ORACLE

Oracle Database 12*c* SQL

ORACLE DATABASE 12*c*

精通 Oracle Database 12*c* SQL & PL/SQL编程(第3版)

[美] Jason Price 著
卢 涛 译

清华大学出版社

精通 Oracle Database 12*c* SQL & PL/SQL 编程

(第 3 版)

[美] Jason Price 著

卢 涛 译

清华大学出版社

北 京

Jason Price
Oracle Database 12*c* SQL
ISBN：978-0-07-179935-5

北京市版权局著作权合同登记号　图字：01-2013-8903

图书在版编目(CIP)**数据**

精通 Oracle Database 12*c* SQL & PL/SQL **编程：第** 3 **版**/(**美**) **普赖斯**(Price，J.) **著；卢涛 译**. —**北京：清华大学出版社，**2014

书名原文：Oracle Database 12*c* SQL

ISBN 978-7-302-36598-3

I. ①**精**… Ⅱ. ①**普**… ②**卢**… Ⅲ. ①**关系数据库系统—程序设计** Ⅳ. ①TP311.138

中国版本图书馆 CIP **数据核字**(2014)**第** 112163 **号**

责任编辑：王　军　李维杰
封面设计：牛艳敏
责任校对：成凤进
责任印制:

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A **座**　　**邮　　编：**100084
社 总 机：010-62770175　　**邮　　购：**010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质　量　反　馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
装 订 者：
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张：**38.25　　**字　　数：**979 **千字**
版　　次：2014 **年** 6 **月第** 1 **版**　　**印　　次：**2014 **年** 6 **月第** 1 **次印刷**
印　　数：1～3500
定　　价：79.80 **元**

产品编号：

译 者 序

本书是专门为从事 Oracle 数据库开发工作的人们准备的，并且此书特别适合 Oracle 数据库开发新手。作为专业社区 ITPUB Oracle 开发版的一名版主，我经常会遇到被某些基础问题困扰的人们，他们很想快速掌握 Oracle SQL 开发技术，经常会询问我有什么好书可以推荐给他们。很遗憾的是，我自己过去参与编写和翻译的一些 Oracle 书籍，都需要一定的 Oracle 开发基础，不能满足他们的需求。在我自己的学习和开发工作过程中，也没有看过专门介绍 Oracle SQL 开发的书籍，基础经验大多是在各种培训班和实际工作中逐步积累起来的。

现在，我郑重向广大读者朋友们推荐面前的这本《精通 Oracle Database 12*c* SQL & PL/SQL 编程(第 3 版)》，您可以把它作为开启 Oracle SQL 开发之路的第一本书。

首先，它内容全面，注重实效。既讲述了通用 SQL 技术，又涵盖了 Oracle 的特有技术，它不同于数据库方面的教科书，没有深奥的理论，而是用实例驱动的方法来讲述知识，是一本马上可以投入实际应用的书籍。本书对所涉及内容进行了全面概述，它的好处是，初学者可以清楚地知道 Oracle 究竟提供了哪些功能，可应用于什么场景，这样就避免了多走弯路，费力去实现一些已有的功能。

其次，它深入浅出，结构合理。基础部分完全不需要任何前提，任何人都能看懂并上手，其余部分则介绍了 Oracle 的高级功能，怎样利用它们来完成一些复杂的任务。只要读者循序渐进地阅读，应该很快就能一窥门径。而通过反复练习，就能掌握日常开发所要用到的基本功能。书中介绍的 SQL 优化方法和技术，有助于读者从一开始就养成良好的 SQL 编程习惯，编写出高效的 SQL 语句。

最后，它推陈出新，紧贴主流。本书添加了最新发布的 Oracle Database 12*c* 和主流的 Oracle Database 11*g*R2 的 SQL 开发新功能，已有开发经验的读者也可以从此书获得有用的知识。其中个人感觉比较重要的有：与 SQL 标准兼容的 top-N 查询语法、行间模式匹配查询语法、在 SQL 的 with 子句中嵌入 PL/SQL 函数用法、递归子查询、listagg()函数等。

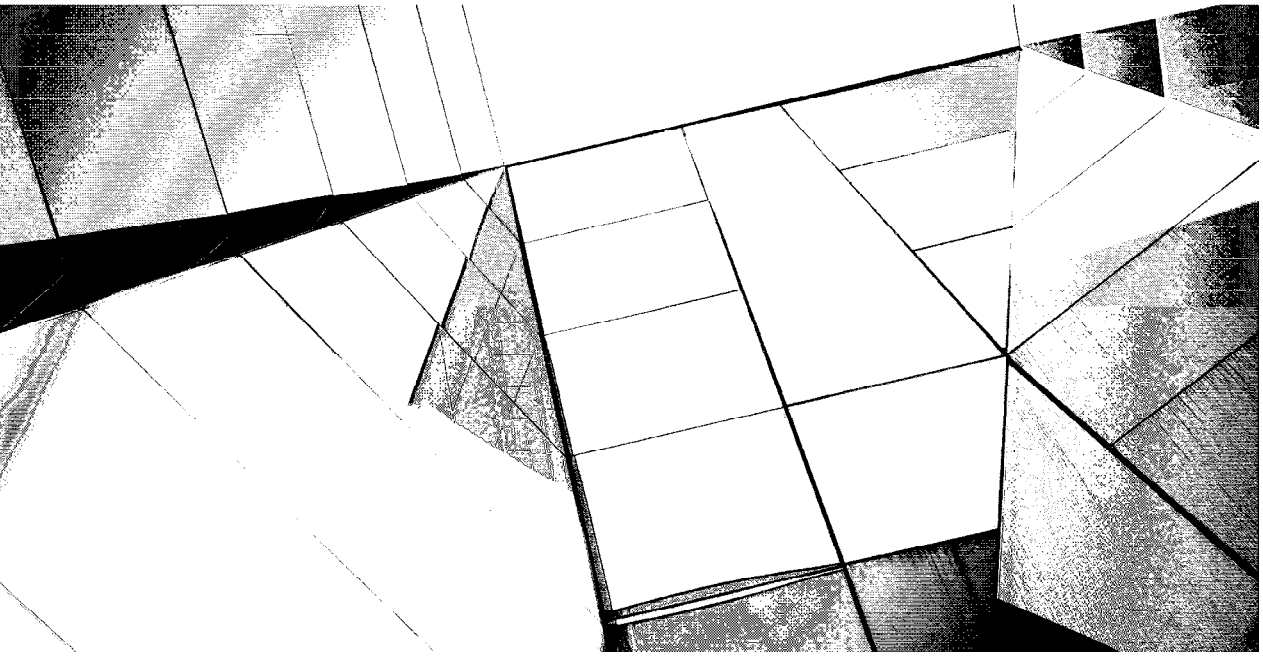
当然，本书也并非完美，由于该书是一本经典之作，历经两次全面更新，一些内容不一定适合于当前主流的 Oracle 版本，唯恐某些读者在理解上有所困惑，我结合自己在学习和开发工作过程中了解到的内容，在一些值得注意的地方添加了注释，希望能对读者有所帮助。

由于本人水平有限，译文中一定还存在着不足之处，欢迎读者批评指正。

卢 涛

译 者 简 介

卢涛，专业社区 ITPUB Oracle **开发版版主。**1995 **年参加工作，**2001 **年转到** IT **部门从事** C/C++**软件开发，**2004 **年开始做系统分析和** Oracle **数据库方面工作。参加过多个全国性普查数据处理项目的开发和运维，目前主要从事统计报表联网填报系统的后台支持和优化。曾参与编写《剑破冰山——**Oracle **开发艺术》一书，并翻译了数本** Oracle **开发和性能优化方面的书籍。**

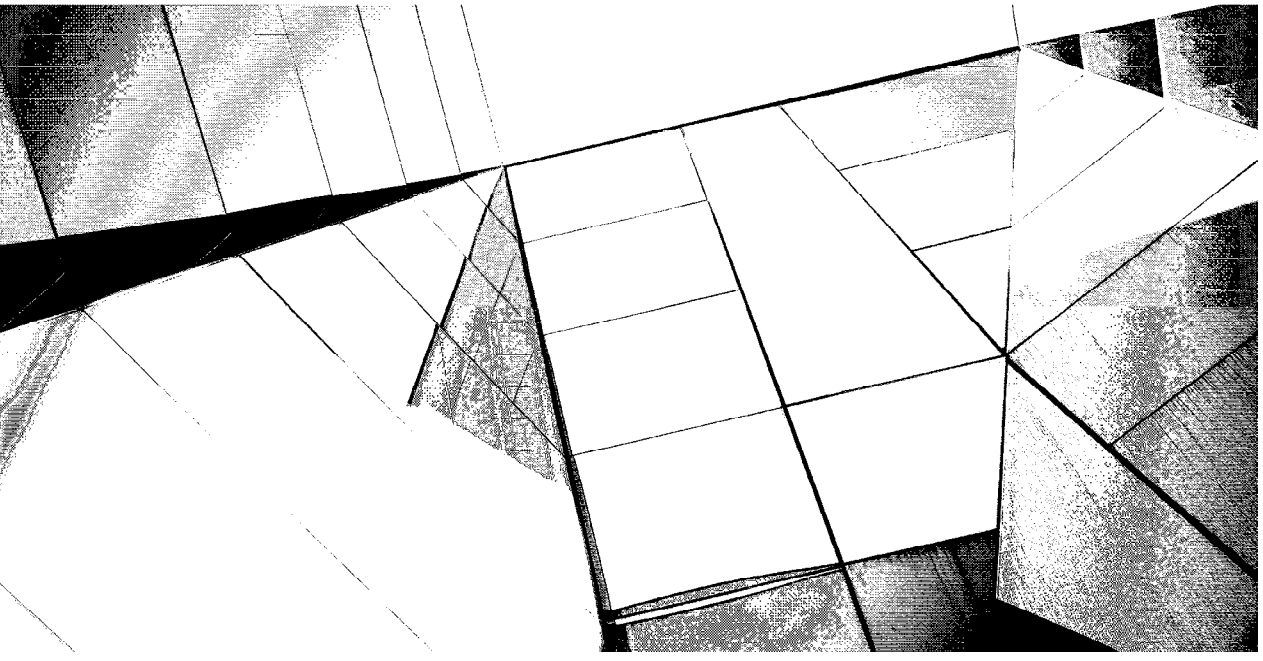


作 者 简 介

Jason Price 是一位职业咨询专家，Oracle **公司前产品经理。他对** Oracle **的众多产品都做出了卓越的贡献，包括数据库、应用服务器和若干** CRM **应用程序。**Jason **是一位经** Oracle **认证的数据库管理员和应用程序开发人员，在软件行业具有** 15 **年以上的从业经验，并执笔撰写了多本关于** Oracle、Java **和**.NET **的优秀图书。**Jason **获得了英国布里斯托尔大学的物理学理学学士学位。**

致谢

感谢 McGraw-Hill Education/Professional **公司的优秀职员。也感谢** Scott Mikolaitis **和** Nidhi Chopra**。**



前　言

现在的数据库管理系统使用一种标准语言——结构化查询语言(Structured Query Language，SQL)访问。此外，SQL 还可以对数据库中的信息进行检索、添加、更新和删除。本书将介绍如何真正掌握 SQL，同时还会给出许多实用的例子。读者可以通过网络获得本书中用到的所有脚本和程序(详细信息参看后文“本书源代码下载”部分)。

通过本书读者可以：

- 掌握标准的 SQL，以及 Oracle 公司为了使用 Oracle 数据库的特性而开发的一些扩展。
- 理解 PL/SQL，它允许用户编写包含 SQL 语句的程序。
- 使用 SQL*Plus 执行 SQL 语句、脚本和报表；SQL*Plus 是一个用于与数据库进行交互的工具。
- 对数据库执行查询、插入、更新和删除操作。
- 创建数据库表、序列、索引、视图和用户。
- 执行包含多条 SQL 语句的事务。
- 定义数据库对象类型，以及创建对象表来处理高级数据。
- 使用大对象来处理包含图像、音乐和电影的多媒体文件。
- 使用分析函数执行复杂计算。
- 实现高性能的优化技术，使 SQL 语句可以快速执行。
- 探讨 Oracle 数据库的 XML 功能。

? 使用最新的 Oracle Database 12*c* SQL 功能。

本书共包含 17 章和一个附录。

第 1 章 简介

本章将介绍有关关系数据库和 SQL 的知识，然后给出几个简单查询，并使用 SQL*Plus 和 SQL Developer 执行这些查询，最后简要介绍 PL/SQL。

第 2 章 从数据库表中检索信息

本章将展示如何使用 SELECT 语句从一个或多个数据库表中检索信息，如何使用算术表达式执行计算，如何使用 WHERE 子句对行进行过滤，以及如何对从表中检索出的行排序。

第 3 章 使用 SQL*Plus

本章将介绍使用 SQL*Plus 来查看表的结构，编辑 SQL 语句，保存并运行脚本，设置列的输出格式，定义并使用变量，以及创建报表。

第 4 章 使用简单函数

本章将介绍有关 Oracle 数据库中内置函数的知识。函数可以接受输入参数，并返回输出参数[1]。使用函数可以实现很多功能，例如计算一组数字的平均值和平方根。

第 5 章 日期和时间的存储与处理

本章将介绍 Oracle 数据库如何处理与存储日期和时间(二者合称时间值)。本章还将介绍如何使用时间戳来存储特定的日期和时间，如何使用时间间隔来存储一定长度的时间段。

第 6 章 子查询

本章将介绍如何在外部的 SQL 语句中放置 SELECT 语句。内部的 SELECT 语句被称为子查询。本章还将介绍子查询的各种类型，以及如何使用子查询从简单部件构建复杂语句。

第 7 章 高级查询

本章将介绍如何执行包含高级操作符和函数的查询。例如，集合操作符可以合并由多个查询返回的行，TRANSLATE()函数可以将一个字符串中的字符转换为另一个字符串中的字符，DECODE()函数可以在一组值中搜索某个特定的值，CASE 表达式可以执行 if-then-else 逻辑，ROLLUP 和 CUBE 子句可以返回包含小计的行。Oracle Database 12*c* 中新增加了 CROSS APPLY 和 OUTER APPLY 来合并两条 SELECT 语句返回的行，还增加了 LATERAL 以返回数据的内联视图。

第 8 章 分析数据

本章将介绍有关分析函数的知识，分析函数可以用来执行复杂计算，例如查找每月销量最

1 译者注：函数必须返回值，而不是输出参数。

高的产品类型、业绩最佳的销售员等。本章还将介绍如何对层次化组织的数据进行查询，并将探讨如何使用 MODEL 子句执行行间计算。最后，我们会讲解 PIVOT 和 UNPIVOT 子句，使用它们可以了解大量数据的整体趋势。Oracle Database 12*c* 中新增加了 MATCH_RECOGNIZE 子句来查找数据中的模式，还增加了 FETCH FIRST 子句来执行 top-N 查询。

第 9 章　修改表的内容

本章将介绍如何使用 INSERT、UPDATE 和 DELETE 语句添加、修改和删除行，如何使用 COMMIT 语句使事务的处理结果永久生效，或者使用 ROLLBACK 语句完全取消事务执行的操作。本章还将介绍 Oracle 数据库如何同时处理多个事务。

第 10 章　用户、特权和角色

本章将介绍有关数据库用户的知识以及如何使用特权和角色来控制用户可以在数据库中执行的特定任务。

第 11 章　创建表、序列、索引和视图

本章将介绍有关表、序列和索引的知识。序列会生成一系列数字，而索引就如同书籍的索引，可以帮助读者快速访问表中的行。本章还将介绍有关视图的知识，视图是对一个或多个表预定义的查询。视图可以对用户屏蔽复杂性，并通过只允许视图访问表中有限的数据集，从另一层面上实现安全特性。本章还将讨论闪回数据归档，这会将对表所做的改变存储一段时间。Oracle Database 12*c* 中新增加了在表中定义可见列和不可见列的能力。

第 12 章　PL/SQL 编程简介

本章将介绍有关 PL/SQL 的知识，PL/SQL 构建在 SQL 基础之上，使用 PL/SQL 可以在数据库中编写包含 SQL 语句的存储程序。PL/SQL 包含标准的编程结构。

第 13 章　数据库对象

本章将介绍如何创建数据库对象类型，数据库对象类型可以包括属性和方法；还将介绍如何使用对象类型来定义列对象和对象表，以及如何使用 SQL 和 PL/SQL 来操纵对象。

第 14 章　集合

本章将介绍如何创建集合类型，集合可以包含多个元素；还将介绍如何使用集合类型来定义表中的列，以及如何使用 SQL 和 PL/SQL 来操纵集合。

第 15 章　大对象

本章将介绍有关大对象的知识，大对象可以用来存储多达 128TB 的字符和二进制数据(也可以是指向外部文件的指针)；此外，还将介绍有关较旧的 LONG 类型的知识，为了保持向后兼容性，在 Oracle Database 12*c* 中依然支持 LONG 类型。

第 16 章　SQL 优化

本章将介绍 SQL 优化的一些技巧，这些技巧可以用来缩短查询执行的时间；本章还将介

绍有关 Oracle 优化器的知识，以及如何向优化器传递一些提示。此外还介绍了如何使用高级调优工具。

第 17 章　XML 和 Oracle 数据库

可扩展标记语言(XML)是一种通用标记语言，可用来在 Internet 上共享结构化数据，并可用来编码数据和其他文档。本章将介绍如何从关系数据生成 XML，以及如何将 XML 保存到数据库中。

附录　Oracle 数据类型

本附录列出了 Oracle SQL 和 PL/SQL 中可以使用的数据类型。

本书读者对象

本书适用于以下读者：

? 需要编写 SQL 和 PL/SQL 的开发人员
? 需要深入了解 SQL 的数据库管理员
? 需要编写 SQL 查询来从自己公司的数据库中获得信息的业务用户
? 需要简单了解 SQL 和 PL/SQL 的技术主管和技术顾问

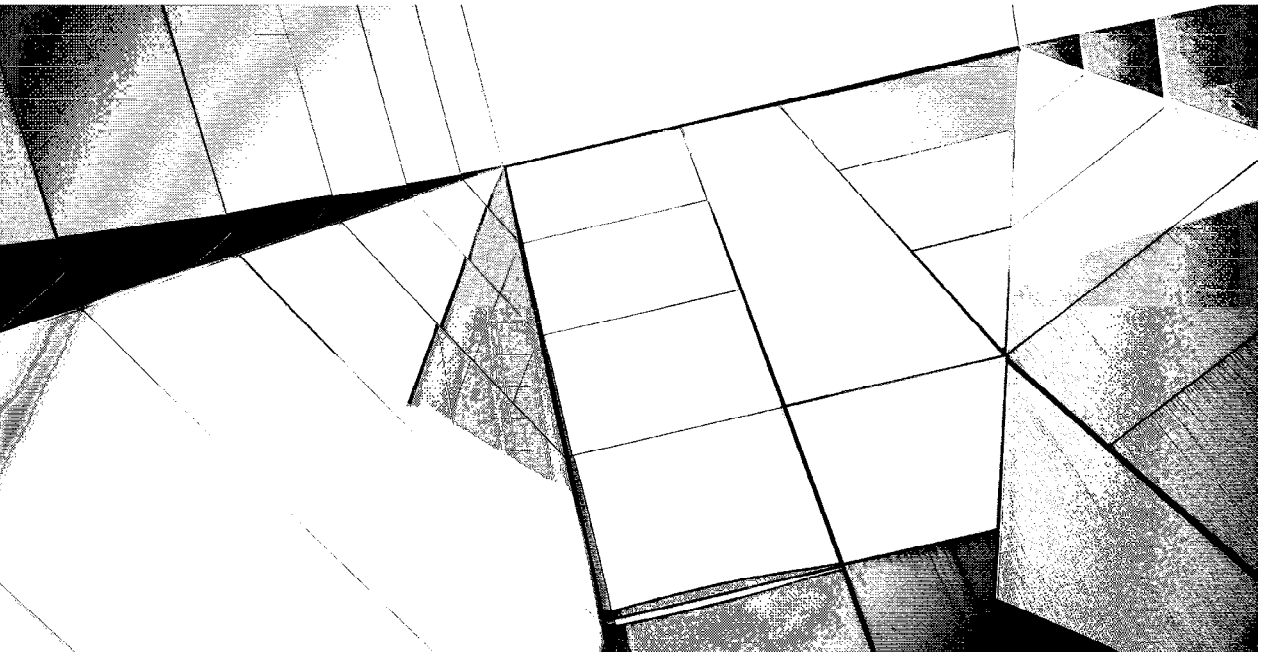
读者阅读本书，不需要预先了解 Oracle 数据库、SQL 或 PL/SQL 的知识；本书为读者提供了成为这方面专家所需的全部知识。

