

Oracle

从入门到精通



DVD-ROM

(11小时多媒体教学视频)

秦 靖 刘存勇 等编著

本书特色

- ※ 基本概念→语法讲解→示例讲解→实践练习→项目实战
- ※ 322个实例、4个项目案例、71个技巧、116个练习题

超值DVD-ROM

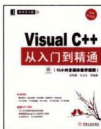
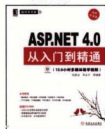
- ※ 11小时多媒体语音教学视频
- ※ 超值赠送SQL Server学习视频
- ※ 本书习题答案、本书教学PPT



机械工业出版社
China Machine Press

本书涵盖内容

Oracle 11g数据库简介
Oracle 11g的安装与测试
熟悉数据库
SQL基础
利用SELECT检索数据
Oracle内置函数
PL/SQL基础
游标——数据的缓存区
视图——数据库中虚拟的表
存储过程——提高程序执行的效率
触发器——保证数据的正确性
事务和锁——确保数据安全
使用Oracle 11g Enterprise Manager
常用工具介绍
控制文件和日志文件
表空间的管理
与数据库安全性有关的对象
备份与恢复
使用RMAN工具
在线考试系统数据库设计
在.NET中连接Oracle
在Java中连接Oracle



客服热线:(010) 88378991, 88361066
购书热线:(010) 68326294, 88379649, 68895259
投稿热线:(010) 88379604
读者信箱:hzsjz@hzbook.com

华章网站 <http://www.hzbook.com>

网上购书: www.china-pub.com



定价: 59.00元 (附光盘)

程序员书库 >>>

视频
实战版

Oracle

从入门到精通

11小时多媒体教学视频)

秦 靖 刘存勇 等编著



机械工业出版社
China Machine Press

Oracle 11g是甲骨文公司精心打造的最新版本的Oracle数据库,该版本数据库凝聚了Oracle三十多年的精华。本书不仅是一本Oracle 11g的入门教材,也是初学者快速掌握Oracle 11g的捷径。全书以Oracle 11g为例,分为4篇,循序渐进地讲述了Oracle 11g的基本语法和基本操作,从数据库的安装开始逐步介绍与数据库交互的语句以及管理数据库中的文件、备份与恢复数据库等操作。在数据库应用篇中,还结合.NET和Java的编程环境讲解了如何连接Oracle 11g数据库。

本书详细介绍了Oracle 11g中数据操作和管理的基本知识,突出了数据库操作的实用性和技巧性,其中大量应用了示例讲解数据库使用的每一个知识点。本书适合正在学习使用Oracle 11g以及想提高数据库管理知识的用户阅读,并可作为开发人员的参考手册。

封底无防伪标均为盗版

版权所有,侵权必究

本书法律顾问 北京市展达律师事务所

图书在版编目(CIP)数据

Oracle从入门到精通:视频实战版 / 秦靖等编著. —北京:机械工业出版社, 2011.1
(程序员书库)

ISBN 978-7-111-32448-5

I. O… II. 秦… III. 关系数据库—数据库管理系统, Oracle 11g IV. TP311.138

中国版本图书馆CIP数据核字(2010)第218044号

机械工业出版社(北京市西城区百万庄大街22号 邮政编码 100037)

责任编辑:鄧朝怡

三河市明辉印装有限公司印刷

2011年1月第1版第1次印刷

203mm×260mm · 29.25印张

标准书号:ISBN 978-7-111-32448-5

ISBN 978-7-89451-756-2(光盘)

定价:59.00元(附光盘)

凡购本书,如有缺页、倒页、脱页,由本社发行部调换

客服热线:(010) 88378991; 88361066

购书热线:(010) 68326294; 88379649; 68995259

投稿热线:(010) 88379604

读者信箱:hzjsj@hzbook.com

丛 书 序

宝剑锋从磨砺出 梅花香自苦寒来

当前,软件开发行业对人才的需求越来越大,所以有越来越多的人开始学习编程,越来越多的电脑学校和培训班开设了编程类课程,图书市场上也涌现出大量的计算机编程类图书,有入门的、高端的、专项技术的,等等。但如此琳琅满目的图书,却并不容易找到非常适合入门人员阅读的图书。通过对已出版图书的分析和研究,我们得出结论:编排不科学,没有注意到入门人员的学习需求和规律是最大的问题所在,因此导致很多图书都不适合入门人员阅读和学习。

为了给广大入门读者提供一套易学好用的编程图书,我们策划了本丛书,希望在本丛书的引领下,读者可以轻松跨入计算机程序设计的大门。本丛书在编写时考虑到了入门读者学习的难点,力求通俗易懂,将学习的门槛降到最低。另外,本丛书在策划时考虑了相关学校和培训机构的课程设置,适合作为相关教材。

丛书书目

《C语言从入门到精通 (视频实战版)》
《C++语言从入门到精通 (视频实战版)》
《Java从入门到精通 (视频实战版)》
《C# 4.0从入门到精通 (视频实战版)》
《Visual C++从入门到精通 (视频实战版)》
《ASP.NET 4.0从入门到精通 (视频实战版)》
《Java Web从入门到精通 (视频实战版)》
《JavaScript从入门到精通 (视频实战版)》
《ActionScript 3.0从入门到精通 (视频实战版)》
《Oracle从入门到精通 (视频实战版)》

丛书特色

1. 全程多媒体语音教学视频

本丛书的每一本图书都配有多媒体语音教学视频,读者通过书盘结合,可以轻松地掌握书中的内容。



2. 内容编排科学, 避免读者走弯路

本丛书遵循“基本概念→语法讲解→示例讲解→实践练习”的模式, 全书最后还安排了项目开发案例。这样的内容安排符合读者的学习规律, 可以避免读者走弯路。

3. 讲解通俗易懂, 易于理解

本丛书在讲解知识时采用通俗易懂的语言, 必要时采用比喻和类比等写作手法, 让抽象的编程知识变得具体化, 读者理解起来毫不费力。

4. 给出了大量实例, 实用性强

本丛书在讲解过程中穿插了大量有针对性的实例, 并且提供了项目开发案例, 读者可以通过学习实例, 加深对概念和语法的理解, 并且通过项目开发演练, 理解实际开发。

5. 代码注释丰富, 易于阅读

本丛书中出现的源代码都为关键代码, 这些代码都提供了丰富的注释, 读者阅读起来比较容易理解, 学习效果好。

6. 光盘内容实用、超值

本丛书的配套光盘中提供了教学视频、书中涉及的源代码、教学PPT, 还特别赠送了一些相关的编程视频和其他学习资料。

7. 提供技术支持

读者如果在阅读过程中遇到问题, 可以通过技术论坛寻求支持, 论坛地址:
<http://www.rzchina.net>。

阅读建议

- ☐ 没有基础的读者从第1章顺次阅读, 尽量不要跳跃。
- ☐ 重视对书中概念的理解, 这样才能为后面的学习打好基础。
- ☐ 亲自动手将书中的实例做一遍, 以加深对内容的理解。
- ☐ 认真阅读书中的源代码, 养成良好的编码习惯, 这会让您受益匪浅。
- ☐ 尝试学完每章后独立完成书中提供的习题。
- ☐ 不妨经常回过头来回顾一下已经学过的知识, 也许会有一个新的认识。
- ☐ 遇到问题时, 学会利用网络资源解决。

希望本丛书能解决您在学习程序设计的过程中遇到的各种疑难问题, 带您轻松跨入编程的大门, 为未来的职业发展奠定一个好的基础。

前 言

Oracle 11g是Oracle产品历经30年的产物，也是当前企业级开发的首选。使用数据库可以说是每一个软件开发人员必须掌握的技能，虽然数据库如此重要，但并不是每一个软件开发人员都能对数据库的使用达到运用自如的程度。而在实际的项目开发中，好的数据库设计方案不仅能够帮助软件开发人员快速完成项目的开发，还能够确保数据库的安全。

鉴于以上的考虑，笔者以Oracle 11g为例，结合数据库在企业中的应用，讲述了从数据库的发展到数据库的使用，再到数据库在实际项目中的应用的全过程。

本书特点

本书深入浅出地讲解了数据库的基础知识，同时结合目前数据库开发常用的软件（如PL/SQL Developer、SQL*Plus）演示在实际应用中常用的数据库操作方法。在每章的最后，笔者还结合本章讲解的内容，为读者提供了练习。

本书基本涵盖了Oracle 11g使用的各个方面的知识，从数据库发展到数据库设计，从PL/SQL的编程到数据库的管理，从Oracle自身的开发到结合实际编程工具进行开发，讲述了各种SQL语句以及PL/SQL语句的基本知识，讲解了在Oracle 11g中提供的各种管理功能的使用方法，还介绍了在.NET环境和Java环境中如何使用Oracle 11g进行软件开发。

本书的特点主要体现在以下几个方面。

- 本书的编排采用循序渐进的方式，以便初中级读者逐步掌握Oracle 11g数据库操作的基本方法、数据库设计和管理精髓。
- 本书深入浅出地介绍了数据库的使用和管理知识，在每章的开始指出本章的学习目标，在每章的结尾指出通过本章学习读者应该能够掌握的知识，并为读者提供有针对性的练习。
- 本书在介绍数据库基本语句时，对每一个语句都给出了有针对性的示例，方便了读者对语句的学习。在介绍数据库的管理方法时，采用了浅显易懂的例子，一步一步地完成数据库的相关操作。同时，也为Oracle 11g数据库的初学者讲解了如何使用企业管理器管理数据库，大大解决了初学者在管理数据库时一些头疼的问题。为了方便软件开发人员快速理解如何使用Oracle 11g数据库，本书的最后还提供了数据库设计以及使用.NET和Java连接Oracle 11g数据库进行软件开发的讲解。
- 本书结合笔者多年的开发经验，在讲解Oracle 11g的使用和管理时，把操作时经常会出现的错误和需要注意的问题也融入到书中，提醒读者在使用数据库时要注意的问题。



本书内容

读书笔记

本书分为4篇,共22章,从数据库的基本概念讲起,再进一步介绍SQL语句以及PL/SQL语句的使用,然后结合目前常用的数据库开发工具,讲解了数据库中基本的查询、语句块编程、控制文件、日志文件以及数据库的备份恢复等常用的数据库操作,最后结合笔者的经验讲解如何在.NET和Java环境中连接Oracle 11g数据库,让软件开发人员能够结合数据库完成实际的软件开发。

第一篇(第1~2章)为数据库安装篇。首先讲述了数据库的发展,Oracle 11g的发展历程,Oracle 11g中的新特性,比较了Oracle 11g与SQL Server数据库的功能。然后在该基础上介绍了如何安装和卸载Oracle 11g数据库。通过这两章的学习,读者应该能够在了解Oracle 11g发展的同时完成Oracle 11g的安装。

第二篇(第3~12章)为数据库基础篇。讲述了数据库使用的基本知识,包括数据库的设计方法及E-R图的使用、SQL语句的使用、PL/SQL语句的使用、视图的使用、存储过程的使用、触发器的使用、游标的使用以及事务和锁的使用。通过这10章的学习,能够使读者基本掌握使用数据库的基本语句。

第三篇(第13~19章)为数据库管理篇。具体讲述了数据库管理的主要操作,包括管理工具的使用、表空间的操作、用户与权限的操作、控制文件和日志文件的操作以及备份与恢复等操作。通过这7章的学习,读者可以从一个数据库的基本操作者转变成为一个数据库管理员。

第四篇(第20~22章)为数据库应用篇。主要介绍如何设计和使用数据库进行软件开发。在第20章中以在线考试系统为例完成了数据库的设计,在第21章和第22章中分别结合.NET和Java开发环境讲解了如何连接Oracle 11g数据库。通过这3章的学习,读者可以使用Oracle 11g结合编程环境完成软件的开发。

本书由浅入深,由理论到实践,尤其适合初级读者逐步学习和完善自己的知识结构。

适合阅读本书的读者

- ☐ 希望学习Oracle 11g的新手。
- ☐ 迫切希望提高Oracle 11g应用水平的开发人员。
- ☐ 具备一定的项目开发经验但缺乏对Oracle 11g实践的软件工程师。
- ☐ 希望在项目开发中使用Oracle 11g的软件开发人员。
- ☐ 希望从事Oracle 11g数据库管理工作的软件开发人员。
- ☐ 希望提高Oracle 11g项目开发水平的数据库管理员。
- ☐ 各类大中专院校或培训学校的学生。

本书作者

本书主要由秦靖、刘存勇编写,其他参与编写和资料整理的人员有高会东、王建超、邓薇、黄丽莉、各晓宁、汪洋、白广元、蔡念光、陈辉、冯彬、刘长江、刘明、沙金、张士强、张洪福、多召英、贾旭、李宽、江宽、陈科、方成林、班晓娟、方中纯、刘兰军、郑雪峰。

编者

2010年12月

目 录

丛书序
前言

第一篇 数据库安装篇

第1章 Oracle 11g数据库简介	1
1.1 认识Oracle 11g	1
1.1.1 Oracle的成长历程	1
1.1.2 了解最新版本Oracle 11g	2
1.1.3 Oracle与SQL Server	2
1.2 回忆Oracle的产品版本	3
1.3 学习Oracle 11g的新特性	4
1.3.1 数据库管理部分	4
1.3.2 PL/SQL部分	5
1.4 小结	5
1.5 习题	5
第2章 Oracle 11g的安装与测试	6
2.1 使用Oracle 11g的基本条件	6
2.1.1 硬件条件	6
2.1.2 软件条件	6
2.2 在Windows 2003系统中安装Oracle 11g	7
2.2.1 获取Oracle 11g的安装文件	7
2.2.2 安装Oracle 11g	8
2.2.3 安装中遇到的问题	16
2.3 移除Oracle 11g	17
2.3.1 停止服务列表的5个服务	17
2.3.2 卸载Oracle产品	18
2.3.3 删除注册表项	18
2.3.4 删除环境变量	19
2.3.5 删除目录并重启计算机	19
2.4 小结	19
2.5 习题	20

第3章 熟悉数据库	21
3.1 什么是数据库	21
3.1.1 了解数据管理的历史	21
3.1.2 数据库的模型	23
3.1.3 学习数据库的三级模式和二级映像	25
3.1.4 数据库中的相关术语	26
3.1.5 数据库设计的完整性	28
3.2 范式——设计关系型数据库的准则	28
3.2.1 第一范式——关系型数据库设计的第一步	29
3.2.2 第二范式——关系型数据库设计的第二步	29
3.2.3 第三范式——关系型数据库设计的第三步	30
3.3 绘制E-R图设计数据库	30
3.3.1 绘制E-R图的基本要素	30
3.3.2 E-R图绘制实例	31
3.4 小结	33
3.5 习题	33
第4章 SQL基础	34
4.1 SQL——数据库沟通的语言标准	34
4.1.1 什么是SQL	34
4.1.2 了解SQL的种类	34
4.2 Oracle 11g中支持的数据类型	35
4.2.1 查看Oracle 11g中的数据类型	35
4.2.2 常用数据类型	37
4.3 数据定义语言 (DDL)	38
4.3.1 使用Create语句创建表	38
4.3.2 使用Alter语句修改表	39
4.3.3 使用Drop语句删除表	41
4.4 约束的使用	41
4.4.1 主键约束	41
4.4.2 外键约束	42
4.4.3 CHECK约束	44
4.4.4 UNIQUE约束	45
4.4.5 NOT NULL约束	46
4.5 数据操纵语言 (DML) 和数据查询语言 (DQL)	47
4.5.1 添加数据就用INSERT	48
4.5.2 修改数据就用UPDATE	50
4.5.3 删除数据就用DELETE	51
4.5.4 查询数据就用SELECT	51



4.5.5 其他数据操纵语句	52
4.6 数据控制语言 (DCL)	55
4.7 小结	55
4.8 习题	55
第5章 利用SELECT检索数据	56
5.1 查询数据必备SELECT	56
5.1.1 SELECT语句语法	56
5.1.2 获取指定字段的数据	57
5.1.3 获取所有字段的数据	58
5.1.4 使用别名替代表中的字段名	59
5.1.5 使用表达式操作查询的字段	59
5.1.6 使用函数操作查询的字段	60
5.1.7 去除检索数据中的重复记录	60
5.2 检索出来的数据排序	61
5.2.1 使用排序的语法	61
5.2.2 使用升序和降序来处理数据	62
5.2.3 排序时对NULL值的处理	62
5.2.4 使用别名作为排序字段	63
5.2.5 使用表达式作为排序字段	64
5.2.6 使用字段的位置作为排序字段	64
5.2.7 使用多个字段排序	65
5.3 使用WHERE子句设置检索条件	66
5.3.1 查询中使用单一条件限制	66
5.3.2 查询中使用多个条件限制	67
5.3.3 模糊查询数据	68
5.3.4 查询条件限制在某个列表范围之内	69
5.3.5 专门针对NULL值的查询	70
5.4 GROUP BY和HAVING子句	70
5.4.1 GROUP BY子句语法及使用	70
5.4.2 HAVING子句的使用	72
5.5 使用子查询	73
5.5.1 子查询返回单行	73
5.5.2 子查询返回多行	74
5.6 连接查询	76
5.6.1 最简单的连接查询	76
5.6.2 内连接	77
5.6.3 自连接	78
5.6.4 外连接	79
5.7 小结	82
5.8 习题	82



第6章 Oracle内置函数	83
6.1 数值型函数	83
6.1.1 绝对值、取余、判断数值正负函数	83
6.1.2 三角函数	84
6.1.3 返回以指定数值为准整数的函数	84
6.1.4 指数、对数函数	85
6.1.5 四舍五入截取函数	86
6.2 字符型函数	86
6.2.1 ASCII码与字符转换函数	86
6.2.2 获取字符串长度函数	87
6.2.3 字符串截取函数	87
6.2.4 字符串连接函数	88
6.2.5 字符串搜索函数	88
6.2.6 字母大小写转换函数	88
6.2.7 带排序参数的字母大小写转换函数	89
6.2.8 为指定参数排序函数	90
6.2.9 替换字符串函数	91
6.2.10 字符串填充函数	91
6.2.11 删除字符串首尾指定字符的函数	92
6.2.12 字符集名称和ID互换函数	93
6.3 日期型函数	93
6.3.1 系统日期、时间函数	93
6.3.2 得到数据库时区函数	94
6.3.3 为日期加上指定月份函数	94
6.3.4 返回指定月份最后一天函数	95
6.3.5 返回指定日期后一周的日期函数	95
6.3.6 返回会话所在时区当前日期函数	95
6.3.7 提取指定日期特定部分的函数	96
6.3.8 得到两个日期之间的月份数	96
6.3.9 时区时间转换函数	97
6.3.10 日期四舍五入、截取函数	97
6.4 转换函数	98
6.4.1 字符串转ASCII类型字符串函数	98
6.4.2 二进制转十进制函数	98
6.4.3 数据类型转换函数	99
6.4.4 字符串和ROWID相互转换函数	99
6.4.5 字符串在字符集间转换函数	100
6.4.6 十六进制字符串与RAW类型相互转换函数	100
6.4.7 数值转换成字符型函数	101
6.4.8 字符转日期型函数	101



6.4.9 字符串转数字函数	102
6.4.10 全角转半角函数	102
6.5 NULL函数	102
6.5.1 返回表达式为NULL的函数	103
6.5.2 排除指定条件函数	103
6.5.3 替换NULL值函数	103
6.6 集合函数	104
6.6.1 求平均值函数	104
6.6.2 求记录数量函数	105
6.6.3 返回最大、最小值函数	106
6.6.4 求和函数	106
6.7 其他函数	107
6.7.1 返回登录名函数	107
6.7.2 返回会话以及上下文信息函数	107
6.7.3 表达式匹配函数	108
6.8 小结	108
6.9 习题	109
第7章 PL/SQL基础	110
7.1 什么是PL/SQL	110
7.1.1 认识PL/SQL	110
7.1.2 PL/SQL的优势	111
7.1.3 PL/SQL的结构	112
7.1.4 PL/SQL的基本规则	114
7.1.5 PL/SQL中的注释	115
7.2 PL/SQL 变量的使用	116
7.2.1 变量、常量的类型及语法	116
7.2.2 标量类型的变量	117
7.2.3 复合类型的变量	120
7.3 表达式	127
7.3.1 数值表达式	127
7.3.2 关系表达式和逻辑表达式	128
7.4 PL/SQL结构控制	129
7.4.1 IF条件控制语句	129
7.4.2 CASE条件控制语句	135
7.4.3 LOOP循环控制语句	138
7.5 PL/SQL中使用DML和DDL语言	144
7.5.1 DML语句的使用	144
7.5.2 DDL语句的使用	146
7.6 PL/SQL中的异常	147
7.6.1 什么是异常	147



7.6.2 处理异常的语法	147
7.6.3 预定义异常	148
7.6.4 非预定义异常	149
7.6.5 自定义异常	150
7.7 PL/SQL函数编写	152
7.7.1 函数的组成	152
7.7.2 函数语法	152
7.7.3 查看函数	155
7.7.4 在PL/SQL Developer中创建函数	157
7.7.5 函数的修改、删除	158
7.8 小结	159
7.9 习题	159
第8章 游标——数据的缓存区	161
8.1 什么是游标	161
8.1.1 游标的概念	161
8.1.2 游标的种类	161
8.2 显式游标	162
8.2.1 游标语法	162
8.2.2 游标的使用步骤	162
8.2.3 游标中的LOOP语句	164
8.2.4 使用BULK COLLECT和FOR语句的游标	165
8.2.5 使用CURSOR FOR LOOP	167
8.2.6 显式游标的属性	167
8.2.7 带参数的游标	171
8.3 隐式游标	172
8.3.1 隐式游标的特点	172
8.3.2 游标中使用异常处理	173
8.3.3 隐式游标的属性	174
8.4 有关游标的案例	177
8.5 小结	181
8.6 习题	181
第9章 视图——数据库中虚拟的表	182
9.1 什么是视图	182
9.1.1 认识视图	182
9.1.2 视图的作用	182
9.1.3 视图的语法	183
9.2 视图的创建	183
9.2.1 创建单表视图	183
9.2.2 创建多表视图	185
9.2.3 创建视图的视图	187



9.2.4 创建没有源表的视图	188
9.2.5 使用PL/SQL工具创建视图	188
9.2.6 创建带约束的视图	191
9.2.7 有关视图的案例	192
9.3 操作视图数据的限制	194
9.3.1 视图READ ONLY设置	194
9.3.2 视图CHECK OPTION设置	195
9.3.3 视图创建语句对视图操作的影响	196
9.4 视图的修改	196
9.4.1 视图结构的修改	197
9.4.2 视图约束的修改	198
9.5 视图的删除	199
9.6 小结	200
9.7 习题	201
第10章 存储过程——提高程序执行的效率	202
10.1 什么是存储过程	202
10.1.1 认识存储过程	202
10.1.2 存储过程的作用	202
10.1.3 存储过程的语法	203
10.2 在SQL*Plus中创建存储过程	203
10.2.1 创建第一个存储过程	203
10.2.2 查看存储过程	205
10.2.3 显示存储过程的错误	205
10.2.4 无参存储过程	206
10.2.5 存储过程中使用游标	208
10.2.6 存储过程中的DDL语句	209
10.2.7 有参存储过程	213
10.3 使用PL/SQL工具创建存储过程	220
10.3.1 在PL/SQL Developer中创建存储过程	220
10.3.2 调试存储过程	224
10.4 修改、删除存储过程	225
10.4.1 修改存储过程	226
10.4.2 删除存储过程	227
10.5 小结	227
10.6 习题	227
第11章 触发器——保证数据的正确性	228
11.1 什么是触发器	228
11.1.1 认识触发器	228
11.1.2 触发器的作用	228
11.1.3 触发器的类型	229

11.1.4	触发器的语法	230
11.2	使用SQL*Plus操作触发器	232
11.2.1	利用SQL*Plus创建触发器	233
11.2.2	查看触发器	233
11.2.3	DML类型触发器	234
11.2.4	触发器执行顺序	244
11.2.5	复合类型触发器	245
11.2.6	INSTEAD OF类型触发器	248
11.2.7	DDL类型触发器	249
11.2.8	用户和系统事件触发器	251
11.3	使用PL/SQL工具操作触发器	252
11.3.1	在PL/SQL Developer中创建触发器	252
11.3.2	设置触发器是否可用	254
11.4	修改、删除触发器	255
11.4.1	修改触发器	255
11.4.2	删除触发器	256
11.5	小结	256
11.6	习题	257
第12章	事务和锁——确保数据安全	258
12.1	什么是事务	258
12.1.1	认识事务	258
12.1.2	事务的类型	259
12.1.3	事务的保存点	261
12.1.4	事务的ACID特性	262
12.2	什么是锁	262
12.2.1	认识锁	263
12.2.2	锁的分类	263
12.2.3	锁的类型	263
12.2.4	锁等待与死锁	265
12.3	小结	269
12.4	习题	269

第三篇 数据库管理篇

第13章	使用Oracle 11g Enterprise Manager	271
13.1	什么是Oracle 11g Enterprise Manager	271
13.1.1	Oracle 11g Enterprise Manager概述	271
13.1.2	启动OEM	272
13.2	使用OEM管理Oracle	273
13.2.1	OEM中的性能菜单	273



13.2.2 OEM中的可用性菜单	274
13.2.3 OEM中的服务器菜单	275
13.2.4 OEM中的方案菜单	292
13.2.5 OEM中的数据移动菜单	302
13.2.6 OEM中的软件和支持菜单	302
13.3 小结	303
13.4 习题	303
第14章 常用工具介绍	304
14.1 什么是SQL*Plus	304
14.1.1 SQL*Plus简介	304
14.1.2 启动SQL*Plus	304
14.2 使用SQL*Plus	305
14.2.1 使用SQL*Plus编辑命令	306
14.2.2 使用SQL*Plus保存命令	310
14.2.3 使用SQL*Plus运行命令	311
14.2.4 使用SQL*Plus格式化查询结果	311
14.2.5 在SQL*Plus中为语句添加注释	319
14.3 使用PL/SQL Developer	321
14.3.1 PL/SQL Developer的安装	321
14.3.2 PL/SQL Developer的布局	322
14.4 小结	323
14.5 习题	323
第15章 控制文件和日志文件	324
15.1 控制文件与日志文件概述	324
15.1.1 什么是控制文件	324
15.1.2 什么是日志文件	325
15.2 初识控制文件	326
15.2.1 控制文件的内容	326
15.2.2 更新控制文件	327
15.3 控制文件的多路复用	328
15.3.1 使用init.ora多路复用控制文件	328
15.3.2 使用SPFILE多路复用控制文件	328
15.4 创建控制文件	330
15.5 日志文件的管理	332
15.5.1 新建日志文件组	332
15.5.2 添加日志文件到日志文件组	335
15.5.3 删除日志文件组和日志文件	336
15.5.4 查询日志文件组和日志文件	338
15.6 小结	339
15.7 习题	339



第16章 表空间的管理	340
16.1 表空间概述	340
16.1.1 相关概念	340
16.1.2 默认表空间	340
16.2 表空间的管理	343
16.2.1 创建表空间	343
16.2.2 重命名表空间	345
16.2.3 设置表空间的读写状态	346
16.2.4 设置表空间的可用状态	347
16.2.5 建立大文件表空间	348
16.2.6 删除表空间	349
16.3 临时表空间的管理	350
16.3.1 建立临时表空间	350
16.3.2 查询临时表空间	351
16.3.3 创建临时表空间组	351
16.3.4 查询临时表空间组	354
16.3.5 删除临时表空间组	355
16.4 数据文件管理	356
16.4.1 移动数据文件	356
16.4.2 删除数据文件	356
16.5 小结	358
16.6 习题	358
第17章 与数据库安全性有关的对象	359
17.1 用户管理	359
17.1.1 什么是用户管理	359
17.1.2 创建用户	359
17.1.3 修改用户信息	363
17.1.4 删除用户	364
17.2 权限管理	365
17.2.1 什么是权限管理	365
17.2.2 授予权限	365
17.2.3 撤销权限	369
17.2.4 查询用户的权限	370
17.3 角色管理	372
17.3.1 什么是角色	372
17.3.2 创建角色	372
17.3.3 设置角色	375
17.3.4 修改角色	376
17.3.5 删除角色	376
17.3.6 查询角色	377



17.4 概要文件PROFILE	377
17.4.1 什么是PROFILE	377
17.4.2 创建PROFILE	377
17.4.3 修改PROFILE	380
17.4.4 删除PROFILE	381
17.4.5 查询PROFILE	381
17.5 小结	382
17.6 习题	382
第18章 备份与恢复	383
18.1 数据库备份与恢复	383
18.1.1 什么是数据库备份	383
18.1.2 什么是数据库恢复	383
18.2 物理备份和恢复数据库	383
18.2.1 对数据库进行脱机备份	384
18.2.2 对数据库进行联机备份	384
18.3 逻辑备份和恢复数据库	387
18.3.1 逻辑导出数据	387
18.3.2 逻辑导入数据	391
18.4 小结	392
18.5 习题	392
第19章 使用RMAN工具	393
19.1 RMAN概述	393
19.1.1 RMAN的特点	393
19.1.2 与RMAN有关的概念	393
19.2 使用恢复目录	394
19.2.1 创建恢复目录	394
19.2.2 使用RMAN连接	396
19.2.3 在恢复目录中注册数据库	398
19.2.4 使用企业管理器创建恢复目录	398
19.3 通道分配	401
19.3.1 什么是通道分配	401
19.3.2 手动通道分配	401
19.3.3 自动通道分配	404
19.4 备份集	405
19.4.1 什么是备份集	405
19.4.2 BACKUP的使用	405
19.5 从备份中恢复	406
19.5.1 使用RESTORE还原	406
19.5.2 使用RECOVER恢复	407
19.6 小结	407
19.7 习题	407



第20章 在线考试系统数据库设计	409
20.1 在线考试系统需求	409
20.2 模块设计	410
20.2.1 模块分类	410
20.2.2 数据库总体结构	411
20.2.3 数据库表结构	411
20.2.4 建表脚本	416
20.3 小结	420
20.4 习题	420
第21章 在.NET中连接Oracle	421
21.1 什么是ADO.NET	421
21.1.1 ADO.NET概述	421
21.1.2 ADO.NET中的对象	421
21.2 使用绑定的方式连接Oracle	422
21.2.1 数据控件概述	422
21.2.2 使用DataGridView控件绑定Oracle数据库	422
21.3 使用写代码的方式连接Oracle	427
21.3.1 使用Command对象操作Oracle数据库	427
21.3.2 使用DataSet对象存储查询结果	429
21.3.3 商品信息存储实例	430
21.4 小结	434
21.5 习题	435
第22章 在Java中连接Oracle	436
22.1 JDBC与ODBC简介	436
22.1.1 什么是JDBC	436
22.1.2 什么是ODBC	437
22.2 Thin方式连接Oracle	438
22.3 JDBC-ODBC桥连接Oracle	446
22.3.1 配置ODBC数据源	446
22.3.2 使用JDBC-ODBC桥连接Oracle	448
22.4 小结	448
22.5 习题	449

第一篇

数据库安装篇

第1章 Oracle 11g数据库简介

Oracle 11g是甲骨文公司推出的最新版本的Oracle数据库软件，它在以前版本的基础上得到了进一步的改进和发展。本章包括以下知识点：

- Oracle 11g的发展
- Oracle 11g的新特性
- Oracle 11g与SQL Server的区别

本章内容基本涵盖了Oracle 产品版本的介绍和Oracle 11g的一些新特性。通过本章的学习，读者可以了解Oracle 11g的特性以及与SQL Server的区别。

1.1 认识Oracle 11g

Oracle 11g是Oracle数据库系列的最新版本，目前已经被企业广泛应用。本节将讲述Oracle的发展和Oracle 11g的介绍以及Oracle 11g与微软的SQL Server数据库的对比。

1.1.1 Oracle的成长历程

Oracle数据库目前已经成为企业级开发的首选，那么Oracle数据库是谁创建的呢？又是由哪个公司创建的？说到Oracle数据库的创始人，不得不提到劳伦斯·埃里森（Larry Ellison），如图1.1所示，他出生于美国纽约布朗克斯，并不是什么名牌大学毕业，甚至就读三所大学都没能取得一个学位。但是，就是这样一个人，却成为世界上第二大软件公司的创始人。

正所谓天生我材必有用，劳伦斯·埃里森自学了计算机编程。由于20世纪60年代美国也动荡不安，所以劳伦斯·埃里森不断地跳槽，适应着不同公司的工作，直到他跳到一家影像器材公司工作，认识了创建Oracle公司的另外两位重要人物Bob Miner和Edward Oates，改变了他的一生。1977年，劳伦斯·埃里森与这两个重要人物在硅谷共同出资创办了一家软件开发



图1.1 Oracle的创始人Larry Ellison



公司，其中劳伦斯·埃里森占有60%的股权。

在劳伦斯·埃里森创建软件开发公司的初期，并不是一开始就确定要开发数据库产品，也不是一开始就把公司的名字称为Oracle。之所以要开发数据库产品，是受IBM的研究员发表的一篇文章《大型共享数据库的关系数据模型》的启发，3个人才开始研发关系型数据库的。他们的第一个项目是给美国政府做的，项目的名字当时就叫Oracle，Oracle在英语中的意思是神谕宣誓、预言或圣言。此后，劳伦斯·埃里森就把研发的数据库叫做Oracle，后来也把自己公司的名字改成了Oracle。

Oracle数据库的第一商用版本是在1979年诞生的，到现在的Oracle 11g版本已经历经30余年了，在这些年中，Oracle公司的Oracle产品也成为家喻户晓的产品，劳伦斯·埃里森的名字也被《福布斯》排行榜收录。这就是Oracle，也是甲骨文公司的成长历程。

1.1.2 了解最新版本Oracle 11g

Oracle 11g是目前最新的Oracle版本，可以在Oracle的官方网站www.oracle.com获取Oracle的版本信息。

本书中要讲解的是Oracle 11g的第1版，所以在这里只对Oracle 11g的各版本做以说明。Oracle 11g分为Oracle 11g 11.1.0.7.0和Oracle 11g 11.1.0.6.0两个版本。其中，Oracle 11g 11.1.0.7.0版本主要安装在Windows Server 2008系统中使用。Oracle 11g 11.1.0.6.0版本中所支持的操作系统比较广泛，主要有：

- ☐ Microsoft Windows (32位)
- ☐ Microsoft Windows (x64)
- ☐ Linux x86
- ☐ Linux x86-64
- ☐ Solaris (SPARC) (64位)
- ☐ AIX (PPC64)
- ☐ HP-UX Itanium
- ☐ HP-UX PA-RISC (64位)

安装到Windows 2003系统或者Windows XP系统中时，使用Microsoft Windows (32位)即可。在第2章中将具体讲述Oracle的安装过程。

1.1.3 Oracle与SQL Server

Oracle 数据库与SQL Server数据库在企业应用当中各自有着用武之地。Oracle数据库的最新版本是Oracle 11g，而SQL Server数据库也有了最新的SQL Server 2008版本。下面从几个方面讲解Oracle数据库与SQL Server数据库的特点。

1. 对操作系统的支持

Oracle数据库对操作系统的支持比SQL Server数据库更多。Oracle可以支持的操作系统有Windows系统、Linux系统、苹果的操作系统等；而SQL Server由于是微软研发的，所以目前支持的操作系统只有Windows 操作系统。另外，对于SQL Server数据库，通常只能在Windows Server系列的操作系统上安装企业版，而普遍使用的Windows XP系统只能安装SQL Server的个人版或开发版。因此，当数据库服务器是非Windows系统时，只能使用Oracle数据库；当数据库服务器是Windows操作系统时，才可以考虑使用SQL Server数据库。



2. 数据库的架构

在Oracle数据库中,一个实例只能管理一个数据库,只有数据库在集群的环境下才能实现多个数据库被一个实例管理;而SQL Server数据库是一个实例管理多个数据库。

3. 数据库的安全性

SQL Server系统数据库没有通过安全性认证,而Oracle数据库是获得ISO安全认证的数据库,所以说Oracle的安全性更好一些。

4. 内存分配

Oracle的内存分配大部分是由INIT.ORA来决定的,而SQL的内存分配主要有动态内存分配和静态内存分配。

1.2 回忆Oracle的产品版本

Oracle发展到目前的Oracle 11g版本,是历经30多年努力实现的成果,本节将带你回忆Oracle的整个发展历程。

- ❑ 最早的Oracle版本是在1979年的夏季发布的,该版本在Oracle出品时被称为Oracle的第2版,这也是出于营销的考虑。这个数据库产品整合了比较完整的SQL实现,其中包括子查询、连接及其他特性。
- ❑ 1983年3月,Oracle发布了第3版。该版本是使用C语言重新编写第2版得到的,这也是为什么Oracle的可移植性非常好的主要原因。
- ❑ 1984年10月,Oracle发布了第4版,产品的稳定性得到很大的提高。
- ❑ 1985年,Oracle发布了第5版。该版本的稳定性又有了很大的提高,这也是首批可以在C/S模式下运行的数据库产品。
- ❑ 1988年,Oracle发布了第6版。在发布第6版之前,Oracle公司已经上市了。第6版中引入了很多新特性,主要有行级锁、联机热备份等功能,在一定程度上增强了Oracle的可用性。
- ❑ 1992年6月,Oracle发布了第7版。在该版本中增加的新特性主要有:分布式事务处理能力、对数据库管理功能的增强,同时也提高了数据库的安全性。Oracle在第7版产品的推进下销售额倍增。
- ❑ 1997年6月,Oracle发布了第8版。Oracle 8支持面向对象的开发及新的多媒体应用,这个版本也为支持Internet、网格计算等奠定了基础。同时,这一版本开始具有同时处理大量用户和海量数据的特性。
- ❑ 1998年9月,Oracle发布了8i版,“i”代表Internet。该版本中添加了大量为支持Internet而设计的特性。同时,这一版本为数据库用户提供了全方位的Java支持。Oracle 8i成为第一个完全整合了本地Java运行环境的数据库,用Java就可以编写Oracle的存储过程。
- ❑ 2001年6月,Oracle发布了Oracle 9i。在Oracle 9i的诸多新特性中,最重要的就是Real Application Clusters (RAC)了。说起Oracle集群服务器,早在第5版时,Oracle就开始开发Oracle并行服务器(Oracle Parallel Server, OPS),并在以后的版本中逐渐完善了其功能。不过,严格来说,尽管OPS算得上是个集群环境,但是并没有体现出集群技术应有的优点。

- 2003年9月8日, Oracle发布了10g版本。10g版本是Oracle应用服务器版本, 这里的“g”代表“grid”(网格)。在这个版本中加入了网格计算的功能。
- 2007年11月, Oracle发布了11g版本。这也是目前最新的版本, 该版本大大提高了系统性能的安全性, 并有了多项的创新, 比如实现了信息生命周期管理、全新的Data Guard将可用性最大化, 同时利用数据压缩技术大幅降低了数据存储的支出; 缩短了应用程序测试环境部署所花费的时间。

1.3 学习Oracle 11g的新特性

Oracle 11g是目前使用比较多的一个版本, 也是性能比较稳定的版本。Oracle 11g在以前版本的基础上又增加了很多新的特性, 本节就带领读者认识Oracle 11g中新增加的一些特性。

1.3.1 数据库管理部分

数据库管理部分是Oracle 11g中的核心, 在这一部分, 甲骨文公司为Oracle增加了如下8个主要特性。

(1) 数据库重放

数据库重放在Oracle 11g中主要体现在两个部分, 一个是数据库重放; 一个是SQL重放。新的数据库重放工具好似数据库内的DVR。利用该方法, 可以方便地以二进制文件格式捕获SQL级别以下的所有数据库活动, 然后在同一数据库或不同数据库内进行重放。此外, 还可以自定义捕获流程, 以包括或排除某些特定类型的活动。SQL重放与数据库重放的特性类似, 但只是捕捉SQL负载部分。

(2) SQL计划管理

之前, 我们可以使用存储大纲和SQL Profile来帮助我们固定某条SQL语句的执行计划, 防止由于执行计划发生变化而导致的性能下降。不过这些技术需要DBA人为的处理, 比如存储大纲, 需要DBA手工创建, 而对于SQL Profile来说, 则要DBA手工应用才能生效。从Oracle 11g开始, 引入了SQL执行计划管理这个新特性, 从而可以让系统自动控制SQL语句执行计划的稳定性, 进而防止由于执行计划发生变化而导致的性能下降。

(3) 自动存储管理

用于管理 ASM 实例的新的SYSASM角色, 用于降低共享池使用的可变的区大小, 以及实例能够读取磁盘组的特定磁盘。

(4) 自动的健康检查

Oracle能够自动地对数据库进行健康检查, 对于有可能导致数据库在将来出现问题的一些因素, 给DBA发送告警信息, 并针对潜在问题给出一些建议。这样可以在问题严重之前发现数据库的问题, 从而避免灾难性的事件发生。

(5) 企业管理器功能的增强

在企业管理器中增加了一个LOGMINER接口, 该接口主要用作日志的查询。通过该接口, 还可以在企业管理器的GUI页面中获取日志挖掘的图形。

(6) 自动诊断知识库

当Oracle探测到重要错误时, 会自动创建一个事件, 并且捕捉到和这一事件相关的信息, 同时自动进行数据库健康检查并通知DBA。

(7) 闪回事务

使用闪回事务可以回退事务, 即使是已经提交的事务。这对于更正一些用户错误非常有用。比如, 用户不小心执行了一些事务, 并且在这些事务里对数据库的数据做了一些更改, 当用户



提交事务后,发现这些更改是错误的,想要回退这些更改,这时就可以使用闪回事务了。DBA只需要简单地将这些事务闪回,即可把用户犯下的错误更正过来。

(8) 自动内存优化

在Oracle 11g中,所有内存可以通过只设定一个参数来实现全表自动优化。你只要告诉Oracle有多少内存可用,它就可以自动指定多少内存分配给PGA、多少内存分配给SGA以及多少内存分配给操作系统进程。

1.3.2 PL/SQL部分

PL/SQL部分是指一些SQL语句的变化,通过这些变化,增强了Oracle 11g中SQL语句的功能。下面就简单讲述3个在Oracle 11g中新增加的特性。

(1) 触发器

在Oracle 11g中除了以前用的触发器之外,还引入了一个复合触发器。复合触发器就是在一个触发器中使用4部分内容,即申明部分、before过程部分、after each row过程部分和after过程部分。此外,在Oracle 11g中还可以设置触发器的顺序,比如为一个表设置几个触发器,可以指定先执行哪个触发器的内容。

(2) 对象依赖性改进

在Oracle 11g之前,如果有函数或者视图依赖于某张表,一旦这张表发生结构变化,无论是否涉及函数或视图所依赖的属性,都会使函数或视图变为invalid。在Oracle 11g中对这种情况进行了调整,如果表改变的属性与相关的函数或视图无关,则相关对象状态不会发生变化。

(3) SQL语法

在SQL语法部分中,我们在调用某一函数时,可以通过=>来为特定的函数参数指定数据。这一语法也同样可以出现在SQL语句中。

除了在数据库管理和PL/SQL部分新增加的特性之外,Oracle 11g还在数据的备份和恢复中增强了RMAN的恢复功能,提供的数据压缩技术可以最多压缩2/3,同时还提供了在线升级等功能。总之,Oracle 11g是Oracle版本中功能最强的版本,由于Oracle 11g提供的新特性很多,这里只列举了几个比较重要的新特性,其他的新特性可以登录Oracle的官方网站查看。

1.4 小结

本章首先讲述了Oracle的创始人和它的成长历程;然后通过Oracle与SQL Server的对比,讲解了Oracle和SQL Server各自的特点及使用的范围;接着讲解了Oracle产品的各个版本发布的时间和特性;最后讲述了Oracle 11g中的一些新特性,比如在数据库管理方面的数据库重放技术、SQL计划管理、自动存储管理等特性。

1.5 习题

简答题

1. 简述Oracle与SQL Server的区别。
2. Oracle的第一个版本是什么时候发布的?它的创始人是谁?
3. Oracle 11g有哪些新特性?

第2章 Oracle 11g的安装与测试

学习一个软件前,首先要安装这个软件。Oracle可以在Windows或Linux等多个操作系统上使用。鉴于目前用户普遍使用的是Windows操作系统,同时也方便初学者学习,本章主要讲解Oracle 11g在Windows 2003系统中的安装过程,包括如下知识点:

- ☐ Oracle的安装环境
- ☐ Oracle的安装
- ☐ Oracle的卸载

本章主要介绍了Oracle 11g的安装与卸载,通过本章的学习,读者能够独立地安装Oracle以及完全卸载该软件。

2.1 使用Oracle 11g的基本条件

任何软件的使用对计算机的环境都是有要求的,包括计算机的软件和硬件。本节将讲述Oracle 11g第1版(第2版暂不能稳定支持Windows系统)安装与运行的基本条件。

2.1.1 硬件条件

从硬件条件来说,目前家用的计算机都能满足要求,但考虑到很多读者都是在虚拟机下安装该软件进行学习(在虚拟机环境下,如果分配的某些硬件标准不够,容易走弯路),下面列出了在Windows系统下对硬件的具体要求,如表2.1所示。

表2.1 硬件要求

硬件项目	项目要求
物理内存(RAM)	1GB以上
虚拟内存	物理内存的两倍左右
硬盘空间	完全安装4.76GB,建议5GB以上
视频适配器	256色
处理器	550MHz以上(Vista应800MHz以上),建议1GMHz以上

其中,内存和硬盘空间是安装时比较容易出现问题的地方,用户需注意。

2.1.2 软件条件

除了硬件之外,Oracle 11g对软件环境也有一定的要求,并不是所有的系统都能够安装该软件,例如,Windows 98、Windows 2000(非SP1版)、Windows NT等操作系统都不能安装该软件。满足要求的软件环境如表2.2所示。

表2.2 软件要求

软件项目	项目要求
操作系统	Windows 2000 SP1版以上
	Windows Server 2003所有版本
	Windows Server 2003 R2所有版本
	Windows XP专业版
	Windows Vista商务版、企业版、全功能版
网络协议	Windows 2008
	支持TCP/IP、SSL加密的TCP/IP、命名管道

以上列出了32位操作系统对Oracle 11g的支持情况（稳定）。

说明 目前，官方网站中已发现有针对32位Windows系统的Oracle 11g第2版可以下载，Oracle 11g第2版支持Windows 7操作系统。用户如果有兴趣可以随时查询。软件下载地址：
<http://www.oracle.com/technology/software/products/database/index.html>。

2.2 在Windows 2003系统中安装Oracle 11g

在Windows 2003系统中安装Oracle对初学者来说并不复杂，读者只要按向导提示就能够完成Oracle 11g的安装，但是从获取Oracle 11g文件到正确安装软件这个过程中有时也会出现一些问题。本节主要讲解Oracle 11g在Windows 2003系统中安装的具体步骤。

2.2.1 获取Oracle 11g的安装文件

用户获取该软件有多种途径，最方便的方法就是从Oracle的官方网站下载该软件，由于Oracle软件用于学习是免费的，所以用户不用担心收费问题。但是，如果用于商业，则需要缴费，具体的费用可以直接在网站上查询。

1) 进入Oracle官方网站<http://www.oracle.com/index.html>，网站页面如图2.1所示。

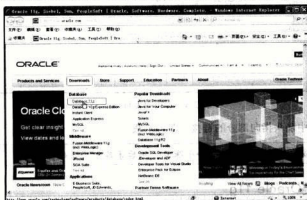


图2.1 Oracle官方网站首页



2) 在如图2.1所示的页面中, 单击【Downloads】选项卡下【Database】项下的【Database 11g】, 进入如图2.2所示页面。

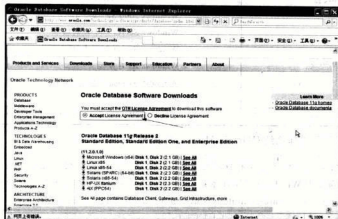


图2.2 下载列表页面1

3) 在如图2.2所示的页面中, 选中【Accept License Agreement】单选按钮, 这时该页面会出现如图2.3所示的可选链接。

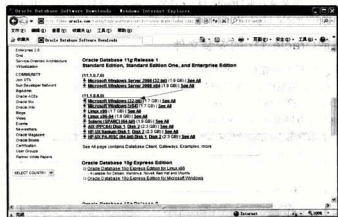


图2.3 下载列表页面2

4) 单击图2.3中箭头指向部分的链接, 即可下载。

注意 第一次下载需要读者免费注册一个账号, 然后才能下载该软件。

2.2.2 安装Oracle 11g

Oracle 11g在官方网站下载后, 软件默认名称是win32_11gR1_database_1013.zip, 解压后

就可以开始安装了。下面就详细讲解Oracle 11g的安装过程。

1. 找到Oracle安裝的可執行文件

将软件win32_11gR1_database_1013.zip解压后进入database文件夹下，文件结构如图2.4所示，其中，setup.exe文件就是Oracle安装的可执行文件。

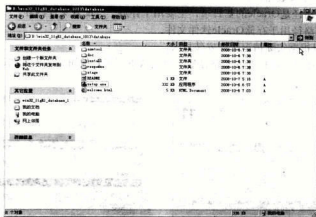


图2.4 database文件结构

2. 安装Oracle 11g软件

安装Oracle 11g软件既包括安装数据库软件本身,也包括安装数据库的实例,为了方便学习,下面把软件 and 数据库的实例分开安装。安装Oracle 11g软件分为选择安装方法、选择安装类型、选择安装路径、检查安装环境、选择配置选项、安装软件6个步骤。

1) 选择安装方法。双击如图2.4所示的setup.exe文件,弹出如图2.5所示的安装界面。这里安装分为【基本安装】和【高级安装】两个选项,其中【基本安装】比较简单,也是默认选项,但需要交互的地方相对较少。为了更全面地了解安装过程,这里选择【高级安装】选项。

2) 选择安装类型。单击【下一步】按钮，打开如图2.6所示的界面，选择安装类型。



图2.5 安装界面

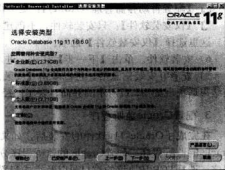


图2.6 版本选择



其中列出了Oracle 11g的4个版本。这里选择功能最全的企业版,用户可以根据自己的需要选择不同的版本,各版本的差别可以参考第1章的内容。在此界面还可以查看已安装的数据库产品和产品的语言。这里选中【企业版】单选按钮。

3) 选择安装路径。单击【下一步】按钮,进入如图2.7所示的界面,可以选择Oracle的安装位置。这里有两个安装路径:

☐ 第1个安装路径是Oracle的基目录,主要用于存放一些配置文件内容。

☐ 第2个安装路径用于存储Oracle软件文件。

这里用户可以根据实际情况选择安装目录。

4) 检查安装环境。选择好安装路径后,单击【下一步】按钮,进入如图2.8所示的界面检查安装环境。



图2.7 配置安装路径



图2.8 先决条件检查

如果用户的计算机满足2.1节所述的Oracle安装条件,那么这部分检查是可以正常通过的。检查完毕后,如果出现图2.8中的“0个要求待验证。”字样,就说明Oracle 11g检查的先决条件是没有问题的。

5) 选择配置选项。单击【下一步】按钮,进入如图2.9所示的界面,选择要安装的内容,这里有3个选项:

☐ 【创建数据库】:安装软件时直接创建数据库,操作相对简单,属于默认安装。

☐ 【配置自动存储管理(ASM)】:更多用在集群环境下,可简化存储管理和提高性能。

☐ 【仅安装软件】:只安装软件,不创建数据库,以后若需要可以单独创建。

这里选择【仅安装软件】选项。

6) 安装软件。单击【下一步】按钮,进入如图2.10所示的界面。

7) 查看好安装信息后,可以单击【安装】按钮进入安装过程,大约20分钟安装完毕。安装结束后出现如图2.11所示的界面,表示已经完成Oracle 11g软件的安装。

3. 配置Oracle监听程序

在完成了Oracle 11g软件的安装后,如果要安装Oracle 11g的数据库实例,就需要先配置Oracle的监听程序。配置Oracle的监听程序分为准备创建监听程序、选择配置内容、添加监听程序、选择TCP/IP协议的端口号以及完成配置并验证5个步骤。

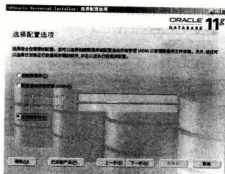


图2.9 选择配置选项



图2.10 安装概要



图2.11 安装结束

1) 准备创建监听程序。创建监听程序需使用Oracle中自带的配置工具Net Configuration Assistant, 配置工具的位置如图2.12所示。

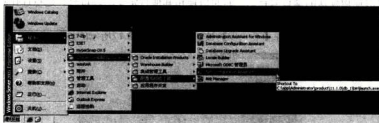


图2.12 准备创建监听程序

说明 这里也可以在【开始】|【运行】框中输入netca命令。

2) 选择配置选项。单击【Net Configuration Assistant】选项, 进入如图2.13所示的界面。其中列出了可以配置的内容, 这里选择【监听程序配置】选项。



3) 添加监听程序。单击【下一步】按钮, 进入如图2.14所示的界面。



图2.13 创建监听程序



图2.14 添加监听程序

由于这是第一次配置监听程序, 所以只能选择【添加】选项, 其他选项都不可选。单击【下一步】按钮, 进入如图2.15所示的界面。其中可以更改监听程序的名称, 默认是LISTENER, 这里使用默认名称即可。

4) 选择TCP/IP协议的端口号。单击【下一步】按钮, 进入如图2.16所示的界面, 在其中选择TCP/IP协议的端口号。



图2.15 监听程序名称



图2.16 TCP/IP端口号

这里可以使用默认的端口号1521, 也可以使用其他自定义的端口号, 这里选择默认端口号即可。

5) 完成配置并验证。单击【下一步】按钮, 进入如图2.17所示的界面。



图2.17 配置另一个监听程序

在图2.17中还可以继续配置监听程序, 如果只配置一个监听程序, 那么单击【下一步】按钮, 即可完成监听程序的配置。

要查看监听程序是否启动, 可以在【开始】|【运行】框中输入services.msc命令来查看监听是否启动, 如图2.18所示。

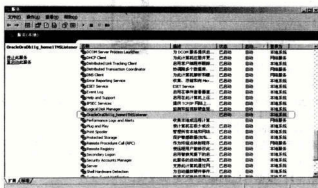


图2.18 查看监听程序是否启动

如果该服务的状态是“已启动”, 那么监听程序就配置成功了。

4. 安装数据库实例

安装好数据库软件并配置好监听程序后, 就可以创建数据库实例了。创建数据库实例分为准备创建数据库实例、选择创建类型、选择创建数据库的模板、填写数据库标识、选择数据库的配置项、设置账户口令、添加示例方案、选择初始化参数、完成安装9个步骤。

1) 准备创建数据库实例。数据库实例是用Oracle中自带的配置工具Database Configuration Assistant来创建的, 配置工具所在的位置如图2.19所示。

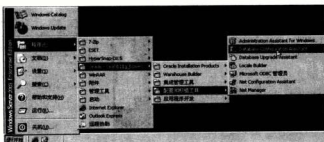


图2.19 数据库实例创建程序

2) 选择创建类型。单击【Database Configuration Assistant】选项, 进入如图2.20所示的界面, 选择要创建的类型。要创建数据库的实例, 这里选择【创建数据库】选项。

3) 选择创建数据库的模板。单击【下一步】按钮, 进入如图2.21所示的界面, 选择数据库的模板。

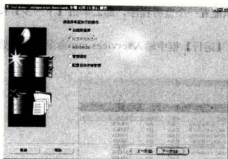


图2.20 创建类型

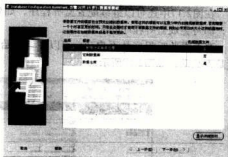


图2.21 数据库模板

在该界面中，有3个模板选项：

- ☐ 【一般用途或事务处理】：适应大多数用途的配置，既能适用于联机事务处理，也能适用于数据仓库。
- ☐ 【定制数据库】：根据自己的需要来定制模板。
- ☐ 【数据仓库】：针对大量数据库，适合做数据挖掘。

这里选择比较通用的【一般用途或事务处理】选项。

4) 填写数据库标识。在图2.21中，单击【下一步】按钮进入如图2.22所示的界面，填写数据库标识。

这里包括全局数据库名和SID标识，其中，全局数据库名主要针对分布式数据库系统来说，例如，你公司（公司名为xxx）在新疆有一台移动项目数据库，在河北也有一台，那么你就可以以orel.yd.xxx.xinj和orel.yd.xxx.heb来命名这两台数据库；SID就好像身份证一样，可以根据业务填写，但是要保证和本机的其他SID不重复。

5) 选择数据库的配置项。完成了数据库标识的填写后，在图2.22中单击【下一步】按钮，进入如图2.23所示的界面。

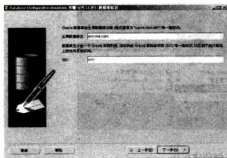


图2.22 填写系统标识符

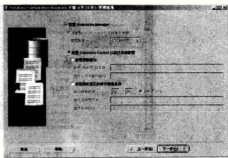


图2.23 数据库管理选项

这里可以直接使用默认选项【配置Enterprise Manager】即可。Oracle Enterprise Manager用于监控和管理Oracle软件基础架构以及应用程序和商务服务。

6) 设置账户口令。单击【下一步】按钮，进入密码管理选项，如图2.24所示。

在该界面中可为数据库设置密码,这里为了方便,可以使所有账户使用同一个口令即可,但是为了数据库的安全性,可以为不同的账户设置不同的密码。

7) 添加示例方案。添加好密码后,单击【下一步】按钮,进入如图2.25所示的界面,选择示例方案。



图2.24 密码管理

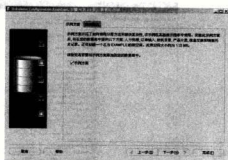


图2.25 示例方案选择

为了方便学习,可以选择该示例,该示例包括了Example的表空间。

8) 选择初始化参数。选择好示例方案后,单击【下一步】按钮,进入如图2.26所示的界面,选择初始化参数。此处是对字符集的设置,这里安装的字符集最好和客户端的一致,否则容易出现乱码。这里使用默认的字符集ZHS16GBK。

9) 完成安装。选择好初始化参数后,单击【下一步】按钮,进入后面的安装过程,以后的安装过程全部选择默认选项即可,这里就不一一给出图例。数据库安装成功后,出现如图2.27所示的界面。

至此,Oracle 11g的数据库实例安装成功。

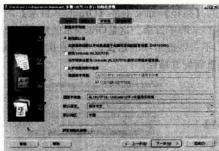


图2.26 参数初始化

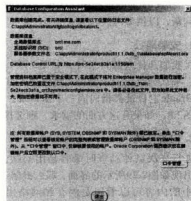


图2.27 安装完成

5. 查看数据库服务

安装成功后Oracle会以服务的形式存在,在【开始】!【运行】框中输入services.msc命令



查看服务列表。如图2.28所示，在图中用矩形框起来的的就是安装后出现的服务。用户可以在安装后自行检查，以确保数据库安装完整。

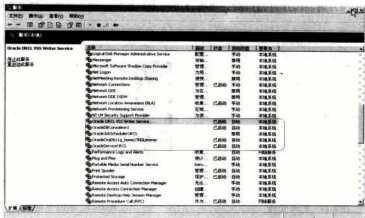


图2.28 服务列表

在这里我们看到了5个服务名称，其中常接触的有OracleDBConsole或cl Oracle企业管理器的服务、OracleServiceORCL数据库实例服务和OracleOraDb11g_home1TNSListener 远程访问监听服务。

2.2.3 安装中遇到的问题

在安装Oracle的过程中会出现一些问题，为了让用户少走弯路，这里列出了安装中容易出现的问题，以供参考。

1. 安装路径

在我们选择安装目录时，如图2.7所示，这里的【路径】如果不是【Oracle基目录】的子目录，那么会给出一个警告：建议把主目录设成基目录的子目录。这是Oracle希望用户创建的多个Oracle主目录都指向同一个基目录。如果我们不采纳提示项的建议，可以直接选择【是】进入图2.8；如果有这方面的需求，则选择【否】，重新配置路径。

2. 安装环境

进行先决条件检查时最容易出问题。如果在安装中出现如图2.29所示的界面，那么根据上面【检查】列表中的内容可以看出是内存和网络出现了问题。如果继续安装，则很可能会安装失败。这里是因为内存不足和没有TCP/IP协议的连接造成的。修改出问题的地方后单击【重试】按钮。如果没有警告才可以继续安装。



图2.29 先决条件检查界面

3. 未启动数据库监听服务

第一次创建数据库实例时，默认情况下监听程序是没有被创建的，此时安装数据库实例时会出现如图2.30所示的提示。

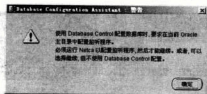


图2.30 未配置监听提示界面

这时只要启动监听服务即可，如果没有配置，可以参照前面配置Oracle监听程序部分的内容。

2.3 移除Oracle 11g

Oracle的卸载和普通软件的卸载有一定差异，很多人开始接触Oracle时经常因为不能很彻底地卸载软件而耽误时间，本节主要讲解如何完全移除Oracle。

Oracle 11g的卸载过程分为停止服务、卸载Oracle产品、删除注册表项、删除环境变量和重启计算机5个步骤，只要按照正确的卸载步骤卸载Oracle 11g，即可把它从你的计算机中完全移除。

2.3.1 停止服务列表的5个服务

在【开始】|【运行】框中输入services.msc命令，出现服务列表，分别选中Oracle的5个服务名称（如图2.31所示矩形框中的部分），右击，在弹出的快捷菜单中选择【停止】选项。

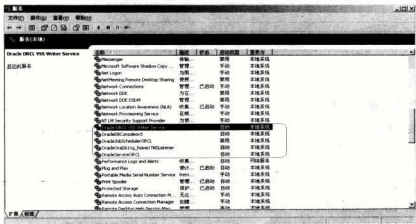


图2.31 停止Oracle服务

2.3.2 卸载Oracle产品

读书笔记

卸载Oracle产品时，要使用Oracle中自带的卸载工具来完成，卸载工具可以在【开始】菜单中的【程序】项中找到，具体路径如图2.32所示。

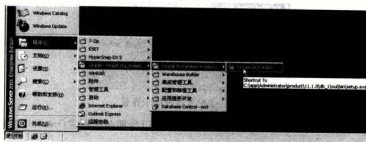


图2.32 选择卸载程序

单击【Universal Installer】选项，弹出如图2.33所示的“产品清单”对话框。在其中选择已经安装的Oracle产品，单击【删除】按钮即可删除选中的产品。



图2.33 选择卸载产品

2.3.3 删除注册表项

在安装数据库时会有很多配置都会写入到注册表中，要彻底删除Oracle 11g数据库，还要把写入到注册表中的内容也全部删除掉。需删除的注册表项如表2.3所示。

表2.3 注册表项

注册表项（如果有）
HKEY_LOCAL_MACHINE\SOFTWARE\ORACLE项
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services节点下的所有Oracle项
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Application F:\Oracle_VSSWriter\ORCL项

说明 在【开始】|【运行】框中输入regedit命令，可以直接进入注册表菜单窗口。



2.3.4 删除环境变量

如果在安装Oracle时为其设置了环境变量，那么在卸载时也需要把环境变量删除。具体的删除过程是：右击【我的电脑】，在弹出的快捷菜单中选择【属性】命令，弹出“系统属性”对话框，切换到【高级】选项卡，如图2.34所示。单击【环境变量】按钮，出现如图2.35所示的对话框。

在“系统变量”列表中找到ORACLE_HOME选项，删除即可。如果系统变量中的CLASSPATH和PATH变量中也存在Oracle设置，那么也一并删除即可。

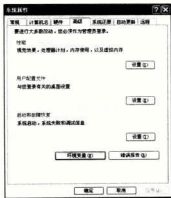


图2.34 系统属性



图2.35 环境变量

2.3.5 删除目录并重启计算机

在完成以上4个步骤后，Oracle 11g的删除工作已经接近尾声，为了更彻底地删除Oracle，还需要把安装时安装目录下的全部内容删除，删除后重新启动计算机即可成功卸载Oracle 11g。

任何软件在卸载过程中都可能出现程序中断，不能正常卸载的情况。Oracle 11g在卸载时，容易出现的问题就是Oracle的服务没有停止导致的卸载失败、注册表中的注册项没有完全删除、没有把所有Oracle安装目录下的文件完全删除。用户只要在卸载过程中按照上述步骤卸载，Oracle 11g即可在计算机中不留痕迹地消失。

2.4 小结

本章介绍了在Windows 2003环境下安装和卸载Oracle 11g的整个步骤。需要注意的是安装Oracle 11g的基本条件：硬件方面要求内存1GB以上，硬盘空间5GB以上，这两点很重要，如果达不到要求，大多会出现安装失败的问题。Oracle 11g的安装分为6个步骤，用户可以进行参考。至于卸载，需要5个步骤，这5个步骤是必需的，如果Oracle卸载不干净，很可能导致下次安装失败，所以这里建议用户严格执行卸载步骤。

2.5 习题

读书笔记

填空题

1. 常用的安装Oracle的操作系统有_____、_____。
2. Oracle 11g卸载需要_____、_____、_____、_____、_____5个步骤。
3. Oracle 11g建议的安装目录是_____。
4. 安装Oracle需要虚拟内存的容量是_____。
5. 安装Oracle需要物理内存的容量至少是_____。

第二篇

数据库基础篇

第3章 熟悉数据库

数据库从字面的意思上理解它就是一个存储数据库的仓库，就像用来存储药品的药库或用来存放粮食的粮库一样。要使用数据库就要先对数据库有一个了解。本章主要学习以下知识点：

- 数据库的历史和模型
- 数据库的三级模式和二级映像
- 关系型数据库的设计
- 实体-联系图的绘制

本章内容基本涵盖了关系型数据库的设计和实体-联系图的绘制以及数据库的基本知识。通过本章的学习，读者可以了解数据库的历史，掌握关系型数据库的设计方法以及实体-联系图的绘制。

3.1 什么是数据库

随着计算机和网络的普及，我们日常的工作和学习都离不开数据库了，如果没有数据库，我们就不能在网上订书，网上也就没有我们浏览的网站了。那么什么是数据库呢？本节就将讲解数据库的历史、数据库的模型、数据库的三级模式和二级映像、数据库相关的术语以及数据库设计的完整性。

3.1.1 了解数据管理的历史

任何事物的发展都有一段历史，数据管理的发展也不例外。在数据管理的发展过程中，主要经历了3个阶段，即人工管理阶段、文件系统阶段、数据库系统阶段。

1. 人工管理阶段

20世纪50年代中期以前是人工管理阶段，世界上第一台计算机（ENIAC）是1946年2月14日在美国诞生的。那时计算机还是一个陌生的词汇，对于计算机来说，存储信息的设备没有磁盘，只有磁带、卡片等存储设备；计算机中也没有操作系统更没有管理软件；处理数据的方式只有批处理方式。在人工管理阶段，主要负责数据管理的是人。人工管理数据具有以下4个特点：

1) 不能长期保存数据。在20世纪50年代中期之前，计算机一般在信息中心这样的研究机构里才能拥有，当时由于存储的设备有限，都是在做实验的时候暂存实验数据，做完实验就把



结果打在纸带或者磁带上带走，所以数据一般不需要长期保存。

2) 数据并不是由应用软件管理的而是由应用程序自己管理的。作为程序员在编写程序时既要设计程序逻辑结构又要设计物理结构以及数据的存取方式等，因此，对程序员编写代码的工作量和质量要求都很高。

3) 数据不能共享。在人工管理阶段，可以说数据是面向程序的，由于每一个应用程序都是独立的，即使要使用的相同数据已经在其他应用程序中存在，在应用程序之间也是不能共享的，这样也造成了大量的数据冗余。

4) 数据不具有独立性。应用程序中只要发生改变，数据的逻辑结构和物理结构就相应地发生变化。因而程序员都要做出相应的修改，给程序员的工作带来很多的负担。

根据人工管理阶段管理数据的特点，如果计算机中有3个应用程序，那么它们和各自数据集之间的对应关系就如图3.1所示。



图3.1 人工管理阶段管理数据

2. 文件系统阶段

20世纪50年代后期到60年代中期，计算机开始应用于数据管理方面。此时，计算机的存储设备也不再是磁带和卡片了，已经可以用磁盘直接存储了。此时，存储数据使用的是文件系统也称为管理软件，文件系统是操作系统中负责管理和存储文件信息的软件机构。文件系统由三部分组成：与文件管理有关的软件、被管理的文件以及实施文件管理所需的数据结构。文件系统阶段存储数据就是以文件的形式来存储，由操作系统统一处理。文件系统阶段也是数据库发展的初级阶段，使用文件系统存储数据具有以下四个特点：

1) 可以长期保存数据。有了大容量的磁盘作为存储设备，计算机开始被用来处理大量的数据并可以存储数据。

2) 有简单的数据管理功能。文件的逻辑结构与物理结构脱钩，程序和数据分离，使数据与程序有了一定的独立性，减少了程序员的工作量。

3) 共享数据能力差。由于每一个文件都是独立的，当需要用到相同的数据时，必须建立各自的文件，数据还是无法共享，也会造成大量的数据冗余。

4) 数据不具有独立性。在此阶段数据仍然不具有独立性，当数据的结构发生改变时，也必须修改应用程序，修改文件的结构定义；而应用程序的改变也将改变数据的结构。

根据文件系统阶段管理数据的特点，如果计算机中有3个应用程序，那么它们和各自文件之间的对应关系就如图3.2所示。



图3.2 文件系统阶段管理数据

从系统角度来看，文件系统是对文件存储存储空间进行组织和分配，负责文件的存储并说明：对存入的文件进行保护和检索的系统。具体地说，它负责为用户建立文件，存入、读出、修改、转储文件，控制文件的存取，当用户不再使用时撤销文件等。

3. 数据库系统阶段

从20世纪60年代后期开始就进入了数据库系统阶段。随着计算机的普及,同时计算机的外存设备磁盘的容量也得到了扩充,需要管理的数据也与日俱增,此时,在文件系统的基础上就开始有了数据库技术。最早的数据库管理系统是在1961年由通用电气公司开发的,此后,数据库的技术得到了飞速的发展。数据库存储数据不仅存储数据的本身同时也存储数据之间的联系。在数据库系统阶段管理数据具有以下4个特点:

1) 实现数据的共享。进入数据库系统阶段之后,数据终于可以得到共享。也就是说,数据在存入数据库后,想使用数据的用户可以同时来访问数据库使用数据,这样就减少了数据的冗余。

2) 数据具有了独立性。在数据库中数据库的逻辑结构和应用程序之间是相互独立的,同时数据库的物理结构变化也不会影响数据库的逻辑结构,这就给程序员减少了很多的工作量。

3) 数据实现集中控制。此时,数据不再由各自的应用程序管理,而是由数据库管理系统统一管理,这样可以保证数据的安全性和可维护性。

4) 故障恢复。在没有数据库管理系统之前,如果数据出现丢失或损坏的情况,那么数据是无法恢复的;但是有了数据库之后,数据库管理系统的备份恢复机制就可以帮助用户找回丢失和损坏的数据,可以最大限度地减少损失。

根据数据库系统阶段管理数据的特点,如果计算机中有3个应用程序,那么它们与数据库之间的对应关系就如图3.3所示。

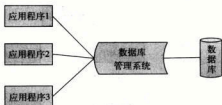


图3.3 数据库系统阶段管理数据

从文件系统发展到数据库系统,这在信息领域中具有里程碑的意义。在文件系统阶段,人们在信息处理中关注的中心问题是系统功能的设计;因此程序设计占主导地位;而在数据库方式下,数据开始占据了中心位置,数据的结构设计成为信息系统首要关心的问题,而应用程序则以既定的结构为基础进行设计。

3.1.2 数据库的模型

数据库的模型从数据库技术出现至今一共有3种比较通用的模型,目前使用最多的是关系型数据模型。下面就分别介绍这3种模型。

1. 层次结构模型

最早使用层次结构模型的是IBM公司的IMS (Information Management System),即数据库管理信息系统,这个系统也是被广泛应用的。层次结构模型类似于倒置的树型,一个父表可以有多个子表,但是每一个子表都对应着一个父表。图3.4所示就是一个学校人员的层次结构模型。



图3.4 学校人员层次结构模型

在图3.4中学校可以看做是父表，在学校下面有教职员工和学生两个子表，教职员工下面又有两个子表，学生下面也有两个子表，但是每一个子表都只有一个父表与之对应。

由于层次结构模型的结构很难改变，而且在表示表之间的关系时也有一些局限性，所以目前使用层次结构模型设计数据库是比较少的。

2. 网状结构模型

网状结构模型是对层次结构模型的改进，使用网状结构模型的代表是DBTG（Data Base Task Group），它是20世纪70年代数据系统语言研究会下属的数据库任务组提出的一个系统方案。网状结构模型打破了层次结构模型使用的限制，可以更全面地描述数据库中表之间的关系，可以一个父表没有子表，也可以一个子表有多个父表，还可以设置两个表之间的多种关系。图3.5所示是一个网状结构模型的例子。



图3.5 技术部门网状结构模型

图3.5中描述的是某软件公司技术部门的组成结构，这里程序员拥有员工宿舍和开发大厅两个父节点，这是层次结构模型不能描述的。尽管网状结构模型已经在层次结构模型的基础上有了一定的改进，但是对数据库的设计者要求很高，必须要非常熟悉数据库才能使用这种网状结构模型。

3. 关系结构模型

关系结构模型可以说是在层次结构模型和网状结构模型的基础上发展而来的，是目前使用最多的数据模型。最早给关系结构模型下定义的是E.F.Codd博士，他说：“关系数据结构保护数据，并且允许以一种可以预测并防止差错的方法操作数据。”关系结构模型实际上就是一个二维表，是由行和列组成的。例如，一个员工信息登记表就是一个二维表即关系模型，如表3.1所示。

表3.1 员工信息登记表

员工编号	姓 名	年 龄	性 别	所在部门	联系方式
1	张三	28	男	软件1部	13112345678
2	李四	30	男	软件2部	13312345678
3	赵五	29	男	软件1部	13212345678

在表3.1中,把一行数据称为一个元组,其中一共就有3个元组;把一列数据称为一个字段或属性,其中一共就有6个属性,即员工编号、姓名、年龄、性别、所在部门、联系方式。要想在设计数据库时使用关系结构模型,必须要做到规范化。关于规范化的问题将在3.2节中详细讲述。

目前大多数的数据库都是属于关系型数据库,这些数据库主要有IBM DB2、Oracle、SQL Server、MySQL、SyBase、Informix、Access、FoxPro等。

说明 关系数据库的产生是在1970年,IBM的研究员E.F.Codd博士在刊物*Communication of the ACM*上发表了一篇名为“A Relational Model of Data for Large Shared Data Banks”的论文,提出了关系模型的概念,奠定了关系模型的理论基础。

3.1.3 学习数据库的三级模式和二级映像

数据库的模式(Schema)是对现实世界的抽象,是对数据库中全体数据的逻辑结构和特征的描述。模式反映的是数据的结构及其联系,数据库系统在其内部具有三级模式和二级映像。三级模式分别为外模式、模式与内模式,二级映像则是外模式/模式映像和模式/内模式映像。

1. 三级模式

美国国家标准学会(American National Standard Institute, ANSI)的数据库管理系统研究小组于1978年提出了标准化的建议,将数据库结构分为3级:面向用户或应用程序员的用户级、面向建立和维护数据库人员的概念级、面向系统程序员的物理级。用户级对应外模式,概念级对应模式,物理级对应内模式。以下就分别讲解这3种模式。

(1) 模式

模式对应着概念级,它是由数据库设计者综合所有用户的数据,按照统一的观点构造的全局逻辑结构,是对数据库中全部数据的逻辑结构和特征的总体描述,是所有用户的公共数据视图。它是由数据库管理系统提供的数据库模式描述语言(Data Description Language, DDL)来描述、定义的,体现并反映了数据库系统的整体观。

(2) 外模式

外模式对应于用户级,它是某个或某几个用户所看到的数据库的数据视图,是与某一应用有关的数据逻辑的表示。外模式是从模式导出的一个子集,包含模式中允许特定用户使用的那部分数据。用户可以通过外模式描述语言来描述、定义对应于用户的数据记录(外模式),也可以利用数据操纵语言(Data Manipulation Language, DML)对这些数据记录进行操作。

说明 DML语言可以对数据进行4种操作,即创建(Create)、读取(Read)、更新(Update)和删除(Delete),也把它说成是对数据库执行CRUD操作。DDL语言是用于描述数据库中要存储的现实世界实体的语言,主要包括DROP、CREATE、ALTER、GRANT、REVOKE、TRUNCATE等操作,这在后面的章节中会详细介绍。



(3) 内模式

内模式对应于物理级，它是数据库中全体数据的内部表示或底层描述，是数据库最低一级的逻辑描述，它描述了数据在存储介质上存储方式的物理结构，对应着实际存储在外存储介质上的数据库。

2. 二级映像

数据库系统的三级模式是对数据的3个抽象级别，它把数据的具体组织留给DBMS管理，使用户能逻辑地、抽象地处理数据，而不必关心数据在计算机中的具体表示与存储。为了能够在内部实现这3个抽象层次的联系和转换，DBMS在这3个级别之间提供了两层映像：外模式/模式映像和模式/内模式映像。

外模式/模式映像使数据具有较高的逻辑独立性。它定义了该外模式与模式之间的对应关系。这些映像定义通常包含在各自外模式的描述中。当模式改变时，DBA要对相关的外模式/模式映像做相应的改变，以使外模式保持不变。应用程序是依据数据的外模式编写的，外模式不变应用程序就没必要修改。所以，外模式/模式映像功能保证了数据与程序的逻辑独立性。

模式/内模式映像使数据具有较高的物理独立性。它定义了数据库全局逻辑结构与存储结构之间的对应关系。该映像定义通常包含在模式描述中。当数据库的存储结束了，DBA要对模式/内模式映像做相应的改变，以使模式保持不变。模式不变，与模式没有直接联系的应用程序也不会改变。所以，模式/内模式映像功能保证了数据与程序的物理独立性。

3.1.4 数据库中的相关术语

在Oracle 11g数据库中每个数据库里面都包含很多对象，主要包括表、视图、存储过程、触发器以及约束。在本小节里只简单介绍一下每一个术语的含义，在后面的章节中会详细地讲解这些术语的使用。

1. 表

表，即在数据库中存放数据用的数据表。每一个数据库中都可以包含多张数据表，但是每一个数据表的名字都是不能重复的。表的一行代表一条记录，每一列都有一个列名，列名是唯一的，行与列的交叉点称为字段。例如，创建一个存放图书信息的表，表中信息包括图书编号、图书名称、图书作者、图书出版社、图书价格、备注，如表3.2所示。

表3.2 图书信息表

图书编号	图书名称	图书作者	图书出版社	图书价格	备 注
0001	Oracle 10g	刘一	× × 出版社	20	
0002	Oracle 11g	王一	× × 出版社	30	

在表3.2中，第一行是表中每一列的列名，后两行每一行都是表中的一条记录。

2. 视图

视图是数据库中的虚拟表。在视图中存放的是从数据库表中查询出来的记录，使用视图主要是为了方便信息查询，同时也能够缩短查询数据的时间。

3. 存储过程

存储过程是由SQL语句和控制流语句组成的语句块。存储过程存储在数据库内，可由应用程序通过存储过程的名称调用执行。

存储过程在开发软件时，可以把大量的数据操作放在服务器端的存储过程中，而只返回需要的数据，这样就减少了数据的传输量，速度也可以大大地提高。

4. 触发器

触发器是特殊的存储过程，也是由SQL语句和程序控制语句组成的。但是，触发器在数据库中是不需调用而自动执行的。例如，在触发器中可以定义在修改某张表记录后执行触发器中的内容。

5. 约束

约束是在数据库中保证数据库里表中数据完整性的手段。在Oracle 11g中使用的约束有主键约束、外键约束、唯一约束、检查约束、非空约束5个，其中主键约束和唯一约束都被认为是唯一约束，而外键约束被认为是参照约束。

(1) 主键 (Primary Key) 约束

主键约束在每个数据表中只能有一个，但是一个主键约束可以由多个列组成，通常把由多个列组成的主键又叫做复合主键或组合主键。主键约束可以保证主键列的数据没有重复值且值不为空，也可以说是唯一地标识表中的一条记录。

(2) 外键 (Foreign Key) 约束

外键约束之所以被认为是参照约束，是因为它主要用作把一个表中的数据和另一个表中的数据进行关联，表和表之间的关联是为了保证数据库中数据的完整性，使用外键保证数据的完整性，也叫参照完整性。在3.1.5小节还会详细地说明数据的完整性。下面就创建商品信息表和商品类型信息表，并建立两个表之间的外键约束，如表3.3所示。

表3.3 商品信息表和商品类型信息表

商品编号	名 称	类 型	价 格(元)	产 地
A01	面包	01	2.0	中国
A02	钢笔	02	10.0	中国
A03	电视	03	5020.0	中国

类型编号	类型名称
01	食品
02	文具
03	家电

在商品信息表中“商品编号”是主键，而在商品类型信息表中“类型编号”是主键，当把商品信息表中的“商品编号”与商品类型信息表中的“类型编号”设置为外键约束后，在商品信息表中的类型信息就可以用商品类型信息表中的类型编号代替。设置完外键约束后，商品信息表中类型字段值必须是在商品类型信息表中存在的，同时当在商品类型信息表中删除一个类型时，如果商品信息表已经使用过该类型，那么商品类型信息表中的数据就无法被删除。这样

就保证了数据库中数据的完整性。

（3）唯一（unique）约束

唯一约束和主键约束一样都是设置表中的列不能重复的约束，区别就是一个表中只能有一个主键约束，而却可以有多个唯一约束。通常情况下设置唯一约束的目的就是使非主键列没有重复值。唯一约束与主键约束的另一个区别是如果数据表中的某一列中有空值，那么就不能把这个列设置为主键列，而可以设置成唯一约束。例如，在商品信息表中把商品编号设置成了主键，但是还要保证商品的名称不重名时，就可以把商品名称设置为唯一约束。

（4）检查（check）约束

检查约束是用来指定表中列的值的取值范围的。例如，在员工信息表中员工年龄的列，如果要使员工年龄列的值为18~50，就可以使用检查约束进行设置，当输入的值不在有效范围内时，就会出现错误。这样就保证了数据库中数据的有效性。

（5）非空（not null）约束

非空约束是用来约束表中的列不允许为空。例如，在员工信息表中员工身份证号码列，要求员工必须输入时，可以使用非空约束来保证该列不能为空。

3.1.5 数据库设计的完整性

使用数据库约束就是保证数据库完整性的方法。数据库设计的完整性实际上就是为了保证数据的正确性。为了保证数据的正确性，在Oracle 11g中涉及的完整性主要有3个，即实体完整性、区域完整性、参照完整性。

1. 实体完整性

实体完整性要求表中的主键字段都不能为空或者重复的值。例如，在学校里每个学生的学号是唯一的，银行卡的卡号也是唯一的，每个人的身份证号码都是唯一的等。

2. 区域完整性

区域完整性是保证输入到数据库中的数据是在有效范围内的，可以使用3.1.4小节中讲的检查约束来设置。例如，输入邮箱的字段要求要有@，输入身份证号码要有15位或18位，输入年龄只能是数字，输入姓名不能有字母等。

3. 参照完整性

参照完整性可以保证数据库中相关联的表里面数据的正确性，使用3.1.4小节讲的外键约束就可以保证参照完整性。确保数据表的参照完整性，就可以避免误删和错加数据。例如，学生选课，如果学生已经选修了某门课程，但是管理员错误地把学生选的课程删除了，那么就会造成学生选修了课程但是无法上课，使用参照完整性设计数据表就会避免类似问题的发生。

3.2 范式——设计关系型数据库的准则

关系型数据库是目前流行和使用广泛的数据库，关系型数据库的设计标准就是数据库的范式，范式分别有第一范式、第二范式、第三范式。



3.2.1 第一范式——关系型数据库设计的第一步

目前,只要是使用关系型数据库来设计数据库,都能够满足数据库设计的第一范式。第一范式(1NF)就是数据库表中的字段都是单一属性的,不可再分。这个单一属性可以是数据库中任何一种基本数据类型,如整型、字符型、日期型等。只要是关系型数据库都会满足第一范式。例如,一个产品信息表(product),描述产品信息的字段有产品编号、产品名称、产品数量、产品价格、产品描述,如表3.4所示,那么这个产品信息表就满足第一范式的要求:每一个字段都是不可再分的单一属性。

表3.4 产品信息表(product)

字段名	数据类型
产品编号	整型
产品名称	字符型
产品数量	整型
产品价格	实型
产品描述	字符型

3.2.2 第二范式——关系型数据库设计的第二步

第二范式是在第一范式的基础上进一步对关系型数据库进行规范,官方给出第二范式的定义是要求在数据库表中不存在非关键字段对任一候选关键字段的部分函数依赖。意思就是说在第二范式中组合主键(AB)里面的A或者B与其他字段不能存在组合重复。为解决这个问题,通常的做法是不用组合主键,添加一个编号列,作为单一主键即可满足第二范式。如果不想添加编号列,就满足组合主键(AB)里面的A或者B与其他字段不能存在组合重复。例如,设计一个购物信息表,字段包括客户编号、产品名称、产品数量、产品类型、产品价格、客户类型。如果用客户编号和产品名称作为组合主键,那么在组合主键中产品名称和产品类型存在一定关系,是由产品名称决定产品的类型,所以不符合第二范式的要求,如果不按照第二范式的要求设计表,就会出现以下4个问题:

1. 数据冗余

同一个产品由 n 个顾客购买,“产品类型”就重复 $n-1$ 次;同一个顾客购买多件产品,那么就会多次记录顾客的个人信息。

2. 更新异常

若调整了某个产品的类型,数据表中所有行的“产品类型”值都要更新,否则会出现同一个产品不同类型的情况。

3. 插入异常

假设新进了一个产品,暂时还没有人购买。这样,由于没有人购买,产品的名称和类型也无法记录到数据库中。

4. 删除异常

假设一批顾客把已经购买完的商品退货,这些产品信息就从数据表中删除了。但是,与此同时,产品名称和产品类型等信息也被删除了。这样就导致了删除异常。

为了消除数据冗余、更新异常、插入异常和删除异常,可以把现有的一个表拆分成3张表:

第1张表是产品类型表,表中有产品类型、产品名称。

第2张表是客户信息表,表中有客户编号、客户类型。

第3张表是产品信息表,表中有产品名称、产品类型、产品价格、产品数量。



3.2.3 第三范式——关系型数据库设计的第三步

第三范式是在第二范式的基础上对数据库设计进行规范，第三范式的要求是数据表中不存在非关键字段对任一候选关键字段的传递函数依赖。所谓传递函数依赖，指的是如果存在A决定B、B决定C的决定关系，则C传递函数依赖于A。因此，满足第三范式的数据库表应该不存在依赖关系，假定员工信息表为employee（员工编号，姓名，年龄，所在部门，部门电话），使用员工编号作为员工信息的主键，那么就存在决定关系：员工编号就决定了姓名、年龄、所在部门、部门电话这些字段。从上面的关系可以看出，在表中有一个主键，数据表的设计符合第二范式的要求。但是它不符合第三范式的要求，因为存在决定关系：员工编号就决定了所在部门，所在部门又决定了所在部门的电话，那么就存在了传递函数依赖关系，即员工编号决定部门电话，那么也会出现不满足第二范式时的数据冗余和更新、插入、删除异常的情况。为了满足第三范式的要求，必须把员工信息表拆分成如下两个数据表：

员工表：员工编号、姓名、年龄、所在部门；

部门表：部门名称、部门电话。

除了上面的三种范式以外，还有一种范式经常使用，即鲍依斯-科得范式（BCNF）。它建立在第三范式的基础上，如果数据库表中不存在任何字段对任一候选关键字段的传递函数依赖，那么就符合BCNF范式。

3.3 绘制E-R图设计数据库

E-R（Entity-Relationship）图又叫实体-联系图，是描述现实世界的概念模型。构成E-R图的基本要素是实体、属性和联系。下面就来详细地讲解如何绘制E-R图。

3.3.1 绘制E-R图的基本要素

在E-R图中涉及的基本要素有实体、联系以及属性，下面就对这3个要素进行详细说明。

（1）实体（Entity）

实体是客观存在并可以相互区别的事物。实体既可以是人、物，也可以是抽象的概念。例如，一个学生、一个老师、一个产品都可以认为是实体。相同类型的实体可以构成一个实体集。例如，全体学生就是一个实体集。在E-R图中实体一般用矩形表示，矩形框内写明实体的名称。例如，写一个老师的实体，如图3.6所示。



图3.6 老师实体的表示

（2）属性（Attribute）

属性是实体所具有的某一特性，一个实体可由若干个属性来刻画。在E-R图中一般用椭圆形表示，并用无向边将其与相应的实体连接起来。例如，产品的名称、价格、类型等都是属性。例如，给图3.6的老师实体加上属性：姓名、年龄、所教专业、所属院系，如图3.7所示。

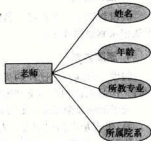


图3.7 老师实体属性的表示

（3）联系（Relationship）

联系，即在信息世界中反映实体内部或实体之间的联系。



实体内部的联系通常是指组成实体的各属性之间的联系；实体之间的联系通常是指不同实体集之间的联系。在E-R图中用菱形表示，菱形框内写明联系名，并用无向边分别与有关实体连接起来，同时在无向边旁标上联系的类型。实体之间存在着3种联系类型，分别是一对一、一对多、多对多，它们反映到E-R图中即为相应的联系类型，即1:1、1:n和m:n。

□ 一对一关系 (1:1)

一对一关系是指实体集A与实体集B，A中的每一个实体至多与B实体集中一个实体有联系，反之，在实体集B中的每个实体至多与实体集A中一个实体有联系。例如，给学生排座，“学生”实体和“座位”实体之间的关系，每一个学生最多可以分得一个座位，同时每一个座位最多只能有一个学生来坐。用图形表示如图3.8所示。



图3.8 一对一关系

□ 一对多关系 (1:n)

一对多关系是指实体集A与实体集B中至少有n (n>0) 个实体有联系，并且实体集B中每一个实体至多与实体集A中一个实体有联系。例如，“学生”实体和“班级”实体之间的关系，一个班级里面可以有若干个学生，而每一个学生都属于这个班级。用图形表示如图3.9所示。

□ 多对多关系 (m:n)

多对多关系是指实体集A中的每一个实体与实体集B中至少m (m>0) 个实体有联系，并且实体集B中的每一个实体与实体集A中的至少n (n>0) 个实体有联系。例如，顾客在商场购买商品，顾客与商品之间就是多对多的关系，每一个顾客都可以购买多种商品，而每一种商品又可以被多个顾客购买。用图形表示如图3.10所示。

其实，实体之间的这3种关系，不仅对两个实体有效，也可以表示多个实体之间的关系。



图3.9 一对多关系



图3.10 多对多关系

3.3.2 E-R图绘制实例

绘制一个网上购物系统的E-R图，在网上购物系统中简单分析出顾客、商品、商品类型、订单4个实体。下面分别绘制每个实体属性图并在最后绘制一个整体的E-R图。

1. 顾客实体属性图

顾客实体主要包括用户编号、姓名、年龄、性别、身份证号、联系方式、送货地址、银行卡卡号8个属性，实体属性图如图3.11所示。

2. 商品实体属性图

商品实体主要包括商品编号、商品名称、商品价格、商品数量、商品描述5个属性，实体属性图如图3.12所示。

3. 商品类型实体属性图

商品类型实体主要包括商品类型编号和商品类型两个属性，实体属性图如图3.13所示。

4. 订单实体属性图

订单实体主要包括订单编号、送货地址、顾客姓名、是否付款、联系方式、所购商品6个



属性，实体属性图如图3.14所示。

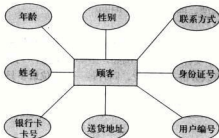


图3.11 顾客实体属性图

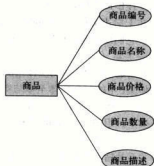


图3.12 商品实体属性图



图3.13 商品类型实体属性图

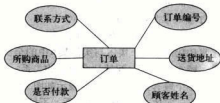


图3.14 订单实体属性图

5. 网上购物系统E-R图

在绘制整体的E-R图之前，先要了解一下网上购物系统的购物流程。首先由顾客选择要购买的商品，之后把购买商品列表生成一个订单，然后网站的售后人员会根据订单的地址送货，在这个网上购物系统里要求一个顾客每次只能生成一个订单。那么，这4个实体之间是什么关系呢？首先商品和顾客之间的关系是多对多的关系，多个商品可以被一个顾客购买，同时多个顾客也可以购买相同的商品；订单和商品之间的关系是一对多的关系，一个订单是由多个商品组成的，多个商品组成一个订单；顾客和订单之间的关系是一对一的关系，一个顾客可以生成一个订单，一个订单只能属于一个顾客；商品和商品类型之间的关系是一对一的关系，一个商品属于一种商品类型。网上购物系统的E-R图如图3.15所示。

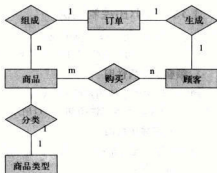


图3.15 网上购物系统E-R图



3.4 小结

通过本章的学习，用户可以了解数据库的历史，掌握关系型数据库设计的规则，即范式的使用，以及使用E-R图绘制数据库中表之间的联系。这些知识也是数据库的基础。此外，本章在介绍数据库历史和设计数据库的基础上，还对数据库中常用的术语，即表、视图、存储过程、触发器以及约束做了简单的介绍，方便读者学习本书后面的章节。

3.5 习题

简答题

1. 在数据管理的人工阶段，数据的存放主要用什么？
2. 简述数据库的三级模式。
3. 什么是E-R图？并绘制一个选课系统的E-R图。
4. 设计一张符合第三范式的数据库表。
5. Oracle 11g是什么类型的数据库？

第4章 SQL基础

学习每一个数据库不能缺少的就是如何访问它的语言，学习一个访问数据库的语言，就是和数据库沟通的第一步。SQL (Structured Query Language) 是每一个数据库都通用的语言，因此，学好Oracle 11g, SQL也是基础。使用SQL语言可以在数据库中创建表、检索数据、操作数据，并对权限进行控制。本章包括以下知识点：

- SQL语言概述以及分类

- SQL中4种类型的语言讲解 (DDL、DML、DQL、DCL)

本章内容基本涵盖了对SQL中4种类型语言的讲解。通过本章的学习，读者可以熟练地使用SQL语言对Oracle 11g数据库进行操作。

4.1 SQL——数据库沟通的语言标准

SQL (结构化查询语言) 的主要功能就是在各种数据库间建立联系，进行沟通。本节将学习什么是SQL及其分类。

4.1.1 什么是SQL

美国国家标准学会 (American National Standards Institute) 成立于1918年，SQL就是美国国家标准学会确定的。SQL主要用于存取数据以及查询、更新和管理关系数据库系统。SQL语言自IBM公司在1981年推出以后，由于其语法结构简洁又简单易学，在数据库中得到了广泛的应用。现行的所有数据库几乎都支持SQL语言，如Sybase、SQL Server、Oracle以及Visual FoxPro等数据库。但是这些数据库各自又对SQL语言进行了改进，如SQL Server数据库使用Transact-SQL语言。我们学习的Oracle数据库使用的是PL/SQL语言。

SQL语言本身可以分成4类，即：

- 定义要在数据库存储哪些信息的数据定义语言 (DDL)；

- 对数据库中的表进行操作的数据操纵语言 (DML)；

- 对数据库中的表进行检索的数据查询语言 (DQL)；

- 对数据库中对象进行权限管理的数据控制语言 (DCL)。

4.1.2 了解SQL的种类

从SQL语言的种类来看，由数据库表的创建到给数据库中的对象进行权限管理全部都可以使用SQL语言，下面就按照SQL语言的使用顺序说明每种SQL语言的作用。

1. 数据定义语言 (DDL)

数据定义语言 (Data Definition Language, DDL) 正如它字面上的意思，是定义数据库中数据要如何存储的。DDL语言包括对数据库中对象的创建、修改、删除的操作，这些对象主要有数据库、数据表、视图、索引等。



2. 数据操纵语言 (DML)

数据操纵语言 (Data Manipulation Language, DML) 也像它字面上的意思, 是对数据库表进行操作的。这些操作主要包括对数据库表中的数据进行增加、删除、修改的操作, 并且在操作时一次可以把表中数据按条件进行多条或全部的处理, 为数据库的使用提供方便。

3. 数据查询语言 (DQL)

数据查询语言 (Data Query Language, DQL) 是对数据库表中的数据进行查询的, 查询时既可以查询一个表也可以进行多表的查询, 并且可以按不同的条件来检索数据, 给数据库的查询统计工作带来了更多的便利。

4. 数据控制语言 (DCL)

数据控制语言 (Data Control Language, DCL) 是对数据库中的对象权限进行权限设置和取消等操作, 但是只有数据库的系统管理员才有权力去执行对数据库对象权限的操作。使用 DCL 可以为数据库中不同的用户设置不同的权限, 这样也能够提高数据库的安全性。

4.2 Oracle 11g中支持的数据类型

数据类型是在向数据表中存储数据前必须设定好的, 就像如果要使用记事本查看文件内容, 那么文件就要是文本的, 不能有图片, 否则图片是查看不了的, 因为记事本中只能查看文本文件。数据类型如此重要, 本节中将首先介绍如何查看 Oracle 11g 中的全部数据类型, 然后讲解其中常用的数据类型。

4.2.1 查看 Oracle 11g 中的数据类型

要使用数据库来存储数据, 首先就要知道这个数据库都能存储什么类型的数据。首先在 Oracle 11g 的企业管理器中查看数据类型。查看的步骤如下所示。

1. 打开企业管理器

在安装完 Oracle 11g 后企业管理器就已经安装完成了, 在【开始】菜单中的【程序】下找到安装的目录, 单击其中的【Database Control-orcl】项, 如图 4.1 所示。企业管理器的登录页面如图 4.2 所示。

注意 这里【Database Control-orcl】项中的 orcl 指的是安装数据库时的数据库实例名。

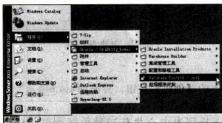


图 4.1 打开企业管理器



图 4.2 企业管理器登录页面



在图4.2所示的企业管理器登录页面中输入用户名和口令，并把连接身份设置为SYSDBA，单击【登录】按钮，登录到企业管理器的主界面，如图4.3所示。

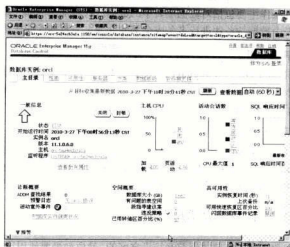


图4.3 企业管理器主界面

2. 进入创建数据表界面

在图4.3所示的企业管理器主界面中，单击【方案】链接，进入方案界面，如图4.4所示。在“数据库对象”一栏里，单击【表】链接，进入创建表界面，如图4.5所示。



图4.4 企业管理器中方案界面



图4.5 创建表界面

3. 查看数据类型

为了查看表中的数据类型，单击创建表界面中的【创建】按钮，界面如图4.6所示。在界面中选中【标准】单选按钮，单击【继续】按钮，界面如图4.7所示。

在图4.7所示的界面中，在“数据类型”下拉列表中一共有23种数据类型，分别是 VARCHAR2、NUMBER、DATE、CHAR、FLOAT、INTEGER、NCHAR、NVARCHAR2、

LONG、LONG RAW、RAW、ROWID、UROWID、BLOB、CLOB、NCLOB、BFILE、TIMESTAMP、INTERVAL YEAR、INTERVAL DAY、BINARY_DOUBLE、BINARY_FLOAT、XML TYPE类型。



图4.6 创建表第一步



图4.7 创建表界面

4.2.2 常用数据类型

在Oracle 11g中提供的数据类型有23种，下面介绍常用的数据类型，并把数据类型分为字符型、数字型、日期类型和其他数据类型4类进行讲解。

1. 字符型

字符型在Oracle 11g中有varchar2、char、nchar、nvarchar2和long五种，它们在数据库中是以ASCII码的格式存储的。下面用一个表格来讲解每种数据类型的作用，如表4.1所示。

表4.1 字符型

数据类型	取值范围(字节)	说明
varchar2	0~4000	可变长度的字符串
nvarchar2	0~1000	用来存储Unicode字符集的变长字符型数据
char	0~2000	用于描述定长的字符型数据
nchar	0~1000	用来存储Unicode字符集的定长字符型数据
long	0~2GB	用来存储变长的字符串

说明 在Oracle 11g中long类型很少使用，最常使用的字符数据类型就是varchar2。

2. 数字型

数字型在Oracle 11g中常用的有number和float类型两种，可以用它们来表示整数和小数。具体取值范围如表4.2所示。



表4.2 数字型

数据类型	取值范围	说 明
number (p,s)	p最大精度是38位（十进制）	p代表的是精度，s代表的是保留的小数位数；可以用来存储定长的整数和小数
float	用来存储126位数据（二进制）	存储的精度是按二进制计算的，精度范围为二进制的1~126，在转化为十进制时需要乘以0.30103

3. 日期类型

日期类型在Oracle 11g 中常用的有date和timestamp两种类型，可以用它们来存放日期和时间。详细说明如表4.3所示。

表4.3 日期类型

数据类型	说 明
date	用来存储日期和时间，范围在公元前4712年1月1日到公元9999年12月31日
timestamp	用来存储日期和时间，与date类型的区别就是在显示日期和时间时更精确，date类型的时间精确到秒，而timestamp的数据类型可以精确到小数秒。此外，使用timestamp存放日期和时间还能够显示当前是上午还是下午

4. 其他数据类型

除了上面讲过的字符型、数字型、日期类型之外，在Oracle 11g中还有存放大数据的数据类型以及存放二进制文件的数据类型。表4.4所示是对这些数据类型的详细说明。

表4.4 其他数据类型

数据类型	取值范围（字节）	说 明
blob	最多可以存放4GB	存储二进制数据
clob	最多可以存放4GB	存储字符串数据
bfile	大小与操作系统有关	用来把非结构化的二进制数据存储在数据库以外的操作系统文件中

4.3 数据定义语言（DDL）

DDL主要包括数据库对象的创建（create）、删除（drop）和修改（alter）的操作。本节中将以前数据表为对象讲解创建、删除、修改的DDL语言，对于其他对象的操作将在后面的章节中详细介绍。

4.3.1 使用Create语句创建表

在DDL语言中第一次使用数据库要用到的就是创建表，创建表使用create table语句完成。具体语法如下：

```
CREATE TABLE table_name
(
```




```
column_name datatype [null|not null],
column_name datatype [null|not null],
...
[constraint]
)
```

【语法说明】

- table_name: 在数据库中创建的数据表的名称, 在一个数据库中数据表名是不能重复的。
- column_name: 表中的列名, 列名在一个表中也是不能重复的。
- datatype: 该列存放数据的数据类型。
- [null|not null]: 允许该列为空或者不允许该列为空, 在创建表时默认为不允许该列为空。
- [constraint]: 为表中的列设置约束, 约束主要包括主键约束、外键约束、检查约束等, 在第3章中有简要的介绍, 在4.4节中将详细讲解如何创建这些约束。

下面利用上面的语句创建一个商品信息表。首先打开sqlplus并以scott用户的身份登录, 商品信息表中有商品编号、商品名称、商品价格、商品数量、商品类型、商品描述、产地7个字段。这些字段的数据类型定义如表4.5所示。

表4.5 商品信息表 (productinfo)

字段名	中文释义	数据类型
ProductId	商品编号	varchar2(10)
ProductName	商品名称	varchar2(20)
ProductPrice	商品价格	number(8,2)
Quantity	商品数量	number(10)
Category	商品类型	varchar2(10)
Desperation	商品描述	varchar2(1000)
Origin	产地	varchar2(10)

按照设置好的字段名和数据类型在sqlplus中创建表productinfo, 结果如图4.8所示。



图4.8 产品信息表

4.3.2 使用Alter语句修改表

如果要对已经创建好的表进行修改, 那么就需要使用alter table语句来修改。修改表的基



本语法如下：

```
ALTER TABLE table_name
ADD column_name | MODIFY column_name | DROP COLUMN column_name;
```

【语法说明】

□ ADD：用于向表中添加列。

□ MODIFY：用来修改表中已经存在的列的信息。

□ DROP COLUMN：删除表中的列，在删除表中的列时经常要加上CASCADE CONSTRAINTS，是要把与该列有关的约束也一并删除掉。

下面就利用上面的知识分别完成下面几个例子。

【示例1】修改productinfo商品信息表，向该表中增加一列

向表中添加列使用的是ADD子句，向表中增加一列备注remark信息，字段类型是varchar2。修改操作如图4.9所示。



图4.9 添加列remark

这样，就完成了在productinfo表中增加一列的修改操作。

注意 这里在登录SQL*Plus时使用的用户是sys，所以在修改表时需要在表的前面加上scott，如果使用scott用户登录就不用再添加scott了。

【示例2】修改productinfo商品信息表，修改列的字段类型

修改字段类型需要使用的是MODIFY子句，修改productinfo中刚添加的remark列的字段类型为number类型。修改操作如图4.10所示。

这样就完成了把remark字段的类型修改成number类型的操作了。

【示例3】修改productinfo商品信息表，删除表中的字段

删除表中的字段要使用DROP子句，下面就删除productinfo表中的remark字段。删除操作如图4.11所示。



图4.10 修改remark列



图4.11 删除remark列



这样就把remark列从表productinfo中删除了。

上面已经练习了如何修改表中的字段。实际上，对表的修改操作并不是一次只能修改一个字段，也可以同时完成对多个字段的修改。下面就运用上面的语句，完成一个综合的实例。

【示例4】修改productinfo商品信息表的多个字段

修改productinfo表中的ProductName字段，把字段的长度修改成25，并添加一个字段remark。具体操作如图4.12所示。

这样就同时修改了表中两个字段，除了修改和添加字段外，还可以删除多余的字段。这里就不一一演示了，请读者自行练习。



图4.12 修改多个字段

在对表的修改操作中还可以修改表中约束的信息，对于约束的修改将在4.4节中详细讲述。

4.3.3 使用Drop语句删除表

在使用数据库中的表时经常需要删除一些不需要的表，删除表需要使用DROP TABLE语句来完成。具体语句如下：

```
DROP TABLE table_name;
```

删除表的语句是非常简单的，只需要指定要删除的表名，即可删除该表。

下面就利用上面删除表的语句完成删除的操作。如果要删除上面创建的productinfo表，只需要下面的语句即可完成：

```
DROP TABLE productinfo;
```

以上就是对数据定义语言（DDL）基本操作的讲解。

4.4 约束的使用

约束是保证数据库表中数据的完整性和一致性的手段，在本书的第3章中已经介绍过Oracle 11g中的5个约束，即主键约束、外键约束、唯一约束、检查约束、非空约束。在本节中将一一讲解每一个约束是如何创建、修改、删除的。

4.4.1 主键约束

主键约束在每一个数据表中只有一个，但是一个主键约束可以由数据表中多个列组成。下面就学习主键约束的使用。

(1) 使用主键约束创建商品类型信息表
在创建表时就创建主键约束，只需要使用primary key（字段名）即可完成。商品类型信息表主要用来存放商品类型信息，包括商品类型编号和商品类型名称，并把商品类型编号设置成主键。这两个字段的数据库类型定义如表4.6所示。

表4.6 商品类型信息表（categoryinfo）

字段名	中文释义	数据类型
CategoryId	商品类型编号	varchar2(10)
CategoryName	商品类型名称	varchar2(30)

按照设置好的字段名和数据类型，在sqlplus中创建表categoryinfo，并把商品类型编号(CategoryId)设置成主键，结果如图4.13所示。



图4.13 商品类型信息表

(2) 使用ALTER TABLE语句为表添加主键约束

在创建表时如果没有创建主键约束，可以在修改表时为表添加主键约束。添加主键约束的语法如下：

```
ALTER TABLE table_name
ADD CONSTRAINTS constraint_name PRIMARY KEY (column_name);
```

【语法说明】

- constraint_name: 约束的名称。
- column_name: 主键约束指定数据表中的列名。

下面就假设在创建商品类型信息表categoryinfo时没有添加主键约束，在修改表时为表添加主键约束，结果如图4.14所示。



图4.14 添加主键约束

这样就创建了主键约束，主键约束的名称是pk_category，创建的主键列是categoryid。

(3) 移除主键约束

如果需要移除表中现有的主键约束，可以使用如下所示的语句完成：

```
ALTER TABLE table_name
DROP CONSTRAINT constraint_name;
```

