$-10^{31}+1$到$10^{31}-1$。

4. Microsoft SQL Server 2008

相比 Oracle 和 DB2，Microsoft SQL Server 有更多用于精确数字数据的数字数据类型。除了 INT、BIGINT、SMALLINT 和 TINYINT，它还提供了 MONEY 和 SMALLMONEY。

- INT（或 INTEGER）存储-2^{31}到$2^{31}-1$之间的整数。它占用四个字节。
- BIGINT 存储-2^{63}到$2^{63}-1$之间的大整数。存储大小是8个字节。BIGINT 用于那些 INTEGER 范围足以满足需要的特殊场合。
- SMALLINT 用于范围从-2^{15}到$2^{15}-1$之间的更小整数。
- TINYINT 为0到255之间的小非负整数提供了方便。它只用一个字节来存储这些数字。
- DECIMAL 符合 SQL:2003 标准。NUMERIC 是 DECIMAL 的同义词（参见表 3-3 了解其他同义词）。有效值的范围是从$-10^{38}+1$到$10^{38}-1$。
- MONEY 是一个特殊的 8 字节 Microsoft SQL Server 数据类型，用于表示货币值。范围从–922 337 203 685 477.5808 到 922 337 203 685 477.5807，精确到千分之十。
- SMALLMONEY 是另一个货币数据类型，用于表示更小的数额。它是 4 个字节，可以存储–214 748.3648 到 214 748.3647 之间的值，精度与 MONEY 一样。

Note 注意：为什么要用特殊的数据类型存储货币值呢？重要原因是一致性。估计所有会计师都知道舍入错误会带来多大麻烦。例如，一个用于美元数额的列被声明为 NUMERIC（12，2），另一个被声明为 NUMERIC（14，4）。如果处理很大总额，误差很容易就会达到数百甚至数千美元。从另一个角度看，实际上用于相同实体的不同数据类型会导致混乱，因此，Oracle 对所有数字数据只使用一种数据类型。读者可以判断哪种方法更有效

5. 用于精确数字的文字

用于精确数字的文字用数字字符串来表示，可以选择在前面添加加或减号，对于 NUMERIC 和 DECIMAL 数据类型，可以选择用一个点（.）来分隔小数部分：

```
123
-33.45
+334.488
```

Oracle 允许用单引号引住文字：

```
'123'
'-677.34'
```

Note 注意：Microsoft SQL Server 有用于 MONEY 和 SMALLMONEY 数据类型的文字格式，用数字字符串表示，带有一个可选小数点，而且可以带有货币符号前缀：

```
$12
$542023.14
```

选择正确的数据类型

误用数据类型是没有经验的数据库开发人员经常做的事情，而且可能导致严重的后果。

例如，把一个与金钱有关的字段定义为 FLOAT 或 NUMERIC（12，1）会导致舍入错误（会计师不免要埋怨）。主键列精度不够（例如，在 ORDER_HEADERD 表中使用 ORDHDR_ID_N NUMBER(5)）会工作一段时间，但插入一定数量的记录（本例中是 99 999）后，将无法再插入新的行——主键的下一个值（100 000）对于 NUMBER（5）精度而言已经不合适了。

上一个例子很容易修正——数字列的精度很容易调整（更多信息将在下一章介绍）。这正是关系数据库优于早期系统的好处之一。不过，找出问题原因并修复它会花费一些时间，而且如果数据库是一个如大型 24/7 订购管理系统之类的系统，用户是不会接受这种延迟的。

3.2.2 近似数字

近似数字是那些不能用绝对精度表示的数字（或者没有一个精确的值）。近似数字数据类型的总结如表 3-4 所示。

www.TopSage.com

表 3-4 近似数字数据类型

SQL:2003	Oracle 11g	DB2 9.5	Microsoft SQL SERVER 2008
FLOAT[(p)]	FLOAT[(p)] NUMBER	FLOAT[(p)]	FLOAT[(p)]
REAL	REAL NUMBER	REAL	REAL
DOUBLE PRECISION	DOUBLE PRECISION NUMBER	BOUBLE [PRECISION]	DOUBLE PRECISION

Note

注意：一个经典例子是数字π，它通常近似为 3.14。这个数字大约在 4 500 年前的古巴比伦和古埃及就开始使用了，并且一直是数学家们感兴趣的问题，从阿基米德到现代科学家。如今，π的 206 158 430 208（$3*2^{36}$）位小数已经被计算出来了。如果把这个值以书面格式存储，则大约需要四千万张纸或 5 万卷书。

1. SQL:2003

SQL:2003 指定了下列用于近似数字的数据类型：FLOAT、REAL 和 DOUBLE PRECISION。

- FLOAT 用于存储浮点数，其精度可由用户随意指定。
- REAL 与 FLOAT 类似，但其精度是固定的。
- DOUBLE PRECISION 实际上与 REAL 一样，只是精度更大。

2. Oracle 11g

前面提到，Oracle 只有一种数字数据类型 NUMBER，用于精确数字和近似数字。Oracle 也支持 FLOAT 和 DOUBLE，但这些只是 NUMBER 数据类型的别名，具有一些其他限制。

3. DB2 9.5

DB2 有用于近似数字的 REAL 单精度数据类型和 DOUBLE 双精度数据类型。FLOAT 是 DOCBLE 的同义词。

- REAL 是一个 4 字节的实数近似值。范围从–3.402E+38 到–1.175E-37，或从 1.175E–37 到 3.402E+38。它也包括 0。
- DOUBLE 需要 8 个字节的存储，而且比 REAL 的精度更高。数字可以是 0，或者是从 –1.79769E+308 到–2.225E-307，或者从 2.225E–307 到 1.79769E+308。

4. Microsoft SQL Server 2008

Microsoft SQL Server 有一个用于浮点数的数据类型——FLOAT(n)。它也有几个符合 SQL:2003 的同义词（参见表 3-4）。

FLOAT 数据类型可以存储与 DB2 中的 DOUBLE 范围相同的实数。实际存储大小为 4 个或 8 个字节，具体取决于 n 的值。

5. 用于近似数字的文字

用于精确数字的文字可用作实数的文字，不过可以另外指定实数，其中两个数字用大写 E 或小写 e 分开（科学记数法）。这两个数字都可以包括加号或减号，第一个数字也可以包含小数点：

```
+1.23E2
-3.345e1
-3.44488E+002
```

常量的值通过第一个数字得到，10 的幂由第二个数字指定。

3.3 日期和时间数据类型

日期和时间处理可能是 SQL 中最复杂和最容易产生矛盾的主题之一。基于我们的个人经验，操作日期经常会引起混淆甚至令人沮丧，这些情况不仅数据库用户会遇到，在开发人员社区中也会发生。

3.3.1 复杂数据类型介绍

其中一个问题是实际日期与 SQL 查询检索到的不同。迄今为止，本章处理的一直是每行存储一个值的简单数据类型。日期和时间数据类型要存储许多元素（年、月、日、小时等）。在编程时，这样的

www.TopSage.com

数据类型称为**复杂**数据类型，经常表示为结构。当作为一个数据库查询结果返回时，日期和时间字段看似字符串，但实际上，它们是结构的一部分，类似于下例中的一些结构（本例子没有使用任何特定编程语言，而是某种伪码）。

```
STRUCTURE DATE
{
  YEAR          DECIMAL(4,0),
  MONTH         DECIMAL(2,0),
  DAY           DECIMAL(2,0)
}

STRUCTURE TIME
{
  HOUR          DECIMAL(2,0),
  MINUTE        DECIMAL(2,0),
  SECOND        DECIMAL(5,3)
}

STRUCTURE DATETIME
{
  YEAR          DECIMAL(4,0),
  MONTH         DECIMAL(2,0),
  DAY           DECIMAL(2,0),
  HOUR          DECIMAL(2,0),
  MINUTE        DECIMAL(2,0),
  SECOND        DECIMAL(5,3)
}
```

显示的值只是这个结构字段的格式和连接。例如，对于YYYY/DD/MM格式，伪代码可能如下：

```
CONCAT(CAST(DATETIME.YEAR, STRING), '/',
       CAST(DATETIME.DAY, STRING), '/',
       CAST(DATETIME.MONTH, STRING))
```

交叉参考：上面例子中的点（.）符号在第1章中介绍过。

DBMS应该提供其他函数来以各种不同的格式表示日期和时间，例如，以12或24小时的形式显示时间、把星期几显示为某个日期、显示世纪、把时间转换为不同的时区等。

3.3.2 日期和时间实现

前面提到过，SQL:2003规定了日期和时间数据类型，不同的RDBMS实现会以不同的方式进行处理。日期和时间数据类型的总结如表3-5所示。

表3-5 日期和时间数据类型

SQL:2003	Oracle 11g	DB2 9.5	Microsoft SQL SERVER 2008
DATE	DATE	DATE	DATETIME SMALLDATETIME
TIME[WITH TIME ZONE]	DATE	TIME	DATETIME SMALLDATETIME
TIMESTAMP[(p)] [WITH TIME ZOME]	DATE TIMESTAMP[(p)] [WITH[LOCAL]TIME ZOME]	TIMESTAMP	DATETIME SMALLDATETIME
INTERVAL	INTERVAL DAY [(p)] TO SECOND [(p)] INTERVAL YEAR [(p)] TO MONTH		

1. SQL:2003

SQL:2003支持DATE、TIME[WITHOUT TIME ZONE]、TIMESTAMP[WITHOUT TIME ZONE]、TIME WITH TIME ZONE、TIMESTAMP WITH TIME ZONE和INTERVAL数据类型。

- DATE数据类型是一个由三个元素组成的结构：年、月和日。年是一个4位数，允许的值从0000到9999；月是一个两位元素，允许的值从01到12；日也是一个两位数，取值范围从01到31。SQL:2003使用上述结构定义日期和时间语义，但实现不需要使用这种方法——假定实现生成同样的结果。一个供应商可能选择类似上面这样的结构；其他供应商则可能实现字符、具有不同小数位数的数字等。
- TIME由小时、分钟和秒组成。小时是一个从00到23的数字；分钟是一个从00到59的两

www.TopSage.com

位数；秒则是一个从 00 到 61 的整数，或者是一个小数位数至少为 3、精度至少为 5 的小数，能存储的值从 00.000 到 61.999。

Note 注意：大于 59 的秒值范围是为了表示闰秒，闰秒有时会添加到地球时间上。三大主要供应商中还没有一个实现这个功能。

- TIMESTAMP 组合了 DATE 和 TIME 数据类型，包括年、月、日、小时、分钟和秒。
- TIME WITH TIME ZONE 类似于 TIME，但如果 TIME 表示本地时间，TIME WITH TIME ZONE 除了存储标准的 TIME 元素，还要存储 **UTC 及时区**信息。
- TIMESTAMP WITH TIME ZONE 扩展了 TIMESTAMP，包含了有关时区的信息。
- INTERVAL 可以是年月时间，也可以是日时间。前者包括 YEAR 和 MONTH 字段；后者包括 DAY、HOUR、MINUTE 和 SECOND 元素。

Note 注意：UTC（Coordinated Universal Time，协调世界时），也称为格林威治平均时（GMT），是一个高度精确的国际时间标准。时区即是 UTC 的正负偏移量。

2. Oracle 11g

Oracle 的数据类型有 DATE、TIMESTAMP、TIMESTAMP WITH TIME ZONE、TIMESTAMP WITH LOCAL TIME ZONE、INTERVAL YEAR TO MONTH 和 INTERVAL DAY TO SECOND。

- DATE 是一个稍微会引起混淆的数据类型，尽管它的名称如此，但它实际上同时存储日期和时间信息，大致相当于 SQL:2003 TIMESTAMP。除了标准 SQL:2003 TIMESTAMP 字段（年、月、日、小时、分钟和秒）外，它还包含世纪。年部分的有效范围从 4712 BC 到 4712 AD。Oracle 中的 DATE 长 7 个字节。第一个字节到第 7 个字节分别表示世纪、年、月、日、小时、分钟和秒。
- TIMESTAMP 数据类型实际上与 DATE 是一样的，但可以为秒的小数部分位数指定一个可选精确度。有效值是从 0 到 9，默认是 6。Oracle 中的 TIMESTAMP 大小可变——长度为 7 到 11 个字节。其他 4 个字节可用于表示毫微秒。
- TIMESTAMP WITH TIME ZONE 数据类型的含义不言而喻，它与 TIMESTAMP 唯一的区别就是时区位移包括在它的值中，时区位移指的是本地时间和 UTC 之间在小时和分钟方面的差值。
- TIMESTAMP WITH LOCAL TIME ZONE 数据类型是 TIMESTAMP（或 TIMESTAMP WITH TIME ZONE）的另一个变体。区别在于它被规范化为数据库时区，位移值没有和它一起保存，不过当用户查询这个列时，结果会以用户的本地会话时区形式返回。
- INTERVAL YEAR TO MONTH 数据类型用年和月字段来存储一段时间。
- INTERVAL DAY TO SECOND 数据类型可以用日、小时、分钟和秒来存储一段时间。

理解 TIMESTAMP WITH LOCA TIME ZONE 数据类型

本例帮助读者理解 TIMESTAMP 和 TIMESTAMP WITH LOCAL TIME ZONE 数据类型之间的差异。

假设 ACME 规模扩大了，现在全球都有它的办事处。John 是华盛顿州西雅图市的销售部门主管，需要查看每个区域在前一天开出的发票（对于本例，可以假定每个区域都有自己的数据库，而且数据库之间可以相互通信）。问题是，一个办事处在纽约，另一个办事处在阿姆斯特丹，还有一个办事处位于曼谷等——这意味着每一个区域都在它自己的时区中。至于 John，他需要查看所有新创建的订单（只要是在西雅图的前一天所产生的）——也就是太平洋标准时间（Pacific Standard Time，PST）。提到的每个办事处都属于一个不同的时区，更不用说夏令时（DST）。因此，西雅图的 2003 年 8 月 31 日下午 6:00 相当于纽约的 2003 年 8 月 31 日下午 9:00、阿姆斯特丹的 2003 年 9 月 1 日上午 3:00、曼谷的 2003 年 9 月 1 日上午 8:00。这意味着不仅日不同，甚至连月也不同！

如果 ORDER_HEADER.ORDHDR_INVOICEDATE_D 列的数据类型是 TIMESTAMP，那么 John 将需要一个报告来以编程方式将所有不同区域的发票日期转换为 PST。

如果把 ORDHDR_INVOICEDATE_D 声明为一个 TIMESTAMP WITH LOCAL TIME ZONE，事情会更简单，那样，每次当 John 从西雅图查询一个远程数据库时，结果会以他的会话时间返回，例如，在曼谷的 9 月 1 日上午 8 点（泰国的当地时间）开出的发票将显示为 PST 时间的 8 月 31 日。

www.TopSage.com

大家网 TopSage.com

警告：用于分钟的 Oracle 格式值是 MI，而不是大多数其他数据库中的 MM。因此，注意如果想要指定一个时间格式的话，它应该看似 HH:MI:SS 这样的形式，而不是 HH:MM:SS。MM 在 Oracle 中代表月。

3. DB2 9.5

DB2 有三个标准的 SQL:2003 日期和时间数据类型：DATE、TIME 和 TIMESAMP。

- DATE 由年、月和日组成。年部分的范围从 0001 到 9999；月可以从 1 到 12；日部分的范围则从 1 到 28、29、30 或 31，具体取决于月。DATE 在 DB2 内部存储为一个 4 字节字符串。每个字节表示两个十进制位数。头两个字节存储年，第三个字节存储月，第 4 个字节存储日值。
- TIME 数据类型在内部被表示为一个三字节字符串——一个字节用于小时，一个用于分钟，还有一个用于秒。小时的范围从 0 到 24（如果值是 24，则分钟和秒都是 0）。分钟和秒的部分的值必须在 0 到 59 之间。
- TIMESTAMP 数据类型结合了 DATE 和 TIME 元素，外加一个微秒部分。内部表示形式是一个 10 字节字符串（4 个 DATE 字节、三个 TIME 字节和另外三个用于微秒的字节）。

4. Microsoft SQL Server 2008

Microsoft SQL Server 有两个日期和时间数据类型——DATETIME 和 SMALLDATETIME，两者都表示日期和时间值的组合，但范围不同。

- DATETIME 可以存储 1753 年 1 月 1 日到 9999 年 12 月 31 日之间的值，精确到 0.00333 秒。它在内部存储为一个 8 字节字符串。头 4 个字节表示 1900 年 1 月 1 日之前或之后的天数，1900 年 1 月 1 日是系统参考日期。后 4 个字节存储自指定日的午夜后经过的时间（以毫秒为单位）。
- SMALLDATETIME 可以存储从 1900 年 1 月 1 日到 2079 年 6 月 6 日之间的日期，精确到分钟。该数据类型内部存储为 4 个字节。同样，第一部分（两个字节）存储系统参考日期（1900 年 1 月 1 日）之后的天数，第二部分存储时间（以午夜后经过的分钟数）。

5. 日期和时间文字

各个供应商的日期和时间文字是大不相同的。

6. Oracle 11g

Oracle 在 DATE 值匹配特殊数据库初始化参数 NLS_DATE_FORMAT 时允许把它们指定为文字，NLS_DATE_FORMAT 默认为 DD-MON-YY（Oracle 初始化参数通常由数据库管理员即 DBA 来处理，本书不对此进行讨论）。也可以使用带有 DATE 前缀的 SQL:2003 文字标准（YYYY-MM-DD），或者使用 Oracle 函数 TO_DATE（在第 10 章中讨论）将文字转换为字符串。下面的三条语句是有效的日期文字例子（SHIPMENT_ARRIVDATE_D 是 Oracle ACME 数据库中的一个 DATE 字段）：

```
UPDATE shipment
SET shipment_arrivdate_d = '03-SEP-02'
WHERE shipment_id_n = 30661;

UPDATE shipment
SET shipment_arrivdate_d = DATE '2003-09-02'
WHERE shipment_id_n = 30661;

UPDATE shipment
SET shipment_arrivdate_d =
TO DATE('September, 02 2003', 'Month, DD YYYY')
WHERE shipment_id_n = 30661;
```

下面是不正确的：

```
UPDATE shipment
SET shipment_arrivdate_d = 'YYYY-MM-DD'
WHERE shipment_id_n = 30661;

UPDATE shipment
SET shipment_arrivdate_d = '03-SEP-02 23:12:45'
WHERE shipment_id_n = 30661;
```

TIMESTAMP 和 TIMESTAMP WITH TIME ZONE 数据类型也接受将 DD-MON-YY 格式用于文字（有一个可选的 TIME 部分）；此外，可以用一个 TIMESTAMP 前缀指定文字：

```
TIMESTAMP '1997-01-31 09:26:50.124'
TIMESTAMP '1997-01-31 09:26:56.66 +02:00'
```


www.TopSage.com

```
TIMESTAMP '1999-04-15 8:00:00 -8:00'
TIMESTAMP '1999-04-15 8:00:00 US/Pacific'
TIMESTAMP '1999-10-29 01:30:00 US/Pacific PDT'
```

上面这些行中的第一行用于TIMESTAMP数据类型，其他4行用于TIMESTAMP WITH TIME ZONE。

Note 注意：TIMESTAMP WITH LOCAL TIME ZONE数据类型没有任何与之关联的文字。

在指定时间间隔值作为文字时，Oracle提供了很大的灵活性。请参见表3-6中的示例。

表3-6 Oracle中的时间间隔文字

时间间隔文字	解释
INTERVAL'23-5'YEAR TO MONTH	23年和5个月的时间间隔
INTERVAL'67'YEAR(3)	67年和0个月的时间间隔
INTERVAL'7'YEAR	7年的时间间隔（映射到INTERVAL'7-0'YEAR TO MONTH）
INTERVAL'74'MONTH	映射到INTERVAL'6-2'YEAR TO MONTH并指示6年和2个月
INTERVAL'76:15'DAY TO MINUTE	7天6小时和15分钟的时间间隔
INTERVAL'40'DAY	40天的时间间隔
INTERVAL'11:20'HOUR TO MINUTE	11小时2分钟的时间间隔
INTERVAL'10:22'MINUTE TO SECOND	10分钟22秒的时间间隔
INTERVAL'25'HOUR	25小时的时间间隔
INTERVAL'40'MINUTE	40分钟的时间间隔

7. DB2 9.5

下列格式用于表示DATE文字：YYYY-MM-DD（ANSI/ISO）、MM/DD/YYYY（IBM美国）和DD.MM.YYYY（IBM欧洲）。

因此，下面三个语句是有效的：

```
UPDATE shipment
SET shipment_arrivdate_d = '2003-09-02'
WHERE shipment_id_n = 30661

UPDATE shipment
SET shipment_arrivdate_d = '09/02/2003'
WHERE shipment_id_n = 30661

UPDATE shipment
SET shipment_arrivdate_d = '02.09.2003'
WHERE shipment_id_n = 30661
```

但下面这个是无效的：

```
UPDATE shipment
SET shipment_arrivdate_d = '02-SEP-2003'
WHERE shipment_id_n = 30661
```

有效的TIME文字格式是HH.MM.SS(ANSI/ISO和IBM欧洲)、HH.MM AM/PM(IBM美国)和HH.MM.SS(日本工业标准)。同样，可以包含尾部空白，前导0可以在时间的小时部分中省略，可以完全忽略秒：

```
'12.23.56'
'23:15 AM'
'8:45'
```

TIMESTAMP文字的有效字符串格式是YYYY-MM-DD-HH.MM.SS.NNNNNN和YYYY-MM-DD HH:MM:SS.NNNNNN：

```
UPDATE shipment
SET shipment_createdate_d = '2003-10-12-23.34.29'
WHERE shipment_id_n = 30661

UPDATE shipment
SET shipment_createdate_d = '2003-10-12 23:34:29.345678'
WHERE shipment_id_n = 30661
```

8. Microsoft SQL Server

在处理日期和时间方面，Microsoft SQL Serve可能是最友好的RDBMS了。它可以识别包含在单引号中以许多不同格式表示的日期和时间文字。例如：

```
'August 15, 2003'
'15 August, 2003'
'15-AUG-2003'
'15 Aug, 2003'
```


www.TopSage.com

```
'030815'
'2003/08/15'
'08/15/03'
'14:30:24'
'04:24 PM'
'15 August, 2003 23:00'
'15-AUG-2003 22:45:34.345'
```

所有这些（以及许多其他）格式对 DATETIME 和 SMALLDATETIME 都是有效的。

3.4 XML 数据类型

第 1 章曾讲过，XML 数据模型与关系数据模型有很大区别，前者基于不匹配 RDBMS 领域的层次递归结构。尽管如此，在关系数据库中使用 XML 还是非常方便的，而且这种功能需求一直在增长，因为 XML 被标记语言的国际标准接受。

交叉参考：请参阅第 15 章，了解在关系数据库中使用 XML 的更多信息。

XML 数据类型实现

XML 数据类型由 SQL:2003 规定，由“三大”RDBMS 供应商实现。

1. SQL:2003 与 SQL/XML:2006

SQL:2003 引入了一种新的 XML 数据类型，专门用于存储、搜索和更新 XML 文档。SQL/XML:2006 中的 SQL/XML 标准经过了极大的扩展和改进。新标准定义了许多函数，如将 XML 文档转换为内部 RDBMS XML 表示形式的 XMLPARSE 和以完全相反的方式工作的 XMLSERIALIZE，它将内部 XML 值转换为 XML 字符串。SQL/XML 引入了对 XQuery 的支持，XQuery 是一种 XML 标准查询语言，当前用于访问常见 SQL 数据和 XML 文档。

XML 字段的声明方式与任何其他数据类型字段的声明方式相同：

```
. . .
prod_report XML
. . .
```

2. Oracle 11g

Oracle 从 9i 版开始支持 XML，但 Oracle 11g 对有关 XML 的功能进行了许多大的改进。在以前的发行版中，XMLType 数据类型实际上是基于 CLOB 的用户定义结构，但现在 Oracle 允许在旧方法（以文本格式存储 XML 字段）和新方法（允许以解析的内部即二进制格式存储 XML 数据）之间选择。这种方法极大改进了数据库查询性能，而且也符合新的 SQL 标准。

Oracle 11g 支持大多数 SQL/XML 2006 功能，包括 XMLPARSE 和 XMLSERIALIZE 函数。也实现了 XQuery。

3. DB2 9.5

DB2 提供了 XML 数据类型，以及各种支持功能。XML 数据以解析的内部形式存储，可以使用 XMLSERIALIZE 函数转换为 XML 字符串格式。DB2 符合大多数 SQL/XML 2006 要求。

4. Microsoft SQL Server 2008

XML 本机数据类型首次在 SQL Server 2005 中引入。该数据类型除了可以在创建表时用于关系列，还可以用于变量声明和参数解析。SQL Server 2008 具有可用在 SQL 查询 XML 列中的特殊函数，但与 SQL:2003 的语法不同。

注意：在所有“三大”RDBMS 实现中，XML 列是可以索引和限制的。这些列可以在一定限制下用在 SELECT、INSERT 和 UPDATE 等语句中。

3.5 结构数据类型和用户定义数据类型

读者已经熟悉了复杂数据类型（日期和时间）的概念，但它们的复杂性对用户是隐藏的。现在，

www.TopSage.com

TopSage.com

我们讨论一些需要更多用户干预的复杂数据类型。理解这些数据类型需要有一些面向对象编程和设计的背景知识。

3.5.1 SQL:2003

SQL:2003 大致定义了三组结构数据类型：引用类型（REF）、行类型（ROW）和集合类型（ARRY 和 MULTISET）。前两种数据类型与 RDBMS 中的面向对象功能是密切相关的。

1. 面向对象数据类型

SQL 允许定义的复杂类型称为结构类型（这样叫是因为它们具有结构），定义的方法用于操作这些类型的实例。这些类型可用作 SQL 列的类型，在这种情况下，存储在这些列的行中的值只是值——操作全部由方法定义的值。此外，SQL 允许声明一种新表，即类型表，它的定义基于结构类型，结构类型的每个属性用于确定该表各列的名称和类型。当普通表中与某列有关的一行（经常但不一定）要引用类型表中的一行时，REF 类型即是该列的数据类型，与此列有关的那一行的值被认为是直接指向类型表相应行的“指针”。

SQL 也允许定义一种元组类型，称为 ROW 类型。ROW 类型定义了大量字段，每个字段都有一个名称和一个数据类型（可以是结构类型、另一个 ROW 类型等等）。ROW 类型与结构类型之间的一个重要区别是，结构类型定义为第一类 SQL 对象，有自己的名称，这些名称在声明该结构类型的列时用在列定义中。相反，ROW 类型不单独定义为第一类 SQL 对象，它没有自己的名称。将某列定义为 ROW 类型的列时，ROW 类型的定义显示在列定义中。另一个重要区别是结构类型实例上的操作可以通过为该类型定义的方法来完成，而 ROW 类型完全不与任何用户定义方法关联。

引用类型

引用类型是一种特殊的数据类型，用于引用类型表中的行，引用类型也称为**引用表**。引用值是只读的，而且在数据库系统目录中是唯一的。要声明 REF 数据类型，首先必须创建一个引用表：

```
CREATE TYPE customer_base_AS
(
 cust_id_n NUMERIC,
 cust_name_s VARCHAR(50)
)

CREATE TABLE custbase OF customer_base
```

现在便可以使用 REF 数据类型声明中的类型表了：

```
. . .
customer_nw  REF(CUSTOMER_BASE) SCOP custbase
. . .
```

行类型

术语“行类型”定义为一个字段序列，通过行类型描述符进行描述。行类型的每个字段有一个值。为了使这个概念更好理解，可以将这种类型想象为一个包含许多字段的普通数据行，每个字段都有自己的数据类型。行类型可以包含数字、字符、日期和时间字段；结构类型；甚至其他行类型；以及一些集合类型字段（本章后面介绍）——以一定顺序排列。每个字段包含字段名称和数据类型。ROW 数据类型列的声明类似如下：

```
. . .
customer_nw  ROW(cust_id_n_NUMERIC, cust_name_s VARCHAR(50))
. . .
```

交叉参考：有关行和列定义的内容，请参阅第 4 章。

2. 集合

集合定义为一个组合值，由某一数据类型的零个或多个元素组成。SQL:2003 指定了两种集合：ARRAY 和 MULTISET。

ARRAY

ARRAY 通过存储一个有序同类数据集合扩展了一个字段的容量。下面是一个电话号码字段的数组数据类型示例

```
. . .
phone_no_s  VARCHAR(10) ARRAY[10]
. . .
```


www.TopSage.com

ARRAY 元素的最大数（也称为基数）是用户定义的，或者是实现定义的（如果用户没有指定最大数）。

Note

注意：到目前为止，电话号码还没有给数据库程序员和设计师们造成任何麻烦。通常，人们只有一个住宅电话号码和一个工作电话号码。但这种情况最近几年已经发生了显著变化。手机现在与电视、影碟机和手表一样流行。许多人甚至拥有多部手机。对正确设计关系数据库而言，这不是什么大问题——只需将电话号码存储出在一个具有指向它们所有者的指针的单独表中。在这种情况下，使用数组数据类型是另一个合适的解决方案。

MULTISET

MULTISET 也存储同类数据，例如字符串或数字。两种集合类型之间的主要区别是 MULTISET 元素不以任何特定顺序存储。此外，MULTISET 是一个未绑定的集合，这意味着该数据类型没有预定义元素数。因此，如果上一电话号码示例中使用 MULTISET 而不是 ARRAY，那么应该表示为：

```
. . .
phone_no_s  VARCHAR(10) MULTISET
. . .
```

如读者所见，没有指定“varchar 多集”字段，所以理论上可以存储任意多的电话号码。

ARRAY 和 MULTISET 的元素也可以是集合数据类型。可以声明数组的数组或多集的数组，这意味着可以进一步嵌套集合——例如，数组的多集的数组的数组等，但是不能在 ARRAY 字段存储 MULTISET 值，反之亦然。

3.5.2 Oracle 11g

Oracle 支持面向对象功能和引用数据类型。集合作为嵌套表和可变数组被支持。

1. 面向对象数据类型

Oracle 中的对象可以简单描述为一个实体，或者另外还有一些方法（函数）来操作它。可以使用下面语法在 Oracle 中创建对象：

```
CREATE TYPE        addr_type AS OBJECT (
street             VARCHAR(30),
city               VARCHAR(30),
state              CHAR(2),
zip                VARCHAR(10),
country            VARCHAR(30)
type_of_addr       CHAR(1));
```

现在可以创建带有 ADDR_TYPE 类型的地址字段的表 CUST1 了：

```
CREATE TABLE  cust1 (
id            NUMBER,
name          VARCHAR(30),
address       ADDR_TYPE );
```

图 3-5 展示了该表，假定一些记录已经被填写。现在可以使用点符号来访问地址字段的属性了：

```
address.street
address.city
address.state
address.zip
address.country
address.type_of_addr
```

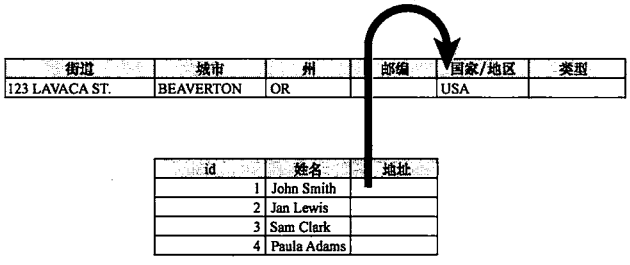


图 3-5 带有声明为对象类型的列的表

到目前为止还没有发现这样做的好处。基本上，只是把多个地址字段存储在了一个列中，不过后面例子中会利用这些新的自定义类型。

www.TopSage.com

2. 集合

Oracle中的集合用嵌套表(对应于SQL:2003的MULTISET)和可变数组(VARRAY,对应于SQL:2003的ARRAY类型)表示。

嵌套表

Oracle中的嵌套表可以定义为一个表(与所有表一样,行数是没有限制的),它是另一个表中的一列。嵌套表的实际数据物理存储在另一个表中,但从用户的角度来看,它就像是主表不可缺少的一部分。

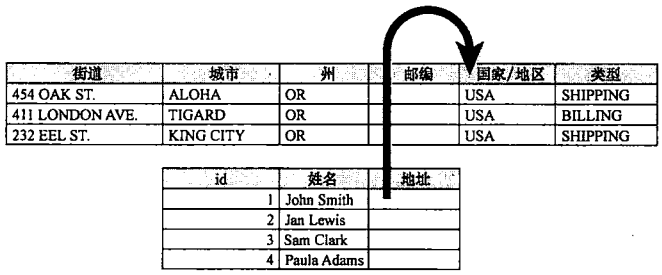
例如,假定表CUST2和ADDRESS之间存在某种关系,现在需要消除这种关系而改用嵌套列。假定已经创建了ADDR_TYPE,接着运行:

```
CREATE TYPE addr_type_tab
AS TABLE OF addr_type
```

这创建了一个新的嵌套表类型的用户定义类型ADDR_TYPE_TAB。现在可以创建实际的表CUST2,以便每一行可以包含一个地址表:

```
CREATE TABLE cust2 (
id        NUMBER,
name      VARCHAR(30),
address   addr_type_tab)
NESTED TABLE address STORE AS addr_storage;
```

代码的最后一行指定了存储表的名称,嵌套表的所有值的行都驻留在存储表中。图3-6显示了CUST2的地址如何存储在嵌套表中。



街道	城市	州	邮编	国家/地区	类型
454 OAK ST.	ALOHA	OR		USA	SHIPPING
411 LONDON AVE.	TIGARD	OR		USA	BILLING
232 EEL ST.	KING CITY	OR		USA	SHIPPING

id	姓名	地址
1	John Smith	
2	Jan Lewis	
3	Sam Clark	
4	Paula Adams	

图3-6 Oracle中的嵌套表

可变数组

VARRAY数据类型与NESTED TABLE非常类似。主要区别在于VARRAY有顺序。另外,必须指定VARRAY的最大元素数,但不必指出存储类型的名称:

```
CREATE TYPE addr_type_varray
AS VARRAY(50) OF addr_type;

create table cust3 (
id        NUMBER,
name      VARCHAR(30),
address   ADDR_TYPE_VARRAY);
```

3.5.3 DB2 9.5

DB2支持用户定义结构类型、引用类型和用户定义的互异类型。

1. 结构类型

结构类型与Oracle的对象类型非常类似。下面这个例子展示了如何创建一个地址结构:

```
CREATE TYPE        addr_type AS (
street             VARCHAR(30),
city               VARCHAR(30),
state              CHAR(2),
zip                VARCHAR(10),
country            VARCHAR(30),
type_of_addr       CHAR(1))
MODE DB2SQL
```

现在可以创建一个表,这个表使用新的用户定义数据类型ADDR_TYPE:

```
CREATE TABLE cust1 (
id        INTEGER,
name      VARCHAR(30),
address   addr_type)
```

大家网 TopSage.com

2. 引用类型

引用类型用于定义对另一个表（某种用户结构类型）中行的引用。引用类似于引用完整性约束，但与引用完整性（需要每个引用值准确标识被引用表中的一行）不同的是，REF 类型要么准确标识被引用表的一行，要么什么也不标识：

```
CREATE TABLE cust2 (
id        INTEGER,
name      VARCHAR(30),
address   REF(addr_type))
```

CUST2 表的地址列引用了以用户定义类型 ADDR_TYPE 来定义的表中的几行。

3. 互异类型

互异类型使用现有内置数据类型定义，并具有一些作为内置类型的功能。例如，可以创建一个新数据类型 usd 来表示美元：

```
CREATE DISTINCT TYPE usd AS DECIMAL(12,2) WITH COMPARISONS
```

现在可以在整个数据库中使用它，就像 Microsoft SQL Server 使用 MONEY 数据类型那样。例如，可以创建具有 USD 类型列的表：

```
CREATE TABLE employee (
id        INTEGER,
same      VARCHAR(30),
salary    USD);
```

3.5.4 Microsoft SQL Server 2008

Microsoft SQL Server 2008 允许创建用户定义数据类型。用户定义数据类型的两种主要类型是**表**类型和**别名**类型。

1. 表类型

表类型的思想和实现与 Oracle 和 DB2 中描述的一样：

```
CREATE TYPE        addr_type AS TABLE (
street             VARCHAR(30),
city               VARCHAR(30),
state              CHAR(2),
zip                VARCHAR(10),
country            VARCHAR(30)
type_of_addr       CHAR(1))
```

现在便可以使用新的用户定义数据类型 ADDR_TYPE 创建表了：

```
CREATE TABLE cust1 (
id        INT,
name      VARCHAR(30),
address   addr_type)
```

2. 别名类型

SQL Server 中的别名类型类似于 DB2 中的互异类型。可以基于系统提供的数据类型创建一个数据类型并指定它自己的规则集：

```
CREATE TYPE usd AS DEC(12,2) NOT NULL
```

3.6 其他数据类型

本章前面几节详细讨论了所有主要的 SQL 数据类型。本节将简要介绍几个数据类型，这些数据类型要么没有被任何大型供应商实现，要么实现的方式中数据类型名称不匹配 SQL:2003 标准。

3.6.1 BOOLEAN

SQL:2003 有一个特殊的 BOOLEAN 数据类型，这个数据类型只包含两个值：TRUE 和 FALSE。Oracle、DB2 和 Microsoft SQL Server 没有 BOOLEAN 数据类型（更确切地说，DB2 和 Oracle 有这个数据类型，但只能在内部使用，或者在程序 SQL 扩展插件中使用；因此，不能把一个列声明为 BOOLEAN 类型）。模拟 BOOLEAN 数据类型很简单，例如，使用 VARCHAR 类型作为用户定义数据类型，它的值只能为 FALSE 和 TRUE。

下面这个例子展示如何在 Microsoft SQL Server 中定义这个数据类型：

www.TopSage.com

```
CREATE RULE bool_rule AS @list in ('TRUE', 'FALSE')
sp_addtype BOOLEAN, 'VARCHAR(5)', 'NULL'
sp_bindrule 'bool_rule', 'BOOLEAN'
```

现在可以将它用作另一个数据类型在 Microsoft SQL Server 实例中使用了。

3.6.2 ROWID

ROWID 是一个特殊的 Oracle 数据类型，用于存储数据库中的每一行的唯一地址。可以为表创建其他显示声明的 ROWID 列，但不建议这么做。

3.6.3 UROWID

UROWID 类似于 ROWID，但用于索引组织表。

3.6.4 BFILE

Oracle 数据类型 BFILE 可以只读访问存储在 Oracle 数据库外部的二进制文件。

3.6.5 DATALINK

DATALINK 是一个 DB2 数据类型，用于管理以外部文件形式表示的大型对象。这些外部文件可以驻留在同一服务器或远程服务器的文件系统中。内部数据库函数用于操作 DATALINK 列。

3.6.6 BIT

Microsoft SQL Server 中的 BIT 数据类型存储一位数据（0 或 1）。一位的文字值是一个字符，它的范围可以包含在单引号中。

3.6.7 TIMESTAMP

Microsoft SQL Server 中的 TIMESTAMP 数据类型与 SQL:2003 的 TIMESTAMP 数据类型是不同的。每个 TIMESTAMP 类型的表只能有一个列。它公开自动生成的二进制数字（在数据库中唯一），主要用于唯一地标识一个数据库行，方式类似于（但并不完全相同）Oracle 的 ROWID，其目的主要是进行版本控制。TIMESTAMP 和 ROWID 之间的主要区别是，TIMESTAMP 列的值会在每次行更改时更新，而 Oracle 的 ROWID 被分配给一个行，只要该行存在。

Microsoft 计划在未来发行版中用 ROWVERSION 数据类型替换 TIMESTAMP 数据类型，以符合 SQL:2003 标准。目前，ROWVERSION 是 TIMESTAMP 的一个同义词。

TIMESTAMP 数据类型的存储大小是 8 个字节，它在语义上等于 binary（8）。

3.6.8 SQL_VARIANT

Microsoft SQL Server 2008 中的 SQL_VARIANT 数据类型使得数据库对象可以支持其他数据类型的值。例如，声明为 SQL_VARIANT 数据类型的表列可以在不同行中包含数字、字符串（CHAR 和 VARCHAR 都被支持，以及 NCHAR 和 NVARCHAR）、日期和二进制值。

不过有一些限制。SQL_VARIANT 不能存储 MAX 数据类型（VARCHAR(MAX)、NVARCHAR(MAX) 或 VARBINARY(MAX)）的值。SQL_VARIANT 不支持 TEXT、NTEXT、IMAGE 和 TIMESTAMP。此外，SQL_VARIANT 不能用另一个 SQL_VARIANT 作为它的基类型。

每个 SQL_VARIANT 列将数据值和它的基数据类型、最大值、小数位数、精度等一起存储。SQL_VARIANT 字段的最大值是 8016 字节，其中 16 个字节用于存储元数据信息，因此实际的值不能大于 8000 字节。SQL_VARIANT 数据类型有它自己的比较规则、运算符优先规则和函数。

3.7 NULL

NULL 是一个特殊的数据库概念，用于描述值不存在的情况。它既不是 0 也不是空字符串，而是一个可用以替代任何数据类型的特殊值。NULL 通常在值未知或无意义时使用。NULL 值可以在以后用

www.TopSage.com

某些实际的数据更新。

例如，当一个新职员被雇用时，他既没有电话号码，也没有被分配到某个部门。在这种情况下，phone 和 department 列就适合使用 NULL 值。

另一种情况是当一个值不合适时，如欧洲的地址中的 STATE 字段。在这种情况下，该字段也可以设置为 NULL。

NULL 值需要特殊处理，而且必须谨慎处理。事实上，任何使用 NULL 作为操作数的运算符也会返回 NULL。

注意：SQL:2003 标准明确说明每个数据类型应该包含一个 NULL 值，它既不等于任何其他值，也不与任何其他值不等，而是表示一个未知的值。NULL 已经被所有主要 RDBMS 供应商实现。

警告：如果使用不当，NULL 会引起严重问题。例如，假设表中有两个列，需要计算它们的差。如果一个列有 NULL 值，则计算结果将就是未定义的（NULL）——例如，100-0=100，但 100-NULL=NULL。必须使用特殊的机制（在第 10 章和第 11 章中讨论）来处理这种以及类似的情况。

3.8 小结

虽然开始不是很明显，但数据类型确实能使工作变得更轻松。本章分类和描述了所有现有数据类型（包括字符和二进制字符串、精确和近似数字、日期和时间、结构和用户定义数据类型，以及其他几个不适合归入上述类别的数据类型），并分析了供应商实现和 SQL:2003 标准之间的差异。

知道在每一个实现中能使用的数据类型，了解这些数据类型与 SQL:2003 标准中定义的数据类型之间的对应关系是非常重要的。

还需要考虑使用某种特殊数据类型来存储某一特殊类型的数据会怎样。

本书中讨论的三大 RDBMS 供应商都拥有自己的专用扩展插件——除了 SQL:2003 标准规定的数据类型。

本章介绍了 NULL 的概念。尽管 NULL 没有定义一个数据类型，但这个概念对理解如何创建、检索和比较数据是至关重要的。

www.TopSage.com

大家网
TopSage.com

PART 2

创建和修改数据库对象

本部分包括：

第4章
创建RDBMS对象

第5章
修改和删除RDBMS对象

www.TopSage.com

www.TopSage.com

第4章
创建RDBMS对象

CHAPTER 4

本章提要：

- 表——主要的RDBMS对象
- 理解约束
- 创建索引以提高性能
- 创建视图
- 创建同义词
- 创建架构
- 创建序列
- 创建域、表空间、字符集、排序规则、目录和其他数据库对象

根据定义，数据库是由对象和管理这些对象的过程组成的集合。在开始使用数据库之前，必须设计数据库，然后创建数据库对象。数据库的设计是一个单独的（但是相关的）SQL主题；我们在附录C中给出了有关RDBMS设计基础的速成课程。本章介绍如何在关系数据库中创建对象。

交叉参考：要创建数据库对象，必须有足够的（特定于实现）安全特权。参阅第12章了解有关SQL安全的详细讨论。

4.1 表

表是任何关系数据库中最核心最重要的对象。任何数据库的主要用途都是存储数据，而这些数据逻辑上存储在表中。

关系数据库的设计原理之一是每一个表都存储有关一个特定事物类型或**实体**的信息。例如，CUSTOMER表将只包含有关客户的数据，而不会包含有关这些客户订购的产品、为客户开出的发票以及为客户下订单的销售人员的数据。在ACME数据库中，CUSTOMER表甚至不包含客户的地址和电话号码，因为这些数据是分别由ADDRESS和PHONE表描述的单独的实体。

行（有时也称为记录）是数据的水平切片；每一行包含有关一个实体项的数据。CUSTOMER表中的一行包含单个客户的信息，ORDER_HEADER中的一行包含单个订单的信息，依此类推。

表数据的垂直切片称为列。一个列存储所有实体记录的某一特定类型的信息。CUSTOMER表中的CUST_NAME_S列包含了所有客户的姓名；ORDER_HEADER中的ORDHDR_INVOICENBR_N包含了所有发票号。

注意：尽管为了简单起见，可以把表想象成按顺序存储在硬盘某处的行和列，但这样一幅图形不能反映出事情的真实面目。首先，表不是顺序的；其次，它们不一定保存在硬盘上（例如，不管提交与否，Oracle都会把所有新的和修改过的行保存在内存中，直到一个特殊的数据库事件出现，这个事件会让内存中的内容与硬盘上对应内容同步）。这是一件数据库管理员需要考虑的事情，无需考虑表的复杂性而将精力集中在表的创建过程中对于用户（甚至程序员）来说，仍然是一项优势。

www.TopSage.com

大家网

Note　注意：SQL:2003 的表定义比 SQL 开发人员和数据库管理员常用的更广泛些。除了与本书中所谓的“表”相对应的基础表之外，SQL 标准还定义了派生表和视图化表，后者大致对应视图（本章稍后将说明）。

CREATE TABLE 语句

尽管我们可以概述数据库表的创建过程，但表的内部实现和 CREATE TABLE 语句的子句因供应商的不同，其差别非常大。例如，Oracle 的 CREATE TABLE 语法图表大约有 15 页长，DB2 大约有 7 页，Microsoft SQL Server 定义最短——只有一页半。

我们将着重介绍 SQL:2003 标准中描述的最常用子句，重点放在供应商的实现差异上。

1. SQL:2003 语法

SQL:2003 提供了以下 CREATE TABLE 语法：

```
CREATE [{GLOBAL | LOCAL} TEMPORARY] TABLE <table_name>
(
 <column name> [<domain_name> |
                <datatype> [<size1>[,<size2>]]
                 [
                   GENERATED [ALWAYS | BY DEFAULT]
                     AS IDENTITY <identity clause>
                 ]
                ]
               [<column_constraint>,...]
               [DEFAULT <default_value>]
               [COLLATE <collation_name>],...
 [<table_constraints>]
) |
LIKE <source_table_name>
          [
           ,<column_name>,...
          ]
[
 [INCLUDING COLUMN DEFAULTS]
 [INCLUDING IDENTITY]
] |
[
 (
   <column_name>,...
 )
]
AS <query_expression>
[ON COMMIT {DELETE | PRESERVE} ROWS]
```

2. Oracle 11g 语法

在 Oracle 11g 中，可以创建三种不同类型的表：关系表、对象表和 XML 类型表。后面两个已经超出了本章的范围。下面给出了关系表的简化 CREATE TABLE 语法：

```
CREATE [GLOBAL TEMPORARY] TABLE [<schema>.]<table_name>
(
 <column name> {[<datatype> [<size1>[,<size2>]]
                  [DEFAULT <default_value>]
                  [ENCRYPT <encryption_specs>]
                  [<column_constraint>,...]
               ] |
               AS <expression> VIRTUAL},...
 [<table_or_column constraint>,...]
 [<physical_properties>]
)
[ON COMMIT {DELETE|PRESERVE} ROWS];
```

New Feature　新增功能：Oracle 11g 语法允许创建由其他列派生的虚拟列，如下面例子所示，其中，column_c 的值为 column_a 加 column_b。

```
CREATE TABLE virtual_column_example
(
  column_a     NUMBER,
  column_b     NUMBER,
  column_c     AS (column_a + column_b) VIRTUAL
)
```

3. DB2 9.5 语法

在 DB2 9.5 中，可以使用这样的语法（省略了一些复杂的子句）来创建表：

www.TopSage.com

```
CREATE TABLE [<schema>.]<table name>
(
  <column_name> <datatype> [<size1>[,<size2>]]
                [<column_constraint>,...]
                [[WITH] DEFAULT [<default_value>] |
                 GENERATED {ALWAYS |
                            BY DEFAULT}
                 AS IDENTITY [<identity_clause>]
                ],...
[<table_constraint>,...] |
[[LIKE <table name> [{INCLUDING | EXCLUDING}
                       {[COLUMN] DEFAULTS | IDENTITY}
                    ]
 ] |
 [AS      <select_statement>
          [{DEFINITION ONLY |
            DATA INITIALLY DEFERRED REFRESH DEFERRED |
            IMMEDIATE}
          ]
 ]
 ]
 [<tablespace_options>]
)
```

4. Microsoft SQL 2008 语法

下面是 Microsoft SQL Server 2008 的语法：

```
CREATE TABLE [[<database name>.]<owner>.][#|##]<table_name>
(
 <column_name> <datatype> [<size1>[,<size2>]]
               [COLLATE <collation_name>]
               [
                [<column_constraint>,...] [DEFAULT <default_value>]
               ] |
               [
                 IDENTITY ([<seed>, <increment>])
                   [NOT FOR REPLICATION]
               ]
[<table_constraint>,...]
 [ON <filegroup>]
 [TEXTIMAGE_ON <filegroup>]
)
```

5. 永久表和临时表

数据库表可以是永久或临时的，这基于表数据的生命周期。

通常，都会希望表是永久的，这意味着插入的数据会一直存在那里，直到有人显式删除表行。在一些较不常见的情况下，可能希望一旦用户**提交**其他表中的更改或注销，这个表的数据就消失。通常，当从其他程序（嵌入式 SQL）发出 SQL 语句，或使用诸如 PL/SQL（Oracle）或 Transact SQL（Microsoft SQL Server）之类的结构化 SQL 扩展来执行复杂的任务时，可能会出现这样的情况。例如，可能希望一个程序从多个不同表中选择列，应用公式对这些列执行计算，把中间结果存储在一个临时表中，然后基于这些结果来更新另一组表。

SQL:2003 中的临时表

SQL:2003 规定了两种类型的临时表：LOCAL 和 GLOBAL。两者不同之处在于它们的数据可见性。即使一个临时表中的数据只在填充它的**会话**（或事务）中可见，GLOBAL 表仍然可以被会话中的任何程序或模块访问；因此，一个存储过程 sp_1 可以创建全局临时表 TEMP_TABLE1；另一个存储过程 sp_2 可以用数据来填充它；然后，其他存储过程 sp_3、sp_4 和 sp_5 可以在它们的处理中使用这些数据，只要所有 5 个存储过程是从同一会话开始的。

与临时表数据不同，临时表的定义是永久的。因此，如果用户 A 创建了一个临时表 B，对其填充数据，然后注销，当该用户第二天（或第二年）重新登录时，这个表依然会存在，只不过它是空的罢了。

Note 注意：会话是从一个用户登录数据库直至他注销这一时间段中的一个或多个事务。事务可以定义为一个逻辑工作单元，由 SQL 语句组成，这些语句通常用于更改数据（更新、插入或删除行）；在事务末尾，所有更改要么使用 COMMIT 语句保存到数据库中，要么被丢弃（回滚）。有关会话、事务以及 COMMIT 和 ROLLBACK 语句的更多内容将在第 7 章讨论。

www.TopSage.com

Oracle 11g 中的临时表

可以在 Oracle 中创建 GLOBAL TEMPORARY 表（LOCA TEMPORARY 表还没有被实现）。GLOBAL TEMPORARY 表的概念与 SQL:2003 概念中的描述稍有不同。表定义对于所有会话都是可见的，但临时表中的数据只对填充数据的会话可见。

```
CREATE GLOBAL TEMPORARY TABLE tmp_customer_order_totals
(
 customer_name    VARCHAR2(30),
 customer_total   NUMBER
)
ON COMMIT DELETE ROWS;
```

换言之，用户 John 可能在一年前创建了 TMP_CUSTOMER_ORDER_TOTALS 表，用户 Mary、Susan 和 Kyle 正在同时使用这个表（假设他们具有适当的特权），但从他们的角度而言，似乎他们中的每一位都在使用自己的临时表（参见图 4-1、图 4-2 和图 4-3）。

CUSTOMER_NAME	CUSTOMER_TOTAL
WILE ELECTROMUSICAL INC.	19824
WILE ELECTRONICS INC.	28672.8

图 4-1　TMP_CUSTOMER_ORDER_TOTALS 临时表中 Mary 的行

CUSTOMER_NAME	CUSTOMER_TOTAL
WILE ELECTROMATIC INC.	30956.2
WILE SEAL CORP.	100771.8

图 4-2　TMP_CUSTOMER_ORDER_TOTALS 临时表中 Susan 的行

CUSTOMER_NAME	CUSTOMER_TOTAL
WILE BESS COMPANY	276775.6

图 4-3　TMP_CUSTOMER_ORDER_TOTALS 临时表中 Kyle 的行

DB2 9.5 中的临时表

在 DB2 9.5 中不能用 CREATE TABLE 创建临时表。可以改用 DECLARE GLOBAL TEMPORARY TABLE 语句来为当前会话声明一个临时表。

声明的临时表不能与其他会话共享；当一个会话结束时，行和表的定义都将被删除。

DECLARE GLOBAL TEMPORARY TABLE 语句的语法类似于 DB2 的 CREATE TABLE 语句。下面是一个例子：

```
DECLARE GLOBAL TEMPORARY TABLE tmp_customer_order_totals
(
 customer_name    VARCHAR(30),
 customer_total   DECIMAL(12,2)
)
ON COMMIT PRESERVE ROWS NOT LOGGED
```

Note　注意：在这里要提及的重要一点是，如果在示例数据库中尝试这个语句，将会失败，并给出一条错误消息，指出用户没有足够大的临时表空间（实际原因是根本没有 USER TEMPORARY TABLESPACE）。我们将在本章后面阐述表空间的概念。

Microsoft SQL Server 2008 中的临时表

Microsoft SQL Server 用于创建临时表的语法与 SQL：2003 标准不一致。要创建一个本地临时表，需要为其加一个前缀符号（#）。（##）表示创建的是全局临时表。

本地临时表仅对当前会话可见。当用户注销时，表数据和表定义都被删除（类似于用 DECLARE GLOBAL TEMPORARY TABLE 语句创建 DB2 临时表）：

```
CREATE TABLE #tmp_customer_order_totals
(
 customer_name VARCHAR(30),
 customer_total MONEY
)
```

Microsoft SQL Server 全局临时表的语义也与 SQL：2003 标准不一致。全局临时表对所有用户都是可见的。在每个查阅这类表的用户断开与 SQL Server 的连接后，它们均被删除。

```
CREATE TABLE ##tmp_customer_order_totals
(
 customer_name   VARCHAR(30),
 customer_total  MONEY
)
```

6. 列定义

表必须有一个或多个列定义，列定义由列名和数据类型组成。

SQL:2003

根据 SQL:2003，域可以代替数据类型用于列（本章稍后将介绍域）。

www.TopSage.com

大学网 TopSage.com

Note 注意：Oracle 11g、DB2 9.5 和 Microsoft SQL Server 2008 在它们的列定义子句中相当一致；唯一的不同是每一个用户实现都使用自己的数据类型。例如，Oracle 列定义如下：

```
customer_name VARCHAR2(30)
customer_total NUMBER
```

在 Microsoft SQL Server 中则替换为：

```
customer_name      VARCHAR(30)
customer_total     MONEY
```

前面提及的供应商中没有一个在列定义中允许域（事实上，他们根本没有域）。

7. 列约束

每一个列可以有一个或多个列约束。SQL:2003 指定了下面的列约束：

NOT NULL 意味着在这个列中不允许 NULL 值。

UNIQUE 意味着这个列中的所有值必须各不相同；允许 NULL 值。

PRIMARY KEY 指定所有列值必须是唯一的，且列不能包含 NULL。换言之，这个约束是上面两个约束的组合。

REFERENCES 意味着这个列是引用表的外键。

CHECK 验证列值服从某些规则；例如，只允许正数，或者只有某一个字符串集是有效的。

所有三个实现都具有所有上述约束，并以类似的方式来处理它们。可以按照数据库命名约定来命名约束，或是不指定约束的名称。对于后面一种情况，RDBMS 将生成默认的名称。下面的例子阐明了列约束（Oracle 或 Microsoft SQL Server）的使用：

```
CREATE TABLE salesman
(
  salesman_id_n          INT            CONSTRAINT pk_salesmanprim PRIMARY KEY,
  salesman_code_s        VARCHAR (2)    CONSTRAINT uk_salescode UNIQUE,
  salesman_name_s        VARCHAR (50)   NOT NULL,
  salesman_status_s      CHAR (1)       CONSTRAINT chk_salesstatus CHECK
                                        (salesman_status_s in ('N', 'Y'))
)

CREATE TABLE address
(
  addr_id_n              INT            CONSTRAINT pk_addrprimary PRIMARY KEY,
  addr_custid_fn         INT,
  addr_salesmanid_fn     INT            CONSTRAINT fk_addr_salesman
                                          REFERENCES salesman (salesman_id_n),
  addr_address_s         VARCHAR(60),
 addr_type_s             VARCHAR(8)     CONSTRAINT chk_addr_type CHECK
                                        (addr_type_s IN ('BILLING', 'SHIPPING')),
  addr_city_s            VARCHAR(18)    CONSTRAINT nn_addr_city NOT NULL,
  addr_state_s           CHAR(2),
  addr_zip_s             VARCHAR(10) NOT NULL,
  addr_country_s         CHAR(3)
)
```

若在 DB2 上运行，则需要对该语句做几处修改：

```
salesman_id_n          INT            CONSTRAINT_pk salesmanprim PRIMARY KEY NOT NULL
salesman_code_s        VARCHAR (2)    CONSTRAINT_uk salescode UNIQUE NOT NULL
addr_id_n              INT            CONSTRAINT_pk addrprimary PRIMARY KEY NOT NULL
```

Note 注意：Oracle 和 Microsoft SQL Server 会在所有的主键和唯一列上隐式创建 NOT NULL 约束，但 DB2 不会这样做，因此，必须在 DB2 表的主键列上同时指定 NOT NULL 约束和 PRIMARY KEY 约束，否则，将会产生一个错误。

8. 列默认值

可以为每一列赋予默认值（在其数据类型范围内）。在本例中，如果一个 INSERT 语句忽略了这个列，则默认值将被自动填入：

```
CREATE TABLE product
(
    prod_id_n          INTEGER            NOT NULL,
    prod_price_n       DECIMAL(10,2),
    prod_num_s         VARCHAR(10),
    prod_description_s VARCHAR(44)        NOT NULL,
    prod_status_s      CHAR(1)            DEFAULT 'Y',
    prod_brand_s       VARCHAR(20)        NOT NULL,
    prod_pltwid_n      DECIMAL(5,2)       NOT NULL,
    prod_pltlen_n      DECIMAL(5,2)       NOT NULL,
```


www.TopSage.com

```
    prod_netwght_n      DECIMAL(10,3),
    prod_shipweight_n   DECIMAL(10,3)
)

INSERT INTO product
(   prod_id_n,
    prod_price_n,
    prod_num_s,
    prod_description_s,
    prod_brand_s,
    prod_pltwid_n,
    prod_pltlen_n,
    prod_netwght_n,
    prod_shipweight_n
)
VALUES
 (990,
  18.24,
  '990',
  'SPRUCE LUMBER 30X40X50',
  'SPRUCE LUMBER',
  4,
  6,
  21.22577,
  24.22577
)

SELECT    prod_id_n,
          prod_price_n,
          prod_status_s
FROM      product

PROD_ID_N PROD PRICE_N PROD_STATUS_S
--------- ------------ -------------
      990         18.24 Y
```

这个例子在“三大”数据库中都有效。

9. 列排序序列

字符串列可以有一个排序序列；因此，可以指定非默认的字符串排序规则。

在“三大”数据库供应商中，只有Microsoft SQL Server允许把排序规则作为列定义的一部分。排序规则名称为字符串数据类型的列指定了排序规则。可以使用Microsoft SQL Server和Microsoft Windows预定义排序规则。例如，如果希望ACME数据库中的所有客户姓名都按照法语排序规则来处理，可以如下修改CUSTOMER表：

```
CREATE TABLE customer
(
    cust_id_n               INT             NOT NULL,
    cust_paytermsid_fn      INT,
    cust_salesmanid_fn      INT,
    cust_status_s           VARCHAR(1)      DEFAULT 'Y' NOT NULL,
    cust_name_s             VARCHAR(50)     COLLATE FRENCH_CI_AI NOT NULL,
    cust_alias_s            VARCHAR(15),
    cust_credhold_s         VARCHAR(1)      DEFAULT 'Y' NOT NULL
)
```

注意：在前面的排序规则定义中，CI代表CASE INSENSITIVE，AI指ACCENT INSENSITIVE。要使用一个区分大小写和区分重音的排序规则，则改用FRENCH_CS_AS。有关可用排序规则的完整列表和描述，请参阅Microsoft SQL Server和Microsoft Windows文档。

交叉参考：Oracle并不实现列定义级别上的排序，而是提供执行区分大小写和区分重音的查询和排序的功能。在第8章中可以了解有关这些功能的更多内容。

10. 表约束

表约束类似于列约束。主要区别是表约束不仅可以用于单独的列，还可以用于列的列表。下面列出了有效的表约束。

UNIQUE：类似于列约束，但能够确保两个或更多列的组合的唯一性。

PRIMARY KEY：约束列中的值的组合必须唯一；不允许NULL值。

FOREIGN KEY：指定表中的一列或一组列，该表引用了被引用表中的一列或一组列。

www.TopSage.com

CHECK：定义一个谓词，该谓词引用一个或多个表中的值；类似于列 CHECK 约束。

下面的例子阐述了表**约束**的使用：

```
CREATE TABLE order_header
(
    ordhdr_id_n                 INTEGER          NOT NULL,
    ordhdr_payterms_fn          INTEGER,
    ordhdr_statusid_fn          INTEGER,
    ordhdr_custid_fn            INTEGER,
    ordhdr_salesmanid_fn        INTEGER,
    ordhdr_nbr_s                VARCHAR(30)      NOT NULL,
    ordhdr_invoicenbr_n         INTEGER,
    ordhdr_orderdate_d          DATETIME,
    ordhdr_invoicedate_d        DATETIME,
    ordhdr_canceldate_d         DATETIME,
    ordhdr_credithold_s         CHAR(1),
    ordhdr_readytoinvoice_s     CHAR(1)          DEFAULT 'N',
    ordhdr_notes_s              VARCHAR(60),
    ordhdr_createdby_s          VARCHAR(10),
    ordhdr_createdate_d         DATETIME,
    CONSTRAINT chk_ordhdr_ready CHECK
            (ordhdr_readytoinvoice_s IN ('N', 'Y')),
    CONSTRAINT chk_ordhdr_credh CHECK
            (ordhdr_credithold_s IN ('N', 'Y')),
    CONSTRAINT pk_ordhdrprim PRIMARY KEY (ordhdr_id_n),
    CONSTRAINT idx_ordhdr ordnbr UNIQUE (ordhdr_nbr_s)
    )

CREATE TABLE shipment
(
    shipment_id_n               INTEGER          NOT NULL,
    shipment_bolnum_s           VARCHAR(6),
    shipment_shipdate_d         DATETIME,
    shipment_arrivdate_d        DATETIME,
    shipment_totalcases_n       INTEGER,
    shipment_trailernbr_s       VARCHAR(12),
    shipment_shpmntfrght_n      DECIMAL(12,2),
    shipment_frtterms_s         VARCHAR(3),
    shipment_createdby_s        VARCHAR(10),
    shipment_createdate_d       DATETIME,
    CONSTRAINT chk_shipfrtterms CHECK
                (shipment_frtterms_s IN ('COL', 'PPD')),
    CONSTRAINT pk_shipmentrprim PRIMARY KEY (shipment_id_n)
    )

CREATE TABLE order_shipment
(
    ordship_ordhdr_id_fn        INTEGER          NOT NULL,
    ordship_shipment_id_fn      INTEGER          NOT NULL,
    CONSTRAINT_pk_ordhdrship    PRIMARY KEY
                (ordship_ordhdr_id fn, ordship_shipment_id_fn),
    CONSTRAINT fk_ordsh_ord FOREIGN KEY (ordship_ordhdr_id_fn)
                REFERENCES order_header(ordhdr_id_n)
)
```

Note 注意：某些约束即可以声明为表约束，也可以声明为列约束，但是其他一些只能在表级别声明为约束——例如，如果约束涉及多列。

这个例子使用的是 Microsoft SQL Server 的语法。要在 Oracle 11g 或 DB2 9.5 上应用表约束，需要把 ORDHDR_ORDERDATE_D、ORDHDR_INVOICEDATE_D、ORDHDR_CANCELDATE_D、ORDHDR_CREATEDATE_D、SHIPMENT_SHIPDATE_D、SHIPMENT_ARRIVDATE_D 和 SHIPMENT_CREATEDATE_D 列的 DATETIME 数据类型修改为特定实现的有效数据类型（DATE 或 TIMESTAMP）。

引用完整性约束

SQL:2003 在创建 FOREIGNKEY 和 REFERENCES 约束时采用了一个可选子句，这个子句指定了当一个行有引用关系，而且该行被从父表删除或被改动时，如何处理该引用表。语法如下：

```
[
  ON {DELETE | UPDATE}
   {NO ACTION | CASCADE | SET NULL | SET DEFAULT | RESTRICT}
]
```

NO ACTION 是默认的行为，意味着如果用户试图删除被其他表引用的行（或更新主键值），会产生一个错误。

CASCADE 表现为当父行或父表上发生更改时，外键也将进行同样的改变。因此，如果父行被删除，

www.TopSage.com

大家网 TopSage.com

则所有引用它的子行也将被删除；或者如果父表的主键被更新，则所有引用它的子行的外键也将用同样的值更新。

SET NULL 意味着如果父行被删除，或者它的主键被更改，那么所有子表的引用外键值都将被设置为 NULL。

SET DEFAULT 与 SET NULL 非常类似，但前者子表的列被设置为默认值而不是 NULL。它假定列的默认值是存在的。

RESTRICT 的行为在多数情况下类似于 NO ACTION——如果子表中存在匹配的记录，则引发一个违反完整性约束的异常。两者区别在于 NO ACTION 检查在 SQL 语句“末尾”进行，因此在 DML 语句执行期间，可以临时违反引用约束，而 RESTRICT 检查是“即时”的，而且且禁止任何违反行为，不过是瞬间的。

所有“三大”供应商都在一定程度上实现了上述 SQL:2003 标准。

Oracle 11g 没有 ON UPDATE 子句。ON DELETE 子句的语法是：

```
[ON DELETE {NO ACTION | CASCADE | SET NULL}]
```

DB2 9.5 的实现除了没有 SET DEFAULT 子句外，与 SQL:2003 标准规范非常接近：

```
[ON DELETE {NO ACTION | RESTRICT | CASCADE | SET NULL}]
[ON UPDATE {NO ACTION | RESTRICT}]
```

Microsoft SQL Server 2008 有 SET DEFAULT 可选项，但是没有 RESTRICT：

```
[ON {DELETE | UPDATE} {NO ACTION | CASCADE | SET NULL | SET DEFAULT}]
```

“三大”实现的默认值都是 NO ACTION。

下面的例子用 Oracle 11g 语法阐明了 NO ACTION 和 CASCADE 之间的区别。

清单 4-1 使用先前创建的 SALESMAN 和 ADDRESS 表，在引用完整性约束 FK_ADDR_SALESMAN 上使用默认选项（NO ACTION）。

程序清单 4-1　默认（NO ACTION）选项

```
INSERT INTO salesman
(
  salesman_id_n,
  salesman_code_s,
  salesman_name_s,
  salesman_status_s
)
VALUES
(
  23,
  '02',
  'FAIRFIELD BUGS ASSOCIATION',
  'Y'
);

1 row created.

INSERT INTO address
(
  addr_id_n,
  addr_custid_fn,
  addr_salesmanid_fn,
  addr_address_s,
  addr_type_s,
  addr_city_s,
  addr_state_s,
  addr_zip_s,
  addr_country_s
)
VALUES
(
  49,
  NULL,
  23,
  '223 E FLAGLER ST.',
  NULL,
  'MIAMI',
  'FL',
  '33131',
  'USA'
);
```


www.TopSage.com

```
1 row created.

DELETE FROM salesman;
DELETE FROM salesman
*
ERROR at line 1:
ORA-02292: integrity constraint (TEMP.FK_ADDR_SALESMAN)
  violated - child record found
```

清单 4-2 在引用完整性约束 FK_ADDR_SALESMAN 上使用 CASCADE 选项重新创建了 SALESMAN 和 ADDRESS 表：

程序清单 4-2 CASCADE 选项

```
DROP TABLE address;

Table dropped.

DROP TABLE salesman;

Table dropped.

CREATE TABLE salesman
(
  salesman_id_n          NUMBER   CONSTRAINT pk_salesmanprim PRIMARY KEY,
  salesman_code_s        VARCHAR2(2)    CONSTRAINT uk_salescode UNIQUE,
  salesman_name_s        VARCHAR2 (50) NOT NULL,
  salesman_status_s      CHAR (1)       CONSTRAINT chk_salesstatus CHECK
                                         (salesman_status_s in ('N', 'Y'))
)

Table created.

CREATE TABLE address
(
  addr_id_n              NUMBER CONSTRAINT pk_addrprimary PRIMARY KEY,
  addr_custid_fn         NUMBER,
  addr_salesmanid_fn     NUMBER CONSTRAINT fk_addr_salesman
                                REFERENCES salesman (salesman_id_n)
                                  ON DELETE CASCADE,
  addr_address_s         VARCHAR2(60),
  addr_type_s            VARCHAR2(8) CONSTRAINT chk_addr_type CHECK
                              (addr_type_s IN ('BILLING', 'SHIPPING')),
  addr_city_s            VARCHAR2(18) CONSTRAINT nn_addr_city  NOT NULL,
  addr_state_s           CHAR(2),
  addr_zip_s             VARCHAR2(10)  NOT NULL,
  addr_country_s         CHAR(3)
);
Table created.

INSERT INTO salesman
(
  salesman_id_n,
  salesman_code_s,
  salesman_name_s,
  salesman_status_s
)
VALUES
(
  23,
  '02',
  'FAIRFIELD BUGS ASSOCIATION',
  'Y'
);

INSERT INTO address
(
  addr_id_n,
  addr_custid_fn,
  addr_salesmanid_fn,
  addr_address_s,
  addr_type_s,
  addr_city_s,
  addr_state_s,
  addr_zip_s,
  addr_country_s
)
VALUES
```

```
(
 49,
 NULL,
 23,
 '223 E FLAGLER ST.',
 NULL,
 'MIAMI',
 'FL',
 '33131',
 'USA'
 );

1 row created.

SELECT * FROM ADDRESS;

ADDR_ID_N      ADDR_CUSTID_FN     ADDR_SALESMANID_FN      ADDR_ADDRESS_S
---------      ---------- ---     ------------ ------     --------------
     49                  23                    223       E FLAGLER ST.
DELETE FROM salesman;

1 row deleted.

SELECT * FROM address;

no rows selected
```

可延迟约束

按照SQL:2003标准，约束可以是DEFERRABLE或NOT DEFERRABLE（默认）。NOT DEFERRABLE约束会在每一个DML语句后检查；DEFERRABLE约束可以在每一个INSERT、DELETE或UPDATE（即时模式）后立即检查，或者在事务末尾检查（延迟模式）。

当没有按特定顺序执行数据加载时，这项功能特别有用——它允许先把数据载入子表，然后再装入父表。另一个用法是在加载不符合某个CHECK约束的数据之后，对其进行适当的更新。

“三大”供应商中只有Oracle 11g遵循可延迟约束规则。其语法是：

```
[[NOT] DEFERRABLE [INITIALLY {IMMEDIATE | DEFERRED}]]
```

或

```
[[INITIALLY {IMMEDIATE | DEFERRED}] [NOT] DEFERRABLE]
```

交叉参考：Microsoft SQL Server允许使用第5章中介绍的ALTER TABLE语法来延迟检查和引用完整性约束。

提示：与任何其他编程语言一样，SQL允许为代码添加注释。如果一行（或其中一部分）SQL代码前以一个破折号（——）为前缀，则这行代码将被视为注释（RDBMS不会试图编译和执行它）。注释文本的另一种方式（通常用于多行注释）是把文本包含在/**/中。

程序清单4-3阐明了可延迟约束的用法。

程序清单4-3 使用可延迟约束

```
-- Create SALESMAN table with NOT DEFERRABLE CHECK constraint
-- chk_salesstatus on salesman_status_s column
DROP TABLE address;
Table dropped.

DROP TABLE salesman;

Table dropped.
CREATE TABLE salesman

(
     salesman_id_n          NUMBER        NOT NULL,
     salesman_code_s        VARCHAR2(2)   NOT NULL,
     salesman_name_s        VARCHAR2(50)  NOT NULL,
     salesman_status_s      CHAR(1)       DEFAULT 'Y',
     CONSTRAINT chk_salesstatus CHECK
               (salesman_status_s in ('N', 'Y')),
     CONSTRAINT pk_salesmanprim PRIMARY KEY (salesman_id_n)
);

Table created.

-- Now, try to insert a row with salesman_status_s = 'A'
INSERT INTO salesman
```


www.TopSage.com

TopSage.com

```
(
   salesman_id_n,
   salesman_code_s,
   salesman_name_s,
   salesman_status_s
)
VALUES
(
   23,
   '02',
   'FAIRFIELD BUGS ASSOCIATION',
   'A'
);

/*
   The result is a constraint violation error
   (constraint is checked immediately).
*/

INSERT INTO SALESMAN
*
ERROR at line 1:
ORA-02290: check constraint (TEST.CHK_SALESSTATUS) violated

-- Drop SALESMAN table and re-create it
-- with DEFERRABLE CHECK constraint
DROP TABLE salesman;

Table dropped.

CREATE TABLE salesman
(
   salesman_id_n         NUMBER        NOT NULL,
   salesman_code_s       VARCHAR2(2)   NOT NULL,
   salesman_name_s       VARCHAR2(50)  NOT NULL,
   salesman_status_s     CHAR(1)       DEFAULT 'Y',
   CONSTRAINT chk_salesstatus CHECK (salesman_status_s in ('N', 'Y'))
                                    DEFERRABLE INITIALLY DEFERRED,
   CONSTRAINT pk_salesmanprim PRIMARY KEY (salesman_id_n)
);

Table created.

CREATE TABLE address
(
   addr_id_n             NUMBER CONSTRAINT pk_addrprimary PRIMARY KEY,
   addr_custid_fn        NUMBER,
   addr_salesmanid_fn    NUMBER CONSTRAINT fk_addr_salesman
                              REFERENCES salesman (salesman_id_n)
                                ON DELETE CASCADE,
   addr_address_s        VARCHAR2(60),
   addr_type_s           VARCHAR2(8) CONSTRAINT chk_addr_type CHECK
                               (addr_type_s IN ('BILLING', 'SHIPPING')),
   addr_city_s           VARCHAR2(18) CONSTRAINT nn_addr_city   NOT NULL,
   addr_state_s          CHAR(2),
   addr_zip_s            VARCHAR2(10)    NOT NULL,
   addr_country_s        CHAR(3)
);

Table created.

-- Try to insert the same row again_works this time
INSERT INTO salesman
(
   salesman_id_n,
   salesman_code_s,
   salesman_name_s,
   salesman_status_s
)
VALUES
(
   23,
   '02',
   'FAIRFIELD BUGS ASSOCIATION',
   'A'
);

1 row created.
```

```
-- Try to commit changes
COMMIT;

-- Error occurs the constraint is checked at the end of the transaction.
COMMIT
*
ERROR at line 1:
ORA-02091: transaction rolled back
ORA-02290: check constraint (TEST.CHK_SALESSTATUS) violated

-- Trying to insert again
INSERT INTO salesman
(
   salesman_id_n,
   salesman_code_s,
   salesman_name_s,
   salesman_status_s
)

VALUES
(
   23,
   '02',
   'FAIRFIELD BUGS ASSOCIATION',
   'A'
);

1 row created.

-- Now update it with an appropriate value ('Y')
UPDATE salesman
SET salesman_status_s = 'Y'
WHERE salesman_status_s = 'A';

1 row updated.

-- COMMIT works this time.
COMMIT;
Commit complete.
```

使用 INITIALLY DEFERRED 约束

INITIALLY DEFERRED 约束状态通常被用于特殊的场合（数据加载、转换等）；当用于日常数据库操作时，延迟约束可能会造成混乱甚至损坏。

例如，想像某人发出了很多 DML 语句，然后试图提交更改，但由于最开始的 INSERT 语句违反了 INITIALLY DEFERRED 约束，提交失败。事务被回滚，因此，用户将不得不重新输入所有的语句。如果在每一个语句之后检查约束，那么用户将会在继续进行任何操作前纠正数据。另一个例子更接近“实际生活”，即时间关键应用程序运行了几个小时，然后由于这样的问题而在提交时崩溃。

解决方案很简单：只针对特殊需求使用 INITIALLY DEFERRED 约束，只要延迟功能不再需要，就将其改为即时模式（如果约束创建为 DEFERRABLE，那么总是可以反复修改它）。第 5 章将学习更多相关知识。

11. ON COMMIT 子句

这个子句只能用于临时表。它指定表行应在事务末尾被隐式删除，还是保留到会话结束，以便后续事务可以使用这个临时表数据。

这个子句可以在 Oracle 的 CREATE TABLE 语句或 DB2 的 DECLARE TABLE 语句中使用。参见前面有关临时表的例子。

12. 物理属性子句

这里只做一个简单介绍，数据一般是物理存储在数据库服务器的硬盘上。精确的定义已经超出了 SQL 著作的范围，但我们将会涉及一些非常基础的知识，以帮助您更好地理解数据库对象的建立。

不同的实现使用的方法具有很大差异，但其思想是一样的：能够根据类型分隔不同的数据库对象，并在理想情况下，把它们放置在不同的物理磁盘上来加快数据库操作。例如，所有的表数据将存储在 Disk1 上，所有的表索引存储在 Disk2 上，而所有的 LOB 将放在 Disk3 上。这种方式的重要性因供应

www.TopSage.com

商而异；许多其他的因素，如数据库的大小、工作负荷、服务器的质量等也发挥着作用。

本书采用的 ACME 样例数据库主要用于教学目的。我们并不期待使用具有多个磁盘的大型服务器，因此，目前讨论的物理存储更多的是理论上的意义。

Oracle 11g

Oracle 使用**表空间**（逻辑数据库结构，本章后面部分将对此进行介绍）。可以为表数据、表索引以及表 LOB 指定不同的表空间：

```
CREATE TABLE phone
(
  phone_id_n            NUMBER        CONSTRAINT pk_phonerimary
                                      PRIMARY KEY USING INDEX
                                      TABLESPACE INDEX01,
  phone_custid_fn       NUMBER,
  phone_salesmanid_fn   NUMBER,
  phone_phonenum_s      VARCHAR2(20),
  phone_type_s          VARCHAR2(20),
  CONSTRAINT chk_phone_type CHECK (phone type_s IN ('PHONE', 'FAX'))
)
TABLESPACE DATA01;
```

这个例子假定表空间 DATA01 存在，表数据将存储在这里，INDEX01 则存储主键的索引。

索引组织表和其他物理属性

指定数据物理上存储在服务器上的什么位置只是在创建数据库表时可以指定的许多选项之一。例如，可以指定多少空间将被保留用于插入行时的更新、控制物理表的增长、基于某些条件对表切片（水平或垂直）、把不同的分区放到不同的物理设备上等。

Oracle 也允许创建索引组织表。索引组织表是一种特殊类型的表，在这个表中，表行保持在基于主键构建的索引中。换言之，不像常规表中的行是无特定顺序存储的，索引组织表中的行总是按主键字段进行排序。因此，访问这样基于主键的表会快得多，但 DML 语句执行速度会明显减慢。例如，要把一行插入到索引组织表中，RDBMS 不得不重新排序并重建整个物理表结构。

下面的语法创建了名为 ORDER_LINE 的索引组织表：

```
CREATE TABLE order_line
(
    ordline_id_n            NUMBER N      OT NULL,
    ordline_ordhdrid_fn     NUMBER        NOT NULL,
    ordline_prodid_fn       NUMBER,
    ordline_ordqty_n        NUMBER,
    ordline_shipqty_n       NUMBER,
    ordline_createdate_d    DATE,
    ordline_createdby_s     VARCHAR2(10),
    CONSTRAINT pk_ordlineprim PRIMARY KEY (ordline_id_n)
)
ORGANIZATION INDEX
;
```

Microsoft SQL Server 用**聚集索引**实现了一个类似的概念，我们将在本章后面部分对聚集索引进行介绍。

DB2 9.5

DB2 也像 Oracle 一样使用表空间。可以为表数据、索引和大型对象指定不同的表空间：

```
CREATE TABLE phone
(
    phone_id_n              INTEGER       NOT NULL,
    phone_custid_fn         INTEGER,
    phone_salesmanid_fn     INTEGER,
    phone_phonenum_s        VARCHAR(20),
    phone_type_s            VARCHAR(20),
    CONSTRAINT chk_phone_type CHECK
            (phone_type_s IN ('PHONE', 'FAX')),
    CONSTRAINT pk_phonerimary PRIMARY KEY (phone_id_n)
)
IN USERDATA01
```

假定一个系统管理的表空间 USERDATA01。我们将在本章末尾关于表空间的小节中介绍如何创建它。

Microsoft SQL Server 2008

Microsoft SQL Server 不使用表空间，而是使用**文件组**。但其思想是非常相似的，差异更多都是语

www.TopSage.com

大家网 TopSage.com

法方面的：

```
CREATE TABLE phone
(
    phone_id_n              INTEGER       NOT NULL,
    phone_custid_fn         INTEGER,
    phone_salesmanid_fn     INTEGER,
    phone_phonenum_s        VARCHAR(20),
    phone_type_s            VARCHAR(20),
    CONSTRAINT chk_phone_type CHECK
              (phone_type_s IN ('PHONE', 'FAX')),
    CONSTRAINT pk_phonerimary
               PRIMARY KEY (phone_id_n) ON INDEX01
)
ON DATA01
```

在这个例子中，我们假定先前创建的文件组 DATA01 和 INDEX01 存在。在本章的后面部分将更详细地讨论 Microsoft SQL 文件组。

13. 标识子句

有时在数据库中，希望生成唯一的连续值，例如，主键列，订单或发票号、客户 ID 等。我们已经在第 3 章中提及了标识列的概念；现在将对其进行更详细的讨论。

SQL:2003

标识列是 SQL:2003 中引入的新功能。在本章开始，我们已经提供了在 CREATE TABLE 定义内创建一个标识列的语法。<identity_clause>语法如下：

```
[
 [START WITH <integer>]
 [INCREMENT BY <integer>]
 [MINVALUE <integer>]
 [MAXVALUE <integer>]
 [CYCLE | NO CYCLE]
]
```

要在创建一个新表时将某列声明为标识列，可以这样做：

```
CREATE TABLE payment terms
(
    payterms_id_n           INTEGER
                             GENERATED ALWAYS AS IDENTITY
                            (START WITH 1
                             INCREMENT BY 1
                             MINVALUE 1
                             MAXVALUE 10000
                             NO CYCLE),
    payterms_code_s         VARCHAR(6),
    payterms_desc_s         VARCHAR(60),
    payterms_discpct_n      DECIMAL(5,2),
    payterms_daystopay_n    INTEGER,
    CONSTRAINT pk_payterms PRIMARY KEY (payterms_id_n)
)
```

Oracle 11g

Oracle 没有标识列，而是改用**序列**这一特殊的数据库对象。可以只创建一个表，它拥有一个要被填充的数字字段。在创建表时，RDBMS 不需要知道这些，例如，按如下方法使用一个序列填充 PAYMENT_TERMS 表的 PAYTERMS_ID_N 字段：

```
CREATE TABLE payment_terms
(
    payterms_id_n           NUMBER   NOT NULL,
    payterms_code_s         VARCHAR(6),
    payterms_desc_s         VARCHAR(60),
    payterms_discpct_n      NUMBER,
    payterms_daystopay_N    NUMBER,
    CONSTRAINT pk_payterms PRIMARY KEY (payterms_id_n)
)
```

Tip　提示：使用序列和触发器的组合可以模拟 Oracle 中的标识列（第 14 章将讨论触发器）。

在本章后面的 CREATE SEQUENCE 部分以及第 5 章的 ALTER SEQUENCE 和 DROP SEQUENCE 部分可以找到有关序列的更多内容。

DB2 9.5

对于一列，可以不使用默认子句而改用标识属性。可以指定起始值、增量、最小值和最大值，当

www.TopSage.com

序列到达最大值时，这个序列是循环还是停止，以及在内存中缓存值的数量：

```
CREATE TABLE payment_terms
(
    payterms_id_n           INTEGER NOT NULL
                            GENERATED ALWAYS AS IDENTITY
                            (START WITH 1,
                             INCREMENT BY 1,
                             CACHE 5),
    payterms_code_s         VARCHAR(6),
    payterms_desc_s         VARCHAR(60),
    payterms_discpct_n      DECIMAL(5,2),
    payterms_daystopay_N    INTEGER,
    CONSTRAINT pk payterms PRIMARY KEY (payterms_id_n)
)
```

默认缓存 20 个值，如果不希望缓存值，可以指定 NO CACHE。

Microsoft SQL Server 2008

标识列创建的方式非常简单，但语法稍有不同：

```
CREATE TABLE payment_terms
(
    -- The first 1 means "start with"; the second stands for "increment by"
    payterms_id_n           INT   NOT NULL IDENTITY (1,1),
    payterms_code_s         VARCHAR(6),
    payterms_desc_s         VARCHAR(60),
    payterms_discpct_n      DECIMAL(5,2),
    payterms_daystopay_n    INT,
    CONSTRAINT pk_payterms PRIMARY KEY (payterms_id_n)
)
```

缓存选项在 Microsoft SQL Server 中不可用。

14. 创建一个新表作为另一个表的副本

有些时候，能够创建一个表作为另一个表的副本是非常有用的。在“三大”数据库中，可以通过创建一个现有的表的精确副本（可以有也可以没有数据）来“克隆”一个现有表，语法稍有不同。

SQL:2003

创建一个新表作为另一个表的精确副本或作为来自多个表的列组合是 SQL:2003 中的另一个新增功能，不过许多 RDBMS 供应商在很早的时候就实现了此功能。除了扩展 CREATE TABLE LIKE 功能来允许复制列默认值并标识源表的列之外，SQL:2003 还引入了 CREATE TABLE AS 子句，允许创建一个具有查询结构的表，包括多表查询以及数据类型、列名等。此外，CREATE TABLE AS 允许重命名原始列。例如：

```
CREATE TABLE customer
(
    cust_id_n               INTEGER       NOT NULL,
    cust_paytermsid_fn      INTEGER,
    cust_salesmanid_fn      INTEGER,
    cust_status_s           VARCHAR(1),
    cust_name_s             VARCHAR(50)   NOT NULL,
    cust_alias_s            VARCHAR(15),
    cust_credhold_s         VARCHAR(1)    NOT NULL DEFAULT 'Y'
)

CREATE TABLE phone
(
    phone_id_n              INTEGER       NOT NULL,
    phone_custid_fn         INTEGER,
    phone_salesmanid_fn     INTEGER,
    phone_phonenum_s        VARCHAR(20),
    phone_type_s            VARCHAR(20)
)

CREATE TABLE customer_phone
(
    customer_name,
    customer_phone,
    credit_hold
)
AS
(
    SELECT    cust_name_s,
              phone_phonenum_s,
              cust_credhold_s
```


www.TopSage.com

```
    FROM        customer,
                phone
 WHERE          cust_id_n = phone_custid_fn
)
WITH NO DATA
```

如果提交了 CREATE TABLE 语句，那么结果如下：

```
CREATE TABLE customer_phone
(
 customer_name      VARCHAR(50)    NOT NULL,
 customer_phone     VARCHAR(20),
 credit_hold        VARCHAR(1)     NOT NULL DEFAULT 'Y'
)
```

Oracle 11g

下面的语句创建了一个临时表，它是 PAYMENT_TERMS 表（包括所有行）的一个副本：

```
CREATE GLOBAL TEMPORARY TABLE payment_terms2
AS
(SELECT * FROM payment_terms);
```

适当地使用 SELECT 语句可以创建一个仅包含目标表中某些列或行的表：

```
CREATE TABLE customer
(
    cust_id_n               NUMBER          NOT NULL,
    cust_paytermsid_fn      NUMBER,
    cust_salesmanid_fn      NUMBER,
    cust_status_s           VARCHAR2(1)     DEFAULT 'Y' NOT NULL,
    cust_name_s             VARCHAR2(50)    NOT NULL,
    cust_alias_s            VARCHAR2(15),
    cust_credhold_s         VARCHAR2(1)     DEFAULT 'Y' NOT NULL,
    CONSTRAINT chk_cust_status CHECK (cust_status_s IN ('N', 'Y')),
    CONSTRAINT chk_cust_credhold CHECK (cust_credhold_s IN ('N', 'Y')),
    CONSTRAINT pk_custprimary PRIMARY KEY (cust_id_n)
);

CREATE TABLE phone
(
    phone_id_n              NUMBER          NOT NULL,
    phone_custid_fn         NUMBER,
    phone_salesmanid_fn     NUMBER,
    phone_phonenum_s        VARCHAR2(20),
    phone_type_s            VARCHAR2(20),
    CONSTRAINT chk_phone_type CHECK (phone_type_s IN ('PHONE', 'FAX')),
    CONSTRAINT pk_phonerimary PRIMARY KEY (phone_id_n)
);

CREATE TABLE customer_phone AS
(
 SELECT         cust_name_s,
                phone_phonenum_s
 FROM           customer,
                phone
 WHERE          cust_id_n = phone_custid_fn
);
```

要在 Oracle 中创建一个空表，可以特别在 WHERE 子句中使用 FALSE 条件：

```
/*
We know that all primary keys in ACME database are positive integers
   so we know that PAYTERMS_ID_N < 0
   condition always evaluates to "FALSE"
*/
CREATE GLOBAL TEMPORARY TABLE payment_terms3
AS
(SELECT * FROM payment_terms WHERE PAYTERMS_ID_N < 0);
```

交叉参考：在附录 K 中可以看到有关 TRUE 和 FALSE 条件的更多内容。

DB2 9.5

下面两条语句都可以在 DB2 中创建一个 PAYMENT_TERMS 表的副本：

```
CREATE TABLE payment_terms2
AS (SELECT * FROM payment_terms)
DEFINITION ONLY

CREATE TABLE payment_terms3
LIKE payment_terms
```

第一种语法（DB2 认为它是汇总表的一种特例——参见侧栏“创建汇总表”）更为灵活，因为它

www.TopSage.com

大家网 TopSage.com

允许基于原始表列的一个子集甚至多表查询来创建表：

```
CREATE TABLE customer
(
    cust_id_n               INTEGER       NOT NULL,
    cust_paytermsid_fn      INTEGER,
    cust_salesmanid_fn      INTEGER,
    cust_status_s           VARCHAR(1)    DEFAULT 'Y'NOT NULL,
    cust_name_s             VARCHAR(50)   NOT NULL,
    cust_alias_s            VARCHAR(15),
    cust_credhold_s         VARCHAR(1)    DEFAULT 'Y' NOT NULL,
    CONSTRAINT chk_cust_status CHECK (cust_status_s IN ('N', 'Y')),
    CONSTRAINT chk_cust_credhold CHECK (cust_credhold_s IN ('N', 'Y')),
    CONSTRAINT pk_custprimary PRIMARY KEY (cust_id_n)
)

CREATE TABLE phone
(
    phone_id_n              INTEGER       NOT NULL,
    phone_custid_fn         INTEGER,
    phone_salesmanid_fn     INTEGER,
    phone_phonenum_s        VARCHAR(20),
    phone_type_s            VARCHAR(20),
    CONSTRAINT chk_phone_type CHECK (phone_type_s IN ('PHONE', 'FAX')),
    CONSTRAINT pk_phonerimary PRIMARY KEY (phone_id_n)
)

CREATE TABLE customer_phone AS
(
 SELECT     cust_name_s,
phone_phonenum_s
 FROM       customer,
            phone
 WHERE      cust_id_n = phone_custid_fn
)
DEFINITION ONLY -- required clause
```

前面的语句创建了一个空 CUSTOMER_PHONE 表，它有两列：CUST_NAME_S 和 PHONE_PHONENUM_S，并具有与基础表中对应列相同的数据类型。可以使用下面的语法获得同样的结果：

```
CREATE TABLE customer_phone AS
(
 cust_name_s        VARCHAR(50)   NOT NULL,
 phone_phonenum_s   VARCHAR(20)
)
```

CREATE TABLE...LIKE 语法的优势在于它可以有选择地创建一个具有所有列默认值或标识列的表的副本。

Tip 提示：在 DB2 中，CREATE TABLE...AS 和 CREATE TABLE...LIKE 语法都可以创建空表。使用 INSERT INTO...SELECT FROM 语句可以填充数据，本章稍后将对此语句进行讨论。

Microsoft SQL Server 2008

在 Microsoft SQL Server 中，可以用下面的语法创建 PAYMENT_TERMS 表的一个副本：

```
SELECT *
INTO #PAYMENT_TERMS2
FROM PAYMENT_TERMS
```

Note 注意：这个语法创建了一个 Microsoft SQL 本地临时表。如果忽略井字符（#），PAYMENT_TERMS2 将创建为一个永久表。

Tip 提示：可以使用与 Oracle 中同样的技巧（一个预先的“FALSE”条件）在 Microsoft Server 中创建一个空表。

创建汇总表

有些 RDBMS 表可以非常大，因此，查询起来会花费相当长的时间。甚至在建立所有相关索引且数据库也正确优化后，一些查询仍然必须执行完整的表扫描，在每一个表中的每一条记录中搜索特定的值以返回期望的结果。当希望用一个查询检索使用聚合函数通过 GROUP BY 子句进行汇总的记录时，尤其如此。

DB2 中引入的汇总表允许在另一个表中维护一个表数据(或来自多个表的数据)的汇总，每当基

www.TopSage.com

础表数据发生更改时，就更新这个汇总表。有关DB2汇总表的最有趣的事情是不仅用户可以查询它们以更快地获得结果，而且RDBMS优化器也可以使用它们来执行用户查询，请求信息已经间接汇总到汇总表中。

请注意，汇总表仅在那些有点静态的表上才会工作得不错。如果源表频繁更改（例如，大量DML语句在其上运行），则系统开销可能变得非常大，总体性能实际上是降低了。

DB2中有两种类型的汇总表。用REFRESH DEFERRED子句创建的表可以使用REFRESH TABLE table_name语句按需更新。如果使用REFRESH IMMEDIATE子句，则每当在其WHERE子句中使用的表发生更改时，汇总表将自动地删除和重新插入行。在这两种情况下，必须在汇总表上至少使用一次REFRESH TABLE...语句，才能开始使用这个汇总表。

汇总表受到许多限制。例如，不能在select_statement中使用ORDER BY、外部联接或不确定性的函数；必须有GROUP BY子句和COUNT（*）列；必须命名所有列等。参阅DB2文档了解所有的规则和限制。

下面是表ORDERLINE_SUMMARY的一个例子，该表会根据需要按序汇总已订购和发货的产品数量：

```
CREATE TABLE order_line
(
    ordline_id_n            INT             NOT NULL,
    ordline_ordhdrid_fn     INT             NOT NULL,
    ordline_prodid_fn       INT,
    ordline_ordqty_n        INT,
    ordline_shipqty_n       INT,
    ordline_createdate_d    DATE,
    ordline_createdby_s     VARCHAR(10),
    CONSTRAINT pk_ordlineprim PRIMARY KEY (ordline_id_n)
)

 CREATE TABLE      orderline_summary
 AS
 (
  SELECT           ordline_ordhdrid fn,
                   SUM(ordline_ordqty n) AS ord_qty_summary,
                   SUM(ordline_shipqty n) AS ship_qty_summary,
                   COUNT (*) AS rowcount
 FROM              order_line
 GROUP BY          ordline_ordhdrid_fn
 )
 DATA INITIALLY DEFERRED REFRESH DEFERRED
```

Oracle通过使用MATERIALIZED VIEWS（在本章后面讨论）提供类似的功能，Microsoft SQL Server则提供一个在视图上创建索引的选项。

4.2 索引

索引是另一种数据库物理结构，它以与表非常类似的方式占用磁盘空间，并被幕后的RDBMS用于改善查询性能。具有足够权限的用户能够创建或删除索引，但是普通SQL表达式和语句永远不会显式引用索引。

例如，假定本章前面创建的PHONE表已经用记录填充，现在提交下面的查询：

```
SELECT *
FROM phone
WHERE phone_custid_fn = 152
```

RDBMS首先检查条件列（PHONE_CUSTID_FN）上是否存在索引。如果存在，那么使用这个索引确定相应行（也就是PHONE_CUSTID_FN=152的两行）的物理位置。如果在该列上没有找到索引，则扫描整个表来寻找具有合适值的行。

数据库索引类似于一本书结尾的索引——它存储指向磁盘上物理行位置的指针，与一本书的索引指向显示相应主题的页的方式相同。从另一个角度而言，它类似于具有两个或更多列的数据库表：一列用于行的物理地址，其余列用于索引表列。换言之，索引告诉RDBMS在磁盘上的什么位置可以找到一个（或一组）特定表行，如图4-4所示。请注意，这个例子只是打一个比方，并不是准确描述索引如何工作。

www.TopSage.com

通过某种程度的简化，我们可以说表列和索引有同样的值集。主要区别在于，在索引中，这些值是被排序了的，因此，找到所需内容的时间会更少。

大多数数据库索引是作为 B 树索引来实现的——也就是说，它们使用 B 树算法，使找到期望记录所需的磁盘访问次数最少，从而加速这个过程。因为磁盘驱动器有机械部分，该部分读写数据比纯粹的电子介质慢得多，所以从硬盘访问数据元素比从内存中访问要慢数千倍。

B 树通过使用具有许多分支（称为子节点）的节点来节省时间。一个 B 树最简单的版本称为二进制树，因为每一个节点都只有两个子节点。图 4-5 举例说明了用一个二进制树来搜索值 100 的过程。这个算法非常简单。从顶端开始，如果顶端的节点值大于正在搜索的值，则移动到左边；如果它小于 100，则转向右边，直到这个值被找到，或者到达一个叶节点而没有找到这个值。

索引可以被创建成唯一的或非唯一的。唯一索引在指定了 PRIMARY KEY 或 UNIQUE 约束的列上隐式创建。不允许重复值。非唯一索引可以在任何列或列组合上创建，不管是否有重复的值。

索引可以在一个或多个列上创建。如果经常在 WHERE 子句中一起使用某些列，则在多个列上创建索引是很有用的。例如，如果某些频繁使用的查询寻找在某一日期某个客户创建的订单，那么可以在 ORDER_HEADER 表的 ORDHDR_CUSTID_FN 和 ORDHDR_CREATEDATE_D 列上创建一个非唯一索引。

SQL:2003 没有为索引指定任何标准（或者甚至根本不需要它们），但实际上所有数据库供应商都提供了用于创建索引的机制，因为如果没有索引，任何产品数据库都将慢得不堪忍受。

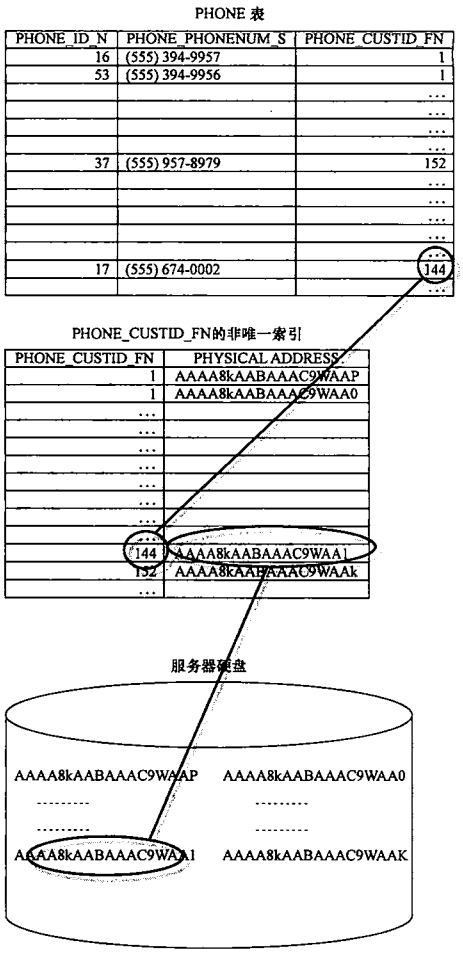


图 4-4 索引的使用

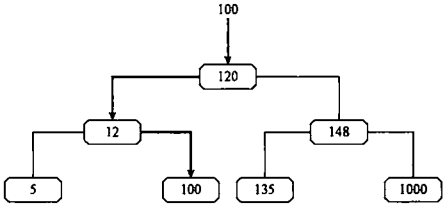


图 4-5 B 树例子

尽管关于何时创建索引没有统一的规则，但可以给出一些一般建议：

- 通常在小表上创建索引没有任何意义——这会降低性能而不是提高它。比如，如果表只有 50 行，那么扫描它会比使用 B 树算法更快。
- 在大表上，只有当涉及索引列的查询检索小百分比的行（通常应小于 15%）时，才应该创建索引。
- 索引通常在用于表联接的列上会有用（默认情况下，主键和唯一列会被索引；索引外键列通常也是一个好主意）。
- 索引会使涉及索引列的 DML 操作减慢——例如，如果更新一个这样列上的值，则这个索引列也必须更新；并且如果插入一行，则相应的索引可能必须重新排序（例如，一个 B 树索引可能必须重新调整）。因此，如果要对一个表进行频繁更新、插入或删除，则推荐使用较少的索引。

CREATE INDEX 语句

对于不同的实现，CREATE INDEX 语句稍有不同。

交叉参考：有关在 XML 列上创建索引的内容将在第 15 章介绍。

www.TopSage.com

大家网 TopSage.com

1. Oracle 11g

下面的代码显示了Oracle 11g的一种稍微简化的CREATE INDEX语法。

```
CREATE [UNIQUE | BITMAP] INDEX [<schema>.]<index_name>
ON [<schema>.]<table_name> ({<column> |
                            <column_expression>
                 }          [ASC | DESC],...)
[<physical_parameters>];
```

这个语句在CUSTOMER表的CUST_NAME_S列上创建一个唯一索引IDX_CUST_NAME，该列值按降序存储：

```
CREATE UNIQUE INDEX idx_cust_name
ON customer(cust_name_s DESC)
```

有时在多列上创建索引是有意义的。例如，如果数据库管理员知道用户经常使用SQL语句通过多项条件来选择记录，例如在WHERE子句中使用cust_name_s、cust_status_s或cust_credhold_s，那么他可以在CUSTOMER表上创建一个多列索引：

```
CREATE INDEX idx_cust_name_status_credhold
ON customer(cust_name_s, cust_status_s, cust_credhold_s)
```

Tip 提示：可以创建涉及同一列的多个索引，如前面示例中，cust_name_s既是idx_cust_name的一部分，也是idx_cust_name_status_credhold的一部分。如果这些行频繁按照升序或降序排列，那么也可以在同一列上创建一个升序索引和一个降序索引。

基于函数的索引

Oracle 11g也允许创建基于函数的索引，其中可以指定一个列表达式（一个确定性的的函数）而不是一列。例如，如果知道用户经常在WHERE子句中使用LOWER函数查询CUSTOMER表，那么可以使用下面的子句：

```
...
WHERE LOWER(cust_name_s) = 'boswell designs corp.'
...
```

在这种情况下，基于函数的索引可能有意义：

```
DROP INDEX idx_cust_name
CREATE UNIQUE INDEX idx_cust_name
ON customer(LOWER(cust_name_s))
```

位图索引

BITMAP关键字表明一个索引将用一个针对每一个不同键的位图来创建，而不是索引每一行。Oracle推荐在那些具有低基数的列上创建位图索引——即诸如GENDER和MARITAL_STATUS之类的只有几个不同值的列。

在ACME数据库中，订单状态可能是一个不错的位图索引候选者，因为一个订单只可能有4种状态（COMPLETE、INVOICED、SHIPPED和CANCELLED）。下面的例子在ORDER_HEADER表的ORDHDR_STATUSID_FN列上创建位图索引IDX_ORDHDR_STATUS：

```
CREATE BITMAP INDEX idx_ordhdr_status
ON order_header (ordhdr_statusid_fn)
```

Note 注意：基于函数的索引和位图索引仅在Oracle企业版中可用，因此，如果试图在个人版或标准版中执行前面的语句，会出现一个错误。

物理存储子句

前面提到，可以为表数据和表索引指定不同的表空间（从数据库性能的角度而言，这样做是有意义的）。其语法与CREATE TABLE的语法一样（下面的例子假定数据库中存在表空间INDEX01）：

```
CREATE UNIQUE INDEX idx_cust_name
ON customer(cust_name_s)
TABLESPACE INDEX01
```

2. DB2 9.5

下面是用于创建索引的简化DB2语法：

```
CREATE [UNIQUE] INDEX [<schema>.]<index_name>
ON [<schema>.]<table_name> (<column_name> [ASC | DESC],...)
[INCLUDE (<column_name>,...)]
```

这个语句在CUSTOMER表的CUST_NAME_S和CUST_ALIAS_S列上创建唯一索引IDX_CUST_NAME_ALS，列值以默认顺序（升序）存储：

```
CREATE UNIQUE INDEX idx_cust_name_als
ON customer(cust_name_s, cust_alias_s)
```


www.TopSage.com

DB2 和 Microsoft SQL Server 语法允许使用 INCLUDE 子句来指定要添加到索引键列集的其他列。其他列不用在索引搜索过程中——换言之，并没有假定这些列是 WHERE 子句的一部分，而是更希望它们是 SELECT 部分的元素。例如，假定知道用户在检索有关客户信息、客户状态和客户信用拥有信息时经常使用 SELECT 语句，并且 WHERE 子句条件是基于客户姓名和客户别名。下面的语法会改善查询性能：

```
CREATE UNIQUE INDEX idx_cust_name_als_plus
ON customer(cust_name_s, cust_alias_s)
INCLUDE (cust_credhold_s, cust_status_s)
```

诀窍就是无论何时找到适当的用户，有关其状态和信用拥有标志就已经存储在索引叶级别了，因此不必使用另外的读取操作来检索数据。

Tip 提示：在 DB2 中创建的索引应该是唯一的，以便使用 INCLUDE 子句。Microsoft SQL Server 没有这个限制。

Note 注意：在 DB2 中，不能为 CREATE INDEX 语句中的索引指定物理位置。一个表的所有索引不是创建在默认表空间中，就是创建在 CREATE TABLE 或 ALTER TABLE INDEX IN 子句指定的表空间中。

3. Microsoft SQL Server 2008

要在 Microsoft SQL Server 中创建一个索引，使用下面的语法：

```
CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED] INDEX <index_name>
[INCLUDE (<column_name>,...)]
ON <table_name> | <view_name> ( column [ ASC | DESC ],...)
[ON filegroup]
```

语法与 DB2 非常类似。可以指定一个文件组在其上创建索引。这个语句在 CUSTOMER 表的 CUST_NAME_ALS 和 CUST_ALIAS_S 列上创建一个唯一索引 IDX_CUST_NAME_ALS，并物理存储在文件组 INDEX01（假定它存在）中。

```
CREATE UNIQUE INDEX idx_cust_name_als
ON customer(cust_name_s, cust_alias_s)
ON INDEX01
```

Tip 提示：在 Microsoft SQL Server 中，可以在视图上创建索引。视图定义必须是明确的并且绑定了架构。创建一个视图索引基本上“物化”了视图，使得它与本章稍后讨论的 Oracle 的 MATERIALIZE VIEW 非常相似。

聚集索引

Microsoft SQL Server 允许通过指定 CLUSTERED 关键字来创建聚集索引（参见前面描述的 CREATE INDEX 语法）。

这个概念非常类似于 Oracle 中的索引组织表——实际表行是带有索引的，而且这些行的物理顺序与它们的索引顺序相同；也就是说，每次插入（或删除）一个新行，就会对这些行重新排序。一个表（或视图）一次只允许有一个聚集索引；因此，可以在一个列上创建一个聚集索引，在以后删除它，并在另一个列上创建一个聚集索引，但不能在同一个表上同时创建两个聚集索引，因为表中的数据在物理上只能按一个顺序进行组织。

下面这些语句创建表 ORDER_LINE，并在它的 ORDLINE_ID_N 列上创建唯一聚集索引：

```
CREATE TABLE order_line
(
    ordline_id_n            INTEGER       NOT NULL,
    ordline_ordhdrid_fn     INTEGER       NOT NULL,
    ordline_prodid_fn       INTEGER,
    ordline_ordqty_n        INTEGER,
    ordline_shipqty_n       INTEGER,
    ordline_createdate_d    DATETIME,
    ordline_createdby_s     VARCHAR(10)
)

CREATE UNIQUE CLUSTERED INDEX idx_ordline_id
ON order_line (ordline_id_n)
```

除了使用 CLUSTERED 自动创建的主键列上的索引外，CREATE INDEX 语句默认是 NONCLUSTERED。

4.3 视图

视图最常描述为一个虚拟表。数据库用户可以从视图中选择行和列；把某个视图与其他视图和表连接；对查询的结果进行限制、排序和分组等。实际上，在大多数情况中下，用户甚至不知道他们是从一个视图还是从一个表中选择值。主要区别在于，试图与表不同，它并不占物理磁盘空间。视图定义作为编译的查询存储在RDBMS中，并动态填充数据以用做用户请求的虚拟表。

具体细节特定于实现——RDBMS能够在后台创建一个临时表，用实际的行填充它，并用它从一个用户的查询返回结果。数据库引擎也可以把用户的查询同一个内部视图定义（也是一个查询）结合起来，并执行得到的查询来返回数据等。从用户角度而言，这无关紧要。

可以通过许多不同方式使用视图。例如，它们可以按用户更加友好的形式从多个表中组合数据，或者使用户利用数据的某种水平或垂直切片来强制安全规则。在本章中，您将学习如何创建不同类型的视图。

4.3.1 CREATE VIEW 语句

本节说明不同RDBMS实现的CREATE VIEW语句。

1. SQL:2003

下面是SQL:2003 CREATE VIEW语句的语法：

```
CREATE VIEW <view_name> [(<column_name>,...)]
AS <query_expression>
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

列名

在大多数情况中，column_name列表是可选的——如果省略它，则基于SELECT语句中的列名来命名视图列。但如果下列条件中至少有一个为真，则column_name列表就不能省略：

- 任意两列可能会具有相同名称（不明确的问题）。
- 任何包含一个计算值（包括连接字符串）的列，并且这个列没有别名。

SELECT语句和可更新的视图

事实上，query_expression可以是任何有效且具有最小限制的SELECT语句。例如，ORDER BY子句不能包含在视图定义中，视图定义不能是循环的；视图不能在它自己的query_expression子句中被引用（直接地或间接地）等。

视图可以是可更新的，也可以是不可更新的。如果一个视图是可更新的，则意味着可以在DML语句中使用它的名称来实际更新、插入和删除基础表的行。只有满足下述所有规则，一个视图就可以是可更新的：

- query_expression不包含任何表联接；也就是说，这个视图是基于一个且只有一个表或视图（对于后者，基础视图也必须是可更新的）。
- 在视图定义中给出了所有基础表的强制（NOT NULL）列。
- 基础查询不包含诸如UNION、EXCEPT或INTERSECT之类的集合运算；也不允许DISTINCT关键字。
- 在query_expression子句中不允许指定聚合函数或表达式。
- 基础查询不能有GROUP BY子句。

Note 注意：设定这些规则的原因是视图的行与物理基础表的行之间必须是一一对应的（但是反过来则不是必须的，因为查询表达式经常筛选基础表中的行），而且视图的列和基础表的列之间也是一一对应的。

视图约束

SQL:2003不允许在视图上创建显式约束，但CHECK OPTION可以被视为某种类型的约束。这个子句只能针对可更新的视图来指定，并且要求在视图上执行的任何DML语句本身除了那些在视图中

www.TopSage.com

大学网 TopSage.com

“可见”的表或视图外，必须没有其他影响。CASCADED 选项（默认的）意味着如果一个视图是基于其他视图的，则基础视图也会被检查。LOCAL 关键字仅在使用这个选项创建的视图级别强制检查。

2. Oracle 11g

Oracle 用下面的语法创建视图：

```
CREATE [OR REPLACE] [FORCE] VIEW [<schema>.]<view_name>
[(<column_name> <column_constrnt>,... [<view_level_constrnt>])]
AS <query_expression>
[WITH {[READ ONLY | CHECK OPTION [<constrnt name>]]};
```

Note 注意：OR REPLACE 子句（经常在创建许多 Oracle 对象——存储过程、函数、包等时使用）主要告诉 RDBMS，如果这个视图已经存在，则删除这个视图，然后用新的语法重新创建它。这是一个非常实用的功能，但必须谨慎使用——如果已经有一个同名的视图，那么 Oracle 将只是用新的对象定义来替换旧的对象定义，而不会给出任何警告。

CREATE OR REPLACE view_name 的伪代码如下：

```
IF EXISTS (view_name) THEN
   DROP VIEW view_name
END IF
CREATE VIEW view_name view_definition
...
```

例如，如果视图 V_CUSTOMER_STATUS 不存在，则创建这个语句；如果存在，则替换旧的 V_CUSTOMER_STATUS 定义：

```
CREATE OR REPLACE VIEW v_customer_status
(
  name,
  status
)
AS
SELECT cust_name_s,
       cust_status_s
FROM customer;
```

视图列如下所示：

```
Name                 Null?    Type
-------------------- -------- ------------
NAME                 NOT NULL VARCHAR2(50)
STATUS               NOT NULL VARCHAR2(1)
```

在后面的情形中，如果省略 OR REPLACE 子句，则会出现一个错误：

```
CREATE VIEW v_customer_status
(
  name,
  status
)
AS
SELECT    cust_name_s,
          cust_status_s
FROM      customer;
ORA-00955: name is already used by an existing object
```

列名/约束子句是可选的：

```
CREATE OR REPLACE VIEW v_customer_status
AS
SELECT    cust_name_s,
          cust_status_s
FROM      customer;
```

注意，在这种情况下，Oracle 会赋予视图列与基础表中相同的列名：

```
Name                     Null?          Type
------------------------ --------       ---------------
CUST_NAME_S              NOT NULL       VARCHAR2(50)
CUST_STATUS_S            NOT NULL       VARCHAR2(1)
```

Note 注意：Oracle 11g 允许在视图列或整个视图上指定完整性约束，但并不强制执行这些约束，因此它们只是声明性的。

WITH READ ONLY 子句会使视图不可更新，即使它满足前面列出的可更新视图的所有条件：

```
CREATE OR REPLACE VIEW v_phone_number
(
  phone_id,
  phone_number
)
AS
```


www.TopSage.com

```
SELECT    phone_id_n,
          phone_phonenum_s
FROM      phone
WHERE     phone_type_s = 'PHONE'
WITH READ ONLY;

UPDATE    v_phone_number
SET       phone_number = NULL
WHERE     phone_id = 1;
ORA-01733: virtual column not allowed here
```

除了CASCADE/LOCAL关键字不可用之外（默认的行为始终是CASCADE），WITH CHECK选项基本上与SQL:2003描述的一样：

```
CREATE OR REPLACE VIEW v_fax_number
(
  fax_id,
  fax_number
)
AS
SELECT    phone_id_n,
          phone_phonenum_s
FROM      phone
WHERE     phone_type_s = 'FAX'
WITH CHECK OPTION;
```

3. DB2 9.5

要在DB2中创建视图，使用下面的语法：

```
CREATE VIEW [<schema>.]<view name>
[(<column_name>,...)]
AS {<query_expression> | <values_statement>}
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

在本书介绍的其他RDBMS供应商不具有的DB2 CREATE VIEW语句中的唯一子句是values_statement，它允许创建一个不引用某个实际表而包含它自己的值列表的视图：

```
CREATE VIEW v_exchange_rate
(
  currency_name,
  exchange_rate,
  converted_price
)
AS VALUES
 ('Canadian Dollars', CAST (0.9552514569 AS DECIMAL(20,10)), 0),
 ('Euro', CAST (1.3802345617 AS DECIMAL(20,10)), 0),
 ('Japanese Yen', CAST (0.0082457044 AS DECIMAL(20,10)), 0)

db2 => SELECT * FROM v exchange_rate
CURRENCY_NAME         EXCHANGE_RATE     CONVERTED_PRICE
----------------      --------------    ---------------
Canadian Dollars      0.9552514569                    0
Euro                  1.3802345617                    0
Japanese Yen          0.0082457044                    0
3 record(s) selected.
```

Note 注意：DB2不执行隐式数据类型转换，因此，在前面的例子中，必须使用函数CAST和INTEGER。有关转换函数的更多内容将在第10章讨论。

前面例子中创建的视图可以按照与使用临时表非常相似的方式来使用——例如，CONVERTED_PRICE列可以基于来自其他表或视图的数据动态填充。

4. Microsoft SQL Server 2008

Microsoft SQL 2008的语法如下：

```
CREATE VIEW [[<database_name>.]<owner>.]<view_name>
[(<column_name>,...)]
[WITH {ENCRYPTION | SCHEMABINDING | VIEW_METADATA,...}]
AS <query_expression>
[WITH CHECK OPTION]
```

Note 注意：下面几个例子假定使用下面的语句创建表PHONE：

```
CREATE TABLE phone
(
    phone_id_n              INTEGER       NOT NULL,
    phone_custid_fn         INTEGER,
    phone_salesmanid_fn     INTEGER,
    phone_phonenum_s        VARCHAR(20),
```


www.TopSage.com

```
    phone_type_s            VARCHAR(20),
    CONSTRAINT chk_phone_type CHECK
               (phone_type_s IN ('PHONE', 'FAX')),
    CONSTRAINT pk_phonerimary
               PRIMARY KEY (phone_id_n)
)
```

Microsoft SQL 使用 CREATE VIEW 语句提供一些附加选项。

WITH ENCRYPTION 子句可以对包含 CREATE VIEW 语句文本的系统表列进行加密。这个功能可以用于隐藏专用代码：

```
CREATE VIEW v_phone_number
(
  phone_id,
  phone_number
)
WITH ENCRYPTION
AS
SELECT    phone_id_n,
phone_phonenum_s
FROM      phone
WHERE     phone_type_s = 'PHONE'
WITH CHECK OPTION
```

WITH SCHEMABINDING 子句可以把视图绑定到架构（有关架构的更多内容将在本章后面介绍）。

WITH VIEW_METADATA 子句指定 SQL Server 将有关视图而不是基础表的信息返回到使用 OLE DB、ODBC 或 DBLIB 的调用程序。

交叉参考：OLE DB、ODBC 和 DBLIB 编程接口（API）将在第 16 章进行更详细的讨论。

在 Microsoft SQL Server 中模拟 OR REPLACE 子句

前面提到，Oracle 的 OR REPLACE 子句是一个非常有用的功能。Microsoft SQL Server 中没有该子句，但使用下面的语法可以很容易地模拟它：

```
IF EXISTS (
 SELECT table_name
 FROM information_schema.views
 WHERE table_name = 'V_CUSTOMER_STATUS'
)
DROP VIEW V_CUSTOMER_STATUS
GO
CREATE VIEW v_customer_status
(
  name,
  status
)
AS
SELECT cust_name_s,
       cust_status_s
FROM   customer
```

本例使用了 Microsoft SQL Server 内置程序语言 Transact SQL，它被 Microsoft SQL Server 开发人员广泛使用，但并不是标准 SQL（按定义是非结构化的）的一部分。

将在第 5 章描述另一个选项使用 ALTER VIEW 语句。

4.3.2 创建复杂视图

我们已经提到几乎可以基于任何 SELECT 语句（具有一些无关紧要的限制）创建视图。SELECT 语句本身是最复杂的 SQL 主题之一，在后面的各章中将会详细讨论。下面几个例子展示了创建一个复杂视图时所使用的主要概念。

1. 将视图与 GROUP BY 子句和聚合函数相结合

V_CUSTOMER_TOTALS 显示了先按 CUSTOMER_NAME 字段再按 ORDER_NUMBER 字段分组计算出的订单价格总计：

```
CREATE VIEW v_customer_totals
(
   customer_name,
   order_number,
   total_price
```

大家网 TopSage.com

```
)
AS
(
 SELECT   customer.cust_name_s,
          order_header.ordhdr_nbr_s,
          sum(product.prod_price_n * order_line.ordline_ordqty_n)
FROM      customer,
          order_header,
          order_line,
          product
 WHERE    customer.cust_id_n = order_header.ordhdr_custid_fn
 AND      order_header.ordhdr_id_n = order_line.ordline_ordhdrid_fn
 AND      product.prod_id_n =order_line.ordline_prodid_fn
 AND      order_line.ordline_ordqty_n IS NOT NULL
 GROUP BY customer.cust_name_s,
          order_header.ordhdr_nbr_s
)
```

2. 基于另一个视图的视图例子

V_CUSTOMER_TOTALS_OVER_15000 视图显示了与它的基础视图 V_CUSTOMER_TOTALS 相同的数据，不过只显示那些总价格超过 1 5000 美元的订单：

```
CREATE VIEW   v_customer_totals_over_15000
AS
SELECT        *
FROM          v_customer totals
WHERE         total_price > 15000
```

3. 使用 UNION 的视图例子

V_CONTACT_LIST 视图显示了包括电话号码和联系类型（客户或销售人员）的客户和销售人员的组合列表：

```
CREATE VIEW v_contact_list
(
 name,
 phone_number,
 contact_type
)
AS
SELECT    cust_name_s,
          phone_phonenum_s,
          'customer'
FROM      customer,
          phone
WHERE     cust_id_n = phone_custid_fn
AND       phone_type_s = 'PHONE'
UNION
SELECT    salesman_name_s,
          phone_phonenum_s,
          'salesperson'
FROM      salesman,
          phone
WHERE     salesman_id_n = phone_salesmanid_fn
AND       phone_type_s = 'PHONE'
```

交叉参考：UNION 是用于联合两个或更多 SQL 查询的集合运算符之一。附录 K 中详细介绍了集合运算符的理论内容；第 7 章则讨论了实用部分。

注意：在 Microsoft SQL Server 中，如果使用本章前面部分提供的语法创建了 CUSTOMER 表（CUST_NAME_S 列的排序规则是 FRENCH_CI_A），那么必须修改 CREATE VIEW 语句以匹配 CUST_NAME_S 和 SALESMAN_NAME_S 列的排序规则。例如，可以修改下面语法：

```
...
SELECT    salesman_name_s,
...
to include collation:
...
SELECT    salesman_name_s COLLATE FRENCH_CI_AI,
...
```

4. 使用子查询的视图

V_WILE_BESS_ORDERS 显示客户 WILE BESS COMPANY 的订单：

```
CREATE VIEW v_wile_bess_orders
```


www.TopSage.com

```
(
 order_number,
 order_date
)
AS
SELECT    ordhdr_nbr_s,
          ordhdr_orderdate_d
FROM      order_header
WHERE     ordhdr_custid_fn IN
   (
    SELECT      cust_id_n
    FROM        customer
    WHERE       cust_name_s = 'WILE BESS COMPANY'
   )
```

前面几个例子均可以在三大 RDBMS 实现中工作。用于创建视图的一些 SELECT 语句将在第 8 章和第 9 章讨论。

4.4 别名和同义词

不同数据库的组织方式是大不相同的。甚至**数据库**这个词语本身在不同的 RDBMS 实现中也具有完全不同的含义。例如，Oracle 数据库是一个完全自包含的独立实体，它自己的用户、表、索引以及其他对象集合对其他数据库是不可见的。每一个 Oracle 数据库用户都可以拥有自己的表、视图、索引等（在 Oracle 的术语中，USER 和 SCHEMA 经常被用做同义词，这引起了混淆）。要访问属于另一个用户（或是位于另一个架构中）的对象，必须被授予适当的权限，并且还必须使用**完全限定名**。例如，如果 USER1 想要在属于 USER3 的 SHIPMENT 表中选择记录，那么查询如下：

```
SELECT *
FROM user3.shipment;
```

如果对于 USER1 而言，SHIPMENT **总是**意味着 USER3.SHIPMENT，则输入完全限定名会使查询变得更长，可读性更差。

图 4-6 展示了 Oracle 的数据库组织结构。

Oracle RDBMS 允许创建同义词来处理这个问题。**同义词**是一个名称，任何时候引用这个名称时，都会把它翻译成另一个名称。换言之，可以说一个同义词是一个数据库对象的替代名称。可以为 USER3.SHIPMENT 创建同义词 SHIPMENT，并在任何需要访问这个 USER3.SHIPMENT 表时使用它。

DB2 的组织结构与 Oracle 非常相似。如果一个数据库对象只是另一个数据库对象的不同名称，则这个数据库对象就被称为 ALIAS。

Microsoft SQL Server 的结构则不同。在单个 SQL Server 中可以有许多数据库。在服务器级别创建用户（或**登录**），并可以访问许多数据库，数据库对象属于一个所有者（通常称为 **dbo**）。参见图 4-7。

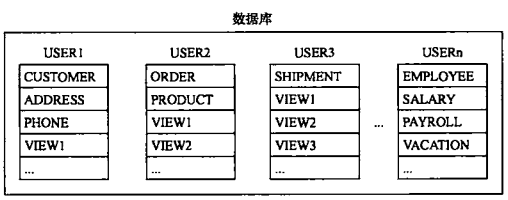


图 4-6 Oracle 中的数据库组织结构

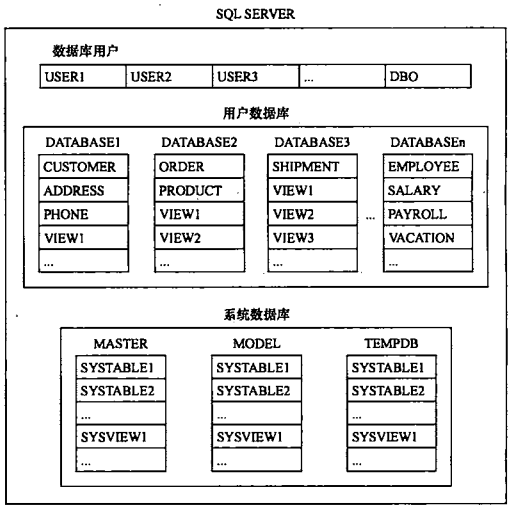


图 4-7 Microsoft SQL Server 中的数据库组织结构

www.TopSage.com

4.4.1 SQL:2003

同义词和别名都不是SQL:2003标准的一部分。

4.4.2 Oracle 11g的CREATE SYNONYM语句

CREATE SYNONYM语句的语法如下所示：

```
CREATE [OR REPLACE] [PUBLIC] SYNONYM [<schema>.]<synonym_name>
FOR [<schema>.]<object_name>[@<dblink>];
```

1. 公共同义词与专用同义词

在Oracle中，可以创建允许所有数据库用户访问的公共同义词，或是仅对它们所有者可见的专用同义词。如果想要创建一个公共同义词，则使用关键字PUBLIC，否则忽略它（关键字PRIVATE是无效的）。

2. 可以创建同义词的对象类型

可以为下列Oracle对象创建同义词：表、视图、序列、存储过程、函数、包、物化视图和Java类架构对象。也可以为另一个同义词创建同义词。

3. 为远程数据库对象创建同义词

可以为位于远程数据库中的对象创建同义词，假定那些数据库有一个**数据库链接**。本章后面部分将介绍有关数据库链接的更多内容。

4. CREATE SYNONYM例子

下面的例子为一个假定的表USER3.SHIPMENT创建公共同义词SHIPMENT：

```
CREATE PUBLIC SYNONYM shipment
FOR user3.shipment;
```

接下来的语句展示为USER2架构中的USERn.EMPLOYEE表创建专用同义词EMP：

```
CREATE SYNONYM user2.emp
FOR usern.employee;
```

如果上面的语句是由USER2用户自己发出的，则可以省略USER2。但是，如果同义词EMP是由数据库管理员为USER2创建的，就不能省略。

注意：尽管同义词非常有用，但它们也会导致许多混乱。最典型的情形是，当一个用户在他自己的架构中拥有一个与公共或专用同义词同名的对象时。Oracle试图首先在用户的架构中查找来解析名称，如果找到这个名称，RDBMS假定这就是要使用的。因此，如果数据库中有视图USER1.VIEW1和USER2.VIEW1，还有用于USER2.VIEW1的公共同义词VIEW1，并希望在所有的用户查询中使用它，则USER1可能会出现严重的程序错误（如果他的VIEW1列与USER2.VIEW1不同）。或者（有时甚至更糟，因为很难注意到这些错误）得到不正确的结果（如果列定义相同，但视图使用不同的基础表或以不同的方式联接它们）。

提示：在Oracle中，可以为不存在的对象创建同义词；如果以后创建这些对象，则可以用这些同义词来引用它们。也可以为对其没有访问权限的对象创建同义词，但这样做并不会得到对那些对象的访问权限。

4.4.3 DB2 9.5的CREATE ALIAS语句

可以为表、视图或其他别名创建别名。语法如下：

```
CREATE ALIAS <alias_name>
FOR <object_name>
```

在DB2中，不能用与当前数据库中已经存在的表、视图或其他别名相同的名称来创建一个别名。这就解决了前面有关Oracle的同义词小节中描述的问题，但使得在DB2中使用别名时缺少了一些灵活性。

下面是在DB2中创建别名的例子：

```
CREATE ALIAS shipmt
FOR user3.shipment

CREATE ALIAS emp
```


www.TopSage.com

```
FOR usern.employee
```

注意，下面的语句会返回一个错误（假定使用如图 4-6 所示的对象），因为架构 USER3 中存在名为 SHIPMENT 的表：

```
CREATE ALIAS SHIPMENT
FOR user3.shipment;
```

同 Oracle 一样，DB2 允许为还不存在的对象创建同义词，但是将给出一条警告。

4.4.4 Microsoft SQL Server 2008 的 CREATE SYNONYM 语句

Microsoft 在它的 2005 版本中引入了同义词。Microsoft SQL Server 2008 提供了下面语法来创建同义词：

```
CREATE SYNONYM [ <schema name1>.] <synonym_name>
FOR [<server_name>.[<database_name>].[<schema_name2>]].<object_name>
```

下面的语法为 SHIPMENT 表创建同义词 MyShipment，该表属于一个假设的远程服务器 AcmeRemote 上虚构 ACMETEST 数据库的 dbo 架构：

```
CREATE SYNONYM MyShipment
FOR AcmeRemote.ACMETSET.dbo.SHIPMENT
```

4.5 架构

架构是一个逻辑上的数据库对象拥有者。SQL:2003 把架构定义为一组已命名的相关对象。当对象有**循环**引用时——即需要创建两个表时，每一个表都有一个外键引用另一个表时，创建架构就会非常有用。不同的实现处理架构的方式略有不同。

CREATE SCHEMA 语句

CREATE SCHEMA 语句在 SQL:2003，Oracle，DB2 以及 Microsoft SQL Server 中具有不同的含义。

1. SQL:2003

SQL:2003 指定 CREATE SCHEMA 语句创建一组对象，这些对象以某种方式在逻辑上彼此相关——这个组有一个被作为**架构名**引用的名称。也可以在 CREATE SCHEMA 语句中授予访问这些对象的特权。语法如下：

```
CREATE SCHEMA
  {<schema_name> |
   AUTHORIZATION <authorization_id> |
   <schema_name> AUTHORIZATION <authorization_id>
}
[DEFAULT CHARACTER SET <character_set>]
[<schema_path_specification>]
[<create_object_statement> | <grant_privilege_statement>,...]
```

架构创建者通常拥有架构中的对象，除非另外指定一个不同的 authorization_id。

可以为新架构指定一个默认的字符集，区别于数据库默认字符集。也可以指定一个架构集。

对象可以创建为架构的一部分，包括表、视图、域、断言、字符集、排序规则、翻译、触发器、转换、架构例程、序列和用户定义对象。可以在 grant_privilege_statement 子句中授予任何有效的特权和角色。

注意：创建域、断言、字符集、排序规则和转换在三大数据库中都不是有效的操作，尽管它们都是 SQL:2003 标准的组成部分。

交叉参考：CRANT PRIVILEGE 语句在第 12 章讨论。

2. Oracle 11g

我们已经谈到，在 Oracle 术语中，“架构”这个词几乎与“用户”相同。在 Oracle 中仍然可以使用 CREATE SCHEMA 语句，但这个操作的唯一用途是在单个事务中创建多个对象。

注意：在 Oracle 中，CREATE SCHEMA 语句并没有实际创建一个架构。架构是在创建用户时自动创建的。

www.TopSage.com

CREATE SCHEMA的语法如下：

```
CREATE SCHEMA AUTHORIZATION <schema=your_user_name>
<create_object_statement>,...
<grant_privilege_statement>,...
;
```

架构名必须与Oracle用户ID一样；否则，这个语句将失败。可以创建表和视图对象，并且可以授予任意用户对这些对象的访问特权。下面的例子在虚拟架构ACMETEST中（假设通过用户ACMETEST运行该语句，ACME测试数据库中不存在该用户）创建了两个表，并将两个表的访问权限授予用户ACME：

```
CREATE SCHEMA AUTHORIZATION ACMETEST
CREATE TABLE address
(
    addr_id_n           INT CONSTRAINT pk_addrprimary PRIMARY KEY,
    addr_custid_fn      INT,
    addr_salesmanid_fn  INT CONSTRAINT fk_addr_salesman
                            REFERENCES salesman (salesman_id_n)
                            ON DELETE CASCADE,
    addr_address_s      VARCHAR2(60),
    addr_type_s         VARCHAR2(8) CONSTRAINT chk_addr_type
                                    CHECK
                                    (addr_type_s IN ('BILLING', 'SHIPPING')),
    addr_city_s         VARCHAR2(18) CONSTRAINT nn_addr_city    NOT NULL,
    addr_state_s        CHAR(2),
    addr_zip_s          VARCHAR2(10)       NOT NULL,
    addr_country_s      CHAR(3)
 )
CREATE TABLE salesman
(
  salesman_id_n         INT CONSTRAINT pk_salesmanprim PRIMARY KEY,
  salesman_code_s       VARCHAR2 (2) CONSTRAINT uk_salescode UNIQUE,
  salesman_name_s       VARCHAR2 (50) NOT NULL,
  salesman_status_s     CHAR (1)   CONSTRAINT chk_salesstatus
                                   CHECK
                                   (salesman_status_s in ('N', 'Y'))
)
GRANT ALL ON salesman TO ACME
GRANT ALL ON address TO ACME
;
```

Note 注意：如果没有把这两个表作为一个事务的一部分来创建，那么第一条CREATE TABLE语句将失败，因为它引用了不存在的SALESMAN表（约束FK_ADDR_SALESMAN）。

如果CREATE SCHEMA中的任何一条语句失败，那么所有其他语句也将被忽略。

3. DB2 9.5

DB2的CREATE SCHEMA语句看起来更接近SQL99标准。语法如下：

```
CREATE SCHEMA {<schema_name> |
               AUTHORIZATION <authorization_id> |
               <schema_name> AUTHORIZATION <authorization_id>
}
<create_object_statement>,...
<grant_privilege_statement>,...
```

在create_object_statement子句中创建的有效对象有表、视图和索引。也可以使用COMMENT语句在各种对象的类别描述上创建注释。架构所有者要么是authorization_id，要么（如果没有指定）是提交这个CREATE SCHEMA语句的用户：

```
CREATE SCHEMA ACMETEST AUTHORIZATION ACMETEST
CREATE TABLE address
(
    addr_id_n           INT NOT NULL CONSTRAINT
                                     pk_addrprimary PRIMARY KEY,
    addr_custid_fn      INT,
    addr_salesmanid_fn  INT CONSTRAINT fk_addr_salesman
                                REFERENCES salesman (salesman_id_n)
                                ON DELETE CASCADE,
    addr_address_s          VARCHAR(60),
    addr_type_s             VARCHAR(8) CONSTRAINT chk_addr_type
                            CHECK
                            (addr_type_s_IN ('BILLING', 'SHIPPING')),
    addr_city_s             VARCHAR(18) CONSTRAINT nn_addr_city NOT NULL,
    addr_state_s            CHAR(2),
    addr_zip_s              VARCHAR(10) NOT NULL,
    addr_country_s          CHAR(3)
)
```


www.TopSage.com

```
CREATE TABLE salesman
(
  salesman_id_n              INT NOT NULL CONSTRAINT
                                  pk_salesmanprim PRIMARY KEY,
  salesman_code_s            VARCHAR (2) NOT NULL CONSTRAINT uk_salescode UNIQUE,
  salesman_name_s            VARCHAR (50) NOT NULL,
  salesman_status_s          CHAR (1) CONSTRAINT chk_salesstatus
                                  CHECK
                                  (salesman_status_s in ('N', 'Y'))
)
GRANT ALL ON salesman TO ACME
GRANT ALL ON address TO ACME
```

4. Microsoft SQL Server 2008

Microsoft SQL Server 创建架构的语法如下：

```
CREATE SCHEMA {<schema_name> |
               AUTHORIZATION <owner>
               <schema_name> AUTHORIZATION <owner>}
{<create_object_statement> |
 <revoke_privilege_statement> |
 <deny_statement> |
 <grant_privilege_statement>,...}
```

所有者必须在数据库中拥有一个有效的安全账户。下面的语句假定账户 ACMETEST 存在：

```
CREATE SCHEMA TEST AUTHORIZATION ACMETEST
CREATE TABLE address
(
    addr_id_n                INT CONSTRAINT pk_addrprimary PRIMARY KEY,
    addr_custid_fn           INT,
    addr_salesmanid_fn       INT CONSTRAINT fk_addr_salesman
                                 REFERENCES salesman (salesman_id_n)
                                 ON DELETE CASCADE,
    addr_address_s           VARCHAR(60),
    addr_type_s              VARCHAR(8) CONSTRAINT chk_addr_type
                                 CHECK
                                 (addr_type_s IN ('BILLING', 'SHIPPING')),
    addr_city_s              VARCHAR(18) CONSTRAINT nn_addr_city NOT NULL,
    addr_state_s             CHAR(2),
    addr_zip_s               VARCHAR(10) NOT NULL,
    addr_country_s           CHAR(3)
)
CREATE TABLE salesman
(
  salesman_id_n              INT CONSTRAINT pk_salesmanprim PRIMARY KEY,
  salesman_code_s            VARCHAR (2) not null CONSTRAINT uk_salescode UNIQUE,
  salesman_name_s            VARCHAR (50) NOT NULL,
  salesman_status_s          CHAR (1) CONSTRAINT chk_salesstatus
                                  CHECK
                                  (salesman_status_s in ('N', 'Y'))
)
GRANT ALL ON salesman TO ACME
GRANT ALL ON ADDRESS TO ACME
```

4.6 序列

序列是一个数据库对象，它具有与标识（在本章前面讨论过）类似的功能。主要区别在于标识绑定到一个表列，而序列完全独立于其他数据库对象；因此，多个用户可以从一个序列生成唯一的数字值，并将它们用于不同的目的。最典型的用法是为一个表（或多个表）生成主键值。

Note

注意：可以使用一个序列来为两个或更多的表生成主键。如果采用无意义主键的概念，那么，实际上不必关心从 1 到 10 以外的整数，数字 1、2 和 5 将用于填充 TABLE1 的主键；3、7 和 10 将用于 TABLE2；4、6 和 8 将成为 TABLE3 的主键值；而 9 根本就没有使用（参见图 4-8）。还有一种更典型的方法是为每一个表的主键创建一个单独的序列。

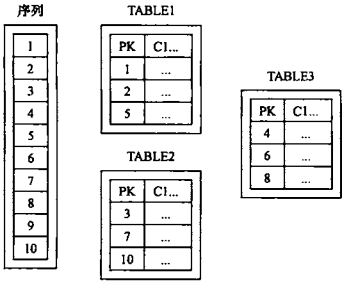


图 4-8 使用序列生成的数字来填充多个表中的主键

通常，序列比标识列更灵活，因为它是一个独立的数据库对象，

www.TopSage.com

大家网 TopSage.com

而标识列是表定义的一部分。例如，当用于为父/子表填充记录的结构化程序（C、Java、COBOL 等）时，使用序列比标识列更方便。

4.6.1　SQL:2003 中的外部序列生成器

SQL:2003 标准既包括内部又包括外部序列生成器。内部序列生成器都是匿名的并且都是作为其他数据库对象的组成部分隐式创建的。例如，前面讨论的标识列都是与内部序列生成器相关联的。

外部序列生成器的作用方式非常相似。主要区别在于它们都是使用 CREATE SEQUENCE 语句创建的命名数据库对象（SQL:2003 无需可选子句按任何特定顺序排列）：

```
CREATE SEQUENCE <sequence_name>
[AS <data_type>]
[START WITH <start_value>]
[INCREMENT BY <increment_value>]
[MAXVALUE <max_value> | NOMAXVALUE]
[MINVALUE <min_value> | NOMINVALUE]
[CYCLE | NOCYCLE]
```

这个序列有一个当前基值，在创建序列时初始化为<start_value>的值，并且每次使用 NEXT VALUE FOR<sequence_name>语法调用一个外部序列生成器的下一个值时，按照<increment_value>的值增加或减少该基值。

现在我们来讨论供应商特定的序列生成器实现。

4.6.2　Oracle 11g 中的序列

在 Oracle 中创建序列的语法如下：

```
CREATE SEQUENCE [<schema>.]<sequence_name>
[START WITH <start_value>]
[INCREMENT BY <increment_value>]
[MAXVALUE <max_value> | NOMAXVALUE]
[MINVALUE <min_value> | NOMINVALUE]
[CYCLE | NOCYCLE ]
[CACHE <value> | NOCACHE]
[ORDER | NOORDER]
```

您可能注意到，CREATE SEQUENCE 语句中的大多数子句都是可选的。如果只是想要创建这样一个序列，从 1 开始，并生成连续的值（2、3、4，等等），直到达到 Oracle 中可能的最大整数（10^{27}）为止，那么这个语句将是：

```
CREATE SEQUENCE my_sequence1;
```

如果需要更做复杂的事情，下面小节描述了还可以做什么。

1. 创建升序和降序序列

可以创建一个从某个值开始的序列（START WITH 子句），然后按照 INCREMENT BY 子句中指定的值生成下一个序列值。要创建一个降序序列，使用一个负增量即可。还必须指定最大和最小序列值：

```
CREATE SEQUENCE my_sequence2
START WITH 500
INCREMENT BY -10
MAXVALUE 500
MINVALUE 0;
```

2. 创建循环序列

在前面的例子中，my_sequence2 将生成值 500、490、480、470，…0。之后，将得到一个错误，说明序列小于 MINVALUE，且不能被实例化。可以创建一个循环序列，也就是说，在序列达到它的最大（在降序序列中是最小）值后，将从头开始。在下一个序列例子中，在 my_sequence3 达到它的最小值后，将从 1000 重新开始：

```
CREATE SEQUENCE my_sequence3
START WITH 1000
INCREMENT BY -10
MAXVALUE 1000
MINVALUE 0
CYCLE;
```

通常很难理解 START WITH 和 MINVALUE（或用于降序序列的 MAXVALUE）子句之间的差别。实际上，这个差别仅对循环序列是重要的。例如，可能希望序列从 100 开始，然后当它达到最大值（比如 10000）时，再从头开始，但这次不是从 100 开始，而是从 10 开始。在这种情况下，为 START WITH

www.TopSage.com

子句指定 100，而为 MINVALUE 子句指定 10：

```
CREATE SEQUENCE my_sequence4
START WITH 100
INCREMENT BY 1
MINVALUE 10
MAXVALUE 10000
CYCLE;
```

3. 缓存序列值

默认情况下，为了更快访问，Oracle 会在内存中缓存 20 个连续的序列值。可以通过指定一个不同的缓存值的数字（10 三 000、1000 或任何喜欢的数字），或者通过使用 NOCACHE 子句来覆盖这个默认行为，NOCACHE 子句可以确保每次使用这个序列来生成值时得到连续值。否则，例如，如果数据库重启，则内存中的值将被清除。

提示：对于循环的序列而言，缓存的值的数目必须小于循环中的值的数目。

4. 确保序列值的顺序

可以通过指定一个 ORDER 子句来确保按顺序生成序列数（例如，如果将它们用作时间戳）。默认是 NOORDER。事实上，这个选项与本书的目的无关。

注意：Oracle 没有标识列，因此序列是生成连续数字的唯一途径。

5. 在 Oracle 中访问序列

可以通过在 SQL 代码中使用 SEQUENCE_NAME.NEXTVAL 来生成新的序列值。要访问当前的序列值（即最后生成的序列数），可以使用 SEQUENCE_NAME.CURRVAL：

```
SELECT my_sequence4.NEXTVAL
FROM dual;
   NEXTVAL
----------
       102
SELECT my_sequence4.CURRVAL
FROM dual;
   CURRVAL
----------
       102
SELECT my_sequence4.CURRVAL
FROM dual;
   CURRVAL
----------
       102
SELECT my_sequence4.NEXTVAL
FROM dual;
   NEXTVAL
----------
       103
```

交叉参考：前面的例子使用了一个哑表 DUAL，在 Oracle 中用于从“nothing”中选择值。参阅第 10 章了解更多内容。

注意：要能够访问 CURRVAL，必须在当前会话中至少生成这个序列一次，否则，将产生一个错误。在某种意义上，DB2 的 PREVVAL 名更准确——实际上是在访问前面生成的值。

4.6.3 DB2 9.5

在 DB2 中创建一个序列的语法如下：

```
CREATE SEQUENCE <sequence_name> [AS SMALLINT |
                                    INTEGER |
                                    BIGINT |
                                    DECIMAL
                                ]
[START WITH <start_value>]
[INCREMENT BY <increment_value>]
[MAXVALUE <max_value> | NOMAXVALUE]
[MINVALUE <min_value> | NOMINVALUE]
[CYCLE | NOCYCLE]
[CACHE <value> | NOCACHE]
```


www.TopSage.com

```
[ORDER | NOORDER]
```

可以看到，它几乎与 Oracle 的语法完全一样，因此前面部分介绍的例子都可以在 DB2 中运行。这里有两点小区别：

- 可以为序列值指定想要的数据类型，只要它具有准确的精度。默认是 INTEGER。如果指定 DECIMAL 数据类型，则小数位数必须是 0。Oracle 总是假设为 NUMBER 数据类型。
- 在 DB2 中，可以创建静态序列，即总是填充同样的值（很难想象我们为什么会需要这样的序列），通过指定 INCREMENT BY 0 或指定 MINVALUE 和 MAXVALUE 使用同样的值即可。Oracle 要求 INCREMENT TO 是一个正的或负的整数——不允许为 0。MAXVALUE 必须大于 MINVALUE。

在 DB2 中访问序列

使用 NEXTVAL 或 PREVVAL 关键字可以检索当前或先前的序列值。NEXTVAL 与 Oracle 中的 NEXTVAL 相同；PREVVAL 相当于 Oracle 的 CURRVAL。对序列值的访问稍有不同：

```
SELECT NEXTVAL FOR my_sequence4 AS NEXTVAL
FROM SYSIBM.SYSDUMMY1
NEXTVAL
-----------
        102

  1 record(s) selected.

SELECT PREVVAL FOR my_sequence4 AS PREVVAL
FROM SYSIBM.SYSDUMMY1
PREVVAL
-----------
        102

  1 record(s) selected.

SELECT PREVVAL FOR my_sequence4 AS PREVVAL
FROM SYSIBM.SYSDUMMY1
PREVVAL
-----------
        102

  1 record(s) selected.
```

交叉参考：DB2 中的 SYSIBM.SYSDUMMY1 表等价于 Oracle 的 DUAL。参阅第 10 章了解更详细的信息。

注意：Microsoft SQL Server 2008 没有序列，但是在 Transact SQL 中利用程序很容易模拟序列。

4.7 其他 SQL:2003 以及特定于实现的对象

至此，您可能认识到 SQL 标准更多的存在于理论而非现实生活中。SQL:2003 定义的许多对象还没有被三大数据库中的任何一个实现。而每一个供应商都有他自己独特的数据库对象类型集，并且在其他的 RDBMS 实现中没有使用，或者在 SQL:2003 标准中没有描述。不过，所有主要供应商都在积极参与 SQL 标准委员会工作，并且主要致力于使他们的产品尽可能地符合 SQL 标准。另一方面，SQL 标准吸收了各种 RDBMS 供应商实现的最佳功能而不断发展。这个过程是动态的并且似乎不会停止。

4.7.1 域（SQL:2003）

域是一个数据库对象，当定义表列时，域可以用作一个数据类型的替代品。除数据类型外，它还可以选择指定一个默认值、一个排序规则和一套约束。语法如下：

```
CREATE DOMAIN <domain_name> [AS] <datatype>
[DEFAULT <default_value>]
[<constraint_definition>,...]
[COLLATE <collation_name>]
```

我们前面提到，Oracle、DB2 或 Microsoft SQL Server 都没有实现域，不过它们都有一些类似的功能来实现相似的目的（例如，Oracle 中的 CREATE TYPE，DB2 中的 CREATE DISTINCT TYPE，Microsoft

www.TopSage.com

SQL Server 中的 CREATE RULE 等。如第 3 章中的例子所示）。

Note 注意：Sybase、PostgreSQL、InterBase、Borland 以及其他一些 RDBMS 供应商实现了域。

4.7.2 字符集（SQL:2003）

CREATE CHARACTER SET 命令创建一个字符集。SQL:2003 不允许应用程序创建一个新的字符集，只能复制现有字符集并指定一个不同的名称。

语法如下：

```
CREATE CHARACTER SET <character_set_name> [AS]
GET <character_set_specification> [<collate_clause>]
```

4.7.3 排序规则（SQL:2003）

CREATE COLLATION 命令定义一个排序规则。SQL:2003 不允许应用程序创建一个新的排序规则，只能复制现有排序规则并指定一个不同的名称和更改它的填充选项

```
CREATE COLLATION <collation_name>
FOR <character_set_specification>
FROM <existing_collation_name>
[NO PAD | PAD SPACE]
```

4.7.4 表空间和文件组

在讨论创建表和索引的过程时，我们已经提到过表空间。表空间的概念不是很直观——它是一个针对**物理**存储的**逻辑**结构。换言之，表空间是当想要指定一个数据库对象的物理位置时会在 SQL 代码中引用的内容。表空间可以由一个或多个**数据文件**组成，数据文件是特殊的系统文件，表、索引或任何其他物理数据以二进制形式保存在这些数据文件中。一个数据文件只能属于一个表空间。图 4-9 显示了表空间和数据文件之间的关系。

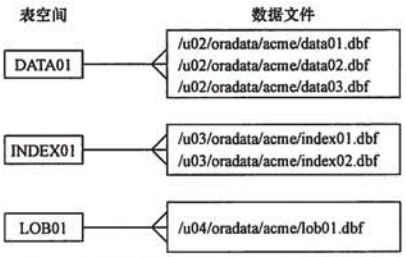


图 4-9 表空间和数据文件之间的关系（一对多）

比如，当用户在 DATA01 表空间中创建表时，他们不能控制这个表的行将在物理上驻留于哪个数据文件中：data01、data02、data03，或者遍布所有三个数据文件。RDBMS 将自己管理这些事情（事实上，在大多数情况下，用户甚至不需要指定表空间；在数据库管理员分配给对象的默认表空间中创建对象。）

Note 注意：数据文件是二进制文件；不能手动编辑它们或打开它们来查看表数据。只有 RDBMS "知道" 它们的内部专用结构，并且可以使用这些文件。

1. Oracle 11g

在 Oracle 中，使用 CREATE TABLESPACE 命令创建表空间。简化的语法如下：

```
CREATE [BIGFILE | SMALLFILE] TABLESPACE <tablespace_name>
DATAFILE  <file_path_and_name> SIZE <size>[K|M|G|T|P|E]
            [REUSE]
            [AUTOEXTEND ON | OFF]
          [, <file_path_and_name> SIZE <size>[K|M|G|T|P|E]
            [REUSE]
            [AUTOEXTEND ON | OFF],..]
[EXTENT MANAGEMENT {LOCAL | DICTIONARY}]
[SEGMENT SPACE MANAGEMENT {AUTO | MANUAL}]
[FLASHBACK {ON | OFF}]
```

SIZE 是一个与字母 K（千字节）、M（兆字节）、G（千兆字节）、T（万亿字节）、P（十亿兆字节）和 E（千兆兆字节）关联的整数。其他子句主要指定表空间管理自动化的程度。

Note 注意：在 Oracle 中创建表空间不是一项简单的任务，通常是由数据库管理员处理。表空间具有许多可选的子句，这些子句已经超出了本书的范围。

下面的例子创建表空间 DATA01，它具有一个大小为 1MB 的数据文件（假定 Windows 上安装了标准的 Oracle 11g）：

www.TopSage.com

```
CREATE TABLESPACE DATA01
DATAFILE 'C:\oracle\ora92\oradata\acme\data01.dbf' SIZE 1M;
```

Oracle在目录C:\oracle\ora92\oradata\acme下创建以Oracle块格式化的文件data01.dbf。除非指定REUSE子句，否则如果这个文件已经存在，将得到一个错误。现在可以尝试本章开始的例子，在表空间DATA01中创建表PHONE。

2. DB2 9.5

DB2中的CREATE TABLESPACE语句也相当复杂，而且通常只由DBA使用。基本的语法如下：

```
CREATE [REGULAR | LARGE] TABLESPACE <tablespace_name>
MANAGED BY [SYSTEM | DATABASE | AUTOMATIC STORAGE]
USING ([FILE | DEVICE] <file_or_device_name> [<size> K|M|G], ...)
[<storage_clause>]
```

REGULAR子句是默认的。只能创建存储LOB对象的LARGE表空间。

注意：系统管理的表空间比数据库管理的表空间需要更小的维护量。而数据库管理的表空间为有经验的DBA提供了更多的灵活性。当创建一个系统管理的表空间时，不能使用FILE | DEVICE或指定大小。

下面的代码在目录C:\DB2\DATA01下（假定为默认安装）创建常规的系统管理表空间USERDATA03：

```
CREATE TABLESPACE USERDATA03
MANAGED BY SYSTEM
USING ('C:\DB2\DATA01')
```

3. Microsoft SQL Serve 2008

在Microsoft SQL Server中没有表空间这样的概念，但是可以创建文件组，并把文件添加到文件组，这与在Oracle和DB2中使用表空间是完全相同的概念。只是术语不同和语法稍有不同——实际上使用ALTER DATABASE语句把文件组和/或文件添加到一个数据库中。

语法如下：

```
ALTER DATABASE
ADD FILEGROUP <filegroup_name>

ALTER DATABASE
ADD FILE
(
  NAME = <logical_file_name>
 [, FILENAME = <os_file_name>]
 [, SIZE = <size> KB|MB|GB|TB]
 [, <other_physical_parameters>]), ...
 [ TO FILEGROUP <filegroup_name> ]
```

可以看到，其概念与您已经了解的没有太大区别：Microsoft SQL Server文件组是一个逻辑结构，就像Oracle和DB2中的表空间，文件就是一个物理存储单元。

下面的代码创建文件组DATA01，然后将一个大小为1MB、逻辑名为DATA0101的文件添加到文件组：

```
ALTER DATABASE acme
ADD FILEGROUP DATA01
GO
ALTER DATABASE acme
ADD FILE
(
 NAME = DATA0101,
 FILENAME ='C:\Program Files\Microsoft SQL Server\MSSQL\Data\data0101.ndf',
 SIZE = 1MB
)
TO FILEGROUP DATA01
```

4.7.5 物化视图（Oracle 11g）

MATERIALIZED VIEW是另一个包含数据并占用物理空间的Oracle对象。其名称有点让人迷惑——我们已经知道视图只是一个编译过的查询，而这里的物化视图更像表，它们有实际包含数据的行，可以动态更新。最接近的类比是DB2中的汇总表。除了数据聚合外，物化视图还可以按许多不同的方式使用，包括数据仓库、数据复制等。除此之外，物化视图比DB2的汇总表限制更少——大多数与CREATE VIEW配合使用的选择语句可以用于创建物化视图。

www.TopSage.com

交叉参考：Microsoft SQL Server 有一个在视图上创建索引的选项，在概念上它类似于 Oracle 中的物化视图概念。

CREATE MATERIALIZED VIEW 语法相当复杂——大多数子句都需要特殊的安全特权，因而只能由数据库管理员来完成，而大多数用户都没有这样的安全特权。例如，CREATE MA-TERIALIZED VIEW 特权允许用户创建物化视图，QUERY REWRITE 允许用户创建优化器在查询重写中使用的物化视图等。

提示：QUERY REWRITE 把一个由表或视图表述的 SQL 语句转换为一个访问在这些详细表中定义的一个或多个物化视图的语句。如果恰当使用，这个功能会显著改善数据库的性能。

下面的列子说明了物化视图几种可能的用法。

1. 在请求时更新物化视图

物化视图 VRM_ORDERLINE_SUMMARY 类似于我们在本章前面谈到的 DB2 的 ORDERLINE_SUMMARY 汇总表。一经请求，它则按订单汇总已订购和已发货的产品数量：

```
CREATE MATERIALIZED VIEW vrm_orderline_summary
BUILD IMMEDIATE
AS
(
 SELECT   ordline_ordhdrid_fn,
          SUM(ordline_ordqty_n) AS ord_qty_summary,
          SUM(ordline_shipqty_n) AS ship_qty_summary,
          COUNT (*) AS rowcount
FROM      order_line
GROUP BY ordline_ordhdrid_fn
);
```

2. 没有用户交互的定期更新的物化视图

物化视图 VRM_CONTACT_LIST 使用与 V_CONTACT_LIST 视图相同的 query_expression。在物化视图创建时进行记录快照，并在每日上午 6 点更新，不需要 DBA、应用程序人员或其他人进行任何显式的操作：

```
CREATE MATERIALIZED VIEW test2
REFRESH START WITH SYSDATE
NEXT (TRUNC(SYSDATE + 1)) + 6/24
AS SELECT     cust_name_s,
              phone_phonenum_s,
              'CUSTOMER' CONTACT
FROM          customer,
              phone
WHERE         cust_id_n = phone_custid_fn
AND           phone_type_s = 'PHONE'
UNION
SELECT        salesman_name_s,
              phone_phonenum_s,
              'SALESPERSON'
FROM          salesman,
              phone
WHERE         salesman_id_n = phone_salesmanid_fn
AND           phone_type_s = 'PHONE';
```

交叉参考：前面代码中的 NEXT（TRUNC（SYSDATE+1））+6/24 语句在 SYSDATE 上加了一天，截取结果得到“12AM tomorrow”，再加 6 小时，得到第二天的上午 6 点。第 10 章将介绍有关 TRUNCATE 函数和日期算法的更多信息。

这个物化视图可以用于常规视图不适合的情形——例如，当基础表太大而数据获取变得太慢时，或者当 query_expression 中使用的表物理存在于一个远程数据库中，并且必须通过网络传送时。物化视图可以在卸载期间更新数据。

注意：前面的例子假定数据库管理员使用下面的语句授予用户 CREATE MATERIALIZED VIEW 特权：

```
GRANT CREATE [ANY] MATERIALIZED VIEW TO <user_name>;
```

第 12 章将介绍有关特权的更多内容。

4.7.6 数据库链接（Oracle 11g）

数据库链接是本地数据库中的一个对象，它使用户能够访问远程数据库对象。远程数据库不必是

www.TopSage.com

Oracle 数据库——可以直接连接到 DB2、Microsoft SQL Server、Sybase 或者任何其他支持 ODBC 和/或 OLE DB 协议的 RDBMS，假定正确配置了 Oracle 网络。可以在 SQL 语句中通过把@<dblink>附加到表或视图名上来引用一个远程表或视图（这里，<dblink>是前面创建的一个数据库链接的名称）。

CREATE DATABASE LINK 的语法如下：

```
CREATE [[SHARED] PUBLIC] DATABASE LINK <dblink>
CONNECT TO {[CURRENT_USER |
              <user_name> IDENTIFIED BY <password>]
           }
USING '<connect_string>';
```

SHARED 关键字指定一个单一的网络连接可以由多个用户使用。PUBLIC 意味着这个数据库链接可以用于所有用户，而不仅是它的创建者。如果远程数据库的用户名和密码与使用这个数据库链接的人的用户名和密码相同，则可以指定 CURRENT_USER；否则必须指定一个远程用户名和密码。

connect_string 参数必须是一个合法的服务名（参见 Oracle 的《**Net Services Administrator's Guide**》来设置服务名）。必须使用单引号包含它。

下面的命令假定有一个有效的服务名 SALES 并指向一个存在用户 SALES_MAN_AGER 的数据库的，创建名为 DBL_SALES 的数据库链接：

```
CREATE DATABASE LINK dbl_sales
CONNECT TO sales_manager IDENTIFIED BY sales123
USING 'SALES';
```

注意：DB2 和 Microsoft SQL Server 都允许连接到远程数据库。DB2 使用 NICKNAMES，Microsoft SQL Server 则提供链接服务器和远程服务器的功能。查看相关供应商的技术文档来了解详细信息。

4.7.7 目录（Oracle 11g）

目录对象在 Oracle 中指定到一个操作系统目录的别名，Oracle 可以读写驻留在这里的外部文件。

交叉参考：在第 14 章中可以找到有关访问外部 OS 文件的更多内容。

创建目录的语法如下：

```
CREATE [OR REPLACE] DIRECTORY <directory_name>
AS <os_full_path>
```

例如：

```
CREATE DIRECTORY acme
AS 'C:\sqlbib\oracle_acme';
```

创建一个目录对象后，可以将它的 READ 和 WRITE 访问权限授予不同的用户：

```
GRANT READ ON DIRECTORY ACME TO SCOTT;
GRANT WRITE ON DIRECTORY ACME TO SCOTT;
```

交叉参考：在第 12 章了解授予数据库对象访问特权的内容。

4.8 CREATE 语句交叉参考

表 4-1 提供了在“三大”数据库和 SQL:2003 的 CREATE 语句标准中可以创建的所有对象的完整列表。本章讨论了其中一些语句；其他一些将在本书后面介绍。例如，我们将在第 12 章讨论 CREATE USER、CREATE ROLE 和 CREATE PROFILE 语句。在第 14 章介绍 CREATE PROCEDURE、CREATE FUNCTION、CREATE PACKAGE 和 CREATE TRIGGER 语法。