本书源代码下载

本书中使用的所有 SQL 脚本、程序以及其他文件，都可以从 Oracle 出版社的网站 www.OraclePressBooks.com 或本书的合作站点 www.tupwk.com.cn/downpage 下载；这些文件都被打包成一个 Zip 文件。下载这个 Zip 文件之后，需要解压缩。这样就会创建 sql_book 目录，其中包含以下子目录：

? **sample_files** 包含第 15 章中使用的示例文件
? **SQL** 包含本书中使用的 SQL 脚本，包括创建和填充示例数据库表的脚本
? **xml_files** 包含第 17 章中使用的 XML

尽情享受吧，希望您喜欢本书！



目　录

第 1 章　简介 …… 1
1.1　关系数据库简介 …… 1
1.2　SQL 简介 …… 2
1.3　使用 SQL*Plus …… 4
1.3.1　启动 SQL*Plus …… 4
1.3.2　从命令行启动 SQL*Plus …… 4
1.3.3　使用 SQL*Plus 执行 SELECT 语句 …… 5
1.4　使用 SQL Developer …… 6
1.5　创建 store 模式 …… 8
1.5.1　检查脚本 …… 8
1.5.2　运行脚本 …… 10
1.5.3　用来创建 store 模式的 DDL 语句 …… 11
1.6　添加、修改和删除行 …… 17
1.6.1　向表中添加行 …… 17
1.6.2　修改表中的现有行 …… 19
1.6.3　从表中删除行 …… 20
1.7　连接数据库和断开连接 …… 20
1.8　退出 SQL*Plus …… 20
1.9　Oracle PL/SQL 简介 …… 21
1.10　小结 …… 22

第 2 章　从数据库表中检索信息 …… 23
2.1　对单表执行 SELECT 语句 …… 24

2.2 选择一个表中的所有列 24
2.3 使用 WHERE 子句限定行 25
2.4 行标识符 25
2.5 行号 26
2.6 执行算术运算 26
2.6.1 执行日期运算 27
2.6.2 列运算 28
2.6.3 算术运算操作符的优先级 29
2.7 使用列别名 29
2.8 使用连接操作合并列的输出结果 30
2.9 空值 31
2.10 禁止显示重复行 32
2.11 比较值 33
2.11.1 使用不等于操作符 33
2.11.2 使用大于操作符 34
2.11.3 使用小于或等于操作符 34
2.11.4 使用 ANY 操作符 34
2.11.5 使用 ALL 操作符 35
2.12 使用 SQL 操作符 35
2.12.1 使用 LIKE 操作符 36
2.12.2 使用 IN 操作符 37
2.12.3 使用 BETWEEN 操作符 38
2.13 使用逻辑操作符 38
2.13.1 使用 AND 操作符 38
2.13.2 使用 OR 操作符 39
2.14 逻辑操作符的优先级 39
2.15 使用 ORDER BY 子句对行进行排序 40
2.16 执行使用两个表的 SELECT 语句 41
2.17 使用表别名 43
2.18 笛卡尔积 43
2.19 执行使用多于两个表的 SELECT 语句 44
2.20 连接条件和连接类型 45
2.20.1 不等连接 46
2.20.2 外连接 46
2.20.3 自连接 50
2.21 使用 SQL/92 语法执行连接 51
2.21.1 使用 SQL/92 标准语法执行两个表的内连接 51
2.21.2 使用 USING 关键字简化连接 51
2.21.3 使用 SQL/92 执行多于两个表的内连接 52
2.21.4 使用 SQL/92 执行多列的内连接 53
2.21.5 使用 SQL/92 执行外连接 53
2.21.6 使用 SQL/92 执行自连接 55
2.21.7 使用 SQL/92 执行交叉连接 55
2.22 小结 55

第 3 章 使用 SQL*Plus 57
3.1 查看表的结构 58
3.2 编辑 SQL 语句 58
3.3 保存、检索并运行文件 60
3.4 格式化列 63
3.5 设置页面大小 65
3.6 设置行大小 66
3.7 清除列的格式 67
3.8 使用变量 67
3.8.1 临时变量 67
3.8.2 已定义变量 70
3.9 创建简单报表 73
3.9.1 在脚本中使用临时变量 73
3.9.2 在脚本中使用已定义变量 73
3.9.3 向脚本中的变量传递值 74
3.9.4 添加页眉和页脚 75
3.9.5 计算小计 76
3.10 从 SQL*Plus 获取帮助信息 78
3.11 自动生成 SQL 语句 79
3.12 断开数据库连接并退出 SQL*Plus 79

3.13 小结 79

第 4 章 使用简单函数 81
4.1 使用单行函数 82
4.1.1 字符函数 82
4.1.2 数值函数 91
4.1.3 转换函数 96
4.1.4 正则表达式函数 106
4.2 使用聚合函数 112
4.2.1 AVG() 113
4.2.2 COUNT() 114
4.2.3 MAX()和 MIN() 114
4.2.4 STDDEV() 115
4.2.5 SUM() 115
4.2.6 VARIANCE() 115
4.3 对行进行分组 116
4.3.1 使用 GROUP BY 子句对行进行分组 116
4.3.2 调用聚合函数的错误用法 119
4.3.3 使用 HAVING 子句过滤行组 120
4.3.4 组合使用 WHERE 和 GROUP BY 子句 120
4.3.5 组合使用 WHERE、GROUP BY 和 HAVING 子句 121
4.4 小结 122

第 5 章 日期和时间的存储与处理 123
5.1 几个简单的存储和检索日期的例子 123
5.2 使用 TO_CHAR()和 TO_DATE()转换时间值 125
5.2.1 使用 TO_CHAR()将时间值转换为字符串 125
5.2.2 使用 TO_DATE()将字符串转换为时间值 130
5.3 设置默认的日期格式 132
5.4 Oracle 对两位年份的处理 133
5.4.1 使用 YY 格式 133
5.4.2 使用 RR 格式 134
5.5 使用时间值函数 135
5.5.1 ADD_MONTHS() 136
5.5.2 LAST_DAY() 137
5.5.3 MONTHS_BETWEEN() 137
5.5.4 NEXT_DAY() 138
5.5.5 ROUND() 138
5.5.6 SYSDATE 139
5.5.7 TRUNC() 139
5.6 使用时区 140
5.6.1 与时区有关的函数 140
5.6.2 数据库时区和会话时区 141
5.6.3 获取时区的时差 142
5.6.4 获取时区名 142
5.6.5 将时间值从一个时区转换为另一个时区 143
5.7 使用时间戳 143
5.7.1 使用时间戳类型 143
5.7.2 与时间戳有关的函数 147
5.8 使用时间间隔 152
5.8.1 使用 INTERVAL YEAR TO MONTH 类型 153
5.8.2 使用 INTERVAL DAY TO SECOND 类型 155
5.8.3 与时间间隔有关的函数 157
5.9 小结 158

第 6 章 子查询 159
6.1 子查询的类型 159
6.2 编写单行子查询 160
6.2.1 在 WHERE 子句中使用子查询 160
6.2.2 使用其他单行操作符 161
6.2.3 在 HAVING 子句中使用子查询 161
6.2.4 在 FROM 子句中使用子查询(内联视图) 162
6.2.5 可能碰到的错误 163

6.3 编写多行子查询 …… 164
6.3.1 在多行子查询中使用 IN 操作符 …… 165
6.3.2 在多行子查询中使用 ANY 操作符 …… 165
6.3.3 在多行子查询中使用 ALL 操作符 …… 166
6.4 编写多列子查询 …… 166
6.5 编写关联子查询 …… 167
6.5.1 关联子查询的例子 …… 167
6.5.2 在关联子查询中使用 EXISTS 和 NOT EXISTS …… 168
6.6 编写嵌套子查询 …… 170
6.7 编写包含子查询的 UPDATE 和 DELETE 语句 …… 172
6.7.1 编写包含子查询的 UPDATE 语句 …… 172
6.7.2 编写包含子查询的 DELETE 语句 …… 172
6.8 使用子查询因子化 …… 173
6.9 小结 …… 174

第 7 章 高级查询 …… 175
7.1 使用集合操作符 …… 176
7.1.1 示例表 …… 176
7.1.2 使用 UNION ALL 操作符 …… 177
7.1.3 使用 UNION 操作符 …… 178
7.1.4 使用 INTERSECT 操作符 …… 179
7.1.5 使用 MINUS 操作符 …… 179
7.1.6 组合使用集合操作符 …… 180
7.2 使用 TRANSLATE()函数 …… 182
7.3 使用 DECODE()函数 …… 183
7.4 使用 CASE 表达式 …… 185
7.4.1 使用简单 CASE 表达式 …… 185
7.4.2 使用搜索 CASE 表达式 …… 186
7.5 层次化查询 …… 187
7.5.1 示例数据 …… 187
7.5.2 使用 CONNECT BY 和 START WITH 子句 …… 189
7.5.3 使用伪列 LEVEL …… 190
7.5.4 格式化层次化查询的结果 …… 190
7.5.5 从非根节点开始遍历 …… 191
7.5.6 在 START WITH 子句中使用子查询 …… 192
7.5.7 从下向上遍历树 …… 192
7.5.8 从层次化查询中删除节点和分支 …… 193
7.5.9 在层次化查询中加入其他条件 …… 194
7.5.10 使用递归子查询因子化查询分层数据 …… 194
7.6 使用 ROLLUP 和 CUBE 子句 …… 198
7.6.1 示例表 …… 199
7.6.2 使用 ROLLUP 子句 …… 200
7.6.3 使用 CUBE 子句 …… 203
7.6.4 使用 GROUPING()函数 …… 204
7.6.5 使用 GROUPING SETS 子句 …… 207
7.6.6 使用 GROUPING_ID() 函数 …… 207
7.6.7 在 GROUP BY 子句中多次使用某个列 …… 209
7.6.8 使用 GROUP_ID()函数 …… 210
7.7 使用 CROSS APPLY 和 OUTER APPLY …… 211
7.7.1 CROSS APPLY …… 212
7.7.2 OUTER APPLY …… 212
7.8 使用 LATERAL …… 213
7.9 小结 …… 214

第 8 章 分析数据 …… 215
8.1 使用分析函数 …… 215
8.1.1 示例表 …… 216
8.1.2 使用评级函数 …… 217
8.1.3 使用反百分位函数 …… 223
8.1.4 使用窗口函数 …… 224

8.1.5 使用报表函数 ……230
8.1.6 使用 LAG()和 LEAD()函数 ……233
8.1.7 使用 FIRST 和 LAST 函数 ……234
8.1.8 使用线性回归函数 ……234
8.1.9 使用假想评级与分布函数 ……235
8.2 使用 MODEL 子句 ……236
8.2.1 MODEL 子句示例 ……236
8.2.2 用位置标记和符号标记访问数据单元 ……238
8.2.3 用 BETWEEN 和 AND 返回特定范围内的数据单元 ……239
8.2.4 用 ANY 和 IS ANY 访问所有的数据单元 ……239
8.2.5 用 CURRENTV()函数获取某个维度的当前值 ……239
8.2.6 用 FOR 循环访问数据单元 ……240
8.2.7 处理空值和缺失值 ……242
8.2.8 更新已有的单元 ……244
8.3 使用 PIVOT 和 UNPIVOT 子句 ……245
8.3.1 PIVOT 子句的简单示例 ……245
8.3.2 转换多个列 ……246
8.3.3 在转换中使用多个聚合函数 ……247
8.3.4 使用 UNPIVOT 子句 ……248
8.4 执行 Top-N 查询 ……249
8.4.1 使用 FETCH FIRST 子句 ……250
8.4.2 使用 OFFSET 子句 ……250
8.4.3 使用 PERCENT 子句 ……251
8.4.4 使用 WITH TIES 子句 ……252
8.5 在数据中发现模式 ……252
8.5.1 在 all_sales2 表中发现 V 形数据模式 ……253
8.5.2 在 all_sales3 表中发现 W 型数据模式 ……256
8.5.3 在 all_sales3 表中发现 V 形数据模式 ……257
8.6 小结 ……258

第 9 章 修改表的内容 ……259
9.1 使用 INSERT 语句添加行 ……260
9.1.1 省略列的列表 ……261
9.1.2 为列指定空值 ……261
9.1.3 在列值中使用单引号和双引号 ……261
9.1.4 从一个表向另一个表复制行 ……262
9.2 使用 UPDATE 语句修改行 ……262
9.3 使用 RETURNING 子句返回聚合函数的计算结果 ……263
9.4 使用 DELETE 语句删除行 ……264
9.5 数据库的完整性 ……264
9.5.1 主键约束 ……264
9.5.2 外键约束 ……265
9.6 使用默认值 ……266
9.7 使用 MERGE 合并行 ……267
9.8 数据库事务 ……269
9.8.1 事务的提交和回滚 ……269
9.8.2 事务的开始与结束 ……270
9.8.3 保存点 ……271
9.8.4 事务的 ACID 特性 ……272
9.8.5 并发事务 ……273
9.8.6 事务锁 ……274
9.8.7 事务隔离级别 ……274
9.8.8 SERIALIZABLE 事务隔离级别的一个例子 ……275
9.9 查询闪回 ……276
9.9.1 授权使用闪回 ……276
9.9.2 时间查询闪回 ……277
9.9.3 SCN 查询闪回 ……278
9.10 小结 ……280

第 10 章 用户、特权和角色 ……281
10.1 用户 ……282
10.1.1 创建用户 ……282
10.1.2 修改用户密码 ……283
10.1.3 删除用户 ……283
10.2 系统特权 ……284
10.2.1 向用户授予系统特权 ……284
10.2.2 检查授予用户的系统特权 ……285
10.2.3 使用系统特权 ……286
10.2.4 撤消用户的系统特权 ……286
10.3 对象特权 ……287
10.3.1 向用户授予对象特权 ……287
10.3.2 检查已授予的对象特权 ……288
10.3.3 检查已接受的对象特权 ……289
10.3.4 使用对象特权 ……291
10.3.5 创建同义词 ……291
10.3.6 创建公共同义词 ……292
10.3.7 撤消用户的对象特权 ……293
10.4 角色 ……293
10.4.1 创建角色 ……293
10.4.2 为角色授权 ……294
10.4.3 将角色授予用户 ……294
10.4.4 检查授予用户的角色 ……294
10.4.5 检查授予角色的系统特权 ……296
10.4.6 检查授予角色的对象特权 ……296
10.4.7 使用已授予角色的特权 ……298
10.4.8 启用和禁用角色 ……299
10.4.9 撤消角色 ……300
10.4.10 从角色中撤消特权 ……300
10.4.11 删除角色 ……300
10.5 审计 ……300
10.5.1 执行审计需要的特权 ……300
10.5.2 审计示例 ……301
10.5.3 审计跟踪视图 ……302
10.6 小结 ……303

第 11 章 创建表、序列、索引和视图 ……305
11.1 表 ……305
11.1.1 创建表 ……306
11.1.2 获得有关表的信息 ……307
11.1.3 获得表中列的信息 ……308
11.1.4 修改表 ……308
11.1.5 重命名表 ……317
11.1.6 向表中添加注释 ……317
11.1.7 截断表 ……318
11.1.8 删除表 ……318
11.1.9 使用 BINARY_FLOAT 和 BINARY_ DOUBLE 数据类型 ……319
11.1.10 使用 DEFAULT ON NULL 列 ……320
11.1.11 在表中使用可见及不可见列 ……321
11.2 序列 ……323
11.2.1 创建序列 ……324
11.2.2 获取有关序列的信息 ……325
11.2.3 使用序列 ……326
11.2.4 使用序列填充主键 ……328
11.2.5 使用序列指定默认列值 ……329
11.2.6 使用标识列 ……329
11.2.7 修改序列 ……330
11.2.8 删除序列 ……330
11.3 索引 ……331
11.3.1 创建 B-树索引 ……331
11.3.2 创建基于函数的索引 ……332
11.3.3 获取有关索引的信息 ……333
11.3.4 获取列索引的信息 ……333
11.3.5 修改索引 ……334

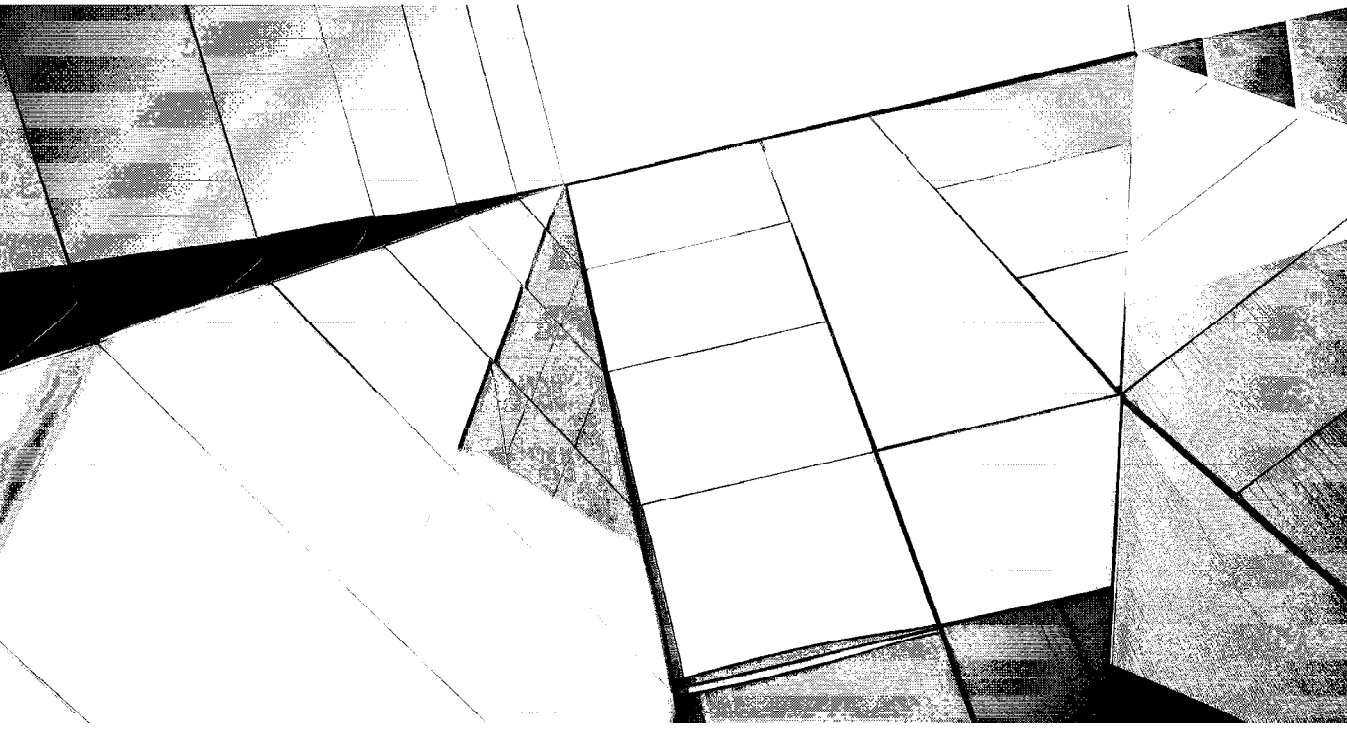
11.3.6 删除索引……334
11.3.7 创建位图索引……334
11.4 视图……335
11.4.1 创建并使用视图……336
11.4.2 修改视图……343
11.4.3 删除视图……343
11.4.4 在视图中使用可见列和不可见列……343
11.5 闪回数据归档……344
11.6 小结……347
第 12 章 PL/SQL 编程简介……349
12.1 块结构……350
12.2 变量和类型……351
12.3 条件逻辑……352
12.4 循环……352
12.4.1 简单循环……353
12.4.2 WHILE 循环……354
12.4.3 FOR 循环……354
12.5 游标……355
12.5.1 步骤(1)：声明用于保存列值的变量……355
12.5.2 步骤(2)：声明游标……355
12.5.3 步骤(3)：打开游标……356
12.5.4 步骤(4)：从游标中取得行……356
12.5.5 步骤(5)：关闭游标……357
12.5.6 完整的示例：product_cursor.sql……357
12.5.7 游标与 FOR 循环……358
12.5.8 OPEN-FOR 语句……359
12.5.9 无约束游标……361
12.6 异常……362
12.6.1 ZERO_DIVIDE 异常……364
12.6.2 DUP_VAL_ON_INDEX 异常……365
12.6.3 INVALID_NUMBER 异常……365
12.6.4 OTHERS 异常……365
12.7 过程……366
12.7.1 创建过程……366
12.7.2 调用过程……368
12.7.3 获取有关过程的信息……369
12.7.4 删除过程……370
12.7.5 查看过程中的错误……370
12.8 函数……371
12.8.1 创建函数……371
12.8.2 调用函数……372
12.8.3 获取有关函数的信息……373
12.8.4 删除函数……373
12.9 包……373
12.9.1 创建包的规范……373
12.9.2 创建包体……374
12.9.3 调用包中的函数和过程……375
12.9.4 获取有关包中函数和过程的信息……376
12.9.5 删除包……376
12.10 触发器……377
12.10.1 触发器启动的时机……377
12.10.2 设置示例触发器……377
12.10.3 创建触发器……377
12.10.4 启动触发器……379
12.10.5 获取有关触发器的信息……380
12.10.6 禁用和启用触发器……382
12.10.7 删除触发器……382
12.11 其他 PL/SQL 特性……382
12.11.1 SIMPLE_INTEGER 类型……382
12.11.2 在 PL/SQL 中使用序列……383
12.11.3 PL/SQL 本地机器代码生成……384
12.11.4 WITH 子句……385
12.12 小结……386