【语法说明】

constraint_name: 要移除的约束名称，这个名称可以在表中任意约束的名称。

下面利用上面的语句，移除商品类型信息表中的主键约束pk_category。具体语句如下：

```
ALTER TABLE categoryinfo
DROP CONSTRAINT pk_category;
```

这样，就完成了主键约束pk_category的移除操作。

4.4.2 外键约束

外键约束可以保证使用外键约束的数据库列与所引用的主键约束的数据列一致，外键约束



在一个数据表中可以有多个。下面就来讲述外键约束的使用。

(1) 使用外键约束创建商品信息表

外键约束是建立在两张表中的约束，需要在创建表的语句后面加上如下语句：

```
CONSTRAINT constraint_name FOREIGN KEY (column_name)
REFERENCE table_name (column_name )
ON DELETE CASCADE;
```

【语法说明】

- constraint_name: 创建的外键约束名字。
- FOREIGN KEY (column_name): 指定外键约束的列名。
- REFERENCE: 要引用的表名 (列名)。
- ON DELETE CASCADE: 设置级联删除，当主键的字段被删除时，外键所对应的字段也被同时删除。

下面就利用上面的语句完成外键约束的创建。在创建商品信息表时有一个字段叫做商品类型，如果要让商品信息表中类型全部来源于商品类型信息表中的类型，就可以把商品信息表中商品类型设置成外键约束。这里为了不删除原有的表，新建一张商品信息表 (productinfo1)。具体操作语句如图4.15所示。

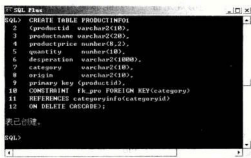


图4.15 创建外键约束

这样，在商品信息表productinfo1的category字段中的数据就必须是商品类型信息表中已经存在的商品类型，也就保证了商品类型信息的一致性。

(2) 在修改数据库表时添加外键约束

在已经存在的数据库表中也是可以添加外键约束的。添加外键约束是在ALTER TABLE语句后面加上如下语句：

```
ADD CONSTRAINT constraint_name FOREIGN KEY (column_name)
REFERENCE table_name (column_name )
ON DELETE CASCADE;
```

上面的语句和创建外键约束的语句一致，这里就不再详细讲解了。

下面就使用上面的语句，完成对之前已经创建好的商品信息表 (productinfo) 添加外键约束的操作。具体操作语句如图4.16所示。



图4.16 添加外键约束

这样，就为表productinfo中的category字段添加了外键约束。

(3) 移除外键约束

移除外键约束与移除主键约束的语法一致，这里以删除productinfo1中的外键约束fk_pro为例删除外键约束。具体删除语句如下：

```
ALTER TABLE productinfo1
DROP CONSTRAINT fk_pro;
```

这样，就移除了表productinfo1中的外键约束fk_pro。

4.4.3 CHECK约束

CHECK约束是检查约束，能够规定每一个列能够输入的值，以保证数据的正确性。下面就详细讲解CHECK约束的使用。

(1) 创建表时添加CHECK约束

创建CHECK约束可以设置在“性别”列中只能输入男或者女，在“年龄”列中只能输入18~30岁的年龄。创建CHECK约束的语句是在创建表的语句后面加上如下语句完成的：

```
CONSTRAINT constraint_name CHECK(condition);
```

其中，condition是检查约束的条件，检查约束的条件要建立在具体的字段中。例如，给字段Age设置为18~30岁，就可以写成age>=18 and age<=30。

下面创建一个顾客信息表，顾客信息表主要包括顾客编号、顾客姓名、顾客年龄、顾客性别、顾客电话、顾客住址6个字段信息。顾客信息表的详细信息如表4.7所示。

表4.7 顾客信息表（custominfo）

字段名	中文释义	数据类型
CustomId	顾客编号	varchar2(10)
Name	顾客姓名	varchar2(10)
Age	顾客年龄	number(2)
Gender	顾客性别	varchar2(2)
Tel	顾客电话	varchar2(11)
Address	顾客住址	varchar2(100)

根据表4.7所示的顾客信息表的信息，创建顾客信息表，并设置Age列的取值范围是18~50岁。具体操作如图4.17所示。



图4.17 创建Age列的检查约束

这样就完成了年龄字段中检查约束的创建，在年龄的字段中只能输入18~50的数字，输入其他的数字就会出现错误。

(2) 在修改数据表时添加CHECK约束

在修改数据表时添加检查约束的方法也比较简单，在ALTER TABLE语句的后面添加如下语句即可：

```
ADD CONSTRAINT constraint_name CHECK(condition);
```

下面利用上面语句为顾客信息表添加性别的检查约束，要求性别列只能输入“男”或者“女”。具体操作如图4.18所示。



图4.18 添加Gender列的检查约束

这样就完成了Gender性别列检查约束的添加，也可以同时添加多个检查约束。

(3) 移除CHECK约束

移除CHECK约束也与移除其他约束一样，只要知道CHECK约束的名字，就可以移除CHECK约束，下面就移除在顾客信息表中Gender列的检查约束chk_gender。具体语句如下：

```
ALTER TABLE custominfo  
DROP CONSTRAINT chk_gender;
```

这样，就可以移除CHECK约束chk_gender。

4.4.4 UNIQUE约束

UNIQUE约束称为唯一约束，可以设置在表中输入的字段值都是唯一的，这个约束和之前学习的主键约束非常相似。不同的就是唯一约束在一个表中可以有多个，而主键约束在一个表中只能有一个。下面就详细讲述UNIQUE约束的使用。

(1) 在创建表时添加UNIQUE约束

在创建表时可以为表中的字段直接添加UNIQUE约束，具体的创建方法是在创建表的语句后面加上下面的语句：

```
CONSTRAINT constraint_name UNIQUE(column_name);
```

下面就创建一个订单信息表，订单信息表中主要包括订单编号、顾客编号、商品编号、订单日期、订货数量、发货日期，订单信息表的详细信息如表4.8所示。

表4.8 订单信息表（orderinfo）

字段名	中文释义	数据类型
OrderId	订单编号	varchar2(10)
CustomId	顾客编号	varchar2(10)
ProductId	商品编号	varchar2(10)
OrderDate	订单日期	varchar2(10)
OrderQuantity	订货数量	number(10)
SendDate	发货日期	varchar2(10)

根据表4.8所示订单信息，创建订单信息表，并把订单编号设置成UNIQUE约束。具体操作如图4.19所示。

这样，就为订单信息表中的订单编号设置了唯一约束，订单编号在订单信息表中的信息就不可以重复了。

(2) 在修改表时添加UNIQUE约束

修改表时添加UNIQUE约束也是在ALTER TABLE语句后面加上如下语句完成的：

```
ADD CONSTRAINT constraint_name UNIQUE (column_name);
```

下面就对订单信息表中的顾客编号加入UNIQUE约束，具体操作如图4.20所示。

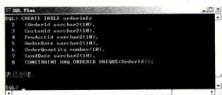


图4.19 添加唯一约束



图4.20 为顾客编号加入UNIQUE约束

这样，就说明了在一个表中可以添加多个UNIQUE约束。

(3) 移除UNIQUE约束

移除UNIQUE约束的方法也和移除其他约束一样，这里移除为订单信息表添加的顾客编号的UNIQUE约束。移除UNIQUE约束的语句如下：

```
ALTER TABLE orderinfo
DROP CONSTRAINT unq_customid;
```

这样，就可以移除顾客编号的UNIQUE约束了。

4.4.5 NOT NULL约束

NOT NULL约束就是非空约束，经常会在创建表时添加非空约束以确保字段必须要输入值。

该约束和之前的约束不同,是直接在创建列时设置字段的非空约束。下面就详细讲解NOT NULL约束的使用。

(1) 创建NOT NULL约束

创建NOT NULL约束的语法在创建表时就已经解释过了,这里创建一个商品管理员信息表,主要包括管理员编号、管理员注册名、管理员密码、管理员真实姓名、管理员联系方式。详细信息如表4.9所示。

表4.9 管理员信息表 (Managerinfo)

字段名	中文释义	数据类型
ManagerId	管理员编号	varchar2(10)
LoginName	管理员注册名	varchar2(10)
Password	管理员密码	varchar2(10)
Name	管理员真实姓名	varchar2(10)
Tel	管理员联系方式	varchar2(11)

根据表4.9所示的信息,创建管理员信息表并且把LoginName和Password两个字段设置为非空约束。具体操作如图4.21所示。

这样,就完成了LoginName和Password两个字段的非空约束的设置,和非空约束对应的还可以设置该字段为空,也就是NULL。

(2) 修改表时设置NOT NULL约束

在修改表时设置NOT NULL约束,也不需要再使用ADD关键字来添加约束,只要使用MODIFY关键字就可以设置表中字段的NOT NULL约束。具体语法如下:

```
ALTER TABLE table_name
MODIFY column NOT NULL;
```

下面就利用上面的语句完成设置管理员信息表真实姓名列不为NOT NULL的操作,具体操作如图4.22所示。



图4.21 设置非空约束



图4.22 为Name列设置NOT NULL 约束

对于非空约束不需要删除,如果要取消某个列非空的约束,直接使用MODIFY语句把该列的非空约束写成NULL即可。

4.5 数据操纵语言 (DML) 和数据查询语言 (DQL)

DML也就是用来操纵数据库中数据所使用的语言,对数据库中的数据操纵无非就是对数据进行增加、删除、修改、查询的操作。对于数据的查询也称为数据查询语言。本节将详细讲



解数据操纵语言的使用。

4.5.1 添加数据就用INSERT

在创建好数据表之后，添加数据是首先要做的工作。在给表中添加数据时要与表中字段类型相匹配，也就是说，如果表中的字段是日期类型，那么在向该字段中添加数据时也要添加日期类型的数据。向表中添加数据的一般语法如下：

```
INSERT INTO table_name(column_name1, column_name2,...) VALUES(data1,data2...);
```

【语法说明】

- column_name1：指定表中要添加数据的列名，可以是1个到多个。
- data1：要填入指定列的数据值，这里要求添加值的数目要与列名的数量一致。

实际上，向数据表中添加数据不仅可以使使用上面的语句来向表中添加值，也可以根据其他数据表中的数据来添加数据。下面就分别讲述这两种向表中添加数据的方法。

(1) 直接添加数据

直接添加数据时使用上面的语句就可以完成。下面就完成向管理员信息表中添加数据的操作。具体操作如图4.23所示。



图4.23 增加数据

这样，在管理员信息表managerinfo中就增加了一条数据。

(2) 通过其他数据表向表中添加数据

如果在数据库中需要新建一个数据表，但是这个表中的数据又与其他表中的数据有些相似，那么就可以直接把其他表中的数据添加到新建的数据表中，这样就能减少添加数据的工作量。具体语法如下：

```
INSERT INTO table_name1 (column_name1, column_name2,...)
select column_name1, column_name2... FROM table_name2;
```

【语法说明】

- table_name1：目标表的名称，也就是要插入数据的表名。
- table_name2：数据的来源表。

注意

在使用来源表向目标表中插入数据时，一定要确保两个表的列的个数和列的数据类型都一致，否则会出现错误。

【示例5】下面就利用上面的语句完成从管理员信息表中把数据添加到新建的账号信息表中

首先要新建一个账号信息表。账号信息表主要就是用来存放所有管理员的登录账号的，信息表中只有用户名和密码两个字段。详细信息如表4.10所示。



表4.10 账号信息表 (LoginInfo)

字段名	中文释义	数据类型
LoginName	管理员注册名	varchar2(10)
LoginPassword	管理员密码	varchar2(10)

根据表4.10所示的账号信息表的详细信息创建账号信息表，然后把管理员信息表中的注册名和密码添加到账号信息表中。具体操作如图4.24所示。现在表logininfo中也就含有了一条数据，并不是说向表中只能添加一条数据，而是因为表managerinfo中只包含了一条数据。

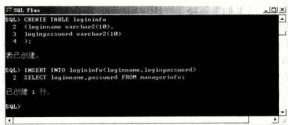


图4.24 从其他表中添加数据

上面介绍的这种添加数据的方式的前提是目标数据表已经存在，也就是logininfo这个表是先创建好的；如果想不创建表就直接通过源数据表在添加数据的同时创建表也是可以实现的。具体语法如下：

```
CREATE TABLE table_name AS SELECT column_name1, column_name2,...FROM source_table;
```

【语法说明】

□ table_name：要新创建的目标表的名称。

□ source_table：创建目标表时数据的来源表。这里可以指定查询表的字段，也可以用“*”代表查询表中的全部字段。

利用上面的语句创建一个表login，数据的来源表仍然选择managerinfo。具体的操作如图4.25所示。

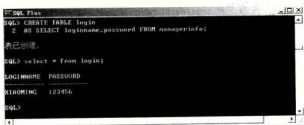


图4.25 直接创建带数据的表

这样就创建了login表，并且可以使用SELECT语句查看到在login表已经存在了一条记录。



4.5.2 修改数据就用UPDATE

读书笔记

具体语法如下：

```
UPDATE table_name SET column_name1=data1,column_name1=data2,...[WHERE condition];
```

【语法说明】

- column_name1：要修改数据列的字段名，可以是一个或多个。
- data1：要赋给字段的新值，这个值的数据类型要与数据表中字段的数据类型一致。
- WHERE：条件，这里如果省略了WHERE语句，那么就意味着要修改表中该字段的所有值，如果加上WHERE语句，那么就可以有选择地修改数据表中的某个字段。下面就利用UPDATE语句完成修改表中字段的全部值和某一个字段的值的操作。

(1) 修改表中指定字段的全部值

修改表中的全部值就是使用不带WHERE子句的语句完成。下面就修改新创建的表login中loginname注册名字段，把注册名都改写成“test”。具体操作如图4.26所示。这样就把表中的所有LoginName全部更改为test了。



图4.26 修改表中指定字段的全部值

(2) 根据条件修改表中指定字段的值

根据条件修改表中的数据可使用WHERE子句来完成。下面就将表login中用户名是“XIAOMING”的密码更改为“654321”。具体操作如图4.27所示。



图4.27 根据条件修改表

这样，就把用户名是“XIAOMING”的密码修改成了“654321”。这里，需要注意的是在查询时要区分大小写。



4.5.3 删除数据就用DELETE

经常要删除数据表中一些没有用的数据，删除数据要使用DELETE关键字来完成。使用它可以根据条件删除指定的数据，也可以删除表中的全部数据。一般的语法如下：

```
DELETE FROM table_name [WHERE condition];
```

其中，[WHERE condition]子句是可以省略的，如果省略了[WHERE condition]子句，就意味着删除数据表中全部的数据，如果加上了[WHERE condition]子句就可以根据条件删除表中的数据。这里，删除数据都是指删除数据表中一条记录并不是删除表中某个字段。

下面就分别使用DELETE语句根据条件删除表中的记录和删除表中全部记录。

(1) 根据条件删除表中的记录

根据条件删除表中的记录就是使用[WHERE condition]子句来完成。下面就删除LOGIN表中用户名是“AAA”的记录。具体操作如图4.28所示。

这样，再查看LOGIN表中的数据时就可以看出其中已经不存在用户名是“AAA”的记录了。

(2) 删除表中全部记录

删除表中全部记录就是不使用[WHERE condition]子句来完成操作。下面就删除LOGIN表中的全部记录。具体操作如图4.29所示。



图4.28 根据条件删除记录

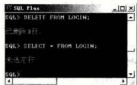


图4.29 删除全部记录

这样就删除了表LOGIN中的全部记录，再查询LOGIN表时就可以看到查询结果是“未选定行”，这代表LOGIN表中已经不存在数据了。

4.5.4 查询数据就用SELECT

数据查询语言也称为DQL，这部分内容将在第5章中详细介绍。在本小节中主要介绍SELECT语句的基本用法。SELECT的一般语法如下：

```
SELECT column_name1, column_name2,... FROM table_name WHERE[condition];
```

【语法说明】

- ☐ column_name1: 代表的是数据表中的字段名，可以查询数据表中的一个或多个字段，同时可以使用“*”号代替数据表中所有的字段。
- ☐ WHERE[condition]: 代表的是查询的条件，如果不指定查询条件则查询数据表中所有的记录；如果指定查询条件，那么就可以根据查询条件来查询记录了。

下面分别使用SELECT语句查询表中的记录。



(1) 查询表中全部数据

查询表中全部数据可以直接使用SELECT语句完成。下面查询LOGIN表中的全部记录，具体操作如图4.30所示。从查询结果就可以看出查询了表LOGIN中全部字段的全部数据。

(2) 查询表中某一字段的数据

查询表中某一个字段的数据可以直接在SELECT语句后面指定要查询的字段名。下面就查询LOGIN表中LOGINNAME的值，具体操作如图4.31所示。



图4.30 查询全部数据



图4.31 查询LOGINNAME列

在SELECT语句后面指定了列名LOGINNAME，这样就可以在LOGIN表中只查询LOGINNAME列的值。

说明

如果在实际应用中只需要表中某些列的值，最好是指定出列名来查询，不要使用“*”号来查询全部的记录，因为查询全部记录会影响查询的效率。

(3) 根据条件查询数据

根据条件查询数据就是使用WHERE[condition]子句来完成操作。下面就查询LOGINNAME是“AAA”的记录。具体操作如图4.32所示。



图4.32 根据条件查询记录

通过带条件的查询可以看出，在LOGIN表中只查询出了LOGINNAME是“AAA”的记录。

4.5.5 其他数据操纵语句

在Oracle 11g中除了上面所讲述的INSERT、UPDATE、DELETE、SELECT语句之外，还有MERGE、TRUNCATE、LOCK TABLE等语句。在本小节中再讲述一下比较常用的TRUNCATE语句和MERGE语句的使用。

(1) TRUNCATE语句

TRUNCATE语句和DELETE语句一样都是用来完成删除数据表中数据的，但是二者是有区别的。使用TRUNCATE语句删除表中的记录都是要把表中的记录全部删除，但是TRUNCATE语句删除表中数据的速度要比使用DELETE语句删除表中的数据更快一点。具体

语法如下:

```
TRUNCATE TABLE table_name;
```

这里,只要指定要删除的表名就可以删除表中的所有记录,它是无条件的删除。下面就使用该语句删除表LOGIN中的记录。具体操作如图4.33所示。

(2) MERGE语句

MERGE语句与UPDATE语句的功能类似,都是修改数据表中数据的,但是MERGE语句与UPDATE语句也是有区别的。使用MERGE语句可以对数据表同时进行增加和修改的操作。具体语法如下:

```
MERGE [INTO] table_name1
USING table_name2
ON ( condition )
WHEN MATCHED THEN merge_update_clause
WHEN NOT MATCHED THEN merge_insert_clause;
```

【语法说明】

- table_name1: 要修改或添加的表。
- table_name2: 参照的更新的表。
- condition: table_name1和table_name2之间的关系,或其他的一些条件。
- merge_update_clause: 如果和参照表table_name2中的条件匹配,就执行更新操作的SQL语句。
- merge_insert_clause: 如果条件不匹配,就执行增加操作的SQL语句。

注意

这里merge_update_clause和merge_insert_clause都是可以省略的,但是在操作时只能省略一个,如果两个语句都省略,那么MERGE语句就失去意义了。

下面就使用MERGE语句完成对LOGIN表的操作。

(1) 省略增加的语句

首先分别查询LOGIN与MANAGERINFO表中的所有数据,然后使用MERGE语句进行更新操作。更新的条件是两个表的编号列相同,当满足条件时把LOGIN表中满足条件记录的LOGINNAME列改写成MANAGERINFO表中的LOGINNAME列。具体操作如图4.34所示。

这样,更新操作完成后就可以把LOGIN_ID是1的记录中的LOGINNAME更新为表MANAGERINFO中的LOGINNAME的值“XIAOMING”。

(2) 省略修改的语句

首先分别查询LOGIN与MANAGERINFO表中的所有数据,然后使用MERGE语句进行增加操作。增加的条件是两个表的编号列不相同,当满足条件时向LOGIN表中增加一条在MANAGERINFO中不存在的数据。具体操作如图4.35所示。

这样,就可以看出LOGIN表中已经增加了两条记录,这两条记录编号是原来在LOGIN_ID中不存在的。在实际操作中,可以使用这种方法把数据表中不存在的记录添加到当前的数据表中。



图4.33 删除表中记录



SQL> SELECT * FROM LOGIN;

LOGINNAME	PASSWORD	LOGIN_ID
DBA	123	1
SCOTT	123	2
ADMIN	456	3

SQL> SELECT * FROM MANAGERINFO;

MANAGER_ID	LOGINNAME	PASSWORD	NAME	TEL
5	XIAOMING	123456	LIJING	13512345678
2	WANG	MANAGERINFO		
3	WANG	LOGIN_ID=MANAGERINFO.MANAGER_ID		
4	WANG	MANAGER_ID=LOGIN_ID		
5	WANG	LOGIN_ID=MANAGERINFO.MANAGER_ID		

SQL> MERGE INTO LOGIN

- 1 USING MANAGERINFO
- 2 ON LOGIN, LOGIN_ID=MANAGERINFO.MANAGER_ID
- 3 WHEN MATCHED THEN UPDATE
- 4 SET LOGIN.LOGINNAME=MANAGERINFO.LOGINNAME
- 5 WHEN NOT MATCHED THEN INSERT
- 6 VALUES(MANAGERINFO.LOGINNAME, MANAGERINFO.PASSWORD, MANAGERINFO.MANAGER_ID)

SQL>

SQL> SELECT * FROM LOGIN;

LOGINNAME	PASSWORD	LOGIN_ID
DBA	123	1
SCOTT	123	2
ADMIN	456	3

SQL>

图4.34 更新LOGIN表

SQL> SELECT * FROM LOGIN;

LOGINNAME	PASSWORD	LOGIN_ID
DBA	123	1
SCOTT	123	2
ADMIN	456	3

SQL> SELECT * FROM MANAGERINFO;

MANAGER_ID	LOGINNAME	PASSWORD	NAME	TEL
1	XIAOMING	123456	LIJING	13512345678
2	WANG	45678	LIJING	13512345678
3	WANG	45678	LIJING	13512345678

SQL> MERGE INTO LOGIN

- 1 USING MANAGERINFO
- 2 ON LOGIN, LOGIN_ID=MANAGERINFO.MANAGER_ID
- 3 WHEN MATCHED THEN UPDATE
- 4 SET LOGIN.LOGINNAME=MANAGERINFO.LOGINNAME, LOGIN.PASSWORD=MANAGERINFO.PASSWORD
- 5 WHEN NOT MATCHED THEN INSERT
- 6 VALUES(MANAGERINFO.LOGINNAME, MANAGERINFO.PASSWORD, MANAGERINFO.MANAGER_ID)

SQL>

SQL> SELECT * FROM LOGIN;

LOGINNAME	PASSWORD	LOGIN_ID
DBA	123	1
SCOTT	123	2
ADMIN	456	3
XIAOMING	123456	4
WANG	45678	5

SQL>

图4.35 增加数据

(3) 增加和修改同时进行

增加和修改同时进行是指当on后面的条件满足时执行修改的操作，不满足时执行增加的操作。首先也是分别查询LOGIN与MANAGERINFO表中的全部数据，然后使用MERGE语句进行增加和修改的操作。具体操作如图4.36所示。

SQL> SELECT * FROM LOGIN;

LOGINNAME	PASSWORD	LOGIN_ID
DBA	123456	1
SCOTT	123456	2
ADMIN	123456	3

SQL> SELECT * FROM MANAGERINFO;

MANAGER_ID	LOGINNAME	PASSWORD	NAME	TEL
5	XIAOMING	123456	LIJING	13512345678
2	LIJING	54321	LIJING	13512345678
4	ZHANGJIAN	54321	ZHANGJIAN	13512345678

SQL> MERGE INTO LOGIN

- 1 USING MANAGERINFO
- 2 ON LOGIN, LOGIN_ID=MANAGERINFO.MANAGER_ID
- 3 WHEN MATCHED THEN UPDATE
- 4 SET LOGIN.LOGINNAME=MANAGERINFO.LOGINNAME
- 5 WHEN NOT MATCHED THEN INSERT
- 6 VALUES(MANAGERINFO.LOGINNAME, MANAGERINFO.PASSWORD, MANAGERINFO.MANAGER_ID)

SQL>

SQL> SELECT * FROM LOGIN;

LOGINNAME	PASSWORD	LOGIN_ID
XIAOMING	123456	1
DBA	123456	2
SCOTT	123456	3
ZHANGJIAN	54321	4
LIJING	54321	5

SQL>

图4.36 增加和修改数据

由于在LOGIN中存在的编号是1、2、3；在MANAGERINFO中存在的编号是1、4、5，所以在经过了MERGE操作后，LOGIN中一共存在了5条记录，并且把编号是1的记录的LOGINNAME更改成了MANAGERINFO中的“XIAOMING”。

4.6 数据控制语言 (DCL)

数据控制离不开数据库的使用者，数据控制语言主要就是对数据库使用者赋予和撤销访问数据库的权限的设置，主要包括授予权限要使用的语句GRANT和收回权限的语句REVOKE。在第17章中详细介绍了如何使用GRANT和REVOKE设置用户权限，这里就不详细讲述了。

4.7 小结

本章详细介绍了与数据库密切相关的SQL语句，首先介绍了在Oracle 11g中支持的数据类型，然后详细介绍了DDL（数据定义语言）和DML（数据操纵语言）的使用，在DDL中还讲述了数据表中约束的使用。在本章的学习中，读者应尽可能多地使用所述的语句进行练习。SQL语句没有什么好的学习方法，只能多练习，以熟能生巧。

4.8 习题

简答题

1. SQL语言中一共有几种语言？
2. Oracle 11g中有哪些比较常用的数据类型？varchar2是什么类型？为什么要加上2呢？
3. 在Oracle中共存在几种约束？能够确保字段输入的值是18~30的约束是下列()约束？
A. Primary Key B. CHECK C. UNIQUE
4. 修改数据表时如果要添加一个约束，使用的语句是什么？
5. 向数据表中增加数据的方法有几种？分别都是什么？
6. DELETE与TRUNCATE语句的区别是什么？

第5章 利用SELECT检索数据

当在数据库的表中存入数据后，就可以查询这些已经存入的数据。查询数据需要用到SELECT语句，可以使用不同复杂程度的查询语句来检索需要的数据。本章知识点主要有：

- ☐ 如何使用SELECT语句
- ☐ 基本的SELECT语句检索数据
- ☐ 利用WHERE检索数据
- ☐ 查询多个表
- ☐ 利用子查询检索数据

5.1 查询数据必备SELECT

SELECT关键字表示数据的检索，它由一系列的子句组成，最终检索出来的数据是由子句决定的。也就是说，检索出来的数据必须满足所有子句的限制。SELECT语句按照复杂程度可以分为简单查询、WHERE条件查询、多表查询、子查询等。

5.1.1 SELECT语句语法

SELECT语句是日常使用最多的语句，它以SELECT开头。其中最主要的部分就是SELECT和FROM关键字，这两项是查询当中必需的部分，其他子句可以根据实际需求进行变动。SELECT语句的主要语法结构如下：

```
01  SELECT
02  [DISTINCT|ALL]
03      select_list
04  FROM table_list
05  [where_clause]
06  [group_by_clause]
07  [HAVING condition]
08  [order_by_clause]
```

【语法说明】

- ☐ SELECT：查询动作关键字，也是必需关键字。
- ☐ [DISTINCT|ALL]：描述列表字段中的数据是否去除重复记录。
- ☐ select_list：需要查询的字段列表，也可以说是占位符。可以是一个字段，也可以是多个字段。
- ☐ FROM：必需关键字，表示数据的来源。
- ☐ [where_clause]：查询的WHERE条件部分。
- ☐ [group_by_clause]：GROUP BY子句部分。
- ☐ [HAVING condition]：HAVING子句部分。

□ [order_by_clause]: 排序。

所谓简单的查询,指的是语法中的第1~4行。这种查询没有条件的限制,只是把要查询的字段中的数据列出来。其中select_list的具体语法如下:

```
{
* |
{ [ schema. ] { table | view } .*
| expr [ ( AS | c_alias )
}
```

【语法说明】

- schema: 模式名称。
- table | view: 表或视图。
- expr: 表达式。
- c_alias: 别名。

SELECT语句中允许利用表达式或函数对符合条件的数据进行处理。

5.1.2 获取指定字段的数据

获取表中指定字段的数据,就是指定表中的某几个字段(列),然后利用SELECT语句得到指定字段的数据,多个字段之间使用逗号分隔。这里以表PRODUCTINFO为例进行演示,该表的数据结构前面已经介绍过了,不熟悉的用户可以到4.3节中查看。

【示例1】检索PRODUCTINFO部分字段

要求查询表PRODUCTINFO中产品ID、产品名称、产品价格的数据。在SQL*Plus下执行如下语句:

```
SELECT productid,productname,productprice FROM productinfo;
```

【执行效果】

执行效果见图5.1。

PRODUCTID	PRODUCTNAME	PRODUCTPRICE
02-400010001	香醇LCE-46C10000	7000
02-400010001	洁尔XQ05-0-9180	1100
02-400010002	三型MF-810655/XSC	3600
02-400010003	三型XQ05-1060/XSC	2500
02-400050001	世世C430T	400
02-400020001	三三BU002路由器	2000
02-400010002	洁尔	10
02-400010001	大堂伞	50

已选择8行。

图5.1 查询指定字段数据

使用查询时,可以指定某个模式下的表或视图的列,以上的查询语句也可以使用下面形式的脚本替代:



```
SELECT scott.productinfo.productid,productinfo.productname,productinfo.productprice
FROM scott.productinfo;
```

【语法说明】

□ scott: 当前模式名称。

□ productinfo: 当前模式下的表名称。

利用这种写法可以查询其他模式下的表数据。

5.1.3 获取所有字段的数据

要想查看某表所有字段的数据，最简单的写法就是利用星号(*)来查询，星号属于通配符的一种，它只能用在SELECT语句中。读者需要注意，星号或者列名至少选一种。

【示例2】查询所有字段数据示例

要求查询表PRODUCTINFO中所有列的数据。在SQL*Plus下执行如下语句：

```
SELECT * FROM productinfo;
```

【执行效果】

执行效果见图5.2。

PRODUCTID	PRODUCTNAME	PRODUCTPRICE	QUANTITY	CATEGORY	DESPERATION	ORIGIN
002-4100-00001	索尼 LCB-46-G1 0000	70000	200	01-0000-000001		日本
002-4100-00001	索尼 XQB55-7180	11000	270	01-0000-000002		日本
002-4100-00002	索尼 R100S5-XSC	26000	12	01-0000-000002		日本
002-4100-00002	索尼 XQB55-186A-XSC	25000	111	01-0000-000002		日本
002-4100-00001	索尼 C4301	4000	129	01-0000-000001		日本
002-4100-00001	索尼 R100S2 路由器	20000	22	01-0000-000001		日本
002-4100-00002	索尼	10		01-0000-000001		日本
002-4100-00002	索尼	59	50	01-0000-000001		美国

图5.2 查询所有字段数据

该语句的执行效果和下面一段脚本的执行效果是一样的，只不过使用星号的书写方式更方便。脚本的执行效果这里不再给出。

```
SELECT productid, productname, productprice, quantity, category, desperation, origin
FROM productinfo;
```

虽然使用星号查询数据比较方便，但这里不建议大家这么使用。我们应该明确返回自己需要列的值，而不是把所有的数据全部返回，这么做主要原因有以下几点：

注意

- 1) 查询明确的列在执行效率上比使用通配符(*)要高。
- 2) 只返回必要的列的数据可以减少网络消耗。
- 3) 如果使用通配符(*)返回所有列的数据，当在表中增加新的字段时，有可能引起应用程序的异常。



5.1.4 使用别名替代表中的字段名

表中的字段名称通常都是英文的，这会给英文不好的客户查看数据带来不便。其实这种情况完全可以避免，SELECT语句中的列名允许我们指定别名，指定别名可以利用AS关键字。

【示例3】查询中使用别名示例

要求为查询出来的字段都加上汉字的别名，脚本如下：

```
SELECT productid 产品编号, productname AS 产品名称, productprice AS 产品价格 FROM productinfo;
```

【执行效果】

执行效果见图5.3。

产品编号	产品名称	产品价格
02-000100001	液晶 LCD-66G1000	7000
02-000100001	液晶 XQ650-9100	1100
02-000100002	手机 BT6632-ABC	2600
02-000100003	手机 XQ655-T81A-RSC	2500
02-000100004	耳机 CA301	500
02-000100001	耳机 L-R0002扬声器	2000
02-000100002	耳机	300
02-000100001	耳机	50

图5.3 查询中使用别名

给字段定义别名可以使用AS关键字，如果不使用AS而是直接在查询的列名后面加上空格然后输入别名也可以达到同样的效果。

5.1.5 使用表达式操作查询的字段

可以针对某个列（字段）使用表达式，这样查询出来的结果就是修改后的数据，但是数据库里的数据不会被修改。

【示例4】查询中使用表达式

查询的产品价格在各产品原价格基础上提高1/4，并列出具计算公式。脚本如下：

```
01 SELECT productid,productname,productprice || '*' || 1.25 || '=' || productprice*1.25 AS
   new_productprice
02 FROM productinfo;
```

【代码解析】

第1行使用了两种操作符，“*”代表乘号，这在算术表达式中经常出现。“||”是连接操作符，它用来连接两个字符串，就像Java语言中用来连接两个字符串的加号。

【执行效果】

执行效果见图5.4。

示例4中产品价格和一个常数相乘，最终得出了新的价格。在查询中各个字段相互之间也可以完成类似的操作，例如产品价格和产品数量相乘得到产品的总价值。这在日常开发过程中经常用到，灵活运用这些方法，可以达到很好的查询效果。

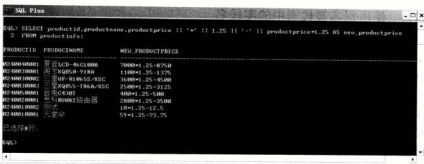


图5.4 查询中含有表达式

5.1.6 使用函数操作查询的字段

查询过程中检索的列允许使用函数对其操作，如果仅仅是查询，那么更多的是利用函数对数据进行类型转换。至于SQL的内置函数，会在第6章对其介绍。

【示例5】查询中使用函数操作字段

利用函数 subStr，对数据库字段进行截取，该函数的用法在SQL函数章节有相关介绍。在SQL*Plus下执行以下查询脚本：

```

SELECT productid 产品编号,subStr(productid,1,6) as 截取后的编号,
       productname AS 产品名称, productprice AS 产品价格
FROM productinfo;

```

【执行效果】

执行效果见图5.5。



图5.5 使用函数操作列值

5.1.7 去除检索数据中的重复记录

当查询数据时有可能遇到重复的记录，这给统计有效数据造成了一定的影响，利用DISTINCT关键字可以去除重复的数据。

【示例6】 去除重复的产品类型编码数据**(1) 查看产品类型编码数据**

产品表PRODUCTINFO中的产品类型编码存在重复数据。利用如下脚本查看所有数据：

```
SELECT category 产品类型 FROM productinfo;
```

【执行效果】

执行效果见图5.6。

(2) 去除重复的产品类型编码数据

利用DISTINCT去除产品表PRODUCTINFO中产品类型编码的重复数据。利用如下脚本查看：

```
SELECT distinct(category) 产品类型 FROM productinfo;
```

【执行效果】

执行效果见图5.7。



图5.6 存在重复数据的产品类型编码



图5.7 去除重复数据

从执行效果中可以看到，使用DISTINCT后已经没有重复的记录。DISTINCT后面如果是多个列名，那么DISTINCT把这些列名看成一个整体，来去除重复记录。

5.2 检索出来的数据排序

要从海量的数据中找到我们模糊记忆中的数据，使用排序是个不错的选择，可以考虑让查询出来的记录根据某个字段进行排序操作。

5.2.1 使用排序的语法

排序需要放置在SELECT语句的后面，不管有什么条件限制，排序关键字只能在最后一项。下面是排序的语法，对应5.1.1小节中SELECT基本语法中的order_by_clause项。

```
ORDER BY
{expr | position | c_alias}
[ASC | DESC ]
[NULLS FIRST | NULLS LAST]
[, {expr | position | c_alias}
[ASC | DESC]
[NULLS FIRST | NULLS LAST]
]
```

第5章 利用SELECT检索数据 5.2.1

☐ ORDER BY: 排序关键字。

□ **expr**: 表达式。

□ position: 表中列的位置。

❑ **c_alias**: 别名。

□ [ASC | DESC]: 升序或降序。

❑ **NULLS FIRST | NULLS LAST**: 对空字段的处理方式。

☐ 可以根据多个字段排序。

排序在整理数据时很有用，客户通常要求报表中的数据是按照一定顺序排列的，例如按照姓名排序或按照时间排序。

可以使用升序方式或降序方式对查询出来的数据进行排序, 如果对某个字段使用了 ORDER BY 子句而不指定排序方式, 那么它将以升序的方式排列指定字段的数据。

查询PRODUCTINFO中的产品名称和数量，并按照数量排序。脚本如下：

```
SELECT productname, quantity FROM productinfo ORDER BY quantity;
```

查询结果见图5.8.

从图5.8中可以看出查询结果已经按照数量进行了升序排列，升序排列是默认排序方式，显式地使用升序排列，只需在排序关键字后指定ASC即可，而利用DESC就可以达到降序的目的。

使查询出来的数据按照数量进行降序排列。脚本如下：

```
SELECT productname, quantity FROM productinfo ORDER BY quantity DESC;
```

执行效果见图5.9。



图5.8 默认排序



图5.9 降序排列

NULL值在排序过程中是个比较特殊的值类型，默认情况下排序时把它看成最大值。也就

是说，当排序的记录中出现NULL值时，默认情况下，升序排列时它在最后，降序排列时它在首位。其实NULL值在排序时，具体在前还是在后开发人员是可以指定的。

【示例9】升序时NULL值在首位

进行升序排列时要求数量为NULL的值排在首位。脚本如下。

```
SELECT productname, quantity FROM productinfo ORDER BY quantity NULLS FIRST;
```

【执行效果】

执行效果见图5.10。

【示例10】 降序时NULL值在末位

进行降序排列时要求数量为NULL的值排在末位。脚本如下。

```
SELECT productname, quantity FROM productinfo ORDER BY quantity NULLS LAST;
```

【执行效果】

执行效果见图5.11。

[illegible][illegible]

图5.10 NULL值在首位

图5.11 NULL值在末位

5.2.4 使用别名作为排序字段

前面介绍过，查询时可以给列名指定一个别名，也可以使用别名指定排序。

【示例11】为别名指定排序

为PRODUCTINFO中的数据指定别名，并为该别名指定排序。脚本如下：

SELECT productname 产品名称, quantity 产品数量 FROM productinfo ORDER BY 产品数量 NULLS LAST;

【执行效果】

执行效果见图5.12。

The screenshot shows a SQL query execution window with the following content:

SQL Plan

```
SQL> SELECT productname 产品名称, quantity 产品数量 FROM productinfo ORDER BY 产品数量 NOELS LAST;
```

产品名称 **产品数量**

三星BF-B10655-ZSC	12
海信 H50-4651mm	20
海信 H50M2222 显示器	22
海信 H50M2222 显示器	29
三星BF-B10655-ZSC	50
三星BF-B10655-ZSC	111
三星BF-B10655-ZSC	129

共返回7行。

图5.12 为别名排序

使用列的别名作为排序字段和使用该列作为排序字段效果是一样的。开发人员可根据自己的习惯进行选择，但在被排序的字段有别名情况下建议使用别名作为排序字段，这样更易于阅读。

5.2.5 使用表达式作为排序字段

查询中允许使用表达式处理字段数据，其实也允许使用表达式作为排序字段。使用表达式作为排序字段和使用别名作为排序字段类似。

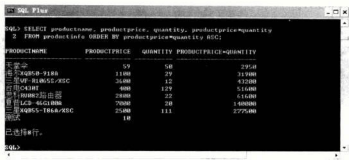
【示例12】表达式作为排序字段

利用产品价格和产品数量的乘积作为排序字段。脚本如下：

```
SELECT productname, productprice, quantity, productprice*quantity
FROM productinfo ORDER BY productprice*quantity ASC;
```

【执行效果】

执行效果见图5.13。



PRODUCTNAME	PRODUCTPRICE	QUANTITY	PRODUCTPRICE*QUANTITY
王老吉	59	50	2950
三鞭酒	1100	29	31900
三星牌	3400	12	40800
可口可乐	400	129	51600
三星牌	2800	22	61600
三星牌	7000	20	140000
三星牌	2500	111	277500
三星牌	10		

图5.13 表达式作为排序字段

注意 NULL值和其他值相乘时结果还是NULL，这一点读者需要注意。

5.2.6 使用字段的位置作为排序字段

排序时允许使用查询列表中字段的位置来作为排序字段，这么做一是为了方便，二是为了防止使用UNION时出现错误。

【示例13】利用字段位置排序

查询表PRODUCTINFO中产品名称、产品价格、产品数量，并利用产品数量所在位置排序。脚本如下：

```
SELECT productname, productprice, quantity FROM productinfo ORDER BY 3 DESC;
```

【执行效果】

执行效果见图5.14。

注意 利用字段在查询列表中的位置作为排序字段时，表示位置的数字不能超出查询列表中的字段的个数。



SQL> SELECT productname, productprice, quantity FROM productinfo ORDER BY 3 DESC;

PRODUCTNAME	PRODUCTPRICE	QUANTITY
测试	10	
三洋CASHI	4000	129
三洋GPS-180A-RSC	25000	111
三星	55	10
松下AQ050-918A	11000	29
飞利浦MG1000	20000	22
三星LCD-86G1000	70000	20
三星NP-8100S-RSC	16000	12

已选择4行。

图5.14 利用字段位置作为排序字段

5.2.7 使用多个字段排序

排序非常灵活，它不仅可以利用单一字段进行排序，还能利用多个字段进行排序，最后查询出来的数据是综合排序后的数据。多个字段排序的操作过程如下：

- 1) 按照第一个字段进行排序。
- 2) 在此基础上按照第二个字段排序。也就是说，当第一个字段的数据相同时，才对这些数据以第二个字段排序。
- 3) 如果还有后面的排序字段，那么这个过程会不断重复，而可以排序的数据范围，都是前面一个字段中重复的数据。

【示例14】利用多个字段排序

要求查询PRODUCTINFO表中的产品名称、产品类型编码、产品数量，并利用后面两个字段排序。脚本如下：

```
SELECT productname, category, quantity FROM productinfo ORDER BY CATEGORY ASC, 3 DESC;
```

【执行效果】

执行效果见图5.15。

SQL> SELECT productname, category, quantity FROM productinfo ORDER BY CATEGORY ASC, 3 DESC;

PRODUCTNAME	CATEGORY	QUANTITY
测试	0100010001	
三星	0100010005	50
飞利浦MG1000	0100020005	22
三星LCD-86G1000	0100010003	20
三星GPS-180A-RSC	0100020002	111
松下AQ050-918A	0100020002	29
三星NP-8100S-RSC	0100010002	12
三洋CASHI	0100010002	129

已选择4行。

图5.15 多字段排序

说明

利用多个字段进行排序，可以单独为每个字段指定排序方式，而且前面介绍过的别名、表达式和位置排序可以混用。



5.3 使用WHERE子句设置检索条件

SELECT...FROM是一个基本的查询语句，它会无差别地返回所有的值，但这通常不是我们想要的，我们希望检索出来的数据是满足某个甚至某些条件的，而利用WHERE子句可以达到我们的目的。WHERE子句就像一个筛选器，它对FROM子句返回的结果进行筛选，每条记录都会按照条件进行判断，如果符合条件，则该记录作为查询结果的一部分，如果不符合条件则不会返回。

WHERE条件子句中可以使用操作符主要有关系操作符、比较操作符和逻辑操作符。

1) 关系操作符包括：<、<=、>、>=、=、!=、<>。

2) 比较操作符包括：

□ IS NULL：如果操作数为NULL返回TRUE。

□ LIKE：模糊比较字符串值。

□ BETWEEN...AND...：验证值是否在范围之内。

□ IN：验证操作数在设定的一系列值中。

3) 逻辑操作符包括：

□ AND：两个条件都必须得到满足。

□ OR：只要满足两个条件中的一个。

□ NOT：与某个逻辑值取反。

简单的WHERE条件语句一般只有一个限制条件，但是如果单一的限制条件不能满足我们的业务需求时，开发人员可以使用多个限制条件查询数据，多个限制条件之间可以使用逻辑操作符相连接。这些在下面的小节中会做详细介绍。

5.3.1 查询中使用单一条件限制

这里所说的单一的查询条件主要针对关系操作符来讲，至于比较操作符的应用会在后面介绍。在WHERE条件中也可以使用函数。

【示例15】查询中使用“>”

检索表PRODUCTINFO中产品名称、产品价格、产品数量，列出的记录要求产品数量大于20，并根据产品数量升序排列。脚本如下：

```
SELECT productname, productprice, quantity FROM productinfo WHERE quantity > 20 ORDER BY quantity;
```

【执行效果】

执行效果见图5.16。

PRODUCTNAME	PRODUCTPRICE	QUANTITY
产品1 (PRODUCT1)	10000	22
产品2 (PRODUCT2)	11000	25
产品3 (PRODUCT3)	12000	28
产品4 (PRODUCT4)	13000	31
产品5 (PRODUCT5)	14000	34

图5.16 数量大于20的数据



【示例16】查询中使用“<>”

检索表PRODUCTINFO中产品名称、产品价格、产品数量、产品类型编码。列出的记录要求“产品类型编码”不为“0100030002”。脚本如下：

```
SELECT productname,productprice, quantity,category FROM productinfo
WHERE category <> '0100030002';
```

【执行效果】

执行效果见图5.17。

利用“<>”作为查询条件时，可以使用“!=”替换；category字段是字符串类型，在查询时字符串类型数据需要用单引号括起。

【示例17】查询条件中使用函数

检索表PRODUCTINFO中产品ID、产品名称、产品价格、产品数量、产品类型编码。列出的记录要求“产品ID”的前6位是“024003”。脚本如下：

```
SELECT productid,productname,productprice, quantity,category FROM productinfo
WHERE SUBSTR(productid,1,6) = '024003';
```

【执行效果】

执行效果见图5.18。

PRODUCTNAME	PRODUCTPRICE	QUANTITY	CATEGORY
01000300000	1000	10	0100030000
01000300010	2000	20	0100030001
01000300020	3000	30	0100030002
01000300030	4000	40	0100030003
01000300040	5000	50	0100030004

图5.17 产品类型编码不为“0100030002”

PRODUCTID	PRODUCTNAME	PRODUCTPRICE	QUANTITY	CATEGORY
02400300000	0240030000	1000	10	0100030000
02400300010	0240030001	2000	20	0100030001
02400300020	0240030002	3000	30	0100030002
02400300030	0240030003	4000	40	0100030003
02400300040	0240030004	5000	50	0100030004

图5.18 利用函数查询数据

5.3.2 查询中使用多个条件限制

查询条件中除了单一的条件也可以设置多个条件，但是这些条件需要使用逻辑操作符连接起来。例如，使用AND表示多个条件需要同时满足，其中有一个条件不能满足，那么该记录就不会返回到查询结果中。而OR表示多个条件中符合其中一个就能返回到查询结果中。

【示例18】查询条件使用AND

检索表PRODUCTINFO中产品ID、产品名称、产品价格、产品数量。返回记录要求“产品价格”在1000～7000之间，并且包含这两端价格。脚本如下：

```
SELECT productid,productname,productprice, quantity FROM productinfo
WHERE productprice >=1000 AND productprice <=7000;
```

【执行效果】

执行效果见图5.19。

该示例的要求除了使用AND连接多个查询条件实现外，也可以利用BETWEEN...AND...语句完成，BETWEEN...AND...语句用来检索指定范围内的数据。下面的示例将演示利用



BETWEEN...AND...完成同样的功能。

【示例19】查询条件使用BETWEEN...AND...

该示例将完成上一个示例中的功能。脚本如下：

```
SELECT productid,productname,productprice, quantity FROM productinfo
WHERE productprice BETWEEN 1000 AND 7000;
```

【执行效果】

执行效果见图5.20。

```
SQL> SELECT productid,productname,productprice, quantity FROM productinfo
2 WHERE productprice < 1000 AND productprice < 7000;
```

PRODUCTID	PRODUCTNAME	PRODUCTPRICE	QUANTITY
4000-00001	三星LCD-46GL008	7000	10
4000-00002	三星UP-R10655/XSC	1600	20
4000-00003	三星C4301	400	179
4000-00002	三星	10	10
4000-00001	三星	59	50

图5.19 利用AND查询

```
SQL> SELECT productid,productname,productprice, quantity FROM productinfo
2 WHERE productprice BETWEEN 1000 AND 7000;
```

PRODUCTID	PRODUCTNAME	PRODUCTPRICE	QUANTITY
4000-00001	三星LCD-46GL008	7000	10
4000-00002	三星UP-R10655/XSC	1600	20
4000-00003	三星C4301	400	179
4000-00002	三星	10	10
4000-00001	三星	59	50

图5.20 利用BETWEEN...AND...指定范围

BETWEEN...AND...语句中的BETWEEN后面需放置低标界数，也就是较小的数，AND后面放置高标界数，也就是较大的数。而且该语句表示的范围是闭区间的。

【示例20】查询条件使用OR

检索表PRODUCTINFO中产品ID、产品名称、产品价格、产品数量。返回记录要求“产品价格”低于1000或高于3000，不包含这两个价格本身。脚本如下：

```
SELECT productid,productname,productprice, quantity FROM productinfo
WHERE productprice < 1000 OR productprice > 3000;
```

【执行效果】

执行效果见图5.21。

```
SQL> SELECT productid,productname,productprice, quantity FROM productinfo
2 WHERE productprice < 1000 OR productprice > 3000;
```

PRODUCTID	PRODUCTNAME	PRODUCTPRICE	QUANTITY
4000-00001	三星LCD-46GL008	7000	10
4000-00002	三星UP-R10655/XSC	1600	20
4000-00001	三星C4301	400	179
4000-00002	三星	10	10
4000-00001	三星	59	50

图5.21 查询条件使用OR

注意 这里需要注意的是SQL中没有“1000<=productprice<=7000”这种形式。

5.3.3 模糊查询数据

当并不能确切地了解查询条件，而是只了解查询条件中的一部分时，或者想检索出包含特



定字符的数据时，可以利用模糊查询。

使用模糊查询的关键字是LIKE，它和两个通配符一起使用，才能实现模糊查询的功能。用这两个通配符可以替代模糊的部分，它们分别是：

□ _：可以替代一个字符。

□ %：可以替代多个字符。

【示例21】利用LIKE查询数据

要求检索出“产品名称”字段中包含“三星”两个字的产品及其价格。脚本如下：

```
SELECT productname,productprice FROM productinfo WHERE productname LIKE '%三星%';
```

【执行效果】

执行效果见图5.22。

PRODUCTNAME	PRODUCTPRICE
UF-B10655/XSC	36000
XQ955-186A/XSC	25000

图5.22 模糊查询“三星”产品

示例中的查询方式表示查询数据中只要有“三星”这两个字就符合查询条件，如果把第一个%去掉，那么表示查询前两个字是“三星”的记录。

5.3.4 查询条件限制在某个列表范围之内

某种情况下，要求查询条件从给定的值中选取，这时就可以利用IN关键字来实现这个功能。它的语法是IN(list)，其中list是值列表。

【示例22】利用IN查询数据

要求利用IN关键字，检索出产品类型编码为“0100030002”和“0100010001”的产品名称和产品价格。脚本如下：

```
SELECT productname,productprice FROM productinfo
WHERE CATEGORY IN('0100030002','0100010001');
```

【执行效果】

执行效果见图5.23。

PRODUCTNAME	PRODUCTPRICE
XQ955-186A	1800
UF-B10655/XSC	36000

图5.23 利用IN查询数据

利用IN检索出来的数据同分别利用“0100030002”和“0100010001”查询出来的数据总数是一样的。这种形式的查询通常出现在带有子查询的SELECT语句中。

如果在IN前面使用NOT关键字，那么检索出来的数据将与示例相反。这一点很有用，读者应了解。

5.3.5 专门针对NULL值的查询

数据库中的数据不会是完美的，更多的时候，由于种种原因，会存在垃圾数据和NULL数据。如果要检索NULL数据，将如何操作呢？利用“=”是不行的，“=”不允许检索NULL数据。如果想要检索NULL数据，利用“IS NULL”就可以达到目的，而利用“IS NOT NULL”就可以检索非NULL的数据。

【示例23】检索NULL数据

要求查询产品数量为NULL的数据。脚本如下：

```
SELECT productid,productname,productprice, quantity FROM productinfo WHERE quantity IS NULL;
```

【执行效果】

执行效果见图5.24。

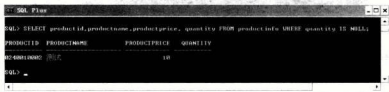


图5.24 查询NULL数据

如果使用“IS NOT NULL”语句，那么将查询除了该条记录以外的所有数据。

5.4 GROUP BY和HAVING子句

GROUP BY子句和HAVING子句同WHERE不一样，它们两个都用于组的查询。使用分组查询可以统计数据，例如利用GROUP BY配合分组函数，可以同时查询出每种类型产品的平均价格。至于什么是分组函数（集合函数），在第6章会做介绍。

5.4.1 GROUP BY子句语法及使用

GROUP BY用于归纳汇总相关数据，它不属于WHERE子句。也就是说，GROUP BY子句可以直接在FROM后面，也可以在WHERE条件后面。它在5.1.1小节基本语法中的表示方式是group_by_clause，下面把它细分。细分后的语法格式如下：

```
GROUP BY
{
  expr
| { ROLLUP | CUBE } ({expr [, expr ]...})
}
```


【语法说明】

□ expr: 通常表示数据库列名。

□ ROLLUP|CUBE: GROUP BY子句的扩展, 可以返回小计和总计记录。

GROUP BY语句和分组函数一起使用, 它可以根据某一列进行分组, 也可以根据某几列进行分组。

【示例24】根据某一个字段分组查询

计算出不同类型产品的平均价格。脚本如下:

```
SELECT category, AVG (productprice) 平均价格 FROM productinfo GROUP BY category;
```

【执行效果】

执行效果见图5.25。

CATEGORY	平均价格
01 (0000-000001)	4000
01 (0000-000001)	24000
01 (0000-000001)	5.9
11 (0000-000001)	1.0
01 (0000-000001)	70000
01 (0000-000001)	24000

图5.25 根据一列分组

【示例25】根据多个字段分组

根据不同产地, 计算不同类型产品的平均价格。脚本如下:

```
SELECT category 产品类型编码, AVG (productprice) 平均价格, origin 产地 FROM productinfo GROUP BY category, origin;
```

【执行效果】

执行效果见图5.26。

产品类型编码	平均价格	产地
01 (0000-000001)	70000	日本
11 (0000-000001)	1.0	
01 (0000-000001)	24000	中国
01 (0000-000001)	4000	中国
01 (0000-000001)	24000	中国
01 (0000-000001)	5.9	中国

图5.26 根据多个字段分组

虽然GROUP BY子句不允许出现在WHERE子句中, 但是允许出现在WHERE子句后面。也就是说, 被分组的数据是经过WHERE子句筛选过的。

【示例26】WHERE条件和分组混用

根据不同产地，计算价格高于1000的、不同类型产品的平均价格。脚本如下：

```
SELECT category, AVG (productprice) 平均价格, origin FROM productinfo
WHERE productprice>1000 GROUP BY category, origin;
```

【执行效果】

执行效果见图5.27。

CATEGORY	平均价格	ORIGIN
010000100001	7000	日本
010000100002	2000	中国
010000200001	2000	中国

图5.27 WHERE条件和分组混用

GROUP BY子句是统计数据时常用的语句。使用该子句时有以下几点需要注意：

- ☐ 当查询中存在GROUP BY子句时，SELECT列表中只能存在分组函数，或出现在GROUP BY子句中的字段。
- ☐ GROUP BY子句不允许出现在WHERE条件中，但允许出现其后，也就是可以和WHERE条件并列使用。

5.4.2 HAVING子句的使用

HAVING子句通常和GROUP BY子句一起使用，限制搜索条件。它和WHERE子句不一样，HAVING子句与组有关，而不与单个的值有关。在GROUP BY子句中，它会作用于GROUP BY创建的组。

【示例27】HAVING的使用方式

计算出不同类型产品的平均价格，并列出现平均价格高于2000的数据。脚本如下：

```
SELECT category, AVG (productprice) 平均价格
FROM productinfo GROUP BY category HAVING AVG (productprice)> 2000;
```

【执行效果】

执行效果见图5.28。

CATEGORY	平均价格
010000200001	2000
010000100002	2000

图5.28 HAVING限制搜索结果

从该示例脚本上可以看出HAVING与WHERE的区别，HAVING对GROUP BY子句负责，而WHERE对FROM负责。

5.5 使用子查询



什么是子查询？子查询就是嵌套查询，它是嵌套在另外一个语句中的SELECT语句。为什么会出现这种情况呢？主要原因是在很多情况下，WHERE后面的条件不是一个确切的值或表达式，而是另外一个查询语句的查询结果。子查询不仅仅出现在SELECT语句中，也会出现在DELETE和UPDATE语句中，它本质上是WHERE后的一个条件表达式。

出现这种情况的一个原因就是数据库设计导致的，但这种设计绝大部分是没有问题的。为了避免出现冗余数据，也为了避免出现不一致的数据，这么设计是完全正确的。例如，产品表PRODUCTINFO中的“产品类型编码”（category）字段引用了产品类型编码表（categoryinfo）的主键。假如知道产品类型，想检索该类型的产品具体包含了哪些产品，这时就不得不使用子查询。因为PRODUCTINFO表中没有“产品类型名称”字段，只有“产品类型编码”。

5.5.1 子查询返回单行

子查询允许返回单行数据，也允许返回多行数据。如果返回的是单行数据（不管是普通查询还是分组查询），那么这是逻辑上最简单的子查询嵌套查询语句。它和在WHERE条件中使用单一或多个条件限制的操作方法一致。

【示例28】单一条件子查询

要求查询产品类型为“MP3”的产品名称和产品价格。脚本如下：

```
SELECT productname,productprice FROM productinfo
WHERE CATEGORY =(SELECT categoryid FROM categoryinfo WHERE categoryname = 'MP3');
```

【执行效果】

执行效果见图5.29。



图5.29 查询产品类型为MP3的产品

【示例29】多个条件子查询

查询出产品价格在其最大值和最小值之间的产品，不包含两端的值。脚本如下：

```
SELECT productname,productprice FROM productinfo
WHERE productprice > (SELECT MIN(productprice) FROM productinfo)
AND productprice < (SELECT MAX(productprice) FROM productinfo)
```

【执行效果】

执行效果见图5.30。

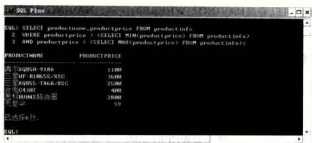


图5.30 查询产品价格在最大值和最小值之间的数据

5.5.2 子查询返回多行

如果子查询返回的值为多行值，那么需要用到IN关键字，此时IN的用法和前面介绍的方式一致。除此之外，也可以使用量化比较关键字SOME、ANY、ALL，这些需要配合<、<=、=、>、>=使用。它们所表示的含义如下：

- ANY：表示满足子查询结果的任何一个。和<、<=搭配，表示小于等于列表中的最大值；而和>、>=配合时表示大于等于列表中的最小值。
- SOME：可以认为和ANY含义相同。
- ALL：表示满足子查询结果的所有结果。和<、<=搭配，表示小于等于列表中的最小值；而和>、>=配合时表示大于等于列表中的最大值。

【示例30】IN示例

查询产品表中产品类型“电视”和“MP3”的数据。脚本如下：

```
SELECT productname,productprice FROM productinfo
WHERE CATEGORY IN
(SELECT categoryid FROM categoryinfo WHERE categoryname = '电视'
OR categoryname = 'MP3');
```

【执行效果】

执行效果见图5.31。



图5.31 使用IN操作子查询

【示例31】ANY示例

从产品表PRODUCTINFO中查询出价格低于指定价格列表中的最大值。指定的价格列表就



是指产品类型编码为“0100030002”的所有产品价格。脚本如下：

```
SELECT productname,productprice FROM productinfo
WHERE productprice <
ANY (SELECT productprice FROM productinfo WHERE category = '0100030002')
AND category <> '0100030002';
```

【执行效果】

执行效果见图5.32。

```
SQL> SELECT productname,productprice FROM productinfo
2 WHERE productprice <
3 ANY (SELECT productprice FROM productinfo WHERE category = '0100030002')
4 AND category <> '0100030002';
5
```

PRODUCTNAME	PRODUCTPRICE
百事可乐	400
百事可乐套餐	1000
百事	10

图5.32 使用ANY操作子查询

【示例32】SOME示例

```
SELECT productname,productprice FROM productinfo
WHERE productprice =
SOME (SELECT productprice FROM productinfo WHERE category = '0100030002')
AND category <> '0100030002';
```

SOME的用法和ANY一样，只不过ANY多用在非“=”的环境中。SOME这里表示找出和子查询中任何价格相等的产品。

执行效果这里不给出，没有符合的数据。

【示例33】ALL示例

找出比指定价格列表还低的产品数据。脚本如下：

```
SELECT productname,productprice FROM productinfo
WHERE productprice <
ALL (SELECT productprice FROM productinfo WHERE category = '0100030002');
```

【执行效果】

执行效果见图5.33。

```
SQL> SELECT productname,productprice FROM productinfo
2 WHERE productprice <
3 ALL (SELECT productprice FROM productinfo WHERE category = '0100030002')
4
```

PRODUCTNAME	PRODUCTPRICE
百事	10
百事可乐	400

图5.33 使用ALL操作子查询



5.6 连接查询

关系型数据库中允许表和表之间存在关系，这种关系可以把两个甚至多个表的数据联系在一起。利用这种关系，可以查询出某种符合条件的数据，这些数据将是一套符合实际业务逻辑的数据，而数据中这些表和表之间的关系将不存在。换句话说，获取真实世界的原始数据后，根据某种规则把它们拆分成了各种独立的数据，假如想从数据库中再次获取原始数据，那么需要依靠当初拆分时的规则。而这种规则也可以看成表和表之间的联系，要再次实现这种联系，需要用到连接查询。连接分为内连接、外连接和全连接，还有一种叫做自连接，其中最常用的是内连接和外连接。

5.6.1 最简单的连接查询

最简单的连接查询是利用逗号完成的，它利用逗号把FROM后的表名隔开，这就构成了最简单的连接查询。但这么做的意义不大。

【示例34】最简单的连接查询

利用表PRODUCTINFO和CATEGORYINFO实现最简单的连接查询。脚本如下：

```
SELECT * FROM productinfo,categoryinfo;
```

【执行效果】

执行效果见图5.34。

PRODUCTID	PRODUCTNAME	PRODUCTPRICE	QUANTITY	CATEGORY	DESCRIPTION	ORIGIN	CATEGORYID	CATEGORYNAME
01-0000-000001	鼠标	90.0000	70000	01-0000-000001		日本	01-0000-000001	鼠标
01-0000-000001	鼠标	90.0000	11000	01-0000-000002		中国	01-0000-000002	鼠标
01-0000-000002	键盘	30.0000	30000	01-0000-000003		中国	01-0000-000003	键盘
01-0000-000002	键盘	30.0000	25000	01-0000-000004		中国	01-0000-000004	键盘
01-0000-000003	显示器	100.0000	1000	01-0000-000005		中国	01-0000-000005	显示器
01-0000-000003	显示器	100.0000	2000	01-0000-000006		中国	01-0000-000006	显示器
01-0000-000004	显示器	100.0000	1000	01-0000-000007		中国	01-0000-000007	显示器
01-0000-000004	显示器	100.0000	1000	01-0000-000008		中国	01-0000-000008	显示器
01-0000-000005	显示器	100.0000	1000	01-0000-000009		中国	01-0000-000009	显示器
01-0000-000005	显示器	100.0000	1000	01-0000-000010		中国	01-0000-000010	显示器
01-0000-000006	显示器	100.0000	1000	01-0000-000011		中国	01-0000-000011	显示器
01-0000-000006	显示器	100.0000	1000	01-0000-000012		中国	01-0000-000012	显示器
01-0000-000007	显示器	100.0000	1000	01-0000-000013		中国	01-0000-000013	显示器
01-0000-000007	显示器	100.0000	1000	01-0000-000014		中国	01-0000-000014	显示器
01-0000-000008	显示器	100.0000	1000	01-0000-000015		中国	01-0000-000015	显示器
01-0000-000008	显示器	100.0000	1000	01-0000-000016		中国	01-0000-000016	显示器
01-0000-000009	显示器	100.0000	1000	01-0000-000017		中国	01-0000-000017	显示器
01-0000-000009	显示器	100.0000	1000	01-0000-000018		中国	01-0000-000018	显示器
01-0000-000010	显示器	100.0000	1000	01-0000-000019		中国	01-0000-000019	显示器
01-0000-000010	显示器	100.0000	1000	01-0000-000020		中国	01-0000-000020	显示器

图5.34 最简单的连接查询

图5.34中给出的结果并不是全部的结果，这里只是给出了一部分。利用这种方式查询数据将得到两个表的笛卡儿积，也就是说得到两个表中记录数的乘积。而这么做显然没有什么意义。

关于笛卡儿积，就是一个表中的每一行与另一个表中的每一行连接在一起而形成的新表，也就是查询结果。查询结果的记录数是这两个记录数的乘积。

【示例35】验证笛卡儿积

验证查询结果的记录数是被查询的两张表记录数的乘积。分别执行以下脚本：

```
SELECT COUNT(*) FROM productinfo;
SELECT COUNT(*) FROM categoryinfo;
SELECT COUNT(*) FROM productinfo,categoryinfo;
```

【执行效果】

执行效果见图5.35。

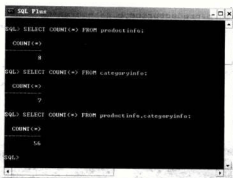


图5.35 验证笛卡儿积

如何才能使查询的数据有实际意义呢？需要使用WHERE条件，有了WHERE子句，就能查询出想要的有意义的数据了。

5.6.2 内连接

内连接也称为简单连接，它会把两个或多个表进行连接，只能查询出匹配的记录，不匹配的记录将无法查询出来。这种连接查询是平时最常用的查询。内连接中最常用的就是等值连接和不等值连接。

1. 等值连接

连接条件中使用“=”（等号）连接两个条件列表。

【示例36】等值连接

查询出PRODUCTINFO表和CATEGORYINFO表中产品类型编码一致的数据。执行以下脚本：

```

SELECT p.productname,p.productprice,c.categoryname
FROM productinfo p , categoryinfo c WHERE p.category = c.categoryid;
  
```

或

```

SELECT p.productname,p.productprice,c.categoryname
FROM productinfo p INNER JOIN categoryinfo c ON p.category = c.categoryid;
  
```

这两段脚本的功能是一样的。只是写法不一样。

【执行效果】

执行效果见图5.36。

2. 不等值连接

不等值连接就是指连接条件中使用“>”、“>=”、“<=”、“<”、“!=”、“<>”、“BETWEEN...AND...”、“IN”等连接两个条件列表，但这种方式通常需要和其他等值运算一起使用，否则检索出来的数据很可能没有实际意义。

【示例37】不等值连接

演示不等值连接的使用方式，执行以下脚本：

```
SELECT p.productname,p.productprice,c.categoryname
FROM productinfo p INNER JOIN categoryinfo c ON p.category IN c.categoryid;
```

```
SELECT p.productname,p.productprice,c.categoryname
FROM productinfo p , categoryinfo c WHERE p.category IN c.categoryid;
```

【执行效果】

执行效果见图5.37。

PRODUCTNAME	PRODUCTPRICE	CATEGORYNAME
PLATINUM	1000	PLATINUM
DIAMOND	1500	DIAMOND
RED DIAMOND	1200	RED
EMERALD	1100	EMERALD
EMERALD TRAIL	1000	EMERALD
EMERALD	900	EMERALD
EMERALD TRAIL	800	EMERALD
EMERALD	700	EMERALD
EMERALD	600	EMERALD
EMERALD	500	EMERALD

图5.36 等值连接

PRODUCTNAME	PRODUCTPRICE	CATEGORYNAME
PLATINUM	1000	PLATINUM
DIAMOND	1500	DIAMOND
RED DIAMOND	1200	RED
EMERALD	1100	EMERALD
EMERALD TRAIL	1000	EMERALD
EMERALD	900	EMERALD
EMERALD TRAIL	800	EMERALD
EMERALD	700	EMERALD
EMERALD	600	EMERALD
EMERALD	500	EMERALD

图5.37 不等值连接

内连接中的关键字“INNER JOIN”可以直接写成“JOIN”，系统会把“JOIN”识别成内连接。但是“ON”关键字不能省略。

5.6.3 自连接

所谓自连接，就是把自身表的一个引用作为另一个表来处理，这样就能获取一些特殊的数据。

【示例38】自连接

获取表PRODUCTINFO中数量相等的不同产品。执行以下脚本：

```
01 SELECT p.productname,p.productprice,pr.productname,pr.productprice,pr.quantity
02 FROM productinfo p,productinfo pr
03 WHERE p.productid != pr.productid
04 AND p.quantity = pr.quantity
05 AND p.rowid < pr.rowid;
```

【代码解析】

- ☐ 为了满足题意，该查询使用了多个连接条件，包括等值连接和不等值连接。
- ☐ 第3行表示查询两个表中产品ID不相等的的数据，这么做是为了避免查询结果的记录中出现两个产品ID一样的数据，这种数据是没有意义的，因为自身同自身的数量比永远是相等的。
- ☐ 第4行表示查询数量相等的的数据。
- ☐ 第5行表示去除重复的记录，只取ROWID比较小的那一条。

【执行效果】

执行效果见图5.38。



图5.38 自连接查询

示例中的脚本和下面的脚本同样是等效的。

```
SELECT p.productname,p.productprice,pr.productname,pr.productprice,pr.quantity
FROM productinfo p JOIN productinfo pr
ON p.productid != pr.productid
AND p.quantity = pr.quantity
AND p.rowid < pr.rowid;
```

语句中省略了“INNER”关键字。

5.6.4 外连接

外连接分为左外连接、右外连接、全外连接。它们所表示的含义如下：

- ☐ 左外连接：又称为左向外连接。使用左外连接的查询，返回的结果不仅仅是符合连接条件的行记录，还包含了左边表中的所有记录。也就是说，如果左表的某行记录在右表中没有匹配项，则在返回结果中右表的所有选择列表列均为空。
- ☐ 右外连接：又称为右向外连接。它与左外连接相反，将右边的表中所有的数据与左表进行匹配，返回的结果除了匹配成功的记录，还包含了右表中未匹配成功的记录，并在其左表对应的列补空值。
- ☐ 全外连接：返回所有匹配成功的记录，并返回左表未匹配成功的记录，也返回右表未匹配成功的记录。

下面用示例来理解这些概念。

1. 左外连接

【示例39】左外连接

要求检索出PRODUCTINFO表中每个产品对应的产品类型名称。执行以下脚本：

```
SELECT p.productname,p.productprice,p.category,c.categoryid,c.categoryname
FROM productinfo p LEFT JOIN categoryinfo c ON p.category = c.categoryid;
```

【代码解析】

- ☐ 脚本中productinfo是左边的表，它在LEFT JOIN的左边。该脚本将列出productinfo表的所有记录，并将表categoryinfo中的数据进行匹配。也就是说，此时以productinfo表为主表，如果该表中的数据在categoryinfo中没有匹配项，那么对应categoryinfo表中字段的位置用NULL值表示。

【执行效果】

执行效果见图5.39。

```
SQL> SELECT p.productname, p.productprice, p.category, c.categoryid, c.categoryname
2 FROM productinfo p LEFT JOIN categoryinfo c ON p.category = c.categoryid;
```

PRODUCTNAME	PRODUCTPRICE	CATEGORY	CATEGORYID	CATEGORYNAME
笔记本电脑	59	01100001100001	01100001100001	计算机
笔记本电脑	2000	01100001100001	01100001100001	计算机
笔记本电脑	7000	01100001100001	01100001100001	计算机
笔记本电脑	2500	01100001100001	01100001100001	计算机
笔记本电脑	3400	01100001100001	01100001100001	计算机
笔记本电脑	1100	01100001100001	01100001100001	计算机
笔记本电脑	800	01100001100001	01100001100001	计算机
笔记本电脑	10	01100001100001		

图5.39 左外连接

从图5.39中标示出来的部分可以看出在表categoryinfo中，没有对应“测试”的产品类型编码，此时使用NULL值来填充。

2. 右外连接

右外连接和左外连接是相反的，它以右边的表为主表。

【示例40】右外连接

查看哪些产品类型还没有对应的产品。执行以下脚本：

```
SELECT p.productname, p.productprice, p.category, c.categoryid, c.categoryname
FROM productinfo p RIGHT JOIN categoryinfo c ON p.category = c.categoryid;
```

【代码解析】

□ 在右外连接中，以categoryinfo为主，它将列出所有的数据，并与productinfo表中数据进行匹配。如果productinfo不能完全匹配categoryinfo的记录，那么categoryinfo没有被匹配的记录中对应productinfo的部分，将以NULL值填充。

【执行效果】

执行效果见图5.40。

```
SQL> SELECT p.productname, p.productprice, p.category, c.categoryid, c.categoryname
2 FROM productinfo p RIGHT JOIN categoryinfo c ON p.category = c.categoryid;
```

PRODUCTNAME	PRODUCTPRICE	CATEGORY	CATEGORYID	CATEGORYNAME
笔记本电脑	7000	01100001100001	01100001100001	计算机
笔记本电脑	1100	01100001100001	01100001100001	计算机
笔记本电脑	3400	01100001100001	01100001100001	计算机
笔记本电脑	2500	01100001100001	01100001100001	计算机
笔记本电脑	800	01100001100001	01100001100001	计算机
笔记本电脑	2000	01100001100001	01100001100001	计算机
笔记本电脑	59	01100001100001	01100001100001	计算机
笔记本电脑				

图5.40 右外连接

3. 全外连接

全外连接就是左外连接和右外连接的综合，它除了返回内连接匹配的数据外，也将返回两



对于以上右外连接脚本可以用以下脚本替代：

```
SELECT p.productname,p.productprice,p.category,c.categoryid,c.categoryname  
FROM productinfo p,categoryinfo c  
WHERE p.category(+) = c.categoryid;
```

使用(+)需要注意的地方有：

- ☐ 该操作符只能用在WHERE子句中，并且不能与OUTER JOIN一起使用。
- ☐ 该操作符不能用于全外连接。
- ☐ 如果外连接有多个条件，那么每一个条件都需要使用该操作符。

5.7 小结

本章主要介绍了SELECT语句的相关内容，包括SELECT的语法、如何指定字段别名、如何在查询中使用函数等。为了使数据更加有条理，可以对检索出来的数据进行排序，利用ORDER BY子句可以根据某个字段或多个字段对查询数据进行排序。如果想检索出特定的数据，那么可以利用WHERE子句设置检索条件，检索条件可以是一个也可以是多个，只需利用AND或OR连接起来即可。关于子查询，它可以作为另一个SELECT语句的查询条件，为动态地查询数据提供了条件。连接查询在日常开发中经常用到，读者应该认真学习。

5.8 习题

一、填空题

1. SELECT语句中，利用_____可以去除重复记录。
2. 为查询出来的数据进行排序，需要用到_____命令。
3. 排序时，降序命令是_____。
4. 模糊查询的命令是_____。
5. 模糊查询中，%表示_____。
6. 分组查询利用_____子句。

二、选择题

1. 内连接的语法是()。
A. INNER JOIN B. OUTER JOIN
2. 外连接分为()。
A. 左外连接 B. 右外连接
C. 全外连接 D. 自连接

第6章 Oracle内置函数

Oracle有多种内置函数，本章将重点介绍其中的两种，它们分别是单行函数和集合函数。这两种类型的函数使用频率比较高。

单行函数是指当查询表或视图时每行都能返回一个结果，可用于SELECT、WHERE、ORDER BY等子句中。而集合函数是作用在多行记录上返回一个结果，可用于带GROUP BY或HAVING子句的查询中。单行函数数量比较多，这里介绍其中常用的几种类型，它们分别是数值型函数、字符型函数、日期型函数、转换函数等。

介绍函数之前先简单介绍一下Oracle的DUAL表。该表是Oracle中真实存在的一个表，任何用户都可以读取，多数情况下可以用在没有目标的SELECT查询语句中。它本身只包含了一个DUMMY字段。DUAL表对Oracle很重要，用户不要试图删除该表，一旦删除，Oracle将无法启动。下面的函数讲解中会以DUAL表作为测试语句的目标表。

6.1 数值型函数

数值类型函数可以输入数字，并返回一个数值。大多数可以达到小数点后38位。一部分则支持30位或36位小数。本节主要介绍一些常用的数值函数。

6.1.1 绝对值、取余、判断数值正负函数

1) ABS(n)函数。用于返回绝对值。该函数输入一个参数，参数类型为数值型，假如参数为可以隐式转换成数值类型，那么也可以。示例脚本如下：

```
SELECT ABS(100),ABS(-100),ABS('100') FROM DUAL;
```

【执行效果】

执行效果见图6.1。

2) MOD(n2, n1)函数。该函数表示返回n2除以n1的余数。参数为任意数值或可以隐式转换成数值的类型。如果n1为0，那么该函数将返回n2。示例脚本如下：

```
SELECT MOD(5,2),MOD(8/3,5),MOD('10',5),MOD(-10,6),MOD(1,0) FROM DUAL;
```

【执行效果】

执行效果见图6.2。



SQL> SELECT ABS(100),ABS(-100),ABS('100') FROM DUAL;
ABS(100) ABS(-100) ABS('100')
100 100 100

图6.1 ABS函数



SQL> SELECT MOD(5,2),MOD(8/3,5),MOD('10',5),MOD(-10,6),MOD(1,0) FROM DUAL;
MOD(5,2) MOD(8/3,5) MOD('10',5) MOD(-10,6) MOD(1,0)
1 2.6666667 0 -4 1

图6.2 MOD函数



3) SIGN(n)函数。返回参数n的符号。正数返回1，0返回0，负数返回-1。但如果n为BINARY_FLOAT或BINARY_DOUBLE类型时，n>=0或者n=NaN函数会返回1。示例脚本如下：

```
SELECT SIGN('9'), SIGN(-9), SIGN(0.00), SIGN(-2*'9') FROM DUAL;
```

【执行效果】

执行效果见图6.3。

6.1.2 三角函数

COS(n)函数。用于返回参数n的余弦，n为弧度表示的角度。示例脚本如下：

```
SELECT COS(3.1415926), COS('3.1415926') FROM DUAL;
```

【执行效果】

执行效果见图6.4。



图6.3 SIGN函数



图6.4 COS函数

与此类函数类似的还有如下几个。

- ☐ ACOS(n)：返回n的反余弦值。
- ☐ COSH(n)：返回n的双曲余弦值。
- ☐ SIN(n)：返回n的正弦值。
- ☐ SINH(n)：返回n的双曲正弦值。
- ☐ ASIN(n)：返回n的反正弦值。
- ☐ TAN(n)：返回n的正切值。
- ☐ TANH(n)：返回n的双曲正切值。
- ☐ ATAN(n)：返回n的反正切值。

6.1.3 返回以指定数值为准整数的函数

1) CEIL(n)函数。其返回结果是大于等于输入参数的最小整数。该输入参数要求是十进制数值类型，或可以隐式地转换成数值的类型，可以是非整数。示例脚本如下：

```
SELECT CEIL(10), CEIL('10.5'), CEIL(-10.2) FROM DUAL;
```

【执行效果】

执行效果见图6.5。

从示例中可以看出，当参数为-10.2时返回了-10，该结果符合正常的数学逻辑，读者平时使用需要留意。

2) FLOOR(n)函数。其返回结果是小于或等于参数的最大整数。该函数输入参数要求是十进制数值类型，或可以隐式地转换成数值的类型。可以是非整数。同CEIL函数相反。示例脚本如下：

```
SELECT FLOOR(10), FLOOR('10.5'), FLOOR(-10.2) FROM DUAL;
```

【执行效果】

执行效果见图6.6。



图6.5 CEIL函数



图6.6 FLOOR函数

6.1.4 指数、对数函数

1) SQRT(n)函数。该函数返回n的平方根。n为数字类型的时候不能为负数，将返回一个实数，当n为BINARY_FLOAT或BINARY_DOUBLE类型时，n<0将返回Nan。示例脚本如下：

```
SELECT SQRT(100), SQRT('53.9') FROM DUAL;
```

【执行效果】

执行效果见图6.7。

2) POWER(n2,n1)函数。利用该函数可以得到n2的n1次幂的结果。这两个参数为任意数值，但如果n2为负数，那么n1必须为整数。示例脚本如下：

```
SELECT POWER(5,2), POWER('5',2), POWER(5.5,2.5), POWER(-5,2), 5*5 FROM DUAL;
```

【执行效果】

执行效果见图6.8。



图6.7 SQRT函数



图6.8 POWER函数

与其相近的函数有：

EXP(n)函数。表示返回e的n次幂，e为数学常量，e = 2.71828183....

3) LOG(n1,n2)函数。该函数可以返回以n1为底n2的对数，n1是除1和0以外的任意正数，n2为正数。示例脚本如下：

```
SELECT LOG(10,100), LOG(10.5, '100'), POWER(10,2) FROM DUAL;
```

【执行效果】

执行效果见图6.9。



图6.9 LOG函数



读书笔记



与其相近的函数有：

LN(n)函数，表示返回n的自然对数。n要求大于0。

6.1.5 四舍五入截取函数

1) ROUND (for number) 函数。该函数的具体原型是ROUND(n,integer)。它将数值n四舍五入成第二个参数指定的形式的十进制数。参数integer要求是整数，如果不是整数，那么它将被自动截取为整数部分。当integer为正整数时，表示n被四舍五入为integer位小数。如果该参数为负数，则n被四舍五入至小数点向左integer位。示例脚本如下：

```
SELECT ROUND(100.23456,4),ROUND( 100.23456,2.56), ROUND(155.23456,-2) FROM DUAL;
```

【执行效果】

执行效果见图6.10。



图6.10 ROUND函数

2) TRUNC (for number) 函数。该函数的具体原型是TRUNC(n,integer)。它把数值n根据integer的值进行截取，截取时和integer的正负有关。参数integer要求是整数，如果不是整数，那么它将被自动截取为整数部分。当integer为正整数时，表示n将截取到integer位小数；如果integer为负数，则截取到小数点左第integer位，被截取部分用0代替。示例脚本如下：

```
SELECT TRUNC(100.23456,4),TRUNC(100.23456,2.56),TRUNC(155.23456,-2),TRUNC(155.23456) FROM DUAL;
```

【执行效果】

执行效果见图6.11。



图6.11 TRUNC函数

6.2 字符型函数

以下函数全都接收的是字符族类型的参数(CHR除外)，其中大部分返回字符类型数据，小部分返回数字类型数据。

6.2.1 ASCII码与字符转换函数

1) CHR(n[USING NCHAR_CS])函数。根据相应的字符集，把给定的ASCII 码转换为字符。



USING NCHAR_CS指明字符集。以下示例用默认字符集，示例脚本如下：

```
SELECT CHR(65)||CHR(66)||CHR(67) ABC, CHR(54678) FROM DUAL;
```

【执行效果】

执行效果见图6.12。

2) ASCII(char)函数。返回参数首字母的ASCII码值。与CHR函数相反。参数char的类型可以是CHAR、VARCHAR2、NCHAR或NVARCHAR2。该返回值总是以用户使用的字符集为基础的，如果用户的数据库字符集是7位的ASCII值，那就得到一ASCII码值。示例脚本如下：

```
SELECT ASCII('明'), ASCII('Adb'), ASCII('ABC') FROM DUAL;
```

【执行效果】

执行效果见图6.13。



图6.12 CHR函数



图6.13 ASCII函数

6.2.2 获取字符串长度函数

LENGTH函数。该函数可以得到指定字符串的长度，返回类型是数字。同样的，LENGTH函数也具有扩展形式，具体结构是{[LENGTH]| [LENGTHB]| [LENGTHC]| [LENGTH2]| [LENGTH4]} (char)，各项参数含义可以参考前面介绍过的函数，这里不再过多解释，其中char是参数。具体的示例脚本如下，这里仅以LENGTH操作为例。

```
SELECT LENGTH('ABCDEFGHI') FROM DUAL;
```

【执行效果】

执行效果见图6.14。



图6.14 LENGTH函数

6.2.3 字符串截取函数

SUBSTR函数。该函数提供截取字符串的功能，而且该函数有很多的扩展形式，其具体语句结构是 {[SUBSTR]| [SUBSTRB]| [SUBSTRC]| [SUBSTR2]| [SUBSTR4]} (char, position[, substring_length])。各参数表示含义如下：

- ☐ SUBSTR：以字符为单位。
- ☐ SUBSTRB：以字节为单位。
- ☐ SUBSTRC：以unicode字符为单位。
- ☐ SUBSTR2：以UCS2代码点为单位。
- ☐ SUBSTR4：以UCS4代码点为单位。
- ☐ char：原始字符串。
- ☐ position：要截取字符串的开始位置。初始为1，如果该值为负数，则表示从char的右边算起。
- ☐ substring_length：截取的长度。

具体的示例脚本如下，这里仅以SUBSTR操作为例。

```
SELECT SUBSTR('ABCDE我FGHI',5,2),SUBSTR('ABCDE我FGHI',-5,2) FROM DUAL;
```

【执行效果】

执行效果见图6.15。



图6.15 SUBSTR函数

6.2.4 字符串连接函数

CONCAT(char1,char2)函数。该函数连接两个参数并返回。char2将连接到char1的尾部。效果和连接符“||”相似。参数类型可以是CHAR、VARCHAR2、NCHAR、NVARCHAR2、CLOB、NCLOB。示例脚本如下：

```
SELECT CONCAT('我的','测试'),'我的'||'测试' FROM DUAL;
```

【执行效果】

执行效果见图6.16。



图6.16 CONCAT函数

6.2.5 字符串搜索函数

INSTR函数。该函数可以让我们在指定的字符串中搜索是否存在另一个字符串。

其具体语句结构是{[INSTR]|[INSTRB]|[INSTRC]|[INSTR2]|[INSTR4]} (string, substring [,position[,occurrence]]). 该函数也具有扩展形式，各项参数表示含义如下：

- INSTR：以字符为单位。
- INSTRB：以字节为单位。
- INSTRC：以unicode字符为单位。
- INSTR2：以UCS2代码点为单位。
- INSTR4：以UCS4代码点为单位。
- string：待搜索的字符串。
- substring：要搜索的字符串。
- position：搜索的开始位置，默认为1，表示字符串左边第一个位置；如果为负数，则表示字符串的右边位置为起始位置。
- occurrence：substring第几次出现，默认是1。

具体的示例脚本如下，这里仅以INSTR操作为例。

```
SELECT INSTR('this is a 测试','测'),INSTR('this is a 测试','s',-1) FROM DUAL;
```

【执行效果】

执行效果见图6.17。



图6.17 INSTR函数



6.2.6 字母大小写转换函数

1) UPPER(char)函数。该函数将指定的参数全部转换成大写字母。参数类型可以是CHAR、VARCHAR2、NCHAR、NVARCHAR2、CLOB、NCLOB。示例脚本如下：

```
SELECT UPPER('c'),UPPER('abcd'),UPPER('this is a test') FROM DUAL;
```

【执行效果】

执行效果见图6.18。

2) LOWER(char)函数。该函数将指定的参数全部转换成小写字母。参数类型可以是CHAR、VARCHAR2、NCHAR、NVARCHAR2、CLOB、NCLOB。示例脚本如下：

```
SELECT LOWER('A'),LOWER('ABCD'),LOWER('THIS IS A TEST') FROM DUAL;
```

【执行效果】

执行效果见图6.19。



图6.18 UPPER函数



图6.19 LOWER函数

3) INITCAP(char)函数。该函数参数的所有单词首字母转换成大写字母。参数类型可以是CHAR、VARCHAR2、NCHAR、NVARCHAR2。

示例脚本如下：

```
SELECT INITCAP('this is a test') FROM DUAL;
```

【执行效果】

执行效果见图6.20。



图6.20 INITCAP函数

6.2.7 带排序参数的字母大小写转换函数

1) NLS_INITCAP(char[,nlsparam])函数。将指定参数的第一个字母转换成大写。nlsparam参数为可选参数，其设置可以到NLS_DATABASE_PARAMETERS表中查询。这两个参数类型可以是CHAR、VARCHAR2、NCHAR、NVARCHAR2。如果该函数没有nlsparam参数，则它和INITCAP函数一样。示例脚本如下：

```
SELECT NLS_INITCAP('a test'),NLS_INITCAP('my test','NLS_SORT = SCHINESE_STROKE_M') FROM DUAL;
```

【执行效果】

执行效果见图6.21。



图6.21 NLS_INITCAP函数

其中'NLS_SORT = SCHINESE_STROKE_M'指按笔画、部首排序。

2) NLS_UPPER(char[,nlsparam]) 函数。将指定参数变成大写。nlsparam参数同NLS_INITCAP函数设置。示例脚本如下：

```
SELECT NLS_UPPER('this is a test','NLS_SORT= SCHINESE_PINYIN_M')
FROM DUAL;
```

【执行效果】

执行效果见图6.22。



图6.22 NLS_UPPER函数

参数中'NLS_SORT= SCHINESE_PINYIN_M'表示按拼音排序。

3) NLS_LOWER(char[,nlsparam]) 函数。将指定参数转换成小写。nlsparam参数同NLS_INITCAP函数设置。示例脚本如下：

```
SELECT NLS_LOWER('ABC','NLS_SORT= XGerman'),
       NLS_LOWER('THIS IS A TEST','NLS_SORT= XGerman')
FROM DUAL;
```

【执行效果】

执行效果见图6.23。



图6.23 NLS_LOWER函数

6.2.8 为指定参数排序函数

NLSSORT(char[,nlsparam]) 函数。根据nlsparam指定的方式对char进行排序。示例脚本如下：

```
SELECT * FROM PRODUCTINFO
ORDER BY NLSSORT(PRODUCTNAME,'NLS_SORT= SCHINESE_PINYIN_M')
```

【执行效果】

执行效果见图6.24。

该示例查询数据根据PRODUCTNAME字段按拼音排序。



SQL: PL/SQL

```
SQL> SELECT * FROM PRODUCT ORDER BY NLSORT(PRODUCTNAME, 'NLS_SORT = SCHINESE_FIMV16_N');
```

PRODUCT	PRODUCTNAME	PRODUCTPRICE	QUANTITY	CATEGORY	DESPERATIO	ORIGIN
02-00001	笔记本电脑	8	0	0		中国
02-00002	笔记本电脑	10000	0	0		中国
02-00003	笔记本电脑	11000	20	01-00001	0.000000	中国
02-00004	笔记本电脑	12000	0	02-00000	0.000000	中国
02-00005	笔记本电脑	20000	10	01-00002	0.000000	中国
02-00006	笔记本电脑	21000	0	01-00003	0.000000	中国
02-00007	笔记本电脑	4000	100	01-00004	0.000000	中国
02-00008	笔记本电脑	5000	100	01-00005	0.000000	中国
02-00009	笔记本电脑	70000	20	01-00006	0.000000	日本

已选择9行。

SQL>

图6.24 NLSORT函数

6.2.9 替换字符串函数

REPLACE 函数。函数具体语法结构是 REPLACE(char,search_string[,replacement_string])，是一个替换字符串的函数。函数中有三个参数，具体代表的含义如下：

- char：表示搜索的目标字符串。
- search_string：在目标字符串中要搜索的字符串。
- replacement_string：该参数可选，用它可替代被搜索到的字符串，如果该参数不用，则表示从 char 参数中删除 search_string 字符串。

SQL: PL/SQL

```
SQL> SELECT REPLACE('this is a test','tes','resul') FROM DUAL;
```

REPLACE('this is a test','tes','resul')
this is a resul

图6.25 REPLACE函数

具体的示例脚本如下：

```
SELECT REPLACE('this is a test','tes','resul') FROM DUAL;
```

【执行效果】

执行效果见图6.25。

6.2.10 字符串填充函数

1) RPAD函数。函数具体语法结构是 RPAD(expr1,n[,expr2])，该函数功能是在字符串 expr1 的右边用字符串 expr2 填充，直到整个字符串长度为 n 时为止。如果 expr2 不存在，则以空格填充。具体的示例脚本如下：

```
SELECT RPAD('test',8,'*rpad'),RPAD('test',15,'*rpad'),RPAD('test',4,'*rpad') FROM DUAL;
```

【执行效果】

执行效果见图6.26。

2) LPAD函数。函数具体语法结构是 LPAD(expr1,n[,expr2])，该函数功能是在字符串 expr1 的左边用字符串 expr2 填充，直到整个字符串长度为 n 时为止。如果 expr2 不存在，则以空格填充。具体的示例脚本如下：

```
SELECT LPAD('test',8,'*rpad'),LPAD('test',15,'*rpad'),LPAD('test',4,'*rpad') FROM DUAL;
```



【执行效果】

执行效果见图6.27。



图6.26 RPAD函数



图6.27 LPAD函数

需要注意的是，expr2总是从左到右填充，可以见示例中LPAD('test', 8, '*rpadd')部分的执行效果。

6.2.11 删除字符串首尾指定字符的函数

1) TRIM函数。该函数将删除指定的前缀或尾随的字符，默认删除空格。其具体语法结构是TRIM([LEADING|TRAILING|BOTH][trim_character FROM] trim_source)，各参数介绍如下：

- LEADING：删除trim_source的前缀字符。
- TRAILING：删除trim_source的后缀字符。
- BOTH：删除trim_source的前缀和后缀字符。
- trim_character：删除的指定字符，默认删除空格。
- trim_source：被操作的字符串。

具体的示例脚本如下：

```
SELECT TRIM(TRAILING 't' FROM 'test'), TRIM(' test ') FROM DUAL;
```

【执行效果】

执行效果见图6.28。

2) RTRIM(char[,set])函数。与RPAD函数相反，该函数会提供将char右边出现在set中的字符删除掉。如果set没有，则默认删除空格。具体的示例脚本如下：

```
SELECT RTRIM('test '), RTRIM('test*ffs', 'fs*') FROM DUAL;
```

【执行效果】

执行效果见图6.29。

3) LTRIM(char[,set])函数。与RTRIM函数相似，该函数会提供将char左边出现在set中的字符删除掉。如果set没有，则默认删除空格。具体的示例脚本如下：



图6.28 TRIM函数

```
SELECT LTRIM(' ftest', ' f') FROM DUAL;
```

【执行效果】

执行效果见图6.30。



图6.29 RTRIM函数



图6.30 LTRIM函数

6.2.12 字符集名称和ID互换函数

1) NLS_CHARSET_ID(string)函数。该函数可以得到字符集名称对应ID。string表示字符集名称。示例脚本如下：

```
SELECT NLS_CHARSET_ID('US7ASCII') FROM DUAL;
```

【执行效果】

执行效果见图6.31。

2) NLS_CHARSET_NAME(number)函数。该函数可以根据字符集ID得到对应名称。number表示字符集ID。示例脚本如下：

```
SELECT NLS_CHARSET_NAME(1) FROM DUAL;
```

【执行效果】

执行效果见图6.32。



图6.31 NLS_CHARSET_ID函数



图6.32 NLS_CHARSET_NAME函数

以上是日常开发中经常用到的字符类型函数。

6.3 日期型函数

日期类型的函数操作日期、时间类型的相关数据，并返回日期或数字类型的数据。

6.3.1 系统日期、时间函数

1) SYSDATE函数。该函数没有参数，可以得到系统的当前日期，是很常用的函数。下面示例演示了将得到的系统时间进行格式化。示例脚本如下：

```
SELECT TO_CHAR(SYSDATE, 'YYYY-MM-DD HH24:MI:SS') FROM DUAL;
```

**【执行效果】**

执行效果见图6.33。

脚本中使用了TO_CHAR()函数，该函数具体用法后面会介绍，严格来说它属于转换函数。

2) SYSTIMESTAMP函数。该函数没有参数，返回系统时间，该时间包含时区信息，精确到微秒。返回类型为带时区信息的TIMESTAMP类型。示例脚本如下：

```
SELECT SYSTIMESTAMP FROM DUAL;
```

【执行效果】

执行效果见图6.34。



图6.33 SYSDATE函数



图6.34 SYSTIMESTAMP函数

该函数可以用于返回远端数据库服务器的时间。

6.3.2 得到数据库时区函数

DBTIMEZONE函数。该函数没有参数，返回数据库时区。示例脚本如下：

```
SELECT DBTIMEZONE FROM DUAL;
```

【执行效果】

执行效果见图6.35。



图6.35 DBTIMEZONE函数

6.3.3 为日期加上指定月份函数

1) ADD_MONTHS(date, integer)函数。该函数

将返回在指定的日期上加一个月份数后的日期。各参数具体含义如下：

□ date：指定的日期。

□ integer：要加的月份数，该值如果为负数，则表示减去的月份数。

该函数有些地方需要注意，当指定的日期是月的最后一天时，最后函数返回的结果也将是新的月的最后一天。而如果新的月份比指定日期月份的天数少，则函数将自动回调有效日期。示例脚本如下：

```
SELECT
  TO_CHAR(ADD_MONTHS(TO_DATE('2009-9-15', 'YYYY-MM-DD'), 1), 'YYYY-MM-DD'),
  TO_CHAR(ADD_MONTHS(TO_DATE('2009-9-30', 'YYYY-MM-DD'), 1), 'YYYY-MM-DD'),
  TO_CHAR(ADD_MONTHS(TO_DATE('2010-1-30', 'YYYY-MM-DD'), 1), 'YYYY-MM-DD')
FROM DUAL;
```

【执行效果】

执行效果见图6.36。

脚本中使用了TO_DATE()函数，该函数具体用法后面会介绍，它属于转换函数。

2) SESSIONTIMEZONE函数。该函数没有参数，可以返回当前会话的时区。示例脚本如下：



```
SELECT SESSIONTIMEZONE FROM DUAL;
```

【执行效果】

执行效果见图6.37。



图6.36 ADD_MONTHS函数



图6.37 SESSIONTIMEZONE函数

6.3.4 返回指定月份最后一天函数

LAST_DAY(date)函数。该函数返回参数指定日期对应月份的最后一天。示例脚本如下：

```
SELECT LAST_DAY(SYSDATE) FROM DUAL;
```

【执行效果】

执行效果见图6.38。

6.3.5 返回指定日期后一周的日期函数

NEXT_DAY(date, char)函数。该函数返回当前日期向后的一周char的对应日期，char表示的是星期几，全称和缩写都允许。但必须有效。示例脚本如下：

```
SELECT SYSDATE, NEXT_DAY(SYSDATE, '星期一') FROM DUAL;
```

【执行效果】

执行效果见图6.39。



图6.38 LAST_DAY函数



图6.39 NEXT_DAY函数

6.3.6 返回会话所在时区当前日期函数

CURRENT_DATE函数。该函数得到会话时区的当前日期。示例脚本如下：

```
SELECT SESSIONTIMEZONE, TO_CHAR(CURRENT_DATE, 'YYYY-MM-DD HH24:MI:SS') FROM DUAL;
```

【执行效果】

执行效果见图6.40。

该脚本查询的是第8时区当前的系统时间，如果更改会话为第6时区，则可以利用如下脚本：

```
ALTER SESSION SET TIME_ZONE = '-6:0';
```

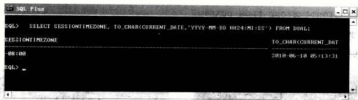


图6.40 CURRENT_DATE函数

6.3.7 提取指定日期特定部分的函数

EXTRACT (datetime)函数。该函数可以从指定的时间当中提取到指定的日期部分，例如从给定的日期得到年、月、分等。示例脚本如下：

```
SELECT
EXTRACT(YEAR FROM SYSDATE ) YEAR,
EXTRACT(MINUTE FROM TIMESTAMP '2010-6-18 12:23:10 ' ) MIN,
EXTRACT(SECOND FROM TIMESTAMP '2010-6-18 12:23:10 ' ) SEC
FROM DUAL;
```

【执行效果】

执行效果见图6.41。



图6.41 EXTRACT函数

6.3.8 得到两个日期之间的月份数

MONTHS_BETWEEN(date1,date2)函数。该函数返回date1和date2之间的月份数。函数两个参数都为日期型数据。当date1>date2时，如果两个参数表示日期是某月中的同一天，或它们都是某月中的最后一天，则该函数返回一整型数；否则，将返回小数。当date1<date2时，则返回一负值。示例脚本如下：

```
SELECT
MONTHS_BETWEEN(TO_DATE('2010-7-1','YYYY-MM-DD'), TO_DATE('2010-6-1','YYYY-MM-DD')) ONE,
MONTHS_BETWEEN(TO_DATE('2010-5-31','YYYY-MM-DD'), TO_DATE('2010-4-30','YYYY-MM-DD')) TWO,
MONTHS_BETWEEN(TO_DATE('2010-5-31','YYYY-MM-DD'), TO_DATE('2010-9-30','YYYY-MM-DD')) THREE
FROM DUAL;
```

【执行效果】

执行效果见图6.42。



图6.42 MONTHS_BETWEEN函数

6.3.9 时区时间转换函数

NEW_TIME(date,timezone1,timezone2)函数。该函数将返回时间date在时区timezone1转换到时区timezone2的时间。示例脚本如下:

```
SELECT TO_CHAR(SYSDATE,'YYYY-MM-DD HH24:MI:SS') ONE,
       TO_CHAR(NEW_TIME(SYSDATE,'PDT','EST'),'YYYY-MM-DD HH24:MI:SS') TWO
FROM DUAL;
```

【执行效果】

执行效果见图6.43。



图6.43 NEW_TIME函数

6.3.10 日期四舍五入、截取函数

1) ROUND(date[,fmt])函数。该函数将date舍入到fmt指定的形式。如果参数fmt被省略,则date将被处理到最近的一天。示例脚本如下:

```
SELECT
       TO_CHAR(ROUND(TO_DATE('2010-5-1 21:00:00','YYYY-MM-DD HH24:MI:SS')),
       'YYYY-MM-DD HH24:MI:SS')
FROM DUAL;
```

【执行效果】

执行效果见图6.44。



图6.44 ROUND函数

2) TRUNC(date[,fmt])函数。该函数将date截取到fmt指定的形式。如果fmt省略,则截取到最近的日期。示例脚本如下:

```
SELECT
  TO_CHAR(TRUNC(TO_DATE('2010-5-1 09:00:00','YYYY-MM-DD HH24:MI:SS')),
    'YYYY-MM-DD HH24:MI:SS')
FROM DUAL;
```

【执行效果】

执行效果见图6.45。



图6.45 TRUNC函数

6.4 转换函数

转换函数可以完成不同数据类型之间的转换,是平常使用比较多的函数类型之一。本节将介绍平常使用率较高的转换函数。

6.4.1 字符串转ASCII类型字符串函数

ASCIISTR(char)函数。该函数可将任意字符集的字符串转换为数据库字符集对应的ASCII字符串。char为字符类型。示例脚本如下:

```
SELECT ASCIISTR('这是测试!') FROM DUAL;
```

【执行效果】

执行效果见图6.46。

6.4.2 二进制转十进制函数

BIN_TO_NUM(data[,data...])函数。该函数可以将二进制转换成对应的十进制。data表示二进制数,一位用“,”隔开。示例脚本如下:

```
SELECT BIN_TO_NUM(1),BIN_TO_NUM(1,0,0),BIN_TO_NUM(1,1,1) FROM DUAL;
```

【执行效果】

执行效果见图6.47。



图6.46 ASCIISTR函数



图6.47 BIN_TO_NUM函数

6.4.3 数据类型转换函数

CAST(expr as type_name)函数。该函数是进行类型转换的，可以把expr参数转换成type_name类型。基本上用于数字与字符之间以及字符与日期类型之间的转换。类型之间的转换Oracle有一套规则表，这里不列出。示例脚本如下：

```
SELECT CAST('123' AS INTEGER) AS VNR,
       CAST(123 AS VARCHAR2(8)) AS NUM,
       CAST(SYSDATE AS VARCHAR2(12)) AS DT
FROM DUAL;
```

【执行效果】

执行效果见图6.48。



图6.48 CAST函数

6.4.4 字符串和ROWID相互转换函数

1) CHARTOROWID(char)函数。该函数将字符串类型转成ROWID类型。char为待转的字符串，其类型可以是CHAR、VARCHAR2、NCHAR、NVARCHAR2，但必须符合ROWID格式，长度为18。每一条记录都有一个rowid，rowid在整个数据库中唯一，可以利用SELECT查询该字段。示例脚本如下：

```
SELECT CHARTOROWID('AAARXnAABAAVgggAB') FROM DUAL;
```

【执行效果】

执行效果见图6.49。

2) ROWIDTOCHAR(rowid)函数。该函数将行记录的ROWID转成字符串。参数rowid长度为18，所以返回结果长度18。示例脚本如下：

```
SELECT ROWIDTOCHAR(ROWID) FROM PRODUCTINFO;
```

【执行效果】

执行效果见图6.50。



图6.49 CHARTOROWID函数



图6.50 ROWIDTOCHAR函数

3) ROWIDTONCHAR(rowid)函数。同ROWIDTOCHAR(rowid)操作相同，但返回类型是NVARCHAR2。这里不再给出示例。

6.4.5 字符串在字符集间转换函数

CONVERT函数。该函数用于把字符串从一个字符集转到另一个字符集。函数的具体语法结构是CONVERT(char,dest_char_set[,source_char_set])，各参数的表示含义如下：

□ char：等待转换的字符。可以是CHAR、VARCHAR2、NCHAR、NVARCHAR2、CLOB、NCLOB类型。

□ dest_char_set：转变后的字符集。

□ source_char_set：原字符集，如果没有该参数，则默认数据库实例字符集。

示例脚本如下：

```
SELECT CONVERT('测试','US7ASCII','ZHS16GBK') FROM DUAL;
```

【执行效果】

执行效果见图6.51。



图6.51 CONVERT函数

6.4.6 十六进制字符串与RAW类型相互转换函数

1) HEXTORAW(char)函数。该函数把十六进制的字符串转换成raw类型的数据。其中参数char表示一个由十六进制字符组成的字符串。示例脚本如下：

```
SELECT HEXTORAW('4d') FROM DUAL;
```

【执行效果】

执行效果见图6.52。

2) RAWTOHEX(raw)函数。与HEXTORAW函数相反，它把raw类型表示成一个由十六进制字符串表示的串，返回VARCHAR2类型。示例脚本如下：

```
SELECT RAWTOHEX('4D') FROM DUAL;
```

【执行效果】

执行效果见图6.53。



图6.52 HEXTORAW函数



图6.53 RAWTOHEX函数



3) RAWTONHEX(raw)函数。同函数RAWTOHEX(raw)转换效果相同,不过返回的类型是NVARCCHAR2类型,而不是VARCHAR2类型。这里不再给出示例。

6.4.7 数值转换成字符型函数

1) TO_CHAR (number)函数。该函数将一个数值型参数转换成字符型数据。其具体语法结构是TO_CHAR(n[,fmt[,nlsparam]]),各参数表示含义如下:

- n: 数值型数据。
 - fmt: 要转成字符的格式。
 - nlsparam: 由该参数指定fmt的特征。通常包括小数点字符、组分分隔符、本地货币符号。
- 该函数如果想用的好需要了解多方面的知识,这里不做详细介绍。示例脚本如下:

```
SELECT TO_CHAR(16.89,'99.9'),TO_CHAR(16.89) FROM DUAL;
```

【执行效果】

执行效果见图6.54。

2) TO_CHAR (date)函数。该函数将一个日期型数据转换成一个字符型数据。它同前面介绍的同名函数一样,只不过转换的对象变化了。具体的语法结构是TO_CHAR (n[,fmt[,nlsparam]]),各参数具体含义如下:

- n: 日期类型数据。
- fmt: 要转成字符的格式。
- nlsparam: 使用的语言类型。

示例脚本如下:

```
SELECT TO_CHAR(SYSDATE,'YYYY-MM-DD'),
TO_CHAR(SYSDATE,'HH24:MI:SS'),
TO_CHAR(SYSDATE,'Month','NLS_DATE_LANGUAGE=ENGLISH')
FROM DUAL;
```

【执行效果】

执行效果见图6.55。

6.4.8 字符转日期型函数

TO_DATE函数。该函数可将字符型数据转换成日期型数据。函数的具体语法结构是TO_DATE(char[,fmt[,nlsparam]]),各参数的具体含义如下:

- char: 待转换的字符。类型可以是CHAR、VARCHAR2、NCHAR、NVARCHAR2。
- fmt: 表示转换的格式。
- nlsparam: 控制格式化时使用的语言类型。

示例脚本如下:

```
SELECT TO_CHAR(TO_DATE('2010-7-1','YYYY-MM-DD'),'MONTH') FROM DUAL;
```



图6.54 TO_CHAR (number)函数



图6.55 TO_CHAR (date)函数

**【执行效果】**

执行效果见图6.56。



图6.56 TO_DATE函数

6.4.9 字符串转数字函数

TO_NUMBER函数。该函数将字符串转成数字。语法结构是TO_NUMBER(expr[,fmt[,nlsparam]]), 各参数表示的具体含义如下:

- ☐ expr: 待转换的字符, 其类型可以是CHAR、VARCHAR2、NCHAR、NVARCHAR2。
 - ☐ fmt: 指定转换的数字格式。
 - ☐ nlsparam: 该参数指定fmt的特征。通常包括小数点字符、组分隔符、本地钱币符号。
- 示例脚本如下:

```
SELECT TO_NUMBER('2456.304', '9999.999') FROM DUAL;
```

【执行效果】

执行效果见图6.57。

6.4.10 全角转半角函数

TO_SINGLE_BYTE(char)函数。该函数将全角转为半角。char的类型可以是CHAR、VARCHAR2、NCHAR、NVARCHAR2。示例脚本如下:

```
SELECT TO_SINGLE_BYTE('THIS IS A TEST') FROM DUAL;
```

【执行效果】

执行效果见图6.58。



图6.57 TO_NUMBER函数



图6.58 TO_SINGLE_BYTE函数

6.5 NULL函数

NULL函数是用来处理空值时比较好的选择。本节介绍几个常用的空值处理函数。

6.5.1 返回表达式为NULL的函数

COALESCE(expr)函数。返回列表中第一个不为null的表达式。如果都为null,则返回一个null。示例脚本如下:

```
SELECT COALESCE(NULL,9-9,NULL) FROM DUAL;
```

【执行效果】

执行效果见图6.59。



图6.59 COALESCE函数

6.5.2 排除指定条件函数

LNNVL(condition)函数。该函数可以得到除了condition要求条件之外的数据,包括NULL的条件,通常用于WHERE条件中。下面的示例将得到PRODUCTINFO表中数量低于70的产品,并包含数量为NULL的数据。示例脚本如下:

```
SELECT* FROM PRODUCTINFO WHERE LNNVL(QUANTITY>=70);
```

【执行效果】

执行效果见图6.60。

PRODUCTID	PRODUCTNAME	PRODUCTPRICE	QUANTITY	CATEGORY	DESCRIPTION	ORIGIN
02-000400001	双片LCD-661000	7000	20	01-0000100001		日本
02-000500001	片-SUPER VIEW	1100	25	01-0000100002		日本
02-000600002	三片-FLM661000	3000	12	01-0000100002		日本
02-000600001	三片-6600010000	2000	22	01-0000100001		日本
02-000700001	两片-6600010000	0	0			日本
02-0008100001	三片-6600010000	5.0	5.0	01-0000100001		日本
02-000900002	三片-6600010000	0	0			日本

图6.60 LNNVL函数

6.5.3 替换NULL值函数

1) NVL(expr1,expr2)函数。替换NULL值,表示如果expr1为NULL值,则返回expr2的值,否则返回expr1的值。该函数要求两个参数类型一致,至少相互间能进行隐式的转换,否则会提示出错。

例如,下面的示例将查询PRODUCTINFO表中的数量,如果记录中有该字段为空的,则用0替换。相关脚本如下:

```
SELECT PRODUCTNAME,NVL(QUANTITY,0),CATEGORY FROM PRODUCTINFO;
```

【执行效果】

执行效果见图6.61。

PRODUCTNAME	NVL(QUANTITY,0)	CATEGORY
显示器 LCD-46G1000	20	01-0000100001
显示器 LCD-46G1000	20	01-0000100002
显示器 LCD-46G1000	12	01-0000100002
显示器 LCD-46G1000	111	01-0000100002
显示器 LCD-46G1000	129	01-0000100001
显示器 LCD-46G1000	22	01-0000100001
显示器 LCD-46G1000	0	0
显示器 LCD-46G1000	0	0
显示器 LCD-46G1000	0	0

图6.61 NVL函数

2) NVL2(expr1,expr2,expr3)函数。该函数同NVL类似，不同的是当expr1为NULL时，函数返回expr3的值；当expr1不为空时，则返回expr2的值。

6.6 集合函数

集合函数经常配合GROUP BY或HAVING子句使用，当然它们也可以单独使用。该类型的函数中除了COUNT函数都会忽略列值为NULL的数据。

6.6.1 求平均值函数

AVG([distinct|all]expr)函数。该函数可求取指定列的平均值，表示某组的平均值，返回数值类型。各参数表示的具体含义如下：

- ☐ distinct：去除重复的值。
- ☐ all：表示所有的值，包括重复的值，也是默认值。
- ☐ expr：表达式。只能是数值类型。

1) 这里以PRODUCTINFO的PRODUCTPRICE字段为例，演示最简单的使用方法。该表的结构可以参考第4章。测试脚本如下：

```
SELECT AVG(ALL PRODUCTPRICE+100) FROM PRODUCTINFO;
```

【执行效果】

执行效果见图6.62。

AVG(ALL PRODUCTPRICE+100)
2151

图6.62 AVG函数

2) 使用该函数时, WHERE条件子句中可以使用条件。例如, 与GROUP BY子句一起使用, 或只用某个范围内的值。下面演示使用GROUP BY子句查询各类产品的平均价格。演示脚本如下:

```
SELECT AVG(ALL PRODUCTPRICE) FROM PRODUCTINFO GROUP BY CATEGORY;
```

【执行效果】

执行效果见图6.63。

3) 第三个示例将演示价格大于2000的所有产品的平均价格。演示脚本如下:

```
SELECT AVG(ALL PRODUCTPRICE) FROM PRODUCTINFO WHERE PRODUCTPRICE > 2000;
```

【执行效果】

执行效果见图6.64。



图6.63 AVG与GROUP BY子句一起使用



图6.64 AVG与WHERE一起使用

该函数的返回值的精度与列的数据类型和是否有小数有关, 希望读者注意。

6.6.2 求记录数量函数

COUNT([*]|[distinct][all]expr)函数。该函数可以用来计算记录的数量或某列的个数。函数中必须指定列名, 或全选使用星号。其中各参数表示的含义如下:

- ☐ *: 表示计算所有记录。
- ☐ distinct: 表示去除重复的记录。
- ☐ all: 代表所有的, 是默认选项。
- ☐ expr: 要计算的對象, 通常是表的列。

下面以几个示例来演示COUNT函数的使用方法。

1) 查询PRODUCTINFO表的所有记录数。演示脚本如下:

```
SELECT COUNT(*) FROM PRODUCTINFO;
```

【执行效果】

执行效果见图6.65。

2) 查询PRODUCTINFO表的PRODUCTPRICE字段低于3000的不重复的记录数。演示脚本如下:

```
SELECT COUNT(DISTINCT PRODUCTPRICE) FROM PRODUCTINFO WHERE PRODUCTPRICE < 3000;
```

【执行效果】

执行效果见图6.66。



图6.65 COUNT函数计算所有记录数

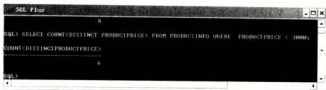


图6.66 COUNT函数计算不重复记录数

6.6.3 返回最大、最小值函数

MAX([distinct]all)expr)函数。该函数可以返回指定列中的最大值，通常都用在WHERE子句中的子查询。其中各参数表示的含义如下：

☐ distinct：表示去除重复的记录。

☐ all：代表所有的，是默认选项。

☐ expr：表的列。

查询PRODUCTINFO表中价格最高的记录，演示脚本如下：

```
SELECT * FROM PRODUCTINFO
WHERE PRODUCTPRICE = (SELECT MAX(PRODUCTPRICE) FROM PRODUCTINFO);
```

【执行效果】

执行效果见图6.67。



图6.67 MAX函数

同该函数效果相反但用法一致的有MIN([distinct]all)expr)函数，此函数获取指定列中的最小值。这里不再给出示例。

6.6.4 求和函数

SUM([distinct]all)expr)函数。该函数不同于COUNT函数，它分组计算指定列的和，如果不使用分组，则函数默认把整个表作为一组。各参数代表含义如下：

☐ distinct：表示去除重复的记录。

☐ all：代表所有的，是默认选项。

☐ expr：表的列。

下面示例将演示利用该函数计算不同类型产品的数量和。演示脚本如下：

```
SELECT SUM(ALL QUANTITY),CATEGORY FROM PRODUCTINFO GROUP BY CATEGORY;
```

【执行效果】



执行效果见图6.68。



图6.68 SUM函数

6.7 其他函数

前面介绍了单行函数以及集合函数，下面介绍几个不属于前面类型的函数，主要是系统环境和编码方面的函数。

6.7.1 返回登录名函数

USER函数。该函数返回当前会话的登录名。演示脚本如下：

```
SELECT USER FROM DUAL;
```

【执行效果】

执行效果见图6.69。

6.7.2 返回会话以及上下文信息函数

1) USERENV(parameter)函数。返回当前会话的信息。例如，当参数为Language时可以返回当前会话对应的语言、字符集等。SESSIONID可返回当前会话ID。ISDBA可返回当前用户是否DBA。

示例将演示返回当前用户是否DBA用户。脚本如下：

```
SELECT USERENV('ISDBA') FROM DUAL;
```

【执行效果】

执行效果见图6.70。



图6.69 USER函数



图6.70 USERENV函数

2) SYS_CONTEXT(namespace,parameter)函数。该函数可以得到Oracle已经创建的context，名为USERENV的属性对应值。

示例将得到当前会话对应的用户名。脚本如下：

```
SELECT SYS_CONTEXT('USERENV','SESSION_USER') SESSION_USER FROM DUAL;
```

【执行效果】

执行效果见图6.71。

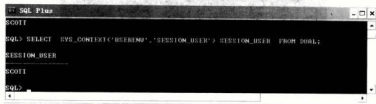


图6.71 SYS_CONTEXT函数

6.7.3 表达式匹配函数

DECODE函数。该函数的具体语法是DECODE(expr,search,result[,search1,result1][,default])。该函数的执行过程是，当expr符合条件search时就返回result的值，该过程可以重复多个，如果最后没有匹配的结果，可以返回默认值default，注意它是一一对一的匹配过程。

下面的示例将演示PRODUCTINFO中产品数量多于100的就显示“充足”，少于或等于100则显示“不足”。脚本如下：

```
SELECT PRODUCTNAME,QUANTITY,  
DECODE(SIGN(QUANTITY-100),1,'充足',-1,'不足',0,'不足') FROM PRODUCTINFO
```

【执行效果】

执行效果见图6.72。



图6.72 DECODE函数

从函数中可以看出它只能和单一的条件匹配，如果要想得到某个范围，可以利用其他的函数做辅助。

6.8 小结

Oracle的内置函数对开发人员快速处理数据有很大的帮助，如果Oracle不为我们提供内置



函数,那么相同的函数需要开发人员自己完成,这将是一件耗时耗力的工作。不过学习这些知识以后,用户即可很方便地使用内置函数完成复杂的业务。本章主要介绍了如下类型的函数:

- ☐ 数值类型函数:主要包括ABS、MOD、SIGN、POWER、SQRT等。
- ☐ 字符类型函数:主要包括LENGTH、SUBSTR、UPPER、LOWER、TRIM等。
- ☐ 日期类型函数:主要包括SYSDATE、ADD_MONTHS、LAST_DAY、NEXT_DAY、ROUND、TRUNC等。
- ☐ 转换函数:主要包括CAST、TO_CHAR、TO_DATE、TO_NUMBER等。
- ☐ NULL函数:主要包括COALESCE、NVL、NVL2等。
- ☐ 集合函数:主要包括AVG、COUNT、MAX、MIN、SUM等。

6.9 习题

一、填空题

- 说出常用的六种数值类型的函数: _____、_____、_____、_____、_____、_____。
- 可以用来截取字符串的函数是_____。
- 用_____函数可以得到系统的当前日期。

二、简答题

- 查询PRODUCTINFO表中产品价格低于平均值的记录。
- 查询PRODUCTINFO表产品的价格,根据价格的高低输出如下形式:

产品	价格	图表
夏普LCD-46G100A	7000	*****

第7章 PL/SQL基础

本章中将介绍PL/SQL的一些基础知识,让读者对PL/SQL概念以及如何使用它有个总体的认识。本章主要介绍如下几个知识点:

- ☐ 了解什么是PL/SQL、PL/SQL的优势、PL/SQL的三个组成部分
- ☐ PL/SQL变量的使用、变量的类型
- ☐ 流程控制语句的使用,包括IF语句、LOOP语句和CASE语句
- ☐ 什么是异常,如何使用异常处理,如何自定义异常
- ☐ 什么是函数,如何自定义函数

7.1 什么是PL/SQL

PL/SQL的使用几乎贯穿了整个Oracle的学习过程,也是作为一个初级开发人员必须掌握的重要知识点。相信通过本章的学习,用户对如何利用PL/SQL进行编程有个全面的认识。

7.1.1 认识PL/SQL

结构化查询语言(Structured Query Language, SQL)是用来访问和操作关系型数据库的一种标准通用语言,它属于第四代语言(4GL),简单易学,使用它可以很方便地调用相应语句来取得结果。该语言的特点就是非过程化。也就是说,使用的时候不用指明执行的具体方法和途径,即不用关注任何的实现细节。但这种语言也有一个问题,就是在某些情况下满足不了复杂业务流程的需求,这就是第四代语言的不足之处。Oracle中的PL/SQL语言正是为了解决这一问题,PL/SQL属于第三代的语言(3GL),也就是过程化的语言,同Java、C#一样可以关注细节,用它可以实现复杂的业务逻辑,是数据库开发人员的利器。

PL/SQL (Procedural Language/Structured Query Language) 是Oracle公司在标准SQL语言基础上进行扩展而形成的一种可以在数据库上进行设计编程的语言,通过Oracle的PL/SQL引擎执行。PL/SQL完全可以像Java语言一样实现逻辑判断、条件循环以及异常处理等,这是标准的SQL很难办到的事情。由于它的基础是标准的SQL语句,这就使得数据库开发人员能快速地掌握并运用,相信这也是Oracle开发人员喜爱它的另一个重要原因。总的来说,PL/SQL有以下几个特点:

- ☐ 支持事务控制和SQL数据操作命令。
- ☐ 它支持SQL的所有数据类型,并且在此基础上扩展了新的数据类型,也支持SQL的函数以及运算符。
- ☐ PL/SQL可以存储在Oracle服务器中。
- ☐ 服务器上的PL/SQL程序可以使用权限进行控制。
- ☐ Oracle有自己的DBMS包,可以处理数据的控制和定义命令。

7.1.2 PL/SQL的优势

由于PL/SQL语言是从SQL语言扩展而来，所以PL/SQL除了支持SQL数据类型和函数外，同时也支持Oracle对象类型。除此之外，同传统的SQL语言相比PL/SQL有以下几个优点：

(1) 可以提高程序的运行性能

标准的SQL被执行时，只能一条一条地向Oracle服务器发送。假如完成一个业务逻辑需要几条甚至几十条SQL语句，那么在这个过程中，客户端会几十次地连接数据库服务器，而连接数据库本身是一个很耗费资源的过程，当这个业务被完成时，会浪费大量的资源在网络连接上。

如果此时换用PL/SQL语句，结果则不一样了。PL/SQL的语句块可以包含多条SQL语句，而语句块可以嵌入到程序中，甚至可以存储到Oracle服务器上。这样用户只需要连接一次数据库就可以把需要的参数传递过去，其他的部分将在Oracle服务器内部执行完成，然后返回最终的结果。这样就大大地节省了网络资源的开销。图7.1描述了PL/SQL同标准的SQL语句多次访问数据库的不同。

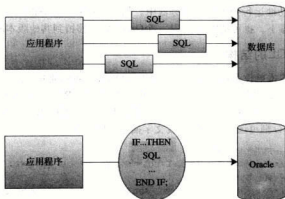


图7.1 普通SQL多条执行同PL/SQL的差异

(2) 可以使程序模块化

在程序块中可以实现一个或几个功能。例如，当想把一个动物的模型存到数据库里时，可能涉及几张表，如果使用标准的SQL完成该功能需要多条语句，而如果使用块，则可以把对多张表的操作都放到一个块内，而对外只提供一个调用方式和需要传入的参数。这对于编程开发人员是一个福音，他们不需要再写过多的SQL语句，只需要给出参数并调用一次PL/SQL的程序块就好。这种操作的优势在介绍存储过程后显得尤其明显。

使用块也可以把数据库数据同客户程序隔离开来，使得数据库表结构发生变化时，对调用者的影响减小到最低程度。

(3) 可以采用逻辑控制语句来控制程序结构

如果一个PL/SQL程序块中只能顺序地执行基本的SQL语句，那么它的意义实在有限。而实际当中PL/SQL可以利用条件或循环语句来控制程序的流程，这么做就大大地增加了PL/SQL的实用性，我们可以利用逻辑控制语句完成复杂的普通SQL语句完成不了的业务。

例如，实现如下的功能：产品很多种类，而在产品表中，产品的类型需要使用产品类型编



码替代，而不是名称，这样当输入记录的时候就需要把产品名称转换成产品编码，在PL/SQL块中就可以利用CASE语句完成判断分类，并把产品名称转换成产品编码。而这些在标准的SQL中很难实现，即便实现了也不是动态的数据。

(4) 利用处理运行时的错误信息

标准的SQL在遇到错误时会提示异常。例如增加数据，一旦有异常就会终止，但是调用者却很难快速地发现错误点在哪儿，即使发现出问题的地方也只能是告诉开发人员该语句程序本身有问题，而不是逻辑上有问题。例如，在产品表里增加数据时，数量只是要求数值型，并没有更细的要求。假如增加的数据中该字段部分是一个负数，正常来说是可以进入数据库的，但这在逻辑上是不允许的，因为没有数量为负的产品。而利用PL/SQL就可以完全避免类似的问题，我们可以利用流程拒绝这部分记录进入数据库。

利用PL/SQL还可以处理一些程序上的异常，不至于因终止SQL操作，而造成调用SQL的展示页面出现生硬的错误提示。

(5) 良好的可移植性

PL/SQL可以成功地运行到不同的服务器中。例如，从Windows的数据库服务器下移植到Linux的数据库服务器下。也可把PL/SQL从一个Oracle版本移植到其他版本的Oracle中。

7.1.3 PL/SQL的结构

PL/SQL程序的基本单位是块（block），而PL/SQL块很明确地分三部分，其中包括声明部分、执行部分和异常处理部分。其中，声明部分以DECLARE作为开始标志，执行部分用BEGIN作为开始标志，而异常处理部分则以EXCEPTION为开始标志。其中的执行部分是必需的，而其余的两个部分则可选。下面的一段文字描述了PL/SQL块的部分：

```
[DECLARE]      --声明开始关键字
                /*这里是声明部分，包括PL/SQL中的变量、常量以及类型等*/
BEGIN          --执行部分开始的标志
                /*这里是执行部分，是整个PL/SQL块的主体部分，该部分在PL/SQL块中
                必须存在，可以是SQL语句或者程序流程控制语句等*/
[EXCEPTION]    --异常开始部分的关键词
                /*这里是异常处理部分，当出现异常时程序流程可以进入此处*/
END;           --执行结束标志
```

需要记住：无论PL/SQL程序段的代码量有多少，它的基本结构只是由这三部分组成。下面将以3个示例演示PL/SQL语句块的3种情况。

【示例1】只有执行体部分的结构

该示例只有执行体部分，也就是只有“BEGIN...END;”部分，该语句块中将输出一句话，这已经是最简单的执行体了。脚本如下：

```
01 BEGIN
02   DBMS_OUTPUT.PUT_LINE('这是执行体部分...');
03 END;
04 /
```

【代码解析】

□ 第1行表示执行体开始。

□ 第2行表示输出一句话到屏幕。

□ 第3行表示执行体结束。

【执行效果】

打开SQL*Plus并执行上面一段脚本，执行结果见图7.2。

从图7.2中可以看到“SET SERVEROUTPUT ON”这么一行脚本，利用该脚本可以打开SQL*Plus的输出功能，否则将看不到输出的语句。后面的章节会频繁用到输出语句，希望读者注意这一点。

【示例2】包含声明和执行体两部分的脚本

该示例除了执行体外还有声明部分，具体操作是声明一个变量，然后为变量赋值，最后输出该变量的值。脚本如下：

```
01 DECLARE
02 v_result NUMBER(8,2);
03 BEGIN
04 v_result := 100/6;
05 DBMS_OUTPUT.PUT_LINE('最后结果是: ' || v_result);
06 END;
```

【代码解析】

□ 第1行表示声明变量的关键字。

□ 第2行表示声明一个变量，名称是v_result，数据类型为NUMBER。

□ 第4行表示给v_result赋值，这里没用动态赋值方式，读者也容易理解。

□ 第5行表示输出最后的结果。

【执行效果】

打开SQL*Plus并执行上面一段脚本，执行结果见图7.3。

通常在PL/SQL编程中，变量都是存储动态得到的数据，这种情况在下一个示例中将演示。

【示例3】包含声明、执行体和异常部分的脚本

该示例将从产品类型表CATEGORYINFO中查询产品类型“雨具”对应的产品类型编码，并把该编码存储到变量中，最后输出到屏幕。脚本如下：

```
01 DECLARE
02 v_categoryid VARCHAR2(12);
03 BEGIN
04 SELECT CATEGORYID
05 INTO v_categoryid
06 FROM CATEGORYINFO
07 WHERE CATEGORYINFO.CATEGORYNAME = '雨具';
08 DBMS_OUTPUT.PUT_LINE('雨具对应的编码是: ' || v_categoryid);
09
10 EXCEPTION
```



图7.2 只有执行体的PL/SQL块



图7.3 有声明和执行体的示例



```

11 WHEN NO_DATA_FOUND THEN
12     DBMS_OUTPUT.PUT_LINE('没有对应的编码!');
13 WHEN TOO_MANY_ROWS THEN
14     DBMS_OUTPUT.PUT_LINE('对应数据过多,请确认!');
15
16 END;
17 /

```

【代码解析】

- 第1~2行是声明部分。
- 第3~16行是执行体部分，不过异常部分是在执行体内的。
- 第4~7行表示把“雨具”对应的产品编码查询出来后放入变量v_categoryid中。
- 第10~14行属于异常处理部分。
- 第11和13行表示出现这两种异常时会输出相应的提示语句。

【执行效果】

打开SQL*Plus并执行上面一段脚本，执行效果见图7.4。

```

SQL> DECLARE
2  v_categoryid VARCHAR2(12);
3  BEGIN
4      SELECT CATEGORYID
5      INTO v_categoryid
6      FROM CATEGORYINFO
7      WHERE CATEGORYINFO.CATEGORYNAME = '雨具';
8      DBMS_OUTPUT.PUT_LINE('产品对应的编码是: ' || v_categoryid);
9
10 EXCEPTION
11 WHEN NO_DATA_FOUND THEN
12     DBMS_OUTPUT.PUT_LINE('没有对应的编码!');
13 WHEN TOO_MANY_ROWS THEN
14     DBMS_OUTPUT.PUT_LINE('对应数据过多,请确认!');
15
16 END;
17 /
产品对应的编码是: 0100010001
PL/SQL 过程已成功完成。
SQL>

```

图7.4 完整的PL/SQL块

SELECT...INTO...语句是PL/SQL特有的赋值语句，该语句表示的意思是SELECT后面列出来要查询的字段列表，INTO后面是变量名称，它表示把查询出来的值存储到变量中。这里有两个问题需要注意，就是SELECT列名顺序和INTO后面的变量名顺序要一一对应，还有就是该类型语句每次只能返回一条记录，如果返回记录超过一条或没有返回记录都会引发异常。

7.1.4 PL/SQL的基本规则

做任何事情都有规则规范，编程也是一样。好的程序会让人眼前一亮，而阅读质量差的程序会增加工作时间，降低工作效率。PL/SQL中有一些基本的规范读者应该了解，在了解这些基础之后就可以学习更加严格的编程规范，这样才能写出高质量的代码，其他开发人员阅读起来也会有一个积极开朗的心态。下面列出了初学者需要注意的规则：

- 1) PL/SQL中允许出现的字符集：

□ 字母，包括大写和小写。



□ 数字，即0~9。

□ 空格、回车符以及制表符。

□ 符号包括+、-、*、/、<、>、=、!、~、^、;、:、,、'、@、%、,、#、\$、&、_、|、(、)、[、]、{、}、?。

2) 下面列出一些PL/SQL必须遵守的要求：

□ 标识符不区分大小写。例如，TEST同Test、test是一样的。所有的名称在存储时都被修改成大写，这一点读者需要注意。

□ 标识符中只允许字母、数字、下划线，并且以字母开头。

□ 标识符最多30个字符。

□ 不能使用保留字。如与保留字同名必须使用双引号括起来。

□ 语句使用分号结束。即使多条语句在同一行，只要它们都正常结束，那么就没有问题。而且在语句块的结束标志END后面同样需要使用分号。

□ 语句的关键词、标识符、字段的名称以及表的名称等都需要空格的分隔。

□ 字符类型和日期类型需要使用单引号括起。

3) 以下是为了增强代码的可读性的相关建议，这些不是必须要遵守的，但通常情况下有些单位也可能把这些规范作为硬性要求。

□ 每行只写一条语句。

□ 全部的保留字、Oracle的内置函数、程序包以及用户定义的数据类型都用大写。

□ 所有的过程名称大写。

□ 所有的变量以及自建的过程或游标、触发器名称都要使用有意义的名称命名。

□ 命名应以“_”的连接方式，而不是用大小写混合的方式（如果只为了方便自己的阅读，可以使用大小写混合）。

□ 变量前最好加上前缀，以表示该变量的数据类型、作用范围等。

□ 每个变量都应加上注释。

□ 在重要的程序段处都应加上注释。

□ 建议3个半角空格替代TAB键进行缩进。

□ 逗号后面以及操作符的前后都应加空格。

以上只是比较基本的规则，可以提高代码的可读性，在企业的每个项目小组中会根据实际的情况做出更细的要求，甚至形成规范文档。在日常开发中应注意这些规范，形成良好的编程习惯。

7.1.5 PL/SQL中的注释

提高代码可读性的最有效的办法就是添加注释。工作中修改其他开发人员的程序是很常见的事，程序如果没有注释，就会很难理解，这样导致的结果轻则会影响修改进度，重则会影响程序开发进度，可见注释对程序是多么重要。通常情况下，程序的注释要求不能低于代码量的20%，注释也是程序的一部分，所以开发人员要养成添加注释的好习惯。

有注释的程序能使阅读者快速地了解代码实现的业务逻辑，并能理解程序的思路。这对自己和其他的开发人员都是有利的，甚至对公司来讲都是利远大于弊。Oracle为使用者提供了两种注释方式，它们分别是：

□ 单行注释：使用“_”两个短划线，可以注释掉后面的语句。



【示例4】注释使用方式示例

示例演示如何在一段程序中使用注释。脚本如下。

```

01 DECLARE
02 v_productprice VARCHAR2(12);      --平均价格
03 BEGIN
04 /*
05 利用AVG函数得到价格为3000以上的产品的平均价格
06 */
07 SELECT AVG(PRODUCTPRICE) INTO v_productprice
08 FROM PRODUCTINFO
09 WHERE PRODUCTPRICE >3000;
10 DBMS_OUTPUT.PUT_LINE('平均价格是: ' || v_productprice); --输出语句
11 END;
12 /

```

【代码解析】

□ 第2行和第10行后面部分是单行注释。

□ 第4~6行部分是多行注释。

【执行效果】

在SQL*Plus中执行脚本，效果见图7.5。

从图7.5中可以看出，注释对程序的正确执行没有任何的影响，只是提高了程序的可读性。所以，在开发程序的时候应该添加适当的注释，以方便自己或他人阅读。



图7.5 有注释的程序执行效果

7.2 PL/SQL 变量的使用

所有的编程语言中变量是使用最频繁的。PL/SQL作为一个面向过程的数据库编程语言同样少不了变量,利用变量可以把PL/SQL块需要的参数传递进来,做到动态执行程序,同时也可以利用变量在PL/SQL内部进行值的相互传递,甚至可以把值传递出去,最终返回给用户。由此可见,变量是PL/SQL不可缺少的一部分。

7.2.1 变量、常量的类型及语法

变量就是它所表示的值是可以变化的，而常量就是当初始化后，其值不可以再改变。PL/SQL是一种强类型的语言，所以当使用变量或常量的时候必须声明，否则提示错误。下面是变量和常量声明的语法介绍。

1. 变量声明语法结构

```
01 variable_name datatype
02 [
03     [ NOT NULL]
```

```
04 {:= | DEFAULT} expression
```

```
05 ];
```

【语法说明】

- ☐ variable_name: 表示变量的名称。名称可以根据读者实际情况自行定义。
- ☐ datatype: 变量的数据类型。
- ☐ 第2~5行表示可选部分。如果没有这部分, 那么只需要变量的名称以及对应的数据类型即可, 声明的时候变量可以不赋值。
- ☐ NOT NULL: 表示非空约束。
- ☐ {:= | DEFAULT}: 当使用NOT NULL属性时, 大括号里的内容为必需, 表示二选一, “:=”表示赋值, DEFAULT表示默认值。
- ☐ expression: 表示变量存储的值, 该项可以是表达式。

2. 常量声明的具体语法结构

```
01 constant_name CONSTANT datatype
```

```
02 [NOT NULL]
```

```
03 { := | DEFAULT } expression;
```

【语法说明】

- ☐ constant_name: 表示声明的常量名称。
- ☐ CONSTANT: 如果表示常量, 该项必须有, 否则表示变量。
- ☐ datatype: 常量的数据类型。
- ☐ NOT NULL: 表示常量值非空。
- ☐ {:= | DEFAULT}: 表示常量必须显式地为其赋值, 方式同变量一样。
- ☐ expression: 值或表达式。

变量和常量的语法结构相似, 其中expression表示的含义完全一样, 都可以是如下类型的表达式:

- ☐ 字符型表达式;
- ☐ 数值型表达式;
- ☐ 日期型表达式;
- ☐ 布尔型表达式;
- ☐ 直接的一个值。

变量和常量的数据类型可以概括性地分为如下三种类型:

- ☐ 标量类型变量: 单一类型, 不存在组合。
- ☐ 复合类型变量: 由几种单一类型组合而成的一个结构体。
- ☐ 引用类型变量: 使用一个其他数据项的引用。

在这三种类型中日常开发最常用的就是前两种, 下面会详细介绍这两种类型。

7.2.2 标量类型的变量

标量类型的变量是最简单类型的变量, 也是普通开发者最常用的一种变量类型, 它本身是单一的值, 不包含任何的类型组合。标量类型主要包含数值类型、字符类型、布尔类型和日期类型。还有一种比较特殊的声明变量类型的方式, 就是利用%TYPE。下面对这几种类型做详





细的介绍。

1) 数值类型，主要用来存放数字型的数据。最常用的就是NUMBER、PLS_INTEGER、BINARY_INTENER类型，还有一个类型是SIMPLE_INTEGER。

□ NUMBER类型可以表示整数和浮点数，该类型以十进制存储。其通用格式是NUMBER(precision, scale)，其中的precision表示精度，也就是数字的位数，可达38位；scale表示小数点后的位数。例如，NUMBER(3,1)可以存储-99.9~99.9之间的数值。该类型在定义的时候可以把precision和scale省略。例如，可以定义成NUMBER(8)或NUMBER。

□ PLS_INTEGER和BINARY_INTENER类型通常可以认为是一样的类型。表示的范围是-2 147 483 648~2 147 483 647之间。二者不一样的地方就是BINARY_INTENER发生溢出的时候能为其指派一个NUMBER类型而不至于发生异常，但PLS_INTEGER溢出会发生异常，建议使用PLS_INTEGER类型。

□ SIMPLE_INTEGER类型属于PLS_INTEGER的子类型，它的取值范围同PLS_INTEGER一样，只是该类型不允许为空。如果数据本身不需要溢出检查而且也不可能是空，那么可以选择该类型，该类型的性能比PLS_INTEGER高。

2) 字符类型，可以用来存储单个的字符或字符串的类型。主要有CHAR、VARCHAR2 (VARCHAR)、NCHAR、NVARCHAR2和LONG类型。

□ CHAR类型，用来描述固定长度的字符串，最长为32 767个字节，默认是长度1。该类型的字符串中如果值的长度达不到定义的长度，那么将以空格补齐。通常定义格式为CHAR(maximum_size)。一旦使用该类型，那么在数据库提取数据时可能要做空格处理。

□ VARCHAR2 类型，作为变量的时候最长为32 767个字节，但作为字段存储的时候是4000个字节。该类型表示可变量长度的字符串。也就是说，当值的长度达不到定义的长度时，不用空格补齐，这样就可以节省一定的空间。

□ NCHAR、NVARCHAR2类型使用方式同CHAR和VARCHAR2相同，只不过它们与国家的字符集有关。

□ LONG类型，以可变的方式存储数据，PL/SQL中作为变量可表示最长可达32 760字节的字符串，如果作为存储字段则可达2GB。

3) 布尔类型，它不能用做定义表中的数据类型。但PL/SQL中该类型可以用来存储逻辑上的值。它有3个值可选：TRUE、FALSE、NULL。

4) 日期类型，主要有DATE和TIMESTAMP。

□ DATE类型可以存储月、年、日、世纪、时、分和秒。

□ TIMESTAMP类型由DATE演变而来，可以存储月、年、日、世纪、时、分和秒以及小数的秒。

5) 使用%TYPE方式定义变量类型。这种定义变量类型的方式和前面所介绍的直接定义变量类型有所不同，它利用已经存在的数据类型来定义新数据的数据类型。例如，当定义多个变量或常量时，只要前面使用过的数据类型，后面的变量就可以利用%TYPE引用。最常见的就是把表中字段类型作为变量或常量的数据类型。使用此种方式的好处有如下几点：

□ 利用%TYPE定义的变量或常量数据类型都一致，当有变动的需求时，只要改变被引用



的变量或常量的数据类型,其他引用处的数据类型自然就变了,避免了逐条修改的麻烦。

- 使用PL/SQL语句块通常都是操作数据库表的数据,操作过程中避免不了出现数据的传递,这时变量利用%TYPE就可以完全兼容提取的数据,而不至于出现数据溢出或不符合的情况。
- 当利用%TYPE定义数据类型时,可以保证变量的数据类型和表中的字段类型同步,当表字段类型发生变化时,PL/SQL块变量的数据类型不需要修改。

【示例5】变量定义示例

示例演示如何定义标量类型的变量。具体演示脚本如下:

```

01 DECLARE
02     v_productid      productinfo.productid%TYPE;           -- 产品ID
03     v_productname     VARCHAR2(20);                        -- 产品名称
04     v_productprice    NUMBER(8,2);                         -- 产品价格
05     v_quantity        NUMBER(10);                          -- 数量
06     v_desperation     CONSTANT v_productname%TYPE:='测试'; -- 测试
07
08     v_spitgr          SIMPLE_INTEGER := 99.9;
09     v_long             LONG := 'LONG类型测试';
10     v_date            DATE := SYSDATE;
11 BEGIN
12     SELECT productid, productname, productprice, quantity
13     INTO v_productid, v_productname, v_productprice, v_quantity
14     FROM productinfo
15     WHERE productid = '0240040001';
16
17     DBMS_OUTPUT.PUT_LINE('v_productid = ' || v_productid);
18     DBMS_OUTPUT.PUT_LINE('v_productname = ' || v_productname
19     || ' 长度=' || LENGTH(v_productname));
20     DBMS_OUTPUT.PUT_LINE('v_productprice = ' || v_productprice);
21     DBMS_OUTPUT.PUT_LINE('v_quantity = ' || v_quantity);
22     DBMS_OUTPUT.PUT_LINE('v_desperation = ' || v_desperation);
23     DBMS_OUTPUT.PUT_LINE('v_spitgr = ' || v_spitgr);
24     DBMS_OUTPUT.PUT_LINE('v_long = ' || v_long);
25     DBMS_OUTPUT.PUT_LINE('v_date = ' || v_date);
26 END;
```

【代码解析】

- 该语句块中的声明部分采用了右对齐的方式。
- 第2行利用%TYPE方式声明变量,表示变量v_productid的类型同表productinfo中的productid字段数据类型一致。
- 第3行表示数据类型是长度为20的VARCHAR2型。前面介绍过,该类型可变地存储数据,可以利用LENGTH()函数查看真实长度。
- 第4~5行是NUMBER类型的不同使用方式。
- 第6行利用%TYPE引用了v_productname的数据类型,并且声明了一个常量。
- 第8行初始值是99.9,但由于数据类型只能表示整数,所以变量v_spitgr的值应为100。
- 第11~26行属于执行体部分。



- 第12~15行表示从产品表查询数据，并存储到变量中。
 □ 第17~25行表示输出结果到屏幕，用于验证变量存储数据。

【执行效果】

打开SQL*Plus执行以上脚本，执行流程见图7.6。



图7.6 变量类型声明演示

7.2.3 复合类型的变量

所谓复合类型的变量，就是每变量包含几个元素，可以存储多个值。这种变量类型同标量类型使用方式稍有差异，复合类型需要先定义，然后才能声明该类型的变量。最常用的是三种类型，一种是记录类型；一种是索引表类型；还有一种是VARRAY数组。

1. PL/SQL “记录类型”

该类型可以包含一个或多个成员，而每个成员的类型可以不同，成员可以是标量类型，也可以是引用其他变量的类型（使用%TYPE）。该类型比较适合处理查询语句中有多个列的情况。最常用的就是在调用某张表中的一行记录时，利用该类型变量存储该行记录。如果想要调用其中的数据，可以用“变量名称.成员名称”的格式进行调用。“记录类型”有两种声明的方式。

1) 第一种声明语法如下：

```
01 TYPE type_name IS RECORD
02 (
03   field_name datatype
04 [
05   { NOT NULL }
06   { := | DEFAULT } expression
07 ]
08 [ , field_name datatype [ { NOT NULL } { := | DEFAULT } expression ] ...
09 );
```



【语法说明】

- 第1行的type_name表示定义的记录类型的名称。其余都是关键词。
- 第3行的field_name表示行记录的成员名称，datatype表示行记录成员数据类型。
- 第5行的[NOT NULL]表示可选部分，可以约束记录的成员非空。
- 第6行的{ := | DEFAULT }表示为记录成员赋值，expression为赋值表达式。
- 第8行同第3~7行表示的含义相同，这么写是让读者明白，记录类型里可以有多个成员。

从上面语法介绍中可以看出记录类型可以包含多个成员，而其成员的定义方式和前面介绍的标量定义方式相同。

【示例6】定义“记录类型”的变量

下面一段PL/SQL脚本将演示如何定义PL/SQL记录类型的变量。

```

01 DECLARE
02 TYPE product_rec IS RECORD
03 (
04     v_productid    productinfo.productid%TYPE,    --产品ID
05     v_productname  VARCHAR2(20),                --产品名称
06     v_productprice NUMBER(8,2)                  --价格
07 );
08
09 v_product product_rec;
10 BEGIN
11     SELECT productid, productname, productprice
12     INTO v_product
13     FROM productinfo
14     WHERE productid = '0240040001';
15
16     DBMS_OUTPUT.PUT_LINE('productid = ' || v_product.v_productid);
17     DBMS_OUTPUT.PUT_LINE('productname = ' || v_product.v_productname);
18     DBMS_OUTPUT.PUT_LINE('productid = ' || v_product.v_productprice);
19 END;
20 /

```

【代码解析】

- 第1~9行是PL/SQL块的声明部分，第10~19行是PL/SQL块的执行体部分。
- 第2~7行属于声明行记录类型。第2行product_rec是行记录的名称。
- 第4~6行表示在行记录中包含了3个成员。这3个成员声明方式同标量类型声明方式一致。从中可以看出它的成员也可以利用%TYPE来声明变量类型。
- 第9行表示声明变量v_product，它的数据类型是product_rec类型。
- 第11~14行利用SELECT...INTO语句为变量赋值，这里INTO后面直接是v_product记录类型，这样赋值会依据声明记录类型时里面成员的顺序依次赋值。这种赋值方式比较方便。
- 第16~18行表示输出结果。

【执行效果】

打开SQL*Plus，首先设置输出参数为TRUE，也就是SET SERVEROUTPUT ON语句。然后执行脚本，执行过程见图7.7。



图7.7 记录类型的变量执行过程

2) 利用%ROWTYPE声明记录类型数据。

前面已经介绍过“记录类型”的变量，该类型是提取行记录时常用的存储数据的方式。除了上面的直接声明方式外还有一种声明记录类型的方式，就是利用%ROWTYPE。这种声明方式可以直接引用表中的行作为变量类型。它同%TYPE类似，可以避免因表中字段的数据类型改变而导致PL/SQL块出错的问题。

【示例7】%ROWTYPE示例

示例将演示如何利用%ROWTYPE声明变量类型。具体脚本如下：

```
01 DECLARE
02 v_product productinfo%ROWTYPE;
03 BEGIN
04     SELECT * INTO v_product
05     FROM productinfo
06     WHERE productid = '0240040001';
07     DBMS_OUTPUT.PUT_LINE('productid = ' || v_product.productid);
08     DBMS_OUTPUT.PUT_LINE('productname = ' || v_product.productname);
09     DBMS_OUTPUT.PUT_LINE('productid = ' || v_product.productprice);
10 END;
11 /
```

【代码解析】

- 第2行声明名称为v_product的变量，其数据类型是表productinfo的行记录类型。这里利用了%ROWTYPE方式声明变量数据类型。
- 第4~6行表示把查询出来的数据存储到变量v_product中。其中从第4行可以看出，这里查询了一行记录，它可以直接把这行记录放进v_product里。
- 第7~9行表示输出结果，提取结果的方式就是前面介绍过的“变量名称.成员名称”。

【执行结果】

打开SQL*Plus执行以上脚本，结果见图7.8。

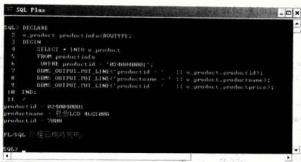


图7.8 利用%ROWTYPE方式声明变量

从执行结果可以看出，它同上一个示例最后结果没有区别，但是这个示例却使用更少的代码完成了同一个功能。

2. PL/SQL索引表类型（关联数组）

该类型和数组相似，它利用键值查找对应的值。这里键值同真正数组的下标不同，索引表中下标允许使用字符串。数组的长度不是固定值，它可以根据需要自动增长。其中的键值是整数或字符串。而其中的值就是普通的标量类型，也可以是记录类型。可以利用“变量名称（键值）”为其赋值或取值，如果某个键值的指向已经有数据了，那么该操作就是更改已有的数据。具体语法如下：

```

01 TYPE type_name IS TABLE OF
02 {
03   column_type |
04   variable_name%TYPE |
05   table_name.column_name%TYPE |
06   table_name%ROWTYPE
07 }
08 [NOT NULL]
09 INDEX BY { PLS_INTEGER | BINARY_INTEGER | VARCHAR2 ( v_size ) }
  
```

【代码解析】

- 第1行的type_name表示该类型的名称。其他为固有关键词。
- 第3~6行表示索引表中的数据类型。其中，第3行表示标量数据类型，第4和第5行表示利用%TYPE引用类型，第6行利用ROWTYPE引用类型。这在下面的示例中将要看到。
- 第8行表示是否可以不为空的约束。
- 第9行表示数组下标的数据类型。

以上的语法只是索引表类型本身的定义语法，并没有包含变量的定义。实际上，这里和变量的定义没有关系。如果想把某个变量声明成索引表类型，按照下面语法就好：

```
variable_name type_name;
```

其中，variable_name就是变量的名称，而type_name就是索引表的名称。日常开发中可以选择用数字作为键值或以字符串作为键值。下面的两个示例将演示如何使用这两种方式操作。

【示例8】数字为键值的索引表示例

示例演示以数字为键值的索引表操作过程。具体脚本如下：

```

01 DECLARE
02 TYPE prod_tab_fst IS TABLE OF productinfo%ROWTYPE -- %ROWTYPE
03     INDEX BY BINARY_INTEGER;                        -- BINARY_INTEGER
04
05 TYPE prod_tab_sec IS TABLE OF VARCHAR2(8)
06     INDEX BY PLS_INTEGER;                            -- PLS_INTEGER
07
08 v_prt_row prod_tab_fst;
09 v_prt      prod_tab_sec;
10
11 BEGIN
12     v_prt(1) := '正数';
13     v_prt(-1) := '负数';
14
15     SELECT * INTO v_prt_row(1)
16     FROM productinfo
17     WHERE productid = '0240040001';
18
19     DBMS_OUTPUT.PUT_LINE('行数据-v_prt_row(1) = ' || v_prt_row(1).productid
20     || '---' || v_prt_row(1).productname);
21
22     DBMS_OUTPUT.PUT_LINE('v_prt(1) = ' || v_prt(1));
23     DBMS_OUTPUT.PUT_LINE('v_prt(-1) = ' || v_prt(-1));
24
25 END;
26 /

```

【代码解析】

- 第2~3行表示声明一个索引表，名称为prod_tab_fst。第2行表示以%ROWTYPE方式声明。也就是说，prod_tab_fst中的元素都是productinfo的行记录。这在下面赋值的时候可以看出来。第3行表示以BINARY_INTEGER类型为索引，可以认为是键值，是BINARY_INTEGER类型。
- 第5~6行表示声明另一个索引表，名称为prod_tab_sec。第5行表示该索引表元素的类型是VARCHAR2类型。第6行表示该索引表以PLS_INTEGER类型为索引。前面介绍过，该类型同BINARY_INTEGER类型区别不大，甚至认为是一样的。
- 第8行表示声明变量，该变量的类型是prod_tab_fst。
- 第9行同样表示声明变量，其类型是prod_tab_sec。
- 第12~13行表示为变量v_prt赋值。可以看到，它赋值的方式同数组类似，以“变量名（索引）”的形式赋值，当取值时也需要同样的规则。
- 第15~17行利用SELECT...INTO语句为变量v_prt_row赋值。从中可以看出它把productinfo表的一行记录存储到了变量中。



□ 第19~24行表示把存储的数据输出到屏幕。

【执行效果】

在SQL*Plus中执行以上脚本，并查看执行结果。执行过程见图7.9。



图7.9 数字为键值的索引表使用过程

以上示例使用数字作为索引类型，也是比较常用的一种方式。但是，索引表还支持一种以字符串为键值的索引方式。

【示例9】字符串为键值的索引表

示例将演示以字符串为键值的索引表如何操作。具体脚本如下：

```
01 DECLARE
02 TYPE prodt_tab_thd IS TABLE OF NUMBER(8)
03 INDEX BY VARCHAR2(20); -- INDEX BY VARCHAR2(20)
04 v_prt_chr prodt_tab_thd;
05
06 BEGIN
07 v_prt_chr('test') := 123;
08 v_prt_chr('test1') := 0;
09
10 DBMS_OUTPUT.PUT_LINE('v_prt_chr(123) = ' || v_prt_chr('test'));
11 DBMS_OUTPUT.PUT_LINE('v_prt_chr(000) = ' || v_prt_chr('test1'));
12 DBMS_OUTPUT.PUT_LINE('v_prt_chr(000) = ' || v_prt_chr.first);
13 DBMS_OUTPUT.PUT_LINE('v_prt_chr(000) = ' || v_prt_chr(v_prt_chr.first));
14 END;
15 /
```

【代码解析】

□ 第2~3行表示创建索引表。其名称为prodt_tab_thd，索引键值为字符串型。

□ 第4行声明变量，类型为表类型。

□ 第7~8行为变量赋值，因为是字符串类型的键值，所以允许以示例中的格式赋值。



□ 第10~13行表示输出屏幕结果。从中可以看到以“变量名称.first”的格式得到第一个或最后一个值。

【执行效果】

在SQL*Plus中执行该示例脚本，执行过程见图7.10。

```

SQL> DECLARE
2 TYPE prod_tab_tbl IS TABLE OF NUMBER(8);
3 INDEX BY VARCHAR2(20); -- INDEX BY VARCHAR2(20)
4 v_prod_chr prod_tab_tbl;
5
6 BEGIN
7 v_prod_chr('test') := 123;
8 v_prod_chr('test1') := 0;
9
10 DBMS_OUTPUT.PUT_LINE('v_prod_chr(123) = ' || v_prod_chr('test')?);
11 DBMS_OUTPUT.PUT_LINE('v_prod_chr(000) = ' || v_prod_chr('test1')?);
12 DBMS_OUTPUT.PUT_LINE('v_prod_chr(000) = ' || v_prod_chr('test1')?);
13 DBMS_OUTPUT.PUT_LINE('v_prod_chr(000) = ' || v_prod_chr(v_prod_chr.first)?);
14 END;
15 /
v_prod_chr(123) = 123
v_prod_chr(000) = 0
v_prod_chr(000) = test
v_prod_chr(000) = 123

PL/SQL 过程已成功完成。

SQL>
  
```

图7.10 字符串为键值的索引表使用过程

3. VARRAY变长数组

该类型的元素个数是需要限制的，它是一个存储有序元素的集合。集合下标从1开始，比较适合较少的数据使用。声明语法如下：

```

TYPE type_name IS { VARRAY | VARYING ARRAY } ( size_limit )
OF element_type [ NOT NULL ]
  
```

【语法说明】

□ type_name：表示该数组的名称。

□ { VARRAY | VARYING ARRAY }：必选项，二选一，表示数组类型。

□ size_limit：该数组的长度。

□ element_type：数组里元素的类型。

【示例10】VARRAY数组示例

示例将演示如何声明该类型的变量。脚本如下：

```

01 DECLARE
02 TYPE varr IS VARRAY(100) OF VARCHAR2(20);
03
04 v_product varr:=varr('1','2');
05 BEGIN
06 v_product(1):='THIS IS A';
07 v_product(2):='TEST';
08 DBMS_OUTPUT.PUT_LINE('productid = ' || v_product(1));
09 DBMS_OUTPUT.PUT_LINE('productid = ' || v_product(2));
  
```



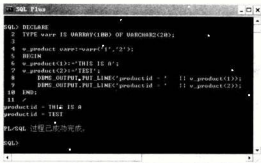
```
10 END;
11 /
```

【代码解析】

- 第2行表示声明VARRAY数组，数组长度为100，名称是varr，元素为VARCHAR2类型，长度为20。
- 第4行表示声明变量，变量名称为v_product，其类型为varr类型，该类型被初始化了两个元素（最多可初始化100个）。也就是说，只能向v_product中下标为1和2的元素赋值。
- 第6~7行表示为数组元素赋值。
- 第8~9行表示输出数组元素的值。

【执行效果】

在SQL*Plus中执行该示例脚本，执行过程见图7.11。



```
SQL> DECLARE
2  TYPE varr IS VARRAY(100) OF VARCHAR2(20);
3
4  v_product varr:=varr('0','2');
5  BEGIN
6  v_product(1):='THIS IS 0';
7  v_product(2):='TEST';
8  DBMS_OUTPUT.PUT_LINE('product_id = ' || v_product(1));
9  DBMS_OUTPUT.PUT_LINE('product_id = ' || v_product(2));
10 END;
11 /
product_id = THIS IS 0
product_id = TEST
PL/SQL 过程已成功完成。
SQL>
```

图7.11 变量为VARRAY数组类型演示

7.3 表达式

数据库中经常使用表达式来计算结果，尤其在变量和常量的使用过程中。在前面已经接触过表达式的使用，它和普通编程语言的表达式很类似。本节将系统地介绍表达式的类型以及如何使用表达式。

表达式根据操作数据类型的不同可以分为如下几类：

- 数值表达式；
- 关系表达式；
- 逻辑表达式。

下面对这几个表达式进行讲解。

7.3.1 数值表达式

数值表达式就是由数值类型的常量、变量以及函数，由算术运算符连接而成。在PL/SQL可以使用的算术运算符有：

- 加号+；
- 减号-；





- ☐ 乘号*;
- ☐ 除号/;
- ☐ 乘方**。

以上构成了PL/SQL中的基础运算，如果对数据有更高的要求，可以使用数值类型的函数来帮助完成。以上运算符的优先级和通常接触到的基础运算的优先级是一样的。

【示例11】计算表达式的平方根

示例演示了如何计算 $\sqrt{58+25 \times 3+(19-9)^2}$ 。具体脚本如下：

```
01 DECLARE
02 v_result NUMBER(10,4);
03 BEGIN
04   v_result := SQRT(58+25*3+(19-9)**2);
05   DBMS_OUTPUT.PUT_LINE('v_result = ' || v_result);
06 END;
07 /
```

【代码解析】

- ☐ 第2行声明NUMBER类型的变量，该变量长度为10，其中4位小数。
- ☐ 第4行利用了函数以及四则运算符进行运算。
- ☐ 第5行输出结果。这里说明一下，其中的“||”用于连接两个字符串，这里隐式地把数值型转换成了字符串类型。

【执行效果】

在SQL*Plus中执行该示例脚本，执行过程见图7.12。

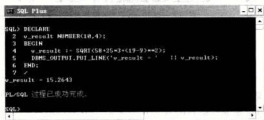


图7.12 数值表达式演示

7.3.2 关系表达式和逻辑表达式

由关系运算符连接起来的字符或数值称为关系表达式。其中关系运算符主要有以下几种：

- ☐ 等于号=;
- ☐ 小于号<;
- ☐ 大于号>;
- ☐ 小于等于号<=;
- ☐ 大于等于号>=;
- ☐ 不等于号!=和<>。

关系表达式很多用在PL/SQL的条件语句中，它们最后的结果是一个布尔类型值。这在后



面介绍的PL/SQL条件语句中会看到，这里不再举例说明。

所谓逻辑表达式，就是由逻辑符号和常量或变量等组成的表达式。逻辑符号比较少，常见的有下面3种：

- ☐ 逻辑非NOT；
- ☐ 逻辑或OR；
- ☐ 逻辑与AND。

注意 关系表达式中的等于号和赋值符号“:=”的差异。

7.4 PL/SQL结构控制

PL/SQL既然是面向过程的编程语言，那么它就有针对逻辑的控制语句，这些语句在日常的PL/SQL编程中起着很重要的作用，可以完成业务逻辑的框架部分。本节就介绍PL/SQL的逻辑控制语句。

7.4.1 IF条件控制语句

条件控制语句就是根据当前某个参数的值来判断进入哪一个流程，这就好像做选择题一样，当满足某个条件时就进入对应的流程，否则进入另外一个流程。

IF条件语句是编程语言中最常见的结构形式，这种结构有着一个或多个布尔表达式，一旦给出的数据使得布尔表达式成立，那么将执行该表达式对应的语句，而后继续往下执行。当布尔表达式不成立时，程序会跳过该表达式对应的语句而继续往下执行。IF语句的整体流程如图7.13所示。

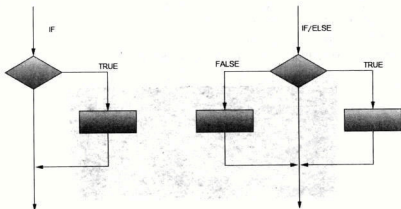


图7.13 IF条件语句流程图

在IF语句中有三种使用方式：IF...、IF...ELSE...、IF...ELSIF...。这三种方式可以根据实际的业务灵活选择。下面分别介绍这三种结构的具体使用方式。

1. IF...结构

这是IF语句中最简单的结构方式，它只有一个IF语句，如果给定的表达式不成立，那么将继续向下执行。其语法结构如下：

```
IF condition THEN
    statements;
END IF;
```

当condition为TRUE的时候程序会执行IF语句对应的statements。需要注意的是，IF后面有关键词“THEN”，这一点和其他编程语言中的IF语句不一样。

【示例12】IF...结构示例

示例演示如何使用IF...结构语句。该示例计算了 $\sqrt{58+25 \times 3 + (19-9)^2}$ 的值并赋给变量，利用IF语句判断，如果变量的值大于10，则将其结果输出。脚本如下：

```
01 DECLARE
02 v_result NUMBER(10,4);
03 BEGIN
04     v_result := SQRT(58+25*3+(19-9)**2);
05     IF v_result > 10 THEN
06         DBMS_OUTPUT.PUT_LINE('v_result 大于10 结果是: ' || v_result);
07     END IF;
08     DBMS_OUTPUT.PUT_LINE('IF条件语句已经执行完毕!');
09 END;
10 /
```

【代码解析】

□ 第5~7行是一个IF条件语句，第5行的 $v_result > 10$ 是条件，这行代码的含义是如果 v_result 的值大于10，那么执行第6行的内容。第6行表示把结果输出到屏幕。第7行是该IF语句的结束标志，并且使用了“;”号。

□ 第8行在IF语句执行完成后会继续执行。

以上是一个最简单的IF条件语句的示例，具体的执行效果见图7.14。

```
SQL> DECLARE
2 v_result NUMBER(10,4);
3 BEGIN
4     v_result := SQRT(58+25*3+(19-9)**2);
5     IF v_result > 10 THEN
6         DBMS_OUTPUT.PUT_LINE('v_result 大于10 结果是: ' || v_result);
7     END IF;
8     DBMS_OUTPUT.PUT_LINE('IF条件语句已经执行完毕!');
9 END;
10 /
v_result 大于10 结果是: 15.2643
IF条件语句已经执行完毕!
SQL>
```

图7.14 IF语句示例

示例演示的是条件 $v_result > 10$ 为TRUE时的执行结果，如果该条件为FALSE，那么最终输出结果将只有：

IF条件语句已经执行完毕!



2. IF...ELSE...结构

该类型的结构表示不是选A就是选B。该结构表示要么执行IF后面的语句，要么执行ELSE后面的语句，是二选一的模式。该结构执行完成后，程序会继续向后执行。其语法结构如下：

```
IF condition THEN
    statements;
ELSE
    statements;
END IF;
```

当condition为TRUE的时候程序会执行IF语句对应的statements。当condition为FALSE时会执行ELSE对应的statements。这两个条件二选一地执行。

【示例13】IF...ELSE结构示例

示例很简单，给变量赋值，判断如果变量大于20则执行其对应的语句，否则提示变量小于20。脚本如下：

```
01 DECLARE
02 v_if_con NUMBER(10);
03 BEGIN
04 v_if_con := 10;
05 IF v_if_con > 20 THEN
06 DBMS_OUTPUT.PUT_LINE('v_if_con > 20');
07 ELSE
08 DBMS_OUTPUT.PUT_LINE('v_if_con < 20');
09 END IF;
10 DBMS_OUTPUT.PUT_LINE('IF...ELSE...已结束!');
11 END;
12 /
```

【代码解析】

- 第4行为变量v_if_con赋值，值为10。
- 第5~9行属于IF...ELSE...语句结构。第5行表示判断如果v_if_con的值大于20则执行下面的第6行语句。与第5行对应的是第7行，表示当v_if_con小于或等于20时则执行第8行的语句。
- 第9行表示该IF语句结束。

【执行效果】

在SQL*Plus中执行以上脚本，执行效果见图7.15。



图7.15 IF...ELSE语句执行过程

3. IF...ELSIF...结构

该结构是前面两种使用方式的综合，它可以提供多个IF条件选择，当程序执行到该结构部分时，它会对每一个条件进行判断，一旦条件为TRUE，程序会执行对应的语句，而后继续判断下一个条件，直到所有条件判断完成。其语法结构如下：



```

01 IF condition1 THEN
02     statements;
03 ELSIF condition2 THEN
04     statements;
05 ...
06 {ELSE statements;}
07 END IF;

```

【语法说明】

- 该结构执行的流程是当第1行的condition1为TRUE时，执行第2行的语句，而后判断第3行的condition2是否为TRUE，如果为TRUE则执行第4行的语句，如果为FALSE则跳过第4行所代表的部分，然后继续向后执行。
- 该结构中可以有多个“ELSIF”分支条件，但每个ELSIF关键词后都需要有THEN关键词。
- 第6行的ELSE部分可以存在也可以不存在，如果存在，则在前面所有条件均为FALSE时会执行其对应的程序。

【示例14】 IF...ELSIF结构示例

示例演示了该结构执行过程，该示例生成一个100~200的随机数，然后利用IF语句判断该随机数所在范围。具体脚本如下：

```

01 DECLARE
02 v_if_con NUMBER(10);
03
04 BEGIN
05 v_if_con := DBMS_RANDOM.VALUE(100,200);
06 IF v_if_con > 100 AND v_if_con < 150 THEN
07     DBMS_OUTPUT.PUT_LINE('v_if_con值在100~150的范围内');
08 ELSIF v_if_con > 150 AND v_if_con < 180 THEN
09     DBMS_OUTPUT.PUT_LINE('v_if_con值在150~180的范围内');
10 ELSIF v_if_con > 180 AND v_if_con < 200 THEN
11     DBMS_OUTPUT.PUT_LINE('v_if_con值在180~200的范围内');
12 ELSE
13     DBMS_OUTPUT.PUT_LINE('v_if_con的值是一个边缘值');
14 END IF;
15 DBMS_OUTPUT.PUT_LINE('IF...ELSE...已结束!');
16 DBMS_OUTPUT.PUT_LINE('v_if_con 的值是 ' || v_if_con);
17 END;
18 /

```

【代码解析】

- 第2行声明数值类型变量v_if_con。
- 第5行表示为该变量赋值，赋值利用随机函数，利用DBMS_RANDOM.VALUE可以得到随机数值，该方式可以得到两个参数之间的38位小数。也就是说，第5行可以得到100~200之间的38位小数，不过由于声明的变量是整数类型，所以Oracle会隐式转换，自动截取整数部分，最终得到一个100~200之间的整数。
- 第6~14行是一个IF语句结构。第6行判断变量是否大于100并且小于150。当这两个条件

同时满足时会执行第7行的内容。IF语句的条件中间使用了逻辑与连接。第8行同第6行一样,不过要求范围在150~180。第12行表示以上条件都失败的情况下会执行第13行。

【执行效果】

在SQL*Plus中执行以上代码,其运行过程见图7.16。



```

SQL> DECLARE
2  v_if_con NUMBER(10);
3
4  BEGIN
5  v_if_con := DBMS_RANDOM.VALUE(100,200);
6  IF v_if_con > 100 AND v_if_con < 150 THEN
7  DBMS_OUTPUT.PUT_LINE('v_if_con落在100~150的范围内');
8  ELSIF v_if_con > 150 AND v_if_con < 200 THEN
9  DBMS_OUTPUT.PUT_LINE('v_if_con落在150~180的范围内');
10 ELSIF v_if_con > 100 AND v_if_con < 200 THEN
11 DBMS_OUTPUT.PUT_LINE('v_if_con落在100~200的范围内');
12 ELSE
13 DBMS_OUTPUT.PUT_LINE('v_if_con的值是一个初始值');
14 END IF;
15 DBMS_OUTPUT.PUT_LINE('IF...ELSE...已结束。');
16 DBMS_OUTPUT.PUT_LINE('v_if_con的值是 ' || v_if_con);
17 END;
18 /
v_if_con落在150~180的范围内
IF...ELSE...已结束!
v_if_con 的值是 172

PL/SQL 过程已成功完成.

SQL>

```

图7.16 IF...ELSIF执行过程

4. 嵌套使用IF语句

IF语句可以嵌套使用,这使得判断的条件更加精细。

【示例15】嵌套结构示例

示例演示了如何使用IF嵌套语句。该示例将首先判断产品表中产品的价格范围,并做出提示,然后在价格范围的基础上进行产品数量的判断,并给出产品数量是否满足需求的提示。具体脚本如下:

```

01 DECLARE
02 v_product productinfo%ROWTYPE;
03
04 BEGIN
05 SELECT * INTO v_product
06 FROM productinfo
07 WHERE productid = '0240040001';
08
09 IF v_product.productprice > 3000 THEN
10 DBMS_OUTPUT.PUT_LINE('该产品属于高价格产品');
11 IF v_product.quantity > 50 THEN
12 DBMS_OUTPUT.PUT_LINE('该产品数量高于50,不缺货。');
13 ELSE
14 DBMS_OUTPUT.PUT_LINE('该产品数量低于50,需要补货。');
15 END IF;
16 ELSIF v_product.productprice < 3000 AND v_product.productprice > 1000 THEN

```

```

17 DBMS_OUTPUT.PUT_LINE('该产品属于中间价格产品');
18 IF v_product.quantity > 80 THEN
19   DBMS_OUTPUT.PUT_LINE('该产品数量高于80, 不缺货。');
20   ELSE
21     DBMS_OUTPUT.PUT_LINE('该产品数量低于80, 需要补货。');
22   END IF;
23 ELSE
24   DBMS_OUTPUT.PUT_LINE('该产品属于低价格产品');
25   IF v_product.quantity > 200 THEN
26     DBMS_OUTPUT.PUT_LINE('该产品数量高于200, 不缺货。');
27     ELSE
28       DBMS_OUTPUT.PUT_LINE('该产品数量低于200, 需要补货。');
29     END IF;
30   END IF;
31
32 DBMS_OUTPUT.PUT_LINE('IF...ELSE...已结束。');
33 END;
34 /

```

【代码解析】

- 代码看起来比较多，但结构相当简单，核心部分由两个嵌套的IF条件语句构成。
- 该示例外层使用了IF...ELSIF...的结构，内层使用了IF...ELSE的结构。
- 第2行表示声明变量，类型为记录类型。
- 第5~7行表示为该变量赋值。
- 第9~30行表示一个IF语句的结构，这是最外层的IF条件判断，它实现了判断产品价格的功能。
- 第11~15行表示第二层的IF条件语句，它判断了产品的数量，并给出提示。

【执行效果】

在SQL*Plus中执行以上脚本，执行过程见图7.17。



图7.17 IF嵌套语句执行过程

虽然IF嵌套语句的使用可以帮我们完成更为复杂的逻辑,但不建议过多地使用该类型语句。其原因主要有两点:

- 1) 过多的嵌套使用会影响其执行效率。
- 2) 过多的嵌套使用会影响其阅读效果,如果允许,应尽量将其改为单一的IF条件语句。

7.4.2 CASE条件控制语句

CASE语句同IF语句类似,也是根据条件选择对应的语句执行。图7.18描述了CASE语句的整体流程。

CASE语句可以分为以下两种类型:

- 1) 一种是简单的CASE语句。它给出一个表达式,并把表达式结果同提供的几个可预见的结果做比较,如果比较成功,则执行对应的语句序列。
 - 2) 另一种是搜索式的CASE语句。它会提供多个布尔表达式,然后选择第一个为TRUE的表达式,执行对应的脚本。
- 下面分别介绍这两种CASE语句的使用方式。

1. 简单CASE语句

该类型CASE语句的语法结构如下:

```
01 [ <<label_name>> ]
02 CASE case_operand
03 WHEN when_operand THEN
04 statement ;
05 [
06 WHEN when_operand THEN
07 statement ;
08 ]...
09 [ ELSE statement [ statement ] ]... ;
10 END CASE [ label_name ] ;
```

【语法说明】

- <<label_name>>: 这是一个标签,可以选择性添加。如果添加标签,建议在CASE语句结束时也标明该标签,表示结束的是某个CASE语句,使用该标签可以提高可读性。
- 第2行case_operand: 这是一个表达式,通常是一个变量。PL/SQL中除了BLOB、BFILE、对象类型、PL/SQL记录类型、索引表、变长数组或嵌套表,其他类型都允许。
- 第3行when_operand: 它是case_operand对应的结果。如果when_operand的值同case_operand的值相同,那么将执行该子句下的语句,也就是第4行的statement。CASE语句中包含一个或多个WHEN...THEN子句。
- [ELSE statement [statement]]: 它所表示的含义是当所有的when_operand值都不能对应case_operand的值时,会执行ELSE处的语句。

【示例16】简单CASE语句示例

示例演示简单CASE语句的使用方式。要求利用产品ID得到对应的产品类型编码,然后利

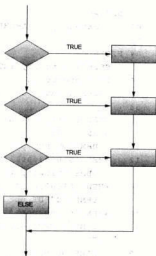


图7.18 CASE语句整体流程





用CASE语句找到产品类型编码对应的产品类型，并输出到屏幕。脚本如下：

```

01 DECLARE
02   v_categoryid VARCHAR2(12);
03
04 BEGIN
05   SELECT category INTO v_categoryid
06   FROM productinfo
07   WHERE productid = '0240040001';
08
09   CASE v_categoryid
10   WHEN '010001000' || '1' THEN
11     DBMS_OUTPUT.put_line(v_categoryid || '对应面具');
12   WHEN '0100030001' THEN
13     DBMS_OUTPUT.put_line(v_categoryid || '对应电视');
14   WHEN '0100030001' THEN
15     DBMS_OUTPUT.put_line(v_categoryid || '对应路由器');
16   WHEN '0100030002' THEN
17     DBMS_OUTPUT.put_line(v_categoryid || '对应洗衣机');
18   ELSE
19     DBMS_OUTPUT.put_line('没有对应的产品类型');
20   END CASE;
21
22   DBMS_OUTPUT.put_line('CASE结构已经完成。');
23 END;
24 /

```

【代码解析】

- 第2行声明变量v_categoryid。
- 第5~7行表示为变量赋值。
- 第9~20行是一个CASE结构。v_categoryid依次与第10、12、14、16行WHEN子句中的值进行对比，如果相等，将执行其对应的输出语句，而CASE语句也将执行完毕。
- 当没有与v_categoryid值相同的项时，会执行第18行ELSE部分。如果没有ELSE子句而且也没有与v_categoryid值相同的项时，系统会给出CASE_NOT_FOUND异常。

【执行效果】

在SQL*Plus中执行示例，其执行过程见图7.19。

从执行结果中可以看出，虽然第12行同第14行的产品类型编码相同，但只执行了第12行对应的输出语句。这说明CASE语句的执行有着从上到下的顺序。而且当WHEN子句对应

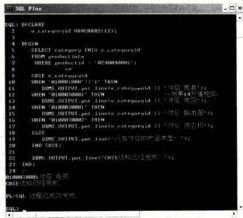


图7.19 简单类型的CASE语句执行过程

的语句被执行后，CASE结构也就执行完毕。

2. 搜索式的CASE语句

搜索式的CASE语句会依次检查布尔值是否为TRUE，一旦为TRUE，那么它所在的WHEN子句会被执行，而且它后面的布尔表达式将不被考虑。如果所有的布尔表达式都不为TRUE，那么程序将转到ELSE子句，如果没有ELSE子句，系统会给出CASE_NOT_FOUND异常。语法结构如下：

```
01 [<label_name>]
02 CASE
03 WHEN boolean_expression THEN statement ;
04 [WHEN boolean_expression THEN statement ;]...
05 [ELSE statement {statement}... ;
06 END CASE [label_name];
```

【代码解析】

- ☐ 从语法中可以看出搜索式的CASE语句在第2行处和简单CASE语句不同。它没有表达式，而简单CASE语句里有表达式。
- ☐ 第3行的boolean_expression为布尔表达式，而且只能是布尔表达式。
- ☐ 语法中其他项代表的含义见“简单CASE语句”的语法说明。

【示例17】搜索式CASE语句使用方式

示例将演示搜索式CASE语句的使用方法。要求根据产品ID得到对应的价格，利用CASE语句判断价格属于的范围并给出输出提示。具体脚本如下：

```
01 DECLARE
02     v_productprice NUMBER(8,2);
03
04 BEGIN
05     SELECT productprice
06     INTO v_productprice
07     FROM productinfo
08     WHERE productid = '0240040001';
09
10     CASE
11     WHEN v_productprice <= 1000 THEN
12         DBMS_OUTPUT.put_line('低价产品，价格是'||v_productprice);
13     WHEN v_productprice > 1000 AND v_productprice <= 3000 THEN
14         DBMS_OUTPUT.put_line('中价产品，价格是'||v_productprice);
15     WHEN v_productprice > 3000 THEN
16         DBMS_OUTPUT.put_line('高价产品，价格是'||v_productprice);
17     ELSE
18         DBMS_OUTPUT.put_line('错误价格：价格是'||v_productprice);
19     END CASE;
20
21     DBMS_OUTPUT.put_line('CASE结构已经完成。');
22 END;
23 /
```

【代码解析】

- 第2行声明变量v_productprice，用于存放价格。
- 第5~8行表示为变量v_productprice赋值。
- 第10~19行是一个CASE结构。程序将依次检测第11、13、15行的布尔表达式是否为TRUE。一旦为TRUE，它所在的WHEN子句将被执行。

【执行效果】

在SQL*Plus中执行脚本，执行过程见图7.20。

```

SQL> DECLARE
2  v_productprice NUMBER(8,2);
3
4  BEGIN
5  SELECT productprice
6  INTO v_productprice
7  FROM productinfo
8  WHERE productid = '002-000-00002';
9
10 CASE
11 WHEN v_productprice < 1000 THEN
12   BEGIN OUTPUT.put_line('该产品: 价格为 '|| v_productprice);
13   WHEN v_productprice < 1000 AND v_productprice < 3000 THEN
14     BEGIN OUTPUT.put_line('该产品: 价格为 '|| v_productprice);
15   WHEN v_productprice < 3000 THEN
16     BEGIN OUTPUT.put_line('该产品: 价格为 '|| v_productprice);
17   ELSE
18     BEGIN OUTPUT.put_line('该产品: 价格为 '|| v_productprice);
19   END CASE;
20
21 BEGIN OUTPUT.put_line('CASE语句已经可用。');
22
23 END;
24
输出产品: 价格为 7000
CASE语句已经可用。
SQL>
  
```

图7.20 搜索式的CASE语句执行过程

7.4.3 LOOP循环控制语句

LOOP语句也叫循环语句，它能让我们重复地执行指定的语句块。LOOP语句有以下四种形式：

- LOOP;
- WHILE...LOOP;
- FOR...LOOP;
- CURSOR FOR LOOP。

本小节将介绍前三种类型，至于CURSOR FOR LOOP会在第8章中介绍。

1. 基本的LOOP

该形式的语句属于LOOP循环控制语句中最基本的结构，它会重复不断地执行LOOP和END LOOP之间的语句序列。由于基本的LOOP语句本身没有包含中断循环的条件，所以通常情况下都是和其他的条件控制语句一起使用，利用EXIT、GOTO等可以中断LOOP循环。当然，异常也能使LOOP语句中断。图7.21描述了基本的LOOP语句的流程。

基本的LOOP语句语法结构如下：

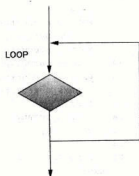


图7.21 LOOP语句流程



```

[<<label_name>>]
LOOP
    statement...
END LOOP [label_name]

```

【语法说明】

- <<label_name>>: LOOP语句的标签, 是可选项。
- LOOP: LOOP循环的开始标志。
- statement: LOOP语句中需要重复执行的语句。
- END LOOP: LOOP循环结束标志。

基本的LOOP语句需要和条件控制语句一起使用, 否则会出现死循环的情况, 直到内存溢出时抛出异常才能终止循环。通常情况下, 正常终止循环的方式有以下两种:

1) 在LOOP循环当中可以使用IF语句与EXIT的组合来结束循环。EXIT必须在循环体内部, 它可以使循环正常无条件终止, 并且终止循环后程序会正常执行循环外的语句。它如果同IF语句一起使用, 表示当满足某个条件时程序退出循环体, 继续向后执行。

【示例18】基本LOOP操作示例

示例简单地演示了LOOP的使用方式, 要求每次循环都需要为变量增加1, 并把变量输出到屏幕。当变量值大于5的时候, 要求终止循环。具体脚本如下:

```

01 DECLARE
02   v_num NUMBER(8) := 1;
03 BEGIN
04   = <<basic_loop>>
05   LOOP
06     DBMS_OUTPUT.put_line('当前v_num变量的值是: ' || v_num);
07     v_num := v_num + 1;
08     IF v_num > 5 THEN
09       DBMS_OUTPUT.put_line('退出! 当前v_num的值是 ' || v_num);
10       EXIT basic_loop;
11     END IF;
12   END LOOP ;
13   DBMS_OUTPUT.put_line('LOOP循环已经结束! ');
14 END ;
15 /

```

【代码解析】

- 第1行为声明变量, 默认值为1。
- 第4行是LOOP循环的标签, 可以选择性地填写。
- 第5~12行属于一个基本的LOOP结构。
- 第7行表示每循环一次都为变量增加1。
- 第8~10行是条件语句, 表示当变量v_num大于5时会退出循环。
- 第10行表示退出语句。basic_loop为LOOP的标签, 这里表示的含义是退出标签名为basic_loop的循环。

【执行效果】

在SQL*Plus中执行以上脚本, 查看执行结果。执行过程见图7.22。



图7.22 LOOP中利用IF条件结束循环

EXIT默认是终止退出当前的循环，但如果使用标签，可以终止并退出指定的LOOP循环。

2) 使用EXIT...WHEN语句来结束循环。这种方式在游标中会经常使用，这种情况会在后面的章节见到。它所代表的含义是：当WHEN后面的条件为TRUE时，EXIT会被触发，终止退出指定的循环，如果EXIT后不加LOOP标签，则表示终止退出当前循环。该语句可以替换简单的IF语句，下面的示例将演示这一点。

【示例19】EXIT...WHEN语句使用示例

要求使用EXIT...WHEN替换示例18中的IF语句。脚本如下：

```
01 DECLARE
02   v_num NUMBER(8) := 1;
03 BEGIN
04   <<basic_loop>>
05   LOOP
06     DBMS_OUTPUT.put_line('当前v_num变量的值是: ' || v_num);
07     v_num := v_num + 1;
08     EXIT basic_loop WHEN v_num > 5;
09   END LOOP ;
10
11   DBMS_OUTPUT.put_line('退出: 当前v_num的值是 ' || v_num);
12   DBMS_OUTPUT.put_line('LOOP循环已经结束: ');
13 END ;
14 /
```

【代码解析】

□ 第8行，利用EXIT...WHEN语句判断变量v_num如果大于5，则退出basic_loop循环。完成判断并退出，只需要一行脚本。

【执行效果】

在SQL*Plus中执行程序，查看结果是否与示例18相同。执行过程见图7.23。



图7.23 LOOP中利用EXIT...WHEN语句结束循环

从图7.23的执行结果中可以看出, EXIT...WHEN语句在某些时候可以替代IF语句, 而EXIT...WHEN语句更为简洁。建议读者如非必要, LOOP循环当中可以优先考虑该类型的语句。

2. WHILE...LOOP语句

WHILE...LOOP结构的语句本身可以终止LOOP循环, 当WHILE后面的布尔表达式为TRUE时, LOOP和END LOOP之间的语句集将执行一次, 而后会重新判断WHILE后面的表达式是否为TRUE。其流程见图7.24。

该语句结构的具体语法如下:

```

[<<label_name>>]
WHILE boolean_expression
LOOP
    statement...
END LOOP [label_name];

```

语法中的boolean_expression项是一个布尔表达式, 只有当该表达式为TRUE的时候, statement部分才能得到执行。其他部分和基本的LOOP语句没有区别。

【示例20】WHILE...LOOP语句示例

要求输出20以内能被3整除的数。脚本如下:

```

01 DECLARE
02  v_num NUMBER(8) := 1;
03 BEGIN
04  DBMS_OUTPUT.put('当前v_num变量的值是: ');
05  <<while_loop>>
06  WHILE v_num < 20
07  LOOP
08  IF MOD(v_num, 3) = 0 THEN

```

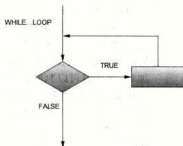


图7.24 WHILE...LOOP流程

```

09      DBMS_OUTPUT.put(v_num || ' ');
10  END IF;
11
12      v_num := v_num + 1;
13  END LOOP ;
14
15  DBMS_OUTPUT.put_line('退出！当前v_num的值是 ' || v_num);
16  DBMS_OUTPUT.put_line('LOOP循环已经结束！');
17 END ;
18 /

```

【代码解析】

- 第2行表示声明变量v_num，默认值为1。
- 第5~13行是WHILE...LOOP语句。
- 第5行是WHILE...LOOP语句的标签。
- 第6行表示当变量v_num<20时，执行下面的LOOP循环。
- 第8~10行利用SQL函数判断v_num是否能被3整除。如果整除，则把该值输出。
- 第12行表示每次循环都为变量v_num增加1。这样就可以判断1~20之间所有的数是否能被3整除。而且当变量增加到20时，循环会终止退出。

【执行效果】

在SQL*Plus中执行以上脚本，执行过程见图7.25。



图7.25 WHILE...LOOP执行过程

这种结构模式执行的顺序是先判断后循环，所以LOOP中的语句序列有可能连一次执行的机会都没有。例如，程序中的v_num初始值若大于19，LOOP循环将得不到执行。想要避免此类情况，可以采用下列脚本的方式：

```

01 ...
02      WHILE v_bol
03      LOOP
04          IF MOD(v_num,3) = 0 THEN
05              DBMS_OUTPUT.put(v_num || ' ');
06          END IF;
07
08          IF v_num >= 20 THEN
09              v_bol := FALSE;
10          END IF;
11          v_num := v_num + 1;
12      END LOOP ;
13 ...

```

【代码解析】

- 第2行的v_bol是一个布尔类型的变量，默认值为TRUE类型。一旦该变量值为FALSE，

循环将终止退出。

- ❑ 第8~10行判断变量v_num, 如果该变量的值大于等于20, 变量v_bol的值将被置为FALSE。

经过修改以后, 它的执行顺序和Java语言当中的DO-WHILE语句执行顺序相同。这种方式读者应该掌握。

读书笔记

3. FOR...LOOP

FOR...LOOP语句循环遍历指定范围内的整数。该范围被FOR和LOOP关键词封闭。当第一次进入循环时, 其循环范围会被确定, 并且以后不会再次计算。每循环一次, 其循环次数将会增加1。

FOR...LOOP语句的语法结构如下:

```
01 [<label_name>]
02 FOR index_name IN
03 [REVERSE]
04 lower_bound .. upper_bound
05 LOOP
06 statement...
07 END LOOP [label_name];
```

【语法说明】

- ❑ index_name: 循环计数器, 该变量可以得到当前的循环次数, 但是不能为其赋值。
- ❑ REVERSE: 该项指定循环的方式。默认的循环方式是由下标界到上标界, 也就是从lower_bound到upper_bound, 如果使用REVERSE关键词, 那么循环方式正好相反, 也就是从上标界到下标界。
- ❑ lower_bound: 循环范围的下标界。
- ❑ upper_bound: 循环范围的上标界。
- ❑ lower_bound和upper_bound两者需要用“..”连接。

【示例21】FOR...LOOP语句使用示例

该示例算出1~20之间所有整数的和。脚本如下:

```
01 DECLARE
02   v_num NUMBER(8) := 0;
03 BEGIN
04   DBMS_OUTPUT.put('1~20之间整数和, ');
05   <<for_loop>>
06   FOR inx IN 1..20 LOOP
07     v_num := v_num + inx;
08   END LOOP;
09
10   DBMS_OUTPUT.put_line(v_num);
11   DBMS_OUTPUT.put_line('LOOP循环已经结束! ');
12 END;
13 /
```

**【代码解析】**

- 第2行表示声明变量。
- 第5~8行是一个FOR...LOOP结构。
- 第6行表示循环范围从1到20。inx为循环计数器名。当inx等于20时，循环会自动退出。
- 第7行表示求出1~20之间整数的和。

【执行效果】

在SQL*Plus中执行该脚本，执行过程见图7.26。

```

SQL> DECLARE
2  v_sum NUMBER(8) := 0;
3  BEGIN
4      DBMS_OUTPUT.put('1~20之间整数的和: ');
5      ((for_inx))
6      FOR inx IN 1..20 LOOP
7          v_sum := v_sum + inx;
8      END LOOP;
9
10     DBMS_OUTPUT.put_line(v_sum);
11     DBMS_OUTPUT.put_line('LOOP循环已经结束: ');
12 END;
13
1~20之间整数的和: 210
LOOP循环已经结束
PL/SQL 过程已成功完成。
SQL>

```

图7.26 FOR...LOOP语句执行过程

FOR...LOOP循环当中的循环范围可以动态地获取。例如，示例中的下标界完全可以用数值型的变量替代。

需要说明的是，当lower_bound和upper_bound相等时，循环中的语句只能执行一次。

7.5 PL/SQL中使用DML和DDL语言

PL/SQL当中除了可以执行查询语句外，也允许执行DML语言和DDL语言。结构控制和DML操作相结合能帮助我们完成很多复杂的业务。不过在PL/SQL中DML语言和DDL语言这两者的执行方式有所差异。

7.5.1 DML语句的使用

由于PL/SQL对标准SQL的兼容，PL/SQL当中允许使用SQL命令，但有些命令在使用方式上有所改变。DML语句在PL/SQL中的使用方式和单独执行DML操作没有区别，而SELECT和DDL的使用方式都有所改变。

【示例22】INSERT语句示例

要求判断产品类型编码是否存在，如果不存在，则增加该编码类型的记录。脚本如下：

```

01 DECLARE
02  v_catgid VARCHAR2(10) := 0;
03  v_bool BOOLEAN := TRUE;
04 BEGIN
05  SELECT CATEGORYID INTO v_catgid
06  FROM CATEGORYINFO

```



```

07 WHERE CATEGORYNAME = '电脑';
08 DBMS_OUTPUT.PUT_LINE('电脑对应的编码存在。' || v_catgid);
09
10 EXCEPTION
11     WHEN NO_DATA_FOUND THEN
12         IF v_bol THEN
13             DBMS_OUTPUT.PUT_LINE('没有对应的编码。为其增加该产品类型');
14             INSERT INTO CATEGORYINFO VALUES ('0100000001', '电脑');
15             COMMIT;
16         END IF;
17     WHEN TOO_MANY_ROWS THEN
18         DBMS_OUTPUT.PUT_LINE('对应数据过多。请确认!');
19 END ;
20 /

```

【代码解析】

- 第5~7行查询产品类型名称为“电脑”的产品类型编码。如果对应编码存在，则将对应的编码输出到屏幕。如果不存在，在SELECT...INTO类型语句中会引发NO_DATA_FOUND异常，程序流程会发生跳转。此时会增加产品类型“电脑”对应的产品类型编码。
- 第10行开始属于PL/SQL块的异常部分。
- 第11行的NO_DATA_FOUND是Oracle的预定义异常，这在后面的相关章节会做介绍。
- 第14行表示，当CATEGORYINFO表中没有对应记录时，会增加该记录。该语句只能在触发NO_DATA_FOUND异常时才会被执行。
- 第15行表示提交事务，否则由于数据库的一致性，其他的用户将看不到该记录。关于事务，后面章节会有详细介绍。
- 第17行，TOO_MANY_ROWS是SELECT...INTO语句可能引发的另一个异常，返回多条记录时，会触发该异常。

【执行效果】

在SQL*Plus中执行以上脚本，并做查询，查看数据是否已经存入到表CATEGORYINFO中。执行过程见图7.27。

从图7.27中可以看出数据已经成功添加到数据库中，说明在PL/SQL和在条件控制语句中执行DDL操作都没有问题，而且查询和DML操作是在PL/SQL块中最常用的操作方式。



图7.27 PL/SQL结构语句中使用DML语句

注意 SELECT...INTO语句是SELECT语句在PL/SQL中的使用方法。

7.5.2 DDL语句的使用

PL/SQL中也允许使用DDL操作语句，但使用方法和DML有所差异。DDL语句要想在PL/SQL块中使用，需要一条命令来执行，该命令是EXECUTE IMMEDIATE，利用它可以执行动态的SQL语句，也就是利用它不仅仅可以执行DDL语句，也可以执行DML语句。该命令替代了DBMS_SQL包，其性能也有所提高，但是依然不建议过程使用该命令。

【示例23】DDL操作示例

要求在PL/SQL块中动态创建表TAB_TEST。表结构见表7.1。

具体脚本如下：

```
01 DECLARE
02   pc_createStr VARCHAR2(200);
03
04 BEGIN
05
06   pc_createStr := 'CREATE TABLE TAB_TEST
07                   (
08                       OPERID      VARCHAR2(10) PRIMARY KEY,
09                       OPERNAME    VARCHAR2(30),
10                       OPERDATE    DATE
11                   )';
12
13   EXECUTE IMMEDIATE pc_createStr;
14 END;
```

【代码解析】

- 第2行声明变量，该变量长度为200，用于存储DDL语句。
- 第6~11行代表的是为变量pc_createStr赋值，值为创建表TAB_TEST的脚本。
- 第13行表示利用EXECUTE IMMEDIATE命令动态创建表TAB_TEST。

【执行效果】

在SQL*Plus中执行脚本，并查看TAB_TEST表的结构。执行过程见图7.28。



图7.28 PL/SQL执行DDL语句

通常情况下,利用EXECUTE IMMEDIATE命令执行DDL语句会用在存储过程当中。这样可以更好地实现代码可移植性。

7.6 PL/SQL中的异常



同其他语言的程序一样,用PL/SQL编写的程序在运行过程当中也会出现各种错误,这些错误也可以称为异常。异常在PL/SQL中是很重要的一部分。

7.6.1 什么是异常

PL/SQL运行过程中有可能会出现问题,这些错误有的来自程序本身,也有的来自开发人员自定义的数据,而所有的这些错误我们称之为异常(编译时的错误不能称为异常)。

为了使程序有更好的可读性和健壮性,PL/SQL采用了捕获并统一处理异常的方式。当异常发生时,程序会无条件跳转到异常块处,将控制权交给异常处理程序。异常处理程序将进行异常匹配,如果当前的块内没有对应的异常名称,则异常会传至当前程序的上一层程序中查找,如果一直向上查找却依然没有找到对应的处理方式,则该异常会被传至当前的主机调用中,而运行的程序也会中断。

【示例24】异常演示示例

脚本演示了除数为0的异常。脚本如下:

```
01 DECLARE
02   v_rslt NUMBER(10) := 0;
03 BEGIN
04   v_rslt := 100/0;
05   DBMS_OUTPUT.PUT_LINE('结果是: ' || v_rslt);
06 END;
07 /
```

【代码解析】

- 第4行表示求除法运算的结果,不过这里除数出现了0,除数为0在四则混合运算中是不被允许的,编译程序时会正常通过,但执行程序时会出现错误提示,也就是有异常抛出。

【执行效果】

在SQL*Plus下执行示例脚本,执行过程见图7.29。

从执行结果中可以看出,程序出现错误,提示除数为0,“ORA-01476”是错误号。除数为0的异常属于Oracle内部定义异常。

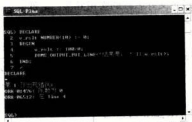


图7.29 PL/SQL异常演示

7.6.2 处理异常的语法

前面在介绍PL/SQL块结构时已经说过,PL/SQL有声明部分、执行体部分和异常部分。异常通常都发生在执行体部分,处理部分在PL/SQL块的最下方。其语法格式如下:



```

01 EXCEPTION
02 WHEN exception1 [OR exception2...] THEN          --异常列表
03 statement [ statement ]...                        --语句序列
04 [WHEN exception3 [OR exception4...] THEN          --异常列表
05 statement [ statement ]...
06 [WHEN OTHERS THEN
07 statement [ statement ]...]
```

【代码解析】

- 第1行的EXCEPTION表示声明异常块部分，它是异常处理部分开始的标志。
- 第2行的WHEN后面接异常名称列表，THEN后面接语句序列，也就是说发生的异常和异常列表里的异常相匹配时，可以执行指定的语句序列，以完成善后操作。
- 允许多个WHEN关键词。
- 第6行的WHEN OTHERS THEN语句通常是异常处理的最后部分，它表示如果抛出的异常在前面没有被捕获，那么将在这个地方被捕获。该语句可以不用，但不被捕获的异常会传递到主机环境。

Oracle中的异常可以分为三类：

- 预定义异常；
- 非预定义异常；
- 自定义异常。

其中，预定义异常和非预定义异常都与Oracle中的错误有关，当出现错误时会自动触发，而自定义异常与Oracle的错误没有关系，它是人为的为某种特殊情况定义的异常，也不会自动触发，需要显式的操作来触发。

7.6.3 预定义异常

Oracle中为每个错误提供一个错误号，而捕获异常则需要异常有名称。Oracle提供了一些已经定义好名称的常用异常，这就是预定义异常。例如，前面在使用SELECT...INTO语句时，如果返回超过一条记录就会触发TOO_MANY_ROWS异常。表7.2列出了一些常用的预定义异常。

表7.2 常用的预定义异常

Exception	ORA Error	SQL CODE	Condition
CASE_NOT_FOUND	ORA-06592	-6592	case语句中，when子句没有相匹配的条件，而且没有else语句，会触发该异常
NO_DATA_FOUND	ORA-01403	+100	select...into语句没有返回记录，触发该异常
TOO_MANY_ROWS	ORA-01422	-1422	select...into语句返回记录多于一条，触发该异常
DUP_VAL_ON_INDEX	ORA-00001	-1	表中唯一索引所对应的列上出现重复值时，引发该异常
VALUE_ERROR	ORA-06502	-06502	赋值时，如果变量长度不够，将引发该异常
ZERO_DIVIDE	ORA-01476	-1476	除数为零时引发该异常
STORAGE_ERROR	ORA-06500	-6500	内存溢出或破坏引发该异常
TIMEOUT_ON_RESOURCE	ORA-00051	-51	等待资源超时引发该异常
CURSOR_ALREADY_OPEN	ORA-06511	-6511	打开一个已经打开的游标引发该异常



预定义异常不止表7.2列出的这些，Oracle一共提供了25种预定义异常。表7.2中只是日常开发中比较常用的异常。利用下面的查询语句可以查看Oracle的预定义异常：

```
SELECT * FROM DBA_SOURCE WHERE NAME='STANDARD' AND TEXT LIKE '%EXCEPTION_INIT%';
```

【示例25】预定义异常示例

该示例中要求当除数为0时，做异常捕捉。脚本如下：

```
DECLARE
v_rslt NUMBER(10) := 0;
BEGIN
    v_rslt := 100/0;
    DBMS_OUTPUT.PUT_LINE('结果是: ' || v_rslt);
EXCEPTION
    WHEN ZERO_DIVIDE THEN
        DBMS_OUTPUT.PUT_LINE('除数是零: 默认用1替代除数。结果是: ' || 100/1);
END;
```

【执行效果】

执行效果如图7.30所示。

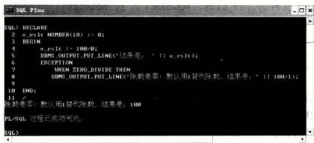


图7.30 捕获除数为0的异常

执行过程中出现除数为0的情况时，程序会马上进入异常捕获部分，当发生的异常和异常列表中的异常名称匹配成功时，执行WHEN...THEN下的语句序列，这样就可以避免因程序中中断而产生的问题。

异常匹配的顺序是从上到下，这一点需要注意。

7.6.4 非预定义异常

Oracle中的异常更多都是非预定义异常。也就是说，它们只有错误编号和相关的错误描述，而没有名称的异常是不能被捕捉的。为了解决该问题，Oracle允许开发人员为这样的异常添加一个名称，使得它们能够被异常处理模块捕捉到。

为一个非预定义异常定义名称需要如下两步：

- 1) 声明一个异常的名称。
- 2) 把这个名称和异常的编号相互关联。

Oracle处理预定义异常和非预定义异常并没有区别。

【示例26】关联非预定义异常示例

由于产品表PRODUCTINFO中的产品类型引用了产品类型表CATEGORYINFO的编码，所以当修改产品类型编码时有可能造成PRODUCTINFO产生垃圾数据。为了避免这种情况，表PRODUCTINFO中可以使用外键约束。这时如果直接修改PRODUCTINFO表中的产品类型编码，就有可能导致ORA-02291错误。

```
01 DECLARE
02 v_ctgy VARCHAR2(10);
03
04 my_2291_exp EXCEPTION;
05 PRAGMA EXCEPTION_INIT(my_2291_exp, -2291);
06
07 BEGIN
08     v_ctgy := '1111111111';
09     UPDATE PRODUCTINFO SET PRODUCTINFO.CATEGORY = v_ctgy;
10
11     EXCEPTION
12         WHEN my_2291_exp THEN
13             DBMS_OUTPUT.PUT_LINE('违反完整约束条件,未找到父项关键字!');
14             DBMS_OUTPUT.PUT_LINE('SQLERRM: ' || SQLERRM);
15             DBMS_OUTPUT.PUT_LINE('SQLCODE: ' || SQLCODE);
16             ROLLBACK;
17
18 END;
19 /
```

【代码解析】

- 第4行表示声明异常的名称：my_2291_exp。
- 第5行利用PRAGMA EXCEPTION_INIT把异常名称和数据库错误号关联。
- 第9行表示修改表PRODUCTINFO中产品类型编码的数据。
- 从第11行开始为异常处理块，第12行为异常名称列表，这里使用了my_2291_exp。也就是说，当出现-2291这个异常时，会被第12行捕捉。
- 第14行中的SQLERRM表示错误信息。
- 第15行中的SQLCODE为错误代码。

【执行效果】

在SQL*Plus中执行脚本，执行过程见

图7.31。

7.6.5 自定义异常

如果开发当中遇到与实际业务相关的错误，例如产品数量不允许为负数，生产日期必须在质保日期之前等，这些和业务相关的问题并不能算系统错误，也不能使用预定义和非预定义异常来捕捉它们。如果要想用异



图7.31 非预定义异常演示

常的方式处理这些问题,那么这样的异常需要开发人员自己编写,而且在调用的时候也需要显式的触发。

【示例27】自定义异常示例

要求根据输入的产品ID得到产品数量,如果产品数量小于0,则抛出异常,并做出提示。脚本如下:

```
01 DECLARE
02 v_prcid PRODUCTINFO.Productid%TYPE := '&产品ID';
03 v_qunty PRODUCTINFO.Quantity%TYPE;
04
05 quantity_exp EXCEPTION;
06 PRAGMA EXCEPTION_INIT(quantity_exp, -20001);
07
08 BEGIN
09
10     SELECT quantity INTO v_qunty
11     FROM PRODUCTINFO
12     WHERE productid = v_prcid;
13
14     IF v_qunty < 0 THEN
15         RAISE quantity_exp;
16     END IF;
17
18     DBMS_OUTPUT.PUT_LINE('该产品的数量是: ' || v_qunty);
19
20 EXCEPTION
21     WHEN quantity_exp THEN
22         DBMS_OUTPUT.PUT_LINE('出现产品数量为空的数据,请核查!');
23         ROLLBACK;
24     WHEN NO_DATA_FOUND THEN
25         DBMS_OUTPUT.PUT_LINE('没有对应的数据!');
26     WHEN TOO_MANY_ROWS THEN
27         DBMS_OUTPUT.PUT_LINE('对应数据过多,请确认!');
28
29 END;
30 /
```

【代码解析】

- 第2行表示声明变量v_prcid,它的值由替换变量“&产品ID”决定,利用替换变量可以达到创建通用脚本的目的。当执行程序时会提示输入替换数据,最后运行到脚本的是替换后的数据。
- 第3行声明变量v_qunty表示产品数量。
- 第5行声明异常名称quantity_exp。
- 第6行表示为异常关联一个错误号。其中错误号的范围是-20 999~-20 000的负整数,该范围内的错误号可以随便使用,不用担心被占用。
- 第10~12行表示根据产品ID查询其数量,并赋值给变量v_qunty。
- 第14~16行表示判断产品数量是否为负数,如果为负数,则抛出quantity_exp异常。利



用RAISE关键词可以显式地抛出异常。

□ 第20~27行代表异常处理部分。异常处理部分由3个WHEN...THEN子句组成。

【执行效果】

在SQL*Plus中执行程序，按照提示输入“产品ID”，这里输入“0240010001”。执行结果见图7.32。



图7.32 自定义异常执行结果

7.7 PL/SQL函数编写

函数由PL/SQL定义，可以操作各种数据项目完成计算并返回计算的结果值。利用函数可以把复杂的计算过程封装起来，避免所有的开发人员面对复杂的算法。

7.7.1 函数的组成

函数主要由以下几个部分组成：

- 输入部分。函数允许有输入的参数，调用函数时需要给这些参数赋值。
- 逻辑计算部分。函数内部将完成对各种数据项目的计算，它可以进行很简单的算术表达式运算，也可以调用多个SQL内置函数或自定义函数进行运算。
- 输出部分。函数要求都有返回值。

7.7.2 函数语法

函数的创建语句结构如下：

```
01 CREATE [ OR REPLACE ] FUNCTION [ schema. ] function_name
02 {
03   ( parameter_declaration [, parameter_declaration] )
04 }
05 RETURN datatype
06 { IS | AS }
07 [ declare_section ]
08 BEGIN
09   statement [ statement | pragma ]...
10   [ EXCEPTION exception_handler [ exception_handler ]... ]
11 END [ name ] ;
```

【语法说明】

- [OR REPLACE]：覆盖同名函数。
- FUNCTION：关键词，表示创建的是函数。



- schema: 模式名称。
- function_name: 函数名称。
- parameter_declaration: 函数的参数。参数有IN、OUT、IN OUT三种类型。
- RETURN datatype: 表示函数的返回类型。
- { IS | AS } : 二选一。该项之后是PL/SQL块。
- declare_section: 语句块部分的变量声明。
- 第9行表示语句或子程序。
- 第10行表示异常处理部分。

函数的作用是计算数据,并返回结果,所以在PL/SQL块中至少有一个RETURN语句。函数不能用来操作数据库,这一点和SQL内置函数一样。在当前模式下创建函数需要有CREATE PROCEDURE系统权限。

【示例28】创建函数示例

函数根据当前的产品数量对产品价格进行打折计算。如果产品数量低于50就打七五折,如果产品数量等于或高于50则打九折。脚本如下:

```
01 CREATE FUNCTION pric
02 (v_pric IN NUMBER,v_qnty IN NUMBER)
03 RETURN NUMBER
04 IS
05 BEGIN
06 IF v_qnty < 50 THEN
07     RETURN(v_pric * 0.75);
08 ELSE
09     RETURN(v_pric * 0.9);
10 END IF;
11 END ;
12 /
```

【代码解析】

- 第1行表示创建名称为pric的函数。
- 第2行声明函数的变量v_pric和v_qnty,分别表示价格和数量。这两个参数为输入类型的参数。
- 第3行表示返回类型为数字型。
- 第6~9行为判断产品数量,并根据产品数量对价格进行打折计算。

【调用函数】

在SQL*Plus中执行函数脚本,如果执行成功,那么该函数就可以正常调用了。执行以下查询脚本:

```
SELECT productid,productname,productprice,pric(productprice,quantity) FROM productinfo;
```

【执行效果】

调用函数结果见图7.33。

自定义的函数同SQL内置函数使用方式区别不大,它作为表达式的一部分,可以在语句中使用,也可以在PL/SQL块中使用。但是它不可以像存储过程一样独立运行。关于存储过程,会在第10章专门对其做介绍。

PRODUCTID	PRODUCTNAME	PRODUCTPRICE	PRICE(PRODUCTPRICE, QUANTITY)
4001	LCD-46G1000	7000	5.25
4002	XQ45H-7100	11000	8.25
4003	DE-4100SE-REC	34000	27.5
4004	XQ455-1914-REC	25000	22.5
4005	XQ410T	4000	3.0
4006	XQ410T	28000	21.0
4001	LCD-46G1000	59	5.25

图7.33 查询中调用prc函数

函数利用RETURN可以返回一个参数，某种情况下需要得到函数内的多个数据，那么可以采用OUT类型参数的方法，使其满足需求。

【示例29】IN OUT类型参数示例

函数将使用IN OUT类型的参数，函数有两个参数分别是产品类型编码和价格，求出该产品类型下比指定价格高的产品的平均价格，并返回该范围内最少的产品数量。

```

01 CREATE FUNCTION AVG_PRICE(V_CTGRY IN VARCHAR2, --产品类型和指定价格
02                             V_PRICE IN OUT VARCHAR2) RETURN NUMBER IS
03     V_QNTY NUMBER; --利用min函数得到的产品数量
04
05 BEGIN
06     IF V_PRICE IS NULL THEN
07         V_PRICE := 0;
08     END IF;
09
10     SELECT AVG(PRODUCTPRICE), MIN(QUANTITY)
11     INTO V_PRICE, V_QNTY
12     FROM PRODUCTINFO
13     WHERE CATEGORY = V_CTGRY
14     AND PRODUCTPRICE > V_PRICE;
15
16     RETURN V_QNTY;
17
18 EXCEPTION
19     WHEN NO_DATA_FOUND THEN
20         DBMS_OUTPUT.PUT_LINE('没有对应的数据!');
21     WHEN TOO_MANY_ROWS THEN
22         DBMS_OUTPUT.PUT_LINE('对应数据过多, 请确认!');
23 END;
24 /

```

【代码解析】

- 第1~2行表示创建函数AVG_PRICE，函数包含两个参数：V_CTGRY表示产品类型编码，是输入参数；V_PRICE表示指定的价格，是输入输出参数。函数本身返回类型为数值型。
- 第3行表示PL/SQL块内部变量，表示产品数量。



- 第6~8行判断如果参数V_PRIC为空,就默认为0。
- 第10~14行表示得到平均价格和最少的产品数量,并赋值到参数内。
- 第16行表示函数返回值是指定范围内的最少的产品数量。
- 第18~22行是异常捕捉块。

【调用函数】

函数执行成功后,在PL/SQL块内调用,只有这样才能获取OUT类型参数的值,否则只能得到利用RETURN返回的值。调用脚本如下:

```
DECLARE
  V_CTGRY VARCHAR2(10) := '0100030002';      --指定的产品类型编码
  V_PRIC  VARCHAR2(20) := 1500;                --指定的价格
  V_QNTY  VARCHAR2(20);                        --数量
BEGIN

  V_QNTY := AVG_PRIC(V_CTGRY, V_PRIC);          --调用函数,函数内部会把V_PRIC重新赋值

  DBMS_OUTPUT.PUT_LINE('平均价格: ' || V_PRIC);
  DBMS_OUTPUT.PUT_LINE('最低的产品数量是: ' || V_QNTY);
END;
```

【执行效果】

在PL/SQL块中执行以上脚本,执行过程见图7.34。

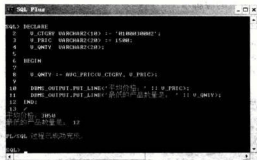


图7.34 PL/SQL调用函数

7.7.3 查看函数

函数一旦创建成功,就会存储在Oracle服务器中,随时可以调用,也可以查看具体脚本。对于当前用户所在模式,用户可以在数据字典USER_PROCEDURES中查看其属性,在数据字典USER_SOURCE中查看其源脚本。这两个数据字典属于于视图,利用这两个视图不仅可以查看函数的相关信息,也可以查看存储过程的相关信息。这一点在第10章会提及。除了这两个视图以外,也可以在数据字典视图DBA_PROCEDURES和数据字典视图DBA_SOURCE查看同样的信息。



下面的两个示例将演示如何查看已经建立的函数。

【示例30】查看已有函数名称的示例

在SQL*Plus下执行查询脚本，查看已有的函数名称。脚本如下，查询过程见图7.35。

```
COL OBJECT_NAME FORMAT A60
```

格式化字段长度，避免出现自动换行的情况。

```
SELECT OBJECT_NAME,OBJECT_ID,OBJECT_TYPE
FROM USER_PROCEURES
ORDER BY OBJECT_TYPE;
```

OBJECT_NAME	OBJECT_ID	OBJECT_TYPE
PRIC	72231	FUNCTION
DEPT	71826	FUNCTION
PRODUCI_SF0011_FNC	72214	FUNCTION
	72215	FUNCTION