表 4-1 有效的 CREATE 语句交叉参考

语句	SQL:2003	Oracle 11g	DB2 9.5	Microsoft SQL Server 2008
CREATE ALIAS		√		
CREATE ASSEMBLY			√	
CREATE ASSERTION	√			
CREATE BUFFERPOOL			√	
CREATE CAST	√			

www.TopSage.com

续表

语句	SQL:2003	Oracle 11g	DB2 9.5	Microsoft SQL Server 2008
CREATE CHARACTER SET	√			
CREATE CLUSTER		√		
CREATE COLLATION	√			
CREATE CONTEXT		√		
CREATE CONTROLFILE		√		
CREATE DATABASE		√	√	√
CREATE DATABASE PARTITION GROUP				
CREATE DATABASE LINK		√		
CREATE DEFAULT		√		
CREATE DIMENSION		√		
CREATE DIRECTORY		√		
CREATE DISKGROUP		√		
CREATE DISTINCT TYPE			√	
CREATE DOMAIN	√			
CREATE EVENT MONITOR			√	
CREATE EVENT NOTIFICATION				√
CREATE FOREIGN TABLE	√			
CREATE FUNCTION		√	√	√
CREATE FUNCTION MAPPING			√	
CREATE INDEX		√	√	√
CREATE INDEX EXTENSION			√	
CREATE INDEXTYPE		√		
CREATE JAVA		√		
CREATE LIBRARY		√		
CREATE MATERIALIZED VIEW		√		
CREATE MATERIALIZED VIEW LOG		√		
CREATE METHOD			√	
CREATE MODULE	√			
CREATE NICKNAME			√	
CREATE OPERATOR		√		
CREATE ORDERING	√			
CREATE OUTLINE		√		
CREATE PACKAGE		√	√	
CREATE PACKAGE BODY		√		
CREATE PFILE		√		
CREATE PROCEDURE		√	√	√
CREATE PROFILE		√		
CREATE QUEUE				√
CREATE ROLE	√	√		
CREATE ROLLBACK SEGMENT		√		
CREATE ROUTINE MAPPING	√			
CREATE RULE				√
CREATE SCHEMA	√	√	√	√
CREATE SECURITY LABEL COMPONENT			√	
CREATE SECURITY LABEL			√	
CREATE SECURITY POLICY			√	
CREATE SEQUENCE	√	√	√	
CREATE SERVER			√	
CREATE SPFILE		√		
CREATE STATISTICS				
CREATE SYNONYM		√		√
CREATE TABLE	√	√	√	√
CREATE TABLESPACE		√	√	
CREATE TABLE HIERARCHY			√	

www.TopSage.com

续表

语句	SQL:2003	Oracle 11g	DB2 9.5	Microsoft SQL Server 2008
CREATE TEMPORARY TABLESPACE		√		
CREATE TRANSFORM	√		√	
CREATE TRANSLATION	√			
CREATE TRIGGER	√	√	√	√
CREATE TYPE	√	√	√	
CREATE TYPE BODY		√		
CREATE TYPE MAPPING			√	
CREATE USER	√	√		√
CREATE USER MAPPING			√	
CREATE VIEW	√	√	√	√
CREATE WRAPPER			√	

许多 CREATE 语句只由数据库管理员使用，因此，本书没有详细阐述它们；参考特定于供应商的文档获得更详细的信息。

4.9　小结

数据库的一种定义是“对象及其操作过程的一个集合”。在关系数据库中，对象可以由 SQL:2003 标准或者供应商特定的 SQL 创建。CREATE<object>语法可以非常复杂，并且通常属于数据库管理员的领域；尽管如此，理解如何创建数据库对象还是很重要的。

本章介绍的最重要的对象包括表、索引、视图、物化视图、同义词、架构和序列等。这里并没有列出在 RDBMS 中使用 SQL 能够创建的所有对象。

核心且最重要的数据库对象是**表**——以数据库文件结构形式实现的一个数据存储的逻辑概念。根据表在 RDBMS 中的生命周期，表可以是临时的或永久的。这一基本概念因供应商的不同，其实现也不同。差异大多出现在临时表中使用的 CREATE 语法上，不过，在永久表中使用的语法更接近 SQL 标准。

索引不是 SQL 标准规范的一部分，并且对用户是隐藏的。索引的目的是加速从一个表中获取数据。

视图不同于表，当请求时，视图的数据来源于它的基表（物化视图除外）。使用 SQL SELECT 查询可以定义视图，视图依赖于这个查询所指定的表。本章介绍了许多使用视图的理由。

同义词用易于使用的名称替换长的完全限定名，从而方便了数据库对象的引用。

序列用于生成唯一的数字值，这些数字值可用于不同的用途。序列的主要用法是生成无意义的主键。

本章提到的一些其他数据库对象是**字符集**、**排序规则**、**域**、**表空间**、**文件组**和**约束**。

每一个供应商在其 RDBMS 软件中实现了数据库对象的一个不同子集——其中一些对象并不是标准 SQL 的一部分，而且一些更高级别的标准对象没有实现。用于创建这些对象的语法在这些供应商之间也是不同的，应该对照供应商的文档进行查看。

www.TopSage.com

大家网
TopSage.com

第5章
修改和删除 RDBMS 对象

CHAPTER 5

本章提要：

- 修改和删除表
- 修改和删除索引
- 修改和删除视图
- 修改和删除其他数据库对象

在一个数据库中创建的对象并非是一成不变的。轻松更改数据库对象的功能使 RDBMS 具有超越非关系模型的强大优势，因为关系数据库的概念隐含了灵活性。这包括创建、修改和删除数据库对象的能力。“三大”供应商都提供了这种能力，而且利用供应商扩展插件通常可以修改一个实体而不会影响其他实体。

Note 注意：本章中的例子假定是在第 4 章开始的 ACME 数据库基础上的进一步扩展。本书配套的网站上下载的数据库脚本中实现的一些约束可能会妨碍某些语句的成功执行。

5.1 表

我们已经知道，表是核心的 RDBMS 对象。在理想世界中，表的结构一旦创建就不能更改。但在现实生活中，经常需要修改表的定义。商业规则的改变通常是最重要的原因；最初的设计错误也并不少见，尤其是在测试环境中。ALTER TABLE 语句的思想就是能够尽可能快速且无影响地改变表的定义。DROP TABLE 语句用于从数据库中删除表（包括它的数据和定义）。

5.1.1 ALTER TABLE 语句

本节介绍不同供应商如何实现 ALTER TABLE。

1. SQL:2003

SQL:2003 的 ALTER TABLE 语法如下：

```
ALTER TABLE <table_name>
{[ADD [COLUMN] <column_definition>] |
 [ALTER [COLUMN] <column_name> {
                      SET DEFAULT <default> |
                      DROP DEFAULT |
                      ADD <scope_clause> |
                      DROP SCOPE <drop_ behavior> |
                      <alter_identity_column_option>
                     }
] |
 [DROP [COLUMN] <column_name> RESTRICT | CASCADE] |
 [ADD <table_constraint>] |
 [DROP CONSTRAINT <constraint_name> RESTRICT | CASCADE]
};
```

基本上，这段代码定义了添加或删除表列或列的默认值，

www.TopSage.com

以及删除表约束的语法。此外，SQL:2003 允许修改标识列选项。语法与创建一个标识列相同：

```
[
 [START WITH <integer>]
 [INCREMENT BY <integer>]
 [MINVALUE <integer>]
 [MAXVALUE <integer>]
 [CYCLE | NO CYCLE]
]
```

Note 注意：与范围相关的 ALTER TABLE 选项只适用于 REF 表。

2. Oracle 11g

在 Oracle 中，ALTER TABLE 语句甚至比 CREATE TABLE 具有更多的选项。下面的语法仅列出了本书涉及的一些最重要选项：

```
ALTER TABLE [<qualifier>.]<table_name>
{[<change_physical_attributes>] |
 [ADD <column_name> <datatype> [<size1>[,<size2>]]
    [DEFAULT <default_value>] [<column_constraint>,...]
 ],... |
 [MODIFY <column_name> <datatype> [<size1>[,<size2>]] |
    [DEFAULT <default_value>] [NULL | NOT NULL]
 ],... |
 [DROP {COLUMN <column_name> | (<column_name>,...)}
    [CASCADE CONSTRAINTS]
 ] |
 [MODIFY CONSTRAINT <constraint_name> <constraint_state>] |
 [ADD {PRIMARY KEY (<column_name>,...) |
       UNIQUE (<column_name>,...) |
       CONSTRAINT <constraint_name> <constraint_definition>
      }
 ] |
 [DROP {PRIMARY KEY |
        UNIQUE (<column_name>,...) |
        CONSTRAINT <contraint_name> [CASCADE]
       }
 ] |
 [DISABLE | ENABLE {PRIMARY KEY |
                    UNIQUE (<column_name>,...) |
                    CONSTRAINT <constraint name> |
                    ALL TRIGGERS |
                    TABLE LOCK
                   }
 ] |
 [RENAME TO <new_table_name>] |
 [MOVE [ONLINE] TABLESPACE <tablespace_name>]
}
```

更改物理属性

在 Oracle 中，可以使用 ALTER TABLE 语句更改表的许多不同物理属性。例如，如果在错误的表空间中创建了表，那么很容易修正。下面的例子假定表空间 DATA01 存在：

```
ALTER TABLE SALESMAN
MOVE TABLESPACE DATA01;
```

Note 注意：在 ACME 样例数据库中，所有对象都在没有明确指定表空间的情况下创建。同样，用户 ACME 也在没有指定默认表空间的情况下创建。这意味着所有的表和索引在默认情况下都在 SYSTEM 表空间中创建（除非数据库最初使用 DEFAULT TABLESPACE 选项创建），在生产数据库中，这是绝对不能接受的。现在可以使用 ALTER TABLE 语句来纠正它。

参阅 Oracle 文档了解有关更改表物理属性的更多内容。

添加列

经常需要在现有表中添加列。例如，ACME 数据库能轻松处理一个客户拥有多个电话号码的情况。现在，假设有一套新的商业规则，它规定如果一个客户或一名销售人员有多部电话，则需要一种方式来知道哪个号码是主要的以及它是哪种类型的电话（公司、手机、住宅等）。因此，需要向 PHONE 添加两个新列：PHONE_PRIMARY_S（默认值 Y）和 PHONE_CATEGORY_S。还需要为 PHONE_PRIMARY_S 指定合法值的范围（Y 和 N）。这个任务实现起来相当简单：

www.TopSage.com

```
ALTER TABLE PHONE
ADD PHONE_PRIMARY_S CHAR(1) DEFAULT 'Y'
    CHECK (PHONE_PRIMARY_S IN ('Y', 'N'))
ADD PHONE_CATEGORY_S CHAR(15);
```

从这个例子中可以看到，Oracle 允许在使用一条 ALTER TABLE 语句的同时添加多个列。也可以在一条 ALTER TABLE 语句中删除多个列。

Tip 提示：如果把一个具有默认值的列添加到现有表中，那么这个新列中的每一行都将自动更新为指定的 DEFAULT 值。

修改现有列

可以更改列的数据类型，增加或减少字符或二进制字符串的大小或数字列的精度。也可以为现有列指定新的默认值，或者为其添加 NOT NULL 约束。

通常情况下会应用某些限制条件。只有当列为空时，也就是说，这个列包含的都是 NULL，才可以减少它的大小或降低精度（不过，可以没有限制地增加大小和提高精度）。也不能把 NOT NULL 约束添加到包含 NULL 的列（不过可以通过其他方式来实现）。如果一个列不为空，则只能把这个列的数据类型更改为兼容的数据类型（例如，CHAR 可以改为 VARCHAR）；对于空列则没有限制。

Note 注意：只有 NOT NULL 约束可以使用 ALTER TABLE 语句的 MODIFY 子句添加到一个表列。所有其他约束可以用 ADD、DROP、DISABLE、ENABLE 和 MODIFY CONSTRAINT 子句来添加、删除、修改和禁用。

下面给出一些例子。在上一例子中，添加到 PHONE 表的 PHONE_CATEGORY_S 列是 CHAR（15）。不过，在确切知道将在列中存储多少字符时，最好使用 CHAR 数据类型，否则，VARCHAR2 就是更好的选择。同时，我们决定把最大的列大小增加到 20 个字符，因为可能需要在这里存储更大的字符串；但是目前要在所有新行的这个列中填充默认的字符串 BUSINESS，而且不希望 PHONE_PRIMARY_S 列中有任何 null 值。

下面的语句成功实现了这个任务：

```
ALTER TABLE PHONE
MODIFY PHONE_PRIMARY_S CHAR(1) NOT NULL
MODIFY PHONE_CATEGORY_S VARCHAR2(20) DEFAULT 'BUSINESS';
```

Note 注意：这个语句是可行的，因为当添加 PHONE_PRIMARY_S 列时，还指定了默认值 Y，因此，它不会包含任何 null 值。与 ALTER TABLE...ADD 不同，上面的 ALTER TABLE...MODIFY 语句不会为现有列填充默认值。

删除表列

现在假设商业规则再次发生改变，不再需要前面示例中添加和修改的列。下面的语句从 PHONE 表中删除 PHONE_PRIMARY_S 和 PHONE_CATEGORY_S 列：

```
ALTER TABLE PHONE
DROP (PHONE_PRIMARY_S, PHONE_CATEGORY_S);
```

Note 注意：删除一个列时，与这个列关联的（如果有）所有索引或约束也都被删除。如果被删除的列被外键或检查约束引用，则必须指定 CASCADE CONSTRAINTS，否则 Oracle 将生成一个错误。

修改约束

MODIFY CONSTRAINT 子句能够把现有列或表约束的 INITIALLY IMMEDIATE 状态更改为 INITIALLY DEFERRED，假定这个约束用关键字 DEFERRABLE 创建。在第 4 章给出的例子中，当数据加载完成后，可以提交下面的命令：

```
ALTER TABLE SALESMAN
MODIFY CONSTRAINT chk_salesstatus INITIALLY IMMEDIATE;
```

目前，这个约束的行为与一个不可延迟约束的行为难以区分；也就是说，在每一个 DDL 语句之后都要检查这个约束。当想要执行另一次数据加载时，下面的语句再次改变这个约束的行为：

```
ALTER TABLE SALESMAN
MODIFY CONSTRAINT chk_salesstatus INITIALLY DEFERRED;
```

创建新约束

随着新列的创建，可能需要在现有表上创建新的约束。这种情形通常类似于可能需要新表列的情形——例如，当实现新的商业规则或修改有缺陷的设计时。例如，可能意识到 SALESMAN 表中的销售

www.TopSage.com

大家网 TopSage.com

人员的代码必须是唯一的，或者表 PRODUCT 丢失了主键。下面的语句可以实现这些任务：

```
ALTER TABLE salesman
ADD CONSTRAINT uk_salesmancode UNIQUE (salesman_code_s);
```

```
ALTER TABLE product
ADD CONSTRAINT pk_productprim PRIMARY KEY (prod_id_n)
```

下面是一些在前一章中创建的表上实现引用完整性约束的例子：

```
ALTER TABLE customer ADD CONSTRAINT fk_cust_payterms
FOREIGN KEY (cust_paytermsid_fn)
REFERENCES payment_terms (payterms_id_n);
```

```
ALTER TABLE customer ADD CONSTRAINT fk_cust_salesman
FOREIGN KEY (cust_salesmanid_fn)
REFERENCES salesman (salesman_id_n);
```

```
ALTER TABLE order_line ADD CONSTRAINT fk_ordline_ordhdr
FOREIGN KEY (ordline_ordhdrid_fn)
REFERENCES order_header (ordhdr_id_n);
```

```
ALTER TABLE order_line ADD CONSTRAINT fk_ordline_product
FOREIGN KEY (ordline_prodid_fn)
REFERENCES product (prod_id_n);
```

交叉参考：有关 ACME 表、索引和约束的完整列表，请参考附录 B。

删除约束

ALTER TABLE 语句能够删除列或表约束。例如，要删除刚刚创建的唯一约束，使用下面的语句：

```
ALTER TABLE SALESMAN
DROP CONSTRAINT uk_salesmancode;
```

禁用和启用约束

在某些情况下，可能需要延迟约束的动作：

```
ALTER TABLE SALESMAN
DISABLE CONSTRAINT chk_salesstatus;

Some data load action here...

ALTER TABLE SALESMAN
ENABLE CONSTRAINT chk_salesstatus;
```

提示：也可以使用 ENABLE | DISABLE ALL TRIGGERS 子句来启用或禁用与指定表关联的所有触发器（第 14 章中讨论的特殊数据库对象）。此外，可以使用 ENABLE | DISABLE ALL LOCK 语法锁定和解锁表。这个选项实际上阻止用户访问表——例如，在需要使用 ALTER 表命令添加或删除列、修改约束、更改默认值等情况时。

重命名表

可以使用 RENAME TO 子句更改表的名称。例如，如果觉得 SALESMAN 这个名称不太准确，可以使用下面的命令将它改为 SALESPERSON：

```
ALTER TABLE SALESMAN
RENAME TO SALESPERSON;
```

注意，更改后，所有程序（包括 PL/SQL 程序、函数、包以及引用这个重命名表的触发器）都将失效。必须在代码中找出并更改所有对 SALESMAN 表的引用，并重新编译这些程序。

提示：在一个表上执行的任何 DDL 语句会使所有引用了修改表的数据库对象无效，但是与重命名一个表的情形不同，当被会话首次引用时，这些对象不会被 Oracle 自动重新编译。

现在您发现这是一项过于庞大的工作，因此决定把名称改回去。使用下面语句即可：

```
ALTER TABLE SALESPERSON
RENAME TO SALESMAN;
```

3. DB2 9.5

DB2 的 ALTER TABLE 语句允许添加或删除列和约束、更改列的长度和数据类型、修改标识列属性和汇总表选项、改变物理属性、防止表及其驻留的表空间被删除，等等。语法（针对本书的目的进行了简化）如下：

www.TopSage.com

```
ALTER TABLE [<qualifier>.]<table_name>
[ADD COLUMN <column_definition>],... |
[DROP COLUMN <column_name> [CASCADE | RESTRICT]]
[ALTER COLUMN [SET {DATA TYPE <new_data_type> |
                    [NOT NULL] DEFAULT <default_value>}] |
              [DROP {IDENTITY | EXPRESSION | DEFAULT | NOT NULL}] |
              [SECURED WITH <security_label_name>]
              [DROP COLUMN SECURITY] |
              [GENERATED {ALWAYS | BY DEFAULT}] |
              [<identity_column_options>]
] |
[ADD CONSTRAINT <constraint_definition>] |
[DROP {PRIMARY KEY |
       [FOREIGN KEY | UNIQUE | CHECK | CONSTRAINT] <constraint_name>
      }
] |
[{ADD | DROP} RESTRICT ON DROP] |
[<physical_options>] |
[SET SUMMARY AS {DEFINITION ONLY | <summary_table_definition>}]
```

向表添加新列

下面的语句把两个新列添加到 PHONE 中：

```
ALTER TABLE PHONE
ADD PHONE_PRIMARY_S     CHAR(1) DEFAULT 'Y'
                        CHECK (PHONE_PRIMARY_S IN ('Y', 'N'))
ADD PHONE_CATEGORY_S    CHAR(15)
```

修改现有列

可以使用 ALTER TABLE 语句的 SET DATA TYPE 子句将一个列的现有数据类型改为兼容的数据类型，或者增加列的指定长度、精度或规模。DB2 中并不支持减少列的长度、精度或规模。

下面的语句把 CUST_ALIAS_S 的大小从 VARCHAR（15）增加到 VARCHAR（20）。

```
ALTER TABLE CUSTOMER
ALTER COLUMN CUST_ALIAS_S SET DATA TYPE VARCHAR (20)
```

下面这条语句也是有效的：

```
ALTER TABLE CUSTOMER
ALTER COLUMN CUST_ALIAS_S SET DATA TYPE CHARACTER (30)
```

下面的语句会失败，因为指定的长度小于原始长度：

```
ALTER TABLE CUSTOMER
ALTER COLUMN CUST_ALIAS_S SET DATA TYPE VARCHAR (10)
```

可以将一个列改为 NOT NULL，但是然后也必须为该列指定一个默认值。例如：

```
ALTER TABLE PHONE
ALTER COLUMN PHONE_PRIMARY_S SET NOT NULL
ALTER COLUMN PHONE_CATEGORY_S SET DEFAULT 'BUSINESS';
```

这个例子是可行的，因为 PHONE_PRIMARY_S 列具有一个默认值，它是在添加该列时指定的。注意反过来这个规则并不是强制的——可以添加 PHONE_PRIMARY_S 列的默认值，但是不一定是 NOT NULL 列。

修改标识列选项

可以修改现有标识列的选项。假定已经创建了表 PARMENT_TERMS，它有一个标识列 PAYTERMS_ID_N（参阅第 4 章），下面的语句以这样一种方式改变标识列：在达到最大值 500 后，从 100 开始循环：

```
ALTER TABLE payment_terms
ALTER COLUMN payterms_id_n
SET CYCLE
SET MAXVALUE 500
RESTART WITH 100
```

Note 注意：如果 PAYTERMS_ID_N 列没有被创建为标识列，则这条语句将失败。

在现有表上创建新的约束

可以使用 ALTER TABLE 语句添加 PRIMARY KEY、UNIQUE、CHECK 或引用完整性约束。下面是一些例子：

```
ALTER TABLE product
ADD CONSTRAINT pk_productprim PRIMARY KEY (prod_id_n)
```

```
ALTER TABLE customer ADD CONSTRAINT fk_cust_payterms
FOREIGN KEY (cust_paytermsid_fn)
REFERENCES payment_terms (payterms_id_n)
```


www.TopSage.com

```
ALTER TABLE customer ADD CONSTRAINT fk_cust_salesman
FOREIGN KEY (cust_salesmanid_fn)
REFERENCES salesman (salesman_id_n)
```

```
ALTER TABLE order_line ADD CONSTRAINT fk_ordline_ordhdr
FOREIGN KEY (ordline_ordhdrid_fn)
REFERENCES order_header (ordhdr_id_n)
```

```
ALTER TABLE order_line ADD CONSTRAINT fk_ordline_product
FOREIGN KEY (ordline_prodid_fn)
REFERENCES product (prod_id_n)
```

删除约束

使用 ALTER TABLE...DROP CONSTRAINT 语句删除现有约束。例如，要删除刚创建的唯一性约束，使用：

```
ALTER TABLE SALESMAN
DROP CONSTRAINT uk_salescode
```

改变汇总表选项

可以改变汇总表选项。例如，如果创建了 ORDERLINE_SUMMARY 表（参阅第 4 章），那么下面的语句会把它从 INITIALLY DEFERRED REFRESH DEFERRED 改为 DEFINITION ONLY：

```
ALTER TABLE orderline_summary
SET SUMMARY AS DEFINITION ONLY
```

4. Microsoft SQL Server 2008

Microsoft SQL Server 允许改变现有列的数据类型和排序规则序列；添加新列和删除现有列；添加、删除和启用约束；禁用或启用表触发器：

```
ALTER TABLE [<qualifier>.]<table_name>
(
  [ALTER COLUMN <column_name> <new_data_type> ]
  [COLLATE <collation_name>] |
  [ADD <column_definition> [<column_constraint>,...]] |
  [DROP COLUMN <column_name>,...]
  [[WITH {CHECK | NOCHECK}] ADD CONSTRAINT
                          <constraint_definition>,...] |
  [DROP CONSTRAINT <constraint_name> [WITH ONLINE = {ON|OFF},...] |
  [{CHECK | NOCHECK} CONSTRAINT {ALL | <constraint_name>,...}] |
  [{ENABLE | DISABLE} TRIGGER {ALL | <trigger_name>,...}]
)
```

Note 注意：只有检查和引用完整性约束支持禁用和启用约束。

向表添加新列

可以使用 Microsoft SQL Server 的 CREATE TABLE 语句中列定义的语法向表添加新列。也可以指定排序规则序列、默认值、标识属性和约束。下面的语句把两个新列添加到表 PHONE：

```
ALTER TABLE PHONE
ADD PHONE_PRIMARY_S     CHAR(1) CONSTRAINT df_phoneprim DEFAULT 'Y'
                        CONSTRAINT chk_phoneprim
                        CHECK (PHONE_PRIMARY_S IN ('Y', 'N')),
PHONE_CATEGORY_S        CHAR(15)
```

修改现有列

Microsoft SQL Server 中修改现有列的规则与 Oracle 或 DB2 不同。例如，不能更改一个约束所基于的或赋予了默认值的列的数据类型。也不能改变具有 TEXT、NTEXT、IMAGE 或 TIMESTAMP 数据类型的列。还有其他一些限制；参阅供应商的文档了解详细内容。

与 Oracle 不同，在 Microsoft SQL Server 中可以减小非空列的大小，只需确保所有现有数据适合新的大小即可。例如，Microsoft SQL Server 允许把 PHONE_TYPE_S 列从 VARCHAR（20）减少到 VARCHAR（10），甚至 VARCHAR（5），因为存储在这个 PHONE_TYPE_S 列中的最长字符串只有 5 个字符长。但是，如果试图把这个列的大小减少到最大 4 个字符，则 RDBMS 会返回一个错误。

Microsoft SQL Server 对数字列的处理也类似。可以减小规模，但只能减小至最大列值的大小。下面的语句把 ORDLINE_ORDQTY_N 列的数据类型从 INTEGER 改为 NUMERIC（3）：

```
ALTER TABLE ORDER_LINE
ALTER COLUMN ORDLINE_ORDQTY_N NUMERIC(3)
```

如果这个列中当前最大值是 100 或更大，那么就不能把它改为 NUMERIC（2）。

www.TopSage.com

大家网 TopSage.com

删除表列

在 Microsoft SQL Server 中删除表列比在 Oracle 中删除表列的限制更多。例如，如果一个列用在索引或约束中，或者与默认值关联，或绑定到一个规则，则不能删除这个列。下面的语句会失败，因为 PHONE_PRIMARY_S 列用于检查约束 CHK_PHONEPRIM，而且还与一个默认值关联：

```
ALTER TABLE PHONE
DROP COLUMN PHONE_PRIMARY_S, PHONE_CATEGORY_S
Server: Msg 5074, Level 16, State 1, Line 1
The object 'df_phoneprim' is dependent on column 'PHONE_PRIMARY_S'.
Server: Msg 5074, Level 16, State 1, Line 1
The object 'chk_phoneprim' is dependent on column 'PHONE_PRIMARY_S'.
Server: Msg 4922, Level 16, State 1, Line 1
ALTER TABLE DROP COLUMN PHONE_PRIMARY_S failed because one or
more objects access this column.
```

下面的语句会成功，因为单独的 PHONE_CA TEGORY_S 列不受任何上述限制：

```
ALTER TABLE phone
DROP COLUMN phone_category_s
```

创建和删除约束

Microsoft SQL Server 中创建新约束或删除现有约束的语法与 Oracle 和 DB2 中的语法相同：

```
ALTER TABLE salesman
ADD CONSTRAINT uk_salesmancode UNIQUE (salesman_code_s);
```

```
ALTER TABLE SALESMAN
DROP CONSTRAINT uk_salesmancode;
```

禁用和启用约束

可以禁用和启用检查约束和引用完整性约束。例如：

```
ALTER TABLE PHONE
NOCHECK CONSTRAINT chk_phoneprim
```

```
ALTER TABLE PHONE
CHECK CONSTRAINT chk_phoneprim
```

5.1.2 DROP TABLE 语句

由于表占用物理空间，出于效率方面的考虑，在不需要这些表时能删除它们十分重要。DROP TABLE 语句在“三大”数据库中的作用方式非常相似。它释放删除的表及其列上的任何索引所占用的物理存储空间。表的定义、与之关联的索引、完整性约束和触发器都将从数据库的数据字典中删除。在大多数主要 RDBMS 实现中，所有引用了要删除表的视图都会变得无效，而且视图必须更改所有对被删除表的引用，或者手动从数据库中删除，这与 SQL:2003 标准不一致，后者要求删除视图以及所有引用了该表的对象。

Caution

警告：DROP TABLE 是 DDL 的一部分，因此是不可逆的。具体而言，它是立即提交到数据库，没有回滚的可能（至少在“三大”数据库中如此，不过这样的行为不符合 SQL:2003 标准。）使用该语句时要极为小心。

1. SQL:2003

SQL:2003 的 DROP TABLE 语法是：

```
DROP TABLE <table_name> [{CASCADE | RESTRICT}]
```

如果指定了 CASCADE 选项，则数据库中所有相关对象（视图、约束、触发器、断言等）也将被删除，其他表中所有引用了该表的行也会被删除。

2. Oracle 11g

Oracle 删除表的语法如下：

```
DROP TABLE [<qualifier>.]<table_name> [CASCADE CONSTRAINTS][PURGE];
```

可选的 CASCADE CONSTRAINTS 子句允许删除具有强制引用完整性约束的表；这些约束也将被删除。如果忽略这个子句且存在这样的引用完整性约束，则 Oracle 将生成一条错误并且不会删除这个表：

```
SQL> DROP TABLE CUSTOMER;

DROP TABLE CUSTOMER
           *
ERROR at line 1:
ORA-02449: unique/primary keys in table referenced by foreign keys
```


www.TopSage.com

问题是 ADDRESS、PHONE 和 ORDER_HEADER 表引用了 CUSTOMER 表（约束 FK_ADDR_CUST、FK_PHONE_CUST 和 FK_ORDHDR_CUSTOMER）。要删除 CUSTOMER 表和上面的约束，应该使用下面这条 SQL 语句：

```
SQL> DROP TABLE customer CASCADE CONSTRAINTS;

Table dropped.
```

PURGE 选项删除表并一步释放表空间。换言之，表和相关对象不会被置于回收箱中。

Note 注意：Oracle 10g 引入了回收箱，这是一个数据字典表，它包含有关删除的表、索引、约束等的信息。这个概念与 Microsoft Windows 的回收箱非常接近——仍然可以访问和还原被删除的数据，直到回收箱被“清空”。

3. DB2 9.5

DB2 删除一个表的语法如下：

```
DROP TABLE [<qualifier>.]<table_name>
```

除了列出的全部对象外（索引、约束和触发器），引用这个表的所有汇总表也将被删除，包括所有关联的引用完整性约束：

```
db2 => DROP TABLE customer

DB20000I The SQL command completed successfully.
```

这种行为可能导致无意的数据丢失等灾难性后果，因此，在 DB2 中使用 DROP TABLE 语句时要特别小心。

4. Microsoft SQL Server 2008

Microsoft SQL Server 语法提供了同时删除多个表的功能：

```
DROP TABLE [<qualifier>.]<table_name>,...
```

与 Oracle 不同，如果任何外键引用了表中的一列则 Microsoft SQL Server 不允许删除该表，因此在删除该表之前，必须先删除引用完整性约束。另一种方法是使用 DROP 语句删除引用和被引用的表——唯一的限制是必须首先列出引用表。

5.2 索引

我们已经知道，索引对大多数数据库用户是不可见的，因此，在大多数情况下，不需要改变索引。而且始终可以删除并重建一个索引，而不是修改它。DROP INDEX 语句用于删除索引。

5.2.1 ALTER INDEX 语句

因为 ALTER INDEX 语句不是 SQL:2003 标准的一部分，所以 RDBMS 供应商以不同方式实现它。

1. Oracle 11g

同样，我们不打算集中于 ALTER INDEX 语句的细节，请参考 Oracle 文档了解更多信息。在本书中，我们只举例说明几个比较重要的子句：

```
ALTER INDEX <index_name>
{[RENAME TO <new_name>] |
 [REBUILD TABLESPACE <tablespace_name>]
};
```

2. 重命名索引

如果使用系统生成的名称创建索引（也就是说，如果没有在对象创建时专门命名它们），则可以使用 RENAME。例如，假定使用下面的语句创建了表 SALESMAN：

```
CREATE TABLE salesman
(
   salesman_id_n              NUMBER         PRIMARY KEY,
   salesman_code_s            VARCHAR2(2)    UNIQUE,
   salesman_name_s            VARCHAR2(50)   NOT NULL,
   salesman_status_s          CHAR(1)        DEFAULT 'Y',
  CONSTRAINT chk_salesstatus CHECK
            (salesman_status_s in ('N', 'Y'))
);
```

Oracle 将自动在 SALESMAN_ID_N 和 SALESMAN_CODE_S 列上创建索引（记住，索引总是在主键

www.TopSage.com

或唯一列上创建），但这些名称并不是描述性的——诸如SYS_C003521和SYS_C003522之类的名称。如果数据库的某些命名约定类似于附录 B 中描述的那样（这个想法不错），那么可能希望把这些索引的名称改为更具描述性的名称：

```
ALTER INDEX SYS_C003521
RENAME TO IDX_SALESMAN_ID;

ALTER INDEX SYS_C003522
RENAME TO IDX_SALESMAN_CODE;
```

3. 将索引重建到不同的表空间

在创建 Oracle 索引时，一个时常发生的错误是在错误的表空间中创建索引。这甚至比使用表发生错误的次数还多，因为 Oracle 没有提供默认索引表空间选项——默认情况下，在分配的默认数据表空间中创建所有物理对象（包括索引）。我们并不总是希望这样，特别是在生产环境中，即数据和索引表空间通常位于单独的物理设备（硬盘）上。在错误的表空间中创建索引会明显降低性能。

可以使用 ALTER INDEX 命令来解决这个问题：

```
ALTER INDEX IDX_SALESMAN_ID
REBUILD TABLESPACE INDEX01;

ALTER INDEX IDX_SALESMAN_CODE
REBUILD TABLESPACE INDEX01;
```

Note 注意：这个例子假定表空间 INDEX01 存在。

4. DB2 9.5

DB2 中的 ALTER INDEX 命令语法提供了修改存储特性、添加索引列、更改聚类分析等更多功能。其语法如下：

```
ALTER INDEX {ADD COLUMN (<column_name> {ASC | DESC}) |
                        [NOT] CLUSTER |
                        [NOT] PADDED |
                        <storage_options> |
                        <connection_options> |
                        <update_characteristics>
```

5. Microsoft SQL Server 2008

Microsoft SQL Server 的 ALTER INDEX 语法允许重建、重组或禁用现有索引；此外，可以重组某些设置选项：

```
ALTER INDEX {<index_name> | ALL } ON <object>
 {
   REBUILD <rebuild_options> |
   REORGANIZE <reorganize_options> |
   DISABLE |
   SET (<set_index_option>,...)
 }
```

下面的例子显示如何禁用 idx_cust_name_als 索引：

```
ALTER INDEX idx_cust_name_als
ON customer
DISABLE
```

5.2.2 DROP INDEX 语句

与 DROP TABLE 语句类似，DROP INDEX 释放已分配的空间，并从数据库信息架构中删除索引定义。不能使用 DROP TABLE 删除为实现 PRIMARY KEY 或 UNIQUE 约束而创建的索引——必须改用 ALTER TABLE...DROP CONSTRAINT 语句。

Cross-Reference 交叉参考：数据库信息架构在第 13 章讨论。

1. Oracle 11g

Oracle 11g 的语法如下：

```
DROP INDEX [<qualifier>.]<index_name>;
```

因为 PK_ORDHDRPRIM 是表 ORDER_HEADER 上的主键，所以第一条语句返回一个错误，但是第二条运行正常（假设索引 idx_phone_cust 存在）：

www.TopSage.com

大家网 TopSage.com

```
SQL> DROP INDEX pk_ordhdrprim;
DROP INDEX pk_ordhdrprim
           *

ERROR at line 1:
ORA-02429: cannot drop index used for enforcement of
  unique/primary key
```

```
SQL> DROP INDEX idx_phone_cust;

Index dropped.
```

当删除一个索引时，所有依赖于基础表的对象都失效。

2. DB2 9.5

DB2 9.5 使用的语法与 Oracle 中的相同：

```
DROP INDEX [<qualifier>.]<index_name>
```

假设要删除 PHONE 表 PHONE_CUSTID_FN 列上的索引 IDX_PHONE_CUST（假设触发器存在）：

```
DROP INDEX idx_phone_cust
```

DB2 会使依赖这个索引的包失效。

3. Microsoft SQL Server 2008

必须指定索引列所在的表名和索引名称：

```
DROP INDEX <index_name> ON <object> [,...]
```

例如（假设 IDX_PHONE_CUST 索引存在）：

```
DROP INDEX idx_phone_cust ON phone
```

Tip　提示：可以在一条 DROP INDEX 语句中删除多个索引。名称之间必须用逗号分隔。

5.3 视图

分别使用 ALTER VIEW 和 DROP VIEW 语句来修改和删除视图。

5.3.1 ALTER VIEW 语句

Oracle 11g、DB2 和 Microsoft SQL Server 都有一条 ALTER VIEW 语句，但其语法和功能有很大不同。

Note　注意：SQL:2003 不包含 ALTER VIEW 规范；不过该语句在未来可能会成为 SQL 标准的一部分。

1. Oracle 11g

在 Oracle 中，ALTER VIEW 的两个主要用用途是重新编译视图，该视图曾因对基础表进行 DDL 修改而失效；以及添加、修改或删除视图约束。其语法如下：

```
ALTER VIEW [<qualifier>.]<view_name>
{[COMPILE] |
 [ADD CONSTRAINT <view_constraint_definition>] |
 [MODIFY CONSTRAINT <constraint_name>] |
 [DROP {CONSTRAINT <constraint_name> | PRIMARY KEY}]
};
```

例如，可能需要重新编译视图 V_FAX_NUMBER，因为它的基础表（PHONE）已经被修改：

```
ALTER VIEW v_fax_number
COMPILE;
```

Tip　提示：实际上，当 Oracle 视图的基础表修改后，不必重新编译这些视图。当 SQL 代码引用它们时，失效的视图将被自动重新编译。手动重新编译视图的主要目的是为了改善性能。

接下来的例子把 UNIQUE 约束添加到视图 V_CONTACT_LIST 中：

```
ALTER VIEW v_contact_list
ADD CONSTRAINT vpk_contactlist UNIQUE (name, phone_number)
DISABLE;
```

要删除这个约束，使用下面的语句：

```
ALTER VIEW v_contact_list
DROP CONSTRAINT vpk_contactlist;
```


www.TopSage.com

Note 注意：Oracle 的 ALTER VIEW 语句不能改变视图定义。如果需要添加或删除视图列、更改数据类型或基础查询等，则必须使用 CREATE OR REPLACE VIEW 语句（或者使用 DROP VIEW 和 CREATE VIEW 语句）。

2. DB2 9.5

DB2 允许使用 ALTER VIEW 语句改变一个引用类型列来添加一个范围。此外，可以指定是否使用视图来优化与该视图相关的查询性能：

```
ALTER VIEW <view_name>
{
 ALTER [COLUMN] <column_name> ADD SCOPE <identifier> |
 { ENABLE | DISABLE} QUERY OPTIMIZATION
}
```

这个功能是高级主题并且在本书中没有涉及。

3. Microsoft SQL Server 2008

Microsoft SQL Server 中的 ALTER VIEW 语句能够改变视图列、基础选择语句和其他视图选项，而不会影响相关数据库对象或改变权限（这与使用 DROP VIEW 和 CREATE VIEW 语句不同）。其语法几乎与 CREATE VIEW 一样：

```
ALTER VIEW [<qualifier>.]<view_name>
[(<column_name>,...)]
[WITH {ENCRYPTION|SCHEMABINDING|VIEW_METADATA}]
AS <select_statement>
[WITH CHECK OPTION]
```

5.3.2 DROP VIEW 语句

DROP VIEW 语句从系统目录中删除视图定义。相关对象变为无效。

1. SQL:2003

SQL:2003 将视图看作是一种特殊的表，因此 DROP VIEW 语句的语法与 DROP TABLE 相同：

```
DROP VIEW <view_name> [ {CASCADE | RESTRICT} ]
```

2. Oracle 11g

Oracle 11g 的 DROP VIEW 语法如下：

```
DROP VIEW view_name [CASCADE CONSTRAINTS];
```

例如：

```
DROP VIEW v_phone_number;
```

选项 CASCADE CONSTRAINTS 子句删除视图中所有引用了主键和唯一键的引用完整性约束。参见前面介绍的 Oracle 的 DROP TABLE 语句。

3. DB2 9.5

DB2 使用下面的语法：

```
DROP VIEW [<qualifier>.]<view_name>
```

4. icrosoft SQL Server 2008

可以使用 Microsoft SQL Server 的 DROP VIEW 语句删除多个视图：

```
DROP VIEW <view_name> [,...]
```

下面的语句一次删除三个视图（V_PHONE_NUMBER、V_CUSTOMER_TOTALS 和 V_WILE_BESS_ORDERS）：

```
DROP VIEW v_phone_number, v_customer_totals, v_wile_bess_orders
GO
```

5.4 别名和同义词

在 Oracle 和 DB2 中，不能修改别名和同义词。如果必须更改一个同义词（别名）的定义，可以使用 DROP SYNONYM（ALIAS）和 CREATE SYNONYM（ALIAS）删除并重建它。

1. Oracle 11g

删除 Oracle 同义词的语法如下：

```
DROP [PUBLIC] SYNONYM <synonym_name> [FORCE];
```

如果要删除的同义词为公共同义词，则必须使用 PUBLIC 关键字。例如：下面说明了如何删除前

www.TopSage.com

一章创建的公共同义词 SHIPMENT：

```
DROP PUBLIC SYNONYM SHIPMENT;
```

删除同义词时，Oracle 不会使同义词所引用的任何对象失效，但物化视图除外。

2. DB2 9.5

在 DB2 中删除别名的语法如下：

```
DROP ALIAS <alias_name>
```

例如：

```
DROP ALIAS SHIPMT
```

所有引用了被删除别名的表、视图和触发器都不再可用。

3. Microsoft SQL Server 2008

Microsoft SQL Server 的语法与 Oracle 一样：

```
DROP SYNONYM <synonym_name>
```

5.5 架构

SQL:2003 不包含 ALTER SCHEMA 语句，但是以后可能会包含。DROP SCHEMA 是 SQL 标准的一部分。Oracle 没有更改或删除架构的语法（根据 Oracle 的架构定义，这样做是有意义的）；DB2 允许删除架构但不能更改它们；Microsoft SQL Server 包括 ALTER SCHEMA 和 DROP SCHEMA 的语法。

5.5.1 SQL:2003

SQL:2003 标准了为 DROP SCHEMA 语句提供了如下语法：

```
DROP SCHEMA <schema_name> [CASCADE | RESTRICT]
```

使用 RESTRICT 关键字指定架构必须为空。CASCADE 表示所有架构对象也将被删除。

5.5.2 DB2 9.5

DB2 删除架构的语法如下：

```
DROP SCHEMA <schema_name> RESTRICT
```

要从数据库中删除一个架构，这个架构必须为空（其中没有定义任何对象）。除非删除 ACMETEST.ADDRESS 和 ACMETEST.SALESMAN 表，否则下面的语句将失败（第 4 章的例子）：

```
DROP SCHEMA acmetest RESTRICT

The object type "SCHEMA" cannot be dropped because there is an object
"ACMETEST.ADDRESS" of type "TABLE" which depends
on it. SQLSTATE=42893
```

```
DROP TABLE ACMETEST.ADDRESS

DB20000I The SQL command completed successfully.
```

```
DROP TABLE ACMETEST.SALESMAN

DB20000I The SQL command completed successfully.
```

```
DROP SCHEMA acmetest RESTRICT

DB20000I The SQL command completed successfully.
```

5.5.3 Microsoft SQL Server 2008

在 Microsoft SQL Server 中，ALTER SCHEMA 只能用于同一数据库内的架构之间的对象移动：

```
ALTER SCHEMA <schema_name>
TRANSFER <another_schema_name>.<object_name>
```

例如，假设架构 TEST 存在（参见前一章的例子），下面的语句将表 phone 从架构 ACME 传送到架构 TEST：

```
ALTER SCHEMA TEST
TRANSFER acme.phone
```

删除空架构的语法如下：

```
DROP SCHEMA <schema_name>
```


www.TopSage.com

大家网 TopSage.com

例如：

```
DROP SCHEMA TEST
```

Note 注意：前面的例子假设架构 TEST 为空——换言之，在执行 DROP SCHEMA TEST 之前已经删除了表 ADDRESS、PHONE 和 SALESMAN。

5.6 序列

在 Oracle 和 DB2 中可以修改或删除序列。SQL:2003 也包含修改和删除序列的语句。

5.6.1 ALTER SEQUENCE 语句

几乎所有可以与 CREATE SEQUENCE 语句（参阅第 4 章）一起使用的序列的选项也可以与 ALTER SEQUENCE 语句一起使用。

1. SQL:2003

ALTER SEQUENCE 能够改变序列增量、最大值和最小值、重新启动值和序列生成器的循环选项：

```
ALTER SEQUENCE <sequence_name>
[START WITH <start_value>]
[INCREMENT BY <increment_value>]
[MAXVALUE <max_value> | NOMAXVALUE]
[MINVALUE <min_value> | NOMINVALUE]
[CYCLE | NOCYCLE]
```

2. Oracle 11g

可以更改现有序列的增量、最小值和最大值、缓存数以及行为。只会影响将来的序列数。对于现有序列，唯一不能修改的子句是 START WITH：

```
ALTER SEQUENCE [<qualifier>.]<sequence_name>
[INCREMENT BY <increment_value>]
[MAXVALUE <max_value> | NOMAXVALUE]
[MINVALUE <min_value> | NOMINVALUE]
[CYCLE | NOCYCLE ]
[CACHE <value> | NOCACHE]
[ORDER | NOORDER]
```

下面的语句把 MY_SEQUENCE4 改为不具有最大值且不进行循环：

```
ALTER SEQUENCE my_sequence4
INCREMENT BY 1
NOMAXVALUE
NOCYCLE;
```

3. DB2 9.5

DB2 允许重新启动序列，更改它的增量（仅对将来的值），设置或消除最小值或最大值，更改缓存值的数目等：

```
ALTER SEQUENCE <sequence_name>
[RESTART WITH <start_value>]
[INCREMENT BY <increment_value>]
[MAXVALUE <max_value> | NOMAXVALUE]
[MINVALUE <min_value> | NOMINVALUE]
[CYCLE | NOCYCLE ]
[CACHE <value> | NOCACHE]
[ORDER | NOORDER]
```

例如：

```
ALTER SEQUENCE my_sequence4
RESTART WITH 1
INCREMENT BY 1
NOMAXVALUE
NOCYCLE;
```

5.6.2 DROP SEQUENCE 语句

可以使用 DROP SEQUENCE 语句删除序列。

1. SQL:2003

SQL:2003 语法包含 CASCADE 和 RESTRICT 关键字：

```
DROP SEQUENCE <sequence_name> [CASCADE | RESTRICT]
```


www.TopSage.com

如果指定了 RESTRICT 选项，则不能删除数据库中任意触发器或**例程**引用的序列，就不能删除它。

Note 注意：触发器和例程将在第 14 章讨论。

2. Oracle 11g

Oracle 11g 中的 DROP SEQUENCE 语法如下：

```
DROP SEQUENCE [<qualifier>.]<sequence_name>;
```

不存在依赖性。换言之，数据库对象不会妨碍删除序列，对象也不会失效。下面这条语句删除序列 MY _SEQUENCE1：

```
DROP SEQUENCE my_sequence1;
```

3. DB2 9.5

DB2 语法与 Oracle 没有太大区别——唯一不同在于末尾的必需有关键字 RESTRICT：

```
DROP SEQUENCE <sequence_name> RESTRICT
```

这个例子等价于前面例子中的 Oracle 语法：

```
DROP SEQUENCE my_sequence2 RESTRICT
```

5.7 其他特定于实现的对象

本节将学习如何更改和删除一些在第 4 章中讨论的特定于实现的对象。

5.7.1 域（SQL:2003）

修改域的语法如下：

```
ALTER DOMAIN <domain_name>
[SET <default_clause | DROP DEFAULT]
[ADD <domain_constraint > | DROP CONSTRAINT <constraint_name>,...]
```

要删除域，使用下面的语法：

```
DROP DOMAIN <domain_name> [CASCADE | RESTRICT]
```

5.7.2 字符集（SQL:2003）

不能修改字符集。从数据库中删除字符集的语句如下：

```
DROP CHARACTER SET <character_set_name>
```

5.7.3 排序规则（SQL:2003）

SQL:2003 标准中也没有改变排序规则的内容。删除排序规则的语法如下：

```
DROP COLLATION <collation_name> [CASCADE | RESTRICT]
```

5.7.4 表空间

Oracle 和 DB2 中都允许更改和删除现有表空间。

1. ALTER TABLESPACE 语句

有几个选项可用于修改现有表空间。

Oracle 11g

对表空间最常用的操作包括把一个新文件添加到现有表空间，扩展到任意范围，针对备份锁定和解锁表空间，以及给表空间指定不同的名称：

```
ALTER TABLESPACE <tablespace_name>
{
 ADD DATAFILE <file_path_and_name> SIZE size {[K|M|G|T|P|E]}|
 COALESCE |
 {BEGIN | END} BACKUP |
 {ONLINE | OFFLINE} |
 RENAME TO <new_tablespace_name>
;
```

下面的命令把一个 1MB 大小的新数据文件 C:\oracle\ora92\oradata\acme\data01.dbf 添加到表空间 DATA01（假设提交这个语句的用户具有相应的特权）：

```
ALTER TABLESPACE DATA01
ADD DATAFILE 'C:\oracle\ora92\oradata\acme\data01.dbf SIZE 1M';
```


www.TopSage.com

Note 注意：Oracle 中还有许多选项可用于 ALTER TABLESPACE 语句，但它们主要供数据库管理员使用，本书不讨论这些选项。

DB2 9.5

与 Oracle 类似，DB2 中的 ALTER TABLESPACE 语句主要供 DBA 使用。例如，可以把一个容器添加到使用 MANAGED BY DATABASE 选项创建的表空间，或者增加它的大小。

Note 注意：我们已经从第 4 章知道 Microsoft SQL Server 2008 使用文件组的方式类似于 Oracle 和 DB2 使用表空间的方式。第 4 章还讨论了 ALTER DATABASE...ADD FILE 命令。

2. DROP TABLESPACE 语句

通常只有数据库管理员有必要的特权来删除现有表空间。请注意，删除表空间会删除表空间中定义的所有对象——应当谨慎处理。

Oracle 11g

在 Oracle 中，可以指定 DROP TABLESPACE 语句的几个选项：

```
DROP TABLESPACE <tablespace_name>
[INCLUDING CONTENTS [AND DATAFILES]]
[CASCADE CONSTRAINTS];
```

如果想删除包含对象的表空间，则必须指定 INCLUDING CONTENTS；否则，Oracle 将生成一个错误。默认情况下，如果实际的工作文件属于“受托管字典”表空间，则不会被删除，因此必须指定 AND DATAFILES 子句，否则，就需要手动删除。指定 CASCADE CONSTRAINTS 删除所有表空间以外的表的引用完整性约束，它们引用了表空间内表的主键和唯一键。

Note 注意：如果这个表空间是“本地管理”，则不需要 AND DATAFILES 子句——OS 文件将自动被删除。

DB2 9.5

DB2 的 DROP TABLESPACE 语法如下：

```
DROP TABLESPACE[S] <tablespace_name>,...
```

可以在一条 DROP TABLESPACE 语句中删除多个表空间。所有由系统管理的表空间的 OS 文件将被删除；由用户创建的容器不会被删除。如果具有 RESTRICT ON DROP 选项的任意表驻留在要被删除的表空间中，则不能删除这个表空间。

5.7.5 物化视图（Oracle 11g）

ALTER MATERIALIZED VIEW 语句有很多选项，但是其中大部分都与数据库管理员职责相关因此没有在本书中讨论。要从数据库中删除一个物化视图，使用下面的语法：

```
DROP MATERIALIZED VIEW <materialized_view_name> [PRESERVE TABLE];
```

如果使用 PRESERVE TABLE 子句，则物化视图容器表及其内容都保留为具有与被删除的物化视图相同名称的关系表。例如，执行下面的语句后，物化视图 TEST2 在数据库中消失；删除之前有一个名为 TEST2 的表，它具有物化视图拥有的全部数据：

```
DROP MATERIALIZED VIEW TEST2 PRESERVE TABLE;
```

5.7.6 数据库链接（Oracle 11g）

不能修改数据库链接。要删除数据库链接，使用下面的语法：

```
DROP [PUBLIC] DATABASE LINK <dblink>;
```

5.7.7 目录（Oracle 11g）

不能修改 Oracle 目录。删除目录的语法如下：

```
DROP DIRECTORY <directory_name>;
```

例如：

```
DROP DIRECTORY ACME;
```

Note 注意：前面的 DROP DIRECTORY 命令不删除实际的 OS 目录——它只删除此目录的 Oracle 引用以及所有相关特权。

www.TopSage.com

大家网 TopSage.com

5.8　ALTER 和 DROP 语句交叉参考

表 5-1 为“三大”数据库中使用 ALTER 语句可以改变的所有对象的完整列表。

表 5-1　　有效的 ALTER 语句交叉参考

语句	SQL:2003	Oracle 11g	DB2 9.5	Microsoft SQL Server 2008
ALTER ASSEMBLY				√
ALTER BUFFERPOOL			√	
ALTER CLUSTER		√		
ALTER DATABASE		√	√	√
ALTER DIMENSION		√		
ALTER DOMAIN	√			
ALTER FUNCTION		√	√	√
ALTER INDEX		√	√	√
ALTER INDEXTYPE		√		
ALTER JAVA		√		
ALTER MATERIALIZED VIEW		√		
ALTER MATERIALIZED VIEW LOG		√		
ALTER METHOD			√	
ALTER NICKNAME			√	
ALTER OUTLINE		√		
ALTER PACKAGE		√		
ALTER PACKAGE BODY		√		
ALTER PROCEDURE		√	√	√
ALTER PROFILE		√		
ALTER RESOURCE COST		√		
ALTER QUEUE				√
ALTER ROLE		√		
ALTER ROLLBACK SEGMENT		√		
ALTER ROUTINE	√			√
ALTER SCHEMA	√			
ALTER SEQUENCE	√	√	√	
ALTER SERVER			√	
ALTER SESSION		√		
ALTER SPFILE		√		
ALTER SYNONYM		√		
ALTER SYSTEM		√		
ALTER TABLE	√	√	√	√
ALTER TABLESPACE		√	√	
ALTER TRANSFORM	√			
ALTER TRANSLATION	√			
ALTER TRIGGER	√	√		√
ALTER TYPE	√	√	√	
ALTER TYPE BODY		√		
ALTER USER	√	√		√
ALTER USER MAPPING			√	
ALTER VIEW			√	
ALTER WRAPPER			√	
ALTER XSROBJECT			√	

表 5-2 为使用 DROP 语句可以删除的所有对象的完整列表。

www.TopSage.com

表 5-2 有效的 DROP 语句交叉参考

语句	SQL:2003	Oracle 11g	DB2 9.5	Microsoft SQL Server 2008
DROP ALIAS			√	
DROP ASSEMBLY				√
DROP ASSERTION	√			
DROP BUFFERPOOL			√	
DROP CHARACTER SET	√			
DROP CLUSTER		√		
DROP COLLATION	√			
DROP CONTEXT		√		
DROP DATABASE			√	√
DROP DATABASE LINK		√		
DROP DATABASE PARTITION GROUP			√	
DROP DEFAULT				√
DROP DIMENSION		√		
DROP DIRECTORY		√		
DROP DISTINCT TYPE			√	
DROP DOMAIN	√			
DROP EVENT MONITOR			√	
DROP EVENT NOTIFICATION				√
DROP FUNCTION		√	√	√
DROP FUNCTION MAPPING			√	
DROP INDEX		√	√	√
DROP INDEX EXTENSION			√	
DROP INDEXTYPE		√		
DROP JAVA		√		
DROP LIBRARY		√		
DROP MATERIALIZED VIEW		√		
DROP MATERIALIZED VIEW LOG		√		
DROP METHOD			√	
DROP MODULE	√			
DROP NICKNAME			√	
DROP OPERATOR		√		
DROP ORDERING	√			
DROP OUTLINE		√		
DROP PACKAGE		√		
DROP PACKAGE BODY		√	√	
DROP PROCEDURE		√	√	√
DROP PROFILE		√		
DROP QUEUE				√
DROP ROLE	√	√		
DROP ROLLBACK SEGMENT		√		
DROP ROUTINE MAPPING	√			
DROP RULE				√
DROP SCHEMA	√		√	√
DROP SEQUENCE	√	√	√	
DROP SECURITY LABEL			√	
DROP SECURITY LABEL COMPONENT			√	
DROP SECURITY POLICY			√	
DROP SEQUENCE			√	

续表

语句	SQL:2003	Oracle 11g	DB2 9.5	Microsoft SQL Server 2008
DROP SERVER			√	
DROP STATISTICS				√
DROP SYNONYM		√		
DROP TABLE	√	√	√	√
DROP TABLESPACE		√	√	
DROP TABLE HIERARCHY			√	
DROP TRANSFORM	√		√	
DROP TRANSLATION	√			
DROP TRIGGER	√	√	√	√
DROP TYPE	√	√	√	
DROP TYPE BODY			√	
DROP TYPE MAPPING				
DROP USER	√	√		√
DROP USER MAPPING			√	
DROP VIEW	√	√	√	√
DROP VIEW HIERARCHY			√	
DROP WRAPPER			√	
DROP XSROBJECT			√	

5.9 小结

数据库中的每个对象都有生命周期。可以创建、修改和删除对象。在RDBMS中，这些工作通过SQL语句完成，但这些SQL语句并不总是遵从SQL标准。一些供应商提供了比其他供应商更多的功能，而且他们对SQL标准的遵从程度也有很大差别。事实上，在数据库中创建的每个对象都可以被改变或删除。

在大多数情况下，修改数据库对象的功能是可选的——换言之，总是可以使用一组替代的语句（如DROP和CREATE）来达到同样的效果（只是会对引用了要删除对象的其他数据库对象产生潜在影响）。

删除数据库对象的功能也非常重要。它释放不再使用的物理对象空间，并且也可以用于更改那些不能被改变的对象（通过删除和重建）。

www.TopSage.com

大家网
TopSage.com

PART 3

数据操作和事务控制

本部分包括：

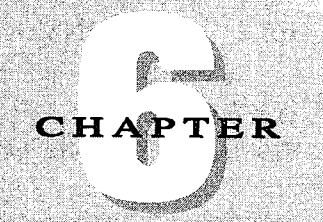
第6章
数据操作语言

第7章
会话、事务和锁

www.TopSage.com

www.TopSage.com

第 6 章
数据操作语言



本章提要：

- 插入新的记录
- 更新表记录
- 删除表记录
- MERGE 语句
- TRUNCATE 语句

第 4 章和第 5 章讨论了在 SQL 中十分重要的数据定义语言（DDL）部分，通过它可以创建、修改和删除不同的数据库对象。我们曾经强调过这样一个事实，表是数据库中最重要的对象，因为可以用它来存储数据，而数据就是数据库的一切。本章所研究的是数据操作语言（DML），它允许把数据添加到数据库中，并根据需要修改数据以及当不再需要这些数据时删除它们。

典型的 DML 由四种语句组成：INSERT，UPDATE，DELETE 以及 MERGE。MERGE 把 INSERT，UPDATE 以及 DELETE 的功能结合在了一起。

注意：本章中的例子是假定在第 4 章和第 5 章中开始的 ACME 数据库的基础上进一步扩展得来的。此数据库脚本中执行的一些约束可能会妨碍某些语句的成功执行。

6.1 INSERT：用数据来填充表

我们知道，在关系数据库中用表代表实体，或者说成至少他们应该代表。例如，CUSTOMER 表中的每一行存储的是与某个特定客户有关的信息；ORDER_HEADER 表中的一行就表示一个明确的定购信息等。通常，出现一个新的“现实”实体就要求插入一个新行。例如，如果 ACME 公司得到一个新客户的话，则需要在 CUSTOMER 表中插入一个新行；当某个客户发出了一个订单时，需要在 ORDER_HEADER 表中插入一个新行；如果 ACME 开始销售一种新产品，就必须在 PRODUCT 表中添加一个新行，依此类推。

使用 INSERT 语句向表中添加新行可以是直接的，也可以通过可更新的视图进行。其语法在 SQL:2003，Oracle 11g，DB2 9.5 以及 MS SQL Server 2008 之间稍有不同，但给出一种能够在所有“三大”数据库中工作的普通的 INSERT 语法是可能的，并且不与 SQL:2003 标准相矛盾。

```
INSERT INTO <table_or_view_name>
[(<column_name>,...)]
{ {VALUES (<literal> |
           <expression> |
           NULL |
           DEFAULT),...} |
  {<select_statement>} }
```

www.TopSage.com

大家网 TopSage.com

我们先来阐述 INSERT 一般的功能，然后再描述一些 SQL:2003 和开发商特有的属性。

6.1.1 一般的 INSERT 语句子句

典型的情况下，在一个 INSERT 语句中，可以一次将多个值插入到一个表中（Oracle 11g 允许执行多表插入操作，但这是较高级的功能）。表名在<table_or_view_name>子句中提供（可以使用所提供的某个可更新的视图名代替表名）。

<column_name>子句是可选的，它决定了要被插入的行的值以什么样的次序填充到列中。它也允许在某一特定的时刻跳过那些不想填充的列，比如那些 NULL 值的列。

警告：不能在<column_name>字句中跳过任何 NOT NULL 列，除非设置为默认填充；否则 RDBMS 会给出一个错误。

要插入的值可以用 VALUES 子句指定，也可以是<select_statement>子句（也被称为**子查询**）的结果集来指定。在第一种情况下，通常只有一行被插入；在第二种情况中，RDBMS 插入的行数与<select_statement>语句执行返回的相同——可以是零行、一行、十行或者一千行。

如果省略了 column_name 就必须在 VALUES 子句中列出所有表列的值，且列出的次序必须与表定义中被指定的列次序完全相同。

交叉参考：数据库对象的内部结构信息将会在第 13 章进行讨论。

如果<column_name>列存在的话，就必须为每一个给出的列指定相应的值。

注意：如果使用的是<select_statement>子句，而不是 VALUES 子句，则产生的结果集必须是按表定义的次序组织的（如果没有指定<column_list>列），或是按<column_list>次序组织的。

1. 为指定的列插入值

通常，插入某些列值为 NULL 的行的情况也较常见。正如我们所知道的，NULL 用于值未知或不适用时。例如，假定知道 ACME 开始销售一种新产品 SPRUCE LUMBER30x40x50x，但这个产品的某些属性（价格和重量）还是未知的。我们可以使用下面的 INSERT 语句在 PRODUCT 表中添加一条记录：

```
INSERT INTO product
(
 prod_id_n,
 prod_num_s,
 prod_description_s,
 prod_status_s,
 prod_brand_s,
 prod_pltwid_n,
 prod_pltlen_n
)
VALUES
(
 991,
 '991',
 'SPRUCE LUMBER 30X40X50',
 'N',
 'SPRUCE LUMBER',
 4,
 6
)
```

不能在<column_name>列中指定 PROD_PRICE_N、PROD_NETWGHT_N 以及 PROD_SHIPWEIGHT_N 列的值，所以，这些值将为 NULL。

下面两个语句将两条记录插入到 PAYMENT_TERMS3 表中（我们会在后面的例子中用到）：

```
INSERT INTO payment_terms3
(
 payterms_id_n,
 payterms_code_s,
 payterms_desc_s,
 payterms_discpct_n,
 payterms_daystopay_n
)
VALUES
(
```


www.TopSage.com

```
 27,
 'N21531',
 '2% 15 NET 30',
 0.02,
 31
)

INSERT INTO payment_terms3
(
 payterms_id_n,
 payterms_code_s,
 payterms_desc_s,
 payterms_discpct_n,
 payterms_daystopay_n
)
VALUES
(
 28,
 'N21530',
 '2% 15 NET 30',
 0.02,
 30
)
```

如果任何被跳过的列有一个默认值，则 RDBMS 将使用这个值，而不是使用 NULL 值：

```
INSERT INTO salesman
(
 salesman_id_n,
 salesman_code_s,
 salesman_name_s
)
VALUES
(
 22,
 '0B',
 'FAIRFIELD MOSQUITO ASSOCIATION'
)
```

因为 SALESMAN 表的 SALESMAN_STATUS_S 列的默认值为 Y，这个被插入的行看起来就像这样：

```
SELECT *
FROM salesman
WHERE salesman_code_s = '0B'

SALESMAN_ID_N SALESMAN_CODE_S SALESMAN_NAME_S
   SALESMAN_STATUS_S
------------ ---------------------------------------------- --------
           22 0B                FAIRFIELD MOSQUITO ASSOCIATION Y
```

这里是使用 CUSTOMER 表的另一个例子：

```
INSERT INTO customer
(
 cust_id_n,
 cust_paytermsid_fn,
 cust_salesmanid_fn,
 cust_name_s,
 cust_alias_s,
 cust_credhold_s
)
VALUES
(
 1,
 27,
 24,
 'WILE SEAL CORP.',
 'MNGA71396',
 'Y'
)
```

Note 注意：如果在读前一章的时候删除了表 CUSTOMER，现在又想运行先前的语句，那么请按照第 4 章的描述重新创建这个表。

因为 CUST_STATUS_S 字段的默认值为 Y，这个列的默认值会被隐式填充：

```
SELECT cust_status_s
FROM customer
WHERE cust_alias_s = 'MNGA71396'

CUST_STATUS_S
-------------
Y
```


www.TopSage.com

2. 为所有的列插入值

假设已具备所有必需的值，并想要填充所有的列，在准备插入一行时，我们会有一个选择。可以列出所有的列名连同所对应的值，也可以将<column_name>子句完全省略。下面的两个语句会产生相同的结果：

```
INSERT INTO product
(
 prod_id_n,
 prod_price_n,
 prod_num_s,
 prod_description_s,
 prod_status_s,
 prod_brand_s,
 prod_pltwid_n,
 prod_pltlen_n,
 prod_netwght_n,
 prod_shipweight_n
)
VALUES
(
 1880,
 33.28,
 '1880',
 'STEEL NAILS 6''''',
 'Y',
 'STEEL NAILS',
 5,
 4,
 38.39148,
 42.39148
)

INSERT INTO product
VALUES
(
 1880,
 33.28,
 '1880',
 'STEEL NAILS 6''''',
 'Y',
 'STEEL NAILS',
 5,
 4,
 38.39148,
 42.39148
)
```

注意，后者的语法要求值完全按照准确的次序排列以匹配列列表。

Tip 提示：尽管第二种语法输入速度更快，但第一种语法却更容易理解和维护，因此，在产品代码中，使用列名会更有意义。

3. 显式插入 NULL 值和默认值

在前面的例子中我们插入的是文本，当某个列名被跳过时，就将 NULL 值填充到这个列中或者以隐式的方式填充这个列的默认值。但是，有的时候可能想要显式将 NULL 值插入到某个列中，或是显式地让 RDBMS 使用这个列的默认值。这可以通过使用相应的关键字 NULL 或 DEFAULT 来实现。下面的语句在新插入的行的 PROD_RPICE_N 列中填充了 NULL 值：

```
INSERT INTO product
(
 prod_id_n,
 prod_price_n,
 prod_num_s,
 prod_description_s,
 prod_status_s,
 prod_brand_s,
 prod_pltwid_n,
 prod_pltlen_n,
 prod_netwght_n,
 prod_shipweight_n
)
VALUES
(
 5786,
```


www.TopSage.com

TopSage.com

```
 NULL,
 '5786',
 'CRATING MATERIAL 12X48X72',
 'Y',
 'CRATING MATERIAL',
 5,
 6,
 20.37674,
 23.37674
)
```

这里是将一行插入到 ORDER_HEADER 表中的另一个语句，并显式地填充 NULL 值到多个列：

```
INSERT INTO order_header
VALUES
(
 30670,
 28,
 NULL,
 1,
 24,
 '523783',
 NULL,
 NULL,
 NULL,
 NULL,
 'N',
 'Y',
 NULL,
 'RR',
 NULL,
)
```

以后，可以使用 UPDATE 语句用实际的数据填充这些 NULL 值字段（发票号、附注、多个日期等）。

此语句在 SALESMAN 表中创建一条新记录，并显式地指定 SALESMAN_STATLU_S 列所使用的是默认值：

```
INSERT INTO salesman
VALUES
(
 24,
 '03',
 'AMERICA GONZALES LIMITED',
 DEFAULT
)
```

4. 插入从其他表中选定的值

插入在其他表中选定的值的一种典型的情形（但是不是唯一的）发生在周期性归档工作中。例如，对于某个大公司而言，一个存储发货信息的表在经历几个较长的时期后会显著地增长。假定某个公司有这样一条商务规则，就是按照常规一般不需要 180 天以前的任何发货信息，除了由于一些的特殊情况偶尔需要旧的发货数据，比如说审计。为了提高性能，我们可以创建表 SHIPMENT_ARCHIVE，这个表的列与 SHIPMENT（参阅第 4 章获得详细信息）的列完全一样。每月将表 SHIPMENT 中存储日期超过 180 天的所有信息行插入到表 SHIPMENT_ARCHIVE 中。然后，使用 DELETE 语句（本章的后面将会讨论）将旧的记录从 SHIPMENT 表中删除。这条语句用 Oracle 语法归档所有存储日期超过 180 天的发货记录：

```
INSERT INTO shipment_archive
SELECT *
FROM shipment
WHERE TRUNC(shipment_createdate_d) < TRUNC(SYSDATE) - 180;
```

Note 注意：DB2 和 Microsoft SQL Server 使用自带的方法和函数处理日期。参阅第 10 章获得详细信息。

5. INSERT 语句和完整性约束

按照某些明确的规则和约束将行插入到一个表中。举例来说，所有的列值必须与对应的列定义具有相同或至少兼容的数据类型和大小。有一些特定于实现的变体，例如，只要有可能，Oracle 就会执行隐式的转换（从字符串数据类型到数字类型，从日期型到字符串型等），而在 DB2 中，就不得不显式地将值转换为一种兼容的数据类型。但 RDBMS 通常不允许将客户名字插入到数字或日期列。这会产生一个错误，而且整个 INSERT 语句会失效。请参阅第 3 章获得更多的相关信息。

www.TopSage.com

Cross-Reference 交叉参考：Microsoft SQL Server 2008 中的 SQL_VARIANT 数据类型允许数据库对象支持其他的数据类型；也就是说，在不同的行中，一个表列的数据类型可以包含数值型、字符串型、日期型以及二进制数值型。

当执行某个基于子查询的插入操作时出现的“一个坏行”问题是很典型的（通常是非常令人讨厌的）。设想一下这种情形，当必须从多个表中选择出一万行，每行二十列，并将得到的结果插入到目的表中时，只要某一行中有一个值比列定义允许的值大就会导致整个插入操作失败。例如，如果目的表列是 NUMERIC（5）型，那么，尝试将任意大于 99 999 的值插入到该列的操作都将失败，即使在这个 INSERT 操作之前可能执行了两个多小时，整个 INSERT 语句也会因此失败。

在试图插入某个违反了约束完整性的值时也会出现类似的问题。如不能在 NOT NULL 列中插入 NULL 值；对于 UNIQUE 和 PRIMARY KEY 列，重复的值会被拒绝，等等。举个例子，由于违反 CHECK 约束，执行下面的语句将会失败：

```
INSERT INTO salesman
(
  salesman_id_n,
  salesman_code_s,
  salesman_name_s,
  salesman_status_s
)
VALUES
(
 26,
 '07',
 'ELMERSON INDUSTRIES INCORPORATED',
 'A'
)
```

在 SALESMAN_STATUS_S 列上的 CHECK 约束表明此列只有两个有效值：Y 和 N，可现在正在插入 A。要实现这个语句，只需把它改为 Y 即可。

New Feature 新增功能：从 10gR2 版本开始，Oracle 提供 DML 错误日志属性。如果将错误日志子句加入到 INSERT、UPDATE、DELETE 或者 MERGE 语句中，那么某些类型错误就不再终止语句的执行。产生的错误以及所影响到的列值将被登记到错误日志表中，而语句仍然可以继续执行。在错误日志表中的内容可以被分析，并且，会在以后修复这些错误。在使用 LOG ERRORS 子句之前，必须先创建错误日志表，可以通过手动的方式也可以使用 DBMS_ERRLOG 系统包来创建。举例说明，下面的语句为 PRODUCT 表创建名为 PROD_ERRLOG 的错误日志表。

```
EXECUTE DBMS_ERRLOG.CREATE_ERROR_LOG('PRODUCT','PROD_ERRLOG');
```

Tip 提示：避免由于单行错误所导致的整个 DML 语句被拒绝的一般做法是使用 SQL 程序扩展——Orale Pl/SQL，Microsoft Server T-SQL，以及 DB2 SQL PL。请仔细阅读第 14 章。

RDBMS Error Messages

每个开发商都有一套专有的针对于某些事件的错误消息。就用词来讲，彼此各不相同，但是所有这三大开发商都提到了违反约束的名字。

Oracle

ORA-02290: 违反了检查约束(ACME.CHK_SALESSTATUS)

DB2

SQL0545N 由于有一行不满足检查约束性“ACME.SALESMAN.CHK_SALESSTATUS”，所以请求的操作被拒绝。

Microsoft SQL Server

INSERT 语句与列 CHECK 约束 ‘CHK_SALESSTATUS’ 冲突。

如果约束名为 **descriptive** 也就是说，如果很快就告诉是哪个表中的列出了问题，就可以更快、更容易的识别和处理问题。

6.1.2 供应商相关的特定 INSERT 语句

在本章一开始就给出了一般的 INSERT 语句语法，它包括了一些最重要的子句，这些子句对“三

www.TopSage.com

大”RDBMS 实现以及 SQL:2003 语法而言是公共的这一节我们将讨论一些供应商特定差异。

1. SQL:2003

一般语法和 SQL9:2003 标准的语法间的主要不同是 SQL:2003 多了一个 DEFAULT VALUES 子句：

```
INSERT INTO <table_or_view_name>
[(<column_name>,...)]
{ {VALUES {<literal> |
          <expression> |
          NULL,...)} |
  {<select_statement>} |
  {DEFAULT VALUES}}
```

DEFAULT VALUES 子句将在本章后面解释。

2. Oracle 11g

Oracle 中的 INSERT 语句有多个备选子句。举个例子，它可以返回行并可以将其存储在预先声明的变量中（UPDATE 以及 DELETE 也有这个功能）。也可以执行多个条件插入（就是说，在一个 INSERT 语句中，插入返回值所派生的值到多个表中）——例如，满足 **condition1** 的行插入到 TABLE1 中，符合 **condition2** 的行插入到 TABLE2 中，等等。可以在 INSERT 语句中使用序列以及 DML 错误日志（如前面提到的）等。

经过简化的 Oracle 11g INSERT 语句的语法是：

```
INSERT
{ INTO <table_or_view_name>
  [(<column_name>,...)]
  {
    VALUES (<literal> |
             <expression> |
             NULL |
             DEFAULT),... |
    <select_statement>
  }
  [RETURNING <expression>,.. INTO <data_item>] |
  ALL INTO <table_or_view_name>
  [(<column_name>,...)]
  {
    VALUES (<literal> |
            <expression> |
            NULL |
            DEFAULT),...
  },... |
  {ALL | FIRST} WHEN <condition> THEN
       <insert_into_clause> [<values_clause>]
  [ ELSE <insert_into_clause> [<values_clause>]]
[LOG ERRORS [[INTO <log_error_table>] [REJECT LIMIT (n)]]]
```

下例显示使用序列插入一行到 ADDRESS 表：

```
CREATE SEQUENCE seq_addr;

INSERT INTO address
(
 addr_id_n,
 addr_custid_fn,
 addr_salesmanid_fn,
 addr_address_s,
 addr_type_s,
 addr_city_s,
 addr_state_s,
 addr_zip_s,
 addr_country_s
)
VALUES
(
 SEQ_ADDR.NEXTVAL,
 NULL,
 23,
 '223 E FLAGLER ST.',
 NULL,
 'MIAMI',
 'FL',
 '33131',
 'USA'
);
```


www.TopSage.com

大家网 TopSage.com

交叉参考：序列在第4章讨论。

下面的例子显示使用单独的 INSERT 语句插入多行：

```
CREATE TABLE status
(
    status_id_n         NUMBER        NOT NULL,
    status_code_s       CHAR(2),
    status_desc_s       VARCHAR2(30),
    CONSTRAINT pk_statusprim PRIMARY KEY (status_id_n)
)

Table created.

INSERT ALL
INTO status (status_id_n, status_code_s, status_desc_s)
VALUES ( 2, '20', 'COMPLETE')
INTO status (status_id_n, status_code_s, status_desc_s)
VALUES ( 6, '60', 'SHIPPED')
INTO status (status_id_n, status_code_s, status_desc_s)
VALUES ( 8, '70', 'INVOICED')
INTO status (status_id_n, status_code_s, status_desc_s)
VALUES ( 9, '80', 'CANCELLED')
SELECT * FROM dual

4 rows created.
```

最后一行的 SELECT 关键字是必要的，虽然在本例中没有太大意义。我们已经知道，DUAL 是一个特殊的表，当语法需要 SELECT 子句时使用这个表。

有时，需要根据某些列值有条件地插入到不同的表。下面显示的是根据产品的运输重量有条件的插入的例子。

```
-- There are 3 rows in product table

SELECT prod_id_n, prod_description_s, prod_shipweight_n
FROM product;

 PROD_ID_N PROD_DESCRIPTION_S                       PROD_SHIPWEIGHT_N
---------- ---------------------------------------- -----------------
       991 SPRUCE LUMBER 30X40X50
      1880 STEEL NAILS 6''                                      42.391
      5786 CRATING MATERIAL 12X48X72                            23.377

-- Create 3 empty tables as copies of PRODUCT (WHERE evaluates
   to FALSE)

CREATE TABLE small_product AS
SELECT * FROM product
WHERE prod_id_n < 0;

CREATE TABLE large_product AS
SELECT * FROM product
WHERE prod_id_n < 0;

CREATE TABLE undetermined_product AS
SELECT * FROM product
WHERE prod_id_n < 0;

-- Perform conditional insert.

INSERT ALL
  WHEN prod_shipweight_n <= 30 THEN
     INTO small_product
  WHEN prod_shipweight_n > 30 THEN
     INTO large_product
  ELSE
     INTO undetermined_product
  SELECT * FROM product;

3 rows created.

SELECT prod_id_n, prod_description_s, prod_shipweight_n
FROM small_product;

PROD_ID_N PROD_DESCRIPTION_S                       PROD_SHIPWEIGHT_N
---------- ---------------------------------------- -----------------
      5786 CRATING MATERIAL 12X48X72                            23.377
```


www.TopSage.com

```
SELECT prod_id_n, prod_description_s, prod_shipweight_n
FROM large_product;

 PROD_ID_N PROD_DESCRIPTION_S                          PROD_SHIPWEIGHT_N
---------- ---------------------------------------- -----------------
      1880 STEEL NAILS 6''                                       42.391

SELECT prod_id_n, prod_description_s, prod_shipweight_n
FROM undetermined_product;

PROD_ID_N PROD_DESCRIPTION_S                          PROD_SHIPWEIGHT_N
---------- ---------------------------------------- -----------------
       991 SPRUCE LUMBER 30X40X50
```

注意产品号为 991（SPRUCE LUMBER 30×PRUCE）的 prod_shipweight_n 的值为 NULL，那么就不满足“小于等于 30”或者“大于 30”的条件，就由可选 ELSE 子句处理。

3. DB2 9.5

DB2 的 INSERT 语法与本章开始的一般语法定义没有多大不同：

```
INSERT INTO <table_or_view_name>
[(<column_name>,...)]
 {
  {VALUES (<literal> |
           <expression> |
           NULL),...},.. |
  <select_statement>
 }
```

下面的语句一次将 4 行记录插入到表 STATUS 中：

```
CREATE TABLE STATUS(
    STATUS_ID_N         INTEGER            NOT NULL,
    STATUS_CODE_S       CHAR(2),
    STATUS_DESC_S       VARCHAR(30),
    CONSTRAINT PK_STATUSPRIM PRIMARY KEY (STATUS_ID_N)
)

INSERT INTO status
(
  status_id_n,
  status_code_s,
  status_desc_s
)
VALUES
 ( 2, '20', 'COMPLETE'),
 ( 6, '60', 'SHIPPED'),
 ( 8, '70', 'INVOICED'),
 ( 9, '80', 'CANCELLED')

SELECT * FROM status

STATUS_ID_N STATUS_CODE_S STATUS_DESC_S
----------- ------------- ------------------------------
          2 20            COMPLETE
          6 60            SHIPPED
          8 70            INVOICED
          9 80            CANCELLED

  4 record(s) selected.
```

Note 注意：这个例子也可以使用 Microsoft SQL Server 2008 来做。Oracle 中没有使用单独的 VALUES 子句插入多个行的选项，但是我们可以通过使用上一节的 INSERT ALL 模拟这个行为。

也可以使用 INSERT 语句中的序列（语法与 Oracle 的相比略有不同）。

```
CREATE SEQUENCE seq_addr

INSERT INTO address
(
 addr_id_n,
 addr_custid_fn,
 addr_salesmanid_fn,
 addr_address_s,
 addr_type_s,
 addr_city_s,
 addr_state_s,
 addr_zip_s,
```

```
 addr_country_s
)
VALUES
(
 NEXTVAL FOR SEQ_ADDR,
 NULL,
 23,
 '223 E FLAGLER ST.',
 NULL,
 'MIAMI',
 'FL',
 '33131',
 'USA'
)
```

要提到的最后一项是标识列。当将某个行插入到含有一个标识列的表中时，DB2为这个列产生一个值。为 GENERATED ALWAYS 标识列产生这个值——事实上，如果尝试显式地插入某个值（除 DEFAULT 以外）到这个列中，将产生一个错误。对于列 GENERATED BY DEFAULT，只要它不是由 VALUES 子句或者子查询显式指定，DB2就会产生一个值。

4. Microsoft SQL Server 2008

除了一般的 INSERT 语法子句外，Microsoft SQL Server 还允许在标识列使用 DEFAULT VALUES 子句以及系统生成值或用户生成值。支持在单一 VALUES 子句中插入多个行的操作：

```
INSERT INTO <table_or_view_name>
[(<column_name>,...)]
{
  {VALUES (<literal> |
           <expression> |
           NULL),...},... |
  {<select_statement>} |
  {DEFAULT VALUES}
}
```

DEFAULT VALUES 子句可用于向表内插入值，当没有为列指定值时（DEFAULT，IDENTITY，NULL 或 TIMESTAMP），将插入一个值。换句话说，DEFAULT VALUES 选项用于添加行，但不提供明确的值：

```
CREATE TABLE defaults
(
  c1 int identity,
  c2 varchar(30) DEFAULT ('column default'),
  c3 timestamp,
  c4 int NULL
)

INSERT INTO defaults
 DEFAULT VALUES

 SELECT *
 FROM defaults

(1 row affected)
c1          c2              c3                  c4
----------- --------------- ------------------- ------------
          1 column default 0x00000000000002A9         NULL

(1 row affected)
```

试图显式地将用户生成的值插入标识列将会产生一个错误，除非使用 Transact-SQL 语法覆盖掉这个列的标识特性。同样，在这个例子中，必须在 INSERT 语句中使用<column_name>列表。

下面的例子阐述将 PAYTERMS_ID_N 作为标识列创建表 PAYMENT_TERMS：

```
-- Trying to insert with IDENTITY_INSERT option off fails
INSERT INTO payment_terms
(
 payterms_id_n,
 payterms_code_s,
 payterms_desc_s,
 payterms_discpct_n,
 payterms_daystopay_n
)
VALUES
 (
  26,
  'N30',
```


www.TopSage.com

TopSage.com

```
  'NET 30',
  0,
  30
)

Msg 544, Level 16, State 1, Server PD-TS-TEST1, Line 1 Cannot insert
   explicit value for identity column in table
'payment_terms' when IDENTITY_INSERT is set to OFF.
-- Set IDENTITY_INSERT option on; insert succeeds.
SET IDENTITY_INSERT payment_terms ON

INSERT INTO payment_terms
(
 payterms_id_n,
 payterms_code_s,
 payterms_desc_s,
 payterms_discpct_n,
 payterms_daystopay_n
)
VALUES
 (
  26,
  'N30',
  'NET 30',
  0,
  30
 )

(1 row affected)
```

6.2 UPDATE：修改表数据

UPDATE 语句用来修改现有数据库信息。本节将讲述两种需要修改数据的一般情况。

一种情况是向表内插入行，却不知道所有的信息（这时 NULL 很有用），过后，当这些信息可用时，可以更新相应的行。举个例子，在知道谁将成为某个新客户的销售人员之前，可能想要先创建这个新客户，或是生成一个不知道发货数量的新订单行记录（在这次订单实际发货之前无法知道这批货物的数量）。

另一种情况是当必须反映出“现实世界”的某些改变时也就是可能需要更新数据库信息的时候。例如，客户可以取消订单，因此必须把这个订单的状态从 COMPLETE 改为 CANCELLED；客户可能积累了“坏账”，因此需要暂停他的信用（在 ACME 数据库中，这意味着将 CUST_CREDHOLD_S 字段值改为 Y）。

UPDATE 语句用于修改表数据。与先前讲述的 INSERT 语句一样，可以直接在表上完成也可以通过可更新的视图来完成。这里是一般的 SQL:2003 语法：

```
UPDATE <table_or_view_name>
SET <column_name> = { <literal> |
                      <expression> |
                      (<single_row_select_statement>) |
                      NULL |
                      DEFAULT
                    },...
[WHERE <predicate>]
```

UPDATE 语句允许一次更新一个表。除此之外，UPDATE 语句为正在更新值的集合提供了很大的灵活性。可以更新一行或多行中的一列或多列，或者（尽管极少使用）也可以更新表中所有行的每一列。UPDATE 语句的不同子句决定了这种粒度。

Note 注意：由于表中没有满足 WHERE 子句条件的行而没有行被更新的情况，RDBMS 不当作一个错误。DELETE 语句（将在接下来的部分中进行讨论）也与此一样。当没有满足 WHERE 子句的行时，就不会更新行，也不会产生错误。

接下来讨论一般 UPDATE 语句的最通用子句。

6.2.1 一般的 UPDATE 语句子句

可以使用在<table_or_name>子句中所提供的功能更新表（或可更新的视图）的名称。

在 SET 子句中，指定要更新的列的名称和要指派给它的新值。也可以指定用逗号分隔的多个对应

www.TopSage.com

的列/值对。指派的值可以与 INSERT 语句中的 VALUES 子句中指定的相同(就是说，值可以是字面值、表达式、NULL、默认值等等)。

WHERE 子句设置“水平”限制，在 SET 子句中指定了要更新的列，现在必须定义一个条件，在这个条件下，一些行将被更新。

WHERE 子句的危险性

WHERE 子句是非常重要的，必须小心地对待。如果意外地越过它，或是将它建立在某个不正确的假设上，将会用相同的值更新表中的所有行，这通常不是我们想要的。例如，要把产品 1880(STEEL NAILS6") 的价格从 33.28 美元更改为 34.76 美元。您可能认为下述语句可行:

```
UPDATE product
SET prod_price_n = 34.76
```

由于忘记添加 WHERE 子句，产生了灾难性的结果，这将导致 ACME 数据库中所有产品现在的价格都是 34.76 美元!

正确语法是:

```
UPDATE product
SET pro_price_n = 34.76
WHERE prod_num_s = '1880'
```

注意，即使记得输入了 WHERE 子句，但只是输入出错，所产生的结果也不会好多少。比如说，如果输入的是>而不是=，那么所有产品号大于 1880 的产品都将被错误地更新。

1. 更新一行中的一列

一个最平常的更新情形是要修改只有单行的某一列。例如，将某个销售人员指派给一个新近创建的客户，取消某个订单或改变某个产品的价格，这些例子都属于这种情形。下面的例子是将某个价格指派给产品 990，这个产品在本章前面部分创建过。

```
UPDATE product
SET prod_price_n = 18.24
WHERE prod_id_n = 990
```

Tip 提示: 在 UPDATE 语句 WHERE 子句中使用主键或带有 UNIQUE 约束的列可以确保只更新由该列的值唯一标识的一行。

2. 更新多列

有时，可能要在一个 UPDATE 语句中更新多列。例如，厂商已经换了产品 990 包装，尺寸由 4×6 尺寸变成 5×7。下面这个 UPDATE 语句将使数据库信息与现实世界的改变保持同步:

```
UPDATE product
SET prod_pltwid_n = 5,
    prod_pltlen_n = 7
WHERE prod_id_n = 990
```

3. 更新所有行中的某一列

尽管更新所有的表行并不常见，但有时可能要执行这种操作。比如，给所有员工增加 5%的工资，暂停所有客户的活动，把所有列值设置为 NULL。需要注意的是，关键词是“所有”。换言之，仅当我们要更新目标表中的每一行时，才可以忽略 WHERE 子句。

下面的 UPDATE 语句将所有产品的价格增加了 10%。

```
UPDATE product
SET pro_price_n = prod_price_n * 1.1
```

Cross-Reference 交叉参考: SQL 查询中运算符的使用将在第 11 章讨论。

4. 使用单行子查询更新列

可以用 SELECT 语句（子查询）得到的结果在 UPDATE 语句中赋值。一定要记住的是，子查询返回的行数不能多于一行（如果没有返回行，则将目标列赋值 NULL)。

可以在其上连接两个或多个列，或者执行数学操作，但是不能列举多个由逗号分隔的列。因此，设置 my_col = (SELECT col1 + col2 …) 是有效的，而 my_col = (SELECT col1 , col2 …) 则是无效的。

Note 注意: Oracle 和 DB2 允许在一个单行子查询的选择列表中指定多个值，本章后面将会讲到。

www.TopSage.com

由另一个值派生为这个赋值

在许多情况下，使用子查询赋值是有益处的。举个例子，想要将 ACME 数据库中订单 30670 的付款项改为 N21531。问题是，在简单的 ACME 关系型数据库中并没有将实际的付款号 N21531 存到 ORDER_HEADER 表列当中；而是用外键作为代号，这个代号是 PAYMENT_TERMS 表中的一个无意义的整数。然而，可以使用下面的子查询完成这个任务：

```
UPDATE order_header
SET ordhdr_payterms_fn =
      (SELECT payterms_id_n
       FROM payment_terms
       WHERE payterms_code_s = 'N21531')
WHERE ordhdr_id_n = 30670
```

上面的语句有两个 WHERE 子句，但是不会发生混淆：第一个属于 SELECT 语句——圆括号中的，用于将结果集限定为一个值——PAYTERMS_CODE_S 列中值 N21531 所在行的主键；属于 UPDATE 语句的第二个WHERE子句确保了表ORDER_HEADER中仅有等于30670的ORDHDR_ID_N的一行被更新。

ORDER_HEADER

ORDHDR_ID_N	ORDHDR_PAYTERMS_FN
...	...
30670	?
...	...

PAYMENT_TERMS

PAYTERMS_ID_N	PAYTERMS_CODE_S
...	...
27	N21531
...	...

图 6-1 使用来自其他表的数据赋值

用相关子查询来更新

前面的例子相对简单，就是通过一个给定值派生所需要的值用于更新。但有时条件会很复杂。举个例子，设想 ACME 的公司规定发生了改变，在客户进行订购时，不再允许与客户默认付款条件不同的付款项存在。可以将一个来自 UPDATE 语句的表和在赋值子查询中指定的表关联（联接）起来，这几乎可以超越“单行”规则了，因为赋值是逐行进行的。

```
UPDATE order_header
SET ordhdr_payterms_fn =
    (SELECT payterms_id_n
     FROM   payment_terms,
            customer
     WHERE  payterms_id_n = cust_paytermsid_fn
     AND    ordhdr_custid_fn = cust_id_n)
```

Note 注意：如果按照接下来的第 5 章中 DROP TABLE 一节的例子删除了表，需要使用在第 4 章中所描述的语法重新建立 CUSTOMER 表。

上面语法的最后一行将 ORDER_HEADER 表（UPDATE 语句）的 ORDHDR_CUSTID_FN 字段与 CUSTOMER 表的 CUST_ID_N 字段联接起来（嵌套子查询）；换言之，客户的 ID 字段是 UPDATE 语句和子查询之间的链接，这个链接将二者联系起来。

Note 注意：由于在 ACME 数据库创建时就使用了特殊符号规则，在这里，不必对表名使用限定性条件。每一个列的前缀均是表名缩写，因此不存在二义性的列名。

Cross-Reference 交叉参考：关于表的别名以及相关子查询将在第 8 章和第 9 章中详细讨论。

对于 ORDER_HEADER 表中的每一行，必须在子查询结果集中找到与其对应的值并给它赋值。图 6-2 表明了这个概念。

Note 注意：此语法使用“旧的”联接声明，“三大”数据库都认识到了这一点。之所以使用这一语法是为了更好地阐述相关查询的概念。

Tip 提示：子查询也可以用在 UPDATE 语句的 WHERE 子句中，这与 SELECT 语句中的 WHERE 子句非常相似。

5. UPDATE 语句和完整性约束

更新表行所遵循的规则和约束与 INSERT 语句的相同。所有的列值必须具有相同的或者兼容的数据类型和尺寸，以及一致的列定义，并且不能违反完整性约束。引用完整性约束有细微的差别，当用 ON UPDATE CASCADE 或 ON UPDATE SET NULL 指定约束时，RDBMS 会成功地更新一个目标表，被引用表的列也会随着相应值的更新而更新。

www.TopSage.com

大家网 TopSage.com

ORDER_HEADER

ORDHDR_ID_N	ORDHDR_CUSTID_FN	ORDHDR_PAYTERMS_FN
...	...	...
30618	152	?
30619	152	?
30620	7	?
...	...	...

CUSTOMER

CUST_ID_N	CUST_PAYTERMSID_FN
...	...
7	42
152	27
...	...

PAYMENT_TERMS

PAYTERMS_ID_N	PAYTERMS_CODE_S
...	...
27	N21531
42	N120
...	...

图 6-2 用相关子查询更新多个行

交叉参考：约束在第 4 章进行了讨论。

6.2.2 供应商特定的 UPDATE 语句细节

像 INSERT 语句一样，UPDATE 也有一些供应商特定功能。这个部分简明地讨论其中最重要的功能。

1. SQL:2003

基本的 SQL:2003 的 UPDATE 语法如下：

```
UPDATE <table_or_view_name> [[AS] <correlation_name>]
SET { <column_name> = { <literal> |
                        <expression> |
                        (<single_row_subquery>) |
                        NULL |
                        DEFAULT
                        },... |
      <multiple_column_assignment> }
[WHERE <predicate>]
```

2. Oracle 11g

Oracle 的 UPDATE 语法与一般语法的主要区别在于，在 Oracle 当中，可以在 SET 子句中指定多个用括号括起并用逗号分隔的列；并可以提供返回多个对应的值的一个查询，这些值位于等号的右侧（请见本节后面的例子）。同样，也可以使用 DML 错误日志返回受影响的值并将他们存储在主机变量：

```
UPDATE <table_or_view_name> [[AS] <correlation_name>]
SET { <column_name> = { <literal> |
                        <expression> |
                        (<single_row_select_statement>) |
                        NULL |
                        DEFAULT
                        },... |
     (<column_name1> [, <column_name2>,..]) = (<select_statement>) }
[WHERE <predicate>]
[RETURNING <expression>,.. INTO <data_item>]
[LOG ERRORS [[INTO <log_error_table>] [REJECT LIMIT (n)]]]
```

下例显示了使用从表 PRODUCT 中查询到匹配的值，再用对应的值更新表 SMALL_PRODUCT 的多个列的情况：

```
UPDATE small_product
SET (prod_pltwid_n, prod_pltlen_n) = (SELECT prod_pltwid<?xmltex_n,
 prod_pltlen_n
                                       FROM product
                                       WHERE prod_id_n = 5786)
WHERE prod_id_n = 5786
```

子查询仍必须返回不能多于一行的结果。

3. DB2 9.5

DB2 的 UPDATE 语法如下：

```
UPDATE <table_or_view_name> [[AS] <correlation_name>]
SET { <column_name> = { <literal> |
                        <expression> |
                        (<single_row_select_statement>) |
                        NULL |
                        DEFAULT
```


www.TopSage.com

```
                    },.. |
    (<column_name1> [, <column_name2>,...]) =
                          ({<select_statement> |
                            <value1> [, <value2>,..]})}
[WHERE <predicate>]
```

可以使用适当形式的子查询所返回的值更新多个列，就像在 Oracle 中一样：

```
-- The first two statements are necessary for this example to work
CREATE TABLE small_product AS
(SELECT * FROM PRODUCT)
DEFINITION ONLY

INSERT INTO small_product
(SELECT * FROM product
 WHERE prod_shipweight_n < 30)

UPDATE small_product
SET (prod_pltwid_n, prod_pltlen_n) = (SELECT
   prod_pltwid_n, prod_pltlen_n
                                      FROM product
                                      WHERE prod_id_n = 5786)
WHERE prod_id_n = 5786
```

另外，可以提供一系列值来代替 SET 子句右边的子查询：

```
UPDATE small_product
SET(prod_pltwid_n,prod_pltlen_n)  = (55,99)
WHERE prod_id_n = 5786
```

4. Microsoft SQL Server 2008

在 Microsoft SQL Server 中，UPDATE 语句有一个可选的 FROM 子句。它指明了为更新操作提供条件所使用的表，与此非常类似，它可以用于以前所讨论过的相关查询。下面的例子实际上执行了与前面部分中的相关查询相同的任务，唯一的不同是在使用前面的语法时，ORDER_HEADER 表的每一行都将无条件地被更新，而 Microsoft SQL Server 的语法会将值为 NULL 的列排除在外，除非使用了 OUTER JOIN（有关连接的更多内容在第 9 章讲述）：

```
UPDATE order_header
SET    ordhdr_payterms_fn = payterms_id_n
FROM   payment_terms JOIN  customer
ON     payterms_id_n = cust_paytermsid_fn JOIN order_header
ON     ordhdr_custid_fn = cust_id_n
```

6.3 DELETE：从表中删除数据

自从引入计算机化的数据库以来，人类已经积累起上 PB 计的数据。数据过量已经成为事实，我们正在被数据淹没，还好，DELETE 提供了一种可以去除不再需要的信息的方式。当数据库中的某些行对应的实体与现实世界不再相关或在现实世界完全消失时，就必须在数据库中删除这些行。举个例子，一个员工离职，一个客户不再发出任何订单，或者不再需要发货信息了。

Note

注意：许多情况下，并不是立即就能从数据库中删除这些实体信息。举个例子，当一个员工离职时，人力资源部依然保留着关于这个员工的历史信息；当一个客户停止与公司的活动已经很长一段时间了，有些时候，“停止”他的活动比删除他会更合理一些。因为在关系数据库中，通常删除一个客户所涉及到的事情远远不只是删除 CUSTOMER 表的记录那么简单，通常在这个表上会有来自其他表的引用完整性约束，合乎规则的其他表又会被更多的表所引用等等。简化这一过程的方式是使用 ON DELETE CASCADE 子句，但结果并不总是符合要求。即使当不再需要某个客户的信息，可能还需要访问客户的订单和发票信息，等等。

可能需要删除插入错误的行，例如，一个订单被插入了两次，或者重复的客户记录被创建。这些情况经常出现，尤其在大公司中，会有几十名职员接受订单并创建新客户的情形。

Tip

提示：好的数据库设计可以减少人为错误的数量。例如，把唯一性约束放置在客户名字字段上有助于防止出现重复的客户（尽管不是万无一失，因为 RDBMS 会把“ALME, INC”和“ACME INC”视为两个不同的字符串）。

DELETE 语句在某个单一的表中删除行（可以是直接的，也可以通过可更新的视图）。其语法是：

```
DELETE FROM <table_or_view_name>
[WHERE <predicate>]
```

注意：FROM 关键字在 Oracle 和 Microsoft SQL Server 中是可选的，但在 DB2 语法中是必需的，并且服从 SQL:2003 标准。

DELETE 执行从一个表中删除行的操作。可以使用一个 DELETE 语句删除一行或多行；当表中不存在满足 WHERE 子句指定条件的行时，就没有行会被删除，也不会产生错误消息。

6.3.1 一般的 DELETE 语句子句

DELETE 语句有可能是所有 DML 语句中最简单的。只需指定表和（可选择的）和删除行的条件。语法简单并不表示语句也简单，DELETE 语句可以相当复杂，必须谨慎对待。如果 WHERE 子句被省略或格式搞错，一些有价值的信息就会从目标表中被删除。往往会出现我们不希望的结果，而数据恢复过程是痛苦和耗时的。

下面的语句从 SALESMAN 表中删除一条销售人员记录：

```
DELETE FROM salesman
WHERE salesman_code_s = '02'
```

这个语句从 PHONE 表中删除所有记录：

```
DELETE FROM phone
```

1. DELETE 语句和完整性约束

DELETE 语句的完整性约束没有 INSERT 和 UPDATE 那么严格。PRIMARY KEY，UNIQUE，NOT NULL 约束都不能阻止删除一行。引用完整性约束的情况不同，如果某一行中存在一个被其他列引用的列的话，就不能删除这一行，除非引用完整性约束上有 ON DELETE CASCADE 选项（SQL:2003 标准，已被所有“三大” RDBMS 开发商实现）。在这种情况下，DELETE 将成功；表中引用这个被约束列的所有行也都将会删除。这样做极其危险，尤其是与构造较差的 WHERE 子句结合使用时，所以使用 ON DELETE CASCADE 要谨慎。设想这样一种情形，表 CUSTOMER 被 ORDER 表引用，而 ORDER 表又引用 SHIPMENT 表。如果 ON CASCADE DELETE 用在这两个关系中，则当删除一条客户记录时，所有相关的订单和发货信息也将丢失。如果此时省略了 WHERE 子句，后果不堪设想。图 6-3 表明了这个例子。

DELETE FROM customer

CUSTOMER

CUST_ID	NAME	ADDRESS
...	...	...
1023	ACME, INC.	
...	...	...

ORDER

ORD_ID	CUST_ID_FK	...
...	...	...
100234	1023	...
100345	1023	...
...	...	...

SHIPMENT

SHIP_ID	ORD_ID_FK	...
...	...	...
200345	100234	...
200346	100234	...
200577	100345	...
200578	100345	...
200579	100345	...
...	...	...

图 6-3 从由 ON DELETE CASCADE 约束引用的表中删除

另一个（危险性较小）引用约束选项是 ON DELETE SET NULL（SQL:2003 标准，已被 Oracle 和 DB2 实现）。该选项不会从引用表中删除记录，但外键列的值将被设置为 NULL，如图 6-4 所示。

DELETE FROM customer

CUSTOMER

CUST_ID	NAME	ADDRESS
...	...	...
1023	ACME, INC.	
...	...	...

ORDER

ORD_ID	CUST_ID_FK	...
...	...	...
100234	NULL	...
100345	NULL	...
...	...	...

SHIPMENT

SHIP_ID	ORD_ID_FK	...
...	...	...
200345	100234	...
200346	100234	...
200577	100345	...
200578	100345	...
200579	100345	...
...	...	...

图 6-4 从由 ON DELETE SET NULL 约束引用的表中删除

如果带有默认（NO ACTION）选项的列上存在引用完整性约束，并且这个列被引用，DELETE 将会失败。

www.TopSage.com

TopSage.com

2. 在 DELETE 语句 WHERE 子句中使用子查询

除了比较运算符、字面值及表达式外，DELETE 语句中的 WHERE 子句可以包含子查询，允许基于其他表的数据删除选择的行。这与关于插入值的 SET 子句部分所描述的非常相似，使用某个子查询，可以按照某些已知值来派生值。比如，希望删除客户 WILE SEAL CORP.的所有订单，但在不知道主键的值（在表 ORDER_HEADER 中是外键），就可以在 WHERE 子句中使用适当的子查询来完成这个任务：

```
DELETE FROM order_header
WHERE ordhdr_custid_fn =
                        (SELECT cust_id_n
                         FROM customer
                         WHERE cust_name_s = 'WILE SEAL CORP.')
```

Tip 提示：相关子查询也可以和 DELETE 一起使用，所使用的方式在很大程度上与本章开始时讨论的 UPDATE 语句相似。

6.3.2 供应商特定 DELETE 语句子句

开发商特定 DELETE 语句的差别不是很大。在这里做简单的介绍。

1. Oracle 11g

Oracle 与一般 DELETE 语法唯一的差别是 Oracle 的 FROM 关键字是可选的，并且可以省略。举个例子，可以使用 DELETE FROM <table_name>，也可以在代码中简单的使用 DELETE <table_name>。

2. Microsoft SQL Server 2008

Microsoft SQL Server 可以识别 WHERE 子句中使用的子查询语法；另外，提供了 FROM 子句语法，并且具有不同的含义。下面是与“标准”语法等价的一般语法，用于删除客户 WILE SEAL CORP.的所有订单：

```
DELETE order_header
FROM order_header JOIN customer
ON ordhdr_custid_fn = cust_id_n
WHERE cust_name_s = 'WILE SEAL CORP.'
```

6.4 MERGE：将 INSERT、UPDATE 以及 DELETE 结合到一个语句中

MERGE 语句是一个新的概念，首先由 Oracle 9i 引入，后来成为 SQL:2003 标准的一部分；并且，现在得到“三大”RDBMS 供应商的支持。可以将它想成是 INSERT、UPDATE 以及 DELETE 的结合。如果在表中还没有这一行，MERGE 就会插入一行；如果先前已经插入了这个目标行，就可以由它来更新基于给定条件的标识列。Oracle、DB2 以及 SQL Server 也允许有选择地删除没有匹配资源行的目标行。

两个用于操作表数据的供应商特有的 SQL 语句是 MERGE 和 TRUNCATE。

6.4.1 一般的 MERGE 语句子句

用于 MERGE 语句的一般语法如下：

```
MERGE INTO [<qualifier>.]<target_table>
USING [<qualifier>.]<source_table> ON (<condition>)
WHEN MATCHED THEN
UPDATE SET {<column> = {<expression> | DEFAULT},...}
WHEN NOT MATCHED THEN
INSERT [(<column>,...)] VALUES (<expression> | DEFAULT),...;
```

这个语句在许多情况下是有实际意义的。例如，想像 ACME，INC.有一个中心数据库，并且对于每一个仓库有一个本地数据库。每一个位置都有自己的 LOCAL_PRODUCT 表，这些表在结构上是 PRODUCT 表的副本：

```
CREATE TABLE LOCAL_PRODUCT
(
  PROD_ID_N             NUMBER             NOT NULL,
  PROD_PRICE_N          NUMBER,
  PROD_NUM_S            VARCHAR (10),
  PROD_DESCRIPTION_S    VARCHAR (44)       NOT NULL,
  PROD_STATUS_S         CHAR (1)            DEFAULT 'Y',
```


www.TopSage.com

```
PROD_BRAND_S          VARCHAR (20)       NOT NULL,
PROD_PLTWID_N         NUMBER             NOT NULL,
PROD_PLTLEN_N         NUMBER             NOT NULL,
PROD_NETWGHT_N        NUMBER,
PROD_SHIPWEIGHT_N     NUMBER,
CONSTRAINT CHK LPRODSTATUS
          CHECK (PROD_STATUS_S in ('N', 'Y')),
CONSTRAINT PK_LPRODUCTPRIM PRIMARY KEY ( PROD_ID_N ) );
```

Note 注意：前面的语法用于 Oracle。要想使其在 DB2 或者 Microsoft SQL Server 中起作用，需要用 INTEGER 替换 NUMBER 数据类型。

可以用 MERGE 语句使表 LOCA_PRODUCT 的内容和表 PRODUCT 中的数据保持同步。很可能不必同步所有的列，因为它们中有些是静态的，换言之，这些字段中的数据很少变化，或者永远都不会变化。假定可以改变的值为价格、活动状态、包装尺寸以及发货重量，则 MERGE 语句如下：

```
MERGE INTO local_product lp
USING product p ON (lp.prod_id_n = p.prod_id_n )
WHEN MATCHED THEN UPDATE
SET       lp.prod_price_n = p.prod_price_n,
          lp.prod_status_s = p.prod_status_s,
          lp.prod_pltwid_n = p.prod_pltwid_n,
          lp.prod_pltlen_n = p.prod_pltlen_n,
          lp.prod_shipweight_n = p.prod_shipweight_n
WHEN NOT MATCHED THEN INSERT
VALUES    (p.prod_id_n,
           p.prod_price_n,
           p.prod_num_s,
           p.prod_description_s,
           p.prod_status_s,
           p.prod_brand_s,
           p.prod_pltwid_n,
           p.prod_pltlen_n,
           p.prod_netwght_n,
           p.prod_shipweight_n)
```

现在，当表 LOCAL_PRODUCT 为空时，第一次运行这个语句，在表 PRODUCT 中的所有行都将被插入到本地表中。如果要持续运行（例如，每小时），则这个 MERGE 语句将追踪表 PRODUCT 中可能发生的所有变化，并会适当地更新 LOCAL_PRODUCT 中现存的行，或者插入新行。

6.4.2 供应商特定的 MERGE 语句子句

正如本章先前提到的，本书中所提到的三个数据库都允许有选择的从没有匹配资源表行的目标表中删除行，本节描述了三者之间的语法区别。而且，Oracle MERGE 语法包含一个可选的 LOG ERRORS 子句，就像其他三个 DML 语句。

1. Oracle 11g

Oracle 的 MERGE 语法如下：

```
MERGE INTO [<qualifier>.]<target_table>
USING [<qualifier>.]<source_table> ON (<condition>)
WHEN MATCHED THEN
UPDATE SET {<column> = {<expression> | DEFAULT},...}
[<where_clause>]
[DELETE <where_clause>]
WHEN NOT MATCHED THEN
INSERT [(<column>,...)] VALUES (<expression> | DEFAULT),...;
[LOG ERRORS [[INTO <log_error_table>] [REJECT LIMIT (n)]]]
```

MERGE 语句对具有资源表值所匹配的目标表进行更新，插入表 PRODUCT 中存在而 LOCAL_PRODUCT 中不存在的记录，并删除表 LOCAL_PRODUCT 中相应的行，即与表 PRODUCT 中 PROD_STATUS_S = 'N'相对应的行。

```
MERGE INTO local_product lp
USING product p ON (lp.prod_id_n = p.prod_id_n )
WHEN MATCHED THEN UPDATE
SET       lp.prod_price_n = p.prod_price_n,
          lp.prod_status_s = p.prod_status_s,
          lp.prod_pltwid_n = p.prod_pltwid_n,
          lp.prod_pltlen_n = p.prod_pltlen_n,
          lp.prod_shipweight_n = p.prod_shipweight_n
DELETE WHERE (p.prod_status_s = 'N')
WHEN NOT MATCHED THEN INSERT
VALUES    (p.prod_id_n,
```


www.TopSage.com

```
            p.prod_price_n,
            p.prod_num_s,
            p.prod_description_s,
            p.prod_status_s,
            p.prod_brand_s,
            p.prod_pltwid_n,
            p.prod_pltlen_n,
            p.prod_netwght_n,
            p.prod_shipweight_n);
```

2. DB2 9.5

DB2 语法稍有不同：

```
MERGE INTO [<qualifier>.]<target_table>
USING [<qualifier>.]<source_table> ON (<condition>)
WHEN MATCHED [AND <search_condition>] THEN
{
 UPDATE SET {<column> = {<expression> | DEFAULT},...} |
 DELETE
},..
WHEN NOT MATCHED [AND <search_condition>] THEN
INSERT [(<column>,...)] VALUES (<expression> | DEFAULT),...
```

下面的代码样例与一般样例方式相同，但是也会删除目标表中相应的行，即与对应表中 PROD_STATUS_S 值为'N 相对应的行：

```
MERGE INTO local_product lp
USING product p ON (lp.prod_id_n = p.prod_id_n )
WHEN MATCHED AND (p.prod_status_s = 'Y') THEN UPDATE
SET       lp.prod_price_n = p.prod_price_n,
          lp.prod_status_s = p.prod_status_s,
          lp.prod_pltwid_n = p.prod_pltwid_n,
          lp.prod_pltlen_n = p.prod_pltlen_n,
          lp.prod_shipweight_n = p.prod_shipweight_n
WHEN MATCHED AND (p.prod status_s = 'N') THEN DELETE
WHEN NOT MATCHED THEN INSERT
VALUES    (p.prod_id_n,
           p.prod_price_n,
           p.prod_num_s,
           p.prod_description_s,
           p.prod_status_s,
           p.prod_brand_s,
           p.prod_pltwid_n,
           p.prod_pltlen_n,
           p.prod_netwght_n,
           p.prod_shipweight_n)
```

3. Microsoft SQL Server 2008

Microsoft SQL Server 增加了一个可选子句，WHEN SOURCE NOT MATCHED 和一个可选关键字 INTO：

```
MERGE [INTO] [<qualifier>.]<target_table>
USING [<qualifier>.]<source_table> ON (<condition>)
[ WHEN MATCHED [AND <search_condition>] THEN
 {
  UPDATE SET {<column> = {<expression> | DEFAULT},...} |
  DELETE
 }
],..
[
 WHEN [TARGET] NOT MATCHED [AND <search_condition>] THEN
  INSERT [(<column>,...)] VALUES (<expression> | DEFAULT,...)
 ]
[
 WHEN SOURCE NOT MATCHED [AND <search_condition>] THEN
 {
  UPDATE SET {<column> = {<expression> | DEFAULT},...} |
  DELETE
 }
],..;
```

先前的 DB2 样例也可以用于 Microsoft SQL Server，此外还可以使用 WHEN SOURCE NOT MATCHED 子句。举个例子，假如想要删除在源表中没有匹配行的目的表行，可以按照如下方式修改语句：

```
MERGE INTO local_product lp
USING product p ON (lp.prod_id_n = p.prod_id_n )
WHEN MATCHED AND (p.prod_status_s = 'Y') THEN UPDATE
SET       lp.prod_price_n = p.prod_price_n,
           lp.prod_status_s = p.prod_status_s,
          lp.prod_pltwid_n = p.prod_pltwid_n,
          lp.prod_pltlen_n = p.prod_pltlen_n,
```


www.TopSage.com

大家网 TopSage.com

```
          lp.prod_shipweight_n = p.prod_shipweight_n
WHEN MATCHED AND (p.prod_status_s = 'N') THEN DELETE
WHEN NOT MATCHED THEN INSERT
VALUES    (p.prod_id_n,
           p.prod_price_n,
           p.prod_num_s,
           p.prod_description_s,
           p.prod_status_s,
           p.prod_brand_s,
           p.prod_pltwid_n,
           p.prod_pltlen_n,
           p.prod_netwght_n,
           p.prod_shipweight_n)
WHEN SOURCE NOT MATCHED THEN DELETE;
```

6.5 TRUNCATE 语句

除了本章中所描述的标准 DML 语句外，Oracle 和 Microsoft SQL Server 还引入了 TRUNCATE 语句，在功能上就像没有 WHERE 子句的 DELETE，它会在目标表中删除所有的行。不同的是，与 DELETE 相比 TRUNCATE 执行更快且占用更少的系统资源。值得注意的是，使用 TRUNCATE 时，不会触发目标表上的触发器。(要获得更多的信息，请见第 14 章)。TRUNCATE 在 Oracle 11g 和 Microsoft SQL Server 2008 中的主要限制就是不能用在启用了 FOREIGN KEY 约束引用的表上。

Tip 提示：有时，删除引用完整性约束，删切表，然后再重建约束会比使用 DELETE 更有效率；同样，在 Oracle 中，也可以禁用，然后重新启用约束（参看第 5 章）。

TRUNCATE 的语法很简单：

```
TRUNCATE TABLE<table_name>
```

这个例子阐明 Oracle 中 TRUNCATE 语句的用法：

```
-- TRUNCATE fails because PRODUCT is referenced
-- by foreign key FK_ORDLINE_PRODUCT

TRUNCATE TABLE PRODUCT;

ERROR at line 1:
ORA-02266: unique/primary keys in table referenced by
   enabled foreign keys

-- Disable the constraint and try again
ALTER TABLE ORDER_LINE DISABLE CONSTRAINT FK_ORDLINE_PRODUCT;

Table altered.

TRUNCATE TABLE PRODUCT;

Table truncated.

-- Re-enable the constraint
ALTER TABLE ORDER_LINE ENABLE CONSTRAINT FK_ORDLINE_PRODUCT;

Table altered.
```

Note 注意：DB2 语法中没有 TRUNCATE 语句。TRUNCATE 也不是 SQL:2003 标准的一部分。SQL:2008 中将会包含这条语句，但是它与 Oracle 和 Microsoft SQL Server 的 TRUNCATE 在功能方面会有很大的不同。

Oracle 和 Microsoft SQL Server TRUNCATE 语句的不同

主要的不同点是，Oracle 中 TRUNCATE 是不可逆的，换句话说，不能使用 ROLLBACK 语句回滚。此外，Oracle 把 TRUNCATE 看作 DDL 语句，这意味着与 Oracle 中任何其他的 DDL 语句一样，TRUNCATE 总是以隐式提交来结束事务。就是说在同一事务中，如果在某些表上执行五个 INSERT 语句，十个更新，三个删除，然后又在另一个表上执行了 TRUNCATE，则这些变化将被立即提交，然后结束这个事务。

Cross-Reference 交叉参考：诸如会话、事务、COMMIT 以及 ROLLBACK 等事务控制术语和命令将在第 7 章中进行详细讨论。

Microsoft SQL Server 的事务控制与 Oracle 的有明显的不同，因此，TRUNCATE 的行为也是不同的。

www.TopSage.com

在显式开始的事务中通过 ROLLBACK 语句实现可逆性，且不会执行隐式 COMMIT。

6.6 小结

四个 DML 语句是 INSERT，UPDATE、DELETE 以及 MERGE。

根据定义，RDDBMS 数据是动态的，因为它所表示的现实世界中的实体是经常发生变化的。当一个与数据库有关的新实体出现的时候，需要使用 INSERT 语句在一个或多个表中创建新的数据行来描述这个实体。当存在于数据库中的实体在现实世界中发生变化时，就要使用 UPDATE 语句来修改数据库中与这个实体相关的信息，当一个实体不再与数据库有关，或是完全从现实世界中消失时，就可以使用 DELETE 语句从数据库中删除与之有关的信息。

MERGE 语句结合了 INSERT、UPDATE 以及 DELETE 语句三者的功能。虽然可以使用 INSERT、UPDATE 以及 DELETE 语句来模拟 MERGE 的功能，但是它的使用为开发者节约了时间并提高了性能。

DML 语句具有不同的粒度。可以插入或删除的最小单位为一行；UPDATE 可以在单独行的单独列上执行修改。尽管可以称为从一个列中删除值，或是向一个列插入值，但实际上是在使用一个 UPDATE 语句相应的将这个列的值设置为 NULL 或是某个值。

理解 UPDATE、DELETE 以及 MERGE 语句的 WHERE 子句的重要性是非常关键的。当 WHERE 子句被省略时，所有的目标表行都将受到影响（被修改或被删除）。

除此之外，一些供应商具有与经典的 DML 语句类似功能的语句。在本章中，描述了由 Oracle 和 Microsoft SQL Server 所支持的 TRUNCATE 语句与没有 WHERE 子句的 DELETE 语句一样。

www.TopSage.com

CHAPTER 7

第7章 会话、事务和锁

本章提要：

- 会话
- 数据库
- 事务引入锁

在理想世界中，一个数据库仅由一个用户访问，对数据库的修改以及对全部修改的保存以恰当的步骤进行。但在现实世界中数据常常是共享的，它们可能来自不同的数据源（其中有些不可信任），格式也千差万别；有时它们出现在特殊的时刻；即使数据库服务器在一个时间只有一个用户访问的情况存在，也是极少的。

幸运的是，关系数据库的设计符合多用户环境需求。当多个用户访问同一数据集时，完全不同的问题集出现了：什么数据对用户应是可见的？哪个修改应优先？在一个冗长的数据库进程执行期间，用什么保证对数据的修改不丢失？以上问题及其他很多问题的答案就来自于会话、事务和锁。

事务提供了潜在的数据一致性问题（本章后面章节将讨论其细节）的解决方案，而锁处理数据并发问题，会话描述了事务、锁及等发生的环境。

7.1 会话

RDBMS 服务器和访问它的用户在通信期间内的一切事情都发生在会话环境中。多用户环境下，要关心的主要事情之一是数据完整性。当客户端应用程序建立了一个到 RDBMS 服务器的连接，就说打开了一个会话。这个会话就成为该应用程序的专有通信通道。该应用程序用户可能在这个会话中改变某些设置（例如，默认语言或是默认日期格式）；这些设置只对该会话环境产生影响且仅在这个会话期间有效。会话的实现方法和默认行为在不同的 RDBMS 间可能不同，但基本原理是一样的。

Oracle、IBM 或 Microsoft 都提供了访问各自数据库的工具，读者应至少熟悉其中一种。

Note 注意：每个 RDBMS 程序包都是资源密集型的软件，通常情况下，不推荐将所有 RDBMS 程序包都安装在同一台机器上。

一旦安装了所选择的 RDBMS，就可以在计算机上运行访问 Oracle 11g 的多个 SQL Plus 实例，访问 IBM DB2 9.5 的多个 Microsoft 的 SQLCMD（如果选择 Microsoft SQL Server 2008）或 IBM 的 Command Line Processor（db2cmd）实例，每一个实例会打开自己的会话，这个会话是与建立在这个 RDBMS 服务器上的其他会话隔离的。

www.TopSage.com

大家网 TopSage.com

Note 注意：数据库厂商为各自的 RDBMS 服务器提供的命令行实用程序差异性很大。他们都提供 GUI（Graphical User Interface，图形用户界面）实用程序，此外还有许多第三方实用程序可用。要获得更多信息请见附录 D。

SQL 标准指定了大量会话参数（见表 7-1），这些参数用于数据库的连接和操作。在丢弃了形式但保留精神实质的情况下，尽管一些因素使其成为专有的语法，大部分参数都被 RDBMS 实现了。

表 7-1 SQL 标准连接与会话管理语句

SQL 语句	SQL:2003 特性	描述
CONNECT TO	F771	建立 SQL 会话
SET CONNECTION	F771	从可用连接中选择一个 SQL 连接
DISCONNECT	F771	结束 SQL 连接
SET SESSION CHARACTERISTICS AS	F761 F111	设置当前 SQL 会话的一个或多个特征
SET SESSION AUTHORIZATION	F321	设置 SQL 会话用户标识符以及当前 SQL 会话上下文的当前用户标识符
SET ROLE	F331	设置当前 SQL 会话上下文的当前角色名
SET TIME ZONE	F411	设置当前 SQL 会话的当前默认时间域
SET CATALOG	F651 F761	为< preparable statement>中不合格的<schema name >设置默认目录名，当前会话中的<schema name>是由<execute immediate statement>或者<prepare statement>来准备的。或者为<direct SQL statement>中被直接调用的<schema name>设置默认目录名
SET SCHEMA	F761	为< preparable statement>中不合格的<schema qualified name >设置默认模式名，当前会话中的<schema name>是由<execute immediate statement>或者<prepare statement>来准备的。或者为<direct SQL statement>中被直接调用的<schema name>设置默认模式名
SET NAMES	F461 F761	为< preparable statement>中的<character string literal >设置默认字符集名，当前 SQL 会话中的< character string literal >是由<execute immediate statement>或者<prepare statement>来准备的。或者为<direct SQL statement>中被直接调用的<schema name>设置默认字符集名
SET PATH	S071	设置 SQL 路径，用于确定这个主题在<preparable statement>中使用了带有不合格的<routine name>的程序<routine invocation>。这个程序是在当前 SQL 会话中由<execute immediate statement>或者<prepare statement>来准备的。或者放在<direct SQL statement>中被分别直接调用。在另一个 SQL 路径成功设置之前仍使用 SQL 会话的当前路径
SET TRANSFORM GROUP	S241	设定组名，它可以识别组中将用户自定义类型映射为预定义数据类型的变换函数
SET COLLATION	F693	为 SQL 会话的一个或多个字符集设置校验规则。一个 SQL 会话校验规则会一直有效直到为相同字符集的另一个校验规则设置成功

在 Oracle 11g 中，要建立一个数据库连接，用户必须要有 CREATE SESSIONALTER SESSION 特权。起初，会话的全部缺省参数值从一个特殊的 Oracle 配置文件加载。这个文件只能由数据库管理员或拥有必要权限的人修改。一旦建立连接（会话已创建），用户就可以根据喜好和任务需求对会话进行修改。

Cross-Reference 交叉参考：关于特权的更多信息请参阅第 12 章。

在 Oracle 中，会话参数可用 ALTER SESSION 语句修改。该语句的语法相对复杂，属于高级数据库话题。尽管这些参数可用此语句修改，但它们与 SQL 编程没有太大的关系，例如 DB_BLOCK_CHECKING，HASH_JOIN_ENABLED，MAX_DUMP_FILE_SIZE 等。这些语句更多的是处理与 RDBMS 管理和优化有关的任务，属 Oracle 高级主题。关于可用 ALTER SESSION 语句修改的完整参数列表，请查看 Oracle 文档。

www.TopSage.com

在这里，我们将使用其中的 NLS_DATE_FORMAT 参数来演示改变会话满足特定需求的概念。可以用该参数修改 SQL 查询返回的日期格式，因为它指定了由 TO_CHAR 和 TO_DATE 函数返回的默认日期格式：

```
SQL> SELECT TO_CHAR(SYSDATE) nls_date
     FROM dual;

NLS_DATE
-----------------
06 - 10 - 07
```

TO_CHAR 函数的输出格式是由初始化参数 NLS_DATE_FORMAT 确定的，它是每个新会话的默认值。会话改变，显示的日期格式就改变：

```
SQL> ALTER SESSION
           SET NLS_DATE_FORMAT = 'DD-MON-YYYY HH24:MI:SS';

Session altered.

SQL > SELECT TO_CHAR(SYSDATE) nls_date
           FROM dual;

NLS_DATE
-----------------------------
06-OCT-2007 11:33:44
```

用 ALTER SESSION 语句所做的修改在会话期有效。要永久保持所作修改，应该使用 ALTER SYSTEM 语句。

交叉参考：要阅读关于 NLS_DATE_FORMAT 的更多内容，参考第 10 章。您也可以通过 SET ROLE 语句来控制会话的特权，要获得更多信息请参考第 12 章。

SQL*Plus 会话参数设置

Oracle 的命令行实用程序 SQL*Plus 提供了多个参数，这些参数要么称为“系统变量”，要么称为“SET 命令变量”，这些参数可在由它初始化的会话中设置。这些参数会影响数据在 SQL*PLUS 中的获取、操作与显示。下面是可在 SQL*Plus 中 SET 的一些选项简短列表（见表 7-2）。

所有的这些选项（还有更多别的，未在此列出）都可在 SQL*Plus 环境中 SET，使用的标准语法如下：

```
SET <option> [<value>]
```

在会话期间设定的选项通常在会话结束时丢失。您可以将这些定制的选项保存到一个脚本文件中，以后可以很方便的装载到 SQL*Plus 中，恢复定制的环境。

注意：Oracle 也提供了 iSQL*Plus 实用工具以通过因特网访问 Oracle 数据库。SQL*Plus 中可用的某些命令在 iSQL*Plus 环境中无对应命令。

要查看给定会话设定的所有参数，用 SHOW ALL 命令：

```
SQL> show all
appinfo is ON and set to "SQL*Plus"
arraysize 15
autocommit OFF
autoprint OFF
autorecovery OFF
autotrace OFF
blockterminator "." (hex 2e)
btitle OFF and is the first few characters of the next SELECT statement
cmdsep OFF
colsep " "
compatibility version NATIVE
concat "." (hex 2e)
copycommit 0
. . .
underline "-" (hex 2d)
USER is "ACME"
verify ON
wrap : lines will be wrapped
. . .
```

还有很多参数没有在此列出，要获得更多信息请参考 Oracle 文档。

www.TopSage.com

表 7-2 SQL*Plus SET 命令变量

SET 选项	描述
AUTO[COMMIT]{ON|OFF| IMMEDIATE}	为数据库中挂起的数据修改设置默认行为。设为 OFF(默认值)时，要求用户使用 COMMIT 语句手动提交修改
CMDS[EP]{.|C|ON|OFF}	为 C 设置非字母数字字符，用于隔开一行中的多个 SQL *Plus 命令
ECHO{ON|OFF}	当命令被执行时，控制 START 命令是否在脚本中列出每个命令
ESC[APE]{\\{C|ON|OFF}	将输入的字符定义为转义符
INSTANCE[instance_path| LOCAL]	为会话更改默认实例到指定实例路径
[LIN]ESIZE n	设置 SQL *Plus 可在一行中显示的最大字符数目，范围从 1 到系统允许的最大值
NULL <txet>	当返回的数据包含 Null 时，用于设置要显示的文本
[PAGES]IZE n	显示查询结果时设置每页中的最大行数目
SOLC[ASE]{MIX[ED]|LO[WER]|UP[PER]	在执行之前改变 SQL 命令的大小写
SOLP[ROMPT]{SQL>|text}	设定 SQL*PLUS 命令提示符
TIMI[NG]{ON|OFF}	控制定时统计的显示
[WRA]P {WRA}{N|OFF}	此命令确定输出数据的显示方式：当返回的行长于当前设置值时 ON 允许在下一行显示；OFF 则按当前行的尺寸截断它
XQUERY BASEURI{text}	定义要使用的基础 URL，在书写 Xquery 一般表达式时，有助于改变欲访问的文件的前缀
XQUERY ORDERING{UNORDERED|ORDERED |DEFAULT}	控制 Xquery 返回结果的排序方式
XQUERY NODE{BYVALUE|BYREFERENCE| DEFAULT	为创建的或返回的节点设置保留模式
XQUERY CONTEXT{text}	指定一个 Xquery 上下文项，可以是节点也可以是数值

IBM DB2 在会话环境内为用户提供的控制极少。它提供 SET SESSION AUTHORIZATION（符合 SQL 标准）和 SET SESSION_USER 语句，两者都可用来改变会话上下文。

最为密切的会话控制方法是 SET PASSTHRU 语句，它打开和关闭一个会话，直接将 SQL 数据提交给数据库。另外，在会话期间创建一个全局临时表，该表可把 SESSION 部分当作一个架构（在临时表名称与某些永久表名称相同或其他一些含糊情形中，用于防止访问表时产生混乱）。Db2cmd(CLP)实用程序中列出的选项可用来修改会话的默认行为。例如，-c 选项通知 CLP 自动提交 SQL 语句，-v 选项让实用程序回显标准输出等。尽管很有用，但它们并非 SQL 的组成部分。

Microsoft SQL Server 2008 有许多语句可用来修改当前会话（其中一些显示在表 7-3 和表 7-4 中）。它们不属于 SQL 标准，而是 Transact-SQL 语言的一部分。可以分为几类：影响日期和时间设置的语句、查询执行语句、统计语句、锁定与事务语句、SQL 标准兼容设置语句以及经常用到的辅助设置。

表 7-3 Microsoft SQL Server 2008 的 SQL 兼容语句

SET 选项	描述
SET ANSI_DEFAULTS{ON|OFF}	指定 ANSI 为会话持续期所用的默认值，此选项目的是与 SQL Server 6.5 以及后续版本相兼容
SET ANSI_NULL_DELT_OFF{ON|OFF}	指定默认情况下列是否包含 NULL，如果设置为 ON，创建的新列可以包含 NULL 值（除非指定了 NOT NULL）,否则会产生错误。对于显示设定了允许 NULL 值的列该命令无效。当数据库的 ANSI NULL 默认选项为 TRUE 时，此命令可以修改新列的默认为空性
SET ANSI_NULL_DELT_OFF{ON|OFF}	此语句和上一语句基本相同，但有一个例外：当数据库的 ANSI NULL 默认选项为 FALSE 时，此命令可以修改新列的默认为空性
SET ANSI_NULLS {ON|OFF}	在应用 EQUAL(=)和 NOT EQUAL(<>)与 NULL 值进行比较时，指定与 SQL-92 相兼容的行为
SET ANSI_PADDING {ON|OFF}	在 CHAR,VCHAR,BINARY 和 VARBINARY 数据类型的数值比列尺寸短时的显示方式
SET ANSI_WARNINGS{ON|OFF}	当进行聚合函数（如 SUM、AVG、COUNT 等）计算的列中出现 NULL 时，如设置为 ON 会产生警告消息；OFF 则不产生警告消息；对于除零错误或算术溢出错误，设置为 ON 时会产生错误消息，语句被回滚；设为 OFF 时返回 NULL 值

www.TopSage.com

大家网 TopSage.com

表 7-4 Microsoft SQL Server 2008 SET 语句选列表

SET 语句	描述
SET DATEFORMAT{<format>|@<format ID>}	指定输入的 DATETIME 与 SMALLDATETIME 日期各组成部分的顺序
SET CONCAT_NULL_YIELDS_NULL{ON|OFF}	指定列值（或表达式）串联的结果，其中之一包含 NULL 或都包含 NULL
SET {LANGUAGE<language>|@<language ID>}	指定会话的默认语言。此设置会影响日期、时间格式以及 SQL Server 返回的系统消息
SET NOEXEC	指示会话编译每个查询但不执行
SET PARSE ONLY	指示会话解析每个 SQL 语句并返回错误，既不编译也不执行
SET TEXTSIZE	指定 varchar (max)、nvarchar (max)、varbinary (max)、text、ntext 和由 SELECT 语句返回的图像数据的大小
SET NOCOUNT{ON|OFF}	SQL Server 通常会返回消息以指明受任何给定语句影响的行数。此命令将阻止返回此消息
SET NUMERIC_ROUNDABORT{ON|OFF}	指定会导致精度损失的错误的严重等级。设置为 OFF 时，舍入不产生错误；为 ON 时则会产生错误并不返回结果。如果有其他某些设置，可能会有 NULL 值返回
SHOW ROWCOUNT<integer>	使用此语句，Microsoft SQL Server 在要求的行数（SET 语句中指定）返回后停止处理查询

尽管对这些设置与其含意的详细讨论远远超出了 SQL 主题范围，但我们将讨论其中一些最重要的语句和它们在 Microsoft SQL Server 20008 中对执行 SQL 语句的影响。

在这里提供了一个设置 ANSI_NULLS 影响当前会话输出的例子。SQL-92 标准规定凡含 NULL 的比较运算计算结果总为 FALSE。当 PHONE_SALESMANID_FN 字段不是 NULL 时，下面的语句应该返回 ACME 数据库的 PHONE 表的全部记录。

```
1> SET ANSI_NULLS ON
2> GO
1> SELECT phone_phonenum_s
2> FROM phone
3> WHERE phone_salesmanid_fn <> NULL
4> GO

PHONE_PHONENUM_S
--------------------
 (0 row(s) affected)
```

尽管应有 12 个记录满足该准则，但查询返回零个记录。设置 ANSI_NULLS OFF 可以改变这种情况（仅对 Microsoft SQL Server 有效，Oracle 和 IBM DB2 UDB 均不支持该功能）：

```
1> SET ANSI_NULLS OFF
2> GO
1> SELECT phone_phonenum_s
2> FROM phone
3> WHERE phone_salesmanid_fn <> NULL
4> GO

PHONE_PHONENUM_S
--------------------
(305) 555-8502
(626) 555-4435
(717) 555-5479
(718) 555-7879
(718) 555-5091
(814) 555-0324
(305) 555-8501
(626) 555-4434
(717) 555-5478
(718) 555-7878
(718) 555-5091
(814) 555-0323
(12 row(s) affected)
```

Note 注意：如果使用更为标准的 IS NULL 语法，可以完全避免此种情况。

```
SELECT phone_phonenum_s
FROM   phone
WHERE  phone_salesmanid_fn IS NOT NULL
```

三种 RDBMS 都返回正确结果。因为 NULL 不是一个明确的值，所以必须区别对待。Oracle 11gi 和 IBM DB2 9.5 都有类似 ANSI_NULLS 的设置，有关 NULL 的更多信息请参阅第 3 章。

www.TopSage.com

会话上下文中可修改的设置超过 40 种，表 7-4 列出了其中一部分。

使用 SET 语句可以将多个选项设定为 ON 或 OFF。例如下面语句就同时设定了两个选项：

```
1> SET NOCOUNT, ANSI_DEFAULTS ON
2> GO
```

要检查会话的选项设置，使用下面语句，它返回当前会话已设定的全部活动选项：

```
1> DBCC USEROPTIONS

Set Option                  Value
--------------------------- ---------------------
textsize                    64512
language                    us_english
dateformat                  mdy
datefirst                   7
quoted_identifier           SET
arithabort                  SET
ansi_null_dflt_on           SET
ansi_defaults               SET
ansi_warnings               SET
ansi_padding                SET
ansi_nulls                  SET
concat_null_yields_null     SET

(12 row(s) affected)

DBCC execution completed. If DBCC printed error messages, contact
  your system administrator.
```

数据库控制台命令（DataBase Console Command，DBCC）包是一个包含了所有 DBA 实用工具的工具箱，这些实用工具中的一些选项允许用户访问。它有 60 多个 DBCC 命令，处理 SQL Server 配置、管理、状态检查等工作。

Note

注意：如果 SET 语句设置在存储过程中，则它在会话的存储过程执行期有效，一旦执行结束，将恢复前值。如果使用动态 SQL（参见第 14 章），SET 语句仅影响在其中指定的批处理；其后语句不受该设置影响。

其他与事务和锁有关的一些 SET 语句将在本章后些章节讨论。

当客户终止会话时，无论是自愿终止还是异常终止，全部会话参数都将消失。此外，对于所有挂起的事务，当会话因为自愿终止或是异常终止而回滚时，一个隐式提交将发出。数据库管理员可以结束会话或是断开会话链接，所用语句的语法因 RDBMS 不同而异。

孤立会话

当客户端应用程序突然终止而不能结束链接到 RDBMS 服务器的会话时，孤立会话便产生了。通常情况下，探测客户端退出并通知服务器是操作系统的责任（在某些实现中，客户端停止活动一段时间后，不管客服端是否还在线，服务器都会询问）。但某些情形下阻止客户端退出是合适的（例如，突发的网络失败）。如果会话处于活跃状态（例如，此刻 RDBMS 正在处理某些命令），它将自动探测客户下线并终止会话。另一方面，如果这个会话是非活跃的，正在等待客户的命令，那么该会话会继续对服务器可用。

这样的会话会消耗系统资源，因此应被清除。通常，该工作会在服务器上配置好的某一时间间隔后自动执行。DBA 也可以手动解除这些会话。

7.2 事务

事务是 SQL 强制实现数据库完整性和维护数据一致性的一种机制。尽管保留了 SQL 标准的精神实质，但具体实现细节因 RDBMS 供应商不同而异。

7.2.1 什么是事务

事务为会话的概念补充了额外粒度——它将会话中的每个操作划分为逻辑工作单元。以这种方式，那些包含数据修改的数据库操作就是逐步执行的，随时都可以回滚，或者是在所有步骤都成功完成后被提交。

www.TopSage.com

Note 注意：涉及创建和清除数据库对象的数据定义语句（data definition language， DDL）结构修改是非事务性的，而且不能回滚。

事务的思想是要提供一种机制，保证多步操作按照单元逐个执行。如果事务中的任何一步操作失败，则整个事务将回滚。如果所有步骤都成功完成，则这个事务可以提交（而把所有改变保存到数据库）或是回滚，撤销所有更改。

SQL 标准从非常基本的原则开始定义事务，再逐步地增强这个概念。依照 SQL 标准，事务由 RDBMS 启动并运行直到 COMMIT 或 ROLLBACK 语句发出。其实现细节留给供应商去完成。

表 7-5 SQL：2003 事务管理语句

SQL 语句	SQL:2003 特性	描述
START TRANSACTION	T241	启动 SQL 事务并设置特征
SET TRANSACTION	T251	为 SQL 代理设置下一个 SQL 事务的特征
SET CONSTRAINTS	F721	如果 SQL 事务在当前是活跃的，则为处于当前 SQL 会话中的该 SQL 事务设定约束模式。如果当前没有活跃的 SQL 事务，则在当前 SQL 会话中为 SQL 代理设置下一个 SQL 事务的约束模式
SAVEPOINT	T271	建立保存点
RELESE SAVEPOINT	T271	释放保存点
COMMIT	T261	以提交方式终止当前 SQL 事务
ROLLBACK	T271 T261	终止当前 SQL 事务并回滚，或是回滚自保存点建立以来所有影响了 SQL 数据或模式的操作

事务必须通过 ACID 测试。

- **原子性**。要么全部改变，或者一点都不改变。
- **一致性**。事务完成或事务回滚中的一个操作所涉及的全部数据必须保持一致状态； 数据库完整性不能被损害。
- **隔离性**。一个事务不应该知道由任何其他的事务对数据所做的修改，除非那些修改已提交到数据库。可以设定不同的隔离级以修改默认行为。
- **持久性**。已成功提交数据库的事务结果要能够维持。

现实中的一个经典例子是 ATM（自动取款机）提款操作中的事务。假定您需要 20 美元，决定从最近的自动取款机上取出。插入银行卡，输入 PIN（Personal identification number，个人识别号）会话就启动了。银行一旦确认了身份，您就可以继续操作下去，请求一个金额为 20 美元的取款操作，事务就此开始了。有这样几个操作将被涉及：机器检查您的账户，验证有否有足够的钱来支付这个事务，从账户中扣除这些钱，并把这些钱交付给您。如果以上步骤（可能还会有其他步骤，视银行政策而定）中有任何一步失败的话，则该事务必须终止，一切都要恢复到事务开始前的状态。这意味着只有从余额中扣除取款额，您才能得到现金；除非余额足够支付这个事务且你实际接受了现金，否则银行不能从余额中扣除（当然您也可以取消操作，全部变化将回滚）。

事务模式如 ANSI/ISO SQL 标准中定义的那样，利用了事务的隐式启动，方法是当全部事务逻辑单元都成功执行时使用显式 COMMIT，未提交的修改需回滚（例如程序异常中止）时使用显式 ROLLBACK。大多数供应商都遵循了这个模式。

显式和隐式事务

隐式事务在 SQL92/99 标准中是默认的。只要某些语句（DDL 或 DML 类型）在会话中执行，它们就启动（或持续运行）事务。使用 COMMIT 语句或 ROLLBACK 语句可以终止事务。

显式事务是由客户端应用程序使用 BEGIN TRANSACTION 语句启动的，并以类似于隐式事务协议的方式终止。显式事务是 Microsoft SQL Server 2008 特有的功能且为默认设置。Microsoft SQL Server 2008 提供了 SET IMPLICIT_TRANSACTIONS {ON | OFF} 语句来配置事务的默认行为。当选项为 ON 时，当下列语句中的一个被指定时，SQL Server 自动开始一个事务：ALTER TABLE，CREATE，DELETE，DROP，FETCH，GRANT，INSERT，OPEN，REVKE，SELECT，TRUNCATE TABLE

www.TopSage.com

以及 UPDATE。事务必须显式地提交或回滚，以上列出语句中的任何语句一旦执行，新的事务就被启动。把 IMPLICIT_TRANSACTIONS 选项设置为 OFF 将使事务返回默认的自动提交事务模式。

在每个 RDBMS 实现中，COMMIT 都会隐式地在任何 DDL 语句之前和之后发出。尽管 SQL 标准并不要求如此。

表 7-6 Microsoft SQL Server 2008 事务控制语句

SET 选项	描述
SET IMPLICIT_TRANSACTIONS{ON|OFF}	设置连接的隐式事务模式。一旦设置，如下语句将会启动一个事务：ALTER、TABLE、FETCH、REVOKE、CREATE、GRANT、SELECT、DELETE、INSTERT、TRUNCATE、TABLE、DROP、OPEN、UDATE
SET REMOTE_PROC_TRANSACTIONS{ON|OFF}	在本地事务上下文每当有远程存储过程被执行，就启动一个由 Microsoft 分布式事务协调器（Microsoft Distributed Transaction Coordinator，MS DTC）管理的 Transact-SQL 分布式事务
SET TRANSACTION ISOLATION LEVEL	控制在连接上下文中发出的 Transact-SQL 语句的锁定与行版本行为
BEGIN{TRAN|TRANSACTION}	启动本地显式事务；将@@TRANCOUNT 加 1
BEGIN DISTRIBUTED TRANSACTION	启动由 MS DTC 管理的分布式事务（跨越几个数据库或几个服务器的事务）
COMMIT TRANSACTION	向数据库提交由隐式或用户自定义事务所作的成功修改，释放连接占有的资源，将@@TRANCOUNT 减 1
COMMIT WORK	除 COMMIT TRANSACTION 可接受用户自定义事务名以外，行为与 COMMIT TRANSACTION 相同
ROLLBACK TRANSACTION	将显示或隐式事务回滚到事务开始之处，或者回滚到事务中的一个保存点
ROLLBACK WORK	除 ROLLBACK TRANSACTION 可接受用户自定义事务名以外，行为与 ROLLBACK TRANSACTION 相同
SAVE TRAMSACTION	在事务中设置一个保存点，使事务能够回滚到此处
SET XACT_ABORT	指定在 Transact-SQL 语句产生运行时错误时，当前事务是否自动回滚

7.2.2 事务的 COMMIT 和 ROLLBACK

COMMIT 语句结束当前事务，永久改变在事务期对数据所做的全部修改。三个 RDBMS 供应商以及 SQL99 标准中的 COMMIT 语句语法都是一样的（尽管某些供应商在其中添加了额外选项），而且非常简明：

```
COMMIT[WORK]
```

关键字 WORK 不是必需的，尽管添加它可能更为清晰。通常只需要简单的一个 COMMIT。

简化的 Oracle 11g 语法看起来像下面这样：

```
COMMIT[WORK][ COMMENT (<text>)[WRITE]][FORCE(<text>),[<int>]];
```

在此使用 COMMENT 子句使您能指定一个注释（长达 255 字节），用于记录挂起的每个事务，并可在 DBA2 _PC_PENDING 字典视图中查看（参阅第 13 章获得有关系统目录的更多信息）。FORCE 子句允许手动提交一个未确定的**分布式事务**（本章后面部分将讨论）；它只提交一个命名事务，不影响其他所有事务。

IBM DB2 9.5 事务控制语句（见表 7-8）语法与标准语法相同。在 IBM 术语中，事务是一个工作单元（unit of work, UOW），它在 SQL 语句对数据库操作时总是自动启动，事务持有的全部锁定都被释放。命名事务不受支持。

表 7-7 Oracle 11g 事务控制语句

SET 选项	描述
SET TRANSACTION	开始一个只读或读写事务，建立隔离级，或将当前事务分配到指定的回滚段
COMMIT	在当前事务持续期实现对数据库的永久修改
SAVEPOINT	通过为当前事务创建标志点为事务提供额外粒度。其后的 ROLLBACK 语句仅取消此标志点之后所作的修改
ROLLBACK	取消对数据库所作的所有修改

www.TopSage.com

表 7-8 IBM DB2 9.5 事务控制语句

SET 选项	描述
COMMIT{[WORK]}	在当前事务持续期实现对数据库的永久修改
{SAVEPOINT<savepoint name> [UNIQUE][ON ROLLBACK RETAINS CURSORS][ON ROLLBACK RETAINS LOCKS]	通过为当前事务创建标志点为事务提供额外粒度。其后的 ROLLBACK 语句仅取消此标志点之后所作的修改
ROLLBACK{[WORK][TO SAVEPOINT]}	取消对数据库所作的所有修改；可以选择执行到某个指定的 SAVEPOINT.全部锁将释放（回滚到保存点时例外）

下面语法对 Oracle 11g 和 IBM DBF2 9.5 都适用（两者中的事务通常是隐式启动的）：

```
UPDATE customer
SET cust_status_s  = 'N';

COMMIT;
```

Microsoft SQL Server 2008 除了支持自己的语法外还同样支持 SQL:2007 标准语法。Microsoft 语法允许提交命名的事务，标准语法则不允许。

```
COMMIT[TRAN[SACTION]][<transaction name>]]
```

可以看到，只有 COMMIT 是必需的，其他都是可选的，而且这些关键字可以被缩写（比如，可用 TRAN 代替 TRANSACTION）。也可以使用 COMMIT WORK，其语法与 COMMIT TRANSACTION 相同，区别在于后者允许用户定义事务名。

下面以 Microsoft SQL Server 2008 显式事务模式为例演示 COMMIT 语句：

```
BEGIN TRAN
    SELECT * FROM customer
    UPDATE customer
    SET cust_status_s = 'N'
COMMIT TRAN
```

在末尾的 COMMIT 执行前，任何改变都不会发生。只有 Microsoft 需要 BEGIN TRANS-ACTION 语句启动一个显式事务；在 Oracle 和 DB2 9.5 中，每个 DML 语句的事务总是隐式启动的。三种 RDBMS 实现中的 DDL 语句都不需要 COMMIT.。

COMMIT 执行时，对 Microsoft SQL Server 2008，要让另一个 COMMIT 执行成功必须要隐式或显式地启动一个事务。如果没有事务启动的话，发出这个命令将导致错误：

```
Server: Msg 3902, Level 16, State 1, Line 1
The COMMIT TRANSACTION request has no corresponding
  BEGIN TRANSACTION.
```

在 Oracle 和 DB2 9.5 中不管执行 COMMIT 多少次都不会出现错误。

嵌套事务

对于嵌套事务，使用命名事务显得尤其方便。Oracle 或 IBM DB2 UDB 都没有实现这个概念，它们使用的是匿名事务。其思想是一个事务套一个事务，这样一直无限嵌套下去。每个“子”事务都知道其上层的“父”事务。只有显式事务可以嵌套。

Note 注意：Oracle 与 IBM DB2 中的匿名事务与主事务不共享锁和资源。在主事务中发出的 ROLLBACK 和 COMMIT 命令对匿名事务无效。

在 Microsoft SQL Server 中，任何时候都可以使用@@TRANSCOUNT 一元函数检查挂起事务的总数。在 Microsoft SQL Server 2008 中，引入嵌套事务仅仅是为了提高可读性。提交一个内部事务并不会真的提交任何东西，只有最外层的 COMMIT 才会真正提交所做修改；所有其他的提交仅仅对事务计数器减数而已。下面例子说明了此概念：

```
BEGIN TRANSACTION trans1
-- the transaction counter @@TRANCOUNT = 1
INSERT INTO <table> VALUES <values>
BEGIN TRANSACTION trans2
-- the transaction counter @@TRANCOUNT = 2
INSERT INTO <table> VALUES <values>
BEGIN TRANSACTION trans3
-- the transaction counter @@TRANCOUNT = 3
INSERT INTO <table> VALUES <values>
COMMIT TRANSACTION trans3
-- Nothing committed at this point but the transaction
-- counter is decremented by 1; @@TRANCOUNT = 2
```


www.TopSage.com

```
COMMIT TRANSACTION trans2
-- Nothing committed at this point but the transaction counter
-- is decremented by 1; @@TRANCOUNT = 1
COMMIT TRANSACTION trans1
-- All INSERTs are committed to the database
-- the transaction counter is decremented by 1; @@TRANCOUNT =0
```

本例中有三个事务被启动，它们把三个记录插入到一个表中，只有最后的 COMMIT 才真正对表做了永久修改。

当对数据库中数据的更改需要被“取消”时，应该使用 ROLLBACK。它可以在最后的 COMMIT 之前任何时候发出，从而使该控制事务启动后发生的所有更改都自动回滚。

除了 Microsoft SQL Server 2008 中使用的用户命名事务以及 Oracle 的一些专有可选子句外，SQL:2007 标准和所有 RDBMS 实现中的语法是相同的。下面的语句试图更新 ACME 数据库中 CUSTOMER 表的 CUST_STATUS_S 列，但所有更改都将回滚。

```
UPDATE customer
SET    cust_status_s = 'N'

ROLLBACK WORK
```

就像使用 COMMIT 语句，当 ROLLBACK 命令被发出，所有的锁定都将释放。供应商特定的 ROLLBACK 语句如表 7-9 所示。

表 7-9　　供应商特定 ROLLBACK 语句

RDBMS	ROLLBACK 语法
Oracle 11g	ROLLBACK [WORK][TO SAVEPOINT <savepoint name>]|[FORCE<text>]
IBM DB2 9.5	ROLLBACK [WORK][TO SAVEPOINT <savepoint name>]
Microsoft SQL Server 2008	ROLLBACK [TRAN[SACTION]][<transaction name>][<savepoint name>]

在 Oracle 11g 中，WORK 子句是可选的，其中的 TO SAVEPOINT 子句将在本章后面章节阐述；FORCE 子句与分布式事务有关，其行为与在 COMMIT 事务语句中的情形非常相似；Microsoft SQL Server 还有一个可选<**transaction name**>子句。

Note　注意：由于 DDL 语句会在执行之前和之后自动发出一个 COMMIT，因此，每一个发生在 DDL 语句之前的数据改变将同样被提交。

下面例子对三个 RDBMS 都有效（假定 Microsoft SQL Server 2008 IMPLICIT_TRANSACTION 选项设置为 ON）：

```
UPDATE customer
SET    cust_status_s = 'N'
WHERE  cust_id_n = 1

DELETE customer
WHERE  cust_id_n = 1

ROLLBACK WORK
```

UPDATE 和 DELETE 都不会提交到数据库，因为整个事务都回滚了。

通常，一个事务包含要 COMMIT 或 ROLLBACK 的多条 SQL 语句。为在事务处理中加入粒度，引入了 SAVEPOINT 概念。它允许在事务中指定命名点，命名点通常位于查询中的一些关键点后。如果之后出现任何错误，则所有的更改不回滚到这个事务开始的地方，而是回滚到特定的 SAVEPOINT 处。显式（或隐式的，比如在一个 DDL 语句之后发出的）COMMIT 会释放在事务中声明的全部 SAVEPOINT。

Oracle 11g 的 SAVEPOINT 语法最简明：

```
SAVEPOINT<savepoint name>;
```

这里是在 Oracle 中使用 SAVEPOINT 的一个例子。

```
UPDATE customer
SET    cust_status_s = 'N'
WHERE  cust_id_n = 1;

SAVEPOINT first_update;

DELETE customer
WHERE  cust_id_n = 2;
```


www.TopSage.com

大家网 TopSage.com

```
SAVEPOINT first_delete;

DELETE customer
WHERE  cust_id_n = 10;

ROLLBACK first_update;

COMMIT;
```

上面的例子中，在 COMMIT 发出后，只有 UPDATE 会被提交到数据库，所有 DELETE 都被回滚，SAVEPOINT first_delete 也被清除，查询占有的资源全都释放了。

当前事务中保存点的名字必须是唯一的，如果一个新保存点使用相同名称，则先前的保存点将被清除。

这里是 IBM DB2 9.5 的 SAVEPOINT 语法：

```
SAVEPOINT<savepoint name>[UNIQUE]
[ON ROLLBACK RETAIN CURSORS]
[ON ROLLBACK RETAIN LOCKS]
```

在此，几个可选的子句可以在标准 SAVEPOINT 语句中指定。UNIQUE 子句指明该会话不重用这个名称，这样保证了该名称的唯一性。如果试图用该名称声明另一个保存点就会导致错误。如果忽略了该语句，而在后面使用这个名称的话，则前一个同名 SAVEPOINT 将被清除，一个新的 SAVEPOINT 将创建。

在发生回滚时，ON ROLLBACK 子句用于指定系统如何处理 SAVEPOINT 语句后打开的显式或隐式游标。最后一个子句 ON ROLLBACK RETAIN LOCKS 用于更改默认行为，以指示 RDBMS 不要释放 SAVEPOINT 语句后请求的锁。

交叉参考：有关显式游标的更多信息请参阅第 14 章。IBM 和 Oracle 都使用隐式游标概念，隐式游标是一种用于数据操作的特殊结构，事实上每个 SELECT 语句都会打开一个隐式游标。对隐式游标的讨论不在本书的范围内。

IBM DB2 9.5 还有 RELEASE SAVEPOINT 语句，这个语句会删除命名的保存点之后创建的所有 SAVEPOINT。

Microsoft SQL Server 2008 建立 SAVEPOINT 的语法最不正统：

```
SAVE TRAN[SACTION],<savepoint name>
```

当回滚到一个特定的 SAVEPOINT 后，所有的数据修改都被取消，但所有的锁都被保留，直到 COMMIT 或（回滚到事务起始点的）完全 ROLLBACK 命令发出。分布式事务不支持 SAVE TRAN[SACTION]语句。

下面例子说明了 Microsoft SQL Server 2008 中 SAVE TRANSACTION 语句的用法：

```
BEGIN TRANSACTION trans1

UPDATE customer
SET    cust_status_s = 'N'
WHERE  cust_id_n = 1

SAVE TRANSACTION cust_1

UPDATE customer
SET    cust_status_s = 'N'
WHERE  cust_id_n = 2

ROLLBACK TRANSACTION cust_1

COMMIT TRANSACTION
```

这段代码从命名事务 TRANS1 开始，更新 ID 为 1 的客户的 CUST_STATUS_S 字段，然后创建名为 CUST_1 的 SAVEPOINT。之后，继续更新另一个客户的状态，随后，通过回滚到 CUST_1 保存点使事务回滚对客户 2 所做修改。最后事务被提交，只有第一个更新真正实现了。

分布式事务

涉及一个以上数据库的事务称为**分布式事务**。就其特性而言是非常复杂的，需要高级技巧和知识。

在 Oracle 11g 中，使用 **dblink** 完全限定了对象，从而隐藏了事物处理的细节。RDBMS 服务器管理这些事务并确保数据一致性；无论何时只要这个事务的状态被数据库设置为未确定，这个会话

www.TopSage.com

会发出一个特殊的 ADVISE 语句，以确定该事务是需要回滚还是提交。

IBM DB2 9.5 把分布式事务标记为 DUOW（Distributed Unit Of Work，分布式工作单元），并使用数据库管理器（Database Manager）协调分布式事务。

Note 注意：分布式事务涉及使用两阶段 COMMIT 来协调处理 DUOW，两阶段 COMMIT 是基于标准 X/Open XA 协议的。一个事务内允许有不同数据源。例如，一个批处理查询就可以在一个会话上下文环境里修改 Oracle 11g、MS SQL Server 2008、IBM DB2 9.5 中的表。

在 Microsoft SQL Server 2008 中，管理分布式事务的任务属于 MSDTC（Microsoft Distributed Transaction Coordinator，Microsoft 分布式事务处理协调器）。（其他服从 X/Open XA 规范的事务管理器可用来替代使用。）可以使用 BEGIN DISTRIBUTED TRANS[ACTION]语句显式开始一个事务。

分布式事务必须在网络失败时最小化丢失数据的风险。在分布式事务中使用两阶段提交协议，不同供应商的具体实现细节尽管有所不同，但通常遵循相同的步骤。

- **准备阶段。**事务管理器接收到一个 COMMIT 请求后将这个请求传达到该事务涉及的所有资源管理器，然后这些资源管理器准备执行一个 COMMIT。
- **提交阶段。**在此阶段，这些资源管理器真正发出 COMMIT 并报告给处理协调器；当所有的 COMMIT 都成功后，处理协调器发通知给客户端应用程序。如果有任何资源管理器没能通知处理协调器，则处理协调器发送 ROLLBACK 命令给所有资源管理器。要在 COMMIT 执行后再执行 ROLLBACK，通常需要使用日志文件。

7.2.3 事务隔离级别

事务隔离级别有多种。隔离级别指事务查看其自身范围之外世界（数据）的能力，即查看任何其他事务修改的数据的能力。SQL:2007 标准隔离级别在表 7-10 中列出。

Oracle 11g 拥有三种隔离级——SERIALIZABLE，READ COMMITED 和 READ ONLY。Oracle 的 SET TRANSACTION 语法比较复杂：

```
SET TRANSACTION
    [READ ONLY] | [READ WRITE]
    [ISOLATION LEVEL [SERIALIZABLE | READ COMMITTED]]
    [USE ROLLBACK SEGMENT <segment name>]
    [NAME <transaction name>]
```

表 7-10 SQL:2007 事务隔离级别

隔离级别	描述
READ UNCOMMITED	最低隔离级。允许脏读（即可见未提交数据的能力）。 不发送锁并忽略所有锁
READ COMMITED	当数据正被读取时，共享锁将保持。不允许脏读（包括未提交数据）。尽管幻影续（即两次读取间行数目发生了改变）可能发生
REPEATABLE READ	查询所选择的数据不允许修改（为实现 UPDATE、DELETE 等目的而锁定），但幻影行可能发生
SERIALIZABLE	事务的最高隔离级。锁定整个数据集。事务结束前不允许任何源自外部的修改

可以看到，此语句可设置许多参数，尽管不能够一次完成。要把一个事务设置为 READ ONLY，可以使用下面的语句：

```
SET TRANSACTION READ ONLY NAME 'trans1';

SELECT *
FROM   CUSTOMER ;

COMMIT;
```

如果将事务设置为 READ ONLY，则在这个事务中不能使用 UPDATE、INSERT 或是 DELETE 语句修改任何数据。

Oracle 是“三大”RDBMS 提供商中唯一为事务提供 READ ONLY 模式的。这个子句把事务设置为只读模式，如果试图改变数据就会生成一个错误。它建立起的语句级行为是会话的默认行为。

DB2 9.5 对事务隔离级别的定义中有一些让人混淆的术语。SQL:2007 标准指定为 SERIALIZABLE，

DB2 9.5 却命名为 REPEATABLE READ（RR），它是 DB2 9.5 的最高隔离级。

SQL:2007 的 REPEATABLE READ 变成了 READ STABILITY（RS），并引入了一个新级别 CURSOR STABILITY。

CURSOR STABILITY（CS）是 IBM DB2 9.5 的默认隔离级别，它类似于 SQL 标准中的 READ COMMITTED。从本质上说，它确保一个数据行保持不变。

UNCOMMITTED READ（UR）级别与标准中的定义完全一样：不获取任何锁，因此“脏”读是可能的。

若从应用程序建立连接，可用 PREP 或 BIND API 指示符指定隔离级别，若从命令行处理器指定隔离级别，可以使用下面的语句：

```
db2 =>CHANGE ISOLATION TO UR
DB20000I The CHANGE ISOLATION command completed successfully
```

提示：连接到 DB2 9.5 时可用 SET CURRENT ISOLATION LEVEL 改变隔离级。隔离级可从一个特殊的寄存器 CURRENT ISOLATION 读取。

Microsoft SQL Server 2008 支持前述四种隔离级别，除此还有 SNAPSHOT 隔离级，它确保在事务过程中所读数据与事务启动时一致。隔离级别是为整个会话设置的，而非仅仅为一个单独的事务设置。在会话中指定一个级别，可使用如下语句：

```
SET TRANSACTION ISOLATION LEVEL <level>
```

这里有一个例子，它使用 Microsoft SQL Server 2008 说明了事务隔离级别对于数据操作一致的重要性。（对此例子作点小修改，也适用于 Oracle 和 DB2 9.5）。这个例子执行一次更新，选择更新值，然后回滚事务（关于 OSQL 接口，参见附录 D 获得更多信息）：

```
1> SELECT cust_status_s
2> FROM customer
3> WHERE cust_id_n = 1
4> GO

 cust_status_s
 -------------
 N
(1 row affected)
1> SET TRANSACTION ISOLATION LEVEL READ COMMITTED
2> GO
1> BEGIN TRAN TRAN1
2> UPDATE customer
3> SET cust_status_s = 'Y'
4> WHERE cust_id_n = 1
5> GO

(1 row affected)

1> SELECT cust_status_s
2> FROM customer
3> WHERE cust_id_n = 1
4> GO

 cust_status_s
 -------------
 Y

(1 row affected)

1> ROLLBACK TRAN TRAN1
2> GO
1> SELECT cust_status_s
2> FROM customer
3> WHERE cust_id_n = 1
4> GO

 cust_status_s
 -------------
 N

(1 row affected)
```

事务 TRANS1 更新了 CUST_STATUS_S 字段，把它从 N 修改为 Y，然后发出一条 SELECT 语句显示改变了的数据。会话的事务隔离级别是 READ COMMITTED，因此，只有提交到数据库的修改才可

www.TopSage.com

能被选择。由于这个 SELECT 是在同一事务中发出的，因此能够看到事务更新造成的未提交更改。在事务隔离级别设置为 READ UNCOMMITTED 的会话内，未提交的数据更改对试图选择它的其他事务来说是可见的；但对于其他隔离级别的事务不可见（如果它们先于 ROLLBACK TRANSACTION 语句被发出的话）。这个例子还表明在事务回滚后，数据保持不变。

7.3 理解锁

并发是在多用户环境中的主要事情之一。当多个会话从共享资源读取数据，或是向共享资源写入数据，数据库就可能失去完整性。为了防止发生这种情况，每个 RDBMS 都实现了有效的并发控制机制。RDBMS 服务器的并发是通过各种锁定机制管理的。三个领先的 RDBMS 都有一套实现并发管理的复杂机制。尽管锁定不是 SQL 和 SQL:2007 标准（指定了游标锁定）的组成部分，但理解这一概念很重要。

Oracle 拥有最高级和最复杂的锁定机制。它遵循读写进程不能互相阻塞的规则，即使它们工作在同一（或相近的）数据集上。每个会话都收到数据的读一致快照。因此，即使其他进程已经开始更改此数据集中的数据只要还未提交所作更改，后来的每个会话都能够读取数据集未更改前的数据；一旦第一个会话提交了所作更改，则其他所有会话将能看到这些更改。锁仅当更改被提交到数据库时才能获取。Oracle 自动选择限制最小的锁。用户可以手动选择锁定资源（例如表），在这种情况下，根据部署的锁类型，其他用户仍然能够访问数据。