第 13 章 数据库对象……387
13.1 对象简介……387
13.2 创建对象类型……388
13.3 使用 DESCRIBE 获取有关对象类型的信息……390
13.4 在数据库表中使用对象类型……391
13.4.1 列对象……391
13.4.2 对象表……394
13.4.3 对象标识符和对象引用……397
13.4.4 比较对象值……399
13.5 在 PL/SQL 中使用对象……402
13.5.1 get_products()函数……403
13.5.2 display_product()过程……403
13.5.3 insert_product()过程……404
13.5.4 update_product_price()过程……405
13.5.5 get_product()函数……406
13.5.6 update_product()过程……406
13.5.7 get_product_ref()函数……407
13.5.8 delete_product()过程……408
13.5.9 product_lifecycle()过程……408
13.5.10 product_lifecycle2()过程……409
13.6 类型继承……410
13.6.1 运行脚本以创建第 2 个对象模式……411
13.6.2 继承属性……411
13.7 用子类型对象代替超类型对象……413
13.7.1 SQL 例子……413
13.7.2 PL/SQL 示例……414
13.7.3 NOT SUBSTITUTABLE 对象……415
13.8 其他有用的对象函数……416
13.8.1 IS OF()函数……416
13.8.2 TREAT()函数……419
13.8.3 SYS_TYPEID()函数……423
13.9 NOT INSTANTIABLE 对象类型……424
13.10 用户自定义的构造函数……425
13.11 重载方法……429
13.12 通用调用……430
13.12.1 运行脚本以创建第 3 个对象模式……431
13.12.2 继承属性……431
13.13 小结……432
第 14 章 集合……435
14.1 集合简介……435
14.2 创建集合类型……436
14.2.1 创建变长数组类型……436
14.2.2 创建嵌套表类型……437
14.3 使用集合类型定义表列……437
14.3.1 使用变长数组类型定义表列……437
14.3.2 使用嵌套表类型定义表列……438
14.4 获取集合信息……438
14.4.1 获取变长数组信息……438
14.4.2 获得嵌套表信息……439
14.5 填充集合元素……441
14.5.1 填充变长数组元素……441
14.5.2 填充嵌套表元素……441
14.6 检索集合元素……442
14.6.1 检索变长数组元素……442
14.6.2 检索嵌套表元素……443
14.7 使用 TABLE()函数将集合视为一系列行……443
14.7.1 将 TABLE()函数应用于变长数组……444
14.7.2 将 TABLE()函数应用于嵌套表……445
14.8 更改集合元素……445
14.8.1 更改变长数组元素……445
14.8.2 更改嵌套表元素……446

14.9 使用映射方法比较嵌套表的内容 447
14.10 使用 CAST()函数将集合从一种类型转换为另一种类型 449
14.10.1 使用 CAST()函数将变长数组转换为嵌套表 449
14.10.2 使用 CAST()函数将嵌套表转换为变长数组 450
14.11 在 PL/SQL 中使用集合 451
14.11.1 操作变长数组 451
14.11.2 操作嵌套表 453
14.11.3 PL/SQL 集合方法 455
14.12 创建和使用多级集合 464
14.12.1 运行脚本创建第二个集合模式 464
14.12.2 使用多级集合 464
14.13 Oracle Database 10*g* 对集合的增强 467
14.13.1 运行脚本以创建第三个集合模式 467
14.13.2 关联数组 467
14.13.3 更改元素类型的大小 468
14.13.4 增加变长数组中元素的数目 469
14.13.5 在临时表中使用变长数组 469
14.13.6 为嵌套表的存储表使用不同的表空间 469
14.13.7 嵌套表对 ANSI 的支持 470
14.14 小结 478
第 15 章 大对象 479
15.1 大对象(LOB)简介 480
15.2 示例文件 480
15.3 理解大对象类型 480
15.4 创建包含大对象的表 481
15.5 在 SQL 中使用大对象 483
15.5.1 使用 CLOB 和 BLOB 对象 483
15.5.2 使用 BFILE 对象 485
15.6 在 PL/SQL 中使用大对象 486
15.6.1 APPEND()方法 489
15.6.2 CLOSE()方法 489
15.6.3 COMPARE()方法 489
15.6.4 COPY()方法 491
15.6.5 CREATETEMPORARY()方法 492
15.6.6 ERASE()方法 492
15.6.7 FILECLOSE()方法 493
15.6.8 FILECLOSEALL()方法 493
15.6.9 FILEEXISTS()方法 494
15.6.10 FILEGETNAME()方法 494
15.6.11 FILEISOPEN()方法 495
15.6.12 FILEOPEN()方法 495
15.6.13 FREETEMPORARY()方法 496
15.6.14 GETCHUNKSIZE()方法 496
15.6.15 GETLENGTH()方法 497
15.6.16 GET_STORAGE_LIMIT()方法 497
15.6.17 INSTR()方法 498
15.6.18 ISOPEN()方法 499
15.6.19 ISTEMPORARY()方法 499
15.6.20 LOADFROMFILE()方法 500
15.6.21 LOADBLOBFROMFILE()方法 501

15.6.22 LOADCLOBFROMFILE()方法……502
15.6.23 OPEN()方法……503
15.6.24 READ()方法……503
15.6.25 SUBSTR()方法……504
15.6.26 TRIM()方法……505
15.6.27 WRITE()方法……506
15.6.28 WRITEAPPEND()方法……507
15.6.29 PL/SQL 示例过程……507
15.7 LONG 和 LONG RAW 类型……524
15.7.1 示例表……524
15.7.2 向 LONG 和 LONG RAW 列添加数据……525
15.7.3 将 LONG 和 LONG RAW 列转换为 LOB……525
15.8 Oracle Database 10g 对大对象的增强……526
15.8.1 CLOB 和 NCLOB 对象之间的隐式转换……527
15.8.2 在触发器中使用 LOB 时:new 属性的用法……528
15.9 Oracle Database 11g 对大对象的增强……528
15.9.1 加密 LOB 数据……528
15.9.2 压缩 LOB 数据……532
15.9.3 删除 LOB 重复数据……533
15.10 Oracle Database 12c 对大对象的增强……533
15.11 小结……534
第 16 章 SQL 优化……535
16.1 SQL 优化简介……535
16.2 使用 WHERE 子句过滤行……536
16.3 使用表连接而不是多个查询……536
16.4 执行连接时使用完全限定的列引用……537
16.5 使用 CASE 表达式而不是多个查询……538
16.6 添加表索引……539
16.6.1 何时创建 B-树索引……539
16.6.2 何时创建位图索引……540
16.7 使用 WHERE 而不是 HAVING……540
16.8 使用 UNION ALL 而不是 UNION……541
16.9 使用 EXISTS 而不是 IN……542
16.10 使用 EXISTS 而不是 DISTINCT……543
16.11 使用 GROUPING SETS 而不是 CUBE……543
16.12 使用绑定变量……543
16.12.1 不相同的 SQL 语句……544
16.12.2 使用绑定变量定义相同的 SQL 语句……544
16.12.3 列出和输出绑定变量……545
16.12.4 使用绑定变量存储 PL/SQL 函数的返回值……545
16.12.5 使用绑定变量存储来自 REFCURSOR 的返回值……546
16.13 比较执行查询的成本……546
16.13.1 检查执行计划……547
16.13.2 比较执行计划……552
16.14 为优化器传递提示……553
16.15 其他优化工具……554
16.15.1 Oracle Enterprise Manager……555
16.15.2 Automatic Database Diagnostic Monitor……555
16.16 小结……556
第 17 章 XML 和 Oracle 数据库……557
17.1 XML 简介……557

17.2 从关系数据生成 XML……558
17.2.1 XMLELEMENT()函数……558
17.2.2 XMLATTRIBUTES()函数……561
17.2.3 XMLFOREST()函数……562
17.2.4 XMLAGG()函数……563
17.2.5 XMLCOLATTVAL()函数……565
17.2.6 XMLCONCAT()函数……566
17.2.7 XMLPARSE()函数……566
17.2.8 XMLPI()函数……567
17.2.9 XMLCOMMENT()函数……567
17.2.10 XMLSEQUENCE()函数……568
17.2.11 XMLSERIALIZE()函数……569
17.2.12 PL/SQL 示例：将 XML 数据写入文件……569
17.2.13 XMLQUERY()函数……571
17.3 将 XML 保存到数据库中……575
17.3.1 示例 XML 文件……575
17.3.2 创建示例 XML 模式……576
17.3.3 从示例 XML 模式检索信息……577
17.3.4 更新示例 XML 模式中的信息……582
17.4 小结……585
附录 Oracle 数据类型……587



第 1 章

简　介

本章内容包括：

- **关系数据库简介**
- **结构化查询语言**(Structured Query Language，SQL)**简介，**SQL **用于访问数据库**
- **使用** Oracle **中基于文本的交互式工具** SQL*Plus **来运行** SQL **语句**
- **介绍** SQL Developer**，它是用于数据库开发的一种图形化工具**
- **简要介绍** Oracle **提供的过程语言** PL/SQL**，**PL/SQL **包含编程语句**

1.1　关系数据库简介

关系数据库(relational database)**的概念最初由** E.F. Codd **博士于** 1970 **年提出。他在**

Communications of the ACM(Association for Computing Machinery，美国计算机学会)1970 年 6 月第 13 卷第 6 期上发表了一篇题为 A Relational Model of Data for Large Shared Data Banks(大型共享数据库的关系数据模型)的重要论文，提出了关系数据库的理论。

关系数据库的基本概念非常简单易懂，它是一组已经被组织为表(table)结构的相关信息的集合。每个表都包含多行(row)数据，这些行数据又被进一步分为多列(column)。这些表在数据库中都被存储在称为模式(schema)的结构中，所谓模式就是数据库用户可以存储表的地方。每个用户都可以为其他用户授予访问自己的表的权限(permission)。

大部分人对于表中存储的数据都非常熟悉，例如，股票价格和列车时刻表有时以表的形式存储。本书使用的一个例子是一个虚拟商店用来记录顾客信息的表，该表存储了顾客的名、姓、生日和电话号码：

```
FIRST_NAME  LAST_NAME  DOB          PHONE
----------- --------- ------------ ------------
John        Brown     01-JAN-1965  800-555-1211
Cynthia     Green     05-FEB-1968  800-555-1212
Steve       White     16-MAR-1971  800-555-1213
Gail        Black                  800-555-1214
Doreen      Blue      20-MAY-1970
```

这个表可以存储为很多种形式：

? 数据库中的表

? 网页中的 HTML 文件

? 档案柜中的纸片

需要注意的一个要点是：构成数据库的信息与用来访问这些信息的系统并不是一回事。用来访问数据库的系统软件称为数据库管理系统(Database Management System，DBMS)。Oracle Database 12*c* 就属于这样一种软件；其他数据库管理系统还包括 Microsoft SQL Server、DB2 和开源的 MySQL。

当然，每个数据库都必须有一些方法来向数据库中存储数据和从数据库中读取数据，这最好是使用所有数据库都能理解的一种通用语言来进行。现在的数据库管理系统实现了一种称为结构化查询语言(Structured Query Language，SQL)的标准语言。可以使用 SQL 来检索、添加、修改和删除数据库中的信息。

1.2 SQL 简介

SQL 是用于访问关系数据库的标准语言。SQL 应该按字母 S-Q-L 来发音。

注意：

根据美国国家标准化组织(American National Standards Institute，ANSI)的规定，SQL 的正确发音是 S-Q-L，但是也常常发音为单一的单词“ sequel ”。

SQL 是在 E.F. Codd 博士突破性工作的基础上发展起来的，其第一个实现由 IBM 在 20 世纪 70 年代中期开发完成。当时 IBM 开展了一个称为 System R 的研究项目，SQL 就是从这个项目中诞生的。后来到 1979 年，一家当时名为 Relational Software Inc.的公司(也就是现在的

Oracle 公司)发布了第一个商业版本的 SQL。

1986 年，SQL 成为美国国家标准化组织(ANSI)的一项标准，但每个软件公司的 SQL 实现之间存在一些差异。

SQL 使用一种很简单的语法，非常易于学习和使用。本章通过几个简单的例子来介绍 SQL 的用法。SQL 语句可分为 5 类，简要概括如下：

- 查询语句 用于检索数据库表中存储的行。可以使用 SQL 的 SELECT 语句编写查询语句。
- 数据操纵语言(Data Manipulation Language，DML)语句 用于修改表的内容。DML 语句有 3 种：
  - **INSERT** 向表中添加行。
  - **UPDATE** 修改行的内容。
  - **DELETE** 删除行。
- 数据定义语言(Data Definition Language，DDL)语句 用于定义构成数据库的数据结构，例如表。DDL 语句有 5 种基本类型：
  - **CREATE** 创建数据库结构。例如，CREATE TABLE 语句用于创建表；另外一个例子是 CREATE USER，用于创建数据库用户。
  - **ALTER** 修改数据库结构。例如，ALTER TABLE 语句用于修改表。
  - **DROP** 删除数据库结构。例如，DROP TABLE 语句用于删除表。
  - **RENAME** 更改表名。
  - **TRUNCATE** 删除表的全部内容。
- 事务控制(Transaction Control，TC)语句 用于将对行所做的修改永久性地保存，或者取消这些修改操作。TC 语句有 3 种：
  - **COMMIT** 永久性地保存对行所做的修改。
  - **ROLLBACK** 取消对行所做的修改。
  - **SAVEPOINT** 设置“保存点”，可以将对行所做的修改回滚到此处。
- 数据控制语言(Data Control Language，DCL)语句 用于修改数据库结构的操作权限。DCL 语句有两种：
  - **GRANT** 授予某个用户对指定的数据库结构的访问权限。
  - **REVOKE** 阻止某个用户访问指定的数据库结构。

Oracle 有一个名为 SQL*Plus 的程序，使用这个程序可以输入 SQL 语句，并获取从数据库返回的结果。SQL*Plus 也可以运行包含 SQL 语句和 SQL*Plus 命令的脚本。

还有其他方法可以运行 SQL 语句，并从数据库中获取返回结果。例如，使用 Oracle Forms 和 Oracle Reports 都可以运行 SQL 语句。SQL 语句也可以嵌入在使用诸如 Java 和 C#之类的语言编写的程序中。如何在 Java 程序中添加 SQL 语句的细节请参考作者的 *Oracle 9i JDBC Programming* (2002 年，Oracle 出版社)一书。如何在 C#程序中添加 SQL 语句的细节请参考作者的 *Mastering C# Database Programming* (Sybex, 2003)一书。

1.3 使用 SQL*Plus

在本节中，将介绍如何启动 SQL*Plus 和运行查询。

1.3.1 启动 SQL*Plus

如果使用的是 Windows 7，请单击 Start 菜单，并选择 All Programs | Oracle | Application Development | SQL Plus 来启动 SQL*Plus。如果使用的是 UNIX 或 Linux，可以在命令行提示符下运行 sqlplus 来启动 SQL*Plus。

图 1-1 显示了在 Windows 7 上运行的 SQL*Plus。

图 1-1

图 1-1 显示 scott 用户连接到了数据库，scott 用户是许多 Oracle 数据库安装时都包含的一个用户，在本人的数据库中，scott 的密码是 oracle。

@字符后面的主机字符串告诉 SQL*Plus 连接到哪个数据库。如果是在自己的本地计算机上运行数据库，一般可以忽略主机字符串。例如，可以输入 scott/oracle 并忽略@字符和 orcl 字符串。如果忽略主机字符串，SQL*Plus 就会尝试连接到正运行 SQL*Plus 的本地计算机上的数据库。如果数据库不是在本地计算机上运行的，那么应该咨询一下数据库管理员(DBA)，以获得主机字符串。

如果在你的数据库中 scott 用户不存在或被锁定，请向 DBA 申请另一用户和密码(对于本章第一部分的示例，可以使用任何用户来连接到数据库)。

1.3.2 从命令行启动 SQL*Plus

要从命令行启动 SQL*Plus，可以使用 sqlplus 命令。sqlplus 命令的完整语法如下：

```
sqlplus [user_name[/password[@ host_string]]]
```

其中：

- user_name 指定数据库用户的名称。
- password 指定数据库用户的密码。
- host_string 指定要连接的数据库。

下面是执行 sqlplus 命令的几个例子：

```
sqlplus scott/oracle
sqlplus scott/oracle@orcl
```

如果你在 Windows 操作系统上使用的是 SQL*Plus，那么 Oracle 安装程序会自动将 SQL*Plus 的目录添加到路径中。如果使用的是 UNIX 或 Linux，那么运行 SQL*Plus 有以下两种选择：

? 使用 cd 命令转到 sqlplus 可执行文件所在的目录，并在此目录路径中运行 sqlplus。
? 将 sqlplus 所在的目录添加到路径中，然后运行 sqlplus。如果需要有关设置目录路径的帮助，请咨询系统管理员。

为安全起见，在连接到数据库时可以隐藏密码。例如，可以输入下面的命令：

```
sqlplus scott@orcl
```

SQL*Plus 会提示用户输入密码。输入密码时，密码会隐藏起来。
也可以只输入如下命令：

```
sqlplus
```

SQL*Plus 会提示输入用户名和密码。通过将主机字符串添加到用户名可以指定主机字符串(例如 scott@orcl)。

1.3.3 使用 SQL*Plus 执行 SELECT 语句

使用 SQL*Plus 登录到数据库之后，输入下面的 SELECT 语句，这条语句会返回当前日期：

```
SELECT SYSDATE FROM dual;
```

注意：
如果你想跟着例子练习，本书中用粗体显示的 SQL 语句是应当输入并运行的。非粗体显示的语句不需要输入。

SYSDATE 是一个内置的数据库函数，它返回当前日期；dual 表只包含单个 dummy 行。下一章将讲述有关 dual 表的更多信息。

注意：
可以使用分号(;)字符终止 SQL 语句。

图 1-2 显示了上一条 SELECT 语句返回的日期。

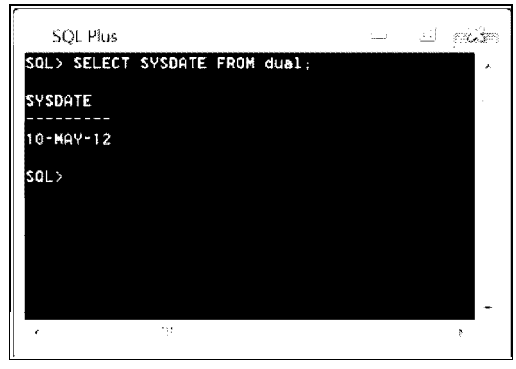


图 1-2

通过输入 EDIT 命令，可以编辑 SQL*Plus 中的最后一条 SQL 语句。在输错 SQL 语句或想修改 SQL 语句时，这项功能非常有用。在 Windows 系统中输入 EDIT 命令后，会启动记事本应用程序。在退出记事本并保存 SQL 语句时，一条新的 SQL 语句就会被传递到 SQL*Plus 中。可以通过输入一