图7.35 查询已有函数

图7.35中标出部分就是前面介绍的两个函数。从该视图查出函数名称后，就可以查看函数的具体源码。接下来的示例将演示如何查看函数源脚本。

【示例31】查看已有函数的源脚本

在SQL*Plus中分别执行如下脚本，查看AVG_PRIC函数的源脚本。执行结果见图7.36。

```
COL NAME FORMAT A15
```

```
COL TEXT FORMAT A80
```

```
SELECT NAME,LINE,TEXT FROM USER_SOURCE WHERE NAME = 'AVG_PRIC';
```

```

NAME          LINE TEXT
-----
AVG_PRIC      1  FUNCTION AVG_PRIC (CUST IN VARCHAR2, ...
AVG_PRIC      2  P UNIT NUMBER) RETURN NUMBER IS
AVG_PRIC      3  P NUMBER;
AVG_PRIC      4  BEGIN
AVG_PRIC      5  IF CUST IS NULL THEN
AVG_PRIC      6  P PRIC := 0;
AVG_PRIC      7  END IF;
AVG_PRIC      8
AVG_PRIC      9  SELECT AVG(PRICE) INTO P_PRIC FROM USER_PROCEURES WHERE NAME = 'AVG_PRIC';
AVG_PRIC     10  RETURN P_PRIC;
AVG_PRIC     11
NAME          LINE TEXT
-----
AVG_PRIC     12  BEGIN RETURN(0);
AVG_PRIC     13  END;
AVG_PRIC     14  END FUNCTION;
AVG_PRIC     15  END;
AVG_PRIC     16  BEGIN
AVG_PRIC     17  IF CUST IS NULL THEN
AVG_PRIC     18  P_PRIC := 0;
AVG_PRIC     19  END IF;
AVG_PRIC     20  SELECT AVG(PRICE) INTO P_PRIC FROM USER_PROCEURES WHERE NAME = 'AVG_PRIC';
AVG_PRIC     21  RETURN P_PRIC;
AVG_PRIC     22  END;
AVG_PRIC     23
NAME          LINE TEXT
-----
AVG_PRIC     24  END;

```

图7.36 查看AVG_PRIC源脚本

注意 查询函数时函数名称需要使用大写字母, 否则查询不到对应数据。

7.7.4 在PL/SQL Developer中创建函数

使用PL/SQL Developer可以创建函数, 该工具可以为开发人员提供一个函数模板, 然后在该模板内编写业务脚本。这种方式比较适合初学者。

创建函数步骤如下:

- 1) 启动PL/SQL Developer工具, 在整个布局的左上角单击【File】, 从弹出的下拉列表中选择【New】然后进入到【Program Window】项, 在列表中找到【Function】项, 如图7.37所示。
- 2) 单击【Function】命令弹出函数创建对话框, 填写3项对应的信息, 见图7.38。

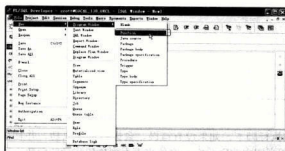


图7.37 创建函数选项



图7.38 创建函数模板窗口

- ☐ Name: 表示创建的函数名称。
- ☐ Parameters: 函数的参数列表。
- ☐ Return type: 函数返回参数类型。

- 3) 单击【OK】按钮进入Program窗口, 见图7.39。该窗口是根据填写的信息自动生成的脚本。

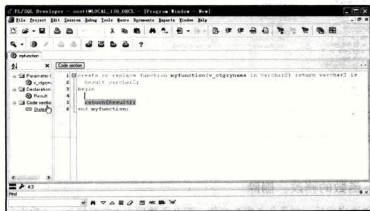


图7.39 自动生成的脚本



窗口分为左、右两部分，左边是结构导航窗口，右边是编码窗口。利用左边的导航可以在右边众多代码中快速定位。

4) 填写业务脚本。这里做一个简单测试，给传进去的参数追加“test”字符串。补全后的脚本见图7.40。

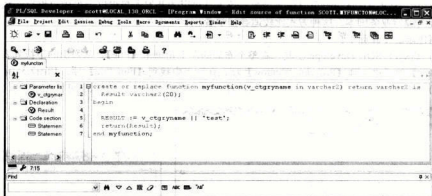


图7.40 填写业务脚本

5) 执行该窗口内的脚本。按F8键执行脚本，如果没有错误，会提示创建成功，并且在PL/SQL Developer左边的【Objects】透视图中的Functions文件夹下看到创建成功的函数。具体效果如图7.41所示。

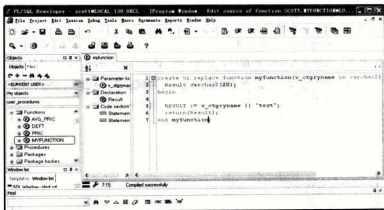


图7.41 成功创建函数

7.7.5 函数的修改、删除

当函数创建完成后，如果业务发生变化，那么函数脚本也需要修改。函数内容的修改需要利用REPLACE关键词，在第7.7.2小节的语法中说明了如何使用该关键词，在创建的函数中加



上OR REPLACE关键词,从而达到完成函数修改的目的,也就是覆盖。

函数的删除同样简单。下面是删除函数的语法:

```
DROP FUNCTION [schema.]function
```

schema: 函数所属的模式名称。

【示例32】删除函数示例

要求删除myfunction。执行删除脚本:

```
DROP FUNCTION myfunction;
```

7.8 小结

PL/SQL是操作Oracle的基础,日常开发中也是最常用的部分。本章利用大量的篇幅介绍了PL/SQL基础方面的知识。下面概括了本章的主要知识点:

- 什么是PL/SQL, PL/SQL的优势和PL/SQL块的结构。这是PL/SQL的基础,在这部分应当了解PL/SQL块分为声明部分、执行体部分和异常处理部分。
- PL/SQL中的变量和常量。介绍了变量和常量的类型。概括来说,变量类型可分为标量类型和复合类型。
- 有关表达式在PL/SQL编程中或标准SQL语句中都有使用。
- PL/SQL既然是编程语言,那么它就有结构控制语句,这里所涉及的结构控制语句有IF语句、CASE语句和LOOP语句。这3种结构中都有分支结构,是PL/SQL编程必不可少的一项。
- PL/SQL块中允许执行SELECT...INTO语句为变量赋值,也允许执行DML和DDL语句,不过执行DDL需要相关的命令。
- PL/SQL中很重要的一部分就是异常处理。Oracle采用了统一处理异常的方式,当PL/SQL块中发生异常时,程序会立刻无条件地转到异常处理部分,然后与异常名称列表进行匹配,如果匹配成功则表示异常被捕捉到,否则异常会一直向外层传递,直到主机调用界面。
- 利用PL/SQL可以自定义函数,函数允许查询数据,但却不允许对数据库进行操作,使用函数主要对数据项进行计算。

7.9 习题

一、填空题

1. PL/SQL块分为_____、_____和_____3部分。
2. IF语句有_____、_____和_____3种形式。
3. 变量的类型主要有_____、_____两种。
4. PL/SQL结构控制语句有_____、_____、_____。
5. 异常分为_____、_____、_____3种。



二、选择题

1. SELECT...INTO语句可以出现的异常是()。
A. CASE_NOT_FOUND B. NO_DATA_FOUND
C. DUP_VAL_ON_INDEX D. TOO_MANY_ROWS
2. 异常处理部分，匹配异常名称的子句是()。
A. IF...THEN子句 B. WHEN...THEN子句

三、简答题

1. 利用程序求出 $\sqrt{34+100/4}$ 的值。
2. 编写PL/SQL程序，自定义异常，当价格低于10时抛出自定义异常。
3. 编写自定义函数，实现功能和SQL内置函数SQRT一致。

第8章 游标——数据的缓存区

游标在操作数据库时是经常用到的。它使用相对灵活，容易理解 and 操作。本章将学习游标的相关知识。本章知识点如下：

- 什么是游标
- 游标的种类
- 如何创建游标
- 如何使用游标

8.1 什么是游标

游标的使用可以让用户像操作数组一样操作查询出来的数据集，这使得使用PL/SQL编程更加方便。实际上，它提供了一种从集合性质的结果中提取单条记录的手段。

8.1.1 游标的概念

可以将游标（Cursor）形象地看成一个变动的光标。它实际上是一个指针，它在一段Oracle存放数据查询结果集或数据操作结果集的内存中，这个指针可以指向结果集中的任何一条记录。这样就可以得到它所指向的数据了，但初始时它指向首记录。这种模型很像编程语言中的数组。

可以简单地理解游标为指向结果集记录的指针，利用游标可以返回它当前指向的行记录（只能返回一行记录）。如果要返回多行，那么需要不断地滚动游标，把想要的查询一遍。用户可以操作游标所在位置行的记录。例如，把返回记录作为另一个查询的条件等。

8.1.2 游标的种类

Oracle中游标分为静态游标和REF游标两类。其中，静态游标就像一个数据快照，打开游标后的结果集是对数据库数据的一个备份，数据不随着对表执行DML操作后而改变。从这个特性来说，结果集是静态的。由于篇幅问题，本章将不对REF游标做介绍。

静态游标包含如下两种类型：

- 显式游标：是指在使用之前必须有着明确的游标声明和定义，这样的游标定义会关联数据查询语句，通常会返回一行或多行。打开游标后，用户可以利用游标的位置对结果集进行检索，使之返回单一的行记录，用户可以操作此记录。关闭游标后，就不能再对结果集进行任何操作。显式游标需要用户自己写代码完成，一切由用户控制。
- 隐式游标：和显式游标不同，它被PL/SQL自动管理，也被称为SQL游标。由Oracle自动管理。该游标用户无法控制，但能得到它的属性信息。

8.2 显式游标

显式游标在PL/SQL编程当中有着重要的作用，通过显式游标用户可以操作返回的数据，使得一些在编程语言中复杂的功能变得更容易实现。

8.2.1 游标语法

既然要学习显式游标，就要学习创建它的语句。具体语法如下：

```
CURSOR cursor_name
[(parameter_name datatype, ...)]
IS select_statement;
```

【语法说明】

- CURSOR cursor_name：声明游标，cursor_name是游标的名称。
- parameter_name：参数名称。
- datatype：参数类型。
- select_statement：游标关联的SELECT语句，但该语句不能是SELECT...INTO...语句。

8.2.2 游标的使用步骤

显式游标的使用顺序可以明确地分成声明游标、打开游标、读取数据和关闭游标4个步骤。下面就具体学习这4个步骤。

(1) 声明游标

声明游标主要用来给游标命名并且使得游标关联一个查询。具体语句如下：

```
DECLARE CURSOR cursor_Name
IS SELECT_STATEMENT
```

(2) 打开游标

游标中任何对数据的操作都是建立在游标被打开的前提下。打开游标初始化了游标指针，游标一旦打开，其结果集都是静态的。也就是说，结果集此时不会反映出数据库中对数据进行的增加、删除、修改操作。具体语句如下：

```
OPEN cursor_Name
```

(3) 读取数据

读取数据要利用FETCH语句完成，它可以把游标指向位置的记录放入到PL/SQL声明的变量当中。它只能取出指针当前行的记录。正常情况下，FETCH要和循环语句一起使用，这样指针会不断前进，直到某个条件不符合要求而退出。使用FETCH时游标属性%ROWCOUNT会不断累加。具体语句如下：

```
FETCH cursor_Name INTO Record_Name
```

(4) 关闭游标

关闭某个名称的游标。此时释放资源，结果集中的数据将不能做任何操作。具体语句如下：

```
CLOSE cursor_Name
```

以上操作步骤可以在后面的示例中得到验证。

【示例1】创建一个简单游标并使用游标

下面创建一个名为pdct_cur的游标，游标以PRODUCTINFO表为查询表，并从游标中查询出相关数据。具体代码如下：



```
01 DECLARE
02     CURSOR pdct_cur
03     IS SELECT * FROM PRODUCTINFO ;
04
05     cur_prodrd productinfo%ROWTYPE;
06
07 BEGIN
08     OPEN pdct_cur;
09     FETCH pdct_cur INTO cur_prodrd;
10     DBMS_OUTPUT.PUT_LINE(cur_prodrd.productid || '-' || cur_prodrd.productname
11                             || '-' || cur_prodrd.productprice);
12     CLOSE pdct_cur;
13 END;
```

【代码解析】

- 第1~2行是游标的声明，声明了一个名为pdct_cur的游标。
- 第3行是游标关联的查询。
- 第5行表示声明了一个变量，该变量类型是基于表PRODUCTINFO的行对象。
- 第8行表示打开游标。
- 第9行表示利用FETCH语句从结果集中提取指针指向的当前行记录。
- 第10行表示输出结果并换行，脚本中输出了表PRODUCTINFO中的PRODUCTID、PRODUCTNAME、PRODUCTPRICE 3个字段的值。
- 第12行表示关闭游标。

【执行效果】

在PL/SQL Developer的SQL窗口中执行以上代码。执行结果见SQL窗口中的Output标签页面，打印输出结果如下：

```
0240010001-天堂伞-59
```

示例1让我们充分了解了显示游标的4个步骤，相信读者通过这个示例已对游标有了感性的认识。脚本中值得注意的地方是第5行，这种变量声明方式并不是唯一的。下面通过示例2演示另一种游标中的变量声明。

【示例2】创建游标并在游标中声明变量

该示例和上一个演示得到的结果是一样的，它的操作是从PRODUCTINFO表中取出PRODUCTID、PRODUCTNAME、PRODUCTPRICE这3个字段的值然后输出结果。

具体代码如下：

```
01 DECLARE
02     cur_productid varchar(10);
03     cur_productname productinfo.Productname%TYPE;
04     cur_productprice productinfo.Productprice%TYPE;
```



```

05
06 CURSOR pdct_cur
07 IS SELECT PRODUCTID,PRODUCTNAME,PRODUCTPRICE FROM PRODUCTINFO
08     WHERE ROWNUM = 1;
09
10 BEGIN
11     OPEN pdct_cur;
12     FETCH pdct_cur INTO cur_productid,cur_productname,cur_productprice;
13     DBMS_OUTPUT.PUT_LINE(cur_productid || '-' || cur_productname || '-' || cur_productprice);
14     CLOSE pdct_cur;
15 END;
```

【代码解析】

- 第2行表示声明一个变量，名称为cur_productid，类型是长度为10的varchar类型，此变量中存放的数据长度不得超过10。此种声明方式更多当做中间变量使用。
- 第3~4行也是声明变量，实现的功能和第2行一样，但有一点区别。第3~4行的变量类型分别和表PRODUCTINFO中PRODUCTNAME、PRODUCTPRICE字段类型一样。这样声明有个优点，那就是如果表PRODUCTINFO中的这两个字段类型改变了，那么游标里声明的这两个变量类型可以不用修改。第2行中的声明方法就没有这种效果。
- 第6行是声明出来的游标名称。
- 第7行是游标关联的查询语句，表示查询PRODUCTINFO表中PRODUCTID、PRODUCTNAME、PRODUCTPRICE这3个字段。
- 第10行和第14行对应，可以保证代码的完整性。
- 第11行表示打开游标。
- 第12行表示利用FETCH从结果集中提取数据放到3个变量中。注意：这里给变量赋值的顺序和游标关联查询字段的顺序是一致的。
- 第13行表示输出cur_productid变量里的内容并换行。
- 第14行表示关闭游标。

【执行效果】

在PL/SQL Developer的SQL窗口中执行以上代码。执行结果见SQL窗口中的Output标签页面，打印输出结果如下：

0240010001-天堂伞-59

此脚本的输出结果和示例1得到的结果没有区别，但操作步骤并不完全一样。该脚本声明变量使用了%TYPE，它和%ROWTYPE不同。下面把这两者的区别简单介绍一下：

- %TYPE用在变量的声明里，使用它可以取得表中的字段类型。
- %ROWTYPE可以声明基于某个表的行类型。

以上两者虽然可以使变量的类型和表的字段或行类型一致，但这仅仅是类型的一致，而字段约束却不包含在内。通过这两个示例可以发现，其中的细节部分虽然不一样，但都存在在本小节开始所述的4个步骤。

8.2.3 游标中的LOOP语句

通常显式游标提取的数据不会只有一条，而是多条记录。这样就需要一个遍历结果集的方法



式，而LOOP语句就能实现该功能。下面示例将演示在游标中如何使用LOOP语句。

【示例3】在游标中使用LOOP语句

具体代码如下：

```

01 DECLARE
02 CURSOR pdct_loop_cur
03 IS SELECT PRODUCTID,PRODUCTNAME,PRODUCTPRICE FROM PRODUCTINFO
04 WHERE PRODUCTPRICE > 2500;
05
06 cur_productid productinfo.Productid%TYPE;
07 cur_productname productinfo.Productname%TYPE;
08 cur_productprice productinfo.Productprice%TYPE;
09
10 BEGIN
11 OPEN pdct_loop_cur;
12 LOOP
13     FETCH pdct_loop_cur INTO cur_productid,cur_productname,cur_productprice;
14     EXIT WHEN pdct_loop_cur%NOTFOUND;
15     DBMS_OUTPUT.PUT_LINE('产品ID: ' || cur_productid || ' 产品名称: ' ||
16     cur_productname || ' 产品价格: ' ||
17     cur_productprice);
18 END LOOP;
19 CLOSE pdct_loop_cur;
20 END;
```

【代码解析】

- 代码第1~4行表示声明游标并关联查询，查询出价格大于2500的产品。
- 第6~8行声明变量，各变量类型同表PRODUCTINFO的对应字段类型一致。
- 第12~18行就是使用的LOOP语句，利用该语句可以遍历结果集。
- 第13行利用FETCH...INTO...语句把取出的值放进变量中。
- 第14行利用游标属性实现没有记录退出循环。
- 第15~17行输出信息。
- 第19行关闭游标。

【执行效果】

在PL/SQL Developer的SQL窗口中执行以上代码。执行结果见SQL窗口中的Output标签页面，打印输出结果如下：

```

产品ID: 0240040001  产品名称: 夏普LCD-46G100A  产品价格: 7000
产品ID: 0240030002  产品名称: 三星WF-R1065S/XSC  产品价格: 3600
产品ID: 0240020001  产品名称: 思科RV082路由器  产品价格: 2800
```

和示例1做对比，该脚本执行后把所有符合条件的记录全部输出，这就是LOOP语句的作用，有了它用户就可以得到符合要求的每一条数据，并可以对数据进行操作，实现诸多复杂的数据转换，以达到满足业务需求的目的。

8.2.4 使用BULK COLLECT和FOR语句的游标

游标中通常使用FETCH...INTO...语句提取数据，这种方式是单条数据提取，在数据量很



大的情况下执行效率不是很理想。而FETCH...BULK COLLECT INTO语句可以批量提取数据，在数据量大的情况下它的执行效率比单条提取数据的高。

此语句的用法可以参考如下示例。

【示例4】利用BULK COLLECT方式批量提取数据

具体代码如下：

```
01 DECLARE
02 CURSOR pdct_collect_cur
03 IS SELECT * FROM PRODUCTINFO ;
04
05 TYPE PDCT_TAB IS TABLE OF PRODUCTINFO%ROWTYPE;
06 pdct_rd PDCT_TAB;
07
08 BEGIN
09 OPEN pdct_collect_cur;
10 LOOP
11
12     FETCH pdct_collect_cur BULK COLLECT INTO pdct_rd LIMIT 2 ;
13     FOR i in 1..pdct_rd.count LOOP
14         DBMS_OUTPUT.PUT_LINE('产品ID: ' || pdct_rd(i).productid || ' 产品名称: ' ||
15                               pdct_rd(i).productname || ' 产品价格: ' ||
16                               pdct_rd(i).productprice);
17     END LOOP;
18     EXIT WHEN pdct_collect_cur%NOTFOUND;
19
20 END LOOP;
21 CLOSE pdct_collect_cur;
22 END;
```

【代码解析】

- 第1~3行声明游标名称，关联查询语句。
- 第5~6行表示的是定义和表PRODUCTINFO 行对象一致的集合类型pdct_rd，该变量用于存放批量得到的数据。
- 第10行和第20行对应，这里是迭代从结果集取数据。
- 第12行表示从结果集批量提取数据，每次提取两条。
- 第13行表示遍历集合对象pdct_rd中的数据，该行的LOOP与第17行对应。
- 第14~16行表示输出结果。
- 第18行判断游标是否到最尾端。

【执行效果】

在PL/SQL Developer的SQL窗口中执行以上代码。执行结果见SQL窗口中的Output标签页面，打印输出结果如下：

```
产品ID: 0240010001  产品名称: 天堂伞  产品价格: 59
产品ID: 0240040001  产品名称: 夏普LCD-46G100A  产品价格: 7000
产品ID: 0240030001  产品名称: 海尔XQB50-918A  产品价格: 1100
产品ID: 0240030002  产品名称: 三星WF-R1065S/XSC  产品价格: 3600
```


产品ID: 0240030003 产品名称: 三星XQB55-T86A/XSC 产品价格: 2500

产品ID: 0240050001 产品名称: 台电C430T 产品价格: 400

产品ID: 0240020001 产品名称: 思科RV082路由器 产品价格: 2800

读书笔记

注意 判断游标是否是到尾端的代码一定不能放到第13行处, 否则会出现丢失数据现象。其中的原理读者可以自己思考。

8.2.5 使用CURSOR FOR LOOP

游标很多机会都是迭代结果集, 在PL/SQL这个过程中可以使用更简单的方式实现, CURSOR FOR LOOP不需要特别的声明变量, 它可以提出行对象类型的数据。可以按照类似示例4中提取数据的方式得到列数据。详细脚本见示例5。

【示例5】使用CURSOR FOR LOOP语句提取数据

具体代码如下:

```
01 DECLARE
02     CURSOR cfl IS SELECT productname, productprice FROM PRODUCTINFO
03                     WHERE productprice > 1200;
04 BEGIN
05     FOR curefl IN cfl
06     LOOP
07         DBMS_OUTPUT.PUT_LINE('名称: ' || curefl.productname
08                               || ' 产品价格: ' || curefl.productprice);
09     END LOOP;
10 END;
```

【代码解析】

□ 第1~3行声明游标并关联查询。

□ 第5行把游标返回数据放到curefl中, 该类型是个%ROWTYPE类型。

□ 第6~9行迭代输出数据。

获取数据的方式和示例4大同小异, 相信读者理解这部分比较容易。这里不详细介绍了。

【执行效果】

执行效果如下:

名称: 夏普LCD-46G100A 产品价格: 7000

名称: 三星WF-R1065S/XSC 产品价格: 3600

名称: 三星XQB55-T86A/XSC 产品价格: 2500

名称: 思科RV082路由器 产品价格: 2800

这种方式在隐式游标中使用更更方便, 如果没有特殊需求, 可以尝试使用该语句。

8.2.6 显式游标的属性

利用游标属性可以得到游标执行的相关信息。显式游标有以下4个属性:

□ %ISOPEN: 用于判断游标是否打开, 如果已经打开则返回TRUE, 如果游标未打开则返回FALSE。

□ %FOUND: 此属性可用来检测行数据是否有效。如果有效该属性返回TRUE, 否则返



回FALSE。

□ %NOTFOUND: 与%FOUND属性恰好相反, 如果没有提取出数据则返回TRUE, 否则返回FALSE。

□ %ROWCOUNT: 累计到当前为止使用FETCH提取数据的行数。

下面利用几个示例来了解以上4个属性的具体使用方法。

1) 要想获取游标的数据, 首要一点就是游标是打开的, 如果在游标关闭状态下操作数据则会出现错误。如果在游标未打开时使用FETCH语句会提示错误。

【示例6】%ISOPEN的使用

具体使用方法如下:

```
01 DECLARE
02 CURSOR pdct_isopen_cur
03 IS SELECT * FROM PRODUCTINFO ;
04
05 cur_prodrdc productinfo%ROWTYPE;
06
07 BEGIN
08 IF pdct_isopen_cur%ISOPEN THEN
09     FETCH pdct_isopen_cur INTO cur_prodrdc;
10     DBMS_OUTPUT.PUT_LINE('产品ID: ' || cur_prodrdc.productid
11     || ' 产品名称: ' || cur_prodrdc.productname
12     || ' 产品价格: ' || cur_prodrdc.productprice);
13
14 ELSE
15     DBMS_OUTPUT.PUT_LINE('游标pdct_isopen_cur没有打开');
16 END IF;
17 END;
```

【代码解析】

□ 第8、14行利用游标属性%ISOPEN判断如果游标已经打开, 则执行THEN后面的语句, 否则执行ELSE后面的语句。

□ 第16行是IF语句的结束标志。

【执行效果】

脚本执行效果如下:

游标pdct_isopen_cur没有打开

由执行效果可以看出脚本走的是ELSE这个流程, 说明此属性判断游标是否打开没有问题。

2) 在FETCH提取数据的时候, 通常开发者都会判断行数据的有效性, 这样可以避免脚本出现技术性的错误。

【示例7】%FOUND的使用

具体使用方法如下:

```
01 DECLARE
02 CURSOR pdct_found_cur
03 IS SELECT * FROM PRODUCTINFO ;
04
```



```

05  cur_prodrd productinfo%ROWTYPE;
06
07  BEGIN
08      OPEN pdct_found_cur;
09      LOOP
10          FETCH pdct_found_cur INTO cur_prodrd;
11          IF pdct_found_cur%FOUND THEN
12              DBMS_OUTPUT.PUT_LINE('产品ID: ' || cur_prodrd.productid
13              || ' 产品名称: ' || cur_prodrd.productname
14              || ' 产品价格: ' || cur_prodrd.productprice);
15          ELSE
16              DBMS_OUTPUT.PUT_LINE('没有数据被提取');
17              EXIT;
18          END IF;
19      END LOOP;
20      CLOSE pdct_found_cur;
21  END;

```

【代码解析】

- 整个脚本在提取数据时做了判断，如果数据存在则输出，否则提示“没有数据被提取”然后退出。
- 第19行表示遍历结果集。
- 第11行用到%FOUND属性，表示如果该游标的指向有效则执行第12~14行输出脚本。
- 第11行的判断如果返回FALSE，则执行第15~17行。第17行的EXIT表示退出循环，否则当游标指向无效时重复执行第16行。

【执行效果】

参考以下执行效果可以更好地理解SQL脚本的含义。效果如下：

```

产品ID: 0240010001  产品名称: 天堂伞  产品价格: 59
产品ID: 0240040001  产品名称: 夏普LCD-46G100A  产品价格: 7000
产品ID: 0240030001  产品名称: 海尔XQB50-918A  产品价格: 1100
产品ID: 0240030002  产品名称: 三星WF-R1065S/XSC  产品价格: 3600
产品ID: 0240030003  产品名称: 三星XQB55-T86A/XSC  产品价格: 2500
产品ID: 0240050001  产品名称: 台电C430T  产品价格: 400
产品ID: 0240020001  产品名称: 思科RV082路由器  产品价格: 2800
没有数据被提取

```

这里注意一下执行结果中的最后一行，如果脚本的第16行放在第17行以后，则输出结果将不包含“没有数据被提取”这句话。

属性%NOTFOUND与%FOUND的含义正好相反，它的使用方式可以参考示例3的第14行，这里不做讲解。

3) %ROWCOUNT也是比较常用的属性，利用它可以知道当前已经返回了多少行数据。

【示例8】%ROWCOUNT的使用

具体使用方法如下：

```

01  DECLARE
02  CURSOR pdct_rowcount_cur

```



```

03  IS SELECT * FROM PRODUCTINFO ;
04
05  TYPE PDCT_TAB IS TABLE OF PRODUCTINFO%ROWTYPE;
06  pdct_count_rd PDCT_TAB;
07
08  BEGIN
09      OPEN pdct_rowcount_cur;
10      LOOP
11
12          FETCH pdct_rowcount_cur BULK COLLECT INTO pdct_count_rd LIMIT 2 ;
13          FOR i IN pdct_count_rd.first..pdct_count_rd.last LOOP
14              DBMS_OUTPUT.PUT_LINE('产品ID: ' || pdct_count_rd(i).productid || ' 产品名称: ' ||
15                  pdct_count_rd(i).productname || ' 产品价格: ' ||
16                  pdct_count_rd(i).productprice);
17          END LOOP;
18          IF mod(pdct_rowcount_cur%ROWCOUNT,2) = 0 THEN
19              DBMS_OUTPUT.PUT_LINE('读取到了第'
20                  || pdct_rowcount_cur%ROWCOUNT
21                  || '条记录!');
22          ELSE
23              DBMS_OUTPUT.PUT_LINE('读取到单条记录为第'
24                  || pdct_rowcount_cur%ROWCOUNT
25                  || '条记录!');
26          END IF;
27          EXIT WHEN pdct_rowcount_cur%NOTFOUND;
28
29      END LOOP;
30      CLOSE pdct_rowcount_cur;
31  END;

```

【代码解析】

- ☐ 第1~12行可以参考示例4自行理解。
- ☐ 第13行表示对批量读取的数据从第一条到最后一条进行遍历。
- ☐ 第18行表示用条件语句进行判断，如果提取出来的数据行数能被2整除就进入THEN后面的输出脚本，如果提取的数据行数不能被2整除则进入第22行ELSE后面的输出脚本。其中，第18行的mod()是取余函数，表示pdct_rowcount_cur%ROWCOUNT对2取余。
- ☐ 第26行是IF语句的结束标志。
- ☐ 第27行利用%NOTFOUND属性判断游标是否走到尾端。此时如果没有发现可用的数据则利用EXIT退出。

【执行效果】

执行效果如下：

```

产品ID: 0240010001  产品名称: 天堂伞  产品价格: 59
产品ID: 0240040001  产品名称: 夏普LCD-46G100A  产品价格: 7000
读取到了第2条记录!
产品ID: 0240030001  产品名称: 海尔XQB50-918A  产品价格: 1100
产品ID: 0240030002  产品名称: 三星WF-R1065S/XSC  产品价格: 3600
读取到了第4条记录!
产品ID: 0240030003  产品名称: 三星XQB55-T86A/XSC  产品价格: 2500

```

产品ID: 0240050001 产品名称: 台电C430T 产品价格: 400

读取到了第6条记录!

产品ID: 0240020001 产品名称: 思科RV082路由器 产品价格: 2800

读取到单条记录为第7条记录!

读书笔记

从执行效果中可以看出脚本每次都取出两条记录。由于该表一共7条数据,所以最后一次取出了单条记录,而脚本流程也发生了变化。

以上是对显式游标4个属性的简单介绍,通常可以利用这4个属性对脚本进行流程控制。实用性比较强,希望读者对其有更深入的了解。

8.2.7 带参数的游标

在使用显式游标时是可以指定参数的,指定的参数包括参数的顺序和参数的类型。参数可以传递给游标在查询中使用,这样就方便了用户根据不同的查询条件进行查询,也方便了游标在存储过程中的使用。本小节将介绍带参数的游标。

【示例9】使用带参数的游标

此类型的游标语法可以参考8.2.1小节。具体代码如下:

```
01 DECLARE
02   cur_productid productinfo.Productid%TYPE := '0240';
03   cur_productprice productinfo.Productprice%TYPE := 1200;
04   cur_prodrdc productinfo%ROWTYPE;
05
06   CURSOR pdct_parameter_cur (id VARCHAR, price NUMBER)
07   IS SELECT * FROM PRODUCTINFO
08   WHERE productid like id || '%'
09   AND productprice > price;
10
11 BEGIN
12   OPEN pdct_parameter_cur(cur_productid, cur_productprice);
13   LOOP
14     FETCH pdct_parameter_cur INTO cur_prodrdc;
15     EXIT WHEN pdct_parameter_cur%NOTFOUND;
16     DBMS_OUTPUT.PUT_LINE('产品ID: ' || cur_prodrdc.productid || ' 产品名称: ' ||
17       cur_prodrdc.productname || ' 产品价格: ' ||
18       cur_prodrdc.productprice);
19   END LOOP;
20   CLOSE pdct_parameter_cur;
21 END;
```

【代码解析】

- ☐ 第2~3行声明变量并赋值,这两个参数是要传递给游标的变量。
- ☐ 第4行声明表productinfo的行对象,用来存放FETCH提取的数据。
- ☐ 第6行声明游标,包括两个参数,参数需要说明类型。
- ☐ 第7~9行是游标关联的查询语句,可以看到第8和第9行的查询条件都使用了游标里的变量。
- ☐ 第12行表示打开游标,并把第2和第3行的变量传入游标中。
- ☐ 其他脚本这里不做解释,读者可参照前面代码自行理解。



【执行效果】

执行效果如下：

```
产品ID: 0240040001 产品名称: 夏普LCD-46G100A 产品价格: 7000
产品ID: 0240030002 产品名称: 三星WF-R1065S/XSC 产品价格: 3600
产品ID: 0240030003 产品名称: 三星XQB55-T86A/XSC 产品价格: 2500
产品ID: 0240020001 产品名称: 思科RV082路由器 产品价格: 2800
```

从输出结果可以看出，得到的数据是经过游标中条件过滤的。这种带参数的游标在实际开发中经常用到，有时候也会根据参数的不同而打开不同的游标。这种情况会在存储过程中出现。

注意

当打开游标时并不单局限示例中的方式，也可以直接写入具体的数值作为参数。当然，也可以变量同具体数值混合使用。关于这一点做过编程语言开发的不会陌生。

8.3 隐式游标

隐式游标和显式游标有所差异，它虽然没有显式游标一样的可操作性，但在实际的工作当中也经常用到。本节将介绍隐式游标的内容。

8.3.1 隐式游标的特点

每当运行SELECT或DML语句时，PL/SQL会打开一个隐式的游标。隐式游标不受用户的控制，这一点和显式游标有明显的不同。下面列出了隐式游标和显式游标的不同处：

- ☐ 隐式游标由PL/SQL自动管理；
- ☐ 隐式游标的默认名称是SQL；
- ☐ SELECT或DML操作产生隐式游标；
- ☐ 隐式游标的属性值始终是最新执行的SQL语句的。

【示例10】隐式游标的使用

具体代码如下：

```
01 DECLARE
02     cur_productname productinfo.Productname%TYPE;
03     cur_productprice productinfo.Productprice%TYPE;
04 BEGIN
05     SELECT productname, productprice INTO cur_productname, cur_productprice
06     FROM PRODUCTINFO
07     where productid = '0240040001';
08     IF SQL%FOUND THEN
09         DBMS_OUTPUT.PUT_LINE('产品名称: ' || cur_productname
10         || ' 产品价格: ' || cur_productprice);
11     END IF;
12 END;
```

【代码解析】

- ☐ 第1~3行声明变量，这部分和显式游标没有区别。
- ☐ 第5~7行把查询的数据放进两个变量中。



- 第8行利用游标属性进入条件语句。关于隐式游标的属性后面会单独做介绍。
- 第9~10行输出结果。
- 第11行为IF语句的介绍标志。
- 第12行是BEGIN结束的标志。

此示例演示了如何使用隐式游标。通过脚本可以看出，隐式游标没有像显式游标那样声明一个游标名称，而是直接使用了SQL名称。这是隐式游标的默认名称，可以直接使用。可以利用下面的执行效果了解隐式游标的工作流程。

【执行效果】

执行效果如下：

产品名称：夏普LCD-46G100A 产品价格：7000

注意

SELECT INTO语句一定保证返回一条记录，如果返回多条记录，在PL/SQL Developer工具中会提示“实际返回的行数超出请求的行数”；如果没有返回记录，那么会提示“未找到任何数据”。如果要想在出现这两种异常时不提示如上语句，则需要在脚本中加入异常处理部分，然后在出现异常时按照用户自己定义的流程继续执行脚本。

8.3.2 游标中使用异常处理

使用游标时，某些情况下得到的数据超出了控制范围，如果不加处理会出现脚本执行中断的情况。这种情况下，脚本开发者通常会使用异常处理来维护脚本的稳定性。可以参考示例11来增加这方面的知识。

【示例11】在游标中使用异常处理

前面介绍过当数据库中的数据记录发生变化时使用SELECT INTO语句返回的结果有可能不是单条记录（可能空，也可能多条），这时会出现脚本中断并报错的现象。为了避免这种硬性的中断，可以使用异常处理。把示例10稍作修改，具体代码如下：

```
01 DECLARE
02     cur_productname productinfo.Productname%TYPE;
03     cur_productprice productinfo.Productprice%TYPE;
04 BEGIN
05     SELECT productname, productprice INTO cur_productname, cur_productprice
06     FROM PRODUCTINFO
07     where productid = '02400400012' ;
08     IF SQL%FOUND THEN
09         DBMS_OUTPUT.PUT_LINE('产品名称: ' || cur_productname
10         || ' 产品价格: ' || cur_productprice);
11     END IF ;
12     EXCEPTION
13     WHEN NO_DATA_FOUND THEN
14         DBMS_OUTPUT.PUT_LINE('没有数据: ');
15     WHEN TOO_MANY_ROWS THEN
16         DBMS_OUTPUT.PUT_LINE('数据过多: ');
17
18 END;
```

**【代码解析】**

- 第1~11行参考示例10的说明。
- 第12行表示异常处理的关键词。
- 第13行表示如果SELECT INTO返回记录为空时会出现NO_DATA_FOUND异常，此时脚本会执行第14行的内容。
- 第15行表示如果SELECT INTO返回多条记录时会出现TOO_MANY_ROWS异常，此时脚本执行第16行的内容。

【执行效果】

由于表PRODUCTINFO没有productid值是02400400012的数据，所以它会引发异常。执行效果如下：

没有数据！

通过异常处理可以有效避免意外引起的脚本错误，在使用游标时需要注意这一点。

8.3.3 隐式游标的属性

隐式游标的属性和显式游标的属性具体表示含义有区别，但属性种类没有变。下面列出隐式游标的属性。

- %ISOPEN属性：该属性永远返回FALSE，它由Oracle自己控制。
- %FOUND属性：此属性可以反应DML操作是否影响到了数据，当DML操作对数据有影响时该属性为TRUE，否则为FALSE。也可以反映出SELECT INTO语句是否返回了数据，当有数据返回时该属性为TURE。
- %NOTFOUND属性：与%FOUND属性相反，当DML操作没有影响数据以及SELECT INTO没有返回数据时该属性为TRUE，其他为FALSE。
- %ROWCOUNT属性：该属性可以反映出DML操作对数据影响的数量。

下面利用几个示例来了解以上4个属性的具体使用方法。

1) 隐式游标中的%ISOPEN属性和显式游标中的该属性不同，隐式游标该属性永远为FALSE。

【示例12】隐式游标中%ISOPEN的使用

下面的脚本验证了%ISOPEN属性返回FALSE的特性。

```
01 DECLARE
02 BEGIN
03     DELETE FROM PRODUCTINFO ;
04     IF SQL%ISOPEN THEN
05         DBMS_OUTPUT.PUT_LINE('游标打开！');
06     ELSE
07         DBMS_OUTPUT.PUT_LINE('游标未打开！');
08     END IF;
09 END;
```

【执行效果】

以上脚本执行后输出结果如下：

游标未打开！

由执行结果可以看出此属性的值是FALSE。当第3行执行完成后它由Oracle自动关闭。

2) %FOUND属性在INSERT、UPDATE、DELETE执行对数据有影响时会返回TRUE，而SELECT INTO语句只要有数据返回，该属性就为TRUE。

【示例13】隐式游标中%FOUND的使用

具体代码如下：

```
01 DECLARE
02     cur_productname productinfo.Productname%TYPE;
03     cur_productprice productinfo.Productprice%TYPE;
04 BEGIN
05     SELECT productname, productprice INTO cur_productname, cur_productprice
06     FROM PRODUCTINFO ;
07
08     EXCEPTION
09     WHEN TOO_MANY_ROWS THEN
10         IF SQL%FOUND THEN
11             DBMS_OUTPUT.PUT_LINE('%FOUND为TRUE');
12             DELETE FROM PRODUCTINFO WHERE productid = '00000000';
13             IF SQL%FOUND THEN
14                 DBMS_OUTPUT.PUT_LINE('删除数据: ');
15             END IF;
16         END IF ;
17
18 END;
```

【代码解析】

- 第1~4行读者理解应该不会有问题，这里不再介绍。
- 第5~6行利用SELECT INTO语句向变量中保存数据，但此语句会返回多条数据，也就是说会引发异常。
- 第9行是此语句可能引发的异常，当返回多条数据时会出现TOO_MANY_ROWS异常，脚本会进入THEN后面的代码流程。
- 第10行在发生TOO_MANY_ROWS异常时检测%FOUND是否为TRUE，如果为TRUE则执行THEN后面的脚本。
- 第12行表示当%FOUND为TRUE时，执行该删除脚本。
- 第13行继续判断该删除脚本产生的游标的%FOUND属性，如果为TRUE，则会执行第14行脚本。
- 第15~18行读者可以自行找到其对应的开始标识。

【执行效果】

执行效果如下：

%FOUND为TRUE

由此脚本的执行效果可以看出该属性的使用方法以及特性，第12行的删除语句由于数据库中符合WHERE后的条件的记录，所以它并没有影响任何数据，此时的%FOUND为FALSE，第14行脚本也就没有得到执行。





注意

在SELECT INTO语句中%FOUND不会因语句是否发生异常而改变，只要有返回值该属性就为TRUE。但有异常发生时，执行流程会马上发生改变。也就是说，在异常代码外检查该属性有可能得不到有效执行。第13行的%FOUND是第12行删除语句游标的属性，这一点读者需要注意。

3) %NOTFOUND属性和%FOUND属性在逻辑上是相反的，这里不再给出具体的示例。读者把%FOUND属性的示例稍作修改即可。

4) 与显式游标不同的是隐式游标中%ROWCOUNT属性反映了DML操作影响的数据数量，而SELECT INTO语句如果发生TOO_MANY_ROWS异常，那么此属性依然是1，而不是实际符合要求的记录数。

【示例14】隐式游标中%ROWCOUNT的使用

具体代码如下：

```
01 DECLARE
02     cur_productname productinfo.Productname%TYPE;
03     cur_productprice productinfo.Productprice%TYPE;
04     cur_count varchar(8);
05 BEGIN
06     SELECT productname, productprice INTO cur_productname, cur_productprice
07     FROM PRODUCTINFO ;
08
09     EXCEPTION
10     WHEN NO_DATA_FOUND THEN
11         DBMS_OUTPUT.PUT_LINE(SQL%ROWCOUNT);
12         DBMS_OUTPUT.PUT_LINE('没有数据！');
13     WHEN TOO_MANY_ROWS THEN
14         cur_count:=SQL%ROWCOUNT;
15         DBMS_OUTPUT.PUT_LINE('SQL%ROWCOUNT值是：' || cur_count);
16         UPDATE PRODUCTINFO SET PRODUCTINFO.productname = '';
17         DBMS_OUTPUT.PUT_LINE('修改脚本影响记录数：' || SQL%ROWCOUNT);
18         ROLLBACK;
19         DBMS_OUTPUT.PUT_LINE('脚本回滚后：' || SQL%ROWCOUNT);
20     END;
```

【代码解析】

- ☐ 此脚本运行时会出现TOO_MANY_ROWS异常，然后进入异常块内部。
- ☐ 第4行声明一个类型为varchar类型的变量，用此变量存放%ROWCOUNT的值。
- ☐ 第15行表示发生异常时把%ROWCOUNT的值保存到变量cur_count中，通过第16行的输出可以测试保存是否成功。
- ☐ 第17行修改表PRODUCTINFO中数据。
- ☐ 第18行输出修改语句影响的记录数。
- ☐ 第19行事务回滚。
- ☐ 第20行输出事务回滚后%ROWCOUNT属性值。

【执行效果】

执行效果如下：

SQL%ROWCOUNT值是：1

修改脚本影响记录数：7

修改脚本影响记录数：0



读书笔记

通过以上执行效果可以看到，使用变量cur_count保存%ROWCOUNT值是成功的，开发过程中用户经常使用此方式保留%ROWCOUNT值，以便控制流程时使用。而当SELECT INTO语句发生TOO_MANY_ROWS异常时，%ROWCOUNT属性依然返回1条记录，而不是实际的记录数目。这是需要注意的。

注意 %ROWCOUNT属性和事务没有关系，即使事务回滚，它的值也不会变成上次操作的值，这可以从第20行的输出结果看出来。

8.4 有关游标的案例

本节将完成一个有关游标的案例，加强读者的动手能力，以达到更加熟练地运用游标解决问题的目的。

本案例涉及两张表，这两张表在第4章已经设计完成，分别是PRODUCTINFO（产品信息表）和CATEGROYINFO（产品类型信息表）。下面列出这两张表的数据记录供读者参考。其中，图8.1为PRODUCTINFO表的数据列表，图8.2为表CATEGROYINFO的数据列表。

CATEGORYID	DEPT	PRODUCTID	PRODUCTNAME	QUANTITY	CATEGORY	DESCRIPTION	ORIGIN
1	0240010001	天香	50	0100010001		玫瑰	
2	0240040001	夏露·CD-490188A	7000	00	0100030001		日本
3	0240010001	黑玫瑰·118A	1100	00	0100030002		中国
4	0240010002	黑玫瑰·1180000000	3600	00	12	0100030002	中国
5	0240010003	黑玫瑰·1180000000	2500	00	111	0100030002	中国
6	0240010001	黑玫瑰·1180000000	400	00	129	0100040001	中国
7	0240010001	黑玫瑰·1180000000	2800	00	27	0100020001	

图8.1 PRODUCTINFO表数据列表

【示例15】利用游标转换这两张表的数据，要求把商品价格高于1000的产地标为“中国”和“杭州”的家电和电子产品放入到表PRODUCTINFO_TMP（该表字段同PRODUCTINFO一致，字段类型可根据实际需要自行定义）中，并且要求商品类型编号换成商品类型，商品价格高于2000的下调5%。

【案例分析】

- ☐ 要建表PRODUCTINFO_TMP。
- ☐ 利用SQL语句把符合要求的数据查询出来。

CATEGORYID	CATEGORYNAME	ROWID
1	玩具	AAAF0AABAAVAAA
2	宠物食品	AAAF0AABAAVAAA
3	宠物	AAAF0AABAAVAAA
4	宠物食品	AAAF0AABAAVAAA
5	宠物食品	AAAF0AABAAVAAA

图8.2 CATEGORYINFO表数据列表

- ☐ 把符合要求的数据放进新表。
- ☐ 把新表的数据商品类型编号换成商品类型。
- ☐ 商品价格下调。

【案例脚本】

按照分析步骤写成脚本如下。

(1) 建表PRODUCTINFO_TMP

首先创建表PRODUCTINFO_TMP。由于要求该表和PRODUCTINFO表字段一致，而且经过观察CATEGORY字段的长度可以暂时容纳表CATEGORYINFO中CATEGORYNAME字段内容，这里不对新表CATEGORY字段长度做修改。利用如下语句完成创建：

```
01 CREATE TABLE productinfo_tmp AS SELECT * FROM productinfo
02 WHERE 1=0
```

【代码解析】

- ☐ 该脚本表示创建结构和表productinfo一样的新表productinfo_tmp。
- ☐ 第2行表示复制表的时候不包括数据。如果WHERE后的条件为TRUE，则复制表的时候会把原表的数据一同复制。

(2) 创建游标完成案例要求

以下是按照案例要求完成的脚本，脚本比较多，读者可以仔细阅读代码解析部分。

```
01 DECLARE
02
03   cur_categoryid categoryinfo.categoryid%TYPE;
04   cur_categoryname categoryinfo.categoryname%TYPE;
05   cur_prodrctd productinfo%ROWTYPE;
06   tmpnum number(8,0);
07
08   CURSOR cur_prdt_catg IS
```



```

09  SELECT * FROM productinfo WHERE productprice >1000 AND origin in('中国','杭州')
10  AND category IN
11  (SELECT categoryid
12   FROM categoryinfo
13   WHERE categoryname
14   IN('路由器','电视','洗衣机','MP3'))
15  );
16
17  CURSOR cur_catg IS
18  SELECT CATEGORYID,CATEGORYNAME FROM categoryinfo
19  WHERE categoryname
20  IN('路由器','电视','洗衣机','MP3');
21
22  BEGIN
23
24  ----把符合要求数据放进表productinfo_tmp
25  OPEN cur_prdt_catg;
26  LOOP
27      FETCH cur_prdt_catg INTO cur_prodrctd;
28      IF cur_prdt_catg%FOUND THEN
29          INSERT INTO productinfo_tmp
30              (productid,productname,productprice,
31               quantity,category,desperation,origin)
32              VALUES
33              (cur_prodrctd.productid,cur_prodrctd.PRODUCTNAME,
34               cur_prodrctd.PRODUCTPRICE,cur_prodrctd.QUANTITY,
35               cur_prodrctd.CATEGORY,cur_prodrctd.DESPERATION,
36               cur_prodrctd.ORIGIN);
37
38      ELSE
39          DBMS_OUTPUT.PUT_LINE('已取出所有数据! 共' || cur_prdt_catg%ROWCOUNT || '条记录');
40      EXIT;
41  END IF;
42  END LOOP;
43  COMMIT;
44
45  ----转换产品类型
46  OPEN cur_catg;
47  tmpnum := 0;
48  LOOP
49      FETCH cur_catg INTO cur_categoryid,cur_categoryname;
50      IF cur_catg%FOUND THEN
51          UPDATE productinfo_tmp SET productinfo_tmp.category = cur_categoryname
52          WHERE category = cur_categoryid;
53          IF SQL%FOUND THEN
54              tmpnum := tmpnum+SQL%ROWCOUNT;
55          END IF;
56      ELSE
57          DBMS_OUTPUT.PUT_LINE('产品类型转换完毕! 共转换' || tmpnum || '条记录');

```



```

58      EXIT;
59    END IF;
60  END LOOP;
61
62  ----产品价格下调
63  UPDATE productinfo_tmp
64  SET productinfo_tmp.productprice = productinfo_tmp.productprice*0.95
65  WHERE productinfo_tmp.productprice > 2000;
66  DBMS_OUTPUT.PUT_LINE('价格下调完毕！共下调' || SQL%ROWCOUNT || '条商品');
67  COMMIT;
68  END;

```

【代码解析】

- ☐ 第3~4行表示声明变量，类型同表字段类型一致。
- ☐ 第5行声明一个行对象类型的变量。
- ☐ 第6行声明一个number类型的变量。
- ☐ 第8~15行表示创建游标得到商品符合价格高于1000的产地标为“中国”和“杭州”的家电和电子产品的数据。
- ☐ 第11~15行表示查询出categoryname字段符合IN括号里面条件的数据，此查询的结果将作为第9~10行的条件。
- ☐ 第17~20行表示创建游标得到表CATEGORYINFO中家电和电子产品的产品编码和产品类型。
- ☐ 第25~27行表示打开游标cur_prdt_catg并进入循环流提取数据。
- ☐ 第28行表示如果%FOUND属性为TRUE，则进入存放数据语句。
- ☐ 第29~36行表示把符合要求的数据插入新表PRODUCTINFO_TMP中。
- ☐ 第38~39行表示当没有符合要求的数据时会进入该流程，它利用游标的%ROWCOUNT属性进行统计共输入新表记录数。EXIT表示退出该游标。
- ☐ 第43行表示提交事务。
- ☐ 第46~60行表示打开游标cur_catg，完成把PRODUCTINFO_TMP表中的产品类型编码改成产品类型名称功能。
- ☐ 第47行为变量tmpnum赋初始值0。
- ☐ 第48~50行表示进入循环流取数据，并利用%FOUND属性判断数据是否提取完毕。
- ☐ 第51~52行对表PRODUCTINFO_TMP中的数据进行修改，把产品类型编码修改为产品类型名称。
- ☐ 第53~55行是比较有趣的地方。这里利用了隐式游标的%FOUND属性判断修改了多少条数据，并利用tmpnum变量和隐式游标的%ROWCOUNT属性最终得到转换数据的数量。
- ☐ 第57行表示输出最终转换的数据数目。
- ☐ 第63~65行完成产品价格下调功能。
- ☐ 第66行表示利用隐式游标的%ROWCOUNT属性得到价格下调成功产品的数量。
- ☐ 第67行表示事务提交。

【执行效果】

在PL/SQL Developer的SQL窗口中执行以上代码。执行效果见SQL窗口中的Output标签页面，打印输出结果如下：

已取出所有数据：共4条记录
 产品类型转换完毕：共转换4条记录
 价格下调完毕：共下调3条商品

最后表PRODUCTINFO_TMP中的内容如图8.3所示，读者可以自行同PRODUCTINFO表做对比。



PRODUCTID	PRODUCTNAME	PRODUCTPRICE	QUANTITY	CATEGORY	DESCRIPTION	ORIGIN
1	2040030001 海狗牌818A	1100.00	20	海狗牌		中国
2	2040030002 2040030002	3420.00	12	海狗牌		中国
3	2040030003 2040030003	2175.00	111	海狗牌		中国
4	2040030001 海狗牌818A	2040.00	22	海狗牌		中国

图8.3 PRODUCTINFO_TMP表数据列表

8.5 小结

游标是提取Oracle数据的重要手段，它可以把符合要求的数据检索出来并存储到缓冲区中。利用循环语句遍历整个结果集，从而获取每条记录，开发人员可以对每条记录进行操作。游标有4个属性，可以通过这4个属性得到游标的执行信息，也可以利用这些信息进行流程控制。游标中也允许异常处理，这和PL/SQL块一致。最后用一个示例演示了如何使用游标帮助解决业务问题。

8.6 习题

一、填空题

- 静态游标包含_____、_____两种类型。
- 通常使用_____命令遍历游标的数据集。
- 游标的使用步骤包括_____、_____、_____、_____。
- 游标4个属性是_____、_____、_____、_____。

二、选择题

- 如果游标没有被打开时就调用，会不会提示出错？（ ）
 A. 不会 B. 会
- 下面有关隐式游标中的%ISOPEN属性，说法正确的是（ ）。
 A. 隐式游标中的%ISOPEN属性永远返回FALSE
 B. 隐式游标中的%ISOPEN属性可以被人为地控制

第9章 视图——数据库中虚拟的表

视图在Oracle中应用相当普遍，所以也比较重要。视图在数据库中可以理解为一张虚拟的表，使用视图可以补充表结构在某些需求方面的不足，可以让开发人员更方便地查询复杂数据，还可以缩短开发周期，节省公司成本。本章包括如下知识点：

- ☐ 什么是视图
- ☐ 如何创建视图
- ☐ 如何查询视图
- ☐ 如何管理视图

9.1 什么是视图

初学者听到视图会比较陌生，实际上视图的创建和操作比较简单。在直观印象中它和表类似，但某些表的功能它不具备。本节就将讲述视图的概念和作用。

9.1.1 认识视图

根据官方的文档可以这样理解视图：它是一个基于一个表或多个表的逻辑表，视图本身不包含任何数据。通俗来说，可以把视图看成是虚拟的表，只是一个查询语句的结果，它的数据最终是从表中获取的，这些表通常称为源表或基表。当基表的数据发生变化时，视图里的数据同样发生变化。通常视图的数据源有下面三种情况：

- ☐ 单一表的子集。
- ☐ 多表操作结果集。
- ☐ 视图的子集。

9.1.2 视图的作用

有的读者会产生疑问，既然视图被称为虚拟的表，那还用它做什么？下面就简单介绍一下视图的作用。

- ☐ 使数据简化。在表中很多数据对业务来说是冗余的，这时开发者会使用比较复杂的SQL语句得到自己想要的。实际开发中不能要求每个人都能做到这一点，所以，通常情况下由一个人把该复杂语句做成视图，其他人员直接调用该视图即可。这样对视图使用人员就简化了数据，隐藏了数据的复杂性。
- ☐ 使数据更加独立。程序开发时，大多数是程序直接访问数据库的表，当这些表的结构随着业务的变化而不得不重新设计时会影响到程序（通常表一旦设计完成就很难再修改），所以可以使得程序直接访问视图。这样视图就可以把程序和数据库的表隔离开来，降低开发者的劳动成本。



- 增加安全性。视图可以查询表指定的列来展现给用户，而不必让使用者完全看见表的所有字段。这种情况很多是一个公司提供给其他合作伙伴查询数据的接口，而视图通常也会设成只读属性。

9.1.3 视图的语法

创建视图相对简单，学习视图时只要掌握下面的语法，就可以达到事半功倍的效果。

主要语法如下：

```
CREATE [ OR REPLACE ] [ [ NO ] FORCE ] VIEW
[ schema. ]view
[(alias,...) inline_constraint(s)]
[out_of_line_constraint(s)]
AS subquery
[
    WITH { READ ONLY | CHECK OPTION [ CONSTRAINT constraint ] }
];
```

【语法说明】

- OR REPLACE：表示新建视图可以覆盖同名视图。
- [NO] FORCE：即FORCE或NOFORCE，表示是否强制创建视图。例如，在基表不存在的情况下就创建视图是有错误的，这时可以用FORCE关键词强制创建视图，然后再创建基表。Oracle中NOFORCE是默认值。
- [schema.]view：这是视图的所属方案名称和视图本身的名称。
- [(alias,...) inline_constraint(s)]：视图字段的别名和内联约束。
- [out_of_line_constraint(s)]：也是约束，是与inline_constraint(s)相反的声明方式。
- WITH READ ONLY：设置视图只读，这样的视图具有更高的安全性。
- WITH CHECK OPTION [CONSTRAINT constraint]：一旦使用该限制，当对视图增加或修改数据时必须满足子查询的条件。也就是说，是把子查询的条件作为一个约束，而constraint是这个约束的名称。

注意 语法中的大括号表示必选语法项，这里只是说明WITH后面必须接大括号里面的内容。

9.2 视图的创建

可以直接使用SQL语句创建一个视图，也可以使用可视化工具来创建。例如，本书会用PL/SQL Developer这款软件介绍如何创建视图。

9.2.1 创建单表视图

操作数据库最基本的方式就是使用SQL语句，很多读者都不习惯直接使用SQL语句来操作数据库，但笔者建议初学数据库时最好多写SQL语句，这样可以有效避免做开发时出现眼高手低的情况，同时也提高了工作效率。下面直接使用SQL*Plus创建视图。

创建视图前先列出表PRODUCTINFO（产品信息）和CATEGROYINFO（产品类型）的记录，方便和视图做对比。

```
SELECT PRODUCTID, PRODUCTNAME, PRODUCTPRICE, CATEGORY, ORIGIN
FROM
PRODUCTINFO:
```

```
SQL> SELECT ORIGINID, ORIGINNAME FROM ORIGINID;

ORIGINID ORIGINNAME
-----
0000000000 北京
0000000001 上海
0000000002 天津
0000000003 广州
0000000004 杭州
0000000005 深圳
```

图9.2 表CATEGORYINFO数据

注意 根据官方提供的资料，在当前用户下创建视图需要有CREATE VIEW系统权限，这里直接给当前用户赋予了DBA权限。

这个单表视图的作用是展示5行产地是“中国”的数据。编写如下脚本在SQL*Plus中执行:

```
01 CREATE OR REPLACE VIEW SIMPLE_PRODUCTINFO_VIEW
02 AS
03 SELECT PRODUCTID, PRODUCTNAME, PRODUCTPRICE, CATEGORY, ORIGIN
04 FROM PRODUCTINFO
05 WHERE ORIGIN = '中国'
06 AND ROWNUM < 6;
```

视图已创建

□ 第1行表示创建或覆盖名称为SIMPLE_PRODUCTINFO_VIEW的视图。如果没有OR REPLACE关键字,表示只创建视图而不能覆盖同名视图。



- 第5行表示只列出产地是“中国”的数据。
- 第6行中的ROWNUM < 6表示列出5条数据。ROWNUM是一个伪列，也就是说表中没有该字段，利用它可以限制返回的行数。它总是从1开始增加，这个特点决定了我们使用它通常都是用ROWNUM < n (n>1) 或ROWNUM = 1的格式作为条件。

【执行效果】

使用SELECT查询语句可以查看视图的效果，语法同查询表数据一样，只需要把FROM后而换成要查询的视图名称即可。如下面语句，执行后将会得到图9.3所示结果。

```
SELECT PRODUCTID 产品ID, PRODUCTNAME 产品名称, PRODUCTPRICE 产品价格,
       CATEGORY 产品类型编码, ORIGIN 产地
FROM SIMPLE_PRODUCTINFO_VIEW;
```

产品ID	产品名称	产品价格	产品类型编码	产地
01-0001-00001	手机	2500	01-0001-00001	中国
02-0001-00001	笔记本电脑	7800	01-0001-00001	中国
03-0001-00001	数码相机	1100	01-0001-00001	中国
04-0001-00002	MP3播放器	3100	01-0001-00002	美国
05-0001-00003	数码相机	2100	01-0001-00003	美国

图9.3 视图SIMPLE_PRODUCTINFO_VIEW查询结果

经过视图里的查询条件过滤后得到的数据和表PRODUCTINFO的数据是不一样的，这样就达到了创建视图的目的。

可以利用下面的语句查看当前用户下的所有视图：

技巧

```
SELECT VIEW_NAME FROM USER_VIEWS
```

当查询出来的某字段列宽不够时可以用下面的语句改变某字段宽度：

```
COL 产品类型编码 FORMAT A15
```

9.2.2 创建多表视图

在实际应用中，更多的视图是基于多个基表的视图，这样的视图能充分展示它的优点。下面就介绍基于两个表的视图。

笔者把表PRODUCTINFO当中的CATEGORY（商品类型）字段设计成存放编码的字段（参见图9.1所示的CATEGORY字段），而商品类型则单独设计成了表CATEGORYINFO（商品类型信息表），详细的表结构可以参考第4章。这样做符合数据设计的原则，也方便开发者的使用。

【示例2】创建多表视图

下面创建的视图是把PRODUCTINFO和CATEGORY这两个表关联起来查询的，目的是把商品类型编码替换成商品类型。编写如下SQL语句并在SQL*Plus中执行：

```

01 CREATE OR REPLACE VIEW MULTI_PRODUCTINFO_VIEW
02 AS
03 SELECT PT.PRODUCTID, PT.PRODUCTNAME, PT.PRODUCTPRICE, PT.CATEGORY,
04        CG.CATEGORYNAME, PT.ORIGIN
05 FROM PRODUCTINFO PT, CATEGORYINFO CG
06 WHERE PT.CATEGORY = CG.CATEGORYID
07 AND PT.ORIGIN = '中国'
08 AND ROWNUM < 10;

```

如果执行成功将会出现如下字样：

视图已创建

【代码解析】

- 第1行表示创建或覆盖名称为MULTI_PRODUCTINFO_VIEW的视图。
- 第3行和第4行中字段前面多了“PT”或“CG”字样，这是表的别名，能直观了解到该字段属于哪个表。
- 第4行的CG.CATEGORYNAME字段表示产品类型名称。
- 第6行表示查询数据的条件之一是把PRODUCTINFO中CATEGORY字段和CATEGORYINFO中CATEGORYID字段值相等的关联起来，以达到把商品类型编码替换成商品类型的目的。
- 第7行和第8行详细解释参考9.2.1小节。

【执行效果】

查询视图MULTI_PRODUCTINFO_VIEW，执行如下语句，会得到图9.4所示结果。

```

SELECT PRODUCTID 产品ID, PRODUCTNAME 产品名称, PRODUCTPRICE 产品价格,
       CATEGORY 产品类型编码, CATEGORYNAME 产品类型, ORIGIN 产地
FROM MULTI_PRODUCTINFO_VIEW;

```

PRODUCTID	PRODUCTNAME	PRODUCTPRICE	CATEGORY	CATEGORYNAME	ORIGIN
1	产品1	1000	1	电子产品	中国
2	产品2	2000	2	服装	中国
3	产品3	3000	3	食品	中国
4	产品4	4000	4	饮料	中国
5	产品5	5000	5	玩具	中国
6	产品6	6000	6	图书	中国
7	产品7	7000	7	体育用品	中国
8	产品8	8000	8	家居用品	中国
9	产品9	9000	9	办公用品	中国
10	产品10	10000	10	其他	中国

图9.4 视图MULTI_PRODUCTINFO_VIEW查询结果

参照查询结果可以发现，在视图中“产品类型编码”和“产品类型”是一一对应的，这样就可以直接使用视图而没有必要写烦琐的关联语句。

在SQL*Plus中如果展示行宽不足,可以使用如下语句增加行宽。这样可以避免出现查询结果换行的情况。

```
SET LINESIZE 200
```

技巧

如果想了解一下SQL语句的执行效率如何,可以使用下面的语句,这样每次执行完后都会给出执行耗时间。对于SQL*Plus中格式的设置,还可以参考第14章。

```
SET TIMING ON
```



9.2.3 创建视图的视图

前面说过视图的数据来源可以是视图的子集。其实这种方式和前两种方式区别不大,不过笔者在工作过程中很少遇到这样的情况。下面就以MULTI_PRODUCTINFO_VIEW这个视图为基础视图,简单展示一下在视图基础上创建新的视图的过程。

【示例3】在视图的基础上创建视图

创建MULTI_PRODUCTINFO_VIEW的视图。执行如下创建视图语句:

```
CREATE OR REPLACE VIEW VI_PRODUCTINFO_VIEW
```

```
AS
```

```
SELECT PRODUCTID, PRODUCTNAME, PRODUCTPRICE, CATEGORYNAME, ORIGIN
FROM MULTI_PRODUCTINFO_VIEW;
```

【代码解析】

从创建语句上可以看到,视图VI_PRODUCTINFO_VIEW就是把MULTI_PRODUCTINFO_VIEW中的CATEGORY字段去除了。

【执行效果】

执行下面的查询语句,得到图9.5所示结果。

```
SELECT PRODUCTID 产品ID, PRODUCTNAME 产品名称, PRODUCTPRICE 产品价格,
       CATEGORYNAME 产品类型, ORIGIN 产地
FROM VI_PRODUCTINFO_VIEW;
```

产品ID	产品名称	产品价格	产地
00000000000000000000000000000000	00000000000000000000000000000000	00000000000000000000000000000000	00000000000000000000000000000000
00000000000000000000000000000000	00000000000000000000000000000000	00000000000000000000000000000000	00000000000000000000000000000000
00000000000000000000000000000000	00000000000000000000000000000000	00000000000000000000000000000000	00000000000000000000000000000000
00000000000000000000000000000000	00000000000000000000000000000000	00000000000000000000000000000000	00000000000000000000000000000000

图9.5 视图VI_PRODUCTINFO_VIEW查询结果



9.2.4 创建没有源表的视图

在数据库开发过程中，如果设计工作做得很好，就可能出现没有源表时先把视图创建出来的情况。因为根据设计，基表是肯定要建的。本小节介绍一下在没有源表的情况下创建一个新视图。

【示例4】创建没有源表的视图

没有源表时，使用正常方式创建视图会出现错误。演示脚本如下：

```
01 CREATE OR REPLACE VIEW NOTABLE_PRODUCTINFO_VIEW AS
02 SELECT PRODUCTID, PRODUCTNAME, PRODUCTPRICE, ORIGIN
03 FROM NOTABLE
```

【代码解析】

第3行里的表NOTABLE在Oracle中是不存在的。

【执行效果】

执行完成后会提示如下错误：

第 3 行出现错误：

ORA-00942: 表或视图不存在

这里说明该视图创建不成功。假如利用FORCE关键词就可以避免这种情况，该关键词表示创建视图时，无视源表是否存在。

【示例5】强制创建没有源表的视图

强制创建没有源表的视图，把示例4的脚本修改如下：

```
01 CREATE OR REPLACE FORCE VIEW NOTABLE_PRODUCTINFO_VIEW AS
02 SELECT PRODUCTID, PRODUCTNAME, PRODUCTPRICE, ORIGIN
03 FROM NOTABLE
```

【执行效果】

警告：创建的视图带有编译错误。

如果出现以上提示，表示该视图已经创建。

9.2.5 使用PL/SQL工具创建视图

使用PL/SQL工具可以比较简洁、方便地创建视图，尤其对忘记视图语法的使用者。（对于PL/SQL工具的安装和使用，可以在第14章中查看到）不过它和SQL Server管理工具不一样，下面就使用该工具展示如何创建视图。

1) 启动PL/SQL Developer工具，在整个布局的左上角单击【File】菜单，从弹出的下拉列表中选择【New】选项，在弹出的子选项中找到【View】项，如图9.6所示。

2) 单击该选项，出现图9.7所示窗口。该窗口就是编辑视图的模板窗口。

- ☐ Name: 视图的名称。
- ☐ Item list: 视图的字段名称。
- ☐ Table list: 视图的源表。
- ☐ Where clause: 创建视图的查询条件。

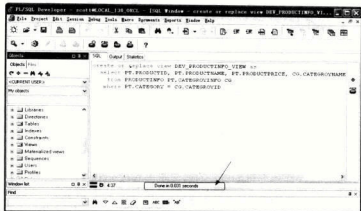


图9.9 视图创建成功

6) 查看创建的视图。在PL/SQL Developer工具中可视化查看视图，具体位置在左边【Objects】透视图的【Views】节点下。具体见图9.10。其中黑框部分就是用PL/SQL Developer创建的视图，其他3个是第9.2.1~9.2.3小节创建的视图。

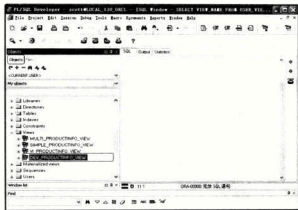


图9.10 当前用户视图列表

7) 查看视图数据。在该视图上右击，从弹出的快捷菜单中选择【Query data】选项，可以查询该视图的数据。具体操作见图9.11。

8) 视图数据列表。此时PL/SQL Developer会在右侧的透视图列出数据，并把查询语句自动打印在SQL窗口中，如图9.12所示。

以上是在PL/SQL Developer中创建视图的整个过程。相信读者已经能完整地创建视图了。

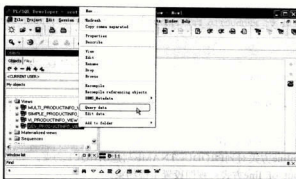


图9.11 查询视图数据操作

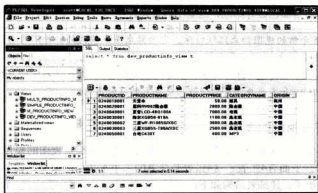


图9.12 视图数据列表

9.2.6 创建带约束的视图

前面介绍了在不同的环境下如何创建视图，相信读者已经掌握方法了。接下来介绍一下如何创建带约束的视图。

【示例6】创建带约束的视图

创建名为CONST_PRODUCTINFO_VIEW的视图，此视图包含唯一约束和一个主键约束。可以在PL/SQL Developer的SQL窗口或Command窗口（和SQL*Plus效果接近）下执行如下脚本：

```
01 CREATE OR REPLACE VIEW CONST_PRODUCTINFO_VIEW
02 (
03     PRODUCTID,
04     PRODUCTNAME CONSTRAINT PRODUCTNAME_UNQ UNIQUE RELY DISABLE NOVALIDATE,
05     PRODUCTPRICE,
06     QUANTITY,
07     CATEGORY,
```



```

08  ORIGIN,
09  CONSTRAINT VI_PRODUCTID_PK PRIMARY KEY (PRODUCTID) RELY DISABLE NOVALIDATE
10 )
11 AS
12 SELECT PRODUCTID, PRODUCTNAME, PRODUCTPRICE, QUANTITY, CATEGORY, ORIGIN
13 FROM PRODUCTINFO
14 WHERE ORIGIN = '中国'
15 WITH CHECK OPTION;

```

【代码解析】

- ☐ 第1~9行表示视图列的名称，共有6个字段。
- ☐ 第4行使用inline方式对列PRODUCTNAME创建UNIQUE约束。
- ☐ 第9行使用out_of_line方式对视图设置主键约束。
- ☐ 第4和第9行中的RELY DISABLE NOVALIDATE表示约束对此前和此后的数据都不进行检查，并告知Oracle此视图现在符合这两种约束条件。
- ☐ 第11~15行前面已经介绍过，这里不再赘述。

注意

本小节所介绍的视图约束比较特殊。它的约束是声明式的，并不能真正作用于视图本身。当创建约束后增加或更新违背约束的数据时，它会执行成功。从代码说明的第3行和第8行解释也能看出，这种声明实际上是告诉Oracle优化器它的数据是符合约束条件的。但Oracle本身并没有验证数据是否真的符合条件。

9.2.7 有关视图的案例

本小节将介绍一个比较真实的练习环境，使读者迅速巩固对视图的操作。这里将接触到使用函数的视图。下面有4张表，这4张表是简单的二手车买卖的原型，其中各表之间的关系约束这里就不给出了。详细表结构如下：

1. 二手车记录表

该表存放库存中二手车的简单信息。共有5个字段，分别是编号、车牌号、商标编码、成色编码、是否售出（1：售出，0：未售出）。数据库中这些字段的数据类型定义如表9.1所示。

表9.1 二手车记录表 (carinfo)

字段名	中文释义	数据类型
carid	编号	varchar2(10)
carrgnum	车牌号	varchar2(8)
carbrandid	商标编码	varchar2(8)
carfettleid	成色编码	varchar2(2)
flag	是否售出	varchar2(1)

2. 买卖记录表

该表存放二手车的交易记录。共有6个字段，分别是序号、车牌号、商标编码、交易日期、交易者、备注。数据库中这些字段的数据类型定义如表9.2所示。

表9.2 买卖记录表 (tradrec)

字段名	中文释义	数据类型
id	序号	varchar2(10)
carrgnum	车牌号	varchar2(8)
carbrandid	商标编码	varchar2(8)
bgdate	交易日期	date
bargainer	交易者	varchar2(10)
remark	备注	varchar2(200)



3. 车牌名称表

该表存放汽车商标（车牌名称）的种类，是字典表。共有两个字段，分别是车牌编码、车牌名称。数据库中这些字段的数据类型定义如表9.3所示。

表9.3 车牌名称表 (carbrandtab)

字段名	中文释义	数据类型
carbrandid	车牌编码	varchar2(8)
carbrand	车牌名称	varchar2(20)

4. 车辆成色表

该表存放待出售汽车的成色，是字典表。共两个字段，分别是成色编码、成色。数据库中这些字段的数据类型定义如表9.4所示。

表9.4 车辆成色表 (carfettleab)

字段名	中文释义	数据类型
carfettleid	成色编码	varchar2(2)
carfettle	成色	varchar2(20)

下面将利用“二手车记录表”、“买卖记录表”、“车牌名称表”以及“车辆成色表”这4张表创建示例7和示例8两个视图。

【示例7】要求列出出售车辆的明细列表，包括序号、车牌号、车牌名称（商标）、售出日期、售出人、成色

【示例分析】

此视图需要用到4张表，并需要利用连接查询的方式把“车牌名称（商标）”、“成色”字段的编码替换成真正的名称。

在PL/SQL Developer的SQL窗口中执行如下脚本，正常情况下会提示创建成功。这部分操作可参考前面所讲内容。

```

01 CREATE OR REPLACE VIEW TRADREC_DETAIL_VIEW
02 AS
03 SELECT TC.ID,TC.CARRGNUM,CB.CARBRAND,TC.BGDATE,TC.BARGAINER,CFB.CARFETTLE
04 FROM TRADREC TC,CARINFO CO,CARBRANDTAB CB,CARFETTLETAB CFB

```



```

05 WHERE TC.CARBRANDID = CB.CARBRANDID
06 AND CO.CARFETTLEID = CFB.CARFETTLEID
07 AND TC.CARRGNUM = CO.CARRGNUM
08 ORDER BY CB.CARBRAND

```

【代码解析】

□ 第3~5行是执行连接查询。这里需要注意的是，因为这4张表都有关系约束，所以这里列出匹配记录就能满足我们的要求，如果没有关系约束，读者可以根据实际情况使用外连接查询。

□ 第8行是根据车牌名称默认方式排序（升序）。

【示例8】要求统计某个日期之后不同品牌的车卖出的数量

【示例分析】

该视图只需用到两张表，分别是tradrec（买卖记录表）和carbrandtab（车牌名称表）。根据要求创建视图需要用到连接和分组，以达到统计的目的。

执行如下脚本：

```

01 CREATE OR REPLACE VIEW TRADREC_STAT_VIEW
02 AS
03 SELECT CB.CARBRAND ,COUNT(CARRGNUM) AS QUANTITY
04 FROM TRADREC TC,CARBRANDTAB CB
05 WHERE TC.BGDATE > TO_DATE('2007-5-10','YYYY-MM-DD')
06 AND TC.CARBRANDID = CB.CARBRANDID
07 GROUP BY CB.CARBRAND

```

【代码解析】

□ 第3行和第7行对应组合表示根据CB.CARBRAND（车牌名称）字段分组，把不同品牌的汽车数量利用函数COUNT()计算出来，并把计算出的数量取别名为QUANTITY。

□ 第5行是一个字符串转成日期的函数。该行效果是把日期大于“2007-5-10”的数据列出。

注意

在视图中函数的使用比较常见，如果这方面比较薄弱，最好多加练习。另外，在使用函数或运算表达式时必须指定列的别名。

9.3 操作视图数据的限制

视图允许做DML操作，但需要注意的地方比较多。因为视图增加或更新数据实际上是在操作视图的源表。除此之外，视图本身可以设置更新限制条件。本节主要介绍对视图数据做更新操作的相关注意事项。

9.3.1 视图READ ONLY设置

创建视图时为了避免用户修改数据，可以把视图设成只读属性，其操作比较简单。

【示例9】创建只读属性的视图

具体创建脚本如下：

```

01 CREATE OR REPLACE VIEW SIMPLE_PRODUCTINFO_VIEW AS

```

```

02 SELECT PRODUCTID, PRODUCTNAME, PRODUCTPRICE, CATEGORY, ORIGIN
03 FROM PRODUCTINFO
04 WHERE ORIGIN = '中国'
05 WITH READ ONLY

```

【代码解析】

□ 第5行是本小节重点，通过这个选项可以使该视图只读。当插入或修改视图数据时，会提示“无法对只读视图执行DML操作”。

9.3.2 视图CHECK OPTION设置

在某些情况下允许修改视图的数据，修改数据的本质是修改视图源表的数据。假如某个视图查询出来的是年龄大于20的所有数据，如果为该视图增加一条年龄为10的记录，那么该记录将不会出现在视图中。显然这是不符合逻辑的。为了避免这种情况的发生，可以利用CHECK OPTION选项来设置视图的检查约束。

CHECK OPTION选项表示视图启动了和子查询条件一样的约束。也就是说，如果对视图修改或插入的数据和查询条件不一致，那么该操作会被中止。

【示例10】创建带检查约束的视图

具体操作脚本如下：

```

01 CREATE OR REPLACE VIEW SIMPLE_PRODUCTINFO_VIEW AS
02 SELECT PRODUCTID, PRODUCTNAME, PRODUCTPRICE, CATEGORY, ORIGIN
03 FROM PRODUCTINFO
04 WHERE ORIGIN = '中国'
05 WITH CHECK OPTION

```

【代码解析】

□ 第5行的选项就开启了条件检查。这时如果要增加或修改数据，就要符合WHERE后面的条件，即ORIGIN字段的值是“中国”。

【执行效果】

以上脚本创建视图的数据要求只列出产地是中国的商品。查询后的列表如图9.13所示。

PRODUCTID	PRODUCTNAME	PRODUCTPRICE	CATEGORY	ORIGIN
0240020001	康师傅方便面	2800.00	0100020001	中国
0240040001	康师傅方便面	7000.00	0100030001	中国
0240030001	康师傅方便面	1100.00	0100030002	中国
0240030002	康师傅方便面	3400.00	0100030002	中国
0240030003	康师傅方便面	2500.00	0100030002	中国
0240030004	康师傅方便面	600.00	0100030001	中国

图9.13 数据列表



为了验证视图的CHECK OPTION设置是否生效，下面在PL/SQL Developer工具中对视图进行DML操作。

(1) 增加数据

执行如下增加语句：

```
INSERT INTO SIMPLE_PRODUCTINFO_VIEW
(PRODUCTID, PRODUCTNAME, PRODUCTPRICE, CATEGORY, ORIGIN)
VALUES('0240050005', '测试', 400.00, '0100040001', '美国')
```

这里增加的数据ORIGIN字段值是“美国”，它和视图的查询条件不一致，出现错误提示，如图9.14所示。



图9.14 错误提示

(2) 修改数据

对视图某条记录进行修改。执行如下语句：

```
UPDATE SIMPLE_PRODUCTINFO_VIEW
SET ORIGIN = '美国'
WHERE PRODUCTID = '0240020001'
```

以上脚本把视图里已有数据的ORIGIN字段值修改为“美国”，但依然提示图9.14所示错误。

(3) 删除数据

在表PRODUCTINFO中有一条数据的ORIGIN字段的值是“美国”，接下来对视图执行删除操作。脚本如下：

```
DELETE FROM SIMPLE_PRODUCTINFO_VIEW
WHERE ORIGIN = '美国'
```

脚本执行后并没有出现错误提示，但提示0行被操作。这不但说明了视图过滤出来的数据和源表的数据在逻辑上彻底分离，也说明了CHECK OPTION项对删除没有作用（如果有作用，删除语句将无法执行）。

9.3.3 视图创建语句对视图操作的影响

如果想要一个可以更新（这里的更新是指增加、删除、修改）的视图，源表应尽量是单表，否则限制比较多。下面的情况一旦出现在视图中，视图就不允许更新。

- ❑ DISTINCT关键字。
- ❑ 集合运算或分组函数，如INTERSECT、SUM、MAX、COUNT等函数。
- ❑ 出现GROUP BY、ORDER BY、MODEL、START WITH等语句。
- ❑ 出现伪列关键字，如ROWNUM。

除了以上情况外，还需要考虑基表的一些约束，这些约束对视图数据的更新都有一定影响。如果需要创建可以更新的视图，可以使用INSTEAD OF触发器。

9.4 视图的修改

视图的修改比较特殊，不能像表一样修改，更准确地说它没有修改选项，可以覆盖原有视

图,但这不会影响视图的使用。因为视图本身不包含数据,所以覆盖原有视图时就不存在数据丢失的问题。

9.4.1 视图结构的修改

对视图进行修改的SQL脚本前面已经接触过,可参见9.1.3小节。从中可以看到OR REPLACE 关键词,创建视图时带上它,如果存在同名视图,新视图就可以覆盖,也就等于完成修改。下面将讲述在PL/SQL Developer中如何修改视图。

1) 进入编辑页面。在PL/SQL Developer工具中左边【Objects】透视图的【Views】节点下找到想要编辑的视图。这里选择SIMPLE_PRODUCTINFO_VIEW视图。右击该视图,在弹出的快捷菜单中选择【Edit】项,进入编辑页面,如图9.15所示。

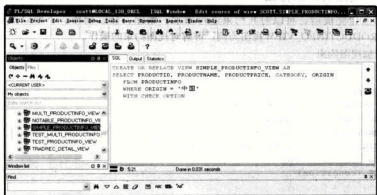


图9.15 视图编辑页面

2) 对视图编辑并执行。在此窗口中对视图进行修改,例如增加QUANTITY字段,并把WITH CHECK OPTION项换成WITH READ ONLY,然后按F8键执行。

3) 检验运行结果。我们改动了两处,首先检查第一处。右击该视图,在弹出的快捷菜单中选择【Describe】命令,出现图9.16所示的视图描述对话框。

图9.16所示对话框中的第4行是修改视图增加的字段,这说明视图第一处修改成功。假如想在SQL* Plus中查看视图列信息,可以执行如下语句:

```
DESC SIMPLE_PRODUCTINFO_VIEW;
```

接下来检查第二处修改。右击该视图,在弹出的快捷菜单中选择【Properties】命令,出现图9.17所示的视图属性对话框。

Name	Type	Nullable	Default	Comments
1. PRODUCTID	VARCHAR2(10)			
2. PRODUCTNAME	VARCHAR2(20)			
3. PRODUCTPRICE	NUMBER(8,2)			
4. QUANTITY	NUMBER(8,2)			
5. CATEGORY	VARCHAR2(10)			
6. ORIGIN	VARCHAR2(10)			

图9.16 视图描述对话框

Property	Value
1. OWNER	SCOTT
2. VIEW_NAME	SIMPLE_PRODUCTINFO_VIEW
3. TEXT_LENGTH	134
4. TEXT	SELECT PRODUCTID, PRODUCTNAME, PRODUCTPRICE, QUANTITY, CATEGORY, ORIGIN FROM PRODUCTINFO
5. WITHCHECKING_VIEW_N	WITH_READ_ONLY
6. WITH_READ_ONLY	Y

图9.17 视图属性对话框

从图9.17中第6行可以看到READ_ONLY的值是Y，即该视图只读，这说明更改视图生效。下面简单说明一下属性窗口中各选项代表的意义。

- ☐ OWNER：视图所有者，这里是SCOTT用户。
- ☐ VIEW_NAME：视图名称。
- ☐ TEXT_LENGTH：下面TEXT的长度。
- ☐ TEXT：视图的查询语句。
- ☐ EDITIONING_VIEW：是否版本视图，Oracle 11.2新特性，表结构经常变动，可利用版本视图消除其对使用者的影响。
- ☐ READ_ONLY：是否只读。

9.4.2 视图约束的修改

当视图创建完成后，可以对其约束进行添加、删除、修改操作。下面以示例1创建的SIMPLE_PRODUCTINFO_VIEW视图为例，讲解对视图约束的操作。

(1) 增加视图约束

主要语法如下：

```
ALTER VIEW [schema.]view
ADD [CONSTRAINT constraint_name]
[UNIQUE (column [, column ]...)]
| PRIMARY KEY (column [, column ]...)
| FOREIGN KEY (column [, column ]...)
    references_clause
| CHECK (condition)
}
[constraint_state]
```

【语法说明】

- ☐ ALTER VIEW：表示修改视图的关键词。
- ☐ ADD [CONSTRAINT constraint_name]：为视图增加一项约束，可以带约束名称。
- ☐ UNIQUE：唯一约束。
- ☐ PRIMARY KEY：主键约束。
- ☐ FOREIGN KEY：外键约束。
- ☐ CHECK：检查约束。
- ☐ constraint_state：约束声明。

【示例11】为视图增加唯一约束

在PL/SQL Developer中执行的具体脚本如下：

```
01 ALTER VIEW SIMPLE_PRODUCTINFO_VIEW
02 ADD CONSTRAINT PUTPRIC_UNQ UNIQUE (PRODUCTPRICE)
03 DISABLE NOVALIDATE;
```

【代码解析】

- ☐ 第1~2行表示视图SIMPLE_PRODUCTINFO_VIEW增加唯一约束。约束名称是PUTPRIC_UNQ。

□ 第3行表示此前数据和以后数据都不进行检查。

【执行效果】

为视图增加唯一约束完成后可以查看执行结果。执行下面的脚本，从用户约束表里查看增加约束是否成功：

```
SELECT CONSTRAINT_NAME, TABLE_NAME, R_OWNER, R_CONSTRAINT_NAME
FROM USER_CONSTRAINTS
WHERE TABLE_NAME = 'SIMPLE_PRODUCTINFO_VIEW';
```

执行完成后发现视图增加约束成功，如图9.18所示。

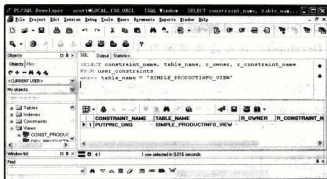


图9.18 增加的视图约束列表

(2) 删除视图约束

主要语法如下：

```
ALTER VIEW [schema.]view DROP CONSTRAINT constraint_name;
```

【示例12】删除视图中的约束

要求删除视图SIMPLE_PRODUCTINFO_VIEW的PUTPRIC_UNQ约束。具体脚本如下：

```
ALTER VIEW SIMPLE_PRODUCTINFO_VIEW
DROP CONSTRAINT PUTPRIC_UNQ;
```

【执行效果】

执行方式参考约束增加部分，提示成功后，再查询约束表数据，发现视图SIMPLE_PRODUCTINFO_VIEW名称为PUTPRIC_UNQ的唯一约束已经被删除。

9.5 视图的删除

视图删除和表删除操作方式一样，可以使用SQL语句删除，也可以使用PL/SQL Developer工具删除。在本节中就将分别讲述使用这两种方式删除视图的方法。

1. 使用SQL语句删除视图

主要语法如下：

```
DROP VIEW [schema.]view [CASCADE CONSTRAINTS]
```



【语法说明】

CASCADE CONSTRAINTS；删除视图时删除约束。

【示例13】删除视图SIMPLE_PRODUCTINFO_VIEW

具体脚本如下：

```
DROP VIEW SIMPLE_PRODUCTINFO_VIEW;
```

【执行效果】

当删除成功时会提示：

View dropped

2. 使用PL/SQL Developer工具删除视图

在PL/SQL Developer工具左边透视图的【Views】节点下选中视图SIMPLE_PRODUCTINFO_VIEW，右击该视图，从弹出的快捷菜单中选择【Drop】选项，如图9.19所示。

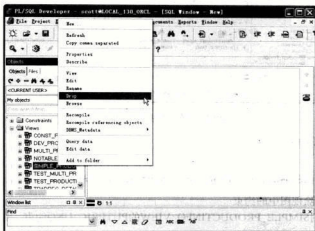


图9.19 删除视图

在弹出的确认窗口中单击【Yes】按钮，完成后会发现左边透视图【Views】节点下该视图已经被删除。

9.6 小结

在Oracle中经常使用视图，视图可以根据业务的需要从不同的角度展示数据，它给数据库管理者和用户都提供了一定的便捷性，也提高了数据的安全性。视图本身不包含任何数据，它只是一个查询，所有的数据都是从其他表或视图中获取的，但数据有着逻辑独立性。通过本章的学习，读者可以掌握什么是视图，如何创建视图，如何修改、删除视图，以及视图的数据更新有限制时对视图的操作。相信以上内容的讲解对刚接触视图的读者有很大帮助。



9.7 习题

一、选择题

1. 下面有关视图的数据来源叙述不正确的是()。
 - A. 视图数据是用户直接添加到视图中的
 - B. 视图数据来源于单表
 - C. 视图数据来源于多表
 - D. 视图数据来源于其他视图
2. 创建视图带SUM函数, 视图数据是否可以更新?()
 - A. 不可以
 - B. 可以
3. 视图中CHECK OPTION设置有什么作用?()
 - A. 没有实际作用
 - B. 检查视图更新数据是否符合视图创建时的查询条件
 - C. 检查数据是否有更新
 - D. 不允许向基表中更新数据

二、简答题

1. 视图的约束是否和表的约束一样?
2. 当对视图做删除数据操作时, 如果视图里没有符合条件的数据, 但基表存在符合条件的数据, 实际会出现什么情况?

第10章 存储过程——提高程序执行的效率

存储过程是Oracle开发者在数据转换或查询报表时最经常使用的方式之一。它就像编程语言一样一旦运行成功,就可以被用户随时调用,这种方式极大地节省了用户的时间,也提高了程序的执行效率。本章包含如下知识点:

- ☐ 什么是存储过程
- ☐ 创建存储过程
- ☐ 使用存储过程
- ☐ 管理存储过程

10.1 什么是存储过程

存储过程在数据库开发中使用比较频繁,它有着普通SQL语句不可替代的作用。如果读者以前学习过编程,那么相信学习存储过程对你来说不是大问题;而没有编程经验的读者通过本节的学习也将会掌握存储过程的概念和存储过程在数据库中的重要性,以及如何定义一个存储过程。

10.1.1 认识存储过程

所谓存储过程,就是一段存储在数据库中执行某种功能的程序,其中包含一条或多条SQL语句,但是它的定义方式和PL/SQL中的块、包等有所区别。存储过程可以通俗地理解为是存储在数据库服务器中的封装了一段或多段SQL语句的PL/SQL代码块。在数据库中有一些是系统默认的存储过程,那么可以直接通过存储过程的名称进行调用。另外,存储过程还可以在编程语言中调用,如Java、C#、VB等编程语言。

10.1.2 存储过程的作用

存储过程编写相对比较复杂,但很多单位或个人都在使用它。显然这不是因为存储过程编写简单,而是因为它有着一系列的优点:

- ☐ 简化复杂的操作。存储过程可以把需要执行的多条SQL语句封装到一个独立单元中,用户只需调用这个单元就能达到目的。这样就实现了一人编写多人调用,同时缩短了平均开发周期,为公司节省了成本。
- ☐ 增加数据独立性。与视图的效果类似,利用存储过程可以把数据库基础数据和程序(或用户)隔离开来,当基础数据的结构发生变化时,可以修改存储过程,这样对程序来说基础数据的变化是不可见的,也就不需要修改程序代码了。
- ☐ 提高安全性。使用存储过程有效地降低了错误出现的几率。如果不使用存储过程要想实现某项操作可能需要执行多条单独的SQL语句,而过多的执行步骤很可能造成更高的出错几率。不仅如此,实际工作中开发人员的水平参差不齐,由高水平的人编写存储过程,水平较低的人员直接调用,这样就能避免很多不必要的错误发生。此外,存储

过程也可以进行权限设置。

- 提高性能。完成一项复杂的功能可能需要多条SQL语句，同时SQL每次执行都需要编译，而存储过程可以包含多条SQL语句，而且创建完成后只需要编译一次，以后就可以直接调用，从这方面来看存储过程可以提高性能。如果程序语言要实现某项比较复杂的功能，它会多次连接数据库，在使用存储过程的情况下，程序只需连接一次就能达到目的。

10.1.3 存储过程的语法

存储过程的创建和视图相比稍微复杂，但也有它固有的模式，所以读者很快能够上手。但想运用自如平时还需要多加练习。下面介绍一下存储过程的主要语法：

```
CREATE [ OR REPLACE ] PROCEDURE [ schema. ] procedure_name
[ parameter_name [ [ IN ] datatype [ { := | DEFAULT } expression ]
| { OUT | IN OUT } [ NOCOPY ] datatype
] [ ... ]
{ IS | AS }
BODY ;
```

其中各项参数介绍如下：

- OR REPLACE：表示如果指定的过程已经存在，则覆盖同名的存储过程。
- schema：表示该存储过程的所属机构。
- procedure_name：创建存储过程的名称。
- parameter_name：表示存储过程中的参数名称。
- [IN] datatype [{ := | DEFAULT } expression]：整个这段语法表示传入参数的数据类型以及默认值。其中，datatype项表示参数的数据类型，[{ := | DEFAULT } expression]项表示参数的默认值的写法。
- { OUT | IN OUT } [NOCOPY] datatype：表示存储过程的参数类型，不过和上面介绍的IN有所区别。其中，OUT表示输出参数，IN OUT表示既可输入也可输出的参数，datatype依旧表示参数类型。
- { IS | AS }：连接词。
- BODY：表示函数体，是存储过程的具体操作部分，通常在begin...end中。

注意

存储过程的参数默认类型是IN型的，也就是说传入型的。当前模式下创建存储过程需要有CREATE PROCEDURE权限。

10.2 在SQL*Plus中创建存储过程

本节将介绍如何在SQL*Plus中创建存储过程，其中包括无参数存储过程的创建以及各类有参数存储过程的创建，同时讲述如何查看、执行存储过程等操作。

10.2.1 创建第一个存储过程

前面已经学习过创建存储过程的语法，下面就利用前面学习过的语法在SQL*Plus中创建第一个存储过程，如示例1所示。



【示例1】创建第一个存储过程
在SQL*Plus中执行如下脚本：

```
01 CREATE PROCEDURE TEST
02 AS
03 BEGIN
04     DBMS_OUTPUT.PUT_LINE('我的第一个过程！');
05 END;
06 /
```

【代码解析】

- 第1行表示创建过程，名为TEST。
- 第4行表示输出一行字符串“我的第一个过程！”。

示例1是最简单的一个示例，通过它可以直观地认识存储过程的创建过程。如果创建成功，会出现如下提示：

过程已创建

到目前为止，只是成功创建了过程，接下来执行该存储过程，查看输出结果。

【执行效果】

(1) SERVEROUTPUT设置

在SQL*Plus中如果想让DBMS_OUTPUT.PUT_LINE成功输出，需要把SERVEROUTPUT选项设置成ON状态。默认情况下，它是OFF状态的。如果不了解该属性的当前设置状态，可以执行如下语句进行查看：

```
SQL> SHOW SERVEROUTPUT
```

如果提示如下字样则表示该属性没有打开：

```
serveroutput OFF
```

接下来打开输出设置，运行下面的脚本：

```
SQL> SET SERVEROUTPUT ON
```

设置成功会出现下面的提示：

```
serveroutput ON SIZE UNLIMITED FORMAT WORD_WRAPPED
```

(2) 执行存储过程

在SQL*Plus中输入如下语句，用于执行已经创建的存储过程：

```
SQL> BEGIN
      TEST;
    END;
```

执行成功后会输出该过程中的输出语句。整个流程如图10.1所示。

技巧 还可以通过在关键词EXEC后面加上存储过程名来执行已经存在的存储过程。

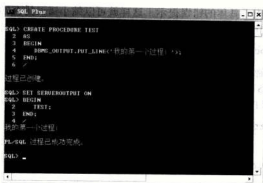


图10.1 存储过程的创建及执行

10.2.2 查看存储过程

存储过程一旦被创建就会存储到数据库服务器上；Oracle允许开发人员查看已经存在的存储过程脚本，这可以到视图USER_SOURCE里查看。

【示例2】查看存储过程TEST的脚本

具体脚本如下：

```
SELECT * FROM USER_SOURCE WHERE NAME = 'TEST' ORDER BY LINE ;
```

【执行效果】

执行查看后效果如图10.2所示。

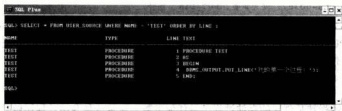


图10.2 查看TEST脚本

通过图10.2可以发现，每条记录中的TEXT字段都存储了一句脚本，把这些脚本综合起来就是创建TEST存储的过程。

当从视图USER_SOURCE中查询过程或函数时需要把名称大写，小写会得不到想要的记录。至于USER_SOURCE是当前用户的视图，如果想查看Oracle所有的存储过程，则需要到ALL_SOURCE视图中查询。

10.2.3 显示存储过程的错误

编写存储过程时由于各种原因都有可能出现问题而导致过程编译失败，在SQL*Plus中可



以利用错误显示语句查看具体的错误提示，这样就可以缩小错误排查的范围，从而提高开发效率。其在存储过程中的使用语法如下：

```
SHOW ERRORS PROCEDURE procedure_name;
```

【示例3】查看存储过程的错误

查看存储过程TEST_ERR中的错误。该示例分如下两步操作。

1) 创建存储过程TEST_ERR。

```
01 CREATE PROCEDURE TEST_ERR
02 AS
03 BEGIN
04     DBMS_OUTPUT.PUT_LINE('我的第一个过程!');
05 END;
06 /
```

【执行效果】

警告：创建的过程带有编译错误。

2) 查看存储过程TEST_ERR错误细节。具体执行的脚本如下：

```
SHOW ERRORS PROCEDURE TEST_ERR;
```

【执行效果】

PROCEDURE TEST_ERR 出现错误：

```
LINE/COL ERROR
```

```
4/5      PL/SQL: Statement ignored
4/17     PLS-00302: 必须声明 'PUT_LINE' 组件
```

从提示中可以看出，该错误是由存储过程TEST_ERR的第4行引起的，正确的写法是DBMS_OUTPUT.PUT_LINE('我的第一个过程!')。相信这种方式能给读者排查存储过程的错误提供一定的帮助。

10.2.4 无参存储过程

无参存储过程就是创建的存储过程不带任何参数，通常这种存储过程用做数据转换的几率比较大。如果读者认真地学习了前面的章节，相信很容易理解这部分内容。

下面将用一个示例来介绍如何编写无参存储过程。

【示例4】无参存储过程

把表PRODUCTINFO中价格最低的3件产品的DESPERATION字段设置成“促销商品”。实现步骤如下：

1) 将PRODUCTINFO中产品价格最低的3件产品查询出来。

2) 把价格最低的3件产品DESPERATION字段加上“促销商品”字样。

创建存储过程的脚本如下：

```
01 CREATE PROCEDURE PRODUCT_UPDATE_PRC
02 AS
```




```

03 BEGIN
04 UPDATE PRODUCTINFO SET DESPERATION = '促销产品'
05 WHERE PRODUCTID IN
06 (
07   SELECT PRODUCTID FROM
08     ( SELECT * FROM PRODUCTINFO ORDER BY PRODUCTPRICE ASC )
09   WHERE ROWNUM < 4
10 );
11 COMMIT;
12 END;
13 /

```

【代码解析】

这是一个结构相对简单的存储过程，它的函数体中只有一个稍微复杂点的SQL语句。下面简单说明一下该脚本。

- 第1行表示创建存储过程名为PRODUCT_UPDATE_PRC。
- 第4~10行实现了示例的要求。
- 第8行表示把表中所有记录根据产品价格进行升序排列。
- 第7~9行表示查询出价格最低的三条记录，这也是实现步骤中的第一点。
- 第11行表示提交更改。

【执行效果】

以上脚本在SQL*Plus中执行后将有如下提示：

过程已创建

此时的存储过程还没有得到执行，只是编译通过。下面执行PRODUCT_UPDATE_PRC，输入执行语句如下：

```
EXEC PRODUCT_UPDATE_PRC
```

执行完成后验证存储过程是否生效，查询表PRODUCTINFO的数据，整个流程如图10.3所示。

```

SQL> EXECUTE PRODUCT_UPDATE_PRC;
PL/SQL procedure successfully completed.

SQL> SELECT * FROM PRODUCTINFO ORDER BY PRODUCTPRICE ASC;

```

PRODUCTID	PRODUCTNAME	PRODUCTPRICE	QUANTITY	DESPERATION	ROWID
1	产品A	10	100	促销产品	AAACAAA...
2	产品B	20	200	促销产品	AAACBAA...
3	产品C	30	300	促销产品	AAACCAA...
4	产品D	40	400	促销产品	AAACDAA...
5	产品E	50	500	促销产品	AAACEAA...
6	产品F	60	600	促销产品	AAACFAA...
7	产品G	70	700	促销产品	AAACGAA...
8	产品H	80	800	促销产品	AAACHAA...
9	产品I	90	900	促销产品	AAACIAA...
10	产品J	100	1000	促销产品	AAACJAA...