IBM DB2 9.5 和 Microsoft SQL Server 2008 都使用这样的锁，这些锁允许读取者阻塞写入者，反之亦然。并发数据访问的问题通过锁定粒度（表、页、行等）稍微缓解。只读查询、DDL 语句、DML 查询等都能获取锁。每种情形用的锁类型不同，接下来我们将详细进行讨论。

用户在多数时间里无需担心锁定，因为 RDBMS 会自动为特定操作选择最恰当的（或多个）锁。只有在逻辑故障出现时，才应尝试使用 SQL 语句手动指定锁。

7.3.1 锁定模式

并发有两大类：乐观并发和悲观并发。顾名思义，乐观并发事务假定一个以上的事务工作于同一数据集时资源冲突不太可能发生（事实上是可能发生）。当更改提交到数据库时乐观事务检查潜在的冲突，通过重新提交数据来解决冲突（这种情况下，最后一个 COMMIT 优先）。悲观并发事务一开始就预防冲突，并锁定它们要使用的全部资源。悲观并发确保了最高级别的数据一致性，但付出代价最大，而且可能使数据库停滞。通常 RDBMS 两种事务都会使用，用户也可以指定使用哪种事务。

注意：锁定粒度对系统性能有重大影响。行级锁定可以增加并发（如果它阻塞其他事务访问表的话），但通常会导致管理上的开销，减缓数据库的运行速度。完全的表锁定在系统资源方面的消耗会更少，但要付出并发和性能方面的代价。所以在设计数据库应用程序时需要紧记这个关系。

悲观事务执行时要用到锁定，尽管有一些类似之处，但每个 RDBMS 都有自己的锁定级别。通常，锁定要么共享要么独占，这就要涉及使用资源（例如表）的方式了。

Oracle 中，当客户进程访问资源时，它可以指定一种锁定类型来显式锁定所访问资源。任何自动的锁定设置都将被这个锁定覆盖。锁定类型如表 7-11 所示。

表 7-11 Oracle 11g 的锁模式

锁定模式	描述
EXCLUSIVE	允许对已锁定表用 SELECT 进行查询；其他事务的所有其他操作（如 UPDATE, DELETE 等）都被禁止
SHARE	允许并发查询，但所有事务的更新操作都被禁止
ROW SHARE	允许对表进行并发访问，但其他用户无法独占对表的锁定。此外，还提供了 SHARE UPDATE 模式实现向后兼容性
ROW EXCLUSIVE	与 ROW SHARE 基本相同，但阻止 SHARE 模式下的锁定
SHARE ROW EXCLUSIVE	锁定整个表。允许查询，但其他事务不能得到表上的任何锁

www.TopSage.com

例如，下面的语句以独占模式锁定 ACME 数据库中的 CUSTOMER 表：

```
LOCK TABLE customer
IN EXCLUSIVE MODE;
```

发出这个语句的事务试图锁定表，受到表 7-11 指定的限制。如果其他任何进程在此表上保留了一个锁，该事务将被插入一个队列，然后按接收的优先级获取锁。这个锁在该事务期（换句话说，直到 COMMIT 被执行）一直为其持有。如果一个事务已经持有这个表上的锁，又试图获取另一个资源上的锁，而第二个事务在那个资源上已有一个锁，那就可能出现死锁。如果一个事务试图锁定已经被锁定的表，可用 NOWAIT 子句引导事务继续运行：

```
LOCK TABLE customer
IN EXCLUSIVE MODE NOWAIT;
```

如果锁定命令是为视图发出的，则 Oracle 将试图为这个视图锁定基表。某些类型的操作需要锁定。只有当表能被锁定时，Oracle 才允许在这个表上执行 DDL 操作。（也可使用该语句锁定 Oracle 中的一些其他类型对象，如 dblink。）

注意：Oracle 允许在 CREATE 和 ALTER TABLE 语句中指定一个特殊的子句来允许或禁止对表的锁定。禁止对表的锁定可以有效阻止任何对那个表的 DDL 操作。

Oracle 提供了几个提示以优化性能，其中某些提示会影响 Oracle 对锁定的使用。尽管提示对 Oracle 数据库的调整和优化非常重要，但它们超出了本书的讨论范围，请参考供应商文档获得更多信息。

注意：某些数据库，尤其是 Microsoft SQL Server 2008、IBM DB2 9.5，实现了**锁升级**。它将大量精细锁转换为粗糙些的高一级锁，从而减少系统开销。Oracle 11g 并不升级锁，而是采用更好的并发机制节约系统资源。

IBM DB2 9.5 与 Oracle 的定制锁定控制类似，但粒度更小。用户可以指定两种表锁定模式：SHARE 或 EXCLUSIVE。例如：

```
db2=>LOCK TABLE customer IN EXCLUSIVE MODE

DB20000I The SQL command completed successfully.
```

除只读 SELECT 外，SHARE 模式阻止任何其他事务对锁定表作任何类型的操作；同样，其他事务都不能获得对这个表的锁。EXCLUSIVE 模式阻止任何对这个表的操作，包括只读操作。

锁在事务期间被保持，在 COMMIT 语句发出后被释放。除这两种模式外，DB2 9.5 数据库操作的锁定由为事务设置的隔离级别管理（在本章前面部分描述过）。

所有其他的锁都由 RDBMS 自行决定。默认为行级锁定可升级为表级锁定（DB2 9.5 没有页级锁定）；锁定升级可用上面的 LOCK TABLE 语句来避免。升级阈值可由 DBA 对很多参数（如 **maxlocks**、**locksize** 等）的配置确定。

Microsoft SQL Server 2008 为事务提供了几个可设定的锁选项（见表 7-12）。这里描述的锁类别可以进一步由指定的 HINTS 划分，其中一些 HINTS 在表 7-13 中给出。

锁模式由 SQL Server 自己选择，或基于所执行操作的类型选择。可以使用表 7-13 列出的查询表级锁定提示来手动指定锁模式。这些锁定提示覆盖事务隔离级别，可以合理应用。表 7-13 仅提供了一些可用样例，并没有完整列出锁模式。

表 7-12　　Microsoft SQL Server 2008 的锁模式

锁定模式	描述
SHARED(S)	用于只读操作
UPDATE(U)	用于更新数据时
EXCLUSIVE(X)	不允许别的事务实施 UPDATE,DELETE 或是 INSERT 操作
INTENT	用于建立一个锁层次结构：意向锁、共享意向锁、独占意向锁以及共享式独占意向锁。意向锁指 SQL Server 有在资源的低层（比如表-页-行）获得共享锁或独占锁的意向；意向锁至少会阻止任何事务获得对此资源的独占锁
SCHEMA	用于 DDL 操作执行时
BULK UPDATE(BU)	用于批量复制时

www.TopSage.com

表 7-13 Microsoft SQL Server 2008 的锁定提示

锁定提示	描述
NOLOCK	此提示在 SELECT 语句中发出，指定不使用共享锁并忽略独占锁。意味着 SELECT 语句可以读取未提交事务（即脏读）
UPDLOCK	当读取数据时指示 SQL Server 使用 UPDATE 锁（与共享锁是相反的），这确保了当有 UPDATE 语句跟随时数据不发生改变
XLOCK	放置一个独占锁直到事务结束。使用此锁可提供了额外的粒度级别
ROWLOCK	指示 SQL Server 使用行级锁定（与页级和表级相反）

例如，要在 SELECT 语句中为这个事务指定行级锁定，可使用如下语法：

```
SELECT  *
FROM    customer
WITH    (ROWLOCK)
```

高粒度会导致性能降低，因为 SQL Server 要为行级锁定操作分配更多资源。

Note 注意：除提供企业管理器（Enterprise Manager）可视化界面外，Microsoft SQL Server 还提供存储过程 sp_lock，它返回系统中所有活动锁的相关信息，但这需要足够的特权级别。

SQL Server 2008 会基于成本自行部署不同锁：默认为行级锁，当超过升级阈值时（SQL 语句在一个表或一个索引上获取的锁至少 5000 个，或是锁的总数目超出了数据库实例的内存或配置阈值），可升级为页级锁，进而成为表级锁。此参数不可配置，而是可以根据每种情形由 SQL Server 自行配置。

7.3.2 处理死锁

当两个（或更多个）会话正在等待获取共享资源上的一个锁，第二个会话在某些其他的资源上也有锁，而那个资源又是第一个会话所必需的从而导致任何一个会话都无法继续运行下去，这就是死锁的典型情形。想像这样一种情形，会话 1 持有资源 A，正试图访问资源 B；同时，会话 2 持有资源 B，并正试图访问资源 A。通常，RDBMS 会结束一个进程并回滚它引起的更改，通过这种方法就可自动解决类似问题。

Oracle 实现了一种复杂而完善的机制，这种机制遵循的是“读进程和写进程不能相互阻塞”规则。此规则的思想是为每个进程提供数据的一致数据映像，此数据映像是不带有未提交更改的。然而，如果死锁确实在 Oracle 中发生了，则通常由 RDBMS 自己解决；在某些罕见情形中需要手动解决即手工选择死锁中的“受害”进程。最普通的死锁类型是 ORA-00060（队列死锁）和 ORA-04020（库缓存死锁）。指定 NOWAIT 子句或设置会话超时时间可以避免死锁，其他的一些技术则要涉及显式锁定和在事务中隔离级别的使用。死锁也可通过 Oracle 界面手动解决。

IBM DB2 运行一个名为死锁检测器（Deadlock Detector）的后台进程来发现和解决死锁。死锁受害会话会被回滚并生成一个特殊错误（SQLCODE-901，SQLSTATE40001）。对于死锁受害会话的选择，只读进程是首选，此外，DB2 还使用“最小代价”准则选择要终止的会话。任何时候，如果死锁问题产生，IBM 推荐使用系统监视工具来收集有关死锁情况的信息，之后通过系统优化或是重新设计与之有关的应用程序来解决。

Microsoft SQL Server 2008 使用一种专有算法检测死锁。并以与 Oracle 和 DB2 9.5 类似的方法解除死锁：死锁可通过企业管理器控制台（Enterprise Manager Console）手动解除或自动解除。可以通过设置会话的 DEADLOCK_PRIORITY 参数使它自动成为死锁受害会话。

```
SET DEADLOCK_PRIORITY LOW
```

处理这种情况的另一种方法是设置会话的 LOCK_TIMEOUT 参数。设置超时意味着会话持有锁定资源的时间不会长于指定的时间间隔。一旦设定的时间过期，SQL Server 会返回一个错误，这个事务将回滚。此种情况的解决方法与其他所有的 RDBMS 相似：当错误显示返回死锁情况时，通过重新运行事务，或通过重新设计应用程序减少或消除死锁可能性等来处理。在 Microsoft SQL Server 2008 中可用跟踪标志（1204 和 1222）跟踪死锁；设置跟踪标志会使 SQL 服务器错误日志记录死锁信息。

www.TopSage.com

大家网 TopSage.com

7.4 小结

所有与RDBMS的通信都发生在会话环境中。当会话在客户程序和RDBMS间建立时，它具有某些默认特性，这些特性决定了它的行为。其中的一些特性可在会话期被更改，而数据库管理员可以让在会话中所作的修改一致有效。可通过SQL语句更改；而有些则依赖于客户端，且必须在客户环境中设置。

下一粒度级是事务，当一条或多条SQL语句（比如批处理）组成一个单一的逻辑工作单元时就构成了一个事务。SQL:2007标准规定SQL语句总是作为事务运行。RDBMS在实现上则是区别对待的，有些RDBMS在默认情况下启动隐式事务，而另一些则要用显式语句开始一个事务。事务必须满足某些准则（ACID测试）以遵从SQL标准，但具体细节通常由RDBMS自行确定。

访问共享资源的事务必须实现某些并发控制机制。事务的特性之一就是隔离级别。隔离级别控制这个事务可访问的数据和允许修改的数据。

SQL标准定义了四个隔离级别，有些RDBMSs（例如Microsoft SQL Server 2008与IBM DB2 9.5）将它们全部实现了，而其他RDBMSs(例如Oracle 11g)只实现了其中三个。

有些RDBMS实现了一套复杂的锁定系统处理并发问题，尽管锁不属于SQL标准的一部分。锁有不同的类型。它们可在SQL语句中指定，或被指定为会话特性。死锁可能出现在有大量事务要处理或是设计不恰当的系统中。死锁可由RDBMS自动解除或由数据库管理员手动解除。

www.TopSage.com

PART 4

检索和传送数据

本部分包括：

第8章
理解 SELECT 语句

第9章
多表查询

第10章
SQL 函数

第11章
SQL 运算符

www.TopSage.com

www.TopSage.com

大家网
TopSage.com

第8章
理解 SELECT 语句

CHAPTER 8

本章讨论使用 SELECT 语句从 RDBMS 表中选取数据。顾名思义，该语句是从 RDBMS 对象（表或视图）中选取数据，用于呈现给用户，或者出于内部目的。SELECT 语句是数据查询语言（DQL）组的唯一语句。

SELECT 的使用比较简单，但有趣的是，其中一般包含一些可选子句。因此，在 SQL 中，SELECT 语句中的子句可能是最易混淆的，所以须相应处理。本章介绍的主题包括子查询、复合运算符，聚合函数等的使用。

Tip 提示：如果您想练习本章及后面章节中的 SQL 示例，可以下载并运行一组 SQL 脚本，在选定的 RDBMS（如 Oracle 11g、DB2 9.5 或 Microsoft SQL Server 2008）中创建（如果需要，则重新创建）ACME 表，然后用样本数据来填充这些表。请参见本书随附网站的链接。

本章提要：

- 单表查询
- SELECT 语句中的必需和可选子句
- 常量、函数和计算列的使用
- 不同 SELECT 子句中子查询的使用
- SELECT 语句的复合运算符
- 对查询结果进行分组和分类
- 合并多重查询结果
- 分析函数和 top N 查询

8.1 单表 SELECT 语句语法

下面是通用 SELECT 语句，因为它是依照 SQL:2003 标准定义的，用于从单表中选取数据。查询包括 SELECT 命令，然后是标识符列表（表或视图列），接下来是必备的包含表名的 FROM 子句（在 Microsoft SQL Server 2008 中为可选），从这些表中选取列。

子句中的其他项是可选的，用于增加查询的选择性、排序和分组能力。所有这些组成了完整的 SELECT 语句。

```
SELECT [DISTINCT]
                {
                 [<qualifier>.]<column_name> |
                 * |
                 <expression> |
                 <pseudocolumn>
                 [AS <column_alias>]
                },...
FROM
       {
        <table_or_view_name> |
        <inline_view>
        [[AS] <table_alias>]
       }
[WHERE <predicate>]
```

www.TopSage.com

```
[GROUP BY [<qualifier>.]<column_name>,...
 [HAVING <predicate>]
]
[ORDER_BY {
           <column_name> |
           <column_number>
          }
          [ASC | DESC],...
];
```

当然，在三个主要的RDBMS实现过程中，运行的语句在语法和用法上各不相同。

8.2 SELECT 子句：选取什么?

在关系数据库中，SELECT语句选取列值、常量值或表达式。返回值可以是任意的有效数据类型，它们可以在客户端应用程序中显示、被写入文件、用于中间计算或被填入数据库表中。

8.2.1 单列选取

只要拥有SELECT权限（参阅第12章），就可以从该表（或视图）中选取任意数目的列。下面的例子只选取一个客户名（数据库ACME的CUSTOMER表中的CUST_NAME_S列）。

```
SELECT cust_name_s
FROM   customer
CUST_NAME_S
--------------------------------------------------
...
DLH INDUSTRIES
FAIR AND SONS AIR CONDTNG
KILBURN GLASS INDUSTRIES
BOSWELL DESIGNS CORP.
WILE ELECTROMATIC INC.
FABRITEK INC.
...
DALCOMP INC.
INTEGRATED POWER DESIGNS
GUARDIAN MANUFACTURING INC.
WILE BESS COMPANY
```

返回结果是包含CUSTOMER表中所有行的CUST_NAME_S列值集合，其中的值不以任何方式排序。但对返回值的排序是可能的，请参阅本章后面在结果集中对数据排序的内容（ORDER BY子句部分）。

该语法在三个RDBMS产品之间以及SQL:2003标准中是相同的。

8.2.2 多列SELECT

尽管单列SELECT很有用，但SQL的能力并不仅限于此，很有可能要在查询时一次选取多个列。

1. 选择多个列

下面的查询从ACME数据库的CUSTOMER表中同时选取三列。

```
SELECT    cust_id_n,
          Cust_status_s,
          Cust_name_s
FROM      customer

CUST_ID_N  C  CUST_NAME_S
---------- - --------------------------------------------
         51 Y DLH INDUSTRIES
          5 Y FAIR AND SONS AIR CONDTNG
         12 Y KILBURN GLASS INDUSTRIES
         61 Y BOSWELL DESIGNS CORP.
         55 Y WILE ELECTROMATIC INC.
          6 Y FABRITEK INC.
         ...
         16 Y DALCOMP INC.
         89 Y INTEGRATED POWER DESIGNS
         85 Y GUARDIAN MANUFACTURING INC.
        152 Y WILE BESS COMPANY
```

同单列SELECT一样，返回结果也是值的集合，不过更确切地说，应该是集合的集合——它包含SELECT语句中被引用的每一列。这些集合出现的顺序与SELECT子句中引用列名的顺序是

www.TopSage.com

一致的。

Note 注意：同一列有可能在一次查询中被多次选取，返回结果将是与之完全相同的值集合。

该语法在三个 RDBMS 产品之间以及 SQL:2003 标准中是相同的。

2. 选取所有列

选取表中所有列可以通过 SELECT 查询子句列出表中每列来实现，或者利用 SQL:2003 标准提供的快捷方式（*），而且利用所有 RDBMS 中任意一个都可实现。

```
SELECT *
FROM status

STATUS_ID_N ST STATUS_DESC_S
----------- -- ------------------------------
          6 60 SHIPPED
          2 20 COMPLETE
          8 70 INVOICED
          9 80 CANCELLED
```

与多行查询一样，返回结果包含各列所有行的选取值。各列在结果集中出现的顺序与在基础表（视图）中完全相同。

3. 选取所有列和一个附加列

有一种相对少见的情况，要从表中选取所有列和一（几）列的副本，这在三个 RDBMS 中都能实现。但是，各供应商的实现过程与 SQL:2003 标准不同，它们之间也存在差异。

Oracle 和 DB2 中要求用星号作为表名或表别名的前缀（星号与其他列的出现顺序无关紧要）：

```
SELECT status.*, status_desc_s
FROM status;
STATUS_ID_N ST STATUS_DESC_S          STATUS_DESC_S
----------- -- ------------------------- ------------------
          6 60 SHIPPED                   SHIPPED
          2 20 COMPLETE                  COMPLETE
          8 70 INVOICED                  INVOICED
          9 80 CANCELLED                 CANCELLED

4 rows selected.
```

该语法在 Microsoft SQL Server 中同样适用，但没有全称的限制规定，因此可以简单如下使用：

```
SELECT *, status_desc_s
FROM status

GO

STATUS_ID_N STATUS_CODE_S STATUS_DESC_S         STATUS_DESC_S
----------- ------------- -------------------- ---------
          2 20 COMPLETE                        COMPLETE
          6 60 SHIPPED                         SHIPPED
          8 70 INVOICED                        INVOICED
          9 80 CANCELLED                       CANCELLED

(4 rows affected)
```

这看起来像一个多余的功能，但假想这样一种情况：您需要浏览表中全部 200 列数据，并且想改变其在最终结果集中的默认顺序，可以选择按想要的顺序列出所有列（即在 SELECT 查询子句中输入所有列），或采用更方便的方法，但要创建值集合副本——**在可以看到的位置**。当然，可以多次把表中任何或全部行合并成一个结果集。

4. 选取唯一值

由于查询中返回期望的数据更具选择性，因此可能需要消除重复值。SQL:2003 标准提供了一种简便的方法来消除最终结果集中的重复值。

数据库 ACME 的表 PAYMENT_TERMS 中存放着客户的折扣数据，并用百分数表示。每个折扣的详细信息可能不同，但显然实际的百分比可能相同。

下面的例子从表中选取 PAYTERMS_DISCPCT_N 列的所有行：

```
SELECT payterms_discpct_n

FROM payment_terms

PAYTERMS_DISCPCT_N
```


www.TopSage.com

大家网 TopSage.com

```
-----------------
               .00
               .02
               .02
               .00
               .00
               .00
               .00
               .00
               .00
               .00
               .02

(11 rows affected)
```

可以看到，在 11 条记录中有很多的重复值。如果想找出公司使用的百分率，可以用关键字 DISTINCT：

```
SELECT DISTINCT payterms_discpct_n

FROM            payment_terms

PAYTERMS_DISCPCT_N
------------------
                 0
               .02

2 rows selected.
```

现在，所有的重复值都被消除了，结果集中只有两个唯一值。

要理解 DISTINCT 针对的是整行而不是一列，这一点很重要；如果 DISTINCT 修饰的是多列，那么集合中所有列中的行值都必须是不重复的。这里给出一个在 Microsoft SQL Server 2008 中运行的例子（在 Oracle 和 DB2 中同样适用）：

```
SELECT DISTINCT payterms_discpct_n,
                payterms_code_s
FROM            payment_terms

PAYTERMS_DISCPCT_N PAYTERMS_CODE_S
------------------ ---------------
.00                N10
.00                N120
.00                N15
.00                N20
.00                N30
.00                N45
.00                N60
.00                N90
.02                CADV
.02                N21530
.02                N21531

(11 row(s) affected)
```

在 PAYTERMS_DISCPCT_N 列的基础上又增加了 PAYTERMS_CODE_S 列，就需要两列的值都是唯一的。因此，只要满足两列中的值都是唯一的，单列中的重复值是允许的。

8.2.3 使用常量、函数和计算列

在关系数据库中，不只可以选取列，SELECT 语句也不是只涉及表。与选取表达式和常量相比，从一个表中选取列是一个很简单的概念。

1. 伪表

一个值直到调用时才出现，因为它通过一个函数返回，或还在动态地计算中，但仍然要从某处选取它。这里的“某处”可以是数据库中存放的任何表，而且必须对它拥有选取权限（参阅第 12 章获得更多信息），这将会很麻烦，而且很难维护，因为用户必须知道数据库中含有哪些表，而且每当删除或重命名使用中的表时都要改变查询。另外，如果从一个实表中选择一个常量或表达式，结果集中的行数和表中是一样的。不同的 RDBMS 供应商对“伪表”概念的实现不同。

SQL:2003

“伪”表没有选取源的能力，也不属于 SQL 标准，确切地说，它只是供应商为取悦客户的独创性

www.TopSage.com

表现形式。

Oracle 11g

当没有实体对象（视图或表）供选择时，Oracle 提供了一个特殊的表 DUAL。假定需要查询 Oracle 数据库的系统时间，RDBMS 提供了一个函数 SYSDATE 来返回系统时间。尽管该函数与 SQL:2003 标准不兼容，却是一个合法的 Oracle 11g 专用 SQL 语句，必须予以重视。所有查询都以 SELECT 开头，该函数中同样也是，但是数据库中不存放信息，有请求时才动态生成信息。这就是 DUAL 表的用途所在——实际上可以从中“选取”任何数据。

```
SQL> SELECT SYSDATE FROM dual;

SYSDATE
---------
21-OCT-07
```

DUAL 表中只有一列（俗称 DUMMY）和一个包含 X 值的行。

注意：DUAL 表作为 Oracle 数据字典的基本对象，由 Oracle 公司的 Chuck Weiss 创建。它从来不单独存在，而是用在一些复杂的 JOIN 操作中（在第 9 章讨论）。它在逻辑上符合 SELECT 概念，因为它暗指每一个语句中都有 FROM 子句。

DB2 9.5

对于选取非列值，DB2 9.5 有本质上相同的概念——在 SYSIBM 系统架构下存放 SYSDUMMY1 表。该表有一个称为 IBMREQD 的单列，和一个包含 Y 值的行。下面的例子使用该表计算两个数的和。

```
db2=> SELECT (5+5) FROM sysibm.sysdummy1

1
---------
             10

1 record(s) selected.
```

这些数的和并不被存储，但是必须按照 SQL:2003 标准进行**选择**。在过程语言（例如 Oracle PL/SQL、DB2 SQL PL 或 Microsoft Transact-SQL）中使用 SQL 函数时，这一点更加重要。

新增功能：DB2 9.5 有了很多以前只属于 Oracle 独有的新功能。这些功能除了 NVL（）和 DECODE（）等函数，伪列 ROWNUM（在本章后面讨论）、NUMBER 和 VARCHAR2 等数据类型和 Oracle 的“旧”表示法——外连接（在第 9 章讨论）之外，现在 DB2 9.5 在 SYSIBM 系统模式下存在与 Oracle 的 DUAL 表结构相同的 DUAL 表。为利用这些功能，并且不使用 SYSIBM 前缀就能引用 DUAL 表，必须将 DB2_COMPATIBILITY_ VECTOR 注册表变量设为一个特定数值（详细信息参见供应商说明文档）。

Microsoft SQL Server 2008

在与此类似的情况下，Microsoft SQL Server 2008 实际上允许完全摒弃 FROM 子句，这里给出一个例子：

```
SELECT (5+5) num_sum

num_sum
-----------
10
```

2. 列别名

在上例中，NUM_SUM 是一个不存在列的列名，这是一个**别名**——三大 RDBMS 供应商都支持的标准功能，也符合 SQL 标准。别名的思想很简单——为在 SQL 查询中选中的列赋予更生动的名称，该名称出现在最终结果集中。本书中的每个 SELECT 例子都能用别名代替列名而重新改写。例如，前面的例子中，选取 PAYTERMS_DISCPCT_N 的唯一值可以写成如下形式：

```
SELECT DISTINCT payterms_discpct_n AS discount_percent
FROM            payment_terms

discount_percent
------------------
                 0
               .02
```

新名称尽管并不简短，但正如列名为程序员提供信息一样（表明该列包含的数据类型：N 代表

NUMERIC），别名对用户来说更具描述性。AS 运算符是可选的，该语法（有或没有 AS）在三类数据库中都是有效的。

3. 常量、函数和表达式

当选取常量和表达式时，别名的作用更大。本章前面提到的例子是选取表达式（5+5），下面是包含常量的 SELECT 语句的例子：

```
SELECT cust_name_s,
       100 AS NUMERIC_CONSTANT,
       'ABC' AS STRING_CONSTANT
FROM   customer

CUST_NAME_S                    NUMERIC_CONSTANT  STRING_CONSTANT
------------------------------ ----------------- ---------------
WILE SEAL CORP.                              100 ABC
WILE ELECTRONICS INC.                        100 ABC
WILE ELECTROMUSICAL INC.                     100 ABC
WILE ELECTROMATIC INC.                       100 ABC
WILE BESS COMPANY                            100 ABC
MAGNETOMETRIC DEVICES INC.                   100 ABC
MAGNETICS USA INC.                           100 ABC
. . .                                        . . .
BURNETTE WILLIAM CORP.                       100 ABC
BOSWELL DESIGNS CORP.                        100 ABC
37 record(s) selected.
```

默认情况下，如果没有指定常量、表达式或函数这些构造列的别名，Oracle 会将常量本身或表达式放在别名的位置；IBM DB2 列编号以 1 开头；Microsoft SQL Server 只是将列名留空。

常量是“标准”SELECT 查询的一部分，在上例中，它们和列名一起列出来。

当得到的值是由列、函数、表达式或常量组合而成时，使用别名非常方便。下面是一个遵循 Oracle 和 DB2 语法的例子，将数据库 ACME 的 PRODUCT 表中的产品 ID（PROD_ID_N 列）连同一个空格以及产品商标（PROD_BRAND_N 列）连接成一个值：

```
SELECT CAST(prod_id_n AS CHAR(5)) || ' ' || prod_brand_s
 AS ID_AND_BRAND
FROM product

ID_AND_BRAND
--------------------------
990   SPRUCE LUMBER
1880  STEEL NAILS
2871  STOOL CAPS
3045  STOOL CAPS
4000  HAND RAILS ROUNDS
4055  GAZEBOS
4761  BAR RAILS
4906  BAR RAILS
4964  BASES
5786  CRATING MATERIAL

10 record(s) selected.
```

交叉参考：连接运算符和函数将分别在第 10 章和第 11 章中讨论。

在 SQL Server 中运行这个查询时，要用一个加号运算符（+）代替连接运算符（||）。

作为 SELECT 语句的一部分，函数和表达式能够进行插入或更新操作。假设要记录每个插入和更新操作的日期/时间标记，下面的 Oracle 11g 语法将 SELECT 查询作为输入，使用 SYSDATE 和 USER 函数实现该功能。

```
SELECT    salesman_code_s,
          Salesman_name_s,
          sysdate,
          user
FROM      salesman;

SA SALESMAN_NAME_S                    SYSDATE   USER
-- ---------------------------------- --------- -------
02 FAIRFIELD BUGS ASSOCIATION         15-OCT-07 ACME
03 AMERICA GONZALES LIMITED           15-OCT-07 ACME
04 HUNTER COMPONENTS                  15-OCT-07 ACME
07 ELMERSON INDUSTRIES INCORPORATED   15-OCT-07 ACME
09 SAM KRISTEL INC                    15-OCT-07 ACME
```


www.TopSage.com

```
10 HENERY INCORPORATED                15-OCT-07 ACME

6 rows selected.
```

DB2 语法实现相同功能的语句如下：

```
SELECT    salesman_code_s,
          Salesman_name_s,
          current_date,
          current_user
FROM salesman
```

为了在 Microsoft SQL Server 中得到相似结果，可以使用如下语句：

```
SELECT    salesman_code_s,
          Salesman_name_s,
          getdate(),
          user_name()
FROM salesman
```

8.2.4 在 SELECT 子句中使用子查询

子查询的概念很简单—— 一个查询嵌套在另一个查询内部，并且第二个查询为第一个查询提供必要数据。SELECT 查询使用嵌套的子查询检索未知数据。嵌套级（在一个查询中可以嵌套的子查询的数量）仅受 RDBMS 容量的限制。

为说明子查询的概念，下面的例子使用 SALES_TAX 表(示例数据库 ACME 只包含部分州的数据)，该表包含每个州税额和两个字母的邮政编码缩写。

如果要生成一个报表，其中包含产品 ID、价格和华盛顿州单位产品的税额，可以使用如下查询：

```
SELECT  prod_num_s,
        Prod_price_n,
        (SELECT stax_amt_n
         FROM sales_tax
         WHERE stax_state_s = 'WA') AS TAX_RATE,
        prod_price_n *
        (SELECT stax_amt_n
         FROM sales_tax
         WHERE stax_state_s = 'WA')/100 AS SALES_TAX
FROM    product
```

```
PROD_NUM_S PROD_PRICE_N          TAX_RATE      SALES_TAX
---------- ------------ -------- ---------     ----------------------
990                18.24          6.500          1.18560000000000000000
1880               33.28          6.500          2.16320000000000000000
2871               26.82          6.500          1.74330000000000000000
3045               15.92          6.500          1.03480000000000000000
4000               11.80          6.500          0.76700000000000000000
4055               16.03          6.500          1.04195000000000000000
4761               23.10          6.500          1.50150000000000000000
4906               27.00          6.500          1.75500000000000000000
4964               23.10          6.500          1.50150000000000000000
5786               17.90          6.500          1.16350000000000000000

  10 record(s) selected.
```

本例中以华盛顿州的缩写（WA）作为参数查询适用的州税额，使用子查询从 SALES_TAX 表中提取该值，在运行时对该值进行计算。注意数据检索涉及结果集中的每一行。

8.3 FROM 子句：从哪里选取？

对于每个 SELECT 语句，FROM 子句都是强制性的——除了本章前面提到的 Microsoft SQL Server 情况。数据库对象能够从表和视图中选取数据。包括多种形式：临时表、内联视图和物化视图等。实际上，从 RDBMS 中只能选取表和视图。

8.3.1 从表和视图中选取

本章前面的例子（包括前几章）使用表和视图作为“可选择的”对象。实际上，在 RDBMS 中真正可选择的对象是表，而视图是基于某些表的查询。根据定义，视图和表不同，当执行 SELECT 查询时，视图不真正包含数据而只是从基表（一个或多个）中收集数据。

www.TopSage.com

注意：Oracle也竞相提出了物化视图的概念。常规视图是基于从基础表中提取数据的查询；该查询在对视图执行SELECT语句时才执行。为加快选取过程，可能要创建一个物化视图，视图中实际上包含要读取数据的快照。这样的视图与表没有什么区别。这里需要牢记的是，物化视图中的数据可能不是最新的，需要刷新以确保最新（刷新机制在物化视图创建期间或使用ALTER MATERIALIZED VIEW命令启动）。

8.3.2 在FROM子句中使用别名

在一个SELECT查询中，对FROM子句中的表名可以使用别名。这样可以缩短标记，而且在其他子句中使用表别名作为列名前缀时更直观。这里给出一个例子，它从数据库ACME的STATUS表中选择三列，表STATUS的别名为s：

```
SELECT status_id_n,
       s.status_code_s,
       s.status_desc_s
FROM   status s
```

SELECT子句中的列可以用（或不用）表别名（或表名）作为前缀；而且列本身也可以设置列别名，用有意义的名称代替含义模糊的列名（参阅本章前面的例子），增加可读性。如果使用这样的前缀，要用<object>.<property>标记表示，如上例所示。

尽管列别名对单表的选取用处不大，但当涉及多个表时，它将简化查询。而且，当两个或多个表具有相同列名时，别名可以帮助消除混淆。请参考第9章获得更多信息。

注意：在SELECT查询的FROM子句中使用表别名，这与第4章中的CREATE ALIAS语句不同，后者创建一个数据库对象，该对象以后能够被其他对象（不一定是一个表）引用，是一种更简便的方法。SELECT查询中的别名和该方法有相同目的：缩短调用标记，但是本质不同，因为它只涉及表（或视图），只在查询运行时存在，运行后便消失。

在SELECT查询的其他子句中，使用别名时有一定的规则。在FROM子句中使用别名的表，在整个查询中都要以该别名引用：SELECT、WHERE、GROUP BY和ORDER BY等。尽管在这些子句中，可以不用表名（或别名）限制列名，但建议使用别名（如果指定了的话）来消除歧义。

8.3.3 在FROM子句中使用子查询（内联视图）

第4章已经讨论过VIEW数据库对象，本节介绍称为**内联视图**的对象。与VIEW对象不同，内联视图在包含它们的查询之外不存在，可以为其命名（或不命名）。思考下面的语句，从一个内联视图CUST中选取客户ID、姓名和状态：

```
SELECT    cust.id,
          cust.cust_name_s,
          cust.active
FROM      (SELECT cust_id_n_AS id,
                  cust_status_s AS active,
                  cust_name_s,
                  cust_alias_s AS alias,
                  cust_credhold_s AS hold
          FROM    customer) cust

ID          CUST_NAME_S               ACTIVE
----------- ------------------------  ---------
51 DLH INDUSTRIES                     Y
5 FAIR AND SONS AIR CONDTNG           Y
12 KILBURN GLASS INDUSTRIES           Y
61 BOSWELL DESIGNS CORP.              Y
55 WILE ELECTROMATIC INC.             Y
6 FABRITEK INC.                       Y
   ...
16 DALCOMP INC.                       Y
89 INTEGRATED POWER DESIGNS           Y
85 GUARDIAN MANUFACTURING INC.        Y
152 WILE BESS COMPANY                 Y

37 record(s) selected.
```

请注意，外部SELECT子句通过内联视图中的列别名引用列——通过该方法传递列。用CUST代

www.TopSage.com

替 CUST.ID。使用 CUST_ID_N 的话会生产生一个无效的列名错误，因为外部查询无法以该列名引用列。不过，CUST_NAME_S 列在子查询中没有设置别名，就用原来的列名引用。

8.4 WHERE 子句：设置水平限制

尽管选择表或视图中的所有内容对于某些操作来说比较有价值，但是，多数情况下要查找的是一些特定信息——有特定电话号码的人或某一日期范围等。这些表中可能含有数百万行数据，完全没有时间自己查找所有的信息。SQL WHERE 子句提供了一种设置水平限制的机制，就是说通过指定一些限制条件或条件组，允许限制查询结果集中的行数。该行数取决于在查询中指定的条件，可能返回 0 条、1 条或多条记录（行）。在 WHERE 子句中指定的搜索标准结果为 TRUE 或 FALSE，布尔代数的所有规则在这里完全适用。

交叉参考：请参阅附录 K。

8.4.1 使用比较运算符

为了在 WHERE 子句中指定条件，SQL 使用了许多**运算符**，第 11 章将详细讨论这些运算符。本节只作简要介绍。

思考下面的查询，运行环境为 Microsoft SQL Server 的 ACME 数据库（在所有“三大”数据库中，语法和运行结果都是相同的），以客户 ID 字段的 CUST_ID_N 为限制条件，返回特定客户的信息。表 CUSTOMER 中的列有 UNIQUE 约束，客户 ID（不是地址记录的）的唯一性受主键约束（一个客户只有一个 ID）。

```
SELECT cust_id_n,
       cust_name_s,
       cust_status_s
FROM   customer
WHERE  cust_id_n = 7

CUST_ID_N     CUST_NAME_S                CUST_STATUS_S
------------- -------------------------- -------------
7             WILE ELECTRONICS INC.       Y

(1 row(s) affected)
```

因为只有一条记录满足条件，所以能准确定位记录。

现在，一个客户可以有一个或多个地址。下面的查询查找客户 ID 为 7 的客户的所有地址记录：

```
SELECT addr_address_s,
       addr_city_s,
       addr_state_s,
       addr_zip_s
FROM   address
WHERE  addr_custid_fn = 7

addr_address_s     addr_city_s             addr_state_s  addr_zip_s
------------------ ----------------------- ------------ ----------
411 S LONDON AVE   EGG HARBOR CITY         NJ            08215
232 EEL ST.        EGG HARBOR CITY         NJ            08215
454 OAK ST.        EGG HARBOR CITY         NJ            08215
456 WILLOW ST.     EGG HARBOR CITY         NJ            08215
678 MAPLE AVE.     EGG HARBOR CITY         NJ            08215
. . .               . . .                  . . .         . . .
865 CEDAR AVE.     EGG HARBOR CITY         NJ            08215
911 MYRTLE ST.     EGG HARBOR CITY         NJ            08215
777 SITKA AVE.     EGG HARBOR CITY         NJ            08215
999 ELK AVE.       EGG HARBOR CITY         NJ            08215

(11 row(s) affected)
```

该查询返回 11 条记录，包括客户号为 7 的客户经商的所有地址。

比较操作符也能确定一定范围的数值。如果想知道数据库中哪些产品价格超过 20 美元，则使用下面的查询：

```
SELECT prod_description_s,
```


www.TopSage.com

大家网 TopSage.com

```
        prod_price_n
FROM    product
WHERE   prod_price_n > 20

PROD_DESCRIPTION_S                        PROD_PRICE_N
----------------------------------------- ------------
STEEL NAILS 6"                                   33.28
STOOL CAPS 5"                                    26.82
BAR RAILS 24X48X128                              23.10
BAR RAILS 30X45X60                               27.00
BASES 30X45X60                                   23.10

(5 rows affected)
```

8.4.2 复合运算符：使用 AND 和 OR

查询中也许会指定多个条件。例如：如果想要查询公司所有销售人员的电话号码，PHONE 表中不只包含销售人员的电话号码，还有客户的电话号码。而且除了电话号码之外，还包含传真号码。为了得到想要的记录，可以使用下面的查询：

```
SELECT          phone_salesmanid_fn,
                phone_phonenum_s,
                phone_type_s
FROM            phone
WHERE           phone_salesmanid_fn IS_NOTNULL
AND             phone_type_s = 'PHONE'

PHONE_SALESMANID_FN PHONE_PHONENUM_S       PHONE_TYPE_S
------------------- ---------------------- --------------------
                 23 (305) 555-8502         PHONE
                 24 (626) 555-4435         PHONE
                 25 (717) 555-5479         PHONE
                 26 (718) 555-7879         PHONE
                 27 (718) 555-5091         PHONE
                 28 (814) 555-0324         PHONE

6 record(s) selected.
```

结果集中只包含 PHONE_SALESMANID_FN 列不为空的且号码类型为 PHONE 的列。

可以使用 OR 复合运算符选取记录。在下面的例子中，选取客户 ID 为 63 的订单或订单号为 30661 的订单：

```
SELECT    ordhdr_id_n,
          ordhdr_custid_fn
FROM      order_header
WHERE     ordhdr_id_n = 30661
OR        ordhdr_custid_fn = 63

ORDHDR_ID_N ORDHDR_CUSTID_FN
----------- ----------------
30613       63
30661       1

2 records(s) selected.
```

这提供了指定完全不同的选取条件的功能，订单号为 30661 的客户 ID 不是 7，但它仍出现在记录列表中，因为您指定您对客户 63 下的订单或订单号为 30661 的订单感兴趣，而不管客户 ID 是什么。在本例中如果使用 AND 运算符，将不会返回结果，因为订单号为 300661 的订单并不属于客户 ID 为 7 的用户，同时满足两个条件的记录不存在。

8.4.3 使用 BETWEEN 运算符

尽管可以结合使用≥（大于等于）和≤（小于等于）运算符达到完全一样的结果，但是，BETWEEN 运算符提供了一种更方便（通常更高效的）的选取一定范围值的方法。

```
SELECT    prod_description_s,
          prod_price_n
FROM      product
WHERE     prod_price_n BETWEEN 23.10 AND 30

PROD_DESCRIPTION_S                        PROD_PRICE_N
----------------------------------------- ------------
STOOL CAPS 5"                                    26.82
```


www.TopSage.com

```
BAR RAILS 24X48X128                          23.10
BAR RAILS 30X45X60                           27.00
BASES 30X45X60                               23.10

4 record(s) selected.
```

请注意，其中包括指定范围内的所有值，也就是说，用于指定范围的边界值也包含在最终结果集中。这个行为是特定于数字的，如果条件中使用其他的数据库类型（例如，字符、日期型），结果可能就会不同。该操作符几乎可以用于所有数据类型：日期、字符、数字等。

8.4.4 使用 IN 运算符：集成员身份测试

当 WHERE 子句中有多个精确条件，而且这些条件不匹配任何值范围时，可以用 OR 语句。思考下面的查询：

```
SELECT    cust_name_s,
          cust_credhold_s
FROM      customer
WHERE     cust_alias_s = 'MNGA71396' OR
          cust_alias_s = 'MNGA71398' OR
          cust_alias_s = 'MNGA71400'

CUST_NAME_S                          CUST_CREDHOLD_S
------------------------------------ ---------------
WILE SEAL CORP.                      Y
MAGNETICS USA INC.                   N
MAGNETOMETRIC DEVICES INC.           N

3 record(s) selected.
```

符合三个指定条件的记录显示在结果集中。使用 IN 运算符更容易得到该结果：

```
SELECT    cust_name_s,
          cust_credhold_s
FROM      customer
WHERE     cust_alias_s IN ('MNGA71396', 'MNGA71398', 'MNGA71400')

CUST_NAME_S                    CUST_CREDHOLD_S
------------------------------ ------------------
WILE SEAL CORP.                Y
MAGNETICS USA INC.             N
MAGNETOMETRIC DEVICES INC.     N

3 record(s) selected.
```

使用运算符 IN 替换大量 OR 语句，可以使形式更简单，而且提高了查询速度。另外，将子查询返回的多个结果值联合时，该方法非常实用。下面的查询返回最近持有信用证的客户下的订单信息：

```
SELECT    ordhdr_id_n,
          ordhdr_nbr_s,
          ordhdr_invoicenbr_n,
          ordhdr_invoicedate_d
FROM order_header
WHERE ordhdr_custid_fn IN ( SELECT cust_id_n
                            FROM customer
                            WHERE cust_credhold_s = 'Y')

ORDHDR_ID_N ORDHDR_NBR_S  ORDHDR_INVOICENBR_N  ORDHDR_INVOICEDATE_D
----------- ------------ -------------------- --------------------
      30613 523726       -                    -
      30661 523774       345799               08/21/2007
      30662 523775       345800               08/21/2007
      30663 523776       345802               08/21/2007
      30664 523777       345801               08/21/2007
      30665 523778       -                    -
      30666 523779       345803               08/21/2007
      30667 523780       -                    -
      30668 523781       -                    -
      30669 523782       -                    -
      30670 523783       -                    -

11 record(s) selected.
```

在本章后面的部分，您将了解更多在 WHERE 子句中使用子查询的内容。

Note 注意：在一个 IN 运算符中指定的所有值必须是同一数据类型。因为它们引用同一列。

www.TopSage.com

8.4.5 NOT运算符

NOT运算符是取反运算，即查找那些与指定条件相反的结果。如果使用NOT，类似的任何运算和查询都将产生相反的结果。下面的例子返回不满足指定条件（名称的第二个字母为I，第三个为L，第4个为E）的所有结果，那些从第二个字母开始的顺序不满足规则的公司名称记录被选择出来：

```
SQL> SELECT   cust_name_s
       FROM    customer
       WHERE   cust_name_s NOT LIKE '_ILE%'

cust_name_s
--------------------------------------------------
MAGNETICS USA INC.
MAGNETOMETRIC DEVICES INC.
FAIR PARK GARDENS
FAIR AND SONS AIR CONDTNG
. . .
KILBURN GLASS INDUSTRIES
CARLTONBATES COMPANY
DABAH BROS INC.
. . .
INSULECTRO INC.
INTEGRATED POWER DESIGNS
EASTERN SATELLITE COMPANY

32 record(s) selected
```

8.4.6 使用IS NULL运算符：NULL值的专用测试

我们在前面提到过，关系数据库中使用一个特定值NULL来表示数据库表的列中不存在数据。由于它不遵循所有其他值满足的规则（例如，比较、运算等），因此不能被相等/比较运算符（=）识别，也就是说，对于语法WHERE<column_name>=NULL，在Oracle中（与在Microsoft SQL Server中一些情况相同）尽管技术上是有效的，但是不会返回任何数据，因为等式的结果总为FALSE。

注意："WHERE <column_name>=NULL"语法在DB2中返回一个错误。

NULL的测试采用IS关键字。下面的例子检索有电话的销售人员的信息：

```
SELECT    phone_salesmanid_fn,
          phone_phonenum_s,
          phone_type_s
FROM      phone
WHERE     phone_custid_fn IS NULL

PHONE_SALESMANID_FN PHONE_PHONENUM_S       PHONE_TYPE_S
------------------- -------------------- --------------------
                 23 (305) 555-8502       PHONE
                 24 (626) 555-4435       PHONE
                 25 (717) 555-5479       PHONE
                 26 (718) 555-7879       PHONE
                 27 (718) 555-5091       PHONE
                 28 (814) 555-0324       PHONE
                 23 (305) 555-8501       FAX
                 24 (626) 555-4434       FAX
                 25 (717) 555-5478       FAX
                 26 (718) 555-7878       FAX
                 27 (718) 555-5091       FAX
                 28 (814) 555-0323       FAX

12 record(s) selected.
```

注意：在Microsoft SQL Server 2008的Transact-SQL中，除非在SQL Server中通过SET ANSI_NULLS OFF命令明确规定，否则一个NULL值与另一个NULL值永不相等。将命令中的参数设置为OFF后，就可以对两个NULL值进行比较。将其设回ON（默认）就又遵循SQL:2003标准了。

可以很容易地用NOT运算符测试特定列中不存在NULL值。根据列中的值是否为NULL，IS NULL测试返回FALSE或TRUE。使用时遵循英语语法：不能用NOT IS NULL（会产生一个错误），需要用IS NOT NULL返回期望值。下面的查询返回有电话（因此，PHONE_CUSTID_FN字段的值不为空）

www.TopSage.com

TopSage.com

的客户记录：

```
SELECT    phone_custid_fn,
          phone_phonenum_s,
          phone_type_s
FROM      phone
WHERE     phone_custid_fn IS NOT NULL

PHONE_CUSTID_FN PHONE_PHONENUM_S          PHONE_TYPE_S
--------------- ------------------------- ---------------
              1 (909) 555-9957            PHONE
              1 (909) 555-9956            FAX
              2 (501) 555-5702            PHONE
              2 (501) 555-5701            FAX
            ...
            152 (541) 555-8979            PHONE
            152 (541) 555-8978            FAX

74 records
```

8.4.7 在 WHERE 子句中使用子查询

在 SELECT 子句中，带有 WHERE 子句的子查询可以查找不存在的值（或值集合）。例如：在表 ORDER_HEADER 中无法只用客户名字查找信息，因为该表包含的是客户 ID 而不是名字。但是，在表 CUSTOMER 中使用客户名字作为条件，可以找到客户 ID，然后再用客户 ID 在表 ORDER_HEADER 中选取数据：

```
SELECT    ordhdr_nbr_s,
          ordhdr_orderdate_d
FROM      order_header
WHERE     ordhdr_custid_fn =
(SELECT   cust_id_n
 FROM     customer
 WHERE    cust_name_s = 'WILE ELECTRONICS INC.')

ordhdr_nbr_s       ordhdr_orderdate_d
-------------- -------------------------
523733         2007-08-15 00:00:00.000
523734         2007-08-15 00:00:00.000
523735         2007-08-15 00:00:00.000

(3 row(s) affected)
```

子查询先执行，并为外部查询提供选取的值集合。本例中，期望得到单一匹配值。如果由于某些原因，子查询返回多行数据（那么多个值与 ORDHDR_CUSTID 字段匹配），将会生成一个错误。

本章前面讨论的运算符都能用在子查询中，但要注意运算符所需的数据类型：IN 运算符接受一定范围的值，而 LIKE 计算的是单一值。例如：下面的查询使用 IN 运算符来计算子查询的返回值：

```
SELECT    phone_phonenum_s,
          phone_type_s
FROM      phone
WHERE     phone_salesmanid_fn IN
(SELECT  salesman_id_n
FROM     salesman
WHERE    salesman_code_s BETWEEN '07' and '10')

PHONE_PHONENUM_S        PHONE_TYPE_S
----------------------- ----------------
(718) 555-7879          PHONE
(718) 555-5091          PHONE
(814) 555-0324          PHONE
(718) 555-7878          FAX
(718) 555-5091          FAX
(814) 555-0323          FAX

6 record(s) selected.
```

当子查询返回一组可能值，外部查询使用运算符对其进行比较，该查询如何区分这些值呢？SQL 引入了 ANY 和 ALL 两个运算符来进行判断。

ACME 数据库中的一个视图包含 WILE BESS COMPANY 公司的全部订单。创建该视图是为了说明如何使用这两个运算符。思考下面的查询：

```
SELECT    v_total.customer_name,
          v_total.total_price
```


www.TopSage.com

```
FROM      v_customer_totals v_total
WHERE     v_total_total_price >
  ANY (SELECT_total_price
       FROM_v_customer_totals_wilebess)
ORDER BY total_price_ASC

CUSTOMER_NAME                      TOTAL_PRICE
---------------------------------- --------------
WILE ELECTROMATIC INC.             7511.00
WILE BESS COMPANY                  7511.00
WILE ELECTROMATIC INC.             7799.20
WILE BESS COMPANY                  8390.00
WILE BESS COMPANY                  8390.00
. . .                              . . .
WILE SEAL CORP.                    15456.80
WILE ELECTRONICS INC.              15456.80

29 rows selected.
```

查找那些价格全部高于 WILE BESS COMPANY 的订单（也就是说，大于子查询返回的集合中的任意值）。这意味着外部查询选取的记录应该大于任意值——实际上是指要大于子查询返回的最小值（也就是 6608.00）。结果集按升序排列使结果更直观。下面的查询会产生同样的结果：

```
SELECT    v_total.customer_name,
          v_total.total_price
FROM      v_customer totals_v_total
WHERE     v_total.total_price >
  (SELECT      MIN(total_price)
   FROM        v_customer_totals_wilebess)
ORDER BY total_price ASC;
```

在这里，子查询返回一个单一值——TOTAL_PRICE 的最小值，该值可以与外部查询的单一值进行比较。

使用 ALL 运算符将会产生完全不同的记录，外部查询会选择 TOTAL_PRICE 大于全部值的记录——即大于子查询返回值中的最大值。结果按降序排列，强调由外部查询返回的视图 V_CUSTOMER_TOTAL 中，TOTAL_PRICE 列中的最小值都大于子查询的最大值：

```
SELECT    v_total.customer_name,
          v_total.total_price
FROM      v_customer_totals_v_total
WHERE     v_total.total_price >
   ALL (SELECT total_price
        FROM_v_customer_totals_wilebess)
ORDER BY total_price DESC

CUSTOMER_NAME                  TOTAL_PRICE
------------------------------ ---------------
WILE ELECTRONICS INC.          15456.8
WILE SEAL CORP.                15456.8

2 rows selected.
```

在子查询中使用 MAX 函数可以得到相同的结果，通过比较，只有一个值（即 15402.20）被返回：

```
SELECT    customer_name,
          total_price
FROM      v_customer_totals
WHERE     total_price > (SELECT MAX(total_price)
                         FROM_v_customer_totals_wilebess)
ORDER BY total_price DESC;
```

嵌套子查询

一个子查询可以包含在另一子查询中，这称为嵌套子查询。在理论上对嵌套层数（一个查询位于另外一个查询内的次数）没有限制，但是一些供应商对其进行了限制。使用子查询查找信息需要很大开支（从计算机资源的角度来讲），因此要慎用。

例如：要查找曾经订购过 CRATING MATERIAL 12x48x72 产品的所有客户，要使用下面的查询：

```
SELECT    cust_name_s,
          cust_alias_s
FROM      customer
WHERE     cust_id_n IN
  (SELECT ordhdr_custid_fn
   FROM order_header
   WHERE ordhdr_id_n IN
     (SELECT ordline_ordhdrid_fn
      FROM order_line
```


www.TopSage.com

```
      WHERE ordline_prodid_fn =
       (SELECT_prod_id n
        FROM product
        WHERE prod_description_s = 'CRATING MATERIAL 12X48X72'
        )
    )
)

CUST_NAME_S                              CUST_ALIAS_S
---------------------------------------- -------------
WILE ELECTROMATIC INC.                    ECEL71460
WILE BESS COMPANY                         BSSE71641

2 rows selected.
```

在本例中，首先执行的是最高层——第三层的查询，然后执行第二层，最后是第一层。当收集完所有必要的数据后，就开始执行主查询。

还有一种子查询特例——**相关子查询**。它与常规子查询的不同之处在于，外部子查询（宿主查询）的值可以作为参数用在内部子查询中。相关子查询在本章讨论组合查询结果集（即 INTERSECT 和 MINUS 关键字）的一节中进一步讨论。

8.5 GROUP BY 和 HAVING 子句：汇总结果

根据一些条件对结果集中的记录进行分组，可以为表中的数据提供一种有价值的信息。例如：如果想看到订单的最终结果集（每个订单可能有多个订购项目）不以录入时的随机顺序排列，而是将属于同一订单的项目分为一组：

```
SELECT    ordline_ordhdrid_fn,
          ordline_ordqty_n AS QTY_PER_ITEM
FROM      order_line
GROUP BY ordline_ordhdrid_fn,
          ordline_ordqty_n;

ordline_ordhdrid_fn QTY_PER_ITEM
------------------- ------------
30607                90
30607                500
30608                275
30608                340
30609                560
. . .                . . .
30666                560
30667                560
30668                72
30668                90
. . .                . . .
30669                120
30669                480
. . .                . . .
30670                126
30670                450

97 records selected.
```

请注意，ORDLINE_ORDHDRID_FN 字段中有重复值，这表明订单标题 ID 可以有一个或多个订单项，而且订单项的数目不同。该信息使聚合函数的使用更加简单，如对订单数量求和或者计算平均数等（参阅本节后面有关 SUM 和 AVG 的例子）。

警告：在使用 GROUP BY 对结果集中的记录排序时，一些 SQL 开发人员有一个有问题的习惯，这有时是实用的，例如，当使用 ORDER BY 子句（在下一节讨论）非法时，问题出在这不是一个受 RDBMS 支持的功能。例如：它无法在 Oracle 10g 以后的版本中运行。对于其他也可能在将来更改 GROUP BY 算法的 RDBMS 供应商而言，这是一个好机会。因此，在 SQL 程序中，永远不要使用 GROUP BY 来对数据进行排序。

注意：在 SELECT 子句列表中指定的所有列，除了聚合列（它们是聚合函数的参数）之外，其他列必须都列在 GROUP BY 子句中。

GROUP BY 子句通常（并不总是）与聚合函数一起使用，聚合函数在第 10 章介绍。聚合函数对一

www.TopSage.com

组值进行运算，并返回一个单一值作为结果。集合被分成多个组分别进行聚合计算。

第10章给出了5个聚合函数的说明，以及在各RDBMS中的特定实现。对于具体函数使用的例子，请参考第10章。本节主要讨论聚合的概念，以及如何将它与SELECT查询的GROUP BY子句配合使用。

为计算表ORDER_LINE中的所有订单（一个订单可能有一个或多个订单项）的数量总和，可以用SUM聚合函数：

```
SELECT SUM(ordline_ordqty_n) AS TOT_QTY

FROM    order_line;
TOT_QTY
--------
31847

1 record selected.
```

对表中所有记录的订购数量计算总和得到一个值，这也许是有用的。但是，对每一个订单的订购数量求和也许会更有价值——那样就会知道在每个订单中共有多少订单项。下面的查询完成该任务：

```
SELECT    ordline_ordhdrid_fn,
          SUM(ordline_ordqty_n) AS TOT_QTY_PER_ORDER
FROM      order_line
GROUP BY ordline_ordhdrid_fn

ORDLINE_ORDHDRID_FN     TOT_QTY_PER_ORDER
---------------------- ------------------
            30607                     590
            30608                     615
            30609                     560
            ...                       ...
            30669                     600
            30670                     720

51 rows selected.
```

这里给出了更加具体的信息，因为根据订单对数量进行了分组，并对每个分组求和，针对每个订单生成一个值（而不是针对整个结果集）。

下面是另外一个聚合函数AVG（计算平均值）的例子。在本例中，计算每个订单的平均数量：

```
SELECT    ordline_ordhdrid_fn,
          AVG(ordline_ordqty_n) AS AVG_QTY_PER_ORDER
FROM      order_line
GROUP BY ordline_ordhdrid_fn

ORDLINE_ORDHDRID_FN AVG_QTY_PER_ORDER
------------------- -----------------
              30607               295
              30608             307.5
              30609               560
              ...                 ...
              30669               300
              30670               180

51 rows selected.
```

注意：在Oracle中，#30608订单的平均数量为307.5，在Microsoft SQL Server和DB2中对其四舍五入得到307，因为订单号是INTEGER数据类型，而Oracle使用NUMBER数据类型表示十进制。参考第3章获得更多信息。

HAVING子句是GROUP BY的专用语句，提供了一种附加的选取方法。如果不需要选取所有GROUP BY查询中的记录，而是那些分组值大于750的记录该怎么办呢。对WHERE子句添加一个条件是无济于事的，因为在查询没有执行完以前，通过GROUP BY计算的记录值在外部查询中不可用。HAVING子句用在GROUP BY子句中，对GROUP BY运算结果添加附加条件。例如，要显示总量大于750的订单，可以用下面的查询：

```
SELECT    ordline_ordhdrid_fn,
          SUM(ordline_ordqty_n) TOT_QTY_PER_ORDER
FROM      order_line
GROUP BY ordline_ordhdrid_fn
HAVING   SUM(ordline_ordqty_n) > 750

ORDLINE_ORDHDRID_FN TOT_QTY_PER_ORDER
------------------- -----------------
              30628               789
```


www.TopSage.com

```
30668                789

2 records selected.
```

可以看到，从前面查询得到的 51 条记录中只有两行满足附加条件限制。

Note 注意：尽管 GROUP BY 会考虑列中的 NULL 值，并对该列进行运算可以得到一个有效的分组，但是，聚合函数并不考虑 NULL 值，而是将其排除在外，不会在结果集中显示 NULL 值。参阅第 10 章获得更多信息。

8.6 ORDER BY 子句：对查询结果排序

查询的结果是满足条件的，但未经排序——也就是说，结果和表中的顺序是一样的。为结果排序（按字母或数字顺序）需要使用 ORDER BY 子句。该子句在“三大”数据库中的功能相同。

下面的查询将输出结果按客户名字字母顺序进行升序排列：

```
SQL> SELECT   cust_name_s,
              cust_alias_s,
              cust_status_s
FROM          customer
ORDER BY      cust_name_s;

cust_name_s                    cust_alias_s       cust_status_s
------------------------------ ---------------- ------------
BOSWELL DESIGNS CORP.          BWSO71471          Y
BURNETTE WILLIAM CORP.         BNRU71477          Y
CARLTONBATES COMPANY           CLRA71445          Y
CHGO SWITCHBOARD INC.          COGH71478          N
COFFMAN DOUGLAS                CFFO71482          Y
. . .
WILE ELECTRONICS INC.          ISCC71419          Y
WILE SEAL CORP.                MNGA71396          Y

37 rows selected.
```

结果既可以按升序排列，也可以按降序排列。按降序排列，必须在列名之后指定 DESC 关键字。按升序排列，可以使用 ASC 关键字（或者像前面的查询一样，完全省略关键字，因为默认的次序是升序）。

ORDER BY 子句中的列不一定在 SELECT 子句中出现，但是它一定存在于表中：

```
SELECT   cust_name_s,
         cust_alias_s
FROM     customer
ORDER BY cust_status_s
```

在该查询中，结果按 CUST_STATUS_S 列排序，此列没有被选取出来显示。

可以同时按多个列进行排序，结果也许并不明显。ORDER BY 子句中列的优先顺序是很重要的：首先，结果按第一列的值进行排序，然后，对已经按第一列的顺序排好的结果，根据第二列进行排序。

还可以对 ORDER BY 子句中的不同列指定不同的顺序（升序或降序）。下面的例子对 PRODUCT 表中的记录进行排序，首先按价格进行升序排列，然后按净重对**每个价格**进行降序排列。

```
SELECT   prod_id_n,
         prod_price_n,
         prod_netwght_n
FROM     product
ORDER BY prod_price_n ASC,
         prod_netwght_n DESC

PROD_ID_N     prod_price_n  prod_netwght_n
------------- ------------- -----------
4000          11.80         15.194
. . .         . . .         . . .
990           18.24         21.226
4964          23.10         18.480
4761          23.10         10.032
2871          26.82         34.552
4906          27.00         22.000
1880          33.28         38.391

10 rows selected
```

在查询的返回结果中（价格按升序排列），恰好有两个产品（PROD_ID_N=4964 和 PROD_ID_N=4761）

www.TopSage.com

有相同的价格——23.10美元。可以看到两个产品的PROD_NETWGHT_N列的值是按降序排列的。

ORDER BY和NULL值

不同的供应商对含NULL值的列使用ORDER BY子句的行为是不同的。SQL:2003标准没有规定如何对NULL值排序，将它排列在非NULL值的上方或下方。Oracle和DB2认为NULL值大于非NULL值，我们认为这是有根据的。因为根据定义，NULL值的ASCII码值大于其他可打印字符。但Oracle允许在ORDER BY表达式后使用NULLS FIRST或NULLS LAST（默认）改变默认行为：

```
SELECT    addr_address_s,
          addr_type_s
FROM      address
ORDER BY addr_type_s NULLS_FIRST

ADDR_ADDRESS_S                                ADDR_TYP
--------------------------------------------- ----------
223 E FLAGLER ST.
2710 SANTA ANITA AVE.
52 GRUMBACHER RD
167 CLYMER ST.
3518 CHURCH AVE.
2820 E COLLEGE AVE.
1165 WALNUT AVE.                              BILLING
...
454 OAK ST.                                   SHIPPING
232 EEL ST.                                   SHIPPING
332 WALNUT ST.                                SHIPPING

53 rows selected.
```

Microsoft SQL Server认为NULL值小于非NULL值。

“三大”RDBMS供应商都允许在ORDER BY子句中使用序数，也就是说，在ORDER BY子句中可以使用数字来代替列名，这个数字是在查询的SELECT语句中列出现的次序。根据这些规则，前面的查询可以重新改为：

```
SELECT    prod_id_n,
          prod_price_n,
          prod_netwght_n
FROM      product
ORDER BY 2 ASC,
          3 DESC
```

在ORDER BY子句中，用顺序号代替列名使用时，所有的列必须在SELECT语句中出现（与上一段内容比较一下）。

ORDER BY子句一般与GROUP BY表达式一起使用，也可以对一个计算列排序。下面的查询说明了这一点，该查询为计算了平均值的列赋予别名，并进行排序：

```
SELECT    ordline_ordhdrid_fn,
          AVG(ordline_ordqty_n) AS AVG_QTY_PER_ORDER
FROM      order_line
GROUP BY ordline_ordhdrid_fn
ORDER BY AVG_QTY_PER_ORDER DESC

ORDLINE_ORDHDRID_FN AVG_QTY_PER_ORDER
------------------- -----------------
              30610               700
              30619               700
              30650               700
              ...
              30668             157.8
              30622             145.2
              30662             145.2

51 rows selected.
```

8.7 联合多个查询的结果

可以联合两个或多个查询来生成一个结果，联合结果集可能是查询的所有记录的一个简单聚合，或者是最终结果集返回前，进行的一些与集合论（参阅附录K）有关的运算。

SQL:2003标准支持使用UNION、INTERSECT和EXCEPT子句合并两个或多个查询的结果。

www.TopSage.com

TopSage.com

8.7.1 UNION

下面的查询返回所有包含客户电话的记录：

```
SELECT    phone_custid_fn OWNER_ID,
          'CUSTOMER PHONE' PHONE_TYPE,
          phone_phonenum_s
FROM      phone
WHERE     phone_type_s = 'PHONE'
AND       phone_salesmanid_fn IS NULL

OWNER_ID       PHONE_TYPE     phone_phonenum_s
-------------- -------------- ------------------
51             CUSTOMER PHONE (817) 555-5524
5              CUSTOMER PHONE (847) 555-2924
12             CUSTOMER PHONE (508) 555-5224
61             CUSTOMER PHONE (541) 555-3091
. . .          . . . . . .
16             CUSTOMER PHONE (201) 555-9302
89             CUSTOMER PHONE (908) 555-3779
85             CUSTOMER PHONE (281) 555-2835
152            CUSTOMER PHONE (541) 555-8979

37 records selected
```

该查询共返回 37 条记录。现在，假设要在结果集中包含销售人员电话列表，下面的查询查找这些销售人员，返回了 6 条记录：

```
SELECT    phone_salesmanid_ fn,
          'SALESMAN PHONE',
          phone_phonenum_s
FROM      phone
WHERE     phone_type_s = 'PHONE'
AND       phone_custid_fn IS NULL

phone_salesmanid_fn                  phone_phonenum_s
-------------------- ---------------- ------------------
23                     SALESMAN PHONE (305) 555-8502
24                     SALESMAN PHONE (626) 555-4435
25                     SALESMAN PHONE (717) 555-5479
26                     SALESMAN PHONE (718) 555-7879
27                     SALESMAN PHONE (718) 555-5091
28                     SALESMAN PHONE (814) 555-0324

6 records selected
```

为将这些记录合并到一个结果集中，需要使用 UNION 语句：

```
SELECT    phone_custid_fn OWNER_ID,
          'CUSTOMER PHONE' PHONE_TYPE,
          phone_phonenum_s
FROM      phone
WHERE     phone_type_s = 'PHONE'
AND       phone_salesmanid_fn IS NULL
UNION
SELECT    phone_salesmanid_fn,
          'SALESMAN PHONE',
          phone_phonenum_s
FROM      phone
WHERE     phone_type_s = 'PHONE'
AND       phone_custid_fn IS NULL
ORDER BY 2, 1

OWNER_ID PHONE_TYPE       PHONE_PHONENUM_S
---------- -------------- --------------------
         1 CUSTOMER PHONE  (909) 555-9957
         2 CUSTOMER PHONE  (501) 555-5702
         3 CUSTOMER PHONE  (520) 555-5513
         4 CUSTOMER PHONE  (802) 555-2091
        ...
        26 SALESMAN PHONE  (718) 555-7879
        27 SALESMAN PHONE  (718) 555-5091
        28 SALESMAN PHONE  (814) 555-0324

43 rows selected.
```

现在，查询出了有关客户的所有记录的列表，并结合了关于销售人员的查询记录。可以形象地认为两个结果集粘合在一起。所有 SQL 语句中的查询（包括 UNION 运算符）的列表中必须有相等数量的表达式。

www.TopSage.com

另外，这些表达式（可以是列、常量或函数结果等）必须是兼容的数据类型。例如，如果一个查询中表达式的值为字符串，那么通过UNION运算符与第一个查询联合时，第二个查询中就不能是数字。

UNION的结果可以排序（就像上例的UNION查询那样），但是ORDER BY子句只用于最终结果集。即对UNION的结果可以用ORDER BY子句，但其中的某一查询不能用。

如果查询可能产生重复记录，就需要筛选重复记录，或者相反，要确保它们能全部显示。默认情况下，UNION运算符不包括重复记录。指定UNION ALL可以保证最终结果集中包括UNION联合的查询返回的所有记录。

下面是一个简单查询，它从ACME数据库的STATUS表中返回状态代码和描述：

```
SELECT    status_code_s,
          status_desc_s
FROM      status

ST STATUS_DESC_S
-- ------------------------------
60 SHIPPED
20 COMPLETE
70 INVOICED
80 CANCELLED

4 rows selected.
```

可以使用完全相同的查询来产生一个结果集，其中包含重复行，并用UNION合并这些结果集：

```
SELECT    status_code_s,
          status_desc_s
FROM      status
UNION
SELECT    status_code_s,
status    statuw_desc_s
FROM      status

ST STATUS_DESC_S
-- ------------------------------
20 COMPLETE
60 SHIPPED
70 INVOICED
80 CANCELLED

4 rows selected.
```

可以看到，重复记录（即在所有相应列中有相同值的那些记录）被排除了。而且最终结果集中仍然是4行记录，这与仅用一个查询的结果是相同的。采用UNION ALL运算符（覆盖这个行为），运行相同的查询返回两个查询的所有记录，不排除重复值：

```
SELECT    status_code_s,
          status_desc_s
FROM      status
UNION ALL
SELECT    status_code_s,
          status_desc_s
FROM      status

ST STATUS_DESC_S
-- ------------------------------
60 SHIPPED
20 COMPLETE
70 INVOICED
80 CANCELLED
60 SHIPPED
20 COMPLETE
70 INVOICED
80 CANCELLED

8 records selected.
```

注意：UNION运算符中的ALL关键字将商业世界引入到我们的讨论中：UNION ALL并不符合集合论中的幂等率A∪A=A（参阅附录K），简单规定一个集合与自身的并集是集合本身。从商业角度来看，将查询的所有记录都合并到最终结果集中更加直观。

8.7.2 INTERSECT

INTERSECT运算符用于两个查询的结果，只包括第一个查询中与第二个查询匹配的记录。本书

www.TopSage.com

中讨论的三大 RDBMS 供应商都实现了该运算符。

考虑下面的查询，从 ACME 数据库的 CUSTOMER 表中选取客户 ID（CUST_ID_N 字段），并与第二个查询的结果进行交叉，产生下订单的客户 ID：

```
SELECT    cust_id_n
FROM      customer
INTERSECT
SELECT    ordhdr_custid_fn
FROM      order_header

CUST_ID_N
------------
           1
           7
          55
          63
         152

5 records selected
```

可以用多种方法得到与此相同的结果。下面的例子使用子查询和 IN 运算符：

```
SELECT    cust_id_n
FROM      customer
WHERE     cust_id_n IN
(SELECT   ordhdr_custid_fn
 FROM     order_header)

CUST_ID_N
----------
           1
           7
          55
          63
         152

5 records selected
```

该查询也可以使用 EXISTS 运算符重新改写：

```
SELECT    cust_id_n
FROM      customer
WHERE EXISTS
   (SELECT  ordhdr_custid_fn
   FROM     order_header
   WHERE    ordhdr_custid_fn = cust_id_n)

cust_id_n
-----------
1
7
55
63
152

(5 row(s) affected)
```

Note 注意：EXISTS 关键字在“三大”数据库中通用。

尽管可以用其他 SQL 方法组合来代替 INTERSECT，但是 INTERSECT 能增加 SQL 代码的透明度。

8.7.3 EXCEPT（MINUS）

当把两个或多个查询的结果合并为一个结果集时，可能需要从第一个查询中排除一些记录，这些记录基于第二个查询的返回结果。在 DB2 9.5 中使用 EXCEPT 关键字实现这一点，而 Oracle 使用 MINUS 关键字，在 Microsoft SQL Server 中，EXCEPT 和 MINUS 都可识别。运算功能完全遵循集合论的规则，集合论在附录 K 中讨论。

思考以下两个查询，两者都选取一些关于客户的信息。下面是第一个查询（返回所有客户）返回的结果集：

```
SELECT    cust_id_n
FROM      customer

cust_id_n
-----------
```

大家网 TopSage.com

```
61
64
14
65
. . .
152
. . .
6
7
1

37 rows selected
```

第一个查询返回37行，下面是第二个查询（返回所有下订单的客户）：

```
SELECT    ordhdr_custid_fn
FROM      order_header

ordhdr_custid_fn
----------------
NULL
NULL
1
1
. . .
1
7
7
. . .
152

(51 row(s) affected)
```

EXCEPT运算的结果是第一个查询的所有记录**减去**第二个查询返回的结果，确切地说，下面的查询返回的是所有客户中除去下订单的客户：

```
SELECT    cust_id_n
FROM      customer
EXCEPT
SELECT    ordhdr_custid_fn
FROM      order_header;

cust_id_n
-----------
61
. .
68
69
15
. . .
9
12
. . .
2
3

(32 row(s) affected)
```

运算后返回32条记录，第二个查询中的5条记录从第一个查询返回的记录中剔除。这里因为篇幅缩短了三个查询的输出结果，但是，您可以在选择的RDBMS中运行查询，幸运的话，结果会符合我们的描述。

也可以使用NOT EXISTS运算符。奇怪的是，这个语法尽管不属于SQL:2003标准，但是在“三大”数据库中是通用的。要得到NOT EXISTS运算符的结果，需要用到**相关**子查询。相关子查询与常规子查询不同，它将外部查询的参数用作WHERE子句中的条件：

```
SELECT    cust_name_s,
          cust_alias_s,
          cust_status_s
FROM      customer_cust
WHERE NOT EXISTS (SELECT *
                  FROM order_header
                  WHERE ordhdr_custid fn = cust.cust_id_n)

cust_name_s                    cust_alias_s       cust_status_s
---------------------------- ------------------ -----------
MAGNETICS USA INC.             MNGA71398            Y
MAGNETOMETRIC DEVICES INC.     MNGA71400            Y
FAIR PARK GARDENS              NULL                 Y
```


www.TopSage.com

```
FAIR AND SONS AIR CONDTNG          FRIA71406          Y
. . .                               ...               ...
INSULECTRO INC.                    IUSN71521          Y
INTEGRATED POWER DESIGNS           IETN71523          Y
EASTERN SATELLITE COMPANY          ETSA71626          Y

(32 row(s) affected)
```

在本例中，使用 CUSTOMER 表中的 CUST_ID_N 字段作为子查询中的条件来限制潜在的记录数目。如果在 ORDER_HEADER 表中有客户 ID（ORDHDR_CUSTID_FN=CUST.CUST_ID_N）的记录，那么该记录将被排除在外部查询的结果集之外。只有那些没有下订单的客户记录被返回。

交叉参考：关于相关查询的详细信息请参阅第 9 章。

8.8 SQL 分析函数和 TOP N 查询

对 SQL 开发人员而言，编写一个查询从一个表中（或从多个表中）返回前 N（或倒数 N）行是不常见的。例如，也许要查找前 10 位高收入的雇员信息，或 15 种最畅销的产品，或者联合订单数量最多的 5 名客户。执行这些任务的查询就称为 **top N 查询**。

8.8.1 分析函数和 SQL:2003 标准

“三大”RDBMS 供应商都有自己实现 top N 查询的方法，而且语法各不相同。幸运的是，SQL:2003 引进了称为窗口函数（也经常称为分析函数）的新实体：ROW_NUMBER ()、RANK () 和 DENSE_RANK ()。这些函数有很多共同点，它们经常与 ORDER BY 语句一起使用，而且为结果集中的行指定数字 1 到 N。这些函数之间的不同之处在于，ROW_NUMBER () 指定的数字是连续的，也是唯一的；RANK () 指定的数字可以既不连续也不唯一；DENSE_RANK () 产生的数字是连续的，但不一定是唯一的。为了说明这些概念，从 PRODUCT 表中选取数据，按 price 列降序排列，使用 ROW_NUMBER () 函数，结果如下：

```
SELECT    ROW_NUMBER() OVER (ORDER BY prod_price_n DESC) AS rownumber,
          prod_id_n,
          prod_description_s,
          prod_price n
FROM      product;

ROWNUMBER     PROD_ID_N PROD_DESCRIPTION_S                PROD_PRICE_N
------------- ------- ---------------------------------- ------------
    1         1880 STEEL NAILS 6"                             33.28
    2         4906 BAR RAILS 30X45X60                         27
    3         2871 STOOL CAPS 5"                              26.82
    4         4964 BASES 30X45X60                             23.1
    5         4761 BAR RAILS 24X48X128                        23.1
    6         990 SPRUCE LUMBER 30X40X50                      18.24
    7         5786 CRATING MATERIAL 12X48X72                  17.9
    8         4055 GAZEBOS 40X30X60                           16.03
    9         3045 STOOL CAPS 9"                              15.92
   10         4000 HAND RAILS ROUNDS 48X48X12                 11.8

10 rows selected.
```

每行指定的数字是连续的，也是唯一的。尽管实际上 4964 号和 4761 号产品有相同的价格 23.10 美元。

使用 RANK () 来代替 ROW_NUMBER () 会生成稍微不同的结果：

```
SELECT    RANK() OVER (ORDER BY prod_price_n DESC) AS rownumber,
          prod_id_n,
          prod_description_s,
          prod_price_n
FROM      product;

ROWNUMBER     PROD_ID_N PROD_DESCRIPTION_S          PROD_PRICE_N
------------- ------- ------------------------- - ----------
    1         1880 STEEL NAILS 6"                   33.28
    2         4906 BAR RAILS 30X45X60               27
    3         2871 STOOL CAPS 5"                    26.82
```


www.TopSage.com

```
   4          4964 BASES 30X45X60                    23.1
   4          4761 BAR RAILS 24X48X128               23.1
   6          990 SPRUCE LUMBER 30X40X50             18.24
   7          5786 CRATING MATERIAL 12X48X72         17.9
   8          4055 GAZEBOS 40X30X60                  16.03
   9          3045 STOOL CAPS 9"                     15.92
  10          4000 HAND RAILS ROUNDS 48X48X12        11.8

10 rows selected.
```

现在，产品4964和4761两行被指派相同的数字4，但是结果集中的下一条记录仍然编号为6，数字5没有用来为行编号。

使用DENSE_RANK（）又产生另外一种结果：

```
SELECT    DENSE_RANK() OVER (ORDER BY prod_price_n DESC) AS rownumber,
          prod_id_n,
          prod_description_s,
          prod_price_n
FROM      product;

ROWNUMBER     PROD_ID_N PROD_DESCRIPTION_S          PROD_PRICE_N
------------- --------- ----------------------------------- ---------------
    1         1880 STEEL NAILS 6"                    33.28
    2         4906 BAR RAILS 30X45X60                27
    3         2871 STOOL CAPS 5"                     26.82
    4         4964 BASES 30X45X60                    23.1
    4         4761 BAR RAILS 24X48X128               23.1
    5         990 SPRUCE LUMBER 30X40X50             18.24
    6         5786 CRATING MATERIAL 12X48X72         17.9
    7         4055 GAZEBOS 40X30X60                  16.03
    8         3045 STOOL CAPS 9"                     15.92
    9         4000 HAND RAILS ROUNDS 48X48X12        11.8

10 rows selected.
```

与产品4964和产品4761对应的行仍然编号为4，但是，现在产品990记录编号为5，下一行被编号为6。

8.8.2 top N查询设计

有人会问，如何与top N查询相关？答案很简单，关联很直接。使用前一节中引入的查询作为top N查询的子查询。例如，如果要用一个查询返回ACME销售的4种最昂贵的产品，可以使用下面的SQL语句（使用Oracle 11g语法）：

```
SELECT *
FROM (SELECT ROW_NUMBER() OVER (ORDER BY prod_price_n DESC) AS
rownumber,
             prod_id_n,
             prod_description_s,
             prod_price_n
     FROM    product
    )
WHERE rownumber <= 4;

ROWNUMBER     PROD_ID_N PROD_DESCRIPTION_S          PROD_PRICE_N
------------- --------- ------------------------- ------------
    1         1880 STEEL NAILS 6"                   33.28
    2         4906 BAR RAILS 30X45X 60              27
    3         2871 STOOL CAPS 5"                    26.82
    4         4761 BAR RAILS 24X48X128              23.1

4 rows selected.
```

从上一节我们知道，有两件产品的价格为23.10美元，查询只返回其中一个。如果商业规则只要求返回4行数据，那么这样做是可行的。但是，如果不是，则需要用RANK（）或DENSE_RANK（）来代替ROW_NUMBER（）来修复这一问题：

```
SELECT *
FROM (SELECT DENSE_RANK() OVER (ORDER BY prod_price_n DESC) AS
rownumber,
             prod_id_n,
             prod_description_s,
             prod_price_n
     FROM    product
    )
WHERE rownumber <= 4;
```


www.TopSage.com

TopSage.com

```
ROWNUMBER      PROD_ID_N PROD_DESCRIPTION_S     PROD_PRICE_N
---------- ---------- ---------------------- ------------
  1           1880 STEEL NAILS 6"                 33.28
  2           4906 BAR RAILS 30X45X               60 27
  3           2871 STOOL CAPS 5"                  26.82
  4           4761 BAR RAILS 24X48X128            23.1
  4           4964 BASES 30X45X60                 23.1

5 rows selected.
```

很容易想到，要返回 4 种最便宜的产品，只需替换分类顺序即可：

```
SELECT *
FROM (SELECT DENSE_RANK() OVER (ORDER BY prod_price_n ASC) AS
rownumber,
              prod_id_n,
              prod_description_s,
              prod_price_n
     FROM     product
    )
WHERE rownumber <= 4;

ROWNUMBER     PROD_ID_N PROD_DESCRIPTION_S       PROD_PRICE_N
---------- ---------- ------------------------ ------------
   1          4000 HAND RAILS ROUNDS 48X48X12      11.8
   2          3045 STOOL CAPS 9"                   15.92
   3          4055 GAZEBOS 40X30X60                16.03
   4          5786 CRATING MATERIAL 12X48X72       17.9

4 rows selected.
```

Note 注意：要使本节的所有查询能够在 DB2 9.5 和 Microsoft SQL Server 2008 中运行，需要使用命名查询（Oracle 不支持）。因此，第一个例子应该写成这样：

```
SELECT *
FROM (SELECT ROW_NUMBER() OVER (ORDER BY prod_price_n DESC) AS
rownumber,
              prod_id_n,
              prod_description_s,
              prod_price_n
     FROM     product
    ) AS dummy
WHERE rownumber <= 4;
```

8.8.3 Limit N 查询

有时需要设计一个查询返回从第 N 行到第 M 行，而不是前 N 行或倒数 N 行。例如，如果想每页显示 N 条记录时，这个查询是很有帮助的。下面的查询假设每页显示三行，第二页的查询为：

```
SELECT *
FROM (SELECT ROW_NUMBER() OVER (ORDER BY prod_price_n ASC) AS
rownumber,
              prod_id_n,
              prod_description_s,
              prod_price_n
     FROM     product_p
    )
WHERE rownumber BETWEEN 4 AND 6;

ROWNUMBER     PROD_ID_N PROD_DESCRIPTION_S      PROD_PRICE_N
---------- ---------- ---------------------- ------------
   4          5786 CRATING MATERIAL 12X48X72       17.9
   5           990 SPRUCE LUMBER 30X40X50          18.24
   6          4761 BAR RAILS 24X48X128             23.1

3 rows selected.
```

上例的语法适用于 Oracle 11g，为了在 DB2 9.5 或 Microsoft SQL Server 2008 中运行查询，需要像前一节说明的那样，使用命名子查询。

特定于供应商的 Top N 查询实现

如本章前面所述，三大 RDBMS 供应商各自有自己的 top N 查询方法。语法各不相同。大多数情况下，这些方法和 SQL:2003 标准比较，缺乏灵活性。因此，对于所有新的开发项目，我们强烈推荐使用后者。但是，很多传统应用程序仍然使用特定于供应商的语法。因此，至少应意识到它们的存在。

www.TopSage.com

Oracle的伪列ROWNUM默认用唯一且连续的数字为查询的每行编号；除了实际的列之外，也可以选取它。下面的例子显示如何用ROWNUM代替ROW_NUMBER（）OVER:

```
SELECT *
FROM (SELECT  prod_id_n,
              prod_description_s,
              prod_price_n
      FROM product
      ORDER BY prod_price_n DESC
     )
WHERE ROWNUM <= 4;
```

该语法看起来更简单，输出结果也相同，但注意使用ROWNUM列来执行RANK()或DENSE_RANK（）函数不能产生相同结果。

DB2有生成top N查询的专用方法，在SELECT语句之后，使用FETCH FIRST n ROW ONLY子句:

```
SELECT    prod_id_n,
          prod_description_s,
          prod_price_n
FROM      product
ORDER BY prod_price_n DESC
FETCH FIRST 4 ROWS ONLY
```

另外，从9.5版开始，DB2通过将DB2_COMPATIBILITY_VECTOR注册表变量设为一个特定的数值（如本章开头所述），就可以识别Oracle的ROWNUM（ROW_NUMBER（）OVER的同义词）。

Microsoft SQL Server语法生成top N查询的方法是，TOP N子句紧跟在SELECT关键字之后:

```
SELECT TOP 4  prod_id_n,
              prod_description_s,
              prod_price_n
FROM          product
ORDER BY      prod_price_n DESC
```

8.9 小结

SELECT语句是整个结构化查询语言中最重要的语句，也是数据查询语言（DQL）的唯一语句，提供了提取数据的方法。

本章介绍了单表SELECT查询的重要概念，以及它的所有构成子句：SELECT、FROM、WHERE、GROUP BY和ORDER BY。

查询只是提取数据和修改提取的数据，但不会改变基础表中的数据。对结果集的运算一般遵循数学上的集合论规则，了解集合论的基础知识有助于理解结果。

查询结果可以按照一定的条件进行排序和分组，聚合函数通常用在GROUP BY子句中。

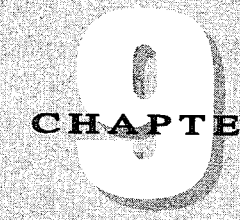
可以使用UNION、INTERSECT或EXCEPT运算符将查询结果合并为一个结果集。不同的RDBMS供应商对其中一些运算符的实现方式有所不同。

子查询可以作为常规SELECT查询的一部分，为了动态检索信息，可以进行嵌套。用其他的方法，很难或根本不可能做到。

在设计top N和Limit N查询时，分析函数ROW_NUMBER（）、RANK（）和DENSE_RANK（）是非常有用的。top N查询的语法最终得到标准化，实际上，现在“三大”数据库中都能够通用。

www.TopSage.com

第9章
多表查询



本章提要：

- 简单内部联接
- 交叉联接
- 自然联接
- 自联接
- 右联接、左联接和完全外部联接
- 合并联接
- 包含内联视图和相关查询的联接
- 提高多表查询的性能

前一章介绍了简单查询，或叫做单表查询。然而，如果只从关系数据库的一个表中查找数据，就会与 RDBMS 的整个思想和规范化规则相矛盾。要得到有意义的结果，通常要在单个查询内从多个表检索信息。所有 RDBMS 实现过程允许基于一个公共列（或多列），联接两个或多个表中的数据，这个（些）列出现在两个表中（列名相同或不同）。例如（考虑一下 ACME 数据库），表 ORDER_HEADER 和表 CUSTOMER 可以通过 ORDHDR_CUSTID 列和 CUST_ID_N 列进行联接。

注意：前面的规则只是逻辑上的限制，您完全可以通过两个不相关的列将两个表联接起来（例如，将表 ORDER_HEADER 和表 PRODUCT 通过 ORDHDR_ID_N 列和 PROD_ID_N 列进行对应联接），但是，这样联接的结果完全没有意义。

所有的联接一般分为两大类：内部联接和外部联接。

交叉参考：集合论基本原理的知识有助于更好地理解表联接的原理。请参阅附录 K。

9.1 内部联接

本节讨论内部联接，它返回两个联接表中有匹配值的那些行，排除其他行。

9.1.1 内部联接的两个语法

在 SQL 中，对于表联接有两个不同的语法：一个是 SQL 标准委员会在 1992 年提出的；另外一个是“旧”的，1992 年以前开始使用，现在仍然在使用。随着 Microsoft SQL Server 7.0、IBM DB2 7.1 的出现，新的标准语法在 20 世纪 90 年代末期站稳了脚跟。最终迎来了 Oracle 9i 版本，它除了支持“旧”语法之外，还支持“新”SQL 语法标准。

SQL:2003 定义的语法使用 SELECT 语句的 FROM 子句来联接表：

```
...
FROM <table1>
[INNER | NATURAL | CROSS] JOIN
      <table2>
[ON <condition>] | [USING <column name>,...],...
...
```

www.TopSage.com

大家网 TopSage.com

Note 注意：SQL:2003 仍然使用“旧”语法，但同时也为应用程序编写者提供了与之不同的、更加明确的语法。

本章后面部分将会给出实例，现在，我们先理解所有可选子句的意思。

1. 内部联接选项

在一个内部联接中，有下列选项：

- INNER 关键字是可选的，它用于明确区分内部联接和外部联接。
- NATURAL 关键字用于指定两个表之间的自然联接，即通过名称相同的列将两个表联接起来。NATURAL 关键字不能同时调用 ON 和 USING 子句，在“三大”RDBMS 中，只有 Oracle 11g 可以使用 NATURAL 关键字。有关自然联接更详细的信息将在本章后面部分讨论。
- CROSS 关键字用于交叉联接，本章后面将详细讨论。该关键字符合 Oracle 11g 和 Microsoft SQL Server 2008 语法，但不被 DB2 9.5 语法识别。

2. ON 和 USING 子句

ON 子句指定联接条件（等值联接或非等值联接，在本章后面部分讨论），“三大”数据库供应商的语法中都包含该子句。

指定一个自然联接（本章后面讨论）时，USING 子句指示要使用的列。当自然联接无法正常工作时，也将用到 USING 子句。例如，相同名称的列数多于联接时实际用到的列数。例如，假设表 A 和表 B 的公共字段是 CUST_ID、PROD_ID 和 ORDER_ID，但是，只想通过 CUST_ID 和 PROD_ID 字段进行联接，可以指定下列内容：

```
...
FROM A JOIN B
USING (CUST_ID, PROD_ID)
...
```

Note 注意：在 USING 子句中，不用表名限定列名。

在“三大”数据库供应商中，只有 Oracle 11g 支持 USING 子句，但用 ON 子句也可以得到相同结果：

```
...
FROM A JOIN B
ON A.CUST_ID = B.CUST_ID
AND A.PROD_ID = B.PROD_ID
...
```

3. 旧语法

每个供应商（Oracle 9i 以后的版本只识别“标准”语法，Microsoft SQL Server 7.0 引进它）通常都使用另一个语法在 WHERE 子句执行联接：

```
...
WHERE [<qualifier>.]<column_name>
       <join_condition>
      [<qualifier>.]<column_name>
[AND  [<qualifier>].<column_name>
       <join_condition>
      [<qualifier>.]<column_name>],...
...
```

在同等联接中，join_condition 可以是等号（=）。在非同等联接中，join_condition 是“不等于”、“大于”、“小于”、“大于等于”或“小于等于”（≠、>、<、≥或≤）。下面解释术语“同等联接”和“非同等联接”。

9.1.2 同等联接

同等联接是一种最常用的表联接类型，两列之间使用完全匹配。例如，在 ACME 数据库的 CUSTOMER 表中不包含客户电话号码的信息，表 PHONE 的 PHONE_CUSTID_FN 字段是 CUSTOMER 表的外键。为了显示两个表的信息，使用同等联接，如图 9-1 所示。

1. SQL：2003 标准语法

下列 SQL:2003 兼容语法产生图 9-1 后半部分所示的输出结果：

```
SELECT        cust_id_n,
              cust_name_s,
```


www.TopSage.com

```
                phone_phonenum_s,
                phone_type_s
FROM            customer JOIN phone
ON              cust_id_n = phone_custid_fn
```

CUSTOMER

cust_name_s	cust_id_n
WILE SEAL CORP.	1
MAGNETICS USA INC.	2
MAGNETOMETRIC DEVICE...	3
FAIR PARK GARDENS	4
FAIR AND SONS AIR CO...	5

PHONE

phone_custid_fn	phone_phonenum_s	phone_type_s
1	(909) 394-9957	PHONE
1	(909) 394-9956	FAX
2	(501) 674-5702	PHONE
2	(501) 674-5701	FAX
3	(520) 885-5513	PHONE
3	(520) 885-5512	FAX
4	(802) 888-2091	PHONE
4	(802) 888-2091	FAX
5	(847) 885-2924	PHONE
5	(847) 885-2923	FAX

CUSTOMER JOIN PHONE ON CUST_ID_N = PHONE_CUSTID_FN

cust_id_n	cust_name_s	phone_phonenum_s	phone_type_s
1	WILE SEAL CORP.	(909) 394-9957	PHONE
1	WILE SEAL CORP.	(909) 394-9956	FAX
2	MAGNETICS USA INC.	(501) 674-5702	PHONE
2	MAGNETICS USA INC.	(501) 674-5701	FAX
3	MAGNETOMETRIC DEVICE...	(520) 885-5513	PHONE
3	MAGNETOMETRIC DEVICE...	(520) 885-5512	FAX
4	FAIR PARK GARDENS	(802) 888-2091	PHONE
4	FAIR PARK GARDENS	(802) 888-2091	FAX
5	FAIR AND SONS AIR CO...	(847) 885-2924	PHONE
5	FAIR AND SONS AIR CO...	(847) 885-2923	FAX

图 9-1 简单内等值联接

2. 自然联接

自然联接是一种在 RDBMS 中执行的同等联接特例，基于下面的假设："总是通过名称相同的一列（或多列）联接表"。换言之，在 SELECT 语句的 FROM 子句中，不需要在 ON 分子句中显式指定列，完全省略分子句 ON。

注意：自然联接的思想与准"匈牙利"表示法概念相矛盾（参阅附录 B）。在 ACME 数据库中，每列都以缩略表名为前缀。所以，数据库中的每个列名都是唯一的，这样就无法使用自然联接。对于主键和外键是否应该共享同一名称，是有争议的。执行自然联接的可能性是这种情况（仅在使用 Oracle 时）的唯一优势，所以在 ACME 数据库中，我们要一直使用这种表示法。

交叉参考：ACME 数据库的命名约定见附录 B。

自然联接可以使查询的编写过程更容易，但是，前提是数据库结构和查询本身足够简单，否则，会产生不合乎要求的联接和不正确的查询结果。

完全限定列名

如果在查询中，有两个或多个表引用一个或多个列名，必须将表名或表别名作为列名的前缀，否则，RDBMS 将不能消除歧义而生成一条错误信息。

显式地限定列引用是一种好的方法，然而，在前面提到的 ACME 示例数据库中，每列的列名都是唯一的（也就是说，在一定程度上已经完全限定），这样就没有必要使用完全限定列名。下面的例子使用和上例相同的查询说明如何使用完全限定列名：

```
SELECT          customer.cust_id_n,
                customer.cust_name_s,
                phone.phone_phonenum_s,
                phone.phone_type_s
FROM            customer JOIN phone
ON              customer.cust_id_n = phone.phone_custid_fn
```

在后面的例子中可随机使用完全限定名。

3. 旧语法

旧语法是在 SELECT 语句的 WHERE 子句中联接表，该语法也有支持者和反对者。反对的主要观

www.TopSage.com

点是 WHERE 子句中不能混合联接和谓词，它的唯一目的应是对产生的结果集设置“垂直”限制。支持者的主要观点是对于联接很多表的查询，旧语法更具可读性。另外一个主要争论点是遗留代码，尤其是对于 Oracle，在 9i 版本之前，标准语法都是不可用的。可以选择任何一种语法（建议所有新开发产品与 SQL:2003 标准兼容），但熟悉两种语法总是好的。

下面的语句与前一节的语句相同，只是采用旧语法：

```
SELECT      c.cust_id_n,
            c.cust_name_s,
            p.phone_phonenum_s
FROM        customer_c,
            phone_p
WHERE       c.cust_id_n = p.phone_custid_fn
```

9.1.3 非同等联接

有时需要基于条件而不是等同性来联接表。同等联接的典型应用是处理主键/外键关系，而非同等联接则不同，非同等联接通常会以一种无意义的方式填充结果集。例如，如果用<>（不等于）代替前一节查询中的=（等于）运算符，那么查询结果将返回所有客户和电话号码的组合，而不包括那些实际上表示有效客户/电话号码的列表（图 9-2）。换言之，结果和笛卡尔积（交叉联接）有些相似，笛卡尔积将在本章后面讨论。

但是，在某些情况下，非同等联接是非常有用的。例如，假设 ACME 公司设置了以下商业规则：10 000 美元以下的订单没有折扣；总量在 10 000 美元和 14 999 美元之间的订单，享受 2%的折扣；总量在 15 000 美元和 19 999 美元之间的，有 3%的折扣；最后，20 000 美元以上的订单享有 4%的折扣。

实现该商业规则的方法之一是建立一个表，表中存放最小和最大订单数，以及对应的折扣百分比。如图 9-3 所示。

CUSTOMER JOIN PHONE ON CUST_ID_N <> PHONE_CUSTID_FN

cust_id_n	cust_name_s	phone_phonenum_s	phone_type_s
1	WILE SEAL CORP.	(817) 236-5524	PHONE
1	WILE SEAL CORP.	(847) 885-2924	PHONE
1	WILE SEAL CORP.	(508) 761-5224	PHONE
1	WILE SEAL CORP.	(541) 994-3091	PHONE
1	WILE SEAL CORP.	(541) 336-3246	PHONE
1	WILE SEAL CORP.	(305) 822-8802	PHONE
1	WILE SEAL CORP.	(760) 355-2335	PHONE
1	WILE SEAL CORP.	(619) 424-6635	PHONE
...	...	...	...
1	WILE SEAL CORP.	(847) 885-2923	FAX
1	WILE SEAL CORP.	(508) 761-5223	FAX
1	WILE SEAL CORP.	(541) 994-3091	FAX
1	WILE SEAL CORP.	(541) 336-3245	FAX
1	WILE SEAL CORP.	(305) 822-8801	FAX
1	WILE SEAL CORP.	(760) 355-2334	FAX
1	WILE SEAL CORP.	(619) 424-6634	FAX
...	...	...	...
152	WILE BESS COMPA	(847) 885-2924	PHONE
152	WILE BESS COMPA	(508) 761-5224	PHONE
152	WILE BESS COMPA	(541) 994-3091	PHONE
152	WILE BESS COMPA	(541) 336-3246	PHONE
152	WILE BESS COMPA	(305) 822-8802	PHONE
152	WILE BESS COMPA	(760) 355-2335	PHONE
...	...	...	...

图 9-2 主键/外键非等值联接

DISCOUNT

DISC_MINAMOUNT_N	DISC_MAXAMOUNT_N	DISC_PCT
0	9999	0
10000	14999	2
15000	19999	3
20000	1000000000	4

图 9-3 DISCOUNT 表的内容

Tip

提示：把可能要改变的值（例如，订单数量超过 20 000 美元的折扣可能从 4%变为 3.5%）存放在一个 RDBMS 表中，而不是将其硬编码到应用程序中，这是一个好方法。当商业规则改变时，只需更新表内的值即可。与改变应用程序代码、重新编译应用程序等方法相比，这更快捷、更轻松。

下面的例子显示了视图 V_CUSTOMER_TOTALS（显示了用订单行数量计算出的每个订单的总额）和表 DISCOUNT 之间的非等值联接。查询通过订单总量乘以根据适当的百分比折扣（这个百分比折扣是根据前面描述的规则得出的）计算出每个总额（TOTAL_DISCOUNT）的适当折扣。

SQL:2003 standard syntax

```
SELECT    ct.order_number,
```


www.TopSage.com

TopSage.com

```
        ct.total_price,
        ct.total_price * d.disc_pct / 100 AS total_discount
FROM    v_customer_totals ct JOIN discountd
ON      ct.total_price >= d.disc_minamount_n
AND     ct.total_price < d.disc_maxamount_n

ORDER_NUMBER               TOTAL_PRICE    TOTAL_DISCOUNT
-------------------------- ------------- ---------------
523774                           6608                0
523778                           6608                0
...
523777                          10010             200.2
523737                          10010             200.2
523730                        10915.8           218.316
...
523781                        15402.2           462.066
523741                        15402.2           462.066
523775                        15456.8           463.704
523735                        15456.8           463.704

51 rows selected.
```

Old syntax

```
SELECT  order_number,
        total_price,
        total_price * disc_pct / 100 AS total_discount
FROM    v_customer_totals ct,
        discount d
WHERE   ct.total_price > d.disc_minamount_n
AND     ct.total_price < d.disc_maxamount_n
```

Note 注意：表之间可以通过多个列进行联接，本例通过联接 V_CUSTOMER_TOTALS 表的 TOTAL_PRICE 列和 DISCOUNT 表的 DISC_MAXAMOUNT_N 列，找出指定范围适合的折扣百分比。多列联接的另外一个典型示例是必须联接组合主键/外键的情况。

9.1.4 自联接

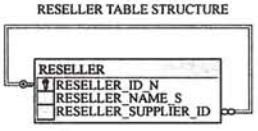
自联接可能是最不直观的 SQL 概念之一，尽管听起来很简单——一个表不与其他表而与自身进行联接。但是，有时会引起混乱。

这里要理解的一个重点是，尽管实际上表与自身进行联接，但是处理的仍然是同一表的**两个实例**，或者说是两个相同的表而不只是一个表，所以，自联接可以认为是多表联接的一种特殊情况。

Note 注意：理解一个表和它实例的不同之处也是很重要的。两个表实例可以认为是表的数据被载入两个不同的存储单元，彼此完全分离。

假设 ACME 公司要打开多个层次的市场，开始时，它将产品卖给中间商，中间商将产品卖给下层的中间商，或直接卖给客户。这表示现在任何 ACME 的客户可以是一个中间商（可以买卖产品的人），或者是供应商（可以从他那里买产品），或者两者兼具。表 RESELLER 中已经实现了这种关系。ACME 公司的 RESELLER_SUPPLIER_ID 列中有 NULL 值，因为它位于树的顶端，没有供应商。该字段的其他列都有另外一个实体的（在 RESELLER 表中）主键值。

图 9-4 说明了这一概念以及多个表实例的思想。



RESELLER R
(FIRST INSTANCE)

RESELLER_ID_N	RESELLER_NAME_S	RESELLER_SUPPLIER_ID
1	ACME, INC	NULL
2	MAGNETICS USA INC.	1
3	MAGNETOMETRIC DEVICE...	1
4	FAIR PARK GARDENS	2
5	FAIR AND SONS AIR CO...	2
6	FABRITEK INC.	2
7	WILE ELECTRONICS INC.	3
8	INTEREX USA	3
9	JUDCO MANUFACTURING INC.	4
10	ELECTRO BASS INC.	5

RESELLER S
(SECOND INSTANCE)

RESELLER_ID_N	RESELLER_NAME_S	RESELLER_SUPPLIER_ID
1	ACME, INC	NULL
2	MAGNETICS USA INC.	1
3	MAGNETOMETRIC DEVICE...	1
4	FAIR PARK GARDENS	2
5	FAIR AND SONS AIR CO...	2
6	FABRITEK INC.	2
7	WILE ELECTRONICS INC.	3
8	INTEREX USA	3
9	JUDCO MANUFACTURING INC.	4
10	ELECTRO BASS INC.	5

图 9-4 自联接

1. SQL:2003 标准语法

下面的查询采用 SQL:2003 兼容的语法，使用自联接来检索中间商 ID 和名称以及供应商的 ID 和名称（RESELLER 表的每个实体）：

www.TopSage.com

```
SELECT    r.reseller_id_n    AS res_id,
          r.reseller_name_s  AS res_name,
          s.reseller_id_n    AS sup_id,
          s.reseller_name_s  AS sup_name
FROM      reseller r JOIN resellers
ON        r.reseller_supplier_id = s.reseller_id_n

RES_ID RES_NAME                      SUP_ID    SUP_NAME
------ ----------------------------- --------- -------------------------
     2 MAGNETICS USA INC.            1         ACME, INC
     3 MAGNETOMETRIC DEVICES INC.    1         ACME, INC
     4 FAIR PARK GARDENS             2         MAGNETICS USA INC.
     5 FAIR AND SONS AIR CONDTNG     2         MAGNETICS USA INC.
     6 FABRITEK INC.                 2         MAGNETICS USA INC.
     7 WILE ELECTRONICS INC.         3         MAGNETOMETRIC DEVICES INC.
     8 INTEREX USA                   3         MAGNETOMETRIC DEVICES INC.
     9 JUDCO MANUFACTURING INC.      4         FAIR PARK GARDENS
    10 ELECTRO BASS INC.             5         FAIR AND SONS AIR CONDTNG

(9 rows affected)
```

以上内容做如下可以解释：ACME 公司将产品直接卖给 MAGNETICS USA 公司和 MAGNETOMETRIC DEVICES 公司，MAGNETOMETRIC DEVICES 公司从 ACME 公司购买产品，然后卖给 WILE ELECTRONICS 公司和 INTEREX USA，依此类推。

2. 旧语法

下面的语句使用相同查询的等价旧语法：

```
SELECT    r.reseller_id_n    AS res_id,
          r.reseller_name_s  AS res_name,
          s.reseller_id_n    AS sup_id,
          s.reseller_name_s  AS sup_name
FROM      resellerr,
          resellers
WHERE     r.reseller_supplier_id = s.reseller_id_n
```

9.1.5 交叉联接（笛卡尔积）

两个表的交叉联接或笛卡尔积，可以定义为从两个源表中，选取可能的匹配行组成另一个表（可能是虚表）。回到本章非同等联接部分提到的客户电话号码例子，对表 CUSTOMER 和表 PHONE 进行交叉联接所返回的结果与图 9-2 非常相似，联接也返回了有效的客户/电话号码组合，排除了先前非同等联接中提到的数据。

1. SQL:2003 标准语法

该查询通过执行表 CUSTOMER 和表 PHONE 的交叉联接，返回客户姓名和电话号码所有可能的组合：

```
SELECT    cust_name_s,
          phone_phonenum_s
FROM      customer CROSS JOIN
phone

CUST_NAME_S                                      PHONE_PHONENUM_S
------------------------------------------------ --------------------
DLH INDUSTRIES                                   (817) 555-5524
FAIR AND SONS AIR CONDTNG                        (817) 555-5524
KILBURN GLASS INDUSTRIES                         (817) 555-5524
BOSWELL DESIGNS CORP.                            (817) 555-5524
WILE ELECTROMATIC INC.                           (817) 555-5524
FABRITEK INC.                                    (817) 555-5524
... ...
DLH INDUSTRIES                                   (847) 555-2924
FAIR AND SONS AIR CONDTNG                        (847) 555-2924
KILBURN GLASS INDUSTRIES                         (847) 555-2924
BOSWELL DESIGNS CORP.                            (847) 555-2924
WILE ELECTROMATIC INC.                           (847) 555-2924
FABRITEK INC.                                    (847) 555-2924
... ...
DLH INDUSTRIES                                   (814) 555-0323
FAIR AND SONS AIR CONDTNG                        (814) 555-0323
KILBURN GLASS INDUSTRIES                         (814) 555-0323
BOSWELL DESIGNS CORP.                            (814) 555-0323
WILE ELECTROMATIC INC.                           (814) 555-0323
FABRITEK INC.                                    (814) 555-0323
```


www.TopSage.com

```
...
3182 rows selected.
```

2. 旧语法

交叉联接的旧语法不过是省略了多表联接中的 WHERE 子句：

```
SELECT    cust_name_s,
          phone_phonenum_s
FROM      customer,
          phone
```

交叉联接是开销很大的运算，尤其是当发生了人为错误，或对那些包含几十万甚至数百万的行的大表联接时更是如此。因为笛卡尔积的结果集是两个表中所有行的每种可能的组合，比如说，第一个表中有 100 000 行数据，第二个表中有 200 000 行，那么查询将返回 20 000 000 000 行数据（100 000 * 200 000= 20 000 000 000），如此庞大的数据量将急剧减慢数据库服务器的速度，并影响所有当前登录的用户。

因此，笛卡尔乘积需要慎用，典型的 SQL 用户并不会经常使用它。为了生成测试数据或达到教育目的（例如，为了说明所有同等联接和非同等联接只是一个交叉联接满足一定条件的子集的概念），交叉联接是很有用的。大多数的现实情况是，笛卡尔积是人为错误导致的，需要 DBA 来修复（可能要终止用户的会话，如第 7 章所述）。

9.1.6 对两个以上的表进行联接

在关系数据库中，经常需要在一个查询中同时从多个表中检索数据，以获取所有必要的信息。在现实生活中，一个多表查询往往是指十几个表甚至上百个表的查询。SQL 程序设计技巧的 90%是基于内部数据库对象的结构知识，以及运用该知识的能力来联接多个表。

两个表的联接结果可以认为是一个新的虚表，该徐表可以在查询中与下一个表联接，产生另外一个虚表，依此类推。例如，如果想知道哪些客户订购了哪些产品，在查询中必须联接 4 个表——CUSTOMER、ORDER_HEADER、ORDER_LINE 和 PRODUCT：

```
SELECT DISTINCT    customer.cust_name_s,
                   product.prod_description_s
FROM               customer
   JOIN            order_header
   ON              customer.cust_id_n = order_header.ordhdr_custid_fn
   JOIN            order_line
   ON              order_header.ordhdr_id_n = order_line.ordline_
                   ordhdrid_fn
   JOIN            product
   ON              product.prod_id_n = order_line.ordline_prodid_fn
```

可以将该查询想象为联接的序列。首先，使用相应的列联接表 CUSTOMER 和表 ORDER_HEADER，然后将结果集与表 ORDER_LINE 进行联接，最后，将结果与表 PRODUCT 进行联接，结果如图 9-5 所示。

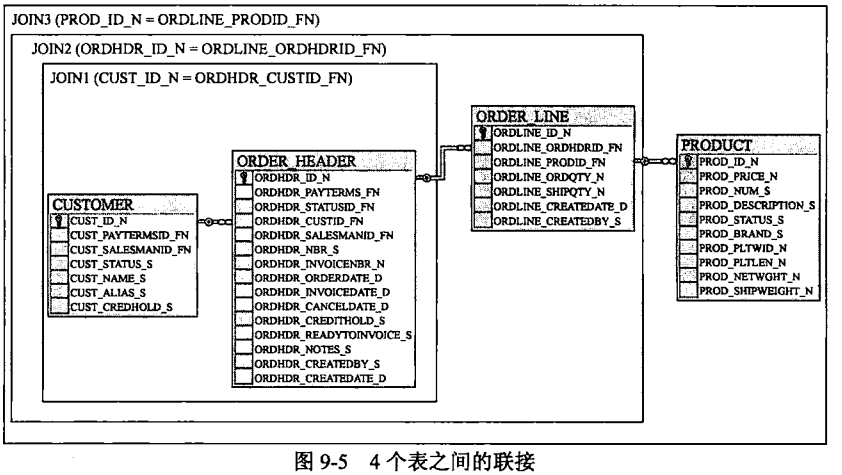


图 9-5 4 个表之间的联接

注意，尽管不需要表 ORDER_HEADER 或表 ORDER_LINE 的任何列，但由于表 CUSTOMER 和 PRODUCT 之间没有直接的联系，所以在查询联接时必须要使用这两个表。

大家网 TopSage.com

1. 联接数目

要记住的一个重要的规则是，在含 n 个表的查询中，应该至少有 n–1 个联接。因此，含 3 个表的查询至少要有 2 个联接；含 4 个表的查询至少要有 3 个联接，依此类推。“至少”这个词很重要，因为在非同等联接的查询中，或者在使用主键/外键组合来联接表的查询中，可能不止 n–1 个联接。但如果多表查询有少于 n–1 个的联接，结果将是一个笛卡尔积。

注意：最常见的（和最危险的）交叉联接类型发生在包含两个以上的表的查询中。在这种情况下，不必省略 WHERE 子句，只需要跳过十几个表中的两个表的联接就足够了。并且，在一个查询中联接的表越多，结果集就越庞大。每个表含 100 行，5 个表的笛卡尔积就会生成 10 000 000 000 行。

2. SQL:2003 标准语法

上例中的查询使用与 SQL:2003 兼容的语法，下面的查询仍然使用与 SQL:2003 兼容的语法，联接表 CUSTOMER、ORDER_HEADER 和表 STATUS，返回客户姓名、订单号和订单状态：

```
SELECT    cust_name_s,
          ordhdr_nbr_s,
          status_desc_s
FROM      customer
   JOIN
          order_header
   ON     cust_id_n = ordhdr_custid_fn
   JOIN
          status
   ON     status_id_n = ordhdr_statusid_fn

CUST_NAME_S             ORDHDR_NBR_S     STATUS_DESC_S
----------------------- -------------- -------------
WILE BESS COMPANY       523731           COMPLETE
WILE BESS COMPANY       523732           COMPLETE
WILE ELECTRONICS INC.   523733           COMPLETE
WILE ELECTRONICS INC.   523734           COMPLETE
WILE ELECTRONICS INC.   523735           COMPLETE
WILE BESS COMPANY       523736           COMPLETE
...                     ...              ...
WILE BESS COMPANY       523738           CANCELLED
WILE BESS COMPANY       523742           CANCELLED
WILE BESS COMPANY       523743           CANCELLED

51 rows selected.
```

3. 旧语法

下面是与上一查询等价的旧语法：

```
SELECT    cust_name_s,
          ordhdr_nbr_s,
          status_desc_s
FROM      customer,
          order_header,
          status
WHERE     cust_id_n = ordhdr_custid_fn
AND       status_id_n = ordhdr_statusid_fn
```

9.2 外部联接：根据包含 NULL 值的列联接表

您也许注意到了，在本章前面提到的 RESELLER 表中，除了 ACME 公司的记录，查询返回了所有其他表记录。这是因为在 RESELLER 表的 RESELLER_SUPPLIER_ID 列中，ACME 公司的记录存在 NULL 值，所以 RDBMS 不能从要联接的表中找到对应的值（本例中是 RESELLER 表的另一个实例）。结果，尽管实际表中有 10 行记录，查询结果只能返回 9 行。

这正是标准（内部）联接的工作原理。有时需要查询返回表 A 的所有行和表 B 中相应的行（如果有），这时，就要用到外部联接。

9.2.1 外部联接的两个语法

和内部联接一样，外部联接也有两种不同的语法。

www.TopSage.com

1. SQL:2003

与 SQL:2003 兼容的语法在 SELECT 语句的 FROM 子句中指示一个外部联接：

```
...
FROM <table1>
   {LEFT | RIGHT | FULL [OUTER]} | UNION JOIN
       <table2>
   [ON <condition>] | [USING <column name>,...],...
...
```

该语法得到三大 RDBMS 供应商的广泛支持。但是，只有 Oracle 实现了 USING 子句，“三大”数据库供应商语法中都不包括 UNION 子句（将在本章后面讨论）。

2. 旧语法

旧语法使用 WHERE 子句，但是 Oracle 和 Microsoft SQL Server 实现方式也不相同（DB2 没有用于外部联接的“旧”语法，它从一开始就使用与 SQL:2003 兼容的语法。实际上，后一个语法起源于 DB2 标准）。

Note 注意：DB2 从 9.5 版本开始，就一直支持 Oracle 外部联接的“旧”声明。为了使用该功能，需要将注册表变量 DB2_COMPATIBILITY_VECTOR 设为一个特定的数值，如第 8 章所述。

Oracle 11 g

旧的右外部联接或左外部联接，通过在表名之后添加一个（+）来表示，并且这个表在=的相应边没有匹配行。完全外部联接只能通过与 SQL:2003 兼容的语法指定，Oracle 有关右外联接和左外联接的旧语法如下所示：

Syntax for right outer join

```
...
WHERE [<qualifier>.]<column_name> =
      [<qualifier>.]<column_name> (+)
[AND  [<qualifier>.]<column_name> =
      [<qualifier>.]<column_name> (+)
],...
...
```

Syntax for left outer join

```
...
WHERE [<qualifier>.]<column_name> (+) =
      [<qualifier>.]<column_name>
[AND  [<qualifier>.]<column_name> (+) =
      [<qualifier>.]<column_name>
],...
...
```

Microsoft SQL Server 2008

Microsoft SQL Server 2008 不再支持外部联接的旧语法，可以通过将数据库兼容性级别设置为 80 或更小来使用旧语法，但不推荐使用。

如前所述，很多传统应用程序仍然在 Microsoft SQL Server 的早期版本中运行，所以熟悉旧语法是有用的。外部联接通过在等号（=）的一边（右边或左边）添加星号（*）表示。下面的旧语法可以用在 Microsoft SQL Server（2008 之前的版本）中连进行右/左外联接：

Syntax for right outer join

```
...
WHERE [<qualifier>.]<column_name> =*
      [<qualifier>.]<column_name>
[AND  [<qualifier>.]<column_name> =*
      [<qualifier>.]<column_name>
],...
...
```

Syntax for left outer join

```
...
WHERE [<qualifier>.]<column_name> *=
      [<qualifier>.]<column_name>
[AND  [<qualifier>.]<column_name> *=
      [<qualifier>.]<column_name>
],...
...
```


www.TopSage.com

9.2.2 左外部联接

实际上，“左外部联接”只是 SQL 程序员的一种习惯叫法。如本章后面演示的那样，可以使用左外部联接或右外部联接得到相同的结果。外部联接背后的总体思想是检索表 A（左）或表 B（右）的所有行，如果在对应的表中没有匹配的值，联接列便为 NULL。一个左（或右）外部联接对所有联接表中不匹配的列也返回 NULL 值（只包括那些含 NULL 联接列的行时）。

1. SQL:2003 标准语法

下面的查询说明如何使用 SQL:2003 兼容的左外部联接生成包含 RESELLER 表中的全部 10 行记录的结果集：

```
SELECT                  r.reseller_id_n    ASres_id,
                        r.reseller_name_s  AS res_name,
                        s.reseller_id_n    AS sup_id,
                        s.reseller_name_s  AS sup_name
FROM                    reseller r
  LEFT OUTER JOIN
                        reseller s
  ON                    r.reseller_supplier_id = s.reseller_id_n

RES_ID RES_NAME                        SUP_ID   SUP_NAME
------ ------------------------------- -------- --------------------
   1 ACME, INC.                        NULL     NULL
   2 MAGNETICS USA INC.                1        ACME, INC
   3 MAGNETOMETRIC DEVICES INC.        1        ACME, INC
   4 FAIR PARK GARDENS                 2        MAGNETICS USA INC.
   5 FAIR AND SONS AIR CONDTNG         2        MAGNETICS USA INC.
   6 FABRITEK INC.                     2        MAGNETICS USA INC.
   7 WILE ELECTRONICS INC.             3        MAGNETOMETRIC DEVICES INC.
   8 INTEREX USA                       3        MAGNETOMETRIC DEVICES INC.
   9 JUDCO MANUFACTURING INC.          4        FAIR PARK GARDENS
  10 ELECTRO BASS INC.                 5        FAIR AND SONS AIR CONDTNG

(10 rows affected)
```

这里再举一个例子，假设要检索客户 152 的姓名和他所有的订单数量，下面的（内部）联接将实现这一目的：

```
SELECT    cust_name_s,
          ordhdr_nbr_s
FROM      customer
  JOIN
          order_header
ON        cust_id_n = ordhdr_custid_fn
WHERE     cust_id_n = 152

CUST_NAME_S         ORDHDR_NBR_S
------------------- ------------
...                 ...
WILE BESS COMPANY   523731
WILE BESS COMPANY   523732
...                 ...

31 rows selected.
```

现在需要得到非常相似的结果，但还需要每个订单对应的付款条件。假设可以简单地通过联接 PAYMENT_TERMS 表来修改查询：

```
SELECT    cust_name_s,
          ordhdr_nbr_s,
          payterms_desc_s
FROM      customer
  JOIN
          order_header
  ON      cust_id_n = ordhdr_custid_fn
  JOIN
          payment_terms
  ON      payterms_id_n = ordhdr_payterms_fn
WHERE     cust_id_n = 152;

CUST_NAME_S         ORDHDR_NBR_S   PAYTERMS_DESC_S
------------------- ------------ ---------------
...                 ...          ...
WILE BESS COMPANY   523732       2% 15 NET 30
...                 ...          ...

30 rows selected.
```


www.TopSage.com

奇怪的是，现在查询返回了 30 行而不是 31 行。这是因为 WILE BESS COMPANY 的 523731 订单在 ORDHDR_PAYTERMS_FN 列有 NULL 值。所以，如果使用内部联接，整行都将被排除在结果集之外。这种情况需要用外部联接：

```
SELECT    cust_name_s,
          ordhdr_nbr_s,
          payterms_desc_s
FROM      customer
  JOIN
          order_header
  ON      cust_id_n = ordhdr_custid_fn
  LEFT  OUTER JOIN
          payment_terms
  ON      payterms_id_n = ordhdr_payterms_fn
WHERE     cust_id_n = 152;

CUST_NAME_S             ORDHDR_NBR_S   PAYTERMS_DESC_S
----------------------- -------------- ---------------
...                     ...            ...
WILE BESS COMPANY       523731         NULL
WILE BESS COMPANY       523732         2% 15 NET 30
...                     ...            ...

31 rows selected.
```

2. 旧语法

供应商的外部联接的旧语法各不相同。

Oracle 11g

直到 Oracle 9i 版本，外部联接的语法才与 SQL:2003 兼容。如果使用 Oracle 的早期版本，外部联接将通过被括号括起来的加号（+）进行声明，放在没有匹配行的表名之后。与上例产生相同结果的查询如下：

```
SELECT    cust_name_s,
          ordhdr_nbr_s,
          payterms_desc_s
FROM      customer,
          order_header,
          payment_terms
WHERE     cust_id_n = ordhdr_custid_fn
AND       ordhdr_payterms_fn = payterms_id_n (+)
AND       cust_id_n = 152
```

在 Oracle 中，因为（+）号位于=号的右边，联接通常被称为“右外部联接”，这便导致了混淆。

DB2 9.5

DB2 9.5 只在左外部联接和右外部联接中使用标准 SQL:2003 语法。

Microsoft SQL Server 2008

如前边所述，Microsoft SQL Server 2008 不支持左外部或右外部联接的旧语法，但是，如果确实需要（例如，当重新编写陈旧代码时），可以将数据库兼容级别设为 80，相当于 Microsoft SQL Server 2000，然后再将其改回 100，相当于 Microsoft SQL Server 2008：

```
ALTER DATABASE ACME SET COMPATIBILITY LEVEL = 80

SELECT    cust_name_s,
          ordhdr_nbr_s,
          payterms_desc_s
FROM      customer,
          order_header,
          payment_terms
WHERE     cust_id_n = ordhdr_custid_fn
AND       ordhdr_payterms_fn *= payterms_id_n
AND       cust_id_n = 152
ALTER DATABASE ACME SET COMPATIBILITY_LEVEL = 100
```

从本例中可以看到，为了得到与 SQL:2003 语法相同的结果，必须在 WHERE 子句中的等号左边放一个星号。

9.2.3 右外部联接

前面提到过，左外部联接和右外部联接之间的唯一不同之处在于查询中表联接的次序不同。为了证明这一点，本节使用与前一节输出结果相同的查询。

1. SQL:2003 标准语法

可以看到，ORDER_HEADER 表和 CUSTOMER 表的内部联接结果集位于 PAYMENT_TERMS 表的右边：

```
SELECT    cust_name_s,
          ordhdr_nbr_s,
          payterms_desc_s
FROM      payment_terms
  RIGHT   OUTERJOIN
          order_header
  ON      payterms_id_n = ordhdr_payterms_fn
  JOIN
          customer
ON        cust_id_n = ordhdr_custid_fn
WHERE     cust_id_n = 152

UST_NAME_S          ORDHDR_NBR_S
------------------- ------------
...                 ...
WILE BESS COMPANY   523731
WILE BESS COMPANY   523732
...                 ...

31 rows selected.
```

2. 旧语法

左外部联接的旧语法已经在前一节解释过了，右外部联接的旧语法是相似的。

Oracle 11g

在讲述左外部联接时已经提到过，在 Oracle 中，左外部联接和右外部联接的旧定义是模糊的。与前一查询等价的旧语法为：

```
SELECT    cust_name_s,
          ordhdr_nbr_s,
          payterms_desc_s
FROM      customer,
          order_header,
          payment_terms
WHERE     cust_id_n = ordhdr_custid_fn
AND       payterms_id_n (+) = ordhdr_payterms_fn
AND       cust_id_n = 152
```

将（+）号与没有匹配行的 PAYMENT_TERMS 表的列名（PAYTERMS_ID_N）一起移到左边。输出结果等同于其他的“等同的”查询结果，这种类型的查询在 Oracle 中被称为“左外部联接”。

注意：很多 Oracle 用户对“新”SQL:2003 外部联接语法感到困惑，问题在于，他们过去调用一个外部联接“左”或“右”取决于外部联接符（+）位于等号（=）的哪一边。“新”语法采用一种不同的方法——“左”或“右”关键字标识表的关系位置，从该表中检索所有行，不管另外一个参与联接运算的表是否有匹配行。

9.2.4 完全外部联接

完全外部联接是左外部联接和右外部联接的组合，它从“左边”的表和“右边”的表返回所有行，不管另外一个表是否有匹配行。例如，在 ACME 数据库中，有一些客户还没有下任何订单，而且一些订单还没有客户预订。

注意：没有客户的订单可能不常见。但是，考虑这样一种情况，每周客户 A 和客户 B 下相似的订单，客服人员创建了一些“准系统”订单以节约时间，保留客户号码字段为空（其他一些字段也为空），所以当一个客户实际下订单时，只需用很少时间来处理请求。

检索所有没有订单的客户以及没有客户订购的订单的查询为：

```
SELECT    customer.cust_name_s,
          order_header.ordhdr_nbr_s
FROM      customer
  FULL    OUTER JOIN
          order_header
  ON      customer.cust_id_n = order_header.ordhdr_custid_fn

CUST_NAME_S                              ORDHDR_NBR_S
---------------------------------------- -------------
...
```


www.TopSage.com

```
WILE SEAL CORP.                          523774
WILE SEAL CORP.                          523775
WILE SEAL CORP.                          523776
WILE SEAL CORP.                          523777
WILE SEAL CORP.                          523778
...                                      ...
WILE BESS COMPANY                        523730
NULL                                     523727
NULL                                     523728
MAGNETICS USA INC.                       NULL
MAGNETOMETRIC DEVICES INC.               NULL
FAIR PARK GARDENS                        NULL
...

  83 record(s) selected.
```

该语法与 SQL:2003 兼容，而且是“三大”数据库中唯一可用的完全外部联接语法。逻辑上讲，建议在 Oracle 中使用 (+) = (+)，在 Microsoft SQL Server 中使用*=*，但是，这些却不是有效的。

9.2.5 合并联接

UNION 联接（不要与 UNION 运算符混淆）基于与内部联接相反的思想。它的结果集只包含那些两个联接表中没有匹配的行。那些没有匹配行的列用 NULL 填充。

“三大”RDBMS 供应商都不支持 UNION 联接，在现实世界中，唯一支持 UNION 联接的数据库是 PostgreSQL。

Note 注意：根据集合论的基本原理（在附录 K 中讨论），可以认为 A UNION B=(A FULL OUTER JOIN B) DIFFERENCE (A INNER JOIN B)。

9.3 包含内联视图的联接

在本章前面提到过，涉及表联接的查询是很复杂的。可以联接表与其他的表、视图或其他特定于 RDBMS 的对象，可以从这些对象中选取行，生成结果集。

可以参与联接的另一种对象类型是内联视图，它只是 FROM 子句中的嵌套查询。

Note 注意：在一个查询中，可以使用内联视图代替常规视图来生成结果集。与常规视图不同，内联视图定义在数据库信息架构中不存在，而且不需要维护。如果您能肯定其他地方没有人使用这个结果，可以考虑在查询中使用内联视图。

下面的查询在 SQL:2003 标准联接中使用内联查询，返回联系的客户订单数多于 5 个的销售人员的电话号码。内联视图计算每个客户的订单数量，结果只限于有 5 个以上订单的客户，返回这些客户对应的销售人员的 ID。然后，结果与 SALESMAN 表进行联接得到销售人员的姓名，接下来，结果集与 PHONE 表联接检索销售人员的电话号码：

```
SELECT    s.salesman_name_s,
          p.phone_phonenum_s
FROM      salesman_s
    JOIN
       (SELECT      cust_salesmanid_fn,
                    ordhdr_custid_fn,
                    COUNT(ordhdr_nbr_s) cnt
       FROM         customer
                 JOIN
                    order_header
                 ON cust_id_n = ordhdr_custid_fn
       GROUP BY     ordhdr_custid_fn,
                    cust_salesmanid_fn
       HAVING       COUNT(ordhdr_nbr_s) > 5) c
    ON    s.salesman_id_n = c.cust_salesmanid_fn
    JOIN  phone_p
    ON    p.phone_salesmanid_fn = s.salesman_id_n
WHERE     p.phone_type_s = 'PHONE'
```

注意，在本例中，主查询和嵌套查询（内联视图）都包含联接。内联视图中的联接首先执行，然后，结果集用于主查询中，下面列出此设计逻辑遵循的步骤。

首先要查找订单数超过 5 个的所有客户，一个相对简单的单表查询可返回期望结果：

大家网 TopSage.com

```
SELECT   ordhdr_custid_fn,
         COUNT(ordhdr_nbr_s) cnt
FROM     order_header
GROUP BY ordhdr_custid_fn
HAVING   COUNT(ordhdr_nbr_s) > 5

ORDHDR_CUSTID_FN        CNT
------------------- ----------
             152       31
               1       10
```

得到了客户编号，但实际上感兴趣的是与客户联系的销售人员的信息，但是，表 ORDER_HEADER 不包含这些信息，所以需要在查询中联接 CUSTOMER 表：

```
SELECT   cust_salesmanid_fn,
         ordhdr_custid fn,
         COUNT(ordhdr_nbr_s) cnt
FROM     customer
   JOIN
         order_header
   ON    cust_id_n = ordhdr_custid_fn
GROUP BY ordhdr_custid_fn,
         cust_salesmanid_fn
HAVING   COUNT(ordhdr_nbr_s) > 5;

CUST_SALESMANID_FN ORDHDR_CUSTID_FN        CNT
------------------ ------------------- -------
                25                   1      10
                28                 152      31
```

这看起来已经不错了，但是还是不够完美。销售人员编号是没有意义的整数，没有给请求报告的业务分析员提供足够多的信息，因为想得到的是销售人员的名字。所以，将前面的查询命名为 c，并作为内联视图，并与表 SALESMAN 联接：

```
SELECT   s.salesman_name_s,
         c.ordhdr_custid_fn,
         c.cnt
FROM     salesman_s
     JOIN
        (SELECT     cust_salesmanid_fn,
                    ordhdr_custid_fn,
                    COUNT(ordhdr_nbr_s) cnt
        FROM        customer
           JOIN
                    order_header
            ON      cust_id_n = ordhdr_custid_fn
        GROUP BY    ordhdr_custid_fn,
                    cust_salesmanid_fn
        HAVING      COUNT(ordhdr_nbr_s) > 5) c
    ON s.salesman_id_n = c.cust_salesmanid_fn

SALESMAN_NAME_S         ORDHDR_CUSTID_FN        CNT
----------------------- ------------------- ----------
HUNTER COMPONENTS                     1       10
HENERY INCORPORATED                 152       31
```

这仍然不是我们想要的，结果集中，并不想看到客户编号或订单数量，而是想要查询返回的销售人员的电话号码。该信息可以在表 PHONE 中找到，将表与查询进行联接，并限制结果仅为电话号码记录，以恰当的形式进行查询：

```
SELECT   s.salesman_name_s,
         p.phone_phonenum_s
FROM     salesman_s
     JOIN
         (SELECT   cust_salesmanid_fn,
                   ordhdr_custid_fn,
                   COUNT(ordhdr_nbr_s) cnt
          FROM     customer
                JOIN
                   order_header
                ON  cust_id_n = ordhdr_custid_fn
           GROUP BY ordhdr_custid_fn,
                    cust_salesmanid_fn
           HAVING   COUNT(ordhdr_nbr_s) > 5) c
     ON       s.salesman_id_n = c.cust_salesmanid_fn
     JOIN     phone p
     ON       p.phone_salesmanid_fn = s.salesman_id_n
WHERE         p.phone_type_s = 'PHONE'
```


www.TopSage.com

```
SALESMAN_NAME_S                            PHONE_PHONENUM_S
------------------------------------------ ----------------
HUNTER COMPONENTS                          (717) 555-5479
HENERY INCORPORATED                        (814) 555-0324
```

9.4 包含相关查询的多表联接

要记住的一个重要的规则是，在一个查询中不要将旧语法和新语法一起使用，那样的话，语法将不能正常工作，势必会引起混淆。

虽然如此，如果需要生成一个相关查询，仅用 SQL:2003 语法是不行的，所以一般建议使用旧语法或使用其他方法代替相关查询。

下面的语句使用相关查询来检索所有下订单的客户姓名和电话号码：

```
SELECT
DISTINCT c.cust_name_s,
         p.phone_phonenum_s
FROM     customer_c,
         phone_p
WHERE    c.cust_id_n = p.phone_custid_fn
AND      EXISTS (SELECT *
                 FROM order_header_oh
                 WHERE oh.ordhdr_custid_fn = c.cust_id_n)
AND      p.phone type s = 'PHONE'

CUST_NAME_S                PHONE_PHONENUM_S
-------------------------- ----------------
WILE BESS COMPANY          (541) 555-8979
WILE ELECTROMATIC INC.     (541) 555-3246
WILE ELECTROMUSICAL INC.   (503) 555-0502
WILE ELECTRONICS INC.      (609) 555-4091
WILE SEAL CORP.            (909) 555-9957

  5 record(s) selected.
```

不使用相关查询的等价查询为：

```
SELECT
DISTINCT c.cust_name_s,
         p.phone_phonenum_s
FROM     customer_c
   JOIN
         order_header_oh
   ON    c.cust_id_n = oh.ordhdr_custid_fn
   JOIN
         phone_p
   ON    c.cust_id_n = p.phone_custid_fn
WHERE    p.phone_type_s = 'PHONE'
```

可以看到，在 SELECT 语句中可以避免使用相关查询。UPDATE 和 DELETE 语句在 Oracle 和 DB2 中更棘手。在第 6 章中讨论过，Microsoft SQL Server 在 UPDATE 和 DELETE 语句中有一个特殊的 FROM 子句。

9.5 提高多表查询效率

每个 RDBMS 都有自己的算法来将 SQL 查询翻译为二进制指令，然后通过 RDBMS 引擎进行解释以生成查询的执行计划。这个计划指定表联接的顺序、WHERE 子句使用的条件，以及是否使用索引等。

联接顺序和 WHERE 子句条件可以极大影响查询性能。例如，假设想联接 CUSTOMER 表和 ORDER_HEADER 表，然后只获取客户 FAIR PARK GARDENS 的所有订单。如果首先在表 CUSTOMER 中限制行集（WHERE customer.cust_name_s='FAIR PARK GARDENS'），然后将结果集与 ORDER_HEADER 表进行联接，操作会节省时间，因为这样只对一行进行了联接。另外，为 CUST_NAME_S 和参与联接运算的列添加索引也是有用的。

交叉参考：参阅第 4 章获取索引创建的建议。

www.TopSage.com

所有现代RDBMS的实现都有称为优化器的特殊机制，它基于某些信息架构中积累的信息为查询生成执行计划。理论上，不再使用复杂的查询编写规则（供应商之间各不相同，并且超出了本书范围），SQL开发人员为达到可接受的查询性能，不必再指定SELECT语句的FROM和WHERE子句中表和列的顺序，优化器会完成这个任务。

“理论上”这个词是有道理的。有时（通常是联接大量表的复杂查询）优化器不能找到分析查询的“最佳”方法。在这种情况下，可以给查询一个提示。使用预定义的、特定于供应商的关键词集合覆盖优化器的算法，并按程序员指定的顺序执行命令。一般来说，提示可以告诉RDBMS需要先联接哪个表，是否用到索引，或执行全表扫描等。

注意：提示是SQL代码中的指令，用于指示RDBMS优化器使用特定方法生成执行计划。

从第4章可以了解，使用索引并不总是有利的。例如，STATUS表很小，全表扫描是比较好的。下面的语法覆盖STATUS_ID_N列的索引，强制优化器执行全表扫描（使用Oracle语法）：

```
SELECT /*+ FULL(status)*/status_id_n,
                         status_code_s,
                         status_desc_s
FROM                     status
```

9.6 小结

多表查询是每个RDBMS必不可少的运算，它通过多种方法将数据集中到一个结果集中，有时包括复杂的商业逻辑。

联接表是SQL世界的重要概念之一，不能忽视。

有很多不同类型的联接，可以分为两大类：内部联接和外部联接。表联接的条件是同等联接或非同等联接，具体取决于用于联接表的列的比较结果（相等、大于、小于等）。

在查询中，除了表之外，其他的实体，如视图，甚至内联视图也可用于联接。SQL语言的强大功能（运算符、函数和子查询）可用于生成联接查询结果集。

为提高数据库执行包含复杂JOIN语句的查询性能，可以使用提示系统。提示是特定于RDBMS的实现，并不是SQL标准的一部分。

www.TopSage.com

大家网
TopSage.com

CHAPTER 10

第 10 章
SQL 函数

SQL 函数使编程变得更容易，并且在某种程度上弥补了基于集合的 SQL 的过程化不足。当一个查询检索某些数据时，查询中的函数便进行验证、转换、计算和获取系统信息等工作。

可以认为 SQL 函数是一种工具，它用于完成定义明确的任务。例如，计算平方根，或者将小写字母转换为大写字母。在 SQL 查询中，通过函数名（通常是一个关键字）调用一个函数，它执行指令并为查询中**每行**返回一个值。有些函数接受参数而有些不接受参数。但是，函数与一个 RDBMS 中的其他可执行模块的区别是，函数总有返回值。

前面已经提到过，SQL 本身不是一种过程语言（至少没有 SQL/PSM），它缺少一些过程特征，比如流程控制和循环，函数的使用弥补了这种不足。

如果对大量的 SQL 函数分类，所有的函数可以分为两大类：**确定性**函数和**非确定性**函数。确定性函数只要输入相同的参数，总是返回相同的结果；非确定性函数即使调用相同的参数，也可能会返回不同的结果。例如，ABS 函数是一个确定性函数，它以传递的数字作为参数，返回数字的绝对值。比如说对于参数–5，不管调用这个函数多少次，它的返回结果总是 5。对于 Microsoft SQL Server 的函数 GETDATE（），它不接受任何参数，只返回 RDBMS 服务器上的当前日期和时间，所以它是一个非确定性函数。每次调用，都返回一个新的日期和时间，即使只差不到一秒的时间。

Note 注意：所有的配置文件、游标、元数据、安全和系统统计函数都是非确定性函数。

有些 RDBMS 实现限制了非确定性函数的使用。例如，Microsoft SQL Server 不允许在创建索引的计算列和创建索引的视图中使用非确定性函数；IBM DB2 9.5 不允许在连接条件表达式中使用非确定性函数，在 Oracle 基于函数的索引中也不能使这类函数。

Note 注意：实际上，每个数据库供应商都为他们的产品提供了过程扩展插件。Oracle 有内置的 PL/SQL，Microsoft 使用专用的 Transact-SQL（与 Sybase——另外一个 RDBMS 供应商共享根）。DB2 9.5 使用 SQL PL。和基于集合的 SQL 不同，这些程序扩展插件允许在自己的宿主环境中创建成熟的程序，用户定义的自定义函数通常使用它们过程语言中的一种来创建。

本章提要：

- SQL 函数及其分类
- 不同场合的函数应用
- SQL 标准和特定于供应商的实现

www.TopSage.com

随着新版本的发行，在一个特定RDBMS实现过程中可用的SQL函数不断增多，而且一些供应商（本书中讨论的三大供应商属于这一类）允许用户定义自定义函数来执行非标准的任务。本章中，我们只给出最常用的函数列表和使用方法，重点阐述各供应商特定实现的不同之处。供应商特定函数的完整列表，请参考附录G。

Note 注意：在查询中使用函数的问题之一是查询在竞争对手的产品中不能正确运行。有些函数在名称和用法上相同，有些函数只有名称不同，而有些函数只能在某种特定的RDBMS实现过程中存在。

并不是所有这些函数（或者说大多数）都是SQL标准——SQL-89、SQL-92、SQL:1999，甚至SQL:2003（或即将发行的SQL:2008）的一部分。实际上，所有这些标准只确定了少数的函数符合特定水平（入门级、中间级或完全级）。古语说，一个标准不能始终为真。SQL2标准指定的函数列表如表10-1中所示。

表10-1 标准SQL标量函数

SQL函数	说明
CAST(值作为数据类型)	将提供的值从一种数据类型转换为另外一种数据类型，完成有意义的转换
CHAR_LENGTH(表达式)	返回表达式的长度。通常是字符组成的字符串
CONVERT(表达式 USING 约定)	返回转换后的字符串，根据转换参数中指定的规则进行转换
CURRENT_DATE	返回系统的当前日期
CURRENT_TIME(精度)	根据指定的精度返回当前系统时间
eCURRENT_TIMESTAMP(精度)	根据指定的精度返回系统当前的日期和时间
EXTRACT(部分 FROM 表达式)	提取日期时间表达式的指定部分
LOWER(表达式)	将大写字符串（混合大小写）转换为小写
OCTET_LENGTH(表达式)	返回表达式的字节数（每个字节含8位）
POSITION(char 表达式 IN 源)	返回char表达式在源中的位置
SUBSTRING(字符串表达式开始位置长度)	返回字符串表达式中的字符串部分，从开始位置开始，长度为指定长度
TRANSLATE(字符串表达式 USING 转换规则)	根据指定的规则，将一个字符串转换为另外一个字符串
TRIM(LEADING | TRAILING | BOTH char 表达式 FROM 字符串表达式)	返回将字符串表达式中的去除了开头、尾部char表达式字符或同时去除了开头和尾部char表达式字符的字符串
UPPER(表达式)	将小写字符串（或混合大小写）转换为大写字母

Cross-Reference 交叉参考：很明显，明确描述最流行的RDBMS实现中可用的函数并在不同实现过程中映射这些函数是有好处的。在附录G中，我们尝试给出所有函数的列表、它们各自的映射以及“三大”数据库——Oracle 11g、IBM DB2 9.5和Microsoft SQL Server 2008的简要说明。

SQL标准为SQL函数（或标准文档中所称的“SQL调用例程”，与“外部调用”相反）建立行为规则。例如，一个SQL调用例程可以拥有一个等价的例程名（也就是同义词）。同一架构中的两个SQL调用函数不允许有相同的标记（也就是说，在同一架构中，相同名称的函数必须有不同的参数个数和类型）等。这些规则对于基于SQL引擎实现而工作的公司（也就是Oracle、Microsoft和IBM）很重要；对于程序员来讲，只有进入UDF（User-Defined Function，用户定义函数）世界的时候，这些规则才变得重要起来。

每个供应商在自己的数据库产品中都有自身支持的函数分类。IBM将它的DB2 9.5函数分为列函数、标量函数、行函数和表函数；Oracle使用单行函数、聚合函数、分析函数和对象引用函数等；Microsoft有着行集函数最详细的分类，包括聚合函数、排名函数和标量函数等。

Note 注意：在SQL:2007标准中采用的一个分类包括这些术语，如niladic、monadic和dyadic分别指那些标记中无参数、有一个参数和有两个参数的函数。这种分类从数学上来讲是正确的，对于标准化也很有用，但却很难满足开发人员的需要，因为开发人员关心的是函数接受和返回哪种类型的参数。

本章主要介绍可用于任何实现背景的函数，不介绍很多专用扩展插件。第15章讨论XML相关函数，第12章讨论安全函数，第13章会介绍元数据函数。

www.TopSage.com

Note 注意：SYSIBM和SYSFUN结构函数在IBM中是有区别的，它们处理数据的方式不同。例如，有时一个人在接收数字输入，而同时另外一个人在接收字符输入。考虑一个IBM实现函数重载的情况，根据传递给函数的参数的数据类型的不同，函数要完成不同的任务。另外，一些不同版本DB2中的函数在IBM DB2 9.5中没有相应的等价函数。

除非另作说明，所有例子都使用ACME数据库。可以直接在选择的RDBMS中根据该数据库运行这些例子。

Cross-Reference 交叉参考：参考附录F了解安装ACME数据库的介绍。参考附录E了解如何为选择的RDBMS启动命令行和/或图形化界面。

10.1 数字函数

这里将某些函数归类到数字函数，因为它们仅对数字进行操作，也就是说，这些函数的输入和输出参数通常为INTEGER、DECIMAL或其他任何数字数据类型。实际的供应商分类可能不同。表10-2中列出了最常见的数字函数。

表10-2 数字函数

Oracle 11g	IBM DB2 9.5	Microsoft SQL Server 2008	描述
ABS(n)	ABS(n)	ABS(n)	返回数字*n*的绝对值
ACOS(n)	ACOS(n)	ACOS(n)	返回*n*的反余弦值，用弧度表示
ASIN(n)	ASIN(n)	ASIN(n)	返回*n*的反正弦值，用弧度表示
ATAN(n)	ATAN(n)	ATAN(n)	返回*n*的反正切值，用弧度表示
ATAN2(n,m)	ATAN2(n,m)	ATN2(n,m)	返回*n*和*m*的反正切值，用弧度表示
CEIL(n)	CEIL[ING](n)	CEILING(n)	返回大于等于*n*的最小整数
COS(n)	COS(n)	COS(n)	返回弧度*n*的余弦值
COS(n)/SIN(n)	COT(n)	COT(n)	返回角*n*（用弧度表示）的余切
N/A	DEGREES(n)	DEGREES(n)	返回用弧度表示的角*n*的度数
EXP(n)	EXP(n)	EXP(n)	返回*n*的指数值
FLOOR(n)	FLOOR(n)	FLOOR(n)	返回小于等于*n*的最大整数
LN(n)	LN(n)	LOG(n)	返回数字*n*的自然对数
LOG(n,m)	LOG(n,m)	N/A	返回底数为*n*的*m*的对数
LOG(10,n)	LOG10(n)	LOG10(n)	返回底数为10的*n*的对数
N/A	PI()	PI()	返回pi（近似为3.1415926...）
MOD(n,m) REMAINDER(n,m)	MOD(n,m)	Operator %	返回*n*除以*m*的余数
POWER(m,n)	POWER(m,n)	POWER(m,n)	返回*m*的*n*次幂
N/A	RADIANS(n)	RADIANS(n)	返回用度数表示的角*n*的弧度
N/A	RAND(n)	RAND(n)	返回0和1之间的随机数。Oracle使用DBMS_RANDOM包实现该功能
ROUND(n,[m])	ROUND(n,[m])	ROUND(n,m,[0])	将数字*n*四舍五入为*m*位小数。Microsoft SQL Server中，最后一个参数设置规定功能：如果是除0外的整数，结果是截断；如果为0，结果则四舍五入
SIN(nj)	SIN	SIN	返回*n*的正弦值（*n*用弧度表示）
SQRT(n)	SQRT(n)	SQRT(n)	返回指定值的平方根
N/A	N/A	SQUARE(n)	返回数值表达式的平方
SIGN(n)	SIGN(n)	SIGN(n)	*n*为负数则返回-1，为正数则返回1，为0则返回0
TAN(n)	TAN(n)	TAN(n)	返回*n*的余切值（*n*用弧度表示）
TRUNC(n,[m])	TRUNC[ATE](n,[m])	ROUND(n,m,<>0)	返回*n*截断到*m*位小数的值。 在Microsoft SQL Server中，当ROUND函数的最后一个参数不为0时，函数的结果为截断

www.TopSage.com

大家网 TopSage.com

Cross-Reference 交叉参考：请参阅第 3 章了解有关 SQL 数据类型更多内容。

10.1.1 CEIL

根据定义，函数 CEIL 返回传递参数的最小整数，这意味着该函数是向上舍入的，其语法和用法在三大数据库中相同。

参数可以是一个常量、一个列值或任意其他数值表达式，下面是查询及输出结果：

```
SELECT
    CEILING (prod_price_n) ceil_price,
    prod_price_n
FROM product;

ceil_price    prod_price_n
------------ ------------
19            18.24
34            33.28
27            26.82
```

函数 FLOOR 实现方式与其相似，正数返回对其截断的最小整数值，负数返回接近于负无穷的值。该函数的语法在三大数据库中相同：

```
SELECT
    FLOOR (prod_price_n) floor_price,
    prod_price_n
FROM product;

floor_price   prod_price_n
------------ ------------
18            18.24
33            33.28
26            26.82
```

Tip 提示：三大 RDBMS 实现中的命令行界面是不同的。Oracle 使用 SQL*Plus，Microsoft 使用 SQLCMD 实用工具，IBM 使用命令行处理器（db2cmd）。要执行 SQL 命令（连接完成之后），在实用工具窗口（Windows 平台中是 DOS 窗口，UNIX/LINUX 平台中是命令提示符）中键入命令行。然后，DB2 9.5 中是按 Enter 键，Oracle 中是输入分号（;）和按 Enter 键。Microsoft SQL Server 则键入 GO 并按 Enter 键。这些接口每个都是可自定义的，可以“驯化”它们，例如，Microsoft SQL Server 可用一个分号来代替 GO 语句。

10.1.2 ROUND

有人可能会问 ROUND 函数与 TRUNC 函数有什么不同？对一个数取整就是确定特定的长度和精度。该函数在三大 RDBMS 实现中的运行机制几乎相同。

下面的例子中，对 PRODUCT 表的 PROD_PRICE_N 列中所有值取整，得到一位小数的十进制数。

```
SELECT
   ROUND(prod_price_n,1) round_price,
   prod_price_n
FROM product;

round_price   prod_price_n
------------ ------------
18.20         18.24
33.30         33.28
26.80         26.82
```

注意，取整时，数字可以进行**上**舍入和**下**舍入，具体取决于数字本身与哪个数更接近。33.28 取整得到 33.30，18.24 取整得 18.20，就是说，分数部分大于 0.5 则向上取整，小于 0.5 则向下取整。

Note 注意：ROUND 函数在 Microsoft SQL Server 的版本中的运行机制与在 Oracle 和 DB2 9.5 中有些不同：它有第三个可选参数（函数），在默认情况下为 0。当省略该参数或将其显式设置为 0 时，结果舍入，和上例的情况相同。当参数值不为 0 时，结果则截断。例如，ROUND(33.28.1)将得到 33.30，ROUND(33.28. 1.1)的结果为 33.20。

第二个参数可以为负值，表示对小数点左边的数，即数字的整数部分取整：

```
SELECT
```


www.TopSage.com

```
    ROUND (prod_price_n,1) round_price_right,
    ROUND (prod_price_n,-1) round_price_left,
    prod_price_n
FROM product;

round_price_right  round_price_left   prod_price_n
-----------------  -----------------  ------------
18.30              20.00               18.32
34.10              30.00               34.09
26.90              30.00               26.92
16.00              20.00               15.98
```

这里，指定-1 作为函数的第二个参数，返回的结果是小数点前数字的舍入值。

10.1.3 TRUNC

函数 TRUNC 返回第一个参数被第二个参数指定的小数位数截取后的值。不管这个数字本身是正数还是负数，截断后返回的结果更接近 0。下面的例子适用于 Oracl 和 IBM DB2 9.5，Microsoft SQL Server 2008 使用 ROUND 函数来截断数字：

```
SELECT
    TRUNC(prod_price_n, 1) trunc_price,
    prod_price_n,
FROM product;

trunc_price   product_price_n
------------ ------------
18.20         18.24
33.20         33.28
26.80         26.82
```

函数 TRUNC 在 Oracle 中被重载，也就是说，随着传递给它的参数类型的不同（在 Microsoft SQL Server 2008 和 IBM DB2 9.5 中没有这种情况），执行过程也有所不同。例如，当对 DATE 数据类型应用 TRUNC 函数时，它产生日期参数的午夜值，也就是说，截断日期中的所有小时、分和秒：

```
SELECT
    SYSDATE,
    TRUNC(SYSDATE) truncated
FROM dual;

SYSDATE                  TRUNCATED
---------------------    ----------------------
9/22/2003 10:53:36 AM    9/22/2003 00:00:00 AM
```

10.1.4 RAND

函数 RAND 用于在运行时生成一些伪随机数字。它的语法和用法在 DB2 9.5 和 Microsoft SQL Server 2008 中几乎相同（尽管在 Oracle 中可以通过使用 PL/SQL 包得到同样结果，但它没有类似的函数）。函数接收一个可选的种子参数（整数），产生一个介于 0 和 1 之间（包括 0 和 1）的伪随机浮点数。

Microsoft SQL Server 2008 的语法如下：

```
SELECT RAND(1) random_number

random_number
---------------------
0.71359199321292355
```

The DB2 9.5 syntax produces analogous results in a somewhat different format:

```
SELECT
    RAND(5)
FROM sysibm.sysdummy1
random_number
---------------------
+1.64799951170385E-03
```

RAND 函数的使用有一些微妙之处：在一个会话中使用同一种子值多次调用该函数会产生相同的输出结果。要得到不同的伪随机数，需要指定不同的种子值或使用不同的会话。

获取不同范围的随机数

当需要的随机数比 1 大时怎么办？在这种情况下，要先用范围因子乘以函数的输出，再用 TRUNCATE 或 ROUND 函数计算结果。下面的例子采用 Microsoft SQL Server 2008 语法生成产生一组 0～10 000 的伪随机数：

www.TopSage.com

```
SELECT ROUND((RAND(15)* 10000),0) from_zero_to_10000

from_zero_to_10000
--------------------------
7139.0
```

在自定义函数中封装这项功能是一个明智的解决办法。三大供应商都提供了在他们的 RDBMS 软件中生成用户定义函数的功能。请参阅第 14 章获得有关用户定义函数（UDF）的更多信息。

10.1.5 SIGN

函数 SIGN 在三大数据库实现中的工作方式完全相同，它用于确定数值表达式参数的符号：如果是正数，函数返回 1；如果是负数，结果为-1；如果参数为 0，结果返回 0。在我们的例子中，因为价格为正数，所以全部返回 1：

```
SELECT
   SIGN (prod_price_n) sign_price,
   prod_price_n
FROM product;

sign_price    prod_price_n
------------ ------------
1             18.24
1             33.28
1             26.82
```

可以用任何数值表达式代替表中的列值，因为所有实现都需要一个 SELECT 语句来执行函数，所以需要从一些地方选取。在 Oracle 中，可以从一个特殊表 DUAL 中选取，IBM DB2 中有一个叫做 sysibm.sysdummy1 的表，在 Microsoft SQL Server 2008 中，查询选择的表是隐含的（Sybase RDBMS 中也是，它与 SQL Server 共享根）。

注意：DUAL 表作为 Oracle 数据字典的基础对象是 Oracle 的 Chuck Weiss 提出的。它并不单独存在，而是用在某些 JOIN 运算（在第 9 章已经讨论）中。在逻辑上，它符合 SELECT 标记，暗指从某些数据源中选取数据（表、视图、子查询、表值函数等）。

10.2 字符串函数

我们将字符串函数归为一类，是因为它们专门针对字符串进行计算，即对字符串进行操作：转换字母的大小写、改变对齐方式、查找 ASCII 码，以及提取子字符串等。通常情况下（也有例外），在 RDBMS 实现中，这些函数的输出结果是字符串（即使 SQL 标准规定结果总是字符串）。

常用的字符串函数如表 10-3 所示。

交叉参考：参考附录 G 查看供应商特定函数的完整列表。

表 10-3 字符串函数

Oracle 11g	IBM DB2 9.5	Microsoft SQL Server 2008	描述
ASCII(字符串)	ASCII(字符串)	ASCII(字符串)	返回字符串第一个字符的 ASCII 码
CHR(数字)	CHR(数字)	CHAR(数字) NCHAR(数字)	返回 ASCII 码对应的字符
CONCAT (字符串 1, 字符串 2)	CONCAT(字符串 1, 字符串 2)	运算符'+'	返回两个字符串串联的结果。 Microsoft SQL Server 重载此函数，并将数值相加
N/A	DIFFERENCE(exp1,exp2)	DIFFERENCE(exp1,exp2)	计算两个 SOUNDEX 值之差；整数 0~4
INITCAP	N/A	N/A	返回首字母大写、其他字母小写的字符串
INSTR(字符串,子字符串,开始位置,出现次数)	LOCATE(字符串 1, 字符串 2,n) POSSTR(字符串 1, 字符串 2,n)	CHARINDEX (字符串 1, 字符串 2,n) PATINDEX (<pattern>, <string>)	返回字符串中子串出现的位置。 IBM DB2 测试中的 POSSTR 函数是区分大小写的

www.TopSage.com

TopSage.com

续表

Oracle 11g	IBM DB2 9.5	Microsoft SQL Server 2008	描述
SUBSTR(字符串,1,n)	LEFT(字符串,n)	LEFT(字符串,n)	返回从左边开始的 *n* 个字符
LENGTH(字符串)	LENGTH(字符串)	LEN (字符串)	返回字符串中字符的个数
LENGTHB(表达式)	LENGTH (表达式)	DATALENGTH (表达式)	返回表达式（可以是任意数据类型）的字节数
LOWER(字符串)	LOWER(字符串) LCASE(字符串)	LOWER(字符串)	将字符串中的所有字符转换为小写
LPAD(字符串 1,n, 字符串 2)	REPEAT(char 表达式,n) SPACE(n)	REPLICATE(char 表达式,n) SPACE(n)	对于 REPEAT 和 REPLICATE 函数，返回第一个参数重复 *n* 次的字符串。对于 Oracle 的 LPAD 函数，返回第一个参数从左侧填充第三个参数 *n* 次。SPACE 函数用于重复空格 *n* 次
LTRIM(字符串,集合)	LTRIM(字符串)	LTRIM(字符串)	返回前导空格被去除的字符串
REPLACE(字符串 1, 字符串 2,字符串 3)	REPLACE(字符串 1, 字符串 2, 字符串 3)	REPLACE(字符串 1, 字符串 2, 字符串 3)	在字符串 2 中，用字符串 3 替换字符串 1
RPAD(字符串 1,n, 字符串 2)	SPACE(n)	SPACE(n)	Oracle 的 RPAD 返回在字符串 1 左侧填充字符串 2 *n* 次。SPACE 函数用于重复空格 *n* 次
LPAD/RPAD 结合使用	REPEAT(字符串,n)	REPLICATE(字符串,n)	返回重复 *n* 次的参数字符串
RTRIM(字符串,集合)	RTRIM(字符串)	RTRIM(字符串)	返回去除末尾空格的字符串。另外，在 Oracle 的实现中，第二个可选参数指定哪个字符被去除
N/A	N/A	REVERSE	返回倒序的字符表达式
TO_CHAR(表达式)	CHAR(表达式)	STR(表达式)	将参数表达式转换为字符串
SUBSTR(字符串,n,m)	SUBSTR(字符串,n,m)	SUBSTR(字符串,n,m)	返回字符串中从 *n* 个字符开始 *m* 长度的子串
TRANSLATE(字符串 1, 字符串 2, 字符串 3)	TRANSLATE(字符串 1, 字符串 2,字符串 3) INSERT(与函数 STUFF 工作原理相同)	STUFF (字符串 1,开始位置, 长度, 字符串 3)	用字符串 3 替换在字符串 2 中字符串 1 出现的位置。函数 STUFF（Microsoft）和 INSERT（IBM）是从起始位置删除指定长度的字符，并在该位置插入字符串 2
TRIM([OTH | LEADING | TRAILING,[trim_cha racter]],string)	STRIP(BOTH | LEADING | TR A I LING,string) TRIM(BOTH | LEADING | TR AILING,string) LTRIM(RTRIM(字符串))	LTRIM(RTRIM(字符串))	从字符串中去除开头或末尾空格，或全部去除。Oracle 版本也允许从字符串中去除任意字符
N/A	N/A	UNICODE(字符串)	返回输入表达式第一个字符的整数，由 Unicode 标准定义
UPPER(字符串)	UPPER(字符串) UCASE(字符串)	UPPER(字符串)	将字符串中的所有字符转换为大写

事实证明，字符串函数是应用最广泛，也是最容易混淆的函数。最常用用法的例子将在本章后面给出。

Note 注意：由于数据库处理不同语言集合的固有能力，几乎每个 RDBMS 都有一个识别非英文字符集的重载函数，或者有处理非英文字符集的独立函数。因此，Oracle 有它最常用字符串函数的 NLS_版本，它有一个附加的输入参数来指定区域。

10.2.1 CONCAT

顾名思义，CONCAT 函数只连接两个字符串。在 SQL Server 中，该函数可以用+运算符代替，在 Oracle 和 DB2 9.5 中用 ‖ 代替。

Note 注意：可以将一个函数的输出作为另外一个函数的输入，这并不是 RDBMS 和 SQL 特有的，这是一般的程序设计概念。

下面是使用 Oracle 语法进行连接的例子：

www.TopSage.com

```
SELECT
    CONCAT( '$', TO_CHAR(prod_price_n)) display_price
FROM product;

display_price
------------------------------
$18.24
$33.28
$26.82
```

下面是DB2 9.5中使用的语法：

```
SELECT
    '$' || CHAR(prod_price_n) display_price
FROM product;

display_price
-------------
$00000018.24
$00000033.28
$00000026.82
```

Note 注意：IBM DB2中的函数CHAR将数值数据转换为固定长度的字符串，并从结果集中的左边起用0来填充。0的个数取决于转换参数的数值数据类型。可以使用附加的格式和/或IBM DB2中的转换函数产生和Oracle或Microsoft SQL Server中相同的结果。

下面是等价的Microsoft SQL Server 2008语法：

```
SELECT
    '$' + CONVERT(varchar, prod_price_n) display_price
FROM product;
```

Note 注意：为了连接两种不同的类型，要使用转换函数CHAR、TO_CHAR和CONVERT将数值表达式转换为字符串数据类型。一些实现会自动转换兼容的数据类型，一些需要显式转换。不依赖隐式转换，而进行显式转换总是好的。本章稍后将介绍更多有关数据类型转换的内容。

10.2.2 CHARINDEX、INSTR、LOCATE和POSSTR

SQL是一种专门用于处理信息的语言。因此，它有处理字符串和字符的丰富工具集。函数INSTR、LOCATE和CHARINDEX用于确定一个字符串中某特定字符（或字符组合）的位置；数据获得的信息，可以用多种方法将文本分离成一些小片段。

例如，要找出第一个单词结束的位置，可以使用空格来分隔描述文字中的单词（假设列PROD_DESCRIPTION_S中的每个值都至少有一个空格）。

在Microsoft SQL Server 2008语法中，空格可以作为常量（两个单引号''）来表示。然而，最可靠的方法是使用函数CHAR(32)，用ASCII码32来显式地指定空格。这样的语法只需经过小的修改，就能在所有RDBMS实现中移植。下面的两个语句是等价的，生成相同的执行结果：

```
SELECT
    CHARINDEX(' ', prod_description_s, 1) char_position
FROM product;

SELECT
    CHARINDEX(CHAR(32),
    prod_description_s, 1) char_position
FROM product;

char_position
-------------
7
6
6
6
5
8
4
4
6
8

(10 row(s) affected)
```


www.TopSage.com

Oracle 的 INSTR 函数语法稍有不同，Oracle 允许一个字符串内多次出现另外一个字符串，第一次、第二次等（作为一个输入参数）。该函数的大多数参数是可选的。