个斜杠(/)重新执行该 SQL 语句。在 Linux 或 UNIX 系统中，默认的编辑器一般是 ed。要保存修改的语句并退出 ed，请输入 wq。

解决尝试编辑报表时出现的错误

尝试在 Windows 中编辑语句时，如果遇到错误 SP2-0110，可以管理员身份运行 SQL*Plus。在 Windows 7 中，做到这一点的方法是右击 SQL*Plus 的快捷方式，并选择“ Run as administrator ”选项。在 Windows 7 中，通过右击 SQL*Plus 的快捷方式，在 Compatibility 选项卡中选择“ Run as administrator ”选项，可以永久保存这个设置。

还可以通过右击 SQL*Plus 的快捷方式并在 Shortcut 选项卡中修改“ Start in ”目录来设置 SQL*Plus 的启动目录。SQL*Plus 在保存和检索文件时将使用该默认目录。例如，可以将目录设置为 C:\My_SQL_files，并且 SQL * Plus 会默认在该目录中存储和检索文件。

在 Windows 版本的 SQL* Plus 中，可以通过按键盘上的向上和向下箭头键来滚动过去曾经运行的命令。

第 3 章将会介绍更多有关 SQL*Plus 的内容。

1.4 使用 SQL Developer

也可以使用 SQL Developer 来输入 SQL 语句。SQL Developer 有图形用户界面，在其中可以输入 SQL 语句、检查数据库表、运行脚本、编辑并调试 PL/SQL 代码，还能完成更多的任务。SQL Developer 可以连接到 9.2.0.1 和更高版本的 Oracle 数据库，可以在许多操作系统中运行。图 1-3 展示了运行中的 SQL Developer。

图 1-3

必须下载包含 Java 软件开发包(Software Development Kit，SDK)的 SQL Developer 版本，

或者必须预先已经将正确版本的 Java 安装在计算机上。根据不同的 SQL Developer 版本，需要的 Java 版本也不同，应当访问 www.oracle.com 上的 SQL Developer 网页来了解详细情况。

成功启动 SQL Developer 后，需要通过右击 Connections 并选择 New Connection 来创建数据库连接，SQL Developer 将显示一个对话框，可以在其中指定数据库连接的详情。下面的图 1-4 显示了连接详情填写完毕后的示例对话框。

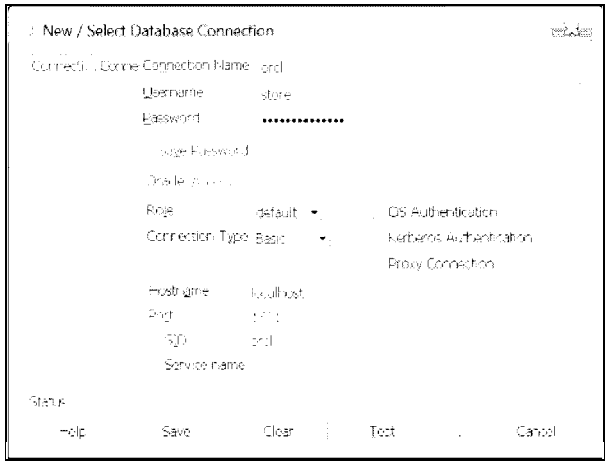


图 1-4

创建连接和测试之后，就可以使用 SQL Developer 检查数据库表和运行查询。下面的图 1-5 展示了名为 customers 的数据库表中的各列，这是本书中用到的一个表。

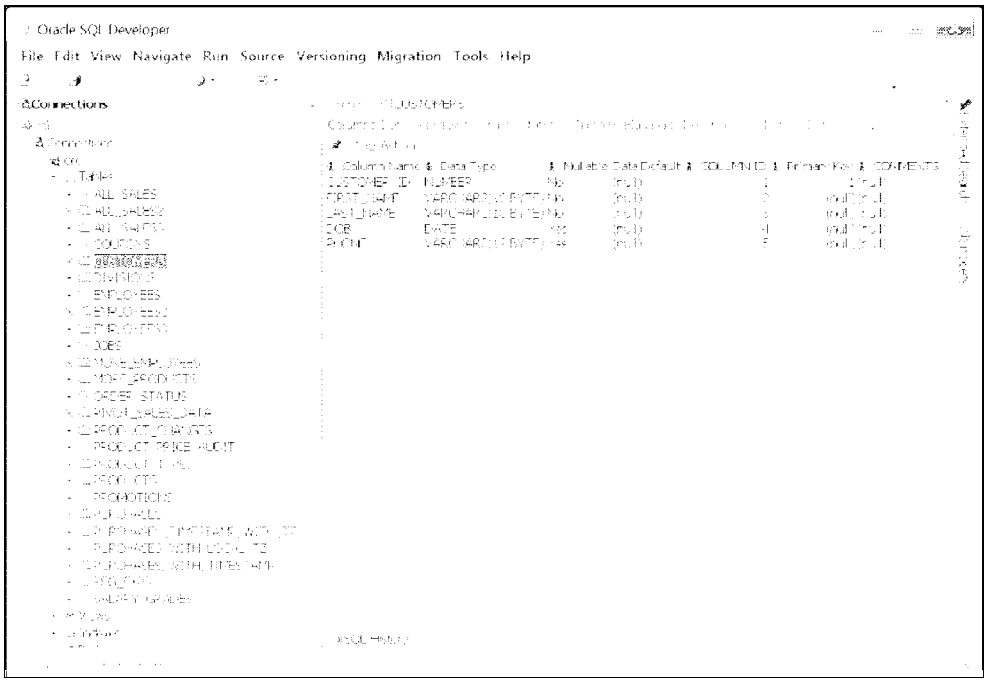


图 1-5

也可以通过选择 Data 选项卡查看表中存储的数据，图 1-6 中显示了 customers 表的数据行。

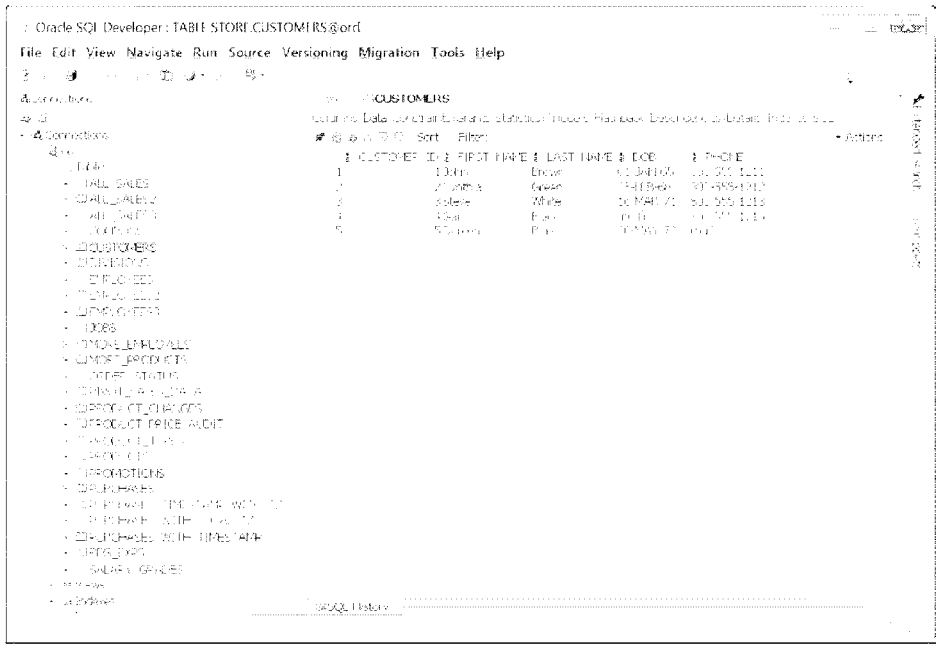


图 1-6

下一节将讨论如何创建本书中使用的 store 数据库模式。

1.5 创建 store 模式

这个假想的商店销售书籍、录像、DVD、CD 之类的商品。此商店的数据库将存放客户、员工、产品和销售的有关信息。创建数据库的 SQL*Plus 脚本命名为 store_schema.sql，它位于解压本书的压缩文件后的 SQL 目录中。store_schema.sql 脚本包含了用来创建 store 模式的 DDL 和 DML 语句。

1.5.1 检查脚本

在编辑器中打开 store_schema.sql 脚本，并检查脚本中的语句。本节介绍脚本中的语句，并引导你完成可能需要对脚本做的任何修改。在本章后面，你将了解更多有关这些脚本语句的信息。

1. 删除和创建用户

store_schema.sql 脚本中的第一条可执行语句如下：

```
DROP USER store CASCADE;
```

这里的 DROP USER 语句是为了在本书后面重新创建该模式时不必手动删除 store 用户。接下来的语句创建 store 用户，密码是 store_password：

```
CREATE USER store IDENTIFIED BY store_password;
```

接下来的语句允许 store 用户连接到数据库并允许创建数据库条目：

```
GRANT connect, resource TO store;
```

2. 分配表空间存储

下面的语句为 store 用户在 users 表空间分配 10MB 的空间：

```
ALTER USER store QUOTA 10M ON users;
```

表空间是数据库用来存储表和其他数据库对象的地方。你将在第 10 章了解更多关于表空间的信息。大多数数据库都有 users 表空间来存储用户数据。要验证这一点，首先以特权用户(例如，system 用户)的身份连接到数据库，然后执行下面的语句：

```
SELECT property_value
FROM database_properties
WHERE property_name = 'DEFAULT_PERMANENT_TABLESPACE';

PROPERTY_VALUE
--------------
USERS
```

此查询返回在 ALTER USER 语句中使用的表空间的名称。在本人的数据库中，此表空间是 users。

如果该查询返回的表空间名不是 users，就必须用前面的查询返回的名字替换脚本的 ALTER USER 语句中的 users。例如，如果表空间的名称是 another_ts，那么将脚本中的这条语句改为：

```
ALTER USER store QUOTA 10M ON another_ts;
```

3. 设置连接

脚本中的以下语句以 store 用户连接：

```
CONNECT store/store_password;
```

如果正在连接其他计算机上的数据库，需要修改脚本中的 CONNECT 语句。例如，如果要连接到名为 orcl 的数据库，就要将脚本中的 CONNECT 语句改为：

```
CONNECT store/store_password@orcl;
```

4. 使用可插拔数据库功能

可插拔数据库(pluggable database)是 Oracle Database 12*c* 中的一项新功能。可插拔数据库在外部容器数据库中创建。可插拔数据库可节省系统资源、简化系统管理，并通常由数据库管理员执行。因此，对可插拔数据库的全面讨论超出了本书的范围。

如果正在使用可插拔数据库功能，就需要修改脚本中的 CONNECT 语句，使之包括可插拔数据库的名称。例如，如果可插拔数据库的名称为 pdborcl，那么将该语句改为：

```
CONNECT store/store_password@pdborcl;
```

如果对 store_schema.sql 脚本做了任何修改，请保存修改后的脚本。

脚本中的其余语句创建示例 store 所需的表和其他条目。本章后面将解释这些语句。

1.5.2 运行脚本

执行以下步骤来创建 store 模式：

(1) 启动 SQL*Plus。

(2) 以具有创建新用户、表和 PL/SQL 包权限的用户身份登录到数据库。在本人的数据库中，以 system 用户身份运行脚本，此用户具有所要求的全部权限。

(3) 如果使用可插拔数据库功能，那么必须将会话数据库容器设置为可插拔数据库名。例如，如果可插拔数据库名为 pdborcl，就运行下面的命令：

```
ALTER SESSION SET CONTAINER=pdborcl;
```

(4) 使用@命令运行 store_schema.sql 脚本。@命令的语法如下：

```
@ directory\store_schema.sql
```

其中，directory 是 store_schema.sql 脚本所在的目录。

例如，如果这个脚本保存在 C:\sql_book\SQL 中，那么应该输入：

```
@ C:\sql_book\SQL\store_schema.sql
```

如果 store_schema.sql 脚本存放在包含空格的目录中，那么必须在@命令之后将目录和脚本置于引号中。例如：

```
@ "C:\Oracle SQL book\sql_book\SQL\store_schema.sql"
```

如果使用的是 UNIX 或 Linux，并且将这个脚本保存到了 tmp 文件系统的 SQL 目录中，那么应该输入：

```
@ /tmp/SQL/store_schema.sql
```

注意：

Windows 在目录路径中使用反斜杠字符(\)，而 UNIX 和 Linux 则使用正斜杠字符(/)。

运行完 store_schema.sql 脚本后，用户将以 store 用户身份连接到数据库。这个用户的密码是 store_password。

在本书后面会要求运行其他脚本。在运行每个脚本前都需要执行本节描述的步骤：

- 如果数据库中没有 users 表空间，就要编辑脚本中的 ALTER USER 语句。
- 如果需要设置主机字符串来连接到某个数据库，就要编辑脚本中的 CONNECT 语句。
- 如果使用可插拔数据库功能，就要编辑脚本中的 CONNECT 语句并且在运行脚本前先执行 ALTER SESSION SET CONTAINER 命令。

现在不必编辑所有的脚本，只要记得每个脚本在运行前可能必须修改就行了。

1.5.3　用来创建 store 模式的 DDL 语句

数据定义语言(Data Definition Language，DDL)语句用于创建用户和表，以及数据库中的各种其他类型的结构。本节将介绍如何使用 DDL 语句创建 store 用户和表。

注意：

本章其余的 SQL 语句与 store_schema.sql 脚本中包含的语句完全相同。不必自己输入这些语句。

接下来将介绍以下内容：

? 如何创建数据库用户
? Oracle 数据库中常用的数据类型
? 虚拟商店使用的一些表

1. 创建数据库用户

要在数据库中创建用户，使用 CREATE USER 语句。CREATE USER 语句的简化语法如下：

```
CREATE USER user_name IDENTIFIED BY password;
```

其中：

? user_name 是用户名
? password 是用户的密码

例如，下面的 CREATE USER 语句创建了 store 用户，密码为 store_password：

```
CREATE USER store IDENTIFIED BY store_password;
```

如果想让这个用户操作数据库，就必须为该用户授予必需的权限。在 store 用户示例中，用户必须能够登录数据库(这需要 connect 权限)，而且能够创建一些诸如数据库表之类的条目(这需要 resource 权限)。权限是由特权用户(例如 system 用户)使用 GRANT 语句授予的。

下面这个例子为 store 用户授予了 connect 和 resource 权限：

```
GRANT connect, resource TO store;
```

本书中的许多例子都使用 store 模式。在开始详细介绍 store 所需要的表之前，首先需要理解 Oracle 数据库中的常用类型。

2. Oracle 数据库中的常用类型

有很多类型可以用来处理 Oracle 数据库中的数据。部分常用的数据类型如表 1-1 所示。本书附录中列出了所有的数据类型。

表 1-1　常用的 Oracle 数据类型

Oracle 数据类型	含　义
CHAR(length)	存储固定长度的字符串。length 参数指定字符串的长度。如果要存储的字符串长度较小，就在末尾填充空格。例如，CHAR(2)可以存储两字符的固定长度的字符串；如果使用该方式来存储'C'，就会在末尾添加一个空格字符。而'CA'则照原样存储，不用添加任何空格

(续表)

Oracle 数据类型	含　义
VARCHAR2(length)	存储可变长度的字符串。length 参数指定字符串的最大长度。例如，VARCHAR2(20)可以用来存储长度最大为 20 个字符的字符串。即使字符串的长度较小，也不用在末尾填充空格
DATE	存储日期和时间。DATE 类型存储的是纪元、4 位的年/月/日/时(24 小时格式)、分和秒。DATA 类型可以用来存储从公元前 4712 年 1 月 1 日到公元 9999 年 12 月 31 日之间的时间
INTEGER	存储整数。整数不包括浮点，而必须是整数数字，例如 1、10 和 115
NUMBER(precision, scale)	存储浮点数。precision(精度)是这个数字可以使用的最大位数，这个位数包括小数点之前的部分和小数点之后的部分。Oracle 数据库支持的最大精度是 38。scale 是小数点右边的最大位数。如果既没有指定 precision，也没有指定 scale，那么可以存储 38 位精度的数字。精度超过 precision 的数字不能存储到数据库中

下面给出了几个例子，显示 NUMBER 类型的数字在数据库中是如何存储的：

格　式	输入的数字	实际存储的数字
NUMBER	1234.567	1234.567
NUMBER(6, 2)	123.4567	123.46
NUMBER(6, 2)	12345.67	输入的数字超过指定的精度，因此数据库无法存储

3. store 模式中的表

下面将会介绍如何创建 store 模式使用的表。store 模式用于存储虚拟商店的详细信息，包括：

- 顾客的详细信息
- 销售的产品类型
- 产品的详细信息
- 顾客购买产品的历史记录
- 虚拟商店中员工的信息
- 工资等级

这些信息使用下列表来存储：

- customers 存储顾客的详细信息
- product_types 存储商店销售的产品类型
- products 存储产品的详细信息
- purchases 存储哪些顾客购买了哪些产品
- employees 存储员工的详细信息
- salary_grades 存储工资等级的详细信息

接下来将会介绍 store 模式中表的详细信息，以及 store_chema.sql 脚本中用来创建这些表的 CREATE TABLE 语句。

customers 表　customers 表用来存储虚拟商店的顾客的详细信息。该表会为商店的每个顾

客存储以下信息：

- 名
- 姓
- 生日(dob)
- 电话号码

每个条目占用 customers 表的一列，而 customers 表是由 store_schema.sql 脚本使用下面的 CREATE TABLE 语句创建的：

```
CREATE TABLE customers (
   customer_id INTEGER CONSTRAINT customers_pk PRIMARY KEY,
   first_name VARCHAR2(10) NOT NULL,
   last_name VARCHAR2(10) NOT NULL,
   dob DATE,
   phone VARCHAR2(12)
);
```

可以看到，customers 表包含 5 列，除了前面列表中给出的 4 个条目各占一列外，还有另外一列，名为 customer_id。下面详细介绍各列的内容：

- **customer_id** 为该表中的每行存储唯一的整数。每个表都有一列或多列来唯一地标识表中的每行，这样的列称为主键(primary key)。CREATE TABLE 语句中的 CONSTRAINT 子句表示 customer_id 列是该表的主键。CONSTRAINT 子句用来限制存储在列中的值，对于 customer_id 列，PRIMARY KEY 关键字表明每行的 customer_id 列必须包含唯一的值。约束可以使用可选名称，可选名称必须紧跟在 CONSTRAINT 关键字之后——在本例中，约束名是 customers_pk。应该始终命名主键约束，这样当发生约束错误时，就很容易确定错误发生在什么地方。
- **first_name** 存储顾客的名。注意 first_name 列使用了 NOT NULL 约束——这意味着当添加一行时，必须为 first_name 列提供值。如果不使用这个约束，用户就可以不提供值。
- **last_name** 存储顾客的姓。本列也使用 NOT NULL 约束，因此当添加一行时必须提供值。
- **dob** 存储顾客的生日。由于没有为本列指定 NOT NULL 约束，因此使用默认的 NULL 约束，即当添加一行时该列的值是可选的。
- **phone** 存储顾客的电话号码。这个列也没有使用 NOT NULL 约束。

store_schema.sql 脚本向 customers 表填充以下行：

```
CUSTOMER_ID FIRST_NAME  LAST_NAME   DOB       PHONE
----------- ---------- ---------- --------- ------------
          1 John       Brown      01-JAN-65 800-555-1211
          2 Cynthia    Green      05-FEB-68 800-555-1212
          3 Steve      White      16-MAR-71 800-555-1213
          4 Gail       Black                800-555-1214
          5 Doreen     Blue       20-MAY-70
```

注意，customer_id 为 4 的顾客生日为空，customer_id 为 5 的顾客电话号码为空。

可以使用 SQL*Plus 执行下面的 SELECT 语句来查看 customers 表中的数据行：

```
SELECT * FROM customers;
```