图10.3 示例4的流程及结果

10.2.5 存储过程中使用游标

存储过程可以灵活地处理各种情况下的数据，它不仅可以用普通的方式处理数据（例如示例4），也可以使用游标来辅助处理更复杂的业务逻辑。下面就创建一个稍微复杂的存储过程，该过程处理数据需要使用游标，具体的业务要求见示例5。

【示例5】使用游标处理数据

要求把PRODUCTINFO表中数据根据不同的产品类型分类把数据输出到屏幕。具体脚本如下：

```

01 CREATE PROCEDURE PRODUCT_CUR_PRC
02 AS
03   cur_ctgy productinfo.category%TYPE;           --存放产品类型编码
04   cur_ctgname categoryinfo.categoryname%TYPE;   --存放产品类型名称
05   cur_ptrinfo productinfo%ROWTYPE;              --存放表productinfo的行记录
06
07   CURSOR cur_category
08   IS
09   SELECT CATEGORY FROM PRODUCTINFO GROUP BY CATEGORY;
10
11 BEGIN
12   OPEN cur_category;
13   LOOP
14     FETCH cur_category INTO cur_ctgy;
15     EXIT WHEN cur_category%NOTFOUND;
16     SELECT CATEGORYINFO.CATEGORYNAME INTO cur_ctgname
17     FROM CATEGORYINFO
18     WHERE CATEGORYID = cur_ctgy;           --根据类型编码得到产品类型名称
19
20     IF SQL%FOUND THEN
21       DBMS_OUTPUT.PUT_LINE('-----');
22       DBMS_OUTPUT.PUT_LINE(cur_ctgname || ':' );
23     END IF;
24
25     FOR my_prdinfo_rec IN
26     (
27       SELECT * FROM PRODUCTINFO WHERE CATEGORY = cur_ctgy
28     )
29     LOOP
30       DBMS_OUTPUT.PUT_LINE(
31         '产品名称: ' || my_prdinfo_rec.PRODUCTNAME
32         || '产品价格: ' || my_prdinfo_rec.PRODUCTPRICE
33         || '产品数量: ' || my_prdinfo_rec.QUANTITY
34       );
35     END LOOP;
36   END LOOP;
37   CLOSE cur_category;
38 END ;
39 /

```



【代码解析】

- 第1行表示创建存储过程，名称是PRODUCT_CUR_PRC。
- 第3~5行表示变量名称以及变量类型，脚本具体含义可参考游标部分的讲解。其中 cur_ctgy 存放产品类型编码，cur_ctgyname 存放产品类型名称，cur_prtifo 存放表 productinfo 的行记录。
- 第7~9行表示创建一个游标，表示从PRODUCTINFO表中查询已有的产品类型。
- 第12~15行表示打开游标cur_category并进入循环流提取数据，当数据提取完毕后退出。
- 第16~18行表示在CATEGROYINFO表中，根据产品类型编码得到产品类型名称，并把得到的结果赋值给变量cur_ctgyname。
- 第20~23行利用隐式游标%FOUND属性判断第16~18行的查询是否有结果，如果有则把产品类型名称输出到屏幕。
- 第25~35行表示利用隐式游标获取某类型的所有产品，并把产品信息输出到屏幕。

【执行效果】

以上脚本编译成功后利用如下脚本执行：

```
EXEC PRODUCT_CUR_PRC;
```

执行效果如图10.4所示。

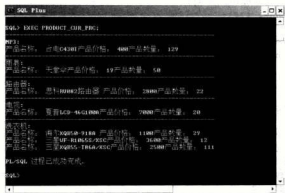


图10.4 示例5执行效果

从执行效果可以看出，此存储过程已经满足了示例的要求，把商品信息按商品类型分类输出到屏幕。事实上，在过程当中使用游标的情况很普遍，该示例中有三处用到了游标的知识（两处利用隐式游标），方便地解决了相关问题。读者应当熟练地掌握这种方式。

10.2.6 存储过程中的DDL语句

开发过程中为了让数据操作起来更方便会用到临时表，而为了让存储过程更具有通用性，可以选择把创建临时表的步骤一并放到过程里。这样的操作会和前面介绍的两种示例写法有所不同，它会用到EXECUTE IMMEDIATE语句，存储过程中会使用它来执行DDL语句和动态SQL语句。具体使用方法可以参考如下的示例。

【示例6】在存储过程中使用临时表

要求把各种不同类型的产品中价格最低的输入到临时表PRODUCTINFO_TMP（此表需要创建，结构参考PRODUCTINFO），并在其DESPERATION字段注明“热销商品”，如果记录中价格低于20则表示数据有问题，需要输出到屏幕。该存储过程比较烦琐，需使用游标以及EXECUTE IMMEDIATE语句处理相关问题。

【示例分析】本示例经过分析后可拆成如下步骤：

- 1) 创建临时表PRODUCTINFO_TMP，该表结构参考PRODUCTINFO。
- 2) 将PRODUCTINFO中各不同类型的产品中价格最低的产品列出来，并放到临时表PRODUCTINFO_TMP中。
- 3) 为PRODUCTINFO_TMP中的DESPERATION字段加上“热销商品”字样。
- 4) 如果记录中的价格低于20则表示数据有问题，该数据需要打印输出到屏幕。

【示例脚本】根据要求创建存储过程脚本如下：

```

01 CREATE PROCEDURE PRODUCT_TMP_UPDATE_PRC
02 AS
03
04   pc_delestr VARCHAR2(50);           --删除临时表记录语句
05   pc_createstr VARCHAR2(500);       --创建临时表
06   pc_insrtr VARCHAR2(500);         --向临时表中插入数据
07   tabext VARCHAR2(10);             --用于判断临时表是否存在中间变量
08
09   cur_ctgy productinfo.Category%TYPE;
10   cur_prtifo productinfo%ROWTYPE;
11
12   CURSOR cur_category              --产品表中的产品类型游标
13   IS
14   SELECT CATEGORY FROM PRODUCTINFO GROUP BY CATEGORY;
15
16   CURSOR cur_proinfo(ctgy varchar) --获取已有的产品类型中价格最低的数据
17   IS
18   SELECT * FROM
19   (SELECT * FROM PRODUCTINFO WHERE CATEGORY = ctgy ORDER BY PRODUCTPRICE ASC)
20   WHERE ROWNUM < 2 ;
21
22 BEGIN
23
24   SELECT COUNT(1) INTO tabext
25   FROM ALL_TABLES
26   WHERE TABLE_NAME = 'PRODUCTINFO_TMP';
27
28   pc_delestr:= 'DELETE FROM PRODUCTINFO_TMP';
29
30   pc_createstr := 'CREATE GLOBAL TEMPORARY TABLE PRODUCTINFO_TMP
31   (
32     PRODUCTID VARCHAR2(10) not null,
33     PRODUCTNAME VARCHAR2(20),
34     PRODUCTPRICE NUMBER(8,2),

```



```

35     QUANTITY      NUMBER(10),
36     CATEGORY      VARCHAR2(10),
37     DESPERATION    VARCHAR2(1000),
38     ORIGIN         VARCHAR2(10)
39 ) On Commit Preserve Rows';
40
41 if tabext=0 then
42     ----不存在临时表就创建一个
43     EXECUTE IMMEDIATE pc_createstr;
44     DBMS_OUTPUT.PUT_LINE('创建临时表成功');
45 else
46     EXECUTE IMMEDIATE pc_delestr;
47     DBMS_OUTPUT.PUT_LINE('删除记录完成!');
48 end if;
49
50 OPEN cur_category;
51 LOOP
52     FETCH cur_category INTO cur_ctgy;
53     EXIT WHEN cur_category%NOTFOUND;
54     OPEN cur_proinfo(cur_ctgy);
55     FETCH cur_proinfo INTO cur_prtifo;
56     IF cur_proinfo%FOUND THEN
57         IF cur_prtifo.PRODUCTPRICE < 20 THEN ----产品价格低于20打印出来
58             DBMS_OUTPUT.PUT_LINE('产品ID: '
59                                     ||cur_prtifo.PRODUCTID
60                                     ||'产品名称: '
61                                     ||cur_prtifo.PRODUCTNAME
62                                     ||'产品价格: '
63                                     ||cur_prtifo.PRODUCTPRICE );
64         ELSE ----非低于20价格的产品输入到临时表 PRODUCTINFO_TMP
65             EXECUTE IMMEDIATE 'INSERT INTO PRODUCTINFO_TMP(
66                                     PRODUCTID , PRODUCTNAME , PRODUCTPRICE,
67                                     QUANTITY, CATEGORY, DESPERATION,ORIGIN ) VALUES
68                                     (
69                                     ||cur_prtifo.PRODUCTID
70                                     ||','',''
71                                     || cur_prtifo.PRODUCTNAME
72                                     ||','',''
73                                     || cur_prtifo.PRODUCTPRICE
74                                     ||','',''
75                                     || cur_prtifo.QUANTITY
76                                     ||','',''
77                                     || cur_prtifo.CATEGORY
78                                     ||','',''
79                                     || cur_prtifo.DESPERATION
80                                     ||','',''
81                                     || cur_prtifo.ORIGIN
82                                     ||','',';'

```



```

83
84             END IF;
85         END IF;
86         CLOSE cur_proinfo;
87     END LOOP;
88     COMMIT;
89     CLOSE cur_category;
90     EXECUTE IMMEDIATE 'UPDATE PRODUCTINFO_TMP SET DESPERATION = '' 热销商品 ''';
91
92 END ;
93 /

```

【代码解析】

该示例脚本看上去很长，有些读者可能担心看不明白。其实大家可以放心，为了增加该示例的可读性，脚本有些地方被人为地断开了，笔者也希望读者自己编写脚本时可以考虑代码的格式，方便其他人阅读。下面对该脚本做具体的分析讲解。

- 第1~2行表示创建存储过程，名为PRODUCT_TMEP_UPDATE_PRC。
- 第4~10行表示声明变量。变量具体的含义可以参考脚本后面的注释。脚本中用“--”来注释单行语句，用“/* */”注释语句块。读者可以灵活运用。
- 第12~14行表示创建游标cur_category。利用GROUP BY语句把PRODUCTINFO表中不同的产品类型编码列出来。
- 第16~20行表示创建游标cur_proinfo，该游标带有参数，其参数表示产品类型编码。游标将根据产品类型不同，获取该产品类型中价格最低的数据。
- 第24~26行表示判断临时表PRODUCTINFO_TMP是否存在。此处利用SELECT INTO语句把结果放到变量tabext中，如果该表存在则结果为1，否则为0。tabext变量将在第41行处使用。
- 第28行表示为变量pc_delestr赋值，它的值是一条SQL语句。该SQL语句表示删除表PRODUCTINFO_TMP中的数据。这种写法常常用在动态SQL语句上。
- 第30行表示为pc_createstr变量赋值。它的值是一条DDL语句。从代码中可以看出此处的脚本用来创建临时表PRODUCTINFO_TMP。
- 第41~48行完成分析步骤中的第1步：创建PRODUCTINFO_TMP。首先判断临时表是否存在，如果不存在则创建临时表，如果存在则删除临时表中的数据，为下面的业务实现做准备。这里使用了EXECUTE IMMEDIATE语句，利用它执行了DDL语句以及动态语句。
- 第50~53行表示打开cur_category游标，并进入循环流取数据。当游标的%NOTFOUND属性为TRUE时退出。
- 第54~56行表示打开游标cur_proinfo，它的参数是cur_category中的结果。
- 第57~63行判断价格是否低于20，如果低于20则输出到屏幕。实现了分析步骤中的第4步。
- 第65~82行表示将价格不低于20的插入到表PRODUCTINFO_TMP中。实现了分析步骤中的第2步。
- 第90行将PRODUCTINFO_TMP表中数据修改为“热销商品”。完成分析步骤中的第3步。

【执行效果】

利用EXEC命令执行该过程，并检查运行效果。效果如图10.5所示。

```
SQL> EXECUTE IMMEDIATE 'SQL> SELECT * FROM PRODUCTINFO_TPI;'
```

PRODUCTID	PRODUCTNAME	PRODUCTPRICE	QUANTITY	CATEGORY	DESCRIPTION	ORIGIN
0010000000	金桔柠檬	4000	100	00-水果蔬菜	什锦凉菜	中国
0020000000	四川麻辣鸡	2000	100	00-水果蔬菜	什锦凉菜	中国
0030000000	四川麻辣鸡	7000	100	00-水果蔬菜	什锦凉菜	中国
0040000000	四川麻辣鸡	11000	100	00-水果蔬菜	什锦凉菜	中国

图10.5 示例6执行效果

从图10.5中可以看到，第一次执行该存储过程会提示“创建临时表成功”，如果数据库中已经存在同名的临时表则会提示“删除记录完成！”，这一点读者可以自行实验。这里说明一点，EXECUTE IMMEDIATE语句尽量少用，它对执行效率会有一定的影响。

技巧

在编写过程脚本时，为了增加代码的可读性，通常会在脚本中加上脚本注释。这么做不仅可以方便其他开发人员阅读，也方便自己今后阅读、修改脚本。

10.2.7 有参存储过程

存储过程允许带有参数，参数的使用将增加存储过程的灵活性，给数据库编程带来极大的方便。本小节将介绍如何使用输入类型参数、输出类型参数和输入输出类型参数。

存储过程中如果使用了参数，在执行存储过程时必须为其指定参数。总的来说，参数可以是常量（也可以是经过初始化的变量）、变量、表达式等。而参数的赋值方式也比较灵活，下面的示例演示了在存储过程中如何传递参数。

过程有输入、输出、输入输出三种参数（这里不是指参数的数据类型）。其中，输入参数是默认的参数，也叫IN类型的参数。下面通过示例演示在存储过程中如何使用输入参数。

【示例7】使用输入参数的存储过程

该示例将根据输入的产品类型从表PRODUCTINFO中搜索符合要求的数据，并将其打印到屏幕。本示例中的参数为输入类型。具体脚本如下：

```
01 CREATE PROCEDURE PRODUCT_INTYPE_PRC (parm_ctgname IN VARCHAR2)
02 AS
03   cur_ctgyid categoryinfo.categoryid%TYPE;          --存放产品类型编码
04   cur_ptinfo productinfo%ROWTYPE;                  --存放表productinfo的行记录
05
06 BEGIN
07   SELECT CATEGORYINFO.CATEGORYID INTO cur_ctgyid
08   FROM CATEGORYINFO
09   WHERE CATEGORYINFO.CATEGORYNAME = parm_ctgname;
10   --根据类型编码得到产品类型名称
11
```



```

12      IF SQL%FOUND THEN
13          DBMS_OUTPUT.PUT_LINE('-----');
14          DBMS_OUTPUT.PUT_LINE(param_ctgname || ':' );
15      END IF;
16      FOR my_prdinfo_rec IN
17      (
18          SELECT * FROM PRODUCTINFO WHERE CATEGORY = cur_ctgyid
19      )
20      LOOP
21          DBMS_OUTPUT.PUT_LINE(
22              '产品名称: ' || my_prdinfo_rec.PRODUCTNAME
23              || '产品价格: ' || my_prdinfo_rec.PRODUCTPRICE
24              || '产品数量: ' || my_prdinfo_rec.QUANTITY
25          );
26      END LOOP;
27
28      EXCEPTION
29      WHEN NO_DATA_FOUND THEN
30          DBMS_OUTPUT.PUT_LINE('没有数据: ');
31      WHEN TOO_MANY_ROWS THEN
32          DBMS_OUTPUT.PUT_LINE('数据过多: ');
33      END ;
34 /

```

【代码解析】

- 第1~2行创建存储过程。存储过程包括IN类型的参数，表示该参数为输入类型。此时可以省略关键字IN。
- 第3~4行表示声明过程内部变量。其中，cur_ctgyid表示产品类型编码，而cur_prtinfo表示行类型的记录。
- 第7~9行表示用传进的参数param_ctgname作为查询条件从表CATEGORYINFO中找出对应的编码类型，并把该类型编码放到变量cur_ctgyid中。
- 第12~15行表示利用隐式游标的属性进行判断：如果有记录则将该类型名称输出到屏幕；如果没有记录或根据条件查询的记录过多，则会进入异常块部分。
- 第16~26行表示根据产品类型编码从表PRODUCTINFO中查询数据，并把得到的数据输出到屏幕。
- 第28~32行表示异常块部分。当出现NO_DATA_FOUND或TOO_MANY_ROWS异常时会中断操作进入异常部分。

【执行效果】

在SQL*Plus中创建该存储过程，并执行，得到的效果如图10.6所示。

从图10.6可以看到，当执行存储过程时输入了参数“洗衣机”，下面列出了3条相关数据。这是一个简单的示例，相信大家能很快掌握。

这里需要注意的是，如果存储过程有使用参数，那么调用该过程时一定要为其指定参数，



图10.6 示例7执行效果



否则会出现错误提示。不过可以采取一些措施避免此类错误的发生,例如使用默认值。接下来的示例将展示存储中的参数如何使用默认值。

【示例8】使用参数的默认值

本示例提供了根据输入的产品类型查询对应产品类型编码的功能。与上个示例不同的是,该示例中的输入参数使用了默认值功能。具体实现见如下脚本。

1) 创建函数。函数将返回“雨具”字符串。

```
01 CREATE OR REPLACE FUNCTION DEFT RETURN VARCHAR2
02 IS
03 BEGIN
04   DBMS_Output.PUT_LINE('----已进入到函数----');
05   DBMS_Output.PUT_LINE('默认类型是雨具');
06   RETURN '雨具';
07 END DEFT;
08 /
```

【代码解析】

□ 第1~2行表示创建一个函数,函数名为DEFT,返回值是VARCHAR2类型。

□ 第4~5行表示简单输出。当调用时会输出这些提示语句。

□ 第6行表示返回的值,这里返回“雨具”。也就是说,该函数返回值是“雨具”。

在SQL*Plus中执行该函数,并检验是否成功。函数的创建以及执行过程如图10.7所示。



图10.7 函数DEFT的创建执行过程

从图10.7中可以看出,该函数已经创建完成并成功返回设定值。除此之外,在SQL*Plus中继续执行如下创建过程脚本。

2) 创建存储过程。该过程输入产品类型,在表CATEGROYINFO中查询符合该产品类型的编码并输出到屏幕。

```
01 CREATE PROCEDURE PRODUCT_INTYPE_DEFT_PRC(param_ctgname
02                                     IN VARCHAR2 DEFAULT DEFT() )
03 AS
04   cur_ctgyid categroyinfo.categroyid%TYPE;
05
06 BEGIN
```



```

07 SELECT CATEGORYINFO.CATEGORYID INTO cur_ctgyid
08 FROM CATEGORYINFO
09 WHERE CATEGORYINFO.CATEGORYNAME = parm_ctgyname;
10
11 IF SQL%FOUND THEN
12     DBMS_OUTPUT.PUT_LINE('-----');
13     DBMS_OUTPUT.PUT_LINE(parm_ctgyname || '对应的编码是: ' || cur_ctgyid);
14 END IF;
15
16 EXCEPTION
17 WHEN NO_DATA_FOUND THEN
18     DBMS_OUTPUT.PUT_LINE('没有数据: ');
19 WHEN TOO_MANY_ROWS THEN
20     DBMS_OUTPUT.PUT_LINE('数据过多: ');
21 END ;
22 /

```

【代码解析】

- 第1~2行表示创建存储过程，该过程是带有IN类型的参数，并且有定义输入类型参数的默认值。该默认值为DEFT()函数。
- 第7~9行根据输入的参数值（产品类型）查询对应的产品类型编码，并放入变量cur_ctgyid中。
- 第11~14行表示如果有符合要求的记录则输出到屏幕。
- 第16~20行表示异常块，其中定义了两个异常，这里不再解释，读者可以参考前面的章节。本示例为输入参数指定了默认值。也就是说，调用该存储过程时，如果没有为其指定参数，则该过程使用自定义的默认值，这里会执行DEFT函数，利用它的返回值作为参数值。

【执行效果】

在SQL*Plus中执行该过程，效果如图10.8所示。



图10.8 示例8执行效果

从图10.8中可看到一共执行了两次该存储过程。第一次没有指定参数，但从效果中可以看出该过程没有提示错误，而是直接执行了DEFT函数的内容，并根据该函数返回的产品类型查询出了对应的产品类型编码。而第二次执行该过程指定了参数为“电视”，这次执行过程中并没有执行函数DEFT。由此可以看出，默认值只能在没有输入参数值的时候起作用。

提示

该过程使用了函数作为参数的默认值。其实，这里完全可以写一个确切的字符串作为默认值，如“雨具”。



存储过程中OUT类型的参数也被称为输出类型的参数。这样的存储过程会有相关返回值给调用它的程序，在程序中几乎可以把它当做一个变量使用，利用它给调用者中的变量初始化。下面的示例演示了如何使用该类型的参数。

【示例9】输出类型参数的使用

完成该示例需要两个步骤，第一部分就是被调用的存储过程，它里面包含OUT类型参数；第二部分就是调用者，这里也使用存储过程。详细实现过程如下：

1) 创建被调用的存储过程。该过程提供根据输入的产品类型查询出对应的产品类型编码功能，并将得到的编码放到输出参数中。相关脚本如下：

```
01 CREATE PROCEDURE PRODUCT_OUTTYPE_PRC (parm_ctgname IN VARCHAR2,
02                                     parm_ctgyid OUT VARCHAR2)
03 AS
04 BEGIN
05     SELECT CATEGORYINFO.CATEGORYID INTO parm_ctgyid
06     FROM CATEGORYINFO
07     WHERE CATEGORYINFO.CATEGORYNAME = parm_ctgname;
08     IF SQL%FOUND THEN
09         DBMS_OUTPUT.PUT_LINE('传出参数是: ' || parm_ctgyid);
10     END IF;
11
12 EXCEPTION
13 WHEN NO_DATA_FOUND THEN
14     DBMS_OUTPUT.PUT_LINE('没有数据: ');
15 WHEN TOO_MANY_ROWS THEN
16     DBMS_OUTPUT.PUT_LINE('数据过多: ');
17 END PRODUCT_OUTTYPE_PRC;
18 /
```

【代码解析】

- 第1~3行表示创建存储过程，该过程有两个参数，其中第一个parm_ctgname为输入参数，表示输入的是产品类型名称，而parm_ctgyid是输出参数，表示的是产品类型编码。
- 第5~7行表示根据产品名称查询产品编码。
- 第12~16行表示异常部分。

2) 创建调用存储过程。该过程根据输入的产品类型以及价格从表PRODUCTINFO中查询符合要求的数据并输出到屏幕。详细脚本如下：

```
01 CREATE PROCEDURE PRODUCT_CLOUTTYPE_PRC (parm_ctgname IN VARCHAR2,
02                                     parm_pric NUMBER)
03 AS
04 cur_ctgyid categoryinfo.categoryid%TYPE;           --存放产品类型编码
05 cur_prinfo productinfo%ROWTYPE;                   --存放表productinfo的行记录
06
07 BEGIN
08     PRODUCT_OUTTYPE_PRC (parm_ctgname, cur_ctgyid);
09     IF SQL%FOUND THEN
10         DBMS_OUTPUT.PUT_LINE('-----');
11         DBMS_OUTPUT.PUT_LINE (parm_ctgname || '对应的编码是: ' || cur_ctgyid);
```



```

12 END IF;
13 FOR my_prdinfo_rec IN
14 (
15     SELECT * FROM PRODUCTINFO
16     WHERE CATEGORY = cur_ctgyid
17     AND PRODUCTINFO.PRODUCTPRICE < parm_pric
18 )
19 LOOP
20     DBMS_OUTPUT.PUT_LINE(
21         '产品名称: ' || my_prdinfo_rec.PRODUCTNAME
22         || ' 产品价格: ' || my_prdinfo_rec.PRODUCTPRICE
23         || ' 产品数量: ' || my_prdinfo_rec.QUANTITY
24     );
25     END LOOP;
26 END ;
27 /

```

【代码解析】

- 第1~3行是创建过程，该过程有两个参数，分别为parm_ctgname和parm_ctgyid，这两个参数都是输入类型的参数。
- 第4~5行表示声明两个变量。
- 第8行表示调用过程PRODUCT_OUTTYPE_PRC，该过程需要两个参数。其中，cur_ctgyid占用了输出参数的位置。也就是说，该过程的返回值将初始化给变量cur_ctgyid。
- 第9~12行判断并输出产品类型名称对应编码。
- 第13~25行表示循环输出符合记录的数据。其中，第15~17行表示查询指定产品类型以及低于指定价格的数据。

【执行效果】

在SQL*Plus中成功创建以上两个存储过程，并执行如下脚本调用PRODUCT_CLOUTTYPE_PRC过程。查询价格在3000以下的洗衣机相关记录。脚本如下：

```
EXEC PRODUCT_CLOUTTYPE_PRC('洗衣机',3000);
```

执行效果见图10.9。

使用OUT类型参数时需要注意以下两点：

- 1) OUT类型的参数需要用变量填充，而不能用常量或表达式填充。
- 2) 如果过程中有被调用的过程因发生未处理的异常而退出，那么调用者通常得不到任何OUT参数的值（包括发生异常之前已经得到的OUT类型的参数值），而在发生的异常被处理的情况下退出，那么之前得到的OUT的参数值将会被调用者获得。

存储过程还有一种参数类型，是输入输出类型，也叫IN OUT类型参数。此类型的参数同OUT类型参数性质相似，不过它既支持输入也支持输出。以下就是该类型参数的示例。



图10.9 示例9执行效果

【示例10】使用输入输出类型的参数

下面演示如何使用输入输出类型参数，完成该示例同样需要两个步骤。详细实现过程如下：

1) 创建被调用的存储过程。该过程提供根据输入的产品类型编码和降价比例对表 PRODUCTINFO 的数据进行降价修改，并返回修改的记录数。脚本如下：

```
01 CREATE PROCEDURE PRODUCT_INOUTTYPE_PRC(parm_ctgyid IN VARCHAR2,
02                                     parmparm_pric IN OUT NUMBER)
03 AS
04 BEGIN
05     UPDATE PRODUCTINFO
06     SET PRODUCTPRICE = PRODUCTINFO.PRODUCTPRICE*(1-parmparm_pric)
07     WHERE PRODUCTINFO.CATEGORY = parm_ctgyid;
08     IF SQL%FOUND THEN
09         parmparm_pric := SQL%ROWCOUNT;
10     END IF;
11 END PRODUCT_INOUTTYPE_PRC;
```

【代码解析】

□ 第1~3行表示创建存储过程，其中包含两个参数：parm_ctgyid表示产品类型编码是输入参数，parmparm_pric用来输入时表示降价比例，而用来输出时则表示修改语句修改的记录数。

□ 第5~7行表示根据提供的产品类型编码和降价比例修改PRODUCTINFO的价格记录。

□ 第9行表示利用游标的属性得到修改的记录数。该记录数实际赋值给了该存储过程的输入输出参数parmparm_pric。

2) 创建调用存储过程。该过程输入产品类型名称，根据产品类型名称查询出产品类型编码并调用PRODUCT_INOUTTYPE_PRC存储过程。最终输出修改的记录数。脚本如下：

```
01 CREATE PROCEDURE PRODUCT_CLINOUTTYPE_PRC(parm_ctgyname IN VARCHAR2)
02
03 AS
04     cur_ctgyid categoryinfo.categoryid%TYPE; --存放产品类型编码
05     cur_pric number;
06
07 BEGIN
08     SELECT CATEGORYINFO.CATEGORYID INTO cur_ctgyid
09     FROM CATEGORYINFO
10     WHERE CATEGORYINFO.CATEGORYNAME = parm_ctgyname;
11
12     PRODUCT_INOUTTYPE_PRC(cur_ctgyid,cur_pric);
13     IF cur_pric > 0 THEN
14         DBMS_OUTPUT.PUT_LINE('共修改'|| cur_pric || '条记录。');
15     END IF;
16
17 EXCEPTION
18     WHEN NO_DATA_FOUND THEN
19         DBMS_OUTPUT.PUT_LINE('没有数据');
20     WHEN TOO_MANY_ROWS THEN
```

```

21 DBMS_OUTPUT.PUT_LINE('数据过多！');
22 END;
23 /

```

【代码解析】

- 第1~3行创建存储过程，参数表示产品类型名称。
- 第4~5行声明变量。cur_ctgyid用于存放产品类型编码，cur_pric用于存储在存储过程PRODUCT_INOUTTYPE_PRC返回的记录数。
- 第8~10行表示根据产品类型查询产品编码。
- 第12行表示调用存储过程PRODUCT_INOUTTYPE_PRC，此时利用返回值初始化cur_pric变量。
- 第13~15行表示利用条件语句判断后输出修改的记录数。
- 第17~21行表示异常块。

该示例同示例9的操作方式差别不大，接下来查看执行效果。

【执行效果】

在SQL*Plus中创建以上两个存储过程，确认没有错误提示后执行该过程。效果如图10.10所示。

给存储过程的参数赋值时，除了以上接触到的按照存储过程的参数顺序赋值之外，还有另外一种赋值的方式。这种方式可以看做显式地为某个变量赋值，它指定了赋值的参数对象，因此这种方式不用考虑原存储过程中参数的顺序。例如，在该示例中PRODUCT_CLINOUTTYPE_PRC过程的第12行，利用如下的赋值方式可以达到同样的效果：



图10.10 示例10执行效果

```
PRODUCT_INOUTTYPE_PRC(param_pric=>cur_pric,param_ctgyid=>cur_ctgyid);
```

这段脚本表示的是参数param_pric的值是变量cur_pric，而参数param_ctgyid的值是变量cur_ctgyid。这样赋值的结果和以前没有差异，读者可以根据实际需求自行把握。

10.3 使用PL/SQL工具创建存储过程

如果有了10.2节的基础，那么在PL/SQL Developer中操作存储过程就显得比较简单。该工具提供了一个相对便利的操作环境，尤其对于存储过程的调试和错误的查找都比SQL*Plus操作方便。本节就介绍如何在PL/SQL Developer中创建、调试存储过程。

10.3.1 在PL/SQL Developer中创建存储过程

PL/SQL Developer提供了创建存储过程模板，该模板允许我们输入存储过程名和参数，然后自动创建脚本。该脚本只是一个简单框架，细节则需要开发者自己完成。示例11演示了如何创建存储过程。

【示例11】使用PL/SQL Developer创建存储过程

创建存储过程分为如下9个步骤：

1) 启动PL/SQL Developer工具, 在整个布局的左上角单击【File】|【New】|【Program Window】菜单, 在列表中找到【Procedure】选项, 如图10.11所示。

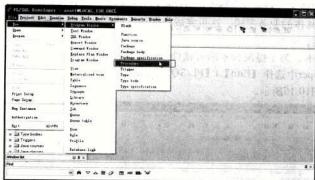


图10.11 PL/SQL Developer中的存储过程选项

2) 单击该选项, 弹出图10.12所示的过程模板窗口。

☐ Name: 表示创建的过程名称。

☐ Parameters: 过程的参数列表。

3) 填写给出的两项信息。填写完成后如图10.13所示。



图10.12 过程创建模板窗口



图10.13 过程的模板参数填写

4) 单击【OK】按钮。在Program窗口生成过程的PL/SQL脚本, 如图10.14所示。

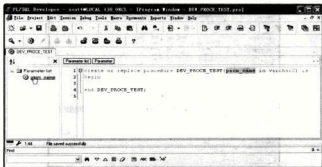


图10.14 按照参数生成的过程脚本



从图10.14中可以发现该窗口垂直分成了两部分，其中左边部分显示过程的结构，而右边部分则显示具体的脚本。当鼠标在左边窗口选中某部分结构时，右边窗口对应部分也会出现选中的情况，如图10.14所示。利用这个特性可以快速定位脚本，这和函数的创建页面布局一样。

5) 手工填写具体的业务逻辑。这里完成根据输入的产品类型名称获取产品类型编码，并输出到屏幕的功能。

6) 格式化脚本。为了提高脚本的可读性以及美观度，开发人员通常都会对脚本格式化，在PL/SQL Developer中选择【Edit】|【PL/SQL Beautifier】菜单，这时选中的脚本会自动进行格式化。效果如图10.15所示。



图10.15 格式化以后的脚本

细心的读者会发现窗口中脚本的关键词都是大写，这需要相应的设置。单击PL/SQL Developer的【Tools】|【Preferences】菜单，在弹出的窗口左边的选项列表中找到【User Interface】下的【Editor】子列表并单击，同时在右边窗口中找到【Keyword case】选项，选择下拉列表中的【Uppercase】选项，设置页面如图10.16所示。单击窗口中的【OK】按钮进行保存。

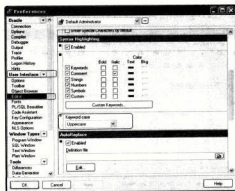


图10.16 关键词大写相关设置

7) 修改存储过程的错误。存储过程的编写并不是每次都能成功, 出现语法错误而导致存储过程不能使用是很常见的现象。当按F8键执行出现编译错误时, 在Program窗口的下方会出现错误提示, 可以根据该提示对程序进行修改。例如, 本示例出现了图10.17所示的错误提示。



图10.17 存储过程编译错误提示

箭头指向的位置是错误提示列表框, 当选中某项错误记录时, 脚本窗口也会高亮显示系统认为出错的行 (有可能出错的地方在提示行附近)。根据经验对其判断修改即可。这里的错误是DBMS_OUTPUT.PUT_LINE脚本本少了个字母E。

8) 创建存储过程。排除脚本的错误后按F8键执行。如果没有错误, 会提示创建成功。并且在PL/SQL Developer左边【Objects】透视图中的Procedures文件夹下看到创建成功的存储过程。具体效果如图10.18所示。

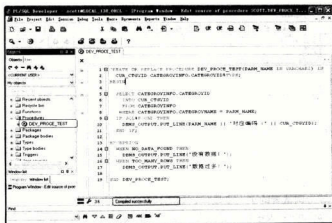


图10.18 成功创建存储过程



9) 执行存储过程。如果存储过程不需要测试，仅仅执行，那么可在SQL窗口使用如下语句块调用存储过程：

```
BEGIN
    dev_proce_test('洗衣机');
END;
```

其结果可以输出到SQL窗口的Output标签页。除了这种执行方式，也可以通过Command窗口利用EXEC关键词执行存储过程。

10.3.2 调试存储过程

调试存储过程是开发者必备的技能，它和语言编程的调试功能类似，可以设置断点，可以分步执行。这些都能帮助我们弄清楚脚本的执行过程。

【示例12】调试存储过程

调试存储过程的步骤分为以下3步：

1) 设置断点。在PL/SQL Developer中找到调试对象。在【Objects】下找到Procedures文件夹，从中选中要调试的存储过程对象，右击该对象，在弹出的快捷菜单中选择【View】选项。此时在右边窗口可以看到整个存储过程的脚本，把光标移到欲设置断点的行上然后使用Ctrl+B快捷键设置断点。具体效果如图10.19所示。

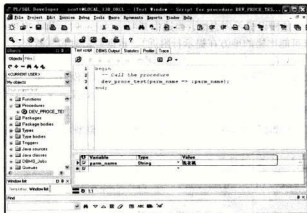


图10.19 设置断点

2) 进入调试界面。选中欲调试的存储过程并右击，从弹出的快捷菜单中选择【Test】选项进入调试页面，如图10.20所示。如果调试的过程指定了参数，那么首先需要输入参数值，可见黑框部分。

3) 开始调试。按F9键开始调试，利用Ctrl+R快捷键运行脚本，这时程序会在第一个经过的断点处停止，等待调试者的下一步操作，如图10.21所示。

此时可以在Variable列表中输入想查看的变量（箭头指向部分），在调试过程中可以利用以下方式帮助调试：

□ Step into：单步调试，每次执行一条语句。



- ☐ Step over: 不进入函数内, 直接跳过并输出结果。
- ☐ Step out: 跳出过程。
- ☐ Run to next exception: 运行到下一断点处。



图10.20 进入调试界面

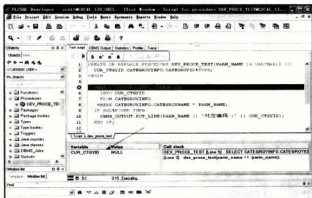


图10.21 开始调试

这4种方式的执行按钮在图中黑框部分, 读者可以一一对应, 调试过程中可以随时利用Ctrl+B快捷键设置或取消断点。

调试程序不仅可以帮我们了解变量的动态情况, 更主要的是可以帮我们了解执行流程, 久而久之对理解更深层的脚本运作都有帮助。

10.4 修改、删除存储过程

存储过程的删除和修改操作很简单, 但是很实用。当根据业务改变存储过程的脚本时几乎不用刻意地考虑这方面就能达到预期效果。

10.4.1 修改存储过程

读书笔记

修改存储过程内容要利用REPLACE关键词，使用方式可以参考10.1.3小节的语法。在SQL*Plus中创建过程时需要开发人员自行带上OR REPLACE关键词，从而完成过程的修改，也就是覆盖。

在PL/SQL Developer中开发人员甚至都不用考虑这方面的内容，选择编辑过程时模板生成的脚本是带有该关键词的。示例13演示了如何修改一个存储过程。

【示例13】修改存储过程

修改存储过程分为以下3个步骤：

1) 选择编辑对象。在【Objects】下的Procedures文件夹中选中准备编辑的存储过程，右击该过程，在弹出的快捷菜单中选择【Edit】选项，如图10.22所示。



图10.22 选择编辑对象

2) 修改存储过程。进入编辑页面后可以发现存储过程使用了OR REPLACE关键词（即使创建时没有该关键词）。如图10.23第一个黑框所示。



图10.23 修改存储过程

修改这个存储过程，例如在第11行添加新的输出语句。

3) 重新创建该过程。修改完成后可以按F8键创建新的存储过程，至此完成存储过程的修改。

10.4.2 删除存储过程

不用的存储过程可以将其删除，删除过程同样十分容易操作。本小节列出了两种操作方式供读者使用。



1) 利用SQL语句删除存储过程。

删除操作语法如下：

```
DROP PROCEDURE [schema.]procedure_name
```

□ schema：表示存储过程的所属机构。

□ procedure_name：表示要删除存储过程的名称。

在SQL窗口或者SQL*Plus中执行如下语句表示删除存储过程DEV_PROCE_TEST。

```
DROP PROCEDURE DEV_PROCE_TEST;
```

2) 利用工具删除存储过程。这种方式是在PL/SQL Developer中选中需要删除的存储过程，右击该过程，在弹出的快捷菜单中选择【Drop】选项，此时会出现确认删除窗口，单击【Yes】按钮即可。

注意

当存储过程有调用关系时，如果删除被调用者，那么重新编译调用者会出现错误。所以做删除操作时最好认清各过程之间的依赖关系。

10.5 小结

存储过程的使用十分普遍，它用做批量处理任务和内部数据转换时效率很高，也是Oracle的重点之一。本章从基础开始讲解，不仅介绍了存储过程的概念和作用，而且还比较详细地介绍了如何创建和操作各种类型的存储过程。本章的示例比较丰富，相信对基础较弱者有很大的帮助。存储过程的使用方式灵活多变，读者在实际开发中应多读代码学习不同的使用方式，达到融会贯通的目的。

10.6 习题

一、填空题

1. 存储过程参数分为_____、_____和_____3种类型。
2. SQL*Plus使用_____命令来执行存储过程。
3. SQL*Plus使用_____命令来显示错误信息。

二、选择题

1. OUT类型的参数可以使用下面哪项填充？（ ）
A. 常量 B. 变量
C. 初始化后的变量 D. 函数
2. 有输入参数的存储过程调用时能否不带参数？（ ）
A. 不能 B. 能

三、简答题

1. 为什么使用存储过程？
2. 存储过程和语句块有什么区别？

第11章 触发器——保证数据的正确性

触发器在Oracle开发中使用也比较普遍,利用它可以得到数据变更的日志记录,还可以强制保证数据的一致性。本章将介绍触发器的作用和使用,主要包含以下知识点:

- ☐ 什么是触发器
- ☐ 触发器的作用
- ☐ 触发器的语法
- ☐ 如何使用触发器

11.1 什么是触发器

学习存储过程后,认识和了解触发器并不复杂,它们之间有着相似之处。相信通过本节对触发器的介绍,读者无论在概念上还是在语法上对触发器都会有一个初步的认识。

11.1.1 认识触发器

前面介绍过存储过程。其实触发器和存储过程比较类似,它由PL/SQL编写并存储在数据库中,它可以调用存储过程,但触发器本身的调用和存储过程调用却是不一样的。存储过程由用户、应用程序、触发器或其他过程调用。但触发器只能由数据库的特定事件来触发。所谓的特定事件主要包括如下几种类型的事件。

1) 用户在指定的表或视图中做DML操作,主要包括如下几种:

- ☐ INSERT操作,在特定的表或视图中增加数据。
- ☐ UPDATE操作,对特定的表或视图修改数据。
- ☐ DELETE操作,删除特定表或视图的数据。

2) 用户做DDL操作,主要包括如下几种:

- ☐ CREATE操作,创建对象。
- ☐ ALTER操作,修改对象。
- ☐ DROP操作,删除对象。

3) 数据库事件,主要包括如下几种:

- ☐ LOGON/LOGOFF用户的登录或注销。
- ☐ STARTUP/SHUTDOWN数据库的打开或关闭。
- ☐ ERRORS特定的错误消息等。

在以上事件中的一种或多种发生时就能使触发器运行。

11.1.2 触发器的作用

触发器可以根据不同的事件进行调用,它有着更加精细的控制能力,这种特性可以帮助开发人员完成很多普通PL/SQL语句完成不了的功能。下面介绍一下触发器的主要作用。



- ☐ 自动生成自增长字段。例如，在表中插入数据前得到序列的最大值和数据库同时插入表中，避免该序列的重复。
- ☐ 执行更复杂的业务逻辑。普通的操作方式只能完成固定的数据变动，而使用触发器则在完成的基础功能上做额外的操作，以达到完成特殊业务的目的。
- ☐ 防止无意义的数据库操作。利用触发器可以把符合某些条件的数据库操作加以限制，使其不能变动。
- ☐ 提供审计。利用触发器可以跟踪对数据库的操作，也可以在指定的表或视图记录改变时，利用触发器把数据库变动日志记录下来。
- ☐ 允许或限制修改某些表。利用触发器可以限制表的变动。
- ☐ 实现完整性规则。当一个表中的数据有变动时可以利用触发器修改这些变动数据在其他表中的关联数据（正常情况下可以利用外键进行限制）。
- ☐ 保证数据的同步复制。

注意 建议开发人员只在必要时使用触发器，因为触发器可能造成比较复杂的相关依赖性，这种情况在大型的数据库中可能会带来麻烦。例如，某个触发器的触发很可能造成多个触发器的连锁触发，一旦这种连锁触发超过32个就会出现异常。

11.1.3 触发器的类型

触发器可分为5种类型，具体内容如下：

- ☐ 数据操纵语言（DML）触发器。此种类型触发器定义到表上，当对表执行INSERT、UPDATE、DELETE操作时可以激发该类型的触发器。利用该类触发器可以复制、检查、替换某种符合指定条件的数据。按照触发级别可以分为两种方式，第一种为行级触发器，此种类型表示每条记录修改时都会激发该触发器；第二种为语句级触发器，此种类型表示当SQL语句执行时会激发该触发器，与修改多少条记录没有关系。如果以数据的更改事件为准，则分为BEFORE和AFTER两种类型。
- ☐ 数据定义语言（DDL）触发器。当CREATE、ALTER、DROP模式对象时会触发相关的触发器，在Oracle中可以简单地理解一个用户就有一个和它同名的模式，利用它可以使得某些表不能被修改或删除。
- ☐ 复合触发器。此种类型的触发器是Oracle 11g的新特性，它相当于在一个触发器中包含了4种类型的触发器，其中包含了BEFORE类型的语句级、BEFORE类型的行级、AFTER类型的语句级、AFTER类型的行级。这种把所有触发器都放到一个代码块中的做法使得变量的传递变得更加方便。
- ☐ INSTEAD OF触发器。此种类型触发器通常作用在视图上。对由多个源表的视图做DML操作通常是不被允许的，如果遇到这种情况就可以利用INSTEAD OF类型触发器解决问题。利用它可以把对视图的DML操作转换成对多个源表进行操作。
- ☐ 用户和系统事件触发器。作用在数据库上由数据库事件激发的触发器，如登录和注销事件的触发器。利用它可以记录数据库的登录情况。

本小节简单介绍了触发器的类型，也有些机构把DML和DDL类型的触发器直接分为行触发器、语句触发器、BEFORE触发器、AFTER触发器。



11.1.4 触发器的语法

了解语法是使用触发器的第一步，它就像一个公式，能让我们迅速地创建出自己的触发器。

下面列出了几种触发器的语法。

1) DML触发器的主要语法如下：

```
01 CREATE [ OR REPLACE ] TRIGGER [schema.] trigger
02 {BEFORE | AFTER | INSTEAD OF}
03 {DELETE | INSERT | UPDATE
04   [OF column [, column]...]}
05 }
06 [ OR {DELETE | INSERT | UPDATE
07   [OF column [, column]...]}
08 ]
09 ...
10 {ON [schema.]table | [schema.] view}
11 [FOR EACH ROW ]
12 [FOLLOWS [ schema.] trigger [, [ schema.] trigger ]...]
13 [ENABLE | DISABLE]
14 [WHEN (condition)]
15 trigger_body
```

【语法说明】

- OR REPLACE：新建的触发器可以覆盖原有同名触发器。
 - TRIGGER：创建触发器的关键词。
 - schema：触发器所属模式（可简单看成用户名），如果不加该项则表示该触发器属于自己。
 - BEFORE：触发器类型为前触发。
 - AFTER：触发器类型为后触发。
 - INSTEAD OF：表示触发器类型为替换类型。
 - DELETE | INSERT | UPDATE：表示触发的事件。
 - [OF column [, column]]：触发条件具体到的某列。
 - 第6~9行表示可以追加多个条件。
 - ON [schema.]table | [schema.] view：该触发器作用的表或视图，INSTEAD OF类型可以作用在视图上。
 - FOR EACH ROW：表示行级触发器，省略则为语句级触发器。
 - FOLLOWS [schema.] trigger：触发器执行的顺序。
 - ENABLE | DISABLE：设置触发器是否可用状态。
 - WHEN (condition)：触发该触发器的条件。
 - trigger_body：表示触发器的函数体。
- 2) DDL和数据库事件触发器语法如下：

```
01 CREATE [ OR REPLACE ] TRIGGER [schema.] trigger
02 { BEFORE | AFTER }
03 { ddl_event [OR ddl_event]...
04   | database_event [OR database_event]...
05 }
```




```

06 ON { [schema.] SCHEMA
07     | DATABASE
08     }
09 [FOLLOWS [ schema.] trigger [, [ schema.] trigger] ...]
10 [ENABLE | DISABLE]
11 [WHEN (condition)]
12 trigger_body

```

【语法说明】

- ❑ OR REPLACE: 新建的触发器可以覆盖原有同名触发器。
- ❑ TRIGGER: 创建触发器的关键词。
- ❑ schema: 触发器所属模式，如果不加该项则表示该触发器属于自己。
- ❑ BEFORE: 触发器类型为前触发。
- ❑ AFTER: 触发器类型为后触发。
- ❑ ddl_event [OR ddl_event]: DDL事件，用OR连接。
- ❑ database_event[OR database_event]: 数据库事件，用OR连接。
- ❑ [schema.] SCHEMA | DATABASE: 触发器可作用在模式上或数据库上。
- ❑ FOLLOWS [schema.] trigger: 触发器执行的顺序。
- ❑ ENABLE | DISABLE: 设置触发器是否可用状态。
- ❑ WHEN (condition): 触发该触发器的条件。
- ❑ trigger_body: 表示触发器的函数体。

表11.1列出了一部分DDL事件，这些事件都可以激发触发器。

表11.1 DDL事件列表

DDL事件	简 介
ALTER	修改对象，例如修改对象的名称约束等
ANALYSE	用来分析统计信息
AUDIT/ NOAUDIT	启用或取消审计
COMMENT	注解列或表的含义
CREATE	创建对象
DROP	删除对象
GRANT	授权操作
RENAME	修改对象名称
REVOKE	取消权限
TRUNCATE	删除行记录

表11.2列出了部分数据库事件，这些事件也会激发触发器。

触发器由三部分组成。它们分别是触发事件或语句、触发器限制、触发器动作。

- ❑ 触发事件或语句是指激发触发器的动作，具体指11.1.1小节介绍的事件操作。
- ❑ 触发器限制指的是WHEN后面的条件，当条件为TRUE时，该触发器会被激发。
- ❑ 触发器动作就是一段过程，当触发器被激发时运行的trigger_body部分。



表11.2 数据库事件列表

数据库事件	简介
STARTUP	数据库打开后被触发，模式下不可以
SHUTDOWN	数据库关闭前被触发，模式下不可以
LOGON	客户程序登录后触发
LOGOFF	客户程序注销前触发
SERVERERROR	错误消息出现后触发

3) 复合触发器也是DML触发器的一种，它的主要语法如下：

```
01 CREATE [OR REPLACE] TRIGGER [schema.] trigger
02 FOR
03 { DELETE | INSERT | UPDATE
04   [OF column [, column ]...]
05 }
06 [OR {DELETE | INSERT | UPDATE
07   [OF column [, column]...]
08 }
09 ]...
10 ON { [schema.]table
11     | [schema.] view
12 }
13 COMPOUND TRIGGER
14 { BEFORE STATEMENT IS tps_body END BEFORE STATEMENT}
15 | BEFORE EACH ROW IS tps_body END BEFORE EACH ROW
16 | AFTER STATEMENT IS tps_body END AFTER STATEMENT
17 | AFTER EACH ROW IS tps_body END AFTER EACH ROW
18 }
```

【语法说明】

□ OR REPLACE：新建的触发器可以覆盖原有同名触发器。

□ TRIGGER：创建触发器的关键词。

□ schema：触发器所属模式，如果不加该项则表示该触发器属于自己。

□ DELETE | INSERT | UPDATE：表示触发事件。

□ COMPOUND TRIGGER：定义触发器时表示为复合类型触发器。

□ BEFORE STATEMENT：前语句级触发。

□ BEFORE EACH ROW：前行级触发。

□ AFTER STATEMENT：后语句级触发。

□ AFTER EACH ROW：后行级触发。

□ tps_body：具体语句或程序。

该类型是Oracle 11g新增加的触发器类型，比较方便地处理4个时间点，这将在后面章节介绍。

11.2 使用SQL*Plus操作触发器

本节将利用SQL*Plus创建常用的触发器，包括前触发、后触发、行级触发、语句级触发



等，相信读者学习这些示例后能掌握触发器的使用及创建方式。

11.2.1 利用SQL*Plus创建触发器

创建触发器的首要条件是要有相关的权限。用户模式下如果想在自己的对象上创建触发器，则必须具有CREATE TRIGGER系统权限，如果想在其他用户上创建触发器，则需要有CREATE ANY TRIGGER权限。除此之外，如果在数据库上创建触发器，则需要有ADMINISTER DATABASE TRIGGER系统权限。

【示例1】一个简单的触发器

这是一个入门的触发器示例，该触发器在执行删除操作时触发，作用在PRODUCTINFO表上。具体脚本如下：

```
01 CREATE TRIGGER FIRST_TGR
02     AFTER DELETE
03     ON PRODUCTINFO
04     BEGIN
05     IF DELETING THEN
06     DBMS_OUTPUT.put_line('删除数据操作: ');
07     END IF;
08     END;
09 /
```

【代码解析】

- 第1行表示创建名为FIRST_TGR的触发器。
- 第2行表示触发器类型为后触发，触发事件是删除操作。
- 第3行表示触发器作用的表是PRODUCTINFO。
- 第5~7行表示如果对表PRODUCTINFO执行删除操作则输出第6行提示。

【执行效果】

在SQL*Plus中执行上述脚本，如果成功会有如下提示：

触发器已创建

【验证触发器】

一旦触发器创建成功，那么对表PRODUCTINFO执行删除操作则会激发该触发器，因为是语句级的触发器，所以激发该触发器的时机是在删除操作之后。如果触发，则输出脚本中第6行的提示。打开SQL*Plus的输出状态，执行如下删除操作：

```
DELETE FROM PRODUCTINFO WHERE PRODUCTID = '0240090001';
```

整个操作流程如图11.1所示。

细心的读者从图11.1中会发现，删除操作中实际并没有数据被删除，但依然激发了触发器。也就是说，语句触发器是删除这个事件本身激发了触发器，而和数据是否真的被删除没有必然联系。

11.2.2 查看触发器

在SQL*Plus下如果有查看触发器的需求，可以先



图11.1 示例1创建执行过程

查看本用户下所有触发器名称，在知道触发器名称的基础上就可以查看其具体的内容了。

【示例2】查看触发器

查看已经存在的触发器的源代码，分为两个步骤。

1) 查看触发器的名称。执行如下脚本：

```
SELECT OBJECT_NAME FROM USER_OBJECTS
WHERE OBJECT_TYPE = 'TRIGGER'
```

2) 查看触发器内容。有了触发器名称就可以查看其具体内容，下面脚本中的FIRST_TGR就是第一步查询出来的结果。执行如下脚本：

```
SELECT * FROM USER_SOURCE WHERE NAME = 'FIRST_TGR' ORDER BY LINE ;
```

通过以上两步就能查看已有触发器的内容了，整个流程如图11.2所示。

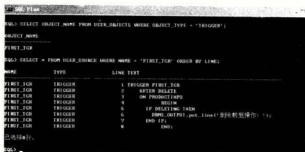


图11.2 查看触发器内容流程

11.2.3 DML类型触发器

该类型触发器在日常开发中比较常用，前面已经介绍过DML触发器主要针对表的INSERT、UPDATE、DELETE操作。本小节将演示多个示例以增强对该类型触发器的了解，通过该小节将理解什么是前触发、后触发以及行级、语句级触发等。

【示例3】创建行级触发器

要求创建行级触发器，当在PRODUCTINFO表中增加数据时将激发该触发器，并把所做的操作记录到表OPERATION_LOG中。该触发器属于单一事件的触发器。创建行级触发器分为以下3个步骤：

1) 创建操作事件记录表。该表数据结构如表11.3所示。

表11.3 LOG_TAB表结构

字段名	中文释义	数据类型
ID	记录ID，主键	varchar2(10)
OPER_TABLE	被操作的表名	varchar2(20)
OPER_TABLE_PK	被操作表的主键	varchar2(50)
OPER_KD	操作类型	varchar2(10)
OPER_DATE	操作时间	date

**【语法说明】**

- ☐ 记录ID: 该ID作为LOG_TAB表的主键, 触发器中实现该功能比较可取的方式是使用序列自动生成。
 - ☐ 被操作的表名: 此字段记录了DML操作的表。
 - ☐ 被操作表的主键: 利用主键可以得到具体的记录, 非常有实用价值。
 - ☐ OPER_KD: 记录INSERT、UPDATE、DELETE操作。
 - ☐ OPER_DATE: 记录DML操作的时间, 这里直接使用Oracle的系统时间。
- 登录SQL*Plus, 执行如下脚本, 创建事件记录表LOG_TAB。

```

01 CREATE TABLE LOG_TAB
02 (
03     ID                VARCHAR2(10) not null,
04     OPER_TABLE        VARCHAR2(20),
05     OPER_KD           VARCHAR2(10),
06     OPER_TABLE_PRK    VARCHAR2(50),
07     OPER_DATE         DATE,
08     constraint LOG_TAB_PRK primary key (ID)
09 )
10 /

```

如果执行成功将会出现如下字样:

表已创建

2) 创建用做LOG_TAB表主键的自增长序列。

在SQL*Plus中执行如下脚本:

```

01 CREATE SEQUENCE LOG_TAB_ID
02 MINVALUE 1000000000
03 MAXVALUE 9999999999
04 START WITH 1000000000
05 INCREMENT BY 1
06 /

```

如果执行成功将会出现如下字样:

序列已创建

【代码解析】

- ☐ 第1行表示创建名为LOG_TAB_ID的序列。
- ☐ 第2行表示该序列的最小值为1 000 000 000。
- ☐ 第3行表示该序列的最大值为9 999 999 999。
- ☐ 第4行表示该序列的值从1 000 000 000开始。
- ☐ 第5行表示该序列增长量为1。

表中主键字段的值不允许重复, 在Oracle中不能直接设置某字段为自增长字段以达到符合主键值要求的目的, 这种情况下的解决方法就是利用序列生成的值作为表主键的值。

3) 创建触发器。

```

01 CREATE TRIGGER PRODUCTINFO_OPER_TGR

```



```
02 BEFORE INSERT
03 ON PRODUCTINFO
04 FOR EACH ROW
05 BEGIN
06     IF INSERTING THEN
07         INSERT INTO LOG_TAB
08             VALUES
09                 (LOG_TAB_ID.NEXTVAL,
10                  'PRODUCTINFO',
11                  'INSERT',
12                  :NEW.PRODUCTID,
13                  SYSDATE);
14         DBMS_OUTPUT.put_line('插入数据主键是 ' || :new.PRODUCTID);
15     END IF;
16 END;
17 /
```

【代码解析】

- 第1行表示创建名为PRODUCTINFO_OPER_TGR的触发器。
- 第2行表示该触发器为前触发，触发事件是INSERT。
- 第3行表示该触发器作用在表PRODUCTINFO上。
- 第4行表示该触发器为行级触发，也就是说每增加一行就会触发一次。
- 第6行表示判断如果是插入数据操作则进入IF语句。
- 第7~13行表示向事件记录表中增加数据。其中第9行的LOG_TAB_ID.NEXTVAL表示得到序列的下一个值，第13行的SYSDATE表示增加数据时的系统时间。
- 第14行输出增加数据的PRODUCTID字段值，该字段是PRODUCTINFO表的主键。行级的触发器里使用:new或:old来访问变更前和变更后的数据。其中，如果是增加新记录操作，则只有:new可以访问；如果是修改操作，则:new和:old都可以访问，:new表示修改后的记录，:old表示修改前的记录；而删除则只有:old可以访问，因为该操作是删除已有的记录。

【执行效果】

在SQL*Plus中执行成功后提示如下：

触发器已创建

【验证触发器】

在表PRODUCTINFO中增加数据并查看是否正常激发触发器。脚本如下：

```
INSERT INTO PRODUCTINFO VALUES('0240090001','触发器测试',1000,0,0000000000,'','测试');
```

【执行效果】

插入数据完成，主键是 0240090001

已创建 1 行

继续查询表PRODUCTINFO和表LOG_TAB，检查是否有对应数据增加。两个表查询结果见图11.3。

以上是一个入门型的示例，该触发器中只包含了一种触发事件，就是增加数据的时候触发。但我们可以在一个触发器中添加多个触发事件，只需要把它们利用条件语句分开即可。下面一个示例演示了如何在一个触发器中使用多种触发事件。



SQL*Plus

```
SQL> SELECT * FROM PRODUCTINFO WHERE PRODUCTID = '0260000001';
```

PRODUCTID	PRODUCTNAME	PRODUCTPRICE	QUANTITY	CATEGORY	DESCRIPTION	ORIGIN
0260000001	精英笔记本电脑	10000	0.0			测试


```
SQL> SELECT * FROM LOG_TAB;
```

ID	OPER_TABLE	OPER_OP	OPER_TABLE_FPK	OPER_DATE
0000000002	PRODUCTINFO	INSERT	0260000001	09 6月 -10

SQL>

图11.3 两个表新增的数据

【示例4】在触发器中使用多种触发事件

把示例3改为当增加、修改、删除记录时都能触发，记录操作事件的表（LOG_TAB）不变。该类型触发器属于多个事件的触发器。修改后的脚本如下：

```
01 CREATE TRIGGER PRODUCTINFO_OPER_DML_TGR
02 AFTER INSERT OR UPDATE OR DELETE
03 ON PRODUCTINFO
04 FOR EACH ROW
05 BEGIN
06 CASE
07 WHEN INSERTING THEN --增加操作
08 INSERT INTO LOG_TAB
09 VALUES
10 (LOG_TAB_ID.NEXTVAL,
11 'PRODUCTINFO',
12 'INSERT',
13 :NEW.PRODUCTID,
14 SYSDATE);
15 DBMS_OUTPUT.PUT_LINE('插入数据完成，主键是 ' || :NEW.PRODUCTID);
16 WHEN UPDATING THEN --修改操作
17 INSERT INTO LOG_TAB
18 VALUES
19 (LOG_TAB_ID.NEXTVAL,
20 'PRODUCTINFO',
21 'UPDATE',
22 :OLD.PRODUCTID,
23 SYSDATE);
24 DBMS_OUTPUT.PUT_LINE('修改数据完成，修改数据主键是 ' || :OLD.PRODUCTID);
25 WHEN DELETING THEN --删除操作
26 INSERT INTO LOG_TAB
27 VALUES
28 (LOG_TAB_ID.NEXTVAL,
29 'PRODUCTINFO',
30 'DELETE',
31 :OLD.PRODUCTID,
32 SYSDATE);
33 DBMS_OUTPUT.PUT_LINE('删除数据完成，删除数据主键是 ' || :OLD.PRODUCTID);
```

```

34 END CASE;
35 END;
36 /

```

【代码解析】

- 第1行表示创建名为PRODUCTINFO_OPER_DML_TGR的触发器。
- 第2行表示在增加、修改、删除操作后触发。
- 第3行表示该触发器作用在PRODUCTINFO表上。
- 第4行表示行级触发。
- 第6行表示使用CASE语句区分各种操作。
- 第7~15行表示当增加数据时，向事件记录表中增加数据并输出到屏幕。其中LOG_TAB_ID.NEXTVAL表示利用序列作为新增数据的主键，:NEW.PRODUCTID表示新增数据的产品ID。
- 第16~24行表示当修改数据时，向事件记录表中增加数据并输出到屏幕。:OLD.PRODUCTID表示修改前该字段的值。
- 第25~33行表示当删除数据时，向事件记录表中增加数据并输出到屏幕。:OLD.PRODUCTID表示删除前该字段的值。

【执行效果】

在SQL*Plus下执行脚本，如果成功，则提示如下：

触发器已创建

【验证触发器】

该触发器是行级触发器，当修改或删除多条记录时会触发多次，而且如果没有数据被修改，即使执行删除或修改操作都不会激发该触发器。在表PRODUCTINFO上分别执行增加、修改、删除操作，查看该触发器是否被正常触发。具体操作步骤如下：

1) 对PRODUCTINFO表执行DML操作。

为PRODUCTINFO表增加一条记录：

```
INSERT INTO PRODUCTINFO VALUES('0240090001','触发器测试',1000,0,0000000000,'','测试');
```

修改PRODUCTINFO表记录，把所有的DESPERATION字段值置成TEST字符串：

```
UPDATE PRODUCTINFO SET DESPERATION = 'TEST';
```

删除表PRODUCTINFO中的一条记录：

```
DELETE FROM PRODUCTINFO WHERE PRODUCTID = '0240090001';
```

以上3条语句执行完成后提示如图11.4所示。

2) 检查事件记录表的数据是否成功。执行如下查询SQL，查询结果如图11.5所示。

```
SELECT * FROM LOG_TAB;
```

对比图11.4和图11.5，可以发现图11.5中LOG_TAB的记录同图11.4中的操作是对应的，也就是说该触发器成功地激发了。该例利用CASE语句对3种不同触发方式分别进行操作，当然也可以利用IF语句实现同样的功能。其中在执行UPDATE操作的时候修改了多条记录，因为是行级触发，那么该触发器会触发多次，把所有修改的数据的主键都得到了，并把相关数据放进了事件记录表。



图11.4 DML操作表PRODUCTINFO提示



图11.5 LOG_TAB表记录

【示例5】在触发器中使用IF语句

该示例将使用IF语句进行条件判断，如果修改数据的日期是当月的25日，并且修改产品的价格高于3000，那么该修改操作将被终止。

这两个条件必须同时满足，否则数据可以正常修改，并把操作记录存入表LOG_TAB中。

具体脚本如下：

```

01 CREATE TRIGGER PRODUCTINFO_OPER_CHK_TGR
02 BEFORE UPDATE OF PRODUCTPRICE ON PRODUCTINFO
03 FOR EACH ROW
04 BEGIN
05     IF (TO_CHAR(SYSDATE, 'dd') = 25 AND :OLD.PRODUCTPRICE > 3000) THEN
06         RAISE_APPLICATION_ERROR(-20000,
07             '今天是25号，不允许修改价格高于3000的数据！');
08     END IF;
09     INSERT INTO LOG_TAB
10     VALUES
11     (LOG_TAB_ID.NEXTVAL,
12      'PRODUCTINFO',
  
```

注意



```

13      'INSERT',
14      :NEW.PRODUCTID,
15      SYSDATE);
16      DBMS_OUTPUT.PUT_LINE('修改数据完成, 主键是 ' || :NEW.PRODUCTID);
17
18 END;
19 /

```

【代码解析】

- 第1行表示创建名为PRODUCTINFO_OPER_CHK_TGR的触发器。
- 第2行表示该触发器为前触发，作用在PRODUCTINFO表的PRODUCTPRICE字段，当修改该字段时触发器会被激发。
- 第3行表示行级触发。
- 第5行使用IF语句判断如果系统时间中包含25，也就是说当日是25号，并且修改前的价格大于3000，则进入IF语句内部。
- 第6行表示错误提示。RAISE_APPLICATION_ERROR可以把应用程序错误传递到客户端，它包含两个参数，第一个参数是错误代码，可以写-20 999--20 000之间的数值，第二个参数是错误提示。
- 第9~15行表示当不符合判断条件时能正常修改数据，并把操作记录增加到LOG_TAB表中。
- 第16行表示输出提示。

【执行效果】

在SQL*Plus下执行脚本，如果成功，则提示如下：

触发器已创建

【验证触发器】

1) 修改PRODUCTINFO表中主键为0240030002的数据，该记录价格为3600，执行如下的脚本：

```
UPDATE PRODUCTINFO SET PRODUCTPRICE= 2000 WHERE PRODUCTID='0240030002';
```

如果当日为25号，则会出现如图11.6所示的效果。

2) 修改PRODUCTINFO表中主键为0240050001的数据，该记录价格为400，这种情况下即使日期为当月的25号，也会正常通过。效果如图11.7所示。



图11.6 修改价格时中断操作



图11.7 数据修改正常通过

利用IF条件可以方便地终止不符合业务逻辑的数据操作，当出现自定义的异常时，SQL语句会在执行操作前就终止。

【示例6】使用WHEN限制条件的触发器

该触发器完成的功能是当在表PRODUCTINFO中增加的数据是“雨具”类型时，需要把当前的价格改为打九折，需要利用WHEN子句完成数据验证。具体触发器脚本如下：

```
01 CREATE TRIGGER PRODUCTINFO_WHEN_OPER_CHK_TGR
02 BEFORE INSERT ON PRODUCTINFO
03 FOR EACH ROW
04 WHEN (NEW.CATEGORY = '0100050001')
05 BEGIN
06 DBMS_OUTPUT.PUT_LINE('原价格: ' || :NEW.PRODUCTPRICE);
07 :NEW.PRODUCTPRICE := :NEW.PRODUCTPRICE * 0.9;
08 DBMS_OUTPUT.PUT_LINE('打折后价格: ' || :NEW.PRODUCTPRICE);
09 END;
10 /
```

【代码解析】

- 第1行表示创建名为PRODUCTINFO_WHEN_OPER_CHK_TGR的触发器。
- 第2行表示该触发器作用在表PRODUCTINFO上，触发事件是增加数据，触发方式为前触发。
- 第3行表示该触发器为行级触发器。
- 第4行表示限制条件，脚本字面解释为触发器激发前增加的数据中CATEGORY字段为0100050001的记录，也就是说增加的数据是“雨具”类型时将激发该触发器。
- 第7行表示把增加数据的价格打九折，然后再存入表中。

【执行效果】

在SQL*Plus下执行脚本，如果成功，则提示如下：

触发器已创建

【验证触发器】

对表PRODUCTINFO执行INSERT操作，增加两条记录。具体脚本如下：

```
01 INSERT INTO PRODUCTINFO VALUES('0240090001','触发器测试',1000,0,'0100020001','','测试1');
02 INSERT INTO PRODUCTINFO VALUES('0240090002','触发器测试',1000,0,'0100050001','','测试2');
```

其中第1行脚本表示增加一条类型为“路由器”的记录，第2行表示增加一条类型为“雨具”的记录。这两条记录的增加过程以及最终结果见图11.8。

PRODUCTID	PRODUCTNAME	PRODUCTPRICE	AMOUNTIN STOCK	DISCOUNT	UNIT
01000001	路由器	1000	100	0.00000000	个
01000002	路由器	1000	100	0.00000000	个
01000003	路由器	1000	100	0.00000000	个
01000004	路由器	1000	100	0.00000000	个
01000005	路由器	1000	100	0.00000000	个
01000006	路由器	1000	100	0.00000000	个
01000007	路由器	1000	100	0.00000000	个
01000008	路由器	1000	100	0.00000000	个
01000009	路由器	1000	100	0.00000000	个
01000010	路由器	1000	100	0.00000000	个
01000011	路由器	1000	100	0.00000000	个
01000012	路由器	1000	100	0.00000000	个
01000013	路由器	1000	100	0.00000000	个
01000014	路由器	1000	100	0.00000000	个
01000015	路由器	1000	100	0.00000000	个
01000016	路由器	1000	100	0.00000000	个
01000017	路由器	1000	100	0.00000000	个
01000018	路由器	1000	100	0.00000000	个
01000019	路由器	1000	100	0.00000000	个
01000020	路由器	1000	100	0.00000000	个
01000021	路由器	1000	100	0.00000000	个
01000022	路由器	1000	100	0.00000000	个
01000023	路由器	1000	100	0.00000000	个
01000024	路由器	1000	100	0.00000000	个
01000025	路由器	1000	100	0.00000000	个
01000026	路由器	1000	100	0.00000000	个
01000027	路由器	1000	100	0.00000000	个
01000028	路由器	1000	100	0.00000000	个
01000029	路由器	1000	100	0.00000000	个
01000030	路由器	1000	100	0.00000000	个
01000031	路由器	1000	100	0.00000000	个
01000032	路由器	1000	100	0.00000000	个
01000033	路由器	1000	100	0.00000000	个
01000034	路由器	1000	100	0.00000000	个
01000035	路由器	1000	100	0.00000000	个
01000036	路由器	1000	100	0.00000000	个
01000037	路由器	1000	100	0.00000000	个
01000038	路由器	1000	100	0.00000000	个
01000039	路由器	1000	100	0.00000000	个
01000040	路由器	1000	100	0.00000000	个
01000041	路由器	1000	100	0.00000000	个
01000042	路由器	1000	100	0.00000000	个
01000043	路由器	1000	100	0.00000000	个
01000044	路由器	1000	100	0.00000000	个
01000045	路由器	1000	100	0.00000000	个
01000046	路由器	1000	100	0.00000000	个
01000047	路由器	1000	100	0.00000000	个
01000048	路由器	1000	100	0.00000000	个
01000049	路由器	1000	100	0.00000000	个
01000050	路由器	1000	100	0.00000000	个
01000051	路由器	1000	100	0.00000000	个
01000052	路由器	1000	100	0.00000000	个
01000053	路由器	1000	100	0.00000000	个
01000054	路由器	1000	100	0.00000000	个
01000055	路由器	1000	100	0.00000000	个
01000056	路由器	1000	100	0.00000000	个
01000057	路由器	1000	100	0.00000000	个
01000058	路由器	1000	100	0.00000000	个
01000059	路由器	1000	100	0.00000000	个
01000060	路由器	1000	100	0.00000000	个
01000061	路由器	1000	100	0.00000000	个
01000062	路由器	1000	100	0.00000000	个
01000063	路由器	1000	100	0.00000000	个
01000064	路由器	1000	100	0.00000000	个
01000065	路由器	1000	100	0.00000000	个
01000066	路由器	1000	100	0.00000000	个
01000067	路由器	1000	100	0.00000000	个
01000068	路由器	1000	100	0.00000000	个
01000069	路由器	1000	100	0.00000000	个
01000070	路由器	1000	100	0.00000000	个
01000071	路由器	1000	100	0.00000000	个
01000072	路由器	1000	100	0.00000000	个
01000073	路由器	1000	100	0.00000000	个
01000074	路由器	1000	100	0.00000000	个
01000075	路由器	1000	100	0.00000000	个
01000076	路由器	1000	100	0.00000000	个
01000077	路由器	1000	100	0.00000000	个
01000078	路由器	1000	100	0.00000000	个
01000079	路由器	1000	100	0.00000000	个
01000080	路由器	1000	100	0.00000000	个
01000081	路由器	1000	100	0.00000000	个
01000082	路由器	1000	100	0.00000000	个
01000083	路由器	1000	100	0.00000000	个
01000084	路由器	1000	100	0.00000000	个
01000085	路由器	1000	100	0.00000000	个
01000086	路由器	1000	100	0.00000000	个
01000087	路由器	1000	100	0.00000000	个
01000088	路由器	1000	100	0.00000000	个
01000089	路由器	1000	100	0.00000000	个
01000090	路由器	1000	100	0.00000000	个
01000091	路由器	1000	100	0.00000000	个
01000092	路由器	1000	100	0.00000000	个
01000093	路由器	1000	100	0.00000000	个
01000094	路由器	1000	100	0.00000000	个
01000095	路由器	1000	100	0.00000000	个
01000096	路由器	1000	100	0.00000000	个
01000097	路由器	1000	100	0.00000000	个
01000098	路由器	1000	100	0.00000000	个
01000099	路由器	1000	100	0.00000000	个
01000100	路由器	1000	100	0.00000000	个
01000101	路由器	1000	100	0.00000000	个
01000102	路由器	1000	100	0.00000000	个
01000103	路由器	1000	100	0.00000000	个
01000104	路由器	1000	100	0.00000000	个
01000105	路由器	1000	100	0.00000000	个
01000106	路由器	1000	100	0.00000000	个
01000107	路由器	1000	100	0.00000000	个
01000108	路由器	1000	100	0.00000000	个
01000109	路由器	1000	100	0.00000000	个
01000110	路由器	1000	100	0.00000000	个
01000111	路由器	1000	100	0.00000000	个
01000112	路由器	1000	100	0.00000000	个
01000113	路由器	1000	100	0.00000000	个
01000114	路由器	1000	100	0.00000000	个
01000115	路由器	1000	100	0.00000000	个
01000116	路由器	1000	100	0.00000000	个
01000117	路由器	1000	100	0.00000000	个
01000118	路由器	1000	100	0.00000000	个
01000119	路由器	1000	100	0.00000000	个
01000120	路由器	1000	100	0.00000000	个
01000121	路由器	1000	100	0.00000000	个
01000122	路由器	1000	100	0.00000000	个
01000123	路由器	1000	100	0.00000000	个
01000124	路由器	1000	100	0.00000000	个
01000125	路由器	1000	100	0.00000000	个
01000126	路由器	1000	100	0.00000000	个
01000127	路由器	1000	100	0.00000000	个
01000128	路由器	1000	100	0.00000000	个
01000129	路由器	1000	100	0.00000000	个
01000130	路由器	1000	100	0.00000000	个
01000131	路由器	1000	100	0.00000000	个
01000132	路由器	1000	100	0.00000000	个
01000133	路由器	1000	100	0.00000000	个
01000134	路由器	1000	100	0.00000000	个
01000135	路由器	1000	100	0.00000000	个
01000136	路由器	1000	100	0.00000000	个
01000137	路由器	1000	100	0.00000000	个
01000138	路由器	1000	100	0.00000000	个
01000139	路由器	1000	100	0.00000000	个
01000140	路由器	1000	100	0.00000000	个
01000141	路由器	1000	100	0.00000000	个
01000142	路由器	1000	100	0.00000000	个
01000143	路由器	1000	100	0.00000000	个
01000144	路由器	1000	100	0.00000000	个
01000145	路由器	1000	100	0.00000000	个
01000146	路由器	1000	100	0.00000000	个
01000147	路由器	1000	100	0.00000000	个
01000148	路由器	1000	100	0.00000000	个
01000149	路由器	1000	100	0.00000000	个
01000150	路由器	1000	100	0.00000000	个
01000151	路由器	1000	100	0.00000000	个
01000152	路由器	1000	100	0.00000000	个
01000153	路由器	1000	100	0.00000000	个
01000154	路由器	1000	100	0.00000000	个
01000155	路由器	1000	100	0.00000000	个
01000156	路由器	1000	100	0.00000000	个
01000157	路由器	1000	100	0.00000000	个
01000158	路由器	1000	100	0.00000000	个
01000159	路由器	1000	100	0.00000000	个
01000160	路由器	1000	100	0.00000000	个
010001					



从图11.8中可以发现第一条记录的价格没有变，依然是1000，而第二条记录的价格则变成了900，这说明该触发器成功激发并且WHEN选项限制条件准确。

利用WHEN限制条件可以比较精确地限制触发器的激发条件，每当操作特定数据时可以考虑使用该条件限制，使得触发器更加灵活。需要说明的是，在WHEN条件中的:NEW或:OLD使用方式和通常使用方式上稍微有点差别，这里直接使用NEW或OLD关键词得到数据，没有前面的冒号。

如果想修改:NEW引用的数据（类似脚本第7行的操作），那么该触发器应为前触发的方式。因为后触发方式中:NEW数据是不允许被修改的，只能被调用，即使能修改也没有任何意义。

为了保证数据的完整性，在某种情况下开发人员会利用触发器实现级联操作功能，如级联修改或级联删除等。

【示例7】实现级联修改的触发器

该示例将提供级联修改产品类型编码的功能，并将原来的编码存放到CATEGROYINFO_BAK（表结构参考CATEGROYINFO）表中。具体来说，就是当表CATEGROYINFO的编码发生变化时，PRODUCTINFO表中CATEGORY字段与之对应的编码同样要变化。分为如下两个步骤实现：

1) 创建备份产品类型编码表CATEGROYINFO_BAK。脚本如下：

```
CREATE TABLE CATEGROYINFO_BAK AS SELECT * FROM CATEGROYINFO WHERE 1<>1
```

该语句表示创建表CATEGROYINFO_BAK，其表结构复制CATEGROYINFO的表结构，但不包括复制数据。如果不包含WHERE后面的条件则将完全复制CATEGROYINFO表，包括里面的数据。

2) 创建触发器。具体脚本如下：

```
01 CREATE TRIGGER MUTLI_OPER_CHK_TGR
02 AFTER UPDATE OF CATEGROYID ON CATEGROYINFO
03 FOR EACH ROW
04 BEGIN
05     UPDATE PRODUCTINFO
06     SET CATEGORY = :NEW.CATEGROYID
07     WHERE CATEGORY = :OLD.CATEGROYID;
08     INSERT INTO CATEGROYINFO_BAK VALUES (:OLD.CATEGROYID, :OLD.CATEGROYNAME);
09
10     DBMS_OUTPUT.PUT_LINE('数据已存入 CATEGROYINFO_BAK表中');
11 END;
12 /
```

【代码解析】

- ☐ 第1行表示创建名为MUTLI_OPER_CHK_TGR的触发器。
- ☐ 第2行表示该触发器在修改CATEGROYINFO表的CATEGROYID字段后触发。
- ☐ 第3行表示该触发器为行触发。
- ☐ 第5~7行表示修改PRODUCTINFO中CATEGORY字段的数据，修改后的数据同

CATEGROYINFO表数据一致。其中:NEW.CATEGROYID表示CATEGROYINFO表修改后的数据;OLD.CATEGROYID为修改前的数据。

□ 第8行表示备份修改前的数据到表CATEGROYINFO_BAK中。

【执行效果】

在SQL*Plus下执行脚本,如果成功,则提示如下:

触发器已创建

【验证触发器】

验证触发器也分为两个步骤:

1) 修改CATEGROYINFO表的CATEGROYID字段。SQL语句如下:

```
UPDATE CATEGROYINFO SET CATEGROYID = '0100010000'
WHERE CATEGROYID = '0100010001'
```

2) 查询PRODUCTINFO、CATEGROYINFO以及CATEGROYINFO_BAK,确认数据修改正确。查询结果见图11.9。

```
SQL> SELECT * FROM CATEGROYINFO WHERE CATEGROYID = '0100010000';
```

CATEGROYID	CATEGROYNAME
0100010000	01

```
SQL> SELECT * FROM PRODUCTINFO WHERE CATEGROYID = '0100010000';
```

PRODUCTID	PRODUCTNAME	PRODUCTPRICE	QUANTITY	CATEGROYID	DELETION	ORIGIN
0100010000	010001	50	50	0100010000	01	01

```
SQL> SELECT * FROM CATEGROYINFO_BAK;
```

CATEGROYID	CATEGROYNAME
0100010001	01

```
SQL>
```

图11.9 触发器数据验证

在很多情况下由于数据库设计的原因,两个表中字段很可能出现级联的关系,这种情况最常见的就在字典表和业务表中。由于业务表使用了字典数据,所以如果字典表发生变化会造成数据的不一致性,因此需要在字典表数据发生变化的同时修改其他表中的该数据。

以上介绍的都是行级触发器,与其对应的是语句级触发器。语句级触发器适合用在整张表的操作控制上,因为它只对语句有效,也就是当执行语句时触发一次,与修改多少条记录没有关系。

【示例8】语句级触发器

该示例将通过语句级触发器控制每个月的1号晚上23点到0点不允许操作表中数据。具体脚本如下:

```
01 CREATE TRIGGER PRODUCTINFO_STMT_CHK_TGR
02 BEFORE INSERT OR UPDATE OR DELETE ON PRODUCTINFO
03 BEGIN
04 IF (TO_CHAR(SYSDATE, 'dd') = 1 AND TO_CHAR(SYSDATE, 'HH24') = '23') THEN
05 RAISE_APPLICATION_ERROR(-20000, '当前时段不允许修改数据');
06 END IF;
07 END;
```



【代码解析】

- 第1行表示创建名为PRODUCTINFO_STMT_CHK_TGR的触发器。
- 第2行表示该触发器在增加、修改、删除前触发。
- 第4行表示判断每月的1号23:00~0:00不允许修改数据。
- 第5行表示把应用程序的错误提到客户端。

【执行效果】

在SQL*Plus下执行脚本，如果成功，则提示如下：

触发器已创建

【验证触发器】

验证该触发器可以向PRODUCTINFO表增加或修改、删除数据，这里以增加做测试语句。SQL语句如下：

```
INSERT INTO PRODUCTINFO VALUES('0240090001','触发器测试',1000,0,0000000000,NULL,'测试');
```

执行效果如图11.10所示。

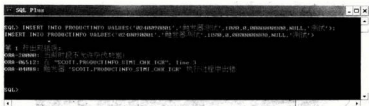


图11.10 验证语句级触发器

利用语句级触发器可以达到在某个条件范围内用户对表数据操作进行限制的目的。

11.2.4 触发器执行顺序

在同一对象上可以作用多个触发器，因此触发器被激发的顺序是有先后关系的。其触发顺序如下：

- 1) 首先被触发的将是前语句级触发器 (before statement trigger)，该触发器会被执行一次。
- 2) 如果有行级的触发器则接下来执行前行级触发器 (before row trigger)，行级触发器执行的次数同SQL修改的记录次数一致。
- 3) 当SQL修改记录完成后会触发行级触发器，这时候的行级触发器为后行级触发器 (after row trigger)，该类型触发的次数同SQL修改记录的次数一致。
- 4) 执行一次语句级的触发器，此时的语句级触发器为后语句级触发器 (after statement trigger)。

如果多个相同类型的、相同事件触发器作用在同一个对象上，如果在Oracle 11g之前，那么最终被执行的会有一定的随机性，而在Oracle 11g中利用FOLLOWS可以控制其顺序。

【示例9】控制触发器的顺序

触发器执行顺序分为以下两个步骤：

- 1) 创建前面介绍过的PRODUCTINFO_OPER_TGR触发器。该触发器为行级触发器，激



发时机为前触发，作用在PRODUCTINFO表上。

2) 创建触发器PRODUCTINFO_OPER_ORD_TGR，该触发器利用FOLLOWS控制其在PRODUCTINFO_OPER_TGR后触发，否则默认在PRODUCTINFO_OPER_TGR前触发。具体脚本如下：

```
01 CREATE TRIGGER PRODUCTINFO_OPER_ORD_TGR
02 BEFORE INSERT
03 ON PRODUCTINFO
04 FOR EACH ROW
05 FOLLOWS PRODUCTINFO_OPER_TGR
06 BEGIN
07 DBMS_OUTPUT.PUT_LINE('触发器顺序测试');
08 END;
```

【代码解析】

- 第1行表示创建触发器，名为PRODUCTINFO_OPER_ORD_TGR。
- 第2行表示该触发器为前触发，触发事件是INSERT。
- 第3行表示该触发器作用在表PRODUCTINFO上。
- 第4行表示为行级触发器。
- 第5行表示该触发器在PRODUCTINFO_OPER_TGR之后被激发。

【执行效果】

在SQL*Plus下执行脚本，如果成功，则提示如下：

触发器已创建

【验证触发器】

创建以上触发器，执行增加数据SQL语句。执行效果见图11.11。



图11.11 使用顺序控制

此时可以看出PRODUCTINFO_OPER_ORD_TGR触发器在PRODUCTINFO_OPER_TGR之后被激发，这是使用FOLLOWS语句的结果（见第5行）。如果不加该语句，那么触发器被激发的顺序正好相反，效果这里不再给出，感兴趣的读者可以自行实验。

11.2.5 复合类型触发器

复合类型的触发器是Oracle 11g的新特性，属于触发器的增强部分。复合类型的触发器相当于在一个触发器中包含了4种不同类型的触发器，分别是语句之前（before statement）、行之前（before row）、行之后（after row）、语句之后（after statement）。这么做可以很轻松地把变量在各状态之间传递，而在该类型触发器出现之前，变量在触发器之间的传递比较麻烦。

利用该类型的触发器还可以方便地解决ORA-04091错误，这里涉及一个变异表的概念，读者可以理解变异表是正在被DML操作修改的表，也是触发器的作用表。而触发器通常不能对变异表进行操作，下面一个示例将利用复合类型的触发器，解决ORA-04091错误。

【示例10】复合型触发器

该触发器试图实现功能：当为PRODUCTINFO表中的数据提高价格时，触发器判断新旧价格差是否高于价格在2000以下的所有产品价格的平均值的20%，如果高于此值，则提示数据有问题。相关步骤如下：

1) 创建普通类型的触发器，并激发，查看是否存在ORA-04091错误。具体脚本如下：

```
01 CREATE TRIGGER PRIC_TGR
02 AFTER UPDATE ON PRODUCTINFO
03 FOR EACH ROW
04 DECLARE
05   V_PRO_AVG NUMBER(10, 2) := 0.0;
06 BEGIN
07   SELECT AVG(PRODUCTPRICE)
08   INTO V_PRO_AVG
09   FROM PRODUCTINFO
10  WHERE PRODUCTINFO.PRODUCTPRICE < 2000;
11
12  IF :NEW.PRODUCTPRICE - :OLD.PRODUCTPRICE > V_PRO_AVG * 0.20 THEN
13    RAISE_APPLICATION_ERROR(-20001, '数据修改错误!');
14  END IF;
15 END;
```

【代码解析】

□ 第1~3行表示创建作用于表PRODUCTINFO上的修改后的行级触发器，名为PRIC_TGR。

□ 第7~10行表示求取价格在2000以下的产品的平均价格，并存入变量V_PRO_AVG中，这种SELECT INTO形式的语句会把触发表作为变异表。

□ 第12~14行判断新增价格如果高于平均价格的20%，则提示出错。

【验证触发器】

该触发器创建完成后，执行如下语句：

```
UPDATE PRODUCTINFO
SET PRODUCTPRICE = PRODUCTPRICE + 300
WHERE PRODUCTPRICE > 2000;
```

执行效果见图11.12。



图11.12 修改表出现ORA-04091错误



2) 创建复合类型的触发器, 解决上一步出现的错误。具体脚本如下:

```
01 CREATE OR REPLACE TRIGGER COMPOUND_TGR
02 FOR UPDATE ON PRODUCTINFO COMPOUND TRIGGER
03
04 V_PRO_AVG NUMBER(10, 2) := 0.0;
05
06 BEFORE STATEMENT IS
07 BEGIN
08     SELECT AVG(PRODUCTPRICE)
09     INTO V_PRO_AVG
10     FROM PRODUCTINFO
11     WHERE PRODUCTINFO.PRODUCTPRICE < 2000;
12 END BEFORE STATEMENT;
13
14 AFTER EACH ROW IS
15 BEGIN
16     IF :NEW.PRODUCTPRICE - :OLD.PRODUCTPRICE > V_PRO_AVG * 0.20 THEN
17         RAISE_APPLICATION_ERROR(-20011, '数据修改错误!');
18     END IF;
19 END AFTER EACH ROW;
20 END;
21 /
```

【代码解析】

- ☐ 第1~2行表示创建复合类型的触发器, 触发事件为UPDATE操作。
- ☐ 第4行表示声明变量, 存放平均价格。
- ☐ 第6~12行表示前语句触发的脚本。此时的操作相当于前语句触发器的操作。
- ☐ 第14~19行表示后行级的触发。也就是说, 它的执行次数和修改的记录数一致。

【验证触发器】

该触发器创建完成后, 执行如下语句:

```
UPDATE PRODUCTINFO
SET PRODUCTPRICE = PRODUCTPRICE + 300
WHERE PRODUCTPRICE > 2000;
```

执行效果见图11.13。



图11.13 复合触发器验证结果

从图11.13中可以看出该触发器已经正常激发，避免了普通触发器出现的ORA-04091错误。以后读者就可以不用费神地解决此类错误了。

11.2.6 INSTEAD OF类型触发器

在该类型的触发器作用下，如果对作用对象执行DML操作，那么该操作会被触发器的内部操作所取代。触发器作用在视图当中，用于解决视图不可更新的问题。至于什么样的视图不可更新，读者可以参考第9章。

【示例11】 INSTEAD OF触发器的使用

该示例将演示利用INSTEAD OF触发器解决视图的不可更新问题。实现步骤如下：

1) 创建视图。具体脚本如下：

```
CREATE VIEW PRODUCTINFO_VIEW AS
SELECT DISTINCT PRODUCTNAME, PRODUCTPRICE, QUANTITY, CATEGORY, ORIGIN
FROM PRODUCTINFO;
```

【执行效果】

在SQL*Plus下执行脚本，如果成功，则提示如下：

视图已创建

2) 创建触发器。具体脚本如下：

```
01 CREATE TRIGGER INSTEAD_OF_TGR --触发器
02 INSTEAD OF INSERT ON PRODUCTINFO_VIEW
03 DECLARE
04 CATEGID VARCHAR2(10); --产品类型编码
05 BEGIN
06 SELECT CATEGROYID
07 INTO CATEGID
08 FROM CATEGROYINFO
09 WHERE CATEGROYINFO.CATEGROYNAME = :NEW.CATEGORY;
10 DBMS_OUTPUT.PUT_LINE('-----' || CATEGID);
11 INSERT INTO PRODUCTINFO
12 VALUES
13 ('0240090001',
14 :NEW.PRODUCTNAME,
15 :NEW.PRODUCTPRICE,
16 :NEW.QUANTITY,
17 CATEGID,
18 '测试',
19 :NEW.ORIGIN);
20 END;
21 /
```

【代码解析】

- 第1~2行表示创建替代类型的触发器作用在视图PRODUCTINFO_VIEW上，触发事件为增加事件，触发器名称为INSTEAD_OF_TGR。
- 第4行表示声明变量，用做存储产品类型编码。
- 第6~9行表示根据输入的产品类型名称查询产品类型编码，并把该值存入CATEGID中。

这步操作有可能引发异常，前面章节都已有相关介绍，为了方便这里不再给出。

- 第11~19行表示向表PRODUCTINFO中增加数据，该操作是这个触发器的主要操作。

这部分的SQL语句替换了针对视图的增加操作，为了解决视图不可更新的目的。其中的14、15、16、19行使用了对执行视图DML操作语句中的值。



【执行效果】

在SQL*Plus下执行脚本，如果成功，则提示如下：

触发器已创建

【验证触发器】

以上步骤操作完成，可以对视图执行增加操作。SQL脚本如下：

```
INSERT INTO PRODUCTINFO_VIEW VALUES('触发器测试',1000,0,'雨具','中国');
```

该脚本执行成功后，对PRODUCTINFO表进行查询，查看该触发器是否成功把数据添加到里面。增加数据以及查询结果如图11.14所示。

PRODUCTID	PRODUCTNAME	PRODUCTPRICE	QUANTITY	CATEGORY	DESCRIPTION	ORIGIN
001000000001	雨伞L/D-46CM	7000	100	01.0000.000001		日本
001000000002	雨伞45cm VIBR	1100	20	01.0000.000002		中国
001000000003	雨衣 45X55-REG	1600	10	01.0000.000003		中国
001000000004	雨衣45-54.6 REG	2500	110	01.0000.000004		中国
001000000005	雨衣41cm	600	100	01.0000.000005		中国
001000000006	雨衣40cm2件装	1800	20	01.0000.000006		中国
001000000007	雨衣40cm	1800	0	01.0000.000007		中国
001000000008	雨衣40	50	50	01.0000.000008		中国

图11.14 替代触发器验证过程

从图11.14中标示部分可以看出已经把数据插入到表PRODUCTINFO中。其中CATEGORY字段值对应CATEGORYID表中的雨衣编码。需要注意的是，该类型的触发器只能做行级的触发。

11.2.7 DDL类型触发器

所谓DDL类型触发器，就是因DDL操作而激发的触发器，主要包括CREATE、ALTER、DROP等事件，更多的操作可以查看11.1.4小节语法部分。

利用DDL类型的触发器可以限制和记录特定的DDL操作。例如，通过DDL触发器可以限制对数据库结构的修改，记录数据库中的更改事件，也可以在修改对象的时候根据实际情况做出其他相应动作。下面通过一个示例演示如何创建和使用该类型的触发器。

【示例12】使用DDL类型触发器

该触发器将提示CREATE创建操作，如果有人执行TEST表的删除操作，将提示错误，也不允许对表使用ALTER操作和RENAME操作。具体脚本如下：

```
01 CREATE TRIGGER DDL_TGR
02 BEFORE CREATE OR ALTER OR DROP OR RENAME ON SCHEMA
03 BEGIN
```

```

04 IF SYSEVENT = 'CREATE' THEN
05     DBMS_OUTPUT.PUT_LINE(DICTIONARY_OBJ_NAME || '创建中...');
06 ELSIF SYSEVENT = 'DROP' THEN
07     IF DICTIONARY_OBJ_NAME = 'TEST' THEN
08         RAISE_APPLICATION_ERROR(-20000, '不允许删除TEST表!');
09     END IF;
10 ELSIF SYSEVENT = 'ALTER' THEN
11     RAISE_APPLICATION_ERROR(-20000, '不允许修改表!');
12 ELSIF SYSEVENT = 'RENAME' THEN
13     RAISE_APPLICATION_ERROR(-20000, '不允许修改表名称!');
14 END IF;
15 END;
16 /

```

【代码解析】

- 第1~2行表示在模式上创建名为DDL_TGR的触发器，触发器事件为CREATE、ALTER、DROP、RENAME。
- 第4~5行表示利用SYSEVENT事件属性得到当前的DDL操作并进行判断。如果为CREATE事件，则输出信息。
- 第7行表示利用DICTIONARY_OBJ_NAME事件属性得到并判断删除的表名是否是TEST，如果条件为TRUE，则提示出错。
- 第10~14行利用同样的方式进行判断。

【执行效果】

在SQL*Plus下执行脚本，如果成功，则提示如下：

触发器已创建

【验证触发器】

以上步骤操作完成，对表TEST执行重命名操作。SQL脚本如下：

```
RENAME TEST TO TEST1;
```

执行效果如图11.15所示。



图11.15 DDL触发器验证

示例中只用了一种验证方式，感兴趣的读者可以试着使用其他3种方式验证。这里面使用了事件属性，它们是SYS拥有的独立函数。利用事件属性可以获取触发事件的信息。下面列出常用的几种事件属性，如表11.4所示。

表11.4 常用事件属性

属性函数	可用的事件	简介
SYSEVENT	所有事件	返回触发器的事件名称
INSTANCE_NUM	所有事件	返回当前数据库的示例号
DATABASE_NAME	所有事件	返回当前数据库名
SERVER_ERROR	SERVERERROR	错误堆栈的指定位置返回错误号
LOGIN_USER	所有事件	返回触发器的用户名
DICTIONARY_OBJ_TYPE	CREATE、ALTER、DROP	返回激活触发器的DDL操作的对象类型
DICTIONARY_OBJ_NAME	CREATE、ALTER、DROP	返回激活触发器的DDL操作的对象名称

11.2.8 用户和系统事件触发器

所谓系统事件触发器，就是基于Oracle系统事件而建立的触发器。该类型的触发器可以审计数据库的登录、注销以及关闭和启动等，更多的系统事件可以参见表11.2。下面以示例13来说明如何创建使用此类触发器。

【示例13】创建数据库级触发器

该示例将记录每个登录用户的时间，并把登录时间存放到用户登录记录表中。具体步骤如下：

- 1) 创建用户登录日志表。表结构如表11.5所示。

表11.5 LOG_USER表结构

字段名	中文释义	数据类型
LOGONID	记录ID，主键	varchar2(10)
LOGONNAME	登录用户名	varchar2(50)
LOGONTIME	登录时间	date

执行如下SQL脚本，创建表LOG_USER：

```
CREATE TABLE LOG_USER (
    LOGONID VARCHAR2(50),
    LOGONNAME VARCHAR2(50),
    LOGONTIME DATE,
    CONSTRAINT LOG_USER_PRK PRIMARY KEY (LOGONID)
);
```

在SQL*Plus下执行以上脚本，即创建了表LOG_USER。

- 2) 创建触发器。该触发器是数据库级，记录每个用户的登录时间。具体脚本如下：

```
01 CREATE TRIGGER LOGON_TGR
02 AFTER
03 LOGON
04 ON DATABASE
05 BEGIN
06 INSERT INTO LOG_USER
07 VALUES(LOG_TAB_ID.NEXTVAL, SYS.LOGIN_USER, SYSDATE);
08 END;
09 /
```



【代码解析】

- ☐ 第1~3行表示创建触发器，在登录之后触发。
- ☐ 第4行表示作用在数据库上，所有该数据库的用户登录动作都将被记录下来。用SCHEMA表示作用在当前模式上，只记录该模式的用户。
- ☐ 第6~7行表示把数据添加到LOG_USER表中。其中LOG_TAB_ID是前面创建的序列，用它做自增长的主键，SYS.LOGIN_USER就是11.2.7小节介绍的事件属性，用于得到登录用户。

【验证触发器】

执行脚本成功后验证该触发器。打开SQL*Plus，登录到数据库，查询LOG_USER表数据，查看登录情况是否被记录。读者可以根据自己的情况自行实验，这里不再给出具体结果。

11.3 使用PL/SQL工具操作触发器

有了前面几章的学习，读者对PL/SQL Developer的使用应该已经比较熟悉了。在该工具中创建触发器同创建存储过程和创建视图有一定的相似性。本节将介绍如何在该工具下操作触发器。

11.3.1 在PL/SQL Developer中创建触发器

在该工具中依然为创建触发器提供了模板窗口，开发人员填写必要的选项，然后生成触发器的原始框架脚本，详细的操作还是要开发者自行手工填写。接下来使用示例14介绍的创建触发器的步骤。

【示例14】使用PL/SQL Developer创建触发器

使用PL/SQL Developer创建触发器分为如下7个步骤：

- 1) 启动PL/SQL Developer工具，在整个布局的左上角单击【File】，从弹出的下拉列表中选择【New】然后进入到【Program Window】项，在列表中找到【Trigger】选项，如图11.16所示。



图11.16 PL/SQL Developer工具中的触发器选项

- 2) 单击该选项，弹出如图11.17所示的触发器模板窗口。

- ☐ Name: 该触发器的名称。
- ☐ Fires: 激发触发器的时机，包括before、after、instead of 3个选项。
- ☐ Event: 激发触发器的事件，包括insert、update、delete 3个选项。

- ☐ Table or view: 选择触发器作用的对象。
☐ Statement level: 是否语句级触发。不选该复选框则为行级触发。
 3) 填写窗口的相关信息。填写完成后如图11.18所示。



图11.17 触发器模板窗口



图11.18 触发器模板参数填写

- 4) 单击【OK】按钮。在Program窗口生成触发器的框架脚本，如图11.19所示。



图11.19 生成的框架脚本

窗口布局在前面的章节已经介绍过，这里不再赘述。不明白的读者可以参考存储过程的创建部分。

5) 创建并调试触发器。在生成的框架上补充脚本，格式化脚本后按F8键执行。这时会在窗口左边【Objects】透视图中的Triggers文件夹下生成名为DEV_TGR的触发器。不过此时的触发器有错误提示，具体效果见图11.20。

图11.20中箭头指向部分就是错误提示，根据错误提示分析脚本，可把错误定位在第8行，该行脚本没有用“;”结束，需要对其进行修改。

6) 完成修改后继续按F8键执行，如果此时触发器修改正确，则不会有错误提示，触发器名称上的红色错误提示也会消失。

7) 验证触发器。打开一个SQL窗口输入如下脚本并执行：

```

INSERT INTO PRODUCTINFO
VALUES ('0240090001', '触发器测试', 1000, 0, 0000000000, NULL, '测试');
  
```

如果触发器成功触发，则会在SQL窗口的Output标签页面中输出提示，如图11.21所示。

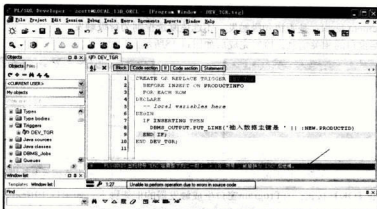


图11.20 创建并调试触发器



图11.21 激发触发器后的提示

11.3.2 设置触发器是否可用

触发器被创建后，会不断地被激发，如果业务上不需要使用该触发器了，则可以设置其是否可用属性，而不必把它删除。设置方式有两种。

第一种设置方式利用ENABLE | DISABLE关键词设置该触发器是否可用。设置语法如下：

```
ALTER TRIGGER [schema.]trigger DISABLE|ENABLE
```

第二种设置方式利用PL/SQL Developer实现该功能。操作步骤如下：

1) 选择触发器。在Triggers文件夹下找到需要修改的触发器，右击该触发器，如图11.22所示。

2) 设置是否可用。见图11.22的方框标示处。由于当前的触发器可用，所以在选项中有Disable项。单击它，此时会发现触发器图标变成灰色，表示该触发器不可用。

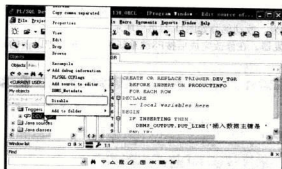


图11.22 选择需要设置的触发器

11.4 修改、删除触发器

创建好触发器后，如果该触发器已经没有用或者需要修改，那该怎么办呢？本节就介绍如何修改或删除已经存在的触发器。

11.4.1 修改触发器

触发器内容的修改同样要利用REPLACE关键词。根据11.1.4小节的语法，在SQL*Plus中创建触发器时带上OR REPLACE关键词，从而完成过程的修改，也就是覆盖。

在PL/SQL Developer中利用模板生成脚本时就已经自动带上了OR REPLACE关键词，如果想对已经存在的触发器进行修改，只需按示例15的方法操作即可完成。

【示例15】在PL/SQL Developer中修改触发器

在PL/SQL Developer中修改触发器分为如下3个步骤：

1) 选择编辑的触发器。在【Objects】下的Triggers文件夹中选中准备编辑的触发器，右击该触发器，在弹出的快捷菜单中选择【Edit】选项，如图11.23所示。

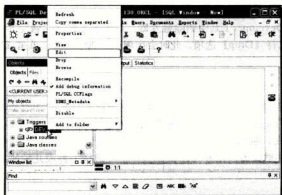


图11.23 选择需要设置的触发器



2) 修改触发器。进入编辑页面后可以对该触发器进行修改, 这里把由INSERT事件触发改成由UPDATE事件触发。改动部分如图11.24黑框所示。

3) 重新创建触发器。在修改完成后按F8键重新执行, 此时把以前的触发器覆盖掉, 也就相当于完成了修改。

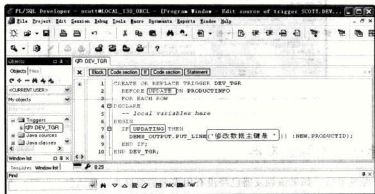


图11.24 修改触发器

11.4.2 删除触发器

对不用的触发器除了设置不可用属性外也可以直接将其删除。可以利用如下两种方式删除触发器。

1) 利用SQL语句删除触发器。

删除操作语法如下:

```
DROP TRIGGER [schema.]trigger
```

如果想删除DEV_TGR触发器, 可以执行如下脚本:

```
DROP TRIGGER DEV_TGR;
```

2) 在PL/SQL Developer工具中删除触发器。选中需要删除的触发器, 右击该触发器, 在弹出的快捷菜单中选择【Drop】选项, 此时会出现确认删除窗口, 单击【Yes】按钮即可。

11.5 小结

触发器是数据库的重要组成部分, 利用触发器可以完成更复杂的业务逻辑, 也可以对数据库的操作行为进行记录。本章介绍了触发器的概念和作用, 并详细地介绍了5种类型触发器的创建和使用方式, 包括DML类型触发器、DDL类型触发器、复合类型触发器、替代类型触发器、系统事件触发器。用户可以根据实际情况选择一种或几种使用, 来实现更复杂的功能。



读书笔记

11.6 习题

一、填空题

1. 触发器可以分为_____、_____、_____和_____5种类型。
2. 触发器使用_____关键词来控制顺序。
3. _____类型触发器是Oracle 11g新增的触发器类型。
4. NSTEAD OF 类型触发器作用在表上会出现_____错误。

二、实训题

利用复合型触发器，测试几种类型触发器的执行顺序。

第12章 事务和锁——确保数据安全

事务和锁是两个联系非常紧密的概念，它们保证了数据库的一致性。由于数据库是一个可以由多个用户共享的资源，因此当多个用户并发地存取数据时，就要保证数据的准确性。事务和锁就完成了这项功能。本章将介绍如下知识点：

- 事务的概念和使用
- 锁的概念和使用

12.1 什么是事务

事务在数据库中主要用于保证数据的一致性，防止出现错误数据。在事务内的语句都会被看成一个单元，一旦有一个失败，那么所有的都会失败。在编程过程中也会经常用到事务，本节将从基础开始，带领读者了解事务的概念和种类。

12.1.1 认识事务

事务就是一组包含一条或多条语句的逻辑单元，每个事务都是一个原子单位，在事务中的语句被作为一个整体，要么一起被提交，作用在数据库上，使数据库数据永久的修改；要么一起被撤销，对数据库不做任何的修改。

对于这个问题比较经典的例子就是银行账户之间的汇款转账操作。该操作在数据库中由以下3步完成：

- 1) 源账户减少存储金额，例如减少1000。
- 2) 目标账户增加存储金额，增加1000。
- 3) 在事务日志中记录该事务。

整个交易过程，我们看做一个事务，如果操作失败，那么该事务就会回滚，所有该事务中的操作将撤销，目标账户和源账户上的资金都不会出现变化；如果操作成功，那么将是对数据库永久的修改，即使以后服务器断电，也不会对该修改结果有影响。

事务在没有提交之前可以回滚，而且在提交前当前用户可以查看已经修改的数据，但其他用户查看不到该数据，一旦事务提交就不能再撤销修改了。Oracle的事务基本控制语句有如下几个：

- SET TRANSACTION：设置事务的属性。
- COMMIT：提交事务。
- SAVEPOINT：设置保存点。
- ROLLBACK：回滚事务。
- ROLLBACK TO SAVEPOINT：回滚至保存点。

注意 事务和程序不同，一条语句或者多条语句甚至一段程序都可能在一个事务中，而一段程序又可以包含多个事务。事务可以根据自己的需要把一段程序分成多个组，然后把每个组都当成一个单元，而这个单元就可以理解为一个事务。



12.1.2 事务的类型

事务分为如下两种类型。

(1) 显式方式

所谓的“显式方式”，就是利用命令完成。语法如下：

```
01 新事务开始
02 sql statement
03 ...
04 COMMIT|ROLLBACK;
```

【语法说明】

□ 第2~3行表示事物内的SQL语句，可以是单条，也可以是多条。

□ 第4行表示事务的提交或回滚。

Oracle中的事务不需要设置开始标志。通常有下列情况之一时，事务会开启：

□ 登录数据库后，第一次执行DML语句。

□ 当事务结束后，第一次执行DML语句。

(2) 隐式方式

该类型的事务没有明确的开始和结束标志。它由数据库自动开启，当一个程序正常结束或使用DDL语言时会自动提交，而操作失败时也会自动回滚。如果设置AUTOCOMMIT为打开状态（默认关闭），则每次执行DML操作都会自动提交。命令语法如下：

```
SET AUTOCOMMIT ON/OFF
```

事务在什么条件下结束读者需要注意，否则有丢失数据的可能。当有下列情况之一时，事务会结束：

□ 使用COMMIT事务提交，ROLLBACK事务回滚。

□ 执行DDL语句，事务自动提交。例如，使用CREATE、DROP、GRANT、REVOKE等命令。

□ 正常退出SQL*Plus时自动提交事务，非正常退出则ROLLBACK事务回滚。

事务可以保证数据的一致性。前面已经介绍过，事务没有提交时，当前会话所做的操作其他会话不会看到，下面用示例1来演示。

【示例1】使用事务保证数据的一致性

演示事务如何保证数据的一致性，这里分为如下4个步骤：

1) 登录SQL*Plus，我们称该窗口为SQL*Plus1。执行如下两条语句：

```
UPDATE PRODUCTINFO SET DESPERATION = '促销产品';
INSERT INTO PRODUCTINFO VALUES('0240090001','一致性测试',1000,0,0000000000,NULL,'测试');
```

当以上操作提示成功后，查询PRODUCTINFO表的内容，验证数据是否修改成功，结果见图12.1。

从图12.1中可以看出DML操作数据成功。注意，此时还没有提交事务的操作。

2) 以同样用户登录新的SQL*Plus，我们称该窗口为SQL*Plus2。同样查询表PRODUCTINFO的数据，查看数据修改情况。查询结果见图12.2。



SQL*Plus

SQL> SELECT * FROM PRODUCTINFO ;

PRODUCTID	PRODUCTNAME	PRODUCTPRICE	QUANTITY	CATEGORY	DESCRIPTION	ORIGIN
002-0001000001	联想LCD-86310000	70000	20	01-00001100001	联想产品	日本
002-0001000001	联想X4650-9100	11000	29	01-00001100002	联想产品	中国
002-0001000002	三星X4655-186A/XSC	36000	12	01-00001100002	联想产品	中国
002-0001000003	三星X4655-186A/XSC	25000	111	01-00001100002	联想产品	中国
002-0001000004	联想C4300	6000	129	01-00001100001	联想产品	中国
002-0001000005	联想I-800002路由器	20000	22	01-00002100001	联想产品	中国
002-0001000006	联想I-800002路由器	10000	0	0	0	中国
002-0001000007	联想I-800002路由器	50	50	01-00001100001	联想产品	中国

已选择9行。

图12.1 查询DML操作结果

SQL*Plus

SQL> SELECT * FROM PRODUCTINFO ;

PRODUCTID	PRODUCTNAME	PRODUCTPRICE	QUANTITY	CATEGORY	DESCRIPTION	ORIGIN
002-0001000001	联想LCD-86310000	70000	20	01-00001100001	联想产品	日本
002-0001000001	联想X4650-9100	11000	29	01-00001100002	联想产品	中国
002-0001000002	三星X4655-186A/XSC	36000	12	01-00001100002	联想产品	中国
002-0001000003	三星X4655-186A/XSC	25000	111	01-00001100002	联想产品	中国
002-0001000004	联想C4300	6000	129	01-00001100001	联想产品	中国
002-0001000005	联想I-800002路由器	20000	22	01-00002100001	联想产品	中国
002-0001000006	联想I-800002路由器	10000	0	0	0	中国
002-0001000007	联想I-800002路由器	50	50	01-00001100001	联想产品	中国

已选择9行。

图12.2 新会话查询表PRODUCTINFO数据

此时可以发现，当会话一的事务没有提交时，会话二不能查看到会话一修改的数据。

3) 在SQL*Plus1窗口提交事务。具体脚本如下：

COMMIT;

执行完成会有如下提示：

提交完成

4) 当事务提交完成后，SQL*Plus2窗口再次查询PRODUCTINFO表数据，结果见图12.3。

SQL*Plus

SQL> SELECT * FROM PRODUCTINFO ;

PRODUCTID	PRODUCTNAME	PRODUCTPRICE	QUANTITY	CATEGORY	DESCRIPTION	ORIGIN
002-0001000001	联想LCD-86310000	70000	20	01-00001100001	联想产品	日本
002-0001000001	联想X4650-9100	11000	29	01-00001100002	联想产品	中国
002-0001000002	三星X4655-186A/XSC	36000	12	01-00001100002	联想产品	中国
002-0001000003	三星X4655-186A/XSC	25000	111	01-00001100002	联想产品	中国
002-0001000004	联想C4300	6000	129	01-00001100001	联想产品	中国
002-0001000005	联想I-800002路由器	20000	22	01-00002100001	联想产品	中国
002-0001000006	联想I-800002路由器	10000	0	0	0	中国
002-0001000007	联想I-800002路由器	50	50	01-00001100001	联想产品	中国

已选择9行。

图12.3 提交事务后查询结果

此时可以发现事务一旦提交，SQL*Plus就能查询到修改的数据。由此可以看出，事务可以保证数据的一致性。

12.1.3 事务的保存点

在事务中可以根据自己的需要设置保存点。保存点可以设置在事务中的任何地方，也可以设置多个点，这样就可以把比较长的事务根据需要分成较小的段，这样做的好处是当对数据的操作出现问题时不用全部回滚，只需要回滚到保存点处即可。例如，在一个事务内前十段数据操作确认准确，而后面的操作没有办法确认，开发人员就可以在第十段操作结束后设置保存点，这样即便后面的操作有错误，开发人员也可以利用保存点回滚到第十段处，而不用从头写。

一旦把事务回滚到某个保存点后，Oracle将把保存点之后持有的锁释放掉，这时先前等待被锁资源的事务就可以继续了。而使事务回滚到保存点，有以下几点需要了解：

- 事务只回滚保存点之后的操作。
- 回滚到某保存点时，它以后的保存点将被删除，但保存点会被保留。
- 保存点之后的锁将被释放，但之前的会被保留。

保存点使用起来非常方便，只需一行脚本就能完成。下面利用示例2演示事务中如何使用保存点。

【示例2】在事务中使用保存点

在事务中使用保存点，该示例分为如下5个步骤：

- 1) 向PRODUCTINFO表增加一条数据，脚本如下，此时隐式事务已经打开。

```
INSERT INTO PRODUCTINFO VALUES('1','保存点测试1',1000,0,0000000000,NULL,'测试');
```