在 Oracle 11g SQL*Plus 中执行下面的查询，查找字符串中从第一个字符开始，**第一次**出现空格的位置：

```
SELECT
    INSTR(PROD_DESCRIPTION_S, CHAR(32),1,1) char_position
FROM product;
```

下面的查询在 IBM DB2 9.5 中执行，生成和前面的 Microsoft SQL Server 相同的结果：查找空格在字符串中第一次出现的位置（开始字符和出现次数都是可选参数，如果省略的话，则默认为 1，即从第一个字符开始，第一次出现的位置）：

```
SELECT
    LOCATE(' ',PROD_DESCRIPTION_S) char_position
FROM product;
```

如果使用 IBM DB2 9.5 的函数 POSSTR，需要改变参数的顺序：

```
SELECT
    POSSTR(PROD_DESCRIPTION_S,' ') char_position
FROM product;
```

这些查询的结果将是相同的，并且与 Microsoft SQL Server 中生成的结果匹配。

注意：可选参数是那些如果没有指定参数也有默认值的参数。因为参数的顺序是固定的，所以指定一个参数之前，必须输入它前面的所有参数。例如，在前面的例子中，只要指定了位置值（第 4 个参数），就不能再省略开始位置（第 3 个参数）。

10.2.3 SUBSTR 和 SUBSTRING

SUBSTR（在 Microsoft SQL Server 中为 SUBSTRING）函数返回传入字符串参数的子串，由开始位置和需要的子字符串（由字符组成）长度决定。下面查询例子使用该函数返回 prod_description_s 列值的前三个字符：

```
SELECT
    SUBSTR(prod_description_s,1,3)
FROM product;
```

第三个参数指定需要的长度。在 Oracle 和 DB2 9.5 中，该参数是可选的，但是在 Microsoft SQL Server 的 SUBSTRING 函数中，该参数是必需的。如果第 3 个参数被省略，那么在 Oracle 和 DB2 9.5 中，函数将返回开始位置之后的所有字符。要在 Microsoft SQL Server 中得到相同结果，要将这个函数与 LEN/LENGTH 函数（请参阅本章后面）结合使用来确定字符串的总长度，或指定一个足够大的数以超过可能的字符串长度(不大于 8 000)。例如，在 SQL Server 中，下面的查询将返回 prod_description_s 列中从第 2 个字符开始的所有字符：

```
SELECT
    SUBSTRING(prod_description_s, 2, LEN(prod_description_s))
FROM product;
```

下面使输出结果稍微复杂一点，以更好地展示函数的用法。假如一个用户用一种特殊的方式处理结果，使输出的结果结合产品号、产品价格和产品描述，并且使一些公司应用程序的产品描述格式为 <first word of product description><hash mark><product_number><pound sign><dollar sign><product_price>。有很多方法可以实现该功能，下面的查询列出其中的一种（采用 Microsoft SQL Server 语法）：

```
SELECT
    LEFT(prod_description_s,
    CHARINDEX(CHAR(32),
    prod_description_s, 1)-1) + '#'+ prod_num_s + '#' + '$' +
   CONVERT(VARCHAR,prod_price_n) display
FROM product;

display
-------------------------
SPRUCE#990#$18.24
STEEL#1880#$33.28
STOOL#2871#$26.82
STOOL#3045#$15.92
HAND#4000#$11.80
```


www.TopSage.com

大家网 TopSage.com

Oracle 使用下面的查询生成结果：

```
SELECT
    SUBSTR(prod_description_s,1
      INSTR(CHAR(32),
      prod_description_s, 1,1)-1) || '#' ||
      prod_num_s || '#' || '$' ||
      TO_CHAR(prod_price_n)display
FROM product;

display
-------------------------
SPRUCE#990#$18.24
STEEL#1880#$33.28
STOOL#2871#$26.82
STOOL#3045#$15.92
HAND#4000#$11.80
```

IBM DB2 9.5 使用 POSSTR（或 LOCATE）函数代替 Oracle 的 INSTR 函数，函数 CHR 产生一个 ASCII 码为 32 的空格，并用 CHAR 函数（代替 Oracle 的 TO_CHAR 函数）将数字转换为字符串：

```
SELECT
    SUBSTR(prod_description_s,1
    POSSTR(prod_description_s, CHR(32))-1) || '#' ||
   prod_num_s || '#' || '$' ||
    CHAR(prod_price_n) display
FROM product;

display
-----------------------------------
SPRUCE#990#$18.24
STEEL#1880#$33.28
STOOL#2871#$26.82
STOOL#3045#$15.92
HAND#4000#$11.80
```

这个查询可能看起来比较复杂，但是却容易理解。CHARINDEX/INSTR/POSSTR 函数找出空格（CHAR(32)）的位置，并从该数中减去 1，因此最终结果集中不包含空格。将位置值作为函数 LEFT 的输入，指定从左边起的哪个位置开始返回字符（Oracle 和 DB2 9.5 使用 SUBSTR 函数；Microsoft SQL Server 使用 SUBSTRING 函数，从某种意义上说，SUBSTRING 函数是它的 LEFT/RIGHT 函数的更通用版本）。其他的是简单的字符串联，前面讨论过。

注意：您可能注意到了，在 DB2 9.5 中没有使用可用函数 LEFT，这是因为该函数与 Microsoft SQL Server 中的同名函数运行机制不同。在 IBM DB2 9.5 数据库中，函数 LEFT 返回**表达式 1** 中最左边的**表达式 2** 字节组成的字符串。

10.2.4 LENGTH

函数 LENGTH 返回参数中的字符数（不是字节的个数，请参阅第 3 章了解详细信息）。如果参数不是字符类型的，在三大 RDBMS 数据库中将隐式转化为字符串，并返回长度。

注意：一些 RDBMS 供应商并不如此通融，不兼容的数据类型（Sybase 就是其中一个）将返回一条错误信息。所以，为保证可移植性，建议在将参数传递给函数之前先验证数据类型。

Oracle 也提供了函数的多种形式。例如，用函数 LENGTH 返回字符数：

```
SELECT
    LENGTH(prod_description_s) length_of_string
FROM product;
```

Oracle 和 SQL Server 分别使用 LENGTHB 和 DATALENGTH 函数来返回表达式中的**字节**数。

注意：IBM 重载 LENGTH 函数，可以返回字符或字节长度，具体长度取决于参数的数据类型：对于参数'5'（一个字符参数）返回 1，参数 5（一个数字参数）则返回 4。

返回字节个数的函数不是将参数转换为字符串，而是给出数据类型在 RDBMS 中定义的内存大小。例如，下面的查询

```
SELECT
    LENGTHB(SYSDATE)
FROM dual;
```

在 Oracle（当前系统日期所占的内存大小）中返回 9，Microsoft SQL Server 中的等价表达式如下：

www.TopSage.com

```
SELECT DATALENGTH(GETDATE())
```

结果显示，在 Microsoft SQL Server 中，用 8 个字节来存储系统日期。在 IBM UDB 中，使用 4 个字节存储日期（和整数相同），使用 10 个字节存储时间戳：

```
SELECT
    LENGTH(CURRENT DATE) date_length,
    LENGTH(CURRENT TIMESTAMP) timestamp_length
FROM sysibm.sysdummy1
date_length        timestamp_length
------------       ------------------
4                  10
```

交叉参考：请参阅第 3 章，了解有关数据类型和内存的详细信息。

10.2.5 LOWER 和 UPPER

函数 LOWER 和 UPPER 符合 SQL:2007 标准，在三大 RDBMS 数据库不修改即可通用，这并不多见，这些函数简单直观，分别将字符串表达式转换为小写字母和大写字母：

```
SELECT
    UPPER(prod_description_s) upper_case,
    LOWER(prod_description_s) lower_case
FROM product;

Uppe_case                    lower_case
-------------------------  -------------------------
SPRUCE LUMBER 30X40X50       spruce lumber 30x40x50
STEEL NAILS 6''              steel nails 6''
STOOL CAPS 5''               stool caps 5''
```

IBM DB2 9.5 还包含函数 LCASE 和 UCASE 的附加版本。很可能它们是长期商业经营的产物——在市场上活动 30 年以上，确实会积累很多东西。

注意：区域字符集的行为会有所不同。一些供应商提供了独立的函数集来处理非英文字符集，尤其是关于分类和排序的函数。请参阅供应商文档获得更多信息。

10.2.6 TO_CHAR、CHAR 和 STR

这些函数同样属于本章后面介绍的广义转换函数，它们用于将非字符数据类型转换为字符数据类型。例如，将数字转换为字符串，或将日期/时间转换为字符串（当生成一个只接受字符数据的报表或用于内部处理时，需要用到这些函数）。除了能够在数据类型之间转换之外，还可以在转换时格式化输出结果，比如日期和数据类型。

这些函数的用法已经在前面 CONCAT 函数的例子中进行了说明。在 IBM DB2 9.5 中，因为 CONCAT 函数只接收字符串，所以需要将所有数字转换为字符串以避免错误。Oracle 11g 会隐式地将所有串联值转换为字符串。

Microsoft 的 SQL 函数 STR 与 Oracle 的 TO_CHAR 和 IBM 的 CHAR 不同，它只接收数字输入，不接收日期/时间或字符串。它用可选参数指定结果（包括小数点）的总长度和小数位数。例如，下面的查询将一个浮点数 123.35（两位小数）转换为字符串（采用 Microsoft SQL Server 语法）：

```
SELECT STR(123.35) result
result
----------
       123
```

由于省略了两个可选参数——总长度（默认为 10）和精度（默认为 0），因此结果被截断。下面的查询期望结果长度为 7 个字符，包含 3 位小数（如果指定 2 位小数，小于现有长度，那么将对最终输出结果进行四舍五入）：

```
SELECT
 STR(123.235,7,3)  result1
,STR(123.23,7,3) result2
,STR(123.2318,7,3)  result3

result1 result2    result3
-------  --------- -------
123.235   123.230   123.232
```


www.TopSage.com

10.2.7 REPLACE

IBM DB2 9.5、Oracle和Microsoft SQL Server中的REPLACE函数返回每次表达式2被表达式3替换的字符串（CHAR数据类型）。例如：

```
SELECT
    REPLACE('aabbaabbaa','aa','bb12')
FROM dual;

result
----------------
bb12bbbb12bbbb12
```

由于每个aa都被bb12替换，所以查询返回bb12bbbb12bbbb12字符串。要在Microsoft SQL Server 2008中运行该查询，只需删除FROM子句即可。在IBM DB2 9.5中运行时，用sysibm.sysdummy1代替dual。

10.2.8 REPLICATE和REPEAT

使用REPLICATE和REPEAT函数重复字符或字符序列。这些函数适用于IBM DB2 9.5和Microsoft SQL Server。在Oracle中，为完成相似的功能，需要结合使用函数LPAD/RPAD。使用这个函数相当直观。在下面的DB2 9.5的例子中，REPEAT产生一个字符串，在这个字符串中，单独的词repeat被重复了三次（注意单独的词之间没有空格分隔）：

```
SELECT
    REPEAT('repeat',3) example
FROM SYSIBM.SYSDUMMY1

example
------------------------------
repeatrepeatrepeat
```

根据参数的数据类型，对于结果字符串的总长度是有限制的。最大重复次数不能超过系统整数值的上限（对于32位机器来说，上限为2 147 483 647）。在Microsoft SQL Server中采用下面的语法得到相同的结果：

```
SELECT
    REPLICATE('repeat',3) example

example
------------------------------
repeatrepeatrepeat
```

下面给出使用Oracle的RPAD函数重复字符串的例子：

```
SELECT
    RPAD('repeat',LENGTH('repeat')*3,'repeat') example
FROM dual;

example
------------------------------
repeatpepeatrepeat
```

这里，函数的第二个参数决定结果字符串的总长度，函数LENGTH决定初始字符串的长度，用3乘以函数LENGTH是指定字符串要重复三次。

10.2.9 TRANSLATE

这是REPLACE函数的智能版。TRANSLATE函数使用模式匹配来查找和替换字符串中的字符。例如，下面的查询用0替换所有数字（0～9），用星号（*）替换除K之外的所有字母，字母K用X替换：

```
SELECT
    TRANSLATE('2KRW229',
    '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ',
    '0000000000**********X**************') translate_example
FROM DUAL;

translate_example
------------------------------
0X**000
```

该函数有助于提高安全性：在SQL代码内部，可以使用精确值（比如说信用卡号码），但是产生的输出结果比较混乱。这里给出一个信用卡的例子，它用星号代替所有数字：

www.TopSage.com

TopSage.com

```
SELECT
    TRANSLATE('4526 43567 6091 1506',
   '0123456789',
   '**********') visa_card_number
FROM dual;

visa_card_number
-----------------------
**** ***** **** ****
```

函数在 Oracle 和 DB2 9.5 中的用法是相同的。尽管 SQL Server 没有相应内置函数来实现这一功能，但可以用 STUFF 函数从某个位置移除指定个数的字符并用另外一个指定的字符串进行填充。第一个参数是字符串本身，第二个参数指定起始字符位置，第三个参数指定移除的字符个数（0 个或多个），第 4 个参数指定插入哪个字符串。下面的例子中，从参数字符串的第三个字符删除一个字符，插入 bbb 字符串：

```
SELECT
    STUFF('aaaaaaaaa', 3, 1, 'bbb') result

result
-----------
aabbbaaaaaa
```

可以使用 STUFF 函数（结合其他 SQL 函数）来创建用户定义的函数以实现 Oracle 和 DB2 9.5 的功能。

10.2.10 TRIM、LTRIM 和 RTTIM

尽管数据中的空格并不明显，但也是一个需要考虑的因素。通常，当键入一些字符值时，空格在用户界面中并不显示，这很容易产生错误。一个 RDBMS 需要绝对的精度——字符串‘user’和字符串‘user ’（结尾处有空格）是完全不同的。

用来去除空格（某些情况下可以是其他字符）的函数在三大 RDBMS 数据库系统中运行方式相似：它们去除字符串表达式的开头和结尾字符。主要的不同之处在于，Oracle 11g 支持大多数的 SQL 标准语法（除了 LTRIM 和 RTRIM 之外，还有 TRIM），允许去除空格以外的其他字符。IBM DB2 9.5 的函数 STRIP、TRIM、LTRIM 和 RTRIM 可用来实现相同的功能，Microsoft SQL Server 只对空格使用该函数。

思考下面的例子，它运行在 IBM DB2 9.5 中：

```
SELECT
    LENGTH(LTRIM('   three_blanks'))   ltrimmed,
    LENGTH(' three_blanks')  with_leading_blanks
FROM sysibm.sysdummy1
ltrimmed      with_leading_blanks
-----------   -------------------
12            15
```

Microsoft SQL Server 中的一段相似脚本会产生相同的结果：

```
SELECT
    LEN(LTRIM('   three_blanks')) ltrimmed,
    LEN('   three_blanks') with_leading_blanks
```

在 Oracle 11g 中使用 LTRIM 函数或 TRIM 函数可以得到相同的结果。LTRIM 的例子几乎与 IBM DB2 9.5 相同（只要用 dual 代替 sysibm.sysdummy1），因此省略：

```
SELECT
    LENGTH(TRIM(LEADING FROM '   three_blanks'))  ltrimmed,
    LENGTH('   three_blanks') with_leading_blanks
FROM dual;

ltrimmed      with_leading_blanks
-----------   -------------------
12            15
```

注意，在 Oracle 11g 中，函数 TRIM 的空格是默认的。为了从一个字符串表达式中去除空格以外的字符，可以使用下面的语法。例如，可以使用该函数从字符串值 M&M 或 IBM 的开头（前导）和末尾（尾部）去除字母 M（该语法在 Oracle 和 IBM DB2 中是一样的）：

```
SELECT
    TRIM(BOTH 'M' FROM 'M&M or IBM') trimmed
FROM dual;
```


www.TopSage.com

```
trimmed
-----------
&M or IB
```

Note 注意：在Oracle中，用于指定需要去除字母的参数总是区分大小写的。

10.3 日期和时间函数

表10-4中的函数处理日期和时间，它们接收范围广泛的参数数据类型，产生输出字符串（字符）、日期/时间或数字。

表10-4 日期和时间函数

ORACLE 11g	IBM DB2 9.5	Microsoft SQL Server 2008	描述
ADD_MONTHS (date,n)	DATE+n MONTHS TIME+n	DATEADD(month, number,date)	返回日期加上n个月（Oracle）；返回日期加上日期部分：年、月、日（Microsoft SQL Server）
CURRENT_DATE CURRENT_TIMESTAMP (precision)	CURRENT DATE CURRENT TIME CURRENT TIMESTAMP	GETDATE GETUTCDATE CURRENT_TIMESTAMP	返回当前的日期、时间，或者会话时区中的日期和时间
TO_DATE(value, format,nls_param)	DATE(value)	CONVERT CAST	返回基于本地特定格式的日期值（Oracle）
EXTRACT(day FROM date_expression)	DAY	DAY	返回指定日期时间表达式中的DAY部分（整数）
TO_CHAR(date, 'DAY'/'MONTH')	DAYNAME MONTHNAME	DATENAME(date Part,datetime)	返回请求日期部分的名称：日或月
EXTRACT(date Part FROM date_expression)	MONTH,DAY, HOUR etc.	DATEPART(date part,datetime)	返回请求的日期部分（日、月、年）
LAST_DAY (date_expression)	LAST_DAY (date_expression)	N/A	返回日期时间表达式参数中的月的最后一天
LOCALTIMESTAMP	CURRENT DATE CURRENT TIME	GETDATE ()	返回客户机系统的日期；可能包括微秒和一些RDBMS实现的时区
MONTH_BETWEEN (date1,date2)	Direct date arithmetic (operators)	DATEDIFF(MONTH, Start_date,end_date)	计算两个日期的差
NEW_TIME	CURRENT TIMEZONE	GETUTCDATE	返回相对于当前服务器日期时间的日期时间
NEXT_DAY	Date arithmetic	DATEADD(day,n,m)	计算相对于指定日期的下一个日期
ROUND (date,[format])	N/A	N/A	返回按照指定单位取整后的一定格式的日期；如果参数秒被省略，日期取整为最近的一天
SYSDATE	CURRENT DATE CURRENT TIME	GETDATE CURRENT_TIMESTAMP	返回RDBMS服务器的当前日期时间
SYSTIMESTAMP	CURRENT TIMESTAMP	CURRENT_TIMESTAMP GETDATE	返回数据库所在系统的系统日期；可能包括微秒和一些RDBMS实现的时区
TO_CHAR(datetime)	CAST(datetime AS CHAR)	CAST(datetime AS CHAR) CONVERT(char(n), date expression)	将日期时间数据类型转换为指定格式的字符串
TO_TIMESTAMP	TIMESTAMP	N/A	将表达式转换为TIMESTAMP数据类型的值
TO_TIMESTAMP_TZ	N/A	N/A	将CHAR、VARCHAR、NCHAR或NVARCHAR数据类型的数据转换为TIMESTAMP WITH TIME ZONE数据类型的值
TO_DSINTERVAL	N/A	N/A	将CHAR、VARCHAR、NCHAR或NVARCHAR数据类型的数据转换为INTERVAL DAY TO SECOND值

www.TopSage.com

续表

ORACLE 11g	IBM DB2 9.5	Microsoft SQL Server 2008	描述
TO_YMINTERVAL	N/A	N/A	将 CHAR、VARCHAR、NCHAR 或 NVARCHAR 数据类型的数据转换为 INTERVAL YEAR TO MONTH 类型，字符是要转换的数据
TRUNC(date)	N/A	N/A	返回按照指定单位截断后的一定格式的日期；如果参数秒被省略，日期截断为最近的一天
TZ_OFFSET (<timezon_name> | [DBTIMEZONE] | [SESSIONTIMEZONE])	CURRENT TIMEZONE	GETUTCDATE () - GETDATE ()	返回当前或协调世界时的指定时区的偏移量
EXTRACT(MONTH)	MONTH	MONTH	返回指定日期时间表达式的 MONTH 部分
EXTRACT(YEAR)	YEAR	YEAR	返回指定日期时间表达式的 YEAR 部分

10.3.1 GETDATE、SYSDATE 和 CURRENT DATE

为了记录数据库的时间变化，需要对系统的日期和时间设定进行存取。Oracle 实现了用伪列（可以认为是完成任务的函数）返回系统的当前日期和时间。Microsoft SQL Server 使用函数 GETDATE ()，IBM DB2 9.5 在查询的 SELECT 部分使用 CURRENT DATE 特殊注册表。表 10-5 中列出了这些函数（附相应输出示例）。

表 10-5　　获取 RDBMS 当前的日期

ORACLE	IBM DB2 9.5	SQL Server
SELECT SYSDATE [FROM DUAL];	SELECT CURRENT DATE [FROM SYSIBM.SYSDUM-MY1]	SELECT GETDATE ()
08-OCT-07 9:47:01 PM	10/8/2007	2007-10-08 10:21:11.571

使用各种供应商特定掩码、参数或转换函数对日期的输出结果进行格式化。请参考 RDBMS 手册获得更多格式化的信息。

时区函数

这些函数处理不同时区。“标准”函数总返回客户机所在位置的时区（也就是说，它们显示的会话时间不可更改），但有时，尤其是从不同时区与中心结点进行远程交换时，该函数就不适用。

Microsoft SQL Server 函数 GETUTCDATE 返回当前 UTC 时间。思考下面的查询，它从 GETDATE 函数和 GETUTCDATE 函数返回结果（后者返回不同时区的服务器时间）：

```
SELECT
    GETUTCDATE() utc_time,
    GETDATE() local_time

utc_time                    local_time
--------------------------  --------------------------
2007-07-30 20:52:15.860     2007-07-30 13:52:15.860
```

Oracle 的近似等效函数是 TZ_OFFSET 和 SYS_EXTRACT_UTC 的组合。NEW_TIME 提供了一种更直观的方法，它返回第一个指定时区按照第二个指定时区调整后的日期/时间。

例如，当华盛顿州的西雅图是 2007 年 7 月 30 晚上 8:52:15 时，想得到纽约的日期和时间，可以在 Oracle RDBMS 软件中执行下面的查询，强制 Oracle SQL*Plus 以扩展格式显示日期/时间（然后使用本章稍后介绍的 ALTER SESSION 语句）：

```
SELECT
    NEW_TIME(TO_DATE('07-30-2007 8:52:15 PM',
    'MM-DD-YY HH:MI:SS PM',
    'PST', 'EST') eastern_time
FROM dual;

eastern_time
--------------------------
7/30/2007  11:52:15 PM
```

表 10-6 中给出了函数 NEW_TIME 的说明。

大家网 TopSage.com

表 10-6 Oracle 的 NEW_TIME 函数中有效的时区值范围

时区	描述
AST(ADT)	大西洋标准（或夏令）时间
BST(BDT)	白令标准（或夏令）时间
CST(CDT)	中部标准（或夏令）时间
EST(EDT)	美国东部标准（或夏令）时间
GMT	格林威治标准时间
HST(HDT)	阿拉斯加-夏威夷标准（或夏令）时间
MST(MDT)	山区标准（或夏令）时间
NST	纽芬兰标准（或夏令）时间
PST(PDT)	太平洋标准（或夏令）时间
YST(YDT)	育空河标准（或夏令）时间

NEW_TIME 只接收表 10-6 指定的参数。函数 FROM_TZ 能够使用更多的时区值。请参考 Oracle 的文档获得详细信息。

在 DB2 9.5 中，可以使用 CURRENT TIMEZONE 特殊注册表来得到 UTC 和当前时区的时间差，在下面的例子中，输出结果显示太平洋时区和 UTC（协调世界时）之间相差 7 小时：

```
SELECT
    HOUR(CURRENT TIMEZONE) utc_difference
FROM sysibm.sysdummy1

utc_difference
--------------
            -7
```

Oracle 的 CURRENT_DATE 函数返回会话的当前日期，而不是服务器的时间；Microsoft SQL Server 的 GETDATE 完成同样功能。

注意：UTC 是协调世界时（Corrdinated Universal Time）的缩写，于 1972 年 1 月 1 日代替了早期使用的格林威治标准时间（Greenwich Mean Time, GMT）。GMT 基于横穿英国的格林威治的零子午线上的时间，它在 19 世纪成为事实上的标准，在很大程度上是因为英国海军和商人的成就。从实用性来说，UTC 本质上就是 GMT。

下面的例子说明了这一点。首先，要使 Oracle 返回扩展的日期/时间值，需要为 SQL*Plus 会话设置默认的日期格式：

```
SQL> ALTER SESSION SET NLS_DATE_FORMAT='DD-MON-YYYY HH24:MI:SS'
Session altered.
```

要使用函数 SYSDATE 得到当前系统日期，使用下面的查询：

```
SELECT
    SYSDATE server_date,
   CURRENT_DATE session_date
FROM dual;
server_date          session_date
--------------------  --------------------
30-JUL-2007 19:05:35  30-JUL-2007 19:05:35
```

可以看到，会话时间和系统时间是相同的。要改变会话时间，如将葡萄牙时区（值为'WET'）提前 8 个小时，使用下面的代码：

```
SQL> ALTER SESSION SET TIME ZONE ='WET';
Session altered.
```

提示：在 Oracle 中，可以指定时间格式为 HH:MI，或为系统提供预定义的 TIME_ZONE 值集。要找出该参数的有效值列表，请查询 Oracle 的动态性能视图 V$TIMEZONE_NAMES。

现在，会话时间比系统时间提前了 8 个小时：

```
SELECT
    SYSDATE server_date,
    CURRENT_DATE session_date
FROM dual;

server_date           session_date
--------------------  --------------------
7/30/2007 2:06:51 PM   7/30/2007 10:06:51 PM
```


www.TopSage.com

这些例子并没有完全包括时区功能和配置的用法，请参考特定 RDBMS 手册了解更多内容。

10.3.2 ADD_MONTHS、DATEADD 和日期算法

Microsoft SQL Server 的 DATEADD 函数返回一个新的日期时间值，它是通过日期参数与指定的日期部分相加得到的。除了可以对日期进行计算之外，Oracle 和 IBM DB2 还可以使用 ADD_MONTHS 函数。

在下面的示例中，查询返回从 2002-09-05 日以后的 5 个月的日期（请参考第 3 章获得有关日期时间类型和日期时间常量使用的更多信息）：

```
SELECT
    DATEADD(month,5,'2008-09-05') months
-------------------------
  2008-02-05 00:00:00.000
```

在 Oracle 中使用下面的查询可产生相同的结果：

```
SELECT
    ADD_MONTHS(TO_DATE('2008-09-05','YYYY-MM-DD'),5) months
FROM dual;

months
-------------------------
02-SEP-2008
```

DATEADD 函数也可以将日期与日、小时和分相加。Oracle 和 DB2 9.5 使用日期算法完成这一任务。例如，在 Oracle 中，使用下面的查询可在当前日期上加上 20 天：

```
SELECT
    SYSDATE + 20
FROM DUAL;
```

IBM DB2 9.5 中等价的查询为：

```
SELECT
    CURRENT DATE + 20 DAY
FROM sysibm.sysdummy1;
```

Oracle 在接收日期常量方面比 IBM 和 Microsoft RDBMS 更严格。因此，需要将常量 2003-09-05 显式转换为日期类型（请参阅第 3 章有关常量使用的内容，本章稍后将介绍转换函数）。

Note 注意：默认的输出格式取决于机器设置，并可以被应用程序覆盖。破折号可以代替斜线，年可以放在月和日的前面。

Cross-Reference 交叉参考：Oracle 11g 引入 INTERVAL 数据类型用于日期/时间计算，有效废除了 ADD_MONTH 函数。参阅第 3 章，了解有关时间间隔常量用法的更多信息。

IBM DB2 9.5 没有日期和时间计算的特定函数。例如，要在当前日期基础上增加 5 个月，可以使用下面的查询：

```
SELECT
    (CURRENT DATE + 5 MONTH) add_months
FROM sysibm.sysdummy1

add_months
------------
  12/30/2007
```

相同的语法可用于增加天数：

```
SELECT
    (CURRENT DATE + 5 DAYS)add_days
FROM sysibm.sysdummy1

add_days
------------
  08/04/2007
```

前面的查询说明了日期算法的概念：使用相应表达式可以增加或减去小时、分和秒。在表的查询中，可以用表达式或常量代替日期值。另外，注意 DAY/DAYS 和 MONTH/MONTHS 是可以互换的。

10.3.3 EXTRACT 和 DATEPART

Oracle 的 EXTRACT 函数返回日期/时间表达式指定部分的值。这些部分可以是 YEAR、MONTH、DAY、HOUR、MINUTE 和 SECOND。此外还允许指定 TIMEZONE_HOUR、TIMEZONE_MINUTE、TIMEZONE_REGION 和 TIMEZONE_ABBR（用于“缩写”）。这些功能调整时区之间的时差。下面的查询从当前日

www.TopSage.com

期中提取 YEAR：

```
SELECT
    EXTRACT(YEAR FROM SYSDATE) year_part
FROM DUAL;

year_part
--------------
         2007
```

日期/时间表达式必须使用正确的格式。例如，如果表达式中没有分钟，那么提取表达式中的 MINUTE 部分是无意义的（默认为 00）。

Microsoft SQL Server 2008 除了一般的 DATEPART（可以返回标准参数的更详细的信息，比如 YEAR、MONTH、DAY、HOUR、MINUTE 和 SECOND，必要时也可以返回毫秒）之外，还提供了一些专门的函数（如 MONTH、DAY 和 YEAR）从日期时间表达式中提取日期部分：

```
SELECT
    DATEPART( month, '08-05-2007') month_part
  MONTH('08-05-2007') month_part_too
month_part       month_part_too
--------------    -------------------
8                8
```

Tip 提示：可以使用函数 GETDATE 代替字符串作为前一节讨论的 Microsoft SQL Server 2008 函数的输入，字符串被隐式转换为日期时间表达式。Oracle 不能将字符串文字作为输入。

IBM DB2 9.5 为每个场合提供函数：YEAR、MONTH、DAY、HOUR、MINUTE 和 SECOND，从而为它的逻辑结论提供专门的方法。例如：

```
SELECT
    YEAR(CURRENT DATE) as year
  ,MONTH(CURRENT DATE) as month
  ,HOUR (CURRENT TIME) as hour
FROM sysibm.sysdummy1

YEAR    MONTH        HOUR
-------    -------    --------
  2007            8         10
```

10.3.4 DAYNAME、MONTHNAME 和 DATENAME

DB2 9.5 有许多专门函数用于返回日期的名称部分：这里给出一个在 IBM DB2 9.5 中运行这种函数的例子：

```
SELECT
    DAYNAME(CURRENT DATE) day_name
  MONTHNAME(CURRENT DATE) month_name
FROM sysibm.sysdummy1

day_name       month_name
-------------- --------------
Saturday       July
```

Microsoft SQL Server 2008 提供了一个 DATENAME 函数，包括 IBM DB2 9.5 中的 DAYNAME 函数和 MONTHNAME 函数的功能。下面的查询返回月份的名称：

```
SELECT
    DATENAME(MONTH, GETDATE()) month_name

month_name
-----------
       July
```

该函数还可以返回星期当中某一天的名称：

```
SELECT
    DATENAME(WEEKDAY, GETDATE()) day_name

day_name
-----------------
     Thursday
```

前面讨论过，Oracle 没有专门实现这些功能的函数。但是可以用适用格式的转换函数，这个函数与 SQL Server 中的 DATENAME 函数是类似的。

下面的查询分两次返回当天日期的日名称：第一次名称全部为大写字母，第二次为大小写混合的

www.TopSage.com

TopSage.com

字母：

```
SELECT
TO_CHAR(SYSDATE,'DAY') day_name_upper,
TO_CHAR(SYSDATE,'Day') day_name_mixed
FROM dual;
day_name_upper          day_name_mixed
--------------- ------------------
THURSDAY                Thursday
```

下面的查询返回当前日期的月名称：

```
SELECT
TO_CHAR(SYSDATE,'MONTH') day_name_upper,
TO_CHAR(SYSDATE,'month') day_name_lower
FROM dual;

day_name_upper          day_name_lower
-----------------       ------------
       JULY             july
```

注意，在 Oracle 11g 中，指定日期部分的名称时使用的字母大小不同，输出格式也不同。但该法则在 Microsoft SQL Server 2008 和 IBM DB2 9.5 中不适用。

10.3.5 MONTHS_BETWEEN 和 DATEDIFF

Oracle 的函数 MONTHS_BETWEEN 返回两个日期之间的月份数。如果第一个日期晚于第二个日期，结果为正数；如果第一个日期早于第二个日期，返回负数。如果被比较的两个日期有相同的日部分，结果总为整数。否则，Oracle 按每月 31 天并计算日期的时间部分，返回一个小数值：

```
SELECT
    MONTHS_BETWEEN
   (TO_DATE('08-05-2007','MM-DD-YYYY'),
    TO_DATE('10-05-2007','MM-DD-YYYY')) months_in_between,
    MONTHS_BETWEEN
   (TO_DATE('08-05-2007','MM-DD-YYYY'),
    TO_DATE('10-06-2007','MM-DD-YYYY')) months_fraction
FROM dual;

months_in_between       months_fraction
-----------------       -------------------
               -2         -2.03225806451613
```

Microsoft SQL Server 中的 DATEDIFF 函数返回两个日期之间的指定日期部分（或全部）。它与 Oracle 的函数遵循相同的规则，但是它的返回值表示日、月、分等：

```
SELECT
datediff(month, '08-05-2007','10-05-2007') months_in_between

months_in_between
-----------------
2
```

IBM DB2 9.5 使用日期算法：

```
SELECT
    (MONTH('08-05-2007') - MONTH('10-05-2007'))
     months_in_between
FROM sysibm.sysdummy1

months_in_between
-----------------
                2
```

使用完全相同的方式，可以通过 DAY、HOUR、MINUTE 或 SECOND 函数分别计算天、小时、分或秒的数量。

10.4 聚合函数

尽管从逻辑上讲聚合函数（参见表 10-7）属于第 8 章讨论的 SELECT 语句，并且已经详细讨论过了，不过本章还是做一个简单的描述以供参考。

SQL 标准包括大量的高级统计函数（也许有人可能永远不会遇到），它完成数据提取、转换和加载任务（ETL，extract-transform-load）。因此建议您参考不同供应商的文档来获取这些函数及用法的详细信息。很多供应商（特别是 Oracle 和 IBM DB2）可以帮助您了解这些信息。聚集列表请参见表 10-7。

www.TopSage.com

表 10-7 标准 SQL 聚合函数

SQL 函数	描述
AVG	返回一组数的平均值；NULL 值被省略；可以与 GROUP BY 子句一起使用
COUNT	返回查询检索到的行数；可以与 GROUP BY 子句一起使用
SUM	返回列中数字的总和；可以与 GROUP BY 子句一起使用
MAX	返回一个给定列中的最大值；可以与 GROUP BY 子句一起使用
MIN	返回一个给定列中的最小值；可以与 GROUP BY 子句一起使用
VAR_POP	返回一组数（不包括 NULL 值）的总方差
VAR_SAMP	返回一组数（不包括 NULL 值）的样本方差
STDDEV_POP	计算总体标准差并返回总体方差的平方根
STDDEV_SAMP	计算累积样本标准差并返回样本方差的平方根
COVAR_POP	返回总体协方差。它是自变量的平均值之差乘以自变量平均值之差，乘积的和值再除以行数。可以用表达式（SUM(X*Y)-SUM(Y)*SUM(X)/n）/n 来表示，n 是数对（X,Y）
COVAR_SAMP	基于前面介绍的总体协方差，但是除以现有行数减 1
CORR	返回总体协方差的比率，通过总体协方差除以自变量和因变量的总体标准差的乘积得到，可以使用方程式 COVAR_POP(x,y)/STDDEV_POP(x)*STDDEV_POP(y)表示
REGR_COUNT	返回组中剩余行的数量
REGR_R2	返回相关系数的平方
REGR_SLOPE	返回自变量/因变量对确定的线性方程的最小平方拟合的斜率
REGR_INTERCEPT	返回根据自变量/因变量对的值，采用最小二乘法计算的的线性方程的 y 轴截距
REGR_SXX	返回自变量的平方的总和
REGR_SYY	返回因变量的平方的总和
REGR_SYX	返回自变量与因变量乘积的总和
REGR_AVGX	返回自变量表达式的平均值
REGR_AVGY	返回因变量表达式的平均值
PERCENTILE_CONT	返回通过参数（0,1 之间的值）标注的连续的行组接收的插值，被认为是组中全部行数的一部分。该函数主要对三个值计算：行数（RN）、最小值（FRN）、最大值（CRN）。如果 RN=CRN=FRN，那么根据 RN 行的表达式计算；否则根据（CRN-RN）*（FRN 行表达式的值）+（RN-FRN）*（CRN 行表达式的值）计算
PERCENTILE_DISC	对定一个值 P，PERCENTILE_DISC 函数首先用 ORDER BY 子句对值进行排序，然后基于排序结果，选取最小累积分布的数
PERCENT_RANK	返回 R 行的相对位置，定义为(RK-1)/(NR-1)。RK 为 R 行的编号，NR 是 Rwindow 分区的总行数
CUME_DIST	返回在 R 行中的相对位置，定义为 NP/NR。NP 定义为在 window 分区（或 R）的排序中 R 行前面（或相等）的行数，NR 定义为 window 分区的总行数
DENSE_RANK	返回 R 行的编号。定义为 R 前面并包括 R 的行数。与 window 排序不同
RANK	返回 R 行的编号。定义为 R 前面但不包含 R 的行数加 1

聚合函数对一组值进行特定运算后返回一个值。聚合函数通常与 SELECT 语句的 GROUP BY 子句绑定，虽然这并不是聚合函数的必要条件。使用 GROUP BY 子句后，每个聚合函数不是针对整列，而是对每组值进行计算产生一个结果。

Note 注意：当然，因为聚合函数本质上是数学运算，专门针对数字进行运算。因此，参数必须为内置的数字数据类型，函数的结果也必须在该数据类型定义的范围之内。

供应商不仅自己决定执行哪种函数，而且在执行细节和名称上也有不同。对于高级统计函数来说更是如此。表 10-8 列出的聚合函数在三大 RDBMS 实现中都能使用。

表 10-8 聚合函数

Oracle 11g	IBM DB2 9.5	Microsoft SQL Server 2008	描述
AVG(number)	AVG(number)	AVG(number)	计算一定范围的数值的平均值
N/A	N/A	CHECKSUM_AGG	返回一组值的校验和。省略 NULL 值。后面可跟 OVER 子句
COLLECT	N/A	N/A	将参数作为任意类型的一列并结合选取的行创建输入类型的嵌套表。为得到函数的结果，需要将它用在 CAST 函数中

www.TopSage.com

续表

Oracle 11g	IBM DB2 9.5	Microsoft SQL Server 2008	描述
COUNT (number)	COUNT(integer) COUNT_BIG(bigint)	COUNT(integer) COUNT_BIG(bigint)	返回 SELECT 语句中的行数
CORR	N/A	N/A	返回一组数对的相关系数
MAX(number)	MAX(number)	MAX(number)	返回选取值中的最大值
MEDIAN	N/A	N/A	返回集合表达式的中间值
MIN(number)	MIN(number)	MIN(number)	返回选取值中的最小值
STDEV_SAMP	STDEV_SAMP	STDEV	返回指定表达式的所有值的标准差
STDEV_POP	STDDEV_POP STDDEV	STDEVP	返回指定表达式的所有值的总体标准差
VAR_SAMP	VARIANCE_SAMP VAR_SAMP	VAR	返回指定表达式的所有值的统计方差
VAR_POP	VARIANCE VAR_POPVAR	VARP	返回指定表达式的所有值的总体统计方差
SUM(number)	SUM(number)	SUM(number)	计算列中选取值的总和

注意：一些供应商（例如 Microsoft）不仅将这些函数作为 SQL 本身的一部分，也在它们的 OLAP 实现中应用这些函数。

10.4.1 SUM

SUM 函数对指定列中（或一个数字表达式/组合、多列等）的所有值求和。例如，如果想知道销售额的总和，可采用下面的查询：

```
SELECT
    SUM(total_price) total_sale
FROM v_customer_totals;

total_sale
---------------
457000.40
```

如果销售额按照客户进行了分组，下面的查询会有用：

```
SELECT
    customer_name,
    SUM(total_price) total_sale
    FROM v_customer_totals
GROUP BY customer_name;

customer_name               total_sale
--------------------------- ----------
WILE BESS COMPANY           276775.60
WILE ELECTROMATIC INC.      30956.20
WILE ELECTROMUSICAL INC.    19824.00
WILE ELECTRONICS INC.       28672.80
WILE SEAL CORP.             100771.80
```

该查询分别对每个客户的 total_price 列的值进行相加，并在 total_sale 列产生每个客户销售总额。

该函数可以使用 ALL 和 DISTINCT 限定词。它的思想很简单：ALL 包括列中的每一个值。而 DISTINCT 对相同的值只计数一次（例如，假设有多个销售额为 6608.00 美元，仅对一个进行计数）。思考下面的查询，比较使用和不使用 DISTINCT 谓词时 SUM 函数的结果：

```
SELECT
    SUM(DISTINCT total_price)distinct_price,
    SUM(ALL total_price) total_sale
FROM v_customer_totals;
distinct_price     total_sale
---------------- --------------------
165405.80          457000.40
```

比较上例查询的结果会发现 ALL 限定词是默认的。例如，当按照指定的价格区间分组并提取销售额的信息时，这些限定词是有用的。

SUM 函数的语法在三大 RDBMS 实现过程中是相同的。

可以使用 WHERE 子句进一步对值进行限定。例如，查找最后一个季度的销售总额。

www.TopSage.com

大家网 TopSage.com

交叉参考：请参考第8章，查看更多有关聚合函数使用和示例的详细讨论。

10.4.2 COUNT

该函数返回表或视图中的记录总数。为了在ACME数据库中查找客户订购订单的数量，需要对视图V_CUSTOMER_TOTALS进行查询：

```
SELECT
    COUNT(order_number) total_orders
    FROM v_customer_totals;

total_orders
------------
51
```

使用另一个COUNT查询会得到相同的结果：

```
SELECT
    COUNT(*) total_orders
FROM v_customer_totals;

total_orders
------------
51
```

结果取决于是否使用了ALL（默认）或DISTINCT谓词。在视图中，ORDER_NUMBER是唯一的，所以计算订单数量与计算视图中记录的总数得到相同的结果。如果在CUSTOMER_NAME列对订购了这些订单的客户进行计数，结果将会不同。下面的查询在CUSTOMER_NAME列中使用两次COUNT，一次使用DISTINCT谓词，另外一次使用ALL谓词（默认）：

```
SELECT
    COUNT(DISTINCT customer_name)total_customers,
    COUNT(ALL customer_name) all_records
FROM v_customer_totals;

total_customers all_records
--------------- -----------
5               51

(1 row(s) affected)
```

为了查找每个客户订购的订单数量，使用GROUP BY子句。下面的查询实现该功能：

```
SELECT
    customer_name,
    COUNT(order number) total_orders
FROM v_customer_totals
GROUP BY customer_name;

customer_name                total_orders
---------------------------- ----------
WILE BESS COMPANY             31
WILE ELECTROMATIC INC.         4
WILE ELECTROMUSICAL INC.       3
WILE ELECTRONICS INC.          3
WILE SEAL CORP.               10
```

注意：因为NULL表示数据不存在，当比较NULL(s)值时采用一种特殊的规则：不要使用等号(=)，因为NULL不与任何值(甚至另外一个NULL)相等。有些数据库允许设置兼容级别（例如，Microsoft SQL Server的SET ANSI_NULLS ON | OFF语句），但是NULL的标准测试是WHERE<field>IS NULL。

NULL和聚合函数

同样需要了解NULL在聚合函数中的使用。NULL值不能进行计算。如果所有的值都为NULL，那么结果将返回NULL，而不是0。思考下面的查询，它返回列PHONE_SALESMANIF_FN的计数：

```
SELECT
    COUNT(phone_salesmanid fn) count_no_nulls,
    COUNT (*)count_all
FROM phone;

count_no_nulls   count_all
---------------- -----------
12                86
```


www.TopSage.com

注意，当计算平均数、和等其他值时，NULL 值完全被省略。当使用 WHERE phone_salesmanid_fn IS NULL 语句显式请求记录时，返回 74 行记录，连同 12 个非 NULL 值组成了 86 行记录。

10.4.3 AVG

该函数生成一组数值的平均值。例如，要想知道视图 V_CUSTOMER_TOTALS 中订单的平均值，可以使用下面的语句：

```
SELECT
    AVG(total_price) average_price
FROM v_customer_totals;

average_price
---------------
8960.792156
```

该函数将视图中每个记录的 total_price 值相加，再除以记录个数。使用 SUM 函数和 COUNT 函数相结合可产生相同的结果：

```
SELECT
    SUM(total_price)/COUNT(order_number) average_price,
    FROM v_customer_totals;

average_price
-------------------
8960.792156
```

为得到每个客户订购的平均订单数，使用 GROUP BY 子句：

```
SELECT
    customer_name,
    AVG(total_price) average_order
FROM v_customer_totals
GROUP BY customer_name;

customer_name               average_order
--------------------------- -------------
WILE BESS COMPANY           8928.245161
WILE ELECTROMATIC INC.      7739.050000
WILE ELECTROMUSICAL INC.    6608.000000
WILE ELECTRONICS INC.       9557.600000
WILE SEAL CORP.             10077.180000
```

10.4.4 MIN 和 MAX

这些函数选取所有列值的最小值和最大值。下面的例子找出所有客户订购的最大数额和最小数额的订单：

```
SELECT
    MAX(total_price) max_order,
    MIN(total_price) min_order
FROM v_customer_totals;

max_order       min_order
--------------- -------------
15456.80        6608.00
```

为了查找特定客户的最大和最小数额订单，在查询中使用 GROUP BY 子句：

```
SELECT customer_name,
    MAX(total_price) max_order,
    MIN(total_price) min_order
FROM v_customer_totals
GROUP BY customer_name;

customer_name               max_order min_order
--------------------------- --------- -----------
WILE BESS COMPANY           15402.20  6608.00
WILE ELECTROMATIC INC.       9038.00  6608.00
WILE ELECTROMUSICAL INC.     6608.00  6608.00
WILE ELECTRONICS INC.       15456.80  6608.00
WILE SEAL CORP.             15456.80  6608.00
```

可以在 AVG、MIN 和 MAX 函数中使用 SQL 的限定词 ALL 和 DISTINCT。

Note 注意：三大 RDBMS 实现过程也将聚合函数与分析子句 OVER 结合使用，在应用相关的窗口函数之前，要对行集进行划分和排序。

www.TopSage.com

交叉参考：在供应商的产品中还实现了更多的聚合函数。供应商特定实现的详细列表，请参考附录G。

10.5 转换函数

有时需要将一种数据类型转换为另一种数据类型。在CONCAT函数的例子中，先要将数字转换为字符串，才能与其他字符串值进行连接。很重要的一点是，英语不是世界上唯一的语言，因此，对区域字符数据库的需求不断增加。转换函数提供了基于英语字符的数据转换，它可以用本地字母表中的字符集正确表示。常用转换函数如表10-9所示。

表10-9 转换函数

ORACLE 11g	IBM DB2 9.5	Microsoft SQL Server 2008	描述
CAST(数据类型 AS 数据类型)	CAST(数据类型 AS 数据类型)	CAST(数据类型 AS 数据类型) CONVERT(为数据类型,值,格式)	将一种数据类型转换为另一种数据类型，进行有意义的转换
CONVERT	N/A	N/A	将字符数据从一种字符集转换为另外一种字符集
TO_CHAR(表达式)	CHAR(表达式)	CAST[表达式 as VARCHAR/CHAR(N)]	将兼容数据类型的表达式转换为字符串
TO_DATE(表达式)	DATE(表达式)	CAST(表达式 asDATETIME)	将兼容数据类型或兼容格式的表达式转换为DATE/DATETIME数据类型

注意：有时一个RDBMS会隐式地将数据从一种类型转换为另外一种类型。这个功能是很方便的，但是有时也容易产生问题。其中的一个例子是INTEGER——在自动转换为INTEGER时，数字会丢失所有的十进制数字。

10.5.1 不同数据类型之间的转换

执行数据类型转换通常使用CAST和CONVERT两个函数。它们将数据从一种数据类型转换为另外一种数据类型。函数CAST在三大RDBMS实现过程中用法几乎相同。然而，在Oracle 11g（本章稍后讨论）中，CONVERT用于字符集之间的转换。在Microsoft SQL Server中，它几乎等同于CAST函数。

CAST函数的语法如下：

```
CAST (<expression> AS <into datatype>)
```

在三大实现过程中，CAST函数的用法稍有不同：IBM DB2 9.5、SQL Server和Oracle可以将任何内置数据类型转换为另外一种数据类型。另外，Oracle允许基于集合的数据类型（如VARRAY）作为该函数的有效参数。

如果转换不能完成，将抛出对应的错误信息。例如，如果表达式不能转换为数字，将返回ORA-01722:invalid number错误。

Oracle 11g的转换函数如表10-10所示。

表10-10 Oracle 11g的转换函数

SQL 函数	描述
ASCIISTR	接收任意字符集的表达式，返回数据库字符集中字符串的ASCII形式。非ASCII字符转换为\XXXX的形式，XXXX表示UTF-16编码单位
BIN_TO_NUM	将位向量（用逗号分割的一组0和1）转换为等值数
CHARTOROWID	将任意字符数据类型的值转换为ROWID数据类型
COMPOSE	接收混合字符串表达式作为参数，返回输入表达式字符集的完全规范化形式
DECOMPOSE	与COMPOSE函数相反：接收任意数据类型的字符串作为参数，返回在与输入相同的字符集中分解后的Unicode字符串
HEXTORAW	将字符集中字符串的十六进制位转换为原始值
NUMTODSINTERVAL	将一个表达式转换为INTERVAL DAY TO SECOND常量
NUMTOYMINTERVAL	将一个表达式转换为INTERVAL YEAR TO MONTH常量

www.TopSage.com

TopSage.com

续表

SQL 函数	描述
RAWTOHEX	将原始数据类型值转换为包含十六进制等价值的字符值
RAWTONHEX	与 RAWTOHEX 相似，它是对 NVARCHAR2 数据类型进行转换（区域字符集）
ROWIDTOCHAR	将一个 rowid 值转换为 VARCHAR2 数据类型（长度通常为 18 个字符）
ROWIDTONCHAR	将一个 rowid 值转换为 NVARCHAR2 数据类型（长度通常为 18 个字符）
TO_BINARY_DOUBLE	返回双精度浮点数。从非双精度数值转换时将会四舍五入
TO_BINARY_FLOAT	返回单精度浮点数。从非双精度数值转换时将会四舍五入
TO_CHAR(字符)	将 NCHAR、NVARCHAR2、CLOB 或 NCLOB 数据转换为数据库字符集。返回值总为 VARCHAR2
TO_CHAR(日期时间)	将日期时间或 DATE、TIMESTAMP、TIMESTAMP WITH TIME ZONE 或 TIMESTAMP WITH LOCAL TIME ZONE 数据类型的区间值转换为 VARCHAR2 数据类型（请参考供应商文档中有关可选格式参数的内容）
TO_CHAR(数字)	将数字转换为 VARCHAR2 数据类型的值（请参考供应商文档中有关可选格式参数的内容）
TO_CLOB	将 LOB 列中的 NCLOB 值或字符串转换为 CLOB 值（请参阅第 3 章关于这些数据类型的描述）
TO_DATE	将 CHAR、VARCHAR2、NCHAR 或 NVARCHAR2 数据类型的表达式转换为 DATE 数据类型。接收可选格式和本地参数
TO_DSINTERVAL	将 CHAR、VARCHAR2、NCHAR 或 NVARCHAR2 数据类型的表达式转换为 INTERVAL DAY TO SECOND 值
TO_LOB	将列中的 LONG 或 LONG RAW 值转换为 LOB 值。该函数只适用于 LONG 或 LONG RAW 列，只能用在 INSERT 语句的子查询的选择列表中
TO_MULTI_BYTE	返回单字节字符转换为相应多字节字符的表达式（用于那些多字节的非 ASCII 字符，在使用 UNICODE 之前使用它）
TO_NCHAR(字符)	将字符串表达式转换为区域字符集。与 TRANSLATE...USING 函数的区域字符集等价
TO_NCHAR(日期时间)	将日期时间或数据库字符集中的 DATE、TIMESTAMP、TIMESTAMP WITH TIME ZONE、TIMESTAMP WITH LOCAL TIME ZONE、INTERVAL MONTH TO YEAR 或 INTERVAL DAY TO SECOND 数据类型的区间值转换为区域字符集
TO_NCHAR(数字)	将数值表达式转换为区域字符集的字符串。接收可选的格式和区域语言设置参数
TO_NCLOB	将 LOB 列的 CLOB 值或字符串转换为 NCLOB 值
TO_NUMBER	将任意数值表达式转换为数字
TO_DSINTERVAL	将字符串数据类型转换为 INTERVAL DAY TO SECOND 值
TO_SINGLE_BYTE	将 char 类型的多字节字符转换为相应的单字节字符（用于那些多字节的非 ASCII 字符，在使用 UNICODE 之前使用它）
TO_TIMESTAMP	将兼容字符表达式转换为 TIMESTAMP 类型的值
TO_TIMESTAMP_TZ	将兼容字符表达式转换为 TIMESTAMP WITH TIME ZONE 类型的值
TO_YMINTERVAL	将兼容字符表达式转换为 INTERVAL YEAR TO MONTH 类型的值
TRANSLATE...USING	将字符表达式在数据库字符集和特定区域字符集之间转换
UNISTR	接收字符文本或表达式作为参数解析为字符数据，用区域字符集形式返回。数据库的区域字符集为 AL16UTF16 或 UTF8

相比之下，SQL Server CONVERT 函数的语法更复杂，增加了很多功能——不仅涉及数据类型，还涉及结果输出的格式。函数标记中参数 length 和 style 是可选的：length 用于具有长度的数据类型，例如 VARCHAR、CHAR、NCHAR 和 VARBINARY；style 用来将 datetime 和 smalldatetime 转换为文本（它定义了结果字符串格式化的方式，例如是否使用百元，或是否使用毫秒）。数据类型必须是内置的，不允许使用用户定义数据类型。

```
CONVERT (<data_type>[ (length) ] ,<expression> [,style] )
```

例如，ACME 数据库的视图 V_CUSTOMER_TOTAL 的列 TOTAL_PRICE 是数字数据类型。为了在数据前面加一个美元符号（$），在连接之前，先要将数字转换为字符数据类型（Oracle 对于该情况会执行隐式的转换，而 DB2 9.5 和 Microsoft SQL Server 都需要显式地转换，否则会产生错误）。

当连接数字（TOTAL_PRICE 列的值）和字符串（$）时，Oracle 会隐式地将数字数据类型（或那种情况下的任何兼容的数据类型）转换为字符串数据类型，如下所示：

```
SELECT
    customer_name,
     ('$' || total_price) price
```


www.TopSage.com

```
FROM v_customer_totals

customer_name                price
---------------------------- ----------
WILE BESS COMPANY            $7511.00
WILE BESS COMPANY            $8390.00
WILE ELECTROMUSICAL INC.     $6608.00
```

Note 注意：有关SQL的数据处理不会探究二进制码的细节。与其他RDBMS供应商不同，Oracle提供将二进制数据转换为数字的函数。例如，十进制的8用二进制表示为1000，函数BIN_TO_NUM(1,0,0,0)返回8。

下面是Microsoft SQL Server 2008中查询的语法（也可以使用CONVERT函数得到相同的结果）：

```
SELECT
    customer_name,
    '$' + CAST(TOTAL_PRICE as VARCHAR(10)) price
FROM v_customer_totals

customer_name                price
---------------------------- ----------
WILE BESS COMPANY            $7511.00
WILE BESS COMPANY            $8390.00
WILE ELECTROMUSICAL INC.     $6608.00
```

在DB2 9.5中使用下面的查询得到相似的结果：

```
SELECT
    customer_name,
    CONCAT('$', CAST(total_price_as CHAR(32)) price
FROM v_customer_totals

customer_name                price
---------------------------- ----------
WILE BESS COMPANY            $7511.00
WILE BESS COMPANY            $8390.00
WILE ELECTROMUSICAL INC.     $6608.00
```

Note 注意：格式模板（模型）是一组常量，用于定义在字符串中显示日期或数字的方式，或为将字符串转换为日期或数字，对字符串如何解释。格式模板不会改变数据的内部表示，只影响显示效果。Oracle和IBM使用格式模板，而Microsoft SQL Server为各种格式分配代码。

如果使用Microsoft SQL Server 2008的CONVERT函数，必须知道如何正确格式化输出结果，以避免混淆。使用下面的查询以mon dd yyyy hh:mi:ss:mmmAM（或PM）的格式显示当前日期，通常用在美国：

```
SELECT
    CONVERT(VARCHAR(25), GETDATE(),109)

formatted_date
--------------------------
Aug 11 2007 6:30:03:037P
```

使用下面的查询将相同的输出结果格式化为日本标准 YYYY/MM/DD：

```
SELECT
     CONVERT(VARCHAR(25), GETDATE(),111)

formatted_date
--------------------------
2007/08/11
```

上面代码的第三个参数（111）用于指定输出格式。表10-11中列出了Microsoft SQL Server 2008的CONVERT函数的一些有用的格式。对于表的最后一列输入/输出，当转换为日期时间数据类型时显示输入，当将日期时间数据转换为字符串时，显示输出。

表10-11 格式化Microsoft SQL Server 2008的CONVERT函数输出日期

标准	使用公元（YYYY）	不使用公元（YY）	输入/输出
默认	0或100	N/A MON DD	YYYY hh:mi AM/PM
美国	101	1	MM/DD/[YY]YY
ANSI	102	2	[YY]YY.mm.dd
英国/法国	103	3	DD/MM/[YY]YY
德国	104	4	DD.MM.YY[YY]

www.TopSage.com

续表

标准	使用公元（YYYY）	不使用公元（YY）	输入/输出
意大利	105	5	DD-MM-[YY]YY
N/A	106	6	DD MM [YY]YY
N/A	107	7	MON DD, [YY]YY
N/A	108	8	hh:mm:ss
默认和毫秒数	9 或 109	N/A	MON DD YYYY hh:mi:ss:mmm AM/PM
美国	110	10	MM-DD-[YY]YY
日本	111	11	[YY]YY/MM/DD
国际标准化组织(ISO)	112	12	[YY]YYMMDD
欧洲默认和毫秒数	13 或 11	3	N/A DD MON YYYY hh:mm:ss:mmm (24 h)
ODBC 规范	20 或 120	N/A	YYYY-MM-DD HH:MI:SS(24 h)
ODBC 规范(使用毫秒数)	21 或 121	N/A	YYYY-MM-DD HH:MI:SS.mmm(24 h)
ISO8601	126	N/A	YYYY-MM-DD Thh:mm:ss.mmm (no spaces)
ISO86012 使用时区 Z	127	N/A	yyyy-mm-ddThh:mi:ss.mmmZ (no spaces)
N/A	114	14	hh:mm:ss:mmm (24 h)

Note

注意：默认情况下，Microsoft SQL Server 2008 是将 2049 年作为截止日期来解释两位数的年，这表示参数 49 被解释为 2049，参数 50 则被解释为 1950，这与其他 Microsoft 应用程序（如 Microsoft Office）不一致，其他应用程序将 2030 年作为截止日期。强烈推荐使用 4 位数的年以避免混淆。

Oracle 为字符与数字之间、字符与日期之间的类型转换提供了多种格式。这里给出一些最常用的格式（如表 10-12 所示）。

表 10-12　　选定的 Oracle 日期时间格式模板

格式元素	描述	例子
AD A.D.	AD 指示符	TO_CHAR(SYSDATE,'YYYY AD')
AM A.M.	子午线指示符（AM/PM）	TO_CHAR(SYSDATE,'HH:MI:SS AM')
BC B.C.	BC 指示符（公元前）	TO_CHAR(SYSDATE,'YYYY BC')
D	星期几（从 1 到 7）	TO_CHAR(SYSDATE,'D')
DAY	日名，填充空格至总长度为 9 个字母	TO_CHAR(SYSDATE,'DAY')
DD	月中的某天（从 1 到 31）	TO_CHAR(SYSDATE,'DD')
DDD	年中的某天（从 1 到 366）	TO_CHAR(SYSDATE,'DDD')
DY	日的缩写	TO_CHAR(SYSDATE,'DY')
FF[1..9]	小数表示的秒；不显示基数字符（使用 X 格式元素添加基数字符）	TO_CHAR(SYSTIMESTAMP,'SS.FF5')
FM	返回无前导或结束空格的值	TO_CHAR(SYSDATE,'FM')
HH	天中小时（从 1 到 12）	TO_CHAR(SYSDATE,'HH')
HH12	天中小时（从 1 到 12）	TO_CHAR(SYSDATE,'HH12')
HH24	天中小时（从 0 到 23）	TO_CHAR(SYSDATE,'HH24')
IW	基于 ISO 标准的年中的周（1-52 或 1-53）	TO_CHAR(SYSDATE,'IW')
IYYY IYY IY I	ISO 标准的年的最后 4、3、2 或 1 位	TO_CHAR(SYSDATE,'IYY')
J	儒略日，从公元前 1 月 1 日起的天数，返回值为整数	TO_CHAR(SYSDATE,'J')
MI	分（从 0 到 59）	TO_CHAR(SYSDATE,'MI')
MM	月（从 01 到 12）	TO_CHAR(SYSDATE,'MO')
MON	月的缩略名	TO_CHAR(SYSDATE,'MON')
MONTH	月名，填充空格至总长度为 9 个字母	TO_CHAR(SYSDATE,'MONTH')
PM	子午线指示符（AM/PM）	TO_CHAR(SYSDATE,'PM')
RM	罗马数字月（从 I 到XII）	TO_CHAR(SYSDATE,'RM')
RR	计算两位数的年	TO_CHAR(SYSDATE,'RR')
SS	秒（从 0 到 59）	TO_CHAR(SYSDATE,'SS')

www.TopSage.com

大家网 TopSage.com

续表

格式元素	描述	例子
SSSSS	午夜后的秒（0-86399）	TO_CHAR(SYSDATE,'SS')
TS	返回短时间格式值。时间各部分（小时，分等）的外观显示取决于TO_CHAR(SYS DA-TE,'SS')初始化参数NLS_TERRITORY和NLS_L-ANGUAGE	TO_CHAR(SYSDATE,'TS')
TZD	夏令时信息。TZD值是包含夏令时信息的缩写时区字符串。它必须与TZR中指定的地区一致	TO_CHAR(SYSDATE,'TZD')
TZH	时区小时（参见TZM格式元素）	TO_CHAR(SYSDATE,'TZH')
TZM	时区分（参见TZH格式元素）	TO_CHAR(SYSDATE,'TZM')
TZR	时区地区信息。值必须是数据支持的时区地区之一	TO_CHAR(SYSDATE,'TZR')
WW	年中的周（1-53）第一周从这一年的第一天开始持续到这一年的第7天	TO_CHAR(SYSDATE,'WW')
W	月中的周（1-5）第一周从月的第一天到第7天	TO_CHAR(SYSDATE,'W')
X	本地基数字符（例如，'HH:MI:SSXFF'）	TO_CHAR(SYSDATE,'X')
Y.YYY	年，中间有逗号	TO_CHAR(SYSDATE,'Y.YYY')
YEAR SYEAR	年，添加BC前缀和减号（-）	TO_CHAR(SYSDATE,'YEAR')
YYYY SYYYY	4位的年。添加BC前缀	TO_CHAR(SYSDATE,'SYYYY')
YYY YY Y	年的最后3位、2位或1位	TO_CHAR(SYSDATE,'YYY')

TO_DATE、TO_TIMESTAMP、TO_TIMESTAMP_TZ、TO_YMINTERVAL和TO_DS_INTERVAL（请参考附录G了解这些函数的语法和简要说明）等与日期相关的函数接受字符到日期/日期到字符格式模板。将字符数据转换为日期类型数据时，函数TO_CHAR也接受这些格式模板。这些格式元素之间的分隔字符可以是破折号（-）、斜线（/）、逗号（,）、分号（;）、单引号（' ）或冒号（:）。表10-12中的示例列说明如何在Oracle的SYSDATE函数返回的当前日期上使用格式模板。

每个元素都能与表中其他有效的元素一起使用。例如，使用下面的SQL查询产生<date><full day's name><full month name><spelled out year>格式字符串来表示当前日期。

```
SELECT
    TO_CHAR(SYSDATE,'DD-DAY-MONTH-YEAR') LONG_DATE
FROM dual;

LONG_DATE
-------------------------------------
31-TUESDAY -JULY [NR22]-TWO THOUSAND SEVEN
```

RR日期格式元素需要进一步解释一下。它代表不完整的两位数的年（是否记得所谓的“千年虫”危机）。如果函数TO_DATE的输入值是年的最后两位数而且小于50，而当前的最后两位小于或等于50，则为21世纪的日期：

```
select
    TO_DATE('09/12/49','DD/MM/RR') a_date
FROM dual;

a_date
--------------
  09-DEC-49
```

为使输出更容易理解，需要看到日期中年的完整4位数。本章前面时区的例子说明如何通过修改会话设置来改变输出显示格式。类似地，下面的语句在Oracle的SQL*Plus程序中显示年的完整4位数：

```
ALTER SESSION
SET  NLS_DATE_FORMAT = 'DD-MON-YYYY';
```

下面的查询同样会输出4位数年：

```
SELECT
    TO_DATE('09/12/49','DD/MM/RR') a_date
FROM dual;

a_date
--------------
  09-DEC-2049
```

如果提供的日期位大于等于50，当年的最后两位小于等于50，则结果日期为20世纪日期（当年

www.TopSage.com

日期的前两位减 1）：

```
SELECT
    TO_DATE('09/12/51','DD/MM/RR') a_date
FROM dual;

a_date
-----------
09-DEC-1951
```

Oracle 同样允许区域字符集的转换。例如，使用下面查询，实现对一个法语表示的日期常量进行转换：

```
SELECT TO_DATE(
    'Elokuu 10, 2007, 11:00',
    'Month dd, YYYY, HH:MI',
    'NLS_DATE_LANGUAGE = FINNISH') as a_date
FROM DUAL;

a_date
----------------------
8/10/2007 11:00:00 AM
```

表 10-10 中的格式模板是用于将数字数据转换为指定格式的字符串。例如，下面的查询显示使用 TO_CHAR 函数转换的结果：

```
SELECT
    TO_CHAR(-1234,'9999MI') result
FROM dual;

result
-----------
  1234-
```

表 10-13　　选定的 Oracle 数字格式模板

格式元素	说明	示例
$	返回值前加一个美元符号	TO_CHAR(1234,'$9999')
0	返回前导和/或尾部 0	TO_CHAR(1234,'09999')
9	返回指定位数的值，正数前加前导空格，负数前加前导减号	TO_CHAR(1234,'9999')
B	当定点小数的整数部分为 0 时，整数部分返回空格（不考虑格式模板中的 0）	TO_CHAR(1234,'B9999')
C	在请求位置返回一个 ISO 货币符号（由 Oracle 的 NLS_- ISO_CURRENCY 参数定义）	TO_CHAR(1234,'C9999')
D	在请求位置返回一个 ISO 十进制字符（由 Oracle 的 NLS_- NUMERIC_CHARACTER 参数定义）	TO_CHAR(1234,'99D99')
EEEE	返回科学记数法表示的值	TO_CHAR(1234,'9.9EEEE')
FM	返回无前导或结尾空格的值	TO_CHAR(1234,'FM9999')
MI	返回结尾带减号的负数，或结尾带空格的正数	TO_CHAR(-1234,'9999MI')
PR	返回带尖括号的负数，有前导和结尾空格的正数	TO_CHAR(-1234,'9999PR')
RN/rn	返回大写或小写罗马数字表示的值	TO_CHAR(1234,'RN')
S	在数字的开头或结尾附加减号或加号	TO_CHAR(1234,'S9999')
X	返回指定小数位数的十六进制值，非整数值则四舍五入	TO_CHAR(1234,'XXXX')

IBM DB2 9.5 没有为函数提供太多的格式设置选项，而是提供大量单个函数来处理日期的各个部分，从而解决解决格式设置问题，如函数 DAY、YEAR、MICROSECOND、MINUTE 和 MIDNIGHT_SECONDS。

要转换为日期或时间格式，文字字符串必须满足表 10-14 和表 10-15 中的格式。

表 10-14　　IBM DB2 9.5 数据字符串格式

格式	模板	示例
国际标准化组织(ISO) 日本工业标准(JIS)	YYYY-MM-DD	2002-09-12
IBM 美国标准	MM/DD/YYYY	09/12/2002
IBM 欧洲标准	DD.MM.YYYY	12.09.2002
Database Custom-Defined	取决于数据库国家/地区代码	N/A

www.TopSage.com

表 10-15 IBM DB2 9.5 时间字符串格式

格式	模板	示例
国际标准化组织(ISO) 日本工业标准(JIS)	HH.MM.SS	22.45.02
IBM 美国标准	HH:MM AM/PM	10.45 PM
IBM 欧洲标准	HH.MM.SS	22.45.02
Database Custom-Defined	取决于数据库国家/地区代码	N/A

例如，下面的查询接收表中的任何一种数据格式，并转换为内部 IBM 格式的 DATE 数据类型：

```
SELECT
    DATE('2007-09-12') ISO
    DATE('09/12/2007') USA
    DATE('12.09.2007') EUR
FROM sysibm.sysdummy1

ISO         USA         EUR
----------- ----------- -----------
09/12/2007  09/12/2007  09/12/2007
```

DB2 9.5 的 TIMESTAMP/TIMESTAMP_FORMAT 函数只有两种格式：YYYY-MM-DD HH:MM:SS 和 YYYY-MM-DD HH:MM:SS:nnnnnnn，实现从字符串到时间戳或时间戳到字符串的转换。

函数 CHAR 和 VARCHAR 用于将日期或时间格式转换为字符串，但没有应用模板。结果字符串总是用系统指定的格式表示，该格式在数据库层是可以配置的。可以通过一个自定义函数来弥补这一缺憾。

一种数据类型只能转换为另外一种兼容的数据类型。更糟糕的是，每个数据库都有各自的兼容标准：在一种 RDBMS 中兼容的数据类型可能在另外一个中不兼容。例如，在前面的例子中，我们在 DB2 9.5 中不能将 DECIMAL 转换为 VARCHAR，因为在这个 RDBMS 中不兼容（如果试图进行转换将弹出以下错误消息：SQL0461 N A value with data type''SYSIBM.DECIMAL''can't be CAST to type "SYSIBM.VARCHAR"），但同样的操作在 Microsoft SQL Server 或 Oracle 中是完全合法的。

10.5.2 在不同字符集之间转换

Microsoft SQL Server 使用函数 NCHAR 和 UNICODE 进行字符集之间的转换。NCHAR 函数按给定的整数代码（根据 Unicode 标准定义）返回 UNICODE 字符，UNICODE 函数返回与 Unicode 字符一致的数字。

例如，下面的操作为斯堪的纳维亚字符ϕ查找一个相应的 NUICODE 数字：

```
SELECT UNICODE('Ø') uni_code

uni_code
----------
216
```

然后，我们可以将数字传给 NCHAR 函数来显示该字符：

```
SELECT NCHAR(216) uni_character

uni_char
--------
Ø
```

10.5.3 Oracle 11g 的 CONVERT、TRANSLATE...USING 和 UNISTR

函数 TRANSLATE...USING 将一种数据库默认字符集中的文本转换为另一种字符集中的文本。从某种意义上来说，它和 Oracle 的 CONVERT 函数很像，只是 CONVERT 函数操作的是 NCHAR 或 NVARCHAR 数据类型的字符串（而不是**文本**）。

下面的例子中要转换三个字符："A 元音变音"（德语和斯堪的纳维亚语中的 A 上方有两个点）、"ϕ"（属于斯堪的纳维亚字母表）、普通的"A"，将它们从 ISO 8859-1 西欧 8 位字符集转换为美国 7 位 ASCII 字符集：

```
SELECT
    CONVERT('··A A Ø', 'US7ASCII', 'WE8ISO8859P1') translation
FROM dual;
```


www.TopSage.com

大学网 TopSage.com

```
translation
--------------------
A A ?
```

可以看到，“A 元音变音”转换为普通的“A”，这是因为在基于英语的美国 7 位 ASCII 字符集中没有该字符。当没有可用的替换对象，就显示一个问号。替换字符的映射规则由字符集本身定义。

注意：CHAR_CS 值返回由服务器指定的 Oracle 数据库的字符集 ID。在区域字符集中等价的 NCHAR_CS 值同样返回字符集 ID。

函数 TRANSLATE...USING 返回相同的结果：

```
SELECT
    TRANSLATE('··Ä A Ø' USING CHAR_CS) translation
FROM dual;

translation
--------------------
A A ?
```

Oracle 的函数 UNSTR 实现从任意字符集到 Unicode 字符集（请参阅第 2 章获取有关 Unicode 更多内容）的字符串转换。其中的反斜杠是一个换码符，它表示由十六进制数字转换为 Unicode 字符而不是常量（参见附录 L 了解有关十六进制数字的更多内容）：

```
SELECT
    UNISTR('\00F5'|| '\00D1') UN
FROM dual;

UN
---
õÑ
```

如果查看 Oracle 为每个字符分配的字节数（使用 Oracle 函数 LENGTHB），将会发现一个有趣的事实：ASCII 和 Unicode 字符集中字符的长度相同，但是，一个 Unicode 字符占 2 个字节，而 ASCII 中一个字符占 1 个字节：

```
SELECT
    LENGTHB(unistr('\00F5'))     in_bytes,
    LENGTH(unistr('\00F5'))      in_chars,
    LENGTHB('A')                 in_bytes,
    LENGTH(unistr('A'))          in_chars
FROM dual;

  IN_BYTES   IN_CHARS   IN_BYTES   IN_CHARS
---------- ---------- ---------- ----------
         2          1          1          1
```

Oracle 也提供了函数将字符各部分组装成一个字符。例如，函数 COMPOSE 和它的反函数 DECOMPOSE 接收字符和限定符，返回数据库字符集中完全规范化的字符：

```
SELECT COMPOSE ('o' || UNISTR('\0302')) AS circumflex
  ,COMPOSE ('o' || UNISTR('\0303')) AS tilde
    ,DECOMPOSE (COMPOSE('o' || UNISTR('\0302'))) AS decomposed
FROM DUAL;

UN
---
  circumflex     tilde   decomposed
------------ ---------- ----------
        ô          õ         o¿
```

注意，在数据库字符集（美国英语）中长音不包含单个字符，因此返回一个问号。

IBM DB2 9.5 使用 VARGRAPHIC 函数进行转换，还可以在数据库级别、绑定应用程序级别和段级别设置不同的代码页。有许多函数集在其他 RDBMS 中并不存在，这表明 DB2 9.5 是 AS/400 系列的遗留产物。IBM 假设为数据库设置特定字符集并保持这个字符集。随着 RDBMS 不断向 UNICODE 字符集靠拢，代码页之间的转换问题变得不太重要。UNICODE 为书面语言提供了大量的字符集。

10.5.4 特定于数据类型的转换函数

除了通用的 CAST 函数之外，Oracle 还有大量专门用于一种数据类型的特定转换函数，DB2 也同样如此，通常在重载数据类型声明时指定转换函数。

用任何一种函数得到的结果相同。使用它们的唯一原因是方便，在某些情况下是因为执行速度快。

下面查询

```
SELECT
    CAST (SYSDATE AS VARCHAR2(10)) char_date
FROM dual;

char_date
----------
10-JUL-07
```

其结果与 Oracle 的相同

```
SELECT
    TO_CHAR(SYSDATE)
FROM dual;

char_date
----------
10-JUL-07
```

与 IBM DB2 9.5 结果也一致：

```
SELECT
    CHAR(CURRENT DATE) char_date
FROM sysibm.sysdummy1

char_date
----------
09/10/07
```

注意：显示的字符串的实际格式取决于系统设置，但通常可以通过格式函数改变。

Oracle 特定类型的转换函数包括 TO_DATE、TO_CHAR、TO_NUMBER 和 TO_CLOB。前面已经提到过，IBM DB2 9.5 数据类型声明通过附加的转换功能重载。

交叉参考：请参阅附录 G，了解供应商特定函数完整列表。

交叉参考：一种数据类型的值可以转换为另外一种数据类型的值，必须保证两种数据类型之间是兼容的。因为每个 RDBMS 实现都有各自的数据类型，所以转换规则也各不相同。附录 K 包含 Microsoft SQL Server、Oracle 11g 和 IBM DB2 9.5 的数据类型转换的矩阵表。

10.6 系统函数

每个 RDBMS 提供了许多用于存取数据库本身信息的函数。根据定义，这些函数**不是** SQL 标准的一部分，因为各种 RDBMS 实现都有本质上的差别。另外，它们大多数是供数据库管理员（Database Administrator，DBA）使用的。

尽管如此，它们在 SQL 程序员的工具箱中可能是有价值的工具。我们将简要讨论其中一些最常用的函数，如表 10-16 所示。

表 10-16 有用的 Oracle 11g 系统函数

函数	参数	描述
SYS_CONTEXT	'namespace' 'parameter',[length]	返回与上下文命名空间有关的参数值。Oracle 提供了内置的命名空间 USERENV，它描述当前的会话；另外，用户可使用 CREATE CONTEXT 语句自定义上下文
SYS_GUID	无	生成并返回一个全局唯一标识符（RAW 值），由 16 个字节组成
SYS_TYPEID	<object type value>	返回操作数（UDT 用户定义类型）的指定类型的 typeid
UID	无	返回唯一识别会话用户（登录用户）的整数
USER	无	以数据类型 VARCHAR2 返回会话用户的名称（登录用户）
USERENV	'parameter'	返回当前会话的信息

系统函数与任何其他内置 SQL 函数用法相同。例如，下面的查询根据使用的语言和地区返回当前会话的数据库字符集、登录用户名称和唯一会话用户标识符：

```
SELECT
     USERENV('LANGUAGE') AS charset,
  USER,
```


www.TopSage.com

```
  UID
FROM customer;

CHARSET                          USER    UID
-------------------------------  ------  -----
AMERICAN_AMERICA.WE8ISO8859P1    ACME    86
```

当调用函数 USERENV 或 SYS_CONTEXT 来检索信息（例如 CLIENT_INFO 或 LANG 等）时，可以使用更多的预定义值。请参考供应商文档获得更多信息。

Microsoft SQL Server 2008 提供了大量查询系统配置的函数。表 10-17 列出一些在查询中有用的函数。

表 10-17 有用的 Microsoft SQL Server 2008 系统和配置函数

格式	参数	描述
@@DATEFIRST	无	返回会话的当前值。参数可以设置为整个服务器，或在会话中通过 SET DATEFIRST 语句进行设置
@@LANGUAGE	无	返回当前使用的语言名称，也可以通过 SET LANGUAGE 语句改变语言
@@VERSION	无	返回当前安装的 SQL Server 的版本、处理器体系结构、创建日期和操作系统
COLLATIONPROPERTY	<collation_name>, <property>	返回指定排序规则的属性。属性包括 LCID（Windows 区域设置标识符）、比较风格（二进制排序规则为 0）和版本（SQL Server 2005 之前的版本返回 0，之后的所有版本全部为 1）
CURRENT_USER	无	返回当前用户的名称，与 USER_NAME（）作用相同
FORMATMESSAGE	<message number>, <prams (up to 20)>	通过现有的 sys.message 构造一条消息；与 RAISERROR 类似，最大的不同的是，后者立刻将消息打印到标准输出，不能在变量中存储
GETANSINULL	['database']	返回数据库默认的为空性。可选的数据库名称参数指定哪个数据库请求该值（如果被省略，那么函数返回当前数据库参数）。 为空性在数据库层使用 SET ANSI_NULL_DFLT_ON ON | OFF 设置，如果 ANSI NULL 设为 ON，则返回 1
HOST_ID	无	返回客户工作站的标识号
HOST_NAME	无	返回客户工作站的 Windows 名称
IDENT_CURRENT	<table_name>	返回任意会话和作用域中指定表或视图生成的最后标识值（类似于 Oracle 和 IBM DB2 中的顺序对象）
IDENT_INCR	<table_name>	返回在包含标识列的表或视图中创建标识列的过程中指定的增量值[以数字形式返回（@@MAXPRECISION,0）]（类似于 Oracle 和 IBM DB2 中的顺序对象）
IDENT_SEED	<table_name>	返回包含标识列的表或创建标识列的视图指定的种子值[以数字形式返回（@@MAXPRECISION,0）]（类似于 Oracle 和 IBM DB2 中的顺序对象）
SESSION_USER	无	SESSION_USER 返回当前数据库的用户名，该函数与大多数内置函数不同，在标识中不需要括号
SCOPE_IDENTITY	无	返回插入相同作用域的标识列的最后标识值。作用域就是一个模块：存储过程、触发器、函数或程序组。因此，在同一预存程序、函数或程序组中的两个语句一定在同一作用域内
SYSTEM_USER	无	返回当前登录到系统的用户。如果模拟（EXECUTE AS）登录，则返回模拟登录
USER_NAME	[user_id]	返回指定标识号的数据库用户名，如果参数被省略，则返回当前会话的用户 ID

下面的例子说明 Microsoft SQL Server 2008 系统函数的用法：

```
SELECT @@LANGUAGE       AS language
,CURRENT_USER           AS [user]
,HOST_NAME()            AS host
,SYSTEM_USER            AS sysuser

language        user      host          sysuser
--------------  --------  ------------  ------------------
us_english      dbo       ACME-T60      ACME-T60\akriegel
```

Note 注意：在上述查询中，“user”代表 SQL 中具有特定含义的关键字。方括号用于通知 SQL Server 它是一个常量。

IBM DB2 UDB 9.5 没有系统函数的概念，而是提供特殊注册表，这个注册表可以认为被认为是接收 0 个参数或“全局”值的 niladic 函数。

www.TopSage.com

大家网 TopSage.com

注册表的用法与其他 SQL 内置函数一样，非常简单。与其他函数不同的是返回值。根据定义，IBM DB2 特殊注册表也可以用于设置值，如表 10-18 所示。

表 10-18　　有用的 IBM DB2 9.5 特殊注册表

注册表	描述
CURRENT CLIENT_APPLNAME	返回一个已经建立连接的应用程序名称的值
CURRENT CLIENT_USERID	返回为连接指定的客户端用户 ID 值
CURRENT DATE	返回基于服务器时间的 SQL 语句执行的日期
CURRENT ISOLATION	为当前会话上下文中的语句返回或设置隔离级别（两个字符的字符串：UR/CS/RR 和 RS）
CURRENT LOCK TIMEOUT	返回或设置返回错误之前等待的秒数
CURRENT PATH	返回或设置 SQL 路径，用于为动态期望的 SQL 语句解析函数引用和数据类型
CURRENT SCHEMA	返回或设置[VARCHAR(128)]值，该值标识的架构名用于限制数据库对象
CURRENT SERVER	返回标识当前应用服务器的值
CURRENT TIME	返回基于服务器时间的执行 SQL 语句的时间戳
CURRENT TIMESTAMP	返回基于应用服务器时间的执行 SQL 语句的时间戳
CURRENT TIME ZONE	返回或设置 UTC 与应用程序服务器上的本地时间的差值
CURRENT USER	返回用于语句授权的授权 ID
SESSION_USER	为当前会话返回或设置授权 ID
SYSTEM_USER	为连接到数据库的用户返回一个授权 ID
USER	返回传递给数据库管理程序的一个运行时授权 ID

10.7　杂项函数

在每个分类中，总有一些函数不符合单个定义明确的分类。我们将这样函数归为“杂项”类别（参见表 10-19）。

表 10-19　　杂项函数

ORACLE 11g	IBM DB2 9.5	Microsoft SQL Server 2008	描述
COALESCE(expression1,expression2, expression3…)	COALESCE(expressi-on1, expression2, expression3…) VALUE	COALESCE(expression1,expression2, expression3…)	返回列表中第一个不为 NULL 的参数
CASE(expression)WHEN <compare value > THEN <substitute value>ELSE END DECODE(expression,compare value,substitute value…)	CASE(expression) WHEN <compare value> THEN <substitute value >ELSE END	CASE(expression) WHEN <compare value> THEN <substitute value >ELSE END	将输入表达式与某些预定义的值进行比较，并输出一个硬编码或计算值作为替代值
GREATEST	N/A	N/A	返回一个或多个表达式列表的最大值。最大值由数字或字母表顺序确定（应用的顺序通过第一个表达式的数据类型确定）
LEAST			返回一个或多个表达式列表的最小值。最大值由数字或字母表顺序确定（应用的顺序通过第一个表达式的数据类型确定）
LNNVL	N/A	N/A	提供一种当一个或两个操作数为 NULL 时，对条件求值的方法。函数只能用在查询的 WHERE 子句中
NULLIF(expression1, expression2)	NULLIF(expression1, expression2)	NULLIF (expression1, expression2)	比较两个表达式，如果全部为 NULL，则返回 NULL；否则，返回第一个表达式
NVL(expression,value)	COALESCE (expression, value)	ISNULL(expression,value)	验证表达式是否为 NULL，如果是，函数返回一个指定值
NVL2(expression,value1, value2)	N/A	N/A	如果表达式为 NULL，返回第一个值；否则返回第二个值
VSIZE	LENGTH	DATALENGTH	返回表达式内部表示的字节数，如果表达式为 NULL，则函数返回 NULL

www.TopSage.com

10.7.1 DECODE 和 CASE

CASE 是 IBM DB2 和 Microsoft SQL Server 中实现的 SQL:1999 关键字，在 Oracle 中以函数 DECODE 形式存在。Microsoft 和 DB2 9.5 允许 CASE 表达式 SQL/Foundation（ISO/IEC 9075-2）与标准 SELECT 语句一起使用，DB2 9.5 也将它作为一个 CASE 语句（与 SWITCH 语句类似），作为 SQL 程序扩展的一部分，CASE 语句用 SQL/PSM（ISO/IEC 9075-4）定义。

Oracle 的 DECODE 函数根据某些条件（IF...THEN...ELSE 逻辑），允许修改 SELECT 语句的输出。它依次将表达式（通常为列值）与搜索值进行比较，如果相匹配，函数返回相应结果，否则返回默认值。如果没有匹配值或没有指定默认值，则函数返回 NULL。除了 DECODE，Oracle 11g 还有一个 CASE 语句，用法和在其他 RDBMS 中相同。

CASE 语句使用更少隐藏的语法产生相似的结果，并且不涉及函数。

例如，可以准备一个包含客户名和信用状况的列表。在 ACME 数据库的 CUSTOMER 表中，列 CUST_CREDHOLD_S 定义是否允许这个特定客户赊欠订购。如果可以，该列值为 Y，否则为 N。检索两列的简单 SELECT 语句如下所示（在三大供应商中都是有效的）：

```
SELECT
    cust_name_s,
    cust_credhold_s
FROM customer;

cust_name_s                  cust_credhold_s
---------------------------- ---------------
WILE SEAL CORP.               Y
MAGNETICS USA INC.            N
MAGNETOMETRIC DEVICES INC.    N
FAIR PARK GARDENS             N
```

尽管这一报表从技术上讲是正确的，但还需要附加信息来解释隐藏的 Y 和 N。在 Oracle 中使用下面的查询可以解决这一问题（使用 DECODE 函数）：

```
SELECT
cust_name_s,
   DECODE(cust_credhold_s, 'Y', 'good credit',
'N', 'on hold', 'undefined')
FROM customer;

cust_name_s
---------------------------- ---------------
WILE SEAL CORP.              'good credit'
MAGNETICS USA INC.           'on hold'
MAGNETOMETRIC DEVICES INC.   'on hold'
FAIR PARK GARDENS            'on hold'
```

在普通英语中，查询中的 DECODE 语句表示：如果列 CUST_CREDHOLD_S 的值为 Y，那么在输出结果中使用 good credit 字符串来代替，如果值为 N，则在它的位置输出 on hold。如果不是 Y 也不是 N，就用 undefined 来代替。函数各部分（包括表达式、搜索条件、匹配值和默认值）的最大数为 255。

下面的例子在 Microsoft SQL Server2008、IBM DB2 9.5 和 Oracle 11g 中产生相同的结果：

```
SELECT
    cust_name_s,
    CASE cust_credhold_s
        WHEN 'Y' THEN 'good credit'
        WHEN 'N' THEN 'on hold'
        ELSE 'undefined'
    END
FROM customer

cust_name_s
---------------------------- ---------------
WILE SEAL CORP.               'good credit'
MAGNETICS USA INC.            'on hold'
MAGNETOMETRIC DEVICES INC.    'on hold'
FAIR PARK GARDENS             'on hold'
```

可以看到，查询变得更加清晰，在三大 RDBMS 实现中更加方便。

Note 注意：Oracle 在 10g 版本中引入了 CASE 表达式，以前的版本只使用 DECODE 函数，只要向后兼容，就仍然支持 DECODE 函数。

www.TopSage.com

Tip 提示：在DB2 9.5中使用CASE表达式时，必须使用ELSE子句。如果没有操作需要执行，而且找不到匹配的情况，就需要添加ELSE NULL。

可以通过多种方法来使用CASE语句，这里给出一些有关CASE基本使用方法的例子。例如，CASE语句不仅可以在SELECT查询中使用，还可以在UPDATE中使用。在ACME数据库中，列CUST_STATUS_S值接收Y和N值。下面的查询例子将这两个值取反，用N代替Y，反之亦然：

```
UPDATE CUSTOMER
    SET cust_status_s = CASE cust_status_s
        WHEN 'Y' then 'N'
        ELSE 'Y'
    END
(37 row(s) affected)
```

该函数还可以在WHERE子句和嵌套查询中使用。参考供应商文档查看这些功能的详细讨论。

10.7.2 COALESCE和NULLIF

这两个函数是Oracle的DECODE函数和CASE表达式的特例，专门处理NULL值。三大数据库中都包含这两个函数。

Note 注意：IBM DB2 9.5中有一个VALUE函数，与COALESCE函数的使用方法完全相同。Oracle的函数VALUE对NULL不采取任何操作，属于面向对象的功能域。

函数NULLIF对两个表达式进行比较。如果相同，则返回NULL，否则返回第一个表达式。例如，在表CUSTOMER中，不仅包含CUST_CREDHOLD_S的信息，而且包含CUST_STATUS_S列，说明客户账户是否为活动账户。如果CUST_CREDHOLD_S列和CUST_STATUS_S列包含相同的值，那么返回NULL，否则返回CUST_CREDHOLD_S列的值。这样的技术可用于发现公司规则的差异。如果客户的信用是on hold(N)，那么它的状态也应该是N。如果发现NULL值，这一矛盾必须通过管理员手动解决：

```
SELECT
    NULLIF(cust_credhold_s, cust_status_s) compare_stat
FROM customer

compare_stat
------------
NULL
N
N
N
```

三大RDBMS实现中的NULLIF函数的语法是相同的。

COALESCE函数进一步应用了NULLIF函数的原理，它返回列表中第一个不为NULL的参数。例如，已提供了很多客户付款方式——信用账户、直接付款和邮寄支票，如果打算为客户寄送账单，但忘记他到底使用哪种支付方法，可以使用下面的查询：

```
SELECT
    customer_id,
    COALESCE(visa_account, direct_deposit, check_account)
FROM account_management
```

ACCOUNT_MANAGEMENT表不在ACME样本数据库中，但是这个例子简要介绍了COALESCE函数的使用方法。在三大RDBMS实现中，该函数的语法是相同的。

10.7.3 NVL、NVL2和ISNULL

这些函数用于在检索值中查找NULL值，并用非NULL值来代替这些NULL值。Oracle中的NVL函数与SQL Server中的ISNULL函数是等价的。在DB2 9.5中没有NULL值检测函数，但是可以使用COALESCE函数得到相似的结果。

下面是Oracle中的一个NVL函数示例：

```
SELECT
    cust_name_s,
    NVL(cust_alias_s, 'undefined')
FROM customer;

cust_name_s                alias_status
-----------------          -----------------
WILE SEAL CORP.            MNGA71396
```


www.TopSage.com

TopSage.com

```
MAGNETICS USA INC.          MNGA71398
MAGNETOMETRIC DEVICES INC.  MNGA71400
FAIR PARK GARDENS           undefined
INTEREX USA                 undefined
```

该查询返回客户名称和相应别名的列表。如果客户的别名没有录入（因此包含 NULL 值），结果列表中就会包含 undefined。如果使用 SQL Server 语法，查询则如下：

```
SELECT
    cust_name_s,
    ISNULL(cust_alias_s, 'undefined') alias_status
FROM customer

cust_name_s                 alias_status
--------------------------- --------------
WILE SEAL CORP.             MNGA71396
MAGNETICS USA INC.          MNGA71398
MAGNETOMETRIC DEVICES INC.  MNGA71400
FAIR PARK GARDENS           undefined
INTEREX USA                 undefined
```

前面已经提到过，在 DB2 9.5 中可以使用 COALESCE 函数，效果相同：

```
SELECT
    cust_name_s,
    COALESCE(cust_alias_s, 'undefined') alias_status
FROM customer

cust_name_s                 alias_status
--------------------------- --------------
WILE SEAL CORP.             MNGA71396
MAGNETICS USA INC.          MNGA71398
MAGNETOMETRIC DEVICES INC.  MNGA71400
FAIR PARK GARDENS           undefined
INTEREX USA                 undefined
```

在 DB2 9.5 和 SQL Server 中查询的结果与前面 Oracle 11g 显示的结果是相同的。

Oracle 11g 有一个经过演化的函数 NVL2，与常规的 NVL 函数不同，前者根据表达式的值是否为 NULL，允许使用多个值替代。例如，如果想查看哪些客户被赋予了别名，哪些没有，可以使用下面的查询：

```
SELECT
    cust_name_s,
    NVL2(cust_alias_s, 'alias assigned', 'not assigned' )
   alias_status
FROM customer;

cust_name_s                  alias_status
---------------------        -----------------
WILE SEAL CORP.              alias assigned
MAGNETICS USA INC.           alias assigned
MAGNETOMETRIC DEVICES INC.   alias assigned
FAIR PARK GARDENS            not assigned
INTEREX USA                  not assigned
```

在上面的查询中，如果客户别名列包含 NULL，则在最终结果集中显示 not assigned，否则显示 alias assigned。

第二个和第三个参数可以是除 LONG 之外的任意数据类型，第一个参数可以是任意数据类型。

Note

注意：在本章中，三大 RDBMS 实现中有很多函数都没有讨论。包括 Oracle 11g 的游标、分析函数和对象引用函数；IBM DB2 9.5 中的表函数和表达式；还有 Microsoft SQL Server2008 中的系统函数、安全函数、文本和图像函数以及行集函数等。这些函数表示供应商特定扩展插件，并不常用，而且需要对 RDBMS 结构有更深层次的理解。附录 G 中介绍了这些函数中的大部分。有些函数（比如安全相关函数）会在相关章节中详细讨论。

转换过程中易犯的错误

对于每个进行值替换的函数，指定的数据符合数据兼容类型十分重要：替代值必须与列值类型匹配。例如，如果用数字 0 来代替 undefined，Microsoft SQL Server 和 IBM DB2 9.5 数据库将生成一条错误，显示无法将 VARCHAR 转换为 INTEGER。

当使用连接运算符时（添加表达式时不能做到），Oracle 会隐式地将 NUMBER 转换为 VARCHAR2。下面的查询说明了 Oracle 的特点：

www.TopSage.com

```
SELECT
 4||'5'.concat_value,
 '4'+ 5 sum_value
FROM dual;

concat_value        sum_value
----------------    -----------
45                   9
```

基于运算符的函数，Oracle 隐式地将常量 4（一个用单引号定义的字符）转换为 NUMBER 数据类型，以此得到 SUM_VALUES（+运算符）值；对于 CONCAT_VALUE（‖运算符）值的计算，数字 5 被转换为字符串。

自动转换使运算更加简便，但是会导致代码不完善。因为每种 RDBMS 的操作各不相同，而且不能保证在未来的版本中会得到支持，所以不能依赖自动转换。

10.8 用户定义函数

用户定义函数扩展了 RDBMS 产品提供的内置功能，是 SQL:2008 标准和 SQL/PSM 标准（当模块中包含函数时）的一部分，将在第 16 章中讨论。

尽管不是每个供应商都提供了添加自定义函数功能，但是“三大”RDBMS 供应商为实现代码重用（同一个函数可以被许多用户或程序调用），在各自的 RDBMS 中都提供了该功能。

用户定义函数一般通过过程化语言实现。包括在 Oracle 中使用 Java、PL/SQL 和 C#（一种.NET 语言）；在 Microsoft SQL Server 2008 中使用 Transact-SQL、C（用于扩展存储过程）和.NET 语言（C#、VB.NET 等）；在 IBM DB2 9.5 中使用 Java 和 C#。实现的语法和细节在这些 RDBMS 中有很大的不同。同时需要注意，由于 Java 和 C#语言都有附加处理层，使用时会有性能冲突。

除了在 RDBMS 中创建和维护函数外，三大供应商提供了脚本和可执行功能来调用由操作系统维护的外部对象。调用外部程序的机制各不相同，但是原理是相同的——提供与外部操作系统通信的能力，并且不受 RDBMS 框架限制。这样的程序可以用 OS 支持的任何语言编写。因为外部程序可以访问系统中的各种资源，所以它是一个功能强大的工具。但是这种功能是以潜在的安全、稳定性和性能风险为代价的，所以需要谨慎使用。

Microsoft SQL Server 从 2000 版开始，还可以通过特殊的系统存储过程集在存储过程的 SQL 代码中调用和执行 OLE（ActiveX）对象。根据这一特征，可以使用 Microsoft 程序（或能够实现 OLE 接口的任意其他程序）在一个存储过程中访问 Excel 或 Outlook 功能。Microsoft 已经开始降低 OLE 功能的重要性，它的标准逐渐被新兴的.NET 技术替代。Microsoft SQL Server 2008 中已经内置了.NET 技术。

10.9 小结

SQL 内置函数弥补了非过程化语言固有不足，它们能完成很多任务：从对数取整到字符串操作、数据类型转换和替换函数的复杂处理逻辑，如 DECODE。

尽管 SQL92/99 标准定义的函数个数相对较少（不过在 SQL:2003 中扩展了），但是每个 RDBMS 供应商增加了自己的常用工具集。因此，各供应商的函数在功能、实现细节或语法方面存在差异，或通过是否包含在实现而表现出差异就不足为奇了。

本章全面概括了一些最重要的函数，以及能够产生相似结果的三大 RDBMS 实现之间的相应语法。本章函数的分类基于函数中执行运算（如字符数据、数值数据或日期）的参数的数据类型和一般功能（如在两个不同数据类型之间转换）。对于一些具有高级或非标准功能的函数，在与这些函数主题相关的章节中讨论：第 12 章讨论安全函数，第 15 章讨论 XML 函数。

www.TopSage.com

第 11 章
SQL 运算符

CHAPTER

本章提要：

- 算术运算符和字符串串联运算符
- 逻辑运算符
- 运算符优先级
- 赋值运算符
- 比较运算符
- 位运算符
- 小结

SQL 中的**运算符**定义为符号和关键字，用于指定对一个或多个被称为**操作数**或**参数**的表达式执行操作。

运算符一般包括两种类型：

- **一元运算符**：一次只对一个操作数进行运算，标准格式是<operator><operand>。
- **二元运算符**：一次对两个操作数进行运算。通常显示的格式为<operand><operator><operand>。

11.1 算术运算符和字符串串联运算符

顾名思义，**算术运算符**是用来进行算术运算的。算术运算符的使用相当直观（假设没有在小学留级的话），而且，几乎可以用 SQL 语句的每条子句中。表 11-1 中给出了算术运算符的完整列表。

表 11-1　　算术运算符

运算符	说明
+	加：将两个数相加，在 Microsoft SQL Server 中则用于串联字符串。除此之外，其用法在三个数据库中是相同的。只有 Microsoft SQL Server 对该运算符进行了重载，用于串联和加法。该运算符在“三大”数据库和 SQL:2003 语法中都是有效的
-	减：从一个数中减去另一个数。其用法在三个数据库中是相同的，并且与 SQL:2003 语法兼容。它还可以作为恒等号或一元负运算符使用
*	乘：将一个数与另外一个数相乘。其用法在三个数据库中是相同的，并且与 SQL:2003 语法兼容
/	除：一个数除以另一个数。其用法在三个数据库中是相同的，并且与 SQL:2003 语法兼容
%	模：计算除法的整数余数。这是一个 Microsoft SQL Server 专用运算符。该运算符的功能在 Oracle 和 DB2 中通过 MOD 函数实现。SQL:2003 中没有定义该运算符

字符串串联运算符是串联两个字符串的二元运算符。并且在某种程度上与两个数相加的加号运算符类似。字符串串联运算符如表 11-2 所示。

表 11-2　　字符串串联运算符

运算符	说明
||	串联运算符：串联字符串，在 Oracle 和 DB2 中有效，由 SQL:2003 标准指定
+	Microsoft SQL Server 中的字符串串联运算符
CONCAT	DB2 专用的字符串串联运算符

www.TopSage.com

大家网 TopSage.com

尽管在 SQL 中进行算术运算比较简单，但必须注意运算中使用的数据类型，对数值来说，指结果的精度和标量；对日期来说，则指结果值的范围等。

如果数据类型与运算符不兼容（例如，字符串值用于加运算符），某些数据库（如 Oracle）会进行隐式转换（只要可能）；而其他的数据库（DB2 和 SQL Server）则要求显式地转换为兼容的数据类型才能执行运算。

下面是几个算术运算符用法的例子。Oracle 使用下面的查询将两个数相加：

```
SELECT 5 + 5  total_value
FROM   dual;

total_value
-----------
         10
```

结果 TOTAL_VALUE 是数值数据类型。如果不使用加运算符，而使用串联运算符，那么结果将大不相同：

```
SELECT 5 || 5  total_value
FROM   dual;

total_value
-----------
55
```

这里，Oracle 隐式地把数字转换成字符，TOTAL_VALUE 是这个字符数据类型的串联结果。DB2 也可以识别串联运算符，但不会执行隐式转换；如果在 DB2 中执行这个例子将生成一个错误，要求显式地将数字转换为字符。

Microsoft SQL Server 采用不同的工作方式：它重载加运算符来实现字符串串联，并根据操作数的数据类型决定将操作数相加还是串联：SQL Server 对数字相加，对字符串串联。下面的例子说明了该功能。这个查询用两个整数作为操作数，SQL Server 计算它们的和（也是一个整数）：

```
SELECT 5 + 5  total_value

total_value
-----------
         10
```

下面的查询使用两个字符（由用于文字值的单引号表示），结果是串联（字符数据）：

```
SELECT '5' + '5'  total_value

total_value
-----------
55
```

该运算符也用于日期运算。尽管有很多函数用于添加和减少日期，但也可以用算术运算符实现同样的功能。下面的例子是在 Microsoft SQL Server 2008 中，在一个日期上加 10 天（日期为常量，已转换成日期时间数据类型以确保正确的运算）：

```
SELECT CAST('09/10/2007 12:00 AM' AS DATETIME) + 10
         AS result_date

result_date
-------------------------
2007-09-20 00:00:00.000
```

日期算法可能非常混乱。Oracle 和 DB2 允许在日期操作中广泛应用算术，而 Microsoft SQL Server 明确要求用户使用日期和时间相关函数。

下面是 Oracle 中的日期算术例子。

第一个查询是在指定的日期上添加特定天数。日期可能来自表或 RDBMS 请求：

```
SELECT SYSDATE,
       (SYSDATE) + 10 result_date
FROM   dual;

SYSDATE     result_date
----------  ------------
9/17/2007   9/27/2007
```

要加 2 小时，可以使用下面的运算：

```
SELECT SYSDATE,
       (SYSDATE) + 2/24 result_date
FROM   dual;
```


www.TopSage.com

```
SYSDATE                result_date
---------------------- ----------------------
9/22/2007 11:04:05 AM  9/22/2007 1:04:05 PM
```

通过加 24/24（等于 1），实际上是加了 1 天；同样地，2/24 就是 2 个小时。

对于分和秒也采用同样的方法：

```
SELECT SYSDATE ,
       (SYSDATE) + 1/(24*60) result_date
FROM   dual;

SYSDATE                result_date
---------------------- ----------------------
9/22/2007 11:08:26 AM  9/22/2007 11:09:26 AM
```

注意：用圆括号把 24*60 括起来非常重要。本章后面将会提到，结果取决于表达式中运算符的优先级；括号确保用 1 除以 24，再乘以 60（分钟数）。

当然，只要操作数是兼容的数据类型，其他运算符也可以使用该操作。例如，不能对日期进行乘或除，只能加和减。对于一般的数值数据类型，任何算术运算符都是有效的。

例如，在 ACME 数据库的 V_CUSTOMER_TOTALS 视图中计算每一笔交易所征的销售税额（比如说 8.5%）：

```
SELECT order_number,
       total_price,
       total_price * 0.085 tax
FROM v_customer_totals

order_number         total_price         tax
----------------     -----------    ---------
523720                    7511.00    638.43500
523721                    8390.00    713.15000
523722                    6608.00    561.68000
```

模运算符（%）计算除法的整数余数。它是 Microsoft SQL Server 特定运算符，在 Oracle 和 IBM 中使用 MOD 函数。下面的查询是在 Microsoft SQL Server 中计算整数 5 除以 3 的模：

```
SELECT 5%3 remainder

remainder
---------
        2
```

这与 Oracle 和 DB2 中的函数 MOD 是完全相同的：

```
SELECT MOD(5,3) remainder
FROM   sysibm.sysdummy1

remainder
---------
        2
```

警告：Microsoft SQL Server 和 Oracle 允许 NULL 值参与算术运算（例如，SELECT SYSDATE+NULL FROM DUAL，结果为 NULL）。对任何运算符，只要给定一个 NULL 操作数，不管其他的操作数是什么，结果总为 NULL，理解这一点很重要。Oracle 的串联运算符不遵循这个规则（例如，'STRING'||NULL='STRING'），而 DB2 不允许 NULL 参与任何运算。

11.2 逻辑运算符

逻辑运算符用来对一组条件求值，返回结果通常为 TRUE，FALSE 或“unknown”。

注意：从 Oracle 9i RDBMS 版本开始，Oracle 就把逻辑运算符作为 **SQL 条件**，前面的版本称为**比较运算符**或**逻辑运算符**。DB2 使用**谓词**来代替**运算符**（它与 SQL:2003 标准一致，但笔者认为这完全是误导）。

表 11-3 中列出了 SQL 支持的所有逻辑运算符。

表 11-3 SQL 逻辑运算符

运算符	操作
ALL	如果一组比较全部为 TRUE，则结果为 TRUE

www.TopSage.com

续表

运算符	操作
AND	如果两个布尔表达式均为 TRUE，则结果为 TRUE
ANY	如果一组比较中任意一个为 TRUE，则结果为 TRUE
BETWEEN	如果操作数在某个范围内，则结果为 TRUE
EXISTS	如果子查询包含任意行，则结果为 TRUE
IN	如果操作数与表达式列表中的某个相等，则结果为 TRUE
LIKE	如果操作数与某个模式匹配，则结果为 TRUE
NOT	对布尔运算符的值取反
OR	如果任意一个布尔表达式是 TRUE，则结果为 TRUE
SOME	如果一组比较中的某些值为 TRUE，则结果为 TRUE

11.2.1 ALL

ALL 用于将一个标量值与一个单列值集进行比较。它与比较运算符结合使用，有时被归为比较运算符。当对指定条件的所有比较组合结果都为 TRUE 时，返回 TRUE；否则，返回 FALSE。

11.2.2 ANY | SOME

将一个标量值与一个单列值集进行比较。关键字 ANY 和 SOME 可以完全互换。如果指定条件对任意一组比较都有效，该运算符返回 TRUE；否则，返回 FALSE。在第 8 章的“在 WHERE 子句中使用子查询”一节中给出了有关 ALL 和 ANY—SOME 的用法例子。

Note 注意：在 Microsoft SQL Server 和 DB2 中，运算符 ANY—SOME 只能与子查询结合使用，Oracle 允许它们与标量值列表结合使用。

11.2.3 BETWEEN<expression>AND<expression>

BETWEEN 运算符允许选择条件“近似”匹配。如果表达式的结果大于等于开始表达式的值，且小于等于结尾表达式的值，则返回 TRUE。如果与非运算符 NOT 一起使用，只有表达式的值小于开始表达式的值，或者大于结尾表达式的值时，才返回 TRUE。

Note 注意：与 BETWEEN 运算符结合使用的 AND 关键字，与本章后面介绍的 AND 运算符是不一样的。

下面的查询从 PRODUCT 表中检索产品价格在 15 到 23.10 美元之间的产品 ID、产品描述和产品价格：

```
SELECT prod_id_n,
       prod_description_s description,
       prod_price_n price
FROM   product
WHERE  prod_price_n BETWEEN 15 AND 23.10

prod_id_n description                price
--------- -------------------------  -----
      990 SPRUCE LUMBER 30X40X50     18.24
     3045 STOOL CAPS 9"              15.92
     4055 GAZEBOS 40X30X60           16.03
     4761 BAR RAILS 24X48X128        23.10
     4964 BASES 30X45X60             23.10
5786 CRATING MATERIAL 12X48X72       17.90

(6 row(s) affected)
```

注意，最终结果集中包括边界值。该运算符的工作方式在三大数据库中完全一样，并可以用于多种数据类型：日期、数字和字符串等。

尽管对字符串求值的规则是相同的，但产生的结果不像使用数字那么直观。字符串的值取决于其中的字符值，并且，除非指定整个字符串，否则不包括边界限制值。例如，如果希望获得其描述是以“C”和“S”之间的字符开头的产品信息，可以使用下面的查询：

```
SELECT prod_id_n,
       prod_description_s_description,
       prod_price_n price
FROM   product
```


www.TopSage.com

TopSage.com

```
WHERE  prod_description_s BETWEEN 'C' AND 'S'

prod_id_n    description                  price
---------    ---------------------------- -----
     4000 HAND RAILS ROUNDS 48X48X12      11.80
     4055 GAZEBOS 40X30X60                16.03
     5786 CRATING MATERIAL 12X48X72       17.90

(3 row(s) affected)
```

注意，尽管 SPRUCE LUMBER30×40×50 描述是以 S 开头，但并没有包括在内；这个分界线标准 S 意味着只有单一 S 构成的描述才满足这个条件；在这种情况下，使用通配符（请参阅本章后面部分）一点用都没有。为包括使用前面描述的产品，必须使用完整的值，或前两个字符是连续的（参见前面的查询），以这样的方式查询：

```
SELECT prod_id_n,
       prod_description_s description,
       prod_price_n price
FROM   product
WHERE  prod_description_s BETWEEN 'C' AND 'ST'

prod_id_n    prod_description_s              prod_price_n
---------    ------------------------------  ------------
      990 SPRUCE LUMBER 30X40X50                     18.24
     4000 HAND RAILS ROUNDS 48X48X12                 11.80
     4055 GAZEBOS 40X30X60                           16.03
     5786 CRATING MATERIAL 12X48X72                  17.90

(4 row(s) affected)
```

为确保所有以 S 开头的字符串都包含在内，条件应改为 WHERE prod_description_s BETWEEN 'C'and 'T'。另一种方法是使用第 10 章讨论的字符串函数，从产品描述字段中提取并比较相应的子串。

11.2.4 IN

该运算符将任意给定的值与列表中的值进行比较，列表中的值可以是常量，也可以是子查询返回的值。下面的查询说明 IN 运算符的概念：

```
SELECT prod_id_n,
       prod_description_s description,
       prod_price_n price
FROM   product
WHERE  prod_price_n IN (10,15,18.24,16.03)

prod_id_n  description                price
---------  -------------------------  -----
      990 SPRUCE LUMBER 30X40X50      18.24
     4055 GAZEBOS 40X30X60            16.03

(2 row(s) affected)
```

因为没有价格正好等于 10 或 15 美元的产品，所以只返回两条匹配的记录。

注意：根据列表进行计算的表达式的数据类型必须与列表值的数据类型一致。一些 RDBMS 可以隐式地在兼容的数据类型之间进行转换（例如，Microsoft SQL Server 2008 和 Oracle 11g 都接受 10.15、'18.24'、16.03 这样的数字与字符串混合的列表，而 DB2 会生成一条错误 SQL0415N，SQLSTATE 42825）。

OR 运算符可以完成（在某种程度上）运算符 IN 的操作。下面的查询将返回与文字列表查询相同的结果集：

```
SELECT prod_id_n,
       prod_description_s description,
       prod_price_n price
FROM   product
WHERE  prod_price_n = 10
OR     prod_price_n = 15
OR     prod_price_n = 18.24
OR     prod_price_n = 16.03

prod_id_n description                price
--------- -------------------------  -----
     990 SPRUCE LUMBER 30X40X50      18.24
     4055 GAZEBOS 40X30X60           16.03
```

```
(2 row(s) affected)
```

可以从子查询中动态生成 IN 列表中的值。下面的查询检索所有销售数量少于 90 件的产品描述：

```
SELECT prod_description_s
FROM   product
WHERE  prod_id_n IN
    (SELECT ordline_prodid_ fn
     FROM   order_line
     WHERE  ordline_ordqty_n < 90)

prod_description_s
------------------------------------------
HAND RAILS ROUNDS 48X48X12
BAR RAILS 24X48X128
BAR RAILS 30X45X60
BASES 30X45X60
CRATING MATERIAL 12X48X72

(5 row(s) affected)
```

NOT 运算符与 IN 结合使用，返回所有不属于特定值列表的记录——预定义的或由子查询生成的。

11.2.5 EXISTS

EXISTS 运算符检测子查询中是否存在包含匹配值的行。子查询可以查询同一个表，也可以查询不同的表，或是两类表的组合（请参阅第 8 章，了解相关查询的信息）。该运算符在三大 RDBMS 实现中的工作方法是相同的。

EXISTS 与 IN 运算符的用法相似（通常用于相关查询）。下面的 SQL 查询结果与前面例子中的查询结果相同：

```
SELECT prod_description_s
FROM   product
WHERE  EXISTS(SELECT *
              FROM  order_line
              WHERE order_line.ordline_prodid_fn = product.prod_id_n
              AND   ordline_ordqty_n < 90)

prod_description_s
------------------------------------------
HAND RAILS ROUNDS 48X48X12
BAR RAILS 24X48X128
BAR RAILS 30X45X60
BASES 30X45X60
CRATING MATERIAL 12X48X72

(5 row(s) affected)
```

必须使用附加条件（ordline_prodid_fn=product.prod_id_n）来限制 PRODUCT 表输出记录在表 ORDER_LINE 中有对应记录，也就是说，只包括那些有订单的产品。如果没有这个条件限制，该查询将返回 PRODUCTS 表中的所有产品。此时在列名前加表名前缀是可选的，只是用于更好地说明相关查询的使用。

Note 注意：EXISTS 运算符对非空值列表求值结果为 TRUE。例如，下面的查询将返回 PRODUCT 表中的所有记录，因为子查询求值结果总为 TRUE：

```
SELECT prod_description_s
FROM   product
WHERE  EXISTS (SELECT * FROM DUAL)
```

运算符 NOT 与 EXISTS 结合使用，将在子查询结果集为空时引入记录。

11.2.6 LIKE

LIKE 运算符属于“模糊逻辑”范围。当 SELECT 查询的 WHERE 子句条件只有部分已知时，就要使用 LIKE 运算符。它使用多种通配符来确定值中缺失的部分（参见表 11-4）。查询中必须包含 LIKE 关键字。

下面的查询从 ACME 数据库的 CUSTOMER 表中检索客户名（CUST_NAME_S 字段）以 WILE 开头的客户信息：

```
SELECT cust_id_n,
       cust_name_s
FROM   customer
WHERE  cust_name_s LIKE 'WILE%'

cust_id_n    cust_name_s
```


www.TopSage.com

```
--------- --------------------------------
      152 WILE BESS COMPANY
       55 WILE ELECTROMATIC INC.
       63 WILE ELECTROMUSICAL INC.
        7 WILE ELECTRONICS INC.
        1 WILE SEAL CORP.

(5 row(s) affected)
```

表 11-4　　与运算符 LIKE 结合使用的通配符

字符	说明	实现
%	匹配包含 0 个或多个字符的任意字符串	Oracle、DB2、Microsoft SQL Server
_（下划线）	匹配字符串中的任意单个字符	Oracle、DB2、Microsoft SQL Server
[]	匹配指定范围内或字符集中的任意单个字符	仅 Microsoft SQL
[^]	匹配不在指定范围内或字符集中的任意单个字符	仅 Microsoft SQL

注意，空格也被认为是搜索的目的字符。

例如，如果需要修改搜索条件来查找名字以 WILE 开头，且第二部分发音类似 EAL（“MEAL”？“SEAL”？）的公司，可以使用下面的查询：

```
SELECT cust_id_n,
       cust_name_s
FROM   customer
WHERE  cust_name_s LIKE 'WILE%_EAL%'
cust_id_n     cust_name_s
---------     ----------------------------
        1     WILE SEAL CORP.

(1 row(s) affected)
```

简单地说，该查询可以解释为“CUSTOMER 表中的 CUST_NAME_S 字段包含以下字符序列的所有记录：值以 WILE 开头，后跟数量不确定的字符，然后是空格，第二部分以某个字母或数字开始，后跟 EAL 组合；其他字符不确定。”

在 Microsoft SQL Server（以及 Sybase）中，还可以使用匹配模式来指定一定范围的字符。下面的查询检索名称第二部分以 S 或 B 开头的客户记录：

```
SELECT cust_id_n,
       cust_name_s
FROM   customer
WHERE  cust_name_s LIKE 'WILE% [S,B]%'

cust_id_n     cust_name_s
---------     ------------------------------------
      152     WILE BESS COMPANY
        1     WILE SEAL CORP.

(2 row(s) affected)
```

ESCAPE 子句与 LIKE 运算符结合使用，允许在搜索字符串中包括通配符本身。允许指定一个转义字符，并用它来识别搜索字符串中应被视为“常规”字符的特殊字符。查询中的任意字符都可以被指定为转义字符，但必须小心以确保不使用值内可能包含的字符（例如，%或 L 用作转义字符将导致错误的结果）。该子句受三大数据库支持，是 SQL 标准的一部分。

下面的例子在 INFORMATION_SCHEMA 视图（Microsoft SQL Server 2008 特定视图，用于访问当前数据库中有关对象的信息）中查询，下划线符号（_）是搜索字符之一：

```
SELECT table_name,
       table_type
FROM   INFORMATION_SCHEMA.TABLES
WHERE  table_name LIKE 'ORD%/_L%' ESCAPE '/'

table_name       table_type
---------------  ----------
ORDER_LINE       BASE TABLE

(1 row(s) affected)
```

该查询从视图中请求那些表名以 ORD 开头，后跟不确定数量的字符，并包含一个下划线“_”，后跟“L”，并以不确定数量的字符结尾的记录。因为下划线字符有作为通配符这一特殊意义，因此，必

www.TopSage.com

须在前面加一个转义字符“/”。可以看到，只有表名ORDER_LINE符合这些要求。

经过一些练习之后，可以构造非常复杂的模式匹配查询。下面是一个例子：在查询中，指定名称的第一部分是以E开头的三个字符，后跟数量不确定的字符，然后是一个空格，第二部分是某个字符开头，然后是ES，最后是数量不确定的字符：

```
SELECT cust_id_n,
       cust_name_s
FROM   customer
WHERE  cust_name_s LIKE '_E%_ES%'

cust_id_n     cust_name_s
---------     ----------------------------------------
       89     INTEGRATED POWER DESIGNS
      152     WILE BESS COMPANY

(2 row(s) affected)
```

这个结果乍一看可能会让人感到惊讶：为什么结果集中包含“INTEGRATED POWER DESIGNS”？要理解这个结果，应该回顾一下%的含义，它可以代表任意字符，包括空格；因此，GRATED POWER字符串符合这个规则；它后跟一个空格，然后是一个包含ES的单词作为第二个和第三个字符。

正则表达式和Oracle REGEXP_函数

正则表达式可以被定义为是一项用字符序列描述复杂模式的技术。它在代码编写简化和节约开发人员时间方面是有用的，被广泛用于多种程序设计语言。然而，与Microsoft SQL Server不同，Oracle中的LIKE运算符不能与正则表达式结合使用。Oracle有很多功能强大的函数，这些函数与IEEE POSIX完全兼容，并扩展了与SQL:2003正则表达式相关的标准。

REGEXP_LIKE与LIKE运算符类似，但实现了用正则表达式匹配来代替简单的模式匹配。例如，在Microsoft SQL Server示例中，检索那些第二部分客户名以S或B开头的客户记录，可以使用下面的查询：

```
SELECT cust_id_n,
       cust_name_s
FROM   customer
WHERE  REGEXP_LIKE (cust_name_s, '(^WILE[ ][S|B])');

CUST_ID_N   CUST_NAME_S
----------  --------------------------------------------------
         1  WILE SEAL CORP.
       152  WILE BESS COMPANY
```

该模式可以解释为：客户名以WILE（^WILE）开头，后跟空格字符（[]），后面是S或B（[S|B]）。如果需要返回任何包含S或B的字符串，且S或B可以位于WILE子串后面的任何位置，不一定在空格之后，结果会有所不同：

```
SELECT cust_id_n,
       cust_name_s
FROM   customer
WHERE  REGEXP_LIKE (cust_name_s, '(^WILE.+[S|B])');

CUST_ID_N   CUST_NAME_S
----------  --------------------------------------------------
         1 WILE SEAL CORP.
         7 WILE ELECTRONICS INC.
        63 WILE ELECTROMUSICAL INC.
       152 WILE BESS COMPANY
```

可以指定不区分大小写模式，在REGEXP_函数中添加i作为最后一个参数：REGEXP_LIKE(cust_name_s,'(^WILE.+[S|B])',i)。

REGEXP_COUNT返回源串中模式匹配的数量。下例在包含子字符串ND的产品描述中返回字符N后跟字符D的匹配次数：

```
SELECT prod_description_s,
       REGEXP_COUNT(prod_description_s, '([N][D])', 1) PATTERN_COUNT
FROM   product
WHERE  REGEXP_LIKE (prod_description_s, '([N][D])');

PROD_DESCRIPTION_S                          PATTERN_COUNT
------------------------------------------- -------------
HAND RAILS ROUNDS48X48X12 2
```


www.TopSage.com

REGEXP_SUBSTR、REGEXP_INSTR 和 REGEXP_REPLACE 与第 10 章中讨论的 SUBSTR、INSTR 和 REPLACE 函数类似。

请参考 Oracle 文档，了解在数据库正则表达式中使用 POSIX 元字符的详细说明。

Note 注意：要搜索发音相似的名称，可以使用第 10 章中讨论的 SOUNDEX 函数。

11.2.7 AND

AND 组合两个布尔表达式，当两个表达式都为 TRUE 时返回 TRUE。下面的查询返回单价超过 20 美元而且产品描述以 S 开头的产品记录：

```
SELECT prod_id_n,
       prod_description_s description,
       prod_price_n price
FROM   product
WHERE  prod_price_n > 20
AND    prod_description_s LIKE 'S%'

prod_id_n     description              price
---------     ------------------------ -----
     1880     STEEL NAILS 6"           33.28
     2871     STOOL CAPS 5"            26.82

(2 row(s) affected)
```

只选择了那些同时满足这两个条件的记录。

当语句中使用多个逻辑运算符时，AND 运算符优先。可以使用圆括号改变计算顺序。

11.2.8 NOT

该运算符对一个布尔输入取反。可以对本章中迄今为止讨论过的其他运算符取反。下面是一个使用 NOT 运算符的简单例子：

```
SELECT prod_id_n,
       prod_description_s description,
       prod_price_n price
FROM   product
WHERE  prod_price_n NOT IN (10,15,18.24,16.03)

prod_id_n     description                  price
---------     ---------------------------- -----
     1880     STEEL NAILS 6"               33.28
     2871     STOOL CAPS 5"                26.82
     3045     STOOL CAPS 9"                15.92
     4000     HAND RAILS ROUNDS 48X48X12   11.80
     4761     BAR RAILS 24X48X128          23.10
     4906     BAR RAILS 30X45X60           27.00
     4964     BASES 30X45X60               23.10
     5786     CRATING MATERIAL 12X48X72    17.90

(8 row(s) affected)
```

该查询返回价格与列表中任何一项都不匹配的产品信息。

11.2.9 OR

OR 运算符按照布尔逻辑的规则组合两个条件（请参阅附录 K 了解有关布尔逻辑的更多信息）。当语句中有多个逻辑运算符时，OR 运算符优先级低于 AND 运算符。但可以通过圆括号来改变计算顺序。在本章前面讨论 IN 运算符时已经给出了 OR 运算符的用法实例。

11.3 运算符优先级

优先级表示同一个表达式中运算符的计算顺序。当几个运算符一起使用时，具有更高优先级的运算符会在较低优先级的运算符之前计算。一般来说，运算符的优先级与高中数学里的运算符优先级遵循同样的规则，有些可能会与常规稍有不同。优先级的顺序如表 11-5 所示。

www.TopSage.com

表 11-5 运算符优先级

运算符	优先级
一元运算符，按位取 NOT（仅 Microsoft SQL Server）	1
乘法和除法	2
加法、减法和串联	3
SQL 条件	4

求值优先级可以极大影响查询结果。以下 Oracle 查询的目的是计算 TOTAL_PRICE+4*0.085 的值：

```
SELECT total_price,
       total_price + 4 * 0.085 tax
FROM   v_customer_totals

total_price        tax
-----------    -------
    7538.20    7538.54
    8420.10    8420.44
    6630.40    6630.74
```

结果取决于打算如何计算。一种是将所有价格增加 4 美元，然后计算结果的 8.5%；另外一种是考虑运算符的优先级规则，那么价格增加 0.34 美分。当然，RDBMS 将遵循优先级的规则，首先用 0.085 乘以 4，运算结果与 TOTAL_PRICE 列的值相加。

您可能会奇怪为什么减（-）和加（+）运算符既属于第一优先级，同时又属于第三优先级。这只是运算符重载的另一个例子：除了进行减和加运算外，它们也表示一元运算符**负**和**符号标识**（例如-5 或+5）。例如：

```
SELECT
    -2 * 3 + 5 result
FROM  dual;
result
------
    -1
```

表达式的结果为-1 而不是预期的-11，因为乘法先执行。如果使用括号保证减号最后计算，那么结果会不同：

```
SELECT
    -(2 * 3 + 5) result
FROM  dual;

result
------
   -11
```

一元运算符+不会影响操作数，而-则会对操作数取负，如下面的例子所示（在所有三个 RDBMS 中都可使用）：

```
SELECT -(+total_price) minus_first,
       +(-total_price) plus_first
FROM   v_customer_totals

minus_first   plus_first
-----------   ----------
-7538.20       -7538.20
-8420.10       -8420.10
. . .          . . .
-6630.40       -6630.40
. . .          . . .
-12138.60      -12138.60
```

可以看到，一元运算符的顺序并没有影响结果；在这两种情况下，结果都为负值。

运算符优先级的重要性

运算符的优先级问题并非如表面看起来的那么简单。例如，一个 Windows 操作系统每个副本绑定的常见计算机程序，如 Microsoft 计算器（或任何现代计算器），表面看起来该程序在标准计算类型和科学计算类型之间是有区别的，例如，如果从视图菜单中选择标准模式，表达式 5+4*3 求值结果为 27，但是，如果从菜单中选择科学模式，同样的表达式答案则为 17，无论如何，常规计算器会认为 27 是正确答案。

www.TopSage.com

由于历史原因，也就是内存的缺乏，第一个计算器只有两个寄存器（用来存储输入的数字），因此，只能存放两个数。接下来，必须执行操作以释放空间来存储下面的数字。在上面的例子中，指令按照下面的次序执行：把 5 放入一个寄存器，把 4 放入第二个寄存器，执行运算（+），把结果保存到其中一个寄存器中，把数字 3 放入第二个寄存器，执行运算（*），依此类推。

每一个运算符按钮按下后，便开始计算结果。随着硬件价格逐渐降低，可以使用两个以上的寄存器来实现在数学中确立几百年的优先级规则。

到那时，就会有大量的计算器以“简单”的方式执行数学计算了。为了不冒消费者反对的风险，供应商们决定使用两种类型的计算器——一种针对“普通”人群，另一种针对“科学领域”人群。当然，这种方法也被移植到软件计算器中。

Tip 提示：可以使用圆括号改变运算（不是运算符）的优先级。

如果使用圆括号，前面例子的表达式求值结果将不同：

```
SELECT total_price,
       (total_price + 4) * 0.085 value1,
       total_price + 4 * 0.085 value2
FROM   v_customer_totals;

total_price   value1       value2
-----------   ---------    -------
    7538.20   641.08700    7538.54
    8420.10   716.04850    8420.44
    6630.40   563.92400    6630.74
```

11.4 赋值运算符

赋值运算符是使用最方便的运算符之一，用于给变量赋值。使用该运算符时产生混淆的唯一原因是重载。所有 RDBMS 都重载该运算符，该预算符具有附加功能——SQL 中的比较功能。

在下面的 SQL 查询中，等号运算符(=)用于赋值，通过在现有价格基础上增加 2%来更新 PRODUCT 表中的价格（PROD_PRICE_N）列：

```
UPDATE product
SET    prod_price_n = prod_price_n * 1.02

(10 row(s) affected)
```

同样的运算符可用在 SQL 语句的 WHERE 子句中，用于值的比较：

```
UPDATE product
SET    prod_price_n = prod_price_n * 1.02
WHERE  prod_id_n = 1880

(1 row(s) affected)
```

该语句对 ID 为 1880 的产品指定 2%的增量。在同一查询中，等号运算符（=）同时有赋值和比较的功能。

Note 注意：在一些 SQL 过程化语言中，赋值和比较运算符是有区别的。Oracle PL/SQL 使用：=赋值，=用作比较；Microsoft SQL Server 的 Transact–SQL 和 DB2 的 SQL PL 只使用一种运算符=来完成两种功能。

11.5 比较运算符

比较运算符用于比较两个或多个值。虽然在任何有效的 SQL 表达式中都允许使用，但通常用在 SELECT 语句的 WHERE 子句中。

除了非标准运算符（!<）和（!>）之外，其他比较运算符在三大数据库中的用法是一样的。DB2 9.5 和 Microsoft SQL Server 2008 可以识别（!<）和（!>），但 Oracle 11g 并不识别这两种运算符。非标准的不等于运算符!=可以在所有三种数据库中使用，但另一个不等于运算符^=只能被 Oracle 和 DB2 识别。

www.TopSage.com

表 11-6 中列出了所有比较运算符。

表 11-6 比较运算符

运算符	说明	运算符	说明
=	等于	<>	不等于
>	大于	! =	不等于*
<	小于	^=	不等于**
>=	大于等于	! <	不小于*
<=	小于等于	! >	不大于*

*不遵循 SQL 标准

**不遵循 SQL 标准，在 Microsoft SQL Server 2008 中无效

我们通过查询 ACME 数据库的 V_CUSTOMER_TOTALS 视图来说明表 11-6 中运算符的用法。

SQL 使用 WHERE 子句限制查询返回的行数。比较运算符在逻辑上满足这一策略，基于某种搜索条件来限制行数。考虑下面这个查询：它返回所有行（这里只显示其中的 5 行）：

```
SELECT customer_name,
       order_number,
       total_price
FROM   v_customer_totals

customer_name        order_number    total_price
-----------------    ------------    -----------
WILE BESS COMPANY    523720               7538.20
WILE BESS COMPANY    523723              11186.00
WILE BESS COMPANY    523724               8745.50
WILE ELECTROMUSICAL INC. 523727           6630.40
...
WILE ELECTROMUSICAL INC. 523728           6630.40
```

例如，如果只想看到 WILE BESS COMPANY 的记录，可以使用下面的查询：

```
SELECT customer_name,
       order_number,
       total_price
FROM   v_customer_totals
WHERE  customer_name = 'WILE BESS COMPANY'

customer_name        order_number     total_price
-----------------    ------------     -----------
WILE BESS COMPANY     523720               7538.20
WILE BESS COMPANY     523723              11186.00
WILE BESS COMPANY     523724               8745.50
```

该查询遍历视图中的所有客户，将 CUSTOMER_NAME 列中的值与 WHERE 子句中提供的值进行比较，只选择匹配 WILE BESS COMPANY 的记录，排除所有其他记录。

比较字符串、日期和时间

比较字符串和日期对一名经验不足的 SQL 程序员来说是一件可怕的事。关键是屏幕上见到的查询结果不一定与源表中存储的内容相同。例如，'WILE BESS COMPANY'和'WILE BESS COMPANY'（末尾有一个多余的空格）在屏幕上是不能区别的，但是表达式'WILE BESS COMPANY'='WILE BESS COMPANY'会返回 FALSE，因为两个字符串不相等。

另一个不错的例子是日期和时间字段的比较。思考下面的 Oracle 语句，用 DUAL（在前一章讨论过）选取当前的日期：

```
SELECT SYSDATE FROM DUAL;

SYSDATE
---------
24-SEP-07
```

可以假设语句（SYSDATE='24-SEP-07'）的值为 TRUE，但是下面的查询返回 0 行：

```
SELECT * FROM DUAL WHERE SYSDATE = '24-SEP-07';
no rows selected
```

原因是 SYSDATE 的实际值由 DATE 和 TIME 两部分组成，但是默认只显示第一部分。为显示更多内容，可以结合使用转换函数 TO_CHAR 和掩码。例如：

```
SELECT TO_CHAR(SYSDATE, 'DD-MON-YY HH:MI:SS') FROM DUAL;
```


www.TopSage.com

```
TO_CHAR(SYSDATE,'D
------------------
24-SEP-07 03:51:25
```

为使上例能够运行，可以使用 TRUNC 函数（请参阅第 10 章获得更多内容）提取 SYSDATE 的日期部分：

```
SELECT * FROM DUAL WHERE TRUNC(SYSDATE) = '24-SEP-07';

D
-
X
```

可以看到，查询运行的时间与最初预期的时间相同。

要找出那些 TOTAL_PRICE 超过某一预设值（比如说$7 500）的记录，可以使用大于运算符（>）：

```
SELECT customer_name,
       order_number,
       total_price
FROM   v_customer_totals
WHERE  total_price > 7500

customer_name       order_number    total_price
------------------  ------------    -----------
WILE BESS COMPANY   523720              7538.20
WILE BESS COMPANY   523723             11186.00
WILE BESS COMPANy   523724              8745.50
```

可以看到，这有效地把所有总价格小于$7 500 的记录排除在外。如果想包括取舍点值，可以使用大于等于（>=）或小于等于（<=）运算符：

```
SELECT customer_name,
       order_number,
       total_price
FROM   v_customer_totals
WHERE  total_price <= 7538.20

customer_name              order_number    total_price
-------------------------- ------------    ------------
WILE BESS COMPANY          523720              7538.20
WILE ELECTROMUSICAL INC.   523727              6630.40
WILE ELECTROMUSICAL INC.   523728              6630.40
```

该查询返回 V_CUSTOMER_TOTALS 视图中小于等于$7 538.20 的记录。

要查询客户中除一位之外，其他客户的订单，可以使用不等于运算符<>或!=（后者是一个非标准的运算符，但三大供应商都支持它）：

```
SELECT customer_name,
       order_number,
       total_price
FROM   v_customer_totals
WHERE  customer_name <> 'WILE BESS COMPANY'

customer_name              order_number     total_price
-------------------------  -------------    -----------
WILE ELECTROMUSICAL INC.   523727               6630.40
WILE ELECTROMUSICAL INC.   523728               6630.40
```

前面已经提到过，不仅 WHERE 子句中可以使用比较运算符，第 10 章中介绍的 CASE 表达式同样可以：

```
SELECT order_number,
    total_price,
     CASE
        WHEN total_price < 7500 THEN 'medium order'
        WHEN total_price > 7500
         AND total_price < 10000 THEN 'big order'
        WHEN total_price > 10000 THEN 'very big order'
        ELSE 'cannot say'
END
FROM v_customer_totals

order_number   total_price
------------   -----------   ---------------
523720            7538.20    big order
523723           11186.00    very big order
523724            8745.50    big order
523727            6630.40    medium order
523728            6630.40    medium order.
```


www.TopSage.com

大家网 TopSage.com

11.6 位运算符

位运算符对整数数据类型执行位运算。表 11-7 中列出了 SQL 中引入的所有位运算符。要理解位运算的结果，必须理解布尔代数的基本原理。

表 11-7 位运算符

运算符	说明	运算符	说明
&	按位 AND	^	按位取反 OR
|	按位 OR	~	按位 NOT

Note 注意：只有 Microsoft SQL Server 提供位运算符；DB2 中的 SQL 专用语言本身不支持位运算，Oracle 11g 有 BITAND 函数，它与 SQL Server 的按位 AND 的工作原理相同。

Cross-Reference 交叉参考：参阅附录 K 了解有关布尔代数的更多信息。

位运算符的操作数可以是整数数据类型，也可以是二进制字符串数据类型（IMAGE 数据类型除外）。数据类型兼容性如表 11-8 所示。对于 SQL 等高级语言来说，位运算符不太常用，因此实例不多。

表 11-8 位操作数的数据类型兼容性

左操作数	右操作数
BINARY	INT、SMALLINT、TINYINT
BIT	INT、SMALLINT、TINYINT、BIT
INT	INT、SMALLINT、TINYINT、BINARY、VARBINARY
SMALLINT	INT、SMALLINT、TINYINT、BINARY、VARBINARY
TINYINT	INT、SMALLINT、TINYINT、BINARY、VARBINARY
VARBINARY	INT、SMALLINT、TINYINT

一个可能的应用是为颜色准备复杂的位掩码，现在所有 RDBMS 都不只支持文本和数值数据。例如，考虑一个 0 和 1 的组合——0101。当对该数字与另外一个二进制数字（例如，1101）应用 AND 时，将产生下面的结果：

```
SELECT 5 & 13 result

result
------
     5
```

5 的二进制表示为 0101。在本例中是逻辑 AND，只有 1 和 1 才能产生 1，任何其他的组合，即 1 和 0 或 0 和 0 都产生 0。

```
0101        (十进制 5)
1101        (十进制 13)
----
0101        (十进制 5)
```

要对数据库中存储的图像像素进行解码和编码，可以使用位掩码和移位。也可以使用 XOR（异或）运算符，基于某些数字密钥来加密数据。

应用布尔逻辑保证数据安全性

对数据库而言，数据安全是极为重要的。软件开始运行，数据安全问题就随之产生了。安全涉及不同的方面（将在第 12 章中进行讨论），包括未授权访问、恶意数据操作等。下面简单介绍数据损坏问题。

有时，技术上的一个小故障就会引起数据损坏——损坏原因存在于硬盘中，硬盘存储了大量 0 和 1，这些 0 和 1 表示数据。或者两个位发生了翻转、位置对调，或全部消失，这是由于功率波动、宇宙射线、地震等因素引起的，问题是如何将数据恢复到一致状态。

解决方法之一是使用 RAID （Redundant Array of Independent Disks，独立磁盘冗余阵列）。尽管 RAID 不直接用于 SQL，但它与 RDBMS 和布尔代数有重要关系。

www.TopSage.com

RAID 的原理是把几个相对廉价的磁盘驱动器组合成一个阵列，阵列作为一个独立的单元工作，从而增加了操作的可靠性和速度。有几种不同类型的 RAID 体系结构（磁盘组合的方式），从 RAID0 到 RAID5（第一个是 RAID0，是单一磁盘的另一个名称）命名。

将 RAID 与布尔逻辑关联起来的基本技术原理是带区。RAID 把阵列中的每一个驱动器划分成带区，这些带区按一定的次序交错。

为使其形象化，想象有一摞薄脆饼，在第一、三、五块薄脆饼涂上黄油。在第二和第四块薄脆饼上涂上果酱。依照该方法，RAID 能够同时对多块磁盘（驱动器）的数据执行不同的操作。在 RAID3，RAID4 和 RAID5 的配置中，能够获得奇偶校验带来安全的这一额外好处。一个驱动器专门用来存储奇偶信息：如果一个驱动器出现故障，磁盘上所有存储的信息可以通过对其余驱动器上的位信息进行异或（XOR）运算而恢复。

思考前面的例子，不执行逻辑 AND，而是执行 XOR 运算。XOR 运算遵循布尔逻辑的规则：0XOR0=0，0XOR1=1，1XOR1=0，1XOR0=1。假设第一个数字位于磁盘 1 上，第二个数字位于磁盘 2 上，进行 XOR 运算的结果（奇偶校验）位于磁盘 3 上。现在，磁盘 2（包含值 1101）出现了故障，通过对磁盘 1 上的数据和磁盘 3 上的数据执行 XOR 运算，可以很容易地恢复毁坏的数据：

```
0101    0101
1101    1000
----    ----
1000    1101
```

当然，如果多个驱动器出现故障，安全系统将无法运行。赌注是计划风险，多个驱动器同时发生故障的可能性非常小。

以上介绍的内容都是很基本的知识，通常超出了 SQL 程序设计的范围。如果需要更深入地了解布尔代数，可以找一本专门论述这个主题的书，而不是只涉及一般 SQL 问题的书。

11.7 小结

尽管 SQL 中没有强调，但运算符通过操作输出结果、指定选择条件和搜索条件发挥了重要作用。

运算符可以是一元或二元的，这取决于是对一个还是两个操作数进行运算。

只要 RDBMS 存在，有关如何看待运算符的争论就会一直持续下去。一些数据库供应商坚决主张将 LIKE、IN、EXISTS、BETWEEN 等关键字归类为运算符，而另一些则主张将其列为谓词。这是理论上的争论，但分类不会影响在 SQL 程序设计中使用运算符或谓词的方式。

通常，运算符在所有数据库供应商中都是一致的。但是，要注意 Microsoft SQL Server 中的位运算符和非标准的比较运算符是例外。

运算符优先级是 RDBMS 对多个运算符的表达式求值时确定的顺序。优先级的顺序非常重要。利用圆括号可以自定义表达式的优先级（与默认优先级次序相反）。

www.TopSage.com

www.TopSage.com

大学网
TopSage.com

PART 5

使用系统目录实现安全

本部分包括：

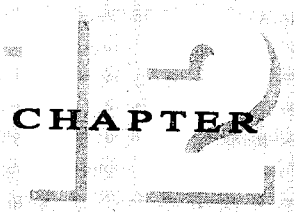
第12章
SQL 和 RDBMS 的安全

第13章
系统目录和 INFORMATION_SCHEMA

www.TopSage.com

www.TopSage.com

第 12 章
SQL 和 RDBMS 的安全



安全已经成为遍及世界的商业活动面对的一个巨大问题，它要求公司保护他们的数据不受来自外部和内部的威胁。2002 年的 Saarbanes-Oxley 法案强制这些公司建立严格的 IT 操作标准；这在数据库供应商中造成冲击，从而实现新的安全功能，包括报表、审计和审核数据访问。作为一种语言，SQL 只提供了有限的安全机制，它依赖实际的 RDBMS 软件来实现一种更健壮的安全框架。基于这种认识，我们决定对 RDBMS 安全进行全面概述，并详细讨论特定于 SQL 的安全语句（既有 SQL 标准强制要求的，也有特定于供应商的实现）。

本章提要：

- 基本安全机制
- 定义数据库用户
- 使用特权来管理安全
- 使用角色来管理安全
- 将视图用于安全
- 将约束用于安全
- 将存储过程和触发器用于安全
- 数据加密
- 数据库审核
- 安全标准

12.1 基本安全机制

数据库安全是一个大的主题，而探索用于指导数据库供应商实现其不同安全机制的方式甚至更大。安全不是与关系数据库一起出现的；密码身份验证、锁、审核和其他安全概念同人类历史一样古老，而 SQL 只不过是添加了一个新花样。按照 SQL:2003 的指导，所有数据库基本上都遵守建立安全过程。所有 RDBMS 都有三种共有的安全级别：

- **身份验证**：用户连接到 RDBMS 以执行特定操作。
- **授权**：基于分配给用户的特权集，用户可以访问数据库或数据库架构对象来执行特定操作。
- **审核**：监视可疑（及其他方面）的活动并执行事后分析。

不同之处是每个关系数据库管理系统（RDBMS）实现这些级别的方式不同。

12.1.1 标识和身份验证

第一道防线是身份验证。在访问 RDBMS 之前，必须提交足够的信息，这些信息由 RDBMS 自身或安装了这个数据库的操作系统来验证。身份被验证后，就可以继续访问数据库资源、对象和数据。

12.1.2 授权和访问控制

用户通过身份验证并被授予访问数据库权限后，RDBMS 会针对特定的数据库对象使用一个复杂的细粒度特权（权限）系统。这些特权包括创建、访问、修改、删除或执行相关数据库对象的权限，以及添加、修改和删除数据。

www.TopSage.com

大家网 TopSage.com

12.1.3 加密

加密提供了另一个安全层次，保护数据免遭未授权查看。即使获得对数据库的访问权，不借助特定的软件客户端或密码，要想把加密数据破解为可读形式也不是件容易的事情。

12.1.4 完整性和一致性

尽管安全主要基于身份验证和授权过程，但数据完整性在保护数据免遭无意或恶意操作方面起了一定的作用。例如，即使一个用户获得了对数据库的访问权（例如，通过窃取密码），仍然必须遵循数据操作的相关规则，其中包括不允许孤记录；如果不了解数据库关系，则用户不能从一个父表中删除记录（不过有些供应商已经实现了CASCADE功能，它可以指导RDBMS在删除父表时删除子记录），不能把重复记录插入到一个受UNIQUE约束保护的列中，不能插入违反CHECK约束的无效数据等。

12.1.5 审核

审核提供了监视数据库活动的方法，包括有效的和未授权的活动。它保留数据库访问尝试的跟踪记录（成功的和失败的）——数据删除和插入（以备要知道发生了什么事情）等。对安全认证而言，审核是一个必要的过程，本章后面部分将对此进行讨论。

12.2 定义数据库用户

用户的概念是各种RDBMS实现中最容易引起混淆的概念之一（从外行人的直观角度来看更是如此）。SQL:2003没有指定任何特殊的语法（甚至一种方式）来在数据库中创建用户。而是留给数据库供应商们自行设计，供应商们设法创建了一些独特的解决方案。例如，Oracle 11g几乎不区分用户和数据库架构，IBM DB2只使用操作系统定义的用户（或由某些外部框架定义的用户），而Microsoft SQL Server结合了两种方式：使用Windows账户和特殊的系统过程将用户添加到数据库。

Note 注意：用户可以是一个RDBMS用户、一个操作系统用户、一个角色或应用程序角色，均定义在数据库级别。考虑到细粒度安全性，在数据库中，一个对象属于一个架构，由用户所有。这样也可以处理通过授予和/或撤消访问特权来控制访问数据库对象的问题。

根据定义，数据库用户是使用RDBMS服务器所提供的服务的人。它可以是应用程序、数据库管理员或者只是在任意给定时刻正好访问数据库的某个人。正如前文所述，用户身份验证是保证安全的第一道防线。

在Oracle 11g中，创建用户的语法如下：

```
CREATE USER <user_name>
IDENTIFIED [BY <password> | EXTERNALLY | GLOBALLY]
[DEFAULT TABLESPACE <def_tablespace_name>]
[TEMPORARY TABLESPACE <tmp_tablespace_name>]
[QUOTA (<integer> (K | M)|UNLIMITED ON <tablespace_name>],...
[PROFILE <profile_name>]
[PASSWORD EXPIRE]
[ACCOUNT [LOCK | UNLOCK]];
```

要创建用户，除了CREATE USER特权(在本章后面说明)，至少还需要指定用户名和密码，因为大多数语句的子句都是可选的。要创建一个由操作系统（OS）或一些第三方控制过程识别的用户，可以使用关键字EXTERNALLY。GLOBALLY子句能够创建全局用户。这样的用户必须由企业目录服务（如Oracle Internet目录）授权。这样便于单点登录（SSO），使用户在一次验证后能访问企业范围的资源。有关OS集成的更多信息，请参阅本章后面的侧栏“OS安全集成”。

Note 注意：外部SSL（Secure Socket Layer，安全套接层）用户会获得一个证书。证书指用户的PKI证书并且在OS外部进行管理。

其他选项是特定于Oracle的，在其他RDBMS实现中没有直接的对应项。例如，通过指定PASSWORD

www.TopSage.com

EXPIRE 子句，实际上告诉 Oracle 这个用户的密码立即就会到期，强制用户在下一次连接到数据库时选择一个不同的密码。ACCOUNT LOCK 允许指定在多少次不成功尝试（避免密码猜测）之后将阻止访问，或者当一位员工离开公司时禁用（但不删除）其用户帐户（Sarbances-Oxley 的一项要求）。这是一种安全手段，避免所有“非活动”用户登录到数据库。有关所有特定于 Oracle 选项的详细讨论属于高级 Oracle 管理范畴，超出了本书的范围。

Microsoft SQL Server 2008 创建一个用户的语法稍有不同：

```
CREATE USER <user_name>
[ [FOR | FROM ]
[ LOGIN <login_name>
| CERTIFICATE <certificate_name>
| ASYMMETRIC KEY <key_name>
] | WITHOUT LOGIN
    ]
 [WITH DEFAULT_SCHEMA = <schema_name>];
```

这些语法选项的含义是不言而喻的。LOGIN 子句指定为用户分配哪一个登录名，CERTIFICATE 和 ASSYMETRIC KEY 分别为要创建的数据库用户指定证书和非对称密钥。与 Oracle 不同，Microsoft SQL Server 2008 选择将诸如证书和非对称密钥之类的安全组件作为 RDBMS 的一部分。

Note 注意：对称和非对称健都可以用于加密。对称密钥是指使用相同的密钥进行加密和解密的算法。非对称密钥的加密/解密循环需要两个不同的密钥（公有和私有）。因此，非对称密钥可以提供更好（更安全）的加密，但代价是复杂和潜在的性能影响。

在前一版本的 SQL Server 中利用证书为客户端提供 SSL（安全套接层）连接。需要手动建立这个连接。从 SQL Server 2005 开始，SQL Server 本身就提供了所有必要的服务。每个 SQL Server 实例具有自己的服务主密钥（Service Master Key），使用操作系统级的数据保护 API 进行加密。此服务用于保证数据库连接的安全，以及提供数据库主密钥的安全。使用下面的语句（需要 CONTROL 权限）创建数据库主密钥：

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD = <password>
```

创建主密钥之后，可以使用它来创建证书和非对称密钥：

```
CREATE CERTIFICATE <certificate_name>
[AUTHORIZATION <user_name>]
FROM
       ASSEMBLY <assembly_name> |
       [EXECUTABLE] FILE = <file_path>
       [WITH PRIVATE KEY
           (
             FILE = <private_key_path>
              [,DECRYPTION BY PASSWORD = <password>]
              [,ENCRYPTION BY PASSWORD = <password>]
           )
           ]
        |
         [,ENCRYPTION BY PASSWORD = <password>]
        WITH SUBJECT = <certificate_subject_name>
         [START_DATE = 'mm/dd/yyy'
| EXPIRY_DATE = 'mm/dd/yyyy']
 [ACTIVE FOR BEGIN_DIALOG = ON|OFF ]
```

可以看到，创建证书的完整语法非常复杂，并且需要理解商业领域和安全问题。（请参考供应商文档了解更多信息。）

创建非对称密钥的语法遵循类似的规则：

```
CREATE CERTIFICATE <certificate_name>
   [AUTHORIZATION <database_principal_name>]
   FROM FILE=<strong_file_name_path>
          |[EXECUTABLE] FILE = <file_path>
          | ASSEMBLY <assembly_name>
        | WITH ALGORITHM = (RSA_512|RSA_1024|RSA_2048)
[ENCRYPTION BY PASSWORD = <password>]
[ACTIVE FOR BEGIN_DIALOG = ON|OFF ]
```

除了识别 USER，还可以使用证书和非对称密钥来加密/解密数据。请参阅本章稍后介绍的相关函数用法。

前面提到，IBM DB2 9.5 不会将 USER 视为数据库托管实体，而是依赖外部操作系统，将 OS 定义的用户映射到数据库授权 ID。请参考 IBM 文档了解更多信息。

www.TopSage.com

Oracle 11g用户PROFILE

PROFILE子句指定了一个环境用户配置文件，通过设置用户不能超越的限制来控制用户的资源访问。如果没有指定配置文件，则使用默认配置文件。要创建一个配置文件，必须拥有CREATE PROFILE特权——直接或者通过角色（本章后面进行说明）。配置文件定义了这样一些重要特性：用户可以维护的并发会话数、CPU处理时间片段、空闲时间会话超时、密码策略（到期、宽限期、复用、复杂规则）、帐户锁定策略等。

使用CREATE PROFILE语句创建配置文件，其语法比较简单：

```
CREATE PROFILE <profile name>
LIMIT [<resource_parameters> | <password_parameters> ];
```

资源参数可以设置成整数、UNLIMITED或DEFAULT。整数明确指定天数、尝试次数等。UNLIMITED则取消这些限制；而DEFAULT告诉Oracle使用其默认值，这些默认值在系统中设置为默认配置文件（分配给任何没有明确指定配置文件的用户）的一部分。

这里使用的大多数参数都要求深入了解Oracle知识。其中很多都与安全直接相关。例如，资源参数SESSIONS_PER_USER限制这个用户ID允许的并发会话数；当超过这个数时，拒绝登录。另一个资源参数CONNECT_TIME设置连接时间限制。参数IDLE_TIME确切告诉Oracle允许一个特定的连接保持多久的空闲，一旦达到这个限制，则终止这个连接。当然，还有许多资源参数可用于把系统资源分配给用户，在Oracle的文档中可以找到相关内容。

实际上每个密码参数都属于安全领域。像FAILED_LOGIN_ATTEMPTS、PASSWORD_LIFE_TIME、PASSWORD_REUSE_TIME、PASSWORD_REUSE_MAX、PASSWORD_LOCK_TIME和RASSWORD_GRACE_TIME这些参数都定义如何创建、使用密码和让密码到期。参数名称说明了它们的功能。第一个参数FAIED_LOGIN_ATTEMPTS防止猜测密码，一旦达到允许次数，则锁定账户。PASSWORD_REUSE_MAX参数指定重用当前密码前更改密码的次数等。其中，参数（PRASSWORD_VERIFY_FUNCTION）允许使用自定义逻辑，通常作为PL/SQL脚本实现，进行额外的密码确认、登录等。

以下是一个非常简单的CREATE USER语句：

```
CREATE USER    new_user
IDENTIFIED BY it_is_me;
User created.
```

通过省略所有可选子句，这个语句创建了一个分配有所有系统默认值的用户：使用系统**表空间**，密码永远不会到期等。但这种情况不会出现在现实产品环境中。除了安全不足外，从维护角度看也是很低效的。

创建用户时，也会创建一个与此用户对应的架构。新创建的用户在RDBMS中没有权限或特权，但必须被授予CONNECT特权以便能够连接到Oracle数据库，并且“按需”分配特权集（本章后面讨论特权和GRANT语句的语法）。在授予这些特权之后，用户ACME可以利用密码ACME访问数据库ACME，并基于授予的特权执行操作。

注意：创建用户是创建“真实架构”的唯一途径。CREATE SCHEMA语句用于创建多个表和视图，并在单个事务内自己的架构中进行多项授权。有关创建数据库对象的更多信息，请参阅第4章。

当然，与大多数Oracle数据库对象一样，稍后可以删除或修改USER。这里是DROP语法：

```
DROP USER <user name> [CASCADE];
```

CASCADE指定在删除用户前删除用户架构中的所有对象。如果不指定这个子句，Oracle不允许删除架构中包含任何对象的用户。

ALTER USER语句为用户指定多种更改。它分配一个不同的配置文件，更改资源限制，改变默认的**表空间**等。它也提供了一个修改用户特性的便利途径，且无需删除用户后再从头重新创建用户。有关ALTER USER语句的完整语法，请参考Oracle文档。

在Oracle数据库中，可以通过系统目录访问用户的相关信息，在第13章中将对此进行详细讨论。

IBM DB2 9.5使用一种外部安全服务和内部访问控制相结合的方式。RDBMS安全的第一步是身份验证。IBM DB2没有提供身份验证服务，而是依赖可以在操作系统内部实现的外部服务，或作为某个

www.TopSage.com

第三方软件产品（具体指数据库作为分布式环境的一部分安装时）实现。

这意味着除非拥有操作系统帐户，可以是 Windows、Unix 或任何其他 OS，否则不能创建用户。同样，在 DB2 数据库外部对所有用户进行身份验证管理。更多信息，请参考本章后面的 OS 安全集成部分或 IBM DB2 文档。

Microsoft SQL Server 2008 可以通过 Windows 操作系统或混合模式身份验证配置用户身份验证。

在第一种情况下，用户必须属于某个预定义的 Windows 账户，允许使用他的系统登录名登录到 SQL Server 数据库，系统登录名必须映射到内部 SQL Server 账户。因此，OS 登录必须映射到这个 OS 登录允许的每个内部数据库。

第二种情况(混合模式)要求用户做到以下两点:(1)使用有效的 Windows 账户来建立与 SQL Server 的连接，(2) 提供一个用户 ID/密码供 SQL Server 进行身份验证。

Microsoft SQL Server 2005 引入了 CREATE USER 语句。与 ALTER USER 和 DROP USER 一样，它是标准的用户管理机制。此外，也有一种通过从 SQL Server 前一版本继承的系统存储过程进行用户管理的机制。请注意，尽管仍然支持用户管理过程，但是在 Microsoft SQL Server 未来的发行版本中可能不再使用用户管理过程。

表 12-1 列出了 SQL Server 中用户管理使用的存储过程。运行这些存储过程所需参数的含义基本上都是不言而喻的。如果刚刚接触 Microsoft SQL Server，请参考 SQL Serve 联机丛书（包含在每一个 RDBMS 安装包中），在使用这些过程时有一些应用限制。

表 12-1　　选择的 Microsoft SQL Serve 用户管理系统存储过程

存储过程	说明
sp_grantdbaccess<windows account name>[,<name inside database>]	把安全账户映射到 SQL Server 数据库中的内部账户，并将用户添加到当前数据库的用户列表中。这个 Windows 账户必须是有效账户(<Domain Name>\<User Name>)。当没有提供第二个参数时，使用 Windows 账户名。如果在应用程序内调用，那么成功时会返回 0，失败时会返回 1。只有 SQL Server 固定角色 SYSADMIN 中的成员才有执行这个过程的权限
sp_revokedbaccess<name inside database>	从当前数据库中删除账户映射
sp_adduser <windows account name>,[<name inside database>] [<group name>]	把用户（安全账户）添加到当前数据库中；Microsoft 提供这个过程只为了向后兼容，推荐使用 sp_grantdbaccess 来代替。如果从应用程序中调用，则成功时返回 0，失败时返回 1
sp_dropuser<name inside database>	在当前数据库中删除用户（安全账户）；Microsoft 提供这个过程只为了向后兼容，推荐使用 sp_revokedbaccess 来代替
sp_addlogin <login name>, [,<password>][,<default database>] [,<default language>][,<security identification number>] [,<encryption option>]	创建新的 SQL Server 登录，使用户能使用 SQL Server 身份验证连接到 SQL Server。通常用于通过网络（即用户没有本地账户）连接到 SQL Server。除登录名外的所有参数都是可选的。如果从应用程序中调用，则成功时返回 0，失败时返回 1
sp_grantlogin<windows account name>	把 SQL Server 访问特权授予一个有效的 Windows 账户。如果从应用程序中调用，则成功时返回 0，失败时返回 1
sp_revokelogin<windows account name>	删除 Windows 账户（或者用户或组）的访问特权。如果从应用程序中调用，则成功时返回 0，失败时返回 1。注意：这个过程撤消单个特权。如果撤消了登录特权的某个用户是具有这些特权的组成员，则这个用户仍能够连接
sp_denylogin <windows account name>	把登录添加到“拒绝列表”中，这个用户的登录将被拒绝。如果从应用程序中调用，则成功时返回 0，失败时返回 1。只有 sysadmin 和 securityadmin 固定 SQL Server 角色的成员可以执行这个过程
sp_droplogin <windows accout name >	从当前数据库中删除登录。如果登录映射到用户，则必须首先使用 sp_dropuser 存储过程删除这个用户；其他限制也会应用。如果从应用程序中调用，则成功时返回 0，失败时返回 1

Note　注意：系统存储过程和用户创建过程之间的主要区别在于它们的作用范围。系统存储过程是全局的，而用户创建的过程对于所在的数据库是局部的。事实上，全部系统存储过程都有前缀 sp_，并且都位于 Master 数据库中。

要执行上述任何存储过程，可以在 SQLCMD 或查询分析器中输入下面的语句：

www.TopSage.com

```
EXEC <stored procedures name> (arg1,arg2,...,argN)
```

有关Microsoft SQL Server用户的信息在第13章和系统存储过程部分详细讨论。

12.3 使用特权来管理安全

RDBMS本质上是一些对象的集合——架构、表、视图、过程以及管理这些对象的过程等。通过**特权**系统来限制访问这些对象是在SQL级别上实现的基本安全机制。

特权表示某特定用户访问、创建、操作和删除数据库中不同对象以及执行管理任务的权力。特权可以授予用户或角色，或者两者（在下一段讨论ROLE的概念）。

特权可以分成两大类——**系统特权**和**对象特权**，在不同的数据库开发商之间，这两类的差别很大。更详细的内容，请参考本章后面有关具体RDBMS实现的内容。

12.3.1 GRANT语句

SQL:2003标准把特权定义为授权用户在对象上和在其访问的系统中执行的操作类型。这些特权对本书中讨论的三个RDBMS都是有效的。

可以将特权授予一个单独的用户或角色。可以使用GRANT语句授予系统特权或对象特权。授予特权的基本语法在所有三个RDBMS包中相当一致（不过在高级选项中有一些明显的区别），在一个单独语句中可以授予多个特权，如下所示：

```
GRANT [ALL [PRIVILEGES]] | <privilege,...>
[ON <object_name>]
TO <user> | <group> | <role>
[WITH GRANT OPTION]
```

特权可以是表12-6中列出的任何一个；可选子句在不同数据库之间是不同的，在表12-2中列出。

表12-2 GRANT语句的选项

选项	说明	应用于
ADMIN OPTION	允许被授权者把角色授予另一个用户或角色（GLOBAL角色除外），以及从另一个用户或角色撤消这个角色，改变角色来更改访问权限或同时删除角色和权限	Oracle 11g
GRANT OPTION	允许被授权者把这个对象级特权授予其他用户或角色	Oracle 11g、Microsoft SQL Server 2008、IBM DB2 9.5、SQL92/99标准
HIERARCHY OPTION	WITH HIERARCHY OPTION（仅Oracle）表明授予的对象特权不只针对对象本身，也针对所有派生对象	Oracle 11g

1. 授予系统级特权

系统特权通常允许用户在给定的RDBMS中执行一些管理任务（创建数据库；创建和删除用户；创建、修改和删除数据库对象等）。需要在这个RDBMS系统中具有足够高的授权级别才能够执行或授予系统特权。区别这些系统特权与对象特权的特征是它们的作用域，有时也包括它们允许用户执行的活动类型。

系统特权是数据库专有的。每一个供应商都实现自己的系统特权集，而且有些系统特权对于不同的供应商可能具有不同的含义。有些系统（如Microsoft SQL Server）甚至没有定义系统特权，而使用所谓的安全对象特权（权限）。参见表12-6的Microsoft SQL Server权限的选择列表。

Oracle 11g拥有众多系统特权（以及绑定在一起的角色），表12-3中给出了其中最常见的一些系统特权。授予系统特权的SQL语句语法非常符合SQL标准。所有被授予的特权立即对被授权者生效（如果被授权者是一个ROLE，那么一旦启用，它将获得这些特权）。下面的代码给出了在Oracle 11g中授予系统特权的一个基本语法：