星号字符(*)表示要检索 customers 表中的所有列。

product_types 表 product_types 表用来存储商店销售的产品类型。product_types 表是由 store_schema.sql 脚本使用下面的 CREATE TABLE 语句创建的：

```
CREATE TABLE product_types (
   product_type_id INTEGER CONSTRAINT product_types_pk PRIMARY KEY,
   name VARCHAR2(10) NOT NULL
);
```

product_types 表包含如下两列：

? **product_type_id 唯一地标识该表中的每一行。**product_type_id **列是该表的主键。** product_types **表中每一行的** product_type_id **列都必须有唯一的整数值。**

? **name 包含的是产品类型。这是** NOT NULL **列，因此当添加一行时必须提供值。**

store_schema.sql 脚本向 product_types 表填充以下行：

```
PRODUCT_TYPE_ID NAME
--------------- ----------
              1 Book
              2 Video
              3 DVD
              4 CD
              5 Magazine
```

该表定义了商店销售的产品类型，商店销售的每个产品都必须是这 5 种类型中的一种。可以使用 SQL*Plus 执行下面的 SELECT 语句来查看 product_types 表中的行：

```
SELECT * FROM product_types;
```

products 表 products 表用来存储本商店销售产品的详细信息。对于每个产品都要存储以下信息：

? **产品类型**

? **产品名称**

? **产品介绍**

? **产品价格**

products 表是由 store_schema.sql 脚本使用下面的 CREATE TABLE 语句创建的：

```
CREATE TABLE products (
 product_id INTEGER CONSTRAINT products_pk PRIMARY KEY,
 product_type_id INTEGER
   CONSTRAINT products_fk_product_types
   REFERENCES product_types(product_type_id),
 name VARCHAR2(30) NOT NULL,
 description VARCHAR2(50),
 price NUMBER(5, 2)
);
```

products 表包含以下 5 列：

- **product_id 唯一标识该表中的每一行，这一列是该表的主键。**
- **product_type_id 为每个产品都关联一种产品类型。该列是对** product_types **表中** product_type_id **列的引用，称为外键**(foreign key)**，因为它引用了其他表中的列。包含这个外键的表**(products **表**)**称为明细表**(detail table)**或子表**(child table)**，被引用的表**(product_types **表**)**称为主表**(master table)**或父表**(parent table)**。这种关系称为主从关系**(master-detail relationship)**或父子关系**(parent-child relationship)**。当添加新产品时，应该通过在** product_type_id **列中提供匹配的** product_types.product_type_id **值，为这种产品关联一种类型**(**稍后将会看到示例**)**。**
- **name 存储产品名称。这是** NOT NULL **列。**
- **description 存储产品的介绍信息，可选。**
- **price 存储产品的价格，可选。该列类型为** NUMBER(5,2)**：精度是** 5**，因此这个数字最大可以是** 5 **位；小数部分是** 2**，因此这** 5 **位中最多只能有两位在小数点的右边。**

下面是 products **表前** 4 **行的数据：**

```
PRODUCT_ID PRODUCT_TYPE_ID NAME          DESCRIPTION       PRICE
---------- --------------- ------------  ------------ ----------
         1               1 Modern        A                 19.95
                             Science     description
                                         Of modern
                                         science

         2               1 Chemistry     Introduction         30
                                         to Chemistry

         3               2 Supernova     A star            25.99
                                         explodes

         4               2 Tank War      Action movie      13.95
                                         about a
                                         future war
```

products **表中第** 1 **行的** product_type_id **值为** 1**，说明该产品是一本书**(**这个** product_type_id **值与** product_types **表中书籍的产品类型匹配**)**。第** 2 **行也表示一本书，而第** 3 **行和第** 4 **行则表示录像**(**它们的** product_type_id **值是** 2**，这与** product_types **表中录像的产品类型匹配**)**。**

可以使用 SQL*Plus **执行下面的** SELECT **语句来查看** products **表中的所有行：**

```
SELECT * FROM products;
```

purchases **表** purchases **表用来存储顾客的购买记录。顾客每次购买产品时，都要存储以下信息：**

- **产品** ID
- **顾客** ID
- **顾客购买的产品数量**

purchases **表是由** store_schema.sql **脚本使用下面的** CREATE TABLE **语句创建的：**

```
CREATE TABLE purchases (
 product_id INTEGER
```

```
    CONSTRAINT purchases_fk_products
    REFERENCES products(product_id),
  customer_id INTEGER
    CONSTRAINT purchases_fk_customers
    REFERENCES customers(customer_id),
  quantity INTEGER NOT NULL,
  CONSTRAINT purchases_pk PRIMARY KEY (product_id, customer_id)
);
```

purchases 表包含如下 3 列：

? **product_id** 存储所购买产品的 ID，该值必须与 products 表中某一行的 product_id 列的值匹配。

? **customer_id** 存储所购买产品的顾客 ID，该值必须与 customers 表中某一行的 customer_id 列的值匹配。

? **quantity** 存储所购买产品的数量。

purchases 表有名为 purchases_pk 的主键约束，该主键约束由表中的两列组成：product_id 和 customer_id。对于该表中的每行来说，这两列值的组合必须唯一。当主键由多列组成时，称为复合主键。

下面显示的是 purchases 表的前 5 行数据：

```
PRODUCT_ID CUSTOMER_ID   QUANTITY
---------- ----------- ----------
         1           1          1
         2           1          3
         1           4          1
         2           2          1
         1           3          1
```

可以看到，product_id 和 customer_id 这两列的值的组合对于每一行都是唯一的。

可以使用 SQL*Plus 执行下面的 SELECT 语句来查看 purchases 表中的所有行：

```
SELECT * FROM purchases;
```

employees 表 employees 表用来存储商店中员工的详细信息，包括以下内容：

? 员工 ID

? 该员工的上级管理者的员工 ID(假设该员工有上级管理者)

? 名

? 姓

? 职位

? 工资

employees 表是由 store_schema.sql 脚本使用下面的 CREATE TABLE 语句创建的：

```
CREATE TABLE employees (
  employee_id INTEGER CONSTRAINT employees_pk PRIMARY KEY,
  manager_id INTEGER,
  first_name VARCHAR2(10) NOT NULL,
  last_name VARCHAR2(10) NOT NULL,
  title VARCHAR2(20),
```

```
  salary NUMBER(6, 0)
);
```

store_schema.sql 脚本向 employees 表填充以下行：

```
EMPLOYEE_ID MANAGER_ID FIRST_NAME LAST_NAME   TITLE             SALARY
----------- ---------- ---------- ----------- ------------- ----------
          1            James      Smith       CEO               800000
          2          1 Ron        Johnson     Sales Manager     600000
          3          2 Fred       Hobbs       Salesperson       150000
          4          2 Susan      Jones       Salesperson       500000
```

从上面可以看到，James Smith 没有上级管理者。这是因为他是此商店的 CEO。

salary_grades 表 salary_grades 表用来存储员工工资的不同等级，包括以下内容：

? 工资等级 ID

? 该级工资的最低工资

? 该级工资的最高工资

salary_grades 表是由 store_schema.sql 脚本使用下面的 CREATE TABLE 语句创建的：

```
CREATE TABLE salary_grades (
  salary_grade_id INTEGER CONSTRAINT salary_grade_pk PRIMARY KEY,
  low_salary NUMBER(6, 0),
  high_salary NUMBER(6, 0)
);
```

store_schema.sql 脚本向 salary_grades 表填充以下行：

```
SALARY_GRADE_ID LOW_SALARY HIGH_SALARY
--------------- ---------- -----------
              1          1      250000
              2     250001      500000
              3     500001      750000
              4     750001      999999
```

1.6 添加、修改和删除行

本节介绍如何使用 INSERT、UPDATE 和 DELETE SQL 语句对数据库表执行添加、修改以及删除行的操作。使用 COMMIT 语句可以永久性地保存对行所做的修改，使用 ROLLBACK 语句可以取消对行所做的修改。本节并不会详细介绍这些语句的具体用法，第 9 章将介绍更多有关这些 SQL 语句的知识。

1.6.1 向表中添加行

INSERT 语句用于向表中添加新行，在 INSERT 语句中可以指定以下信息：

? 向哪个表中插入行

? 为哪些列指定值

? 存储到这些列中的值

在插入行时，至少要指定主键和其他所有被定义为 NOT NULL 列的值。除此之外，所有

列的值都可以不指定。如果忽略它们的值，这些列的值都会被自动设置为空。

可以使用 SQL*Plus 的 DESCRIBE 命令来查看哪些列被定义为 NOT NULL，如下所示：

```
SQL> DESCRIBE customers
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 CUSTOMER_ID                               NOT NULL NUMBER(38)
 FIRST_NAME                                NOT NULL VARCHAR2(10)
 LAST_NAME                                 NOT NULL VARCHAR2(10)
 DOB                                                DATE
 PHONE                                              VARCHAR2(12)
```

可以看到，customer_id、first_name 和 last_name 列都是 NOT NULL，这意味着必须为这些列提供值。dob 和 phone 列则不需要提供值—— 如果不希望为这些列提供值，就可以将其忽略，这些列都会被默认设置为空。

继续运行下面这条 INSERT 语句，此语句向 customers 表中添加一行。注意 VALUES 列表中值的顺序必须与列的列表中指定的列的顺序一致。

```
SQL> INSERT INTO customers (
  2   customer_id, first_name, last_name, dob, phone
  3 )  VALUES (
  4   6, 'Fred', 'Brown', '01-JAN-1970', '800-555-1215'
  5 );

1 row created.
```

注意：

在每一行末尾按下 ENTER 键之后，SQL*Plus 会自动编上行号。

在上面这个例子中，执行 INSERT 语句之后，SQL*Plus 会作出响应，说明已经成功创建了一行。可以通过执行下面的 SELECT 语句来验证新添加了一行：

```
SELECT *
FROM customers;

CUSTOMER_ID  FIRST_NAME LAST_NAME  DOB         PHONE
----------- ---------- ---------- ----------- ------------
          1  John       Brown      01-JAN-65   800-555-1211
          2  Cynthia    Green      05-FEB-68   800-555-1212
          3  Steve      White      16-MAR-71   800-555-1213
          4  Gail       Black                  800-555-1214
          5  Doreen     Blue       20-MAY-70
          6  Fred       Brown      01-JAN-70   800-555-1215
```

注意新的一行已添加到表的末尾。

默认情况下，Oracle 数据库按照 DD-MON-YY 的格式来显示日期[1]，其中 DD 是日，MON 是月的前三个字母(大写)，YY 是年份的最后两位。数据库实际上为年份存储了 4 位，但是默

1 译者注：中文环境下的默认日期格式与此不同。

认情况下只显示最后两位。

当向 customers 表中添加一行时，customer_id 列的值必须是唯一的。如果新增加行的主键值在表中已经存在，Oracle 数据库会阻止这种添加行为。例如，下面的 INSERT 语句会产生错误，因为 customer_id 为 1 的行已经存在：

```
SQL> INSERT INTO customers (
  2   customer_id, first_name, last_name, dob, phone
  3  ) VALUES (
  4   1, 'Lisa', 'Jones', '02-JAN-1971', '800-555-1225'
  5  );

INSERT INTO customers (
*
ERROR at line 1:
ORA-00001: unique constraint (STORE.CUSTOMERS_PK) violated
```

注意，错误信息中显示了约束名(STORE.CUSTOMERS_PK)。这就是为什么应该总是命名主键约束的原因。否则，Oracle 数据库会为约束分配由系统生成的名称(例如 SYS_C0011277)，这种名称很难定位问题。

1.6.2 修改表中的现有行

可以使用 UPDATE 语句来修改表中现有的行。通常，在使用 UPDATE 语句时，需要指定以下内容：

? 包含要修改的行的表
? 指明要被修改的行的 WHERE 子句
? 列名的列表及其新值，这两部分内容使用 SET 子句来指定

使用同一条 UPDATE 语句可以修改一行或多行。如果指定了多行，就会对这些行进行相同的修改操作。下面这条 UPDATE 语句将 customers 表中 customer_id 列值为 2 的行的 last_name 列修改为 Orange：

```
UPDATE customers
SET last_name = 'Orange'
WHERE customer_id = 2;

1 row updated.
```

SQL*Plus 会确认有一行已被更新。

警告：
如果忘记在 UPDATE 语句中加上 WHERE 子句，那么所有的行都会被更新。

可以使用下面的查询语句显示被更新的行：

```
SELECT *
FROM customers
WHERE customer_id = 2;

CUSTOMER_ID FIRST_NAME LAST_NAME   DOB       PHONE
```

```
---------- ---------- ---------- --------- ------------
         2 Cynthia    Orange     05-FEB-68  800-555-1212
```

1.6.3 从表中删除行

DELETE 语句用于从表中删除行。一般情况下，使用 WHERE 子句来限制想要删除的行。如果不使用 WHERE 子句，就会删除表中所有的行。

下面这条 DELETE 语句从 customers 表中删除 customer_id 为 6 的顾客：

```
DELETE FROM customers
WHERE customer_id = 6;

1 row deleted.
```

要取消对数据库所做的修改，可以使用 ROLLBACK 命令：

```
ROLLBACK;

Rollback complete.
```

注意：
使用 COMMIT 可以永久性地保存对行所做的修改，具体操作参见第 9 章。

1.7 连接数据库和断开连接

当连接到数据库时，SQL*Plus 会维护一个数据库会话。当断开数据库连接时，该会话就结束了。可以通过输入 DISCONNECT 断开数据库连接并保持 SQL*Plus 继续运行：

```
DISCONNECT
```

在默认情况下，当断开数据库连接时，会自动执行 COMMIT 命令。
可以通过输入 CONNECT 命令重新连接某个数据库。要重新连接到 store 模式，可输入用户名 store 和密码 store_password：

```
CONNECT store/store_password
```

1.8 退出 SQL*Plus

可以使用 EXIT 命令退出 SQL*Plus。下面这个例子使用 EXIT 命令退出 SQL*Plus：

```
EXIT
```

默认情况下，当使用 EXIT 命令退出 SQL*Plus 时，会自动执行 COMMIT。如果 SQL*Plus 异常中止——例如，正在运行 SQL*Plus 的计算机崩溃——就自动执行 ROLLBACK。更多有关 COMMIT 和 ROLLBACK 的内容参见第 9 章。

1.9 Oracle PL/SQL 简介

PL/SQL 是 Oracle 的一种过程语言，可以用来添加一些基于 SQL 的编程结构。PL/SQL 主要用来在数据库中创建过程和函数，以实现业务逻辑。PL/SQL 中包含了一些标准的编程结构，例如：

? 变量声明
? 条件逻辑(if-then-else 等)
? 循环
? 过程和函数定义

下面这条 CREATE PROCEDURE 语句定义了一个名为 update_product_price()的过程。这个过程将指定产品的价格乘以提供的一个因子。如果指定的产品不存在，该过程就不执行任何操作；否则，就将该产品的价格更新为原来的价格与这个因子的乘积。

注意：

现在不要太过关注下面这段 PL/SQL 代码的细节，第 12 章将介绍 PL/SQL 的所有内容。现在只需对 PL/SQL 有基本的了解即可。

```
CREATE PROCEDURE update_product_price (
   p_product_id IN products.product_id%TYPE,
   p_factor     IN NUMBER
) AS
   v_product_count INTEGER;
BEGIN
   -- count the number of products with the
   -- supplied product_id (will be 1 if the product exists)
   SELECT COUNT(*)
   INTO v_product_count
   FROM products
   WHERE product_id = p_product_id;

   -- if the product exists (v_product_count = 1) then
   -- update that product's price
   IF v_product_count = 1 THEN
     UPDATE products
     SET price = price * p_factor
     WHERE product_id = p_product_id;
     COMMIT;
   END IF;
EXCEPTION
   WHEN OTHERS THEN
     ROLLBACK;
END update_product_price[2];
/
```

2 译者注：上述过程的写法不佳，实际工作中不应该先 SELECT 计数，而应直接用 UPDATE 更新，不符合条件的行自然不会更新。

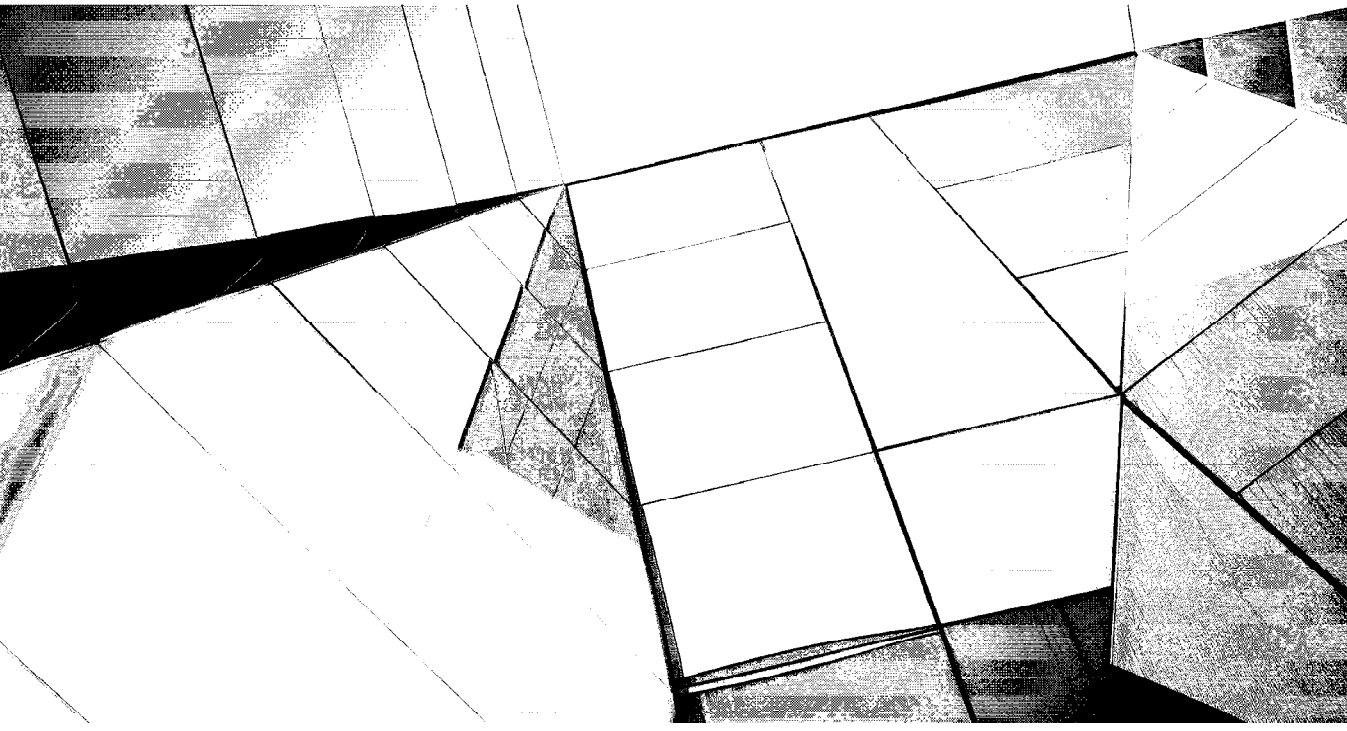
异常部分用来处理 PL/SQL 代码中产生的错误。在上面这个例子中，只要代码抛出任何异常，EXCEPTION 代码块就会执行 ROLLBACK。

1.10 小结

本章介绍了以下内容：

? 关系数据库是一组已经被组织为表结构的相关信息的集合。
? 结构化查询语言(SQL)是用来访问数据库的标准语言。
? 可以使用 SQL*Plus 来运行 SQL 语句和 SQL*Plus 命令。
? SQL Developer 是数据库开发的图形化工具。
? PL/SQL 是 Oracle 的一种过程语言，包含编程语句。

第 2 章将介绍更多有关从数据库表中检索信息的知识。



第 2 章

从数据库表中检索信息

本章内容包括：

? **使用** SELECT **语句检索信息**

? **使用算术表达式进行计算**

? **使用** WHERE **子句对行的检索进行限定**

? **对从表中检索到的行进行排序**

注意：

在继续之前，重新运行 store_schema.sql **来重建** store **表，以便你的查询与本章中显示的匹配。**

2.1 对单表执行 SELECT 语句

SELECT 语句用于从数据库表中检索信息。在 SELECT 语句最简单的形式中，只需指定要从中检索数据的表以及要查询的列名即可。SELECT 语句也被称为查询。