- 2) 执行如下脚本，用于创建保存点，名为FST。

```
SAVEPOINT FST;
```

如果保存点创建成功，则有以下提示：

保存点已创建

- 3) 当保存点创建完成后，继续向PRODUCTINFO表增加一条数据。脚本如下：

```
INSERT INTO PRODUCTINFO VALUES('2','保存点测试2',1000,0,0000000000,NULL,'测试');
```

- 4) 以上三步按顺序创建完成后，查看PRODUCTINFO表数据，如图12.4所示。

PRODUCTID	PRODUCTNAME	PRODUCTPRICE	QUANTITY	CATEGORY	DESCRIPTION	ORIGIN
02-0001-00001	惠普 L23-4650000	7000	210	01-0000-00001		
02-0001-00001	富士 X0550-9100	1100	210	01-0000-00001		
02-0001-00002	三星 SF-6100SS/800	2600	110	01-0000-00002		
02-0001-00001	三星 X055-910A/800	2100	110	01-0000-00002		
02-0001-00001	华硕 C4100	6000	120	01-0000-00001		
02-0001-00001	三星 X055-910A/800	2600	220	01-0000-00001		
02-0001-00002	三星 X055-910A/800	1000	0	0		
02-0001-00002	三星 X055-910A/800	1000	0	0		

图12.4 未提交事务查询结果

图12.4中两处标示即为增加的数据。

- 5) 回滚到保存点，执行如下脚本，并验证数据。执行过程如图12.5所示。



ROLLBACK TO FST;

PRODUCTID	PRODUCTNAME	PRODUCTPRICE	QUANTITY	CATEGORY	RESPECTION	ORIGIN
00-0000-000001	1.2-4000000	70000	200	01-0000-000001		日本
00-0000-000002	2.0-0000000000	11000	750	01-0000-000002		日本
00-0000-000003	3.0-0000000000	11000	12	01-0000-000003		日本
00-0000-000004	4.0-0000000000	20000	110	01-0000-000004		日本
00-0000-000005	5.0-0000000000	10000	175	01-0000-000005		日本
00-0000-000006	6.0-0000000000	20000	220	01-0000-000006		日本
00-0000-000007	7.0-0000000000	10000	80			日本
00-0000-000008	8.0-0000000000	5000	50	01-0000-000008		日本

图12.5 回滚到保存点处数据

从图12.5中可以看出，事务已经成功回滚到保存点处。相信读者看了该示例后对保存点的使用已经没有问题，但笔者建议不要过分依赖保存点，而应尽量把长的事务改成较短的事务操作。

12.1.4 事务的ACID特性

事务有4个特性，它们分别是原子性、一致性、分离性、持久性。

1) 原子性：事务的原子性是指，事务中程序是数据库的逻辑工作单位，它对数据的修改要么全部执行，要么完全不执行。原子也意味着不可分割，不管有多少程序，只要在一个事务中，那么它们就是一个整体，如果都执行成功才意味着该事务成功，而有一个操作失败，那么同一个事务中的其他操作即使执行成功也没有用，事务会使其全部撤销。

2) 一致性：事务的一致性指事务执行的前后数据库都必须处于一致性状态，它是相对脏读而言的。只有在事务完成后才能被所有使用者看见，保证了数据的完整性。例如在银行转账时，从A账户取款但没有放到B账户中时数据是不一致的，同时也是不完整的，其他使用者此时不能看到A中修改后的数据，只有存到B账户中，交易完成并提交事务，这时才算数据一致，所有用户也会看到修改后的数据。

3) 分离性：分离性是指并发事务之间不能相互的干扰。也就是说，一个事务操作的数据不会被其他事务看到和操作。

4) 持久性：持久性是指一旦事务提交完成，那么这将对数据永久的修改，即使被修改后的数据遭到破坏，也不会出现回到修改之前的情况。

注意

事务的提交很重要，但不建议频繁地提交事务，因为每次提交事务都需要时间，如果10 000行记录，每行记录都提交事务，那么事务本身将是性能的主要消耗者。所以，适当地减少事务提交次数比较重要。例如，可以每1000行提交一次。读者可以根据自己的实际情况自行决定。

12.2 什么是锁

数据库是一个庞大的多用户数据管理系统，由于在多用户的系统中，同一时刻多个用户同



时操作某相同资源的情况时有发生，而在逻辑上这些用户想同时操作该资源是不可能的，而数据库中利用锁消除了多用户操作同一资源时可能产生的隐患。本节将介绍锁的概念、作用以及分类等与锁相关的知识。

12.2.1 认识锁

锁出现在数据共享的环境中，它是一种机制，在访问相同资源时，可以防止事务之间的破坏性交互。例如，在多个会话同时操作某表时，优先操作的会话需要对其锁定。

事务的分离性要求当前事务不能影响其他的事务，所以当多个会话访问相同的资源时，数据库系统会利用锁确保它们像队列一样依次进行。Oracle处理数据时用到的锁是自动获取的，我们不用对此有过多的关注，但Oracle允许我们手动锁定数据。

Oracle利用很低的约束提供了最大程度的并发性，例如某会话正在修改一条记录，那么仅仅该记录会被锁定。而其他会话可以随时做读取操作，但读取的依然是修改前的数据。

Oracle的锁保证了数据的完整性。例如，当一个会话对表A的某行记录进行修改时，另一个会话也来修改该行记录，在没有任何处理的情况下保留的数据会有随机性，而这种数据是没有任何意义的，为脏数据。如果此时使用了行级锁，第一个会话修改记录时封锁该行，那么第二个会话此时只能等待，这样就避免了脏数据的产生。

12.2.2 锁的分类

Oracle中分为两种模式的锁，一种是排他锁（X锁），另一种是共享锁（S锁）。

- 排他锁也可以叫写锁。这种模式的锁防止资源的共享，用做数据的修改。假如有事务T给数据A加上该锁，那么其他的事务将不能对A加任何的锁，所以此时只允许T对该数据进行读取和修改，直到事务完成将该类型的锁释放为止。
- 共享锁也可以叫读锁。该模式锁下的数据只能被读取，不能被修改。如果有事务T给数据A加上共享锁后，那么其他事务不能对其加排他锁，只能加共享锁。加了该锁的数据可以被并发地读取。

锁是实现并发的主要手段，在数据库中应用频繁，但很多都由数据库自动管理，当事务提交后会自动释放锁。

12.2.3 锁的类型

Oracle为了使数据库实现高度的并发访问，它使用了不同类型的锁来管理并发会话对数据对象的操作。Oracle的锁按作用对象不同分为如下几种类型。

- DML锁：该类型的锁被称为数据锁，用于保护数据。
- DDL锁：可以保护模式中对象的结构。
- 内部锁：保护数据库的内部结构，完全自动调用。

其中，DML锁主要保证了并发访问时数据的完整性。如果再细分，它又可以分为如下两种类型的锁：

1) 行级锁（TX），也可以称为事务锁。当修改表中某行记录时，需要对将要修改的记录加行级锁，防止两个事务同时修改相同记录，事务结束，该锁也会释放，是粒度最细的锁。该锁只能属于排他锁（X锁）。



2) 表级锁 (TM)，主要作用是防止在修改表的数据时，表的结构发生变化。例如，会话 S 在修改表 A 的数据时它会得到表 A 的 TM 锁，而此时将不允许其他会话对该表进行变更或删除操作。该情况的验证过程如下：

首先，打开 SQL*Plus，修改表 PRODUCTINFO 的记录。脚本如下：

```
UPDATE PRODUCTINFO SET ORIGIN = '修改' WHERE PRODUCTID = 0240090001;
```

此时已经锁定该表，表级锁将不允许在事务结束前其他会话对表 PRODUCTINFO 进行 DDL 操作。

其次，打开另一个 SQL*Plus 窗口，对该表执行 DDL 操作。脚本如下：

```
DROP TABLE PRODUCTINFO;
```

执行后会提示 ORA-00054 错误，效果见图 12.6。

在执行 DML 操作的时候，数据库会先申请数据对象上的共享锁，防止其他的会话对该对象执行 DDL 操作。一旦申请成功，则会对将要修改的记录申请排他锁，如果此时其他会话正在修改该记录，那么等待其事务结束后再为修改的记录加上排他锁。



图 12.6 删除被锁定的表时的提示

表级锁包含如下几种模式：

- ☐ ROW SHARE，行级共享锁 (RS)。该模式下不允许其他的并行会话对同一张表使用排他锁，但允许其利用 DML 语句或 lock 命令锁定同一张表中的其他记录。SELECT ... FROM FOR UPDATE 语句就是给记录加上了 RS 锁。
- ☐ ROW EXCLUSIVE，行级排他锁 (RX)。该模式下允许并行会话对同一张表的其他数据进行修改，但不允许并行会话对同一张表使用排他锁。
- ☐ SHARE，共享锁 (S)。该模式下，不允许会话更新表，但允许对表添加 RS 锁。
- ☐ SHARE ROW EXCLUSIVE，共享行级排他锁 (SRX)。该模式下，不能对同一张表进行 DML 操作，也不能添加 S 锁。
- ☐ EXCLUSIVE，排他锁 (X)。该模式下，其他的并行会话不能对表进行 DML 和 DDL 操作，该表只能读。

表 12.1 列出了以上 5 种模式相互之间的兼容关系。其中，✓ 表示相互兼容，× 表示相互不兼容。

表 12.1 TM5 种模式的相互兼容性

	RS	S	RX	SRX	X
RS	✓	✓	✓	✓	×
S	✓	✓	×	×	×
RX	✓	×	✓	✓	×
SRX	✓	×	×	×	×
X	×	×	×	×	×

如表 12.2 所示是 Oracle 中的各种 SQL 语句所产生的表级锁模式以及允许的锁定模式情况的汇总。

表12.2 SQL语句所产生的表级锁情况

SQL语句	表锁模式	RS	S	RX	SRX	X
SELECT...FROM table...	NONE	Y	Y	Y	Y	Y
INSERT INTO...	RX	Y	N	Y	N	N
UPDATE table...	RX	Y	N	Y	N	N
DELETE FROM table...	RX	Y	N	Y	N	N
SELECT * FROM table FOR UPDATE	RX	Y	N	Y	N	N
LOCK TABLE table IN ROW SHARE MODE	RS	Y	Y	Y	Y	N
LOCK TABLE table IN ROW EXCLUSIVE MODE	RX	Y	N	Y	N	N
LOCK TABLE table IN SHARE MODE	S	Y	N	N	N	N
LOCK TABLE table IN SHARE ROW EXCLUSIVE MODE	SRX	Y	N	N	N	N
LOCK TABLE table IN EXCLUSIVE MODE	X	N	N	N	N	N

在Oracle中除了执行DML时自动为表添加TM锁外，也可以主动地为表添加TM锁。语法如下：

```
LOCK TABLE [schema.] table IN
[EXCLUSIVE]
[SHARE]
[ROW EXCLUSIVE]
[SHARE ROW EXCLUSIVE]
[ROW SHARE* | SHARE UPDATE*]
MODE [NOWAIT]
```

如果要释放它们，只需使用ROLLBACK命令。

DDL锁也可以称为数据字典锁，主要作用是保护模式中对对象的结构。当执行DDL操作时，首先Oracle会自动地隐式提交一次事务，然后自动地给处理对象加上锁；当DDL结束时，Oracle会隐式地提交事务并释放DDL锁。与DML不同的是，用户不能显式地要求使用DDL锁。

DDL锁分为如下3类：

- ☐ Exclusive DDL Lock，排他DDL锁定。如果对象加上了该类型的锁，那么对象不能被其他会话修改，而且该对象也不能再增加其他类型的DDL锁。如果是表，此时可以读取数据。
- ☐ Shared DDL Lock，共享DDL锁定。保护对象的结构，其他会话不能修改该对象的结构，但是允许修改数据。
- ☐ Breakable Parsed Lock，能打破的解析锁定。该类型的锁可以被打断，不能禁止DDL操作。

12.2.4 锁等待与死锁

在某些情况下由于占用的资源不能及时释放，而造成锁等待，也可叫锁冲突。锁等待会严重影响数据库性能和日常工作。例如，当一个会话修改表A的记录时，它会对该记录加锁，而此时如果另一个会话也来修改此记录，那么第二个会话将因得不到排他锁而一直等待，此时会出现执行SQL时数据库长时间没有响应的现象。直到第一个会话把事务提交，释放锁，第二个会话才能对数据进行操作。

【示例3】锁等待

该示例将演示锁等待的现象，具体分为如下两个步骤。

1) 打开SQL*Plus窗口，修改PRODUCTINFO表中PRODUCTID字段为0240090001的记录。脚本如下：

```
UPDATE PRODUCTINFO SET ORIGIN = '修改1' WHERE PRODUCTID = 0240090001;
```

若执行成功则提示：

```
已更新 1 行
```

此时虽然提示已更新，但事务并没有提交。接下来进行第二步操作。

2) 打开另一个SQL*Plus窗口，同样修改PRODUCTINFO表中PRODUCTID字段为0240090001的记录。脚本如下：

```
UPDATE PRODUCTINFO SET ORIGIN = '修改2' WHERE PRODUCTID = 0240090001;
```

此时的执行效果不会提示已更新，而是一直等待。效果见图12.7。



图12.7 等待更新数据

此时的情况是因为第一个会话封锁了该记录，但事务没有结束，锁不会释放，而这时第二个会话也要修改同一条记录，但它却没有办法获得锁，所以只能等待。如果第一个会话修改数据的事务结束，那么第二个会话就会结束等待。及时地结束事务是解决锁等待情况发生的有效方法。

死锁的发生和锁等待不同，它是锁等待的一个特例，通常发生在两个或多个会话之间。假设一个会话想要修改两个资源对象，可以是表也可以是字段，修改这两个资源的操作在一个事务当中。当它修改第一个对象时需要对其锁定，然后等待第二个对象，这时如果另外一个会话也需要修改这两个资源对象，并且已经获得并锁定了第二个对象，那么就会出现死锁，因为当前会话锁定了第一个对象等待第二个对象，而另一个会话锁定了第二个对象等待第一个对象。这样，两个会话都不能得到想要得到的对象，于是出现死锁。

【示例4】死锁的发生

下面是演示死锁发生的示例。具体分为如下4个步骤：

1) 打开第一个SQL*Plus窗口，创建第一个会话，执行如下脚本，修改PRODUCTINFO表中PRODUCTID字段为0240090001的记录。脚本如下：

```
UPDATE PRODUCTINFO SET ORIGIN = '修改' WHERE PRODUCTID = 0240090001;
```

2) 打开第二个SQL*Plus窗口，创建第二个会话，执行如下脚本，修改PRODUCTINFO表中PRODUCTID字段为0240090002的记录。脚本如下：

```
UPDATE PRODUCTINFO SET ORIGIN = '修改' WHERE PRODUCTID = 0240090002;
```

到目前为止，第一个会话锁定了PRODUCTID字段为0240090001的记录，第二个会话锁定了PRODUCTID字段为0240090002的记录。

3) 第一个会话修改第二个会话已经修改的记录。执行脚本如下:

```
UPDATE PRODUCTINFO SET ORIGIN = '修改' WHERE PRODUCTID = 0240090002;
```

此时第一个会话将出现锁等待,因为它修改的对象已经被第二个会话锁定。效果如图12.8所示。



图12.8 第一个会话出现锁等待

4) 第二个会话修改第一个会话已经修改的记录。执行脚本如下:

```
UPDATE PRODUCTINFO SET ORIGIN = '修改' WHERE PRODUCTID = 0240090001;
```

此时会出现死锁的情况。Oracle会自动检测死锁的情况,并释放一个冲突锁,并把消息传递给对方事务。此时在第一个会话窗口中会提示检测到死锁,如图12.9所示。



图12.9 死锁提示

此时Oracle自动做出处理,并重新回到锁等待的情况。出现锁等待的情况时应尽快地找出错误原因并对其进行处理,避免影响数据库性能。实际开发中出现此类情况大致有以下几种原因:

- 1) 用户没有良好的编程习惯,偶尔会忘记提交事务,导致长时间占用资源。
- 2) 操作的记录过多,而且操作过程中没有很好地对其分组。前面介绍过,对于数据量很大的操作,可以将其分成几组提交事务,这样可以避免长时间地占用资源。
- 3) 逻辑错误,两个会话都想得到已占有的资源。

如果操作的数据对象长时间地没有响应,应该考虑锁阻塞的可能。此时可以利用OEM管理器查看。OEM的使用方式后面会有专门的介绍,这里不做过多讲解。

【示例5】使用企业管理器OEM终止锁阻塞情况

该示例可以分为如下3个步骤来查看锁阻塞情况并终止锁阻塞:

1) 登录OEM管理器,在浏览器中输入<https://Oracle服务器IP:1158/em>,然后会提示有认证安装,单击安装即可。输入用户名/密码,单击【登录】按钮,进入图12.10所示的页面。此时可以在该页面的预警列表中发现有阻塞会话统计,见下面的标记。

2) 进入实例锁页面。在图12.10所示的页面中,单击【性能】选项,进入图12.11所示的页面。

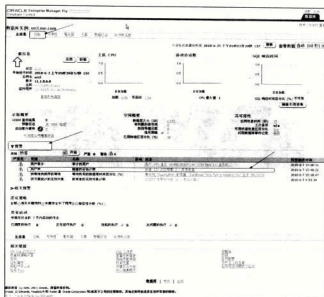


图12.10 登录进入OEM首页面

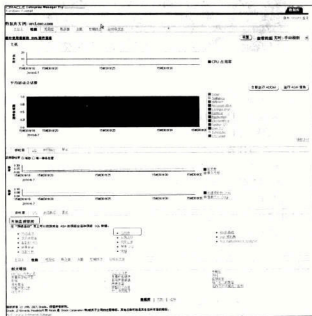


图12.11 【性能】标签页面

在该页面的“其他监视链接”中选择“实例锁”，进入实例锁的操作页面，如图12.12所示。

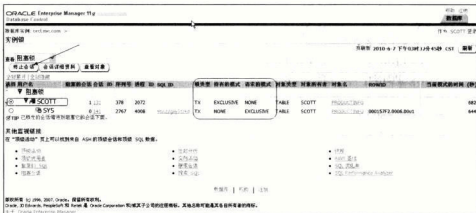


图12.12 实例锁操作页面

3) 终止阻塞锁。从图12.12框出部分可以看出会话ID是131的会话持有一个行级的排他模式的锁，而会话ID是141的会话则在请求同样的锁，所以出现了阻塞的情况。此时开发人员可以选择终止会话ID是131的会话来解决这个问题。选择会话ID是131的会话，单击【终止会话】按钮。在弹出的窗口使用默认选项，单击【是】按钮就完成了会话ID是131会话的终止，而此时的锁等待现象也解决了。

12.3 小结

本章介绍了什么是事务和锁，主要包括事务的类型，如何使用事务以及事务的特性。事务很重要，它保证了数据的一致性。而锁和事务两者联系紧密，对于锁，本章介绍了其分类和类型，以及什么是死锁和锁等待现象，并讲述了如何利用OEM管理器解决锁冲突。通过学习本章，用户可以了解到数据库底层操作数据的理论。

12.4 习题

一、填空题

1. Oracle中使用_____命令提交事务。
2. Oracle中使用_____命令回滚事务。
3. Oracle中使用_____命令设置保存点。
4. 锁被分成_____、_____两种基本类型。

二、选择题

1. TM锁中下列哪种模式和其他模式都不兼容？（ ）

- A. RS
- B. S
- C. RX
- D. X

2. 下面不属于事务操作语句的是（ ）。

- A. ROLLBACK
- B. COMMIT
- C. SAVEPOINT
- D. BEGIN

三、简答题

1. 事务有哪些特性？
2. 保存点的作用是什么？

第三篇

数据库管理篇

第13章 使用Oracle 11g Enterprise Manager

Oracle 11g Enterprise Manager (简称OEM) 可以说在一定程度上解决了初学Oracle 11g的程序员使用数据库的一些问题, 使用OEM可以减少对数据库操作时语句的编写。本章包括以下知识点:

- 什么是Oracle 11g Enterprise Manager
- 如何使用OEM管理Oracle

本章内容基本涵盖了使用OEM管理数据库的一些常用的操作方法。通过本章的学习, 可以熟练地使用OEM来管理Oracle 11g。

13.1 什么是Oracle 11g Enterprise Manager

Oracle 11g中的企业管理是一个Web版的操作页面, 给读者使用OEM带来了别具一格的感覺, 通过本节的学习可以了解到OEM的主要功能以及如何启动OEM。

13.1.1 Oracle 11g Enterprise Manager概述

Oracle 11g中的企业管理器 (OEM) 是以一个图形化界面的方式来实现对数据库管理的, 它为Oracle数据库的使用提供了方便。使用OEM企业管理器可以快速地使用Oracle 11g数据库, 同时也让初学者更加了解Oracle 11g的功能。

OEM为管理Oracle 11g数据库提供的功能如图13.1所示, 即OEM主页面。

从OEM的主页面中就可以看到OEM中提供的功能, 每一个菜单项都是一个操作数据库的内容。具体菜单如下。

- 主目录: 主要用于显示当前数据库中的状态, 提供数据库中的容量、活动会话数、SQL响应时间等性能的显示功能。
- 性能: 主要是以图表的形式显示数据库的运行状态, 有主机的CPU占用率、平均活动会话数等图表显示。
- 可用性: 主要提供数据库的备份和恢复的工作。
- 服务器: 主要是对控制文件、表空间、数据库配置等信息的管理。
- 方案: 主要是对数据库对象、程序、用户定义类型等信息的管理。

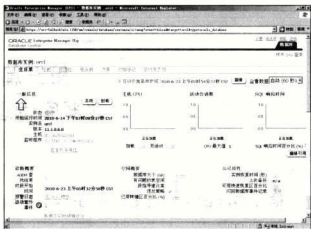


图13.1 OEM主页面

- 数据移动：主要是对数据库中导入导出数据等操作的管理。
- 软件和支持：主要是对数据库的配置和测试等信息的管理。

13.1.2 启动OEM

上面已经看到了进入OEM之后的主页面，那么如何启动OEM呢？实际上，在安装Oracle 11g的同时，就会自动为系统装上OEM的。

1) 启动OEM只需要在开始菜单中打开Oracle 的安装目录，如图13.2所示，就可以看到【Database Control-ocrl】选项，单击后就可以进入OEM的安全警报界面，如图13.3所示。

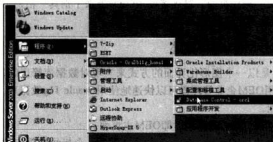


图13.2 启动OEM菜单

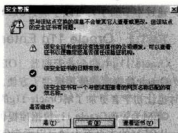


图13.3 安全警报界面

2) 在图13.3所示界面中，单击【是】按钮，接受该证书，即可转到图13.4所示的登录OEM页面。

3) 在图13.4所示页面上输入用户名和口令，并选择连接身份即可登录OEM。连接身份 SYSDBA代表的是系统管理员的身份，Normal代表普通用户身份，登录的身份不同能够使用的功能也不同。在图13.4中就可以看出OEM的端口号是1158，直接在IE浏览器中输入网址即可登录。

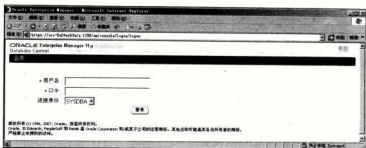


图13.4 OEM登录页面

13.2 使用OEM管理Oracle

前面已经看到了OEM的主页面，那么如何使用主页面的菜单对数据库进行操作呢？本节就将解决这些问题。通过本节的学习就可以掌握如何在OEM中管理数据库了。

13.2.1 OEM中的性能菜单

在OEM的主页面即图13.1所示页面中，单击【性能】标签，出现图13.5所示页面。

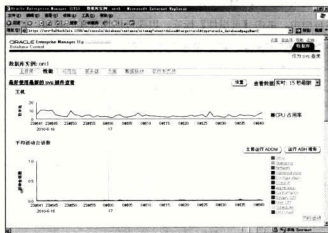


图13.5 OEM中性能菜单页面

在图13.5所示页面中，第一个图显示的是主机的CPU占有率的情况；第二个图显示的是平均活动会话数的情况。主机的CPU占有率情况主要是描述Oracle在使用时占用的CPU情况，平均活动会话数就是与数据库建立连接的个数。在OEM的性能显示页面上也可以使用用户自定义设置要显示的指标。设置的方法是，在图13.5所示的界面中，单击【设置】按钮，会出现图13.6所示的性能页设置页面。

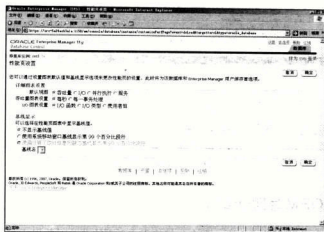


图13.6 性能页设置页面

在性能页设置页面上，可以选择要在图表中显示的内容，如I/O、并行执行、服务等。还可以设置在图表上显示的基线的设置，设置好后单击【确定】按钮即可完成设置。

13.2.2 OEM中的可用性菜单

在OEM的主页面中，单击【可用性】标签，出现可用性管理的页面，如图13.7所示。



图13.7 可用性页面

在可用性页面中可以对Oracle 11g提供备份恢复以及Oracle安全备份。在进行备份和恢复时可以先进行相应的设置，单击【备份设置】选项，进入备份设置页面，如图13.8所示。

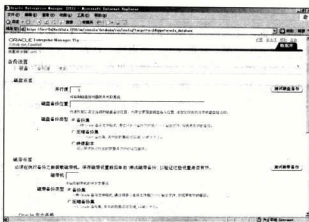


图13.8 备份设置页面

在图13.8所示页面中，可以对备份的设备（磁盘、磁带）进行选择，还可以对备份集以及策略进行设置。

在设置好备份和恢复的条件之后，就可以使用OEM完成管理菜单中的调度备份、管理当前备份、备份报告、管理还原点、执行恢复、查看和管理事务处理等操作。

13.2.3 OEM中的服务器菜单

在OEM主页面中，单击【服务器】标签，进入图13.9所示的服务器选项页面。



图13.9 服务器选项页面

由于服务器选项页面中的内容是数据库维护重要的组成部分，所以下面对每一个选项进行详述。



1. 存储选项

读书笔记

服务器选项页面是数据库管理员经常操作的一个页面，在【存储】选项中，主要提供了对控制文件、表空间、临时表空间组、数据文件、回退段、日志等信息的管理。例如，要管理表空间，直接单击【表空间】选项，进入表空间的操作页面，如图13.10所示。

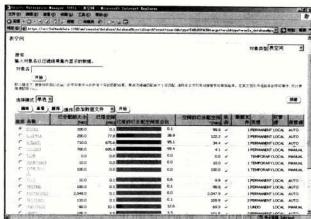


图13.10 表空间操作页面

可以在表空间操作页面对表空间进行编辑、查看、删除以及创建的操作，同时在表空间的操作页面上根据输入的对象名进行查询。但是在OEM中查询默认是模糊查询，如果要进行精确查询，需使用英文的双引号把要查询的内容括起来才行。具体对表空间操作的详细过程可以参考第16章的内容。其他在存储选项中的对象与表空间的操作页面类似，这里就不一一说明了。

2. 数据库配置选项

在数据库配置选项中，主要提供对数据库的内存指导、自动还原管理、初始化参数、查看数据库功能使用情况等操作。下面分别讲解这4个功能的使用。

(1) 数据库的内存指导

在图13.9所示的界面中，单击【内存指导】选项，进入内存指导操作页面，如图13.11所示。

在内存指导页面里可以设置是否启用自动内存管理。默认情况下，Oracle是使用自动内存管理的，数据库的自动内存管理对初学者数据库的使用者来说是必须要使用的，数据库管理员在有经验之后可以自己根据条件进行设置。在自动内存管理下面有一个图表，是显示内存分配的历史记录，其中分为SGA和PGA，SGA称为系统全局区，包含Oracle数据库的数据和控制信息，在数据库实例启动时，就会在内存中分配SGA。在图13.11下方就有对SGA具体分配情况的显示，如图13.12所示。

从图13.12所示的页面中就可以查看到当前SGA组件内存的分配情况，其中SGA的组件包括共享池、缓冲区高速缓存、大型池、Java池以及其他SGA组件。

PGA是一个内存缓冲区，在图13.12所示的页面中，单击【PGA】选项，进入图13.13所示页面中。从图13.13上可以看出，PGA主要包括服务器进程的数据和控制信息，是当服务器进程启动时才创建PGA的。

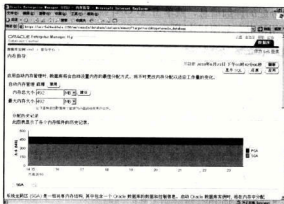


图13.11 内存指导页面

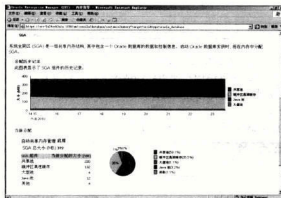


图13.12 SGA内存分配情况



图13.13 PGA内存分配情况

如果要查看PGA分配的详细信息，就单击【PGA内存使用情况详细资料】按钮，此时会弹出PGA内存的详细使用情况，如图13.14所示。



图13.14 PGA内存使用情况详细资料

在图13.14中现在使用的显示方式是【执行百分比】，也可以选择【执行次数】选项，使图表按执行次数显示；还可以对PGA内存使用详细情况大小进行设置，默认是15MB。查看后单击【确定】按钮或直接关闭窗口即可直接返回图13.13所示页面。

(2) 自动还原管理

在图13.9所示的服务器选项页面中，单击【自动还原管理】选项，进入图13.15所示页面。

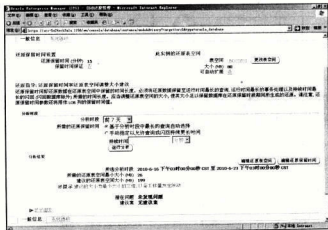


图13.15 自动还原管理一般信息页面

在自动还原管理的一般信息页面上可以看到当前数据库中的还原保留时间和还原数据库用的还原表空间，如果要修改还原表空间和还原保留时间，直接单击页面上的【编辑还原表空间】

或【编辑还原保留时间】按钮即可。在此页面上单击【显示图形】链接，即可查看到以图形的
方式显示的自动还原管理的情况，如图13.16所示。

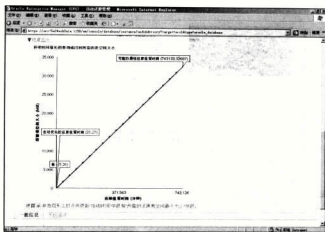


图13.16 自动还原管理

在图13.15所示页面中，单击【系统活动】选项，即可查看到当前数据库中的动态信息，
如图13.17所示。

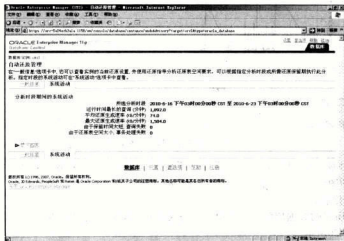


图13.17 自动还原管理系统活动页面

在此页面上就可以查看到当前数据库在指定时间段的系统活动信息，在页面下方单击【
显示图形】链接，即可查看到系统活动信息的图形，如图13.18和图13.19所示。

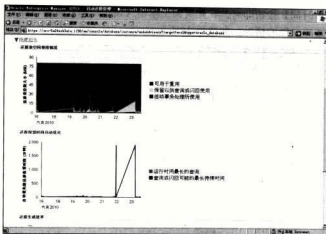


图13.18 系统活动信息图形（1）

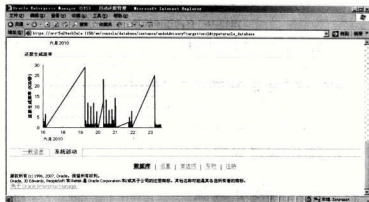


图13.19 系统活动信息图形（2）

在图13.18所示的图形中可以查看到还原表空间的使用情况和还原保留时间的情况，在图13.19所示的图形中可以查看到还原生成速率的情况。

（3）初始化参数

在图13.9所示的数据库配置一栏中，单击【初始化参数】选项，即可进入初始化参数操作页面，如图13.20所示。

在此页面中，可以对数据库的初始化参数信息进行修改。这里所显示的参数都是数据库正在运行时的参数。除了在当前状态下，还可以在SPFile模式下修改。SPFile是用来在服务器端存放初始化参数信息的，只能在系统中才可以修改，不能直接修改文件。SPFile页面如图13.21所示。

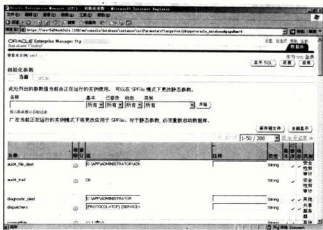


图13.20 初始化参数操作页面

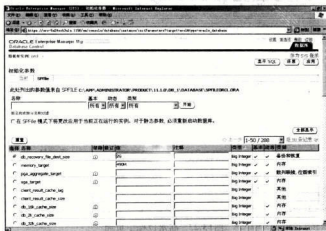


图13.21 SPFile页面

在此页面中可以修改初始化参数的静态值，修改完毕后需要重新启动数据库才可以生效。

(4) 查看数据库功能使用情况

在数据库配置中最后一项内容是查看数据库功能使用情况，数据库管理可以通过此功能方便对数据库的管理。在图13.9所示页面中，单击数据库配置一栏中的【查看数据库功能使用情况】选项，进入图13.22所示页面。

在此页面中可以查看到数据库各个功能的使用情况，包括当前使用的功能以及第一次和上一次使用的时间等信息。通过查看数据库功能选项还可以查看数据库使用时设置的指标最大值，单击【高水位标记】选项，出现图13.23所示页面。



3. Oracle 任务

作业就是执行特定任务的功能，在图13.9所示Oracle Scheduler一栏中选择【作业】选项，进入如图13.24所示页面。

宝剑锋从磨砺出 梅花香自苦寒来

说明

【类似创建】功能就是按照现在选定的作业为模板创建一个新的作业，而【创建】的功能就是直接添加一个新作业，不以原来的作业为模板。

读书笔记

[illegible]

图13.24 作业浏览页面

在此页面还可以通过切换选项卡，查看正在运行的作业以及作业使用的历史记录。这里只演示查看历史记录。单击【历史记录】选项，进入图13.25所示页面。

[illegible]

图13.25 查看作业的历史记录

在此可以通过页面上提供的【查看作业状态】、【清除日志】、【查看作业定义】以及【清除所有日志】按钮来操作历史记录。

4. 统计信息管理

统计信息管理主要用来存储数据库中的一些统计信息，在统计信息管理的栏目中共有两个



选项，一个是自动工作量资料档案库；另一个是AWR基线。下面分别讲解这两个功能的使用。

(1) 自动工作量资料档案库

自动工作量资料档案库主要用于存储用于优化性能的数据库统计信息。在图13.9所示页面的统计信息管理一栏中单击【自动工作量资料档案库】选项，进入图13.26所示页面。

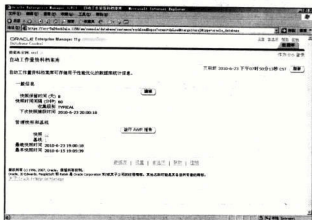


图13.26 自动工作量资料档案库页面

在此页中有统计的一般信息，主要是快照保留时间以及快照的时间间隔等信息，并可单击【编辑】按钮来编辑当前快照的信息；还有管理快照和基线的信息。这里还可以单击【运行AWR报告】按钮来查看AWR的信息，如图13.27所示。

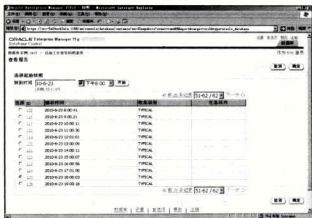


图13.27 查看快照信息

在此页面还可以按照快照的时间来查询信息。

(2) AWR基线

AWR (Automatic Workload Repository) 基线是Oracle中收集数据库统计信息的工具，是

在 Oracle 10g 中新增的功能, 它可以修改快照的收集和保存时间。在图 13.9 所示的统计信息管理一栏中单击【AWR 基线】选项, 进入 AWR 基线的操作页面, 如图 13.28 所示。

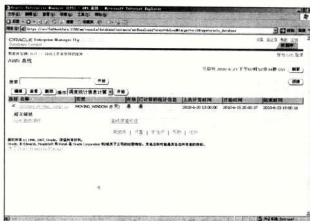


图 13.28 AWR 基线操作页面

在此页面中可以对 AWR 基线信息进行编辑、查看、删除、创建以及查询的操作。这里就不一一介绍了, 请读者自行练习。

5. 资源管理器

资源管理器一栏中的功能主要是配置资源管理。通过对资源管理器一栏中的【入门】选项里提供的功能介绍, 可以更加了解资源管理器一栏中给予的 5 个功能。

- 使用者组: 定义根据资源处理需求将用户会话分组的使用者组。资源计划将资源分配给使用者组。指定一些用户和角色, 他们可以在每个使用者组中开始会话或者将他们的会话切换到每个使用者组中。
- 使用者组映射: 定义使用者组映射规则, 此规则基于会话属性 (如用户名、服务名、模块名等) 将用户会话映射到使用者组。要解决映射冲突, 请按映射规则优先级的顺序应用映射规则。
- 计划: 定义资源计划, 其中包含指定将资源分配给使用者组方式的指令。例如, 对于每个计划, 需要指定分配给每个使用者组的 CPU 资源百分比。可以选择指定其他限制, 如使用者组中的会话可以执行或保持空闲的最大时间, 或者会话在自动切换到低优先级的使用者组之前消耗的 CPU 或 I/O 资源的最大值。
- 设置: 查看当前活动的资源计划, 激活资源计划。
- 资源管理器统计信息: 监视当前启用的资源计划的统计信息。按使用者组监视 CPU 和 I/O 的使用情况, 并按使用者组监视资源管理器为 CPU 执行的约束数。

下面一一演示每一个功能。

(1) 使用者组

在图 13.9 所示页面中选择资源管理器一栏中的【使用者组】选项, 即可进入到图 13.29 所示页面。

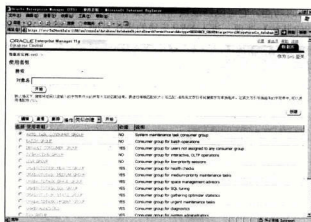


图13.29 使用者组操作页面

在此页面上可以对使用者组的信息进行编辑、查看、删除以及创建的操作，同时也提供了查询的功能。

(2) 使用者组映射

在图13.9所示页面中选择资源管理器一栏中的【使用者组映射】选项，即可进入到图13.30所示页面。

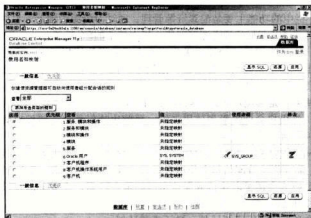


图13.30 使用者组映射页面

在此页面中可以为使用者组添加映射。添加映射的方法是，选择一个使用者组映射，单击【添加所选类型的规则】按钮，进入图13.31所示页面。

在此为当前的使用者组添加映射规则，添加好映射规则之后，单击【确定】按钮，完成映射规则的添加。

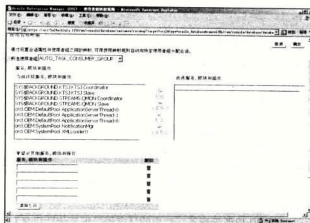


图 13.31 添加使用者组映射页面

(3) 计划

在图13.9所示页面中选择资源管理器一栏中的【计划】选项，即可进入到图13.32所示页面。这里的计划指的就是资源计划。



图 13.32 资源计划操作页面

在此就可以对资源计划进行编辑、查看、删除以及创建的操作，同时也提供了查询的功能。

(4) 设置

在图13.9所示页面中选择资源管理器一栏中的【设置】选项，即可进入到图13.33所示页面。设置就是指对资源管理器的设置。

在此页面上可以通过单击【查看所选资源计划】按钮，查看资源计划的详细信息，也可以使当前所选的资源计划通过单击【激活所选资源计划】按钮激活。

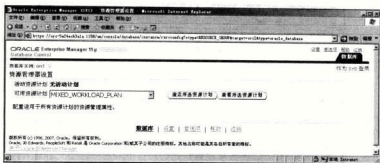


图13.33 资源管理器设置页面

(5) 资源管理器统计信息

在图13.9所示页面中选择资源管理器一栏中的【资源管理器统计信息】选项，即可进入到图13.34所示页面。

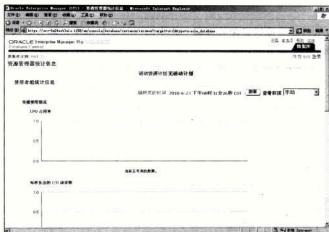


图13.34 资源管理器统计信息页面

在此就可以查看到资源管理器中的统计信息，可以通过刷新来查看实时的情况。

6. 安全性

在服务器选项页面的安全性一栏中主要提供数据库中的用户、角色、概要文件、审计设置、透明数据加密、虚拟专用数据库策略、应用程序上下文7项确保数据库安全的功能维护工作。其中，用户、角色以及概要文件的管理详细操作可以参见第17章。下面对剩余的4项确保安全性的操作进行一一讲解。

(1) 审计设置

审计设置是可以把审计信息写入数据库或操作系统的文件中。在图13.9所示页面的安全性一栏中单击【审计设置】选项，进入图13.35所示页面。

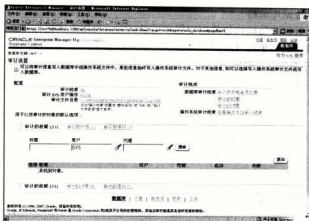


图13.35 审计设置页面

在此页面中是对审计的权限进行设置。除了对权限进行设置外，还可以对设计的对象以及语句进行编辑。

(2) 透明数据加密

透明数据加密是用来保护数据库列中敏感数据的机制，在使用透明数据加密时一定要确保 Oracle Wallet 存在并且处于打开的状态。在图13.9所示页面的安全性一栏中单击【透明数据加密】选项，进入图13.36所示页面。

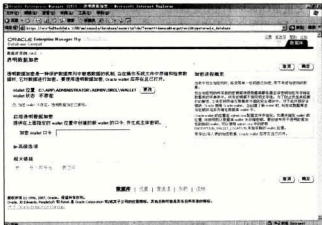


图13.36 透明数据加密页面

在此页面中可以看出透明数据加密功能处于不可用状态，由于Oracle Wallet不存在，如果要启用透明数据加密，需选择页面中的高级选项，在其中选择重新生成密钥之后再启用透明数据加密的功能。

(3) 虚拟专用数据库策略

使用虚拟专用数据库策略构建的应用程序可以在表、视图上实施行级安全策略。在图13.9所示页面的安全性一栏中单击【虚拟专用数据库策略】选项，进入图13.37所示页面。

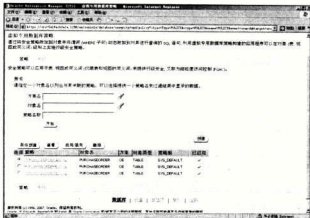


图13.37 虚拟专用数据库策略页面

在此，可以对安全策略进行创建、查看、启用/禁用以及删除的操作，同时还提供了按方案名、对象名或者策略名进行查询的操作。

(4) 应用程序上下文

应用程序上下文也是Oracle 11g中提供的一个确保数据库安全的功能。在图13.9所示页面的安全性一栏中单击【应用程序上下文】选项，进入图13.38所示页面。

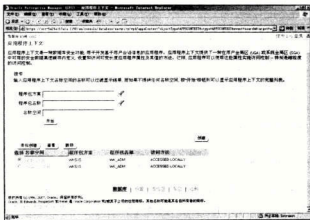


图13.38 应用程序上下文页面

在此，可以对应用程序上下文进行创建、查看以及删除的操作，同时也提供了按程序包方

案、程序包名称或者名称空间进行查询的操作。当不输入名称空间时,还可以得到应用程序上下文的完整列表。

7. 查询优化程序

查询优化的处理对于数据库性能是至关重要的,特别是复杂的SQL语句更是如此。查询优化程序是指确定每一次查询的最佳策略。在查询优化程序一栏中,OEM中提供的操作有管理优化程序统计信息、SQL Plan Control两项。下面分别讲解这两项的使用。

(1) 管理优化程序统计信息

管理优化程序统计信息提供对优化程序统计信息的操作,在图13.9所示页面的查询优化程序一栏中单击【管理优化程序统计信息】选项,进入图13.39所示页面。



图13.39 管理优化程序统计信息页面

在此,可以对优化程序统计信息进行搜集、还原、锁定、取消锁定、删除的操作。这里只以搜集优化程序统计信息为例进行讲解,其他的选项就不一一介绍了,请读者自行尝试。在图13.39所示页面中,单击【搜集优化程序统计信息】选项,进入图13.40所示页面。



图13.40 搜集优化程序统计信息页面



在此页面上可以首先选择要搜索的对象类型，然后单击【下一步】按钮，根据提示完成搜索优化程序统计信息的操作。

(2) SQL Plan Control

SQL Plan Control选项主要包括SQL的辅助信息，可以利用它来优化程序。在图13.9所示页面的查询优化程序一栏中单击【SQL Plan Control】选项，进入图13.41所示页面。



图13.41 SQL Plan Control页面

13.2.4 OEM中的方案菜单

在OEM主界面中，单击【方案】标签，进入图13.42所示的方案选项页面。



图13.42 方案选项页面

在方案选项页面中可以对数据库对象、程序、实体化视图、用户定义类型等信息进行操作。



在这里由于对数据库对象操作比较频繁，所以对这部分内容进行详细的讲解。数据库对象主要包括表、索引、视图、同义词、序列、数据库链接、目录对象、重组对象。这里主要对表、索引、同义词和序列做详细的讲解。

1. 表

表是数据库中最常用的数据库对象，没有表数据库就将失去意义。在第4章中已经讲解过如何使用语句来创建表、修改表、删除表，初学者也可以通过企业管理器操作表。下面分别介绍使用企业管理器创建表、修改表、删除表的操作。

【示例1】创建学生表

下面就利用企业管理器创建一个学生表，假设学生信息包括学生的学号、姓名、年龄这3个字段。具体创建步骤如下。

(1) 进入创建表的页面

在图13.42所示的页面中，单击数据库对象一栏中的【表】选项，进入图13.43所示页面。



图13.43 数据库对象表页面

在此页面中可以浏览到当前在指定方案下的表信息，如果要浏览当前方案SYS下的表信息，直接单击【开始】按钮，即可搜索出相应的表信息，如图13.44所示。

(2) 在SYS方案下创建表

如果在SYS方案下创建表，就可以直接单击图13.44所示页面中的【创建】按钮，进入创建表页面，如图13.45所示。

创建表的第一步就是要为表确定使用的组织，有标准表、索引表，同时还可以选择创建临时表。选择好表创建的类型后，单击【继续】按钮创建表，如图13.46所示。

在此就可以看到在页面上填入的数据与第4章学习的使用SQL语句创建表是一致的。首先要给表设置一个名称，然后选择方案。这里是在SYS方案下直接创建的，所以显示的方案是SYS。也可以根据自己的需要选择不同的方案，选择方案只需单击方案后面的小手电图标即可完成。然后选择表空间，如果不选择，则使用默认表空间，这样就定义完表的结构了。接着就可以向表中添加数据了。添加数据的方法也是先添加表中的列名，之后是数据类型、大小、小数位数、是否为空、设置默认值等操作。具体创建过程如图13.47所示。

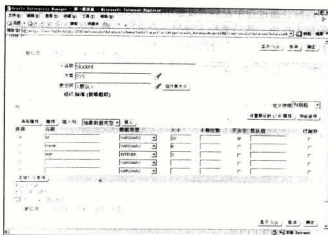


图13.47 添加表中字段页面

在此，可以看到已经为表student增加了3个字段，分别是id、name、age。并选中id字段的“不为空”选项，表示id字段是不允许为空的，没有选中此项的其他两个字段name和age中的值是可以为空的。

(3) 给表设置约束

如果需要在表中为字段添加约束，在图13.47所示页面中单击【约束条件】选项即可，如图13.48所示。

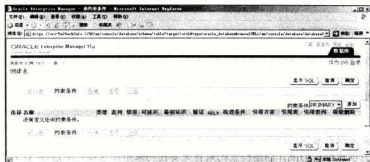


图13.48 添加约束页面

在此页面中可以根据选择不同的约束条件为表中的字段添加约束。这里将表中的id设置为主键，并给age字段设置一个检查约束，来指定在age字段中只能输入18~30的数。先加入一个主键约束，在选择条件的下拉列表框中选择PRIMARY选项，单击【添加】按钮，进入添加主键的页面，如图13.49所示。

主键可以由表中的一列或多列组成，但是在一个表中只能有一个主键。在图13.49所示的页面中，左侧的列表框中存放的是表中的所有字段，右侧的列表框中存放的是要设置成主键的



字段。设置主键字段时，通过选中左侧列表框中的字段然后单击【移动】选项，即可把学生编号id移动到右侧的列表框中，然后要给主键起一个名字，默认的名字就是图13.49上显示的名字。单击【确定】按钮，即可完成主键的添加，如图13.50所示。

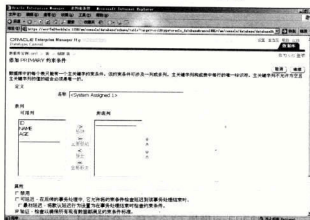


图13.49 添加主键页面

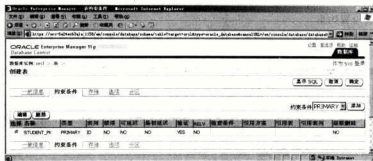


图13.50 约束列表页面

在此，就可以查看到已经创建的主键约束STUDENT_PK。

下面接着创建检查约束。创建检查约束首先要约在约束条件选择框中选择【CHECK】选项，单击【添加】按钮，即可进入添加检查约束的页面，如图13.51所示。

在此，可以看到设置的检查约束的名字是Student_chk，检查条件是年龄在18~30。设置好后，单击【继续】按钮出现图13.52所示页面，即可完成检查约束的创建。

在此，可以看到已经为表student添加了主键约束和检查约束两个约束。至此，表student就创建完成了。对于其他约束的添加操作，请读者自行练习。

【示例2】修改表

修改表操作使用企业管理器操作也是比较简单的，只要在图13.44所示页面中选择要修改的表，然后单击【编辑】按钮，即可进入图13.53所示页面。

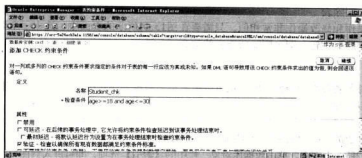


图13.51 添加检查约束页面

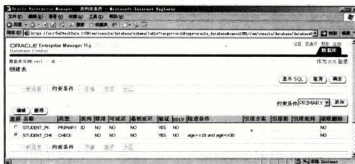


图13.52 查看约束页面



图13.53 修改表页面

在修改表信息时只能修改表中的字段和表的名称，其他内容是不能修改的。修改好相应的信息后，单击【应用】按钮，即可完成修改表的操作。

【示例3】删除表

删除的操作和修改操作非常相似。只需要在图13.44所示的页面中，选中一个要删除的表，然后单击【删除】按钮，即可删除所选表的信息。

2. 索引

索引能够节省数据的查询时间，也可以说索引是数据库优化的一个手段，索引在企业管理器中也是可以创建的。

【示例4】创建索引

为前面创建的student表中age列添加一个标准-B树索引。

在图13.42所示页面中，单击数据库对象一栏中的【索引】选项，即可进入到图13.54所示页面。



图13.54 索引浏览页面

在索引浏览页面中，单击【创建】按钮，即可进入创建索引页面，如图13.55所示。



图13.55 创建索引页面

在创建索引页面中，填入索引的名称并选择索引的类型。索引类型包括标准-B树索引和位图索引两种，标准-B树索引是默认和常用的索引类型，位图索引是对于小型数据比较有用的一种索引类型。在基本信息添加完成后，就可以添加设置索引的表了，表名可以通过在文本框后面单击小手电的图标来添加，并选择设置索引的列。具体的操作方法如图13.56所示。

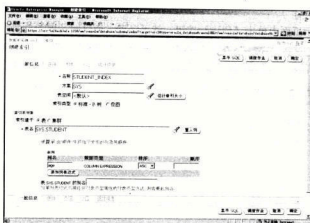


图13.56 填入索引信息页面

在此，为前面创建的student表中age列添加了一个标准-B树索引，单击【确定】按钮后即可完成索引的创建。在创建索引时还可以为创建索引的表起别名，这样就方便了表的使用。对于索引的修改以及删除的操作在这里就不详细说明了，请读者自行测试。

3. 同义词

同义词的概念在数据库中就是给数据库中的对象设置别名，这些数据库对象可以是表、视图、函数等，定义好的同义词存储在数据字典中。使用同义词会提高数据库的安全性。下面就讲解如何在企业管理器中创建同义词。

【示例5】创建同义词

(1) 进入同义词浏览页面

在企业管理器中浏览同义词只需要在图13.42所示页面中单击【同义词】选项，进入图13.57所示页面。



图13.57 浏览同义词页面



(2) 创建同义词

在图13.57所示页面中单击【创建】按钮，就可以进入创建同义词的页面，如图13.58所示。

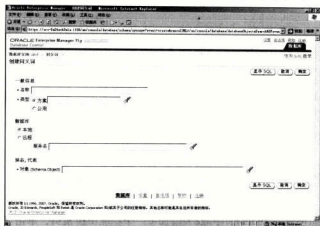


图13.58 添加同义词页面

在此页面中，可以看到同义词的对象分为方案和公用两种类型，方案类型又称为专用类型，意思就是定义的一个方案类型的同义词，那么这个同义词仅能够被该用户或者授权用户所使用，而公用类型的同义词，可以被数据库中的所有用户使用。默认的同义词类型是方案类型，这样会保证数据库的安全性。

这里选择方案类型，并为同义词选择一个方案，单击“类型”文本框后面的小手电的图标，进入图13.59所示页面。在此页面中选择一个对象类型，然后填入方案名以及对象名，即可完成选择。如果记不清具体的方案名和对象名，可以单击【开始】按钮，在数据库中查询，如图13.60所示。

从查询结果中可以看到要查询的方案和对象名存在，单击【选择】按钮，就可以完成方案的选择。方案选择完成后就可以填入数据库等信息，信息添加完成后单击【确定】按钮即可完成同义词的创建。除了在企业管理器中可以完成同义词的创建外，还可以对同义词信息进行编辑、查看以及删除的操作，这部分内容请读者自行完成。



图13.59 选择方案页面



图13.60 查询方案页面

4. 序列

序列在数据库中创建表时经常使用,序列就相当于SQL Server数据库中的自增长的列,在Oracle中没有直接设置表中字段自增长的选项,只能使用序列来完成这个工作。序列的特点是可以提供一组唯一的数值来表示数据表中需要显示唯一值的字段,但是使用序列时并不一定能够保证序列会产生一系列连续的值。在企业管理中创建序列也是非常容易的。下面就创建一个序列。

【示例6】创建序列

(1) 进入序列浏览页面

在企业管理器中浏览序列只需要在图13.42所示页面中单击【序列】选项,进入图13.61所示页面。

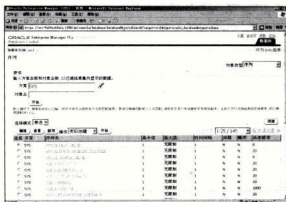


图13.61 序列浏览页面

(2) 创建序列

创建序列是在图13.61所示页面中单击【创建】按钮,进入图13.62所示的页面。

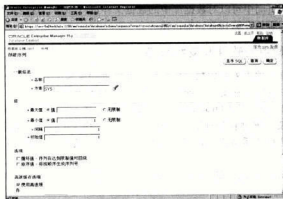


图13.62 创建序列页面

在“一般信息”栏中填入序列的名称、方案；在“值”栏中可以填入序列的最大值和最小值以及每个值之间的间隔、初始值，默认的情况就是最小值、间隔以及初始值均为1，也就是序列是从1开始每个值之间增加1。除了上面的两项之外，还可以选择序列值产生的方式，有循环值和排序值两种方式。循环值是指序列在达到限制值时回绕，排序值是按顺序生成序列号。对于序列的编辑、查看以及删除的操作可以在图13.61所示的页面中完成。

13.2.5 OEM中的数据库移动菜单

对于OEM中的数据库移动的功能，主要提供对数据库的导入导出以及移动数据文件的操作。数据移动菜单是在OEM主页面图13.1中，单击【数据移动】标签，进入图13.63所示的数据移动选项页面。

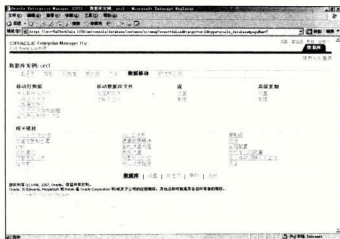


图13.63 数据移动选项页面

在数据移动选项页面中可以对数据库中的数据文件进行移动，这也是数据库备份和恢复必要的操作。这部分内容将在第18章详细讲述。

13.2.6 OEM中的软件和支持菜单

软件和支持菜单选项用来完成数据库配置以及处理数据库中的一些补丁问题，在图13.1所示的主页面中，单击【软件和支持】标签，进入图13.64所示页面。

在图13.64中可以看到对于Oracle软件本身的一些操作，如配置、数据库软件打补丁、真实应用程序测试以及部署过程管理器，还有就是支持工作台的操作。这里，可以尝试使用真实应用程序测试中数据库重放选项比较两个数据库。单击【数据库重放】选项进入图13.65所示页面。

通过图13.65所示的页面，就可以在任务列表中选择捕获工作量、预处理捕获的工作量以及重放工作量的任务。



图13.64 软件和支持菜单页面



图13.65 数据库重放页面

13.3 小结

本章详细讲述了如何启动企业管理器OEM以及企业管理器中的性能菜单、可用性菜单、服务器菜单、方案菜单、数据移动菜单、软件和支持菜单，并着重讲解了性能菜单、服务器菜单以及方案菜单中的内容。通过本章的学习，初学Oracle的读者可以快速地使用企业管理器OEM来操作数据库，避免了使用语句操作数据库的麻烦。

13.4 习题

简答题

1. 使用企业管理器创建用户USERTEST，并给用户赋予管理员的权限。
2. 使用企业管理器查看当前数据库使用情况。
3. 使用企业管理器创建商品信息表，表中的字段有商品编号、商品名称、商品价格，要求将商品编号列设置为主键，给商品价格设置一个检查约束，要求价格大于0。
4. 使用企业管理器创建序列，并将该序列应用在第3题中的商品编号上。
5. 使用企业管理器创建第3题需创建的表的同义词。

第14章 常用工具介绍

SQL*Plus是一个操作Oracle数据库使用的工具,它是在DOS界面下操作的一个数据库工具,也是数据库管理员经常使用的一个管理数据库的工具。本章包括以下知识点:

- SQL*Plus简介
- 如何使用SQL*Plus工具
- PL/SQL Developer介绍

本章内容涵盖了SQL*Plus的基本使用方法。通过本章的学习,可以熟练地使用SQL*Plus操作数据库。

14.1 什么是SQL*Plus

SQL*Plus是操作Oracle数据库的工具,它是与Oracle数据库一起安装的。本节将讲述如何启动SQL*Plus。

14.1.1 SQL*Plus简介

在安装完数据库之后,就可以直接使用SQL*Plus来管理数据库。除了SQL*Plus之外,现在还有iSQL*Plus这个操作数据库的工具,它们的区别就是SQL*Plus使用DOS页面,而iSQL*Plus使用Web页面的形式,也就是通过浏览器就可以访问数据库。

通过SQL*Plus可以完成对数据库的操作主要有:

- 对数据库的数据进行增加、删除、修改、查询的操作;
- 可以对查询出的结果进行格式化的显示;
- 对数据库对象进行管理,如用户、表空间、角色等对象。

14.1.2 启动SQL*Plus

使用SQL*Plus的前提是确保Oracle数据库安装成功,在连接SQL*Plus时需要用户名和密码,那么在连接SQL*Plus之前可以在数据库中查询好用户名和密码,便于操作SQL*Plus。启动SQL*Plus的方法有多种方式,直接在【开始】菜单中选择【程序】|【Oracle-OraDb11g_home1】|【应用程序开发】|【SQL Plus】选项,如图14.1所示,即可启动SQL*Plus的连接页面。

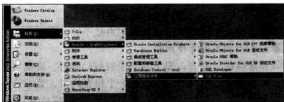


图14.1 连接SQL*Plus

除了直接启动SQL*Plus之外,还可以用Windows的DOS窗口来完成启动SQL*Plus的操作。下面就在Windows的DOS窗口下分别完成连接默认数据库和连接指定数据库的操作。

1. 连接默认数据库

连接默认数据库也就是连接当前正在使用的数据库。连接默认数据库的具体步骤如下:

1) 打开Windows命令窗口。在Windows 2003中选择【开始】|【运行】菜单,出现如图14.2所示的【运行】对话框。在此对话框中输入“cmd”,进入Windows的命令页面,如图14.3所示。



图14.2 Windows运行窗口



图14.3 Windows命令页面

2) 输入“sqlplus”命令。在图14.3所示页面中的命令提示符处输入“sqlplus”命令,进入图14.4所示页面。

3) 输入用户名和密码。在图14.4所示页面中,用户名这里使用的是sys,密码是安装数据库时设置的密码。输入用户名和密码后即可进入到图14.5所示的页面。



图14.4 进入sqlplus页面



图14.5 连接默认数据库页面

出现SQL>就说明连接默认数据库成功。

2. 连接指定数据库

连接指定数据库,是在已经连接到默认数据库之后再进行其他数据库的连接。在SQL>命令后输入以下语句:

```
SQL >connect username/password @Oracle net名称
```

说明

除了在已经连接到默认数据库后连接其他数据库外,也可以在进入sqlplus时直接连接到其他数据库中。使用语句如下:

```
c: >sqlplus username/password @Oracle net名称
```

14.2 使用SQL*Plus

前面已经讲解过如何启动SQL*Plus,本节主要讲解SQL*Plus的编辑、保存、运行以及格

式化查询结果的操作。

14.2.1 使用SQL*Plus编辑命令

在SQL*Plus中所做的工作就是输入不同语句完成不同的操作结果，这在输入语句时就要用到SQL*Plus中的编辑功能。学习SQL*Plus的编辑功能之后，就可以方便输出结果的显示以及输入SQL语句。

在学习编辑命令之前首先要知道如何在SQL*Plus中运行语句。运行语句是比较简单的，只要在SQL*Plus中的SQL>提示符后输入语句，按【Enter】键即可运行。同时，如果要运行上一次输入的语句，直接按键盘上的“/”键即可。

(1) 追加文本

在执行完一个语句之后，需要在上一个语句的基础上增加一些内容时，可以使用追加文本的命令完成操作。具体语法如下：

Append text;

其中，text就是要追加的文本。

【示例1】 查询dba_users表，追加排序语句

先查询表中用户名 (username) 和用户ID (user_id) 字段, 查询结果如图14.6所示。从此图显示的查询结果中可以看出, username并没有按顺序, 下面使用追加命令向该语句追加一个按username排序的语句。操作结果如图14.7所示。

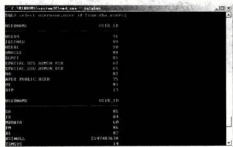


图 14.6 查询dba_users表



图14.7 追加排序语句

运行追加命令后,会出现追加后的一条完成语句,之后在SQL>命令符后面输入“/”,即可运行该语句。查询结果如图14.8所示。

从排序后的查询结果可以看出，结果已经按照username进行排序了。

(2) 替換文本

在编辑时，有时需要对原有的语句进行更改，这时SQL*Plus就提供了文本替换的方法。语句如下：

Change /old text/new text

【语法说明】

- ☐ old text: 被替换字符串中原来的内容。
☐ new text: 要替换的新内容。



从图14.10中可以看出，在执行完插入语句向表test中插入语句后，如果删除了第二行，那么使用“/”再次执行该语句时就可以发现缺少了第二个语句。也就是在SQL*Plus中使用删除命令指的是删除在缓存中的语句。

- 删除缓冲区中全部的语句
- 删除缓冲区中全部语句的命令如下：

```
CLEAR BUFFER
```

该语句是删除缓冲区的全部内容。

【示例4】删除缓冲区中全部内容

利用上面的语句，在执行增加一条数据的语句之后，删除缓冲区，然后再使用“/”。具体效果如图14.11所示。



图14.11 删除缓冲区中全部语句

在删除缓冲区的内容之后，再次用“/”执行语句时，就会显示缓冲区中没有可运行的程序。这就说明了删除缓冲区的操作成功。

(4) 添加行

在编辑语句时，除了删除和替换之外，常用的操作还有添加操作。添加行的操作是通过INPUT命令实现的，添加行的操作也是指在缓冲中执行过的语句中增加。具体语句如下：

```
INPUT text
```

【示例5】添加一条语句

下面就利用添加行的语句完成在执行完查询dba_users语句之后，再添加一个排序语句的功能。执行查询dba_users的用户名和用户ID操作的结果如图14.6所示，使用INPUT语句添加按用户名排序后，结果如图14.12所示。

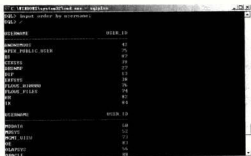


图14.12 排序后的结果

在原来查询语句select username,user_id from dba_users后面添加了order by username，使用“/”执行语句，结果就是排序后的结果，说明添加语句操作成功，也可以根据自己的需要，

添加多行语句。



说明 使用append追加命令与input添加命令的区别就是，append是在缓冲区中最后一条语句后面添加内容，而input是添加行的对象，虽然也是在缓冲区执行的语句中添加，但是当缓冲区中没有语句时，使用input之后就相当于在缓冲区中写入了第一行语句。这是要特别注意的。

(5) 显示缓冲区中的内容

在SQL*Plus执行语句后内容会保存在缓冲区中，如果要查询缓冲区中的内容，SQL*Plus提供了LIST命令完成查询操作。使用LIST命令可以查询出全部缓冲区的内容，也可以查询出指定行数的内容。LIST命令的具体语法如下：

```
LIST [n/LAST/]
```

这里，在LIST后面如果省略后面的参数就代表查询缓冲区全部的内容；n代表显示缓冲区中指定行的内容；LAST代表查询出缓冲区中最后一行语句。

下面利用LIST语句分别完成查询缓冲区中全部内容、查询第二行内容、查询最后一行内容的操作。

【示例6】查询缓冲区中全部内容

首先清空缓冲区中的全部内容，然后在缓冲区中使用建表语句创建一个表listtest并执行。使用LIST语句查询缓冲区中全部内容的操作如图14.13所示。



图14.13 查询缓冲区中全部内容

【示例7】查询缓冲区中第二行内容和最后一行内容

查询缓冲区中第二行内容使用LIST n这个命令结构，查询缓冲区中最后一行内容使用LIST LAST命令。具体查询结果如图14.14所示。

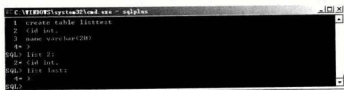


图14.14 查询缓冲区中第二行内容和最后一行内容

上面所讲的编辑命令全部要在SQL*Plus中使用，如果想在记事本中编辑SQL*Plus中缓冲区的内容，则在SQL*Plus窗口中使用“ed”命令，就可以直接调出记事本进行缓冲区内容的编辑，编辑完成后关闭记事本后即可回到SQL*Plus中。



14.2.2 使用SQL*Plus保存命令

读书笔记



在SQL*Plus中执行过的命令如何保存呢？在14.2.1小节中已经提到可以在缓冲区中调出记事本。也就是说，可以通过记事本来保存缓冲区的内容。除了使用记事本之外，还可以通过SQL*Plus中的SAVE命令来完成保存的操作。下面就分别讲解如何使用记事本和SAVE命令来保存缓冲区中的内容。

1. 通过记事本保存缓冲区中的内容

(1) 在SQL*Plus中打开记事本

在SQL*Plus中的SQL>后输入ed（在Windows环境下），打开一个含有缓冲区内容的记事本文件，如图14.15所示。



图14.15 打开记事本

在输入ed命令之后，可以看到在下一行出现“已写入file afiedt.buf”，说明已经把缓冲区的内容写入记事本中。

(2) 保存缓冲区的内容

在图14.15所示的页面中，选择【文件】|【另存为】选项，在弹出的【另存为】对话框中为记事本选择一个存放的位置即可完成缓冲区文件的保存操作。

2. 通过SAVE命令保存缓冲区的内容

在SQL*Plus中保存缓冲区的内容是数据库管理员经常会使用的操作。使用SAVE命令完成保存的语法如下：

```
SAVE file_name;
```

这里file_name可以只写一个文件名，也可以写上具体的保存路径，如果只写一个文件名，就表示把当前缓冲区的内容保存到默认的文件夹中。

下面就利用保存命令完成把缓冲区的内容保存到c:\下。具体操作如图14.16所示。



图14.16 使用SAVE命令保存缓冲区内容

在此，先使用LIST命令查询缓冲区的全部内容，再使用SAVE命令把缓冲区的内容写入到C盘下的buftest.sql文件中。运行SAVE命令后，如果出现图中的“已创建 file c:\buftest.sql”，说明保存缓冲区内容操作成功。

技巧

写入到缓冲区的内容保存到文件后, 可以使用SQL>@filename的方法运行刚写入文件的缓冲区的内容。



14.2.3 使用SQL*Plus运行命令

在SQL*Plus中运行命令的方法前面已经使用过“/”的方式, 而在语句之后加上“;”也可以运行语句。那么除了前面已经使用的方式之外, 还可以使用RUN命令或者START命令运行的方式。下面分别讲解使用RUN命令和START命令运行语句。

1. 使用RUN命令运行语句

RUN命令运行的语句也是缓冲区中的内容。具体语法如下:

RUN;

【示例8】运行缓冲区的内容

下面使用RUN命令运行在图14.16中的缓冲区的内容, 运行结果如图14.17所示。



图14.17 使用RUN命令

2. 使用 START命令运行语句

使用START命令可以运行指定文件中的语句。具体语法如下:

START file_name

其中, file_name是要运行的文件名, 这里文件名可以加上扩展名也可以不加扩展名, 如果不加扩展名, 默认的扩展名就是.sql。

【示例9】运行指定文件中的内容

下面利用START命令运行使用SAVE命令保存的文件buftest.sql, 运行结果如图14.18所示。



图14.18 使用START命令运行文件

在此可以看出, 在使用START命令运行文件时, 是不显示要运行的文件内容的, 而是直接显示运行结果。

14.2.4 使用SQL*Plus格式化查询结果

在SQL*Plus中查询数据时经常会碰到的问题就是由于查询的列过多无法在一行全部显示出来, 结果看起来非常混乱。例如, 查询dba_users中的全部数据。查询结果如图14.19所示。

从查询结果中可以看出由于查询出的列比较多, 显示时根本看不出具体的查询结果, 只是知道查询出了41条数据。为了能够在SQL*Plus中显示出比较清晰的查询结果, SQL*Plus提供了多种设置查询结果显示的方法。下面讲解几个比较常用的设置命令。



图14.19 查询dba_users中全部数据

1. 使用COLUMN命令

使用COLUMN命令既可设置显示列的别名也可以设置显示列的格式。下面分别讲解COLUMN命令的两种用法。

(1) 使用COLUMN命令设置别名

使用COLUMN命令设置别名与在SQL语句中设置别名的方法类似。具体语法如下:

COLUMN	oldname	HEADING	newname
--------	---------	---------	---------

【语法说明】

- ❑ oldname: 要查询表中的列名。
- ❑ newname: 新的列名。

【示例10】 设置别名

下面就使用上面的命令，查询dba_users中的用户名（username）和用户ID（user_id），并把username的列名使用COLUMN命令换成“用户名”。查询结果如图14.20所示。

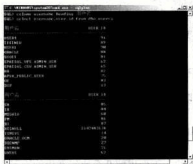


图14.20 使用COLUMN命令设置查询结果的列名



在此，username列的列名即被替换成“用户名”，设置查询结果操作成功。

(2) 使用COLUMN命令格式化数据显示的格式

在使用SQL*Plus查询数据时经常会遇到需要格式化显示的数字格式，在SQL*Plus中仍然可以使用COLUMN命令完成。具体语法如下：

COLUMN column_name FORMAT dataformat

【语法说明】

□ column_name：格式化查询结果的列名。

□ dataformat：格式化后显示的格式。设置数据格式一般规则如表14.1所示。

【示例11】格式化查询结果

下面使用COLUMN命令格式化查询结果，查询dba_users中的用户名（username）和用户ID（user_id），并把user_id的列名显示的格式写成{0, 0}。格式化后的查询结果如图14.21所示。

在此，通过COLUMN命令格式化后，已经把user_id的显示格式改写成0.0的格式了。

表14.1 数据格式设置

格 式	含 义
9	代表一个数字字符
0	在指定的位置显示前导0或后置0
\$	美元货币符号
B	显示一个空
MI	显示负号
,	显示千分位分隔符
.	显示小数点
G	显示千分位分组符号
L	显示本地区域的货币符号

使用COLUMN命令设置列的格式之后，这个列的格式是会一直保持的，除非重新设置

说明 该列的格式或者取消该列的格式。取消列的格式可以用COLUMN column_name clear命令来完成。例如，如果要取消对username的列设置，结果如图14.22所示。

```

SQL> column user_id format 0.0
SQL> select username,user_id from dba_users;

用户名                USER_ID
-----
SYS                     0.1
SYSTEM                  0.2
SYSDBA                  0.3
SYSRAC                  0.4
SYSBACKUP               0.5
SYSKM                   0.6
SYSOPER                 0.7
SYSPRIVILEGE            0.8
SYSDG                   0.9
SYSRAC                  0.10
SYSBACKUP               0.11
SYSKM                   0.12
SYSOPER                 0.13
SYSPRIVILEGE            0.14
SYSDG                   0.15
SYSRAC                  0.16
SYSBACKUP               0.17
SYSKM                   0.18
SYSOPER                 0.19
SYSPRIVILEGE            0.20
SYSDG                   0.21
SYSRAC                  0.22
SYSBACKUP               0.23
SYSKM                   0.24
SYSOPER                 0.25
SYSPRIVILEGE            0.26
SYSDG                   0.27
SYSRAC                  0.28
SYSBACKUP               0.29
SYSKM                   0.30
SYSOPER                 0.31
SYSPRIVILEGE            0.32
SYSDG                   0.33
SYSRAC                  0.34
SYSBACKUP               0.35
SYSKM                   0.36
SYSOPER                 0.37
SYSPRIVILEGE            0.38
SYSDG                   0.39
SYSRAC                  0.40
SYSBACKUP               0.41
SYSKM                   0.42
SYSOPER                 0.43
SYSPRIVILEGE            0.44
SYSDG                   0.45
SYSRAC                  0.46
SYSBACKUP               0.47
SYSKM                   0.48
SYSOPER                 0.49
SYSPRIVILEGE            0.50
SYSDG                   0.51
SYSRAC                  0.52
SYSBACKUP               0.53
SYSKM                   0.54
SYSOPER                 0.55
SYSPRIVILEGE            0.56
SYSDG                   0.57
SYSRAC                  0.58
SYSBACKUP               0.59
SYSKM                   0.60
SYSOPER                 0.61
SYSPRIVILEGE            0.62
SYSDG                   0.63
SYSRAC                  0.64
SYSBACKUP               0.65
SYSKM                   0.66
SYSOPER                 0.67
SYSPRIVILEGE            0.68
SYSDG                   0.69
SYSRAC                  0.70
SYSBACKUP               0.71
SYSKM                   0.72
SYSOPER                 0.73
SYSPRIVILEGE            0.74
SYSDG                   0.75
SYSRAC                  0.76
SYSBACKUP               0.77
SYSKM                   0.78
SYSOPER                 0.79
SYSPRIVILEGE            0.80
SYSDG                   0.81
SYSRAC                  0.82
SYSBACKUP               0.83
SYSKM                   0.84
SYSOPER                 0.85
SYSPRIVILEGE            0.86
SYSDG                   0.87
SYSRAC                  0.88
SYSBACKUP               0.89
SYSKM                   0.90
SYSOPER                 0.91
SYSPRIVILEGE            0.92
SYSDG                   0.93
SYSRAC                  0.94
SYSBACKUP               0.95
SYSKM                   0.96
SYSOPER                 0.97
SYSPRIVILEGE            0.98
SYSDG                   0.99
SYSRAC                  1.00
SYSBACKUP               1.01
SYSKM                   1.02
SYSOPER                 1.03
SYSPRIVILEGE            1.04
SYSDG                   1.05
SYSRAC                  1.06
SYSBACKUP               1.07
SYSKM                   1.08
SYSOPER                 1.09
SYSPRIVILEGE            1.10
SYSDG                   1.11
SYSRAC                  1.12
SYSBACKUP               1.13
SYSKM                   1.14
SYSOPER                 1.15
SYSPRIVILEGE            1.16
SYSDG                   1.17
SYSRAC                  1.18
SYSBACKUP               1.19
SYSKM                   1.20
SYSOPER                 1.21
SYSPRIVILEGE            1.22
SYSDG                   1.23
SYSRAC                  1.24
SYSBACKUP               1.25
SYSKM                   1.26
SYSOPER                 1.27
SYSPRIVILEGE            1.28
SYSDG                   1.29
SYSRAC                  1.30
SYSBACKUP               1.31
SYSKM                   1.32
SYSOPER                 1.33
SYSPRIVILEGE            1.34
SYSDG                   1.35
SYSRAC                  1.36
SYSBACKUP               1.37
SYSKM                   1.38
SYSOPER                 1.39
SYSPRIVILEGE            1.40
SYSDG                   1.41
SYSRAC                  1.42
SYSBACKUP               1.43
SYSKM                   1.44
SYSOPER                 1.45
SYSPRIVILEGE            1.46
SYSDG                   1.47
SYSRAC                  1.48
SYSBACKUP               1.49
SYSKM                   1.50
SYSOPER                 1.51
SYSPRIVILEGE            1.52
SYSDG                   1.53
SYSRAC                  1.54
SYSBACKUP               1.55
SYSKM                   1.56
SYSOPER                 1.57
SYSPRIVILEGE            1.58
SYSDG                   1.59
SYSRAC                  1.60
SYSBACKUP               1.61
SYSKM                   1.62
SYSOPER                 1.63
SYSPRIVILEGE            1.64
SYSDG                   1.65
SYSRAC                  1.66
SYSBACKUP               1.67
SYSKM                   1.68
SYSOPER                 1.69
SYSPRIVILEGE            1.70
SYSDG                   1.71
SYSRAC                  1.72
SYSBACKUP               1.73
SYSKM                   1.74
SYSOPER                 1.75
SYSPRIVILEGE            1.76
SYSDG                   1.77
SYSRAC                  1.78
SYSBACKUP               1.79
SYSKM                   1.80
SYSOPER                 1.81
SYSPRIVILEGE            1.82
SYSDG                   1.83
SYSRAC                  1.84
SYSBACKUP               1.85
SYSKM                   1.86
SYSOPER                 1.87
SYSPRIVILEGE            1.88
SYSDG                   1.89
SYSRAC                  1.90
SYSBACKUP               1.91
SYSKM                   1.92
SYSOPER                 1.93
SYSPRIVILEGE            1.94
SYSDG                   1.95
SYSRAC                  1.96
SYSBACKUP               1.97
SYSKM                   1.98
SYSOPER                 1.99
SYSPRIVILEGE            2.00
SYSDG                   2.01
SYSRAC                  2.02
SYSBACKUP               2.03
SYSKM                   2.04
SYSOPER                 2.05
SYSPRIVILEGE            2.06
SYSDG                   2.07
SYSRAC                  2.08
SYSBACKUP               2.09
SYSKM                   2.10
SYSOPER                 2.11
SYSPRIVILEGE            2.12
SYSDG                   2.13
SYSRAC                  2.14
SYSBACKUP               2.15
SYSKM                   2.16
SYSOPER                 2.17
SYSPRIVILEGE            2.18
SYSDG                   2.19
SYSRAC                  2.20
SYSBACKUP               2.21
SYSKM                   2.22
SYSOPER                 2.23
SYSPRIVILEGE            2.24
SYSDG                   2.25
SYSRAC                  2.26
SYSBACKUP               2.27
SYSKM                   2.28
SYSOPER                 2.29
SYSPRIVILEGE            2.30
SYSDG                   2.31
SYSRAC                  2.32
SYSBACKUP               2.33
SYSKM                   2.34
SYSOPER                 2.35
SYSPRIVILEGE            2.36
SYSDG                   2.37
SYSRAC                  2.38
SYSBACKUP               2.39
SYSKM                   2.40
SYSOPER                 2.41
SYSPRIVILEGE            2.42
SYSDG                   2.43
SYSRAC                  2.44
SYSBACKUP               2.45
SYSKM                   2.46
SYSOPER                 2.47
SYSPRIVILEGE            2.48
SYSDG                   2.49
SYSRAC                  2.50
SYSBACKUP               2.51
SYSKM                   2.52
SYSOPER                 2.53
SYSPRIVILEGE            2.54
SYSDG                   2.55
SYSRAC                  2.56
SYSBACKUP               2.57
SYSKM                   2.58
SYSOPER                 2.59
SYSPRIVILEGE            2.60
SYSDG                   2.61
SYSRAC                  2.62
SYSBACKUP               2.63
SYSKM                   2.64
SYSOPER                 2.65
SYSPRIVILEGE            2.66
SYSDG                   2.67
SYSRAC                  2.68
SYSBACKUP               2.69
SYSKM                   2.70
SYSOPER                 2.71
SYSPRIVILEGE            2.72
SYSDG                   2.73
SYSRAC                  2.74
SYSBACKUP               2.75
SYSKM                   2.76
SYSOPER                 2.77
SYSPRIVILEGE            2.78
SYSDG                   2.79
SYSRAC                  2.80
SYSBACKUP               2.81
SYSKM                   2.82
SYSOPER                 2.83
SYSPRIVILEGE            2.84
SYSDG                   2.85
SYSRAC                  2.86
SYSBACKUP               2.87
SYSKM                   2.88
SYSOPER                 2.89
SYSPRIVILEGE            2.90
SYSDG                   2.91
SYSRAC                  2.92
SYSBACKUP               2.93
SYSKM                   2.94
SYSOPER                 2.95
SYSPRIVILEGE            2.96
SYSDG                   2.97
SYSRAC                  2.98
SYSBACKUP               2.99
SYSKM                   3.00
SYSOPER                 3.01
SYSPRIVILEGE            3.02
SYSDG                   3.03
SYSRAC                  3.04
SYSBACKUP               3.05
SYSKM                   3.06
SYSOPER                 3.07
SYSPRIVILEGE            3.08
SYSDG                   3.09
SYSRAC                  3.10
SYSBACKUP               3.11
SYSKM                   3.12
SYSOPER                 3.13
SYSPRIVILEGE            3.14
SYSDG                   3.15
SYSRAC                  3.16
SYSBACKUP               3.17
SYSKM                   3.18
SYSOPER                 3.19
SYSPRIVILEGE            3.20
SYSDG                   3.21
SYSRAC                  3.22
SYSBACKUP               3.23
SYSKM                   3.24
SYSOPER                 3.25
SYSPRIVILEGE            3.26
SYSDG                   3.27
SYSRAC                  3.28
SYSBACKUP               3.29
SYSKM                   3.30
SYSOPER                 3.31
SYSPRIVILEGE            3.32
SYSDG                   3.33
SYSRAC                  3.34
SYSBACKUP               3.35
SYSKM                   3.36
SYSOPER                 3.37
SYSPRIVILEGE            3.38
SYSDG                   3.39
SYSRAC                  3.40
SYSBACKUP               3.41
SYSKM                   3.42
SYSOPER                 3.43
SYSPRIVILEGE            3.44
SYSDG                   3.45
SYSRAC                  3.46
SYSBACKUP               3.47
SYSKM                   3.48
SYSOPER                 3.49
SYSPRIVILEGE            3.50
SYSDG                   3.51
SYSRAC                  3.52
SYSBACKUP               3.53
SYSKM                   3.54
SYSOPER                 3.55
SYSPRIVILEGE            3.56
SYSDG                   3.57
SYSRAC                  3.58
SYSBACKUP               3.59
SYSKM                   3.60
SYSOPER                 3.61
SYSPRIVILEGE            3.62
SYSDG                   3.63
SYSRAC                  3.64
SYSBACKUP               3.65
SYSKM                   3.66
SYSOPER                 3.67
SYSPRIVILEGE            3.68
SYSDG                   3.69
SYSRAC                  3.70
SYSBACKUP               3.71
SYSKM                   3.72
SYSOPER                 3.73
SYSPRIVILEGE            3.74
SYSDG                   3.75
SYSRAC                  3.76
SYSBACKUP               3.77
SYSKM                   3.78
SYSOPER                 3.79
SYSPRIVILEGE            3.80
SYSDG                   3.81
SYSRAC                  3.82
SYSBACKUP               3.83
SYSKM                   3.84
SYSOPER                 3.85
SYSPRIVILEGE            3.86
SYSDG                   3.87
SYSRAC                  3.88
SYSBACKUP               3.89
SYSKM                   3.90
SYSOPER                 3.91
SYSPRIVILEGE            3.92
SYSDG                   3.93
SYSRAC                  3.94
SYSBACKUP               3.95
SYSKM                   3.96
SYSOPER                 3.97
SYSPRIVILEGE            3.98
SYSDG                   3.99
SYSRAC                  4.00
SYSBACKUP               4.01
SYSKM                   4.02
SYSOPER                 4.03
SYSPRIVILEGE            4.04
SYSDG                   4.05
SYSRAC                  4.06
SYSBACKUP               4.07
SYSKM                   4.08
SYSOPER                 4.09
SYSPRIVILEGE            4.10
SYSDG                   4.11
SYSRAC                  4.12
SYSBACKUP               4.13
SYSKM                   4.14
SYSOPER                 4.15
SYSPRIVILEGE            4.16
SYSDG                   4.17
SYSRAC                  4.18
SYSBACKUP               4.19
SYSKM                   4.20
SYSOPER                 4.21
SYSPRIVILEGE            4.22
SYSDG                   4.23
SYSRAC                  4.24
SYSBACKUP               4.25
SYSKM                   4.26
SYSOPER                 4.27
SYSPRIVILEGE            4.28
SYSDG                   4.29
SYSRAC                  4.30
SYSBACKUP               4.31
SYSKM                   4.32
SYSOPER                 4.33
SYSPRIVILEGE            4.34
SYSDG                   4.35
SYSRAC                  4.36
SYSBACKUP               4.37
SYSKM                   4.38
SYSOPER                 4.39
SYSPRIVILEGE            4.40
SYSDG                   4.41
SYSRAC                  4.42
SYSBACKUP               4.43
SYSKM                   4.44
SYSOPER                 4.45
SYSPRIVILEGE            4.46
SYSDG                   4.47
SYSRAC                  4.48
SYSBACKUP               4.49
SYSKM                   4.50
SYSOPER                 4.51
SYSPRIVILEGE            4.52
SYSDG                   4.53
SYSRAC                  4.54
SYSBACKUP               4.55
SYSKM                   4.56
SYSOPER                 4.57
SYSPRIVILEGE            4.58
SYSDG                   4.59
SYSRAC                  4.60
SYSBACKUP               4.61
SYSKM                   4.62
SYSOPER                 4.63
SYSPRIVILEGE            4.64
SYSDG                   4.65
SYSRAC                  4.66
SYSBACKUP               4.67
SYSKM                   4.68
SYSOPER                 4.69
SYSPRIVILEGE            4.70
SYSDG                   4.71
SYSRAC                  4.72
SYSBACKUP               4.73
SYSKM                   4.74
SYSOPER                 4.75
SYSPRIVILEGE            4.76
SYSDG                   4.77
SYSRAC                  4.78
SYSBACKUP               4.79
SYSKM                   4.80
SYSOPER                 4.81
SYSPRIVILEGE            4.82
SYSDG                   4.83
SYSRAC                  4.84
SYSBACKUP               4.85
SYSKM                   4.86
SYSOPER                 4.87
SYSPRIVILEGE            4.88
SYSDG                   4.89
SYSRAC                  4.90
SYSBACKUP               4.91
SYSKM                   4.92
SYSOPER                 4.93
SYSPRIVILEGE            4.94
SYSDG                   4.95
SYSRAC                  4.96
SYSBACKUP               4.97
SYSKM                   4.98
SYSOPER                 4.99
SYSPRIVILEGE            5.00
SYSDG                   5.01
SYSRAC                  5.02
SYSBACKUP               5.03
SYSKM                   5.04
SYSOPER                 5.05
SYSPRIVILEGE            5.06
SYSDG                   5.07
SYSRAC                  5.08
SYSBACKUP               5.09
SYSKM                   5.10
SYSOPER                 5.11
SYSPRIVILEGE            5.12
SYSDG                   5.13
SYSRAC                  5.14
SYSBACKUP               5.15
SYSKM                   5.16
SYSOPER                 5.17
SYSPRIVILEGE            5.18
SYSDG                   5.19
SYSRAC                  5.20
SYSBACKUP               5.21
SYSKM                   5.22
SYSOPER                 5.23
SYSPRIVILEGE            5.24
SYSDG                   5.25
SYSRAC                  5.26
SYSBACKUP               5.27
SYSKM                   5.28
SYSOPER                 5.29
SYSPRIVILEGE            5.30
SYSDG                   5.31
SYSRAC                  5.32
SYSBACKUP               5.33
SYSKM                   5.34
SYSOPER                 5.35
SYSPRIVILEGE            5.36
SYSDG                   5.37
SYSRAC                  5.38
SYSBACKUP               5.39
SYSKM                   5.40
SYSOPER                 5.41
SYSPRIVILEGE            5.42
SYSDG                   5.43
SYSRAC                  5.44
SYSBACKUP               5.45
SYSKM                   5.46
SYSOPER                 5.47
SYSPRIVILEGE            5.48
SYSDG                   5.49
SYSRAC                  5.50
SYSBACKUP               5.51
SYSKM                   5.52
SYSOPER                 5.53
SYSPRIVILEGE            5.54
SYSDG                   5.55
SYSRAC                  5.56
SYSBACKUP               5.57
SYSKM                   5.58
SYSOPER                 5.59
SYSPRIVILEGE            5.60
SYSDG                   5.61
SYSRAC                  5.62
SYSBACKUP               5.63
SYSKM                   5.64
SYSOPER                 5.65
SYSPRIVILEGE            5.66
SYSDG                   5.67
SYSRAC                  5.68
SYSBACKUP               5.69
SYSKM                   5.70
SYSOPER                 5.71
SYSPRIVILEGE            5.72
SYSDG                   5.73
SYSRAC                  5.74
SYSBACKUP               5.75
SYSKM                   5.76
SYSOPER                 5.77
SYSPRIVILEGE            5.78
SYSDG                   5.79
SYSRAC                  5.80
SYSBACKUP               5.81
SYSKM                   5.82
SYSOPER                 5.83
SYSPRIVILEGE            5.84
SYSDG                   5.85
SYSRAC                  5.86
SYSBACKUP               5.87
SYSKM                   5.88
SYSOPER                 5.89
SYSPRIVILEGE            5.90
SYSDG                   5.91
SYSRAC                  5.92
SYSBACKUP               5.93
SYSKM                   5.94
SYSOPER                 5.95
SYSPRIVILEGE            5.96
SYSDG                   5.97
SYSRAC                  5.98
SYSBACKUP               5.99
SYSKM                   6.00
SYSOPER                 6.01
SYSPRIVILEGE            6.02
SYSDG                   6.03
SYSRAC                  6.04
SYSBACKUP               6.05
SYSKM                   6.06
SYSOPER                 6.07
SYSPRIVILEGE            6.08
SYSDG                   6.09
SYSRAC                  6.10
SYSBACKUP               6.11
SYSKM                   6.12
SYSOPER                 6.13
SYSPRIVILEGE            6.14
SYSDG                   6.15
SYSRAC                  6.16
SYSBACKUP               6.17
SYSKM                   6.18
SYSOPER                 6.19
SYSPRIVILEGE            6.20
SYSDG                   6.21
SYSRAC                  6.22
SYSBACKUP               6.23
SYSKM                   6.24
SYSOPER                 6.25
SYSPRIVILEGE            6.26
SYSDG                   6.27
SYSRAC                  6.28
SYSBACKUP               6.29
SYSKM                   6.30
SYSOPER                 6.31
SYSPRIVILEGE            6.32
SYSDG                   6.33
SYSRAC                  6.34
SYSBACKUP               6.35
SYSKM                   6.36
SYSOPER                 6.37
SYSPRIVILEGE            6.38
SYSDG                   6.39
SYSRAC                  6.40
SYSBACKUP               6.41
SYSKM                   6.42
SYSOPER                 6.43
SYSPRIVILEGE            6.44
SYSDG                   6.45
SYSRAC                  6.46
SYSBACKUP               6.47
SYSKM                   6.48
SYSOPER                 6.49
SYSPRIVILEGE            6.50
SYSDG                   6.51
SYSRAC                  6.52
SYSBACKUP               6.53
SYSKM                   6.54
SYSOPER                 6.55
SYSPRIVILEGE            6.56
SYSDG                   6.57
SYSRAC                  6.58
SYSBACKUP               6.59
SYSKM                   6.60
SYSOPER                 6.61
SYSPRIVILEGE            6.62
SYSDG                   6.63
SYSRAC                  6.64
SYSBACKUP               6.65
SYSKM                   6.66
SYSOPER                 6.67
SYSPRIVILEGE            6.68
SYSDG                   6.69
SYSRAC                  6.70
SYSBACKUP               6.71
SYSKM                   6.72
SYSOPER                 6.73
SYSPRIVILEGE            6.74
SYSDG                   6.75
SYSRAC                  6.76
SYSBACKUP               6.77
SYSKM                   6.78
SYSOPER                 6.79
SYSPRIVILEGE            6.80
SYSDG                   6.81
SYSRAC                  6.82
SYSBACKUP               6.83
SYSKM                   6.84
SYSOPER                 6.85
SYSPRIVILEGE            6.86
SYSDG                   6.87
SYSRAC                  6.88
SYSBACKUP               6.89
SYSKM                   6.90
SYSOPER                 6.91
SYSPRIVILEGE            6.92
SYSDG                   6.93
SYSRAC                  6.94
SYSBACKUP               6.95
SYSKM                   6.96
SYSOPER                 6.97
SYSPRIVILEGE            6.98
SYSDG                   6.99
SYSRAC                  7.00
SYSBACKUP               7.01
SYSKM                   7.02
SYSOPER                 7.03
SYSPRIVILEGE            7.04
SYSDG                   7.05
SYSRAC                  7.06
SYSBACKUP               7.07
SYSKM                   7.08
SYSOPER                 7.09
SYSPRIVILEGE            7.10
SYSDG                   7.11
SYSRAC                  7.12
SYSBACKUP               7.13
SYSKM                   7.14
SYSOPER                 7.15
SYSPRIVILEGE            7.16
SYSDG                   7.17
SYSRAC                  7.18
SYSBACKUP               7.19
SYSKM                   7.20
SYSOPER                 7.21
SYSPRIVILEGE            7.22
SYSDG                   7.23
SYSRAC                  7.24
SYSBACKUP               7.25
SYSKM                   7.26
SYSOPER                 7.27
SYSPRIVILEGE            7.28
SYSDG                   7.29
SYSRAC                  7.30
SYSBACKUP               7.31
SYSKM                   7.32
SYSOPER                 7.33
SYSPRIVILEGE            7.34
SYSDG                   7.35
SYSRAC                  7.36
SYSBACKUP               7.37
SYSKM                   7.38
SYSOPER                 7.39
SYSPRIVILEGE            7.40
SYSDG                   7.41
SYSRAC                  7.42
SYSBACKUP               7.43
SYSKM                   7.44
SYSOPER                 7.45
SYSPRIVILEGE            7.46
SYSDG                   7.47
SYSRAC                  7.48
SYSBACKUP               7.49
SYSKM                   7.50
SYSOPER                 7.51
SYSPRIVILEGE            7.52
SYSDG                   7.53
SYSRAC                  7.54
SYSBACKUP               7.55
SYSKM                   7.56
SYSOPER                 7.57
SYSPRIVILEGE            7.58
SYSDG                   7.59
SYSRAC                  7.60
SYSBACKUP               7.61
SYSKM                   7.62
SYSOPER                 7.63
SYSPRIVILEGE            7.64
SYSDG                   7.65
SYSRAC                  7.66
SYSBACKUP               7.67
SYSKM                   7.68
SYSOPER                 7.69
SYSPRIVILEGE            7.70
SYSDG                   7.71
SYSRAC                  7.72
SYSBACKUP               7.73
SYSKM                   7.74
SYSOPER                 7.75
SYSPRIVILEGE            7.76
SYSDG                   7.77
SYSRAC                  7.78
SYSBACKUP               7.79
SYSKM                   7.80
SYSOPER                 7.81
SYSPRIVILEGE            7.82
SYSDG                   7.83
SYSRAC                  7.84
SYSBACKUP               7.85
SYSKM                   7.86
SYSOPER                 7.87
SYSPRIVILEGE            7.88
SYSDG                   7.89
SYSRAC                  7.90
SYSBACKUP               7.91
SYSKM                   7.92
SYSOPER                 7.93
SYSPRIVILEGE            7.94
SYSDG                   7.95
SYSRAC                  7.96
SYSBACKUP               7.97
SYSKM                   7.98
SYSOPER                 7.99
SYSPRIVILEGE            8.00
SYSDG                   8.01
SYSRAC                  8.02
SYSBACKUP               8.03
SYSKM                   8.04
SYSOPER                 8.05
SYSPRIVILEGE            8.06
SYSDG                   8.07
SYSRAC                  8.08
SYSBACKUP               8.09
SYSKM                   8.10
SYSOPER                 8.11
SYSPRIVILEGE            8.12
SYSDG                   8.13
SYSRAC                  8.14
SYSBACKUP               8.15
SYSKM                   8.16
SYSOPER                 8.17
SYSPRIVILEGE            8.18
SYSDG                   8.19
SYSRAC                  8.20
SYSBACKUP               8.21
SYSKM                   8.22
SYSOPER                 8.23
SYSPRIVILEGE            8.24
SYSDG                   8.25
SYSRAC                  8.26
SYSBACKUP               8.27
SYSKM                   8.28
SYSOPER                 8.29
SYSPRIVILEGE            8.30
SYSDG                   8.31
SYSRAC                  8.32
SYSBACKUP               8.33
SYSKM                   8.34
SYSOPER                 8.35
SYSPRIVILEGE            8.36
SYSDG                   8.37
SYSRAC                  8.38
SYSBACKUP               8.39
SYSKM                   8.40
SYSOPER                 8.41
SYSPRIVILEGE            8.42
SYSDG                   8.43
SYSRAC                  8.44
SYSBACKUP               8.45
SYSKM                   8.46
SYSOPER                 8.47
SYSPRIVILEGE            8.48
SYSDG                   8.49
SYSRAC                  8.50
SYSBACKUP               8.51
SYSKM                   8.52
SYSOPER                 8.53
SYSPRIVILEGE            8.54
SYSDG                   8.55
SYSRAC                  8.56
SYSBACKUP               8.57
SYSKM                   8.58
SYSOPER                 8.59
SYSPRIVILEGE            8.60
SYSDG                   8.61
SYSRAC                  8.62
SYSBACKUP               8.63
SYSKM                   8.64
SYSOPER                 8.65
SYSPRIVILEGE            8.66
SYSDG                   8.67
SYSRAC                  8.68
SYSBACKUP               8.69
SYSKM                   8.70
SYSOPER                 8.71
SYSPRIVILEGE            8.72
SYSDG                   8.73
SYSRAC                  8.74
SYSBACKUP               8.75
SYSKM                   8.76
SYSOPER                 8.77
SYSPRIVILEGE            8.78
SYSDG                   8.79
SYSRAC                  8.80
SYSBACKUP               8.81
SYSKM                   8.82
SYSOPER                 8.83
SYSPRIVILEGE            8.84
SYSDG                   8.85
SYSRAC                  8.86
SYSBACKUP               8.87
SYSKM                   8.88
SYSOPER                 8.89
SYSPRIVILEGE            8.90
SYSDG                   8.91
SYSRAC                  8.92
SYSBACKUP               8.93
SYSKM                   8.94
SYSOPER                 8.95
SYSPRIVILEGE            8.96
SYSDG                   8.97
SYSRAC                  8.98
SYSBACKUP               8.99
SYSKM                   9.00
SYSOPER                 9.01
SYSPRIVILEGE            9.02
SYSDG                   9.03
SYSRAC                  9.04
SYSBACKUP               9.05
SYSKM                   9.06
SYSOPER                 9.07
SYSPRIVILEGE            9.08
SYSDG                   9.09
SYSRAC                  9.10
SYSBACKUP               9.11
SYSKM                   9.12
SYSOPER                 9.13
SYSPRIVILEGE            9.14
SYSDG                   9.15
SYSRAC                  9.16
SYSBACKUP               9.17
SYSKM                   9.18
SYSOPER                 9.19
SYSPRIVILEGE            9.20
SYSDG                   9.21
SYSRAC                  9.22
SYSBACKUP               9.23
SYSKM                   9.24
SYSOPER                 9.25
SYSPRIVILEGE            9.26
SYSDG                   9.27
SYSRAC                  9.28
SYSBACKUP               9.29
SYSKM                   9.30
SYSOPER                 9.31
SYSPRIVILEGE            9.32
SYSDG                   9.33
SYSRAC                  9.34
SYSBACKUP               9.35
SYSKM                   9.36
SYSOPER                 9.37
SYSPRIVILEGE            9.38
SYSDG                   9.39
SYSRAC                  9.40
SYSBACKUP               9.41
SYSKM                   9.42
SYSOPER                 9.43
SYSPRIVILEGE            9.44
SYSDG                   9.45
SYSRAC                  9.46
SYSBACKUP               9.47
SYSKM                   9.48
SYSOPER                 9.49
SYSPRIVILEGE            9.50
SYSDG                   9.51
SYSRAC                  9.52
SYSBACKUP               9.53
SYSKM                   9.54
SYSOPER                 9.55
SYSPRIVILEGE            9.56
SYSDG                   9.57
SYSRAC                  9.58
SYSBACKUP               9.59
SYSKM                   9.60
SYSOPER                 9.61
SYSPRIVILEGE            9.62
SYSDG                   9.63
SYSRAC                  9.64
SYSBACKUP               9.65
SYSKM                   9.66
SYSOPER                 9.67
SYSPRIVILEGE            9.68
SYSDG                   9.69
SYSRAC                  9.70
SYSBACKUP               9.71
SYSKM                   9.72
SYSOPER                 9.73
SYSPRIVILEGE            9.74
SYSDG                   9.75
SYSRAC                  9.76
SYSBACKUP               9.77
SYSKM                   9.78
SYSOPER                 9.79
SYSPRIVILEGE            9.80
SYSDG                   9.81
SYSRAC                  9.82
SYSBACKUP               9.83
SYSKM                   9.84
SYSOPER                 9.85
SYSPRIVILEGE            9.86
SYSDG                   9.87
SYSRAC                  9.88
SYSBACKUP               9.89
SYSKM                   9.90
SYSOPER                 9.91
SYSPRIVILEGE            9.92
SYSDG                   9.93
SYSRAC                  9.94
SYSBACKUP               9.95
SYSKM                   9.96
SYSOPER                 9.97
SYSPRIVILEGE            9.98
SYSDG                   9.99
SYSRAC                  10.00
SYSBACKUP               10.01
SYSKM                   10.02
SYSOPER                 10.03
SYSPRIVILEGE            10.04
SYSDG                   10.05
SYSRAC                  10.06
SYSBACKUP               10.07
SYSKM                   10.08
SYSOPER                 10.09
SYSPRIVILEGE            10.10
SYSDG                   10.11
SYSRAC                  10.12
SYSBACKUP               10.13
SYSKM                   10.14
SYSOPER                 10.15
SYSPRIVILEGE            10.16
SYSDG                   10.17
SYSRAC                  10.18
SYSBACKUP               10.19
SYSKM                   10.20
SYSOPER                 10.21
SYSPRIVILEGE            10.22
SYSDG                   10.23
SYSRAC                  10.24
SYSBACKUP               10.25
SYSKM                   10.26
SYSOPER                 10.27
SYSPRIVILEGE            10.28
SYSDG                   10.29
SYSRAC                  10.30
SYSBACKUP               10.31
SYSKM                   10.32
SYSOPER                 10.33
SYSPRIVILEGE            10.34
SYSDG                   10.35
SYSRAC                  10.36
SYSBACKUP               10.37
SYSKM                   10.38
SYSOPER                 10.39
SYSPRIVILEGE            10.40
SYSDG                   10.41
SYSRAC                  10.42
SYSBACKUP               10.43
SYSKM                   10.44
SYSOPER                 10.45
SYSPRIVILEGE            10.46
SYSDG                   10.47
SYSRAC                  10.48
SYSBACKUP               10.49
SYSKM                   10.50
SYSOPER                 10.51
SYSPRIVILEGE            10.52
SYSDG                   10.53
SYSRAC                  10.54
SYSBACKUP               10.55
SYSKM                   10.56
SYSOPER                 10.57
SYSPRIVILEGE            10.58
SYSDG                   10.59
SYSRAC                  10.60
SYSBACKUP               10.61
SYSKM                   10.62
SYSOPER                 10.63
SYSPRIVILEGE            10.64
SYSDG                   10.65
SYSRAC                  10.66
SYSBACKUP               10.67
SYSKM                   10.68
SYSOPER                 10.69
SYSPRIVILEGE            10.70
SYSDG                   10.71
SYSRAC                  10.72
SYSBACKUP               10.73
SYSKM                   10.74
SYSOPER                 10.75
SYSPRIVILEGE            10.76
SYSDG                   10.77
SYSRAC                  10.78
SYSBACKUP               10.79
SYSKM                   10.80
SYSOPER                 10.81
SYSPRIVILEGE            10.82
SYSDG                   10.83
SYSRAC                  10.84
SYSBACKUP               10.85
SYSKM                   10.86
SYSOPER                 10.87
SYSPRIVILEGE            10.88
SYSDG                   10.89
SYSRAC                  10.90
SYSBACKUP               10.91
SYSKM                   10.92
SYSOPER                 10.93
SYSPRIVILEGE            10.94
SYSDG                   1
```



（1）使用SET命令设置查询结果的行数

在使用SQL*Plus查询数据时，经常是把所有的数据都显示在整个页面上，这就造成了页面显示数据非常乱，无法识别的问题。在SQL*Plus中提供对每页显示行数的设置。具体命令如下：

```
SET PAGESIZE n
```

这里，n代表每页显示的行数，默认在每页显示的行数是24。

除了设置每页显示的行数之外，还可以设置每页之间的空格数，这样就使每页之间都有间隔，更方便数据库管理员查看数据。具体的命令如下：

```
SET NEWPAGE n
```

这里，n代表每页之间间隔的空格数。

【示例12】设置查询结果显示的行数

下面就利用上面的两个命令完成查询结果的设置。从dba_users表中查询用户名(username)，设置每页显示5行，每页之间间隔1个空格。具体操作如图14.23所示。

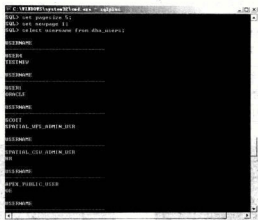


图14.23 设置查询结果显示效果

在此，可以看出每页显示的记录都是2而不是5，这是因为使用set pagesize 5设置的并不是一页中所显示数据表中的记录行数，而是还要包括每页显示的标题以及页之间的间隔。

如果在查询数据之前要查看一下当前在SQL*Plus中每页显示行数以及页之间的空格，可以通过下面的命令来完成：

```
SHOW PAGESIZE
SHOW NEWPAGE
```

【示例13】查询显示页的信息

下面利用上面的两个查看命令，查看当前SQL*Plus中PAGESIZE与NEWPAGE的值。查询结果如图14.24所示。



图14.24 显示页的信息



说明

如果将每页显示行数设置成0，那么就不会显示列标题；如果将每页之间的间隔设置成0，那么页与页会直接连上。

(2) 设置每行显示的字符数

在设置了每页显示行数的基础上，也可以对每行显示的字符数进行设置。具体设置每行显示字符数的命令如下：

```
SET LINESIZE n
```

这里，n代表每行显示的字符数，默认显示的字符数是80。

显示当前SQL*Plus中的LINESIZE，使用的命令如下：

```
SHOW LINESIZE
```

【示例14】设置每行显示字符数

下面利用LINESIZE的设置完成对查询结果的每行显示字符数的设置。从dba_users表中查询用户名 (username)、密码 (password)、用户ID (user_id)，并设置每行显示的字符数是100。查询结果如图14.25所示。