```
GRANT ALL [PRIVILEGES] |
      <system privilege,...> |
      <role>
TO <user> | <role> | <PUBLIC>
[IDENTIFIED BY <password>]
[WITH ADMIN OPTION];
```


www.TopSage.com

表 12-3 常用的 Oracle 11g 系统特权

系统特权	说明
ALTER DATABASE	允许被授权者改变 Oracle 数据库
ALTER SYSTEM	允许被授权者改变 Oracle 系统，允许执行系统更改语句
AUDIT SYSTEM	允许被授权者提交 AUDIT SQL 语句
CREATE [PUBLIC] DATABASE LINK	允许被授权者在被授权者的架构中创建私有/公共数据库链接
DROP [PUBLIC] DATABASE LINK	允许被授权者删除公共数据库链接
CREATE ANY INDEX	允许被授权者在任何架构中创建域 INDEX 或在任何架构的任何表中创建索引
ALTER ANY INDEX	允许被授权者改变任何架构中的任何 INDEX
DROP ANY INDEX	允许被授权者删除任何架构中的任何 INDEX
CREATE [ANY] OPERATOR	允许被授权者在自己的架构中（如果使用 ANY 子句，则在任何架构中）创建运算符及其绑定
DROP ANY OPERATOR	允许被授权者删除任何架构中的运算符
EXECUTE ANY OPERATOR	允许被授权者执行任何架构中的运算符
CREATE [ANY] PROCEDURE	允许被授权者在自己的架构中（如果使用 ANY 子句，则在任何架构中）创建独立或封装的过程（或函数）
DROP ANY PROCEDURE	允许被授权者删除任何架构中的独立或封装的过程（或函数）
EXECUTE ANY PROCEDURE	允许被授权者执行任何架构中的独立或封装的过程（或函数）
CREATE PROFILE	允许被授权者创建配置文件
ALTER PROFILE	允许被授权者改变现有配置文件
DROP PROFILE	允许被授权者删除现有配置文件
CREATE ROLE	允许被授权者创建角色
ALTER ANY ROLE	允许被授权者改变现有数据库中的任何角色
DROP ANY ROLE	允许被授权者删除数据库中现有的任何角色
GRANT ANY ROLE	允许被授权者授予数据库中现有的任何角色
CREATE [ANY] SEQUENCE	允许被授权者在自己的架构中（如果使用 ANY 子句，则在任何架构中）创建序列
ALTER ANY SEQUENCE	允许被授权者改变任何架构中的任何序列
DROP ANY SEQUENCE	允许被授权者删除任何架构中的任何序列
SELECT ANY SEQUENCE	允许被授权者访问任何架构中的任何序列
CREATE SESSION	允许被授权者连接到 Oracle 数据库
ALTER SESSION	允许被授权者提交 ALTER SESSION 语句
CREATE [ANY|PUBLIC] SYNONYM	允许被授权者在自己的架构中（如果使用 ANY 子句，则在任何架构中）创建私有或公共同义词
DROP [ANY|PUBLIC] SYNONYM	允许被授权者在自己的架构中（如果使用 ANY 子句，则在任何架构中）删除公共同义词
CREATE [ANY] TABLE	允许被授权者在自己的架构中（如果使用 ANY 子句，则在任何架构中）创建表
ALTER ANY TABLE	允许被授权者改变任何架构中的任何表
DELETE ANY TABLE	允许被授权者删除任何架构中的任何表或视图中的数据
DROP ANY TABLE	允许被授权者删除或截断任何架构中的任何表
INSERT ANY TABLE	允许被授权者把数据插入任何架构中的表和视图
LOCK ANY TABLE	允许被授权者锁定任何架构中的表和视图
SELECT ANY TABLE	允许被授权者从任何架构的任何表或视图中选择数据
UPDATE ANY TABLE	允许被授权者更新任何架构中表或视图中的数据
CREATE TABLESPACE	允许被授权者创建表空间
ALTER TABLESPACE	允许被授权者改变表空间
DROP TABLESPACE	允许被授权者删除表空间
CREATE [ANY] TRIGGER	允许被授权者在自己的架构中（如果使用 ANY 子句，则在任何架构中）创建数据库触发器
ALTER ANY TRIGGER	允许被授权者改变（启用、禁用或编译）任何架构中的任何触发器
DROP ANY TRIGGER	允许被授权者删除任何架构中的数据库触发器
CREATE [ANY] TYPE	允许被授权者在自己的架构中（如果使用 ANY 子句，则在任何架构中）创建对象类型和对象体

大家网 TopSage.com

续表

系统特权	说明
ALTER ANY TYPE	允许被授权者改变任何架构中的对象类型
DROP ANY TYPE	允许被授权者删除任何架构中的任何对象类型和对象体
EXECUTE ANY TYPE	允许被授权者使用任何架构中的任意用户定义对象类型或集合，并允许调用在这些对象类型中定义的方法
UNDER ANY TYPE	允许被授权者创建任何非最终对象类型的子类型
CREATE USER	允许被授权者创建用户，同时在任何表空间上分配配额，设置默认临时表空间，并分配一个 PROFILE
ALTER USER	允许被授权者改变任何用户，即改变用户的身份验证方法，在任何表空间上分配配额，设置默认的临时表空间，并分配一个 PROFILE 和一些默认角色
BECOME USER	允许被授权者成为另一个用户
DROP USER	允许被授权者删除其他用户
CREATE [ANY] VIEW	允许被授权者在自己的架构中（如果使用 ANY 子句，则在任何架构中）创建视图
DROP ANY VIEW	允许被授权者删除任何架构中的视图
UNDER ANY VIEW	允许被授权者为任何对象视图创建子视图
COMMENT ANY TABLE	允许被授权者在任何架构中的任何表、视图或列上添加注释
GRANT ANY PRIVILEGE	允许被授权者授予任何系统特权
SELECT ANY DICTIONARY	允许被授权者查询 Oracle SYS 架构中的任何数据字典对象

系统特权可以是表 12-3 中列出的任何一个（这里没有包括一些更复杂或更晦涩的系统特权）。既可以授予一个特权也可以授予一个角色（已经被授予某些特权）。ALL 关键字是指立刻授予所有特权，后面可能跟可选的 PRIVILEGES 关键字，它是为了兼容 SQL 标准而引入 Oracle 中的。

Tip　提示：通过在 Oracle 字典视图中查询 DBA_SYS_PRIVS 可以查看与用户关联的所有系统特权。可用于会话的特权显示在目录视图 SESSION_PRIVS 中。在第 13 章查看有关系统目录的更多内容。

特权或角色可以授予用户、角色（预定义的或创建的）或 PUBLIC（实际上意味着在 RDBMS 中定义的所有用户）。IDENTIFIED BY 子句为现有用户指定密码或者告诉 Oracle 创建这样一个用户（如果用户还不存在）。如果被授权者是一个角色，则这个子句无效，因为必须显式创建它。

SQL:2003 标准中指定了 WITH ADMIN OPTION，并且在 Oracle 11g 和 IBM DB2 9.5 中予以实现。本质上，它意味着允许用户或一个角色的成员把已分配的系统特权授予一些其他用户或角色（GLOBAL 角色除外），从另一个用户或角色撤消这个特权等。尽管在用法上有一些微妙的差异，但它的工作方式与用于对象级特权的 WITH GRANT OPTION 子句极为相似。参考开发商文档了解完整说明。

下面是一些基于 ACME 数据库的例子。要授予一个用户在数据库中创建表的特权，然后把这个特权传递给其他用户，可以使用下面的语句：

```
GRANT CREATE TABLE
TO new_user
IDENTIEIED BY it_is_me
WITH ADMIN OPTION;
Grant succeeded.
```

假设您具有足够的特权，那么将创建通过密码 IT_IS_ME 来识别的用户 NEW_USER。不过，如果用户 NEW_USER 没有被授予 CREATE SESSION 系统特权（需要使用这个特权访问数据库），则不能使用这个用户 ID 和密码连接到 Oracle 数据库，而会生成错误 ORA-01045: user NEW_USER lacks CREATE SESSION privilege;logon denied。

要解决这个问题，需要授予新创建的用户下面的特权：

```
GRANT CREATE SESSION
TO new_user
IDENTIFIED BY it_is_me
WITH ADMIN OPTION;
Grant succeeded.
```

现在可以使用 NEW_USER/IT_IS_ME 凭据连接到数据库，而且由于使用了 WITH ADMIN OPTION，还可以把这个特权授予其他用户。

www.TopSage.com

Oracle 中还有两个系统特权值得单独讨论：SYSDBA 和 SYSOPER，如表 12-4 所示。这些特权的行为很像角色，因为它们包含一些其他的系统特权。当连接到 Oracle 数据库时，可以指定连接 AS SYSDBA 或 AS SYSOPER，假设已经将这些特权授予这个用户。SYSDBA 是可以授予的最高特权之一。

表 12-4 Oracle 11g 的 SYSDBA 和 SYSOPER 系统特权

特权	说明
SYSDBA	允许被授权者执行 STARTUP 和 SHUTDOWN 操作；CREATE DATABASE、ALTER DATABASE（打开、装入、备份和更改默认字符集）；ARCHIVELOG、RECOVERY、CREATE SPFILE，包括 RESTRICTED SESSION 特权
SYSOPER	允许被授权者执行 STARTUP 和 SHUTDOWN 操作；ALTER DATABASE（仅打开、装入和备份）；ARCHIVELOG 和 RECOVERY；CREATE SPFILE，包括 RESTRICTED SESSION 特权
CONNECT、RESCOURCE、DBA	提供这些角色是为了兼容前面版本的 Oracle 数据库。可以通过查询 DBA_SYS_PRIVS 数据字典视图确定这些角色拥有的特权。Oracle 推荐针对数据库安全设计自己的角色而不是依赖于这些角色。未来版本的 Oracle 数据库可能不再自动提供这些角色

Note 注意：在某些平台上，对于 Oracle 11g，可以通过初始化参数 OS_USERS 把特权分配给数据库用户，OS_USERS 参数允许使用操作系统的程序授予角色。对于这样的用户，也不能使用 GRANT 语句来授予额外的角色，不过这对于其他用户和角色是可行的。

IBM DB2 类似于 Oracle。它具有系统特权，而且有些特权与授权级别（在本章稍后了解更多内容）关联。特权层次结构将 SYSADM 授权（系统管理员）置于顶端，下一级别包括 DBADM（数据库管理员）和 SYSCTRL（系统资源管理员），接下来是 SYSMAINT（系统维护管理员）。数据库用户处于层次结构的末端。SYSADM 的授权范围是整个系统，包括创建/删除对象的特权、提交数据库级别的命令，以及授予或撤消 CONTROL 和单独的特权。SYSCTRL 授权限制为数据库级别的系统资源，并且不允许直接访问数据。SYSMAINT 授权的限制甚至更多。它只能执行与系统（实例）关联的所有数据库上的维护任务，包括备份和还原，以及监视数据库操作。

Note 注意：命令 GET AUTHORIZATIONS 会报告当前用户的管理特权的状态（假设用户首先具有连接特权）。

表 12-5 中显示了 DB2 的一些系统级特权。

表 12-5 常用的 IBM DB2 9.5 授权

系统特权	说明
BINDADD	允许被授权者创建包；包的创建者自动拥有对象级 CONTROL 特权
CONNECT	允许被授权者访问 DB2 数据库
CREATETAB	允许被授权者在数据库中创建表（并在所有创建的对象上自动授予 CONTROL 对象级特权，而且即使撤消 CREATETAB 系统特权，这些 CONTROL 对象级特权也会继续保留）
CREATE_NOT_FENCED	授予用户注册在数据库管理器主进程中执行的函数的权限
CREATE_EXTERNAL_ROUTINE	指创建一个能调用驻留在 IBM DB2 实例之外模块的过程/函数的能力。这些模块可以用 C/Java/.NET 和其他通用语言实现（与 SQL PL 不同，后者在 IBM DB2 实例的上下文中创建和执行）
IMPLICIT_SCHEMA	允许被授权者隐式创建架构
DBADM	授予数据库管理员权限；DBA 拥有所有特权并能够把这些特权授予其他人
LOAD	允许被授权者使用 LOAD 实用工具把数据迁移到数据库；要成功执行加载，还需要其他对象级的权限
SECADMIN	这个权限（只能由 SYSADM 授予）可以用于创建/删除授予和/或撤消安全标签、安全策略、会话用户特权和在数据库对象上执行 SQL 语句 TRANSFER OWNERSHIP。没有其他权限（包括 SYSADM）具有这些特权
SYSADM	最高级别的管理权限。可以授予和撤消任何其他权限，包括 SECADM，以及运行任何命令、创建/删除任何数据库对象和查询数据的能力
SYSCTRL	提供在数据库管理器实例上执行维护和实用工具操作的能力，但是不允许直接访问数据库中的数据
SYSMAINT	本质上与 SYSCTRL 权限相同，不过有更多限制。例如，具有这样授权的用户不能创建或删除数据库
SYSMON	使用户能执行数据库系统监视器快照

www.TopSage.com

在DB2中，一般的GRANT语句遵循下面的语法：

```
GRANT PRIVILEGES | <system privilege,...>
ON DATABASE
TO USER <user> | GROUP <group> | PUBLIC
```

可以看到，DB2没有WITH ADMIN OPTION子句（Oracle中有这个子句），而且不能对特定数据库使用ALL PRIVILEGES，尽管授予DBADM在本质上是为了同一个目的。

以下是在ACME数据库中把CREATETAB系统特权授予PUBLIC（所有用户）的例子：

```
GRANT CREATETAB
ON DATABASE
TO PUBLIC

DB0000I The SQL command completed successfully
```

注意，与Oracle或Microsoft SQL Server不同，在IBM DB2 9.5中，必须指定关键字USER和GROUP。把系统特权（在IBM DB2术语中称为数据库权限）授予名为SALES的组的语法如下：

```
GRANT CREATETAB
ON DATABASE
TO GROUP sales
DB0000I The SQL command completed successfully
```

如果既没有指定USER，又没有指定GROUP关键字，那么DB2就使用一套安全授权规则来解决潜在冲突。如果在OS中这个名称定义为GROUP，则假定为GROUP。如果在OS中定义为USER或未定义，则假定为USER。如果这个名称既是指GROUP又是指USER（一个GROUP和一个USER具有相同的名称是可能的），那么将生成一个错误。如果使用了外部DCE身份验证，会生成同样的错误。本章后面有更多关于身份验证方法的内容，在供应商的文档中可以找到详细的论述。

要授予DBADM权限，用户必须拥有SYSADM权限。SYSADM和DBADM都可以把其他特权授予用户或组。本章后面有更多关于IBM DB2权限的内容。

Note 注意：IBM DB2 9.5提供了保护数据的其他方式，即使是具有SYSADM权限的用户。它被称为基于标签的访问控制（Label-Based Access Control，LBAC）。可以用于保护全部的行、列，或者两者。由表安全策略管理这种控制，既可以与表一起创建（使用SECURED WITH语句）也可以稍后添加（使用ADD SECURITY POLICY）。关于此功能的更多信息，请参考供应商的文档。

Microsoft SQL Server 2008没有**系统特权**，或者至少与Oracle或IBM不同。针对特定的SQL语句把特权授予一个用户（或角色）。一旦授予了特权，用户就可以执行这个语句定义的操作。SQL Server 2008定义了许多**安全对象**，即可以授予**主体**（用户、组、角色和应用程序角色）的数据库服务器资源（登录、数据库、角色、用户、架构等）。在Windows操作系统级别（域登录、本地登录）、SQL Server级别（SQL Server登录）和数据库级别（数据库用户/角色和应用程序角色）定义这些主体。表12-6列出了授予这些主体的一些权限。

表12-6 常用的Microsoft SQL Server 2008权限

许可	可应用的安全对象
SELECT	同义词、表、视图、列、表值函数
UPDATE	同义词、表、视图、列
REFERENCES	标量函数和聚合函数、表、视图、列、表值函数、service broker队列
INSERT	同义词、表、视图、列
DELETE	同义词、表、视图、列
EXECUTE	过程、函数（标量和聚合）、同义词
RECEIVE	Service broker队列
VIEW DEFINITION	同义词、表、视图、表值函数、标量函数和聚合函数、过程
CREATE	过程、函数（标量和聚合）、同义词、service broker队列、数据库
ALTER	过程、函数（标量和聚合）、同义词、service broker队列
TAKE OWNERSHIP	过程、函数（标量和聚合）、同义词、service broker队列
CONTROL	过程、函数（标量和聚合）、同义词、service broker队列

www.TopSage.com

TopSage.com

Note 注意：要查看权限，从 of_buildt_permissions 的一个表值函数（传递一个安全对象的名称，例如程序集、用户、应用程序角色等）选择，或查询系统视图 sys.database_permissions 和 sys.server_permissions。

注意 SQL Server 没有针对这些特权的 WITH OPTION 子句：

```
GRANT ALL | <statement,...>
TO <security_account>
```

需要特别权限（特权）的语句是那些如果被误用就会造成巨大损害的语句，例如向数据库添加新的对象，改变现有对象以及执行某些管理任务。第 4 章中详细讨论了这些语句中的大多数。

系统权限被绑定到数据库（Microsoft SQL Server 也使用这个概念；最接近的 Oracle 等价概念是**架构**），而且是层次结构的。例如，要授予执行 CREATE DATABASEA 语句的特权，必须处于 SQL Server 的 **master** 数据库环境中，因为这个语句产生的结果会影响 SQL Server 2008 的整个实例。

安全账户是指 SQL Server 用户、SQL Server 角色、Windows 用户或 Windows 组。由 SQL Server 定义的安全账户有一定粒度。因此，授予某个用户的特权（在 SQL Server 或 Windows 上）只影响这个用户。授予某个角色或 Windows NT 组的特权将影响这个角色或组的所有成员。在组/角色和它们的成员之间发生某个特权冲突时，最具限制性的特权 DENY 获得优先权（在本章后面讨论）。

Note 注意：为了利用 Windows 操作系统的组和账户有效地管理 SQL Server 安全，必须理解底层的 Windows OS 安全。

在 ACME 数据库中，把 CREATE DATABASE 语句授予用户/角色会产生下面的错误：

```
USE acme

GRANT CREATE DATABASE
TO PUBLIC

CREATE DATABASE permission can only be granted in the
master database.
```

Note 注意：USE 关键字不是 SQL 标准的一部分。它只在 Microsoft SQL Server 和 Sybase Adaptive Server 使用的 Transact-SQL 用语中有效。

把环境更改为 **master** 数据库可以解决这个问题：

```
USE master

GRANT CREATE DATABASE
TO PUBLIC

The command(s) completed successfully.
```

要授予某些更局部的、适合某个数据库本身的特权，需要一个更小的范围。要授予在 ACME 数据库中创建视图的特权，必须处于 ACME 数据库中：

```
USE acme

GRANT CREATE VIEW
TO PUBLIC

The command(s) completed successfully.
```

不能授予执行某些 Transact-SQL 语句所需的特权；被授权者必须是某个预定义的**固定服务器角色**的成员（在本章后面进行讨论）。这意味着要执行 KILL（停止 SQL Server 安装过程中的一个进程）等语句，必须是 processadmin 固定角色的成员。为了能够授予 ALL 语句权限，则必须是 sysadmin 固定角色的成员。db_owner 角色的成员可以授予或撤消其数据库中的任何特权等。

2. 授予对象级特权

就本质而言，对象级特权具有比系统级特权更细的粒度。在 GRANT 语句的语法中可以反映出来。这些特权可以一直到列级（如果对象是数据库表或视图），或者到达数据库中的任何其他对象，如存储过程、函数和触发器。表 12-7 中列出了 SQL 对象级特权。

表 12-7 SQL 对象级特权

对象特权	适用于	说明
INSERT	SQL:2003、IBM DB2、Oracle、Microsoft	允许被授权者在数据库表（或视图）中插入数据。这个权限可以进一步限制到特定的列

www.TopSage.com

续表

对象特权	适用于	说明
SELECT	SQL:2003、IBM DB2、Oracle、Microsoft	允许被授权者从数据库表、视图或某些其他特定于实现的对象（序列、快照等）中选择数据。这个权限可以进一步限制到特定的列
UPDATE	SQL:2003、IBM DB2、Oracle、Microsoft	允许被授权者更新数据库表或视图中的数据。这个权限可以进一步限制到特定的列
DELETE	SQL:2003、IBM DB2、Oracle、Microsoft	允许被授权者删除数据库表或视图中的数据
ALTER	IBM DB2、Oracle、Microsoft	尽管通常被认为是一个系统级特权，但允许被授权者改变某些数据库对象，例如表和视图。一些对象可能是特定于实现的。尽管这个特权不是标准SQL的一部分，但所有三个供应商都实现了
INDEX	IBM DB2、Oracle、Microsoft	允许被授权者在现有表上创建一个索引
UNDER	SQL:2003、Oracle	允许被授权者在视图下创建子视图
EXECUTE	SQL:2003、IBM DB2、Oracle、Microsoft	允许被授权者执行现有存储过程或函数
REFERENCES	SQL:2003、IBM DB2、Oracle、Microsoft	允许被授权者修改现有表（或创建新表），该表集成了一个引用某些其他表的外键约束

下面是向数据库对象授予特权的Oracle 11g通用语法：

```
GRANT [ALL [PRIVILEGES]] | <object_privilege,...>
[ON [<schema>].<object>]
TO <user> | <role> | <PUBLIC>
[WITH {GRANT OPTION | HIERARCHY OPTION}];
```

与GRANT系统特权语句一样，需要提供一个想要授予的所有特权列表（参见表12-3了解相关对象特权列表）。指定ALL将授予所有特权，但作为一个用户，必须具有足够的系统特权才能授予这个选项。另外可以指定想要授予访问权限的列的列表（如果可用）。然后指定被授予特权的对象——表、视图、过程、包、序列、同义词和任何其他有效的Oracle数据库对象（新的JAVA和DIRECTORY子句不是SQL的一部分，而且超出了本书的范围）。

注意，不是每个对象都有给定的特权：某些特权与对象无关。例如，如果试图把REFERENCES特权分配给一个Oracle序列，则这个REFERENCES特权没有什么意义，同样，EXECUTE特权对一个表而言也没有意义。因此，如果指定ALL特权，则只授予对象类型允许的那些特权。下面的GRANT语句将生成一个错误：

```
SQL> GRANT EXECUTE
     ON deduction
     TO PUBLIC;

ORA-02224: EXECUTE privilege not allowed for tables
```

对象特权可以授予用户、角色或PUBLIC（这是一种把特权授予数据库中每个用户的特殊方式）。WITH GRANT OPTION表示被授权者能将这个特权授权其他用户或角色。

Note 注意：WITH HIERARCHY OPTION（SQL:2003标准的一部分，只有Oracle实现了它）与面向对象功能相关，表示不仅对这个对象本身授予特权，而且针对所有子对象授予。这个选项只有与在对象上授予的SELECT特权一起使用时才有意义。

Note 注意：在Oracle 11g中，只有当被授权者是一个用户或PUBLIC时，才可以指定WITH GRANT OPTION。当授予一个角色时，这个选项无效。

下面是一个非常清楚的例子。它把ACME数据库中PRODUCT表的ALL特权授予SALES_FORCE角色。只要执行下面的语句，任何属于SALES_FORCE角色的用户都能够执行这些特权：

```
SQL> GRANT SELECT, UPDATE, DELETE
     ON product
     TO sales_force;

Grant succeeded.
```

谈及对象级特权时，IBM DB2可能具有最多变的语法。除了表12-7中列出的对象级特权外，它还拥有一些自己的对象级特权（参见表12-8）。

这些特权中的每一个都遵守特定的规则。不能想当然地认为这些规则在不同的对象之间可以传递。参考DB2文档（或特定于IBM DB2的书籍）获得更多信息。

www.TopSage.com

表 12-8 一些 IBM DB2 对象级特权

对象特权	语法	说明	适用于
CONTROL	GRANT CONTROL ON OBJECT <object_name> TO USER <user> | GROUP <group> | PUBLIC	允许被授权者删除对象	索引、包、表、视图、别名
BIND	GRANT BIND ON PACKAGE <package name> TO USER <user> | GROUP <group> | PUBLIC	允许被授权者绑定包	包
ALTERIN	GRANT ALTERIN ON SCHEMA <schema name> TO USER <user> | GROUP <group> | PUBLIC	允许被授权者改变架构中现有对象或向对象添加注释	架构
CREATEIN	GRANT ALTERIN ON SCHEMA < schema name> TO USER <user> | GROUP <group> | PUBLIC	许可被授权者在架构中创建对象	架构
DROPIN	GRANT ALTERIN ON SCHEMA < schema name> TO USER <user> | GROUP <group> | PUBLIC	允许被授权者删除架构中的对象	架构
USAGE	GRANT USAGE ON SEQUENCE <sequence_name> TO USER <user> | GROUP <group> | PUBLIC	允许被授权者通过 NEXTVAL 或 PREVVAL 表达式访问序列	序列
USE OF TABLESPACE	GRANT USE OF TABLESPACE <toblespace_name> TO USER <user> | GROUP <group> | PUBLIC	允许被授权者访问并使用指定的表空间	表空间
PASSTHRU	GRANT PASSTHRU ON SERVER <server_name> TO USER <user> | GROUP <group> | PUBLIC	允许被授权者以传递模式访问并使用数据源	服务器

最重要的事情（而且与 SQL 标准密切相关）是向表或视图授予特权。下面是更特殊的语法，用于把这样的特权授予表（或视图、别名）：

```
GRANT [ALL [PRIVILEGES]] |
      ALTER |
      CONTROL |
      DELETE |
      INDEX |
      INSERT |
      REFERENCES (<column name>,...) |
      SELECT [(<column name>,...)] |
      UPDATE [(<column name>,...)]
ON [TABLE] <table_name> | <view_name> | <nickname>
TO USER <user> | GROUP <group>| PUBLIC
[WITH GRANT OPTION]
```

Note 注意：在 IBM DB2 中，GRANT ALL 语句授予除 CONTROL 特权外的所有特权。

例如，要把引用一个表的特权（基于这个表的列创建一个外键）授予 PUBLIC（所有用户），可以使用下面的语句：

```
GRANT REFERENCES
ON TABLE customer(cust_paytermsid_fn)
TO PUBLIC
WITH GRANT OPTION

DB0000I The SQL command completed successfully
```

这样授予特权来创建和删除一个引用这个表（作为父表）的外键。当然，要执行任何 GRANT 语句，授予者必须具有足够的特权来执行任何 GRANT 语句。

下面授予对象级特权的一般语法在 Microsoft SQL Server 2008 中有效：

```
GRANT
    {ALL [PRIVILEGES]}
    | <permission1>,<permission2>,.....
        [(<column1>,<column2>...)] [ ON <table> | <view>]
    | [ ON <stored_procedure> | <extended_procedure>]
    | [ ON <user_defined_function>]
TO <security_account>,...
[WITH GRANT OPTION]
[AS <group> | <role>]
```

Note 注意：PRIVLEGES 是一个可选的 SQL:2003 关键字，可以在三个 RDBMS 的任何一个中用于符合标准。

很多对象级特权的 GRANT 语句类似于系统级特权使用的 GRANT 语句。例如，ALL 关键字同时授予所有的特权，而 security_account 指完全相同的事情（参考前面的段落），等等。

由于前面代码中的语句授予对象（以及对象的组成部分，例如表中的列）特权，所以，它有许多不同的选项。可以在表或视图或者其中的一个或多个列，以及 SQL Server 数据库中某些其他有效的现

www.TopSage.com

大家网 TopSage.com

有对象上授予特权（表 12-6 列表中的）。这个语法允许同时列出多个子对象（列、表等），但不能同时列出多个非关联的对象，如表和视图。

Microsoft SQL Server 与 Oracle 和 IBM DB2 之间的显著区别是 SQL Server 有一个额外的 AS 子句。当特权（在 Microsoft 的术语中指权限）授予一个组或角色的同时授予那些不是该组或角色成员的用户时，使用这个子句。因此，组或角色没有足够的权限。在这种情况下，必须使用 WITH GRANT OPTION 和 AS 子句。这样被授权者可以在 AS 子句中指定的组或角色的权限下授予这个特权。

下面的 SQL 语句把表 PRODUCT 的 SELECT 对象级特权授予 Microsoft SQL Server 2008 预定义的角色 PUBLIC（参阅本章后面有关安全模型概述的段落）：

```
GRANT SELECT
ON product
TO PUBLIC

The command(s) completed successfully.
```

现在，任何属于角色 PUBLIC 的验证用户能够对表 PRODUCT 发出一个 SELECT 语句。下面给出一个更复杂的语句例子，它限制用户查看的列。这个语句授予 PUBLIC 角色只能从表 PRODUCTS 中选择和更新两列的权限，隐藏了其余列：

```
GRANT SELECT,
      UPDATE (prod_id_n,
              prod_price_n)
ON product
TO PUBLIC

The command(s) completed successfully.
```

Tip　提示：在一般的 Microsoft SQL Server 中，选择使用下列任意一个 GRANT 语句的语法没有什么区别：

```
GRANT <...> (prod_id_n, prod_price_n) ON product
```

或

```
GRANT <...> ON product (prod_id_n, prod_price_n)
```

这些列的列表可以在表名后也可以在表名前。SQL Server 负责解决依赖性问题。

12.3.2　REVOKE 特权

这个命令撤消数据库用户、角色或组的系统级或对象级特权。角色是被授予了特权集的命名实体，它的撤消方式与撤消用户完全相同。下面的语法使用 SQL:2003 标准撤消系统特权：

```
REVOKE [GRANT OPTION FOR]
ALL [PRIVILEGES] | <privilege>,...
FROM USER <user> | GROUP <group> | PUBLIC
CASCADE | RESTRICT
```

可以看到，既可以撤消一个特权，也可以撤消那个特权的 GRANT OPTION，这意味着用户/角色将失去把这个特权授权其他用户的能力。他们自己的特权会保留而不受影响。

SQL:2003 也指定了两个子句：RESTRICT 和 CASCADE。使用第一个子句，只有当数据库中没有废弃的特权时，这个语句才成功。例如，当用户已经把这个特权授予某些其他用户时，可能会出现这样的情况。当决定无论如何都要撤消这个特权，而且要使这个改变扩展到所有接受了来自这个用户的特权的用户时，就必须指定 CASCADE 子句。在“三大”RDBMS 实现中，这些子句都是可选的，但是在 SQL:2003 标准中至少需要一个。

通常，每一个供应商对于实现这个语句都有自己的考虑。

1. 撤消系统级特权

Oracle 11g 的语法遵循 SQL:2003 标准，但不支持 CASCADE 和 RESTRICT 子句。它也不允许撤消 GRANT/ADMIN OPTION。

```
REVOKE ALL [PRIVILEGES] | <role> | <system_privilege>,...
FROM    <user> | <role> | PUBLIC;
```

当然，要撤消一个系统特权或角色，需要有足够的特权，或者仅有 ADMIN OPTION 特权。

Note　注意：前面的语句只能撤消通过 GRANT 语句授予的特权和角色。如果这个特权是用初始化参数 OS_USERS（参阅 GRANT 特权部分的“注意”内容）授予的，则不能用 REVOKE 语句撤消。

www.TopSage.com

可以用下面的语句撤消用户 NEW_USER 的系统级特权 CREATE TABLE：

```
SQL> REVOKE CREATE TABLE
     FROM new_user;
Revoke succeeded.
```

REVOKE 会立即生效——Oracle 执行语句的时候，用户就失去了这个特权。如果这个特权是从 PUBLIC 上撤消的，则每一个**通过 PUBLIC 授权**的用户都会失去这个特权。不过在这种情况下，直接或通过角色授予用户的特权会保留，不受影响。撤消系统特权的规则很复杂。例如，如果从角色撤消一个特权（或角色），则只是在角色上撤消它。如果这个角色已经把这个特权（或角色）授予另外一个用户（或角色），则那个用户可以继续执行这个特权（或角色）。在 Oracle 中，系统特权的撤消没有级联效应。例如，假定授予 user1 系统级特权 WITH ADMIN 选项，接着 user1 把这个特权授予 user2。那么，即使以后从 user1 撤消这个特权，user2 仍将继续保留这个特权。

同时请记住，从用户撤消特权不会影响用户已经创建的任何对象。撤消特权时，已经使用这些特权创建的所有对象都将继续保留在数据库中。

在 IBM DB2 中，撤消系统级特权很简单——只需遵循相反的 GRANT 语句路径：

```
REVOKE PRIVILEGES | <system privilege>,...
ON DATABASE
FROM USER <user> | GROUP <group> | PUBLIC
```

下面是讨论 DB2 中的 GRANT 语句时的例子，它撤消了数据库 ACME 中 PUBLIC（所有用户）的 CREATETAB 系统特权：

```
REVOKE CREATETAB
ON DATABASE
FROM PUBLIC

DB0000I The SQL command completed successfully
```

当没有指定 USER 或 GROUP 时，DB2 查看系统目录（有关系统目录的更多内容在第 13 章讨论）来确定被授权者的类型。如果用户具有某些更高权限或者属于拥有这个特权的组，则撤消一个特权未必会撤消执行这个操作的能力。

Microsoft SQL Server 2008 针对**语句权限**（就是 Microsoft 的**系统特权**）的 REVOKE 语句类似于 Oracle 的 REVOKE 语句。同 Oracle 一样，在这个级别不支持 CASCADE 和 RESTRICT，也不能撤消 GRANT OPTION：

```
REVOKE [GRANT OPTION FOR]
    [ALL [PRIVILEGES]] | <permission> [column,...]]
    [ON <securable>TO | FROM <security_account>,...
[CASCADE] [AS principal]
```

在系统级别，REVOKE 语句中的 ALL 关键字只能由 SYSADMIN 固定角色的成员使用。本章中针对 SQL Server 2008 的 GRANT 部分中提到的所有限制和注意内容在这里也适用。

选项[GRANT OPTION FOR]指定授予选项的功能也将被撤消。如果使用[CASCADE]子句时需要这个选项。[As principal]指示主体首先被授予该特权。

例如，要从用户 NEW_USER 撤消创建表的特权，可以使用下面的语句：

```
REVOKE CREATE TABLE
FROM new_user

The command(s) completed successfully.
```

2. 撤消对象级特权

SQL:2003 标准在撤消系统级或对象级特权间没有区别，两者使用相同的语法。但在供应商的 RDBMS 实现中，这一情况大不相同；就像对象级特权的 GRANT 语句一样，REVOKE 语句可以相当复杂。本书中的信息只涵盖了这个主题的基础内容，因为它是由供应商实现的。参考特定于供应商的文档来获得更多的信息。

除了大量像 JAVA 和 DIRECTORY 这样的可选子句（为简单起见，跳过这些内容）外，Oracle 11g 的撤消对象特权的语法还算标准：

```
REVOKE ALL [PRIVILEGES] | <object_privilege,...> [(<column>)]
ON [<schema>].<object>]
FROM <user> | <role> | PUBLIC
CASCADE CONSTRAINTS [FORCE];
```

对象的特权可以是表 12-3 中列出的任何一个（假定它与对象有关）。COLUMN 子句指定这个语句

www.TopSage.com

可适用的一个可选列或列的列表。被撤消特权的对象可以是任何数据库对象。

可以从用户、角色或PUBLIC撤消特权。在特权撤消后，用户立即终止行使这些特权。

如果从一个角色撤消特权，则授予了该角色的所有用户都将失去这个特权，且立即生效。从PUBLIC撤消的特权就是从通过PUBLIC授予了这个特权的每一个用户上撤消。不过，如果特权是直接或者通过角色授予用户的，则这个撤消语句没有影响。

下面的语句从PUBLIC撤消对表CUSTOMER（ACME数据库）的CUST_PAYTERMSID_FN和CUST_SALESMANID_FN列的INSERT、DELETE和UPDATE特权（假定之前已经授予了这些特权）：

```
SQL> REVOKE INSERT,
            DELETE,
            UPDATE (cust_paytermsid_fn, cust_salesmanid_fn)
    ON CUSTOMER
    FROM PUBLIC;

Revoke succeeded.
```

与被撤消的特权有关的所有依赖对象（例如，对基础表失去了特权的用户所创建的视图）都变为无效。

Note 注意：如果对象上没有授予特权，则Oracle既不会执行任何操作，也不会返回错误。

只有撤消REFERENCES特权或ALL时，才与CASCADE CONSTRAINTS子句有关。它解除用户使用这个特权定义的所有引用完整性约束。

撤消EXECUTE对象特权时指定FORCE将导致特权被删除。即使它留下了用户定义类型的对象，所有依赖对象都将无效。

与系统特权不同，在Oracle中撤消对象特权有级联效应。假定通过WITH GRANT选项授予user1一个特权，user1随即又把这个特权授予user2。如果随后从user1撤消这个特权，那么user2也同时失去特权。此外，user2基于这个特权进行的任何操作都将被取消，就好像user2从来没有拥有过这个特权一样。

与IBM DB2中的系统特权一样，它的对象级特权使用与GRANT语句类似的语法进行撤消。DB2不支持撤消GRANT OPTION，也不支持CASCADE–RESTRICT子句。例如：

```
REVOKE ALL [PRIVILEGES] | <privilege,...>
ON <object> <object_name>
FROM USER <user> | GROUP <group> | PUBLIC
```

下面的例子从IBM DB2的PUBLIC（所有用户）撤消引用一个表的特权（基于这个表的列创建一个外键）：

```
REVOKE REFERENCES
ON TABLE customer (cust_paytermsid_fn)
FROM PUBLIC

DB0000I The SQL command completed successfully
```

同样，如果没有被指定USER或GROUP，则IBM DB2查询系统目录中的信息，如果被撤消特权的用户由一个外部DCE进行身份验证，则生成一个错误（参见本章后面的OS安全集成）。

Note 注意：当撤消用于创建依赖对象的特权时，要遵循一些规则。例如，当用户失去SELECT特权时，会撤消该用户的由一个表传递到一个视图的SELECT特权；正如他不能从这个表中进行选择一样，这个用户也不能从视图中进行选择。

如果用户具有某些更高级别的权限或者属于一个拥有被撤消特权的组，那么撤消一个对象级特权未必会撤消执行这个操作的能力。

Microsoft SQL Server 2008支持撤消GRANT OPTION和CASCADE子句（但不支持撤消RESTRICT）。下面的REVOKE语句只适用于当前数据库环境：

```
REVOKE [GRANT OPTION FOR]
ALL [PRIVILEGES] | <permission>,...
[(<column,...>)] ON [<table> | <view>] |
[ON <table> | <view>] [(<column>,...)] |
[ON <stored_procedure> | <extended_procedure>] |
[ON <user_defined_function>]
FROM <security_account>,...
[CASCADE]
```


www.TopSage.com

```
[AS <group> | <role>]
```

特权不能从系统固定角色中撤消。当从一个 Windows NT 组撤消权限时，安全账户参数的格式应为 BUILTIN\<domain>，其中 domain 代表计算机名或定义此用户的域。

Note 注意：要撤消特权，必须有足够的权限。SYSADMIN 固定服务器角色的成员 DB_OWNER 和 DB_SECURITYADMIN 默认在其相应的范围内授予 REVOKE 特权。

例如，要撤消 ACME 数据库中表 PRODUCT 上的 SELECT 权限，可以提交下面的语句：

```
REVOKE SELECT
ON product
FROM PUBLIC

The command(s) completed successfully.
```

如果一开始就授予 WITH GRANT OPTION，那么必须指定 GRANT OPTION FOR 和 CASCADE。如果没有指定 CASCADE 选项（默认情况），则通过安全账户（从这个安全账户撤消特权）的用户不会失去这个特权。

SQL Server 2008 的 DENY 语句

特定于 Microsoft 的 DENY 语句为系统特权增加了一个额外的粒度级别，这在 Oracle 和 DB2 中是没有的。SQL:2003 标准也没有定义这样的语句。Microsoft SQL Server 的未来发布版本可能也不会支持它。

如果使用角色把特权集授予用户，则可能会出现这样一种情况：这个特权集应该针对一个且只有一个用户进行修改，同时要保留授予这个角色的所有其他用户的这个特权集。试图从通过角色授予的用户上撤消特权将导致一个错误。可以创建另一个角色，通过第一个角色授予特权子集，然后将它授予用户。也可以绕过角色，把这些特权直接授予用户。这些方法中的任何一种都是可行的，但会导致冗余并可能引起混乱。DENY 语句为这个问题提供了恰当的解决方案。它允许授予角色完整的特权集，然后拒绝将其中一些特权授予用户。

下面的 DENY 语句既可以用于系统级特权也可以用于对象级特权，使用的语法与 GRANT 和 REVOKE 语句的语法非常类似：

```
DENY [ALL] [PRIVILEGES] | <permission>,...
[(<column>)] ON <table> | <view>] |
ON [<table> | <view>] [(<column>,...)] |
ON [<stored_procedure> | <extended_procedure>] |
ON [<user_defined_function>]
TO <security_account>,...
[CASCADE]
```

CASCADE 子句意味着被 security_account 拒绝的特权（权限）也将被具有此 security_account 权限的所有其他用户/角色/组拒绝。同样，Microsoft SQL Server 2008 也不推荐使用 ALL 权限，提供此权限只是为了向后兼容。尽管如此，在早期版本的 RDBMS（7.0 到 2005）中，这些权限仍然是很有用的。

例如，有一个角色 SALES，它具有数据库中的所有特权，用户 NEW_USER 需要除 CREATE TABLE 外的所有这些特权。下面的语句将适当的特权授予这个用户：

```
GRANT sales TO new_user
GO
DENY CREATE TABLE TO new_user
GO
```

DENY 语句总是具有优先权。因此，尽管用户 NEW_USER 具有从 SALES 角色继承的特权，但如果他试图创建一个表，将收到一条错误消息。授予了 SALES 角色的其他成员这样做则不会有任何问题。

12.4 使用角色管理安全

ROLE 是在关系数据库中引入的一个抽象概念，通过按照某些标准（通常是一个作业函数）对用户的特权进行分组来方便用户管理任务。例如，如果多达 100 人的会计团队除了需要某些系统级特权外，还需要他们日常访问的几十个对象的特权，那么有两种选择方案。第一种是遍历每一个用户并单

www.TopSage.com

独授予他所有必需的特权。第二种是创建一个组（角色），如ACCOUNTANTS，把所有特权授予这个角色，从而把这个角色授予组中的所有用户。撤消特权会面临同样的选择。可以很明显地看出哪种选择更好。

除了具有一些可以授予用户的预定义系统角色外，有些RDBMS还提供了角色创建能力。Oracle 11g和Microsoft SQL Server 2008具有这种功能，而DB2只使用固定的预定义角色（权限）。

Oracle 11g的角色是可以授予（或从中撤消）一个用户或另一个角色的特权集合，因而提供了一个特权层次结构。角色被启用后（使用SET ROLE语句或由数据库管理员完成），才能够传递授予它的所有特权。

Oracle 11g有若干预定义角色，通过这些角色向用户授予特权。表12-9中列出了这些角色及简要说明。

表12-9 Oracle 11g的预定义角色

预定义角色	说明
CONNECT RESOURCE DBA	用于与以前的Oracle版本兼容。Oracle明确声明在未来的发布版可能不再支持这些角色。在Oracle 11g中，这些角色中的每一个都有一组关联的特权。例如，DBA角色有124个特权；RESOURCE和CONNECT各有8个。其中一些有重叠的特权（如CREATE TABLE或CREATE VIEW），还有一些是唯一的。参考DBA_SYS_PRIVS字典视图来了解特权的完整列表及其说明
DELETE_CATALOG_ROLE EXECUTE_CATALOG_ROLE SELECT_CATALOG_ROLE	提供给需要访问数据字典视图的用户
EXP_FULL_DATABASE IMP_FULL_DATABASE	提供给需要执行全部数据库输入或输出的用户
AQ_USER_ROLE AQ_ADMINISTRATOR_ROLE	Oracle的高级队列功能所需的特权
SNMPAGENT	供企业管理器/智能代理使用
RECOVERY_CATALOG_OWNER	拥有一个恢复目录的用户所需要的特权
HS_ADMIN_ROLE	授予对DBMS_HS包的访问权限，它是异类服务管理所必需的

既不能把新特权添加到一个预定义角色，也不能从角色中撤消任何特权。

下面的Oracle语法创建一个自定义角色，非常简单：

```
CREATE ROLE <role name>
[NOT IDENTIFIED][IDENTIFIED {BY <password> |
              EXTERNALLY |
              GLOBALLY |
              USING [<schema>].[<package>]
             }
];
```

在这个语句中，只有角色名是必需的参数，其余都是可选的。如果这个角色要拥有自己的密码，必须使用INENTIFIED子句。NOT IDENTIFIED子句指示启用它不需要密码。

对于已识别的用户，INENTIFIED BY <password>子句实际上创建了一个使用这个密码的本地用户（假如这个用户之前不存在）。不过，启用这个角色时，需要指定密码。USING [<schema>].[<package>]子句创建一个应用程序角色（将它与Microsoft SQL Server中的应用程序角色比较），这意味着可以由特定的应用程序利用已授权的包来启用这个角色。

Note 注意：在Oracle RDBMS中，包是预编译例程的集合（通常用PL/SQL编写），并驻留在RDBMS自身中。用户可以通过包中定义的过程和函数来访问数据库功能。

使用EXTERNALLY子句创建一个外部用户（参阅本章后面的“OS安全集成”侧栏），使用GLOBALLY子句创建一个由企业目录服务授权的全局用户。

下面是使用一组默认选项为ACME数据库创建角色的例子：

```
CREATE ROLE sales_force;

Role created.
```

现在，可以把特权授予这个角色（参阅本章前面介绍GRANT语句的段落来了解更多信息），稍后通过向用户分配这个sales_force角色将这些特权授予需要的每一个用户。

www.TopSage.com

Note 注意：可以在当前数据库会话期间使用 SET ROLE 语句来启用或禁用 ROLE。数据库管理员可以对并发的角色数进行限制。

ALTER ROLE 和 DROP ROLE 语句可分别用于改变或删除自定义角色。

任何访问 IBM DB2 数据库的用户都必须有一个有效的 OS 帐户。通过验证后，用户对数据库对象的访问由 RDBMS 内部的一个权限（角色）和特权系统进行管理。DB2 不支持用户定义的角色，而是依赖系统的预定义权限（角色），用户可以是其中的成员，同时也依赖 GROUP，其行为几乎与 Oracle 和 Microsoft SQL Server 2008 中使用的角色行为一样。

系统权限（角色）包括系统管理（SYSADM）、系统控制（SYSCTRL）、系统维护（SYSMAINT）以及数据库管理（DBADM），表 12-10 中列出了这些系统权限。这些角色中的每一个都暗含某些特权。某些类型的特权自动授予由 OS 验证的每一个用户。

表 12-10 IBM DB2 系统权限

系统权限	说明
SYSADM	系统管理，包括所有其他系统权限的所有特权以及授予和撤消 DBAMD 权限的能力
SYSCTRL	系统控制，包括创建、更新和删除数据库的特权。不允许直接访问数据
SYSMAINT	系统维护，能够在与一个实例关联的所有数据库中进行数据库维护任务，包括修改配置文件的权限以及备份和还原数据库的权限
DBADM	数据库管理，拥有一个单独数据库中的所有特权

在 DB2 中，特权定义为已验证的用户访问和使用表、视图和存储过程这样的数据库资源的权限；本章后面部分将对这些权限进行更详细的讨论。

表 12-7 中列出了授予权限（表 12-10）的大多数 IBM DB2 特权（包括系统级和对象级）。

Note 注意：有些数据库对象的特权并不与讨论的所有 RDBMS 都相关。例如，PACKAGE 对象可以在 DB2 或 Oracle 数据库中找到，但它并不存在于 Microsoft SQL Server 中。

最高权限级别属于 SYSADM，它能完全控制所有数据库对象以及包含这个数据库的 DB2 安装。在 SYSADM_GROUP 参数中指定 SYSADM 权限的操作系统组，该参数与操作系统有关。如果指定为 NULL，那么在 Windows NT/2000/XP 上，SYSADM_GROUP 默认为管理员组。对基于 UNIX 的系统，如果初始值为 NULL，那么 SYSADM_GROUP 默认为数据库实例所有者的主组。

SYSCTRL 和 SYSMAINT 角色表示层次结构的较低级别，紧接着是 DBADM。没有授予其中任何一种角色的用户逐个对象授予特权。DBADM 可以创建一个新组，然后把特权授予这个组和组中的用户。这是一种管理数据库中具有类似职责的大量用户的特权的便捷方式。它在功能上类似于 Oracle 和 Microsoft SQL Server 中的角色。

对于授予系统权限的语法，可以参考本章的 GRANT 语句部分。

OS 安全集成

请注意，操作系统（OS）在第一个计算机数据库发明之前就已经存在了，如果没有某种类型的 OS，RDBMS 是无法运转的。每一个操作系统都带有它自己的安全机制。本书中讨论的所有 RDBMS 在某种程度上都提供了与运行它们的 OS 的安全集成。本质上，这归结为使用操作系统帐户和特权来访问数据库，而不是依赖 RDBMS 自身。

例如，Microsoft SQL Server 2008 就有紧密的 OS 集成安全，它允许具有有效 Windows 帐户的用户基于他们的 Windows NT/2000/XP 凭据进行身份验证。用户不用提供单独的用户 ID 和密码，登录到运行 RDBMS 的机器上即可自动访问 SQL Server 2008。

Oracle 11g 和 IBM DB2 把这个功能扩展到多个操作系统，实现的细节和用法就像这些系统一样有所区别。

DB2 使用一个外部工具进行用户身份验证——既可以是操作系统的，也可以是一个分布式计算环境（DCE）工具。用户必须具有有效 OS 帐户（登录和用户 ID），以便访问数据库。

对于在外部进行身份验证的常规用户，Oracle 数据库管理员必须创建一个用户帐户，这个帐户需要

www.TopSage.com

使用来自 Oracle 初始化参数 OS_AUTHENT_PREFIX 的字符值作为前缀，并标记为 AUTHENTICATED EXTERNALLY。Oracle 使用参数 OS_AUTHENT_PREFIX 的值来检查任何在数据库中创建的用户名是否使用这个值作为前缀。如果一个用户名没有这个前缀，则不允许进行外部身份验证。这只是由 Oracle 实现的基本机制的一个简要说明。有关这个主题的更多信息，请参考供应商的文档。

对于那些打算管理数据库的已通过 EXTERNALLY 验证的用户，Oracle 11g 需要在运行操作系统的机器上创建两个预定义的 OSDBA 和 OSOPER 组。Oracle 11g 也需要使这些用户成为其中一个组的成员。此外，Oracle 启动参数 REMOTE_LOGIN_PASSWORDFILE 必须被设置为其有效值中的一个。

运行在某些操作系统（如 Windows 9x）上的 RDBMS 没有 OS 安全集成，因为 OS 本身没有为此提供工具。

在身份验证级别上，Microsoft SQL Server 2008 提供了两种选择：Windows OS 身份验证和混合身份验证模式。通常认为第一种更好，因为它依赖更健壮的操作系统的安全机制，并且通过网络访问数据库时无需发送无保护的登录信息。第二种提供了更加“个性化的方式”，允许很多用户连接到同一个 SQL Server，而不必添加到 Windows 用户组。Microsoft 定义了一个特殊的系统管理员用户，通常表示为 sa，它负责 Microsoft SQL Server 2008 中的所有管理任务。当直接通过 SSL（与本章前面介绍的证书配合使用）访问数据库时，就需要另一种级别的身份验证。它允许在客户和要加密的数据库之间传递数据，还可以使用 Internet 协议安全（Internet Protocol Security，IPSecs）进行加密。

注意：SQL Server 2008 与 Windows OS 紧密集成。无论在安装期间选择了何种身份验证模式，Windows 身份验证模式将始终允许登录。

授权由 SQL Server 自身来执行。在服务器的系统表中记录有关用户被授予权限的所有信息（通过 INFORMATION_SCHEMA 视图可以访问一部分）。同 IBM DB2 和 Oracle 一样，Microsoft SQL Server 通过**固定角色**和**应用程序角色**实现的用户层次结构系统来强制实施安全（本章前面部分说明了角色的概念），以及数据库对象的权限集。

固定服务器角色提供了一个服务器范围作用内的层次结构，其中允许每个角色执行特定的活动。SYSADMIN 处于最高级别，并拥有执行任何活动的特权。表 12-11 中列出了这些角色。

表 12-11　Microsoft SQL Server 固定服务器角色

固定服务器角色	允许的操作
SYSADMIN	可以在 SQL Server 中执行任何活动（这是最高特权级别）
SERVERADMIN	能够启动/停止服务器并修改服务器的配置
SETUPADMIN	能够管理链接服务器并修改启动过程
SECURITYADMIN	管理登录、密码和权限。允许阅读错误日志
PROCESSADMIN	允许管理 SQL Server 的进程
PUBLIC	本质上针对每一个人；默认情况下只授予这个角色 CONNECT 特权
DBCREATOR	拥有创建、改变和删除数据库的权限
DISKADMIN	允许管理 SQL Server 磁盘文件
BULKADMIN	允许执行 BULK INSERT 操作

不能改变固定角色。不能创建新的固定服务器角色。可以利用 SQL Server 系统存储过程或通过图形用户界面（GUI）向角色添加新成员或从中删除成员。

SQL Server 2008 安全层次结构中的下一级是**固定数据库角色**，如表 12-12 所示。SQL Server 实例中定义的每一个数据库都具有一组预定义的（固定的）数据库角色，可以添加任何数据库用户（登录）。这些角色的范围更有限——限制在声明它们的数据库中。与固定服务器角色一样，这些角色的权限不能改变，但可以创建新的数据库角色（不同于固定服务器角色）。

表 12-12　Microsoft SQL Server 固定数据库角色

固定数据库角色	允许的操作
DB_OWNER	这个组的成员有权限做任何事情——在数据库范围内
DB_ACCESSADMIN	这个角色的成员可以向数据库添加或删除用户

www.TopSage.com

续表

固定数据库角色	允许的操作
DB_SECURITYADMIN	这个角色的成员管理安全——所有的特权、对象、角色等
DB_DDLADMIN	这个角色的成员可以提交任何 DDL 语句，但不能提交 GRANT、REVOKE 和 DENY 语句
DB_BACKUPOPERATOR	这个角色的成员可以提交 DBCC、CHECKPOINT 和 BACKUP 语句
DB_DATAREADER	允许这个角色的成员从数据库中的任何用户表中选择所有数据
DB_DATAWRITER	允许这个角色的成员修改数据库中任何用户表中的任何数据
DB_DENYDATAREADER	不允许这个角色的成员从数据库中的任何用户表中选择任何数据
DB_DENYDATAWRITER	不允许这个角色的成员修改数据库中任何用户表中的任何数据

应用程序角色是 SQL Server 2008 独有的（Oracle 11g 具有些特殊之处）。它们只能由访问 RDBMS 的应用程序来激活。没有预定义的应用程序角色。在某种程度上，它们只是另一种管理组权限的方法，如果用户总是通过某些帐户管理程序连接到数据库服务器，那么 SQL Server DBA 可以为这个帐户管理程序创建一个角色并赋予进行常规工作所需的所有特权。当 SQL Server 接受来自这个帐户管理程序的请求时，就会为这个应用程序激活角色——不早也不晚。如果这个应用程序逐步停止并被一个新的应用程序取代，那么 DBA 为阻止前一个应用程序的访问，必须删除与之关联的应用程序角色。

应用程序角色不包含成员，也没有预定义的应用程序角色。有关应用程序角色的内容远不止这里所描述的。如果需要使用它们，可以参考 SQL Server（和 Oracle）文档。

Note 注意：在系统表中记录列级别的特权，并可以通过 INFORMATION_SCHEMA 的视图 COLUMN_PRIVILEGES 查看。INFORMATION_SCHEMA 视图将在第 13 章详细讨论。通过 sys.database_principals 视图可以访问完整列表。

与其他 RDBMS 一样，可以在对象级分配权限（特权），例如，在表或存储过程中直接分配给列。列是用户可以对其拥有特权的最小对象。除了对象粒度外，特权可以按类型来区分——EXECUTE 特权、SELECT 特权、DELETE 特权等，可以任意组合。特权（包括系统级和对象级）已经在本章前面讨论过了。

SQL Server 的以前版本中没有 CREATE ROLE 语句，由系统存储过程来代替。表 12-13 中给出了一些相关过程的列表。

表 12-13 一些 Microsoft SQL Server 2008 角色管理系统存储过程

固定数据库角色	允许的操作
sp_addrole<role name>,[<role owner>]	在当前数据库中创建一个新角色
sp_droprole<role name>	从当前数据库中删除一个角色
sp_addapprole<role name>,<password>	在当前数据库中创建一个新的应用程序角色
sp_setapprole<role name>,[<password>],[<encryption>]	激活与当前数据库中某个应用程序角色关联的权限
sp_dropapprole<role name>	从当前数据库中删除一个应用程序角色
sp_addrolemember<role name>,<user name>	向一个现有数据库角色添加一个成员
sp_droprolemember<role name>,<user name>	从现有角色中删除一个成员
sp_addsrvrolemember<role name>,<user name>	向现有固定服务器角色添加一个成员
sp_dropsrvrolemember<role name>,<user name>	从固定服务器角色中删除一个成员

Note 注意：Microsoft 提供了许多 hepler 系统存储过程（都以 sp_help 开始）来获得有关角色（固定的或用户定义的）的信息。

SQL Server 2008 允许使用 CREATE ROLE 创建自定义角色。下面的语法非常简单：

```
<code>CREATE ROLE <role_name> [AUTHORIZATION <owner_name>]
```

<owner_name>指定这个新角色所属的数据库用户或角色。如果没有指定所有者，则角色属于执行这个语句的用户/角色。

角色是数据库级别的安全对象。因此，需要使用 GRANT、REVOKE 和/或 DENY 语句配置这些权限。

12.5 使用视图实现安全

可以用于实现安全的机制之一是 SQL 视图（在第 4 章已经讨论过）。使用视图可以限制某个用户

www.TopSage.com

能访问的数据、用户通过视图可以执行的操作类型，或者同时限制两者。

考虑下面的DDL SQL语句，所有三大RDBMS实现一般都可以接受：

```
CREATE VIEW v_customer_status
(
   name,
   status
)
AS
SELECT cust_name_s,
       cust_status_s
FROM   customer
```

这个视图只从表CUSTOMER中选择了两个字段，这个表总共有7个字段。这称为垂直限制，因为它限制了对列（字段）子集的访问。其他字段可能包含只能由更上层管理访问的机密信息。如果把视图的SELECT特权授予某个角色（例如，ROLE "staff"），那么属于那个角色的每一个人都能够看到客户的姓名和状态，而表包含的其他信息对他们来说仍然不能访问。

如果视图V_CUSTOMER_STATUS的SELECT语句由一个经授权的用户执行，将产生下面的结果（但同样的语句由没有授予那个视图特权的用户提交将会生成一个错误）：

```
SELECT *
FROM v_customer_status

name                                                status
--------------------------------------------------- ------
WILE SEAL CORP.                                      Y
MAGNETICS USA INC.                                   Y
MAGNETOMETRIC DEVICES INC.                           Y
. . . . . .
CHGO SWITCHBOARD INC.                                N
```

也可以通过指定一个行的子集在水平方向上限制访问。一个例子是想要授权访问历史数据，这些数据可能是一年前或更早输入的数据，并禁止访问那个日期之后添加的数据。也可以使用来自ACME数据库的例子——假定有一支销售精英队伍，按照职责和经验级别分成两组（一组销售人员处理那些订单总额超过15 000美元的客户，另一组处理生成较少订单的客户）。对于后面的例子，所有三个数据库的SQL语法如下所示：

```
CREATE VIEW v_customer_totals_over_15000
AS
SELECT   *
FROM     v_customer_totals
WHERE    total_price < 15000
```

从下面的视图只选择总额超过15 000的客户记录：

```
SELECT   *
FROM     v_customer_totals_over_15000

customer_name          order_number       total_price
---------------------- ---------------  ------------------
WILE ELECTRONICS INC.  523735             15613.60
WILE BESS COMPANY      523741             15464.10
. . . . . .
WILE SEAL CORP.        523775             15613.60
WILE SEAL CORP.        523781             15464.10
```

Note 注意：视图V_CUSTOMER_TOTALS_OVER_15000建立在另一个视图V_CUSTOMER_TOTALS之上。可以为这些对象中的每一个分配不同的特权。使用这种方法，可以建立一个复杂的、细粒度的安全层次结构。

当然，水平和垂直选择都可以组合到单个视图中。

视图也可以把显示数据限制为汇总信息，如在V_CUSTOMER_TOTALS视图中，有关销售的信息在按照订单号分组的同时进行汇总（参阅附录B了解创建这个视图的SQL语句）。

在视图中可以实现的其他限制包括WHERE子句和JOIN条件。当视图中涉及多个表时，这些子句是有用的。例如，可以限制视图只显示那些已经发出订单的客户而隐藏所有其他客户，如下所示（对所有三个数据库都是有效的语法）：

```
SELECT DISTINCT cust_name_s
FROM   customer cu
  JOIN
       order_header oh
```


www.TopSage.com

TopSage.com

```
    ON   cu.cust_id_n = oh.ordhdr_custid_fn

CUST_NAME_S
--------------------------------------------------
WILE BESS COMPANY
WILE ELECTROMATIC INC.
WILE ELECTROMUSICAL INC.
. . . .
WILE ELECTRONICS INC.
WILE SEAL CORP.
```

视图不仅可以用于 SELECT，还可以用于 UPDATE、INSERT 和 DELETE 语句。其中一些操作由视图对象的固有属性控制，而有些则可以在创建视图对象时指定。例如，不能更新或插入使用聚合函数创建的视图——试图那样做会生成一个错误。这是一种固有行为。另一方面，对于可以更新的视图，可以创建一个约束，它可以基于某些标准接受或拒绝数据修改。约束在下一节讨论。

12.6 利用约束实现安全

约束通常用于维护完整性，可以是引用完整性、数据完整性（也称为**实体**完整性）或域完整性（第4章有更详细的讨论）。这里，我们将只从安全的角度来讨论约束的使用。

Note 注意：尽管所有的 CONSTRAINT 例子都是作为 ALTER TABLE 给出的，但它们也可以用 CREATE TABLE 语法创建。

诸如 CHECK 约束之类的域完整性约束验证数据具有正确的格式和内容。例如，在 ACME 数据库中，紧接着的 CHK_ADDR_TYPE 约束验证地址是'SHIPPING'类型的还是'BILLING'类型的：

```
ALTER TABLE address
ADD CONSTRAINT chk_addr_type
CHECK (addr_type_s = 'SHIPPING' OR addr_type_s = 'BILLING')
```

现在，任何（合法的或不合法的）输入无效数据（如‘HOME’地址类型）的尝试都会生成一个错误。保护数据维护一致性状态。在数据提交到数据库表之前验证是一种非常有效的安全层次。

Note 注意：如果正试图添加的约束已经存在于数据库中，则上面的例子会生成一个错误。要运行这个例子，可能需要删除这个约束。

强制实施域完整性的另一个机制是 DEFAULT 值。指定该值时，保证如果从查询中省略任何数据，就使用一个默认值代替空格或 NULL。在下面的例子中（Microsoft SQL Server 语法），只有 DEFAULT 值 DF_CUSTOMER_CUST_S 在每个没有提供这个值的插入语句中分配默认值 Y：

```
ALTER TABLE customer
ADD CONSTRAINT df_customer_cust_s
DEFAULT ('Y')
FOR cust_status_s
```

Note 注意：DEFAULT 代表的是一个安全突破口还是一个安全实施机制的讨论是有争议的。一方面，它保证了数据一致性，这是件好事。另一方面，它在数据的入口端的准确度要求较低——通过防止省略/马虎错误，恶意入侵者要插入数据是很容易的。

实体完整性本质上指一个数据行，使用索引和 PRIMARY KEY 或 UNIQUE 之类的约束来维护。它有效地防止用户输入重复的值。例如，在社会保险号（Social Security Number，SSN）列上使用这些约束可以防止不良人员多次利用相同的 SSN 和不同的姓名来收集有价值的信息。

引用完整性维护声明这些完整性的表之间的健全关系。这是“不让一个孩子掉队”政策的 RDBMS 版本。它规定如果在父表中没有与之对应的记录，则在子表中不能有这样一条记录，或者只要在子表中有相应的记录，则不能删除父表中的这条记录。下面是一个利用来自 ACME 数据库的 ADDRESS 表的例子。FOREIGN KEY 约束规定在 ACME 数据库中没有记录，除非它引用了一个 CUSTOMER 表中的有效客户。来自 ADDRESS 表的字段 ADDR_CUSTID_FN 引用 CUSTOMER 表的主键字段 CUST_ID_N，如下所示：

```
ALTER TABLE address
ADD CONSTRAINT fk_addr_cust
FOREIGN KEY(addr_custid_fn)
REFERENCES customer(cust_id_n)
```

约束本身只是保证数据库安全的一个非常有限的部分，因此，应该把它视为对整个 RDBMS 安全

所提供的更健壮机制的补充。

12.7 利用存储过程和触发器实现安全

存储过程和触发器允许非常细粒度的对象级安全。两者都是用某些编程语言实现的编译模块（SQL:2003标准的组成部分，定义架构级例程和模块级例程），可能是内部的，也可能是外部的，并且通常存储在RDBMS服务器内。所有最新版本的RDBMS允许使用通用语言（如Java或C#）创建持久化模块。

利用存储过程实现安全的思想是把特定的业务逻辑封装在持久化模块内部，这些模块存储在服务器端（或由RDBMS管理），并限制用户的数据库通信只能使用这些过程。例如，可以通过这样一种方式来实现一组存储过程，即每个SELECT、UPDATE和DELETE会通过存储过程进行。只能通过存储过程授予用户这些特权，并拒绝直接访问这些存储过程所基于的表。在存储过程的内部，可以实现商业安全规则来控制数据插入、查询、更新或删除的方式。甚至可以使用存储过程创建数据库对象（不过可能有一些特定于实现的限制）。

由标准的普通SQL提供的工具来实现复杂的业务逻辑是不够的。尽管SQL:2003指定了持久的服务器端模块，但是由不同数据库供应商开发的实现细节都远离了标准——包括语法和语言。Oracle使用它自己的针对SQL的PL/SQL过程化扩展。Microsoft SQL Server 2008使用它自己的Transace-SQL语言。IBM使用它自己的SQL PL，有证据表明，它最接近SQL/PSM标准。

Note 注意：Oracle和IBM也允许使用Java来创建存储过程。Microsoft SQL Server 2008支持.NET语言，除了SSIS（SQL Server集成服务）外，.NET语言并不等价于存储过程，可以用于通过.NET Framework和语言来访问和转换数据。

过程化扩展插件（以及Java和Microsoft .NET协议簇）已经超出了本书的范围，本书只提供一些非常基本的用于安全的例子。

下面是一个简单的过程，处理对ACME数据库的CUSTOMER表的插入操作。

Note 注意：这个例子假定表DELINQUENT_CUSTOMER存在，它收集由于未付款而被删除的客户的相关信息。实际的ACME数据库中不包含这样的表。

使用Microsoft SQL Server语法，这个存储过程可以如下实现：

```
CREATE PROCEDURE sp_cust_insert
    @cust_id     INT,
    @cust_paytermsid   INT,
    @cust_salesmanid   INT,
    @cust_status VARCHAR(1),
    @cust_name VARCHAR(50),
    @cust_alias VARCHAR(15),
    @cust_credhold VARCHAR(1)
AS
   IF NOT EXISTS(
SELECT
   cust_id_n
FROM delinquent_customer
WHERE cust_name_s = @cust_name)
INSERT INTO customer
(
    cust_id_n,
    cust_paytermsid_fn,
    cust_salesmanid_fn,
    cust_status_s,
    cust_name_s,
    cust_alias_s,
    cust_credhold_s
)
VALUES
(
    @cust_id,
    @cust_paytermsid,
    @cust_salesmanid,
    @cust_status,
    @cust_name,
```


www.TopSage.com

```
        @cust_alias,
        @cust_credhold
    )
    ELSE
      RAISERROR ('Delinquent customer',19,2)
```

这里的实际情况是调用这个存储过程（sp_cust_insert）的应用程序给它传递给 7 个参数。其中一个参数是**新客户的姓名**。在将数据插入表 CUSTOMER 之前，这个过程检查这个客户是否在欠款客户列表中。如果不在，则允许添加记录。否则，产生一个错误并把这个错误的说明和严重性传回到进行调用的应用程序。

SQL 注入

SQL 注入是一种利用使用动态构建语句的数据库中的弱点进行攻击的技术。这种技术的实质就是通过在 WHERE 子句中提供更宽的选择条件或将另外的 SQL 命令附加到语句末尾的方式来更改基础语句。例如，如果客户端应用程序使用下面语句检查一个特定的 user_ID:

```
SELECT 1 FROM Users WHERE user_ID = '" + <user_ID> + "'"
```

预期结果是如果没有返回任何行，则身份验证失败。现在假设恶意攻击者在 user_ID 字段中输入'somebody' OR 'a'='a',。这会翻译为下面的语句:

```
SELECT 1 FROM Users WHERE user_ID = 'somebody' OR 'a' = 'a'
```

因为并没有用户 ID "somebody"，所以身份验证失败。但是，第二部分，'a'='a'的值为 TRUE，并返回一行。因此，这个用户通过身份验证。

在前面的例子中，攻击者通过输入<user_ID>;<另一个 SQL 语句>也可能迫使服务器执行预期之外的语句。例如:

```
SELECT 1 FROM Users WHERE user_ID = 'somebody' ; DELETE FROM Customer;
```

一旦执行，上面的语句将删除 Customer 表中的全部记录（这要求攻击者具备一些数据库结构的知识或者进行巧妙的猜测）。

可以在很多级别上预防 SQL 注入攻击。第一道防线是分配所需的最小权限集。在上面的例子中，因为用户不会有 CUSTOMER 表上的 DELETE 权限，所以可以防止删除记录。不过，对于第一种情况仍然是无效的。要确保 RDBMS 只执行预期的语句，可以在应用程序层执行检查逻辑，或者借助存储过程（在存储过程主体内执行额外的检查）。此外，RDBMS 可以进行快速解析来检测任何预料之外的布尔表达式。

这个过程使用 Transact-SQL 语言（通常超出了本书的范围）。示例代码的语法尽可能简单，并尽量保持最小化。Oracle 和 IBM DB2 的语法有很大差别，不过思想是一样的。

交叉参考：第 14 章将讨论存储过程、用户函数和触发器。

相同的功能可以作为一个触发器来实现。触发器是一个自动执行的代码模块，用于响应特定事件。在前面的例子中，事件是针对 CUSTOMER 表执行的 INSERT 语句。可以设置一个触发器，只要一个应用程序试图在表中插入、更新数据或从中删除数据，就会触发（执行）这个触发器。

注意：ACME 数据库包含一个针对与安全相关审核而使用触发器的例子——只要发生更新就会触发，并记录进行这些更改的用户数据。

12.8 数据加密

加密是一种把信息从直接可读格式转换为没有解密就不能使用的格式的方法。加密的数据通常可以利用加密时所使用的相同过程（算法）进行解密。加密**不是** SQL 标准的一部分。因此，每个供应商提供了不同的有关的服务。

RDBMS 内部的数据是作为纯文本（ASCII、Unicode）或二进制（BLOBS、IMAGE 和类似的数据类型）保存的。为了防止未授权的用户（正好授予了这个用户访问包含这些数据的表的权限）查看这些数据，或者通过不安全的网络发送数据提取，可以对数据加密。在数据输入数据库之前，也可以通过某些客户端软件对数据进行加密，或者在 RDBMS 内部利用它自己的工具完成。

这提供了一种额外的安全级别，当需要按可读格式（文本或图片、音频文件或可执行文件）查看数据时，用户需要一个密码和解密工具，可以基于 RDBMS，也可以在客户端软件内部。尽管在身份验证、用户访问、公开密钥基础结构等方面维护高度安全，但 Oracle 11g 在数据库自身内部没有提供更多可供用户访问的加密功能，不过它利用附加产品进行补偿。唯一可以利用 RDBMS 提供的功能进行加密的工具是 PL/SQL 代码，它们包含在 Oracle 的包规范和包主体中并使用实用工具 **wrap.exe**（位于 UNIX 上的$ORACLE_HOME/bin 目录中和 Windows 机器上的\Oracle11\bin 目录中）的独立过程和函数。Oracle 11g 还提供了一个混淆包（DBMS_OBFUSCATION_TOOLKIT），通过把代码转换为 ASCII 乱码来隐藏源代码和数据。它使用 DES 实现算法。加密与混淆不同，前者具有更高的安全性并且不受可读的字符范围的限制。Oracle 总是在通过网络发送密码之前使用修改过的 AES（Adanced Encryption Standard，高级加密标准）算法对其加密。

注意：此外，当使用 ENCRYPT 子句创建或修改表时，可以对 Oracle 表中的数据加密，这样就可以使用明确的数据加密功能对正在定义的列加密。可以加密 CHAR、NCHAR、VARCHAR2、NVARCHAR2、NUMBER、DATE 和 RAW 类型的列。有效的加密算法有 3DES168、AES128、AES192 和 AES256。如果忽略这个子句，则数据库使用 AES192。请参考第 4 章了解更多信息。

IBM DB2 9.5 提供了几个用于数据加密的函数（在表 12-14 中列出）。例如，下面这个查询返回 ACME 数据库的 PRODUCT 表的产品商标：

```
SELECT prod_brand_s
FROM   product
PROD_BRAND_S
---------------
SPRUCE LUMBER
STEEL NAILS
```

表 12-14　　IBM DB2 加密函数

函数	说明
ENCRYPT(<*data to encrypt*>,[<*password*>],[<*hint*>])	使用一个密码——CHAR 或 VARCHAR 字符串（至少 6 字节，不超过 127 字节长）加密 CHAR 或 VARCHAR 数据（最多 32633 字节长）。HINT（CHAR 或 VARCHAR，最多 32 字节长）是一个可选参数；如果使用它，可以通过 GETHINT 函数利用提示表达式提供恢复密码的功能
DECRYPT_BIN(<*encrypted data*>,[<*password*>])	解密用 ENCRYPT 函数加密的二进制数据（BLOB、CLOB 等）
DECRYPT_CHAR(<*encrypted data*>,[<*password*>])	解密用 ENCRYPT 函数加密的字符数据
GETHINT(<*encrypted data*>)	如果找到用于加密数据的提示，则这个函数返回这个提示。用户利用这个提示可以恢复密码

要生成加密的数据，例如，来自 PRODUCT 表的商标列表（要通过不安全的网络发送到某个部门），在 IBM DB2 中，可以使用下面的 SQL 语句：

```
SELECT ENCRYPT(prod_brand_s, 'PASSWORD') encrypted
FROM   product

ENCRYPTED
----------------------------------------------------
x'00E61AFFE404A6D596757C7CC7AC70467884E127B6A50726'
x'00DC24FFE404A0D5F736C8A4156922A6709DD5D609EBE762'
```

要解密上面这些看上去无意义的字符串，可以使用 DECRYPT_CHAR 函数（因为我们正在使用字符数据），使用完全相同的密码。这会把数据还原到它的最初形式。数值数据不能直接使用这个函数进行加密（也可以对数字的二进制或字符表示形式进行加密）。

注意：将加密密码设置为环境变量（而不是将它传递到函数中）是可行的。其语法是 SET ENCRYPTION PASSWORD<string constant|host variable>，其中 **host variable** 必须是 6 到 127 个字节的 CHAR/VARCHAR 数据类型。在会话期间这个密码将是有效的。

Microsoft SQL Server 2008 允许加密（或默认加密）下面的内容：

- 登录和应用程序角色密码（存储在服务器端）；

www.TopSage.com

- ✦ 存储过程体（实际的实现代码）；
- ✦ 用户定义函数体（实际的实现代码）；
- ✦ 视图定义（实际的 SQL 语句）；
- ✦ 触发器（实际的实现代码）；
- ✦ 规则和默认定义；
- ✦ 在 SQL Server 和客户端应用程序之间发送的数据包。

登录和密码存储在 Microsoft SQL Server 2008 系统表中并且始终加密。其使用的算法是专用的，而且不能直接查看密码（除非使用 **NULL** 值作为密码）。

当一个存储过程、函数、视图或触发器被编译并保存在 SQL Server 中时，创建者可以选择对实际的实现代码进行加密，从而防止其他用户或拥有数据库系统对象的访问权限的第三方进行查看。这个加密选项在 CREATE 语句中。例如，要对 Microsoft SQL Server 2008 中的一个 ACME 数据库视图进行加密，可以使用下面的语句：

```
CREATE VIEW v_customer_status
(
  name,
  status
)
WITH ENCRYPTION
AS
SELECT cust_name_s,
       cust_status_s
FROM   customer
```

WITH ENCRYPTION 选项采用加密形式而不是标准的纯文本把实际的 Transact-SQL 代码保存在 SYSCOMMENTS 系统表内。加密 SQL Server 对象有一个问题——一旦加密，就不能修改这个对象；如果需要修改，则必须删除这个对象并重新创建。

注意：实际上有多达几百种数据加密算法——包括自定义和公共的。设计并实现加密算法除了要有高等数学知识外，还需要熟悉编程原理。以下是目前流行的算法：DES（由 IBM 于 1970 年设计，由 NIST 在 1976 年用于处理未分类数据）、RC5（来自 RSA Data Security）、CMEA（由电信行业协会开发，用于加密数字蜂窝电话数据）、FEAL（由日本电话电报开发）、TEA、MD5、Tiger 和 CAST，这里只列出的只是其中很少一部分。

如果选择 TCP/IP 作为通信协议（通常是大多数网络和 Internet 连接的情况），可以利用安全套接层（Secure Socket Layer，SSL）对在 SQL Server 2008 和客户端应用程序之间发送的数据进行加密。当使用多个协议时，必须有个应用程序专门调用 Windows PRC 加密 API（应用程序编程接口）。这种加密的实际强度（密钥的长度）取决于安装了这个软件的 Windows OS 的版本。

注意：SSL（安全套接层）是一种通信协议（最初由 Netscape 开发），确保在 Internet 上安全地传输文档。它使用公钥加密数据。按照惯例，SSL 连接 Internet 的链接以前缀 https://开始，而不是标准的 http://（Hypertext Transfer Protocol，超文本传输协议）。

Microsoft 提出了具有其 2005 版本中引入的强加密功能的一些特性。表 12-15 显示了成对出现的函数提供了加密和解密服务（参看本章前面段落中有关 USER 定义的内容，了解有关安全证书和密钥的信息）。

表 12-15　　Microsoft SQL Server 2008 加密函数

函数	说明
EncryptByKey	使用一个对称密钥加密数据（返回 VARBINARY，最多 8 000 个字节）
EncryptByPassPhrase	使用用户提供的密码加密数据（返回 VARBINARY，最多 8 000 个字节）
DecryptByKey	使用对称密钥解密数据（返回 VARBINARY，最多 8 000 个字节）
DecryptByPassPhrase	使用最初加密所用的密码短语解密数据（返回 VARBINARY，最多 8 000 个字节）
Key_ID	返回当前数据库中对称密钥的 ID（按照密钥名，返回整数）
Key_GUID	返回在数据库中定义的对称密钥的 GUID（Global Unqiue Identity，全局统一标识符）（返回 VARBINARY）
EncryptByAsmKey	使用非对称密钥加密数据（返回 VARBINARY，最多 8 000 个字节）

www.TopSage.com

续表

函数	说明
EncryptByCert	使用在数据库中创建的一个证书的 PUBLIC 密钥加密数据（返回 VARBINARY，最多 8 000 个字节）
DecryptByAsmKey	使用非对称密钥解密数据（返回 VARBINARY，最多 8 000 个字节）
DecryptByCert	使用数据库中创建的证书的 PRIVATE 密钥解密数据（返回 VARBINARY，最多 8 000 个字节）
Get_ID	返回在数据库中定义的证书的 ID（返回整数）
AsymKey_ID	返回在数据库中定义的非对称密钥的 ID（返回整数）
CertProperty	返回指定的证书的属性值
SignByAsymKey	使用非对称密钥签署纯文本（返回 VARBINARY，最多 8 000 个字节）
SignByCert	使用证书签署纯文本并返回签名（VARBINARY，最多 8 000 个字节）
VerifySignedByAsmKey	测试从数字签署数据（使用非对称密钥）后数据是否改变——如果签名匹配则返回 1，否则返回 0
VerfySignedByCert	测试从数字签署数据（使用证书）后数据是否改变——如果签名匹配则返回 1，否则返回 0
DecryptByKeyAutoCert	使用非对称密钥解密数据，使用证书自动解密（返回 VARBINARY，最多 8 000 个字节）

这些加密函数提供了行业级强度的加密，只要数据安全需要，就应当使用。下面的例子使用非对称密钥（加密和解密使用相同的密码）加密一个社会保险号（Social Security Number，SSN）；假设已经创建了相应的证书和非对称密钥：

```
OPEN SYMMETRIC KEY ssn_key DECRYPTION BY CERTIFICATE hr_cert
   INSERT employees (f_name, l_name, ssn)
        VALUES ('Alan','Parsons',
EncryptByKey(Key_GUID ('ssn_key','555-55-5555')
```

注意，加密字符串所在的列必须声明为 VARBINARY 数据类型才能符合 EncryptByKey 函数的返回值。

Note　*注意：应当基于特定的安全约束和性能需求决定使用对称还是非对称密钥加密。对称密钥解密和加密比较快，并且需要较少维护；非对称密钥提供最强的加密，但代价是性能下降。Microsoft 推荐使用强对称密钥加密数据，并使用非对称密钥加密这个对称密钥。*

可以利用 Microsoft SQL Server 2008 的位运算符 XOR（逻辑上与 OR 互斥）通过 Transact-SQL 实现非常简单的基于密码的数据加密。这种类型的加密非常容易实现，但也容易破解。

12.9　数据库审核

审核提供跟踪数据库内部信息流的能力，包括连接尝试、数据更新、删除、插入和选择、执行功能等。这对于事后分析和实时监视以检测未授权活动都是很有用的。

审核与 SQL 标准无关，并且在功能、实现细节等方面严格依赖供应商。下面的内容概述 RDBMS 审核。

Oracle 11g 允许选择操作系统审核线索或数据库审核线索或两者都选。当然，第一个选项依赖于操作系统，并只包含那些对 OS 进行编程而保护的信息。由 Windows 生成的审核线索与基于 UNIX 逻辑生成的有很大区别，即使 Oracle 数据库设置完全相同。

不管在什么 OS 上安装 Oracle，数据库审核线索都完全相同。另一个优势是能够利用 Oracle 内置工具产生审核报告。在 Oracle 中，审核信息（数据库语句、特权等）存储在 SYS.AUD$目录表中，通常作为**审核追踪信息**被引用。本质上，由您自行选择希望监视的事件。在 Oracle 中，默认情况下会审核若干事件：实例启动、实例停止以及使用管理特权连接数据库的尝试。使用特定于 Oracle 的 AUDIT 语句可以选择指定自定义审核选项来监视在 Oracle 安装中发生的其他事件。表 12-16 显示了 Oracle 11g 的审核级别。

表 12-16　Oracle 11g 的审核级别

级别	操作
STATEMENT	启动一些特定 SQL 语句的审核。例如，AUDIT TABLE 命令启动对 CREATE TABLE、TRUNCATE TABLE、COMMENT ON TABLE 和 DELETE [FROM] TABLE 语句的审核

www.TopSage.com

TopSage.com

续表

级别	操作
PRIVILEGE	启动对使用特定的系统特权创建的 SQL 语句的审核。例如，AUDIT CREATE USER 将监视使用这个特定的系统特权提交的所有语句
OBJECT	启动对属于特定对象（如 TABLE 或 VIEW）的事件的审核

在设置审核级别时，也可以指定一些选项，缩小希望监视的事件范围（参见表 12-17）。

表 12-17 Oracle 11g 的审核级别选项

级别	说明
BY SESSION/BY ACCESS	第一个选项累积在会话期间提交的所有的 SQL 语句信息，第二个选项令 Oracle 为每个访问写一条记录
WHENEVER SUCCESSFUL	只记录成功语句的审核信息
WHENEVER NOT SUCCESSFUL	只记录失败或产生错误的语句的审核信息

例如，要审核执行多种数据库操作的所有失败尝试，可以使用下面的语句：

```
AUDIT SELECT TABLE,
      INSERT TABLE,
      DELETE TABLE,
      EXECUTE PROCEDURE
BY ACCESS
WHENEVER NOT SUCCESSFUL;
```

一旦执行这个语句，Oracle 就开始收集所有尝试执行审核子句中列出的操作而执行失败的信息。参考 Oracle 11g 文档了解有关 RDBMS 审核功能的更多详细信息。

Tip 提示：记住，累积的信息量很容易超过合理的限制。要明智地选择希望监视的事件以及监视多久。

IBM DB2 引入了审核工具 db2audit 来监视数据库事件，并使用日志记录收集的信息。审核是在实例级别执行的，而且这个工具的用户必须具有 SYSADM 权限。审核追踪信息是针对一系列预定义的事件生成的。针对几类事件可能希望生成一个审核追踪信息；可以监视到其中每个事件的失败、成功或两者。表 12-18 中列出了这些类别。

表 12-18 可用于审核的 IBM DB2 事件类别

类别	说明
AUDIT	无论何时改变审核设置都生成一条日志记录
CHECKING	当执行授权检查时生成一条日志记录
OBJMAINT	当创建或删除数据库对象时生成一条日志记录
SECMAINT	当授予或撤消特权和权限时生成一条日志记录；同样，当改变安全配置参数时也生成一条日志记录
SYSADMIN	当执行一个需要高级权限的操作时生成一条日志记录（例如，STAERT_DB2、CREATE_DATABASE 等）
VALIDATE	当对一个用户进行身份验证或获取他/她的安全信息时生成一条日志记录
CONTEXT	生成操作环境的日志记录；当分析其他事件记录时可能会有帮助。由于其自身特点，这样一条记录可能非常大，因此应该谨慎使用（例如，CONNECT、BACKUP_DB 等）

按预定义的结构将收集到的信息写入日志文件。每一个事件类别生成自己的文件，并具有自己的结构。这个文件是常规的 ASCII 文件，针对载入到表进行了优化；这个文件没有加密，但在操作系统的安全框架内受到保护。装载也通过 **db2audit** 实用工具进行，其中使用 **extract** 参数作为变量。

审核工具从命令行运行并接受大量参数（20 多个）。下面是一个非常简单的例子：

```
db2audit start checking both
```

随即开始审核所有属于 CHECKING 类别的事件，包括失败的和成功的事件。

审核工具非常复杂，简短说明其用法也需要专门的一章；参考 IBM DB2 文档来获得更多的信息。

Microsoft SQL Server 2008 提供了一个 SQL Profiler 工具，可以用于审核，并且像在 Windows 环境中期待的那样，它有可视化界面。从 SQL Server 企业管理器的“工具”菜单可以调用它。不过，只允许 SYSADMIN 安全固定角色的成员启用它；对于所有其他用户，这个选项是不可用的。

SQL Profiler 为审核事件提供了可视化界面。这些事件属于几个类别：用户活动、数据库管理活动、

www.TopSage.com

服务器事件等。每一个类别代表一个单独选择或一起选择的事件集合。表 12-19 中显示了可用于监视的事件类别。

表 12-19 Microsoft SQL Server 2008 事件类别

事件类别	说明
CURSOR	由游标操作生成的事件
DATABASE	当扩展/收缩发生时，由数据和日志文件生成的事件
ERRORS AND WARNINGS	由进程中出现的异常生成的事件
LOCKS	在操作期间发生的锁定生成的事件
OBJECTS	由对象的创建、删除、打开等操作生成的事件
PERFORMANCE	与 DML（数据操作语言）执行有关的事件的集合
SCANS	在一个数据库对象（如表或索引）上执行的扫描而生成的事件
SECURITY AUDIT	由与安全有关的操作（例如，授予特权）生成的事件
SESSIONS	由连接到 SQL Server 和从 SQL Server 断开连接而生成的事件
STORED PROCEDURES	在 SQL Server 中执行存储过程而生成的事件
TRANSACTIONS	Microsoft 分布式事务处理协调器（Microsoft Distributed Transaction Coordinator，MS DTC）执行事务或写入到事务日志而生成的事件
TSQL	执行从客户端传递到 SQL Server 实例的 Transact SQL 语句而产生的事件
USER CONFIGURABLE	允许用户指定自定义事件

记录的信息包含日期/时间戳、用户 ID、事件类型、结果（成功/失败）、源（机器名、IP 地址）、被访问对象的名称以及 SQL 语句的全文。所有这些信息都记录在审核文件中，并处于操作系统的安全保护之下。

SQL Server 为每一种事件类别都提供了细粒度的监视能力（例如，事件出现所需的 CPU 时间）。此外，还可以指定筛选器、状态，例如，只希望监视由某个特定应用程序生成的选中的事件，而忽略其他的事件。

Note 注意：Microsoft SQL Server 支持 C2 审核需求（在本章的后面可以看到有关 C2 级安全认证的更多信息）。要启动 C2 审核选项，SYSADMIN 固定角色的某个成员必须运行系统存储过程 sp_configure，并将 c2 审核模式选项设置为 1：

```
EXEC sp_configure 'c2 audit mode','1'
```

要访问高级 audit mode 选项，首先运行下面这个脚本：

```
EXEC sp_configure 'show advanced options', '0'
RECONFIGURE
```

启用这个选项将打开一个安全追踪模板，它收集 C2 级安全审核所需的全部信息，就像由安全标准指定的那样。

12.10 安全标准

尽管与 SQL 不是直接相关，但安全标准定义使用它的基础框架，因此对 SQL 的用户也比较重要。通常，RDBMS 软件在一定程序上符合这些标准——或者自愿，或者迫于政府机构的压力，为了软件能被接受，必须满足政府规定的要求。

第一次使计算机系统的安全程序标准化的全国性尝试是 1985 年由美国国家计算机安全中心（U.S.National Computer Security Center，NCSC）开始进行的。为了能被政府合同纳入考虑范围，供应商必须实现某种安全级别使其产品通过监督测试。很多供应商历经数年测试过程（这个过程平均要花费三年时间），只为能够把它们的产品卖给政府机构。像 Sun、Oracle 和 Novell 这样的供应商在 20 世纪 90 年代初期得到了他们的认证（C1 或 B2），紧接着要求所有存储敏感信息的计算机系统必须通过 C2 认证。

什么是 C2 安全级别？

C2 等级是由美国国家计算机安全中心（NCSC）建立的安全等级。它被授予那些通过了可信计算机系统评估标准（Trusted Computer System Evaluation Criteria，TCSEC）测试（也就是橙皮书）的

www.TopSage.com

产品，这个测试由美国国防部主持。对于要在政府机构或部门用于收集和处理敏感安全信息的产品来说，这个等级是必需的最低安全级别。

TCSEC 标准于 1985 年建立，并从那时起经历了多次更新。系统安全根据 TCSEC 被评为多个级别，从 A1 等级到 D 等级。

D 等级定义为**最低安全**；本质上意味着——“听天由命”。

C1 等级定义为**自定义安全保护**（Discretionary Security Protection）；评估为这个级别的系统通过控制用户对数据的访问来满足安全需求。

C2 等级定义为**受控访问保护**（Controlled Access Protection）；通过增加额外的可说明功能对 C1 等级进行补充，例如登录过程、验证所有用户的操作（例如，访问、读取、写入或删除任何对象的尝试）的审核能力、细粒度的访问特权等。

B1 等级定义为**标签安全保护**（Labeled Security Protection）；处于这个级别的系统必须具有一个规定的策略模型和专门标注的数据。

B2 等级定义为**结构化保护**（Structured Protection），为 B1 需求添加了更多明确的和正式的安全策略。

B3 等级定义为**安全域**（Security Domains），添加了严格的工程和监视要求。

A1 等级定义为**验证的设计**（Verified Design）；处于这个级别的系统在功能上等价于 B3 系统，但除了所有前面各级别的全部功能外，它们还必须接受正式的功能分析以确保安全。

12.10.1 国际安全标准

BS7799 及其国际等效标准 ISO 17799 是得到世界最广泛认可的安全标准。在美国，与其最接近的是 B1 安全级。

ISO 17799 在多个方面提供了详细指导，并且要求其产品执行这个标准的公司必须处理下面所有方面。

- **商业连续性规划**：规定不管主要系统失败还是灾难后果都能继续商业活动。
- **系统访问控制**：关注对信息的访问控制，确保网络服务的保护，检测和阻止未经授权的活动，确保分布式移动应用程序的信息安全。
- **系统开发和维护**：规定内置安全（而不是在外部）；预防数据丢失和滥用，以及处理信息的机密性、真实性和完整性。
- **物理和环境安全**：处理防止对高层商业基础和信息的未经授权的访问、损害和干扰，防止信息和信息处理工具的丢失、损害或失窃。
- **遵从性**：避免违反任何刑法或民法、各种法规和规章制度以及合同义务；确保组织机构中的每一个系统都遵从已确立的组织安全策略和标准；最小化审核程序与商业实践的冲突。
- **人员安全**：减小人为错误、偷窃、欺诈或工具滥用所导致的风险，万一发生这样的事故则最小化造成的损害；对用户进行正确的策略程序教育。
- **安全组织**：管理组织中的信息安全；维护第三方对组织的工具进行访问的安全，例如，当信息保护的职责外包给第三方时。
- **计算机和操作管理**：管理工具的操作策略；确保网络和支撑基础框架的信息安全，防止组织间的交换数据的丢失、滥用或未经授权的修改。
- **资产分类和控制**：维护公司资产保护。
- **安全策略**：在组织中建立和管理一个可行的安全策略。

尽管制定了详细的标准，但这些标准的实际实现在不同组织间有很大区别。造成这些不同的原因之一是标准太多；而且认证过程非常昂贵，对许多公司而言不是一个可行选择。例如，美国的大多数银行不使用 ISO 标准，而是依靠 SAS 70 审核标准，而其他公司则更喜欢采用 ISO 9000/2000 标准。

Note 注意：有关信息系统安全的更多内容可以访问 www.infosyssec.com、www.firstgov.gov、www.sas70.com 和 http:csrc.nist.gov/。

www.TopSage.com

也有一些新兴的标准，比如公共标准（Common Criteria，CC）程序。这个程序最初在英国、德国、法国 荷兰以及来自国家信息保证联盟（National Information Assurance Partnership， NIAP）的强烈支持下于1996年启动。自那以后，又有18个国家加入了这个程序：澳大利亚、新西兰、加拿大、捷克共和国、丹麦、芬兰、希腊、匈牙利、印度、以色列、意大利、马来西亚、挪威、新加坡、西班牙、瑞典和土耳其。更多信息请访问 www.niap-ccevs.org。

在2002年7月，国家安全局（National Security Agency， NSA）开始要求所有新的国家安全系统（包括RDBMS软件）必须通过由CC要求的严格测试；也表明这可能会扩展到每个政府组织。

Note 注意：通常数据库供应商都要通过C2级认证。至于公共标准程序，只有Oracle的产品通过了EAL4 CC认证级别。Microsoft SQL Server 2008获得了NSA的C2级安全认证，IBM DB2也通过了认证。

12.10.2 规章制度的遵从性

在我们的互联世界中，安全、隐私和机密性正在变得越来越重要。我们需要知道维护何种信息以及由谁维护；采取什么步骤来保护这些信息；以及出现违反的情况时公司的责任是什么。引入了一些法律根据常识来定义这些问题，有一些是知名的犯罪调查案的结果（例如，Enron）。

在RDBMS中，这些问题主要在幕后处理，开发人员很少需要特定法律规章制度遵从性的本质细节。不过，理解一些基本知识有助于设计可靠的、健壮的和安全的数据库应用程序，其中SQL是重要组成部分。表12-20列出了美国采用的一些主要规章制度的遵从法案。

表12-20 规章制度遵从法案

法案/法律	规章制度领域
Sarbanes-Oxley	公开交易的公司维护财务记录的完整性和隐私
Health Insurance Portability and Accountability（HIPAA）	卫生保健提供程序维护的信息的完整性、隐私（机密性）和可用性
支付卡工业（Payment Card Industry，PCI）数据安全标准	供应商维护信用卡信息的机密性
Gramm-Leachy Bliley(GLBA)	财务机构维护的信息的完整性、隐私（机密性）和可用性
SB 1386	在加利福尼亚州进行商业活动的组织存储的信息的完整性、隐私（机密性）和可用性
BASEL II	财务机构维护和传递的财务信息的完整性、隐私（机密性）和可用性；涵盖财务事务的完整性和安全性

Sarbanes-Oxley，404节紧跟着公司财务丑闻（例如，Enron）之后被引入并在2002年签署为法律。主要目的是在公开交易的公司财务报告的透明度上给予投资者更大信心。Sarbanes-Oxley（即熟知的SOX）是IT安全背后的重要驱动力，它要求管理层负责财务信息的安全和完整性。SOX审核过程集中于以下方面。

- 机密性：没有适当授权不能访问机密信息。
- 完整性：软件需要提供没有按照正确过程修改/变更数据的证据。
- 可用性：经授权的个人必须无障碍地访问信息。
- 访问控制：通过角色和帐户的受控访问权限管理。
- 审核和日志记录：对包含敏感信息的系统中的事件进行日志记录和审核。
- 变更管理：如果系统包含财务数据流，则对流过程的任何更改都必须进行控制并记入日志。

作为响应，RDBMS供应商集成了很多额外的安全功能，使公司用于确保SOX遵从性。

在1996年，美国国会通过了《健康保险便利及责任性法案》（Health Insurance Portability and Accountability，HIPAA）。通过卫生保健提供程序，它建立了联邦法律，控制电子保护健康信息（electronic Protected Health Information，ePHI）的处理，例如医疗记录和个人信息。这样，在管理（正确流程）、物理（物理访问）和技术层次（系统架构、软件设计和实现）上必须保护信息。其余的需求与SOX法案要求的内容非常类似。

支付卡工业（Payment Card Industry，PCI）数据安全标准是由VISA和MasterCard这样的主要信

www.TopSage.com

用卡公司创建的。PCI 建立了针对存储、传输和处理来自卡持有者的信息的安全标准。它主要涉及机密性和授权，以及日志记录和审核。

SB 1386 是加利福尼亚州的法律，对在加利福尼亚州进行商业活动的公司施加影响；它保证客户信息具有机密性，例如社会保险号、驾驶执照号码、银行账户号码等。

BASEL II 是由 Basel 委员会建立的一个框架，这个委员会是有多个国家的中央银行组成的联盟。它主要涉及衡量银行资本生存能力的国际标准并且针对银行专业人士。要求防止不恰当的信息泄露并保护免遭拦截和干扰，以及防止由于硬件或软件失效而引起的服务中断。

12.11 小结

SQL 本身只提供了有限的安全机制。本质上，它使用 GRANT 和 REVOKE 语句通过系统特权来控制对数据库对象的访问。关系数据库系统需要更健壮的安全性，RDBMS 供应商以多种非标准的方式实现了安全性。

有几种不同的安全宏观层次：身份验证、授权和审核。也有用于在最基本级别保护数据的不同技术。

所有 RDBMS 都把用户的概念看作是一个连接到数据库并执行操作的实体。本书中讨论的所有三大供应商都通过某种方式实现了角色，角色是一种管理特权集的方法。角色可以是系统定义的（固定的）或用户定义的。

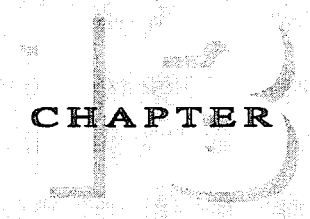
用户可以通过 RDBMS 本身或者安装了 RDBMS 的操作系统进行身份验证。一旦通过验证，用户就可以对数据库对象执行已授权的操作。通过一个**特权**系统来处理授权。

使用 GRANT 或 REVOKE 可以使权限（特权）分配或拒绝分配给用户或角色，并使用一些规则来控制这个过程。

通过数据库自身提供的多种机制可以实现额外的安全性：约束、视图、存储过程和触发器。数据本身具有最低级别的防御，通过加密将数据显示为不可读形式。

推荐（但非必需）使用一些国家和国际安全标准；一些数据库供应商选择其中一些获得认证，而另一些则没有选择认证。已经制定了一些法律用于处理安全问题；本书讨论的最新版本的 RDBMS 已经实现了很多功能来帮助用户遵守这些规章制度。

www.TopSage.com



第 13 章
系统目录和 INFORMATION_SCHEMA

本章提要：

- SQL:2003 信息架构
- Oracle 数据字典
- IBM DB2 系统目录
- Microsoft SQL Server 信息架构
- Microsoft SQL Server 系统存储过程

为了跟踪所有对象及它们的关系等，RDBMS 使用它们一贯提倡的技术（一组关系表和视图）。这一方法最初是在 SQL:1992 标准（ISO/IEC 9075-2:199x）中定义的，并且所有主要的 RDBMS 都在一定程度上实现了此标准。最新的更新体现在 ISO/IEC 9075-11 中，它定义了两种架构——INFORMATIOn_SCHEMA 和 DEFINITION_SCHEMA；前者提供了有关持久化数据库对象（表、过程等）的信息，后者则包含有关 DEFINITION_SCHEMA 本身的信息（数据模型）。

13.1 SQL 系统目录

在 SQL 标准中，CATALOG 是一个架构集合，其中包含 INFORMATION_SCHEMA 等内容。它由提供了数据库中定义的所有其他对象和记录（架构、表、特权等）的全部信息的表和视图组成。最新的标准也包括结构和完整性约束信息，以及 SQL 数据的安全性和授权规范。主要的思想是为用户和 RDBMS 提供访问元数据（有关数据的数据：表定义、用户定义类型等）以及某些系统信息的一致的标准化方法。根据定义，INFORMATION_SCHEMA 表和视图不能直接更新，不过有些供应商允许这么做（例如，IBM DB2）。按照 SQL 标准的规定，INFORMATION_SCHEMA 包含视图，而对应的 DEFINITION_SCHEMA 假定是作为基础表实现的（例如，持久化结构）。

表 13-1 列出了 SQL:2003 标准中指定的 INFORMATION_SCHEMA 视图。

表 13-1　SQL:2003 标准的 INFORMATION_SCHEMA 视图

INFORMATION_SCHEMA 视图	说明
ASSERTIONS	列出数据库中创建的所有断言；没有被任何主要供应商实现
ATTRIBUTES	列出某给定用户/角色可以访问的目录中定义的用户定义类型的属性
CHARACTER_SETS	说明用户/角色可访问的字符集定义；每个集合占用一行
CHECK_CONSTRAINTS	说明用户/角色可访问的表的检查约束；每个约束占用一行

www.TopSage.com

TopSage.com

续表

INFORMATION_SCHEMA 视图	说明
CHECK_CONSTRAINTS_ROUTINE_USAGE	说明由某给定用户/角色拥有的每个 SQL 调用的例程，在这个目录中定义的域约束、表检查约束或断言依赖于此
COLLATIONS	说明用户/角色可访问的排序规则；每个排序规则占用一行
COLLATION_CHARACTER_SET_APPLICABILITY	说明每个排序规则可应用的字符集
COLUMNS	说明当前用户对数据库中的每一个表可访问的列；每个列占用一行
COLUMN_COLUMN_USAGE	说明每一种这样的情况：其中一个生成的列依赖于由给定用户/角色拥有的基础表中的一个基础列
COLUMN_DOMAIN_USAGE	包含当前用户对其拥有权限的对象的信息
COLUMN_PRIVILEGES	说明授予用户/角色的列级特权；每个列的每个特权占用一行
COLUMN_UDT_USAGE	说明已定义的列，它依赖于在这个目录中定义的一个用户定义类型并且由一个给定用户/角色拥有
CONSTRAINT_COLUMN_USAGE	说明在每一个约束中引用的列；每个列占用一行
CONSTRAINT_TABLE_USAGE	说明在每一个约束中引用的表；每个约束的每个表占用一行
DOMAINS	说明用户/角色可访问的域（数据类型、限制等）；每个域占用一行
DATA_TYPE_PRIVILEGES	说明包含在数据类型描述符中并且某给定用户/角色可以访问的架构对象
DIRECT_SUPERTABLES	说明与定义在这个目录中并且由某给定用户/角色拥有的表有关的直接超表
DIRECT_SUPERTYPES	说明与定义在这个目录中并且由某给定用户/角色拥有的表有关的直接超类型
INFORMATION_SCHEMA_CATALOG_NAME	针对用户/角色的数据库名称；每个名称占用一行；没有被任何主要供应商实现
DOMAIN_CONSTRAINTS	说明用户/角色可访问的域约束；每个域约束占用一行
ELEMENT_TYPES	说明在这个目录中定义的集合元素类型，允许给定的用户/角色访问
ENABLED_ROLES	说明当前 SQL 会话已启用的角色
FIELDS	说明在这个目录中定义的字段类型，允许给定的用户/角色访问
KEY_COLUMN_USAGE	说明在基于键的约束（主键、外键、唯一键等）中使用的列；每个约束占用一行
METHOD_SPECIFICATION_PARAMETERS	说明在 METHOD_SPECIFICATIONS 视图中描述的方法规范的 SQL 参数，给定用户/角色可以访问
METHOD_SPECIFICATIONS	说明目录中 SQL 调用的方法，给定用户/角色可以访问
PARAMETERS	说明在这个目录中定义的 SQL 调用例程的 SQL 参数，允许给定用户/角色访问
REFERENTIAL_CONSTRAINTS	说明用户/角色可访问的表的外键约束；每个约束占用一行
REFERENCED_TYPES	说明在这个目录中定义的引用类型的的已引用类型，允许给定用户/角色访问
ROLE_COLUMN_GRANTS	说明在这个目录中定义的可用的或者由当前已启用角色授予的列特权
ROLE_ROUTINE_GRANTS	说明在这个目录中定义的可用的或者由当前已启用角色授予的 SQL 调用例程的特权
ROLE_TABLE_GRANTS	说明在这个目录中定义的可用的或者由当前可应用角色授予的表特权
ROLE_TABLE_METHOD_GRANTS	说明在这个目录中定义的可用的或者由当前已启用角色授予的结构化类型的方法上的特权
ROLE_USAGE_GRANTS	说明在这个目录中定义的可用的或者由当前已启用的角色授予的在对象上的 USAGE 特权
ROLE_UDT_GRANTS	说明在这个目录中定义的可用的或者由当前已启用角色授予的用户定义类型上的特权
ROUTINE_COLUMN_USAGE	说明由给定用户/角色拥有的并且在这个目录中定义的 SQL 例程所依赖的列
ROUTINE_PRIVILEGES	说明在这个目录中定义的可用的或者由当前已启用角色授予的用 SQL 调用例程的特权
ROUTINE_ROUTINE_USAGE	说明由给定用户/角色拥有的并且在这个目录中定义的 SQL 例程所依赖的每个 SQL 调用例程
ROUTINE_SEQUENCE_USAGE	说明由给定用户/角色拥有的并且在这个目录中定义的某个 SQL 例程所依赖的每个外部序列生成器
ROUTINE_TABLE_USAGE	说明由给定用户/角色拥有的并且在这个目录中定义的 SQL 例程所依赖的表
ROUTINES	说明在这个目录中的 SQL 调用例程，并且允许给定用户/角色访问
SCHEMATA	说明包含在数据库中的架构；每个架构占用一行
SEQUENCES	说明在这个目录中定义的外部序列生成器，并且允许给定用户/角色访问

www.TopSage.com

续表

INFORMATION_SCHEMA 视图	说明
SQL_FEATURES	列出这个标准的功能和子功能，并且指明 SQL 实现支持其中的哪一个
SQL_IMPLEMENTATION_INFO	列出在这个标准中定义的 SQL 实现信息项，并且对于其中每一个都表明 SQL 实现支持的取值
SQL_LANGUAGES	包含 RDBMS 支持的有关语言的信息；标识符合程度、选项和 SQL 实现处理这个目录中定义的数据所支持的用语
SQL_PACKAGES	列出这个标准的包，并指明 SQL 实现支持其中哪一个
SQL_PARTS	列出这个标准的组成部分，并指明 SQL 实现支持其中哪一个
SQL_SIZING	列出在这个标准中定义的改变大小的项，并且对于其中每一个都指明 SQL 实现所支持的大小
SQL_SIZING_PROFILES	列出在这个标准中定义的改变大小的项，并且对于其中每一个都指明一个或多个标准概要中需要的大小
TABLES	说明用户/角色可访问的每一个表；每个表/视图占用一行
TABLE_METHOD_PRIVILEGES	说明在那些目录中定义的可用的或者由给定用户/角色授予的针对结构化类型的表方法的特权
TABLE_CONSTRAINTS	说明为表声明的约束（主键、检查约束等）；每个约束占用一行
TABLE_PRIVILEGES	说明授予用户/角色的所有特权；每个特权占用一行
TRANSFORMS	说明在这个目录中定义的用户定义类型的转换，并且给定用户/角色可以访问
TRANSLATIONS	翻译用户/角色可访问的定义
TRIGGERED_UPDATE_COLUMNS	说明来自一个 UPDATE 触发器的触发事件的显式列的列表的列，并且给定用户/角色可以访问
TRIGGER_COLUMN_USAGE	说明在这个目录中定义的并且由给定用户拥有的某个触发器上的列，取决于因为在搜索条件中引用它们还是因为它们出现在触发器的主体中
TRIGGER_ROUTINE_USAGE	说明给定用户/角色拥有的每个 SQL 调用的例程，在这个目录中定义的某个触发器依赖于它
TRIGGER_SEQUENCE_USAGE	说明给定用户/角色拥有的每个外部序列生成器，在这个目录中定义的某个触发器依赖于它
USAGE_PRIVILEGES	包含有关授予一个用户/角色的特权信息；每个特权占用一行
TRIGGER_TABLE_USAGE	说明在这个目录中定义的并且由某给定用户/角色拥有的触发器所依赖的表
TRIGGERS	说明在这个目录中的表上定义的触发器，并且允许给定用户/角色访问
UDT_PRIVILEGES	说明在这个目录中定义的，并且可以访问或者由某给定用户/角色授予的用户定义类型上的特权
USAGE_PRIVILEGES	说明在这个目录中定义的，并且可以访问或者由某给定用户/角色授予的对象上的 USAGE 特权
USER_DEFINED_TYPES	说明在这个目录中定义的用户定义类型，并且允许给定用户/角色访问
VIEWS	说明用户/角色可以访问的每个视图；每个视图占用一行
VIEW_COLUMN_USAGE	说明用户可以访问的视图所引用的列；每个列占用一行
VIEW_TABLE_USAGE	说明用户可以访问的视图所引用的表；每个表占用一行

RDBMS 供应商用具有不同名称的视图或完全不同的方式实现了相似的功能。请参考本章中的特定 RDBMS 小节（即 Oracle、DB2 或 MS SQL Server）了解更多信息。

Note 注意：重要的是记住 INFORMATION_SCHEMA 视图提供了对数据库元数据和用法信息的标准化访问。本书中讨论的每个数据库都展示了信息本身。通过一些其他供应商定义的渠道可以获得这些信息。

13.2 Oracle 11g 数据字典

Oracle 将术语“数据字典”用于系统目录。每个 Oracle 数据库都有它自己的一组系统表和视图，其中存储了有关物理和逻辑数据库结构的信息；数据字典对象存储在 SYSTEM 表空间中。数据字典对象是只读的，这意味着数据库用户无法手动修改它们；不过，Oracle RDBMS 自身可以响应特定操作从而自动更新这些对象中的数据。例如，当用户 ACME 创建了新对象（表、视图、存储过程等），把列

www.TopSage.com

或约束添加到表等时，后台就会立即更新适当的数据字典表，通过系统视图（本章后面讨论）可以看到相应更改。Oracle 用户 SYS 拥有数据字典的全部基表和用户可访问的视图。

Oracle 的数据字典由几百个不同的视图和表组成，这些视图和表在逻辑上属于不同的类别，但它们中的大多数只有数据库管理员才会感兴趣，而且已经超出了本书的范围。我们只列出信息架构中的主要对象组并简要说明每个类别中最常用的对象。

13.2.1 Oracle 数据字典结构

数据字典通常由基表和用户可访问的视图组成。

基表包含由 Oracle RDBMS 动态更新的所有数据库信息。Oracle 严格禁止使用这些表，即使只是为了选择；数据库用户通常没有访问它们的权限，甚至 DBA 也不能像通常那样直接查询这些表。存储在基表中的信息是隐秘和难以理解的。

用户可访问的视图汇总并显示存储在基表中的信息；它们利用联接、列别名等以可读或简化的形式显示基表中的信息。不同 Oracle 用户可以对不同数据库视图拥有 SELECT 特权。

Note 注意：所有 Oracle 数据字典对象都属于一个称为 SYS 的特殊用户。Oracle 创建公共同义词以简化用户对这些对象的访问（参阅第 4 章）。这意味着，假定如果具有访问 SYS.DBA_TABLES 视图的适当特权，则不必使用完全限定名来引用 SYS.DBA_TABLES，使用 DBA_TABLES 即可。

Note 注意：数字字典视图由静态和动态视图组成。名称“静态”指明在这个视图组中的信息只有在修改数据字典（添加一个列到一个表，创建一个新数据库用户等）时才会改变。动态视图在一个数据库使用期间不断更新；它们的内容主要与性能有关，而与本书无关。动态数据字典视图可以通过前缀 V_$来区分，这些视图的公共同义词以 V$开头。静态视图可以分为三组。每组中的视图前缀分别是 USER_、ALL_和 DBA_，如表 13-2 所示。

表 13-2 静态视图前缀

前缀	范围
USER	用户的视图（用户架构中的对象）
ALL	扩展用户的视图（用户可访问的所有对象）
DBA	数据库管理员的视图（所有用户架构中的所有对象）

列集合在这些视图中几乎是一致的——也就是说，USER_TABLES、ALL_TABLES 和 DBA_TABLES 都具有相同的列，只是 USER_TABLES 没有 OWNER 列（这个列是不需要的，因为这个视图只有属于查询视图的用户的表的有关信息）。

表 13-3 给出了最常用的静态视图的有关信息。

表 13-3 一些 Oracle 数据字典视图

数据字典视图	包含的信息关于
ALL_ALL_TABLES	用户可访问的所有对象和关系表
ALL_CATALOG	用户可访问的所有表、视图、同义词和序列
ALL_COL_PRIVS	授权用户可访问的列
ALL_CONSTRAINTS.	可访问的表上的约束定义
ALL_CONS_COLUMNS	用户可访问的约束定义中的列信息
ALL_INDEXES	用户可访问的表上的索引
ALL_OBJECTS	用户可访问的所有对象
ALL_SEQUENCES	用户可访问的数据库序列
ALL_SYNONYMS	用户可访问的所有同义词
ALL_TABLES	用户可访问的关系表
ALL_TAB_COLUMNS	用户可访问的表、视图和聚集的列
ALL_TRIGGERS	当前用户可访问的触发器

www.TopSage.com

大家网 TopSage.com

续表

数据字典视图	包含的信息关于
ALL_USERS	当前用户可见的数据库的所有用户的信息
ALL_VIEWS	用户可访问的视图
DBA_ALL_TABLES	数据库中的所有对象和关系表
DBA_CATALOG	所有的数据库表、视图、同义词和序列
DBA_COL_PRIVS	数据库中所有列上的授权
DBA_CONSTRAINTS	所有表上的约束定义
DBA_CONS_COLUMNS	数据库约束定义中的所有列的信息
DBA_INDEXES	数据库中的所有索引
DBA_OBJECTS	所有数据库对象
DBA_SEQUENCES	数据库中的所有序列
DAB_SYNONYMS	数据库中的所有同义词
DBA_TABLES	数据库中的所有关系表
DBA_TAB_COLUMNS	数据库中的所有表、视图和聚集的列的说明
DBA_TRIGGERS	数据库中的所有触发器
DBA_USERS	数据库中所有用户的有关信息
DBA_VIEWS	数据库中的所有视图
USER_ALL_TABLES	用户拥有的所有对象和关系表
USER_CATALOG	用户拥有的表、视图、同义词和序列
USER_COL_PRIVS	授予用户作为这些列的所有者、授权者或被授权者
USER_CONSTRAINTS	用户自己的表上的约束定义
USER_CONS_CONLUMNS	用户拥有的约束定义中的列信息
USER_DB_LINKS	用户拥有的数据库链接
USER_INDEXES	用户自己的索引
USER_OBJECTS	用户拥有的对象
USER_SEQUENCES	用户自己的数据库序列
USER_SYNONYMS	用户的专用同义词
USER_TABLES	用户自己的关系表
USER_TAB_COLUMNS	用户的表、视图和聚集的列
USER_TRIGGERS	用户拥有的触发器
USER_USERS	当前用户的信息
USER_VIEWS	用户自己的视图

USER_和 ALL_视图（以及列出的 V$视图）的选择特权默认授予 PUBLIC；DBA_视图只对有特权的用户可见。

13.2.2　Oracle 数据字典和 SQL:2007 标准

我们已经提到，Oracle 是三个数据库中对 SQL:2003 的 INFORMATION_SCHEMA 标准遵从最少的。Oracle 已经实现了它自己的针对系统目录对象的命名约定，这些约定不符合标准。但是，大部分“标准化”信息（至少 Oracle 实现的那些对象）都可以从 Oracle 的数据字典中获取。表 13-4 列出了 SQL:2003 INFORMATION_SCHEMA 视图和 Oracle 数据字典对象之间的大致对应关系。

表 13-4　与 SQL:2007 对应的 Oracle 数据字典的 INFORMATION_SCHEMA 视图

INFORMATION_SCHEMA 视图	Oracle 数据字典视图
CHECK_CONSTRAINTS	USER_CONSTRAINTS USER_OBJECTS
COLUMNS	USER_TAB_COLUMNS
COLUMN_PRIVILEGES	USER_COL_PRIVS
CONSTRAINT_COLUMN_USAGE	USER_CONS_COLUMNS
CONSTRAINT_TABLE_USAGE	USER_CONSTRAINTS
KEY_COLUMN_USAGE	USER_CONS_COLUMNS

www.TopSage.com

续表

INFORMATION_SCHEMA 视图	Oracle 数据字典视图
REFERENTIAL_CONSTRAINTS	USER_CONSTRAINTS
TABLES	USER_TABLES USER_OBJECTS
TABLE_CONSTRAINTS	USER_CONSTRAINTS
TABLE_PRIVILEGES	USER_COL_PRIVS
USAGE_PRIVILEGES	USER_COL_PRIVS
VIEWS	USER_VIEWS USER_OBJECTS
VIEW_COLUMN_USAGE	USER_TAB_COLUMNS

下面的查询（在 ACME 示例数据库的 ACME 用户提交时）获取属于当前用户的所有表的名称和创建日期：

```
SELECT object_name,
       created
FROM   user_objects
WHERE  object_type = 'TABLE';

OBJECT_NAME                                      CREATED
------------------------------------------------ ---------
ADDRESS                                            27-AUG-07
CUSTOMER                                           27-AUG-07
DISCOUNT                                           27-AUG-07
ORDER_HEADER                                       27-AUG-07
. . .                                              . . .
RESELLER                                         27-AUG-07
SALESMAN                                         27-AUG-07
SHIPMENT                                         27-AUG-07
STATUS                                           27-AUG-07

13 rows selected.
```

这个查询结果表明当前在 ACME 数据库中有 13 个表属于用户 ACME，而且所有这些表都是在 2007 年 8 月 27 日创建的。

Note 注意：Oracle 11g 也包括 DBMS_METADATA 包，这些包为提取数据库对象的完整定义提供了接口。这些定义可以表示为 XML 或 SQL DDL。

正如 Oracle 中任何其他视图或表那样，可以联合系统目录视图来产生某些组合输出。接下来的查询把 USER_TABLES 和 USER_TAB_COLS 数据字典视图结合起来，产生属于用户 ACME 的 ADDRESS 表中所有列的列表：

```
SELECT    table_name,
          column_name
FROM      user_tables
  JOIN
          user_tab_cols
  USING   (table_name)
WHERE     table_name = 'ADDRESS';

TABLE_NAME                     COLUMN_NAME
------------------------------ ------------------------------
ADDRESS                        ADDR_ID_N
ADDRESS                        ADDR_CUSTID_FN
. . .                          . . .
ADDRESS                        ADDR_STATE_S
ADDRESS                        ADDR_ZIP_S
ADDRESS                        ADDR_COUNTRY_S

9 rows selected.
```

13.2.3 更深一层：有关元数据的数据

Oracle 数据字典的总体思想是保存有关 RDBMS 内部和 Oracle 用户两者使用数据的数据。与只包含少量视图的 SQL:2003 的 INFORMATION_SCHEMA 不同，Oracle 的数据字典由一千多个对象组成，每一个对象都有数十列，由此引发了一个问题——在哪里寻找数据字典中的特定信息。幸运的是，Oracle 通过一些包含有关系统对象信息的对象提供了这种元信息的一个子集。两个主要视图是 DICTIONARY 和 DICT_COLUMNS，前者包含数据字典表和视图的说明，后者说明这些对象的列。

www.TopSage.com

可以使用一个简单的 SQL 查询来搜索包含需要信息的对象。例如，如果想知道对哪些表中的哪些列有修改权限，那么使用类似下面的查询可以帮助发现它们：

```
SELECT * FROM dictionary
WHERE UPPER(comments) LIKE '%UPDAT%';

TABLE_NAME                COMMENTS
------------------------- -----------------------------------
ALL_UPDATABLE_COLUMNS     Description of all updatable columns
USER_UPDATABLE_COLUMNS    Description of updatable columns
```

查询 ALL_UPDATABLE_COLUMNS 或 USER_UPDATABLE_COLUMNS 中的任何一个都会提供要寻找的信息。

另一个视图 DICT_COLUMNS 提供数据字典对象单个列的信息。接下来的查询显示 USER_OBJECTS 视图的所有列及这些列的注释：

```
SELECT * from dict_columns
WHERE   table_name = 'USER_OBJECTS';

TABLE_NAME        COLUMN_NAME     COMMENTS
----------------- --------------- ----------------------
USER_OBJECTS      OBJECT_NAME     Name of the object
USER_OBJECTS      SUBOBJECT_NAME  Name of the sub-object (for example,
partition)
. . .             . . .           . . .
                                  this object itself?
USER_OBJECTS      GENERATED       Was the name of this object
                                  system generated?
USER_OBJECTS      SECONDARY       Is this a secondary object
                                  created as part of icreate
                                  for domain indexes?
12 rows selected.
```

Tip 提示：可以使用 DESCRIBE 命令获得有关数据字典视图和表的一些基本信息，这与使用它来查询任何其他数据库对象的方式完全相同，例如：

```
DESCRIBE user_sequences

Name                 Null?      Type
-----------------    --------   -------------
SEQUENCE_NAME        NOT NULL   VARCHAR2(30)
MIN_VALUE                       NUMBER
MAX_VALUE                       NUMBER
INCREMENT_BY         NOT NULL   NUMBER
CYCLE_FLAG                      VARCHAR2(1)
ORDER_FLAG                      VARCHAR2(1)
CACHE_SIZE           NOT NULL   NUMBER
LAST_NUMBER          NOT NULL   NUMBER
```

13.3 IBM DB2 9.5 系统目录

IBM DB2 维护两组 INFORMATION_SCHEMA 视图——一组在 SYSCAT 架构中，另一组子集在 SYSSTAT 架构中（SQL 优化器使用它来改善查询性能）。只要运行 CREATE DATABASE 命令，就会创建所有视图；组成目录的视图不能被显式地删除、改变或更新，SYSSTAT 视图中的某些列除外。SYSIBM 架构添加了一组信息视图，它们更紧密地匹配 SQL:2003 标准的视图，INFORMATION_SCHEMA 标准弥补了两者之间的差距。

13.3.1 DB2 中的 INFORMATION_SCHEMA 对象

表 13-5 显示了我们认为最有用的一些视图。

表 13-5 一些 IBM DB2 9.5 UDB INFORMATION_SCHEMA 视图

INFORMATION_SCHEMA 视图	包含的信息关于
SYSCAT.ATTRIBUTES	结构化数据类型的属性
SYSCAT.DBAUTH	数据库授权
SYSCAT.CHECKS	检查约束；对应 SQL:2003 标准的 CHECK_CONSTRAINTS 视图
SYSCAT.COLAUTH	列级特权

www.TopSage.com

TopSage.com

续表

INFORMATION_SCHEMA 视图	包含的信息关于
SYSCAT.COLUMNS	当前用户可访问的列
SYSCAT.COLCHECKS	由检查约束引用的列
SYSCAT.KEYCOLUSE	在键（主键或外键）中使用的列
SYSCAT.CONSTDEP	约束依赖性
SYSCAT.DATATYPES	有效数据类型
SYSCAT.INDEXAUTH	索引特权
SYSCAT.INDEXCOLUSE	组成索引的列
SYSCAT.INDEXDEP	索引依赖性
SYSCAT.INDEXES	索引
SYSCAT.PACKAGES	包
SYSCAT.PACKAGEAUTH	包特权
SYSCAT.PACKAGEDEP	包依赖性
SYSCAT.PROCOPTIONS	存储过程选项
SYSCAT.PROCPARMOPTIONS	存储过程参数选项
SYSCAT.PROCPARMS	存储过程参数
SYSCAT.REFERENCES	引用约束；对应 SQL:2003 标准的 REFERENTIAL_CONSTRAINTS
SYSCAT.SCHEMAUTH	架构特权
SYSCAT.SCHEMATA	为数据库定义的所有架构
SYSCAT.SEQUENCES	数据库序列
SYSCAT.PROCEDURES	存储过程
SYSCAT.TABCONST	为表定义的约束
SYSCAT.TABAUTH	表级特权
SYSCAT.TABLES	在数据库中创建的表
SYSCAT.TABLESPACES	数据库表空间
SYSCAT.TABLESPACEAUTH	表空间特权
SYSCAT.TRIGDEP	触发器依赖性
SYSCAT.TRIGGERS	在数据库中创建的触发器
SYSCAT.FUNCTIONS	用户定义函数
SYSCAT.VIEWS	在数据库中创建的视图

Note 注意：为遵从 SQL 标准，在 SYSIBM 架构中引入了 INFORMATION_SCHEMA 视图，并在系统基表上进行维护。

默认将视图的 SELECT 特权授予 PUBLIC。IBM 明确表示，不同版本中视图的列可能不同，并推荐使用 SELECT * FROM SYSCAT.<view>语法来查询这些表。尽管如此，IBM 特别“保证”某些列能够与相应的视图一起使用（参见表 13-6）。

表 13-6 INFORMATION_SCHEMA 视图的列名

对象	选择的列名
TABLE	TABSCHEMA、TABNAME、COLCOUNT、KEYCOLUMNS
INDEX	INDSCHEMA、INDNAME
VIEW	VIEWSCHEMA、VIEWNAME、TABID、COLNO、COLTYPE
CONSTRAINT	CONSTSCHEMA、CONSTNAME
TRIGGER	TRIGSCHEMA、TRIGNAME
PACKAGE	PKGSCHEMA、PCKGNAME
TYPE	TYPESCHEMA、TYPENAME、TYPEID
FUNCTION	FUNCSCHEMA、FUNCNAME、FUNCID
COLUMN	COLNAME、COLNO、DEFAULT、REMARKS
SCHEMA	SCHEMANAME

下面的查询检索在 ACME 数据库中创建的 CUSTOMER 表的有关信息。为了显示有意义的结果，

www.TopSage.com

把选择的列数限制为三列，因为这个表包含 30 个以上的列。

```
db2 => SELECT tabschema,
               colcount,
               create_time
FROM           syscat.tables
WHERE          tabname = 'CUSTOMER'

TABSCHEMA              COLCOUNT    CREATE_TIME
-------------------    --------    --------------------------
ACME                          7    2007-09-13-19.55.50.95300

1 record(s) selected.
```

返回的信息显示 CUSTOMER 表属于 ACME 架构，创建于 9/13/2007，并包含 7 个列。连接系统目录的表可以产生联合结果。下面的查询连接两个表 SYSCAT.TABLES 和 SYSCAT.COLUMNS，从两个表中提取一个单独的值集：

```
db2 => SELECT tbl.tabname,
               cl.colname
FROM           syscat.tables tbl, syscat.columns cl
WHERE          syscat.tables.tabname = syscat.columns.tabname
AND            syscat.tables.tabname = 'CUSTOMER'

TABNAME         COLNAME
------------    ------------------------------
CUSTOMER        CUST_ID_N
CUSTOMER        CUST_PAYTERMSID_FN
CUSTOMER        CUST_SALESMANID_FN
CUSTOMER        CUST_STATUS_S
CUSTOMER        CUST_NAME_S
CUSTOMER        CUST_ALIAS_S
CUSTOMER        CUST_CREDHOLD_S

  7 record(s) selected.
```

包含在 SYSSTAT 架构中的一些 INFORMATION_SCHEMA 视图是可更新的，如表 13-7 所示。

表 13-7　　可更新的 IBM DB2 9.5 INFORMATION_SCHEMA 视图

INFORMATION_SCHEMA 视图	说明
SYSTAT.COLUMNS	包含每个表的列的有关信息
SYSTAT.INDEXES	包含为数据库表创建的索引的有关信息
SYSTAT.COLDIST	包含有关列用法的详细统计
SYSTAT.TABLES	包含有关数据库表的信息
SYSSTAT.FUNCTIONS	包含有关用户定义函数的信息

Tip　提示：为了与 DB2 Universal Database for OS/390 兼容，IBM 在 SYSCAT 架构中维护 SYSDUMMY1 目录表。这个表由一个 CHAR(1)数据类型的列（IBMREC）和一行组成。参阅第 8 章了解有关使用这个表的更多信息。

13.3.2　获得有关 INFORMATION_SCHEMA 对象的信息

在 DB2 中，DESCRIBE TABLE *<table_name>*命令可以用于获得有关 INFORMATION_SCHEMA 对象内部结构的信息。例如：

```
db2 =>  describe table syscat.views

Column          Type          Type name  Length Scale  Nulls
name            schema
--------------- ------------- ---------- ------- ------ --
VIEWSCHEMA      SYSIBM        VARCHAR       128      0 No
VIEWNAME        SYSIBM        VARCHAR       128      0 No
DEFINER         SYSIBM        VARCHAR       128      0 No
SEQNO           SYSIBM        INTEGER         4      0 No
VIEWCHECK       SYSIBM        CHARACTER       1      0 No
READONLY        SYSIBM        CHARACTER       1      0 No
VALID           SYSIBM        CHARACTER       1      0 No
QUALIFIER       SYSIBM        VARCHAR       128      0 No
FUNC_PATH       SYSIBM        VARCHAR       254      0 No
TEXT            SYSIBM        CLOB      2097152      0 No

  10 record(s) selected.
```


www.TopSage.com

13.4 Microsoft SQL Server 2008 系统目录

Microsoft SQL Server 2008 提供了多种获得系统信息的方式——通过 INFORMATION_SCHEMA 视图或通过系统存储过程和函数。建议使用前者，并且在一定程度上不支持使用过程/函数（Sybase 的遗留产物）；在这个范围中支持的任何功能只是为了向后兼容。

Tip 提示：获得有关 Microsoft SQL Server 2008 系统信息的方式之一是直接查询系统表——也就是包含当前数据库信息的表和视图（例如，sysobjects、sysindexes、sysusers 等，共 19 个表）。这些存储在 Master 数据库中的表包含 RDBMS 自身的信息。尽管对于一个具有足够特权的用户来说，查询这些视图和表是可行的，但 Microsoft 强烈反对这样做，并声明系统表是 SQL Server 自身专用的，而且在未来版本中，其名称和结构可能会改变（而且他们的确这样做了——SQL Server 的每一个版本都会添加一些新表，并删除一些旧表，还有一些名称会发生改变）。建议不使用这个“后门”，而是使用合法的接口来获得信息。

13.4.1 MS SQL Server 2008 的 INFORMATION_SCHEMA 视图

为遵从 SQL-92 标准，在 Microsoft SQL Server 2000 7.0 中首次引入 INFORMATION_SCHEMA 系统视图，并在后续版本中进一步增强。RDBMS 中包含的每一个数据库都定义了这些视图。它们基于系统表，不应直接查询系统表（参阅上一小节的“提示”）。

表 13-8 包含了有关 Microsoft SQL Server 2008 的 INFORMATION_SCHEMA 视图的信息。

表 13-8 Microsoft SQL Server 2008 的 INFORMATION_SCHEMA 视图

INFORMATION_SCHEMA 视图	包含的信息关于
CHECK_CONSTRAINTS	当前数据库的所有检查约束；基于 sysobjects 和 syscomments 系统表。只显示当前用户对其拥有权限的约束
COLUMN_DOMAIN_USAGE	所有用户定义数据类型；基于 sysobjects、syscomments 和 systypes 系统表。只显示当前用户对其拥有权限的数据类型
COLUMN_PRIVILEGES	授予用户或由用户授予的所有特权；基于 sysprotects、sysobjects 和 syscolumns 系统表。只显示当前用户对其拥有权限的特权
COLUMNS	用户可访问的当前数据库中的每个表的每一个列；基于 sysobjects、spt_data type_info、systypes、syscolumns、syscomments、sysconfigures 和 syscharsets 系统表。只显示当前用户对其拥有权限的列
CONSTRAINT_COLUMN_USAGE	数据库中每一个设置了约束的列；基于 sysobjects、syscolumns 和 systypes 系统表。只显示当前用户对其拥有权限的列
CONSTRAINT_TABLE_USAGE	当前数据库中每一个定义了约束的表；基于 sysobjects 系统表。只显示当前用户对其拥有权限的表
DOMAIN_CONSTRAINTS	每一个绑定了规则的用户定义数据类型；基于 sysobjects 和 systypes 系统表。只显示当前用户对其拥有权限的类型
DOMAINS	当前数据库中声明的用户定义类型；基于 spt_data type_info、systypes、syscomments、sysconfigures 和 syscharesets 系统表。只显示当前用户对其拥有的权限的 UDT
KEY_COLUMN_USAGE	当前数据库的每个表中声明为一个键（主键或外键）的每个列；基于 sysobjects、syscolumns、sysreferences、spt_values 和 sysindexes 系统表。只显示当前用户对其拥有权限的键列
PARAMETERS	一个用户定义的存储过程或函数定义的每个参数；也显示一个函数的返回数据参数；基于 sysobjects 和 syscolumns 系统表。只显示当前用户对其拥有权限的存储过程和函数的信息
REFERENTIAL_CONSTRAINTS	当前数据库中定义的每一个外键约束；基于 sysreferences、sysindexes 和 sysobjects 系统表。只显示当前用户对其拥有权限的约束
ROUTINES	当前数据库中定义的每一个存储过程或函数；基于 sysobjects 和 syscolumns 系统表。只显示当前用户对其拥有权限的过程和函数
ROUTINE_COLUMNS	均为表值函数（包含一个 SELECT 语句）的用户函数；基于 sysobjects 和 syscolumns 系统表。只显示当前用户对其拥有权限的函数

www.TopSage.com

大家网 TopSage.com

续表

INFORMATION_SCHEMA 视图	包含的信息关于
SCHEMATA	一个用户可访问的所有数据库；基于 sysdatabase、sysconfigures 和 syscharsets 系统表。只显示当前用户对其拥有权限的数据库
TABLE_CONSTRAINTS	当前数据库中定义的表约束；基于 sysobjects 系统表。只显示当前用户对其拥有权限的约束
TABLE_PRIVILEGES	授予到用户或由用户授予的每一个表特权；基于 sysobjects 和 sysprotects 系统表。只显示当前用户对其拥有权限的特权
TABLES	当前数据库中定义的每个表；基于 sysobjects 系统表。只显示当前用户对其拥有权限的表
VIEW_COLUMN_USAGE	一个视图定义中使用的每一列；基于 sysobjects 和 sysdepends 系统表。只显示当前用户对其拥有权限的列
VIEW_TABLE_USAGE	用作视图的一个基表的表；基于 sysobjects 和 sysdepends 系统表。只显示当前用户对其拥有权限的表
VIEWS	当前数据库中用户可访问的视图；基于 sysobjects 和 syscomments 系统表

就像数据库中任何其他视图或表那样查询 INFORMATION_SCHEMA 视图，但是这里有一个重要的区别：视图名前必须使用 INFORMATION_SCHEMA 限定符。每一个视图都包含多列，并且可以在这些视图包含的列上指定搜索条件（要获得 INFORMATION_SCHEMA 视图列的完整列表，请参考供应商文档）。

Note　注意：从 SQL Server 2005 开始，数据库中所有对象的元数据对于 PUBLIC 角色的成员不再可见；为了保证这些表的安全而限定为拥有这些表或授予了权限的用户才能访问这些表。请参阅第 12 章了解有关 SQL Server 安全的更多信息。

下面几个例子展示 SQL Server 2008 中 INFORMATION_SCHEMA 视图的用法。

下面这个查询返回拥有权限的 ACME 数据库中每一列的有关信息（稍稍缩短了显示的结果，因为这个查询会返回数据库中每一个表的所有行，包括 SQL Server 用于对象跟踪的 19 个系统表；因为视图总共包含 23 列，所以只请求返回 4 列）：

```
SELECT table_name,
       column_name,
       column_default,
       data_type
FROM   information_schema.columns

TABLE_NAME          COLUMN_NAME          DATA_TYPE
----------------    ---------------      -------------
ORDER_LINE          ORDLINE_ID_N         int
ORDER_LINE          ORDLINE_ORDHDRID_FN  int
. . .               . . .                . . .
SHIPMENT            SHIPMENT_ID_N        int
SHIPMENT            SHIPMENT_BOLNUM_S    varchar
SHIPMENT            SHIPMENT_SHIPDATE_D  datetime
SHIPMENT            SHIPMENT_ARRIVDATE_D datetime
. . .               . . .                . . .      . . .
STATUS              STATUS_ID_N          int
STATUS              STATUS_CODE_S        char
STATUS              STATUS_DESC_S        varchar
. . .               . . .                . . .
v_customer_totals   customer_name        varchar
v_customer_totals   order_number         varchar
. . .               . . .                . . .
SALESMAN            SALESMAN_NAME_S      varchar
SALESMAN            SALESMAN_STATUS_S    char

(112 row(s) affected)
```

例如，如果希望只查询 SALESMAN 表的信息，则查询如下：

```
SELECT table_name,
       column_name,
       column_default,
       data_type
FROM   information_schema.columns
WHERE  table_name = 'SALESMAN'

TABLE_NAME          COLUMN_NAME          DATA_TYPE
----------------    ---------------      -------------
SALESMAN            SALESMAN_ID_N        int
```


www.TopSage.com

```
SALESMAN            SALESMAN_CODE_S       varchar
SALESMAN            SALESMAN_NAME_S       varchar
SALESMAN            SALESMAN_STATUS_S     char

(4 row(s) affected)
```

就像使用“常规”视图或表那样，联接 INFORMATION_SCHEMA 视图或使用任何其他的 SQL 表达式都是可行的。例如，要找出哪些表在其列上没有声明约束，可以使用下面的查询：

```
SELECT tbl.table_name,
       tbl.table_type
FROM   information_schema.tables tbl
WHERE  tbl.table_name NOT IN
  (SELECT table_name
   FROM   information_schema.constraint_column_usage)

TABLE_NAME              TABLE_TYPE
----------------        ---------------
sysconstraints          VIEW
syssegments             VIEW
v_contact_list          VIEW
v_customer_status       VIEW
v_customer_totals       VIEW
. . .                   . . .
v_fax_number            VIEW
v_phone_number          VIEW
v_wile_bess_orders      VIEW

(10 row(s) affected)
```

结果显示，ACME 数据库中的的每个表都有为其声明的某种类型的约束（至少是主键或外键），但视图没有。注意最终的结果集包含两个系统视图。

13.4.2 Microsoft SQL Server 系统存储过程

根据目的和执行的任务，Microsoft SQL Server 2008 支持许多类别的系统存储过程。本章只比较详细地讨论目录过程。

表 13-9 包含有关 Microsoft SQL Server 2008 INFORMATION_SCHEMA 系统存储过程类别的信息。

表 13-9 Microsoft SQL Server 2008 系统存储过程类别

类别	说明
Active Directory 过程	用于通过 Microsoft Windows Active Directory 注册 SQL Server
目录过程	返回有关系统对象的信息；执行 ODBC 数据字典功能
游标过程	执行游标功能（参阅第 14 章了解有关游标的更多信息）
数据库引擎过程	用于数据库引擎的一般维护
数据库维护计划过程	用于建立和执行核心数据库维护任务
分布式查询过程	用于执行和管理分布式查询
全文检索过程	属于全文检索功能；要使用全文索引需要特殊的设置
自动化过程	允许在标准 Transact-SQL 代码中使用自动化对象（ActiveX、.NET）
复制过程	用于建立和管理复制过程
安全过程	安全管理过程
数据库邮件和 SQL 邮件过程	将电子邮件操作与 SQL Server 集成
SQL 事件探查器过程	SQL 事件探查器附件用于监视 SQL Server 性能
SQL Server 代理过程	SQL Server 代理用于管理预定计划的活动
系统维护过程	用于整个 RDBM 系统的维护任务
Web 助手过程	SQL Server Web 助手通过 Internet Information Server 发布信息
XML 过程	用于在 XML 文档上执行操作（参阅第 17 章有关 XML/SQL 集成的内容）
常规扩展过程	引用能够访问基础 Windows OS 资源的存储过程

目录存储过程是本段的重点，我们将研究其中一些最有用的内容，并提供例子。Microsoft SQL Server 2008 列出了 12 个存储过程，用于提供系统信息。可以直接从 SQLCMD 的命令行界面、SQL 查询分析器或通过提供的任何编程接口访问 SQL Server 的客户端应用程序来使用这些过程。最初，这些过程的目的是为了实现 ODBC 数据字典功能以便将 ODBC 应用程序可能的更改与 SQL Server 系统表

www.TopSage.com

结构分离。

Tip　提示：系统存储过程的使用是 Microsoft SQL Server 2008 和 Sybase adaptive Server 独有的，因为两者都源自 Microsoft、Sybase 和 Ashton-Tate 于 1988 年开始的合作项目。INFORMATION_SCHEMA 视图是从 SQL Server 7.0 版本才开始引入的。

这些存储过程返回的信息通常与当前数据库有关；在服务器范围的情况中，它与当前安装中的数据库有关。此外，这些过程可能返回通过数据库网关可访问的对象的有关信息（即注册为 SQL Server 合法数据源）。由于实现的细节随数据源的不同而不同，所以有些信息可能不可用。

1. sp_tables

这个过程返回当前数据库中的表和视图（以及一些与之有关的信息）的列表。表 13-10 中给出了返回的数据集的格式。

表 13-10　由 sp_tables 系统存储过程返回的结果集

列名	说明
TABLE_QUALIFIER	表限定符名。在 SQL Server 中，这个列代表数据库名。这个字段可以是 NULL
TABLE_OWNER	包含表的所有者姓名，通常是 DBO（数据库所有者）
TABLE_NAME	包含表的名称
TABLE_TYPE	指定对象的类型：表、系统表或视图
REMARKS	可以把注释添加到表，但这个列通常返回 NULL

这个过程接受一些可选的参数，这些参数对应于表 13-10 中的列名；如果省略，则将列出当前数据库中每个单独的表和视图。下面是一个例子：

```
1> exec sp_tables
2> go
TABLE_QUALIFIER  TABLE_OWNER  TABLE_NAME    TABLE_TYPE    REMARKS
---------------  -----------  -----------   -----------   ------
acme             dbo          syscolumns     SYSTEM TABLE NULL
. . .            . . .        . . .         . . .         . . .
acme             dbo          ADDRESS       TABLE         NULL
acme             dbo          v_fax_number  VIEW          NULL
```

注意，这个过程列出了当前数据库环境中的所有的系统表和视图。

Note　注意：如果由系统存储过程 sp_server_info（参阅本章稍后的内容）返回的服务器属性 ACCESSIBLE_TABLES 是 Y，那么，将只返回当前用户具有访问权限的表。

2. sp_columns

这个过程返回当前会话中可访问的所有列（以及特定于列的信息）。下面是一个基本例子：

```
exec sp_columns 'CUSTOMER'
```

Cross-Reference　交叉参考：这个过程等价于 ODBC API SQL 列。参阅第 16 章了解有关 ODBC 的详细内容。

3. sp_server_info

执行这个不带任何参数的过程将返回 29 行，而指定属性 ID 将准确返回一行数据。返回的信息表示一个被称为 SQLGetInfo 的 ODBC 所提供数据的子集。

```
1> exec sp_server_info
2> go

attribute_id  attribute_name              attribute_value
------------  --------------------------  ---------------------
1             DBMS_NAME                   Microsoft SQL Server
. . .         . . .                       . . .
12            MAX_OWNER_NAME_LENGTH       128
13            TABLE_LENGTH                128
. . .         . . .                       . . .
500           SYS_SPROC_VERSION           10.0.1019

(29 row(s) affected)
```

如果指定了数据库网关或链接数据源，那么这个过程就能够返回非 SQL Server RDBMS 的信息。

4. sp_databases

这个过程返回所有 SQL Server 数据库的列表。下面是返回数据的例子，这些数据是附录 D 中描述

www.TopSage.com

TopSage.com

的执行 Microsoft SQL Server 安装的有关数据：

```
1> exec sp_databases
2> go

DATABASE_NAME            DATABASE_SIZE REMARKS
------------------------ ------------- --------------
acme                     2176          NULL
master                   4608          NULL
model                    1664          NULL
msdb                     5888          NULL
tempdb                   8704          NULL
```

返回的数据包含在 Master 数据库的 SYSDATABASES 表中。这个过程没有相应的 ODBC 函数。

获得帮助

在获得有关数据库对象的信息方面，最有用的过程之一是表 13-11 中列出的 sp_help<>存储过程组。

表 13-11　　在 Microsoft SQL Server 2008 中选择的存储过程

存储过程	说明
sp_help	返回当前数据库中数据库对象的信息
sp_helpuser	返回有关数据库用户、数据库角色等的信息
sp_helptrigger(<tabname>)	返回当前数据库在指定表上定义的有关触发器的信息
Sp_helpserver	返回远程和/或复制服务器的信息
sp_helprotect	返回当前数据库中用户权限的信息
sp_helpindex	返回表或视图上的索引的信息

SP_HELP 大概是这些过程中最通用的。如果不带任何参数使用，它将返回当前数据库 sysobjects 表中列出的每个数据库对象（表、视图、存储过程、索引、默认值等等）的信息。

```
1> exec sp_help
2> go

Name                 Owner        Object_type
-------------------- ------------ -----------------
sysconstraints       dbo          view
. . .
v_phone_number       dbo          view
. . .                . . .        . . .
ADDRESS              dbo          user table
CUSTOMER             dbo          user table
PK_CUSTPRIMARY       dbo          primary key cns
. . .                . . .        . . .
sp_productadd        dbo          stored procedure
. . .                . . .        . . .
CHK_CUST_CREDHOLD    dbo          check cns
```

再次查询，使用对象名作为参数进行传递，它返回有关这个对象的丰富信息——例如，ACME 示例数据库中的 CUSTOMER 表：

```
exec sp_help 'CUSTOMER'
```

上面的例子返回多个结果集，每一个都描述 CUSTOMER 表。由于内容较多，这里没有显示输出结果。

几乎每一个数据库对象都有 sp_help<object>系统存储过程。（Microsoft SQL Server2008 包含 80 多个 sp_help<..>过程。）请参考 Microsoft SQL Server 2008 文档了解更多的信息。在使用这些过程时，通常会返回多个结果集。

13.4.3 Microsoft SQL Server 2008 系统函数

Microsoft SQL Server 2008 也提供了许多函数，用于返回有关 RDBMS 服务器和包含对象的信息。附录 G 给出了这些函数的完整列表。函数的分组遵循 Microsoft 文档的分组方式，而函数的数量可能会在未来的版本中增加。

1. 配置函数

@@CONNECTIONS 一元函数返回从启动 RDBMS 开始这段时间的连接（或尝试连接）数。

```
SELECT @@CONNECTIONS AS 'Connections Total'
```


www.TopSage.com

```
Connections Total
-----------------
49

(1 row(s) affected)
```

一元函数@@VERSION 返回特定 SQL Server 安装的完整版信息。

```
1> SELECT @@VERSION AS ' SQL Server Info'
2> go

SQL Server Info
-------------------------------------------------------------
Microsoft SQL Server code name "Katmai" (CTP) - 10.0.1075.23
(Intel X86) Nov 8 2007 14:16:26 Copyright (c)
1988-2007 Microsoft Corporation Developer Edition on Windows
NT 5.1 <X86> (Build 2600: Service Pack 2)
(1 row(s) affected)
```

Microsoft SQL Server 2008 在配置函数类别中列出了 15 个函数。

2. 元数据函数

如果没有指定参数，DB_NAME 函数就返回当前数据库的名称，否则返回与一个数值型数据库 ID 对应的数据库名。如果没有指定任何参数，则函数 DB_ID 返回当前数据库的数值型 ID，如果指定了数值型 ID，则返回与那个 ID 对应的数据库名。

```
1> USE ACME
2> SELECT DB_NAME() AS 'database',
3>        DB_NAME(7) AS 'db_by_id',
4>        DB_ID() AS 'id'
5> go

db_by_id             databaseid           id
-------------------  -----------          ------
acme                 acme                      7

(1 row(s) affected)
```

上面的例子使用了 DB_NAME 和 DB_ID 函数；USE ACME 关键字确保指定了 ACME 数据库环境。Microsoft SQL Server 2008 提供了 40 多个不同的元数据函数。

3. 系统函数

如果应用程序进行了设置，则 APP_NAME 函数返回已初始化连接的应用程序名。例如，使用 SQLCMD 命令行实用工具将产生下面的输出（参阅附录 E 了解更多信息）。

```
1> select app_name() as 'application'
2> go

application
---------------
OSQL-32

(1 row affected)
```

@@ROWCOUNT 返回会话范围内上一条 SELECT 语句影响的行数。例如，如果在上面例子的语句后立刻提交@@ROWCOUNT，将得到下面的结果：

```
1> select @@ROWCOUNT as 'rows'
2> go

rows
----------
         1

(1 row affected)
```

Note 注意：这个语句本身会影响结果。例如，如果前面的语句返回 100 条记录，那么语句 SELECT@@ROWCOUNT 将返回 100；在这之后立刻提交这条语句，将返回 1（即受语句自身影响的行）。

下面是 SUSER_SNAME 和 USER_NAME 系统函数用法的例子。给定一个用户标识号，那么第一个函数应该返回登录标识名（如果省略标识号，则是当前用户），第二个函数返回数据库用户名。启动两个 SQLCMD 会话或在 SQL 查询分析器中建立两个连接；然后第一个会话使用 Windows 身份验证，第二个会话使用 SQL Server 身份验证（用户 ID acme，密码 acme）（请参考附录 E 了解详细信息）。

www.TopSage.com

下面是第一个会话（Windows 身份验证）中的函数返回的结果；登录名由计算机名/用户名表示。

```
1> SELECT suser_sname() AS LOGIN_NAME,
2>        user_name() AS DB_USER
3> go

LOGIN_NAME                  DB_USER
--------------------------- -----------
ALEX-KRIEGEL\alex_kriegel   dbo

(1 row(s) affected)
```

下列的例子显示了在通过 SQL Server 身份验证启动的会话中执行的相同查询：

```
1> SELECT suser_sname() AS LOGIN_NAME,
2>        user_name() AS DB_USER
3> go

LOGIN_NAME                  DB_USER
--------------------------- -----------
acme                        dbo

(1 row(s) affected)
```

13.5 小结

在实际的 RDBMS 实现进入市场很久之后，INFORMATION_SCHEMA 才被 ISO/ANSI 组织认可。从而导致一些供应商以系统表的形式实现了他们自己的元数据知识库版本。

可以通过多种方式收集来自这些表的信息，通常是通过视图，这是供应商专门为此目的（INFORMATION_SCHEMA 背后的思想）提供的，或通过供应商提供的存储过程或函数，或者通过供应商提供的函数实现。

大多数供应商都明确表示不支持用户直接访问系统表，因为它们的结构可能会在没有任何通知的情况下发生改变，而表中包含的信息并不保证就是您所认为的信息。简言之，系统表是针对系统的；而视图才是针对用户的。必须通过直接查询基础系统表进行最佳猜测的时代已经结束了。

尽管引入标准是向统一元数据接口的正确方向迈出的一步，但这不会在一夜之间发生（想一想 INFORMATION_SCHEMA 是在 SQL-92 标准中才首次引入的）。在 SQL:2003 和 SQL:2007 标准中指定的系统目录的结构是实现高级标准一致性的必要条件；大多数 RDBMS 产品只兼容级别 1，而且在可预见的未来可能继续维持这种状况（随着 Microsoft SQL Server 2008 的到来，它将成为本书中讨论的供应商中最接近标准的实现）。

www.TopSage.com

大家网
TopSage.com

www.TopSage.com

PART 6

超越 SQL：过程编程和数据库访问机制

本部分包括：

第 14 章
存储过程、触发器和用户定义函数
第 15 章
SQL 和 XML
第 16 章
SQL 和过程化程序设计
第 17 章
SQL 的未来
附录 A
配套网站
附录 B
ACME 示例数据库
附录 C
关系数据库设计基础
附录 D
安装 RDBMS 软件
附录 E
访问 RDBMS
附录 F
安装 ACME 数据库
附录 G
SQL 函数
附录 H
SQL 语法参考
附录 I
SQL 保留关键字
附录 J
其他 RDBMS
附录 K
数值系统、布尔代数和集合论简介

www.TopSage.com

www.TopSage.com

大家网
TopSage.com

CHAPTER 14

第 14 章
存储过程、触发器和用户定义函数

本章提要：

- SQL 过程化扩展
- SQL 过程化语言的关键元素
- 存储过程
- 用户定义函数
- 触发器

前面已有提及，从定义上来说，SQL 是一种非过程化的语言。因此，SQL 不具备过程化结构，尤其无法在（可命名）分层逻辑块中操作，不能接受和返回值，完成迭代，执行条件语句等。

Note

注意：大多数编程语言都是过程化语言。本质上，过程化程序是逐步执行的指令列表，告诉计算机要做什么。例如，过程化程序告诉计算机重复读入用户的输入，使预先定义的常量与输入相乘，再把结果存储到数据库的表中。过程化程序也可以进行错误处理。例如，如果输入了非数字的值，则显示一条错误信息，但程序继续进行。同样，过程化程序可以不把结果存储到表中，而是把结果传递给另一个不同的程序（子程序），当该子程序得到执行时，可以进行下一步计算，并把结果传递给另一个程序模块，依此类推。

最初的 RDBMS 实现不具备过程化语言的能力——使用嵌入式编程来完成所有的过程化数据处理。当时流行的主要过程化语言（C、COBOL 和 Pascal 等）都有（至今仍然有）专门的扩展插件（预编译器），允许程序员在编程语言代码中直接嵌入 SQL 语句。预编译就是把 SQL 翻译成相应的语言结构，随后这些语言结构再编译成二进制代码。嵌入式编程最接近的化身是 SQLJ，SQLJ 把这个概念扩展到了 Java 编程语言中。

Cross-Reference

交叉参考：第 16 章将讨论嵌入式编程。

关系数据库正日趋完善，更多的内部控制授权给了 RDBMS。因而，以编译（二进制）形式或解释形式把过程化程序设计模块内建在 RDBMS 的想法是切实可行的。问题在于，ANSI/ISO 标准当时没有（或几乎没有）为这样的要素制定出指导方针。这样，每一个供应商都推出了自己的内部 RDBMS 过程化模块版本。

SQL1999/SQL2003 为永久存储模块（PSM 于 1996 年首次引入）提供了健壮的支持，并且在 SQL 标准中增加了触发器，而在此刻，所有大型 RDBMS 供应商都已经拥有了自己的过程化语言。Oracle 的过程化 SQL 扩展称为 PL/SQL；MS SQL Server 使用 Transact-SQL；IBM DB2 则引入了自己版本的 SQL PL。有关 SQL 触发例程与扩展触发例程的讨论，请参考第 10 章。

www.TopSage.com

Note 注意：DB2 在 7.1 版本出现之前，一直没有自己的过程化 SQL 扩展插件。不过，编写存储例程时，可以选择 C 或 COBOL 以及后来的 Java。Java 也可以当作 PL/SQL 的一个替代方案，用于在 Oracle 中编写过程化代码。最近，Oracle、IBM DB2 和 Microsoft SQL Server 增加了新的功能，可以使用.NET 家族的语言（主要是 C#和 VB.NET）创建可在 RDBMS 内部部署的存储过程、触发器和函数。

尽管三种语言的基本语法元素类似，但在高级功能上却有明显差别。因为本书规模的限制，不可能深入地探讨所有三种语法，这里仅仅解释常用的基本功能，更多信息，建议参考特定于供应商的文档。

Note 注意：PSM 包括存储过程、用户定义函数和被称为模块的特殊结构，这些模块可以包含过程、函数和为其声明的共享变量。在 Oracle 中，模块称作**包**，MS SQL Server 和 IBM DB2 9.5 都没有实现它们，因此本书不会提及。

在 RDBMS 的过程化例程中，两种最常见的形式是存储过程和触发器，具体来说，这是两种不同的过程化程序设计方法：线性的和事件驱动的。用户定义函数可以看作存储过程的一种特例，许多绑定在一起的存储过程和函数就形成了模块（包）。

顺序的和事件驱动的程序设计

在顺序程序设计中，执行的次序由应用程序控制。换言之，代码以一种预定的顺序执行，而且只需要最少（如果有）的用户干预。对于执行大量顺序任务的批处理程序来说，采用的典型方式就是顺序的或线性的程序设计。在线性程序设计中，最流行的语言有 COBOL、FORTRAN 和 C。

在事件驱动的程序设计中，代码的执行依赖于某个用户动作或系统事件。因此，执行的顺序由用户动作和系统事件控制。通常情况下，应用程序开始运行后，就在等待某个事件（鼠标单击、键击或某个系统事件，如时间程序）的发生。根据特定的事件，程序执行某个动作，完成后即开始等待下一个事件。大多数现代的可视化应用程序都属于事件驱动的程序。像订购项目、金融交易、时间跟踪、银行业务以及许多其他应用程序，启动时即打开主菜单屏幕，接着等待用户执行某个动作。编写事件驱动的应用程序时，最典型的语言是 Visual C++、Visual Basic、C#、VB.NET 和 Java。（当然，COBOL 也可以用于创建事件驱动程序，或者用 C++创建过程化程序；这里强调的只是典型或最普遍的情况。）

在 RDBMS 内部，事件通常更多地与数据库而非上述的情形相关。主要的数据库事件包括用户登录和注销、插入或删除表中的行，以及更新某些列。

14.1 过程化扩展的用途和好处

能够以多种不同的方式并根据各种原因使用存储过程、用户定义函数和触发器。主要用途包括改善性能、减少网络通信量、增强数据库安全和代码的可重用性。

14.1.1 性能和网络通信量

可以使用存储例程改善应用程序的性能。通常情况下，过程是编译后的代码，存储在 PDBMS 中，因此，执行速度通常比未经编译的 SQL 语句快（通常每当用户或程序调用 SQL 语句时，这些语句就被编译；对于 CPU 周期和内存来说，编译过程代价很高）。

因为 SQL 语句已经编译过并且存储在服务器上，所以就不需要发送 SQL 语句，因而显著减少了网络通信量。单独一个语句的作用或许不足以影响整个网络的性能，但对于拥有数千名用户和大量 SQL 语句的大型系统，影响将非常明显。

14.1.2 数据库安全

存储过程、函数和触发器可以用于数据库安全。

存储过程（或函数）是单独的数据库对象，拥有自己的数据库特权。因此，可以这样设计存储过程并分配权限，例如，仅更新表中的某些列。某些列仍然可以更新，即使执行该过程的用户未经允许

www.TopSage.com

而无法完成更新操作——只需要执行特定的存储过程的权限。从某种意义上说，这增加了另一层类似于权限管理的安全性。

对安全实现来说，触发器尤为重要。例如，可以用这样的方式进行设计：用户在对象上执行某些动作，并将其写入专门的数据库表或 OS 文件。日后可以复审这些记录。此外，触发器可以包含逻辑结构，用于拒绝一些操作（如 INSERT/UPDATE/DELETE），防止数据被修改。触发器通常与诸如表之类的数据库对象一起使用。对于整个数据库来说，触发器也可能是全局性的。

交叉参考：更多利用存储过程和触发器来实施数据库安全的内容，请参阅第 12 章。

14.1.3 代码的可重用性

存储例程的另一重要事项是代码的可重用性，存储过程或用户定义函数经过设计并编译后，即可供多个用户（或应用程序）反复使用，从而节省了重复输入大量 SQL 语句的时间，另外也减少了可能的人为错误。同样，需要改变一个永久模块时，只要所有的调用参数（过程的签名）保持相同，那么这个改变就不会影响任何访问它的客户端应用程序（如果嵌有 SQL 语句，这个改变可能需要更新所有的客户端代码）。

14.2 SQL 过程化语言的关键元素

本节将介绍 SQL 过程化语言的主要元素。

14.2.1 变量和赋值

变量可以被看作是值的存储单元。变量可以是 RDBMS 支持的任何一种数据类型。例如，对于 Oracle PL/SQL 的声明变量，NUMBER 和 VARCHAR2 是有效的数据类型，但 MONEY 或 SMALLDATETME 就不是。本书讨论的 SQL 过程化语言的所有变种都是非常典型的——使用变量前必须先进行声明。变量的适用范围取决于变量在代码中的可见性（即使用能力）。通常情况下，变量在声明它们的程序中是局部的。

交叉参考：数据类型在第 3 章讨论过。

不同供应商的声明语法也不同。下面的例子显示了声明整数类型变量 v_prodcount。

1. PL/SQL (Oracle)

该语言中，在程序设计单元的声明部分（首部和单词 BEGIN 之间）声明变量。在 PL/SQL 中，所有语句都以分号结束：

```
v_prodcount NUMBER;
```

声明时可以进行赋值（默认值），也可以在程序体中赋值。在 PL/SQL 中，赋值运算符表示为冒号后跟一个等号（：=）。

```
v_prodcount : = 0;
```

2. IBM DB2 9.5

DB2 的过程化语言 SQL PL 的变量声明用 DECLARE 关键字来实现。在存储程序体内完成声明，而非在声明部分中进行。语句以分号结束：

```
DECLARE v_prodcount NUMBER;
```

使用关键字 SET 给变量赋值：

```
SET v_prodcount = 0;
```

在程序设计单元的作用域内，可以读取和/或改变存储在变量中的值。

3. Transact-SQL (MS SQL Server)

与 IBM DB2 SQL PL 的语法相似，在 Transact-SQL 中，关键字 DECLARE 也可用于变量声明。所有局部变量的前缀必须是 at 符号（@）；全局变量的前缀必须是双 at 符号（@@）。

```
DECLARE @ v_prodcount INTEGER
```

大家网 TopSage.com

变量赋值可以采用以下两种不同的方式：

```
SELECT @ v_prodcount = 0
```

或者：

```
SET @ v_prodcount = 0
```

Note 注意：使用 SET 和 SELECT 赋值存在着细微的差别。首先，SET 语句是变量赋值的 ANSI 标准，而 SELECT 不是。遵循标准可以让代码具有更好的移植性。缺点是 SET 只用于单赋值（SET @variable = <value>），而 SELECT 可用于多个赋值（SELECT @var1 = <value1>,@var2 = <value2>）。其他优点还包括客户端的可编程性（例如，从存储过程返还结果集的能力）。更多信息，请参考供应商的手册。

14.2.2　模块化、子程序和块结构

过程化语言的一个重要特征是模块化，允许开发人员提高代码的可重用性，并使用命名块隐藏执行某个特定操作执行时的复杂性。模块化有两种级别，第一级别为在单一程序中创建顺序逻辑块的能力。第二级别为一个程序调用另一个程序的能力，而被调用的程序又可以进而调用其他程序，依此类推。数个存储过程或函数可以绑定在一个包中。

第一级别的模块化用于构造程序结构，处理相应的错误（本章后面部分将讨论错误处理）在 PL/SQL 中这一点尤其方便，其中每一个程序块都拥有自己的 EXCEPTION 部分：