下面这个例子中的 SELECT 语句从 customers 表中检索 customer_id、first_name、last_name、dob 和 phone 列的内容：

```
SELECT customer_id, first_name, last_name, dob, phone
FROM customers;
```

在这条语句中，SELECT 关键字后面紧跟着要检索的列名；FROM 关键字后面是要检索列的表名。SQL 语句以分号(;)结束。

不需要确切地告诉 Oracle 数据库如何访问想要的信息；只需要指定所需的数据并让软件来检索这些数据即可。

在 SQL 语句末尾按 Enter 键之后，这条语句就会被执行，结果显示在屏幕上，输出内容如下：

```
CUSTOMER_ID  FIRST_NAME LAST_NAME   DOB        PHONE
-----------  ---------- ----------- ---------- ------------
          1  John       Brown       01-JAN-65  800-555-1211
          2  Cynthia    Green       05-FEB-68  800-555-1212
          3  Steve      White       16-MAR-71  800-555-1213
          4  Gail       Black                  800-555-1214
          5  Doreen     Blue        20-MAY-70
```

数据库返回的所有行称为结果集(result set)。请注意示例结果集中的下列内容：

- Oracle 数据库将列名全部转换成了大写的形式。
- 字符和日期列是左对齐的。
- 数字列则是右对齐的。
- 默认情况下，Oracle 数据库会以 DD-MON-YY 的形式显示日期，其中 DD 是日，MON 是月份的前三个字母(大写)，而 YY 是年份的最后两位。数据库实际上会为年份存储 4 位数字，但是默认情况下只会显示最后两位。

虽然列名和表名既可以使用小写字母来指定，也可以使用大写字母来指定，但最好是统一成一种风格。本书中的例子对于保留字使用大写，对于其他字符则全部使用小写。

2.2 选择一个表中的所有列

如果希望选择一个表中的所有列，可以用星号字符(*)取代列的列表。在下面这个查询中，SELECT 语句使用星号来表示要检索 customers 表中所有的列：

```
SELECT *
FROM customers;

CUSTOMER_ID FIRST_NAME LAST_NAME  DOB        PHONE
```

```
----------- ---------- ---------- --------- ------------
          1  John       Brown      01-JAN-65  800-555-1211
          2  Cynthia    Green      05-FEB-68  800-555-1212
          3  Steve      White      16-MAR-71  800-555-1213
          4  Gail       Black                 800-555-1214
          5  Doreen      Blue      20-MAY-70
```

可以看到，customers **表中所有的列都显示出来了。**

2.3　使用 WHERE 子句限定行

在查询中可以使用 WHERE **子句来限定想要检索的行。这种功能非常重要，因为** Oracle **可以在一个表中存储很多行，而用户可能只对其中很小的一个子集感兴趣。**WHERE **子句应放在** FROM **子句的后面：**

```
SELECT list of items
FROM list of tables
WHERE list of conditions;
```

在下面这个查询中，WHERE **子句用于限定从** customers **表中检索** customer_id **列值为** 2 **的行：**

```
SELECT *
FROM customers
WHERE customer_id = 2;

CUSTOMER_ID FIRST_NAME LAST_NAME   DOB        PHONE
----------- ---------- ----------  ---------  ------------
          2 Cynthia    Green       05-FEB-68  800-555-1212
```

2.4　行标识符

Oracle **数据库中的每一行都有唯一的行标识符，或称为** rowid。Oracle **数据库内部使用行标识符来存储行的物理位置。**rowid **是一个** 18 **位数字，采用** base-64 **编码。可以在查询的选择列表中指定** ROWID **列来查看表中各行的** rowid **值。**

例如，下面这个查询就检索 customers **表的** ROWID **和** customer_id **列，注意输出结果中** ROWID **的数字采用** base-64 **编码：**

```
SELECT ROWID, customer_id
FROM customers;

ROWID              CUSTOMER_ID
------------------ -----------
AAAF4yAABAAAHeKAAA           1
AAAF4yAABAAAHeKAAB           2
AAAF4yAABAAAHeKAAC           3
AAAF4yAABAAAHeKAAD           4
AAAF4yAABAAAHeKAAE           5
```

当使用 SQL*Plus 的 DESCRIBE 命令查看表的结构时，命令的输出结果中并没有 ROWID。这是因为该列只在数据库内部使用。ROWID 通常称为伪列(pseudo column)。下面的例子描述了 customers 表，注意 ROWID 并不出现在输出结果中：

```
DESCRIBE customers
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 CUSTOMER_ID                               NOT NULL NUMBER(38)
 FIRST_NAME                                NOT NULL VARCHAR2(10)
 LAST_NAME                                 NOT NULL VARCHAR2(10)
 DOB                                                DATE
 PHONE                                              VARCHAR2(12)
```

2.5 行号

另一个伪列是 ROWNUM，它返回每一行在结果集中的行号。查询返回的第一行的行号是 1，第二行的行号是 2，依此类推。

下列查询从 customers 表中检索行时包含 ROWNUM：

```
SELECT ROWNUM, customer_id, first_name, last_name
FROM customers;

    ROWNUM CUSTOMER_ID FIRST_NAME LAST_NAME
---------- ----------- ---------- ----------
         1           1  John       Brown
         2           2  Cynthia    Green
         3           3  Steve      White
         4           4  Gail       Black
         5           5  Doreen     Blue
```

下面是另一个例子：

```
SELECT ROWNUM, customer_id, first_name, last_name
FROM customers
WHERE customer_id = 3;

    ROWNUM CUSTOMER_ID FIRST_NAME LAST_NAME
---------- ----------- ---------- ----------
         1           3  Steve      White
```

2.6 执行算术运算

在 SQL 语句中可以使用算术表达式来进行算术运算，包括加减乘除四则运算。算术表达式由两个操作数(数字或日期)和一个操作符组成。四则运算操作符如表 2-1 所示。

表 2-1　算术操作符

操　作　符	说　　明
+	加法
-	减法
*	乘法
/	除法

下面这个查询显示了如何使用乘法操作符(*)来计算 2×6(2 和 6 是操作数)：

```
SELECT 2*6
FROM dual;

       2*6
----------
        12
```

可以看到，屏幕上显示的是正确结果 12。在这条 SQL 语句中，使用的 2*6 就是表达式。表达式中可以包含列、字面值和操作符的组合。

2.6.1　执行日期运算

可以对日期进行加法和减法运算，例如可以对日期加上数字(表示天数)。下面这个例子就对 2012 年 7 月 25 日加上 2 天，结果如下：

```
SELECT TO_DATE('25-JUL-2012') + 2
FROM dual;

TO_DATE
---------
27-JUL-12
```

注意：

TO_DATE()是函数，功能是将字符串转换为日期。第 5 章将会介绍有关 TO_DATE()的更多内容。

dual 表

dual 表经常与返回值的函数和表达式连用，而不需要在查询中引用基础表。例如，dual 表可用于返回算术表达式的查询，或用于调用诸如 TO_DATE()的函数的查询。当然，如果表达式或函数调用中引用了某个基础表中的列，就不能使用 dual 表了。

下面这个 DESCRIBE 命令的输出显示了 dual 表的结构，它包括一个 VARCHAR2 类型的列，该列名为 dummy：

```
DESCRIBE dual
 Name                                      Null?    Type
 ----------------------------------------- -------- ------------
 DUMMY                                              VARCHAR2(1)
```

下列查询检索 dual 表中唯一的行，它在 dummy 列中包含字符 X。

```
SELECT *
FROM dual;

D
-
X
```

下面这个例子从 2012 年 8 月 2 日减去 3 天：

```
SELECT TO_DATE('02-AUG-2012') - 3
FROM dual;

TO_DATE
---------
30-JUL-12
```

也可以从一个日期减去另一个日期，结果是这两个日期之间相差的天数。下面这个例子就是从 2012 年 8 月 2 日减去 2012 年 7 月 25 日：

```
SELECT TO_DATE('02-AUG-2012') - TO_DATE('25-JUL-2012')
FROM dual;

TO_DATE('02-AUG-2012')-TO_DATE('25-JUL-2012')
---------------------------------------------
                                            8
```

2.6.2 列运算

操作数不一定必须是字面数字或日期，也可以是表中的列。在下面这个查询中，name 和 price 列都是从 products 表中检索出来的；对 price 列的值加 2，组成的表达式为 price + 2：

```
SELECT name, price + 2
FROM products;

NAME                           PRICE+2
------------------------------ ----------
Modern Science                      21.95
Chemistry                             32
Supernova                           27.99
Tank War                            15.95
Z Files                             51.99
2412: The Return                    16.95
Space Force 9                       15.49
From Another Planet                 14.99
Classical Music                     12.99
Pop 3                               17.99
Creative Yell                       16.99
My Front Line                       15.49
```

在一个表达式中可以组合使用多个操作符。在下面这个查询中，price 列首先被乘以 3，然后再加上 1，结果如下：

```
SELECT name, price * 3 + 1
FROM products;

NAME                          PRICE*3+1
----------------------------- ----------
Modern Science                     60.85
Chemistry                             91
Supernova                          78.97
Tank War                           42.85
Z Files                           150.97
2412: The Return                   45.85
Space Force 9                      41.47
From Another Planet                39.97
Classical Music                    33.97
Pop 3                              48.97
Creative Yell                      45.97
My Front Line                      41.47
```

2.6.3　算术运算操作符的优先级

算术运算操作符的优先规则在 SQL **中也同样适用：乘法和除法优先，然后是加法和减法。如果操作符的优先级相同，那么运算的顺序为从左到右。**

例如，如果要使用表达式 10*12/3-1**，那么首先计算** 10 **乘以** 12**，结果是** 120**；然后将** 120 **除以** 3**，结果是** 40**；最后，从** 40 **中减去** 1**，结果是** 39**：**

```
SELECT 10 * 12 / 3 – 1
FROM dual;

  10*12/3-1
----------
        39
```

圆括号可以用来指定操作符的执行顺序。例如：

```
SELECT 10 * (12 / 3 – 1)
FROM dual;

10*(12/3-1)
-----------
         30
```

在这个例子中，使用圆括号来说明首先计算 12/3-1**，然后将所得的结果乘以** 10**—— 所以最终结果为** 30**。**

2.7　使用列别名

当从表中选择一列时，Oracle **在输出结果中使用该列列名的大写形式作为列的标题。例如，当选择** price **列时，输出结果中的标题是** PRICE**。在使用表达式时，**Oracle **会去掉表达式中的空格，并将其作为标题。**

也可以使用别名指定标题。在下面这个查询中，表达式 price*2 就用了别名 DOUBLE_PRICE：

```
SELECT price * 2 DOUBLE_PRICE
FROM products;

 DOUBLE_PRICE
 ------------
         39.9
           60
        51.98
         27.9
        99.98
         29.9
        26.98
        25.98
        21.98
        31.98
        29.98
        26.98
```

如果希望在别名中使用空格并保持别名文本的大小写形式，就必须使用双引号(" ")将别名文本引起来，例如：

```
SELECT price * 2 "Double Price"
FROM products;

Double  Price
------------
         39.9
...
```

在别名之前，也可以使用可选的关键字 AS，如下面这个查询所示：

```
SELECT 10 * (12 / 3 - 1) AS "Computation"
FROM dual;

Computation
-----------
         30
```

2.8 使用连接操作合并列的输出结果

可以使用连接操作来合并列的输出结果。这样可以创建界面更友好、更有意义的输出。例如，在 customers 表中，first_name 和 last_name 列合起来才是顾客的姓名。可以使用连接操作符(||)将这两列合并，如下面这个查询所示，从而实现这种功能。注意在输出中：first_name 和 last_name 之间用了一个空格字符来分隔。

```
SELECT first_name || ' ' || last_name AS "Customer Name"
FROM customers;
```

```
Customer Name
--------------------
John Brown
Cynthia Green
Steve White
Gail Black
Doreen Blue
```

在输出结果中，first_name 和 last_name 列的值合并起来作为 Customer Name 别名的值。

2.9　空值

数据库如何表示未知的值呢？答案是使用一种特殊的值，我们称之为空值(null value)。空值并不是空的字符串，而是特殊的值。空值就表示该列的值未知。

当检索一个包含空值的列时，看到的结果是该列中不包含任何内容。例如：

```
SELECT *
FROM customers;

CUSTOMER_ID FIRST_NAME LAST_NAME  DOB        PHONE
----------- ---------- ---------- ---------  ------------
          1 John       Brown      01-JAN-65  800-555-1211
          2 Cynthia    Green      05-FEB-68  800-555-1212
          3 Steve      White      16-MAR-71  800-555-1213
          4 Gail       Black                 800-555-1214
          5 Doreen     Blue       20-MAY-70
```

customer_id 为 4 的 dob 列是空值，customer_id 为 5 的 phone 列也是空值。

可以使用 IS NULL 子句来检查空值。在下面这个查询中，可以检索出 customer_id 为 4 的行，因为其 dob 列值为空：

```
SELECT customer_id, first_name, last_name, dob
FROM customers
WHERE dob IS NULL;

CUSTOMER_ID FIRST_NAME LAST_NAME DOB
----------- ---------- ---------- ---------
          4 Gail       Black
```

同样，在下面这个例子中，可以查询出 customer_id 为 5 的行，因为其 phone 列值为空：

```
SELECT customer_id, first_name, last_name, phone
FROM customers
WHERE phone IS NULL;

CUSTOMER_ID FIRST_NAME LAST_NAME PHONE
----------- ---------- ---------- ------------
          5 Doreen      Blue
```

既然空值不会显示任何内容，那么如何区分空值和空字符串呢？答案是使用 Oracle 内置的

函数 NVL()。NVL()函数可以将空值转换成另外一个值。NVL()函数需要接受两个参数：列(或者更确切地说，是可以返回值的任意表达式)和值；如果第一个参数是空值，就将其替换成第二个参数的值。在下面这个查询中，使用 NVL()函数将 phone 列中的空值转换为字符串 Unknown phone number：

```
SELECT customer_id, first_name, last_name,
  NVL(phone, 'Unknown phone number') AS PHONE_NUMBER
FROM customers;

CUSTOMER_ID FIRST_NAME LAST_NAME PHONE_NUMBER
----------- ---------- ---------- --------------------
          1 John       Brown      800-555-1211
          2 Cynthia    Green      800-555-1212
          3 Steve      White      800-555-1213
          4 Gail       Black      800-555-1214
          5 Doreen     Blue       Unknown phone number
```

NVL()函数除了用于转换包含空值的字符串列之外，还可以用于转换空的数字列和日期列。在下面这个查询中，NVL()函数用来将 dob 列中的空值转换成日期 01-JAN-2000：

```
SELECT customer_id, first_name, last_name,
 NVL(dob, '01-JAN-2000') AS DOB
FROM customers;

CUSTOMER_ID FIRST_NAME LAST_NAME  DOB
----------- ---------- ---------- ---------
          1 John       Brown      01-JAN-65
          2 Cynthia    Green      05-FEB-68
          3 Steve      White      16-MAR-71
          4 Gail       Black      01-JAN-00
          5 Doreen     Blue       20-MAY-70
```

注意 customer_id 为 4 的 dob 列显示成 01-JAN-00，该顾客的 dob 列是空值。

2.10 禁止显示重复行

假设想了解哪些顾客购买了产品，可以使用下面这个查询从 purchases 表中检索出 customer_id 列的内容：

```
SELECT customer_id
FROM purchases;

CUSTOMER_ID
-----------
          1
          2
          3
          4
          1
```

```
          2
          3
          4
          3
```

customer_id 列包含购买了产品的顾客 ID。从查询返回的输出结果中可以看到，有些顾客购买过不止一次，所以在结果中出现了多次。

可以使用 DISTINCT 关键字删除那些包含相同顾客 ID 的重复行。在下面这个查询中，DISTINCT 关键字用于禁止显示重复的行：

```
SELECT DISTINCT customer_id
FROM purchases;

CUSTOMER_ID
-----------
          1
          2
          4
          3
```

在这个列表中，更容易看出 customer_id 为 1、2、3 和 4 的顾客购买了产品。

2.11　比较值

表 2-2 列出了比较值时可以使用的操作符：

表 2-2　比较操作符

操　作　符	说　　明
=	等于
<> 或 !=	不等于 应当使用<>，因为它是美国国家标准组织(American National Standards Institute，ANSI)规定使用的
<	小于
>	大于
<=	小于或等于
>=	大于或等于
ANY	将一个值与一个列表中的任何值进行比较
SOME	等同于 ANY 操作符。应该使用 ANY 替代 SOME，因为 ANY 使用更为广泛，且从本人的观点来看，更易阅读
ALL	将一个值与一个列表中的所有值进行比较

2.11.1　使用不等于操作符

下面的查询在 WHERE 子句中使用不等于(<>)操作符从 customers 表中检索 customer_id 不等于 2 的行：

```
SELECT *
FROM customers
WHERE customer_id <> 2;

CUSTOMER_ID FIRST_NAME LAST_NAME  DOB        PHONE
----------- ---------- ---------- ---------  ------------
          1 John       Brown      01-JAN-65  800-555-1211
          3 Steve      White      16-MAR-71  800-555-1213
          4 Gail       Black                 800-555-1214
          5 Doreen     Blue       20-MAY-70
```

2.11.2 使用大于操作符

下面这个查询使用大于(>)操作符从 products 表中检索 product_id 和 name 列，要满足的条件是 product_id 列大于 8：

```
SELECT product_id, name
FROM products
WHERE product_id > 8;

PRODUCT_ID  NAME
----------- ---------------
          9 Classical Music
         10 Pop 3
         11 Creative Yell
         12 My Front Line
```

2.11.3 使用小于或等于操作符

下面这个查询使用 ROWNUM 伪列和小于等于(<=)操作符从 products 表中检索前 3 行[1]：

```
SELECT ROWNUM, product_id, name
FROM products
WHERE ROWNUM <= 3;

    ROWNUM PRODUCT_ID  NAME
---------- ---------- --------------
         1          1  Modern Science
         2          2  Chemistry
         3          3  Supernova
```

2.11.4 使用 ANY 操作符

可以使用 ANY 操作符将一个值与某个列表中的任何值进行比较。此时必须在 ANY 之前添加一个 =、<>、<、>、<= 或 >= 操作符。下面的查询使用 ANY 操作符从 customers 表中检索 customer_id 列大于 2、3 或 4 中任意值的行：

```
SELECT *
```

1 译者注：如果不用 order by 语句指定排序列，就不能保证检索出的结果是按某个顺序排列的“前”几行。

```
FROM customers
WHERE customer_id > ANY (2, 3, 4);

CUSTOMER_ID FIRST_NAME LAST_NAME  DOB        PHONE
----------- ---------- ---------- --------- ------------
          3 Steve      White      16-MAR-71  800-555-1213
          4 Gail       Black                 800-555-1214
          5 Doreen     Blue       20-MAY-70
```

2.11.5　使用 ALL 操作符

可以使用 ALL 操作符将一个值与某个列表中的所有值进行比较，此时必须在 ALL 之前放上一个 =、<>、<、>、<= 或 >= 操作符。下面的查询使用 ALL 操作符从 customers 表中检索 customer_id 列比 2、3 和 4 都大的行：

```
SELECT *
FROM customers
WHERE customer_id > ALL (2, 3, 4);

CUSTOMER_ID FIRST_NAME LAST_NAME  DOB        PHONE
----------- ---------- ---------- --------- ------------
          5 Doreen     Blue       20-MAY-70
```

返回结果中只有 customer_id 为 5 的顾客，因为只有 5 才比 2、3、4 都大。

2.12　使用 SQL 操作符

SQL 操作符可以通过对字符串或值列表、值范围以及空值进行模式匹配，来限定查询返回的行。SQL 操作符如表 2-3 所示。

表 2-3　SQL 操作符

操　作　符	说　　明
LIKE	匹配字符串中的模式
IN	匹配值列表
BETWEEN	匹配值范围
IS NULL	匹配空值
IS NAN	匹配 NAN 这个特殊值，意思是“非数字”
IS INFINITE	匹配 BINARY_FLOAT 和 BINARY_DOUBLE 中的“无穷”值