USERNAME	PASSWORD	USER_ID
SYSTEM	EXTERNAL	91
SYSTEM	EXTERNAL	92
SYSTEM	PASSWORD	93
SYSTEM	PASSWORD	94
SYSTEM	PASSWORD	95
SYSTEM	PASSWORD	96
SYSTEM	PASSWORD	97
SYSTEM	PASSWORD	98
SYSTEM	PASSWORD	99
SYSTEM	PASSWORD	100
SYSTEM	PASSWORD	101
SYSTEM	PASSWORD	102
SYSTEM	PASSWORD	103
SYSTEM	PASSWORD	104
SYSTEM	PASSWORD	105
SYSTEM	PASSWORD	106
SYSTEM	PASSWORD	107
SYSTEM	PASSWORD	108
SYSTEM	PASSWORD	109
SYSTEM	PASSWORD	110
SYSTEM	PASSWORD	111
SYSTEM	PASSWORD	112
SYSTEM	PASSWORD	113
SYSTEM	PASSWORD	114
SYSTEM	PASSWORD	115
SYSTEM	PASSWORD	116
SYSTEM	PASSWORD	117
SYSTEM	PASSWORD	118
SYSTEM	PASSWORD	119
SYSTEM	PASSWORD	120
SYSTEM	PASSWORD	121
SYSTEM	PASSWORD	122
SYSTEM	PASSWORD	123
SYSTEM	PASSWORD	124
SYSTEM	PASSWORD	125
SYSTEM	PASSWORD	126
SYSTEM	PASSWORD	127
SYSTEM	PASSWORD	128
SYSTEM	PASSWORD	129
SYSTEM	PASSWORD	130
SYSTEM	PASSWORD	131
SYSTEM	PASSWORD	132
SYSTEM	PASSWORD	133
SYSTEM	PASSWORD	134
SYSTEM	PASSWORD	135
SYSTEM	PASSWORD	136
SYSTEM	PASSWORD	137
SYSTEM	PASSWORD	138
SYSTEM	PASSWORD	139
SYSTEM	PASSWORD	140
SYSTEM	PASSWORD	141
SYSTEM	PASSWORD	142
SYSTEM	PASSWORD	143
SYSTEM	PASSWORD	144
SYSTEM	PASSWORD	145
SYSTEM	PASSWORD	146
SYSTEM	PASSWORD	147
SYSTEM	PASSWORD	148
SYSTEM	PASSWORD	149
SYSTEM	PASSWORD	150
SYSTEM	PASSWORD	151
SYSTEM	PASSWORD	152
SYSTEM	PASSWORD	153
SYSTEM	PASSWORD	154
SYSTEM	PASSWORD	155
SYSTEM	PASSWORD	156
SYSTEM	PASSWORD	157
SYSTEM	PASSWORD	158
SYSTEM	PASSWORD	159
SYSTEM	PASSWORD	160
SYSTEM	PASSWORD	161
SYSTEM	PASSWORD	162
SYSTEM	PASSWORD	163
SYSTEM	PASSWORD	164
SYSTEM	PASSWORD	165
SYSTEM	PASSWORD	166
SYSTEM	PASSWORD	167
SYSTEM	PASSWORD	168
SYSTEM	PASSWORD	169
SYSTEM	PASSWORD	170
SYSTEM	PASSWORD	171
SYSTEM	PASSWORD	172
SYSTEM	PASSWORD	173
SYSTEM	PASSWORD	174
SYSTEM	PASSWORD	175
SYSTEM	PASSWORD	176
SYSTEM	PASSWORD	177
SYSTEM	PASSWORD	178
SYSTEM	PASSWORD	179
SYSTEM	PASSWORD	180
SYSTEM	PASSWORD	181
SYSTEM	PASSWORD	182
SYSTEM	PASSWORD	183
SYSTEM	PASSWORD	184
SYSTEM	PASSWORD	185
SYSTEM	PASSWORD	186
SYSTEM	PASSWORD	187
SYSTEM	PASSWORD	188
SYSTEM	PASSWORD	189
SYSTEM	PASSWORD	190
SYSTEM	PASSWORD	191
SYSTEM	PASSWORD	192
SYSTEM	PASSWORD	193
SYSTEM	PASSWORD	194
SYSTEM	PASSWORD	195
SYSTEM	PASSWORD	196
SYSTEM	PASSWORD	197
SYSTEM	PASSWORD	198
SYSTEM	PASSWORD	199
SYSTEM	PASSWORD	200

图14.25 设置查询结果每行显示的字符数

(3) 显示查询数据所用的时间

在SQL*Plus中查询数据一般只能看到一共查询到多少条数据，在实际的应用中为了检验语句的性能，通常要知道每条语句执行的时间。在SQL*Plus中显示查询语句所用时间的命令如下：

```
SET TIMING ON/OFF
```

这里，设置成ON，则显示语句的执行时间；若设置成OFF，则取消显示语句的执行时间。

【示例15】显示语句执行时间

下面利用显示语句执行时间的设置完成对图14.25所示结果的设置，结果如图14.26所示。



图14.26 显示语句的执行时间

从图14.26中可以看出当前的查询语句执行的时间是39毫秒。

(4) 设置查询结果是否显示列标题

如果在保存查询结果时为了保证数据库的安全可以隐藏掉每列显示的标题。具体命令如下：

```
SET HEADING ON/OFF
```

这里，ON表示显示列标题；OFF表示不显示列标题。

【示例16】取消显示查询结果的标题

下面利用上面的命令完成取消显示查询结果列标题的操作。取消显示dba_user表中查询结果的列标题操作如图14.27所示。



图14.27 取消显示列标题

此时，显示的查询结果中就取消了列标题，但是还是保持了原有的格式输出。

(5) 设置查询结果是否显示“已选择行数”

在SQL*Plus中每一个查询结果后面都会默认有一个已选择行数的提示，如果要取消该提示可以使用下面的命令：

```
SET FEEDBACK ON/OFF
```

这里，ON代表显示“已选择行数”的提示；OFF代表不显示“已选择行数”的提示。

【示例17】设置不显示“已选择行数”

下面就以查询TEST表中的USERNAME为例，设置不显示“已选择行数”。操作结果如图14.28所示。



图14.28 不显示“已选择行数”



设置完不显示“已选择行数”后，执行完查询后并没有显示“已选择行数”，如果想设置显示“已选择行数”，操作如图14.29所示。

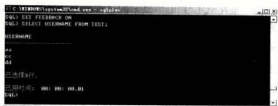


图14.29 显示“已选择行数”

3. 使用SPOOL命令输出查询结果

在SQL*Plus查询出的结果经常需要保存到文件中。保存结果的命令如下：

SPOOL filename

SPOOL OFF

【语法说明】

□ filename：保存输出结果的文件名，文件的扩展名可以写成.sql。

□ SPOOL OFF：当执行完SPOOL OFF之后，才把查询结果真正写入到指定的文件中。

【示例18】把查询结果写入文件

下面利用上面的语句完成把查询结果写入到文件test.sql中。把从dba_users表中的查询结果写入到test.sql文件中。写入操作如图14.30所示。

当完成查询后，执行SPOOL OFF即可完成把查询结果写入到test.sql文件中。操作如图14.31所示。



图14.30 创建文件test.sql存储查询结果



图14.31 写入查询结果

此时，查看文件test.sql，即可看到写入文件中的查询结果，如图14.32所示。



图14.32 查看文件中的查询结果

在此就可以看出在文件test.sql中不仅有查询结果还有查询的语句存在。由于这是数据库使用者必备的知识，所以希望读者自己多加练习。

在上面的练习中查询结果在SQL*Plus中显示的过程也是省略的，但是只是在执行语句的脚本时才会省略具体的执行过程。命令如下：

```
SET TERM ON/OFF
```

这里，ON是指SQL*Plus中显示查询结果，OFF是指SQL*Plus中不显示查询结果。

【示例19】隐藏查询结果，把结果写入到文件中

下面利用上面的命令隐藏查询结果直接把结果写入到文件中。操作步骤如下：

(1) 创建一个脚本文件

在脚本文件中输入查询语句，查询dba_users中用户名（username）、密码（password）、用户ID（user_id）。

创建好的脚本文件保存成test.sql，脚本文件如图14.33所示。



图14.33 test.sql脚本文件

(2) 执行脚本文件

在SQL*Plus中执行脚本文件可以使用@filename的方式。在执行脚本文件前使用set term off语句设置不在SQL*Plus中显示查询结果，操作结果如图14.34所示。此时，没有在图14.34中显示出脚本的查询结果。

为了对比不显示执行结果语句的方式，在执行了图14.34所示的文件之后，使用set term on设置显示查询结果后，再次执行脚本文件，结果如图14.35所示。



图14.34 执行脚本文件



图14.35 显示执行结果

(3) 查看文件TESTPOOL.sql

为了验证是否已经把test.sql中的查询结果写入到TESTPOOL.sql，打开TESTPOOL.sql，如图14.36所示。

至此，向TESTPOOL.sql文件中写入脚本文件test.sql的执行结果操作成功。

4. 使用TTITLE命令设置标题

在SQL*Plus中经常会用到的就是报表的制作，那么如何给报表设置标题呢？SQL*Plus为报表设置标题的命令如下：



TTITLE title

USERID	GLOBAL	91
TESTNEW		89
USER1		98
ORACLE		88
SCOTT		81
SPATIAL_MF1_ADMIN_USER		82
SPATIAL_CSV_ADMIN_USER		85

图14.36 查看TESTPOOL.sql

这里，title是指设置的标题名。注意，标题名要用单引号括起来。

【示例20】设置标题

下面使用TTITLE命令为查询结果设置标题。继续使用示例19中创建的脚本文件test.sql，在执行脚本文件之前先设置标题为“查询用户信息”。具体操作如图14.37所示。

图14.37 设置标题

图14.37中所查询的信息全部保存到TITLETEST文件中，TITLETEST文件的内容如图14.38所示。

USERID	PASSWORD	USER_ID
USER1	GLOBAL	91
TESTNEW		89

图14.38 TITLETEST文件

在图14.38中就可以看到“查询用户信息”的标题显示在文件中。

除了使用TTITLE添加标题之外，还可以使用BTITLE定义尾标题。这里就不举例说明了，请读者对上述文件自行添加尾标题。

14.2.5 在SQL*Plus中为语句添加注释

注释是每一个数据库管理员经常使用的工具，如果在语句中没有注释，那么其他人就无法读懂你写的语句，甚至自己写的语句过了很长时间之后自己也不知道当初为什么这么写。在SQL*Plus中写语句也是一样的，前面已经学习过把执行的语句保存起来，但是如果语句都没有说明，以后也很难再使用。SQL*Plus也为数据库使用者提供了在语句中加入注释的方法。常用的方法有以下几种。



1. 使用/*.....*/方法

这种在语句中加入注释的方法在很多编程语言中都是经常使用的，在/* 和*/之间的内容就是语句的注释内容。注释可以写在语句中的任何位置，在编译语句时是不会编译注释的。下面就以“/*.....*/”这种注释方式为脚本文件test.sql添加注释，如图14.39所示。

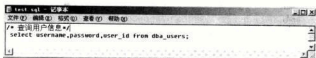


图14.39 添加注释

在执行添加注释后的脚本时执行效果是不会改变的，请读者自己尝试一下。

2. 使用REMARK命令

使用REMARK命令进行注释。REMARK命令的作用就是告诉SQL*Plus接下来的文字不是命令而是注释。具体命令如下：

```
REMARK comment
```

这里，comment就是要添加的注释信息。

【示例21】添加注释

下面就利用REMARK命令在SQL*Plus中添加注释。仍然是写一个查询dba_users表中信息的语句，然后在语句前面加上“查询用户信息”的注释。具体操作如图14.40所示。



图14.40 使用REMARK命令添加注释

在此，“查询用户信息”的注释就被加入到REMARKTEST.SQL文件。查看REMARKTEST.SQL文件的内容，如图14.41所示。



图14.41 REMARKTEST.SQL文件内容

如果要想把加注释的语句写入到文件中，一定要在SPOOL命令之后输入注释，因为在**注意** 调用SPOOL之后，后面的语句才是向文件中写入的语句。但是在语句最后还要加上SPOOL OFF才算真正把语句写入文件中。



14.3 使用PL/SQL Developer

Oracle SQL Developer是Oracle提供的免费图形化开发工具，TOAD和PL/SQL Developer是商业性的工具，需要付费，但使用的人比较多。相对来说，PL/SQL Developer更容易上手，比较适合初学者，笔者曾经工作过的几家公司都使用PL/SQL Developer，它是一种集成的开发环境，专门用于开发、测试、调试和优化Oracle PL/SQL存储程序单元。在前面的章节中已经使用过PL/SQL Developer工具操作数据库，本节主要讲述PL/SQL Developer的安装与布局。

14.3.1 PL/SQL Developer的安装

安装并使用PL/SQL Developer的先决条件是在本机上有Oracle客户端或相当于客户端的其他软件。下面就安装该软件并连接本地的Oracle数据库，主要分为5个步骤。

- 1) 安装Oracle的客户端。安装Oracle的客户端在第2章中已经详细介绍了，这里就不再赘述。
- 2) 下载软件。如果没有该软件，可以到<http://www.allroundautomations.com/bodydownloads.html#PLS>下载。下载页面如图14.42所示。



图14.42 PL/SQL Developer下载页面

单击黑框标示出来的链接，会弹出下载窗口。当前最新版本是8.04，下载的软件有30天的免费试用期。软件下载完成后，在下载目录中可以看到一个以“.exe”结尾的文件，如图14.43所示。这里用的是8.02版本。

3) 双击运行该安装文件。运行该安装文件后，经过解压，会出现图14.44所示的协议界面。在该界面的协议中提示有30天的试用期。

单击 **[I Agree]** 按钮，继续安装，选择自定义路径后，按照提示一直默认就可以安装成功。安装成功后，会在桌面创建一个快捷方式。

4) 输入登录信息。双击运行该快捷方式，出现图14.45所示登录界面。界面上各项表示含义如下：

- ☐ Username: Oracle的用户名。



图14.43 PL/SQL Developer下载程序



图14.44 协议界面

- ☐ Password: 用户对应的密码。
- ☐ Database: 这里的列表实际上是客户端tnsnames.ora文件中对Oracle的配置名称。“tnsnames.ora”所在目录是“xxx\app\ Administrator\product\ 11.1.0\ client_1\network\admin”。
- ☐ Connect as: 有3项, 这里使用普通身份登录。

5) 登录PL/SQL Developer。登录信息设置完成后, 单击【OK】按钮, 登录进入PL/SQL Developer, 见图14.46, 此时的PL/SQL Developer已经连上了Oracle。



图14.45 登录界面

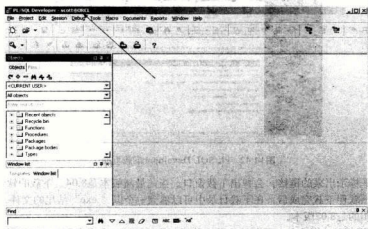


图14.46 登录进入PL/SQL Developer

由箭头标出部分可以看出此时已经登录到数据库中。

14.3.2 PL/SQL Developer的布局

通过14.3.1小节的讲述, 已经可以连接到Oracle数据库了。为了更方便地使用PL/SQL Developer, 还需要了解PL/SQL Developer的页面布局, 如图14.47所示。

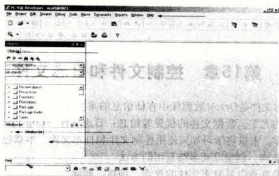


图14.47 PL/SQL Developer布局

按照从上到下的顺序，有4个箭头，那么所指的地方分别表示：

- 1) 工具栏。所有的操作都可以从这里找到。
- 2) 对象列表。以图形的方式列出了指定范围的表、视图、函数、存储过程、触发器等。
- 3) 模板列表。包含很多函数的语法、语句结构的语法等。
- 4) 窗口列表。打开的编辑页面都列在这个地方。

14.4 小结

SQL*Plus是一个比较常用的Oracle数据库操作工具，本书中的实例大部分也是使用SQL*Plus完成的。通过本章的学习，读者可以熟练地使用SQL*Plus的编辑命令、保存命令、运行命令、格式化命令以及在SQL*Plus中执行的语句或者文件添加注释。另外，PL/SQL Developer也是一个比较常用的工具，在实际应用中也是比较广泛的。希望读者能够熟练使用这两款数据库工具。

14.5 习题

简答题

1. 如何启动SQL*Plus?
2. 分别写出替换文本的命令和追加文本的命令。以查询dba_users为例，分别使用这两个命令执行不同的查询功能。
3. 举例说明追加文本和添加行这两个命令的区别。
4. 在SQL*Plus中执行查询dba_users表，并把执行命令的语句保存到test.sql文件中。
5. 写出运行命令RUN与START的区别。
6. 对dba_users表的查询结果完成下列格式化处理（这里只查询用户名、密码以及用户ID）：
 - 1) 设置显示时的列标题为中文标题；
 - 2) 设置每页显示10行；
 - 3) 设置显示查询执行的时间；
 - 4) 设置不显示“已选择行数”；
 - 5) 设置每页显示标题“查询用户信息表”。
7. 把第6题中设置的结果保存到result.sql文件中。

第15章 控制文件和日志文件

控制文件和日志文件是Oracle数据库中存储信息的重要文件。控制文件（Control File）主要用来存放数据库的名字、数据文件的位置等信息；日志文件（Log File）主要用来存放数据库中数据变化的操作。本章将学习如何使用控制文件和日志文件。本章包括以下知识点：

- 控制文件的查询、创建以及多路复用的方法
- 日志文件的检查点以及日志文件的查询、创建、删除

本章内容基本涵盖了控制文件和日志文件的操作方法。通过本章的学习，可以熟练地管理和使用控制文件与日志文件。

15.1 控制文件与日志文件概述

控制文件和日志文件是数据库中两个主要的文件，没有控制文件数据库就无法启动，没有日志文件数据库的信息就无法完全恢复。本节将学习什么是控制文件和日志文件。

15.1.1 什么是控制文件

控制文件是数据库中的一个二进制文件，它主要用来记录数据库的名字、数据库的数据文件存放的位置等信息。因此，有人也把控制文件比做数据库的心脏，那么心脏如果丢失或者损坏了，数据库将不复存在。对于控制文件来说，保护是至关重要的。

每个数据库都存在控制文件，但是一个控制文件只属于一个数据库。这就像每个人都有身份证，但是一个身份证只属于一个人。控制文件在创建数据库时自动被创建，当数据库的信息发生改变时，控制文件也随之被改变；控制文件不能手动修改，只能由Oracle数据库本身自己来修改。控制文件在数据库启动和关闭时都要使用，如果没有控制文件，数据库将无法工作。

那么，控制文件究竟是什么样的呢？使用下面的语句就可在数据字典中查看控制文件的信息：

```
desc v$controlfile
```

【示例1】查询控制文件数据字典中的描述

查看数据字典的结果如图15.1所示。



图15.1 数据字典v\$controlfile



【示例4】查询数据库中日志状态
查询当前所有数据库的日志状态结果如图15.4所示。



图15.4 查询数据库的日志状态

说明 日志的状态NOARCHIVELOG指的是非归档模式，而ARCHIVELOG指的是归档模式。

15.2 初识控制文件

什么是控制文件在15.1节中已经介绍过了，在本节中主要讲解控制文件中包含哪些内容，以及如何更新控制文件。

15.2.1 控制文件的内容

控制文件的存放位置和状态可以从数据字典v\$controlfile中查询，查询方法有两种，一种是在SQL*Plus下查询；另一种是在企业管理器中查询。

【示例5】在SQL*Plus中查询控制文件信息

在SQL*Plus中查询控制文件信息，语句如下：

```
select name,status from v$controlfile;
```

查询结果如图15.5所示。



图15.5 在SQL*Plus中查询控制文件的信息

【示例6】在企业管理器中查询控制文件信息

在OEM（企业管理器）中直接查看控制文件的信息，查询方法是在企业管理器的页面中单击【服务器】，在服务器信息页面的存储栏下选择【控制文件】，显示结果如图15.6所示。在界面上选择【高级】选项，则显示如图15.7所示控制文件的信息。

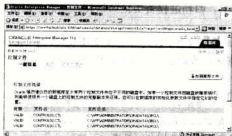


图15.6 在OEM中查询控制文件的一般信息



图15.7 在OEM中查询控制文件的详细信息

从上面的查询结果中可以看出数据库的控制文件的扩展名是ctl。在v\$sqlcontrolfile中status列一般都为空。每一个控制文件中都记录数据库的名称、创建数据库的时间、数据文件的名字及数据库存储的位置、重做日志文件的名字及位置、归档日志的信息、表空间信息、日志历史记录、备份信息、当前的日志序列号、最近检查点的信息等数据库使用信息。这些信息都是在操作数据库时自动写入到控制文件中的，而不是手动写入的。例如，在创建一个数据库时，控制文件中就会自动写入数据库的名称、数据库的创建时间等信息；在增加或者删除表空间时，表空间的信息也会自动写入控制文件中。

控制文件的大小取决于创建数据库时所提供的参数信息，因此当添加数据库中的文件时，控制文件的大小保持不变。参数信息如表15.1所示。

表15.1 创建数据库参数说明

参 数 名	说 明
MAXLOGFILES	指定数据库可建的日志文件组的最大值，Oracle利用这个值决定在控制文件中使用的空间大小，该空间用于存储日志文件名。其默认值、最大值和最小值的变化依赖于OS
MAXLOGMEMBERS	指定日志文件组的最多的成员数。Oracle使用该值决定在控制文件中分配给日志文件名的空间大小。最小值为1，最大值和默认值是变化的，依赖于OS
MAXLOGHISTORY	指定归档日志文件的最大数目，用于并行服务器选项的介质自动恢复。Oracle利用该值决定在控制文件中分配给归档日志文件名的空间大小。默认值为MAXINSTANCES值的倍数，是可变的，决定于OS。最大值受控制文件的大小所限制
MAXDATAFILES	指定数据库可建立的最大数据文件数，其最小值为1，最大值和默认值决定于OS。实例可存取的数据文件的数目受初始参数DB_FILES的限制
MAXINSTANCES	指定可同时装配或打开该数据库的实例的最大数目。这个值优先于初始化参数INSTANCES。最小值为1，最大值和默认值依赖于OS

15.2.2 更新控制文件

当增加、重命名、删除一个数据文件时，Oracle服务器进程（Server Process）会立即更新控制文件以反映数据库结构的这种变化。每次在数据库的结构发生变化后，为了防止数据丢失都要备份控制文件。按照各进程分工不同分别把数据库的更改后信息写入到控制文件中：日志写入进程（LGWR）负责把当前日志序列号记录到控制文件中；校验点进程



(CKPT) 负责把校验点的信息记录到控制文件中, 归档进程负责把归档日志的信息记录到控制文件中。

通常情况下, 数据库管理员会使用镜像来管理控制文件, 把每个控制文件分布到不同的物理磁盘, 发生灾难时, 即使其中一个控制文件损坏, 数据也不会丢失, 也不会使整个数据库陷于瘫痪。

当打开数据库提示“文件比控制文件更新—旧的控制文件”, 此时的问题就是数据文件里面的控制文件序列号比控制文件的高, 那么这个问题的出现可能是更改控制文件的位置所造成的。补救的方法可以用数据库中的日志文件重新创建一下控制文件。

15.3 控制文件的多路复用

控制文件在数据库中的作用是不可比拟的。保护控制文件实际就是在保护数据库。Oracle 的多路复用的特性就可以帮助数据库管理员保护好控制文件。多路复用的特性可以把控制文件的副本创建到不同的磁盘上, 这样即使一个磁盘发生故障, Oracle 仍然可以从其他磁盘上恢复, 从而达到保护数据库的作用。

15.3.1 使用init.ora多路复用控制文件

在图15.7中的提示里面就可以看到Oracle给予提示可以在数据库初始化文件中修改控制文件存放的位置, init.ora就是数据库初始化文件, 它是在创建数据库时自动创建的。要修改init.ora, 首先要找到它的存放位置, 这个文件存储在安装文件目录下的admin\orcl\pfile里面。

在修改init.ora文件之前, 通过复制把控制文件复制到不同的位置, 然后再修改init.ora中的control_files参数。init.ora中参数control_files的形式如图15.8所示。

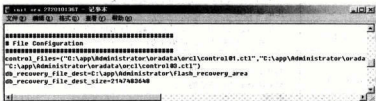


图15.8 init.ora中control_files参数

按照现有的control_files形式进行修改, 在修改时需要注意的是每一个控制文件之间是通过逗号分隔的, 并且每一个控制文件都是用双引号括起来的。另外, 在进行控制文件路径修改之前, 最好把控制文件复制一份保存, 以免数据库无法启动。

15.3.2 使用SPFILE多路复用控制文件

使用SPFILE方式实现多路复用控制文件的原理和修改init.ora初始化参数的文件中control_files参数是一样的, 只是修改方法不同。只需要在SQL*Plus中写语句就可以完成control_files参数的修改。



【示例7】使用SPFILE多路复用控制文件

具体步骤如下：

(1) 修改control_files参数

在确保数据库是打开状态时，使用命令修改control_files参数。语法如下：

```
Alter system set control_files='文件的路径1',
                             '文件的路径2'...,
                             '文件的路径n' scope=spfile;
```

在control_files中增加一个控制文件，结果如图15.9所示。

(2) 关闭数据库

在数据库打开时，数据库中的任何文件都是无法操作的。关闭数据库的命令如下：

```
shutdown immediate;
```

运行结果如图15.10所示。



图15.9 增加一个控制文件



图15.10 关闭数据库

(3) 在DOS下复制文件到指定位置

在DOS窗口下使用复制命令在指定位置增加一个控制文件。复制命令语法如下：

```
copy 旧文件, 新文件
```

使用复制命令复制的结果如图15.11所示。



图15.11 复制文件

(4) 启动数据库实例并验证

在文件复制完成后，使用startup重新启动数据库的实例。在数据库字典controlfile中重新查询现存的控制文件，查询结果如图15.12所示。



图15.12 查询控制文件

注意

在Oracle中的控制文件个数不能少于2个；复制文件时一定要使用命令复制，否则数据库实例无法重新启动。

15.4 创建控制文件

控制文件在数据库中的作用前面已经讲过了，但是无论怎么保护都有可能控制文件丢失和损坏的问题。本节将学习创建控制文件。

控制文件在创建数据库时就自动创建了，并且在配置文件init.ora中已经记录了文件的路径。前面已经学习过控制文件记录的内容以及控制文件的重要性，控制文件一般情况下是不需要手工创建的。只有在两种情况下才考虑用手工的方式来创建控制文件，一种是当控制文件全部损坏，无法恢复时；一种是需要永久地修改数据库的参数设置时，因为手工创建控制文件会出现一些无法预计的问题，只能说是一种补救办法。

【示例8】手工创建控制文件

手工创建控制文件分为找原有的数据文件和重做日志的路径、关闭数据库、创建新的控制文件、修改init.ora中controlfiles参数、验证控制文件5个步骤。

(1) 找原有的数据文件和重做日志的路径

数据文件和重做日志信息可以通过两种途径获取，一种方法是当控制文件没有损坏时，从控制文件中直接获取；一种方法是控制文件损坏了，从数据字典v\$datafile中获取数据文件的信息，从数据字典v\$logfile中获取日志文件的信息。前面的两种途径都是在数据库能够正常启动时使用的方法，如果数据库不能够正常启动，那么此时就需要根据系统提供的错误信息来查找原因。在创建新的控制文件并且使用它打开数据库之后，Oracle会对数据字典和控制文件的内容进行检查。如果发现数据字典包含了某个数据文件而控制文件中没有列出这个数据文件，Oracle数据库就会报错。数据库管理员就只能凭借这些信息来判断是否缺少必要的数据文件，一步一步查找直到找到真正的数据文件。

本例是在数据库能正常启动的情况下，通过datafile和logfile数据字典分别获得数据文件列表和日志文件列表，结果如图15.13所示。

```

SQL> select name from v$datafile;

NAME
-----
C:\APP\ADMINISTRATOR\ORADATA\ORCL\SYSTEM1.DBF
C:\APP\ADMINISTRATOR\ORADATA\ORCL\SYSAUX1.DBF
C:\APP\ADMINISTRATOR\ORADATA\ORCL\UNDOT101.DBF
C:\APP\ADMINISTRATOR\ORADATA\ORCL\USERS01.DBF
C:\APP\ADMINISTRATOR\ORADATA\ORCL\EXAMPLE1.DBF
C:\APP\ADMINISTRATOR\ORADATA\ORCL\STMT01.DBF

SQL> select member from v$logfile;

MEMBER
-----
C:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO01.LOG
C:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO02.LOG
C:\APP\ADMINISTRATOR\ORADATA\ORCL\REDO03.LOG
  
```

图15.13 查询datafile和logfile



(2) 关闭数据库

在创建控制文件之前要关闭数据库，最好是正常关闭数据库。采用正常模式关闭，可以减少数据库重新启动过程中可能出现的问题。为保证数据库的安全，在关闭数据库后，应该把数据库的日志文件、数据文件、参数文件等备份到其他磁盘上。关闭数据库使用如下语句：

```
shutdown immediate;
```

(3) 创建新的控制文件

在创建新的控制文件之前，最好把原来的控制文件备份到其他位置，这样创建的效果比较明显。此外，还要启动一个数据库的实例，这个数据库实例是安装数据库时自带的。启动实例过程如图15.14所示。

启动实例后就可以创建控制文件了，在创建控制文件时就需要用到日志文件和数据文件的列表。创建控制文件的语句如下：

```
create controlfile
reuse database "数据库实例名" noresetlogs//是否重做日志或重命名数据库 noarchivelog//归档状态
maxlogfiles //最大日志文件大小
maxlogmembers //日志文件组的成员数
maxinstances //最大实例的个数
maxloghistory //最大历史日志文件个数
logfile //日志文件
group1 '日志文件的路径1' size 日志文件的大小,
...
Groupn '日志文件的路径n' size 日志文件的大小
datafile //数据文件
'路径1',
...
'路径n'
Character set we8dec
```

创建控制文件的结果如图15.15所示。



图15.14 启动nomount实例

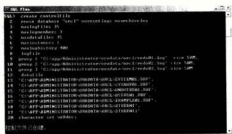


图15.15 创建控制文件

在创建控制文件时如果不需要重做日志文件和重命名数据库，就使用noresetlogs，否则就使用resetlogs；在创建控制文件时也可以设置归档状态，如果设置为noarchivelog，则指非归档状态，若设置为archivelog，则是归档状态。



(4) 修改init.ora中controlfiles参数

使用前面讲过的spfile方法，修改init.ora中的参数controlfiles。

(5) 验证控制文件

重启数据库后，查询v\$controlfile数据字典，检查控制文件是否全部正确加载。如果数据库启动不了，可以重启数据库服务。本例验证结果如图15.16所示。



图15.16 验证控制文件

至此，控制文件创建成功。

15.5 日志文件的管理

前面已经学习过日志文件的概念以及日志文件的分类，日志文件全部存放在日志文件组中。本节将讲述如何创建、删除、查询日志文件组，以及如何添加日志文件到日志文件组。

15.5.1 新建日志文件组

创建日志文件组是每一个数据库管理员必须要掌握的技能，日志文件组能够帮助数据库管理员管理日志文件。创建日志文件组可以使用企业管理器创建，也可以使用语句在SQL*Plus中创建。下面分别采用这两种方式创建日志文件组。

1. 使用企业管理器创建日志文件组

【示例9】在企业管理器中创建日志文件组

在企业管理器中创建日志文件组是很容易的，通过下面的几个步骤即可完成。

(1) 找到重做日志文件组页面

在企业管理器中创建日志文件组是在企业管理器的服务器选项页面中，如图15.17所示。



图15.17 企业管理器中服务器选项页面

在此页面中选择“存储”栏中的【重做日志组】选项，进入到图15.18所示页面。



图15.18 重做日志文件组页面

(2) 创建重做日志文件组

在图15.18所示页面中，在【操作】下拉列表框中，选择类似创建选项，然后单击【开始】按钮，如图15.19所示。



图15.19 创建重做日志组

在图15.19所示页面中修改组号和文件大小。由于每一个日志文件只能属于一个重做日志组，所以要修改重做日志的成员。单击【编辑】按钮，进入图15.20所示的页面。

在此，修改好文件名和目录之后，单击【继续】按钮，日志文件组成员修改完成。

(3) 验证添加的重做日志组

在修改完成之后重组日志组主页面就会出现图15.21所示的页面，显示已成功创建对象。

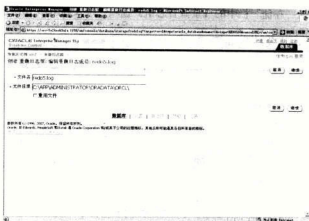


图15.20 修改日志成员

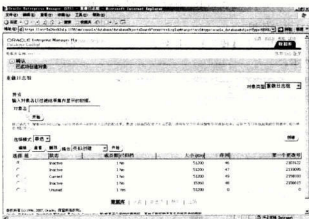


图15.21 对象创建成功

至此，重做日志组创建成功。

2. 在SQL*Plus中创建重做日志组

在SQL*Plus中创建重做日志组需要使用修改数据库的语句来完成。具体语法如下：

```
ALTER DATABASE [database_name]
ADD LOGFILE GROUP n
filename SIZE m
```

【语法说明】

- database_name：要修改的数据库名，省略该项表示当前的数据库。
- n：创建重组日志组的组号，组号在重做日志组中都是唯一的。

□ filename: 日志文件组存储的位置。

□ m: 日志文件组的大小, 默认的大小是50MB。

【示例10】使用语句在SQL*Plus中创建重做日志文件组

下面利用上面的语句完成重做日志组的创建。创建的日志文件组的组号是8, 添加的数据文件名为NEWLOG8, 大小是15MB。具体操作如图15.22所示。



图15.22 创建重做日志文件组

由此可以看出已经在数据库中创建了新的重做日志文件组。

15.5.2 添加日志文件到日志文件组

日志文件组创建完成后, 一个日志文件组中可以包含多个日志文件, 至少含有一个日志文件。向已经存在的日志文件组中添加日志文件也可以在企业管理器和SQL*Plus中完成。下面用这两种方式分别添加日志文件。

1. 使用企业管理器添加日志文件

【示例11】在企业管理器中添加日志文件

在企业管理器中添加日志文件到文件组是很容易的, 只要在图15.18所示的页面中, 选择一个要添加日志文件的组, 即可进入图15.23所示页面。

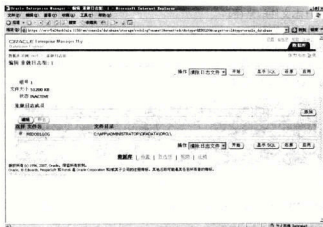


图15.23 编辑重做日志组

在此, 单击【添加】按钮, 进入图15.24所示页面。

添加后, 单击【继续】按钮, 添加成功。



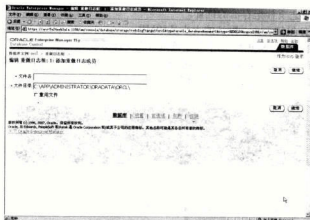


图15.24 添加重做日志成员

2. 在SQL*Plus中添加日志文件

在SQL*Plus中添加日志文件的语法和添加日志文件组的语法类似。具体语法如下：

```
ALTER DATABASE [database_name]
ADD LOGFILE MEMBER
filename TO GROUP n
```

【语法说明】

- ☐ database_name：数据库的名称，默认是当前的数据库。
- ☐ filename：日志文件的路径。
- ☐ n：日志文件填入的组号。

【示例12】使用语句在SQL*Plus下添加日志文件

下面利用上面的语句完成添加日志文件到日志文件组1中。添加的日志文件名为NEWLOG10.log。具体操作如图15.25所示。



图15.25 添加日志文件

至此，日志文件NEWLOG10.log就添加到日志文件组1中，读者可自行查询验证。

15.5.3 删除日志文件组和日志文件

无论删除日志文件组还是删除日志文件都可以在企业管理器和SQL*Plus中完成。下面分别在这两个环境中完成删除日志文件组和日志文件的操作。

1. 使用企业管理器删除日志文件组和日志文件

【示例13】在企业管理器中删除日志文件组

删除日志文件组时无论日志文件组中是否存在日志文件都可以一并删除，只需要在文件组列表表中选择一个文件组，再单击【删除】按钮，即可出现图15.26所示页面。

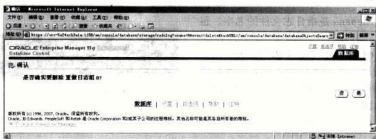


图15.26 删除日志文件组确认页面

单击【是】按钮后，即可成功删除日志文件组。

【示例14】在企业管理器中删除日志文件

删除日志文件时首先要找到日志文件所在的日志文件组。如图15.23所示，在页面中选择一个日志文件，然后单击【移去】按钮，即可移除所选的日志文件。

注意

当日志文件组中只有一个日志文件时是不能够删除的，如果要删除，只能把整个日志文件组全部删除。

2. 使用SQL*Plus删除日志文件组和日志文件

(1) 删除日志文件组

删除日志文件组使用的也是修改数据库的语句。具体语法如下：

```
ALTER DATABASE [database_name]
DROP LOGFILE
GROUP n
```

这里，n代表的是日志文件组的组号。

【示例15】在SQL*Plus中删除日志文件组

下面利用上面的语句完成日志文件组的删除操作。删除组号是4的日志文件组，具体操作如图15.27所示。

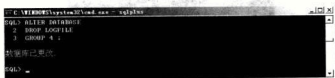


图15.27 删除日志文件组



(2) 删除日志文件

删除日志文件与删除日志文件组的语法类似。具体语法如下：

```
ALTER DATABASE [database_name] DROP LOGFILE MEMBER filename
```

这里，filename是指日志文件的名字，包括路径。

【示例16】在SQL*Plus中删除日志文件

下面利用上面的语句完成日志文件删除的操作。删除名为NEWLOG11的日志文件，具体操作如图15.28所示。

至此，日志文件删除成功。



图15.28 删除日志文件

15.5.4 查询日志文件组和日志文件

查询日志文件组和日志文件可以直接在企业管理器中查看，本小节主要讲述使用语句在SQL*Plus中查看。下面分别讲解在SQL*Plus中查询日志文件组和日志文件。

1. 查询日志文件组

查询日志文件组可以在v\$LOG的视图中完成。

【示例17】查询日志文件组

下面查询v\$LOG中的组号（GROUP#）、成员数（MEMBER）以及状态（STATUS）的信息，结果如图15.29所示。



图15.29 查询日志文件组

2. 查询日志文件

查询日志文件可以在v\$LOGFILE中查看。

【示例18】查询日志文件

下面查询v\$LOGFILE中的组号（GROUP#）、成员（MEMBER）的信息，结果如图15.30所示。

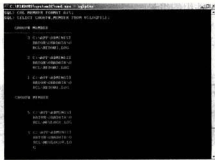


图15.30 查询日志文件

15.6 小结

本章首先讲述了数据库中控制文件和日志文件对于数据库的重要性，并解释了日志文件和控制文件的概念，然后讲解了如何在企业管理器中使用向导操作控制文件和日志文件，同时讲解了如何使用语句在SQL*Plus中创建、修改以及删除日志文件组和日志文件的操作。通过本章的学习，读者能够掌握对数据库中控制文件和日志文件的使用。



15.7 习题

简答题

1. 在企业管理器中查看当前数据库中的控制文件。
2. 使用语句在SQL*Plus中创建控制文件，并验证是否已经创建成功。
3. 什么是控制文件的多路复用？
4. 在SQL*Plus中创建日志文件以及日志文件组，并验证是否已经创建成功。

第16章 表空间的管理

表空间是使用Oracle数据库必须具备的知识，在Oracle中创建数据库的同时就需要指定数据库建立的表空间。本章将介绍如何管理表空间以及如何管理数据文件等内容。本章包括以下知识点：

- 与表空间有关的概念以及表空间的分类
- 如何管理表空间
- 如何管理临时表空间
- 如何管理数据文件

本章内容基本涵盖了表空间管理的相关知识。通过本章的学习，可以熟练地在创建数据库时指定表空间和大文件表空间。

16.1 表空间概述

说到Oracle数据库，不得不提的就是表空间这个名词。本节将介绍几个与表空间有关的概念以及Oracle 11g中的默认表空间。

16.1.1 相关概念

在Oracle中表空间和数据文件的概念经常是成对出现的，每一个数据文件只对应一个表空间，一个表空间可以存放多个数据文件。在创建表空间的同时必须创建数据文件，同理，如果要创建数据文件必须要指定表空间。

一个Oracle数据库是由一个或多个表空间组成的，在表空间中可以存储数据文件。这些数据文件也不是任意格式的，也要按照Oracle运行的操作系统的物理结构。数据文件中存放的就是要存放在数据库中的数据。在表空间中的逻辑存储单位是段（segment）。例如，我们为表创建一个索引，那么就会在这个段中又创建一个区，这个区就叫做区段（extent），也叫数据扩展，每一个区段只能存在于一个数据文件中。区段再进一步划分还有区块（block）。但是，一个文件在磁盘上存储一般都是不连续的，所以，在表空间中的段要由不同数据文件中的区段组成。块是Oracle数据库中最小的空间分配单位。Oracle中常见的块大小是2、4、8、16KB。

16.1.2 默认表空间

在Oracle 11g数据库存在6个默认表空间。查看默认表空间的方法主要有两种方式，一种是通过企业管理器直接查看；另一种是在数据字典中查看。下面分别用这两种方式查看默认表空间。

1. 使用企业管理器查看默认表空间

(1) 查看所有的默认表空间

进入Oracle 11g的企业管理器后，选择【服务器】选项，如图16.1所示。



图16.1 【服务器】选项卡

在图16.1中,选择【表空间】选项,其中就列出了所有的默认表空间,如图16.2所示。

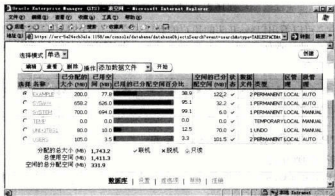


图16.2 默认表空间

从图16.2中可以看出,在Oracle 11g中默认的表空间有6个,分别是EXAMPLE、SYSAUX、SYSTEM、TEMP、UNDOTBS1、USERS。其中:

- ☐ EXAMPLE表空间:用于安装Oracle 11g数据库使用示例数据库。
- ☐ SYSAUX表空间:作为EXAMPLE的辅助表空间。
- ☐ SYSTEM表空间:用来存储SYS用户的表、视图以及存储过程等数据库对象。
- ☐ TEMP表空间:用于存储SQL语句处理的表和索引的信息。
- ☐ UNDOTBS1表空间:用于存储撤销信息。
- ☐ USERS表空间:存储数据库用户创建的数据库对象。

(2) 查看指定用户的默认表空间

如果要查询当前登录的用户所使用的表空间,在页面上直接输入对象名即可以查看到。下

而以sys开头的用户（除了sys还包括system、sysman）为例查看该用户的默认表空间，如图16.3所示。



图16.3 sys用户的默认表空间

从图16.3中，可以看出以sys开头的用户的默认表空间有SYSAUX和SYSTEM两个。每一个用户所使用表空间可能有所不同，如果想查看当前用户所使用的表空间，可以通过输入相应的用户名进行查看。

说明 如果要查看某一个表空间中存放的数据文件具体信息，可以直接单击相应的表空间名称进行查看。

2. 在SQL*Plus下查看登录用户的默认表空间

(1) 查看所有的默认表空间

使用SQL*Plus登录Oracle 11g数据库，在数据字典DBA_TABLESPACES中查看所有默认的表空间。查询结果如图16.4所示。



图16.4 查看DBA_TABLESPACES

从图16.4所示的结果中可以看出，这和使用企业管理器查看默认表空间的结果是一致的。

(2) 查看指定用户的默认表空间

如果要查看某一个用户的默认表空间，可以通过DBA_USERS数据字典进行查询。下面分



别从DBA_USERS数据字典中查看SYS、SYSTEM、SYSMAN三个用户的默认表空间。查看结果如图16.5所示。



图16.5 查看以sys开头用户的默认表空间

其中，SYSTEM和SYS用户的默认表空间是SYSTEM，而SYSMAN用户的默认表空间是SYS_AUX。

说明

如果要在SQL*Plus中查看表空间的使用情况，可以使用数据字典dba_free_space查询；还可以从dba_data_files数据字典中查看表空间中数据文件的信息。

16.2 表空间的管理

表空间的管理无非就是对表空间进行创建、修改、删除等操作，通过本节的学习将学会如何创建表空间、建立大文件表空间、修改表空间的状态以及删除表空间的操作。

16.2.1 创建表空间

创建表空间是每一个使用Oracle 11g数据库的人员经常要做的工作，创建表空间也可以通过企业管理器和SQL*Plus两种方式。

1. 使用企业管理器创建表空间

【示例1】在企业管理器中创建表空间

在企业管理器的查看默认表空间的界面（图16.2所示）上，单击【创建】按钮，出现创建表空间的界面，如图16.6所示。

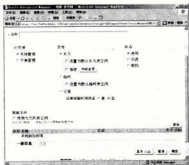


图16.6 创建表空间



在图16.6所示的界面中，在【名称】文本框中填入表空间的名称EMTEST，选择相应的区管理方式、表空间的类型以及表空间的状态。这里全部使用默认选项，然后单击【添加】按钮为表空间添加数据文件。添加数据文件的界面如图16.7所示。

在图16.7所示的界面中填入数据文件的名称，并且选中【数据文件满后自动扩展】复选框，【增量】输入128，如图16.8所示。



图16.7 为EMTEST表空间添加数据文件



图16.8 添加数据文件详细信息

单击【继续】按钮，回到添加表空间的界面即图16.6所示，再单击【确定】按钮即完成了表空间EMTEST的创建。

2. 在SQL*Plus中创建表空间

虽然在企业管理器中创建表空间很简单，但是在实际工作中数据库管理员还是使用语句创建表空间的比较普遍。下面就讲解如何在SQL*Plus中使用语句创建表空间。创建表空间的一般语法如下：

```
CREATE TABLESPACE tablespace_name
DATAFILE filename SIZE size
[AUTOEXTEND {ON/OFF}] NEXT size
[MAXSIZE size]
[PERMANENT|TEMPORARY]
[EXTENT MANAGEMENT
{DICTIONARY | LOCAL
[AUTOALLOCATE | UNIFORM. {SIZE integer[K|M] } ] ]
```

【语法说明】

- TABLESPACE：指定要创建的表空间的名称。
 - DATAFILE：指定在表空间中存放数据文件的文件名，这里还要指出文件存放的路径。
 - SIZE：指定数据文件的大小。
 - AUTOEXTEND：指定数据文件的扩展方式，ON代表自动扩展，OFF代表非自动扩展。
- 另外，如果要把数据文件指定为自动扩展，应该在NEXT后面指定具体的大小。
- MAXSIZE：指定数据文件为自动扩展方式时的最大值。
 - PERMANENT|TEMPORARY：指定表空间的类型，PERMANENT是指永久表空间，

TEMPORARY是指临时表空间。在创建表空间时默认都是永久表空间。

- EXTENT MANAGEMENT DICTONARY| LOCAL：指定表空间的管理方式，DICTIONARY是指字典管理方式，LOCAL是指本地的管理方式。在创建表空间时默认的管理方式是本地的管理方式。



说明

使用本地表空间管理的方式可以减少数据字典表的争用现象，并且也不需要表空间进行回收。因此，Oracle推荐使用本地表空间管理的方式创建表空间。

【示例2】使用语句创建表空间

根据上面给出的创建表空间的语法，首先使用最简单的方式在SQL*Plus中创建一个表空间TESTONE，结果如图16.9所示。TESTONE表空间中的数据文件是TESTONE.DBF，大小是10MB。

【示例3】使用语句创建自动扩展表空间

下面再创建一个可以自动扩展的表空间TESTTWO，并且设置扩展的最大值是128KB。创建结果如图16.10所示。



图16.9 创建TESTONE表空间



图16.10 创建TESTTWO表空间

16.2.2 重命名表空间

如果在创建好表空间后，需要重命名表空间也是非常容易的。重命名表空间通常也可以用两种方式，一种是在企业管理器中；另一种是使用语法在SQL*Plus中。

1. 使用企业管理器重命名表空间

【示例4】在企业管理器中重命名表空间

在查看表空间的列表图16.2所示界面中，选择使用企业管理器创建的表空间EMTEST，单击【查看】按钮，出现如图16.11所示界面。单击【编辑】按钮，修改表空间的名称后，单击【应用】按钮，出现图16.12所示界面。



图16.11 EMTEST表空间查看界面

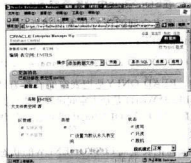


图16.12 编辑表空间界面

至此，已经把表空间EMTEST重命名为EMTES。

2. 使用SQL*Plus重命名表空间

重命名表空间也可以使用语句在SQL*Plus中完成。具体语法如下：

```
ALTER TABLESPACE oldname rename to newname;
```

但是，能够完成重命名的前提是要重命名的表空间已经存在，所以要在重命名之前查看数据字典中是否存在该表空间。

【示例5】在SQL*Plus中重命名表空间

下面就把之前定义的TESTONE表空间重命名为TESTONENEW，如图16.13所示。



图16.13 重命名TESTONE表空间

注意

不是所有的表空间都可以重命名，SYSTEM和SYSaux表空间就不能重命名；除此之外，当表空间处于OFFLINE状态时也不可以重命名。

16.2.3 设置表空间的读写状态

表空间在创建时如果不指定状态，默认是读写状态，除了读写状态之外，还有只读状态。设置表空间的读写状态也可以在企业管理器和SQL*Plus中完成。

1. 在企业管理器中设置表空间的读写状态

如果要更改现有表空间的读写状态，需要在表空间的编辑页面（如图16.12所示）更改表空间的状态。

【示例6】在企业管理器中设置只读状态

下面把EMTES表空间的读写状态更改为只读状态，单击【应用】按钮后如图16.14所示。至此，表空间的状态更改完成。

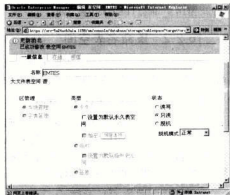


图16.14 更改表空间的状态

2. 在SQL*Plus中设置表空间的读写状态

设置表空间的读写状态的语法比较简单。具体语法如下：

```
ALTER TABLESPACE tablespace READ {ONLY|WRITE};
```

其中，READ ONLY是把表空间设置为只读状态；READ WRITE是把表空间设置为读写状态。

【示例7】在SQL*Plus中设置表空间状态为只读

下面把表空间TESTONENEW的状态更改为只读状态，如图16.15所示。



图16.15 更改表空间的状态

至此, TESTONENEW表空间的状态就更改成了只读状态。

如果要把表空间TESTONENEW的状态更改成读写状态, 则语句如下:

```
ALTER TABLESPACE TESTONENEW READ WRITE;
```



注意 在把表空间更改成只读状态时, 要把表空间设置成联机状态。

16.2.4 设置表空间的可用状态

表空间的可用状态是指表空间的联机和脱机状态, 如果把表空间设置成联机状态, 那么表空间就可以被用户操作, 反之设置成脱机状态, 表空间就是不可用的。下面以企业管理器和SQL*Plus两种方式讲解如何设置表空间的可用状态。

1. 使用企业管理器设置表空间的可用状态

在表空间的编辑页面即图16.12中, 可以把表空间的状态更改成脱机状态。

【示例8】 在企业管理器中更改表空间的状态为脱机

如果要把表空间EMTES设置成脱机状态, 就选中**【脱机】**选项, 再单击**【应用】**按钮即可, 如图16.16所示。

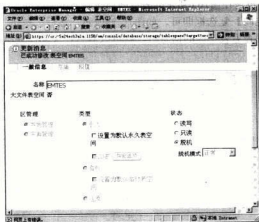


图16.16 设置表空间的可用状态

把表空间设置为脱机状态后, 脱机模式一栏也变成不可用状态。如果要设置成可用状态, 直接把状态设置成读写或者只读即可。

2. 使用SQL*Plus设置表空间的可用状态

设置表空间的可用状态与设置表空间的读写状态语法类似。具体语法如下:

```
ALTER TABLESPACE tablespace { ONLINE | OFFLINE [ NORMAL | TEMPORARY | IMMEDIATE ] }
```

【语法说明】

☐ **ONLINE**: 设置表空间为联机状态, 即可用状态。



❑ OFFLINE：设置表空间为脱机状态，即不可用状态。这里还包括3种方式，NORMAL指的是正常状态；TEMPORARY指的是临时状态；IMMEDIATE指的是立即状态。

【示例9】在SQL*Plus中设置表空间的状态为脱机

下面就把表空间TESTONENew设置成脱机状态，并选择临时状态的脱机方式，结果如图16.17所示。

如果要把表空间设置成联机状态，可以使用下面的语句：

```
ALTER TABLESPACE TESTONENew ONLINE;
```

16.2.5 建立大文件表空间

建立大文件的表空间与前面讲过的创建表空间有些类似，大文件表空间是在正常表空间不够用的情况下才需要创建的。创建大文件表空间也可以在企业管理器和SQL*Plus中完成。

1. 使用企业管理器创建大文件表空间

创建大文件表空间也是在图16.6所示的表空间创建界面中完成的。只需要在创建表空间时，选中【使用大文件表空间】复选框即可，如图16.18所示。然后，单击【应用】按钮，完成大文件表空间的创建。在查看表空间的信息时，就可以看到在大文件表空间处显示为“是”，如图16.19所示。



图16.18 创建大文件表空间



图16.19 查看表空间信息

2. 使用SQL*Plus创建大文件表空间

在SQL*Plus中创建大文件表空间与创建普通表空间的语法非常相似。定义大文件表空间的语法如下：

```
CREATE BIGFILE TABLESPACE tablespace
DATAFILE filename SIZE size
```

【示例10】在SQL*Plus中创建大文件表空间

使用上面的语法创建一个名字为TESTBIGFILE的大文件表空间，如图16.20所示。

这里,创建的表空间中的数据文件“bigfiletest.dbf”的大小是2GB,其实创建的大文件的表空间不仅限于2GB的大小,它最大可以是128TB。因此,可以说大文件表空间是存储大数据文件同时也是扩展表空间存储的好方式。

16.2.6 删除表空间

表空间的管理可以使用本地管理的方式,也可以使用数据字典的方式,在删除表空间时由于管理的方式不同,那么删除的速度也会受到影响。通过大量的试验可以发现,使用本地方式管理的表空间在删除时速度更快一些。因此,要删除表空间时可以考虑把表空间的管理方式修改成本地方式管理后再删除。

删除表空间既可以使用企业管理器完成,也可以在SQL*Plus中完成。下面就分别讲解这两种方式是如何删除表空间的。

1. 使用企业管理器删除表空间

在企业管理器中删除表空间的同时如果想把表空间中的数据文件也一起删除,那么,不需要先删除表空间中的数据文件再删除表空间,直接选择把表空间中的数据文件也一起删除就可以。

【示例11】在企业管理器中删除表空间

打开企业管理器后,找到表空间的页面,如图16.2所示,选择要删除的表空间,然后单击【删除】按钮,进入图16.21所示界面。

在图16.21所示界面中,可以看到一个“从存储删除相关联的数据文件”选项,如果想把表空间中的数据文件全部删除,可以选中该选项,如果只是删除表空间而不删除其中的数据文件,那么就可以取消选中该选项。在默认删除表空间时,该选项是选中状态的。



图16.20 创建大文件表空间



图16.21 删除表空间提示

注意 当在企业管理器中删除表空间并且删除表空间的数据文件后,要注销后重新登录才能看出表空间中的数据文件被删除。

2. 在SQL*Plus中使用语句删除表空间

使用语句删除表空间,也可以选择把表空间中的数据文件一并删除。删除表空间的语法如下:

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

【语法说明】

- ☐ [INCLUDING CONTENTS]: 如果在删除表空间时要把表空间中的数据文件也删除,可以在删除的表空间语句后面加上该语句。
- ☐ [CASCADE CONSTRAINTS]: 如果在删除表空间时要把表空间中的完整性也删除,可以在删除的表空间语句后面加上该语句。

**【示例12】在SQL*Plus中删除表空间**

下面就以删除表空间TEST，并且删除表空间中的数据文件为例说明，在SQL*Plus中执行结果如图16.22所示。

至此，表空间TEST和TEST中的数据文件已经被删除。



图16.22 删除表空间TEST

16.3 临时表空间的管理

在Oracle 11g数据库中除了上面讲述的表空间外，还有临时表空间，临时表空间主要是用来保存临时的数据信息的。本节将讲述如何创建和查询临时表空间以及在使用临时表空间时容易出现的问题。

16.3.1 建立临时表空间

临时表空间一般是指在数据库中存储数据，当内存不够时写入的空间，这个空间并不像一般的表空间，当执行完对数据库的操作后，该空间的内容自动清空。临时表空间经常会在使用一些操作时使用，如连接没有索引的两个表，查询数据时都会用到。建立临时表空间也可以在企业管理器中直接创建或在SQL*Plus中使用语句创建。下面分别讲解使用这两种方式创建临时表空间。

1. 在企业管理器中创建临时表空间

在企业管理器中创建临时表空间与创建一般的表空间的方法基本相同。

【示例13】在企业管理器中创建临时表空间TEMPTEST

在创建表空间时类型选为【临时】选项，如图16.23所示，此时创建的就是临时表空间。同时，也可以把【设置为默认临时表空间】复选框选中，该临时表空间就成为默认的临时表空间。创建了临时表空间后，在表空间页面中就显示出本次创建的TEMPTTEST表空间的类型就是TEMPORARY，如图16.24所示。



图16.23 创建临时表空间



图16.24 查看表空间的类型

2. 使用SQL*Plus创建临时表空间

只要掌握了前面的创建一般表空间的语法，学习创建临时表空间的语法还是比较容易的。



创建临时表空间的语法如下:

```
CREATE TEMPORARY TABLESPACE tablespace_name
TEMPFILE 'filename.dbf' SIZE
```

【示例14】在SQL*Plus中创建临时表空间temptable

下面使用上面的语句创建一个名为temptable的临时表空间。具体操作如图16.25所示。

设置临时表空间为默认表空间的语法如下:

```
ALTER DATABASE DEFAULT TEMPORARY TABLESPACE tablespace
```

【示例15】设置默认表空间

下面把上面创建的临时表空间temptable设置成默认的临时表空间。具体操作如图16.26所示。



图16.25 创建临时表空间



图16.26 设置默认临时表空间

16.3.2 查询临时表空间

如果要查看Oracle 11g数据库中都存在哪些临时表空间,一种方法是在企业管理器(图16.23所示的界面)中查看类型为TEMPORARY的表空间;另一种方法是在SQL*Plus中使用数据字典DBA_TEMP_FILES查看。

【示例16】查询临时表空间

下面就演示如何在数据字典DBA_TEMP_FILES中查看临时表空间。具体操作如图16.27所示。



图16.27 查询临时表空间

如果要查询临时表空间中用户的信息也可以在数据字典DBA_USERS中查看。查看说明 DBA_USERS数据字典时可以先使用DESC DBA_USERS语句得到字典中存在的字段信息,然后再使用SELECT语句把需要的信息查询出来。

16.3.3 创建临时表空间组

临时表空间组是由多个临时表空间组成的,每一个临时表空间组至少要有有一个临时表空间存在,并且临时表空间组的名称也不能和其他表空间重名。对于临时表空间组的创建,下面也从企业管理器创建方式和使用SQL*Plus创建两种方式讲解。



1. 使用企业管理器创建临时表空间组

在企业管理器的服务器菜单页面即图16.1中，单击【临时表空间组】选项，会出现图16.28所示的查看临时表空间组界面。

目前，在数据库中是不存在临时表空间组的。

【示例17】在企业管理器中创建临时表空间组TEMPGROUP

在创建临时表空间组时一定要先为表空间组设置临时表空间，然后再添加临时表空间。在图16.28所示界面中，单击【创建】按钮，进入创建表空间组的界面，如图16.29所示。



图16.28 查看临时表空间组



图16.29 创建临时表空间组

为临时表空间组输入名称后，在临时表空间栏处单击【添加/删除】按钮，进入为临时表空间组添加临时表空间的界面，如图16.30所示。

在左侧的“可用表空间”列表中选中当前表空间组要使用的表空间，然后单击【移动】选项，即可完成临时表空间的添加。同时，也可以使用全部移动、移去、全部移去3个选项来操作临时表空间。

说明

为临时表空间组添加表空间的前提是已经为数据创建了临时表空间，否则在图16.30的可用表空间一栏中是没有临时表空间的。

这里，在可用表空间中选中TEMPTABLE并将其移动到所选表空间中，效果如图16.31所示。



图16.30 添加临时表空间



图16.31 添加临时表空间TEMPTABLE



为临时表空间组添加临时表空间后，单击【确定】按钮，完成临时表空间的添加，界面如图16.32所示。单击【确定】按钮，即完成临时表空间组的创建，如图16.33所示。



图16.32 完成临时表空间的添加



图16.33 临时表空间组创建成功

2. 在SQL*Plus中创建临时表空间组

临时表空间组的创建实际上就是为表空间设置一个组，所以创建临时表空间组的语法和创建临时表空间的语法类似。在创建临时表空间时可以有两种方法，一种是在创建临时表空间组的同时也创建临时表空间；另一种是在创建临时表空间组时把已经存在的临时表空间移动到该临时表空间组中。

(1) 创建新的临时表空间存入临时表空间组中

创建新的临时表空间存入临时表空间组中，这种创建方式与创建临时表空间的语法非常类似。具体语法如下：

```
CREATE TEMPORARY TABLESPACE tablespace_name
TEMPFILE filename SIZE size TABLESPACE GROUP group_name;
```

【示例18】在SQL*Plus中新建临时表空间添加到临时表空间组

下面就利用上面的语法创建一个名为tempgroup的临时表空间组，并在临时表空间组中存入一个临时表空间temptable1。具体操作如图16.34所示。

(2) 把原来的临时表空间移到新创建的临时表空间组中

使用把原来的临时表空间移到新创建的临时表空间组中这种方式创建临时表空间组的前提是，已经存在要移动的临时表空间。具体语法如下：

```
ALTER TABLESPACE tablespace_name TABLESPACE GROUP group_name;
```

【示例19】在SQL*Plus中创建表空间组并加入临时表空间temptable1

下面把已经存入临时表空间组的tempgroup的临时表空间temptable1转移到tempgroup1中。具体操作如图16.35所示。

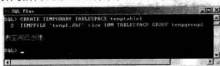


图16.34 创建临时表空间组



图16.35 更改临时表空间组

16.3.4 查询临时表空间组

临时表空间组创建完成后，如果想要查看临时表空间组中存放的表空间信息或者在当前数据库中存在的所有临时表空间组信息，可以在企业管理器或SQL*Plus中使用语句查看。下面分别用两种方式查看临时表空间组的信息。

1. 使用企业管理器查看临时表空间组的信息

在企业管理器中查看临时表空间组的信息可以直接在图16.1中单击【临时表空间组】选项，在16.3.3小节中已经创建过临时表空间组，下面就查看一下这些临时表空间组的具体信息，如图16.36所示。



图16.36 查看临时表空间组

在图16.36中可以查看到TEMPGROUP和TEMPGROUP1两个临时表空间组的信息，并且可以看到在临时表空间中存在的临时表空间数以及文件的大小等信息。如果想查看一个临时表空间组的详细信息，直接单击临时表空间组的名称或者选择一个临时表空间组单击【查看】按钮即可。

2. 使用SQL*Plus查询临时表空间组的信息

临时表空间组的信息存放在数据字典DBA_TABLESPACE_GROUPS中。

【示例20】查询临时表空间组信息

查询DBA_TABLESPACE_GROUPS即可查看到临时表空间组的信息，具体操作如图16.37所示。



图16.37 查询DBA_TABLESPACE_GROUPS



查询数据字典后，就列出了数据库中的临时表空间组的名称以及临时表空间组存在的临时表空间名称。

16.3.5 删除临时表空间组

如果删除数据库中多余的临时表空间组，不需要先把临时表空间组中的临时表空间移除，只要删除临时表空间组中的所有临时表空间同时就会把临时表空间组删除掉。删除临时表空间组也可以通过企业管理器和在SQL*Plus中使用语句删除两种方式。下面分别用这两种方式讲述如何删除临时表空间组。

1. 使用企业管理器删除临时表空间组

在企业管理器中删除临时表空间组是非常容易的，只要在图16.36所示的查看临时表空间组的列表中选中要删除的临时表空间组，单击【删除】按钮，就会出现图16.38所示的删除确认界面。

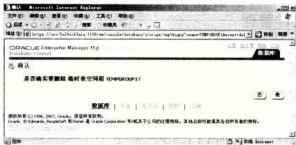


图16.38 确认删除临时表空间组

在图16.38所示界面中，单击【是】按钮即可删除该临时表空间组。

注意 临时表空间组删除后不能恢复，所以在执行删除操作时一定要慎重。另外，在删除临时表空间组后，临时表空间的文件并没有删除，如果要删除临时表空间组中的临时表空间，那么就要先把临时表空间组中的临时表空间移除。

2. 在SQL*Plus中删除临时表空间组

删除临时表空间组时一定要把临时表空间组中存在的临时表空间也一并删除。具体删除临时表空间组的语法如下：

```
DROP TABLESPACE tablespace_name INCLUDING CONTENTS AND DATAFILES
```

【示例21】删除临时表空间组

下面利用上面的语法，删除临时表空间组TEMPGROUP中的TEMPTABLE1临时表空间。具体操作如图16.39所示。



图16.39 删除表空间



由于在临时表空间组TEMPGROUP中只存在一个临时表空间TEMPTABLE1，所以当删除了临时表空间TEMPTABLE1时，再查询DBA_TABLESPACE_GROUPS数据字典，结果如图16.40所示。

```

SQL> SELECT * FROM DBA_TABLESPACE_GROUPS;

GROUP_NAME          TABLESPACE_NAME
-----
TEMPGROUP            TEMPTABLE1
  
```

图16.40 查询临时表空间组

注意 在删除临时表空间时，不能删除默认的临时表空间。

16.4 数据文件管理

前面已经学习了如何创建和管理表空间，在创建和管理表空间时都会用到数据文件。本节就将学习如何移动数据文件、删除数据文件以及查看数据库中的数据文件。

16.4.1 移动数据文件

在创建表空间时数据文件就已经创建好了，如果想把当前表空间中的数据文件移动到其它表空间中，在Oracle 11g的企业管理器中是无法完成的，只能在SQL*Plus中通过写语句来完成。移动数据文件的基本步骤如下：

1) 把要存放数据文件所用的表空间设置成脱机状态。语句如下：

```
ALTER TABLESPACE tablespace_name OFFLINE
```

2) 可以手动把要移动的文件移动到其他的表空间中。

3) 更改数据文件的名称。语句如下：

```
ALTER TABLESPACE tablespace_name RENAME DATAFILE oldfilename TO newfilename;
```

4) 把该表空间设置成联机状态。语句如下：

```
ALTER TABLESPACE tablespace_name ONLINE
```

16.4.2 删除数据文件

在使用数据文件时经常会去除一些没有用的数据文件，但是删除数据文件也有前提条件。当数据文件处于以下3种情况时它是不能够被删除的：

- ☐ 数据文件中存在数据；
- ☐ 数据文件是表空间中唯一或第一个的数据文件；
- ☐ 数据文件或数据文件所在的表空间处于只读状态。

如果要删除的数据不具有上面的3种情况，那么删除数据文件既可以在企业管理器中直接删除，也可以在SQL*Plus中删除。下面就分两种情况进行讲解。



1. 使用企业管理器删除数据文件

在图16.1所示的页面中，单击【数据文件】选项就可以查看到数据库中存在的的所有数据文件，如图16.41所示。

【示例22】在企业管理器中删除数据文件

在图16.41所示界面中选择要删除的数据文件并单击【删除】按钮，会出现一个删除提示页面，如图16.42所示。



图16.41 显示所有数据文件



图16.42 删除数据文件提示

单击【是】按钮，即可删除该数据文件。

说明

既然数据文件是存在表空间中的，所以删除数据文件也可以在表空间中进行，但是也不移除表空间中第一个或者唯一的一个数据文件。

2. 使用SQL*Plus删除数据文件

在SQL*Plus中删除数据文件的基本语法如下：

```
ALTER TABLESPACE tablespace_name DROP DATAFILE 'filename';
```

【示例23】在SQL*Plus中删除数据文件

下面就利用上面的语句删除test表空间中的数据文件TESTNEW。具体操作如图16.43所示。

删除完成后，可以在数据字典DBA_DATA_FILES中查看数据文件TESTNEW是否在test表空间中被删除。具体操作如图16.44所示。



图16.43 删除数据文件TESTNEW



图16.44 查询数据文件

由此，可以看出数据文件TESTNEW已经被删除了。

16.5 小结

通过对本章的学习，读者应该既能够掌握一般表空间、临时表空间、临时表空间组以及大文件表空间在企业管理器中的创建、修改、删除的操作，同时也能够掌握在SQL*Plus中使用语句完成创建、修改、删除的操作。俗话说得好：“师傅领进门，修行在个人”，所以在学习本章时尽量要在看书的同时进行实际操作，这样学习效果会更好。

16.6 习题

简答题

1. 什么是表空间？如何创建一个表空间？
2. 临时表空间有什么用？临时表空间组中可以没有临时表空间吗？
3. 数据文件在任何情况下都可以删除吗？
4. 创建一个名为TEST的临时表空间，并把它存放到TESTGROUP中。（使用语句创建）
5. 创建一个名为BIGSPACE的大文件表空间。

第17章 与数据库安全性有关的对象

用户管理和权限管理是确保数据库中数据安全的重要手段。如果能够正确地对数据库中的用户赋予权限,那么就能在很大程度上提高数据库的安全性。本章包括以下知识点:

- ☐ 用户的创建与管理
- ☐ 权限的创建与管理
- ☐ 角色的创建与使用
- ☐ 概要文件的创建与使用

本章内容基本涵盖了对数据库中的用户、权限、角色以及概要文件的创建与管理的方法。通过本章的学习,可以熟练地掌握数据库中的用户、权限、角色以及概要文件的使用方法。

17.1 用户管理

Oracle中的用户管理主要是针对数据库管理员说的,只有管理员才有权限创建、修改以及删除用户。本节就将介绍如何创建以及管理用户。

17.1.1 什么是用户管理

Oracle的用户管理应该说是每个数据库管理员都会遇到的一个问题,对用户管理涉及的主要问题就是用户所赋予的权限,根据每个用户访问Oracle数据库的需求不同,分配给用户的权限也就不同。如果数据库管理员对Oracle数据库用户的权限分配得合理,那么就能够提高数据库的安全性;相反,如果对Oracle数据库的用户权限分配得不合理,那么就会给数据库造成很大的隐患。对于权限分配的问题将在17.2节中继续介绍。

在Oracle中用户登录数据库的方式主要有三种:第一种是一般的密码验证方式。第二种是外部验证方式,这种方式并没有把验证密码存放在Oracle数据库中,其验证的密码通常与数据库所在的操作系统的密码一致。第三种就是全局验证方式,这种验证方式也不常用,它也不是把密码存放在Oracle数据库中的。这三种验证方式中最常用的就是密码验证的方式,同时这种方式的安全性也更高一些。

17.1.2 创建用户

在Oracle中创建用户必须拥有数据库管理员的权限才能创建,在创建用户时还需要注意的是创建的用户的密码必须是以字母开头的。创建用户也可以在企业管理器中或者在SQL*Plus中使用语句创建。

1. 使用企业管理器创建用户

使用企业管理器创建用户相对比较简单,但是登录的用户也要是以管理员身份登录的才可以。

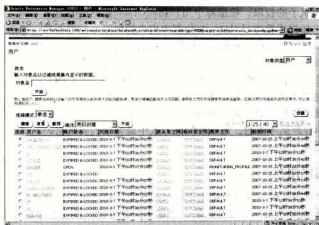
(1) 打开企业管理器中的【服务器】选项卡

进入企业管理器后，打开【服务器】选项卡，如图17.1所示。



(2) 打开用户管理页面

在图17.1所示界面中，单击【安全性】选项下面的【用户】选项，出现如图17.2所示界面。



(3) 创建新用户

在图17.2所示界面中，单击【创建】按钮，出现图17.3所示的创建新用户界面。

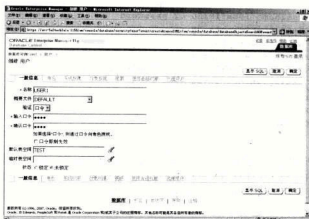


图17.3 创建新用户界面

在图17.3所示界面中,各选项前带有“*”的是必填项,其中必填项有用户的名称、输入口令和确认口令。其他项如果不填,则使用当前数据库中默认的文件。

说明 在“验证”选项中有3个选项，一个是口令，当选择“口令”选项时必须输入用户的口令；一个是外部，当选择“外部”选项时，则由操作系统、网络或其他外部源向角色授权，不必输入口令；一个是全局，当选择“全局”选项时，则由某一企业目录服务向角色全局授权，也不必输入口令。

输入用户的信息后,单击【确定】按钮,会出现图17.4所示页面。在此页面中就可以查看到已经创建的用户USER1的信息。

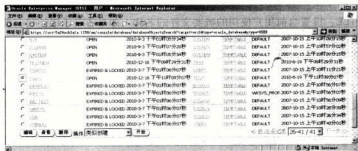


图17.4 创建用户完成页面

在此页面中就可以看到USER1用户使用的表空间是TEST，临时表空间是TEMPTABLE，状态是OPEN状态等信息。

2. 使用SQL*Plus创建用户

在SQL*Plus中创建用户使用的语法选项是比较多的。一般创建用户的语法如下：

```
CREATE USER username IDENTIFIED BY password
OR EXTERNALLY AS certificate_DN
OR GLOBALLY AS directory_DN
[DEFAULT TABLESPACE tablespace]
[TEMPORARY TABLESPACE tablespace|tablespace_group_name]
[QUOTA size|UNLIMITED ON tablespace]
[PROFILE profile]
[PASSWORD EXPIRE]
[ACCOUNT LOCK|UNLOCK]
```

【语法说明】

- ☐ 在创建用户时有三种验证方式，与用企业管理器创建用户一样，以口令作为验证方式时选择【IDENTIFIED BY password】选项即可，以外部作为验证方式时选择【EXTERNALLY AS certificate_DN】选项；以全局作为验证方式时选择【GLOBALLY AS directory_DN】选项即可。
- ☐ DEFAULT TABLESPACE：设置默认表空间，如果省略了该语句，那么这个新创建的用户就存放在数据库的默认表空间中，如果在数据库没有设置默认表空间，那么创建的用户就存放在SYSTEM表空间中。
- ☐ TEMPORARY TABLESPACE：设置临时表空间或临时表空间组，可以把临时表空间存放在临时表空间组中，如果省略了该语句，那么就会把临时的文件存放到当前数据库中默认的临时表空间中；如果没有默认的临时表空间，那么会把临时文件存放到SYSTEM的临时表空间中。
- ☐ QUOTA：设置当前用户使用表空间的最大值，在创建用户时可以有多个QUOTA来设置用户在不同表空间中能够使用的表空间大小。如果设置成UNLIMITED，表示对表空间的使用没有限制。

注意 创建用户时不能设置用户在临时表空间上使用的范围。

- ☐ PROFILE：设置当前用户使用的概要文件的名称，如果省略了该子句，那么用户就使用当前数据库中默认的概要文件。
- ☐ PASSWORD EXPIRE：设置当前用户密码立即处于过期状态，用户如果想再登录数据库必须要更改密码。
- ☐ ACCOUNT：设置用户的锁定状态，如果设置成LOCK，那么该用户不能访问数据库；如果设置成UNLOCK，那么用户则可以访问数据库。在Oracle 11g中默认的用户状态都为锁定的状态。

利用创建用户的语句，分别创建以口令验证方式的用户和以外部方式验证的用户。

【示例2】在SQL*Plus中创建以口令方式验证的用户USER2

创建一个以口令方式验证的用户USER2，并且设置成密码立即过期的方式。具体实现代码如下：

```
01 CREATE USER USER2
02 IDENTIFIED BY abcd
03 DEFAULT TABLESPACE test
04 QUOTA 10M ON test
05 TEMPORARY TABLESPACE temp
```

```
06 PROFILE pro_test;
07 PASSWORD EXPIRE;
```

【代码解析】

- ☐ 第2行设置用户USER2的口令是abcd。
- ☐ 第3行设置用户USER2的默认表空间是test。
- ☐ 第4行设置用户USER2可以在表空间test中使用的磁盘限额是10MB。
- ☐ 第5行设置用户USER2的临时表空间是temp。
- ☐ 第6行设置用户USER2使用的概要文件是pro_test。
- ☐ 第7行设置用户的密码是立即过期状态。

【示例3】在SQL*Plus中创建以外部方式验证的用户USER3
创建一个验证方式是外部的用户USER3。具体代码如下：

```
01 CREATE USER USER3
02 IDENTIFIED EXTERNALLY
03 DEFAULT TABLESPACE test
04 QUOTA 5M ON test
05 PROFILE pro_test;
```

【代码解析】

- ☐ 第2行设置用户USER3的验证方式是外部验证。
- ☐ 第3行设置用户USER3的默认表空间是test。
- ☐ 第4行设置用户USER3在test表空间中能够使用的磁盘配额是5MB。
- ☐ 第5行设置用户USER3所使用的概要文件是pro_test。

17.1.3 修改用户信息

创建好用户后有时会对用户的设置有所改变，下面就分别使用企业管理器和SQL*Plus两种方式修改已经创建好的用户。

1. 使用企业管理器修改用户

在企业管理器中修改用户信息与创建用户一样，都是在图17.2所示页面上进行操作的。

【示例4】在企业管理器中修改用户

在图17.2所示的页面上选择要修改的用户，然后单击**【编辑】**按钮，进入图17.5所示界面。



图17.5 编辑用户页面



在图17.5所示的页面中除了用户的名称不能修改之外，其他的信息都是能够修改的。修改好用户信息后单击【应用】按钮，用户信息就修改成功。

2. 在SQL*Plus中修改用户

在SQL*Plus中修改用户的语法与创建用户的选项非常类似。具体的语法如下：

```
ALTER USER user IDENTIFIED
{ BY password [ REPLACE old_password ]
| EXTERNALLY [ AS 'certificate_DN' ]
| GLOBALLY [ AS '[directory_DN]' ]
}
[DEFAULT TABLESPACE tablespace]
[TEMPORARY TABLESPACE { tablespace | tablespace_group_name }]
[QUOTA { size_clause| UNLIMITED}ON tablespace]
[PROFILE profile]
[PASSWORD EXPIRE]
[ACCOUNT { LOCK | UNLOCK }]
```

上面语法的选项含义已经在创建用户时讲解过了，这里就不再说明。

下面利用上面的语法完成三个修改用户信息的操作。

【示例5】修改用户USER2密码

具体语句如下：

```
ALTER USER USER2
IDENTIFIED BY newabed
DEFAULT TABLESPACE test;
```

【示例6】为用户USER2添加临时表空间

具体语句如下：

```
ALTER USER USER2
TEMPORARY TABLESPACE newspacel;
```

【示例7】为用户USER1设置密码立即过期

具体语句如下：

```
ALTER USER USER1PASSWORD EXPIRE;
```

17.1.4 删除用户

数据库管理员经常会去除一些废弃不用的用户，而删除用户的同时也要把该用户所使用的数据库对象一并删除掉。删除用户的操作也从使用企业管理器删除和在SQL*Plus中使用语句删除两种方式讲解。

1. 使用企业管理器删除用户

在企业管理器中删除用户，也是在用户的列表页面（如图17.2所示）上进行操作的。

【示例8】在企业管理器中删除用户USER1

在图17.2所示页面上选择要删除的用户，单击【删除】选项，就会出现图17.6所示的删除提示页面。

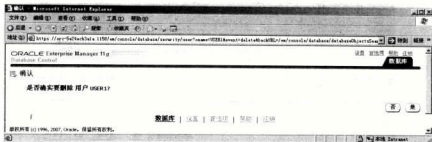


图17.6 删除用户提示

单击【是】按钮之后，用户USER1就被成功删除了。

2. 在SQL*Plus中删除用户

在SQL*Plus中删除用户也是比较简单的，只要知道用户名即可删除，而且是把该用户下所有的数据文件一起删除。具体删除的语法如下：

```
DROP USER user CASCADE
```

说明 如果要删除的用户中没有任何数据库对象，那么就可以省略CASCADE关键字。

17.2 权限管理

权限管理主要是针对17.1节中讲的用户还有后面要讲解的角色进行管理的。数据库管理员如果为了保证数据库的安全就要控制好每一个用户或者角色的权限。本节就将讲述如何授予、撤销以及查询用户或角色的权限。

17.2.1 什么是权限管理

在Oracle数据库中，权限有系统权限和对象权限两类。系统权限主要是指SESSION权限、USER权限等，也就是说对数据库的系统级的操作都可以称为系统权限。对象权限主要是指表对象、序列、触发器等操作的权限。

17.2.2 授予权限

授予权限的对象就是用户或者角色，授予权限的操作包括授予系统权限和授予对象权限，下面分别在企业管理器和SQL*Plus中讲解如何授予系统权限与对象权限。

1. 在企业管理器中授予系统权限

【示例9】在企业管理中添加系统权限

这里以授予用户的权限为例进行授予用户权限。首先要选择授予权限的用户，在图17.2所示页面中，选择一个用户单击【编辑】按钮，进入用户修改界面，如图17.5所示。在页面上选择【系统权限】选项卡，出现图17.7所示界面。

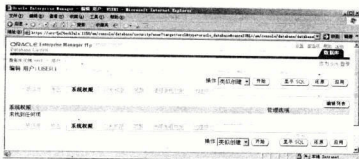


图 17.7 编辑系统权限

在编辑系统权限页面中，单击【编辑列表】选项，出现选择系统权限的页面，如图17.8所示。

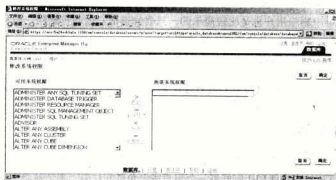


图 17.8 选择系统权限

在图17.8中，左侧是系统中的所有可用的系统权限，如果要为用户添加权限，可以直接选中权限，之后单击【移动】或【全部移动】选项，添加的系统权限就会出现在右侧的“所选系统权限”列表框中。相反，如果想从所选的系统权限中移除权限，也可以选择【移去】或者【全部移去】选项。在添加好权限后，单击【确定】按钮，权限就添加成功了，出现图17.9所示界面。



图 17.9 权限添加完成



系统权限添加完成后，再单击【应用】按钮，系统权限就添加到指定的用户上了。这里，在系统权限列表中还有一个管理选项，如果把该选项选中，那么这个用户就会拥有授予 ALTER USER 的权限。在默认情况下，该选项是未被选中的。

2. 使用企业管理器添加对象权限

在企业管理器中添加对象权限的方法与添加系统权限的方法类似。

【示例10】在企业管理器中添加对象权限

在图17.5所示界面中，单击【对象权限】选项，添加对象权限的界面如图17.10所示。

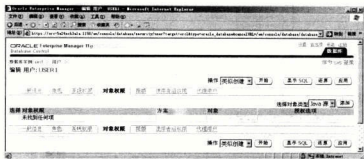


图17.10 添加对象权限

在添加对象权限的页面中，首先选择要添加的对象权限的类型，这个对象类型主要有Java源、Java类、作业类、序列、表等对象。选择好对象类型后，单击【添加】按钮，出现图17.11所示的界面。这里选择表对象。



图17.11 添加表对象的权限

在添加表对象权限之前首先要选择表对象，然后单击选择表对象后面的图标，会出现图17.12所示的界面。

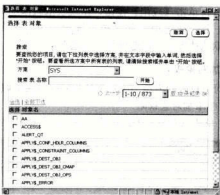


图 17.12 选择表对象

选择表对象后，就可以为用户使用该表对象授予权限，如图 17.13 所示。



图 17.13 授予对象权限

授予对象权限与授予系统权限一样，这里就不详细说明。选择好权限后，单击【确定】按钮，完成对用户对象权限的授予。

3. 在 SQL*Plus 中授予系统权限

授予权限的操作一般是由数据库管理员来处理的，只有拥有了足够的权限才能够给其他用户授予权限。授予用户权限的基本语法如下：

```
GRANT system_privilege
| ALL PRIVILEGES TO {user IDENTIFIED BY password | role}
[WITH ADMIN OPTION]
```

【语法说明】

□ system_privilege：创建的系统权限名称。

- ☐ ALL PRIVILEGES: 可以设置除SELECT ANY DICTIONARY权限之外的所有系统权限。
- ☐ {user IDENTIFIED BY password |role}: 设置权限的对象, user IDENTIFIED BY password子句代表的是设置指定用户的权限; role代表的是设置角色的权限。
- ☐ WITH ADMIN OPTION: 设置该子句后, 表示当前给予授权的用户还可以给其他用户进行系统权限的授予。

【示例11】在SQL*Plus中授予用户系统权限

下面就利用上面的语法, 授予用户一个系统权限session。具体语法如下:

```
GRANT create session to USER1
```

也可以为它加上WITH ADMIN OPTION管理选项, 让USER1也具有授予create session的权限。具体代码如下:

```
GRANT create session to USER1 WITH ADMIN OPTION
```

4. 在SQL*Plus中授予用户对象权限

授予对象权限与授予系统权限的语法相似, 只是有一点区别。具体语法如下:

```
GRANT object_privilege|ALL
ON schema.object
TO user|role
[WITH ADMIN OPTION ]
[WITH THE GRANT ANY OBJECT ]
```

【语法说明】

- ☐ object_privilege: 对象权限的名称。
- ☐ ALL: 如果选择ALL, 则代表授予用户所有的对象权限, 这个权限在使用的时候一定要慎重。
- ☐ schema.object: 为用户授予的对象权限使用的对象。
- ☐ user|role: user代表的是用户, role代表的是角色, 角色的授予会在17.3节中详细讲述。
- ☐ WITH ADMIN OPTION: 设置该子句后, 表示当前给予授权的用户还可以给其他用户进行系统授权。
- ☐ WITH THE GRANT ANY OBJECT: 设置该子句后, 表示当前给予授权的用户还可以给予其他用户对象权限。

【示例12】在SQL*Plus中授予用户对象权限

利用上面的语句, 为用户USER1授予表对象AA删除的权限。具体代码如下:

```
GRANT DELETE ON AA TO USER1
```

17.2.3 撤销权限

撤销权限也叫收回权限, 也就是删除用户的系统权限或者对象权限。使用企业管理器删除用户的系统权限或者对象权限非常简单, 只是在添加系统权限或对象权限时, 选择把当前用户的权限移除就可以, 如图17.13所示, 把所选权限列表框中的ALTER权限去除, 只需要选中该权限, 然后单击【移去】按钮即可完成。



撤销权限在SQL*Plus中的语句就和授予权限有很大差别了，这里重点讲解如何在SQL*Plus中撤销系统权限和对象权限。

1. 撤销系统权限

撤销系统权限的前提是当前的用户已经存在要撤销的系统权限，另外，撤销权限的操作只有数据库管理员才可以操作。具体语法如下：

```
REVOKE system_privilege
FROM user|role
```

【示例13】撤销USER1的系统权限

利用上面的语法，完成撤销USER1的create session权限。具体代码如下：

```
REVOKE create session FROM USER1
```

2. 撤销对象权限

撤销对象权限也使用REVOKE关键字完成。具体语法如下：

```
REVOKE object_privilege|ALL
ON schema.object
FROM user|role
[CASCADE CONSTRAINTS]
```

这里需要说明的就是CASCADE CONSTRAINTS选项，它表示该用户授予其他用户的权限也一并撤销。

【示例14】撤销用户USER1的对象权限

利用上面的语法，完成撤销用户USER1的DELETE权限。具体代码如下：

```
REVOKE DELETE ON AA FROM USER1;
```

说明 在撤销用户权限时，撤销系统权限与撤销对象权限是不同的。如果撤销用户的系统权限，那么该用户授予其他用户的系统权限仍然存在；而撤销了用户的对象权限后，用户授予其他用户的对象权限也同时被撤销了。

17.2.4 查询用户的权限

查询用户权限也是数据库管理员经常要做的工作，由于权限都是针对用户或角色的，这里，只查询用户的权限。在查询权限时都通过用户名去查询该用户权限。查询用户权限在企业管理器或者SQL*Plus中都可以完成。下面就分别用这两种方式来查询用户的权限。

1. 在企业管理器中查询用户的权限

在企业管理器中查询用户的权限是非常容易的。

【示例15】在企业管理器中查询用户ANONYMOUS的权限

在图17.2所示页面中，选择想要查看权限的用户ANONYMOUS，单击【查看】按钮，即可查看该用户的权限信息以及其他信息，如图17.14所示。

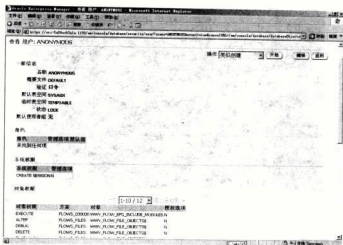


图17.14 查看用户权限

由此，就可以看出ANONYMOUS用户的系统权限和对象权限。

2. 在SQL*Plus中查询用户权限

Oracle 11g中的用户权限存放在数据库的数据字典中，用户的系统权限存放在数据字典DBA_SYS_PRIVS中，用户的对象权限存放在数据字典DBA_TAB_PRIVS中。

查询在企业管理器中查询过的用户ANONYMOUS的权限，可以检查在企业管理器中查看的结果是否正确。

【示例16】查询ANONYMOUS用户的系统权限

具体操作如图17.15所示。



图17.15 查询系统权限

【示例17】查询ANONYMOUS用户的对象权限

具体操作如图17.16所示。

除了可以在DBA_SYS_PRIVS和DBA_TAB_PRIVS数据字典中查询权限之外，还可以

说明 直接在数据字典USER_SYS_PRIVS中查询当前登录用户的系统权限；在数据字典ALL_TAB_PRIVS中查询当前登录用户的对象权限。

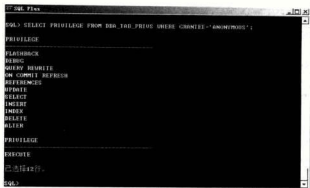


图17.16 查询对象权限

17.3 角色管理

角色在数据库中也是非常重要的一个名词，在数据库中角色并不是某个用户所独有的，而是用户可以根据权限的不同授予用户不同的角色。本节将讲述如何创建和使用角色。

17.3.1 什么是角色

在数据库中提到的比较多的名词“角色”，究竟是什么呢？实际上角色就是由一组权限组成，用户也是可以被授予权限的。那么角色与用户有什么区别呢？

用户是数据库的使用者而角色是权限的授予对象，给用户授予角色，也可以理解成给用户授予一组权限。数据库中的角色可以授予多个用户也可以不授予用户，并且一个用户可以被授予多个角色。角色是在数据库中由数据库管理员定义的权限集合，方便对不同用户的权限授予。例如，如果在数据库中设置一个拥有能够查询数据库中表的权限的角色，那么凡是用户需要拥有查询数据库中表的权限时，都可以直接授予该角色。

17.3.2 创建角色

在理解了角色是数据库管理员的好帮手之后，掌握在数据库中创建角色是至关重要的。创建角色也可以在企业管理器和SQL*Plus中完成。下面就分别通过这两种方式讲解如何创建角色。

1. 在企业管理器中创建角色

在企业管理器中的服务器选项页面（如图17.1所示），选择【安全性】列表中的【角色】选项，进入角色浏览页面，如图17.17所示。

【示例18】在企业管理器中创建角色

创建用户角色可以分为两个步骤：

（1）创建角色的名称

在图17.17所示页面中，单击【创建】按钮，出现创建角色界面，如图17.18所示。

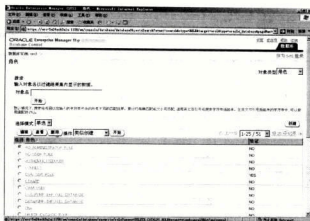


图17.17 角色浏览页面

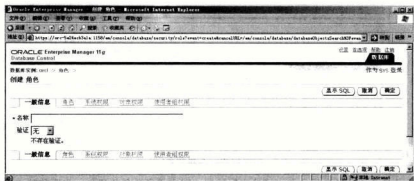


图17.18 创建角色界面

在图17.18所示界面中，填入角色的名称，并选择角色的验证方式即可创建一个角色。角色的验证方式除了同用户的验证方式一样的口令验证、外部验证、全局验证3种之外，还有就是没有验证的方式，可以根据需要创建不同验证方式的角色。

(2) 授予角色权限或其他角色

角色创建好后，角色中不包含任何角色，就像新建的一个文件夹一样，没有任何文件。如果要想使用角色还要为角色授予权限，这个权限也包括系统权限和对象权限两类。前面已经学习过如何为用户授予权限，授予角色的权限与授予用户的权限类似，这里就不再重复讲解了。这里主要讲解一个为角色授予其他角色。授予其他已经在数据库中存在的角色，那么当前的角色就拥有了这个角色中存在的权限。为当前的角色授予角色，选择图17.18中的【角色】选项卡，出现如图17.19所示界面。

在图17.19所示界面中，单击【编辑列表】按钮，即可为当前的角色添加角色。页面如图17.20所示。

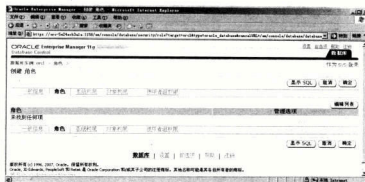


图17.19 查看角色信息

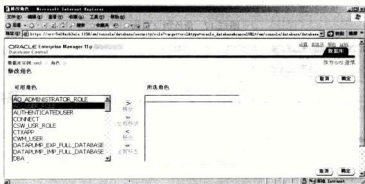


图17.20 编辑角色

为当前角色添加角色后，单击【确定】按钮，即完成了角色的添加。当要移除当前角色中的角色时，也可以在【所选角色】列表框中移去多余的角色。

2. 在SQL*Plus中创建角色

在企业管理器中创建角色分为创建角色和授予角色权限两个步骤，而在SQL*Plus中创建角色也分为两个步骤。

(1) 创建角色

创建角色就是在数据库中创建一个只有名称的角色。具体语法如下：

```
CREATE ROLE role
[NOT IDENTIFIED|IDENTIFIED BY [password]| IDENTIFIED BY
EXTERNALLY|IDENTIFIED BY GLOBALLY]
```

【语法说明】

在创建角色时要注意验证的方式，可以选择的验证方式有四种，第一种是NOT IDENTIFIED，不需要验证；第二种是IDENTIFIED BY [password]，这是口令验证的方式；第三种是IDENTIFIED BY EXTERNALLY，这是外部验证的方式；第四种是IDENTIFIED BY

GLOBALLY, 这是全局验证方式。

(2) 授予角色权限

授予角色权限与授予用户权限所使用的语法一样, 只是授予对象不是用户而是角色。这里只给出一个授予角色系统权限的语法, 授予角色对象权限的语法可以参考17.2节自己动手修改一下。具体语法如下:

```
GRANT system_privilege
|ALL PRIVILEGES TO role
[WITH ADMIN OPTION]
```

在给角色授予权限时, 数据库管理员必须拥有GRANT_ANY_PRIVILEGES权限才可以给角色赋予任何权限。

【示例19】在SQL*Plus中创建角色

下面就利用上面的两个步骤来创建角色TESTROLE, 并给角色授予CREATE SESSION权限。具体代码如下:

```
CREATE ROLE TESTROLE
NOT IDENTIFIED;
```

```
GRANT CREATE SESSION TO TESTROLE;
```

这样, 就在数据库中创建了一个拥有CREATE SESSION的角色TESTROLE。用户可以根据创建角色的步骤, 创建不同验证方式和授予不同权限的角色。

17.3.3 设置角色

角色创建完成后并不能直接使用它, 而是要把角色授予用户才能使角色生效。具体语法如下:

```
GRANT role TO user
```

下面就把TESTROLE角色授予给USER1用户。具体代码如下:

```
GRANT TESTROLE TO USER1;
```

说明

在企业管理器中也可以把角色授予给用户, 在用户的编辑界面中选择所需要的角色即可, 这里不再详细讲述。

由于一个用户可以同时拥有多个角色, 所以也可以设置哪些角色生效哪些角色不生效。设置的生效与失效方法如下:

```
01 SET ROLE role
02 SET ROLE ALL
03 SET ROLE ALL EXCEPT role
04 SET ROLE NONE
```

【代码解析】

- 第1行设置的是指定的角色生效。
- 第2行设置用户的所有角色都生效。
- 第3行设置在EXCEPT后的角色不失效。





□ 第4行设置用户的所有角色都失效。

【示例20】在SQL*Plus中设置角色生效和失效

下面利用上面的语法，设置角色TESTROLE在当前用户上生效。具体代码如下：

```
SET ROLE TESTROLE
```

设置角色TESTROLE在当前用户上失效。具体代码如下：

```
SET ROLE ALL EXCEPT TESTROLE
```

17.3.4 修改角色

角色创建完成后，如果想要修改其内容，可以使用企业管理器，也可以在SQL*Plus中完成修改。在企业管理器中修改角色的操作可以在图17.17所示的界面中进行操作，可以修改角色中已经授予的角色以及权限。在SQL*Plus中修改角色的语法如下：

```
ALTER ROLE role
[NOT IDENTIFIED|IDENTIFIED BY [password]] IDENTIFIED BY
EXTERNALLY|IDENTIFIED BY GLOBALLY
```

上面的语法只是修改角色本身，如果修改已经授予角色的权限或者角色，则使用GRANT或者REVOKE来完成。

17.3.5 删除角色

如果要移除不用的角色，可以在企业管理器中删除角色，也可以在SQL*Plus中删除角色。下面就分别讲解用这两种方式删除角色。

1. 使用企业管理器删除角色

在企业管理器中删除角色是很容易的。

【示例21】在企业管理器中删除角色

在企业管理器中的角色浏览页面如图17.17所示。首先从中选择要删除的角色，然后单击【删除】按钮，会出现如图17.21所示的删除提示界面。



图17.21 删除角色提示

在图17.21所示界面中，单击【是】按钮，角色删除成功。

说明 在删除角色的同时, 所有拥有该角色的用户也将自动撤销该角色所授予的权限。

2. 在SQL*Plus中删除角色

在SQL*Plus中删除角色也是非常容易的, 删除角色的语法如下:

```
DROP ROLE rolename
```

这里, rolename就是数据库中要删除的角色名称。

17.3.6 查询角色

查询角色也是数据库管理员必须要做的工作, 在企业管理器中查询角色是比较简单的。在企业管理器的服务器选项页面上直接选择角色就可以查看到数据库已经存在的角色, 如图17.17所示。除了查看全部的角色信息外, 还可以查看某一个用户上的角色, 这只需要在查询用户信息时查看角色的信息即可。这里就不再演示如何在企业管理器中查询角色的操作了。

在SQL*Plus中查询角色是在数据库的数据字典DBA_ROLE_PRIVS中查询指定用户的角色。

【示例22】在数据字典DBA_ROLE_PRIVS中查询SYSTEM用户的角色

下面利用DBA_ROLE_PRIVS数据字典查询用户SYSTEM的角色, 查询结果如图17.22所示。



```
SQL> SELECT GRANTED_ROLE, DEFAULT_ROLE FROM DBA_ROLE_PRIVS WHERE GRANTEE='SYSTEM'
1
```

GRANTED_ROLE	DEF
DBA_ADMINISTRATOR_ROLE	YES
DBA	YES
MONITOR_USER	YES

```
SQL>
```

图17.22 查询SYSTEM用户的角色

17.4 概要文件PROFILE

PROFILE是Oracle中的概要文件, 在PROFILE中主要存放的就是数据库中的系统资源或者数据库使用限制的一些内容。在本节中将讲述如何创建、修改、删除以及查询概要文件PROFILE。

17.4.1 什么是PROFILE

PROFILE就是Oracle中的概要文件, 在Oracle系统中如果不创建概要文件, 默认会在系统中使用默认的概要文件DEFAULT。如果创建用户时没有为用户设置概要文件, 那么默认都会使用数据库中的默认概要文件。概要文件会给数据库管理员带来很大的方便, 数据库管理员可以先对数据库中的用户分组, 按照每一组的权限不同, 建立不同的概要文件。需要说明的一点是, 虽然概要文件可以用于用户, 但是概要文件是不能在角色中使用的。

17.4.2 创建PROFILE

在每一个数据库中默认的都是概要文件PROFILE, 如果需要添加概要文件, 在企业管理器和SQL*Plus中都可以创建。下面就分别用这两种方式创建概要文件PROFILE。



1. 使用企业管理器创建概要文件PROFILE

在企业管理器中概要文件的管理也是在服务器选项页面的“安全性”列表中的，在图17.1所示界面中，单击【概要文件】选项，出现概要文件浏览页面，如图17.23所示。

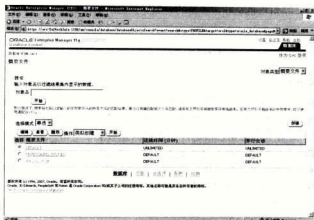


图17.23 浏览概要文件

【示例23】在企业管理器中创建概要文件

在图17.23所示界面中，单击【创建】按钮，出现创建概要文件的页面，如图17.24所示。

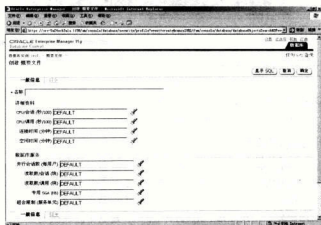


图17.24 创建概要文件

在图17.24所示的界面中，创建概要文件的名称是必须输入的，其他的选项如概要文件的详细资料、数据库服务都通过单击每一个选项后的图标添加，如果不添加则使用每一项的默认值；如果需要为概要文件添加口令，那么就单击【口令】选项，如图17.25所示。

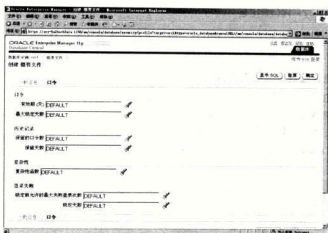


图 17.25 为概要文件添加口令

在图17.25所示的界面中，可以为概要文件添加口令、历史记录、复杂性以及登录失败的选项，如果没有添加，就使用相应的默认值。

添加完概要文件的信息后，单击【确定】按钮，概要文件就添加完成。

2. 在SQL*Plus中创建概要文件

在SQL*Plus中创建概要文件要比在企业管理器中复杂多了，所以初学者在创建概要文件时可以选择在企业管理器中创建。在SQL*Plus中创建概要文件的语法如下：

```
CREATE PROFILE profile
LIMIT
{resource_parameters|password_parameters}
```

【语法说明】

- resource_parameters：资源参数，这些参数主要有CPU_PER_SESSION，代表允许一个会话占用CPU的总量；CPU_PER_CALL，代表允许一个调用占用CPU的最大值；CONNECT_TIME代表允许一个持续的会话的最大值。这些资源参数还有很多，在这里就不一一介绍了。
- password_parameters：口令参数，这里也列举几个比较常用的参数，PASSWORD_LIFE_TIME，指的是多少天后口令失效；PASSWORD_REUSE_TIME，是指密码保留的时间；PASSWORD_GRACE_TIME，是指设置密码失效后锁定。

【示例24】在SQL*Plus中创建概要文件TESTPROFILE

下面就利用上面的语法创建一个概要文件TESTPROFILE，并设置密码的失效天数是30。具体代码如下：

```
CREATE PROFILE TESTPROFILE
LIMIT
PASSWORD_LIFE_TIME 30;
```

读书笔记

記

【示例25】在企业管理器中修改概要文件DEFAULT

[illegible]

修改好概要文件的相关信息后,单击【应用】按钮,即可完成概要文件的修改。

```
ALTER PROFILE profile
LIMIT
(resource_parameters|password_parameters)
```

```
ALTER PROFILE TESTPROFILE
LIMIT
CPU_PER_SESSION 1000;
```



17.4.4 删除PROFILE

在数据库中的概要文件也可以用企业管理器和SQL*Plus两种方式删除，下面分别讲解使用这两种方式如何删除概要文件PROFILE。

1. 使用企业管理器删除概要文件

【示例27】使用企业管理器删除概要文件TESTPRO

在企业管理器中删除概要文件只要在概要文件的浏览页面即图17.23中，选择要删除的概要文件TESTPRO，单击【删除】按钮，就会出现图17.27所示的删除提示页面。



图17.27 删除概要文件提示

单击【是】按钮，删除概要文件TESTPRO操作成功。

注意 在Oracle中默认的概要文件是不能删除的。

2. 在SQL*Plus中删除概要文件

使用语句删除概要文件是很容易的，使用DROP语句就可以完成。具体语法如下：

```
DROP PROFILE profile [CASCADE]
```

【语法说明】

- CASCADE：如果要删除的概要文件已经被用户使用过，那么在删除概要文件时要加上该关键字，把用户所使用的概要文件也撤销；如果要删除的概要文件没有被用户使用过，那么就可以省略该关键字。

【示例28】在SQL*Plus中删除概要文件TESTPRO

下面就利用上面的语句，删除概要文件TESTPRO。代码如下：

```
DROP PROFILE TESTPRO CASCADE
```

17.4.5 查询PROFILE

查询数据库的概要文件PROFILE，在企业管理器中可以直接在概要文件的浏览页面查看到，如图17.23所示。在SQL*Plus中查看概要文件，要在数据字典DBA_PROFILES中查询得到。

【示例29】在SQL*Plus中查看概要文件

下面在数据字典中查询在数据库存在的所有概要文件，查询的结果如图17.28所示。



图17.28 查询数据字典DBA_PROFILES

在图17.28中为了去除重复的概要文件，加上了关键字DISTINCT。从查询结果可以看出，与在企业管理器中查询到的概要文件的名称是完全相同的。

17.5 小结

通过本章的学习，读者能够掌握数据库中几个确保数据库安全的对象，能够掌握用户、权限、角色以及概要文件的使用。在掌握它们的使用的基础上，能够分清用户、权限以及角色三者之间的关系。同时作为数据库的管理员，还要掌握概要文件在数据库中的使用，这样才能够确保数据库的安全。除了在本章介绍的几个与数据库安全有关的对象之外，读者还可以学习一下关于数据库审计的设置等安全性的对象。

17.6 习题

简答题

1. 用户、权限以及角色之间的关系是什么？
2. 查询数据库中sys用户的权限有哪些？
3. 创建一个角色ROLE1，并给角色授予CREATE SESSION的系统权限。
4. 创建一个概要文件，并删除已经创建的概要文件。

第18章 备份与恢复

数据库的备份和恢复是每一个学习数据库的人都要掌握的技能,备份是保存数据库的副本,恢复就是把以前从数据库中备份的文件还原到数据库中。本章将学习如何备份与恢复Oracle数据库,包括以下知识点:

- 备份数据库
- 恢复数据库

本章内容基本涵盖了备份和恢复数据库的操作方法。通过本章的学习,读者可以熟练地备份和恢复数据库。

18.1 数据库备份与恢复

备份和恢复数据库是每一个数据库必须具备的功能,同时也是数据库管理员必须具备的能力。对数据库进行备份和恢复是确保数据库中数据安全的一个关键技术。本节将讲述数据库备份和恢复的基本概念。

18.1.1 什么是数据库备份

数据库备份就是将数据库的内容全部复制出来保存到计算机的另一个位置或者其他存储设备上。数据库备份也分很多种,主要有物理备份和逻辑备份。

物理备份是指通常所说的归档模式备份(又叫热备份)和非归档模式备份(又叫冷备份),归档模式备份是当数据库的模式设置成归档模式时对数据库进行的备份;而非归档模式备份是当数据库的模式设置成非归档模式时对数据库的备份。逻辑备份主要是指对数据库的导入和导出操作,在Oracle 10g之前使用IMP/EMP的方式进行导入和导出操作,从Oracle 10g开始引入了数据泵技术,使用EXPDP/IMPDP的方式对数据进行导入和导出的操作。