```
BEGIN -- Main program
   ...
   <statement>,...
   BEGIN -- Nested block level 1
      <statement>,...
      BEGIN -- Nested block level 2
         <statement>,...
      EXCEPTION
         <error_handling_statement>,...
      END; -- Nested block level 2
      <statement>,...
   EXCEPTION
      <error_handling_statement>,...
   END; -- Nested block level 1
   <statement>,...
EXCEPTION
   <error_handling_statement>,...
END; -- Main program
```

其他供应商的过程化 SQL 扩展插件同样实现了嵌套块的思想，目的是达到相似的准则，以及编写功能复杂的大型程序。更多详情，请参考特定于供应商的文档。

下面这个示例展示二级模块化如何调用独立模块。存储过程 A 通过调用过程 B 和 C，把记录插入表 BB 和 CC，每个过程都会把记录插入到相应的表中，下面的示例使用 PL/SQL 风格的伪代码：

```
CREATE PROCEDURE b (field1 INTEGER,
                    field2 INTEGER,
                    field3 INTEGER)
AS
BEGIN
   ...
   INSERT INTO bb
   VALUES (field1, field2, field3);
     ...
END;

CREATE PROCEDURE c (field1 INTEGER,
                    field2 INTEGER,
                    field3 INTEGER)
AS
BEGIN
   ...
   INSERT INTO cc
   VALUES (field1, field2, field3);
     ...
END;

CREATE PROCEDURE a (field1 INTEGER,
                    field2 INTEGER,
                    field3 INTEGER)
```


www.TopSage.com

```
AS
BEGIN
   b(field1, field2, field3);
   c(field1, field2, field3);
END;
```

14.2.3 参数传递

过程化编程与模块化密切相关的另一个特征是能够接受参数或把参数传递给另一个编程模块。参数（也称作变量）一定程度上与变量类似。其主要的不同在于，通常情况下变量是程序内部的，而参数则是模块调用时传递给程序的。例如，一个存储过程或用户定义函数可以没有或具有多个参数，这些参数以后可用于程序逻辑决定的某些动作。参数名和数据类型在程序设计单元的头部定义。

注意：参数可以传递给存储过程或用户定义函数，但是，因为触发器的事件驱动特性，参数不能传递给触发器。“传值”和“传引用”这两种参数传递是有区别的。请参见本章后面的相关讨论。

1. PL/SQL

PL/SQL 和其他专用过程化 SQL 扩展插件之间的区别在于，在 PL/SQL 过程的里，参数声明不能指定数据类型的精度、小数位数或长度。例如，v_prodnum VARCHAR2（10）或 v_prodprice DECIMAL（10,2）将生成一条错误。签名里的参数应以圆括号限定。

下面的示例显示了用 PL/SQL 编写的 SP_PRODUCTADD 过程的声明部分：

```
CREATE PROCEDURE sp_productadd
(
 v_prodid               NUMBER,
 v_prodprice            NUMBER,
 v_prodnum              VARCHAR2,
 v_proddesc             VARCHAR2,
 v_prodstatus           CHAR,
 v_prodbrand            VARCHAR2,
 v_prodpltwid           NUMBER,
 v_prodpltlen           NUMBER,
 v_prodnetwgt           NUMBER,
 v_prodshipwgt          NUMBER
)
...
```

2. Transact-SQL

Transact-SQL 中的参数像变量一样以 at（@）符号作前缀。不需要圆括号：

```
CREATE PROCEDURE sp_productadd
  @v_prodid              INTEGER,
  @v_prodprice           MONEY,
  @v_prodnum             VARCHAR (10),
  @v_proddesc            VARCHAR (44),
  @v_prodstatus          CHAR,
  @v_prodbrand           VARCHAR (20),
  @v_prodpltwid          DECIMAL(5, 2),
  @v_prodpltlen          DECIMAL(5, 2),
  @v_prodnetwgt          DECIMAL(10, 3),
  @v_prodshipwgt         DECIMAL(10, 3)

...
```

3. IBM DB2 SQL PL

其语法类似于 PL/SQL，但需要指定长度或精度及小数位数：

```
CREATE PROCEDURE sp_productadd
(
 v_prodid               INTEGER,
 v_prodprice            INTEGER,
 v_prodnum              VARCHAR(44),
 v_proddesc             VARCHAR(10),
 v_prodstatus           CHAR(1),
 v_prodbrand            VARCHAR(20),
 v_prodpltwid           DECIMAL(5,2),
 v_prodpltlen           DECIMAL(5,2),
 v_prodnetwgt           DECIMAL(10,3),
 v_prodshipwgt          DECIMAL(10,3)
)
...
```

传递参数中的传名和传值

大多数过程化语言都实现了两种主要的参数类型：by value和by reference。两者主要的差别在于，如果从调用模块把变量作为by value参数发送，则实际发送的就是这个变量的一个副本，因此，在子例程内部无论对它做了什么，对原始的变量值都不会产生任何影响。然而，如果以by reference的方式发送这个变量，这个变量的内存地址被传递给子程序，因而可以实际改变这个变量。下面的示例（以伪代码编写的）说明了这个概念：

```
CREATE PROCEDURE POWER_V (x INTEGER BYVALUE)
-- The parameter x is passed by value
AS
BEGIN
  x := x * x;
END;

CREATE PROCEDURE POWER_R (x INTEGER BYREFERENCE)
-- The parameter x is passed by reference
AS
BEGIN
  x := x * x;
END;

CREATE PROCEDURE SQUARE (x INTEGER)
AS
-- Declare local variable y of type integer
DECLARE y INTEGER;
BEGIN
-- Assign variable y the value of x
y := x;
-- Display the value of variable y
PRINT (y);
-- Call procedure POWER_V.
EXECUTE POWER_V(y);
-- The value of y does not change
PRINT (y);
-- Call procedure POWER_R.
EXECUTE POWER_R(y);
-- The variable y was passed by reference
-- so its value actually changes (squared)
PRINT (y);
END;

EXECUTE SQUARE(5);

5
5
25
```

所有“三大”数据库都允许以传值或传引用的方式传递参数，默认为by value。

Oracle和IBM DB2都有两种类型的by reference参数：OUT和IN OUT（DB2中是INOUT）。第一种意味着不能使用这个参数来传递值（通常改为传递NULL），不过，嵌套程序可以给这个参数赋值，主模块可以读取这个值。第二种类型意味着可以向子程序实际发送一个值，子程序将使用这个值进行计算并根据需要进行改变，然后发回调用模块。MS SQL Server只有一种称作OUTPUT类型的by reference变量，大体上与IN OUT一致。

各种实现之间的语法有所不同：

Oracle 11g

```
<variable_name> [IN | OUT | IN OUT] <datatype>
```

例如：

```
my_value1 IN OUT NUMBER;
my_value2 OUT VARCHAR2;
my_value3 IN DATE;
```

注意，最后的声明可以更简单

```
my_value3 DATE;
```

DB2 9.5

```
[IN | OUT | INOUT] <variable_name> <datatype>
```

例如：

```
INOUT my_value1 INTEGER;
OUT my_value2 VARCHAR(30);
IN my_value3 DATE;
```


www.TopSage.com

TopSage.com

最后的语句等价于下面的简单命令行：

```
my_value3 DATE;
```

Microsoft SQL Server 2008

```
@<variable_name> <datatype> [OUTPUT]
```

例如：

```
@my_value1 INTEGER OUTPUT
@my_value3 DATETIME
```

其他两种 RDBM 中常用的 IN 语句没有替代方法。语句@my_values DATETIME IN[PUT]是无效的语法。

14.2.4 有条件地执行

程序有时需要根据不同的条件（通常根据前面程序代码执行的结果）执行不同的动作。例如，可能希望把表中检索到的数乘以不同的系数，系数取决于这个数的大小。

在所有三个专用过程化 SQL 扩展插件中，主要的条件语句采用某种形式的 IF...ELSE 结构：

```
IF <condition1> THEN
   <perform action 1>
[ELSE IF <condition2>
   <perform action2>
]
...
[ELSE IF <conditionN>
   <perform actionN>
]
...
[ELSE
   <perform default action>
]
END IF;
```

这里的<condition...>是布尔表达式，求值结果为 TRUE 或 FALSE。

1. PL/SQL

其语法稍稍不同于前面列出的普通语法。主要区别是用关键字 ELSIF（one word）代替了 ELSE IF：

```
IF v_prodcount = 0 THEN
    INSERT INTO product ...
ELSIF v_prodcount = 1 THEN
    ...
    UPDATE product ...
ELSE
    ...
END IF;
```

2. IBM DB2 9.5

其语法与 PL/SQL 类似，只是用 ELSEIF 关键字代替了 ELSIF：

```
IF (v_prodcount) = 0 THEN
 INSERT INTO product ...
ELSEIF (v_prodcount = 1) THEN
   ...
   UPDATE product ...
ELSE
   ...
END IF;
```

3. Transact-SQL

Transact-SQL 中未使用关键字 THEN。END 只能结束程序或逻辑块，不能终止条件语句。下面的语法说明了 Transact-SQL 中条件结构的用法：

```
IF @v_prodcount = 0
   INSERT INTO product ...
ELSE IF @v_prodcount = 1
   ...
   UPDATE product...
ELSE
   ...
```

Note 注意：在 Transact-SQL 中，如果最后一个 ELSE 子句包含多条语句，那么这几条语句应当包含在 BEGIN-END 中间。

www.TopSage.com

交叉参考：条件执行的另一个例子是 CASE 函数，已经在第 10 章讨论过了。

14.2.5 重复执行

有时，可能需要重复执行同一个执行路径中的某个代码块。例如，可能需要程序从一个文件中读取一行，解析成令牌，再把值赋给变量，然后，把这一行数据插入到表中，反复执行这个操作，直到该文件结束。也有可能根据用户的输入，读取某些预定的输入次数，或者根据变量执行某些计算，直到得到某个结果，等等。允许反复执行的程序设计结构被称为**循环**。

大多数程序设计语言具备以下三种类型的循环。

- WHILE 循环重复执行一个预定义的动作，只要条件保持为 TRUE。
- REPEAT UNTIL 循环重复执行一个代码块，直到条件成为 TRUE。
- FOR 循环根据某个预先设定的次数重复执行。

通常一种语法就可以模拟所有三种循环类型，例如，可以只用 WHILE 循环模拟 REPEAT UNTIL 和 FOR 循环。只有 Oracle PL/SQL 具备所有三种循环类型，但所有“三大”数据库都具备 WHILE 循环，因此本章对其进行讨论。有关使用 WHILE 循环模拟其他循环的详细信息，请参考供应商的文档。

PL/SQL

```
WHILE <condition>
LOOP
   <statement1>;
   [<statement2>;]...
END LOOP;
```

IBM SQL PL

```
WHILE <condition>
DO
   <statement1>;
   [<statement2>;]...
END WHILE;
```

Transact-SQL

```
WHILE <condition>
  [BEGIN]
      <statement1>
      [ BREAK ]
      [<statement2>]...
      [ CONTINUE ]
  [END]
```

在上述代码中，关键字 BREAK 和 CONTINUE 都很重要。BREAK 用于从 WHILE 循环中退出；然后执行 END 关键字之后的语句。CONTINUE 重新启动 WHILE 循环，忽略 CONTINUE 关键字之后的任何语句。

注意：**Oracle** 和 IBM DB2 分别使用 EXIT 和 LEAVE/ITERATE 关键字达到类似的目的。更多信息请查阅供应商的文档。

14.2.6 游标

游标是一种特殊的程序设计结构，允许创建一个命名的工作区，用于从表中提取数据，并且可以按顺序访问存储的信息。游标的主要优势在于，能够逐行操作，而不是把记录集看作一个整体（就像基于集合的 SQL 语句一样）。例如，所有的 DML 语句都可以操作记录集，因此可以改变其“全部否则全不”的操作方式。如果更新语句正在改变成百上千的行，只要有一行违反了某个列约束，整个语句都会失败并回滚所有发生过的改变。然后修正数据，重新开始语句的执行。游标能够单独操作每一行，从而可以妥善处理这样的情况。如果一个逻辑结合了游标、循环和条件语句，则可以生成一个警告，把出错行的更新信息存储在专门的表中，并进一步处理。同样，游标也能灵活处理提交和回滚（提交可以发生在每行或任意行后），有时这样做非常有利于节省系统内存空间，并可以更好地控制程序的执行。在用于 DML 语句之前，可以在某些值上应用条件逻辑并执行计算；游标能够用相同的值更新多个表，还可以实现其他多种目标。

不同的 RDBMS 供应商按不同的方式实现游标。因为语法和功能方面均不同，所以很难讨论某种通用的游标。SQL: 2003 标准要求游标可以滚动，也就是说，应该能够在记录集中从一条记录向前或向后移到另一条记录，但至今为止，仅有少数 RDBMS 供应商（需要注意的是 MS SQL Server）实现

www.TopSage.com

了该功能。主要原因是实现游标的滚动需要耗费相当多的资源，并非每个系统都能胜任。实际上，许多MS SQL Server开发人员都被明确告知，除非绝对必要，否则不要使用游标，但与此同时，PL/SQL程序员的日常工作却离不开游标的使用。

虽然有种种差别，但所有游标实现都有一些共同的特征。通常能在游标上执行的主要操作有DECLARE、OPEN、FETCH和CLOSE。

1. DECLARE

DECLARE将游标名与某条SELECT语句关联，另外定义了一个保存相应列（列可以是不同的数据类型）的内存结构。简化后的SQL：2003语法如下所示：

```
DECLARE <cursor_name> [SENSITIVE | INSENSITIVE] [SCROLL] CURSOR
FOR <select_statement>
[FOR [READ ONLY | UPDATE [OF <column_name>,...]]
```

INSENSITIVE构造基于<select_statement>数据的临时副本，因此对于后来可能发生的任何基础表DML更改，游标获取都是不敏感的。默认为SENSITIVE，这意味着以后发生的任何一次存取都使用某个具体时刻的实际数据。因为在内存中的操作要比读磁盘快很多，所以为了节省资源并加速操作的执行，应尽量使用前一种方式。

SCROLL把游标指定为可滚动的，意味着可以使用额外的FETCH选项（稍后讨论）。

READ ONLY选项防止游标改变数据，UPDATE则指定在游标内部可以更新的列。

同样的SQL：2003标准语法在Transact-SQL（尽管本书或SQL：2003不会提及它拥有许多其他的高级游标功能）中是有效的。Oracle PL/SQL的游标声明差别却很大：

```
CURSOR <cursor_name> [<parameter_list>] [RETURN <return_type>]
IS <select_statement>;
```

上述语句省略了关键字DECALRE，而用IS关键字代替了FOR；此外，PL/SQL游标不能滚动，但可以有选择地获取参数并返回值。

IBM DB2 9.5游标也不允许滚动；除此之外，下面的语法与SQL：2003标准类似：

```
DECLARE <cursor_name> CURSOR
FOR <select_statement>
```

2. OPEN

OPEN语句执行基础查询，同时标识由所有满足游标声明中指定条件的行组成的结果集。其基本语法在所有三个RDBMS中相同：

```
OPEN <cursor_name>
```

3. FETCH

FETCH检索当前行（游标不可滚动）或特定行（游标可滚动），解析这个行，并在预定义的变量集中置入列值。Transact-SQL的语法如下所示：

```
FETCH [ NEXT | PRIOR | FIRST | LAST | ABSOLUTE n | RELATIVE n ]
FROM <cursor_name>
INTO @<variable_name>,...
```

对于不可滚动游标，PS/SQL和DB2的SQL PL的语法如下：

```
FETCH <cursor_name>
INTO <variable_name>,...
```

Note 注意：在游标的声明部分，<select_statement>的SELECT子句中指定的列数必须正确匹配FETCH语句的INTO子句中指定的变量数。此外，Oracle使用锚定表的数据类型结构，该结构声明后可用于存储从游标获取的值。

不可滚动游标的行通常从循环中提取，这样一直处理到最后一行；可滚动的游标可用作更复杂的处理逻辑的组成部分。

4. CLOSE

CLOSE语句清空内存、释放锁，并置游标的结果集为未定义。其语法如下：

```
CLOSE <cursor_name>
```

OPEN语句可重新打开游标。

Note 注意：除了本节描述的4个主要的游标操作外，Transact-SQL要求利用DEALLOCATE语句解除所有游标，该语句释放组成游标的所有数据结构：

```
DEALLOCATE <cursor_name>
```


www.TopSage.com

大家网 TopSage.com

5. 游标例子

正如上面所述，游标可逐行操作，提取过程通常发生在循环结构内。下面的例子非常简单，主要目的是说明如何声明、打开、提取和关闭游标，并着重强调各种实现之间的差异。

基于 V_CONTACT_LIST 视图的查询检索记录，SP_PRINTCUSTPHONE 过程声明并打开游标 cur_custphone，然后逐个提取，并将其打印到标准输出。

Note　注意：术语“标准输出”表示程序把输出发送到的设备。情况下，默认的标准输出通常是监视器的显示器。标准输出可重定向到 OS 文件。

PL/SQL

每一个 PL/SQL 游标都有 4 个与之关联的属性，这些属性可以附加到游标名，返回某个信息。下面是这几个属性。

- %FOUND：如果上一个 FETCH 返回一行，则为 TRUE；否则为 FALSE。
- %ISOPEN：如果游标已打开，则为 TRUE；否则为 FALSE。
- %NOTFOUND：逻辑上与%FOUND 相反。
- %ROWCOUNT：至目前为止提取的行数。

下面的例子使用%FOUND 确定何时该退出循环（产生 FALSE 时，则表明再没有需要提取的行，在代码中，注释以斜体表示）：

```
CREATE OR REPLACE PROCEDURE SP_PRINTCUSTPHONE
-- The procedure displays customer names and phone numbers
IS
  -- Declare local variables to fetch cursor values into
  v_custname VARCHAR(50);
  v_phone VARCHAR(20);
  -- Declare cursor CURSOR cur_custphone IS
  SELECT name,
         phone_number
  FROM    v_contact_list
  WHERE   contact_type = 'customer';
BEGIN
  -- Open cursor
  OPEN cur_custphone;
  -- Fetch cursor for the first time
  FETCH cur_custphone
  INTO v_custname,
        v_phone;
  -- Enable standard output
     DBMS_OUTPUT.ENABLE;
  -- Fetch records in loop until the last one is reached
  WHILE cur_custphone%FOUND
  LOOP
     FETCH cur_custphone
     INTO  v_custname,
           v_phone;
     -- Put line into the standard output
     DBMS_OUTPUT.PUT_LINE(v_custname || v_phone);
   END LOOP;
   -- Close cursor CLOSE cur_custphone;
END SP_PRINTCUSTPHONE;
  /
```

Transact-SQL

Transact-SQL 使用游标全局函数得到与游标有关的信息。两个最常用的是@@FETCH_STATUS 和@@CURSOR_ROWS，前者表示如果最近的提取成功，则返回 0，否则返回一个负数，后者返回最近打开的游标中当前存在的合格行的数量。

Note　注意：前面已经提及，Transact-SQL 游标可以滚动。不同于 PL/SQL 或 DB2 的 SQL PL 游标，使用这些游标时，只能到达记录集中的下一个记录，除非关闭并重新打开游标，否则无法使用前面的行。Transact-SQL 可选择在游标所包含的整个记录集中跳转——非常类似于使用 CD 播放器按钮在 CD 盘的歌曲间跳转。声明游标后，可以使用多种获取游标其他信息的系统存储过程。

下面的 Transact-SQL 例子使用@@FETCH_STATUS 确定何时终止循环；游标没有声明为可滚动，这样做是为了与前面的例子取得一致：

www.TopSage.com

```
CREATE PROCEDURE SP_PRINTCUSTPHONE
-- The procedure displays customer names and phone numbers
AS
  -- Declare local variables to fetch cursor values into
  DECLARE @v_custname VARCHAR(50)
  DECLARE @v_phone VARCHAR(20)
  -- Declare cursor
  DECLARE cur_custphone CURSOR FOR
  SELECT name,
         phone_number
  FROM   v_contact_list
  WHERE  contact_type = 'customer'
BEGIN
  -- Open cursor
  OPEN cur_custphone
  -- Fetch cursor for the first time
  FETCH cur_custphone
  INTO @v_custname,@v_phone
  -- Fetch records in loop until the last one is reached
  WHILE @@fetch_status = 0
  BEGIN
     PRINT @v_custname + @v_phone
     FETCH cur_custphone
     INTO  @v_custname,
           @v_phone
END
  -- Close_cursor
  CLOSE cur_custphone
  DEALLOCATE cur_custphone
END
```

Note 注意：有关创建存储过程的更多信息，请参阅本章后面部分。

14.2.7 错误处理

本书讨论的三种过程化 SQL 扩展插件中，错误处理大不相同。为了处理程序执行期间出现的异常情况，Oracle、IBM DB2 和 SQL Server 2008 使用结构化的错误处理。当发生错误时，RDBMS 生成一个**异常**（各供应商之间的实际语法差别很大）。异常可以预定义（例如，NO_DATA_FOUND 作为 Oracle 中最普通的异常之一，在 SELECT...INTO 语句未返回行时发生此异常）或用户自定义。MS SQL Server 2008 允许在程序体内引发、捕获和处理错误。

错误处理是一个非常复杂的过程。本章给出了基本的错误处理；更多详情，请参考特定于供应商的文档。

1. PL/SQL

所有 PL/SQL 程序设计块都拥有一个可选的 EXCEPTION 部分，如果发生错误，程序的控制则传递到这里。Oracle 拥有一组预定义异常。其中就有已经提及过的 NO_DATA_FOUND，另一个常见的异常是 TOO_MANY_ROWS，如果单行子查询返回多行，则会出现这个异常。任何类型都可以生成的通用异常是 OTHERS。在程序的声明部分可以声明自己的异常，且对其进行实例化，然后在程序体中引发该异常：

```
CREATE PROCEDURE ...
AS
   ...
   /* To handle error conditions that have no predefined
   name, you must use the pragma EXCEPTION_INIT.
A pragma is a compiler directive that is processed at compile time.
   */
   my_exception EXCEPTION;
   PRAGMA EXCEPTION_INIT(my_exception, -400);
   ...
BEGIN
   ...
   IF <special_condition>
      RAISE my_exception;
   ...
EXCEPTION
  WHEN no data_found THEN
      <statements to handle no_data_found>
  WHEN too_many_rows THEN
```


www.TopSage.com

```
        < statements to handle too_many_rows >
    WHEN my_exception THEN
        < statements to handle my_exception >
    WHEN others THEN
        < statements to handle all other exceptions >
END;
```

处理具体异常的操作可以大不相同。例如，可能需要回滚（或提交）事务，然后退出程序；或者只是简单地继续执行，不需要做任何动作；也可能要求显示一条错误消息（或者，写入某个表或 OS 文件）等。

2. IBM DB2 9.5

当 IBM DB2 出现一个错误时，条件处理程序决定 SQL 程序的行为。DB2 中的三个普通条件是 NOT FOUND、SQLEXCEPTION 和 SQLWARNING。如果 SQL 程序中的某条语句引发了 SQLWARNING 或 NOT FOUND 条件，并且已经声明了一个针对相应条件的处理程序，那么控件将传递给相应的处理程序；否则，DB2 把控件传递到程序体中的下一条语句。当某条语句引发 SQLEXCEPTION 条件时，情况会有所不同。如果声明了这个处理程序，控件依然会传递给相应的处理程序；如果未声明，程序将终止执行，并且控件将被传递给调用模块。

IBM DB2 共有三个主要的处理程序类型：CONTINUE、EXIT 和 UNDO。CONTINUE 意味着遇到发生错误的语句时，程序会继续执行该语句后面的语句（在完成特定错误处理语句之后）；EXIT 则指示程序跳转到程序中的最后一条语句；UNDO 类似于 EXIT，不过会回滚程序中的所有 DML 语句：

```
CREATE PROCEDURE ...
...
LANGUAGE SQL
BEGIN ATOMIC
  ...
  DECLARE [CONTINUE | EXIT | UNDO] HANDLER FOR <condition>
    [<error-handling statements>]
  ...
END
@
```

3. Transact-SQL

Transact-SQL 通常利用前面描述的条件语句（IF...ELSE）进行处理错误。最新的版本仍然允许此语法向后兼容，不过会逐渐淘汰。处理关键问题的老办法是使用 RAISERROR 语句，该语句会终止程序，并把控件返回给调用模块；不过，如果需要程序的执行继续下去，可以使用 GOTO 语句指向需要执行的语句块：

```
CREATE PROCEDURE ...
AS
   ...
BEGIN
   ...
   IF <special_condition>
      RAISERROR(<parameters_for_raiserror>)
   ...
   IF <another_special_condition>
      GOTO <label>
   ...
   <label>
      <error_handling_statements>
   ...
END
```

对于最近执行的 Transact-SQL 语句的错误代码，可使用特殊的全局一元函数@@error 返回。如果这个语句成功执行，则结果被设置为 0。如果出现错误，则返回错误消息代码。

从 2005 版开始，SQL Server 实现了结构化的错误处理机制，其中使用 TRY...CATCH 块。当 TRY 块抛出一个错误时，立即把控件传递到 CATCH 块中处理。TRY 块可以内嵌于 CATCH 块中，在错误处理过程中对错误抛出自身进行处理。值得注意的是，在 SQL Server 2008 中，错误的严重性将决定它是否被提交给错误处理例程；严重性等于 10 或小于 10 的所有错误将不被认定为错误，因而无需交予 CATCH 块进行处理；严重性等于 20 或大于 20 的所有错误通常会导致数据库引擎关闭客户端连接，且 CATCH 块也无法对其进行处理（不过，如果连接并未关闭，CATCH 可以处理）。在下面这个示例中，使用 TRY...CATCH 错误处理方式改写了前述过程：

www.TopSage.com

TopSage.com

```
CREATE PROCEDURE <procedure name>
AS
BEGIN
   ...
   BEGIN TRY
   -- some SQL statements that might raise an error
END TRY
BEGIN CATCH
     <error_handling_statements>
  END CATCH
END
```

14.3 存储过程

之前已有提及，存储过程是线性或顺序性的程序。各种实现的语法大不相同，但各种实现还是具有一些共同的特征。存储过程都可以接受参数，允许局部变量声明，而且都是结构化的，还允许使用子模块；同样，也允许重复执行条件语句。

14.3.1 CREATE PROCEDURE 语法

在各种 RDBMS 实现中，CERATE PROCEDURE 语法有所不同。SQL：2003 和三个主要的 RDBMS 供应商的简化语法如下：

SQL:2003

```
CREATE PROCEDURE <procedure_name>
<procedure_definition>
```

PL/SQL

```
CREATE [OR REPLACE] PROCEDURE [qualifier.]<procedure_name>
([<argument_name> IN | OUT | IN OUT <datatype>
                                    [DEFAULT <default>],...])
{IS | AS}
  [<variable_name <datatype> [DEFAULT <default>];],...
BEGIN
  <procedure_body>
[EXCEPTION
  <exception_statements>]
END;
```

IBM DB2 SQL PL

```
CREATE PROCEDURE [qualifier.]<procedure_name>
([IN | OUT | INOUT <argument_name> <datatype>,...])
{MODIFIES SQL DATA |
 NO SQL |
 CONTAINS SQL |
 READS SQL DATA}
[[NOT] DETERMINISTIC]
LANGUAGE SQL
BEGIN [ATOMIC]
  <procedure_body>
END
```

Transact-SQL

```
CREATE PROC[EDURE] <procedure_name>
   [@<parameter_name> <datatype> [ = <default>] [OUTPUT] ] ,...
AS
   <procedure_body>
```

14.3.2 创建一个简单的存储过程

本存储过程示例实现了以下商业逻辑：新产品经常要输入到 ACME 示例数据库的 PRODUCT 表中。问题是，职员有时输入的产品名称采用不同的大小写字母组合，从而引起重复。例如，SPRUCE LUMBER 30×40×50、spruce lumber 30×40×50 和 Spruce Lumber 30×40×50 应指同一条记录，却被当作三条单独的记录输入，这三条记录的主键各不相同，其他表的外键在引用这些主键时，会导致数据完整性问题。

如果给定了产品描述的产品不存在，SP_PRODUCTADD 过程将把一行添加到 PRODUCT 表，否则，SP_PRODUCTADD 过程将用新值更新当前的记录。该过程接受 10 个参数，对应 PRODUCT 表中的每一列，

www.TopSage.com

然后检查是否存在相同的（大写或小写）的产品描述，再根据结果执行相应的动作（INSERT 或 UPDATE）。

1. Oracle 11g

典型的 Oracle 存储过程的组成部分有：标头、声明部分、主体和异常处理部分。标头（在关键字 CREATE 和 IS 之间）包括参数名和数据类型。声明部分可以声明局部变量。主体部分存储过程的逻辑，可选的异常部分则处理异常——过程执行期间可能发生的错误。下面是实现这个逻辑的语法：

```
CREATE PROCEDURE sp_productadd
/* This procedure adds new product to PRODUCT table */
(
 v_prodid               NUMBER,
 v_prodprice            NUMBER,
 v_prodnum              VARCHAR2,
 v_proddesc             VARCHAR2,
 v_prodstatus           CHAR,
 v_prodbrand            VARCHAR2,
 v_prodpltwid           NUMBER,
 v_prodpltlen           NUMBER,
 v_prodnetwgt           NUMBER,
 v_prodshipwgt          NUMBER
)

IS
   -- Local variable declaration
   v_prodcount NUMBER := 0;
   v_prodid_existing NUMBER;
BEGIN
   -- Check if product with this name already exists
   SELECT COUNT (*)
   INTO    v_prodcount
   FROM    product
   WHERE   UPPER(prod_description_s) = UPPER(v_proddesc);
   -- Product does not exist
   IF v_prodcount = 0 THEN
     -- Insert row into PRODUCT based on arguments passed
     INSERT INTO product
     VALUES
           (
            v_prodid,
            v_prodprice,
            v_prodnum,
            v_proddesc,
            v_prodstatus,
            v_prodbrand,
            v_prodpltwid,
            v_prodpltlen,
            v_prodnetwgt,
            v_prodshipwgt
           );
     -- Product with this name already exists
     ELSIF v_prodcount = 1 THEN
       -- Find the product's primary key number
       SELECT prod_id_n
       INTO   v_prodid_existing
       FROM   product
       WHERE UPPER(prod_description_s) = UPPER(v_proddesc);
       -- Update the existing product with values
       -- passed as arguments
       UPDATE product
       SET    prod_price_n = v_prodprice,
              prod_description_s = v_proddesc,
              prod_status_s = v_prodstatus,
              prod_brand_s = v_prodbrand,
              prod_pltwid_n = v_prodpltwid,
              prod_pltlen_n = v_prodpltlen,
              prod_netwght_n = v_prodnetwgt,
              prod_shipweight_n = v_prodshipwgt
       WHERE prod_id_n = v_prodid_existing;
     END IF;

     -- No errors; perform COMMIT
     COMMIT;

  -- Exception section -- the execution flow goes here
  -- if an error occurs during the execution
  EXCEPTION
    WHEN OTHERS THEN
```


www.TopSage.com

```
        -- Enable standard output
        DBMS_OUTPUT.ENABLE;
        -- Put line into the standard output
        DBMS_OUTPUT.PUT_LINE('Error');
        -- Rollback all changes
        ROLLBACK;
    END sp_productadd;
    /
```

SQL*Plus 命令行可以直接编译上面的存储过程，TOAD 或 Oracle 企业管理器之类的 GUI 工具也可以完成同样的工作，然后使用下面的语法来调用：

```
EXEC SP_PRODUCTADD
    (1, 23.67, 1, 'TEST PRODUCT', 'Y', 'TEST', 1, 3, 45, 33);
```

2. IBM DB2 9.5

IBM DB2 9.5 存储过程的结构与 PL/SQL 类似，但是没有 EXCEPTION 部分。使用 DB2 SQL 过程化语言创建存储过程的语法如下：

```
CREATE PROCEDURE sp_productadd
/* This procedure adds new product to PRODUCT table */
(
 v_prodid          INTEGER,
 v_prodprice       DECIMAL(10,2),
 v_prodnum         VARCHAR(44),
 v_proddesc        VARCHAR(10),
 v_prodstatus      CHAR(1),
 v_prodbrand       VARCHAR(20),
 v_prodpltwid      DECIMAL(5,2),
 v_prodpltlen      DECIMAL(5,2),
 v_prodnetwgt      DECIMAL(10,3),
 v_prodshipwgt     DECIMAL(10,3)
)

LANGUAGE SQL
-- Transaction has to be atomic if we
-- want to be able to roll back changes
BEGIN ATOMIC
  -- Local variable declaration
  DECLARE v_prodcount INTEGER;
  DECLARE v_prodid_existing INTEGER;
  DECLARE v_result_set_end INTEGER DEFAULT 0;

  DECLARE UNDO HANDLER FOR SQLEXCEPTION
    BEGIN
      SET v_result_set_end = 1;
    END;

  SET v_prodcount = 0;
  SELECT COUNT (*)
  INTO v_prodcount
  FROM product
  WHERE UPPER(prod_description_s) = UPPER(v_proddesc);

  IF (v_prodcount) = 0 THEN
   -- Insert row into PRODUCT based on arguments passed
   INSERT INTO product
   VALUES
         (
          v_prodid,
          v_prodprice,
          v_prodnum,
          v_proddesc,
          v_prodstatus,
          v_prodbrand,
          v_prodpltwid,
          v_prodpltlen,
          v_prodnetwgt,
          v_prodshipwgt
   );
  ELSEIF (v_prodcount = 1) THEN
    -- Find the product's primary key number
    SELECT prod_id_n
    INTO v_prodid_existing
    FROM product
    WHERE UPPER(prod_description_s) = UPPER(v_proddesc);
    -- Update the existing product with values
    -- passed as arguments
    UPDATE product
```


www.TopSage.com

大家网 TopSage.com

```
    SET    prod_price_n_= v_prodprice,
           prod_description_s = v_proddesc,
           prod_status_s = v_prodstatus,
           prod_brand_s = v_prodbrand,
           prod_pltwid_n = v_prodpltwid,
           prod_pltlen_n = v_prodpltlen,
           prod_netwght_n = v_prodnetwgt,
           prod_shipweight_n = v_prodshipwgt
    WHERE prod_id_n = v_prodid_existing;
  END IF;

  -- perform COMMIT
  COMMIT;
END
@@
```

要使用 IBM DB2 的命令行处理器（Command Line Processor，CLP）工具编译上述存储过程，可以使用下面的语法（假设上面的代码保存在文件 C：\SQLBIB\DB2\SP_PRODUCT.SQL 中）：

```
db2 -td@ -vf "C:\SQLBIB\DB2_ACME\SP_PRODUCT.SQL"
```

3. Microsoft SQL Server 2008

Microsoft SQL Server Transact-SQL 存储过程也由标头和主体组成；在过程的主体中声明变量。功能与 DB2 过程相似的存储过程的语法如下：

```
CREATE PROCEDURE sp_productadd
/* This procedure adds new product to PRODUCT table */
  @v_prodid              INTEGER,
  @v_prodprice           MONEY,
  @v_prodnum             VARCHAR (10),
  @v_proddesc            VARCHAR (44),
  @v_prodstatus          CHAR,
  @v_prodbrand           VARCHAR (20),
  @v_prodpltwid          DECIMAL(5, 2),
  @v_prodpltlen          DECIMAL(5, 2),
  @v_prodnetwgt          DECIMAL(10, 3),
  @v_prodshipwgt         DECIMAL(10, 3)
AS
  -- Local variable declaration and preassignment
   declare @v_prodcount INTEGER
   select @v_prodcount = 0
   declare @v_prodid_existing INTEGER
BEGIN
  -- Begin transaction
  BEGIN TRANSACTION

  BEGIN TRY
  -- Check if product with this name already exists
  SELECT @v_prodcount=COUNT(*)
  FROM product
  WHERE UPPER(prod_description_s) = UPPER(@v_proddesc)
  -- Product does not exist
  IF @v_prodcount = 0
   -- Insert row into PRODUCT based on arguments passed
   INSERT INTO product
   VALUES
   (
          @v_prodid,
          @v_prodprice,
          @v_prodnum,
          @v_proddesc,
          @v_prodstatus,
          @v_prodbrand,
          @v_prodpltwid,
          @v_prodpltlen,
          @v_prodnetwgt,
          @v_prodshipwgt
   )

  -- Product with this name already exists
  ELSE IF @v_prodcount = 1
   -- Find the product's primary key number
   SELECT @v_prodid_existing = PROD_ID_N
   FROM   product
   WHERE  UPPER(prod_description_s) = UPPER(@v_proddesc)
   -- Update the existing product with
   -- values passed as arguments
   UPDATE product
   SET    prod_price_n = @v_prodprice,
```


www.TopSage.com

```
        prod_description_s = @v_proddesc,
        prod_status_s = @v_prodstatus,
        prod_brand_s = @v_prodbrand,
        prod_pltwid_n = @v_prodpltwid,
        prod_pltlen_n = @v_prodpltlen,
        prod_netwght_n = @v_prodnetwgt,
        prod_shipweight_n = @v_prodshipwgt
  WHERE  prod_id_n = @v_prodid_existing
      END TRY
      BEGIN CATCH
       ROLLBACK TRANSACTION
   END CATCH
  -- No errors; perform COMMIT and exit
  COMMIT TRANSACTION
END
```

您可能会发现 Oracle（或 IBM DB2）与 Microsoft SQL Server 语法之间存在一些差别，例如，在 Microsoft SQL Server 中，参数不一定包含在圆括号中，并以不同的方式进行错误处理，变量前缀为 at 符号(@)等。编译存储过程可以使用 SQLCMD 命令行工具、查询分析器或其他与 Microsoft SQL Server 兼容的工具；使用下面的语法执行这个存储过程：

```
EXEC SP_PRODUCTADD
    1, 23.67, 1, 'TEST PRODUCT', 'Y', 'TEST', 1, 3, 45, 33
```

14.3.3 删除存储过程

对于所有三个数据库而言，删除存储过程的基本语法完全一样：

```
DROP PROCEDURE [qualifier.]<procedure_name>
```

注意：在一条 DROP PROCEDURE 语句中，Transact-SQL 允许删除多个过程。必须用逗号分隔这些过程名。

14.4 用户自定义函数

用户自定义函数结合了存储过程的优势与 SQL 预定义函数的性能，能够接受参数，根据一个或多个 SELECT 语句检索的数据进行指定运算，并且直接向调用 SQL 语句返回运算结果。用户自定义函数可以是标量型的，也可以是表值型的（即可以用于 FROM 子句）。有关这些高级功能的详细讨论已经超出了本书的范围（请参考供应商的文档或 Wiley Publishing 出版的书籍《**SQL 函数**》，ISBN：978-0-7645-6901-2）。

交叉参考：内置 SQL 函数已在第 10 章讨论过了。

14.4.1 CREATE FUNCTION 语法

不同实现的 CREATE FUNCTION 语法不同。简化的 SQL 2003 以及所有“三大”SQL 过程化扩展的语法如下：

SQL:2003

```
CREATE FUNCTION <function_name>
<function_definition_includes_return_statement>
```

PL/SQL

```
CREATE [OR REPLACE] FUNCTION [qualifier.]<function_name>
({[<argument_name> IN | OUT | IN OUT <datatype>
                                    [DEFAULT <default>],...])
RETURN <datatype>
{IS | AS}
  [<variable_name <datatype> [DEFAULT <default>];],...
BEGIN
  <function_body_includes_return_statement>
[EXCEPTION
  <exception_statements>]
END;
```

IBM DB2 SQL PL

```
CREATE FUNCTION <function_name>
([<argument_name> <datatype>,...])
```


www.TopSage.com

```
RETURNS <datatype>
LANGUAGE SQL
[[NOT] DETERMINISTIC]
{MODIFIES SQL DATA |
 NO SQL |
 CONTAINS SQL |
 READS SQL DATA}
RETURN {<value> | <sql_statement>}
```

Transact-SQL

```
CREATE FUNCTION <function_name>
    ([@<parameter_name> <datatype> [ = <default>]],...)
RETURNS <datatype>
[AS]
BEGIN
   <function_body>
   RETURN <value>
END
```

14.4.2 创建一个简单的函数

与存储过程相比，函数主要的优势在于可以被调用。例如，假设这样一种情况：需要提取客户的订购总额，并且在总价格中添加州营业税。各州的营业税率互不相同。例如，在撰写本书时，加利福尼亚州的营业税率是7.25%。此外，ACME数据库未在任何地方存储订单的总价格（规范化的折中），各个订单项的价格乘以该项目的数量等于各个项目的总价格，然后再把这些价格相加，从而动态地计算出全部项目总价格。ACME的V_CUSTOMER_TOTALS视图（参阅第4章）负责计算各个项目的总价格。用户自定义函数UF_ORDERTAX需要接受两个参数：税系数和订单号，最后返回乘以这个系数后的总订购金额。

1. PL/SQL

创建用户自定义函数的PL/SQL语法与创建存储过程的大致相同。主要差别在于，返回数据类型必须在标头指定，并且需要从函数主体的某处返回一个值。在函数主体中，将忽略RETURN语句后面的所有代码。创建函数UF_ORDERTAX的代码示例如下。

对于使用SQLPLUS命令行编译的函数代码，结尾处的斜杠（/）必不可少（编译PL/SQL存储过程和触发器也有相同的要求）。使用GUI工具时，则通常可省略：

```
CREATE OR REPLACE FUNCTION uf_ordertax
(
  v_tax      NUMBER,
  v_ordnum   VARCHAR2
)
RETURN NUMBER
AS
    -- Declare local variables
    v_result NUMBER;
    v_ordamt NUMBER;
BEGIN
    -- This query performs variable v_ordamt assignment
    SELECT  total_price
    INTO    v_ordamt
    FROM    v_customer_totals
    WHERE   order_number = v_ordnum;
    -- Variable v_result is v_ordamt multiplied by tax
    v_result := v_ordamt * v_tax;
    -- Return result
    RETURN v_result;
END;
/
```

Note 注意：与存储过程相同，PL/SQL用户自定义函数同样具有可选EXCEPTION子句。上述的例子跳过了EXCEPTION部分。

前面已经提及，可以从SQL语句的内部调用函数，把值或实际的列各作为参数：

```
SELECT ordhdr_nbr_s,
       ordhdr_orderdate_d,
       uf_ordertax(1.0725, ordhdr_nbr_s) AS amt_incl_tax
FROM   order_header
   JOIN
       customer
   ON  (ordhdr_custid_fn = cust_id_n)
```


www.TopSage.com

TopSage.com

```
   JOIN
        address
   ON   (cust_id_n = addr_custid_fn)
WHERE   addr_state_s = 'CA'

 ORDHDR_NBR_S                  ORDHDR_ORDERDATE_D      AMT_INCL_TAX
 ----------------------------  ----------------------  ------------
 523774                        2002-08-21 00:00:00.000       7037.52
 523775                        2002-08-21 00:00:00.000      16461.49
 523776                        2002-08-21 00:00:00.000      13734.45
 523777                        2002-08-21 00:00:00.000      10660.65
 523778                        2002-08-21 00:00:00.000       7037.52
 523779                        2002-08-21 00:00:00.000       7037.52
 523780                        2002-08-21 00:00:00.000       7037.52
 523781                        2002-08-21 00:00:00.000      16403.34
 523782                        2002-08-21 00:00:00.000       8984.34
 523783                        2002-08-21 00:00:00.000      12927.60

10 rows selected.
```

请注意，该函数共执行了 10 次，SELECT 语句每返回一行就执行一次。

2. IBM DB2 SQL PL

DB2 SQL PL 用于创建用户自定义函数的语法与 PL/SQL 和 Transact SQL 的语法(在本章后面讨论)差别很大：

```
CREATE FUNCTION uf_ordertax
(
   v_tax      DECIMAL(12,2),
   v_ordnum   VARCHAR(10)
)
RETURNS DECIMAL(12,2)
LANGUAGE SQL NOT DETERMINISTIC
READS SQL DATA
RETURN
    -- This query performs variable v_ordamt assignment
    SELECT   total_price * v_tax
    FROM     v_customer_totals
    WHERE    order_number = v_ordnum
@
```

这个函数的编译方法与 IBM DB2 存储过程的编译方式完全相同：

```
db2 -td@ -vf "C:\SQLBIB\DB2_ACME\UF_PRODUCT.SQL"
```

然后就可以从一条 SQL 语句中调用这个函数了，使用的语法已经在前面的“PL/SQL”小节中给出。

3. Transact-SQL

用户定义函数的 Transact-SQL 语法与 PL/SQL 的非常类似。参数都包含在圆括号中，函数体则包括在关键字 BEGIN 和 END 之间：

```
CREATE FUNCTION uf_ordertax
(
    @v_tax    NUMERIC(12,4),
    @v_ordnum VARCHAR(30)
)
RETURNS NUMERIC(12,4)
AS
BEGIN
    -- Declare local variables
    declare @v_result NUMERIC(12,4)
    declare @v_ordamt NUMERIC(12,4)
    -- Assign variable @v_ordamt using SELECT statement
    SELECT @v_ordamt = total_price
    FROM v_customer_totals
    WHERE ORDER_NUMBER = @v_ordnum;
    -- Variable @v_result is @v_ordamt multiplied by tax
    SET @v_result = @v_ordamt * @v_tax
    -- Return result
    RETURN @v_result
END
```

从 SQL 语句中调用的用户定义函数必须用用户名限定，形成所谓的两部分函数名称：

```
SELECT ordhdr_nbr_s,
       ordhdr_orderdate_d,
       dbo.uf_ordertax(1.065, ordhdr_nbr_s)
FROM   order_header
   JOIN
       customer
   ON  (ordhdr_custid_fn = cust_id_n)
```


www.TopSage.com

```
    JOIN
        address
    ON  (cust_id_n = addr_custid_fn)
WHERE   addr_state_s = 'CA'
```

函数也可以用文字值作为两个参数进行调用：

```
SELECT dbo.uf_ordertax(1.065, '523774') AS ORDER_TOTAL
GO
ORDER_TOTAL
 ----------------
     7037.5200
(1 row affected)
```

14.4.3 删除用户定义函数

在所有三个数据库中，删除用户定义函数的基本语法是一致的：

```
DROP FUNCTION [qualifier.]<function_name>
```

就像在存储过程中一样，Transact-SQL 也允许多个函数在一条 DROP FUNCTION 语句中删除。

14.5 触发器

触发器是一种特殊类型的存储过程（不过从技术上说并不一定是这样），只要数据库中出现特定的事件，就会自动触发触发器。例如，可以在一行插入某个指定的表，或者在更新某个表列时调用触发器。

14.5.1 CREATE TRIGGER 语法

该语法在各个 RDBMS 产品之间各不相同，任何一个都不符合 SQL 2003 标准。

1. SQL: 2003

SQL: 2003 仅仅规定了发生 DML 事件时自动执行的触发器。语法如下：

```
CREATE TRIGGER <trigger_name>
[BEFORE | AFTER]
{INSERT | UPDATE | DELETE}
ON <table_name>
[FOR EACH ROW]
<trigger_body>
```

BEFORE 和 AFTER 关键字指定触发器是在 DML 事件实际发生之前还是之后触发。也就是说，如果指定了 BEFORE 关键字，且任何时候表 A 中只要插入一行，就执行特定的触发器，那么，在 RDBMS 试图执行这个插入前，会触发这个触发器；如果该触发器产生了一个错误，则不会执行这次插入。AFTER 触发器的触发发生在 DML 语句完成时；如果出现错误，则执行事务回滚。

有时把带有 FOR EACH ROW 选择子句的触发器称作**行触发器**。对于 DML 语句影响的每一行，行触发器都会对其执行一次；例如，如果一条 UPDATE 语句修改了 10 行的列值，则会触发这个触发器 10 次。如果省略了这个子句，无论影响到多少行，触发器只会执行一次（**表触发器**）。

2. PL/SQL

Oracle 的 PL/SQL 具备 SQL: 2003 标准要求的所有选项，另外，还有大量其他选项。下面的语法只涉及 DML 触发器子句；此外，PL/SQL 允许创建 DDL 和数据库触发器，在指定的 DDL（CREATE、ALTER、DROP 等）或数据库事件（用户登录/注销、数据库启动/关闭等）发生时，将会触发这些触发器。

```
CREATE [OR REPLACE] TRIGGER [qualifier.]<trigger_name>
{BEFORE | AFTER | INSTEAD OF}
{INSERT | DELETE | UPDATE OF <column_name>,...} [OR],...
ON <table_or_view_name>
[REFERENCING OLD AS <name_for_old> NEW AS <name_for_new>]
[FOR EACH ROW]
[WHEN <condition> <sql_block>],... |
<trigger_body>
```

正像许多其他 Oracle 对象（存储过程、函数、视图、同义词、序列等）一样，PL/SQL 语法具有一个可选的 OR REPLACE 子句（已经在第 4 章讨论过了）。

除了 BEFORE 和 AFTER 关键字，还可以指定 INSTEAD OF（仅适用于视图）；在这种情况下，Oracle 会执行触发器主体中的动作，而不会执行触发事件。不可更新视图模拟可更新视图（参阅第 4

www.TopSage.com

章）时，这一点特别有用。例如，如果视图中定义了一条 GROUP BY 或 DISTINCT 子句，就不能在其中使用 DML；不过，INSTEAD OF 触发器包含特殊的逻辑，可以使用过程化语言结构取代这条规则，因此能够执行相应的 DML 语句。

UPDATE OF 子句是另一个高级功能。不同于 SQL: 2003 标准，标准中规定被触发的表更新任何列时，才允许执行触发器，在 PL/SQL 中，可以指定更新哪一列（或一组列）时会触发触发器。

REFERENCING 子句把可选相关名指定给用于前缀列名的 OLD 和 NEW 关键字，从而相应地存取旧的（未被修改的）或新的（修改过的）列值。

其他一些 PL/SQL 高级功能包括创建多个 DML 事件的触发器和在触发器主体中为不同的事件指定不同的动作：

```
CREATE TRIGGER multi_action
BEFORE INSERT OR UPDATE ON my_table
...
WHEN INSERTING
   <code_to_execute_on_insert>
WHEN UPDATING
   <code_to_execute_on_update>
...
```

下面的例子创建一个 TRBU_PRODUCT 触发器，每当 PRODUCT 表中的 PROD_PRICE_N 值发生改变时，该触发器就把一行插入到专门的审核表 PRODUCT_AUDIT 中。信息包括被修改行的主键、新的和旧的价格、修改该记录的用户名以及时间戳。在 PRODUCT 中，每一个满足条件且被修改的行都会触发这个触发器，因此，如果某处执行一条 UPDATE 语句，更改了 5 个产品的价格，则会向 PRODUCT_AUDIT 中插入 5 五行。

```
CREATE OR REPLACE TRIGGER trbu_product
BEFORE  UPDATE OF prod_price_n ON product
FOR EACH ROW
BEGIN
    INSERT INTO product_audit
    VALUES (:NEW.prod_id_n,
            :OLD.prod_price_n,
            :NEW.prod_price_n,
            USER,
            SYSDATE);
END;
/
```

Note 注意：在上述示例中，触发器的名称扩展了在其他章节讨论的匈牙利标记法（参阅第 15 章和第 16 章）。TRBU 代表 Trigger for each Row Before Update（每行在更新前触发）。

3. IBM DB2 SQL PL

DB2 的过程化语言创建触发器的语法如下：

```
CREATE TRIGGER <trigger_name>
[NO CASCADE BEFORE | AFTER]
{INSERT | DELETE | UPDATE [OF COLUMN <column_name>,...]}
ON <table_name>
[FOR EACH ROW MODE DB2SQL | FOR EACH STATEMENT]
[REFERENCING OLD AS <name_for_old> NEW AS <name_for_new>]
<trigger_body>
```

从整体上来说，这个语法类似于 SQL: 2003 规范，只在某些次要的细节上有所不同。例如，BEFORE 还不够完善——必须指定 NO CASCADE BEFORE。同样，为了创建行触发器，必须在 FOR EACH ROW 子句中添加 MODE DB2 SQL 关键字。

Note 注意：MODE 子句指定触发器的模式；目前在 DB2 中只有 DB2 SQL 支持这一模式。

REFERENING 子句类似于 PL/SQL，但是，虽然子句本身可选，但如果想要在触发器的主体中使用，就必须为 OLD 和 NEW 指定替换名称。

下面示例中的代码创建了 TRAU_PRODUCT 触发器，其功能与本章前面 PL/SQL 例子中阐述的 TRBU_PRODUCT 触发器相同：

```
CREATE TRIGGER trau_product
AFTER UPDATE OF prod_price_n ON product
REFERENCING NEW AS NNN OLD AS OOO
FOR EACH ROW MODE DB2SQL
```


www.TopSage.com

大家网 TopSage.com

```
BEGIN ATOMIC
        INSERT INTO product_audit
        VALUES (NNN.prod_id_n,
                OOO.prod_price_n,
                NNN.prod_price_n,
                USER,
                CURRENT DATE);
END
@
```

4. Transact-SQL

下面是创建触发器的一般 Transact-SQL 语法：

```
CREATE TRIGGER <trigger_name>
ON <table_or_view>
[WITH <DML_trigger_option>]
{ FOR | AFTER | INSTEAD OF }
{INSERT | UPDATE | DELETE}
[WITH APPEND]
AS
[ENCRYPTION]
[IF UPDATE ( <column_name> )
          [AND | OR UPDATE ( <column_name> ) ],...]
<trigger_body>
```

Transact-SQL 的 FOR 等同于 SQL 2003 关键字 BEFORE。

与 PL/SQL 不同，INSTEAD OF 子句可以指定给表触发器或视图触发器。

WITH APPEND 子句指定触发器可以附加到另一个同类型的触发器，这样可以增加处理逻辑，而非取而代之。触发顺序取决于声明或创建顺序。

IF UPDATE 子句指定更新那些触发触发器的列（或列的组合）。

下面的语法创建一个触发器，其功能与前面部分中讨论过的 PL/SQL 触发器相同：

```
CREATE TRIGGER trbu_product ON product
FOR UPDATE
AS
IF UPDATE (prod_price_n)
    INSERT INTO product_audit
    SELECT  i.prod_id_n,
            d.prod_price_n,
            i.prod_price_n,
            USER,
            GETDATE()
    FROM inserted i
       JOIN
            deleted d
       ON i.prod_id_n = d.prod_id_n
```

这个示例中有几点需要解释。不同于 PL/SQL 或 SQL PL，Transact-SQL 不存在专门的 OLD 和 NEW 结构来保存表列原始的和修改过的值。作为替代方案，Microsoft SQL Server 提供了两个特殊的虚拟表：DELETED(保存 DELETE 和 UPDATE 操作期间受影响行的副本)和 INSERTED(保存 INSERT 和 UPDATE 期间受影响行的副本)。可以像上面代码中那样联接这两个表来获得要插入 PRODUCT_AUDIT 中的值。

Tip　提示：可以使用游标在 Transact-SQL 中模拟 FOR EACH ROW 功能。详细内容请参阅供应商的文档。

14.5.2　删除触发器

下面这个语法从系统目录中删除已存在的触发器定义：

```
DROP TRIGGER [qualifier.]<trigger_name>
```

所有三个数据库的语法完全相同。在 Transact-SQL 中，可以删除多个触发器，实现方式与删除存储过程、用户定义函数或任何其他数据库对象相同。

14.6　.Net 存储过程和函数

使用 SQL 过程化扩展插件的存储过程和函数都在 RDBMS 自身内部运行——即运行在 Oracle、IBM DB2 和 Microsoft SQL Server 程序的内部。相反，使用.NET 编写的过程可作为外部程序来运行。实现（比如使用 C#编写）其存储过程的全部代码可存储为.NET 组件（等同于 DLL——动态链接库），

www.TopSage.com

这些组件都存储在数据库服务器之外的文件系统中。

使用.NET 语言创建过程并编译为组件后，在 Microsoft SQL Server 2008 中，注册这个.NET 过程分为两步：

```
CREATE ASSEMBLY <assembly(DLL) name>
FROM '\\<machine name>\<path>\<assembly(DLL) name>.dll'
GO

CREATE PROCEDURE dbo.<procedure_name>
(
      @param1 INTEGER
      ,@param2 NVARCHAR(1024)
      ...
)
AS EXTERNAL NAME
  <assembly(DLL) name>.<public class>.<method>;
```

基本原理相同，但 C#代码使用 Microsoft 的 Microsoft.SqlServer.Server 库，为了使相应 RDBMS 可以访问该过程，本项目必须参考 Oracle.DataAccess 库或 IBM.Data.DB2 库。

Oracle 和 IBM 两家都提供了.NET 库，以便把用 C# 编写的过程部署在 Windows 平台上。.NET 代码在 CLR（公共语言运行时）内运行，这与 JVM（Java 虚拟机）非常相似。调用.NET 存储过程时会装载包含这些代码的组件，并在一个外部 CLR 宿主中得到执行，RDBMS 只负责调度参数并检索结果。

用.NET（C#、VB.NET）创建过程后，就必须进行安装——组件（DLL）应当与数据库一起注册，同时应在数据库上创建一个包装程序。如果供应商没有提供安装向导，这项任务很难完成。这之后，可以在数据库目录里检查过程在 RDBMS 端的 SQL 包装。

过程安装完毕后，无论这个过程是由用户定义还是由供应商提供，它在系统里的使用方式与其他任何函数或过程相同。使用外部存储过程时，理解其衍生概念相当重要，即：安全性（过程并不在 RDBMS 环境内部运行，因此可以访问某些特许资源）、性能（因为需要进行调度，这样的过程从本质上来说就比较缓慢）、可维护性（外部 DLL 会产生依赖关系）等。任何一个 RDBMS 中，使用.NET 实现存储过程的主要优点是可以利用成熟、丰富和健壮的编程语言的性能（使用 Java 实现存储过程亦有同样的好处）。

Note

注意：虽然已被采纳为 ECMA 标准，并且成为一种非常有前途的统一开发平台，.NET 在很大程度上仍然是 Windows 专用概念。因此，该过程只能在 Windows 操作系统中进行编译和调用。Mono 项目是一种新生开发方式，已经成功地把 .NET CLR（公共语言运行时）移植到 UNIX 和与 UNIX 相似的系统（Linux、FreeBSD、Solaris）。这个项目得到 Novell 和 Microsoft 以及正蓬勃发展的开放源代码社区的大力支持。虽然没有得到 RDBMS 供应商的明确支持，但应当确信，运行在 Linux 上的 Oracle 和 IBM DB2 没有理由不利用 .NET 的性能，这样做还可以得到 Mono 库的帮助。

14.7 小结

本章介绍了 SQL 过程化语言的关键概念，该语言的主要目的是修正 SQL 自身非过程化的不足。ISO/ANSI 标准提供了处理数据的 RDBMS 存储模块的框架，却没有指定实现时该使用哪种语言。根据自身的设备情况，供应商们的实现方式各有千秋——从各个供应商自己专用的 SQL 扩展插件，比如 PL/SQL（Oracle）、Transact-SQL（Microsoft SQL Server 和 Sybase），SQL PL（IBM DB2 9.5），到 Java、Cobol、.NET（C#、VB.NET）和 Visual Basic（Data Transformation Services，数据转换服务）。

利用这些语言设计 RDBMS 基本上与用任何其他的程序设计语言创建程序相同，值得注意的是，这些 SQL 扩展插件都是面向数据库的，包含许多用于特殊任务的特殊结构（如游标，可以进行逐行的数据操作）。

过程化 SQL 程序可以编译成多种模块类型：存储过程、用户定义函数和触发器。从风格上来说，它们也许有所不同，调用方法也各不相同等，但基本原理完全一致：模块经编译并存储在服务器端，其实现方式或是利用某种专用 SQL 扩展语言，或者是利用某种通用语言（比如 Java）；模块包含过程化语言和 SQL 语句的结构，且在 RDBMS 环境中得到执行。

SQL 标准不包括过程化语言（尽管其中指定了永久模块），甚至直到最近，某些流行的 SQL 产品仍未实现这种能力。

www.TopSage.com

CHAPTER 15

第 15 章
SQL 和 XML

本章提要：

- XML 标准概述
- SQL/XML 标准
- RDBMS 的 XML 支持

这部分概述无意涉及 XML 的方方面面；实际上，其目的仅在于向不熟悉 XML 的读者介绍 XML，重点讲解本书读者可能感兴趣的内容。

15.1 简介

XML 是信息交换的流行语言，正日益成为事实上的标准。虽然自诞生到如今已历经 10 个年头，它的能力仍然处于探索之中，而且得到不断的增强。

XML 的多样性来源自于其堪称经典的前身 SGML。原始的标准通用标记语言（Standard Generalized Markup Language，SGML）被视作把文本文档表示为电子形式的一种方法。它的一个显著特点是可以根据不同用途标记具有特殊标记的文本段。

可扩展标记语言（eXtensible Markup Language，XML）仍然大量使用标记——实际上比 SGML 使用得更多，然而它不再简单地在两个标记间标记文本，而是用于建立结构。此外，XML 不再像 SGML 那样局限于使用一套特别的、预先确定的标记；某个语法范畴内的任何字符串都可以视作一个标记。在 XML 中标记都应与一个结束标记相匹配；否则，一个 XML 的片断就会被称作“不完整”，XML 每个语法分析器也不能对其进行分析。

15.2 XML 的结构

完整的 XML 文档具备树形结构（或者在更加复杂的复合文档中表现为森林结构）。通常情况下，树的结点称作元素（虽然也可能是属性、注释等），而且必须仅有一个根元素。每个元素都位于一个标记和相应的结束标记之间，其中包括标记本身。

```
<tagExample>Element example</tagExample>
```

正像前例所显示的那样，标记被封装在一对尖括号中（或者在“小于号”和“大于号”之间），结束标记除了开始的那个尖括号后紧随着一根斜线“/”之外，其余部分与相应的标记相同。

```
<emptyElementTag></emptyElementTag>
```

这个示例表明在两个标记之间，一个元素可以什么也不包括——这也是允许的。对于这种情况，还有一种简洁的表达方式：

```
</emptyElementTag>
```

www.TopSage.com

在目前所涉及的范围内，这些例子是可行的，不过它们的作用范围实际上不会很大。元素必须包含某些东西才更让人感兴趣，而且也不是仅仅只包含一点点文本。在考虑包含什么之前，首先应该提到元素属性。一个元素可以包括一个或多个属性：

```
<element atrribute1="attrlValue" attribute2="attr2Value">contents</element>
```

元素标记里定义了属性，作为键值对，属性的值位于引号之间。属性与属性之间用空格隔开。

元素的内容和属性之间有什么区别？从概念上来说，这很清楚：我们不难说出一本书的内容与作者、定价以及是否值得这么多钱等特征之间的区别。然而在 XML 的实践过程中，有时这是风格或便利的问题，信息的一部分定义为元素的属性，还是定义为元素的内容；元素的内容可能比想像的要复杂得多。另外，与子元素不同，父元素的属性都应该独一无二。

实际上，一个元素也可以包含其它元素以及文本，这样就产生了结构：

```
<book>
    <name>SQL Bible</name>
    <authors>
        <author>Alex Kriegel</author>
        <author>Boris M. Trukhnov</author>
    </authors>
    <publisher>Wiley Publishing, Inc.</publisher>
    <price>$49.99</price>
</book>
```

在上述这个例子中，元素“book”包括几个元素，其中“authors”又包括多个元素（都具有相似的标记，但文本不同）。

15.2.1 格式化 XML

为了更直观地了解整个结构，习惯上打印 XML 时对其进行了分行，且首行缩进。不过对于前面的例子，真正的 XML 会在一行列出所有的内容，不能在元素之间插入空格或分行符。

```
<book><name>SQL Bible</name><authors><author>Alex Kriegel</author>
<author>Boris M. Trukhnov</author></authors><publisher>Wiley
Publishing, Inc.</publisher><price>$00.00</price></book>
```

这很像数据库中多个表的查询结果，而且这些表是一对多的关系：有来自 Book 表的记录，里面包括 columns name、publisher、price，以及一些来自 Authors 表的相应记录。

XML 的结构是**层次化**的，在许多情形下，与关系数据库很相像。一个 XML 文档就是一棵树：例如，在前面的那个片断中，<book>是父结点；<name>、<publisher>、<price>及<author>是<book>的子结点；<author>结点有两个<author>子结点。

也可以用略微不同的方式来表示数据：

```
<book name="SQL Bible" publisher="Wiley Publishing, Inc."
price="$49.99">
    <author name="Alex Kriegel"/>
    <author name="Boris M. Trukhnov"/>
</book>
```

（因为存在着两个作者，所以不能像属性一样把它们放在里面，至少没有遵从某些规则之前必须如此）。在一个元素中，属性应当具有唯一的名称。

15.2.2 DTD 和架构

XML 分析器只接收格式良好的 XML 文档。格式良好的 XML 文档必须遵循一定的规则，比如只具有一个根元素，每个标记都有一个匹配的结束标记，每一个属性都已赋值等。

然而，XML 的目的旨在定义结构；完整的 XML 文档由于谬误仍然具有让人迷惑的结构：

```
<book>
    <name>SQL Bible</name>
    <authors>
        <book>How To Make Friends And Influence People</book>
        <author>Boris M. Trukhnov</author>
    </authors>
    <publisher>Wiley Publishing, Inc.</publisher>
    <price>$49.99</price>
</book>
```

分析器将不会注意到<book>元素的错误位置，在这里它成为<author>元素的一个子元素，其中原

www.TopSage.com

因是，<book>元素已经用正确的结束标记包括起来，且没有违反任何语法规则。

为了防止此类意外发生，必须为那些**结构上**合法的文档创建验证程序。有两种途径可用于定义XML文档的结构，那就是使用DTD和XSD。

1. DTD

文档类型定义是较老的约定。它是一种定义XML文档结构的特殊语言：

```
<!DOCTYPE book [
    <!ELEMENT book (name,authors,publisher,price)>
    <!ELEMENT name (#PCDATA)>
    <!ELEMENT authors (author+)>
    <!ELEMENT author (#PCDATA)>
    <!ELEMENT publisher (#PCDATA)>
    <!ELEMENT price (#PCDATA)>
    <!ELEMENT author (#PCDATA)>
]>
```

2. XSD

XML架构定义是比较新的开发成果。因为使用XML自身而非其他语言，所以更具有吸引力；功能更强大，允许定义值和数据类型，以及其他有用的验证信息。

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   targetNamespace="http://www.wiley.com/SQLBible"
        xmlns:sql="http://www.wiley.com/SQLBible"
        elementFormDefault="qualified">
   <xsd:include schemaLocation="Book.xsd"/>
     <xs:element name="book">
       <xs:complexType>
          <xs:all>
             <xs:element name="name" type="xs:string"/>
             <xs:element name="authors" type="sql:authorInfo"/>
             <xs:element name="publisher" type="xs:string"/>
             <xs:element name="price" type="xs:integer"/>
          </xs:all>
       </xs:complexType>
   </xs:element>
<xs:complexType name="authorInfo">
        <xs:sequence>
            <xs:element ref="sql:Book" name="author" type="xs:string"
minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
   </xs:complexType>
</xs:schema>
```

XML架构正在取代DTD，成为强制性的XML结构遵守规范。

注意：XSD更加灵活。在前一个示例中，<xs:all>和DTD限定子元素只能出现一次，但XSD并没有限定它们出现的次序。

15.2.3 命名空间

敏锐的读者也许已经注意到，在前面的XSD示例中，元素标记中出现了许多冒号。这是为什么？

示例中的标记是**具有前缀的命名空间**。如果熟悉Java和.Net语言，那么对命名空间的概念应当不会陌生。在Java中，与之相同的概念是包，而.Net甚至使用了同样的名词：命名空间。

命名空间就是一个标记唯一存在的范围。您可能有两个名唤Peter的朋友。可以借助于他们的姓来区分这两个人：<Jones:Peter>或<Mackenzie:Peter>。一个学校里也可能存在两个同名的学生。校长会把他们分别叫做<Grade3:MarySmith>或<Grade5:MarySmith>。

在XML中，当同一个文档中合并多个XML片断时，命名空间尤为重要。原始的片断也许具有相同的标记，但在各自的片断里身份不同；如果把这些片断合并到一起，就会发生混淆。消除这种混淆的方法是声明一个片断里的标记都从属于同一个命名空间，而其他标记属于其他的空间。

这样，就可以把两个定义了命名空间的XML片断合并到一起：其中一个的命名空间为“MyNamespace”，另一个为“MyNamespace”。现在该怎么办？

为了避免此类激烈冲突，XML里的标记不会使用实际的命名空间，而是使用命名空间前缀。在这两上让人困惑的片断里，“MyNamespace”不是**名称**；它是一个别名。

必须解决命名空间的别名或前缀的问题。命名空间在元素标记里进行定义，并且此定义对于该元

www.TopSage.com

素及其子元素都是合法的。为了定义命名空间，应使用特殊属性 xmlns：

```
<MyNamespace:book xmlns:MyNamespace="http://www.wiley.com/SQLBible">
    <MyNamespace:name>SQL Bible</MyNamespace:name>
    <MyNamespace:authors>
        <MyNamespace:author>Alex Kriegel</MyNamespace:author>
        <MyNamespace:author>Boris M. Trukhnov</MyNamespace:author>
    </MyNamespace:authors>
    <MyNamespace:publisher>Wiley Publishing, Inc.
        </MyNamespace:publisher>
    <MyNamespace:price>$49.99</MyNamespace:price>
</MyNamespace:book>
```

有两点很重要：

1. 定义命名空间使用的 URL 并不需要分析器来处理。实际上，根本没必要是一个合法的 URL——仅仅是定义命名空间的一种方法。

2. 与“真正的”URL 不同，定义命名空间的字符串是区分大小写的。下面的字符串定义了不同的命名空间：

- www.wiley.com/SQLBible
- www.wiley.com/SqlBible
- www.Wiley.com/SQLBible

下面的字符串也可以定义不同的命名空间：

- www.wiley.com/SQL Bible
- www.wiley.com/SQL%20Bible
- www.wiley.com/~SQLBible
- www.wiley.com/%7eSQLBible
- www.wiley.com/%7ESQLBible

命名空间前缀不需要用在元素里。实际上，默认命名空间可以这样定义：

```
<book xmlns ="http://www.wiley.com/SQLBible">
    <name>SQL Bible</name>
    <authors>
        <author>Alex Kriegel</author>
        <author>Boris M. Trukhnov</author>
    </authors>
    <publisher>Wiley Publishing, Inc.</publisher>
    <price>$00.00</price>
</book>
```

在上一个示例中，请注意 xmlns 后面并没有出现冒号和别名，同时在元素标记里也没有出现命名空间前缀。

一个元素可以包含多个命名空间。

```
<book xmlns =http://www.wiley.com/SQLBible
xmlns:auth=http://www.wiley.com/authors>
      <name>SQL Bible</name>
      <auth:authors>
      < auth:author>Alex Kriegel</auth:author>
          < auth:author>Boris M. Trukhnov</auth:author>
          </auth:authors>
      <publisher>Wiley Publishing, Inc.</publisher>
      <price>$00.00</price>
</book>
```

这里有两个命名空间：默认的，以及另一个叫做 auth 的命名空间。

15.3 作为数据源的 XML

从 XML 文档里提取信息时，可以使用多个模型。目前使用最广泛的是文档对象模型（Document Object Model, DOM）。

15.3.1 在应用程序里访问 XML 文档

有两种主要方法供程序员与 XML 文档交互：DOM 和 SAX：

www.TopSage.com

- SAX（XML的简单API），实际上就是一个XML分析器：读取XML文档并在读取过程中对其进行分析。在分析过程中，会触发事件（例如，“开始元素”、“结束属性”等），然后由开发人员处理这些事件。通常情况下，处理一个事件时，某些XML文本是看不到的。
- DOM（文档对象模型），把格式良好的XML文档表示成由结点组成的树。当得到DOM文档时，所有的XML文本就已经处理完毕，且随时可用。

15.3.2 XPath

在DOM文档中，每个结点都有一个通向该结点的路径，正像文件系统中每一个文件都有一个唯一的从根目录通向该文件的路径。通向XML结点的路径使用另一种语言XPath来创建。如下所示：

```
/book/author[@name='Alex Kriegel']
```

在这个示例中，我们在寻找**book**结点的子结点**author**——book又是根结点的子结点，author有一个属性**name**，该属性的值为`Alex Kriegel`。非常类似于在不同的文件夹中存在着同名的文件，前面的路径将返回数个元素：实际上也就是Alex Kriegel撰写或合著的所有书籍。

在DOM树中，不仅元素被表示为结点，属性和其他内容（内容的值、注释）亦然。每个结点都由XPath表达式进行编址。

15.3.3 XQuery

XML文档可以自行查询。XML的查询语言称作XQuery，并且是基于XPath的。Xquery与XML的关系类似SQL与数据库的关系。XPath 2.0与XQuery同时发展，并且以兼容XPath 1.0的语法和语义为第一目标。本章的后面部分将安排许多XQuery的示例。

15.4 编码XML

在XML文档中，很多字符具有特殊含义，比如<或/。如果这些字符要成为XML文档的文本内容，必须进行“转义”，或者叫做**编码**，即，必须指示解析器忽略其特殊含义，将其视作普通字符。有两种方法实现这一点。

1. 实体

在某些情况下，只需要处理很少的特殊字符。可以代之以字符引用，或定义为实体后，再代之以实体引用（实体引用是前置和后置于实体的短名称）。

表15-1列出了一些预定义的实体。

表15-1 XML预定义实体

缩写	全名	符号
amp	和	&
apos	省略号	'
gt	右侧尖括号	>
lt	左侧尖括号	<
quot	引号	"

如果XML文档中的一些文本需要这些具有特殊意义的字符，则可以使用实体引用：

```
x &gt;10 &amp;&amp;x &lt;15
```

上一行代码翻译如下：

```
x > 10 && x < 15
```

除了以字符编码取代了名称之外，字符引用类似于实体引用，字符编码的表示法可以是十进制的，也可以是十六进制的；例如：

```
x &#62; 10 &#38; &#38; x &#60; 15
```

2. CDATA

有时文本里的这些特殊字符将导致严重的混淆，比如XML需要在XML文档里使用引号的情况。为了应付这种情况，XML定义了CDATA部分。CDATA部分以下列代码开始：

www.TopSage.com

```
<![CDATA[
```

结束为：

```
]]>
```

两者中间的任何内容都会被视作普通字符，对分析器来说不具有特别的含义。假设当前阅读的页面就采用了 XML 编码方式（可能像 Web 页面一样传送）。每一个 XML 示例都将包括在 CDATA 部分，如下所示：

```
<![CDATA[
    <book>
        <name>SQL Bible</name>
        <authors>
            <author>Alex Kriegel</author>
            <author>Boris M. Trukhnov</author>
        </authors>
        <publisher>Wiley Publishing, Inc.</publisher>
        <price>$49.99</price>
    </book>
]]>
```

15.4.1 表示 XML 文档

在互联网发展的早期，传输的信息包括了**内容**与**方式**两个方面。信息传输的语言 HTML（同样来自 SGML）其实是另一种语言的改进版本，用于描述**如何**表示文本。因此从本质上来说，这个文本可以以任何方式传输，并且用某些特殊的单词（标记）修饰，表明显示效果。

最终，**内容**与**方式**分离。CSS（Cascading Style Sheet，层叠样式表）时期到来了。CSS 也是一种语言，用于描述 HTML 文档各个组成部分（即，字体、背景颜色等）的表示方式。

在前面提及的 XML 时代，便有了**结构化**（以及半结构化）文本。结构加强了样式的功能：如今，样式指令的粒度变得更细。

XSL 和 XSLT

XSL 是一种新型的样式语言。使用 XML 自身去描述 XML 文档的各个组成部分的表示方式。XSL 语言使用 XPath（本章前面已有提及）识别 XML 文档需要表现的具体部件。XSLT 是另一种基于 XML 的语言，用于将 XML 源文档描述为另一种 XML 文档或 XHTML。

Note 注意：不同于 XSLT（“T” 代表 “转换”），XSL 常被称作 “XSL FO”（“FO” 代表 “格式化对象”）。

现在来看看 XML 文档如何转换为 XHTML 文档。在下面这个示例中，向一个完整的 XML 文档中放入了<book>片断：

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="WileyBooks.xsl"?>
<books>
    <book>
        <name>SQL Bible</name>
        <authors>
            <author>Alex Kriegel</author>
            <author>Boris M. Trukhnov</author>
        </authors>
        <publisher>Wiley Publishing, Inc.</publisher>
        <price>$00.00</price>
    </book>
</books>
```

请注意，文档的第二行（阴影部分）引用了 WileyBooks.xsl 文件中的一个样式表。

现在，来定义 XSL 样式表（突出显示了 XPath 表达式）：

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="/">
     <html>
       <body>
         <h2>Wiley Books</h2>
         <table border="0">
           <tr bgcolor="tan">
             <th align="left">Title</th>
             <th align="left">Price</th>
           </tr>
```


www.TopSage.com

```
        <xsl:for-each select="books/book">
          <tr>
            <td><xsl:value-of select="name"/></td>
            <td><xsl:value-of select="price"/></td>
          </tr>
        </xsl:for-each>
      </table>
    </body>
  </html>
 </xsl:template>
</xsl:stylesheet>
```

如果把这个样式表复制到编辑器并保存为WileyBooks.xsl，然后再复制样式表上面的XML文档并保存为WileyBooks.xml，那么就可以在IE或其他与XSL兼容的浏览器中打开这个文档并查看结果。

原始的XML文档将转换为XHTML，并显示在浏览器中。

15.4.2 XML和RDBMS

XML代表了一种开放式标准，把数据、内容和元数据（即有关数据的信息）合并在一个自描述的单一文档中，该文档可以由机器，也可以人工分析和阅读。可扩展性是其另一个特征，使它日益成为（根据某些评估是事实上的）标准的数据交换格式。

面向服务架构（Service Oriented Architecture，SOA）成为企业级开发的架构，该架构把XML格式用于服务间的通信，XML成为许多流行软件工具（Microsoft Word OOXML、Open Office ODF、Adobe FrameMaker——这些仅是其中的一小部分）的文档格式的基础，把XML与可靠的、可测试的和真正传统意义上的数据存储RDBMS无缝整合到一起，其重要性不言而喻。

存储XML是一种字符数据，使用了CHAR、VARCHAR以及CLOB/BLOB等内部数据类型，因为仅仅搜索、访问或修改单一结点时就需要对整个文档进行提取和分析，所以效率很低。下一个需要尝试的方法是拆散（分解）XML文档，并把分解后的各个部件映射到一组关系表。这样比较有效，但开销仍然很大，并不能应付复杂度迅速增加的XML标准，如今，XML已经包含递归结构以及其他不能轻易地映射为关系模型的结构。此外，分解后的XML文档已经无法再称之为一个文档，失去了其真实性（对于审核来说，这一点尤其重要），难以改变和控制。

显然，RDBMS供应商和SQL标准委员会都有必要提供对本机XML的支持。新的SQL/XML标准提供了这样一个机制，可以把XML整合进关系范例，并把XQuery语句引入SQL领域。供应商各自提供了一些SQL/XML实现，同时全面（或部分）支持标准的SQL/XML语法。

SQL标准引入了一些ISO/IEC 9075-14:2003内与XML相关的规范，然后又用ISO/IEC 9075-14:2006取而代之。根据ISO文件，它定义了向SQL数据库中导入和存储XML数据的方法，并可以在数据库中对数据进行操作，以XML的形式发布XML或常规SQL数据。此外，ISO文件提供了一种工具，允许应用程序在自身的SQL代码中整合万维网联盟（W3C）发布的XQuery和XML查询语言，从而可以同时访问常规SQL数据和XML文档。表15-2列出了SQL/XML发布函数。

表15-2 SQL/XML发布函数

SQL/XML	函数描述
Xmlelement	创建命名的XML元素（结点）
Xmlattributes	创建XML属性
Xmlroot	指定XML文档的根结点
Xmlcomment	创建XML文档的注释
Xmlpi	创建XML处理指令
Xmlparse	分析字符串并返回分析过的XML值
Xmlforest	使用表的列值创建XML值的森林
Xmlconcat	把单独的XML值连接成为一个值
xmlagg()	把包含XML值的各行聚合为XML值的森林

根据RDBMS的表PRODUCT，下面的SQL/XML查询创建了一个XML文档：

www.TopSage.com

TopSage.com

```
        SELECT
        xmlelement(name "product",
        xmlelement(name "product_id",p.PROD_ID_N),
        xmlelement(name "description",p.PROD_DESCRIPTION_S))
FROM product p
```

得到的 XML 文档的结构如下：

```
<product>
       <product_id>990</product_id>
       <description>SPRUCE LUMBER 30X40X50</description>
</product>
```

使用 XML FOREST 函数，前面的查询可能会更精确些，结果相同：

```
SELECT
xmlelement(name "product",
     xmlforest(
       p.PROD_ID_N as product_id,
       p.PROD_DESCRIPTION_S as description
             )
       )
FROM product p
```

这些基本查询所用的语法在 IBM DB 9.5 和 Oracle 11g 中是一致的，但 Microsoft SQL Server 2008 并未提供支持。

在实现同一个标准时，各个 RDBMS 供应商采用了不同的方法，因此，并不存在保证在各个 RDBMS 间移植查询的途径。在关系数据库的世界里，有关 XML 的主题极其广泛，单这一方面就需要一本书来论述（这样的书确实存在——例如，Jim Melton 和 Stephen Buxton 合著的《**Querying XML:XQuery, XPath, and SQL/XML in Context**》。此处只提供基本概念，更深层次的信息，请参阅供应商文档。

Note 注意：在这一领域还存在许多新型开发工具。其中之一是 SPARQL，一种用于 Semantic Web（与 Web 2.0 竞争时荣获亚军）的新型查询语言和数据访问协议。SPA RQL 规范由 RDF 数据访问工作组（Data Access Working Group，DAWG）开发。根据 W3C 的 RDF 数据模型对其进行了定义，并可处理任何能够映射到 RDF 的数据源。虽然这并非正式标准（尽管在 2008 年上半年极有可能发布），但任何留意下一代 Web 架构的人都应当给予适当的关注。这个规范在一定程度上遵循了为人熟知的 SQL 语法（SELECT 语句），并且能以 XML 文档的形式返回结果。

15.4.3 Oracle 11g XML DB

为了支持 XML，Oracle 开发了 Oracle XML DB，它的首次露面即伴随着旗舰性的数据库版本 9i R1（Oracle 8i 已具备一些基本的 XML 处理能力）的推出。随后的版本得到了进一步加强。最新的版本 Oracle 11g 遵循混合平台范例，为 XML 提供了健壮的支持，把关系型和本机型 XML 数据管理整合在一起。

Oracle 全面支持 XML 标准，其中包括支持命名空间、XML 架构、SQL/XML、XQuery、XSLT 和 DOM。XML 文档可以利用中间库和内建的 PL/SQL 包存储为自身的原始格式，或者分解为一组关系表。XML 文档的存储选项包括：

- 基于文本的存储（确保文档的绝对真实性，可以符合法规）。
- 基于对象的存储，允许结点级别的数据操作；通常与静态 XML 架构一起使用，并可用于数据交换配置；提供与关系型相近的性能。
- 二进制 XML 存储；用于高度动态化的文档，这种文档中的 XML 结构变化频繁。

Note 注意：任何特殊存储机制对于应用 XML 存储的应用程序都是不可知的，认识这一点非常重要。

Oracle 11g 全面支持 SQL/XML，但基于 XQuery 的处理方式提供了更大的灵活性和更快的速度。本章后面部分阐述了利用 SQL/XML 的基本查询。

下面是把数据作为字符串载入 XML 数据类型的列的基本语法：

```
UPDATE product
SET PROD_REPORT_X = XMLType
(' <productinfo>
        <name>SPF</name>
        <spec>
             <thickness>2</thickness>
```


www.TopSage.com

```
                <width>4</width>
            </spec>
            <grade type="commercial">2BTR</grade>
</productinfo>')
WHERE PROD_ID_N=990;
```

此外，可以从文件载入XML文档，也可以从HTTP流载入；对本主题的更详细的讨论，请参阅Oracle文档。

如有必要，文档可以绑定到架构（参见本章的前面部分）。XML架构通常用作验证机制，用于检证符合给定的XML架构的实例文档；当创建表或XMLType的列时，可用作约束，或用于定义XMLType实例内容的存储方式。

使用之前，必须使用DBMS_XMLSCHEMA内置包在Oracle 11g数据库中注册该架构（全局范围或局部范围），下面只显示了其语法框架：

```
DBMS_XMLSCHEMA.registerschema(
   'report.xsd',' <xs:schema> ...schema elements... </xs:schema>
   TRUE,
   TRUE,
   FALSE,
   TRUE
  );
END;
```

然后可以使用显式架构绑定来重写前面的INSERT语句：

```
UPDATE product SET PROD_REPORT_X = XMLType('
<productinfo>
        <name>SPF</name>
         <spec>
               <thickness>2</thickness>
               <width>4</width>
        </spec>
        <grade type="commercial">2BTR</grade>
</productinfo>') .createSchemaBasedXML('product.xsd')
WHERE PROD_ID_N=990;
```

Note 注意：创建架构的语法远不止于此。更多信息，请参见www.w3.org或供应商文档。

数据被加载后，就可以进行查询、修改或删除了。这些操作的语法比较简单。下面一个示例使用了XMLType的getCLOBVal方法（参见表15-3，查阅XMLType方法的完整列表）：

```
SELECT p.PROD_REPORT_X.getCLOBVal()
FROM product p;
```

输出为XML字符串。对于XMLType数据的查询，可以使用SQL/XML标准函数XMLExists、XMLQuery、XMLTable和XMLCast，也可以使用SQL/XML标准的Oracle专有实现existNode、extract和extractValue。

表15-3 Oracle 11g XMLType方法

XMLType方法	描述
CREATENONSCHEMABASEDXML	从输入基于架构的实例创建非基于架构的XML
CREATESCHEMABASEDXML	使用输入架构的URL从基于非输入架构的实例创建基于架构的XMLType实例
CREATEXML	创建和返回一个XMLType实例的静态函数
EXISTNODE	使用XMLType实例和XPath并返回1或0，表示应用XPath后是否返回结点的非空集合
EXTRACT	使用XMLType实例和XPath，应用XPath表达式，返回一个XMLType的结果
GETBLOBVAL	以BLOB的类型返回XMLType实例的值
GETCLOBVAL	以CLOB的类型返回XMLType实例的值
GETNAMESPACE	从基于架构的文档中返回一级文档的名字空间
GETNUMBERVAL	以NUMBER的类型返回XMLType实例的值。仅适用于输入XMLType实例包含纯文本结点并可转换为数字
GETROOTELEMENT	返回输入实例的根元素。如果实例是一个片断，则返回NULL
GETSCHEMAURL	如果输入是基于架构的XML，则返回XML架构的URL
GETSTRINGVAL	以字符串的形式返回XMLType实例的值
ISFRAGMENT	检查输入的XMLType实例是否为一个片断。片断也是一种XML实例，不过具有多个根元素
ISSCHEMABASED	返回1或0，表示输入XMLType实例是否基于架构
ISSCHEMAVALID	根据指定的架构URL，检查输入实例是否有效

www.TopSage.com

续表

XMLType 方法	描述
ISSCHEMAVALIDATED	根据架构检查该实例是否已验证
SCHEMAVALIDATE	根据 XML 架构检证输入实例。如果输入实例并非基于架构的，则生成一个错误
SETSCHEMAVALIDATED	为避免开销过大的架构检证，设置架构合法标志
TOOBJECT	把 XMLType 实例转换为一个对象类型
TRANSFORM	使用 XMLType 实例和一个关联的样式表(也是一个 XMLType 实例)，应用样式表并以 XML 的形式返回结果
XMLTYPE	构造一个 XMLType 数据类型的实例。构造器可以把 XML 作为 CLOB 或 VARCHAR2 导入，也可以作为对象类型导入

XMLEXISTS SQL 函数用于验证具体的元素、属性或值的存在。例如，为了确定存储在 PROD_REPORT_X 列中的 XML 文档是否包括产品名为 F/L 的信息，可以使用下面的查询：

```
SELECT PROD_REPORT_X
  FROM product
  WHERE XMLExists('/productinfo[name="F/L"]'
                  PASSING PROD_REPORT_X);
```

EXISTNODE 和 EXTRACTVALUE SQL 函数是 Oracle 专有的。前者检查给定的 XPath 表达式是否至少指向一个 XML 元素结点或文本结点，后者根据 XMLType 实例的 XPath 求值结果返回相应的 SQL 标量值。虽然不是标准功能，但如果需要解析 XML 文档，且只需要选择特定结点表示的记录时，使用这些函数非常方便。

```
SELECT
extractValue(PROD_REPORT_X, '/productinfo/name')          name,
            extractValue(PROD_REPORT_X, '/productinfo/spec/thickness')
            width,
            extractValue(PROD_REPORT_X, '/productinfo/spec/width')
dimension,
    CASE
         WHEN existsNode(PROD_REPORT_X, '/productinfo/grade') = 1
           THEN extractValue(PROD_REPORT_X, '/productinfo/grade')
grade
            ELSE 'no grade assigned'
       END "grade",
       PROD_DESCRIPTION_Sdescription
FROM product
    ORDER BY extractValue(PROD_REPORT_X,'//name') DESC;
```

EXTRACT SQL 函数用于从 XPath 表达式识别过的文档中提取元素或元素的集合：

```
SELECT extract(PROD_REPORT_X, '/productinfo/grade') "grade"
  FROM product
  WHERE existsNode(PROD_REPORT_X, '/productinfo[name="F/L"]') = 1;
```

XMLCAST SQL 函数将第一个参数转换为第二个参数指定的 SQL 标量数据类型。第一个参数是需要求值的 SQL 表达式。第二个参数可以使用的数据类型有 NUMBER、VARCHAR2 或任何日期和时间数据类型。下面的查询仅返回 PROD_REPORT_X 列 XMLType 的值，该列的<name>子元素的值为 SPF：

```
SELECT XMLCast(
XMLQuery('/productinfo/name' PASSING PROD_REPORT_X
RETURNING CONTENT)
AS VARCHAR2(1000)) "product_name"
FROM product
  WHERE XMLExists('/productinfo[name="SPF"]'
                  PASSING PROD_REPORT_X);
```

EXTRACTVALUE SQL 函数返回从 XML 文档提取的值，该文档存储于 XMLType 列：

```
SELECT extractValue(PROD_REPORT_X, '/productinfo/name') "name"
  FROM product
  WHERE XMLExists('/productinfo/name'
                  PASSING PROD_REPORT_X);
```

更新存储为 CLOB 值（非结构化存储）的非基于架构的 XML 文档时，将更新整个 XML 文档（即替换整个文档）。更新存储为二进制 XML 文档时，将会在粒度更细的层次上进行，即元素和属性自身。有几个 SQL 函数可用于增量更新 XML 数据——即替换、插入或删除 XML 数据，不需要替换周围全部的 XML 文档。

- updateXML：替换 XML 元素。下面的查询返回<grade>元素被替换为不同的值的 XML 文档。当然，基础数据并未受本次操作的影响。只有返回的 XML 经过修改：

www.TopSage.com

大家网 TopSage.com

```
SELECT updateXML(PROD_REPORT_X,
'/productinfo/grade', '<grade>NSP</grade>') AS XML
FROM product p;
```

✦ insertChildXML：在给定的元素结点中插入XML元素或属性元素作为子结点：

```
UPDATE product
  SET PROD_REPORT_X =
      insertChildXML(PROD_REPORT_X,
                     '/productinfo/spec',
                     'length',
                     XMLType('<length>
                               <bf>1000</bf>
                               <ln>1000/12*144</ln>
                             </length>'))
WHERE PROD_ID_N=990;
```

✦ insertXMLbefore：在指定的元素中（除了属性结点之外）即时插入任何类型的XML元素：

```
UPDATE product
  SET PROD_REPORT_X =
      insertXMLbefore(
         PROD_REPORT_X,
     '/productinfo/spec',
     XMLType('<grade type="commercial">
     <description>3 or better</description>
<abbreviation>3BTR</abbreviation>
</grade>')
   )
WHERE PROD_ID_N=990;
```

✦ appendChildXML：在指定的元素中添加任何类型的XML元素作为最后一个子元素。下面的查询插入一个新结点<grade>，同时插入的还有作为<productinfo>的子结点的属性：

```
UPDATE product
  SET PROD_REPORT_X =
      appendChildXML(PROD_REPORT_X,
                     '/productinfo',
                     XMLType('<grade type="commercial">3BTR ↵
                     </grade>'))
WHERE PROD_ID_N=990;
```

✦ deleteXML：删除任何类型的XML结点。例如，下面的查询从product ID = 990删除结点<grade>。

```
UPDATE product
  SET PROD_REPORT_X =
      deleteXML(PROD_REPORT_X, '/productinfo/grade')
WHERE PROD_ID_N=990;
```

对于“普通”数据类型的列，Oracle允许对XML表和列进行索引操作。这个语法是标准CREATE INDEX语句的一部分。XML索引可以按照XML数据类型加速查询的执行，不过，要想正确使用索引，需要深入而全面地理解XML和索引操作。

例如，为了创建PRODUCT表中的唯一索引，且约束设定为唯一的名称，则可以使用下面的查询：

```
CREATE UNIQUE INDEX xml_fun_idx ON product
   (extractValue(PROD_REPORT_X, '/productinfo/name'));
```

这个索引不仅改善了在<name>元素中的检索，而且在整个文档范围内增加了其唯一性。

Oracle XML DB同时提供了一个特殊的XMLIndex结构，专门用于非结构化的和混合存储类型的XML数据。与B树索引不同，它定义为一个特殊的数据库列，该列表示独立的XML元素或属性，默认情况下，XMLIndex索引非常常规。除非指定了较小的适用范围（明确指定索引中是否使用XPath表达式），它适用于XML数据的所有XPath表达式。创建XMLIndex的语法稍有不同（可以变得更加复杂）。

```
CREATE INDEX xmlindex_idx ON product
       (extract(PROD_REPORT_X, '/productinfo/name'))
INDEXTYPE IS XDB.XMLIndex;
```

所有与索引有关的操作（比如ALTER和DROP）都完全适用于XML索引。本主题的更多信息，请参考第4章。

本章仅涉及Oracle 11g实现的最基本的XML支持；要了解详细内容，请参阅供应商文档。

注意：通过数据库本机Web服务，可以了解作为Web服务的PL/SQL存储过程、函数和包，同时包括支持执行动态SQL查询和XQuery表达式（数据库HTTP服务器，作为Oracle XML DB知识库的一部分）的Web服务。因为XML是Web服务的通用语言，本机XML数据类型提供了设计SOA系统的独特的机会。

www.TopSage.com

15.4.4 IBM DB 9.5 pureXML

IBM DB 9.5 数据库中的 XML 功能由 pureXML 插件实现，同时该插件本身支持一种 XML 数据类型（并非文本或分解成一组关系型数据表），并全面集成到 DB2 架构中。支持 XQuery、SQL/XML 或两者都支持；存储着架构和 DTD 的 XML 架构知识库支持 XML 验证。为了扩展 SQL 对 XML 数据类型的支持，增加了一组 SQL/XML 函数（更多详情，请参阅第 10 章）。本章后面部分阐述了使用 SQL/XML 的基本查询。

Note 注意：IBM 提供了许多支持 XML 数据类型的 ETL 工具，以及一个用于批操作的修改过的 API（db2Import、db2Export 和 db2Load）。

在 IBM DB 9.5 数据库中，XML 数据类型的 INSERT 语法更直观：对字符串无须进行任何特别修改，就能够被 XML 识别。唯一的要求是必须完整：

```
INSERT INTO product (PROD_REPORT_X) VALUES ('
<productinfo>
       <name>SPF</name>
        <spec>
             <thickness>2</thickness>
             <width>4</width>
        </spec>
        <grade type="commercial">2BTR</grade>
</productinfo>');
```

本书的 ACME 数据库中包含 XML 列被设为 NULL 的数据。可以使用更新语法操作其中的数据：

```
UPDATE product SET PROD_REPORT_X = ('
<productinfo>
        <name>SPF</name>
         <spec>
               <thickness>2</thickness>
               <width>4</width>
        </spec>
        <grade type="commercial">2BTR</grade>
</productinfo>
')
WHERE PROD_ID_N=990;
```

更新 XML 列时需要一定的技巧，因为其中有许多选项需要考虑：更新硬编码的值，更新参数化的值，更新带有架构验证的值，以及更新存储在列中的 XML 文档的自身结构。在此我们将只讨论最常见的方案；更深层次的讨论，请参考 IBM 文档。

使用架构验证 XML 文档是一个良好的习惯。IBM DB2 9.5 提供了注册 xsd 架构的工具（内建于 SQL 语法），操作 XML 文档时即可使用。例如：

```
REGISTER XMLSCHEMA 'http://acme.org'
FROM 'file:///c:/SQLBible/xml/product.xsd' AS acme.product
COMPLETE;
```

注册架构后，便可使用下面的 INSERT 语句：

```
INSERT INTO product (PROD_REPORT_X)
VALUES (XMLVALIDATE (XMLPARSE
(DOCUMENT
<productinfo xmlns="http://acme.org">
        <name>SPF</name>
         <spec>
               <thickness>2</thickness>
               <width>4</width>
        </spec>
        <grade type="commercial">2BTR</grade>
</productinfo>' PRESERVE WHITESPACE )
ACCORDING TO XMLSCHEMA ID acme.product))
```

下面的查询用于在特定的列结点永久更新信息：

```
UPDATE product
SET PROD_REPORT_X = xmlquery( 'transform
         copy $new_doc := $PROD_REPORT_X
              modify do replace value of
   $new_doc/productinfo/grade[@type="commercial"] with "3BTR"
              return $new_doc')
WHERE PROD_ID_N = 990;
```

显然，基本元素是一个 XMLQUERY 函数，允许嵌入 XQuery 表达式。转换从可选关键词“transform”开始，后跟 COPY、MODIFY 和 RETURN。MODIFY 后面的操作有：replace、delete 和 insert。

www.TopSage.com

本语法的某些元素（例如 TRANSFORM 关键词）与 XQuery 开发标准（更新工具）一致，该标准将于2008年结束，不过，这个查询中的其他一些语法可能不会结束。

另一个选项是在查询中（不会改变列中的原始值）选择数据的时候，动态地进行更新/转换：

```
SELECT xmlquery('transform
copy $new_doc:= $PROD_REPORT_X
modify do replace value of $new_doc/productinfo/grade  with "2BTR"
return $new_doc')
FROM product
WHERE PROD_ID_N = 990;
```

通常情况下，特别是在存储过程和应用程序指定的操作中，需要传递参数化的值去更新 XML 列的内容：

```
UPDATE product
SET PROD_REPORT_X = xmlquery( 'transform
         copy $new_doc:= $PROD_REPORT_X
         modify do replace value of $new_doc/productinfo grade with
         $val
         return $new_doc' passing cast(? AS VARCHAR(10)) AS "val")
WHERE PROD_ID_N = 990;
```

可以验证数据的更新是否符合与此文档相关的架构中设定的规则。把验证处理转移到 DB2 数据库，可以使用下面的语法（假设已预先创建并注册了架构）。

```
UPDATE product
SET PROD_REPORT_X = (xmlquery( 'transform copy $new_doc
:= $PROD_REPORT_X
modify do replace value of
$new_doc /productinfo/grade with "3BTR"
return $new_doc')
according to xmlschema id "acme.product_schema")
WHERE PROD_ID_N = 990;
```

删除整个 XML 文档与删除任何其他数据类型相同，但删除文档中的属性就大不相同了。下面的查询从文档中删除 TYPE 属性。

```
UPDATE product
SET PROD_REPORT_X = xmlquery( 'transform
        copy $new_doc := $PROD_REPORT_X
        modify do delete $new_doc /productinfo/grade/@type
        return $new_doc')
WHERE PROD_ID_N = 990;
```

可以使用下面的语法重命名文档中（示例中跟在 prod_grade 的 pgrade 后面的部分）的一个元素结点（或属性）：

```
UPDATE product
SET PROD_REPORT_X = xmlquery( 'transform
copy $new_doc := $PROD_REPORT_X
modify do rename $new_doc/productinfo/grade as     "prod_grade"
return $new_doc')
WHERE PROD_ID_N = 990;
```

对 XML 数据类型的操作扩展了增加/移除文档元素的功能。本质上是使用相同的语法把整个结点结构替换为新的结构。

```
UPDATE product
  SET PROD_REPORT_X = xmlquery('
  copy $new_doc := $PROD_REPORT_X
  modify do replace $new_doc/productinfo/prod_grade
  with <prod_grade>customer</prod_grade>
  return $new_doc')
WHERE PROD_ID_N = 990;
```

增加新结点结构要求在文档中能够精确定位。可以把一个元素（INTO）插入另一个元素，或者把一个元素 AS FIRST|LAST INTO 添加到另一个元素，使这个新元素分别成为已有元素的第一个或最后一个结点。

Note 注意：使用圆括号把 modify 里的多个 do 动作括在一起，成为一组，则可能对一个单一的 XQuery 表达式进行多个操作。

在层次结构中，可以指定一个新元素插入前面或后面的位置。例如，下面的代码插入一个新结点<product>，并明确地把结点<productinfo>定义为它的父结点：

```
UPDATE  product
SET PROD_REPORT_X = xmlquery('copy $new_doc := $PROD_REPORT_X
        modify
```


www.TopSage.com

TopSage.com

```
        do insert <product type="custom">block</product>
        into $new_doc/productinfo
        return $new_doc' )
where PROD_ID_N = 990;
```

更新 XML 文档时，可以指定数据库中其他列的值作为查询的一部分。下面的示例把 PROD_ID 属性添加到 XML，使用了来自 PROD_ID_N 列的值：

```
UPDATE poduct
SET PROD_REPORT_X = xmlquery(
'transform copy $new_doc := $PROD_REPORT_X
modify do insert attribute PROD_ID {$PROD_ID_N }
into $new_doc/productinfo
return $new_doc' );
```

上述的查询将更新所有行级别上的 XML 文档，以相应的 PROD_ID_N 作为属性插入<productinfo>元素中。

需要记住的一个重要事项是，处理 XML 数据类型时，文档必须完整；否则，IBM DB2 将抛出许多错误并允许查明错误的来源。

修改重复的结点（此时，需要 FLWOR，读作“flower”）XQuery 语法时，需要特别地小心，FLWOR 在 XQuery 里是最常见的表达式。FLWOR 用于 For、Let、Where、Oder by 以及 Return。这些关键词允许在 XQuery 中构建循环并有条件地执行。

Note 注意：XSLT（参见本章前面的“表示 XML 文档”的部分）可用于转换 XML 文档。因为 XSLT 在 XML 里得到表达，所以也可以存储在数据库的 XML 类型的列。

可以索引 DB2 9.5 的 XML 列，以便改善其性能。虽然类似于对其他任何本机数据类型索引的创建，XML 列仍然需要某些不同的方式来创建索引。不同于其他数据类型的索引，XML 列可以对 XML 文档中的部分数据创建索引。

要指定索引 XML 文档的哪一部分，可以使用某种形式的 XPath，同时指明用在索引中的数据类型。

Note 注意：在创建索引的过程中，如果某个元素无法转换为指定的数据类型，那么就不会创建索引目录，也不会返回错误。

只有单一的 XML 文档可以创建索引；不支持复合索引。同时，对于单一的 XML 文档可以创建多个不同的索引。

例如，下面的命令为 PRODUCT 表的 XML 列 PROD_REPORT_X 创建了唯一索引：

```
CREATE UNIQUE INDEX prod_report_idx ON product(PROD_REPORT_X)
   GENERATE KEY USING XMLPATTERN
  '/productinfo/@ PROD_ID'
AS SQL DOUBLE
```

索引的属性是 PROD_ID（请注意它是区分大小写的），可以加速对相关文档的检索。

如果要论述 IBM DB 9.5 的全部 XML 相关功能需要较大篇幅；本章仅按需涉及了与 SQL 标准直接相关的最基本的功能；更多详情，请参阅 IBM 文档。

15.4.5 Microsoft SQL Server 2008

当计算机领域的考古学家们挖掘到 SQL Server 更古老的地层时，他们找到了一些 XML 早期应用的迹象，大约在 2000 年。Microsoft SQL Server 2000 致力于把 XML 数据映射为一组关系表。在这个版本中，Microsoft 的 SQL 变种引入了几个新关键字，实现了 XML 文档中的存储和检索，这些关键字为：FOR XML、AUTO、EXPLICIT、RAW、XMLDATA、ELEMENTS、ROWSET、OPENXML，以及大量存储过程，用于处理 XML 文档。

1. SQL Server 2000

SQL Server 2000 定义了两个处理 XML 的关键词：FOR XML 和 OPENXML。

FOR XML 用在 SELECT 语句中，指示 SQL Server 以 XML 格式获取查询结果：

```
SELECT PROD_ID_N as ID, PROD_PRICE_N as "Price"
FROM product
FOR XML RAW
          <row ID="990" Price="18.24" />
        <row ID ="1880" Price="33.28" />
SELECT PROD_ID_N as ID, PROD_PRICE_N as Price
FROM product
```


www.TopSage.com

```
FOR XML AUTO

      <product ID="990" Price="18.24" />
    <product ID ="1880" Price="33.28" />
```

RAW、AUTO、EXPLICIT 和 PATH 模式决定结果 XML 文档的格式（请注意，除了外部文件或字符串外，这里未提及存储 XML 文档的概念）。RAW 模式使每一行成为单一的元素，表的列定义为属性；如果不考虑参与查询的表的数量，这个方法比较管用。AUTO 模式根据查询中涉及的父/子表的层次结构格式化文档；单一的表生成的结果其实 RAW 模式的处理结果相同。

EXPLICIT 模式把执行查询所取得的行集转换为 XML 文档。为了使 EXPLICIT 生成 XML 文档，行集需要具有指定的格式。PATH 模式与内嵌 FOR XML 查询能力一起，以一种极其简洁很多的方式提供了与 EXPLICIT 模式一样的灵活性，但同样要求数据集遵守严格的规格。第一列必须命名为 TAG，且只能包含非负整数；第二列必须命名为 PARENT，且必须是 NULL 或非负整数；后面的其他列格式必须为 TAGNAME! TAGID! ATTRIBUTENAME[! ..]，其中 TAGID 应是一个正整数。考虑下面的查询：

```
SELECT 1    as Tag,
       NULL as Parent,
       PROD_ID_N as [element!1!PROD_ID_N],
       PROD_PRICE_N as [element!1!PROD_PRICE_N]
FROM product FOR XML EXPLICIT

<element PROD_ID_N="990" PROD_PRICE_N="18.24" />
<element PROD_ID_N="1880" PROD_PRICE_N="33.28" />
```

OPENXML 的功能却与之相反：从已有文件系统的 XML 文档中创建行集。当需要动态生成 XML 文档时，该函数将与内置存储过程 sp_xml_preparedocument 和 sp_xml_remove document 结合使用。前者创建实际的文档，此文档可以使用 OPENXML 函数装载，后者移除不再需要的文档。使用这些过程的限制颇多；更多详情，请参见供应商文件。

除了完整且新颖的本机 XML 支持之外，SQL Server 2005 和 2008 保留（为了向后兼容）并扩展了对这种方法的支持。使用更加符合标准的 XPath/XQuery 功能看起来更安全。

SQL Server 2008 的各个组件已经集成了对 XML 的支持，这些支持包括下列内容：

- 支封 xml 数据类型。
- 按照存储在列中和 xml 类型变量中的 XML 数据指定 XQuery 查询的能力。
- 增强型 OPENROWSET 允许批量装载 XML 数据。
- 增强型 FOR XML 子句和 OPENXML 函数。

SQL Server 2005 引入本机 XML 数据类型，后来在 SQL Server 2008 中得到进一步的强化。这个新数据类型允许存储并操作 XML 文档和片断；同时，根据 XQuery 标准实现了一组方法。SQL Server 2005 以及后来的版本允许类型化和非类型化 XML 的使用。如果没有为 XML 数据指定架构，或者因为某些原因（例如，XML 数据包含未得到支持的组件）不希望对进行架构验证，就可以使用非类型化 XML。类型化 XML 允许充分使用架构验证、存储，并根据数据类型对查询进行优化。类型化 XML 用于列、参数和变量时，可包含 XML 内容。

XML 数据可以从文件（BULK 导入/导出能力）或标准的 SQL 查询中的字符串输入 XML 数据类型的列。例如：

```
UPDATE product SET PROD_REPORT_X ='
<productinfo>
       <name>SPF</name>
        <spec>
             <thickness>2</thickness>
             <width>4</width>
       </spec>
       <grade type="commercial">2BTR</grade>
</productinfo>'
WHERE PROD_ID_N=990;
```

可以使用 XML 架构验证 XML，XML 架构在当前数据库中使用新的 CREATE XML SCHEMA COLLECTION 语句创建。然后，XML 数据类型便可以依赖于这个特定的架构：

```
CREATE TABLE PRODUCT(
    PROD_ID_N           INTEGER NOT NULL,
                . . .
    PROD_REPORT_X       XML (<XML schema name>),
```


www.TopSage.com

```
)
GO
```

2. XML DML

SQL Server 2008 提供了 XML DML，用于操作包括在 XML 数据类型字段中的 XML 文档。XML DML 在 XQuery 中增加了下面这些区分大小写的关键词：insert、delete 和 replace value of；它们与 XML 数据类型本身实现的 modify() 方法一起使用（XML 数据类型方法的完整列表，请参见表 15-4）。

表 15-4　　SQL Server 2008 XML 数据类型方法

数据类型方法	描述
Query	根据 xml 数据类型实例指定 XQuery。结果是 xml 类型。该方法返回一个非类型化 XML 的实例
Value	根据 XML 执行 XQuery 并返回 SQL 类型的值。该方法返回一个标量值
Exist	返回表示下列条件的一个比特位：1，表示 True，假如一个查询中的 XQuery 表达式返回一个非空结果（至少为一个 XML 结点）；0 表示 False，假如返回一个空结果，如果查询所执行的 xml 数据类型实例包含 NULL，则返回 NULL
modify	修改 XML 文档的内容。使用此方法修改 xml 类型的变量或列的内容。本方法使用 XML DML 语句进行插入、更新或从 XML 数据中删除结点。xml 数据类型的方法仅能用于 UPDATE 语句的 SET 子句
nodes	把 xml 数据类型实例分解为关系数据时，该方法极为有用。它允许确定映射到新行的结点

向存储在 XML 数据类型的列中已有的 XML 文档插入数据/结点的语法非常简单：

```
insert
     XQuery Expression|XML Instance
(
      {as first | as last} into | after | before
      XQuery Expression|XML Instance
   )
```

例如，向本章创建的 XML 文档中插入一个新元素，可以使用下面的查询：

```
UPDATE product
   SET PROD_REPORT_X.modify(
   'insert <bf>1000</bf> as first
   into(/productinfo/spec)[1]')
WHERE PROD_ID_N=990;
```

这样就创建了一个新元素<bf>1000</bf>，并使它成为父元素<spec>的第一个子结点。

Note　注意：XQuery 表达式[1]末尾的规范确保了单一的目标元素——也是第一个，虽然在文档中存在着多个元素。

可以使用下面的语句插入元素的属性：

```
UPDATE product
SET PROD_REPORT_X.modify(
         insert attribute type{"internal"}
              into(/productinfo/spec)[1]')
WHERE PROD_ID_N=990;
```

使用类似的方法可以插入处理说明、注释等。同一个查询中可以完成多个插入：

```
UPDATE product
SET PROD_REPORT_X.modify
('
          insert(
                <bf>1000</bf>,
                <ln>100</ln>)
              into(/productinfo/spec)[1]')
WHERE PROD_ID_N=990;
```

在 XML 文档中替换值同样非常透明，也可以在 XML 数据类型的 modify() 方法中完成。例如，想要替换<name>元素的值，可以使用下面的语句。

```
UPDATE product
SET PROD_REPORT_X.modify
('replace
       value of (/productinfo/name/text())[1]
       with  "F/L"')
WHERE PROD_ID_N=990;
```

Note　注意：插入/替换/删除必须使用 IF 表达式，并在 XQuery 表达式的特定条件下进行。

可以使用下面的查询修改元素的属性：

```
UPDATE product
  SET PROD_REPORT_X.modify
```


www.TopSage.com

大家网 TopSage.com

```
('replace
          value of (/productinfo/grade/@type)[1]
          with "custom"')
WHERE PROD_ID_N=990;
```

至于删除，只需要使用modify()方法中的delete XQuery表达式即可。例如，下面的查询从XML文档中删除<grade>元素：

```
UPDATE product
  SET PROD_REPORT_X.modify
          ('delete(/productinfo/grade)[1]')
WHERE PROD_ID_N=990;
```

删除属性的方法如下：

```
UPDATE product
  SET PROD_REPORT_X.modify
          ('delete/productinfo/grade/@type')
WHERE PROD_ID_N=990;
```

从XML文档中选择信息，可以使用XML数据类型实例的query()方法（参见表15-3）。例如，下面的查询只提取<spec>的信息。

```
SELECT PROD_REPORT_X.query('/productinfo/spec')
FROM product
WHERE PROD_ID_N=990;
```

从一个属性或元素中提取单一值，可以使用value()方法。下面的查询提取第一个<width>元素的值，并把它转换为整数：

```
SELECT
  PROD_REPORT_X.value
            ('(/productinfo/spec/width)[1]', 'int' )
FROM product
WHERE PROD_ID_N=990;
```

转换为XML数据类型不允许公共语言运行时（CLR）用户定义类型或image、text、ntext或sql_variant数据类型，不过，允许使用其他任何内置SQL和用户定义的数据类型。

要使用包含在XML文档中的值，SQL Server 2008为该XML数据类型提供了exist()方法。例如，下面的查询返回产品的所有产品ID，该产品在PROD_REPORT_X文档的<name>元素中有一个值SPF。

```
SELECT PROD_ID_N
FROM product
  WHERE PROD_REPORT_X.exist
  ('/productinfo[name="SPF"]')=1
```

可以索引SQL Server 2008中的XML数据类型的列，以增强其性能。索引可以分为两大类：主要索引和辅助索引。

Note 注意：可以为XML文档创建全文索引，索引XML值的内容，但忽略XML标记。

主要索引应用于XML数据类型的列中所有的元素、值和XML实例中的路径。为了创建一个XML主要索引，包含XML列的表必须在表的主键上具备聚集索引。创建索引的语法必须遵循CREATE INDEX语句的通用规则：

```
CREATE PRIMARY XML INDEX
report_idx on product (PROD_REPORT_X)
```

创建XML主要索引后，可能希望创建数个XML辅助索引，以增强检索性能。XML辅助索引可以是PATH、VALUE或PROPERTY类型。创建这些XML索引类型所用的语法与创建主要索引的语法相同。

```
CREATE XML INDEX report_idx_path ON product(PROD_REPORT_X)
USING XML INDEX report_idx FOR PATH
GO

CREATE XML INDEX report_idx_value ON product(PROD_REPORT_X)
USING XML INDEX report_idx
FOR VALUE
GO

CREATE XML INDEX report_idx_property ON product(PROD_REPORT_X)
USING XML INDEX report_idx
FOR PROPERTY
GO
```

Note 注意：查询NFORMATION_SCHEMA视图sys.xml_indexes可以得到XML索引信息。

删除和修改XML索引与操作Microsoft SQL Server 2008支持的其他任何索引类型相同。

www.TopSage.com

这里所给出的信息和示例仅涵盖 SQL Server 2008 中 XML 实现的基础。对该主题的深入讨论，请参考 Microsoft 文档。

15.5 小结

三大 RDBMS 都支持 XML 数据类型和相关的操作以及 SQL 2006 标准定义的大多数功能。

各个 RDBMS 实现在细节上有差异。

在 RDBMS 中，XML 文档可存储为多种格式，其中包括分解后适用于关系表的非结构化字符串，以及本机 XML 数据类型。

存储、检索和操作 XML 信息包括理解 XQuery、XPath 和 SQL/XML 概念，XML 操作可以在细化级别执行——即 XML 元素和属性级别。

可以索引 XML 数据类型的列，与其他内置数据类型列相同。

甚至就其本身来说，XML 也是一个范围甚广的主题，有关这个主题的信息可以参考许多资源，其中包括前述的 `www.w3.org`，这是一个“绝对可靠”的站点，提供原始的规范（虽然并不总是易于理解的）。其他资源包括供应商文档、各种博客和网页，当然也包括书籍。

www.TopSage.com

CHAPTER 16

第 16 章
SQL 和过程化程序设计

本章提要：

- SQL 语句处理步骤
- 嵌入式 SQL 和动态 SQL
- SQL/CLI 标准
- 开放式数据库连接和数据库对象连接与嵌入
- SQL 和 Java
- Oracle API 选项
- IBM DB2 调用级接口
- Microsoft 数据访问接口

我们已经知道，SQL 是一种非过程化语言。在前一章的学习中，已经讲述过如何使用专有 SQL 过程化扩展创建过程化程序。RDBMS 内部存储这些程序（存储过程、用户自定义函数、触发器等）。尽管这种方式自 20 世纪 90 年代以后非常流行，但并非首次授权 SQL 过程化语言功能。在 SQL 过程化扩展得到发展之前很久，就出现了嵌入式 SQL 的想法。IBM 在 20 世纪 80 年代初引入这个概念，后来在许多其他的 SQL 供应商中得以实现。从逻辑上讲，动态 SQL 是嵌入式 SQL 原理的延续，消除了后者的一些局限性和不便之处。

在执行单一 SQL 语句之前，提交这个语句的客户端应用程序必须以某种方式连接 RDBMS。可以通过多种机制建立连接——范围从专有调用级接口到 Java 数据库连接（JDBC）和 ADO.NET。

虽然严格地讲不是 SQL 的领域，但是，理解应用程序编程接口的概念对于学习 RDBMS 及本机 SQL 非常重要。

16.1 SQL 语句的处理步骤

每一个 RDBM 的 SQL 语句的内部处理机制都不尽相同，但总体上来说，一个 SQL 语句的处理需要涉及 5 个步骤：

1. RDBMS 把 SQL 语句分解成单独的单词（令牌），对其进行**分析**，并验证语句的语法。其目的是确保语句中不存在输入错误和逻辑错误（如果有错误存在，处理将会停止，从而节省时间和系统资源）。例如，下面的语句在第一个步骤中就会被拒绝，因为第一个语句拼错了关键字 SELECT，第二个语句中的 FROM 和 WHERE 子句的顺序不正确：

```
SELCT *
FROM CUSTOMER

SELECT *
WHERE CUST_ID_N = 65
FROM CUSTOMER
```

因为无需真正地访问数据库，在通常情况下，分析应当是最迅速的步骤。

Note

注意：任何 SQL 语句通常由三类令牌组成：关键字（如 SELECT、UPDATE、WHERE 等）、表、视图、约束或其他数据库对象名和列名；分配给列或者用于 WHERE 和 HAVING 子句谓词的值（或者这些值的特殊变量，在运行时，可以用实际的值替换）。此处所描述的处理步骤 1 中，仅仅第一组令牌通过了验证。

www.TopSage.com

2. 接着 RDBMS 根据信息模式检查语句验证该语句。在这个步骤中，实际上如果确定数据库中存在所有的表和列名，则可通过第二组令牌的验证。此外，还检查执行该语句的用户特权，并且确保列名不存在二义性等等。例如在步骤 2 中，下面的语句将无法通过（假使试图在 ACME 示例数据库上执行），因为 CUSTOMER 表中不存在列 CUST_ID：

```
UPDATE CUSTOMER
SET CUST_NAME_S = 'THE BEST CUSTOMER'
WHERE CUST_ID = 65
```

3. 在下一个步骤中，将对语句进行**优化**。不同的 RDBMS 供应商的查询优化处理之间存在着显著的差别，但思想却是一致的：找出执行 SQL 语句的最有效途径。例如，将要确定以什么样的顺序联接表、何时应用搜索条件、是否使用某些列已有的索引等。优化条件的选择通常非常消耗 CPU；对于同一个查询，可能存在着多达几十种不同的处理方式，RDBMS 需要从中进行选择，特别当查询比较复杂，并且涉及多表的联接和条件时更是如此。不过，如果能找出正确的途径，优化将极大地改善查询性能。

4. 根据步骤 3 完成的优化处理，下一步即要为语句**生成一个执行计划**。执行计划只是优化后的动作序列二进制表示。

5. RDBMS 执行步骤 4 中创建的二进制指令集。大多数数据库供应商都提供了这种机制，此时第三组令牌中的变量可替换为实际值。

Note 注意：除了处理所有 SQL 语句的五个标准步骤外，处理 SELECT 语句时，还必须有一个额外的步骤，即以某种有组织的方式返回查询结果。

16.2 嵌入式 SQL 和动态 SQL

嵌入式 SQL 的思想是把 SQL 语句植入某种用传统过程化语言（比如 C、COBOL、FORTRAN 或 Pascal）编写的程序中。这种方法允许用 SQL 各种功能和过程化语言的所有优势（变量、参数、子程序、迭代、条件语句等）来处理关系数据库表。

动态 SQL 被视作一种嵌入式 SQL 扩展，可以把所有五个步骤延期至实际运行时执行，从而克服了自身的某些局限。静态 SQL 将不允许动态建立查询，因为在编译宿主语言程序时，就已经完成了前四个步骤（语句的分析、验证、优化以及生成二进制执行计划）。这意味着在程序中，SQL 语句将以硬编码的形式出现，值的替换只能发生于第三组令牌。使用动态 SQL 时，可以在程序执行期间，根据用户提供的参数（比如表名、列名、检索条件等）动态地创建 SQL 代码。

16.2.1 嵌入式 SQL

嵌入式 SQL 是 IBM 标准成为 SQL: 2003 标准的一个经典范例。正像前面所提及的，在 20 世纪 80 年代早期，IBM 提出并实现了嵌入式 SQL 原理。从那以后，大多数供应商只是遵循着这些标准，各自也仅仅产生了一些微小的变化。SQL-92 和 SQL: 99、以及最终的 SQL2003 规范先后认可了这一标准。SQL: 2003 为 Ada、C、COBOL、FORTRAN、MUMPS、Pascal 和 PL/I 提供了嵌入式 SQL 支持。

本书提及的所有“三大”供应商都在某种程度上支持该标准，但是最近几年，随着效率更高的新型 API（本章后面部分将进行讨论）不断推出，对嵌入式 SQL 的支持趋向于中断而不是扩展。Oracle 11g 为 C、C++和提供了预编译器；DB2 9 支持用于 C、COBOL、PL/I、FORTRAN 和 REXX（已经放弃了对后两者的支持）的嵌入式 SQL；MS SQL Server 只允许使用 C 语言，但 MS SQL Server 2008 已不再支持该功能，为了使用这个功能，发须安装前一版本的 DLL。

1. 嵌入式 SQL 基本元素

尽管不同的实现的嵌入式 SQL 语法略有不同（主要是由于专有 SQL 语法的不同），但从概念和原理是来说都是相同的。比如，必须声明宿主变量；建立数据库连接；执行一个或多个 SQL 语句并获得结果；按照需要执行某些错误处理。

Tip 提示：为了能够在宿主程序中使用嵌入式 SQL，必须为其提供特殊指令，用于如何发现宿主程序将使用的 RDBMS 的特殊预编译器文件。

www.TopSage.com

宿主变量

宿主变量是宿主程序和服务器之间通信的关键。用于把数据传递到RDBMS或者接受返回的信息。也就是说，连接SQL语句与宿主语言（C、COBOL等）语句都必须通过宿主变量。例如，在C语言程序中，可以利用先前声明的宿主变量，把来自命令行的参数传递到SQL语句。在DECLARE部分，可以使用下面的形式完成宿主变量的声明：

```
EXEC SQL BEGIN DECLARE SECTION <language-specific delimiter>
      <language-specific variable declaration>,...
EXEC SQL END DECLARE SECTION <language-specific delimiter>
```

下面的例子显示使用C语言语法声明一个字符串宿主变量（这个变量可以存储50个字符）和另一个整型变量：

```
EXEC SQL BEGIN DECLARE SECTION;
      char custname [51];
      int custid;
      ...
EXEC SQL END DECLARE SECTION;
```

EXEC SQL关键词用于在宿主语言程序中启动嵌入式SQL块；C语言中的定界符是分号（;），COBOL中则为关键字END-EXEC，而FORTRAN不需要定界符。宿主变量的数据类型通常按照宿主语言的变量声明规则来进行声明。

正如前面所述，使用宿主变量的目的在于为宿主程序和RDBMS之间提供通信。通常情况下，宿主变量分成两大类：输入宿主变量和输出宿主变量。输入宿主变量从宿主程序传送信息到RDBMS，而输出宿主变量则接收来自数据库的数据。宿主变量可应用于任何表达式；不过在SQL语句中，它们需要用冒号（:）作为前缀，以便与数据库模式名进行区分。

Note 注意：宿主变量和SQL数据类型之间的对应关系非常复杂并且很不直观。事实上，某些宿主语言的数据类型与互不兼容的SQL数据类型的名称相同，这导致更多的混淆。更多详情，请参考供应商特定文档。

数据库连接

程序开始之前，必须与目标数据库建立连接。嵌入式SQL可使用关键字CONNECT连接数据库；该语法也是供应商特定的，下面是一般形式的语法：

```
EXEC SQL
    CONNECT <connect_string>
<language-specific delimiter>
```

对于不同的RDBMS来说，<connect_string>是不同的，对于不同的宿主语言，定界符也不相同。例如，使用Pro*C（Oracle的C/C++预编译器）连接到Oracle，可以选择以下嵌入式SQL语句（假定已经正确声明了宿主变量相应数据类型的username和password，并为它们指定了某些有意义的值；并假定数据库使用了默认连接）：

```
EXEC SQL CONNECT :username IDENTIFIED BY :password;
```

本示例表明了这样一个典型的情形，输入宿主变量把信息（用户名和密码）从宿主程序传递到RDBMS。

数据检索和SELECT INTO语句

嵌入式SQL相对于非过程化SQL的主要优点之一是能够在数据库服务器中检索数据，把数据赋予宿主变量，然后在宿主程序中使用这些数据。SELECT INTO是SELECT语句的特殊的变种，用于完成上述任务。它的语法与常规SQL SELECT语句的语法非常相似。主要的不同之处在于，在SELECT INTO语句的INTO子句中，将指定宿主变量存储结果集，而不是把结果集发送到标准输出。

使用SELECT INTO存在着某些限制，不过可以实现常规SELECT的大多数功能，其中包括分组以及使用聚合函数等（SELECT语句在第8章讨论）。SELECT子句中列出的列数必须与INTO子句中的宿主变量数相匹配，且它们的数据类型必须兼容。查询同样必须返回一行且仅只能返回一行，否则将生成TOO MANY ROWS或NO DATA FOUND RDBMS错误。

采取本章前面的宿主变量声明，从ACME示例数据库CUSTOMER表中，下面的SELECT INTO语句为一个别名为MNGA71396的客户选择CUST_ID_N和CUST_NAME_S列（COBOL语法）：

```
EXEC SQL SELECT CUST_ID_N, CUST_NAME_S
INTO :CUSTID, :CUSTNAME
```


www.TopSage.com

```
FROM CUSTOMER
WHERE CUST_ALIAS_S = 'MNGA71396' END-EXEC.
```

这个 SQL 语句中使用的宿主变量是输出宿主变量；可以在一条 SELECT INTO 语句中同时使用输入和输出宿主变量。

Note 注意：大许多情况下，SELECT INTO 极为有用，但它也有一个非常严格的限制——每次只能返回一行。在第 14 章中引入了 CURSOR 的概念。请回忆一下在游标上可以进行 4 种基本操作：DECLEAR、OPEN、FETCH 和 CLOSE。在 DECLARE 和 FETCH 中，可以同时使用输入和输出宿主变量。

错误处理

在任何应用程序中，错误处理都是重要的组成部分。对于一个具有嵌入式 SQL 的程序来说，错误处理意味着能够检测错误并从 SQL 语句的执行错误（除非程序存在任何其他的错误）中恢复。应当在每一个 DML 语句之后检查错误条件，确保它处理了所有应该处理的数据，这一点非常关键；使用 SELECT INTO 语句时，通常需要检查一个单一行查询是否未返回数据（NO DATA 条件），或者返回了多于一行数据（TOO MANY ROWS 条件）。

SQL: 2003 指定了一个特殊的变量 SQLSTATE，专用于错误处理。它由错误类以及错误子类组成，几个实现在这方面完全一致。表 16-1 列出了几个 SQLSTATE 类及其子类，以及各自的含义。

通常情况下，可以使用程序语言指定的条件语句，在宿主程序中执行错误处理。例如（使用 C 语法）：

```
...
EXEC SQL <single_row_select>;
if (SQLSTATE != "02000")
{
  <do_something>,...
}
...
```

Note 注意：在程序中使用 SQLSTATE 之前必须预先对其进行声明；详细内容请参考特定于供应商文档。

表 16-1 选择的 SQLSTATE 代码

代码	代码条件	子代码	子代码条件
00	成功完成	000	成功完成
01	警告	000 001 007	警告 游标操作冲突 未授予特权
02	没有数据	000	没有数据
08	连接异常	000 002 006	连接异常 连接名在用 连接失败
0A	功能不被支持	000 001	功能不被支持 多服务器事务
23	违反完整性约束	000 001	违反完整性约束 违反限制
27	违反触发数据改变	000	违反触发数据改变

2. 普通嵌入式 SQL 语句

表 16-2 中列出了其他一些有用的嵌入式 SQL 语句。

表 16-2 有用的嵌入式 SQL 语句

一般语句的语法	解释
EXEC SQL COMMIT [WORK] [RELEASE] <language-specific delimiter>	终止当前事务，永久改变所有数据库。可选关键字 RELEASE 释放资源并断开数据库连接
EXEC SQL ROLLBACK [WORK] [RELEASE] <language-specific delimiter>	终止当前事务，放弃所有的数据库改变。可选关键字 RELEASE 释放资源并断开数据库连接
EXEC SQL CONNECT RESET <language-specific delimiter>	断开数据库连接

www.TopSage.com

大家网 TopSage.com

续表

一般语句的语法	解释
EXEC SQL INSERT INTO <table name> [<columns_clause>] VALUES <values_clause> <language-specific delimiter>	在表中添加行
EXEC SQL UPDATE <table_name> SET <set_clause> [< where_clause >] <language-specific delimiter>	改变表中现有的值
EXEC SQL DELETE FROM <table_name> [< where_clause >] <language-specific delimiter>	从表中移除行

3. 典型的嵌入式 SQL 程序流程

下面的例子显示了包含嵌入式 SQL 的程序的典型流程：

```
EXEC SQL BEGIN DECLARE
  <host_variable_declaration>,...
EXEC SQL END DECLARE SECTION <language-specific delimiter>

EXEC SQL INCLUDE SQLCA <language-specific delimiter>

EXEC SQL WHENEVER <condition> <action>
<language-specific delimiter>

EXEC SQL CONNECT <vendor-specific connect string>
<language-specific delimiter>

EXEC SQL SELECT INTO <single_select>
<vendor-specific delimiter>

EXEC SQL <embedded_dml_statement>
<language-specific delimiter>

EXEC SQL COMMIT <vendor-specific delimiter>

EXEC SQL CONNECT RESET <vendor-specific delimiter>

<error handling section referred from WHENEVER directive>
```

16.2.2　动态 SQL 技术

在许多情况下，前面部分讨论的嵌入式静态 SQL 技术很有用，但是，它们有时灵活性不够，不能满足所有的要求。对于这种情形，基于 GUI 的应用程序是一个经典的例子，其中允许用户通过下拉列表选择列（甚至选择表）名，以及选择 WHERE 子句的条件，从而建立自己的特定查询。应用程序会动态地建立相应的查询，再发送到 RDBMS，接收结果，并且以清晰、易读的形式显示出来。

1. 动态 SQL 的两个变种

通常情况下，动态 SQL 都可以分成两类。第一类包括所有动态创建的 DML 和 DDL 语句，另一类则处理动态查询。

两类之间的根本差异在于特定的 SQL 处理。正如本章开始部分提及的，SELECT 语句处理中存在一个额外步骤，即返回有序的查询结果。由于 DML（或 DDL）语句要么成功要么失败，因此通常情况下，需要从 RDBMS 返回的就是返回代码（SQLSTATE）；SQL 查询返回结果集，该结果集由从表中选择的列组成。在实际的程序执行前，精确的列数和列的数据类型可能是未知前。这就意味着在建立动态查询之后，并提交到 RDBMS 执行之前，宿主程序必须分配一个适当的数据结构，用来保存预期的结果集。

由于上述原因，动态 DML/DDL 的处理比动态查询处理更加简单。令人庆幸的是，大多数现代程序设计语言，其中包括 Visual Basic、Visual C++、PowerBuilder、Delphi、Java 以及许多其他的语言，对编程人员隐藏了复杂内容而将这些内容委托给程序设计语言的内部机制。编程人员只需动态地配置一个表示这个动态查询的字符串，并把它的值赋给相应的（经常在一个类中预定义的）变量。

手动创建和处理动态查询，因此已经基本废除，在讨论动态查询技术时，不会非常详细地深入研

www.TopSage.com

究。请参阅供应商特定文档了解详细内容。

2. 动态 SQL 和 SQL2003 标准

我们已经认识到 ANSI/ISO 标准通常与供应商实际的实现存在着巨大差别。动态 SQL 的情形也是如此，在 ANSI/SO 标准被接受之前，动态 SQL 就已经存在很久了。早在 20 世纪 80 年代初，IBM 就实现了自己版本的动态 SQL，在某种程度上，所有主要的 RDBMS 供应商都遵循这些标准，但各自都做了一些改变。在动态查询的处理上，这一点表现得尤其明显。处理动态查询时，每个 RDBMS 都用于处理查询返回的结果集的机制。例如，本书讨论的三个主要的数据库都使用专有的动态 SQL 数据结构，也被称作 SQL 数据区域（SQL Data Area，SQLDA），可以处理动态查询的输出，但都实现了这个结构的自有版本，且相互之间并不兼容。SQL: 2003 标准用所谓的动态 SQL 描述符的类似结构取代了 SQLDA，它所起的作用与 SQLDA 相同，但结构完全不同。

3. 动态 SQL 的基本原理

从原理上来说，动态 SQL 与静态嵌入式 SQL 基本相同。两者都包括 SQLCA、数据库连接和断开数据库连接、声明宿主变量、错误处理以及执行 COMMIT 和 ROLLBACK 等。嵌入语句从 EXCE SQL 关键字开始，并以特定语言的定界符结束。由于 SQL 语句处理必须在运行时进行，而不是在编译时进行，因此，动态 SQL 引入了一些额外的元素，方便用户动态地创建语句。正如前面所述，DML/DDL 语句比动态查询更容易处理，原因是动态查询需要其他准备。本章接下来的部分将简要介绍两者的操作技术。

动态 DML 和 DDL

动态 SQL 提供了两种方法，用于在宿主程序中动态执行几乎任何的 DML 或 DDL 语句。第一种方法称作 EXECUTE IMMEDIATE，允许把一个说明 DML/DDL 语句的过程化配置的串提交到 RDBMS。另一种方法，也称作两步动态执行，包括两个语句：PREPARE 和 EXECUTE。

一步执行是在宿主程序中执行动态 SQL 的最简单的方式。首先，程序根据用户的输入、命令行参数等建立 SQL 语句，并将其存储在一个这之前声明过的字符串变量中。然后，再利用 EXECUTE IMMEDIATE 把这个变量传递到 RDBMS；数据库引擎负责该语句的执行。（此刻将执行所有五个处理步骤。）RDBMS 使用 SQLSTATE 把完成状态返回到宿主程序。一般语法如下：

```
EXEC SQL EXECUTE IMMEDIATE <sql_string> <language-specific delimiter>
```

下面的示例表明了如何建立和动态执行 SQL 语句，在 PRODUCT 表中，该 SQL 语句使用 C 语法把符合用户条件的行的 PROD_PRICE_N 列值更新为 25.50：

```
...
EXEC SQL BEGIN DECLARE SECTION;
...
char buffer[101];
...
EXEC SQL END DECLARE SECTION;
...
char searchcond[51];
...
strcpy(buffer, "UPDATE PRODUCT SET PROD_PRICE_N = 25.50 WHERE");
printf("ENTER SEARCH CONDITION:");
gets(searchcond);
strcat(buffer, searchcond);
EXEC SQL EXECUTE IMMEDIATE :buffer;
...
```

查询的框架如下：

```
UPDATE PRODUCT
SET PROD_PRICE_N = 25.50
WHERE
```

因此用户能够提交任何有效的搜索条件，构成合法的查询；例如，PROD_ID_N = 990、PROD_BRAND_S = 'STEEL NAILS'、PROD_PRICE_N = 33.28 等。

Note 注意：本示例中的代码仅仅是为了说明 EXECUTE IMMEDIATE 语句的概念；实际的程序可能至少会为 PROD_PRICE_N 使用一个宿主变量，而不会使用硬编码的常量。

两步执行较为复杂。创建 SQL 语句的与 EXECUTE IMMEDIATE 相同，唯一的不同点是两步执行可以使用问号（？），称作**参数标记**或**占位符**，用它来替代稍后被实际值替换的第三组中的任何令牌（本章的前面讨论过）。然后，语句将作为 PREPARE 语句的一个参数被提交，前 4 个 SQL 语处理步骤（分析、验证、优化以及生成执行计划）由 PREPARE 语句执行。在最后的步骤中，EXECUTE 把参数标记

www.TopSage.com

替换为实际的值并执行该SQL语句。

Note 注意：Oracle的参数标记不用问号，而使用宿主变量标记法（DB2或MS SQL Server不允许）。

PREPARE和EXCECUTE命令的一般语法如下：

```
EXEC SQL
PREPARE <statement_name> FROM {:<host_string> | <string_literal>}
<language-specific delimiter>

EXEC SQL
EXECUTE <statement_name> [USING <host_variable_list>]
<language-specific delimiter>
```

在准备好的动态SQL语句中，每个参数标记（如果有）都必须对应于USING子句中一个不同的宿主变量。使用Oracle标记法时，占位符的名称无须匹配宿主变量的名称；不过，在准备好的动态SQL语句中，占位符的次序应当匹配USING子句中相应的宿主变量的次序。

与EXECUTE IMMEDIATELY相比，两步执行可以获得更好的性能，特别在使用不同的参数多次执行准备好的语句时更是如此。对于使用不同值替换占位符的大量EXECUTE语句，PREPARE语句仅执行一次，因此，RDBMS也没有必要从第1步到第4步一遍一遍地执行SQL语句的处理。在宿主程序中，只要放上一个简单的循环即可。如果在程序的执行过程中只需执行SQL语句一次，可以选择EXECUTE IMMEDIATELY。

动态查询

本章前面提到，动态查询远比动态DML和DDL复杂，而且供应商的实现间存在巨大的差异。本节将使用ANSI/ISO语法介绍动态查询。

Note 注意：为了提高示例的可用性，我们在创建示例时采用了Oracle的SQL: 2003兼容版本的语法。实际上这只是在Oracle中操作动态查询的诸多方法中的一种；例如，SQLDA可以用来替代动态SQL描述符。

开始操作动态查询之前，需要学习更多动态SQL语句。

ALLOCATE DESCRIPTOR命令分配描述符给“输入”和“输出”参数：

```
EXEC SQL ALLOCATE DESCRIPTOR <'descriptor_name'>
<language-specific delimiter>
```

DESCRIBE语句获取准备好的SQL语句的信息。DESCRIBE INPUT为准备好的动态语句描述输入宿主变量。DESCRIBE OUTPUT则给出了输出列的数目、类型和长度：

```
EXEC SQL DESCRIBE [INPUT | OUTPUT] <sql_statement>
USING [SQL] DESCRIPTOR <'descriptor_name'>
<language-specific delimiter>
```

SET DESCRIPTOR语句为SELECT语句中的WHERE子句指定输入值。必须为每一个宿主变量使用单独的SET DESCRIPTOR语句。可以指定类型、长度和数据值；同时，也必须指定VALUE，这是动态SQL语句中宿主变量的相对位置：

```
EXEC SQL SET DESCRIPTOR <'descriptor_name'>
[VALUE <item_sequence_number>,]
[TYPE = <:host_variable1>,]
[LENGTH = <:host_variable2>,]
DATA = <:host_variable3>
<language-specific delimiter>
```

例如，假使语句是SELECT: v1，:v2，:v3，...，那么V1的VALUE为1；:v2值为2，依此类推。TYPE是ANSI类型代码，请参见表16-3。

DEALLOCATE DESCRIPTOR释放不再需要的描述符所占内存：

```
EXEC SQL DEALLOCATE DESCRIPTOR <'descriptor_name'>
<language-specific delimiter>
```

表16-3 ANSI/ISO SQL数据类型代码

数据类型	类型代码	数据类型	类型代码
CHARACTER	1	FLOAT	6
CHARACTER VARYING	12	INTEGER	4
DATE	9	NUMERIC	2
DECIMAL	3	REAL	7
DOUBLE PRECISION	8	SMALLINT	5

www.TopSage.com

除了这些语句外，还应当了解如何使用动态游标。动态游标与静态游标差异不大，同样允许执行 4 种操作：DECLARE、OPEN、FETCH 和 CLOSE。主要的不同在于，声明动态游标时，不能使用硬编码的 SELECT 语句指定查询，而必须使用 PREPARE 语句准备好的语句名称间接引用。动态游标语句的语法如下所示：

```
EXEC SQL DECLARE <cursor_name> CURSOR FOR <statement_id>
<language-specific delimiter>

EXEC SQL OPEN <cursor_name> CURSOR
USING DESCRIPTOR <'descriptor_name'>
<language-specific delimiter>

EXEC SQL FETCH <cursor_name>
INTO DESCRIPTOR <'descriptor_name'>
<language-specific delimiter>

EXEC SQL CLOSE <cursor_name> <language-specific delimiter>
```

动态查询可通过下面的执行步骤来执行：

1. 声明变量，其中包括一个串缓冲区，用于保存将要执行的语句。
2. 为输入和输出变量分配描述符。
3. 用 PREPARE...USING DESCRIPTOR 语句准备语句。
4. 描述输入描述符。
5. 设置输入描述符。
6. 声明并打开动态游标。
7. 为每个输出宿主变量设置输出描述符。
8. 从循环中提取游标；使用 GET DESCRIPTOR 检索每行数据。
9. 把检索出的数据用在程序中。
10. 关闭动态游标。
11. 释放输入与输出描述符。

下面的例子使用了 C 语言语法，按上述步骤在宿主程序中使用动态查询：

```
...
EXEC SQL BEGIN DECLARE SECTION;
 ...
 char* buffer= "SELECT CUST_ID_N, CUST_NAME_S FROM CUSTOMER
                WHERE CUST_PAYTERMSID_N = :payterm_data";
 int payterm_type = 4, payterm_len = 2, payterm_data = 28;
 int custid_type = 4, custid_len = 4;
 int custid_data;
 int name_type = 12, name_len = 50;
 char name_data[51] ;
 ...
EXEC SQL END DECLARE SECTION;
...
long SQLCODE = 0 ;
...
main ()
{
 ...
 EXEC SQL ALLOCATE DESCRIPTOR 'in' ;

 EXEC SQL ALLOCATE DESCRIPTOR 'out' ;

 EXEC SQL PREPARE S FROM :buffer;

 EXEC SQL DESCRIBE INPUT S USING DESCRIPTOR 'in' ;

 EXEC SQL SET DESCRIPTOR 'in' VALUE 1 TYPE = :payterm_type,
             LENGTH = :payterm_len, DATA = :payterm_data ;
 EXEC SQL DECLARE cur CURSOR FOR S;

 EXEC SQL OPEN cur USING DESCRIPTOR 'in' ;

 EXEC SQL DESCRIBE OUTPUT S USING DESCRIPTOR 'out' ;

 EXEC SQL SET DESCRIPTOR 'out' VALUE 1 TYPE = :custid_type,
    LENGTH = :custid_len, DATA = :custid_data ;

 EXEC SQL SET DESCRIPTOR 'out' VALUE 2 TYPE = :name_type,
```


www.TopSage.com

```
    LENGTH = :name_len, DATA = :name_data ;

EXEC SQL WHENEVER NOT FOUND DO BREAK ;

while (SQLCODE == 0)
{
   EXEC SQL FETCH cur INTO DESCRIPTOR 'out' ;
   EXEC SQL GET DESCRIPTOR 'out' VALUE 1 :custid_data = DATA;
   EXEC SQL GET DESCRIPTOR 'out' VALUE 2 :name_data = DATA ;
   printf("\nCustomer ID = %s Customer Name = %s",
          custid_data, name_data) ;
}

EXEC SQL CLOSE cur;

EXEC SQL DEALLOCATE DESCRIPTOR 'in';

EXEC SQL DEALLOCATE DESCRIPTOR 'out' ;
...
}
```

嵌入式（静态）SQL 与动态 SQL 的优点和缺点比较

您也许会问：嵌入式 SQL 与动态 SQL 哪个更具效率？大多数消息来源（SQL 书籍、Web 网站等）都提倡嵌入式 SQL。理由非常简单——对于嵌入式 SQL，编译时就已经完成了 SQL 语句处理的前 4 个步骤，运行时无须再次重复。从理论上讲确实如此，但实际应用却没有这么简单。

理解该问题的关键是在步骤 3，这是 SQL 处理的优化步骤。大多数现代 RDBMS 都具有特殊的内置组件，称为优化器，优化器为每一条 SQL 语句创建执行计划，通常都是些基于动态的系统目录信息。这就意味着 SQL 语句的执行计划在这一天中可能是最优的，但在一月中未必最优，在一年中可能就更加称不上优化了。对于大型的 DML 密集型的数据库，更是如此，最优执行计划每天甚至每小时都会发生改变。问题的根源在于，计划通过硬编码放置到程序中去的，只有通过重新编译才能改变。因此，对于程序中 SQL 语句不佳的执行计划，系统付出的代价可能非常高。实际上，有可能比多次重复所有五个 SQL 处理步骤还要高很多。

两步动态 SQL 可以极大地改善性能——PREPARE SQL 语句只需要准备一次，然后在程序运行期间可多次执行。

前面提到的优势中都不适用于静态数据库——这样的数据库的优化执行计划短期内不会改变。

采用哪种 SQL 视情况而定；二者各有利弊，开发人员必须根据具体案例做出判断。

4. 嵌入式 SQL 和动态 SQL 的未来

嵌入式 SQL 首次使用了扩展 SQL 及其过程化语言功能。主要的功能是对 RDBMS 内部进行控制和管理。此外还提供编程功能，用来连接数据库，检索、处理、修改数据等。第一个任务现在主要由存储过程、函数以及触发器来完成，它们具有更完善的机制，可以妥善处理这些事情。对于第二个任务的大部分工作，目前依然是通过不同的程序设计语言来实现。但是，数据库的结构处在不断的发展过程中。大多数现代的编程语言都提供了访问 REBMS 的不同（有时效率更高和/或使用更容易）途径。其中包括 ODBC、JDBC、CLI 和 ADO.NET——仅仅列出了其中的少数几个（本章后面的部分将讨论访问 RDBMS 的其他方法）。

尽管如此，在现代程序设计世界中，嵌入式/动态 SQL 依然拥有自己生存的小环境——嵌入式数据库市场。嵌入式数据库在移动和无线设备、消费电子、医疗器械等领域获得广泛的应用。使用嵌入式/动态 SQL，仅需少许的内存和处理器能力，并能够快速可靠地访问相对静态的嵌入式数据库，嵌入式数据库正大量进入到各种新型的设备——手机、微波、个人数字助理（PDA）等，嵌入式 SQL 正在经历一个复兴过程。

16.3 SQL/CLI 标准

SQL 访问组（SQL Access Group，SAG）的小组委员会创建了调用级接口（CLI）标准。SAG 于 1989 年成立，宣称其将“定义和推动数据库的互操作性”。这个组包括 Oracle、Informix、Hewlett-Packard

www.TopSage.com

以及其他成员。列表中的成员正在不断地增加，现在几乎包括了涉足数据库领域的所有大公司。

同时，Microsoft、Lotus、Sybase 和 DEC 共同创建了 SQL 联接规范，随后该规范并入了基于 SAG 的规范。

1992 年发布了尚处在发展中的 CLI 标准，同年 Microsoft 推出了标准 CLI 规范的首个商业实现——Windows 的开放式数据库连接版本 1.0（Open Database Connectivity for Windows version 1.0）。这个版本扩展了原始的规范，并把它分成三个兼容层，其核心直接与 SAG CLI 标准相对应。

经过几次转变之后，原始的 SAG CLI 规范（经过修订和扩展）在 1994 年更名为 X/Open 预备规范（X/Open Preliminary Specification）。Microsoft 则在同年发布了 ODBC for Windows version 2.0。在排他性的源代码许可下，Visigenic Software 也于当年发布了用于非 Windows 平台的 ODBC 软件开发工具包（Software Development Kit，SDK）。

1993 年，ISO/ANSI SQL 委员会接受了作为 SQL/CLI 基本文档的 ODBC。1995 年，它成为 ISO/IEC 文档 9075-3: 1995 而后包括了 SQL99 标准扩展，最终成为了 SQL: L2003 标准的一部分[ISO/IEC 9075-3：2003 信息技术——数据库语言-SQL-第 3 部分：调用级接口（SQL/CLI）]。

对于 SQL/CLI 标准术语，我们谈论的是代表绑定机制的**绑定风格**——例如，嵌入式 SQL、模块语言和直接调用。

本章前面部分讨论过嵌入式 SQL，因 SQL 语句直接嵌入到宿主语言程序而得名。包含嵌入式 SQL 的应用程序进行编译和执行之前，首先必须使用供应商提供的预编译器进行预编译处理。

模块语言（亦称作过程化语言）借助 C、Cobol 或 Java 以及 Oracle 的 PL/SQL、DB2 SQL PL，或者 Microsoft/Sybase 的 Transact-SQL 等供应商专有语言，引用存储在服务器上的编译过的模块。

直接调用定义了一组 SQL 语句，使用特定供应商定义的某些机制，这些 SQL 语句可以直接在 RDBMS 中执行。

缺少通用标准阻碍了可移植应用程序的发展和代码的重用，为了适应各个不同的 RDBMS 访问并对返回数据进行处理，几乎要重写整个的应用程序。

SQL/CLI 标准把所有供应商特定的细节封装到若干统一的函数，因而可以相对独立于专有数据库接口。

在某种程度上，根据 SQL: 2003 标准，“三大”RDBMS 供应商都有自己风格的调用级接口。Oracle 的产品称为 OCI（Oracle 调用接口），IBM 开发了 DB2 CLI，Microsoft SQL Server 产品目前叫做 SQL 本机客户端，用于取代 ODBC 和 OLE DB。

Tip 提示：几乎所有的 SQL/CLI 函数都在 DB2 调用级接口上得到实现。这些命名转换同样遵循了 SQL: 2003 标准，除了前缀为“SQL”的 CLI 函数（例如，在 DB2 中，`AllocHandle` 函数称为 `SQLAllocHandle` 等）。

SQL/CLI 函数可以划分为 6 个主要分组，用于处理下列任务：

- 资源分配；
- 数据库服务器连接；
- 执行 SQL 语句；
- 获取数据库信息；
- 事务控制；
- 获取 CLI 实现信息。

表 16-4 提供了 SQL: 2003 标准定义的 SQL/CLI 函数列表。

表 16-4 SQL/CLI 函数

SQL: 2003 例程名称	SQL/CLI 例程分组	例程描述
`AllocConnect`	资源分配	配置 SQL 连接，返回句柄
`AllocEnv`	资源分配	配置 SQL 环境，返回句柄
`AllocHandle`	资源分配	配置环境、连接、语句或描述符句柄，并返回句柄
`AllocStmt`	资源分配	配置 SQL 语句，返回句柄

www.TopSage.com

大家网 TopSage.com

续表

SQL: 2003 例程名称	SQL/CLI 例程分组	例程描述
BindCol	执行 SQL 语句	为返回值分配特定数据类型的存储
BindParameter	执行 SQL 语句	在 SQL 语句中为参数分配存储
Cancel	执行 SQL 语句	尝试取消 SQL 语句的执行
CloseCursor	执行 SQL 语句	关闭已打开的游标（通过句柄）
ColAttribute	数据库信息	从结果集中获取列的属性
ColumnPrivileges	数据库信息	返回信息架构中存储的表列和相关特权的列表
Columns	数据库信息	返回信息模式中存储的表列的列表
Connect	数据库服务器	创建与指定驱动器的连接
CopyDesc	资源分配	复制一个 CLI 描述符
DataSources	数据库信息	返回所有可用的数据资源和描述信息（如果可用）的列表
DescribeCol	数据库信息	获取列的属性
Disconnect	数据库服务器连接	断开 SQLConnect 创建的数据库连接
EndTran	事务控制	终止（提交或回滚）SQL 事务
Error	数据库信息	返回 SQL 错误信息
ExecDirect	执行 SQL 语句	执行一条 SQL 语句（非经准备）
Execute	执行 SQL 语句	执行一条准备好的 SQL 语句
Fetch	执行 SQL 语句	获取下一个游标的行
FetchScroll	执行 SQL 语句	把游标移动到指定的位置，并获取当前行的值
ForeignKeys	数据库信息	返回表中包含外键的所有列（如果有）
FreeConnect	资源分配	释放已分配的连接资源
FreeEnv	资源分配	释放已分配的环境资源
FreeHandle	资源分配	通过句柄释放已分配的环境、连接等
FreeStmt	资源分配	停止语句的处理，并释放所有与该句柄关联的资源
GetConnectAttr	资源分配	提取连接属性的值
GetCursorName	数据库信息	提取游标名字
GetData	数据库信息	从结果集中返回一个单一的行的部分或全部的列值
GetDescField	数据库信息	从 CLI 描述符区域中返回一个值
GetDescRec	数据库信息	从 CLI 描述符区域中返回常用值
GetDiagField	数据库信息	从 CLI 描述符区域中返回诊断信息
GetDiagRec	数据库信息	从 CLI 描述符区域中返回常用的诊断信息
GetEnvAttr	数据库信息	提取环境属性的值
GetFeatureInfo*	CLI 实现信息	获取 CLI 实现支持的与功能有关的信息
GetFunctions	数据库信息	确定是否支持指定的功能
GetInfo	CLI 实现信息	返回有关实现的信息
GetLength	数据库信息	提取大对象的长度
GetParamData*	执行 SQL 语句	提取运态输出参数的值
GetPosition	执行 SQL 语句	提取大对象内字符串的位置
GetSessionInfo*	数据库信息	提取数据库会话信息
GetStmtAttr	执行 SQL 语句	提取 SQL 语句属性的值
GetSubString	执行 SQL 语句	提取大对象表示的字符串的一部分
GetTypeInfo	数据库信息	返回所支持的数据类型的信息
MoreResults	执行 SQL 语句	返回额外的结果集（如果有）
NextResult	执行 SQL 语句	返回下一个结果集（如果有）
NumResultCols	执行 SQL 语句	返回结果列的数量
ParamData	执行 SQL 语句	在运行时提供参数的值
Prepare	执行 SQL 语句	准备 SQL 语句，用于执行
PrimaryKeys	数据库信息	返回表中包含主键的所有列的列表（如果有）
PutData	数据库信息	为参数发送部分或全部数据
RowCount	数据库信息	返回 SQL 语句影响的行的数量

www.TopSage.com

续表

SQL: 2003 例程名称	SQL/CLI 例程分组	例程描述
SetConnectAttr	资源分配	设置连接属性
SetCursorName	执行 SQL 语句	定义游标名称
SetDescField	资源分配	在 CLI 描述符区域中设置值
SetDescRec	资源分配	在 CLI 描述符区域中设置常用值
SetEnvAttr	资源分配	设置环境属性
SetStmtAttr	执行 SQL 语句	设置语句的属性
SpecialColumns	数据库信息	返回表中唯一确定一行的列其最佳设定的信息
StartTran*	事务控制	显式地开始 SQL 事务
TablePrivileges	数据库信息	为指定表返回所有表和所有特权的列表
Tables	数据库信息	从数据源返回表名的列表

16.4 开放式数据库连接以及数据库对象链接和嵌入

ODBC 表示开放式数据库连接（Open Database Connectivity）。1992 年，它只是 Microsoft 自己的标准，到了 1995 年，被采纳为独立于供应商的 SQL/CLI 标准。ODBC 接口存在多个版本，最新版本为 3.5.x。ODBC 3.5.x 开始趋同于现行的 SQL 标准[ISO/IEC 9075-3:2003 信息技术-数据库语言-SQL-第 3 部分：调用级接口(SQL/CLI)]。

OLEDB 或 OLE DB(中间有一个空格)是数据库对象链接和嵌入的缩写。最初 Microsoft 用 OLEDB 来取代 ODBC，除了支持传统的关系数据库外，还可以支持对象数据库。OLEDB 的实现使用了 COM（组件对象模型），或更精确地说，使用了 ADO（ActiveX 数据对象）。

16.4.1 ODBC

多种 ODBC 驱动程序实现了 ODBC API 中的函数，应用程序通过 ODBC 驱动程序管理器调用驱动程序，函数和驱动程序之间的通信也由 ODBC 驱动程序管理器处理。

驱动程序和驱动程序管理器均可移植到非 Windows 的平台（例如，UNIX、Linux 和 Macintosh）上。驱动程序管理器收到应用程序的调用后，就装载相应的 ODBC 驱动程序，然后把这个调用转发到 RDBMS 接口。从数据库收到响应后，驱动程序把该响应转发到驱动程序管理器，驱动程序管理器再把调用的结果返回到应用程序。请求 ODBC 连接的应用程序负责装载和卸载驱动程序管理器，然后管理器自身完成其余的工作。驱动程序管理器也负责 ODBC 函数基本检查（正确的参数、数据类型等）。

图 16-1 显示了通过 ODBC 驱动程序访问 RDBMS 的一般方式。

驱动程序代表实现 ODBC 函数的库。每个 RDBMS 都需要自己的 ODBC 驱动程序。驱动程序可以视作是一种解释程序：理解应用程序的通用语言（ODBC API 调用），然后翻译成特定的 RDBMS 术语。驱动程序无须理解、执行、甚至翻译 SQL；RDBMS 负责完成这些任务。

ODBC 驱动程序执行以下任务：

- 建立和关闭到数据源的连接；
- 检查 ODBC 函数调用错误；
- 以对于应用程序透明的方式初始化事务；
- 把 SQL 传递给 DBMS 执行；
- 把数据发送到 DBMS 并从中检索数据；
- 把特定于 DBMS 供应商的错误映射到 ODBC SQLSTATES 预定义的错误集合。

虽然一些编程人员认为，开放式数据库连接已经过时，即将被 JDBC、ADO.NET 等更先进的技术取代，但是，作为一种简单、可靠和低廉的数据库连接解决方案，ODBC 几乎可以用于微软 Windows、UNIX、Linux 和 Macintosh 平台上的每一种客户端 RDBMS 产品，因而至今仍然非常流行。ODBC 应用方式亦是灵活多变，可以提取 RDBMS 数据，使用 Microsoft 查询把 RDBMS 数据传送到 Microsoft

www.TopSage.com

Office 文档，提供异类服务——例如，直接从 Oracle 访问 Microsoft SQL Server 或 DB2 的表。

16.4.2　OLEDB

OLEDB 并未按 Microsoft 的计划成为 ODBC 的继承者。实际上，它成为用于 COM 应用程序的 ODBC 的替代方案，如今已经过时，Microsoft 已不主张利用它进行新的程序开发。

OLEDB 驱动程序可用于多种数据库（关系的或非关系的数据库），提供给了多个供应商，包括 Microsoft 、IBM、Oracle、OpenLink、DataDirect 等；但是，与 ODBC 不同，OLEDB 专为 Windows（基于 COM）应用程序设计，因而即使从理论上来说，UNIX 平台上的 OLEDB 也是不能实现的。

图 16-2 显示了通过 OLEDB 访问 RDBMS 的一般方式。

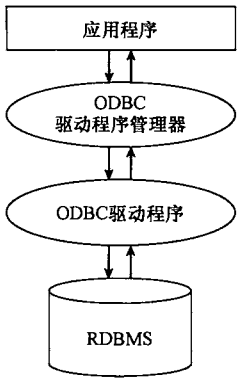


图 16-1　通过 ODBC 访问 RDBMS

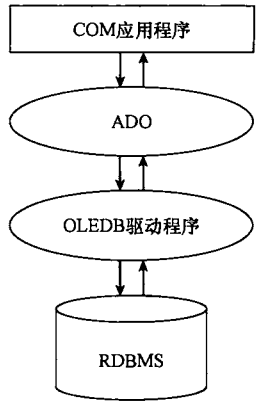


图 16-2　通过 OLEDB 访问 RDBMS

16.5　SQL 和 Java

Java 的日渐风行同样引起了 SQL 标准委员会的注意。SQL: 2003 包括两部分与 Java 相关的内容。ISO/IEC 9075-10：2003 – 第 10 部分：对象语言绑定（SQL/OLB）定义了 SQL 扩展，用于支持 Java 程序中嵌入 SQL 语句。ISO/IEC 9075-13：2003 – 第 13 部分：使 Java 程序设计语言（SQL/JRT）的 SQL 运行时和类型明确了如何以 SQL 调用常例的方式调用 Java 方法，以及如何以 SQL 将 Java 类作为用户自定义类型来使用。

16.5.1　Java 数据库连接（JDBC）

Sun 公司提议（并且实现）了 Java 数据库连接，从而为流行的 Java 编程语言（有人称之为“程序设计平台”，此处不多争论）提供一套统一的数据库访问方法。可以使用 JDBC 连接到数据源（不一定是 RDBMS 数据源），执行 SQL 语句（即 SQL 语句转发到 SQL 处理软件（如 RDBMS）去执行），并处理 SQL 语句的输出。最新的 JDBC 标准是 JDBC 4.0，包含于 Java SE 6 (以前称作 J2SE)和 Java EE 6（以前称作 J2EE）。

Tip　提示：J2SE（标准版）和 J2EE（企业版）之间的不同主要体面在范围和功能上。两者皆为使用 Java 编程语言开发应用程序的标准；Java EE 虽然基于 Java SE，但另外提供 Web 服务、组件模型、管理和通信 API，可用于创建企业级多层应用程序。

JDBC 拥有两套接口：一套用于数据库应用程序编程人员，一套用于数据库驱动程序编程人员。

Note　注意：许多业界重量级角色都在其产品中支持 JDBC 标准，其中包括 IBM、Oracle、Gupta Technologies、Borland 等。Microsoft 从 SQL Server 2000 开始，以提供 JDBC 的驱动程序的方式支持了 JDBC/ODBC。

图 16-3 显示了通过 JDBC 驱动程序访问 RDBMS 的一般方式。

java.sql 包（与 Java Core 一起发货）包含了 JDBC 的功能（基本的和扩展的）。

如今共有 4 种类型的 JDBC 驱动程序可供使用：JDBC-ODBC 驱动程序、本机 API 驱动程序、网络协议驱动程序和本机协议驱动程序。

www.TopSage.com

1. JDBC–ODBC 驱动程序（类型 1）

类型 1 驱动程序提供了在标准 ODBC 驱动程序上的 JDBC API 访问。从本质上增加了额外的通信层，从而减慢了整个处理过程。它们是依赖于平台的，因此除非没有其他选择，通常不建议使用。这种方法唯一的优点是，当前的架构可以用于基于 Java 的应用程序。

图 16-4 显示了通过 JDBC-ODBC 桥接的方式（类型 1）访问 RDBMS。

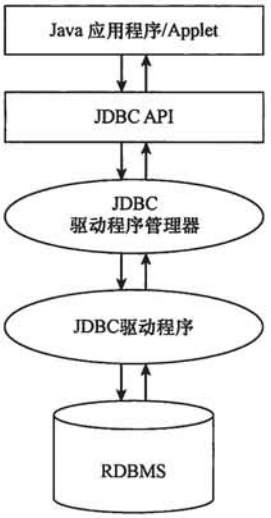


图 16-3　通过 JDBC 接口访问 RDBMS

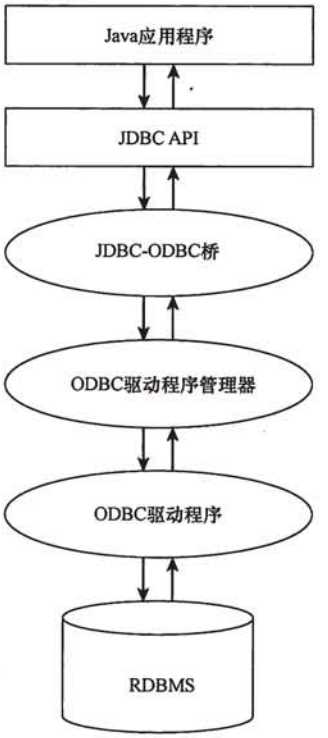


图 16-4　通过 ODBC-JDBC 桥接的方式（类型 1 驱动程序）访问 RDBMS

2. 本机 API 部分允许 Java 技术驱动程序（类型 2）

类型 2 驱动器把 JDBC 调用翻译成对 RDBMS 本地接口的直接调用（CLI）。它们比类型 1 要快很多，但依然存在许多缺陷。本机 API 驱动程序是操作系统依赖性的，要求在客户端安装供应商软件，并且不可用于 Internet 应用程序。