还可以使用 NOT 使操作符的含义相反，比如：

- NOT LIKE
- NOT IN
- NOT BETWEEN
- IS NOT NULL
- IS NOT NAN

? IS NOT INFINITE

下面将分别介绍 LIKE、IN 和 BETWEEN 操作符。

2.12.1 使用 LIKE 操作符

可以使用 LIKE 操作符来搜索匹配指定的模式的字符串。模式需要使用普通字符和以下两个通配符的组合指定：

? 下划线字符(_) 匹配指定位置的一个字符。

? 百分号字符(%) 匹配从指定位置开始的任意多个字符。

例如，考虑如下模式：

```
'_o%'
```

下划线匹配第一个字符位置处的任意一个字符，o 匹配第二个位置处的字符 o，百分号匹配 o 字符之后的任意多个字符。

下面的查询使用 LIKE 操作符，并指定对 customers 表的 first_name 列应用_o%模式：

```
SELECT *
FROM customers
WHERE first_name LIKE '_o%';

CUSTOMER_ID FIRST_NAME LAST_NAME   DOB        PHONE
----------- ---------- ---------- ---------- ------------
          1 John       Brown      01-JAN-65  800-555-1211
          5 Doreen     Blue       20-MAY-70
```

可以看到，结果返回两行，因为字符串 John 和 Doreen 的第二个字符都是 o。

下面这个查询使用 NOT LIKE 来检索与上一个查询相反的结果：

```
SELECT *
FROM customers
WHERE first_name NOT LIKE '_o%';

CUSTOMER_ID FIRST_NAME LAST_NAME  DOB        PHONE
----------- ---------- ---------- ---------  ------------
          2 Cynthia    Green      05-FEB-68  800-555-1212
          3 Steve      White      16-MAR-71  800-555-1213
          4 Gail       Black                 800-555-1214
```

如果需要对字符串中实际的下划线或百分号字符进行搜索，可以使用 ESCAPE 选项来标识这些字符。例如，考虑如下模式：

```
'%\%%' ESCAPE '\'
```

ESCAPE 后面的字符告诉数据库如何区分要搜索的字符与通配符，此例中使用反斜杠(\)。第一个%是通配符，匹配任意多个字符；第二个%是要搜索的实际字符；第三个%是通配符，匹配任意多个字符。

下面这个查询使用 promotions 表，该表包含商店打折产品的详细信息。此查询使用 LIKE 操作符来搜索 promotions 表的 name 列，采用的模式是'%\%%' ESCAPE '\'：

```
SELECT name
FROM promotions
WHERE name LIKE '%\%%' ESCAPE '\';

NAME
------------------------------
10% off Z Files
20% off Pop 3
30% off Modern Science
20% off Tank War
10% off Chemistry
20% off Creative Yell
15% off My Front Line
```

从这一结果中可以看到，查询返回其名称包含一个百分号字符的那些行。

2.12.2　使用 IN 操作符

可以使用 IN 操作符来检查一个值是否在值的列表中。下面的查询使用 IN 操作符，从 customers 表中检索 customer_id 列的值为 2、3 或 5 的行：

```
SELECT *
FROM customers
WHERE customer_id IN (2, 3, 5);

CUSTOMER_ID FIRST_NAME LAST_NAME  DOB        PHONE
----------- ---------- ---------- --------- ------------
          2 Cynthia    Green      05-FEB-68  800-555-1212
          3 Steve      White      16-MAR-71  800-555-1213
          5 Doreen     Blue       20-MAY-70
```

下面的查询用 NOT IN 检索那些未被 IN 检索出来的行：

```
SELECT *
FROM customers
WHERE customer_id NOT IN (2, 3, 5);

CUSTOMER_ID FIRST_NAME  LAST_NAME  DOB       PHONE
----------- ----------  ---------- --------- ------------
          1 John        Brown      01-JAN-65 800-555-1211
          4 Gail        Black                800-555-1214
```

需要注意的是，如果列表中包含空值，那么 NOT IN 返回 false。下面这个查询可以说明这一点，它不返回任何行，因为列表中包含空值：

```
SELECT *
FROM customers
WHERE customer_id NOT IN (2, 3, 5, NULL);

no rows selected
```

警告：

如果列表中有值为空值，那么 NOT IN **就会返回** false**。这一点很重要，因为既然可以在列表中使用任何表达式，而不只是字面值，那么要发现何时会产生空值就很困难。应当考虑对可能返回空值的表达式使用** NVL()**函数。**

2.12.3 使用 BETWEEN 操作符

可以使用 BETWEEN **操作符来检查一个值是否包含在指定的值区间内。区间是闭区间，这就意味着包含区间的两个端点。下面的查询使用** BETWEEN **操作符，从** customers **表中检索** customer_id **列的值在** 1 **和** 3 **之间的行：**

```
SELECT *
FROM customers
WHERE customer_id BETWEEN 1 AND 3;

CUSTOMER_ID FIRST_NAME LAST_NAME  DOB        PHONE
----------- ---------- ---------- --------- ------------
          1 John       Brown      01-JAN-65  800-555-1211
          2 Cynthia    Green      05-FEB-68  800-555-1212
          3 Steve      White      16-MAR-71  800-555-1213
```

NOT BETWEEN **检索那些未被** BETWEEN **检索出来的行：**

```
SELECT *
FROM customers
WHERE customer_id NOT BETWEEN 1 AND 3;

CUSTOMER_ID FIRST_NAME LAST_NAME  DOB       PHONE
----------- ---------- ---------- --------- ------------
          4 Gail       Black                800-555-1214
          5 Doreen     Blue       20-MAY-70
```

2.13 使用逻辑操作符

逻辑操作符根据逻辑条件来限定行。逻辑操作符如表 2-4 **所示：**

表 2-4 **逻辑操作符**

操 作 符	说 明
x AND *y*	**当** *x* **和** *y* **都为** true **时，才返回** true
x OR *y*	**当** *x* **和** *y* **中有一个为** true **时，就返回** true
NOT *x*	**如果** *x* **为** false**，就返回** true**；如果** *x* **为** true**，就返回** false

2.13.1 使用 AND 操作符

下面这个查询使用 AND **操作符从** customers **表中检索同时满足下列两个条件的行：**

? dob **列大于** 1970 **年** 1 **月** 1 **日**

? customer_id 列大于 3

```
SELECT *
FROM customers
WHERE dob > '01-JAN-1970'
AND customer_id > 3;

CUSTOMER_ID FIRST_NAME  LAST_NAME  DOB        PHONE
----------- ----------  ---------- --------- ------------
          5 Doreen      Blue       20-MAY-70
```

2.13.2 使用 OR 操作符

下面这个查询使用 OR 操作符从 customers 表中检索满足下列两个条件之一的行：

? dob 列大于 1970 年 1 月 1 日

? customer_id 列大于 3

```
SELECT *
FROM customers
WHERE dob > '01-JAN-1970'
OR customer_id > 3;

CUSTOMER_ID FIRST_NAME  LAST_NAME   DOB        PHONE
----------- ----------  ----------- --------- ------------
          3 Steve       White       16-MAR-71  800-555-1213
          4 Gail        Black                  800-555-1214
          5 Doreen      Blue        20-MAY-70
```

在 WHERE 子句中也可以使用逻辑操作符 AND 和 OR 将多个表达式组合在一起使用，你稍后将会看到这种用法。

2.14 逻辑操作符的优先级

如果在同一个表达式中同时使用 AND 和 OR 操作符，那么 AND 的优先级要高于 OR(这就意味着首先执行 AND 操作)；而比较操作符的优先级高于 AND。当然，可以使用圆括号来改变表达式的执行顺序。

下面这个例子从 customers 表中检索符合以下两个条件之一的行：

? dob 列的值大于 1970 年 1 月 1 日

? customer_id 列的值小于 2，而且电话号码以 1211 结尾

```
SELECT *
FROM customers
WHERE dob > '01-JAN-1970'
OR customer_id < 2
AND phone LIKE '%1211';

CUSTOMER_ID FIRST_NAME   LAST_NAME DOB        PHONE
----------- -----------  --------- --------- ------------
```

```
          1 John         Brown     01-JAN-65  800-555-1211
          3 Steve White            16-MAR-71  800-555-1213
          5 Doreen Blue            20-MAY-70
```

如前所述，AND 操作符的优先级高于 OR，因此可以认为上面这个查询的 WHERE 子句等价于下面的表达式：

```
dob > '01-JAN-1970' OR (customer_id < 2 AND phone LIKE '%1211')
```

因此，结果会返回 customer_id 为 1、3、5 的行。

2.15 使用 ORDER BY 子句对行进行排序

使用 ORDER BY 子句可以对查询检索出来的行进行排序。ORDER BY 子句可以指定一列或多列，查询结果会根据这些列对数据进行排序；ORDER BY 子句必须位于 FROM 或 WHERE 子句之后。

下面这个查询使用 ORDER BY 子句对从 customers 表中获得的数据根据 last_name 列的值进行排序：

```
SELECT *
FROM customers
ORDER BY last_name;

CUSTOMER_ID FIRST_NAME LAST_NAME  DOB        PHONE
----------- ---------- ---------- --------- ------------
          4 Gail       Black                800-555-1214
          5 Doreen     Blue       20-MAY-70
          1 John       Brown      01-JAN-65  800-555-1211
          2 Cynthia    Green      05-FEB-68  800-555-1212
          3 Steve      White      16-MAR-71  800-555-1213
```

默认情况下，ORDER BY 子句会按照升序对数据进行排序(较小的值先出现)。可以使用 DESC 关键字指定按照降序对数据进行排序(较大的值先出现)。也可以使用 ASC 关键字显式地说明采用升序进行排序，虽然升序是默认设置，但是为了更加明了，也可以指定这个选项。

下面这个查询使用 ORDER BY 子句，对从 customers 表中检索出的数据首先根据 first_name 列的值进行升序排序，然后再根据 last_name 列的值进行降序排序：

```
SELECT *
FROM customers
ORDER BY first_name ASC, last_name DESC;

CUSTOMER_ID FIRST_NAME LAST_NAME  DOB       PHONE
----------- ---------- ---------- --------- ------------
          2 Cynthia    Green      05-FEB-68 800-555-1212
          5 Doreen     Blue       20-MAY-70
          4 Gail       Black                800-555-1214
          1 John       Brown      01-JAN-65 800-555-1211
          3 Steve      White      16-MAR-71 800-555-1213
```

在 ORDER BY 子句中，也可以根据列的位置序号指定对哪一列进行排序：1 表示按第 1 列排序，2 表示按第 2 列排序，依此类推。在下面这个查询中，按列 1[2](customer_id 列)进行排序：

```
SELECT customer_id, first_name, last_name
FROM customers
ORDER BY 1;

CUSTOMER_ID FIRST_NAME LAST_NAME
----------- ---------- ----------
          1 John       Brown
          2 Cynthia    Green
          3 Steve      White
          4 Gail       Black
          5 Doreen     Blue
```

由于 customer_id 列位于 SELECT 关键字之后的第一个位置，因此在排序中就使用这一列。结果集显示按照 customer_id 排序后的行。

2.16　执行使用两个表的 SELECT 语句

数据库模式通常有多个表，这些表分别用于保存不同的数据。例如，store 模式就包含很多表，这些表分别用于存储顾客、产品和员工等信息。到现在为止，本书中所有的查询都只从一个表中检索行。在实际应用中，常常需要从多个表中检索信息——例如，检索产品名和产品的类型。本节将会介绍如何执行使用两个表的查询，稍后会介绍使用多于两个表的查询。

下面再次考虑这样一个例子，我们想要获得 product_id 为 3 的产品的名称及类型。产品的名称保存在 products 表的 name 列中，产品类型保存在 product_types 表的 name 列中。products 表和 product_types 表通过外键列 product_type_id 彼此关联在一起。products 表的 product_type_id 列(外键)指向 product_types 表的 product_type_id 列(主键)。

下面这个查询从 products 表中选择 product_id 为 3 的行的 name 和 product_type_id 列：

```
SELECT name, product_type_id
FROM products
WHERE product_id = 3;

NAME                           PRODUCT_TYPE_ID
------------------------------ ---------------
Supernova                                    2
```

接下来这个查询从 product_types 表中检索 product_type_id 为 2 的 name 列：

```
SELECT name
FROM product_types
WHERE product_type_id = 2;

NAME
```

2　译者注：这个序号是指在查询语句 SELECT 子句中列的顺序，不是表中列的顺序。

```
----------
Video
```

从上面这个结果中可以看出 product_id 为 3 的产品是视频录像。但是，为得到上述结果使用了两个查询。

使用表连接(table join)可以在查询中检索出产品名称和产品类型。要在查询中将两个表连接起来，就需要在查询的 FROM 子句中同时指定两个表，并且在 WHERE 子句中指明两个表中的相关列。

对于我们的示例查询，FROM 子句应该如下所示：

```
FROM products, product_types
```

WHERE 子句如下所示：

```
WHERE products.product_type_id = product_types.product_type_id
AND products.product_id = 3;
```

WHERE 子句中的第一个条件是连接(products.product_type_id = product_types.product_type_id)。一般情况下，连接中使用的列是其中一个表的主键和另一个表的外键。WHERE 子句中的第二个条件(products.product_id = 3)检索 product_id 为 3 的产品。

注意在 WHERE 子句中同时包含了表名和它们的列名。由于在 products 和 product_types 表中都包含名为 product_type_id 的列，因此就需要告诉数据库到底使用哪个表中的列(如果这些列有不同的名称，可以省略表名，但为了清楚地表明列来自哪里，始终应该包含表名)。

此查询的 SELECT 子句如下所示：

```
SELECT products.name, product_types.name
```

注意同样要指定表名和它们的列名。

现在把所有子句都放到一起，完整的 SELECT 语句如下所示：

```
SELECT products.name, product_types.name
FROM products, product_types
WHERE products.product_type_id = product_types.product_type_id
AND products.product_id = 3;

NAME                           NAME
------------------------------ ----------
Supernova                      Video
```

非常完美！单个查询返回了产品名称和产品类型。

下面这个查询得到所有产品，并按照 products.name 列对它们进行排序：

```
SELECT products.name, product_types.name
FROM products, product_types
WHERE products.product_type_id = product_types.product_type_id
ORDER BY products.name;

NAME                           NAME
------------------------------ ----------
2412: The Return               Video
```

```
Chemistry                 Book
Classical Music           CD
Creative Yell             CD
From Another Planet       DVD
Modern Science            Book
Pop 3                     CD
Space Force 9             DVD
Supernova                 Video
Tank War                  Video
Z Files                   Video
```

然而还要注意，结果集中并没有名为 My Front Line 的那个产品。这个产品的 product_type_id 是 null，而连接条件不返回这一行。在 2.20.2 节中将介绍如何得到这一行。

本节介绍的连接语法使用的是 Oracle 的连接语法，其基础是 ANSI 的 SQL/86 标准。在 Oracle Database 9*i* 及更高的版本中，数据库还实现了 ANSI SQL/92 标准的连接语法，在 2.21 “使用 SQL/92 语法执行连接”中将会介绍这种新语法。当使用 Oracle Database 9*i* 及更高版本时，查询中应该使用 SQL/92 标准的语法；只有当使用 Oracle Database 8*i* 及更低版本时，查询中才应该使用 SQL/86 标准的语法。

2.17　使用表别名

你之前看到过如下查询：

```
SELECT products.name, product_types.name
FROM products, product_types
WHERE products.product_type_id = product_types.product_type_id
ORDER BY products.name;
```

注意，在 SELECT 和 WHERE 子句中都使用了 products 和 product_types 表名。可以在 FROM 子句中定义表的别名，这样当在查询中的其他地方要引用该表时，就可以使用表的别名。

例如，下面的查询使用 p 作为 products 表的别名，使用 pt 作为 product_types 表的别名。注意，表的别名要在 FROM 子句中指定，且别名位于查询中其余列之前：

```
SELECT p.name, pt.name
FROM products p, product_types pt
WHERE p.product_type_id = pt.product_type_id
ORDER BY p.name;
```

使用表的别名能减少查询中输入的文本数量，并且可能还会减少在输入过程中的错误。

2.18　笛卡尔积

如果在多表查询中不指定连接条件，就会导致将一个表中的所有行都连接到另一个表中的所有行，这个结果集就称为笛卡尔积(cartesian product)。

例如，假设第一个表包含 50 行，第二个表包含 100 行。如果不使用连接条件从这两个表中检索行，就会返回 5000 行。这是因为第一个表中的每一行都被连接到第二个表中的每一行，

这样得到的总行数就是 50 乘以 100，也就是 5000 行。

下面这个例子显示了由 product_types 和 products 表得到的笛卡尔积的部分结果：

```
SELECT pt.product_type_id, p.product_id
FROM product_types pt, products p;

PRODUCT_TYPE_ID PRODUCT_ID
--------------- ----------
              1          1
              1          2
              1          3
              1          4
              1          5
...
              5          8
              5          9
              5         10
              5         11
              5         12

60 rows selected.
```

一共选择了 60 行，因为 product_types 表和 products 表分别包含 5 行和 12 行，因此结果就是 5×12 = 60 行。

在某些情况下，可能需要用笛卡尔积来完成工作。大多数时候不需要这么做，因此应根据需要包含连接条件。

2.19 执行使用多于两个表的 SELECT 语句

可以在查询中连接多个表。下面这个公式可以计算出在 WHERE 子句中需要的连接个数：

连接数=查询中使用的表的总数－1

例如，下面这个查询使用了两个表，因此用了一个连接：

```
SELECT p.name, pt.name
FROM products p, product_types pt
WHERE p.product_type_id = pt.product_type_id
ORDER BY p.name;
```

现在考虑一个从 4 个表中检索顾客购买信息的例子。假设希望查看以下信息：

? 已经购买过产品的顾客(来自 purchases 表)

? 顾客的姓名(来自 customers 表)

? 顾客购买的产品名称(来自 products 表)

? 产品类型(来自 product_types 表)

一共用到了 4 个表，因此需要 3 个连接。所需的连接如下所示：

1) 要获得曾经购买过产品的顾客，需要使用 customers 和 purchases 表中的 customers_id

列将这两个表连接起来(连接是 customers.customer_id = purchases.customer_id)。

2) 要获得顾客购买的产品，需要使用 products 和 purchases 表中的 product_id 列将这两个表连接起来(连接是 products.product_id = purchases.product_id)。

3) 要获得产品类型，需要使用 products 和 product_types 表中的 product_type_id 列将这两个表连接起来(连接是 products.product_type_id = product_types.product_type_id)。

下面的查询使用这些连接。注意使用了表的别名，并将产品的标题重命名为 PRODUCT，将产品类型重命名为 TYPE：

```
SELECT c.first_name, c.last_name, p.name AS PRODUCT, pt.name AS TYPE
FROM customers c, purchases pr, products p, product_types pt
WHERE c.customer_id = pr.customer_id
AND p.product_id = pr.product_id
AND p.product_type_id = pt.product_type_id
ORDER BY p.name;

FIRST_NAME LAST_NAME   PRODUCT                        TYPE
---------- ---------- ------------------------------ ----------
John       Brown      Chemistry                      Book
Cynthia    Green      Chemistry                      Book
Steve      White      Chemistry                      Book
Gail       Black      Chemistry                      Book
John       Brown      Modern Science                 Book
Cynthia    Green      Modern Science                 Book
Steve      White      Modern Science                 Book
Gail       Black      Modern Science                 Book
Steve      White      Supernova                      Video
```

到现在为止，介绍的多表查询在连接条件中都使用了等于操作符(=)，正是由于这个原因，这些连接被称为等值连接(equijoin)。在下一节中你将会看到，还可以使用其他类型的连接。

2.20　连接条件和连接类型

本节将会介绍有关连接条件和连接类型的知识，这些知识可以用来构建更高级的查询。

根据连接中使用的操作符的不同，连接条件(join condition)可以分为两类：

? 等值连接(equijoin) 在连接中使用等于操作符(=)。

? 不等连接(non-equijoin) 在连接中使用除等号之外的操作符，例如<、>、BETWEEN 等。

连接有 3 种不同的类型：

? 内连接(inner join) 只有当连接中的列包含满足连接条件的值时才会返回一行。这就是说，如果某一行的连接条件中的一列是空值，那么这行就不会返回。到现在为止，你看到的例子都是内连接。