18.1.2 什么是数据库恢复

数据库恢复就是把从数据库中备份出来的数据重新还原给原来的数据库,数据库的恢复技术分为完全恢复和不完全恢复两种。完全恢复是指把数据库恢复到数据库失败时的数据库状态,不完全恢复是指将数据库恢复到数据库失败前的某一时刻的数据库状态。数据库备份分物理备份和逻辑备份,数据库恢复也分物理恢复和逻辑恢复,物理恢复就是把从数据库中备份的文件重新复制到原来的数据库中;逻辑恢复就是把从数据库中导出的数据再导入原来的数据库。

18.2 物理备份和恢复数据库

物理备份数据库的方式是比较容易完成的,主要是把数据库中的文件复制到磁盘的不同位置,是在数据库出现问题时进行恢复使用的。本节将讲述在脱机和联机两种状态下备份与恢复数据库中的文件。



18.2.1 对数据库进行脱机备份

脱机备份称为冷备份。首先，管理员身份的用户使用shutdown命令关闭数据库的服务，之后复制需要的文件，包括把数据文件和控制文件等相关的内容复制到其他磁盘的路径上。如果数据库出现问题，那么就可以把从数据库中复制出来的相关内容再复制回原来的数据库目录中。

18.2.2 对数据库进行联机备份

联机备份称为热备份，是在数据库的归档模式下进行的备份。查看数据库中日志的命令如下：

```
archive log list
```

【示例1】查询本机数据库的日志状态

具体操作如图18.1所示。



图18.1 查询日志状态

从示例1的查询结果中可以看出，目前数据库的日志模式是非存档模式，同时自动存档方式也是禁用的。设置数据库日志模式为归档模式，使用下面的语句完成：

```
01 alter system set log_archive_start=true scope=spfile;
02 shutdown immediate;
03 startup mount;
04 alter database archivelog;
```

【代码解析】

- ☐ 第1行修改系统的日志方式是归档模式。
- ☐ 第2行关闭数据库。
- ☐ 第3行启动mount实例，但不启动数据库。
- ☐ 第4行更改数据库为归档日志模式。

【示例2】更改数据库中日志状态

利用上面的语句把数据库的日志状态更改为归档模式，具体操作如图18.2所示。

更改完成后使用archive log list语句查看日志模式的状态，即可看到当前日志模式已经被修改为归档模式，并且自动存档方式已经启用。

把数据库设置成归档模式后，就可以进行数据库的备份与恢复操作了。

【示例3】备份表空间TEST

(1) 将数据库的状态设置成打开状态

改变数据库的状态为open。具体的语句如下：

```
alter database open;
```

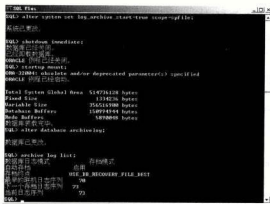


图18.2 更改日志模式为归档模式

(2) 开始备份表空间

假设备份的表空间是TEST。命令如下：

```
alter tablespace TEST begin backup;
```

(3) 打开数据库中的oradata文件夹（一般数据库对象都存放在该文件夹中），把文件复制到磁盘中的另一个文件夹或其他磁盘上。

(4) 结束表空间的备份

在完成前面的操作之后，执行下面的命令结束备份：

```
alter tablespace TEST end backup;
```

此时就完成了表空间TEST的备份操作。具体操作效果如图18.3所示。



图18.3 备份表空间TEST

至此，已经把表空间备份到其他位置了。

【示例4】恢复表空间中的数据文件

当数据库出现问题时，需要恢复表空间TEST。按照下面的步骤完成恢复表空间的操作：

(1) 对当前的日志进行归档

对当前使用的日志进行归档的命令如下：

```
alter system archive log current;
```

(2) 切换日志文件

由于在一个数据库中一般有3个日志文件，所以需要3次使用下面的语句来切换日志文件：



```
alter system switch logfile;
```

(3) 关闭数据库服务

这里为了模拟TEST表空间中的数据文件丢失，先把数据库关闭，然后删除TEST表空间中的数据文件TEST01.DBF。关闭数据库的命令如下：

```
shutdown immediate
```

(4) 删除数据文件并重新启动数据库

删除数据文件首先要找到存放数据文件的位置，在默认情况下数据文件会存放在数据库的ORADATA文件夹中，也有在创建表空间添加数据文件时指定的目录。如果不清楚数据文件存放的位置，可以在企业管理器中查看一下表空间中的数据文件，也可以直接在DOS窗口下的v\$datafile数据字典中查看表空间中数据文件的位置。找到数据文件后直接将其删除即可，然后启动数据库。启动数据库的命令如下：

```
startup;
```

在使用startup命令启动数据库后，会出现图18.4所示的错误提示界面。

从图18.4中可以看到现在已经缺少了编号为10的数据文件，也可以通过查看数据字典v\$recover_file确认缺少的数据文件，查看的结果如图18.5所示。



图 18.4 启动后出现错误



图 18.5 查询数据字典v\$recover_file

从查询的结果可以看到，丢失的数据文件编号确实是10。

(5) 将数据文件设置成脱机状态并删除

在恢复数据文件之前需要先把数据文件设置成脱机状态（offline状态），并且删除该数据文件。具体命令如下：

```
alter database datafile 10 offline drop;
```

(6) 把数据库的状态设置成OPEN

在完成上述操作后，就可以为恢复数据库做好准备，把数据库的状态设置成OPEN。具体命令如下：

```
alter database open;
```

(7) 恢复数据文件

在数据库的状态是OPEN时就可以开始恢复数据库了。恢复表空间TEST的数据文件命令如下：

```
recover datafile 10;
```

这里的编号10仍然是之前查看到的数据文件的编号，这也是需要注意的问题。在恢复时数



据文件的编号要一致。恢复效果如图18.6所示。



图18.6 恢复效果

在使用上面的命令恢复数据文件时会出现指定日志的选项，这里直接输入AUTO即可。

(8) 设置数据文件为联机状态

在恢复完数据库后还需要把数据文件设置成联机状态。具体的命令如下：

```
alter database datafile 10 online
```

至此，就完成了数据文件的恢复操作。为了验证数据文件是否恢复成功，可以重新启动数据库看一下效果。

注意

在恢复数据库中的数据文件时，把数据库文件设置成脱机状态后，就需要把之前备份好的数据文件复制到原来的数据文件存放的位置，否则会出现图18.7所示的错误。



图18.7 恢复数据文件出错

18.3 逻辑备份和恢复数据库

逻辑备份数据库是数据库管理员使用最多的，逻辑备份除了在DOS下进行外，还可以在企业管理器的OEM中进行。本节将讲述数据库的逻辑导出与导入操作。

18.3.1 逻辑导出数据

导出数据可以使用EXP工具完成，也可以使用在Oracle 10g以后出现的EXPDP工具完成。下面分别使用这两种方式对数据库进行导出备份。

1. 使用EXP工具备份

EXP工具可以将数据库中的对象有选择性地备份出来，可以使用EXP工具导出的数据库对象有表、方案、表空间以及数据库。下面分别讲解如何使用EXP工具导出表、方案、表空间、数



数据库4个数据库对象。使用EXP工具并不是在SQL*Plus状态下，而是在DOS命令窗口下完成的。

(1) 导出表

进入DOS命令窗口，输入下面的语句：

```
C:\>EXP username/password
```

这里的username、password就是登录数据库的用户名和密码，但是这里的用户不能是SYS。

【示例5】根据提示导出表productinfo

下面就利用上面的语句导出表productinfo。首先使用SCOTT用户登录数据库，然后按照提示导出表数据。具体操作过程如图18.8所示。

```
C:\WINDOWS\system32\cmd.exe - EXP scott/tiger
Microsoft Windows [版本 5.2.3790]
(C) 版权所有 1985-2003 Microsoft Corp.

C:\Documents and Settings\Administrator\EXP scott/tiger
Export: Release 11.1.0.6.0 - Production on 星期一 7月 19 11:27:19 2010
Copyright (c) 1982, 2007, Oracle. All rights reserved.

连接到: Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
输入数据块大小(字节): 8096 > 200

导出文件: EXP01.DMP > c:\productinfo.dmp
(ORACLE用户) 或 (C:\>) > c
导出表数据 (yes/no)? yes > yes
表: SCOTT.PRODUCTINFO
已导出 ZHS16GB 字节集和 AL32UTF16 NCHAR 字节集
按指定表名或模式名称指定要导出的表...
要导出的表 (C) 或分区 (C1 P): (C) 或 (C1 P): <按 RETURN 退出> > productinfo
正在导出表 PRODUCTINFO导出了 # 行
要导出的表 (C) 或分区 (C1 P): (C) 或 (C1 P): <按 RETURN 退出> >
```

图18.8 导出表productinfo的过程

从图18.8所示的导出过程可以看到，在导出文件部分是为导出文件存放保存文件的路径，这里用的是c:\productinfo.dmp。如果选择导出的是表，那么可以在后面输入要导出的表名。在导出文件时可以选择E<完整的数据库>来导出数据库所有的文件。

除了像上面一步一步选择之外，还可以直接导出表。

【示例6】直接导出表productinfo

具体操作方法如图18.9所示。

```
C:\WINDOWS\system32\cmd.exe
C:\exp scott/tiger File=c:\test.dmp tables=productinfo
Export: Release 11.1.0.6.0 - Production on 星期一 7月 19 11:42:00 2010
Copyright (c) 1982, 2007, Oracle. All rights reserved.

连接到: Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
已导出 ZHS16GB 字节集和 AL32UTF16 NCHAR 字节集
按指定表名或模式名称指定要导出的表...
直接导出表 PRODUCTINFO导出了 # 行
按为终止导出，按有出现警告。
C:\>
```

图18.9 直接导出表productinfo

图18.9中导出的表productinfo是在SCOTT方案中的表,如果要导出其他方案中的表,则需要表名的前面加上方案名。假设要导出SYSTEM方案下的TEST表,那么就要写成SYSTEM.TEST。如果要导出多个表,可以在表名之间加上“,”间隔开。

注意 在导出语句后面一定不能加上分号,如果加上分号,那么系统就会认为导出的表是表名加上分号的,如图18.10所示。

(2) 导出表空间

导出表空间与导出表不同,导出表空间的用户必须是数据库的管理员角色。导出表空间的命令如下:

```
C:\>EXP username/password FILE="filename.dmp" TABLESPACES="tablespaces_name"
```

【语法说明】

- ☐ username/password: 登录数据库使用的用户名和密码,一定是具有数据库管理员权限的用户。
- ☐ filename.dmp: 存放备份的表空间的数据文件。
- ☐ tablespaces_name: 要备份的表空间名称。

【示例7】导出表空间TEST

下面利用上面的语句备份表空间TEST。具体操作如图18.11所示。



图18.10 导出表错误



图18.11 导出表空间

2. 使用EXPDP导出数据

EXPDP是Oracle 10g开始引入的数据泵技术,数据泵技术是在数据库之间或者在数据库与操作系统之间传输数据的工具。EXPDP是数据泵导出的工具,它可以把数据库中的对象导出到操作系统中。使用EXPDP工具与EXP不同的是,在使用EXPDP时要先创建目录对象,通过这个对象就可以找到要备份数据的数据库服务器,并且使用EXPDP工具备份出来的数据必须存放在目录对象对应的操作系统的目录中。下面将分步讲解如何使用EXPDP导出数据。

(1) 创建目录对象

创建目录对象是使用EXPDP工具进行导出的前提。创建目录对象的语法如下:

```
SQL:>CREATE DIRECTORY directory_name AS 'file_name'
```

【语法说明】

- ☐ directory_name: 创建的目录名称。

(2) 给使用目录的用户赋权限

GRANT READ,WRITE ON DIRECTORY directory name TO SCOTT

【示例8】创建目录并给用户SCOTT赋权限

下面就利用上面的命令完成目录DIR的创建和SCOTT用户权限的授予，具体操作流程如图18.12所示。



前面已经创建好了目录，使用EXPDP工具导出数据的方法与EXP导出的方法类似，也是在DOS的命令窗口中实现的。

下面就利用导出表的语句，再次导出表productinfo。具体操作如图18.13所示。



如果想备份其中的某一个表，可以使用下面的命令完成：

```
C:\>EXPDP username/password DIRECTORY= directory_name DUMPFILE=file_name TABLES=table_name
```

【语法说明】

□ directory_name：存放导出数据的目录名称。

□ file_name：导出数据存放的文件名。

□ table_name：准备导出的表名，对于多个表可以用逗号隔开。

例如，备份productinfo表，命令如下：

```
EXPDP scott/tiger DIRCETORY=DIR DUMPFILE=temp.dmp TABLES=productinfo
```

18.3.2 逻辑导入数据

逻辑导入数据是逻辑导出数据过程的逆过程，导入数据可以使用与EMP对应的IMP工具，也可以使用与EMPDP对应的IMPDP工具。本小节将分别讲述如何使用IMP和IMPDP工具完成数据的导入工作。

1. 使用IMP导入数据

导入数据是将数据库之前导出的数据导入到数据库中。

【示例10】使用IMP导入表productinfo

导入之前使用EMP工具导出表productinfo。具体导入方法如图18.14所示，根据导入提示一步一步完成导入操作。

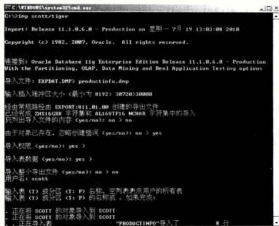


图18.14 导入表productinfo

除了按照上述步骤导入数据之外，还可以直接进行导入表的操作。

【示例11】使用IMP导入指定表productinfo

直接把productinfo.dmp文件中的数据导入到表productinfo中。命令如下：

```
C:\>imp scott/tiger file=cproductinfo.dmp tables=productinfo
```



2. 使用IMPDP导入数据

使用IMPDP导入数据的前提是数据是使用EMPDP导出的，同样也是在DOS窗口下直接输入IMPDP和登录数据库的用户名，即可导入数据。

【示例12】使用IMPDP导入数据

具体操作流程如图18.15所示。

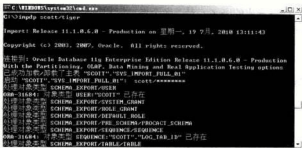


图18.15 使用IMPDP导入数据

使用IMPDP不指定目录时，系统会找到用户使用的目录对象，将目录对象中的内容全部导入数据库中。除了这种全部导入的方式，也可以根据需求把目录中某一个文件导入。

【示例13】使用IMPDP导入指定的表productinfo

具体导入方法如下：

```
C:>impdp scott/tiger directory=dir dumpfile= temp.dmp tables=productinfo
```

不仅可以导入表，还可以导入方案和表空间。请读者自行练习，这里就不再赘述了。

18.4 小结

通过本章的学习，读者可以掌握数据库的物理备份和逻辑备份的方法，并且在物理备份里还学习了使用脱机备份和连接备份两种方式来备份数据库；使用逻辑备份时不仅使用了常用的导入（IMP）、导出（EMP）方法，还学习了使用数据泵技术导入（IMPDP）和导出（EMPDP）数据。除了上面讲解的导入和导出方法之外，还可以使用企业管理器完成导入和导出的操作。这里就不详细讲解了，这部分内容希望读者自行完成。

18.5 习题

简答题

1. 使用物理备份的方法备份数据文件。
2. 在scott方案下创建表空间test，并在test表空间中创建表student，在student表中设置学生编号、学生姓名、学生年龄3个字段，并给表增加5条记录。分别使用EMP和EMPDP导出表中的数据，把数据存放在d:\中。
3. 根据第2题导出的数据，使用IMP和IMPDP导入数据。

第19章 使用RMAN工具

RMAN是Oracle数据库提供了一种恢复和备份数据库的工具，也是数据库管理员管理数据库常用的工具之一。本章将学习如何使用RMAN工具恢复和备份数据库，包括以下知识点：

- ☐ RMAN概述
- ☐ 恢复目录的使用
- ☐ 通道的使用
- ☐ 备份集的使用
- ☐ 常用命令的使用

本章内容基本涵盖了RMAN工具的操作方法。通过本章的学习，读者可以熟练地使用RMAN工具完成数据库的备份和恢复工作。

19.1 RMAN概述

RMAN是Recovery Manager的缩写，为Oracle的恢复管理器，主要用于备份和恢复数据库，这也是Oracle推荐使用的一款恢复备份工具。本节主要讲述RMAN的特点以及与RMAN有关的一些概念。

19.1.1 RMAN的特点

RMAN既然是Oracle的恢复管理器，那么RMAN究竟有什么特点呢？在Oracle官方网站上给出的RMAN的特点主要有以下4个：

- ☐ 它可以备份数据库、表空间、数据文件、控制文件以及日志文件。
- ☐ 压缩备份可以只备份发生变化的内容。
- ☐ 集成了第三方的磁带媒介软件。
- ☐ 可以在Oracle数据库的目录中存放备份信息。

19.1.2 与RMAN有关的概念

使用RMAN进行备份和恢复操作时，不可避免地要提到下面的一些常用概念，主要有目标数据库、闪回区、介质管理等。下面就一一介绍这些概念。

1. 目标数据库

当在使用RMAN进行备份时，就会提到目标数据库这个概念。目标数据库就是使用RMAN工具进行备份和还原的数据库。

2. RMAN客户端

当使用RMAN工具进行数据备份时，使用的前提就是计算机要拥有RMAN客户端。一般情况下，这个客户端是不用单独安装的，只要安装Oracle系统就自动安装了RMAN客户端。通常安装的目录与Oracle数据库的目录一致。

3. 闪回区

闪回区 (Flash Recovery Area) 是在磁盘上的一个区域, 在这个区域中存放与数据库的备份和恢复相关的一些文件, 使用闪回区能够方便用户备份和还原数据库。闪回区这个概念是在 Oracle 10g时引出的。

4. 介质管理

介质管理设备通常被称为 SBT (System Backup to Tape) 设备, 也就是把数据库备份到磁带中。RMAN通过介质管理器将数据备份到磁带上, 介质管理器通常由第三方软件商提供。它将数据块中的数据流从RMAN通道进程传递到对应的磁带上。

5. 恢复目录

恢复目录 (Recovery Catalog) 是一个独立的数据库, 用于存放目标数据库的备份, 这个目标数据库可以是一个, 也可以是多个。

19.2 使用恢复目录

恢复目录是使用RMAN工具进行备份时要使用的存储备份信息的数据库, 这也是Oracle推荐使用的一种方式, 这样存储要比直接把目标数据库中内容存放到控制文件中更节省空间。本节将讲述如何使用恢复目录备份数据。

19.2.1 创建恢复目录

为了确保数据的安全, 一般情况下都是把恢复目录数据库创建到另外一个Oracle服务器中。在创建恢复目录时还要考虑数据库的容量, 这个容量的大小当然要取决于目标数据库容量的大小。一般情况下, 在恢复目录数据库设置的恢复目录表空间可以设置为50MB左右。

创建恢复目录分为连接恢复目录的数据库、创建恢复目录的用户、给恢复目录用户赋角色以及创建恢复目录4个步骤。

1. 连接恢复目录的数据库

恢复目录的用户就是指在备份目标数据库时使用的用户, 这个用户与其他用户不同, 必须要赋予RECOVERY_CATALOG_OWNER的角色才可以。

首先在SQL*Plus中连接到恢复目录所在的数据库。连接数据库的语法如下:

```
conn username/password@servicename;
```

【语法说明】

- ☐ username: 登录恢复目录数据库的用户名。
- ☐ password: 登录恢复目录数据库的密码。
- ☐ servicename: 恢复目录数据库的服务名。

【示例1】连接恢复目录数据库

下面就利用上面的语句连接恢复目录数据库, 使用SYSTEM用户名, 密码是abc123, 连接的服务名是RM。代码如下:

```
SQL>conn SYSTEM/abc123@RM;
```



2. 创建恢复目录的用户

由于使用恢复目录时需要一些权限，所以说最好单独为恢复目录创建一个用户。创建用户的语句如下：

```
CREATE USER username IDENTIFIED BY password
[DEFAULT TABLESPACE tablespace_name]
```

【语法说明】

- ☐ username：新创建的恢复数据库的用户名。
- ☐ password：新创建的恢复数据库的密码。
- ☐ [DEFAULT TABLESPACE tablespace_name]：可选项，是给当前用户设置一个默认表空间，如果不设置默认表空间，则默认的表空间是SYSTEM表空间。最好给恢复目录设置一个默认的表空间，这样便于管理备份的数据。

【示例2】创建恢复目录用户RMANUSER

下面就利用上面的语句完成恢复目录用户的创建。这里用户名设置成RMANUSER，密码设置成RMAN，使用的默认表空间为RMANTABLESPACE。具体的创建语句如下：

```
CREATE USER RMANUSER IDENTIFIED BY RMAN
DEFAULT TABLESPACE RMANTABLESPACE
```

3. 给恢复目录用户赋角色

只创建一个数据库的用户是不能实现RMAN备份与恢复工作的，还需要给该用户赋予权限和角色。恢复目录用户应该拥有数据库管理员的权限，并在此基础上还要拥有RECOVERY_CATALOG_OWNER的权限。在此假设恢复目录用户已经拥有了数据库管理员的权限，这里只给恢复目录用户赋予RECOVERY_CATALOG_OWNER的角色。具体的语句如下：

```
GRANT RECOVERY_CATALOG_OWNER TO RMANUSER
```

4. 创建恢复目录

使用恢复目录用户登录数据库后就可以创建恢复目录，上面的3步都是在SQL*Plus中完成的，下面需要在恢复目录管理器中完成，也就是在图19.1所示界面中完成。



图19.1 进入恢复管理器

在图19.1所示界面中，连接到恢复目录数据库。具体的连接语句如下：

```
RMAN>connect catalog RMANUSER/ RMAN@RM
```

连接到RM恢复目录数据库之后，可以使用下面的语句创建恢复目录：



RMAN>create catalog

网络数据库恢复案例

为了方便自行测试恢复目录的使用，可以直接在本机的数据库中创建一个新的表空间，并在数据库上新创建一个用户，赋予RECOVERY_CATALOG_OWNER的角色，创建好后直接使用该用户登录本机数据库，并在本机数据库上创建一个恢复目录。具体操作效果如图19.2所示。

在本机创建恢复目录时经常会遇到以下两个问题：

(1) 在连接数据库时出现无监听程序的提示

在登录本机的恢复目录数据库时，如果输入了服务名，就会出现图19.3所示的界面。



图19.2 在本机数据库中创建恢复目录



图19.3 无监听程序错误界面

要解决无监听程序错误的问题，直接去掉Oracle的服务名即可，登录远程数据库才需要使用数据库的服务名。

(2) 在创建恢复目录时出现无权限的创建

在创建恢复目录时如果创建恢复目录的用户没有系统管理员的权限，那么就会出现图19.4所示的界面。



图19.4 无权限创建错误界面

出现无权限创建的页面后，只需要给用户RMANUSER赋予相应的权限就可以解决。如果把上述的两个问题解决了，则在创建恢复目录时就能够出现图19.2所示的界面。

使用恢复目录可以在一处同时存放多个目标数据库的文件，方便数据库管理员的管理，这就要求尽量把恢复目录数据库放置到与目标数据库不同的服务器上。另外，一个最重要的优点就是恢复目录可以长时间保存RMAN备份的源数据。

19.2.2 使用RMAN连接

前面已经学习了如何创建恢复目录，在Oracle 11g数据库中，连接恢复目录数据库的一种

方式是直接在DOS窗口下使用命令RMAN进入到恢复管理器状态；另一种方式是在企业管理器中连接。本小节在DOS方式下连接数据库。

1. 使用目标数据库中的控制文件备份数据

当备份数据库时不使用恢复目录，而是使用目标数据库中的控制文件来存放数据，那么使用目标数据库中的控制文件代替恢复目录。具体登录语法如下：

```
C:\>RMAN TARGET username/password nocatalog;
```

【语法说明】

□ TARGET：代表要连接的是目标数据库。

□ nocatalog：代表不使用恢复目录。

【示例3】连接目标数据库

下面就利用上面的语句连接目标数据库。具体操作如图19.5所示。

从图19.5所示的界面中可以看到，目前已经用用户名SYS和密码abc123连接到目标数据库，并且当前的状态是使用目标数据库控制文件替代恢复目录。



如果目标数据库不在本地，需要远程连接，那么可以在用户名和密码之后输入@目标数据库的服务名的方式连接目标数据库。例如，如果orcl是远程数据库的服务名，那么在本例中写成RMAN TARGET sys/abc123@orcl即可。

2. 连接到恢复目录数据库

连接恢复目录数据库的方法前面已经讲过了，下面就具体学习一下连接到恢复目录数据库的语法。具体语法如下：

```
C:\>rman target username/password@servicename catalog username/password
```

【语法说明】

catalog：指恢复目录，在catalog后面是恢复目录数据库的用户名和密码。

【示例4】在C:\>下连接到恢复目录数据库

下面就利用上面的语句登录恢复目录数据库，如图19.6所示。



图19.5 使用控制文件替代恢复目录



图19.6 连接到恢复目录数据库

这里，RMANUSER/RMAN是恢复目录的用户名和密码，现在连接的目标数据库和恢复目录数据库都是使用本地数据库，所以没有加上服务名。

除了使用上面的这种方法连接数据库外，还可以使用CONNECT命令在RMAN>下连接恢复目录数据库。具体的命令如下：

```
RMAN:>CONNECT TARGET username/password@servicename
```



【示例5】 在RMAN:>下连接恢复数据库

下面就使用上面的命令连接恢复数据库，具体操作如图19.7所示。



图19.7 使用CONNECT命令连接恢复目录数据库

由此可以看出，使用CONNECT与直接在C:>目录下使用RMAN命令连接恢复数据库的效果是一样的，在实际操作中哪种方法都可以。

19.2.3 在恢复目录中注册数据库

在恢复目录中注册数据库是以备份目标数据库为前提的，可以使用REGISTER命令在恢复目录数据库中注册数据库。其语法如下：

REGISTER database

【示例6】注册数据库

注册数据库时首先要连接到恢复目录数据库，具体操作如图19.8所示。

在此可以看到，使用REGISTER命令后，已经完成了在恢复目录中注册数据库的操作。

【示例7】 查询RMAN的配置

实际上，在目标数据库中已经存在的RMAN的配置也是可以查看到的，即使用SHOW ALL命令即可实现。具体操作如图19.9所示。



图19.8 注册数据库



图19.9 RMAN的配置参数

19.2.4 使用企业管理器创建恢复目录

在企业管理器中创建恢复目录是在企业管理器的【可用性】选项卡中完成的，如图19.10所示。



图19.10 企业管理器的【可用性】选项卡

【示例8】在企业管理器中添加恢复目录

在图19.10所示界面上，选择【备份/恢复】选项中的恢复目录设置，可以添加恢复目录，如图19.11所示。在此页面中，可以选择恢复目录的方式，一种方式是目标数据库的控制文件；另一种方式就是使用恢复目录。在这里为了讲解恢复目录的创建，单击【添加恢复目录】按钮，如图19.12所示。



图19.11 恢复目录设置页面

在此，可以看到5个必填的内容，“主机”是指恢复目录所在的数据库计算机名或者计算机的IP地址；“端口”是指恢复目录所在数据库的端口号，这个端口号一般都是1521；SID是指数据库的服务名，也是数据库的实例名；“恢复目录用户名”就是连接恢复目录数据库的用户名；“恢复目录口令”就是连接恢复目录数据库的密码。这里，按照要求输入恢复目录信息，如图19.13所示。



图 19.12 添加恢复目录



图 19.13 添加恢复目录信息

在此，添加好了恢复目录的信息，然后单击【下一步】按钮，出现图19.14所示的界面。此时可以看到恢复目录数据库的配置和恢复目录的用户，单击【完成】按钮，即可完成恢复目录的添加，如图19.15所示。

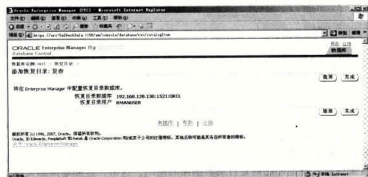


图 19.14 复查恢复目录



图19.15 添加恢复目录完成

添加好恢复目录后，就可以在恢复目录设置中选择【使用恢复目录】选项来备份数据库了。

19.3 通道分配

通道分配是在使用RMAN备份时必须提到的概念，通道分配分为手动和自动两种方式。本节将讲述通道分配的概念以及手动通道分配与自动通道分配方法。

19.3.1 什么是通道分配

什么是通道呢？通道就是指数据库与某一个设备关联，这个设备就是指存储的介质——磁带或磁盘。目前存储介质使用最多的还是磁带。通道分配就是确定连接数据库备份的设置个数，每设置一个设备就代表RMAN会自动启动一个服务器会话，由此来完成数据库的备份与恢复的操作。通道分配有手动分配和自动分配两种，这两种方式使用的命令不同：在手动分配通道时要使用RUN命令实现，而自动分配则只需要使用CONFIGURE命令即可完成。

19.3.2 手动通道分配

手动通道分配可以根据用户自定义分配通道完成数据库的备份工作，并且如果在数据库中定义了手动通道分配，会优先使用手动分配的通道。这里，要使用手动通道分配就要学习RUN命令。RUN命令的一般语法如下：

```
RUN
{
  ALLOCATE CHANNEL channel_name1 DEVICE TYPE type_name1;
  ALLOCATE CHANNEL channel_name2 DEVICE TYPE type_name2;
  ...
  BACKUP ...
}
```

【语法说明】

□ channel_name：分配的通道名称，type_name是分配的设备类型，这个设备类型是磁带

(sbt) 和磁盘 (disk)，并且可以手动分配多个通道，即可以含有多个 ALLOCATE CHANNEL 语句。

❑ BACKUP：备份数据库的关键字，可以在 BACKUP 后面写上要备份的表空间等信息。下面就利用上面的语法完成数据库的备份操作。

【示例9】使用磁盘备份表空间 USERS

代码如下：

```
RUN
{
  ALLOCATE CHANNEL C DEVICE TYPE disk;
  BACKUP tablespace USERS
}
```

在具体操作时使用 BACKUP 备份数据库中的文件，如果直接利用上面的语句，则会出现图 19.16 所示的错误。



图 19.16 备份时出现错误

出现上面的错误主要是因为数据库的状态是正在运行的，不能够备份数据库中的文件。如果要备份数据库中的文件，可以先关闭数据库，然后把数据库启动到 MOUNT 的状态。这样就可以完成数据库的备份操作。具体操作过程如图 19.17 所示。



图 19.17 使用通道备份文件

至此，表空间USERS已经备份成功。

注意 在使用RUN命令前一定要在RMAN命令状态下，并且确保已经连接到目标数据库。



【示例10】使用多个通道备份表空间USERS

在定义通道的语法中可以看到，通道是可以同时定义多个的。也就是说，可以用多个通道完成数据库的备份工作。下面就建立两个通道备份表空间USERS。代码如下：

```
RUN
{
  ALLOCATE CHANNEL C1 DEVICE TYPE disk;
  ALLOCATE CHANNEL C2 DEVICE TYPE disk;
  BACKUP tablespace USERS
}
```

这里，两个通道的名字是不能重复的。为了让读者理解使用通道备份的过程，下面分步演示运行效果。

(1) 连接到目标数据库

在命令窗口下直接输入RMAN命令，即可连接到目标数据库，如图19.18所示。



图19.18 连接到目标数据库

(2) 关闭数据库并启动MOUNT

为了让数据库正常完成备份的操作，需要关闭数据库。具体操作如图19.19所示。



图19.19 关闭数据库

(3) 使用RUN命令创建通道并备份表空间USERS

在数据库关闭的状态下可以开始备份表空间了，效果如图19.20所示。

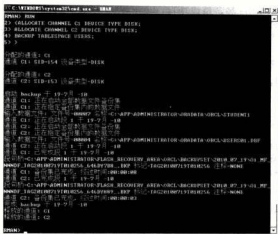


图19.20 使用多通道备份表空间

至此，使用多通道备份表空间完成。从这里可以看出，在两个通道中都备份了表空间USERS，备份后再依次释放通道。对于其他备份操作，在19.4.2小节中将详细介绍。

19.3.3 自动通道分配

前面已经介绍了手动通道分配，实际上不使用手动通道分配的方法，数据库就会使用自动通道分配的方式来备份数据库。自动通道分配时通道名称由数据库自己指定，具体的命令如下：

```
01 CONFIGURE DEVICE TYPE type_name PARALLELISM n
02 CONFIGURE DEFAULT DEVICE TYPE TO type_name
```

【语法说明】

- 第1行用来指定设备的类型以及通道的个数，type_name是类型的名称，n代表通道的个数。
- 第2行用来指定默认设备的类型，如果使用的设备基本都是磁盘，那么可以把默认设备设置成磁盘。

【示例11】自动分配通道

下面使用上面的命令完成表空间USERS的备份，创建两个通道备份。先自动分配两个通道，分配方法如下：

```
CONFIGURE DEVICE TYPE disk PARALLELISM 2
```

运行效果如图19.21所示。



图19.21 自动分配通道

使用自动分配的通道备份表空间USERS，具体操作如图19.22所示。

【示例12】 利用BACKUP命令备份数据库

BACKUP database

备份效果如图19.23所示。

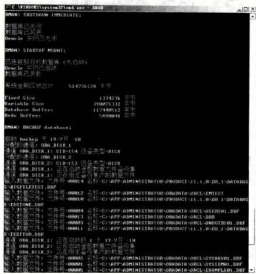


图19.23 备份数据库

在使用BACKUP命令备份数据库之前，也需要把数据库关闭并启动mount，才可以完成备份的操作。图19.23中由于备份的内容过多没有完全显示，当备份操作全部结束后，应该显示一个“完成BACKUP”的信息，并重新回到RMAN>下。

19.5 从备份中恢复

前几节中已经介绍了如何使用RMAN备份数据库，那么备份好的数据库如何恢复呢？本节就将回答这个问题。

19.5.1 使用RESTORE还原

使用RESTORE命令可以还原备份的信息。RESTORE命令的具体语法如下：

RESTORE database_object *— 从备份中还原数据库对象*

【语法说明】

database_object: 数据库对象, 可以是DATABASE (数据库)、TABLESPACE (表空间)。

DATAFILE (数据文件)、CONTROLFILE (控制文件)、ARCHIVELOG (归档日志)、SPFILE (参数文件)。其中, 只能在MOUNT状态下使用的对象有DATABASE、CONTROLFILE、SPFILE, 只能在OPEN状态下使用的对象是TABLESPACE。

【示例13】还原数据库

还原数据库代码如下:

```
RESTORE DATABASE
```



读书笔记

注意 在还原数据库时, 要保证数据库的状态是MOUNT。

19.5.2 使用RECOVER恢复

使用RECOVER命令可以恢复数据库, 该命令是负责把归档执行日志文件用于重建的数据文件, 来完成数据库的同步恢复。但是在执行RECOVER命令时RMAN还需要读取归档日志, 在恢复数据库时尽量要把数据库的状态设置成归档模式。RECOVER的命令如下:

```
RECOVER database_object
```

【语法说明】

database_object: 可以是DATABASE (数据库)、TABLESPACE (表空间)、DATAFILE (数据文件)。其中, DATABASE只能在MOUNT状态下使用, TABLESPACE只能在OPEN状态下使用。

【示例14】恢复数据库

下面利用上面的语句恢复数据库。代码如下:

```
RECOVER DATABASE
```

注意 在恢复数据库时, 要保证数据库的状态是MOUNT。

19.6 小结

通过本章的学习, 读者可以了解到恢复目录的创建、通道的自动分配和手动分配、备份集的概念以及RMAN中常用的命令RUN、BACKUP、RESTORE、RECOVER等的使用。在了解了这些命令的基础上, 要熟练掌握使用RUN命令分配通道和使用BACKUP命令备份数据库等常用的操作。读者可以根据RMAN的备份与恢复知识自行备份和恢复自己的数据库。

19.7 习题

简答题

1. 分别使用企业管理器和命令创建恢复目录。
2. 创建两个通道备份命名空间SYSTEM。
3. 什么是备份集? 备份片的大小有限制吗?
4. 使用BACKUP命令都能够备份哪些数据库对象?
5. 使用RESTORE和RECOVER命令还原与恢复数据库。

使用 RMAN 命令进行数据库恢复。RMAN 命令用于管理数据库的备份和恢复。RMAN 命令可以用于创建备份、恢复数据库、恢复数据文件、恢复控制文件、恢复归档日志等。RMAN 命令可以用于恢复数据库的整个实例，也可以用于恢复数据库的特定部分。RMAN 命令可以用于恢复数据库的整个实例，也可以用于恢复数据库的特定部分。RMAN 命令可以用于恢复数据库的整个实例，也可以用于恢复数据库的特定部分。

6.1 数据库恢复

数据库恢复是指将数据库恢复到一致状态的过程。数据库恢复可以分为物理恢复和逻辑恢复。物理恢复是指将数据库的物理文件恢复到一致状态的过程。逻辑恢复是指将数据库的逻辑文件恢复到一致状态的过程。数据库恢复可以分为物理恢复和逻辑恢复。物理恢复是指将数据库的物理文件恢复到一致状态的过程。逻辑恢复是指将数据库的逻辑文件恢复到一致状态的过程。

6.2 数据库恢复的步骤

数据库恢复的步骤包括：1. 检查数据库的状态。2. 确定恢复的目标。3. 恢复数据库。4. 验证数据库。5. 恢复数据文件。6. 恢复控制文件。7. 恢复归档日志。8. 恢复数据库的整个实例。9. 恢复数据库的特定部分。10. 恢复数据库的整个实例。11. 恢复数据库的特定部分。12. 恢复数据库的整个实例。13. 恢复数据库的特定部分。

6.3 数据库恢复的命令

数据库恢复的命令包括：1. RMAN 命令。2. SQL 命令。3. 操作系统命令。4. 数据库命令。5. 恢复命令。6. 恢复数据文件命令。7. 恢复控制文件命令。8. 恢复归档日志命令。9. 恢复数据库的整个实例命令。10. 恢复数据库的特定部分命令。11. 恢复数据库的整个实例命令。12. 恢复数据库的特定部分命令。

第四篇

数据库应用篇

第20章 在线考试系统数据库设计

学习数据库的目的除了管理数据之外,就是进行数据库的设计。本章将完成在线考试系统的数据库设计,该系统的适用群体是学校或培训机构。

20.1 在线考试系统需求

所谓需求分析,就是指分析软件用户的需求是什么。现实生活中不管做什么,肯定需要一个理由,那么需求分析就是我们做软件的理由,它可以告诉我们客户为什么要做这个软件,以及客户想要把这个软件做成什么样的。

做软件不可以闭门造车,如果我们花费了大量的人力、物力、财力,而开发出来的软件却不是客户需要的,那么我们所做的工作都是徒劳的。如何避免这种情况呢?需求分析就是解决该类问题的首要条件。下面就对在线考试系统进行整体的分析。

开发该系统的目的就是方便老师和学生,为老师和学生提供一个平台。利用该系统,老师可以足不出户就能为学生制定要考核的内容,而学生也可以足不出户地利用该系统的题目检测自己的知识掌握程度,这样不方便出行的学生就可以在家里完成测试。

既然是考试系统,那么就要有出题的一方和做题的一方。系统中要求老师出检测试卷和判卷,而学生只可以答卷和提问。在这两类对象外,还需要有管理者,它们可以协调整个考试过程,例如安排测试时间,指定出题教师,以及指定判卷老师等。作为一个系统还需要有一个权限管理者,他可以维护系统的基础信息,例如在数据库中增加老师、管理者或学生。系统架构图见图20.1。

与该系统有关的流程如图20.2所示。

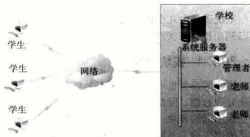


图20.1 系统架构图



图20.2 系统流程图

20.2 模块设计

系统通常由几个模块组成，每个模块负责自己专有的功能，如果一个模块调用另一个模块，可以利用接口来实现，这样的系统才能健壮，易于维护。

20.2.1 模块分类

模块的分类也可以让使用人员更方便，由于职责所在，每个不同职责的人只允许在指定的模块处理数据。除此之外，分成模块，也能使开发变得更具有目的性，指定不同的开发组或个人来完成不同的模块，这样就避免在开发过程中出现过多的交集。如果开发的项目交集过多，那么可能出现分工不明确、工作效率低下等情况。

在线考试系统根据需求可以分成如下几个模块：用户权限管理模块、出题模块、考试模块、考试管理模块、判卷模块、教学管理模块、基本信息模块。

各模块功能介绍如下：

1) 用户权限管理模块。该模块主要针对已有的用户,由于每个用户的职责不同,所以允许他们所做的操作也不相同。该模块中利用角色来给用户分配权限,用户拥有什么角色,他就能做相关操作。其他操作将不被允许。

2) 出题模块。该模块只允许出题老师登录,老师可以根据实际的教学计划进行试卷编写。对于选择题和判断题,可以利用系统提供的模板完成。

3) 考试模块。学生登录成功后,进行信息验证,符合考试资格则可见考试信息,此时不能答卷,当到计时指定时间方可答卷。当考试时间剩余15分钟时,系统提示考生剩余时间。交卷分为人工提交试卷和系统自动提交。

4) 考试管理模块。教学管理人员或老师可以使用该模块功能,利用该模块可以管理考试试卷和学生提交的试卷,并提供现场在线咨询功能。

5) 判卷模块。判卷老师可以登录,进行判卷。

6) 教学管理模块。教学管理者可以操作。在该模块中可以指定试卷的出题教师、考试时间和判卷教师,列出考试安排的明细信息。

7) 基本信息模块。包括用户基本信息。用户是指所有用户以及各种字典表。

20.2.2 数据库总体结构

该系统中一共包含了16张表,如表20.1所示。

表20.1 系统包含的表

序 号	表名(英文)	说 明
1	Sys_User	用户信息表
2	Sys_Roles	角色表
3	Sys_Resource	资源表
4	Sys_User_Roles	用户角色表
5	Sys_Role_Resource	角色资源表
6	Teachers	教师表
7	Managers	教务人员表
8	Duty	职务表
9	Students	学生表
10	Courses	课程表
11	CoursesType	课程类型表
12	Student_Course	学生选课关系表
13	Exam	考试表
14	ExamPaper	试卷表
15	AnswerPaper	学生答卷表
16	Graders_Exam	考试批卷关系表

20.2.3 数据库表结构

本小节列出了16张表的表结构。利用表结构,开发人员可以快速地了解表的字段以及业务。



1. 用户信息表 (Sys_User)

用户信息表记录了所有的系统中用户的信息，并标明该用户是否可以用，如表20.2所示。

表20.2 用户信息表 (Sys_User)

字 段	数据类型	长 度	允许空	说 明
userId	varchar2	10	N	ID，主键
userName	varchar2	20	N	用户名
passWord	varchar2	32	N	密码
identity	varchar2	10	N	身份
userState	char	1	N	用户状态：1—正常，0—禁用
remark	varchar2	200	Y	备注

2. 角色表 (Sys_Roles)

角色表定义了系统中可能用到的角色，如表20.3所示。

表20.3 角色表 (Sys_Roles)

字 段	数据类型	长 度	允许空	说 明
roleId	varchar2	10	N	ID，主键
roleName	varchar2	20	N	角色名

3. 资源表 (Sys_Resource)

资源表定义了系统中的所有资源，也就是URL，可以为角色指定拥有的URL操作，如表20.4所示。

表20.4 资源表 (Sys_Resource)

字 段	数据类型	长 度	允许空	说 明
resourceId	varchar2	10	N	ID，主键
ResourceName	varchar2	50	N	资源名称
ResourceUrl	varchar2	100	Y	URL地址

4. 用户角色表 (Sys_User_Roles)

用户角色表结构如表20.5所示。

表20.5 用户角色表 (Sys_User_Roles)

字 段	数据类型	长 度	允许空	说 明
UserId	varchar2	10	N	引用自Sys_User表主键，主键
RoleId	varchar2	10	N	引用自Sys_Roles表主键，主键

5. 角色资源表 (Sys_Role_Resource)

角色资源表结构如表20.6所示。

表20.6 角色资源表 (Sys_Role_Resource)

字 段	数据类型	长 度	允许空	说 明
RoleId	varchar2	10	N	引用自Sys_Roles表主键, 主键
ResourceId	varchar2	10	N	引用自Sys_Resource表主键, 主键



6. 教师表 (Teachers)

教师表结构如表20.7所示。

表20.7 教师表 (Teachers)

字 段	数据类型	长 度	允许空	说 明
teacherId	varchar2	10	N	教师编号, 主键
teacherName	varchar2	20	N	教师姓名
duty	varchar2	100	Y	职务, 可以存储多个职务编码
remark	varchar2	200	Y	备注

7. 教务人员表 (Managers)

教务人员表结构如表20.8所示。

表20.8 教务人员表 (Managers)

字 段	数据类型	长 度	允许空	说 明
managerId	varchar2	10	N	教务人编号, 主键
managerName	varchar2	20	N	教务人姓名
duty	varchar2	100	Y	职务, 可以存储多个职务编码
remark	varchar2	10	Y	备注

8. 职务表 (Duty)

职务表结构如表20.9所示。

表20.9 职务表 (Duty)

字 段	数据类型	长 度	允许空	说 明
dutyId	varchar2	10	N	职务ID, 主键
dutyName	varchar2	50	Y	职务名称
remark	varchar2	200	Y	备注

9. 学生表 (Students)

学生表结构如表20.10所示。



表20.10 学生表 (Students)

字 段	数据类型	长 度	允许空	说 明
studentId	varchar2	10	N	学生学号, 主键
studentName	varchar2	20	Y	学生姓名
Grade	number	(3,0)	Y	年级
className	varchar2	20	Y	班级名称

10. 课程表 (Courses)

课程表结构如表20.11所示。

表20.11 课程表 (Courses)

字 段	数据类型	长 度	允许空	说 明
courseId	varchar2	10	N	课程编号, 主键
courseName	varchar2	10	N	课程名称
coursesType	varchar2	4	N	课程类型

11. 课程类型表 (CoursesType)

课程类型表结构如表20.12所示。

表20.12 课程类型表 (CoursesType)

字 段	数据类型	长 度	允许空	说 明
coursesTypeId	varchar2	4	N	课程类型编号, 主键
coursesType	varchar2	20	N	课程类型名称

12. 学生选课关系表 (Student_Course)

学生选课关系表结构如表20.13所示。

表20.13 学生选课关系表 (Student_Course)

字 段	数据类型	长 度	允许空	说 明
studentId	varchar2	10	N	表主键, 主键
courseId	varchar2	10	N	表主键, 主键
term	varchar2	20	N	选课学期

13. 考试表 (Exam)

考试表结构如表20.14所示。

表20.14 考试表 (Exam)

字 段	数据类型	长 度	允许空	说 明
examId	varchar2	10	N	ID, 主键
startDate	date		N	考试开始时间

(续)

字段	数据类型	长度	允许空	说明
endDate	date		N	考试结束时间
courseId	varchar2	10	N	课程ID
publisherId	varchar2	10	Y	出题教师ID
term	varchar2	20	Y	学期



14. 试卷表 (ExamPaper)

试卷表结构如表20.15所示。

表20.15 试卷表 (ExamPaper)

字段	数据类型	长度	允许空	说明
paperId	varchar2	10	N	考试试卷的ID, 主键
examId	varchar2	10	N	考试ID
courseId	varchar2	10	N	课程ID
publisherId	varchar2	8	N	出卷人
paperCode	varchar2	10	Y	卷号
duration	number	8	Y	考试时长
totalScore	number	(8,1)	Y	总分
submitState	number	2	N	出卷状态
uri	varchar2	200	Y	试卷存放地址

15. 学生答卷表 (AnswerPaper)

学生答卷表结构如表20.16所示。

表20.16 学生答卷表 (AnswerPaper)

字段	数据类型	长度	允许空	说明
answerPaperId	varchar2	10	N	学生答卷ID, 主键
studentId	varchar2	10	N	学生ID
examId	varchar2	10	N	考试ID
totalScore	number	(8,1)	Y	学生得分
availability	char	1	Y	是否有效
submitTime	timestamp		N	交卷时间
isfinish	char	1	Y	是否交卷
uri	varchar	200	Y	答卷地址

16. 考试批卷关系表 (Graders_Exam)

考试批卷关系表结构如表20.17所示。

表20.17 考试批卷关系表 (Graders_Exam)

字 段	数据类型	长 度	允许空	说 明
Id	varchar2	10	N	ID, 主键
paperId	varchar2	10	N	考试试卷ID
teacherId	varchar2	10	N	判卷教师ID
subjectNumber	number	3	N	试卷中的大题号

20.2.4 建表脚本

本小节写出了创建16张表的建表脚本。具体内容如下。

1. 用户信息表 (Sys_User)

建表脚本如下：

```
CREATE TABLE SYS_USER
(
    USERID VARCHAR2(10) NOT NULL,
    USERNAME VARCHAR2(20) NOT NULL,
    PASSWORD VARCHAR2(32) NOT NULL,
    IDENTITY VARCHAR2(10) NOT NULL,
    USERSTATE CHAR(1) NOT NULL,
    REMARK VARCHAR2(200),
    PRIMARY KEY (USERID)
);
```

2. 角色表 (Sys_Roles)

建表脚本如下：

```
CREATE TABLE SYS_ROLES
(
    ROLEID VARCHAR2(10) NOT NULL,
    ROLENAME VARCHAR2(20) NOT NULL,
    PRIMARY KEY (ROLEID)
);
```

3. 资源表 (Sys_Resource)

建表脚本如下：

```
CREATE TABLE SYS_RESOURCE
(
    RESOURCEID VARCHAR2(10) NOT NULL,
    RESOURCENAME VARCHAR2(50) NOT NULL,
    RESOURCEURL VARCHAR2(100),
    PRIMARY KEY (RESOURCEID)
);
```



4. 用户角色表 (Sys_User_Roles)

建表脚本如下:

```
CREATE TABLE SYS_USER_ROLES
```

```
(
    USERID VARCHAR2(10) NOT NULL,
    ROLEID VARCHAR2(10) NOT NULL
);
```

5. 角色资源表 (Sys_Role_Resource)

建表脚本如下:

```
CREATE TABLE SYS_ROLE_RESOURCE
```

```
(
    ROLEID VARCHAR2(10) NOT NULL,
    RESOURCEID VARCHAR2(10) NOT NULL
);
```

6. 教师表 (Teachers)

建表脚本如下:

```
CREATE TABLE TEACHERS
```

```
(
    TEACHERID VARCHAR2(10) NOT NULL,
    TEACHERNAME VARCHAR2(20) NOT NULL,
    DUTY VARCHAR2(10),
    REMARK VARCHAR2(200),
    PRIMARY KEY (TEACHERID)
);
```

7. 教务人员表 (Managers)

建表脚本如下:

```
CREATE TABLE MANAGERS
```

```
(
    MANAGERID VARCHAR2(10) NOT NULL,
    MANAGERNAME VARCHAR2(20) NOT NULL,
    DUTY VARCHAR2(10),
    REMARK VARCHAR2(200),
    PRIMARY KEY (MANAGERID)
);
```

8. 职务表 (Duty)

建表脚本如下:

```
CREATE TABLE DUTY
```

```
(
    DUTYID VARCHAR2(10) NOT NULL,

```



```

DUTYNAME VARCHAR2(50),
REMARK VARCHAR2(200),
PRIMARY KEY (DUTYID)
);
    
```

9. 学生表 (Students)

建表脚本如下:

```

CREATE TABLE STUDENTS
(
    STUDENTID VARCHAR2(10) NOT NULL,
    STUDENTNAME VARCHAR2(20),
    GRADE NUMBER(3,0),
    CLASSNAME VARCHAR2(20),
    PRIMARY KEY (STUDENTID)
);
    
```

10. 课程表 (Courses)

建表脚本如下:

```

CREATE TABLE COURSES
(
    COURSEID VARCHAR2(10) NOT NULL,
    COURSENAME VARCHAR2(10) NOT NULL,
    COURSESTYPE VARCHAR2(4) NOT NULL,
    PRIMARY KEY (COURSEID)
);
    
```

11. 课程类型表 (CoursesType)

建表脚本如下:

```

CREATE TABLE COURSESTYPE
(
    COURSESTYPEID VARCHAR2(4) NOT NULL,
    COURSESTYPE VARCHAR2(20),
    PRIMARY KEY (COURSESTYPEID)
);
    
```

12. 学生选课关系表 (Student_Course)

建表脚本如下:

```

CREATE TABLE STUDENT_COURSE
(
    STUDENTID VARCHAR2(10) NOT NULL,
    COURSEID VARCHAR2(10) NOT NULL,
    TERM VARCHAR2(20)
);
    
```



13. 考试表 (Exam)

建表脚本如下:

CREATE TABLE EXAM

```
(
    EXAMID VARCHAR2(10) NOT NULL,
    STARTDATE DATE NOT NULL,
    ENDDATE DATE NOT NULL,
    COURSEID VARCHAR2(10) NOT NULL,
    PUBLISHERID VARCHAR2(10),
    TERM VARCHAR2(20),
    PRIMARY KEY (EXAMID)
);
```

14. 试卷表 (ExamPaper)

建表脚本如下:

CREATE TABLE EXAMPAPER

```
(
    PAPERID VARCHAR2(10) NOT NULL,
    EXAMID VARCHAR2(10) NOT NULL,
    COURSEID VARCHAR2(10) NOT NULL,
    PUBLISHERID VARCHAR2(8) NOT NULL,
    PAPERCODE VARCHAR2(10),
    DURATION NUMBER(8,0),
    TOTALSCORE NUMBER(8,1),
    SUBMITSTATE NUMBER(2,0) NOT NULL,
    URI VARCHAR2(200),
    PRIMARY KEY (PAPERID)
);
```

15. 学生答卷表 (AnswerPaper)

建表脚本如下:

CREATE TABLE ANSWERPAPER

```
(
    PAPERID VARCHAR2(10) NOT NULL,
    STUDENTID VARCHAR2(10) NOT NULL,
    EXAMID VARCHAR2(10) NOT NULL,
    TOTALSCORE NUMBER(8,1),
    AVAILABILITY CHAR(1) NOT NULL,
    SUBMITTIME TIMESTAMP NOT NULL,
    ISFINISH CHAR(1) NOT NULL,
    URI VARCHAR2(200),
    PRIMARY KEY (PAPERID)
);
```



16. 考试批卷关系表 (Graders_Exam)

建表脚本如下：

```
CREATE TABLE GRADERS_EXAM (
    ID VARCHAR2(10) NOT NULL,
    PAPERID VARCHAR2(10) NOT NULL,
    TEACHERID VARCHAR2(10) NOT NULL,
    SUBJECTNUMBER NUMBER(3,0) NOT NULL,
    PRIMARY KEY (ID)
);
```

20.3 小结

本章用一个实例系统介绍了如何设计数据库，设计数据库前一定要有比较充分的需求分析，做到基本了解用户的需求。设计数据库时，要按照模块的形式进行设计，做到业务分明，减少耦合度，使得系统更加健壮。设计数据库时不可避免地要考虑各表之间的关系，如非必要，设计时尽量把表的主外键关系放到程序中处理，而不是在数据库里处理，这样可以在一定程度上提高数据库的性能。

20.4 习题

思考题

设计一个订餐系统的数据库，要求用户只能由系统管理员创建，普通用户能从已有的餐饮列表中选取食品，并能查看本人某时间段内的订餐明细记录和存款情况，也能查看其他人员当天的订餐具体情况。至于存款情况，只能由订餐人员操作。

第21章 在.NET中连接Oracle

目前使用.NET是主流的开发平台，Oracle是企业级用户首选的数据库产品，在.NET中连接Oracle数据库也是非常方便的。本章将学习如何在.NET中连接数据库。本章包括以下知识点：

- ☐ ADO.NET概述
- ☐ 使用绑定的方式连接Oracle数据库
- ☐ 使用代码的方式连接Oracle数据库

本章内容基本涵盖了在.NET中常用的连接Oracle的方法。通过本章的学习，可以熟练地在.NET环境中连接Oracle数据库。

21.1 什么是ADO.NET

ADO.NET就相当于.NET中连接数据库的数据接口，使用ADO.NET就可以在.NET的环境中连接各种数据库。本节将讲述ADO.NET的基本概念以及ADO.NET中的对象。

21.1.1 ADO.NET概述

ADO.NET并不是ADO简单的升级，使用ADO.NET可以访问的数据源不仅仅是Oracle数据库，还可以是SQL Server数据库、Access数据库等常见的数据库。此外，它还可以提供对XML以及Excel这样的数据源进行访问。

21.1.2 ADO.NET中的对象

ADO.NET组件由数据提供程序和数据集两部分组成，数据提供程序包括Connection、Command、DataReader、DataAdapter 4个对象，数据集就是指DataSet对象。下面就分别讲解这5个对象的概念以及使用方法。

1. Connection对象

Connection对象是连接对象，主要用来根据提供的数据库连接串连接数据库，对数据库连接的操作提供了打开数据库连接和关闭数据库连接两个方法。此外，在创建连接不同的数据库时，Connection对象要引入的命名空间以及名字也略有区别。例如，连接Oracle数据库使用OracleConnection，连接SQL Server数据库时使用的是SqlConnection。

2. Command对象

Command对象是命令对象，主要用来实现对数据库的操作，包括增加、删除、修改以及查询的操作。同样，Command对象根据不同的数据库命名空间以及对象的名字也有一些区别。但是无论是对哪个数据库中的数据进行管理，其SQL语句是不变的。例如，连接Oracle数据库使用的是OracleCommand，连接SQL Server数据库使用的是SqlCommand。

3. DataReader对象

DataReader对象是数据读取对象，它的主要作用是接收使用Command命令查询得到的结果。同样，在接收不同数据库中的查询结果时使用的对象名也是有区别的。例如，读取Oracle数据库中的查询结果使用的是OracleReader对象，读取SQL Server数据库中的查询结果使用的是SqlDataReader。但是DataReader对象的特点是只能按顺序来读取数据库中的数据。

4. DataAdapter对象

DataAdapter对象是数据适配器对象，它的主要作用是把从数据库中查询出来的结果填充到下面要讲的DataSet数据集对象中。根据查询的不同，数据库DataAdapter的名字也不同。例如，读取Oracle数据库中的数据使用的是OracleDataAdapter，而读取SQL Server数据库中的数据使用的是SqlDataAdapter。

5. DataSet对象

DataSet对象是数据集对象，它是数据库中的虚拟的表。把查询出来的数据通过DataAdapter数据适配器对象填充到DataSet数据集对象中后，就可以对DataSet数据集中的数据进行增加、删除、修改以及查询的操作。此时把数据库中的数据填充到数据集之后，数据集就与数据库断开使用了，所以在对数据集中的数据进行操作后是不会改变数据库中的数据的。如果要改变数据库中的数据，还需要在对数据集中的数据修改后使用数据集的更新方法把数据更新到数据库中。这种使用DataSet来存放数据库中的查询结果并对数据库中的数据进行操作的方法称为断开式数据连接。这也是ADO.NET中的一个亮点。由于本书主要是讲解Oracle的，这部分内容就不再详细讲解了，对.NET感兴趣的读者可以参考其他.NET方面的书籍学习。

21.2 使用绑定的方式连接Oracle

Oracle数据库在.NET编程中应用也是比较普遍的，本节以Windows应用程序为例，讲解如何使用Visual Studio 2008中自带的数据库控件连接Oracle数据库。

21.2.1 数据库控件概述

数据库控件是指在Visual Studio 2008中工具箱里“数据”一栏中的控件，如图21.1所示。

在工具箱“数据”一栏中的控件主要有DataSet数据集控件、DataGridView数据表格控件、BindingSource绑定数据源控件以及BindingNavigator绑定导航控件。其中，BindingSource与BindingNavigator两个控件经常搭配到一起使用。

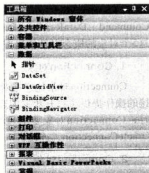


图21.1 数据库控件

21.2.2 使用DataGridView控件绑定Oracle数据库

DataGridView是开发Windows应用程序中必不可少的组件之一，本小节以DataGridView为例讲解绑定Oracle数据库的全过程。使用DataGridView控件绑定Oracle数据库分为以下几

个步骤。

1. 创建一个Windows应用程序项目

创建Windows应用程序项目非常简单，只需要在Visual Studio 2008中选择【文件】|【新建】|【项目】选项，即会出现图21.2所示界面。



图21.2 新建项目

在图21.2所示页面中，可以选择项目存放的位置，然后单击【确定】按钮，即可成功创建一个Windows窗体应用程序项目。

2. 在窗体上添加DataGridView控件

创建好项目后，打开一个窗体，从工具箱中拖曳一个DataGridView控件到窗体中，并适当地改变窗体和DataGridView控件的位置，如图21.3所示。

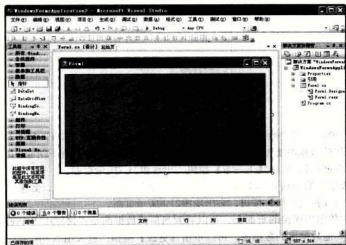


图21.3 添加窗体页面



在图21.3所示页面中，可以看到创建一个Windows应用程序的项目目录。如果需要在项目中继续添加其他窗体，可以在项目文件名上右击，从弹出的快捷菜单中选择【新建项】命令。

3. 添加数据源

1) 添加窗体后，右击DataGridView右上角的箭头，开始为DataGridView添加数据源，进入图21.4所示页面。在DataGridView操作菜单页面上，单击【选择数据源】下拉列表框，出现图21.5所示页面。

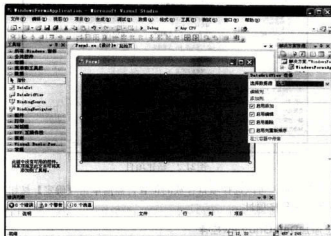


图21.4 DataGridView操作菜单



图21.5 选择数据源

2) 由于当前没有可选数据源，这里单击【添加项目数据源】选项，如图21.6所示。在这里可以看到可用的数据源类型，这些类型有数据库、服务、对象。由于Oracle属于数据库，所以这里选择【数据库】，单击【下一步】按钮，出现图21.7所示页面。

3) 选择数据库的连接实际上就是创建一个与数据库连接的连接串，如果当前没有可选的连接，那么单击【新建连接】按钮，会出现图21.8所示页面。

4) 在添加数据库连接的页面上，可以看到当前的数据源是连接SQL Server数据库的，如果要选择连接其他数据库的数据源，可以单击【更改】按钮，进入图21.9所示页面。

5) 在此页面上的数据源列表框中选择【Oracle数据库】选项，进入图21.10所示页面。

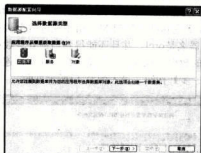


图21.6 选择数据源类型

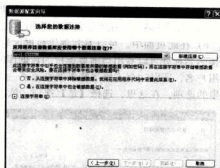


图21.7 选择数据库的连接

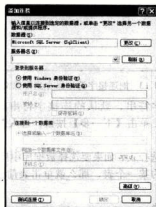


图21.8 添加数据库连接



图21.9 更改数据源

6) 在此,添加Oracle数据库的服务器名,这个服务器名是在安装Oracle数据库时创建的。接着添加登录到数据库的用户名和密码,这里由于SYS用户是系统管理员用户,所以不能在这里直接使用,可以使用system、scott等Oracle数据库中默认的用户,也可以使用自定义的用户登录。这里,使用system用户登录,如图21.11所示。



图21.10 添加Oracle数据库连接



图21.11 添加连接信息



7) 在图21.11所示页面中, 添加完连接数据库的相关信息后, 单击【测试连接】按钮, 可以测试当前连接是否成功。在添加完数据库的连接之后回到【数据源配置向导】页面, 如图21.12所示。

8) 在此页面中, 可以看到创建数据库连接的名称是orcl.SYSTEM。由于通过这个数据库连接的名称就可以看出当前要连接的数据库的服务器名是orcl, 并且还能得到当前登录数据库的用户名, 所以这个连接字符串就包含了一些敏感的数据, 也因此会有是否继续使用该连接字符串的选项。在这里, 选择【是】选项, 单击【下一步】按钮, 进入图21.13所示页面。



图21.12 数据源配置向导



图21.13 把连接字符串保存到配置文件中

9) 在此, 可以设置保存到配置文件中的连接字符串, 默认的连接字符串的名称是ConnectionString, 这里修改成OracleConnectionString。单击【下一步】按钮, 进入图21.14所示页面。

在此, 可以选择数据库中的对象表或者视图。这里选择表, 如果要选择数据库中所有的表, 则直接单击表前面的复选框即可; 如果选择数据库中的某一个表, 那么就单击表前面的“+”号, 查看数据库中的表, 然后选择, 如图21.15所示。



图21.14 选择数据库对象

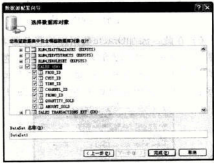


图21.15 选择表

10) 这里选择数据库中的SALES表, 单击【完成】按钮, 即可完成对DataGridView数据源的创建。创建好数据源后, DataGridView就与选中的SALES表绑定上了, 在绑定表的同时也可以选择绑定表中的列。如果再选中表前面的复选框, 那么就相当于选中表中的所有列。界面如图21.16所示。

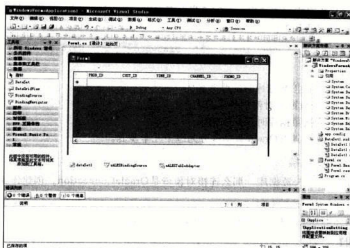


图21.16 绑定DataGrid View

此时，已经完成了绑定DataGrid View的工作，下面运行该窗体看一下效果，如图21.17所示。

这样就完成了DataGrid View与Oracle数据库的连接操作。

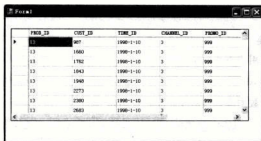


图21.17 运行效果

21.3 使用写代码的方式连接Oracle

使用写代码的方式连接数据库的原理实际上与直接绑定是一样的，这里需要使用前面介绍的ADO.NET的5个对象。本节将讲述如何使用这5个对象完成对数据库的操作。

21.3.1 使用Command对象操作Oracle数据库

使用Command对象操作Oracle数据库，在查询数据库操作时是把结果存放在DataReader对象中的。无论是使用什么方法连接数据库，都必须保证使用到的对象就是数据库的连接对象Connection。在使用Connection对象连接数据库时，需要使用连接数据库的连接串。本小节主



要介绍连接Oracle数据库的连接串。在21.2节中绑定DataGridView时，就配置过数据库的连接串。一般的连接串写法如下：

```
Data Source="服务名" User ID="用户名" Password="密码" Unicode="TRUE"
```

【语法说明】

- ☐ Data Source：连接数据库时的服务名。
- ☐ User ID：连接数据库时使用的用户名。
- ☐ Password：连接数据库时用户的密码。

使用Command对象连接Oracle数据库分为以下几个步骤。

1. 创建数据库的连接

由于连接的是Oracle数据库，那么连接对象就是OracleConnection。创建连接对象的方法如下：

```
OracleConnection 对象名 = new OracleConnection("连接字符串");
```

【语法说明】

- ☐ 对象名：可以是任意的名称，但是定义的对象名要以字母开头或者下划线开头，不能以数字开头。
- ☐ 连接字符串：连接串就是连接数据库的字符串，可以直接把连接串写到括号里，也可以先定义一个变量来存放连接字符串，再把这个变量放在括号里。

例如，创建一个对象名是conn的数据库连接对象。代码如下：

```
OracleConnection conn = new OracleConnection();
```

2. 打开数据库连接

打开数据库连接使用的方法如下：

```
创建的数据库连接名称.Open();
```

例如，打开上述的数据库连接对象conn。代码如下：

```
conn.Open();
```

3. 使用command命令操作数据库

对数据库中表的操作主要就是增加、删除、修改以及查询4种常用的操作。使用command命令完成这些操作只使用两种方法就可以，一种是执行增加、删除、修改操作时要使用的ExecuteNonQuery方法；另一种是执行查询的操作的ExecuteReader方法。无论使用哪种方法，首先要执行的是创建连接Oracle数据库的命令对象。代码如下：

```
OracleCommand 命令对象的名称 = new OracleCommand(连接对象名称,SQL语句);
```

【语法说明】

- ☐ 连接对象名称：前面创建的OracleConnection对象的名称conn。
- ☐ SQL语句：执行对数据库表操作的增加、删除、修改、查询操作的语句。

例如，创建名为cmd的命令对象。代码如下：



```
OracleCommand cmd = new OracleCommand(conn, SQL语句);
```

在创建连接Oracle数据库的命令对象后,就可以执行对数据库的操作了。执行查询操作的代码如下:

```
创建连接Oracle数据库的命令对象名.ExecuteReader();
```

这里,创建连接Oracle数据库的命令对象名就是指前面创建的cmd。执行查询操作后,查询结果是OracleDataReader类型的。

例如,执行查询语句,代码如下:

```
OracleCommand cmd = new OracleCommand(conn, "select * from SALES");
OracleDataReader dr = cmd.ExecuteReader();
```

执行增加、删除、修改操作的语句也就是非查询语句,代码如下:

```
创建连接Oracle数据库的命令对象名.ExecuteNonQuery();
```

这里,在执行非查询的操作后,返回的结果是int类型,如果结果是-1,则代表操作失败。

例如,执行非查询语句,代码如下:

```
OracleCommand cmd = new OracleCommand(conn, "delete from SALES");
OracleDataReader dr = cmd.ExecuteReader();
```

4. 关闭数据库的连接

在对数据库表的操作执行完毕后,要关闭数据库的连接,否则就会浪费数据库的资源。关闭数据库的连接是非常简单的。代码如下:

```
数据库的连接对象.Close();
```

例如,关闭上面创建的数据库连接对象conn。代码如下:

```
conn.Close();
```

21.3.2 使用DataSet对象存储查询结果

使用DataSet对象连接数据库进行数据库表的查询,是一种断开式的查询方式,也是企业级开发推荐使用的。使用DataSet对象完成数据库表的查询分为以下6个步骤。

1. 创建数据库的连接

创建数据库连接的方法与前面使用Command对象操作数据库的方法一样。代码如下:

```
OracleConnection 对象名 = new OracleConnection("连接字符串");
```

2. 打开数据库连接

打开数据库连接仍然使用Open方法。代码如下:

```
创建的数据库连接名称.Open();
```

3. 创建数据适配器

数据适配器是用来把数据表中的数据填充到DataSet数据集中的工具。创建数据适配器的



代码如下：

```
OracleDataAdapter 数据适配器对象名 = new OracleDataAdapter("数据库连接对象名称", "查询语句");
```

【语法说明】

- ☐ 数据适配器对象名：任意的变量名称。
- ☐ 数据库连接对象名称：创建OracleConnection对象的名称。
- ☐ 查询语句：查询数据库的语句，也就是select语句。

4. 创建数据集对象

数据集就是DataSet对象，DataSet就是数据库中虚拟的表，把数据表中的数据填充到数据集后，就可以操作数据集中的数据了。创建数据集对象的代码如下：

```
DataSet 数据集对象名 = new DataSet();
```

这里，数据集对象名也可以是任意的名称。

5. 填充数据集

填充数据集时就要用到数据适配器的Fill方法。代码如下：

```
数据适配器对象.Fill(数据集的对象名);
```

这里，数据适配器对象就是指前面使用OracleDataAdapter创建的对象，数据集对象就是使用DataSet创建的对象。

6. 关闭数据库连接

关闭数据库连接的代码如下：

```
数据库的连接对象.Close();
```

说明 使用command对象不仅可以操作SQL语句，还可以操作存储过程，这时就要用到command对象的CommandType这个属性来设置是执行SQL语句还是执行存储过程。当CommandType的属性值是CommandType.StoredProcedure时就可以执行存储过程，当CommandType的属性值是CommandType.Text时就可以执行普通的SQL语句了。在.NET中默认执行的是SQL语句。

21.3.3 商品信息存储实例

商品信息主要是指商品的编号、商品的名称、商品的价格等信息，在本实例中主要完成对数据表商品信息的增加、删除、修改以及查询的操作。这里结合.NET中Windows应用程序完成商品信息存储的简单功能。下面就分别讲述在.NET中如何使用Oracle数据库完成增加、删除、修改以及查询的操作。

1. 查询功能的实现

在Windows应用程序中实现查询功能一般是使用前面所使用的DataGridview控件显示数据库的数据的，在这里主要实现一个显示数据库中全部数据的功能。当运行Windows窗体时就显示出数据库中全部的商品信息，可以在页面的加载事件（Load）中加入如下代码：



```
private void Form1_Load(object sender, EventArgs e)
{
    01 OracleConnection conn = new OracleConnection(@"Data Source=orcl;
    User ID=system; Password=abc123; Unicode=TRUE");
    02 conn.Open();
    03 OracleCommand cmd = new OracleCommand();
    04 cmd.Connection = conn;
    05 cmd.CommandText = "select * from scott.productinfo";
    06 OracleDataReader dr = cmd.ExecuteReader();
    07 BindingSource bs = new BindingSource();
    08 bs.DataSource = dr;
    09 dataGridView1.DataSource = bs;
}
```

【代码解析】

- 第1行创建一个数据库的连接，这里连接的数据源是orcl，用户名是system，密码是abc123。
- 第2行打开数据库连接。
- 第3行创建一个数据库连接命令对象。
- 第4行给命令对象设置连接属性，即connection属性。
- 第5行给命令对象赋给要执行的SQL语句，该SQL语句就是要查询出产品信息表中的全部信息。
- 第6行创建数据读取对象来存放数据读取记录。
- 第7行创建绑定数据源对象，因为要给DataGridView绑定数据源，就需要用到BindingSource的对象才可以。
- 第8行给BindingSource对象设置数据源，这个数据源是前面读取数据的数据读取对象dr。
- 第9行给页面上的DataGridView控件设置数据源，这个数据源是前面用到的BindingSource对象bs。

在页面的加载事件中加入上述代码后，运行效果如图21.18所示。

在DataGridView中显示的是商品信息表中的全部记录，如果想根据条件查询可以在SQL语句上加WHERE子句来完成，也可以实现根据产品名称查询商品的功能。只要在商品浏览的页面上加一个文本框和一个查询按钮，在文本框中输入查询条件，在【查询】按钮的单击事件中加入系列代码即可。

2. 增加商品信息

增加商品信息就是把商品信息添加到数据库中，在商品信息浏览界面（图21.18）中单击【添加】按钮，弹出如图21.19所示的商品信息添加界面。在文本框中



图21.18 商品管理浏览页面



图21.19 添加商品页面



输入商品的编号、商品的名称、商品的价格等信息，这里在数据库中商品的编号是主键，也就是说商品的编号是不能够重复的。一般在程序中是不用手动添加编号的，但是在这里为了能够让读者看到编号的添加就在页面上显示了。

输入好添加的信息后，单击【确定】按钮，即可把商品信息添加到数据库中。在【确定】按钮的单击事件中加入的代码如下：

```
01 OracleConnection conn = new OracleConnection(@"Data Source=orcl;  
User ID=system; Password=abc123;  
02 Unicode=TRUE");  
03 conn.Open();  
04 OracleCommand cmd = new OracleCommand();  
05 cmd.Connection = conn;  
06 cmd.CommandText = "insert into scott.productinfo(productid,productname,productprice)  
values('' +  
07 textBox1.Text + ',' + textBox2.Text + ',' + textBox3.Text + ')';  
08 cmd.CommandType = CommandType.Text;  
09 cmd.ExecuteNonQuery();  
10 conn.Close();
```

【代码解析】

- ☐ 第1~5行代码在商品浏览功能中已经用到过了。
- ☐ 第6~7行是给命令对象cmd的CommandText属性设置增加商品信息的SQL语句。
- ☐ 第8行设置命令对象中CommandText属性存放的类型，可以是文本、存储过程，如果不指定具体的类型，默认的类型是文本类型，也就是一般的SQL语句类型。
- ☐ 第9行执行的是增加商品信息SQL语句的操作。

3. 修改商品信息

修改商品信息是在图21.18所示的页面中，选择要修改的商品，然后单击【修改】按钮，弹出修改商品信息的页面，如图21.20所示。

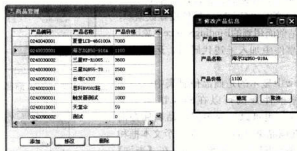


图21.20 修改商品信息

可以看出在图21.20中弹出的修改产品信息窗口中已经存在了产品原有的信息，那么这个功能是如何实现的呢？在进入修改商品信息窗口之前，通过在【修改】按钮的单击事件中加入下面的代码实现：

```
01 string productid= dataGridView1.SelectedRows[0].Cells[0].Value.ToString();
```

```

02 string productname = dataGridView1.SelectedRows[0].Cells[1].Value.ToString();
03 string productprice = dataGridView1.SelectedRows[0].Cells[2].Value.ToString();
04 UpdateProduct up = new UpdateProduct(productid, productname, productprice);
05 up.Show();

```



读书笔记

【代码解析】

- 第1~3行得到选中项的产品编码、产品名称、产品价格3个值。取得DataGridView单元格中值的方法是DataGridView的名字加上SelectedRows[0].Cells[0].Value.ToString(), SelectedRows[0]是指选中的行数, Cells[0]是指选中行的第几个单元格。这里0就代表第1个单元格产品编码,因为单元格的编号是从0开始的。
- 第4行创建修改商品信息对象,并给修改商品信息的窗体传递参数,即产品编码、产品名称、产品价格的信息。
- 第5行使用Show方法来显示修改商品的窗体。

上面从商品浏览界面中需要传递给修改商品信息的窗体的信息只有3个,可以使用上面的方法。如果传递的信息过多,那么就不适用上面的方法,一种方式是把要传递的值存储到一个数组中传递过去;另一种方式是只传递一个商品的编号,然后在商品修改界面的加载事件即Load事件中,根据传递过来的商品编号从数据库中查询出商品的相应信息。

技巧

在把原有的信息传递给修改商品信息的页面之后,就可以对商品的信息进行修改,修改完成后,单击【确定】按钮即可。在【确定】按钮中要加入下面的代码:

```

01 OracleConnection conn=new OracleConnection(@"Data Source=orcl;
    User ID=system; Password=abc123; Unicode=TRUE");
02 conn.Open();
03 OracleCommand cmd=new OracleCommand();
04 cmd.Connection=conn;
05 cmd.CommandText="update scott.productinfo set productname="+textBox2.Text+",
    productprice="+textBox3.Text+"where productid="+textBox1.Text+" ";
06 cmd.CommandType = CommandType.Text;
07 cmd.ExecuteNonQuery();
08 conn.Close();

```

【代码解析】

- 第1~4、6~8行与前面的增加商品信息的操作相同,这里就不再赘述了。
- 第5行在命令对象的CommandText中设置修改的语句。这里需要说明的是,修改商品信息来源是在界面上的文本框,获取文本框内容的方法是文本框的名称.Text方法。

在图21.20所示界面中,如果不想修改商品的信息或者修改完商品信息后取消操作,可以单击【取消】按钮,关闭修改商品的窗口。在【取消】按钮的单击事件中输入的代码如下:

```
this.Close();
```

这里, this就代表了当前的窗体,即修改商品信息的窗体。



除了上面给出的拼接SQL语句的方法之外，还可以使用string.Format方法来格式化字符串。上面的UPDATE语句可以更改为：

```
string sql = update scott.productinfo set productname=
" {0} ",productprice="{1}" where productid="{2} ";
sql=string.Format(sql, textBox2.Text, textBox3.Text, textBox1.Text);
```

技巧

通过上面的方法可以实现与原来的SQL语句同样的操作，并且使用这种方式不容易出现错误。这里使用{0}、{1}占位符。在格式化方法中占位符都是从0开始的，并且在给占位符赋值时，参数的个数必须要与占位符的个数相同，否则会出现错误。

4. 删除商品信息

删除商品信息的功能是通过选择一条商品信息，然后单击【删除】按钮来完成的。只需要在【删除】按钮的单击事件中加入下面的代码即可：

```
01 string productid = dataGridView1.SelectedRows[0].Cells[0].Value.ToString();
02 OracleConnection conn = new OracleConnection(@"Data Source=orcl; User ID=system;
    Password=abc123; Unicode=TRUE");
03 conn.Open();
04 OracleCommand cmd = new OracleCommand();
05 cmd.Connection = conn;
06 cmd.CommandText = "delete from scott.productinfo where productid="+productid+"";
07 cmd.CommandType = CommandType.Text;
08 cmd.ExecuteNonQuery();
```

【代码解析】

- 第1行用来取得要删除商品信息的商品编码，因为在商品信息表中商品编码是主键，获取的是选中行的第一个单元格的内容，并把单元格中的内容转换成字符串类型。
- 第6行为命令对象的CommandText属性设置删除语句，一定要注意不要把where条件语句丢掉，否则就代表删除商品信息表中的全部记录。where条件中的商品编码就是通过第1行代码获取的商品编号。

在加入上面的代码删除商品信息时，并没有看到商品信息从商品列表中消失，这主要是因为删除数据库中数据后没有更新页面的显示信息。如果想要在删除之后，在页面上显示已删除的记录，可以在删除数据之后，重新绑定DataGridView。重新绑定DataGridView也就是再执行本例中的Load事件中的内容。

通过上面的4个步骤就可以完成对商品信息的基本操作，相信读者已经了解了如何在.NET中连接数据库，并实现对数据库的基本操作。

21.4 小结

通过本章的学习，读者可以学习到.NET中连接数据库的组件ADO.NET的使用，掌握ADO.NET组件中的对象，并能够分别使用代码和绑定的方式完成简单的数据库操作。本章还引入了一个数据库开发的实例，在实例的开发中只使用SQL语句操作数据库，有兴趣的读者还可以在实例中加入存储过程的操作，如使用存储过程增加数据、修改数据等。

21.5 习题

简答题

1. 什么是ADO.NET?
2. 使用DataGridView控件如何绑定Oracle数据库?
3. 使用ADO.NET中的对象, 完成数据库的查询操作, 写出具体实现的代码。
4. 模仿商品信息存储的实例, 完成学生信息存储的练习, 学生信息包括学生的学号、学生的姓名、学生的年龄、学生的专业4个信息, 学生的学号是主键。



读书笔记

第22章 在Java中连接Oracle

作为当今最为流行的编程语言之一Java语言，它要与数据库沟通需要用到JDBC，本章会对JDBC做一个比较全面的介绍。本章包括以下知识点：

- ☐ JDBC与ODBC简介
- ☐ Thin方式连接Oracle
- ☐ JDBC-ODBC桥连接Oracle

通过学习本章，能够使用Java语言应用两种数据库的连接方式创建自己的连接Oracle数据库的程序。

22.1 JDBC与ODBC简介

JDBC与ODBC虽然从命名上只差了一个字母，但使用方法上却有很大差异，具体它们代表什么含义以及在何时使用，相信读者通过下面的内容会对其有一个全面的了解。

22.1.1 什么是JDBC

JDBC全称是Java DataBase Connectivity (Java数据库连接)，它是一套Java应用程序接口，用来执行到数据库的SQL语句，包含在Java类库中。

JDBC是程序开发人员的福音，它独立于关系数据库，可以为不同类型关系数据库提供统一访问。也就是说，开发者只需要写一个程序就可以访问不一样的数据库，而不用为此重新编写代码（如Oracle、MySQL、SQL Server或其他数据库）。它由Java语言编写，因为Java语言的“与平台无关性”，编写的程序可以运行到任何支持Java平台的环境下。同样地，我们不用为不同软硬件环境再编写不同的程序，真正做到“一次编写，随处运行”。例如，企业升级某个基于Java的项目的时候可能把Oracle从Windows环境转入Linux环境下，或者改用其他类型的数据库，而这时JDBC部分只需做微量修改而更多时候则不需要做任何修改。

JDBC简单易用，很容易上手，要想实现基本操作，几个小时之内就能得到你想要的（甚至更短时间）。总的来说，它能完成下面几个操作：

- ☐ 建立与数据库或数据源的连接；
- ☐ 发送要执行的SQL语句；
- ☐ 处理返回的结果。

以上3步将会在后面的示例中得到验证。

要想操作数据库，我们还需要有该数据库的JDBC Driver，这个驱动一般由独立厂商提供，它就像一个桥梁一样连通JDBC API和数据库。依照JDBC的规范，它支持4种类型的驱动程序。这4种驱动类型分别是：

1) JDBC-ODBC桥驱动。使用这种类型的驱动的前提条件是客户端计算机安装了ODBC驱动程序，通过JDBC调用ODBC，然后由ODBC连接数据库。但这种驱动执行效率不高，而且

也不适合基于网络的应用，因为不是所有的客户计算机都安装有ODBC。

2) 本地应用程序接口部分支持Java的驱动。此种类型和第1种类型驱动相似，它将JDBC调用转成某种特定数据库客户端的调用。也就是说，这种情况需要本地机器安装特定数据库的客户端。这种类型同样不适合网络应用。

3) 数据库中间件的纯Java驱动。此类型驱动将把JDBC调用转成一个中间件需要的协议，然后由中间件服务转成数据库网络协议，由中间件负责连接不同的数据库。这种类型实际上是由三层构成的。它在中间件服务器上可以根据实际需求做一些调配。例如，在中间件这层使用第2种类型的驱动。

4) 直接连到数据库的纯Java驱动程序。这种类型驱动完全由Java编写，使用方不需要安装客户端软件。它将JDBC调用直接转成数据库使用的网络协议，允许客户端直接访问数据库管理系统服务器。该类型驱动通常被称为第4种类型的驱动，后面的示例将对其做详细介绍。

这几种驱动类型当中，后两种是推荐使用类型，也是最常用的类型；而前两种方式对资源的依赖相对更多，通常作为临时方案或测试方案。其中第1种类型将会在ODBC中作介绍。Java应用程序、JDBC驱动以及数据库的关系如图22.1所示。该图在层次上描述了几者所处的位置，这种关系在访问数据库的示例中会体现出来。

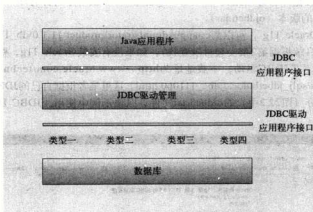


图22.1 层次关系

22.1.2 什么是ODBC

ODBC在JDBC之前就已经存在了，全称是Open Database Connectivity（开放式数据库连接），由C语言实现。它是由微软推出的统一的访问数据库（也可以是数据源）的接口技术，应用比较广泛。这种技术允许在Windows环境中的应用程序在不用修改的情况下可以访问不一样的数据库管理系统。它使得应用程序独立于数据库管理系统，所有对数据库的操作都由数据库管理系统对应的ODBC驱动程序来完成。这与JDBC类似。但这种访问方式效率比较低，通常在测试或过渡时期使用。

从官方文档了解到ODBC组件主要包括：

□ ODBC 管理器。相当于一个容器，把DBMS（数据库管理系统）配置成应用程序可用

的数据源。

- ☐ ODBC 驱动程序管理器。对使用者来说是透明的，加载ODBC 数据库驱动程序，包含在Odbc32.dll（动态链接库）中。
- ☐ ODBC API。函数库。
- ☐ ODBC 数据库驱动程序。主要是一些DLL，用来处理ODBC的调用。
- ☐ 数据源。

Java语言中访问ODBC数据源是通过JDBC-ODBC桥驱动来完成的，这样的操作方式限制比较多，目前用的时候不多，只需了解即可。

22.2 Thin方式连接Oracle

前面介绍过，想要用JDBC连接数据库就要有该数据库的JDBC Driver支持，通常它以jar文件的形式提供。这一点需要注意。

目前绝大部分的数据库都提供了JDBC Driver，如Oracle、DB2、MySQL等，但不同类型的数据库驱动是不一样的（甚至同种类型数据库不同版本之间的驱动名称也是不一样的，具体情况可以查看各数据库官方网站）。Oracle 11g自带JDBC驱动分为针对JDK 1.5（ojdbc5.jar）的版本和JDK 1.6的版本（ojdbc6.jar）。

如果安装了Oracle 11g，可以到c:\app\Administrator\product\11.1.0\db_1\jdbc\lib目录下查看自带驱动（该目录根据第2章安装目录查找），如果没有安装Oracle 11g，则可以到Oracle官方网站根据需求下载JDBC驱动。下载地址为http://www.oracle.com/technology/global/cn/software/tech/java/sqlj_jdbc/htdocs/jdbc_111060.html。这里笔者根据自己的JDK版本（JDK 1.5）选择下载黑框部分（图22.2中）的ojdbc5.jar文件，以便示例中使用。JDBC Driver下载页面如图22.2所示。



图22.2 JDBC Driver下载页面

企业中Java操作Oracle的开发最常用的就是第4类驱动，也叫JDBC Thin类型。它比较方便，客户投资低，效率高，开发者接受快。接下来看如何用该方式连接Oracle。



笔者将使用Eclipse作为开发工具，演示使用Java语言如何访问Oracle 11g，分为新建工程、导入数据库驱动包、创建配置文件、创建数据库连接初始化文件、创建数据库连接类以及编写测试类6个步骤。

(1) 新建工程

首先打开Eclipse开发工具，选择图22.3所示的菜单，新建一个Java工程。这里新建一个Java工程，用默认选项即可。这里工程取名为MyDbDemo。

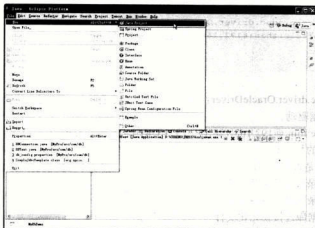


图22.3 新建Java工程

(2) 导入数据库驱动包

右击该项目名称，从弹出的快捷菜单中选择【Properties】选项，出现图22.4所示界面。

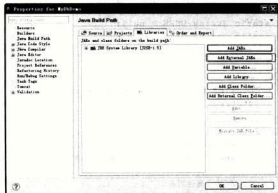


图22.4 导入Oracle驱动包

单击【Add External JARs】按钮，找到ojdbc5.jar文件，选中后单击【打开】按钮，回到图22.4。单击【OK】按钮，这样Oracle 11g的JDBC驱动就加载到了工程中。

**注意**

如果没有把 ojdbc5.jar 文件加入到该工程，会提示数据库驱动有问题，上面的这种方法只是临时方法，一般都是把该文件直接复制到工程下，然后加载即可。

(3) 创建配置文件

在工程中的src目录下创建一个名字为db_config.properties的文件，在该文件中存放Oracle的连接配置信息。这样当改动目标数据库时不用修改Java代码。具体内容如下：

```
#Oracle
drivers=oracle.jdbc.driver.OracleDriver
url=jdbc:oracle:thin:@192.168.128.130:1521:ORCL
user=scott
pwd=tiger
```

【代码解析】

- oracle.jdbc.driver.OracleDriver：Oracle连接驱动字符串。在代码中将获取该变量以加载Oracle数据库驱动。
- jdbc:oracle:thin:@192.168.128.130:1521:ORCL：Oracle数据库连接URL。其中具体描述了要连接的目标数据库。其中，“192.168.128.130:1521”是数据库所在机器IP以及端口，“ORCL”是数据库SID。
- scott：登录数据库的用户名。
- tiger：登录数据库的密码。

(4) 创建数据库连接初始化文件

在工程src目录下创建com.db包，再创建DBConfig.java文件。代码如下：

```
01 public class DBConfig {
02
03     private static String db_config = "db_config.properties";
04     public static String DRIVERS = null;
05     public static String URL = null;
06     public static String USER = null;
07     public static String PASSWORD = null;
08
09     static {
10
11         Properties props = new Properties();
12         InputStream inStr = null;
13
14         try {
15             //读取属性文件
16             inStr = ClassLoader.getSystemResourceAsStream(db_config);
17
18             //读取属性列表
19             props.load(inStr);
20         } catch (IOException e) {
21             // TODO Auto-generated catch block
22             e.printStackTrace();
23         }
```



```

24
25     DRIVERS = props.getProperty("drivers"); //得到Oracle连接驱动字符串
26     URL = props.getProperty("url"); //得到Oracle数据库连接URL
27     USER = props.getProperty("user"); //得到用户名
28     PASSWORD = props.getProperty("pwd"); //得到密码
29 }
30
31 }

```

【代码解析】

- ☐ 以上代码从db_config.properties文件中得到对应的数据，用于进行Oracle连接的初始化工作。为了使用方便，这里把变量声明成了静态变量。
- ☐ 第1行定义了一个类，名为DBConfig。
- ☐ 第3~7行表示声明类的属性，也就是变量，并为其进行初始化。
- ☐ 第9~29行是一个静态代码块。这里的功能主要是获取数据库连接参数，并赋值给第3~7行声明的变量。
- ☐ 第11行表示创建一个属性文件操作类的实例，名为props。
- ☐ 第12行表示声明一个输入流的对象。
- ☐ 第14行表示捕捉try后面的大括号内所有代码可能出现的异常。
- ☐ 第20~23行表示异常处理区。
- ☐ 第16行表示把属性文件转换成流文件。
- ☐ 第19行表示利用props对象，读取db_config.properties文件内的数据。
- ☐ 第25~28行表示为DBConfig类的属性赋值。

(5) 创建数据库连接类

数据库连接类的功能包括打开数据库连接和关闭连接。具体代码如下：

```

01 import java.sql.Connection;
02 import java.sql.DriverManager;
03 import java.sql.SQLException;
04
05 public class DBConnection {
06     private static String drivers = DBConfig.DRIVERS;
07     private static String url = DBConfig.URL;
08     private static String user = DBConfig.USER;
09     private static String password = DBConfig.PASSWORD;
10
11     /**
12      * 获取数据库连接，返回Connection对象
13      *
14      * @return
15      */
16     public static Connection GetConnection() {
17         Connection conn = null;
18         try {
19             // 这里使用Class.forName()方法创建驱动程序的实例并且自动调用
20             // DriverManager对其注册

```



```

20         Class.forName(drivers).newInstance();
21     } catch (InstantiationException e) {
22         e.printStackTrace();
23     } catch (IllegalAccessException e) {
24         e.printStackTrace();
25     } catch (ClassNotFoundException e) {
26         e.printStackTrace();
27     }
28
29     try {
30         // 通过DriverManager获取数据库连接
31         // 这里getConnection方法里面的3个参数均来自db_config.properties文件
32         System.out.println("url==" + url);
33         System.out.println("user==" + user);
34         System.out.println("password==" + password);
35         conn = DriverManager.getConnection(url, user, password);
36     } catch (SQLException e) {
37         e.printStackTrace();
38     }
39
40     return conn;
41 }
42
43 /**
44  * 关闭连接
45  *
46  * @param conn
47  */
48 public static void close(Connection conn) {
49     try {
50         if (conn != null && !conn.isClosed()) {
51             conn.close();
52         } catch (SQLException e) {
53             e.printStackTrace();
54         }
55     }
56 }
57 }

```

【代码解析】

- ☐ 该类的主要功能是对“数据库连接”进行控制，包括获取、打开数据库连接以及关闭数据库连接。
- ☐ 第1~3行表示引入其他工具类，它们分别是Connection类、DriverManager类、SQLException类。
- ☐ 第5行表示创建数据库连接控制类，名为DBConnection。
- ☐ 第6~9行表示创建类变量并为其初始化数据，这些变量所代表的含义和DBConfig类中同名的变量含义相同。
- ☐ 第11~15行是对方法getConnection的注释。



- 第16~41行属于方法GetConnection。该方法是一个静态方法。
- 第17行表示声明一个局部变量，变量名为conn，变量类型为Connection。
- 第20行表示注册驱动程序。这里使用了Java的反射机制，得到了“oracle.jdbc.driver.OracleDriver”的实例。
- 第32~34行表示输出数据结果到屏幕，可供参考。
- 第35行表示利用指定的数据库URL、用户名、密码得到数据库连接，以供外部调用。
- 第49行表示创建关闭连接的方法close。数据库连接一旦用完，要么回归连接池，要么关闭，否则会出现内存泄露等情况。

使用反射的方式实现并不是唯一方式，这样写的目的是可以重用该部分代码。真实的操作过程是这样的：

- 说明**
- 1) 创建oracle.jdbc.driver.OracleDriver的实例。
 - 2) 在实例化OracleDriver时Oracle提供的驱动中会利用DriverManager.registerDriver()对其自己创建的OracleDriver实例进行注册。
 - 3) 利用DriverManager的getConnection方法得到数据库连接。
- 以上步骤要从上到下顺序执行。

(6) 编写测试类

下面做一个测试类，查询Oracle 11g中scott用户下表DEPT的数据。代码如下：

```
01 import java.sql.Connection;
02 import java.sql.PreparedStatement;
03 import java.sql.ResultSet;
04 import java.sql.SQLException;
05 import java.sql.Statement;
06
07 public class DBTest {
08
09 /**
10 * 测试方法
11 */
12 public void TestDb() {
13     Connection conn = null;           // 声明Connection对象
14     PreparedStatement pstmt = null;    // 声明PreparedStatement对象，
                                         // 用来存储预编译语句
15     ResultSet rs = null;              // 声明ResultSet对象
16     try {
17         conn = DBConnection.GetConnection(); // 得到连接
18         pstmt = getStatement(conn, " SELECT DEPTNO , DNAME FROM DEPT ");
19         rs = executeQuery(pstmt);         // 执行查询
20         while (rs.next()) {
21             System.out.println(rs.getString("DEPTNO") + "=="
22                                 + rs.getString("DNAME"));
23         }
24     } catch (SQLException e) {
```



```

26         System.out.println(e.toString());
27         e.printStackTrace();
28     } finally {
29         close(rs);           // 关闭结果集
30         close(pstmt);        // 关闭语句
31         DBConnection.close(conn); // 关闭连接
32     }
33 }
34
35 /**
36  * 得到PreparedStatement对象
37  * @param conn
38  * @param strsql
39  * @return
40  */
41 public static PreparedStatement getState(Connection conn, String strsql) {
42     if (strsql == null || "".equals(strsql)) {
43         System.out.println("SQL为空...");
44         return null;
45     }
46     if (conn == null) {
47         System.out.println("连接为空...");
48         return null;
49     }
50     try {
51         return conn.prepareStatement(strsql, // 预编译语句得到PreparedStatement对象
52             ResultSet.TYPE_SCROLL_INSENSITIVE,
53             ResultSet.CONCUR_UPDATABLE);
54     } catch (SQLException e) {
55         // TODO Auto-generated catch block
56         e.printStackTrace();
57     }
58     return null;
59 }
60
61 /**
62  * 得到ResultSet
63  * @param pstmt
64  * @return
65  */
66 public static ResultSet executeQuery(PreparedStatement pstmt) {
67     try {
68         if (pstmt != null)
69             return pstmt.executeQuery(); // 查询
70     } catch (SQLException e) {
71         // TODO Auto-generated catch block

```



```

75 e.printStackTrace();
76 }
77 return null;
78 }
79
80 /**
81  * 关闭Statement对象
82  */
83 * @param stmt
84 */
85 public static void close(Statement stmt) {
86     try {
87         if (stmt != null) {
88             stmt.close();
89         }
90     } catch (SQLException e) {
91         e.printStackTrace();
92     }
93 }
94
95 /**
96  * 关闭结果集
97  */
98 * @param rs
99 */
100 public static void close(ResultSet rs) {
101     try {
102         if (rs != null) {
103             rs.close();
104         }
105     } catch (SQLException e) {
106         e.printStackTrace();
107     }
108 }
109 }

```

【代码解析】

- 代码中共有5个方法，根据业务可以拆分成两个类，但为了讲述方便这里不做拆分。
- 第1~5行表示引入的外部类。
- 第12行的TestDb是一个测试方法。执行过程如下：
 - 1) 调用连接方法，得到数据库连接。
 - 2) 得到PreparedStatement对象。
 - 3) 执行查询得到ResultSet对象。
 - 4) 利用ResultSet的next()方法，迭代输出结果集内容。
 - 5) 依序关闭ResultSet对象、PreparedStatement对象、Connection对象。
- 第42行的getStatement()方法是根据数据库连接和SQL语句得到PreparedStatement对象，其核心语句是连接调用PreparedStatement方法部分，其中的后面两个参数是为了生成允

- 许滚动和允许更新的 ResultSet 对象（默认情况下是不允许的）。调用该方法把SQL语句传给Oracle进行预编译，为查询做准备，在频繁执行该SQL语句时此种方式效率比较高。
- 第69行的executeQuery()方法的主要工作是执行查询语句，从Oracle中得到数据。通常称其为“结果集”，结果集中的数据可以利用迭代语句得到。
 - 第85行和第100行是两个关闭方法。这两个close方法利用了Java的方法重载原理，分别负责关闭Statement类型对象和结果集。

注意

当执行完查询结果时要按顺序关闭ResultSet、PreparedStatement、Connection 3个对象，特殊情况除外，否则会造成资源的浪费，直到连接不上Oracle。

执行结果如下：

```
10===ACCOUNTING
20===RESEARCH
30===SALES
40===OPERATIONS
```

22.3 JDBC-ODBC桥连接Oracle

使用JDBC-ODBC桥连接这种方式连接Oracle的前提是客户计算机上要有ODBC驱动程序，而且需要安装Oracle的客户端程序，也可以安装相当于客户端程序的其他程序，但这在Java中用得并不多。

22.3.1 配置ODBC数据源

使用JDBC-ODBC桥连接数据库的前提是要在计算机中安装ODBC数据源。具体的安装步骤如下。

(1) 找到【数据源 (ODBC)】选项

进入【控制面板】界面，从中找到【管理工具】，双击进入该页面，查看文件列表，找到【数据源 (ODBC)】，如图22.5所示。

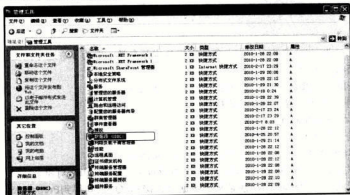


图22.5 数据源程序



说明 如果没有此程序，则需要安装，安装过程这里不讲述。

(2) 开始配置ODBC数据源

双击【数据源 (ODBC)】选项，出现ODBC数据源管理器界面，如图22.6所示。



图22.6 数据源管理器界面

从图22.6中可以看到很多标签。前3个都是有关DSN的，DSN表示数据源名称，应用程序要用它来和ODBC连接，里面存有数据库名称、数据库的用户名以及密码等。具体含义如下：

- ☐ 用户DSN：为某个用户创建的数据源，该用户可以使用。其信息存储在Windows注册表HKEY_CURRENT_USER\Software\Odbc\Odbc.ini\Odbc下。
- ☐ 系统DSN：整个系统的数据源，只要有访问权限就可以使用这里的DSN。其信息存储在Windows注册表HKEY_LOCAL_MACHINE\Software\Odbc\Odbc.ini\Odbc下。
- ☐ 文件DSN：不保存到注册表中，而是一个文件，可以让使用者连接数据提供者。默认位置“C:\Program Files\Common Files\ODBC\Data Sources”可以被任何本地安装了ODBC驱动程序的用户访问。
- ☐ 驱动程序：列出了本机所有已经存在的驱动程序。从该标签可以看到每个驱动程序的版本，所属公司、日期等。
- ☐ 跟踪：允许启用跟踪，使用者能够查看ODBC的调用日志，可以设置文件存储路径。
- ☐ 连接池：不用的连接返回到连接池中，等待下次调用，适合并发访问调用，节省重新连接服务器的时间。
- ☐ 关于：ODBC核心组件。

要想使用JDBC-ODBC桥连接数据库需要创建DSN，这里创建“系统DSN”。选择图22.6中的【系统DSN】标签，单击【添加】按钮，在列出的驱动程序中选择“Oracle in OraClient11g_home1”，如图22.7所示。进入Oracle ODBC驱动配置页面，在该页面输入必要的信息，详细见图22.8。

- ☐ Data Source Name：数据源名称，这里根据个人需求输入。
- ☐ Description：对数据源的一个描述。
- ☐ TNS Service Name：Transparent Network Substrate，负责本地和数据库服务器的远程连接，如果读者的Oracle服务器和ODBC数据源在一台计算机中，那么此处名称就是SID。
- ☐ UserID：用户名。

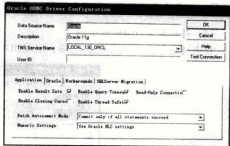


图22.8 驱动配置页面



图22.9 连接成功提示

完成了ODBC数据源安装后,如果要使用JDBC-ODBC桥连接Oracle数据库,只需要修改前面JDBC Thin程序,使之能用JDBC-ODBC桥连接。修改相当简单,由于代码的可复用性,只需要修改数据库的连接属性文件即可。修改db_config.properties属性文件如下:

```
#Oracle,jdbc-odbc
drivers=sun.jdbc.odbc.JdbcOdbcDriver
url=jdbc:odbc:Oracle
user=scott
pwd=tiger
```

注意 url对应的值jdbc:odbc:Oracle里的“Oracle”是配置数据源的名称。除了该属性文件外，其他程序代码部分不需要修改，由此可以看出JDBC的优势。

10---ACCOUNTING
20---RESEARCH
30---SALES
40---OPERATIONS

在本章里可以了解到什么是JDBC、什么是ODBC、在Java语言中如何使用JDBC从Oracle



11g中查询数据。而利用代码的复用性，只需要对底层的属性配置文件做修改，就可以实现利用ODBC桥连接Oracle。本章虽然介绍了两种连接Oracle的方法，但在企业Java的应用中，最常使用的就是Thin方式连接数据库。这种方式对客户端机器要求很低，资源占用比较少，平时可以多了解这方面的资料。

22.5 习题

一、填空题

1. JDBC主要可以完成的操作有_____、_____、_____。
2. 4种驱动类型是_____、_____、_____、_____。
3. ODBC组件主要包括_____、_____、_____、_____。

二、选择题

要想利用JDBC连接Oracle，需要使用()系统类。

- A. Connection B. DriverManager
C. TrustManager D. KeyManager

三、思考题

要求利用JDBC连接Oracle中的scott用户，并查询PRODUCTINFO表中价格大于2000的数据。



专业成就人生
立体服务大众

www.hzbook.com

填写读者调查表 加入华章书友会 获赠精彩技术书 参与活动和抽奖

尊敬的读者：

感谢您选择华章图书。为了聆听您的意见，以便我们能够为您提供更优秀的图书产品，敬请您抽出宝贵的时间填写本表，并按底部的地址邮寄给我们（您也可通过www.hzbook.com填写本表）。您将加入我们的“华章书友会”，及时获得新书资讯，免费参加书友会活动。我们将定期选出若干名热心读者，免费赠送我们出版的图书。请一定填写书名书号并留全您的联系信息，以便我们联络您，谢谢！

书名：

书号：7-111-()

姓名：	性别： <input type="checkbox"/> 男 <input type="checkbox"/> 女	年龄：	职业：
通信地址：		E-mail：	
电话：	手机：	邮编：	

1. 您是如何获知本书的：

☐ 朋友推荐 ☐ 书店 ☐ 图书目录 ☐ 杂志、报纸、网络等 ☐ 其他

2. 您从哪里购买本书：

☐ 新华书店 ☐ 计算机专业书店 ☐ 网上书店 ☐ 其他

3. 您对本书的评价是：

技术内容	<input type="checkbox"/> 很好	<input type="checkbox"/> 一般	<input type="checkbox"/> 较差	<input type="checkbox"/> 理由_____
文字质量	<input type="checkbox"/> 很好	<input type="checkbox"/> 一般	<input type="checkbox"/> 较差	<input type="checkbox"/> 理由_____
版式封面	<input type="checkbox"/> 很好	<input type="checkbox"/> 一般	<input type="checkbox"/> 较差	<input type="checkbox"/> 理由_____
印装质量	<input type="checkbox"/> 很好	<input type="checkbox"/> 一般	<input type="checkbox"/> 较差	<input type="checkbox"/> 理由_____
图书定价	<input type="checkbox"/> 太高	<input type="checkbox"/> 合适	<input type="checkbox"/> 较低	<input type="checkbox"/> 理由_____

4. 您希望我们的图书在哪些方面进行改进？

5. 您最希望我们出版哪方面的图书？如果有英文版请写出书名。

6. 您有没有写作或翻译技术图书的想法？

☐ 是，我的计划是_____ ☐ 否

7. 您希望获取图书信息的形式：

☐ 邮件 ☐ 信函 ☐ 短信 ☐ 其他_____

请寄：北京市西城区百万庄南街1号 机械工业出版社 华章公司 计算机图书策划部收
邮编：100037 电话：(010) 88379512 传真：(010) 68311602 E-mail: hzsj@hzbook.com