图 16-5 显示了通过类型 2 JDBC 驱动程序访问 RDBMS。

3. 网络协议完全 Java 技术允许的驱动程序（类型 3）

类型 3 驱动程序把 JDBC 调用翻译成独立于 DBMS 的网络协议，随后，由中间件服务器翻译成特定于 DBMS 的协议。这些驱动程序都是依赖于平台的——在中间件服务器上处理一切与平台相关的事务。网络协议驱动程序快速、灵活并且可能是四种驱动程序中效率最高的；唯一的缺点是增加了一个需要维护的层（通常是应用程序服务器）。

图 16-6 显示了通过网络协议（类型 3）驱动程序访问 RDBMS。

4. 本机协议驱动程序（类型 4）

这些驱动程序也被称作直接数据库纯 Java 驱动程序（Direct to Database Pure Java Driver），把 JDBC 调用直接翻译成特定数据库的网络协议。类型 4 完全是依赖于平台的，可靠且性能良好。唯一的不足之处是，必须在客户端为每一个 RDBMS（即 Oracle、DB2、Microsoft SQL Server 等）安装不同的驱动程序。

图 16-7 显示了通过本机协议（类型 4）驱动程序访问 RDBMS。

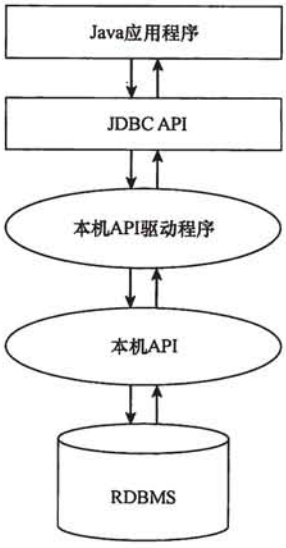


图 16-5　通过本机 API 驱动程序（类型 2）访问 RDBMS

www.TopSage.com

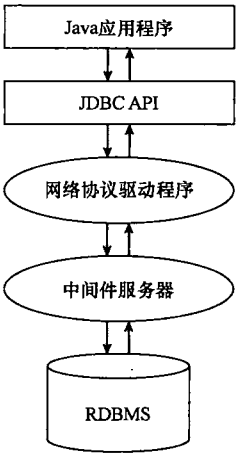


图16-6 通过网络协议驱动程序（类型3）访问RDBMS

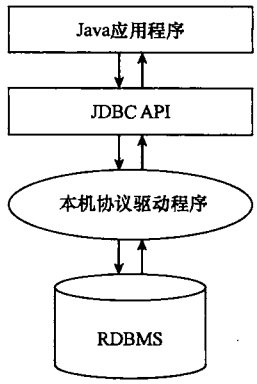


图16-7 通过本机协议驱动程序（类型4）访问RDBMS

提示：驱动程序通常实现RDBMS中可用的功能和函数子集；检查一致性级别时，并非所有功能都受支持。通常情况下，供应商的驱动程序才最为完整。

下面的Java代码示例显示了使用JDBC连接到Oracle，根据相应的标准提取记录，并把记录显示在屏幕上：

```
// JDBC sample
package jdbc_test;

// Import JDBC-related and other relevant classes
import java.sql.*;
import java.util.*;

public class JDBCTest {
    // Declare connection
    private Connection dbConn;
    // Main program starts here
    public static void main( String args[] ) {
        JDBCTest db_tst = new JDBCTest( "CUSTOMER"
                                       ,"CUST_NAME_S"
                                       ,"CUST_ID_N"
                                       ," CUST_NAME_S LIKE 'WI%'");
        System.out.println("done");
}

// constructor
public JDBCTest(String sTable,
                String sField1,
                String sField2,
                String sWhereClase){
try{
  // assemble connection string and establish db connection.
  Class.forName("oracle.jdbc.driver.OracleDriver").newInstance();
    String connString="jdbc:oracle:thin:@localhost:1521:acme";
    dbConn=DriverManager.getConnection(connString,"acme","acme");

    Statement stmt = dbConn.createStatement();
    // assemble and execute query; store results in result set rs
    ResultSet rs = stmt.executeQuery("SELECT " + sField2 + ","
                                                + sField1
                                               + " FROM "
                                               + sTable
                                               + " WHERE "
                                               + sWhereClase);

   // Print the retrieved records on the screen
while (rs.next()) {
   System.out.println(rs.getString(1) + "::" + s.getString(2));
}
        // do some cleanup
        rs.close();
        stmt.close();
        dbConn.close();
        }
```


www.TopSage.com

```
        catch ( SQLException exSQL ) {
              // database related exceptions
              exSQL.printStackTrace();
        }
        catch ( Exception ex ) {
            // generic exceptions
            ex.printStackTrace();
        }
    }
}
```

程序的输出如下所示：

```
152::WILE BESS COMPANY
55::WILE ELECTROMATIC INC.
63::WILE ELECTROMUSICAL INC.
7::WILE ELECTRONICS INC.
1::WILE SEAL CORP.
done
```

16.5.2 SQLJ

SQLJ 是 Java 程序中嵌入式 SQL 语句的标准。换句话说，SQLJ 是 Java 中的嵌入式 SQL，就像 Pro*C 是 C 语言的嵌入式 SQL。编译 SQLJ 程序之前，必须先对其进行预编译。

所有的 SQLJ 语句应从井号（#）开始，并以分号（;）结束：

```
# sql [ctx] {
             UPDATE PRODUCT
             SET PROD_PRICE_N = :v_price
             WHERE PROD_ID_N = :v_prod_id
            };
```

上述语句截自一个 Java 程序，该语句更新了 PRODUCT 表，把变量 v_prod_id 所确定的产品价格设定为变量 v_price 的值。当然，我们假定变量已经过声明，数据库连接已建立，所有必须的 Java 类都已经在程序中预先导入。

下面的 Java 示例代码显示了在 Oracle 中使用 SQLJ 从 CUSTOMER 表中检索单一的记录：

```
// SQLJ sample
package sqlj_test;
// Import JDBC-related, SQLJ-related, and other relevant classes
import java.sql.*;
import oracle.sqlj.runtime.Oracle;
import java.util.*;
public class SQLJTest {
// main program
public static void main( String args[] ) {
 try {
 //connect to the database
   Oracle.connect("jdbc:oracle:thin:@localhost:1521:acme","acme",
   acme" );

 //declare variables
 String sCustomerName = null;
 String sCustomerID = null;

 String sTableName = "CUSTOMER";
 String sField1 = "CUST_NAME_S";
 String sField2 = "CUST_ID_N";
 String sWhereClase = "CUST_NAME_S = 'WILE SEAL CORP.'";

 //sqlj section
  #sql {
    SELECT :{sField1}, :{sField2}
    INTO :sCustomerName, :sCustomerID
    FROM :{sTableName}
    WHERE :{sWhereClase}
 };

 //print the output
 System.out.println(sCustomerID + "::" + sCustomerName);
 System.out.println("done");

 Oracle.close();

 } catch ( SQLException exSQL ) {
    // database related exceptions
    exSQL.printStackTrace();
```


www.TopSage.com

大家网 TopSage.com

```
    } catch ( Exception ex ) {
      // generic exceptions
      ex.printStackTrace();
    }
  }
}
```

程序的输出如下：

```
1::WILE SEAL CORP.
done
```

上述代码是本章讨论过的动态 SQL 的一个典型示例；它具有不同的名称（SQLJ），用于 Java 程序而不是 C 或 COBOL 应用程序，但思想完全相同——动态地装配并执行 SQL 语句。

Tip　提示：程序必须经过预编译才会被 Java 编译器理解。要达到这个目的，可以使用 sqlj 命令、JDeveloper 或其他特定于供应商的工具。

请注意，本示例中使用了 SELECT INTO 语句，假定结束集中已存在单一的记录。如果希望查询返回多个行，则应当使用**迭代器**。请参见供应商特定文档了解更多信息。

16.6　Oracle API 选项

Oracle API 选项包括 Oracle 调用接口（Oracle Call Interface，OCI）、Oracle 的 OLE 对象（Oracle Objects for OLE，OO4O），以及 Oracle 的.NET 数据提供程序（Oracle Data Provider for .NET，ODP.NET）。

16.6.1　Oracle 调用接口

Oracle 调用接口（OCI）是一组 API（Application Programming Interface，应用程序编程接口），允许开发人员按计划访问 Oracle 的 RDBMS，并控制 SQL 语句执行的各个阶段。

OCI 提供了应用程序可以连接的标准动态运行时库 OCILIB。从本质上来说，这就意味着嵌入式 SQL 终于成为过去。OCI 接口可以由多种第三代程序设计语言使用，比如 C/C++、COBOL、Fortran、Java（使用 OCI JDBC 驱动程序）、.NET（使用 ODP.NET 驱动程序）等。可以越过典型的 OCI 开发过程（创建嵌入式 SQL 应用程序的阶段），直接连接到 OCI 库，与使用嵌入式 SQL 相比，这样做大大改善了其性能。

Tip　提示：如果需要在程序中同时使用 OCI 调用和嵌入式 SQL，请参考供应商文档。

可以根据功能对 Oracle OCI 函数进行分组。表 16-5 显示了函数的主要分组，以及每个分组的一些示例。

Note　注意：在 C/C++程序中使用 OCI 时，至少应包括一个 oci.h 头文件和一个 oci.lib 的链接。

最近几年，在使用 Oracle 进行应用程序开发的人员中间，OCI 编程接口风头正劲，其主要原因是编写的程序运行速度更快。

表 16-5　选定的 Oracle OCI 函数

函数分组	示例	用法
关系函数	OCIAttrGet(), OCIAttrSet(), OCIBreak(), OCILogoff(), OCILogon(), OCISessionBegin(), OCISessionEnd(), OCIStmtExecute(), OCIStmtFetch(), OCIStmtPrepare(), OCIStmtSetPieceInfo(), OCITransCommit(), OCITransPrepare(), OCITransRollback(), OCITransStart()	管理数据库访问并处理 SQL 语句

www.TopSage.com

续表

函数分组	示例	用法
浏览型函数	OCICacheFlush(), OCICacheFree(), OCICacheRefresh(), OCICacheUnmark(), OCICacheUnpin(), OCIObjectArrayPin(), OCIObjectCopy(), OCIObjectExists(), OCIObjectFlush(), OCIObjectFree(), OCIObjectGetAttr(), OCIObjectRefresh(), OCIObjectSetAttr(), OCIObjectUnmark(), OCIObjectUnmarkByRef(), OCIObjectUnpin(), OCITypeArrayByName(), OCITypeArrayByRef(), OCITypeByName(), OCITypeByRef()	操作从 Oracle RDBMS 检索出来的记录和对象
数据类型映射和操作函数	OCICollAppend(), OCICollAssign(), OCICollAssignElem(), OCIDateGetDate(), OCIDateGetTime(), OCIDateLastDay(), OCIDateNextDay(), OCIDateSetDate(), OCINumberIntPower(), OCINumberIsZero(), OCIStringSize(), OCITableDelete(), OCITableExists(), OCITableFirst(), OCITableLast(), OCITableNext(), OCITablePrev(), OCITableSize()	操作数据属性并确保正确的数据转换
外部过程函数	OCIExtProcAllocCallMemory(), OCIExtProcRaiseExcp(), OCIExtProcRaiseExcp- WithMsg(), OCIExtProcGetEnv()	实现从 PL/SQL 程序的 C 类型回调

16.6.2 OLE 的 Oracle 对象

20 世纪 90 年代后半期引入了 OLE 的 Oracle 对象（OO4O），用于利用取得巨大成功的 Microsoft COM（组件对象模型）标准的优势。OO4O 允许 COM 兼容的应用程序直接连接到 Oracle RDBMS，这样就可以绕过 ODBC，从而增加应用程序的效率和基本速度。

OO4O 包括一个进程内 OLE 自动服务器（DLL），它为兼容 COM 的应用程序提供了 OLE/ActiveX 接口，特别是 Visual Basic 和 Visual Basic for Application（VBA）应用程序；用于数据操作的 OCX 定制的数据控制；以及两个 C++ 类库，一个针对 Microsoft 基础类库（MFC）规范，另一个则针对 Borland（OWL）。

OO4O 与 Oracle 客户端一起安装。在机器上配置好 SQL*Net 后（请参阅 Oracle 文档了解详细内容），可以使用该接口连接到 Oracle RDBMS。

Oracle 网关

Oracle 开放网关（Oracle Open Gateway）接口用于从非 Oracle 数据库访问数据，其中包括 IBM

www.TopSage.com

DB2、Microsoft SQL Server、Sybase 等；另外，还提供有限的访问非关系型数据源功能。

下表列出了 4 种类型的 Oracle 开放网关。

Oracle	网关用法
透明网关	链接非 Oracle RDBMS 的最常用类型；处理复杂的内部通信和语法细节等。同时管理分布式查询和传输（即异类查询）
过程化网关	用于把外部数据库函数映射到定制的 Oracle PL/SQL 过程，从而实现与非关系型数据源之间的通信
访问管理器	专门用于 IBM AS/400 和 MVS 平台
Oracle 复制服务器	用于异类数据的复制

Oracle 开放网关可用于各种主要的数据库产品，其中包括 Microsoft SQL Server 和 IBM DB2。

16.6.3　.NET 的 Oracle 数据提供程序

随着.NET 框架的出现，Microsoft COM 技术逐渐过时。用户仍然可以使用 OLE DB.NET 从.NET 应用程序访问 Oracle，但通过.NET 操作 Oracle 数据库最有效的途径是使用.NET 的 Oracle 数据提供程序（ODP.NET），ODP.NET 由 Oracle 9iR2 最先引入，可向后兼容 Oracle 8.0 和 Oracle 8i。去除了应用程序和数据库之间额外的一层，从而改善了其性能，提供了先进的 Oracle 功能（XML、先进的数据类型、PL/SQL 等等），可用于事务控制和资源管理等。

16.7　IBM DB2 调用级接口

DB2 调用级接口（DB2 CLI）是 DB2 数据库服务器家族的 IBM 标准。为适应 C/C++ API 程序设计进行了调整，可用于建立连接、传递动态 SQL 语句，或者调用永久模块（如存储过程）。

DB2 CLI 基于本章前面讨论过的 SQL/CLI 的 SQL: 2003 标准。（或者说得更清楚点，SQL/CLI 标准正是从 DB2 CLI 发展而来。）图 16-8 显示了应用程序通过 CLI 驱动程序访问 DB2 RDBMS 服务器的方式。

与嵌入式 SQL 相比，DB2 CLI 的主要优势是灵活性和方便性。嵌入式 SQL 需要通过预编译器把 SQL 语句转换成可执行的代码，然后再在 DB2 数据库绑定这些可执行的代码。DB2 CLI 的实现并未绑定到具体的产品或者环境，因此代码具备可移植性；它遵循 SQL 标准，能够连接多个数据库（对于绑定概念来说真的很多！），能够使用动态参数，而且在通常情况下，更加适合 GUI（图形用户界面）应用程序。

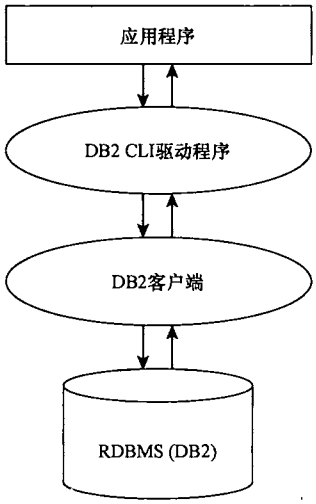


图 16-8　通过 CLI 访问 IBM DB2

16.8　Microsoft 数据访问接口

在 Microsoft 产品中，数据访问机制历经许多阶段。尽管调用级接口（Call Level Interface）速度快，使用却不容易，而且不易掌握。Microsoft 也认识到这一点，于是引入了封装了同样的功能的类，为编程人员提供了更为友好的接口。随着快速开发工具的出现（Visual Basic、Delphi、PowerBuilder 等等），编程人员友好接口允许用户把精力集中于商业逻辑的实现，而不会深陷入低层次的操作，这成了 Microsoft 的战略要点。数据访问对象（类）也随之迅速更替：数据访问对象（DAO）、远程数据对象（RDO）、活动数据对象（ADO）以及最新的 ADO.NET。尽管服务的目的相同，但接口却全然不同。例如，DAO 使用 JET 引擎（Microsoft Access 库）连接 SQL Server，而 RDO 却无需使用；它们的语法差异很大，而且不总是直观的。

www.TopSage.com

TopSage.com

16.8.1 ActiveX 数据对象

20 世纪 90 年代末，基于组件的体系结构成为了主流。尽管是谁（且何时）引入了该技术存在争议，但勿庸置疑，Microsoft 通过 OLE/ActiveX 技术（非 Microsoft 的阵营里其对手应是 CORBA）把它推介给范围最广的受众。ADO 是 ActiveX 对象的层次结构，汇集了许多 ADO 数据的库。它与 OLEDB 数据提供程序（假如有可用的 OLEDB）同时使用；对于标准 ODBC 连接，就必须使用 ODBC 的 OLEDB 提供程序。这就意味着数据访问抽象层越来越多，数据检索和操作速度减慢。更糟糕的是，ADO 最初限制在连接的环境中运作（即，必须维持数据源的永久连接）；后来才引入中断连接的数据访问模型，不过这个模型也有自身的问题。即使是现在，尽管.NET 计划已经牢牢地抓住了 Microsoft 开发人员群体，ADO 依然非常活跃，并且可能会持续几年（Microsoft 宣称至 2008 年将结束对“经典”Visual Basic 的支持，但这决不会意味着将转换所有的应用程序）。

16.8.2 ADO.NET

ADO.NET 是 Microsoft 数据访问机制的最新典型化身，专用于.NET 框架，而.NET 被认为将推动程序设计范例的变更。虽然 ADO.NET 名称的一部分仍然与过去相同，但却已经是一个完全不同的数据访问机制了。首先，它不是一个外部库，无须以某种方式链接到应用程序和再分配。ADO.NET 是.NET 框架的一个组成部分（最终将成为 Windows 操作系统的组成部分）。其次，它不再基于 OLE/ActiveX；即与 COM（Microsoft 组件对象模型）无关。

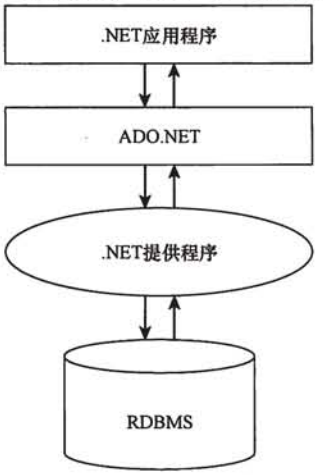


图 16-9 通过 ADO.NET 访问 RDBMS

图 16-9 显示了通过 ADO.NET 访问 RDBMS。

ADO.NET 设计用于支持连接和断开的环境，并支持 XML。

ADO.NET 的功能包括在下列命名空间中（命名空间是.NET 的一个概念，但是比较直观的说法——类似于库、头文件或 Java 包），表 16-6 给出了命名空间列表。此命名空间可用于 VB.NET 和 C#（读作 C sharp）。

表 16-6 ADO.NET 顶级命名空间

命名空间	描述
System.Data	ADO.NET 层次结构的顶级命名空间
System.Data.Common	.NET 提供程序可共享“基础”类
System.Data.OleDb	用于 OLEDB 类的.NET
System.Data.SqlClient	SQL Server 特定类：专为 SQL Server7.0 和以后的版本进行过优化
System.Data.SqlTypes	映射 RDBMS 数据类型，处理返回值
System.Data.Odbc	用于 ODBC 的.NET 框架数据提供程序
System.Data.Sql	包含支持 SQL Server 特定功能的类
Sysem.Data.OracleClient	用于 Oracle 的.NET 框架数据提供程序
System.Data.Design	包含创建定制类型化数据集的类
System.Data.SqlServerce	SQL Server 精简版本的管理数据提供程序

Microsoft DBLIB 遗产

DBLIB（DataBase Libray）是 Microsoft 专有的调用级接口，其中包括 C 函数和宏（类似于 Oracle OCI），可用于第三代语言（通常是 C）编写的应用程序访问 Microsoft SQL Server。这是一种原始的接口，随着 Microsoft SQL Server 7.0 版本的发布而被淘汰。

SQL Server 6.5 版本以后，DB-Library API 就未得到增强，这就意味着即使能够用于 Microsoft SQL Server 2008 的通信，其适用范围也局限在 Microsoft SQL Server 2008 的 6.5 版本的功能集（所有的一切都回到了 1997 年）。

DBLIB 的原理类似于 Oracle OCI 接口：包括头文件，从而把所有的定义引入 C 程序，再编译并

www.TopSage.com

进行链接。

Microsoft 文档中有与设计 DBLIB 有关的更多帮助，但该接口即将废弃（但如果服务器是版本 6.5 或 6，只能运行在 MS-DOS 和 OS/2 环境的时候，可能没有太多的选择）。

现在使用.NET 家族语言中的一种，即使用 C#来说明 ADO.NET 的使用；其他移植到.NET 的语言（VB.NET、COBOL、Eiffel、带有管理扩展的 C++、这里只列出了少数几个）的语法可能有所不同，但基础机制完全相同，原因是 ADO.NET 已经集成到了.NET 框架中。

下面的 C# 代码片断演示了使用 ADO.NET SQLData 连接到 Microsoft SQL Server 2008。要尝试本示例需启动 Visual Studio..NET IDE（Integrated Development Environment，集成开发环境），选择控制台应用程序，并命名为 SQL_Connect，然后输入下面显示的代码，最后编译运行。应记住，C#区分大小写，因此 SqlConnection 不等于 SQLConnection。命名空间可以使用也可以不使用，它将自动出现在代码窗格中。

```
using System;
using System.Data;
using System.Data.SqlClient;

class SQL_Connect
{
 static void Main(string[] args)
 {
 //
 //prepare connection string
 //use integrated security (Windows Authentication)
 //as opposed to UserID/Password pair
 //
    string s_conn = @"server=(local);" +
                    "Integrated Security=true;" +
                    "database=ACME";
 //
 //Create SqlConnection object instance
 //
    SqlConnection connSQL = new SqlConnection(s_conn);
 try
  {
    //
    //open connection to the local instance
    //of your SQL Server
    //
       connSQL.Open();
 string s_SQL = "SELECT * FROM customer " +
                "WHERE CUST_NAME_S LIKE 'MA%'";
    //
    //create command object and pass to it
    //connection object and SQL query string
    //
       SqlCommand commSQL = new SqlCommand(s_SQL,connSQL);
    //
    //retrieve all the records returned by the
    //query into SqlDatReader object
    //
       SqlDataReader sqlRead=commSQL.ExecuteReader();
       Console.WriteLine("Example demonstrating .NET Data Provider");
       Console.WriteLine("connection to Microsoft SQL Server 2008");
    //
    //scroll the result set and print the output
    //onto standard output console
    //
    while (sqlRead.Read())
    {
     Console.WriteLine("{0}|{1}",
         sqlRead["CUST_NAME_S"].ToString().PadRight(30),
         sqlRead["CUST_ID_N"].ToString());
     Console.WriteLine("--------------");
    }
   }
   catch(Exception e)
   {
    //
    // in the case an error occurred
    // display a message, including error source
    //
       Console.WriteLine("Error occurred:" + e.Message);
       Console.WriteLine("Error Source:" + e.Source);
   }
   finally
   {
```


www.TopSage.com

```
        //
        //Close connection to SQL Server
           //
           connSQL.Close();
           Console.WriteLine("Connection closed.");
        //
        //wait for user input
        //to keep MS-DOS Window
        //
           Console.ReadLine();
     }
   }
}
```

如果使用 VisualStudio.NET IDE，则可以从任务栏或按键盘上的 F5 来运行程序，也可以使用命令行进入编译目录。程序的输出如下：

```
Example demonstrating .NET Data Provider
connection to Microsoft SQL Server 2008
MAGNA SOUND DIVISION CORP      |11
--------------
MAGNETICS USA INC.             |2
--------------
MAGNETOMETRIC DEVICES INC.     |3
--------------
Connection closed.
```

为了保证与以前的接口兼容，可以使用为 ODBC 和 OLEDB 设计的.NET 数据提供程序。

下面的示例用 VB.NET 编写，使用 Microsoft SQL Server 的 OLEDB 提供程序和 ADO.NET 类连接到本地服务器，并检索信息：

```
Sub Main()
    Dim objDataReader As SqlClient.SqlDataReader
    Dim strConn As String
    Dim strSQL As String
    Dim lCount As Integer = 0
    '
    ' assemble connection string using SQLOLEDB provider
    '
    strConn = "Provider=SqlOleDb;Data Source=localhost;"
    strConn = "Initial Catalog=Acme;"
    strConn = "User ID=acme;Password=acme;"
    '
    ' assemble SQL query
    '
    strSQL = "SELECT * FROM customer"
    strSQL = strSQL & " WHERE CUST_NAME_S LIKE 'WI%'"
    '
    ' write comments onto console
    '
    Console.WriteLine("An example of using SQLOLEDB")
    Console.WriteLine("Provider with VB.NET")
    Console.WriteLine("--------------------")
    '
    ' connect to the SQL Server and fetch the data
    '
    Try
        '
        ' create new Connection object
        ' using the connection string
        Dim objConnection As New SqlClient.SqlConnection(strConn)
        '
        ' open connection to the SQL server
        '
        objConnection.Open()
        '
        ' create a command object, with the SQL statement
        ' and connection object as initialization params
        '
        Dim objCommand As New    SqlClient.SqlCommand_
                                 (strSQL, objConnection)
        '
        ' retrieve results into DataReader object
        '
        objDataReader = objCommand.ExecuteReader()
        '
        ' scroll the data reader, and reteieve
        ' value from the 4th field in the recordset
        ' CUST_NAME_S, and increment counter
        '
        Do While objDataReader.Read
            Console.WriteLine_
                (objDataReader.GetSqlString(4).ToString)
```


www.TopSage.com

大家网 TopSage.com

```
                lCount = lCount + 1
            Loop
            '
            ' output the final message
            '
            Console.WriteLine("------------------")
            Console.WriteLine("Total records: " & lCount)
            '
            ' wait for the input
            '
            Console.ReadLine()
            '
            ' close the connection within
            ' an appropriate scope
            '
            objConnection.Close()
        Catch e As SqlClient.SqlException
            Console.WriteLine(e.Message)
        Finally
            '
            ' close DataReader
            '
            objDataReader.Close()
        End Try
End Sub
```

上面代码执行的结果如下：

```
An example of using SQLOLEDB
Provider with VB.NET
------------------
WILE BESS COMPANY
WILE ELECTROMATIC INC.
WILE ELECTRONICS INC.
WILE SEAL CORP.
------------------
Total records: 5
```

16.9　小结

理解 SQL 语句处理的 5 个基本步骤对应用程序设计者来说非常重要，同时，也是在具体案例中正常运用 API 的关键。

嵌入式 SQL 是过程化语言扩展 SQL 的第一个商业解决方案。允许用宿主语言编写的程序连接到 RDBMS，并且可以执行几乎所有的 SQL 操作（检索和修改数据，创建、改变以及删除数据库对象，授予和撤销特权等等）。动态 SQL 是嵌入式（静态）SQL 的合理扩展，可以在运行时创建和执行 SQL 语句。

动态 DML 和 DDL 语句较容易实现，因为不需要把数据返回到宿主程序。动态查询标准相对复杂，实现困难。大多数现代程序设计语言都将其实现的复杂性委托给程序设计语言的内部机制，这样做可以节省编程人员的时间。

随着新数据库技术的出现，嵌入式/动态 SQL 逐渐废弃，但因为存在着大量的遗留程序设计代码，短期内还不会消失。嵌入式 SQL 也在现代技术中找到了一席之地，也是唯一的选择——嵌入式数据库。同时，嵌入式/动态 SQL 的主要原理演化成了新型 API 技术。

CLI/SQL 标准建立了一组函数，应用程序应用这些函数以统一的方式访问数据源，并执行与数据相关的任务。

数据库连接标准已经是既定事实，Microsoft ODBC 3.0 是取得巨大成功的商业实现。对于所有类型的数据源，对应着几十种 ODBC 驱动程序，这些数据源包括 RDBMS 平面文本文件，可用于几乎所有平台——Microsoft Windows、各种 UNIX 版本和 Mac OS，这里只列出了其中的少数几个。ODBC 驱动程序负责应用程序和数据源之间的所有通信，在驱动程序管理器的要求下装载和卸装 ODBC 驱动程序。ODBC（CLI/SQL）驱动程序的工作方式类似于生活中的翻译人员——对应用程序（通过驱动管理器）讲通用语言（ODBC 函数集），同时又能说 RDBMS 的方言。

用于定义应用程序-数据库交互的技术不断涌现。当前最流行的是 Java 数据库连接（JDBC）和 ADO.NET。

每个 RDBMS 供应商都拥有自己的调用级接口：Oracle 的 Oracle 调用接口（OCI）、IBM DB2 的调用级接口（CLI）以及 Microsoft 的 DBLIB。尽管它们在原始速度方面提供了最佳性能，但其中一些正逐渐被废弃。

www.TopSage.com

第 17 章
SQL 的未来

CHAPTER

本章提要：

- OLAP 和商业智能
- SQL 的 LINQ
- 关系数据库中的对象
- 关系范例之外

SQL 已经存在 20 多年了，并开始显露出老态——特别是面对其他语言更是如此。面向对象程序设计范例于 20 世纪 90 年代初引入，当时 SQL 已经成为主流。互联网对结构化数据的需求似乎永远无法得到满足，同时也为利用和创建信息开辟了新的途径。随着计算机的功能更为强大，新的应用已经出现——例如，实时分析海量数据，揭示其中的隐藏趋势，跨互联网的复制，以新的数据格式（视频、MP3 等）存储数据。

这些发展都迫使 SQL 进行转变和加强适应。新型的 SQL:2003 标准引入了新的数据类型，从而弥合了面向文本的 SQL 和面向对象的软件之间的鸿沟。在标准的关系框架内，RDBMS 供应商增加了在标准关系框架（例如，.NET 和 Java 存储过程、AetiveX (OLE)自动化、XML 等）内使用对象的功能。新型面向对象的数据系统（OODBS）和面向对象的关系数据库系统（OORDBMS）已经开始萌芽。

本章涉及一些与 SQL 有关的主要新技术。本章只简要介绍一些正在发展（并确立）的新技术，例如，EJB（Enterprise Java Beans）或 Web 服务（Internet 上的暴露功能）。读者如有兴趣把这些技术与 SQL 结合起来，可以阅读有关该主题的书籍。

Note 注意：有关 SQL 未来的更多话题，建议阅读 Ed Roman、Scott W. Ambler 和 Tyler Jewell 合著的《Mastering Enterprise JavaBeans》（第二版），Wiley 于 2001 年出版。

17.1 OLAP 和商业智能

每一个已部署的数据库系统皆可归入两大类：OLTP 数据库和 OLAP 数据库（广义角度上的术语），某些已部署数据库可能表现两者的混合体。

联机事务处理（Online Transaction Processing，OLTP）系统设计支持事务——例如，订购处理、存货跟踪、记录雇员数据等，且在粒度非常细的层次上进行。这样的系统设计可以尽快地处理大量并发事务。简言之，这样的系统其主要目的就是累积结构化信息。

联机分析处理（Online Analytical Processing，OLAP）系统与聚合一起使用，并分析累积的数据。这些系统侧重于发现趋势，分析关键性的因素，进行统计分析等等。OLAP 查询通常需

www.TopSage.com

要处理海量数据，尽管执行速度很重要，但并非这样的系统最主要的特性。

OLAP 的基础是数据仓库。与 OLTP 数据库不同，数据仓库通常未经标准化（至少未达到 OLTP 数据库的程度）。有两种方法可以处理数据仓库的结构：标准化和维度化。前者具有先锋性质，是 William Inmon（数据仓库之父）极力推荐的。后者的支持者为 Ralph Kimball，他是第一个明确表达此概念的人。标准化数据仓库结合了聚合化和标准化（通常在 3NF 中）的数据模型。因此，这个模型更依赖开发人员提供可视化信息和分析信息的机制。

数据仓库的维度化数据模型利用了完全不同的范例，致力于信息的商业表示。维度化数据仓库的基础是星型和雪花型架构。基础表称作**事实数据表**，其中包括某些商业实体的核心信息——比如说，客户。围绕着这张表的（以及通过主/外键关系与之相关的）就是空间表，其中包括选定的属性，选择这些属性应基于商业需求、概念和所提供的信息。星型架构仅限于此，而雪花架构允许更为复杂的设计，其中的空间表也可以成为事实数据表。事实证明，这样的设计更加“对商业用户友好”，并允许利用“商业现货供应（COTS）”工具更加直观地以面向商业的方式导航这个数据模型。

通常情况下，数据是从不同的数据源收集而来（其中包括非结构化数据源，比如，文本文件、电子数据表等）。数据仓库在装载之前，必须进行检证和清洗；这就引入了 ETL（提取-转换-装载）处理过程。数据载入数据仓库后，即可利用可视化工具（比如 Cognos、Brio、MS 分析服务等）转换成为多维的 CUBE，做这一步的工作也可利用 SQL 的 MDX 扩展——这是另一种 SQL 方言，专门用于解决 OLAP“切片和分块”的复杂问题。

OLAP 规则

E.F.Codd 博士在 1993 年引入了 OLAP 这个术语，约 20 年前，也是他第一个提出关系数据模型。OLAP 数据库的核心是 CUBE——信息的多维聚合。它的风格多变（取决于实际的数据存储类型）——ROLAP（关系 OLAP）、MOLAP（多维 OLAP）、HOLAP（混合 OLAP）——就是把数据分析从手动、繁琐的科学与艺术的结合转变为一门计算机辅助的、精确的科学。OLAP 程序仍然需要数据分析；但在分析海量数据方面比其他方法有了巨大改进。Codd 博士创建了 12 个 OLAP 规则，大多数 OLAP 产品都以某种方式遵照执行了这些规则。

- **多维概念视图。**OLAP 操作的数据立方体（CUBE）说明了数据的多维构造。尽管从名称上来说，它暗示着三维数据，但实际上并未限制可能的维数。
- **透明性。**OLAP 系统应当是支持异类数据源的开放式系统的组成部分。
- **可访问性。**OLAP 应当向用户表示数据的单一逻辑架构。
- **一致的报告性能。**性能不应随着模型中维数的增加而降低。
- **客户端/服务器体系结构。**该结构应该基于开放的模块化系统。
- **一般性维度。**不局限于 3D，也不倾向于任何具体的维度。适用于一个维度的函数也应该适用于另一个维度。
- **动态稀疏矩阵处理。**稀疏矩阵与关系数据库中 NULL 的概念相关，同时也与压缩大文件的概念相关，这种矩阵的每一个单元并不全部包含数据。OLAP 应该适应发生改变的存储以及数据处理选项。
- **多用户支持。**OLAP 系统应该能同时支持多个用户。
- **无限制的跨维操作。**与一般化维度的规则类似，应当平等创建所有的维度，而且跨数据维度的操作不能限制单元之间的关系。
- **直观的数据操作。**在理想的情况下，用户不必使用菜单或执行复杂的多步操作，而是通过直观的拖放动作来完成相关工作。
- **灵活的报告。**保存树。用户应该根据需要打印，对基础金融模型的任何更改都能自动地反映出来。
- **无限制的维度化和聚合级别。**OLAP 立方体能够以不限制维度的方式建立，并且所包含数据的聚合不存在具体的限制。

www.TopSage.com

大多数 OLAP 工具（不管是集成的或者是独立的）通常都符合上述规则。理论研究者另外定义了更多的规则，也有一些成为既定事实的规则，后者是由具有重大影响的数据库市场从业人员所确立。更多的信息，请参考特定于 OLAP 的专著和供应商文档。

OLAP 可用于决策支持、销售分析、营销、数据整合等。数据得到累积后，OLAP 便可以介入，沿着预先定义的维度分析（时间、区域、客户的年龄等），从中找出实际的含义。通过 CUBE 结构，OLAP 提供了 OLTP 数据仓库中的数据的多维表达，使用 CUBE 结构，可以按照不同的标准集创建数据视图，并使用复杂的分析函数对其进行操作。

ROLAP、MOLAP 和 HOLAP

这些缩写是 CUBE 数据（OLAP 查询的主要操作单元）的存储方式。在三个主要的 RDBMS 中，OLAP 的功能、方法以及原理完全相同。

- **多维 OLAP**（Multidimensional OLAP，MOLAP）指 CUBE 的关系数据与聚合数据一起存储在 CUBE 自身的情况。多维 OLAP 响应最快，适合频繁使用（比如，可用于不需要实时数据但需要立即响应的 OLAP）。
- **关系 OLAP**（Relational OLAP，ROLAP）指 CUBE 的关系数据与聚合数据一起存储在关系数据库的情况。它提供了实时查询，因为所有的数据都需要从暂存区进行装配，响应比 MOLAP 慢。
- **混合 OLAP**（Hybrid OLAP，HOLAP）指 CUBE 的关系数据存储到关系数据库中，而聚合数据存储到 CUBE 自身的情况。混合 OLAP 的设计兼具前两种方式的优点：比 ROLAP 快，CUBE 结构比 MOLAP 小很多。

BI 市场经过整合后，Orace（收购了 Hyperion）、IBM（收购了 Cognos）、Microsoft、SAP（收购了 BusinessObject）和 SAS 成了领袖级的 OLAP 供应商。尽管 RDBMS 是 OLAP 服务的基础，但通常情况下，这些工具并不与具体的数据库集成或绑定在一起。例如，Oracle 的 Data Mining 实用工具和 Oracle Express 可以使用“本机”Oracle 数据库或 Hyperion Essbase，或者 IBM DB2 9.5。

17.1.1 Oracle 11g

Oracle 的 RDBMS 企业版提供了嵌入式 OLAP 服务器，从而在 Oracle 11g 数据库中直接集成了商业智能功能。允许直接在 OLTP 数据库中编译 OLAP CUBE，而无需把 OLAP CUBE 转移到特定的 OLAP 数据库。这种方式优缺点兼备；优点是省去了耗时且昂贵的数据转移（和转换）过程，缺点是在产品数据库上运行 OLAP 查询是资源密集型处理，可能会使操作变慢。

Oracle 11g 数据库为 Oracle OLAP 打下了基础，为其提供了数据存储和管理能力、分析功能、安全等，而且 OLAP 服务本身又可以支持多维计算、预测功能、模型等。同时也提供了许多向导来指导用户进行选择。

可以使用 SQL 以附加的 OLAP 相关函数直接查询 Oracle CUBE；因为 CUBE 已经在许多层面上进行了聚合，因此访问这些已经高度优化的数据时，无需使用 `GROUP BY` 子句和任何连接。为了利用 CUBE 的维度特性，应用程序应使用 Java OLAP API（应用程序编程接口）；Oracle 的 Discoverer Plus OLAP 是利用 OLAP API 的注重维度的商业智能工具的例子。此外，Java OLAP API 允许用户设计丰富的功能，可以利用 Java 应用程序、applet、Java 服务器页面（Java Server Pages）等建立跨平台的解决方案。它可以单独安装，在中间级硬件上安装，或集成到 RDBMS。

Note 注意：Oracle OLAP CUBE 与 `GROUP BY CUBE` 子句无关，后者根据值的所有可能的组合对所选中的行进行分组，并且为每一个分组生成单一的聚合行（交叉造表）。

17.1.2 IBM DB2 9.5

IBM DB2 的早期版本通过 DB2 OLAP Server 和 OLAP Server Analyzer 提供了 OLAP 功能。两者都是与 Hyperion（及其 Essbase 产品）合作开发的插件，用于 Windows 和 UNIX 客户，并且与 SPSS 合

www.TopSage.com

作为iSeries客户开发了类似的产品。IBM从销售市场退出，不再支持DB2 OLAP服务器（Oracle收购了Hyperion后，Hyperion和SPSS产品的OEM版本），同时引入DB2数据仓库版本（DB2 DWE）。

使用早期DB2 OLAP服务器的用户可以迁移到Hyperion或SPSS。Hyperion产品基于Java并使用Essbase的JAPI。IBM仅支持ROLAP和MOLAP功能，IBM DB2还具有OLAP Miner功能——作为IBM OLAP Server的“机会发现”组件。它为OLAP CUBE提供了数据挖掘算法，用于精准地定位“特殊”区域，并且把它们提交给分析者，供其进行进一步的研究。

Note

注意：存储功能影响我们对数据库的认识。我们的思绪仍然停留在两维的层面上，未超越原始的纸张（比如羊皮纸），正是在那上面曾经画出第一张表格。因为每个有关计算机永久存储的观点最终都是归结到平面上的1和0串，我们的对数据的视觉印象也是如此。无论如何看待它，访问特定的数据片断都是通过坐标（x，y）的，所有这些都极大地限制了思维。一旦涉及多维存储（例如全息存储），数据表就变了样，比如变成了一个球体，此时，访问数据就得通过球形坐标以及位置矢量。因此，随着范例的改变，访问和操作这样的结构也许需要不同的语言。

17.1.3 Microsoft SQL Server 2008

Microsoft通过Microsoft分析服务（Microsoft Analysis Services，SSAS）提供了OLAP功能，SSAS与SQL Server 2008绑定在一起（在SQL Server 7.0中是OLAP服务组件）。

多维表达式（Multidimensional Expressions，MDX）语言是基础单元，用于操作任何OLAP分析——CUBE。在许多方面，这种语言与SQL类似，可以操作存储在OLAP CUBE中的数据。Microsoft也提供了外部访问接口，比如OLEDB、ADO/ADO.NET和AMO（Analysis Management Objects，分析管理对象），籍此可以使用SQL Server 2008中的OLAP功能。

Cross-Reference

交叉参考：有关编程接口方面的更多信息，请参阅第16章。

除了预定义的函数之外，MDX也允许创建用在OLAP CUBE中的定制函数。尽管语法与SQL类似，但MDX却并非SQL的扩展；它是专为OLAP设计的不同的语言。虽然不是开放式的标准（由Microsoft引入），却是一种十分流行的OLAP工具，IBM、SAP、SAS、Brio、Microstrategy、Business Objects等OLAP开发商都为它提供了全力支持。

行与列

行和列作为存储数据“自然的”的方式正遭到非议。最近一次冲击来自Google、Sybase以及一些不知名公司推出的面向列的数据库仓库。其中还有一些关系世界里的重量级人物，比如Michael Stonebaker，他参与创建了Ingres RDBMS（这个RDBMS已经找到加入PostgreSQL的途径，PostgreSQL是最先进的开放源代码数据库），通过他的Vertical Database宣传自己的观点。

面向列的数据库不再把数据放入行中，而是把相似的数据放入列，这样做可以减少访问时间，并加速典型的OLAP操作。只有存储在磁盘上的数据才具备这样的性能——数据以单一的簇的方式读取，而非只读取信息的几个数位。当然，这样会妨碍写入过程，而当前的OLTP RDBMS已经对写入过程进行过高度优化；因此，面向列的数据库主要用于数据仓库OLAP，而非OLTP类型的数据存储。

当前的实现包括Sybase IQ Database、Google的BigTable，以及Vertical Database，这是StoneBaker的新公司Vertical System Inc.的产品。

17.2 SQL的LINQ

创建像Transact-SQL、PL/SQL和SQL PL等过程化扩展的目的是为了修正基于集合的语言SQL本身固有的弱点，在其中添加了过程化的概念，比如循环、条件逻辑等等。Microsoft的LINQ项目把这个思路推进到下一个层次，把嵌入式程序设计范例与.NET家族的面向对象程序设计语言的程序设计

www.TopSage.com

范例结合在一起。LINQ 项目（Language Integrated Query，语言集成查询）提供了一个通用查询设备，可以与关系数据源进行通信，比如 XML 数据源（包括可以存储在 XML 数据类型列中的数据）。下面是使用 C#编写的 SQL 的 LINQ 查询：

```
using System;
using System.Linq;
using System.Collections.Generic;

class linqTest {
static void Main() {
DataContext db = new DataContext(CconnectionString);
// Get a typed table to run queries
Table<ORDER_HEADER> ord = db.GetTable<ORDER_HEADER>();
Table<CUSTOMER> cust = db.GetTable<CUSTOMER>();

  var query = from ord in db.ORDER_HEADER
          from cust in db.CUSTOMER
          where ord.ORDHDR_CREDITHOLD_S <>"N"
&& (ord.ORDHDR_CUSTID_FN == cust.CUST_ID_N)
          select new {cust.CUST_NAME_S};
// query becomes IEnumerable<T>
// (T being the anonymous type generated by the compiler)
foreach (var dType in query)
{
Console.WriteLine("Customer on Hold = {0}",
   dType.CUST_NAME_S.ToUpper));
}
 }
}
```

LINQ 可以应用于不同的数据源——包括关系数据库、XML、内存中的结构，这里只列出少数几个。通过更为传统的 ADO.NET 数据访问机制，LINQ 可以轻易地融合进.NET 代码（例如，最新的 Microsoft 技术 Silverlight 1.1 就支持 LINQ 与数据库的通信）。

17.3 对象

面向对象的方法（20 世纪 80 年代有趣的学术话题）在 20 世纪 90 年代初成为主流，并且从此成为每一个新发展的标准（本章后面部分将解释面向对象这一术语）。

几乎所有的现代程序设计语言都是（或者声称是）面向对象的；但是，尽管面向对象的方法在计算机编程领域（提高开发速度、增强健壮性等方面）取得了成功，但是直到现在，在数据库市场中仍不是非常流行的概念。尽管存在着纯面向对象的数据库（在本章的后面部分讨论），但它们占有的市场份额却微不足道。不同概念的实现（提供协同 RDBMS 工作的对象和工具）看起来更加现实。许多重要的 RDBMS 供应商（其中包括 Oracle 和 DB2 9.5）都提供了用于传统 RDBMS 的对象（Object Oriented RDBMS，面向对象的 RDBMS 方法——OORDBMS）。因此，尽管面向对象的概念并不与本书的内容直接相关，我们仍然会在本章的数据库上下文部分对其进行简短的介绍。

在 RDBMS 领域中，OO 指数据库存储和检索对象实例（以下段落中将进行解释）的能力——这方面类似于 XML 文档：解析成文本并根据请求进行重构，或者原样保存整个对象——可以是 Java、ActiveX 或.NET 对象。此外，某些数据库提出**对象数据类型**，从而把 OOP 原理引入过程化 SQL 程序设计（有关 SQL 过程化扩展的更多信息，请参阅第 14 章）。这些都可以看作是用户自定义的数据类型（有关数据类型的详情，请参阅第 3 章），其中同样包括一些方法或函数。

17.3.1 OOP 范例

面向对象的方法把所有内容视为对象——即包含数据和相关方法的程序设计结构。通常情况下，对象表示现实生活中的实体，它把实体的复杂性简化为几个明确定义的特性及能够执行的任务。一个人、一棵树、一辆汽车、一个机构——都可以表示为对象；依此类推，像“银行账户”或“数据访问对象”这样的实体也可以表示为抽象对象。现在考虑这样一个对象，它构造了银行账户模型：拥有“余额”和“账户 ID”属性，以及“提款”和“存款”方法——所有这些都表示该类型的对象应具备的功能。面向对象的要素有封装、继承、多态以及标识，接下来的部分将讨论这些概念。

www.TopSage.com

大家网

1. 对象和类

每一个**对象**都拥有自己的属性和方法。例如，对象汽车的属性可能有引擎容量、引擎类型、车轮尺寸、内部颜色、外部颜色、变速类型。方法则可以包括“驾驶”、“转弯（左/右）”等等。可以通过**类**这一程序设计概念定义对象。对象是类的实例。可以这样比喻，类是房屋的设计图纸，而对象则是基于该图纸实际建造的房屋。同一个类可以实例化许多对象，这很像根据相同的设计图纸建造出许多房屋。

2. 封装

封装的主要思想是隐藏实现的细节，只能通过明确定义的方法对其访问，这样就减少了对象内部变化造成的影响，同时也增强了安全性。在非编程人员领域比如，家中拥有录像机或微波炉这样的可编程电子设备：通常只能通过按钮来操作。

Note 注意：通常可以通过公有、私有以及受保护方法来保证安全。公有方法对所有用户开放；私有方法只能在对象内部访问，受保护的方法只允许继承父类的类实例化的对象访问。

3. 继承

继承是一种允许编程人员根据（或者说继承）旧（现有的）类创建新类的机制。这个新类（理所当然地）称为子类，它具备旧类的所有属性（特性）和方法，但可以忽略或修改其中某些属性或方法，此外，还可以添加新特征。例如，子类 Chevrolet、Ford、Honda、Toyota 以及 Nissan 可以继承 CAR 类。允许重用该类所含的代码，并使开发过程更加严谨。

4. 多态

多态是指可以对不同的对象应用同一个给定的方法。例如，如果所得到参数数目不同，或者参数的数据类型不同时，同一个方法执行的动作在逻辑上却可以完全一致。一个假定的 ADD 函数可用于数字相加、串联字符串以及根据给定天数增加日期。这就意味着，只要编程人员的参数调用方法正确，不必关心实现细节，就可以得到正确的结果。

5. 标识

因为两个对象可能继承自同一个类，它们的所有特征可能完全相同，为了区分，两者标识必然不相同。每个对象都有自己唯一的标识符，或称为**句柄**。通常情况下，系统把句柄隐式分配给对象。

Note 注意：在普通的关系数据库的术语中使用术语“对象”时，有可能导致某些混淆。最初在 RDBMS 的语言中，**数据库对象**意味着表、视图、索引等等。程序中的对象并不是数据库中的对象。

17.3.2 SQL: 2003 标准中的对象语言绑定（SQL/OLB）

ISO/IEC 9075 定义了 SQL 数据库语言的扩展，用于支持 SQL 语句嵌入到使用 Java 程序设计语言编写的程序中，也就是通常所说的 SQLJ。该标准规定了 SQLJ 的语法和语义，以及多种保证 SQLJ 包可移植性的机制。该标准也是对 SQL/JRT 标准的补充，后者规定了 Java 程序设计语言的 SQL 运行时和类型。

与更为传统的 JDBC 应用程序相比，使用 SQLJ 创建的对象兼备诸多优点和缺点。其中，主要的优点是 SQLJ 高度类型化——如果类通过了编译就不会产生不可预期的错误，因为所有的嵌入式 SQL 语句都已经根据数据库进行了检查。第二个优点是，SQLJ 程序比相应的 JDBC 程序更为简洁，原因是 SQLJ 在更为抽象的层次上进行操作。缺点包括编译中需要额外的步骤（所有的 SQLJ 程序都必须首先经过处理器的处理），并缺乏对许多流行框架的支持（例如 Hibernate）。

标准化的 SQLJ 通过 OO 前端（即对象—关系映射）来暴露数据库的关系本质。此外，许多数据库支持外部模型引用，将它作为 RDBMS 定义数据类型的一部分；SQLJ 完全适合完成这项任务。

17.3.3 Oracle 11g 的支持

Oracle 允许创建**对象类型**——含有对象属性和方法的特殊结构。其语法包括两个部分。首先，声

www.TopSage.com

明对象类型：

```
CREATE [OR REPLACE] TYPE <object_name> AS OBJECT
(
[EXTERNAL NAME <class_name>]
[LANGUAGE JAVA USING <data_provider>]
 <attribute_name> <attribute_datatype>,...
 [MEMBER FUNCTION <function_name>
  [<function_parameter> <parameter_datatype>,...
    RETURN <datatype>]...
[MEMBER PROCEDURE <procedure_name>
[<procedure_parameter> <parameter_datatype>,...]...);
```

该对象可以包含内部数据类型（内置的和 UDT），还可以包含 SQLJ 中实现的类型。

接着可以定义这个对象类型的主体，对象类型的主体包含所有功能的实现代码。除非在对象类型的声明中已声明了任何成员函数或过程，否则这个部分是可选的。其语法可能迅速变得异常复杂，而不是以下示例所示的如此简单（假定在 CREATE OBJECT TYPE 部分，已经声明了一个成员函数和一个成员过程）：

```
CREATE OR REPLACE OBJECT TYPE BODY <object_name>
AS
  MEMBER FUNCTION <function_name>
 [<function_parameter> <parameter_datatype>,...
  RETURN <datatype>]...
 IS
 [<declaration_section>]
 BEGIN
  <function_body>
 [EXCEPTION
  <exception_body>]
 END;
MEMBER PROCEDURE <procedure_name>
[<procedure_parameter> <parameter_datatype>,...]...
IS
 [<declaration_section>]
 BEGIN
  <procedure_body>
 [EXCEPTION
  <exception_body>]
 END;
END;
```

该对象的实例可以存储到 Oracle 的表，并在 Oracle 的 PL/SQL 存储过程中调用。

在 Oracle 11g 中使用.NET 的能力局限于使用这些语言创建的存储过程；不可能像使用内部 UDT 或 SQLJ 一样，使用的用户自定义类型（User Defined Type）。

17.3.4 IBM DB2 9.5 的支持

DB2 允许使用类似 Oracle 中的方式创建对象，但语法差别很大，且更加复杂。这里是来自 IBM DB2 9.5 的简化版本：

```
CREATE TYPE <type_name> [UNDER <supertype_name>]
AS ( <attribute_name> <attribute_datatype>,...)
INSTANTIABLE | NOT INSTANTIABLE
[WITHOUT COMPARISONS] [NOT FINAL]
MODE DB2SQL
METHOD <method_name>
(<attribute_name> <attribute_datatype>,...)
RETURNS <datatype>
<method_specification>
```

除了可选的用户定义类型，执行 CREATE TYPE 语句可以生成诸多方法，用于检索和更新属性值，构造结构化类型的实例，以及支持比较运算符（=、<>、<、<=、>和>=）。

对于每一个经 CREATE TYPE 语句声明的方法，可以使用 CREATE METHOD 语句把方法的规范与主体联系起来。下面就是语法：

```
CREATE METHOD <method_name>
RETURNS <datatype>
[LANGUAGE SQL]|[LANGUAGE C|JAVA|OLE]
FOR TYPE <type_name>
RETURN {<value> | <sql_statement>}
```

LANGUAGE 语句允许确定使用的外部库；该值可能是 C、JAVA 或 OLE，后者是指 Microsoft 对象连接和嵌入自动接口，只适用于 Windows 平台，不过目前已经被看作一种技术遗产。

www.TopSage.com

17.3.5 Microsoft SQL Server 2008

Microsoft SQL Server 2008 在自己的 RDBMS 产品内部并未提供面向对象的功能；最可能实现的功能是 TABLE 系统数据类型，该类型允许把 SQL 查询返回的数据调整为一组行。它不能用于列数据类型，因此无法用于永久存储。外部应用程序创建的对象实例可以在数据库中以二进制的格式存储，但这样做与 RDBMS 中任何其他二进制存储无异。

最初，Microsoft SQL Server 中的用户自定义数据类型不允许创建结构化数据类型（比如记录字段），也不允许包含方法。从 SQL Server 2005 开始，增加了使用 CLR 用户自定义类型的能力。这样的类型将映射为一个处在外部组件中的.NET 类，因而可以利用成熟的程序设计环境，并拥有方法、特性等等。创建 CLR UDT 的语法基本与 Transact-SQL UDT 的语法相同，但为了适应在数据库服务器空间之外执行代码的外部性质，必须增加某些额外的措施。创建 UDT 之前，必须在 SQL Server 2008 中注册.NET 组件。例如：

```
CREATE ASSEMBLY <assembly_name>
FROM '\\<computer_name>\<dir>\<assembly>.dll';
GO

CREATE TYPE <my_type>
EXTERNAL NAME <assembly_name>
GO
```

然后，像使用系统里其他任何的 UDT 一样使用这些类型，其中包括表的创建。

集成了 CLR 后，SQL Server 2008 除了具备通过系统存储过程在 Transact-SQL 代码中调用旧的 ActiveX/OLE 对象之外，还可以使用.NET 类增强数据库的功能。

SQL Server 同时提供元数据服务（Meta Data Server），允许存储和管理有关信息系统和应用程序的元数据。它可充当中央知识库，用于数据和组件定义、开发和部署模型、可重用软件组件以及数据仓库描述；它通过为开发者和分析师建立普通框架，促进了开发和建模过程，很大程度上借签了 UML（Universal Modeling Language，通用模型语言）。

集合和数组

集合和数组长久以来一直是所有程序设计语言的标准工具，如今已经开始入侵 RDBMS 领域。它们极大地促进了程序设计过程，消除了对额外表的需要。数组和集合都可以包含数据元素，这些数据类型的主要区别是，数组所包含的数据元素数据类型相同，而集合可包含不同数据类型的元素。

可以想象，虽然对良好的数据库设计进行了无数次论证（有关数据库设计的更多信息，请参见附录 C），仍然有可能需要在存储有其他雇员信息的表中存储一个雇员的教育证书。如果雇员们通常拥有三个文凭，那么就可能考虑声明 EDUCATION_CREDENTIALS 列，类型为 VARRAY（5）OF VARCHAR（25）——使用 Oracle 9i 的功能。现在，对于每个雇员来说，将存在 5 个可能的记录用来存储信息。

雇员 ID	教育机构
1	Stanford University
	Massachusetts Insitute of Technology
	Kalamazoo Community College

为了应付获取的教育证书多于 5 个的情况，可以使用嵌套表（即一个列就是一个表的类型，因此可包含无限量的数据——Oracler 的功能）。Oracle 和 IBM DB2 9.5 都支持数组和集合，而 Microsoft 并不支持。

17.3.6 抽象数据类型

SQL99 引入抽象数据类型（ADT），不过实际上很难把它与类的定义区分开来。ADT 规范包括两个组成部分：属性规范和方法规范。ADT 的行为完全封装在类型定义内。ADT 允许数据库编程人员使用操作这些数据类型的内置方法，在数据库中构造应用程序特有的数据类型。与往常一样，执行标准总是是滞后——这个功能没有被三个主要的开发商实现。

www.TopSage.com

大家网 TopSage.com

17.3.7 面向对象的数据库

很明显，有关面向对象的数据库的话题已经超出了本书的范围，这里只简述它的发展形势。

随着面向对象的程序设计的出现，**对象存储**的想法产生了。虽然关系数据库拥有全部的现代因素、全新的数据类型以及功能，但总体上来说，它们仍然都是基于文本——分析、存储和检索文本数据。尽管各种新型媒体格式（视频、声音、PowerPoine 演示文稿、Microsoft Word 结构化存储）因为包含进的新数据类型而取得成功，不过从本质上说，这些仍然是基于文本的方法，只是在某种程度上做了一些扩展。

随着 OO 语言（C++、Java、C#、Visual Basic.Net、Delpli、Smalltalk、Eiffel 等）越来越流行，合理地在对象中存储信息成为可能，因为在这些语言中，对象是由实现的类定义的。想象如果存在一个 Java 的类 EMPLOYEE，它有自己的特性（属性）和已定义的方法；当应用程序创建这个类的一个实例，用于添加新雇员时，它将填充其特性，然后在数据库中保存整个的对象。对于 RDBMS 来说，这就意味使用文本信息填充大量的表；对于面向对象的数据库来说，这就意味着原封不动地保存对象——其版本可以是二进制、字节代码或者是文本说明，同时，仍然能够跟踪这个对象，比如说，通过雇员 ID 来跟踪。这就是 OODBMS 和 OORDBMS、要完成的所有任务。

尽管纯粹的面向对象数据库仍然力求实现自身存储信息的“纯 OO”方法，ORDBM 系统却依赖于已得到证明的关系技术，建立对象的层次结构。新的标准正在出现（参见表 17-1），新的想法正在成形。

表 17-1 RDBMS/OODBMS 标准对比

	SQL92	SQL1999	SQL2003	JDBC 和 SQLJ	JDO	ODMG 3.0
模型	关系	关系/对象	关系/对象	关系/对象	增强型 Java 对象模型	Java、C++；Smalltalk 等对象模型；OMG 普通对象模型
DDL	SQL	SQL PL/SQL，Transact-SQL	SQL PL/SQL，Transact-SQL	SQL	Java 和 XML	对象定义语言（ODL），部分基于 SQL92
DML	嵌入式/动态 SQL、CLI	用于 SQL、Java、PL/SQLTransact-SQL 的 CLI	用于 SQL、Java、PL/SQL.NET 的 CLI	嵌入式 SQL、CLI	Java	Java、C++、C#、Smalltalk 等
DQL	嵌入式/动态 SQL、CLI	CLI、OCI、ODBC	CLI、OCI、ODBC、.Net 提供程序	CLI、嵌入式 SQL、Java	JDO 查询语言（JQL）	对象查询语言（OQL），部分基于 SQL92

由于 OO 数据库与程序设计语言的紧密结合，为某个数据库编写的应用程序可能无法与其他某些产品一起使用。除了许多自有的方式外，为 OODBMS 建立标准的数个计划正在进行之中，如今，ODMG 3.0 标准已经产生。

尽管仍然属于新生事物，但 OODBMS 正在成为学术和公司领域中的主流，采用面向对象的技术是其中主要的推动因素，这些技术包括 Java（以及 EJB——Enterprise Java Beans）和 Smalltalk。

在表 17-2 中列出了当今市场上能够提供的一些面向对象数据库。

表 17-2 支持纯 OO 技术的数据库产品

公司	产品	版本
db4o	db4o - database for objects	6.3
eXcelon Corporation,Object Design Division	ObjectStore Enterprise Edition	7.0
Fresher Information Corporation	MATISSE	8.0
JYD Software Engineering Pty Ltd.	JYD Object Database	2.2
Micro Data Base Systems, Inc.	TITANIUM	8.1
Objectivity, Inc.	Objectivity/DB	9.0
Orient Technologies	Orient Enterprise Edition	2.3
	Orient Just Edition	2.3
The Ozone Database Project/Open Source	ozone	1.2
Versant, Inc.	FastVersant Object Databases	7.0.1
	FastObjects .NET	7.010
InterSystems Corp	Cache	2007.1
SYSRA/Open Source	EyeDB	2.7.10

www.TopSage.com

与ORDBMS相比，OODBMS的主要优势体现在消除了从应用程序（客户端）对象到RDBMS结构的映射。在当今的环境中，数据依然采用文本或数字的形式，RDBMS比同等的OODBMS/OORDBMS快得多；同时在某些情形下，对象方法可能优于“旧式”关系模型。尽管看起来ORDBMS数据库近年来取得了很大的发展，目前还未有最终的定论。

17.4 小结

新的技术不断改变收集、存储以及消费信息的方式。信息的基本特性永远在不断地变化之中，而且经常涉及新的媒体和新的格式。其中像行和列等最基本的格式有可能像渡渡鸟一样逐渐消亡，新的范例将最终取代它们。

OLAP已经成为BI（Business Intelligence，商业智能）的标准。自从文明开始之后，累积起来的数据（结构化或者其他形式）极其庞大，人们对数据的理解一定会进入下一个层次，即发现统计学上的趋势，实现这一点仅仅是时间问题。虽然BI并非RDBMS技术的组成部分，但如果不存在特定类型的数据库，BI的存在就没有意义——在我们的案例中，即是关系数据库。信息的主要处理单元是多维立方体（CUBE），操作这个多维立方体时，可以利用某些通用语言（比如Java）或某些专有语言（比如MDX）。一些供应商把商业智能工具与它们的RDBMS绑定；有些BI工具则是第三方的公司开发的独立工具。

面向对象的方式被奉为事实上的应用程序设计标准，对于面向对象的数据库来说，它同样是一个引人注目的原由。使用对象对周围的世界进行建模，需要储存这些对象的空间。RDBMS把对象映射为单词；而OODBMS则原封不动地接受它们。可以把对象比作一本书，大脑使用纯粹的文字创建出书中的图像；电影则存储并直接把可视对象传达到感官，从而绕过了语言表达的步骤。

OODBMS一定会成为未来的潮流。面向对象的数据库在许多公司中得到了实现，可用于存储和检索某些特定语言[比如Java、C++、Smalltalk以及C#（.NET）]创建的对象。从本质上来说，新的标准即将出现，性能上的差距（如果存在）将被消除，RDBMS将成为历史。就目前来说，RDBMS仍然是工商业界的支柱，虽然口头上为对象提供了支持，并把对象作为数据类型集成进来，但在使用这些对象时也应注意其低效性。

www.TopSage.com

计算机精品学习资料大放送

软考官方指定教材及同步辅导书下载 | 软考历年真是解析与答案

软考视频 | 考试机构 | 考试时间安排

Java 一览无余：Java 视频教程 | Java SE | Java EE

.Net 技术精品资料下载汇总：ASP.NET 篇

.Net 技术精品资料下载汇总：C#语言篇

.Net 技术精品资料下载汇总：VB.NET 篇

撼世出击：C/C++编程语言学习资料尽收眼底 电子书+视频教程

Visual C++(VC/MFC)学习电子书及开发工具下载

Perl/CGI 脚本语言编程学习资源下载地址大全

Python 语言编程学习资料(电子书+视频教程)下载汇总

最新最全 Ruby、Ruby on Rails 精品电子书等学习资料下载

数据库管理系统(DBMS)精品学习资源汇总：MySQL 篇 | SQL Server 篇 | Oracle 篇

平面设计优秀资源学习下载 | Flash 优秀资源学习下载 | 3D 动画优秀资源学习下载

最强 HTML/xHTML、CSS 精品学习资料下载汇总

最新 JavaScript、Ajax 典藏级学习资料下载分类汇总

网络最强 PHP 开发工具+电子书+视频教程等资料下载汇总

UML 学习电子资下载汇总 软件设计与开发人员必备

经典 LinuxCBT 视频教程系列 Linux 快速学习视频教程一帖通

天罗地网：精品 Linux 学习资料大收集(电子书+视频教程) Linux 参考资源大系

Linux 系统管理员必备参考资料下载汇总

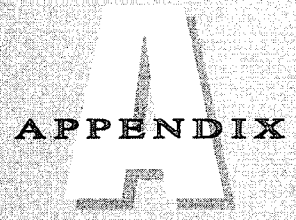
Linux shell、内核及系统编程精品资料下载汇总

UNIX 操作系统精品学习资料<电子书+视频>分类总汇

FreeBSD/OpenBSD/NetBSD 精品学习资源索引 含书籍+视频

Solaris/OpenSolaris 电子书、视频等精华资料下载索引

附录 A
配套网站



本附录提要：

- 描述网站

您可以查看本书的网站：www.wiley.com/go/SQLBible2。

这个网站包含用于创建和填充本书样例数据库 ACME 的全部 SQL 脚本（IIP 格式）。这些脚本都是 RDBMS 特定的，附录 F 中详细说明了运行这些脚本的顺序。使用书中和网站上提供的链接可以从 Internet 上下载各个 RDBMS 安装包。

我们也会提供一些我们认为对读者有用的相关网站链接，包括供应商文档、第三方工具，以及随时关注 SQL 和关系数据库技术的论坛。

注意：各个 RDBMS 软件的下载链接将定期更新，因为它们可能发生变化。如果安装过程改变（因为在编写本书时有些 RDBMS 还没有发布），更新后的过程也将在网站上发布。

此外，配套网站还提供了反馈途径和一个勘误页。欢迎所有建设性的评论和建议。读者提出的所有错误和疏漏将被评估，并与我们的说明和答谢一起发布到勘误页上。

www.TopSage.com

大家网
TopSage.com

APPENDIX B

附录 B
ACME 示例数据库

本附录提要：

- 一般信息和业务规则
- 命名约定
- 表间关系
- 列约束和默认值
- 索引
- 视图

ACME 是销售建筑材料的同名公司的数据库。它包含 15 个表，本章详细说明这些表。

B.1 一般信息和业务规则

ACME 拥有购买不同产品的客户。每一个客户都有一个或多个地址（如账单和发货地址），以及一个或多个电话或传真号码。客户也有默认的付款条件，付款条件指定客户必须在多少天之内付款才能获得特定折扣，例如，在 31 天内付款的客户可以得到 2%的折扣。没有按时付款的客户信用可能被中断，这意味着在他们支付账单前不能进行任何订购。另外，超过两年没有从 ACME 买过东西的客户会标记为不活跃。通过赋予客户别名来引用客户（仅用于公司内部）。

每一位客户都分配了默认的销售人员，这个销售人员负责客户和公司之间的联络。每个销售人员都有地址、电话和传真号码、用于引用他的代码以及其他一些属性。如果一个销售人员的所有客户都变为不活跃状态，则这个销售人员在系统中就是不活跃的。

ACME 实现了一种多级销售。它把产品卖给分销商，分销商又把产品卖给更低一级的分销商或直接卖给客户。这意味着任何一位 ACME 客户都可以是分销商（购买并再转售产品）也可以是供应商（从它这里购买产品），或两者兼具。

ACME 销售多种产品。每一个产品都有价格、说明、品牌、重量以及其他一些特征。

客户可以订购产品。当一个客户一次订购多件产品时，所有的订购项逻辑上分组到同一个订单。订单存储一般信息，如订购号、默认的销售人员和客户信息、订购数量、日期、发票号等。实际的发货数量可以不同于订购数量（例如，由于没有足够的存货）。

在选定一个客户时某些字段被默认填充（付款条件、销售人员），但可以进行修改。其他一些字段必须手动填充。订单和发票号是自动填充的并且不能修改。根据合计的订单总价格确定订单是否符合总额折扣。例如，所有总额在 $15 000 和 $20 000 之间的订单符合 3%的折扣条件。

新订单在创建时，被赋予 COMPLETE 状态。然后就可以发货给客户了，发货之后，其状态就改为 SHIPPED。销售部门的职员为这个订单创建发票，并将订单状态改为 INVOICED。客户可以取消订单；则状态改为 CANCELED。

www.TopSage.com

数据库也包含特定的发货信息——例如，提货单号码、发货总箱数、尾部号码、货运条款、日期等。通常，一批货包含一个订单，但特殊情况时一个订单也可以分为几批，当然，一批货包含多个订单也可以处理。

B.2 命名约定

ACME 数据库在其命名约定中使用一种简化的匈牙利表示法概念。其思想是把有关对象的信息体现在它们的名称中。许多数据库开发人员发现这个概念非常有用，尤其当处理由数百个表和数千个列组成的的大型数据库时。

Note 注意：匈牙利表示法是一种命名约定，它允许开发人员确定标识符的类型和用途。它是由 Charles Simonyi 博士发明的，Charles Simonyi 博士从 1981 年起就一直为 Microsoft 工作。这个命名法在 C++和 Java 开发人员中非常流行。Simonyi 博士称它为“匈牙利”只是一个玩笑。这个表示法是为了让代码更易读，但事实上，它看起来并不易读，倒不如用匈牙利语来写。

ACME 主要命名规则如下：

- 每个表有一个缩写，作为表中每个列、索引或约束名的一部分。例如 ADDRESS 表缩写为 ADDR。索引名以 IDX_为前缀，检查约束以 CHK_开始等等：

```
ADDR_ID_N (column name)
CHK_ADDR_TYPE (check constraint name)
IDX_ADDR_CUST (index name)
FK_ADDR_CUST (referential integrity constraint)
```

最后一行的外键名也表明引用完整性的方向（从 ADDRESS 到 CUSTOMER）。

- 所有列都有一个后缀，用符号表示出了列的一般数据类型（N 表示数值数据类型，S 表示字符串，D 表示日期，X 表示 XML 字段等等）。例如：

```
ADDR_ID_N
```

- 主键列包括一个后缀 ID：

```
ADDR_ID_N
```

- 外键列包括父表的主键名和后缀 F。例如：ADDR_CUSTID_FN 是 CUSTOMER 表中 CUSTOMER_ID_N 的外键。

Note 注意：为了示范而简化了 ACME 的命名约定。例如，S 表示任何字符串数据类型，并不区分 CHARACTER 和 VARCHAR；N 表示任何数值数据类型（INTEGER、DECIMAL、FLOAT）等。

B.3 表间关系

ACME 中的表通过主/外键关系连接（就像任何关系数据库中那样）。数据库在一定程度上被标准化了（参见附录 C）。图 B-1 显示了 ACME 的各表及它们之间的关系。

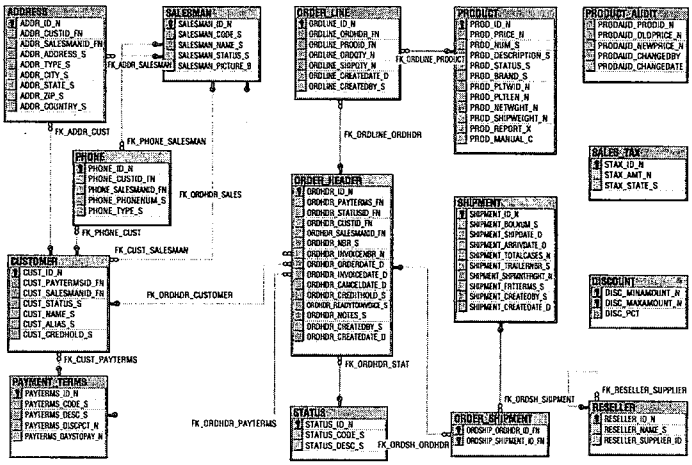


图 B-1 ACME 数据库架构

www.TopSage.com

B.4　列约束和默认值

ACME 中的列约束基于业务规则和这个附录前面所讨论的设计原理。例如，在为订单开具发票之前，发票号和发票日期是未定义的；取消的日期仅对被取消的订单才有意义，因此，ORDER_HEADER 表中的字段 ORDHDR_INVOICENBB_N、ORDHDR_INVOICEDATE_D 和 ORDHDR_CANCELDATE_D 允许 NULL 值。ACME 仅为货运条款定义了两个有效值，预付（PPD）和由收件人付款（COL），因此，在 SHIPMENT 表的 SHIPMENT_FRTTERMS_S 列上有一个检查约束。大多数表都使用引用完整性约束来强制表之间关系。每一个表都有一个**无意义主键**，也就是说，每个表中的每条记录都有一个唯一数字用于表联接和其他目的，仅对数据库程序员可用，对其他非技术人员是隐藏的。

交叉参考：填充无意义主键的过程在不同的实现之间是不同的。Oracle 使用序列，Microsoft SQL Server 通常使用标识列，而 DB2 允许这两种方式。参阅第 4 章了解更多信息。

有些列具有默认值。例如，ORDER_HEADER.ORDHDR_READYTOINVOICE_S 列的值 Y 表明这个订单已完成，已发货，并由销售部门结束，因此，要准备开发票。这个列的默认值是 N，因此，当创建一个新订单时，会自动填充默认值。

B.5　索引

ACME 中的所有主键和外键列都有索引。索引都具有前缀 IDX_。

注意：对于这种规模的数据库而言，外键上的索引是不必要的甚至是无效的，但不管怎么说，我们创建了这些索引，如同在处理拥有数千客户和数百万订单的一个大型数据库一样。

B.6　视图

ACME 数据库具有大量视图。每个视图都基于一个或多个表。视图说明了本书中介绍的一个特定 SQL 脚本。所有视图都具有前缀 V_。

B.7　创建 ACME 数据库对象的 SQL 脚本

下面语句在 DB2 中创建 ACME 数据库：

```
--
-- TABLE: ADDRESS
--

CREATE TABLE ADDRESS(
    ADDR_ID_N           INTEGER       NOT NULL,
    ADDR_CUSTID_FN      INTEGER,
    ADDR_SALESMANID_FN  INTEGER,
    ADDR_ADDRESS_S      VARCHAR(60),
    ADDR_TYPE_S         VARCHAR(8),
    ADDR_CITY_S         VARCHAR(18)   NOT NULL,
    ADDR_STATE_S        CHAR(2),
    ADDR_ZIP_S          VARCHAR(10)   NOT NULL,
    ADDR_COUNTRY_S      CHAR(3),
    CONSTRAINT CHK_ADDR_TYPE CHECK
       (ADDR_TYPE_S IN ('BILLING', 'SHIPPING')),
    CONSTRAINT PK_ADDRPRIMARY PRIMARY KEY (ADDR_ID_N)
);
--
-- TABLE: CUSTOMER
--

CREATE TABLE CUSTOMER(
    CUST_ID_N             INTEGER          NOT NULL,
```


www.TopSage.com

TopSage.com

```
    CUST_PAYTERMSID_FN    INTEGER,
    CUST_SALESMANID_FN    INTEGER,
    CUST_STATUS_S         VARCHAR(1)        DEFAULT 'Y' NOT NULL,
    CUST_NAME_S           VARCHAR(50)       NOT NULL,
    CUST_ALIAS_S          VARCHAR(15),
    CUST_CREDHOLD_S       VARCHAR(1)        DEFAULT 'Y' NOT NULL,
    CONSTRAINT CHK_CUST_STATUS CHECK (CUST_STATUS_S IN ('N', 'Y')),
    CONSTRAINT CHK_CUST_CREDHOLD CHECK (CUST_CREDHOLD_S IN
    ('N', 'Y')),
    CONSTRAINT PK_CUSTPRIMARY PRIMARY KEY (CUST_ID_N)
);

--
-- TABLE: ORDER_HEADER
--

CREATE TABLE ORDER HEADER(
    ORDHDR_ID_N                  INTEGER       NOT NULL,
    ORDHDR_PAYTERMS_FN           INTEGER,
    ORDHDR_STATUSID_FN           INTEGER,
    ORDHDR_CUSTID_FN             INTEGER,
    ORDHDR_SALESMANID_FN         INTEGER,
    ORDHDR_NBR_S                 VARCHAR(30)   NOT NULL,
    ORDHDR_INVOICENBR_N          INTEGER,
    ORDHDR_ORDERDATE_D           DATE,
    ORDHDR_INVOICEDATE_D         DATE,
    ORDHDR_CANCELDATE_D          DATE,
    ORDHDR_CREDITHOLD_S          CHAR(1),
    ORDHDR_READYTOINVOICE_S      CHAR(1)       DEFAULT 'N',
    ORDHDR_NOTES_S               VARCHAR(60),
    ORDHDR_CREATEDBY_S           VARCHAR(10),
    ORDHDR_CREATEDATE_D          DATE,
    CONSTRAINT CHK_ORDHDR_READY CHECK
       (ORDHDR_READYTOINVOICE_S IN ('N', 'Y')),
    CONSTRAINT CHK_ORDHDR_CREDH CHECK
       (ORDHDR_CREDITHOLD_S IN ('N', 'Y')),
    CONSTRAINT PK_ORDHDRPRIM PRIMARY KEY (ORDHDR_ID_N),
    CONSTRAINT IDX_ORDHDR ORDNBR UNIQUE (ORDHDR_NBR_S)
);

--
-- TABLE: ORDER_LINE
--

CREATE TABLE ORDER_LINE(
    ORDLINE_ID_N             INTEGER       NOT NULL,
    ORDLINE_ORDHDRID_FN      INTEGER       NOT NULL,
    ORDLINE_PRODID_FN        INTEGER,
    ORDLINE_ORDQTY_N         INTEGER,
    ORDLINE_SHIPQTY_N        INTEGER,
    ORDLINE_CREATEDATE_D     DATE,
    ORDLINE_CREATEDBY_S      VARCHAR(10),
    CONSTRAINT PK_ORDLINEPRIM PRIMARY KEY (ORDLINE_ID_N)
);

--
-- TABLE: ORDER_SHIPMENT
--

CREATE TABLE ORDER_SHIPMENT(
    ORDSHIP_ORDHDR_ID_FN         INTEGER   NOT NULL,
    ORDSHIP_SHIPMENT_ID_FN       INTEGER   NOT NULL,
    CONSTRAINT PK_ORDHDRSHIP PRIMARY KEY
       (ORDSHIP_ORDHDR_ID_FN, ORDSHIP_SHIPMENT_ID_FN)
);

--
-- TABLE: PAYMENT_TERMS
--

CREATE TABLE PAYMENT_TERMS(
    PAYTERMS_ID_N            INTEGER            NOT NULL,
    PAYTERMS_CODE_S          VARCHAR(6),
    PAYTERMS_DESC_S          VARCHAR(60),
    PAYTERMS_DISCPCT_N       DECIMAL(5,2),
    PAYTERMS_DAYSTOPAY_N     INTEGER,
    CONSTRAINT PK_PAYTERMS PRIMARY KEY (PAYTERMS_ID_N)
);
```


www.TopSage.com

```
--
-- TABLE: PHONE
--

CREATE TABLE PHONE(
    PHONE_ID_N           INTEGER              NOT NULL,
    PHONE_CUSTID_FN      INTEGER,
    PHONE_SALESMANID_FN  INTEGER,
    PHONE_PHONENUM_S     VARCHAR(20),
    PHONE_TYPE_S         VARCHAR(20),
    CONSTRAINT CHK_PHONE_TYPE CHECK
       (PHONE_TYPE_S_IN ('PHONE', 'FAX')),
    CONSTRAINT PK_PHONERIMARY PRIMARY KEY (PHONE_ID_N)
);

--
-- TABLE: PRODUCT
--

CREATE TABLE PRODUCT(
    PROD_ID_N               INTEGER            NOT NULL,
    PROD_PRICE_N            DECIMAL(10,2),
    PROD_NUM_S              VARCHAR(10),
    PROD_DESCRIPTION_S      VARCHAR(44)        NOT NULL,
    PROD_STATUS_S           CHAR(1)            DEFAULT 'Y',
    PROD_BRAND_S            VARCHAR(20)        NOT NULL,
    PROD_PLTWID_N           DECIMAL(5,2)       NOT NULL,
    PROD_PLTLEN_N           DECIMAL(5,2)       NOT NULL,
    PROD_NETWGHT_N          DECIMAL(10,3),
    PROD_SHIPWEIGHT_N       DECIMAL(10,3),
    PROD_REPORT_X           XML,
    PROD_MANUAL_C           CLOB,
    CONSTRAINT CHK_PRODSTATUS CHECK (PROD_STATUS_S in ('N', 'Y')),
    CONSTRAINT PK_PRODUCTPRIM PRIMARY KEY (PROD_ID_N)
);

--
-- TABLE: SALESMAN
--

CREATE TABLE SALESMAN(
    SALESMAN_ID_N           INTEGER       NOT NULL,
    SALESMAN_CODE_S         VARCHAR(2)    NOT NULL,
    SALESMAN_NAME_S         VARCHAR(50)   NOT NULL,
    SALESMAN_STATUS_S       CHAR(1)       DEFAULT 'Y',
    SALESMAN_PICTURE_B      BLOB,
    CONSTRAINT CHK_SALESSTATUS CHECK (SALESMAN_STATUS_S in ('N', 'Y')),
    CONSTRAINT PK_SALESMANPRIM PRIMARY KEY (SALESMAN_ID_N)
);

--
-- TABLE: SHIPMENT
--

CREATE TABLE SHIPMENT(
    SHIPMENT_ID_N           INTEGER       NOT NULL,
    SHIPMENT_BOLNUM_S       VARCHAR(6),
    SHIPMENT_SHIPDATE_D     DATE,
    SHIPMENT_ARRIVDATE_D    DATE,
    SHIPMENT_TOTALCASES_N   INTEGER,
    SHIPMENT_TRAILERNBR_S   VARCHAR(12),
    SHIPMENT_SHPMNTFRGHT_N  DECIMAL(12,2),
    SHIPMENT_FRTTERMS_S     VARCHAR(3),
    SHIPMENT_CREATEDBY_S    VARCHAR(10),
    SHIPMENT_CREATEDATE_D   TIMESTAMP,
    CONSTRAINT CHK_SHIPFRTTERMS CHECK
       (SHIPMENT_FRTTERMS_S IN ('COL', 'PPD')),
    CONSTRAINT PK_SHIPMENTRPRIM PRIMARY KEY (SHIPMENT_ID_N)
);

--
-- TABLE: STATUS
--

CREATE TABLE STATUS(
    STATUS_ID_N         INTEGER           NOT NULL,
    STATUS_CODE_S       CHAR(2),
```


www.TopSage.com

```
    STATUS_DESC_S       VARCHAR(30),
    CONSTRAINT PK_STATUSPRIM PRIMARY KEY (STATUS_ID_N)
);

--
-- TABLE: DISCOUNT
--

CREATE TABLE DISCOUNT
(
    DISC_MINAMOUNT_N        DECIMAL(14,4) NOT NULL,
    DISC_MAXAMOUNT_N        DECIMAL(14,4) NOT NULL,
    DISC_PCT                DECIMAL(5,3),
    CONSTRAINT PK_DISCOUNT PRIMARY KEY
      (DISC_MINAMOUNT_N, DISC_MAXAMOUNT_N)
);

--
-- TABLE: RESELLER
--

CREATE TABLE RESELLER
(
    RESELLER_ID_N           INT                     NOT NULL,
    RESELLER_NAME_S         VARCHAR (30),
    RESELLER_SUPPLIER_ID    INT,
    CONSTRAINT PK_RESELLER PRIMARY KEY (RESELLER_ID_N)
);

--
-- TABLE: PRODUCT_AUDIT
--

CREATE TABLE PRODUCT_AUDIT
(
    PRODAUD_PRODID_N     INTEGER,
    PRODAUD_OLDPRICE_N   DECIMAL(12,2),
    PRODAUD_NEWPRICE_N   DECIMAL(12,2),
    PRODAUD_CHANGEDBY    VARCHAR(10),
    PRODAUD_CHANGEDATE   DATE
);

--
-- TABLE: SALES_TAX
--

CREATE TABLE SALES_TAX
(
    STAX_ID_N     INTEGER NOT NULL,
    STAX_AMT_N    DECIMAL(6,3),
    STAX_STATE_S CHAR(2),
    CONSTRAINT PK_SALES_TAX PRIMARY KEY (STAX_ID_N)
);

--
-- INDEXES
--

CREATE INDEX IDX_ADDR_CUST ON ADDRESS(ADDR_CUSTID_FN);

CREATE INDEX IDX_CUST_PAYTERMS ON CUSTOMER(CUST_PAYTERMSID_FN);

CREATE INDEX IDX_CUST_SALESMAN ON CUSTOMER(CUST_SALESMANID_FN);

CREATE INDEX IDX_ORDHDR_CUST ON ORDER_HEADER(ORDHDR_CUSTID_FN);

CREATE INDEX IDX_ORDHDR_STATUS ON ORDER_HEADER(ORDHDR_STATUSID_FN);

CREATE INDEX IDX_ORDHDR_PAYTERM ON ORDER_HEADER(ORDHDR_PAYTERMS_FN);

CREATE INDEX IDX_ORDHDR_SALES ON ORDER_HEADER(ORDHDR_SALESMANID_FN);

CREATE INDEX IDX_ORDLINE_ORDHDR ON ORDER_LINE(ORDLINE_ORDHDRID_FN);

CREATE INDEX IDX_ORDLINE_PROD ON ORDER_LINE(ORDLINE_PRODID_FN);

CREATE INDEX IDX_ORDSHIP_ORD ON ORDER_SHIPMENT(ORDSHIP_ORDHDR_ID_FN);

CREATE INDEX IDX_ORDSHIP_SHIP
```


www.TopSage.com

大象网 TopSage.com

```
ON ORDER_SHIPMENT(ORDSHIP_SHIPMENT_ID_FN);

CREATE INDEX IDX_PHONE_CUST ON PHONE(PHONE_CUSTID_FN);

CREATE UNIQUE INDEX idx_cust_name ON CUSTOMER(cust_name_s DESC);

CREATE UNIQUE INDEX idx_cust_name_als
ON CUSTOMER(cust_name_s, cust_alias_s);

CREATE INDEX IDX_RESEL_RSSPID ON RESELLER(RESELLER_SUPPLIER_ID);

--
-- FOREIGN KEYS
--

ALTER TABLE ADDRESS ADD CONSTRAINT FK_ADDR_CUST
    FOREIGN KEY (ADDR_CUSTID_FN)
    REFERENCES CUSTOMER(CUST_ID_N);

ALTER TABLE ADDRESS ADD CONSTRAINT FK_ADDR_SALESMAN
    FOREIGN KEY (ADDR_SALESMANID_FN)
    REFERENCES SALESMAN (SALESMAN_ID_N);

ALTER TABLE CUSTOMER ADD CONSTRAINT FK_CUST_PAYTERMS
    FOREIGN KEY (CUST_PAYTERMSID_FN)
    REFERENCES PAYMENT_TERMS(PAYTERMS_ID_N);

ALTER TABLE CUSTOMER ADD CONSTRAINT FK_CUST_SALESMAN
    FOREIGN KEY (CUST_SALESMANID_FN)
    REFERENCES SALESMAN(SALESMAN_ID_N);

ALTER TABLE ORDER_HEADER ADD CONSTRAINT FK_ORDHDR_PAYTERMS
    FOREIGN KEY (ORDHDR_PAYTERMS_FN)
    REFERENCES PAYMENT_TERMS(PAYTERMS_ID_N);

ALTER TABLE ORDER_HEADER ADD CONSTRAINT FK_ORDHDR_CUSTOMER
    FOREIGN KEY (ORDHDR_CUSTID_FN)
    REFERENCES CUSTOMER(CUST_ID_N);

ALTER TABLE ORDER_HEADER ADD CONSTRAINT FK_ORDHDR_STAT
    FOREIGN KEY (ORDHDR_STATUSID_FN)
    REFERENCES STATUS(STATUS_ID_N);

ALTER TABLE ORDER_HEADER ADD CONSTRAINT FK_ORDHDR_SALES
    FOREIGN KEY (ORDHDR_SALESMANID_FN)
    REFERENCES SALESMAN(SALESMAN_ID_N);

ALTER TABLE ORDER_LINE ADD CONSTRAINT FK_ORDLINE_ORDHDR
    FOREIGN KEY (ORDLINE_ORDHDRID_FN)
    REFERENCES ORDER_HEADER(ORDHDR_ID_N);

ALTER TABLE ORDER_LINE ADD CONSTRAINT FK_ORDLINE_PRODUCT
    FOREIGN KEY (ORDLINE_PRODID_FN)
    REFERENCES PRODUCT(PROD_ID_N);

ALTER TABLE ORDER_SHIPMENT ADD CONSTRAINT FK_ORDSH_ORD
    FOREIGN KEY (ORDSHIP_ORDHDR_ID_FN)
    REFERENCES ORDER_HEADER(ORDHDR_ID_N);

ALTER TABLE ORDER_SHIPMENT ADD CONSTRAINT FK_ORDSH_SHIP
    FOREIGN KEY (ORDSHIP_SHIPMENT_ID_FN)
    REFERENCES SHIPMENT(SHIPMENT_ID_N);

ALTER TABLE PHONE ADD CONSTRAINT FK_PHONE_CUST
    FOREIGN KEY (PHONE_CUSTID_FN)
    REFERENCES CUSTOMER(CUST_ID_N);

ALTER TABLE PHONE ADD CONSTRAINT FK_SALESMAN_CUST
    FOREIGN KEY (PHONE_SALESMANID_FN)
    REFERENCES SALESMAN (SALESMAN_ID_N);

ALTER TABLE RESELLER ADD CONSTRAINT FK_RESEL_SUP
  FOREIGN KEY (RESELLER_SUPPLIER_ID)
   REFERENCES RESELLER (RESELLER_ID_N);

--
-- VIEWS
--
```


www.TopSage.com

```
--
-- VIEW V_CUSTOMER_TOTALS
--

CREATE VIEW V_CUSTOMER_TOTALS
(
CUSTOMER_NAME,
ORDER_NUMBER,
TOTAL_PRICE
)
 AS
(
SELECT   customer.cust_name_s,
         order_header.ordhdr_nbr_s,
         sum(product.prod_price_n * order_line.ordline_ordqty_n)
FROM     customer,
         order_header,
         order_line,
         product
WHERE    customer.cust_id_n = order_header.ordhdr_custid_fn
AND      order_header.ordhdr_id_n = order_line.ordline_ordhdrid_fn
AND      product.prod_id_n = order_line.ordline_prodid_fn
AND      order_line.ordline_ordqty_n IS NOT NULL
GROUP BY customer.cust_name_s,
         order_header.ordhdr_nbr_s
);

--
-- CREATE VIEW V_CUSTOMER_STATUS
--

CREATE VIEW V_CUSTOMER_STATUS
(
   NAME,
   STATUS
)
AS
SELECT       cust_name_s,
             cust_status_s
FROM         customer;


--

-- CREATE VIEW V_PHONE NUMBER
--
CREATE VIEW V_PHONE_NUMBER
(
  PHONE_ID,
  PHONE_NUMBER
)
AS

SELECT      phone_id_n,
            phone_phonenum_s
FROM        phone
WHERE       phone_type_s = 'PHONE';


--
-- CREATE VIEW V_FAX NUMBER
--

CREATE VIEW V_FAX_NUMBER
(
  FAX_ID,
  FAX_NUMBER
)
AS
SELECT       phone_id_n,
             phone_phonenum_s
FROM          phone
WHERE         phone_type_s = 'FAX'
WITH CHECK OPTION;


--
-- CREATE VIEW V_CUSTOMER_TOTALS_OVER_15000
```


www.TopSage.com

```
--

CREATE VIEW V_CUSTOMER_TOTALS_OVER_15000
AS
SELECT    *
FROM      v_customer_totals
WHERE     total_price > 15000;


--
-- CREATE VIEW V_CONTACT_LIST
--

CREATE VIEW V_CONTACT_LIST
(
 NAME,
 PHONE_NUMBER,
 CONTACT_TYPE
)
AS
SELECT      cust_name_s,
       phone_phonenum_s,
            'customer'
FROM        customer,
            phone
WHERE       cust_id_n = phone_custid_fn
AND         phone_type_s = 'PHONE'
UNION
SELECT      salesman_name_s,
            phone_phonenum_s,
            'salesperson'
FROM        salesman,
            phone
WHERE       salesman_id_n = phone_salesmanid_fn
AND         phone_type_s = 'PHONE';

--
--CREATE VIEW V_WILE_BESS_ORDERS
--
CREATE VIEW V_WILE_BESS_ORDERS
(
 ORDER_NUMBER,
 ORDER_DATE
)
AS
SELECT      ordhdr_nbr_s,
            ordhdr_orderdate_d
FROM        order_header
WHERE       ordhdr_custid_fn IN
              (
             SELECT  cust_id_n
             FROM    customer
             WHERE   cust_name_s = 'WILE BESS COMPANY'
            );
--

-- CREATE VIEW V_CUSTOMER_TOTALS_WILEBESS
--
CREATE VIEW V_CUSTOMER_TOTALS_WILEBESS

AS
SELECT   customer_name,
         total_price
FROM     v_customer_totals
WHERE    customer_name = 'WILE BESS COMPANY';
```

使用相同的脚本集，对数据类型稍作修改就可以在Oracle或Microsoft SQL Server中创建ACME数据库。

注意：上面的脚本已做到尽可能通用；配套网站上的实际SQL语句更加特定于实现。

www.TopSage.com

大家网
TopSage.com

APPENDIX C

附录 C
关系数据库设计基础

本附录提要：

- OLTP 和 OLAP 数据库
- 实体和属性
- 规范化
- 关系数据库设计缺陷

数据库设计仍然更像是一门艺术而非精密的科学。数据库并不是存在于真空之中的某种独立的事物，它服务于某些特殊的商业目的，并且在其设计中得到反映。一般的数据库类型可以分成两大类：操作数据库和分析数据库。操作数据库（OLTP——联机事务处理）处理日常操作：记录数据、打印工资单等。这样一个数据库中的数据更改非常频繁。这种数据库类型的例子包括本书提供的 ACME 数据库。

分析数据库（OLAP——联机分析处理）用于存储历史数据，这些数据用于分析报表、生成统计数据等。这样一个数据库中的信息大多数是静态的；可以添加新数据，但通常不能修改历史数据。此外，OLAP 数据库中的信息以聚合状态存储，因此是非规范化的（参见本章稍后介绍的规范化）；基于特定的目的，可以有很多级别的聚合。设计 OLAP 数据库不在该附录介绍范围内，因此这里只简要提及一下。

注意：在第 1 章中列出了多种数据库类型，全书中的全部例子都使用 ACME 数据库，这是一家虚构硬件公司的一个虚构数据库。

数据库设计过程分为两个阶段。在第一个阶段中，需要确定和定义表和字段，建立关系等。这涉及详尽的商业领域知识。首先收集用户需求并将需求形式化，然后在特定的商业模型中分析这些需求并标识实体；每一个实体都可能成为数据库中的表。

注意：这个过程（和术语）不同于面向对象的数据库，甚至不同于面向对象的分析和设计方法。

在设计 ERD（Entity/Relationship Diagram，实体/关系图）过程中，标识这些表后，要继续定义它们的属性。属性就像一个形容词；它描述一个实体，就像用“甜”来描述糖那样。对于 `Customer` 实体，需要考虑表中列的标识，如姓名、地址、电话号码等。这些属性将成为表中的列。不过，虽然进行精心设计，但这组表还不能称为关系数据库；表间关系才是关系数据库的关键。表之间的逻辑关系通过主-外键对来建立。在 RDBMS 术语中，这种关系被称为父-子关系，其中父表中的每条记录对应子表中的一条或多条记录。这称为一对多关系，是关系数据库的基本特征；RDBMS 中常见的另一个类型是多对多关系，它需要一些中间表用于明确的解决方案。

www.TopSage.com

第二个阶段在一些 RDBMS 包中创建构想的数据库。这个过程绝不是线性的；对数据库需求进行细化时，会发现需要在这两个阶段之间交替进行。本书提供了创建数据库对象的 SQL 语法（参阅第 4 章），它实际上与供应商实现无关。

注意：数据库对象的设计可以完全独立于特定 RDBMS 软件，也可以结合这些软件进行。尽管有人可能认为专门针对 Oracle 或 Microsoft 进行设计会提高性能，但坚持一种简洁方法（在可能的情况下）可以使设计更具可移植性并会减少令人头痛的维护和升级活动。

C.1 标识实体和属性

考虑一下数据库的目的。例如，要创建用于记录客户下订单的数据库，可以从 CUSTOMER 表开始。一个客户的属性包括什么？姓名、物理地址、产品订单、订单信息、信用等级等。

这个阶段的一个重要部分是选择适当的数据类型来表示所需的/标识的属性。选择数据类型时，需要考虑很多平衡关系，通常涉及性能与逻辑设计的清晰性。例如，在 Oracle 中，销售额和价格适合使用数值数据类型（并且 Oracle 主张使用 NUMBER 数据类型，而不是像 DECIMAL、INTEGER 等这样特殊的数值类型）。在 Microsoft SQL Server 中把这个数据确定为 INTEGER 将导致一个错误，因为将把这个值四舍五入为美元值；可以选择 MONEY 或 SMALLMONEY 数据类型。除此之外，在数值字段上搜索通常比在包含字符的列上执行得更快。当需要存储像 XML 图像或二进制文件这样更复杂的信息时，使用适合该列数据的数据类型。

交叉参考：参阅第 3 章了解有关数据类型的更多信息。

表 C-1 是在第 1 章中讨论的平面文件设计的一个例子——在单一表中罗列了全部内容。这样的设计几乎不能适应任何企业，经营厨房餐桌的公司除外。要创建关系数据库，这个原始表必须规范化。

表 C-1 “单表”设计

CUST_NAME	ADDRESS	SALES_REGION	SALES_REGION_ID	REGION_ID	DESCRIPTION	CREDIT_RATING
Lone Dove Ent.	1234 Elm Street, Salem, OR 95679	北太平洋	3	NA1234-89	Paper Pulp	很好
ACME, Inc.	567 Pine Street, Chicago, IL 07891	中西部	4	GA4358-92	Sawdust	好

C.2 规范化

标识基表并建立数据库实体间的关系后．就可以进行数据库**规范化**了。

规范化是指将大型数据库表分解成许多更小的表从而减少数据冗余。当然，这意味着修改表之间的一些关系，甚至发现新实体。规范化的步骤被称为**范式**，迄今为止，数据库理论家已经定义了 5 个范式，这是数据库能达到的最大规范化程度。

正如看到的那样，设计数据库是一个反复的过程——不断重复同样的步骤直到获得满意的结果。设计的规范化阶段同样如此。然而，完全规范化的数据库是不存在的：首先，任何超越第三范式的数据库都会出现明显的性能问题（因为它要应付单个表中数据所需的所有额外联接）；其次，要为进一步将数据分成第四和第五范式确定线索很困难的。要根据个人判断来确定数据库规范化的程度。单凭经验来说，第三范式足以应付大多数应用。

范式是应用于表的一组规则，用于确保没有冗余信息（不仅指重复信息，还指在请求时可能派生的信息）。例如，如果除了员工的出生日期外，还存储了员工的年龄，则每年都要更新员工的年龄列。

C.2.1 第一范式

第一范式处理数据中重复的组。例如，对表 C-1 中显示的 CUSTOMER 表进行规范化。首先记录所

www.TopSage.com

有相关信息：客户姓名、地址、下的订单等等。

尽管确信订单是属于客户的，但把它们放在 CUSTOMER 表中将是存储空间的一个巨大浪费（并且违反了关系数据库的原则）。如果将订单和客户放在同一个表中，则需要为每一个新订单复制客户信息，而且需要为每个下相同订单的客户复制订单信息（如描述、供应商等）。合乎逻辑的（也是关系的）解决方案是从 CUSTOMER 表中删除订单并创建一个 ORDER 表。每一个客户将得到一个唯一的 ID，这个 ID 就是 CUSTOMER 表的主键，而所有关于订单的信息都将移到 ORDER 表中。为了把客户和他们的订单联系在一起，ORDER 表必须把 CUSTOMER_ID 的列作为外键，如表 C-2 和表 C-3 所示。

表 C-2　　第一范式之后的 CUSTOMER 表

CUST_ID	CUST_NAME	SALES_REGION	SALES_REGION_ID	ADDRESS	CREDIT_RATING
1	Lone Dove Ent.	北太平洋	3	1234 Elm Street, Salem, OR 95679	很好
2	ACME Corp	中西部	4	567 Pine Street, Chicago, IL 07891	好

表 C-3　　新的 ORDER 表

CUST-ID	ORDER-ID	PRODUCT-ID	DESCRIPTION
1	NA1234-89	P01245	Paper Pulp
2	GA4358-92	SD3457	Sawdust
1	HA5432-02	WC7863	Wood Chips

尽管为客户信息删除了冗余内容，但这个解决方案并没有解决信息的冗余问题。订单是唯一的，而 CUSTOMER 和 ORDER 表之间是一对多关系，其中每一个客户可以有一个或多个订单，但每个订单有且仅有一位客户。在这里，订单是针对某些特定产品的；将产品信息与订单并排记录，其结果是为同一产品的新订单重复输入了产品信息。此外，这个表的设计隐式地假设了一个订单中有且仅有一种产品，在实际生活中大多不会如此；因此，该列是进一步规范化的对象。

对表中其他组实施规范化过程：客户地址、银行账户、信用等级等。

C.2.2 第二范式

第二范式消除冗余数据：确保没有依赖部分（或全部）主键的非键属性（字段）。ORDER 表需要进一步规范化：可以用 ORDER_ID 和 PRODUCT_ID 构成复合主键，但产品描述只依赖于 PRODUCT_ID，而不是 ORDER_ID。如果每一个订单只由一种产品组成，则一切顺利；但是某个客户可能为节省运费发出包含多种产品的订单。而且产品描述可能需要更改——那么您不得不更新包含这个产品的每一个订单。这就需要引入 PRODUCT 表。在 ORDER 表中保留 PRODUCT_ID，并将其作为外键。

通过移动 CREDIT_RATING 列到 CREDIT 表（没有显示出来），可以再次将表 CUSTOMER 进一步规范化到第一范式（1NF）。参见表 C-4、表 C-5 和表 C-6。

表 C-4　　第二范式后的 CUSTOMER 表

CUST_ID	CUST_NAME	SALES_REGION	SALES_REGION_ID	ADDRESS
1	Lone Dove Ent.	北太平洋	3	1234 Elm Street, Salem, OR 95679
2	ACME Corp.	中西部	4	567 Pine Street, Chicago, IL 07891

表 C-5　　新的 ORDER 表

CUST_ID	ORDER_ID	PRODUCT_ID
1	NA1234-89	P01245
2	GA4358-92	SD3457
1	HA5432-02	WC7863

表 C-6　　新的 PRODUCT 表

PRODUCT-ID	QUANTITY	DESCRIPTION
P01245	20 000	Paper Pulp
SD3457	400 000	Sawdust
WC7863	70 000	Wood Chips

www.TopSage.com

大家网 TopSage.com

新设计仍然依赖于产品和订单之间的一对一关系。为了能在一个订单中包含多个产品，还需要引入另一个表来解决多对多关系（表 C-7、表 C-8 和表 C-9）。

表 C-7 新的 ORDER 表

CUST-ID	ORDER-ID
1	NA1234-89
2	GA4358-92
1	HA5432-02

表 C-8 新的 PRODUCT 表

PRODUCT-ID	DESCRIPTION
P01245	Paper Pulp
SD3457	Sawdust
WC7863	Wood Chips

表 C-9 ORDER_PRODUCT 表

PRODUCT-ID	ORDER-ID	QUANTITY
P01245	NA1234-89	20 000
SD3457	GA4358-92	400 000
SD3457	HA5432-02	10 000
WC7863	HA5432-02	70 000

现在每个订单可以包含多个产品而不存储冗余信息（例如，只存储一次产品描述）。

C.2.3 第三范式

第三范式要求没有依赖于其他非键属性的属性（字段）。这意味着只有说明一个实体主键的相关信息才能在放在表中。表 CUSTOMER 包含 SALES_REGION_ID 和 SALES_REGION_NAME，哪一个都没有描述客户。因此这个数据应当移动到一个单独的表中（比如 REGIONS）来达到表 C-10 和表 C-11 中所示的第三范式。

注意：第四范式规定对多重关系进行分离。它只应用于包括一对多和多对多关系的设计。这一原理要求表不包含两个或更多非直接相关的一对多或多对多关系。

第五范式需要隔离语义上关联的多个关系。可能有一些理由甚至要将逻辑关联的属性分隔到单独的表，但是如果决定这样做，则要密切关注性能；谨慎处理。

事情进展得相当不错，但现在有一个问题：客户打开了多个位置，现在一个订单（或一个装运）可能与多个客户关联，并装运到多个目的地；同时，一个客户仍然可以发出多个订单。为了解决多对多的关系，引入了一个中间表。可以将 ACME 数据库中的 ORDER_SHIPMENT 表作为示例（参见附录 B）。ACME 数据库链接了 ORDER 表和 SHIPMENT 表，因为一个装运可以有多个订单，而在多次装运中可能发出每一个订单。ORDER_SHIPMENT 表由两个列组成，这两个列代表外键：一个（ORDSHIP_ORDHDR_ID_FN）来自 ORDER_HEADER 表，另一个（ORDSHIP_SHIPMENT_ID_FN）来自 SHIPMENT 表。

表 C-10 第一范式后的 CUSTOMER 表

CUST_ID	CUST_NAME	SALES_REGION_NAME	SALES_REGION_ID	ADDRESS
1	Lone Dove Ent.	北太平洋	3	1234 Elm Street, Salem, OR 95679
2	ACME Corp.	中西部	4	567 Pine street, Chicago, IL 07891

表 C-11 新的 REGION 表

SALES_REGION_ID	SALES_REGION_NAME
3	北太平洋
4	中西部
5	北大西洋

www.TopSage.com

C.3 指定约束

应该考虑用一些约束来强制数据库及其包含的数据的完整性。主键和外键强制**引用完整性**是指子表中的每条记录都链接到父表中的一条记录（也就是说，没有**孤立记录**）。在数据库设计过程中，数据库模型不断演化，但是应当遵循 RDBMS 的原理。

主键和 UNIQUE 约束还强制了**实体完整性**——这个概念是指确保表中的一行在表中是唯一的（也就是说，即使一些列中可能有重复的数据，但是其列值的组合在整个表中应是唯一的）。实体完整性也强制使用 NOT NULL 约束（说明这个特定的列中不允许有 NULL 值）。应该为创建的每一个表选择一个主键，并为每一个表中的每一个不应为 NULL 的列应用 NOT NULL 约束（例如，客户的一条记录必须有一个 CUST_ID 字段并且字段的值必须是唯一的——任意两个客户的 ID 都不相同）。

域完整性是指数据自身，并且通过使用适当的数据类型来强制执行，例如，CHECK 约束和 DEFAULT。CHECK 约束确保只有特定的数据（在特定范围或格式内）被输入到这个列中；DEFAULT 约束指定如果在插入新记录时没有为其提供一个值时则使用该列的默认值。

交叉参考：在第 4 章和第 5 章中讨论了约束。

数据类型还能防止输入无效数据。在一个 NUMBER 列中不能有 VARCHAR 数据，而 SQL Server 的 IMAGE 数据不可能插入到一个 INTEGER 数据类型的列中。

除了这些内置的完整性约束外，通过使用触发器、存储过程和其他一些 RDBMS 特定功能可以强制执行自定义约束（参阅第 14 章了解更多信息）。

C.4 OLTP 与 OLAP 设计

数据库设计反映了所创建的数据库的用途。OLTP（联机事务处理）需要高度规范化的数据库来记录企业需要的每分钟详细信息。换言之，OLTP 是一种记录数据库。允许捕获数据的同一设计在实现检索数据时并不理想。规范化设计可能需要许多 JOIN 和聚合操作在可以显示数据之前按照有意义的方式来组合数据。这种商业需求以及与日俱增的**商业智能**应用程序需要意味着一种完全不同的数据库设计数据仓库。

OLAP（联机分析处理）本质上是一种只读数据库，至少从用户的角度看是如此。为了提供更好的性能，OLAP 数据库中的数据通常是非规范化的并且通常是聚合的。

C.4.1 数据仓库和数据市场

数据仓库为决策支持提供基础，其中包含数据快照——大多数是静态历史数据。数据仓库之父 Bill Inmon 将数据仓库定义为：

- 面向主题——数据库中的所有数据围绕一些对象、概念、事件来组织；
- 时间变量——沿时间线跟踪数据，因此可以发现趋势；
- 稳定——收集到数据仓库的数据不应修改；本质上成为只读的；
- 集成的——仓库数据是自包含的；也就是说，它从多个不同的源收集数据并把所有必要的数据集成到一个单独的场所中。

数据市场是数据仓库的子集。类似情况符合现实中的仓库操作，在单独的场所（仓库）收集产品（信息），然后分发到较小的场所——店铺或市场。在某种意义上，数据仓库是一般意义的仓库，而数据市场是专门的数据库，专门针对一个特定用户或一组用户。

关系数据仓库通过 3NF OLTP 表示，维度数据仓库和数据市场都是围绕两种基本架构进行设计的——星型和雪花型。

C.4.2 星型和雪花型架构

星型和雪花型架构使用相同的关系原理，并且所有 RDBMS 都基于这一原理。它们引入了**事实数**

www.TopSage.com

据表和维度表的概念。事实数据表包含一些实际的、可定量的数据；维度表包含一些定性的、说明性数据。前者通常包含大量数据（数百万行），而后者可能小得多。**事实**和**维度**这两个名称并不是固定不变的，而是取决于特定的情况。在区别两者时可以使用一个简单的经验法则：事实数据表包含要分析的“内容”，而维度表包含要“通过”什么来分析事实数据。名称“星型”是指可视化——由维度表包围的一个事实数据表，所有维度表都通过主/外键关系与这个事实数据表关联。

雪花型架构本质上是星型架构的变体，其中维度表具有其自己的维度。架构图类似于一朵雪花——从而得名。雪花型架构减少了维度表的数据冗余，通过处理更多的粒度进行“分析”。需要在性能与聚合级别之间权衡。

C.5 关系数据库设计的缺陷

正如谚语所言，对于拿着一把锤子的人来说，所有的问题看上去都像是钉子。这对于尝试进行数据库设计时常犯错误的人而言再贴切不过了。

具有非关系数据库经验的程序员可能倾向于设计类似于层次结构或网络数据库的数据库，甚至平面文件和电子表格设计，如第1章所述。

如果是从头开始设计数据库——很幸运，您可以使用数据库著作中能找到的任何一项技术。但更多时候需要重新设计数据库以适应新的商业需求、提高性能等。无论做什么，不要将现有的数据库结构作为一个新数据库的基础。对于特定任务来说这是错误的，因为它的设计不能适应要实现的新功能。要提出全新的见解，不要受以往设计的限制。如果能够重用某些部分或片断，那么重新设计可能是不必要的，可能只需要在现有数据库上进行改进。重新设计数据库来保护遗留数据不是一个小任务，应该谨慎处理。

另一个普遍存在的问题来自倾向于利用某个特定RDBMS供应商提供的每一个单独功能。尽管改善了性能（至少有可能，或者大多数时候），但这种方式会将用户限定在那个供应商的产品中，如果把数据库迁移到一个不同的供应商，可能会付出时间和金钱方面的昂贵代价。不管相信与否，曾经有一个时期，dBase、Btrieve、FoxPro和Sybase都很流行。坚持一些合理的原则可能无法充分提升一个数据库的性能，但决定应当随一个不同的RDBMS供应商坚持下去时，就会从中受益，甚至只是使数据库维护任务在某个统一框架下更便捷。

对数据库设计进行简要介绍的目的不是把您变成一位数据库设计专家，而是为迈出设计过程的第一步提供一些帮助。您可以阅读有关这个主题的更全面的书籍。

www.TopSage.com

大家网
TopSage.com

APPENDIX D

附录 D
安装 RDBMS 软件

本附录提要：

- 安装 Oracle 11g
- 安装 IBM DB2 9.5
- 安装 Microsoft SQL Server 2008

请记住，不管安装哪种 RDBMS，都只是出于教育/评估目的。安装 Oracle、IBM DB2 和 Microsoft SQL Server 有多种不同的方式，但就本书的范围而言，推荐您遵照建议的指导和命名约定，以方便 ACME 数据库的创建和填充（不管选择哪个 RDBMS）。ACME 是本书实例和练习中使用的数据库。

三个主要 RDBMS 供应商都有其功能完善的 RDBMS 系统试用版，或者数据库软件的免费版，如 Oracle Database XE、DB2 Express-C 和 Microsoft SQL Server Express 版（参考供应商特定许可协议和条款了解详细内容）。如果系统拥有足够内存、空间和 CPU 容量，就可以选择尝试任何想要的产品。否则，我们推荐安装 DB2 Express（或 DB2 Express-C），它需要的系统资源最少。

警告：在生产环境中安装任何 RDBMS，需要有经验的专家来帮助您。

D.1 安装 Oracle 11g

如果想在机器上安装 Oracle 11g，建议您不要依赖最低硬件要求，而要超过推荐的要求。

D.1.1 在 Linux 上安装 Oracle11g release 1（11.1.0.6.0）企业版

在 Linux 上安装 Oracle 需要扎实的操作系统和 `root` 特权知识。我们假设在服务器上没有安装任何以前版本的 Oracle。

表 D-1 列出了在 Linux 上安装 Oracle 11g 的最低要求。

表 D–1　在 Linux 上安装 Oracle 11g 的最低要求

操作系统	Asianux 2.0-3.0 Oracle Enterprise Linux 4.0-5.0 Red Hat Enterprise Linux 4.0-5.0 SUSE Enterprise Linux 10.0
内核	Asianux 2.0, Oracle Enterprise Linux 4.0, 2.6.9 Red Hat Enterprise Linux 4.0 Asianux 3.0, Oracle Enterprise Linux 5.0, 2.6.18 Red Hat Enterprise Linux 5.0 SUSE Enterprise Linux 10.0 2.6.16.21

www.TopSage.com

续表

内存	RAM：1024MB 交换内存：2×RAM
硬盘	数据库软件和示例数据库文件需要4.7GB。 /tmp目录需要400MB
Web浏览器	Netscape Navigator 7.2或更高 Mozilla 1.7或更高 Firefox 1.0.4或更高
其他	X Windows（Gnome、KDE或任意喜欢的）make、ar、ld、nm可执行文件

*此外，可能需要安装本附录后面讨论的一些额外的OS包。

1. 作为root执行的预安装任务

在开始Oracle RDBMS安装之前必须执行下面的root任务：

（1）配置内核参数。编辑/etc/sysctl.conf并添加下列行：

```
kernel.shmall = 2097152
kernel.shmmax = 2147483648
kernel.shmmni = 4096
kernel.sem = 250 32000 100 128
fs.file-max = 6553600
net.ipv4.ip_local_port_range = 1024 65000
net.core.rmem_default = 4194304
net.core.rmem_max = 4194304
net.core.wmem_default = 262144
net.core.wmem_max = 262144
```

执行sysctl -p命令，应用新设置。

（2）编辑/etc/pam.d/login文件并添加下列行：

```
session required pam_limits.so
```

（3）编辑/etc/security/limits.conf文件并添加下列行：

```
oracle    soft   nproc   2047
oracle    hard   nproc   16384
oracle    soft   nofile  1024
oracle    hard   nofile  65536
```

（4）创建用户oracle，作为dba和oinstall组的成员并具有与其他用户主目录一致的主目录（但是与Oracle的主目录不一样）。由于本书的目的，我们建议用户oracle的密码为“acme”——只是为了一致性。当然，在生产环境中要使用更安全的方式。

可以使用喜欢的GUI工具或命令行界面，例如：

```
# groupadd oinstall
# groupadd dba
# useradd -g oinstall -G dba -md /home/oracle -s /bin/bash oracle
# passwd oracle
Changing password for oracle.
New password: ****
Reenter New password: ****
Password changed.
```

（5）创建一个Oracle安装目录，并把它分配给用户oracle和oinstall组：

```
mkdir /u01
chown -R oracle:oinstall /u01
```

2. 作为用户oracle执行的预安装任务

在开始Oracle RDBMS安装之前，您必须作为用户oracle（在前面的步骤中创建的）执行下面的任务：

（1）如表D-2所述，在oracle账户的/home/oracle/.bash_profile或/home/oracle/.bashrc中设置环境变量。例如：

```
export ORACLE_BASE=/u01/app/oracle
export ORACLE_HOME=/u01/app/oracle/product/11.1.0
export ORACLE_SID=ACME
export PATH=$ORACLE_HOME/bin:/bin:/usr/bin:/usr/bin/X11:$PATH
export LD_LIBRARY_PATH=$ORACLE_HOME/lib
```

表D-2 在Linux上安装Oracle 11g时必要的环境变量

变量	说明	推荐值
ORACLE_BASE	Oracle软件结构的顶层	/u01/app/oracle
ORACLE_HOME	包含针对一个特定发行版的软件的目录	/u01/app/oracle/product/11.1.0

www.TopSage.com

续表

变量	说明	推荐值
ORACLE_SID	在安装期间 Oracle 服务器实例使用的系统标识符	ACME（出于本书的目的）
PATH	Shell 对可执行文件的搜索路径（顺序很重要）	$ORACLE_HOME/bin;/usr/bin:/bin:/usr/bin/X11:/usr/local/bin（如果存在）
LD_LIBRARY_PATH	Shell 对库的搜索路径（顺序很重要）	在 LD_LIBRARY_PATH 变量的开始添加 $ORACLE_HOME/lib

（2）为当前的 oracle 会话（这个会话将执行这个安装）更新环境：

```
# /home/oracle/.bash_profile
```

或

```
# /home/oracle/.bashrc
```

3. 安装指导

现在可以准备开始安装 Oracle RDBMS 了（图 D-1）：

（1）作为用户 oracle，从 Oracle 网站 http://www.oracle.com/technology/software/products/database/index.html 下载和保存 Oracle 11g release 1（11.1.0.6.0），并保存在预定的位置，例如在用户 oracle 的主目录（/home/oracle）中。

（2）使用下面的命令解压缩文件：

```
# unzip linux_11gR1_database.zip
```

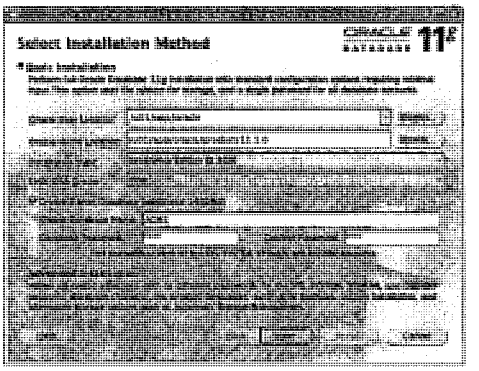


图 D-1 Oracle 11g "Select Installation Method（选择安装方法）" 窗口

（3）开始 Oracle 软件安装过程。假设当前目录仍然是/home/oracle（或者选定任意目录供下载和解压缩文件），提交下面的命令：

```
# database/runInstaller
```

第一个画面提示您选择安装 Oracle 服务器的位置。如果需要则更正 "Oracle 基本位置"、"Oracle 主位置" 和 "UNIX DBA 组"（考虑本书的目的，我们推荐使用默认值）。接受默认 "安装类型"（企业版）。选择 "创建启动数据库" 选项（默认）。在 "全局数据库名" 中输入 ACME，"数据库密码" 和 "确认密码" 中输入 acme，并单击 "Next（下一步）" 按钮。

Note 注意：可以选择不同的安装类型，如 Standard Edition 或 Standard Edition One。对于讨论的内容而言，这些安装步骤会非常相似，但是请注意仍会有一些区别。

Tip 提示：在 Oracle 11g 中，密码区分大小写。

（4）接受 Inventory 目录的默认安装设置和 OS 默认组，如图 D-2 所示，选择 "Next（下一步）" 按钮。

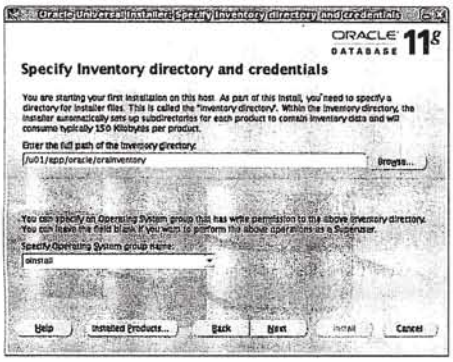
（5）下一步是验证系统满足全部安装需求。请滚动查看每个 "检查" 项并确保 "状态" 为 "成功"。在页面底部的灰色可滚动文本框中显示了详细信息（可以将它复制、粘贴和保存为一个文本文档以便更易于阅读）。如果任何先决条件项是 "通过" 以外的状态，则必须在继续进行之前更正。例如，很可能是因为没有在系统上安装全部必要的 OS 包。在这种情况下，会看到类似下面的信息：

```
============================================================
Checking operating system package requirements ...
...
Checking for gcc-4.1.0; Not found.                    Failed <<<<
Checking for libelf-0.8.5; Not found.                 Failed <<<<
Checking for libaio-devel-0.3.104; Not found.         Failed <<<<
Checking for libstdc++-devel-4.1.0; Not found.        Failed <<<<
Checking for unixODBC-2.2.11; Not found.              Failed <<<<
Checking for unixODBC-devel-2.2.11; Not found.        Failed <<<<
Checking for sysstat-6.0.2; Not found.                Failed <<<<
Check complete. The overall result of this check is:  Failed <<<<
...
Problem: Some packages required for the Oracle Database 11g to
function properly are missing (see above).
Recommendation: Install the required packages before continuing
with the installation.
============================================================
```

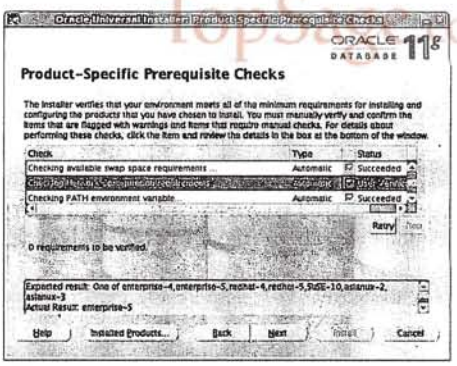
如果有一些包没有安装，那么从安装媒体安装它们，或者从供应商特定的位置下载。这样必须单击 "Cancel（取消）" 按钮（图 D-3）停止 Oracle 11g 安装过程，并在更正问题后从步骤 3 重新启动安装。

www.TopSage.com

检查成功完成后，单击“Next（下一步）”按钮（图D-3）。



图D-2 Oracle 11g“Specify Inventory directory and credentials（指定Inventory目录和证书）”窗口

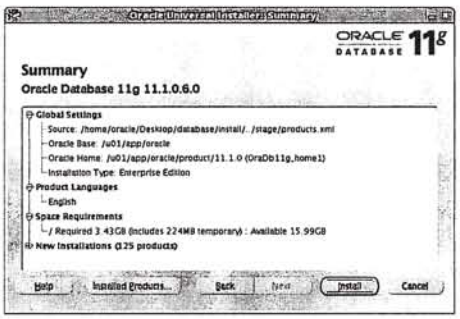


图D-3 “Product-Specific Prerequisites Checks（产品特定必备检查）”窗口

（6）跳过**配置管理器注册**窗口（除非拥有一个有效的Metalink账户并要注册自己的产品）。单击“Next（下一步）”按钮（图D-4）即可。

（7）检查**摘要**窗口，单击“Install（安装）”（图D-5）。

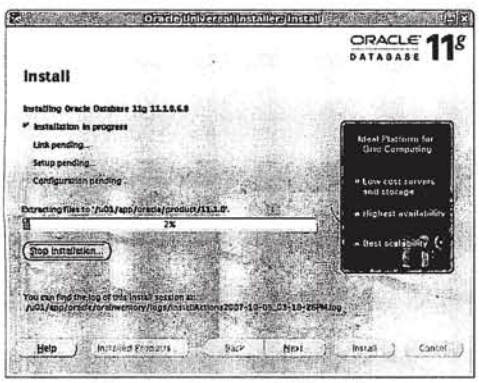
图D-4 “Configuration Manager Registration（配置管理器注册）”窗口



图D-5 “Summary（摘要）”窗口

（8）等待安装过程完成（图D-6）。

（9）当软件安装过程完成时，**配置助手**窗口出现（图D-7）。默认安装配置Oracle网络服务和代理，创建示例数据库并启动Oracle HTTP服务器。



图D-6 “Install（安装）”窗口

图D-7 “Configuration Assistants（配置助手）”窗口

Tip 提示：如果在一台DHCP机器或一台无网络的计算机上安装Oracle，则“配置助手”会失败。有些地方涉及修改/etc/hosts和$ORACLE_HOME/network/admin/listener.ora文件；参考供应商文档了解更多详细内容。

www.TopSage.com

（10）现在要创建数据库（图 D-8）。

（11）当数据库创建完成时，**数据库配置助手**窗口提供解锁用户或修改系统用户密码的选项，如图 D-9 所示。出于本书的目的，我们推荐跳过这个步骤；只单击“OK（确定）”按钮。

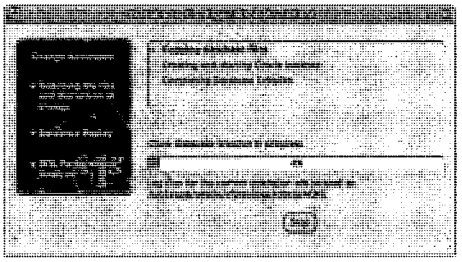


图 D-8 Oracle 11g “Database Configuration Assistant（数据库配置助手）”窗口

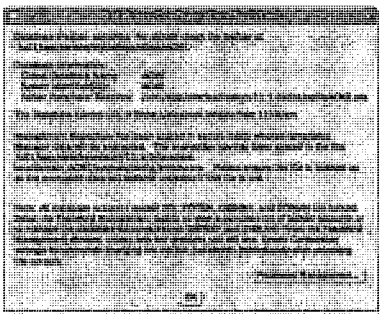


图 D-9 Oracle 11g 在“数据库配置助手”窗口上的数据库创建摘要/密码管理选项

（12）下一个画面提示运行一些配置脚本。作为 root 用户登录并从终端窗口按照指定的顺序运行这些脚本（图 D-10）。当脚本运行完成（图 D-11），单击“OK（确定）”按钮。

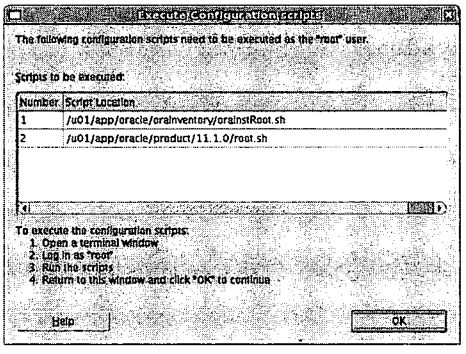


图 D-10 Oracle 11g “Execute Configuration scripts（执行配置脚本）”窗口

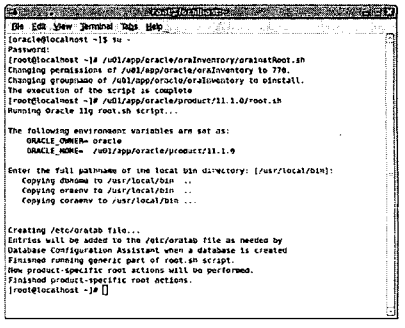


图 D-11 运行 root 脚本

（13）在**安装结束**窗口（图 D-12）中单击“Exit（退出）”，然后单击“Yes（是）”确认。

4. 测试 Oracle 11g 的安装

（1）打开浏览器，输入下面的 URL：`https://localhost:1158/em`（可以使用机器名或 IP 地址代替“localhost”）。选择“Accept this certificate temporarily for this session（暂时为这个会话接受这个认证）”或“Accept this certificate permanently（永久接受这个认证）”，并单击“OK（确定）”按钮（图 D-13）。

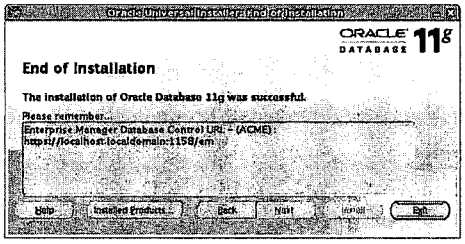


图 D-12 Oracle 11g 的“End of Installation（安装结束）”窗口

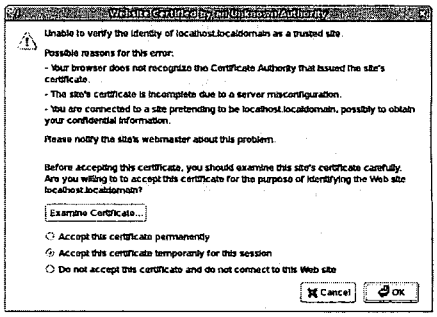


图 D-13 “Website Certified by an Unknown Authority（由未知授权认证的网站）”窗口

（2）在“User Name（用户名）”和“Password（密码）”中分别输入 `system` 和 `acme`，并单击“Login（登录）”（图 D-14）。

www.TopSage.com

（3）**Oracle 企业管理器数据库主页**出现，确认 Oracle 11g 安装成功（图 D-15）。

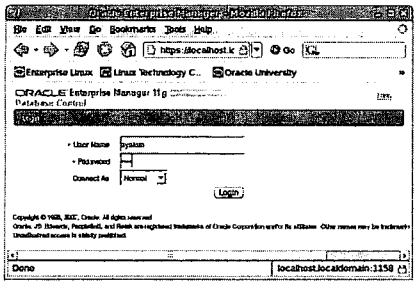


图 D-14 Oracle 11g 的“Enterprise Manager Login（企业管理器登录）”窗口

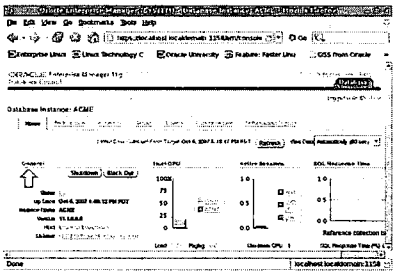


图 D-15 Oracle 11g“企业管理器数据库主页”窗口

5. 启动和停止 Oracle 11g

Oracle 11g 可以占用大量系统资源，因此如果只是出于教育目的使用它，那么我们建议在不使用它时停止它。

要启动和停止 Oracle，必须作为用户 `oracle` 或作为 dba 组的另一个成员登录。另外，要确保正确设置了表 D-2 中所示的会话环境变量。

停止 Oracle 11g

（1）在 Linux 提示符下，输入 `sqlplus/as sysdba`。

（2）输入 `shutdown immediate` 并按 Enter 键。

（3）输入 `quit` 退回到 Linux 提示符。

（4）在 Linux 提示符下，输入 `lsnrctl stop`，按 Enter 键。

（5）在 Linux 提示符下，输入 `emctl stop dbconsole`，停止 Oracle 企业管理器。

启动 Oracle 11g

（1）在 Linux 提示符下，输入 `emctl start dbconsole`，启动 Oracle 企业管理器（如果打算使用 OEM，才需要这样做；否则可以跳过此步骤）。

（2）在 Linux 提示符下，输入 `lsnrctl start` 并按 Enter 键。

（3）在 Linux 提示符下，输入 `sqlplus /as sysdba`。

（4）输入 `startup` 启动 Oracle 11g 实例，并按 Enter 键。

（5）输入 `quit` 退回到 Linux 提示符。

Tip 提示：可以使用 Oracle 企业管理器启动和停止 Oracle 数据库实例和服务；请参考 Oracle 11g 文档了解详细内容。

6. 卸载 Oracle 11g

在某些情况下可能要卸载 Oracle。为此，按照下面指导进行（作为 OS 用户 `oracle`）：

（1）停止在这个 Oracle 主机上运行的任何过程：

```
# emctl stop dbconsole
# lsnrctl stop
```

（2）启动 Oracle Universal Installer：

```
# $ORACLE_HOME/oui/bin/runInstaller
```

（3）“**Oracle Universal Installer：Welcome**”窗口出现，单击“Deinstall Products（卸载产品）”（图 D-16）。“Inventory（目录）”窗口弹出。

（4）选择 Oracle 主目录和想要卸载的产品并单击“Remove（删除）”按钮（图 D-17）。

（5）Oracle Universal Installer 显示一个确认窗口，询问确认要卸载产品及其相关组件。选择“Yes（是）”（图 D-18）。一条警告消息询问在卸载完成之后是否要删除 ORACLE_HOME 目录。单击“Yes（是）”。

（6）随着 Oracle Universal Installer 卸载软件，它会显示一个进度指示条。当卸载完成时，单击“Inventory（目录）”窗口的“Close（关闭）”，选择“Cancel（取消）”从 Oracle Universal Installer 退出，并单击“Yes（是）”确认（图 D-19）。

www.TopSage.com

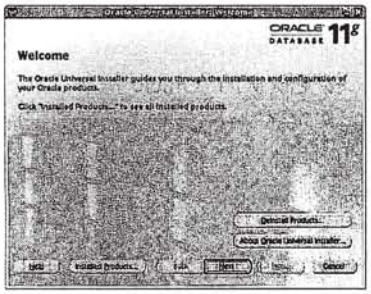


图 D-16 Oracle 11g 的“Universal Installer：Welcome”窗口

图 D-17 删除 Oracle 11g：“Inventory（目录）”窗口

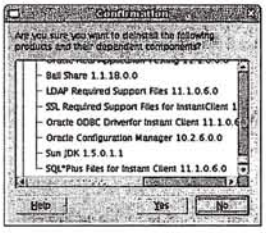


图 D-18 删除 Oracle 11g：“Confirmation（确认）”窗口

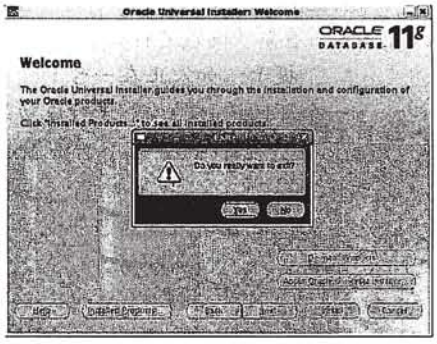


图 D-19 从 Oracle Universal Installer 退出

（7）手动删除/u01 下的内容；并删除/etc/oratab、/usr/local/bin/coraenv、/usr/local/bin/dbhome 和/usr/local/bin/oraenv 中的文件。

D.1.2 在 Windows 上安装 Oracle 11g release 1（11.1.0.6.0）个人版

下面部分提供在 Windows Vista Enterprise 上安装 Oracle 11g 个人版的安装指导。参考表 D-3 了解其他支持的 Microsoft Windows 需求。

多数安装画面看起来与在 Linux 安装 Oracle（前一节）的那些画面相同或者非常相似，因此可以参考那些画面。

表 D-3 列出在 Microsoft Windows 上安装 Oracle 11g 的最低要求。

表 D-3 Microsoft Windows 上的最低安装要求

操作系统	Windows 2000 SP1 Windows Server 2003 Windows Server 2003 R2 Windows XP Professional Windows Vista Business Windows Vista Enterprise Windows Vista Ultimate
内存	RAM：1024MB 虚拟内存：2RAM
硬盘	4.55GB（基本安装）
Web 浏览器	Netscape Navigator 7.2 或更高 Mozilla 1.7 或更高 Firefox 1.0.4 或更高 Microsoft Internet Explorer 6.0 SP2 或更高
其他	处理器：Intel（x86）、AMD64 和 Intel EM64T 550MHz（至少）

1. 安装指导

（1）从 Oracle 网站 http://www.oracle.com/technology/software/products/database/index.html 下载和保存 Oracle 11g release 1（11.1.0.6.0）软件（win32_11gR1_database.zip），

www.TopSage.com

并保存在指定位置，例如，保存在C:\Oracle11g目录下。

（2）将文件解压缩到C:\Oracle11g\目录。

（3）双击C:\Oracle11g\win32_11gR1_database\dabase\setup.exe，启动Oracle软件安装过程。第一个画面提示选择将Oralce服务器安装在哪里。接受默认的“Oracle基本目录”和“Oracle主目录”；“安装类型”选择个人版。选择“创建启动数据库”选项（默认）。在“全局数据库名”中输入ACME，在“数据库密码”和“确认密码”中输入acme，并单击“Next（下一步）”按钮。

Tip 提示：在Oracle 11g中，密码是区分大小写的。

（4）下一步是验证系统满足全部安装需求。滚动每一个“检查”项并确保状态指示为“成功”。在页面底部的灰色可滚动的文本框中显示详细信息（为获得更好的可读性，可以复制、粘贴并把它保存为一个文本文档）。如果有任何先决条件项状态不是“通过”，就必须在继续安装之前予以纠正。单击“取消”按钮停止Oracle 11g安装，在解决问题之后从第三步重新开始。当检查成功完成时，单击“Next（下一步）”按钮。

Tip 提示：如果正在一台DHCP计算机或一台未联网的计算机上安装Oracle，必须安装一个环回适配器才能进行成功的Oracle 11g安装。请参考Windows文档或登录http://download.oracle.com/docs/cd/B19306_01/install.102/b14316/reqs.htm#BABGHEFB了解详细指导。

（5）跳过**Oracle配置管理器注册**窗口（除非有一个有效的Metalink账户并想注册自己的产品）。单击“Next（下一步）”按钮即可。

（6）查看**摘要**窗口，并单击“安装”，等待安装过程完成。

（7）**配置助手**窗口出现。默认安装配置Oracle网络服务和代理，创建示例数据库，并启动Oracle HTTP服务器。

（8）当数据库创建完成时，**数据库配置助手**窗口中可以选择解锁用户或修改系统用户的密码。考虑本书的目的，我们建议跳过此步骤；只需单击“确定”按钮。

（9）查看**安装结束**窗口信息，并单击“退出”。单击“是”确认。

2. 测试Oracle 11g安装

（1）打开浏览器并输入下面的URL：https://localhost:1158/em（可以使用自己的机器名或IP地址替换“localhost”）。

（2）输入用户名system和密码acme，并选择“Login（登录）”。

（3）**Oracle企业管理器数据库主页**出现，确认Oracle 11g安装成功（图D-15）。

3. 启Oracle 11g

Oracle 11g可以占用大量系统资源，因此如果只是出于教育目的而使用它，那么建议在不使用它时停止它。

停止Oracle 11g实例

（1）打开一个Windows命令行窗口并输入sqlplus/a sysdba。

（2）输入shutdown immediate，并按Enter键。

（3）输入quit退出。

启动Oracle 11g实例

（1）打开一个Windows命令行窗口并输入sqlplus/a sysdba。

（2）输入startup启动Oracle 11g实例，并按Enter键。

（3）输入quit退出。

停止侦听器和数据库控制服务

使用Windows服务（开始→控制面板→管理工具→服务）停止和启动Oracle侦听器和Oracle数据库控制。参阅供应商文档了解详细内容。

Tip 提示：可以使用Oracle企业管理器启动和停止Oracle数据库实例和服务；请参考Oracle 11g文档了解详细内容。

www.TopSage.com

D.2 安装 IBM DB2 9.5

与 Oracle 不同，DB2 需要更少的系统资源。安装过程快速且简单；它不需要过多的用户干预，并且简单直观。

在 Windows 上安装 DB2 9.5 软件

下面的内容提供了在 Windows Vista Enterprise 上一步步安装 DB2 9.5 Express 版的指导。参见表 D-4 列出的其他支持的 Microsoft Windows 版本。

1. 系统需求

表 D-4 列出了在 Windows 上安装 DB2 9.5 的要求。

2. 预安装任务

必须有一个本地定义的用户账户，并且是本地管理员组的成员。

表 D-4　　在 Windows 上安装 DB2 的平台要求

处理器（CPU）	能够运行所支持的 Windows 操作系统的全部 Intel 和 AMD 处理器
操作系统	Windows 2003 标准版 Windows 2003 企业版 Windows 2003 数据中心版 Windows XP Professional Windows Vista Ultimate Windows Vista Business Windows Vista Enterprise
服务包	Windows XP Service Pack2 或.NET 客户端应用程序的 IBM Database Server Provider，以及需要.NET 1.1 SP1 或.NET 2.0 Framework 运行时的 CLR 服务器端过程
内存	512MB 或更多
硬盘	1GB
Web 浏览器	Internet Explorer 6 或以上 Mozilla 1.4 或以上 Firefox 1.0 或以上 Netscape 7.0 或以上
其他	Java SDK 1.4.2 到 SDK 5 支持的 CD-ROM 驱动器

Tip　提示：在开始安装前，建议关闭所有其他的程序。

3. 安装指导

按照下面这些步骤进行：

（1）如果没有安装媒体，可以从 IBM 网站下载和保存 DB2 9.5 安装软件（db2_v95_nt32_server.zip）：http://www-306.ibm.com/software/data/db2/9/download.html（必须首先注册和接受 IBM 条款）。

（2）将文件解压缩到选择的文件夹中，例如，可以使用 C:\DB295。

（3）双击 SERVER 目录中的 setup.exe，启动 DB2 9.5 软件安装过程；即，假设将安装文件解压缩到 C:\DB295 目录，则 setup.exe 的完整路径是“C:\DB295\SERVER\setup.exe”。

（4）**DB2 安装启动栏**屏幕打开，如图 D-20 所示。单击“安装一项产品”。

（5）**选择想要安装的产品**屏幕将出现。向下滚动，选择“DB2 Express Version 9.5 Edition”并单击“Next（下一步）”按钮。

Tip　提示：可以不选择本节中介绍的 DB2 Express Edition，而是安装 Enterprise Edition、Workgroup Server Edition、Personal Edition 以及诸如 Connect Server 和 DB2 Client 等一些其他产品。如果想安装 DB2 9.5 的 Express-C 版本，那么需要下载的文件是 db2exc_950_WIN_x86.zip。对于所有的 DB2 9.5 版本而言，安装指导都极为相同。

www.TopSage.com

（6）**准备安装窗口出现：稍后 DB2 安装向导将自动启动**（图 D-21）。单击“Next（下一步）”按钮。

图 D-20　DB2 9.5 安装启动栏窗口

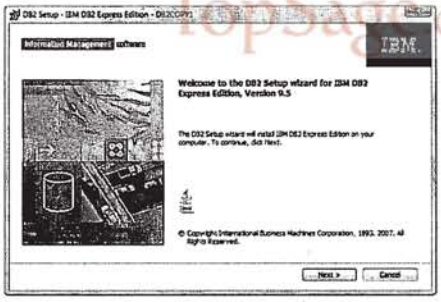


图 D-21　DB2 9.5 安装向导屏幕

（7）在**许可协议**屏幕上选择“Iaccept the terms in the license agreement（我接受许可协议中的条款）”选项，并单击“Next（下一步）”按钮。

（8）在**选择安装类型**屏幕上选择“Typical（典型）”安装类型。确保硬盘（分区）上有足够的空间。参见图 D-22。

（9）单击“Next（下一步）”按钮，接受**选择安装、响应文件或两者**屏幕上的默认设置（“在这台计算机上安装 IBM DB2 Express Edition 并在响应文件中保存我的设置”）。

（10）单击“Next（下一步）”按钮，接受**选择安装文件夹**屏幕上的默认目录（图 D-23）。

图 D-22　DB2 9.5“Select the installation type（选择安装类型）”屏幕

图 D-23　DB2 9.5 的“Select the installation folder（选择安装文件夹）”屏幕

（11）在下一个屏幕上选择“本地系统帐户”，确保选中“对其余的 DB2 服务使用相同的用户名和密码”复选框，并单击“Next（下一步）”按钮（图 D-24）。

（12）在**配置 DB2 实例**屏幕上单击“Next（下一步）”按钮。

（13）查看**开始复制文件和创建响应文件**屏幕上的设置（图 D-25）并单击“Finish（完成）”按钮。安装开始。

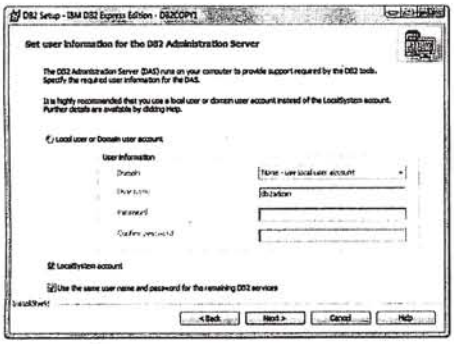


图 D-24　DB2 9.5 的“Set user information for the DB2 Administration Server（为 DB2 管理服务器设置用户信息）”窗口

图 D-25　DB2 9.5“Start copying files and create response file（开始复制文件和创建响应文件）”窗口

www.TopSage.com

（14）在 DB2 9.5 安装完成时，将看到**安装完成**窗口（图 D-26）。滚到到必要的步骤（如果愿意，可以复制、粘贴和保存这些信息），单击“Next（下一步）”按钮。

（15）忽略**安装其他产品**屏幕；只需单击“完成”（图 D-27）。

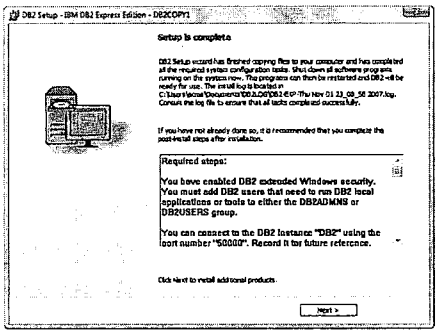


图 D-26 DB2 9.5 的“Setup is complete（安装完成）”窗口

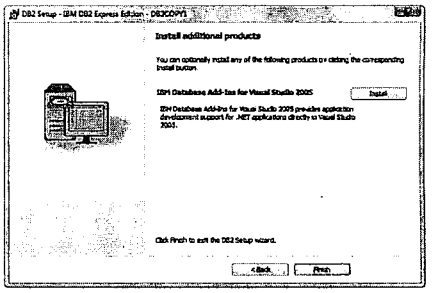


图 D-27 DB2 9.5 的“Install additional products（安装其他产品）”窗口

4. 安装后的任务

需要修改账户的安全设置以便能够连接到 DB2。默认情况下没有授予执行 DB2 9.5 安装的用户的连接特权。必须使这个用户（或选择的任何其他 OS 用户）成为 DB2ADMINS 和/或 DB2USERS 组的成员。我们也将指导如何创建一个空 ACME 数据库，稍后会在附录 F 中使用这些指导填充该数据库，并将它用于本书中的所有例子。

修改安全设置

（1）打开“控制面板”→“管理工具”→“计算机管理”并选择“本地用户和组”选项。

（2）双击要使用 DB2 9.5 的用户（示例安装中是用户“acme”）。转到**成员**选项卡，单击“添加”按钮，打开**选择组**窗口。单击“立即查找”按钮，滚动搜索结果，并突出显示 DB2ADMINS 和 DB2USERS 组（图 D-28）。单击“确定”按钮返回到用户属性窗口；再次单击“确定”按钮接受修改。

（3）重新启动 Windows。

创建空的 ACME 数据库

（1）打开 DB2 控制中心（“开始”→“所有程序”→IBM DB2→DB2COPY1→“一般管理工具”→“控制中心”）。

（2）接受**控制中心视图**上的高级选项并单击“OK（确定）”按钮。

（3）展开树型视图直到“数据库”。右键单击“数据库”文件夹并选择“创建数据库”→“标准”（图 D-29）。**创建数据库向导**窗口出现。

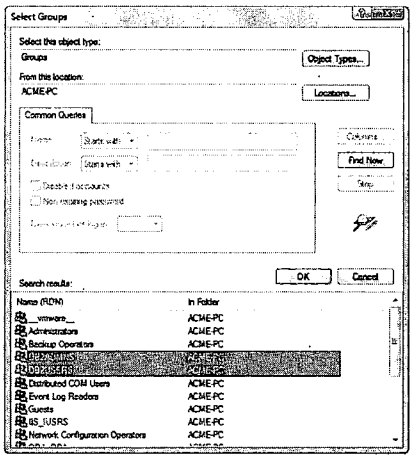


图 D-28 “Select Groups（选择组）”窗口

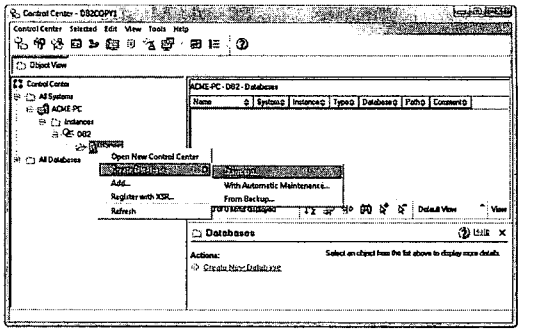


图 D-29 “控制中心”窗口

www.TopSage.com

（4）在“数据库库名”字段中输入 ACME 并单击“完成”按钮（图 D-30）。“进度”窗口弹出，表明所用的时间。此外，还可以不断单击“Next（下一步）”按钮，查看默认设置，然后在最后的屏幕上单击“完成”，不过完全可以不必这样做，因为我们没有修改任何默认设置。

（5）在完成之后，弹出的 **DB2 消息**窗口表明已经成功创建了数据库。单击“关闭”按钮。

（6）在“数据库”文件夹下面会看到 ACME 的图标。现在安装后的任务完成。

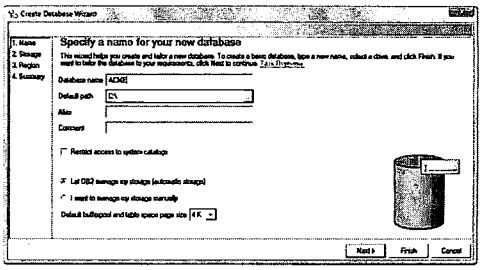


图 D-30 在“创建数据库向导”中创建 ACME 数据库

Note 注意：创建空的 ACME 数据库是一个可选步骤；当运行附录 F 中介绍的数据库创建脚本时将重新创建这个数据库。

5. 卸载 DB2 9.5

使用下面的步骤卸载 DB2 9.5：

（1）删除 ACME 数据库（以及已经创建的任何其他数据库）。打开控制中心（“开始”→“所有程序”→IBM DB2→DB2COPY1→“一般管理工具”→“控制中心”），展开树形视图直到“数据库”文件夹（“实例”→DB2→“数据库”），选择 ACME，并从右键单击菜单（或从 Selected 菜单）中选择“删除”。“确认”窗口出现。保留“删除”复选框的选中状态并单击“确定”。对已经创建的任何其他数据库重复这个过程。

（2）关闭控制中心窗口和任何其他可能已经打开的与 DB2 有关的程序。

（3）转到“控制面板”→“程序和功能”。

（4）找到 IBM DB2 Express Edition 并单击“卸载”按钮。在“确认”窗口中单击“是”。

（5）可能出现**提示**窗口询问是否要终止运行 DB2 进程。回答“是”继续。

（6）可能仍然需要手动删除一些目录（假设 `C:\Program Files\IBM\SQLLIB` 是默认目录）。

6. 启动和停止 DB2 9.5

如果接受了安装默认设置，则 DB2 将在启动时自动启动；否则，打开控制中心（“开始”→“所有程序”→IBM DB2→DB2COPY1→“一般管理工具”→“控制中心”），展开树形视图直到 DB2 文件夹，右键单击并选择“启动”。要停止实例，重复同样的步骤，不过要选择“停止”选项。

要停止其他与 DB2 有关的服务，则使用 Windows 服务（“开始”→“控制面板”→“管理工具”→“服务”）。查找以“DB2”开头的服务名。参考供应商文档了解详细信息。

Tip 提示：在 Windows 通知区右键单击 DB2 图标（红色或绿色圆柱形）也可以停止数据库管理器实例。

D.3 安装 Microsoft SQL Server 2008

接下来是在 Windows Vista 上安装 Microsoft SQL Server 2008 的指导。参见表 D-5 了解其他支持的平台。

Note 注意：在编写本书时，Microsoft SQL Server 2008 只有 CTP（Community Technology Preview，社区技术预览）版本可以使用。Microsoft 特别指出，最终的软件发行版可能采用不同的方式运行并包含一些不同的功能。

D.3.1 安装 Microsoft SQL Server 2008（CTP，2007 年 11 月发布）的必备条件

SQL Server 本身就是针对 Windows 的，因此对在 Windows 上运行进行了最大优化。在着手安装 Microsoft SQL Server 2008 之前，要检查系统以满足软件和硬件需求。

最低系统要求

表 D-5 列出了针对 Microsoft SQL Server 2008 CTP 的最低系统要求。

www.TopSage.com

TopSage.com

表 D-5	Windows 上的 Microsoft SQL Server2008 安装要求
处理器（CPU）	Intel Pentium III 600MHz 或更高（推荐 1GHz）
操作系统	Windows Server 2003 SP1 Windows Server 2003 Enterprise Edition SP1 Windows XP Professional SP2 Windows Vista Business Windows Vista Enterprise Windows Server 2003 64 位 x64 SP1
内存（RAM）	512MB（推荐 1GB）
硬盘	最小 280MB；依据安装的组件可达到 2GB
Web 浏览器	Internet Explorer 6 SP1 或以上（通常包含所有 Windows 版本）

D.3.2 在 Windows Vista Enterprise Edition 上安装

所有支持的 Windows 风格的安装指导基本相同。

Note 注意：我们作为具有管理员特权的 OS 用户 ACME 执行安装。开始在计算机上安装 SQL Server 2008 之前，必须卸载所有以前版本的 Microsoft SQL Server。

1. 安装指导

执行下面的步骤，在机器上安装 Microsoft SQL Server 2008：

（1）从 Microsoft 网站 `http://www.microsoft.com/sql/2008/prodinfo/download.mspx` 下载 Microsoft SQL Server 2008 针对 X86 的试用软件的自解压版本（首先必须注册和接受 Microsoft 条款）。

Note 注意：到本书出版时，Microsoft SQL Server 2008 的最终商业版本可能会发布；我们会随时更新本书的网站。

（2）下载完成后，在 Windows 资源管理器中双击文件名，选择一个目标目录，单击“确定”按钮，然后等待程序解压缩文件。

（3）双击 servers 目录中的 setup.exe，启动 Microsoft SQL Server 2008 软件安装过程；即，如果将安装解压缩到 C:\SQL SERVER 2008 目录下，那么 `setup.exe` 文件的完整路径就是：`C:\SQL SERVER 2008\servers\setup.exe`。

（4）选择“I accept the license terms（我接受许可条款）”复选框并单击“Next（下一步）”按钮（图 D-31）。

（5）下一个屏幕说明在安装 SQL Server 2008 之前有关软件安装必备条件的信息。单击“Install（安装）”继续（图 D-32）。

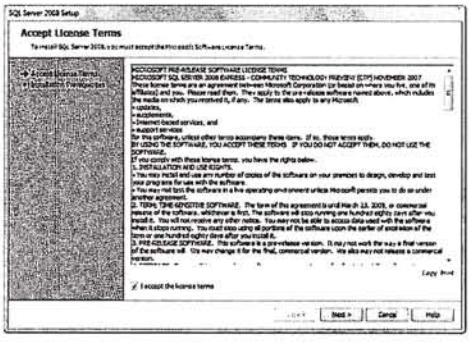


图 D-31 Microsoft SQL Server 2008 “Accept License Terms（接受许可条款）”窗口

图 D-32 Microsoft SQL Server 2008 “Installation Prerequisites（安装必看条件）”窗口

（6）成功安装必备组件之后，单击“Next（下一步）”按钮。

（7）安装程序进行系统配置检查，然后显示 **SQL Server 安装中心**窗口（图 D-33）。选择“New Installation（新建安装）”。

（8）**系统配置检查**窗口打开。等到看到“Execution completed（执行完成）”消息并确保每项规则的状态均为“通过”后，单击“Next（下一步）”按钮。

www.TopSage.com

（9）在下一个屏幕上要选择安装哪个 SQL Server 组件。在示例安装中我们选择“数据库引擎服务”和“客户端工具”（图 D-34）；也可以选择“SQL Server 联机丛书”来安装 SQL Server 文档。单击“Next（下一步）”按钮。

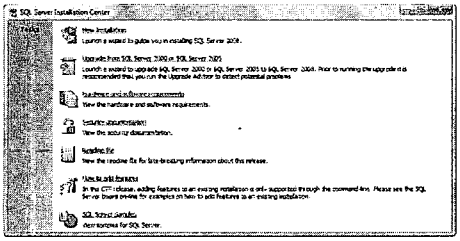


图 D-33　SQL Server 2008“安装中心”窗口

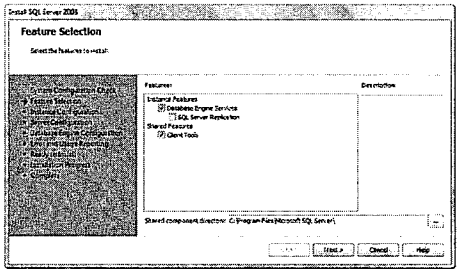


图 D-34　Microsoft SQL Server 2008 “Feature Selection（功能选择）”窗口

（10）在下一个屏幕上选择安装一个默认实例还是创建一个命名实例。我们想要创建一个名为 ACME 的实例，因此选择“命名实例”，在给出的空白中输入 ACME，接受默认的“实例根目录”并单击“Next（下一步）”按钮。

（11）下一个屏幕提示选择服务账户。选择“针对所有服务使用相同的账户”选项，在提供的字段中输入一个有效的 OS 用户名和密码（用户应当具有管理员特权），单击“应用到全部”按钮。单击“Next（下一步）”按钮。

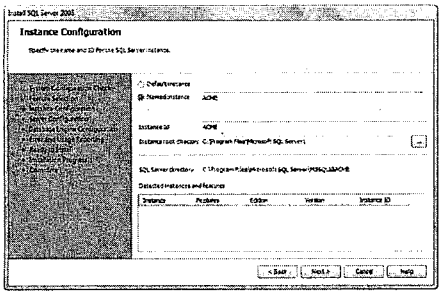


图 D-35　Microsoft SQL Server 2008“Instance Configuration（实例配置）”窗口

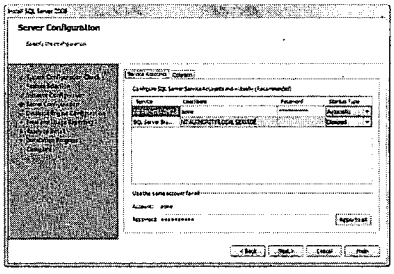


图 D-36　Microsoft SQL Server 2008“Server Configuration（服务器配置）”窗口

（12）在下一个屏幕上选择“Windows 身份验收模式”，单击“添加当前用户”按钮，然后单击“Next（下一步）”按钮。

（13）如果不想让系统自动向 Microsoft 发送消息，则取消选择**错误和使用报告**屏幕上的两个复选框。单击“Next（下一步）”按钮。

（14）**准备安装**窗口显示要安装的组件（图 D-38）。单击“Install（安装）”。

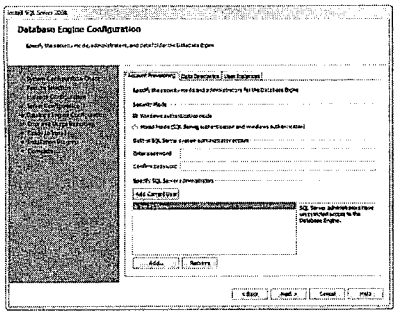


图 D-37　Microsoft SQL Server 2008“Database Engine Configuration（数据库引擎配置）”窗口

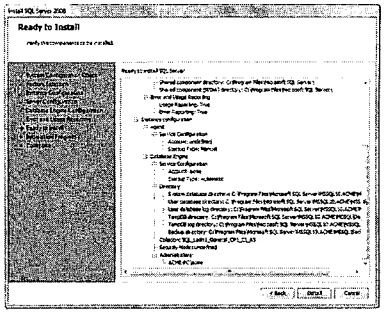


图 D-38　Microsoft SQL Server 2008“Ready to Install（准备安装）”窗口

www.TopSage.com

（15）安装过程开始。**安装进度**窗口中显示组件安装状态（图 D-39）。当所有元素安装完成时，确保列表中每个功能的“状态”均为“成功”，单击“Next（下一步）”按钮。

（16）查看**完成**屏幕上的信息，并单击“Close（关闭）”按钮。现在安装结束。

现在可以连接它并运行随本书提供的 SQL 脚本来安装 ACME 数据库，并使用数据填充它；请参考附录 E 和附录 F。

2. 安装后的任务

为了能够连接到 Microsoft SQL Server 2008，可能需要修改帐户的安全设置。必须使执行安装的用户（或任何其他选择的 OS 用户）成为 SQL Server 组的成员。

（1）打开“控制面板”→“管理工具”→“计算机管理”并选择“本地用户和组”（图 D-40）。

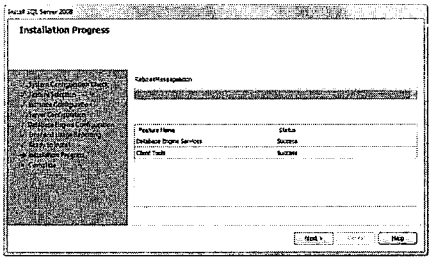


图 D-39 Microsoft SQL Server 2008“Installation Progress（安装进度）”窗口

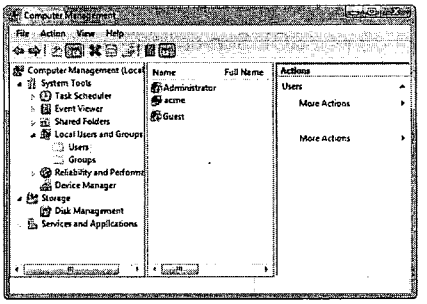


图 D-40 Microsoft Windows Vista“Computer Management（计算机管理）”窗口

（2）双击要使用 Microsoft SQL Server 2008 的用户，示例安装使用用户“acme”）。转到**成员**选项卡，单击“Add（添加）”按钮，打开**选择组**窗口。单击“Find Now（立即查找）”按钮，滚动搜索结果，并突出显示以“SQLServer”开头的所有组（图 D-41）。单击“OK（确定）”按钮返回到用户属性窗口；再次单击“OK（确定）”按钮接受修改。