? 外连接(outer join) 即使连接条件中的一列包含空值也会返回一行。

? 自连接(self join) 返回连接到同一表中的行。

接下来将会介绍有关不等连接、外连接和自连接的知识。

2.20.1 不等连接

不等连接在连接中使用除等于操作符之外的操作符，包括不等于(<>)、小于(<)、大于(>)、小于等于(<=)、大于等于(>=)、LIKE、IN 和 BETWEEN。

下面这个例子想要查询员工的工资等级。首先，下面这个查询从 salary_grades 表中检索工资等级：

```
SELECT *
FROM salary_grades;

SALARY_GRADE_ID LOW_SALARY HIGH_SALARY
--------------- ---------- -----------
              1          1      250000
              2     250001      500000
              3     500001      750000
              4     750001      999999
```

下一个查询使用不等连接来检索员工的工资和工资等级，工资等级用 BETWEEN 操作符来确定：

```
SELECT e.first_name, e.last_name, e.title, e.salary, sg.salary_grade_id
FROM employees e, salary_grades sg
WHERE e.salary BETWEEN sg.low_salary AND sg.high_salary
ORDER BY salary_grade_id;

FIRST_NAME LAST_NAME  TITLE                SALARY SALARY_GRADE_ID
---------- ---------- ------------------ ---------- ---------------
Fred       Hobbs      Salesperson            150000               1
Susan      Jones      Salesperson            500000               2
Ron        Johnson    Sales Manager          600000               3
James      Smith      CEO                    800000               4
```

在此查询中，如果员工的工资在工资等级的范围内，那么 BETWEEN 操作符返回 true。当 BETWEEN 操作符返回 true 时，就可以从 salary_grades 表找到此员工的工资等级 ID。

例如，Fred Hobbs 的工资是$150 000，位于$1 和$250 000 之间，在 salary_grades 表中其 salary_grade_id 是 1，因此 Fred Hobbs 的工资等级是 1。

同样，Susan Jones 的工资是$500 000，位于$250 001 和$500 000 之间，其 salary_grade_id 是 2，因此 Susan Jones 的工资等级是 2。

最后，Ron Johnson 和 James Smith 的工资等级分别是 3 和 4。

2.20.2 外连接

即使连接中的列包含空值，外连接也会返回一行。可以在连接条件中使用外连接操作符来执行外连接；Oracle 特有的外连接操作符是使用圆括号括起来的加号：(+)。

还记得前面的查询并没有显示 My Front Line 这个产品吗？因为它的 product_type_id 是 null。可以使用外连接得到那个产品：

```
SELECT p.name, pt.name
FROM products p, product_types pt
```

```
WHERE p.product_type_id = pt.product_type_id (+)
ORDER BY p.name;

NAME                          NAME
----------------------------- ----------
2412: The Return              Video
Chemistry                     Book
Classical Music               CD
Creative Yell                 CD
From Another Planet           DVD
Modern Science                Book
My Front Line
Pop 3                         CD
Space Force 9                 DVD
Supernova                     Video
Tank War                      Video
Z Files                       Video
```

注意，现在检索到了 My Front Line**，也就是** product_type_id **为** null **的产品。**
上一个查询的 WHERE **子句是：**

```
WHERE p.product_type_id = pt.product_type_id (+);
```

外连接操作符(+)**在相等操作符的右边，而** product **表的** p.product_type_id **列在相等操作符的左边。**p.product_type_id **列包含空值。**

注意：
应该将外连接操作符(+)**和包含空值的列分别放在等于操作符的两边。**

下面这个查询会返回与上一个查询完全相同的结果，但是要注意现在 Oracle **外连接操作符位于连接条件中等于操作符的左边，而含有空值的列在等于操作符的右边：**

```
SELECT p.name, pt.name
FROM products p, product_types pt
WHERE pt.product_type_id (+) = p.product_type_id
ORDER BY p.name;
```

1. **左外连接和右外连接**

外连接可以分为两类：
? **左外连接**
? **右外连接**
要理解左外连接和右外连接之间的区别，考虑下面的语法：

```
SELECT ...
FROM table1, table2
...
```

假设这两个表对 table1.column1 **和** table2.column2 **进行连接；再假设** table1 **中包含** column1 **为空值的一行。要执行左外连接，**WHERE **子句如下：**

```
WHERE table1.column1 = table2.column2 (+);
```

注意：

在左外连接中，外连接操作符(+)实际上是在等于操作符的右边。

接下来，假设 table2 **中包含** column2 **为空值的一行。要执行右外连接，需要将** Oracle **外连接操作符放到等于操作符的左边，**WHERE **子句就变成：**

```
WHERE table1.column1 (+) = table2.column2;
```

注意：

可想而知，如果 table1 **和** table2 **都包含其中有列为空值的行，那么根据使用的是左外连接还是右外连接，会得到不同的结果。**

现在来看几个具体的例子，以便更清晰地理解左外连接和右外连接。

左外连接的例子 下面这个查询给出了左外连接的用法；注意 Oracle **外连接操作符在等于操作符的右边：**

```
SELECT p.name, pt.name
FROM products p, product_types pt
WHERE p.product_type_id = pt.product_type_id (+);
ORDER BY p.name;

NAME                           NAME
------------------------------ ----------
2412: The Return               Video
Chemistry                      Book
Classical Music                CD
Creative Yell                  CD
From Another Planet            DVD
Modern Science                 Book
My Front Line
Pop 3                          CD
Space Force 9                  DVD
Supernova                      Video
Tank War                       Video
Z Files                        Video
```

注意，结果显示了 products **表中的所有行，包括** My Front Line **这一行，也就是** product_type_id **列为空值的那一行。**

右外连接的例子 product_types **表包含** products **表没有引用的产品类型** magazine(products **表中没有杂志)；注意，**magazine **产品类型在结果列表的最后：**

```
SELECT *
FROM product_types;

PRODUCT_TYPE_ID NAME
--------------- ----------
              1 Book
```

```
2 Video
3 DVD
4 CD
5 Magazine
```

使用右外连接来连接 products **和** product_types **表可以检索到** magazine **产品类型，如下面这个查询所示；注意，**Oracle **外连接操作符位于等于操作符的左边：**

```
SELECT p.name, pt.name
FROM products p, product_types pt
WHERE p.product_type_id (+) = pt.product_type_id;
ORDER BY p.name;

NAME                           NAME
------------------------------ ----------
2412: The Return               Video
Chemistry                      Book
Classical Music                CD
Creative Yell                  CD
From Another Planet            DVD
Modern Science                 Book
Pop 3                          CD
Space Force 9                  DVD
Supernova                      Video
Tank War                       Video
Z Files                        Video
                               Magazine
```

2. **外连接的限制**

外连接的使用有一些限制，只能在连接的一端使用外连接操作符，而不能在两端同时使用外连接操作符。如果试图在连接的两端同时使用 Oracle **外连接操作符，就会得到错误，如下面这个例子所示：**

```
SQL> SELECT p.name, pt.name
  2 FROM products p, product_types pt
  3 WHERE p.product_type_id (+) = pt.product_type_id (+);
WHERE p.product_type_id (+) = pt.product_type_id (+)
                            *
ERROR at line 3:
ORA-01468: a predicate may reference only one outer-joined table
```

不能同时使用外连接条件和另一个使用 OR **操作符的连接条件：**

```
SQL> SELECT p.name, pt.name
  2 FROM products p, product_types pt
  3 WHERE p.product_type_id (+) = pt.product_type_id
  4 OR p.product_type_id = 1;
WHERE p.product_type_id (+) = pt.product_type_id
                            *
ERROR at line 3:
ORA-01719: outer join operator (+) not allowed in operand of OR or IN
```

注意：

这里只提供了使用外连接操作符的常见限制。要了解所有限制，请参考 Oracle 公司的 *Oracle Database SQL Reference* 手册。

2.20.3 自连接

自连接是对同一个表进行的连接。要执行自连接，必须使用不同的表别名来标识在查询中每次对表的引用。现在考虑一个例子：employees 表中有一列 manager_id，它包含每个员工管理者的 employee_id，如果员工没有管理者，那么 manager_id 是空值。

employees 表包含的行如下：

```
EMPLOYEE_ID MANAGER_ID FIRST_NAME LAST_NAME  TITLE           SALARY
----------- ---------- ---------- ---------- -----------   --------
          1            James      Smith      CEO             800000
          2          1 Ron        Johnson    Sales Manager   600000
          3          2 Fred       Hobbs      Salesperson     150000
          4          2 Susan      Jones      Salesperson     500000
```

可以看到，CEO James Smith 的 manager_id 为空，这就是说他没有管理者。Susan Jones 和 Fred Hobbs 受 Ron Johnson 的管理，而 Ron Johnson 则受 James Smith 的管理。

可以使用自连接来显示每个雇员及其管理者的名字。在下面这个查询中，employees 表被引用了两次，分别使用了两个别名：w 和 m。别名 w 用于获得员工的姓名，而别名 m 则用于获得管理者的姓名。自连接是对 w.manager_id 和 m.employee_id 进行的：

```
SELECT w.first_name || ' ' || w.last_name || ' works for '||
  m.first_name || ' ' || m.last_name
FROM employees w, employees m
WHERE w.manager_id = m.employee_id
ORDER BY w.first_name;

W.FIRST_NAME||''||W.LAST_NAME||'WORKSFOR'||M.FIRST_NA
----------------------------------------------------
Fred Hobbs works for Ron Johnson
Ron Johnson works for James Smith
Susan Jones works for Ron Johnson
```

由于 James Smith 的 manager_id 为空，因此不会为他显示任何行。

可以同时使用外连接和自连接。下面的查询在前面那个例子的自连接基础上又使用了外连接，这样就可以看到 James Smith 这一行了。注意，此处使用 NVL()函数来说明 James Smith 为股东工作(他是 CEO，因此他向商店的股东汇报)：

```
SELECT w.last_name || ' works for ' ||
 NVL(m.last_name, 'the shareholders')
FROM employees w, employees m
WHERE w.manager_id = m.employee_id (+)
ORDER BY w.last_name;

W.LAST_NAME||'WORKSFOR'||NVL(M.LAST_N
```

```
-------------------------------------
Hobbs works for Johnson
Johnson works for Smith
Jones works for Johnson
Smith works for the shareholders
```

2.21　使用 SQL/92 语法执行连接

到现在为止我们看到的连接都是使用 Oracle 的连接语法。Oracle 语法的基础是 ANSI SQL/86 标准。在 Oracle Database 9*i* 和更高的版本中，数据库还实现了 ANSI SQL/92 标准的连接语法，应该在查询中使用 SQL/92 标准的语法。在本节中，读者将看到如何使用 SQL/92，包括如何用它来避免产生不必要的笛卡尔积。

2.21.1　使用 SQL/92 标准语法执行两个表的内连接

在前面已经介绍过下面这个查询，它使用 SQL/86 标准的语法来执行内连接：

```
SELECT p.name, pt.name
FROM products p, product_types pt
WHERE p.product_type_id = pt.product_type_id
ORDER BY p.name;
```

SQL/92 引入了 INNER JOIN 和 ON 子句来执行内连接。下面这个例子使用 INNER JOIN 和 ON 子句重写前面的查询：

```
SELECT p.name, pt.name
FROM products p INNER JOIN product_types pt
ON p.product_type_id = pt.product_type_id
ORDER BY p.name;
```

不等连接操作符和 ON 子句可以同时使用。前面已经介绍过下面这个查询使用 SQL/86 标准执行不等连接：

```
SELECT e.first_name, e.last_name, e.title, e.salary, sg.salary_grade_id
FROM employees e, salary_grades sg
WHERE e.salary BETWEEN sg.low_salary AND sg.high_salary
ORDER BY salary_grade_id;
```

下面这个例子使用 SQL/92 标准重写了这个查询：

```
SELECT e.first_name, e.last_name, e.title, e.salary, sg.salary_grade_id
FROM employees e INNER JOIN salary_grades sg
ON e.salary BETWEEN sg.low_salary AND sg.high_salary
ORDER BY salary_grade_id;
```

2.21.2　使用 USING 关键字简化连接

SQL/92 标准可以使用 USING 子句对连接条件进一步进行简化，但是只有在查询满足以下限制时才能进行简化：

? **查询必须是等值连接的**

? **等值连接中的列必须同名**

我们将要执行的大部分连接都是等值连接，如果总是对外键使用和主键相同的名字，就可以满足上面这些限制。

下面这个查询使用 USING **子句代替** ON **子句：**

```
SELECT p.name, pt.name
FROM products p INNER JOIN product_types pt
USING (product_type_id);
```

此时如果希望查看 product_type_id **的值，那么在** SELECT **子句中只能指定该列名，不能使用表名或别名。例如：**

```
SELECT p.name, pt.name, product_type_id
FROM products p INNER JOIN product_types pt
USING (product_type_id);
```

如果试图在列的前面使用表的别名，例如 p.product_type_id**，就会看到错误。例如：**

```
SQL> SELECT p.name, pt.name, p.product_type_id
  2 FROM products p INNER JOIN product_types pt
  3 USING (product_type_id);
SELECT p.name, pt.name, p.product_type_id
                        *
ERROR at line 1:
ORA-25154: column part of USING clause cannot have qualifier
```

另外，在 USING **子句中也只能单独使用列名。例如，如果在上面这个查询中试图使用** USING(p.product_type_id)**代替** USING(product_type_id)**，就会得到如下错误：**

```
SQL> SELECT p.name, pt.name, p.product_type_id
 2   FROM products p INNER JOIN product_types pt
 3   USING (p.product_type_id);
USING (p.product_type_id)
        *
ERROR at line 3:
ORA-01748: only simple column names allowed here
```

警告：

在 USING **子句中引用列时不要使用表名或别名，否则就会出现错误。**

2.21.3 使用 SQL/92 执行多于两个表的内连接

你在前面已经见到过下面这个使用 SQL/86 **的查询，它对** customers、purchases、products **和** product_types **表进行检索：**

```
SELECT c.first_name, c.last_name, p.name AS PRODUCT, pt.name AS TYPE
FROM customers c, purchases pr, products p, product_types pt
WHERE c.customer_id = pr.customer_id
AND p.product_id = pr.product_id
```

```
AND p.product_type_id = pt.product_type_id
ORDER BY p.name;
```

下面这个例子使用 SQL/92 对这个查询进行了重写；这里的外键关系是通过多个 INNER JOIN 和 USING 子句实现的：

```
SELECT c.first_name, c.last_name, p.name AS PRODUCT, pt.name AS TYPE
FROM customers c INNER JOIN purchases pr
USING (customer_id)
INNER JOIN products p
USING (product_id)
INNER JOIN product_types pt
USING (product_type_id)
ORDER BY p.name;
```

2.21.4　使用 SQL/92 执行多列的内连接

如果连接使用了两个表中的多个列，那么可以在 ON 子句中使用 AND 操作符并逐一列出这些列。举例来说，假设有两个表，表名分别为 table1 和 table2，您希望使用这两个表中的 column1 和 column2 列进行连接。查询语句应该如下所示：

```
SELECT ...
FROM table1 INNER JOIN table2
ON table1.column1 = table2.column1
AND table1.column2 = table2.column2;
```

可以使用 USING 子句进一步简化，条件是执行等值连接，而且列名相同。例如，下面这个查询使用 USING 子句对前面的查询进行了简化：

```
SELECT ...
FROM table1 INNER JOIN table2
USING (column1, column2);
```

2.21.5　使用 SQL/92 执行外连接

你在前面已经看到过如何使用外连接操作符(+)执行外连接，外连接操作符(+)是 Oracle 特有的语法。SQL/92 使用一种不同的语法来执行外连接。SQL/92 不使用(+)操作符，而是在 SELECT 语句的 FROM 子句中指定连接类型，语法如下所示：

```
FROM table1 { LEFT | RIGHT | FULL } OUTER JOIN table2
```

其中：

- table1 和 table2 指定希望连接的表
- LEFT 说明希望执行左外连接
- RIGHT 说明希望执行右外连接
- FULL 说明希望执行全外连接；全外连接使用 table1 和 table2 中所有的行，包括连接列为空值的行。不能使用 Oracle 连接操作符(+)执行全外连接

接下来将会介绍如何使用 SQL/92 的语法来执行左外连接、右外连接以及全外连接。

1. 使用 SQL/92 执行左外连接

你在前面已经看到过下面这个查询使用 Oracle 的连接操作符(+)执行左外连接：

```
SELECT p.name, pt.name
FROM products p, product_types pt
WHERE p.product_type_id = pt.product_type_id (+)
ORDER BY p.name;
```

下面这个例子使用 SQL/92 的 LEFT OUTER JOIN 关键字对这个查询进行了重写：

```
SELECT p.name, pt.name
FROM products p LEFT OUTER JOIN product_types pt
USING (product_type_id)
ORDER BY p.name;
```

2. 使用 SQL/92 执行右外连接

你在前面已经看到过下面这个查询使用 Oracle 的连接操作符(+)执行右外连接：

```
SELECT p.name, pt.name
FROM products p, product_types pt
WHERE p.product_type_id (+) = pt.product_type_id
ORDER BY p.name;
```

下面这个例子使用 SQL/92 的 RIGHT OUTER JOIN 关键字对这个查询进行了重写：

```
SELECT p.name, pt.name
FROM products p RIGHT OUTER JOIN product_types pt
USING (product_type_id)
ORDER BY p.name;
```

3. 使用 SQL/92 执行全外连接

全外连接使用连接表中所有的行，包括连接中使用的列为空值的那些行。下面这个例子给出了一个使用 SQL/92 的 FULL OUTER JOIN 关键字的查询：

```
SELECT p.name, pt.name
FROM products p FULL OUTER JOIN product_types pt
USING (product_type_id)
ORDER BY p.name;

NAME                           NAME
------------------------------ ----------
2412: The Return               Video
Chemistry                      Book
Classical Music                CD
Creative Yell                  CD
From Another Planet            DVD
Modern Science                 Book
My Front Line
Pop 3                          CD
```

```
Space Force 9                  DVD
Supernova                      Video
Tank War                       Video
Z Files                        Video
                               Magazine
```

注意，products **表中的** My Front Line **和** product_types **表中的** Magazine **现在都显示出来了。这些行都有空值。**

2.21.6　使用 SQL/92 执行自连接

下面这个例子使用 SQL/86 **对** employees **表执行自连接：**

```
SELECT w.last_name || ' works for ' || m.last_name
FROM employees w, employees m
WHERE w.manager_id = m.employee_id;
```

下面这个例子使用 SQL/92 **的** INNER JOIN **和** ON **关键字对这个查询进行了重写：**

```
SELECT w.last_name || ' works for ' || m.last_name
FROM employees w INNER JOIN employees m
ON w.manager_id = m.employee_id;
```

2.21.7　使用 SQL/92 执行交叉连接

你在前面已经看到如果省略两个表之间的连接条件，就会导致笛卡尔积。通过使用 SQL/92 **的连接语法，就可以避免产生笛卡尔积，因为在对表进行连接时，通常都必须提供** ON **或** USING **子句。**

如果的确想使用笛卡尔积， SQL/92 **标准要求必须在查询中使用** CROSS JOIN **关键字显式地进行声明。在下面这个查询中，就使用** CROSS JOIN **关键字在** product_types **和** products **表之间生成笛卡尔积：**

```
SELECT *
FROM product_types CROSS JOIN products;
```

2.22　小结

本章介绍了以下内容：

- **如何执行单表查询和多表查询。**
- **如何在查询中使用星号(*)从表中选择所有的列。**
- **在** Oracle **数据库内部如何使用行标识符(rowid)来存储行的位置。**
- **如何在** SQL **语句中执行算术运算。**
- **如何对日期进行加法和减法操作。**
- **如何使用别名来引用表和列。**
- **如何使用连接操作符(||)合并列输出。**
- **如何使用空值来表示未知值。**
- **如何使用** DISTINCT **操作符禁止显示重复行。**

- 如何使用 WHERE 子句限制返回结果。
- 如何使用 ORDER BY 子句对结果进行排序。
- 如何使用 SQL/86 和 SQL/92 的语法执行内连接、外连接和自连接。

第 3 章将会介绍 SQL*Plus。