3. 启动和停止 Microsoft SQL Server 2008

Microsoft SQL Server 2008 可以占用显著的系统资源，因此如果只是出于教育目的而在常用的机器上安装它，那么实际打算使用它时再启动可能是一个好主意。

停止 Microsoft SQL Server 2008

（1）打开 SQL Server 配置管理器（“开始”→“所有程序”→“Microsoft SQL Server 2008”→“配置工具”→“SQL Server 配置管理器”）。

（2）选择“SQL Server 服务”。

（3）对于处在“运行”状态的每项服务，选择“服务名”并单击应用程序工具栏上的红色“Stop Service（停止服务）”按钮（图 D-42）。

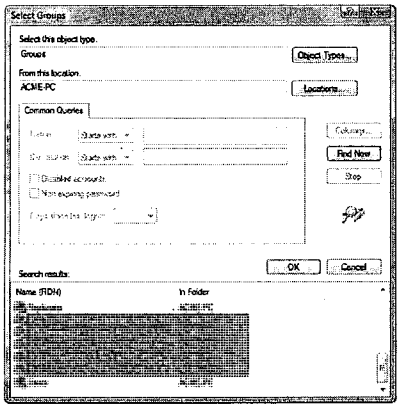


图 D-41 Microsoft Windows Vista“选择组”窗口

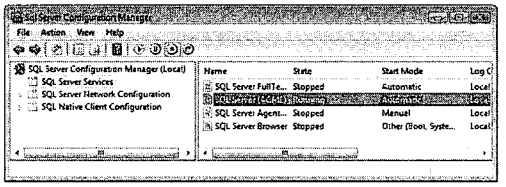


图 D-42 Microsoft SQL Server 2008“配置管理器”

www.TopSage.com

大家网 TopSage.com

启动 Microsoft SQL Server 2008

（1）打开 SQL Server 配置管理器（“开始”→“所有程序”→Microsoft SQL Server 2008→“配置工具”→“SQL Server 配置管理器”）。

（2）选择“SQL Server 服务”。

（3）对于处在“停止”状态的每项服务，选择“服务名”并单击应用程序工具栏上的绿色“Start Service（启动服务）”按钮[实际上，出于本书的目的，除了启动 SQL Server（ACME）服务之外，并不需要其他服务]。

禁止 SQL Server 服务自动启动

（1）打开 SQL Server 配置管理器（“开始”→“所有程序”→Microsoft SQL Server 2008→“配置工具”→“SQL Server 配置管理器”）。

（2）选择“SQL Server 服务”。

（3）对于“启动模式”不是“手动”的每项服务，选择“服务名”，单击应用程序工具栏上的“Properties（属性）”按钮或者只双击“服务名”。选择**服务**选项卡，将“启动模式”改为“手动”，单击“OK（确定）”按钮。

4. 卸载 Microsoft SQL Server 2008

要卸载 Microsoft SQL Server 2008，执行下面的步骤：

（1）选择“开始”→“控制面板”并选择“程序和功能”选项。

（2）选择“Microsoft SQL Server 2008”并单击“Uninstall（卸载）”按钮。

（3）选择“删除所选实例和共享组件”选项（图 D-43）。从下拉列表中选择要删除的实例（对于本示例安装是 ACME）。单击“Next（下一步）”按钮。

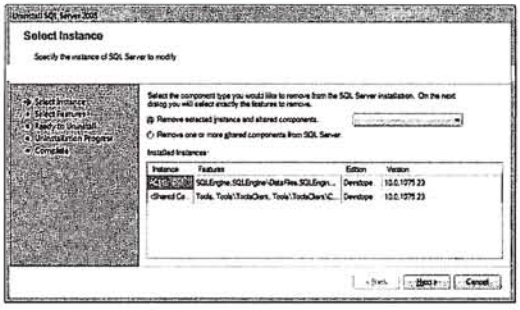


图 D-43　卸载 SQL Server 2008“Select Instance（选择实例）”窗口

（4）取消选择“Select Features（选择功能）”窗口上的全部复选框并单击“Next（下一步）”按钮。

（5）在下一个屏幕（**准备卸载**）上单击“Uninstall（卸载）”按钮，等待 Microsoft SQL Server 2008 安装程序完成卸载。

www.TopSage.com

附录 E
访问 RDBMS

APPENDIX

本附录提要：

- 访问 Oracle 11g
- 访问 DB2 9.5
- 访问 Microsoft SQL Server 2008

不管相信与否，在当今计算机上如此流行的图形用户界面（Graphical User Interface，GUI）并不是一直就有的，它是伴随阴极射线管（CRT）显示器的使用而发展起来的，并随着指点设备（最熟知的就是鼠标）的出现，产生了真正的飞跃。因此，每个能够交互式访问的数据库除了 GUI 工具外，还具有某种命令行界面，允许用户提交一个请求（通常是一个 SQL 查询），并最终接收某些响应——可以是被请求的数据或来自 RDBMS 的一条错误消息。

E.1 使用 ORACLE 11g 实用工具访问 RDBMS

Oracle 11g 为与 RDBMS 进行通信提供了多种实用工具，如 SQL*Plus、SQL Developer、Oracle 企业管理器和 Application Express。Oracle 9i 首次引入的另一个工具 iSQL*Plus 不再是 Oracle 11g 的组成部分，并且计划到 2010 年不再受支持。

还有许多第三方产品都可以与 Oracle 11g 数据库一起使用，最流行的是 Quest Software Ltd.（`www.quest.com`）的 TOAD 和 Synametrics（`www.synametrics.com`）的 WinSQL。

E.1.1 SQL*Plus

SQL*Plus 是一个交互式查询工具，它提供了一种机制，可以将 SQL 命令提交到 Oracle RDBMS 来进行执行、检索结果、执行数据库管理等更多操作。本书中的所有 Oracle 例子都是利用 Oracle SQL*Plus 界面生成的，本附录提供了访问 Oracle 数据库和运行交互式命令或相应脚本文件的基础知识。SQL*PIus 也具有平台无关的优点——完全相同的命令可以用于 Windows、Sun Solaris、Linux 或 VAX VMS。

启动 SQL*Plus 应用程序有两个选择：从 MS-DOS（或 UNIX、Linux 等）命令提示符或（假设正在使用 Microsoft Windows）从“开始”菜单（“开始”→“程序”→Oracle→“应用程序开发”→SQL*Plus）。本节说明如何在本地计算机上使用 SQL*Plus，假定已按照附录 D 中的 Oracle 11g 安装指导进行了安装。

在命令行提示符下，输入 `sqlplus/nolog` 并按 Enter 键，屏幕应如图 E-1 所示。

www.TopSage.com

现在需要连接到数据库。CONNECT命令的语法如下：

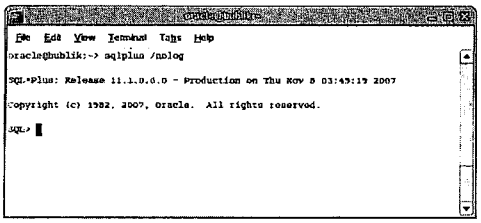
```
Connect <username>/<password>[@<database_sid>]
```

例如，如果遵循了附录D中的安装指导，那么可以使用用户SYSTEM和密码ACME连接：

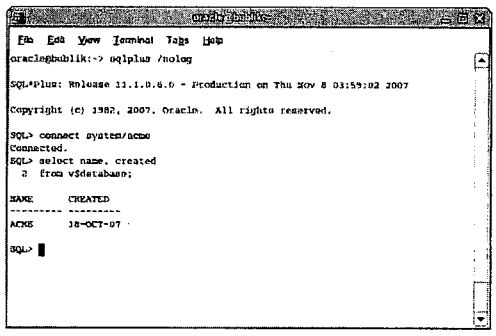
```
SQL> connect system/acme
```

Tip 提示：从SQL*Plus 11g版开始，密码就区分大小写了，因此，如果在安装过程中输入的数据库密码是ACME或Acme而不是acme，则使用这个密码，即ACME；否则将看到“无效的用户名/密码”消息。

接下来将得到SQL提示符，然后就可以输入和执行SQL语句了。要将SQL语句发送到Oracle，需要在查询的末尾添加一个分号并按Enter键。如果没有包括分号就按了Enter键，则SQL*Plus将继续一个新行；每一行均以一个序列号作为前缀。下面的例子检索有关当前数据库的一些信息（图E-2）。



图E-1 从Linux启动的SQL*Plus命令行界面



图E-2 从SQL*Plus执行SQL命令

Tip 提示：可以用一个斜杠（/）代替分号，但这主要用在PL/SQL和批处理中。

结果将显示在同一个窗口中。可以设置一些参数来自定义SQL*Plus界面的外观。请参考SQL*Plus手册获得更详细的信息。

要载入一个脚本文件，可以使用语法@["][<path>]<file_name>["]；例如：

```
SQL> @"C:\sqlbib\oracle_acme\load.sql"
```

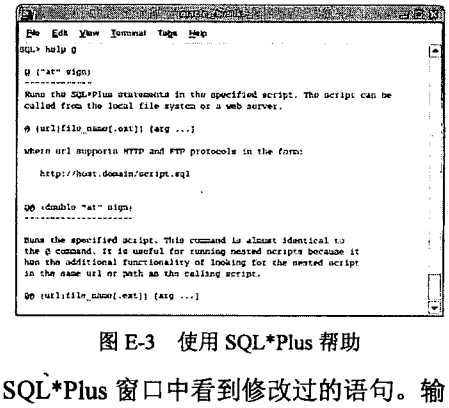
或

```
SQL> @load.sql
```

第一个例子假定文件load.sql不在当前目录中，因此必须指定这个文件的完整路径。双引号是可选的，并且只有当路径中的目录名包含空格时才需要。第二个例子从当前目录中载入文件。

Oracle并不假设必须立即运行一个文件中的脚本，因此，如果脚本没有以一个分号结束，它就不会运行，直到添加了分号并按Enter键才运行。

Tip 提示：要从SQL*Plus内部访问帮助，可以输入help <topic>（已安装了所提供的帮助）。帮助提供了有关指定主题的信息并显示了Oracle网站的URL，通过网站可以研究SQL*Plus资源并阅读其他Oracle文档（图E-3）。



图E-3 使用SQL*Plus帮助

可以使用SQL*Plus命令行或操作系统的默认编辑器（通常对于Windows OS来说是记事本，在Linux/Unix上是vi）来编辑SQL语句。要从SQL*Plus编辑一个查询，可以在SQL*Plus提示符下输入edit或ed。这将调用默认的编辑器打开最后的SQL语句。编辑、保存后退出编辑器后，将在SQL*Plus窗口中看到修改过的语句。输入一个分号或斜杠来执行语句。

Note 注意：修改过的文本将存储在缓冲区文件afiedt.buf中，这个文件位于启动SQL*Plus会话时所在那个目录。

可以使用50多条SQL*Plus命令；如果需要了解有关这个界面的更多信息，请参考Oracle文档。

www.TopSage.com

为了轻松使用 SQL*Plus，表 E-1 中只包括了可能需要且最常用的 SQL*Plus 命令。

Note 注意：不应当把表 E-1 中的命令看作是 SQL 的一部分；Oracle 称之为 SQL*Plus 命令。

Note 注意：SQL*Plus 命令都是区分大小写的。

表 E-1 最常用的 SQL*Plus 命令

SQL*PLUS 命令	说明	例子
@	运行指定脚本中的 SQL 语句。可以从本地文件系统或某个 web 服务器调用这个脚本	SQL> @load_data_ora.dat
@@	类似于@；常用于从一个脚本（嵌套脚本）调用另一个脚本	@@load_data_ora.dat
/（斜杠）	执行 SQL 缓冲区中的内容	SQL>/<query results>
CLEAR SCREEN	清除监视器屏幕	SQL>clear screen
CONN[ECT]	连接指定的 Oracle 用户	SQL>connect acme/acme
DEF[INE]	指定用户变量	SQL>define cust_id=152
DESC[RIBE]	列出指定对象（表、视图等）的列定义	SQL>desc address
DISC[CONNECT]	在提交未完成的修改后断开当前用户的连接	SQL>disc
ED[IT]	调用默认的 OS 编辑器	SQL>ed create_acme.sql
EXIT	退出 SQL*Plus。所有更改将被提交	SQL>exit
GET	将 OS 文件载入 SQL 缓冲区（但并不执行）	SQL>get create_acme.sql
HELP	访问 SQL*Plus 帮助	SQL>help get
HOST	在不离开 SQL*Plus 的情况下执行主机 OS 命令	SQL>host dir
PASSW[ORD]	允许修改用户密码	SQL>passw acme
QUIT	与 EXIT 相同	SQL>exit
R[UN]	列出并执行 SQL 缓冲区中的内容	SQL>run create_acme.sql
SAV[E]	将 SQL 缓冲区的内容保存到主 OS 文件中	SQL>save tmp.out
SPOOL	将查询结果存储在 OS 文件中	SQL>spool tmp.out SQL>select *from status; SQL>spool off
SET	设置当前会话的系统变量（自动提交、行和页大小等）	SQL>set autocommit on SQL>set linesize 1000 SQL>set pagesize 300 SQL>set pause on
START	与@相同	SQL>start create_acme.sql
UNDEFINE	删除用户变量（前面使用 DEFINE 命令创建的）	SQL>undefined cust_id
WHENEVER SQLERROR	如果 SQL 语句返回一条错误则停止执行脚本并退出 SQL*Plus（默认行为）	SQL>whenever sqlerror continue
WHENEVER OSERROR	如果发生操作系统错误（即，到数据库的连接丢失），则停止脚本的执行并退出 SQL*Plus（默认行为）	SQL>whenever oserror exit

E.1.2 SQL Developer

Oracle 已经将 SQL Developer 作为 SQL*Plus 的 GUI 版本推出，它是为了方便地执行基本 SQL 相关任务而设计的，默认情况下随基于模板的数据库安装或数据库客户端安装一起安装。

除了通过标准的 Oracle 身份验证连接到 Oracle 数据库，SQL Developer 也能够连接并使用选定的非 Oracle 数据库（Oracle 11.1.0.6 随附的 SQL Developer 1.1.3 列表包括 MySQL、Microsoft SQL Server 和 Microsoft Access）。

1. 启动 SQL Developer

要在 Linux（或 Unix）上启动 SQL Developer，可以打开$ORACLE_HOME/sqldeveloper 目录，并输入./sqldeveloper.sh（作为用户 oracle）：

```
#cd \$ORACLE_HOME/sqldeveloper
#./sqldeveloper.sh
```

要在 Microsoft Windows 上启动 SQL Developer，直接打开 SQL Developer（“开始”→“程序”→Oracle→“应用程序开发”→SQL Developer）。

www.TopSage.com

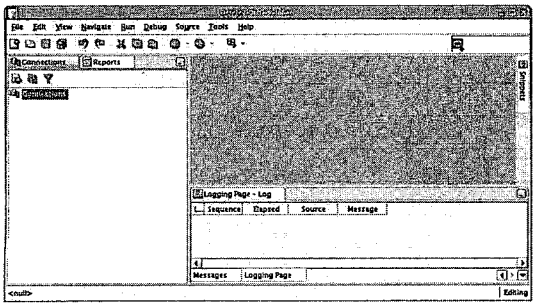
2. 创建一个数据库连接

要使用数据库对象，需要创建和保存一个数据库连接。下面的例子中为用户SYSTEM创建一个到ACME数据库的连接。稍后在执行附录F中的数据库脚本时，“安装ACME数据库”创建用户ACME，ACME是本书示例中使用的数据库，是示例中所有数据库对象的所有者。可以为用户ACME创建另一个数据库连接。创建步骤相同；只需用ACME替换前两个字段中的SYSTEM。

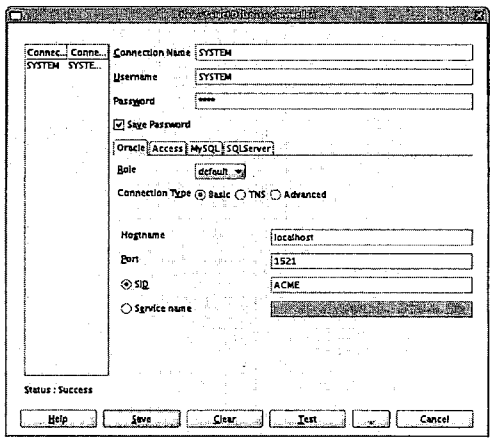
（1）在**SQL Developer**窗口的左上角（图E-4）右键单击“Connections（连接）”并选择“New Connection（新建连接）”。

（2）在“Connection Name（连接名）”和“User Name（用户名）”字段中输入SYSTEM；在“Password（密码）”和SID字段中输入acme。保留其他设置不变，单击“Test（测试）”按钮。

（3）窗口左下角显示“状态：成功”。单击“Save（保存）”按钮并关闭窗口（图E-5）。现在可以展开“连接”下面的新SYSTEM图标并使用数据库对象了（要查看属于其他用户的对象，可以展开“其他用户”文件夹）。



图E-4 Oracle SQL Developer窗口



图E-5 在Oracle SQL Developer中创建新连接

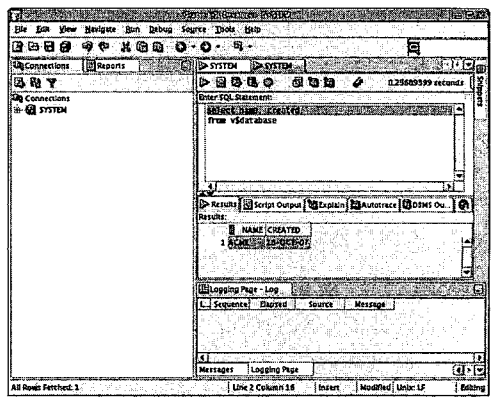
3. 使用SQL Worksheet

SQL Worksheet是SQL Developer的一部分，可以用于执行SQL、PL/SQL和SQL*Plus语句。

可以使用多种不同的方式启动SQL Worksheet；例如，右键单击一个连接并选择“打开SQL Worksheet”，从“工具”菜单中选择“SQL Worksheet”，或单击“使用SQL Worksheet”图标。后两种方式也需要选择正确的连接并单击“确定”按钮。

然后使可以输入和执行SQL语句了（图E-6）。

Oracle SQL Developer是一个非常强大的工具，可以出于很多不同的目的按照很多不同的方式使用它，这不在本书讨论范围内。参考这里的Oracle文档了解更多信息：http://download.oracle.com/docs/cd/B28359_01/appdev.111/b31695/intro.htm#i1013610。



图E-6 使用SQL Developer/SQL Worksheet执行SQL语句

E.1.3 Oracle企业管理器（SQL Worksheet）

Oracle企业管理器有一个称为SQL Worksheet的内置组件，它是Oracle 11g基于Web浏览器的界面。它能够执行的任务与通过SQL*Plus命令行版本执行的任务相同，不同之处是SQL Worksheet使用Web浏览器。

www.TopSage.com

Oracle 企业管理器是专为有经验的 Oracle 专家设计的高级工具，因此我们不打算在本书中深入介绍；不过，SQL Worksheet 非常有用，因为它是基于 Web 的并且不需要安装任何软件，我们将简要介绍它的使用方法。

（1）按附录 D“安装 RDBMS 软件”中的说明打开 **Oracle 企业管理器**。

（2）滚动页面并单击“相关链接”区域中的“SQL Worksheet”链接。

（3）SQL Worksheet 浏览器窗口弹出。现在可以在“SQL Commands（SQL 命令）”区域输入 SQL 并单击“Execute（执行）”按钮来执行 SQL 语句（图 E-7）。

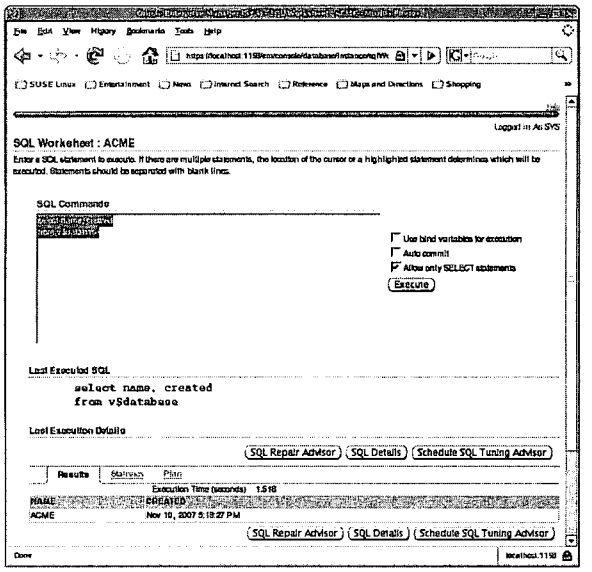


图 E-7 使用 Oracle 企业管理器/SQL Worksheet 执行 SQL 语句

Tip 提示：请注意，从 Oracle 企业管理器内可以使用的 SQL Worksheet 与我们在 SQL Developer 部分讨论的 SQL Worksheet 是完全不同的产品。

E.2 使用 IBM DB2 9.5 实用工具访问 RDBMS

DB 9.5 既提供了命令行实用工具（如 CLP），也提供了 GUI 工具（命令中心）来操作数据库对象。

E.2.1 命令行处理器（Command-Line Processor，CLP）

CLP 是 IBM DB2 命令行实用工具，允许执行 SQL 语句和调用联机帮助。本附录提供 DB2 数据库中运行交互式命令或脚本时 CLP 可以使用的基本命令和选项。与 Oracle 的 SQL*Plus 类似，CLP 是平台无关的。可以通过交互式输入模式、命令模式或批处理模式使用它。

1. 交互模式

要进入交互模式，打开“命令行处理器”窗口（“开始”→“所有程序”→IBM DB2→IBMCOPY1→“命令行工具”→“命令行处理器”）。参见图 E-8。

要了解最常用的选项，参见表 E-2。

表 E-2 常用 CLP 选项

选项	说明
-c	自动提交 SQL 语句。这个选项默认是打开的；即，所有语句都将自动提交，除非使用+c 选项（db2 +c）启动会话
-f <filename>	从文件<filename>中读取命令输入。必须指定文件的完整路径，除非它位于当前目录中： db2 -f /home/btrukhnov/db2/queries/my_query.sql

www.TopSage.com

大家网 TopSage.com

续表

选项	说明
-l <filename>	创建命令的日志文件。例如： db2 -f my_query.sql -l logfiles/my_query.log
-r <filename>	将命令输出记录到日志文件<filename>
-s	出现错误时停止执行；如果一个语句失败而期望脚本的执行终止时，通常与-t选项一起使用
-t	使用一个分号作为语句终止字符
-v	将命令文本回显到标准输出
-w	显示SQL语句警告消息
-z <filename>	将全部输出重定向到文件<filename>

2. 命令模式

打开“命令行处理器”窗口（“开始”→“所有程序”→IBM DB2→IBMCOPY1→“命令行工具”→“命令窗口”）。在此初始化环境变量，因此，可以输入以db2为前缀的CLP命令（参见图E-9）。

图E-8 在交互模式下启动DB2 9.5 CLP

图E-9 在命令模式下使用DB2 9.5 CLP

3. 批处理模式

批处理模式允许执行存储在操作系统文件中的SQL语句。它使用-f选项来调用（参见图E-2）。例如：

```
C:\Program Files\IBM\SQLLIB\BIN>db2 -f C:\myfiles\query.sql
```

执行C:\驱动器下myfiles目录中query.sql文件的内容（假定它存在）。

命令行处理器拥有许多命令；其中大多数用于数据库管理，但与本书内容无关。表E-3列出了一些比较有用的CLP命令。

表E-3 常用CLP命令

CLP命令	说明	例子
!	调用一条操作系统命令	db2 =>!dir
?	调用联机帮助	db2 =>? db2 =>?echo
DESCRIBE	说明表列或表的索引	db2 describe table address db2 descirbe indexes for table customer
ECHO	写入标准输出	db2 echo "Enter your query"
GET CONNECTION STATE	显示当前连接的状态	db2 get connection state
HELP	调用信息中心	db2 help
LIST ACTIVE DATABASES	显示准备用于连接的数据库列表	db2 list active databases
QUIT	退出CLP交互式输入模式	db2 =>quit
TERMINATE	类似于QUIT，但是终止全部后台处理并释放内存	db2 =>terminate

E.2.2 命令编辑器

DB2 9.5命令编辑器是一个GUI实用工具，用于生成、执行和编辑SQL语句，处理输出等。启动命令编辑器的最简单方式是在一个独立视图模式中（“开始”→“所有程序”→IBM DB2→DB2COPY1→“命令行工具”→“命令编辑器”）。从“目标”下拉列表中选择数据库（或者通过单击“Add（添加）”按钮配置），输入SQL语句，并单击“Execute（执行）”按钮（左上角的绿色三角）。参见图E-10了解详细信息。

www.TopSage.com

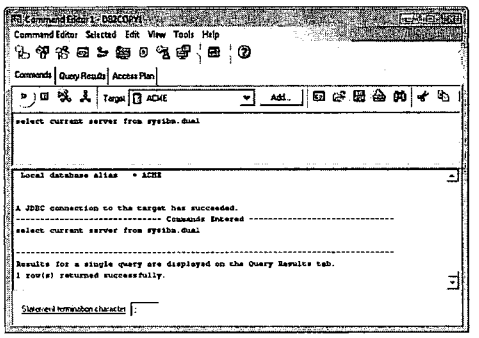


图 E-10 使用 DB2 9.5 命令编辑器执行 SQL 语句

另外，通过选择“工具”→“命令行编辑器”，也可以从“命令中心”内启动命令编辑器（参见附录 D）。此外，通过命令中心可以右键单击任意“可选择的”对象，即表或视图，并选择“Query（查询）”（这样可以节省输入时间）。命令编辑器将打开一个已经具有基本查询形式的语句：

```
SELECT * FROM <object_name>
```

E.3 使用 Microsoft SQL Server 2008 实用工具访问 RDBMS

Microsoft SQL Server 2008 提供了 SQLCMD 命令行实用工具来访问数据库；ISQL 和 OSQL 已经废弃。此外，Microsoft SQL Server 2008 包括 Microsoft 查询分析器——用于执行即席查询和脚本及分析输出的一个图形用户界面。

E.3.1 使用 SQLCMD

SQLCMD 是一个强大的命令行实用工具，它运行 Transact-SQL 语句和脚本的方式与 Oracle 的 SQL*Plus 或 DB2 的 CLP 类似。

要启动 SQLCMD，打开 Microsoft 命令提示窗口，输入 `SQLCMD -S .\ACME`，并按 Enter 键；屏幕如图 E-11 所示。SQLCMD 提示符（1>）表明已成功连接到期望的 SQL 服务器实例，在本例中是 ACME 位于本地服务器上。

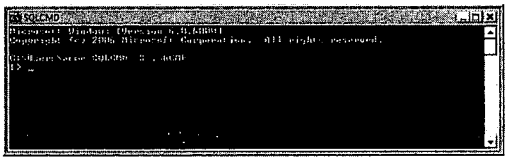


图 E-11 Microsoft SQL Server 2008 SQLCMD 提示符

根据 SQL Server 的身份验证模式、安装类型等，SQLCMD 可以有多个命令行选项。在本例中只需要指定服务器名（ACME），因为我们正在使用 Windows 身份验证（假设已经遵循了附录 D 中的安装指导），而且已经创建了一个本地命名实例 ACME 而不是称为 SQLEXPRESS 的默认实例。表 E-4 包含了一些常用的 SQLCMD 命令行选项；请参考 Microsoft SQL Server 文件查看完整列表。

表 E-4 常用 SQLCMD 命令行选项

选项	说明
-U <username>	用户登录 ID
-P <password>	用户指定的区分大小写的密码
-E	用户信任的连接而不是用户名和密码
-S <servername>	要连接的服务器名
-d <database_name>	要连接的数据库
-i [<path>]<filenanme>,...	包含待执行的 SQL 语句的输入文件
-o [<path>]<filenanme>	输出文件
-Q <command_line_query>	执行一个查询并退出。必须使用双引号封闭查询体

现在可以开始输入和执行 Transact-SQL 语句了。要执行语句，在新一行上输入 GO 并按 Enter 键。如果没有输入 GO 就按 Enter 键，那么 SQLCMD 会继续开始一个新行；每一行都以一个序号作为前缀（图 E-12）。

www.TopSage.com

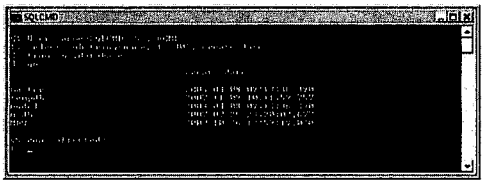


图 E-12 从 SQLCMD 执行 SQL 命令

表 E-5 中列出了一些最常用的 SQLCMD 命令。这些命令都区分大小写；对于大多数 SQLCMD 命令，前面的冒号都是可选的。

表 E-5 常用 SQLCMD 命令

CLP 命令	说明
:HELP	列出全部 SQLCMD 命令及其简要说明
[:]!!（双感叹号）	从 Microsoft 命令提示符执行一个命令
[:]ED	调用默认的内置编辑器。确保指定了一个默认编辑器
[:]EXIT [<(statement)>]或[:]QUIT	退出 SQLCMD 实用工具。EXIT 也允许返回一个查询结果
GO [<count>]	执行缓存的 SQL 语句。可选参数<count>能够多次执行 SQL 语句
[:]RESET	清除语句缓存

E.3.2 使用 SQL Server Management Studio

Windows 是一个图形环境，而 Microsoft SQL Server 2008 附带了一个功能完善的 GUI 实用工具——SQL Server Management Studio，它将 SQL Server 以前版本的企业管理器、查询分析器和分析管理器的功能合并到一个单一的环境中。它是 Microsoft SQL Server 最常用和最方便的查询界面。

可以从“开始”→“所有程序”→Microsoft SQL Server 2008→SQL Server Management Studio 调用这个实用工具。确保**连接到服务器**屏幕使用 Windows 身份验证；也要验证“服务器名”字段不为空；然后单击“连接”按钮（图 E-13）。

要执行 SQL 或 Transact-SQL 语句，单击 **SQL Server Management Studio** 窗口左上角的“新建查询”按钮，在右侧的**查询**窗口输入 SQL 语句，并单击“执行”按钮。查询结果将显示在下方的**结果**选项卡中（图 E-14）。

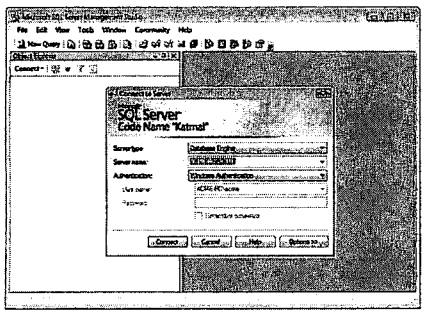


图 E-13 使用 SQL Server Management Studio 连接到一个数据库

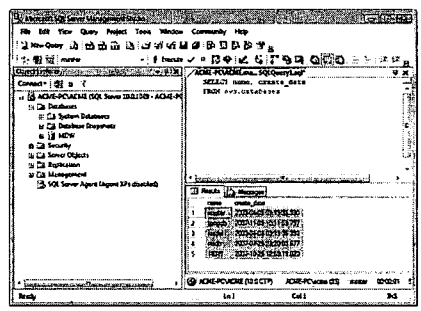


图 E-14 使用 SQL Server Management Studio 执行 SQL 查询

Tip

提示：也可以使用查询分析器在图形模式中创建查询，调整和分析查询，将输出重定向到一个文件，并执行很多其他任务，这些都超出了本书的范围。参见 Microsoft 文档了解详细信息。

www.TopSage.com

大家网
TopSage.com

APPENDIX

附录 F
安装 ACME 数据库

本附录介绍如何在选择的 RDBMS 上安装 ACME 示例数据库。请参考与已经创建的数据库平台的相应部分。

本附录提要：

- 在 Oracle 11g 上安装 ACME 数据库
- 在 DB2 9.5 上安装 ACME 数据库
- 在 Microsoft SQL Server 2008 上安装 ACME 数据库

F.1 在 Oracle 11g 上使用 SQL*Plus 安装 ACME 数据库

下面是有关使用 SQL*Plus 实用工具（在附录 E 中进行了说明）在 Oracle 11g 上安装 ACME 示例数据库的指导。

F.1.1 Linux/UNIX

（1）启动 Oracle 实例。确保会话中的 ORACLE_SID 环境变量指向正确的实例：

```
# echo $ORACLE_SID
ACME
#
```

交叉参考：附录D的表D-2中列出了运行sqlplus实用工具所必需的Unix/Linux环境变量。

（2）创建目录：/home/oracle/sqlbib/oracle_acme。

（3）从本书配套的网站把相应的脚本下载到已创建的目录中。

（4）将当前目录更改为/home/oracle/sqlbib/oracle_acme 并输入下面命令启动 SQL*Plus：

```
# sqlplus /nolog
```

（5）如果希望创建日志文件，则使用 spool 命令（在附录 E 中进行了说明）。例如，如果希望在与其他文件相同的目录中创建结果日志文件 load.log，则输入：

```
SQL> spool load.log
```

（6）运行脚本 sqlbib/oracle_acme/load.sql：

```
SQL> @load.sql
```

（7）这个脚本创建 Oracle ACME 数据库。通过发出如下命令停止后台处理（如果已启动）：

```
SQL> spool off
```

（8）输入 quit，退出 SQL*Plus。

（9）查看文件 load.log，了解可能出现的错误。

F.1.2 Microsoft Windows

（1）确保 Oracle 服务已启动。

（2）在计算机上创建目录：C:\sqlbib\oracle_acme，并从本书配套的网站将相应的 Oracle 脚本下载到已创建的目录中。

www.TopSage.com

（3）使C:\sqlbib\oracle_acme成为当前目录，从命令行启动SQL*Plus：

```
C:\> CD C:\sqlbib\oracle_acme
C:\sqlbib\oracle_acme> sqlplus /nolog
```

（4）如果希望创建一个日志文件，则使用spool命令（在附录E中进行了说明）。例如，如果希望在当前目录中创建结果日志文件load.log，则使用下面的命令：

```
SQL> spool load.log
```

（5）运行脚本C:\sqlbib\oracle_acme\load.sql：

```
SQL> @load.sql
```

（6）这个脚本创建Oracle ACME数据库。通过发出如下命令停止后台处理（如果已启动）：

```
SQL> spool off
```

（7）输入quit，退出SQL*Plus。

（8）查看文件load.log，了解可能出现的错误。

Note　注意：脚本load.sql运行两个另外的脚本：create_acme_ora.sql和load_data_ora.dat。第一个脚本创建RDBMS对象（表、索引、约束、视图等等），第二个脚本用数据填充ACME表。在尝试第6章中的例子之前，可能要运行第一个脚本。相应地，可以多次运行load.sql和create_acme_ora.sql，创建一个已加载的或空的ACME数据库。

在首次运行脚本时，可能会看到ORA-00942错误（表或视图不存在）。这不是一个问题；这个脚本正在试图删除还不存在的表。

表F-1中列出了可能在日志文件中看到的最常见错误。

表F-1　运行Oracle脚本的常见错误

错误	说明	解决方案
ORA-12560:TNS: protocol adapter error	Oracle没有启动	启动Oracle并尝试再次运行脚本
ORA-12154:TNS: couldn't resolve service name	提供了一个不正确的服务名	不必提供服务名，只需要用户ID和密码
ORA-01031: insufficient privileges	不是ORD_DBA ORD_DBA（Microsoft Windows）或dba（Linux/Unix）OS组的成员	作为ORA_DBA组成员的一个用户登录或将当前用户添加到ORA_DBA组
ORA-01033: Oracle initialization or shutdown in progress	Oracle正在启动（或停止）	等待并再次尝试

F.2　在DB2 9.5（Windows）上使用CLP安装ACME数据库

下面是有关如何在DB2 9.5(Windows)上使用CLP实用工具(在附录E中进行了说明)安装ACME示例数据库的指导：

（1）在计算机上创建目录C:\sqlbib\db2_acme并从本书配套的网站将相应的DB2脚本下载到已创建的目录中。

（2）确保创建如附录D“安装后的任务”小节中描述的ACME数据库并且数据库已经启动了。

（3）打开DB2命令窗口（“开始”→“程序”→IBM DB2→DB2COPY1→“命令行工具”→“命令窗口”）。

（4）使C:\sqlbib\db2_acme成为当前目录：

```
C:\Program Files\IBM\SQLLIB\BIN> CD C:\sqlbib\db2_acme
```

（5）在命令提示符下发出下面两个命令（假设希望日志文件在C:\sqlbib\db2_acme\中）：

```
C:\sqlbib\db2_acme_ db2 -tvf create_acme_db2.sql -z create_acme.log
C:\sqlbib\db2_acme_ db2 -tvf load_data_db2.dat -z load_data.log
```

（6）查看文件create_acme.log和load_data.log，了解可能出现的错误。

Note　注意：create_acme_ora.sql脚本创建RDBMS对象（表、索引、约束、视图等等），load_data_ora.dat脚本用数据填充ACME表。在尝试第6章中的例子之前，可能要运行第一个脚本。相应地，可以多次运行create_acme_ora.sql创建一个已加载的或空的ACME数据库。

www.TopSage.com

Tip 提示：在首次运行脚本时，在日志文件开头可能会看到一些这样的消息，如 <TABLE_NAME> IS UNDEFINED NAME。这种消息不是错误而是警告；因为还没有创建表，所以忽略这条警告。

表 F-2 中列出了运行 DB2 脚本时的一些常见错误。

表 F-2 运行 DB2 脚本时的常见错误

错误	说明	解决方案
DB21061E Command line environment not initialized	正在尝试从命令行提示符而不是 DB2 命令窗口执行 DB2 命令	启动 DB2 命令窗口并再次尝试
SQL0900N The application state is in error. A database connection does not exist. SQLSATE=08003	提供了一个不正确的服务名	如果遵守了附录 D 中推荐的 DB2 安装指导，则不必指定服务名
SQL1032N No start database manager command was issued. SQLSTATE=57019	数据库服务没有启动	如附录 D 中说明的那样启动数据库服务
SQL1035N The database is currently in use SQLSTATE=57019 SQL1005N The database alias... already exists...	脚本不能删除和重建现有数据库，因为目前另一个会话正在使用它。在连接到 ACME 数据库期间重新运行脚本时会产生这些错误	确保从全部 DB2 9.5 应用程序（CLP、命令编辑器、命令中心等）退出，再次尝试运行这些脚本
SQL0601N The name of the object to be created is identical to the existing name...SQLSTATE=42710	这个对象已经存在于数据库中	如果重建数据库失败而导致这条消息（参见前面的错误），则按照建议修复并再次运行脚本。否则在重建之前删除这个对象

F.3 在 Microsoft SQL Server 2008 上使用 SQLCMD 实用工具安装 ACME 数据库

下面是有关如何在 Microsoft SQL Server 2008 上使用 SQLCMD 实用工具（在附录 E 中进行了说明）安装 ACME 示例数据库的指导：

（1）如果 Microsoft SQL Server 服务没有运行，则如附录 D 中说明的那样启动这些服务。

（2）在计算机上创建目录 C:\sqlbib\mssql_acme 并从本书配套的网站下载全部 Microsoft SQL Server 脚本。

（3）使 C:\sqlbib\mssql_acme 成为当前目录并发出下面的命令（假设希望日志文件位于 C:\sqlbib\mssql_acme 中）：

```
sqlcmd -S .\ACME -i create_acme_mssql.sql -o create_acme.log
sqlcmd -S .\ACME -i load_data_mssql.dat -o load_data.log
```

（4）查看文件 create_acme.log 和 load_data.log，了解可能出现的错误。

表 F-3 中列出了运行 Microsoft SQL Server 脚本时的一些常见错误。

表 F-3 运行 Microsoft SQL Server 脚本的常见错误

错误	说明	解决办法
HResult 0xFFFFFFFF, Level 16, State 1 SQL Server Network Interfaces: Error Locating Server/Instance Specified [xFFFFFFFF]. Sqlcmd: Error: Microsoft SQL Server Native Client 10.0 : An error has occurred while establishing a connection to the server. When connecting to SQL Server 2005 or above, this failure may be caused by the fact that under the default settings SQL Server does not allow remote connections.. Sqlcmd: Error: Microsoft SQL Server Native Client 10.0 : Login timeout expired	Microsoft SQL Server 没有运行	启动 Microsoft SQL Server（参阅附录 D）
Msg 3702, Level 16, State 4, Server ACME-PC\ACME, Line 4 Cannot drop database "acme" because it is currently in use	当建立一个到 ACME 数据库的连接时脚本正试图删除该数据库。在使用像 Microsoft SQL Server Management Studio 这样的数据库工具连接到 ACME 数据库时重新运行脚本会产生这种错误	关闭应用程序，并再次尝试

www.TopSage.com

大家网
TopSage.com

APPENDIX G

附录 G
SQL 函数

本附录提要：

- Oracle 11g 函数
- IBM DB2 9.5 函数
- Microsoft SQL Server 2008 SQL 函数

每一个 RDBMS 都维护特定于它自己版本的一个不断扩展的 SQL 函数列表。本书第 10 章讨论了一些最有用的函数。这个附录提供了每一个 RDBMS 当前版本（Oracle 11g、IBM DB2 9.5 和 Microsoft SQL Server 2008）可用的完整函数列表。

所有函数按类似于在每一个相应的供应商文档中对它们的组织方式在表 G-1 到表 G-24 中进行分组。Oracle 11g 允许函数重载，这意味着有些函数根据数据类型和参数数目可以按多种方式使用；因此这样的函数列在多个类别中。IBM DB2 9.5 和 Microsoft SQL Server 2008 很少使用重载。

某些函数需要适当的特权才能运行，而一些函数返回的结果不能有效显示（二进制、对象、表值等）。表中省略了函数的参数，因为这个附录的目的是显示哪些函数可用，而不深入具体细节。参考每个供应商的文档来获得更详细的语法和说明。

表 G-1　　Oracle 11g SQL 数值函数

函数名	简要说明
ABS	返回 n 的绝对值
ACOS	返回 n 的反余弦
ASIN	返回 n 的反正弦
ATAN	返回 n 的反正切
ATAN2	返回 n 和 m 的反正切
BITAND	返回一个整数，表示对两个表达式 expr1 和 expr2（作为参数而传递的）的位进行 AND 操作
CEIL	返回大于等于 n 的最小整数
COS	返回 n 的余弦
COSH	返回 n 的双曲余弦
EXP	返回 e 的 n 次幂，这里 e 是自然对数的基数（e~2.71828183）
FLOOR	返回小于等于 n 的最大整数
LN	返回 n 的自然对数，这里 n 大于 0
LOG	返回以 m 为底的 n 的对数
MOD	返回 m 除以 n 的余数。如果 n 是 0，则返回 m
NANVL(m,n)	如果输入值是 NaN（不是一个数字），则返回另一个值 n，NANVL 函数只对浮点数值类型 BINARY_FLOAT 或 BINARY_DOUBLE 才有用

- www.TopSage.com

续表

函数名	简要说明
POWER	返回 m 的 n 次幂
REMAINDER(m,n)	返回 m 除以 n 的余数
SIGN	如果 n<0，则 SIGN 返回-1。如果 n=0，则这个函数返回 0。如果 n>0，则 SIGN 返回 1
SINH	返回 n 的双曲正弦
SQRT	返回 n 的平方根
TAN	返回 n 的正切
TANH	返回 n 的双曲正切
TRUNC	返回一个被截断到 m 小数位的值。如果省略 m，则截取 n 到 0 位。m 可以是负值以截断小数点左边的 m 位（变为 0）
WIDTH_BUCKET	构造等宽直方图，其中直方图的范围分为等量的间隔

表 G-2 Oracle 11g SQL 字符函数

函数名	简要说明
ASCII	返回 char 第一个字符在数据库字符集中的十进制表示形式
CHR	返回具有等价于 n 的二进制字符
CONCAT	返回连接了 char2 的 char1
INITCAP	返回 char，第一个字母大写，其余所有字母小写
INSTR	返回一个整数，指示这个字符串的第一个字符出现在一个字符串中的位置。INSTR 使用像输入字符集那样定义的字符来计算字符串。INSTRB 使用字节而不是字符。INSTRC 使用 Unicode 完整字符。INSTR2 使用 UCS2 代码点。INSTR4 使用 UCS4 代码点
LENGTH	使用像输入字符集那样定义的字符返回 char.LENGTH 计算的长度。LENGTHB 使用字节而不是字符。LENGTHC 使用 Unicode 完整字符。LENGTH2 使用 UCS2 代码点。LENGTH4 使用 UCS4 代码点
LOWER	返回 char，所有字母小写
LPAD	返回 char1，使用 char2 中的字符序列从左开始填充，长度为 n；如果 char1 大于 n，则这个函数返回 char1 匹配 n 的部分
LTRIM	从 char 的左边删除字符，删除所有出现在 set 最左边的字符；set 默认为单个空白
NLS_CHARSET_DECL_LEN	返回一个 NCHAR 列的声明长度（按字符数）。参数 byte_count 是列宽
NLS_CHARSET_ID	返回与字符集名称字符串对应的字符集 ID 号
NLS_CHARSET_NAME	返回与 ID 号对应的字符集名称
NLS_INITCAP	返回 char，每一个单词的首字母大写，其余所有字母小写
NLS_LOWER	返回 char，所有字母小写
NLSSORT	返回用于排序 char 的字节字符串
NLS_UPPER	返回 char，所有字母大写
REGEXP_INSTR	返回一个整数，指示匹配子字符串的开始或结束位置，具体取决于参数 return_option 的值。如果没有找到匹配的串，则函数返回 0
REGEXP_REPLACE	返回源<char>，其中每次出现的正则表达式样式替换为 replace<string>
REGEXP_SUBSTR	搜索一个字符串的正则表达式样式，并返回子字符串本身
REPLACE	返回 char，其中每次出现的 search_string 替换为 replacement_string
RPAD	返回 char1，需要时多次复制 char2 来右填充到长度 n
RTRIM	返回 char，删除所有出现在 set 最右边的字符
SOUNDEX	返回一个包含 char 的语音形式的字符串
SUBSTR	返回 string 的一部分，在一个长度为 substring_length 个字符的指定字符位置开始。SUBSTR 使用由输入字符集定义的字符来计算长度
SUBSTRB	与 STRING 一样，只是 SUBSTRB 使用字节而不是字符
SUBSTRC	与 STRING 一样，只是 SUBSTRC 使用 Unicode 完整字符
SUBSTR2	与 STRING 一样，只是 SUBSTR2 使用 UCS2 代码点
SUBSTR4	与 STRING 一样，只是 SUBSTR4 使用 UCS4 代码点
TRANSLATE	返回 char，在 from_string 中出现的每一个字符都替换为其在 to_string 中的对应字符
TREAT	返回表达式的一个声明类型
UPPER	返回 char，所有字母都大写。char 可以是任意数据类型：CHAR、VARCHAR2、NCHAR、NVARCHAR2、CLOB 或 NCLOB。返回值是与 char 相同的数据类型

www.TopSage.com

表 G-3 Oracle 11g SQL 日期时间函数

函数名	简要说明
ADD_MONTHS	返回日期 *d* 加上 *n* 月
CURRENT_DATE	使用一个具有 DATE 数据类型的阳历格式的值返回会话时区的当前日期和时间
CURRENT_TIMESTAMP	使用 TIMESTAMP WITH TIME ZONE 数据类型的值返回会话时区的当前日期和时间
DBTIMEZONE	返回数据库时区的值
EXTRACT	从一个日期时间或时间间隔值表达式返回一个指定日期时间字段的值
FROM_TZ	把一个时区中的一个时间戳值转换成一个 TIMESTAMP WITH TIME ZONE 值。time_zone_value 是一个格式为 TZH：TZM 的字符串或一个字符表达式，这个表达式以具有可选 TZD 格式的 TZR 返回一个字符串
LAST_DAY	返回包含这个日期的那个月的最后一天
LOCALTIMESTAMP	在一个 TIMESTAMP 数据类型的值中返回会话时区的当前日期和时间
MONTH_BETWEEN	返回日期 date1 和 date2 之间的月数
NEW_TIME	当时区 zone1 中的日期和时间都是日期参数时，返回时区 zone2 中的日期和时间
NEXT_DAY	返回日期参数之后由 char 指定的第一个星期日
NUMTODSINTERVAL	把 *n* 转换为一个 INTERVAL DAY TO SECOND 文字值。*n* 可以是一个数字，也可以是解析成一个数字的一个表达式
NUMTOYMINTERVAL	把 *n* 转换为一个 INTERNAL YEAR TO MONTH 文字值。*n* 可以是一个数字，也可以是解析成一个数字的一个表达式
ROUND	截断日期的时间部分
SESSIONTIMEZONE	返回当前会话的时区值
SYS_EXTRACT_UTC	从一个具有时区位移的日期时间中提取 UTC（Coordinated Universal Time，协调世界时）
SYSTEMSTAMP	返回系统日期，包括秒和数据库的时区
SYSDATE	返回当前的日期和时间
TO_DSINTERVAL	把一个 CHAR、VARCHAR2、NCHAR 或 NVARCHAR2 数据类型的字符串转换为一个 INTERVAL DAY TO SECOND TYPE
TO_TIMSTAMP	把 CHAR、VARCHAR2、NCHAR 或 NVARCHAR2 数据类型的 char 转换为一个 TIMESTAMP 数据类型的值
TO_TIMSTAMP_TZ	把 CHAR、VARCHAR2、NCHAR 或 NVARCHAR2 数据类型的 char 转换为一个 TIMESTAMP WITH TIME ZONE 值
TO_YMINTERVAL	把一个 CHAR、VARCHAR2、NVARCHAR 或 NVARCHAR2 数据类型的字符串转换为一个 INTERVALYEAR TO MONTH 类型
TRUNC	返回日期的时间部分被截取到由格式模型模式指定的单位日期。如果省略 format，date 将被截取到最近的日
TZ_OFFSET	基于执行语句的日期，返回与输入值对应的时区偏移量。可以输入一个有效的时区名、一个相对 UTC 的时区偏移量（简单地返回它自身），或者关键字 SESSIONTIMEZONE 或 DBTIMEZONE。要获得一个有效值列表，可以查询 V$TIMEZONE_NAMES 动态性能视图的 TZHAME 列

表 G-4 Oracle 11g 内置数据挖掘函数*

函数名	简要说明
CLUSTER_ID	返回在 mining_attribute_clause 中指定的预测器组中具有最高可能性的已预测聚集的标识符
CLUSTER_PROBABILITY	返回在一个群集中与指定模型关联的输入行的成员受信程度测量值
CLUSTER_SET	返回包含一个给定行所属的全部可能聚集的对象列
FEATURE_ID	返回一个 NUMBER，它是具有最大系数值的功能的标识符
FEATURE_SET	返回包含全部可能功能的对象列
FEATURE_VALUE	返回一个给定功能的值
PREDICTION	返回模型的最佳预测
PREDICTION_COST	返回作为一个 NUMBER 的一个给定预测的成本测量
PREDICTION_DETAILS	返回一个 XML 字符串，它包含与输入行得分有关的模型特定信息
PREDICTION_PROBABILITY	返回作为一个 NUMBER 的一个给定预测的可能性
PREDICTION_SET	返回包含一个多类分类场景中的全部类的对象列

*通过这些函数使用 DBMS_DATA_MINING 包或 Oracle 数据挖掘 Java API 创建的聚类分析模型。

www.TopSage.com

TopSage.com

表 G-5 Oracle 11g 内置 XML 函数

函数名	简要说明
APPENDCHILDXML	将一个用户提供的值作为由一个 XPath 表达式表示的节点的子节点附加到目标 XML
DELETEXML	删除目标 XML 中由 XPath 表达式匹配的一个或多个节点
DEPTH	返回由 UNDER_PATH 条件（具有同样的相关变量）指定路径中的层级数
EXTRACT(XML)	应用一个 VARCHAR2 XPath 字符串并返回一个包含一段 XML 的 XMLType 实例
EXISTNODE	确定利用指定路径遍历一个 XML 文档是否会产生任何节点
EXTRACTIVALUE	将一个 XMLType 实例和一个 XPath 表达式作为参数并返回一个作为结果节点的标量值
INSERTCHILDXML	将一个用户提供的值插入到目标 XML 中由 XPath 表达式指示的节点
INSERTXMLBEFORE	将一个用户提供的值插入到目标 XMLType 中由 XPath 表达式指示的节点之前
PATH	返回父条件中指定资源所在的相对路径
SYS_DBURIGEN	将一个或多个列或属性以及一个可选 rowid 作为其参数，并生成一个数据类型为 DBURIType 的 URL，指向一个特定的列或行对象
SYS_XMLAGG	聚合由 expr 表示的全部 XML 文档或片段并生成一个单独的 XMLType 文档。它添加了一个新的封闭元素，默认名是 ROWSET
SYS_XMLGEN	采用一个对数据库的一个特定行和列求值的表达式，返回一个包含一个 XML 文档的 XMLType 类型的实例
UPDATEXML	将一个 XMLType 实例和一个 XPath 值对作为参数，返回一个具有已更新值的 XMLType 实例
XMLAGG	采用一个 XML 片段的集合，返回一个已聚合的 XML 文档
XMLCDATA	对<expr>求值生成一个 CDATA 片段。<expr>必须解析为一个字符串
XMLCOLATTVAL	创建一个 XML 片段，然后扩展生成的 XML，以便每个 XML 片段具有使用了属性名的名称列
XMLCOMMENT	使用<expr>的求值结果生成一个 XML 注释
XMLCONCAT	将一组 XMLType 实例作为输入，连接每行的一系列元素，并返回连接后的系列。XMLConcat 是 XMLSequence 的逆序
XMLFOREST	将它的每个变量参数转换为 XML，然后返回一个 XML 片段，它连接了这些转换后的参数
XMLPARSE	从<expr>的求值结果解析生成一个 XML 实例。<expr>必须解析为一个字符串
XMLPI	使用一个标识符和可选的<expr>求值结果生成一条 XML 处理指令
XMLQUERY	使用 XQuery 表达式，作为字符串文字、一个可选的上下文项和其他绑定变量，返回对使用这些输入值的 XQuery 表达式的一个求值结果
XMLROOT	允许通过提供的版本和一个现有 XML 值的 XML 根信息（序言）中的单独属性来创建一个新的 XML 值
XMLSEQUENCE	为游标的每一行返回一个 XMLSequence 类型的 XML 文档。返回 XMLType 的顶层节点的一列
XMLSERIALIZE	创建一个字符串或包含<expr>内容的 LOB
XMLTABLE	将一个 XQuery 求值结果映射到相关的行和列
XMLTRANSFORM	将一个 XMLType 实例和一个 XSL 样式表作为参数，它们本身是一个 XMLType 实例形式。它对实例应用样式表并返回一个 XMLType

表 G-6 Oracle 11g 内置集合函数

函数名	简要说明
CARDINALITY	返回一个嵌套表中的元素数
COLLECT	将任意类型的一列作为其参数，创建在选择的行之外的输入类型的嵌套表。要得到这个函数的结果，必须在一个 CAST 函数内使用它
POWERMULTISET	将一个嵌套表和一个基数作为输入，返回包含输入嵌套表中全部非空子集（称为子多集合）的嵌套表中的一个
POWERMULTISET_BY_CARDINALITY	将一个嵌套表和一个基数作为输入，返回包含指定基数的嵌套表中全部非空子集（称为子多集合）的嵌套表中的一个
SET	返回一个嵌套表，其元素彼此各不相同

表 G-7 Oracle 11g 内置模型函数

函数名	简要说明
CV	返回从一个规则左边传送到右边的一个维度列的当前值
ITERATION_NUMBER	返回一个表示贯穿模型规则的已完成迭代的整数
PRESENTNNV(<cell_ref>, Expr1, expr2)	在 model_clause 执行之前，当 cell_reference 存在且不为空时返回 expr1；否则返回 expr2
PRESENTV	在 model_clause 执行之前，当 cell_reference 存在时返回 expr1；否则返回 expr2
PREVIOUS	在每次迭代开始时返回 cell_reference 的值

www.TopSage.com

表 G-8 Oracle 11g SQL 转换函数

函数名	简要说明
ASCIISTR	以数据库字符集返回一个 ASCII 字符串
BIN_TO_NUM	把一个位向量转换为等价数字。这个函数中的每一个参数代表位向量中的一位。每一个表达式必须等于 0 或 1。这个函数返回 NUMBER
CAST	把一个内置数据类型或集合类型值转换为另一个内置数据类型或集合类型值
CHARTOROWID	把一个值从 CHAR、VARCHAR2、NCHAR 或 NVARCHAR2 数据类型转换为 ROWID 数据类型
COMPOSE	按照与输入相同的字符集返回一个完全规范化形式的 Unicode 字符串
CONVERT	把一个字符串从一个字符集转换为另一个字符集。返回值的数据类型是 VARCHAR2
DECOMPOSE	按照与输入相同的字符集返回在规范分解之后的一个 Unicode 字符串
HEXTORAW	把包含十六进制数字的 CHAR、VARCHAR2、NVARCHAR 或 NVARCHAR2 字符集的 char 转换成一个 raw 值
NUMTODSINTERVAL	把 *n* 转换成一个 INTERVAL DAY TO SECOND LITERAL。*n* 可以是一个数字或一个解析成一个数字的表达式
NUMTOYMINTERVAL	把数字 *n* 转换成一个 INTERVAL YEAR TO MONTH 文字。*n* 可以是一个数字或一个解析成一个数字的表达式
RAWTOHEX	把 raw 数据类型转换成一个包含其十六进制等价值的字符值
RAWTONHEX	把 raw 数据类型转换成一个包含其十六进制等价值的 NVARCHAR2 字符值
ROWIDTOCHAR	把一个 rowid 值转换成一个 VARCHAR2 数据类型。这个转换的结果总是 18 个字符长
ROWIDTONCHAR	把一个 rowid 值转换成一个 NVARCHAR2 数据类型。这个转换的结果总是 18 个字符长
TO_CHAR	把 NCHAR、VARCHAR2、LOB 或 NCLOB 数据类型转换为数据库字符集
TO_CHAR	把 DATE、IMESTAMP、IMESTAMP WITH TIME ZONE 或 TIMESTAMP WITH A LOCAL TIME ZONE 数据类型的日期转换成一个由日期格式指定格式的 VARCHAR2 数据类型的值
TO_CHAR	把 NUMBER 数据类型的 *n* 转换成一个 VARCHAR2 数据类型的值
TO_CLOB	把一个 LOB 列中的 NCLOB 值或其他字符串转换成 CLOB 值
TO_DATE	把 CHAR、VARCHAR2、NCHAR 或 NVARCHAR2 数据类型的 char 转换成一个 DATE 数据类型的值
TO_DSINTERVAL	把一个 CHAR、VARCHAR2、NCHAR 或 NVARCHAR2 数据类型的字符串转换成一个 INTERVAL DAY TO SECOND TYPE
TO_LOB	把 long_column 列中的 LONG 或 LONG RAW 值转换成 LOB 值
TO_MULTI_BYTE	返回 char，所有单字节字符都转换成它们相应的多字节字符。返回值具有相同的数据类型 char
TO_NCHAR	把字符串 CLOB、NCLOB 从数据库字符集转换为区域字符集
TO_NCHAR	把 DATE、TIMESTAMP、TIMESTAMP WITH TIME ZONE 等数据类型的字符串从数据库字符集转换为区域字符集
TO_NCHAR	把一个数字转换成一个 NVARCHAR2 字符集的字符串
TO_NCLOB	把一个 LOB 列的 CLOB 值或其他字符串转换成 NCLOB 值
TO_NUMBER	把 char 转换成一个 NUMBER 数据类型的值
TO_SINGLE_BYTE	返回 char，所有多字节字符都转换成相应的单字节字符
TO_YMINTERVAL	把 CHAR、VARCHAR2、NCHAR 或 NVARCHAR2 数据类型的字符串转换成一个 INTERVAL YEAR TO MONTH 数据类型，这里 char 是要被转换的字符串
TRANSLATE...USING	把文本转换成为在数据库字符集和区域字符集之间进行转换而指定的字符集
UNISTR	把其参数作为一个任意字符集的字符串，并返回数据库 Unicode 字符集中的 Unicode 形式

表 G-9 Oracle 11g SQL 杂项单行函数

函数名	简要说明
BFILENAME	返回一个 BFILE 定位符，它与服务器文件系统的一个物理 LOB 二进制文件相关联
COALESCE	返回表达式列表中第一个非 NULL 表达式
DECODE	把一个表达式同每一个搜索值依次比较。如果表达式等于某个搜索值，则 Oracle 返回相应的结果。如果没有找到匹配，就返回默认值，或者，如果省略默认值，则返回 NULL
DUMP	返回一个 VARCHAR2 值，它包含字节形式表示的数据类型代码长度和表达式的内部表示
EMPTY_BLOB	返回一个空的 LOB 定位符，可以用于初始化一个 LOB 变量或者一条 INSERT 或 UPDATE 语句
EMPTY_CLOB	返回一个空的 LOB 定位符，可以用于初始化一个 LOB 变量或者一条 INSERT 或 UPDATE 语句
EXISTSNODE	确定利用路径遍历文档是否会产生任何节点。它将包含一个 XML 文档的 XML Type 实例和一个指定了路径的 VARCHAR2 字符串作为参数

www.TopSage.com

续表

函数名	简要说明
EXTRACT	从一个日期时间或时间间隔值表达式中返回指定日期时间字段的值
GREATEST	返回表达式列表中的最大值
LEAST	返回表达式列表中的最小值
NLS_CHARSET_DECL_LEN	返回一个NCHAR列声明的宽度（以字符数）
NLS_CHARSET_ID	返回对应于字符集名称文本的字符集ID号
NLS_CHARSET_NAME	返回对应于ID号的字符集名称
NULLIF	如果expression1和expression2相等，则返回NULL。如果它们不等，则这个函数返回expression1
NVL	如果expression1是NULL，则NVL返回expression2。如果不是NULL，则NVL返回expression1
NVL2	如果expression1不是NULL，则NVL2返回expression2，如果expression2是NULL，NVL2返回expression3
ORA_HASH	计算一个给定表达式的哈希值
SYS_CONNECT_BY_PATH	返回从根到节点的一个列值的路径，对CONNECT BY条件返回的每一行由char分隔列值
SYS_CONTEXT	返回与上下文命名空间关联的参数的值
SYS-DBURIGEN	以一个或多个列或属性以及一个可选的rowid作为参数，生成到一个特定列或行对象的DBUriType数据类型的URL。然后可以用这个URL从数据库中获得一个XML文档
SYS_EXTRACT_UTC	从一个具有时区位移的日期时间中获取UTC（Coordinated Universal Time，协调世界时，以前称为格林威治标准时间）。返回的日期时间具有时区位移
SYS_GUID	返回一个由16个字节构成的全局唯一标识符（RAW值）
SYS_TYPEID	返回操作数的最确切类型typeid
SYS_XMLAGG	聚合所有XML文档或由一个表达式表示的片段并生成一个单独的XML文档
SYS_XMLGEN	采用一个对数据库的某特定行或列求值的表达式，并返回一个包含一个XML文档的SYS.XMLType类型的实例
UID	返回一个唯一识别会话用户（登录的用户）的整数
USER	以VARCHAR2数据类型返回会话用户（登录的用户）的名称
USERENV	以VARCHAR2数据类型返回有关当前会话的信息。USERENV是一个为向后兼容而保持的遗留函数；使用SYS_CONTEXT函数来代替
VSIZE	VSIZE返回表达式内部表示的字节数。如果表达式是NULL，则这个函数返回NULL

表G-10 Oracle 11g SQL聚合函数

函数名	简要说明
AVG	返回一个表达式的平均值，可以与GROUP BY一起使用
COLLECT	以任意类型的一列作为其参数，创建已选择的行以外的输入类型的一个嵌套表。要得到这个函数的结果，必须在一个CAST函数内使用它
CORR	返回一个数值对集合的相关系数
COUNT	返回查询中的行数
COVAR_POP	返回一个数值对集合的总体协方差
COVAR_SAMP	返回一个数值对集合的样本协方差
CUME_DIST	作为一个聚合函数，为一个由函数的参数和一个相应的排序规范识别的假定行R计算行R在聚合组中的相对位置
DENSE_RANK	作为一个聚合函数，计算由函数的参数并参考一个给定的排序规范所识别的一个假定行的密秩
FIRST	返回一个排序分组的第一行的值
GROUP_ID	识别由一个GROUP BY规范产生的重复分组。因此，它在筛选查询结果中重复的分组方面是有用的。它返回一个NUMBER来唯一地识别重复的分组
GROUPING	从常规分组行中区分超聚合行。诸如ROLLUP和CUBE这样的GROUP BY扩展插件生成超聚合行，其中所有值的集合通过NULL表示。使用GROUPING函数可以区分表示一个超聚合行中所有值集合的一个NULL和常规行中的NULL
GROUPING_ID	返回一个数值，它对应与一个行关联的GROUPING位向量。GROUPING_ID只适用于包含诸如ROLLUP或CUBE这样的GROUP BY扩展插件以及GROUPING函数的SELECT语句
LAST	返回一个排序分组的最后一行的值
MAX	返回一个表达式的最大值

www.TopSage.com

大家网 TopSage.com

续表

函数名	简要说明
MIN	返回一个表达式的最小值
PERCENTILE_CONT	是一个反分布函数，它假定了一个连续的分布模型。它采用一个百分点值和一个排序规范，并返回一个就排序规范而言属于这个百分点值的一个内插值。在计算中忽略NULL
PERCENTILE_DISC	是一个反分布函数，它假定了一个离散的分布模型。它采用一个百分点值和一个排序规范并返回集合中的一个元素。在计算中忽略NULL
PERCENT_RANK	类似于CUME_DIST（累积分布）函数。由PERCENT_RANK返回的值的范围是0～1，包括0和1。任何集中的第一行都有一个PERCENT_RANK为0
RANK	作为一个聚合函数，计算由函数的参数以及一个给定的排序规范所标识的一个假定行的秩。在每一个聚合分组中，函数的参数必须等于常量表达式，因为它们标识每一个分组中的一个单个行
STATS_BINOMIAL_TEST	实现用于二叉变量的一个准确的概率测试，其中只存在两个可能值。它测试一个样本比例与一个给定比例之间的不同
STATS_CROSSTAB	用于分析两个标称变量的一种方法。STATS_CROSSTAB函数使用三个参数：两个表达式和一个VARCHAR2类型的返回值。expr1和expr2是要分析的两个变量。函数返回一个数值，它取决于第三个参数的值
STATS_F_TEST	测试两种差异是否的确不同；采用三个参数：expr1是分组或独立的变量，而expr2是值的样本。函数返回一个数值，它取决于第三个参数的值
STATS_KS_TEST	实现Kolmogorov-Smirnov函数，它比较两个样本，测试它们是否来自同一个总体或具有相同分布的总体
STATS_MODE	采用一组值作为参数，并返回出现频率最大的值。如果存在多个模式，则只选择一个
STATS_MW_TEST	Mann Whitney测试的实现，比较两个独立样本，测试具有相同分布函数的两个总体的空假设，以及两个分布函数不同的另一种假设
STATS_ONE_WAY_ANOVA	实现方差函数的单向分析，它通过比较两种不同的方差估计使用统计显著性的方式（对于组或变量）来测试不同
STATS_T_TEST_ONE STATS_T_TEST_PAIRED STATS_T_TEST_INDEP STATS_T_TEST_INDEPU	这个t测试组的函数测量各种方式的显著性差异。它包括： 单样本的t测试 双样本、成对的t测试（也就是所谓的交叉t测试） 两个独立的具有相同方差（合并方差）的组的t测试 两个独立的具有不等方差（非合并方差）的组的t测试
STATS_WSR_TEST	实现成对样本的Wilcoxon Signed Rank测试，确定样本之间差异的中值是否显著地区别于0。这些差异的绝对值是有序的且赋予了秩
STDDEV	返回一个表达式的样本标准偏差，一组数值；与STDDEV_SAMP不同，当它只有1行输入数据时，STDDEV返回0，而STDDEV_SAMP返回NULL
STDDEV_POP	计算总体标准偏差并返回总体方差的平方根
STDDEV_SAMP	计算累积样本标准偏差并返回样本方差的平方根
SUM	返回一个表达式的值之和
VAR_POP	在丢弃了集合中的NULL之后，返回一个数值集合的总体方差
VAR_SAMP	在丢弃了集合中的NULL之后，返回一个数值集合的样本方差
VARIANCE	返回一个表达式方差。可以用作一个聚合函数或分析函数

表G-11 Oracle 11g SQL分析函数

函数名	简要说明
AVG	返回一个表达式的平均值，可以返回分组中的多行
CORR	返回一个数值对集合的相关系数
COVAR_POP	返回一个数值对集合的总体协方差
COVAR_SAMP	返回一个数值对集合的样本协方差。可以把它用作一个聚合函数或分析函数
COUNT	返回查询中的行数
CUME_DIST	计算一个值组中的一个值的累积分布。由CUME_DIST返回的值的范围是大于0小于等于1
DENSE_RANK	计算一个有序行的分组中的一行的秩。秩是从1开始的连续整数。最大的秩值是由查询返回的惟一值的数目。秩不会在绑定事件中被跳过。对秩准则而言具有等值的行获得同样的秩
FIRST_VALUE	返回一个排序值集的第一个值
LAG	不使用一个自连接就可以提供同时对一个表的多行访问。给定从一个查询返回的一系列行和一个游标位置，LAG提供对位于那个位置之前给定的物理偏移量的一行的访问
LAST	返回一个排序分组的最后一行的值

www.TopSage.com

续表

函数名	简要说明
LAST_VALUE	返回一个排序值集的最后一个值
LEAD	不使用一个自连接就可以提供同时对一个表的多行访问。给定从一个查询返回的一系列行和一个游标位置，LEAD 提供对位于那个位置之后的给定物理偏移量的一行的访问权限
MAX	返回一个表达式的最大值
MIN	返回一个表达式的最小值
NTILE	把一个有序的数据集分成由一个表达式表示的若干存储段，并把适当的存储段数分配到每一行。存储段从 1 到表达式进行编号，而且对于每个分区而言，表达式必须解析成一个正常数
PERCENT_RANK	类似于 CUME_DIST（累积分布）函数。由 PERCENT_RANK 返回的值的范围是 0～1，包括 0 和 1。任何集中的第一行都有一个为 0 的 PERCENT_RANK
PERCENTILE_COUNT	是一个反分布函数，它假定了一个连续的分布模型。它采用一个百分点值和一个排序规范，并返回一个对于排序规范而言属于这个百分点值的一个内插值。在计算中忽略 NULL
PERCENTILE_DISC	是一个反分布函数，它假定了一个离散的分布模型。它采用一个百分点值和一个排序规范并返回集合中的一个元素。在计算中忽略 NULL
RANK	计算一个值分组中的一个值的秩。对于秩准则而言等值的行获得同样的秩；这些秩不能是连续的数值
RATIO_TO_REPORT	计算一个值与值集之和的比。如果表达式等于 NULL，则 ratio-to-report 的值也将为 NULL
ROW_NUMBER	把一个唯一的数值分配到应用它的每一行（可以是分区中每一行或者返回的每一行），按照 order_by_clause 中指定的行的顺序，且从 1 开始
REGR_SLOPE	线性回归函数遵循使一个普通最小平方回归线适合一个数值对集合： REGR_SLOPE 返回线性的倾斜度。返回值是一个数值并且可以为 NULL
REGR_INTERCEPT	REGR_INTERCEPT 返回回归线的 y 截距
REGR_COUNT	REGR_COUNT 返回一个整数，它是用于适合回归线的非 NULL 数值对的数目
REGR_R2	REGR_R2 返回回归的决定系数（也被称为“R 平方”或“适合度”）
REGR_AVGX	REGR_AVGX 计算回归线的自变量（expr2）的平均值
REGR_AVGY	REGR_AVGY 计算回归线的因变量（expr1）的平均值
REGR_SXX REGR_SYY REGR_SXY	REGR_SXY、REGR_SXX 和 REGR_SYY 是辅助函数，用于计算不同的诊断统计值
STDDEV	返回一个表达式的样本标准偏差，一个数值集；与 STDDEV_SAMP 不同，当它只有 1 行输入数据时，STDDEV 返回 0，而 STDDEV_SAMP 返回一个 NULL
STDDEV_POP	计算总体标准偏差并返回总体方差的平方根
STDDEV_SAMP	计算累积样本标准偏差并返回样本方差的平方根
SUM	返回一个表达式的值之和
VAR_POP	在丢弃了集合中的 NULL 之后，返回一个数值集的总体方差
VAR_SAMP	在丢弃了集合中的 NULL 之后，返回一个数值集的样本方差
VARIANCE	返回一个表达式方差。可以用作一个聚合函数或分析函数

表 G-12 Oracle 11g SQL 对象引用函数

函数名	简要说明
DEREF	返回一个参数表达式的对象引用，这里表达式必须返回一个对一个对象的 REF
MAKE_REF	创建对一个对象视图的一行或一个其对象标识符是基于主键的对象表的一行的一个 REF
REF	为一个绑定到变量或行的对象实例返回一个 REF 值
REFTOHEX	把一个参数表达式转换成一个包含其十六进制等价值的字符值。表达式必须返回一个 REF
VALUE	把与一个对象表的一行相关联的相关变量（表别名）作为其参数，并返回存储在对象表中的对象实例。对象实例的类型与对象表相同

表 G-13 IBM DB2 9.5 内置标量函数

函数名	简要说明
ABS[VAL]	返回一个数字 *n* 的绝对值
ACOS	返回作为一个以弧度表示的角的参数的反余弦
ASCII	返回一个字符串的第一个字符的 ASCII 码
ASIN	返回作为一个以弧度表示的角的参数的反正弦

www.TopSage.com

续表

函数名	简要说明
ATAN	返回作为一个以弧度表示的角的参数的反正切
ATAN2	返回作为一个以弧度表示的角的*x*和*y*坐标的反正切。*x*和*y*坐标分别由第一个和第二个参数指定
BIGINT	返回一个数值或一个整数常量形式的字符串的一个64位整数表示形式
BLOB	返回任意类型的一个字符串的BLOB表示形式
CEIL[ING]	返回大于等于*n*的最小整数
CHAR	把参数表达式转换成一个字符串
CHARACTER_LENGTH	返回在指定字符串参数中的一个表达式的长度
CHR	返回ASCII码的一个字符
CLOB	返回一个字符串类型的一个CLOB表示形式
COALESCE	返回列表中第一个不是NULL的参数
CONCAT	返回两个字符串串联的结果
COS	返回参数的余弦，这里参数是一个以弧度表示的角
COSH	返回参数的双曲余弦，这里参数是一个以弧度表示的角
COT	返回参数的余且，这里参数是一个以弧度表示的角
DATAPARTITIONNUM	返回行所在的数据分区的序列号（SYSDATAPARTITIONS.SEQNO）
DATE	从一个值返回一个日期
DAY	返回一个值的日部分
DAYNAME	返回一个大小写混合的字符串，它包含基于启动数据库所在地区的参数的日部分名称
DAYOFWEEK	返回参数中的星期几，参数是一个1～7的整数值，这里1表示星期日
DAYOFWEEK_ISO	返回参数中的星期几，参数是一个1～7的整数值，这里1表示星期一
DAYOFYEAR	返回参数中年的日，参数是一个1～366的整数值
DAYS	返回日期的整数表示形式
DBCLOB	返回图形字符串类型的DBCLOB表示形式
DBPARTITIONNUM	返回某行的数据库分区数
DEC[IMAL]	返回数值的十进制表示形式、十进制数值的字符串表示形式或整数数值的字符串表示形式
DECODE	DECODE函数将每个exp2与exp1比较。如果exp1等于expression2，或expression1和exp2都为空，则返回后面的结果表达式。如果没有匹配exp的exp2，则返回其他表达式的值；否则返回一个空值；它类似于CASE表达式（除了对NULL的处理之外）
DECRYPT_BIN	返回一个值，它是加密数据的解密结果
DECRYPT_CHAR	返回一个值，它是加密数据的解密结果
DEGREES	返回从以弧度表示的参数转换来的角度数
DEREF	返回参数的目标类型的一个实例
DIFFERENCE	返回一个从0～4的值，它表示基于对字符串应用SOUNDEX函数的两个字符串的声音之间的差别。值为4表明最佳声音匹配
DIGITS	返回数值的字符串表示形式
DLCOMMENT	从一个DATALINK值返回注释值，如果它存在
DLLINKTYPE	从一个DATALINK值返回链接类型值
DLURLCOMPLETE	从一个具有URL链接类型的DATALINK值返回数据位置属性
DLURLPATH	从一个具有某种URL链接类型的DATALINK值返回访问给定服务器内某个文件所必需的路径和文件名
DLURLPATHONLY	从一个具有某种URL链接类型的DATALINK值返回访问给定服务器中某个文件所必需的路径
DLURLSCHEME	从一个具有某种URL链接类型的DATALINK值返回模式
DLURLSERVER	从一个具有某种URL链接类型的DATALINK值返回文件服务器
DLVALUE	返回一个DATALINK值
DOUBLE	如果参数是一个数值表达式字符串，则返回对应于一个数值的一个浮点数值，或者如果参数是一个字符串表达式，则返回一个数值表示
ENCRYPT	返回一个值，它是一个数据串表达式的加密结果
EVENT_MON_STATE	返回一个事件监视器的当前状态
EXP	返回*n*的指数值
FLOAT	返回一个数值的浮点表示

www.TopSage.com

大学网 TopSage.com

续表

函数名	简要说明
FLOOR	返回小于等于 *n* 的最大整数
GETHINT	如果在加密数据中找到一个密码提示，则返回这个密码提示
GENERATE_UNIQUE	返回一个 13 个字节长的位数据字符串（CHAR(13) FOR BIT DATA），当与同一函数的任何其他执行进行比较时，这个位数据字符串是唯一的
GRAPHIC	返回一个图形串类型的一个 GRAPHIC 表示
HASHEDVALUE	返回行的分布映射索引（0～4095）
HEX	返回作为一个字符串的一个值的十六进制表示
HOUR	返回一个值的小时部分
IDENTITY_VAL_LOCAL	返回为一个标识列最近分配的值，这里发生的赋值是利用 VALUES 子句的一个的单行 INSERT 语句的结果
INSERT	返回一个字符串，这里，expression3 字节已经从 expression1 的在 expression2 开始的位置被删除，而 expression4 已经被插入到 expression1 中从 expression2 开始的位置
INTEGER	返回一个数值或一个整数常量形式的字符串的一个整数表示
JULIAN_DAY	返回一个整数值，它表示公元前 4712 年 1 月 1 日（儒略日期历法的开始）到参数中指定日期值之间的天数
LOWER	把一个字符串中的所有字符转换成小写
LCASE	把一个字符串中的所有字符转换成小写
LEFT	返回从左边开始的字符数 *n*
LENGTH	返回一个字符串中的字符数
LN	返回参数的自然对数
LOCATE	返回字符串中一个子串出现的一个位置
LOG	返回参数的自然对数（与 LN 相同）
LOG10	返回参数的以 10 为底的对数
LONG_VARCHAR	返回一个字符串数据类型的一个 LONG VARCHAR 表示
LONG_VARGRAPHIC	返回一个双字节字符串的一个 LONG VARGRAPHIC 表示
LTRIM	截断字符串前面的空格
MICROSECOND	返回一个值的微秒部分
MIDNIGHT_SECONDS	返回一个整数值，它表示在午夜和在参数中指定的时间值之间的秒数
MINUTE	返回一个值的分钟部分
MOD	返回 *n* 除以 *m* 的余数
MONTH	返回一个值的月份部分
MONTHNAME	返回一个大小写混合的字符串，它包含基于启动数据库所在地区的参数的月份部分的月份名
MQPUBLISH	返回发布到 MQSeries 的数据
MQREAD	从接收服务指定的 MQSeries 位置返回一条消息。读取是无损的——即消息本身仍保持在队列中
MQRECEIVE	同上，但队列中的消息会被删除
MQSEND	使用由服务策略定义的服务政策质量，把包含在消息数据中的数据发送到由发送服务指定的 MQSeries 位置
MQSUBSCRIBE	用于注册某个指定主题发布的 MQSeries 消息
MQUNSUBSCRIBE	用于取消一个现有消息订阅
MULTIPLY_ALT	作为一个十进制值返回两个参数的乘积
NODENUMBER	返回行的分区号
NULLIF	如果参数相等，返回一个 NULL 值，否则，返回第一个参数的值
NVL	NVL 函数返回不为空的第一个参数；它与函数 COALESCE 等价
OCTET_LENGTH	以 8 位字节返回一个表达式的长度
PARTITION	返回在行的分区键值上应用分区函数获得的行的分区映射索引
POSITION(arg1,arg2)	返回字符串 argument1 中字符串/字符 argument2 的起始位置
POSSTR	返回字符串中一个子串的一个出现位置。POSSTR 测试区分大小写敏感
POWER	返回 *m* 的 *n* 次幂的值
QUARTER	返回一个 1～4 之间的整数值，它表示参数中指定的日期的季度
RADIANS	返回从一个参数转换来的弧度数，用角度表示

www.TopSage.com

续表

函数名	简要说明
RAISE_ERROR	使用指定的 SQLSTATE、SQLCODE 和诊断字符串让包含这个函数的语句返回一个错误
RAND	将参数用作可选的种子值，返回一个 0～1 的随机浮点值
REAL	返回一个数值的单精度浮点表示
REC2XML	返回一个用 XML 标记格式化的且包含列名和列数据的字符串
REPEAT	返回重复 *n* 次的 string1
REPLACE	用 expression3 替换 expression1 中出现的所有 expression2
RIGHT	返回一个由 expression1 中最右边的 expression2 个字节组成的字符串
ROUND	返回数值 *n* 四舍五入到 *m* 小数位的值
RTRIM	返回删除了结尾空白的参数字符
SECLABEL	返回一个未命名的安全标签
SECLABEL_BY_NAME	返回一个特定的安全标签
SECLABEL_TO_CHAR	接受一个安全标签作为一个参数并返回一个包含全部元素的字符串
SECOND	返回一个时间值/表达式的秒部分
SIGN	返回参数符号的一个指示符。如果参数小于 0，则返回-1；如果参数等于 0，则返回 0；如果参数大于 0，则返回 1
SIN	返回参数的正弦，这里参数是一个以弧度表示的角度
SINH	返回参数的双曲正弦；参数是一个以弧度表示的角度
SMALLINT	返回一个数值或以小整数常量形式表示的字符串
SOUNDEX	返回一个四字符代码，它表示参数中单词的发音
SPACE	返回一个 *n* 空格的串
SQRT	返回参数的平方根
STRIP	从一个字符串表达式中删除开头或结尾的空白（默认），或者其他指定的开头或结尾字符
SUBSTR[ING]	返回一个字符串的一部分，对于 *m* 个字符的长度从第 *n* 个字符开始
TABLE_NAME	返回在解析任何别名链之后找到的对象的未限定名称
TABLE_SCHEMA	返回在解析任何别名链之后找到的对象的模式名称
TAN	返回参数的正切，这里参数是一个以弧度表示的角度
TANH	返回参数的双曲正切；参数是一个以弧度表示的角度
TIME	从一个值返回一个时间
TAMESTAMP	从一个值或一个值对返回一个时间戳
TIMESTAMP_FORMAT	从一个字符串（argument1）返回一个时间戳，已经使用一个格式模板（argument2）解释了这个字符串
TIMESTAMP_ISO	返回基于日期、时间或时间戳参数的一个时间戳值
TIMESTAMPDIFF	基于两个时间戳之间的不同，返回由第一个参数定义的类型的一个预计的间隔数
TO_CHAR	返回一个时间戳的字符表示
TO_DATE	从一个字符串返回一个时间戳
TRANSLATE	把 string2 中出现的所有 string1 替换成 string3
TRUN[CATE]	返回截取到 *m* 小数位的 *n*
TYPE_ID	返回表达式的动态数据类型的内部类型标识符
TYPE_NAME	返回表达式的动态数据类型的内部类型的未限定名称
TYPE_SCHEMA	返回表达式的动态数据类型的内部类型的模式名称
UCASE	返回一个被转换成大写的字符串
UPPER	返回一个被转换成大写的字符串
VALUE	返回第一个不是 NULL 的参数
VARCHAR	返回一个字符串、日期时间值或图形串的一个可变长度字符串表示
VARCHAR_FORMAT	返回一个时间戳（argument1）的字符表示，且按照一个模板（argument2）格式化
VARGRAPHIC	返回一个字符串值的一个图形字符串表示，单字节字符被转换成双字节字符；或者如果第一个参数是任何图形串类型，则返回一个图形字符串值
WEEK	返回参数的年的星期，且参数是一个 1～54 的整数值。星期从星期日开始
WEEK_ISO	返回参数的年的星期，且参数是一个 1～53 的整数值
YEAR	返回一个值的年部分

www.TopSage.com

表 G-14 IBM DB2 9.5 内置列函数

函数名	简要说明
AVG	返回一个数值集合的平均值
CORR[ELATION]	返回一个数值对集合的相关系数
COUNT	返回一个行或值的集合中的行或值的数目
COUNT_BIG	返回一个行或值的集合中的行或值的数目。它类似于 COUNT，只是结果可以大于最大整数值
COVARIANCE	返回一个数值对集合的（总体）协方差
GROUPING	结合 GROUP BY 子句，返回一个值，这个值指明一个在 GROUP BY 集合中返回的行是否是由一个排除了表达式表示的列的分组集合所生成的一行
MAX	返回一个值集合中的最大值
MIN	返回一个值集合中的最小值
STDDEV	返回一个数值集的标准偏差
XMLACG	这个聚合函数返回一个 XML 序列，其中为 XML 值集中的每个非空值包含一个项目
SUM	返回一个数值集的和
VAR[IANCE]	返回一个数值集的方差

表 G-15 IBM DB2 9.5 内置表函数

函数名	简要说明
MQREADALL	使用服务策略的质量，从接收服务指定的 MQSeries 位置返回一个包含消息和消息元数据的表
MQRECEIVEALL	从接收服务指定的 MQSeries 位置返回一个包含消息和消息元数据的表。读取是无损的，即消息本身会保持在队列中
SQLCACHE_SNAPSHOT	返回一个 DB2 动态 SQL 语句缓存的快照结果

表 G-16 IBM DB2 9.5 内置 XML 函数

函数名	简要说明
XMLAGG	返回 XML 值集中每个非空值的项目的 XML 序列
XMLATTRIBUTES	从参数中构造 XML 属性
XMLCOMMENT	返回一个 XML 值，它具有一个单独的 XQuery 注释节点，用输入参数作为其内容
XMLCONCAT	返回一个序列，它包含 XML 输入参数的一个变量数的连接
XMLDOCUMENT	返回一个 XML 值，具有一个包含 0 个或多个子节点的单个 XQuery 文档节点
XMLELEMENT	返回一个 XML 值，它是一个 XML 元素节点
XMLFOREST	返回一个 XML 值，它是一个 XML 节点的序列
XMLNAMESPACES	从参数中构造命名空间声明
XMLPARSE	将参数解析为一个 XML 文档并返回一个 XML 值
XMLPI	返回一个 XML 值，它具有一个单独的 XQuery 处理指令节点
XMLQUERY	使用指定的输入参数作为 XQuery 变量从一个 XQuery 表达式的求值中返回一个 XML 值
XMLSERIALIZE	从参数中返回生成的指定数据类型的一个序列化的 XML 值
XMLTABLE	从 XQuery 表达式的求值中返回一个表，可能使用指定的输入参数作为 XQuery 变量。行 XQuery 表达式结果序列中的每个序列项表示结果表中的一行
XMLTEXT	返回一个 XML 值，它具有将输入参数作为其内容的一个 XQuery 文本节点
XMLVALIDATE	返回输入 XML 值的一个副本，这个值具有从 XML 架构验证中得到的信息，包括默认值和类型注释
XMLXPROBJECTID	返回 XML 架构的一个 XSR 对象标识符，这个架构验证作为一个参数来传递的文档

表 G-17 IBM DB2 9.5 内置过程

函数名	简要说明
GET_ROUTINE_SAR	获取必要的信息，以便在运行于相同操作系统相同级别的另一个数据库服务器上安装相同的例程
PUT_ROUTINE_SAR	传递必要的文件，在服务器上创建一个 SQL 例程并定义这个例程

表 G-18 Microsoft SQL Server 2008 内置字符串函数

函数名	简要说明
ASCII	返回表达式中第一个字符的 ASCII 代码
CHAR	返回 ASCII 代码的字符
CHARINDEX	返回表达式在另一个表达式中第一次出现的第一个位置
DIFFERENCE	返回两个 SOUNDEX 表达式之间的整数差

www.TopSage.com

大家网 TopSage.com

续表

函数名	简要说明
LEFT	返回从一个指定字符到左边的字符串部分
LEN	返回表达式的字符数，不包含结尾空格
LOWER	返回一个全部字符已转换为小写的表达式
LTRIM	返回一个没有左结尾空格的表达式
NCHAR	从代码数返回一个 Unicode 字符
PATINDEX	返回指定表达式中一个式样第一次出现的起始位置
REPLACE	返回使用第三个表达式替换第一个表达式中所有出现的第二个表达式后的字符串
QUOTENAME	返回添加了用分隔符进行验证 Unicode 表达式
REPLICATE	返回由第一个参数重复 *n* 次所组成的表达式
REVERSE	返回字符表达式的反转
RIGHT	返回从一个指定字符开始到右边的表达式部分
RTRIM	返回截去所有结尾空格的一个字符串
SOUNDEX	返回由四字符码来评价两个表达式发音的相似性
SPACE	返回由空格重复 *n* 次组成的字符串
STR	返回数值数据类型的字符数据
STUFF	删除指定的字符数，并在指定点插入另一组字符
SUBSTRING	返回一个字符串从一个指定点开始跨越一个指定字符数的部分
UNICODE	返回表达式中第一个字符的一个 Unicode 整数码
UPPER	返回已转换为大写字母的一个字符串

表 G-19 Microsoft SQL Server 2008 内置数学函数

函数名	简要说明
ABS	返回表达式的绝对值
ACOS	返回给定余弦的以弧度表示的角度
ASIN	返回给定正弦的以弧度表示的角度
ATAN	返回给定正切的以弧度表示的角度
ATN2	返回以弧度表示的角度，其正切位于两个给定浮点值之间
CEILING	返回大于等于所给表达式的最小整数
COS	从给定的角度（以弧度表示）返回余弦
COT	从给定的角度（以弧度表示）返回余切
DEGREES	从给定的表达式（以弧度表示）返回一个角度
EXP	返回表达式的指数值
FLOOR	返回小于等于给定表达式的最大整数
LOG	返回给定表达式的自然对数
LOG10	返回给定表达式的以 10 为底的对数
PI	返回 PI（3.1415926535897931…）
POWER	返回给定表达式乘指定次方的值
RADIANS	用角度值返回弧度值
RAND	返回 0～1 之间的随机浮点数
ROUND	返回一个四舍五入为指定长度或精度的数值表达式
SIGN	对正的表达式返回+1，对 0 返回 0，对负的表达式返回-1
SIN	从给定角度（以弧度表示）返回正弦
SQUARE	返回表达式的平方值
SQRT	返回给定表达式的平方根
TAN	返回给定表达式的正切

表 G-20 Microsoft SQL Server 2008 内置日期和时间函数

函数名	简要说明
DATEADD	基于传递的值加上一个给定的间隔返回一个新的日期时间值
DATEDIFF	返回传递的两个日期之间的时间单位数（秒、日、年等）

www.TopSage.com

续表

函数名	简要说明
DATENAME	返回表示日期的一个指定日期部分的字符串
DATEPART	返回表示指定日期部分的一个整数
DAY	返回表示日期的天部分的一个整数
GETDATE	返回当前系统的日期和时间
GETUTCDATE	返回当前 UTC 时间的日期时间值
MONTH	返回表示日期月份的一个整数
YEAR	返回表示日期年份的一个整数

表 G-21 Microsoft SQL Server 2008 内置聚合函数

函数名	简要说明
AVG	返回组中值的平均值；忽略所有 NULL
BINARY_CHECKSUM	返回对表中的一行或表达式列表计算的二进制校验值
CHECKSUM	返回对表中的一行或表达式列表计算的校验值
CHECKSUM_AGG	返回一个组中值的校验值
COUNT	返回选定行或输入值的数目
COUNT_BIG	返回选定行或输入值的数目
GROUPING	输出一个附加列，其值为 1 或 0
MIN	返回最小的输入值
MAX	返回最大的输入值
STDEV	返回表达式中所有值的统计标准偏差
STDEVP	返回表达式中所有值的总体统计标准偏差
SUM	返回输入值的和
VAR	返回给定表达式中所有值的统计方差
VARP	返回给定表达式中所有值中 *s* 的一个总体统计方差

表 G-22 Microsoft SQL Server 2008 内置文本和图像函数

函数名	简要说明
PATINDEX	返回在表达式中指定模式第一次出现的起始位置
TEXTPTR	以 varbinary 格式返回对应于 text、ntext 或 image 的文本指针
TEXTVALID	返回检查 text、ntext 或 image text-pointer 是否有效的结果

表 G-23 Microsoft SQL Server 2008 内置游标函数

函数名	简要说明
@@CURSOR_ROWS	返回最后打开的游标中的行数
@@FETCH_STATUS	返回当前会话中任何游标所在的最后 FETCH 语句的状态
CURSOR_STATUS	一个标量函数，表示对于一个给定参数，该过程是否已返回一个游标和一个结果集

表 G-24 Microsoft SQL Server 2008 内置元数据函数

函数名	简要说明
ASSEMBLYPROPERTY	返回有关汇编（.NET）属性的信息
COL_LENGTH	返回一个列的定义长度
COL_NAME	从列 ID 返回列名
COLUMNPROPERTY	返回有关一个列或一个过程的一个参数的信息
DATABASEPROPERTY	返回给定数据库的属性值
DATABASEPROPERTYEX	返回指定属性的当前设置
DB_ID	返回数据库标识号
DB_NAME	返回当前数据库名
FILE_ID	返回给定逻辑文件的文件标识号
FILE_IDEX	返回当前数据库的数据、日志或全文文件的指定逻辑文件名的标识号（Transact-SQL）
FILE_NAME	返回给定文件标识号的文件名

www.TopSage.com

续表

函数名	简要说明
FILEGROUP_ID	返回给定文件组的文件标识号
FILEGROUP_NAME	返回给定标识号的文件组名
FILEGROUPPROPERTY	对于给定的文件组，返回指定的文件组属性值
FILEPROPERTY	对于给定的文件，返回指定的文件属性值
FULLTEXTCATALOGPROPERTY	返回有关全文目录的信息
FULLTEXTSERVICEPROPERTY	返回有关全文服务级别属性的信息
INDEX_COL	返回索引列名称
INDEXDEY_PROPERTY	返回有关索引键的信息
INDEXPROPERTY	从给定的索引名和表ID返回属性值
OBJECT_ID	返回数据库对象的ID给定名称
OBJECT_NAME	根据ID返回数据库对象名
OBJECTPROPERTY	返回当前数据库中对象的有关信息
OBJECTPROPERTYEX	返回当前数据库在架构范围内的对象的有关信息
@@PROCID	返回当前过程的存储过程标识符
RYPEPROPERTY	返回有关数据类型的信息
SQL_VARIANT_PROPERTY	返回有关除其他属性信息之外的基本数据类型的信息
FN_LISTEXTENDEDPROPERTY	返回数据库对象的扩展属性值

表G-25 Microsoft SQL Server 2008内置配置函数

函数名	简要说明
@@CONNECTIONS	返回已打开或尝试过的连接数
@@DATEFIRST	返回SET DATEFIRST参数值
@@DBTS	返回datestamp数据类型的当前值
@@LANGUAGE	返回当前会话/数据库使用的语言名
@@LANGID	返回当前会话/数据库所用的语言ID
@@LOCK_TIMEOUT	返回锁定超时值，单位为秒
@@MAX_CONNECTIONS	返回同时用户连接的最大数
@@MAX_PRECISION	返回数值数据类型所用的精度级别
@@NESTLEVEL	返回当前存储过程的嵌套级别
@@OPTIONS	返回当前SET选项的位掩码信息
@@REMSERVER	返回远程服务器名称
@@SPID	返回当前过程/进程的标识符号（ID）
@@SERVERNAME	返回本地服务器名称
@@SERVICENAME	返回正在运行的SQL Server实例的注册表键名
@@TEXTSIZE	返回TEXTSIZE选项的当前值
@@VERSION	返回SQL Server当前版本的日期、版本和处理器类型

表G-26 Microsoft SQL Server 2008内置安全函数

函数名	简要说明
CURRENT_USER	返回当前用户的名称；它等价于USER_NAME
sys.fn_buildin_permissions	返回某个用户的内置权限的层次结构的描述
has_perms_by_name	确定当前用户是否具有权限
IS_MEMBER	表明当前用户是否是一个Windows NT组或SQL Server角色的一个成员
IS_SRVROLEMEMBER	指明当前的登录是否是指定角色的成员
PERMISSIONS	返回语句的位图指示、对象或当前用户的列权限
SHCEMA_ID	返回一个架构名的架构ID
SCHEMA_NAME	返回一个架构ID的架构名
SESSION_USER	返回当前数据库的当前上下文的用户名
SETUSER	对于sysadmin或db_owner固定角色允许模仿另一个用户；将被取代——推荐使用EXECUTE AS语句

www.TopSage.com

TopSage.com

续表

函数名	简要说明
SUSER_ID	返回当前用户的登录ID
SUSER_NAME	返回当前用户的登录名
SUSER_SID	返回登录名的用户安全标识号（SID）
SUSER_SNAME	返回安全标识号（SID）的用户登录名
USER_ID	返回用户名的用户数据库标识号
USER_NAME	返回标识号的数据库用户名
USER	返回当前用户的数据库名
HAS_DBACCESS	表明当前用户是否可以访问指定的数据库
fn_trace_geteventinfo	返回有关要跟踪的事件信息
fn_trace_getfilterinfo	返回有关应用于指定跟踪的筛选器的信息
fn_trace_getinfo	返回有关跟踪的信息
fn_trace_gettable	返回表格式的跟踪信息

表G-27 Microsoft SQL Server 2008内置系统函数

函数名	简要说明
APP_NAME	返回当前会话的应用程序名（如果进行了设置）
CASE	计算条件列表并返回一个值
CAST	显式地把一种数据类型转换为另一种数据类型
COALESCE	返回列表上第一个非空表达式
CONVERT	显式地把一种数据类型转换成另一种数据类型，作用类似于CAST函数
COLLATIONPROPERTY	返回一个指定排序规则的属性
COLUMNS_UPDATED	返回varbinary位式样，它指示在表或视图中更新或插入了哪些列。可以在INSERT和UPDATE语句内使用
CURRENT_TIMESTAMP	返回当前的日期和时间；等价于GETDATE()
CURRENT_USER	返回当前的用户。此函数等价于USER_NAME()
DATALENGTH	返回一个表达式中的字节数
@@ERROR	返回最后的Transact-SQL语句的错误号
ERROR_LINE	返回在TRY...CATCH块内出现、捕获的错误所在的行号
ERROR_MESSAGE	返回在TRY...CATCH块内捕获的错误的消息
ERROR_PROCEDURE	对于TRY...CATCH块内捕获的错误返回存储过程名
ERROR_SEVERITY	对于TRY...CATCH块内捕获的错误返回严重程度
ERROR_STATE	对于TRY...CATCH块内捕获的错误返回状态号
FORMATMESSAGE	对sysmessages表中的一个现有消息进行格式化
GETANSINULL	返回会话及数据库的默认NULL性
HOST_ID	返回计算机的ID
HOST_NAME	返回主机名
IDENT_INCR	返回一个表或视图中的任意标识列的增量值
IDENT_SEED	返回一个表或视图中的任意标识列的标识种子值
@@IDENTITY	返回最后插入的标识值
IDENTITY	用于插入到一个标识列
ISDATE	确定一个表达式是否是一个有效的日期类型（或者可以转换成一个有效的日期类型）
ISNULL	确定表达式是否为NULL
ISNUMERIC	确定表达式是否为数值
NEWID	返回UNIQUEIDENTIFIER数据类型的一个唯一值
NULLIF	如果两个表达式等价，则返回NULL
PARSENAME	返回对象名的指定路径
ORIGINAL_LOGIN	返回建立了到SQL Server连接的登录名，即使发生多个上下文切换也是如此
PERMISSIONS	返回一个针对位图的值，它指定当前用户及对象的权限
@@ROWCOUNT	返回受最后语句影响的行数

www.TopSage.com

续表

函数名	简要说明
ROWCOUNT_BIG	返回受最后语句影响的行数（bigint）
SESSION_USER	返回在没有指定默认值时要插入到一个表中的用户名
STATS_DATE	返回最后一次更新指定索引统计的日期
SYSTEM_USER	返回具有管理员特权（Window 身份验证）的当前用户名
@@TRANCOUNT	返回当前会话挂起的事务数
USER_NAME	返回一个给定标识号的数据库用户名
COLLATIONPROPERTY	返回一个给定排序规则的属性
SCOPE_IDENTITY	返回插入到当前作用域的标识列内的最后一个标识值
SERVERPROPERTY	返回指定服务器属性值
SESSIONPROPERTY	返回指定会话属性值
SESSION_USER	返回当前数据库中当前上下文中的用户名
STATS_DATE	返回更新了指定索引的统计信息时的日期
sys.dm_db_index_physical_stats	返回指定表/视图的数据和索引的大小及详细信息
UPDATE	返回布尔值，指示是否对视图/表中的特定列执行了 UPDATE 或 INSERT
USER_NAME	返回特定 ID 的数据库用户名
XACT_STATE	返回当前请求的用户事务状态
fn_helpcollation	返回支持的排序规则的列表
fn_servershareddrives	返回群集服务器可以使用的共享驱动器名称
fn_virtualservernodes	返回一个可以运行虚拟服务器的节点列表

表 G-28 Microsoft SQL Server 2008 内置系统统计函数

函数名	简要说明
@@CONNECTIONS	返回从 SQL Server 启动以来的连接数（无论成功还是失败）
@@CPU_BUSY	返回从 SQL Server 启动以来的 CPU 工作时间（以毫秒为单位）
@@IDLE	返回从 SQL Server 启动以来的空闲时间（以毫秒为单位）
@@IO_BUSY	返回从 SQL Server 启动以来的输入和输出操作所花费的时间（以毫秒为单位）
@@PACK_RECEIVED	返回从 SQL Server 启动以来接收的输入数据包数目
@@PACK_SENT	返回从 SQL Server 启动以来接收的输出数据包数目
@@PACKET_ERRORS	返回从 SQL Server 启动以来在网络上发生错误的数据包数目
@@TIMETICKS	返回每个 CPU 单位的毫秒数
@@TOTAL_ERRORS	返回从 SQL Server 启动以来的磁盘读/写错误数
@@TOTAL_READ	返回从 SQL Server 启动以来读取物理磁盘的次数
@@TOTAL_WRITE	返回从 SQL Server 启动以来写入物理磁盘的次数
fn_virtualfilestats	返回对数据库文件的 I/O 统计

表 G-29 Microsoft SQL Server 2008 内置行集/XML 函数

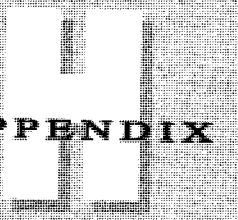
函数名	简要说明
CONTAINSTABLE	返回某些列组成的一个表，这些列包含针对一个单词或词组的精确或模糊匹配的基于字符的数据类型
FREETEXTTABLE	返回某些列组成的一个表：这些列包含针对那些匹配含义而不是确切单词的值的基于字符的数据类型
OPENQUERY	对一个链接的 OLEDB 数据源执行一个指定的传递查询
OPENROWSET	提供一个通过 OLEDB 连接和访问远程数据的即时方法
OPENDATASOURCE	不使用链接服务器，而提供临时连接信息，并作为一个四部分对象名的一部分
OPENXML	提供 XML 文档的一个行集视图

表 G-30 Microsoft SQL Server 2008 内置秩函数

函数名	简要说明
RANK[OVER]	返回一个结果集分区内的每一行的秩
DENSE_RANK[OVER]	返回一个结果集分区内的每一行的秩，包括已压缩的间隔
NTILE(n)[OVER]	把一个有序分区中的行分发到指定数量的分组中
ROW_NUMBER	返回在一个结果集分区内的一行的顺序；每个分区中的第一行都从 1 开始

www.TopSage.com

附录 H
SQL 语法参考



本附录提要：

- DDL 语句
- DCL 语句
- DML 语句
- DQL 语句
- 事务控制语句
- 语法

SQL:2003 语法使用 BNF（Backus_Naur Form）符号来说明 SQL 的标准语法。

注意：这种符号是以 John Backus 和 Peter Naur 命名的，他们首次引入一种正式符号来描述一种给定语言的语法。

这种符号非常复杂，因此在本书我们只使用最重要的元素来进行简化处理。此外，因为“三大”数据库软件的语法之间存在着显著差异（而且也与 SQL:2003 语法不同），所以我们的目的是提供一种尽可能通用的符号，它们能应用到所有三大软件。对于本附录中没有提及的另外功能，请参考供应商特定文档。

表 H-1 中列出了我们使用的符号及其含义。

表 H-1　　简化的 BNF 符号

符号	含义
<>	封闭术语名
|	分隔可选项（除了 OR）
[]	指定可选术语
{}	表明至少需要一个术语
,...	表明可选择的重复使用术语

此外，全部 SQL 关键字都是大写字母；其他则为小写。

H.1　DDL 语句

这部分提供了 DDL（Data Definition Language，数据定义语言）语句的 BNF 符号。

H.1.1　表

下面的符号分别用于创建、修改和删除数据库表：

```
CREATE TABLE <table_name>
(
 <column_name> <datatype> [<column_constraint>,...]
    [DEFAULT <default_value>],...
[<table_constraint>,...]
[physical_options]
)

ALTER TABLE <table_name>
{
 <vendor_specific_add_column_clause> |
 <vendor_specific_alter_column_clause> |
 <vendor_specific_add_constraint_clause> |
 <vendor_specific_drop_constraint_clause>
}
```

www.TopSage.com

大家网 TopSage.com

Note 注意：ALTER TABLE 语句子句随不同的实现方式而变化并且不能泛化。参看第 5 章了解更多信息。

```
DROP TABLE <table_name>
```

H.1.2 索引

下面两个符号用于创建和删除数据库索引：

```
CREATE [UNIQUE] INDEX <index_name>
ON <table_name> (<column_name> [ASC|DESC],...)

DROP INDEX <index_name>
```

H.1.3 视图

下面的符号分别用于创建、修改和删除数据库视图：

```
CREATE VIEW <view_name> [(column_name,...)]
AS <select_statement>
[WITH CHECK OPTION]
ALTER VIEW <view_name>
<vendor_specific_alter_view_clause>
DROP VIEW <view_name>
```

H.1.4 架构

下面两个符号用于创建和删除数据库架构：

```
CREATE SCHEMA <schema_name>
AUTHORIZATION <authorization_id>
<create_object_statement>,...
<grant_privilege_statement>,...
```

Note 注意：在 Oracle 中，schema_name 符号是无效的。只能用自己的架构在 Oracle 中创建架构，并且只能使用自己的 authorization_id。

```
DROP SCHEMA <schema_name> RESTRICT
```

Note 注意：前面所述的语法只用于 DB2；Oracle 和 Microsoft SQL Server 在其语法中没有 DROP SCHEMA。

H.1.5 存储过程

创建一个存储过程的 BNF 符号如下：

```
CREATE PROCEDURE <procedure_name> [<parameter_section>]
<procedure_definition>
```

Note 注意：前面所述的规范是通用的；实际的语法与实现有关。参考供应商特定文档了解详细内容。

下面的符号用于删除一个存储过程：

```
DROP PROCEDURE <procedure_name>
```

H.1.6 用户定义函数

创建一个用户定义函数的 BNF 符号如下：

```
CREATE FUNCTION <function_name>
<function_definition_includes_return_statement>
```

Note 注意：前面所述的规范是通用的；实际的语法与实现有关。参考供应商特定文档了解详细内容。

下面的符号用于删除一个用户定义函数：

```
DROP FUNCTION <function_name>
```

H.1.7 触发器

创建一个触发器的 BNF 符号如下：

```
CREATE TRIGGER <trigger_name>
[BEFORE | AFTER]
{INSERT | UPDATE | DELETE}
ON <table_name>
```


www.TopSage.com

```
[FOR EACH ROW]
<trigger_body>
```

Note 注意：前面所述的语法只说明了基本的触发器功能；实际的实现具有更多选项。

下面的符号用于删除一个触发器：

```
DROP TRIGGER [qualifier.]<trigger_name>
```

H.2 DCL 语句

主要的两个 DCL（Data Control Language，数据控制语言）语句的 BNF 符号是：

```
GRANT {[ALL [PRIVILEGES]] | <privilege,...>}
[ON <object_name>]
TO <user_group_or_role>
[WITH GRANT OPTION]
REVOKE {[ALL [PRIVILEGES]] | <privilege,...>}
ON <object_name>
FROM <user_group_or_role>
```

H.3 DML 语句

DML（Data Manipulation Language，数据操纵语言）语句的 BNF 符号是：

```
INSERT INTO <table_or_view_name>
[(<column_name>,...)]
{{VALUES (<literal> |
          <expression> |
          NULL |
          DEFAULT,...)} |
 {<select_statement>} }
UPDATE <table_or_view_name>
SET {<column_name> = <literal> |
                     <expression> |
                     <single_row_select_statement> |
                     NULL |
                     DEFAULT,...}
[WHERE <predicate>]
DELETE FROM <table_or_view_name>
WHERE <predicate>
```

H.4 DQL 语句

数据查询语言（DQL，Data Query Language）只由 SELECT 语句组成。一个 SELECT 语句可以是单表（只从一个表选择记录）或多表（从多个表选择行，通常使用某种连接）。

H.4.1 单表 SELECT

创建单表 SELECT 的 BNF 符号如下：

```
SELECT [ALL | DISTINCT]
{[<qualifier>.]<column_name> | * | <expression>}
   [AS <column_alias>],...
FROM <table_or_view_name> | <inline_view>
   [<table_alias>]
[WHERE <predicate>]
[GROUP BY [<qualifier>.]<column_name>,...
   [HAVING <predicate>]]
[ORDER_BY [<qualifier>.]<column_name> | <column_number>
   [ASC | DESC],...];
```

H.4.2 多表 SELECT

既可以使用“新”语法也可以使用“旧”语法创建一个多表 SELECT；对于内部和外部连接的语法也稍有不同。

www.TopSage.com

Inner join

```
SELECT [ALL | DISTINCT]
{[<qualifier>.]<column_name> | * | <expression>}
    [AS <column_alias>],...
FROM <table_or_view_name> | <inline_view>
    [<table_alias>]
    [INNER | NATURAL | CROSS] JOIN
      <table_or_view_name> | <inline_view>
    [<table_alias>]
    [ON [<qualifier>.]<column_name>
           <join_condition>
        [<qualifier>.]<column_name>]
    [[INNER | NATURAL | CROSS] JOIN
      <table_or_view_name> | <inline_view>
    [<table_alias>]
    [ON [<qualifier>.]<column_name>
          <join_condition>
          [<qualifier>.]<column_name>],...]
    [WHERE <predicate>]
    [GROUP BY [<qualifier>.]<column_name>,...
        [HAVING <predicate>]]
    [ORDER BY [<qualifier>.]<column_name> | <column_number>
        [ASC | DESC],...];
```

Outer join

```
SELECT [ALL | DISTINCT]
{[<qualifier>.]<column_name> | * | <expression>}
  [AS <column_alias>],...
FROM <table or view name> | <inline_view>
   [<table_alias>]
  {[LEFT | RIGHT | FULL [OUTER]] JOIN
     <table_or_view name> | <inline_view>
   [<table_alias>]
  {ON [<qualifier>.]<column_name>
        <join_condition>
      [<qualifier>.]<column_name>}
  [{[LEFT | RIGHT | FULL [OUTER]] JOIN
     <table_or_view_name> | <inline_view>
  [<table_alias>]
 {ON [<qualifier>.]<column_name>
       <join_condition>
     [<qualifier>.]<column_name>},...]
[WHERE <predicate>]
[GROUP BY [<qualifier>.]<column_name>,...
    [HAVING <predicate>]]
[ORDER BY [<qualifier>.]<column_name> | <column_number>
    [ASC | DESC],...];
```

注意：内部和外部连接的语法可以结合为一个单独的查询；即可以使用一个内部连接组合一些表，然后使用一个外部连接将结果集与另一个表连接起来等等。

H.5 事务控制语句

事务控制语句的BNF语法如下：

```
COMMIT [WORK]

ROLLBACK [WORK]
```

H.6 谓词

在简化的BNF中，可以将一个谓词定义为一条能求值为TRUE或FALSE的语句。表H-2“解析”了SQL谓词的语法。

表H-2 谓词

元素	语法
<predicate>	<boolean_term> [{AND | OR} <boowlean_term>,. . .]
<boolean_term>	[NOT] <search_test>

www.TopSage.com

续表

元素	语法
<search_test>	<comparison_test> | <between_test> | <in_test> | <like_test> | <null_test> | <exists_test> | <quantified_comparison_test>
<comparison_test>	<column value> {= | <> | != | < | > | <= | >=} {<expression> | <single_row_subquery>}
<between_test>	<column_value> [NOT] BETWEEN <expression> AND <expression>
<in_test>	<column_value> [NOT] IN <expression_list> | <table_subquery>
<like_test>	<column_value> [NOT] LIKE <pattern> [ESCAPE <value>]
<null_test>	<column_value> IS [NOT] NULL
<exists_test>	EXISTS <table_subquery>
<quantified_comparison_test>	<column_value> {= | <> | != | < | > | <= | >= } {ALL | SOME | ANY} <table_subquery>
<expression_list>	(<expression>,. . .)
<expression>	<sub-expression> {+ | - | * | /} <sub-expression>
<sub-expression>	[- | +] {<value> | <function>}
<value>	<literal> | <variable>
<column_value>	[qualifier.]<column_name>

www.TopSage.com

APPENDIX

附录I
SQL保留关键字

本附录提要：

- 保留关键字列表

具有一套有限的词汇的SQL是一种相对有效的语言（与很多其他编程语言比较）；SQL:2003标准定义了大约300个关键字，而开发商目前为止只实现了一个小的子集。

Oracle 11g列出了100个以上的关键字（并且在PL/SQL和数据库管理命令和标志中使用了更多；从V$RESERVED_WORDS视图中可以获得Oracle 11g中保留关键字的完整列表），IBM DB2 9.5具有超过300个的关键字，而Microsoft SQL Server 2008保留了超过200个关键字。在SQL:2003标准中可以找到大多数开发商的保留关键字，但还有更多。这些保留关键字都不应当用作变量标识符，这样使用会影响SQL代码的可移植性。在一些系统中，这样做将产生一个错误（SQLSTATE 42939）。幸运的是，很多基于GUI的SQL编辑器（例如，Microsoft Query Analyzer、TOAD、WinSQL等）会高亮显示特定术语的关键字。

Note 注意：除了这里列出的关键字之外，每个开发商也有一个针对未来用途的保留关键字列表。这些列表经常更新。参考详细的RDBMS文档。

SQL:2003标准的保留关键字：

```
ABS
ALL
ALLOCATE
ALTER
AND
ANY
ARE
ARRAY
AS
ASENSITIVE
ASYMMETRIC
AT
ATOMIC
AUTHORIZATION
AVG
BEGIN
BETWEEN
BIGINT
BINARY
BLOB
BOOLEAN
BOTH
BY
CALL
CALLED
CARDINALITY
CASCADED
CASE
CAST
CEIL
CEILING
CHAR
CHARACTER
CHAR_LENGTH
CHARACTER_LENGTH
CHECK
CLOB
CLOSE
COALESCE
COLLATE
COLLECT
COLUMN
COMMIT
CONDITION
CONNECT
CONSTRAINT
CONVERT
CORR
CORRESPONDING
COUNT
COVAR_POP
COVAR_SAMP
CROSS
CUBE
CUME_DIST
CURRENT
CURRENT_DATE
CURRENT_DEFAULT
   TRANSFORM_GROUP
CURRENT_PATH
CURRENT_ROLE
CURRENT_TIME
CURRENT_TIMESTAMP
CURRENT_TRANSFORM
  _GROUP_FOR_TYPE
CURRENT_USER
CURSOR
CYCLE
DATE
DAY
DEALLOCATE
DEC
DECIMAL
DECLARE
DEFAULT
DELETE
DENSE_RANK
DEREF
DESCRIBE
DETERMINISTIC
DISCONNECT
DISTINCT
DOUBLE
DROP
DYNAMIC
EACH
ELEMENT
ELSE
END
END-EXEC
ESCAPE
EVERY
EXCEPT
```

www.TopSage.com

EXEC
EXECUTE
EXISTS
EXP
EXTERNAL
EXTRACT
FALSE
FETCH
FILTER
FLOAT
FLOOR
FOREIGN
FREE
FROM
FULL
FUNCTION
FUSION
GROUP
GROUPING
HAVING
HOLD
HOUR
IDENTITY
INDICATOR
INNER
INOUT
INSENSITIVE
INSERT
INT
INTEGER
INTERSECT
INTERSECTION
INTERVAL
INTO
IS
JOIN
LANGUAGE
LARGE
LATERAL
LEADING
LEFT
LIKE
LN
LOCAL
LOCALTIME
LOCALTIMESTAMP
LOWER
MATCH
MAX
MEMBER
MERGE
METHOD
MIN
MOD
MODIFIES
MODULE
MONTH
MULTISET
NATIONAL
NATURAL
NCHAR
NCLOB
NEW
NO
NONE
NORMALIZE
NOT
NULL
NULLIF
NUMERIC
OCTET_LENGTH
OF
OLD
ON
ONLY
OPEN
OR
ORDER
OUT
OUTER
OVER
OVERLAPS
OVERLAY
PARAMETER
PERCENT_RANK
PERCENTILE_CONT
PERCENTILE_DISC
POSITION
POWER
PRECISION
PREPARE
PRIMARY
PROCEDURE
RANGE
RANK
READS
REAL
RECURSIVE
REF
REFERENCES
REFERENCING
REGR_AVGX
REGR_AVGY
REGR_COUNT
REGR_INTERCEPT
REGR_R2
REGR_SLOPE
REGR_SXX
REGR_SXY
REGR_SYY
RELEASE
RETURNS
REVOKE
RIGHT
ROLLBACK
ROLLUP
ROWS
ROW_NUMBER
SAVEPOINT
SCOPE
SCROLL
SEARCH
SECOND
SELECT
SENSITIVE
SESSION_USER
SET
SIMILAR
SMALLINT
SOME
SPECIFIC
SPECIFICTYPE
SQLEXCEPTION
SQLSTATE
SQLWARNING
SQRT
START
STATIC
STDDEV_POP
STDDEV_SAMP
SUBMULTISET
SUBSTRING
SUM
SYMMETRIC
SYSTEM
SYSTEM_USER
TABLE
TABLESAMPLE
THEN
TIME
TIMESTAMP
TIMEZONE_HOUR
TIMEZONE_MINUTE
TO
TRAILING
TRANSLATE
TRANSLATION
TREAT
TRIGGER
TRIM
TRUE
UESCAPE
UNION
UNIQUE
UNKNOWN
UNNEST
UPPER
USER
USING
VALUE
VALUES
VARCHAR
VARYING
VAR_POP
VAR_SAMP
WHEN
WHENEVER
WHERE
WIDTH_BUCKET
WINDOW
WITH
WITHIN
WITHOUT

Oracle 11g 的 SQL 保留关键字：

ACCESS
ABS
ADD
ALL
ALTER
AND
ANY
AS
ASC
AUDIT
BETWEEN
BY
CHAR
CHECK
CLUSTER
COLUMN
COMMENT
COMPRESS
CONNECT
CREATE
CURRENT
DATE
DECIMAL
DEFAULT
DELETE
DESC
DISTINCT
DROP
ELSE
EXCLUSIVE
EXISTS
FILE
FLOAT
FOR
FROM
GRANT
GROUP
HAVING
IDENTIFIED
IMMEDIATE
IN
INCREMENT
INDEX
INITIAL
INSERT
INTEGER
INTERSECT
INTO

www.TopSage.com

大象网 TopSage.com

IS
LEVEL
LIKE
LOCK
LONG
MAXEXTENTS
MINUS
MLSLABEL
MODE
MODIFY
NOAUDIT
NOCOMPRESS
NOT
NOWAIT
NULL
NUMBER
OF
OFFLINE
ON
ONLINE
OPTION
OR
ORDER
PCTFREE
PRIOR
PRIVILEGES
PUBLIC
RAW
RENAME
RESOURCE
REVOKE
ROW
ROWID
ROWNUM
ROWS
SELECT
SESSION
SET
SHARE
SIZE
SMALLINT
START
SUCCESSFUL
SYNONYM
SYSDATE
TABLE
THEN
TO
TRIGGER
UID
UNION
UNIQUE
UPDATE
USER
VALIDATE
VALUES
VARCHAR
VARCHAR2
VIEW
WHENEVER
WHERE
WITH

IBM DB2 9.5 的保留关键字：

ACTIVATE
ADD
AFTER
ALIAS
ALL
ALLOCATE
ALLOW
ALTER
AND
ANY
AS
ASENSITIVE
ASSOCIATE
ASUTIME
AT
ATTRIBUTES
AUDIT
AUTHORIZATION
AUX
AUXILIARY
BEFORE
BEGIN
BETWEEN
BINARY
BUFFERPOOL
BY
CACHE
CALL
CALLED
CAPTURE
CARDINALITY
CASCADED
CASE
CAST
CCSID
CHAR
CHARACTER
CHECK
CLOSE
CLUSTER
COLLECTION
COLLID
COLUMN
COMMENT
COMMIT
CONCAT
CONDITION
CONNECT
CONNECTION
CONSTRAINT
CONTAINS
CONTINUE
COUNT
COUNT_BIG
CREATE
CROSS
CURRENT
CURRENT_DATE
CURRENT_LC_CTYPE
CURRENT_PATH
CURRENT_SCHEMA
CURRENT_SERVER
CURRENT_TIME
CURRENT_TIMESTAMP
CURRENT_TIMEZONE
CURRENT_USER
CURSOR
CYCLE
DATA
DATABASE
DATAPARTITIONNAME
DATAPARTITIONNUM
DATE
DAY
DAYS
DB2GENERAL
DB2GENRL
DB2SQL
DBINFO
DBPARTITIONNAME
DBPARTITIONNUM
DEALLOCATE
DECLARE
DEFAULT
DEFAULTS
DEFINITION
DELETE
DENSE_RANK
DENSERANK
DESCRIBE
DESCRIPTOR
DETERMINISTIC
DIAGNOSTICS
DISABLE
DISALLOW
DISCONNECT
DISTINCT
DO
DOUBLE
DROP
DSSIZE
DYNAMIC
EACH
EDITPROC
ELSE
ELSEIF
ENABLE
ENCODING
ENCRYPTION
END
END-EXEC
ENDING
ERASE
ESCAPE
EVERY
EXCEPT
EXCEPTION
EXCLUDING
EXCLUSIVE
EXECUTE
EXISTS
EXIT
EXPLAIN
EXTERNAL
EXTRACT
FENCED
FETCH
FIELDPROC
FILE
FINAL
FOR
FOREIGN
FREE
FROM
FULL
FUNCTION
GENERAL
GENERATED
GET
GLOBAL
GO
GOTO
GRANT
GRAPHIC
GROUP
HANDLER
HASH
HASHED_VALUE
HAVING
HINT
HOLD
HOUR
HOURS
IDENTITY
IF
IMMEDIATE
IN
INCLUDING
INCLUSIVE
INCREMENT
INDEX
INDICATOR
INHERIT
INNER
INOUT
INSENSITIVE
INSERT
INTEGRITY

www.TopSage.com

INTERSECT
INTO
IS
ISOBID
ISOLATION
ITERATE
JAR
JAVA
JOIN
KEY
LABEL
LANGUAGE
LATERAL
LC_CTYPE
LEAVE
LEFT
LIKE
LINKTYPE
LOCAL
LOCALDATE
LOCALE
LOCALTIME
LOCALTIMESTAMP
LOCATOR
LOCATORS
LOCK
LOCKMAX
LOCKSIZE
LONG
LOOP
MAINTAINED
MATERIALIZED
MAXVALUE
MICROSECOND
MICROSECONDS
MINUTE
MINUTES
MINVALUE
MODE
MODIFIES
MONTH
MONTHS
NEW
NEW_TABLE
NEXTVAL
NO
NOCACHE
NOCYCLE
NODENAME
NODENUMBER
NOMAXVALUE
NOMINVALUE
NONE
NOORDER
NORMALIZED
NOT
NULL
NULLS
NUMPARTS
OBID
OF
OLD
OLD_TABLE
ON
OPEN
OPTIMIZATION
OPTIMIZE
OPTION
OR
ORDER
OUT
OUTER
OVER
OVERRIDING
PACKAGE
PADDED
PAGESIZE
PARAMETER
PART
PARTITION
PARTITIONED
PARTITIONING
PARTITIONS
PASSWORD
PATH
PIECESIZE
PLAN
POSITION
PRECISION
PREPARE
PREVVAL
PRIMARY
PRIQTY
PRIVILEGES
PROCEDURE
PROGRAM
PSID
QUERY
QUERYNO
RANGE
RANK
READ
READS
RECOVERY
REFERENCES
REFERENCING
REFRESH
RELEASE
RENAME
REPEAT
RESET
RESIGNAL
RESTART
RESTRICT
RESULT
RESULT_SET_LOCATOR
RETURN
RETURNS
REVOKE
RIGHT
ROLLBACK
ROUTINE
ROW
ROW_NUMBER
ROWNUMBER
ROWS
ROWSET
RRN
RUN
SAVEPOINT
SCHEMA
SCRATCHPAD
SCROLL
SEARCH
SECOND
SECONDS
SECQTY
SECURITY
SELECT
SENSITIVE
SEQUENCE
SESSION
SESSION_USER
SET
SIGNAL
SIMPLE
SOME
SOURCE
SPECIFIC
SQL
SQLID
STACKED
STANDARD
START
STARTING
STATEMENT
STATIC
STAY
STOGROUP
STORES
STYLE
SUBSTRING
SUMMARY
SYNONYM
SYSFUN
SYSIBM
SYSPROC
SYSTEM
SYSTEM_USER
TABLE
TABLESPACE
THEN
TIME
TIMESTAMP
TO
TRANSACTION
TRIGGER
TRIM
TYPE
UNDO
UNION
UNIQUE
UNTIL
UPDATE
USAGE
USER
USING
VALIDPROC
VALUE
VALUES
VARIABLE
VARIANT
VCAT
VERSION
VIEW
VOLATILE
VOLUMES
WHEN
WHENEVER
WHERE
WHILE
WITH
WITHOUT
WLM
WRITE
XMLELEMENT
YEAR
YEARS

Microsoft SQL Server 2008 的保留关键字：

ADD
ALL
ALTER
AND
ANY
AS
ASC
AUTHORIZATION
BACKUP
BEGIN
BETWEEN
BREAK
BROWSE
BULK
BY
CASCADE
CASE
CHECK
CHECKPOINT
CLOSE
CLUSTERED

COALESCE	GRANT	READTEXT
COLLATE	GROUP	RECONFIGURE
COLUMN	HAVING	REFERENCES
COMMIT	HOLDLOCK	REPLICATION
COMPUTE	IDENTITY	RESTORE
CONSTRAINT	IDENTITY_INSERT	RESTRICT
CONTAINS	IDENTITYCOL	RETURN
CONTAINSTABLE	IF	REVERT
CONTINUE	IN	REVOKE
CONVERT	INDEX	RIGHT
CREATE	INNER	ROLLBACK
CROSS	INSERT	ROWCOUNT
CURRENT	INTERSECT	ROWGUIDCOL
CURRENT_DATE	INTO	RULE
CURRENT_TIME	IS	SAVE
CURRENT_TIMESTAMP	JOIN	SCHEMA
CURRENT_USER	KEY	SECURITYAUDIT
CURSOR	KILL	SELECT
DATABASE	LEFT	SESSION_USER
DBCC	LIKE	SET
DEALLOCATE	LINENO	SETUSER
DECLARE	LOAD	SHUTDOWN
DEFAULT	NATIONAL	SOME
DELETE	NOCHECK	STATISTICS
DENY	NONCLUSTERED	SYSTEM_USER
DESC	NOT	TABLE
DISK	NULL	TABLESAMPLE
DISTINCT	NULLIF	TEXTSIZE
DISTRIBUTED	OF	THEN
DOUBLE	OFF	TO
DROP	OFFSETS	TOP
DUMP	ON	TRAN
ELSE	OPEN	TRANSACTION
END	OPENDATASOURCE	TRIGGER
ERRLVL	OPENQUERY	TRUNCATE
ESCAPE	OPENROWSET	TSEQUAL
EXCEPT	OPENXML	UNION
EXEC	OPTION	UNIQUE
EXECUTE	OR	UNPIVOT
EXISTS	ORDER	UPDATE
EXIT	OUTER	UPDATETEXT
EXTERNAL	OVER	USE
FETCH	PERCENT	USER
FILE	PIVOT	VALUES
FILLFACTOR	PLAN	VARYING
FOR	PRECISION	VIEW
FOREIGN	PRIMARY	WAITFOR
FREETEXT	PRINT	WHEN
FREETEXTTABLE	PROC	WHERE
FROM	PROCEDURE	WHILE
FULL	PUBLIC	WITH
FUNCTION	RAISERROR	WRITETEXT
GOTO	READ	

I.1 未来的关键字

Microsoft 针对未来用途也保留了一些关键字。在 Microsoft SQL Server 2008 中还没有实现这些关键字，因此不能表示有效的 SQL 命令。不过，最好避免在代码中使用这些关键字（例如，作为标识符），因为在未来的版本中它们可能应用在代码中。注意，这里列出的一些关键字实际上是由 SQL:2003 标准规定的。

Microsoft SQL Server 保留的未来关键字：

ABSOLUTE	BEFORE	CHAR
ACTION	BINARY	CHARACTER
ADMIN	BIT	CLASS
AFTER	BLOB	CLOB
AGGREGATE	BOOLEAN	COLLATION
ALIAS	BOTH	COMPLETION
ALLOCATE	BREADTH	CONNECT
ARE	CALL	CONNECTION
ARRAY	CASCADED	CONSTRAINTS
ASSERTION	CAST	CONSTRUCTOR
AT	CATALOG	CORRESPONDING

www.TopSage.com

TopSage.com

CUBE
CURRENT_PATH
CURRENT_ROLE
CYCLE
DATA
DATE
DAY
DEC
DECIMAL
DEFERRABLE
DEFERRED
DEPTH
DEREF
DESCRIBE
DESCRIPTOR
DESTROY
DESTRUCTOR
DETERMINISTIC
DICTIONARY
DIAGNOSTICS
DISCONNECT
DOMAIN
DYNAMIC
EACH
END-EXEC
EQUALS
EVERY
EXCEPTION
FALSE
FIRST
FLOAT
FOUND
FREE
FULLTEXTTABLE
GENERAL
GET
GLOBAL
GO
GROUPING
HOST
HOUR
IGNORE
IMMEDIATE
INDICATOR
INITIALIZE
INITIALLY
INOUT
INPUT
INT
INTEGER
INTERVAL
ISOLATION
ITERATE
LANGUAGE
LARGE
LAST
LATERAL
LEADING
LESS
LEVEL
LIMIT
LOCAL
LOCALTIME
LOCALTIMESTAMP
LOCATOR
MAP
MATCH
MINUTE
MODIFIES
MODIFY
MODULE
MONTH
NAMES
NATURAL
NCHAR
NCLOB
NEW
NEXT
NO
NONE
NUMERIC
OBJECT
OLD
ONLY
OPERATION
ORDINALITY
OUT
OUTPUT
PAD
PARAMETER
PARAMETERS
PARTIAL
PATH
POSTFIX
PREFIX
PREORDER
PREPARE
PRESERVE
PRIOR
PRIVILEGES
READS
REAL
RECURSIVE
REF
REFERENCING
RELATIVE
RESULT
RETURNS
ROLE
ROLLUP
ROUTINE
ROW
ROWS
SAVEPOINT
SCROLL
SCOPE
SEARCH
SECOND
SECTION
SEQUENCE
SESSION
SETS
SIZE
SMALLINT
SPACE
SPECIFIC
SPECIFICTYPE
SQL
SQLEXCEPTION
SQLSTATE
SQLWARNING
START
STATE
STATEMENT
STATIC
STRUCTURE
TEMPORARY
TERMINATE
THAN
TIME
TIMESTAMP
TIMEZONE_HOUR
TIMEZONE_MINUTE
TRAILING
TRANSLATION
TREAT
TRUE
UNDER
UNKNOWN
UNNEST
USAGE
USING
VALUE
VARCHAR
VARIABLE
WHENEVER
WITHOUT
WORK
WRITE
YEAR
ZONEE

I.2 ODBC 的保留关键字

尽管这些关键字还没有包含在规定的 SQL 核心中，但在应用程序中应当避免这些关键字以便保持与符合 SQL:2003 驱动程序的兼容性。

ODBC 3.0 的保留关键字：

ABSOLUTE
ACTION
ADA
ADD
ALL
ALLOCATE
ALTER
AND
ANY
ARE
AS
ASC
ASSERTION
AT
AUTHORIZATION
AVG
BEGIN
BETWEEN
BIT
BIT_LENGTH
BOTH
BY
CASCADE
CASCADED
CASE
CAST
CATALOG
CHAR
CHAR_LENGTH
CHARACTER
CHARACTER_LENGTH
CHECK
CLOSE
COALESCE
COLLATE
COLLATION
COLUMN
COMMIT
CONNECT
CONNECTION
CONSTRAINT
CONSTRAINTS

www.TopSage.com

CONTINUE
CONVERT
CORRESPONDING
COUNT
CREATE
CROSS
CURRENT
CURRENT_DATE
CURRENT_TIME
CURRENT_TIMESTAMP
CURRENT_USER
CURSOR
DATE
DAY
DEALLOCATE
DEC
DECIMAL
DECLARE
DEFAULT
DEFERRABLE
DEFERRED
DELETE
DESC
DESCRIBE
DESCRIPTOR
DIAGNOSTICS
DISCONNECT
DISTINCT
DOMAIN
DOUBLE
DROP
ELSE
END
END-EXEC
ESCAPE
EXCEPT
EXCEPTION
EXEC
EXECUTE
EXISTS
EXTERNAL
EXTRACT
FALSE
FETCH
FIRST
FLOAT
FOR
FOREIGN
FORTRAN
FOUND
FROM
FULL
GET
GLOBAL
GO
GOTO
GRANT
GROUP
HAVING
HOUR
IDENTITY
IMMEDIATE
IN
INCLUDE
INDEX
INDICATOR
INITIALLY
INNER
INPUT
INSENSITIVE
INSERT
INT
INTEGER
INTERSECT
INTERVAL
INTO
IS
ISOLATION
JOIN
KEY
LANGUAGE
LAST
LEADING
LEFT
LEVEL
LIKE
LOCAL
LOWER
MATCH
MAX
MIN
MINUTE
MODULE
MONTH
NAMES
NATIONAL
NATURAL
NCHAR
NEXT
NO
NONE
NOT
NULL
NULLIF
NUMERIC
OCTET_LENGTH
OF
ON
ONLY
OPEN
OPTION
OR
ORDER
OUTER
OUTPUT
OVERLAPS
PAD
PARTIAL
PASCAL
POSITION
PRECISION
PREPARE
PRESERVE
PRIMARY
PRIOR
PRIVILEGES
PROCEDURE
PUBLIC
READ
REAL
REFERENCES
RELATIVE
RESTRICT
REVOKE
RIGHT
ROLLBACK
ROWS
SCHEMA
SCROLL
SECOND
SECTION
SELECT
SESSION
SESSION_USER
SET
SIZE
SMALLINT
SOME
SPACE
SQL
SQLCA
SQLCODE
SQLERROR
SQLSTATE
SQLWARNING
SUBSTRING
SUM
SYSTEM_USER
TABLE
TEMPORARY
THEN
TIME
TIMESTAMP
TIMEZONE_HOUR
TIMEZONE_MINUTE
TO
TRAILING
TRANSACTION
TRANSLATE
TRANSLATION
TRIM
TRUE
UNION
UNIQUE
UNKNOWN
UPDATE
UPPER
USAGE
USER
USING
VALUE
VALUES
VARCHAR
VARYING
VIEW
WHEN
WHENEVER
WHERE
WITH
WORK
WRITE
YEAR
ZONE

www.TopSage.com

附录 J
其他 RDBMS

本附录提要：

- 不同 RDBMS 实现概述

本书内容集中在三大供应商——Oracle、IBM 和 Microsoft，但他们并不是唯一的关系数据库软件提供商。我们有意试图绕开 RDBMS 供应商的“激烈争论”，并且希望已经奏效。

实际上除此之外，还有很多不是很出名的 RDBMS 包，并且它们的价值一直没有通过它们的市场份额得到反映。它们中的大多数在成本（其中一些是在 GPL、LGPL、MPLS、BSD、IPL 和 IDPL 许可下是免费分发的；BSD 是其中限制最少的许可证）甚至性能方面与“三大”RDBMS 供应商相比毫不逊色。表 J-1 中列出了其中一些最流行的 RDBMS 包；但远不止这些。

www.TopSage.com

大家网 TopSage.com

表 J-1 RDBMS 软件

RDBMS	SQL	操作系统	许可	供应商信息
ADABAS D	92/99	UNIX、Linux、Windows、mainframe	免费软件/专用	Software AG www.softwareagusa.com
Alpha Five v8	92	Windows	专用	Alpha Software www.alphasoftware.com
Derby(Java DB; Cloudscape)v10	92/99	实现 JVM 的任何平台	开源（Apache 2.0 许可证）	IBM www.cloudscape.com
Empress	92	Sun Solaris、HP_UX、IBM AIX、True64 UNIX IRIX、Linux、FreeBSD、Unicos、Microsoft Windows NT/2000/XP、LynxOS、QNX、RTU、Bluecat、RTLinux	专用	Empress Software Inc. www.empress.com
eXtremeDB	92	执行 JVM 的任何平台	专用/开源（Prest）	www.mcobject.com
FirstBase v9.3.2	92	UNIX, Linux	开源（GNU GPL）	FirstBase 软件 http://www.firstbasesoftware.com/fb_builder.htm
FireBird v3.0	92/99/2003	UNIX、Linux、Windows	开源（IPL、IDPL）	www.firedbirdsql.org
ForxPro/FoxBase	92	Microsoft Windows 9x/NT/2000/XP	专用	Microsoft Corportation www.microsoft.com
H2	92	执行 JVM 的任何平台	开源（MPL 1.1）	www.h2database.com
HSQLDB	99/2003	执行 JVM 的任何平台	开源（BSD）	www.hsqldb.org
INGRES II	92	Windows NT/2000、UNIX、Open VMS、Linux	开源（GNU GPL）	Ingres http://ingres.com
Informix	92	IBM AIX、SGI IRIX; Linux; Reliant UNIX; Sun Solaris; Compaq Tru64; Windows NT/2000/XP; UNIXWare、HP-UX	专用	IBM www.informix.com www.ibm.com
InterBase	92	Windows NT/2000/XP、UNIX、Linux	专用	Borland Software Corporation www.borland.com
LEAP RDBMS v1.2.6	92	UNIX、Linux	开源（GNU GPL）	http://leap.sourcebreakforge.net
LucidDB	92/99	Linux、Windows	开源（GNU GPL）	www.luciddb.org
MaxDB	92/99	HP-UX、Sun Solaris、Linux、Windows	开源（GNU GPL/商业）	https://www.sdn.sap.com/irj/sdn/maxdb
MonetDB	92/2003	Linux、Windows	开源	www.monetdb.nl
SmallSQL	92/99	执行 JVM 的任何平台	开源（LGPL）	www.smallsql.de
SQLite	92	Linux、Windows	开源（GNU GPL）	www.sqlite.org
Mimer SQLite	92/99	Windows、Mac OS X、Linux、Symbian OS、UNIX OpenVMS	专用	www.mimer.com
Microsoft Access 2007	92/99	Windows 9x/NT/2000/XP/Vista	专用	Microsoft Corportation www.microsoft.com
Microsoft Visual FoxPro	92/99	Windows	专用	Microsoft Corportation www.microsoft.com
mSQL 3.8	92	UNIX、Linux、MacOS	专用	Hughes Technologies Pty. Ltd. www.hughes.com.au
MySQL 5.1/6.0	92/99	Linux、FreeBSD、Windows 9x/NT/2000/XP、Sun Solaris、HP-UX、Mac OS X、AIX、SGI Irix,、SCO Unix、DEC OSF、BSDi	GNU GPL	MySQL AB，www.mysql.com
Non-Stop SQL/MX	92/99	UNIX、Linux、Windows NT/2000	专用	Hewlett-Packard Company http://h71028.www7.hp.com/NonStopComputing/cache/81318-0-0-0-121.html
OpenBase SQL v10		Windows、Mac OS X、Linux	专用	www.openbase.com

www.TopSage.com

续表

RDBMS	SQL	操作系统	许可	供应商信息
OpenLink Virtuoso	92/99/2003	Windows、HP-UX、AIX、Sun Solaris、DEC、FreeBSD、SCO、Linux、MacOS X	开源（GNU GPL）/专用	Virtuoso.openlinksw.com
PostgreSQL v8	92/99	AIX、BeOS、FreeBSD、BSD、Tru64 UNIX、DG/UX、Linux、Mac OS X、SCO OpenServer、SCO UNIXWare、SGI Irix、Sun Solaris、Microsoft Windows NT/2000	开源（BSD）	开源软件 www.postgresql.org GNU General Public License
Progress ObjectStore v7.0	92	Sun Solaris、SCO Open Server、IBM AIX、True64 UNIX, HP-UX、Linux、DG/UX Intel、Citrix MetaFrame、Windows 9x/NT/2000/XP	专用	Progress Software Corp. www.progress.com
Pervasive SQL 2000i	92/99	NetWare、Windows 9x/NT/2000、Microsoft-DOS、Linux	专用	Pervasive Software www.pervasive.com
SQLBase v8.0	92	Windows 98/NT/2000/XP、Netware 5.0&6	专用	Gupta Technologies, LLC http://www.unify,com /Products/ SQLBase/ default.aspx
StorHouse/RM	92	HP-UX、Sun Solaris	专用	FileTek, Inc. www.filetek.com
Sybase Adaptive Server	92/99	Sun Solaris、HP-UX、SGI Irix、IBM AIX、Windows NT/2000/XP、Linux、Mac OS X	专用	Sybase, Inc. www.sybase.com
tdbengine	92	Linux、FreeBSD、Windows	免费软件/开源（GNU GPL）	www.tdbengine.org
Teradata	92/99	NCR UNIX SVR4 MP-RAS、Windows、Linux	专用	Teradata Corporation www.teradata.com
txtSQL	92	基于 PHP	开源（GNU GPL）	www.txtsql.com
TimesTen RDBMS/Oracle TimesTen v6.0	92	Sun Solaris、HP-UX、AIX、Windows NT/2000/XP、LynxOS、Linux	专用	Oracle Corporation www.timesten.com www.oracle.com
Versant/SQL v7.0	92	Microsoft Windows 2000、XP、2003，Sun Solaris 2.9、2.10，IBM AIX 5.1、5.3，HP-UX 11i，Red Hat RHEL 3.0、4.0	专用	Versant Corporation www.versant.com
VistaDB	92	Windows、Linux(w/Mono)	专用	VistaDB Software Inc. www.vistadb.net
UniVerse	92	Windows NT、Linux、UNIX	专用	IBM www-306.ibm.com/software/data/u2/universe

www.TopSage.com

APPENDIX K

附录 K 数值系统、布尔代数和集合论简介

本附录提要：

- 数值系统
- 布尔代数的逻辑元素
- 集合论

我们强烈认为，掌握了作为 SQL 基础的坚实数学和集合论基础的 RDBMS 和 SQL 日常用户将获益匪浅。

编写和阅读 SQL 代码不需要很高的数学水平，但是了解二进制和十六进制数与十进制之间的差异和它们彼此之间如何转换的知识有助于更好地理解与其直接相关的问题。其中包括安全、数据加密、数据存储原理以及 Oracle ROWID 数据类型——稍有提及。

了解集合论尤其重要。基本上，任何数据库查询的结果都是一个值集合，因此了解集合论有益于了解使用集合创建有效率的查询的规则。

K.1 数值系统

我们在日常生活中随意使用的十进制数值系统并不是世界上唯一的数值系统。大多数人习惯于十进制系统，认为这是理所当然的，甚至可能认为它是唯一可能的数值集合，但是只有在谈论现代人类时，这种假设才是有效的。机器并不使用十进制系统。而且，人类本身，现在和过去都使用过其他系统。例如，六千年前古代闪族文明使用了一种基本的六十位数值系统；即，他们使用了 60 个不同的字符来表示数字。在我们这个时代，英文单词“dozen”就表明了一种不同于十进制的数值系统的存在。

K.1.1 RDBMS 连接

在表 K-1 中列出通常与计算机科学特别是 SQL 相关的基本数值系统：

表 K-1 数值系统

系统	元素
十进制	{0,1,2,3,4,5,6,7,8,9}
二进制	{0,1}
十六进制	{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}

十六进制系统对于表示较大的二进制数很有用。例如，一个 16 位二进制序列，在编程术语中称为一个“字”（word），可以分为四组，每组有四位，然后每组可以很方便地表示为一个十六进制数（表 K-2）从而更便于使用。

www.TopSage.com

TopSage.com

表 K-2 使用十六进制组表示的二进制数

二进制数	0010111011111100			
二进制组	0010	1110	1111	1110
十六进制组	2	E	F	C

注意：因为可能的最大 4 位数值（1111）是十进制数 15，所以在十进制系统中不能只使用一个字符表示它（以及数字 10、11、12、13 和 14）。在这种情况下十六进制系统是非常方便的，因为 1111 可以表示为十六进制的 F（等价于十进制数 15）。

K.1.2 转换数值

使用一种数学算法，即一系列操作，可以把数从一种数字系统转换为另一种。

1. 二进制到十进制转换

使用下面的语法可以将一个二进制数转换为十进制：从右到左对组成数（0 和 1）的元素计数，对于第一个元素使用 0，第二个使用 1，直到最后一个元素 *n*。然后从 n^{th}（最左边）开始，计算每个元素乘以 2 的 *n* 次幂的总和：

$$(A_n\times2)^n+(A_{n-1}\times2)^{n-1}+\ldots+(A_0\times2)^0$$

表 K-3 举例说明了二进制数（10011）到十进制的转换。

表 K-3 二进制到十进制转换

二进制数	1	0	0	1	1
n	4	3	2	1	0
计算	$(1\times2)^4$	$(0\times2)^3$	$(0\times2)^2$	$(1\times2)^1$	$(1\times2)^0$
结果	16	0	0	2	1
求和	16+0+0+32+1=19				

2. 十进制到二进制转换

使用这种逻辑可以将一个十进制数转换为二进制：使用整数除法用 2 除一个十进制数。写下余数（从右到左）。使用结果数重复操作。直到结果数为零：

```
SET j to 0
SET nj to N
WHILE nj <> 0
{
 aj := remainder of nj / 2
 nj + 1 := floor (nj / 2)
 j := j + 1
}
```

表 K-4 举例说明了十进制数 123 到其二进制形式的转换。结果的二进制数是 1111011。

表 K-4 十进制到二进制转换

N=123	123						
J	0	1	2	3	4	5	6
nj	123	61	30	15	7	3	1
aj	61	30	15	7	3	1	0
余数	1	1	0	1	1	1	1

K.2 布尔代数的逻辑元素

布尔代数是以 George Boole 命名的系统，George Boole 是 19 世纪中叶的英国数学家。布尔代数基于本章前面所述的二进制数字系统。除了二进制元素（0 和 1）之外还包括一组运算符。4 个基本的布尔运算符是：

- NOT
- AND
- OR

www.TopSage.com

- XOR

还有两个运算符派生自三个基本运算符的：

- NAND
- NOR

注意：从第 11 章可以知道，运算符可以是一元的也可以是二元的（在这里的词语“二元”与二进制数无关；它只是表示运算符需要两个操作数）。NOT 是一个一元运算符；本附录中涵盖的其他运算符都是二元的。

K.2.1 NOT（补集或求反）

NOT 运算符接受布尔变量的值作为其输入并输出该值的相反值。参见表 K-5。

表 K-5 NOT 真值表

VALUE	0	1
NOT(VALUE)	1	0

K.2.2 AND（布尔积）

AND 运算符接受两个布尔变量作为输入并输出二者的布尔积。参见表 K-6。

表 K-6 AND 真值表

VALUE 1	0	0	1	1
VALUE 2	0	1	0	1
VALUE 1 AND VALUE 2	0	0	0	1

K.2.3 OR（布尔和）

OR 运算符接受两个布尔变量作为输入并输出二者的布尔和。参见表 K-7。

表 K-7 OR 真值表

VALUE 1	0	0	1	1
VALUE 2	0	1	0	1
VALUE 1 OR VALUE 2	0	1	1	1

K.2.4 XOR（异或）

XOR 运算符接受两个布尔变量作为输入并输出二者的异或布尔和（对于 XOR 输出为 1 则必有一个输入变量为 1）。参见表 K-8。

表 K-8 XOR 真值表

VALUE 1	0	0	1	1
VALUE 2	0	1	0	1
VALUE 1 XOR VALUE 2	0	1	1	0

K.2.5 NAND（与非）

NAND 运算符接受两个布尔变量作为输入并输出二者布尔积的相反值。参见表 K-9。

表 K-9 NAND 真值表

VALUE 1	0	0	1	1
VALUE 2	0	1	0	1
VALUE 1 NAND VALUE 2	1	1	1	0

注意：NOT（A AND B）并不等同于 NOT(A) AND NOT(B)。

K.2.6 NOR（或非）

NOR 运算符接受两个或多个布尔变量作为输入并输出它们布尔和的补集。参见表 K-10。

www.TopSage.com

表 K-10 NOR 真值表

VALUE 1	0	0	1	1
VALUE 2	0	1	0	1
VALUE 1 NOR VALUE 2	1	0	0	0

Note 注意：NOT（A OR B）并不等同于 NOT(A) + NOT(B)。

K.2.7 优先级规则

表 K-11 说明了布尔代数运算符的优先级规则。

表 K-11 布尔代数运算符优先级

优先级	运算符
1	括号 ()
2	布尔补集 NOT
3	布尔积 AND
4	布尔和 OR

Note 注意：括号具有最高优先级，即，首先计算括号内的表达式结果。

表 K-12 举例说明在对 NOT（TRUE 或 FALSE）OR TRUE AND FALSE 表达式求值时如何使用优先级规则。

Note 注意：记住，在布尔代数中 1 表示 TRUE 而 0 表示 FALSE。

表 K-12 优先级规则示例

步骤	表达式	说明
1	NOT(1 OR 0) OR 1 AND 0	1 OR 0 位于括号中，因此首先对其求值。结果是 1，因此用 1 替换（1 OR 0）
2	= NOT(1) OR 1 AND 0	接下来对补集求值。NOT(1) = 0。用 0 替换 NOT(1)
3	= 0 OR 1 AND 0	接下来求布尔积。1 AND 0 = 0。使用 0 替换 1 AND 0
4	= 0 OR 0	现在求布尔和。0 OR 0 = 0，因此表达式结果是 0
5	= 0	现在计算完成

表 K-13 包含了布尔代数的主要恒等式。

表 K-13 布尔代数恒等式

名称	相应的注解
补余律	X OR NOT(X) = TRUE X AND NOT(X) = FALSE
双补余律	NOT(NOT(X)) = X
幂等律	X OR X = X X AND X = X
恒等律	X OR FALSE = X X AND TRUE = X
优势律	X OR TRUE = TRUE X AND FALSE = FALSE
交换律	X OR Y = Y OR X X AND Y = Y AND X
结合律	X OR(Y OR Z) = (X OR Y) OR Z X AND(Y AND Z) = (X AND Y) AND Z
分配律	X OR(Y AND Z) = (X OR Y) AND (X OR Z) X AND(Y OR Z) = (X AND Y) OR (X AND Z)
DeMorgan 律	NOT(X AND Y) = NOT(X) OR NOT(Y) NOT(X OR Y) = NOT(X) AND NOT(Y)
吸收律	X AND (X OR Y) = X

www.TopSage.com

大家网 TopSage.com

K.3 集合论

集合是所有数学分支中的一个基本概念并且也是SQL的重要概念之一，因为SQL查询就是操作记录集。集合的概念非常直观：一组对象的集合。这些对象可以具有相同类型或不同类型。在数学论中，**集合**是定义良好的对象的一个集合；这些对象被称为集合的**元素**。集合的元素（或成员）就是指元素**属于**集合。

K.3.1 集合列表

通常使用大写字母表示集合，而集合的元素用小写字母（或数字）表示。一个集合可以包含任意内容并且集合元素不必是相同类型的对象。在数学符号中，通过在括号之间列出元素或使用集合构造符号可以定义一个集合：

$$A = \{1,2,5,8\}$$

$$B = \{x \mid x \text{ in N}, 10 <= x <= 200\}$$

这些元素的顺序可以任意排列：

$$\{1,5,2,6\} = \{1,2,5,6\} = \{6,1,2,5\}$$

集合可以包含其他集合作为其元素：

$$A = \{1,3,\{1,2\},4,7\}$$

假设$B = \{1,2\}$且$C = \{7\}$，$A = \{1,3,B,4,C\}$。

空集合就是没有元素的集合，表示为{}或{0}。

在传统的集合论中，忽略重复的元素，或者更明确的是，重复的元素均当作一个单独的元素：

$$\{a,a,c,b,c\} = \{a,b,c\}$$

集合未必是有限的，一个无限集合的典型例子是所有整数的集合：

$$Z = \{\ldots,-2,-1,0,1,2,\ldots\}$$

另一个典型例子是自然数的集合，它由所有正整数组成：

$$N = \{1,2,3,\ldots\}$$

还有一个例子是整数集合：

$$W = \{0,1,2,\ldots\}$$

有时很难或不可能列出一个集合；例如从10到200的所有自然数集合。在这种情况下我们可以通过表示其元素的属性来定义集合：

$$A = \{x \mid x \text{ in } N, 10 <= x <= 200\}$$

这种表示符号读作A是全部x的集合，其中x是一个自然数（N）并且在10和200之间。

K.3.2 子集

如果集合P的每个元素也是集合Q的一个元素，那么我们说集合P是集合Q的一个**子集**。这种情况不排除$P = Q$的可能性。

例如，如果：

$$A = \{1,2,3,4,5,6\}$$

$$B = \{1,3,7\}$$

$$C = \{2,4,6\}$$

那么C是A的子集，而B不是A的子集，因为它包含的一个元素7不是集合A的成员。

注意：与子集的概念有关的是超集的概念。A是C的一个超集，意味着A包含C。

K.3.3 集合等式

如果集合A是集合B的一个子集且B是A的一个子集，那么这两个集合相等。换言之，如果A

www.TopSage.com

的每个元素也属于 B 而 B 的每个元素也属于 A，那么 $A=B$。

K.3.4 集合运算

对集合进行运算包括联合、交集、补集、差集和笛卡尔乘积。

1. 联合

两个集合 A 和 B 的联合是包含 A 或 B 或两者中全部元素的集合。写作 $A \cup B$。n 个集合的联合，$A_1,A_2,\ldots,A_n$ 是全部对象的集合，它属于至少其中一个集合：

$$A = \{2,5\}$$
$$B = \{1,9\}$$
$$A \cup B = \{1,2,5,9\}$$

图 K-1 举例说明了这个概念。

Note 注意：在 SQL 中，UNION 运算符就像这里说明的那样使用。如果得到的结果集合包含重复元素，则会予以排除。例如，{1,2,5} ∪ {1,5,9} = {1,2,5,9}。要保留重复的元素，SQL 使用另一个运算符，UNION ALL，它不是经典的集合论的组成部分。如果你使用 UA 表示 UNION ALL，那么{1,2,5}UA{1,5,9} = {1,1,2,5,5,9}。

2. 交集

两个集合 A 和 B 的交集是在 A 和 B 中找到的全部共有元素的集合。写作 $A \cap B$。n 个集合的交集，$A_1,A_2,\ldots,A_n$ 是属于每一个集合的全部对象的集合：

$$A = \{a,b,d\}$$
$$B = \{d,e,f\}$$
$$A \cap B = \{d\}$$

图 K-2 显示了交集。

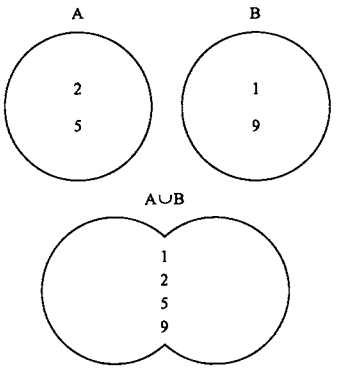


图 K-1 集合的联合

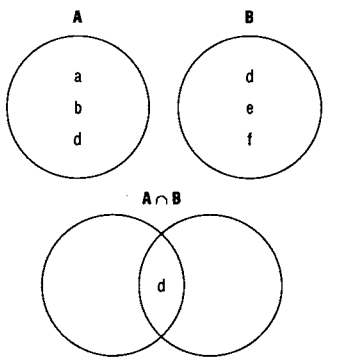


图 K-2 集合的交集

3. 补集

一个集合 A 的补集（或绝对补集）是集合 NOT(A)或包含所有不在 A 中的元素的 A'。这个定义需要存在一个通用集合 U：

$$A = \{a,b,c\}$$
$$U = \{a,b,c\ldots,z\}$$
$$A' = \{d,e,f,\ldots,z\}$$

Note 注意：在 SQL:2003 标准中（以及所有“三大”数据库），使用 NOT 运算符表示补集。

4. 差集

集合 A 和 B 的差集被定义为集合 $A-B$，由不属于 B 的所有 A 元素组成。A 和 B 的差集与 B 和 A 的差集不同。集合差集的定义并不意味着 A 和 B 具有共有的部分，也不能说明有关它们相对大小的内容：

www.TopSage.com

$$A = \{1,3,5,7,8\}$$
$$U = \{3,4,6,8,10\}$$
$$A-B = \{1,5,7\}$$
$$B-A = \{4,6,10\}$$

在图 K-3 中举例说明了集合差集。

Note 注意：在 SQL:2003 和 DB2（在 Oracle 中是 MINUS）中通过 EXCEPT 表示 DIFFERENCE 集合运算符。Microsoft SQL Server 可以识别 EXCEPT 和 MINUS；参阅第 8 章了解更多信息。

5. 笛卡尔乘积

两个集合 A 和 B 的笛卡尔乘积表示为 $A \times B$（也称为乘积集合或 A 和 B 的乘积），它是一组有序对的集合，其中每一对的第一个元素是第一个集合的成员，而第二个元素是第二个集合的成员：

$$A = \{1,2,3\}$$
$$B = \{7,8\}$$
$$A \times B = \{(1,7),(1,8),(2,7),(2,8),(3,7),(3,8)\}$$

图 K-4 中表示了笛卡尔乘积。

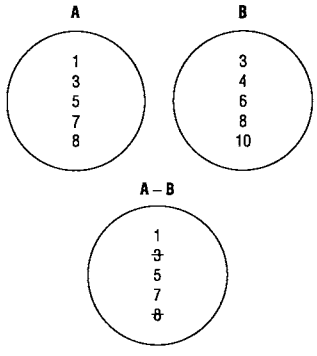


图 K-3 集合的差集

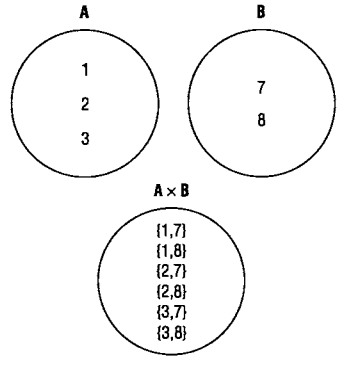


图 K-4 集合的笛卡尔乘积

Note 注意：在 SQL:2003 和 Microsoft SQL Server 中笛卡尔乘积就是众所周知的 CROSS JOIN。

6. 多个运算

对于多个联合和交集，可以使用括号来划分运算顺序，例如，$(A \cap B) \cup C$。

7. 集合基数（cardinality）

基数就是描述一个集合中元素数的术语。它表示为$|A|$：

$$A = \{1,3,5\}$$
$$|A| = 3$$
$$B = \{a,b,f,r,t,y\}$$
$$|B| = 6$$

Note 注意：对于空集合，$|\{\}| = 0$。

对于多个操作数的集合运算

集合论完全适用于 SQL 查询中返回的结果集。在多数情况下，一个查询返回一个以上的列，这就意味着两个主要规则。

首先，如果两个集合具有完全相同的子元素数，则可以对它们进行集合运算。（不要把这个规则与集合基数混淆！）那就意味着可以进行集合 A={(1,2),(5,6),(10,12)}和集合 B={(1,3),(16,18)}的联合（$A \cup B$={(1,2),(5,6),(10,12),(1,3),(16,18)}），但是如果 A={(1,2),(5,6),(10,12)}而 B={(1,3,5),(16,18,7)}，则联合不成立（在 SQL 中）。

www.TopSage.com

其次，元素中的子元素被当作一个单独实体；也就是前面的例子中的结果 $A \cup B$ 包含(1,2)和(1,3)的原因。

TopSage.com

K.3.5 集合代数恒等式

在表 K-14 中列出了集合代数的恒等式。

表 K-14 集合代数恒等式

名称	恒等式
幂等律	$A \cup A = A$ $A \cap A = A$
结合律	$A \cup (B \cup C) = (A \cup B) \cup C$ $A \cap (B \cap C) = (A \cap B) \cap C$
交换律	$A \cup B = B \cup A$ $A \cap B = B \cap A$
分配律	$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
恒等律	$A \cup \{\} = A$ $A \cup U = U$ $A \cap \{\} = \{\}$ $A \cap U = A$
补余律	$A \cup A' = U$ $A \cap A' = U$ $(A')' = A$ $U' = \{\}$ $\{\}' = U$
DeMorgan 律	$(A \cup B)' = A' \cup B'$ $(A \cap B)' = A' \cap B'$

www.TopSage.com

计算机精品学习资料大放送

软考官方指定教材及同步辅导书下载 | 软考历年真是解析与答案

软考视频 | 考试机构 | 考试时间安排

Java 一览无余：Java 视频教程 | Java SE | Java EE

.Net 技术精品资料下载汇总：ASP.NET 篇

.Net 技术精品资料下载汇总：C#语言篇

.Net 技术精品资料下载汇总：VB.NET 篇

撼世出击：C/C++编程语言学习资料尽收眼底 电子书+视频教程

Visual C++(VC/MFC)学习电子书及开发工具下载

Perl/CGI 脚本语言编程学习资源下载地址大全

Python 语言编程学习资料(电子书+视频教程)下载汇总

最新最全 Ruby、Ruby on Rails 精品电子书等学习资料下载

数据库管理系统(DBMS)精品学习资源汇总：MySQL 篇 | SQL Server 篇 | Oracle 篇

平面设计优秀资源学习下载 | Flash 优秀资源学习下载 | 3D 动画优秀资源学习下载

最强 HTML/xHTML、CSS 精品学习资料下载汇总

最新 JavaScript、Ajax 典藏级学习资料下载分类汇总

网络最强 PHP 开发工具+电子书+视频教程等资料下载汇总

UML 学习电子资下载汇总 软件设计与开发人员必备

经典 LinuxCBT 视频教程系列 Linux 快速学习视频教程一帖通

天罗地网：精品 Linux 学习资料大收集(电子书+视频教程) Linux 参考资源大系

Linux 系统管理员必备参考资料下载汇总

Linux shell、内核及系统编程精品资料下载汇总

UNIX 操作系统精品学习资料<电子书+视频>分类总汇

FreeBSD/OpenBSD/NetBSD 精品学习资源索引 含书籍+视频

Solaris/OpenSolaris 电子书、视频等精华资料下载索引